

GigaDevice Semiconductor Inc.

GD32F3x0

Arm[®] Cortex[®]-M4 32-bit MCU

**HAL 固件库
使用指南**

1.0 版本

（2023 年 9 月）

目录

目录.....	2
图索引.....	6
表索引.....	7
1. 介绍.....	22
1.1. 文档和 HAL 固件库规则.....	22
1.1.1. 外设缩写.....	22
1.1.2. 命名规则.....	23
2. HAL 固件库概述.....	24
2.1. HAL 固件库框架.....	24
2.2. HAL 固件库特点.....	25
2.3. HAL 固件库代码框架介绍.....	25
2.4. 文件组织结构.....	30
2.4.1. Examples 文件夹.....	30
2.4.2. Firmware 文件夹.....	31
2.4.3. Utilities 文件夹.....	31
2.5. 代码移植说明.....	31
2.5.1. 新建工程.....	31
2.5.2. 代码生成.....	33
2.5.3. 代码复制.....	34
2.5.4. 编译下载.....	36
2.5.5. 运行代码.....	38
2.6. HAL 固件库文件描述.....	38
2.7. HAL 固件库 FAQ.....	39
2.7.1. 在很多模块中, 函数的第一个形参均是设备信息结构体, 该结构体的作用是什么?	39
2.7.2. HAL 库提供的函数具体是如何使用的? 有没有例程可以参考?	40
2.7.3. 为什么使用上位机自动生成配置代码后, 外设无法工作?	40
2.7.4. HAL 库能否进行裁剪, 有哪些特性是可以修改的?	40
2.7.5. 前缀为 hal_ 的函数与前缀为 hals_ 的函数有何区别?	42
3. 外设 HAL 固件库.....	43
3.1. 外设 HAL 固件库概述.....	43
3.2. ADC.....	43
3.2.1. 外设寄存器说明.....	43
3.2.2. 外设库函数说明.....	44
3.3. CEC.....	73

3.3.1.	外设寄存器说明	74
3.3.2.	外设库函数说明	74
3.4.	CMP	83
3.4.1.	外设寄存器说明	83
3.4.2.	外设库函数说明	83
3.5.	CRC	93
3.5.1.	外设寄存器说明	93
3.5.2.	外设库函数说明	93
3.6.	CTC.....	98
3.6.1.	外设寄存器描述	98
3.6.2.	外设库函数说明	99
3.7.	DAC	107
3.7.1.	外设寄存器说明	107
3.7.2.	外设库函数说明	108
3.8.	SYS.....	118
3.8.1.	外设寄存器说明	118
3.8.2.	外设库函数说明	119
3.9.	DMA.....	127
3.9.1.	外设寄存器说明	127
3.9.2.	外设库函数说明	128
3.10.	EXTI.....	139
3.10.1.	外设寄存器说明	139
3.10.2.	外设库函数说明	139
3.11.	FMC.....	147
3.11.1.	外设寄存器说明	147
3.11.2.	外设库函数说明	148
3.12.	FWDGT.....	164
3.12.1.	外设寄存器说明	165
3.12.2.	外设库函数说明	165
3.13.	GPIO.....	168
3.13.1.	外设寄存器说明	168
3.13.2.	外设库函数说明	169
3.14.	I2C.....	173
3.14.1.	外设寄存器说明	174
3.14.2.	外设库函数说明	174
3.15.	SMBUS.....	203
3.15.1.	外设寄存器说明	203
3.15.2.	外设库函数说明	204
3.16.	NVIC	216

3.16.1.	外设寄存器说明	216
3.16.2.	外设库函数说明	216
3.17.	PMU.....	219
3.17.1.	外设寄存器说明	219
3.17.2.	外设库函数说明	219
3.18.	RCU.....	228
3.18.1.	外设寄存器说明	228
3.18.2.	外设库函数说明	229
3.19.	RTC.....	258
3.19.1.	外设寄存器描述	258
3.19.2.	外设库函数描述	259
3.20.	SPI.....	277
3.20.1.	外设寄存器说明	277
3.20.2.	外设库函数说明	277
3.21.	I2S.....	296
3.21.1.	外设寄存器说明	296
3.21.2.	外设库函数说明	297
3.22.	TIMER.....	311
3.22.1.	外设寄存器说明	311
3.22.2.	外设库函数说明	312
3.23.	TSI.....	384
3.23.1.	外设寄存器描述	384
3.23.2.	外设库函数说明	385
3.24.	UART.....	395
3.24.1.	外设寄存器说明	395
3.24.2.	外设库函数说明	396
3.25.	USRT.....	410
3.25.1.	外设寄存器说明	410
3.25.2.	外设库函数说明	411
3.26.	IrDA.....	427
3.26.1.	外设寄存器说明	427
3.26.2.	外设库函数说明	428
3.27.	SMARTCARD.....	442
3.27.1.	外设寄存器说明	442
3.27.2.	外设库函数说明	442
3.28.	WWDGT.....	456
3.28.1.	外设寄存器说明	456
3.28.2.	外设库函数说明	456
3.29.	USBF.....	465

3.29.1.	外设寄存器说明	465
3.29.2.	外设库函数说明	465
4.	版本历史	466

图索引

图 2-1. GD32F3x0 HAL 固件库实现框架	24
图 2-2. 回调函数实现	27
图 2-3. DMA 设备中断回调函数结构体	28
图 2-4. DMA 中 hal_dma_irq()函数	29
图 2-5. ADC_irq_handle_set 函数	29
图 2-6. GD32F3X0 HAL 固件库文件组织结构	30
图 2-7. 新建工程界面	32
图 2-8. 生成代码界面	33
图 2-9. 代码生成后界面	34
图 2-10. 工程对应地址查找	35
图 2-11. 复制以及重命名过程	36
图 2-12. 编译工程	37
图 2-13. 调试工程	37
图 2-14. 运行代码	38

表索引

表 1-1. 外设缩写	22
表 2-1. 固件函数库文件描述.....	39
表 3-1. 外设固件库函数描述格式.....	43
表 3-2. ADC 寄存器.....	43
表 3-3. ADC 库函数.....	44
表 3-4. 枚举 hal_adc_struct_type_enum	45
表 3-5. 枚举 hal_adc_state_enum.....	45
表 3-6. 枚举 hal_adc_error_enum.....	46
表 3-7. 枚举 hal_adc_oversample_shift_enum.....	46
表 3-8. 枚举 hal_adc_oversample_ratio_enum	47
表 3-9. 枚举 hal_adc_routine_sequence_enum	47
表 3-10. 枚举 hal_adc_inserted_sequence_enum	48
表 3-11. 结构体 hal_adc_irq_struct.....	48
表 3-12. 结构体 hal_adc_dma_handle_cb_struct	49
表 3-13. 结构体 hal_adc_dev_struct.....	49
表 3-14. 结构体 hal_adc_init_struct	49
表 3-15. 结构体 hal_adc_routine_rank_config_struct.....	49
表 3-16. 结构体 hal_adc_routine_config_struct	50
表 3-17. 结构体 hal_adc_inserted_rank_config_struct	50
表 3-18. 结构体 hal_adc_inserted_config_struct.....	50
表 3-19. 结构体 hal_adc_watchdog_config_struct	50
表 3-20. 函数 hal_adc_struct_init	51
表 3-21. 函数 hal_adc_deinit.....	52
表 3-22. 函数 hal_adc_init.....	52
表 3-23. 函数 hal_adc_calibration_start	53
表 3-24. 函数 hal_adc_routine_channel_config	54
表 3-25. 函数 hal_adc_routine_rank_config.....	55
表 3-26. 函数 hal_adc_start.....	55
表 3-27. 函数 hal_adc_stop.....	56
表 3-28. 函数 hal_adc_routine_conversion_poll	57
表 3-29. 函数 hal_adc_routine_software_trigger_enable.....	57
表 3-30. 函数 hal_adc_start_interrupt	58
表 3-31. 函数 hal_adc_stop_interrupt	59
表 3-32. 函数 hal_adc_start_dma.....	59
表 3-33. 函数 hal_adc_stop_dma.....	60
表 3-34. 函数 hal_adc_inserted_channel_config.....	61
表 3-35. 函数 hal_adc_inserted_rank_config.....	62
表 3-36. 函数 hal_adc_inserted_start.....	62
表 3-37. 函数 hal_adc_inserted_stop.....	63
表 3-38. 函数 hal_adc_inserted_conversion_poll.....	63

表 3-39. 函数 hal_adc_inserted_software_trigger_enable.....	64
表 3-40. 函数 hal_adc_inserted_start_interrupt.....	65
表 3-41. 函数 hal_adc_inserted_stop_interrupt.....	65
表 3-42. 函数 hal_adc_watchdog_config.....	66
表 3-43. 函数 hal_adc_watchdog_interrupt_enable.....	67
表 3-44. 函数 hal_adc_watchdog_interrupt_disable.....	68
表 3-45. 函数 hal_adc_watchdog_event_poll.....	68
表 3-46. 函数 hal_adc_irq.....	69
表 3-47. 函数 hal_adc_irq_handle_set.....	69
表 3-48. 函数 hal_adc_irq_handle_all_reset.....	70
表 3-49. 函数 hal_adc_routine_value_get.....	71
表 3-50. 函数 hal_adc_inserted_value_get.....	72
表 3-51. 函数 hal_adc_error_get.....	72
表 3-52. 函数 hal_adc_state_get.....	73
表 3-53. CEC 寄存器.....	74
表 3-54. CEC 库函数.....	74
表 3-55. 枚举 hal_cec_state_enum	74
表 3-56. 枚举 hal_cec_struct_type_enum.....	75
表 3-57. 枚举 hal_cec_error_enum	75
表 3-58. 结构体 hal_cec_irq_struct.....	75
表 3-59. 结构体 hal_cec_dev_struct.....	75
表 3-60. 结构体 hal_cec_init_struct.....	76
表 3-61. 函数 hal_cec_struct_init.....	76
表 3-62. 函数 hal_cec_init.....	77
表 3-63. 函数 hal_cec_deinit.....	78
表 3-64. 函数 hal_cec_start.....	78
表 3-65. 函数 hal_cec_start.....	79
表 3-66. 函数 hal_cec_transmit_interrupt.....	79
表 3-67. 函数 hal_cec_receive_interrupt.....	80
表 3-68. 函数 hal_cec_irq_handle_set.....	81
表 3-69. 函数 hal_cec_irq_handle_all_reset.....	82
表 3-70. 函数 hal_cec_irq.....	83
表 3-71. CMP 寄存器.....	83
表 3-72. CMP 库函数.....	84
表 3-73. 枚举 hal_cmp_state_enum	84
表 3-74. 枚举 hal_cmp_struct_type_enum.....	84
表 3-75. 枚举 hal_cmp_noninverting_input_enum	84
表 3-76. 枚举 hal_cmp_exti_type_enum.....	85
表 3-77. 结构体 hal_cmp_init_struct.....	85
表 3-78. 结构体 hal_cmp_dev_struct.....	85
表 3-79. 结构体 hal_cmp_irq_struct.....	85
表 3-80. 函数 hal_cmp_struct_init.....	85
表 3-81. 函数 hal_cmp_deinit.....	86
表 3-82. 函数 hal_cmp_init.....	87

表 3-83. 函数 hal_cmp_start.....	87
表 3-84. 函数 hal_cmp_stop.....	88
表 3-85. 函数 hal_cmp_start_interrupt.....	88
表 3-86. 函数 hal_cmp_stop_interrupt.....	89
表 3-87. 函数 hal_cmp_irq_handle_set.....	90
表 3-88. 函数 hal_cmp_irq_handle_all_reset.....	90
表 3-89. 函数 hal_cmp_irq.....	91
表 3-90. 函数 hal_cmp_lock.....	91
表 3-91. 函数 hal_cmp_output_level_get.....	92
表 3-92. 函数 hal_cmp_state_get.....	92
表 3-93. CRC 寄存器.....	93
表 3-94. CRC 库函数.....	94
表 3-95. 枚举 hal_crc_state_enum.....	94
表 3-96. 枚举 hal_crc_struct_type_enum.....	94
表 3-97. 结构体 hal_crc_dev_struct.....	94
表 3-98. 结构体 hal_crc_init_struct.....	94
表 3-99. 函数 hal_crc_struct_init.....	95
表 3-100. 函数 hal_crc_init.....	95
表 3-101. 函数 hal_crc_deinit.....	96
表 3-102. 函数 hal_crc_single_data_calculate.....	97
表 3-103. 函数 hal_crc_block_data_calculate.....	97
表 3-104. CTC 寄存器.....	99
表 3-105. CTC 库函数.....	99
表 3-106. 枚举 hal_ctc_struct_type_enum.....	99
表 3-107. 枚举 hal_ctc_error_enum.....	99
表 3-108. 枚举 hal_ctc_state_enum.....	100
表 3-109. 结构体 hal_ctc_irq_struct.....	100
表 3-110. 结构体 hal_ctc_init_struct.....	100
表 3-111. 结构体 hal_ctc_dev_struct.....	100
表 3-112. 函数 hal_ctc_deinit.....	101
表 3-113. 函数 hal_ctc_init.....	101
表 3-114. 函数 hal_ctc_irc48m_trim_value_config.....	102
表 3-115. 函数 hal_ctc_struct_init.....	103
表 3-116. 函数 hal_ctc_start.....	103
表 3-117. 函数 hal_ctc_stop.....	104
表 3-118. 函数 hal_ctc_irq.....	104
表 3-119. 函数 hal_ctc_irq_handle_set.....	105
表 3-120. 函数 hal_ctc_irq_handle_all_reset.....	105
表 3-121. 函数 hal_ctc_start_interrupt.....	106
表 3-122. 函数 hal_ctc_stop_interrupt.....	107
表 3-123. DAC 寄存器.....	107
表 3-124. DAC 库函数.....	108
表 3-125. 枚举 hal_dac_state_enum.....	108
表 3-126. 枚举 hal_dac_struct_type_enum.....	108

表 3-127. 枚举 hal_dac_error_enum	109
表 3-128. 结构体 hal_dac_irq_struct.....	109
表 3-129. 结构体 hal_dac_dma_handle_cb_struct.....	109
表 3-130. 结构体 hal_dac_dev_struct.....	109
表 3-131. 结构体 hal_dac_init_struct.....	109
表 3-132. 函数 hal_dac_init.....	110
表 3-133. 函数 hal_dac_deinit.....	111
表 3-134. 函数 hal_dac_struct_init.....	111
表 3-135. 函数 hal_dac_start.....	112
表 3-136. 函数 hal_dac_start.....	112
表 3-137. 函数 hal_dac_start_interrupt	113
表 3-138. 函数 hal_dac_stop_interrupt.....	114
表 3-139. 函数 hal_dac_start_dma.....	115
表 3-140. 函数 hal_dac_stop_dma.....	116
表 3-141. 函数 hal_dac_irq.....	116
表 3-142. 函数 hal_dac_irq_handle_set.....	117
表 3-143. 函数 hal_dac_irq_handle_all_reset.....	118
表 3-144. SYS DBG & SysTick 寄存器.....	118
表 3-145. SYS 库函数.....	119
表 3-146. 枚举 hal_sys_debug_cfg_enum	119
表 3-147. 枚举 hal_sys_timebase_source_enum	119
表 3-148. 函数 hal_sys_deinit.....	120
表 3-149. 函数 hal_sys_debug_init.....	120
表 3-150. 函数 hal_sys_timesource_init.....	121
表 3-151. 函数 hal_sys_basetick_count_get.....	122
表 3-152. 函数 hal_sys_basetick_timeout_check.....	122
表 3-153. 函数 hal_sys_basetick_delay_ms.....	123
表 3-154. 函数 hal_sys_basetick_suspend.....	124
表 3-155. 函数 hal_sys_basetick_resume.....	124
表 3-156. 函数 hal_sys_basetick_irq.....	125
表 3-157. 函数 hal_sys_basetick_irq_handle_set.....	126
表 3-158. 函数 hal_sys_basetick_irq_handle_all_reset.....	126
表 3-159. DMA 寄存器.....	127
表 3-160. DMA 库函数.....	128
表 3-161. 枚举 dma_channel_enum	128
表 3-162. 枚举 hal_dma_struct_type_enum.....	128
表 3-163. 枚举 hal_dma_transfer_state_enum.....	128
表 3-164. 枚举 hal_dma_error_enum	129
表 3-165. 枚举 hal_dma_state_enum	129
表 3-166. 结构体 hal_dma_irq_struct	129
表 3-167. 结构体 hal_dma_dev_struct.....	129
表 3-168. 结构体 hal_dma_init_struct.....	130
表 3-169. 函数 hal_dma_init.....	130
表 3-170. 函数 hal_dma_struct_init.....	131

表 3-171. 函数 hal_dma_deinit	131
表 3-172. 函数 hal_dma_start.....	132
表 3-173. 函数 hal_dma_stop.....	133
表 3-174. 函数 hal_dma_irq.....	133
表 3-175. 函数 hal_dma_irq_handle_set.....	134
表 3-176. 函数 hal_dma_irq_handle_all_reset.....	134
表 3-177. 函数 hal_dma_start_poll.....	135
表 3-178. 函数 hal_dma_start_interrupt.....	136
表 3-179. 函数 hal_dma_stop_interrupt.....	137
表 3-180. 函数 hal_dma_channel_remap_enable.....	137
表 3-181. 函数 hal_dma_channel_remap_enable.....	138
表 3-182. EXTI 寄存器.....	139
表 3-183. EXTI HAL 库函数.....	139
表 3-184. hal_exti_type_enum.....	140
表 3-185. hal_exti_line_enum.....	140
表 3-186. hal_exti_internal_line_enum.....	141
表 3-187. hal_exti_irq_index.....	142
表 3-188. 函数 hal_exti_gpio_deinit.....	142
表 3-189. 函数 hal_exti_internal_deinit.....	143
表 3-190. 函数 hal_exti_gpio_init.....	144
表 3-191. 函数 hal_exti_internal_init.....	145
表 3-192. 函数 hal_exti_gpio_irq_handle_set.....	145
表 3-193. 函数 hal_exti_gpio_irq_handle_all_reset	146
表 3-194. 函数 hal_exti_gpio_irq.....	147
表 3-195. FMC 寄存器.....	148
表 3-196. FMC 固件库函数.....	148
表 3-197. fmc_state_enum	149
表 3-198. 结构体 ob_pam_struct	149
表 3-199. 结构体 hal_sector_addr_range_struct.....	149
表 3-200. 结构体 hal_fmc_irq_struct.....	149
表 3-201. 结构体 hal_ob_pam_config_struct.....	150
表 3-202. 函数 hal_fmc_unlock.....	150
表 3-203. 函数 hal_fmc_lock.....	150
表 3-204. 函数 hal_fmc_unlock.....	151
表 3-205. 函数 hal_fmc_wait_state_enable.....	151
表 3-206. 函数 hal_fmc_wait_state_disable.....	152
表 3-207. 函数 hal_fmc_ready_wait.....	152
表 3-208. 函数 hal_fmc_wsnt_set.....	153
表 3-209. 函数 hal_fmc_region_read.....	153
表 3-210. 函数 hal_fmc_word_program.....	154
表 3-211. 函数 hal_fmc_halfword_program	154
表 3-212. 函数 hal_fmc_region_write.....	155
表 3-213. 函数 hal_fmc_page_erase	155
表 3-214. 函数 hal_fmc_mass_erase.....	156

表 3-215. 函数 hal_fmc_region_erase.....	156
表 3-216. 函数 hal_fmc_irq.....	157
表 3-217. 函数 hal_fmc_irq_handle_set.....	157
表 3-218. 函数 hal_fmc_irq_handle_all_reset.....	158
表 3-219. 函数 hal_ob_unlock.....	158
表 3-220. 函数 hal_ob_lock.....	159
表 3-221. 函数 hal_ob_reset.....	159
表 3-222. 函数 hal_ob_erase.....	160
表 3-223. 函数 hal_ob_security_protection_config.....	160
表 3-224. 函数 hal_ob_user_write.....	161
表 3-225. 函数 hal_ob_data_program.....	161
表 3-226. 函数 hal_ob_wp_enable.....	162
表 3-227. 函数 hal_ob_wp_disable.....	162
表 3-228. 函数 hal_ob_pam_get.....	163
表 3-229. 函数 hal_ob_write_protection_enable.....	163
表 3-230. 函数 hal_ob_pam_config.....	164
表 3-231. FWDGT 寄存器.....	165
表 3-232. FWDGT HAL 库函数.....	165
表 3-233. hal_fwdgt_state_enum.....	165
表 3-234. hal_fwdgt_struct_type_enum.....	166
表 3-235. hal_fwdgt_dev_struct.....	166
表 3-236. hal_fwdgt_init_struct.....	166
表 3-237. 函数 hal_fwdgt_struct_init.....	166
表 3-238. 函数 hal_fwdgt_init.....	167
表 3-239. 函数 hal_fwdgt_deinit.....	168
表 3-240. GPIO 寄存器.....	169
表 3-241. GPIO 库函数.....	169
表 3-242. 枚举 hal_gpio_mode_enum.....	169
表 3-243. 枚举 hal_gpio_pull_enum.....	169
表 3-244. 枚举 hal_gpio_ospeed_enum.....	170
表 3-245. 枚举 hal_gpio_af_enum.....	170
表 3-246. 结构体 hal_gpio_init_struct.....	170
表 3-247. 函数 hal_gpio_init.....	170
表 3-248. 函数 hal_gpio_bit_set.....	171
表 3-249. 函数 gpio_bit_reset.....	172
表 3-250. 函数 hal_gpio_struct_init.....	172
表 3-251. 函数 hal_gpio_deinit.....	173
表 3-252. I2C 寄存器.....	174
表 3-253. I2C 库函数.....	174
表 3-254. 枚举 i2c_flag_enum.....	175
表 3-255. 枚举 i2c_interrupt_flag_enum.....	176
表 3-256. 枚举 i2c_interrupt_enum.....	176
表 3-257. 枚举 hal_i2c_struct_type_enum.....	177
表 3-258. 枚举 hal_i2c_run_state_enum.....	177

表 3-259. 结构体 i2c_buffer_struct.....	177
表 3-260. 结构体 hal_i2c_irq_struct.....	178
表 3-261. 结构体 hal_i2c_slave_address_struct.....	178
表 3-262. 结构体 hal_i2c_dev_struct.....	178
表 3-263. 结构体 hal_i2c_init_struct.....	178
表 3-264. 函数 hal_i2c_struct_init.....	179
表 3-265. 函数 hal_i2c_deinit.....	179
表 3-266. 函数 hal_i2c_init.....	180
表 3-267. 函数 hal_i2c_error_irq.....	181
表 3-268. 函数 hal_i2c_event_irq.....	181
表 3-269. 函数 hal_i2c_start.....	182
表 3-270. 函数 hal_i2c_stop.....	182
表 3-271. 函数 hal_i2c_irq_handle_set.....	183
表 3-272. 函数 hal_i2c_irq_handle_all_reset.....	184
表 3-273. 函数 hal_i2c_master_transmit_poll.....	184
表 3-274. 函数 hal_i2c_master_receive_poll.....	185
表 3-275. 函数 hal_i2c_slave_transmit_poll.....	186
表 3-276. 函数 hal_i2c_slave_receive_poll.....	186
表 3-277. 函数 hal_i2c_memory_write_poll.....	187
表 3-278. 函数 hal_i2c_memory_read_poll.....	188
表 3-279. 函数 hal_i2c_master_transmit_interrupt.....	189
表 3-280. 函数 hal_i2c_master_receive_interrupt.....	190
表 3-281. 函数 hal_i2c_slave_transmit_interrupt.....	190
表 3-282. 函数 hal_i2c_slave_receive_interrupt.....	191
表 3-283. 函数 hal_i2c_memory_write_interrupt.....	192
表 3-284. 函数 hal_i2c_memory_read_interrupt.....	193
表 3-285. 函数 hal_i2c_master_transmit_dma.....	194
表 3-286. 函数 hal_i2c_master_receive_dma.....	194
表 3-287. 函数 hal_i2c_slave_transmit_dma.....	195
表 3-288. 函数 hal_i2c_slave_receive_dma.....	196
表 3-289. 函数 hal_i2c_memory_write_dma.....	197
表 3-290. 函数 hal_i2c_memory_read_dma.....	198
表 3-291. 函数 hal_i2c_device_ready_check.....	198
表 3-292. 函数 hal_i2c_master_serial_transmit_interrupt.....	199
表 3-293. 函数 hal_i2c_master_serial_receive_interrupt.....	200
表 3-294. 函数 hal_i2c_slave_serial_transmit_interrupt.....	201
表 3-295. 函数 hal_i2c_slave_serial_receive_interrupt.....	201
表 3-296. 函数 hal_i2c_address_listen_interrupt_enable.....	202
表 3-297. 函数 hal_i2c_address_listen_interrupt_disable.....	203
表 3-298. I2C 寄存器.....	204
表 3-299. SMBUS 库函数.....	204
表 3-300. 结构体 hal_smbus_irq_struct.....	205
表 3-301. 结构体 hal_smbus_init_struct.....	205
表 3-302. 结构体 hal_smbus_dev_struct.....	206

表 3-303. 函数 hal_smbus_struct_init.....	206
表 3-304. 函数 hal_smbus_deinit.....	207
表 3-305. 函数 hal_smbus_init.....	207
表 3-306. 函数 hal_smbus_slave_receive_interrupt.....	208
表 3-307. 函数 hal_smbus_event_irq.....	208
表 3-308. 函数 hal_smbus_error_irq.....	209
表 3-309. 函数 hal_smbus_start.....	210
表 3-310. 函数 hal_smbus_stop.....	210
表 3-311. 函数 hal_smbus_irq_handle_set.....	211
表 3-312. 函数 hal_smbus_irq_handle_all_reset.....	212
表 3-313. 函数 hal_smbus_master_transmit_interrupt.....	212
表 3-314. 函数 hal_smbus_master_receive_interrupt.....	213
表 3-315. 函数 hal_smbus_slave_transmit_interrupt.....	214
表 3-316. 函数 hal_smbus_slave_receive_interrupt.....	214
表 3-317. 函数 hal_smbus_disable_alert_interrupt.....	215
表 3-318. NVIC 寄存器.....	216
表 3-319. IRQn_Type.....	216
表 3-320. DAC 库函数.....	217
表 3-321. 函数 hal_nvic_irq_priority_group_set.....	217
表 3-322. 函数 hal_nvic_irq_enable.....	218
表 3-323. PMU 寄存器.....	219
表 3-324. PMU 库函数.....	219
表 3-325. 枚举 hal_pmu_lvd_voltage_enum.....	219
表 3-326. 枚举 hal_pmu_struct_type_enum.....	220
表 3-327. 枚举 hal_pmu_error_enum.....	220
表 3-328. 枚举 hal_pmu_state_enum.....	220
表 3-329. 结构体 hal_pmu_lvd_irq_struct.....	220
表 3-330. 结构体 hal_pmu_init_struct.....	220
表 3-331. 结构体 hal_pmu_dev_struct.....	221
表 3-332. 函数 hal_pmu_deinit.....	221
表 3-333. 函数 hal_pmu_struct_init.....	221
表 3-334. 函数 hal_pmu_lvd_init.....	222
表 3-335. 函数 hal_pmu_wakeup_pin_enable.....	223
表 3-336. 函数 hal_pmu_lvd_start.....	223
表 3-337. 函数 hal_pmu_lvd_stop.....	224
表 3-338. 函数 hal_pmu_wakeup_pin_disable.....	224
表 3-339. 函数 hal_pmu_lvd_irq.....	225
表 3-340. 函数 hal_pmu_lvd_irq_handle_set.....	226
表 3-341. 函数 hal_pmu_lvd_irq_handle_reset.....	226
表 3-342. 函数 hal_pmu_lvd_start_interrupt.....	227
表 3-343. 函数 hal_pmu_lvd_stop_interrupt.....	228
表 3-344. RCU 寄存器.....	228
表 3-345. RCU 库函数.....	229
表 3-346. 枚举 hal_rcu_periph_enum.....	230

表 3-347. 枚举 hal_rcu_periph_sleep_enum	231
表 3-348. 枚举 hal_rcu_periph_reset_enum.....	231
表 3-349. 枚举 hal_rcu_flag_enum.....	232
表 3-350. 枚举 hal_rcu_int_flag_enum	232
表 3-351. 枚举 hal_rcu_int_flag_clear_enum	233
表 3-352. 枚举 hal_rcu_int_enum	233
表 3-353. 枚举 hal_rcu_adc_clksrc_enum	234
表 3-354. 枚举 hal_rcu_clock_freq_enum.....	234
表 3-355. 枚举 hal_rcu_osci_type_enum.....	235
表 3-356. 枚举 hal_rcu_struct_type_enum	235
表 3-357. 枚举 hal_rcu_rtc_clksrc_enum.....	235
表 3-358. 枚举 hal_rcu_usart_clksrc_enum.....	235
表 3-359. 枚举 hal_rcu_usbfs_clksrc_enum	236
表 3-360. 枚举 hal_rcu_cec_clksrc_enum	236
表 3-361. 枚举 hal_rcu_sysclk_src_enum	236
表 3-362. 枚举 hal_rcu_ck48mclk_src_enum.....	237
表 3-363. 枚举 hal_rcu_sysclk_ahbdiv_enum.....	237
表 3-364. 枚举 hal_rcu_ahbclk_apb1div_enum.....	237
表 3-365. 枚举 hal_rcu_ahbclk_apb2div_enum.....	238
表 3-366. 枚举 hal_rcu_osc_state_enum.....	238
表 3-367. 枚举 hal_rcu_pll_src_enum	238
表 3-368. 枚举 hal_rcu_pll_prediv_enum.....	239
表 3-369. 枚举 hal_rcu_pll_mul_enum	239
表 3-370. 枚举 hal_rcu_pll_presel_enum.....	241
表 3-371. 枚举 hal_rcu_ckout_src_enum.....	241
表 3-372. 枚举 hal_rcu_ckout_div_enum.....	242
表 3-373. 结构体 hal_rcu_periphclk_struct.....	242
表 3-374. 结构体 hal_rcu_clk_struct	243
表 3-375. 结构体 hal_rcu_irq_struct.....	243
表 3-376. 结构体 hal_rcu_hxtal_struct.....	243
表 3-377. 结构体 hal_rcu_lxtal_struct.....	244
表 3-378. 结构体 hal_rcu_irc8m_struct	244
表 3-379. 结构体 hal_rcu_irc28m_struct.....	244
表 3-380. 结构体 hal_rcu_irc48m_struct.....	244
表 3-381. 结构体 hal_rcu_irc40k_struct.....	244
表 3-382. 结构体 hal_rcu_pll_struct.....	245
表 3-383. 结构体 hal_rcu_osci_struct.....	245
表 3-384. 函数 hal_rcu_osci_config.....	245
表 3-385. 函数 hal_rcu_clock_out_config.....	247
表 3-386. 函数 hal_rcu_deinit.....	247
表 3-387. 函数 hal_rcu_struct_init	248
表 3-388. 函数 hal_rcu_periph_clk_enable.....	248
表 3-389. 函数 hal_rcu_periph_clk_disable	249
表 3-390. 函数 hal_rcu_hxtal_clock_monitor_enable	249

表 3-391. 函数 hal_rcu_hxtal_clock_monitor_disable	250
表 3-392. 函数 hal_rcu_periph_reset_enable.....	250
表 3-393. 函数 hal_rcu_periph_reset_disable.....	251
表 3-394. 函数 hal_rcu_periph_clock_config.....	251
表 3-395. 函数 hal_rcu_periph_clkfreq_get	252
表 3-396. 函数 hal_rcu_osci_config_get.....	253
表 3-397. 函数 hal_rcu_clock_config	254
表 3-398. 函数 hal_rcu_clock_config_get.....	255
表 3-399. 函数 hal_SystemCoreClockUpdate.....	256
表 3-400. 函数 hal_rcu_irq	256
表 3-401. 函数 hal_rcu_irq_handle_set.....	257
表 3-402. 函数 hal_rcu_irq_handle_all_reset.....	257
表 3-403. RTC 寄存器	258
表 3-404. RTC 库函数	259
表 3-405. 枚举 hal_rtc_state_enum.....	259
表 3-406. 枚举 hal_rtc_error_enum.....	260
表 3-407. 枚举 hal_rtc_struct_type_enum	260
表 3-408. 结构体 hal_rtc_irq_struct.....	260
表 3-409. 结构体 hal_rtc_dev_struct	260
表 3-410. 结构体 hal_rtc_timestamp_struct.....	261
表 3-411. 结构体 hal_rtc_init_struct.....	261
表 3-412. 结构体 hal_rtc_alarm_output_config_struct.....	261
表 3-413. 结构体 hal_rtc_alarm_config_struct	261
表 3-414. 结构体 hal_rtc_tamper_config_struct.....	262
表 3-415. 函数 hal_rtc_init.....	262
表 3-416. 函数 hal_rtc_alarm_output_config	263
表 3-417. 函数 hal_rtc_alarm_config.....	264
表 3-418. 函数 hal_rtc_tamp_config.....	265
表 3-419. 函数 hal_rtc_timestamp_config	266
表 3-420. 函数 hal_rtc_calib_config.....	267
表 3-421. 函数 hal_rtc_refclock_detection_config.....	268
表 3-422. 函数 hal_rtc_struct_init.....	268
表 3-423. 函数 hal_rtc_deinit.....	269
表 3-424. 函数 hal_rtc_interrupt_enable.....	269
表 3-425. 函数 hal_rtc_interrupt_disable.....	270
表 3-426. 函数 hal_rtc_irq.....	271
表 3-427. 函数 hal_rtc_irq_handle_set.....	271
表 3-428. 函数 hal_rtc_irq_handle_all_reset.....	272
表 3-429. 函数 hal_rtc_current_time_get.....	272
表 3-430. 函数 hal_rtc_alarm_get	273
表 3-431. 函数 hal_rtc_alarm_event_poll.....	274
表 3-432. 函数 hal_rtc_timestamp_get.....	274
表 3-433. 函数 hal_rtc_timestamp_event_poll	275
表 3-434. 函数 hal_rtc_tamper0_event_poll.....	275

表 3-435. 函数 hal_rtc_tamper1_event_poll.....	276
表 3-436. 函数 hal_rtc_daylight_saving_time_adjust.....	276
表 3-437. SPI/I2S 寄存器.....	277
表 3-438. SPI 库函数.....	277
表 3-439. 枚举 hal_spi_struct_type_enum.....	278
表 3-440. 枚举 hal_spi_run_state_enum.....	278
表 3-441. 枚举 hal_spi_trans_mode_enum.....	279
表 3-442. 枚举 hal_spi_prescaler_enum.....	279
表 3-443. 结构体 hal_spi_irq_struct.....	279
表 3-444. 结构体 hal_spi_buffer_struct.....	279
表 3-445. 结构体 hal_spi_dev_struct.....	280
表 3-446. 结构体 hal_spi_user_callback_struct	280
表 3-447. 结构体 hal_spi_init_struct	280
表 3-448. 函数 hal_spi_deinit	281
表 3-449. 函数 hal_spi_struct_init.....	281
表 3-450. 函数 hal_spi_init.....	282
表 3-451. 函数 hal_spi_transmit_poll.....	283
表 3-452. 函数 hal_spi_receive_poll.....	284
表 3-453. 函数 hal_spi_transmit_receive_poll.....	285
表 3-454. 函数 hal_spi_transmit_interrupt.....	286
表 3-455. 函数 hal_spi_receive_interrupt.....	287
表 3-456. 函数 hal_spi_transmit_receive_interrupt.....	288
表 3-457. 函数 hal_spi_transmit_dma.....	289
表 3-458. 函数 hal_spi_receive_dma.....	290
表 3-459. 函数 hal_spi_transmit_receive_dma.....	291
表 3-460. 函数 hal_spi_irq.....	292
表 3-461. 函数 hal_spi_irq_handle_set.....	292
表 3-462. 函数 hal_spi_irq_handle_all_reset	293
表 3-463. 函数 hal_spi_start.....	293
表 3-464. 函数 hal_spi_stop.....	294
表 3-465. 函数 hal_spi_abort.....	294
表 3-466. 函数 hal_spi_dma_pause.....	295
表 3-467. 函数 hal_spi_dma_resume	295
表 3-468. 函数 hal_spi_dma_stop.....	296
表 3-469. SPI/I2S 寄存器	296
表 3-470. I2S 库函数	297
表 3-471. 枚举 hal_i2s_init_struct.....	297
表 3-472. 枚举 hal_i2s_run_state_enum.....	298
表 3-473. 枚举 hal_i2s_standard_enum.....	298
表 3-474. 枚举 hal_i2s_audiosample_enum.....	298
表 3-475. 结构体 hal_i2s_buffer_struct.....	298
表 3-476. 结构体 hal_i2s_irq_struct.....	299
表 3-477. 结构体 hal_i2s_dev_struct.....	299
表 3-478. 结构体 hal_i2s_user_callback_struct.....	299

表 3-479. 结构体 hal_i2s_init_struct.....	299
表 3-480. 函数 hal_i2s_deinit.....	300
表 3-481. 函数 hal_i2s_struct_init.....	300
表 3-482. 函数 hal_i2s_init.....	301
表 3-483. 函数 hal_i2s_transmit_poll.....	302
表 3-484. 函数 hal_i2s_receive_poll.....	302
表 3-485. 函数 hal_i2s_transmit_interrupt.....	303
表 3-486. 函数 hal_i2s_receive_interrupt.....	304
表 3-487. 函数 hal_i2s_transmit_dma.....	305
表 3-488. 函数 hal_i2s_receive_dma.....	306
表 3-489. 函数 hal_i2s_irq.....	307
表 3-490. 函数 hal_i2s_irq_handle_set.....	307
表 3-491. 函数 hal_i2s_irq_handle_all_reset.....	308
表 3-492. 函数 hal_i2s_start.....	308
表 3-493. 函数 hal_i2s_stop.....	309
表 3-494. 函数 hal_i2s_dma_pause.....	309
表 3-495. 函数 hal_i2s_dma_resume.....	310
表 3-496. 函数 hal_i2s_dma_stop.....	310
表 3-497. TIMER 寄存器.....	311
表 3-498. TIMER 库函数.....	312
表 3-499. TSI 寄存器.....	384
表 3-500. TSI 库函数.....	385
表 3-501. 枚举 hal_tsi_state_enum.....	385
表 3-502. 枚举 hal_tsi_error_enum.....	385
表 3-503. 枚举 hal_tsi_struct_type_enum.....	386
表 3-504. 结构体 hal_tsi_irq_struct.....	386
表 3-505. 结构体 hal_tsi_dev_struct.....	386
表 3-506. 结构体 hal_tsi_init_struct.....	386
表 3-507. 函数 hal_tsi_init.....	387
表 3-508. 函数 hal_tsi_struct_init.....	388
表 3-509. 函数 hal_tsi_deinit.....	388
表 3-510. 函数 hal_tsi_start.....	389
表 3-511. 函数 hal_tsi_stop.....	389
表 3-512. 函数 hal_tsi_start_interrupt.....	390
表 3-513. 函数 hal_tsi_stop_interrupt.....	391
表 3-514. 函数 hal_tsi_irq.....	391
表 3-515. 函数 hal_tsi_irq_handle_set.....	392
表 3-516. 函数 hal_tsi_irq_handle_all_reset.....	392
表 3-517. 函数 hal_tsi_group_cycle_get.....	393
表 3-518. 函数 hal_tsi_pins_config.....	394
表 3-519. 函数 hal_tsi_poll_transfer.....	394
表 3-520. USART 寄存器.....	395
表 3-521. UART 库函数.....	396
表 3-522. 枚举 hal_uart_struct_type_enum.....	396

表 3-523. 枚举 hal_uart_work_mode_enum.....	397
表 3-524. 结构体 hal_uart_init_struct.....	397
表 3-525. 结构体 hal_uart_irq_struct.....	398
表 3-526. 枚举 hal_uart_run_state_enum.....	398
表 3-527. 结构体 hal_uart_dev_struct.....	398
表 3-528. 结构体 hal_uart_user_callback_struct.....	399
表 3-529. 函数 hal_uart_struct_init.....	399
表 3-530. 函数 hal_uart_deinit.....	399
表 3-531. 函数 hal_uart_init.....	400
表 3-532. 函数 hal_uart_irq.....	401
表 3-533. 函数 hal_uart_irq_handle_set.....	401
表 3-534. 函数 hal_uart_irq_handle_all_reset.....	402
表 3-535. 函数 hal_uart_transmit_poll.....	403
表 3-536. 函数 hal_uart_receive_poll.....	403
表 3-537. 函数 hal_uart_transmit_interrupt.....	404
表 3-538. 函数 hal_uart_receive_interrupt.....	405
表 3-539. 函数 hal_uart_transmit_dma.....	406
表 3-540. 函数 hal_uart_receive_dma.....	407
表 3-541. 函数 hal_uart_dma_pause.....	408
表 3-542. 函数 hal_uart_dma_resume.....	409
表 3-543. 函数 hal_uart_transmit_stop.....	409
表 3-544. 函数 hal_uart_receive_stop.....	410
表 3-545. USART 寄存器.....	410
表 3-546. USRT 库函数.....	411
表 3-547. 枚举 hal_usrt_struct_type_enum.....	412
表 3-548. 结构体 hal_usrt_init_struct.....	412
表 3-549. 结构体 hal_usrt_irq_struct.....	412
表 3-550. 枚举 hal_usrt_run_state_enum.....	412
表 3-551. 结构体 hal_usrt_dev_struct.....	413
表 3-552. 结构体 hal_usrt_user_callback_struct.....	413
表 3-553. 函数 hal_usrt_struct_init.....	413
表 3-554. 函数 hal_usrt_deinit.....	414
表 3-555. 函数 hal_usrt_init.....	414
表 3-556. 函数 hal_usrt_irq.....	415
表 3-557. 函数 hal_usrt_irq_handle_set.....	416
表 3-558. 函数 hal_usrt_irq_handle_all_reset.....	417
表 3-559. 函数 hal_usrt_transmit_poll.....	417
表 3-560. 函数 hal_usrt_receive_poll.....	418
表 3-561. 函数 hal_usrt_transmit_receive_poll.....	419
表 3-562. 函数 hal_usrt_transmit_interrupt.....	419
表 3-563. 函数 hal_usrt_receive_interrupt.....	420
表 3-564. 函数 hal_usrt_transmit_receive_interrupt.....	421
表 3-565. 函数 hal_usrt_transmit_dma.....	422
表 3-566. 函数 hal_usrt_receive_dma.....	424

表 3-567. 函数 hal_usrt_transmit_receive_dma.....	425
表 3-568. 函数 hal_usrt_dma_pause	426
表 3-569. 函数 hal_usrt_dma_resume	426
表 3-570. 函数 hal_usrt_transfer_stop.....	427
表 3-571. USART 寄存器.....	428
表 3-572. IrDA 库函数	428
表 3-573. 枚举 hal_irda_struct_type_enum.....	429
表 3-574. 结构体 hal_irda_init_struct	429
表 3-575. 结构体 hal_irda_irq_struct.....	429
表 3-576. 枚举 hal_irda_run_state_enum	429
表 3-577. 结构体 hal_irda_dev_struct.....	430
表 3-578. 结构体 hal_irda_user_callback_struct	430
表 3-579. 函数 hal_irda_struct_init.....	430
表 3-580. 函数 hal_irda_deinit	431
表 3-581. 函数 hal_irda_init.....	431
表 3-582. 函数 hal_irda_irq.....	432
表 3-583. 函数 hal_irda_irq_handle_set.....	433
表 3-584. 函数 hal_irda_irq_handle_all_reset	433
表 3-585. 函数 hal_irda_transmit_poll	434
表 3-586. 函数 hal_irda_receive_poll.....	435
表 3-587. 函数 hal_irda_transmit_interrupt.....	435
表 3-588. 函数 hal_irda_receive_interrupt	436
表 3-589. 函数 hal_irda_transmit_dma.....	437
表 3-590. 函数 hal_irda_receive_dma.....	438
表 3-591. 函数 hal_irda_dma_pause.....	440
表 3-592. 函数 hal_irda_dma_resume	440
表 3-593. 函数 hal_irda_transmit_stop.....	441
表 3-594. 函数 hal_irda_receive_stop.....	441
表 3-595. USART 寄存器.....	442
表 3-596. SMARTCARD 库函数.....	442
表 3-597. 枚举 hal_smartcard_struct_type_enum	443
表 3-598. 结构体 hal_smartcard_init_struct.....	443
表 3-599. 结构体 hal_smartcard_irq_struct.....	444
表 3-600. 枚举 hal_smartcard_run_state_enum.....	444
表 3-601. 结构体 hal_smartcard_dev_struct	444
表 3-602. 结构体 hal_smartcard_user_callback_struct.....	444
表 3-603. 函数 hal_smartcard_struct_init	445
表 3-604. 函数 hal_smartcard_deinit.....	445
表 3-605. 函数 hal_smartcard_init.....	446
表 3-606. 函数 hal_smartcard_irq	447
表 3-607. 函数 hal_smartcard_irq_handle_set	448
表 3-608. 函数 hal_smartcard_irq_handle_all_reset.....	449
表 3-609. 函数 hal_smartcard_transmit_poll.....	449
表 3-610. 函数 hal_smartcard_receive_poll	450

表 3-611. 函数 hal_smartcard_transmit_interrupt.....	451
表 3-612. 函数 hal_smartcard_receive_interrupt.....	452
表 3-613. 函数 hal_smartcard_transmit_dma.....	453
表 3-614. 函数 hal_smartcard_receive_dma.....	454
表 3-615. 函数 hal_smartcard_transmit_stop.....	455
表 3-616. 函数 hal_smartcard_receive_stop.....	455
表 3-617. WWDGT 寄存器.....	456
表 3-618. WWDGT HAL 库函数.....	456
表 3-619. hal_wwdgt_state_enum.....	457
表 3-620. hal_wwdgt_struct_type_enum.....	457
表 3-621. hal_wwdgt_dev_struct.....	457
表 3-622. hal_wwdgt_init_struct.....	458
表 3-623. hal_wwdgt_irq_struct.....	458
表 3-624. 函数 hal_wwdgt_struct_init.....	458
表 3-625. 函数 hal_wwdgt_init.....	459
表 3-626. 函数 hal_wwdgt_deinit.....	460
表 3-627. 函数 hal_wwdgt_irq.....	460
表 3-628. 函数 hal_wwdgt_irq_handle_set.....	461
表 3-629. 函数 hal_wwdgt_irq_handle_all_reset.....	462
表 3-630. 函数 hal_wwdgt_start.....	462
表 3-631. 函数 hal_wwdgt_irq_handle_set.....	463
表 3-632. 函数 hal_wwdgt_reload.....	464
表 4-1. 版本历史.....	466

1. 介绍

本手册介绍了32位基于ARM微控制器GD32F3X0 HAL固件库。

该HAL固件库是一个固件函数包，它由程序、数据结构和宏组成，包括了GD32F3X0所有外设的性能特征。该HAL固件库还包括每一个外设的驱动描述和基于评估板的HAL固件库使用例程。通过使用本HAL固件库，用户无需深入掌握细节，也可以轻松应用每一个外设。使用本HAL固件库可以大大减少用户的编程时间，从而降低开发成本。

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

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

此份HAL固件库使用手册的整体架构如下：

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

1.1. 文档和 HAL 固件库规则

1.1.1. 外设缩写

表 1-1. 外设缩写

外设缩写	说明
ADC	模数转换器
CRC	循环冗余校验计算单元
CEC	HDMI-CEC控制器
CMP	比较器
CTC	时钟校准控制器
DAC	数模转换器
DBG	调试模块
DMA	直接存储器访问控制器
EXTI	外部中断事件控制器
FMC	闪存控制器
FWDGT	独立看门狗
GPIO	通用和备用输入/输出接口
I2C	内部集成电路总线接口

外设缩写	说明
NVIC	嵌套中断向量列表控制器
PMU	电源管理单元
RCU	复位和时钟单元
RTC	实时时钟
SPI/I2S	串行外设接口/片上音频接口
TIMER	定时器
TSI	触摸传感控制器
USART	通用同步异步收发器
WWDGT	窗口看门狗

1.1.2. 命名规则

HAL固件库遵从以下命名规则：

- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32f3x0_hal_”作为开头，例如：gd32f3x0_hal_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. HAL 固件库概述

2.1. HAL 固件库框架

HAL固件库.c文件的实现与外设模块的具体功能相对应,总体来说,HAL库实现框架如下所示:

图 2-1. GD32F3x0 HAL 固件库实现框架



HAL库实现的代码按照功能和特性,大体可以分为五大类,分别为:General、Internal、Analog、I/O device、Ticker。

General类主要为基础性外设功能,任何应用代码均会使用到这部分的实现。包括外设: RCU、GPIO、EXTI、DMA、NVIC。

Internal类主要为不涉及到外部PIN脚的外设功能。包括外设: CRC、FMC、WWDGT、FWDGT、PMU、SYSCFG、CTC。

Analog类主要为模拟相关的外设功能。包括外设ADC、CMP、DAC、TSI。

Ticker类主要为计数/定时器相关的外设功能。包括外设RTC、TIMER、SYSTICK。

I/O device类主要为通讯相关的外设功能。包括外设I2C、SPI、USART、CEC。

2.2. HAL 固件库特点

HAL 库代码资源可以与上位机配合使用, 以实现配置代码自动生成的功能。因此它具有统一化、抽象化等特点。

■ 统一的初始化函数

各外设功能的初始化函数均高度集中, 需配置的参数较多时, 统一使用结构体赋值的方式实现。但必须注意: 使用 HAL 库函数, 必须先调用 HAL 库中对应模块的初始化函数。

■ 统一的中断处理函数

对于具有中断机制的模块, 其代码均实现了一个中断处理函数, 命名为 `hal_xxx_irq()`。可以实现对本模块各种中断标志位的判断及对应处理。配合中断回调机制, 用户可以根据需要随意设置某种中断到来后的处理方式。

■ 中断回调机制

对于具有中断机制的模块, HAL 库统一使用了中断回调机制用于管理中断。

通常, 用户在使用中断时, 需要在中断入口添加具体的执行代码, 并处理中断标志位的判断及清除操作。

然而, 使用 HAL 库时, 用户仅需将库中对应的中断处理函数放置在中断入口即可, 该函数会正确处理对应中断标志位的判断并在退出中断前, 完成标志位清除的具体操作。用户仅需在使用到某种具体中断时, 将需要执行的代码以回调函数指针的形式, 通过具体函数的形参传入即可。HAL 库中的中断处理函数会在该中断到来时调用该指针, 并执行用户期望的具体操作。

■ 统一的延时管理机制

HAL 库提供了统一的延时管理机制, `hal_basetick.c` 为其实现。用户可以使用其中的延时函数实现精准延时。由于 HAL 库所有提供 `timeout` 机制的函数均是使用该延时管理机制实现, 因此使用 HAL 库函数前, 必须先调用 `hal_basetick_init()` 进行延时管理的初始化。

■ 统一的抽象化输入/输出函数

针对 I/O device 类, HAL 库实现了统一的抽象化输入/输出函数。在用户层面统一了不同通信类外设函数调用的方式, 便于用户快速的理解和使用。每一种通信类外设(包括 USART/SPI/I2C 等)均提供了轮询、中断、DMA 三种模式下的发送/接收函数。

■ 统一的结构体初始化函数

HAL 库中, 各模块提供了统一的结构体初始化函数 `hal_xxx_struct_init`。用户在使用到某个模块的初始化函数 `hal_xxx_init` 前, 对于不关心的初始化结构体成员, 仅需调用该接口, 即可完成其初值的赋值工作。

2.3. HAL 固件库代码框架介绍

HAL 库代码开发的初衷是为了配合 IDE 上位机, 实现“代码自动化生成”功能, 同时面向用户

提供更多方便的、集成度高的功能类函数。因此需要保证 IDE 上位机软件调用到的各外设模块函数具有统一性和规则性，同类型的不同功能外设模块需要具有相似的初始化、功能性函数实现。

整套 HAL 库代码包括三大类函数，分别为：1.初始化类函数。2.配置类函数。3.功能类函数。

■ 初始化类函数

为 IDE 上位机软件“代码自动化生成”功能会调用到的函数，一般为实现外设模块功能的初始化。IDE 上位机会自动生成成套对应的 `init` 和 `deinit` 函数。因此其具有以下特点：1.函数名称以 `_init()` 结尾。2.配套实现了对应的 `_deinit()` 函数。3.参数尽量使用结构体指针的形式进行传参。

■ 配置类函数

为 IDE 上位机软件“代码自动化生成”功能会调用到的函数，一般为实现外设模块某项具体功能的配置修改。

■ 功能类函数

需用户自己调用的函数，将外设的具体功能抽象集成，形成的集成度高的功能类函数。对于涉及中断和 DMA 的外设模块，提供以下函数实现，不涉及中断或 DMA 的外设模块实现在此基础上进行了相应缩减：

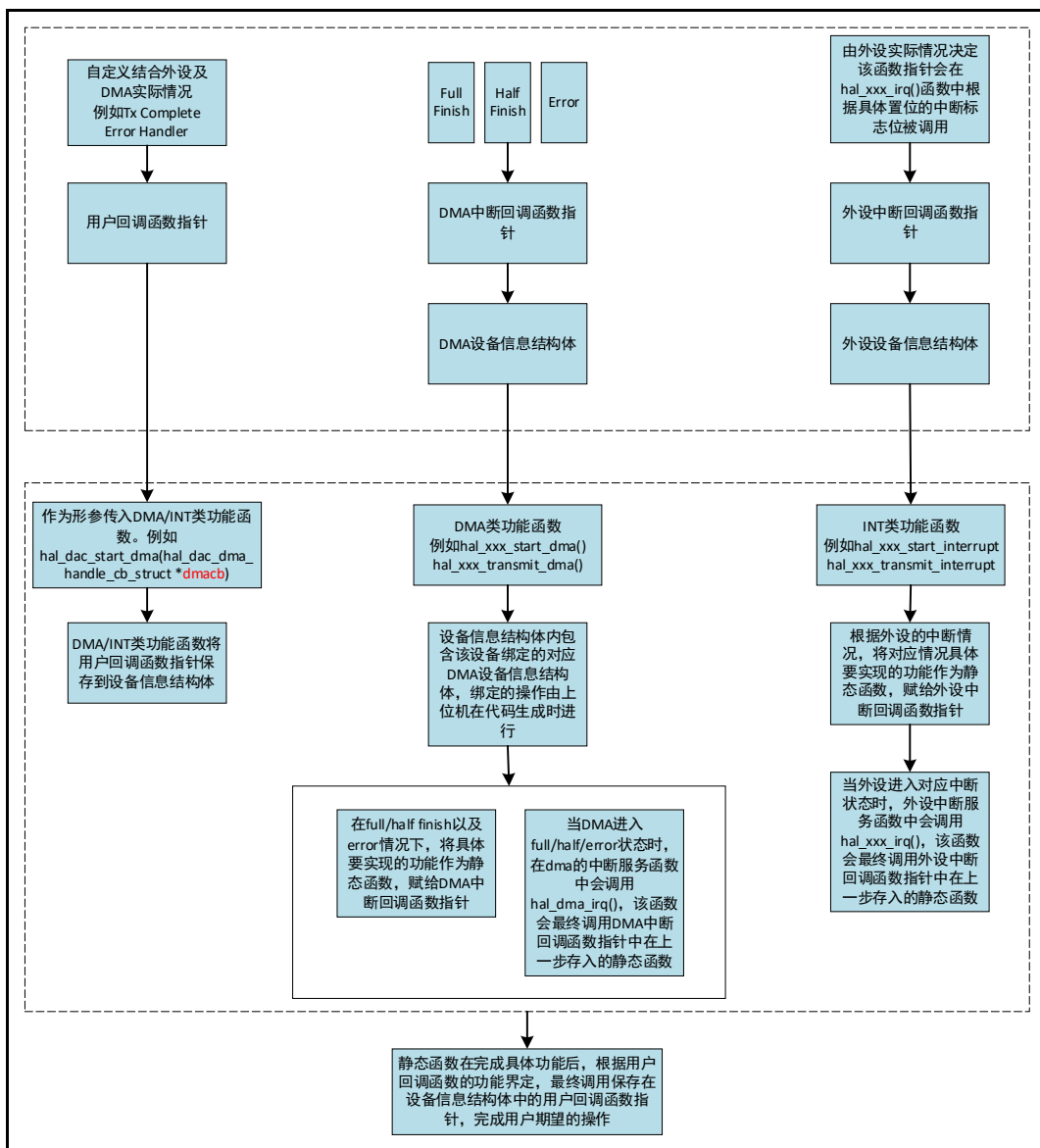
- 1.实现中断服务函数中使用的统一中断处理函数。对外设模块所有中断标志位进行判断，并根据实际情况跳转不同的回调函数进行中断处理。
- 2.实现用户中断回调函数的注册（`hal_xxx_irq_handle_set`）和解绑（`hal_xxx_irq_handle_all_reset`）。
- 3.实现 POLL / DMA / INT 三种方式的发送与接收，或者开始与终止。同时保证这类函数具备可重入性。

功能类函数实现特点

DMA/INT 类的功能函数的实现与中断进行了结合，通过中断标志位指引外设的状态并完成复杂的功能逻辑。功能函数的形参包含了回调函数指针（若提供给用户多个状态的回调函数接口，则会定义包含所有回调函数接口的结构体，并使用结构体指针进行形参传递），作为提供给用户的接口，指引用户在各种功能状态下进行希望的操作。

回调函数作为功能类函数实现中较核心的实现方式，大体设计和调用原则如下图所示：

图 2-2. 回调函数实现



为了实现上述功能，HAL 库中将会共性的定义几种特殊结构体，用于方便上述特点的实现：

【特殊结构体】

除“general 类”部分模块外，其余模块维护着几类特殊结构体，以 USART 为例，具体如下：

“设备信息结构体”，统一的命名方式为 `hal_xxx_dev_struct`，例如 USART 模块的 `hal_uart_dev_struct`

该结构体用于保存一些重要信息，方便各函数调用，因此设备信息结构体是结合具体外设的具体函数实现进行定义的，它用于保存外设基地址、中断回调函数（如果有中断的话），关联 DMA 设备信息结构体（如果有 DMA 功能的话）等。

由于“设备信息结构体”为全局变量，并且作为外设大部分函数的首个形参进行传递，因此“功能类函数”在实现的过程中，会利用“设备信息结构体”来进行参数的传递和保存。

原则上“设备信息结构体”的维护全部由 HAL 库函数完成，用户不需要直接对该结构体进行

赋值，只需要创建并将其作为参数传入函数即可。HAL 库函数会将传入的形参赋值给结构体的对应成员。

“设备中断回调函数结构体”，统一的命名方式为 `hal_xxx_irq_struct`，例如 DMA 模块的 `hal_dma_irq_struct`

图 2-3. DMA 设备中断回调函数结构体

```
typedef struct {  
    __IO hal_irq_handle_cb error_handle; .....  
    __IO hal_irq_handle_cb half_finish_handle; .....  
    __IO hal_irq_handle_cb full_finish_handle; .....  
} hal_dma_irq_struct;
```

该结构体作为“设备信息结构体”的一个成员，用于存储用户期望的中断来临时，用户期望调用的回调函数的指针。

“设备中断回调函数结构体”会在各外设模块代码中的唯一中断服务函数中用到，以 DMA 为例为 `hal_dma_irq()`。

图 2-4. DMA 中 hal_dma_irq()函数

```

int32_t hal_dma_irq(hal_dma_dev_struct *dma_dev)
{
    #if (1 == HAL_PARAMETER_CHECK)
        /* check the DMA pointer address and the number length parameter */
        if (NULL == dma_dev) {
            HAL_DEBUGE("parameter [dma_dev] value is invalid");
            return HAL_ERR_ADDRESS;
        }
    #endif /* 1 == HAL_PARAMETER_CHECK */
    /* full transfer finish interrupt handler */
    if (SET == hals_dma_interrupt_flag_get(dma_dev->channel, DMA_INTF_FTFIF)) {
        hals_dma_interrupt_flag_clear(dma_dev->channel, DMA_INTF_FTFIF);
        if (dma_dev->dma_irq.full_finish_handle != NULL) {
            dma_dev->dma_irq.full_finish_handle(dma_dev);
            /* unlock DMA */
            dma_dev->state = HAL_DMA_STATE_READY;
            HAL_UNLOCK(dma_dev);
        }
    }

    /* half transfer finish interrupt handler */
    if (SET == hals_dma_interrupt_flag_get(dma_dev->channel, DMA_INTF_HTFIF)) {
        /* if DMA not in circular mode */
        if (0 == (DMA_CHCTL(dma_dev->channel) & DMA_CHXCTL_CMEN)) {
            hals_dma_interrupt_disable(dma_dev->channel, DMA_INT_HTF);
        }
        hals_dma_interrupt_flag_clear(dma_dev->channel, DMA_INTF_HTFIF);
        if (dma_dev->dma_irq.half_finish_handle != NULL) {
            dma_dev->dma_irq.half_finish_handle(dma_dev);
        }
    }

    /* error interrupt handler */
    if (SET == hals_dma_interrupt_flag_get(dma_dev->channel, DMA_INTF_ERRIF)) {
        hals_dma_interrupt_flag_clear(dma_dev->channel, DMA_INTF_GIF);
        dma_dev->state = HAL_DMA_STATE_READY;
        dma_dev->error_state = HAL_DMA_ERROR_TRANSFER;
        /* unlock DMA */
        HAL_UNLOCK(dma_dev);
        if (dma_dev->dma_irq.error_handle != NULL) {
            dma_dev->dma_irq.error_handle(dma_dev);
        }
    }

    return HAL_ERR_NONE;
}

```

hal_xxx_irq()函数实现了外设模块所有中断标志位的判断及清中断操作，这样用户就不需要关心各类中断应该怎样处理，仅需要将对应中断来临时希望被调用到的函数以回调函数指针的形式传入“设备中断回调函数结构体”中即可。当中断来临时，该函数会判断回调函数指针是否有赋值，并进行回调。

特别的，如用户不需要使用 POLL/DMA/INT 函数，而是想自己指定特定的回调函数，可以通过调用各模块的 xxx_irq_handle_set 函数，来对“设备信息结构体”中的这个成员进行赋值，如 ADC 模块为

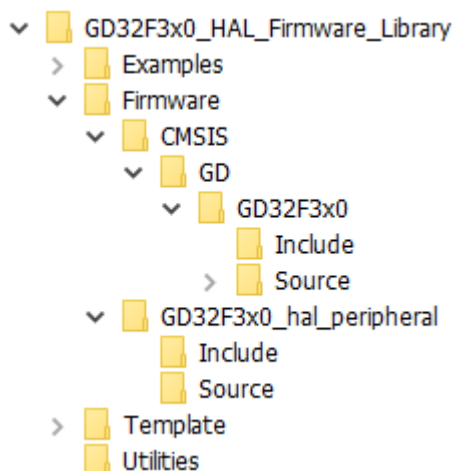
图 2-5. ADC_irq_handle_set 函数

```
void hal_adc_irq_handle_set(hal_adc_dev_struct *adc_dev, hal_adc_irq_struct *p_irq)
{
    /* EOC interrupt handler set */
    if(NULL != p_irq->adc_eoc_handle) {
        adc_dev->adc_irq.adc_eoc_handle = p_irq->adc_eoc_handle;
        hals_adc_interrupt_enable(ADC_INT_EOC);
    } else {
        adc_dev->adc_irq.adc_eoc_handle = NULL;
        hals_adc_interrupt_enable(ADC_INT_EOC);
    }
    /* EOIC interrupt handler set */
    if(NULL != p_irq->adc_eoic_handle) {
        adc_dev->adc_irq.adc_eoic_handle = p_irq->adc_eoic_handle;
        hals_adc_interrupt_enable(ADC_INT_EOIC);
    } else {
        adc_dev->adc_irq.adc_eoic_handle = NULL;
        hals_adc_interrupt_disable(ADC_INT_EOIC);
    }
    /* watchdog interrupt handler set */
    if(NULL != p_irq->adc_watchdog_handle) {
        adc_dev->adc_irq.adc_watchdog_handle = p_irq->adc_watchdog_handle;
        hals_adc_interrupt_enable(ADC_INT_WDE);
    } else {
        adc_dev->adc_irq.adc_watchdog_handle = NULL;
        hals_adc_interrupt_disable(ADC_INT_WDE);
    }
}
```

2.4. 文件组织结构

GD32F3X0_HAL_Firmware_Library，文件组织结构见下图：

图 2-6. GD32F3X0 HAL 固件库文件组织结构



2.4.1. Examples 文件夹

文件夹Examples，对应每一个GD32外设均包含一个子文件夹。每个子文件夹包含了关于本外设的一个或多个例程，来示范如何使用对应外设。每个例程子文件夹包含inc和src两个文件夹一个readme文件和一个gdc文件，其中文件夹inc包含的内容如下：

readme.txt：关于本例程的简单描述和使用说明；

xxx.gdc：关于Embedded Builder软件对应的配置信息；

inc文件夹包含的内容:

- **gd32f3x0_libopt.h**: 该头文件由不同的“#include”语句组成(默认情况下,不需要修改,所有外设均打开),该头文件可以设置例程所使用到的外设及HAL库的特殊功能(内部flash写和擦的工作模式,调试信息是否打印,发生错误是否阻塞等);
- **gd32f3x0_it.h**: 该头文件包含了所有的中断处理程序的原形;
- **gd32f3x0_hal_init.h**: 该头文件包含了所有使用到的外设的初始化函数和复位函数的原形。

src文件夹包含的内容:

- **gd32f3x0_it.c**: 该源文件包含了所有的中断处理程序(如果未使用到中断,则所有的函数体都为空);
- **gd32f3x0_hal_init.c**: 该源文件包含了所有使用到的外设的初始化函数和复位函数;
- **main.c**: 例程代码注:所有的例程的使用,都不受不同软件开发环境的影响。

2.4.2. Firmware 文件夹

Firmware文件夹包含组成HAL固件库核心的所有子文件夹和文件:

- **CMSIS**子文件夹包含有基于Cortex M4内核处理器的启动代码和库引导文件以及基于GD32F3x0的全局头文件和系统配置文件;
- **GD32F3x0_hal_peripheral**子文件夹;
- **Include**子文件夹包含了HAL固件函数库所需的头文件,用户无需修改该文件夹;
- **Source**子文件夹包含了HAL固件函数库所需的源文件,用户无需修改该文件夹;

2.4.3. Utilities 文件夹

Utilities文件夹包含运行HAL固件库例程评估板的文件:

- **gd32f3x0r_eval.h**文件是运行HAL固件库例程所需关于评估板的头文件;
- **gd32f3x0r_eval.c**文件是运行HAL固件库例程所需关于评估板的源文件。

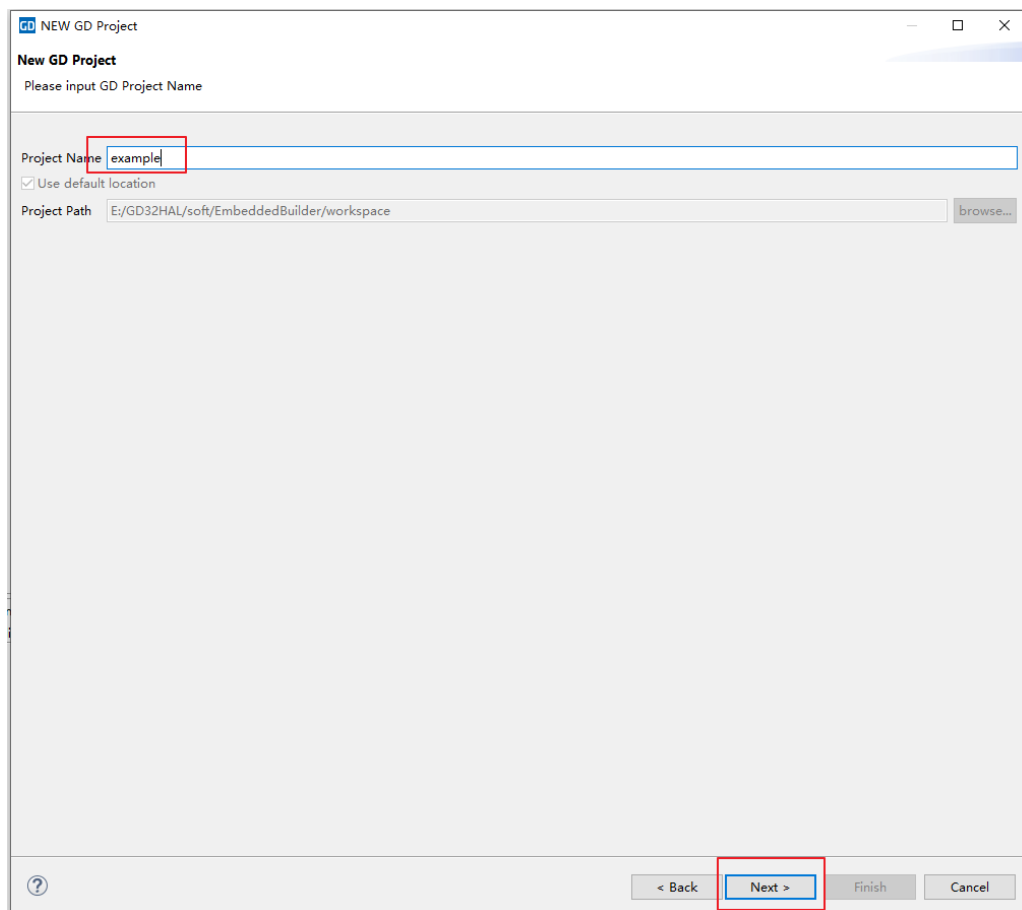
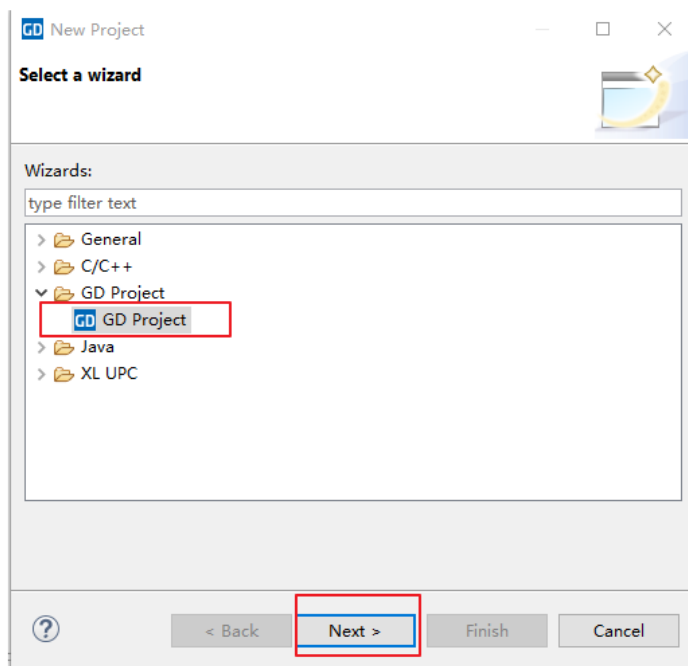
2.5. 代码移植说明

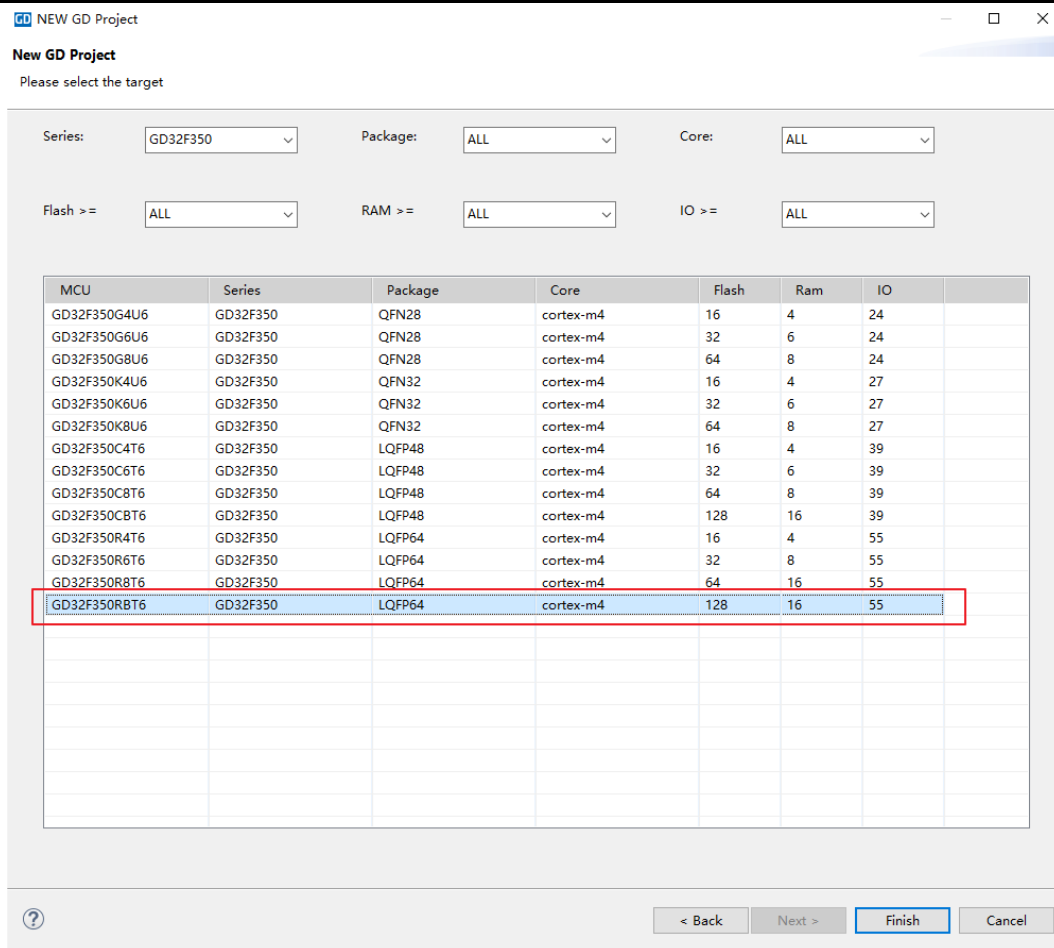
代码移植的目的是引导用户移植 **Examples** 文件夹下的 **example** 到 **EmbeddedBuilder** 软件中,并能正确使用。

2.5.1. 新建工程

打开 **Embedded Builder** 软件,新建工程,如[图 2-7. 新建工程界面](#)所示。

图 2-7. 新建工程界面

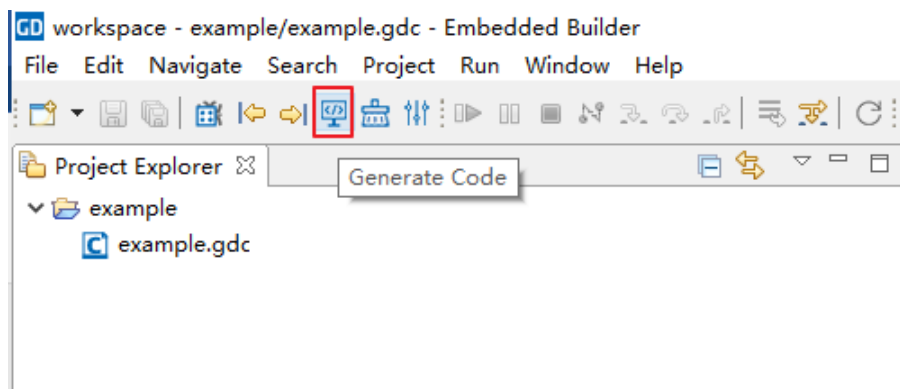


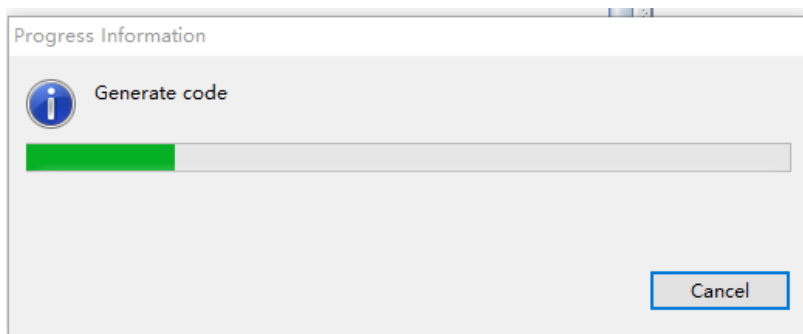


2.5.2. 代码生成

完成新建工程后，界面如[图 2-8. 生成代码界面](#)所示，此时，点击 **Generate Code** 界面。

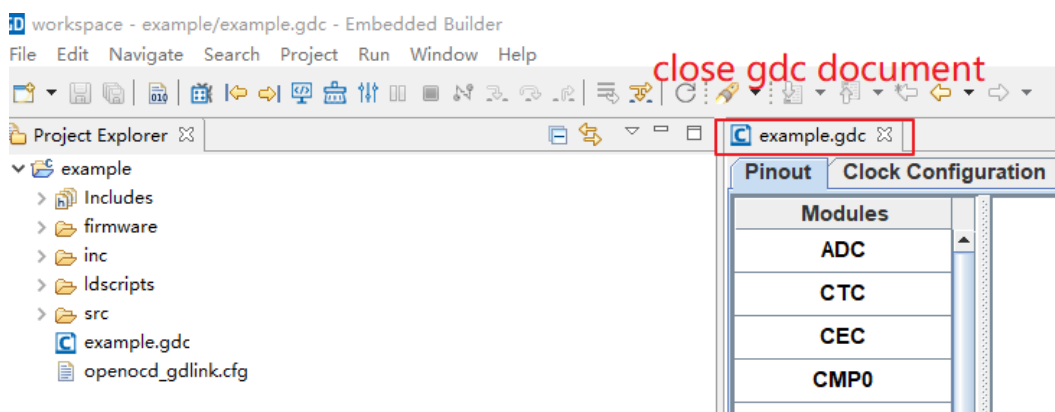
图 2-8. 生成代码界面





完成代码生成后的界面如[图 2-9. 代码生成后界面](#)所示，此时关闭对应 gdc 文件（特别注意，否则会造成替换工程不能正常使用）。

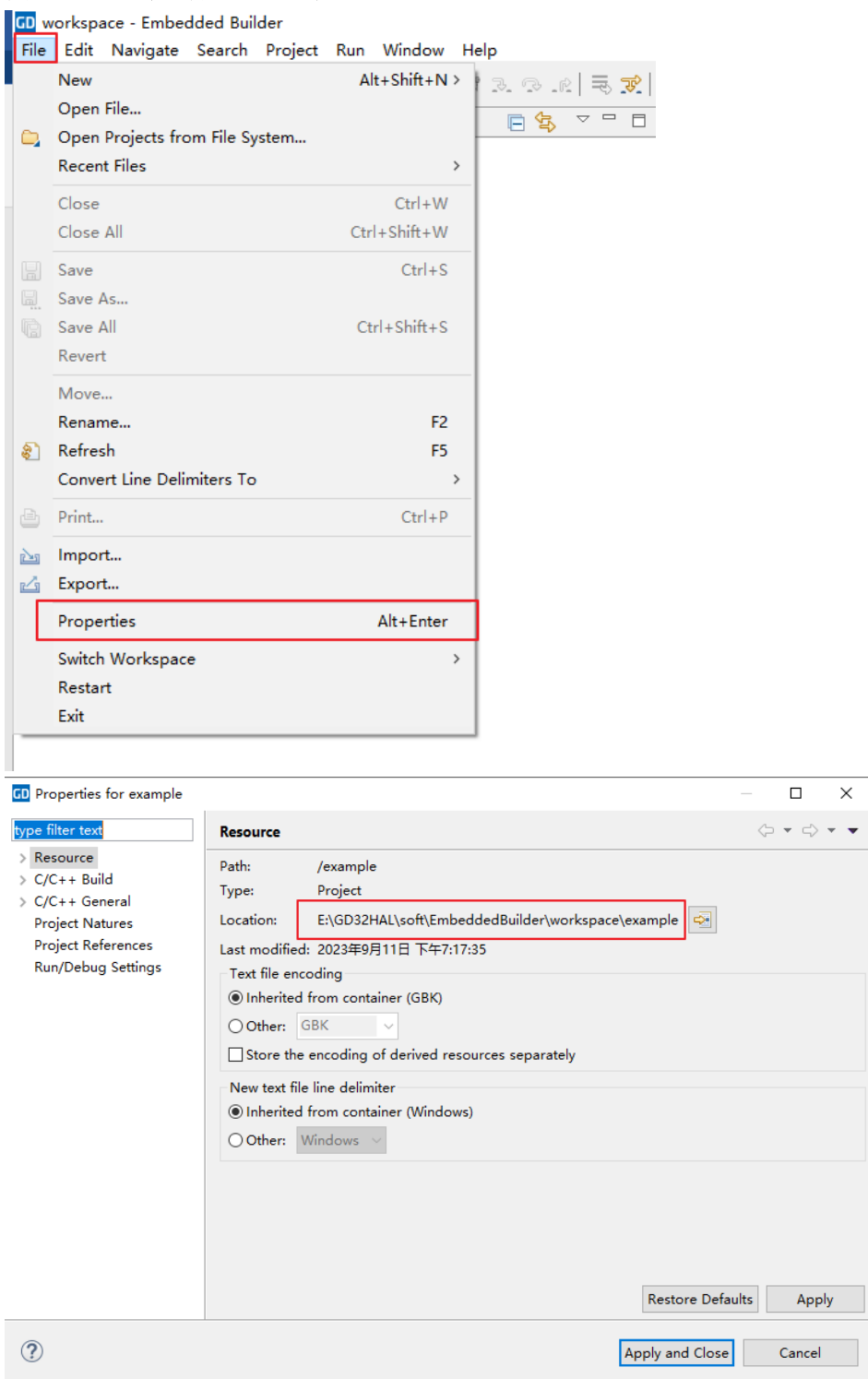
图 2-9. 代码生成后界面



2.5.3. 代码复制

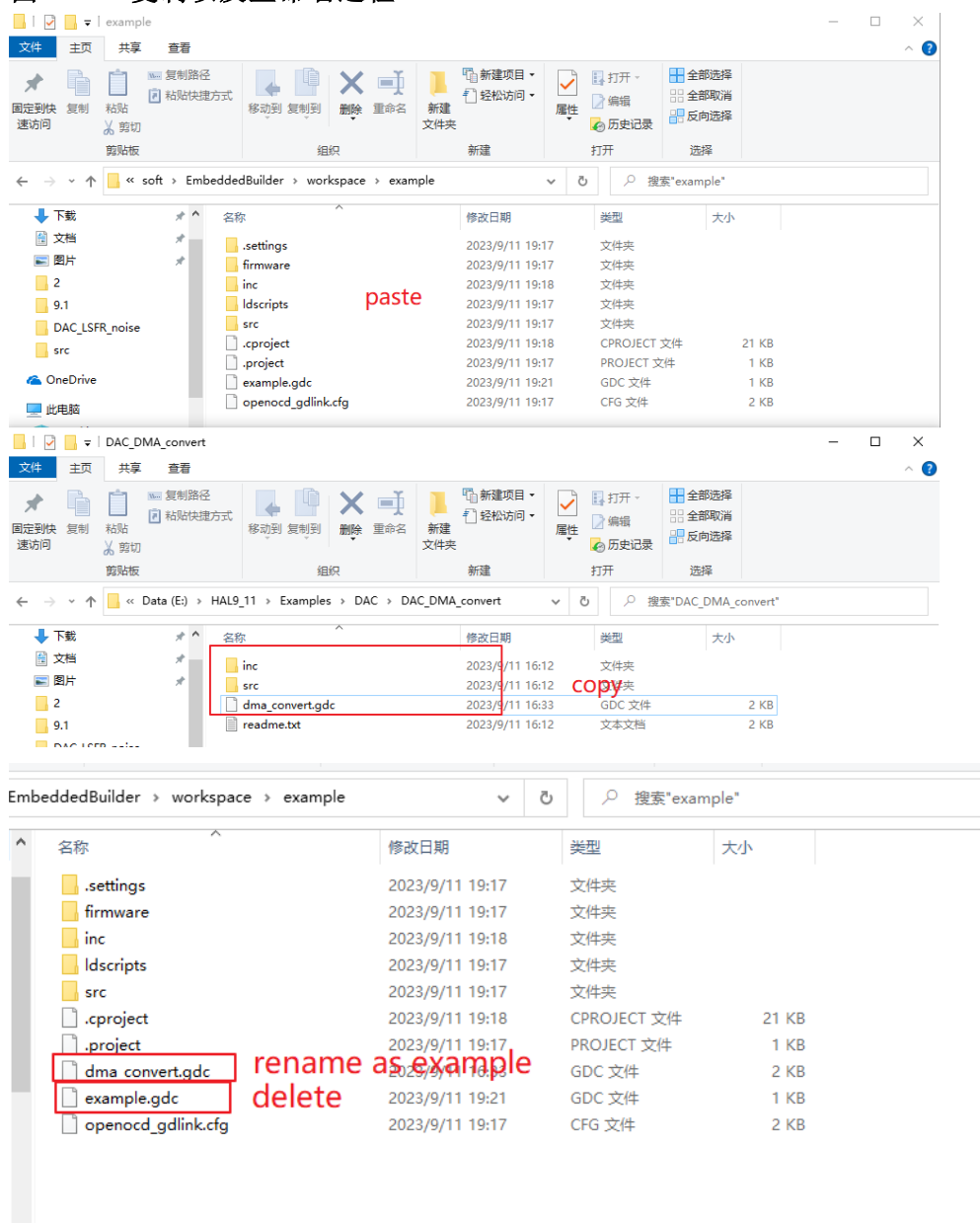
打开代码所在的文件夹地址，可从对应地址找到文件对应地址，如[图 2-10. 工程对应地址查找](#)所示。

图 2-10. 工程对应地址查找



此时将 Examples 文件夹下对应的文件复制到工程对应的文件下，并完成对应文件的删除以及重命名，如 [图 2-11. 复制以及重命名过程](#) 所示。

图 2-11. 复制以及重命名过程

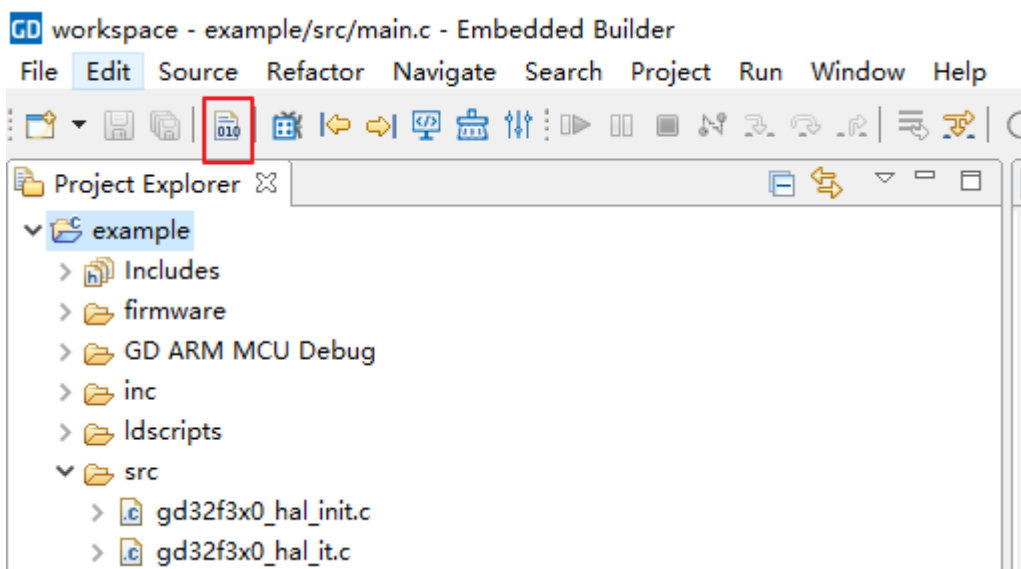


部分代码会使用到 Utilities 文件夹下的文件，此时将 gd32f3x0r_hal_eval.c 和 gd32f3x0r_hal_eval.h 文件复制带对应的 inc 和 src 文件目录下。

2.5.4. 编译下载

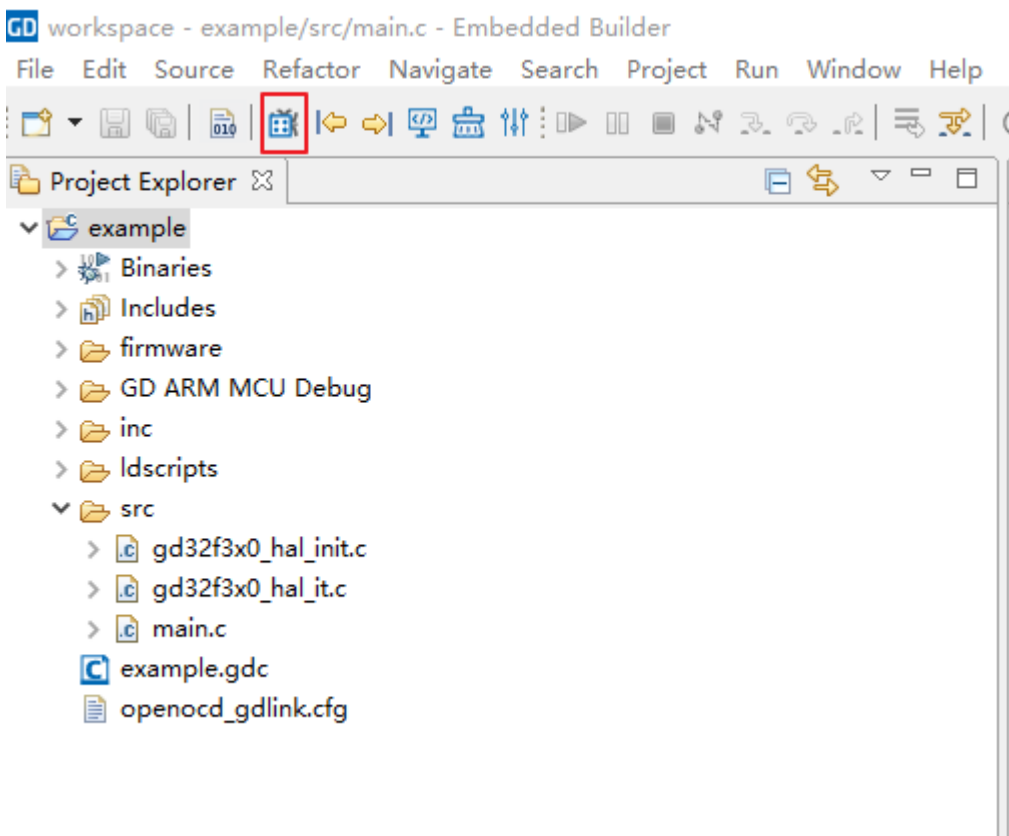
点击 Build all 完成代码编译。

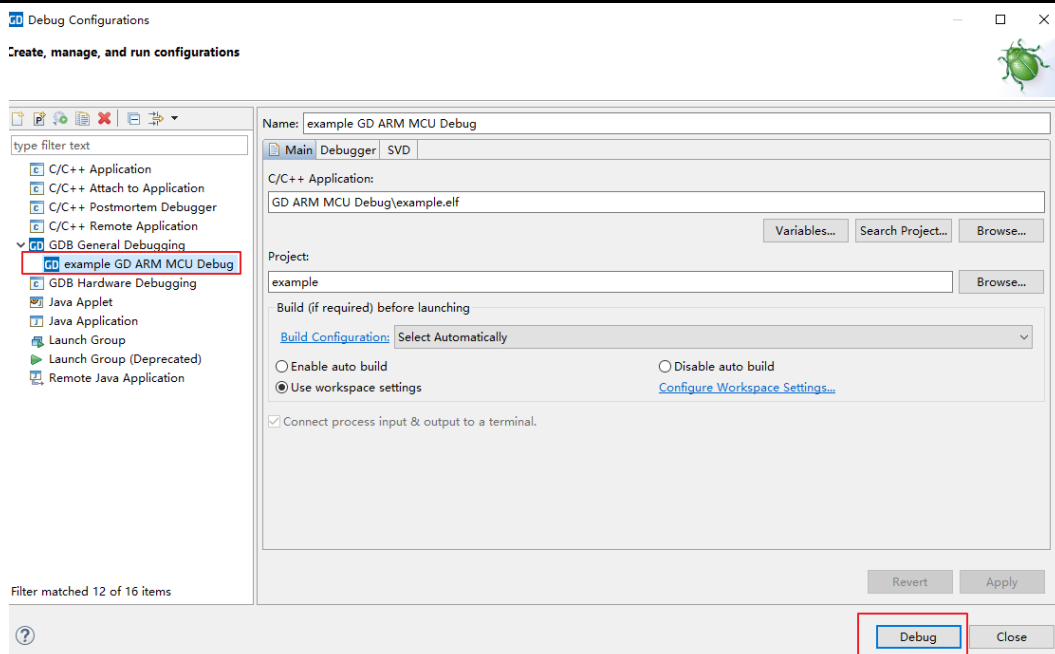
图 2-12. 编译工程



按照图 2-13. 调试工程操作完成工程调试。

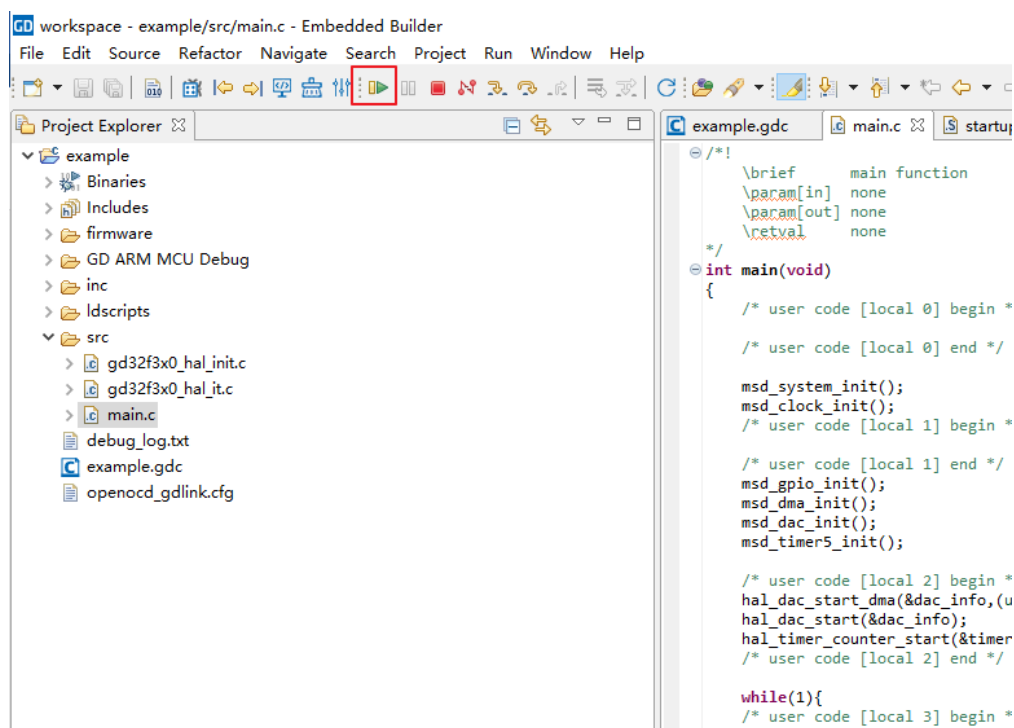
图 2-13. 调试工程





2.5.5. 运行代码

图 2-14. 运行代码



2.6. HAL 固件库文件描述

下表列举和描述了HAL固件库使用的主要文件。

表 2-1. 固件函数库文件描述

文件名	描述
gd32f3x0_libopt	包含了所有外设的头文件的头文件。它是唯一一个用户需要包括在自己应用中的HAL库配置文件，起到应用和库之间配置界面的作用。
main.c	主函数体示例。
gd32f3x0_hal_it.h	头文件，包含所有中断处理函数原型。
gd32f3x0_hal_it.c	外设中断函数文件。用户可以加入自己的中断程序代码。对于指向同一个中断向量的多个不同中断请求，可以利用函数通过判断外设的中断标志位来确定准确的中断源。 startup_gd32f3x0.s 文件中提供了这些函数的名称。
gd32f3x0_hal_xxx.h	外设PPP的头文件。包含外设PPP函数的定义，以及这些函数使用的变量。
gd32f3x0_hal_xxx.c	由C语言编写的外设PPP的驱动源程序文件。
gd32f3x0_hal_init.c	各外设模块初始化和复位函数所在文件
gd32f3x0_hal_init.h	头文件，包含所有外设模块初始化和复位函数原型声明
readme.txt	HAL固件库例程使用及配置说明文档

2.7. HAL 固件库 FAQ

2.7.1. 在很多模块中，函数的第一个形参均是设备信息结构体，该结构体的作用是什么？

在Analog、Ticker、I/Odevice类以及General类的部分模块中，均会有个特殊的结构体类型——设备信息结构体，命名风格为hal_xxx_dev_struct。

该结构体的功能类似于外设功能的一个实例，用于保存外设的某些重要信息，用于在不同函数中传递和调用。为了实现许多复杂的功能，让不同的函数能够协同工作，需要一个载体用于保存信息，同时由于HAL库并没有内存管理的功能，因此这个载体需要用户通过代码自行创建。正因如此，该结构体的使用具有如下几个特点：

1. 设备信息结构体必须创建为全局变量。
2. 设备信息结构体不需要用户去修改，用户需完成创建和调用对应的hal_xxx_struct_init函数完成结构体的初始化，此后仅需要将其作为形参传入函数即可。
3. 设备信息结构体与外设一一对应。例如用户同时使用了USART0、USART1两个外设。则需要对应两个设备信息结构体去分别保存其的设备信息。

2.7.2. HAL 库提供的函数具体是如何使用的？有没有例程可以参考？

HAL库的example文件夹内有丰富的例程可以供用户参考，例程基本覆盖了所有HAL库提供的函数。所有例程均为在上位机自动生成的代码基础上加工完成。

2.7.3. 为什么使用上位机自动生成配置代码后，外设无法工作？

考虑到用户可能希望模块在恰当的时机开始工作，HAL库中各模块初始化函数仅完成模块配置工作，并不使能模块。当代码自动生成后，需手动调用模块对应的使能函数hal_xxx_start()或功能类函数如hal_xxx_transmit_poll等，启动模块功能。

2.7.4. HAL 库能否进行裁剪，有哪些特性是可以修改的？

通过修改gd32f3x0_libopt.h头文件可以对HAL库进行裁剪并修改特性。

特性

HAL库提供了三个特性可以选择开启或关闭，如下图所示。

```

40  /*if set, flash operation (write and eraser) will reserve original data location
41  ...in out of targeted scope*/
42  #define FLASH_OPER_RESERVE_ORIGINAL_DATA ..... 1
43  /*if set, the parameters check will be implemented in function*/
44  #define HAL_PARAMETER_CHECK ..... 0
45  /*if set, print debug message according to level of marco 'HAL_DEBUG_PRINTF'
46  ...and halt code according to level of marco 'HAL_DEBUG_HALT_LEVEL'*/
47  #define HAL_DEBUG ..... 0
48
49  #if (1 == HAL_DEBUG)
50  #define HAL_DEBUG_PRINTF ..... printf
51  #define HAL_DEBUG_PRINTF_LEVEL ..... HAL_DEBUG_LVL_ALL
52  #define HAL_DEBUG_HALT_LEVEL ..... HAL_DEBUG_LVL_NONE
53
54  #define HAL_DEBUG_UART ..... USART0
55  #define HAL_DEBUG_EXTRA_DO
56  #endif /* 1 == HAL_DEBUG */

```

1. FLASH_OPER_RESERVE_ORIGINAL_DATA

控制MCU内部flash擦写模式的宏编译选项，影响gd32f3x0_hal_fmc.c文件的hal_fmc_region_write()/hal_fmc_region_erase()函数。

该宏值为0时，hal_fmc_region_write()在操作flash中非全0xFF区域时，会先将该区域所在page擦除，之后再执行写入操作，同page未被写入的区域存放的数据将会丢失。hal_fmc_region_erase()擦除单位为一个page。

该宏值为1时，hal_fmc_region_write()在操作flash中非全0xFF区域时，会先将同page非操作区保证操作区域外的数据不会丢失。hal_fmc_region_erase()擦除单位为1Byte。

2. HAL_PARAMETER_CHECK

控制HAL库函数是否进行形参的参数检查。

该宏值为0时，所有函数不进行形参的参数检查。

该宏值为1时，所有函数进行形参的参数检查。编译后代码的大小会增大。

3. HAL_DEBUG

控制HAL库代码的DEBUG功能

该宏值为0时，关闭DEBUG功能。

该宏值为1时，可以根据HAL_DEBUG_PRINTF_LEVEL及HAL_DEBUG_HALT_LEVEL的值打印HAL库函数运行的实时信息（是否发生警告或错误），并在检查到错误时（HAL_DEBUG_HALT_LEVEL的值需为HAL_DEBUG_LVL_FATAL | HAL_DEBUG_LVL_ERROR）中止代码的运行。开启后，编译后代码大小会增大。

裁剪

gd32f3x0_libopt.h头文件默认包含全部模块的头文件，如下图所示。若用户不需要使用某些模块，可以将其头文件进行屏蔽，之后编译器在编译时会忽略该模块。

#include "gd32f3x0_hal_dma.h"	#include "gd32f3x0_hal_dma.h"
#include "gd32f3x0_hal_fmc.h"	#include "gd32f3x0_hal_fmc.h"
#include "gd32f3x0_hal_pmu.h"	#include "gd32f3x0_hal_pmu.h"
#include "gd32f3x0_hal_dac.h"	#include "gd32f3x0_hal_dac.h"
#include "gd32f3x0_hal_gpio.h"	#include "gd32f3x0_hal_gpio.h"
#include "gd32f3x0_hal_rcu.h"	#include "gd32f3x0_hal_rcu.h"
#include "gd32f3x0_hal_exti.h"	#include "gd32f3x0_hal_exti.h"
#include "gd32f3x0_hal_sys.h"	#include "gd32f3x0_hal_sys.h"
#include "gd32f3x0_hal_syscfg.h"	#include "gd32f3x0_hal_syscfg.h"
#include "gd32f3x0_hal_nvic.h"	#include "gd32f3x0_hal_nvic.h"
#include "gd32f3x0_hal_cmp.h"	#include "gd32f3x0_hal_cmp.h"
#include "gd32f3x0_hal_cec.h"	#include "gd32f3x0_hal_cec.h"
#include "gd32f3x0_hal_crc.h"	#include "gd32f3x0_hal_crc.h"
#include "gd32f3x0_hal_adc.h"	#include "gd32f3x0_hal_adc.h"
#include "gd32f3x0_hal_ctc.h"	#include "gd32f3x0_hal_ctc.h"
#include "gd32f3x0_hal_fwdgt.h"	#include "gd32f3x0_hal_fwdgt.h"
#include "gd32f3x0_hal_tsi.h"	#include "gd32f3x0_hal_tsi.h"
#include "gd32f3x0_hal_wwdgt.h"	//#include "gd32f3x0_hal_wwdgt.h"
#include "gd32f3x0_hal_spi_com.h"	//#include "gd32f3x0_hal_spi_com.h"
#include "gd32f3x0_hal_spi.h"	//#include "gd32f3x0_hal_spi.h"
#include "gd32f3x0_hal_i2s.h"	#include "gd32f3x0_hal_i2s.h"
#include "gd32f3x0_hal_usart_com.h"	#include "gd32f3x0_hal_usart_com.h"
#include "gd32f3x0_hal_uart.h"	#include "gd32f3x0_hal_uart.h"
#include "gd32f3x0_hal_usrt.h"	#include "gd32f3x0_hal_usrt.h"
#include "gd32f3x0_hal_irda.h"	#include "gd32f3x0_hal_irda.h"
#include "gd32f3x0_hal_smartcard.h"	#include "gd32f3x0_hal_smartcard.h"
#include "gd32f3x0_hal_rtc.h"	//#include "gd32f3x0_hal_rtc.h"
#include "gd32f3x0_hal_i2c_com.h"	#include "gd32f3x0_hal_i2c_com.h"
#include "gd32f3x0_hal_i2c.h"	#include "gd32f3x0_hal_i2c.h"
#include "gd32f3x0_hal_smbus.h"	#include "gd32f3x0_hal_smbus.h"
#include "gd32f3x0_hal_timer.h"	#include "gd32f3x0_hal_timer.h"
#endif /* GD32F3X0_LIBOPT_H */	#endif /* GD32F3X0_LIBOPT_H */

2.7.5. 前缀为 **hal_**的函数与前缀为 **hals_**的函数有何区别？

HAL固件库各外设模块代码的命名可以分为两大类：1.前缀为**hal_**开头。2.前缀为**hals_**开头。

其中“前缀为**hal_**”的函数为HAL固件库的标准函数，各个外设模块的实现具备统一的风格和特性，实现的功能和特点符合2.2/2.3章节的相关介绍。

“前缀为**hals_**”的函数为补充性质的简单功能函数，对于HAL固件库函数架构体系来说，其不是必须存在的。各个外设模块的具体函数实现差异性很大，仅用于方便用户对外设模块的寄存器特定位或位域进行单一快速操作时使用。

3. 外设 HAL 固件库

3.1. 外设 HAL 固件库概述

外设固件库函数的描述格式如下表：

表 3-1. 外设固件库函数描述格式

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

3.2. ADC

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

3.2.1. 外设寄存器说明

ADC寄存器列表如下表所示：

表 3-2. ADC 寄存器

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

寄存器名称	寄存器描述
ADC_RSQ2	常规序列寄存器2
ADC_ISQ	注入序列寄存器
ADC_IDATAx	注入数据寄存器x
ADC_RDATA	常规数据寄存器
ADC_OVSAMPCTL	过采样控制寄存器

3.2.2. 外设库函数说明

ADC 库函数列表如下表所示：

表 3-3. ADC 库函数

库函数名称	库函数描述
hal_adc_struct_init	初始化ADC结构体
hal_adc_deinit	复位ADC外设
hal_adc_init	初始化ADC
hal_adc_calibration_start	ADC自校准
hal_adc_routine_channel_config	配置ADC常规通道
hal_adc_routine_rank_config	配置ADC常规通道的使用序号
hal_adc_start	启动ADC常规序列
hal_adc_stop	停止ADC常规序列
hal_adc_routine_conversion_poll	轮询方式实现ADC常规序列采样
hal_adc_routine_software_trigger_enable	使能ADC常规序列采样软件触发
hal_adc_start_interrupt	ADC常规序列中断启动
hal_adc_stop_interrupt	ADC常规序列中断停止
hal_adc_start_dma	ADC常规序列DMA启动
hal_adc_stop_dma	ADC常规序列DMA停止
hal_adc_inserted_channel_config	配置ADC注入通道
hal_adc_inserted_rank_config	配置ADC注入通道的使用序号
hal_adc_inserted_start	启动ADC注入序列
hal_adc_inserted_stop	停止ADC注入序列
hal_adc_inserted_conversion_poll	轮询方式实现ADC注入序列采样
hal_adc_inserted_software_trigger_enable	使能ADC注入序列采样软件触发
hal_adc_inserted_start_interrupt	ADC注入序列中断启动
hal_adc_inserted_stop_interrupt	ADC注入序列中断停止
hal_adc_watchdog_config	配置ADC看门狗事件
hal_adc_watchdog_interrupt_enable	使能ADC看门狗事件中断
hal_adc_watchdog_interrupt_disable	禁能ADC看门狗事件中断
hal_adc_watchdog_event_poll	轮询方式实现ADC看门狗事件
hal_adc_irq	ADC中断处理函数
hal_adc_irq_handle_set	设置ADC中断回调函数并使能ADC中断

库函数名称	库函数描述
hal_adc_irq_handle_all_reset	清除ADC中断回调函数
hal_adc_routine_value_get	获取ADC常规序列数据寄存器
hal_adc_inserted_value_get	获取ADC注入序列数据寄存器
hal_adc_error_get	获取ADC错误信息
hal_adc_state_get	获取ADC状态信息

枚举 hal_adc_struct_type_enum

表 3-4. 枚举 hal_adc_struct_type_enum

成员名称	功能描述
HAL_ADC_INIT_STRUCTURE	ADC初始化结构体
HAL_ADC_ROUTINE_RANK_CONFIG_STRUCTURE	ADC常规通道序号配置结构体
HAL_ADC_ROUTINE_CONFIG_STRUCTURE	ADC常规通道配置结构体
HAL_ADC_INSERTED_RANK_CONFIG_STRUCTURE	ADC注入通道序号配置结构体
HAL_ADC_INSERTED_CONFIG_STRUCTURE	ADC注入通道配置结构体
HAL_ADC_IRQ_STRUCTURE	ADC中断回调函数指针结构体
HAL_ADC_DMA_HANDLE_CB_STRUCTURE	ADC DMA回调函数指针结构体
HAL_ADC_WATCHDOG_CONFIG_STRUCTURE	ADC看门狗配置结构体
HAL_ADC_DEV_STRUCTURE	ADC设备信息结构体

枚举 hal_adc_state_enum

表 3-5. 枚举 hal_adc_state_enum

成员名称	功能描述
HAL_ADC_STATE_RESET	ADC禁能
HAL_ADC_STATE_READY	ADC准备

HAL_ADC_STATE_BUSY_SYSTEM	ADC内部繁忙（校准、初始化）
HAL_ADC_STATE_TIMEOUT	ADC超时
HAL_ADC_STATE_ROUTINE_BUSY	ADC常规序列正在转换
HAL_ADC_STATE_ROUTINE_EOC	ADC常规序列转换完成
HAL_ADC_STATE_INSERTED_BUSY	ADC注入序列正在转换
HAL_ADC_STATE_INSERTED_EOC	ADC注入序列转换完成
HAL_ADC_STATE_WATCHDOG	ADC模拟看门狗

枚举 hal_adc_error_enum

表 3-6. 枚举 hal_adc_error_enum

成员名称	功能描述
HAL_ADC_ERROR_NONE	无错误
HAL_ADC_ERROR_SYSTEM	ADC内部错误，如果时钟问题，使能、禁能错误状态
HAL_ADC_ERROR_DMA	DMA传输错误
HAL_ADC_ERROR_CONFIG	产生配置错误

枚举 hal_adc_oversample_shift_enum

表 3-7. 枚举 hal_adc_oversample_shift_enum

成员名称	功能描述
ADC_OVERSAMPLING_SHIFT_NONE	不移位
ADC_OVERSAMPLING_SHIFT_1B	移1位
ADC_OVERSAMPLING_SHIFT_2B	移2位
ADC_OVERSAMPLING_SHIFT_3B	移3位
ADC_OVERSAMPLING_SHIFT_4B	移4位
ADC_OVERSAMPLING_SHIFT_5B	移5位

ADC_OVERSAMPL E_SHIFT_6B	移6位
ADC_OVERSAMPL E_SHIFT_7B	移7位
ADC_OVERSAMPL E_SHIFT_8B	移8位

枚举 hal_adc_oversample_ratio_enum

表 3-8. 枚举 hal_adc_oversample_ratio_enum

成员名称	功能描述
ADC_OVERSAMPL E_RATIO_MUL2	过采样*2
ADC_OVERSAMPL E_RATIO_MUL4	过采样*4
ADC_OVERSAMPL E_RATIO_MUL8	过采样*8
ADC_OVERSAMPL E_RATIO_MUL16	过采样*16
ADC_OVERSAMPL E_RATIO_MUL32	过采样*32
ADC_OVERSAMPL E_RATIO_MUL64	过采样*64
ADC_OVERSAMPL E_RATIO_MUL128	过采样*128
ADC_OVERSAMPL E_RATIO_MUL256	过采样*256

枚举 hal_adc_routine_sequence_enum

表 3-9. 枚举 hal_adc_routine_sequence_enum

成员名称	功能描述
ADC_ROUTINE_SE QUENCE_0	ADC常规序列0
ADC_ROUTINE_SE QUENCE_1	ADC常规序列1
ADC_ROUTINE_SE QUENCE_2	ADC常规序列2
ADC_ROUTINE_SE QUENCE_3	ADC常规序列3
ADC_ROUTINE_SE QUENCE_4	ADC常规序列4
ADC_ROUTINE_SE QUENCE_5	ADC常规序列5

ADC_ROUTINE_SEQUENCE_6	ADC常规序列6
ADC_ROUTINE_SEQUENCE_7	ADC常规序列7
ADC_ROUTINE_SEQUENCE_8	ADC常规序列8
ADC_ROUTINE_SEQUENCE_9	ADC常规序列9
ADC_ROUTINE_SEQUENCE_10	ADC常规序列10
ADC_ROUTINE_SEQUENCE_11	ADC常规序列11
ADC_ROUTINE_SEQUENCE_12	ADC常规序列12
ADC_ROUTINE_SEQUENCE_13	ADC常规序列13
ADC_ROUTINE_SEQUENCE_14	ADC常规序列14
ADC_ROUTINE_SEQUENCE_15	ADC常规序列15

枚举 hal_adc_inserted_sequence_enum

表 3-10. 枚举 hal_adc_inserted_sequence_enum

成员名称	功能描述
ADC_INSERTED_SEQUENCE_0	ADC注入序列0
ADC_INSERTED_SEQUENCE_1	ADC注入序列1
ADC_INSERTED_SEQUENCE_2	ADC注入序列2
ADC_INSERTED_SEQUENCE_3	ADC注入序列3

结构体 hal_adc_irq_struct

表 3-11. 结构体 hal_adc_irq_struct

成员名称	功能描述
adc_eoc_handle	EOC中断处理回调函数
adc_eoic_handle	EOIC中断处理回调函数
adc_watchdog_handle	看门狗事件中断处理回调函数

结构体 hal_adc_dma_handle_cb_struct

表 3-12. 结构体 hal_adc_dma_handle_cb_struct

成员名称	功能描述
full_transcom_handle	ADC DMA发送完成中断处理回调函数
half_transcom_handle	ADC DMA半传输发送完成中断处理回调函数
error_handle	ADC DMA错误中断处理回调函数

结构体 hal_adc_dev_struct

表 3-13. 结构体 hal_adc_dev_struct

成员名称	功能描述
adc_irq	ADC中断回调函数指针结构体
*p_dma_adc	DMA设备信息结构体指针
adc_dma	ADC DMA回调函数指针结构体
error_state	ADC错误信息
mutex	ADC互斥集
state	ADC状态信息

结构体 hal_adc_init_struct

表 3-14. 结构体 hal_adc_init_struct

成员名称	功能描述
data_alignment	ADC采样数据对齐方式
resolution	ADC采样数据分辨率
scan_mode	ADC扫描模式
hardware_oversampling	过采样使能
oversample_trigger_mode	过采样触发模式
oversampling_shift	过采样移位
oversampling_ratio	过采样倍数

结构体 hal_adc_routine_rank_config_struct

表 3-15. 结构体 hal_adc_routine_rank_config_struct

成员名称	功能描述
channel	ADC常规序列通道
sampling_time	ADC常规序列采样时间
routine_sequence	ADC常规序列的序号

结构体 hal_adc_routine_config_struct

表 3-16. 结构体 hal_adc_routine_config_struct

成员名称	功能描述
routine_sequence_conversions	常规序列使能
routine_sequence_length	ADC常规序列长度
routine_sequence_external_trigger_select	ADC常规序列外部触发选择
continuous_mode	连续采样模式
discontinuous_mode	间断采样模式
number_of_conversions_in_discontinuous_mode	间断采样通道长度

结构体 hal_adc_inserted_rank_config_struct

表 3-17. 结构体 hal_adc_inserted_rank_config_struct

成员名称	功能描述
channel	ADC注入序列通道
sampling_time	ADC注入序列采样时间
data_offset	ADC注入序列数据偏移
inserted_sequence	ADC注入序列的序号

结构体 hal_adc_inserted_config_struct

表 3-18. 结构体 hal_adc_inserted_config_struct

成员名称	功能描述
inserted_sequence_conversions	注入序列使能
inserted_sequence_length	ADC注入序列长度
inserted_sequence_external_trigger_select	ADC注入序列外部触发选择
auto_convert	ADC注入序列自动转换模式
discontinuous_mode	间断采样模式

结构体 hal_adc_watchdog_config_struct

表 3-19. 结构体 hal_adc_watchdog_config_struct

成员名称	功能描述
------	------

routine_sequence_analog_watchdog	常规序列模拟看门狗使能
inserted_sequence_analog_watchdog	注入序列模拟看门狗使能
analog_watchdog_mode	模拟看门狗模式
analog_watchdog_channel_select	模拟看门狗通道选择
analog_watchdog_high_threshold	ADC模拟看门狗高阈值
analog_watchdog_low_threshold	ADC模拟看门狗低阈值

函数 hal_adc_struct_init

函数hal_adc_struct_init描述见下表：

表 3-20. 函数 hal_adc_struct_init

函数名称	hal_adc_struct_init
函数原型	void hal_adc_struct_init(hal_adc_struct_type_enum hal_struct_type, void *p_struct);
函数描述	初始化ADC结构体
先决条件	-
被调用函数	-
输入参数{in}	
hal_struct_type	结构体类型
HAL_ADC_INIT_STRUCTURE	ADC初始化结构体
HAL_ADC_ROUTINE_CONFIG_STRUCTURE	ADC常规通道配置结构体
HAL_ADC_ROUTINE_RANK_CONFIG_STRUCTURE	ADC常规通道的序号配置结构体
HAL_ADC_INSERTED_CONFIG_STRUCTURE	ADC注入通道配置结构体
HAL_ADC_INSERTED_RANK_CONFIG_STRUCTURE	ADC注入通道的序号配置结构体
HAL_ADC_IRQ_STRUCTURE	ADC中断回调函数指针结构体
HAL_ADC_DMA_HANDLER_CB_STRUCTURE	ADC DMA回调函数指针结构体
HAL_ADC_WATCHDOG_CONFIG_STRUCTURE	ADC看门狗配置结构体

HAL_ADC_DEV_STRU CT	ADC设备信息结构体
输入参数{in}	
p_struct	指向包含配置信息的ADC结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the ADC structure with the default values */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_struct_init(HAL_ADC_DEV_STRUCTURE, &adc_info);
```

函数 hal_adc_deinit

函数hal_adc_deinit描述见下表：

表 3-21. 函数 hal_adc_deinit

函数名称	hal_adc_deinit
函数原型	int32_t hal_adc_deinit(hal_adc_dev_struct *adc_dev);
函数描述	复位ADC外设
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如：

```
/* deinitialize ADC */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_deinit(&adc_info);
```

函数 hal_adc_init

函数hal_adc_init描述见下表：

表 3-22. 函数 hal_adc_init

函数名称	hal_adc_init
------	--------------

函数原型	int32_t hal_adc_init(hal_adc_dev_struct *adc_dev, hal_adc_init_struct *p_init);
函数描述	初始化ADC
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
p_init	指向ADC初始化结构体指针，结构体成员参考 表3-14. 结构体 hal_adc_init_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如：

```

/* initialize ADC */

hal_adc_dev_struct adc_info;

hal_adc_init_struct adc_init_parameter;

adc_init_parameter.data_alignment = ADC_LSB_ALIGNMENT;

adc_init_parameter.resolution = ADC_RESOLUTION_12B;

adc_init_parameter.scan_mode = DISABLE;

adc_init_parameter.hardware_oversampling = DISABLE;

hal_adc_init(&adc_info,&adc_init_parameter);

```

函数 hal_adc_calibration_start

函数hal_adc_calibration_start描述见下表：

表 3-23. 函数 hal_adc_calibration_start

函数名称	hal_adc_calibration_start
函数原型	int32_t hal_adc_calibration_start(hal_adc_dev_struct *adc_dev);
函数描述	ADC自校准
先决条件	在ADC初始化之后，启动采样之前调用
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	

-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如：

```
/* ADC calibration */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_calibration_start(&adc_info);
```

函数 hal_adc_routine_channel_config

函数hal_adc_routine_channel_config描述见下表：

表 3-24. 函数 hal_adc_routine_channel_config

函数名称	hal_adc_routine_channel_config
函数原型	int32_t hal_adc_routine_channel_config(hal_adc_dev_struct *adc_dev, hal_adc_routine_config_struct *p_rchannel);
函数描述	配置ADC常规通道
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
p_rchannel	ADC常规通道初始化结构体指针，结构体成员参考 表3-16. 结构体 hal_adc_routine_config_struct 。
输出参数{out}	
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如：

```
/* initialize ADC routine channel */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_routine_config_struct adc_routine_config_parameter;
```

```
adc_routine_config_parameter.routine_sequence_conversions = ENABLE;
```

```
adc_routine_config_parameter.routine_sequence_length = 1;
```

```
adc_routine_config_parameter.routine_sequence_external_trigger_select =  
ADC_EXTTRIG_ROUTINE_NONE;
```

```
adc_routine_config_parameter.continuous_mode = DISABLE;
```

```
adc_routine_config_parameter.discontinuous_mode = DISABLE;
```

```
hal_adc_routine_channel_config(&adc_info,&adc_routine_config_parameter);
```

函数 hal_adc_routine_rank_config

函数hal_adc_routine_rank_config描述见下表:

表 3-25. 函数 hal_adc_routine_rank_config

函数名称	hal_adc_routine_rank_config
函数原型	int32_t hal_adc_routine_rank_config(hal_adc_dev_struct *adc_dev, hal_adc_routine_rank_config_struct *p_rrank);
函数描述	配置ADC常规通道的使用序号
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
p_rrank	指向ADC常规通道的序号结构体的指针，结构体成员参考 表3-15. 结构体 hal_adc_routine_rank_config_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如:

```
/* configure ADC routine sequence: Sequence 0 */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_routine_rank_config_struct adc_routine_rank_config_parameter;
```

```
adc_routine_rank_config_parameter.channel = ADC_CHANNEL_0;
```

```
adc_routine_rank_config_parameter.sampling_time = ADC_SAMPLETIME_1POINT5;
```

```
adc_routine_rank_config_parameter.routine_sequence = ADC_ROUTINE_SEQUENCE_0;
```

```
hal_adc_routine_rank_config(&adc_info,&adc_routine_rank_config_parameter);
```

函数 hal_adc_start

函数hal_adc_start描述见下表:

表 3-26. 函数 hal_adc_start

函数名称	hal_adc_start
函数原型	hal_adc_start(hal_adc_dev_struct *adc_dev);

函数描述	启动ADC常规序列
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如：

```
/* enable ADC and start the conversion of routine sequence */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_start(&adc_info);
```

函数 hal_adc_stop

函数hal_adc_stop描述见下表：

表 3-27. 函数 hal_adc_stop

函数名称	hal_adc_stop
函数原型	hal_adc_stop(hal_adc_dev_struct *adc_dev);
函数描述	停止ADC常规序列
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如：

```
/* stop the conversion of routine sequence and disable ADC */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_stop(&adc_info);
```

函数 hal_adc_routine_conversion_poll

函数hal_adc_routine_conversion_poll描述见下表：

表 3-28. 函数 hal_adc_routine_conversion_poll

函数名称	hal_adc_routine_conversion_poll
函数原型	int32_t hal_adc_routine_conversion_poll(hal_adc_dev_struct *adc_dev, uint32_t timeout_ms);
函数描述	轮询方式实现ADC常规序列采样
先决条件	-
被调用函数	hal_sys_basetick_count_get
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
timeout_ms	超时时间，单位为ms
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_TIMEOUT

例如：

```
/* polling for ADC routine sequence conversion */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_routine_conversion_poll(&adc_info, 2);
```

函数 hal_adc_routine_software_trigger_enable

函数hal_adc_routine_software_trigger_enable描述见下表：

表 3-29. 函数 hal_adc_routine_software_trigger_enable

函数名称	hal_adc_routine_software_trigger_enable
函数原型	void hal_adc_routine_software_trigger_enable(hal_adc_dev_struct *adc_dev);
函数描述	使能ADC常规序列采样软件触发
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC routine sequence software trigger */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_routine_software_trigger_enable(&adc_info);
```

函数 hal_adc_start_interrupt

函数hal_adc_start_interrupt描述见下表：

表 3-30. 函数 hal_adc_start_interrupt

函数名称	hal_adc_start_interrupt
函数原型	int32_t hal_adc_start_interrupt(hal_adc_dev_struct *adc_dev, hal_adc_irq_struct *p_irq);
函数描述	ADC常规序列中断启动
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
p_irq	指向ADC中断回调函数结构体的指针，结构体成员参考 表3-11. 结构体 hal_adc_irq_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如：

```
/* start ADC EOC interrupt */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_irq_struct adc_irq_parameter;
```

```
void adc_irq_routine_sequence(void)
```

```
{
```

```
    /* user defined content */
```

```
}
```

```
adc_irq_parameter.adc_eoc_handle = adc_irq_routine_sequence;
```

```
hal_adc_start_interrupt (&adc_info, &adc_irq_parameter);
```

函数 hal_adc_stop_interrupt

函数hal_adc_stop_interrupt描述见下表：

表 3-31. 函数 hal_adc_stop_interrupt

函数名称	hal_adc_stop_interrupt
函数原型	int32_t hal_adc_stop_interrupt(hal_adc_dev_struct *adc_dev);
函数描述	ADC常规序列中断停止
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如：

```
/* stop ADC EOC interrupt */
hal_adc_dev_struct adc_info;
hal_adc_stop_interrupt(&adc_info);
```

函数 hal_adc_start_dma

函数hal_adc_start_dma描述见下表：

表 3-32. 函数 hal_adc_start_dma

函数名称	hal_adc_start_dma
函数原型	int32_t hal_adc_start_dma(hal_adc_dev_struct *adc_dev, uint32_t *pdata, uint32_t length, hal_adc_dma_handle_cb_struct *dmacb);
函数描述	ADC常规序列DMA启动
先决条件	-
被调用函数	hal_dma_struct_init \ hal_dma_start_interrupt
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
pdata	指向数据存储区的指针
输入参数{in}	
length	传输数据长度
输入参数{in}	
dmacb	指向ADC用户回调函数结构体的指针，结构体成员参考 表3-12. 结构体 hal_adc_dma_handle_cb_struct 。
输出参数{out}	
-	-
返回值	

int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_HARDWARE
----------------	---

例如:

```
/* start ADC DMA transmission */

hal_adc_dev_struct adc_info;

hal_adc_dma_handle_cb_struct adc_dma_parameter;

uint16_t adc_value[8];

void adc_dma_complete(void)
{
    /* user defined content */
}

/* initilize the user callback structure */

adc_dma_parameter.full_transcom_handle = adc_dma_complete;

adc_dma_parameter.error_handle      = NULL;

hal_adc_start_dma(&adc_info, (uint32_t *)&adc_value, 8, &adc_dma_parameter);
```

函数 hal_adc_stop_dma

函数hal_adc_stop_dma描述见下表:

表 3-33. 函数 hal_adc_stop_dma

函数名称	hal_adc_stop_dma
函数原型	int32_t hal_adc_stop_dma(hal_adc_dev_struct *adc_dev);
函数描述	ADC常规序列DMA停止
先决条件	-
被调用函数	hal_dma_stop
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针, 结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如:

```
/* stop ADC DMA transmission */

hal_adc_dev_struct adc_info;
```

```
hal_adc_stop_dma(&adc_info);
```

函数 hal_adc_inserted_channel_config

函数hal_adc_inserted_channel_config描述见下表:

表 3-34. 函数 hal_adc_inserted_channel_config

函数名称	hal_adc_inserted_channel_config
函数原型	int32_t hal_adc_inserted_channel_config(hal_adc_dev_struct *adc_dev, hal_adc_inserted_config_struct *p_ichannel);
函数描述	配置ADC注入通道
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
p_ichannel	指向ADC注入通道初始化结构体的指针，结构体成员参考 表3-18. 结构体 hal_adc_inserted_config_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT

例如:

```
/* initialize ADC inserted channel */

hal_adc_dev_struct adc_info;

hal_adc_inserted_config_struct adc_inserted_config_parameter;

adc_inserted_config_parameter.inserted_sequence_conversions = ENABLE;

adc_inserted_config_parameter.inserted_sequence_length = 2;

adc_inserted_config_parameter.inserted_sequence_external_trigger_select =
ADC_EXTTRIG_INSERTED_T0_TRGO;

adc_inserted_config_parameter.discontinuous_mode = DISABLE;

adc_inserted_config_parameter.auto_convert = DISABLE;

hal_adc_inserted_channel_config(&adc_info,&adc_inserted_config_parameter);
```

函数 hal_adc_inserted_rank_config

函数hal_adc_inserted_rank_config描述见下表:

表 3-35. 函数 `hal_adc_inserted_rank_config`

函数名称	<code>hal_adc_inserted_rank_config</code>
函数原型	<code>int32_t hal_adc_inserted_rank_config(hal_adc_dev_struct *adc_dev, hal_adc_inserted_rank_config_struct *p_irank);</code>
函数描述	配置ADC注入通道的使用序号
先决条件	-
被调用函数	-
输入参数{in}	
<code>adc_dev</code>	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 <code>hal_adc_dev_struct</code> 。
输入参数{in}	
<code>p_irank</code>	指向ADC注入通道的序号初始化结构体的指针，结构体成员参考 表3-17. 结构体 <code>hal_adc_inserted_rank_config_struct</code> 。
输出参数{out}	
-	-
返回值	
<code>int32_t</code>	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT

例如：

```

/* configure ADC inserted channel: Sequence 0 */

hal_adc_dev_struct adc_info;

hal_adc_inserted_rank_config_struct adc_inserted_rank_config_parameter;

adc_inserted_rank_config_parameter.data_offset = 0;

adc_inserted_rank_config_parameter.channel = ADC_CHANNEL_0;

adc_inserted_rank_config_parameter.sampling_time = ADC_SAMPLETIME_1POINT5;

adc_inserted_rank_config_parameter.inserted_sequence = ADC_INSERTED_SEQUENCE_0;

hal_adc_inserted_rank_config(&adc_info,&adc_inserted_rank_config_parameter);

```

函数 `hal_adc_inserted_start`

函数`hal_adc_inserted_start`描述见下表：

表 3-36. 函数 `hal_adc_inserted_start`

函数名称	<code>hal_adc_inserted_start</code>
函数原型	<code>int32_t hal_adc_inserted_start(hal_adc_dev_struct *adc_dev);</code>
函数描述	启动ADC注入序列
先决条件	-
被调用函数	-
输入参数{in}	

adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 <i>hal_adc_dev_struct</i> 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如：

```
/* enable ADC and start the conversion of inserted sequence */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_inserted_start(&adc_info);
```

函数 **hal_adc_inserted_stop**

函数hal_adc_inserted_stop描述见下表：

表 3-37. 函数 hal_adc_inserted_stop

函数名称	hal_adc_inserted_stop
函数原型	int32_t hal_adc_inserted_stop(hal_adc_dev_struct *adc_dev);
函数描述	停止ADC注入序列
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 <i>hal_adc_dev_struct</i> 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT

例如：

```
/* stop the conversion of inserted sequence and disable ADC */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_inserted_stop(&adc_info);
```

函数 **hal_adc_inserted_conversion_poll**

函数hal_adc_inserted_conversion_poll描述见下表：

表 3-38. 函数 hal_adc_inserted_conversion_poll

函数名称	hal_adc_inserted_conversion_poll
函数原型	int32_t hal_adc_inserted_conversion_poll(hal_adc_dev_struct *adc_dev, uint32_t timeout_ms);

函数描述	轮询方式实现ADC注入序列采样
先决条件	-
被调用函数	hal_sys_basetick_count_get
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
timeout_ms	超时时间，单位为ms
输出参数{out}	
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE, HAL_ERR_TIMEOUT

例如：

```
/* polling for ADC inserted sequence conversion */
hal_adc_dev_struct adc_info;
hal_adc_inserted_conversion_poll(&adc_info, 2);
```

函数 hal_adc_inserted_software_trigger_enable

函数hal_adc_inserted_software_trigger_enable描述见下表：

表 3-39. 函数 hal_adc_inserted_software_trigger_enable

函数名称	hal_adc_inserted_software_trigger_enable
函数原型	void hal_adc_inserted_software_trigger_enable(hal_adc_dev_struct *adc_dev);
函数描述	使能ADC注入序列采样软件触发
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC inserted sequence software trigger */
hal_adc_dev_struct adc_info;
hal_adc_inserted_software_trigger_enable(&adc_info);
```


函数 hal_adc_inserted_start_interrupt

函数hal_adc_inserted_start_interrupt描述见下表：

表 3-40. 函数 hal_adc_inserted_start_interrupt

函数名称	hal_adc_inserted_start_interrupt
函数原型	int32_t hal_adc_inserted_start_interrupt(hal_adc_dev_struct *adc_dev, hal_adc_irq_struct *p_irq);
函数描述	ADC注入序列中断启动
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
p_irq	指向ADC中断回调函数结构体的指针，结构体成员参考 表3-11. 结构体 hal_adc_irq_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_HARDWARE

例如：

```

/* start ADC EOIC interrupt */

hal_adc_dev_struct adc_info;

hal_adc_irq_struct adc_irq_parameter;

void adc_irq_inserted_sequence(void)
{
    /* user defined content */
}

adc_irq_parameter.adc_eoic_handle = adc_irq_inserted_sequence;

hal_adc_inserted_start_interrupt(&adc_info, &adc_irq_parameter);

```

函数 hal_adc_inserted_stop_interrupt

函数hal_adc_inserted_stop_interrupt描述见下表：

表 3-41. 函数 hal_adc_inserted_stop_interrupt

函数名称	hal_adc_inserted_stop_interrupt
函数原型	int32_t hal_adc_inserted_stop_interrupt(hal_adc_dev_struct *adc_dev);
函数描述	ADC注入序列中断停止

先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT

例如：

```
/* stop ADC EOIC interrupt */
hal_adc_dev_struct adc_info;
hal_adc_inserted_stop_interrupt(&adc_info);
```

函数 hal_adc_watchdog_config

函数hal_adc_watchdog_config描述见下表：

表 3-42. 函数 hal_adc_watchdog_config

函数名称	hal_adc_watchdog_config
函数原型	int32_t hal_adc_watchdog_config(hal_adc_dev_struct *adc_dev, hal_adc_watchdog_config_struct *p_watchdog);
函数描述	配置ADC看门狗事件
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
p_watchdog	指向ADC看门狗配置结构体的指针，结构体成员参考 表3-19. 结构体 hal_adc_watchdog_config_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* configure ADC watchdog */
hal_adc_dev_struct adc_info;
hal_adc_watchdog_struct adc_watchdog_config_parameter;
```

```

adc_watchdog_config_parameter.routine_sequence_analog_watchdog = ENABLE;

adc_watchdog_config_parameter.inserted_sequence_analog_watchdog = DISABLE;

adc_watchdog_config_parameter.analog_watchdog_mode
ADC_WATCHDOG_MODE_SINGLE_CHANNEL;

adc_watchdog_config_parameter.analog_watchdog_channel_select = ADC_CHANNEL_0;

adc_watchdog_config_parameter.analog_watchdog_high_threshold = 0xAFF;

adc_watchdog_config_parameter.analog_watchdog_low_threshold = 0x0;

hal_adc_watchdog_config(&adc_info, &adc_watchdog_config_parameter);

```

函数 hal_adc_watchdog_interrupt_enable

函数hal_adc_watchdog_interrupt_enable描述见下表：

表 3-43. 函数 hal_adc_watchdog_interrupt_enable

函数名称	hal_adc_watchdog_interrupt_enable
函数原型	int32_t hal_adc_watchdog_interrupt_enable(hal_adc_dev_struct *adc_dev, hal_adc_irq_struct *p_irq);
函数描述	使能ADC看门狗事件中断
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
p_irq	指向ADC中断回调函数结构体的指针，结构体成员参考 表3-11. 结构体 hal_adc_irq_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/ HAL_ERR_ADDRESS

例如：

```

/* enable watchdog interrupt */

hal_adc_dev_struct adc_info;

hal_adc_irq_struct adc_irq_parameter;

void adc_irq_watchdog(void)

{

    /* user defined content */

```

}

```
adc_irq_parameter.adc_watchdog_handle = adc_irq_watchdog;
```

```
hal_adc_watchdog_interrupt_enable(&adc_info, &adc_irq_parameter);
```

函数 hal_adc_watchdog_interrupt_disable

函数hal_adc_watchdog_interrupt_disable描述见下表:

表 3-44. 函数 hal_adc_watchdog_interrupt_disable

函数名称	hal_adc_watchdog_interrupt_disable
函数原型	int32_t hal_adc_watchdog_interrupt_disable(hal_adc_dev_struct*adc_dev);
函数描述	禁能ADC看门狗事件中断
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如:

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_watchdog_interrupt_disable(&adc_info, &adc_irq_parameter);
```

函数 hal_adc_watchdog_event_poll

函数hal_adc_watchdog_event_poll描述见下表:

表 3-45. 函数 hal_adc_watchdog_event_poll

函数名称	hal_adc_watchdog_event_poll
函数原型	int32_t hal_adc_watchdog_event_poll(hal_adc_dev_struct*adc_dev, uint32_t timeout_ms);
函数描述	轮询方式实现ADC看门狗事件
先决条件	-
被调用函数	hal_sys_basetick_count_get
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
timeout_ms	超时时间，单位为ms
输出参数{out}	

-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_TIMEOUT

例如：

```
/* polling for ADC inserted sequence conversion */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_watchdog_event_poll(&adc_info, 2);
```

函数 hal_adc_irq

函数hal_adc_irq描述见下表：

表 3-46. 函数 hal_adc_irq

函数名称	hal_adc_irq
函数原型	void hal_adc_irq(hal_adc_dev_struct *adc_dev);
函数描述	ADC中断处理函数
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the function is used in the relative interrupt routine */
```

```
hal_adc_dev_struct adc_info;
```

```
void ADC_CMP_IRQHandler (void)
```

```
{
```

```
    hal_adc_irq(&adc_info);
```

```
}
```

函数 hal_adc_irq_handle_set

函数hal_adc_irq_handle_set描述见下表：

表 3-47. 函数 hal_adc_irq_handle_set

函数名称	hal_adc_irq_handle_set
------	------------------------

函数原型	void hal_adc_irq_handle_set(hal_adc_dev_struct *adc_dev, hal_adc_irq_struct *p_irq);
函数描述	设置ADC中断回调函数并使能ADC中断
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
p_irq	指向ADC中断回调函数结构体的指针
hal_irq_handle_cb	指向ADC中断回调函数结构体的指针，结构体成员参考 表3-11. 结构体 hal_adc_irq_struct 。
NULL	结构体为空
HAL_INTERRUPT_ENABLE_ONLY	使能ADC中断，但中断回调函数为空
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* set user-defined interrupt callback function */

hal_adc_dev_struct adc_info;

hal_adc_irq_struct adc_irq_parameter;

void adc_irq_routine_sequence (void)
{
    /* user defined content */
}

/* set the EOC handle function */

adc_irq_parameter.adc_eoc_handle = adc_irq_routine_sequence;

hal_adc_irq_handle_set(&adc_info, &adc_irq_parameter);

```

函数 hal_adc_irq_handle_all_reset

函数hal_adc_irq_handle_all_reset描述见下表：

表 3-48. 函数 hal_adc_irq_handle_all_reset

函数名称	hal_adc_irq_handle_all_reset
函数原型	void hal_adc_irq_handle_all_reset(hal_adc_dev_struct *adc_dev);

函数描述	清除ADC中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback function */
```

```
hal_adc_dev_struct adc_info;
```

```
hal_adc_irq_handle_all_reset(&adc_info);
```

函数 hal_adc_routine_value_get

函数hal_adc_routine_value_get描述见下表：

表 3-49. 函数 hal_adc_routine_value_get

函数名称	hal_adc_routine_value_get
函数原型	uint16_t hal_adc_routine_value_get(hal_adc_dev_struct *adc_dev);
函数描述	获取ADC常规序列数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
Int16_t	the routine sequence conversion result(0-0xFFFF)

例如：

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

```
hal_adc_dev_struct adc_info;
```

```
uint16_t adc_value = 0;
```

```
adc_value = hal_adc_routine_value_get (&adc_info);
```

函数 hal_adc_inserted_value_get

函数hal_adc_inserted_value_get描述见下表：

表 3-50. 函数 hal_adc_inserted_value_get

函数名称	hal_adc_inserted_value_get
函数原型	uint16_t hal_adc_inserted_value_get(hal_adc_dev_struct *adc_dev, uint8_t inschannel_sequence);
函数描述	获取ADC注入序列数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输入参数{in}	
inschannel_sequence	注入序列的序号
ADC_INSERTED_SEQUENCE_x (x=0..3)	注入序列的序号x，x=0,1,2,3
输出参数{out}	
-	-
返回值	
uint16_t	ADC注入序列转换值(0-0xFFFF)

例如：

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

```
hal_adc_dev_struct adc_info;
```

```
uint16_t adc_value = 0;
```

```
adc_value = hal_adc_inserted_value_get(&adc_info, ADC_INSERTED_SEQUENCE_0);
```

函数 hal_adc_error_get

函数hal_adc_error_get描述见下表：

表 3-51. 函数 hal_adc_error_get

函数名称	hal_adc_error_get
函数原型	uint32_t hal_adc_error_get(hal_adc_dev_struct *adc_dev);
函数描述	获取ADC错误信息
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	

-	-
返回值	
uint32_t	ADC错误信息

例如：

```
/* get ADC error */
hal_adc_dev_struct adc_info;
uint32_t adc_error = 0;
adc_error = hal_adc_error_get(&adc_info);
```

函数 hal_adc_state_get

函数hal_adc_state_get描述见下表：

表 3-52. 函数 hal_adc_state_get

函数名称	hal_adc_state_get
函数原型	uint32_t hal_adc_state_get(hal_adc_dev_struct *adc_dev);
函数描述	获取ADC状态信息
先决条件	-
被调用函数	-
输入参数{in}	
adc_dev	指向ADC设备信息结构体的指针，结构体成员参考 表3-13. 结构体 hal_adc_dev_struct 。
输出参数{out}	
-	-
返回值	
uint32_t	ADC状态

例如：

```
/* get ADC state */
hal_adc_dev_struct adc_info;
uint32_t adc_state = 0;
adc_state = hal_adc_state_get (&adc_info);
```

3.3. CEC

消费电子控制（CEC）是 HDMI（高清多媒体接口）标准的一部分。CEC作为一种协议，提供了在用户环境中各种音像制品之间的高级控制功能。章节[3.3.1](#)描述了CEC的寄存器列表，章节[3.3.2](#)对CEC库函数进行说明。

3.3.1. 外设寄存器说明

CEC寄存器列表如下表所示：

表 3-53. CEC 寄存器

寄存器名称	寄存器描述
CEC_CTL	控制寄存器
CEC_CFG	配置寄存器
CEC_TDATA	数据发送寄存器
CEC_RDATA	数据接收寄存器
CEC_INTF	中断标志寄存器
CEC_INTEN	中断使能寄存器

3.3.2. 外设库函数说明

CEC库函数列表如下表所示：

表 3-54. CEC 库函数

库函数名称	库函数描述
hal_cec_struct_init	初始化CEC结构体
hal_cec_init	初始化CEC
hal_cec_deinit	复位CEC外设
hal_cec_start	启动CEC
hal_cec_stop	停止CEC
hal_cec_transmit_interrupt	通过中断方式传输数据
hal_cec_receive_interrupt	通过中断方式接收数据
hal_cec_irq_handle_set	设置CEC中断回调函数并使能CEC中断
hal_cec_irq_handle_all_reset	清除CEC中断回调函数
hal_cec_irq	CEC中断处理函数

枚举 hal_cec_state_enum

表 3-55. 枚举 hal_cec_state_enum

成员名称	功能描述
HAL_CEC_STATE_NONE	无（默认值）
HAL_CEC_STATE_RESET	CEC未被初始化或停止
HAL_CEC_STATE_READY	CEC就绪
HAL_CEC_STATE_BUSY	CEC忙
HAL_CEC_STATE_BUSY_RX	CEC正在接收
HAL_CEC_STATE_BUSY_TX	CEC正在发送

枚举 `hal_cec_struct_type_enum`

表 3-56. 枚举 `hal_cec_struct_type_enum`

成员名称	功能描述
<code>HAL_CEC_INIT_STRUCT</code>	CEC初始化结构体
<code>HAL_CEC_IRQ_STRUCT</code>	CEC中断回调函数指针结构体
<code>HAL_CEC_DEV_STRUCT</code>	CEC设备信息结构体

枚举 `hal_cec_error_enum`

表 3-57. 枚举 `hal_cec_error_enum`

成员名称	功能描述
<code>HAL_CEC_ERROR_NONE</code>	无错误
<code>HAL_CEC_ERROR_RO</code>	溢出错误
<code>HAL_CEC_ERROR_BRE</code>	位上升错误
<code>HAL_CEC_ERROR_BPSE</code>	短位周期错误
<code>HAL_CEC_ERROR_BPLE</code>	长位周期错误
<code>HAL_CEC_ERROR_RAE</code>	接收ACK错误
<code>HAL_CEC_ERROR_ARBF</code>	仲裁失败
<code>HAL_CEC_ERROR_TU</code>	发送数据缓冲区欠载
<code>HAL_CEC_ERROR_TERR</code>	发送错误
<code>HAL_CEC_ERROR_TAERR</code>	发送ACK错误标志位

结构体 `hal_cec_irq_struct`

表 3-58. 结构体 `hal_cec_irq_struct`

成员名称	功能描述
<code>cec_tx_handle</code>	CEC发送处理中断回调函数
<code>cec_rx_handle</code>	CEC接收处理中断回调函数

结构体 `hal_cec_dev_struct`

表 3-59. 结构体 `hal_cec_dev_struct`

成员名称	功能描述
<code>cec_irq</code>	CEC中断回调函数指针结构体
<code>state</code>	CEC状态
<code>error_state</code>	CEC错误状态
<code>*tx_buffer</code>	发送缓存地址
<code>tx_count</code>	发送字节数
<code>*rx_buffer</code>	接收缓存地址
<code>rx_count</code>	接收字节数
<code>*err_callback</code>	CEC错误回调函数
<code>*tx_callback</code>	CEC发送回调函数
<code>*rx_callback</code>	CEC接收回调函数

成员名称	功能描述
mutex	互斥锁和解锁状态
*priv	隐私数据

结构体 hal_cec_init_struct

表 3-60. 结构体 hal_cec_init_struct

成员名称	功能描述
signal_free_time	信号空闲时间
reception_bit_timing_tolerance	接收位时间宽容度
sft_start_option_bit	SFT开始选项位
listen_mode	监听模式使能位
own_address	自身地址
bre_stop_receive	监测到BRE时是否停止接收信息
bre_generate_error	在单次传播模式下监测到BRE的时候产生错误位
blpe_generate_error	在单播模式下监测到BPLe的时候产生错误位
not_generate_error_broadcast	广播信息模式下不产生错误位

函数 hal_cec_struct_init

函数hal_cec_struct_init描述见下表:

表 3-61. 函数 hal_cec_struct_init

函数名称	hal_cec_struct_init
函数原型	void hal_cec_struct_init(hal_cec_struct_type_enum hal_struct_type, void *p_struct);
功能描述	初始化CEC结构体
先决条件	-
被调用函数	-
输入参数{in}	
hal_struct_type	结构体类型
HAL_CEC_INIT_ST RUCT	CEC初始化结构体
HAL_CEC_IRQ_ST RUCT	CEC中断回调函数指针结构体
HAL_CEC_DEV_ST RUCT	CEC设备信息结构体
输入参数{in}	
p_struct	指向包含配置信息的CEC结构体的指针
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* initialize the CEC structure with the default values */

hal_cec_dev_struct cec_info;

hal_cec_struct_init(HAL_CEC_DEV_STRUCT, &cec_info);
```

函数 hal_cec_init

函数hal_cec_init描述见下表:

表 3-62. 函数 hal_cec_init

函数名称	hal_cec_init
函数原型	int32_t hal_cec_init(hal_cec_dev_struct *cec_dev, hal_cec_init_struct *cec);
功能描述	初始化CEC外设
先决条件	-
被调用函数	-
输入参数{in}	
cec_dev	指向CEC设备信息结构体的指针，结构体成员参考 表3-59. 结构体 hal_cec_dev_struct
输入参数{in}	
cec	指向CEC初始化结构体的指针，结构体成员参考 表3-60. 结构体 hal_cec_init_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* initialize CEC */

hal_cec_init_struct cec_init_parameter;

hal_cec_struct_init(HAL_CEC_INIT_STRUCT, &cec_init_parameter);

hal_cec_struct_init(HAL_CEC_DEV_STRUCT, &cec_info);

cec_init_parameter.signal_free_time = CEC_SFT_PROTOCOL_PERIOD;

cec_init_parameter.reception_bit_timing_tolerance = CEC_STANTARD_RTOL;

cec_init_parameter.sft_start_option_bit = CEC_SFT_START_STAOM;

cec_init_parameter.listen_mode = CEC_PARTIAL_LISTENING_MODE;

cec_init_parameter.own_address = CEC_OWN_ADDRESS2;

cec_init_parameter.bre_stop_receive = CEC_NOT_STOP_RECEPTION;

cec_init_parameter.bre_generate_error = CEC_NO_RISING_PERIOD_ERROR;
```

```
cec_init_parameter.blpe_generate_error = CEC_NO_LONG_PERIOD_ERROR;

cec_init_parameter.not_generate_error_broadcast = CEC_GEN_BROADCAST_ERROR;

hal_cec_init(&cec_info,&cec_init_parameter);
```

函数 hal_cec_deinit

函数hal_cec_deinit描述见下表:

表 3-63. 函数 hal_cec_deinit

函数名称	hal_cec_deinit
函数原型	int32_t hal_cec_deinit(hal_cec_dev_struct *cec_dev)
功能描述	复位CEC外设
先决条件	-
被调用函数	_cec_interrupt_disable
输入参数{in}	
cec_dev	指向cec设备信息结构体的指针，结构体成员参考 表3-59. 结构体 hal_cec_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* deinitialize CEC */

hal_cec_dev_struct cec_info;

hal_cec_deinit(&cec_info);
```

函数 hal_cec_start

函数hal_cec_start描述见下表:

表 3-64. 函数 hal_cec_start

函数名称	hal_cec_start
函数原型	int32_t hal_cec_start(hal_cec_dev_struct *cec_dev);
功能描述	启动CEC
先决条件	-
被调用函数	-
输入参数{in}	
cec_dev	指向CEC设备信息结构体的指针，结构体成员参考 表3-59. 结构体 hal_cec_dev_struct
输出参数{out}	
-	-
返回值	

int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE
---------	-------------------------------

例如:

```
/* start CEC */

hal_cec_dev_struct cec_info;

hal_cec_start(&cec_info);
```

函数 hal_cec_stop

函数hal_cec_stop描述见下表:

表 3-65. 函数 hal_cec_start

函数名称	hal_cec_stop
函数原型	int32_t hal_cec_stop(hal_cec_dev_struct *cec_dev);
功能描述	停止CEC
先决条件	-
被调用函数	-
输入参数{in}	
cec_dev	指向CEC设备信息结构体的指针, 结构体成员参考 表3-59. 结构体 hal_cec_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* stop CEC */

hal_cec_dev_struct cec_info;

hal_cec_stop(&cec_info);
```

函数 hal_cec_transmit_interrupt

函数hal_cec_transmit_interrupt描述见下表:

表 3-66. 函数 hal_cec_transmit_interrupt

函数名称	hal_cec_transmit_interrupt
函数原型	int32_t hal_cec_transmit_interrupt(hal_cec_dev_struct *cec_dev, uint32_t tx_length, uint8_t *p_buffer, uint8_t header_addr, uint8_t destination_addr, hal_cec_user_cb p_func);
功能描述	通过中断方式传输数据
先决条件	-
被调用函数	-
输入参数{in}	

cec_dev	指向CEC设备信息结构体的指针，结构体成员参考 表3-59. 结构体 <i>hal_cec_dev_struct</i>
输入参数{in}	
tx_length	发送数据长度
输入参数{in}	
p_buffer	指向数据缓存
输入参数{in}	
header_addr	头地址
输入参数{in}	
destination_addr	目的地址
输入参数{in}	
p_func	CEC发送中断回调
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```

/* transmit amounts of data by interrupt method */

cec_info.err_callback = HAL_CEC_ErrCallback;

uint32_t Tx_Size      = 0x0;

uint8_t Tx_Burrer[2] = {0xa5, 0x5a};

__IO uint8_t txstatus = 0U;

void HAL_CEC_TxCpltCallback(hal_cec_dev_struct *cec_dev)
{
    /* End of transmission */

    txstatus = 1;
}

hal_cec_transmit_interrupt(&cec_info, Tx_Size, Tx_Burrer, 1, 2, HAL_CEC_TxCpltCallback);

```

函数 hal_cec_receive_interrupt

函数hal_cec_receive_interrupt描述见下表:

表 3-67. 函数 hal_cec_receive_interrupt

函数名称	hal_cec_receive_interrupt
函数原型	int32_t hal_cec_receive_interrupt(hal_cec_dev_struct *cec_dev, uint8_t *p_buffer, hal_cec_user_cb p_func);
功能描述	通过中断接收数据

先决条件	-
被调用函数	-
输入参数{in}	
cec_dev	指向CEC设备信息结构体的指针，结构体成员参考 表3-59. 结构体 hal_cec_dev_struct
输入参数{in}	
p_buffer	指向数据缓存
输入参数{in}	
p_func	CEC接收中断回调
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```

/* receive amounts of data by interrupt method */
cec_info.err_callback = HAL_CEC_ErrCallback;
uint8_t Rx_Burrer[CEC_MAX_BUFFER_LEN] = {0};
__IO uint8_t rxstatus = 0U;

void HAL_CEC_RxCpltCallback(hal_cec_dev_struct *cec_dev)
{
    rxstatus = 1;
}

hal_cec_receive_interrupt(&cec_info, Rx_Burrer, HAL_CEC_RxCpltCallback);

```

函数 hal_cec_irq_handle_set

函数hal_cec_irq_handle_set描述见下表:

表 3-68. 函数 hal_cec_irq_handle_set

函数名称	hal_cec_irq_handle_set
函数原型	void hal_cec_irq_handle_set(hal_cec_dev_struct *cec_dev, hal_cec_irq_struct *p_irq);
功能描述	设置CEC中断回调函数并使能CEC中断
先决条件	-
被调用函数	-
输入参数{in}	
cec_dev	指向CEC设备信息结构体的指针，结构体成员参考 表3-59. 结构体 hal_cec_dev_struct
输入参数{in}	

p_irq	指向CEC中断回调函数结构体的指针，结构体成员参考 表3-58. 结构体 <i>hal_cec_irq_struct</i>
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* set user-defined interrupt callback function */
```

```
hal_cec_dev_struct cec_info;
```

```
hal_cec_irq_struct cec_irq;
```

```
cec_irq.cec_tx_handle = cec_tx_func;
```

```
hal_cec_irq_handle_set(&cec_info,&cec_irq);
```

函数 **hal_cec_irq_handle_all_reset**

函数hal_cec_irq_handle_all_reset描述见下表:

表 3-69. 函数 hal_cec_irq_handle_all_reset

函数名称	hal_cec_irq_handle_all_reset
函数原型	void hal_cec_irq_handle_all_reset(hal_cec_dev_struct *cec_dev);
功能描述	清除CEC中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	
cec_dev	指向CEC设备信息结构体的指针，结构体成员参考 表3-59. 结构体 <i>hal_cec_dev_struct</i>
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* reset all user-defined interrupt callback function */
```

```
hal_cec_dev_struct cec_info;
```

```
hal_cec_irq_handle_all_reset(&cec_info);
```

函数 **hal_cec_irq**

函数hal_cec_irq描述见下表:

表 3-70. 函数 hal_cec_irq

函数名称	hal_cec_irq
函数原型	void hal_cec_irq(hal_cec_dev_struct *cec_dev);
功能描述	CEC中断处理函数
先决条件	-
被调用函数	-
输入参数{in}	
cec_dev	指向CEC设备信息结构体的指针，结构体成员参考 表3-59. 结构体 hal_cec_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* the function is used in the relative interrupt routine */
hal_cec_dev_struct cec_info;

CEC_IRQHandler (void)
{
    hal_cec_irq(&cec_info);
}
```

3.4. CMP

CMP通用比较器可独立工作，其输出端口可用于I/O口，也可和定时器结合使用。比较器可通过模拟信号将MCU从低功耗模式中唤醒，在一定的条件下，可将模拟信号作为触发源，结合定时器的PWM输出，可以实现电流控制。章节[3.4.1](#)描述了CMP的寄存器列表，章节[3.4.2](#)对CMP库函数进行说明。

3.4.1. 外设寄存器说明

CMP寄存器列表如下表所示:

表 3-71. CMP 寄存器

寄存器名称	寄存器描述
CMP_CS	控制状态寄存器

3.4.2. 外设库函数说明

CMP库函数列表如下表所示:

表 3-72. CMP 库函数

库函数名称	库函数描述
hal_cmp_struct_init	结构体变量恢复缺省值
hal_cmp_deinit	初始化设备信息结构体，复位外设
hal_cmp_init	初始化外设
hal_cmp_start	启动CMP模块功能
hal_cmp_stop	停止CMP模块功能
hal_cmp_start_interrupt	启动CMP模块并打开相关中断
hal_cmp_stop_interrupt	停止CMP模块并关闭相关中断
hal_cmp_irq_handle_set	中断回调函数设置
hal_cmp_irq_handle_all_reset	复位中断回调函数
hal_cmp_irq	CMP中断服务函数
hal_cmp_lock	CMP上锁
hal_cmp_output_level_get	获取CMP输出电平
hal_cmp_state_get	获取CMP状态信息

枚举 hal_cmp_state_enum

表 3-73. 枚举 hal_cmp_state_enum

成员名称	功能描述
HAL_CMP_STATE_NONE	无（默认值）
HAL_CMP_STATE_RESET	CMP未被初始化或停止
HAL_CMP_STATE_BUSY	CMP繁忙
HAL_CMP_STATE_TIMEOUT	CMP产生超时
HAL_CMP_STATE_ERROR	CMP出错
HAL_CMP_STATE_READY	CMP准备

枚举 hal_cmp_struct_type_enum

表 3-74. 枚举 hal_cmp_struct_type_enum

成员名称	功能描述
HAL_CMP_INIT_STRUCT	CMP初始化结构体
HAL_CMP_DEV_STRUCT	CMP设备信息结构体

枚举 hal_cmp_noninverting_input_enum

表 3-75. 枚举 hal_cmp_noninverting_input_enum

成员名称	功能描述
CMP0_IP_PA1	CMP0正向输入选择PA1
CMP0_IP_PA4_SWCLOSE	CMP0正向输入选择PA4（开关模式使能）
CMP1_IP_PA3	CMP1正向输入选择PA3
CMP1_IP_OF_CMP0	CMP1正向输入选择和CMP0一致（窗口比较模式）

枚举 `hal_cmp_exti_type_enum`

表 3-76. 枚举 `hal_cmp_exti_type_enum`

成员名称	功能描述
<code>CMP_EXTI_NONE</code>	无EXTI事件/中断使能
<code>CMP_EXTI_INT_RISING</code>	使能EXTI上升沿中断
<code>CMP_EXTI_INT_FALLING</code>	使能EXTI下降沿中断
<code>CMP_EXTI_INT_BOTH</code>	使能EXTI上升、下降沿中断
<code>CMP_EXTI_EVENT_RISING</code>	使能EXTI上升沿事件
<code>CMP_EXTI_EVENT_FALLING</code>	使能EXTI下降沿事件
<code>CMP_EXTI_EVENT_BOTH</code>	使能EXTI上升、下降沿事件

结构体 `hal_cmp_init_struct`

表 3-77. 结构体 `hal_cmp_init_struct`

成员名称	功能描述
<code>inverting_input</code>	比较器方向输入端选择
<code>noninverting_input</code>	比较器正向输入端选择
<code>outputsel</code>	输出选择
<code>polarity</code>	输出极性选择
<code>mode</code>	工作模式选择
<code>hysteresis</code>	迟滞等级选择
<code>exti_type</code>	中断/事件选择，触发边沿选择

结构体 `hal_cmp_dev_struct`

表 3-78. 结构体 `hal_cmp_dev_struct`

成员名称	功能描述
<code>periph</code>	模块实例指定， <code>CMP0</code> 或 <code>CMP1</code>
<code>cmp_irq</code>	中断回调函数结构体
<code>state</code>	比较器状态
<code>output_level</code>	输出电平

结构体 `hal_cmp_irq_struct`

表 3-79. 结构体 `hal_cmp_irq_struct`

成员名称	功能描述
<code>output_changed_handle</code>	比较输出结果变化中断回调函数

函数 `hal_cmp_struct_init`

函数`hal_cmp_struct_init`描述见下表:

表 3-80. 函数 `hal_cmp_struct_init`

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

函数原型	uint32_t hal_cmp_struct_init(hal_cmp_struct_type_enum hal_struct_type, void *p_struct)
功能描述	结构体变量恢复缺省值
先决条件	-
被调用函数	-
输入参数{in}	
hal_struct_type -	结构体类型-
HAL_CMP_INIT_STRUCTURE	初始化结构体
HAL_CMP_DEV_STRUCTURE	设备信息结构体
输入参数{out}	
p_struct	需要被初始化的结构体指针
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如：

```
/* CMP initializing structure initialize */
hal_cmp_init_struct cmp_init_parameter;
hal_cmp_struct_init(HAL_CMP_INIT_STRUCTURE, &cmp_init_parameter);
```

函数 hal_cmp_deinit

函数hal_cmp_deinit描述见下表：

表 3-81. 函数 hal_cmp_deinit

函数名称	hal_cmp_deinit
函数原型	int32_t hal_cmp_deinit(hal_cmp_dev_struct *cmp_dev);
功能描述	初始化设备信息结构体，复位外设
先决条件	-
被调用函数	hal_exti_internal_deinit
输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```
/* CMP device structure initialize */
hal_cmp_dev_struct cmp_dev_parameter;
hal_cmp_deinit(&cmp_dev_parameter);
```

函数 hal_cmp_init

函数hal_cmp_init描述见下表:

表 3-82. 函数 hal_cmp_init

函数名称	hal_cmp_init
函数原型	int32_t hal_cmp_init(hal_cmp_dev_struct *cmp_dev, uint32_t periph, hal_cmp_init_struct *p_init);
功能描述	CMP初始化
先决条件	-
被调用函数	-
输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输入参数{in}	
periph	指定模块实例, CMP0或CMP1
输入参数{in}	
p_init	初始化结构体指针,成员信息参考 表 3-77. 结构体hal_cmp_init_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如:

```
/* CMP initialize */
```

```
hal_cmp_dev_struct cmp_dev_parameter;
```

```
hal_cmp_init_struct cmp_init_parameter;
```

```
hal_cmp_init(&cmp_dev_parameter, CMP0, &cmp_init_parameter);
```

函数 hal_cmp_start

函数hal_cmp_start描述见下表:

表 3-83. 函数 hal_cmp_start

函数名称	hal_cmp_start
函数原型	int32_t hal_cmp_start(hal_cmp_dev_struct *cmp_dev);
功能描述	启动CMP模块功能
先决条件	-
被调用函数	-
输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输出参数{out}	
-	-
返回值	

int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK
---------	---

例如：

```
/* start the CMP module function */

hal_cmp_dev_struct cmp_dev_parameter;

hal_cmp_start(&cmp_dev_parameter);
```

函数 hal_cmp_stop

函数hal_cmp_stop描述见下表：

表 3-84. 函数 hal_cmp_stop

函数名称	hal_cmp_stop
函数原型	int32_t hal_cmp_stop(hal_cmp_dev_struct *cmp_dev);
功能描述	停止CMP模块功能
先决条件	-
被调用函数	-
输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```
/* stop the CMP module function */

hal_cmp_dev_struct cmp_dev_parameter;

hal_cmp_stop(&cmp_dev_parameter);
```

函数 hal_cmp_start_interrupt

函数hal_cmp_start_interrupt描述见下表：

表 3-85. 函数 hal_cmp_start_interrupt

函数名称	hal_cmp_start_interrupt
函数原型	int32_t hal_cmp_start_interrupt(hal_cmp_dev_struct *cmp_dev, hal_cmp_irq_struct *p_irq)
功能描述	启动CMP模块功能并打开中断
先决条件	-
被调用函数	-
输入参数{in}	
cmp_dev	设备信息结构体指针, 成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输入参数{in}	

p_irq	中断回调函数结构体指针，成员信息参考 表 3-79. 结构体hal_cmp_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```
/* start the CMP module function in interrupt mode */

hal_cmp_dev_struct cmp_dev_parameter;

hal_cmp_irq_struct cmp_irq;

void xxx_function(void)
{
    /* user defined content */
}

cmp_irq.output_changed_handle = xxx_function;

hal_cmp_start_interrupt(&cmp_dev_parameter, &cmp_irq);
```

函数 hal_cmp_stop_interrupt

函数hal_cmp_stop_interrupt描述见下表：

表 3-86. 函数 hal_cmp_stop_interrupt

函数名称	hal_cmp_stop_interrupt
函数原型	int32_t hal_cmp_stop_interrupt(hal_cmp_dev_struct *cmp_dev);
功能描述	停止CMP模块功能并关闭相关中断
先决条件	-
被调用函数	-
输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```
/* stop the CMP module function and interrupt */

hal_cmp_dev_struct cmp_dev_parameter;

hal_cmp_stop_interrupt(&cmp_dev_parameter);
```

函数 hal_cmp_irq_handle_set

函数hal_cmp_irq_handle_set描述见下表：

表 3-87. 函数 hal_cmp_irq_handle_set

函数名称	hal_cmp_irq_handle_set
函数原型	void hal_cmp_irq_handle_set(hal_cmp_dev_struct *cmp_dev, hal_cmp_irq_struct *p_irq);
功能描述	中断回调函数设置
先决条件	-
被调用函数	-
输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输入参数{in}	
p_irq	中断回调函数结构体指针, 成员信息参考 表 3-79. 结构体hal_cmp_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the comparator external trigger interrupt callback function */
hal_cmp_dev_struct cmp_dev_parameter;
hal_cmp_irq_struct cmp_irq;
void xxx_function(void)
{
    /* user defined content */
}
cmp_irq.output_changed_handle = xxx_function;
hal_cmp_irq_handle_set(&cmp_dev_parameter, &cmp_irq);
```

函数 hal_cmp_irq_handle_all_reset

函数hal_cmp_irq_handle_all_reset描述见下表：

表 3-88. 函数 hal_cmp_irq_handle_all_reset

函数名称	hal_cmp_irq_handle_all_reset
函数原型	void hal_cmp_irq_handle_all_reset(hal_cmp_dev_struct *cmp_dev);
功能描述	复位中断回调函数
先决条件	-
被调用函数	-

输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset the comparator external trigger interrupt callback function */
```

```
hal_cmp_dev_struct cmp_dev_parameter;
```

```
hal_cmp_irq_handle_all_reset(&cmp_dev_parameter);
```

函数 hal_cmp_irq

函数hal_cmp_irq描述见下表:

表 3-89. 函数 hal_cmp_irq

函数名称	hal_cmp_irq
函数原型	void hal_cmp_irq(hal_cmp_dev_struct *cmp_dev);
功能描述	中断服务函数
先决条件	-
被调用函数	-
输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* cmp external trigger interrupt handler content function */
```

```
hal_cmp_dev_struct cmp_dev_parameter;
```

```
hal_cmp_irq(&cmp_dev_parameter);
```

函数 hal_cmp_lock

函数hal_cmp_lock描述见下表:

表 3-90. 函数 hal_cmp_lock

函数名称	hal_cmp_lock
函数原型	int32_t hal_cmp_lock(hal_cmp_dev_struct *cmp_dev);
功能描述	锁定CMP模块
先决条件	-

被调用函数	-
输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_ALREADY_DONE

例如:

```
/* lock the CMP module */

hal_cmp_dev_struct cmp_dev_parameter;

hal_cmp_lock(&cmp_dev_parameter);
```

函数 hal_cmp_output_level_get

函数hal_cmp_output_level_get描述见下表:

表 3-91. 函数 hal_cmp_output_level_get

函数名称	hal_cmp_output_level_get
函数原型	uint32_t hal_cmp_output_level_get(hal_cmp_dev_struct *cmp_dev);
功能描述	获取CMP输出电平
先决条件	-
被调用函数	-
输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输出参数{out}	
-	-
返回值	
uint32_t	输出电平

例如:

```
/* get output level of comparator */

hal_cmp_dev_struct cmp_dev_parameter;

uint32_t output_level;

output_level = hal_cmp_output_level_get (&cmp_dev_parameter);
```

函数 hal_cmp_state_get

函数hal_cmp_state_get描述见下表:

表 3-92. 函数 hal_cmp_state_get

函数名称	hal_cmp_state_get
------	-------------------

函数原型	hal_cmp_state_enum hal_cmp_state_get(hal_cmp_dev_struct *cmp_dev);
功能描述	获取CMP状态信息
先决条件	-
被调用函数	-
输入参数{in}	
cmp_dev	设备信息结构体指针,成员信息参考 表 3-78. 结构体hal_cmp_dev_struct
输出参数{out}	
-	-
返回值	
hal_cmp_state_enum	CMP设备状态信息

例如:

```
/* get the state of CMP device */

hal_cmp_dev_struct cmp_dev_parameter;

hal_cmp_state_enum cmp_state;

cmp_state = hal_cmp_state_get(&cmp_dev_parameter);
```

3.5. CRC

循环冗余校验码是一种用在数字网络和存储设备上的差错校验码,可以校验原始数据的偶然误差。章节[3.5.1](#)描述了CRC的寄存器列表,章节[3.5.2](#)对CRC库函数进行说明。

3.5.1. 外设寄存器说明

CRC寄存器列表如下表所示:

表 3-93. CRC 寄存器

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

3.5.2. 外设库函数说明

CRC库函数列表如下表所示:

表 3-94. CRC 库函数

库函数名称	库函数描述
hal_crc_struct_init	使用默认值初始化CRC结构体
hal_crc_init	初始化CRC
hal_crc_deinit	复位CRC计算单元
hal_crc_single_data_calculate	计算单个数据的CRC值
hal_crc_block_data_calculate	计算一组数据的CRC值

枚举 hal_crc_state_enum

表 3-95. 枚举 hal_crc_state_enum

成员名称	功能描述
HAL_CRC_STATE_NONE	无（默认值）
HAL_CRC_STATE_RESET	CRC未被初始化或停止
HAL_CRC_STATE_BUSY	CRC繁忙
HAL_CRC_STATE_TIMEOUT	CRC产生超时
HAL_CRC_STATE_ERROR	CRC出错
HAL_CRC_STATE_READY	CRC准备

枚举 hal_crc_struct_type_enum

表 3-96. 枚举 hal_crc_struct_type_enum

成员名称	功能描述
HAL_CRC_INIT_STRUCT	CRC初始化结构体
HAL_CRC_DEV_STRUCT	CRC设备信息结构体

结构体 hal_crc_dev_struct

表 3-97. 结构体 hal_crc_dev_struct

成员名称	功能描述
state	CRC状态
mutex	互斥锁和解锁状态
*priv	隐私数据

结构体 hal_crc_init_struct

表 3-98. 结构体 hal_crc_init_struct

成员名称	功能描述
polynomial_value	多项式
init_data_value	CRC初始化值
input_data_reverse_mode	输入数据翻转模式
output_data_reverse	输出数据翻转
polynomial_size	多项式大小

函数 hal_crc_struct_init

函数hal_crc_struct_init描述见下表：

表 3-99. 函数 hal_crc_struct_init

函数名称	hal_crc_struct_init
函数原型	int32_t hal_crc_struct_init(hal_crc_struct_type_enum hal_struct_type, void *p_struct);
功能描述	初始化CRC结构体
先决条件	-
被调用函数	-
输入参数{in}	
hal_struct_type	结构体类型
HAL_CRC_INIT_STRUCTURE	CRC初始化结构体
HAL_CRC_DEV_STRUCTURE	CRC设备信息结构体
输入参数{in}	
p_struct	指向包含配置信息的CRC结构体的指针
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* initialize the CRC structure with the default values */
```

```
hal_crc_dev_struct crc_info;
```

```
hal_crc_struct_init(HAL_CRC_DEV_STRUCTURE, &crc_info);
```

函数 hal_crc_init

函数hal_crc_init描述见下表：

表 3-100. 函数 hal_crc_init

函数名称	hal_crc_init
函数原型	int32_t hal_crc_init(hal_crc_dev_struct *crc_dev, hal_crc_init_struct *p_crc_init);
功能描述	初始化CRC外设
先决条件	-
被调用函数	-
输入参数{in}	
crc_dev	指向CRC设备信息结构体的指针，结构体成员参考 表3-97. 结构体 hal_crc_dev_struct 。

输入参数{in}	
p_crc_init	指向CRC初始化结构体的指针，结构体成员参考 表3-98. 结构体 hal_crc_init_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* initialize CRC */
hal_crc_init_struct crc_init_parameter;
hal_crc_dev_struct crc_info;
hal_crc_struct_init(HAL_CRC_INIT_STRUCT, &crc_init_parameter);
hal_crc_struct_init(HAL_CRC_DEV_STRUCT, &crc_info);
crc_init_parameter.polynomial_value = 0x04C11DB7;
crc_init_parameter.init_data_value = 0xFFFFFFFF;
crc_init_parameter.input_data_reverse_mode = CRC_INPUT_REVERSE_NONE;
crc_init_parameter.output_data_reverse = CRC_OUTPUT_REVERSE_DISABLE;
crc_init_parameter.polynomial_size = CRC_POLYNOMIAL_SIZE_32BIT;
hal_crc_init(&crc_info,&crc_init_parameter);
```

函数 hal_crc_deinit

函数hal_crc_deinit描述见下表:

表 3-101. 函数 hal_crc_deinit

函数名称	hal_crc_deinit
函数原型	int32_t hal_crc_deinit(hal_crc_dev_struct *crc_dev);
功能描述	复位CRC外设
先决条件	-
被调用函数	-
输入参数{in}	
crc_dev	指向CRC设备信息结构体的指针，结构体成员参考 表3-97. 结构体 hal_crc_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* deinitialize CRC */
hal_crc_dev_struct crc_info;
hal_crc_struct_init(HAL_CRC_DEV_STRUCT, &crc_info);
hal_crc_deinit(&crc_info);
```


函数 hal_crc_single_data_calculate

函数hal_crc_single_data_calculate描述见下表：

表 3-102. 函数 hal_crc_single_data_calculate

函数名称	hal_crc_single_data_calculate
函数原型	uint32_t hal_crc_single_data_calculate(hal_crc_dev_struct *crc_dev, uint32_t sdata, uint8_t data_format);
功能描述	计算单个数据的CRC值
先决条件	-
被调用函数	
输入参数{in}	
crc_dev	指向CRC设备信息结构体的指针，结构体成员参考 表3-97. 结构体 hal_crc_dev_struct 。
输入参数{in}	
sdata	输入数据
输入参数{in}	
data_format	输入数据的格式
INPUT_FORMAT_WORD	字
INPUT_FORMAT_HALFWORD	半字
INPUT_FORMAT_BYTE	字节
输出参数{out}	
-	-
返回值	
int32_t	CRC计算结果

例如：

```
/* CRC calculate a 32-bit data */
uint32_t valcrc = 0;
uint32_t val_32 = 0xabcd1234;
hal_crc_dev_struct crc_info;
hal_crc_struct_init(HAL_CRC_DEV_STRUCT, &crc_info);
valcrc = hal_crc_single_data_calculate(&crc_info, val_32, INPUT_FORMAT_WORD);
```

函数 hal_crc_block_data_calculate

函数hal_crc_block_data_calculate描述见下表：

表 3-103. 函数 hal_crc_block_data_calculate

函数名称	hal_crc_block_data_calculate
函数原型	uint32_t hal_crc_block_data_calculate(hal_crc_dev_struct *crc_dev, void

	*array, uint32_t size, uint8_t data_format)
功能描述	计算一组数据的CRC值
先决条件	-
被调用函数	
输入参数{in}	
crc_dev	指向CRC设备信息结构体的指针，结构体成员参考 表3-97. 结构体 hal_crc_dev_struct 。
输入参数{in}	
array	输入数组
输入参数{in}	
size	数组的大小
输入参数{in}	
data_format	输入数据的格式
INPUT_FORMAT_WORD	字
INPUT_FORMAT_HALFWORD	半字
INPUT_FORMAT_BYTE	字节
输出参数{out}	
-	-
返回值	
int32_t	CRC计算结果

例如:

```
/*calculate the 32-bit CRC value of a 32-bit data array */
uint32_t valcrc = 0;
uint32_t val_32[4] = {0xabcd1234, 0x3456cdef, 0x789a1524, 0x12346792};
hal_crc_dev_struct crc_info;
hal_crc_struct_init(HAL_CRC_DEV_STRUCT, &crc_info);
valcrc = hal_crc_block_data_calculate(&crc_info, val_32, val_size,
INPUT_FORMAT_WORD);
```

3.6. CTC

CTC模块基于外部高精度的参考信号源来校准IRC48M的时钟频率，通过自动的或手动的调整校准值，以得到一个精准的IRC48M时钟。章节[3.6.1](#)描述了CTC的寄存器列表，章节[3.6.2](#)对CTC库函数进行说明。

3.6.1. 外设寄存器描述

CTC寄存器列表如下表所示:

表 3-104. CTC 寄存器

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

3.6.2. 外设库函数说明

CTC库函数列表如下表所示：

表 3-105. CTC 库函数

库函数名称	库函数描述
hal_ctc_deinit	复位CTC单元
hal_ctc_init	CTC外设结构体配置
hal_ctc_irc48m_trim_value_config	配置IRC48M时钟校准值
hal_ctc_struct_init	初始化CTC外设结构体
hal_ctc_start	启动CTC外设
hal_ctc_stop	停止CTC外设
hal_ctc_irq	CTC中断处理函数
hal_ctc_irq_handle_set	设置CTC中断回调函数
hal_ctc_irq_handle_all_reset	清除CTC中断回调函数
hal_ctc_start_interrupt	中断模式启动CTC
hal_ctc_stop_interrupt	关闭中断模式启动CTC的模式

枚举 hal_ctc_struct_type_enum

表 3-106. 枚举 hal_ctc_struct_type_enum

成员名称	功能描述
HAL_CTC_INIT_STRUCT	CTC初始化结构体
HAL_CTC_IRQ_STRUCT	CTC中断回调函数指针结构体
HAL_CTC_DEV_STRUCT	CTC设备信息结构体

枚举 hal_ctc_error_enum

表 3-107. 枚举 hal_ctc_error_enum

成员名称	功能描述
HAL_CTC_ERROR_NONE	无错误
HAL_CTC_ERROR_SYSTEM	CTC内部错误，如果时钟问题，使能、禁能错误状态
HAL_CTC_ERROR_CONFIG	产生配置错误

枚举 `hal_ctc_state_enum`

表 3-108. 枚举 `hal_ctc_state_enum`

成员名称	功能描述
<code>HAL_CTC_STATE_NONE</code>	无（默认值）
<code>HAL_CTC_STATE_RESET</code>	CTC未被初始化或停止
<code>HAL_CTC_STATE_BUSY</code>	CTC繁忙
<code>HAL_CTC_STATE_TIMEOUT</code>	CTC产生超时
<code>HAL_CTC_STATE_ERROR</code>	CTC出错
<code>HAL_CTC_STATE_READY</code>	CTC准备

结构体 `hal_ctc_irq_struct`

表 3-109. 结构体 `hal_ctc_irq_struct`

成员名称	功能描述
<code>ctc_ckok_handle</code>	时钟校准成功中断处理回调函数
<code>ctc_ckwarn_handle</code>	时钟校准警告中断处理回调函数
<code>ctc_err_handle</code>	错误中断处理回调函数
<code>ctc_eref_handle</code>	期望参考中断处理回调函数

结构体 `hal_ctc_init_struct`

表 3-110. 结构体 `hal_ctc_init_struct`

成员名称	功能描述
<code>cal_style</code>	计算方式
<code>source</code>	参考信号来源
<code>polarity</code>	参考信号源极性
<code>prescaler</code>	参考信号源预分频
<code>limit_value</code>	时钟校准时基限值
<code>reload_value</code>	CTC 计数器重载值
<code>frequency</code>	参考信号时钟频率

结构体 `hal_ctc_dev_struct`

表 3-111. 结构体 `hal_ctc_dev_struct`

成员名称	功能描述
<code>ctc_irq</code>	CTC中断回调函数指针结构体
<code>error_state</code>	CTC错误信息
<code>state</code>	CTC信息
<code>mutex</code>	互斥锁和解锁状态
<code>*priv</code>	隐私数据

函数 hal_ctc_deinit

函数hal_ctc_deinit描述见下表：

表 3-112. 函数 hal_ctc_deinit

函数名称	hal_ctc_deinit
函数原形	int32_t hal_ctc_deinit(hal_ctc_dev_struct *ctc_dev)
功能描述	复位CTC外设
先决条件	-
被调用函数	hal_rcu_periph_reset_enable / hal_rcu_periph_reset_disable
输入参数{in}	
ctc_dev	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 hal_ctc_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* reset CTC peripheral */
hal_ctc_dev_struct ctc_info;
hal_ctc_deinit(&ctc_info);
```

函数 hal_ctc_init

函数hal_ctc_init描述见下表：

表 3-113. 函数 hal_ctc_init

函数名称	hal_ctc_init
函数原形	int32_t hal_ctc_init(hal_ctc_dev_struct *ctc_dev, hal_ctc_init_struct *ctc_init)
功能描述	CTC外设结构体配置
先决条件	-
被调用函数	-
输入参数{in}	
ctc_dev	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 hal_ctc_dev_struct
ctc_init	指向hal_ctc_init_struct结构体的指针，结构体成员参考 表3-110. 结构体 hal_ctc_init_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* initialize CTC init structure */

hal_ctc_dev_struct ctc_info;

hal_ctc_init_struct ctc_init_parameter;

hal_ctc_struct_init(HAL_CTC_INIT_STRUCT, &ctc_init_parameter);

ctc_init_parameter.source = CTC_REFSOURCE_LXTAL;

ctc_init_parameter.cal_style = CTC_VALUE_AUTO_CAL;

ctc_init_parameter.polarity = CTC_REFSOURCE_POLARITY_FALLING;

ctc_init_parameter.prescaler = CTC_REFSOURCE_PSC_OFF;

hal_ctc_init(&ctc_info,&ctc_init_parameter);
```

函数 hal_ctc_irc48m_trim_value_config

函数hal_ctc_irc48m_trim_value_config描述见下表:

表 3-114. 函数 hal_ctc_irc48m_trim_value_config

函数名称	hal_ctc_irc48m_trim_value_config
函数原形	void hal_ctc_irc48m_trim_value_config(hal_ctc_dev_struct *ctc_dev, uint8_t trim_value)
功能描述	配置IRC48M时钟校准值
先决条件	-
被调用函数	-
输入参数{in}	
ctc_dev	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 hal_ctc_dev_struct
trim_value	6bit校准值(0x0-0x3F)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the IRC48M trim value */

hal_ctc_dev_struct ctc_info;

hal_ctc_irc48m_trim_value_config (&ctc_info, 0x30);
```

函数 hal_ctc_struct_init

函数hal_ctc_struct_init描述见下表:

表 3-115. 函数 hal_ctc_struct_init

函数名称	hal_ctc_struct_init
函数原形	void hal_ctc_struct_init(hal_ctc_struct_type_enum hal_struct_type, void *p_struct)
功能描述	初始化CTC外设结构体
先决条件	-
被调用函数	-
输入参数{in}	
hal_struct_type	指向hal_ctc_struct_type_enum的指针，枚举成员参考 表3-106. 枚举 hal_ctc_struct_type_enum
输出参数{out}	
p_struct	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 hal_ctc_dev_struct
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* initialize CTC init structure */
```

函数 hal_ctc_start

函数hal_ctc_start描述见下表：

表 3-116. 函数 hal_ctc_start

函数名称	hal_ctc_start
函数原形	int32_t hal_ctc_start(hal_ctc_dev_struct *ctc_dev)
功能描述	启动CTC外设
先决条件	-
被调用函数	-
输入参数{in}	
ctc_dev	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 hal_ctc_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* start CTC module function */
```

```
hal_ctc_dev_struct ctc_info;
```

```
hal_ctc_start(&ctc_info);
```

函数 hal_ctc_stop

函数hal_ctc_stop描述见下表：

表 3-117. 函数 hal_ctc_stop

函数名称	hal_ctc_stop
函数原形	int32_t hal_ctc_stop(hal_ctc_dev_struct *ctc_dev)
功能描述	停止CTC外设
先决条件	-
被调用函数	-
输入参数{in}	
ctc_dev	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 hal_ctc_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* stop CTC module function */
hal_ctc_dev_struct ctc_info;
hal_ctc_stop(&ctc_info);
```

函数 hal_ctc_irq

函数hal_ctc_irq描述见下表：

表 3-118. 函数 hal_ctc_irq

函数名称	hal_ctc_irq
函数原形	void hal_ctc_irq(hal_ctc_dev_struct *ctc_dev)
功能描述	CTC中断处理函数
先决条件	-
被调用函数	-
输入参数{in}	
ctc_dev	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 hal_ctc_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CTC interrupt handler content function */
```



```
hal_ctc_dev_struct ctc_info;
```

```
hal_ctc_irq(&ctc_info);
```

函数 hal_ctc_irq_handle_set

函数hal_ctc_irq_handle_set描述见下表：

表 3-119. 函数 hal_ctc_irq_handle_set

函数名称	hal_ctc_irq_handle_set
函数原形	void hal_ctc_irq_handle_set(hal_ctc_dev_struct *ctc_dev, hal_ctc_irq_struct *p_irq)
功能描述	CTC中断处理函数
先决条件	-
被调用函数	-
输入参数{in}	
ctc_dev	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 hal_ctc_dev_struct
p_irq	指向CTC中断回调函数结构体的指针，结构体成员参考 表3-109. 结构体 hal_ctc_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set user-defined interrupt callback function */
```

```
hal_ctc_dev_struct ctc_info;
```

```
hal_ctc_irq_struct ctc_irq;
```

```
void ctc_irq_test(void *p);
```

```
ctc_irq.ctc_ckok_handle = ctc_irq_test;
```

```
hal_ctc_irq_handle_set(&ctc_info, &ctc_irq);
```

函数 hal_ctc_irq_handle_all_reset

函数hal_ctc_irq_handle_all_reset描述见下表：

表 3-120. 函数 hal_ctc_irq_handle_all_reset

函数名称	hal_ctc_irq_handle_all_reset
函数原形	void hal_ctc_irq_handle_all_reset(hal_ctc_dev_struct *ctc_dev)
功能描述	清除CTC中断回调函数
先决条件	-
被调用函数	-

输入参数{in}	
ctc_dev	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 <i>hal_ctc_dev_struct</i>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset user-defined interrupt callback function */
```

```
hal_ctc_dev_struct ctc_info;
```

```
hal_ctc_irq_handle_reset(&ctc_info);
```

函数 **hal_ctc_start_interrupt**

函数hal_ctc_start_interrupt描述见下表：

表 3-121. 函数 hal_ctc_start_interrupt

函数名称	hal_ctc_start_interrupt
函数原形	int32_t hal_ctc_start_interrupt(hal_ctc_dev_struct *ctc_dev, hal_ctc_irq_struct *p_irq)
功能描述	中断模式启动CTC
先决条件	-
被调用函数	-
输入参数{in}	
ctc_dev	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 <i>hal_ctc_dev_struct</i>
p_irq	指向CTC中断回调函数结构体的指针，结构体成员参考 表3-109. 结构体 <i>hal_ctc_irq_struct</i>
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* start CTC with interrupt */
```

```
hal_ctc_dev_struct ctc_info;
```

```
hal_ctc_irq_struct ctc_irq;
```

```
void ctc_irq_test(void *p);
```

```
ctc_irq.ctc_ckok_handle = ctc_irq_test;
```

```
hal_ctc_start_interrupt(&ctc_info, &ctc_irq);
```

函数 hal_ctc_stop_interrupt

函数hal_ctc_stop_interrupt描述见下表:

表 3-122. 函数 hal_ctc_stop_interrupt

函数名称	hal_ctc_stop_interrupt
函数原形	int32_t hal_ctc_stop_interrupt(hal_ctc_dev_struct *ctc_dev)
功能描述	停止中断模式CTC
先决条件	-
被调用函数	-
输入参数{in}	
ctc_dev	指向CTC设备信息结构体的指针，结构体成员参考 表3-111. 结构体 hal_ctc_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* stop CTC with interrupt */
hal_ctc_dev_struct ctc_info;
hal_ctc_stop_interrupt(&ctc_info);
```

3.7. DAC

数字/模拟转换器可以将12位的数字数据转换为外部引脚上的电压输出，章节[3.7.1](#)描述了DAC的寄存器列表，章节[3.7.2](#)对DAC库函数进行说明。GD32F330和GD32F310没有DAC外设。

3.7.1. 外设寄存器说明

DAC寄存器列表如下表所示:

表 3-123. DAC 寄存器

寄存器名称	寄存器描述
DAC_CTL	DAC控制寄存器
DAC_SWT	DAC软件触发寄存器
DAC_R12DH	DAC 12位右对齐数据保持寄存器
DAC_L12DH	DAC 12位左对齐数据保持寄存器
DAC_R8DH	DAC 8位右对齐数据保持寄存器
DAC_DO	DAC数据输出寄存器
DAC_STAT	DAC状态寄存器

3.7.2. 外设库函数说明

DAC库函数列表如下表所示：

表 3-124. DAC 库函数

库函数名称	库函数描述
hal_dac_init	初始化DAC
hal_dac_deinit	复位DAC外设
hal_dac_struct_init	初始化DAC结构体
hal_dac_start	启动DAC
hal_dac_stop	停止DAC
hal_dac_start_interrupt	DAC DMA下溢中断启动
hal_dac_stop_interrupt	DAC DMA下溢中断停止
hal_dac_start_dma	DAC DMA启动
hal_dac_stop_dma	DAC DMA停止
hal_dac_irq	DAC中断处理函数
hal_dac_irq_handle_set	设置DAC中断回调函数并使能DAC中断
hal_dac_irq_handle_all_reset	清除DAC中断回调函数

枚举 hal_dac_state_enum

表 3-125. 枚举 hal_dac_state_enum

成员名称	功能描述
DAC_STATE_NONE	无（默认值）
DAC_STATE_RESET	DAC未被初始化或停止
DAC_STATE_BUSY	DAC繁忙
DAC_STATE_TIMEOUT	DAC产生超时
DAC_STATE_ERROR	DAC出错
DAC_STATE_READY	DAC准备

枚举 hal_dac_struct_type_enum

表 3-126. 枚举 hal_dac_struct_type_enum

成员名称	功能描述
HAL_DAC_INIT_STRUCT	DAC初始化结构体
HAL_DAC_IRQ_STRUCT	DAC中断回调函数指针结构体
HAL_DAC_DMA_HANDLE_CB_STRUCT	DAC DMA回调函数指针结构体
HAL_DAC_DEV_STRUCT	DAC设备信息结构体

枚举 `hal_dac_error_enum`

表 3-127. 枚举 `hal_dac_error_enum`

成员名称	功能描述
<code>DAC_ERROR_NONE</code>	无错误
<code>DAC_ERROR_SYSTEM</code>	DAC内部错误，如果时钟问题，使能、禁能错误状态
<code>DAC_ERROR_DMA_UNDERFLOW</code>	DAC DMA下溢错误
<code>DAC_ERROR_CONFIG</code>	产生配置错误

结构体 `hal_dac_irq_struct`

表 3-128. 结构体 `hal_dac_irq_struct`

成员名称	功能描述
<code>dac_underflow_handle</code>	DAC DMA下溢中断处理回调函数

结构体 `hal_dac_dma_handle_cb_struct`

表 3-129. 结构体 `hal_dac_dma_handle_cb_struct`

成员名称	功能描述
<code>full_transcom_handle</code>	DAC DMA全传输完成中断处理回调函数
<code>half_transcom_handle</code>	DAC DMA半传输完成中断处理回调函数
<code>error_handle</code>	DAC DMA错误中断处理回调函数

结构体 `hal_dac_dev_struct`

表 3-130. 结构体 `hal_dac_dev_struct`

成员名称	功能描述
<code>dac_irq</code>	DAC中断回调函数指针结构体
<code>*p_dma_dac</code>	DMA设备信息结构体指针
<code>dac_dma</code>	DAC DMA回调函数指针结构体
<code>error_state</code>	DAC错误信息
<code>state</code>	DAC信息
<code>mutex</code>	互斥锁和解锁状态
<code>*priv</code>	隐私数据

结构体 `hal_dac_init_struct`

表 3-131. 结构体 `hal_dac_init_struct`

成员名称	功能描述
<code>output_waveform_selection</code>	输出波形选择
<code>noise_wave_bit_width</code>	噪声位宽选择
<code>output_buffer_enable</code>	输出缓冲区使能
<code>trigger_enable</code>	触发使能
<code>trigger_selection</code>	触发选择

成员名称	功能描述
aligned_mode	对齐模式
output_value	输出数据

函数 hal_dac_init

函数hal_dac_init描述见下表：

表 3-132. 函数 hal_dac_init

函数名称	hal_dac_init
函数原型	int32_t hal_dac_init(hal_dac_dev_struct *dac_dev, hal_dac_init_struct *dac);
功能描述	初始化DAC外设
先决条件	-
被调用函数	-
输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 hal_dac_dev_struct 。
输入参数{in}	
dac	指向DAC初始化结构体的指针，结构体成员参考 表3-131. 结构体 hal_dac_init_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```

/* initialize DAC */

hal_dac_init_struct dac_init_parameter;

hal_dac_dev_struct dac_info;

dac_init_parameter.output_waveform_selection = DAC_WAVE_DISABLE;

dac_init_parameter.trigger_enable = ENABLE;

dac_init_parameter.trigger_selection = DAC_TRIGGER_EXTI_9;

dac_init_parameter.aligned_mode = DAC_R12_ALIGNED_MODE;

dac_init_parameter.output_buffer_enable = ENABLE;

dac_init_parameter.output_value = 0;

hal_dac_init(&dac_info,&dac_init_parameter);

```

函数 hal_dac_deinit

函数hal_dac_deinit描述见下表：

表 3-133. 函数 hal_dac_deinit

函数名称	hal_dac_deinit
函数原型	int32_t hal_dac_deinit(hal_dac_dev_struct *dac_dev);
功能描述	复位DAC外设
先决条件	-
被调用函数	hal_rcu_periph_reset_enable / hal_rcu_periph_reset_disable
输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 hal_dac_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* deinitialize DAC */
hal_dac_dev_struct dac_info;
hal_dac_deinit(&dac_info);
```

函数 hal_dac_struct_init

函数hal_dac_struct_init描述见下表:

表 3-134. 函数 hal_dac_struct_init

函数名称	hal_dac_struct_init
函数原型	void hal_dac_struct_init(hal_dac_struct_type_enum hal_struct_type, void *p_struct);
功能描述	初始化DAC结构体
先决条件	-
被调用函数	-
输入参数{in}	
hal_struct_type	结构体类型
HAL_DAC_INIT_STRUCTURE	DAC初始化结构体
HAL_DAC_IRQ_STRUCTURE	DAC中断回调函数指针结构体
HAL_DAC_DMA_HANDLER_CALLBACK_STRUCTURE	DAC DMA回调函数指针结构体
HAL_DAC_DEV_STRUCTURE	DAC设备信息结构体
输入参数{in}	
p_struct	指向包含配置信息的DAC结构体的指针

输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* initialize the DAC structure with the default values */
hal_dac_dev_struct dac_info;
hal_dac_struct_init(HAL_DAC_DEV_STRUCTURE, &dac_info);
```

函数 hal_dac_start

函数hal_dac_start描述见下表:

表 3-135. 函数 hal_dac_start

函数名称	hal_dac_start
函数原型	int32_t hal_dac_start(hal_dac_dev_struct *dac_dev);
功能描述	启动DAC
先决条件	-
被调用函数	-
输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 hal_dac_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* start DAC */
hal_dac_dev_struct dac_info;
hal_dac_start(&dac_info);
```

函数 hal_dac_stop

函数hal_dac_stop描述见下表:

表 3-136. 函数 hal_dac_stop

函数名称	hal_dac_stop
函数原型	int32_t hal_dac_stop(hal_dac_dev_struct *dac_dev);
功能描述	停止DAC
先决条件	-
被调用函数	-

输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 <i>hal_dac_dev_struct</i> 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* stop DAC */
hal_dac_dev_struct dac_info;
hal_dac_stop(&dac_info);
```

函数 hal_dac_start_interrupt

函数hal_dac_start_interrupt描述见下表:

表 3-137. 函数 hal_dac_start_interrupt

函数名称	hal_dac_start_interrupt
函数原型	int32_t hal_dac_start_interrupt(hal_dac_dev_struct *dac_dev, hal_dac_irq_struct *p_irq);
功能描述	DAC DMA下溢中断启动
先决条件	-
被调用函数	-
输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 <i>hal_dac_dev_struct</i> 。
输入参数{in}	
p_irq	指向DAC中断回调函数结构体的指针，结构体成员参考 表3-128. 结构体 <i>hal_dac_irq_struct</i> 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* start DAC under error interrupt */
uint8_t convertarr[10] = {0x00, 0x33, 0x66, 0x99, 0xCC, 0xFF, 0xCC, 0x99, 0x66, 0x33};
hal_dac_init_struct dac_init_parameter;
hal_dac_dev_struct dac_info;
hal_dac_irq_struct dac_irq;
```

```
hal_dac_start_dma(&dac_info,(uint32_t*)convertarr,10,DAC_R8_ALIGNED_MODE,&dac_d
ma_handle_cb);
```

```
dac_irq.dac_underflow_handle = dac_underflow_func;
```

```
hal_dac_irq_handle_set(&dac_info,&dac_irq);
```

```
hal_dac_start_interrupt(&dac_info,&dac_irq);
```

函数 hal_dac_stop_interrupt

函数hal_dac_stop_interrupt描述见下表：

表 3-138. 函数 hal_dac_stop_interrupt

函数名称	hal_dac_stop_interrupt
函数原型	int32_t hal_dac_stop_interrupt(hal_dac_dev_struct *dac_dev);
功能描述	DAC DMA下溢中断停止
先决条件	-
被调用函数	-
输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 hal_dac_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* stop DAC under error interrupt */
```

```
uint8_t convertarr[10] = {0x00, 0x33, 0x66, 0x99, 0xCC, 0xFF, 0xCC, 0x99, 0x66, 0x33};
```

```
hal_dac_init_struct dac_init_parameter;
```

```
hal_dac_dev_struct dac_info;
```

```
hal_dac_start_dma(&dac_info,(uint32_t*)convertarr,10,DAC_R8_ALIGNED_MODE,&dac_d
ma_handle_cb);
```

```
dac_irq.dac_underflow_handle = dac_underflow_func;
```

```
hal_dac_irq_handle_set(&dac_info,&dac_irq);
```

```
hal_dac_stop_interrupt(&dac_info);
```

函数 hal_dac_start_dma

函数hal_dac_start_dma描述见下表：

表 3-139. 函数 hal_dac_start_dma

函数名称	hal_dac_start_dma
函数原型	int32_t hal_dac_start_dma(hal_dac_dev_struct *dac_dev, uint32_t* pdata, uint32_t length, uint32_t aligned_mode, hal_dac_dma_handle_cb_struct *dmacb);
功能描述	DAC DMA启动
先决条件	-
被调用函数	-
输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 hal_dac_dev_struct 。
输入参数{in}	
p_data	指向数据存储区的指针
输入参数{in}	
length	传输数据长度
输入参数{in}	
aligned_mode	选择对齐方式
DAC_R12_ALIGNED_MODE	12位右对齐
DAC_L12_ALIGNED_MODE	12位左对齐
DAC_R8_ALIGNED_MODE	8位右对齐
输入参数{in}	
dmacb	用户回调函数结构体的指针，结构体成员参考 表3-129. 结构体 hal_dac_dma_handle_cb_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* start DAC with DMA */
```

```
uint8_t convertarr[10] = {0x00, 0x33, 0x66, 0x99, 0xCC, 0xFF, 0xCC, 0x99, 0x66, 0x33};
```

```
hal_dac_init_struct dac_init_parameter;
```

```
hal_dac_dev_struct dac_info;
```

```
hal_dac_start_dma(&dac_info, (uint32_t*)convertarr, 10, DAC_R8_ALIGNED_MODE, &dac_dma_handle_cb);
```

```
dac_irq.dac_underflow_handle = dac_underflow_func;
```

```
hal_dac_start_dma(&dac_info, (uint32_t*)convertarr, 10, DAC_R8_ALIGNED_MODE, &dac_d
```

```
ma_handle_cb);
```

```
hal_dac_start(&dac_info);
```

函数 hal_dac_stop_dma

函数hal_dac_stop_dma描述见下表：

表 3-140. 函数 hal_dac_stop_dma

函数名称	hal_dac_stop_dma
函数原型	int32_t hal_dac_stop_dma(hal_dac_dev_struct *dac_dev);
功能描述	DAC DMA停止
先决条件	-
被调用函数	-
输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 hal_dac_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* stop DAC with DMA */
```

```
hal_dac_dev_struct dac_info;
```

```
hal_dac_stop_dma(&dac_info);
```

函数 hal_dac_irq

函数hal_dac_irq描述见下表：

表 3-141. 函数 hal_dac_irq

函数名称	hal_dac_irq
函数原型	int32_t hal_dac_irq(hal_dac_dev_struct *dac_dev);
功能描述	DAC中断处理函数
先决条件	-
被调用函数	-
输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 hal_dac_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* the function is used in the relative interrupt routine */

hal_dac_dev_struct dac_info;

TIMER5_DAC_IRQHandler(void)

{

    hal_dac_irq(&dac_info);

}
```

函数 hal_dac_irq_handle_set

函数hal_dac_irq_handle_set描述见下表:

表 3-142. 函数 hal_dac_irq_handle_set

函数名称	hal_dac_irq_handle_set
函数原型	int32_t hal_dac_irq_handle_set(hal_dac_dev_struct *dac_dev, hal_dac_irq_struct *p_irq);
功能描述	设置DAC中断回调函数并使能DAC中断
先决条件	-
被调用函数	-
输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 hal_dac_dev_struct 。
输入参数{in}	
p_irq	指向DAC中断回调函数结构体的指针，结构体成员参考 表3-128. 结构体 hal_dac_irq_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* set user-defined interrupt callback function */

hal_dac_dev_struct dac_info;

hal_dac_irq_struct dac_irq;

dac_irq.dac_underflow_handle = dac_underflow_func;

hal_dac_irq_handle_set(&dac_info,&dac_irq);
```

函数 hal_dac_irq_handle_all_reset

函数hal_dac_irq_handle_all_reset描述见下表:

表 3-143. 函数 hal_dac_irq_handle_all_reset

函数名称	hal_dac_irq_handle_all_reset
函数原型	int32_t hal_dac_irq_handle_all_reset(hal_dac_dev_struct *dac_dev);
功能描述	清除DAC中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	
dac_dev	指向DAC设备信息结构体的指针，结构体成员参考 表3-130. 结构体 hal_dac_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* reset all user-defined interrupt callback function */
hal_dac_dev_struct dac_info;
hal_dac_irq_handle_all_reset(&dac_info);
```

3.8. SYS

SYS包含DBG和Basetick模块，通过SYS可以配置系统Debug方式，以及配置系统计时器源，为系统提供精度为1ms的计时器，章节[3.8.1](#)描述了DBG和Systick的寄存器列表，章节[3.8.2](#)对SYS库函数进行说明。

3.8.1. 外设寄存器说明

SYS DBG & Systick寄存器列表如下表所示：

表 3-144. SYS DBG & Systick 寄存器

寄存器名称	寄存器描述
SYS_DBG_ID	DBG ID寄存器
SYS_DBG_CTL0	DBG控制寄存器0
SYS_DBG_CTL1	DBG控制寄存器1
SYS_SYSTICK_CTRL	Systick状态与控制寄存器
SYS_SYSTICK_LOAD	Systick重载值寄存器
SYS_SYSTICK_VAL	Systick当前值寄存器
SYS_SYSTICK_CALIB	Systick标定寄存器

3.8.2. 外设库函数说明

SYS库函数列表如下表所示：

表 3-145. SYS 库函数

库函数名称	库函数描述
hal_sys_deinit	复位SYS(DBG&Basetick)模块
hal_sys_debug_init	初始化DBG外设
hal_sys_timesource_init	初始化系统计时器源
hal_sys_basetick_count_get	获取当前系统计时器值
hal_sys_basetick_timeout_check	检测系统计时器是否超时
hal_sys_basetick_delay_ms	以ms为单位设置系统延时
hal_sys_basetick_suspend	暂停系统计时器
hal_sys_basetick_resume	恢复系统计时器
hal_sys_basetick_irq	系统计时器中断服务函数
hal_sys_basetick_irq_handle_set	设置用户自定义中断服务函数
hal_sys_basetick_irq_handle_all_reset	复位用户自定义中断服务函数

枚举 hal_sys_debug_cfg_enum

表 3-146. 枚举 hal_sys_debug_cfg_enum

成员名称	功能描述
SYS_DEBUG_DISABLE	失能调试
SYS_DEBUG_SERIAL_WIRE	串行调试

枚举 hal_sys_timebase_source_enum

表 3-147. 枚举 hal_sys_timebase_source_enum

成员名称	功能描述
SYS_TIMEBASE_SOURCE_SYSTICK	设置Systick为系统计时器源
SYS_TIMEBASE_SOURCE_TIMER0	设置Timer0为系统计时器源
SYS_TIMEBASE_SOURCE_TIMER1	设置Timer1为系统计时器源
SYS_TIMEBASE_SOURCE_TIMER2	设置Timer2为系统计时器源
SYS_TIMEBASE_SOURCE_TIMER5	设置Timer5为系统计时器源
SYS_TIMEBASE_SOURCE_TIMER13	设置Timer13为系统计时器源
SYS_TIMEBASE_SOURCE_TIMER14	设置Timer14为系统计时器源
SYS_TIMEBASE_SOURCE_TIMER15	设置Timer15为系统计时器源
SYS_TIMEBASE_SOURCE_TIMER16	设置Timer16为系统计时器源

函数 hal_sys_deinit

函数hal_sys_deinit描述见下表：

表 3-148. 函数 hal_sys_deinit

函数名称	hal_sys_deinit
函数原型	int32_t hal_sys_deinit(void);
功能描述	复位SYS(DBG&Basetick)模块
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输入参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_VAL

例如：

```
/* SYS(DBG & Basetick) deinitialize */
```

```
hal_sys_deinit();
```

函数 hal_sys_debug_init

函数hal_sys_debug_init描述见下表：

表 3-149. 函数 hal_sys_debug_init

函数名称	hal_sys_debug_init
函数原型	int32_t hal_sys_debug_init(hal_sys_debug_cfg_enum debug_cfg);
功能描述	初始化DBG外设
先决条件	-
被调用函数	-
输入参数{in}	
debug_cfg	DBG配置选项信息参考 表3-146. 枚举hal_sys_debug_cfg_enum
输出参数{out}	

-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_VAL

例如：

```
/* SYS DBG initialize */
```

```
hal_sys_debug_init (SYS_DEBUG_SERIAL_WIRE);
```

函数 hal_sys_timesource_init

函数hal_sys_timesource_init描述见下表：

表 3-150. 函数 hal_sys_timesource_init

函数名称	hal_sys_timesource_init
函数原型	int32_t hal_sys_timesource_init(hal_sys_timebase_source_enum timebase_source);
功能描述	初始化系统计时器源
先决条件	-
被调用函数	-
输入参数{in}	
timebase_source	系统计时器源配置选项信息参考 表3-147. 枚举 hal_sys_timebase_source_enum
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_VAL

例如：

```
/* SYS timebase initialize */
```

```
hal_sys_timesource_init (SYS_TIMEBASE_SOURCE_SYSTICK);
```

函数 hal_sys_basetick_count_get

函数hal_sys_basetick_count_get描述见下表：

表 3-151. 函数 `hal_sys_basetick_count_get`

函数名称	<code>hal_sys_basetick_count_get</code>
函数原型	<code>uint32_t hal_sys_basetick_count_get(void);</code>
功能描述	获取当前系统计时器值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>int32_t</code>	32位系统计数器值，单位为ms

例如：

```
/* get the basetick count function */
uint32 basetick_cout = 0x00000000u;
basetick_cout = hal_sys_basetick_count_get ();
```

函数 `hal_sys_basetick_timeout_check`

函数`hal_sys_basetick_timeout_check`描述见下表：

表 3-152. 函数 `hal_sys_basetick_timeout_check`

函数名称	<code>hal_sys_basetick_timeout_check</code>
函数原型	<code>FlagStatus hal_sys_basetick_timeout_check(uint32_t time_start, uint32_t delay);</code>
功能描述	检测系统计时器是否超时
先决条件	-
被调用函数	-
输入参数{in}	
<code>time_start</code>	设置系统Delay时间的起始时间（精度：ms）
输入参数{in}	
<code>delay</code>	设置的系统Delay时间（精度：ms）

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

例如:

```
uint32_t count = 0;

/* Get current system tick counter*/

count = hal_sys_basetick_count_get();

/* check whether the delay is finished */

while(RESET == hal_sys_basetick_timeout_check(count, 500));

{

    /* delay not finished */

}

/* delay finished */
```

函数 hal_sys_basetick_delay_ms

函数hal_sys_basetick_delay_ms描述见下表:

表 3-153. 函数 hal_sys_basetick_delay_ms

函数名称	hal_sys_basetick_delay_ms
函数原型	void hal_sys_basetick_delay_ms(uint32_t time_ms);
功能描述	以ms为单位设置系统延时
先决条件	-
被调用函数	-
输入参数{in}	
time_ms	设置的系统Delay时间（单位：ms）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* Set 500ms system time delay */
```

```
hal_sys_basetick_delay_ms(500);
```

函数 hal_sys_basetick_suspend

函数hal_sys_basetick_suspend描述见下表：

表 3-154. 函数 hal_sys_basetick_suspend

函数名称	hal_sys_basetick_suspend
函数原型	void hal_sys_basetick_suspend(void);
功能描述	暂停系统计时器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* suspend system tick */
```

```
hal_sys_basetick_suspend();
```

函数 hal_sys_basetick_resume

函数hal_sys_basetick_resume描述见下表：

表 3-155. 函数 hal_sys_basetick_resume

函数名称	hal_sys_basetick_resume
函数原型	void hal_sys_basetick_resume(void);
功能描述	恢复系统计时器
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* resume system tick */
```

```
hal_sys_basetick_resume ();
```

函数 hal_sys_basetick_irq

函数hal_sys_basetick_irq描述见下表:

表 3-156. 函数 hal_sys_basetick_irq

函数名称	hal_sys_basetick_irq
函数原型	void hal_sys_basetick_irq(void);
功能描述	系统计时器中断服务函数
先决条件	-
被调用函数	SysTick_Handler()/ TIMERx_IRQHandler()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* the function is used in the relative interrupt routine */
```

```
SysTick_Handler (void)
```

```
{
```

```
    hal_sys_basetick_irq ();
```

}

函数 hal_sys_basetick_irq_handle_set

函数hal_sys_basetick_irq_handle_set描述见下表：

表 3-157. 函数 hal_sys_basetick_irq_handle_set

函数名称	hal_sys_basetick_irq_handle_set
函数原型	void hal_sys_basetick_irq_handle_set(hal_sys_basetick_irq_handle_cb irq_handler);
功能描述	设置用户自定义中断服务函数
先决条件	-
被调用函数	-
输入参数{in}	
irq_handler	用户自定义中断回调函数指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* set user-defined interrupt callback function */
hal_sys_basetick_irq_handle_cb sys_basetick_user_irq(void)
{
}

hal_sys_basetick_irq_handle_set (sys_basetick_user_irq);

```

函数 hal_sys_basetick_irq_handle_all_reset

函数hal_sys_basetick_irq_handle_all_reset描述见下表：

表 3-158. 函数 hal_sys_basetick_irq_handle_all_reset

函数名称	hal_sys_basetick_irq_handle_all_reset
函数原型	void hal_sys_basetick_irq_handle_all_reset(void);
功能描述	复位用户自定义中断服务函数

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

例如:

```
/* reset all user-defined interrupt callback function */
```

```
hal_sys_basetick_irq_handle_all_reset()
```

3.9. DMA

DMA控制器提供了一种硬件的方式在外设和存储器之间或者存储器和存储器之间传输数据，而无需CPU的介入，从而使CPU可以专注在处理其他系统功能上。章节[3.9.1](#)描述了DMA的寄存器列表，章节[3.9.2](#)对DMA库函数进行说明。

3.9.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-159. DMA 寄存器

寄存器名称	寄存器描述
DMA_INTF	DMA中断标志位寄存器
DMA_INTC	DMA中断标志位清除寄存器
DMA_CHxCTL (x=0..6)	通道x控制寄存器
DMA_CHxCNT (x=0..6)	通道x计数寄存器
DMA_CHxPADDR (x=0..6)	通道x外设基地址寄存器
DMA_CHxMADDR (x=0..6)	通道x存储器基地址寄存器

3.9.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-160. DMA 库函数

库函数名称	库函数描述
hal_dma_init	初始化DMA通道x
hal_dma_struct_init	将DMA结构体中所有参数初始化为默认值
hal_dma_deinit	复位DMA通道x的所有寄存器
hal_dma_start	启动DMA传输
hal_dma_stop	停止DMA传输
hal_dma_irq	DMA中断处理函数
hal_dma_irq_handle_set	设置用户定义的中断回调函数
hal_dma_irq_handle_all_reset	复位用户定义的中断回调函数
hal_dma_start_poll	检查DMA轮询传输完成
hal_dma_start_interrupt	启动DMA中断传输
hal_dma_stop_interrupt	停止DMA中断传输
hal_dma_channel_remap_enable	使能DMA通道映射
hal_dma_channel_remap_disable	失能DMA通道映射

枚举 dma_channel_enum

表 3-161. 枚举 dma_channel_enum

成员名称	功能描述
DMA_CH0	DMA通道0
DMA_CH1	DMA通道1
DMA_CH2	DMA通道2
DMA_CH3	DMA通道3
DMA_CH4	DMA通道4
DMA_CH5	DMA通道5
DMA_CH6	DMA通道6

枚举 hal_dma_struct_type_enum

表 3-162. 枚举 hal_dma_struct_type_enum

成员名称	功能描述
HAL_DMA_INIT_STRUCT	DMA初始化结构体
HAL_DMA_DEV_STRUCT	DMA设备信息结构体
HAL_DMA_IRQ_STRUCT	DMA中断回调函数指针结构体

枚举 hal_dma_transfer_state_enum

表 3-163. 枚举 hal_dma_transfer_state_enum

成员名称	功能描述
HAL_DMA_TARNSFER_HALF	DMA半传输完成
HAL_DMA_TARNSFER_FULL	DMA传输完成

枚举 hal_dma_error_enum

表 3-164. 枚举 hal_dma_error_enum

成员名称	功能描述
HAL_DMA_ERROR_NONE	无错误
HAL_DMA_ERROR_TRANSFER	DMA传输错误
HAL_DMA_ERROR_NOTTRANSFER	DMA未在传输状态错误
HAL_DMA_ERROR_TIMEOUT	DMA传输超时错误
HAL_DMA_ERROR_UNSUPPORT	DMA传输模式配置错误

枚举 hal_dma_state_enum

表 3-165. 枚举 hal_dma_state_enum

成员名称	功能描述
HAL_DMA_STATE_NONE	无（默认状态）
HAL_DMA_STATE_RESET	复位状态
HAL_DMA_STATE_BUSY	忙状态
HAL_DMA_STATE_TIMEOUT	传输超时状态
HAL_DMA_STATE_ERROR	传输错误状态
HAL_DMA_STATE_READY	准备状态

结构体 hal_dma_irq_struct

表 3-166. 结构体 hal_dma_irq_struct

成员名称	功能描述
error_handle	DMA通道传输错误中断处理回调函数
half_finish_handle	DMA通道半传输完成中断处理回调函数
full_finish_handle	DMA通道传输完成中断处理回调函数

结构体 hal_dma_dev_struct

表 3-167. 结构体 hal_dma_dev_struct

成员名称	功能描述
channel	DMA传输通道
dma_irq	DMA中断回调函数指针结构体
transfer_state	DMA传输状态
error_state	DMA错误状态
state	DMA设备状态
*p_periph	用户自定义外设指针
mutex	互斥锁和解锁状态

结构体 hal_dma_init_struct

表 3-168. 结构体 hal_dma_init_struct

成员名称	功能描述
direction	数据传输方向
priority	通道软件优先级
mode	循环传输模式
periph_inc	外设地址生成算法模式
memory_inc	存储器地址生成算法模式
periph_width	外设数据传输宽度
memory_width	存储器数据传输宽度

函数 hal_dma_init

函数hal_dma_init描述见下表:

表 3-169. 函数 hal_dma_init

函数名称	hal_dma_init
函数原型	int32_t hal_dma_init(hal_dma_dev_struct *dma_dev, dma_channel_enum channelx, hal_dma_init_struct *dma);
功能描述	初始化DMA通道x (x=0..6)
先决条件	-
被调用函数	-
输入参数{in}	
dma_dev	指向DMA设备信息结构体的指针, 结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输入参数{in}	
channelx	DMA通道
DMA_CHx (x=0..6)	DMA通道选择
输入参数{in}	
dma	指向DMA初始化结构体的指针, 结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE, HAL_ERR_VAL

例如:

```
/* initialize DMA channel0 */
```

```
hal_dma_init_struct dma_init_parameter;
```

```
hal_dma_dev_struct dma_info;
```

```
dma_init_parameter.direction = DMA_DIR_MEMORY_TO_PERIPH;
```

```

dma_init_parameter.periph_inc = DISABLE;
dma_init_parameter.memory_inc = ENABLE;
dma_init_parameter.periph_width = DMA_PERIPH_SIZE_8BITS;
dma_init_parameter.memory_width = DMA_MEMORY_SIZE_8BITS;
dma_init_parameter.mode = DMA_NORMAL_MODE;
dma_init_parameter.priority = DMA_PRIORITY_LEVEL_LOW;
hal_dma_init(&dma_info, DMA_CH1, &dma_init_parameter);

```

函数 hal_dma_struct_init

函数hal_dma_struct_init描述见下表：

表 3-170. 函数 hal_dma_struct_init

函数名称	hal_dma_struct_init
函数原型	int32_t hal_dma_struct_init(hal_dma_struct_type_enum struct_type, void *p_struct);
功能描述	初始化DMA结构体
先决条件	-
被调用函数	-
输入参数{in}	
hal_struct_type	结构体类型
HAL_DMA_INIT_STRUCT	DMA初始化结构体
HAL_DMA_IRQ_STRUCT	DMA中断回调函数指针结构体
HAL_DMA_DEV_STRUCT	DMA设备信息结构体
输入参数{in}	
p_struct	指向包含配置信息的DMA结构体的指针
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE, HAL_ERR_VAL

例如：

```

/* initialize the DMA structure with the default values */
hal_dma_dev_struct dma_info;
hal_dma_struct_init(HAL_DMA_DEV_STRUCT, &dma_info);

```

函数 hal_dma_deinit

函数hal_dma_deinit描述见下表：

表 3-171. 函数 hal_dma_deinit

函数名称	hal_dma_deinit
------	----------------

函数原型	int32_t hal_dma_deinit(hal_dma_dev_struct *dma_dev);
功能描述	复位DMA外设
先决条件	-
被调用函数	-
输入参数{in}	
dma_dev	指向DMA设备信息结构体的指针，结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* deinitialize DMA device structure and init structure */
```

```
hal_dma_dev_struct dma_info;
```

```
hal_dma_deinit (&dma_info);
```

函数 hal_dma_start

函数hal_dma_start描述见下表:

表 3-172. 函数 hal_dma_start

函数名称	hal_dma_start
函数原型	int32_t hal_dma_start(hal_dma_dev_struct *dma_dev, uint32_t src_addr, \n uint32_t dst_addr, uint16_t length);
功能描述	启动DMA传输
先决条件	-
被调用函数	-
输入参数{in}	
dma_dev	指向DMA设备信息结构体的指针，结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输入参数{in}	
src_addr	DMA传输源地址
输入参数{in}	
dst_addr	DMA传输目的地址
输入参数{in}	
length	DMA传输数据数目
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_BUSY, HAL_ERR_NONE

例如:

```
/* start DMA transfer */
```

```
#define TRANSFER_NUM    1024
int src_array[TRANSFER_NUM];
int dst_array[TRANSFER_NUM];
hal_dma_dev_struct dma_info;
hal_dma_start(&dma_info, src_array, dst_array, TRANSFER_NUM);
```

函数 hal_dma_stop

函数hal_dma_stop描述见下表：

表 3-173. 函数 hal_dma_stop

函数名称	hal_dma_stop
函数原型	int32_t hal_dma_stop(hal_dma_dev_struct *dma_dev);
功能描述	停止DMA传输
先决条件	-
被调用函数	-
输入参数{in}	
dma_dev	指向DMA设备信息结构体的指针，结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE, HAL_ERR_ALREADY_DONE

例如：

```
/* stop DMA transfer */
hal_dma_dev_struct dma_info;
hal_dma_stop(&dma_info);
```

函数 hal_dma_irq

函数hal_dma_irq描述见下表：

表 3-174. 函数 hal_dma_irq

函数名称	hal_dma_irq
函数原型	int32_t hal_dma_irq(hal_dma_dev_struct *dma_dev);
功能描述	DMA中断处理函数
先决条件	-
被调用函数	-
输入参数{in}	
dma_dev	指向DMA设备信息结构体的指针，结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输出参数{out}	
-	-
返回值	

int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE
---------	-------------------------------

例如:

```
/* the function is used in the relative interrupt routine */
hal_dma_dev_struct dma_info;
void DMA_Channel1_2_IRQHandler(void)
{
    hal_dma_irq(&dma_info);
}
```

函数 hal_dma_irq_handle_set

函数hal_dma_irq_handle_set描述见下表:

表 3-175. 函数 hal_dma_irq_handle_set

函数名称	hal_dma_irq_handle_set
函数原型	int32_t hal_dma_irq_handle_set(hal_dma_dev_struct *dma_dev, hal_dma_irq_struct *p_irq);
功能描述	设置DMA中断回调函数并使能DMA中断
先决条件	-
被调用函数	-
输入参数{in}	
dma_dev	指向DMA设备信息结构体的指针, 结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输入参数{in}	
p_irq	指向DMA中断回调函数结构体的指针, 结构体成员参考 表3-166. 结构体 hal_dma_irq_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* set user-defined interrupt callback function */
hal_dma_dev_struct dma_info;
hal_dma_irq_struct dma_irq;
dma_irq.full_finish_handle = dma_full_finish_func;
hal_dma_irq_handle_set(&dma_info,&dma_irq);
```

函数 hal_dma_irq_handle_all_reset

函数hal_dma_irq_handle_all_reset描述见下表:

表 3-176. 函数 hal_dma_irq_handle_all_reset

函数名称	hal_dma_irq_handle_all_reset
------	------------------------------

函数原型	int32_t hal_dma_irq_handle_all_reset(hal_dma_dev_struct *dma_dev);
功能描述	清除DMA中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	
dma_dev	指向DMA设备信息结构体的指针，结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* reset all user-defined interrupt callback function */
```

```
hal_dma_dev_struct dma_info;
```

```
hal_dma_irq_handle_all_reset(&dma_info);
```

函数 hal_dma_start_poll

函数hal_dma_start_poll描述见下表:

表 3-177. 函数 hal_dma_start_poll

函数名称	hal_dma_start_poll
函数原型	int32_t hal_dma_start_poll(hal_dma_dev_struct *dma_dev, hal_dma_transfer_state_enum transfer_state, uint32_t timeout_ms);
功能描述	检查DMA轮询传输完成
先决条件	-
被调用函数	-
输入参数{in}	
dma_dev	指向DMA设备信息结构体的指针，结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输入参数{in}	
transfer_state	DMA传输完成状态
HAL_DMA_TARNS FER_HALF	DMA传输半完成
HAL_DMA_TARNS FER_FULL	DMA传输完成
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_HARDWARE, HAL_ERR_NONE, HAL_ERR_NO_SUPPORT, HAL_ERR_ALREADY_DONE,

	HAL_ERR_TIMEOUT
--	-----------------

例如:

```
/* start amounts of data, poll transmit process and completed status */
#define TIMEOUT    10
hal_dma_dev_struct dma_info;
hal_dma_start_poll(&dma_info, HAL_DMA_TARNSEFER_FULL, TIMEOUT);
```

函数 hal_dma_start_interrupt

函数hal_dma_start_interrupt描述见下表:

表 3-178. 函数 hal_dma_start_interrupt

函数名称	hal_dma_start_interrupt
函数原型	int32_t hal_dma_start_interrupt(hal_dma_dev_struct *dma_dev, uint32_t src_addr, uint32_t dst_addr, uint16_t length, hal_dma_irq_struct *p_irq)
功能描述	启动DMA中断传输
先决条件	-
被调用函数	-
输入参数{in}	
dma_dev	指向DMA设备信息结构体的指针，结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输入参数{in}	
src_addr	DMA传输源地址
输入参数{in}	
dst_addr	DMA传输目的地址
输入参数{in}	
length	DMA传输数据数目
输入参数{in}	
p_irq	指向DMA中断回调函数结构体的指针，结构体成员参考 表3-166. 结构体 hal_dma_irq_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* start amounts of data by interrupt method */
uint8_t src_arr[10] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A};
uint8_t dst_arr[10];
hal_dma_init_struct dma_init_parameter;
hal_dma_dev_struct dma_info;
hal_dma_irq_struct dma_irq;
dma_irq.full_finish_handle = dma_full_finish_func;
```



```

dma_init_parameter.direction = DMA_DIR_MEMORY_TO_MEMORY;
dma_init_parameter.periph_inc = DISABLE;
dma_init_parameter.memory_inc = ENABLE;
dma_init_parameter.periph_width = DMA_PERIPH_SIZE_8BITS;
dma_init_parameter.memory_width = DMA_MEMORY_SIZE_8BITS;
dma_init_parameter.mode = DMA_NORMAL_MODE;
dma_init_parameter.priority = DMA_PRIORITY_LEVEL_LOW;
hal_dma_init(&dma_info,DMA_CH1, &dma_init_parameter);
hal_dma_irq_handle_set(&dma_info,&dma_irq);
hal_dma_start_interrupt(&dma_info,src_arr,dst_arr,10,&dma_irq);

```

函数 hal_dma_stop_interrupt

函数hal_dma_stop_interrupt描述见下表：

表 3-179. 函数 hal_dma_stop_interrupt

函数名称	hal_dma_stop_interrupt
函数原型	int32_t hal_dma_stop_interrupt(hal_dma_dev_struct *dma_dev);
功能描述	停止DMA中断传输
先决条件	-
被调用函数	-
输入参数{in}	
dma_dev	指向DMA设备信息结构体的指针，结构体成员参考 表3-167. 结构体 hal_dma_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE, HAL_ERR_ALREADY_DONE

例如：

```

/* stop amounts of data by interrupt method */
hal_dma_stop_interrupt (&dma_info);

```

函数 hal_dma_channel_remap_enable

函数hal_dma_channel_remap_enable描述见下表：

表 3-180. 函数 hal_dma_channel_remap_enable

函数名称	hal_dma_channel_remap_enable
函数原型	int32_t hal_dma_channel_remap_enable(uint32_t dma_remap);
功能描述	使能DMA通道映射
先决条件	-
被调用函数	-
输入参数{in}	
dma_remap	映射指定DMA请求

DMA_REMAP_TIMER16_CH0_UP	将TIMER16通道0和UP的DMA请求事件映射到DMA通道1（默认是DMA通道0）
DMA_REMAP_TIMER15_CH0_UP	将TIMER15通道2和UP的DMA请求事件映射到DMA通道3（默认是DMA通道2）
DMA_REMAP_USART0_RX	将USART0接收DMA请求事件映射到DMA通道4（默认是DMA通道2）
DMA_REMAP_USART0_TX	将USART0发送DMA请求事件映射到DMA通道3（默认是DMA通道1）
DMA_REMAP_ADC	将ADC DMA请求事件映射到DMA通道1（默认是DMA通道0）
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE, HAL_ERR_ALREADY_DONE

例如:

```
/* enable the DMA channels remapping */
hal_dma_channel_remap_enable (DMA_REMAP_USART0_TX);
```

函数 hal_dma_channel_remap_disable

函数hal_dma_channel_remap_disable描述见下表:

表 3-181. 函数 hal_dma_channel_remap_enable

函数名称	hal_dma_channel_remap_disable
函数原型	int32_t hal_dma_channel_remap_disable(uint32_t dma_remap);
功能描述	失能DMA通道映射
先决条件	-
被调用函数	-
输入参数{in}	
dma_remap	失能映射指定DMA请求
DMA_REMAP_TIMER16_CH0_UP	失能TIMER16通道0和UP的DMA请求事件映射
DMA_REMAP_TIMER15_CH0_UP	失能TIMER15通道2和UP的DMA请求事件映射
DMA_REMAP_USART0_RX	失能USART0接收DMA请求事件映射
DMA_REMAP_USART0_TX	失能USART0发送DMA请求事件映射
DMA_REMAP_ADC	失能ADC DMA请求事件映射
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE, HAL_ERR_ALREADY_DONE

例如:

```
/* disable the DMA channels remapping */
hal_dma_channel_remap_disable (DMA_REMAP_USART0_TX);
```

3.10. EXTI

EXTI是MCU中的中断/事件控制器，包括19个相互独立的边沿检测电路并且能够向处理器内核产生中断请求或唤醒事件。章节[3.10.1](#)描述了EXTI的寄存器列表，章节[3.10.2](#)对EXTI库函数进行说明。

3.10.1. 外设寄存器说明

EXTI寄存器列表如下表所示:

表 3-182. EXTI 寄存器

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

3.10.2. 外设库函数说明

EXTI HAL库函数列表如下表所示:

表 3-183. EXTI HAL 库函数

库函数名称	库函数描述
hal_exti_gpio_deinit	复位EXTI线使用的GPIO
hal_exti_internal_deinit	复位EXTI内部线
hal_exti_gpio_init	初始化EXTI线使用的GPIO
hal_exti_internal_init	初始化EXTI内部线
hal_exti_gpio_irq_handle_set	设置用户定义的中断回调函数
hal_exti_gpio_irq_handle_all_reset	重置所有用户定义的中断回调函数
hal_exti_gpio_irq	EXTI GPIO的中断处理内容功能

枚举 `hal_exti_type_enum`

表 3-184. `hal_exti_type_enum`

枚举名称	枚举描述
<code>EXTI_EVENT_TRIG_RISING</code>	事件上升沿触发
<code>EXTI_EVENT_TRIG_FALLING</code>	事件下降沿触发
<code>EXTI_EVENT_TRIG_BOTH</code>	事件上升沿和下降沿均触发
<code>EXTI_INTERRUPT_TRIG_RISING</code>	中断上升沿触发
<code>EXTI_INTERRUPT_TRIG_FALLING</code>	中断下降沿触发
<code>EXTI_INTERRUPT_TRIG_BOTH</code>	中断上升沿和下降沿均触发

枚举 `hal_exti_line_enum`

表 3-185. `hal_exti_line_enum`

枚举名称	枚举描述
<code>EXTI_PIN_0</code>	EXTI线0
<code>EXTI_PIN_1</code>	EXTI线1
<code>EXTI_PIN_2</code>	EXTI线2
<code>EXTI_PIN_3</code>	EXTI线3
<code>EXTI_PIN_4</code>	EXTI线4
<code>EXTI_PIN_5</code>	EXTI线5
<code>EXTI_PIN_6</code>	EXTI线6
<code>EXTI_PIN_7</code>	EXTI线7
<code>EXTI_PIN_8</code>	EXTI线8
<code>EXTI_PIN_9</code>	EXTI线9
<code>EXTI_PIN_10</code>	EXTI线10
<code>EXTI_PIN_11</code>	EXTI线11

枚举名称	枚举描述
EXTI_PIN_12	EXTI线12
EXTI_PIN_13	EXTI线13
EXTI_PIN_14	EXTI线14
EXTI_PIN_15	EXTI线15
EXTI_LVD_16	EXTI线16
EXTI_RTC_ALARM_17	EXTI线17
EXTI_USBFS_WAKEUP_18	EXTI线18
EXTI_RTC_TAMPER_TIMESTAMP_19	EXTI线19
EXTI_CMP_OUTPUT_T_21	EXTI线21
EXTI_CMP_OUTPUT_22	EXTI线22
EXTI_USART0_WAKEUP_25	EXTI线25
EXTI_CEC_WAKEUP_P_27	EXTI线27

枚举 hal_exti_internal_line_enum

表 3-186. hal_exti_internal_line_enum

枚举名称	枚举描述
EXTI_LINE_16_LVD	低电压检测EXTI线
EXTI_LINE_17_RTC_ALARM	RTC闹钟EXTI线
EXTI_LINE_18_USBFS_WAKEUP	USB唤醒EXTI线
EXTI_LINE_19_RTC_TAMPER_TIMESTAMP_TAMP	RTC侵入时间戳EXTI线
EXTI_LINE_21_CM	比较器0输出EXTI线

枚举名称	枚举描述
P_OUTPUT	
EXTI_LINE_22_CM P_OUTPUT	比较器1输出EXTI线
EXTI_LINE_25_US ART0_WAKEUP	USART0唤醒EXTI线
EXTI_LINE_27_CE C_WAKEUP	CEC唤醒EXTI线

枚举 `hal_exti_irq_index`

表 3-187. `hal_exti_irq_index`

枚举名称	枚举描述
EXTI_0_1_IRQHandler_USED	EXTI线[1:0]中断处理
EXTI_2_3_IRQHandler_USED	EXTI线[3:2]中断处理
EXTI_4_15_IRQHandler_USED	EXTI线[15:4]中断处理

函数 `hal_exti_gpio_deinit`

函数`hal_exti_gpio_deinit`描述见下表:

表 3-188. 函数 `hal_exti_gpio_deinit`

函数名称	<code>hal_exti_gpio_deinit</code>
函数原形	<code>int32_t hal_exti_gpio_deinit(uint32_t gpio_periph, uint32_t pin)</code>
功能描述	复位EXTI线使用的GPIO
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
<i>GPIOx</i>	端口选择(x = A,B,C, D,F)
输入参数{in}	
pin	GPIO引脚

<i>GPIO_PIN_x</i>	引脚选择 (x=0..15)
<i>GPIO_PIN_ALL</i>	所有引脚
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_VAL / HAL_ERR_NONE

例如:

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

```
hal_exti_gpio_deinit(GPIOA, GPIO_PIN_9);
```

函数 **hal_exti_internal_deinit**

函数hal_exti_internal_deinit描述见下表:

表 3-189. 函数 hal_exti_internal_deinit

函数名称	hal_exti_internal_deinit
函数原形	void hal_exti_internal_deinit(hal_exti_internal_line_enum line)
功能描述	复位EXTI内部线
先决条件	-
被调用函数	-
输入参数{in}	
line	EXTI内部线, 枚举成员参考 表3-186. hal_exti_internal_line_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* deinitialize the RTC_ALARM line */
```

```
hal_exti_internal_deinit(EXTI_LINE_17_RTC_ALARM);
```

函数 **hal_exti_gpio_init**

函数hal_exti_gpio_init描述见下表:

表 3-190. 函数 hal_exti_gpio_init

函数名称	hal_exti_gpio_init
函数原形	int32_t hal_exti_gpio_init(uint32_t gpio_periph, uint32_t pin, uint32_t pull, hal_exti_type_enum exti_type)
功能描述	初始化EXTI线使用的GPIO
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C, D,F)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
输入参数{in}	
pull	GPIO引脚的上拉/下拉模式
HAL_GPIO_PULL_NONE	浮空模式
HAL_GPIO_PULL_UP	上拉模式
HAL_GPIO_PULL_DOWN	下拉模式
输入参数{in}	
exti_type	EXTI类型，枚举成员参考 表3-184. hal_exti_type_enum
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_VAL / HAL_ERR_ALREADY_DONE / HAL_ERR_NONE

例如：

```
/* initialize the configuration of EXTI gpio */
```



```
hal_exti_gpio_init(GPIOA, GPIO_PIN_9, HAL_GPIO_PULL_NONE, EXTI_EVENT_TRIG_RISING);
```

函数 hal_exti_internal_init

函数hal_exti_internal_init描述见下表：

表 3-191. 函数 hal_exti_internal_init

函数名称	hal_exti_internal_init
函数原形	void hal_exti_internal_init(hal_exti_internal_line_enum line, hal_exti_type_enum exti_type)
功能描述	初始化EXTI内部线
先决条件	-
被调用函数	-
输入参数{in}	
line	EXTI内部线，枚举成员参考 表3-186. hal_exti_internal_line_enum
输入参数{in}	
exti_type	EXTI类型，枚举成员参考 表3-184. hal_exti_type_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the configuration of EXTI internal */
```

```
hal_exti_internal_init (EXTI_LINE_17_RTC_ALARM, EXTI_EVENT_TRIG_RISING);
```

函数 hal_exti_gpio_irq_handle_set

函数hal_exti_gpio_irq_handle_set描述见下表：

表 3-192. 函数 hal_exti_gpio_irq_handle_set

函数名称	hal_exti_gpio_irq_handle_set
函数原形	int32_t hal_exti_gpio_irq_handle_set(hal_gpio_irq_handle_cb irq_handler)
功能描述	设置用户定义的中断回调函数
先决条件	-

被调用函数	-
输入参数{in}	
irq_handler	配置EXTI回调函数，由用户自己实现
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS / HAL_ERR_NONE

例如：

```

/* set EXTI line 13 interrupt callback function */
void s_gpio_irq_handle(uint32_t id, uint32_t event)
{
    if(EXTI_PIN_13 == id){
        gd_eval_led_toggle(LED2);
    }
}

hal_exti_gpio_irq_handle_set(exti_irq_handler);

```

函数 hal_exti_gpio_irq_handle_all_reset

函数hal_exti_gpio_irq_handle_all_reset描述见下表：

表 3-193. 函数 hal_exti_gpio_irq_handle_all_reset

函数名称	hal_exti_gpio_irq_handle_all_reset
函数原形	void hal_exti_gpio_irq_handle_all_reset(void)
功能描述	重置所有用户定义的中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback function */
```

```
hal_exti_gpio_irq_handle_all_reset();
```

函数 hal_exti_gpio_irq

函数hal_exti_gpio_irq描述见下表：

表 3-194. 函数 hal_exti_gpio_irq

函数名称	hal_exti_gpio_irq
函数原形	void hal_exti_gpio_irq(hal_exti_irq_index index)
功能描述	EXTI GPIO的中断处理内容功能
先决条件	-
被调用函数	-
输入参数{in}	
index	指示该函数将由哪个中断源调用，枚举成员参考 表3-187. hal_exti_irq_index
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* EXTI_GPIO interrupt handler content function */
```

```
hal_exti_gpio_irq(EXTI_0_1_IRQHandler_USED);
```

3.11. FMC

FMC是MCU中的Flash控制器，其中包括存储数据的主编程块和选项字节。章节[3.11.1](#)描述了FMC的寄存器列表，章节[3.11.2](#)对FMC库函数进行说明。

3.11.1. 外设寄存器说明

FMC寄存器列表如下：

表 3-195. FMC 寄存器

寄存器	描述
FMC_WS	等待状态寄存器
FMC_KEY	解锁寄存器
FMC_OBKEY	选项字节解锁寄存器
FMC_STAT	状态寄存器
FMC_CTL	控制寄存器
FMC_ADDR	地址寄存器
FMC_OBSTAT	选项字节状态寄存器
FMC_WP	写保护寄存器
FMC_WSEN	等待状态使能寄存器
FMC_PID	产品ID寄存器

3.11.2. 外设库函数说明

FMC固件库函数列举如下表：

表 3-196. FMC 固件库函数

函数名称	函数描述
hal_fmc_unlock	解锁FMC主编程块操作
hal_fmc_lock	锁定FMC主编程块操作
hal_fmc_wait_state_enable	FMC等待状态使能
hal_fmc_wait_state_disable	FMC等待状态失能
hal_fmc_ready_wait	检查FMC是否准备好
hal_fmc_wscnt_set	设置FMC等待状态计数值
hal_fmc_region_read	读取flash指定区域的数据
hal_fmc_word_program	对指定地址进行字编程
hal_fmc_halfword_program	对指定地址进行半字编程
hal_fmc_region_write	对指定区域进行编程
hal_fmc_page_erase	对指定地址进行页擦除
hal_fmc_mass_erase	全片擦除
hal_fmc_region_erase	对指定区域进行擦除
hal_fmc_irq	处理所有FMC中断请求
hal_fmc_irq_handle_set	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
hal_fmc_irq_handle_all_reset	复位所有的用户自定义的中断回调函数
hal_ob_unlock	解锁选项字节
hal_ob_lock	锁定选项字节
hal_ob_reset	复位使得选项字节生效
hal_ob_erase	擦除选项字节
hal_ob_security_protection_config	设置安全保护
hal_ob_user_write	写选项字节user值
hal_ob_data_program	写选项字节data值

函数名称	函数描述
hal_ob_wp_enable	使能指定区域写保护
hal_ob_wp_disable	除能指定区域写保护
hal_ob_parm_get	获取当前生效的选项字节数值
hal_ob_write_protection_enable	根据当前选项字节启用选项字节写保护（OB_WP）
hal_ob_parm_config	整体配置选项字节数值

枚举 fmc_state_enum

表 3-197. fmc_state_enum

枚举名称	枚举描述
FMC_READY	操作完成
FMC_BUSY	操作进行中
FMC_PGERR	编程错误
FMC_PGAERR	编程对齐错误
FMC_WPERR	写保护错误
FMC_TOERR	超时错误
FMC_OB_HSPC	高安全保护级别

结构体 ob_parm_struct

表 3-198. 结构体 ob_parm_struct

成员名称	功能描述
spc	选项字节spc参数
user	选项字节用户参数
data0	选项字节data0
data1	选项字节data1
wp0	选项字节wp0参数
wp1	选项字节wp1参数

结构体 hal_sector_addr_range_struct

表 3-199. 结构体 hal_sector_addr_range_struct

成员名称	功能描述
sector_start_addr	实际生效区域的起始地址
sector_end_addr	实际生效区域的截止地址

结构体 hal_fmc_irq_struct

表 3-200. 结构体 hal_fmc_irq_struct

成员名称	功能描述
error_handle	操作错误回调函数
finish_handle	操作完成回调函数

结构体 hal_ob_parm_config_struct

表 3-201. 结构体 hal_ob_parm_config_struct

成员名称	功能描述
ob_type	选项字节类型: OB_TYPE_WRP(选项字节写保护),OB_TYPE_SPC(选项字节安全保护),OB_TYPE_USER(选项字节user)和OB_TYPE_DATA(选项字节data)
wrp_state	写保护状态: OB_WRP_ENABLE (使能写保护) 和OB_WRP_DISABLE (除能写保护)
wrp_addr	写保护目标区域起始地址
wrp_size	写保护目标区域大小
spc_level	安全保护级别: OB_SPC_0 (无安全保护), OB_SPC_1 (低级安全保护) 和 OB_SPC_2 (高级安全保护)
user	目标user值, 设置后会与前user值与运算
data_value	目标data值

函数 hal_fmc_unlock

函数hal_fmc_unlock描述见下表:

表 3-202. 函数 hal_fmc_unlock

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

例如:

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

```
hal_fmc_unlock();
```

函数 hal_fmc_lock

函数hal_fmc_lock描述见下表:

表 3-203. 函数 hal_fmc_lock

函数名称	hal_fmc_lock
函数原型	void hal_fmc_lock(void);
功能描述	锁定FMC主编程块操作
先决条件	hal_fmc_unlock

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
hal_fmc_lock();
```

函数 hal_fmc_unlock

函数hal_fmc_unlock描述见下表：

表 3-204. 函数 hal_fmc_unlock

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

例如：

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

```
hal_fmc_unlock();
```

函数 hal_fmc_wait_state_enable

函数hal_fmc_wait_state_enable描述见下表：

表 3-205. 函数 hal_fmc_wait_state_enable

函数名称	hal_fmc_wait_state_enable
函数原型	void hal_fmc_wait_state_enable(void);
功能描述	使能FMC等待状态
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable fmc wait state */
```

```
hal_fmc_wait_state_enable();
```

函数 hal_fmc_wait_state_disable

函数hal_fmc_wait_state_disable描述见下表：

表 3-206. 函数 hal_fmc_wait_state_disable

函数名称	hal_fmc_wait_state_disable
函数原型	void hal_fmc_wait_state_disable(void);
功能描述	失能FMC等待状态
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable fmc wait state */
```

```
hal_fmc_wait_state_disable();
```

函数 hal_fmc_ready_wait

函数hal_fmc_ready_wait描述见下表：

表 3-207. 函数 hal_fmc_ready_wait

函数名称	hal_fmc_ready_wait
函数原型	fmc_state_enum hal_fmc_ready_wait(uint32_t timeout);
功能描述	检查FMC是否准备好
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：


```
/* check whether FMC is ready or not */
```

```
hal_fmc_ready_wait(0xFF);
```

函数 hal_fmc_wscnt_set

函数hal_fmc_wscnt_set描述见下表：

表 3-208. 函数 hal_fmc_wscnt_set

函数名称	hal_fmc_wscnt_set
函数原型	void hal_fmc_wscnt_set(uint8_t wscnt);;
功能描述	设置等待状态计数值
先决条件	-
输入参数{in}	
wscnt	等待状态计数值
WS_WSCNT_0	FMC 0个等待状态
WS_WSCNT_1	FMC 1个等待状态
WS_WSCNT_2	FMC 2个等待状态
输出参数{out}	
-	-
返回值	
-	-

例如：

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

```
hal_fmc_wscnt_set (WS_WSCNT_1);
```

函数 hal_fmc_region_read

函数hal_fmc_region_read描述见下表：

表 3-209. 函数 hal_fmc_region_read

函数名称	hal_fmc_region_read
函数原型	void hal_fmc_region_read(uint32_t start_addr, uint8_t *data, uint32_t size);
功能描述	读指定区域数值
先决条件	-
输入参数{in}	
start_addr	指定区域首地址
输入参数{in}	
data	指向所读取数据的指针
输入参数{in}	
size	指定区域的大小
输出参数{out}	
data	指向所读取数据的指针
返回值	

-	-
---	---

例如:

```
/* read flash target region */
```

```
uint8_t *data;
```

```
hal_fmc_region_read(0x08004000, data, 0x400);
```

函数 hal_fmc_word_program

函数hal_fmc_word_program描述见下表:

表 3-210. 函数 hal_fmc_word_program

函数名称	hal_fmc_word_program
函数原型	fmc_state_enum hal_fmc_word_program(uint32_t addr, uint32_t data);
功能描述	字编程
先决条件	hal_fmc_unlock
输入参数{in}	
addr	编程操作的目标地址
输入参数{in}	
data	编程操作的目标数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	参考 枚举fmc_state_enum

例如:

```
/* write flash target address in word */
```

```
hal_fmc_word_program(0x8004000, 0x12345678);
```

函数 hal_fmc_halfword_program

函数hal_fmc_doubleword_program描述见下表:

表 3-211. 函数 hal_fmc_halfword_program

函数名称	hal_fmc_halfword_program
函数原型	fmc_state_enum hal_fmc_halfword_program(uint32_t addr, uint16_t data);
功能描述	对指定地址半字编程
先决条件	hal_fmc_unlock
输入参数{in}	
addr	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	

-	-
返回值	
fmc_state_enum	参考 枚举fmc_state_enum

例如:

```
/* write flash target address in word */
```

```
fmc_state_enum state = hal_fmc_doubleword_program (0x08004000, 0xaabbccddeeffeef);
```

函数 hal_fmc_region_write

函数hal_fmc_region_write描述见下表:

表 3-212. 函数 hal_fmc_region_write

函数名称	hal_fmc_region_write
函数原型	int32_t hal_fmc_region_write(uint32_t start_addr, uint8_t *data, uint32_t size);
功能描述	将指定数据块写入指定区域
先决条件	hal_fmc_unlock
输入参数{in}	
start_addr	指定区域的起始地址
输入参数{in}	
data	指向所写入数据块的指针
输入参数{in}	
size	指定区域的大小
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如:

```
/* write flash target region with amounts of data */
```

```
uint8_t data[0x400] = { 0 };
```

```
fmc_state_enum state = hal_fmc_region_write (0x08004000, data, 0x400);
```

函数 hal_fmc_page_erase

函数hal_fmc_page_erase描述见下表:

表 3-213. 函数 hal_fmc_page_erase

函数名称	hal_fmc_page_erase
函数原型	fmc_state_enum hal_fmc_page_erase(uint32_t start_addr);
功能描述	擦除目标地址所在的页
先决条件	hal_fmc_unlock
输入参数{in}	

start_addr	擦除的目标地址-
输出参数{out}	
-	-
返回值	
fmc_state_enum	参考 枚举fmc_state_enum

例如:

```
/* erase the page which start address locating in */
```

```
fmc_state_enum state = hal_fmc_page_erase(0x8004000);
```

函数 hal_fmc_mass_erase

函数hal_fmc_mass_erase描述见下表:

表 3-214. 函数 hal_fmc_mass_erase

函数名称	hal_fmc_mass_erase
函数原型	fmc_state_enum hal_fmc_mass_erase(void);
功能描述	全片擦除
先决条件	hal_fmc_unlock
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	参考 枚举fmc_state_enum

例如:

```
/* erase the whole flash */
```

```
fmc_state_enum state = hal_fmc_mass_erase();
```

函数 hal_fmc_region_erase

函数hal_fmc_region_erase描述见下表:

表 3-215. 函数 hal_fmc_region_erase

函数名称	hal_fmc_region_erase
函数原型	int32_t hal_fmc_region_erase(uint32_t start_addr, uint32_t size);
功能描述	擦除指定区域
先决条件	hal_fmc_unlock
输入参数{in}	
start_addr	指定区域的起始地址
输入参数{in}	
size	指定区域的大小
输出参数{out}	

-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如:

```
/* erase flash target region */
```

```
hal_fmc_region_erase(0x8004000, 0x400);
```

函数 hal_fmc_irq

函数hal_fmc_irq描述见下表:

表 3-216. 函数 hal_fmc_irq

函数名称	hal_fmc_irq
函数原型	void hal_fmc_irq();
功能描述	处理所有FMC中断请求
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* process the FMC interrupt */
```

```
hal_fmc_irq_struct p_irq;
```

```
hal_uart_irq();
```

函数 hal_fmc_irq_handle_set

函数hal_fmc_irq_handle_set描述见下表:

表 3-217. 函数 hal_fmc_irq_handle_set

函数名称	hal_fmc_irq_handle_set
函数原型	void hal_fmc_irq_handle_set(hal_fmc_irq_struct *p_irq);
功能描述	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
先决条件	-
输入参数{in}	
p_irq	指向flash中断结构体，结构体成员参考 结构体hal_fmc_irq_struct
输出参数{out}	
-	-

返回值	
-	-

例如：

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

```
hal_fmc_irq_struct p_irq;
```

```
hal_fmc_irq_handle_set (&p_irq);
```

函数 hal_fmc_irq_handle_all_reset

函数hal_fmc_irq_handle_all_reset描述见下表：

表 3-218. 函数 hal_fmc_irq_handle_all_reset

函数名称	hal_fmc_irq_handle_all_reset
函数原型	void hal_fmc_irq_handle_all_reset(hal_fmc_irq_struct *p_irq);
功能描述	复位所有的用户自定义的中断回调函数
先决条件	-
输入参数{in}	
p_irq	指向flash中断结构体，结构体成员参考 结构体hal_fmc_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_fmc_irq_struct p_irq;
```

```
hal_fmc_irq_handle_reset(&p_irq);
```

函数 hal_ob_unlock

函数hal_ob_unlock描述见下表：

表 3-219. 函数 hal_ob_unlock

函数名称	hal_ob_unlock
函数原型	void hal_ob_unlock(void);
功能描述	解锁选项字节
先决条件	hal_fmc_unlock
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* unlock option byte */

void hal_ob_unlock(void);
```

函数 hal_ob_lock

函数hal_ob_lock描述见下表：

表 3-220. 函数 hal_ob_lock

函数名称	hal_ob_lock
函数原型	void hal_ob_lock(void);
功能描述	锁定选项字节
先决条件	hal_ob_unlock
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock option byte */

hal_ob_lock();
```

函数 hal_ob_reset

函数hal_ob_reset描述见下表：

表 3-221. 函数 hal_ob_reset

函数名称	hal_ob_reset
函数原型	void hal_ob_reset(void);
功能描述	系统复位致选项字节重新装载
先决条件	hal_ob_unlock
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset option byte */

hal_ob_reset();
```

函数 hal_ob_erase

函数hal_ob_erase描述见下表：

表 3-222. 函数 hal_ob_erase

函数名称	hal_ob_erase
函数原型	fmc_state_enum hal_ob_erase(void);
功能描述	擦除选项字节
先决条件	hal_ob_unlock
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	参考 枚举fmc_state_enum

例如：

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

```
fmc_state_enum state = hal_ob_erase();
```

函数 hal_ob_security_protection_config

函数hal_ob_security_protection_config描述见下表：

表 3-223. 函数 hal_ob_security_protection_config

函数名称	hal_ob_security_protection_config
函数原型	fmc_state_enum hal_ob_security_protection_config(uint8_t ob_spc);
功能描述	设定保护级别
先决条件	hal_ob_unlock
输入参数{in}	
ob_spc	特定的安全保护数值
FMC_NSPC	无安全保护
FMC_LSPC	低安全保护等级
FMC_HSPC	高安全保护等级
输出参数{out}	
-	-
返回值	
fmc_state_enum	参考 枚举fmc_state_enum

例如：

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

```
fmc_state_enum state = hal_ob_security_protection_config(FMC_LSPC);
```


函数 hal_ob_user_write

函数hal_ob_user_write描述见下表：

表 3-224. 函数 hal_ob_user_write

函数名称	hal_ob_user_write
函数原型	fmc_state_enum hal_ob_user_write(uint8_t ob_user);
功能描述	配置选项字节user数值
先决条件	hal_ob_unlock
输入参数{in}	
ob_user	用户选项字节
OB_FWDGT_HW	配置为硬件看门狗
OB_DEEPSLEEP_RST	当进入深度睡眠状态时不复位
OB_STDBY_RST	当进入standby状态时不复位
OB_BOOT1_SET_1	设置BOOT1为1
OB_VDDA_DISABLE	关闭VDDA监测器
OB_SRAM_PARITY_ENABLE	使能SRAM极性检查
输出参数{out}	
-	-
返回值	
fmc_state_enum	参考 枚举fmc_state_enum

例如：

```
/* write option byte user */
```

```
fmc_state_enum state = hal_ob_user_write (OB_FWDGT_HW);
```

函数 hal_ob_data_program

函数hal_ob_data_program描述见下表：

表 3-225. 函数 hal_ob_data_program

函数名称	hal_ob_data_program
函数原型	fmc_state_enum hal_ob_data_program(uint16_t ob_data);
功能描述	配置选项字节data值
先决条件	hal_ob_unlock
输入参数{in}	
ob_data	选项字节data值
输出参数{out}	
-	-
返回值	
fmc_state_enum	参考 枚举fmc_state_enum

例如：

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

```
fmc_state_enum state = hal_ob_data_program (0x1357) ;
```

函数 hal_ob_wp_enable

函数hal_ob_wp_enable描述见下表：

表 3-226. 函数 hal_ob_wp_enable

函数名称	hal_ob_wp_enable
函数原型	hal_sector_addr_range_struct hal_ob_wp_enable(uint32_t start_addr, uint32_t data_size);
功能描述	使能flash指定区域写保护
先决条件	hal_ob_unlock
输入参数{in}	
start_addr	指定区域起始地址
输入参数{in}	
data_size	指定区域大小
输出参数{out}	
-	-
返回值	
hal_sector_addr_range	实际生效的地址范围，参考 结构体hal_sector_addr_range_struct

例如：

```
/* enable the targeted address region written protection */
```

```
hal_sector_addr_range_struct sector = hal_ob_wp_enable(0x8004000, 0x1000);
```

函数 hal_ob_wp_disable

函数hal_ob_wp_disable描述见下表：

表 3-227. 函数 hal_ob_wp_disable

函数名称	hal_ob_wp_disable
函数原型	hal_sector_addr_range_struct hal_ob_wp_disable(uint32_t start_addr, uint32_t data_size);
功能描述	除能flash指定区域写保护
先决条件	hal_ob_unlock
输入参数{in}	
start_addr	指定区域起始地址
输入参数{in}	
data_size	指定区域大小
输出参数{out}	

-	-
返回值	
hal_sector_addr_range	实际生效的地址范围，参考 结构体hal_sector_addr_range_struct

例如：

```
/* disable the targeted address region written protection */
```

```
hal_sector_addr_range_struct sector = hal_ob_wp_disable(0x8004000, 0x1000);
```

函数 hal_ob_parm_get

函数hal_ob_parm_get描述见下表：

表 3-228. 函数 hal_ob_parm_get

函数名称	hal_ob_parm_get
函数原型	int32_t hal_ob_parm_get(ob_parm_struct *p_parm)
功能描述	获取当前生效的选项字节值
先决条件	-
输入参数{in}	
-	-
输出参数{out}	
p_parm	参考 结构体ob_parm_struct
返回值	
fmc_state_enum	参考 枚举fmc_state_enum

例如：

```
/* enable option byte write protection(OB_WP) depending on current option byte */
```

```
hal_ob_write_protection_enable(0x01);
```

函数 hal_ob_write_protection_enable

函数hal_ob_write_protection_enable描述见下表：

表 3-229. 函数 hal_ob_write_protection_enable

函数名称	hal_ob_write_protection_enable
函数原型	fmc_state_enum hal_ob_write_protection_enable(uint16_t ob_wp)
功能描述	根据当前选项字节启用选项字节写保护（OB_WP）
先决条件	-
输入参数{in}	
ob_wp	0-0xFFFF
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如:

```
/* get option byte parameters, stored in register FMC_OBSTAT and FMC_WP */  
  
ob_parm_struct ob_parm_get;  
  
hal_ob_parm_get(&ob_parm_get);
```

函数 hal_ob_parm_config

函数hal_ob_parm_config描述见下表:

表 3-230. 函数 hal_ob_parm_config

函数名称	hal_ob_parm_config
函数原型	int32_t hal_ob_parm_config(hal_ob_parm_config_struct *ob_parm);
功能描述	整体配置选项字节
先决条件	hal_ob_unlock
输入参数{in}	
ob_parm	参考 结构体hal_ob_parm_config_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如:

```
/* configure option byte parameters thoroughly */  
  
hal_ob_parm_config_struct ob_parm_config;  
  
ob_parm_config.ob_type = OB_TYPE_WRP|OB_TYPE_SPC|OB_TYPE_USER |  
OB_TYPE_DATA;  
  
ob_parm_config.spc_level = OB_SPC_1;  
  
ob_parm_config.user = 0x43;  
  
ob_parm_config.data_value = 0x1234;  
  
ob_parm_config.wrp_addr = 0x8006000;  
  
ob_parm_config.wrp_size = 0x5000;  
  
ob_parm_config.wrp_state = OB_WRP_ENABLE;  
  
hal_ob_parm_config(&ob_parm_config);
```

3.12. FWDGT

独立看门狗定时器（FWDGT）是一个硬件计时电路，用来监测由软件故障导致的系统故障。适合于需要独立环境且对计时精度要求不高的场合。章节[3.12.1](#)描述了FWDGT的寄存器列表，

章节[3.12.2](#)对FWDGT库函数进行说明

3.12.1. 外设寄存器说明

FWDGT寄存器列表如下表所示：

表 3-231. FWDGT 寄存器

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

3.12.2. 外设库函数说明

FWDGT HAL库函数列表如下表所示：

表 3-232. FWDGT HAL 库函数

库函数名称	库函数描述
hal_fwdgt_struct_init	初始化FWDGT结构体
hal_fwdgt_init	初始化FWDGT
hal_fwdgt_deinit	复位初始化FWDGT

枚举 hal_fwdgt_state_enum

表 3-233. hal_fwdgt_state_enum

枚举名称	枚举描述
HAL_FWDGT_STATE_NONE	无（默认值）
HAL_FWDGT_STATE_RESET	独立看门狗状态重置
HAL_FWDGT_STATE_READY	独立看门狗准备
HAL_FWDGT_STATE_BUSY	独立看门狗忙

枚举 `hal_fwdgt_struct_type_enum`

表 3-234. `hal_fwdgt_struct_type_enum`

枚举名称	枚举描述
<code>HAL_FWDGT_INIT_STRUCT</code>	独立看门狗初始化结构体
<code>HAL_FWDGT_DEV_STRUCT</code>	独立看门狗设备信息结构体

结构体 `hal_fwdgt_dev_struct`

表 3-235. `hal_fwdgt_dev_struct`

成员名称	成员描述
<code>state</code>	独立看门狗状态
<code>mutex</code>	互斥锁和解锁状态

结构体 `hal_fwdgt_init_struct`

表 3-236. `hal_fwdgt_init_struct`

成员名称	成员描述
<code>fwdgt_pre_select</code>	独立看门狗时钟分频选择
<code>wdgt_cnt_value</code>	看门狗计数器窗口值
<code>fwdgt_cnt_reload_value</code>	独立看门狗重装载值

函数 `hal_fwdgt_struct_init`

函数 `hal_fwdgt_struct_init` 描述见下表：

表 3-237. 函数 `hal_fwdgt_struct_init`

函数名称	<code>hal_fwdgt_struct_init</code>
函数原形	<code>void hal_fwdgt_struct_init(hal_fwdgt_struct_type_enum hal_struct_type, void *p_struct);</code>
功能描述	初始化FWDGT结构体
先决条件	-
被调用函数	-
输入参数 {in}	

hal_struct_type	指针，指向 hal_fwdgt_struct_type_enum 枚举，枚举成员参考 表 3-234. hal_fwdgt_struct_type_enum
输入参数{in}	
*p_struct	指向包含配置信息的FWDGT结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the FWDGT STRUCT */
```

```
hal_fwdgt_struct_init(HAL_FWDGT_INIT_STRUCT, &fwdgt_init_parameter);
```

函数 hal_fwdgt_init

函数hal_fwdgt_init描述见下表：

表 3-238. 函数 hal_fwdgt_init

函数名称	hal_fwdgt_init
函数原形	int32_t hal_fwdgt_init(hal_fwdgt_dev_struct *fwdgt_dev, hal_fwdgt_init_struct *p_fwdgt_init);
功能描述	启动FWDGT
先决条件	-
被调用函数	-
输入参数{in}	
fwdgt_dev	指向 FWDGT 设备信息结构体的指针，结构体成员参考 表 3-235. hal_fwdgt_dev_struct
输入参数{in}	
p_fwdgt_init	指向FWDGT初始化结构体的指针，结构体成员参考 表3-235. hal_fwdgt_dev_struct
输出参数{out}	
-	-
返回值	

int32_t	HAL_ERR_VAL / HAL_ERR_NONE
---------	----------------------------

例如:

```
/* initialize the FWDGT */
```

```
hal_fwdgt_init(&fwdgt_info,&fwdgt_init_parameter);
```

函数 hal_fwdgt_deinit

函数hal_fwdgt_deinit描述见下表:

表 3-239. 函数 hal_fwdgt_deinit

函数名称	hal_fwdgt_deinit
函数原形	void hal_fwdgt_deinit(hal_fwdgt_dev_struct *fwdgt_dev);
功能描述	
先决条件	-
被调用函数	-
输入参数{in}	
fwdgt_dev	指向FWDGT设备信息结构体的指针，结构体成员参考 表3-235. hal_fwdgt_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* deinitialize the configuration of FWDGT */
```

```
hal_fwdgt_deinit(&fwdgt_info);
```

3.13. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.13.1](#)描述了GPIO的寄存器列表，章节[3.13.2](#)对GPIO库函数进行说明。

3.13.1. 外设寄存器说明

GPIO寄存器列表如下表所示:

表 3-240. GPIO 寄存器

寄存器名称	寄存器描述
GPIOx_CTL	端口控制寄存器
GPIOx_OMODE	端口输出模式寄存器
GPIOx_OSPD0	端口输出速度寄存器0
GPIOx_PUD	端口上拉/下拉寄存器
GPIOx_ISTAT	端口输入状态寄存器
GPIOx_OCTL	端口输出控制寄存器
GPIOx_BOP	端口位操作寄存器
GPIOx_LOCK	端口配置锁定寄存器
GPIOx_AFSEL0	备用功能选择寄存器0
GPIOx_AFSEL1	备用功能选择寄存器1
GPIOx_BC	位清除寄存器
GPIOx_TG	端口位翻转寄存器
GPIOx_OSPD1	端口输出速度寄存器1

3.13.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-241. GPIO 库函数

库函数名称	库函数描述
hal_gpio_init	GPIOx PINx初始化
hal_gpio_bit_set	置位引脚值
hal_gpio_bit_reset	复位引脚值
hal_gpio_struct_init	GPIO初始化结构体初始化
hal_gpio_deinit	复位GPIOx PINx

枚举 hal_gpio_mode_enum

表 3-242. 枚举 hal_gpio_mode_enum

成员名称	功能描述
GPIO_MODE_ANALOG	模拟模式
GPIO_MODE_INPUT	输入模式
GPIO_MODE_OUTPUT_PP	推挽输出
GPIO_MODE_OUTPUT_OD	开漏输出
GPIO_MODE_AF_PP	复用推挽
GPIO_MODE_AF_OD	复用开漏

枚举 hal_gpio_pull_enum

表 3-243. 枚举 hal_gpio_pull_enum

成员名称	功能描述
GPIO_PULL_NONE	悬空模式

GPIO_PULL_UP	端口上拉模式
GPIO_PULL_DOWN	端口下拉模式

枚举 hal_gpio_ospeed_enum

表 3-244. 枚举 hal_gpio_ospeed_enum

成员名称	功能描述
GPIO_OSPEED_2MHZ	输出最大速度2M（复位值）
GPIO_OSPEED_10MHZ	输出最大速度10M
GPIO_OSPEED_50MHZ	输出最大速度50M
GPIO_OSPEED_MAX	最大速度超过50MHz

枚举 hal_gpio_af_enum

表 3-245. 枚举 hal_gpio_af_enum

成员名称	功能描述
GPIO_AF_0	选择AF0功能（复位值）
GPIO_AF_1	选择AF1功能
GPIO_AF_2	选择AF2功能
GPIO_AF_3	选择AF3功能
GPIO_AF_4	选择AF4功能（仅端口A,B支持）
GPIO_AF_5	选择AF5功能（仅端口A,B支持）
GPIO_AF_6	选择AF6功能（仅端口A,B支持）
GPIO_AF_7	选择AF7功能（仅端口A,B支持）

结构体 hal_gpio_init_struct

表 3-246. 结构体 hal_gpio_init_struct

成员名称	功能描述
mode	端口模式选择，参考 表3-242. 枚举hal_gpio_mode_enum
pull	端口上拉/下拉选择，参考 表3-243. 枚举hal_gpio_pull_enum
ospeed	端口速度选择，参考 表3-244. 枚举hal_gpio_ospeed_enum
af	端口备用功能选择，参考 表3-245. 枚举hal_gpio_af_enum

函数 hal_gpio_init

函数hal_gpio_init描述见下表：

表 3-247. 函数 hal_gpio_init

函数名称	hal_gpio_init
函数原型	int32_t hal_gpio_init(uint32_t gpio_periph, uint32_t pin, hal_gpio_init_struct *p_init);

功能描述	GPIOx初始化
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择(x = A,B,C,D,F)
输入参数{in}	
pin	PIN引脚
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	所有引脚
输入参数{in}	
p_init	指向GPIO初始化结构体的指针，结构体成员参考 表 3-246. 结构体 hal_gpio_init_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如：

```

/* configure GPIO pin : PF6 */
hal_gpio_init_struct gpio_init_parameter;
gpio_init_parameter.mode = GPIO_MODE_OUTPUT_PP;
gpio_init_parameter.pull = GPIO_PULL_NONE;
gpio_init_parameter.ospeed = GPIO_OSPEED_50MHZ;
gpio_init_parameter.af = HAL_GPIO_AF_0;
hal_gpio_init(GPIOF, GPIO_PIN_6, &gpio_init_parameter);

```

函数 hal_gpio_bit_set

函数hal_gpio_bit_set描述见下表：

表 3-248. 函数 hal_gpio_bit_set

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

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set PA0*/
hal_gpio_bit_set (GPIOA, GPIO_PIN_0);
```

函数 hal_gpio_bit_reset

函数hal_gpio_bit_reset描述见下表:

表 3-249. 函数 gpio_bit_reset

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

例如:

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

函数 hal_gpio_struct_init

函数hal_gpio_struct_init描述见下表:

表 3-250. 函数 hal_gpio_struct_init

函数名称	hal_gpio_struct_init
函数原型	int32_t hal_gpio_struct_init(hal_gpio_init_struct *p_init);
功能描述	将GPIO初始化结构体中所有参数初始化为默认值
先决条件	-

被调用函数	-
输入参数{in}	
p_init	指向一个已经定义的GPIO初始化结构体的指针
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如:

```
/* initialize the GPIO initialization structure */
hal_gpio_init_struct gpio_init_parameter;
hal_gpio_struct_init(&gpio_init_parameter);
```

函数 hal_gpio_deinit

函数hal_gpio_deinit t描述见下表:

表 3-251. 函数 hal_gpio_deinit

函数名称	hal_gpio_deinit
函数原型	int32_t hal_gpio_deinit(uint32_t gpio_periph, uint32_t pin);
功能描述	复位GPIOx PINx
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	GPIOx(x = A,B,C,D,F)
输入参数{in}	
pin	PIN引脚
GPIO_PIN_x	GPIO_PIN_x(x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_VAL

例如:

```
/* deinitialize GPIO pin : PB6 */
hal_gpio_deinit(GPIOB, GPIO_PIN_6);
```

3.14. I2C

I2C（内部集成电路总线）模块提供了符合工业标准的两线串行制接口，可用于MCU和外部I2C设备的通讯。章节[3.14.1](#)描述了I2C的寄存器列表，章节[3.14.2](#)对I2C库函数进行说明。

3.14.1. 外设寄存器说明

I2C寄存器列表如下表所示：

表 3-252. I2C 寄存器

寄存器名称	寄存器描述
I2C_CTL0	控制寄存器0
I2C_CTL1	控制寄存器1
I2C_SADDR0	从机地址寄存器0
I2C_SADDR1	从机地址寄存器1
I2C_DATA	传输缓冲区寄存器
I2C_STAT0	传输状态寄存器0
I2C_STAT1	传输状态寄存器1
I2C_CKCFG	时钟配置寄存器
I2C_RT	上升时间寄存器
I2C_FMPCFG	快速+模式配置寄存器

3.14.2. 外设库函数说明

I2C库函数列表如下表所示：

表 3-253. I2C 库函数

库函数名称	库函数描述
hal_i2c_struct_init	初始化I2C结构体
hal_i2c_deinit	复位I2C外设
hal_i2c_init	初始化I2C模块
hal_i2c_error_irq	I2C错误中断处理
hal_i2c_event_irq	I2C事件中断处理
hal_i2c_start	启动I2C模块函数
hal_i2c_stop	停止I2C模块函数
hal_i2c_irq_handle_set	设置用户定义的中断回调函数
hal_i2c_irq_handle_all_reset	重置所有用户定义的中断回调函数
hal_i2c_master_transmit_poll	轮询方式实现I2C主机发送
hal_i2c_master_receive_poll	轮询方式实现I2C主机接收
hal_i2c_slave_transmit_poll	轮询方式实现I2C从机发送
hal_i2c_slave_receive_poll	轮询方式实现I2C从机接收
hal_i2c_memory_write_poll	轮询方式实现I2C写存储器
hal_i2c_memory_read_poll	轮询方式实现I2C读存储器
hal_i2c_master_transmit_interrupt	中断方式实现I2C主机发送
hal_i2c_master_receive_interrupt	中断方式实现I2C主机接收
hal_i2c_slave_transmit_interrupt	中断方式实现I2C从机发送
hal_i2c_slave_receive_interrupt	中断方式实现I2C从机接收
hal_i2c_memory_write_interrupt	中断方式实现I2C写存储器

库函数名称	库函数描述
hal_i2c_memory_read_interrupt	中断方式实现I2C读存储器
hal_i2c_master_transmit_dma	DMA方式实现I2C主机发送
hal_i2c_master_receive_dma	DMA方式实现I2C主机接收
hal_i2c_slave_transmit_dma	DMA方式实现I2C从机发送
hal_i2c_slave_receive_dma	DMA方式实现I2C从机接收
hal_i2c_memory_write_dma	DMA方式实现I2C写存储器
hal_i2c_memory_read_dma	DMA方式实现I2C读存储器
hal_i2c_device_ready_check	检查I2C设备是否准备就绪
hal_i2c_master_serial_transmit_interrupt	中断方式实现I2C主机连续发送
hal_i2c_master_serial_receive_interrupt	中断方式实现I2C主机连续接收
hal_i2c_slave_serial_transmit_interrupt	中断方式实现I2C从机连续发送
hal_i2c_slave_serial_receive_interrupt	中断方式实现I2C从机连续接收
hal_i2c_address_listen_interrupt_enable	从机模式下使能地址监听
hal_i2c_address_listen_interrupt_disable	从机模式下禁能地址监听

枚举 i2c_flag_enum

表 3-254. 枚举 i2c_flag_enum

成员名称	功能描述
I2C_FLAG_SBSEND	主机模式下发送START起始位
I2C_FLAG_ADDSENT	主机模式下成功发送了地址，从机模式下接收到了地址并且和自身的地址匹配
I2C_FLAG_BTC	字节发送结束中断使能
I2C_FLAG_ADD10SEND	主机模式下10位地址的地址头被发送
I2C_FLAG_STPDET	从机模式下监测到STOP结束位
I2C_FLAG_RBNE	接收期间I2C_DATA非空
I2C_FLAG_TBNE	发送期间I2C_DATA为空
I2C_FLAG_BERR	总线错误，表示I2C总线上发生了预料之外的START起始位或STOP结束位
I2C_FLAG_LOSTARB	主机模式下仲裁丢失
I2C_FLAG_AERR	应答错误
I2C_FLAG_OUERR	从机接收模式下，发生了过载或欠载事件
I2C_FLAG_PECERR	接收数据时PEC错误
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_SMBALT	SMBus警报状态
I2C_FLAG_MASTER	表明I2C时钟在主机模式还是从机模式的标志位
I2C_FLAG_I2CBSY	忙标志

I2C_FLAG_TR	I2C作发送端还是接收端
I2C_FLAG_RXGC	I2C作发送端还是接收端
I2C_FLAG_DEFSMB	从机模式下SMBus主机地址头
I2C_FLAG_HSTSMB	从机模式下监测到SMBus主机地址头
I2C_FLAG_DUMOD	从机模式下双标志位表明哪个地址和双地址模式匹配

枚举 i2c_interrupt_flag_enum

表 3-255. 枚举 i2c_interrupt_flag_enum

成员名称	功能描述
I2C_INT_FLAG_SB SEND	主机模式下发送START起始位中断标志
I2C_INT_FLAG_AD DSEND	主机模式下成功发送了地址，从机模式下接收到了地址并且和自身的地址匹配中断标志
I2C_INT_FLAG_BT C	字节发送结束
I2C_INT_FLAG_AD D10SEND	主机模式下10位地址的地址头被发送中断标志
I2C_INT_FLAG_ST PDET	从机模式下监测到STOP结束位中断标志
I2C_INT_FLAG_RB NE	接收期间I2C_DATA非空中断标志
I2C_INT_FLAG_TB E	发送期间I2C_DATA为空中断标志
I2C_INT_FLAG_BE RR	总线错误，表示I2C总线上发生了预料之外的START起始位或STOP结束位中断标志
I2C_INT_FLAG_LO STARB	主机模式下仲裁丢失中断标志
I2C_INT_FLAG_AE RR	应答错误中断标志
I2C_INT_FLAG_OU ERR	从机接收模式下，发生了过载或欠载事件中断标志
I2C_INT_FLAG_PE CERR	接收数据时PEC错误中断标志
I2C_INT_FLAG_SM BTO	SMBus模式下超时信号中断标志
I2C_INT_FLAG_SM BALT	SMBus警报状态中断标志

枚举 i2c_interrupt_enum

表 3-256. 枚举 i2c_interrupt_enum

成员名称	功能描述
I2C_INT_ERR	错误中断使能

I2C_INT_EV	事件中断使能
I2C_INT_BUF	缓冲区中断使能

枚举 hal_i2c_struct_type_enum

表 3-257. 枚举 hal_i2c_struct_type_enum

成员名称	功能描述
HAL_I2C_INIT_STRUCTURE	I2C初始化结构
HAL_I2C_DEV_STRUCTURE	I2C设备信息结构
HAL_I2C_IRQ_STRUCTURE	I2C设备中断回调函数指针结构

枚举 hal_i2c_run_state_enum

表 3-258. 枚举 hal_i2c_run_state_enum

成员名称	功能描述
HAL_I2C_STATE_READY	I2C准备好使用状态
HAL_I2C_STATE_BUSY	I2C正在进行传输状态
HAL_I2C_STATE_MEMORY_BUSY_TX	I2C正在进行写memory
HAL_I2C_STATE_MEMORY_BUSY_RX	I2C正在进行读memory
HAL_I2C_STATE_LISTEN	I2C正在从机模式下寻址监听
HAL_I2C_STATE_BUSY_LISTEN	I2C正在寻址监听和数据传输

结构体 i2c_buffer_struct

表 3-259. 结构体 i2c_buffer_struct

成员名称	功能描述
buffer	指向传输缓冲区的指针
length	传输缓冲区长度
pos	传输缓冲区位置

结构体 hal_i2c_irq_struct

表 3-260. 结构体 hal_i2c_irq_struct

成员名称	功能描述
event_handle	事件中断回调函数
error_handle	错误中断回调函数

结构体 hal_i2c_slave_address_struct

表 3-261. 结构体 hal_i2c_slave_address_struct

成员名称	功能描述
device_address	设备地址
memory_address	存储器地址
address_size	存储器地址宽度
address_complete	寻址完成标志
address_count	寻址次数
second_addressing	第二次寻址标志

结构体 hal_i2c_dev_struct

表 3-262. 结构体 hal_i2c_dev_struct

成员名称	功能描述
periph	I2C外设
i2c_irq	中断回调函数
p_dma_rx	DMA接收指针
p_dma_tx	DMA发送指针
txbuffer	发送缓冲区
rxbuffer	接收缓冲区
slave_address	设备地址
transfer_option	发送方向
last_error	最新产生的错误状态
error_state	错误状态
tx_state	发送状态
rx_state	接收状态
previous_state	前一个传输状态
rx_callback	接收回调函数
tx_callback	发送回调函数
mutex	互斥锁和解锁状态

结构体 hal_i2c_init_struct

表 3-263. 结构体 hal_i2c_init_struct

成员名称	功能描述
duty_cycle	在快速/快速+模式下占空比

成员名称	功能描述
clock_speed	时钟速度
address_format	地址格式，7位地址或10地址
own_address1	I2C自身地址
dual_address	双地址模式使能
own_address2	I2C第二个自身地址，在双地址模式开启时有效
general_call	广播呼叫使能
no_stretch	禁止时钟延长使能

函数 hal_i2c_struct_init

函数hal_i2c_struct_init描述见下表：

表 3-264. 函数 hal_i2c_struct_init

函数名称	hal_i2c_struct_init
函数原型	void hal_i2c_struct_init(hal_i2c_struct_type_enum struct_type, void *p_struct);
功能描述	初始化I2C结构体
先决条件	-
输入参数{in}	
struct_type	结构体类型
HAL_I2C_INIT_STR UCT	I2C初始化结构体
HAL_I2C_DEV_ST RUCT	I2C设备信息结构体
HAL_I2C_IRQ_STR UCT	I2C中断回调函数结构体
输入参数{in}	
p_struct	指向包含配置信息的I2C结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the I2C structure with the default values */
```

```
hal_i2c_init_struct i2c0_init_parameter;
```

```
hal_i2c_struct_init(HAL_I2C_INIT_STRUCTURE, &i2c0_init_parameter);
```

函数 hal_i2c_deinit

函数hal_i2c_deinit描述见下表：

表 3-265. 函数 hal_i2c_deinit

函数名称	hal_i2c_deinit
------	----------------

函数原型	void hal_i2c_deinit(hal_i2c_dev_struct *i2c_dev);
功能描述	复位外设I2Cx
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体， 表 3-262. 结构体hal_i2c_dev_struct。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize I2C */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
hal_i2c_deinit(&i2c0_info);
```

函数 hal_i2c_init

函数hal_i2c_init描述见下表：

表 3-266. 函数 hal_i2c_init

函数名称	hal_i2c_init
函数原型	int32_t hal_i2c_init(hal_i2c_dev_struct *i2c_dev, uint32_t periph, hal_i2c_init_struct *p_init)
功能描述	初始化I2C
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct。
输入参数{in}	
periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
p_init	I2C初始化结构体，结构体成员参考 表 3-263. 结构体hal_i2c_init_struct。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_VAL/HAL_ERR_ADDRESS

例如：

```
/* initialize I2C */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
hal_i2c_init_struct i2c0_init_parameter;
```

```

i2c0_init_parameter.duty_cycle = I2C_DTCY_2;

i2c0_init_parameter.clock_speed = 400000;

i2c0_init_parameter.address_format = I2C_ADDFORMAT_7BITS;

i2c0_init_parameter.own_address1 = 0x82;

i2c0_init_parameter.dual_address = I2C_DUADEN_DISABLE;

i2c0_init_parameter.own_address2 = 0x00;

i2c0_init_parameter.general_call = I2C_GCEN_DISABLE;

i2c0_init_parameter.no_stretch = I2C_SCLSTRETCH_DISABLE;

hal_i2c_init(&i2c0_info, I2C0, &i2c0_init_parameter);

```

函数 hal_i2c_error_irq

函数hal_i2c_error_irq描述见下表：

表 3-267. 函数 hal_i2c_error_irq

函数名称	hal_i2c_error_irq
函数原型	void hal_i2c_error_irq(hal_i2c_dev_struct *i2c_dev);
功能描述	I2C错误中断处理
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* I2C error interrupt handler*/

hal_i2c_dev_struct i2c0_info;

hal_i2c_error_irq(&i2c0_info);

```

函数 hal_i2c_event_irq

函数hal_i2c_event_irq描述见下表：

表 3-268. 函数 hal_i2c_event_irq

函数名称	hal_i2c_event_irq
函数原型	void hal_i2c_event_irq(hal_i2c_dev_struct *i2c_dev);
功能描述	I2C事件中断处理
先决条件	-

输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C event interrupt handler */

hal_i2c_dev_struct i2c0_info;

hal_i2c_event_irq(&i2c0_info);
```

函数 hal_i2c_start

函数hal_i2c_start描述见下表：

表 3-269. 函数 hal_i2c_start

函数名称	hal_i2c_start
函数原型	void hal_i2c_start(hal_i2c_dev_struct *i2c_dev);
功能描述	启动I2C模块函数
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* start I2C module function */

hal_i2c_dev_struct i2c0_info;

hal_i2c_start(&i2c0_info);
```

函数 hal_i2c_stop

函数hal_i2c_stop描述见下表：

表 3-270. 函数 hal_i2c_stop

函数名称	hal_i2c_stop
函数原型	void hal_i2c_stop(hal_i2c_dev_struct *i2c_dev);
功能描述	停止I2C模块函数
先决条件	-
输入参数{in}	

i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stop I2C module function */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
hal_i2c_stop(&i2c0_info);
```

函数 hal_i2c_irq_handle_set

函数hal_i2c_irq_handle_set描述见下表：

表 3-271. 函数 hal_i2c_irq_handle_set

函数名称	hal_i2c_irq_handle_set
函数原型	void hal_i2c_irq_handle_set(hal_i2c_dev_struct *i2c_dev, hal_i2c_irq_struct *p_irq);
功能描述	设置用户自定义的中断回调函数，当相应的中断被触发时，该函数将被注册并调用
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_irq	I2C中断回调函数结构体，结构体成员参考 表 3-260. 结构体hal_i2c_irq_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set user-defined interrupt callback function */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
hal_i2c_irq_struct i2c0_irq_parameter;
```

```
hal_i2c_irq_handle_set(&i2c0_info, &i2c0_irq_parameter);
```

函数 hal_i2c_irq_handle_all_reset

函数hal_i2c_irq_handle_all_reset描述见下表：

表 3-272. 函数 hal_i2c_irq_handle_all_reset

函数名称	hal_i2c_irq_handle_all_reset
函数原型	void hal_i2c_irq_handle_all_reset(hal_i2c_dev_struct *i2c_dev);
功能描述	重置所有用户自定义的中断回调函数，当相应的中断被触发时，该函数将被注册并调用
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback function */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
hal_i2c_irq_handle_all_reset(&i2c0_info);
```

函数 hal_i2c_master_transmit_poll

函数hal_i2c_master_transmit_poll描述见下表：

表 3-273. 函数 hal_i2c_master_transmit_poll

函数名称	hal_i2c_master_transmit_poll
函数原型	int32_t hal_i2c_master_transmit_poll(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	轮询方式实现I2C主机发送
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	发送缓冲区
输入参数{in}	
length	发送数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUFFER

例如：

```
/* transmit amounts of data in master mode, poll transmit process and completed status */
hal_i2c_dev_struct i2c0_info;

uint8_t tx_buffer[] = "I2C communication based on polling";

#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))

hal_i2c_master_transmit_poll(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE, 10000);
```

函数 hal_i2c_master_receive_poll

函数hal_i2c_master_receive_poll描述见下表：

表 3-274. 函数 hal_i2c_master_receive_poll

函数名称	hal_i2c_master_receive_poll
函数原型	int32_t hal_i2c_master_receive_poll(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	在主模式下接收数据，轮询接收过程和完成状态
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUFFER_FULL

例如：

```
/* receive amounts of data in master mode, poll receive process and completed status */
hal_i2c_dev_struct i2c0_info;

#define RXBUFFERSIZE 10

uint8_t rx_buffer[RXBUFFERSIZE];

hal_i2c_master_receive_poll(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE, 10000);
```

函数 hal_i2c_slave_transmit_poll

函数hal_i2c_slave_transmit_poll描述见下表：

表 3-275. 函数 hal_i2c_slave_transmit_poll

函数名称	hal_i2c_slave_transmit_poll
函数原型	int32_t hal_i2c_slave_transmit_poll(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	在从机模式下传输数据量，轮询传输过程和完成状态
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	发送缓冲区
输入参数{in}	
length	发送数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如：

```
/* transmit amounts of data in slave mode, poll transmit process and completed status */
hal_i2c_dev_struct i2c0_info;

uint8_t tx_buffer[] = "I2C communication based on polling";

#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))

hal_i2c_slave_transmit_poll(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE, 10000);
```

函数 hal_i2c_slave_receive_poll

函数hal_i2c_slave_receive_poll描述见下表：

表 3-276. 函数 hal_i2c_slave_receive_poll

函数名称	hal_i2c_slave_receive_poll
函数原型	int32_t hal_i2c_slave_receive_poll(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	轮询方式实现I2C从机接收

先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUFFER

例如：

```
/* receive amounts of data in slave mode, poll receive process and completed status */
hal_i2c_dev_struct i2c0_info;

#define RXBUFFERSIZE 10

uint8_t rx_buffer[RXBUFFERSIZE];

hal_i2c_slave_receive_poll(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE, 10000);
```

函数 hal_i2c_memory_write_poll

函数hal_i2c_memory_write_poll描述见下表：

表 3-277. 函数 hal_i2c_memory_write_poll

函数名称	hal_i2c_memory_write_poll
函数原型	int32_t hal_i2c_memory_write_poll(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	轮询方式实现I2C写存储器
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct。
输入参数{in}	
p_buffer	待写数据缓冲区
输入参数{in}	
length	待写数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	

-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_VA L

例如:

```
/* write amounts of data to memory, poll transmit process and completed status */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
uint8_t tx_buffer[] = "I2C write memory test";
```

```
#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))
```

```
hal_i2c_master_transmit_poll(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE, 10000);
```

注意: 用户需根据存储器特性进行写操作, 如AT24C02, 只能进行字节写或者页(8字节)写, 此时tx_buffer只能小于或者等于8字节, 若需写大于8字节需要进行分页后再调用hal_i2c_master_transmit_poll进行写操作。

函数 hal_i2c_memory_read_poll

函数hal_i2c_memory_read_poll描述见下表:

表 3-278. 函数 hal_i2c_memory_read_poll

函数名称	hal_i2c_memory_read_poll
函数原型	int32_t hal_i2c_memory_read_poll(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	从存储器中读取数据量, 轮询接收进程和完成状态
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体, 结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_VA L

例如:

```
/* read amounts of data from memory, poll receive process and completed status */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
#define RXBUFFERSIZE 10
```

```
uint8_t rx_buffer[RXBUFFERSIZE];
```

```
hal_i2c_memory_read_poll(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE, 10000);
```

函数 hal_i2c_master_transmit_interrupt

函数hal_i2c_master_transmit_interrupt描述见下表：

表 3-279. 函数 hal_i2c_master_transmit_interrupt

函数名称	hal_i2c_master_transmit_interrupt
函数原型	int32_t hal_i2c_master_transmit_interrupt(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	中断方式实现I2C主机发送
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	发送缓冲区
输入参数{in}	
length	发送数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如：

```
/* transmit amounts of data in master mode by interrupt method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
uint8_t tx_buffer[] = "I2C communication based on interrupt";
```

```
#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))
```

```
hal_i2c_master_transmit_interrupt(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE, &fun_user);
```

函数 hal_i2c_master_receive_interrupt

函数hal_i2c_master_receive_interrupt描述见下表：

表 3-280. 函数 hal_i2c_master_receive_interrupt

函数名称	hal_i2c_master_receive_interrupt
函数原型	int32_t hal_i2c_master_receive_interrupt(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	中断方式实现I2C主机接收
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如：

```
/* receive amounts of data in master mode by interrupt method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
#define RXBUFFERSIZE 10
```

```
uint8_t rx_buffer[RXBUFFERSIZE];
```

```
hal_i2c_master_receive_interrupt(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE, &fun_user);
```

函数 hal_i2c_slave_transmit_interrupt

函数hal_i2c_slave_transmit_interrupt描述见下表：

表 3-281. 函数 hal_i2c_slave_transmit_interrupt

函数名称	hal_i2c_slave_transmit_interrupt
函数原型	int32_t hal_i2c_slave_transmit_interrupt(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	中断方式实现I2C从机发送

先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct。
输入参数{in}	
p_buffer	发送缓冲区
输入参数{in}	
length	发送数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如：

```
/* transmit amounts of data in slave mode by interrupt method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
uint8_t tx_buffer[] = "I2C communication based on interrupt";
```

```
#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))
```

```
hal_i2c_slave_transmit_interrupt(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE, &fun_user);
```

函数 hal_i2c_slave_receive_interrupt

函数hal_i2c_slave_receive_interrupt描述见下表：

表 3-282. 函数 hal_i2c_slave_receive_interrupt

函数名称	hal_i2c_slave_receive_interrupt
函数原型	int32_t hal_i2c_slave_receive_interrupt(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	中断方式实现I2C从机接收
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	

p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUFFER_OVERFLOW

例如:

```
/* receive amounts of data in slave mode by interrupt method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
#define RXBUFFERSIZE 10
```

```
uint8_t rx_buffer[RXBUFFERSIZE];
```

```
hal_i2c_slave_receive_interrupt(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE,
&fun_user);
```

函数 hal_i2c_memory_write_interrupt

函数hal_i2c_memory_write_interrupt描述见下表:

表 3-283. 函数 hal_i2c_memory_write_interrupt

函数名称	hal_i2c_memory_write_interrupt
函数原型	int32_t hal_i2c_memory_write_interrupt(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_callback p_user_func);
功能描述	中断方式实现I2C写存储器
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体, 结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	待写数据缓冲区
输入参数{in}	
length	待写数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT, HAL_ERR_VALUE

例如:


```
/* write amounts of data to memory by interrupt method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
uint8_t tx_buffer[] = "I2C write memory test";
```

```
#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))
```

```
hal_i2c_memory_write_interrupt(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE,
&fun_user);
```

注意：用户需根据存储器特性进行写操作，如AT24C02，只能进行字节写或者页（8字节）写，此时tx_buffer只能小于或者等于8字节，若需写大于8字节需要进行分页后再调用hal_i2c_memory_write_interrupt进行写操作。

函数 hal_i2c_memory_read_interrupt

函数hal_i2c_memory_read_interrupt描述见下表：

表 3-284. 函数 hal_i2c_memory_read_interrupt

函数名称	hal_i2c_memory_read_interrupt
函数原型	int32_t hal_i2c_memory_read_interrupt(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	中断方式实现I2C读存储器
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT, HAL_ERR_VAL

例如：

```
/* read amounts of data from memory by interrupt method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
#define RXBUFFERSIZE 10
```

```
uint8_t rx_buffer[RXBUFFERSIZE];
```

```
hal_i2c_memory_read_interrupt(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE,
&fun_user);
```

函数 hal_i2c_master_transmit_dma

函数hal_i2c_master_transmit_dma描述见下表：

表 3-285. 函数 hal_i2c_master_transmit_dma

函数名称	hal_i2c_master_transmit_dma
函数原型	int32_t hal_i2c_master_transmit_dma(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	DMA方式实现I2C主机发送
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	发送缓冲区
输入参数{in}	
length	发送数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如：

```
/* transmit amounts of data in master mode by dma method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
uint8_t tx_buffer[] = "I2C communication based on interrupt";
```

```
#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))
```

```
hal_i2c_master_transmit_dma(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE, &fun_user);
```

函数 hal_i2c_master_receive_dma

函数hal_i2c_master_receive_dma描述见下表：

表 3-286. 函数 hal_i2c_master_receive_dma

函数名称	hal_i2c_master_receive_dma
------	----------------------------

函数原型	int32_t hal_i2c_master_receive_dma(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	DMA方式实现I2C主机接收
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如：

```
/* receive amounts of data in master mode by dma method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
#define RXBUFFERSIZE 10
```

```
uint8_t rx_buffer[RXBUFFERSIZE];
```

```
hal_i2c_master_receive_dma(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE, &fun_user);
```

函数 hal_i2c_slave_transmit_dma

函数hal_i2c_slave_transmit_dma描述见下表：

表 3-287. 函数 hal_i2c_slave_transmit_dma

函数名称	hal_i2c_slave_transmit_dma
函数原型	int32_t hal_i2c_slave_transmit_dma(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	DMA方式实现I2C从机发送
先决条件	
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	发送缓冲区
输入参数{in}	

length	发送数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如:

```
/* transmit amounts of data in slave mode by dma method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
uint8_t tx_buffer[] = "I2C communication based on interrupt";
```

```
#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))
```

```
hal_i2c_slave_transmit_dma(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE, &fun_user);
```

函数 hal_i2c_slave_receive_dma

函数hal_i2c_slave_receive_dma描述见下表:

表 3-288. 函数 hal_i2c_slave_receive_dma

函数名称	hal_i2c_slave_receive_dma
函数原型	int32_t hal_i2c_slave_receive_dma(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	DMA方式实现I2C从机接收
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体, 结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如:

```
/* receive amounts of data in slave mode by dma method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
#define RXBUFFERSIZE 10
```

```
uint8_t rx_buffer[RXBUFFERSIZE];
```

```
hal_i2c_slave_receive_dma(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE, &fun_user);
```

函数 hal_i2c_memory_write_dma

函数hal_i2c_memory_write_dma描述见下表：

表 3-289. 函数 hal_i2c_memory_write_dma

函数名称	hal_i2c_memory_write_dma
函数原型	int32_t hal_i2c_memory_write_dma(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	DMA方式实现I2C写存储器
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	待写数据缓冲区
输入参数{in}	
length	待写数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT, HAL_ERR_VAL

例如：

```
/* write amounts of data to memory by dma method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
uint8_t tx_buffer[] = "I2C write memory test";
```

```
#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))
```

```
hal_i2c_memory_write_dma(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE, &fun_user);
```

注意：用户需根据存储器特性进行写操作，如AT24C02，只能进行字节写或者页（8字节）写，

此时tx_buffer只能小于或者等于8字节，若需写大于8字节需要进行分页后再调用hal_i2c_memory_write_dma进行写操作。

函数 hal_i2c_memory_read_dma

函数hal_i2c_memory_read_dma描述见下表：

表 3-290. 函数 hal_i2c_memory_read_dma

函数名称	hal_i2c_memory_read_dma
函数原型	int32_t hal_i2c_memory_read_dma(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	DMA方式实现I2C读存储器
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT, HAL_ERR_VAL

例如：

```

/* read amounts of data from memory by dma method */

hal_i2c_dev_struct i2c0_info;

void fun_user(hal_i2c_dev_struct *i2c_dev);

#define RXBUFFERSIZE 10

uint8_t rx_buffer[RXBUFFERSIZE];

hal_i2c_memory_read_dma(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE, &fun_user);

```

函数 hal_i2c_device_ready_check

函数hal_i2c_device_ready_check描述见下表：

表 3-291. 函数 hal_i2c_device_ready_check

函数名称	hal_i2c_device_ready_check
函数原型	int32_t hal_i2c_device_ready_check(hal_i2c_dev_struct *i2c_dev, uint32_t

	timeout_ms);
功能描述	检查I2C设备是否准备就绪
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct。
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_NONE

例如：

```
/* check whether the device is ready for access */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
hal_i2c_device_ready_check(&i2c0_info, 1000);
```

函数 hal_i2c_master_serial_transmit_interrupt

函数hal_i2c_master_serial_transmit_interrupt描述见下表：

表 3-292. 函数 hal_i2c_master_serial_transmit_interrupt

函数名称	hal_i2c_master_serial_transmit_interrupt
函数原型	int32_t hal_i2c_master_serial_transmit_interrupt(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	中断方式实现I2C主机连续发送
先决条件	
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct。
输入参数{in}	
p_buffer	发送缓冲区
输入参数{in}	
length	发送数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如：

```
/* serial transmit amounts of data in master mode by interrupt method */
```

```

hal_i2c_dev_struct i2c0_info;

void fun_user(hal_i2c_dev_struct *i2c_dev);

uint8_t tx_buffer[] = "I2C serial communication";

#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))

hal_i2c_master_serial_transmit_interrupt(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE,
&fun_user);

```

函数 hal_i2c_master_serial_receive_interrupt

函数hal_i2c_master_serial_receive_interrupt描述见下表：

表 3-293. 函数 hal_i2c_master_serial_receive_interrupt

函数名称	hal_i2c_master_serial_receive_interrupt
函数原型	int32_t hal_i2c_master_serial_receive_interrupt(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	中断方式实现I2C主机连续接收
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考表 3-262. 结构体hal_i2c_dev_struct。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如：

```

/* serial receive amounts of data in master mode by interrupt method */

hal_i2c_dev_struct i2c0_info;

void fun_user(hal_i2c_dev_struct *i2c_dev);

#define RXBUFFERSIZE 10

uint8_t rx_buffer[RXBUFFERSIZE];

hal_i2c_master_serial_receive_interrupt(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE,
&fun_user);

```


函数 hal_i2c_slave_serial_transmit_interrupt

函数hal_i2c_slave_serial_transmit_interrupt描述见下表:

表 3-294. 函数 hal_i2c_slave_serial_transmit_interrupt

函数名称	hal_i2c_slave_serial_transmit_interrupt
函数原型	int32_t hal_i2c_slave_serial_transmit_interrupt(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	中断方式实现I2C从机连续发送
先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体, 结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输入参数{in}	
p_buffer	发送缓冲区
输入参数{in}	
length	发送数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如:

```
/* serial transmit amounts of data in slave mode by interrupt method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
uint8_t tx_buffer[] = " I2C serial communication";
```

```
#define TXBUFFERSIZE (uint32_t)(sizeof(tx_buffer) / sizeof(*(tx_buffer)))
```

```
hal_i2c_slave_serial_transmit_interrupt(&i2c0_info, (uint8_t*)tx_buffer, TXBUFFERSIZE, &fun_user);
```

函数 hal_i2c_slave_serial_receive_interrupt

函数hal_i2c_slave_serial_receive_interrupt描述见下表:

表 3-295. 函数 hal_i2c_slave_serial_receive_interrupt

函数名称	hal_i2c_slave_serial_receive_interrupt
函数原型	int32_t hal_i2c_slave_serial_receive_interrupt(hal_i2c_dev_struct *i2c_dev, uint8_t *p_buffer, uint32_t length, hal_i2c_user_cb p_user_func);
功能描述	中断方式实现I2C从机连续接收

先决条件	-
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct。
输入参数{in}	
p_buffer	接收缓冲区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_ADDRESS/HAL_ERR_TIMEOUT/HAL_ERR_BUSY

例如：

```
/* serial receive amounts of data in slave mode by interrupt method */
```

```
hal_i2c_dev_struct i2c0_info;
```

```
void fun_user(hal_i2c_dev_struct *i2c_dev);
```

```
#define RXBUFFERSIZE 10
```

```
uint8_t rx_buffer[RXBUFFERSIZE];
```

```
hal_i2c_slave_serial_receive_interrupt(&i2c0_info, (uint8_t*)rx_buffer, RXBUFFERSIZE,
&fun_user);
```

函数 hal_i2c_address_listen_interrupt_enable

函数hal_i2c_address_listen_interrupt_enable描述见下表：

表 3-296. 函数 hal_i2c_address_listen_interrupt_enable

函数名称	hal_i2c_address_listen_interrupt_enable
函数原型	int32_t hal_i2c_address_listen_interrupt_enable(hal_i2c_dev_struct *i2c_dev, hal_i2c_user_cb p_user_func);
功能描述	从机模式下使能地址监听
先决条件	
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct。
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	

int32_t	HAL_ERR_NONE/HAL_ERR_BUSY/HAL_ERR_ADDRESS
---------	---

例如：

```
/* enable address listen in slave mode by interrupt method */

hal_i2c_dev_struct i2c0_info;

void fun_user(hal_i2c_dev_struct *i2c_dev);

hal_i2c_address_listen_interrupt_enable(&i2c0_info, &fun_user);
```

函数 hal_i2c_address_listen_interrupt_disable

函数hal_i2c_address_listen_interrupt_disable描述见下表：

表 3-297. 函数 hal_i2c_address_listen_interrupt_disable

函数名称	hal_i2c_address_listen_interrupt_disable
函数原型	int32_t hal_i2c_address_listen_interrupt_disable(hal_i2c_dev_struct *i2c_dev);
功能描述	从机模式下禁能地址监听
先决条件	
输入参数{in}	
i2c_dev	I2C设备信息结构体，结构体成员参考 表 3-262. 结构体hal_i2c_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE/HAL_ERR_BUSY/HAL_ERR_ADDRESS

例如：

```
/* disable address listen in slave mode by interrupt method */

hal_i2c_dev_struct i2c0_info;

hal_i2c_address_listen_interrupt_disable(&i2c0_info);
```

3.15. SMBUS

SMBus系统管理总线（System Management Bus，简称为SMBus或SMB）是一种结构简单的单端双线制总线，属于I2C的一种衍生总线形式，可实现轻量级的通信需求。一般来说，SMBus最常见于计算机主板，主要用于电源传输ON/OFF指令的通信。章节[3.15.1](#)描述了I2C的寄存器列表，章节[3.15.2](#)对SMBUS库函数进行说明。

3.15.1. 外设寄存器说明

I2C-SMBUS寄存器列表如下表所示：

表 3-298. I2C 寄存器

寄存器名称	寄存器描述
I2C_CTL0	控制寄存器0
I2C_CTL1	控制寄存器1
I2C_SADDR0	从机地址寄存器0
I2C_SADDR1	从机地址寄存器1
I2C_DATA	传输缓冲区寄存器
I2C_STAT0	传输状态寄存器0
I2C_STAT1	传输状态寄存器1
I2C_CKCFG	时钟配置寄存器
I2C_RT	上升时间寄存器
I2C_FMPCFG	快速+模式配置寄存器

3.15.2. 外设库函数说明

SMBUS库函数列表如下表所示：

表 3-299. SMBUS 库函数

库函数名称	库函数描述
hal_smbus_struct_init	结构体初始化
hal_smbus_deinit	去除初始化，使回到复位状态
hal_smbus_init	SMBUS初始化
hal_smbus_enable_alert_interrupt	报警模式使能并开启中断
hal_smbus_event_irq	SMBUS事件中断处理程序
hal_smbus_error_irq	SMBUS错误中断处理程序
hal_smbus_start	开启SMBUS模块
hal_smbus_stop	关闭SMBUS模块
hal_smbus_irq_handle_set	SMBUS用户自定义中断函数入口设置
hal_smbus_irq_handle_all_reset	SMBUS中断函数入口复位
hal_smbus_master_transmit_interrupt	主机发送中断模式
hal_smbus_master_receive_interrupt	主机接收中断模式
hal_smbus_slave_transmit_interrupt	从机发送中断模式
hal_smbus_slave_receive_interrupt	从机接收中断模式
hal_smbus_disable_alert_interrupt	报警模式关闭并关闭中断

枚举 hal_smbus_run_state_enum

Table 3-1. 枚举 hal_smbus_run_state_enum

成员名称	功能描述
HAL_SMBUS_STATE_READY	SMBUS准备状态
HAL_SMBUS_STATE_FREE	SMBUS空闲状态

HAL_SMBUS_STATE _BUSY	SMBUS忙碌状态
--------------------------	-----------

枚举 hal_smbus_struct_type_enum

Table 3-2. 枚举 hal_smbus_struct_type_enum

成员名称	功能描述
HAL_SMBUS_INIT_S TRUCT	SMBUS初始化结构
HAL_SMBUS_DEV_S TRUCT	SMBUS设备信息结构
HAL_SMBUS_IRQ_S TRUCT	SMBUS设备中断回调函数指针结构

结构体 hal_smbus_irq_struct

表 3-300.结构体 hal_smbus_irq_struct

成员名称	功能描述
event_handle	事件中断处理函数
error_handle	错误中断处理函数

结构体 smbus_buffer_struct

Table 3-3.结构体 smbus_buffer_struct

成员名称	功能描述
buffer	指向传输缓冲区的指针
length	传输缓冲区长度
pos	传输缓冲区位置

结构体 hal_smbus_init_struct

表 3-301.结构体 hal_smbus_init_struct

成员名称	功能描述
clock_speed	SMBUS通信速率
address_format	地址模式
own_address1	SMBUS地址
dual_address	双地址模式
own_address2	双地址模式下SMBUS地址2
smbus_type	SMBUS类型(SMBUS_DEVICE, SMBUS_HOST)
smbus_pec	PEC计算使能
smbus_arp	ARP地址协议使能
general_call	是否响应广播呼叫
no_stretch	在从机模式下数据未就绪是否将SCL拉低

结构体 `hal_smbus_dev_struct`

表 3-302. 结构体 `hal_smbus_dev_struct`

成员名称	功能描述
<code>periph</code>	SMBUS外设号
<code>slave_address</code>	从机地址
<code>smbus_irq</code>	SMBUS中断请求结构体
<code>txbuffer</code>	发送缓存结构体
<code>rxbuffer</code>	接收缓存结构体
<code>smbus_pec_transfer</code>	SMBUS的PEC传输
<code>last_error</code>	SMBUS最后产生的错误
<code>error_state</code>	SMBUS错误状态
<code>tx_state</code>	SMBUS发送错误状态
<code>rx_state</code>	SMBUS接收错误状态
<code>*rx_callback</code>	接收完成后，用户使用的回调函数
<code>*tx_callback</code>	发送完成后，用户使用的回调函数
<code>*priv</code>	用户自定义指针
<code>mutex</code>	互斥锁和解锁状态

函数 `hal_smbus_struct_init`

函数`hal_smbus_struct_init`描述见下表：

表 3-303. 函数 `hal_smbus_struct_init`

函数名称	<code>hal_smbus_struct_init</code>
函数原型	<code>void hal_smbus_struct_init(hal_smbus_struct_type_enum hal_struct_type, void *p_struct);</code>
功能描述	SMBUS相关结构体初始化
先决条件	-
输入参数{in}	
hal_struct_type	结构体类型
<code>HAL_SMBUS_INIT_STRUCT</code>	初始化结构体
<code>HAL_SMBUS_DEV_STRUCT</code>	设备信息结构体
<code>HAL_SMBUS_IRQ_STRUCT</code>	中断请求结构体
输入参数{in}	
p_struct	指向需要被初始化的结构体
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* init HAL_SMBUS_INIT_STRUCT */

hal_smbus_init_struct i2c0_smbus_init_parameter;

hal_smbus_struct_init(HAL_SMBUS_INIT_STRUCT, &i2c0_smbus_init_parameter);
```

函数 hal_smbus_deinit

函数hal_smbus_deinit描述见下表：

表 3-304. 函数 hal_smbus_deinit

函数名称	hal_smbus_deinit
函数原型	void hal_smbus_deinit(hal_smbus_dev_struct *smbus_dev);
功能描述	去除初始化，使回到复位状态
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize SMBUS */

hal_smbus_dev_struct i2c0_smbus_info;

hal_smbus_deinit(&i2c0_smbus_info);
```

函数 hal_smbus_init

函数hal_smbus_init描述见下表：

表 3-305. 函数 hal_smbus_init

函数名称	hal_smbus_init
函数原型	int32_t hal_smbus_init(hal_smbus_dev_struct *smbus_dev, uint32_t periph, hal_smbus_init_struct *p_init);
功能描述	SMBUS初始化
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输入参数{in}	
periph	指定哪一个SMBUS

输入参数{in}	
p_init	SMBUS初始化结构体，结构体成员参考 表 3-301.结构体 hal_smbus_init_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如：

```
/* initialize SMBUS registers */

hal_smbus_dev_struct i2c0_smbus_info;

hal_smbus_init_struct i2c0_smbus_init_parameter;

hal_smbus_init(&i2c0_smbus_info, I2C0, &i2c0_smbus_init_parameter);
```

函数 hal_smbus_enable_alert_interrupt

函数hal_smbus_enable_alert_interrupt描述见下表：

表 3-306. 函数 hal_smbus_slave_receive_interrupt

函数名称	hal_smbus_enable_alert_interrupt
函数原型	int32_t hal_smbus_enable_alert_interrupt(hal_smbus_dev_struct*smbus_dev);
功能描述	报警模式使能并开启中断
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输出参数{out}	
-	-
返回值	
error code	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* enable the SMBUS alert mode with Interrupt */

hal_smbus_dev_struct i2c0_smbus_info;

hal_smbus_enable_alert_interrupt(&i2c0_smbus_info);
```

函数 hal_smbus_event_irq

函数hal_smbus_event_irq描述见下表：

表 3-307. 函数 hal_smbus_event_irq

函数名称	hal_smbus_event_irq
------	---------------------

函数原型	void hal_smbus_event_irq(hal_smbus_dev_struct *smbus_dev);
功能描述	SMBus事件中中断处理程序
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 event interrupt */

hal_smbus_dev_struct i2c0_smbus_info;

void I2C0_EV_IRQHandler(void)
{
    hal_smbus_event_irq(&i2c0_smbus_info);
}
```

函数 hal_smbus_error_irq

函数hal_smbus_error_irq描述见下表：

表 3-308. 函数 hal_smbus_error_irq

函数名称	hal_smbus_error_irq
函数原型	void hal_smbus_error_irq(hal_smbus_dev_struct *smbus_dev);
功能描述	SMBus错误中断处理程序
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 error interrupt */

hal_smbus_dev_struct i2c0_smbus_info;

void I2C0_ER_IRQHandler(void)
```

```
{
    hal_smbus_error_irq(&i2c0_smbus_info);
}
```

函数 hal_smbus_start

函数hal_smbus_start描述见下表：

表 3-309. 函数 hal_smbus_start

函数名称	hal_smbus_start
函数原型	void hal_smbus_start(hal_smbus_dev_struct *smbus_dev);
功能描述	开启SMBUS模块
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* start SMBUS module function */
hal_smbus_dev_struct i2c0_smbus_info;
hal_smbus_start(&i2c0_smbus_info);
```

函数 hal_smbus_stop

函数hal_smbus_stop描述见下表：

表 3-310. 函数 hal_smbus_stop

函数名称	hal_smbus_stop
函数原型	void hal_smbus_stop(hal_smbus_dev_struct *smbus_dev);
功能描述	关闭SMBUS模块
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* stop SMBUS module function */

hal_smbus_dev_struct i2c0_smbus_info;

hal_smbus_stop(&i2c0_smbus_info);
```

函数 hal_smbus_irq_handle_set

函数hal_smbus_irq_handle_set描述见下表:

表 3-311. 函数 hal_smbus_irq_handle_set

函数名称	hal_smbus_irq_handle_set
函数原型	void hal_smbus_irq_handle_set(hal_smbus_dev_struct *smbus_dev, hal_smbus_irq_struct *p_irq);
功能描述	SMBUS用户自定义中断函数入口设置
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体, 结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输入参数{in}	
p_irq	SMBUS中断请求结构体, 结构体成员参考 表 3-300.结构体 hal_smbus_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

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

void _smbus_alert_event(void *smbus_dev)
{
    LED_ON;
}

i2c0_smbus_info.smbus_irq.error_handle = _smbus_alert_event;

hal_smbus_irq_handle_set(&i2c0_smbus_info, &i2c0_smbus_info.smbus_irq);
```

函数 hal_smbus_irq_handle_all_reset

函数hal_smbus_irq_handle_all_reset描述见下表:

表 3-312. 函数 hal_smbus_irq_handle_all_reset

函数名称	hal_smbus_irq_handle_all_reset
函数原型	void hal_smbus_irq_handle_all_reset(hal_smbus_dev_struct *smbus_dev);
功能描述	SMBUS中断函数入口复位
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback */
```

```
hal_smbus_dev_struct i2c0_smbus_info;
```

```
hal_smbus_irq_handle_all_reset(&i2c0_smbus_info);
```

函数 hal_smbus_master_transmit_interrupt

函数hal_smbus_master_transmit_interrupt描述见下表：

表 3-313. 函数 hal_smbus_master_transmit_interrupt

函数名称	hal_smbus_master_transmit_interrupt
函数原型	int32_t hal_smbus_master_transmit_interrupt(hal_smbus_dev_struct *smbus_dev, uint8_t *p_buffer, uint32_t length, hal_smbus_user_cb p_user_func)
功能描述	主机发送中断模式
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输入参数{in}	
p_buffer	指向发送的缓存区
输入参数{in}	
length	发送数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
error code	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_BUSY

例如：

```
/* SMBUS master transmit in interrupt mode */

hal_smbus_dev_struct i2c0_smbus_info;

buffertx[2] = {0, 0};

hal_smbus_master_transmit_interrupt(&i2c0_smbus_info, &buffertx, 2, NULL);
```

函数 hal_smbus_master_receive_interrupt

函数hal_smbus_master_receive_interrupt描述见下表：

表 3-314. 函数 hal_smbus_master_receive_interrupt

函数名称	hal_smbus_master_receive_interrupt
函数原型	int32_t hal_smbus_master_receive_interrupt(hal_smbus_dev_struct *smbus_dev, uint8_t *p_buffer, uint32_t length, hal_smbus_user_cb p_user_func)
功能描述	主机接收中断模式
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输入参数{in}	
p_buffer	指向接收的缓存区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
error code	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_BUSY, HAL_ERR_TIMEOUT

例如：

```
/* SMBUS master receive in interrupt mode */

hal_smbus_dev_struct i2c0_smbus_info;

uint8_t bufferrx[2];

hal_smbus_master_receive_interrupt(&i2c0_smbus_info, &bufferrx, 2, NULL);
```

函数 hal_smbus_slave_transmit_interrupt

函数hal_smbus_slave_transmit_interrupt描述见下表：

表 3-315. 函数 `hal_smbus_slave_transmit_interrupt`

函数名称	<code>hal_smbus_slave_transmit_interrupt</code>
函数原型	<code>int32_t hal_smbus_slave_transmit_interrupt(hal_smbus_dev_struct *smbus_dev, uint8_t *p_buffer, uint32_t length, hal_smbus_user_cb p_user_func)</code>
功能描述	从机发送中断模式
先决条件	-
输入参数{in}	
<code>smbus_dev</code>	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 <code>hal_smbus_dev_struct</code>
输入参数{in}	
<code>p_buffer</code>	指向发送的缓存区
输入参数{in}	
<code>length</code>	发送数据长度
输入参数{in}	
<code>p_user_func</code>	用户回调函数
输出参数{out}	
-	-
返回值	
<code>error code</code>	<code>HAL_ERR_NONE</code> , <code>HAL_ERR_ADDRESS</code> , <code>HAL_ERR_BUSY</code> , <code>HAL_ERR_TIMEOUT</code>

例如：

```
/* SMBUS slave transmit in interrupt mode */
hal_smbus_dev_struct i2c0_smbus_info;
buffertx[2] = {0, 0};
hal_smbus_slave_transmit_interrupt (&i2c0_smbus_info, &buffertx, 2, NULL);
```

函数 `hal_smbus_slave_receive_interrupt`

函数`hal_smbus_slave_receive_interrupt`描述见下表：

表 3-316. 函数 `hal_smbus_slave_receive_interrupt`

函数名称	<code>hal_smbus_slave_receive_interrupt</code>
函数原型	<code>int32_t hal_smbus_slave_receive_interrupt(hal_smbus_dev_struct *smbus_dev, uint8_t *p_buffer, uint32_t length, hal_smbus_user_cb p_user_func)</code>
功能描述	从机接收中断模式
先决条件	-
输入参数{in}	
<code>smbus_dev</code>	SMBUS设备信息结构体，结构体成员参考 表 3-302.结构体 <code>hal_smbus_dev_struct</code>

输入参数{in}	
p_buffer	指向接收的缓存区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
error code	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_BUSY, HAL_ERR_TIMEOUT

例如:

```
/* SMBUS slave receive in interrupt mode */
```

```
hal_smbus_dev_struct i2c0_smbus_info;
```

```
uint8_t bufferrx[2];
```

```
hal_smbus_slave_receive_interrupt(&i2c0_smbus_info, &buffertx, 2, NULL);
```

函数 hal_smbus_disable_alert_interrupt

函数hal_smbus_disable_alert_interrupt描述见下表:

表 3-317. 函数 hal_smbus_disable_alert_interrupt

函数名称	hal_smbus_disable_alert_interrupt
函数原型	int32_t hal_smbus_disable_alert_interrupt(hal_smbus_dev_struct *smbus_dev);
功能描述	报警模式关闭并关闭中断
先决条件	-
输入参数{in}	
smbus_dev	SMBUS设备信息结构体, 结构体成员参考 表 3-302.结构体 hal_smbus_dev_struct
输出参数{out}	
-	-
返回值	
error code	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如:

```
/* disable the SMBUS alert mode with Interrupt */
```

```
hal_smbus_dev_struct i2c0_smbus_info;
```

```
hal_smbus_disable_alert_interrupt(&i2c0_smbus_info);
```

3.16. NVIC

嵌套向量中断控制器（NVIC）。章节[3.16.1](#)描述了NVIC的寄存器列表，章节[3.16.2](#)对NVIC库函数进行说明。

3.16.1. 外设寄存器说明

NVIC寄存器列表如下表所示：

表 3-318. NVIC 寄存器

寄存器名称	寄存器描述
ISER ⁽¹⁾	中断使能寄存器
ICER ⁽¹⁾	中断禁能寄存器
ISPR ⁽¹⁾	中断挂起寄存器
ICPR ⁽¹⁾	中断清除寄存器
IABR ⁽¹⁾	中断活动状态寄存器
ITNS ⁽¹⁾	中断不安全状态寄存器
IPR ⁽¹⁾	中断优先级寄存器
CPUID ⁽²⁾	CPUID寄存器
ICSR ⁽²⁾	中断控制及状态寄存器
VTOR ⁽²⁾	向量表偏移量寄存器
AIRCR ⁽²⁾	应用程序中断及复位控制寄存器
SCR ⁽²⁾	系统控制寄存器
CCR ⁽²⁾	配置与控制寄存器
SHPR ⁽²⁾	系统异常优先级寄存器
SHCSR ⁽²⁾	系统异常控制及状态寄存器

1. 参考 core_cm4.h 文件中定义的结构体类型 NVIC_Type

2. 参考 core_cm4.h 文件中定义的结构体类型 SCB_Type

3.16.2. 外设库函数说明

枚举 IRQn_Type

表 3-319. IRQn_Type

成员名称	功能描述
WWDGT_IRQn	窗口看门狗中断
LVD_IRQn	连接到 EXTI 线的 LVD 中断
RTC_IRQn	RTC 全局中断
FMC_IRQn	FMC 全局中断
RCU_IRQn	RCU 全局中断
EXTI0_1_IRQn	EXTI 线 0 和线 1 中断
EXTI2_3_IRQn	EXTI 线 2 和线 3 中断

EXTI4_15_IRQn	EXTI 线 4 至线 15 中断
DMA_Channel0_IRQn	DMA 通道 0 全局中断
DMA_Channel1_2_IRQn	DMA 通道 1 和通道 2 全局中断
DMA_Channel3_4_IRQn	DMA 通道 3 和通道 4 全局中断
ADC_CMP_IRQn	ADC 和 CMP 全局中断
TIMER0_BRK_UP_TRG_COM_IRQn	TIMER0 中止&更新&触发&换向中断
TIMER0_Channel_1_IRQn	TIMER0 通道捕获比较中断
TIMER2_IRQn	TIMER2 全局中断
TIMER5_IRQn	TIMER5 全局中断
TIMER13_IRQn	TIMER13 全局中断
TIMER14_IRQn	TIMER14 全局中断
TIMER15_IRQn	TIMER15 全局中断
TIMER16_IRQn	TIMER16 全局中断
I2C0_EV_IRQn	I2C0 事件中断
I2C1_EV_IRQn	I2C1 事件中断
SPI0_IRQn	SPI0 全局中断
SPI1_IRQn	SPI1 全局中断
USART0_IRQn	USART0 全局中断
USART1_IRQn	USART1 全局中断
I2C0_ER_IRQn	I2C0 错误中断
I2C1_ER_IRQn	I2C1 错误中断

NVIC库函数列表如下表所示：

表 3-320. DAC 库函数

库函数名称	库函数描述
hal_nvic_irq_priority_group_set	设置NVIC请求优先级
hal_nvic_irq_enable	使能NVIC外设请求
hal_nvic_set_priority	设置系统NVIC请求优先级

函数 hal_nvic_irq_priority_group_set

函数hal_nvic_irq_priority_group_set描述见下表：

表 3-321. 函数 hal_nvic_irq_priority_group_set

函数名称	hal_nvic_irq_priority_group_set
函数原型	void hal_nvic_irq_priority_group_set(uint32_t nvic_prigroup);
功能描述	设置NVIC请求优先级组
先决条件	-

被调用函数	-
输入参数{in}	
nvic_prigroup	NVIC中断优先级组
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set priority group NVIC_PRIGROUP_PRE0_SUB4 */
```

```
hal_nvic_irq_priority_group_set (NVIC_PRIGROUP_PRE0_SUB4);
```

函数 hal_nvic_irq_enable

函数hal_nvic_irq_enable描述见下表:

表 3-322. 函数 hal_nvic_irq_enable

函数名称	hal_nvic_irq_enable
函数原型	void hal_nvic_irq_enable(IRQn_Type nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
功能描述	使能NVIC请求并设置外部中断优先级
先决条件	-
被调用函数	-
输入参数{in}	
nvic_irq	NVIC中断请求, 枚举成员参考 表 3-319. IRQn_Type
输入参数{in}	
nvic_irq_pre_priority	设置外部中断父优先级
输入参数{in}	
nvic_irq_sub_priority	设置外部中断子优先级
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable USART0 NVIC request */
```

```
hal_nvic_irq_enable((USART0, 0, 0);
```

3.17. PMU

电源管理单元提供了三种省电模式，包括睡眠模式，深度睡眠模式和待机模式。章节 [3.17.1](#) 描述了 PMU 的寄存器列表，章节 [3.17.2](#) 对 PMU 库函数进行说明。

3.17.1. 外设寄存器说明

PMU 寄存器列表如下表所示：

表 3-323. PMU 寄存器

寄存器名称	寄存器描述
PMU_CTL	PMU控制寄存器
PMU_CS	PMU控制和状态寄存器

3.17.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-324. PMU 库函数

库函数名称	库函数描述
hal_pmu_deinit	复位外设PMU
hal_pmu_struct_init	初始化PMU外设结构体
hal_pmu_lvd_init	PMU外设结构体配置
hal_pmu_wakeup_pin_enable	WKUP引脚唤醒使能
hal_pmu_lvd_start	开启低电压检测器
hal_pmu_lvd_stop	关闭低电压检测器
hal_pmu_wakeup_pin_disable	WKUP引脚唤醒失能
hal_pmu_lvd_irq	PMU中断处理函数
hal_pmu_lvd_irq_handle_set	设置PMU中断回调函数
hal_pmu_lvd_irq_handle_all_reset	清除PMU中断回调函数
hal_pmu_lvd_start_interrupt	中断模式启动PMU
hal_pmu_lvd_stop_interrupt	关闭中断模式启动PMU的模式

枚举 hal_pmu_lvd_voltage_enum

表 3-325. 枚举 hal_pmu_lvd_voltage_enum

成员名称	功能描述
PMU_LVDT_0	低电压检测器阈值为 2.1V
PMU_LVDT_1	低电压检测器阈值为2.3V
PMU_LVDT_2	低电压检测器阈值为2.4V
PMU_LVDT_3	低电压检测器阈值为2.6V
PMU_LVDT_4	低电压检测器阈值为2.7V
PMU_LVDT_5	低电压检测器阈值为2.9V
PMU_LVDT_6	低电压检测器阈值为3.0V

成员名称	功能描述
PMU_LVDT_7	低电压检测器阈值为3.1V

枚举 `hal_pmu_struct_type_enum`

表 3-326. 枚举 `hal_pmu_struct_type_enum`

成员名称	功能描述
HAL_PMU_INIT_STRUCT	PMU初始化结构体
HAL_PMU_IRQ_STRUCT	PMU中断回调函数指针结构体
HAL_PMU_DEV_STRUCT	PMU设备信息结构体

枚举 `hal_pmu_error_enum`

表 3-327. 枚举 `hal_pmu_error_enum`

成员名称	功能描述
HAL_PMU_ERROR_NONE	无错误
HAL_PMU_ERROR_SYSTEM	PMU内部错误，如果时钟问题，使能、禁能错误状态
HAL_PMU_ERROR_CONFIG	PMU产生配置错误

枚举 `hal_pmu_state_enum`

表 3-328. 枚举 `hal_pmu_state_enum`

成员名称	功能描述
HAL_PMU_STATE_NONE	无（默认值）
HAL_PMU_STATE_RESET	PMU未被初始化或停止
HAL_PMU_STATE_BUSY	PMU繁忙
HAL_PMU_STATE_TIMEOUT	PMU产生超时
HAL_PMU_STATE_ERROR	PMU出错
HAL_PMU_STATE_READY	PMU准备

结构体 `hal_pmu_lvd_irq_struct`

表 3-329. 结构体 `hal_pmu_lvd_irq_struct`

成员名称	功能描述
<code>pmu_lvd_handle</code>	LVD 中断处理回调函数

结构体 `hal_pmu_init_struct`

表 3-330. 结构体 `hal_pmu_init_struct`

成员名称	功能描述
<code>int_event_mode</code>	低电压检测器中断或事件模式选择
<code>trig_type</code>	触发类型
<code>lvd_threshold</code>	低电压检测器阈值选择

结构体 hal_pmu_dev_struct

表 3-331. 结构体 hal_pmu_dev_struct

成员名称	功能描述
pmu_lvd_irq	PMU中断回调函数指针结构体
error_state	PMU错误信息
state	PMU信息
mutex	互斥锁和解锁状态
priv	隐私数据

函数 hal_pmu_deinit

函数hal_pmu_deinit描述见下表：

表 3-332. 函数 hal_pmu_deinit

函数名称	hal_pmu_deinit
函数原形	int32_t hal_pmu_deinit(hal_pmu_dev_struct *pmu_dev)
功能描述	复位PMU外设
先决条件	-
被调用函数	hal_rcu_periph_reset_enable / hal_rcu_periph_reset_disable
输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 hal_pmu_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* reset PMU peripheral */
hal_pmu_dev_struct pmu_info;
hal_pmu_deinit(&pmu_info);
```

函数 hal_pmu_struct_init

函数hal_pmu_struct_init描述见下表：

表 3-333. 函数 hal_pmu_struct_init

函数名称	hal_pmu_struct_init
函数原形	void hal_pmu_struct_init(hal_pmu_struct_type_enum hal_struct_type, void *p_struct)
功能描述	初始化PMU外设结构体
先决条件	-
被调用函数	-

输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 <i>hal_pmu_dev_struct</i>
输出参数{out}	
p_struct	指向包含配置信息的PMU结构体的指针
返回值	
-	-

例如：

```
/* initialize the PMU structure */
hal_pmu_dev_struct pmu_info;
hal_pmu_struct_init(HAL_PMU_DEV_STRUCT, &pmu_info);
```

函数 **hal_pmu_lvd_init**

函数hal_pmu_lvd_init描述见下表：

表 3-334. 函数 hal_pmu_lvd_init

函数名称	hal_pmu_lvd_init
函数原形	int32_t hal_pmu_lvd_init(hal_pmu_dev_struct *pmu_dev, hal_pmu_init_struct *pmu_lvd_init)
功能描述	低电压检测器配置
先决条件	-
被调用函数	-
输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 <i>hal_pmu_dev_struct</i>
pmu_lvd_init	指向hal_pmu_init_struct结构体的指针，结构体成员参考 表3-334. 函数 <i>hal_pmu_lvd_init</i>
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* configure EXTI_16 and then configure low voltage detector threshold */
hal_pmu_dev_struct pmu_info;
hal_pmu_lvd_init_struct pmu_init_parameter;
pmu_init_parameter.lvd_threshold = PMU_LVDT_6;
pmu_init_parameter.int_event_mode = PMU_LVD_EVENT_MODE;
```

```
pmu_init_parameter.trig_type = PMU_LVD_TRIG_RISING;
```

```
hal_pmu_lvd_init(&pmu_info,&pmu_init_parameter);
```

函数 hal_pmu_wakeup_pin_enable

函数hal_pmu_wakeup_pin_enable描述见下表:

表 3-335. 函数 hal_pmu_wakeup_pin_enable

函数名称	hal_pmu_wakeup_pin_enable
函数原形	int32_t hal_pmu_wakeup_pin_enable(hal_pmu_dev_struct *pmu_dev, uint32_t wakeup_pin)
功能描述	WKUP引脚唤醒使能
先决条件	-
被调用函数	-
输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针, 结构体成员参考 表3-331. 结构体 hal_pmu_dev_struct
wakeup_pin	WKUP引脚
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN4	WKUP Pin 4 (PC5)
PMU_WAKEUP_PIN5	WKUP Pin 5 (PB5)
PMU_WAKEUP_PIN6	WKUP Pin 6 (PB15)
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* enable PMU wakeup pin PA0 */
```

```
hal_pmu_dev_struct pmu_info;
```

```
hal_pmu_wakeup_pin_enable(&pmu_info,PMU_WAKEUP_PIN0);
```

函数 hal_pmu_lvd_start

函数hal_pmu_lvd_start描述见下表:

表 3-336. 函数 hal_pmu_lvd_start

函数名称	hal_pmu_lvd_start
函数原形	int32_t hal_pmu_lvd_start(hal_pmu_dev_struct *pmu_dev)
功能描述	开启低电压检测器
先决条件	-
被调用函数	-

输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 hal_pmu_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* start LVD detector */
hal_pmu_dev_struct pmu_info;
hal_pmu_lvd_start(&pmu_info);
```

函数 hal_pmu_lvd_stop

函数hal_pmu_lvd_stop描述见下表：

表 3-337. 函数 hal_pmu_lvd_stop

函数名称	hal_pmu_lvd_stop
函数原形	int32_t hal_pmu_lvd_stop(hal_pmu_dev_struct *pmu_dev)
功能描述	停止低电压检测器
先决条件	-
被调用函数	-
输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 hal_pmu_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* stop LVD detector */
hal_pmu_dev_struct pmu_info;
hal_pmu_lvd_stop(&pmu_info);
```

函数 hal_pmu_wakeup_pin_disable

函数hal_pmu_wakeup_pin_disable描述见下表：

表 3-338. 函数 hal_pmu_wakeup_pin_disable

函数名称	hal_pmu_wakeup_pin_disable
函数原形	int32_t hal_pmu_wakeup_pin_disable(hal_pmu_dev_struct *pmu_dev,

	uint32_t wakeup_pin)
功能描述	WKUP引脚唤醒失能
先决条件	-
被调用函数	-
输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 hal_pmu_dev_struct
wakeup_pin	WKUP引脚
PMU_WAKEUP_PIN0	WKUP Pin 0 (PA0)
PMU_WAKEUP_PIN1	WKUP Pin 1 (PC13)
PMU_WAKEUP_PIN4	WKUP Pin 4 (PC5)
PMU_WAKEUP_PIN5	WKUP Pin 5 (PB5)
PMU_WAKEUP_PIN6	WKUP Pin 6 (PB15)
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* disable PMU wakeup pin PA0 */
```

```
hal_pmu_dev_struct pmu_info;
```

```
hal_pmu_wakeup_pin_disable(&pmu_info,PMU_WAKEUP_PIN0);
```

函数 hal_pmu_lvd_irq

函数hal_pmu_lvd_irq描述见下表：

表 3-339. 函数 hal_pmu_lvd_irq

函数名称	hal_pmu_lvd_irq
函数原形	void hal_pmu_lvd_irq(hal_pmu_dev_struct *pmu_dev)
功能描述	PMU中断处理函数
先决条件	-
被调用函数	-
输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 hal_pmu_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU interrupt handler content function,which is merely used in pmu_lvd_handler */
```

```
hal_pmu_dev_struct pmu_info;
```

```
hal_pmu_lvd_irq(&pmu_info);
```

函数 hal_pmu_lvd_irq_handle_set

函数hal_pmu_lvd_irq_handle_set描述见下表：

表 3-340. 函数 hal_pmu_lvd_irq_handle_set

函数名称	hal_pmu_lvd_irq_handle_set
函数原形	void hal_pmu_lvd_irq_handle_set(hal_pmu_dev_struct *pmu_dev, hal_pmu_lvd_irq_struct *irq_handle)
功能描述	设置PMU中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 hal_pmu_dev_struct
irq_handle	指向hal_pmu_lvd_irq_struct结构体的指针，结构体成员参考 表3-329. 结构体 hal_pmu_lvd_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set user-defined interrupt callback function */
```

```
hal_pmu_dev_struct pmu_info;
```

```
hal_pmu_lvd_irq_struct lvd_irq;
```

```
void lvd_irq_test(void *p);
```

```
lvd_irq.pmu_lvd_handle = lvd_irq_test;
```

```
hal_pmu_lvd_irq_handle_set(&pmu_info, &lvd_irq);
```

函数 hal_pmu_lvd_irq_handle_reset

函数hal_pmu_lvd_irq_handle_reset描述见下表：

表 3-341. 函数 hal_pmu_lvd_irq_handle_reset

函数名称	hal_pmu_lvd_irq_handle_reset
函数原形	void hal_pmu_lvd_irq_handle_reset(hal_pmu_dev_struct *pmu_dev)
功能描述	清除PMU中断回调函数
先决条件	-
被调用函数	-

输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 hal_pmu_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback function */
```

```
hal_pmu_dev_struct pmu_info;
```

```
hal_pmu_lvd_irq_handle_reset(&pmu_info);
```

函数 hal_pmu_lvd_start_interrupt

函数hal_pmu_lvd_start_interrupt描述见下表：

表 3-342. 函数 hal_pmu_lvd_start_interrupt

函数名称	hal_pmu_lvd_start_interrupt
函数原形	int32_t hal_pmu_lvd_start_interrupt(hal_pmu_dev_struct *pmu_dev, hal_pmu_lvd_irq_struct *irq_handle)
功能描述	中断模式启动PMU
先决条件	-
被调用函数	-
输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 hal_pmu_dev_struct
irq_handle	指向hal_pmu_lvd_irq_struct结构体的指针，结构体成员参考 表3-329. 结构体 hal_pmu_lvd_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* start PMU lvd with interrupt method */
```

```
hal_pmu_dev_struct pmu_info;
```

```
hal_pmu_lvd_irq_struct lvd_irq;
```

```
void lvd_irq_test(void *p);
```

```
lvd_irq.pmu_lvd_handle = lvd_irq_test;
```

```
hal_pmu_lvd_start_interrupt(&pmu_info, &lvd_irq);
```

函数 hal_pmu_lvd_stop_interrupt

函数hal_pmu_lvd_stop_interrupt描述见下表：

表 3-343. 函数 hal_pmu_lvd_stop_interrupt

函数名称	hal_pmu_lvd_stop_interrupt
函数原形	int32_t hal_pmu_lvd_stop_interrupt(hal_pmu_dev_struct *pmu_dev)
功能描述	关闭中断模式启动PMU的模式
先决条件	-
被调用函数	-
输入参数{in}	
pmu_dev	指向PMU设备信息结构体的指针，结构体成员参考 表3-331. 结构体 hal_pmu_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* stop PMU lvd with interrupt method */
hal_pmu_dev_struct pmu_info;
void lvd_irq_test(void *p);
lvd_irq.pmu_lvd_handle = lvd_irq_test;
hal_pmu_lvd_stop_interrupt(&pmu_info);
```

3.18. RCU

RCU 是复位和时钟单元，复位控制包括三种控制方式：电源复位、系统复位和备份域复位。时钟控制单元提供了一系列频率的时钟功能。章节 [3.18.1](#) 描述了 RCU 的寄存器列表，章节 [3.18.2](#) 对 RCU 库函数进行说明。

3.18.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-344. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL0	控制寄存器0
RCU_CFG0	配置寄存器0
RCU_INT	中断寄存器

寄存器名称	寄存器描述
RCU_APB2RST	APB2复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_AHBEN	AHB使能寄存器
RCU_APB2EN	APB2使能寄存器
RCU_APB1EN	APB1使能寄存器
RCU_BDCTL	备份域控制寄存器
RCU_RSTSCK	复位源/时钟寄存器
RCU_AHBRST	AHB复位寄存器
RCU_CFG1	配置寄存器1
RCU_CFG2	配置寄存器2
RCU_CTL1	控制寄存器1
RCU_ADDCTL	附加时钟控制寄存器
RCU_ADDINT	附加时钟中断寄存器
RCU_ADDAPB1EN	APB1附加使能寄存器
RCU_ADDAPB1RST	APB1附加复位寄存器
RCU_VKEY	电源解锁寄存器
RCU_DSV	深度睡眠模式电压寄存器

3.18.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-345. RCU 库函数

库函数名称	库函数描述
hal_rcu_osci_config	配置RCU晶振
hal_rcu_clock_out_config	配置时钟输出到CKOUT pin脚
hal_rcu_deinit	复位RCU
hal_rcu_struct_init	初始化RCU相关结构体
hal_rcu_periph_clk_enable	使能外设时钟
hal_rcu_periph_clk_disable	失能外设时钟
hal_rcu_hxtal_clock_monitor_enable	使能外部高速晶振时钟监视器
hal_rcu_hxtal_clock_monitor_disable	失能外部高速晶振时钟监视器
hal_rcu_periph_reset_enable	外设复位
hal_rcu_periph_reset_disable	失能外设复位
hal_rcu_periph_clock_config	配置RCU扩展外设(RTC, Usart0, ADC, USBFS, CEC) 时钟
hal_rcu_periph_clkfreq_get	获取外设时钟频率
hal_rcu_osci_config_get	获取RCU晶振配置
hal_rcu_clock_config	配置RCU时钟
hal_rcu_clock_config_get	获取RCU时钟配置
hal_SystemCoreClockUpdate	系统时钟更新
hal_rcu_irq	RCU中断处理函数
hal_rcu_irq_handle_set	设定用户定义的中断回调函数

库函数名称	库函数描述
hal_rcu_irq_handle_all_reset	复位用户定义的中断回调函数

枚举 hal_rcu_periph_enum

表 3-346. 枚举 hal_rcu_periph_enum

成员名称	功能描述
RCU_DMA	DMA时钟
RCU_CRC	CRC时钟
RCU_GPIOA	GPIOA时钟
RCU_GPIOB	GPIOB时钟
RCU_GPIOC	GPIOC时钟
RCU_GPIOD	GPIOD时钟
RCU_GPIOF	GPIOF时钟
RCU_TSI	TSI时钟
RCU_CFGCMP	CGFCMP时钟
RCU_ADC	ADC时钟
RCU_TIMER0	TIMER0时钟
RCU_SPI0	SPI0时钟
RCU_USART0	USART0时钟
RCU_TIMER14	TIMER14时钟
RCU_TIMER15	TIMER15时钟
RCU_TIMER16	TIMER16时钟
RCU_TIMER1	TIMER1时钟
RCU_TIMER2	TIMER2时钟
RCU_TIMER13	TIMER13时钟
RCU_WWDGT	WWDGT时钟
RCU_SPI1	SPI1时钟
RCU_USART1	USART1时钟
RCU_I2C0	I2C0时钟
RCU_I2C1	I2C1时钟
RCU_PMU	PMU时钟
RCU_DAC	DAC时钟（仅对GD32F350适用）
RCU_CEC	CEC时钟（仅对GD32F350适用）
RCU_TIMER5	TIMER5时钟（仅对GD32F350适用）
RCU_USBFS	USBFS时钟（仅对GD32F350适用）
RCU_RTC	RTC时钟
RCU_CTC	CTC时钟

枚举 hal_rcu_periph_sleep_enum

表 3-347. 枚举 hal_rcu_periph_sleep_enum

成员名称	功能描述
RCU_SRAM_SLP	SRAM时钟
RCU_FMC_SLP	FMC时钟

枚举 hal_rcu_periph_reset_enum

表 3-348. 枚举 hal_rcu_periph_reset_enum

成员名称	功能描述
RCU_GPIOARST	GPIOA复位
RCU_GPIOBRST	GPIOB复位
RCU_GPIOCRST	GPIOC复位
RCU_GPIODRST	GIPOD复位
RCU_GPIOFRST	GPIOF复位
RCU_TSIRST	TSI复位
RCU_CFGCMRST	CFGCMP复位
RCU_ADCRST	ADC复位
RCU_TIMER0RST	TIMER0复位
RCU_SPI0RST	SPI0复位
RCU_USART0RST	USART0复位
RCU_TIMER14RST	TIMER14复位
RCU_TIMER15RST	TIMER15复位
RCU_TIMER16RST	TIMER16复位
RCU_TIMER1RST	TIMER1复位
RCU_TIMER2RST	TIMER2复位
RCU_TIMER13RST	TIMER13复位
RCU_WWDGTRST	WWDGT复位
RCU_SPI1RST	SPI1复位
RCU_USART1RST	USART1复位
RCU_I2C0RST	I2C0复位
RCU_I2C1RST	I2C1复位
RCU_PMURST	PMU复位
RCU_DACRST	DAC复位（仅对GD32F350适用）
RCU_CECRST	CEC复位（仅对GD32F350适用）
RCU_TIMER5RST	TIMER5复位（仅对GD32F350适用）
RCU_USBFSRST	USBFS复位（仅对GD32F350适用）
RCU_CTCRST	CTC复位

枚举 hal_rcu_flag_enum

表 3-349. 枚举 hal_rcu_flag_enum

成员名称	功能描述
RCU_FLAG_IRC40KSTB	IRC40K振荡器稳定标志
RCU_FLAG_LXTALSTB	外部低速晶振稳定标志
RCU_FLAG_IRC8MSTB	IRC8M振荡器稳定标志
RCU_FLAG_HXTALSTB	外部高速晶振稳定标志
RCU_FLAG_PLLSTB	PLL稳定标志
RCU_FLAG_IRC28MSTB	IRC28M振荡器稳定标志
RCU_FLAG_IRC48MSTB	IRC48M振荡器稳定标志
RCU_FLAG_V12RST	1.2V电源域复位标志
RCU_FLAG_OBLRST	可选字节装载器复位标志
RCU_FLAG_EPRST	外部引脚复位标志
RCU_FLAG_PORRST	电源复位标志
RCU_FLAG_SWRST	软件复位标志
RCU_FLAG_FWDGTRST	独立看门狗复位标志
RCU_FLAG_WWDGTRST	窗口看门狗复位标志
RCU_FLAG_LPRST	低功耗复位标志

枚举 hal_rcu_int_flag_enum

表 3-350. 枚举 hal_rcu_int_flag_enum

成员名称	功能描述
RCU_INT_FLAG_IRC40KSTB	IRC40K时钟稳定中断标志
RCU_INT_FLAG_LXTALSTB	外部低速晶振时钟稳定中断标志
RCU_INT_FLAG_IRC8MSTB	IRC8M时钟稳定中断标志

成员名称	功能描述
C8MSTB	
RCU_INT_FLAG_H XTALSTB	外部高速晶振时钟稳定中断标志
RCU_INT_FLAG_P LLSTB	PLL时钟稳定中断标志
RCU_INT_FLAG_IR C28MSTB	IRC28M时钟稳定中断标志
RCU_INT_FLAG_C KM	外部高速晶振时钟阻塞中断标志
RCU_INT_FLAG_IR C48MSTB	IRC48M时钟稳定中断标志

枚举 hal_rcu_int_flag_clear_enum

表 3-351. 枚举 hal_rcu_int_flag_clear_enum

成员名称	功能描述
RCU_INT_FLAG_IR C40KSTB_CLR	IRC40K时钟稳定中断清除标志
RCU_INT_FLAG_L XTALSTB_CLR	外部低速晶振时钟稳定中断清除标志
RCU_INT_FLAG_IR C8MSTB_CLR	IRC8M时钟稳定中断清除标志
RCU_INT_FLAG_H XTALSTB_CLR	外部高速晶振时钟稳定中断清除标志
RCU_INT_FLAG_P LLSTB_CLR	PLL时钟稳定中断清除标志
RCU_INT_FLAG_IR C28MSTB_CLR	IRC28M时钟稳定中断清除标志
RCU_INT_FLAG_C KM_CLR	外部高速晶振时钟阻塞中断清除标志
RCU_INT_FLAG_IR C48MSTB_CLR	IRC48M时钟稳定中断清除标志

枚举 hal_rcu_int_enum

表 3-352. 枚举 hal_rcu_int_enum

成员名称	功能描述
RCU_INT_IRC40KS TB	IRC40K时钟稳定中断
RCU_INT_LXTALS TB	外部低速晶振时钟稳定中断
RCU_INT_IRC8MS TB	IRC8M时钟稳定中断

成员名称	功能描述
RCU_INT_HXTALS TB	外部高速晶振时钟稳定中断
RCU_INT_PLLSTB	PLL时钟稳定中断
RCU_INT_IRC28M STB	IRC28M时钟稳定中断
RCU_INT_IRC48M STB	IRC48M时钟稳定中断

枚举 `hal_rcu_adc_clksrc_enum`

表 3-353. 枚举 `hal_rcu_adc_clksrc_enum`

成员名称	功能描述
RCU_ADCCCK_IRC2 8M_DIV2	选择IRC28M的2分频作为ADC的时钟源
RCU_ADCCCK_IRC2 8M	选择IRC28M作为ADC的时钟源
RCU_ADCCCK_APB 2_DIV2	选择APB2的2分频作为ADC的时钟源
RCU_ADCCCK_AHB _DIV3	选择AHB的3分频作为ADC的时钟源
RCU_ADCCCK_APB 2_DIV4	选择APB2的4分频作为ADC的时钟源
RCU_ADCCCK_AHB _DIV5	选择AHB的5分频作为ADC的时钟源
RCU_ADCCCK_APB 2_DIV6	选择APB2的6分频作为ADC的时钟源
RCU_ADCCCK_AHB _DIV7	选择AHB的7分频作为ADC的时钟源
RCU_ADCCCK_APB 2_DIV8	选择APB2的8分频作为ADC的时钟源
RCU_ADCCCK_AHB _DIV9	选择AHB的9分频作为ADC的时钟源

枚举 `hal_rcu_clock_freq_enum`

表 3-354. 枚举 `hal_rcu_clock_freq_enum`

成员名称	功能描述
CK_SYS	系统时钟
CK_AHB	AHB时钟
CK_APB1	APB1时钟
CK_APB2	APB2时钟
CK_ADC	ADC时钟
CK_CEC	CEC时钟

成员名称	功能描述
CK_USART	USART时钟

枚举 hal_rcu_osci_type_enum

表 3-355. 枚举 hal_rcu_osci_type_enum

成员名称	功能描述
RCU_HXTAL	外部高速振荡器
RCU_LXTAL	外部低速振荡器
RCU_IRC8M	IRC8M振荡器
RCU_IRC28M	IRC28M振荡器
RCU_IRC48M	IRC48M振荡器
RCU_IRC40K	IRC40K振荡器
RCU_PLL_CK	锁相环时钟

枚举 hal_rcu_struct_type_enum

表 3-356. 枚举 hal_rcu_struct_type_enum

成员名称	功能描述
HAL_RCU_CLK_STRUCTURE	RCU时钟结构体类型
HAL_RCU_OSCI_STRUCTURE	RCU晶振结构体类型
HAL_RCU_PERIPH_CLK_STRUCTURE	RCU外设时钟结构体类型

枚举 hal_rcu_rtc_clksrc_enum

表 3-357. 枚举 hal_rcu_rtc_clksrc_enum

成员名称	功能描述
RCU_RTC_CLKSRC_C_NONE	无时钟选择
RCU_RTC_CLKSRC_C_LXTAL	选择外部低速振荡器作为RTC时钟源
RCU_RTC_CLKSRC_C_IRC40K	选择IRC40K振荡器作为RTC时钟源
RCU_RTC_CLKSRC_C_HXTAL_DIV32	选择外部高速振荡器的32分频作为RTC时钟源

枚举 hal_rcu_usart_clksrc_enum

表 3-358. 枚举 hal_rcu_usart_clksrc_enum

成员名称	功能描述
RCU_USART0_CLK	CK_USART0选择CK_APB2

成员名称	功能描述
SRC_APB2	
RCU_USART0_CLK SRC_SYS	CK_USART0选择CK_SYS
RCU_USART0_CLK SRC_LXTAL	CK_USART0选择LXTAL
RCU_USART0_CLK SRC_IRC8M	CK_USART0选择IRC8M

枚举 hal_rcu_usbfs_clksrc_enum

表 3-359. 枚举 hal_rcu_usbfs_clksrc_enum

成员名称	功能描述
RCU_PLLCLK_USB FS_DIV1_5	USBFS时钟预分频选择CK_PLL/1.5
RCU_PLLCLK_USB FS_DIV1	USBFS时钟预分频选择CK_PLL
RCU_PLLCLK_USB FS_DIV2_5	USBFS时钟预分频选择CK_PLL/2.5
RCU_PLLCLK_USB FS_DIV2	USBFS时钟预分频选择CK_PLL/2
RCU_PLLCLK_USB FS_DIV3_5	USBFS时钟预分频选择CK_PLL/3.5
RCU_PLLCLK_USB FS_DIV3	USBFS时钟预分频选择CK_PLL/3

枚举 hal_rcu_cec_clksrc_enum

表 3-360. 枚举 hal_rcu_cec_clksrc_enum

成员名称	功能描述
RCU_CEC_CLKSR C_DIV244	CK_CEC时钟源选择IRC8M/244
RCU_CEC_CLKSR C_LXTAL	CK_CEC时钟源选择LXTAL

枚举 hal_rcu_sysclk_src_enum

表 3-361. 枚举 hal_rcu_sysclk_src_enum

成员名称	功能描述
RCU_SYSCLK_SR C_IRC8M	系统时钟源选择IRC8M
RCU_SYSCLK_SR C_HXTAL	系统时钟源选择HXTAL
RCU_SYSCLK_SR	系统时钟源选择PLL

成员名称	功能描述
C_PLL	

枚举 `hal_rcu_ck48mclk_src_enum`

表 3-362. 枚举 `hal_rcu_ck48mclk_src_enum`

成员名称	功能描述
RCU_USB_CK48M SRC_PLL48M	CK48M时钟源选择PLL48M
RCU_USB_CK48M SRC_IRC48M	CK48M时钟源选择IRC48M

枚举 `hal_rcu_sysclk_ahbdiv_enum`

表 3-363. 枚举 `hal_rcu_sysclk_ahbdiv_enum`

成员名称	功能描述
RCU_SYSCLK_AH BDIV1	AHB预分频选择CK_SYS
RCU_SYSCLK_AH BDIV2	AHB预分频选择CK_SYS/2
RCU_SYSCLK_AH BDIV4	AHB预分频选择CK_SYS/4
RCU_SYSCLK_AH BDIV8	AHB预分频选择CK_SYS/8
RCU_SYSCLK_AH BDIV16	AHB预分频选择CK_SYS/16
RCU_SYSCLK_AH BDIV32	AHB预分频选择CK_SYS/32
RCU_SYSCLK_AH BDIV64	AHB预分频选择CK_SYS/64
RCU_SYSCLK_AH BDIV128	AHB预分频选择CK_SYS/128
RCU_SYSCLK_AH BDIV256	AHB预分频选择CK_SYS/256
RCU_SYSCLK_AH BDIV512	AHB预分频选择CK_SYS/512

枚举 `hal_rcu_ahbclk_apb1div_enum`

表 3-364. 枚举 `hal_rcu_ahbclk_apb1div_enum`

成员名称	功能描述
RCU_AHBCLK_AP B1DIV1	AHB的1分频作为APB1的时钟源
RCU_AHBCLK_AP	AHB的2分频作为APB1的时钟源

成员名称	功能描述
B1DIV2	
RCU_AHBCLK_APB1DIV4	AHB的4分频作为APB1的时钟源
RCU_AHBCLK_APB1DIV8	AHB的8分频作为APB1的时钟源
RCU_AHBCLK_APB1DIV16	AHB的16分频作为APB1的时钟源

枚举 `hal_rcu_ahbclk_apb2div_enum`

表 3-365. 枚举 `hal_rcu_ahbclk_apb2div_enum`

成员名称	功能描述
RCU_AHBCLK_APB2DIV1	AHB的1分频作为APB2的时钟源
RCU_AHBCLK_APB2DIV2	AHB的2分频作为APB2的时钟源
RCU_AHBCLK_APB2DIV4	AHB的4分频作为APB2的时钟源
RCU_AHBCLK_APB2DIV8	AHB的8分频作为APB2的时钟源
RCU_AHBCLK_APB2DIV16	AHB的16分频作为APB2的时钟源

枚举 `hal_rcu_osc_state_enum`

表 3-366. 枚举 `hal_rcu_osc_state_enum`

成员名称	功能描述
RCU_OSC_NONE	无晶振配置
RCU_OSC_OFF	关闭晶振
RCU_OSC_ON	打开晶振
RCU_OSC_BYPASS	外部晶振为旁路模式
RCU_OSC_ADCCTL	ADC的时钟源为IRC28M

枚举 `hal_rcu_pll_src_enum`

表 3-367. 枚举 `hal_rcu_pll_src_enum`

成员名称	功能描述
RCU_PLL_SRC_HXTAL_IRC48M	PLL时钟源选择HXTAL或IRC48M
RCU_PLL_SRC_IRC8M_DIV2	PLL时钟源选择IRC8M/2

枚举 `hal_rcu_pll_prediv_enum`

表 3-368. 枚举 `hal_rcu_pll_prediv_enum`

成员名称	功能描述
<code>RCU_PLL_PREDIV_1</code>	PLL无分频
<code>RCU_PLL_PREDIV_2</code>	PLL 2分频
<code>RCU_PLL_PREDIV_3</code>	PLL 3分频
<code>RCU_PLL_PREDIV_4</code>	PLL 3分频
<code>RCU_PLL_PREDIV_5</code>	PLL 5分频
<code>RCU_PLL_PREDIV_6</code>	PLL 6分频
<code>RCU_PLL_PREDIV_7</code>	PLL 7分频
<code>RCU_PLL_PREDIV_8</code>	PLL 8分频
<code>RCU_PLL_PREDIV_9</code>	PLL 9分频
<code>RCU_PLL_PREDIV_10</code>	PLL 10分频
<code>RCU_PLL_PREDIV_11</code>	PLL 11分频
<code>RCU_PLL_PREDIV_12</code>	PLL 12分频
<code>RCU_PLL_PREDIV_13</code>	PLL 13分频
<code>RCU_PLL_PREDIV_14</code>	PLL 14分频
<code>RCU_PLL_PREDIV_15</code>	PLL 15分频
<code>RCU_PLL_PREDIV_16</code>	PLL 16分频

枚举 `hal_rcu_pll_mul_enum`

表 3-369. 枚举 `hal_rcu_pll_mul_enum`

成员名称	功能描述
<code>RCU_PLL_MULT2</code>	PLL源时钟2倍频
<code>RCU_PLL_MULT3</code>	PLL源时钟3倍频

成员名称	功能描述
RCU_PLL_MULT4	PLL源时钟4倍频
RCU_PLL_MULT5	PLL源时钟5倍频
RCU_PLL_MULT6	PLL源时钟6倍频
RCU_PLL_MULT7	PLL源时钟7倍频
RCU_PLL_MULT8	PLL源时钟8倍频
RCU_PLL_MULT9	PLL源时钟9倍频
RCU_PLL_MULT10	PLL源时钟10倍频
RCU_PLL_MULT11	PLL源时钟11倍频
RCU_PLL_MULT12	PLL源时钟12倍频
RCU_PLL_MULT13	PLL源时钟13倍频
RCU_PLL_MULT14	PLL源时钟14倍频
RCU_PLL_MULT15	PLL源时钟15倍频
RCU_PLL_MULT16	PLL源时钟16倍频
RCU_PLL_MULT17	PLL源时钟17倍频
RCU_PLL_MULT18	PLL源时钟18倍频
RCU_PLL_MULT19	PLL源时钟19倍频
RCU_PLL_MULT20	PLL源时钟20倍频
RCU_PLL_MULT21	PLL源时钟21倍频
RCU_PLL_MULT22	PLL源时钟22倍频
RCU_PLL_MULT23	PLL源时钟23倍频
RCU_PLL_MULT24	PLL源时钟24倍频
RCU_PLL_MULT25	PLL源时钟25倍频
RCU_PLL_MULT26	PLL源时钟26倍频
RCU_PLL_MULT27	PLL源时钟27倍频
RCU_PLL_MULT28	PLL源时钟28倍频
RCU_PLL_MULT29	PLL源时钟29倍频
RCU_PLL_MULT30	PLL源时钟30倍频
RCU_PLL_MULT31	PLL源时钟31倍频
RCU_PLL_MULT32	PLL源时钟32倍频
RCU_PLL_MULT33	PLL源时钟33倍频
RCU_PLL_MULT34	PLL源时钟34倍频
RCU_PLL_MULT35	PLL源时钟35倍频
RCU_PLL_MULT36	PLL源时钟36倍频
RCU_PLL_MULT37	PLL源时钟37倍频
RCU_PLL_MULT38	PLL源时钟38倍频
RCU_PLL_MULT39	PLL源时钟39倍频
RCU_PLL_MULT40	PLL源时钟40倍频
RCU_PLL_MULT41	PLL源时钟41倍频
RCU_PLL_MULT42	PLL源时钟42倍频
RCU_PLL_MULT43	PLL源时钟43倍频
RCU_PLL_MULT44	PLL源时钟44倍频

成员名称	功能描述
RCU_PLL_MULT45	PLL源时钟45倍频
RCU_PLL_MULT46	PLL源时钟46倍频
RCU_PLL_MULT47	PLL源时钟47倍频
RCU_PLL_MULT48	PLL源时钟48倍频
RCU_PLL_MULT49	PLL源时钟49倍频
RCU_PLL_MULT50	PLL源时钟50倍频
RCU_PLL_MULT51	PLL源时钟51倍频
RCU_PLL_MULT52	PLL源时钟52倍频
RCU_PLL_MULT53	PLL源时钟53倍频
RCU_PLL_MULT54	PLL源时钟54倍频
RCU_PLL_MULT55	PLL源时钟55倍频
RCU_PLL_MULT56	PLL源时钟56倍频
RCU_PLL_MULT57	PLL源时钟57倍频
RCU_PLL_MULT58	PLL源时钟58倍频
RCU_PLL_MULT59	PLL源时钟59倍频
RCU_PLL_MULT60	PLL源时钟60倍频
RCU_PLL_MULT61	PLL源时钟61倍频
RCU_PLL_MULT62	PLL源时钟62倍频
RCU_PLL_MULT63	PLL源时钟63倍频
RCU_PLL_MULT64	PLL源时钟64倍频

枚举 hal_rcu_pll_presel_enum

表 3-370. 枚举 hal_rcu_pll_presel_enum

成员名称	功能描述
RCU_PLL_PRESEL _HXTAL	PLL时钟源选择HXTAL
RCU_PLL_PRESEL _IRC48M	PLL时钟源选择IRC48M

枚举 hal_rcu_ckout_src_enum

表 3-371. 枚举 hal_rcu_ckout_src_enum

成员名称	功能描述
RCU_CKOUT_SRC _NONE	无时钟选择
RCU_CKOUT_SRC _IRC28M	CK_OUT时钟源选择IRC28M
RCU_CKOUT_SRC _IRC40K	CK_OUT时钟源选择IRC40K
RCU_CKOUT_SRC _LXTAL	CK_OUT时钟源选择LXTAL

成员名称	功能描述
RCU_CLKOUT_SRC_CKSYS	CK_OUT时钟源选择CKSYS
RCU_CLKOUT_SRC_IRC8M	CK_OUT时钟源选择IRC8M
RCU_CLKOUT_SRC_HXTAL	CK_OUT时钟源选择HXTAL
RCU_CLKOUT_SRC_CKPLL_DIV2	CK_OUT时钟源选择PLL/2
RCU_CLKOUT_SRC_CKPLL_DIV1	CK_OUT时钟源选择PLL

枚举 hal_rcu_ckout_div_enum

表 3-372. 枚举 hal_rcu_ckout_div_enum

成员名称	功能描述
RCU_CLKOUT_DIV_1	CK_OUT 1分频
RCU_CLKOUT_DIV_2	CK_OUT 2分频
RCU_CLKOUT_DIV_4	CK_OUT 4分频
RCU_CLKOUT_DIV_8	CK_OUT 8分频
RCU_CLKOUT_DIV_16	CK_OUT 16分频
RCU_CLKOUT_DIV_32	CK_OUT 32分频
RCU_CLKOUT_DIV_64	CK_OUT 64分频
RCU_CLKOUT_DIV_128	CK_OUT 128分频

结构体 hal_rcu_periphclk_struct

表 3-373. 结构体 hal_rcu_periphclk_struct

成员名称	功能描述
uint32_t periph_clock_type	外设时钟类型选择
hal_rcu_rtc_clksrc_enum	RTC时钟源选择
hal_rcu_usart_clksrc_enum	usart0时钟源选择
hal_rcu_adc_clksrc_enum	ADC时钟源选择

enum	
hal_rcu_usbfs_clksrc_enum	USBFS时钟源选择
hal_rcu_cec_clksrc_enum	CEC时钟源选择

结构体 hal_rcu_clk_struct

表 3-374. 结构体 hal_rcu_clk_struct

成员名称	功能描述
uint32_t clock_type	配置的RCU时钟类型
hal_rcu_sysclk_src_enum	系统时钟源
hal_rcu_ck48mclk_src_enum	内部48M时钟源
hal_rcu_sysclk_ahb_div_enum	AHB时钟分频
hal_rcu_ahbclk_apb1_div_enum	APB1时钟分频
hal_rcu_ahbclk_apb2div_enum	APB2时钟分频

结构体 hal_rcu_irq_struct

表 3-375. 结构体 hal_rcu_irq_struct

成员名称	功能描述
pll_stable_handle	PLL时钟稳定中断
irc40k_stable_handle	IRC40K时钟稳定中断
irc8m_stable_handle	IRC8M时钟稳定中断
irc28m_stable_handle	IRC28M时钟稳定中断
irc48m_stable_handle	IRC28M时钟稳定中断
lxtal_stable_handle	LXTAL时钟稳定中断
hxtal_stable_handle	HXTAL时钟稳定中断
hxtal_stuck_handle	HXTAL时钟停滞异常中断

结构体 hal_rcu_hxtal_struct

表 3-376. 结构体 hal_rcu_hxtal_struct

成员名称	功能描述
ControlStatus	HXTAL时钟配置标志

hal_rcu_osc_state_ enum	HXTAL时钟状态
----------------------------	-----------

结构体 hal_rcu_lxtal_struct

表 3-377. 结构体 hal_rcu_lxtal_struct

成员名称	功能描述
ControlStatus	LXTAL时钟配置标志
hal_rcu_osc_state_ enum	LXTAL时钟状态

结构体 hal_rcu_irc8m_struct

表 3-378. 结构体 hal_rcu_irc8m_struct

成员名称	功能描述
ControlStatus	IRC8M时钟配置标志
uint8_t adjust_value	IRC8M时钟调整值
hal_rcu_osc_state_ enum	IRC8M时钟状态

结构体 hal_rcu_irc28m_struct

表 3-379. 结构体 hal_rcu_irc28m_struct

成员名称	功能描述
ControlStatus	IRC28M时钟配置标志
uint8_t adjust_value	IRC28M时钟调整值
hal_rcu_osc_state_ enum	IRC28M时钟状态

结构体 hal_rcu_irc48m_struct

表 3-380. 结构体 hal_rcu_irc48m_struct

成员名称	功能描述
ControlStatus	IRC48M时钟配置标志
hal_rcu_osc_state_ enum	IRC48M时钟状态

结构体 hal_rcu_irc40k_struct

表 3-381. 结构体 hal_rcu_irc40k_struct

成员名称	功能描述
ControlStatus	IRC40K时钟配置标志
hal_rcu_osc_state_ enum	IRC40K时钟状态

结构体 hal_rcu_pll_struct

表 3-382. 结构体 hal_rcu_pll_struct

成员名称	功能描述
ControlStatus	晶振配置标志
hal_rcu_osc_state_enum	晶振状态
hal_rcu_pll_src_enum	PLL输入时钟源
hal_rcu_pll_prediv_enum	PLL输入时钟的预分频因子
hal_rcu_pll_mul_enum	PLL输入时钟的倍频系数
hal_rcu_pll_presel_enum	PLL输入时钟预选

结构体 hal_rcu_osci_struct

表 3-383. 结构体 hal_rcu_osci_struct

成员名称	功能描述
hal_rcu_hxtal_struct	HXTAL状态结构体
hal_rcu_lxtal_struct	LXTAL状态结构体
hal_rcu_irc8m_struct	IRC8M状态结构体
hal_rcu_irc28m_struct	IRC28M状态结构体
hal_rcu_irc48m_struct	IRC48M状态结构体
hal_rcu_irc40k_struct	IRC40K状态结构体
hal_rcu_pll_struct	PLL状态结构体

函数 hal_rcu_osci_config

函数hal_rcu_osci_config描述见下表:

表 3-384. 函数 hal_rcu_osci_config

函数名称	hal_rcu_osci_config
函数原型	int32_t hal_rcu_osci_config(hal_rcu_osci_struct *rcu_osci)
功能描述	配置RCU晶振
先决条件	-
被调用函数	hal_rcu_deinit hals_rcu_system_clock_source_get hals_rcu_osci_off

	<hals_rcu_osci_bypass_mode_disable< h1=""> <hals_rcu_osci_on< h1=""> <hals_rcu_osci_stab_wait< h1=""> <hal_sys_basetick_count_get< h1=""> <hal_sys_basetick_timeout_check< h1=""> <hal_rcu_periph_clk_enable< h1=""> <hals_pmu_backup_write_enable< h1=""> <hals_rcu_irc8m_adjust_value_set< h1=""> <hals_rcu_irc28m_adjust_value_set< h1=""> <hals_rcu_pll_preselection_config< h1=""> <hals_rcu_hxtal_prediv_config< h1=""> <hals_rcu_pll_config< h1=""> </hals_rcu_pll_config<></hals_rcu_hxtal_prediv_config<></hals_rcu_pll_preselection_config<></hals_rcu_irc28m_adjust_value_set<></hals_rcu_irc8m_adjust_value_set<></hals_pmu_backup_write_enable<></hal_rcu_periph_clk_enable<></hal_sys_basetick_timeout_check<></hal_sys_basetick_count_get<></hals_rcu_osci_stab_wait<></hals_rcu_osci_on<></hals_rcu_osci_bypass_mode_disable<>
输入参数{in}	
rcu_osci	指向RCU设备信息结构体的指针，结构体成员参考 表3-383. 结构体 hal_rcu_osci_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_TIMEOUT, HAL_ERR_VAL, HAL_ERR_NONE

例如：

```

/* configure the RCU oscillators */
hal_rcu_osci_struct rcu_osci_parameter;

rcu_osci_parameter.irc8m.need_configure = ENABLE;

rcu_osci_parameter.irc8m.state = RCU_OSC_ON;

rcu_osci_parameter.irc8m.adjust_value = 0;

rcu_osci_parameter.pll.need_configure = ENABLE;

rcu_osci_parameter.pll.state = RCU_OSC_ON;

rcu_osci_parameter.pll.pll_source = RCU_PLL_SRC_IRC8M_DIV2;

rcu_osci_parameter.pll.pll_mul = RCU_PLL_MULT27;

rcu_osci_parameter.pll.pll_presel = RCU_PLL_PRESEL_HXTAL;

if(HAL_ERR_NONE != hal_rcu_osci_config(&rcu_osci_parameter)){

    while(1);

}

```

函数 hal_rcu_clock_out_config

函数hal_rcu_clock_out_config描述见下表：

表 3-385. 函数 hal_rcu_clock_out_config

函数名称	hal_rcu_clock_out_config
函数原型	void hal_rcu_clock_out_config(hal_rcu_ckout_src_enum ckout_src, hal_rcu_ckout_div_enum ckout_div)
功能描述	配置时钟输出到CKOUT pin脚
先决条件	-
被调用函数	hal_gpio_init hals_rcu_ckout_config
输入参数{in}	
ckout_src	指向RCU设备信息枚举类型，枚举类型成员参考 表3-371. 枚举 hal_rcu_ckout_src_enum
输入参数{in}	
ckout_div	指向RCU设备信息枚举类型，枚举类型成员参考 表3-372. 枚举 hal_rcu_ckout_div_enum
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the clock out to output on CKOUT pin */
```

```
hal_rcu_clock_out_config(RCU_CKOUT_SRC_NONE, RCU_CLKOUT_DIV1);
```

函数 hal_rcu_deinit

函数hal_rcu_deinit描述见下表：

表 3-386. 函数 hal_rcu_deinit

函数名称	hal_rcu_deinit
函数原型	void hal_rcu_deinit(void)
功能描述	复位RCU
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the RCU */
```

```
hal_rcu_deinit();
```

函数 hal_rcu_struct_init

函数hal_rcu_struct_init描述见下表：

表 3-387. 函数 hal_rcu_struct_init

函数名称	hal_rcu_struct_init
函数原型	void hal_rcu_struct_init(hal_rcu_struct_type_enum rcu_struct_type, void *p_struct)
功能描述	初始化RCU相关结构体
先决条件	-
被调用函数	-
输入参数{in}	
rcu_struct_type	指向RCU设备信息枚举类型，枚举类型成员参考 表3-356. 枚举 hal_rcu_struct_type_enum
输入参数{in}	
p_struct	指向包含配置信息的RCU结构体的指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the RCU structure with the default values */
```

```
hal_rcu_clk_struct rcu_clk_parameter;
```

```
hal_rcu_struct_init(HAL_RCU_CLK_STRUCT, &rcu_clk_parameter);
```

函数 hal_rcu_periph_clk_enable

函数hal_rcu_periph_clk_enable描述见下表：

表 3-388. 函数 hal_rcu_periph_clk_enable

函数名称	hal_rcu_periph_clk_enable
函数原型	void hal_rcu_periph_clk_enable(hal_rcu_periph_enum periph)
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	指向RCU设备信息枚举类型，枚举类型成员参考 表3-346. 枚举 hal_rcu_periph_enum

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the peripherals clock */
```

```
hal_rcu_periph_clk_enable(RCU_DMA);
```

函数 hal_rcu_periph_clk_disable

函数hal_rcu_periph_clk_disable描述见下表:

表 3-389. 函数 hal_rcu_periph_clk_disable

函数名称	hal_rcu_periph_clk_disable
函数原型	void hal_rcu_periph_clk_disable(hal_rcu_periph_enum periph)
功能描述	失能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	指向RCU设备信息枚举类型，枚举类型成员参考 表3-346. 枚举 hal_rcu_periph_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the peripherals clock */
```

```
hal_rcu_periph_clk_disable(RCU_DMA);
```

函数 hal_rcu_hxtal_clock_monitor_enable

函数hal_rcu_hxtal_clock_monitor_enable描述见下表:

表 3-390. 函数 hal_rcu_hxtal_clock_monitor_enable

函数名称	hal_rcu_hxtal_clock_monitor_enable
函数原型	void hal_rcu_hxtal_clock_monitor_enable(void)
功能描述	使能外部高速晶振时钟监视器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable the HXTAL clock monitor */
hal_rcu_hxtal_clock_monitor_enable();
```

函数 hal_rcu_hxtal_clock_monitor_disable

函数hal_rcu_hxtal_clock_monitor_disable描述见下表：

表 3-391. 函数 hal_rcu_hxtal_clock_monitor_disable

函数名称	hal_rcu_hxtal_clock_monitor_disable
函数原型	void hal_rcu_hxtal_clock_monitor_disable(void)
功能描述	失能外部高速晶振时钟监视器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL clock monitor */
hal_rcu_hxtal_clock_monitor_disable();
```

函数 hal_rcu_periph_reset_enable

函数hal_rcu_periph_reset_enable描述见下表：

表 3-392. 函数 hal_rcu_periph_reset_enable

函数名称	hal_rcu_periph_reset_enable
函数原型	void hal_rcu_periph_reset_enable(hal_rcu_periph_reset_enum periph_reset)
功能描述	外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	指向RCU设备信息枚举类型，枚举类型成员参考 表3-348. 枚举 hal_rcu_periph_reset_enum
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* reset the peripherals */
```

```
hal_rcu_periph_reset_enable(RCU_GPIOARST);
```

函数 hal_rcu_periph_reset_disable

函数hal_rcu_periph_reset_disable描述见下表:

表 3-393. 函数 hal_rcu_periph_reset_disable

函数名称	hal_rcu_periph_reset_disable
函数原型	void hal_rcu_periph_reset_disable(hal_rcu_periph_reset_enum periph_reset)
功能描述	失能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	指向RCU设备信息枚举类型, 枚举类型成员参考 表3-348. 枚举 hal_rcu_periph_reset_enum
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable reset the peripherals */
```

```
hal_rcu_periph_reset_disable(RCU_GPIOARST);
```

函数 hal_rcu_periph_clock_config

函数hal_rcu_periph_clock_config描述见下表:

表 3-394. 函数 hal_rcu_periph_clock_config

函数名称	hal_rcu_periph_clock_config
函数原型	int32_t hal_rcu_periph_clock_config(hal_rcu_periphclk_struct *periph_clk)
功能描述	配置RCU扩展外设(RTC, Usart0, ADC, USBFS, CEC) 时钟
先决条件	-
被调用函数	hal_rcu_periph_clk_enable hals_pmu_backup_write_enable hal_sys_basetick_count_get hal_sys_basetick_timeout_check

	<hals_rcu_bkp_reset_enable< hals_rcu_bkp_reset_disable<br=""></hals_rcu_bkp_reset_enable<> hals_rcu_osc_stab_wait hals_rcu_rtc_clock_config hal_rcu_periph_clk_disable hals_rcu_usart_clock_config hals_rcu_adc_clock_config hals_rcu_cec_clock_config hals_rcu_usbfs_clock_config
输入参数{in}	
periph_clk	指向RCU设备信息结构体的指针，结构体成员参考 表3-373. 结构体 <i>hal_rcu_periphclk_struct</i>
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_TIMEOUT, HAL_ERR_VAL, HAL_ERR_NONE

例如：

```

/* initialize the RCU extended peripherals(RTC, Usart0, ADC, USBFS, CEC) clocks */
hal_rcu_periphclk_struct rcu_periphclk_parameter;
rcu_periphclk_parameter.periph_clock_type = RCU_PERIPH_CLKTYPE_ADC;
rcu_periphclk_parameter.adc_clock_source = RCU_ADCK_APB2_DIV2;
if(HAL_ERR_NONE != hal_rcu_periph_clock_config(&rcu_periphclk_parameter)){
    while(1);
}

```

函数 hal_rcu_periph_clkfreq_get

函数hal_rcu_periph_clkfreq_get描述见下表：

表 3-395. 函数 hal_rcu_periph_clkfreq_get

函数名称	hal_rcu_periph_clkfreq_get
函数原型	uint32_t hal_rcu_periph_clkfreq_get(uint32_t periph_clk)
功能描述	获取外设时钟频率
先决条件	-
被调用函数	hals_rcu_clock_freq_get
输入参数{in}	
periph_clk	外设时钟类型
RCU_PERIPH_CLKTY PE_NONE	无外设时钟

RCU_PERIPH_CLKTY PE_RTC	RTC时钟类型
RCU_PERIPH_CLKTY PE_USART0	usart0时钟类型
RCU_PERIPH_CLKTY PE_ADC	ADC时钟类型
RCU_PERIPH_CLKTY PE_APB1TIMER	APB1定时器时钟类型
RCU_PERIPH_CLKTY PE_APB2TIMER	APB2定时器时钟类型
RCU_PERIPH_CLKTY PE_CEC	CEC时钟类型
输出参数{out}	
-	-
返回值	
int32_t	periph_freq

例如：

```
/* get the peripherals clock frequency */
```

```
hal_rcu_periph_clkfreq_get(RCU_PERIPH_CLKTYPE_USART0);
```

函数 hal_rcu_osci_config_get

函数hal_rcu_osci_config_get描述见下表：

表 3-396. 函数 hal_rcu_osci_config_get

函数名称	hal_rcu_osci_config_get
函数原型	void hal_rcu_osci_config_get(hal_rcu_osci_struct *rcu_osci)
功能描述	获取RCU晶振配置
先决条件	-
被调用函数	-
输入参数{in}	
rcu_osci	指向RCU设备信息结构体的指针，结构体成员参考 表3-383. 结构体 hal_rcu_osci_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* get the RCU oscillators configuration */
```

```
hal_rcu_osci_struct rcu_osci_parameter;
```

```
rcu_osci_parameter.htal.need_configure = ENABLE;

rcu_osci_parameter.htal.state = RCU_OSC_BYPASS;

rcu_osci_parameter.pll.need_configure = ENABLE;

rcu_osci_parameter.pll.state = RCU_OSC_ON;

rcu_osci_parameter.pll.pll_source = RCU_PLL_SRC_HXTAL_IRC48M;

rcu_osci_parameter.pll.pll_mul = RCU_PLL_MULT2;

rcu_osci_parameter.pll.pll_presel = RCU_PLL_PRESEL_HXTAL;

rcu_osci_parameter.pll.pre_div = RCU_PLL_PREDIV16;

hal_rcu_osci_config_get(&rcu_osci_parameter);
```

函数 hal_rcu_clock_config

函数hal_rcu_clock_config描述见下表:

表 3-397. 函数 hal_rcu_clock_config

函数名称	hal_rcu_clock_config
函数原型	int32_t hal_rcu_clock_config(hal_rcu_clk_struct *rcu_clk)
功能描述	配置RCU时钟
先决条件	-
被调用函数	hals_rcu_flag_get hals_rcu_system_clock_source_config hal_sys_basetick_count_get hal_sys_basetick_timeout_check hals_rcu_ck48m_clock_config hals_rcu_ahb_clock_config hals_rcu_apb1_clock_config hals_rcu_apb2_clock_config hals_rcu_clock_freq_get hal_sys_timesource_init
输入参数{in}	
rcu_clk	指向RCU设备信息结构体的指针，结构体成员参考 表3-374. 结构体 hal_rcu_clk_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_TIMEOUT, HAL_ERR_NONE

例如:

```
/* configure RCU clock */
```

```

hal_rcu_clk_struct rcu_clk_parameter;

rcu_clk_parameter.clock_type = RCU_CLKTYPE_SYSCLK | RCU_CLKTYPE_AHBCLK |
RCU_CLKTYPE_APB1CLK | RCU_CLKTYPE_APB2CLK;

rcu_clk_parameter.sysclk_source = RCU_SYSCLK_SRC_IRC8M;

rcu_clk_parameter.ahbclk_divider = RCU_SYSCLK_AHBDIV1;

rcu_clk_parameter.apb1clk_divider = RCU_AHBCLK_APB1DIV1;

rcu_clk_parameter.apb2clk_divider = RCU_AHBCLK_APB2DIV1;

if(HAL_ERR_NONE != hal_rcu_clock_config(&rcu_clk_parameter)){

    while(1);

}

```

函数 hal_rcu_clock_config_get

函数hal_rcu_clock_config_get描述见下表：

表 3-398. 函数 hal_rcu_clock_config_get

函数名称	hal_rcu_clock_config_get
函数原型	void hal_rcu_clock_config_get(hal_rcu_clk_struct *rcu_clk)
功能描述	获取RCU时钟配置
先决条件	-
被调用函数	-
输入参数{in}	
rcu_clk	指向RCU设备信息结构体的指针，结构体成员参考 表3-374. 结构体 hal_rcu_clk_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* get configure RCU clock */

hal_rcu_clk_struct rcu_clk_parameter;

rcu_clk_parameter.clock_type = RCU_CLKTYPE_SYSCLK | RCU_CLKTYPE_AHBCLK |
RCU_CLKTYPE_APB1CLK | RCU_CLKTYPE_APB2CLK;

rcu_clk_parameter.sysclk_source = RCU_SYSCLK_SRC_IRC8M;

rcu_clk_parameter.ahbclk_divider = RCU_SYSCLK_AHBDIV1;

rcu_clk_parameter.apb1clk_divider = RCU_AHBCLK_APB1DIV1;

```

```
rcu_clk_parameter.apb2clk_divider = RCU_AHBCLK_APB2DIV1;
```

```
hal_rcu_clock_config_get(&rcu_clk_parameter);
```

函数 hal_SystemCoreClockUpdate

函数hal_SystemCoreClockUpdate描述见下表:

表 3-399. 函数 hal_SystemCoreClockUpdate

函数名称	hal_SystemCoreClockUpdate
函数原型	int32_t hal_SystemCoreClockUpdate(void)
功能描述	系统时钟更新
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
int32_t	g_systemcoreclock

例如:

```
/* update the SystemCoreClock with current core clock retrieved from cpu registers */
```

```
int32_t sys_clk = hal_SystemCoreClockUpdate();
```

函数 hal_rcu_irq

函数hal_rcu_irq描述见下表:

表 3-400. 函数 hal_rcu_irq

函数名称	hal_rcu_irq
函数原型	void hal_rcu_irq(void)
功能描述	RCU中断处理函数
先决条件	-
被调用函数	hals_rcu_interrupt_flag_clear
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* RCU interrupt handler content function,which is merely used in RCU_handler */
```



```
hal_rcu_irq();
```

函数 hal_rcu_irq_handle_set

函数hal_rcu_irq_handle_set描述见下表：

表 3-401. 函数 hal_rcu_irq_handle_set

函数名称	hal_rcu_irq_handle_set
函数原型	void hal_rcu_irq_handle_set(hal_rcu_irq_struct *prcu_irq)
功能描述	设定用户定义的中断回调函数
先决条件	-
被调用函数	hals_rcu_interrupt_enable hals_rcu_interrupt_disable
输入参数{in}	
prcu_irq	指向RCU设备信息结构体的指针，结构体成员参考 表3-375. 结构体 hal_rcu_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set HXTAL stable interrupt handler */
hal_irq_handle_cb hxtal_cb;
hal_rcu_irq_struct hal_rcu_irq;
hal_rcu_irq.hxtal_stable_handle = hxtal_cb;
hal_rcu_irq_handle_set(&hal_rcu_irq);
```

函数 hal_rcu_irq_handle_all_reset

函数hal_rcu_irq_handle_all_reset描述见下表：

表 3-402. 函数 hal_rcu_irq_handle_all_reset

函数名称	hal_rcu_irq_handle_all_reset
函数原型	void hal_rcu_irq_handle_all_reset(void);
功能描述	复位用户定义的中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	

-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset all user-defined callback functions */
```

```
hal_rcu_irq_handle_all_reset();
```

3.19. RTC

实时时钟RTC通常被用作时钟日历。位于备份域中的RTC电路，包含一个32位的累加计数器、一个闹钟、一个预分频器、一个分频器以及RTC时钟配置寄存器。章节[3.19.1](#)描述了RTC的寄存器列表，章节[3.19.2](#)对RTC库函数进行说明。

3.19.1. 外设寄存器描述

RTC寄存器列表如下表所示:

表 3-403. RTC 寄存器

寄存器名称	寄存器描述
RTC_TIME	RT时间寄存器
RTC_DATE	RTC日期寄存器
RTC_CTL	RTC控制寄存器
RTC_STAT	RTC状态寄存器
RTC_PSC	RTC预分频寄存器
RTC_ALRM0TD	RTC闹钟0时间日期寄存器
RTC_WPK	RTC写保护钥匙寄存器
RTC_SS	RTC亚秒寄存器
RTC_SHIFTCTL	RTC移位控制寄存器
RTC_TTS	RTC时间戳时间寄存器
RTC_DTS	RTC时间戳日期寄存器
RTC_SSTS	RTC时间戳亚秒寄存器
RTC_HRFC	RTC高精度频率补偿寄存器
RTC_TAMP	RTC侵入寄存器
RTC_ALRM0SS	RTC闹钟0亚秒寄存器
RTC_BKP0	RTC备份域寄存器0
RTC_BKP1	RTC备份域寄存器1
RTC_BKP2	RTC备份域寄存器2

寄存器名称	寄存器描述
RTC_BKP3	RTC备份域寄存器3
RTC_BKP4	RTC备份域寄存器4

3.19.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-404. RTC 库函数

库函数名称	库函数描述
hal_rtc_init	初始化RTC
hal_rtc_alarm_output_config	配置RTC闹钟输出功能
hal_rtc_alarm_config	配置RTC闹钟功能
hal_rtc_tamp_config	配置RTC侵入功能
hal_rtc_timestamp_config	配置RTC时间戳功能
hal_rtc_calib_config	配置RTC校准功能
hal_rtc_refclock_detection_config	配置RTC参考时钟检测功能
hal_rtc_struct_init	使用默认值初始化RTC结构体
hal_rtc_deinit	复位RTC设备并初始化结构体
hal_rtc_interrupt_enable	使能RTC中断
hal_rtc_interrupt_disable	禁能RTC中断
hal_rtc_irq	RTC中断处理函数，会在rtc_handler中调用
hal_rtc_irq_handle_set	设置用户定义的中断回调函数
hal_rtc_irq_handle_all_reset	复位所有用户定义的的中断回调函数
hal_rtc_current_time_get	获取当前时间和日期
hal_rtc_alarm_get	获取闹钟时间
hal_rtc_alarm_event_poll	轮询闹钟事件
hal_rtc_timestamp_get	获取RTC时间戳时间和日期
hal_rtc_timestamp_event_poll	轮询RTC时间戳事件
hal_rtc_tamper0_event_poll	轮询RTC侵入事件0
hal_rtc_tamper1_event_poll	轮询RTC侵入事件1
hal_rtc_daylight_saving_time_adjust	通过加/减1个小时来调整夏令时

枚举 hal_rtc_state_enum

表 3-405. 枚举 hal_rtc_state_enum

成员名称	功能描述
HAL_RTC_STATE_NONE	无（默认值）
HAL_RTC_STATE_RESET	RTC未被初始化或停止
HAL_RTC_STATE_BUSY	RTC繁忙
HAL_RTC_STATE_TIMEOUT	RTC产生超时
HAL_RTC_STATE_ERROR	RTC出错
HAL_RTC_STATE_READY	RTC准备

枚举 hal_rtc_error_enum

表 3-406. 枚举 hal_rtc_error_enum

成员名称	功能描述
HAL_RTC_ERROR_NONE	无错误
HAL_RTC_ERROR_SYSTEM	RTC内部错误
HAL_RTC_ERROR_CONFIG	RTC配置错误发生

枚举 hal_rtc_struct_type_enum

表 3-407. 枚举 hal_rtc_struct_type_enum

成员名称	功能描述
HAL_RTC_INIT_STRUCT	RTC初始化结构体
HAL_RTC_ALARM_OUTPUT_CONFIG_STRUCT	RTC闹钟输出配置结构体
HAL_RTC_ALARM_CONFIG_STRUCT	RTC闹钟配置结构体
HAL_RTC_TAMPER_CONFIG_STRUCT	RTC侵入配置结构体
HAL_RTC_TIMESTAMP_STRUCT	RTC时间戳结构体
HAL_RTC_IRQ_STRUCT	RTC irq结构体
HAL_RTC_DEV_STRUCT	RTC设备结构体

结构体 hal_rtc_irq_struct

表 3-408. 结构体 hal_rtc_irq_struct

成员名称	功能描述
rtc_timestamp_handle	RTC时间戳处理函数
rtc_alarm_handle	RTC闹钟处理函数
rtc_tamper0_handle	RTC侵入0处理函数
rtc_tamper1_handle	RTC侵入1处理函数

结构体 hal_rtc_dev_struct

表 3-409. 结构体 hal_rtc_dev_struct

成员名称	功能描述
tsi_irq	RTC中断回调函数结构体指针
error_state	RTC错误状态
state	RTC状态
mutex	互斥量
priv	私有数据

结构体 hal_rtc_timestamp_struct

表 3-410. 结构体 hal_rtc_timestamp_struct

成员名称	功能描述
rtc_timestamp_month	RTC时间戳月份
rtc_timestamp_date	RTC时间戳日期
rtc_timestamp_weekday	RTC时间戳星期
rtc_timestamp_hour	RTC时间戳小时
rtc_timestamp_minute	RTC 时间戳分钟
rtc_timestamp_second	RTC 时间戳秒
rtc_timestamp_subsecond	RTC 时间戳亚秒
rtc_am_pm	RTC 时间戳 AM/PM

结构体 hal_rtc_init_struct

表 3-411. 结构体 hal_rtc_init_struct

成员名称	功能描述
rtc_year	RTC年份
rtc_month	RTC月份
rtc_date	RTC日期
rtc_day_of_week	RTC星期
rtc_hour	RTC 小时
rtc_minute	RTC 分钟
rtc_second	RTC 秒
rtc_subsecond	RTC 亚秒
rtc_am_pm	RTC AM/PM
rtc_daylight_saving	RTC 夏令时
rtc_clock_format	RTC 时钟显示格式
rtc_psc_factor_s	RTC 同步分频因子
rtc_psc_factor_a	RTC 异步分频器因子

结构体 hal_rtc_alarm_output_config_struct

表 3-412. 结构体 hal_rtc_alarm_output_config_struct

成员名称	功能描述
rtc_alarm_output_polarity	闹钟输出极性
rtc_alarm_output_type	闹钟输出类型 / PC13输出值

结构体 hal_rtc_alarm_config_struct

表 3-413. 结构体 hal_rtc_alarm_config_struct

成员名称	功能描述
rtc_alarm_am_pm	RTC闹钟AM/PM
rtc_alarm_weekday_mask	RTC闹钟日期掩码

成员名称	功能描述
rtc_alarm_hour_mask	RTC闹钟小时掩码
rtc_alarm_minute_mask	RTC闹钟分钟掩码
rtc_alarm_second_mask	RTC 闹钟秒掩码
rtc_alarm_subsecond_mask	RTC 闹钟亚秒掩码
rtc_weekday_or_date	RTC 闹钟星期/日期
rtc_alarm_day	RTC 闹钟日期
rtc_alarm_hour	RTC 闹钟小时
rtc_alarm_minute	RTC 闹钟分钟
rtc_alarm_second	RTC 闹钟秒
rtc_alarm_subsecond	RTC 闹钟亚秒

结构体 hal_rtc_tamper_config_struct

表 3-414. 结构体 hal_rtc_tamper_config_struct

成员名称	功能描述
rtc_tamper_filter	RTC侵入滤波器
rtc_tamper_sample_frequency	RTC侵入采样频率
rtc_tamper_precharge_time	预充电使能下的RTC侵入预充电时间
rtc_tamper_precharge_enable	电压电平检测下的RTC预充电功能
rtc_tamper_with_timestamp	带有侵入检测的 RTC 时间戳功能
rtc_tamper0_trigger	RTC 侵入 0 触发
rtc_tamper1_trigger	RTC 侵入 1 触发
rtc_tamper0_source	RTC 侵入 0
rtc_tamper1_source	RTC 侵入 1

函数 hal_rtc_init

函数hal_rtc_init描述见下表：

表 3-415. 函数 hal_rtc_init

函数名称	hal_rtc_init
函数原型	int32_t hal_rtc_init(hal_rtc_dev_struct *rtc_dev, hal_rtc_init_struct *rtc_init);
功能描述	初始化RTC
先决条件	-
被调函数	-
输入参数{in}	
rtc_dev	指向RTC设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输入参数{in}	
rtc_init	指向RTC初始化结构体的指针，结构体成员参考 表3-411. 结构体 hal_rtc_init_struct
输出参数{out}	
-	-

返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* initialize RTC */
```

```
hal_rtc_dev_struct rtc_info;
```

```
hal_rtc_init_struct rtc_init_parameter;
```

```
rtc_init_parameter.rtc_hour = 16;
```

```
rtc_init_parameter.rtc_minute = 30;
```

```
rtc_init_parameter.rtc_second = 0;
```

```
rtc_init_parameter.rtc_daylight_saving = RTC_DAYLIGHT_SAVING_NONE;
```

```
rtc_init_parameter.rtc_year = 23;
```

```
rtc_init_parameter.rtc_month = RTC_JUL;
```

```
rtc_init_parameter.rtc_date = 5;
```

```
rtc_init_parameter.rtc_day_of_week = RTC_WEDSDAY;
```

```
rtc_init_parameter.rtc_clock_format = RTC_24H_FORMAT;
```

```
rtc_init_parameter.rtc_psc_factor_a = 127;
```

```
rtc_init_parameter.rtc_psc_factor_s = 255;
```

```
hal_rtc_init(&rtc_info,&rtc_init_parameter);
```

函数 hal_rtc_alarm_output_config

函数hal_rtc_alarm_output_config描述见下表:

表 3-416. 函数 hal_rtc_alarm_output_config

函数名称	hal_rtc_alarm_output_config
函数原型	int32_t hal_rtc_alarm_output_config(hal_rtc_dev_struct *rtc_dev, hal_rtc_alarm_output_config_struct *rtc_alarm_output_config);
功能描述	配置RTC闹钟输出功能
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输入参数{in}	
rtc_alarm_output_config	指向hal_rtc_alarm_output_config_struct的指针，结构体成员参考 表3-416. 函数 hal_rtc_alarm_output_config

输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* configure RTC alarm output */

hal_rtc_dev_struct rtc_info;

hal_rtc_alarm_output_config_struct rtc_alarm_output_config;

rtc_alarm_output_config.rtc_alarm_output_polarity = RTC_ALARM_HIGH;

rtc_alarm_output_config.rtc_alarm_output_type = RTC_ALARM_OUTPUT_OD;

hal_rtc_alarm_output_config (&rtc_info,& rtc_alarm_output_config);
```

函数 hal_rtc_alarm_config

函数hal_rtc_alarm_config描述见下表:

表 3-417. 函数 hal_rtc_alarm_config

函数名称	hal_rtc_alarm_config
函数原型	int32_t hal_rtc_alarm_config(hal_rtc_dev_struct *rtc_dev, hal_rtc_alarm_config_struct *rtc_alarm_config);
功能描述	配置RTC闹钟
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输入参数{in}	
rtc_alarm_config	指向hal_rtc_alarm_config_struct的指针，结构体成员参考 表3-413. 结构体 hal_rtc_alarm_config_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE, HAL_ERR_VAL, HAL_ERR_TIMEOUT

例如:

```
/* configure RTC alarm */

hal_rtc_dev_struct rtc_info;

rtc_alarm_config rtc_alarm_config_parameter;
```



```

rtc_alarm_config_parameter.rtc_alarm_hour = 16;

rtc_alarm_config_parameter.rtc_alarm_minute = 30;

rtc_alarm_config_parameter.rtc_alarm_second = 15;

rtc_alarm_config_parameter.rtc_alarm_subsecond = 0;

rtc_alarm_config_parameter.rtc_alarm_hour_mask =
RTC_ALARM_HOUR_MASK_ENABLE;

rtc_alarm_config_parameter.rtc_alarm_minute_mask =
RTC_ALARM_MINUTE_MASK_ENABLE;

rtc_alarm_config_parameter.rtc_alarm_second_mask =
RTC_ALARM_SECOND_MASK_DISABLE;

rtc_alarm_config_parameter.rtc_alarm_subsecond_mask = RTC_MASKSSC_0_14;

rtc_alarm_config_parameter.rtc_alarm_weekday_mask =
RTC_ALARM_WEEKDAY_MASK_ENABLE;

rtc_alarm_config_parameter.rtc_weekday_or_date = RTC_ALARM_SELECT_DATE;

rtc_alarm_config_parameter.rtc_alarm_day = 1;

hal_rtc_alarm_config(&rtc_info,&rtc_alarm_config_parameter);

```

函数 hal_rtc_tamp_config

函数hal_rtc_tamp_config描述见下表：

表 3-418. 函数 hal_rtc_tamp_config

函数名称	hal_rtc_tamp_config
函数原型	int32_t hal_rtc_tamp_config(hal_rtc_dev_struct *rtc_dev, hal_rtc_tamper_config_struct *rtc_tamper_config);
功能描述	配置RTC侵入检测功能
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输入参数{in}	
rtc_tamper_config	指向hal_rtc_tamper_config_struct的指针，结构体成员参考 表3-414. 结构体 hal_rtc_tamper_config_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* configure RTC tamper */

hal_rtc_dev_struct rtc_info;

hal_rtc_tamper_config_struct rtc_tamper_config_parameter;

rtc_tamper_config_parameter.rtc_tamper_filter = RTC_FLT_EDGE;

rtc_tamper_config_parameter.rtc_tamper_sample_frequency = RTC_FREQ_DIV32768;

rtc_tamper_config_parameter.rtc_tamper_precharge_time = RTC_PRCH_1C;

rtc_tamper_config_parameter.rtc_tamper_precharge_enable = RTC_PRCH_ENALE;

rtc_tamper_config_parameter.rtc_tamper_with_timestamp =
RTC_TAMPER_TIMESTAMP_DISALE;

rtc_tamper_config_parameter.rtc_tamper0_trigger =
RTC_TAMPER_TRIGGER_EDGE_FALLING;

rtc_tamper_config_parameter.rtc_tamper0_source = ENABLE;

rtc_tamper_config_parameter.rtc_tamper1_source = DISABLE;

hal_rtc_tamp_config(&rtc_info,&rtc_tamper_config_parameter);
```

函数 hal_rtc_timestamp_config

函数hal_rtc_timestamp_config描述见下表：

表 3-419. 函数 hal_rtc_timestamp_config

函数名称	hal_rtc_timestamp_config
函数原型	int32_t hal_rtc_timestamp_config(hal_rtc_dev_struct *rtc_dev, uint32_t rtc_timestamp_config);
功能描述	配置RTC时间戳功能
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输入参数{in}	
rtc_timestamp_config	RTC时间戳配置
RTC_TIMESTAMP_RISING_EDGE	上升沿为时间戳事件的有效边沿
RTC_TIMESTAMP_FALLING_EDGE	下降沿为时间戳事件的有效边沿
输出参数{out}	

-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* configure RTC time stamp */
```

```
hal_rtc_dev_struct rtc_info;
```

```
hal_rtc_timestamp_config(&rtc_info, RTC_TIMESTAMP_RISING_EDGE);
```

函数 hal_rtc_calib_config

函数hal_rtc_calib_config描述见下表:

表 3-420. 函数 hal_rtc_calib_config

函数名称	hal_rtc_calib_config
函数原型	int32_t hal_rtc_calib_config(hal_rtc_dev_struct *rtc_dev, uint8_t rtc_calib_config);
功能描述	配置RTC校准功能
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输入参数{in}	
rtc_calib_config	RTC 校准配置
RTC_CALIBRATION_NONE	无校准输出
RTC_CALIBRATION_OUTPUT_512HZ	512HZ校准输出
RTC_CALIBRATION_OUTPUT_1HZ	1HZ校准输出
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* configure RTC calibration */
```

```
hal_rtc_dev_struct rtc_info;
```

```
hal_rtc_calib_config(&rtc_info, RTC_CALIBRATION_OUTPUT_512HZ);
```

函数 hal_rtc_refclock_detection_config

函数hal_rtc_refclock_detection_config描述见下表：

表 3-421. 函数 hal_rtc_refclock_detection_config

函数名称	hal_rtc_refclock_detection_config
函数原型	int32_t hal_rtc_refclock_detection_config(hal_rtc_dev_struct *rtc_dev, ControlStatus refclock_detection);
功能描述	配置RTC参考时钟检测功能
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输入参数{in}	
refclock_detection	RTC参考时钟检测
DISABLE	禁能RTC参考时钟检测
ENABLE	使能RTC参考时钟检测
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* configure RTC reference clock detection function */
hal_rtc_dev_struct rtc_info;
hal_rtc_refclock_detection_config(&rtc_info, ENABLE);
```

函数 hal_rtc_struct_init

函数hal_rtc_struct_init描述见下表：

表 3-422. 函数 hal_rtc_struct_init

函数名称	hal_rtc_struct_init
函数原型	void hal_rtc_struct_init(hal_rtc_struct_type_enum hal_struct_type, void *p_struct);
功能描述	使用默认值初始化RTC结构体
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输入参数{in}	

p_struct	指向包含RTC配置信息的结构体
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* initialize RTC init structure */
```

```
hal_rtc_dev_struct rtc_info;
```

```
hal_rtc_struct_init(HAL_RTC_DEV_STRUCT, &rtc_info);
```

函数 hal_rtc_deinit

函数hal_rtc_deinit描述见下表:

表 3-423. 函数 hal_rtc_deinit

函数名称	hal_rtc_deinit
函数原型	int32_t hal_rtc_deinit(hal_rtc_dev_struct *rtc_dev);
功能描述	复位RTC设备并初始化结构体
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针, 结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如:

```
/* deinitialize RTC */
```

```
hal_rtc_dev_struct rtc_info;
```

```
hal_rtc_deinit(&rtc_info);
```

函数 hal_rtc_interrupt_enable

函数hal_rtc_interrupt_enable描述见下表:

表 3-424. 函数 hal_rtc_interrupt_enable

函数名称	hal_rtc_interrupt_enable
函数原型	int32_t hal_rtc_interrupt_enable(hal_rtc_dev_struct *rtc_dev, hal_rtc_irq_struct *p_irq);
功能描述	使能RTC中断

先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输入参数{in}	
p_irq	指向hal_rtc_irq_struct的指针，结构体成员参考 表3-408. 结构体 hal_rtc_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* enable RTC interrupt */
hal_rtc_dev_struct rtc_info;
hal_rtc_irq_struct rtc_irq;
hal_rtc_interrupt_enable(&rtc_info, &rtc_irq);
```

函数 hal_rtc_interrupt_disable

函数hal_rtc_interrupt_disable描述见下表：

表 3-425. 函数 hal_rtc_interrupt_disable

函数名称	hal_rtc_interrupt_disable
函数原型	int32_t hal_rtc_interrupt_disable(hal_rtc_dev_struct *rtc_dev);
功能描述	禁能RTC中断
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* disable RTC interrupt */
hal_rtc_dev_struct rtc_info;
hal_rtc_interrupt_disable(&rtc_info);
```

函数 hal_rtc_irq

函数hal_rtc_irq描述见下表：

表 3-426. 函数 hal_rtc_irq

函数名称	hal_rtc_irq
函数原型	void hal_rtc_irq(hal_rtc_dev_struct *rtc_dev);
功能描述	RTC中断处理函数，会在rtc_handlerh中调用
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the function is used in the relative interrupt routine */
```

```
hal_rtc_dev_struct rtc_info;
```

```
void RTC_IRQHandler(void)
```

```
{
```

```
    hal_rtc_irq(&rtc_info);
```

```
}
```

函数 hal_rtc_irq_handle_set

函数hal_rtc_irq_handle_set描述见下表：

表 3-427. 函数 hal_rtc_irq_handle_set

函数名称	hal_rtc_irq_handle_set
函数原型	void hal_rtc_irq_handle_set(hal_rtc_dev_struct *rtc_dev, hal_rtc_irq_struct *p_irq);
功能描述	设置用户定义的中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输入参数{in}	
p_irq	指向hal_rtc_irq_struct的指针，结构体成员参考 表3-408. 结构体

	hal_rtc_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set RTC interrupt callback function */
```

```
hal_rtc_dev_struct rtc_info;
```

```
hal_rtc_irq_struct rtc_irq;
```

```
hal_rtc_irq_handle_set(&rtc_info, &rtc_irq);
```

函数 hal_rtc_irq_handle_all_reset

函数hal_rtc_irq_handle_all_reset描述见下表：

表 3-428. 函数 hal_rtc_irq_handle_all_reset

函数名称	hal_rtc_irq_handle_all_reset
函数原型	void hal_rtc_irq_handle_all_reset(hal_rtc_dev_struct *rtc_dev);
功能描述	复位所有用户定义的中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal_rtc_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure RTC reference clock detection function */
```

```
hal_rtc_dev_struct rtc_info;
```

```
hal_rtc_irq_handle_all_reset(&rtc_info);
```

函数 hal_rtc_current_time_get

函数hal_rtc_current_time_get描述见下表：

表 3-429. 函数 hal_rtc_current_time_get

函数名称	hal_rtc_current_time_get
函数原型	void hal_rtc_current_time_get(hal_rtc_init_struct *rtc_calendar);

功能描述	获取当前时间和日期
先决条件	-
被调用函数	-
输入参数{in}	
rtc_calendar	指向hal_rtc_init_struct的指针，结构体成员参考 表3 49. 结构体 hal rtc init struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* get rtc current time */
hal_rtc_init_struct rtc_init;
hal_rtc_current_time_get(&rtc_init);
```

函数 hal_rtc_alarm_get

函数hal_rtc_alarm_get描述见下表：

表 3-430. 函数 hal_rtc_alarm_get

函数名称	hal_rtc_alarm_get
函数原型	void hal_rtc_alarm_get(hal_rtc_alarm_config_struct *rtc_alarm_time);
功能描述	获取RTC闹钟时间
先决条件	-
被调用函数	-
输入参数{in}	
rtc_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-409. 结构体 hal rtc dev struct
输入参数{in}	
rtc_alarm_output_config	指向hal_rtc_alarm_output_config_struct的指针，结构体成员参考 表3-416. 函数 hal rtc alarm output config
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* get rtc alarm time */
hal_rtc_alarm_config_struct rtc_alarm_time;
hal_rtc_alarm_get(&rtc_alarm_time);
```

函数 hal_rtc_alarm_event_poll

函数hal_rtc_alarm_event_poll描述见下表：

表 3-431. 函数 hal_rtc_alarm_event_poll

函数名称	hal_rtc_alarm_event_poll
函数原型	int32_t hal_rtc_alarm_event_poll(uint32_t timeout_ms);
功能描述	轮询闹钟事件
先决条件	-
被调用函数	-
输入参数{in}	
timeout_ms	事件轮询超时时间：0x0 – 0xFFFFFFFF
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_TIMEOUT, HAL_ERR_HARDWARE, HAL_ERR_NONE

例如：

```
/* configure RTC alarm output */

/* poll for RTC alarm event with 100 ms timeout */

hal_rtc_alarm_event_poll (100);
```

函数 hal_rtc_timestamp_get

函数hal_rtc_timestamp_get描述见下表：

表 3-432. 函数 hal_rtc_timestamp_get

函数名称	hal_rtc_timestamp_get
函数原型	void hal_rtc_timestamp_get(hal_rtc_timestamp_struct *rtc_timestamp);
功能描述	获取RTC时间戳时间和日期
先决条件	-
被调用函数	-
输入参数{in}	
rtc_timestamp	指向hal_rtc_timestamp_struct的指针，结构体成员参考 表3-410. 结构体 hal_rtc_timestamp_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* get rtc timestamp time */

hal_rtc_timestamp_struct rtc_timestamp;
```

```
hal_rtc_timestamp_get(&rtc_alarm_time);
```

函数 hal_rtc_timestamp_event_poll

函数hal_rtc_timestamp_event_poll描述见下表:

表 3-433. 函数 hal_rtc_timestamp_event_poll

函数名称	hal_rtc_timestamp_event_poll
函数原型	int32_t hal_rtc_timestamp_event_poll(uint32_t timeout_ms);
功能描述	轮询RTC时间戳事件
先决条件	-
被调用函数	-
输入参数{in}	
timeout_ms	事件轮询超时时间: 0x0 - 0xFFFFFFFF
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_TIMEOUT, HAL_ERR_HARDWARE, HAL_ERR_NONE

例如:

```
/* poll for RTC timestamp event with 100 ms timeout */
```

```
hal_rtc_timestamp_event_poll(100);
```

函数 hal_rtc_tamper0_event_poll

函数hal_rtc_tamper0_event_poll描述见下表:

表 3-434. 函数 hal_rtc_tamper0_event_poll

函数名称	hal_rtc_tamper0_event_poll
函数原型	int32_t hal_rtc_tamper0_event_poll(uint32_t timeout_ms);
功能描述	轮询RTC侵入0事件
先决条件	-
被调用函数	-
输入参数{in}	
timeout_ms	事件轮询超时时间: 0x0 - 0xFFFFFFFF
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_TIMEOUT, HAL_ERR_HARDWARE, HAL_ERR_NONE

例如:

```
/* poll for RTC tamper0 event with 100 ms timeout */
```

```
hal_rtc_tamper0_event_poll(100);
```

函数 hal_rtc_tamper1_event_poll

函数hal_rtc_tamper1_event_poll描述见下表：

表 3-435. 函数 hal_rtc_tamper1_event_poll

函数名称	hal_rtc_tamper1_event_poll
函数原型	int32_t hal_rtc_tamper1_event_poll(uint32_t timeout_ms);
功能描述	轮询RTC侵入1事件
先决条件	-
被调用函数	-
输入参数{in}	
timeout_ms	事件轮询超时时间：0x0 - 0xFFFFFFFF
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_TIMEOUT, HAL_ERR_HARDWARE, HAL_ERR_NONE

例如：

```
/* poll for RTC tamper1 event with 100 ms timeout */
```

```
hal_rtc_tamper1_event_poll(100);
```

函数 hal_rtc_daylight_saving_time_adjust

函数hal_rtc_daylight_saving_time_adjust描述见下表：

表 3-436. 函数 hal_rtc_daylight_saving_time_adjust

函数名称	hal_rtc_daylight_saving_time_adjust
函数原型	void hal_rtc_daylight_saving_time_adjust(uint32_t operation, uint32_t record);
功能描述	通过加/减1小时来调整夏令时
先决条件	-
被调用函数	-
输入参数{in}	
operation	小时调整操作
RTC_DAYLIGHT_SAVING_ADD_1H	加1小时
RTC_DAYLIGHT_SAVING_SUB_1H	减1小时
RTC_DAYLIGHT_SAVING_NONE	不调整
输入参数{in}	
record	夏令时调整操作标记
RTC_RECORD_DAYLIGHTSAVING_SET	置位夏令时调整标记

RTC_RECORD_DAYLIGHTSAVING_RESET	复位记夏令时调整标记
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* adjust the daylight saving time by adding one hour from the current time */
```

```
hal_rtc_daylight_saving_time_adjust(RTC_DAYLIGHT_SAVING_ADD_1H,
RTC_RECORD_DAYLIGHTSAVING_SET);
```

3.20. SPI

SPI模块可以通过SPI协议与外部设备进行通信。章节[3.20.1](#)描述了SPI/I2S的寄存器列表，章节[3.20.2](#)对SPI库函数进行说明。

3.20.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

表 3-437. SPI/I2S 寄存器

寄存器名称	寄存器描述
SPI_CTL0	控制寄存器0
SPI_CTL1	控制寄存器1
SPI_STAT	状态寄存器
SPI_DATA	数据寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_RCRC	接收CRC寄存器
SPI_TCRC	发送CRC寄存器
SPI_I2SCTL	I2S控制寄存器
SPI_I2SPSC	I2S时钟分频寄存器
SPI_QCTL	四路SPI控制寄存器

3.20.2. 外设库函数说明

SPI库函数列表如下表所示：

表 3-438. SPI 库函数

库函数名称	库函数描述
hal_spi_deinit	重新初始化SPI设备信息结构体
hal_spi_struct_init	初始化SPI结构体

库函数名称	库函数描述
hal_spi_init	初始化SPI
hal_spi_transmit_poll	发送数据，轮询发送过程和完成状态，该函数为阻塞式
hal_spi_receive_poll	接收数据，轮询接收过程和完成状态该函数为阻塞式
hal_spi_transmit_receive_poll	发送接收数据，轮询发送/接收过程和完成状态，该函数为阻塞式
hal_spi_transmit_interrupt	采用中断方式发送数据，该函数为非阻塞式
hal_spi_receive_interrupt	采用中断方式接收数据，该函数为非阻塞式
hal_spi_transmit_receive_interrupt	采用中断方式发送和接收数据，该函数为非阻塞式
hal_spi_transmit_dma	采用DMA方式发送数据，该函数为非阻塞式
hal_spi_receive_dma	采用DMA方式接收数据，该函数为非阻塞式
hal_spi_transmit_receive_dma	采用DMA方式发送接收数据，该函数为非阻塞式
hal_spi_irq	处理所有SPI中断请求，仅用于spi_handler
hal_spi_irq_handle_set	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
hal_spi_irq_handle_all_reset	复位所有的用户自定义的中断回调函数
hal_spi_start	打开SPI模块
hal_spi_stop	关闭SPI模块
hal_spi_abort	终止SPI模块
hal_spi_dma_pause	暂停SPI的DMA传输
hal_spi_dma_resume	恢复SPI的DMA传输
hal_spi_dma_stop	关闭SPI的DMA传输和DMA通道

枚举 hal_spi_struct_type_enum

表 3-439. 枚举 hal_spi_struct_type_enum

成员名称	功能描述
HAL_SPI_INIT_STRUCT	SPI初始化结构体
HAL_SPI_DEV_STRUCT	SPI设备信息结构体

枚举 hal_spi_run_state_enum

表 3-440. 枚举 hal_spi_run_state_enum

成员名称	功能描述
HAL_SPI_STATE_READY	外设已初始化，可以使用
HAL_SPI_STATE_BUSY	一个内部过程正在进行
HAL_SPI_STATE_BUSY_TX	数据发送过程正在进行
HAL_SPI_STATE_BUSY_RX	数据接收过程正在进行
HAL_SPI_STATE_BUSY_TX_RX	数据发送和接收过程正在进行
HAL_SPI_STATE_ERROR	SPI错误状态
HAL_SPI_STATE_ABORT	SPI中止正在进行

枚举 hal_spi_trans_mode_enum

表 3-441. 枚举 hal_spi_trans_mode_enum

成员名称	功能描述
SPI_TRANSMODE_FULLDUPLEX	SPI发送接收数据全双工通信
SPI_TRANSMODE_RECEIVEONLY	SPI只接收数据
SPI_TRANSMODE_BDRCEIVE	双向接收数据
SPI_TRANSMODE_BDRSMIT	双向发送数据
SPI_TRANSMODE_QUADRECEIVE	四线接收数据
SPI_TRANSMODE_QUADTRANSMIT	四线发送数据

枚举 hal_spi_prescaler_enum

表 3-442. 枚举 hal_spi_prescaler_enum

成员名称	功能描述
SPI_PSC_2	SPI时钟分频因子为2
SPI_PSC_4	SPI时钟分频因子为4
SPI_PSC_8	SPI时钟分频因子为8
SPI_PSC_16	SPI时钟分频因子为16
SPI_PSC_32	SPI时钟分频因子为32
SPI_PSC_64	SPI时钟分频因子为64
SPI_PSC_128	SPI时钟分频因子为128
SPI_PSC_256	SPI时钟分频因子为256

结构体 hal_spi_irq_struct

表 3-443. 结构体 hal_spi_irq_struct

成员名称	功能描述
receive_handler	中断接收处理函数指针
transmit_handler	中断发送处理函数指针
transmit_receive_handler	中断发送接收处理函数指针
error_handler	中断错误处理函数指针

结构体 hal_spi_buffer_struct

表 3-444. 结构体 hal_spi_buffer_struct

成员名称	功能描述
buffer	SPI传输缓冲区指针

length	SPI传输长度
pos	SPI传输位置

结构体 `hal_spi_dev_struct`

表 3-445. 结构体 `hal_spi_dev_struct`

成员名称	功能描述
periph	SPI外设地址
spi_irq	中断回调处理函数结构体变量
p_dma_rx	DMA接收指针
p_dma_tx	DMA发送指针
txbuffer	发送缓冲区
rxbuffer	接收缓冲区
rx_callback	接收完成回调指针
tx_callback	发送完成回调指针
tx_rx_callback	接收发送完成回调指针
state	当前通信过程状态
error_code	当前通信错误状态
mutex	hal_mutex_enum

结构体 `hal_spi_user_callback_struct`

表 3-446. 结构体 `hal_spi_user_callback_struct`

成员名称	功能描述
complete_func	传输完成回调函数指针
error_func	传输错误回调函数指针

结构体 `hal_spi_init_struct`

表 3-447. 结构体 `hal_spi_init_struct`

成员名称	功能描述
device_mode	主机或设备模式配置
trans_mode	传输模式
frame_size	数据帧格式配置
nss	NSS由软件或硬件控制配置
endian	大端或小端模式配置
clock_polarity_phase	相位和极性配置
prescaler	预分频器配置
crc_calculation	CRC计算是否开启
crc_poly	CRC计算多项式
nssp_mode	NSSP模式是否开启
ti_mode	TI模式是否开启

函数 hal_spi_deinit

函数hal_spi_deinit描述见下表:

表 3-448. 函数 hal_spi_deinit

函数名称	hal_spi_deinit
函数原形	void hal_spi_deinit(hal_spi_dev_struct *spi);
功能描述	重新初始化SPI设备信息结构体
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
hal_spi_dev_struct spi0_info;

/* deinitialize the device information structure */

hal_spi_deinit(&spi0_info);
```

函数 hal_spi_struct_init

函数hal_spi_struct_init描述见下表:

表 3-449. 函数 hal_spi_struct_init

函数名称	hal_spi_struct_init
函数原形	void hal_spi_struct_init(hal_spi_struct_type_enum hal_struct_type, void *p_struct);
功能描述	根据初始化结构体
先决条件	-
输入参数{in}	
hal_struct_type	参考 枚举hal_spi_struct_type_enum
输入参数{in}	
p_struct	指向结构体的空指针
输出参数{out}	
-	-
返回值	
-	-

例如:

```
hal_spi_dev_struct spi0_info;

/* initialize a spi device information structure */
```

```
hal_spi_struct_init(HAL_SPI_DEV_STRUCT, &spi0_info);
```

函数 hal_spi_init

函数hal_spi_init描述见下表:

表 3-450. 函数 hal_spi_init

函数名称	hal_spi_init
函数原形	int32_t hal_spi_init(hal_spi_dev_struct *spi, uint32_t periph, hal_spi_init_struct *p_init);
功能描述	初始化SPI
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	
periph	SPI外设
SPIx	x=0,1
输入参数{in}	
p_init	指向初始化结构体的指针，结构体成员参考结构体 结构体hal_spi_init_struct
输出参数{out}	
-	-
返回值	
int32_t	错误代码
HAL_ERR_NONE	无错误
HAL_ERR_ADDRESSES	超出地址范围
HAL_ERR_VAL	非法值

例如:

```
hal_spi_dev_struct spi0_info;
```

```
hal_spi_init_struct spi0_init_parameter;
```

```
/* initialize the structures */
```

```
hal_spi_struct_init(HAL_SPI_DEV_STRUCT, &spi0_info);
```

```
hal_spi_struct_init(HAL_SPI_INIT_STRUCT, &spi0_init_parameter);
```

```
/* initialize SPI0 */
```

```
spi0_init_parameter.trans_mode = SPI_TRANSMODE_FULLDUPLEX;
```

```
spi0_init_parameter.device_mode = SPI_MASTER;
```

```
spi0_init_parameter.frame_size = SPI_FRAME_SIZE_8BIT;
```

```
spi0_init_parameter.clock_polarity_phase = SPI_CLOCK_POLARITY_HIGH_PHASE_2EDGE;
```

```

spi0_init_parameter.nss                = SPI_NSS_SOFT;

spi0_init_parameter.prescale           = SPI_PSC_8;

spi0_init_parameter.endian             = SPI_ENDIAN_MSB;

spi0_init_parameter.crc_calculation    = SPI_CRC_DISABLE;

spi0_init_parameter.ti_mode            = SPI_TIMODE_DISABLE;

spi0_init_parameter.nssp_mode          = SPI_NSSP_DISABLE;

hal_spi_init(&spi0_info, SPI0, &spi0_init_parameter);

```

函数 hal_spi_transmit_poll

函数hal_spi_transmit_poll描述见下表：

表 3-451. 函数 hal_spi_transmit_poll

函数名称	hal_spi_transmit_poll
函数原型	int32_t hal_spi_transmit_poll(hal_spi_dev_struct *spi, uint8_t *p_txbuffer, uint32_t length, uint32_t timeout_ms);
功能描述	发送数据，轮询发送过程和完成状态，该函数为阻塞式
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	
p_txbuffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	错误代码
HAL_ERR_NONE	无错误
HAL_ERR_VAL	非法值
HAL_ERR_BUSY	忙
HAL_ERR_TIMEOUT	超时

例如：

```

#define ARRAYNUM(arr_name)      (uint32_t)(sizeof(arr_name) / sizeof(*(arr_name)))

#define TRANSMIT_SIZE          (ARRAYNUM(txbuffer))

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B,

```

```
0x0C, 0x0D, 0x0E, 0x0F};
```

```
/* transmit using polling mode */
```

```
hal_spi_transmit_poll(&spi0_info, txbuffer, TRANSMIT_SIZE, 0x1FFFF);
```

函数 hal_spi_receive_poll

函数hal_spi_receive_poll描述见下表：

表 3-452. 函数 hal_spi_receive_poll

函数名称	hal_spi_receive_poll
函数原型	int32_t hal_spi_receive_poll(hal_spi_dev_struct *spi, uint8_t *p_rxbuffer, uint32_t length, uint32_t timeout_ms);
功能描述	接收数据，轮询接收过程和完成状态，该函数为阻塞式
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	
p_rxbuffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	错误代码
HAL_ERR_NONE	无错误
HAL_ERR_HARDWARE	外设标志错误
HAL_ERR_VAL	非法值
HAL_ERR_BUSY	忙
HAL_ERR_TIMEOUT	超时

例如：

```
#define TRANSFER_SIZE          16

uint8_t rxbuffer[TRANSFER_SIZE];

/* receive using polling mode */

hal_spi_receive_poll(&spi0_info, rxbuffer, TRANSFER_SIZE, 0x7FFFFFFF);
```

函数 hal_spi_transmit_receive_poll

函数hal_spi_transmit_receive_poll描述见下表：

表 3-453. 函数 hal_spi_transmit_receive_poll

函数名称	hal_spi_transmit_receive_poll
函数原型	int32_t hal_spi_transmit_receive_poll(hal_spi_dev_struct *spi, uint8_t *p_txbuffer, uint8_t *p_rxbuffer, uint32_t length, uint32_t timeout_ms);
功能描述	发送接收数据，轮询发送/接收过程和完成状态，该函数为阻塞式
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	
p_txbuffer	指向发送数据缓冲区的指针
输入参数{in}	
p_rxbuffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	错误代码
HAL_ERR_NONE	无错误
HAL_ERR_VAL	非法值
HAL_ERR_BUSY	忙
HAL_ERR_TIMEOUT	超时
HAL_ERR_HARDWARE	外设标志错误

例如：

```
#define ARRAYNUM(arr_name)      ((uint32_t)(sizeof(arr_name) / sizeof(*(arr_name))))

#define TRANSFER_SIZE          (ARRAYNUM(txbuffer))

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};

uint8_t rxbuffer[TRANSFER_SIZE];

/* transmit and receive using polling mode */

hal_spi_transmit_receive_poll(&spi0_info, txbuffer, rxbuffer, TRANSFER_SIZE, 6000);
```

函数 hal_spi_transmit_interrupt

函数hal_spi_transmit_interrupt描述见下表:

表 3-454. 函数 hal_spi_transmit_interrupt

函数名称	hal_spi_transmit_interrupt
函数原型	int32_t hal_spi_transmit_interrupt(hal_spi_dev_struct *spi, uint8_t *p_txbuffer, \n uint32_t length, hal_spi_user_callback_struct *p_user_func);
功能描述	采用中断方式发送数据, 该函数为非阻塞式
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针, 结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	
p_txbuffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	错误代码
HAL_ERR_NONE	无错误
HAL_ERR_VAL	非法值
HAL_ERR_BUSY	忙

例如:

```
hal_spi_user_callback_struct user_call;

#define ARRAYNUM(arr_name)      (uint32_t)(sizeof(arr_name) / sizeof(*(arr_name)))

#define TRANSMIT_SIZE          (ARRAYNUM(txbuffer))

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B,
0x0C, 0x0D, 0x0E, 0x0F};

void spi_callback(hal_spi_dev_struct *spi)
{
    printf("\n\r  SPI interrupt transmit success ");
}

/* transmit using interrupt mode */

user_call.complete_func = spi_callback;

hal_spi_transmit_interrupt(&spi0_info, txbuffer, TRANSMIT_SIZE, &user_call);
```

函数 hal_spi_receive_interrupt

函数hal_spi_receive_interrupt描述见下表：

表 3-455. 函数 hal_spi_receive_interrupt

函数名称	hal_spi_receive_interrupt
函数原型	int32_t hal_spi_receive_interrupt(hal_spi_dev_struct *spi, uint8_t *p_rxbuffer, \n uint32_t length, hal_spi_user_callback_struct *p_user_func);
功能描述	采用中断方式接收数据，该函数为非阻塞式
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	
p_rxbuffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	错误代码
HAL_ERR_NONE	无错误
HAL_ERR_VAL	非法值
HAL_ERR_BUSY	忙

例如：

```
hal_spi_user_callback_struct user_call;

#define TRANSFER_SIZE                16

uint8_t rxbuffer[TRANSFER_SIZE];

void spi_callback(hal_spi_dev_struct *spi)
{
    printf("\n\r  SPI interrupt receive success ");
}

/* receive using interrupt mode */

user_call.complete_func = spi_callback;

hal_spi_receive_interrupt(&spi0_info, rxbuffer, TRANSFER_SIZE, &user_call);
```

函数 hal_spi_transmit_receive_interrupt

函数hal_spi_transmit_receive_interrupt描述见下表:

表 3-456. 函数 hal_spi_transmit_receive_interrupt

函数名称	hal_spi_transmit_receive_interrupt
函数原型	int32_t hal_spi_transmit_receive_interrupt(hal_spi_dev_struct *spi, uint8_t *p_txbuffer, uint8_t *p_rxbuffer, uint32_t length, hal_spi_user_callback_struct *p_user_func);
功能描述	采用中断方式发送接收数据，在该函数为非阻塞式
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	
p_txbuffer	指向发送数据缓冲区的指针
输入参数{in}	
p_rxbuffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	错误代码
HAL_ERR_NONE	无错误
HAL_ERR_VAL	非法值
HAL_ERR_BUSY	忙

例如:

```
hal_spi_user_callback_struct user_call;

#define ARRAYNUM(arr_name)      ((uint32_t)(sizeof(arr_name) / sizeof(*(arr_name))))

#define TRANSFER_SIZE          (ARRAYNUM(txbuffer))

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B,
0x0C, 0x0D, 0x0E, 0x0F};

uint8_t rxbuffer[TRANSFER_SIZE];

void spi_callback(hal_spi_dev_struct *spi)
{
    printf("\n\r SPI interrupt transmit and receive success ");
}
```



```
/* transmit and receive using interrupt mode */
```

```
user_call.complete_func = spi_callback;
```

```
hal_spi_transmit_receive_interrupt (&spi0_info, txbuffer, rxbuffer, TRANSMIT_SIZE, &user_call);
```

函数 hal_spi_transmit_dma

函数hal_spi_transmit_dma描述见下表:

表 3-457. 函数 hal_spi_transmit_dma

函数名称	hal_spi_transmit_dma
函数原型	int32_t hal_spi_transmit_dma(hal_spi_dev_struct *spi, uint8_t *p_txbuffer, \ uint32_t length, hal_spi_user_callback_struct *p_user_func);
功能描述	采用DMA方式发送数据，该函数为非阻塞式
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	
p_txbuffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	错误代码
HAL_ERR_NONE	无错误
HAL_ERR_VAL	非法值
HAL_ERR_BUSY	忙

例如:

```
hal_spi_user_callback_struct user_call;
```

```
#define ARRAYNUM(arr_name) ((uint32_t)(sizeof(arr_name) / sizeof(*(arr_name))))
```

```
#define TRANSMIT_SIZE (ARRAYNUM(txbuffer))
```

```
uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B,
0x0C, 0x0D, 0x0E, 0x0F};
```

```
void spi_callback(hal_spi_dev_struct *spi)
```

```
{
    printf("\n\r SPI DMA transmit success ");
```

```

}

/* transmit using DMA mode */

user_call.complete_func = spi_callback;

hal_spi_transmit_dma(&spi0_info, txbuffer, TRANSMIT_SIZE, &user_call);

```

函数 hal_spi_receive_dma

函数hal_spi_receive_dma描述见下表：

表 3-458. 函数 hal_spi_receive_dma

函数名称	hal_spi_receive_dma
函数原型	int32_t hal_spi_receive_dma(hal_spi_dev_struct *spi, uint8_t *p_rxbuffer, uint32_t length, hal_spi_user_callback_struct *p_user_func);
功能描述	采用DMA方式接收数据，该函数为非阻塞式
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	
p_rxbuffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	错误代码
HAL_ERR_NONE	无错误
HAL_ERR_VAL	非法值
HAL_ERR_BUSY	忙

例如：

```

hal_spi_user_callback_struct user_call;

#define TRANSFER_SIZE                16

uint8_t rxbuffer[TRANSFER_SIZE];

void spi_callback(hal_spi_dev_struct *spi)
{
    printf("\n\r SPI DMA receive success ");
}

```

```
/* receive using DMA mode */
```

```
user_call.complete_func = spi_callback;
```

```
hal_spi_receive_dma(&spi0_info, rxbuffer, TRANSMIT_SIZE, &user_call);
```

函数 hal_spi_transmit_receive_dma

函数hal_spi_transmit_receive_dma描述见下表：

表 3-459. 函数 hal_spi_transmit_receive_dma

函数名称	hal_spi_transmit_receive_dma
函数原型	int32_t hal_spi_transmit_receive_dma(hal_spi_dev_struct *spi, uint8_t *p_txbuffer, uint8_t *p_rxbuffer, uint32_t length, hal_spi_user_callback_struct *p_user_func);
功能描述	采用DMA方式发送接收数据，该函数为非阻塞式
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	
p_txbuffer	指向发送数据缓冲区的指针
输入参数{in}	
p_rxbuffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	错误代码
HAL_ERR_NONE	无错误
HAL_ERR_VAL	非法值
HAL_ERR_BUSY	忙

例如：

```
hal_spi_user_callback_struct user_call;
```

```
#define ARRAYNUM(arr_name) ((uint32_t)(sizeof(arr_name) / sizeof(*(arr_name))))
```

```
#define TRANSMIT_SIZE (ARRAYNUM(txbuffer))
```

```
uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};
```

```
uint8_t rxbuffer[TRANSFER_SIZE];
```

```
void spi_callback(hal_spi_dev_struct *spi)
{
    printf("\n\r  SPI DMA transmit and receive success ");
}

/* transmit and receive using DMA mode */

user_call.complete_func = spi_callback;

hal_spi_transmit_receive_dma(&spi0_info, txbuffer, rxbuffer ,TRANSMIT_SIZE, &user_call);
```

函数 hal_spi_irq

函数hal_spi_irq描述见下表：

表 3-460. 函数 hal_spi_irq

函数名称	hal_spi_irq
函数原型	void hal_spi_irq(hal_spi_dev_struct * spi);
功能描述	处理所有SPI中断请求，仅用于spi_handler
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* process the SPI interrupt */

hal_spi_irq(&spi0_info);
```

函数 hal_spi_irq_handle_set

函数hal_spi_irq_handle_set描述见下表：

表 3-461. 函数 hal_spi_irq_handle_set

函数名称	hal_spi_irq_handle_set
函数原型	void hal_spi_irq_handle_set(hal_spi_dev_struct *spi, hal_spi_irq_struct *p_irq);
功能描述	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输入参数{in}	

p_irq	指向中断结构体的指针，结构体成员参考 结构体hal_spi_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_spi_irq_struct spi_irq;

hal_spi_struct_init(HAL_SPI_IRQ_INIT_STRUCTURE, &spi_irq);

/* set the spi transmit interrupt handle function */

spi_irq.transmit_handler = _spi_transmit_interrupt;

hal_spi_irq_handle_set(&spi0_info, &spi_irq);
```

函数 hal_spi_irq_handle_all_reset

函数hal_spi_irq_handle_all_reset描述见下表：

表 3-462. 函数 hal_spi_irq_handle_all_reset

函数名称	hal_spi_irq_handle_all_reset
函数原型	void hal_spi_irq_handle_all_reset(hal_spi_dev_struct *spi);
功能描述	复位所有的用户自定义的中断回调函数
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback */

hal_spi_irq_handle_all_reset(&spi0_info);
```

函数 hal_spi_start

函数hal_spi_start描述见下表：

表 3-463. 函数 hal_spi_start

函数名称	hal_spi_start
函数原型	void hal_spi_start(hal_spi_dev_struct *spi);
功能描述	使能SPI模块
先决条件	-

输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* start SPI module */
```

```
hal_spi_start(&spi0_info);
```

函数 hal_spi_stop

函数hal_spi_stop描述见下表：

表 3-464. 函数 hal_spi_stop

函数名称	hal_spi_stop
函数原型	void hal_spi_stop(hal_spi_dev_struct *spi);
功能描述	关闭SPI模块
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stop SPI module */
```

```
hal_spi_stop(&spi0_info);
```

函数 hal_spi_abort

函数hal_spi_abort描述见下表：

表 3-465. 函数 hal_spi_abort

函数名称	hal_spi_abort
函数原型	int32_t hal_spi_abort(hal_spi_dev_struct *spi);
功能描述	终止SPI模块通信
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输出参数{out}	
-	-

返回值	
int32_t	错误代码
<i>HAL_ERR_NONE</i>	无错误
<i>HAL_ERR_ADDRESSES</i>	超出地址范围

例如：

```
/* abort SPI module */
hal_spi_abort(&spi0_info);
```

函数 hal_spi_dma_pause

函数hal_spi_dma_pause描述见下表：

表 3-466. 函数 hal_spi_dma_pause

函数名称	hal_spi_dma_pause
函数原型	void hal_spi_dma_pause(hal_spi_dev_struct *spi);
功能描述	暂停SPI的DMA传输
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* pause SPI DMA transfer */
hal_spi_dma_pause(&spi0_info);
```

函数 hal_spi_dma_resume

函数hal_spi_dma_resume描述见下表：

表 3-467. 函数 hal_spi_dma_resume

函数名称	hal_spi_dma_resume
函数原型	void hal_spi_dma_resume(hal_spi_dev_struct *spi);
功能描述	恢复SPI的DMA传输
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* resume SPI DMA transfer */
hal_spi_dma_resume(&spi0_info);
```

函数 hal_spi_dma_stop

函数hal_spi_dma_stop描述见下表：

表 3-468. 函数 hal_spi_dma_stop

函数名称	hal_spi_dma_stop
函数原型	void hal_spi_dma_stop(hal_spi_dev_struct *spi);
功能描述	关闭SPI的DMA
先决条件	-
输入参数{in}	
spi	指向设备信息结构体的指针，结构体成员参考 结构体hal_spi_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stop SPI DMA transfer */
hal_spi_dma_stop(&spi0_info);
```

3.21. I2S

I2S模块可以通过I2S音频协议与外部设备进行通信。章节[3.21.1](#)描述了SPI/I2S的寄存器列表，章节[3.21.2](#)对I2S库函数进行说明。

3.21.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

表 3-469. SPI/I2S 寄存器

寄存器名称	寄存器描述
SPI_CTL0	控制寄存器0
SPI_CTL1	控制寄存器1
SPI_STAT	状态寄存器
SPI_DATA	数据寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_RCRC	接收CRC寄存器

寄存器名称	寄存器描述
SPI_TCR	发送CRC寄存器
SPI_I2SCTL	I2S控制寄存器
SPI_I2SPSC	I2S时钟分频寄存器
SPI_QCTL	四路SPI控制寄存器

3.21.2. 外设库函数说明

I2S HAL库函数列表如下表所示：

表 3-470. I2S 库函数

库函数名称	库函数描述
hal_i2s_deinit	重新初始化I2S
hal_i2s_struct_init	初始化I2S结构体
hal_i2s_init	初始化I2S
hal_i2s_transmit_poll	采用轮询方式发送数据，该函数为阻塞式
hal_i2s_receive_poll	采用轮询方式接收数据，该函数为阻塞式
hal_i2s_transmit_interrupt	采用中断方式发送数据，在发送完成后调用用户回调函数，该函数为非阻塞式
hal_i2s_receive_interrupt	采用中断方式接收数据，在接收完成后调用用户回调函数，该函数为非阻塞式
hal_i2s_transmit_dma	采用DMA方式发送数据，在发送完成后调用用户回调函数，该函数为非阻塞式
hal_i2s_receive_dma	采用DMA方式接收数据，在接收完成后调用用户回调函数，该函数为非阻塞式
hal_i2s_irq	处理所有I2S中断请求
hal_i2s_irq_handle_set	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
hal_i2s_irq_handle_all_reset	复位所有的用户自定义的中断回调函数
hal_i2s_start	打开I2S模块
hal_i2s_stop	关闭I2S模块
hal_i2s_dma_pause	传输过程中暂停I2S的DMA传输
hal_i2s_dma_resume	传输过程中恢复I2S的DMA传输
hal_i2s_dma_stop	关闭I2S的DMA传输和DMA通道

枚举 hal_i2s_struct_type_enum

表 3-471. 枚举 hal_i2s_init_struct

成员名称	功能描述
HAL_I2S_INIT_STRUCT	I2S初始化结构体
HAL_I2S_DEV_STRUCT	I2S设备信息结构体

枚举 `hal_i2s_run_state_enum`

表 3-472. 枚举 `hal_i2s_run_state_enum`

成员名称	功能描述
<code>HAL_I2S_STATE_RESET</code>	I2S尚未初始化或禁用
<code>HAL_I2S_STATE_READY</code>	I2S初始化并准备使用
<code>HAL_I2S_STATE_BUSY</code>	I2S内部处理正在进行中
<code>HAL_I2S_STATE_BUSY_TX</code>	正在进行数据传输
<code>HAL_I2S_STATE_BUSY_RX</code>	数据接收过程正在进行
<code>HAL_I2S_STATE_TIMEOUT</code>	I2S超时
<code>HAL_I2S_STATE_ERROR</code>	I2S错误状态

枚举 `hal_i2s_standard_enum`

表 3-473. 枚举 `hal_i2s_standard_enum`

成员名称	功能描述
<code>I2S_STD_PHILLIPS</code>	I2S飞利浦标准
<code>I2S_STD_MSB</code>	I2S MSB标准
<code>I2S_STD_LSB</code>	I2S LSB标准
<code>I2S_STD_PCMSHORT</code>	I2S PCM短标准
<code>I2S_STD_PCMLONG</code>	I2S PCM长标准

枚举 `hal_i2s_audiosample_enum`

表 3-474. 枚举 `hal_i2s_audiosample_enum`

成员名称	功能描述
<code>I2S_AUDIOSAMPLE_8K</code>	I2S音频采样率为8KHz
<code>I2S_AUDIOSAMPLE_11K</code>	I2S音频采样率为11KHz
<code>I2S_AUDIOSAMPLE_16K</code>	I2S音频采样率为16KHz
<code>I2S_AUDIOSAMPLE_22K</code>	I2S音频采样率为22KHz
<code>I2S_AUDIOSAMPLE_32K</code>	I2S音频采样率为32KHz
<code>I2S_AUDIOSAMPLE_44K</code>	I2S音频采样率为44KHz
<code>I2S_AUDIOSAMPLE_48K</code>	I2S音频采样率为48KHz
<code>I2S_AUDIOSAMPLE_96K</code>	I2S音频采样率为96KHz
<code>I2S_AUDIOSAMPLE_192K</code>	I2S音频采样率为192KHz

结构体 `hal_i2s_buffer_struct`

表 3-475. 结构体 `hal_i2s_buffer_struct`

成员名称	功能描述
<code>buffer</code>	指向I2S传输缓冲区的指针
<code>length</code>	I2S传输长度
<code>pos</code>	I2S传输位置

结构体 hal_i2s_irq_struct

表 3-476. 结构体 hal_i2s_irq_struct

成员名称	功能描述
receive_handler	中断接收处理函数指针
transmit_handler	中断发送处理函数指针
error_handle	中断错误处理函数指针

结构体 hal_i2s_dev_struct

表 3-477. 结构体 hal_i2s_dev_struct

成员名称	功能描述
periph	SPI/I2S外设地址
i2s_irq	中断回调处理函数结构体变量
p_dma_rx	DMA接收指针，指向DMA设备信息结构体
p_dma_tx	DMA发送指针，指向DMA设备信息结构体
txbuffer	发送缓冲区
rxbuffer	接收缓冲区
rx_callback	接收完成回调指针
tx_callback	发送完成回调指针
error_callback	通信错误回调指针
state	当前通信过程状态
error_code	当前通信错误状态
mutex	hal_mutex_enum

结构体 hal_i2s_user_callback_struct

表 3-478. 结构体 hal_i2s_user_callback_struct

成员名称	功能描述
complete_func	传输完成回调函数指针
error_func	传输错误回调函数指针

结构体 hal_i2s_init_struct

表 3-479. 结构体 hal_i2s_init_struct

成员名称	功能描述
mode	I2S运行模式
standard	I2S标准
frameformat	I2S数据长度和通道长度
mckout	I2S_MCK输出使能
audiosample	I2S音频采样频率
ckpl	I2S空闲状态时钟极性

函数 hal_i2s_deinit

函数hal_i2s_deinit描述见下表:

表 3-480. 函数 hal_i2s_deinit

函数名称	hal_i2s_deinit
函数原形	void hal_i2s_deinit (hal_i2s_dev_struct *i2s);
功能描述	重新初始化I2S
先决条件	-
输入参数{in}	
I2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
hal_i2s_dev_struct i2s0_info;
```

```
/* deinitialize the device information structure */
```

```
hal_i2s_deinit(&i2s0_info);
```

函数 hal_i2s_struct_init

函数hal_i2s_struct_init描述见下表:

表 3-481. 函数 hal_i2s_struct_init

函数名称	hal_i2s_struct_init
函数原形	void hal_i2s_struct_init(hal_i2s_struct_type_enum hal_struct_type, void *p_struct);
功能描述	初始化结构体
先决条件	-
输入参数{in}	
hal_struct_type	结构体类型
HAL_I2S_INIT_STR UCT	初始化结构体
HAL_I2S_DEV_STR UCT	设备信息结构体
输入参数{in}	
p_struct	指向结构体的空指针
输出参数{out}	
-	-
返回值	
-	-

例如:

```
hal_i2s_dev_struct i2s0_info;

/* initialize a i2s device information structure */

hal_i2s_struct_init(HAL_I2S_DEV_STRUCT, &i2s0_info);
```

函数 hal_i2s_init

函数hal_i2s_init描述见下表:

表 3-482. 函数 hal_i2s_init

函数名称	hal_i2s_init
函数原形	int32_t hal_i2s_init(hal_i2s_dev_struct *i2s, uint32_t periph, hal_i2s_init_struct *p_init);
功能描述	初始化I2S
先决条件	-
输入参数{in}	
i2s	指向设备信息结构体的指针, 结构体成员参考 结构体hal_i2s_dev_struct
输入参数{in}	
periph	I2S外设
SPIx	x=0
输入参数{in}	
p_init	指向初始化结构体的指针, 结构体成员参考结构体 结构体hal_i2s_init_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如:

```
hal_i2s_dev_struct i2s0_info;

hal_i2s_init_struct i2s0_init_parameter;

/* initialize the structures */

hal_i2s_struct_init(HAL_I2S_INIT_STRUCT, &i2s0_init_parameter);

hal_i2s_struct_init(HAL_I2S_DEV_STRUCT, &i2s0_info);

/* initialize SPI0 */

i2s0_init_parameter.mode = I2S_MODE_MASTERTX;

i2s0_init_parameter.standard = I2S_STD_PHILLIPS;

i2s0_init_parameter.ckpl = I2S_CKPL_LOW;

i2s0_init_parameter.frameformat = I2S_FRAMEFORMAT_DT16B_CH16B;
```

```
i2s0_init_parameter.mckout = I2S_MCKOUT_DISABLE;

i2s0_init_parameter.audiosample = I2S_AUDIOSAMPLE_8K;

hal_i2s_init(&i2s0_info, SPI0, &i2s0_init_parameter);
```

函数 hal_i2s_transmit_poll

函数hal_i2s_transmit_poll描述见下表:

表 3-483. 函数 hal_i2s_transmit_poll

函数名称	hal_i2s_transmit_poll
函数原型	int32_t hal_i2s_transmit_poll(hal_i2s_dev_struct *i2s, uint16_t *p_buffer, uint16_t length, uint32_t timeout_ms);
功能描述	采用轮询方式发送数据，该函数为阻塞式
先决条件	-
输入参数{in}	
i2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_VAL

例如:

```
#define ARRAYNUM(arr_name) ((uint32_t)(sizeof(arr_name) / sizeof(*(arr_name)))

#define TRANSMIT_SIZE ((ARRAYNUM(txbuffer))

uint16_t txbuffer[] = {0x0001, 0x0203, 0x0405, 0x0607, 0x0809, 0x0A0B, 0x0C0D, 0x0E0F};

/* transmit using polling mode */

hal_i2s_transmit_poll(&i2s0_info, txbuffer, TRANSMIT_SIZE, 0x1FFFF);
```

函数 hal_i2s_receive_poll

函数hal_i2s_receive_poll描述见下表:

表 3-484. 函数 hal_i2s_receive_poll

函数名称	hal_i2s_receive_poll
函数原型	int32_t hal_i2s_receive_poll(hal_i2s_dev_struct *i2s, uint16_t *p_buffer, uint16_t length, uint32_t timeout_ms);

功能描述	采用轮询方式接收数据，该函数为阻塞式
先决条件	-
输入参数{in}	
l2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输入参数{in}	
p_buffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_VAL

例如：

```
#define TRANSFER_SIZE                16

uint16_t rxbuffer[TRANSFER_SIZE];

/* receive using polling mode */

hal_i2s_receive_poll(&i2s0_info, rxbuffer, TRANSFER_SIZE, 0x7FFFFFFF);
```

函数 hal_i2s_transmit_interrupt

函数hal_i2s_transmit_interrupt描述见下表：

表 3-485. 函数 hal_i2s_transmit_interrupt

函数名称	hal_i2s_transmit_interrupt
函数原型	int32_t hal_i2s_transmit_interrupt(hal_i2s_dev_struct *i2s, uint16_t *p_buffer, uint16_t length, hal_i2s_user_callback_struct *p_user_func);
功能描述	采用中断方式发送数据，在发送完成后调用用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
l2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	

int32_t	HAL_ERR_NONE, HAL_ERR_VAL
---------	---------------------------

例如:

```
hal_i2s_user_callback_struct user_call;

#define ARRAYNUM(arr_name)      (uint32_t)(sizeof(arr_name) / sizeof(*(arr_name)))

#define TRANSMIT_SIZE           (ARRAYNUM(txbuffer))

uint16_t txbuffer[] = {0x0001, 0x0203, 0x0405, 0x0607, 0x0809, 0x0A0B, 0x0C0D, 0x0E0F};

void i2s_callback(hal_i2s_dev_struct *i2s)
{
    printf("\n\r  I2S interrupt transmit success ");
}

/* transmit using interrupt mode */

user_call.complete_func = i2s_callback;

hal_i2s_transmit_interrupt(&i2s0_info, txbuffer, TRANSMIT_SIZE,&user_call);
```

函数 hal_i2s_receive_interrupt

函数hal_i2s_receive_interrupt描述见下表:

表 3-486. 函数 hal_i2s_receive_interrupt

函数名称	hal_i2s_receive_interrupt
函数原型	int32_t hal_i2s_receive_interrupt(hal_i2s_dev_struct *i2s, uint16_t *p_buffer, uint16_t length, hal_i2s_user_callback_struct *p_user_func);
功能描述	采用中断方式接收数据，在接收完成后调用用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
i2s	指向设备信息结构的指针，结构体成员参考 结构体hal_i2s_dev_struct
输入参数{in}	
p_buffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_VAL

例如:


```

hal_i2s_user_callback_struct user_call;

#define TRANSFER_SIZE                16

void i2s_callback(hal_i2s_dev_struct *i2s)
{
    printf("\n\r I2S interrupt receive success ");
}

uint16_t rxbuffer[TRANSFER_SIZE];

/* receive using interrupt mode */

user_call.complete_func = i2s_callback;

hal_i2s_receive_interrupt(&i2s0_info, rxbuffer, TRANSFER_SIZE,&user_call);

```

函数 hal_i2s_transmit_dma

函数hal_i2s_transmit_dma描述见下表：

表 3-487. 函数 hal_i2s_transmit_dma

函数名称	hal_i2s_transmit_dma
函数原型	int32_t hal_i2s_transmit_dma(hal_i2s_dev_struct *i2s, uint16_t *p_buffer, uint16_t length, hal_i2s_user_callback_struct *p_user_func);
功能描述	采用DMA方式发送数据，在发送完成后调用用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
I2s	指向设备信息结构的指针，结构体成员参考 结构体hal_i2s_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_VAL

例如：

```

hal_i2s_user_callback_struct user_call;

#define ARRAYNUM(arr_name)    ((uint32_t)(sizeof(arr_name) / sizeof(*(arr_name))))

```

```
#define TRANSMIT_SIZE                (ARRAYNUM(txbuffer))

uint16_t txbuffer[] = {0x0001, 0x0203, 0x0405, 0x0607, 0x0809, 0x0A0B, 0x0C0D, 0x0E0F};

void i2s_callback(hal_i2s_dev_struct *i2s)
{
    printf("\n\r  I2S interrupt DMA transmit success ");
}

/* transmit using DMA mode */

user_call.complete_func = i2s_callback;

hal_i2s_transmit_dma(&i2s0_info, txbuffer, TRANSMIT_SIZE,&user_call);
```

函数 hal_i2s_receive_dma

函数hal_i2s_receive_dma描述见下表:

表 3-488. 函数 hal_i2s_receive_dma

函数名称	hal_i2s_receive_dma
函数原型	int32_t hal_i2s_receive_dma(hal_i2s_dev_struct *i2s, uint16_t *p_buffer, uint16_t length, hal_i2s_user_callback_struct *p_user_func);
功能描述	采用DMA方式接收数据，在接收完成后调用用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
i2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输入参数{in}	
p_buffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_VAL

例如:

```
hal_i2s_user_callback_struct user_call;

#define TRANSFER_SIZE                16

void i2s_callback(hal_i2s_dev_struct *i2s)
{
```

```

printf("\n\r I2S DMA receive success ");

}

uint16_t rxbuffer[TRANSFER_SIZE];

/* receive using DMA mode */

user_call.complete_func = i2s_callback;

hal_i2s_receive_dma(&i2s0_info, rxbuffer, TRANSMIT_SIZE,&user_call);

```

函数 hal_i2s_irq

函数hal_i2s_irq描述见下表：

表 3-489. 函数 hal_i2s_irq

函数名称	hal_i2s_irq
函数原型	void hal_i2s_irq(hal_i2s_dev_struct * i2s);
功能描述	处理所有I2S中断请求
先决条件	-
输入参数{in}	
i2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* process the I2S interrupt */

hal_i2s_irq(&i2s0_info);

```

函数 hal_i2s_irq_handle_set

函数hal_i2s_irq_handle_set描述见下表：

表 3-490. 函数 hal_i2s_irq_handle_set

函数名称	hal_i2s_irq_handle_set
函数原型	void hal_i2s_irq_handle_set(hal_i2s_dev_struct *i2s, hal_i2s_irq_struct *p_irq);
功能描述	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
先决条件	-
输入参数{in}	
i2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输入参数{in}	

p_irq	指向中断结构体的指针，结构体成员参考hal_i2s_irq_stuct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_i2s_irq_struct i2s_irq;

hal_i2s_struct_init(HAL_I2S_IRQ_INIT_STRUCTURE, &i2s_irq);

/* set the i2s transmit interrupt handle function */

i2s_irq.transmit_handler = _i2s_transmit_interrupt;

hal_i2s_irq_handle_set(&i2s0_info, &i2s_irq);
```

函数 hal_i2s_irq_handle_all_reset

函数hal_i2s_irq_handle_all_reset描述见下表：

表 3-491. 函数 hal_i2s_irq_handle_all_reset

函数名称	hal_i2s_irq_handle_all_reset
函数原型	void hal_i2s_irq_handle_all_reset(hal_i2s_dev_struct *i2s);
功能描述	复位所有的用户自定义的中断回调函数
先决条件	-
输入参数{in}	
i2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback */

hal_i2s_irq_handle_all_reset(&i2s0_info);
```

函数 hal_i2s_start

函数hal_i2s_start描述见下表：

表 3-492. 函数 hal_i2s_start

函数名称	hal_i2s_start
函数原型	void hal_i2s_start(hal_i2s_dev_struct *i2s);
功能描述	使能I2S模块
先决条件	-

输入参数{in}	
i2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* start I2S module */
```

```
hal_i2s_start(&i2s0_info);
```

函数 hal_i2s_stop

函数hal_i2s_stop描述见下表：

表 3-493. 函数 hal_i2s_stop

函数名称	hal_i2s_stop
函数原型	void hal_i2s_stop(hal_i2s_dev_struct *i2s);
功能描述	关闭I2S模块
先决条件	-
输入参数{in}	
i2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stop I2S module */
```

```
hal_i2s_stop(&i2s0_info);
```

函数 hal_i2s_dma_pause

函数hal_i2s_dma_pause描述见下表：

表 3-494. 函数 hal_i2s_dma_pause

函数名称	hal_i2s_dma_pause
函数原型	void hal_i2s_dma_pause(hal_i2s_dev_struct *i2s);
功能描述	传输过程中暂停I2S的DMA传输
先决条件	-
输入参数{in}	
i2s	指向设备信息结构体的指针，结构体成员参考 结构体hal_i2s_dev_struct
输出参数{out}	
-	-

返回值	
-	-

例如:

```
/* pause I2S DMA transfer */
```

```
hal_i2s_dma_pause(&i2s0_info);
```

函数 hal_i2s_dma_resume

函数hal_i2s_dma_resume描述见下表:

表 3-495. 函数 hal_i2s_dma_resume

函数名称	hal_i2s_dma_resume
函数原型	void hal_i2s_dma_resume(hal_i2s_dev_struct *i2s);
功能描述	传输过程中恢复I2S的DMA传输
先决条件	-
输入参数{in}	
I2s	指向设备信息结构体的指针, 结构体成员参考 结构体hal_i2s_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* resume I2S DMA transfer */
```

```
hal_i2s_dma_resume(&i2s0_info);
```

函数 hal_i2s_dma_stop

函数hal_i2s_dma_stop描述见下表:

表 3-496. 函数 hal_i2s_dma_stop

函数名称	hal_i2s_dma_stop
函数原型	void hal_i2s_dma_stop(hal_i2s_dev_struct *i2s);
功能描述	关闭I2S的DMA传输和DMA通道
先决条件	-
输入参数{in}	
i2s	指向设备信息结构体的指针, 结构体成员参考 结构体hal_i2s_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* stop I2S DMA transfer */
```

```
hal_i2s_dma_stop(&i2s0_info);
```

3.22. TIMER

定时器含有可编程的一个无符号计数器，支持输入捕获和输出比较，分为五种类型：高级定时器(TIMER0)，通用定时器L0(TIMERx, x=1, 2)，通用定时器L2(TIMER13)，通用定时器L3(TIMER14)，通用定时器L4(TIMERx, x=15, 16)，基本定时器(TIMER5)，不同类型的定时器具体功能有所差别。章节[3.22.1](#)描述了TIMER的寄存器列表，章节[3.22.2](#)对TIMER库函数进行说明。

3.22.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

表 3-497. TIMER 寄存器

寄存器名称	寄存器描述
TIMER_CTL0(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	控制寄存器 0
TIMERx_CTL1(TIMERx, x=0, 1, 2, 5, 14, 15, 16)	控制寄存器 1
TIMERx_SMCFG(TIMERx, x=0, 1, 2, 14)	从模式配置寄存器
TIMERx_DMAINTEN(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	DMA 和中断使能寄存器
TIMERx_INTF(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	中断标志寄存器
TIMERx_SWEVG(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	软件事件产生寄存器
TIMERx_CHCTL0(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	通道控制寄存器 0
TIMERx_CHCTL1(TIMERx, x=0, 1, 2)	通道控制寄存器 1
TIMERx_CHCTL2(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	通道控制寄存器 2
TIMERx_CNT(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	计数器寄存器
TIMERx_PSC(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	预分频寄存器
TIMERx_CAR(TIMERx, x=0, 1, 2, 5, 13, 14, 15, 16)	计数器自动重载寄存器
TIMERx_CREP(TIMERx, x=0, 14, 15, 16)	重复计数寄存器
TIMERx_CH0CV(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	通道 0 捕获/比较寄存器
TIMERx_CH1CV(TIMERx, x=0, 1, 2, 14)	通道 1 捕获/比较寄存器
TIMERx_CH2CV(TIMERx, x=0, 1, 2)	通道 2 捕获/比较寄存器
TIMERx_CH3CV(TIMERx, x=0, 1, 2)	通道 3 捕获/比较寄存器
TIMERx_CCHP(TIMERx, x=0, 14, 15, 16)	互补通道保护寄存器
TIMERx_DMACFG(TIMERx, x=0, 1, 2, 14, 15, 16)	DMA 配置寄存器
TIMERx_DMATB(TIMERx, x=0, 1, 2, 14, 15, 16)	DMA 发送缓冲区寄存器
TIMERx_IRMP(TIMERx, x=13)	通道输入重映射寄存器
TIMERx_CFG(TIMERx, x=0, 1, 2, 13, 14, 15, 16)	配置寄存器

3.22.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-498. TIMER 库函数

库函数名称	库函数描述
hal_timer_init	TIMER时基初始化
hal_timer_input_capture_config	TIMER输入捕获模式配置
hal_timer_output_compare_config	TIMER输出比较模式配置
hal_timer_break_config	TIMER中止功能配置
hal_timer_ocpre_clear_source_config	TIMER OxCPR信号的清除源配置
hal_timer_cio_input_select	TIMER CIO触发输入配置
hal_timer_single_pulse_mode_config	TIMER单脉冲模式配置
hal_timer_clock_source_config	TIMER时钟源配置
hal_timer_slave_mode_config	TIMER从模式配置
hal_timer_decoder_config	TIMER译码器模式配置
hal_timer_hall_sensor_config	TIMER霍尔传感器模式配置
hal_timer_struct_init	初始化TIMER结构体为默认值
hal_timer_deinit	复位TIMER和设备信息结构体
hal_timer_counter_start	TIMER计数器启动
hal_timer_counter_stop	TIMER计数器停止
hal_timer_counter_start_interrupt	TIMER计数器启动，同时使能更新(update)中断
hal_timer_counter_stop_interrupt	TIMER计数器停止，同时禁能更新(update)中断
hal_timer_counter_start_dma	TIMER计数器启动，同时使能更新(update)事件DMA请求
hal_timer_counter_stop_dma	TIMER计数器停止，同时禁能更新(update)事件DMA请求
hal_timer_input_capture_start	TIMER通道的输入捕获模式启动
hal_timer_input_capture_stop	TIMER通道的输入捕获模式停止
hal_timer_input_capture_start_interrupt	TIMER通道的输入捕获模式启动，同时使能比较/捕获中断
hal_timer_input_capture_stop_interrupt	TIMER通道的输入捕获模式停止，同时禁能比较/捕获中断
hal_timer_input_capture_start_dma	TIMER通道的输入捕获模式启动，同时使能比较/捕获DMA请求
hal_timer_input_capture_stop_dma	TIMER通道的输入捕获模式停止，同时禁能比较/捕获DMA请求
hal_timer_output_compare_start	TIMER通道的输出比较模式启动
hal_timer_output_compare_stop	TIMER通道的输出比较模式停止
hal_timer_output_compare_start_interrupt	TIMER通道的输出比较模式启动，同时使能比较/捕获中断
hal_timer_output_compare_stop_interrupt	TIMER通道的输出比较模式停止，同时禁能比较/捕获中断

库函数名称	库函数描述
hal_timer_output_compare_start_dma	TIMER通道的输出比较模式启动，同时使能比较/捕获DMA请求
hal_timer_output_compare_stop_dma	TIMER通道的输出比较模式停止，同时禁能比较/捕获DMA请求
hal_timer_output_compare_complementary_channel_start	TIMER互补通道的输出比较模式启动
hal_timer_output_compare_complementary_channel_stop	TIMER互补通道的输出比较模式停止
hal_timer_output_compare_complementary_channel_start_interrupt	TIMER互补通道的输出比较模式启动，同时使能比较/捕获中断
hal_timer_output_compare_complementary_channel_stop_interrupt	TIMER互补通道的输出比较模式停止，同时禁能比较/捕获中断
hal_timer_output_compare_complementary_channel_start_dma	TIMER互补通道的输出比较模式启动，同时使能比较/捕获DMA请求
hal_timer_output_compare_complementary_channel_stop_dma	TIMER互补通道的输出比较模式停止，同时禁能比较/捕获DMA请求
hal_timer_single_pulse_mode_channel_config	TIMER单脉冲模式通道配置
hal_timer_single_pulse_start	TIMER通道的单脉冲模式启动
hal_timer_single_pulse_stop	TIMER通道的单脉冲模式禁能
hal_timer_single_pulse_start_interrupt	TIMER通道的单脉冲模式启动，同时使能比较/捕获中断
hal_timer_single_pulse_stop_interrupt	TIMER通道的单脉冲模式停止，同时禁能比较/捕获中断
hal_timer_single_pulse_complementary_channel_start	TIMER互补通道的单脉冲模式启动
hal_timer_single_pulse_complementary_channel_stop	TIMER互补通道的单脉冲模式停止
hal_timer_single_pulse_complementary_channel_start_interrupt	TIMER互补通道的单脉冲模式启动，同时使能比较/捕获中断
hal_timer_single_pulse_complementary_channel_stop_interrupt	TIMER互补通道的单脉冲模式停止，同时禁能比较/捕获中断
hal_timer_slave_mode_interrupt_config	TIMER从模式配置，同时使能触发(TRG)中断
hal_timer_decoder_start	TIMER译码器模式启动
hal_timer_decoder_stop	TIMER译码器模式停止
hal_timer_decoder_start_interrupt	TIMER译码器模式启动，同时使能通道比较/捕获中断
hal_timer_decoder_stop_interrupt	TIMER译码器模式停止，同时禁能通道比较/捕获中断
hal_timer_decoder_start_dma	TIMER译码器模式启动，同时使能通道比较/捕获DMA请求
hal_timer_decoder_stop_dma	TIMER译码器模式停止，同时禁能通道比较/捕获DMA请求
hal_timer_hall_sensor_start	TIMER霍尔传感器模式启动
hal_timer_hall_sensor_stop	TIMER霍尔传感器模式停止

库函数名称	库函数描述
hal_timer_hall_sensor_start_interrupt	TIMER霍尔传感器模式启动，同时使能通道比较/捕获中断
hal_timer_hall_sensor_stop_interrupt	TIMER霍尔传感器模式停止，同时禁能通道比较/捕获中断
hal_timer_hall_sensor_start_dma	TIMER霍尔传感器模式启动，同时使能通道比较/捕获DMA请求
hal_timer_hall_sensor_stop_dma	TIMER霍尔传感器模式停止，同时禁能通道比较/捕获DMA请求
hal_timer_dma_transfer_write_start	DMA写模式启动，memory值写入TIMER若干连续寄存器
hal_timer_dma_transfer_write_stop	DMA写模式停止
hal_timer_dma_transfer_read_start	DMA读模式启动，读取TIMER若干连续寄存器值传输到memory
hal_timer_dma_transfer_read_stop	DMA读模式停止
hal_timer_commutation_event_config	TIMER换相事件配置
hal_timer_commutation_event_interrupt_config	TIMER换相事件配置，同时使能换相中断
hal_timer_commutation_event_dma_config	TIMER换相事件配置，同时使能换相DMA请求
hal_timer_irq_handle_set	中断回调函数设置
hal_timer_irq_handle_all_reset	复位所有中断回调函数
hal_timer_irq	中断服务程序

枚举 hal_timer_state_enum

Table 3-4. 枚举 hal_timer_state_enum

成员名称	功能描述
HAL_TIMER_STATE_NONE	无（默认值）
HAL_TIMER_STATE_RESET	TIMER未被初始化
HAL_TIMER_STATE_BUSY	TIMER繁忙
HAL_TIMER_STATE_TIMEOUT	TIMER产生超时
HAL_TIMER_STATE_ERROR	TIMER出错
HAL_TIMER_STATE_READY	TIMER就绪

枚举 hal_timer_error_enum

Table 3-5. 枚举 hal_timer_error_enum

成员名称	功能描述
HAL_TIMER_ERROR_NONE	无错误
HAL_TIMER_ERROR_SYSTEM	TIMER内部错误
HAL_TIMER_ERROR_DMA	TIMER DMA错误
HAL_TIMER_ERROR_CONFIG	配置错误

枚举 `hal_timer_service_channel_enum`

Table 3-6. 枚举 `hal_timer_service_channel_enum`

成员名称	功能描述
<code>HAL_TIMER_SERVICE_CHANNEL_NONE</code>	当前无通道使用
<code>HAL_TIMER_SERVICE_CHANNEL_0</code>	当前使用的通道为通道0
<code>HAL_TIMER_SERVICE_CHANNEL_1</code>	当前使用的通道为通道1
<code>HAL_TIMER_SERVICE_CHANNEL_2</code>	当前使用的通道为通道2
<code>HAL_TIMER_SERVICE_CHANNEL_3</code>	当前使用的通道为通道3

枚举 `hal_timer_struct_type_enum`

Table 3-7. 枚举 `hal_timer_struct_type_enum`

成员名称	功能描述
<code>HAL_TIMER_INIT_STRUCT</code>	TIMER时基初始化配置结构体类型
<code>HAL_TIMER_INPUT_CAPTURE_STRUCT</code>	TIMER输入捕获配置结构体类型
<code>HAL_TIMER_OUTPUT_COMPARE_STRUCT</code>	TIMER输出比较配置结构体类型
<code>HAL_TIMER_BREAK_STRUCT</code>	TIMER中止和互补通道保护配置结构体类型
<code>HAL_TIMER_CLEAR_SOURCE_STRUCT</code>	TIMER OxCPRE清除源配置结构体类型
<code>HAL_TIMER_CLOCK_SOURCE_STRUCT</code>	TIMER时钟源配置结构体类型
<code>HAL_TIMER_SLAVE_MODE_STRUCT</code>	TIMER从模式配置结构体类型
<code>HAL_TIMER_DECODER_STRUCT</code>	TIMER译码器模式配置结构体类型
<code>HAL_TIMER_DECODER_DMA_CONFIG_STRUCT</code>	TIMER译码器模式DMA传输配置结构体类型
<code>HAL_TIMER_HALL_SENSOR_STRUCT</code>	TIMER霍尔传感器模式配置结构体类型
<code>HAL_TIMER_SINGLE_PULSE_STRUCT</code>	TIMER单脉冲模式配置结构体类型
<code>HAL_TIMER_DMA_TRANSFER_CONFIG_STRUCT</code>	TIMER DMA传输配置结构体类型
<code>HAL_TIMER_IRQ_STRUCT</code>	TIMER中断回调函数指针结构体类型
<code>HAL_TIMER_DMA_HANDLE_CB_STRUCT</code>	TIMER DMA中断回调函数指针结构体类型
<code>HAL_TIMER_DEV_STRUCT</code>	TIMER设备信息结构体类型

枚举 `hal_timer_dma_transfer_start_address_enum`

Table 3-8. 枚举 `hal_timer_dma_transfer_start_address_enum`

成员名称	功能描述
<code>TIMER_DMA_START_ADDRESS_CTL0</code>	DMA起始传输地址: <code>TIMER_CTL0</code>
<code>TIMER_DMA_START_ADDRESS_CTL1</code>	DMA起始传输地址: <code>TIMER_CTL1</code>
<code>TIMER_DMA_START_ADDRESS_SMCFG</code>	DMA起始传输地址: <code>TIMER_SMCFG</code>
<code>TIMER_DMA_START_ADDRESS_DMAINTEN</code>	DMA起始传输地址: <code>TIMER_DMAINTEN</code>
<code>TIMER_DMA_START_ADDRESS_INTF</code>	DMA起始传输地址: <code>TIMER_INTF</code>
<code>TIMER_DMA_START_ADDRESS_SWEVG</code>	DMA起始传输地址: <code>TIMER_SWEVG</code>
<code>TIMER_DMA_START_ADDRESS_CHCTL0</code>	DMA起始传输地址: <code>TIMER_CHCTL0</code>

TIMER_DMA_START_ADDRESS_CHCTL1	DMA起始传输地址: TIMER_CHCTL1
TIMER_DMA_START_ADDRESS_CHCTL2	DMA起始传输地址: TIMER_CHCTL2
TIMER_DMA_START_ADDRESS_CNT	DMA起始传输地址: TIMER_CNT
TIMER_DMA_START_ADDRESS_PSC	DMA起始传输地址: TIMER_PSC
TIMER_DMA_START_ADDRESS_CAR	DMA起始传输地址: TIMER_CAR
TIMER_DMA_START_ADDRESS_CREP	DMA起始传输地址: TIMER_CREP
TIMER_DMA_START_ADDRESS_CH0CV	DMA起始传输地址: TIMER_CH0CV
TIMER_DMA_START_ADDRESS_CH1CV	DMA起始传输地址: TIMER_CH1CV
TIMER_DMA_START_ADDRESS_CH2CV	DMA起始传输地址: TIMER_CH2CV
TIMER_DMA_START_ADDRESS_CH3CV	DMA起始传输地址: TIMER_CH3CV
TIMER_DMA_START_ADDRESS_CCHP	DMA起始传输地址: TIMER_CCHP
TIMER_DMA_START_ADDRESS_DMACFG	DMA起始传输地址: TIMER_DMACFG

枚举 hal_timer_dma_transfer_length_enum

Table 3-9. 枚举 hal_timer_dma_transfer_length_enum

成员名称	功能描述
TIMER_DMACFG_DMATC_1TRANSFER	DMA传输1次
TIMER_DMACFG_DMATC_2TRANSFER	DMA传输2次
TIMER_DMACFG_DMATC_3TRANSFER	DMA传输3次
TIMER_DMACFG_DMATC_4TRANSFER	DMA传输4次
TIMER_DMACFG_DMATC_5TRANSFER	DMA传输5次
TIMER_DMACFG_DMATC_6TRANSFER	DMA传输6次
TIMER_DMACFG_DMATC_7TRANSFER	DMA传输7次
TIMER_DMACFG_DMATC_8TRANSFER	DMA传输8次
TIMER_DMACFG_DMATC_9TRANSFER	DMA传输9次
TIMER_DMACFG_DMATC_10TRANSFER	DMA传输10次
TIMER_DMACFG_DMATC_11TRANSFER	DMA传输11次
TIMER_DMACFG_DMATC_12TRANSFER	DMA传输12次
TIMER_DMACFG_DMATC_13TRANSFER	DMA传输13次
TIMER_DMACFG_DMATC_14TRANSFER	DMA传输14次
TIMER_DMACFG_DMATC_15TRANSFER	DMA传输15次
TIMER_DMACFG_DMATC_16TRANSFER	DMA传输16次
TIMER_DMACFG_DMATC_17TRANSFER	DMA传输17次
TIMER_DMACFG_DMATC_18TRANSFER	DMA传输18次

枚举 hal_timer_trgo_selection_enum

Table 3-10. 枚举 hal_timer_trgo_selection_enum

成员名称	功能描述
TIMER_TRI_OUT_SRC_RESET	复位事件（UPG位被置1或从模式产生复位）作为触发输出TRGO
TIMER_TRI_OUT_SRC_ENABL	计数器使能信号作为触发输出TRGO

TIMER_TRI_OUT_SRC_UPDATE	更新事件作为触发输出TRGO
TIMER_TRI_OUT_SRC_CH0	通道0发生捕获或比较成功脉冲作为触发输出TRGO
TIMER_TRI_OUT_SRC_O0CPRE	O0CPRE作为触发输出TRGO
TIMER_TRI_OUT_SRC_O1CPRE	O1CPRE作为触发输出TRGO
TIMER_TRI_OUT_SRC_O2CPRE	O2CPRE作为触发输出TRGO
TIMER_TRI_OUT_SRC_O3CPRE	O3CPRE作为触发输出TRGO

枚举 hal_timer_output_compare_enum

Table 3-11. 枚举 hal_timer_output_compare_enum

成员名称	功能描述
TIMER_OC_MODE_TIMING	时基模式
TIMER_OC_MODE_ACTIVE	匹配时设置为高
TIMER_OC_MODE_INACTIVE	匹配时设置为低
TIMER_OC_MODE_TOGGLE	匹配时翻转
TIMER_OC_MODE_LOW	强制为低
TIMER_OC_MODE_HIGH	强制为高
TIMER_OC_MODE_PWM0	PWM模式0
TIMER_OC_MODE_PWM1	PWM模式1

枚举 hal_timer_clock_source_enum

Table 3-12. 枚举 hal_timer_clock_source_enum

成员名称	功能描述
TIMER_CLOCK_SOURCE_CK_TIMER	时钟输入源: CK_TIMER, 内部时钟
TIMER_CLOCK_SOURCE_ITI0	时钟输入源: ITI0, 外部时钟模式0
TIMER_CLOCK_SOURCE_ITI1	时钟输入源: ITI1, 外部时钟模式0
TIMER_CLOCK_SOURCE_ITI2	时钟输入源: ITI2, 外部时钟模式0
TIMER_CLOCK_SOURCE_ITI3	时钟输入源: ITI3, 外部时钟模式0
TIMER_CLOCK_SOURCE_CIOFE0	时钟输入源: CIOFE0, 外部时钟模式0
TIMER_CLOCK_SOURCE_CI1FE1	时钟输入源: CI1FE1, 外部时钟模式0
TIMER_CLOCK_SOURCE_CIOFED	时钟输入源: CIOFED, 外部时钟模式0
TIMER_CLOCK_SOURCE_ETIMODE0	时钟输入源: ETI, 外部时钟模式0
TIMER_CLOCK_SOURCE_ETIMODE1	时钟输入源: ETI, 外部时钟模式1

枚举 hal_timer_slave_mode_enum

Table 3-13. 枚举 hal_timer_slave_mode_enum

成员名称	功能描述
TIMER_SLAVE_DISABLE_MODE	关闭从模式
TIMER_SLAVE_MODE_RESTART_MODE	复位模式
TIMER_SLAVE_MODE_PAUSE_MODE	暂停模式
TIMER_SLAVE_MODE_EVENT_MODE	事件模式

枚举 `hal_timer_input_trigger_source_enum`

Table 3-14. 枚举 `hal_timer_input_trigger_source_enum`

成员名称	功能描述
<code>TIMER_TRIGGER_SOURCE_ITI0</code>	触发输入源: ITI0
<code>TIMER_TRIGGER_SOURCE_ITI1</code>	触发输入源: ITI1
<code>TIMER_TRIGGER_SOURCE_ITI2</code>	触发输入源: ITI2
<code>TIMER_TRIGGER_SOURCE_ITI3</code>	触发输入源: ITI3
<code>TIMER_TRIGGER_SOURCE_CIOFE0</code>	触发输入源: CIOFE0
<code>TIMER_TRIGGER_SOURCE_CI1FE1</code>	触发输入源: CI1FE1
<code>TIMER_TRIGGER_SOURCE_CIOFED</code>	触发输入源: CIOFED
<code>TIMER_TRIGGER_SOURCE_ETIFP</code>	触发输入源: ETIFP
<code>TIMER_TRIGGER_SOURCE_DISABLE</code>	无触发输入源

枚举 `hal_timer_decoder_mode_enum`

Table 3-15. 枚举 `hal_timer_decoder_mode_enum`

成员名称	功能描述
<code>TIMER_QUADRATURE_DECODER_MODE0</code>	正交译码器模式0
<code>TIMER_QUADRATURE_DECODER_MODE1</code>	正交译码器模式1
<code>TIMER_QUADRATURE_DECODER_MODE2</code>	正交译码器模式2

结构体 `hal_timer_decoder_dma_config_struct`

Table 3-16. 结构体 `hal_timer_decoder_dma_config_struct`

成员名称	功能描述
<code>mem_addr0</code>	TIMER译码器模式DMA传输memory地址0
<code>mem_addr1</code>	TIMER译码器模式DMA传输memory地址1
<code>length</code>	TIMER译码器模式DMA传输长度

结构体 `hal_timer_dma_transfer_config_struct`

Table 3-17. 结构体 `hal_timer_dma_transfer_config_struct`

成员名称	功能描述
<code>start_addr</code>	TIMER DMA传输TIMER起始地址
<code>mem_addr</code>	TIMER DMA传输memory地址
<code>length</code>	TIMER DMA传输长度

结构体 `hal_timer_irq_struct`

Table 3-18. 结构体 `hal_timer_irq_struct`

成员名称	功能描述
<code>update_handle</code>	更新事件中中断回调函数
<code>channelx_capture_handle</code>	通道捕获功能中断回调函数

channelx_compare_handle	通道比较功能中断回调函数
commutation_handle	换相事件中断回调函数
trigger_handle	触发事件中断回调函数
break_handle	中止事件中断回调函数

结构体 hal_timer_dma_handle_cb_struct

Table 3-19. 结构体 hal_timer_dma_handle_cb_struct

成员名称	功能描述
update_dma_full_transcom_handle	更新事件DMA传输完成中断回调函数
channelx_capture_dma_full_transcom_handle	通道捕获功能DMA传输完成中断回调函数
channelx_compare_dma_full_transcom_handle	通道比较功能DMA传输完成中断回调函数
commutation_dma_full_transcom_handle	换相事件DMA传输完成中断回调函数
trigger_dma_full_transcom_handle	触发事件DMA传输完成中断回调函数
error_handle	DMA传输错误中断回调函数

结构体 hal_timer_dev_struct

Table 3-20. 结构体 hal_timer_dev_struct

成员名称	功能描述
periph	当前使用的TIMER(x=0...2,5,13..16)
service_channel	当前使用的通道x
timer_irq	TIMER中断回调函数指针结构体
*p_dma_timer[7]	DMA设备信息结构体指针数组，TIMER共7个DMA请求 (数组索引取值范围：TIMER_DMA_ID_UP, TIMER_DMA_ID_CH0, TIMER_DMA_ID_CH1, TIMER_DMA_ID_CH2, TIMER_DMA_ID_CH3, TIMER_DMA_ID_CMT, TIMER_DMA_ID_TRG)
timer_dma	TIMER DMA中断回调函数指针结构体
error_state	TIMER错误状态
state	TIMER状态
mutex	互斥锁和解锁状态
*priv	隐私数据

结构体 hal_timer_init_struct

Table 3-21. 结构体 hal_timer_init_struct

成员名称	功能描述
prescaler	预分频值，TIMERx_PSC
alignedmode	计数器对齐模式
counter_direction	计数方向
period	周期值，TIMERx_CAR
clock_division	时钟分频因子
repetition_counter	重复计数器值，TIMERx_CREP

autoreload_shadow	自动重载影子使能/禁能, TIMERx_CTL0:bit7(ARSE)
master_slave_mode	主从模式使能/禁能, TIMERx_SMCFG:bit7(MSM)
trgo_selection	主模式控制, TIMERx_CTL1:bit4~6(MMC)

结构体 hal_timer_input_capture_struct

Table 3-22. 结构体 hal_timer_input_capture_struct

成员名称	功能描述
ic_polarity	通道输入捕获极性
ic_selection	通道输入模式选择
ic_prescaler	通道输入捕获预分频
ic_filter	通道输入捕获滤波 (0~15)

结构体 hal_timer_output_compare_struct

Table 3-23. 结构体 hal_timer_output_compare_struct

成员名称	功能描述
compare_mode	通道输出比较模式控制
oc_pulse_value	通道比较值
oc_polarity	通道输出极性
ocn_polarity	互补通道输出极性
oc_idlestate	空闲状态下通道输出
ocn_idlestate	空闲状态下互补通道输出
oc_shadow	通道输出比较影子寄存器使能/禁能
oc_fastmode	通道输出比较快速使能/禁能
oc_clearmode	通道输出比较清0使能/禁能

结构体 hal_timer_break_struct

Table 3-24. 结构体 hal_timer_break_struct

成员名称	功能描述
run_offstate	运行模式下“关闭状态”配置
idel_offstate	空闲模式下“关闭状态”配置
dead_time	死区时间 (0~255)
break_polarity	中止信号极性
output_autostate	自动输出使能
protect_mode	互补寄存器保护控制
break_state	中止使能/禁能

结构体 hal_timer_clear_source_struct

Table 3-25. 结构体 hal_timer_clear_source_struct

成员名称	功能描述
clear_source	TIMER OxCPRE信号清除源选择

exttrigger_polarity	外部触发极性
exttrigger_prescaler	外部触发预分频
exttrigger_filter	外部触发滤波（0~15）

结构体 hal_timer_clock_source_struct

Table 3-26. 结构体 hal_timer_clock_source_struct

成员名称	功能描述
clock_source	TIMER时钟源选择
clock_polarity	时钟输入源极性
clock_prescaler	时钟输入源预分频
clock_filter	时钟输入源滤波（0~15）

结构体 hal_timer_slave_mode_struct

Table 3-27. 结构体 hal_timer_slave_mode_struct

成员名称	功能描述
slavemode	TIMER从模式选择
trigger_selection	触发源选择
trigger_polarity	触发输入源极性
trigger_prescaler	触发输入源预分频
trigger_filter	触发输入源滤波（0~15）

结构体 hal_timer_decoder_struct

Table 3-28. 结构体 hal_timer_decoder_struct

成员名称	功能描述
decoder_mode	译码器模式选择
ci0_polarity	通道0输入捕获极性
ci0_selection	通道0输入模式选择
ci0_prescaler	通道0输入捕获预分频
ci0_filter	通道0输入捕获滤波（0~15）
ci1_polarity	通道1输入捕获极性
ci1_selection	通道1输入模式选择
ci1_prescaler	通道1输入捕获预分频
ci1_filter	通道1输入捕获滤波（0~15）

结构体 hal_timer_hall_sensor_struct

Table 3-29. 结构体 hal_timer_hall_sensor_struct

成员名称	功能描述
cmt_delay	换相延迟（通道1比较值）
ci0_polarity	通道0输入捕获极性
ci0_selection	通道0输入模式选择

ci0_prescaler	通道0输入捕获预分频
ci0_filter	通道0输入捕获滤波（0~15）

结构体 `hal_timer_single_pulse_struct`

Table 3-30. 结构体 `hal_timer_single_pulse_struct`

成员名称	功能描述
sp_compare_mode	单脉冲模式输出通道的比较模式控制
sp_oc_pulse_value	单脉冲模式输出通道的比较值
sp_oc_polarity	单脉冲模式输出通道的极性
sp_ocn_polarity	单脉冲模式互补输出通道的极性
sp_oc_idlestate	单脉冲模式输出通道空闲状态下的输出
sp_ocn_idlestate	单脉冲模式互补输出通道空闲状态下的输出
sp_oc_fastmode	单脉冲模式输出通道的比较快速使能/禁能
sp_oc_clearmode	单脉冲模式输出通道的比较清0使能/禁能
sp_ic_polarity	单脉冲模式输入通道的捕获极性
sp_ic_selection	单脉冲模式输入通道的模式选择
sp_ic_filter	单脉冲模式输入通道的捕获滤波（0~15）

函数 `hal_timer_init`

函数`hal_timer_init`描述见下表：

Table 3-31. 函数 `hal_timer_init`

函数名称	hal_timer_init
函数原形	int32_t hal_timer_init(hal_timer_dev_struct *timer_dev, uint32_t periph, hal_timer_init_struct *timer)
功能描述	TIMER时基初始化
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
periph	定时器外设名称
TIMERx(x=0..2, 5, 13..16)	TIMER外设选择
输入参数{in}	
timer	指针，指向hal_timer_init_struct结构体，结构体成员参考 Table 3-21. 结构体hal_timer_init_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如:

```
/*initialize TIMER timebase */

hal_timer_dev_struct timer0_info;

hal_timer_init_struct timer0_init_parameter;

timer0_init_parameter.prescaler = 0;

timer0_init_parameter.alignedmode = TIMER_COUNTER_EDGE;

timer0_init_parameter.counter_direction = TIMER_COUNTER_UP;

timer0_init_parameter.period = 65535;

timer0_init_parameter.clock_division = TIMER_CKDIV_DIV1;

timer0_init_parameter.repetition_counter = 0;

timer0_init_parameter.autoreload_shadow = TIMER_CARL_SHADOW_DISABLE;

timer0_init_parameter.master_slave_mode = DISABLE;

timer0_init_parameter.trgo_selection = TIMER_TRI_OUT_SRC_RESET;

hal_timer_init(&timer0_info,TIMER0,&timer0_init_parameter);
```

函数 hal_timer_input_capture_config

函数hal_timer_input_capture_config描述见下表:

Table 3-32. 函数 hal_timer_input_capture_config

函数名称	hal_timer_input_capture_config
函数原形	int32_t hal_timer_input_capture_config(hal_timer_dev_struct *timer_dev, uint16_t channel, hal_timer_input_capture_struct *timer_inputcapture)
功能描述	TIMER输入捕获模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针, 指向hal_timer_dev_struct结构体, 结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输入参数{in}	
timer_inputcapture	指针, 指向hal_timer_inputcapture_struct结构体, 结构体成员参考 Table

	3-22. 结构体hal_timer_input_capture_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT

例如:

```
/* configure TIMER input capture mode */
hal_timer_dev_struct timer0_info;
hal_timer_input_capture_struct timer0_input_capture_parameter;
timer0_input_capture_parameter.ic_polarity = TIMER_IC_POLARITY_RISING;
timer0_input_capture_parameter.ic_selection = TIMER_IC_SELECTION_DIRECTTI;
timer0_input_capture_parameter.ic_prescaler = TIMER_IC_PRESCALER_OFF;
timer0_input_capture_parameter.ic_filter = 0;
hal_timer_input_capture_config(&timer0_info,TIMER_CH_0,&timer0_input_capture_parameter);
```

函数 hal_timer_output_compare_config

函数hal_timer_output_compare_config描述见下表:

Table 3-33. 函数 hal_timer_output_compare_config

函数名称	hal_timer_output_compare_config
函数原形	int32_t hal_timer_output_compare_config(hal_timer_dev_struct *timer_dev, uint16_t channel, hal_timer_output_compare_struct *timer_outputcompare)
功能描述	TIMER输出比较模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针, 指向hal_timer_dev_struct结构体, 结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输入参数{in}	
timer_outputcompare	指针, 指向hal_timer_output_compare_struct结构体, 结构体成员参考

Table 3-23. 结构体hal_timer_output_compare_struct	
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT

例如:

```
/* configure TIMER output compare mode: TIMER_OUTPUT_TOGGLE_MODE */
hal_timer_dev_struct timer0_info;

hal_timer_output_compare_struct timer0_output_compare_parameter;

timer0_output_compare_parameter.compare_mode = TIMER_OC_MODE_TOGGLE;

timer0_output_compare_parameter.oc_pulse_value = 500;

timer0_output_compare_parameter.oc_polarity = TIMER_OC_POLARITY_HIGH;

timer0_output_compare_parameter.oc_idlestate = TIMER_OC_IDLE_STATE_LOW;

timer0_output_compare_parameter.oc_shadow = TIMER_OC_SHADOW_DISABLE;

timer0_output_compare_parameter.oc_clearmode = TIMER_OC_CLEAR_DISABLE;

hal_timer_output_compare_config(&timer0_info,TIMER_CH_0,&timer0_output_compare_parameter);
```

函数 hal_timer_break_config

函数hal_timer_break_config描述见下表:

Table 3-34. 函数 hal_timer_break_config

函数名称	hal_timer_break_config
函数原形	int32_t hal_timer_break_config(hal_timer_dev_struct *timer_dev, hal_timer_break_struct *timer_break)
功能描述	TIMER中止功能配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针, 指向hal_timer_dev_struct结构体, 结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
timer_break	指针, 指向hal_timer_break_struct结构体, 结构体成员参考 Table 3-24. 结构体hal_timer_break_struct
输出参数{out}	
-	-

返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT

例如:

```
/* configure TIMER break function */

hal_timer_dev_struct timer0_info;

hal_timer_break_struct timer0_break_parameter;

timer0_break_parameter.run_offstate = TIMER_ROS_STATE_DISABLE;

timer0_break_parameter.idel_offstate = TIMER_IOS_STATE_DISABLE;

timer0_break_parameter.break_polarity = TIMER_BREAK_POLARITY_HIGH;

timer0_break_parameter.output_autostate = TIMER_OUTAUTO_DISABLE;

timer0_break_parameter.protect_mode = TIMER_CCHP_PROT_OFF;

timer0_break_parameter.break_state = TIMER_BREAK_ENABLE;

hal_timer_break_config(&timer0_info,&timer0_break_parameter);
```

函数 hal_timer_ocpre_clear_source_config

函数hal_timer_ocpre_clear_source_config描述见下表:

Table 3-35. 函数 hal_timer_ocpre_clear_source_config

函数名称	hal_timer_ocpre_clear_source_config
函数原形	int32_t hal_timer_ocpre_clear_source_config(hal_timer_dev_struct *timer_dev, hal_timer_clear_source_struct *timer_clearsource)
功能描述	TIMER OxCPRE信号的清除源配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针, 指向hal_timer_dev_struct结构体, 结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
timer_clearsource	指针, 指向hal_timer_clear_source_struct结构体, 结构体成员参考 Table 3-25. 结构体hal_timer_clear_source_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL

例如:

```

/* configure TIMER OCPRE clear source */

hal_timer_dev_struct timer0_info;

hal_timer_clear_source_struct timer0_clear_source_parameter;

timer0_clear_source_parameter.clear_source = TIMER_OCPRE_CLEAR_SOURCE_ETIF;

timer0_clear_source_parameter.exttrigger_polarity = TIMER_EXT_TRI_POLARITY_RISING;

timer0_clear_source_parameter.exttrigger_prescaler =
TIMER_EXT_TRI_PRESCALER_OFF;

timer0_clear_source_parameter.exttrigger_filter = 0;

hal_timer_ocpre_clear_source_config(&timer0_info,&timer0_clear_source_parameter);

```

函数 hal_timer_ci0_input_select

函数hal_timer_ci0_input_select描述见下表：

Table 3-36. 函数 hal_timer_ci0_input_select

函数名称	hal_timer_ci0_input_select
函数原形	int32_t hal_timer_ci0_input_select(hal_timer_dev_struct *timer_dev, uint32_t ci0_select)
功能描述	TIMER CI0触发输入配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
ci0_select	CI0选择
TIMER_CI0_CH0IN	CI0连接到：CH0_IN
TIMER_CI0_XOR_CH012	CI0连接到：CH0_IN、CH1_IN和CH2_IN的异或信号
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL

例如：

```

/* configure TIMER ci0 trigger input */

hal_timer_dev_struct timer0_info;

hal_timer_ci0_input_select(&timer0_info,TIMER_CI0_XOR_CH012);

```

函数 hal_timer_single_pulse_mode_config

函数hal_timer_single_pulse_mode_config描述见下表：

Table 3-37. 函数 hal_timer_single_pulse_mode_config

函数名称	hal_timer_single_pulse_mode_config
函数原形	int32_t hal_timer_single_pulse_mode_config(hal_timer_dev_struct *timer_dev, uint32_t single_pulse)
功能描述	TIMER单脉冲模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
single_pulse	单脉冲使能状态
ENABLE	单脉冲模式使能
DISABLE	单脉冲模式禁能
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL

例如：

```
/* configure TIMER single pulse mode */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_single_pulse_mode_config(&timer0_info,ENABLE);
```

函数 hal_timer_clock_source_config

函数hal_timer_clock_source_config描述见下表：

Table 3-38. 函数 hal_timer_clock_source_config

函数名称	hal_timer_clock_source_config
函数原形	int32_t hal_timer_clock_source_config(hal_timer_dev_struct *timer_dev, hal_timer_clock_source_struct *timer_clocksource)
功能描述	TIMER时钟源配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct

输入参数{in}	
timer_clock	指针，指向hal_timer_clock_source_struct结构体，结构体成员参考 Table 3-26. 结构体hal timer clock source struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL

例如：

```
/* configure TIMER clock source */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_clock_source_struct timer0_clock_source_parameter;
```

```
timer0_clock_source_parameter.clock_source = TIMER_CLOCK_SOURCE_ETIMODE1;
```

```
timer0_clock_source_parameter.clock_polarity =  
TIMER_CLOCK_TRIGGER_ETI_POLARITY_RISING;
```

```
timer0_clock_source_parameter.clock_prescaler = TIMER_EXT_TRI_PRESCALER_OFF;
```

```
timer0_clock_source_parameter.clock_filter = 0;
```

```
hal_timer_clock_source_config(&timer0_info,&timer0_clock_source_parameter);
```

函数 hal_timer_slave_mode_config

函数hal_timer_slave_mode_config描述见下表：

Table 3-39. 函数 hal_timer_slave_mode_config

函数名称	hal_timer_slave_mode_config
函数原形	int32_t hal_timer_slave_mode_config(hal_timer_dev_struct *timer_dev, hal_timer_slave_mode_struct *timer_slavemode)
功能描述	TIMER从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal timer dev struct
输入参数{in}	
timer_slavemode	指针，指向hal_timer_slave_mode_struct结构体，结构体成员参考 Table 3-27. 结构体hal timer slave mode struct
输出参数{out}	
-	-
返回值	

int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL
---------	--

例如:

```
/* configure TIMER slave mode */
hal_timer_dev_struct timer0_info;
hal_timer_slave_mode_struct timer0_slave_mode_parameter;
timer0_slave_mode_parameter.slavemode = TIMER_SLAVE_MODE_RESTART_MODE;
timer0_slave_mode_parameter.trigger_selection = TIMER_TRIGGER_SOURCE_ETIFP;
timer0_slave_mode_parameter.trigger_polarity =
TIMER_CLOCK_TRIGGER_ETI_POLARITY_RISING;
timer0_slave_mode_parameter.trigger_prescaler = TIMER_EXT_TRI_PRESCALER_OFF;
timer0_slave_mode_parameter.trigger_filter = 0;
hal_timer_slave_mode_config(&timer0_info,&timer0_slave_mode_parameter);
```

函数 hal_timer_decoder_config

函数hal_timer_decoder_config描述见下表:

Table 3-40. 函数 hal_timer_decoder_config

函数名称	hal_timer_decoder_config
函数原形	int32_t hal_timer_decoder_config(hal_timer_dev_struct *timer_dev, hal_timer_decoder_struct *timer_decoder)
功能描述	TIMER译码器模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针, 指向hal_timer_dev_struct结构体, 结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
timer_decoder	指针, 指向hal_timer_decoder_struct结构体, 结构体成员参考 Table 3-28. 结构体hal_timer_decoder_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT

例如:

```
/* configure TIMER decoder mode */
```

```

hal_timer_dev_struct timer0_info;

hal_timer_decoder_struct timer0_decoder_parameter;

timer0_decoder_parameter.decoder_mode = TIMER_QUADRATURE_DECODER_MODE0;

timer0_decoder_parameter.ci0_polarity = TIMER_IC_POLARITY_RISING;

timer0_decoder_parameter.ci0_selection = TIMER_IC_SELECTION_DIRECTTI;

timer0_decoder_parameter.ci0_prescaler = TIMER_IC_PRESCALER_OFF;

timer0_decoder_parameter.ci0_filter = 0;

timer0_decoder_parameter.ci1_polarity = TIMER_IC_POLARITY_RISING;

timer0_decoder_parameter.ci1_selection = TIMER_IC_SELECTION_DIRECTTI;

timer0_decoder_parameter.ci1_prescaler = TIMER_IC_PRESCALER_OFF;

timer0_decoder_parameter.ci1_filter = 0;

hal_timer_decoder_config(&timer0_info,&timer0_decoder_parameter);

```

函数 hal_timer_hall_sensor_config

函数hal_timer_hall_sensor_config描述见下表：

Table 3-41. 函数 hal_timer_hall_sensor_config

函数名称	hal_timer_hall_sensor_config
函数原形	int32_t hal_timer_hall_sensor_config(hal_timer_dev_struct *timer_dev, hal_timer_hall_sensor_struct *timer_hallsensor)
功能描述	TIMER霍尔传感器模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
timer_hallsensor	指针，指向hal_timer_hall_sensor_struct结构体，结构体成员参考 Table 3-29. 结构体hal_timer_hall_sensor_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT

例如：

```
/* configure TIMER hall sensor mode */
```

```

hal_timer_dev_struct timer0_info;

hal_timer_hall_sensor_struct timer0_hall_sensor_parameter;

timer0_hall_sensor_parameter.cmt_delay = 50;

timer0_hall_sensor_parameter.ci0_polarity = TIMER_IC_POLARITY_RISING;

timer0_hall_sensor_parameter.ci0_selection = TIMER_IC_SELECTION_ITS;

timer0_hall_sensor_parameter.ci0_prescaler = TIMER_IC_PRESCALER_OFF;

timer0_hall_sensor_parameter.ci0_filter = 0;

hal_timer_hall_sensor_config(&timer0_info,&timer0_hall_sensor_parameter);

```

函数 hal_timer_struct_init

函数hal_timer_struct_init描述见下表：

Table 3-42. 函数 hal_timer_struct_init

函数名称	hal_timer_struct_init
函数原形	int32_t hal_timer_struct_init(hal_timer_struct_type_enum hal_struct_type, void *p_struct)
功能描述	初始化TIMER结构体为默认值
先决条件	-
被调用函数	-
输入参数{in}	
hal_struct_type	初始化的结构体类型，参考 Table 3-7. 枚举 hal_timer_struct_type_enum
输入参数{in}	
p_struct	指针，指向对应结构体实参
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如：

```

/* initialize the TIMER hal_timer_dev_struct structure with the default values */

hal_timer_dev_struct timer0_info;

hal_timer_struct_init(HAL_TIMER_DEV_STRUCT, &timer0_info);

```

函数 hal_timer_deinit

函数hal_timer_deinit描述见下表：

Table 3-43. 函数 hal_timer_deinit

函数名称	hal_timer_deinit
函数原形	int32_t hal_timer_deinit(hal_timer_dev_struct *timer_dev)
功能描述	复位TIMER和设备信息结构体
先决条件	-
被调用函数	hal_rcu_periph_reset_enable / hal_rcu_periph_reset_disable
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* deinitialize TIMER and device structure */
hal_timer_dev_struct timer0_info;
hal_timer_deinit(&timer0_info);
```

函数 hal_timer_counter_start

函数hal_timer_counter_start描述见下表：

Table 3-44. 函数 hal_timer_counter_start

函数名称	hal_timer_counter_start
函数原形	int32_t hal_timer_counter_start(hal_timer_dev_struct *timer_dev)
功能描述	TIMER计数器启动
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```
/* start TIMER counter */
hal_timer_dev_struct timer0_info;
hal_timer_counter_start(&timer0_info);
```

函数 hal_timer_counter_stop

函数hal_timer_counter_stop描述见下表：

Table 3-45. 函数 hal_timer_counter_stop

函数名称	hal_timer_counter_stop
函数原形	int32_t hal_timer_counter_stop(hal_timer_dev_struct *timer_dev)
功能描述	TIMER计数器停止
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK, HAL_ERR_NO_SUPPORT

例如：

```
/* stop TIMER counter */
hal_timer_dev_struct timer0_info;
hal_timer_counter_stop(&timer0_info);
```

函数 hal_timer_counter_start_interrupt

函数hal_timer_counter_start_interrupt描述见下表：

Table 3-46. 函数 hal_timer_counter_start_interrupt

函数名称	hal_timer_counter_start_interrupt
函数原形	int32_t hal_timer_counter_start_interrupt(hal_timer_dev_struct *timer_dev, hal_timer_irq_struct *p_irq)
功能描述	TIMER计数器启动，同时使能更新(update)中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
p_irq	指针，指向hal_timer_irq_struct结构体，结构体成员参考 Table 3-18. 结构体hal_timer_irq_struct
输出参数{out}	
-	-

返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如:

```
/* start TIMER counter and update interrupt */

void timer0_update_interrupt_handle_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_irq_struct timer0_irq_parameter;

uint8_t update_num = 0;

timer0_irq_parameter.update_handle = timer0_update_interrupt_handle_userdefine;

hal_timer_counter_start_interrupt(&timer0_info,&timer0_irq_parameter);

void timer0_update_interrupt_handle_userdefine(void *ptr)
{
    update_num++;
}
```

函数 hal_timer_counter_stop_interrupt

函数hal_timer_counter_stop_interrupt描述见下表:

Table 3-47. 函数 hal_timer_counter_stop_interrupt

函数名称	hal_timer_counter_stop_interrupt
函数原形	int32_t hal_timer_counter_stop_interrupt(hal_timer_dev_struct *timer_dev)
功能描述	TIMER计数器停止, 同时禁能更新(update)中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针, 指向hal_timer_dev_struct结构体, 结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK, HAL_ERR_NO_SUPPORT

例如:

```
/* stop TIMER counter and update interrupt */

hal_timer_dev_struct timer0_info;
```

```
hal_timer_counter_stop_interrupt(&timer0_info);
```

函数 hal_timer_counter_start_dma

函数hal_timer_counter_start_dma描述见下表：

Table 3-48. 函数 hal_timer_counter_start_dma

函数名称	hal_timer_counter_start_dma
函数原形	int32_t hal_timer_counter_start_dma(hal_timer_dev_struct *timer_dev, hal_timer_dma_handle_cb_struct *dmacb, uint32_t *mem_addr, uint16_t dma_length)
功能描述	TIMER计数器启动，同时使能更新(update)事件DMA请求
先决条件	-
被调用函数	hal_dma_start_interrupt
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
dmacb	指针，指向hal_timer_dma_handle_cb_struct结构体，结构体成员参考 Table 3-19. 结构体hal_timer_dma_handle_cb_struct
输入参数{in}	
mem_addr	TIMER DMA传输memory地址
输入参数{in}	
dma_length	TIMER DMA传输长度（0~65535）
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```
/* start TIMER counter and update DMA request */
void timer0_dmaerror(void *ptr);
void timer0_update_dmatc_userdefine(void *ptr);
hal_timer_dev_struct timer0_info;
hal_timer_dma_handle_cb_struct timer_dma_handle_cb;
uint32_t timer_cnt[3] = {100,200,300};
uint8_t timer0_dmaerror_num = 0;
uint8_t timer0_update_dmatc_num = 0;

timer_dma_handle_cb.          update_dma_full_transcom_handle          =          &
timer0_update_dmatc_userdefine;
```



```

timer_dma_handle_cb.error_handle = &timer0_dmaerror;

hal_timer_counter_start_dma(&timer0_info, &timer_dma_handle_cb, timer_cnt, 3);

void timer0_dmaerror(void *ptr)
{
    timer0_dmaerror_num++;
}

void timer0_update_dmatc_userdefine(void *ptr)
{
    timer0_update_dmatc_num++;
}

```

函数 hal_timer_counter_stop_dma

函数hal_timer_counter_stop_dma描述见下表：

Table 3-49. 函数 hal_timer_counter_stop_dma

函数名称	hal_timer_counter_stop_dma
函数原形	int32_t hal_timer_counter_stop_dma(hal_timer_dev_struct *timer_dev)
功能描述	TIMER计数器停止，同时禁能更新(update)事件DMA请求
先决条件	-
被调用函数	hal_dma_stop
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK, HAL_ERR_NO_SUPPORT

例如：

```

/* stop TIMER counter and update DMA request */

hal_timer_dev_struct timer0_info;

hal_timer_counter_stop_dma(&timer0_info);

```

函数 hal_timer_input_capture_start

函数hal_timer_input_capture_start描述见下表：

Table 3-50. 函数 hal_timer_input_capture_start

函数名称	hal_timer_input_capture_start
函数原形	int32_t hal_timer_input_capture_start(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER通道的输入捕获模式启动
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```
/* start TIMER channel input capture mode */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_input_capture_start(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_input_capture_stop

函数hal_timer_input_capture_stop描述见下表：

Table 3-51. 函数 hal_timer_input_capture_stop

函数名称	hal_timer_input_capture_stop
函数原形	int32_t hal_timer_input_capture_stop(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER通道的输入捕获模式停止
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0

<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
<i>TIMER_CH_3</i>	通道3
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS , HAL_ERR_NO_SUPPORT , HAL_ERR_LOCK

例如：

```
/* stop TIMER channel input capture mode */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_input_capture_stop(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_input_capture_start_interrupt

函数hal_timer_input_capture_start_interrupt描述见下表：

Table 3-52. 函数 hal_timer_input_capture_start_interrupt

函数名称	hal_timer_input_capture_start_interrupt
函数原形	int32_t hal_timer_input_capture_start_interrupt(hal_timer_dev_struct *timer_dev, uint16_t channel, hal_timer_irq_struct *p_irq)
功能描述	TIMER通道的输入捕获模式启动，同时使能比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal timer dev struct
输入参数{in}	
channel	指定通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
<i>TIMER_CH_3</i>	通道3
输入参数{in}	
p_irq	指针，指向hal_timer_irq_struct结构体，结构体成员参考 Table 3-18. 结构体hal timer irq struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如:

```

/* start TIMER channel input capture mode and channel interrupt */

void timer0_input_capture_interrupt_handle_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_irq_struct timer0_irq_parameter;

uint8_t input_capture_num = 0;

timer0_irq_parameter.channelx_capture_handle =
timer0_input_capture_interrupt_handle_userdefine;

hal_timer_input_capture_start_interrupt(&timer0_info,                TIMER_CH_0,
&timer0_irq_parameter);

void timer0_input_capture_interrupt_handle_userdefine(void *ptr)
{
    input_capture_num++;
}

```

函数 hal_timer_input_capture_stop_interrupt

函数hal_timer_input_capture_stop_interrupt描述见下表:

Table 3-53. 函数 hal_timer_input_capture_stop_interrupt

函数名称	hal_timer_input_capture_stop_interrupt
函数原形	int32_t hal_timer_input_capture_stop_interrupt(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER通道的输入捕获模式停止，同时禁能比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输出参数{out}	
-	-
返回值	

int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL, HAL_ERR_LOCK
---------	---

例如：

```
/* stop TIMER channel input capture mode and channel interrupt */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_input_capture_stop_interrupt(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_input_capture_start_dma

函数hal_timer_input_capture_start_dma描述见下表：

Table 3-54. 函数 hal_timer_input_capture_start_dma

函数名称	hal_timer_input_capture_start_dma
函数原形	int32_t hal_timer_input_capture_start_dma(hal_timer_dev_struct *timer_dev, uint16_t channel, hal_timer_dma_handle_cb_struct *dmacb, uint32_t *mem_addr, uint16_t dma_length)
功能描述	TIMER通道的输入捕获模式启动，同时使能比较/捕获DMA请求
先决条件	-
被调用函数	hal_dma_start_interrupt
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输入参数{in}	
dmacb	指针，指向hal_timer_dma_handle_cb_struct结构体，结构体成员参考 Table 3-19. 结构体hal_timer_dma_handle_cb_struct
输入参数{in}	
mem_addr	TIMER DMA传输memory地址
输入参数{in}	
dma_length	TIMER DMA传输长度（0~65535）
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```

/* start TIMER channel input capture mode and channel DMA request */

void timer0_dmaerror(void *ptr);

void timer0_input_capture_dmatc_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_dma_handle_cb_struct timer_dma_handle_cb;

uint32_t timer_ch0val[3] = {0};

uint8_t timer0_dmaerror_num = 0;

uint8_t timer0_input_capture_dmatc_num = 0;

timer_dma_handle_cb.channelx_capture_dma_full_transcom_handle = &
timer0_input_capture_dmatc_userdefine;

timer_dma_handle_cb.error_handle = &timer0_dmaerror;

hal_timer_input_capture_start_dma(&timer0_info, TIMER_CH_0 ,&timer_dma_handle_cb,
timer_ch0val, 3);

void timer0_dmaerror(void *ptr)
{
    timer0_dmaerror_num++;
}

void timer0_input_capture_dmatc_userdefine(void *ptr)
{
    timer0_input_capture_dmatc_num++;
}

```

函数 hal_timer_input_capture_stop_dma

函数hal_timer_input_capture_stop_dma描述见下表：

Table 3-55. 函数 hal_timer_input_capture_stop_dma

函数名称	hal_timer_input_capture_stop_dma
函数原形	int32_t hal_timer_input_capture_stop_dma(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER通道的输入捕获模式停止，同时禁能比较/捕获DMA请求
先决条件	-
被调用函数	hal_dma_stop
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结

	结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL, HAL_ERR_LOCK

例如:

```
/* stop TIMER channel input capture mode and channel DMA request */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_input_capture_stop_dma (&timer0_info, TIMER_CH_0);
```

函数 hal_timer_output_compare_start

函数hal_timer_output_compare_start描述见下表:

Table 3-56. 函数 hal_timer_output_compare_start

函数名称	hal_timer_output_compare_start
函数原形	int32_t hal_timer_output_compare_start(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER通道的输出比较模式启动
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针, 指向hal_timer_dev_struct结构体, 结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK,

例如：

```
/* start TIMER channel output compare mode */

hal_timer_dev_struct timer0_info;

hal_timer_output_compare_start(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_output_compare_stop

函数hal_timer_output_compare_stop描述见下表：

Table 3-57. 函数 hal_timer_output_compare_stop

函数名称	hal_timer_output_compare_stop
函数原形	int32_t hal_timer_output_compare_stop(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER通道的输出比较模式停止
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_LOCK,

例如：

```
/* stop TIMER channel output compare mode */

hal_timer_dev_struct timer0_info;

hal_timer_output_compare_stop(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_output_compare_start_interrupt

函数hal_timer_output_compare_start_interrupt描述见下表：

Table 3-58. 函数 hal_timer_output_compare_start_interrupt

函数名称	hal_timer_output_compare_start_interrupt
------	--

函数原形	int32_t hal_timer_output_compare_start_interrupt(hal_timer_dev_struct *timer_dev, uint16_t channel, hal_timer_irq_struct *p_irq)
功能描述	TIMER通道的输出比较模式启动，同时使能比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal timer dev struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输入参数{in}	
p_irq	指针，指向hal_timer_irq_struct结构体，结构体成员参考 Table 3-18. 结构体hal timer irq struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK,

例如：

```

/* start TIMER channel output compare mode and channel interrupt */
void timer0_output_compare_interrupt_handle_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_irq_struct timer0_irq_parameter;

uint8_t output_compare_num = 0;

timer0_irq_parameter.channelx_compare_handle =
timer0_output_compare_interrupt_handle_userdefine;

hal_timer_output_compare_start_interrupt(&timer0_info,                TIMER_CH_0,
&timer0_irq_parameter);

void timer0_output_compare_interrupt_handle_userdefine(void *ptr)
{
    output_compare_num++;
}

```

函数 hal_timer_output_compare_stop_interrupt

函数hal_timer_output_compare_stop_interrupt描述见下表:

Table 3-59. 函数 hal_timer_output_compare_stop_interrupt

函数名称	hal_timer_output_compare_stop_interrupt
函数原形	int32_t hal_timer_output_compare_stop_interrupt(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER通道的输出比较模式停止，同时禁能比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL, HAL_ERR_LOCK

例如:

```
/* stop TIMER channel output compare mode and channel interrupt */
hal_timer_dev_struct timer0_info;
hal_timer_output_compare_stop_interrupt(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_output_compare_start_dma

函数hal_timer_output_compare_start_dma描述见下表:

Table 3-60. 函数 hal_timer_output_compare_start_dma

函数名称	hal_timer_output_compare_start_dma
函数原形	int32_t hal_timer_output_compare_start_dma(hal_timer_dev_struct *timer_dev, uint16_t channel, hal_timer_dma_handle_cb_struct *dmacb, uint32_t *mem_addr, uint16_t dma_length)
功能描述	TIMER通道的输出比较模式启动，同时使能比较/捕获DMA请求
先决条件	-
被调用函数	hal_dma_start_interrupt

输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
<i>TIMER_CH_3</i>	通道3
输入参数{in}	
dmacb	指针，指向hal_timer_dma_handle_cb_struct结构体，结构体成员参考 Table 3-19. 结构体hal_timer_dma_handle_cb_struct
输入参数{in}	
mem_addr	TIMER DMA传输memory地址
输入参数{in}	
dma_length	TIMER DMA传输长度（0~65535）
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```

/* start TIMER channel output compare mode and channel DMA request */

void timer0_dmaerror(void *ptr);

void timer0_output_compare_dmatc_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_dma_handle_cb_struct timer_dma_handle_cb;

uint32_t timer_ch0val[3] = {0};

uint8_t timer0_dmaerror_num = 0;

uint8_t timer0_output_compare_dmatc_num = 0;

timer_dma_handle_cb.channelx_compare_dma_full_transcom_handle = &
timer0_output_compare_dmatc_userdefine;

timer_dma_handle_cb.error_handle = &timer0_dmaerror;

hal_timer_output_compare_start_dma(&timer0_info, TIMER_CH_0, &timer_dma_handle_cb,
timer_ch0val, 3);

void timer0_dmaerror(void *ptr)

```

```

{
    timer0_dmaerror_num++;
}

void timer0_output_compare_dmatc_userdefine(void *ptr)
{
    timer0_output_compare_dmatc_num++;
}

```

函数 hal_timer_output_compare_stop_dma

函数hal_timer_output_compare_stop_dma描述见下表：

Table 3-61. 函数 hal_timer_output_compare_stop_dma

函数名称	hal_timer_output_compare_stop_dma
函数原形	int32_t hal_timer_output_compare_stop_dma(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER通道的输出比较模式停止，同时禁能比较/捕获DMA请求
先决条件	-
被调用函数	hal_dma_stop
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
TIMER_CH_3	通道3
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_NO_SUPPORT, HAL_ERR_LOCK

例如：

```

/* stop TIMER channel output compare mode and channel DMA request */
hal_timer_dev_struct timer0_info;

hal_timer_output_compare_stop_dma(&timer0_info, TIMER_CH_0);

```

函数 hal_timer_output_compare_complementary_channel_start

函数hal_timer_output_compare_complementary_channel_start描述见下表：

Table 3-62. 函数 hal_timer_output_compare_complementary_channel_start

函数名称	hal_timer_output_compare_complementary_channel_start
函数原形	<pre>int32_t hal_timer_output_compare_complementary_channel_start(hal_timer_dev_s struct *timer_dev, uint16_t channel)</pre>
功能描述	TIMER互补通道的输出比较模式启动
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```
/* start TIMER complementary channel output compare mode */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_output_compare_complementary_channel_start(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_output_compare_complementary_channel_stop

函数hal_timer_output_compare_complementary_channel_stop描述见下表：

Table 3-63. 函数 hal_timer_output_compare_complementary_channel_stop

函数名称	hal_timer_output_compare_complementary_channel_stop
函数原形	<pre>int32_t hal_timer_output_compare_complementary_channel_stop(hal_timer_dev_s struct *timer_dev, uint16_t channel)</pre>
功能描述	TIMER互补通道的输出比较模式停止
先决条件	-
被调用函数	-
输入参数{in}	

timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_LOCK

例如：

```
/* stop TIMER complementary channel output compare mode */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_output_compare_complementary_channel_stop(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_output_compare_complementary_channel_start_interrupt

函数hal_timer_output_compare_complementary_channel_start_interrupt描述见下表：

Table 3-64. 函数 hal_timer_output_compare_complementary_channel_start_interrupt

函数名称	hal_timer_output_compare_complementary_channel_start_interrupt
函数原形	int32_t hal_timer_output_compare_complementary_channel_start_interrupt(hal_timer_dev_struct *timer_dev, uint16_t channel, hal_timer_irq_struct *p_irq)
功能描述	TIMER互补通道的输出比较模式启动，同时使能比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
输入参数{in}	
p_irq	指针，指向hal_timer_irq_struct结构体，结构体成员参考 Table 3-18. 结构体hal_timer_irq_struct
输出参数{out}	
-	-

返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```

/* start TIMER complementary channel output compare mode and channel interrupt */
void timer0_output_compare_interrupt_handle_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_irq_struct timer0_irq_parameter;

uint8_t output_compare_num = 0;

timer0_irq_parameter.channelx_compare_handle =
timer0_output_compare_interrupt_handle_userdefine;

hal_timer_output_compare_complementary_channel_start_interrupt(&timer0_info,
TIMER_CH_0, &timer0_irq_parameter);

void timer0_output_compare_interrupt_handle_userdefine(void *ptr)
{
    output_compare_num++;
}

```

函数 hal_timer_output_compare_complementary_channel_stop_interrupt

函数hal_timer_output_compare_complementary_channel_stop_interrupt描述见下表：

Table 3-65. 函数 hal_timer_output_compare_complementary_channel_stop_interrupt

函数名称	hal_timer_output_compare_complementary_channel_stop_interrupt
函数原形	int32_t hal_timer_output_compare_complementary_channel_stop_interrupt(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER互补通道的输出比较模式停止，同时禁能比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2

输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_NO_SUPPORT, HAL_ERR_LOCK

例如:

```
/* stop TIMER complementary channel output compare mode and channel interrupt */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_output_compare_complementary_channel_stop_interrupt(&timer0_info,  
TIMER_CH_0);
```

函数 hal_timer_output_compare_complementary_channel_start_dma

函数hal_timer_output_compare_complementary_channel_start_dma描述见下表:

Table 3-66. 函数 hal_timer_output_compare_complementary_channel_start_dma

函数名称	hal_timer_output_compare_complementary_channel_start_dma
函数原形	int32_t hal_timer_output_compare_complementary_channel_start_dma(hal_timer_dev_struct *timer_dev, uint16_t channel, hal_timer_dma_handle_cb_struct *dmacb, uint32_t *mem_addr, uint16_t dma_length)
功能描述	TIMER互补通道的输出比较模式启动, 同时使能比较/捕获DMA请求
先决条件	-
被调用函数	hal_dma_start_interrupt
输入参数{in}	
timer_dev	指针, 指向hal_timer_dev_struct结构体, 结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
输入参数{in}	
dmacb	指针, 指向hal_timer_dma_handle_cb_struct结构体, 结构体成员参考 Table 3-19. 结构体hal_timer_dma_handle_cb_struct
输入参数{in}	
mem_addr	TIMER DMA传输memory地址
输入参数{in}	
dma_length	TIMER DMA传输长度 (0~65535)
输出参数{out}	
-	-
返回值	

int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK
----------------	--

例如:

```
/* start TIMER complementary channel output compare mode and channel DMA request */
```

```
void timer0_dmaerror(void *ptr);
```

```
void timer0_output_compare_dmatc_userdefine(void *ptr);
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_dma_handle_cb_struct timer_dma_handle_cb;
```

```
uint32_t timer_ch0val[3] = {0};
```

```
uint8_t timer0_dmaerror_num = 0;
```

```
uint8_t timer0_output_compare_dmatc_num = 0;
```

```
timer_dma_handle_cb.channelx_compare_dma_full_transcom_handle = &
timer0_output_compare_dmatc_userdefine;
```

```
timer_dma_handle_cb.error_handle = &timer0_dmaerror;
```

```
hal_timer_output_compare_complementary_channel_start_dma(&timer0_info,
TIMER_CH_0 ,&timer_dma_handle_cb, timer_ch0val, 3);
```

```
void timer0_dmaerror(void *ptr)
```

```
{
    timer0_dmaerror_num++;
}
```

```
void timer0_output_compare_dmatc_userdefine(void *ptr)
```

```
{
    timer0_output_compare_dmatc_num++;
}
```

函数 hal_timer_output_compare_complementary_channel_stop_dma

函数hal_timer_output_compare_complementary_channel_stop_dma描述见下表:

Table 3-67. 函数 hal_timer_output_compare_complementary_channel_stop_dma

函数名称	hal_timer_output_compare_complementary_channel_stop_dma
函数原形	<pre>int32_t hal_timer_output_compare_complementary_channel_stop_dma(hal_timer_dev_struct *timer_dev, uint16_t channel)</pre>

功能描述	TIMER互补通道的输出比较模式停止，同时禁能比较/捕获DMA请求
先决条件	-
被调用函数	hal_dma_stop
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```
/* stop TIMER complementary channel output compare mode and channel DMA request */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_output_compare_complementary_channel_stop_dma(&timer0_info,  
TIMER_CH_0);
```

函数 hal_timer_single_pulse_mode_channel_config

函数hal_timer_single_pulse_mode_channel_config描述见下表：

Table 3-68. 函数 hal_timer_single_pulse_mode_channel_config

函数名称	hal_timer_single_pulse_mode_channel_config
函数原形	int32_t hal_timer_single_pulse_mode_channel_config(hal_timer_dev_struct *timer_dev, hal_timer_single_pulse_struct *timer_singlepulse, uint16_t channel_out, uint16_t channel_in)
功能描述	TIMER单脉冲模式通道配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
timer_singlepulse	指针，指向hal_timer_single_pulse_struct结构体，结构体成员参考 Table 3-30. 结构体hal_timer_single_pulse_struct
输入参数{in}	
channel_out	指定输出通道

<code>TIMER_CH_0</code>	通道0
<code>TIMER_CH_1</code>	通道1
输入参数{in}	
<code>channel_in</code>	指定输入通道
<code>TIMER_CH_0</code>	通道0
<code>TIMER_CH_1</code>	通道1
输出参数{out}	
-	-
返回值	
<code>int32_t</code>	错误码: <code>HAL_ERR_NONE</code> , <code>HAL_ERR_ADDRESS</code> , <code>HAL_ERR_VAL</code>

例如:

```

/* TIMER single pulse mode configure */
hal_timer_dev_struct timer0_info;
hal_timer_single_pulse_struct timer0_singlepulse_parameter;
timer0_singlepulse_parameter.sp_compare_mode = TIMER_OC_MODE_TOGGLE;
timer0_singlepulse_parameter.sp_oc_pulse_value = 500;
timer0_singlepulse_parameter.sp_oc_polarity = TIMER_OC_POLARITY_HIGH;
timer0_singlepulse_parameter.sp_oc_idlestate = TIMER_OC_IDLE_STATE_LOW;
timer0_singlepulse_parameter.sp_ocn_polarity = TIMER_OCN_POLARITY_HIGH;
timer0_singlepulse_parameter.sp_ocn_idlestate = TIMER_OCN_IDLE_STATE_LOW;
timer0_singlepulse_parameter.sp_oc_fastmode = TIMER_OC_FAST_DISABLE;
timer0_singlepulse_parameter.sp_oc_clearmode = TIMER_OC_CLEAR_DISABLE;
timer0_singlepulse_parameter.sp_ic_polarity = TIMER_IC_POLARITY_RISING;
timer0_singlepulse_parameter.sp_ic_selection = TIMER_IC_SELECTION_DIRECTTI;
timer0_singlepulse_parameter.sp_ic_filter= 15;

hal_timer_single_pulse_mode_channel_config(&timer0_info,
&timer0_singlepulse_parameter, TIMER_CH_0, TIMER_CH_1);

```

函数 `hal_timer_single_pulse_start`

函数`hal_timer_single_pulse_start`描述见下表:

Table 3-69. 函数 `hal_timer_single_pulse_start`

函数名称	<code>hal_timer_single_pulse_start</code>
函数原形	<code>int32_t hal_timer_single_pulse_start(hal_timer_dev_struct *timer_dev)</code>
功能描述	TIMER通道的单脉冲模式启动

先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```
/* start TIMER single pulse mode */
hal_timer_dev_struct timer0_info;
hal_timer_single_pulse_start (&timer0_info);
```

函数 hal_timer_single_pulse_stop

函数hal_timer_single_pulse_stop描述见下表：

Table 3-70. 函数 hal_timer_single_pulse_stop

函数名称	hal_timer_single_pulse_stop
函数原形	int32_t hal_timer_single_pulse_stop(hal_timer_dev_struct *timer_dev)
功能描述	TIMER通道的单脉冲模式禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_LOCK

例如：

```
/* stop TIMER single pulse mode */
hal_timer_dev_struct timer0_info;
hal_timer_single_pulse_stop(&timer0_info);
```

函数 hal_timer_single_pulse_start_interrupt

函数hal_timer_single_pulse_start_interrupt描述见下表：

Table 3-71. 函数 hal_timer_single_pulse_start_interrupt

函数名称	hal_timer_single_pulse_start_interrupt
函数原形	int32_t hal_timer_single_pulse_start_interrupt(hal_timer_dev_struct *timer_dev, hal_timer_irq_struct *p_irq)
功能描述	TIMER通道的单脉冲模式启动，同时使能比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
p_irq	指针，指向hal_timer_irq_struct结构体，结构体成员参考 Table 3-18. 结构体hal_timer_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```

/* start TIMER single pulse mode and channel interrupt */

void timer0_input_capture_interrupt_handle_userdefine(void *ptr);

void timer0_output_compare_interrupt_handle_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_irq_struct timer0_irq_parameter;

uint8_t input_capture_num = 0;

uint8_t output_compare_num = 0;

timer0_irq_parameter.channelx_capture_handle =
timer0_input_capture_interrupt_handle_userdefine;

timer0_irq_parameter.channelx_compare_handle =
timer0_output_compare_interrupt_handle_userdefine;

hal_timer_single_pulse_start_interrupt(&timer0_info, &timer0_irq_parameter);

void timer0_input_capture_interrupt_handle_userdefine(void *ptr)
{
    input_capture_num++;
}

void timer0_output_compare_interrupt_handle_userdefine(void *ptr)

```

```
{
    output_compare_num++;
}
```

函数 hal_timer_single_pulse_stop_interrupt

函数hal_timer_single_pulse_stop_interrupt描述见下表:

Table 3-72. 函数 hal_timer_single_pulse_stop_interrupt

函数名称	hal_timer_single_pulse_stop_interrupt
函数原形	int32_t hal_timer_single_pulse_stop_interrupt(hal_timer_dev_struct *timer_dev)
功能描述	TIMER通道的单脉冲模式停止，同时禁能比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS , HAL_ERR_NO_SUPPORT, HAL_ERR_LOCK

例如:

```
/* stop TIMER single pulse mode and channel interrupt */
hal_timer_dev_struct timer0_info;
hal_timer_single_pulse_stop_interrupt (&timer0_info);
```

函数 hal_timer_single_pulse_complementary_channel_start

函数hal_timer_single_pulse_complementary_channel_start描述见下表:

Table 3-73. 函数 hal_timer_single_pulse_complementary_channel_start

函数名称	hal_timer_single_pulse_complementary_channel_start
函数原形	int32_t int32_t hal_timer_single_pulse_complementary_channel_start(hal_timer_dev_struct *timer_dev, uint16_t channel_out)
功能描述	TIMER互补通道的单脉冲模式启动
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct

	结构体hal_timer_dev_struct
输入参数{in}	
channel_out	指定输出通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如:

```
/* start TIMER complementary channel single pulse mode */
hal_timer_dev_struct timer0_info;
hal_timer_single_pulse_complementary_channel_start(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_single_pulse_complementary_channel_stop

函数hal_timer_single_pulse_complementary_channel_stop描述见下表:

Table 3-74. 函数 hal_timer_single_pulse_complementary_channel_stop

函数名称	hal_timer_single_pulse_complementary_channel_stop
函数原形	int32_t hal_timer_single_pulse_complementary_channel_stop(hal_timer_dev_struct *timer_dev, uint16_t channel_out)
功能描述	TIMER互补通道的单脉冲模式停止
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针, 指向hal_timer_dev_struct结构体, 结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel_out	指定输出通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL, HAL_ERR_LOCK

例如:

```
/* stop TIMER complementary channel single pulse mode */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_single_pulse_complementary_channel_stop(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_single_pulse_complementary_channel_start_interrupt

函数hal_timer_single_pulse_complementary_channel_start_interrupt描述见下表：

Table 3-75. 函数 hal_timer_single_pulse_complementary_channel_start_interrupt

函数名称	hal_timer_single_pulse_complementary_channel_start_interrupt
函数原形	<pre>int32_t hal_timer_single_pulse_complementary_channel_start_interrupt(hal_timer_dev_struct *timer_dev, uint16_t channel_out, hal_timer_irq_struct *p_irq)</pre>
功能描述	TIMER互补通道的单脉冲模式启动，同时使能比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel_out	指定输出通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
输入参数{in}	
p_irq	指针，指向hal_timer_irq_struct结构体，结构体成员参考 Table 3-18. 结构体hal_timer_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```
/* start TIMER complementary channel single pulse mode and channel interrupt */
void timer0_input_capture_interrupt_handle_userdefine(void *ptr);
void timer0_output_compare_interrupt_handle_userdefine(void *ptr);
hal_timer_dev_struct timer0_info;
hal_timer_irq_struct timer0_irq_parameter;
uint8_t input_capture_num = 0;
uint8_t output_compare_num = 0;
timer0_irq_parameter.channelx_capture_handle =
```



```

timer0_input_capture_interrupt_handle_userdefine;

timer0_irq_parameter.channelx_compare_handle =
timer0_output_compare_interrupt_handle_userdefine;

hal_timer_single_pulse_complementary_channel_start_interrupt(&timer0_info,
TIMER_CH_0, &timer0_irq_parameter);

void timer0_input_capture_interrupt_handle_userdefine(void *ptr)
{
    input_capture_num++;
}

void timer0_output_compare_interrupt_handle_userdefine(void *ptr)
{
    output_compare_num++;
}

```

函数 hal_timer_single_pulse_complementary_channel_stop_interrupt

函数hal_timer_single_pulse_complementary_channel_stop_interrupt描述见下表:

Table 3-76. 函数 hal_timer_single_pulse_complementary_channel_stop_interrupt

函数名称	hal_timer_single_pulse_complementary_channel_stop_interrupt
函数原形	int32_t hal_timer_single_pulse_complementary_channel_stop_interrupt(hal_timer_dev_struct *timer_dev, uint16_t channel_out)
功能描述	TIMER互补通道的单脉冲模式停止，同时禁能比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel_out	指定输出通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL, HAL_ERR_LOCK

例如:

```
/* stop TIMER complementary channel single pulse mode and channel interrupt */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_single_pulse_complementary_channel_stop_interrupt(&timer0_info,  
TIMER_CH_0);
```

函数 hal_timer_slave_mode_interrupt_config

函数hal_timer_slave_mode_interrupt_config描述见下表：

Table 3-77. 函数 hal_timer_slave_mode_interrupt_config

函数名称	hal_timer_slave_mode_interrupt_config
函数原形	int32_t hal_timer_slave_mode_interrupt_config(hal_timer_dev_struct *timer_dev, hal_timer_slave_mode_struct *timer_slavemode, hal_timer_irq_struct *p_irq)
功能描述	TIMER从模式配置，同时使能触发(TRG)中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
timer_slavemode	指针，指向hal_timer_slave_mode_struct结构体，结构体成员参考 Table 3-27. 结构体hal_timer_slave_mode_struct
输入参数{in}	
p_irq	指针，指向hal_timer_irq_struct结构体，结构体成员参考 Table 3-18. 结构体hal_timer_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL

例如：

```
/* configure TIMER slave mode and interrupt */
```

```
void timer0_trigger_interrupt_handle_userdefine(void *ptr);
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_irq_struct timer0_irq_parameter;
```

```
hal_timer_slave_mode_struct timer0_slave_mode_parameter;
```

```
uint8_t trigger_num = 0;
```

```
timer0_irq_parameter.trigger_handle = timer0_trigger_interrupt_handle_userdefine;
```

```

timer0_slave_mode_parameter.slavemode = TIMER_SLAVE_MODE_RESTART_MODE;

timer0_slave_mode_parameter.trigger_selection = TIMER_TRIGGER_SOURCE_ETIFP;

timer0_slave_mode_parameter.trigger_polarity
TIMER_CLOCK_TRIGGER_ETI_POLARITY_RISING;

timer0_slave_mode_parameter.trigger_prescaler = TIMER_EXT_TRI_PRESCALER_OFF;

timer0_slave_mode_parameter.trigger_filter = 0;

hal_timer_slave_mode_interrupt_config(&timer0_info,&timer0_slave_mode_parameter,
&timer0_irq_parameter);

void timer0_trigger_interrupt_handle_userdefine(void *ptr)

{

    trigger_num++;

}

```

函数 hal_timer_decoder_start

函数hal_timer_decoder_start描述见下表：

Table 3-78. 函数 hal_timer_decoder_start

函数名称	hal_timer_decoder_start
函数原形	int32_t hal_timer_decoder_start(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER译码器模式启动
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_0_1	通道0和通道1
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```
/* start TIMER decoder mode */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_decoder_start(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_decoder_stop

函数hal_timer_decoder_stop描述见下表：

Table 3-79. 函数 hal_timer_decoder_stop

函数名称	hal_timer_decoder_stop
函数原形	int32_t hal_timer_decoder_stop(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER译码器模式停止
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_0_1	通道0和通道1
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```
/* stop TIMER decoder mode */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_decoder_stop(&timer0_info, TIMER_CH_0);
```

函数 hal_timer_decoder_start_interrupt

函数hal_timer_decoder_start_interrupt描述见下表：

Table 3-80. 函数 hal_timer_decoder_start_interrupt

函数名称	hal_timer_decoder_start_interrupt
函数原形	int32_t hal_timer_decoder_start_interrupt(hal_timer_dev_struct *timer_dev, uint16_t channel, hal_timer_irq_struct *p_irq)
功能描述	TIMER译码器模式启动，同时使能通道比较/捕获中断
先决条件	-

被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_0_1	通道0和通道1
输入参数{in}	
p_irq	指针，指向hal_timer_irq_struct结构体，结构体成员参考 Table 3-18. 结构体hal_timer_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```

/* start TIMER decoder mode and channel interrupt */

void timer0_input_capture_interrupt_handle_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_irq_struct timer0_irq_parameter;

uint8_t input_capture_num = 0;

timer0_irq_parameter.channelx_capture_handle =
timer0_input_capture_interrupt_handle_userdefine;

hal_timer_decoder_start_interrupt(&timer0_info, TIMER_CH_0, &timer0_irq_parameter);

void timer0_input_capture_interrupt_handle_userdefine(void *ptr)
{
    input_capture_num++;
}

```

函数 hal_timer_decoder_stop_interrupt

函数hal_timer_decoder_stop_interrupt描述见下表：

Table 3-81. 函数 hal_timer_decoder_stop_interrupt

函数名称	hal_timer_decoder_stop_interrupt
函数原形	int32_t hal_timer_decoder_stop_interrupt(hal_timer_dev_struct *timer_dev,

	uint16_t channel)
功能描述	TIMER译码器模式停止，同时禁能通道比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_0_1	通道0和通道1
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```
/* stop TIMER decoder mode and channel interrupt */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_decoder_stop_interrupt (&timer0_info, TIMER_CH_0);
```

函数 hal_timer_decoder_start_dma

函数hal_timer_decoder_start_dma描述见下表：

Table 3-82. 函数 hal_timer_decoder_start_dma

函数名称	hal_timer_decoder_start_dma
函数原形	int32_t hal_timer_decoder_start_dma(hal_timer_dev_struct *timer_dev, uint16_t channel, hal_timer_dma_handle_cb_struct *dmacb, hal_timer_decoder_dma_config_struct *decoder_dma)
功能描述	TIMER译码器模式启动，同时使能通道比较/捕获DMA请求
先决条件	-
被调用函数	hal_dma_start_interrupt
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_0_1	通道0和通道1

输入参数{in}	
dmacb	指针，指向hal_timer_dma_handle_cb_struct结构体，结构体成员参考 Table 3-19. 结构体hal timer dma handle cb struct
输入参数{in}	
decoder_dma	指针，指向hal_timer_decoder_dma_config_struct结构体，结构体成员参考 Table 3-16. 结构体hal timer decoder dma config struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```

/* start TIMER decoder mode and channel DMA request */
void timer0_dmaerror(void *ptr);
void timer0_input_capture_dmatc_userdefine(void *ptr);
hal_timer_dev_struct timer0_info;
hal_timer_dma_handle_cb_struct timer_dma_handle_cb;
hal_timer_decoder_dma_config_struct timer_decoder_dma_config;
uint32_t timer_ch0val[3] = {0};
uint32_t timer_ch1val[3] = {0};
uint8_t timer0_dmaerror_num = 0;
uint8_t timer0_input_capture_dmatc_num = 0;
timer_decoder_dma_config.mem_addr0 = timer_ch0val;
timer_decoder_dma_config.mem_addr1 = timer_ch1val;
timer_decoder_dma_config.length = 3;
timer_dma_handle_cb.channelx_capture_dma_full_transcom_handle = &
timer0_input_capture_dmatc_userdefine;
timer_dma_handle_cb.error_handle = &timer0_dmaerror;
hal_timer_decoder_start_dma(&timer0_info, TIMER_CH_0_1, &timer_dma_handle_cb,
&timer_decoder_dma_config);
void timer0_dmaerror(void *ptr)
{
    timer0_dmaerror_num++;

```

```

}

void timer0_input_capture_dmatc_userdefine(void *ptr)
{
    timer0_input_capture_dmatc_num++;
}

```

函数 hal_timer_decoder_stop_dma

函数hal_timer_decoder_stop_dma描述见下表：

Table 3-83. 函数 hal_timer_decoder_stop_dma

函数名称	hal_timer_decoder_stop_dma
函数原形	int32_t hal_timer_decoder_stop_dma(hal_timer_dev_struct *timer_dev, uint16_t channel)
功能描述	TIMER译码器模式停止，同时禁能通道比较/捕获DMA请求
先决条件	-
被调用函数	hal_dma_stop
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
channel	指定通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_0_1	通道0和通道1
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```

/* stop TIMER decoder mode and channel DMA request */
hal_timer_dev_struct timer0_info;
hal_timer_decoder_stop_dma(&timer0_info, TIMER_CH_0_1);

```

函数 hal_timer_hall_sensor_start

函数hal_timer_hall_sensor_start描述见下表：

Table 3-84. 函数 hal_timer_hall_sensor_start

函数名称	hal_timer_hall_sensor_start
------	-----------------------------

函数原形	int32_t hal_timer_hall_sensor_start(hal_timer_dev_struct *timer_dev)
功能描述	TIMER霍尔传感器模式启动
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```
/* start TIMER hall sensor mode */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_hall_sensor_start(&timer0_info);
```

函数 hal_timer_hall_sensor_stop

函数hal_timer_hall_sensor_stop描述见下表：

Table 3-85. 函数 hal_timer_hall_sensor_stop

函数名称	hal_timer_hall_sensor_stop
函数原形	int32_t hal_timer_hall_sensor_stop(hal_timer_dev_struct *timer_dev)
功能描述	TIMER霍尔传感器模式停止
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_LOCK

例如：

```
/* stop TIMER hall sensor mode */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_hall_sensor_stop (&timer0_info);
```

函数 hal_timer_hall_sensor_start_interrupt

函数hal_timer_hall_sensor_start_interrupt描述见下表：

Table 3-86. 函数 hal_timer_hall_sensor_start_interrupt

函数名称	hal_timer_hall_sensor_start_interrupt
函数原形	int32_t hal_timer_hall_sensor_start_interrupt(hal_timer_dev_struct *timer_dev, hal_timer_irq_struct *p_irq)
功能描述	TIMER霍尔传感器模式启动，同时使能通道比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
p_irq	指针，指向hal_timer_irq_struct结构体，结构体成员参考 Table 3-18. 结构体hal_timer_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如：

```

/* start TIMER hall sensor mode and channel interrupt */

void timer0_input_capture_interrupt_handle_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_irq_struct timer0_irq_parameter;

uint8_t input_capture_num = 0;

timer0_irq_parameter.channelx_capture_handle =
timer0_input_capture_interrupt_handle_userdefine;

hal_timer_hall_sensor_start_interrupt(&timer0_info, &timer0_irq_parameter);

void timer0_input_capture_interrupt_handle_userdefine(void *ptr)
{
    input_capture_num++;
}

```

函数 hal_timer_hall_sensor_stop_interrupt

函数hal_timer_hall_sensor_stop_interrupt描述见下表：

Table 3-87. 函数 hal_timer_hall_sensor_stop_interrupt

函数名称	hal_timer_hall_sensor_stop_interrupt
函数原形	int32_t hal_timer_hall_sensor_stop_interrupt(hal_timer_dev_struct *timer_dev)
功能描述	TIMER霍尔传感器模式停止，同时禁能通道比较/捕获中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_LOCK

例如：

```
/* stop TIMER hall sensor mode and channel interrupt */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_hall_sensor_stop_interrupt(&timer0_info);
```

函数 hal_timer_hall_sensor_start_dma

函数hal_timer_hall_sensor_start_dma描述见下表：

Table 3-88. 函数 hal_timer_hall_sensor_start_dma

函数名称	hal_timer_hall_sensor_start_dma
函数原形	int32_t hal_timer_hall_sensor_start_dma(hal_timer_dev_struct *timer_dev, hal_timer_dma_handle_cb_struct *dmacb, uint32_t *mem_addr, uint16_t dma_length)
功能描述	TIMER霍尔传感器模式启动，同时使能通道比较/捕获DMA请求
先决条件	-
被调用函数	hal_dma_start_interrupt
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
dmacb	指针，指向hal_timer_dma_handle_cb_struct结构体，结构体成员参考 Table 3-19. 结构体hal_timer_dma_handle_cb_struct
输入参数{in}	
mem_addr	TIMER DMA传输memory地址
输入参数{in}	
dma_length	TIMER DMA传输长度（0-65535）

输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_LOCK

例如:

```
/* start TIMER hall sensor mode and channel DMA request */
```

```
void timer0_dmaerror(void *ptr);
```

```
void timer0_input_capture_dmatc_userdefine(void *ptr);
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_dma_handle_cb_struct timer_dma_handle_cb;
```

```
uint32_t timer_ch0val[3] = {0};
```

```
uint8_t timer0_dmaerror_num = 0;
```

```
uint8_t timer0_input_capture_dmatc_num = 0;
```

```
timer_dma_handle_cb.channelx_capture_dma_full_transcom_handle =
&timer0_input_capture_dmatc_userdefine;
```

```
timer_dma_handle_cb.error_handle = &timer0_dmaerror;
```

```
hal_timer_hall_sensor_start_dma(&timer0_info, &timer_dma_handle_cb, timer_ch0val, 3);
```

```
void timer0_dmaerror(void *ptr)
```

```
{
    timer0_dmaerror_num++;
}
```

```
void timer0_input_capture_dmatc_userdefine(void *ptr)
```

```
{
    timer0_input_capture_dmatc_num++;
}
```

函数 hal_timer_hall_sensor_stop_dma

函数hal_timer_hall_sensor_stop_dma描述见下表:

Table 3-89. 函数 hal_timer_hall_sensor_stop_dma

函数名称	hal_timer_hall_sensor_stop_dma
函数原形	int32_t hal_timer_hall_sensor_stop_dma(hal_timer_dev_struct *timer_dev)
功能描述	TIMER霍尔传感器模式停止, 同时禁能通道比较/捕获DMA请求

先决条件	-
被调用函数	hal_dma_stop
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal timer dev struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_NO_SUPPORT, HAL_ERR_LOCK

例如：

```
/* stop TIMER hall sensor mode and channel DMA request */
hal_timer_dev_struct timer0_info;
hal_timer_hall_sensor_stop_dma(&timer0_info);
```

函数 hal_timer_dma_transfer_write_start

函数hal_timer_dma_transfer_write_start描述见下表：

Table 3-90. 函数 hal_timer_dma_transfer_write_start

函数名称	hal_timer_dma_transfer_write_start
函数原形	int32_t hal_timer_dma_transfer_write_start(hal_timer_dev_struct *timer_dev, uint32_t dmareq, hal_timer_dma_transfer_config_struct *dmatcfg, hal_timer_dma_handle_cb_struct *dmacb)
功能描述	DMA写模式启动，memory值写入TIMER若干连续寄存器
先决条件	-
被调用函数	hal_dma_start_interrupt
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal timer dev struct
输入参数{in}	
dmareq	DMA请求选择
TIMER_DMA_UPD	更新DMA请求
TIMER_DMA_CH0D	通道0比较/捕获DMA请求
TIMER_DMA_CH1D	通道1比较/捕获DMA请求
TIMER_DMA_CH2D	通道2比较/捕获DMA请求
TIMER_DMA_CH3D	通道3比较/捕获DMA请求
TIMER_DMA_CMTD	换相DMA请求
TIMER_DMA_TRGD	触发DMA请求
输入参数{in}	
dmatcfg	指针，指向hal_timer_dma_transfer_config_struct结构体，结构体成员参考 Table 3-17. 结构体hal timer dma transfer config struct

输入参数{in}	
dmacb	指针，指向hal_timer_dma_handle_cb_struct结构体，结构体成员参考 Table 3-19. 结构体hal timer dma handle cb struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```

/* start TIMER DMA transfer mode for writing data to TIMER */
void timer0_dmaerror(void *ptr);
void timer0_update_dmatc_userdefine(void *ptr);
hal_timer_dev_struct timer0_info;
hal_timer_dma_handle_cb_struct timer_dma_handle_cb;
hal_timer_dma_transfer_config_struct timer_dma_transfer_config;
uint32_t timer_chval[3] = {0};
uint8_t timer0_dmaerror_num = 0;
uint8_t timer0_update_dmatc_num = 0;

timer_dma_transfer_config.start_addr = TIMER_DMA_START_ADDRESS_CH0CV;
timer_dma_transfer_config.mem_addr = timer_chval;
timer_dma_transfer_config.length= TIMER_DMACFG_DMATC_3TRANSFER;

timer_dma_handle_cb.update_dma_full_transcom_handle = &timer0_update_dmatc_userdefine;

timer_dma_handle_cb.error_handle = &timer0_dmaerror;

hal_timer_dma_transfer_write_start(&timer0_info, TIMER_DMA_UPD,
&timer_dma_transfer_config, &timer_dma_handle_cb);

void timer0_dmaerror(void *ptr)
{
    timer0_dmaerror_num++;
}

void timer0_update_dmatc_userdefine(void *ptr)
{

```

```

timer0_update_dmatc_num++;

}

```

函数 hal_timer_dma_transfer_write_stop

函数hal_timer_dma_transfer_write_stop描述见下表:

Table 3-91. 函数 hal_timer_dma_transfer_write_stop

函数名称	hal_timer_dma_transfer_write_stop
函数原形	int32_t hal_timer_dma_transfer_write_stop(hal_timer_dev_struct *timer_dev, uint32_t dmareq)
功能描述	DMA写模式停止
先决条件	-
被调用函数	hal_dma_stop
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
dmareq	DMA请求选择
TIMER_DMA_UPD	更新DMA请求
TIMER_DMA_CH0D	通道0比较/捕获DMA请求
TIMER_DMA_CH1D	通道1比较/捕获DMA请求
TIMER_DMA_CH2D	通道2比较/捕获DMA请求
TIMER_DMA_CH3D	通道3比较/捕获DMA请求
TIMER_DMA_CMTD	换相DMA请求
TIMER_DMA_TRGD	触发DMA请求
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如:

```

/* stop TIMER DMA transfer mode for writing data to TIMER */
hal_timer_dev_struct timer0_info;

hal_timer_dma_transfer_write_stop(&timer0_info, TIMER_DMA_UPD);

```

函数 hal_timer_dma_transfer_read_start

函数hal_timer_dma_transfer_read_start描述见下表:

Table 3-92. 函数 hal_timer_dma_transfer_read_start

函数名称	hal_timer_dma_transfer_read_start
------	-----------------------------------

函数原形	int32_t hal_timer_dma_transfer_read_start(hal_timer_dev_struct *timer_dev, uint32_t dmareq, hal_timer_dma_transfer_config_struct *dmatcfg, hal_timer_dma_handle_cb_struct *dmacb)
功能描述	DMA读模式启动，读取TIMER若干连续寄存器值传输到memory
先决条件	-
被调用函数	hal_dma_start_interrupt
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
dmareq	DMA请求选择
TIMER_DMA_UPD	更新DMA请求
TIMER_DMA_CH0D	通道0比较/捕获DMA请求
TIMER_DMA_CH1D	通道1比较/捕获DMA请求
TIMER_DMA_CH2D	通道2比较/捕获DMA请求
TIMER_DMA_CH3D	通道3比较/捕获DMA请求
TIMER_DMA_CMTD	换相DMA请求
TIMER_DMA_TRGD	触发DMA请求
输入参数{in}	
dmatcfg	指针，指向hal_timer_dma_transfer_config_struct结构体，结构体成员参考 Table 3-17. 结构体hal_timer_dma_transfer_config_struct
输入参数{in}	
dmacb	指针，指向hal_timer_dma_handle_cb_struct结构体，结构体成员参考 Table 3-19. 结构体hal_timer_dma_handle_cb_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL, HAL_ERR_LOCK

例如：

```

/* start TIMER DMA transfer mode for read data from TIMER */
void timer0_dmaerror(void *ptr);
void timer0_input_capture_dmatc_userdefine(void *ptr);
hal_timer_dev_struct timer0_info;
hal_timer_dma_handle_cb_struct timer_dma_handle_cb;
hal_timer_dma_transfer_config_struct timer_dma_transfer_config;
uint32_t timer_chval[3] = {0};
uint8_t timer0_dmaerror_num = 0;

```



```
uint8_t timer0_input_capture_dmatc_num = 0;

timer_dma_transfer_config.start_addr = TIMER_DMA_START_ADDRESS_CH0CV;

timer_dma_transfer_config.mem_addr = timer_chval;

timer_dma_transfer_config.length= TIMER_DMACFG_DMATC_3TRANSFER;

timer_dma_handle_cb.channelx_capture_dma_full_transcom_handle      =      &
timer0_input_capture_dmatc_userdefine;

timer_dma_handle_cb.error_handle = &timer0_dmaerror;

hal_timer_dma_transfer_read_start(&timer0_info,                      TIMER_DMA_CH0D,
&timer_dma_transfer_config, &timer_dma_handle_cb);

void timer0_dmaerror(void *ptr)
{
    timer0_dmaerror_num++;
}

void timer0_input_capture_dmatc_userdefine(void *ptr)
{
    timer0_input_capture_dmatc_num++;
}
```

函数 hal_timer_dma_transfer_read_stop

函数hal_timer_dma_transfer_read_stop描述见下表：

Table 3-93. 函数 hal_timer_dma_transfer_read_stop

函数名称	hal_timer_dma_transfer_read_stop
函数原形	int32_t hal_timer_dma_transfer_read_stop(hal_timer_dev_struct *timer_dev, uint32_t dmareq)
功能描述	DMA读模式停止
先决条件	-
被调用函数	hal_dma_stop
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
dmareq	DMA请求选择
TIMER_DMA_UPD	更新DMA请求
TIMER_DMA_CH0D	通道0比较/捕获DMA请求
TIMER_DMA_CH1D	通道1比较/捕获DMA请求

<code>TIMER_DMA_CH2D</code>	通道2比较/捕获DMA请求
<code>TIMER_DMA_CH3D</code>	通道3比较/捕获DMA请求
<code>TIMER_DMA_CMTD</code>	换相DMA请求
<code>TIMER_DMA_TRGD</code>	触发DMA请求
输出参数{out}	
-	-
返回值	
<code>int32_t</code>	错误码: <code>HAL_ERR_NONE</code> , <code>HAL_ERR_ADDRESS</code> , <code>HAL_ERR_VAL</code> , <code>HAL_ERR_LOCK</code>

例如:

```
/* stop TIMER DMA transfer mode for read data from TIMER */
hal_timer_dev_struct timer0_info;
hal_timer_dma_transfer_read_stop(&timer0_info, TIMER_DMA_CH0D);
```

函数 `hal_timer_commutation_event_config`

函数`hal_timer_commutation_event_config`描述见下表:

Table 3-94. 函数 `hal_timer_commutation_event_config`

函数名称	<code>hal_timer_commutation_event_config</code>
函数原形	<code>int32_t hal_timer_commutation_event_config(hal_timer_dev_struct *timer_dev, uint32_t trigger_source, uint32_t com_source)</code>
功能描述	TIMER换相事件配置
先决条件	-
被调用函数	-
输入参数{in}	
<code>timer_dev</code>	指针, 指向 <code>hal_timer_dev_struct</code> 结构体, 结构体成员参考 Table 3-20. 结构体<code>hal_timer_dev_struct</code>
输入参数{in}	
<code>trigger_source</code>	触发源选择
<code>TIMER_TRIGGER_SOURCE_ITI0</code>	输入触发源为: ITI0
<code>TIMER_TRIGGER_SOURCE_ITI1</code>	输入触发源为: ITI1
<code>TIMER_TRIGGER_SOURCE_ITI2</code>	输入触发源为: ITI2
<code>TIMER_TRIGGER_SOURCE_ITI3</code>	输入触发源为: ITI3
<code>TIMER_TRIGGER_SOURCE_DISABLE</code>	无输入触发源
输入参数{in}	
<code>com_source</code>	CMT事件源选择

<i>TIMER_UPDATECTL_CCU</i>	CMTG位被置1时更新影子寄存器（CHxEN, CHxNEN和CHxCOMCTL位）
<i>TIMER_UPDATECTL_CCUTRI</i>	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新（CHxEN, CHxNEN和CHxCOMCTL位）
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* configure TIMER commutation event */
```

```
hal_timer_dev_struct timer0_info;
```

```
hal_timer_commutation_event_config(&timer0_info,  
TIMER_TRIGGER_SOURCE_DISABLE, TIMER_UPDATECTL_CCU);
```

函数 hal_timer_commutation_event_interrupt_config

函数hal_timer_commutation_event_interrupt_config描述见下表：

Table 3-95. 函数 hal_timer_commutation_event_interrupt_config

函数名称	hal_timer_commutation_event_interrupt_config
函数原形	int32_t hal_timer_commutation_event_interrupt_config(hal_timer_dev_struct *timer_dev, uint32_t trigger_source, uint32_t com_source, hal_timer_irq_struct *p_irq)
功能描述	TIMER换相事件配置，同时使能换相中断
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
trigger_source	触发源选择
TIMER_TRIGGER_SOURCE_ITI0	输入触发源为：ITI0
TIMER_TRIGGER_SOURCE_ITI1	输入触发源为：ITI1
TIMER_TRIGGER_SOURCE_ITI2	输入触发源为：ITI2
TIMER_TRIGGER_SOURCE_ITI3	输入触发源为：ITI3
TIMER_TRIGGER_SOURCE_DISABLE	无输入触发源

输入参数{in}	
com_source	CMT事件源选择
<i>TIMER_UPDATECTL_CCU</i>	CMTG位被置1时更新影子寄存器（CHxEN, CHxNEN和CHxCOMCTL位）
<i>TIMER_UPDATECTL_CCUTRI</i>	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新（CHxEN, CHxNEN和CHxCOMCTL位）
输入参数{in}	
p_irq	指针，指向hal_timer_irq_struct结构体，结构体成员参考 Table 3-18. 结构体hal_timer_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```

/* configureTIMER commutation event and enable CMT interrupt */

void timer0_commutation_interrupt_handle_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_irq_struct timer0_irq_parameter;

uint8_t commutation_num = 0;

timer0_irq_parameter.commutation_handle =
timer0_commutation_interrupt_handle_userdefine;

hal_timer_commutation_event_interrupt_config(&timer0_info,
TIMER_TRIGGER_SOURCE_DISABLE,                TIMER_UPDATECTL_CCU,
&timer0_irq_parameter);

void timer0_commutation_interrupt_handle_userdefine(void *ptr)
{
    commutation_num++;
}

```

函数 hal_timer_commutation_event_dma_config

函数hal_timer_commutation_event_dma_config描述见下表：

Table 3-96. 函数 hal_timer_commutation_event_dma_config

函数名称	hal_timer_commutation_event_dma_config
函数原形	int32_t hal_timer_commutation_event_dma_config(hal_timer_dev_struct *timer_dev, uint32_t trigger_source, uint32_t com_source, hal_timer_dma_handle_cb_struct *dmacb)

功能描述	TIMER换相事件配置，同时使能换相DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
trigger_source	触发源选择
TIMER_TRIGGER_SO URCE_ITI0	输入触发源为：ITI0
TIMER_TRIGGER_SO URCE_ITI1	输入触发源为：ITI1
TIMER_TRIGGER_SO URCE_ITI2	输入触发源为：ITI2
TIMER_TRIGGER_SO URCE_ITI3	输入触发源为：ITI3
TIMER_TRIGGER_SO URCE_DISABLE	无输入触发源
输入参数{in}	
com_source	CMT事件源选择
TIMER_UPDATECTL_ CCU	CMTG位被置1时更新影子寄存器（CHxEN, CHxNEN和CHxCOMCTL位）
TIMER_UPDATECTL_ CCUTRI	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新（CHxEN, CHxNEN和CHxCOMCTL位）
输入参数{in}	
dmacb	指针，指向hal_timer_dma_handle_cb_struct结构体，结构体成员参考 Table 3-19. 结构体hal_timer_dma_handle_cb_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```

/* configure TIMER commutation event and enable CMT DMA request */
void timer0_dmaerror(void *ptr);
void timer0_commutation_dmatc_userdefine(void *ptr);
hal_timer_dev_struct timer0_info;
hal_timer_dma_handle_cb_struct timer_dma_handle_cb;
uint8_t timer0_dmaerror_num = 0;
uint8_t timer0_commutation_dmatc_num = 0;

```

```

timer_dma_handle_cb.commutation_dma_full_transcom_handle =
&timer0_commutation_dmatc_userdefine;

timer_dma_handle_cb.error_handle = &timer0_dmaerror;

hal_timer_commutation_event_dma_config(&timer0_info,
TIMER_TRIGGER_SOURCE_DISABLE,          TIMER_UPDATECTL_CCU,
&timer_dma_handle_cb);

void timer0_dmaerror(void *ptr)
{
    timer0_dmaerror_num++;
}

void timer0_commutation_dmatc_userdefine(void *ptr)
{
    timer0_commutation_dmatc_num++;
}

```

函数 hal_timer_irq_handle_set

函数hal_timer_irq_handle_set描述见下表:

Table 3-97. 函数 hal_timer_irq_handle_set

函数名称	hal_timer_irq_handle_set
函数原形	int32_t hal_timer_irq_handle_set(hal_timer_dev_struct *timer_dev, hal_timer_irq_struct *p_irq)
功能描述	中断回调函数设置
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针, 指向hal_timer_dev_struct结构体, 结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输入参数{in}	
p_irq	指针, 指向hal_timer_irq_struct结构体, 结构体成员参考 Table 3-18. 结构体hal_timer_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码: HAL_ERR_NONE, HAL_ERR_ADDRESS

例如:

```
/* set user-defined interrupt callback function */
```

```
void timer0_input_capture_interrupt_handle_userdefine(void *ptr);

hal_timer_dev_struct timer0_info;

hal_timer_irq_struct timer0_irq_parameter;

uint8_t input_capture_num = 0;

timer0_irq_parameter.channelx_capture_handle =
timer0_input_capture_interrupt_handle_userdefine;

hal_timer_irq_handle_set(&timer0_info, &timer0_irq_parameter);

void timer0_input_capture_interrupt_handle_userdefine(void *ptr)
{
    input_capture_num++;
}
```

函数 hal_timer_irq_handle_all_reset

函数hal_timer_irq_handle_all_reset描述见下表：

Table 3-98. 函数 hal_timer_irq_handle_all_reset

函数名称	hal_timer_irq_handle_all_reset
函数原形	int32_t hal_timer_irq_handle_all_reset(hal_timer_dev_struct *timer_dev)
功能描述	复位所有中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	错误码：HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* reset all user-defined interrupt callback function */

hal_timer_dev_struct timer0_info;

hal_timer_irq_handle_all_reset(&timer0_info);
```

函数 hal_timer_irq

函数hal_timer_irq描述见下表：

Table 3-99. 函数 hal_timer_irq

函数名称	hal_timer_irq
函数原形	void hal_timer_irq(hal_timer_dev_struct *timer_dev)
功能描述	中断服务程序
先决条件	-
被调用函数	-
输入参数{in}	
timer_dev	指针，指向hal_timer_dev_struct结构体，结构体成员参考 Table 3-20. 结构体hal_timer_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TIMER interrupt handler content function, which is merely used in timer_handler */
hal_timer_dev_struct timer0_info;

hal_timer_irq (&timer0_info);
```

3.23. TSI

触摸传感控制器（TSI）为触摸按键、滑块、电容近距离感测等应用提供了简易的解决方案。章节[3.23.1](#)描述了TSI的寄存器列表，章节[3.23.2](#)对TSI库函数进行说明。

3.23.1. 外设寄存器描述

TSI寄存器列表如下表所示：

Table 3-499. TSI 寄存器

寄存器名称	寄存器描述
TSI_CTL0	控制寄存器 0
TSI_INTEN	中断使能寄存器
TSI_INTC	中断标志位清除寄存器
TSI_INTF	中断标志位寄存器
TSI_PHM	引脚迟滞模式寄存器
TSI_ASW	模拟开关寄存器
TSI_SAMPCFG	采样配置寄存器
TSI_CHCFG	通道配置寄存器
TSI_GCTL	组控制寄存器
TSI_GxCYCN (x=0..5)	组 x 周期数寄存器
TSI_CTL1	控制寄存器 1

3.23.2. 外设库函数说明

TSI库函数列表如下表所示：

表 3-500. TSI 库函数

库函数名称	库函数描述
hal_tsi_init	初始化TSI
hal_tsi_struct_init	使用默认值初始化TSI结构体
hal_tsi_deinit	复位TSI设备并初始化结构体
hal_tsi_start	启动TSI
hal_tsi_stop	停止TSI
hal_tsi_start_interrupt	启动TSI中断
hal_tsi_stop_interrupt	停止TSI中断
hal_tsi_irq	TSI中断处理函数，会在tsi_handler中调用
hal_tsi_irq_handle_set	设置用户定义的中断回调函数
hal_tsi_irq_handle_all_reset	复位所有用户定义的中断回调函数
hal_tsi_group_cycle_get	获取指定组的电荷转移序列完成周期数
hal_tsi_pins_config	配置TSI引脚
hal_tsi_poll_transfer	轮询电荷转移序列完成标志

枚举 hal_tsi_state_enum

表 3-501. 枚举 hal_tsi_state_enum

成员名称	功能描述
HAL_TSI_STATE_NONE	无（默认值）
HAL_TSI_STATE_RESET	TSI未被初始化或停止
HAL_TSI_STATE_BUSY	TSI繁忙
HAL_TSI_STATE_TIMEOUT	TSI产生超时
HAL_TSI_STATE_ERROR	TSI出错
HAL_TSI_STATE_READY	TSI准备

枚举 hal_tsi_error_enum

表 3-502. 枚举 hal_tsi_error_enum

成员名称	功能描述
HAL_TSI_ERROR_NONE	无错误
HAL_TSI_ERROR_SYSTEM	TSI内部错误
HAL_TSI_ERROR_MNERR	TSI最大周期数错误
HAL_TSI_ERROR_CONFIG	TSI配置错误

枚举 `hal_tsi_struct_type_enum`

表 3-503. 枚举 `hal_tsi_struct_type_enum`

成员名称	功能描述
HAL_TSI_INIT_STRUCT	TSI初始化结构体
HAL_TSI_IRQ_STRUCT	TSI irq结构体
HAL_TSI_DEV_STRUCT	TSI设备结构体

结构体 `hal_tsi_irq_struct`

表 3-504. 结构体 `hal_tsi_irq_struct`

成员名称	功能描述
tsi_cctcf_handle	TSI电荷转移完成处理函数
tsi_cctcf_handle	TSI最大周期数处理函数

结构体 `hal_tsi_dev_struct`

表 3-505. 结构体 `hal_tsi_dev_struct`

成员名称	功能描述
tsi_irq	TSI中断回调函数结构体指针
error_state	TSI错误状态
state	TSI状态
mutex	互斥量
priv	私有数据

结构体 `hal_tsi_init_struct`

表 3-506. 结构体 `hal_tsi_init_struct`

成员名称	功能描述
charge_time	充电状态时间
transfer_time	电荷转移状态时间
ctclk_div	电荷转移时钟分频因子
seq_max	最大周期数
extend_charge_state	扩展充电使能
ecclk_div	扩展充电时钟分频因子
extend_charge_time	扩展充电状态最大持续时间
pin_mode	引脚模式
edge_sel	边沿选择
trig_mode	触发模式选择
sample_pins	采样引脚
channel_pins	通道引脚
shield_pins	屏蔽引脚

函数 hal_tsi_init

函数hal_tsi_init描述见下表：

表 3-507. 函数 hal_tsi_init

函数名称	hal_tsi_init
函数原型	int32_t hal_tsi_init(hal_tsi_dev_struct *tsi_dev, hal_tsi_init_struct *tsi_init);
功能描述	初始化TSI
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输入参数{in}	
tsi_init	指向TSI初始化结构体的指针，结构体成员参考 表3-506. 结构体 hal_tsi_init_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```

/* initialize TSI */

hal_tsi_init_struct tsi_init_parameter;

hal_tsi_dev_struct tsi_info;

tsi_init_parameter.charge_time = TSI_CHARGE_2CTCLK;

tsi_init_parameter.transfer_time = TSI_TRANSFER_2CTCLK;

tsi_init_parameter.ctclk_div = TSI_CTCDIV_DIV32;

tsi_init_parameter.seq_max = TSI_MAXNUM2047;

tsi_init_parameter.extend_charge_state = TSI_EXTEND_CHARGE_DISABLE;

tsi_init_parameter.pin_mode = TSI_OUTPUT_LOW;

tsi_init_parameter.trig_mode = TSI_HW_TRIGGER_DISABLE;

tsi_init_parameter.sample_pins = TSI_G5P0;

tsi_init_parameter.channel_pins = TSI_G5P1 | TSI_G5P2 | TSI_G5P3;

hal_tsi_init(&tsi_info, &tsi_init_parameter);

```

函数 hal_tsi_struct_init

函数hal_tsi_struct_init描述见下表：

表 3-508. 函数 hal_tsi_struct_init

函数名称	hal_tsi_struct_init
函数原型	void hal_tsi_struct_init(hal_tsi_struct_type_enum hal_struct_type, void *p_struct);
功能描述	使用默认值初始化TSI结构体
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输入参数{in}	
p_struct	指向包含TSI配置信息的结构体
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* initialize TSI initialization structure */
```

```
hal_tsi_init_struct tsi_init_parameter;
```

```
hal_tsi_struct_init(HAL_TSI_INIT_STRUCTURE, &tsi_init_parameter);
```

函数 hal_tsi_deinit

函数hal_tsi_deinit描述见下表：

表 3-509. 函数 hal_tsi_deinit

函数名称	hal_tsi_deinit
函数原型	int32_t hal_tsi_deinit(hal_tsi_dev_struct *tsi_dev);
功能描述	复位TSI设备并初始化结构体
先决条件	-
被调用函数	hal_rcu_periph_reset_enable / hal_rcu_periph_reset_disable
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* deinitialize TSI device structure and init structure */
```

```
hal_tsi_dev_struct tsi_info;
```

```
hal_tsi_deinit(&tsi_info);
```

函数 hal_tsi_start

函数hal_tsi_start描述见下表：

表 3-510. 函数 hal_tsi_start

函数名称	hal_tsi_start
函数原型	int32_t hal_tsi_start(hal_tsi_dev_struct *tsi_dev);
功能描述	启动TSI
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* start TSI module function */
```

```
hal_tsi_dev_struct tsi_info;
```

```
hal_tsi_start(&tsi_info);
```

函数 hal_tsi_stop

函数hal_tsi_stop描述见下表：

表 3-511. 函数 hal_tsi_stop

函数名称	hal_tsi_stop
函数原型	int32_t hal_tsi_stop(hal_tsi_dev_struct *tsi_dev);
功能描述	停止TSI
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输出参数{out}	

-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* stop tsi module function */
```

```
hal_tsi_dev_struct tsi_info;
```

```
hal_tsi_stop(&tsi_info);
```

函数 hal_tsi_start_interrupt

函数hal_tsi_start_interrupt描述见下表：

表 3-512. 函数 hal_tsi_start_interrupt

函数名称	hal_tsi_start_interrupt
函数原型	int32_t hal_tsi_start_interrupt(hal_tsi_dev_struct *tsi_dev, hal_tsi_irq_struct *p_irq);
功能描述	启动TSI中断
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输入参数{in}	
p_irq	指向TSI设备信息结构体的指针，结构体成员参考 表3-504. 结构体 hal_tsi_irq_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* start TSI interrupt */
```

```
hal_tsi_dev_struct tsi_info;
```

```
hal_tsi_irq_struct tsi_irq;
```

```
hal_tsi_start_interrupt(&tsi_info, &tsi_irq);
```

函数 hal_tsi_stop_interrupt

函数hal_tsi_stop_interrupt描述见下表：

表 3-513. 函数 hal_tsi_stop_interrupt

函数名称	hal_tsi_stop_interrupt
函数原型	int32_t hal_tsi_stop_interrupt(hal_tsi_dev_struct *tsi_dev);
功能描述	停止TSI中断
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_ADDRESS, HAL_ERR_NONE

例如：

```
/* stop TSI interrupt */
hal_tsi_dev_struct tsi_info;
hal_tsi_stop_interrupt(&tsi_info);
```

函数 hal_tsi_irq

函数hal_tsi_irq描述见下表：

表 3-514. 函数 hal_tsi_irq

函数名称	hal_tsi_irq
函数原型	void hal_tsi_irq(hal_tsi_dev_struct *tsi_dev);
功能描述	TSI中断处理函数，会在tsi_handler中调用
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TSI initialization structure */
/* the function is used in the relative interrupt routine */
hal_tsi_dev_struct tsi_info;
```

```
void TSI_IRQHandler(void)
{
    hal_tsi_irq(&tsi_info);
}
```

函数 hal_tsi_irq_handle_set

函数hal_tsi_irq_handle_set描述见下表：

表 3-515. 函数 hal_tsi_irq_handle_set

函数名称	hal_tsi_irq_handle_set
函数原型	void hal_tsi_irq_handle_set(hal_tsi_dev_struct *tsi_dev, hal_tsi_irq_struct *p_irq);
功能描述	设置用户定义的中断回调函数
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输入参数{in}	
p_irq	指向TSI设备信息结构体的指针，结构体成员参考 表3-504. 结构体 hal_tsi_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TSI interrupt */
hal_tsi_dev_struct tsi_info;
hal_tsi_irq_struct tsi_irq;
hal_tsi_irq_handle_set(&tsi_info, &tsi_irq);
```

函数 hal_tsi_irq_handle_all_reset

函数hal_tsi_irq_handle_all_reset描述见下表：

表 3-516. 函数 hal_tsi_irq_handle_all_reset

函数名称	hal_tsi_irq_handle_all_reset
函数原型	void hal_tsi_irq_handle_all_reset(hal_tsi_dev_struct *tsi_dev);
功能描述	复位所有用户定义的中断回调函数
先决条件	-

被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback function */
```

```
hal_tsi_dev_struct tsi_info;
```

```
hal_tsi_irq_handle_all_reset (&tsi_info);
```

函数 hal_tsi_group_cycle_get

函数hal_tsi_group_cycle_get描述见下表：

表 3-517. 函数 hal_tsi_group_cycle_get

函数名称	hal_tsi_group_cycle_get
函数原型	uint16_t hal_tsi_group_cycle_get(hal_tsi_dev_struct *tsi_dev, uint8_t group_id);
功能描述	获取指定组的电荷转移序列完成周期数
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输入参数{in}	
group_id	TSI组ID
TSI_GROUP_IDX0	TSI组0
TSI_GROUP_IDX1	TSI组1
TSI_GROUP_IDX2	TSI组2
TSI_GROUP_IDX3	TSI组3
TSI_GROUP_IDX4	TSI组4
TSI_GROUP_IDX5	TSI组5
输出参数{out}	
-	-
返回值	
uint16_t	0x0 – 0x3FFF

例如：

```
/* get group0 cycle value */
```

```
hal_tsi_dev_struct tsi_info;
```

```
uint16_t sample_value
```

```
hal_tsi_group_cycle_get(&tsi_info, TSI_GROUP_IDX0);
```

函数 hal_tsi_pins_config

函数hal_tsi_pins_config描述见下表：

表 3-518. 函数 hal_tsi_pins_config

函数名称	hal_tsi_pins_config
函数原型	void hal_tsi_pins_config(hal_tsi_dev_struct *tsi_dev, hal_tsi_init_struct *pins_config);
功能描述	配置TSI引脚
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输入参数{in}	
pins_config	指向hal_tsi_init_struct的指针，结构体成员参考 表3-506. 结构体 hal_tsi_init_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TSI pins */
hal_tsi_init_struct tsi_init_parameter;
hal_tsi_dev_struct tsi_info;
tsi_init_parameter.sample_pins = TSI_G4P0;
tsi_init_parameter.channel_pins = TSI_G4P1 | TSI_G4P2 | TSI_G4P3;
hal_tsi_pins_config(&tsi_info, &tsi_init_parameter);
```

函数 hal_tsi_poll_transfer

函数hal_tsi_poll_transfer描述见下表：

表 3-519. 函数 hal_tsi_poll_transfer

函数名称	hal_tsi_poll_transfer
函数原型	void hal_tsi_poll_transfer(hal_tsi_dev_struct *tsi_dev);

功能描述	轮询电荷转移序列完成标志
先决条件	-
被调用函数	-
输入参数{in}	
tsi_dev	指向TSI设备信息结构体的指针，结构体成员参考 表3-505. 结构体 hal_tsi_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* TSI poll for charge-transfer sequence complete */
```

```
hal_tsi_dev_struct tsi_info;
```

```
hal_tsi_poll_transfer(&tsi_info);
```

3.24. UART

通用同步异步收发器(USART)提供了一个灵活方便的串行数据交换接口，章节[3.24.1](#)描述了USART的寄存器列表，章节[3.24.2](#)对UART库函数进行说明。

3.24.1. 外设寄存器说明

USART寄存器列表如下表所示：

表 3-520. USART 寄存器

寄存器名称	寄存器描述
USART_CTL0	控制寄存器0
USART_CTL1	控制寄存器1
USART_CTL2	控制寄存器2
USART_BAUD	波特率寄存器
USART_GP	保护时间和预分频器寄存器
USART_RT	接收超时寄存器
USART_CMD	请求寄存器
USART_STAT	状态寄存器
USART_INTC	中断标志清除寄存器
USART_RDATA	数据接收寄存器
USART_TDATA	数据发送寄存器
USART_RFCS	接收FIFO控制和状态寄存器

3.24.2. 外设库函数说明

UART库函数列表如下表所示：

表 3-521. UART 库函数

库函数名称	库函数描述
hal_uart_struct_init	根据已有的hal_uart_struct_type_enum类型，初始化UART结构体，当定义了一个结构体后需要调用该函数将该结构体成员赋为初始值
hal_uart_deinit	重新初始化UART设备信息结构体
hal_uart_init	根据初始化结构体及USART外设信息，对UART设备信息结构体进行初始化
hal_uart_irq	处理所有UART中断请求
hal_uart_irq_handle_set	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
hal_uart_irq_handle_all_reset	复位所有的用户自定义的中断回调函数
hal_uart_transmit_poll	采用轮询方式发送数据，该函数为阻塞式
hal_uart_receive_poll	采用轮询方式接收数据，该函数为阻塞式
hal_uart_transmit_interrupt	采用中断方式发送数据，在发送完成后调用用户回调函数，该函数为非阻塞式
hal_uart_receive_interrupt	采用中断方式接收数据，在接收完成后调用用户回调函数，该函数为非阻塞式
hal_uart_transmit_dma	采用DMA方式发送数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
hal_uart_receive_dma	采用DMA方式接收数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
hal_uart_dma_pause	传输过程中暂停UART的DMA传输
hal_uart_dma_resume	传输过程中恢复UART的DMA传输
hal_uart_transmit_stop	停止UART发送数据，该函数为阻塞式
hal_uart_receive_stop	停止UART接收数据，该函数为阻塞式

枚举 hal_uart_struct_type_enum

表 3-522. 枚举 hal_uart_struct_type_enum

成员名称	功能描述
HAL_UART_INIT_STRUCT	UART初始化结构体
HAL_UART_DEV_STRUCT	UART中断回调函数指针结构体
HAL_UART_USER_CALLBACK_STRUCT	UART DMA回调函数指针结构体
HAL_UART_IRQ_INIT_STRUCT	UART设备信息结构体

枚举 `hal_uart_work_mode_enum`

表 3-523. 枚举 `hal_uart_work_mode_enum`

成员名称	功能描述
<code>UART_WORK_MODE_ASYNC</code>	UART异步模式
<code>UART_WORK_MODE_SINGLE_WIRE</code>	UART单线模式
<code>UART_WORK_MODE_MULTIPROCESSOR</code>	UART多处理器模式
<code>UART_WORK_MODE_LIN</code>	UART LIN模式

结构体 `hal_uart_init_struct`

表 3-524. 结构体 `hal_uart_init_struct`

成员名称	功能描述
<code>work_mode</code>	工作模式
<code>baudrate</code>	波特率
<code>parity</code>	奇偶校验
<code>word_length</code>	一帧数据的长度
<code>stop_bit</code>	停止位
<code>direction</code>	传输方向
<code>over_sample</code>	过采样模式
<code>sample_method</code>	采样方式
<code>hardware_flow</code>	硬件流控
<code>rx_fifo_en</code>	接收FIFO使能
<code>timeout_enable</code>	接收超时使能
<code>timeout_value</code>	接收超时时间
<code>first_bit_msb</code>	MSB先发送
<code>tx_rx_swap</code>	Tx引脚和Rx引脚交换
<code>rx_level_invert</code>	反转Rx引脚有效电平
<code>tx_level_invert</code>	反转Tx引脚有效电平
<code>data_bit_invert</code>	数据反转
<code>overrun_disable</code>	禁能过载
<code>rx_error_dma_stop</code>	接收错误时禁能DMA
<code>break_frame_length</code>	LIN断开帧长度
<code>rs485_mode</code>	RS485模式
<code>de_polarity</code>	RS485模式下驱动使能极性
<code>de_assertion_time</code>	RS485模式下驱动使能置高时间
<code>de_deassertion_time</code>	RS485模式下驱动使能置低时间
<code>wakeup_mode</code>	多处理器模式下的唤醒模式
<code>address</code>	多处理器模式下的唤醒地址
<code>addr_length</code>	多处理器模式下的唤醒地址长度

结构体 `hal_uart_irq_struct`

表 3-525. 结构体 `hal_uart_irq_struct`

成员名称	功能描述
<code>receive_complete_handle</code>	接收完成回调函数
<code>receive_timeout_handle</code>	接收超时回调函数
<code>transmit_ready_handle</code>	发送就绪回调函数
<code>transmit_complete_handle</code>	发送完成回调函数
<code>error_handle</code>	错误回调函数
<code>wakeup_handle</code>	唤醒回调函数
<code>idle_line_detected_handle</code>	检测到空闲帧回调函数
<code>address_match_handle</code>	地址匹配回调函数
<code>lin_break_detected_handle</code>	检测到LIN断开回调函数
<code>cts_change_handle</code>	CTS变化回调函数

枚举 `hal_uart_run_state_enum`

表 3-526. 枚举 `hal_uart_run_state_enum`

成员名称	功能描述
<code>UART_STATE_FREE</code>	UART空闲
<code>UART_STATE_BUSY</code>	UART繁忙

结构体 `hal_uart_dev_struct`

表 3-527. 结构体 `hal_uart_dev_struct`

成员名称	功能描述
<code>periph</code>	USART外设
<code>uart_irq</code>	中断回调函数指针，形参成员参考 表 3-525. 结构体 <code>hal_uart_irq_struct</code>
<code>p_dma_rx</code>	DMA接收指针，指向DMA设备信息结构体
<code>p_dma_tx</code>	DMA发送指针，指向DMA设备信息结构体
<code>txbuffer</code>	发送缓冲区
<code>rxbuffer</code>	接收缓冲区
<code>data_bit_mask</code>	数据屏蔽位
<code>last_error</code>	上一个错误码
<code>error_state</code>	错误码
<code>tx_state</code>	发送状态，形参成员参考 表3-526. 枚举 <code>hal_uart_run_state_enum</code>
<code>rx_state</code>	接收状态，形参成员参考 表3-526. 枚举 <code>hal_uart_run_state_enum</code>
<code>rx_callback</code>	接收回调指针
<code>tx_callback</code>	发送回调指针
<code>mutex</code>	互斥锁和解锁状态
<code>priv</code>	私有指针

结构体 `hal_uart_user_callback_struct`

表 3-528. 结构体 `hal_uart_user_callback_struct`

成员名称	功能描述
<code>complete_func</code>	传输完成回调
<code>error_func</code>	错误回调

函数 `hal_uart_struct_init`

函数`hal_uart_struct_init`描述见下表：

表 3-529. 函数 `hal_uart_struct_init`

函数名称	<code>hal_uart_struct_init</code>
函数原型	<code>void hal_uart_struct_init(hal_uart_struct_type_enum hal_struct_type, void *p_struct);</code>
功能描述	根据已有的 <code>hal_uart_struct_type_enum</code> 类型，初始化UART结构体，当定义了一个结构体后需要调用该函数将该结构体成员赋为初始值
先决条件	-
输入参数{in}	
hal_struct_type	结构体类型，形参成员参考 表3-522. 枚举hal_uart_struct_type_enum
输入参数{in}	
p_struct	指向结构体的空指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_uart_irq_struct uart_irq;
```

```
/* initialize a uart irq structure */
```

```
hal_uart_struct_init(HAL_UART_IRQ_INIT_STRUCT, &uart_irq);
```

函数 `hal_uart_deinit`

函数`hal_uart_deinit`描述见下表：

表 3-530. 函数 `hal_uart_deinit`

函数名称	<code>hal_uart_deinit</code>
函数原型	<code>void hal_uart_deinit(hal_uart_dev_struct *uart);</code>
功能描述	重新初始化UART设备信息结构体
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct

输出参数{out}	
-	-
返回值	
-	-

例如:

```
hal_uart_dev_struct uart0_info;
```

```
/* deinitialize the device information structure */
```

```
hal_uart_deinit(&uart0_info);
```

函数 hal_uart_init

函数hal_uart_init描述见下表:

表 3-531. 函数 hal_uart_init

函数名称	hal_uart_init
函数原型	int32_t hal_uart_init(hal_uart_dev_struct *uart, uint32_t periph, hal_uart_init_struct *p_init);
功能描述	根据初始化结构体及USART外设信息, 对UART设备信息结构体进行初始化
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针, 结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输入参数{in}	
periph	USART外设
USARTx	x=0,1
输入参数{in}	
p_init	指向初始化结构体的指针, 结构体成员参考 表 3-524. 结构体 hal_uart_init_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如:

```
hal_uart_dev_struct uart0_info;
```

```
hal_uart_init_struct uart0_init;
```

```
/* initialize the structures */
```

```
hal_uart_struct_init(HAL_UART_DEV_STRUCT, &uart0_info);
```

```
hal_uart_struct_init(HAL_UART_INIT_STRUCT, &uart0_init);
```



```

uart0_init.work_mode = UART_WORK_MODE_ASYNC;

uart0_init.baudrate = 115200;

uart0_init.parity = UART_PARITY_NONE;

uart0_init.word_length = UART_WORD_LENGTH_8BIT;

uart0_init.stop_bit = UART_STOP_BIT_1;

uart0_init.direction = UART_DIRECTION_RX_TX;

uart0_init.over_sample = UART_OVER_SAMPLE_16;

uart0_init.hardware_flow = UART_HARDWARE_FLOW_NONE;

uart0_init.sample_method = UART_THREE_SAMPLE_BIT;

hal_uart_init(&uart0_info, USART0, &uart0_init);

```

函数 hal_uart_irq

函数hal_uart_irq描述见下表：

表 3-532. 函数 hal_uart_irq

函数名称	hal_uart_irq
函数原型	void hal_uart_irq(hal_uart_dev_struct *uart);
功能描述	处理所有UART中断请求
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* process the UART interrupt */

hal_uart_irq(&uart0_info);

```

函数 hal_uart_irq_handle_set

函数hal_uart_irq_handle_set描述见下表：

表 3-533. 函数 hal_uart_irq_handle_set

函数名称	hal_uart_irq_handle_set
函数原型	void hal_uart_irq_handle_set(hal_uart_dev_struct *uart, hal_uart_irq_struct *p_irq);

功能描述	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输入参数{in}	
p_irq	指向中断结构体的指针，结构体成员参考 表 3-525. 结构体 hal_uart_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_uart_irq_struct uart_irq;

hal_uart_struct_init(HAL_UART_IRQ_INIT_STRUCT, &uart_irq);

/* set the wakeup handle function */

uart_irq.wakeup_handle = uart_wakeup_cb;

hal_uart_irq_handle_set(&uart0_info, &uart_irq);
```

函数 hal_uart_irq_handle_all_reset

函数hal_uart_irq_handle_all_reset描述见下表：

表 3-534. 函数 hal_uart_irq_handle_all_reset

函数名称	hal_uart_irq_handle_all_reset
函数原型	void hal_uart_irq_handle_all_reset(hal_uart_dev_struct *uart);
功能描述	复位所有的用户自定义的中断回调函数
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback */

hal_uart_irq_handle_all_reset(&uart0_info);
```

函数 hal_uart_transmit_poll

函数hal_uart_transmit_poll描述见下表:

表 3-535. 函数 hal_uart_transmit_poll

函数名称	hal_uart_transmit_poll
函数原型	int32_t hal_uart_transmit_poll(hal_uart_dev_struct *uart, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	采用轮询方式发送数据, 该函数为阻塞式
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针, 结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY, HAL_ERR_TIMEOUT

例如:

```
#define TRANSMIT_SIZE          6

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};

/* transmit using polling mode */

hal_uart_transmit_poll(&uart0_info, txbuffer, TRANSMIT_SIZE, 0x1FFFFF);
```

函数 hal_uart_receive_poll

函数hal_uart_receive_poll描述见下表:

表 3-536. 函数 hal_uart_receive_poll

函数名称	hal_uart_receive_poll
函数原型	int32_t hal_uart_receive_poll(hal_uart_dev_struct *uart, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	采用轮询方式接收数据, 该函数为阻塞式
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针, 结构体成员参考 表 3-527. 结构体

	hal_uart_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY, HAL_ERR_TIMEOUT

例如：

```
#define TRANSFER_SIZE                16

uint8_t rxbuffer[TRANSFER_SIZE];

/* receive using polling mode */

hal_uart_receive_poll(&uart1_info, rxbuffer, TRANSFER_SIZE, 0x7FFFFFFF);
```

函数 hal_uart_transmit_interrupt

函数hal_uart_transmit_interrupt描述见下表：

表 3-537. 函数 hal_uart_transmit_interrupt

函数名称	hal_uart_transmit_interrupt
函数原型	int32_t hal_uart_transmit_interrupt(hal_uart_dev_struct *uart, uint8_t *p_buffer, uint32_t length, hal_uart_user_cb p_user_func);
功能描述	采用中断方式发送数据，在发送完成后调用用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSMIT_SIZE          6

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};

__IO FlagStatus tx_end = RESET;

void tx_complete(hal_uart_dev_struct *uart) {
    tx_end = SET;
}

/* transmit using interrupt mode */

hal_uart_transmit_interrupt(&uart0_info, txbuffer, TRANSMIT_SIZE, tx_complete);

/* check whether the transfer is completed or not */

while(RESET == tx_end){
}
```

函数 hal_uart_receive_interrupt

函数hal_uart_receive_interrupt描述见下表：

表 3-538. 函数 hal_uart_receive_interrupt

函数名称	hal_uart_receive_interrupt
函数原型	int32_t hal_uart_receive_interrupt(hal_uart_dev_struct *uart, uint8_t *p_buffer, uint32_t length, hal_uart_user_cb p_user_func);
功能描述	采用中断方式接收数据，在接收完成后调用用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSFER_SIZE
```

16

```
uint8_t rxbuffer[TRANSFER_SIZE];

__IO FlagStatus rx_end = RESET;

void rx_complete(hal_uart_dev_struct *uart) {

    rx_end = SET;

}

/* receive data using interrupt mode */

hal_uart_receive_interrupt(&uart0_info, rxbuffer, TRANSFER_SIZE, rx_complete);

/* check whether the transfer is completed or not */

while(RESET == rx_end){

}
```

函数 hal_uart_transmit_dma

函数hal_uart_transmit_dma描述见下表:

表 3-539. 函数 hal_uart_transmit_dma

函数名称	hal_uart_transmit_dma
函数原型	int32_t hal_uart_transmit_dma(hal_uart_dev_struct *uart, uint8_t *p_buffer, uint16_t length, hal_uart_user_callback_struct *p_func);
功能描述	采用DMA方式发送数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_func	执行用户回调结构体的指针
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如:

```
#define TRANSMIT_SIZE          6

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};
```

```

__IO FlagStatus tx_end = RESET;

hal_uart_user_callback_struct user_cb;

void tx_complete(hal_uart_dev_struct *uart) {

    tx_end = SET;

}

/* initialize the user callback structure */

hal_uart_struct_init(HAL_UART_USER_CALLBACK_STRUCT, &user_cb);

user_cb.error_func = NULL;

user_cb.complete_func = tx_complete;

/* transmit using DMA mode */

hal_uart_transmit_dma(&uart0_info, txbuffer, TRANSMIT_SIZE, &user_cb);

/* check whether the transfer is completed or not */

while(RESET == tx_end){

}

```

函数 hal_uart_receive_dma

函数hal_uart_receive_dma描述见下表：

表 3-540. 函数 hal_uart_receive_dma

函数名称	hal_uart_receive_dma
函数原型	int32_t hal_uart_receive_dma(hal_uart_dev_struct *uart, uint8_t *p_buffer, uint16_t length, hal_uart_user_callback_struct *p_func);
功能描述	采用DMA方式接收数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_func	执行用户回调结构体的指针
输出参数{out}	
-	-

返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如:

```
#define TRANSFER_SIZE                16

uint8_t rxbuffer[TRANSFER_SIZE];

__IO FlagStatus rx_end = RESET;

hal_uart_user_callback_struct user_cb;

void rx_complete(hal_uart_dev_struct *uart) {
    rx_end = SET;
}

/* initilize the user callback structure */

hal_uart_struct_init(HAL_UART_USER_CALLBACK_STRUCT, &user_cb);

user_cb.error_func = NULL;

user_cb.complete_func = rx_complete;

/* receive data using interrupt mode */

hal_uart_receive_dma(&uart1_info, rxbuffer, TRANSFER_SIZE, &user_cb);

/* check whether the transfer is completed or not */

while(RESET == rx_end){
}
```

函数 hal_uart_dma_pause

函数hal_uart_dma_pause描述见下表:

表 3-541. 函数 hal_uart_dma_pause

函数名称	hal_uart_dma_pause
函数原型	int32_t hal_uart_dma_pause(hal_uart_dev_struct *uart);
功能描述	传输过程中暂停UART的DMA传输
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* pause UART DMA transfer */
hal_uart_dma_pause(&uart0_info);
```

函数 hal_uart_dma_resume

函数hal_uart_dma_resume描述见下表：

表 3-542. 函数 hal_uart_dma_resume

函数名称	hal_uart_dma_resume
函数原型	int32_t hal_uart_dma_resume(hal_uart_dev_struct *uart);
功能描述	传输过程中恢复UART的DMA传输
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* resume UART DMA transfer */
hal_uart_dma_resume(&uart0_info);
```

函数 hal_uart_transmit_stop

函数hal_uart_transmit_stop描述见下表：

表 3-543. 函数 hal_uart_transmit_stop

函数名称	hal_uart_transmit_stop
函数原型	int32_t hal_uart_transmit_stop(hal_uart_dev_struct *uart);
功能描述	停止UART发送数据，该函数为阻塞式
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* stop UART transmit transfer */
```

```
hal_uart_transmit_stop(&uart0_info);
```

函数 hal_uart_receive_stop

函数hal_uart_receive_stop描述见下表:

表 3-544. 函数 hal_uart_receive_stop

函数名称	hal_uart_receive_stop
函数原型	int32_t hal_uart_receive_stop(hal_uart_dev_struct *uart);
功能描述	停止UART接收数据，该函数为阻塞式
先决条件	-
输入参数{in}	
uart	指向设备信息结构体的指针，结构体成员参考 表 3-527. 结构体 hal_uart_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如:

```
/* stop UART receive transfer */
```

```
hal_uart_receive_stop(&uart0_info);
```

3.25. USRT

通用同步异步收发器(USART)提供了一个灵活方便的串行数据交换接口，章节[3.25.1](#)描述了USART的寄存器列表，章节[3.25.2](#)对USRT库函数进行说明。

3.25.1. 外设寄存器说明

USART寄存器列表如下表所示:

表 3-545. USART 寄存器

寄存器名称	寄存器描述
USART_CTL0	控制寄存器0
USART_CTL1	控制寄存器1
USART_CTL2	控制寄存器2
USART_BAUD	波特率寄存器
USART_GP	保护时间和预分频器寄存器
USART_RT	接收超时寄存器
USART_CMD	请求寄存器
USART_STAT	状态寄存器

寄存器名称	寄存器描述
USART_INTIC	中断标志清除寄存器
USART_RDATA	数据接收寄存器
USART_TDATA	数据发送寄存器
USART_RFCS	接收FIFO控制和状态寄存器

3.25.2. 外设库函数说明

USRT库函数列表如下表所示：

表 3-546. USRT 库函数

库函数名称	库函数描述
hal_usrt_struct_init	根据已有的hal_usrt_struct_type_enum类型，初始化USRT结构体，当定义了一个结构体后需要调用该函数将该结构体成员赋为初始值
hal_usrt_deinit	重新初始化USRT设备信息结构体
hal_usrt_init	根据初始化结构体及USART外设信息，对USRT设备信息结构体进行初始化
hal_usrt_irq	处理所有USRT中断请求
hal_usrt_irq_handle_set	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
hal_usrt_irq_handle_all_reset	复位所有的用户自定义的中断回调函数
hal_usrt_transmit_poll	采用轮询方式发送数据，该函数为阻塞式
hal_usrt_receive_poll	采用轮询方式接收数据，该函数为阻塞式
hal_usrt_transmit_receive_poll	采用轮询方式发送接收数据，该函数为阻塞式
hal_usrt_transmit_interrupt	采用中断方式发送数据，在发送完成后调用用户回调函数，该函数为非阻塞式
hal_usrt_receive_interrupt	采用中断方式接收数据，在接收完成后调用用户回调函数，该函数为非阻塞式
hal_usrt_transmit_receive_interrupt	采用中断方式发送接收数据，在传输完成后调用用户回调函数，该函数为非阻塞式
hal_usrt_transmit_dma	采用DMA方式发送数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
hal_usrt_receive_dma	采用DMA方式接收数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
hal_usrt_transmit_receive_dma	采用DMA方式发送接收数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
hal_usrt_dma_pause	传输过程中暂停USRT的DMA传输
hal_usrt_dma_resume	传输过程中恢复USRT的DMA传输
hal_usrt_transfer_stop	停止USRT传输数据，该函数为阻塞式

枚举 `hal_usrt_struct_type_enum`

表 3-547. 枚举 `hal_usrt_struct_type_enum`

成员名称	功能描述
<code>HAL_USRT_INIT_STRUCT</code>	USRT初始化结构体
<code>HAL_USRT_DEV_STRUCT</code>	USRT中断回调函数指针结构体
<code>HAL_USRT_USER_CALLBACK_STRUCT</code>	USRT DMA回调函数指针结构体

结构体 `hal_usrt_init_struct`

表 3-548. 结构体 `hal_usrt_init_struct`

成员名称	功能描述
<code>baudrate</code>	波特率
<code>parity</code>	奇偶校验
<code>word_length</code>	一帧数据的长度
<code>stop_bit</code>	停止位
<code>direction</code>	传输方向
<code>rx_fifo_en</code>	接收FIFO使能
<code>clock_polarity</code>	时钟极性
<code>clock_phase</code>	时钟相位
<code>clock_length_lastbit</code>	时钟长度（最后一位时钟是否输出到CK管脚）

结构体 `hal_usrt_irq_struct`

表 3-549. 结构体 `hal_usrt_irq_struct`

成员名称	功能描述
<code>receive_complete_handle</code>	接收完成回调函数
<code>transmit_ready_handle</code>	发送就绪回调函数
<code>transmit_complete_handle</code>	发送完成回调函数
<code>error_handle</code>	错误回调函数

枚举 `hal_usrt_run_state_enum`

表 3-550. 枚举 `hal_usrt_run_state_enum`

成员名称	功能描述
<code>USRT_STATE_FREE</code>	USRT空闲
<code>USRT_STATE_BUSY</code>	USRT繁忙
<code>USRT_STATE_BUSY_TX_RX</code>	USRT发送接收繁忙

结构体 hal_usrt_dev_struct

表 3-551. 结构体 hal_usrt_dev_struct

成员名称	功能描述
periph	USART外设
usrt_irq	中断回调函数指针，形参成员参考 表 3-549. 结构体hal_usrt_irq_struct
p_dma_rx	DMA接收指针，指向DMA设备信息结构体
p_dma_tx	DMA发送指针，指向DMA设备信息结构体
txbuffer	发送缓冲区
rxbuffer	接收缓冲区
data_bit_mask	数据屏蔽位
last_error	上一个错误码
error_state	错误码
tx_state	发送状态
rx_state	接收状态
rx_callback	接收回调指针
tx_callback	发送回调指针
mutex	互斥锁和解锁状态
priv	私有指针

结构体 hal_usrt_user_callback_struct

表 3-552. 结构体 hal_usrt_user_callback_struct

成员名称	功能描述
complete_func	传输完成回调
error_func	错误回调

函数 hal_usrt_struct_init

函数hal_usrt_struct_init描述见下表：

表 3-553. 函数 hal_usrt_struct_init

函数名称	hal_usrt_struct_init
函数原型	void hal_usrt_struct_init(hal_usrt_struct_type_enum hal_struct_type, void *p_struct);
功能描述	根据已有的hal_usrt_struct_type_enum类型，初始化USRT结构体，当定义了一个结构体后需要调用该函数将该结构体成员赋为初始值
先决条件	-
输入参数{in}	
hal_struct_type	结构体类型，形参成员参考 表3-547. 枚举hal_usrt_struct_type_enum
输入参数{in}	
p_struct	指向结构体的空指针
输出参数{out}	

-	-
返回值	
-	-

例如：

```
hal_usrt_dev_struct usrt0_info;
```

```
/* initialize a usrt device information structure */
```

```
hal_usrt_struct_init(HAL_USRT_DEV_STRUCT, &usrt0_info);
```

函数 hal_usrt_deinit

函数hal_usrt_deinit描述见下表：

表 3-554. 函数 hal_usrt_deinit

函数名称	hal_usrt_deinit
函数原型	void hal_usrt_deinit(hal_usrt_dev_struct *usrt);
功能描述	重新初始化USRT设备信息结构体
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_usrt_dev_struct usrt0_info;
```

```
/* deinitialize the device information structure */
```

```
hal_usrt_deinit(&usrt0_info);
```

函数 hal_usrt_init

函数hal_usrt_init描述见下表：

表 3-555. 函数 hal_usrt_init

函数名称	hal_usrt_init
函数原型	int32_t hal_usrt_init(hal_usrt_dev_struct *usrt, uint32_t periph, hal_usrt_init_struct *p_init);
功能描述	根据初始化结构体及USART外设信息，对USRT设备信息结构体进行初始化
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体

	<u>hal_usrt_dev_struct</u>
输入参数{in}	
periph	USART外设
<i>USARTx</i>	x=0,1
输入参数{in}	
p_init	指向初始化结构体的指针，结构体成员参考 <u>表 3-548. 结构体 hal_usrt_init_struct</u>
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如：

```

hal_usrt_dev_struct usrt0_info;

hal_usrt_init_struct usrt0_init;

/* initialize the structures */

hal_usrt_struct_init(HAL_USRT_DEV_STRUCT, &usrt0_info);
hal_usrt_struct_init(HAL_USRT_INIT_STRUCT, &usrt0_init);

usrt0_init.baudrate = 115200;

usrt0_init.parity = USRT_PARITY_NONE;

usrt0_init.word_length = USRT_WORD_LENGTH_8BIT;

usrt0_init.stop_bit = USRT_STOP_BIT_1;

usrt0_init.direction = USRT_DIRECTION_RX_TX;

usrt0_init.rx_fifo_en = DISABLE;

usrt0_init.clock_polarity = USRT_CLOCK_POLARITY_LOW;

usrt0_init.clock_phase = USRT_CLOCK_PHASE_1CK;

usrt0_init.clock_length_lastbit = USRT_LAST_BIT_NOT_OUTPUT;

hal_usrt_init(&usrt0_info, USART0, &usrt0_init);

```

函数 **hal_usrt_irq**

函数hal_usrt_irq描述见下表：

表 3-556. 函数 hal_usrt_irq

函数名称	hal_usrt_irq
函数原型	void hal_usrt_irq(hal_usrt_dev_struct *usrt);
功能描述	处理所有USRT中断请求

先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* process the USRT interrupt */
```

```
hal_usrt_irq(&usrt0_info);
```

函数 hal_usrt_irq_handle_set

函数hal_usrt_irq_handle_set描述见下表：

表 3-557. 函数 hal_usrt_irq_handle_set

函数名称	hal_usrt_irq_handle_set
函数原型	void hal_usrt_irq_handle_set(hal_usrt_dev_struct *usrt, hal_usrt_irq_struct *p_irq);
功能描述	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输入参数{in}	
p_irq	指向中断结构体的指针，结构体成员参考 表 3-549. 结构体hal_usrt_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_usrt_irq_struct usrt_irq;
```

```
/* set the error handle function */
```

```
usrt_irq.error_handle = usrt_error_cb;
```

```
hal_usrt_irq_handle_set(&usrt0_info, &usrt_irq);
```


函数 hal_usrt_irq_handle_all_reset

函数hal_usrt_irq_handle_all_reset描述见下表：

表 3-558. 函数 hal_usrt_irq_handle_all_reset

函数名称	hal_usrt_irq_handle_all_reset
函数原型	void hal_usrt_irq_handle_all_reset(hal_usrt_dev_struct *usrt);
功能描述	复位所有的用户自定义的中断回调函数
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback */
```

```
hal_usrt_irq_handle_all_reset(&usrt0_info);
```

函数 hal_usrt_transmit_poll

函数hal_usrt_transmit_poll描述见下表：

表 3-559. 函数 hal_usrt_transmit_poll

函数名称	hal_usrt_transmit_poll
函数原型	int32_t hal_usrt_transmit_poll(hal_usrt_dev_struct *usrt, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	采用轮询方式发送数据，该函数为阻塞式
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY,

	HAL_ERR_TIMEOUT
--	-----------------

例如：

```
#define TRANSMIT_SIZE          6

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};

/* transmit using polling mode */

hal_usrt_transmit_poll(&usrt0_info, txbuffer, TRANSMIT_SIZE, 0x1FFFFF);
```

函数 hal_usrt_receive_poll

函数hal_usrt_receive_poll描述见下表：

表 3-560. 函数 hal_usrt_receive_poll

函数名称	hal_usrt_receive_poll
函数原型	int32_t hal_usrt_receive_poll(hal_usrt_dev_struct *usrt, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	采用轮询方式接收数据，该函数为阻塞式
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY, HAL_ERR_TIMEOUT

例如：

```
#define TRANSFER_SIZE          16

uint8_t rxbuffer[TRANSFER_SIZE];

/* receive using polling mode */

hal_usrt_receive_poll(&usrt1_info, rxbuffer, TRANSFER_SIZE, 0x7FFFFFFF);
```

函数 hal_usrt_transmit_receive_poll

函数hal_usrt_transmit_receive_poll描述见下表：

表 3-561. 函数 hal_usrt_transmit_receive_poll

函数名称	hal_usrt_transmit_receive_poll
函数原型	int32_t hal_usrt_transmit_receive_poll(hal_usrt_dev_struct *usrt, uint8_t *p_tx_buffer, uint8_t *p_rx_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	采用轮询方式发送接收数据，该函数为阻塞式
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输入参数{in}	
p_tx_buffer	指向发送数据缓冲区的指针
输入参数{in}	
p_rx_buffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间（设置为HAL_TIMEOUT_FOREVER意味着死等）
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSFER_SIZE                16

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B,
0x0C, 0x0D, 0x0E, 0x0F};

uint8_t rxbuffer[TRANSFER_SIZE];

/* transmit & receive using polling mode */

hal_usrt_transmit_receive_poll(&usrt1_info,txbuffer,rxbuffer,TRANSFER_SIZE, 0x1FFFF);
```

函数 hal_usrt_transmit_interrupt

函数hal_usrt_transmit_interrupt描述见下表：

表 3-562. 函数 hal_usrt_transmit_interrupt

函数名称	hal_usrt_transmit_interrupt
函数原型	int32_t hal_usrt_transmit_interrupt(hal_usrt_dev_struct *usrt, uint8_t *p_buffer, uint32_t length, hal_usrt_user_cb p_user_func);

功能描述	采用中断方式发送数据，在发送完成后调用用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 <i>hal_usrt_dev_struct</i>
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSMIT_SIZE          6

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};

__IO FlagStatus tx_end = RESET;

void tx_complete(hal_usrt_dev_struct *usrt){
    tx_end = SET;
}

/* transmit using interrupt mode */

hal_usrt_transmit_interrupt(&usrt0_info, txbuffer, TRANSMIT_SIZE, tx_complete);

/* check whether the transfer is completed or not */

while(RESET == tx_end){
}
```

函数 hal_usrt_receive_interrupt

函数hal_usrt_receive_interrupt描述见下表：

表 3-563. 函数 hal_usrt_receive_interrupt

函数名称	hal_usrt_receive_interrupt
函数原型	int32_t hal_usrt_receive_interrupt(hal_usrt_dev_struct *usrt, uint8_t *p_buffer, uint32_t length, hal_usrt_user_cb p_user_func);
功能描述	采用中断方式接收数据，在接收完成后调用用户回调函数，该函数为非阻塞式
先决条件	-

输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSFER_SIZE                16

uint8_t rxbuffer[TRANSFER_SIZE];

__IO FlagStatus rx_end = RESET;

void rx_complete(hal_usrt_dev_struct *usrt){
    rx_end = SET;
}

/* receive data using interrupt mode */

hal_usrt_receive_interrupt(&usrt0_info, rxbuffer, TRANSFER_SIZE, rx_complete);

/* check whether the transfer is completed or not */

while(RESET == rx_end){
}
```

函数 hal_usrt_transmit_receive_interrupt

函数hal_usrt_transmit_receive_interrupt描述见下表：

表 3-564. 函数 hal_usrt_transmit_receive_interrupt

函数名称	hal_usrt_transmit_receive_interrupt
函数原型	int32_t hal_usrt_transmit_receive_interrupt(hal_usrt_dev_struct *usrt, uint8_t *p_tx_buffer, uint8_t *p_rx_buffer, uint32_t length, hal_usrt_user_cb p_user_func);
功能描述	采用中断方式发送接收数据，在传输完成后调用用户回调函数，该函数为非阻塞式
先决条件	-

输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输入参数{in}	
p_tx_buffer	指向发送数据缓冲区的指针
输入参数{in}	
p_rx_buffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSFER_SIZE                16

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B,
0x0C, 0x0D, 0x0E, 0x0F};

uint8_t rxbuffer[TRANSFER_SIZE];

__IO FlagStatus transfer_end = RESET;

void transfer_complete(hal_usrt_dev_struct *usrt){

    transfer_end = SET;

}

/* transmit and receive data */

hal_usrt_transmit_receive_interrupt(&usrt0_info, txbuffer, rxbuffer, TRANSFER_SIZE,
transfer_complete);

while(RESET == transfer_end){

}
```

函数 hal_usrt_transmit_dma

函数hal_usrt_transmit_dma描述见下表：

表 3-565. 函数 hal_usrt_transmit_dma

函数名称	hal_usrt_transmit_dma
函数原型	int32_t hal_usrt_transmit_dma(hal_usrt_dev_struct *usrt, uint8_t *p_buffer, uint16_t length, hal_usrt_user_callback_struct *p_func);

功能描述	采用DMA方式发送数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_func	执行用户回调结构体的指针
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSMIT_SIZE          6

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};

__IO FlagStatus tx_end = RESET;

hal_usrt_user_callback_struct user_cb;

void tx_complete(hal_usrt_dev_struct *usrt){
    tx_end = SET;
}

/* initialize the user callback structure */

hal_usrt_struct_init(HAL_USRT_USER_CALLBACK_STRUCT, &user_cb);

user_cb.error_func = NULL;

user_cb.complete_func = tx_complete;

/* transmit using DMA mode */

hal_usrt_transmit_dma(&usrt0_info, txbuffer, TRANSMIT_SIZE, &user_cb);

/* check whether the transfer is completed or not */

while(RESET == tx_end){
}
```

函数 hal_usrt_receive_dma

函数hal_usrt_receive_dma描述见下表：

表 3-566. 函数 hal_usrt_receive_dma

函数名称	hal_usrt_receive_dma
函数原型	int32_t hal_usrt_receive_dma(hal_usrt_dev_struct *usrt, uint8_t *p_buffer, uint16_t length, hal_usrt_user_callback_struct *p_func);
功能描述	采用DMA方式接收数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_func	执行用户回调结构体的指针
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSFER_SIZE                16

uint8_t rxbuffer[TRANSFER_SIZE];

__IO FlagStatus rx_end = RESET;

hal_usrt_user_callback_struct user_cb;

void rx_complete(hal_usrt_dev_struct *usrt){

    rx_end = SET;

}

/* initilize the user callback structure */

hal_usrt_struct_init(HAL_USRT_USER_CALLBACK_STRUCT, &user_cb);

user_cb.error_func = NULL;

user_cb.complete_func = rx_complete;

/* receive data using interrupt mode */
```



```
hal_usrt_receive_dma(&usrt1_info, rxbuffer, TRANSFER_SIZE, &user_cb);

/* check whether the transfer is completed or not */

while(RESET == rx_end){

}
```

函数 hal_usrt_transmit_receive_dma

函数hal_usrt_transmit_receive_dma描述见下表：

表 3-567. 函数 hal_usrt_transmit_receive_dma

函数名称	hal_usrt_transmit_receive_dma
函数原型	int32_t hal_usrt_transmit_receive_dma(hal_usrt_dev_struct *usrt, uint8_t *p_tx_buffer, uint8_t *p_rx_buffer, uint16_t length, hal_usrt_user_callback_struct *p_func);
功能描述	采用DMA方式发送接收数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或错误会执行相应的用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输入参数{in}	
p_tx_buffer	指向发送数据缓冲区的指针
输入参数{in}	
p_rx_buffer	指向接收数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_func	执行用户回调结构体的指针
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSFER_SIZE                16

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B,
0x0C, 0x0D, 0x0E, 0x0F};

uint8_t rxbuffer[TRANSFER_SIZE];

__IO FlagStatus transfer_end = RESET;

hal_usrt_user_callback_struct user_cb;
```

```
void transfer_complete(hal_usrt_dev_struct *usrt){
    transfer_end = SET;
}

hal_usrt_struct_init(HAL_USRT_USER_CALLBACK_STRUCT, &user_cb);

user_cb.error_func = NULL;

user_cb.complete_func = transfer_complete;

/* transmit and receive data */

hal_usrt_transmit_receive_dma(&usrt0_info, txbuffer, rxbuffer, TRANSFER_SIZE, &user_cb);

while(RESET == transfer_end){
}
}
```

函数 hal_usrt_dma_pause

函数hal_usrt_dma_pause描述见下表:

表 3-568. 函数 hal_usrt_dma_pause

函数名称	hal_usrt_dma_pause
函数原型	int32_t hal_usrt_dma_pause(hal_usrt_dev_struct *usrt);
功能描述	传输过程中暂停USRT的DMA传输
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如:

```
/* pause USRT DMA transfer */

hal_usrt_dma_pause(&usrt0_info);
```

函数 hal_usrt_dma_resume

函数hal_usrt_dma_resume描述见下表:

表 3-569. 函数 hal_usrt_dma_resume

函数名称	hal_usrt_dma_resume
函数原型	int32_t hal_usrt_dma_resume(hal_usrt_dev_struct *usrt);
功能描述	传输过程中恢复USRT的DMA传输

先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* resume USRT DMA transfer */
```

```
hal_usrt_dma_resume(&usrt0_info);
```

函数 hal_usrt_transfer_stop

函数hal_usrt_transfer_stop描述见下表：

表 3-570. 函数 hal_usrt_transfer_stop

函数名称	hal_usrt_transfer_stop
函数原型	int32_t hal_usrt_transfer_stop(hal_usrt_dev_struct *usrt);
功能描述	停止USRT传输数据，该函数为阻塞式
先决条件	-
输入参数{in}	
usrt	指向设备信息结构体的指针，结构体成员参考 表 3-551. 结构体 hal_usrt_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* stop USART transmit and receive transfer data */
```

```
hal_usrt_transfer_stop(&usrt0_info);
```

3.26. IrDA

IrDA是一种异步半双工通信协议模式，其为USART外设的一种通信模式，章节[3.26.1](#)描述了USART的寄存器列表，章节[3.26.2](#)对IrDA库函数进行说明。

3.26.1. 外设寄存器说明

USART寄存器列表如下表所示：

表 3-571. USART 寄存器

寄存器名称	寄存器描述
USART_CTL0	控制寄存器0
USART_CTL1	控制寄存器1
USART_CTL2	控制寄存器2
USART_BAUD	波特率寄存器
USART_GP	保护时间和预分频器寄存器
USART_RT	接收超时寄存器
USART_CMD	请求寄存器
USART_STAT	状态寄存器
USART_INTC	中断标志清除寄存器
USART_RDATA	数据接收寄存器
USART_TDATA	数据发送寄存器
USART_RFCS	接收FIFO控制和状态寄存器

3.26.2. 外设库函数说明

IrDA库函数列表如下表所示：

表 3-572. IrDA 库函数

库函数名称	库函数描述
hal_irda_struct_init	根据已有的hal_irda_struct_type_enum类型，初始化IrDA结构体，当定义了一个结构体后需要调用该函数将该结构体成员赋为初始值
hal_irda_deinit	重新初始化IrDA设备信息结构体
hal_irda_init	根据初始化结构体及USART外设信息，对IrDA设备信息结构体进行初始化
hal_irda_irq	处理所有IrDA中断请求
hal_irda_irq_handle_set	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
hal_irda_irq_handle_all_reset	复位所有的用户自定义的中断回调函数
hal_irda_transmit_poll	采用轮询方式发送数据，该函数为阻塞式
hal_irda_receive_poll	采用轮询方式接收数据，该函数为阻塞式
hal_irda_transmit_interrupt	采用中断方式发送数据，在发送完成后调用用户回调函数，该函数为非阻塞式
hal_irda_receive_interrupt	采用中断方式接收数据，在接收完成后调用用户回调函数，该函数为非阻塞式
hal_irda_transmit_dma	采用DMA方式发送数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
hal_irda_receive_dma	采用DMA方式接收数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式

库函数名称	库函数描述
hal_irda_dma_pause	传输过程中暂停IrDA的DMA传输
hal_irda_dma_resume	传输过程中恢复IrDA的DMA传输
hal_irda_transmit_stop	停止IrDA发送数据，该函数为阻塞式
hal_irda_receive_stop	停止IrDA接收数据，该函数为阻塞式

枚举 hal_irda_struct_type_enum

表 3-573. 枚举 hal_irda_struct_type_enum

成员名称	功能描述
HAL_IRDA_INIT_STRUCT	IRDA初始化结构体
HAL_IRDA_DEV_STRUCT	IRDA中断回调函数指针结构体
HAL_IRDA_USER_CALLBACK_STR UCT	IRDA DMA回调函数指针结构体

结构体 hal_irda_init_struct

表 3-574. 结构体 hal_irda_init_struct

成员名称	功能描述
baudrate	波特率
parity	奇偶校验
word_length	一帧数据的长度
direction	传输方向
rx_fifo_en	接收FIFO使能
mode	低功耗模式
prescaler	预分频器

结构体 hal_irda_irq_struct

表 3-575. 结构体 hal_irda_irq_struct

成员名称	功能描述
receive_complete_handle	接收完成回调函数
transmit_ready_handle	发送就绪回调函数
transmit_complete_handle	发送完成回调函数
error_handle	错误回调函数

枚举 hal_irda_run_state_enum

表 3-576. 枚举 hal_irda_run_state_enum

成员名称	功能描述
IRDA_STATE_FREE	IRDA空闲
IRDA_STATE_BUSY	IRDA繁忙

结构体 hal_irda_dev_struct

表 3-577. 结构体 hal_irda_dev_struct

成员名称	功能描述
periph	USART外设
irda_irq	中断回调函数指针，形参成员参考 表 3-575. 结构体hal_irda_irq_struct
p_dma_rx	DMA接收指针，指向DMA设备信息结构体
p_dma_tx	DMA发送指针，指向DMA设备信息结构体
txbuffer	发送缓冲区
rxbuffer	接收缓冲区
data_bit_mask	数据屏蔽位
last_error	上一个错误码
error_state	错误码
tx_state	发送状态
rx_state	接收状态
rx_callback	接收回调指针
tx_callback	发送回调指针
mutex	互斥锁和解锁状态
priv	私有指针

结构体 hal_irda_user_callback_struct

表 3-578. 结构体 hal_irda_user_callback_struct

成员名称	功能描述
complete_func	传输完成回调
error_func	错误回调

函数 hal_irda_struct_init

函数hal_irda_struct_init描述见下表：

表 3-579. 函数 hal_irda_struct_init

函数名称	hal_irda_struct_init
函数原型	void hal_irda_struct_init(hal_irda_struct_type_enum hal_struct_type, void *p_struct);
功能描述	根据已有的hal_irda_struct_type_enum类型，初始化IrDA结构体，当定义了一个结构体后需要调用该函数将该结构体成员赋为初始值
先决条件	-
输入参数{in}	
hal_struct_type	结构体类型，形参成员参考 表 3-574. 结构体hal_irda_init_struct
输入参数{in}	
p_struct	指向结构体的空指针
输出参数{out}	
-	-

返回值	
-	-

例如：

```
hal_irda_dev_struct irda0_info;
```

```
/* initialize a irda device information structure */
```

```
hal_irda_struct_init(HAL_IRDA_DEV_STRUCT, &irda0_info);
```

函数 hal_irda_deinit

函数hal_irda_deinit描述见下表：

表 3-580. 函数 hal_irda_deinit

函数名称	hal_irda_deinit
函数原型	void hal_irda_deinit(hal_irda_dev_struct *irda);
功能描述	重新初始化IrDA设备信息结构体
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 hal_irda_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_irda_dev_struct irda0_info;
```

```
/* deinitialize the device information structure */
```

```
hal_irda_deinit(&irda0_info);
```

函数 hal_irda_init

函数hal_irda_init描述见下表：

表 3-581. 函数 hal_irda_init

函数名称	hal_irda_init
函数原型	int32_t hal_irda_init(hal_irda_dev_struct *irda, uint32_t periph, hal_irda_init_struct *p_init);
功能描述	根据初始化结构体及USART外设信息，对IrDA设备信息结构体进行初始化
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 hal_irda_dev_struct

输入参数{in}	
periph	USART外设
USARTx	x=0
输入参数{in}	
p_init	指向初始化结构体的指针，结构体成员参考 表 3-574. 结构体 hal_irda_init_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如：

```

hal_irda_dev_struct irda0_info;

hal_irda_init_struct irda0_init;

/* initialize the structures */

hal_irda_struct_init(HAL_IRDA_DEV_STRUCT, &irda0_info);
hal_irda_struct_init(HAL_IRDA_INIT_STRUCT, &irda0_init);

irda0_init.baudrate = 115200;

irda0_init.parity = IRDA_PARITY_NONE;

irda0_init.word_length = IRDA_WORD_LENGTH_8BIT;

irda0_init.direction = IRDA_DIRECTION_RX_TX;

irda0_init.rx_fifo_en = DISABLE;

irda0_init.mode = IRDA_NORMAL_MODE;

irda0_init.prescaler = 1;

hal_irda_init(&irda0_info, USART0, &irda0_init);

```

函数 hal_irda_irq

函数hal_irda_irq描述见下表：

表 3-582. 函数 hal_irda_irq

函数名称	hal_irda_irq
函数原型	void hal_irda_irq(hal_irda_dev_struct *irda);
功能描述	处理所有IrDA中断请求
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 hal_irda_dev_struct

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* process the IrDA interrupt */
hal_irda_irq(&irda0_info);
```

函数 hal_irda_irq_handle_set

函数hal_irda_irq_handle_set描述见下表:

表 3-583. 函数 hal_irda_irq_handle_set

函数名称	hal_irda_irq_handle_set
函数原型	void hal_irda_irq_handle_set(hal_irda_dev_struct *irda, hal_irda_irq_struct *p_irq);
功能描述	设置用户自定义的中断回调函数，当相应的中断被触发时，对应的回调函数将会被执行
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 hal_irda_dev_struct
输入参数{in}	
p_irq	指向中断结构体的指针，结构体成员参考 表 3-575. 结构体 hal_irda_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
hal_irda_irq_struct irda_irq;
/* set the error handle function */
irda_irq.error_handle = irda_error_cb;
hal_irda_irq_handle_set(&irda0_info, &irda_irq);
```

函数 hal_irda_irq_handle_all_reset

函数hal_irda_irq_handle_all_reset描述见下表:

表 3-584. 函数 hal_irda_irq_handle_all_reset

函数名称	hal_irda_irq_handle_all_reset
函数原型	void hal_irda_irq_handle_all_reset(hal_irda_dev_struct *irda);

功能描述	复位所有的用户自定义的中断回调函数
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 <i>hal_irda_dev_struct</i>
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset all user-defined interrupt callback */
```

```
hal_irda_irq_handle_all_reset(&irda0_info);
```

函数 hal_irda_transmit_poll

函数hal_irda_transmit_poll描述见下表：

表 3-585. 函数 hal_irda_transmit_poll

函数名称	hal_irda_transmit_poll
函数原型	int32_t hal_irda_transmit_poll(hal_irda_dev_struct *irda, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	采用轮询方式发送数据，该函数为阻塞式
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 <i>hal_irda_dev_struct</i>
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间（设置为HAL_TIMEOUT_FOREVER意味着死等）
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY, HAL_ERR_TIMEOUT

例如：

```
#define TRANSMIT_SIZE
```

```
6
```

```
uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};
```

```
/* transmit using polling mode */
```

```
hal_irda_transmit_poll(&irda0_info, txbuffer, TRANSMIT_SIZE, 0x1FFFFF);
```

函数 hal_irda_receive_poll

函数hal_irda_receive_poll描述见下表：

表 3-586. 函数 hal_irda_receive_poll

函数名称	hal_irda_receive_poll
函数原型	int32_t hal_irda_receive_poll(hal_irda_dev_struct *irda, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	采用轮询方式接收数据，该函数为阻塞式
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 hal_irda_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY, HAL_ERR_TIMEOUT

例如：

```
#define TRANSFER_SIZE          16

uint8_t rxbuffer[TRANSFER_SIZE];

/* receive using polling mode */

hal_irda_receive_poll(&irda0_info, rxbuffer, TRANSFER_SIZE, 0x7FFFFFFF);
```

函数 hal_irda_transmit_interrupt

函数hal_irda_transmit_interrupt描述见下表：

表 3-587. 函数 hal_irda_transmit_interrupt

函数名称	hal_irda_transmit_interrupt
函数原型	int32_t hal_irda_transmit_interrupt(hal_irda_dev_struct *irda, uint8_t *p_buffer, uint32_t length, hal_irda_user_cb p_user_func);
功能描述	采用中断方式发送数据，在发送完成后调用用户回调函数，该函数为非阻塞式

先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 <i>hal_irda_dev_struct</i>
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSMIT_SIZE          6

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};

__IO FlagStatus tx_end = RESET;

void tx_complete(hal_irda_dev_struct *irda){
    tx_end = SET;
}

/* transmit using interrupt mode */

hal_irda_transmit_interrupt(&irda0_info, txbuffer, TRANSMIT_SIZE, tx_complete);

/* check whether the transfer is completed or not */

while(RESET == tx_end){
}
```

函数 hal_irda_receive_interrupt

函数hal_irda_receive_interrupt描述见下表：

表 3-588. 函数 hal_irda_receive_interrupt

函数名称	hal_irda_receive_interrupt
函数原型	int32_t hal_irda_receive_interrupt(hal_irda_dev_struct *irda, uint8_t *p_buffer, uint32_t length, hal_irda_user_cb p_user_func);
功能描述	采用中断方式接收数据，在接收完成后调用用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	

irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 <i>hal_irda_dev_struct</i>
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSFER_SIZE                16

uint8_t rxbuffer[TRANSFER_SIZE];

__IO FlagStatus rx_end = RESET;

void rx_complete(hal_irda_dev_struct * irda){
    rx_end = SET;
}

/* receive data using interrupt mode */

hal_irda_receive_interrupt(&irda0_info, rxbuffer, TRANSFER_SIZE, rx_complete);

/* check whether the transfer is completed or not */

while(RESET == rx_end){
}
```

函数 hal_irda_transmit_dma

函数hal_irda_transmit_dma描述见下表：

表 3-589. 函数 hal_irda_transmit_dma

函数名称	hal_irda_transmit_dma
函数原型	int32_t hal_irda_transmit_dma(hal_irda_dev_struct *irda, uint8_t *p_buffer, uint16_t length, hal_irda_user_callback_struct *p_func);
功能描述	采用DMA方式发送数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体

	<u>hal_irda_dev_struct</u>
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_func	执行用户回调结构体的指针
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSMIT_SIZE          6

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};

__IO FlagStatus tx_end = RESET;

hal_irda_user_callback_struct user_cb;

void tx_complete(hal_irda_dev_struct *irda){
    tx_end = SET;
}

/* initialize the user callback structure */
hal_irda_struct_init(HAL_IRDA_USER_CALLBACK_STRUCT, &user_cb);

user_cb.error_func = NULL;

user_cb.complete_func = tx_complete;

/* transmit using DMA mode */
hal_irda_transmit_dma(&irda0_info, txbuffer, TRANSMIT_SIZE, &user_cb);

/* check whether the transfer is completed or not */
while(RESET == tx_end){
}
```

函数 hal_irda_receive_dma

函数hal_irda_receive_dma描述见下表：

表 3-590. 函数 hal_irda_receive_dma

函数名称	hal_irda_receive_dma
函数原型	int32_t hal_irda_receive_dma(hal_irda_dev_struct *irda, uint8_t *p_buffer,

	uint16_t length, hal_irda_user_callback_struct *p_func);
功能描述	采用DMA方式接收数据，用户可指定自定义的传输完成和错误回调函数，当数据传输完成或出现错误会执行相应的用户回调函数，该函数为非阻塞式
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 hal_irda_dev_struct
输入参数{in}	
p_buffer	指向数据缓冲区的指针
输入参数{in}	
length	待传输的数据长度
输入参数{in}	
p_func	执行用户回调结构体的指针
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSFER_SIZE                16

uint8_t rxbuffer[TRANSFER_SIZE];

__IO FlagStatus rx_end = RESET;

hal_irda_user_callback_struct user_cb;

void rx_complete(hal_irda_dev_struct *irda){
    rx_end = SET;
}

/* initilize the user callback structure */
hal_irda_struct_init(HAL_IRDA_USER_CALLBACK_STRUCT, &user_cb);

user_cb.error_func = NULL;

user_cb.complete_func = rx_complete;

/* receive data using interrupt mode */
hal_irda_receive_dma(&irda0_info, rxbuffer, TRANSFER_SIZE, &user_cb);

/* check whether the transfer is completed or not */
while(RESET == rx_end){
}
```

函数 hal_irda_dma_pause

函数hal_irda_dma_pause描述见下表：

表 3-591. 函数 hal_irda_dma_pause

函数名称	hal_irda_dma_pause
函数原型	int32_t hal_irda_dma_pause(hal_irda_dev_struct *irda);
功能描述	传输过程中暂停IrDA的DMA传输
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 hal_irda_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* pause IrDA DMA transfer */
hal_irda_dma_pause(&irda0_info);
```

函数 hal_irda_dma_resume

函数hal_irda_dma_resume描述见下表：

表 3-592. 函数 hal_irda_dma_resume

函数名称	hal_irda_dma_resume
函数原型	int32_t hal_irda_dma_resume(hal_irda_dev_struct *irda);
功能描述	传输过程中恢复IrDA的DMA传输
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 hal_irda_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* resume IrDA DMA transfer */
hal_irda_dma_resume(&irda0_info);
```


函数 hal_irda_transmit_stop

函数hal_irda_transmit_stop描述见下表：

表 3-593. 函数 hal_irda_transmit_stop

函数名称	hal_irda_transmit_stop
函数原型	int32_t hal_irda_transmit_stop(hal_irda_dev_struct *irda);
功能描述	停止IrDA发送数据，该函数为阻塞式
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 hal_irda_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* stop USART transmit transfer */
hal_irda_transmit_stop(&irda0_info);
```

函数 hal_irda_receive_stop

函数hal_irda_receive_stop描述见下表：

表 3-594. 函数 hal_irda_receive_stop

函数名称	hal_irda_receive_stop
函数原型	int32_t hal_irda_receive_stop(hal_irda_dev_struct *irda);
功能描述	停止IrDA接收数据，该函数为阻塞式
先决条件	-
输入参数{in}	
irda	指向设备信息结构体的指针，结构体成员参考 表 3-577. 结构体 hal_irda_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
/* stop USART receive transfer */
hal_irda_receive_stop(&irda0_info);
```

3.27. SMARTCARD

SMARTCARD是一种异步半双工通信协议模式，其为USART外设的一种通信模式，主要用于智能卡通信，章节[3.27.1](#)描述了USART的寄存器列表，章节[3.27.2](#)对SMARTCARD库函数进行说明。

3.27.1. 外设寄存器说明

USART寄存器列表如下表所示：

表 3-595. USART 寄存器

寄存器名称	寄存器描述
USART_CTL0	控制寄存器0
USART_CTL1	控制寄存器1
USART_CTL2	控制寄存器2
USART_BAUD	波特率寄存器
USART_GP	保护时间和预分频器寄存器
USART_RT	接收超时寄存器
USART_CMD	请求寄存器
USART_STAT	状态寄存器
USART_INTC	中断标志清除寄存器
USART_RDATA	数据接收寄存器
USART_TDATA	数据发送寄存器
USART_RFCS	接收FIFO控制和状态寄存器

3.27.2. 外设库函数说明

SMARTCARD库函数列表如下表所示：

表 3-596. SMARTCARD 库函数

库函数名称	库函数描述
hal_smartcard_struct_init	初始化SMARTCARD相关结构体
hal_smartcard_deinit	SMARTCARD去初始化
hal_smartcard_init	SMARTCARD初始化
hal_smartcard_irq	SMARTCARD中断处理函数
hal_smartcard_irq_handle_set	用户自定义中断回调函数
hal_smartcard_irq_handle_all_reset	复位全部中断回调函数
hal_smartcard_transmit_poll	轮询方式发送数据
hal_smartcard_receive_poll	轮询方式接收数据
hal_smartcard_transmit_interrupt	中断方式发送数据
hal_smartcard_receive_interrupt	中断方式接收数据
hal_smartcard_transmit_dma	DMA方式发送数据
hal_smartcard_receive_dma	DMA方式接收数据

库函数名称	库函数描述
hal_smartcard_transmit_stop	停止SMARTCARD发送
hal_smartcard_receive_stop	停止SMARTCARD接收

枚举 hal_smartcard_struct_type_enum

表 3-597. 枚举 hal_smartcard_struct_type_enum

成员名称	功能描述
HAL_SMARTCARD_INIT_STRUCT	SMARTCARD初始化结构体
HAL_SMARTCARD_DEV_STRUCT	SMARTCARD中断回调函数指针结构体
HAL_SMARTCARD_USER_CALLBACK_STRUCT	SMARTCARD DMA回调函数指针结构体
HAL_SMARTCARD_IRQ_INIT_STRUCTURE	SMARTCARD设备信息结构体

结构体 hal_smartcard_init_struct

表 3-598. 结构体 hal_smartcard_init_struct

成员名称	功能描述
baudrate	通信波特率
parity	校验模式
word_length	数据位长度
stop_bit	停止位个数
direction	传输方向
clock_polarity	时钟极性
clock_phase	时钟相位
clock_length_lastbit	最后一个数据位是否伴随有时钟脉冲输出
prescaler	预分频值
guard_time	保护时间
nack_state	NACK状态
early_nack	提前NACK
timeout_enable	接收超时使能
timeout_value	接收超时时间
sample_method	采样方式
block_length	块长
auto_retry_count	自动重发次数
first_bit_msb	MSB先发送
tx_rx_swap	Tx引脚和Rx引脚交换
rx_level_invert	反转Rx引脚有效电平
tx_level_invert	反转Tx引脚有效电平
data_bit_invert	数据反转
overrun_disable	禁能过载
rx_error_dma_stop	接收错误时禁能DMA

结构体 hal_smartcard_irq_struct

表 3-599. 结构体 hal_smartcard_irq_struct

成员名称	功能描述
receive_complete_handle	接收完成回调函数
transmit_ready_handle	发送就绪回调函数
transmit_complete_handle	发送完成回调函数
error_handle	传输错误回调函数

枚举 hal_smartcard_run_state_enum

表 3-600. 枚举 hal_smartcard_run_state_enum

成员名称	功能描述
SMARTCARD_STATE_FREE	SMARTCARD空闲
SMARTCARD_STATE_BUSY	SMARTCARD繁忙

结构体 hal_smartcard_dev_struct

表 3-601. 结构体 hal_smartcard_dev_struct

成员名称	功能描述
periph	USARTx外设(USART0)
smartcard_irq	中断回调函数
p_dma_rx	DMA接收指针
p_dma_tx	DMA发送指针
txbuffer	发送Buffer
rxbuffer	接收Buffer
last_error	最近一次错误状态
error_state	当下错误状态
tx_state	发送状态
rx_state	接收状态
rx_callback	接收回调函数指针
tx_callback	发送回调函数指针
mutex	互斥锁和解锁状态
priv	私有函数指针

结构体 hal_smartcard_user_callback_struct

表 3-602. 结构体 hal_smartcard_user_callback_struct

成员名称	功能描述
complete_func	用户自定义传输完成中断回调函数
error_func	用户自定义传输错误中断回调函数

函数 hal_smartcard_struct_init

函数hal_smartcard_struct_init描述见下表：

表 3-603. 函数 hal_smartcard_struct_init

函数名称	hal_smartcard_struct_init
函数原型	void hal_smartcard_struct_init(hal_smartcard_struct_type_enum hal_struct_type, void *p_struct);
功能描述	初始化SMARTCARD相关结构体
先决条件	-
输入参数{in}	
hal_struct_type	SMARTCARD结构体类型，形参成员参考 表3-597. 枚举 hal_smartcard_struct_type_enum
输入参数{in}	
p_struct	初始化结构体指针
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_smartcard_dev_struct smartcard0_info;
```

```
/* initialize SMARTCARD device information structure with the default values */
```

```
hal_smartcard_struct_init(HAL_SMARTCARD_DEV_STRUCT, &smartcard0_info);
```

函数 hal_smartcard_deinit

函数hal_smartcard_deinit描述见下表：

表 3-604. 函数 hal_smartcard_deinit

函数名称	hal_smartcard_deinit
函数原型	void hal_smartcard_deinit(hal_smartcard_dev_struct *smartcard);
功能描述	SMARTCARD去初始化
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_smartcard_dev_struct smartcard0_info;
```

```
/* deinitialize SMARTCARD*/
```

```
hal_smartcard_deinit(&smartcard0_info);
```

函数 hal_smartcard_init

函数hal_smartcard_init描述见下表:

表 3-605. 函数 hal_smartcard_init

函数名称	hal_smartcard_init
函数原型	int32_t hal_smartcard_init(hal_smartcard_dev_struct *smartcard, uint32_t periph, hal_smartcard_init_struct *p_init);
功能描述	SMARTCARD初始化
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体, 结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输入参数{in}	
periph	外设USARTx
USARTx	x=0
输入参数{in}	
p_init	初始化结构体, 结构体成员参考 表 3-598. 结构体 hal_smartcard_init_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如:

```
hal_smartcard_dev_struct smartcard0_info;
```

```
hal_smartcard_init_struct smartcard0_init;
```

```
/* initialize SMARTCARD device information structure with the default values */
```

```
hal_smartcard_struct_init(HAL_SMARTCARD_INIT_STRUCT, &smartcard0_init);
```

```
smartcard0_init.baudrate = 10752;
```

```
smartcard0_init.parity = SMARTCARD_PARITY_EVEN;
```

```
smartcard0_init.word_length = SMARTCARD_WORD_LENGTH_9BIT;
```

```
smartcard0_init.stop_bit = SMARTCARD_STOP_BIT_1_5;
```

```
smartcard0_init.direction = SMARTCARD_DIRECTION_RX_TX;
```

```
smartcard0_init.clock_polarity = SMARTCARD_CLOCK_POLARITY_LOW;
```

```
smartcard0_init.clock_phase = SMARTCARD_CLOCK_PHASE_1CK;
```

```

smartcard0_init.clock_length_lastbit = SMARTCARD_LAST_BIT_NOT_OUTPUT;

smartcard0_init.prescaler = 9;

smartcard0_init.guard_time = 4;

smartcard0_init.nack_state = SMARTCARD_NACK_DISABLE;

smartcard0_init.early_nack = DISABLE;

smartcard0_init.rx_fifo_en = DISABLE;

smartcard0_init.timeout_enable = DISABLE;

smartcard0_init.timeout_value = 0;

smartcard0_init.sample_method = SMARTCARD_THREE_SAMPLE_BIT;

smartcard0_init.block_length = 0;

smartcard0_init.auto_retry_count = 0;

smartcard0_init.first_bit_msb = DISABLE;

smartcard0_init.rx_rx_swap = DISABLE;

smartcard0_init.tx_level_invert = DISABLE;

smartcard0_init.rx_level_invert = DISABLE;

smartcard0_init.data_bit_invert = DISABLE;

smartcard0_init.overrun_disable = DISABLE;

smartcard0_init.rx_error_dma_stop = DISABLE;

/* initialize SMARTCARD*/

hal_smartcard_init(&smartcard0_info, USART0, &smartcard0_init);

```

函数 **hal_smartcard_irq**

函数hal_smartcard_irq描述见下表：

表 3-606. 函数 hal_smartcard_irq

函数名称	hal_smartcard_irq
函数原型	void hal_smartcard_irq(hal_smartcard_dev_struct *smartcard);
功能描述	SMARTCARD中断处理函数具体内容
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输出参数{out}	
-	-

返回值	
-	-

例如：

```
hal_smartcard_dev_struct smartcard0_info;

/* SMARTCARD interrupt handler content function */

hal_smartcard_irq(&smartcard0_info);
```

函数 hal_smartcard_irq_handle_set

函数hal_smartcard_irq_handle_set描述见下表：

表 3-607. 函数 hal_smartcard_irq_handle_set

函数名称	hal_smartcard_irq_handle_set
函数原型	void hal_smartcard_irq_handle_set(hal_smartcard_dev_struct *smartcard, hal_smartcard_irq_struct *p_irq);
功能描述	用户自定义中断回调函数
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输入参数{in}	
p_irq	中断回调函数结构体，结构体成员参考 表 3-599. 结构体 hal_smartcard_irq_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_smartcard_dev_struct smartcard0_info;

void smartcard_error_handler(hal_smartcard_dev_struct *smartcard);

hal_smartcard_irq_struct smartcard0_irq;

/* initialize interrupt callback structure with the default values */

hal_smartcard_struct_init(HAL_SMARTCARD_IRQ_INIT_STRUCT, &smartcard0_irq);

smartcard0_irq.error_handle = (hal_irq_handle_cb)smartcard_error_handler;

/* set user-defined interrupt callback function */

hal_smartcard_irq_handle_set(&smartcard0_info, &smartcard0_irq);
```


函数 hal_smartcard_irq_handle_all_reset

函数hal_smartcard_irq_handle_all_reset描述见下表：

表 3-608. 函数 hal_smartcard_irq_handle_all_reset

函数名称	hal_smartcard_irq_handle_all_reset
函数原型	void hal_smartcard_irq_handle_all_reset(hal_smartcard_dev_struct *smartcard);
功能描述	复位全部中断回调函数
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
hal_smartcard_dev_struct smartcard0_info;

/* reset all user-defined interrupt callback function */

hal_smartcard_irq_handle_all_reset(&smartcard0_info)
```

函数 hal_smartcard_transmit_poll

函数hal_smartcard_transmit_poll描述见下表：

表 3-609. 函数 hal_smartcard_transmit_poll

函数名称	hal_smartcard_transmit_poll
函数原型	int32_t hal_smartcard_transmit_poll(hal_smartcard_dev_struct *smartcard, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	轮询方式发送数据
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输入参数{in}	
p_buffer	发送数据存储区
输入参数{in}	
length	发送数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	

-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY, HAL_ERR_TIMEOUT

例如：

```
#define TRANSMIT_SIZE          6

uint8_t txdata[] = {0x01,0x02,0x03,0x04,0x05};

/* transmit data by poll method */

hal_smartcard_transmit_poll(&smartcard0_info, txdata, TRANSMIT_SIZE, 0x1FFF);
```

函数 hal_smartcard_receive_poll

函数hal_smartcard_receive_poll描述见下表：

表 3-610. 函数 hal_smartcard_receive_poll

函数名称	hal_smartcard_receive_poll
函数原型	int32_t hal_smartcard_receive_poll(hal_smartcard_dev_struct *smartcard, uint8_t *p_buffer, uint32_t length, uint32_t timeout_ms);
功能描述	轮询方式接收数据
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输入参数{in}	
p_buffer	接收数据存储区
输入参数{in}	
length	接收数据长度
输入参数{in}	
timeout_ms	超时时间
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY, HAL_ERR_TIMEOUT

例如：

```
#define TRANSFER_SIZE          16

uint8_t rxbuffer[TRANSFER_SIZE];

/* receive using polling mode */

hal_smartcard_receive_poll(&smartcard0_info, rxbuffer, TRANSFER_SIZE, 0x7FFFFFFF);
```

函数 hal_smartcard_transmit_interrupt

函数hal_smartcard_transmit_interrupt描述见下表：

表 3-611. 函数 hal_smartcard_transmit_interrupt

函数名称	hal_smartcard_transmit_interrupt
函数原型	int32_t hal_smartcard_transmit_interrupt(hal_smartcard_dev_struct *smartcard, uint8_t *p_buffer, uint32_t length, hal_smartcard_user_cb p_user_func);
功能描述	中断方式发送数据
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输入参数{in}	
p_buffer	发送数据存储区
输入参数{in}	
length	发送数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSMIT_SIZE          6

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};

__IO FlagStatus tx_end = RESET;

void tx_complete(hal_smartcard_dev_struct * smartcard){
    tx_end = SET;
}

/* transmit using interrupt mode */

hal_smartcard_transmit_interrupt(&smartcard0_info,    txbuffer,    TRANSMIT_SIZE,
tx_complete);

/* check whether the transfer is completed or not */

while(RESET == tx_end){
}
```

函数 hal_smartcard_receive_interrupt

函数hal_smartcard_receive_interrupt描述见下表：

表 3-612. 函数 hal_smartcard_receive_interrupt

函数名称	hal_smartcard_receive_interrupt
函数原型	int32_t hal_smartcard_receive_interrupt(hal_smartcard_dev_struct *smartcard, uint8_t *p_buffer, uint32_t length, hal_smartcard_user_cb p_user_func);
功能描述	中断方式接收数据
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输入参数{in}	
p_buffer	接收数据存储区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调函数
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSFER_SIZE                16

uint8_t rxbuffer[TRANSFER_SIZE];

__IO FlagStatus rx_end = RESET;

void rx_complete(hal_smartcard_dev_struct *smartcard){
    rx_end = SET;
}

/* receive data using interrupt mode */

hal_smartcard_receive_interrupt(&smartcard0_info,    rxbuffer,    TRANSFER_SIZE,
rx_complete);

/* check whether the transfer is completed or not */

while(RESET == rx_end){
}
```

函数 hal_smartcard_transmit_dma

函数hal_smartcard_transmit_dma描述见下表：

表 3-613. 函数 hal_smartcard_transmit_dma

函数名称	hal_smartcard_transmit_dma
函数原型	int32_t hal_smartcard_transmit_dma(hal_smartcard_dev_struct *smartcard, uint8_t *p_buffer, uint16_t length, hal_smartcard_user_callback_struct *p_user_func);
功能描述	DMA方式发送数据
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输入参数{in}	
p_buffer	发送数据存储区
输入参数{in}	
length	发送数据长度
输入参数{in}	
p_user_func	用户回调结构体，结构体成员参考 表 3-602. 结构体 hal_smartcard_user_callback_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```
#define TRANSMIT_SIZE          6

uint8_t txbuffer[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};

__IO FlagStatus tx_end = RESET;

hal_smartcard_user_callback_struct user_cb;

void tx_complete(hal_smartcard_dev_struct *smartcard){
    tx_end = SET;
}

/* initialize the user callback structure */

hal_smartcard_struct_init(HAL_SMARTCARD_USER_CALLBACK_STRUCT, &user_cb);

user_cb.error_func = NULL;

user_cb.complete_func = tx_complete;
```

```

/* transmit using DMA mode */

hal_smartcard_transmit_dma(&smartcard0_info, txbuffer, TRANSMIT_SIZE, &user_cb);

/* check whether the transfer is completed or not */

while(RESET == tx_end){

}

```

函数 hal_smartcard_receive_dma

函数hal_smartcard_receive_dma描述见下表：

表 3-614. 函数 hal_smartcard_receive_dma

函数名称	hal_smartcard_receive_dma
函数原型	int32_t hal_smartcard_receive_dma(hal_smartcard_dev_struct *smartcard, uint8_t *p_buffer, uint16_t length, hal_smartcard_user_callback_struct *p_user_func);
功能描述	DMA方式接收数据
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输入参数{in}	
p_buffer	接收数据存储区
输入参数{in}	
length	接收数据长度
输入参数{in}	
p_user_func	用户回调结构体，结构体成员参考 表 3-602. 结构体 hal_smartcard_user_callback_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL , HAL_ERR_BUSY

例如：

```

#define TRANSFER_SIZE                16

uint8_t rxbuffer[TRANSFER_SIZE];

__IO FlagStatus rx_end = RESET;

hal_smartcard_user_callback_struct user_cb;

void rx_complete(hal_smartcard_dev_struct *smartcard){

    rx_end = SET;
}

```

```

}

/* initialize the user callback structure */

hal_smartcard_struct_init(HAL_SMARTCARD_USER_CALLBACK_STRUCT, &user_cb);

user_cb.error_func = NULL;

user_cb.complete_func = rx_complete;

/* receive data using interrupt mode */

hal_smartcard_receive_dma(&smartcard0_info, rxbuffer, TRANSFER_SIZE, &user_cb);

/* check whether the transfer is completed or not */

while(RESET == rx_end){

}

```

函数 hal_smartcard_transmit_stop

函数hal_smartcard_transmit_stop描述见下表：

表 3-615. 函数 hal_smartcard_transmit_stop

函数名称	hal_smartcard_transmit_stop
函数原型	int32_t hal_smartcard_transmit_stop(hal_smartcard_dev_struct *smartcard);
功能描述	停止SMARTCARD发送
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```

hal_smartcard_dev_struct smartcard0_info;

/* stop SMARTCARD transmit */

hal_smartcard_transmit_stop(&smartcard0_info);

```

函数 hal_smartcard_receive_stop

函数hal_smartcard_receive_stop描述见下表：

表 3-616. 函数 hal_smartcard_receive_stop

函数名称	hal_smartcard_receive_stop
------	----------------------------

函数原型	int32_t hal_smartcard_receive_stop(hal_smartcard_dev_struct *smartcard);
功能描述	停止SMARTCARD接收
先决条件	-
输入参数{in}	
smartcard	SMARTCARD设备信息结构体，结构体成员参考 表 3-601. 结构体 hal_smartcard_dev_struct
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS

例如：

```
hal_smartcard_dev_struct smartcard0_info;

/* stop SMARTCARD receive */

hal_smartcard_receive_stop(&smartcard0_info);
```

3.28. WWDGT

窗口看门狗定时器(WWDGT)用来监测由软件故障导致的系统故障。章节[3.28.1](#)描述了WWDGT的寄存器列表，章节[3.28.2](#)对WWDGT库函数进行说明。

3.28.1. 外设寄存器说明

WWDGT寄存器列表如下表所示：

表 3-617. WWDGT 寄存器

寄存器名称	寄存器描述
WWDGT_CTL	控制寄存器
WWDGT_CFG	配置寄存器
WWDGT_STAT	状态寄存器

3.28.2. 外设库函数说明

WWDGT HAL库函数列表如下表所示：

表 3-618. WWDGT HAL 库函数

库函数名称	库函数描述
hal_wwdgt_struct_init	初始化WWDGT结构体
hal_wwdgt_init	初始化WWDGT

库函数名称	库函数描述
hal_wwdgt_deinit	复位初始化WWDGT
hal_wwdgt_irq	WWDGT中断处理函数
hal_wwdgt_irq_handle_set	WWDGT用户回调函数
hal_wwdgt_irq_handle_all_reset	清除WWDGT用户回调函数

枚举 hal_wwdgt_state_enum

表 3-619. hal_wwdgt_state_enum

枚举名称	枚举描述
HAL_WWDGT_STATE_NONE	无（默认值）
HAL_WWDGT_STATE_RESET	窗口看门狗状态重置
HAL_WWDGT_STATE_READY	窗口看门狗准备
HAL_WWDGT_STATE_BUSY	窗口看门狗忙

枚举 hal_wwdgt_struct_type_enum

表 3-620. hal_wwdgt_struct_type_enum

枚举名称	枚举描述
HAL_WWDGT_INIT_STRUCT	窗口看门狗初始化结构体
HAL_WWDGT_DEV_STRUCT	窗口看门狗设备信息结构体

结构体 hal_wwdgt_dev_struct

表 3-621. hal_wwdgt_dev_struct

成员名称	成员描述
state	窗口看门狗状态
mutex	互斥锁和解锁状态

结构体 hal_wwdgt_init_struct

表 3-622. hal_wwdgt_init_struct

成员名称	成员描述
wwdgt_pre_select	窗口看门狗时钟分频选择
wwdgt_cnt_value	窗口看门狗计数窗口值
wwdgt_downcnt_value	窗口看门狗计数值

结构体 hal_wwdgt_irq_struct

表 3-623. hal_wwdgt_irq_struct

成员名称	成员描述
early_wakeup_handler	窗口看门狗中断函数

函数 hal_wwdgt_struct_init

函数hal_wwdgt_struct_init描述见下表：

表 3-624. 函数 hal_wwdgt_struct_init

函数名称	hal_wwdgt_struct_init
函数原形	void hal_wwdgt_struct_init(hal_wwdgt_struct_type_enum hal_struct_type,void *p_struct);
功能描述	初始化WWDGT结构体
先决条件	-
被调用函数	-
输入参数{in}	
hal_struct_type	指针，指向hal_wwdgt_struct_type_enum枚举，枚举成员参考 表3-620. hal_wwdgt_struct_type_enum
输入参数{in}	
*p_struct	指向包含配置信息的WWDGT结构体的指针
输出参数{out}	
-	-
返回值	

-	-
---	---

例如:

```
/* initialize the WWDGT STRUCT */
```

```
hal_wwdgt_struct_init(HAL_WWDGT_INIT_STRUCT, &wwdgt_init_parameter);
```

函数 hal_wwdgt_init

函数hal_wwdgt_init描述见下表:

表 3-625. 函数 hal_wwdgt_init

函数名称	hal_wwdgt_init
函数原形	int32_t hal_wwdgt_init(hal_wwdgt_dev_struct *wwdgt_dev,hal_wwdgt_init_struct *p_wwdgt_init);
功能描述	启动WWDGT
先决条件	-
被调用函数	-
输入参数{in}	
wwdgt_dev	指向 WWDGT 设备信息结构体的指针，结构体成员参考 表 3-621. hal_wwdgt_dev_struct 。
输入参数{in}	
p_wwdgt_init	指向 WWDGT 初始化结构体的指针，结构体成员参考 表 3-622. hal_wwdgt_init_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_VAL / HAL_ERR_NONE

例如:

```
/* initialize the WWDGT */
```

```
hal_wwdgt_init(&wwdgt_info,&wwdgt_init_parameter);
```

函数 hal_wwdgt_deinit

函数hal_wwdgt_deinit描述见下表:

表 3-626. 函数 hal_wwdgt_deinit

函数名称	hal_wwdgt_deinit
函数原形	void hal_wwdgt_deinit(hal_wwdgt_dev_struct *wwdgt_dev);
功能描述	
先决条件	-
被调用函数	-
输入参数{in}	
wwdgt_dev	指向WWDGT设备信息结构体的指针，结构体成员参考 表3-621. hal_wwdgt_dev_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the configuration of WWDGT */
```

```
hal_wwdgt_deinit(&wwdgt_info);
```

函数 hal_wwdgt_irq

函数hal_wwdgt_irq描述见下表：

表 3-627. 函数 hal_wwdgt_irq

函数名称	hal_wwdgt_irq
函数原形	void hal_wwdgt_irq(hal_wwdgt_dev_struct *wwdgt_dev);
功能描述	WWDGT中断处理函数
先决条件	-
被调用函数	-
输入参数{in}	
wwdgt_dev	指向WWDGT设备信息结构体的指针，结构体成员参考 表3-621. hal_wwdgt_dev_struct 。
输出参数{out}	
-	-

返回值	
-	-

例如：

/* the function is used in the relative interrupt routine */

hal_wwdgt_dev_struct wwdgt_info;

WWDGT_IRQHandler(void)

```
{
    hal_wwdgt_irq(&wwdgt_info);
}
```

函数 hal_wwdgt_irq_handle_set

函数hal_wwdgt_irq_handle_set描述见下表：

表 3-628. 函数 hal_wwdgt_irq_handle_set

函数名称	hal_wwdgt_irq_handle_set
函数原形	void hal_wwdgt_irq_handle_set(hal_wwdgt_dev_struct *wwdgt_dev, hal_wwdgt_irq_struct *p_irq);
功能描述	
先决条件	-
被调用函数	-
输入参数{in}	
wwdgt_dev	向WWDGT设备信息结构体的指针，结构体成员参考 表3-621. hal_wwdgt_dev_struct
输入参数{in}	
p_irq	指向 WWDGT 中断回调函数结构体的指针，结构体成员参考 表 3-623. hal_wwdgt_irq_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* set user-defined interrupt callback function */

hal_wwdgt_dev_struct wwdgt_info;

hal_wwdgt_irq_struct wwdgt_irq;

wwdgt_irq.early_wakeup_handle = wwdgt_early_wakeup_func;

hal_dac_irq_handle_set(&dac_info,&dac_irq);

```

函数 hal_wwdgt_irq_handle_all_reset

函数hal_wwdgt_irq_handle_all_reset描述见下表：

表 3-629. 函数 hal_wwdgt_irq_handle_all_reset

函数名称	hal_wwdgt_irq_handle_all_reset
函数原形	void hal_wwdgt_irq_handle_all_reset(hal_wwdgt_dev_struct *wwdgt_dev);
功能描述	
先决条件	-
被调用函数	-
输入参数{in}	
wwdgt_dev	向WWDGT设备信息结构体的指针，结构体成员参考 表3-621. hal_wwdgt_dev_struct.
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* reset all user-defined interrupt callback function */

hal_wwdgt_dev_struct wwdgt_info;

hal_wwdgt_irq_handle_all_reset(&wwdgt_info);

```

函数 hal_wwdgt_start

函数hal_wwdgt_start描述见下表：

表 3-630. 函数 hal_wwdgt_start

函数名称	hal_wwdgt_start
------	-----------------

函数原形	void hal_wwdgt_start(void);
功能描述	开启WWDGT模块
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* start WWDGT module*/
```

```
hal_wwdgt_start ();
```

函数 hal_wwdgt_start_interrupt

函数hal_wwdgt_start_interrupt描述见下表:

表 3-631. 函数 hal_wwdgt_irq_handle_set

函数名称	hal_wwdgt_start_interrupt
函数原形	void hal_wwdgt_start_interrupt (hal_wwdgt_dev_struct *wwdgt_dev, hal_wwdgt_irq_struct *p_irq);
功能描述	
先决条件	-
被调用函数	-
输入参数{in}	
wwdgt_dev	向WWDGT设备信息结构体的指针，结构体成员参考 表3-621. hal_wwdgt_dev_struct 。
输入参数{in}	
p_irq	指向 WWDGT 中断回调函数结构体的指针，结构体成员参考 表 3-623. hal_wwdgt_irq_struct 。
输出参数{out}	

-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL

例如：

```
/* set user-defined interrupt callback function */
hal_wwdgt_dev_struct wwdgt_info;
hal_wwdgt_irq_struct wwdgt_irq;
wwdgt_irq.early_wakeup_handle = wwdgt_early_wakeup_func;
hal_wwdgt_start_interrupt (&dac_info,&dac_irq);
```

函数 hal_wwdgt_reload

函数hal_wwdgt_reload描述见下表：

表 3-632. 函数 hal_wwdgt_reload

函数名称	hal_wwdgt_reload
函数原形	int32_t hal_wwdgt_reload(hal_wwdgt_init_struct *p_wwdgt);
功能描述	-
先决条件	-
被调用函数	-
输入参数{in}	
p_wwdgt	指向WWDGT初始化结构体的指针，结构体成员参考 表3-621. hal_wwdgt_dev_struct 。
输出参数{out}	
-	-
返回值	
int32_t	HAL_ERR_NONE, HAL_ERR_ADDRESS, HAL_ERR_VAL_

例如：

```
/* start WWDGT module*/
hal_wwdgt_init_struct wwdgt_init_parameter;
wwdgt_init_parameter.wwdgt_pre_select = WWDGT_PSC_DIV8;
wwdgt_init_parameter.wwdgt_cnt_value = 127;
```

```
wwdgt_init_parameter.wwdgt_downcnt_value = 80;  
  
hal_wwdgt_reload(&wwdgt_init_parameter);hal_wwdgt_start ();
```

3.29. USBF

3.29.1. 外设寄存器说明

3.29.2. 外设库函数说明

4. 版本历史

表 4-1. 版本历史

版本号	描述	日期
1.0	初稿发布	2023 年 9 月 1 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company 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.