GR551x HAL及LL驱动用户手册

版本： **1.8**

发布日期： **2021-04-25**

深圳市汇顶科技股份有限公司

前言

编写目的

本文档介绍了GR551x系列芯片外设模块驱动的架构、文件分类、API分类、驱动命名规则等，并详细描述了HAL（Hardware Abstraction Layer）驱动和LL（Low Layer）驱动的使用方法、结构体定义以及API接口说明，旨在帮助开发者掌握各外设驱动的使用方法，能够利用HAL驱动中的API接口实现上层应用程序与底层硬件外设的交互，利用LL驱动中的API接口进行RTOS下的驱动移植及适配。

读者对象

本文适用于以下读者：

- GR551x用户
- GR551x开发人员
- GR551x测试人员
- 文档工程师

版本说明

本文档为第6次发布，对应的产品系列为GR551x。

修订记录

版本	日期	修订内容
1.0	2019-12-08	首次发布
1.3	2020-03-16	删除原“HAL CGC驱动”章节，修改日历相关接口、结构体等
1.5	2020-05-30	• 新增各外设睡眠时挂起配置寄存器和唤醒时恢复配置寄存器，新增禁止指定的AON GPIO唤醒系统API • 删除“PWR驱动API描述”章节下的Memory电源控制接口、对AON GPIO唤醒中断处理接口
1.6	2020-06-3	• 修改ADC参考电压值，涉及章节包括：“HAL ADC通用驱动”、“ADC驱动的结构体” • 更新“HAL HMAC通用驱动”章节，轮询、中断、DMA计算模式的使用方式、以及部分HMAC驱动API • 更新“Calendar驱动的结构”体章节中部分参数值
1.7	2020-12-18	• “pwm_init_t”和“ ll_pwm_init_t”章节新增PWM对齐方式描述 • “ rng_init_t”和“ll_rng_init_t”章节补充RNG_OUTPUT_FR0_S0的使用限制说明
1.8	2021-04-25	“HAL AES通用驱动”和“HAL PKC”通用驱动章节删除DMA描述

目录

1 概述

1.1 驱动架构

GR551x外设驱动软件架构分为硬件抽象层（Hardware Abstraction Layer，HAL）驱动和底层（Low Layer，LL）驱动（下文简称为HAL驱动和LL驱动），驱动架构如图 1-1所示。

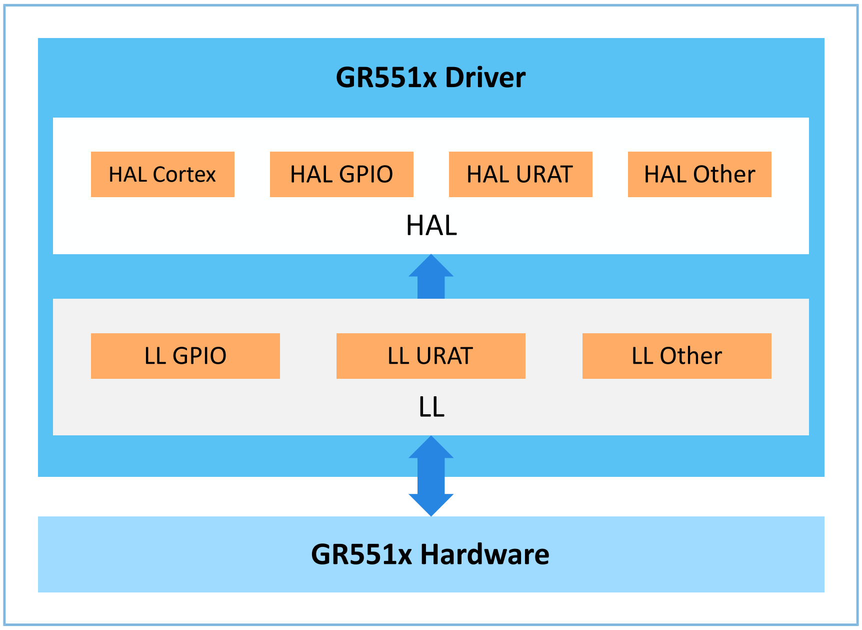

图 1-1 GR551x驱动架构

HAL层与LL层在软件架构上是相互关联的，HAL内部对寄存器的访问需要调用LL层的API。通常情况下，开发者需要使用HAL层的API来实现相应的功能。而对于HAL层无法覆盖的特殊功能，开发者可调用LL层接口快速封装所需的API接口来实现。

1.1.1 HAL驱动

HAL驱动将各个外设最常用的功能封装为一系列简单易用的API接口，开发者可利用这些接口轻松实现底层外设硬件与上层应用程序的交互。

HAL驱动主要具有以下特点：

- 驱动API可兼容同系列的芯片，部分芯片的特殊功能通过扩展API实现。
- 提供三种API编程模式：轮询、中断、DMA（Direct Memory Access）。
- API完全可重入，可在RTOS实时操作系统中使用。
- 多实例支持。同一API可在同一外设的多个实例（如I2C0、I2C1）中调用。
- 可在外设初始化/反初始化API中调用用户回调函数，以实现GPIO（General Purpose Input/Output）、中断、DMA的初始化/反初始化。
- 可在外设中断事件或错误中调用回调函数，通知用户相应事件被触发。
- 支持锁机制，可实现共享资源的安全访问。

- 驱动中的轮询处理采用超时机制实现，可避免程序死循环。

1.1.2 LL驱动

LL驱动通过内联函数的方式封装了各个外设寄存器的原子操作。因此，LL驱动比HAL驱动更加接近硬件，提供全面覆盖外设功能的API函数接口。开发者可使用LL驱动对HAL驱动无法覆盖到的外设功能进行配置。对于性能要求苛刻或存储空间不足的应用场景，开发者可直接使用LL驱动。

LL驱动主要具有以下特点：

- 采用内联函数实现，无函数调用开销。
- 屏蔽了寄存器操作，可移植性高，易于复用。
- 功能覆盖全面。

1.2 文件分类

根据文件的内容，可将文件分为驱动文件和用户程序文件两大类：

- 驱动文件：包括设备相关头文件、HAL驱动和LL驱动文件。
- 用户程序文件：指用户在创建工程时需要引用或实现的文件。

1.2.1 驱动文件

1.2.1.1 设备头文件

表 1-1 设备头文件

名称	描述
gr55xx.h	GR55xx系列芯片公共头文件，包括各子系列芯片的头文件，部分公共宏定义及枚举声明，例如：flag_status_t，SET_BIT(REG, BIT)，以及gr55xx_hal.h。
gr551xx.h	GR551x系列处理器的头文件，包含该处理器所有的外设控制寄存器结构体的声明和外设声明的结构体定义以及宏定义。

1.2.1.2 HAL驱动文件

GR551x的HAL层驱动文件的包含关系如图 1-2所示：

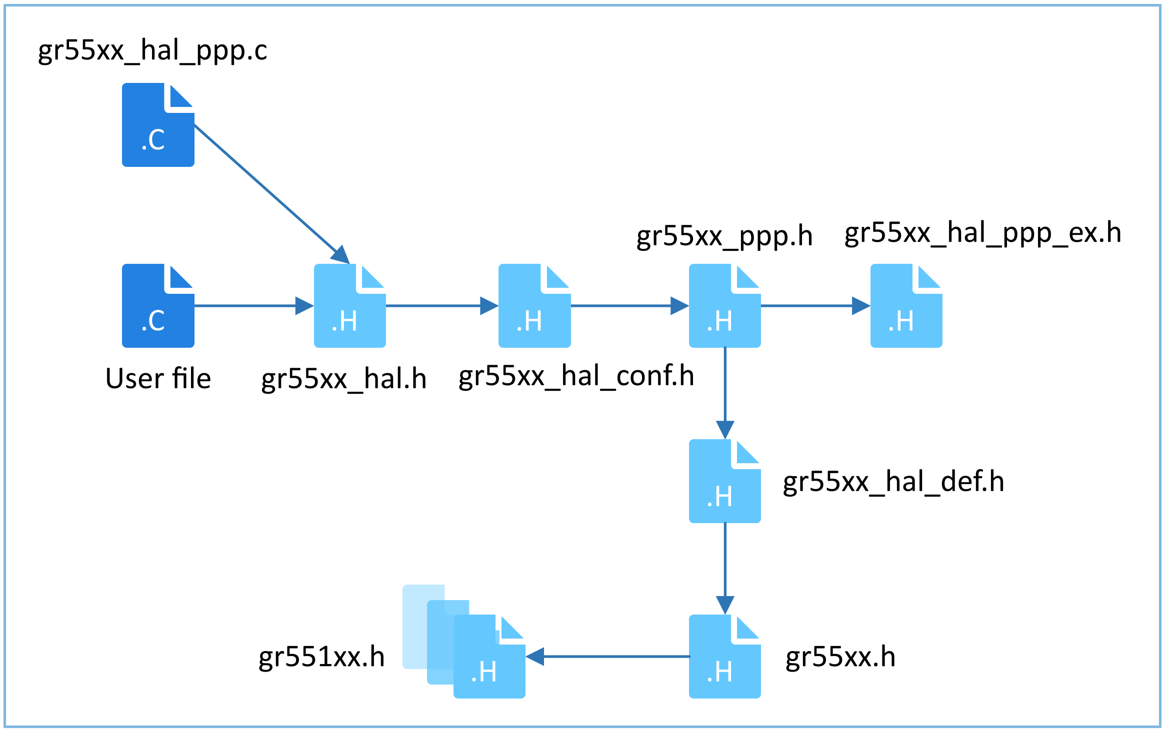

图 **1-2** 文件包含关系

HAL驱动文件描述如表 1-2：

表 1-2 HAL驱动文件描述

名称	描述
gr55xx_hal_conf.h	HAL驱动的配置文件，包含所有外设驱动的HAL头文件，开发者可通过修改该文件指定需要编译的外设模块以及其它的配置参数。
gr55xx_hal_ppp.h	各个模块HAL驱动的头文件，包含驱动API函数的声明、结构体类型声明、枚举声明、宏定义等，例如：*gr55xx_hal_uart.h*。
gr55xx_hal_ppp.c	各个模块HAL驱动的源文件，包含驱动API函数的实现，例如：*gr55xx_hal_uart.c*。
gr55xx_hal_ppp_ex.h	部分模块驱动程序的扩展功能的头文件，包含只有部分芯片支持的扩展功能的API函数声明，例如：*gr55xx_hal_gpio_ex.h*中包含了引脚复用的相关定义。
gr55xx_hal_ppp_ex.c	部分模块驱动程序的扩展功能的源文件，目前暂未实现。
gr55xx_hal.h	HAL驱动的公共头文件，包含了hal_init接口以及Tick相关函数，开发者在应用中包含该文件后，即可以使用GR551x的HAL驱动，该文件在用户应用与HAL驱动之间起桥接作用。
gr55xx_hal.c	HAL驱动的公共源文件，包括hal_init接口、Tick相关接口实现（weak类型，可根据需要进行重定义）。
gr55xx_hal_msp_template.c	hal_ppp_msp_init()和hal_ppp_msp_deinit()接口实现的模板，可对所有外设的Msp接口进行统一配置，其中hal_ppp_msp_init()在hal_ppp_init()中进行调用，主要进行相应模块GPIO复用、时钟、DMA、中断的配置。
gr55xx_hal_def.h	HAL驱动类型定义文件，包含公共宏的定义，结构体、枚举的声明，以及部分编译器相关的定义，如hal_status_t等。

名称	描述
	包含所有外设驱动使用的通用数据类型及常量。

说明:

ppp：表示外设名称，如gpio、qspi、uart等。

1.2.1.3 LL驱动文件

LL驱动文件描述如表 1-3：

表 1-3 驱动文件描述

名称	描述
gr55xx_ll_ppp.h	各个模块LL驱动的头文件，包含LL驱动的宏定义、结构体声明、以及LL驱动寄存器访问的内联接口实现，例如：*gr55xx_ll_gpio.h*。
gr55xx_ll_ppp.c	各个模块LL驱动的源文件，包含LL驱动的init及deinit接口，例如：*gr55xx_ll_gpio.c*。

1.2.2 用户程序文件

用户程序文件描述如表 1-4：

表 1-4 用户程序文件描述

名称	描述
system_gr55xx.c	包含SystemInit实现，在系统复位后进行系统初始化，目前也在该文件进行系统时钟配置。
startup_gr55xx.s	GR551x系列芯片的启动文件。
gr55xx_hal_msp.c	可选文件，用于统一放置各外设驱动模块的msp_init()及msp_deinit()接口，使用外设较少时也可放于*main.c*文件。
gr55xx_it.c/.h	可选文件，用于统一放置各个外设的中断处理函数，使用外设较少时也可直接放于*main.c*文件。
main.c/.h	主函数及头文件。

1.3 API分类

HAL驱动和LL驱动的API接口分为通用API和扩展API。

1.3.1 通用API

通用API是一系列对GR551x系列芯片提供一致操作的通用接口。下文将分别介绍HAL通用API和LL通用API的类别。

1.3.1.1 HAL通用API

根据接口的作用，HAL驱动通用API分为：

- 初始化类：用于初始化/反初始化外设及该外设使用的公用系统资源，如GPIO引脚上下拉及功能复用、NVIC（Nested Vectored Interrupt Controller）中断使能、DMA通道初始化等，例如：hal_uart_init()。
- IO操作类：用于串行接口的数据收发功能，例如：hal_uart_transmit()。
- 中断处理及回调函数类：用于外设各个类型的中断处理及回调函数的调用，例如：hal_uart_irq_handler()、hal_uart_tx_cplt_callback()。
- 控制类：用于设置外设驱动相应功能的参数，例如：hal_spi_set_tx_fifo_threshold()。
- 状态及错误类：用于获取HAL驱动的运行状态及错误码，例如：hal_i2c_get_state()。

1.3.1.2 LL通用API

根据接口的作用，LL驱动通用API分为：

- 初始化类：用于初始化/反初始化外设，例如：ll_pwm_init()。
- 功能使能类：用于启用/禁用外设的特定功能，例如：ll_dma_enable_channel()。
- 参数设置类：用于设置外设相应功能的参数，例如：ll_dma_set_data_transfer_direction()。
- 标志与状态类：用于判断外设寄存器的状态及标志，例如：ll_i2c_is_active_flag_stop_det()。
- 中断控制类：用于启用/禁用指定类型的中断，例如：ll_i2c_enable_it_stop_det()。
- DMA控制类：用于启用/禁用外设的DMA请求，例如：ll_i2c_enable_dma_req_tx()。

1.3.2 扩展API

扩展API是一系列只在特定的芯片中可用的接口，提供通用API之外的扩展操作。HAL/LL扩展API的分类与其通用API的分类相同。

1.4 驱动命名规则

GR551x驱动的命名规则包括基本命名规则、HAL驱动API命名规则及LL驱动API命名规则。

1.4.1 基本命名规则

基本命名规则为HAL驱动及LL驱动通用的命名规则，包括文件、模块、结构体、宏等的命名规则。

具体的命名规则描述如下：

表 1-5 基本命名规则

类别	命名样式	命名说明	示例
文件	ccc_ddd_ppp.c/h ccc_ddd_ppp_ex.c/h	文件名称主要由芯片型号、驱动类型、外设名称组成。如果为扩展驱动，则需加“_ex”后缀。	gr55xx_hal_gpio.(c/h) gr55xx_hal_gpio_ex.(c/h) gr55xx_ll_gpio.(c/h)

类别	命名样式	命名说明	示例
模块	HAL_PPP_MODULE	模块名称主要由驱动类型和外设名称、“_MODULE”后缀组成。	HAL_I2C_MODULE
宏定义	PPP_PARAM LL_PPP_PARAM	宏定义名称采用大写英文字母，并且LL驱动的宏定义名称添加了“LL_”前缀。	UART_DATABITS_8 LL_UART_PARITY_NONE
结构体	ppp_sss_t ll_ppp_sss_t	结构体名称主要由外设名称、结构体类型组成，并且结构体后面直接跟“_t”后缀，LL驱动的结构体还添加了“ll_”前缀。	qspi_handle_t ll_uart_init_t
枚举	ddd_ppp_enumname_t	枚举名称主要由驱动类型、外设名称、枚举类型、“_t”后缀组成。	hal_uart_state_t
枚举标签	DDD_PPP_ENUM	枚举标签名称由驱动类型、外设名称、标签含义组成，采用大写英文字母。	HAL_UART_STATE_RESET
寄存器	REGISTERNAME	寄存器名称采用大写英文字母，命名方式与《GR551x Datasheet》中基本保持一致（部分过长的寄存器名会适当缩写）。	MODEM_CTRL
寄存器结构体	ppp_regs_t	寄存器结构体名称由外设名称、“_regs_t”后缀组成。	uart_regs_t

命名说明：

- ccc：芯片系列型号，例如：gr55xx。
- DDD/ddd：驱动类型，例如：HAL/hal，LL/ll。
- PPP/ppp：外设名称，例如：GPIO/gpio，QSPI/qspi，URAT/uart等。
- sss：结构体类型，例如：handle，init。
- PARAM：外设参数。
- ENUM：枚举标签名。
- REGISTERNAME：寄存器名称。

1.4.2 HAL驱动API命名规则

HAL驱动各类API的命名规则如下：

表 1-6 HAL层API命名规则

API类别	命名样式	命名说明
初始化类	hal_ppp_init hal_ppp_deinit	函数名主要由驱动类型（hal）、外设名称、初始化/反初始化组成。
IO操作类	hal_ppp_operate	函数名主要由驱动类型（hal）、外设名称、操作方式（比如发送，接收，回调函数处理等）组成。

API类别	命名样式	命名说明
	hal_ppp_command_operate	对于命令的收发，在外设名称与操作之间添加了command。
	hal_ppp_operate_it	中断模式下的操作添加了“_it”后缀。
	hal_ppp_operate_dma	DMA模式下的操作添加了“_dma”后缀。
中断处理及回调类	hal_ppp_irq_handler	中断处理函数命名为驱动类型（hal）、外设名称、“_irq_handler”后缀组成。
	hal_ppp_operate_cplt_callback	操作完成的回调函数名，直接在操作方式后加“_cplt_callback”后缀。
	hal_ppp_error_callback	函数名主要由驱动类型（hal）、外设名称、后缀“_error_callback”构成。
控制类	hal_ppp_set_parameter hal_ppp_get_parameter	函数名主要由驱动类型（hal）、外设名称、参数名称组成。
状态及错误类	hal_ppp_get_state hal_ppp_get_error	函数名主要由驱动类型（hal）、外设名称、状态/错误操作组成。

说明:

- PPP/ppp：外设名称，例如：QSPI/qspi，URAT/uart等。
- operate：操作类型，例如：transmit/tx，receive/rx，abort等。
- parameter：参数名称，例如：fifo_threshold，timeout等。

下面以QSPI为例说明HAL驱动API的命名规则。

表 1-7 QSPI的HAL层API命名

API类别	函数名	函数描述
初始化类	hal_qspi_init	初始化QSPI，设置时钟、引脚复用等参数。
	hal_qspi_deinit	反初始化QSPI，恢复初始设置。
	hal_qspi_mspinit	初始化QSPI使用的GPIO、中断NVIC、DMA。
	hal_qspi_mspdeinit	反初始化QSPI使用的GPIO、中断NVIC、DMA。
IO操作类	hal_qspi_command_transmit	轮询方式的数据收发接口。
	hal_qspi_command_receive	
	hal_qspi_command	
	hal_qspi_transmit	
	hal_qspi_receive	
	hal_qspi_command_transmit_it	中断方式的数据收发接口。
	hal_qspi_command_receive_it	
	hal_qspi_command_it	
	hal_qspi_transmit_it	

API类别	函数名	函数描述
	hal_qspi_receive_it	
	hal_qspi_command_transmit_dma	DMA方式的数据收发接口。
	hal_qspi_command_receive_dma	
	hal_qspi_command_dma	
	hal_qspi_transmit_dma	
	hal_qspi_receive_dma	
	hal_qspi_abort	终止当前正在进行的传输操作。
	hal_qspi_abort_it	
中断处理及回调类	hal_qspi_irq_handler	中断处理函数。
	hal_qspi_tx_cplt_callback	发送完成回调函数。
	hal_qspi_rx_cplt_callback	接收完成回调函数。
	hal_qspi_error_callback	错误检测回调函数。
	hal_qspi_abort_cplt_callback	中止完成回调函数。
控制类	hal_qspi_set_timeout	设置超时。
	hal_qspi_set_tx_fifo_threshold	设置FIFO阈值。
	hal_qspi_set_rx_fifo_threshold	
	hal_qspi_get_tx_fifo_threshold	读取FIFO阈值。
	hal_qspi_get_rx_fifo_threshold	
状态及错误类	hal_qspi_get_state	读取外设状态。
	hal_qspi_get_error	读取错误码。

1.4.3 LL驱动API命名规则

LL驱动各类API的命名规则如下：

表 1-8 LL层API命名规则

API类别	命名样式	命名说明
初始化类	ll_ppp_init ll_ppp_deinit	由驱动类型（ll）、外设名称、初始化/反初始化组成。
功能使能类	ll_ppp_enable_function ll_ppp_disable_function ll_ppp_is_enabled_function	由驱动类型（ll）、外设名称、使能/禁止/是否使能、功能名组成。
IO操作类	ll_ppp_transmit_dataN ll_ppp_receive_dataN	由驱动类型（ll）、外设名称、transmit/receive、data及位宽组成。
参数设置类	ll_ppp_set_parameter ll_ppp_get_parameter	由驱动类型（ll）、外设名称、设置/读取、参数名组成。
标志与状态类	ll_ppp_is_active_flag_flagname	该类型API的命名方式主要包括以下两种：

API类别	命名样式	命名说明
	ll_ppp_clear_flag_flagname ll_ppp_clear_flag ll_ppp_clear_flagtype_flag ll_ppp_get_flagtype_flag	• 由驱动类型（ll）、外设名称、is_active/clear、_flag、标志名称组成，用于判断外设状态或清除单个标志 • 由驱动类型（ll）、外设名称、get/clear、标志类型、后缀"_flag"组成，用于获取或清除某类型的标志，其中部分外设无标志类型
中断控制类	ll_ppp_enable_it_itname ll_ppp_disable_it_itname ll_ppp_is_enabled_it_itname ll_ppp_enable_it ll_ppp_disable_it ll_ppp_is_enabled_it	由驱动类型（ll）、外设名称、使能/禁止/是否使能、it、中断名称组成，其中没有中断名称的接口（如ll_ppp_enable_it）可用于控制多个中断。
DMA控制类	ll_ppp_enable_dma_req_tx/rx ll_ppp_disable_dma_req_tx/rx ll_ppp_is_enabled_dma_req_tx/rx	由驱动类型（ll）、外设名称、使能/禁止/是否使能、DMA操作请求类型组成。

说明:

- PPP/ppp：外设名称，例如：QSPI/qspi，URAT/uart。
- function：需要操作的外设功能名称，例如：I2C中的general_call。
- N：IO操作中的数据位宽，可为8、16、32。
- parameter：参数名称，例如：fifo_threshold，timeout。
- flagname：标志名称，例如：I2C中的STOP_DET中断的flagname为stop_det。
- flagtype：需要清除或获取的标志类型，例如：it、line_status。
- itname：中断名称，例如：UART中的RDA中断的itname为rda。

下面以UART为例说明LL驱动API的命名规则。

表 1-9 UART的LL层API命名

API类别	函数名	函数描述
初始化类	ll_uart_init	初始化UART，设置波特率、数据位等参数。
	ll_uart_deinit	反初始化UART，恢复初始设置。
功能使能类	ll_uart_enable_fifo	控制及判断FIFO的使能状态。
	ll_uart_disable_fifo	
	ll_uart_is_enabled_fifo	
IO操作类	ll_uart_transmit_data8	发送单字节数据。
	ll_uart_receive_data8	接收单字节数据。
参数设置类	ll_uart_set_parity	设置奇偶校验位。

API类别	函数名	函数描述
	ll_uart_get_parity	获取奇偶校验位。
标志与状态类	ll_uart_is_active_flag_rff	判断RFF标志是否置1。
	ll_uart_get_line_status_flag	获取Line状态。
	ll_uart_clear_line_status_flag	清除Line状态。
	ll_uart_get_it_flag	获取中断状态。
中断控制类	ll_uart_enable_it_rda	控制及判断RDA接收数据可用中断的状态。
	ll_uart_disable_it_rda	
	ll_uart_is_enabled_it_rda	
	ll_uart_enable_it	控制及判断多个中断状态。
	ll_uart_disable_it	
	ll_uart_is_enabled_it	
DMA控制类	无	UART的DMA请求由硬件管理，软件不需要设置。

1.5 数据结构

每个外设模块的HAL驱动中均包含外设句柄结构体、初始化结构体以及配置结构体。

此外，为了简化单独使用LL驱动时的参数配置流程，每个外设的LL驱动中也定义了初始化结构体，以便外设进行必要的初始化配置。

1.5.1 外设句柄结构体

HAL驱动采用多实例架构，可允许同一外设模块多个实例的连续执行。在该架构中，最主要的结构体就是外设句柄结构体ppp_handle_t *handle。可用于定义相应外设每个实例的句柄，保存着每个实例的外设配置参数、寄存器结构体指针以及各种运行时变量。

外设句柄的作用主要包括：

- 多实例支持：每个外设实例均有各自独立的句柄，即每个实例的外设配置参数、运行时变量等均是独立的。
- API间通信：句柄保存着运行时的共享变量，可实现各个API之间的数据交互。
- 存储：句柄也可用于存储和管理指定外设驱动的全局变量。

下面以SPI的句柄结构体举例说明：

```
typedef struct
{
    ssi_regs_t    *p_instance; /**< SPI registers base address*/
    spi_init_t    init;               /**< SPI communication parameters*/
    uint8_t     *p_tx_buffer; /**< Pointer to SPI Tx transfer Buffer */
    __IO uint32_t  tx_xfer_size; /**< SPI Tx Transfer size*/
    __IO uint32_t  tx_xfer_count; /**< SPI Tx Transfer Counter*/
    uint8_t     *p_rx_buffer; /**< Pointer to SPI Rx transfer Buffer */
```

```
    __IO uint32_t  rx_xfer_size; /**< SPI Rx Transfer size*/
    __IO uint32_t  rx_xfer_count ; /**< SPI Rx Transfer Counter*/
    void (*write_fifo)(struct _spi_handle *p_spi); /**< Pointer to SPI Tx transfer
                                                  FIFO write function */
    void (*read_fifo)(struct _spi_handle *p_spi); /**< Pointer to SPI Rx transfer
                                                 FIFO read function  */
    dma_handle_t   *p_dmatx;  /**< SPI Tx DMA Handle parameters*/
    dma_handle_t   *p_dmarx;  /**< SPI Rx DMA Handle parameters*/
    __IO hal_lock_t  lock;   /**< Locking object*/
    __IO hal_spi_state_t state;   /**< SPI communication state*/
    __IO uint32_t  error_code;  /**< SPI Error code*/
    uint32_t    timeout;  /**< timeout for the SPI memory access*/
} spi_handle_t;
```

说明:

被其它模块共用的系统外设（例如：GPIO、SYSTICK、NVIC、PWR等）没有句柄结构体。

1.5.2 初始化结构体

初始化结构体主要用于保存初始化外设的配置参数。

下面以UART的初始化结构体（可对串口的波特率、数据位、停止位、硬件流控、接收超时使能进行设置）举例说明：

```
typedef struct
{
    uint32_t baud_rate;
    uint32_t data_bits;
    uint32_t stop_bits;
    uint32_t parity;
    uint32_t hw_flow_ctrl;
    uint32_t rx_timeout_mode;
} uart_init_t;
```

1.5.3 配置结构体

配置结构体主要用于配置子模块或者子实例的参数。

下面以PWM中的通道初始化结构体举例说明：

```
typedef struct
{
 uint8_t duty; /**< Specifies the duty in PWM output mode. This parameter must be a number
 between 0 ～ 100.*/
 uint8_t drive_polarity; /**< Specifies the drive polarity in PWM output mode.This parameter
 can be a value of @ref PWM_DRIVEPOLARITY.*/
} pwm_channel_init_t;
```

2 HAL驱动

2.1 简介

本节将对HAL驱动中各个外设模块所使用的公共资源及如何使用HAL驱动进行简单介绍。

2.1.1 HAL公共资源

在GR551x的HAL驱动中，各个外设模块使用的公共枚举、结构体、宏定义在*gr55xx_hal_def.h*中，包括如下内容：

- HAL状态：HAL状态用于除布尔函数及中断处理函数之外的API接口，表示API的运行状态。其定义如下：

```
typedef enum
{
    HAL_OK       = 0x00U,
    HAL_ERROR    = 0x01U,
    HAL_BUSY     = 0x02U,
    HAL_TIMEOUT  = 0x03
} hal_status_t;
```

- HAL锁：HAL锁用于避免共享资源的非法访问，其定义如下：

```
typedef enum
{
    HAL_UNLOCKED = 0x00U,
    HAL_LOCKED   = 0x01
} hal_lock_t;
```

- 公共宏：主要包括超时机制中的最大延时宏、外设与DMA实例句柄的连接宏、用于消除未使用参数编译警报的宏等，其定义如下：

```
#define HAL_MAX_DELAY      0xFFFFFFFFU

#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_)  \
do{                                                                 \
          (__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_);        \
          (__DMA_HANDLE_).p_parent = (__HANDLE__);                  \
} while(0U)

#define UNUSED(x)  ((void)(x))
```

2.1.2 如何使用HAL驱动

GR551x的HAL层驱动的调用流程如图 2-1所示。

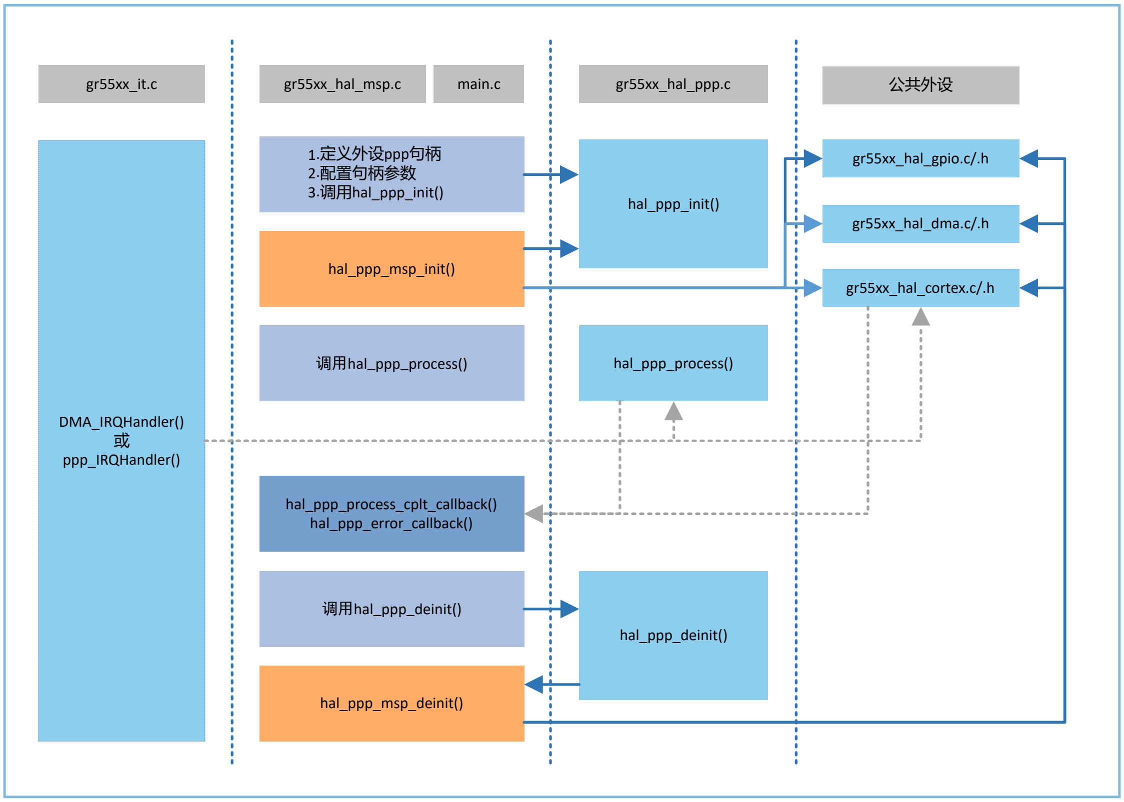

图 2-1 HAL驱动调用流程

说明:

- 图中 浅蓝框内的函数为HAL驱动实现的函数。
- 图中 深蓝框内的操作为开发者需在用户程序中实现的代码。
- 图中 橙色框内的函数为在用户程序中实现的msp（MCU Specific Package）函数。
- 图中 紫色框线内的函数为在中断处理函数中调用的函数。

具体的调用流程描述如下：

1. 开发者在用户程序文件（*main.c*或*gr55xx_msp.c*等）中重写外设PPP的msp函数hal_ppp_msp_init()及hal_ppp_msp_deinit()。
2. 如果需要使用中断方式的API，则还需重写相应的回调函数，如hal_ppp_process_cplt_callback()。
3. 开发者在用户程序文件中定义外设PPP的句柄（handle），并配置相关参数。
4. 开发者调用hal_ppp_init()初始化外设PPP，其中hal_ppp_init()由PPP的驱动文件实现，初始化过程中调用开发者重写的hal_ppp_msp_init() 以初始化PPP使用的GPIO引脚、NVIC中断、DMA通道。
5. 对于轮询、中断、DMA三种IO操作方式，各自的调用流程如下：
   - 轮询方式下，开发者调用驱动文件中的hal_ppp_process()进行IO操作，操作完成后才会退出当前函数。

- 中断方式下，开发者调用驱动文件中的hal_ppp_process_it()进行IO操作，中断使能后退出当前函数，IO操作会在PPP的中断处理函数PPP_IRQHandler中进行，操作完成时调用开发者重写的回调函数，从而进行事件完成通知。
- DMA方式下，开发者调用驱动文件中的hal_ppp_process_dma()进行IO操作，传输开始后退出当前函数，IO操作由DMA外设实现，操作完成时DMA的中断处理函数DMA_IRQHandler()会调用开发者重写的回调函数，从而进行事件完成通知。

6. 外设PPP使用完成时，开发者可调用PPP驱动文件中的hal_ppp_deinit()反初始化外设PPP，恢复相应的寄存器为默认值。反初始化过程中会调用开发者重写的hal_ppp_msp_deinit()以反初始化PPP使用的GPIO引脚、NVIC中断及DMA通道。

下文将对HAL驱动调用流程中的初始化、IO操作、超时检测及错误检查进行详细说明。

2.1.2.1 HAL驱动初始化

2.1.2.1.1 HAL全局初始化

*gr55xx_hal.c*中提供了若干API接口，可对HAL驱动所使用的SysTick进行初始化和反初始化，从而实现外设驱动轮询方式数据收发的超时检测。

- hal_init()：系统启动后可调用该函数实现以下功能：

  调用hal_msp_init()初始化时钟、GPIO、中断、DMA等。
- hal_deinit()：调用hal_msp_deinit()反初始化时钟、GPIO、中断、DMA等。

2.1.2.1.2 外设MSP初始化

在初始化外设PPP的过程中，hal_ppp_init()会调用hal_ppp_msp_init()函数对该外设所使用的GPIO、中断、DMA进行初始化。在反初始化的过程中，则会调用hal_ppp_msp_deinit()。hal_ppp_msp_init()及hal_ppp_msp_deinit()为weak类型的空函数，在实际使用过程中，开发者需根据具体应用重写，这两个函数在驱动中的weak类型的空函数如下：

```
__weak void hal_ppp_msp_init(ppp_handle_t *p_ppp)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(p_ppp);
}

__weak void hal_ppp_msp_deinit(ppp_handle_t *p_ppp)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(p_ppp);
}
```

2.1.2.2 HAL驱动IO操作

HAL驱动对外设的IO读写操作包括三种读写方式：轮询、中断、DMA。

2.1.2.2.1 轮询方式

在轮询方式中，数据的读写操作是轮询式的，即读写API会在数据的读写操作完成时才退出。读写成功完成时，读写API返回HAL_OK状态，否则返回HAL_ERROR或HAL_TIMEOUT状态。通过hal_ppp_get_state()及hal_ppp_get_error()可获取具体的状态及错误码。在轮询方式的读写API中，为了避免程序陷入死循环，HAL驱动采用了超时检测机制，可使用API中的timeout参数指定超时时间。

轮询方式读写API的典型定义如下：

```
hal_status_t hal_ppp_transmit(ppp_handle_t *p_ppp, uint8_t *p_data, uint16_t size, uint32_t
 timeout)
{
    if ((NULL == p_data) || (0U == size))
    {
        return  HAL_ERROR;
    }
    (…)
    while (data processing is running)
   {
       if (timeout reached)
      {
           return HAL_TIMEOUT;
       }
    }
(…)
return HAL_OK;
}
```

2.1.2.2.2 中断方式

在中断方式中，数据的读写操作是非轮询式的，即读写API会在使能读写中断后退出，读写操作将在外设中断处理函数中进行。读写完成时，程序将调用开发者实现的回调函数来通知APP。开发者可以通过hal_ppp_get_state()接口来获取当前的读写状态。

在中断方式中，驱动中主要定义了如下接口函数：

- hal_ppp_process_it()：中断方式的读写API。
- hal_ppp_irq_handler()：外设PPP的中断处理函数。
- _weak hal_ppp_process_cplt_callback()：操作完成时的回调函数，由开发者实现。
- _weak hal_ppp_process_error_callback()：操作出错时的回调函数，由开发者实现。

使用中断方式的API接口时，开发者首先需要在*gr55xx_it.c*中注册中断处理函数hal_ppp_irq_handler()，然后调用hal_ppp_process_it()进行数据读写。

回调函数在外设的HAL驱动中被定义为weak类型，即开发者需自己实现要使用的回调函数，从而在数据读写完成时进行缓冲区内存的释放等操作。

例如，通过中断方式进行UART0读写操作的接口使用示例代码如下：

*main.c*文件中：

```
uart_handle_t uart_handle;
int main(void)
{
    uart_handle.p_instance          = UART0;
    uart_handle.init.baud_rate      = 115200;
    uart_handle.init.data_bits      = UART_DATABITS_8;
    uart_handle.init.stop_bits      = UART_STOPBITS_1;
    uart_handle.init.parity         = UART_PARITY_NONE;
    uart_handle.init.hw_flow_ctrl   = UART_HWCONTROL_NONE;
    hal_uart_init(&uart_handle);

    char *p_tx_buff = "Hello World!\r\n";
    hal_uart_transmit_it(&uart_handle,p_tx_buff,strlen((char*)p_tx_buff));
    while (hal_uart_get_state(&uart_handle) != HAL_UART_STATE_READY);

    hal_uart_deinit(&uart_handle);
}

void hal_uart_tx_cplt_callback(uart_handle_t *p_uart)
{
    (…)

}
void hul_uart_error_callback(uart_handle_t *p_uart)
{
    (…)
}
```

*gr55xx_it.c*文件中：

```
void UART0_IRQHandler(void)
{
    Hal_uart_irq_handler(&uart_handle);
}
```

说明:

UART0_IRQHandler()也可直接放在*main.c*中。

2.1.2.2.3 DMA方式

在DMA方式中，数据的读写操作也是非轮询式的。在数据读写完成时，中断处理函数调用相应的回调函数通知APP。开发者还可通过hal_ppp_get_state()接口来读取当前读写操作的状态。

在DMA方式中，驱动中主要定义了如下API接口：

- hal_ppp_process_dma()：DMA方式的读写API。
- hal_ppp_irq_handler()：外设PPP的中断处理函数。

- _weak hal_ppp_process_cplt_callback()：操作完成时的回调函数，由开发者实现。
- _weak hal_ppp_process_error_callback()：操作出错时的回调函数，由开发者实现。

使用DMA方式的API接口时，开发者首先需注册中断处理函数hal_dma_irq_handler()，对于部分外设还需要注册hal_ppp_irq_handler()，如I2C。然后在hal_ppp_msp_init()中初始化需要使用的DMA通道，最后再调用hal_ppp_process_dma()进行数据读写。

例如，通过DMA方式进行UART0的读写操作，接口使用的示例代码如下：

- *main.c*文件

```
uart_handle_t uart_handle;
int main(void)
{
    uart_handle.p_instance          = UART0;
    uart_handle.init.baud_rate      = 115200;
    uart_handle.init.data_bits      = UART_DATABITS_8;
    uart_handle.init.stop_bits      = UART_STOPBITS_1;
    uart_handle.init.parity         = UART_PARITY_NONE;
    uart_handle.init.hw_flow_ctrl   = UART_HWCONTROL_NONE;
    hal_uart_init(&uart_handle);

    char *p_tx_buff = “Hello World!\r\n”;
    hal_uart_transmit_dma(&uart_handle, p_tx_buff, strlen((char*)p_tx_buff));
    while (hal_uart_get_state(&uart_handle) != HAL_UART_STATE_READY);
    hal_uart_deinit(&uart_handle);
}

void hal_uart_tx_cplt_callback(uart_handle_t *p_uart)
{
    (…)
}
void hul_uart_error_callback(uart_handle_t *p_uart)
{
    (…)
}
```

- *gr55xx_hal_msp.c*文件

```
void hal_uart_msp_init (uart_handle_t *p_uart)
{
    static dma_handle_t hdma_tx;
    static dma_handle_t hdma_rx;
    (…)
    hal_dma_init(&hdma_tx);
    hal_dma_init(&hdma_rx);

    __HAL_LINKDMA(p_uart, p_dmatx, hdma_tx);
    __HAL_LINKDMA(p_uart, p_dmarx, hdma_rx);
    (…)
```

```
}
```

- *gr55xx_it.c*文件

```
void UART0_IRQHandler(void)
{
    hal_uart_irq_handler(&uart_handle);
}
void DMA_IRQHandler(void)
{
    Hal_uart_irq_handler(uart_handle.p_dmatx);
}
```

说明:

hal_uart_msp_init()和UART0_IRQHandler()均可直接放在*main.c*中。

2.1.2.3 HAL驱动超时检测及错误检查

2.1.2.3.1 超时检测

在轮询方式的API接口中，HAL驱动通过超时检测来避免程序因为错误状态而进入死循环中。如下为SPI轮询方式发送数据的API接口：

```
hal_status_t hal_spi_transmit(spi_handle_t *p_spi, uint8_t *p_data, uint32_t length,
                              uint32_t timeout)
```

接口中的输入参数timeout为发送数据所需要的最大超时时间。一旦超过指定的超时时间，API接口将返回HAL_TIMEOUT。

HAL驱动中的超时时间在文件*gr55xx_hal_def.h*中定义，取值范围为0 ～ HAL_MAX_DELAY，其中HAL_MAX_DELAY = 0xFFFFFFFF。具体的超时时间取值及描述如表 2-1 所示：

表 2-1 超时的取值

超时时间值	描述
0	无超时，不等待，若标志检测条件不满足则立即退出循环。
1 ～ (HAL_MAX_DELAY - 1)	超时时间，单位ms。
HAL_MAX_DELAY	一直循环，直到操作成功完成才退出。

此外，在某些情况下，固定长度的超时时间会在某些外设的HAL驱动中使用，例如I2C的busy检测超时时间为25 ms。与API接口输入参数中的超时时间不同，固定长度的超时时间在API内部以宏的方式进行使用，并且不能被修改。

2.1.2.3.2 错误检查

为了提高驱动程序的健壮性，避免出现不可预期的错误，HAL驱动中实现了错误检查机制。

- 输入参数有效性检查

  在给开发者提供的API函数中，作为输入参数的变量需确保是已定义的且有效的，否则会导致程序崩溃或者进入未定义的状态，因此在HAL驱动中实现了输入参数的有效性检查。

  例如，在API函数hal_uart_transmit_it()中会检查输入参数的缓冲区指针和长度的有效性，代码如下：

```
hal_status_t hal_uart_transmit_it(uart_handle_t *p_uart, uint8_t *p_data, uint16_t size)
{
    /* Check that a Tx process is not already ongoing */
    if(HAL_UART_STATE_READY == p_uart->g_state)
    {
        if((NULL == p_data ) || (0U == size))
        {
            return HAL_ERROR;
        }
        (…)
    }
}
```

- 句柄有效性检查

  外设句柄是外设HAL驱动中最重要的参数，其存储着外设驱动的配置参数以及各种运行时变量等，因此在外设的初始化函数hal_ppp_init()中实现外设句柄有效性检查。

  例如，在UART的初始化函数hal_uart_init中检查句柄的有效性，代码如下：

```
hal_status_t hal_uart_init(uart_handle_t *p_uart)
{
    /* Check the UART handle allocation */
    if(NULL == p_uart)
    {
        return HAL_ERROR;
    }
   (…)
}
```

- 超时错误检查

  在轮询方式的API函数中，需要对操作时间进行检查。一旦超时，需返回超时状态。

  例如，UART在轮询方式下接收数据时，检查是否超时，代码如下：

```
hal_status_t hal_uart_receive(uart_handle_t *p_uart, uint8_t *p_data, uint16_t size,
                              uint32_t timeout)
{
(…)
    /* as long as data have to be received */
    while(0U < p_uart->rx_xfer_count)
    {
        if(HAL_OK != uart_wait_fifo_flag_until_timeout(p_uart, UART_FLAG_FIFO_RFNE, RESET,
                                                        tickstart, timeout)
```

```
        {
            return HAL_TIMEOUT;
        }
        (…)
    }
(…)
}
```

在外设句柄中定义了error_code这一全局变量，用于存储API操作过程中的错误码。因此，在API函数返回HAL_ERROR后，开发者可调用hal_ppp_get_error()函数获取具体的错误类型，例如：

```
uint32_t hal_uart_get_error(uart_handle_t *p_uart)
{
    return p_uart -> error_code;
}
```

2.1.2.3.3 运行时参数检查

HAL驱动提供了输入参数的运行时检查，用于判断输入的参数是否在允许的取值范围内。运行时检查是通过gr_assert_param宏实现的，该宏定义在*gr55xx_hal_conf.h*头文件中，可通过USE_FULL_ASSERT宏来启用/禁用运行时检查功能。

例如，在I2C的初始化函数中对I2C实例、速度、本机设备地址、设备地址模式等参数进行判断，代码如下：

```
hal_status_t hal_i2c_init(i2c_handle_t *p_i2c)
{
    (…)
    /* Check the parameters */
    gr_assert_param(IS_I2C_ALL_INSTANCE(p_i2c->instance));
    gr_assert_param(IS_I2C_SPEED(p_i2c->init.speed));
    gr_assert_param(IS_I2C_OWN_ADDRESS(p_i2c->init.own_address));
    gr_assert_param(IS_I2C_ADDRESSING_MODE(p_i2c->init.addressing_mode));
    gr_assert_param(IS_I2C_GENERAL_CALL(p_i2c->init.general_call_mode));
    (…)
}
```

如果gr_assert_param宏的判断结果是false，则assert_failed()函数将会被调用，并返回判断出错的文件名及行号，其中的assert_failed()函数需要由开发者进行实现。gr_assert_param宏的定义及实现的代码如下：

```
#ifdef  USE_FULL_ASSERT
/**
  * @brief The gr_assert_param macro is used for function's parameterscheck.
  * @param  expr If expr is false, it calls assert_failed function
  *         which reports the name of the source file and the source
  *         line number of the call that failed.
  *         If expr is true, it returns no value.
  * @retval None
  */
#define gr_assert_param(expr) ((expr) ? (void)0U : assert_failed((char *)__FILE__, __LINE__))
```

```
/* Exported functions ------------------------------------------------------- */
  void assert_failed(char* file, uint32_t line);
#else
  #define gr_assert_param(expr) ((void)0U)
#endif /* USE_FULL_ASSERT */
```

2.2 HAL Cortex通用驱动

2.2.1 Cortex驱动功能

通过对*core_cm4.h*中NVIC相关API的二次封装，Cortex驱动提供了一系列中断控制及SysTick配置API，它主要实现了以下功能：

- 中断的优先级配置及管理。
- 中断的使能、禁止。
- 中断的挂起、清除、查询。
- SysTick计数器的初始化及时钟源设置。

2.2.2 如何使用Cortex驱动

Cortex HAL驱动的使用方法如下：

1. 开发者可在重写的hal_msp_init()中调用hal_nvic_set_priority_grouping()，从而实现中断的优先级分组配置。
2. 开发者可在重写的hal_ppp_msp_init()中调用hal_nvic_clear_pending_irq()、hal_nvic_enable_irq()以及hal_nvic_set_priority()，从而实现外设ppp的中断优先级配置和使能。
3. 开发者可在重写的hal_ppp_msp_deinit()中调用hal_nvic_disable_irq()，从而实现外设ppp的中断禁止。
4. 开发者可通过调用hal_systick_config()实现SysTick的初始化配置。

2.2.3 Cortex驱动API描述

Cortex驱动的API主要包括：

表 2-2 Cortex驱动的HAL层APIs

API类别	API名称	描述
初始化	hal_nvic_set_priority_grouping()	设置中断优先级分组配置。
	hal_nvic_set_priority()	设置中断优先级。
	hal_nvic_enable_irq()	使能中断。
	hal_nvic_disable_irq()	禁止中断。
	hal_nvic_system_reset()	系统复位。
	hal_systick_config()	SysTick配置。

API类别	API名称	描述
控制类	hal_nvic_get_priority_grouping()	获取中断优先级分组配置。
	hal_nvic_get_priority()	获取中断优先级。
	hal_nvic_set_pending_irq()	挂起中断。
	hal_nvic_get_pending_irq()	获取中断挂起状态。
	hal_nvic_clear_pending_irq()	清除中断挂起状态。
	hal_nvic_get_active()	获取中断活跃状态。
	hal_systick_clk_source_config()	设置SysTick时钟源。
中断处理及回调函数	hal_systick_irq_handler()	中断处理函数。
	hal_systick_callback()	中断回调函数。

下面章节将对各API进行详细描述。

2.2.3.1 hal_nvic_set_priority_grouping

表 2-3 hal_nvic_set_priority_grouping接口

函数原型	void hal_nvic_set_priority_grouping(uint32_t priority_group)
功能说明	设置中断优先级分组配置
输入参数	priority_group：待设置的优先级分组配置参数，该参数可以是下列值中的任意一个： • NVIC_PRIORITYGROUP_0（0位抢占优先级、8位子优先级） • NVIC_PRIORITYGROUP_1（1位抢占优先级、7位子优先级） • NVIC_PRIORITYGROUP_2（2位抢占优先级、6位子优先级） • NVIC_PRIORITYGROUP_3（3位抢占优先级、5位子优先级） • NVIC_PRIORITYGROUP_4（4位抢占优先级、4位子优先级） • NVIC_PRIORITYGROUP_5（5位抢占优先级、3位子优先级） • NVIC_PRIORITYGROUP_6（6位抢占优先级、2位子优先级） • NVIC_PRIORITYGROUP_7（7位抢占优先级、1位子优先级）
返回值	无
备注	当priority_group设置为NVIC_PRIORITYGROUP_0时，抢占优先级不可用。

2.2.3.2 hal_nvic_set_priority

表 2-4 hal_nvic_set_priority接口

函数原型	void hal_nvic_set_priority(IRQn_Type IRQn, uint32_t preempt_priority, uint32_t sub_priority)
功能说明	设置指定的中断号对应的中断抢占优先级和子优先级。
输入参数	IRQn：待设置中断的中断号，具体参考*gr551xx.h*中的中断号列表。 preempt_priority：配置的抢占优先级，取值范围0 ～ 127，具体参考*gr551xx.h*中的Cortex_NVIC_Priority_Table表中所述。

	sub_priority：配置的子优先级，取值范围0 ～ 255，具体参考*gr55xx_hal_cortex.c*中的Cortex_NVIC_Priority_Table表。
返回值	无
备注	

2.2.3.3 hal_nvic_enable_irq

表 2-5 hal_nvic_enable_irq接口

函数原型	void hal_nvic_enable_irq(IRQn_Type IRQn)
功能说明	使能指定的中断号对应的中断。
输入参数	IRQn：待使能中断的中断号，取值范围见*gr551xx.h*中的中断号列表。
返回值	无
备注	

2.2.3.4 hal_nvic_disable_irq

表 2-6 hal_nvic_disable_irq接口

函数原型	void hal_nvic_disable_irq(IRQn_Type IRQn)
功能说明	禁止指定的中断号对应的中断。
输入参数	IRQn：待禁止中断的中断号，取值范围见*gr551xx.h*中的中断号列表。
返回值	无
备注	

2.2.3.5 hal_nvic_system_reset

表 2-7 hal_nvic_system_reset接口

函数原型	void hal_nvic_system_reset(void)
功能说明	执行系统复位。
输入参数	无
返回值	无
备注	

2.2.3.6 hal_systick_config

表 2-8 hal_systick_config接口

函数原型	uint32_t hal_systick_config(uint32_t ticks)
功能说明	初始化SysTick计数器，设置计数初值，使能中断并启动计数。
输入参数	ticks：待设置的计数初值，取值范围0x0000_0000 ～ 0xFFFF_FFFF。

返回值	初始化状态，该参数可以是下列值中的任意一个： • 0（初始化成功） • 1（初始化失败）
备注	

2.2.3.7 hal_nvic_get_priority_grouping

表 2-9 hal_nvic_get_priority_grouping接口

函数原型	uint32_t hal_nvic_get_priority_grouping(void)
功能说明	获取优先级分组配置参数。
输入参数	无
返回值	获取的优先级分组配置参数，该参数可以是下列值中的任意一个： • NVIC_PRIORITYGROUP_0（0位抢占优先级、8位子优先级） • NVIC_PRIORITYGROUP_1（1位抢占优先级、7位子优先级） • NVIC_PRIORITYGROUP_2（2位抢占优先级、6位子优先级） • NVIC_PRIORITYGROUP_3（3位抢占优先级、5位子优先级） • NVIC_PRIORITYGROUP_4（4位抢占优先级、4位子优先级） • NVIC_PRIORITYGROUP_5（5位抢占优先级、3位子优先级） • NVIC_PRIORITYGROUP_6（6位抢占优先级、2位子优先级） • NVIC_PRIORITYGROUP_7（7位抢占优先级、1位子优先级）
备注	

2.2.3.8 hal_nvic_get_priority

表 2-10 hal_nvic_get_priority接口

函数原型	void hal_nvic_get_priority(IRQn_Type IRQn, uint32_t priority_group, uint32_t *p_preempt_priority, uint32_t *p_sub_priority)
功能说明	根据优先级分组，获取指定的中断号对应的中断抢占优先级及子优先级。
输入参数	IRQn：待使能中断的中断号，具体参考*gr551xx.h*中的中断号列表。 priority_group：待设置的优先级分组数值，该参数可以下列值中的任意一个： • NVIC_PRIORITYGROUP_0（0位抢占优先级、8位子优先级） • NVIC_PRIORITYGROUP_1（1位抢占优先级、7位子优先级） • NVIC_PRIORITYGROUP_2（2位抢占优先级、6位子优先级） • NVIC_PRIORITYGROUP_3（3位抢占优先级、5位子优先级） • NVIC_PRIORITYGROUP_4（4位抢占优先级、4位子优先级） • NVIC_PRIORITYGROUP_5（5位抢占优先级、3位子优先级） • NVIC_PRIORITYGROUP_6（6位抢占优先级、2位子优先级） • NVIC_PRIORITYGROUP_7（7位抢占优先级、1位子优先级） p_preempt_priority：指向无符号整形变量的指针，该变量用于存储获取的抢占优先级。 p_sub_priority：指向无符号整形变量的指针，该变量用于存储获取的子优先级。
返回值	无

备注	

2.2.3.9 hal_nvic_set_pending_irq

表 2-11 hal_nvic_set_pending_irq接口

函数原型	void hal_nvic_set_pending_irq(IRQn_Type IRQn)
功能说明	挂起指定的中断号对应的中断
输入参数	IRQn：待挂起中断的中断号， 具体参考*gr551xx.h*中的中断号列表。
返回值	无
备注	

2.2.3.10 hal_nvic_get_pending_irq

表 2-12 hal_nvic_get_pending_irq接口

函数原型	uint32_t hal_nvic_get_pending_irq(IRQn_Type IRQn)
功能说明	获取指定的中断号对应的中断的挂起状态。
输入参数	IRQn：待获取中断的中断号，取值范围见*gr551xx.h*中的中断号列表。
返回值	中断挂起状态，该参数可以是下列值中的任意一个： • 0（中断处于未挂起状态） • 1（中断处于挂起状态）
备注	

2.2.3.11 hal_nvic_clear_pending_irq

表 2-13 hal_nvic_clear_pending_irq接口

函数原型	void hal_nvic_clear_pending_irq(IRQn_Type IRQn)
功能说明	清除指定的中断号对应的中断的挂起状态。
输入参数	IRQn：待清除挂起状态的中断的中断号， 取值范围见*gr551xx.h*中的中断号列表。
返回值	无
备注	

2.2.3.12 hal_nvic_get_active

表 2-14 hal_nvic_get_active接口

函数原型	uint32_t hal_nvic_get_active(IRQn_Type IRQn)
功能说明	获取指定的中断号对应的中断的活跃状态。
输入参数	IRQn：待获取中断的中断号，取值范围见*gr551xx.h*中的中断号列表。
返回值	中断活跃状态，该参数可以是下列值中的任意一个： • 0（中断还未被处理）

	• 1（中断正在被处理）
备注	

2.2.3.13 hal_systick_clk_source_config

表 2-15 hal_systick_clk_source_config接口

函数原型	void hal_systick_clk_source_config(uint32_t clk_source)
功能说明	设置SysTick计数器的时钟源。
输入参数	clk_source：指定的时钟源，该参数可以是下列值中的任意一个： • SYSTICK_CLKSOURCE_REFCLK（外部参考时钟） • SYSTICK_CLKSOURCE_HCLK（AHB时钟）
返回值	无
备注	

2.2.3.14 hal_systick_irq_handler

表 2-16 hal_systick_irq_handler接口

函数原型	void hal_systick_irq_handler(void)
功能说明	处理SYSTICK中断请求。
输入参数	无
返回值	无
备注	

2.2.3.15 hal_systick_callback

表 2-17 hal_systick_callback接口

函数原型	void hal_systick_callback(void)
功能说明	SYSTICK中断回调函数。
输入参数	无
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.3 HAL System驱动

2.3.1 System驱动功能

System外设的HAL驱动主要实现了以下功能：

- 配置并使能SysTick。
- 获取驱动的版本信息。

提供的API大部分是weak类型的函数，所以开发者可根据实际应用场景重写相关的API。

2.3.2 如何使用System驱动

System HAL驱动的使用方法如下：

1. 在程序启动时调用hal_init()，完成SysTick的初始化。
2. 调用hal_get_hal_version可以获取驱动的版本信息。

2.3.3 System驱动API描述

System驱动的API主要包括：

表 2-18 System驱动的APIs

API类别	API名称	描述
初始化	hal_init()	初始化System驱动。
	hal_deinit()	反初始化System驱动。
	hal_msp_init()	初始化System驱动相关的时钟、GPIO、中断等配置。
	hal_msp_deinit()	反初始化System驱动相关的时钟、GPIO、中断等配置。
	hal_init_tick()	初始化SysTick。
控制	hal_suspend_tick()	暂停并挂起当前的Tick计数。
	hal_resume_tick()	恢复当前的Tick计数。
状态及错误	hal_get_hal_version()	获取当前的hal驱动版本号。

下面章节将对各API进行详细描述。

2.3.3.1 hal_init

表 2-19 hal_init接口

函数原型	hal_status_t hal_init(void)
功能说明	初始化System驱动，并调用hal_init_tick()完成Tick的初始化配置。
输入参数	无
返回值	HAL状态，该参数可以是下列值中的任意一个： • HAL_OK（正常） • HAL_ERROR（运行错误） • HAL_BUSY（忙） • HAL_TIMEOUT（超时）
备注	程序开始运行时需要调用该API。如果Tick计数源为SysTick，则需重写SysTick_IRQHandler，否则轮询方式的API会被永远轮询，导致程序无法继续执行。

2.3.3.2 hal_deinit

表 2-20 hal_deinit接口

函数原型	hal_status_t hal_deinit(void)
功能说明	反初始化System驱动，禁用Tick。
输入参数	无
返回值	HAL状态。
备注	

2.3.3.3 hal_msp_init

表 2-21 hal_msp_init接口

函数原型	void hal_msp_init(void)
功能说明	初始化System驱动相关的时钟、GPIO、中断等配置。
输入参数	无
返回值	无
备注	该函数为weak类型的空函数，开发者可重写该API以进行相关GPIO、中断等的初始化配置。

2.3.3.4 hal_msp_deinit

表 2-22 hal_msp_deinit接口

函数原型	void hal_msp_deinit(void)
功能说明	反初始化System驱动相关的时钟、GPIO、中断等配置。
输入参数	无
返回值	无
备注	该函数为weak类型的空函数，开发者可重写该API以进行相关GPIO、中断等的反初始化配置。

2.3.3.5 hal_init_tick

表 2-23 hal_init_tick接口

函数原型	hal_status_t hal_init_tick(uint32_t tick_priority)
功能说明	初始化Tick的默认计数源SysTick，设置计数间隔为1 ms，并设置中断优先级为tick_priority。
输入参数	tick_priority：需要设置的SysTick优先级，取值范围为0 ~ 15。
返回值	HAL状态。
备注	该函数为weak类型，且会在hal_init()中被自动调用。如果开发者需要采用其他Tick计数源，可以重写该API。

2.3.3.6 hal_suspend_tick

表 2-24 hal_suspend_tick接口

函数原型	void hal_suspend_tick(void)
功能说明	暂停并挂起Tick计数。
输入参数	无
返回值	无
备注	该函数为weak类型，开发者可根据实际应用重写该API。

2.3.3.7 hal_resume_tick

表 2-25 hal_resume_tick接口

函数原型	void hal_resume_tick(void)
功能说明	恢复Tick计数。
输入参数	无
返回值	无
备注	该函数为weak类型，开发者可根据实际应用重写该API。

2.3.3.8 hal_get_hal_version

表 2-26 hal_get_hal_version接口

函数原型	uint32_t hal_get_hal_version(void)
功能说明	获取HAL驱动的版本号。
输入参数	无
返回值	HAL版本号：格式为0xAaBbCcDd，其中Aa为主版本号、Bb为子版本号1、Cc为子版本号2、Dd为候选发布版本号，例如：v0.0.1.0版本号表示为0x00000100。
备注	

2.4 HAL GPIO通用驱动

2.4.1 GPIO驱动功能

GPIO外设的HAL驱动主要实现以下功能：

- 支持32个IO引脚的输入、输出、引脚复用模式。
- 支持每个引脚的4种中断触发方式：低电平触发、高电平触发、上升沿触发、下降沿触发。
- 支持每个引脚的上拉电阻及下拉电阻使能及禁止。
- 支持中断触发后的回调函数。

2.4.2 如何使用GPIO驱动

GPIO HAL驱动的使用方法如下：

1. 使用hal_gpio_init()配置GPIO引脚。
2. 使用gpio_init_t结构体的“mode”成员配置IO模式。
3. 使用gpio_init_t结构体的“pull”成员激活上拉或下拉电阻。
4. 使用gpio_init_t结构体的“mux”成员启动IO复用功能。
5. 调用hal_nvic_set_priority()配置GPIO的中断优先级，并且调用hal_nvic_enable_irq()使能GPIO中断处理。
6. 使用hal_gpio_read_pin()获取在输入模式中配置的PIN电平。
7. 使用hal_gpio_write_pin()/hal_gpio_toggle_pin()设置/重置在输出模式中配置的PIN电平。

2.4.3 GPIO驱动结构体

2.4.3.1 gpio_init_t

GPIO驱动的初始化结构体gpio_init_t的定义如下：

表 2-27 gpio_init_t结构体

数据域	域段描述	取值
uint32_t pin	要配置的GPIO引脚	该参数的取值可以是下列值的组合： • GPIO_PIN_0（引脚0） • GPIO_PIN_1（引脚1） • GPIO_PIN_2（引脚2） • GPIO_PIN_3（引脚3） • GPIO_PIN_4（引脚4） • GPIO_PIN_5（引脚5） • GPIO_PIN_6（引脚6） • GPIO_PIN_7（引脚7） • GPIO_PIN_8（引脚8） • GPIO_PIN_9（引脚9） • GPIO_PIN_10（引脚10） • GPIO_PIN_11（引脚11） • GPIO_PIN_12（引脚12） • GPIO_PIN_13（引脚13） • GPIO_PIN_14（引脚14） • GPIO_PIN_15（引脚15） • GPIO_PIN_ALL（引脚0 ～15）
uint32_t mode	指定所选引脚的操作模式	该参数的取值可以是下列值中的任意一个： • GPIO_MODE_INPUT（输入模式）

数据域	域段描述	取值
		• GPIO_MODE_OUTPUT（输出模式） • GPIO_MODE_MUX（复用模式） • GPIO_MODE_IT_RISING（上升沿触发检测的外部中断模式） • GPIO_MODE_IT_FALLING（下降沿触发检测的外部中断模式） • GPIO_MODE_IT_HIGH（高电平触发检测的外部中断模式） • GPIO_MODE_IT_LOW（低电平触发检测的外部中断模式）
uint32_t pull	所选引脚上拉或下拉电阻使能或者禁能	该参数的取值可以是下列值中的任意一个： • GPIO_NOPULL（禁用内部上下拉电阻） • GPIO_PULLUP（启用内部上拉电阻） • GPIO_PULLDOWN（启用内部下拉电阻）
uint32_t mux	与引脚相连接的外围设备	参考2.5 HAL GPIO扩展驱动

2.4.4 GPIO驱动API描述

GPIO驱动的API主要包括：

表 2-28 GPIO驱动的APIs

API类别	API名称	描述
初始化	hal_gpio_init()	初始化指定的GPIO引脚。
	hal_gpio_deinit()	反初始化指定的GPIO引脚。
IO操作	hal_gpio_read_pin()	读取引脚的输入电平。
	hal_gpio_write_pin()	设置引脚的输出电平。
	hal_gpio_toggle_pin()	翻转引脚的输出电平。
中断处理及回调函数	hal_gpio_exti_irq_handler()	中断处理函数。
	hal_gpio_exti_callback()	中断回调函数。

下面章节将对各API进行详细描述。

2.4.4.1 hal_gpio_init

表 2-29 hal_gpio_init接口

函数原型	void hal_gpio_init(gpio_regs_t* GPIOx, gpio_init_t *p_gpio_init)
功能说明	根据 gpio_init_t 指定的参数初始化GPIO外设。
输入参数	GPIOx：x可以是0或1，用于确定GR551x家族中被操作的GPIO外设。 p_gpio_init：指向gpio_init_t 结构体变量的指针，该结构体变量包含指定的GPIO引脚的配置信息。
返回值	无
备注	

2.4.4.2 hal_gpio_deinit

表 2-30 hal_gpio_deinit接口

函数原型	void hal_gpio_deinit(gpio_regs_t* GPIOx, uint32_t gpio_pin)
功能说明	将GPIO外设寄存器反初始化为它们的默认重置值。
输入参数	GPIOx：x可以是0或1，用于确定GR551x家族中被操作的GPIO外设。 gpio_pin：指定要写入的引脚位。该参数的取值可以是下列值的组合： • GPIO_PIN_0（引脚0） • GPIO_PIN_1（引脚1） • GPIO_PIN_2（引脚2） • GPIO_PIN_3（引脚3） • GPIO_PIN_4（引脚4） • GPIO_PIN_5（引脚5） • GPIO_PIN_6（引脚6） • GPIO_PIN_7（引脚7） • GPIO_PIN_8（引脚8） • GPIO_PIN_9（引脚9） • GPIO_PIN_10（引脚10） • GPIO_PIN_11（引脚11） • GPIO_PIN_12（引脚12） • GPIO_PIN_13（引脚13） • GPIO_PIN_14（引脚14） • GPIO_PIN_15（引脚15） • GPIO_PIN_ALL（引脚0 ～15）
返回值	无
备注	

2.4.4.3 hal_gpio_read_pin

表 2-31 hal_gpio_read_pin接口

函数原型	gpio_pin_state_t hal_gpio_read_pin(gpio_regs_t* GPIOx, uint16_t gpio_pin)
功能说明	读取指定输入端口的引脚。
输入参数	GPIOx：x可以是0或1，用于确定GR551x家族中被操作的GPIO外设。 gpio_pin：指定要读取的引脚位。该参数的取值是下列值中的任意一个： • GPIO_PIN_0（引脚0） • GPIO_PIN_1（引脚1） • GPIO_PIN_2（引脚2） • GPIO_PIN_3（引脚3） • GPIO_PIN_4（引脚4） • GPIO_PIN_5（引脚5） • GPIO_PIN_6（引脚6） • GPIO_PIN_7（引脚7）

	• GPIO_PIN_8（引脚8） • GPIO_PIN_9（引脚9） • GPIO_PIN_10（引脚10） • GPIO_PIN_11（引脚11） • GPIO_PIN_12（引脚12） • GPIO_PIN_13（引脚13） • GPIO_PIN_14（引脚14） • GPIO_PIN_15（引脚15）
返回值	返回输入端引脚值。
备注	

2.4.4.4 hal_gpio_write_pin

表 2-32 hal_gpio_write_pin接口

函数原型	void hal_gpio_write_pin(gpio_regs_t* GPIOx, uint16_t gpio_pin, gpio_pin_state_t pin_state)
功能说明	设置或清除所选的数据端口位。
输入参数	GPIOx：x可以是0或1，用于确定GR551x家族中被操作的GPIO外设。 gpio_pin：指定要写入的引脚位。该参数的取值可以是下列值的组合： • GPIO_PIN_0（引脚0） • GPIO_PIN_1（引脚1） • GPIO_PIN_2（引脚2） • GPIO_PIN_3（引脚3） • GPIO_PIN_4（引脚4） • GPIO_PIN_5（引脚5） • GPIO_PIN_6（引脚6） • GPIO_PIN_7（引脚7） • GPIO_PIN_8（引脚8） • GPIO_PIN_9（引脚9） • GPIO_PIN_10（引脚10） • GPIO_PIN_11（引脚11） • GPIO_PIN_12（引脚12） • GPIO_PIN_13（引脚13） • GPIO_PIN_14（引脚14） • GPIO_PIN_15（引脚15） • GPIO_PIN_ALL（引脚0 ～ 15） pin_state：指定要写入到所选位的值。这个参数可以下列值中的任意一个： • GPIO_PIN_RESET（低电平） • GPIO_PIN_SET（高电平）
返回值	无
备注	

2.4.4.5 hal_gpio_toggle_pin

表 2-33 hal_gpio_toggle_pin接口

函数原型	void hal_gpio_toggle_pin(gpio_regs_t* GPIOx, uint16_t gpio_pin)
功能说明	切换指定端口引脚。
输入参数	GPIOx：x可以是0或1，用于确定GR551x家族中被操作的GPIO外设。 gpio_pin：指定要切换的引脚位。该参数的取值可以是下列值的组合： • GPIO_PIN_0（引脚0） • GPIO_PIN_1（引脚1） • GPIO_PIN_2（引脚2） • GPIO_PIN_3（引脚3） • GPIO_PIN_4（引脚4） • GPIO_PIN_5（引脚5） • GPIO_PIN_6（引脚6） • GPIO_PIN_7（引脚7） • GPIO_PIN_8（引脚8） • GPIO_PIN_9（引脚9） • GPIO_PIN_10（引脚10） • GPIO_PIN_11（引脚11） • GPIO_PIN_12（引脚12） • GPIO_PIN_13（引脚13） • GPIO_PIN_14（引脚14） • GPIO_PIN_15（引脚15） • GPIO_PIN_ALL（引脚0 ～ 15）
返回值	无
备注	

2.4.4.6 hal_gpio_exti_irq_handler

表 2-34 hal_gpio_exti_irq_handler接口

函数原型	void hal_gpio_exti_irq_handler(gpio_regs_t* GPIOx)
功能说明	处理GPIO中断请求。
输入参数	GPIOx：x可以是0或1，用于确定GR551x家族中被操作的GPIO外设。
返回值	无
备注	

2.4.4.7 hal_gpio_exti_callback

表 2-35 hal_gpio_exti_callback接口

函数原型	void hal_gpio_exti_callback(gpio_regs_t* GPIOx, uint16_t gpio_pin)
功能说明	GPIO回调函数。

输入参数	GPIOx：x可以是0或1，用于确定GR551x家族中被操作的GPIO外设。 gpio_pin：触发本次中断的引脚。该参数的取值可以是下列值的组合： • GPIO_PIN_0（引脚0） • GPIO_PIN_1（引脚1） • GPIO_PIN_2（引脚2） • GPIO_PIN_3（引脚3） • GPIO_PIN_4（引脚4） • GPIO_PIN_5（引脚5） • GPIO_PIN_6（引脚6） • GPIO_PIN_7（引脚7） • GPIO_PIN_8（引脚8） • GPIO_PIN_9（引脚9） • GPIO_PIN_10（引脚10） • GPIO_PIN_11（引脚11） • GPIO_PIN_12（引脚12） • GPIO_PIN_13（引脚13） • GPIO_PIN_14（引脚14） • GPIO_PIN_15（引脚15） • GPIO_PIN_ALL（引脚0 ～ 15）
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.5 HAL GPIO扩展驱动

HAL GPIO Extension驱动主要为GPIO各个引脚的复用模式在不同芯片下的宏定义。

2.5.1 GPIO驱动定义

2.5.1.1 GPIO复用功能选择

- 通用配置项：

表 2-36 GPIO引脚的通用配置项

宏定义	描述
GPIO_PIN_MUX_GPIO	引脚配置为普通GPIO。

说明:

该宏为通用宏，适用于所有引脚，即所有引脚都可以配置的选项。

- GPIO0的引脚0的可配置项：

表 2-37 GPIO0的引脚0的配置项

宏定义	描述
GPIO0_PIN0_MUX_SWD_CLK	GPIO0的引脚0配置为SWD_CLK。
GPIO0_PIN0_MUX_I2C0_SCL	GPIO0的引脚0配置为I2C0_SCL。
GPIO0_PIN0_MUX_I2C1_SCL	GPIO0的引脚0配置为I2C1_SCL。
GPIO0_PIN0_MUX_UART1_RTS	GPIO0的引脚0配置为UART1_RTS。
GPIO0_PIN0_MUX_UART0_TX	GPIO0的引脚0配置为UART0_TX。
GPIO0_PIN0_MUX_UART1_TX	GPIO0的引脚0配置为UART1_TX。
GPIO0_PIN0_MUX_UART0_RTS	GPIO0的引脚0配置为UART0_RTS。

- GPIO0的引脚1的可配置项：

表 2-38 GPIO0的引脚1的配置项

宏定义	描述
GPIO0_PIN1_MUX_SWD_IO	GPIO0的引脚1配置为SWD_IO。
GPIO0_PIN1_MUX_I2C0_SDA	GPIO0的引脚1配置为I2C0_SDA。
GPIO0_PIN1_MUX_I2C1_SDA	GPIO0的引脚1配置为I2C1_SDA。
GPIO0_PIN1_MUX_UART1_CTS	GPIO0的引脚1配置为UART1_CTS。
GPIO0_PIN1_MUX_UART0_RX	GPIO0的引脚1配置为UART0_RX。
GPIO0_PIN1_MUX_UART1_RX	GPIO0的引脚1配置为UART1_RX。
GPIO0_PIN1_MUX_UART0_CTS	GPIO0的引脚1配置为UART0_CTS。

- GPIO0的引脚2的可配置项：

表 2-39 GPIO0的引脚2的配置项

宏定义	描述
GPIO0_PIN2_MUX_UART0_CTS	GPIO0的引脚2配置为UART0_CTS。
GPIO0_PIN2_MUX_SIM_PRESENCE	GPIO0的引脚2配置为SIM_PRESENCE。
GPIO0_PIN2_MUX_SWV	GPIO0的引脚2配置为MUX_SWV。
GPIO0_PIN2_MUX_SPIS_CS_N	GPIO0的引脚2配置为SPIS_CS_N。
GPIO0_PIN2_MUX_I2C0_SDA	GPIO0的引脚2配置为I2C0_SDA。
GPIO0_PIN2_MUX_PWM0_A	GPIO0的引脚2配置为PWM0通道A。
GPIO0_PIN2_MUX_FERP_TRIG	GPIO0的引脚2配置为FERP_TRIG。

- GPIO0的引脚3的可配置项：

表 2-40 GPIO0的引脚3的配置项

宏定义	描述
GPIO0_PIN3_MUX_UART0_TX	GPIO0的引脚3配置为UART0_TX。

宏定义	描述
GPIO0_PIN3_MUX_SIM_RST_N	GPIO0的引脚3配置为SIM_RST_N。
GPIO0_PIN3_MUX_SPIM_CLK	GPIO0的引脚3配置为SPIM_CLK。
GPIO0_PIN3_MUX_SPIS_CLK	GPIO0的引脚3配置为SPIS_CLK。
GPIO0_PIN3_MUX_SPIM_CS1	GPIO0的引脚3配置为SPIM_CS1。
GPIO0_PIN3_MUX_PWM0_B	GPIO0的引脚3配置为PWM0通道B。
GPIO0_PIN3_MUX_COEX_BLE_TX	GPIO0的引脚3配置为COEX_BLE_TX。

- GPIO0的引脚4的可配置项：

表 2-41 GPIO0的引脚4的配置项

宏定义	描述
GPIO0_PIN4_MUX_UART0_RX	GPIO0的引脚4配置为UART0_RX。
GPIO0_PIN4_MUX_SIM_IO	GPIO0的引脚4配置为SIM_IO。
GPIO0_PIN4_MUX_SPIM_MOSI	GPIO0的引脚4配置为SPIM_MOSI。
GPIO0_PIN4_MUX_SPIS_MISO	GPIO0的引脚4配置为SPIS_MISO。
GPIO0_PIN4_MUX_SPIM_CS0	GPIO0的引脚4配置为SPIM_CS0。
GPIO0_PIN4_MUX_PWM0_C	GPIO0的引脚4配置为PWM0通道C。
GPIO0_PIN4_MUX_COEX_BLE_RX	GPIO0的引脚4配置为COEX_BLE_RX。

- GPIO0的引脚5的可配置项：

表 2-42 GPIO0的引脚5的配置项

宏定义	描述
GPIO0_PIN5_MUX_I2C0_SCL	GPIO0的引脚5配置为I2C0_SCL。
GPIO0_PIN5_MUX_UART0_RTS	GPIO0的引脚5配置为UART0_RTS。
GPIO0_PIN5_MUX_SPIS_MOSI	GPIO0的引脚5配置为SPIS_MOSI。
GPIO0_PIN5_MUX_SPIM_MISO	GPIO0的引脚5配置为SPIM_MISO。
GPIO0_PIN5_MUX_SIM_CLK	GPIO0的引脚5配置为SIM_CLK。
GPIO0_PIN5_MUX_COEX_WLAN_TX	GPIO0的引脚5配置为COEX_WLAN_TX。

- GPIO0的引脚6的可配置项：

表 2-43 GPIO0的引脚6的配置项

宏定义	描述
GPIO0_PIN6_MUX_I2C0_SDA	GPIO0的引脚6配置为I2C0_SDA。
GPIO0_PIN6_MUX_I2SM_WS	GPIO0的引脚6配置为I2SM_WS。
GPIO0_PIN6_MUX_I2SS_WS	GPIO0的引脚6配置为I2SS_WS。
GPIO0_PIN6_MUX_SPIM_MOSI	GPIO0的引脚6配置为SPIM_MOSI。

宏定义	描述
GPIO0_PIN6_MUX_SPIM_CS0	GPIO0的引脚6配置为SPIM_CS0。
GPIO0_PIN6_MUX_UART1_RX	GPIO0的引脚6配置为UART1_RX。
GPIO0_PIN6_MUX_COEX_WLAN_RX	GPIO0的引脚6配置为COEX_WLAN_RX。

- GPIO0的引脚7的可配置项：

表 2-44 GPIO0的引脚7的配置项

宏定义	描述
GPIO0_PIN7_MUX_I2SM_TX_SDO	GPIO0的引脚7配置为I2SM_TX_SDO。
GPIO0_PIN7_MUX_I2SS_TX_SDO	GPIO0的引脚7配置为I2SS_TX_SDO。
GPIO0_PIN7_MUX_SPIM_CS1	GPIO0的引脚7配置为SPIM_CS1。
GPIO0_PIN7_MUX_UART1_TX	GPIO0的引脚7配置为UART1_TX。
GPIO0_PIN7_MUX_SPIM_CLK	GPIO0的引脚7配置为SPIM_CLK。
GPIO0_PIN7_MUX_PWM1_A	GPIO0的引脚7配置为PWM1通道A。
GPIO0_PIN7_MUX_COEX_BLE_PROC	GPIO0的引脚7配置为COEX_BLE_PROC。

- GPIO0的引脚8的可配置项：

表 2-45 GPIO0的引脚8的配置项

宏定义	描述
GPIO0_PIN8_MUX_XQSPIM_IO_0	GPIO0的引脚8配置为XQSPIM_IO_0。
GPIO0_PIN8_MUX_QSPIM1_IO_0	GPIO0的引脚8配置为QSPIM1_IO_0。
GPIO0_PIN8_MUX_I2C1_SDA	GPIO0的引脚8配置为I2C1_SDA。
GPIO0_PIN8_MUX_UART1_RX	GPIO0的引脚8配置为UART1_RX。
GPIO0_PIN8_MUX_PWM1_B	GPIO0的引脚8配置为PWM1通道B。

- GPIO0的引脚9的可配置项：

表 2-46 GPIO0的引脚9的配置项

宏定义	描述
GPIO0_PIN9_MUX_XQSPIM_CLK	GPIO0的引脚9配置为XQSPIM_CLK。
GPIO0_PIN9_MUX_QSPIM1_CLK	GPIO0的引脚9配置为QSPIM1_CLK。
GPIO0_PIN9_MUX_I2C1_SCL	GPIO0的引脚9配置为I2C1_SCL。
GPIO0_PIN9_MUX_UART1_TX	GPIO0的引脚9配置为UART1_TX。
GPIO0_PIN9_MUX_PWM1_C	GPIO0的引脚9配置为PWM1通道C。

- GPIO0的引脚10的可配置项：

表 2-47 GPIO0的引脚10的配置项

宏定义	描述
GPIO0_PIN10_MUX_I2SM_RX_SDI	GPIO0的引脚10配置为I2SM_RX_SDI。
GPIO0_PIN10_MUX_I2SS_RX_SDI	GPIO0的引脚10配置为I2SS_RX_SDI。
GPIO0_PIN10_MUX_UART0_TX	GPIO0的引脚10配置为UART0_TX。
GPIO0_PIN10_MUX_I2C0_SCL	GPIO0的引脚10配置为I2C0_SCL。
GPIO0_PIN10_MUX_PWM1_B	GPIO0的引脚10配置为PWM1通道B。
GPIO0_PIN10_MUX_COEX_BLE_TX	GPIO0的引脚10配置为COEX_BLE_TX。

- GPIO0的引脚11的可配置项：

表 2-48 GPIO0的引脚11的配置项

宏定义	描述
GPIO0_PIN11_MUX_I2SM_SCLK	GPIO0的引脚11配置为I2SM_SCLK。
GPIO0_PIN11_MUX_I2SS_SCLK	GPIO0的引脚11配置为I2SS_SCLK。
GPIO0_PIN11_MUX_UART0_RX	GPIO0的引脚11配置为UART0_RX。
GPIO0_PIN11_MUX_I2C0_SDA	GPIO0的引脚11配置为I2C0_SDA。
GPIO0_PIN11_MUX_PWM1_C	GPIO0的引脚11配置为PWM1通道C。

- GPIO0的引脚12的可配置项：

表 2-49 GPIO0的引脚12的配置项

宏定义	描述
GPIO0_PIN12_MUX_XQSPIM_IO_3	GPIO0的引脚12配置为XQSPIM_IO_3。
GPIO0_PIN12_MUX_SPIM_CLK	GPIO0的引脚12配置为SPIM_CLK。
GPIO0_PIN12_MUX_QSPIM1_IO3	GPIO0的引脚12配置为QSPIM1_IO3。
GPIO0_PIN12_MUX_SIM_PRESENCE	GPIO0的引脚12配置为SIM_PRESENCE。
GPIO0_PIN12_MUX_I2SM_WS	GPIO0的引脚12配置为I2SM_WS。
GPIO0_PIN12_MUX_I2SS_WS	GPIO0的引脚12配置为I2SS_WS。
GPIO0_PIN12_MUX_SPIS_CS	GPIO0的引脚12配置为SPIS_CS。

- GPIO0的引脚13的可配置项：

表 2-50 GPIO0的引脚13的配置项

宏定义	描述
GPIO0_PIN13_MUX_XQSPIM_IO_2	GPIO0的引脚13配置为XQSPIM_IO_2。
GPIO0_PIN13_MUX_SPIM_MOSI	GPIO0的引脚13配置为SPIM_MOSI。
GPIO0_PIN13_MUX_QSPIM1_IO_2	GPIO0的引脚13配置为QSPIM1_IO_2。
GPIO0_PIN13_MUX_SIM_RST_N	GPIO0的引脚13配置为SIM_RST_N。

宏定义	描述
GPIO0_PIN13_MUX_I2SM_TX_SDO	GPIO0的引脚13配置为I2SM_TX_SDO。
GPIO0_PIN13_MUX_I2SS_TX_SDO	GPIO0的引脚13配置为I2SS_TX_SDO。
GPIO0_PIN13_MUX_SPIS_CLK	GPIO0的引脚13配置为SPIS_CLK。

- GPIO0的引脚14的可配置项：

表 2-51 GPIO0的引脚14的配置项

宏定义	描述
GPIO0_PIN14_MUX_XQSPIM_IO_1	GPIO0的引脚14配置为XQSPIM_IO_1。
GPIO0_PIN14_MUX_SPIM_MISO	GPIO0的引脚14配置为SPIM_MISO。
GPIO0_PIN14_MUX_QSPIM1_IO1	GPIO0的引脚14配置为QSPIM1_IO1。
GPIO0_PIN14_MUX_SIM_IO	GPIO0的引脚14配置为SIM_IO。
GPIO0_PIN14_MUX_I2SM_RX_SDI	GPIO0的引脚14配置为I2SM_RX_SDI。
GPIO0_PIN14_MUX_I2SS_RX_SDI	GPIO0的引脚14配置为I2SS_RX_SDI。
GPIO0_PIN14_MUX_SPIS_MISO	GPIO0的引脚14配置为SPIS_MISO。

- GPIO0的引脚15的可配置项：

表 2-52 GPIO0的引脚15的配置项

宏定义	描述
GPIO0_PIN15_MUX_XQSPIM_CS_N	GPIO0的引脚15配置为XQSPIM_CS_N。
GPIO0_PIN15_MUX_SPIM_CS0	GPIO0的引脚15配置为SPIM_CS0。
GPIO0_PIN15_MUX_QSPIM1_CS_N	GPIO0的引脚15配置为QSPIM1_CS_N。
GPIO0_PIN15_MUX_SIM_CLK	GPIO0的引脚15配置为SIM_CLK。
GPIO0_PIN15_MUX_I2SM_SCLK	GPIO0的引脚15配置为I2SM_SCLK。
GPIO0_PIN15_MUX_I2SS_SCLK	GPIO0的引脚15配置为I2SS_SCLK。
GPIO0_PIN15_MUX_SPIS_MOSI	GPIO0的引脚15配置为SPIS_MOSI。

- GPIO1的引脚0的可配置项：

表 2-53 GPIO1的引脚0的配置项

宏定义	描述
GPIO1_PIN0_MUX_ISO_SYNC	GPIO1的引脚0配置为ISO_SYNC。
GPIO1_PIN0_MUX_SPIM_MISO	GPIO1的引脚0配置为SPIM_MISO。
GPIO1_PIN0_MUX_QSPIM0_IO_1	GPIO1的引脚0配置为QSPIM0_IO_1。
GPIO1_PIN0_MUX_SPIS_MOSI	GPIO1的引脚0配置为SPIS_MOSI。
GPIO1_PIN0_MUX_SIM_IO	GPIO1的引脚0配置为SIM_IO。
GPIO1_PIN0_MUX_I2SM_RX_SDI	GPIO1的引脚0配置为I2SM_RX_SDI。

宏定义	描述
GPIO1_PIN0_MUX_I2SS_RX_SDI	GPIO1的引脚0配置为I2SS_RX_SDI。

- GPIO1的引脚1的可配置项：

表 2-54 GPIO1的引脚1的配置项

宏定义	描述
GPIO1_PIN1_MUX_SPIM_CS0	GPIO1的引脚1配置为SPIM_CS0。
GPIO1_PIN1_MUX_SPIS_CS	GPIO1的引脚1配置为SPIS_CS。
GPIO1_PIN1_MUX_SIM_CLK	GPIO1的引脚1配置为SIM_CLK。
GPIO1_PIN1_MUX_I2SM_SCLK	GPIO1的引脚1配置为I2SM_SCLK。
GPIO1_PIN1_MUX_I2SS_SCLK	GPIO1的引脚1配置为I2SS_SCLK。
GPIO1_PIN1_MUX_QSPIM0_IO_2	GPIO1的引脚1配置为QSPIM0_IO_2。
GPIO1_PIN1_MUX_COEX_BLE_RX	GPIO1的引脚1配置为COEX_BLE_RX。

- GPIO1的引脚2的可配置项：

表 2-55 GPIO1的引脚2的配置项

宏定义	描述
GPIO1_PIN2_MUX_QSPIM0_CS_N	GPIO1的引脚2配置为QSPIM0_CS_N。
GPIO1_PIN2_MUX_XQSPIM_IO_CS_N	GPIO1的引脚2配置为XQSPIM_IO_CS_N。

- GPIO1的引脚3的可配置项：

表 2-56 GPIO1的引脚3的配置项

宏定义	描述
GPIO1_PIN3_MUX_QSPIM0_IO_3	GPIO1的引脚3配置为QSPIM0_IO_3。
GPIO1_PIN3_MUX_XQSPIM_IO_3	GPIO1的引脚3配置为XQSPIM_IO_3。

- GPIO1的引脚4的可配置项：

表 2-57 GPIO1的引脚4的配置项

宏定义	描述
GPIO1_PIN4_MUX_QSPIM0_CLK	GPIO1的引脚4配置为QSPIM0_CLK。
GPIO1_PIN4_MUX_XQSPIM_CLK	GPIO1的引脚4配置为XQSPIM_CLK。

- GPIO1的引脚5的可配置项：

表 2-58 GPIO1的引脚5的配置项

宏定义	描述
GPIO1_PIN5_MUX_QSPIM0_IO_2	GPIO1的引脚5配置为QSPIM0_IO_2。
GPIO1_PIN5_MUX_XQSPIM_IO_2	GPIO1的引脚5配置为XQSPIM_IO_2。

- GPIO1的引脚6的可配置项：

表 2-59 GPIO1的引脚6的配置项

宏定义	描述
GPIO1_PIN6_MUX_QSPIM0_IO_1	GPIO1的引脚6配置为QSPIM0_IO_1。
GPIO1_PIN6_MUX_XQSPIM_IO_1	GPIO1的引脚6配置为XQSPIM_IO_1。

- GPIO1的引脚7的可配置项：

表 2-60 GPIO1的引脚7的配置项

宏定义	描述
GPIO1_PIN7_MUX_QSPIM0_IO_0	GPIO1的引脚7配置为QSPIM0_IO_0。
GPIO1_PIN7_MUX_XQSPIM_IO_0	GPIO1的引脚7配置为XQSPIM_IO_0。

- GPIO1的引脚8的可配置项：

表 2-61 GPIO1的引脚8的配置项

宏定义	描述
GPIO1_PIN8_MUX_SPIM_CLK	GPIO1的引脚8配置为SPIM_CLK。
GPIO1_PIN8_MUX_SPIS_CLK	GPIO1的引脚8配置为SPIS_CLK。
GPIO1_PIN8_MUX_SIM_PRESENCE	GPIO1的引脚8配置为SIM_PRESENCE。
GPIO1_PIN8_MUX_I2SM_WS	GPIO1的引脚8配置为I2SM_WS。
GPIO1_PIN8_MUX_I2SS_WS	GPIO1的引脚8配置为I2SS_WS。
GPIO1_PIN8_MUX_QSPIM0_CLK	GPIO1的引脚8配置为QSPIM0_CLK。
GPIO1_PIN8_MUX_COEX_WLAN_TX	GPIO1的引脚8配置为COEX_WLAN_TX。

- GPIO1的引脚9的可配置项：

表 2-62 GPIO1的引脚9的配置项

宏定义	描述
GPIO1_PIN9_MUX_SPIM_MOSI	GPIO1的引脚9配置为MUX_SPIM_MOSI。
GPIO1_PIN9_MUX_SPIS_MISO	GPIO1的引脚9配置为SPIS_MISO。
GPIO1_PIN9_MUX_SIM_RST_N	GPIO1的引脚9配置为SIM_RST_N。
GPIO1_PIN9_MUX_I2SM_TX_SD0	GPIO1的引脚9配置为I2SM_TX_SD0。
GPIO1_PIN9_MUX_I2SS_TX_SD0	GPIO1的引脚9配置为I2SS_TX_SD0。
GPIO1_PIN9_MUX_QSPIM0_IO_0	GPIO1的引脚9配置为QSPIM0_IO_0。
GPIO1_PIN9_MUX_COEX_BLE_PROC	GPIO1的引脚9配置为COEX_BLE_PROC。

- GPIO1的引脚10的可配置项：

表 2-63 GPIO1的引脚10的配置项

宏定义	描述
GPIO1_PIN10_MUX_I2C1_SDA	GPIO1的引脚10配置为I2C1_SDA。
GPIO1_PIN10_MUX_UART1_RX	GPIO1的引脚10配置为UART1_RX。
GPIO1_PIN10_MUX_I2C0_SDA	GPIO1的引脚10配置为I2C0_SDA。
GPIO1_PIN10_MUX_PWM0_C	GPIO1的引脚10配置为PWM0通道C。
GPIO1_PIN10_MUX_PWM1_C	GPIO1的引脚10配置为PWM1通道C。
GPIO1_PIN10_MUX_UART0_RX	GPIO1的引脚10配置为UART0_RX。

- GPIO1的引脚11的可配置项：

表 2-64 GPIO1的引脚11的配置项

宏定义	描述
GPIO1_PIN11_MUX_UART1_RTS	GPIO1的引脚11配置为UART1_RTS。
GPIO1_PIN11_MUX_UART0_RTS	GPIO1的引脚11配置为UART0_RTS。

- GPIO1的引脚12的可配置项：

表 2-65 GPIO1的引脚12的配置项

宏定义	描述
GPIO1_PIN12_MUX_UART1_CTS	GPIO1的引脚12配置为UART1_CTS。
GPIO1_PIN12_MUX_UART0_CTS	GPIO1的引脚12配置为UART0_CTS。

- GPIO1的引脚14的可配置项：

表 2-66 GPIO1的引脚14的配置项

宏定义	描述
GPIO1_PIN14_MUX_I2C1_SCL	GPIO1的引脚14配置为I2C1_SCL。
GPIO1_PIN14_MUX_UART1_TX	GPIO1的引脚14配置为UART1_TX。
GPIO1_PIN14_MUX_I2C0_SCL	GPIO1的引脚14配置为I2C0_SCL。
GPIO1_PIN14_MUX_PWM0_B	GPIO1的引脚14配置为PWM0通道B。
GPIO1_PIN14_MUX_PWM1_B	GPIO1的引脚14配置为PWM1通道B。
GPIO1_PIN14_MUX_UART0_TX	GPIO1的引脚14配置为UART0_TX。
GPIO1_PIN14_MUX_COEX_BLE_TX	GPIO1的引脚14配置为COEX_BLE_TX。

- GPIO1的引脚15的可配置项：

表 2-67 GPIO1的引脚15的配置项

宏定义	描述
GPIO1_PIN15_MUX_SPIM_CS1	GPIO1的引脚15配置为SPIM_CS1。
GPIO1_PIN15_MUX_PWM0_A	GPIO1的引脚15配置为PWM0通道A。

宏定义	描述
GPIO1_PIN15_MUX_PWM1_A	GPIO1的引脚15配置为PWM1通道A。
GPIO1_PIN15_MUX_QSPIM0_IO_3	GPIO1的引脚15配置为QSPIM0_IO_3。
GPIO1_PIN15_MUX_COEX_WLAN_TX	GPIO1的引脚15配置为COEX_WLAN_TX。

2.6 HAL AON GPIO通用驱动

2.6.1 AON GPIO驱动功能

AON GPIO（Always-on GPIO）外设的HAL驱动主要实现了以下功能：

- 支持8个IO引脚的输入、输出模式。
- 每个引脚支持4种中断触发方式：低电平触发、高电平触发、上升沿触发、下降沿触发。
- 支持深度睡眠模式下的引脚电平保持。
- AON_GPIO_5可输出2 MHz时钟信号。
- 支持中断触发后的回调函数。

2.6.2 如何使用AON GPIO驱动

AON GPIO HAL驱动的使用方法如下：

1. 使用hal_aon_gpio_init()配置GPIO引脚。
    - 使用aon_gpio_init_t结构体的“mode”成员配置IO模式。
    - 使用aon_gpio_init_t结构体的“pull”成员激活上拉或下拉电阻。
    - 使用aon_gpio_init_t结构体的“mux”成员启动IO复用功能。
    - 调用hal_nvic_enable_irq()使能AON_GPIO中断处理。
2. 如果需要使用AON_GPIO的输入中断，则需调用hal_nvic_set_priority()配置AON_GPIO的中断优先级，并且调用hal_nvic_enable_irq()使能AON_GPIO中断处理。
3. 使用hal_aon_gpio_read_pin()获取在输入模式中配置的PIN电平。
4. 使用hal_aon_gpio_write_pin()/hal_aon_gpio_toggle_pin()设置/重置在输出模式中配置的PIN电平。

2.6.3 AON GPIO驱动的结构体

2.6.3.1 aon_gpio_init_t

AON GPIO驱动的结构体aon_gpio_init_t的定义如下：

表 2-68 aon_gpio_init_t结构体

数据域	域段描述	取值
uint32_t pin	指定要配置的AON GPIO引脚。	该参数的取值可以是下列值的组合： • AON_GPIO_PIN_0 • AON_GPIO_PIN_1 • AON_GPIO_PIN_2 • AON_GPIO_PIN_3 • AON_GPIO_PIN_4 • AON_GPIO_PIN_5 • AON_GPIO_PIN_6 • AON_GPIO_PIN_7 • AON_GPIO_PIN_ALL
uint32_t mode	指定所选引脚的操作模式。	该参数的取值可以是下列值中的任意一个： • AON_GPIO_MODE_INPUT （输入模式） • AON_GPIO_MODE_OUTPUT（输出模式） • AON_GPIO_MODE_IT_RISING（上升沿触发检测的外部中断模式） • AON_GPIO_MODE_IT_FALLING（下降沿触发检测的外部中断模式） • AON_GPIO_MODE_IT_HIGH（高电平触发检测的外部中断模式） • AON_GPIO_MODE_IT_LOW（低电平触发检测的外部中断模式）
uint32_t pull	指定所选引脚上拉或下拉激活。	该参数的取值可以是下列值中的任意一个： • AON_GPIO_NOPULL（没有上拉或下拉激活） • AON_GPIO_PULLUP（上拉激活） • AON_GPIO_PULLDOWN（下拉激活）
uint32_t mux	与引脚相连接的外围设备	参考2.7 HAL AON GPIO扩展驱动

2.6.4 AON GPIO驱动API描述

AON GPIO驱动的API主要包括：

表 2-69 AON GPIO驱动的APIs

API类别	API名称	描述
初始化	hal_aon_gpio_init()	初始化指定的AON GPIO引脚。
	hal_aon_gpio_deinit()	反初始化指定的AON GPIO引脚。
IO操作	hal_aon_gpio_read_pin()	读取引脚的输入电平。
	hal_aon_gpio_write_pin()	设置引脚的输出电平。
	hal_aon_gpio_toggle_pin()	翻转引脚的输出电平。
中断处理及回调函数	hal_aon_gpio_irq_handler()	中断处理函数。

API类别	API名称	描述
	hal_aon_gpio_callback()	中断回调函数。

下面章节将对各API进行详细描述。

2.6.4.1 hal_aon_gpio_init

表 2-70 hal_aon_gpio_init接口

函数原型	void hal_aon_gpio_init(aon_gpio_init_t *p_aon_gpio_init)
功能说明	根据aon_gpio_init_t指定参数初始化AON GPIO外设。
输入参数	p_aon_gpio_init：指向hal_aon_gpio_init结构体变量的指针，该结构体主要包含AON GPIO外设实例的配置信息。
返回值	无
备注	

2.6.4.2 hal_aon_gpio_deinit

表 2-71 hal_aon_gpio_deinit接口

函数原型	void hal_aon_gpio_deinit(uint32_t aon_gpio_pin)
功能说明	将AON GPIO外设寄存器反初始化为它们的默认重置值。
输入参数	aon_gpio_pin：指定要写入的端口位。该参数可以是下列值的组合： • AON_GPIO_PIN_0 • AON_GPIO_PIN_1 • AON_GPIO_PIN_2 • AON_GPIO_PIN_3 • AON_GPIO_PIN_4 • AON_GPIO_PIN_5 • AON_GPIO_PIN_6 • AON_GPIO_PIN_7 • AON_GPIO_PIN_ALL
返回值	无
备注	

2.6.4.3 hal_aon_gpio_read_pin

表 2-72 hal_aon_gpio_read_pin接口

函数原型	aon_gpio_pin_state_t hal_aon_gpio_read_pin(uint16_t aon_gpio_pin)
功能说明	读取指定的引脚的输入电平。
输入参数	aon_gpio_pin：指定待读取的引脚。该参数可以是下列值中的任意一个： • AON_GPIO_PIN_0 • AON_GPIO_PIN_1

	• AON_GPIO_PIN_2 • AON_GPIO_PIN_3 • AON_GPIO_PIN_4 • AON_GPIO_PIN_5 • AON_GPIO_PIN_6 • AON_GPIO_PIN_7
返回值	输入引脚的电平，只能是下面之中的一个值： • AON_GPIO_PIN_RESET（低电平） • AON_GPIO_PIN_SET（高电平）
备注	

2.6.4.4 hal_aon_gpio_write_pin

表 2-73 hal_aon_gpio_write_pin接口

函数原型	void hal_aon_gpio_write_pin(uint16_t aon_gpio_pin, aon_gpio_pin_state_t pin_state)
功能说明	设置指定引脚的输出电平。
输入参数	aon_gpio_pin：指定需要设置的引脚。该参数可以是下列值的组合： • AON_GPIO_PIN_0 • AON_GPIO_PIN_1 • AON_GPIO_PIN_2 • AON_GPIO_PIN_3 • AON_GPIO_PIN_4 • AON_GPIO_PIN_5 • AON_GPIO_PIN_6 • AON_GPIO_PIN_7 • AON_GPIO_PIN_ALL pin_state：指定设置的电平。该参数可以是下列值中的任意一个： • AON_GPIO_PIN_RESET（低电平） • AON_GPIO_PIN_SET（高电平）
返回值	无
备注	

2.6.4.5 hal_aon_gpio_toggle_pin

表 2-74 hal_aon_gpio_toggle_pin接口

函数原型	void hal_aon_gpio_toggle_pin(uint16_t aon_gpio_pin)
功能说明	翻转指定引脚的电平。
输入参数	aon_gpio_pin：指定要翻转的引脚。该参数可以是下列值的组合： • AON_GPIO_PIN_0 • AON_GPIO_PIN_1 • AON_GPIO_PIN_2 • AON_GPIO_PIN_3

	• AON_GPIO_PIN_4 • AON_GPIO_PIN_5 • AON_GPIO_PIN_6 • AON_GPIO_PIN_7 • AON_GPIO_PIN_ALL
返回值	无
备注	

2.6.4.6 hal_aon_gpio_irq_handler

表 2-75 hal_aon_gpio_irq_handler接口

函数原型	void hal_aon_gpio_irq_handler(void)
功能说明	处理AON GPIO中断请求。
输入参数	无
返回值	无
备注	

2.6.4.7 hal_aon_gpio_callback

表 2-76 hal_aon_gpio_callback接口

函数原型	void hal_aon_gpio_callback(uint16_t aon_gpio_pin)
功能说明	AON GPIO中断回调函数。
输入参数	aon_gpio_pin：触发本次中断的引脚。该参数可以是下列值的组合： • AON_GPIO_PIN_0 • AON_GPIO_PIN_1 • AON_GPIO_PIN_2 • AON_GPIO_PIN_3 • AON_GPIO_PIN_4 • AON_GPIO_PIN_5 • AON_GPIO_PIN_6 • AON_GPIO_PIN_7 • AON_GPIO_PIN_ALL
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.7 HAL AON GPIO扩展驱动

HAL AON GPIO Extension驱动主要为AON GPIO各个引脚的复用模式在不同芯片下的宏定义。

2.7.1 AON GPIO驱动定义

2.7.1.1 AON GPIO复用功能选择

- 通用配置项：

表 2-77 AON GPIO引脚的通用配置项

宏定义	描述
AON_GPIO_PIN_MUX_GPIO	引脚配置为普通GPIO。

说明:

该宏为通用宏，适用于所有引脚，即所有引脚都可以配置的选项。

- AON GPIO的引脚1的可配置项：

表 2-78 AON GPIO的引脚1的配置项

宏定义	描述
AON_GPIO_PIN1_MUX_QSPIM0_CS_N	AON GPIO的引脚1配置为QSPIM0_CS_N。
AON_GPIO_PIN1_MUX_COEX_BLE_TX	AON GPIO的引脚1配置为COEX_BLE_TX。

- AON GPIO的引脚2的可配置项：

表 2-79 AON GPIO的引脚2的配置项

宏定义	描述
AON_GPIO_PIN2_MUX_SIM_PRESENCE	AON GPIO的引脚2配置为SIM_PRESENCE。
AON_GPIO_PIN2_MUX_QSPIM1_CS_N	AON GPIO的引脚2配置为QSPIM1_CS_N。
AON_GPIO_PIN2_MUX_I2S_WS	AON GPIO的引脚2配置为I2S_WS。
AON_GPIO_PIN2_MUX_I2S_S_WS	AON GPIO的引脚2配置为I2S_S_WS。
AON_GPIO_PIN2_MUX_PWM0_C	AON GPIO的引脚2配置为PWM0的通道C。
AON_GPIO_PIN2_MUX_COEX_BLE_PROC	AON GPIO的引脚2配置为COEX_BLE_PROC。

- AON GPIO的引脚3的可配置项：

表 2-80 AON GPIO的引脚3的配置项

宏定义	描述
AON_GPIO_PIN3_MUX_SIM_RST_N	AON GPIO的引脚3配置为SIM_RST_N。
AON_GPIO_PIN3_MUX_QSPIM1_IO_0	AON GPIO的引脚3配置为QSPIM1_IO_0。
AON_GPIO_PIN3_MUX_I2S_TX_SDO	AON GPIO的引脚3配置为I2S_TX_SDO。
AON_GPIO_PIN3_MUX_I2S_S_TX_SDO	AON GPIO的引脚3配置为I2S_S_TX_SDO。
AON_GPIO_PIN3_MUX_PWM1_A	AON GPIO的引脚3配置为PWM1的通道A。
AON_GPIO_PIN3_MUX_COEX_WLAN_RX	AON GPIO的引脚3配置为COEX_WLAN_RX。

- AON GPIO的引脚4的可配置项：

表 2-81 AON GPIO的引脚4的配置项

宏定义	描述
AON_GPIO_PIN4_MUX_SIM_IO	AON GPIO的引脚4配置为SIM_IO。
AON_GPIO_PIN4_MUX_QSPIM1_IO_1	AON GPIO的引脚4配置为QSPIM1_IO_1。
AON_GPIO_PIN4_MUX_I2S_RX_SDI	AON GPIO的引脚4配置为I2S_RX_SDI。
AON_GPIO_PIN4_MUX_I2S_S_RX_SDI	AON GPIO的引脚4配置为I2S_S_RX_SDI。
AON_GPIO_PIN4_MUX_PWM1_B	AON GPIO的引脚4配置为PWM1的通道B。
AON_GPIO_PIN4_MUX_COEX_BLE_RX	AON GPIO的引脚4配置为COEX_BLE_RX。

- AON GPIO的引脚5的可配置项：

表 2-82 AON GPIO的引脚5的配置项

宏定义	描述
AON_GPIO_PIN5_MUX_SIM_CLK	AON GPIO的引脚5配置为SIM_CLK。
AON_GPIO_PIN5_MUX_QSPIM1_CLK	AON GPIO的引脚5配置为QSPIM1_CLK。
AON_GPIO_PIN5_MUX_I2S_SCLK	AON GPIO的引脚5配置为I2S_SCLK。
AON_GPIO_PIN5_MUX_I2S_S_SCLK	AON GPIO的引脚5配置为I2S_S_SCLK。
AON_GPIO_PIN5_MUX_PWM1_C	AON GPIO的引脚5配置为PWM1的通道C。
AON_GPIO_PIN5_MUX_COEX_WLAN_TX	AON GPIO的引脚5配置为COEX_WLAN_TX。

2.8 HAL MSIO通用驱动

2.8.1 MSIO驱动功能

MSIO（MSIO）外设的HAL驱动主要实现了以下功能：

- 支持5个IO引脚的输入、输出模式。
- 可配置为ADC输入。

2.8.2 如何使用MSIO驱动

MSIO HAL驱动的使用方法如下：

1. 使用hal_msio_init()配置MSIO引脚。
    - 使用msio_init_t结构体的“direction”成员配置IO输入输出方向。
    - 使用msio_init_t结构体的“mode”成员配置IO模式。
    - 使用msio_init_t结构体的“pull”成员激活上拉或下拉电阻。
    - 使用msio_init_t结构体的“mux”成员启动IO复用功能。

2. 如果需要使用MSIO的ADC输入功能，则需配置IO模式为MSIO_MODE_ANALOG。

3. 使用hal_msio_read_pin()获取在输入模式中配置的PIN电平。

4. 使用hal_msio_write_pin()/hal_msio_toggle_pin()设置/重置在输出模式中配置的PIN电平。

2.8.3 MSIO驱动的结构体

2.8.3.1 msio_init_t

MSIO 驱动的结构体msio_init_t的定义如下：

表 2-83 msio_init_t结构体

数据域	域段描述	取值
uint32_t pin	指定要配置的MSIO引脚。	该参数的取值可以是下列值的组合： • MSIO_PIN_0 • MSIO_PIN_1 • MSIO_PIN_2 • MSIO_PIN_3 • MSIO_PIN_4 • MSIO_PIN_ALL
uint32_t direction	指定所选引脚的输入输出方向。	该参数的取值可以是下列值中的任意一个： • MSIO_DIRECTION_NONE（禁止输入和输出） • MSIO_DIRECTION_INPUT（使能输入） • MSIO_DIRECTION_OUTPUT（使能输出） • MSIO_DIRECTION_INOUT（使能输入和输出）
uint32_t mode	指定所选引脚的操作模式。	该参数的取值可以是下列值中的任意一个： • MSIO_MODE_ANALOG（模拟模式） • MSIO_MODE_DIGITAL（数字模式）
uint32_t pull	指定所选引脚上拉或下拉激活。	该参数的取值可以是下列值中的任意一个： • MSIO_NOPULL（没有上拉或下拉激活） • MSIO_PULLUP（上拉激活） • MSIO_PULLDOWN（下拉激活）
uint32_t mux	与引脚相连接的外围设备	参考2.9 HAL MSIO扩展驱动

2.8.4 MSIO驱动API描述

MSIO驱动的API主要包括：

表 2-84 MSIO驱动的APIs

API类别	API名称	描述
初始化	hal_msio_init()	初始化指定的MSIO引脚。
	hal_msio_deinit()	反初始化指定的MSIO引脚。

API类别	API名称	描述
IO操作	hal_msio_read_pin()	读取引脚的输入电平。
	hal_msio_write_pin()	设置引脚的输出电平。
	hal_msio_toggle_pin()	翻转引脚的输出电平。

下面章节将对各API进行详细描述。

2.8.4.1 hal_msio_init

表 2-85 hal_msio_init接口

函数原型	void hal_msio_init(msio_init_t *p_msio_init)
功能说明	根据msio_init_t指定参数初始化MSIO外设。
输入参数	p_msio_init：指向msio_init_t结构体变量的指针，该结构体主要包含MSIO外设实例的配置信息。
返回值	无
备注	

2.8.4.2 hal_msio_deinit

表 2-86 hal_msio_deinit接口

函数原型	void hal_msio_deinit(uint32_t msio_pin)
功能说明	将MSIO外设寄存器反初始化为它们的默认重置值。
输入参数	msio_pin：指定要写入的端口位。该参数可以是下列值的组合： • MSIO_PIN_0 • MSIO_PIN_1 • MSIO_PIN_2 • MSIO_PIN_3 • MSIO_PIN_4 • MSIO_PIN_ALL
返回值	无
备注	

2.8.4.3 hal_msio_read_pin

表 2-87 hal_msio_read_pin接口

函数原型	msio_pin_state_t hal_msio_read_pin(uint16_t msio_pin)
功能说明	读取指定的引脚的输入电平。
输入参数	msio_pin：指定待读取的引脚。该参数可以是下列值中的任意一个： • MSIO_PIN_0 • MSIO_PIN_1 • MSIO_PIN_2

	• MSIO_PIN_3 • MSIO_PIN_4
返回值	输入引脚的电平，只能是下面之中的一个值： • MSIO_PIN_RESET（低电平） • MSIO_PIN_SET（高电平）
备注	

2.8.4.4 hal_msio_write_pin

表 2-88 hal_msio_write_pin接口

函数原型	void hal_msio_write_pin(uint16_t msio_pin, msio_pin_state_t pin_state)
功能说明	设置指定引脚的输出电平。
输入参数	msio_pin：指定需要设置的引脚。该参数可以是下列值的组合： • MSIO_PIN_0 • MSIO_PIN_1 • MSIO_PIN_2 • MSIO_PIN_3 • MSIO_PIN_4 • MSIO_PIN_ALL pin_state：指定设置的电平。该参数可以是下列值中的任意一个： • MSIO_PIN_RESET（低电平） • MSIO_PIN_SET（高电平）
返回值	无
备注	

2.8.4.5 hal_msio_toggle_pin

表 2-89 hal_msio_toggle_pin接口

函数原型	void hal_msio_toggle_pin(uint16_t msio_pin)
功能说明	翻转指定引脚的电平。
输入参数	msio_pin：指定要翻转的引脚。该参数可以是下列值的组合： • MSIO_PIN_0 • MSIO_PIN_1 • MSIO_PIN_2 • MSIO_PIN_3 • MSIO_PIN_4 • MSIO_PIN_ALL
返回值	无
备注	

2.9 HAL MSIO扩展驱动

HAL MSIO Extension驱动主要为MSIO各个引脚的复用模式在不同芯片下的宏定义。

2.9.1 MSIO驱动定义

2.9.1.1 MSIO复用功能选择

- 通用配置项：

表 2-90 MSIO引脚的通用配置项

宏定义	描述
MSIO_PIN_MUX_GPIO	引脚配置为普通GPIO。

说明:

该宏为通用宏，适用于所有引脚，即所有引脚都可以配置的选项。

- MSIO的引脚0的可配置项：

表 2-91 MSIO的引脚0的配置项

宏定义	描述
MSIO_PIN0_MUX_PWM0_A	MSIO的引脚0配置为PWM0的通道A。
MSIO_PIN0_MUX_UART0_TX	MSIO的引脚0配置为UART0_TX。
MSIO_PIN0_MUX_UART1_TX	MSIO的引脚0配置为UART1_TX。
MSIO_PIN0_MUX_I2C0_SCL	MSIO的引脚0配置为I2C0_SCL。
MSIO_PIN0_MUX_I2C1_SCL	MSIO的引脚0配置为I2C1_SCL。

- MSIO的引脚1的可配置项：

表 2-92 MSIO的引脚1的配置项

宏定义	描述
MSIO_PIN1_MUX_PWM0_B	MSIO的引脚1配置为PWM0的通道B。
MSIO_PIN1_MUX_UART0_RX	MSIO的引脚1配置为UART0_RX。
MSIO_PIN1_MUX_UART1_RX	MSIO的引脚1配置为UART1_RX。
MSIO_PIN1_MUX_I2C0_SDA	MSIO的引脚1配置为I2C0_SDA。
MSIO_PIN1_MUX_I2C1_SDA	MSIO的引脚1配置为I2C1_SDA。

- MSIO的引脚2的可配置项：

表 2-93 MSIO的引脚2的配置项

宏定义	描述
MSIO_PIN2_MUX_PWM0_C	MSIO的引脚2配置为PWM0的通道C。

- MSIO的引脚3的可配置项：

表 2-94 MSIO的引脚3的配置项

宏定义	描述
MSIO_PIN3_MUX_PWM1_A	MSIO的引脚3配置为PWM1的通道A。
MSIO_PIN3_MUX_UART0_RTS	MSIO的引脚3配置为UART0_RTS。
MSIO_PIN3_MUX_UART1_RTS	MSIO的引脚3配置为UART1_RTS。
MSIO_PIN3_MUX_I2C0_SCL	MSIO的引脚3配置为I2C0_SCL。
MSIO_PIN3_MUX_I2C1_SCL	MSIO的引脚3配置为I2C1_SCL。

- MSIO的引脚4的可配置项：

表 2-95 MSIO的引脚4的配置项

宏定义	描述
MSIO_PIN4_MUX_PWM1_B	MSIO的引脚4配置为PWM1的通道B。
MSIO_PIN4_MUX_UART0_CTS	MSIO的引脚4配置为UART0_CTS。
MSIO_PIN4_MUX_UART1_CTS	MSIO的引脚4配置为UART1_CTS。
MSIO_PIN4_MUX_I2C0_SDA	MSIO的引脚4配置为I2C0_SDA。
MSIO_PIN4_MUX_I2C1_SDA	MSIO的引脚4配置为I2C1_SDA。

2.10 HAL ADC通用驱动

2.10.1 ADC驱动功能

ADC外设的HAL驱动主要实现了以下功能：

- 支持单端和差分两种输入模式
- 支持最高1 Msps的采样速率
- 支持1 MHz、1.6 MHz、2 MHz、4 MHz、8 MHz、16 MHz六种不同速率时钟
- 采样分辨率13 bits
- 支持0.85 V、1.28 V、1.6 V内部参考电压调节
- 支持外部参考电压输入
- 支持DMA方式采样

2.10.2 如何使用ADC驱动

ADC HAL驱动使用方法如下：

1. 声明一个adc_handle_t句柄结构体，例如：adc_handle_t adc_handle。
2. 重写hal_adc_msp_init() API以初始化ADC底层资源：

(1) 配置ADC引脚：调用hal_msio_init()配置MSIO模式为MSIO_MODE_ANALOG（模拟方式），并指定需要配置为模拟IO的MSIO引脚。

(2) 如果开发者要使用DMA流程（hal_adc_start_dma()），则需配置DMA：

- ADC通道只需要声明一条DMA通道。
- 为ADC通道声明DMA句柄结构体，例如：dma_handle_t hdma。
- 配置DMA句柄中的参数，例如数据交互通道。
- 将初始的DMA句柄关联到adc_handle变量中的p_dma指针。
- 配置DMA的中断优先级、使能DMA的NVIC中断。

3. 配置adc_handle句柄的init结构体中的参考电压等参数。

4. 调用hal_adc_init() API初始化ADC寄存器。

说明:

如果参考电压选择外部参考源（ADC_REF_SRC_IOx，x可以是0 ～ 3），参考源输入电压取值范围是0.7 ～ 1.9 V。

2.10.3 ADC驱动结构的结构体

2.10.3.1 adc_init_t

ADC驱动的初始化结构体adc_init_t的定义如下：

表 2-96 adc_init_t结构体

数据域	域段描述	取值
uint32_t channel_p	通道P的输入源	该参数的取值可以是下列值中的任意一个： • ADC_INPUT_SRC_IO0（MSIO0输入） • ADC_INPUT_SRC_IO1（MSIO1输入） • ADC_INPUT_SRC_IO2（MSIO2输入） • ADC_INPUT_SRC_IO3（MSIO3输入） • ADC_INPUT_SRC_IO4（MSIO4输入） • ADC_INPUT_SRC_TMP（温度传感器输入） • ADC_INPUT_SRC_BAT（电池电压输入） • ADC_INPUT_SRC_REF（参考电压输入）
uint32_t channel_n	通道N的输入源	该参数的取值可以是下列值中的任意一个： • ADC_INPUT_SRC_IO0（MSIO0输入） • ADC_INPUT_SRC_IO1（MSIO1输入） • ADC_INPUT_SRC_IO2（MSIO2输入） • ADC_INPUT_SRC_IO3（MSIO3输入）

数据域	域段描述	取值
		• ADC_INPUT_SRC_IO4（MSIO4输入） • ADC_INPUT_SRC_TMP（温度传感器输入） • ADC_INPUT_SRC_BAT（电池电压输入） • ADC_INPUT_SRC_REF（参考电压输入）
uint32_t input_mode	采样方式	该参数的取值可以是下列值中的任意一个： • ADC_INPUT_SINGLE（单端输入采样） • ADC_INPUT_DIFFERENTIAL（差分输入采样）
uin32_t ref_source	参考源类型	该参数的取值可以是下列值中的任意一个： • ADC_REF_SRC_BUF_INT（内部Buffered参考源） • ADC_REF_SRC_IO0（MSIO0输入电压） • ADC_REF_SRC_IO1（MSIO1输入电压） • ADC_REF_SRC_IO2（MSIO2输入电压） • ADC_REF_SRC_IO3（MSIO3输入电压）
uin32_t ref_value	内部参考电压	该参数的取值可以是下列值中的任意一个： • ADC_REF_VALUE_0P8（0.85 V） • ADC_REF_VALUE_1P2（1.28 V） • ADC_REF_VALUE_1P6（1.6 V） **说明：** 外部输入信号的量程是0 ~ 2*ref_value，用户可按照实际的使用场景进行参数配置。
uin32_t clock	采样时钟	该参数的取值可以是下列值中的任意一个： • ADC_CLK_16M（16 MHz时钟） • ADC_CLK_1P6M（1.6 MHz时钟） • ADC_CLK_8M（8 MHz时钟） • ADC_CLK_4M（4 MHz时钟） • ADC_CLK_2M（2 MHz时钟） • ADC_CLK_1M（1 MHz时钟）

2.10.3.2 adc_handle_t

ADC驱动的句柄结构体adc_handle_t的定义如下：

表 2-97 adc_handle_t结构体

数据域	域段描述	取值
adc_init_t init	初始化结构体（参考adc_init_t结构体）。	N/A。
uint16_t *p_buffer	指向数据接收缓冲区的指针（驱动负责管理，无需开发者初始化）。	N/A。
__IO uint32_t buff_size	接收缓冲区大小（驱动负责管理，无需开发者初始化）。	N/A。

数据域	域段描述	取值
__IO uint32_t buff_count	接收缓冲区计数（驱动负责管理，无需开发者初始化）。	N/A。
dma_handle_t *p_dma	指向接收通道的DMA句柄dma_handle_t结构体的指针。	N/A。
__IO hal_lock_t lock	ADC锁（驱动负责管理，无需开发者初始化）。	N/A。
__IO hal_adc_state_t state	ADC运行状态（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_ADC_STATE_RESET（未初始化） • HAL_ADC_STATE_READY（已初始化且空闲） • HAL_ADC_STATE_BUSY（忙） • HAL_ADC_STATE_ERROR（错误）
__IO uint32_t error_code	ADC错误码（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_ADC_ERROR_NONE（无错误） • HAL_ADC_ERROR_TIMEOUT（超时） • HAL_ADC_ERROR_DMA（DMA传输错误） • HAL_ADC_ERROR_INVALID_PARAM（非法参数）
uint32_t retention[2]	保存ADC寄存器信息（驱动负责管理，无需开发者初始化）。	N/A

2.10.4 ADC驱动API描述

ADC驱动的API主要包括：

表 2-98 ADC驱动的APIs

API类别	API名称	描述
初始化	hal_adc_init()	初始化ADC外设，配置参考电压等参数。
	hal_adc_deinit()	反初始化ADC外设。
	hal_adc_msp_init()	初始化ADC外设所使用的MSIO引脚、NVIC中断、DMA通道。
	hal_adc_msp_deinit()	反初始化ADC外设所使用的MSIO引脚、NVIC中断、DMA通道。
IO操作	hal_adc_poll_for_conversion()	轮询方式采样数据。
	hal_adc_start_dma()	DMA方式采样数据（非轮询）。
	hal_adc_stop_dma()	DMA方式采样终止。
中断处理及回调函数	hal_adc_conv_cplt_callback()	非轮询采样完成回调，由开发者定义。
状态及错误	hal_adc_get_state()	获取驱动运行状态。
	hal_adc_get_error()	获取错误码。

API类别	API名称	描述
控制	hal_adc_set_dma_threshold()	设置DMA阈值。
	hal_adc_get_dma_threshold()	获取DMA阈值。
睡眠相关	hal_adc_suspend_reg()	睡眠时挂起与ADC配置相关的寄存器。
	hal_adc_resume_reg()	唤醒时恢复与ADC配置相关的寄存器。

下面章节将对各API进行详细描述。

2.10.4.1 hal_adc_init

表 2-99 hal_adc_init接口

函数原型	hal_status_t hal_adc_init(adc_handle_t *p_adc)
功能说明	根据adc_init_t里的参数初始化ADC外设和初始化关联句柄。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。
返回值	HAL状态。
备注	

2.10.4.2 hal_adc_deinit

Table 2-100: hal_adc_deinit接口

函数原型	hal_status_t hal_adc_deinit(adc_handle_t *p_adc)
功能说明	反初始化ADC外设。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。
返回值	HAL状态。
备注	

2.10.4.3 hal_adc_msp_init

表 2-101 hal_adc_msp_init接口

函数原型	void hal_adc_msp_init(adc_handle_t *p_adc)
功能说明	初始化ADC所使用的MOSI引脚、NVIC中断、DMA通道等配置。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成MSIO引脚选择、NVIC中断、DMA通道的初始化。

2.10.4.4 hal_adc_msp_deinit

表 2-102 hal_adc_msp_deinit接口

函数原型	void hal_adc_msp_deinit(adc_handle_t *p_adc)
功能说明	反初始化ADC所使用的MOSI引脚、NVIC中断、DMA通道等配置。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成MSIO引脚选择、NVIC中断、DMA通道的反初始化。

2.10.4.5 hal_adc_poll_for_conversion

表 2-103 hal_adc_poll_for_conversion接口

函数原型	hal_status_t hal_adc_poll_for_conversion(adc_handle_t *p_adc, uint16_t *p_data, uint32_t length)
功能说明	转换ADC数据，以轮询方式读取转换后的数据。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。 p_data：指向存储ADC转换结果的数据缓冲区的指针。 length：数据缓冲区长度。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_adc_get_error获取具体的错误码。

2.10.4.6 hal_adc_start_dma

表 2-104 hal_adc_start_dma接口

函数原型	hal_status_t hal_adc_start_dma(adc_handle_t *p_adc, uint16_t *p_data, uint32_t length)
功能说明	转换ADC数据，以DMA方式读取转换后的数据。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。 p_data：指向存储ADC转换结果的数据缓冲区的指针。 length：数据缓冲区长度。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_adc_get_error()获取具体的错误码。

2.10.4.7 hal_adc_stop_dma

表 2-105 hal_adc_stop_dma接口

函数原型	hal_status_t hal_adc_stop_dma(adc_handle_t *p_adc)
功能说明	中止正在进行的转换，以DMA的方式读取转化后的ADC数据
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。

返回值	HAL状态。
备注	此过程只能用于中止在DMA模式中启动的转换。 此过程执行以下步骤： 1. 禁用ADC时钟，停止转换 2. 调用hal_dma_abort禁止DMA传输 3. 设置Handle状态为READY

2.10.4.8 hal_adc_conv_cplt_callback

表 2-106 hal_adc_conv_cplt_callback接口

函数原型	void hal_adc_conv_cplt_callback(adc_handle_t *p_adc)
功能说明	转换完成回调函数。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成采样完成后的操作。

2.10.4.9 hal_adc_get_state

表 2-107 hal_adc_get_state接口

函数原型	hal_status_t hal_adc_get_state(adc_handle_t *p_adc)
功能说明	返回ADC句柄状态。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。
返回值	无
备注	ADC状态，该参数的取值可能是下列值中的任意一个： • HAL_ADC_STATE_RESET（未初始化） • HAL_ADC_STATE_READY（已初始化且空闲） • HAL_ADC_STATE_BUSY（忙） • HAL_ADC_STATE_ERROR（错误）

2.10.4.10 hal_adc_get_error

表 2-108 hal_adc_get_error接口

函数原型	uint32_t hal_adc_get_error(adc_handle_t *p_adc)
功能说明	返回ADC句柄错误码。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。
返回值	ADC错误码，该参数的取值可以是下列值中的任意一个： • HAL_ADC_ERROR_NONE（无错误） • HAL_ADC_ERROR_DMA（DMA传输错误）

	• HAL_ADC_ERROR_INVALID_PARAM（无效参数） • HAL_ADC_ERROR_TIMEOUT（超时）
备注	

2.10.4.11 hal_adc_set_dma_threshold

表 2-109 hal_adc_set_dma_threshold接口

函数原型	hal_status_t hal_adc_set_dma_threshold(adc_handle_t *p_adc, uint32_t threshold)
功能说明	设置触发DMA传输的FIFO阈值。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。 threshold：FIFO触发阈值（取值范围0 ~ 64）。
返回值	HAL状态。
备注	

2.10.4.12 hal_adc_get_dma_threshold

表 2-110 hal_adc_get_dma_threshold接口

函数原型	uint32_t hal_adc_get_dma_threshold(adc_handle_t *p_adc)
功能说明	获取触发DMA传输的FIFO阈值。
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。
返回值	无
备注	FIFO阈值（取值范围0 ~ 64）。

2.10.4.13 hal_adc_suspend_reg

表 2-111 hal_adc_suspend_reg接口

函数原型	hal_status_t hal_adc_suspend_reg(adc_handle_t *p_adc)
功能说明	睡眠时挂起与ADC配置相关的寄存器
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。
返回值	HAL状态
备注	

2.10.4.14 hal_adc_resume_reg

表 2-112 hal_adc_resume_reg接口

函数原型	hal_status_t hal_adc_resume_reg(adc_handle_t *p_adc);
功能说明	唤醒时恢复与ADC配置相关的寄存器
输入参数	p_adc：指向adc_handle_t结构体变量的指针，该结构体变量包含指定的ADC的配置信息。
返回值	HAL状态

备注	

2.11 HAL DMA通用驱动

2.11.1 DMA驱动功能

DMA（Direct Memory Access）外设的HAL驱动主要实现了以下功能：

- 支持普通、循环两种操作模式。
- 支持外设到内存、内存到外设、外设到外设、内存到内存四种传输方向。
- 支持递增、递减、不变三种地址增量模式。
- 支持字节、半字、字三种数据位宽。
- 支持通道的优先级配置。
- 支持轮询、中断两种传输方式。
- 支持传输完成、块传输完成、中止完成的中断回调函数。
- 支持获取驱动的运行状态及错误码。

2.11.2 如何使用DMA驱动

DMA HAL驱动的使用方法如下：

1. 使能和配置要连接到DMA通道的外设（SRAM内存：不需要初始化）。
2. 对于给定的通道，使用hal_dma_init()函数对这些参数进行配置：DMA源和目的外设、传输方向、源和目的数据格式、循环或普通模式、通道优先级、源和目的地址增量模式。
3. 使用hal_dma_get_state()函数获取DMA状态，并在错误检测时使用hal_dma_get_error()获取DMA错误码。
4. 使用hal_dma_abort()函数中止当前传输。

DMA驱动支持轮询、中断两种传输模式，这两种模式的区别在于传输完成的判断方式（轮询模式需要循环检测DMA传输状态，而中断模式则通过传输完成中断来实现）。两种传输模式具体的使用方法如下：

轮询模式IO操作

1. 配置完源地址和目的地址以及传输数据长度之后，调用hal_dma_start()开始DMA传输。
2. 调用hal_dma_poll_for_transfer()轮询DMA传输状态，直到传输完毕或超时为止。在这种情况下，开发者可根据应用程序配置超时时间。

中断模式IO操作

1. 调用hal_nvic_set_priority()配置DMA中断优先级。
2. 调用hal_nvic_enable_irq()使能DMA中断处理。

3. 调用hal_dma_start_it()配置源地址和目的地址以及传输数据长度和开启DMA中断传输。

4. 在数据传输结束时执行hal_dma_irq_handler()函数，并调用开发者通过hal_dma_register_callback()注册的回调函数。

2.11.3 DMA驱动的结构体

2.11.3.1 dma_init_t

DMA驱动的初始化结构体dma_init_t的定义如下：

表 2-113 dma_init_t结构体

数据域	域段描述	取值
uint32_t src_request	源外设类型。	该参数的取值可以是下列值中的任意一个： • DMA_REQUEST_SPIM_TX（SPIM发送） • DMA_REQUEST_SPIM_RX（SPIM接收） • DMA_REQUEST_SPIS_TX（SPIS发送） • DMA_REQUEST_SPIS_RX（SPIS接收） • DMA_REQUEST_QSPI0_TX（QSPI0发送） • DMA_REQUEST_QSPI0_RX（QSPI0接收） • DMA_REQUEST_I2C0_TX（I2C0发送） • DMA_REQUEST_I2C0_RX（I2C0接收） • DMA_REQUEST_I2C1_TX（I2C1发送） • DMA_REQUEST_I2C1_RX（I2C1接收） • DMA_REQUEST_I2S_S_TX（I2SS发送） • DMA_REQUEST_I2S_S_RX（I2SS接收） • DMA_REQUEST_UART0_TX（UART0发送） • DMA_REQUEST_UART0_RX（UART0接收） • DMA_REQUEST_QSPI1_TX（QSPI1发送） • DMA_REQUEST_QSPI1_RX（QSPI1接收） • DMA_REQUEST_I2S_M_TX（I2SM发送） • DMA_REQUEST_I2S_M_RX（I2SM接收） • DMA_REQUEST_SNSADC（Sense ADC） • DMA_REQUEST_MEM（内存）
uint32_t dst_request	目的外设类型。	该参数的取值可以是下列值中的任意一个： • DMA_REQUEST_SPIM_TX（SPIM发送） • DMA_REQUEST_SPIM_RX（SPIM接收） • DMA_REQUEST_SPIS_TX（SPIS发送） • DMA_REQUEST_SPIS_RX（SPIS接收） • DMA_REQUEST_QSPI0_TX（QSPI0发送） • DMA_REQUEST_QSPI0_RX（QSPI0接收） • DMA_REQUEST_I2C0_TX（I2C0发送） • DMA_REQUEST_I2C0_RX（I2C0接收） • DMA_REQUEST_I2C1_TX（I2C1发送）

数据域	域段描述	取值
		• DMA_REQUEST_I2C1_RX（I2C1接收） • DMA_REQUEST_I2S_S_TX（I2SS发送） • DMA_REQUEST_I2S_S_RX（I2SS接收） • DMA_REQUEST_UART0_TX（UART0发送） • DMA_REQUEST_UART0_RX（UART0接收） • DMA_REQUEST_QSPI1_TX（QSPI1发送） • DMA_REQUEST_QSPI1_RX（QSPI1接收） • DMA_REQUEST_I2S_M_TX（I2SM发送） • DMA_REQUEST_I2S_M_RX（I2SM接收） • DMA_REQUEST_SNSADC（Sense ADC） • DMA_REQUEST_MEM（内存）
uint32_t direction	数据传输方式。	该参数的取值可以是下列值中的任意一个： • DMA_MEMORY_TO_MEMORY（内存到内存） • DMA_MEMORY_TO_PERIPH（内存到外设） • DMA_PERIPH_TO_MEMORY（外设到内存） • DMA_PERIPH_TO_PERIPH（外设到外设）
uint32_t src_increment	源地址增量模式。	该参数的取值可以是下列值中的任意一个： • DMA_SRC_INCREMENT（源地址递增） • DMA_SRC_DECREMENT（源地址递减） • DMA_SRC_NO_CHANGE（源地址不变）
uint32_t dst_increment	目的地址增量模式。	该参数的取值可以是下列值中的任意一个： • DMA_DST_INCREMENT（目的地址递增） • DMA_DST_DECREMENT（目的地址递减） • DMA_DST_NO_CHANGE（目的地址不变）
uint32_t src_data_alignment	源地址数据位宽/对齐方式。	该参数的取值可以是下列值中的任意一个： • DMA_SDATAALIGN_BYTE（字节对齐） • DMA_SDATAALIGN_HALFWORD（半字对齐） • DMA_SDATAALIGN_WORD（字对齐）
uint32_t dst_data_alignment	目的地址数据位宽/对齐方式。	该参数的取值可以是下列值中的任意一个： • DMA_DDATAALIGN_BYTE（字节对齐） • DMA_DDATAALIGN_HALFWORD（半字对齐） • DMA_DDATAALIGN_WORD（字对齐）
uint32_t mode	操作模式。	该参数的取值可以是下列值中的任意一个： • DMA_NORMAL（普通模式，单次传输） • DMA_CIRCULAR（循环模式，多次传输）
uint32_t priority	通道优先级。	该参数的取值可以是下列值中的任意一个： • DMA_PRIORITY_LOW（低优先级） • DMA_PRIORITY_MEDIUM（中优先级） • DMA_PRIORITY_HIGH（高优先级） • DMA_PRIORITY_VERY_HIGH（最高优先级）

2.11.3.2 dma_handle_t

DMA驱动的句柄结构体dma_handle_t的定义如下：

表 2-114 dma_handle_t结构体

数据域	域段描述	取值
dma_channel_t channel	DMA通道实例。	该参数的取值可以是下列值中的任意一个： • DMA_Channel0（通道0） • DMA_Channel1（通道1） • DMA_Channel2（通道2） • DMA_Channel3（通道3） • DMA_Channel4（通道4） • DMA_Channel5（通道5） • DMA_Channel6（通道6） • DMA_Channel7（通道7）
dma_init_t init	初始化结构体。	参考dma_init_t结构体。
__IO hal_lock_t lock	DMA锁（无需开发者初始化）。	N/A
__IO hal_dma_state_t state	DMA运行状态（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_DMA_STATE_RESET（未初始化） • HAL_DMA_STATE_READY（已初始化且空闲） • HAL_DMA_STATE_BUSY（忙） • HAL_DMA_STATE_TIMEOUT（超时） • HAL_DMA_STATE_ERROR（错误）
void *p_parent	使用当前通道实例的外设句柄指针，可以是其它外设的句柄指针。	N/A
void (*xfer_tfr_callback)(struct __dma_handle_t *p_dma)	传输完成回调函数。 可通过hal_dma_register_callback()及hal_dma_unregister_callback()进行回调函数的注册及注销。	N/A
void (*xfer_blk_callback)(struct __dma_handle_t *p_dma)	块传输完成回调函数。 可通过hal_dma_register_callback()及hal_dma_unregister_callback()进行回调函数的注册及注销。	N/A
void (*xfer_error_callback)(struct __dma_handle_t *p_dma)	错误回调函数。可通过hal_dma_register_callback()及hal_dma_unregister_callback()进行回调函数的注册及注销。	N/A

数据域	域段描述	取值
void (*xfer_abort_callback)(struct __dma_handle_t *p_dma)	中止完成回调函数。可通过hal_dma_register_callback()及hal_dma_unregister_callback()进行回调函数的注册及注销。	N/A
__IO uint32_t error_code	DMA错误码（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_DMA_ERROR_NONE（无错误） • HAL_DMA_ERROR_TE（传输错误） • HAL_DMA_ERROR_NO_XFER（无正在进行的数据传输） • HAL_DMA_ERROR_TIMEOUT（超时）
uint32_t retention[5];	保存DMA寄存器信息（驱动负责管理，无需开发者初始化）。	N/A

2.11.4 DMA驱动API描述

DMA驱动的API主要包括：

表 2-115 DMA驱动的APIs

API类别	API名称	描述
初始化	hal_dma_init()	初始化DMA外设的指定通道，配置外设及目的外设的参数。
	hal_dma_deinit()	反初始化DMA外设的指定通道。
IO操作	hal_dma_start()	启动DMA传输，轮询方式，传输完成时无中断。
	hal_dma_start_it()	启动DMA传输，中断方式，传输完成时触发中断。
	hal_dma_poll_for_transfer()	轮询DMA传输，传输完成时退出该函数，与hal_dma_start()配合使用。
	hal_dma_abort()	中止传输，中止完成时不调用中止完成回调函数。
	hal_dma_abort_it()	中止传输，中止完成时调用中止完成回调函数。
中断处理及回调函数	hal_dma_irq_handler()	中断处理函数。
	hal_dma_register_callback()	注册DMA的中断回调函数。
	hal_dma_unregister_callback()	注销DMA的中断回调函数。
状态及错误	hal_dma_get_state()	获取驱动运行状态。
	hal_dma_get_error()	获取错误码。
睡眠相关	hal_dma_suspend_reg()	睡眠时挂起与DMA配置相关的寄存器。
	hal_dma_resume_reg()	唤醒时恢复与DMA配置相关的寄存器。

下面章节将对各API进行详细描述。

2.11.4.1 hal_dma_init

表 2-116 hal_dma_init接口

函数原型	hal_status_t hal_dma_init(dma_handle_t *p_dma)
功能说明	根据dma_init_t中的指定参数初始化DMA外设的指定通道。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。
返回值	HAL状态。
备注	该函数只对DMA通道进行初始化配置，可调用hal_dma_start()和hal_dma_start_it()进行DMA传输。

2.11.4.2 hal_dma_deinit

表 2-117 hal_dma_deinit接口

函数原型	hal_status_t hal_dma_deinit(dma_handle_t *p_dma)
功能说明	反初始化DMA外设。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。
返回值	HAL状态。
备注	

2.11.4.3 hal_dma_start

表 2-118 hal_dma_start接口

函数原型	hal_status_t hal_dma_start(dma_handle_t *p_dma, uint32_t src_address, uint32_t dst_address, uint32_t data_length)
功能说明	开始DMA传输。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。 src_address：源内存缓冲区地址，该地址必须与hal_dma_init()配置的src_data_alignment对齐。 dst_address：目的内存缓冲区地址，该地址必须与hal_dma_init()配置的dst_data_alignment对齐。 data_length：从源到目的地传输的数据的长度，该长度的最小数据单元与src_data_alignment相同。
返回值	HAL状态。
备注	

2.11.4.4 hal_dma_start_it

表 2-119 hal_dma_start_it接口

函数原型	hal_status_t hal_dma_start_it(dma_handle_t *p_dma, uint32_t src_address, uint32_t dst_address, uint32_t data_length)

功能说明	使能DMA传输并使能中断。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。 src_address：源内存缓冲区地址，该地址必须与hal_dma_init()配置的src_data_alignment对齐。 dst_address：目的内存缓冲区地址，该地址必须与hal_dma_init()配置的dst_data_alignment对齐。 data_length：从源到目的地传输的数据的长度，该地址必须与hal_dma_init()配置的src_data_alignment对齐。
返回值	HAL状态。
备注	

2.11.4.5 hal_dma_abort

表 2-120 hal_dma_abort接口

函数原型	hal_status_t hal_dma_abort(dma_handle_t *p_dma)
功能说明	中止DMA传输。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。
返回值	HAL状态。
备注	

2.11.4.6 hal_dma_abort_it

表 2-121 hal_dma_abort_it接口

函数原型	hal_status_t hal_dma_abort_it(dma_handle_t *p_dma)
功能说明	在中断模式下中止DMA传输。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。
返回值	HAL状态。
备注	

2.11.4.7 hal_dma_poll_for_transfer

表 2-122 hal_dma_poll_for_transfer接口

函数原型	hal_status_t hal_dma_poll_for_transfer(dma_handle_t *p_dma, uint32_t timeout)
功能说明	轮询传输完成。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。 timeout：超时时间。
返回值	HAL状态。
备注	

2.11.4.8 hal_dma_irq_handler

表 2-123 hal_dma_irq_handler接口

函数原型	void hal_dma_irq_handler(dma_handle_t *p_dma)
功能说明	处理DMA中断请求。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。
返回值	无
备注	

2.11.4.9 hal_dma_register_callback

表 2-124 hal_dma_register_callback接口

函数原型	hal_status_t hal_dma_register_callback(dma_handle_t *p_dma, hal_dma_callback_id_t id, void (* callback)(dma_handle_t *p_dma))
功能说明	注册回调函数。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。 id：回调类型标识，该参数可以是下列值中的任意一个： • HAL_DMA_XFER_TFR_CB_ID（传输完成回调函数ID） • HAL_DMA_XFER_BLK_CB_ID（块传输完成回调函数ID） • HAL_DMA_XFER_ABORT_CB_ID（中止完成回调函数ID） callback：指向私有回调函数的指针。
返回值	HAL状态。
备注	

2.11.4.10 hal_dma_unregister_callback

表 2-125 hal_dma_unregister_callback接口

函数原型	hal_status_t hal_dma_unregister_callback(dma_handle_t *p_dma, hal_dma_callback_id_t callback_id)
功能说明	注销回调函数。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。 callback_id：回调类型标识，该参数可以是下列值中的任意一个： • HAL_DMA_XFER_TFR_CB_ID（传输完成回调函数ID） • HAL_DMA_XFER_BLK_CB_ID（块传输完成回调函数ID） • HAL_DMA_XFER_ABORT_CB_ID（中止完成回调函数ID） • HAL_DMA_XFER_ALL_CB_ID（所有的完成回调函数ID）
返回值	HAL状态。
备注	

2.11.4.11 hal_dma_get_state

表 2-126 hal_dma_get_state接口

函数原型	hal_dma_state_t hal_dma_get_state(dma_handle_t *p_dma)
功能说明	获取DMA运行状态。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。
返回值	DMA运行状态，该参数的取值可能是下列值中的任意一个： • HAL_DMA_STATE_RESET（未初始化） • HAL_DMA_STATE_READY（已初始化且空闲） • HAL_DMA_STATE_BUSY（忙） • HAL_DMA_STATE_TIMEOUT（超时） • HAL_DMA_STATE_ERROR（错误）
备注	

2.11.4.12 hal_dma_get_error

表 2-127 hal_dma_get_error接口

函数原型	hal_dma_state_t hal_dma_get_state(dma_handle_t *p_dma)
功能说明	获取DMA 错误码。
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。
返回值	DMA错误码，该参数的取值可能是下列值中的任意一个： • HAL_DMA_ERROR_NONE（无错误） • HAL_DMA_ERROR_TE（传输错误） • HAL_DMA_ERROR_NO_XFER（无正在进行的数据传输） • HAL_DMA_ERROR_TIMEOUT（超时）
备注	

2.11.4.13 hal_dma_suspend_reg

表 2-128 hal_dma_suspend_reg接口

函数原型	hal_status_t hal_dma_suspend_reg(dma_handle_t *p_dma)
功能说明	睡眠时挂起与DMA配置相关的寄存器
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。
返回值	HAL状态
备注	

2.11.4.14 hal_dma_resume_reg

表 2-129 hal_dma_resume_reg接口

函数原型	hal_status_t hal_dma_resume_reg(dma_handle_t *p_dma);

功能说明	唤醒时恢复与DMA配置相关的寄存器
输入参数	p_dma：指向dma_handle_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。
返回值	HAL状态
备注	

2.12 HAL DUAL TIMER通用驱动

2.12.1 DUAL TIMER驱动功能

DUAl TIMER外设的HAL驱动主要实现了以下功能：

- 支持32位计数初值配置。
- 支持无分频、16分频、256分频三种时钟分频配置。
- 支持单次、循环两种计数模式。
- 支持轮询、中断两种计数方式。
- 支持停止轮询及中断方式方式下的计数操作。
- 支持计数完成的中断回调函数。
- 支持获取驱动运行状态。

2.12.2 如何使用DUAL TIMER驱动

DUAL TIMER HAL驱动的使用方法如下：

1. 声明一个dual_timer_handle_t句柄结构，例如：dual_timer_handle_t dtim_handle。
2. 重写hal_dual_timer_base_msp_init()以初始化DUAL TIMER底层资源。如要使用中断方式的API函数hal_dual_timer_base_start_it()计数，则需通过调用相关的NVIC接口来配置：
   - 调用hal_nvic_set_priority()配置DUAL TIMER中断优先级。
   - 调用hal_nvic_enable_irq()使能DUAL TIMER中断处理。
3. 配置dtim_handle句柄init结构中的计数初值、计数模式、分频系数。
4. 调用hal_dual_timer_base_init() API初始化DUAL TIMER外设。
5. 如果要使用轮询方式的API函数hal_dual_timer_base_start()计数，开发者需调用hal_dual_timer_get_state()获取当前的驱动运行状态，以判断当前计数是否完成。
6. 如果要使用中断方式的API函数hal_dual_timer_base_start_it()计数，开发者需重写hal_dual_timer_period_elapsed_callback()中断回调函数，DUAL TIMER计数完成中断触发时，该回调函数会被自动调用。

7. 如果配置的计数模式为单次计数，则计数完成后计数器停止，开发者需要重新初始化DUAL TIMER以启动下次计数；如果配置的计数模式为循环计数，则计数完成后计数器重新加载计数初值，自动开始下次计数。

2.12.3 DUAl TIMER驱动的结构体

2.12.3.1 dual_timer_init_t

TIMER驱动的初始化结构体dual_timer_init_t的定义如下：

表 2-130 dual_timer_init_t结构体

数据域	域段描述	取值
uint32_t prescaler	分频系数。	该参数的取值可以是下列值中的任意一个： • DUAL_TIMER_PRESCALER_DIV0（无分频） • DUAL_TIMER_PRESCALER_DIV16（16分频） • DUAL_TIMER_PRESCALER_DIV256（256分频）
uint32_t counter_mode	计数模式。	该参数的取值可以是下列值中的任意一个： • DUAL_TIMER_COUNTERMODE_LOOP（循环模式） • DUAL_TIMER_COUNTERMODE_ONESHOT（单次模式）
uint32_t auto_reload	自动加载的计数值。	0x0000_0000 ～ 0xFFFF_FFFF

2.12.3.2 dual_timer_handle_t

DUAL TIMER驱动的句柄结构体dual_timer_handle_t的定义如下：

表 2-131 dual_timer_handle_t结构体

数据域	域段描述	取值
dual_timer_regs_t *p_instance	TIMER外设实例。	该参数的取值可以是下列值中的任意一个： • DUAL_TIMER0 • DUAL_TIMER1
dual_timer_init_t init	初始化结构体。	参考dual_timer_handle_t结构体。
__IO hal_lock_t lock	DUAL TIMER锁（无需开发者初始化）。	N/A
__IO hal_dual_timer_state_t state	DUAL TIMER运行状态（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_DUAL_TIMER_STATE_RESET（未初始化） • HAL_DUAL_TIMER_STATE_READY（已初始化且空闲） • HAL_DUAL_TIMER_STATE_BUSY（忙） • HAL_DUAL_TIMER_STATE_ERROR （错误）

2.12.4 DUAl TIMER驱动API描述

DUAL TIMER驱动的API主要包括：

表 2-132 DUAL TIMER驱动的APIs

API类别	API名称	描述
初始化	hal_dual_timer_base_init()	初始化DUAL TIMER外设，配置计数初值等参数。
	hal_dual_timer_base_deinit()	反初始化DUAL TIMER外设。
	hal_dual_timer_base_msp_init()	初始化DUAL TIMER外设所使用的NVIC中断。
	hal_dual_timer_base_msp_deinit()	反初始化DUAL TIMER外设所使用的NVIC中断。
IO操作	hal_dual_timer_base_start()	启动计数，轮询方式。
	hal_dual_timer_base_stop()	停止计数，轮询方式。
	hal_dual_timer_base_start_it()	启动计数，中断方式。
	hal_dual_timer_base_stop_it()	停止计数，中断方式。
控制	hal_dual_timer_set_config()	配置计数器参数。
中断处理及回调函数	hal_dual_timer_irq_handler()	中断处理函数。
	hal_dual_timer_period_elapsed_callback()	计数完成的中断回调函数。
状态及错误	hal_dual_timer_get_state()	获取驱动运行状态。

下面章节将对各API进行详细描述。

2.12.4.1 hal_dual_timer_base_init

表 2-133 hal_dual_timer_base_init接口

函数原型	hal_status_t hal_dual_timer_base_init(dual_timer_handle_t *p_dual_timer)
功能说明	根据dual_timer_handle_t中的指定参数初始化DUAL TIMER时间基单元，并初始化相关的句柄。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。
返回值	HAL状态。
备注	

2.12.4.2 hal_dual_timer_base_deinit

表 2-134 hal_dual_timer_base_deinit接口

函数原型	hal_status_t hal_dual_timer_base_deinit(dual_timer_handle_t *p_dual_timer)
功能说明	反初始化DUAL TIMER外设。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。

返回值	HAL状态。
备注	

2.12.4.3 hal_dual_timer_base_msp_init

表 2-135 hal_dual_timer_base_msp_init接口

函数原型	void hal_dual_timer_base_msp_init(dual_timer_handle_t *p_dual_timer)
功能说明	初始化DUAL TIMER所使用的NVIC中断配置。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成NVIC中断的初始化。

2.12.4.4 hal_dual_timer_base_msp_deinit

表 2-136 hal_dual_timer_base_msp_deinit接口

函数原型	void hal_dual_timer_msp_base_deinit(dual_timer_handle_t *p_dual_timer)
功能说明	反初始化DUAL TIMER的NVIC中断配置。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成NVIC中断的反初始化。

2.12.4.5 hal_dual_timer_base_start

表 2-137 hal_dual_timer_base_start接口

函数原型	hal_status_t hal_dual_timer_base_start(dual_timer_handle_t *p_dual_timer)
功能说明	使能DUAL TIMER外设，开始计数，轮询方式。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。
返回值	HAL状态。
备注	该API未使能DUAL TIMER中断，开发者需调用hal_dual_timer_get_state()获取计数状态。

2.12.4.6 hal_dual_timer_base_stop

表 2-138 hal_dual_timer_base_stop接口

函数原型	hal_status_t hal_dual_timer_base_stop(dual_timer_handle_t *p_dual_timer)
功能说明	禁用DUAL TIMER外设，停止轮询方式下的计数。

输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。
返回值	HAL状态。
备注	该API未禁止DUAL TIMER中断，可与hal_dual_timer_base_start()配合使用。

2.12.4.7 hal_dual_timer_base_start_it

表 2-139 hal_dual_timer_base_start_it接口

函数原型	hal_status_t hal_dual_timer_base_start_it(dual_timer_handle_t *p_dual_timer)
功能说明	使能DUAL TIMER外设，开始计数，中断方式。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。
返回值	HAL状态。
备注	该API会使能DUAL TIMER中断，计数完成时会调用回调函数hal_dual_timer_period_elapsed_callback()。

2.12.4.8 hal_dual_timer_base_stop_it

表 2-140 hal_dual_timer_base_stop_it接口

函数原型	hal_status_t hal_dual_timer_base_stop_it(dual_timer_handle_t *p_dual_timer)
功能说明	禁用DUAL TIMER外设，停止中断方式下的计数。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。
返回值	HAL状态。
备注	该API禁止DUAL TIMER中断，可与hal_dual_timer_base_start_it()配合使用。

2.12.4.9 hal_dual_timer_set_config

表 2-141 hal_dual_timer_set_config接口

函数原型	hal_status_t hal_dual_timer_set_config(dual_timer_handle_t *p_dual_timer, dual_timer_init_t *p_structure)
功能说明	配置DUAL TIMER计数器参数。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。 p_structure：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的计数器配置信息。
返回值	HAL状态。
备注	

2.12.4.10 hal_dual_timer_irq_handler

表 2-142 hal_dual_timer_irq_handler接口

函数原型	void hal_dual_timer_irq_handler(dual_timer_handle_t *p_dual_timer)
功能说明	DUAL TIMER中断处理函数。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。
返回值	无
备注	该API适合在初始化DUAL TIMER之后更改计数器配置。

2.12.4.11 hal_dual_timer_period_elapsed_callback

表 2-143 hal_dual_timer_period_elapsed_callback接口

函数原型	void hal_dual_timer_period_elapsed_callback(dual_timer_handle_t *p_dual_timer)
功能说明	DUAL TIMER外设计数完成的中断回调函数。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.12.4.12 hal_dual_timer_get_state

表 2-144 hal_dual_timer_get_state接口

函数原型	hal_dual_timer_state_t hal_dual_timer_get_state(dual_timer_handle_t *p_dual_timer)
功能说明	返回DUAL TIMER句柄状态。
输入参数	p_dual_timer：指向dual_timer_handle_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER模块的配置信息。
返回值	DUAL TIMER状态，该参数的取值可能是下列值中的任意一个： • HAL_DUAL_TIMER_STATE_RESET（未初始化） • HAL_DUAL_TIMER_STATE_READY（已初始化且空闲） • HAL_DUAL_TIMER_STATE_BUSY（忙） • HAL_DUAL_TIMER_STATE_ERROR（错误）
备注	

2.13 HAL AES 通用驱动

2.13.1 AES驱动功能

AES（Advanced Encryption Standard）外设的HAL驱动主要实现了以下功能：

- 支持128 bits，192 bits和256 bits密钥。
- 支持ECB和CBC模式下的加密和解密。
- 支持多种密钥加载模式：MCU，KPORT。
- 支持反DPA攻击。
- 支持轮询、中断两种操作模式。
- 支持中断模式下的回调函数。
- 支持驱动运行状态及错误码的获取。
- 支持超时设置。

2.13.2 如何使用AES驱动

2.13.2.1 初始化

初始化AES HAL驱动的方法如下：

1. 声明一个aes_handle_t句柄结构，例如：aes_handle_t aes_handle。
2. 重写hal_aes_msp_init()以初始化AES底层资源。若要使用中断模式，则需通过调用相关的NVIC接口来配置：
   - 调用hal_nvic_set_priority()配置AES中断优先级。
   - 调用hal_nvic_enable_irq()使能AES中断。
3. 配置aes_handle句柄的p_instance和init，包括AES外设实例、密钥长度、加解密块操作模式、密钥指针、CBC模式初始化向量、安全模式以及随机数种子。
4. 调用hal_aes_init()初始化AES寄存器。

2.13.2.2 ECB加解密

ECB加解密支持轮询、中断两种计算模式。这两种模式的区别在于计算数据的加载方式和计算完成的判断方式：轮询模式需要循环检测完成状态，而中断模式则通过计算完成中断来实现。两种计算模式的具体使用方法如下：

轮询模式IO操作

1. 使用hal_aes_ecb_encrypt()加密数据，hal_aes_ecb_decrypt()解密数据。
2. 直至计算完成或超时返回。若返回错误，可调用hal_aes_get_error()查看错误代码；加解密数据较多时可重复步骤1。

中断模式IO操作

1. 开发者可根据需要实现hal_aes_done_callback()、hal_aes_error_callback()和hal_aes_abort_cplt_callback()。

2. 使用hal_aes_ecb_encrypt_it()加密数据，hal_aes_ecb_decrypt_it()解密数据。

3. 计算完成时hal_aes_done_callback()会被调用，若计算出错，则hal_aes_error_callback()会被调用；加解密数据较多时可重复步骤2。

4. 如需中止计算，则可调用hal_aes_abort()和hal_aes_abort_it()。hal_aes_abort()直接中止当前计算，hal_aes_abort_it()中止当前计算后会调用hal_aes_abort_cplt_callback()。

2.13.2.3 CBC加解密

CBC加解密支持轮询、中断两种计算模式，这两种模式的区别在于计算数据的加载方式和计算完成的判断方式：轮询模式需要循环检测完成状态，而中断模式则通过计算完成中断来实现。两种计算模式具体的使用方法如下：

轮询模式IO操作

1. 重载初始化向量p_init_vector，并使用hal_aes_cbc_encrypt()加密数据，hal_aes_cbc_decrypt()解密数据。若数据流太长无法一次性加解密完成，则需将数据分段处理。对于非首段数据的处理，加密时p_init_vector为上一次的计算结果的最后16 bytes，解密时p_init_vector为上一次待解密数据的最后16 bytes。

2. 直至计算完成或超时返回，若返回错误，可调用hal_aes_get_error()查看错误代码；加解密数据较多时可重复步骤1。

中断模式IO操作

1. 开发者可根据需要实现hal_aes_done_callback()、hal_aes_error_callback()和hal_aes_abort_cplt_callback()。

2. 重载初始化向量p_init_vector，并使用hal_aes_cbc_encrypt_it()加密数据，hal_aes_cbc_decrypt_it()解密数据。若数据流太长无法一次性加解密完成，则需将数据分段处理。对于非首段数据的处理，加密时p_init_vector为上一次的计算结果的最后16 bytes，解密时p_init_vector为上一次待解密数据的最后16 bytes。

3. 计算完成时hal_aes_done_callback()会被调用，计算出错时hal_aes_error_callback()会被调用；加解密数据较多时可重复步骤2。

4. 如需中止计算可调用hal_aes_abort()和hal_aes_abort_it()。hal_aes_abort()直接中止当前计算，hal_aes_abort_it()中止当前计算后会调用hal_aes_abort_cplt_callback()。

2.13.3 AES驱动的结构体

2.13.3.1 aes_init_t

AES驱动的初始化结构体aes_init_t的定义如下：

表 2-145 aes_init_t结构体

数据域	域段描述	取值
uint32_t key_size	密钥长度。	该参数的取值可以是下列值中的任意一个： • AES_KEYSIZE_128BITS（128位） • AES_KEYSIZE_192BITS（192位） • AES_KEYSIZE_256BITS（256位）
uint32_t operation_mode	工作模式（加密或解密）。	该参数的取值可以是下列值中的任意一个： • AES_OPERATION_MODE_ENCRYPT（加密） • AES_OPERATION_MODE_DECRYPT（解密）
uint32_t chaining_mode	数据流加解密方式。	该参数的取值可以是下列值中的任意一个： • AES_CHAININGMODE_ECB（ECB数据流操作） • AES_CHAININGMODE_CBC（CBC数据流操作）
uint32_t *p_key	密钥。	指向密钥的指针。
uint32_t *p_init_vector	初始化向量，此参数在CBC模式下有效。	指向初始化向量的指针。
uint32_t dpa_mode	是否启用安全模式。	该参数的取值可以是下列值中的任意一个： • ENABLE（启用） • DISABLE（禁用）
uint32_t *p_seed	安全模式的随机种子。	指向长度为16字节的数组。

2.13.3.2 aes_handle_t

AES驱动的句柄结构体aes_handle_t的定义如下：

表 2-146 aes_handle_t结构体

数据域	域段描述	取值
aes_regs_t *p_instance	AES外设实例。	该参数的取值为AES。
aes_init_t init	初始化结构体。	参考aes_init_t结构体。
uint32_t *p_cryp_input_buffer	指向待加解密数据流缓冲区的指针（无需开发者初始化）。	N/A
uint32_t *p_cryp_output_buffer	指向加解密结果缓冲区的指针（无需开发者初始化）。	N/A
uint32_t block_size	待加解密数据块的大小（无需开发者初始化）	N/A
uint32_t block_count	待加解密数据块的计数（无需开发者初始化）。	初始化为block_size，计算过程中递减到0。
__IO hal_lock_t lock	AES锁（无需开发者初始化）。	N/A
__IO hal_aes_state_t state	AES运行状态（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_AES_STATE_RESET（未初始化）

数据域	域段描述	取值
		• HAL_AES_STATE_READY（已初始化且空闲） • HAL_AES_STATE_BUSY（忙） • HAL_AES_STATE_ERROR（运行错误） • HAL_AES_STATE_TIMEOUT（超时） • HAL_AES_STATE_SUSPENDED（已挂起）
__IO uint32_t error_code	AES错误码（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_AES_ERROR_NONE（无错误） • HAL_AES_ERROR_TIMEOUT（超时） • HAL_AES_ERROR_TRANSFER（传输错误） • HAL_AES_ERROR_INVALID_PARAM（非法参数）
uint32_t timeout	AES超时时间（无需开发者初始化）。	N/A
uint32_t retention[18]	保存AES寄存器信息（驱动负责管理，无需开发者初始化）。	N/A

说明:

N/A表示参数没有取值选项。

2.13.4 AES驱动API描述

AES驱动的API主要包括：

表 2-147 AES驱动的APIs

API类别	API名称	描述
初始化	hal_aes_init()	初始化AES外设，配置密钥等参数。
	hal_aes_deinit()	反初始化AES外设。
	hal_aes_msp_init()	初始化AES外设所使用的NVIC中断。
	hal_aes_msp_deinit()	反初始化AES外设所使用的NVIC中断。
IO操作	hal_aes_ecb_encrypt()	ECB模式加密，轮询方式。
	hal_aes_ecb_decrypt()	ECB模式解密，轮询方式。
	hal_aes_cbc_encrypt()	CBC模式加密，轮询方式。
	hal_aes_cbc_decrypt()	CBC模式解密，轮询方式。
	hal_aes_ecb_encrypt_it()	ECB模式加密，中断方式。
	hal_aes_ecb_decrypt_it()	ECB模式解密，中断方式。
	hal_aes_cbc_encrypt_it()	CBC模式加密，中断方式。
	hal_aes_cbc_decrypt_it()	CBC模式解密，中断方式。
	hal_aes_abort()	中止加解密，轮询方式。

API类别	API名称	描述
	hal_aes_abort_it()	中止加解密，中断方式。
中断处理及回调函数	hal_aes_irq_handler()	中断处理函数。
	hal_aes_done_callback()	加解密完成中断回调函数。
	hal_aes_error_callback()	错误中断回调函数。
	hal_aes_abort_cplt_callback()	中止完成中断回调函数。
状态及错误	hal_aes_get_state()	获取驱动运行状态。
	hal_aes_get_error()	获取错误码。
控制	hal_aes_set_timeout()	设置超时时间。
睡眠相关	hal_aes_suspend_reg()	睡眠时挂起AES配置相关的寄存器。
	hal_aes_resume_reg()	唤醒时恢复AES配置相关的寄存器。

下面章节将对各API进行详细描述。

2.13.4.1 hal_aes_init

表 2-148 hal_aes_init接口

函数原型	hal_status_t hal_aes_init(aes_handle_t *p_aes)
功能说明	根据aes_init_t结构体里的参数初始化AES外设。
输入参数	p_aes：指向aes_handle_t结构体变量的指针，该结构体变量包含指定的AES配置信息。
返回值	HAL状态。
备注	

2.13.4.2 hal_aes_deinit

表 2-149 hal_aes_deinit接口

函数原型	hal_status_t hal_aes_deinit(aes_handle_t *p_aes)
功能说明	反初始化AES外设，将AES模块的寄存器恢复为默认值。
输入参数	p_aes：指向aes_handle_t结构体变量的指针，包含指定的寄存器基址。
返回值	HAL状态。
备注	

2.13.4.3 hal_aes_msp_init

表 2-150 hal_aes_msp_init接口

函数原型	void hal_aes_msp_init(aes_handle_t *p_aes)
功能说明	初始化AES外设所使用的NVIC中断、DMA通道等配置。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。

返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成NVIC中断、DMA通道的初始化。

2.13.4.4 hal_aes_msp_deinit

表 2-151 hal_aes_msp_deinit接口

函数原型	void hal_aes_msp_deinit(aes_handle_t *p_aes)
功能说明	反初始化AES外设所使用的NVIC中断、DMA通道等配置。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成NVIC中断、DMA通道的反初始化。

2.13.4.5 hal_aes_ecb_encrypt

表 2-152 hal_aes_ecb_encrypt接口

函数原型	hal_status_t hal_aes_ecb_encrypt(aes_handle_t *p_aes, uint32_t *p_plain_data, uint32_t number, uint32_t *p_cypher_data, uint32_t timeout)
功能说明	ECB模式加密，轮询方式。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。 p_plain_data：指向待加密数据的指针。 number：待加密数据的长度，单位byte，最大取值为32768，且必须为16的整数倍。 p_cypher_data：指向存储加密结果的内存空间的指针。 timeout：超时时间，单位ms。
返回值	HAL状态。
备注	待加密数据流超过最大长度时，可分多次加密。

2.13.4.6 hal_aes_ecb_decrypt

表 2-153 hal_aes_ecb_decrypt接口

函数原型	hal_status_t hal_aes_ecb_decrypt(aes_handle_t *p_aes, uint32_t *p_cypher_data, uint32_t number, uint32_t *p_plain_data, uint32_t timeout)
功能说明	ECB模式解密，轮询方式。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。 p_cypher_data：指向待解密数据的指针。 number：待解密数据的长度，单位byte，最大取值为32768，且必须为16的整数倍。 p_plain_data：指向存储解密结果的内存空间的指针。 timeout：超时时间，单位ms。
返回值	HAL状态

备注	待解密数据流超过最大长度时，可分多次解密

2.13.4.7 hal_aes_cbc_encrypt

表 2-154 hal_aes_cbc_encrypt接口

函数原型	hal_status_t hal_aes_cbc_encrypt(aes_handle_t *p_aes, uint32_t *p_plain_data, uint32_t number, uint32_t *p_cypher_data, uint32_t timeout)
功能说明	CBC模式加密，轮询方式。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。 p_plain_data：指向待加密数据的指针。 number：待加密数据的长度，单位byte，最大取值为32768，且必须为16的整数倍。 p_cypher_data：指向存储加密结果的内存空间的指针。 timeout：超时时间，单位ms。
返回值	HAL状态。
备注	待加密数据流超过最大长度时，可分多次加密，并且从第二次开始需重载初始化向量（p_aes->init.p_init_vector）为上一次加密结果的最后16 bytes数据。

2.13.4.8 hal_aes_cbc_decrypt

表 2-155 hal_aes_cbc_decrypt接口

函数原型	hal_status_t hal_aes_cbc_decrypt(aes_handle_t *p_aes, uint32_t *p_cypher_data, uint32_t number, uint32_t *p_plain_data, uint32_t timeout)
功能说明	CBC模式解密，轮询方式。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。 p_cypher_data：指向待解密数据的指针。 number：待解密数据的长度，单位byte，最大取值为32768，且必须为16的整数倍。 p_plain_data：指向存储解密结果的内存空间的指针。 timeout：超时时间，单位ms。
返回值	HAL状态。
备注	待解密数据流超过最大长度时，可分多次解密，从第二次开始需重载初始化向量（p_aes->init.p_init_vector）为上一次待解密数据的最后16 bytes数据。

2.13.4.9 hal_aes_ecb_encrypt_it

表 2-156 hal_aes_ecb_encrypt_it接口

函数原型	hal_status_t hal_aes_ecb_encrypt_it(aes_handle_t *p_aes, uint32_t *p_plain_data, uint32_t number, uint32_t *p_cypher_data)
功能说明	ECB模式加密，中断方式。

输入参数	p_aes：指向aes_handle_t结构体变量的指针。 p_plain_data：指向待加密数据的指针。 number：待加密数据的长度，单位byte，最大取值为32768，且必须为16的整数倍。 p_cypher_data：指向存储加密结果的内存空间的指针。
返回值	HAL状态。
备注	待加密数据流超过最大长度时，可分多次加密。

2.13.4.10 hal_aes_ecb_decrypt_it

表 2-157 hal_aes_ecb_decrypt_it接口

函数原型	hal_status_t hal_aes_ecb_decrypt_it(aes_handle_t *p_aes, uint32_t *p_cypher_data, uint32_t number, uint32_t *p_plain_data)
功能说明	ECB模式解密，中断方式。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。 p_cypher_data：指向待解密数据的指针。 number：待解密数据的长度，单位byte，最大取值为32768，且必须为16的整数倍。 p_plain_data：指向存储解密结果的内存空间的指针。
返回值	HAL状态。
备注	待解密数据流超过最大长度时，可分多次解密。

2.13.4.11 hal_aes_cbc_encrypt_it

表 2-158 hal_aes_cbc_encrypt_it接口

函数原型	hal_status_t hal_aes_cbc_encrypt_it(aes_handle_t *p_aes, uint32_t *p_plain_data, uint32_t number, uint32_t *p_cypher_data)
功能说明	CBC模式加密，中断方式。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。 p_plain_data：指向待加密数据的指针。 number：待加密数据的长度，单位byte，最大取值为32768，且必须为16的整数倍。 p_cypher_data：指向存储加密结果的内存空间的指针。
返回值	HAL状态。
备注	待加密数据流超过最大长度时，可分多次加密，从第二次开始需重载初始化向量（p_aes->init.p_init_vector）为上一次加密结果的最后16 bytes数据。

2.13.4.12 hal_aes_cbc_decrypt_it

表 2-159 hal_aes_cbc_decrypt_it接口

函数原型	hal_status_t hal_aes_cbc_decrypt_it(aes_handle_t *p_aes, uint32_t *p_cypher_data, uint32_t number, uint32_t *p_plain_data)
功能说明	CBC模式解密，中断方式。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。 p_cypher_data：指向待解密数据的指针。 number：待解密数据的长度，单位byte，最大取值为32768，且必须为16的整数倍。 p_plain_data：指向存储解密结果的内存空间的指针。
返回值	HAL状态。
备注	待解密数据流超过最大长度时，可分多次解密，从第二次开始需重载初始化向量（p_aes->init.p_init_vector）为上一次待解密数据的最后16 bytes数据。

2.13.4.13 hal_aes_abort

表 2-160 hal_aes_abort接口

函数原型	hal_status_t hal_aes_abort(aes_handle_t *p_aes)
功能说明	中止加解密，轮询方式。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	HAL状态。
备注	该接口用于中止非轮询方式的加解密过程。轮询式接口，中止操作完成后返回状态。

2.13.4.14 hal_aes_abort_it

表 2-161 hal_aes_abort_it接口

函数原型	hal_status_t hal_aes_abort_it(aes_handle_t *p_aes)
功能说明	中止加解密，中断方式。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	HAL状态。
备注	该接口用于中止非轮询方式的加解密过程。非轮询式接口，立即返回状态，中止操作完成后调用hal_aes_abort_cplt_callback()。

2.13.4.15 hal_aes_irq_handler

表 2-162 hal_aes_irq_handler接口

函数原型	void hal_aes_irq_handler(aes_handle_t *p_aes)
功能说明	处理AES中断请求。

输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	无
备注	

2.13.4.16 hal_aes_done_callback

表 2-163 hal_aes_done_callback接口

函数原型	void hal_aes_done_callback(aes_handle_t *p_aes)
功能说明	AES加解密完成回调函数。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	无
备注	中断或DMA方式下加解密操作完成时该接口会被调用。该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.13.4.17 hal_aes_error_callback

表 2-164 hal_aes_error_callback接口

函数原型	void hal_aes_error_callback(aes_handle_t *p_aes)
功能说明	AES加解密错误回调函数。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	无
备注	中断或DMA方式下加解密操作出现错误时该接口会被调用。该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.13.4.18 hal_aes_abort_cplt_callback

表 2-165 hal_aes_abort_cplt_callback接口

函数原型	void hal_aes_abort_cplt_callback(aes_handle_t *p_aes)
功能说明	AES加解密中止回调函数。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	无
备注	中断或DMA方式下加解密操作被中止时该接口会被调用。该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.13.4.19 hal_aes_get_state

表 2-166 hal_aes_get_state接口

函数原型	hal_aes_state_t hal_aes_get_state(aes_handle_t *p_aes)

功能说明	获取AES运行状态。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	AES运行状态，该参数的取值可能是下列值中的任意一个： • HAL_AES_STATE_RESET（未初始化） • HAL_AES_STATE_READY（已初始化且空闲） • HAL_AES_STATE_BUSY（忙） • HAL_AES_STATE_ERROR（运行错误） • HAL_AES_STATE_TIMEOUT（超时） • HAL_AES_STATE_SUSPENDED（已挂起）
备注	

2.13.4.20 hal_aes_get_error

表 2-167 hal_aes_get_error接口

函数原型	uint32 hal_aes_get_error(aes_handle_t *p_aes)
功能说明	获取AES错误码。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	AES错误码，该参数的取值可能是下列值中的任意一个： • HAL_AES_ERROR_NONE（无错误） • HAL_AES_ERROR_TIMEOUT（超时） • HAL_AES_ERROR_TRANSFER（传输错误） • HAL_AES_ERROR_INVALID_PARAM（非法参数）
备注	

2.13.4.21 hal_aes_set_timeout

表 2-168 hal_aes_set_timeout接口

函数原型	void hal_aes_set_timeout(aes_handle_t *p_aes)
功能说明	设置AES操作超时时间。
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	无
备注	

2.13.4.22 hal_aes_suspend_reg

表 2-169 hal_aes_suspend_reg接口

函数原型	hal_status_t hal_aes_suspend_reg(aes_handle_t *p_aes)
功能说明	睡眠时挂起AES配置相关的寄存器
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	HAL状态

备注	

2.13.4.23 hal_aes_resume_reg

表 2-170 hal_aes_resume_reg接口

函数原型	hal_status_t hal_aes_resume_reg(aes_handle_t *p_aes)
功能说明	唤醒时恢复AES配置相关的寄存器
输入参数	p_aes：指向aes_handle_t结构体变量的指针。
返回值	HAL状态。
备注	

2.14 HAL HMAC通用驱动

2.14.1 HMAC驱动功能

HMAC（Hash-based Message Authentication Code）外设的HAL驱动主要实现了以下功能：

- 兼容SHA-256哈希算法。
- 支持用户自定义哈希初始值。
- 支持多种密钥加载模式：MCU，DMA，KPORT。
- 支持反DPA攻击。
- 支持轮询、中断、DMA三种计算方式。
- 支持回调函数。

2.14.2 如何使用HMAC驱动

2.14.2.1 初始化

HMAC HAL驱动初始化的使用方法如下：

1. 声明一个hmac_handle_t句柄结构，例如hmac_handle_t hmac_handle。
2. 重写hal_hmac_msp_init()接口以初始化HMAC底层资源。若开发者要使用中断或DMA模式，则需通过调用相关的NVIC接口来配置：
   - 通过hal_nvic_set_priority()配置HMAC中断优先级。
   - 通过hal_nvic_enable_irq()使能HMAC中断。
3. 配置hmac_handle句柄p_instance和init，包括HMAC外设实例、操作模式（HMAC或SHA）、密钥指针、用户初始化哈希值指针以及安全模式。
4. 调用hal_hmac_init()初始化HMAC寄存器。

2.14.2.2 使用SHA-256计算消息摘要

SHA-256支持轮询、中断和DMA三种计算模式。这三种模式的区别在于计算数据的加载方式和计算完成的判断方式：轮询模式需要循环检测完成状态，而中断和DMA模式则通过计算完成中断来实现。三种计算模式的具体使用方法如下：

轮询模式IO操作

1. 开发者可根据需要重载自定义哈希初始值p_user_hash。初始化HMAC时，需要禁止中断，禁止DMA，选择SHA模式；使用hal_hmac_sha256_digest()计算消息摘要。若数据流太长无法一次性计算完成，则需将数据分段处理。对于非首段数据的处理，需将自定义的哈希初始值p_user_hash重载为上一次的计算结果。

2. 直至计算完成或超时返回。若返回错误，可调用hal_hmac_get_error()查看错误代码；计算数据较多时可重复步骤1。

中断模式IO操作

1. 开发者可根据需要实现hal_hmac_done_callback()和hal_hmac_error_callback()。

2. 开发者可根据需要重载自定义哈希初始值p_user_hash。初始化HMAC时，需要使能中断，禁止DMA，选择SHA模式；使用hal_hmac_sha256_digest()计算摘要。若数据流太长无法一次性计算完成，则需将数据分段处理。对于非首段数据的处理，需将自定义的哈希初始值p_user_hash重载为上一次的计算结果。

3. 计算完成时hal_hmac_done_callback()会被调用，计算出错时hal_hmac_error_callback()会被调用；计算数据较多时可重复步骤2。

DMA模式IO操作

1. 开发者可根据需要实现hal_hmac_done_callback()和hal_hmac_error_callback()。

2. 开发者可根据需要重载自定义哈希初始值p_user_hash。初始化HMAC时，需要禁止中断，使能DMA，选择SHA模式；使用hal_hmac_sha256_digest()计算摘要。若数据流太长无法一次性计算完成，则需将数据分段处理。对于非首段数据的处理，需将自定义的哈希初始值p_user_hash重载为上一次的计算结果。

3. 计算完成时hal_hmac_done_callback()会被调用，计算出错时hal_hmac_error_callback()会被调用；加解密数据较多时可重复步骤2。

2.14.2.3 使用HMAC计算消息签名

HMAC支持轮询、中断和DMA三种计算模式。这三种模式的区别在于计算数据的加载方式和计算完成的判断方式：轮询模式需要循环检测完成状态，而中断和DMA模式则通过计算完成中断来实现。三种计算模式的具体使用方法如下：

轮询模式IO操作

1. 开发者可根据需要重载自定义哈希初始值p_user_hash和密钥p_key。初始化HMAC时，需要禁止中断，禁止DMA，选择HMAC模式；使用hal_hmac_sha256_digest()计算签名。
2. 计算完成时hal_hmac_done_callback()会被调用，计算出错时hal_hmac_error_callback()会被调用；加解密数据较多时可重复步骤1。
3. 直至计算完成或超时返回。若返回错误，可调用hal_hmac_get_error()查看错误代码。

中断模式IO操作

1. 开发者可根据需要实现hal_hmac_done_callback()和hal_hmac_error_callback()。
2. 开发者可根据需要重载自定义哈希初始值p_user_hash和密钥p_key。初始化HMAC时，需要使能中断，禁止DMA，选择HMAC模式；使用hal_hmac_sha256_digest()计算签名。
3. 计算完成时hal_hmac_done_callback()会被调用，计算出错时hal_hmac_error_callback()会被调用。

DMA模式IO操作

1. 开发者可根据需要实现hal_hmac_done_callback()和hal_hmac_error_callback()。
2. 开发者可根据需要重载自定义哈希初始值p_user_hash和密钥p_key。初始化HMAC时，需要禁止中断，使能DMA，选择HMAC模式；使用hal_hmac_sha256_digest()计算签名。
3. 每次计算完成时hal_hmac_done_callback()会被调用，计算出错时hal_hmac_error_callback()会被调用。

2.14.3 HMAC驱动的结构体

2.14.3.1 hmac_init_t

HMAC驱动的初始化结构体hmac_init_t的定义如下：

表 2-171 hmac_init_t结构体

数据域	域段描述	取值
uint32_t mode	指定计算模式。	该参数的取值可以是下列值中的任意一个： • HMAC_MODE_SHA（SHA模式） • HMAC_MODE_HMAC（HMAC模式）
uint32_t *p_key	指定密钥。	指向密钥的指针。
uint32_t *p_user_hash	指定自定义初始哈希值。	指向初始哈希值的指针。
uint32_t dpa_mode	安全模式使能标志。	该参数的取值可以是下列值中的任意一个： • ENABLE（使能） • DISABLE（禁能）
uint32_t key_fetch_type	Key的来源	该参数的取值可以是下列值中的任意一个： • HAL_HMAC_KEYTYPE_MCU（MCU配置） • HAL_HMAC_KEYTYPE_AHB（来自AHB） • HAL_HMAC_KEYTYPE_KRAM（来自KERAM）

数据域	域段描述	取值
uint32_t enable_irq	是否使能中断模式	该参数的取值可以是下列值中的任意一个： • HAL_HMAC_ENABLE_IRQ（使能IRQ） • HAL_HMAC_DISABLE_IRQ（禁能IRQ）
uint32_t enable_dma_mode	是否使能DMA模式	该参数的取值可以是下列值中的任意一个： • HAL_HMAC_ENABLE_DMA（使能DMA） • HAL_HMAC_DISABLE_DMA（禁能DMA）

2.14.3.2 hmac_handle_t

HMAC驱动的句柄结构体hmac_handle_t的定义如下：

表 2-172 hmac_handle_t结构体

数据域	域段描述	取值
hmac_regs_t * p_instance	HMAC外设实例。	该参数的取值为HMAC。
hmac_init_t init	初始化结构体。	参考hmac_init_t结构体。
uint32_t * p_message	指向待计算的消息缓冲区的指针(无需开发者初始化)。	N/A
uint32_t * p_digest	指向计算结果缓冲区的指针(无需开发者初始化)。	N/A
uint32_t block_size	待计算消息块的大小(无需开发者初始化)。	该参数取值范围：1 ～ 512。
uint32_t block_count	待计算消息块的计数(无需开发者初始化)。	初始化为block_size，计算过程中递减到0。
uint32_t is_last_trans	最后计算块标志(无需开发者初始化)。	N/A
__IO hal_lock_t lock	HMAC锁(无需开发者初始化)。	N/A
__IO hal_hmac_state_t state	HMAC运行状态(无需开发者初始化)。	该参数的取值可以是下列值中的任意一个： • HAL_HMAC_STATE_RESET（未初始化） • HAL_HMAC_STATE_READY（已初始化且空闲） • HAL_HMAC_STATE_BUSY（忙） • HAL_HMAC_STATE_ERROR（错误） • HAL_HMAC_STATE_TIMEOUT（超时） • HAL_HMAC_STATE_SUSPENDED（已挂起）
__IO uint32_t error_code	HMAC错误码（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_HMAC_ERROR_NONE（无错误） • HAL_HMAC_ERROR_TIMEOUT（超时） • HAL_HMAC_ERROR_TRANSFER（传输错误 • HAL_HMAC_ERROR_INVALID_PARAM（非法参数）
uint32_t timeout	HMAC计算超时时间（无需开发者初始化）。	N/A
uint32_t retention[17]	保存HAMC寄存器信息（驱动负责管理，无需开发者初始化）。	N/A

2.14.4 HMAC驱动API描述

HMAC驱动的API主要包括：

表 2-173 HMAC驱动的APIs

API类别	API名称	描述
初始化	hal_hmac_init()	初始化HMAC外设，配置计算模式等参数。
	hal_hmac_deinit()	反初始化HMAC外设。
	hal_hmac_msp_init()	初始化HMAC外设所使用的NVIC中断、DMA通道。
	hal_hmac_msp_deinit()	反初始化HMAC外设所使用的NVIC中断、DMA通道。
IO操作	hal_hmac_sha256_digest()	通过配置enable_irq和enable_dma_mode来区分使用轮询、中断和DMA计算模式。 通过配置mode来区分SHA还是HMAC模式。
中断处理及回调函数	hal_hmac_irq_handler()	中断处理函数。
	hal_hmac_done_callback()	计算完成中断回调函数。
	hal_hmac_error_callback()	错误中断回调函数。
状态及错误	hal_hmac_get_state()	获取驱动运行状态。
	hal_hmac_get_error()	获取错误码。
控制	hal_hmac_set_timeout()	设置超时时间。
睡眠相关	hal_hmac_suspend_reg()	睡眠时挂起HMAC配置相关的寄存器。
	hal_hmac_resume_reg()	唤醒时恢复HMAC配置相关的寄存器。

下面章节将对各API进行详细描述。

2.14.4.1 hal_hmac_init

表 2-174 hal_hmac_init接口

函数原型	hal_status_t hal_hmac_init(hmac_handle_t *p_hmac)
功能说明	根据hmac_init_t结构体里的参数初始化HMAC外设。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针，该结构体变量包含指定的HMAC配置信息。
返回值	HAL状态。
备注	

2.14.4.2 hal_hmac_deinit

表 2-175 hal_hmac_deinit接口

函数原型	hal_status_t hal_hmac_deinit(hmac_handle_t *p_hmac)
功能说明	反初始化HMAC外设。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针，包含指定的寄存器基址。

返回值	HAL状态。
备注	

2.14.4.3 hal_hmac_msp_init

表 2-176 hal_hmac_msp_init接口

函数原型	void hal_hmac_msp_init(hmac_handle_t *p_hmac)
功能说明	初始化HMAC外设所使用的NVIC中断等配置。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成NVIC中断、DMA通道的初始化。

2.14.4.4 hal_hmac_msp_deinit

表 2-177 hal_hmac_msp_deinit接口

函数原型	void hal_hmac_msp_deinit(hmac_handle_t *p_hmac)
功能说明	反初始化HMAC外设所使用的NVIC中断等配置。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成NVIC中断、DMA通道的反初始化。

2.14.4.5 hal_hmac_sha256_digest

表 2-178 hal_hmac_sha256_digest接口

函数原型	hal_status_t hal_hmac_sha256_digest(hmac_handle_t *p_hmac, uint32_t *p_message, uint32_t number, uint32_t *p_digest, uint32_t timeout)
功能说明	基于SHA-256计算消息摘要/签名，支持轮询、中断、DMA方式。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。 p_message：指向待计算消息的指针。 number：待计算消息的长度，单位byte，最大取值为32768，且必须为64的整数倍。 p_digest：指向计算结果的指针。 timeout：计算超时时间，单位ms。
返回值	HAL状态。
备注	待计算消息数据量超过最大长度时，可分多次计算。

2.14.4.6 hal_hmac_irq_handler

表 2-179 hal_hmac_irq_handler接口

函数原型	void hal_hmac_irq_handler(hmac_handle_t *p_hmac)

功能说明	处理HMAC中断请求。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。
返回值	无
备注	

2.14.4.7 hal_hmac_done_callback

Table 2-180: hal_hmac_done_callback接口

函数原型	void hal_hmac_done_callback(hmac_handle_t *p_hmac)
功能说明	HMAC计算完成回调函数。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。
返回值	无
备注	中断或DMA方式下计算操作完成时该接口会被调用。该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.14.4.8 hal_hmac_error_callback

表 2-181 hal_hmac_error_callback接口

函数原型	void hal_hmac_error_callback(hmac_handle_t *p_hmac)
功能说明	HMAC计算错误回调函数。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。
返回值	无
备注	中断或DMA方式下计算操作错误时该接口会被调用。该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.14.4.9 hal_hmac_get_state

表 2-182 hal_hmac_get_state接口

函数原型	hal_hmac_state_t hal_hmac_get_state(hmac_handle_t *p_hmac)
功能说明	获取HMAC运行状态。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。
返回值	HMAC运行状态，该参数的取值可能是下列值中的任意一个： • HAL_HMAC_STATE_RESET（未初始化） • HAL_HMAC_STATE_READY（已初始化且空闲） • HAL_HMAC_STATE_BUSY（忙） • HAL_HMAC_STATE_ERROR（错误） • HAL_HMAC_STATE_TIMEOUT（超时） • HAL_HMAC_STATE_SUSPENDED（已挂起）
备注	

2.14.4.10 hal_hmac_get_error

表 2-183 hal_hmac_get_error接口

函数原型	unit32_t hal_hmac_get_error(hmac_handle_t *p_hmac)
功能说明	获取HMAC错误码。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。
返回值	HMAC错误码，该参数的取值可以是下列值中的任意一个： • HAL_HMAC_ERROR_NONE（无错误） • HAL_HMAC_ERROR_TIMEOUT（超时） • HAL_HMAC_ERROR_TRANSFER（传输错误） • HAL_HMAC_ERROR_INVALID_PARAM（非法参数）
备注	

2.14.4.11 hal_hmac_set_timeout

表 2-184 hal_hmac_set_timeout接口

函数原型	void hal_hmac_set_timeout(hmac_handle_t *p_hmac)
功能说明	设置HMAC超时时间。
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。
返回值	无
备注	

2.14.4.12 hal_hmac_suspend_reg

表 2-185 hal_hmac_suspend_reg接口

函数原型	hal_status_t hal_hmac_suspend_reg(hmac_handle_t *p_hmac)
功能说明	睡眠时挂起HMAC配置相关的寄存器
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。
返回值	HAL状态
备注	

2.14.4.13 hal_hmac_resume_reg

表 2-186 hal_hmac_resume_reg接口

函数原型	hal_status_t hal_hmac_resume_reg(hmac_handle_t *p_hmac)
功能说明	唤醒时恢复HMAC配置相关的寄存器
输入参数	p_hmac：指向hmac_handle_t结构体变量的指针。
返回值	HAL状态
备注	

2.15 HAL PKC通用驱动

2.15.1 PKC驱动功能

PKC（Public Key Cipher）外设的HAL驱动主要实现了以下功能：

- 支持FIPS-180-3标准，P-256椭圆曲线的标量乘法运算。
- 支持256 bits到2048 bits长度可配置的蒙哥马利模乘运算。
- 支持256 bits到2048 bits长度可配置的蒙哥马利部分求逆运算。
- 支持256 bits到2048 bits长度可配置的模加运算。
- 支持256 bits到2048 bits长度可配置的模减运算。
- 支持256 bits到2048 bits长度可配置的模比较运算。
- 支持256 bits到2048 bits长度可配置的模左移运算。
- 支持256 bits到1024 bits长度可配置的大数乘法运算。
- 支持256 bits到2048 bits长度可配置的大数加法运算。
- 支持硬件开启Dummy乘法运算。
- 支持随机时钟加扰功能。
- 使用1280 bytes大小，32 bits数据位宽的单口RAM，支持MCU方式读写该RAM。
- 支持轮询、中断两种运算方式。
- 支持中止中断方式下的运算操作。
- 支持运算完成、错误、溢出、中止完成的中断回调函数。
- 支持获取驱动的运行状态及错误码。
- 支持超时时间设置。

2.15.2 如何使用PKC驱动

PKC驱动的使用方法如下：

1. 声明一个pkc_handle_t句柄结构体变量，例如pkc_handle_t pkc_handle。
2. 重写hal_pkc_msp_init()以初始化PKC底层资源：
   (1) 可调用__HAL_PKC_RESET()复位PKC模块。
   (2) 如果开发者要使用中断方式的API，则需通过调用相关的NVIC接口来配置：
      - 调用hal_nvic_set_priority()配置PKC中断优先级。
      - 调用hal_nvic_enable_irq()使能PKC的NVIC中断。

3. 配置pkc_handle中init初始化结构体中的数据位宽、安全模式、注册随机数产生函数和ECC椭圆参数等。

4. 调用hal_pkc_init()配置PKC寄存器，配置过程中hal_pkc_init()会自动调用开发者重写的hal_pkc_msp_init()函数初始化PKC所使用的NVIC中断等底层资源。

5. 开发者可根据实际应用调用相应的API完成相应的数学运算，PKC的HAL驱动提供轮询及中断两种运算方式。

2.15.3 PKC驱动结构和定义

2.15.3.1 ecc_point_t

PKC驱动的ECC点描述结构体ecc_point_t的定义如下：

表 2-187 ecc_point_t结构体

数据域	域段描述	取值
uint32_t X[ECC_U32_LENGTH]	ECC点的X轴坐标。	0 ～ 2^256 - 1
uint32_t Y[ECC_U32_LENGTH]	ECC点的Y轴坐标。	0 ～ 2^256 - 1

2.15.3.2 ecc_curve_init_t

PKC驱动的ECC椭圆曲线描述结构体ecc_curve_init_t的定义如下：

表 2-188 ecc_curve_init_t结构体

数据域	域段描述	取值
uint32_t A[ECC_U32_LENGTH]	操作数A数组。	0 ～ 2^256 - 1
uint32_t B[ECC_U32_LENGTH]	操作数B数组。	0 ～ 2^256 - 1
uint32_t P[ECC_U32_LENGTH]	质数P数组。	0 ～ 2^256 - 1
uint32_t PRSquare[ECC_U32_LENGTH]	PRSquare数组。	R^2 mod P，其中R = 2^256。
uint32_t ConstP	ConstP数组。	质数P的蒙哥马利常数。
uint32_t N[ECC_U32_LENGTH]	质数N数组。	0 ～ 2^256 - 1
uint32_t NRSquare[ECC_U32_LENGTH]	NRSquare数组。	R^2 mod N，其中 R = 2^256。
uint32_t ConstN	ConstN数组。	质数N的蒙哥马利常数。
uint32_t H	参数H。	0 ～ 2^32 - 1
ll_ecc_point_t G	ECC点。	参考ll_ecc_point_t。

2.15.3.3 pkc_init_t

PKC驱动的初始化结构体pkc_init_t的定义如下：

表 2-189 pkc_init_t结构体

数据域	域段描述	取值
ecc_curve_init_t *p_ecc_curve	指向椭圆曲线描述类型的指针。	参考ecc_curve_init_t结构体。
uint32_t data_bits	计算数据位宽。	256 ～ 2048
uint32_t secure_mode	是否启用安全模式。	该参数的取值可以是下列值中的任意一个： • PKC_SECURE_MODE_DISABLE（禁用） • PKC_SECURE_MODE_ENABLE（启用）
uint32_t (*random_func)(void)	指定的随机数生成函数。	函数指针。

2.15.3.4 pkc_handle_t

PKC驱动的句柄结构体pkc_handle_t的定义如下：

表 2-190 pkc_handle_t结构体

数据域	域段描述	取值
pkc_regs_t *p_instance	PKC外设实例。	该参数的取值是：PKC。
pkc_init_t init	初始化结构体。	参考pkc_init_t结构体。
void *p_result	指向数据运算结果的指针。	开发者需要在每次调用运算API前指定该参数。
uint32_t *k_kout	指向蒙哥马利反演运算结果的指针。	开发者需要在每次调用运算API前指定该参数。
uint32_t shift_count	移位计数（驱动负责管理，无需开发者初始化）。	N/A
__IO hal_lock_t lock	PKC锁（驱动负责管理，无需开发者初始化）。	N/A
__IO hal_qspi_state_t state	PKC运行状态（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_PKC_STATE_RESET（未初始化） • HAL_PKC_STATE_READY（已初始化且空闲） • HAL_PKC_STATE_BUSY（忙） • HAL_PKC_STATE_ERROR（错误） • HAL_PKC_STATE_TIMEOUT（超时）
__IO uint32_t error_code	PKC错误码（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_PKC_ERROR_NONE（无错误） • HAL_PKC_ERROR_TIMEOUT（超时） • HAL_PKC_ERROR_TRANSFER（传输错误） • HAL_PKC_ERROR_OVERFLOW（溢出错误） • HAL_PKC_ERROR_INVALID_PARAM（非法参数） • HAL_PKC_ERROR_INVERSE_K（逆运算输出参数中K系数错误） • HAL_PKC_ERROR_IRREVERSIBLE（逆运算输入参数不可逆）

数据域	域段描述	取值
uint32_t timeout	PKC（超时时间无需开发者初始化）。	N/A
uint32_t retention[1]	保存PKC寄存器信息（驱动负责管理，无需开发者初始化）	N/A

2.15.3.5 pkc_ecc_point_multi_t

PKC驱动的ECC点乘运算结构体pkc_ecc_point_multi_t的定义如下：

表 2-191 pkc_ecc_point_multi_t结构体

数据域	域段描述	取值
uint32_t *p_K	输入参数K。	0 ～ 2^256 - 1
ecc_point_t *p_ecc_point	输入参数ECCPoint。	参考ecc_point_t

2.15.3.6 pkc_rsa_modular_exponent_t

PKC驱动的RSA模幂运算结构体pkc_rsa_modular_exponent_t的定义如下：

表 2-192 pkc_rsa_modular_exponent_t结构体

数据域	域段描述	取值
uint32_t *p_A	输入参数运算数A。	指向data_bits长度的数据。
uint32_t *p_B	输入参数运算数B。	指向data_bits长度的数据。
uint32_t *p_P	输入参数质数P。	指向data_bits长度的数据。
uint32_t *p_R2	输入参数R2。	R2 = R^2 mod P， R = 2^data_bits
uint32_t ConstP	输入参数ConstP。	质数P的蒙哥马利常数。

2.15.3.7 pkc_modular_add_t

PKC驱动的模加运算结构体pkc_modular_add_t的定义如下：

表 2-193 pkc_modular_add_t结构体

数据域	域段描述	取值
uint32_t *p_A	输入参数运算数A。	指向data_bits长度的数据。
uint32_t *p_B	输入参数运算数B。	指向data_bits长度的数据。
uint32_t *p_P	输入参数质数P。	指向data_bits长度的数据。

2.15.3.8 pkc_modular_sub_t

PKC驱动的模减运算结构体pkc_modular_sub_t的定义如下：

表 2-194 pkc_modular_sub_t结构体

数据域	域段描述	取值
uint32_t *p_A	输入参数运算数A。	指向data_bits长度的数据。
uint32_t *p_B	输入参数运算数B。	指向data_bits长度的数据。
uint32_t *p_P	输入参数质数P。	指向data_bits长度的数据。

2.15.3.9 pkc_modular_shift_t

PKC驱动的模左移运算结构体pkc_modular_shift_t的定义如下：

表 2-195 pkc_modular_shift_t结构体

数据域	域段描述	取值
uint32_t *p_A	输入参数运算数A。	指向data_bits长度的数据。
uint32_t shift_bits	输入参数左移位数。	1 ～ data_bits
uint32_t *p_P	输入参数质数P。	指向data_bits长度的数据。

2.15.3.10 pkc_modular_compare_t

PKC驱动的模比较运算结构体pkc_modular_compare_t的定义如下：

表 2-196 pkc_modular_compare_t结构体

数据域	域段描述	取值
uint32_t *p_A	输入参数运算数A。	指向data_bits长度的数据。
uint32_t *p_P	输入参数质数P。	指向data_bits长度的数据。

2.15.3.11 pkc_montgomery_multi_t

PKC驱动的蒙哥马利乘运算结构体pkc_montgomery_multi_t的定义如下：

表 2-197 pkc_montgomery_multi_t结构体

数据域	域段描述	取值
uint32_t *p_A	输入参数运算数A。	指向data_bits长度的数据。
uint32_t *p_B	输入参数运算数B。	指向data_bits长度的数据。
uint32_t *p_P	输入参数质数P。	指向data_bits长度的数据。
uint32_t ConstP	输入参数ConstP。	质数P的蒙哥马利常数。

2.15.3.12 pkc_montgomery_inversion_t

PKC驱动的蒙哥马利逆运算结构体pkc_montgomery_inversion_t的定义如下：

表 2-198 pkc_montgomery_inversion_t结构体

数据域	域段描述	取值
uint32_t *p_A	输入参数运算数A。	指向data_bits长度的数据。
uint32_t *p_P	输入参数质数P。	指向data_bits长度的数据。

2.15.3.13 pkc_big_number_multi_t

PKC驱动的大数乘运算结构体pkc_big_number_multi_t的定义如下：

表 2-199 pkc_big_number_multi_t结构体

数据域	域段描述	取值
uint32_t *p_A	输入参数运算数A。	指向data_bits长度的数据。
uint32_t *p_B	输入参数运算数B。	指向data_bits长度的数据。

2.15.3.14 pkc_big_number_add_t

PKC驱动的大数加运算结构体pkc_big_number_add_t的定义如下：

表 2-200 pkc_big_number_add_t结构体

数据域	域段描述	取值
uint32_t *p_A	输入参数运算数A。	指向data_bits长度的数据。
uint32_t *p_B	输入参数运算数B。	指向data_bits长度的数据。

2.15.4 PKC驱动API描述

PKC驱动的API主要包括：

表 2-201 PKC驱动的APIs

API类别	API名称	描述
初始化	hal_pkc_init()	初始化PKC外设，配置数据位宽等参数。
	hal_pkc_deinit()	反初始化PKC外设。
	hal_pkc_msp_init()	初始化PKC外设所使用的NVIC中断。
	hal_pkc_msp_deinit()	反初始化PKC外设所使用的NVIC中断。
IO操作	hal_pkc_rsa_modular_exponent()	RSA模幂运算，轮询方式。
	hal_pkc_ecc_point_multi()	ECC点乘运算，轮询方式。
	hal_pkc_modular_add()	模加运算，轮询方式。
	hal_pkc_modular_sub()	模减运算，轮询方式。
	hal_pkc_modular_left_shift()	模左移运算，轮询方式。
	hal_pkc_modular_compare()	模比较运算，轮询方式。
	hal_pkc_montgomery_multi()	蒙哥马利乘运算，轮询方式。

API类别	API名称	描述
	hal_pkc_montgomery_inversion()	蒙哥马利逆运算，轮询方式。
	hal_pkc_big_number_multi()	大数乘运算，轮询方式。
	hal_pkc_big_number_add()	大数加运算，轮询方式。
	hal_pkc_ecc_point_multi_it()	ECC点乘运算，中断方式。
	hal_pkc_modular_add_it()	模加运算，中断方式。
	hal_pkc_modular_sub_it()	模减运算，中断方式。
	hal_pkc_modular_left_shift_it()	模左移运算，中断方式。
	hal_pkc_modular_compare_it()	模比较运算，中断方式。
	hal_pkc_montgomery_multi_it()	蒙哥马利乘运算，中断方式。
	hal_pkc_montgomery_inversion_it()	蒙哥马利逆运算，中断方式。
	hal_pkc_big_number_multi_it()	大数乘运算，中断方式。
	hal_pkc_big_number_add_it()	大数加运算，中断方式。
中断处理及回调函数	hal_pkc_irq_handler()	中断处理函数。
	hal_pkc_done_callback()	运算完成中断回调函数。
	hal_pkc_error_callback()	错误中断回调函数。
	hal_pkc_overflow_callback()	运算溢出中断回调函数。
状态及错误	hal_pkc_get_state()	获取驱动运行状态。
	hal_pkc_get_error()	获取错误码。
控制	hal_pkc_set_timeout()	设置超时时间。
睡眠相关	hal_pkc_suspend_reg()	睡眠时挂起PKC配置相关的寄存器。
	hal_pkc_resume_reg()	唤醒时恢复PKC配置相关的寄存器。

下面章节将对各API进行详细描述。

2.15.4.1 hal_pkc_init

表 2-202 hal_pkc_init接口

函数原型	hal_status_t hal_pkc_init(pkc_handle_t *p_pkc)
功能说明	根据pkc_init_t里的参数初始化PKC外设和初始化关联句柄。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针，该结构体变量包含指定的PKC的配置信息。
返回值	HAL状态。
备注	

2.15.4.2 hal_pkc_deinit

表 2-203 hal_pkc_deinit接口

函数原型	hal_status_t hal_pkc_deinit(pkc_handle_t *p_pkc)
功能说明	反初始化PKC外设。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针，该结构体变量包含指定的PKC的配置信息。
返回值	HAL状态。
备注	

2.15.4.3 hal_pkc_msp_init

表 2-204 hal_pkc_msp_init接口

函数原型	void hal_pkc_msp_init(pkc_handle_t *p_pkc)
功能说明	初始化PKC外设所使用的NVIC中断等配置。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成NVIC中断的初始化。

2.15.4.4 hal_pkc_msp_deinit

表 2-205 hal_pkc_msp_deinit接口

函数原型	void hal_pkc_msp_deinit(pkc_handle_t *p_pkc)
功能说明	反初始化PKC外设所使用的NVIC中断等配置。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成NVIC中断的反初始化。

2.15.4.5 hal_pkc_rsa_modular_exponent

表 2-206 hal_pkc_rsa_modular_exponent接口

函数原型	hal_status_t hal_pkc_rsa_modular_exponent(pkc_handle_t *p_pkc, pkc_rsa_modular_exponent_t *p_input, uint32_t timeout)
功能说明	实现RSA算法中模幂运算：Result = A^B mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，轮询方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_rsa_modular_exponent_t结构体变量的指针。 timeout：超时时间。
返回值	HAL状态。

备注	

2.15.4.6 hal_pkc_ecc_point_multi

表 2-207 hal_pkc_ecc_point_multi接口

函数原型	hal_status_t hal_pkc_ecc_point_multi(pkc_handle_t *p_pkc, pkc_ecc_point_multi_t *p_input, uint32_t timeout)
功能说明	实现ECC椭圆算法中点乘运算：Result = K * Point，运算数最大位数2048 bits，结果数最大位数2048 bits，轮询方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_ecc_point_multi_t结构体变量的指针。 timeout：超时时间。
返回值	HAL状态。
备注	

2.15.4.7 hal_pkc_ecc_point_multi_it

表 2-208 hal_pkc_ecc_point_multi_it接口

函数原型	hal_status_t hal_pkc_ecc_point_multi_it(pkc_handle_t *p_pkc, pkc_ecc_point_multi_t *p_input)
功能说明	实现ECC椭圆算法中点乘运算：Result = K * Point，运算数最大位数2048 bits，结果数最大位数2048 bits，中断方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_ecc_point_multi_t结构体变量的指针。
返回值	HAL状态。
备注	

2.15.4.8 hal_pkc_modular_add

表 2-209 hal_pkc_modular_add接口

函数原型	hal_status_t hal_pkc_modular_add(pkc_handle_t *p_pkc, pkc_modular_add_t *p_input, uint32_t timeout)
功能说明	实现模加运算：Result = (A + B) mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，轮询方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_modular_add_t结构体变量的指针。 timeout：超时时间。
返回值	HAL状态。
备注	

2.15.4.9 hal_pkc_modular_add_it

表 2-210 hal_pkc_modular_add_it接口

函数原型	hal_status_t hal_pkc_modular_add_it(pkc_handle_t *p_pkc, pkc_modular_add_t *p_input)
功能说明	实现模加运算：Result = (A + B) mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，中断方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_modular_add_t结构体变量的指针。
返回值	HAL状态。
备注	

2.15.4.10 hal_pkc_modular_sub

表 2-211 hal_pkc_modular_sub接口

函数原型	hal_status_t hal_pkc_modular_sub(pkc_handle_t *p_pkc, pkc_modular_sub_t *p_input, uint32_t timeout)
功能说明	实现模减运算：Result = (A - B) mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，轮询方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_modular_sub_t结构体变量的指针。 timeout：超时时间。
返回值	HAL状态。
备注	

2.15.4.11 hal_pkc_modular_sub_it

表 2-212 hal_pkc_modular_sub_it接口

函数原型	hal_status_t hal_pkc_modular_sub_it(pkc_handle_t *p_pkc, pkc_modular_sub_t *p_input)
功能说明	实现模减运算：Result = (A - B) mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，中断方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_modular_sub_t结构体变量的指针。
返回值	HAL状态。
备注	

2.15.4.12 hal_pkc_modular_left_shift

表 2-213 hal_pkc_modular_left_shift接口

函数原型	hal_status_t hal_pkc_modular_left_shift(pkc_handle_t *p_pkc, pkc_modular_shift_t *p_input, uint32_t timeout)
功能说明	实现模左移运算：Result = (A << ShiftBits) mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，轮询方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_modular_shift_t结构体变量的指针。 timeout：超时时间。
返回值	HAL状态。
备注	

2.15.4.13 hal_pkc_modular_left_shift_it

表 2-214 hal_pkc_modular_left_shift_it接口

函数原型	hal_status_t hal_pkc_modular_left_shift_it(pkc_handle_t *p_pkc, pkc_modular_shift_t *p_input)
功能说明	实现模左移运算：Result = (A << ShiftBits) mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，中断方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_modular_shift_t结构体变量的指针。
返回值	HAL状态。
备注	

2.15.4.14 hal_pkc_modular_compare

表 2-215 hal_pkc_modular_compare接口

函数原型	hal_status_t hal_pkc_modular_compare(pkc_handle_t *p_pkc, pkc_modular_compare_t *p_input, uint32_t timeout)
功能说明	实现模比较运算：Result = A mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，轮询方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_modular_compare_t结构体变量指针。 timeout：超时时间。
返回值	HAL状态。
备注	

2.15.4.15 hal_pkc_modular_compare_it

表 2-216 hal_pkc_modular_compare_it接口

函数原型	hal_status_t hal_pkc_modular_compare_it(pkc_handle_t *p_pkc, pkc_modular_compare_t *p_input)
功能说明	实现模比较运算：Result = A mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，中断方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_modular_compare_t结构体变量指针。
返回值	HAL状态。
备注	

2.15.4.16 hal_pkc_montgomery_multi

表 2-217 hal_pkc_montgomery_multi接口

函数原型	hal_status_t hal_pkc_montgomery_multi(pkc_handle_t *p_pkc, pkc_montgomery_multi_t *p_input, uint32_t timeout)
功能说明	实现模乘运算：Result = A * B mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，轮询方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_montgomery_multi_t结构体变量的指针。 timeout：超时时间。
返回值	HAL状态。
备注	

2.15.4.17 hal_pkc_montgomery_multi_it

表 2-218 hal_pkc_montgomery_multi_it接口

函数原型	hal_status_t hal_pkc_montgomery_multi_it(pkc_handle_t *p_pkc, pkc_montgomery_multi_t *p_input)
功能说明	实现模乘运算：Result = A * B mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，中断方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_montgomery_multi_t结构体变量的指针。
返回值	HAL状态。
备注	

2.15.4.18 hal_pkc_montgomery_inversion

表 2-219 hal_pkc_montgomery_inversion接口

函数原型	hal_status_t hal_pkc_montgomery_inversion(pkc_handle_t *p_pkc, pkc_montgomery_inversion_t *p_input, uint32_t *p_K, uint32_t timeout)
功能说明	实现模逆运算：Result = A^(-1) mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，轮询方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_montgomery_inversion_t结构体变量的指针。 p_K：指向输出参数K的指针。 timeout：超时时间。
返回值	HAL状态。
备注	

2.15.4.19 hal_pkc_montgomery_inversion_it

表 2-220 hal_pkc_montgomery_inversion_it接口

函数原型	hal_status_t hal_pkc_montgomery_inversion_it(pkc_handle_t *p_pkc, pkc_montgomery_inversion_t *p_input, uint32_t *p_K)
功能说明	实现模逆运算：Result = A^(-1) mod P，运算数最大位数2048 bits，结果数最大位数2048 bits，中断方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_montgomery_inversion_t结构体变量的指针。 p_K：指向输出参数K的指针。
返回值	HAL状态。
备注	

2.15.4.20 hal_pkc_big_number_multi

表 2-221 hal_pkc_big_number_multi接口

函数原型	hal_status_t hal_pkc_big_number_multi(pkc_handle_t *p_pkc, pkc_big_number_multi_t *p_input, uint32_t timeout)
功能说明	实现大数乘运算：Result = A * B，运算数最大位数1024 bits，结果数最大位数2048 bits，轮询方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_big_number_multi_t结构体变量的指针。 timeout：超时时间。
返回值	HAL状态。

备注	

2.15.4.21 hal_pkc_big_number_multi_it

表 2-222 hal_pkc_big_number_multi_it接口

函数原型	hal_status_t hal_pkc_big_number_multi_it(pkc_handle_t *p_pkc, pkc_big_number_multi_t *p_input)
功能说明	实现大数乘运算：Result = A * B，运算数最大位数1024 bits，结果数最大位数2048 bits，中断方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_big_number_multi_t结构体变量的指针。
返回值	HAL状态。
备注	

2.15.4.22 hal_pkc_big_number_add

表 2-223 hal_pkc_big_number_add接口

函数原型	hal_status_t hal_pkc_big_number_add(pkc_handle_t *p_pkc, pkc_big_number_add_t *p_input, uint32_t timeout)
功能说明	实现大数加运算：Result = A + B，运算数最大位数2048 bits，结果数最大位数2048 bits，轮询方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_big_number_add_t结构体变量的指针。 timeout：超时时间。
返回值	HAL状态。
备注	

2.15.4.23 hal_pkc_big_number_add_it

表 2-224 hal_pkc_big_number_add_it接口

函数原型	hal_status_t hal_pkc_big_number_add_it(pkc_handle_t *p_pkc, pkc_big_number_add_t *p_input)
功能说明	实现大数加运算：Result = A + B，运算数最大位数2048 bits，结果数最大位数2048 bits，中断方式。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 p_input：指向输入参数pkc_big_number_add_t结构体变量的指针。
返回值	HAL状态。
备注	

2.15.4.24 hal_pkc_irq_handler

表 2-225 hal_pkc_irq_handler接口

函数原型	void hal_pkc_irq_handler(pkc_handle_t *p_pkc)
功能说明	处理PKC中断请求。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。
返回值	无
备注	

2.15.4.25 hal_pkc_done_callback

表 2-226 hal_pkc_done_callback接口

函数原型	void hal_pkc_done_callback(pkc_handle_t *p_pkc)
功能说明	PKC运算完成回调。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.15.4.26 hal_pkc_error_callback

表 2-227 hal_pkc_error_callback接口

函数原型	void hal_pkc_error_callback(pkc_handle_t *p_pkc)
功能说明	PKC运算错误回调。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.15.4.27 hal_pkc_overflow_callback

表 2-228 hal_pkc_overflow_callback接口

函数原型	void hal_pkc_overflow_callback(pkc_handle_t *p_pkc)
功能说明	PKC大数乘/加运算溢出回调。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.15.4.28 hal_pkc_get_state

表 2-229 hal_pkc_get_state接口

函数原型	hal_pkc_state_t hal_pkc_get_state(pkc_handle_t *p_pkc)
功能说明	获取PKC运行状态。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。
返回值	PKC运行状态，该参数的取值可以是下列值中的任意一个： • HAL_PKC_STATE_RESET（未初始化） • HAL_PKC_STATE_READY（已初始化且空闲） • HAL_PKC_STATE_BUSY（忙） • HAL_PKC_STATE_ERROR（错误） • HAL_PKC_STATE_TIMEOUT（超时）
备注	

2.15.4.29 hal_pkc_get_error

表 2-230 hal_pkc_get_error接口

函数原型	uint32_t hal_pkc_get_error(pkc_handle_t *p_pkc)
功能说明	获取PKC错误码。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。
返回值	PKC错误码，该参数的取值可以是下列值中的任意一个： • HAL_PKC_ERROR_NONE（无错误） • HAL_PKC_ERROR_TIMEOUT（超时） • HAL_PKC_ERROR_TRANSFER（传输错误） • HAL_PKC_ERROR_OVERFLOW（溢出错误） • HAL_PKC_ERROR_INVALID_PARAM（非法参数） • HAL_PKC_ERROR_INVERSE_K（逆运算输出参数中K系数错误） • HAL_PKC_ERROR_IRREVERSIBLE（逆运算输入参数不可逆）
备注	

2.15.4.30 hal_pkc_set_timeout

表 2-231 hal_pkc_set_timeout接口

函数原型	void hal_pkc_set_timeout(pkc_handle_t *p_pkc , uint32_t timeout)
功能说明	设置PKC运算超时时间。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针。 timeout：运算超时时间。
返回值	无
备注	

2.15.4.31 hal_pkc_suspend_reg

表 2-232 hal_pkc_suspend_reg接口

函数原型	hal_status_t hal_pkc_suspend_reg(pkc_handle_t *p_pkc)
功能说明	睡眠时挂起PKC配置相关的寄存器。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针
返回值	PKC运行状态，该参数的取值可以是下列值中的任意一个： • HAL_PKC_STATE_RESET（未初始化） • HAL_PKC_STATE_READY（已初始化且空闲） • HAL_PKC_STATE_BUSY（忙） • HAL_PKC_STATE_ERROR（错误） • HAL_PKC_STATE_TIMEOUT（超时）
备注	

2.15.4.32 hal_pkc_resume_reg

表 2-233 hal_pkc_resume_reg接口

函数原型	hal_status_t hal_pkc_resume_reg(pkc_handle_t *p_pkc)
功能说明	唤醒时恢复PKC配置相关的寄存器。
输入参数	p_pkc：指向pkc_handle_t结构体变量的指针
返回值	PKC运行状态，该参数的取值可以是下列值中的任意一个： • HAL_PKC_STATE_RESET（未初始化） • HAL_PKC_STATE_READY（已初始化且空闲） • HAL_PKC_STATE_BUSY（忙） • HAL_PKC_STATE_ERROR（错误） • HAL_PKC_STATE_TIMEOUT（超时）
备注	

2.16 HAL I2C通用驱动

2.16.1 I2C驱动功能

I2C（Inter-integrated circuit）外设的HAL驱动主要实现了以下功能：

- 支持标准模式（0～100 Kb/s）、快速模式（≤ 400 Kb/s）、快速+模式（≤ 1000 Kb/s）、高速模式（≤ 2.8 Mb/s）下的数据读写。
- 支持主设备或从设备模式的自动切换。
- 支持7位或10位地址模式配置。
- 支持7位或10位混合地址模式。
- 支持外部存储设备的读写操作。

- 支持轮询、中断、DMA三种IO操作方式。
- 支持中止中断及DMA方式下的数据收发/读写。
- 支持主设备及从设备模式下的发送完成、接收完成的中断回调函数。
- 支持Memory模式下的写入完成、读取完成的中断回调函数。
- 支持中止完成、IO错误的中断回调函数。
- 支持获取驱动的I2C模式、运行状态及错误码。

2.16.2 如何使用I2C驱动

I2C的HAL驱动的使用方法如下：

1. 定义一个i2c_handle_t句柄结构体变量，例如：i2c_handle_t i2c_handle（i2c_handle_t结构体由I2C的HAL驱动定义，开发者在使用时需要定义一个该结构体类型的变量）。
2. 重写hal_i2c_msp_init()接口以初始化I2C底层资源：
   (1) 配置I2C对应GPIO引脚的功能复用、使能上拉电阻。
   (2) 如果需要使用中断或DMA方式的IO操作接口，则需通过调用相关的NVIC接口来配置：
      - 调用hal_nvic_set_priority()配置I2C的中断优先级。
      - 调用hal_nvic_enable_irq()使能I2C的NVIC中断。
   (3) 如果需要使用DMA方式的IO操作接口，则还需要配置使用的DMA通道：
      - 定义用于发送/接收的dma_handle_t句柄结构体变量，如dma_handle_t dma_tx、dma_rx。
      - 配置DMA句柄dma_tx及dma_rx中的参数，如指定TX或RX通道。
      - 将I2C Handler结构体变量中的p_dmatx和p_dmarx指针分别指向已初始化的DMA句柄变量dma_tx和dma_rx。
      - 配置DMA的中断优先级、使能DMA的NVIC中断。
3. 配置I2C初始化结构体中的数据传输速率、本机设备地址、设备地址模式以及广播地址监测模式。
4. 调用hal_i2c_init()配置I2C寄存器，配置过程中hal_i2c_init()会自动调用开发者重写的hal_i2c_msp_init()函数初始化I2C所使用的GPIO等底层资源。
5. 对于I2C的IO读写或IO内存读写操作，I2C的HAL驱动提供三种操作方式的接口：轮询、中断及DMA。

2.16.2.1 轮询方式的IO读写操作

1. 作为主设备以轮询方式发送大量数据时使用hal_i2c_master_transmit()。
2. 作为主设备以轮询方式接收大量数据时使用hal_i2c_master_receive()。

3. 作为从设备以轮询方式发送大量数据时使用hal_i2c_slave_transmit()。
4. 作为从设备以轮询方式接收大量数据时使用hal_i2c_slave_receive()。

2.16.2.2 轮询方式的IO内存读写操作

1. 以轮询方式向指定地址写入大量数据时使用hal_i2c_mem_write()。
2. 以轮询方式从指定地址读取大量数据时使用hal_i2c_mem_read()。

2.16.2.3 中断方式的IO读写操作

1. 作为主设备以中断非轮询方式发送大量数据时使用hal_i2c_master_transmit_it()，发送完成时回调函数hal_i2c_master_tx_cplt_callback()将会被调用。
2. 作为主设备以中断非轮询方式接收大量数据时使用hal_i2c_master_receive_it()，接收完成时回调函数hal_i2c_master_rx_cplt_callback()将会被调用。
3. 作为从设备以中断非轮询方式发送大量数据时使用hal_i2c_slave_transmit_it()，发送完成时回调函数hal_i2c_slave_tx_cplt_callback()将会被调用。
4. 作为从设备以中断非轮询方式接收大量数据时使用hal_i2c_slave_receive_it()，接收完成时回调函数hal_i2c_slave_rx_cplt_callback()将会被调用。
5. 如果数据收发过程中发生了错误，则hal_i2c_error_callback()回调函数将会被调用。
6. 作为主设备时，如果需要中止数据收发，则可以使用hal_i2c_master_abort_it()，中止完成后回调函数hal_i2c_abort_cplt_callback()将会被调用。

说明:

对于上述回调函数，开发者均可通过重写函数完成指定的操作。

2.16.2.4 中断方式的IO内存读写操作

1. 以中断非轮询方式向指定地址写入大量数据时使用hal_i2c_mem_write_it()，写入完成时回调函数hal_i2c_mem_tx_cplt_callback()将会被调用。
2. 以中断非轮询方式从指定地址读取大量数据时使用hal_i2c_mem_read_it()，读取完成时回调函数hal_i2c_mem_rx_cplt_callback()将会被调用。
3. 如果数据收发过程中发生了错误，则hal_i2c_error_callback()回调函数将会被调用。

说明:

对于上述回调函数，开发者均可通过重写函数完成指定的操作。

2.16.2.5 DMA方式的IO读写操作

1. 作为主设备以DMA非轮询方式发送大量数据时使用hal_i2c_master_transmit_dma()，发送完成时回调函数hal_i2c_master_tx_cplt_callback()将会被调用。
2. 作为主设备以DMA非轮询方式接收大量数据时使用hal_i2c_master_receive_dma()，接收完成时回调函数hal_i2c_master_rx_cplt_callback()将会被调用。
3. 作为从设备以DMA非轮询方式发送大量数据时使用hal_i2c_slave_transmit_dma()，发送完成时回调函数hal_i2c_slave_tx_cplt_callback()将会被调用。
4. 作为从设备以DMA非轮询方式接收大量数据时使用hal_i2c_slave_receive_dma()，接收完成时回调函数hal_i2c_slave_rx_cplt_callback()将会被调用。
5. 如果数据收发过程中发生了错误，则hal_i2c_error_callback()回调函数将会被调用。
6. 作为主设备时，如果需要中止数据收发，则可以使用hal_i2c_master_abort_it()，中止完成后回调函数hal_i2c_abort_cplt_callback()将会被调用。

说明:

对于上述回调函数，开发者均可通过重写函数完成指定的操作。

2.16.2.6 DMA方式的IO内存读写操作

1. 以DMA非轮询方式向指定地址写入大量数据时使用hal_i2c_mem_write_dma()，写入完成时回调函数hal_i2c_mem_tx_cplt_callback()将会被调用。
2. 以DMA非轮询方式从指定地址读取大量数据时使用hal_i2c_mem_read_dma()，读取完成时回调函数hal_i2c_mem_rx_cplt_callback()将会被调用。
3. 如果数据收发过程中发生了错误，则hal_i2c_error_callback()回调函数将会被调用。

说明:

对于上述回调函数，开发者均可通过重写函数完成指定的操作。

2.16.3 I2C驱动的结构体

2.16.3.1 i2c_init_t

I2C驱动的初始化结构体i2c_init_t的定义如下：

表 2-234 i2c_init_t结构体

数据域	域段描述	取值
uint32_t speed	数据传输速度。	该参数的取值可以是下列值中的任意一个： • I2C_SPEED_100K（100 Kb/s）

数据域	域段描述	取值
		• I2C_SPEED_400K（400 Kb/s） • I2C_SPEED_1000K（1000 Kb/s） • I2C_SPEED_2000K（2.0 Mb/s）
uint32_t own_address	本机设备地址。	7位地址：0x08 ～ 0x77 10位地址：0x008 ～ 0x077, 0x080 ～ 0x3FE
uint32_t addressing_mode	本机设备地址及对端设备地址的格式。	该参数的取值可以是下列值中的任意一个： • I2C_ADDRESSINGMODE_7BIT（7位地址） • I2C_ADDRESSINGMODE_10BIT（10位地址）
uint32_t general_call_mode	是否启用广播地址监测功能。	该参数的取值可以是下列值中的任意一个： • I2C_GENERALCALL_DISABLE（禁用） • I2C_GENERALCALL_ENABLE（启用）

2.16.3.2 i2c_handle_t

I2C驱动的句柄结构体i2c_handle_t的定义如下：

表 2-235 i2c_handle_t结构体

数据域	域段描述	取值
i2c_regs_t *p_instance	I2C外设实例。	该参数的取值可以是下列值中的任意一个： • I2C0 • I2C1
i2c_init_t init	初始化结构体。	参考i2c_init_t结构体。
uint8_t *p_buffer	指向数据传输缓冲区的指针（驱动负责管理，无需开发者初始化）。	N/A
uint16_t xfer_size	数据传输长度（驱动负责管理，无需开发者初始化）。	N/A
__IO uint16_t xfer_count	数据传输计数（驱动负责管理，无需开发者初始化）。	N/A
__IO uint16_t master_ack_count	作为主设备时接收数据的ACK计数（驱动负责管理，无需开发者初始化）。	N/A
__IO uint32_t xfer_options	顺序传输选项（驱动负责管理，无需开发者初始化）。	N/A
__IO uint32_t previous_state	上一个通信状态（驱动负责管理，无需开发者初始化）。	N/A
hal_status_t(*xfer_isr)(struct _i2c_handle *p_i2c, uint32_t it_source, uint32_t abort_sources)	数据传输的中断处理函数（驱动负责管理，无需开发者初始化）。	N/A

数据域	域段描述	取值
dma_handle_t *p_dmatx	指向I2C TX通道的DMA句柄的指针。	发送通道的DMA句柄dma_handle_t结构体。
dma_handle_t *p_dmarx	指向I2C RX通道的 DMA句柄的指针。	接收通道的DMA句柄dma_handle_t结构体。
__IO hal_lock_t lock	I2C锁（驱动负责管理，无需开发者初始化）。	N/A
__IO hal_i2c_state_t state	I2C运行状态（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_I2C_STATE_RESET（未初始化） • HAL_I2C_STATE_READY（已初始化且空闲） • HAL_I2C_STATE_BUSY（忙） • HAL_I2C_STATE_BUSY_TX（正在发送） • HAL_I2C_STATE_BUSY_RX （正在接收） • HAL_I2C_STATE_ABORT（中止） • HAL_I2C_STATE_TIMEOUT（超时） • HAL_I2C_STATE_ERROR（错误）
__IO hal_i2c_mode_t mode	I2C工作模式（驱动负责管理，无需开发者初始化）。	N/A
__IO uint32_t error_code	I2C错误码（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_I2C_ERROR_NONE（无错误） • HAL_I2C_ERROR_ARB_LOST（仲裁丢失） • HAL_I2C_ERROR_NOACK（无ACK） • HAL_I2C_ERROR_OVER（接收溢出） • HAL_I2C_ERROR_DMA（DMA传输错误） • HAL_I2C_ERROR_TIMEOUT（超时）
uint32_t retention[10]	保存I2C寄存器信息（驱动负责管理，无需开发者初始化）。	N/A。

2.16.4 I2C驱动API描述

I2C驱动的API主要包括：

表 2-236 I2C驱动的APIs

API类别	API名称	描述
初始化	hal_i2c_init()	初始化I2C外设，配置传输速度等参数。
	hal_i2c_deinit()	反初始化I2C外设。
	hal_i2c_msp_init()	初始化I2C外设所使用的GPIO引脚复用、NVIC中断、DMA通道。
	hal_i2c_msp_deinit()	反初始化I2C外设所使用的GPIO引脚复用、NVIC中断、DMA通道。
IO操作	hal_i2c_master_transmit()	作为主设备向从设备发送数据，轮询方式。
	hal_i2c_master_receive()	作为主设备从从设备接收数据，轮询方式。

API类别	API名称	描述
	hal_i2c_slave_transmit()	作为从设备向主设备发送数据，轮询方式。
	hal_i2c_slave_receive()	作为从设备从主设备接收数据，轮询方式。
	hal_i2c_mem_write()	向指定的地址写入数据，轮询方式。
	hal_i2c_mem_read()	从指定的地址读取数据，轮询方式。
	hal_i2c_master_transmit_it()	作为主设备向从设备发送数据，中断方式。
	hal_i2c_master_receive_it()	作为主设备从从设备接收数据，中断方式。
	hal_i2c_slave_transmit_it()	作为从设备向主设备发送数据，中断方式。
	hal_i2c_slave_receive_it()	作为从设备从主设备接收数据，中断方式。
	hal_i2c_mem_write_it()	向指定的地址写入数据，中断方式。
	hal_i2c_mem_read_it()	从指定的地址读取数据，中断方式。
	hal_i2c_master_sequential_transmit_it()	作为主设备发送帧数据，中断方式。
	hal_i2c_master_sequential_receive_it()	作为主设备接收帧数据，中断方式。
	hal_i2c_slave_sequential_transmit_it()	作为从设备发送帧数据，中断方式。
	hal_i2c_slave_sequential_receive_it()	作为从设备接收帧数据，中断方式。
	hal_i2c_enable_listen_it()	作为主设备监听信号，中断方式。
	hal_i2c_disable_listen_it()	禁用作为主设备监听信号，中断方式。
	hal_i2c_master_transmit_dma()	作为主设备向从设备发送数据，DMA方式。
	hal_i2c_master_receive_dma()	作为主设备从从设备接收数据，DMA方式。
	hal_i2c_slave_transmit_dma()	作为从设备向主设备发送数据，DMA方式。
	hal_i2c_slave_receive_dma()	作为从设备从主设备接收数据，DMA方式。
	hal_i2c_mem_write_dma()	向指定的地址写入数据，DMA方式。
	hal_i2c_mem_read_dma()	从指定的地址读取数据，DMA方式。
	hal_i2c_master_abort_it()	中止中断/DMA方式下的数据传输。
中断处理及回调函数	hal_i2c_irq_handler()	中断处理函数。
	hal_i2c_master_tx_cplt_callback()	主设备发送完成中断回调函数。
	hal_i2c_master_rx_cplt_callback()	主设备接收完成中断回调函数。
	hal_i2c_slave_tx_cplt_callback()	从设备发送完成中断回调函数。
	hal_i2c_slave_rx_cplt_callback()	从设备接收完成中断回调函数。
	hal_i2c_mem_tx_cplt_callback()	写入完成中断回调函数。
	hal_i2c_mem_rx_cplt_callback()	读取完成中断回调函数。
	hal_i2c_listen_cplt_callback()	监听中断回调函数。
	hal_i2c_error_callback()	错误中断回调函数。
	hal_i2c_abort_cplt_callback()	中止完成中断回调函数。
状态及错误	hal_i2c_get_state()	获取驱动运行状态。

API类别	API名称	描述
	hal_i2c_get_mode()	获取当前工作模式。
	hal_i2c_get_error()	获取错误码。
睡眠相关	hal_i2c_suspend_reg()	睡眠时挂起I2C配置相关的寄存器。
	hal_i2c_resume_reg()	唤醒时恢复I2C配置相关的寄存器。

下面章节将对各API进行详细描述。

2.16.4.1 hal_i2c_init

表 2-237 hal_i2c_init接口

函数原型	hal_status_t hal_i2c_init(i2c_handle_t *p_i2c)
功能说明	根据i2c_init_t中的指定参数初始化I2C外设和初始化关联句柄。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	HAL状态。
备注	

2.16.4.2 hal_i2c_deinit

表 2-238 hal_i2c_deinit接口

函数原型	hal_status_t hal_i2c_deinit(i2c_handle_t *p_i2c)
功能说明	反初始化I2C外设。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	HAL状态。
备注	

2.16.4.3 hal_i2c_msp_init

表 2-239 hal_i2c_msp_init接口

函数原型	void hal_i2c_msp_init(i2c_handle_t *p_i2c)
功能说明	初始化I2C所使用的GPIO引脚复用、NVIC中断、DMA通道等配置。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成GPIO引脚复用、NVIC中断、DMA通道的初始化。

2.16.4.4 hal_i2c_msp_deinit

表 2-240 hal_i2c_msp_deinit接口

函数原型	void hal_i2c_msp_deinit(i2c_handle_t *p_i2c)
功能说明	反初始化I2C所使用的GPIO引脚复用、NVIC中断、DMA通道等配置。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成GPIO引脚复用、NVIC中断、DMA通道的反初始化。

2.16.4.5 hal_i2c_master_transmit

表 2-241 hal_i2c_master_transmit接口

函数原型	hal_status_t hal_i2c_master_transmit(i2c_handle_t *p_i2c, uint16_t dev_address, uint8_t *p_data, uint16_t size, uint32_t timeout)
功能说明	作为I2C主设备，发送大量数据，轮询方式。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 p_data：指向数据缓冲区的指针。 size：待发送数据的长度。 timeout：超时时间。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_i2c_get_error()获取具体的错误码。

2.16.4.6 hal_i2c_master_receive

表 2-242 hal_i2c_master_receive接口

函数原型	hal_status_t hal_i2c_master_receive(i2c_handle_t *p_i2c, uint16_t dev_address, uint8_t *p_data, uint16_t size, uint32_t timeout)
功能说明	作为I2C主设备，接收大量数据，轮询方式。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 p_data：指向数据缓冲区的指针。 size：待接收数据的长度。 timeout：超时时间。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_i2c_get_error()获取具体的错误码。

2.16.4.7 hal_i2c_slave_transmit

表 2-243 hal_i2c_slave_transmit接口

函数原型	hal_status_t hal_i2c_slave_transmit(i2c_handle_t *p_i2c, uint8_t *p_data, uint16_t size, uint32_t timeout)
功能说明	作为I2C从设备，发送大量数据，轮询方式。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 p_data：指向数据缓冲区的指针。 size：待发送数据的长度。 timeout：超时时间。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_i2c_get_error()获取具体的错误码。

2.16.4.8 hal_i2c_slave_receive

表 2-244 hal_i2c_slave_receive接口

函数原型	hal_status_t hal_i2c_slave_receive(i2c_handle_t *p_i2c, uint8_t *p_data, uint16_t size, uint32_t timeout)
功能说明	作为I2C从设备，接收大量数据，轮询方式。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 p_data：指向数据缓冲区的指针。 size：待接收数据的长度。 timeout：超时时间。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_i2c_get_error()获取具体的错误码。

2.16.4.9 hal_i2c_mem_write

表 2-245 hal_i2c_mem_write接口

函数原型	hal_status_t hal_i2c_mem_write(i2c_handle_t *p_i2c, uint16_t dev_address, uint16_t mem_address, uint16_t mem_addr_size, uint8_t *p_data, uint16_t size, uint32_t timeout)
功能说明	以轮询模式向从设备的指定地址写入大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 mem_address：从设备的内部指定地址。 mem_addr_size：从设备的内部指定地址的位宽，该参数的取值可以是下列值中的任意一个： • I2C_MEMADD_SIZE_8BIT（8位） • I2C_MEMADD_SIZE_16BIT（16位）

	p_data：指向数据缓冲区的指针。 size：待写入数据的长度。 timeout：超时时间
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_i2c_get_error()获取具体的错误码。

2.16.4.10 hal_i2c_mem_read

表 2-246 hal_i2c_mem_read接口

函数原型	hal_status_t hal_i2c_mem_read(i2c_handle_t *p_i2c, uint16_t dev_address, uint16_t mem_address, uint16_t mem_addr_size, uint8_t *p_data, uint16_t size, uint32_t timeout)
功能说明	以轮询模式从从设备的指定地址读取大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 mem_address：从设备的内部指定地址。 mem_addr_size：从设备的内部指定地址的位宽，该参数的取值可以是下列值中的任意一个： • I2C_MEMADD_SIZE_8BIT（8位） • I2C_MEMADD_SIZE_16BIT（16位） p_data：指向数据缓冲区的指针。 size：待读取数据的长度。 timeout：超时时间。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_i2c_get_error()获取具体的错误码。

2.16.4.11 hal_i2c_master_transmit_it

表 2-247 hal_i2c_master_transmit_it接口

函数原型	hal_status_t hal_i2c_master_transmit_it(i2c_handle_t *p_i2c, uint16_t dev_address, uint8_t *p_data, uint16_t size)
功能说明	作为I2C主设备，以中断非轮询模式发送大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 p_data：指向数据缓冲区的指针。 size：待发送数据的长度。
返回值	HAL状态。
备注	• 发送完成时，回调函数hal_i2c_master_tx_cplt_callback()会被调用。 • 发送过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_master_tx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

	• 传输过程中，I2C中断处理无法及时响应，可能出现发送失败的情况。

2.16.4.12 hal_i2c_master_receive_it

表 2-248 hal_i2c_master_receive_it接口

函数原型	hal_status_t hal_i2c_master_receive_it(i2c_handle_t *p_i2c, uint16_t dev_address, uint8_t *p_data, uint16_t size)
功能说明	作为I2C主设备，以中断非轮询模式接收大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 p_data：指向数据缓冲区的指针。 size：待接收数据的长度。
返回值	HAL状态。
备注	• 接收完成时，回调函数hal_i2c_master_rx_cplt_callback()会被调用。 • 接收过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_master_rx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。 • 传输过程中，I2C中断处理无法及时响应，可能出现接收失败的情况。

2.16.4.13 hal_i2c_slave_transmit_it

表 2-249 hal_i2c_slave_transmit_it接口

函数原型	hal_status_t hal_i2c_slave_transmit_it(i2c_handle_t *p_i2c, uint8_t *p_data, uint16_t size)
功能说明	作为I2C从设备，以中断非轮询模式发送大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 p_data：指向数据缓冲区的指针。 size：待发送数据的长度。
返回值	HAL状态
备注	• 发送完成时，回调函数hal_i2c_slave_tx_cplt_callback()会被调用。 • 发送过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_slave_tx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。 • 传输过程中，I2C中断处理无法及时响应，可能出现发送数据错误的情况。

2.16.4.14 hal_i2c_slave_receive_it

表 2-250 hal_i2c_slave_receive_it接口

函数原型	hal_status_t hal_i2c_slave_receive_it(i2c_handle_t *p_i2c, uint8_t *p_data, uint16_t size)
功能说明	作为I2C从设备，以中断非轮询模式接收大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。

	p_data：指向数据缓冲区的指针。 size：待接收数据的长度。
返回值	HAL状态。
备注	• 接收完成时，回调函数hal_i2c_slave_rx_cplt_callback()会被调用。 • 接收过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_slave_rx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。 • 传输过程中，I2C中断处理无法及时响应，可能出现接收数据错误的情况。

2.16.4.15 hal_i2c_mem_write_it

表 2-251 hal_i2c_mem_write_it接口

函数原型	hal_status_t hal_i2c_mem_write_it(i2c_handle_t *p_i2c, uint16_t dev_address, uint16_t mem_address, uint16_t mem_addr_size, uint8_t *p_data, uint16_t size)
功能说明	以中断非轮询模式向从设备的指定地址写入大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 mem_address：从设备的内部指定地址。 mem_addr_size：从设备的内部指定地址的位宽，该参数的取值可以是下列值中的任意一个： • I2C_MEMADD_SIZE_8BIT（8位） • I2C_MEMADD_SIZE_16BIT（16位） p_data：指向数据缓冲区的指针。 size：待写入数据的长度。
返回值	HAL状态。
备注	• 写入完成时，回调函数hal_i2c_mem_tx_cplt_callback()会被调用。 • 写入过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_mem_tx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。 • 传输过程中，I2C中断处理无法及时响应，可能出现发送失败的情况。

2.16.4.16 hal_i2c_mem_read_it

表 2-252 hal_i2c_mem_read_it接口

函数原型	hal_status_t hal_i2c_mem_read_it(i2c_handle_t *p_i2c, uint16_t dev_address, uint16_t mem_address, uint16_t mem_addr_size, uint8_t *p_data, uint16_t size)
功能说明	以中断非轮询模式从从设备的指定地址读取大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 mem_address：从设备的内部指定地址。 mem_addr_size：从设备的内部指定地址的位宽，该参数的取值可以是下列值中的任意一个：

	• I2C_MEMADD_SIZE_8BIT（8位） • I2C_MEMADD_SIZE_16BIT（16位） p_data：指向数据缓冲区的指针。 size：待读取数据的长度。
返回值	HAL状态。
备注	• 读取完成时，回调函数hal_i2c_mem_rx_cplt_callback()会被调用。 • 读取过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_mem_rx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。 • 传输过程中，I2C中断处理无法及时响应，可能出现接收失败的情况。

2.16.4.17 hal_i2c_master_abort_it

表 2-253 hal_i2c_master_abort_it接口

函数原型	hal_status_t hal_i2c_master_abort_it(i2c_handle_t *p_i2c)
功能说明	通过中断方式中止I2C主设备中断或DMA方式的数据传输。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	HAL状态。
备注	该函数为非轮询式函数，使能TX_ABRT中断后，该函数立即返回，TX_ABRT中断触发后中止完成，此时hal_i2c_abort_cplt_callback()将会被调用。

2.16.4.18 hal_i2c_master_transmit_dma

表 2-254 hal_i2c_master_transmit_dma接口

函数原型	hal_status_t hal_i2c_master_transmit_dma(i2c_handle_t *p_i2c, uint16_t dev_address, uint8_t *p_data, uint16_t size)
功能说明	作为I2C主设备，发送大量数据，DMA方式。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 p_data：指向数据缓冲区的指针。 size：待发送数据的长度。
返回值	HAL状态。
备注	• 发送完成时，回调函数hal_i2c_master_tx_cplt_callback()会被调用。 • 发送过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_master_tx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。 • 传输过程中，I2C中断处理无法及时响应，可能出现传输失败的情况。

2.16.4.19 hal_i2c_master_receive_dma

表 2-255 hal_i2c_master_receive_dma接口

函数原型	hal_status_t hal_i2c_master_receive_dma(i2c_handle_t *p_i2c, uint16_t dev_address, uint8_t *p_data, uint16_t size)
功能说明	作为I2C主设备，以DMA非轮询模式接收大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 p_data：指向数据缓冲区的指针。 size：待接收数据的长度。
返回值	HAL状态。
备注	• 接收完成时，回调函数hal_i2c_master_rx_cplt_callback()会被调用。 • 接收过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_master_rx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.16.4.20 hal_i2c_slave_transmit_dma

表 2-256 hal_i2c_slave_transmit_dma接口

函数原型	hal_status_t hal_i2c_slave_transmit_dma(i2c_handle_t *p_i2c, uint8_t *p_data, uint16_t size)
功能说明	作为I2C从设备，以DMA非轮询模式发送大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 p_data：指向数据缓冲区的指针。 size：待发送数据的长度。
返回值	HAL状态。
备注	• 发送完成时，回调函数hal_i2c_slave_tx_cplt_callback()会被调用。 • 发送过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_slave_tx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.16.4.21 hal_i2c_slave_receive_dma

表 2-257 hal_i2c_slave_receive_dma接口

函数原型	hal_status_t hal_i2c_slave_receive_dma(i2c_handle_t *p_i2c, uint8_t *p_data, uint16_t size)
功能说明	作为I2C从设备，以DMA非轮询模式接收大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 p_data：指向数据缓冲区的指针。 size：待接收数据的长度。
返回值	HAL状态。

备注	• 接收完成时，回调函数hal_i2c_slave_rx_cplt_callback()会被调用。 • 接收过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_slave_rx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.16.4.22 hal_i2c_mem_write_dma

表 2-258 hal_i2c_mem_write_dma接口

函数原型	hal_status_t hal_i2c_mem_write_dma(i2c_handle_t *p_i2c, uint16_t dev_address, uint16_t mem_address, uint16_t mem_addr_size, uint8_t *p_data, uint16_t size)
功能说明	以DMA非轮询模式向指定从设备的指定地址写入大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 mem_address：从设备的内部指定地址。 mem_addr_size：从设备的内部指定地址的位宽，该参数的取值可以是下列值中的任意一个： • I2C_MEMADD_SIZE_8BIT（8位） • I2C_MEMADD_SIZE_16BIT（16位） p_data：指向数据缓冲区的指针。 size：待写入数据的长度。
返回值	HAL状态。
备注	• 写入完成时，回调函数hal_i2c_mem_tx_cplt_callback()会被调用。 • 写入过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_mem_tx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.16.4.23 hal_i2c_mem_read_dma

表 2-259 hal_i2c_mem_read_dma接口

函数原型	hal_status_t hal_i2c_mem_read_dma(i2c_handle_t *p_i2c, uint16_t dev_address, uint16_t mem_address, uint16_t mem_addr_size, uint8_t *p_data, uint16_t size)
功能说明	以DMA非轮询模式从指定从设备的指定地址读取大量数据。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。 dev_address：从设备地址。 mem_address：从设备的内部指定地址。 mem_addr_size：从设备的内部指定地址的位宽，该参数的取值可以是下列值中的任意一个： • I2C_MEMADD_SIZE_8BIT（8位） • I2C_MEMADD_SIZE_16BIT（16位） p_data：指向数据缓冲区的指针。 size：待读取数据的长度。
返回值	HAL状态

备注	• 读取完成时，回调函数hal_i2c_mem_rx_cplt_callback()会被调用。 • 读取过程中出现错误时，回调函数hal_i2c_error_callback()会被调用，开发者可在回调函数中调用hal_i2c_get_error()获取具体的错误码。 • 回调函数hal_i2c_mem_rx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.16.4.24 hal_i2c_irq_handler

表 2-260 hal_i2c_irq_handler接口

函数原型	void hal_i2c_irq_handler(i2c_handle_t *p_i2c)
功能说明	处理I2C中断请求。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无
备注	

2.16.4.25 hal_i2c_master_tx_cplt_callback

表 2-261 hal_i2c_master_tx_cplt_callback接口

函数原型	void hal_i2c_master_tx_cplt_callback(i2c_handle_t *p_i2c)
功能说明	主设备发送完成回调函数。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.16.4.26 hal_i2c_master_rx_cplt_callback

表 2-262 hal_i2c_master_rx_cplt_callback接口

函数原型	void hal_i2c_master_rx_cplt_callback(i2c_handle_t *p_i2c)
功能说明	主设备接收完成回调函数。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.16.4.27 hal_i2c_slave_tx_cplt_callback

表 2-263 hal_i2c_slave_tx_cplt_callback接口

函数原型	void hal_i2c_slave_tx_cplt_callback(i2c_handle_t *p_i2c)
功能说明	从设备发送完成回调函数。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无

备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.16.4.28 hal_i2c_slave_rx_cplt_callback

表 2-264 hal_i2c_slave_rx_cplt_callback接口

函数原型	void hal_i2c_slave_rx_cplt_callback(i2c_handle_t *p_i2c)
功能说明	从设备接收完成回调函数。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.16.4.29 hal_i2c_mem_tx_cplt_callback

表 2-265 hal_i2c_mem_tx_cplt_callback接口

函数原型	void hal_i2c_mem_tx_cplt_callback(i2c_handle_t *p_i2c)
功能说明	从设备写入完成回调函数。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.16.4.30 hal_i2c_mem_rx_cplt_callback

表 2-266 hal_i2c_mem_rx_cplt_callback接口

函数原型	void hal_i2c_mem_rx_cplt_callback(i2c_handle_t *p_i2c)
功能说明	从设备读取完成回调函数。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.16.4.31 hal_i2c_error_callback

表 2-267 hal_i2c_error_callback接口

函数原型	void hal_i2c_error_callback(i2c_handle_t *p_i2c)
功能说明	I2C错误回调函数。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.16.4.32 hal_i2c_abort_cplt_callback

表 2-268 hal_i2c_abort_cplt_callback接口

函数原型	void hal_i2c_abort_cplt_callback(i2c_handle_t *p_i2c)
功能说明	I2C中止完成回调函数。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.16.4.33 hal_i2c_get_state

表 2-269 hal_i2c_get_state接口

函数原型	hal_i2c_state_t hal_i2c_get_state(i2c_handle_t *p_i2c)
功能说明	获取I2C运行状态。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	I2C运行状态，该参数的取值可以是下列值中的任意一个： • HAL_I2C_STATE_RESET（未初始化） • HAL_I2C_STATE_READY（已初始化且空闲） • HAL_I2C_STATE_BUSY（忙） • HAL_I2C_STATE_BUSY_TX（正在发送） • HAL_I2C_STATE_BUSY_RX （正在接收） • HAL_I2C_STATE_LISTEN（正在地址监听） • HAL_I2C_STATE_BUSY_TX_LISTEN（正在发送和地址监听） • HAL_I2C_STATE_BUSY_RX_LISTEN（正在接收和地址监听） • HAL_I2C_STATE_ABORT（被中断） • HAL_I2C_STATE_TIMEOUT（超时） • HAL_I2C_STATE_ERROR（错误）
备注	

2.16.4.34 hal_i2c_get_mode

表 2-270 hal_i2c_get_mode接口

函数原型	hal_i2c_mode_t hal_i2c_get_mode(i2c_handle_t *p_i2c)
功能说明	返回I2C模式：主设备、从设备、Memory或者无
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	I2C模式，该参数的取值可以是下列值中的任意一个： • HAL_I2C_MODE_NONE（无） • HAL_I2C_MODE_MASTER（作为主设备） • HAL_I2C_MODE_SLAVE（作为从设备） • HAL_I2C_MODE_MEM（读写存储设备）

备注	

2.16.4.35 hal_i2c_get_error

表 2-271 hal_i2c_get_error接口

函数原型	uint32_t hal_i2c_get_error(i2c_handle_t *p_i2c)
功能说明	返回I2C句柄错误码。
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	I2C错误码，该参数的取值可以是下列值中的任意一个： • HAL_I2C_ERROR_NONE（无错误） • HAL_I2C_ERROR_ARB_LOST（仲裁丢失） • HAL_I2C_ERROR_NOACK（无ACK） • HAL_I2C_ERROR_OVER（接收溢出） • HAL_I2C_ERROR_DMA（DMA传输错误） • HAL_I2C_ERROR_TIMEOUT（超时）
备注	

2.16.4.36 hal_i2c_suspend_reg

表 2-272 hal_i2c_suspend_reg接口

函数原型	hal_status_t hal_i2c_suspend_reg(i2c_handle_t *p_i2c)
功能说明	睡眠时挂起I2C配置相关的寄存器
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	HAL状态
备注	

2.16.4.37 hal_i2c_resume_reg

表 2-273 hal_i2c_resume_reg接口

函数原型	hal_status_t hal_i2c_resume_reg(i2c_handle_t *p_i2c)
功能说明	唤醒时恢复I2C配置相关的寄存器
输入参数	p_i2c：指向i2c_handle_t结构体变量的指针，该结构体变量包含指定的I2C的配置信息。
返回值	HAL状态
备注	

2.17 HAL QSPI通用驱动

2.17.1 QSPI驱动功能

QSPI（Quad-SPI）外设的HAL驱动主要实现了以下功能：

- 支持Standard、Dual、Quad三种传输模式。
- 支持最大32位的数据传输位宽。
- 支持最高32 MHz的传输速率（Standard模式）。
- 支持时钟极性（CPOL）及时钟相位（CPHA）配置。
- 支持指令和地址的位宽及传输模式配置。
- 支持发送FIFO、接收FIFO的阈值设置及获取。
- 支持轮询、中断、DMA三种数据读写方式。
- 支持中止在中断及DMA方式下的数据读写。
- 支持发送完成、接收完成、错误、中止完成的中断回调函数。
- 支持获取驱动运行状态及错误码。
- 支持超时时间设置。

2.17.2 如何使用QSPI驱动

QSPI HAL驱动使用方法如下：

1. 声明一个qspi_handle_t句柄结构体，例如：qspi_handle_t qspi_handle。
2. 重写hal_qspi_msp_init() API以初始化QSPI底层资源：
   (1) 配置QSPI引脚：调用hal_gpio_init()配置GPIO模式为GPIO_MODE_MUX（复用模式），并将相应的GPIO的复用功能配置为QSPI。

   (2) 如果开发者要使用中断流程(hal_qspi_transmit_it()和hal_qspi_receive_it())，则需通过调用相关的NVIC接口来配置：
   - 调用hal_nvic_set_priority()配置QSPI中断优先级。
   - 调用hal_nvic_enable_irq()使能QSPI中断处理。

   (3) 如果开发者要使用DMA流程(hal_qspi_transmit_dma()和hal_qspi_receive_dma())，则需配置DMA：
   - TX/RX通道只需要声明一条DMA通道。
   - 为TX/RX通道声明DMA句柄结构体，例如：dma_handle_t hdma。
   - 使用所需的TX/RX参数配置声明的DMA句柄结构体。
   - 配置DMA TX/RX信道。
   - 将初始的DMA句柄关联到QSPI DMA TX/RX句柄。
   - 在DMA TX/RX信道上配置优先级和使能传输完成中断。

   (4) 配置qspi_handle句柄的init结构体中的时钟分频等参数。

(5) 调用hal_qspi_init() API初始化QSPI寄存器。

2.17.3 QSPI驱动结构的结构体

2.17.3.1 qspi_init_t

QSPI驱动的初始化结构体qspi_init_t的定义如下：

表 2-274 qspi_init_t结构体

数据域	域段描述	取值
uint32_t clock_prescaler	时钟分频系数。	0x0000 ～ 0xFFFF之间的偶数 QSPI传输速率 = 系统时钟 / 分频系数
uint32_t clock_mode	时钟极性及相位模式。	该参数的取值可以是下列值中的任意一个： • QSPI_CLOCK_MODE_0（CPOL = 0，CPHA = 0） • QSPI_CLOCK_MODE_1（CPOL = 0，CPHA = 1） • QSPI_CLOCK_MODE_2（CPOL = 1，CPHA = 0） • QSPI_CLOCK_MODE_3（CPOL =1，CPHA = 1）
uint32_t rx_sample_delay	RX延时采集参数	0x0 ～ 0x7

2.17.3.2 qspi_handle_t

QSPI驱动的句柄结构体qspi_handle_t的定义如下：

表 2-275 qspi_handle_t结构体

数据域	域段描述	取值
ssi_regs_t*p_instance	QSPI外设实例。	该参数的取值可以是下列值中的任意一个： • QSPI0 • QSPI1
qspi_init_t init	初始化结构体(参考qspi_init_t结构体)。	N/A
uint8_t *p_tx_buffer	指向数据发送缓冲区的指针（驱动负责管理，无需开发者初始化）。	N/A
__IO uint32_t tx_buffer_size	数据发送长度（驱动负责管理，无需开发者初始化）。	N/A
__IO uint32_t tx_xfer_count	数据发送计数（驱动负责管理，无需开发者初始化）。	N/A
uint8_t *p_rx_buffer	指向数据接收缓冲区的指针（驱动负责管理，无需开发者初始化）。	N/A

数据域	域段描述	取值
__IO uint32_t rx_buffer_size	数据接收长度（驱动负责管理，无需开发者初始化）。	N/A
__IO uint32_t rx_xfer_count	数据接收计数（驱动负责管理，无需开发者初始化）。	N/A
void (*write_fifo)(struct _qspi_handle *p_qspi)	指向QSPI TX写FIFO函数的指针（驱动负责管理，无需开发者初始化）。	N/A
void (*read_fifo)(struct _qspi_handle *p_qspi)	指向QSPI RX读FIFO函数的指针（驱动负责管理，无需开发者初始化）。	N/A
dma_handle_t *p_dma	指向数据收发通道的DMA句柄dma_handle_t结构体的指针。	N/A
__IO hal_lock_t lock	QSPI锁（驱动负责管理，无需开发者初始化）。	N/A
__IO hal_qspi_state_t state	QSPI运行状态（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_QSPI_STATE_RESET（未初始化） • HAL_QSPI_STATE_READY（已初始化且空闲） • HAL_QSPI_STATE_BUSY（忙） • HAL_QSPI_STATE_BUSY_INDIRECT_TX（正在发送） • HAL_QSPI_STATE_BUSY_INDIRECT_RX（正在接收） • HAL_QSPI_STATE_ABORT（被中断） • HAL_QSPI_STATE_ERROR（错误）
__IO uint32_t error_code	QSPI错误码（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_QSPI_ERROR_NONE（无错误） • HAL_QSPI_ERROR_TIMEOUT（超时） • HAL_QSPI_ERROR_TRANSFER（传输错误） • HAL_QSPI_ERROR_DMA（DMA传输错误） • HAL_QSPI_ERROR_INVALID_PARAM（非法参数）
uint32_t timeout	QSPI超时时间（无需开发者初始化）。	N/A

数据域	域段描述	取值
uint32_t retention[8]	保存QSPI寄存器信息（驱动负责管理，无需开发者初始化）。	N/A

2.17.3.3 qspi_command_t

QSPI驱动的命令结构体qspi_command_t的定义如下：

表 2-276 qspi_command_t结构体

数据域	域段描述	取值
uint32_t instruction	指令	0x0000 ～ 0xFFFF
uint32_t address	地址	0x0000_0000 ～ 0xFFFF_FFFF
uint32_t instruction_size	指令位宽	该参数的取值可以是下列值中的任意一个： • QSPI_INSTSIZE_00_BITS（0位） • QSPI_INSTSIZE_04_BITS（4位） • QSPI_INSTSIZE_08_BITS（8位） • QSPI_INSTSIZE_16_BITS（16位）
uint32_t address_size	地址位宽	该参数的取值可以是下列值中的任意一个： • QSPI_ADDRSIZE_00_BITS（0位） • QSPI_ADDRSIZE_04_BITS（4位） • QSPI_ADDRSIZE_08_BITS（8位） • QSPI_ADDRSIZE_12_BITS（12位） • QSPI_ADDRSIZE_16_BITS（16位） • QSPI_ADDRSIZE_20_BITS（20位） • QSPI_ADDRSIZE_24_BITS（24位） • QSPI_ADDRSIZE_28_BITS（28位） • QSPI_ADDRSIZE_32_BITS（32位）
uint32_t dummy_cycles	读写转换插入时钟周期	0 ～ 31
uint32_t data_size	有效数据位	该参数的取值可以是下列值中的任意一个： • LL_SSI_DATASIZE_4BIT • LL_SSI_DATASIZE_5BIT • LL_SSI_DATASIZE_6BIT • LL_SSI_DATASIZE_7BIT • LL_SSI_DATASIZE_8BIT • LL_SSI_DATASIZE_9BIT • LL_SSI_DATASIZE_10BIT • LL_SSI_DATASIZE_11BIT • LL_SSI_DATASIZE_12BIT • LL_SSI_DATASIZE_13BIT • LL_SSI_DATASIZE_14BIT • LL_SSI_DATASIZE_15BIT

数据域	域段描述	取值
		• LL_SSI_DATASIZE_16BIT • LL_SSI_DATASIZE_17BIT • LL_SSI_DATASIZE_18BIT • LL_SSI_DATASIZE_19BIT • LL_SSI_DATASIZE_20BIT • LL_SSI_DATASIZE_21BIT • LL_SSI_DATASIZE_22BIT • LL_SSI_DATASIZE_23BIT • LL_SSI_DATASIZE_24BIT • LL_SSI_DATASIZE_25BIT • LL_SSI_DATASIZE_26BIT • LL_SSI_DATASIZE_27BIT • LL_SSI_DATASIZE_28BIT • LL_SSI_DATASIZE_29BIT • LL_SSI_DATASIZE_30BIT • LL_SSI_DATASIZE_31BIT • LL_SSI_DATASIZE_32BIT
uint32_t instruction_address_mode	指令及地址传输模式	该参数的取值可以是下列值中的任意一个： • QSPI_INST_ADDR_ALL_IN_SPI（指令和地址采用Standard SPI模式传输） • QSPI_INST_IN_SPI_ADDR_IN_SPIFRF（指令采用Standard SPI模式传输，地址采用Dual/Quad SPI模式） • QSPI_INST_ADDR_ALL_IN_SPIFRF（指令和地址都采用Dual/Quad SPI模式传输）
uint32_t data_mode	数据传输模式	该参数的取值可以是下列值中的任意一个： • QSPI_DATA_MODE_SPI（Standard SPI模式） • QSPI_DATA_MODE_DUALSPI（Dual SPI模式） • QSPI_DATA_MODE_QUADSPI（Quad SPI模式）
uint32_t length	数据长度	0x0000_0000 ～ 0xFFFF_FFFF

2.17.4 QSPI驱动API描述

QSPI驱动的API主要包括：

表 2-277 QSPI驱动的APIs

API类别	API名称	描述
初始化	hal_qspi_init()	初始化QSPI外设，配置时钟分频系数等参数。
	hal_qspi_deinit()	反初始化QSPI外设。
	hal_qspi_msp_init()	初始化QSPI外设所使用的GPIO引脚复用、NVIC中断、DMA通道。

API类别	API名称	描述
	hal_qspi_msp_deinit()	反初始化QSPI外设所使用的GPIO引脚复用、NVIC中断、DMA通道。
IO操作	hal_qspi_command_transmit()	发送数据（包含：指令，地址，数据），轮询方式。
	hal_qspi_command_receive()	接收数据（包含：指令，地址，数据），轮询方式。
	hal_qspi_command()	发送指令，轮询方式。
	hal_qspi_transmit()	以SPI方式发送数据，轮询方式。
	hal_qspi_receive()	以SPI方式接收数据，轮询方式。
	hal_qspi_command_transmit_it()	发送数据（包含：指令，地址，数据），中断方式。
	hal_qspi_command_receive_it()	接收数据（包含：指令，地址，数据），中断方式。
	hal_qspi_command_it()	发送指令，中断方式。
	hal_qspi_transmit_it()	以SPI方式发送数据，中断方式。
	hal_qspi_receive_it()	以SPI方式接收数据，中断式。
	hal_qspi_command_transmit_dma()	发送数据（包含：指令，地址，数据），DMA方式。
	hal_qspi_command_receive_dma()	接收数据（包含：指令，地址，数据），DMA方式。
	hal_qspi_command_dma()	发送指令，DMA方式。
	hal_qspi_transmit_dma()	以SPI方式发送数据，DMA方式。
	hal_qspi_receive_dma()	以SPI方式接收数据，DMA方式。
	hal_qspi_abort()	中止中断及DMA方式下的数据传输，轮询方式。
	hal_qspi_abort_it()	中止中断及DMA方式下的数据传输，中断方式。
中断处理及回调函数	hal_qspi_irq_handler()	中断处理函数。
	hal_qspi_tx_cplt_callback()	发送完成中断回调函数。
	hal_qspi_rx_cplt_callback()	接收完成中断回调函数。
	hal_qspi_error_callback()	错误中断回调函数。
	hal_qspi_abort_cplt_callback()	中止完成中断回调函数。
状态及错误	hal_qspi_get_state()	获取驱动运行状态。
	hal_qspi_get_error()	获取错误码。
控制	hal_qspi_set_timeout()	设置超时时间。
	hal_qspi_set_tx_fifo_threshold()	设置TX FIFO阈值。
	hal_qspi_set_rx_fifo_threshold()	设置RX FIFO阈值。
	hal_qspi_get_tx_fifo_threshold()	获取TX FIFO阈值。
	hal_qspi_get_rx_fifo_threshold()	获取RX FIFO阈值。
睡眠相关	hal_qspi_suspend_reg()	睡眠时挂起QSPI配置相关的寄存器。
	hal_qspi_resume_reg()	唤醒时恢复QSPI配置相关的寄存器。

下面章节将对各API进行详细描述。

2.17.4.1 hal_qspi_init

表 2-278 hal_qspi_init接口

函数原型	hal_status_t hal_qspi_init(qspi_handle_t *p_qspi)
功能说明	根据qspi_init_t中的指定参数初始化QSPI模式，并初始化相关句柄。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	HAL状态。
备注	

2.17.4.2 hal_qspi_deinit

表 2-279 hal_qspi_deinit接口

函数原型	hal_status_t hal_qspi_deinit(qspi_handle_t *p_qspi)
功能说明	反初始化QSPI外设。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	HAL状态。
备注	

2.17.4.3 hal_qspi_msp_init

表 2-280 hal_qspi_msp_init接口

函数原型	void hal_qspi_msp_init(qspi_handle_t *p_qspi)
功能说明	初始化QSPI所使用的GPIO引脚复用、NVIC中断、DMA通道等配置。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成GPIO引脚复用、NVIC中断、DMA通道的初始化。

2.17.4.4 hal_qspi_msp_deinit

表 2-281 hal_qspi_msp_deinit接口

函数原型	void hal_qspi_msp_deinit(qspi_handle_t *p_qspi)
功能说明	反初始化QSPI所使用的GPIO引脚复用、NVIC中断、DMA通道等配置。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成GPIO引脚复用、NVIC中断、DMA通道的反初始化。

2.17.4.5 hal_qspi_command_transmit

表 2-282 hal_qspi_command_transmit接口

函数原型	hal_status_t hal_qspi_command_transmit(qspi_handle_t *p_qspi, qspi_command_t *p_cmd, uint8_t *p_data, uint32_t timeout)
功能说明	发送数据（包含：指令，地址，数据），轮询方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_cmd：指向qspi_command_t结构体变量的指针，该结构体变量包含命令配置信息。 p_data：指向数据缓冲区的指针。 timeout：超时时间。
返回值	HAL状态。
备注	

2.17.4.6 hal_qspi_command_receive

表 2-283 hal_qspi_command_receive接口

函数原型	hal_status_t hal_qspi_command_receive(qspi_handle_t *p_qspi, qspi_command_t *p_cmd, uint8_t *p_data, uint32_t timeout)
功能说明	接收数据(包含：指令，地址，dummy，数据)，轮询方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_cmd：指向qspi_command_t结构体变量的指针，该结构体变量包含命令配置信息。 p_data：指向数据缓冲区的指针。 timeout：超时时间。
返回值	HAL状态。
备注	

2.17.4.7 hal_qspi_command

表 2-284 hal_qspi_command接口

函数原型	hal_status_t hal_qspi_command(qspi_handle_t *p_qspi, qspi_command_t *p_cmd, uint32_t timeout)
功能说明	发送指令，轮询方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_cmd：指向qspi_command_t结构体变量的指针，该结构体变量包含命令配置信息。 timeout：超时时间。
返回值	HAL状态。
备注	该接口只用于轮询方式下的指令发送，可与hal_qspi_transmit()及hal_qspi_receive()配合使用进行指令、地址、数据的收发，或者可直接使用hal_qspi_command_transmit()和hal_qspi_command_receive()进行指令、地址、数据的收发。

2.17.4.8 hal_qspi_transmit

表 2-285 hal_qspi_transmit接口

函数原型	hal_status_t hal_qspi_transmit(qspi_handle_t *p_qspi, uint8_t *p_data, uint32_t length, uint32_t timeout)
功能说明	以SPI方式发送大量数据，轮询方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待发送数据的长度。 timeout：超时时间。
返回值	HAL状态。
备注	该函数只在Standard SPI模式里使用。

2.17.4.9 hal_qspi_receive

表 2-286 hal_qspi_receive接口

函数原型	hal_status_t hal_qspi_receive(qspi_handle_t *p_qspi, uint8_t *p_data, uint32_t length, uint32_t timeout)
功能说明	以SPI方式接收大量数据，轮询方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待接收数据的长度。 timeout：超时时间。
返回值	HAL状态。
备注	该函数只在Standard SPI模式里使用。

2.17.4.10 hal_qspi_command_transmit_it

表 2-287 hal_qspi_command_transmit_it接口

函数原型	hal_status_t hal_qspi_command_transmit_it(qspi_handle_t *p_qspi, qspi_command_t *p_cmd, uint8_t *p_data)
功能说明	发送数据（(包含：指令，地址，数据)，中断方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_cmd：指向qspi_command_t结构体变量的指针，该结构体变量包含命令配置信息。 p_data：指向数据缓冲区的指针。
返回值	HAL状态。
备注	传输过程中，QSPI中断处理无法及时响应，可能出现发送数据错误的情况。

2.17.4.11 hal_qspi_command_receive_it

表 2-288 hal_qspi_command_receive_it接口

函数原型	hal_status_t hal_qspi_command_receive_it(qspi_handle_t *p_qspi, qspi_command_t *p_cmd, uint8_t *p_data)
功能说明	接收数据（(包含：指令，地址，dummy，数据)，中断方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_cmd：指向qspi_command_t结构体变量的指针，该结构体变量包含命令配置信息。 p_data：指向数据缓冲区的指针。
返回值	HAL状态。
备注	传输过程中，QSPI中断处理无法及时响应，可能出现接收数据错误的情况。

2.17.4.12 hal_qspi_command_it

表 2-289 hal_qspi_command_it接口

函数原型	hal_status_t hal_qspi_command_it(qspi_handle_t *p_qspi, qspi_command_t *p_cmd)
功能说明	发送指令，中断方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_cmd：指向qspi_command_t结构体变量的指针，该结构体变量包含命令配置信息。
返回值	HAL状态。
备注	该接口只用于中断方式下的指令发送，可与hal_qspi_receive_it()及hal_qspi_transmit_it()配合使用进行指令、地址、数据的收发，或者直接使用hal_qspi_command_receive_it()和hal_qspi_command_transmit_it()进行指令、地址、数据的收发。

2.17.4.13 hal_qspi_transmit_it

表 2-290 hal_qspi_transmit_it接口

函数原型	hal_status_t hal_qspi_transmit_it(qspi_handle_t *p_qspi, uint8_t *p_data, uint32_t length)
功能说明	以SPI方式发送数据，中断方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待发送数据的长度。
返回值	HAL状态。
备注	• 该函数只在Standard SPI模式里使用。 • 传输过程中，QSPI中断处理无法及时响应，可能出现发送数据错误的情况。

2.17.4.14 hal_qspi_receive_it

表 2-291 hal_qspi_receive_it接口

函数原型	hal_status_t hal_qspi_receive_it(qspi_handle_t *p_qspi, uint8_t *p_data, uint32_t length)
功能说明	以SPI方式接收数据，中断方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待接收数据的长度。
返回值	HAL状态。
备注	• 该函数只在Standard SPI模式里使用。 • 传输过程中，QSPI中断处理无法及时响应，可能出现接收数据错误的情况。

2.17.4.15 hal_qspi_command_transmit_dma

表 2-292 hal_qspi_command_transmit_dma接口

函数原型	hal_status_t hal_qspi_command_transmit_dma(qspi_handle_t *p_qspi, qspi_command_t *p_cmd, uint8_t *p_data)
功能说明	发送数据(包含：指令，地址，数据)，DMA方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_cmd：指向qspi_command_t结构体变量的指针，该结构体变量包含命令配置信息。 p_data：指向数据缓冲区的指针。
返回值	HAL状态。
备注	

2.17.4.16 hal_qspi_command_receive_dma

表 2-293 hal_qspi_command_receive_dma接口

函数原型	hal_status_t hal_qspi_command_receive_dma(qspi_handle_t *p_qspi, qspi_command_t *p_cmd, uint8_t *p_data)
功能说明	接收数据（包含：指令，地址，dummy，数据），DMA方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_cmd：指向qspi_command_t结构体变量的指针，该结构体变量包含命令配置信息。 p_data：指向数据缓冲区的指针。
返回值	HAL状态。
备注	

2.17.4.17 hal_qspi_command_dma

表 2-294 hal_qspi_command_dma接口

函数原型	hal_status_t hal_qspi_command_dma(qspi_handle_t *p_qspi, qspi_command_t *p_cmd)
功能说明	发送指令，DMA方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_cmd：指向qspi_command_t结构体变量的指针，该结构体变量包含命令配置信息。
返回值	HAL状态。
备注	该接口只用于DMA方式下的指令发送，可与hal_qspi_receive_dma()及hal_qspi_transmit_dma()配合使用进行指令、地址、数据的收发，或者可直接使用hal_qspi_command_receive_dma()和hal_qspi_command_transmit_dma()进行指令、地址、数据的收发。

2.17.4.18 hal_qspi_transmit_dma

表 2-295 hal_qspi_transmit_dma接口

函数原型	hal_status_t hal_qspi_transmit_dma(qspi_handle_t *p_qspi, uint8_t *p_data, uint32_t length)
功能说明	以SPI方式发送数据，DMA方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待发送数据的长度。
返回值	HAL状态。
备注	该函数只在Standard SPI模式里使用： • 如果DMA外围设备访问被配置为半字节，那么数据长度和fifo阈值应该以半字节对齐。 • 如果DMA外围设备访问被配置为字节，那么数据长度和fifo阈值应该以字节对齐。

2.17.4.19 hal_qspi_receive_dma

表 2-296 hal_qspi_receive_dma接口

函数原型	hal_status_t hal_qspi_receive_dma(qspi_handle_t *p_qspi, uint8_t *p_data, uint32_t length)
功能说明	以SPI方式接收数据，DMA方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待接收数据的长度。
返回值	HAL状态。
备注	该函数只在Standard SPI模式里使用： • 如果DMA外围设备访问被配置为半字节，那么数据长度和fifo阈值应该以半字节对齐。 • 如果DMA外围设备访问被配置为字节，那么数据长度和fifo阈值应该以字节对齐。

2.17.4.20 hal_qspi_abort

表 2-297 hal_qspi_abort接口

函数原型	hal_status_t hal_qspi_abort(qspi_handle_t *p_qspi)
功能说明	中止中断及DMA方式下的数据传输，轮询方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	HAL状态。
备注	

2.17.4.21 hal_qspi_abort_it

表 2-298 hal_qspi_abort_it接口

函数原型	hal_status_t hal_qspi_abort_it(qspi_handle_t *p_qspi)
功能说明	中止中断及DMA方式下的数据传输，中断方式。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	HAL状态。
备注	

2.17.4.22 hal_qspi_irq_handler

表 2-299 hal_qspi_irq_handler接口

函数原型	void hal_qspi_irq_handler(qspi_handle_t *p_qspi)
功能说明	处理QSPI中断请求。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	无
备注	

2.17.4.23 hal_qspi_tx_cplt_callback

表 2-300 hal_qspi_tx_cplt_callback接口

函数原型	void hal_qspi_tx_cplt_callback(qspi_handle_t *p_qspi)
功能说明	发送完成回调。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.17.4.24 hal_qspi_rx_cplt_callback

表 2-301 hal_qspi_rx_cplt_callback接口

函数原型	void hal_qspi_rx_cplt_callback(qspi_handle_t *p_qspi)
功能说明	接收完成回调。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.17.4.25 hal_qspi_error_callback

表 2-302 hal_qspi_error_callback接口

函数原型	void hal_qspi_error_callback(qspi_handle_t *p_qspi)
功能说明	传输错误回调。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.17.4.26 hal_qspi_abort_cplt_callback

表 2-303 hal_qspi_abort_cplt_callback接口

函数原型	void hal_qspi_abort_cplt_callback(qspi_handle_t *p_qspi)
功能说明	中止完成回调。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.17.4.27 hal_qspi_get_state

表 2-304 hal_qspi_get_state接口

函数原型	hal_qspi_state_t hal_qspi_get_state(qspi_handle_t *p_qspi)
功能说明	获取QSPI运行状态。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	QSPI运行状态，该参数的取值可以是下列值中的任意一个： • HAL_QSPI_STATE_RESET（未初始化） • HAL_QSPI_STATE_READY（已初始化且空闲） • HAL_QSPI_STATE_BUSY（忙） • HAL_QSPI_STATE_BUSY_INDIRECT_TX（正在发送） • HAL_QSPI_STATE_BUSY_INDIRECT_RX （正在接收）

	• HAL_QSPI_STATE_ABORT（被中断） • HAL_QSPI_STATE_ERROR（错误）
备注	

2.17.4.28 hal_qspi_get_error

表 2-305 hal_qspi_get_error接口

函数原型	uint32_t hal_qspi_get_error(qspi_handle_t *p_qspi)
功能说明	获取QSPI错误码。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	QSPI错误代码，该参数的取值可以是下列值中的任意一个： • HAL_QSPI_ERROR_NONE（无错误） • HAL_QSPI_ERROR_TIMEOUT（超时） • HAL_QSPI_ERROR_TRANSFER（传输错误） • HAL_QSPI_ERROR_DMA（DMA传输错误） • HAL_QSPI_ERROR_INVALID_PARAM（非法参数）
备注	

2.17.4.29 hal_qspi_set_timeout

表 2-306 hal_qspi_set_timeout接口

函数原型	void hal_qspi_set_timeout(qspi_handle_t *p_qspi, uint32_t timeout)
功能说明	设置QSPI超时时间。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 timeout：QSPI内存访问超时时间。
返回值	无
备注	

2.17.4.30 hal_qspi_set_tx_fifo_threshold

表 2-307 hal_qspi_set_tx_fifo_threshold接口

函数原型	hal_status_t hal_qspi_set_tx_fifo_threshold(qspi_handle_t *p_qspi, uint32_t threshold)
功能说明	设置QSPI TX FIFO阈值。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 threshold：待设置的TX FIFO阈值（取值范围0 ～ 7，其中0表示TX FIFO为空，7表示TX FIFO差一个字节满）。
返回值	HAL状态
备注	

2.17.4.31 hal_qspi_set_rx_fifo_threshold

表 2-308 hal_qspi_set_rx_fifo_threshold接口

函数原型	hal_status_t hal_qspi_set_rx_fifo_threshold(qspi_handle_t *p_qspi, uint32_t threshold)
功能说明	设置QSPI RX FIFO阈值。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。 threshold：待设置的RX FIFO阈值（取值范围0 ～ 7，其中0表示RX FIFO中有一个字节，7表示RX FIFO满）。
返回值	HAL状态
备注	

2.17.4.32 hal_qspi_get_tx_fifo_threshold

表 2-309 hal_qspi_get_tx_fifo_threshold接口

函数原型	uint32_t hal_qspi_get_tx_fifo_threshold(qspi_handle_t *p_qspi)
功能说明	获取QSPI TX FIFO阈值。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	TX FIFO阈值（取值范围0 ～ 7，其中0表示TX FIFO为空，7表示TX FIFO差一个字节满）。
备注	

2.17.4.33 hal_qspi_get_rx_fifo_threshold

表 2-310 hal_qspi_get_rx_fifo_threshold接口

函数原型	uint32_t hal_qspi_get_rx_fifo_threshold(qspi_handle_t *p_qspi)
功能说明	获取QSPI RX FIFO阈值。
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	RX FIFO阈值（取值范围0 ～ 7，其中0表示RX FIFO中有一个字节，7表示RX FIFO满）。
备注	

2.17.4.34 hal_qspi_suspend_reg

表 2-311 hal_qspi_suspend_reg接口

函数原型	hal_status_t hal_qspi_suspend_reg(qspi_handle_t *p_qspi)
功能说明	睡眠时挂起QSPI配置相关的寄存器
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	HAL状态
备注	

2.17.4.35 hal_qspi_resume_reg

表 2-312 hal_qspi_resume_reg接口

函数原型	hal_status_t hal_qspi_resume_reg(qspi_handle_t *p_qspi)
功能说明	唤醒时恢复QSPI配置相关的寄存器
输入参数	p_qspi：指向qspi_handle_t结构体变量的指针，该结构体变量包含指定的QSPI的配置信息。
返回值	HAL状态
备注	

2.18 HAL PWM通用驱动

2.18.1 PWM驱动功能

PWM（Pulse Width Modulation）外设的HAL驱动主要实现了以下功能：

- 支持时钟频率最高与系统时钟相等。
- 配备2个PWM模块，每个模块提供3个输出通道。
- 可配置的输出频率，支持动态更新输出频率。
- 支持两种输出方式：固定占空比模式和呼吸模式（占空比变化：0→100%→0，如此循环）。
- 支持两种对齐方式：每个周期的占空比以左边缘对齐和以中心对齐。
- 支持输出暂停。

2.18.2 如何使用PWM驱动

PWM驱动的使用方法如下：

1. 声明一个pwm_handle_t句柄结构体变量，例如：pwm_handle_t pwm_handle。
2. 重写hal_pwm_msp_init()以初始化PWM底层资源：使用hal_gpio_int()配置PWM各通道对应GPIO引脚模式为GPIO_PIN_MUX（复用模式），并设置对应的mux模式。
3. 配置pwm_handle中init初始化结构体中的输出模式、对齐方式、输出频率、输出通道，若普通占空比输出模式还需配置通道占空比和输出极性；若输出模式为呼吸模式，则还需配置Breath周期和Hold周期。
4. 调用hal_pwm_init(&pwm_handle)配置PWM寄存器，配置过程中hal_pwm_init()会自动调用开发者重写的hal_pwm_msp_init()函数初始化PWM所使用的GPIO引脚等底层资源。
5. 声明一个pwm_channel_init_t通道初始化结构体，例如：pwm_channel_init_t channel_init。
6. 根据输出模式对channel_init中的通道占空比及输出极性进行配置：
   - 固定占空比模式：配置通道占空比及输出极性。

- 呼吸模式：仅需配置输出极性。

7. 调用hal_pwm_config_channel(&pwm_handle,&channel_init,HAL_PWM_ACTIVE_CHANNEL_x)配置输出通道HAL_PWM_ACTIVE_CHANNEL_x，其中x可以是A或B或C或ALL。
8. 调用hal_pwm_start()使能PWM输出。
9. 调用hal_pwm_stop()暂停PWM输出，此时开发者还可调用hal_pwm_config_channel()修改输出通道配置。

2.18.3 PWM驱动的结构体

2.18.3.1 pwm_channel_init_t

PWM驱动的通道描述结构体pwm_channel_init_t的定义如下：

表 2-313 pwm_channel_init_t结构体

数据域	域段描述	取值
uint8_t duty	占空比	0 ～ 100
uint8_t drive_polarity	驱动极性	该参数的取值可以是下列值中的任意一个： • PWM_DRIVEPOLARITY_NEGATIVE（反相驱动） • PWM_DRIVEPOLARITY_POSITIVE（正相驱动）

2.18.3.2 pwm_init_t

PWM驱动的初始化结构体pwm_init_t的定义如下：

表 2-314 pwm_init_t结构体

数据域	域段描述	取值
uint32_t mode	PWM输出模式	该参数的取值可以是下列值中的任意一个： • PWM_MODE_FLICKER（固定占空比） • PWM_MODE_BREATH（呼吸输出）
uint32_t align	PWM对齐方式。	该参数的取值可以是下列值中的任意一个： • PWM_ALIGNED_EDGE（左边缘对齐） • PWM_ALIGNED_CENTER（中心对齐）
uint32_t freq	PWM输出频率	1 ～ 32000000，建议小于500000。
uint32_t bperiod	呼吸周期	1 ～ 67108@64MHz 1 ～ 89478@48MHz 1 ～ 134217@32MHz 1 ～ 178956@24MHz 1 ～ 268435@16MHz 单位ms。

数据域	域段描述	取值
uint32_t hperiod	屏息周期	1 ～ 262@64MHz 1 ～ 349@48MHz 1 ～ 524@32MHz 1 ～ 699@24MHz 1 ～ 1048@16MHz 单位ms。
pwm_channel_init_t channel_a	输出通道A配置参数	参考pwm_channel_init_t
pwm_channel_init_t channel_b	输出通道B配置参数	参考pwm_channel_init_t
pwm_channel_init_t channel_c	输出通道C配置参数	参考pwm_channel_init_t

2.18.3.3 pwm_handle_t

PWM驱动的句柄结构体pwm_handle_t的定义如下：

表 2-315 pwm_handle_t结构体

数据域	域段描述	取值
pwm_regs_t *p_instance	PWM外设实例	该参数的取值可以是下列值中的任意一个： • PWM0 • PWM1
pwm_init_t init	PWM初始化结构体	参考pwm_init_t
hal_pwm_active_channel_t active_channel	PWM使能输出通道	该参数的取值可以是下列值中的任意一个： • HAL_PWM_ACTIVE_CHANNEL_A • HAL_PWM_ACTIVE_CHANNEL_B • HAL_PWM_ACTIVE_CHANNEL_C • HAL_PWM_ACTIVE_CHANNEL_ALL • HAL_PWM_ACTIVE_CHANNEL_CLEARED
__IO hal_lock_t lock	PWM锁（驱动负责管理，无需开发者初始化）	N/A
__IO hal_pwm_state_t state	PWM运行状态（无需开发者初始化）	该参数的取值可以是下列值中的任意一个： • HAL_PWM_STATE_RESET（未初始化） • HAL_PWM_STATE_RESET（未初始化） • HAL_PWM_STATE_READY（已初始化且空闲） • HAL_PWM_STATE_BUSY（忙） • HAL_PWM_STATE_ERROR（错误）
uint32_t retention[11]	保存PWM寄存器信息（驱动负责管理，无需开发者初始化）	N/A

2.18.4 PWM驱动API描述

PWM驱动的API主要包括：

表 2-316 PWM驱动的APIs

API类别	API名称	描述
初始化	hal_pwm_init()	初始化PWM外设，配置输出参数。
	hal_pwm_deinit()	反初始化PWM外设。
	hal_pwm_msp_init()	初始化PWM外设所使用的GPIO。
	hal_pwm_msp_deinit()	反初始化PWM外设所使用的GPIO。
IO操作	hal_pwm_start()	使能PWM输出。
	hal_pwm_stop()	停止PWM输出。
	hal_pwm_update_freq()	更新PWM输出频率。
	hal_pwm_config_channel()	配置PWM通道参数。
状态及错误	hal_pwm_get_state()	获取驱动运行状态。
睡眠相关	hal_pwm_suspend_reg()	睡眠时挂起PWM配置相关的寄存器。
	hal_pwm_resume_reg()	唤醒时恢复PWM配置相关的寄存器。

下面章节将对各API进行详细描述。

2.18.4.1 hal_pwm_init

表 2-317 hal_pwm_init接口

函数原型	hal_status_t hal_pwm_init(pwm_handle_t *p_pwm)
功能说明	根据pwm_init_t里的参数初始化PWM外设和初始化关联句柄。
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针，该结构体变量包含指定的PWM的配置信息。
返回值	HAL状态。
备注	

2.18.4.2 hal_pwm_deinit

表 2-318 hal_pwm_deinit接口

函数原型	hal_status_t hal_pwm_deinit(pwm_handle_t *p_pwm)
功能说明	反初始化PWM外设。
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针，该结构体变量包含指定的PWM的配置信息。
返回值	HAL状态。
备注	

2.18.4.3 hal_pwm_msp_init

表 2-319 hal_pwm_msp_init接口

函数原型	void hal_pwm_msp_init(pwm_handle_t *p_pwm)
功能说明	初始化PWM外设所使用的GPIO等配置。
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成GPIO的初始化。

2.18.4.4 hal_pwm_msp_deinit

表 2-320 hal_pwm_msp_deinit接口

函数原型	void hal_pwm_msp_deinit(pwm_handle_t *p_pwm)
功能说明	反初始化PWM外设所使用的GPIO等配置。
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成GPIO的反初始化。

2.18.4.5 hal_pwm_start

表 2-321 hal_pwm_start接口

函数原型	hal_status_t hal_pwm_start(pwm_handle_t *p_pwm)
功能说明	使能PWM输出。
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针，该结构体变量包含指定的PWM的配置信息。
返回值	HAL状态。
备注	当输出通道的参数配置好后调用该接口，PWM通道输出波形。

2.18.4.6 hal_pwm_stop

表 2-322 hal_pwm_stop接口

函数原型	hal_status_t hal_pwm_stop(pwm_handle_t *p_pwm)
功能说明	停止PWM输出
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针，该结构体变量包含指定的PWM的配置信息。
返回值	HAL状态。
备注	PWM输出波形时，调用该接口可以停止PWM波形输出。

2.18.4.7 hal_pwm_update_freq

表 2-323 hal_pwm_update_freq接口

函数原型	hal_status_t hal_pwm_update_freq(pwm_handle_t *p_pwm, uint32_t freq)
功能说明	更新PWM输出频率
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针，该结构体变量包含指定的PWM的配置信息。 freq：输出频率。取值范围：1 ~ 32000000，建议小于500000。
返回值	HAL状态。
备注	PWM输出波形时，调用该接口可以改变PWM波形输出频率。

2.18.4.8 hal_pwm_config_channel

表 2-324 hal_pwm_config_channel接口

函数原型	hal_status_t hal_pwm_config_channel(pwm_handle_t *p_pwm, pwm_channel_init_t *p_config, hal_pwm_active_channel_t channel)
功能说明	配置PWM输出通道。
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针，该结构体变量包含指定的PWM的配置信息。 p_config：指向pwm_channel_init_t结构体变量的指针，该结构体变量包含PWM通道的配置信息。 channel：待配置的通道。
返回值	HAL状态。
备注	PWM输出停止时，调用该接口可以重新配置通道参数。

2.18.4.9 hal_pwm_get_state

表 2-325 hal_pwm_get_state接口

函数原型	hal_pwm_state_t hal_pwm_get_state(pwm_handle_t *p_pwm)
功能说明	获取PWM运行状态。
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针，该结构体变量包含指定的PWM的配置信息。
返回值	PWM运行状态，参数的取值可以是下列值中的任意一个： • HAL_PWM_STATE_RESET（未初始化） • HAL_PWM_STATE_READY（已初始化且空闲） • HAL_PWM_STATE_BUSY（忙） • HAL_PWM_STATE_ERROR（错误）
备注	

2.18.4.10 hal_pwm_suspend_reg

表 2-326 hal_pwm_suspend_reg接口

函数原型	hal_status_t hal_pwm_suspend_reg(pwm_handle_t *p_pwm)

功能说明	睡眠时挂起PWM配置相关的寄存器。
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针，该结构体变量包含指定的PWM的配置信息。
返回值	PWM运行状态，参数的取值可以是下列值中的任意一个： • HAL_PWM_STATE_RESET（未初始化） • HAL_PWM_STATE_READY（已初始化且空闲） • HAL_PWM_STATE_BUSY（忙） • HAL_PWM_STATE_ERROR（错误）
备注	

2.18.4.11 hal_pwm_resume_reg

表 2-327 hal_pwm_resume_reg接口

函数原型	hal_status_t hal_pwm_resume_reg(pwm_handle_t *p_pwm)
功能说明	唤醒时恢复PWM配置相关的寄存器。
输入参数	p_pwm：指向pwm_handle_t结构体变量的指针，该结构体变量包含指定的PWM的配置信息。
返回值	PWM运行状态，参数的取值可以是下列值中的任意一个： • HAL_PWM_STATE_RESET（未初始化） • HAL_PWM_STATE_READY（已初始化且空闲） • HAL_PWM_STATE_BUSY（忙） • HAL_PWM_STATE_ERROR（错误）
备注	

2.19 HAL PWR 通用驱动

2.19.1 PWR驱动功能

PWR外设的HAL驱动主要实现了以下功能：

- 支持六种深度休眠唤醒模式：AON_GPIO、AON SLEEP TIMER、BLE_TIMER、CALENDAR、COMP、BOD。
- 支持4种AON GPIO唤醒方式：高电平唤醒、低电平唤醒、上升沿唤醒、下降沿唤醒。
- 支持设置AON SLEEP TIMER的唤醒时间。
- 支持BLE Core及BLE Timer的电源管理及模式切换。

2.19.2 如何使用PWR驱动

PWR驱动主要用于控制MCU中的BLE Core、BLE Timer的电源及深度休眠模式。开发者根据需要调用相关的接口。

2.19.2.1 BLE电源配置

GR551x系列芯片可分别对BLE Core及BLE Timer的电源进行管理，支持的BLE电源状态包括：上电态、掉电态。

- 上电态（Power On）：BLE Core/Timer供电开启，处于正常运行状态。
- 掉电态（Power Down）：BLE Core/Timer供电停止，处于停止运行状态。

此外，芯片还分别支持BLE Core及BLE Timer的复位模式及运行模式的配置，可实现BLE Core及BLE Timer在两种模式下的切换。

开发者可调用hal_pwr_set_comm_power()对BLE Core和BLE Timer的电源进行管理，可调用hal_pwr_set_comm_mode()进行模式切换。

2.19.2.2 深度休眠配置

GR551x系列芯片支持深度休眠模式，深度休眠模式下MCU子系统中所有外设及BLE Core掉电，此时，芯片处于低功耗模式。

在进入深度休眠模式前，需要配置唤醒条件，支持的唤醒条件包括：External、Timer、BLE、External + Timer + BLE：

- External：可通过AON GPIO唤醒，需要设置用于唤醒的引脚、唤醒类型。
- Timer：可通过AON SLEEP TIMER唤醒，需要设置用于唤醒MCU的时间间隔，其中，该AON SLEEP TIMER的时钟频率为40 kHz。
- BLE：可通过BLE TIMER唤醒。
- External + Timer + BLE：可通过AON GPIO或AON SLEEP TIMER或BLE TIMER唤醒，需要设置用于唤醒的引脚、唤醒类型及用于唤醒MCU的时间间隔。

开发者可调用hal_pwr_set_wakeup_condition()配置唤醒条件；如果唤醒条件包含External，则需要调用hal_pwr_config_timer_wakeup()对AON GPIO引脚及唤醒类型进行配置；如果唤醒条件包含Timer，则需要调用hal_pwr_config_ext_wakeup()对AON SLEEP TIMER的计数值进行配置。

2.19.3 PWR驱动API描述

PWR驱动的API主要包括：

表 2-328 PWR驱动的APIs

API类别	API名称	描述
控制	hal_pwr_set_wakeup_condition()	设置深度休眠的唤醒条件。
	hal_pwr_config_timer_wakeup()	配置AON Sleep Timer的唤醒参数。
	hal_pwr_config_ext_wakeup()	配置AON GPIO的唤醒参数。
	hal_pwr_set_comm_power()	设置BLE Core和BLE Timer的电源状态。
	hal_pwr_set_comm_mode()	设置BLE Core和BLE Timer的模式。

API类别	API名称	描述
	hal_pwr_enter_chip_deepsleep()	进入深度休眠。
	hal_pwr_get_timer_current_value()	获取当前定时器值。
	hal_pwr_disable_ext_wakeup()	禁止指定的AON GPIO唤醒系统
中断处理及回调函数	hal_pwr_sleep_timer_irq_handler()	SleepTimer中断处理函数。
	hal_pwr_sleep_timer_elapsed_callback()	SleepTimer中断回调函数。

下面章节将对各API进行详细描述。

2.19.3.1 hal_pwr_set_wakeup_condition

表 2-329 hal_pwr_set_wakeup_condition接口

函数原型	void hal_pwr_set_wakeup_condition(uint32_t condition)
功能说明	设置深度休眠模式下的唤醒条件。
输入参数	condition：唤醒条件，该参数可以是下列值中的任意一个： • PWR_WKUP_COND_EXT（AON_GPIO） • PWR_WKUP_COND_TIMER（AON Sleep Timer） • PWR_WKUP_COND_BLE（BLE Timer） • PWR_WKUP_COND_CALENDAR（Calendar Timer） • PWR_WKUP_COND_BOD_FEDGE（PMU Bod） • PWR_WKUP_COND_MSIO_COMP（Comparator） • PWR_WKUP_COND_ALL（所有睡眠唤醒源）
返回值	无
备注	

2.19.3.2 hal_pwr_config_timer_wakeup

表 2-330 hal_pwr_config_timer_wakeup接口

函数原型	void hal_pwr_config_timer_wakeup(uint8_t timer_mode, uint32_t load_count)
功能说明	设置深度休眠模式下用于唤醒MCU的AON SLEEP TIMER的计数值。
输入参数	timer_mode: Sleep timer的计数模式，该参数可以是下列值中的任意一个： • PWR_SLP_TIMER_MODE_NORMAL（睡眠模式，深度休眠时计数，唤醒后停止计数） • PWR_SLP_TIMER_MODE_SINGLE（单次计数，唤醒后继续计数） • PWR_SLP_TIMER_MODE_RELOAD（自动加载模式，唤醒后继续计数） • PWR_SLP_TIMER_MODE_DISABLE（禁用） load_count：进入深度休眠后，唤醒MCU的时间计数值，取值范围为0 ～ 0xFFFFFFFFU。
返回值	无
备注	该API仅在唤醒条件包含AON TIMER时可用。

2.19.3.3 hal_pwr_config_ext_wakeup

表 2-331 hal_pwr_config_ext_wakeup接口

函数原型	void hal_pwr_config_ext_wakeup(uint32_t ext_wakeup_pinx, uint32_t ext_wakeup_type)
功能说明	设置深度休眠模式下用于唤醒MCU的AON_GPIO的引脚及唤醒类型。
输入参数	ext_wakeup_pinx：唤醒MCU的AON_GPIO的引脚，该参数可以是下列值的组合： • PWR_EXTWKUP_PIN0（AON_GPIO引脚0） • PWR_EXTWKUP_PIN1（AON_GPIO引脚1） • PWR_EXTWKUP_PIN2（AON_GPIO引脚2） • PWR_EXTWKUP_PIN3（AON_GPIO引脚3） • PWR_EXTWKUP_PIN4（AON_GPIO引脚4） • PWR_EXTWKUP_PIN5（AON_GPIO引脚5） • PWR_EXTWKUP_PIN6（AON_GPIO引脚6） • PWR_EXTWKUP_PIN7（AON_GPIO引脚7） ext_wakeup_type：唤醒MCU的AON_GPIO的唤醒类型，该参数可以是下列值中的任意一个： • PWR_EXTWKUP_TYPE_LOW（低电平触发） • PWR_EXTWKUP_TYPE_HIGH（高电平触发） • PWR_EXTWKUP_TYPE_RISING（上升沿触发） • PWR_EXTWKUP_TYPE_FALLING（下降沿触发）
返回值	无
备注	该API仅在唤醒条件包含AON GPIO时可用。

2.19.3.4 hal_pwr_set_comm_power

表 2-332 hal_pwr_set_comm_power接口

函数原型	void hal_pwr_set_comm_power(uint32_t timer_power_state, uint32_t core_power_state)
功能说明	设置BLE Core和BLE Timer的电源状态。
输入参数	timer_power_state：BLE Timer的电源状态，该参数可以是下列值中的任意一个： • PWR_COMM_TIMER_POWER_DOWN（BLE Timer掉电） • PWR_COMM_TIMER_POWER_UP（BLE Timer上电） core_power_state：BLE Core的电源状态，该参数可以是下列值中的任意一个： • PWR_COMM_CORE_POWER_DOWN（BLE Core掉电） • PWR_COMM_CORE_POWER_UP（BLE Core上电）
返回值	无
备注	如果需要使用BLE功能，则需要在MCU启动后设置BLE Core和BLE Timer的电源状态为上电状态。

2.19.3.5 hal_pwr_set_comm_mode

表 2-333 hal_pwr_set_comm_mode接口

函数原型	void hal_pwr_set_comm_mode(uint32_t timer_mode, uint32_t core_mode)

功能说明	设置BLE Core和BLE Timer的模式。
输入参数	timer_mode：BLE Timer的模式，该参数可以是下列值中的任意一个： • PWR_COMM_TIMER_MODE_RESET（BLE Timer复位） • PWR_COMM_TIMER_MODE_RUNNING（BLE Timer运行） core_mode：BLE Core的模式，该参数可以是下列值中的任意一个 • PWR_COMM_CORE_MODE_RESET（BLE Core复位） • PWR_COMM_CORE_MODE_RUNNING（BLE Core运行）
返回值	无
备注	如果需要使用BLE功能，则需要在BLE Core和BLE Timer上电后设置其为运行模式。

2.19.3.6 hal_pwr_enter_chip_deepsleep

表 2-334 hal_pwr_enter_chip_deepsleep接口

函数原型	void hal_pwr_enter_chip_deepsleep(void)
功能说明	设置MCU进入深度休眠模式，并设置深度休眠模式下需要保持的内存块及唤醒后需要满电的内存块。
输入参数	无
返回值	无
备注	无

2.19.3.7 hal_pwr_get_timer_current_value

表 2-335 hal_pwr_get_timer_current_value接口

函数原型	hal_status_t hal_pwr_get_timer_current_value(uint32_t timer_type, uint32_t *p_value)
功能说明	获取当前计数器数值。
输入参数	timer_type：计数器类型，该参数可以是下列值中的任意一个： • PWR_TIMER_TYPE_CAL_TIMER（CAL Timer） • PWR_TIMER_TYPE_AON_WDT（AON_WDT Timer） • PWR_TIMER_TYPE_SLP_TIMER（SLEEP Timer） • PWR_TIMER_TYPE_CAL_ALARM（CAL Alarm） P_value：内存指针，该参数由开发者指定
返回值	HAL状态。
备注	该接口可获取当前计数器数值。

2.19.3.8 hal_pwr_disable_ext_wakeup

表 2-336 hal_pwr_disable_ext_wakeup接口

函数原型	void hal_pwr_disable_ext_wakeup(uint32_t disable_wakeup_pinx);
功能说明	禁止指定的AON GPIO唤醒系统。

输入参数	disable_wakeup_pinx：特定的AON GPIO pin脚，该参数可以是下列值中的任意一个： • PWR_EXTWKUP_PIN0 • PWR_EXTWKUP_PIN1 • PWR_EXTWKUP_PIN2 • PWR_EXTWKUP_PIN3 • PWR_EXTWKUP_PIN4 • PWR_EXTWKUP_PIN5 • PWR_EXTWKUP_PIN6 • PWR_EXTWKUP_PIN7 • PWR_EXTWKUP_PIN_ALL
返回值	无
备注	无

2.19.3.9 hal_pwr_sleep_timer_irq_handler

表 2-337 hal_pwr_sleep_timer_irq_handler接口

函数原型	void hal_pwr_sleep_timer_irq_handler(void)
功能说明	处理PWR Sleep Timer中断请求。
输入参数	无
返回值	无
备注	

2.19.3.10 hal_pwr_sleep_timer_elapsed_callback

表 2-338 hal_pwr_sleep_timer_elapsed_callback接口

函数原型	void hal_pwr_sleep_timer_elapsed_callback(void)
功能说明	Sleep Timer中断回调函数。
输入参数	无
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.20 HAL SPI通用驱动

2.20.1 SPI驱动功能

SPI（Serial Peripheral Interface）外设的HAL驱动主要实现了以下功能：

- 支持Motorola模式。
- 支持主从模式，主设备支持双从设备选择。
- 支持最大32位的数据传输位宽。

- 支持最高32 MHz的传输速率。
- 支持时钟极性（CPOL）及时钟相位（CPHA）配置。
- 支持全双工、单工发送、单工接收、EEPROM读取四种工作模式。
- 支持发送FIFO和接收FIFO的阈值设置及获取。
- 支持轮询、中断、DMA三种数据读写方式。
- 支持中止中断及DMA方式下的数据读写。
- 支持在错误发生时，执行发送完成、收发完成、中止完成的中断回调函数。
- 支持获取驱动运行状态及错误码。
- 支持超时时间设置。

2.20.2 如何使用SPI驱动

SPI的HAL驱动的使用方法如下：

1. 定义一个spi_handle_t句柄结构体变量，例如：spi_handle_t spi_handle。
2. 重写hal_spi_msp_init()接口以初始化SPI底层资源：
   (1) 配置SPI引脚的功能复用、使能上拉电阻。
   (2) 如果需要使用中断方式的IO操作接口，则需通过调用相关的NVIC接口来配置：
      - 调用hal_nvic_set_priority()配置SPI中断优先级。
      - 调用hal_nvic_enable_irq()使能SPI的NVIC中断。

   (3) 如果需要使用DMA方式的IO操作接口，则还需要配置使用的DMA通道：
      - 定义用于发送/接收的dma_handle_t句柄结构体变量，如dma_handle_t dma_tx，dma_handle_t dma_rx。
      - 配置DMA句柄dma_tx及dma_rx中的参数，如指定TX或RX通道。
      - 将spi_handle变量中的p_dmatx和p_dmarx指针分别指向已初始化的DMA句柄变量dma_tx和dma_rx。
      - 配置DMA的中断优先级、使能DMA的NVIC中断。

3. 配置SPI初始化结构体中的数据传输方向、数据位宽、时钟极性、时钟相位、波特率分频、TI模式以及从设备选择等参数。
4. 调用hal_spi_init(&spi_handle)函数来配置SPI寄存器，hal_spi_init()会自动调用hal_spi_msp_init(&spi_handle)完成SPI底层资源的初始化。
5. 对于SPI的IO读写或IO内存读写操作，SPI的HAL驱动提供三种操作方式：轮询、中断及DMA。

2.20.3 SPI驱动的结构体

2.20.3.1 spi_init_t

SPI驱动的初始化结构体spi_init_t的定义如下：

表 2-339 spi_init_t结构体

数据域	域段描述	取值
uint32_t direction	传输方向。	该参数的取值可以是下列值中的任意一个： • SPI_DIRECTION_SIMPLEX_TX（SPI单工发送） • SPI_DIRECTION_SIMPLEX_RX（SPI单工接收） • SPI_DIRECTION_READ_EEPROM（SPI读EEPROM）
uint32_t data_size	数据位宽。	该参数的取值可以是下列值中的任意一个： • SPI_DATASIZE_4BIT（4位） • SPI_DATASIZE_5BIT（5位） • SPI_DATASIZE_6BIT（6位） • SPI_DATASIZE_7BIT（7位） • SPI_DATASIZE_8BIT（8位） • SPI_DATASIZE_9BIT（9位） • SPI_DATASIZE_10BIT（10位） • SPI_DATASIZE_11BIT（11位） • SPI_DATASIZE_12BIT（12位） • SPI_DATASIZE_13BIT（13位） • SPI_DATASIZE_14BIT（14位） • SPI_DATASIZE_15BIT（15位） • SPI_DATASIZE_16BIT（16位） • SPI_DATASIZE_17BIT（17位） • SPI_DATASIZE_18BIT（18位） • SPI_DATASIZE_19BIT（19位） • SPI_DATASIZE_20BIT（20位） • SPI_DATASIZE_21BIT（21位） • SPI_DATASIZE_22BIT（22位） • SPI_DATASIZE_23BIT（23位） • SPI_DATASIZE_24BIT（24位） • SPI_DATASIZE_25BIT（25位） • SPI_DATASIZE_26BIT（26位） • SPI_DATASIZE_27BIT（27位） • SPI_DATASIZE_28BIT（28位） • SPI_DATASIZE_29BIT（29位） • SPI_DATASIZE_30BIT（30位） • SPI_DATASIZE_31BIT（31位） • SPI_DATASIZE_32BIT（32位）
uint32_t clk_polarity	时钟极性。	该参数的取值可以是下列值中的任意一个： • SPI_POLARITY_LOW（时钟空闲状态为低电平）

数据域	域段描述	取值
		• SPI_POLARITY_HIGH（时钟空闲状态为高电平）
uint32_t clk_phase	时钟相位。	该参数的取值可以是下列值中的任意一个： • SPI_PHASE_1EDGE（在时钟线的第一个跳变处采样数据） • SPI_PHASE_2EDGE（在时钟线的第二个跳变处采样数据）
uint32_t baud_rate_prescaler	波特率分频系数。	0x0000 ～ 0xFFFF之间的偶数 SPI传输速率=系统时钟/分频系数
uint32_t ti_mode	是否启用TI模式。	该参数的取值可以是下列值中的任意一个： • SPI_TIMODE_DISABLE（禁用TI模式） • SPI_TIMODE_ENABLE（启用TI模式）
uint32_t slave_select	从设备选择。	该参数的取值可以是下列值中的任意一个： • SPI_SLAVE_SELECT_0（从设备0） • SPI_SLAVE_SELECT_1（从设备1） • SPI_SLAVE_SELECT_ALL（从设备0和从设备1）

2.20.3.2 spi_handle_t

SPI驱动的句柄结构体spi_handle_t的定义如下：

表 2-340 spi_handle_t结构体

数据域	域段描述	取值
ssi_regs_t *p_instance	SPI外设实例	该参数的取值可以是下列值中的任意一个： • SPIM • SPIS
spi_init_t init	初始化结构体	参考spi_init_t结构体
uint8_t *p_tx_buffer	指向数据发送缓冲区的指针（驱动负责管理，无需开发者初始化）	N/A
__IO uint32_t tx_buffer_size	数据发送长度（驱动负责管理，无需开发者初始化）	N/A
__IO uint32_t tx_xfer_count	数据发送计数（驱动负责管理，无需开发者初始化）	N/A
uint8_t *p_rx_buffer	指向数据接收缓冲区的指针（驱动负责管理，无需开发者初始化）	N/A
__IO uint32_t rx_buffer_size	数据接收长度（驱动负责管理，无需开发者初始化）	N/A
__IO uint32_t rx_xfer_count	数据接收计数（驱动负责管理，无需开发者初始化）	N/A

数据域	域段描述	取值
void (*write_fifo)(struct _spi_handle *p_spi)	指向SPI TX写FIFO函数的指针（驱动负责管理，无需开发者初始化）。	N/A
void (*read_fifo)(struct _spi_handle *p_spi)	指向SPI RX读FIFO函数的指针（驱动负责管理，无需开发者初始化）。	N/A
void (*read_write_fifo)(struct _spi_handle *p_spi)	指向SPI读写FIFO函数的指针（驱动负责管理，无需开发者初始化）。	N/A
dma_handle_t *p_dmatx	指向数据发送通道的DMA句柄的指针	发送通道的DMA句柄dma_handle_t结构体
dma_handle_t *p_dmarx	指向数据接收通道的DMA句柄的指针	接收通道的DMA句柄dma_handle_t结构体
__IO hal_lock_t lock	SPI锁（驱动负责管理，无需开发者初始化）	N/A
__IO hal_spi_state_t state	SPI运行状态（无需开发者初始化）	该参数的取值可以是下列值中的任意一个： • HAL_SPI_STATE_RESET（未初始化） • HAL_SPI_STATE_READY（已初始化且空闲） • HAL_SPI_STATE_BUSY（忙） • HAL_SPI_STATE_BUSY_TX（正在发送） • HAL_SPI_STATE_BUSY_RX（正在接收） • HAL_SPI_STATE_BUSY_TX_RX（正在收发） • HAL_SPI_STATE_ABORT（被中断） • HAL_SPI_STATE_ERROR（错误）
__IO uint32_t error_code	SPI错误码（无需开发者初始化）	该参数的取值可以是下列值中的任意一个： • HAL_SPI_ERROR_NONE（无错误） • HAL_SPI_ERROR_TIMEOUT（超时） • HAL_SPI_ERROR_TRANSFER（传输错误） • HAL_SPI_ERROR_DMA（DMA传输错误） • HAL_SPI_ERROR_INVALID_PARAM（非法参数）
uint32_t timeout	SPI超时时间（无需开发者初始化）	N/A
uint32_t retention[8]	保存SPI寄存器信息（驱动负责管理，无需开发者初始化）	N/A

2.20.4 SPI驱动API描述

SPI驱动的API主要如下：

表 2-341 SPI驱动的APIs

API类别	API名称	描述
初始化	hal_spi_init()	初始化SPI外设，配置时钟分频系数等参数。
	hal_spi_deinit()	反初始化SPI外设。
	hal_spi_msp_init()	初始化SPI外设所使用的GPIO引脚复用、NVIC中断、DMA通道。
	hal_spi_msp_deinit()	反初始化SPI外设所使用的GPIO引脚复用、NVIC中断、DMA通道。
IO操作	hal_spi_transmit()	发送数据，轮询方式。
	hal_spi_receive()	接收数据，轮询方式。
	hal_spi_transmit_receive()	收发数据，轮询方式。
	hal_spi_read_eeprom()	读取EEPROM，轮询方式。
	hal_spi_transmit_it()	发送数据，中断方式。
	hal_spi_receive_it()	接收数据，中断方式。
	hal_spi_transmit_receive_it()	收发数据，中断方式。
	hal_spi_read_eeprom_it()	读取EEPROM，中断方式。
	hal_spi_transmit_dma()	发送数据，DMA方式。
	hal_spi_receive_dma()	接收数据，DMA方式。
	hal_spi_transmit_receive_dma()	收发数据，DMA方式。
	hal_spi_read_eeprom_dma()	读取EEPROM，DMA方式。
	hal_spi_abort()	中止中断及DMA方式下的数据传输，轮询方式。
	hal_spi_abort_it()	中止中断及DMA方式下的数据传输，中断方式。
中断处理及回调函数	hal_spi_irq_handler()	中断处理函数。
	hal_spi_tx_cplt_callback()	发送完成中断回调函数。
	hal_spi_rx_cplt_callback()	接收完成中断回调函数。
	hal_spi_tx_rx_cplt_callback()	收发完成中断回调函数。
	hal_spi_error_callback()	错误中断回调函数。
	hal_spi_abort_cplt_callback()	中止完成中断回调函数。
状态及错误	hal_spi_get_state()	获取驱动运行状态。
	hal_spi_get_error()	获取错误码。
控制	hal_spi_set_timeout()	设置超时时间。
	hal_spi_set_tx_fifo_threshold()	设置TX FIFO阈值。

API类别	API名称	描述
	hal_spi_set_rx_fifo_threshold()	设置RX FIFO阈值。
	hal_spi_get_tx_fifo_threshold()	获取TX FIFO阈值。
	hal_spi_get_rx_fifo_threshold()	获取RX FIFO阈值。
睡眠相关	hal_spi_suspend_reg()	睡眠时挂起SPI配置相关的寄存器。
	hal_spi_resume_reg()	唤醒时恢复SPI配置相关的寄存器。

下面章节将对各API进行详细描述。

2.20.4.1 hal_spi_init

表 2-342 hal_spi_init接口

函数原型	hal_status_t hal_spi_init(spi_handle_t *p_spi)
功能说明	根据spi_init_t里的参数初始化SPI外设和初始化关联句柄。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	HAL状态。
备注	

2.20.4.2 hal_spi_deinit

表 2-343 hal_spi_deinit接口

函数原型	hal_status_t hal_spi_deinit(spi_handle_t *p_spi)
功能说明	反初始化SPI外设。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	HAL状态。
备注	

2.20.4.3 hal_spi_msp_init

表 2-344 hal_spi_msp_init接口

函数原型	void hal_spi_msp_init(spi_handle_t *p_spi)
功能说明	初始化SPI所使用的GPIO引脚复用、NVIC中断、DMA通道等配置。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成GPIO引脚复用、NVIC中断、DMA通道的初始化。

2.20.4.4 hal_spi_msp_deinit

表 2-345 hal_spi_msp_deinit接口

函数原型	void hal_spi_msp_deinit(spi_handle_t *p_spi)
功能说明	反初始化SPI所使用的GPIO引脚复用、NVIC中断、DMA通道等配置。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成GPIO引脚复用、NVIC中断、DMA通道的反初始化。

2.20.4.5 hal_spi_transmit

表 2-346 hal_spi_transmit接口

函数原型	hal_status_t hal_spi_transmit(spi_handle_t *p_spi, uint8_t *p_data, uint32_t length, uint32_t timeout)
功能说明	发送大量数据，轮询方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待发送数据的长度。 timout：超时时间。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_spi_get_error()获取具体的错误码。

2.20.4.6 hal_spi_receive

表 2-347 hal_spi_receive接口

函数原型	hal_status_t hal_spi_receive(spi_handle_t *p_spi, uint8_t *p_data, uint32_t length, uint32_t timeout)
功能说明	接收大量数据，轮询方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待接收数据的长度。 timout：超时时间。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_spi_get_error()获取具体的错误码。

2.20.4.7 hal_spi_transmit_receive

表 2-348 hal_spi_transmit_receive接口

函数原型	hal_status_t hal_spi_transmit_receive(spi_handle_t *p_spi, uint8_t *p_tx_data, uint8_t *p_rx_data, uint32_t length, uint32_t timeout)
功能说明	发送和接收大量数据，全双工，轮询方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_tx_data：指向发送数据缓冲区的指针。 p_rx_data：指向接收数据缓冲区的指针。 length：待发送和接收数据的长度。 timout：超时时间。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_spi_get_error()获取具体的错误码。

2.20.4.8 hal_spi_read_eeprom

表 2-349 hal_spi_read_eeprom接口

函数原型	hal_status_t hal_spi_read_eeprom(spi_handle_t *p_spi, uint8_t *p_tx_data, uint8_t *p_rx_data, uint32_t tx_number_data, uint32_t rx_number_data, uint32_t timeout)
功能说明	读取EEPROM中的数据，半双工，轮询方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_tx_data：指向发送数据缓冲区的指针。 p_rx_data：指向接收数据缓冲区的指针。 tx_number_data：待发送数据的长度。 rx_number_data：待接收数据的长度。 timout：超时时间。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_spi_get_error()获取具体的错误码。

2.20.4.9 hal_spi_transmit_it

表 2-350 hal_spi_transmit_it接口

函数原型	hal_status_t hal_spi_transmit_it(spi_handle_t *p_spi, uint8_t *p_data, uint32_t length)
功能说明	发送大量数据，中断方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待发送数据的长度。
返回值	HAL状态。

备注	• 发送完成时，回调函数hal_spi_tx_cplt_callback()会被调用。 • 发送过程中出现错误时，回调函数hal_spi_error_callback()会被调用，可在回调函数中调用hal_spi_get_error()获取具体的错误码。 • hal_spi_tx_cplt_callback()回调函数被调用前，请勿释放data指向的数据缓冲区的内存。 • 传输过程中，SPI中断处理无法及时响应，可能出现发送数据错误的情况。

2.20.4.10 hal_spi_receive_it

表 2-351 hal_spi_receive_it接口

函数原型	hal_status_t hal_spi_receive_it(spi_handle_t *p_spi, uint8_t *p_data, uint32_t length)
功能说明	接收大量数据，中断方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待接收数据的长度。
返回值	HAL状态
备注	• 接收完成时，回调函数hal_spi_rx_cplt_callback()会被调用。 • 接收过程中出现错误时，回调函数hal_spi_error_callback()会被调用，可在回调函数中调用hal_spi_get_error()获取具体的错误码。 • hal_spi_rx_cplt_callback()回调函数被调用前，请勿释放data指向的数据缓冲区的内存。 • 传输过程中，SPI中断处理无法及时响应，可能出现接收数据错误的情况。

2.20.4.11 hal_spi_transmit_receive_it

表 2-352 hal_spi_transmit_receive_it接口

函数原型	hal_status_t hal_spi_transmit_receive_it(spi_handle_t *p_spi, uint8_t *p_tx_data, uint8_t *p_rx_data, uint32_t length)
功能说明	发送和接收大量数据，全双工，中断方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息 p_tx_data：指向发送数据缓冲区的指针。 p_rx_data：指向接收数据缓冲区的指针。 length：待发送和接收数据的长度。
返回值	HAL状态。
备注	• 收发完成时，回调函数hal_spi_tx_rx_cplt_callback()会被调用。 • 收发过程中出现错误时，回调函数hal_spi_error_callback()会被调用，可在回调函数中调用hal_spi_get_error()获取具体的错误码。 • hal_spi_tx_rx_cplt_callback()回调函数被调用前，请勿释放data指向的数据缓冲区的内存。 • 传输过程中，SPI中断处理无法及时响应，可能出现收发数据错误的情况。

2.20.4.12 hal_spi_read_eeprom_it

表 2-353 hal_spi_read_eeprom_it接口

函数原型	hal_status_t hal_spi_read_eeprom_it(spi_handle_t *p_spi, uint8_t *p_tx_data, uint8_t *p_rx_data, uint32_t tx_number_data, uint32_t rx_number_data)
功能说明	读取EEPROM中的数据，半双工，中断方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_tx_data：指向发送数据缓冲区的指针。 p_rx_data：指向接收数据缓冲区的指针。 tx_number_data：待发送数据的长度。 rx_number_data：待接收数据的长度。
返回值	HAL状态。
备注	• 读取完成时，回调函数hal_spi_rx_cplt_callback()会被调用。 • 读取过程中出现错误时，回调函数hal_spi_error_callback()会被调用，可在回调函数中调用hal_spi_get_error()获取具体的错误码。 • hal_spi_rx_cplt_callback()回调函数被调用前，请勿释放data指向的数据缓冲区的内存。 • 传输过程中，SPI中断处理无法及时响应，可能出现读数据错误的情况。

2.20.4.13 hal_spi_transmit_dma

表 2-354 hal_spi_transmit_dma接口

函数原型	hal_status_t hal_spi_transmit_dma(spi_handle_t *p_spi, uint8_t *p_data, uint32_t length)
功能说明	发送大量数据，DMA方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待发送数据的长度。
返回值	HAL状态。
备注	• 发送完成时，回调函数hal_spi_tx_cplt_callback()会被调用。 • 发送过程中出现错误时，回调函数hal_spi_error_callback()会被调用，可在回调函数中调用hal_spi_get_error()获取具体的错误码。 • hal_spi_tx_cplt_callback()回调函数被调用前，请勿释放data指向的数据缓冲区的内存。

2.20.4.14 hal_spi_receive_dma

表 2-355 hal_spi_receive_dma接口

函数原型	hal_status_t hal_spi_receive_dma(spi_handle_t *p_spi, uint8_t *p_data, uint32_t length)
功能说明	接收大量数据，DMA方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_data：指向数据缓冲区的指针。 length：待接收数据的长度。

返回值	HAL状态。
备注	• 接收完成时，回调函数hal_spi_rx_cplt_callback()会被调用。 • 接收过程中出现错误时，回调函数hal_spi_error_callback()会被调用，可在回调函数中调用hal_spi_get_error()获取具体的错误码。 • hal_spi_rx_cplt_callback()回调函数被调用前，请勿释放data指向的数据缓冲区的内存。

2.20.4.15 hal_spi_transmit_receive_dma

表 2-356 hal_spi_transmit_receive_dma接口

函数原型	hal_status_t hal_spi_transmit_receive_dma(spi_handle_t *p_spi, uint8_t *p_tx_data, uint8_t *p_rx_data, uint32_t length)
功能说明	发送和接收大量数据，全双工，DMA方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_tx_data：指向发送数据缓冲区的指针。 p_rx_data：指向接收数据缓冲区的指针。 length：待发送和接收数据的长度。
返回值	HAL状态
备注	• 收发完成时，回调函数hal_spi_tx_rx_cplt_callback()会被调用。 • 收发过程中出现错误时，回调函数hal_spi_error_callback()会被调用，可在回调函数中调用hal_spi_get_error()获取具体的错误码。 • hal_spi_tx_rx_cplt_callback()回调函数被调用前，请勿释放data指向的数据缓冲区的内存。

2.20.4.16 hal_spi_read_eeprom_dma

表 2-357 hal_spi_read_eeprom_dma接口

函数原型	hal_status_t hal_spi_read_eeprom_dma(spi_handle_t *p_spi, uint8_t *p_tx_data, uint8_t *p_rx_data, uint32_t tx_number_data, uint32_t rx_number_data)
功能说明	读取EEPROM中的数据，半双工，DMA方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 p_tx_data：指向发送数据缓冲区的指针。 p_rx_data：指向接收数据缓冲区的指针。 tx_number_data：待发送数据的长度。 rx_number_data：待接收数据的长度。
返回值	HAL状态
备注	• 读取完成时，回调函数hal_spi_rx_cplt_callback()会被调用。 • 读取过程中出现错误时，回调函数hal_spi_error_callback()会被调用，可在回调函数中调用hal_spi_get_error()获取具体的错误码。 • hal_spi_rx_cplt_callback()回调函数被调用前，请勿释放data指向的数据缓冲区的内存。

2.20.4.17 hal_spi_abort

表 2-358 hal_spi_abort接口

函数原型	hal_status_t hal_spi_abort(spi_handle_t *p_spi)
功能说明	中止中断及DMA方式下的数据传输，轮询方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	HAL状态。
备注	该函数为轮询式函数，传输中止完成后退出函数。

2.20.4.18 hal_spi_abort_it

表 2-359 hal_spi_abort_it接口

函数原型	hal_status_t hal_spi_abort_it(spi_handle_t *p_spi)
功能说明	中止中断及DMA方式下的数据传输，中断方式。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	HAL状态。
备注	该函数为非轮询式函数，使能收发中断后，该函数立即返回。TX_ABRT中断触发后中止完成，此时hal_spi_abort_cplt_callback()将会被调用。

2.20.4.19 hal_spi_irq_handler

表 2-360 hal_spi_irq_handler接口

函数原型	void hal_spi_irq_handler(spi_handle_t *p_spi)
功能说明	处理SPI中断请求。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	无
备注	

2.20.4.20 hal_spi_tx_cplt_callback

表 2-361 hal_spi_tx_cplt_callback接口

函数原型	void hal_spi_tx_cplt_callback(spi_handle_t *p_spi)
功能说明	发送完成中断回调函数。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.20.4.21 hal_spi_rx_cplt_callback

表 2-362 hal_spi_rx_cplt_callback接口

函数原型	void hal_spi_rx_cplt_callback(spi_handle_t *p_spi)
功能说明	接收完成中断回调函数。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.20.4.22 hal_spi_tx_rx_cplt_callback

表 2-363 hal_spi_tx_rx_cplt_callback接口

函数原型	void hal_spi_tx_rx_cplt_callback(spi_handle_t *p_spi)
功能说明	收发完成中断回调函数。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.20.4.23 hal_spi_error_callback

表 2-364 hal_spi_error_callback接口

函数原型	void hal_spi_error_callback(spi_handle_t *p_spi)
功能说明	SPI错误回调函数。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.20.4.24 hal_spi_abort_cplt_callback

表 2-365 hal_spi_abort_cplt_callback接口

函数原型	void hal_spi_abort_cplt_callback(spi_handle_t *p_spi)
功能说明	SPI中止完成回调函数。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.20.4.25 hal_spi_get_state

表 2-366 hal_spi_get_state接口

函数原型	hal_spi_state_t hal_spi_get_state(spi_handle_t *p_spi)
功能说明	获取SPI运行状态。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	SPI运行状态，该参数的取值可以是下列值中的任意一个： • HAL_SPI_STATE_RESET（未初始化） • HAL_SPI_STATE_READY（已初始化且空闲） • HAL_SPI_STATE_BUSY（忙） • HAL_SPI_STATE_BUSY_TX（正在发送） • HAL_SPI_STATE_BUSY_RX（正在接收） • HAL_SPI_STATE_BUSY_TX_RX（正在收发） • HAL_SPI_STATE_ABORT（被中断） • HAL_SPI_STATE_ERROR（错误）
备注	

2.20.4.26 hal_spi_get_error

表 2-367 hal_spi_get_error接口

函数原型	uint32_t hal_spi_get_error(spi_handle_t *p_spi)
功能说明	获取SPI错误码。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	SPI错误码，该参数的取值可以是下列值中的任意一个： • HAL_SPI_ERROR_NONE（无错误） • HAL_SPI_ERROR_TIMEOUT（超时） • HAL_SPI_ERROR_TRANSFER（传输错误） • HAL_SPI_ERROR_DMA（DMA传输错误） • HAL_SPI_ERROR_INVALID_PARAM（非法参数）
备注	

2.20.4.27 hal_spi_set_timeout

表 2-368 hal_spi_set_timeout接口

函数原型	void hal_spi_set_timeout(spi_handle_t *p_spi, uint32_t timeout)
功能说明	设置SPI内部API的超时时间。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 timeout：超时时间（ms）。
返回值	无
备注	

2.20.4.28 hal_spi_set_tx_fifo_threshold

表 2-369 hal_spi_set_tx_fifo_threshold接口

函数原型	hal_status_t hal_spi_set_tx_fifo_threshold(spi_handle_t *p_spi, uint32_t threshold)
功能说明	设置TX FIFO阈值。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 threshold：TX FIFO阈值。
返回值	HAL状态。
备注	

2.20.4.29 hal_spi_set_rx_fifo_threshold

表 2-370 hal_spi_set_rx_fifo_threshold接口

函数原型	hal_status_t hal_spi_set_rx_fifo_threshold(spi_handle_t *p_spi, uint32_t threshold)
功能说明	设置RX FIFO阈值。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。 threshold：RX FIFO阈值。
返回值	HAL状态。
备注	

2.20.4.30 hal_spi_get_tx_fifo_threshold

表 2-371 hal_spi_get_tx_fifo_threshold接口

函数原型	uint32_t hal_spi_get_tx_fifo_threshold(spi_handle_t *p_spi)
功能说明	获取TX FIFO阈值。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	TX FIFO阈值（取值范围0 ～ 7，其中0表示TX FIFO为空，7表示TX FIFO差一个字节满）。
备注	

2.20.4.31 hal_spi_get_rx_fifo_threshold

表 2-372 hal_spi_get_rx_fifo_threshold接口

函数原型	uint32_t hal_spi_get_rx_fifo_threshold(spi_handle_t *p_spi)
功能说明	获取RX FIFO阈值。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	RX FIFO阈值（取值范围0 ～ 7，其中0表示RX FIFO中有一个字节，7表示RX FIFO满）。
备注	

2.20.4.32 hal_spi_suspend_reg

表 2-373 hal_spi_suspend_reg接口

函数原型	hal_status_t hal_spi_suspend_reg(spi_handle_t *p_spi)
功能说明	睡眠时挂起SPI配置相关的寄存器。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	HAL状态。
备注	

2.20.4.33 hal_spi_resume_reg

表 2-374 hal_spi_resume_reg接口

函数原型	hal_status_t hal_spi_resume_reg(spi_handle_t *p_spi)
功能说明	唤醒时恢复SPI配置相关的寄存器。
输入参数	p_spi：指向spi_handle_t结构体变量的指针，该结构体变量包含指定的SPI的配置信息。
返回值	HAL状态。
备注	

2.21 HAL TIMER通用驱动

2.21.1 TIMER驱动功能

TIMER外设的HAL驱动主要实现了以下功能：

- 支持32位计数初值配置。
- 支持轮询、中断两种计数方式。
- 支持停止轮询及中断方式下的计数操作。
- 支持计数完成的中断回调函数。
- 支持获取驱动运行状态。

2.21.2 如何使用TIMER驱动

TIMER HAL驱动的使用方法如下：

1. 声明一个timer_handle_t句柄结构，例如：timer_handle_t timer_handle。
2. 重写hal_timer_base_msp_init()以初始化TIMER底层资源：
   (1) 如果开发者使用中断方式的API函数hal_timer_base_start_it()计数，则需调用相关的NVIC接口来配置：
      - 调用hal_nvic_set_priority()配置TIMER中断优先级。

- 调用hal_nvic_enable_irq()使能TIMER的NVIC中断。

(2) 对timer_handle句柄init结构中的计数初值进行配置。

(3) 调用hal_timer_base_init()API初始化TIMER外设。

3. 如果用轮询方式的API函数hal_timer_base_start()计数，开发者可调用hal_timer_get_state()获取当前的驱动运行状态，以判断当前计数是否完成。

4. 如果用中断方式的API函数hal_timer_base_start_it()计数，开发者可重写hal_timer_period_elapsed_callback()中断回调函数，TIMER计数完成中断触发时，该回调函数会被自动调用。

2.21.3 TIMER驱动的结构体

2.21.3.1 timer_init_t

TIMER驱动的初始化结构体timer_init_t的定义如下：

表 2-375 timer_init_t结构体

数据域	域段描述	取值
uint32_t auto_reload	自动加载的计数初值。	0x0000_0000 ～ 0xFFFF_FFFF

2.21.3.2 timer_handle_t

TIMER驱动的句柄结构体timer_handle_t的定义如下：

表 2-376 timer_handle_t结构体

数据域	域段描述	取值
timer_regs_t *p_instance	TIMER外设实例。	该参数的取值可以是下列值中的任意一个： • TIMER0 • TIMER1
timer_init_t init	初始化结构体。	参考timer_init_t结构体。
__IO hal_lock_t lock	TIMER锁（无需开发者初始化）。	N/A
__IO hal_timer_state_t state	TIMER运行状态（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_TIMER_STATE_RESET • HAL_TIMER_STATE_READY • HAL_TIMER_STATE_BUSY • HAL_TIMER_STATE_ERROR

2.21.4 TIMER驱动API描述

TIMER驱动的API主要如下：

表 2-377 TIM驱动的APIs

API类别	API名称	描述
初始化	hal_timer_base_init()	初始化TIMER外设，配置计数初值等参数。
	hal_timer_base_deinit()	反初始化TIMER外设。
	hal_timer_base_msp_init()	初始化TIMER外设所使用的NVIC中断。
	hal_timer_base_msp_deinit()	反初始化TIMER外设所使用的NVIC中断。
IO操作	hal_timer_base_start()	启动计数，轮询方式。
	hal_timer_base_stop()	停止计数，轮询方式。
	hal_timer_base_start_it()	启动计数，中断方式。
	hal_timer_base_stop_it()	停止计数，中断方式。
控制	hal_timer_set_config()	配置计数器参数。
中断处理及回调函数	hal_timer_irq_handler()	中断处理函数。
	hal_timer_period_elapsed_callback()	计数完成的中断回调函数。
状态及错误	hal_timer_get_state()	获取驱动运行状态。

下面章节将对各API进行详细描述。

2.21.4.1 hal_timer_base_init

表 2-378 hal_timer_base_init接口

函数原型	hal_status_t hal_timer_base_init(timer_handle_t *p_timer)
功能说明	根据timer_init_t中的指定参数初始化TIMER时间基单元，并初始化相关的句柄。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	HAL状态。
备注	

2.21.4.2 hal_timer_base_deinit

表 2-379 hal_timer_base_deinit接口

函数原型	hal_status_t hal_timer_base_deinit(timer_handle_t *p_timer)
功能说明	反初始化TIMER外设。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	HAL状态。
备注	

2.21.4.3 hal_timer_base_msp_init

表 2-380 hal_timer_base_msp_init接口

函数原型	void hal_timer_base_msp_init(timer_handle_t *p_timer)
功能说明	初始化TIMER所使用的NVIC中断配置。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成NVIC中断的初始化。

2.21.4.4 hal_timer_base_msp_deinit

表 2-381 hal_timer_base_msp_deinit接口

函数原型	void hal_timer_base_msp_deinit(timer_handle_t *p_timer)
功能说明	反初始化TIMER的NVIC中断配置。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成NVIC中断的反初始化。

2.21.4.5 hal_timer_base_start

表 2-382 hal_timer_base_start接口

函数原型	hal_status_t hal_timer_base_start(timer_handle_t *p_timer)
功能说明	使能TIMER外设，开始计数，轮询方式。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	HAL状态。
备注	该API未使能TIMER中断，开发者需调用hal_timer_get_state()获取计数状态。

2.21.4.6 hal_timer_base_stop

表 2-383 hal_timer_base_stop接口

函数原型	hal_status_t hal_timer_base_stop(timer_handle_t *p_timer)
功能说明	禁用TIMER外设，停止轮询方式下的计数。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	HAL状态。

备注	该API未禁用TIMER中断，可与hal_timer_base_start()配合使用。

2.21.4.7 hal_timer_base_start_it

表 2-384 hal_timer_base_start_it接口

函数原型	hal_status_t hal_timer_base_start_it(timer_handle_t *p_timer)
功能说明	使能TIMER外设，开始计数，中断方式。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	HAL状态。
备注	该API会使能TIMER中断，计数完成时会调用回调函数hal_timer_period_elapsed_callback()。

2.21.4.8 hal_timer_base_stop_it

表 2-385 hal_timer_base_stop_it接口

函数原型	hal_status_t hal_timer_base_stop_it(timer_handle_t *p_timer)
功能说明	禁用TIMER外设，停止中断方式下的计数。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	HAL状态。
备注	该API禁止TIMER中断，可与hal_timer_base_start_it()配合使用。

2.21.4.9 hal_timer_set_config

表 2-386 hal_timer_set_config接口

函数原型	hal_status_t hal_timer_set_config(timer_handle_t *p_timer, timer_init_t *p_structure)
功能说明	配置TIMER计数器参数。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。 p_structure：指向timer_init_t结构体变量的指针，该结构体变量包含指定的TIMER计数器参数。
返回值	HAL状态。
备注	该API适合在初始化TIMER之后更改计数器配置。

2.21.4.10 hal_timer_irq_handler

表 2-387 hal_timer_irq_handler接口

函数原型	void hal_timer_irq_handler(timer_handle_t *p_timer)
功能说明	TIMER中断处理函数。

输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	无
备注	

2.21.4.11 hal_timer_period_elapsed_callback

表 2-388 hal_timer_period_elapsed_callback接口

函数原型	void hal_timer_period_elapsed_callback(timer_handle_t *p_timer)
功能说明	TIMER外设计数完成的中断回调函数。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.21.4.12 hal_timer_get_state

表 2-389 hal_timer_get_state接口

函数原型	hal_timer_state_t hal_tim_get_state(timer_handle_t *p_timer)
功能说明	获取TIMER运行状态。
输入参数	p_timer：指向timer_handle_t结构体变量的指针，该结构体变量包含指定的TIMER模块的配置信息。
返回值	TIM运行状态，该参数的取值可以是下面中的任意一个值： • HAL_TIMER_STATE_RESET • HAL_TIMER_STATE_READY • HAL_TIMER_STATE_BUSY • HAL_TIMER_STATE_ERROR
备注	

2.22 HAL Calendar通用驱动

2.22.1 Calendar驱动功能

Calendar外设的HAL驱动主要实现了以下功能：

- 32位计数器，时钟源为32.768 kHz的RTC时钟。
- 支持多种方式预分频，包括：1、32、64、128、256分频。
- 支持闹铃功能。
- 支持回环中断方式。

- 支持设置年月日分秒和获取当前时间。

2.22.2 如何使用Calendar驱动

Calendar HAL驱动的使用方法如下：

1. 声明一个calendar_handle_t句柄结构，例如：calendar_handle_t calendar_handle。
2. 调用hal_calendar_init()API初始化Calendar外设，配置calendar_handle句柄init结构中的时间初值，并以回环中断方式启动Calendar。
3. 调用hal_calendar_init_time()函数，将根据calendar_time_t信息更新Calendar时间基准。
4. 调用hal_calendar_get_time()函数获取当前的Calendar时间。
5. 如果调用hal_calendar_set_alarm()函数配置闹铃，开发者可重写hal_calendar_alarm_callback()中断回调函数。Calendar计数完成触发闹铃时，该回调函数会被自动调用。
6. 如果调用hal_calendar_set_tick()函数配置闹铃的毫秒计数，开发者可重写hal_calendar_tick_callback()中断回调函数。Calendar计数完成触发闹铃时，该回调函数会被自动调用。

2.22.3 Calendar驱动的结构体

2.22.3.1 calendar_time_t

Calendar驱动的时间结构体calendar_time_t的定义如下：

表 2-390 calendar_time_t结构体

数据域	域段描述	取值
uint8_t sec	秒	0～59
uint8_t min	分	0～59
uint8_t hour	小时	0～23
uint8_t data	日	1～31
uint8_t mon	月	1～12
uint8_t year	年	0～99
uint8_t week	星期	0～6
uint16_t ms	毫秒	0～999

2.22.3.2 calendar_alarm_t

Calendar驱动的句柄结构体calendar_alarm_t的定义如下：

表 2-391 calendar_alarm_t结构体

数据域	域段描述	取值
uint8_t min	Calendar闹铃分钟。	0 ～ 59
uint8_t hour	Calendar闹铃小时	0 ～ 23
uint8_t alarm_sel	Calendar闹铃周期。	该参数的取值可以是下列值中的任意一个： • CALENDAR_ALARM_SEL_DATE • CALENDAR_ALARM_SEL_WEEKDAY
uint8_t alarm_data_week_mask	Calendar闹铃日期。	当alarm_sel配置 为CALENDAR_ALARM_SEL_DATE时，该参数的取值范围是1 ～ 31。 当alarm_sel配置 为CALENDAR_ALARM_SEL_WEEKDAY时，该参数的取值可以是下列值中的任意一个： • CALENDAR_ALARM_WEEKDAY_SUN • CALENDAR_ALARM_WEEKDAY_MON • CALENDAR_ALARM_WEEKDAY_TUE • CALENDAR_ALARM_WEEKDAY_WED • CALENDAR_ALARM_WEEKDAY_THU • CALENDAR_ALARM_WEEKDAY_FRI • CALENDAR_ALARM_WEEKDAY_SAT

2.22.3.3 calendar_handle_t

Calendar驱动的句柄结构体calendar_handle_t的定义如下：

表 2-392 calendar_handle_t结构体

数据域	域段描述	取值
calendar_time_t time_init	Calendar时间结构体。	参考calendar_time_t结构体。
calendar_alarm_t alarm	Calendar闹铃结构体。	参考calendar_alarm_t结构体。
__IO hal_lock_t lock	Calendar锁（无需开发者初始化）	N/A
uint32_t prev_ms	Calendar毫秒计数时间累加值	N/A
uint32_t interval	闹钟的毫秒计数	5 ～ 3600000(ms)
uint8_t mode	闹钟模式（无需开发者初始化）	N/A
uint8_t sec	日期闹钟中用于保存当前的秒数（无需开发者初始化）	N/A
uint16_t ms	日期闹钟中用于保存当前的毫秒数（无需开发者初始化）	N/A

2.22.4 Calendar驱动API描述

Calendar驱动的API主要如下：

表 2-393 Calendar驱动的APIs

API类别	API名称	描述
初始化	hal_calendar_init()	初始化Calendar外设，以回环中断方式启动Calendar。
	hal_calendar_deinit()	反初始化Calendar外设。
IO操作	hal_calendar_init_time()	初始化Calendar当前时间。
	hal_calendar_get_time()	获取Calendar当前时间。
	hal_calendar_set_alarm()	设置Calendar闹铃时间，并启动闹铃功能。
	hal_calendar_set_tick()	设置calendar闹铃的毫秒计数，并启动闹铃功能
	hal_calendar_disable_event()	停止Calendar闹铃功能。
中断处理及回调函数	hal_calendar_irq_handler()	中断处理函数。
	hal_calendar_alarm_callback()	闹铃的中断回调函数。
	hal_calendar_tick_callback()	闹铃毫秒计数的中断回调函数

下面章节将对各API进行详细描述。

2.22.4.1 hal_calendar_init

表 2-394 hal_calendar_init接口

函数原型	hal_status_t hal_calendar_init(calendar_handle_t *p_calendar)
功能说明	初始化Calendar外设，以回环中断方式启动Calendar。
输入参数	p_calendar：指向calendar_handle_t结构体变量的指针，该结构体变量包含指定的Calendar模块的配置信息。
返回值	HAL状态。
备注	

2.22.4.2 hal_calendar_deinit

表 2-395 hal_calendar_deinit接口

函数原型	hal_status_t hal_calendar_deinit(calendar_handle_t *p_calendar)
功能说明	反初始化Calendar外设。
输入参数	p_calendar：指向calendar_handle_t结构体变量的指针，该结构体变量。包含指定的Calendar模块的配置信息。
返回值	HAL状态。
备注	

2.22.4.3 hal_calendar_init_time

表 2-396 hal_calendar_init_time接口

函数原型	hal_status_t hal_calendar_init_time(calendar_handle_t *p_calendar, calendar_time_t *p_time)
功能说明	初始化Calendar当前时间。
输入参数	p_calendar：指向calendar_handle_t结构体变量的指针，该结构体变量包含指定的Calendar模块的配置信息。 p_time：指向calendar_time_t结构体变量的指针，该结构体变量包含指定的Calendar的时间配置信息。
返回值	HAL状态
备注	调用该接口时，其中year的最小值为10，即2010年。

2.22.4.4 hal_calendar_get_time

表 2-397 hal_calendar_get_time接口

函数原型	hal_status_t hal_calendar_get_time(calendar_handle_t *p_calendar, calendar_time_t *p_time)
功能说明	获取Calendar当前时间。
输入参数	p_calendar：指向calendar_handle_t结构体变量的指针，该结构体变量包含指定的Calendar模块的配置信息。 p_time：指向calendar_time_t结构体变量的指针，该结构体变量包含当前Calendar的时间信息。
返回值	HAL状态。
备注	

2.22.4.5 hal_calendar_set_alarm

表 2-398 hal_calendar_set_alarm接口

函数原型	hal_status_t hal_calendar_set_alarm(calendar_handle_t *p_calendar, calendar_alarm_t *p_alarm)
功能说明	根据p_alarm配置Calendar闹铃时间，并启动闹铃功能。
输入参数	p_calendar：指向calendar_handle_t结构体变量的指针，该结构体变量包含指定的Calendar模块的配置信息。 p_alarm：指向calendar_alarm_t结构体变量的指针，该结构体变量包含指定的闹铃时间的配置信息。
返回值	HAL状态。
备注	该API将使能calendar中断，计时到闹铃时间时回调函数hal_calendar_alarm _callback()会被调用。

2.22.4.6 hal_calendar_set_tick

表 2-399 hal_calendar_set_tick接口

函数原型	hal_status_t hal_hal_calendar_set_tick(calendar_handle_t *p_calendar, uint32_t interval)
功能说明	设置闹钟的毫秒计数。
输入参数	p_calendar：指向calendar_handle_t结构体变量的指针，该结构体变量包含指定的Calendar模块的配置信息。
返回值	HAL状态。
备注	最小值是5 ms，最大值为3600*1000 ms。

2.22.4.7 hal_calendar_disable_event

表 2-400 hal_calendar_disable_event接口

函数原型	hal_status_t hal_calendar_disable_event(calendar_handle_t *p_calendar, uint32_t disable_mode)
功能说明	停止Calendar闹铃功能。
输入参数	p_calendar：指向calendar_handle_t结构体变量的指针，该结构体变量包含指定的Calendar模块的配置信息。 disable_mode:选择关闭哪类闹钟，参数可以配置为CALENDAR_ALARM_DISABLE_DATE、CALENDAR_ALARM_DISABLE_TICK、CALENDAR_ALARM_DISABLE_ALL。
返回值	HAL状态。
备注	该API禁止Calendar闹铃功能，可与hal_calendar_set_alarm()配合使用。

2.22.4.8 hal_calendar_irq_handler

表 2-401 hal_calendar_irq_handler接口

函数原型	void hal_calendar_irq_handler(calendar_handle_t *p_calendar)
功能说明	Calendar中断处理函数。
输入参数	p_calendar：指向calendar_handle_t结构体变量的指针，该结构体变量包含指定的Calendar模块的配置信息。
返回值	无
备注	

2.22.4.9 hal_calendar_alarm_callback

表 2-402 hal_calendar_alarm_callback接口

函数原型	void hal_calendar_alarm_callback(calendar_handle_t *p_calendar)
功能说明	闹铃的中断回调函数。

输入参数	p_calendar：指向calendar_handle_t结构体变量的指针，该结构体变量包含指定的Calendar模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.22.4.10 hal_calendar_tick_callback

表 2-403 hal_calendar_tick_callback接口

函数原型	void hal_calendar_tick_callback(calendar_handle_t *p_calendar)
功能说明	毫秒闹铃的中断回调函数。
输入参数	p_calendar：指向calendar_handle_t结构体变量的指针，该结构体变量包含指定的Calendar模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.23 HAL UART通用驱动

2.23.1 UART驱动功能

UART（Universal Asynchronous Receiver/Transmitter）外设的HAL驱动主要实现了以下功能：

- 支持波特率9600～921600。
- 支持数据位（5、6、7、8）、奇偶校验位（无、奇校验、偶校验、强制为0、强制为1）、停止位（1、1.5、2）。
- 支持自动流控。
- 支持轮询、中断、DMA三种数据读写方式。
- 支持中止中断及DMA方式下的发送、接收、发送及接收操作。
- 支持DMA方式下的收发暂停、恢复、停止。
- 支持发送完成、接收完成、错误、收发中止完成、发送中止完成、接收中止完成的中断回调函数。
- 支持获取驱动的运行状态及错误码。

2.23.2 如何使用UART驱动

UART HAL驱动使用方法如下：

1. 声明一个uart_handle_t句柄结构，例如：uart_handle_t uart_handle。
2. 重写hal_uart_msp_init()以初始化UART底层资源：
   (1) 配置UART引脚：调用hal_gpio_init()配置GPIO模式为GPIO_MODE_MUX，并将相应的GPIO的复用功能配置为UART。

(2) 如果开发者要使用中断流程（hal_uart_transmit_it()和hal_uart_receive_it()APIs），则需通过调用相关的NVIC接口来配置：

- 调用hal_nvic_set_priority()配置UART中断优先级。
- 调用hal_nvic_enable_irq()使能UART的NVIC中断。

(3) 如果开发者要使用DMA流程（hal_uart_transmit_DMA()和hal_uart_receive_DMA()APIs），则需要配置DMA：

- 为TX/RX通道声明DMA句柄结构，例如：dma_handle_t htxdma。
- 使用所需的TX/RX参数配置声明的DMA句柄结构。
- 配置DMA TX/RX信道。
- 将初始的DMA句柄关联到UART DMA TX/RX句柄。
- 在DMA TX/RX信道上配置优先级和使能传输完成中断。

3. 配置uart_handle句柄init结构中的波特率、数据比特位、停止位、奇偶校验位、硬件流控和模式（接收/发送）。
4. 调用hal_uart_init()API初始化UART寄存器。

2.23.3 UART驱动的结构体

2.23.3.1 uart_init_t

UART驱动的初始化结构体uart_init_t的定义如下：

表 2-404 uart_init_t结构体

数据域	域段描述	取值
uint32_t baud_rate	波特率	9600 ～ 921600
uint32_t data_bits	数据位	该参数的取值可以是下列值中的任意一个： • UART_DATABITS_5B（5位） • UART_DATABITS_6B（6位） • UART_DATABITS_7B（7位） • UART_DATABITS_8B（8位）
uint32_t stop_bits	停止位	该参数的取值可以是下列值中的任意一个： • UART_STOPBITS_1（1位） • UART_STOPBITS_1_5（1.5位） • UART_STOPBITS_2（2位） UART_STOPBITS_1_5仅在数据位为UART_DATABITS_5时可用，UART_STOPBITS_2在数据位为UART_DATABITS_5时不可用

数据域	域段描述	取值
uint32_t parity	校验位	该参数的取值可以是下列值中的任意一个： UART_PARITY_NONE（无校验） UART_PARITY_EVEN（偶校验） UART_PARITY_ODD（奇校验） UART_PARITY_SP0（校验位始终为0） UART_PARITY_SP1（校验位始终为1）
uint32_t hw_flow_ctrl	硬件流控	该参数的取值可以是下列值中的任意一个： • UART_HWCONTROL_NONE（无流控） • UART_HWCONTROL_RTS_CTS（自动流控）
uint32_t rx_timeoout_mode	接收超时	该参数的取值可以是下列值中的任意一个： • UART_RECEIVER_TIMEOUT_DISABLE（停止接收超时） • UART_RECEIVER_TIMEOUT_ENABLE（开启接收超时）

2.23.3.2 uart_handle_t

UART驱动的句柄结构体uart_handle_t的定义如下：

表 2-405 uart_handle_t结构体

数据域	域段描述	取值
uart_regs_t *p_instance	UART外设实例	该参数的取值可以是下列值中的任意一个： • UART0 • UART1
uart_init_t init	初始化结构体	参考uart_init_t结构体。
uint8_t *p_tx_buffer	指向数据发送缓冲区的指针（驱动负责管理，无需开发者初始化）	N/A
__IO uint16_t tx_xfer_size	数据发送长度（驱动负责管理，无需开发者初始化）	N/A
__IO uint16_t tx_xfer_count	数据发送计数（驱动负责管理，无需开发者初始化）	N/A
uint8_t *p_rx_buffer	指向数据接收缓冲区的指针（驱动负责管理，无需开发者初始化）	N/A
__IO uint16_t rx_xfer_size	数据接收长度（驱动负责管理，无需开发者初始化）	N/A
__IO uint16_t rx_xfer_count	数据接收计数（驱动负责管理，无需开发者初始化）	N/A

数据域	域段描述	取值
dma_handle_t *p_dmatx	指向数据发送通道的DMA句柄的指针（驱动负责管理，无需开发者初始化）	N/A
dma_handle_t *p_dmarx	指向数据接收通道的DMA句柄的指针	接收通道的DMA句柄dma_handle_t结构体
__IO hal_lock_t lock	UART锁（驱动负责管理，无需开发者初始化）	N/A
__IO hal_uart_state_t tx_state	UART运行状态（无需开发者初始化）	该参数的取值可以是下列值中的任意一个： • HAL_UART_STATE_RESET（未初始化） • HAL_UART_STATE_READY（已初始化且空闲） • HAL_UART_STATE_BUSY（忙） • HAL_UART_STATE_BUSY_TX（正在发送） • HAL_UART_STATE_BUSY_RX （正在接收） • HAL_UART_STATE_TIMEOUT（超时） • HAL_UART_STATE_ERROR（错误）
__IO hal_uart_state_t rx_state	UART接收运行状态（无需开发者初始化）	该参数的取值可以是下列值中的任意一个： • HAL_UART_STATE_RESET（未初始化） • HAL_UART_STATE_READY（已初始化且空闲） • HAL_UART_STATE_BUSY（忙） • HAL_UART_STATE_BUSY_TX（正在发送） • HAL_UART_STATE_BUSY_RX （正在接收） • HAL_UART_STATE_TIMEOUT（超时） • HAL_UART_STATE_ERROR（错误）
__IO hal_uart_mode_t mode	UART工作模式（驱动负责管理，无需开发者初始化）	N/A
__IO uint32_t error_code	UART错误码（无需开发者初始化）	该参数的取值可以是下列值中的任意一个： • HAL_UART_ERROR_NONE（无错误） • HAL_UART_ERROR_PE（校验位错误） • HAL_UART_ERROR_FE（帧错误） • HAL_UART_ERROR_OE（溢出错误） • HAL_UART_ERROR_BI（Line Break错误） • HAL_UART_ERROR_DMA（DMA传输错误） • HAL_UART_ERROR_BUSY（忙）
uint32_t retention[8]	保存UART寄存器信息（驱动负责管理，无需开发者初始化）	N/A

2.23.4 UART驱动API描述

UART驱动的API主要如下：

表 2-406 UART驱动的APIs

API类别	API名称	描述
初始化	hal_uart_init()	初始化UART外设，配置时钟分频系数等参数。
	hal_uart_deinit()	反初始化UART外设。
	hal_uart_msp_init()	初始化UART外设所使用的GPIO引脚复用、NVIC中断、DMA通道。
	hal_uart_msp_deinit()	反初始化UART外设所使用的GPIO引脚复用、NVIC中断、DMA通道。
IO操作	hal_uart_transmit()	发送数据，轮询方式。
	hal_uart_receive()	接收数据，轮询方式。
	hal_uart_transmit_it()	发送数据，中断方式。
	hal_uart_receive_it()	接收数据，中断方式。
	hal_uart_transmit_dma()	发送数据，DMA方式。
	hal_uart_receive_dma()	接收数据，DMA方式。
	hal_uart_dma_pause()	暂停DMA传输。
	hal_uart_dma_resume()	恢复DMA传输。
	hal_uart_dma_stop()	停止DMA传输。
	hal_uart_abort()	中止中断及DMA方式下的数据收发，轮询方式。
	hal_uart_abort_transmit()	中止中断及DMA方式下的数据发送，轮询方式。
	hal_uart_abort_receive()	中止中断及DMA方式下的数据接收，轮询方式。
	hal_uart_abort_it()	中止中断及DMA方式下的数据收发，中断方式。
	hal_uart_abort_transmit_it()	中止中断及DMA方式下的数据发送，中断方式。
	hal_uart_abort_receive_it()	中止中断及DMA方式下的数据接收，中断方式。
中断处理及回调函数	hal_uart_irq_handler()	中断处理函数。
	hal_uart_tx_cplt_callback()	发送完成中断回调函数。
	hal_uart_rx_cplt_callback()	接收完成中断回调函数。
	hal_uart_error_callback()	错误中断回调函数。
	hal_uart_abort_cplt_callback()	收发中止完成中断回调函数。
	hal_uart_abort_transmit_cplt_callback()	发送中止完成中断回调函数。
	hal_uart_abort_receive_cplt_callback()	接收中止完成中断回调函数。
状态及错误	hal_uart_get_state()	获取驱动运行状态。
	hal_uart_get_error()	获取错误码。
睡眠相关	hal_uart_suspend_reg()	睡眠之前挂起和UART配置相关的寄存器。
	hal_uart_resume_reg()	唤醒时恢复和UART配置相关的寄存器。

下面章节将对各API进行详细描述。

2.23.4.1 hal_uart_init

表 2-407 hal_uart_init接口

函数原型	hal_status_t hal_uart_init(uart_handle_t *p_uart)
功能说明	根据uart_init_t中的指定参数初始化UART模式，并初始化相关句柄。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。
备注	

2.23.4.2 hal_uart_deinit

表 2-408 hal_uart_deinit接口

函数原型	hal_status_t hal_uart_deinit(uart_handle_t *p_uart)
功能说明	反初始化UART外设。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。
备注	

2.23.4.3 hal_uart_msp_init

表 2-409 hal_uart_msp_init接口

函数原型	void hal_uart_msp_init(uart_handle_t *p_uart)
功能说明	初始化UART所使用的GPIO引脚复用、NVIC中断、DMA通道等配置。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	无
备注	该API为weak类型的空函数，开发者需要重写该API完成GPIO引脚复用、NVIC中断、DMA通道的初始化。

2.23.4.4 hal_uart_msp_deinit

表 2-410 hal_uart_msp_deinit接口

函数原型	void hal_uart_msp_deinit(uart_handle_t *p_uart)
功能说明	反初始化UART所使用的GPIO引脚复用、NVIC中断、DMA通道等配置。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	无
备注	该API为weak类型的空函数，开发者需重写该API完成GPIO引脚复用、NVIC中断、DMA通道的反初始化。

2.23.4.5 hal_uart_transmit

表 2-411 hal_uart_transmit接口

函数原型	hal_status_t hal_uart_transmit(uart_handle_t *p_uart, uint8_t *p_data, uint16_t size, uint32_t timeout)
功能说明	发送大量数据，轮询方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。 p_data：指向数据缓冲区的指针。 size：待发送数据的长度。 timout：超时时间。
返回值	HAL状态。
备注	

2.23.4.6 hal_uart_receive

表 2-412 hal_uart_receive接口

函数原型	hal_status_t hal_uart_receive(uart_handle_t *p_uart, uint8_t *p_data, uint16_t size, uint32_t timeout)
功能说明	接收大量数据，轮询方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。 p_data：指向数据缓冲区的指针。 size：待接收数据的长度。 timout：超时时间。
返回值	HAL状态。
备注	

2.23.4.7 hal_uart_transmit_it

表 2-413 hal_uart_transmit_it接口

函数原型	hal_status_t hal_uart_transmit_it(uart_handle_t *p_uart, uint8_t *p_data, uint16_t size)
功能说明	发送大量数据，中断方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。 p_data：指向数据缓冲区的指针。 size：待发送数据的长度。
返回值	HAL状态。
备注	

2.23.4.8 hal_uart_receive_it

表 2-414 hal_uart_receive_it接口

函数原型	hal_status_t hal_uart_receive_it(uart_handle_t *p_uart, uint8_t *p_data, uint16_t size)
功能说明	接收大量数据，中断方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。 p_data：指向数据缓冲区的指针。 size：待接收数据的长度。
返回值	HAL状态。
备注	

2.23.4.9 hal_uart_transmit_dma

表 2-415 hal_uart_transmit_dma接口

函数原型	hal_status_t hal_uart_transmit_dma(uart_handle_t *p_uart, uint8_t *p_data, uint16_t size)
功能说明	发送大量数据，DMA方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。 p_data：指向数据缓冲区的指针。 size：待发送数据的长度。
返回值	HAL状态。
备注	

2.23.4.10 hal_uart_receive_dma

表 2-416 hal_uart_receive_dma接口

函数原型	hal_status_t hal_uart_receive_dma(uart_handle_t *p_uart, uint8_t *p_data, uint16_t size)
功能说明	接收大量数据，DMA方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。 p_data：指向数据缓冲区的指针。 size：待接收数据的长度。
返回值	HAL状态。
备注	

2.23.4.11 hal_uart_dma_pause

表 2-417 hal_uart_dma_pause接口

函数原型	hal_status_t hal_uart_dma_pause(uart_handle_t *p_uart)
功能说明	暂停UART DMA传输。

输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。
备注	

2.23.4.12 hal_uart_dma_resume

表 2-418 hal_uart_dma_resume接口

函数原型	hal_status_t hal_uart_dma_resume(uart_handle_t *p_uart)
功能说明	恢复UART DMA传输。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。
备注	

2.23.4.13 hal_uart_dma_stop

表 2-419 hal_uart_dma_stop接口

函数原型	hal_status_t hal_uart_dma_stop(uart_handle_t *p_uart)
功能说明	停止UART DMA传输。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。
备注	

2.23.4.14 hal_uart_abort

表 2-420 hal_uart_abort接口

函数原型	hal_status_t hal_uart_abort(uart_handle_t *p_uart)
功能说明	中止中断及DMA方式下的数据收发，轮询方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。
备注	该函数可用于中止中断方式或DMA方式的数据收发，该函数执行如下操作： • 禁用TX中断和RX中断 • 中止DMA传输 • 设置p_uart中的state为HAL_UART_STATE_READY

2.23.4.15 hal_uart_abort_transmit

表 2-421 hal_uart_abort_transmit接口

函数原型	hal_status_t hal_uart_abort_transmit(uart_handle_t *p_uart)
功能说明	中止中断及DMA方式下的数据发送，轮询方式。

输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。
备注	该函数可用于中止中断方式或DMA方式的数据发送，该函数执行如下操作： • 禁用TX中断 • 中止DMA传输 • 设置p_uart中的state为HAL_UART_STATE_READY

2.23.4.16 hal_uart_abort_receive

表 2-422 hal_uart_abort_receive接口

函数原型	hal_status_t hal_uart_abort_receive(uart_handle_t *p_uart)
功能说明	中止中断及DMA方式下的数据接收，轮询方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。
备注	该函数可用于中止中断方式或DMA方式的数据接收，该函数执行如下操作： • 禁用RX中断 • 中止DMA传输 • 设置p_uart中的state为HAL_UART_STATE_READY

2.23.4.17 hal_uart_abort_it

表 2-423 hal_uart_abort_it接口

函数原型	hal_status_t hal_uart_abort_it(uart_handle_t *p_uart)
功能说明	中止中断及DMA方式下的数据收发，中断方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。
备注	该函数可用于中止中断方式或DMA方式的数据收发，该函数执行如下操作： • 禁用TX中断和RX中断； • 中止DMA传输； • 设置p_uart中的state为HAL_UART_STATE_READY 中止完成后，hal_uart_abort_cplt_callback()会被调用。

2.23.4.18 hal_uart_abort_transmit_it

表 2-424 hal_uart_abort_transmit_it接口

函数原型	hal_status_t hal_uart_abort_transmit_it(uart_handle_t *p_uart)
功能说明	中止中断及DMA方式下的数据发送，中断方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。

备注	该函数可用于中止中断方式或DMA方式的数据发送，该函数执行如下操作： • 禁用TX中断 • 中止DMA传输 • 设置p_uart中的state为HAL_UART_STATE_READY 中止完成后，hal_uart_abort_transmit_cplt_callback()会被调用。

2.23.4.19 hal_uart_abort_receive_it

表 2-425 hal_uart_abort_receive_it接口

函数原型	hal_status_t hal_uart_abort_receive_it(uart_handle_t *p_uart)
功能说明	中止中断及DMA方式下的数据接收，中断方式。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。
备注	该函数可用于中止中断方式或DMA方式的数据接收，该函数执行如下操作： • 禁用RX中断 • 中止DMA传输 • 设置p_uart中的state为HAL_UART_STATE_READY 中止完成后，hal_uart_abort_receive_cplt_callback()会被调用。

2.23.4.20 hal_uart_irq_handler

表 2-426 hal_uart_irq_handler接口

函数原型	void hal_uart_irq_handler(uart_handle_t *p_uart)
功能说明	处理UART中断请求。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	无
备注	

2.23.4.21 hal_uart_tx_cplt_callback

表 2-427 hal_uart_tx_cplt_callback接口

函数原型	void hal_uart_tx_cplt_callback(uart_handle_t *p_uart)
功能说明	发送完成中断回调函数。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.23.4.22 hal_uart_rx_cplt_callback

表 2-428 hal_uart_rx_cplt_callback接口

函数原型	void hal_uart_rx_cplt_callback(uart_handle_t *p_uart)
功能说明	接收完成中断回调函数。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.23.4.23 hal_uart_error_callback

表 2-429 hal_uart_error_callback接口

函数原型	void hal_uart_error_callback(uart_handle_t *p_uart)
功能说明	UART错误中断回调函数。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.23.4.24 hal_uart_abort_cplt_callback

表 2-430 hal_uart_abort_cplt_callback接口

函数原型	void hal_uart_abort_cplt_callback(uart_handle_t *p_uart)
功能说明	UART中止完成中断回调函数。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.23.4.25 hal_uart_abort_tx_cplt_callback

表 2-431 hal_uart_abort_tx_cplt_callback接口

函数原型	void hal_uart_abort_tx_cplt_callback(uart_handle_t *p_uart)
功能说明	UART中止发送完成中断回调函数。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.23.4.26 hal_uart_abort_rx_cplt_callback

表 2-432 hal_uart_abort_rx_cplt_callback接口

函数原型	void hal_uart_abort_rx_cplt_callback(uart_handle_t *p_uart)
功能说明	UART中止接收完成中断回调函数。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	无
备注	

2.23.4.27 hal_uart_get_state

表 2-433 hal_uart_get_state接口

函数原型	hal_uart_state_t hal_uart_get_state(uart_handle_t *p_uart)
功能说明	获取UART运行状态。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	UART运行状态，值可以是下面中的任意一个： • HAL_UART_STATE_RESET（未初始化） • HAL_UART_STATE_READY（已初始化且空闲） • HAL_UART_STATE_BUSY（忙） • HAL_UART_STATE_BUSY_TX（正在发送） • HAL_UART_STATE_BUSY_RX （正在接收） • HAL_UART_STATE_TIMEOUT（超时） • HAL_UART_STATE_ERROR（错误）
备注	

2.23.4.28 hal_uart_get_error

表 2-434 hal_uart_get_error接口

函数原型	uint32_t hal_uart_get_error(uart_handle_t *p_uart)
功能说明	获取UART错误码。
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	UART错误码，值可以是下面中的任意一个： • HAL_UART_ERROR_NONE（无错误） • HAL_UART_ERROR_PE（校验位错误） • HAL_UART_ERROR_FE（帧错误） • HAL_UART_ERROR_OE（溢出错误） • HAL_UART_ERROR_BI（Line Break错误） • HAL_UART_ERROR_DMA（DMA传输错误） • HAL_UART_ERROR_BUSY（忙）

2.23.4.29 hal_uart_suspend_reg

表 2-435 hal_uart_suspend_reg接口

函数原型	hal_status_t hal_uart_suspend_reg(uart_handle_t *p_uart)
功能说明	睡眠之前挂起和UART配置相关的寄存器
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态。

2.23.4.30 hal_uart_resume_reg

表 2-436 hal_uart_resume_reg接口

函数原型	hal_status_t hal_uart_resume_reg(uart_handle_t *p_uart)
功能说明	唤醒时恢复和UART配置相关的寄存器
输入参数	p_uart：指向uart_handle_t结构体变量的指针，该结构体变量包含指定的UART的配置信息。
返回值	HAL状态
备注	

2.24 HAL I2S通用驱动

2.24.1 I2S驱动功能

I2S（Inter-IC Sound）外设的HAL驱动主要实现了以下功能：

- 支持飞利浦I2S协议标准
- 发送和接收独立，支持全双工
- 支持主、从模式
- 支持12、16、20、24、32位的音频数据分辨率
- 支持轮询、中断、DMA三种IO操作方式。
- 支持中止中断及DMA方式下的数据收发/读写。
- 支持主设备及从设备模式下的发送完成、接收完成的中断回调函数。
- 支持中止完成、IO错误的中断回调函数。
- 支持获取驱动的I2S模式配置、运行状态及错误码。

2.24.2 如何使用I2S驱动

I2S的HAL驱动的使用方法如下：

1. 定义一个i2s_handle_t句柄结构体变量，例如：i2s_handle_t i2s_handle（i2s_handle_t结构体由I2S的HAL驱动定义，开发者在使用时需要定义一个该结构体类型的变量）。

2. 重写hal_i2s_msp_init()接口以初始化I2S底层资源：

   (1) 配置I2S对应GPIO引脚的功能复用、使能上拉电阻。

   (2) 如果需要使用中断或DMA方式的IO操作接口，则需调用相关的NVIC接口来配置：

   - 调用hal_nvic_set_priority()配置I2S的中断优先级。
   - 调用hal_nvic_enable_irq()使能I2S的NVIC中断。

   (3) 如果需要使用DMA方式的IO操作接口，则还需配置使用的DMA通道：

   - 定义用于发送/接收的dma_handle_t句柄结构体变量，如dma_handle_t dma_tx, dma_rx。
   - 配置DMA句柄dma_tx及dma_rx中的参数，如指定Tx或RX通道。
   - 将i2s_handler变量中的p_dmatx和p_dmarx指针分别指向已初始化的DMA句柄变量dma_tx和dma_rx。
   - 配置DMA的中断优先级、使能DMA的NVIC中断。

3. 配置I2S初始化结构体中的数据传输宽度、时钟源、音频频率。
4. 调用hal_i2s_init()配置I2S寄存器，配置过程中hal_i2s_init()会自动调用开发者重写的hal_i2s_msp_init()函数初始化I2S所使用的GPIO等底层资源。
5. 对于I2S的IO读写，支持三种操作方式：轮询、中断及DMA。

2.24.2.1 轮询方式的IO读写操作

1. 以轮询方式发送大量数据时使用hal_i2s_transmit()。
2. 以轮询方式接收大量数据时使用hal_i2s_receive()。
3. 以轮询方式发送与接收大量数据时使用hal_i2s_transmit_receive()。

2.24.2.2 中断方式的IO读写操作

1. 以中断非轮询方式发送大量数据时使用hal_i2s_transmit_it()，发送完成时回调函数hal_i2s_tx_cplt_callback()将会被调用。
2. 中断非轮询方式接收大量数据时使用hal_i2s_receive_it()，发送和接收完成时回调函数hal_i2s_rx_cplt_callback()将会被调用。
3. 中断非轮询方式发送与接收大量数据时使用hal_i2s_transmit_receive_it()，接收完成时回调函数hal_i2s_tx_rx_cplt_callback()将会被调用。
4. 如果数据收发过程中发生了错误，则hal_i2s_error_callback()回调函数将会被调用。
5. 如果需要中止数据收发，则可以使用hal_i2s_abort()。

说明:

对于上述回调函数，开发者均可通过重写函数完成指定的操作。

2.24.2.3 DMA方式的IO读写操作

1. 作为主设备以DMA非轮询方式发送大量数据时使用hal_i2s_transmit_dma()，发送完成时回调函数hal_i2s_tx_cplt_callback()将会被调用。
2. 作为主设备以DMA非轮询方式接收大量数据时使用hal_i2s_receive_dma()，接收完成时回调函数hal_i2s_rx_cplt_callback()将会被调用。
3. 作为主设备以DMA非轮询方式发送与接收大量数据时使用hal_i2s_transmit_receive_dma()，接收完成时回调函数hal_i2s_tx_rx_cplt_callback()将会被调用。
4. 如果数据收发过程中发生了错误，则hal_i2s_error_callback()回调函数将会被调用。

说明:

对于上述回调函数，开发者均可通过重写函数完成指定的操作。

2.24.3 I2S驱动的结构体

2.24.3.1 i2s_init_t

I2S驱动的初始化结构体i2s_init_t的定义如下：

表 2-437 i2s_init_t结构体

数据域	域段描述	取值
uint32_t data_size	数据传输宽度	该参数的取值可以是下列值中的任意一个： • I2S_DATASIZE_12BIT（12 bits） • I2S_DATASIZE_16BIT（16 bits） • I2S_DATASIZE_20BIT（20 bits） • I2S_DATASIZE_24BIT（24 bits） • I2S_DATASIZE_32BIT（32 bits） **说明** • data_size = I2S_DATASIZE_12BIT（12 bits），传输的数据以16bit地址对齐存放，高4 bits数据被忽略；硬件使用的WSS（字采样长度）为16 sclk cycles，高4 bits被忽略； • data_size = I2S_DATASIZE_20BIT（20 bits），传输的数据以32 bit地址对齐存放，高12 bits数据被忽略；硬件使用的WSS（字采样长度）为24 sclk cycles，高4 bit被忽略； • data_size = I2S_DATASIZE_24BIT（24 bits），传输的数据以32 bit地址对齐存放，高8 bits数据被忽略；硬件使用的WSS（字采样长度）为24 sclk cycles。

数据域	域段描述	取值
uint32_t clock_source	时钟源	该参数的取值可以是下列值中的任意一个： • I2S_CLOCK_SRC_96M • I2S_CLOCK_SRC_32M
uint32_t audio_freq	音频频率	audio_freq = fsck/(2 * wss)，fsck是I2S的串行时钟频率，最大取值为3027 kHz；WSS依赖于位宽参数可取值16、24、32，比如位宽配置为16 bit，WSS取值为16，audio_freq最大可配置为96 kHz。

2.24.3.2 i2s_handle_t

I2S驱动的句柄结构体i2s_handle_t的定义如下：

表 2-438 i2s_handle_t结构体

数据域	域段描述	取值
i2s_regs_t *p_instance	I2S实例。	该参数的取值可以是下列值中的任意一个： • I2S_M • I2S_S
i2s_init_t init	初始化结构体。	参考i2s_init_t结构体。
uint16_t *p_tx_buffer	指向数据传输缓冲区的指针（驱动负责管理，无需开发者初始化）。	N/A
__IO uint32_t tx_xfer_size	数据传输长度（驱动负责管理，无需开发者初始化）。	N/A
__IO uint32_t tx_xfer_count	数据传输计数（驱动负责管理，无需开发者初始化）。	N/A
uint16_t *p_rx_buffer	指向数据接收缓冲区的指针（驱动负责管理，无需开发者初始化）。	N/A
__IO uint32_t rx_xfer_size	数据接收长度（驱动负责管理，无需开发者初始化）。	N/A
__IO uint32_t rx_xfer_count	数据接收计数（驱动负责管理，无需开发者初始化）。	N/A

数据域	域段描述	取值
void (*write_fifo)(struct _i2s_handle *p_i2s)	指向I2S TX写FIFO函数的指针（驱动负责管理，无需开发者初始化）。	N/A
void (*read_fifo)(struct _i2s_handle *p_i2s)	指向I2S RX读FIFO函数的指针（驱动负责管理，无需开发者初始化）。	N/A
dma_handle_t *p_dmatx	指向I2S TX通道的DMA句柄的指针。	发送通道的DMA句柄dma_handle_t结构体。
dma_handle_t *p_dmarx	指向I2S RX通道的 DMA句柄的指针。	接收通道的DMA句柄dma_handle_t结构体。
__IO hal_lock_t lock	I2S锁（驱动负责管理，无需开发者初始化）。	N/A
__IO hal_i2s_state_t state	I2S运行状态（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_I2S_STATE_RESET（未初始化） • HAL_I2S_STATE_READY（已初始化且空闲） • HAL_I2S_STATE_BUSY（忙） • HAL_I2S_STATE_BUSY_TX（正在发送） • HAL_I2S_STATE_BUSY_RX （正在接收） • HAL_I2S_STATE_BUSY_TX_RX （正在进行发送与接收） • HAL_I2S_STATE_ABORT（被中断） • HAL_I2S_STATE_ERROR（错误）
__IO uint32_t error_code	I2S错误码（无需开发者初始化）。	该参数的取值可以是下列值中的任意一个： • HAL_I2S_ERROR_NONE（无错误） • HAL_I2S_ERROR_TIMEOUT（超时） • HAL_I2S_ERROR_TRANSFER（传输错误） • HAL_I2S_ERROR_DMA（DMA传输错误） • HAL_I2S_ERROR_INVALID_PARAM（参数错误）

2.24.4 I2S驱动API描述

I2S驱动的API主要包括：

表 2-439 I2S驱动的APIs

API类别	API名称	描述
初始化	hal_i2s_init()	初始化I2S外设，配置参数。
	hal_i2s_deinit()	反初始化I2S外设。
	hal_i2s_msp_init()	初始化I2S外设所使用的GPIO引脚复用、NVIC中断、DMA通道。

API类别	API名称	描述
	hal_i2s_msp_deinit()	反初始化I2S外设所使用的GPIO引脚复用、NVIC中断、DMA通道。
IO操作	hal_i2s_transmit()	数据发送，轮询方式。
	hal_i2s_receive()	数据接收，轮询方式。
	hal_i2s_transmit_receive()	数据发送与接收，轮询方式。
	hal_i2s_transmit_it()	数据发送，中断方式。
	hal_i2s_receive_it()	数据接收，中断方式。
	hal_i2s_transmit_receive_it()	数据发送与接收，中断方式。
	hal_i2s_transmit_dma()	数据发送，DMA方式。
	hal_i2s_receive_dma()	数据接收，DMA方式。
	hal_i2s_transmit_receive_dma()	数据发送与接收，DMA方式。
	hal_i2s_abort()	中止中断及DMA方式下的数据传输。
中断处理及回调函数	hal_i2s_irq_handler()	中断处理函数。
	hal_i2s_tx_cplt_callback()	发送完成中断回调函数。
	hal_i2s_rx_cplt_callback()	接收完成中断回调函数。
	hal_i2s_error_callback()	错误中断回调函数。
状态及错误	hal_i2s_get_state()	获取驱动运行状态。
	hal_i2s_get_error()	获取错误码。
时钟控制	hal_i2s_start_clock()	作为主模式的时候开始输出时钟。
	hal_i2s_stop_clock()	作为主模式的时候停止输出时钟。
FIFO阈值操作函数	hal_i2s_set_tx_fifo_threshold	设置TX FIFO阈值。
	hal_i2s_set_rx_fifo_threshold	设置RX FIFO阈值。
	hal_i2s_get_tx_fifo_threshold	获取TX FIFO阈值。
	hal_i2s_get_rx_fifo_threshold	获取RX FIFO阈值。
睡眠相关	hal_i2s_suspend_reg()	睡眠之前挂起和I2S配置相关的寄存器。
	hal_i2s_resume_reg()	唤醒时恢复和I2S配置相关的寄存器。

下面章节将对各API进行详细描述。

2.24.4.1 hal_i2s_init

表 2-440 hal_i2s_init接口

函数原型	hal_status_t hal_i2s_init(i2s_handle_t *p_i2s)
功能说明	根据i2s_init_t中的指定参数初始化I2S和关联句柄。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	HAL状态。

备注	

2.24.4.2 hal_i2s_deinit

表 2-441 hal_i2s_deinit接口

函数原型	hal_status_t hal_i2s_deinit(i2s_handle_t *p_i2s)
功能说明	反初始化I2S。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	HAL状态。
备注	

2.24.4.3 hal_i2s_msp_init

表 2-442 hal_i2s_msp_init接口

函数原型	void hal_i2s_msp_init(i2s_handle_t *p_i2s)
功能说明	初始化I2S所使用的GPIO引脚复用、NVIC中断、DMA通道等配置。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成GPIO引脚复用、NVIC中断、DMA通道的初始化。

2.24.4.4 hal_i2s_msp_deinit

表 2-443 hal_i2s_msp_deinit接口

函数原型	void hal_i2s_msp_deinit(i2s_handle_t *p_i2s)
功能说明	反初始化I2S所使用的GPIO引脚复用、NVIC中断、DMA通道等配置。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息
返回值	无
备注	该函数为weak类型的空函数，开发者需重写该API完成GPIO引脚复用、NVIC中断、DMA通道的反初始化。

2.24.4.5 hal_i2s_transmit

表 2-444 hal_i2s_transmit接口

函数原型	hal_status_t hal_i2s_transmit(i2s_handle_t *p_i2s, uint16_t *p_data, uint32_t length, uint32_t timeout)
功能说明	发送大量数据，轮询方式。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。 p_data：指向数据缓冲区的指针。

	length：待发送数据的长度，单声道的数据量，单位2 Bytes。 timeout：超时时间，单位ms。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_i2s_get_error()获取具体的错误码。

2.24.4.6 hal_i2s_receive

表 2-445 hal_i2s_receive接口

函数原型	hal_status_t hal_i2s_receive(i2s_handle_t *p_i2s, uint16_t *p_data, uint32_t length, uint32_t timeout)
功能说明	接收大量数据，轮询方式。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。 p_data：指向数据缓冲区的指针。 length：待接收数据的长度，单声道的数据量，单位2 Bytes。 timeout：超时时间，单位ms。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_i2s_get_error()获取具体的错误码。

2.24.4.7 hal_i2s_transmit_receive

表 2-446 hal_i2s_transmit_receive接口

函数原型	hal_status_t hal_i2s_transmit_receive(i2s_handle_t *p_i2s, uint16_t *p_tx_data, uint16_t *p_rx_data, uint32_t length, uint32_t timeout)
功能说明	发送与接收大量数据，轮询方式。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。 p_tx_data：指向发送数据缓冲区的指针。 p_rx_data：指向接收数据缓冲区的指针。 length：发送与接收数据的长度，单位Byte。 timeout：超时时间，单位ms。
返回值	HAL状态。
备注	返回HAL_ERROR时可调用hal_i2s_get_error()获取具体的错误码。

2.24.4.8 hal_i2s_transmit_it

表 2-447 hal_i2s_transmit_it接口

函数原型	hal_status_t hal_i2s_transmit_it(i2s_handle_t *p_i2s, uint16_t *p_data, uint32_t length)
功能说明	发送大量数据，中断方式。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。 p_data：指向数据缓冲区的指针。

	length：待发送数据的长度，单声道的数据量，单位2 Bytes。
返回值	HAL状态。
备注	• 发送完成时，回调函数hal_i2s_tx_cplt_callback()会被调用。 • 发送过程中出现错误时，回调函数hal_i2s_error_callback()会被调用，开发者可在回调函数中调用hal_i2s_get_error()获取具体的错误码。 • 回调函数hal_i2s_tx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.24.4.9 hal_i2s_receive_it

表 2-448 hal_i2s_receive_it接口

函数原型	hal_status_t hal_i2s_receive_it(i2s_handle_t *p_i2s, uint16_t *p_data, uint32_t length)
功能说明	接收大量数据，中断方式。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。 p_data：指向数据缓冲区的指针。 length：待接收数据的长度，单声道的数据量，单位2 Bytes。
返回值	HAL状态。
备注	• 接收完成时，回调函数hal_i2s_rx_cplt_callback()会被调用。 • 发送和接收过程中出现错误时，回调函数hal_i2s_error_callback()会被调用，开发者可在回调函数中调用hal_i2s_get_error()获取具体的错误码。 • 回调函数hal_i2s_rx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.24.4.10 hal_i2s_transmit_receive_it

表 2-449 hal_i2s_transmit_receive_it接口

函数原型	hal_status_t hal_i2s_transmit_receive_it(i2s_handle_t *p_i2s, uint16_t *p_tx_data, uint16_t *p_rx_data, uint32_t length)
功能说明	发送与接收大量数据，中断方式。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。 p_tx_data：指向发送数据缓冲区的指针。 p_rx_data：指向接收数据缓冲区的指针。 length：发送与接收数据的长度，单位Byte。
返回值	HAL状态。
备注	• 接收完成时，回调函数hal_i2s_tx_rx_cplt_callback ()会被调用。 • 接收过程中出现错误时，回调函数hal_i2s_error_callback()会被调用，开发者可在回调函数中调用hal_i2s_get_error()获取具体的错误码。 • 回调函数hal_i2s_rx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.24.4.11 hal_i2s_abort

表 2-450 hal_i2s_abort接口

函数原型	hal_status_t hal_i2s_abort(i2s_handle_t *p_i2s)

功能说明	中止I2S中断或DMA方式的数据传输。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	HAL状态。
备注	该函数为轮询式函数，退出函数时，中止完成。

2.24.4.12 hal_i2s_transmit_dma

表 2-451 hal_i2s_transmit_dma接口

函数原型	hal_status_t hal_i2s_transmit_dma(i2s_handle_t *p_i2s, uint16_t *p_data, uint32_t length)
功能说明	发送大量数据，DMA方式。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。 p_data：指向数据缓冲区的指针。 length：待发送数据的长度。
返回值	HAL状态。
备注	• 发送完成时，回调函数hal_i2s_tx_cplt_callback()会被调用。 • 发送过程中出现错误时，回调函数hal_i2s_error_callback()会被调用，开发者可在回调函数中调用hal_i2s_get_error()获取具体的错误码。 • 回调函数hal_i2s_tx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.24.4.13 hal_i2s_receive_dma

表 2-452 hal_i2s_receive_dma接口

函数原型	hal_status_t hal_i2s_receive_dma(i2s_handle_t *p_i2s, uint16_t *p_data, uint32_t length)
功能说明	以DMA非轮询模式接收大量数据。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。 p_data：指向数据缓冲区的指针。 length：待接收数据的长度。
返回值	HAL状态。
备注	• 接收完成时，回调函数hal_i2s_rx_cplt_callback()会被调用。 • 接收过程中出现错误时，回调函数hal_i2s_error_callback()会被调用，开发者可在回调函数中调用hal_i2s_get_error()获取具体的错误码。 • 回调函数hal_i2s_rx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.24.4.14 hal_i2s_transmit_receive_dma

表 2-453 hal_i2s_transmit_receive_dma接口

函数原型	hal_status_t hal_i2s_transmit_receive_dma(i2s_handle_t *p_i2s, uint16_t *p_tx_data, uint16_t *p_rx_data, uint32_t length)
功能说明	发送与接收大量数据，DMA方式。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。

	p_tx_data：指向发送数据缓冲区的指针。 p_rx_data：指向接收数据缓冲区的指针。 length：发送与接收数据的长度，单位Byte。
返回值	HAL状态。
备注	• 接收完成时，回调函数hal_i2s_tx_rx_cplt_callback()会被调用。 • 发送和接收过程中出现错误时，回调函数hal_i2s_error_callback()会被调用，开发者可在回调函数中调用hal_i2s_get_error()获取具体的错误码。 • 回调函数hal_i2s_rx_cplt_callback()被调用前，请勿释放data指向的数据缓冲区的内存。

2.24.4.15 hal_i2s_irq_handler

表 2-454 hal_i2s_irq_handler接口

函数原型	void hal_i2s_irq_handler(i2s_handle_t *p_i2s)
功能说明	处理I2S中断请求。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	无
备注	

2.24.4.16 hal_i2s_tx_cplt_callback

表 2-455 hal_i2s_tx_cplt_callback接口

函数原型	void hal_i2s_tx_cplt_callback(i2s_handle_t *p_i2s)
功能说明	发送完成回调函数。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.24.4.17 hal_i2s_tx_rx_cplt_callback

表 2-456 hal_i2s_tx_rx_cplt_callback接口

函数原型	void hal_i2s_tx_rx_cplt_callback(i2s_handle_t *p_i2s)
功能说明	发送与接收完成回调函数。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.24.4.18 hal_i2s_rx_cplt_callback

表 2-457 hal_i2s_rx_cplt_callback接口

函数原型	void hal_i2s_rx_cplt_callback(i2s_handle_t *p_i2s)
功能说明	接收完成回调函数。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.24.4.19 hal_i2s_error_callback

表 2-458 hal_i2s_error_callback接口

函数原型	void hal_i2s_error_callback(i2s_handle_t *p_i2s)
功能说明	I2S错误回调函数。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.24.4.20 hal_i2s_get_state

表 2-459 hal_i2s_get_state接口

函数原型	hal_i2s_state_t hal_i2s_state_t hal_i2s_get_state(i2s_handle_t *p_i2s)
功能说明	获取I2S运行状态。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	I2S运行状态，该参数的取值可以是下列值中的任意一个： • HAL_I2S_STATE_RESET（未初始化） • HAL_I2S_STATE_READY（已初始化且空闲） • HAL_I2S_STATE_BUSY（忙） • HAL_I2S_STATE_BUSY_TX（正在发送） • HAL_I2S_STATE_BUSY_RX （正在接收） • HAL_I2S_STATE_BUSY_TX_RX （正在进行发送与接收） • HAL_I2S_STATE_ABORT（被中断） • HAL_I2S_STATE_ERROR（错误）
备注	

2.24.4.21 hal_i2s_get_error

表 2-460 hal_i2s_get_error接口

函数原型	uint32_t hal_i2s_get_error(i2s_handle_t *p_i2s)
功能说明	返回I2S句柄错误码。

输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	I2S错误码，该参数的取值可以是下列值中的任意一个： • HAL_I2S_ERROR_NONE（无错误） • HAL_I2S_ERROR_TIMEOUT（超时） • HAL_I2S_ERROR_TRANSFER（传输错误） • HAL_I2S_ERROR_DMA（DMA传输错误） • HAL_I2S_ERROR_INVALID_PARAM（参数错误）
备注	

2.24.4.22 hal_i2s_start_clock

表 2-461 hal_i2s_start_clock接口

函数原型	hal_status_t hal_i2s_start_clock(i2s_handle_t *p_i2s)
功能说明	开始输出时钟（作为主设备）。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	HAL状态。
备注	

2.24.4.23 hal_i2s_stop_clock

表 2-462 hal_i2s_stop_clock接口

函数原型	hal_status_t hal_i2s_stop_clock(i2s_handle_t *p_i2s)
功能说明	停止输出时钟（作为主设备）。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	HAL状态。
备注	

2.24.4.24 hal_i2s_set_tx_fifo_threshold

表 2-463 hal_i2s_set_tx_fifo_threshold接口

函数原型	hal_status_t hal_i2s_set_tx_fifo_threshold(i2s_handle_t *p_i2s, uint32_t threshold)
功能说明	设置TX FIFO阈值。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。 threshold:待设置的阈值参数。
返回值	HAL状态。
备注	

2.24.4.25 hal_i2s_set_rx_fifo_threshold

表 2-464 hal_i2s_set_rx_fifo_threshold接口

函数原型	hal_status_t hal_i2s_set_rx_fifo_threshold(i2s_handle_t *p_i2s, uint32_t threshold)
功能说明	设置RX FIFO阈值。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。 threshold: 待设置的阈值参数。
返回值	HAL状态。
备注	

2.24.4.26 hal_i2s_get_tx_fifo_threshold

表 2-465 hal_i2s_get_tx_fifo_threshold接口

函数原型	uint32_t hal_i2s_get_tx_fifo_threshold(i2s_handle_t *p_i2s)
功能说明	获取TX FIFO阈值。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	TX FIFO阈值。
备注	

2.24.4.27 hal_i2s_get_rx_fifo_threshold

表 2-466 hal_i2s_get_rx_fifo_threshold接口

函数原型	uint32_t hal_i2s_get_tx_fifo_threshold(i2s_handle_t *p_i2s)
功能说明	获取RX FIFO阈值。
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	RX FIFO阈值。
备注	

2.24.4.28 hal_i2s_suspend_reg

表 2-467 hal_i2s_suspend_reg接口

函数原型	hal_status_t hal_i2s_suspend_reg(i2s_handle_t *p_i2s)
功能说明	睡眠之前挂起和I2S配置相关的寄存器
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	HAL状态。
备注	

2.24.4.29 hal_i2s_resume_reg

表 2-468 hal_i2s_resume_reg接口

函数原型	hal_status_t hal_i2s_resume_reg(i2s_handle_t *p_i2s)
功能说明	唤醒时恢复和I2S配置相关的寄存器
输入参数	p_i2s：指向i2s_handle_t结构体变量的指针，该结构体变量包含指定的I2S的配置信息。
返回值	HAL状态。
备注	

2.25 HAL RNG通用驱动

2.25.1 RNG驱动功能

RNG（Random Number Generation）外设的HAL驱动主要实现了以下功能：

- 支持真随机数和伪随机数的生成。
- 生成的随机数可通过NIST SP 800-22标准测试。
- 支持多种后处理方式，包含：跳位、位计数、冯纽曼。
- 支持中断和轮询模式。

2.25.2 如何使用RNG驱动

RNG HAL驱动使用方法如下：

1. 重写hal_rng_msp_init()接口，在该接口中调用hal_nvic_set_priority()及hal_nvic_enable_irq()使能RNG的NVIC中断。
2. 声明一个rng_handle_t句柄结构体，例如：rng_handle_t p_rng，并设置“p_instance”成员为RNG实例。
3. 配置p_rng句柄的初始化结构体init成员中的计数初值和复位模式。
4. 调用hal_rng_init()初始化RNG外设。
5. 调用hal_rng_generate_random_number()以轮询方式生成随机数或调用hal_rng_generate_random_number_it()以中断方式生成随机数。若采用RNG_SEED_USER方式，需提供59bits或128bits的随机数种子。
6. 以中断方式生成随机数时，当随机数生成后回调函数hal_rng_ready_data_callback()会被调用，开发者可根据需要重写该API。

2.25.3 RNG驱动的结构体

2.25.3.1 rng_init_t

RNG驱动的初始化结构体rng_init_t的定义如下：

表 2-469 rng_init_t结构体

数据域	域段描述	取值
uint32_t seed_mode	指定LFSR的种子方式	该参数的取值可以是下列值中的任意一个： • RNG_SEED_FR0_S0（LFSR种子来自开关振荡器s0） • RNG_SEED_USER（LFSR种子由用户配置）
uint32_t lfsr_mode	LFSR配置模式	该参数的取值可以是下列值中的任意一个： • RNG_LFSR_MODE_59BIT（59bit LFSR） • RNG_LFSR_MODE_128BIT（128bit LFSR）
uint32_t out_mode	随机数输出模式	该参数的取值可以是下列值中的任意一个： • RNG_OUTPUT_FR0_S0（数字RNG直接输出） • RNG_OUTPUT_CYCLIC_PARITY（LFSR和RNG循环采样和奇偶校验生成） • RNG_OUTPUT_CYCLIC（LFSR和RNG循环采样） • RNG_OUTPUT_LFSR_RNG（LFSR ⊕ RNG） • RNG_OUTPUT_LFSR（LFSR直接输出） 说明: 当seed_mode选择为RNG_SEED_USER时，out_mode不能选择为RNG_OUTPUT_FR0_S0。
uint32_t post_mode	后处理模式	该参数的取值可以是下列值中的任意一个： • RNG_POST_PRO_NOT（不进行处理） • RNG_POST_PRO_SKIPPING（跳位处理） • RNG_OUTPUT_CYCLIC（位计数处理） • RNG_OUTPUT_LFSR_RNG（冯纽曼处理）

2.25.3.2 rng_handle_t

RNG驱动的句柄结构体rng_handle_t的定义如下：

表 2-470 rng_handle_t结构体

数据域	域段描述	取值
rng_regs_t *p_instance	RNG外设实例	该参数的取值可以是RNG。
rng_init_t init	初始化结构体	参考rng_init_t结构体
__IO hal_lock_t lock	RNG锁（无需开发者初始化）	N/A
__IO hal_rng_state_t g_state	RNG运行状态（无需开发者初始化）	该参数的取值可以是下列值中的任意一个：

数据域	域段描述	取值
		• HAL_RNG_STATE_RESET（未初始化） • HAL_RNG_STATE_READY（已初始化且空闲） • HAL_RNG_STATE_BUSY（忙） • HAL_RNG_STATE_TIMEOUT（超时） • HAL_RNG_STATE_ERROR（错误）
uint32_t random_number	最后生成的随机数	取值范围：0x00000000 ～ 0xFFFFFFFF。
uint32_t retention[1]	保存RNG寄存器信息（驱动负责管理，无需开发者初始化）	N/A

2.25.4 RNG驱动API描述

RNG驱动的API主要如下：

表 2-471 RNG驱动的APIs

API类别	API名称	描述
初始化	hal_rng_init()	初始化RNG外设，配置计数初值等参数。
	hal_rng_deinit()	反初始化RNG外设。
	hal_rng_msp_init()	初始化RNG外设所使用的NVIC中断。
	hal_rng_msp_deinit()	反初始化RNG外设所使用的NVIC中断。
IO操作	hal_rng_generate_random_number	生成随机数，轮询方式。
	hal_rng_generate_random_number_it	生成随机数，中断方式。
	hal_rng_read_last_random_number	获取最后生成的随机数。
中断处理及回调函数	hal_rng_irq_handler	中断处理函数。
	hal_rng_ready_data_callback	计数完成回调函数。
睡眠相关	hal_rng_suspend_reg()	睡眠之前挂起和RNG配置相关的寄存器。
	hal_rng_resume_reg()	唤醒时恢复和RNG配置相关的寄存器。

下面章节将对各API进行详细描述。

2.25.4.1 hal_rng_init

表 2-472 hal_rng_init接口

函数原型	hal_status_t hal_rng_init(rng_handle_t *p_rng)
功能说明	根据RNG句柄中的指定参数初始化RNG。
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。
返回值	HAL状态。
备注	

2.25.4.2 hal_rng_deinit

表 2-473 hal_rng_deinit接口

函数原型	hal_status_t hal_rng_deinit(rng_handle_t *p_rng)
功能说明	将RNG外设寄存器反初始化为它们的默认值。
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。
返回值	HAL状态。
备注	

2.25.4.3 hal_rng_msp_init

表 2-474 hal_rng_msp_init接口

函数原型	void hal_rng_msp_init(rng_handle_t *p_rng)
功能说明	初始化RNG外设所使用的NVIC中断。
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需要重写该API以初始化RNG中断。

2.25.4.4 hal_rng_msp_deinit

表 2-475 hal_rng_msp_deinit接口

函数原型	void hal_rng_msp_deinit(rng_handle_t *p_rng)
功能说明	反初始化RNG外设所使用的NVIC中断。
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需要重写该API以关闭NVIC中断。

2.25.4.5 hal_rng_generate_random_number

表 2-476 hal_rng_generate_random_number接口

函数原型	hal_status_t hal_rng_generate_random_number(rng_handle_t *p_rng, uint16_t *p_seed, uint32_t *p_random32bit)
功能说明	以轮询方式生产随机数。
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。 p_seed：指向用户配置的随机数种子的指针。当rng_init_t结构体seed_mode成员被配置为RNG_SEED_USER时，该参数才能生效。 p_random32bit：指向RNG模块生成的随机数的指针。
返回值	HAL状态。

备注	

2.25.4.6 hal_rng_generate_random_number_it

表 2-477 hal_rng_generate_random_number_it接口

函数原型	hal_status_t hal_rng_generate_random_number_it(rng_handle_t *p_rng, uint16_t *p_seed, uint32_t *p_random32bit)
功能说明	以中断方式生产随机数。
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。 p_seed：指向用户配置的随机数种子的指针。当rng_init_t结构体seed_mode成员被配置为RNG_SEED_USER时，该参数才能生效。 p_random32bit：指向RNG模块生成的随机数的指针。
返回值	HAL状态。
备注	

2.25.4.7 hal_rng_read_last_random_number

表 2-478 hal_rng_read_last_random_number接口

函数原型	uint32_t hal_rng_read_last_random_number(rng_handle_t *p_rng)
功能说明	获取最后生成的随机数值。
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。
返回值	最后生成的随机数，取值范围：0x00000000 ～ 0xFFFFFFFF。
备注	

2.25.4.8 hal_rng_irq_handler

表 2-479 hal_rng_irq_handler接口

函数原型	void hal_rng_irq_handler(rng_handle_t *p_rng)
功能说明	处理RNG中断请求。
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。
返回值	无
备注	

2.25.4.9 hal_rng_ready_data_callback

表 2-480 hal_rng_ready_data_callback接口

函数原型	void hal_rng_ready_data_callback(rng_handle_t *p_rng, uint32_t random32bit)
功能说明	RNG生成完成中断回调函数。
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。

	random32bit：RNG模块生成的随机数。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.25.4.10 hal_rng_suspend_reg

表 2-481 hal_rng_suspend_reg接口

函数原型	hal_status_t hal_rng_suspend_reg(rng_handle_t *p_rng);
功能说明	睡眠之前挂起和RNG配置相关的寄存器
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。
返回值	HAL状态
备注	

2.25.4.11 hal_rng_resume_reg

表 2-482 hal_rng_resume_reg接口

函数原型	hal_status_t hal_rng_resume_reg(rng_handle_t *p_rng)
功能说明	唤醒时恢复和RNG配置相关的寄存器
输入参数	p_rng：指向rng_handle_t结构体变量的指针，该结构体变量包含指定的RNG模块的配置信息。
返回值	HAL状态
备注	

2.26 HAL AON WDT通用驱动

2.26.1 AON WDT驱动功能

AON WDT（Always-on Watchdog Timer）外设的HAL驱动主要实现了以下功能：

- 支持复位模式的使能及禁止。复位模式使能时，AON WDT向下计数到alarm_counter后触发中断，继续向下计数到0后进行系统复位。
- 支持计数初值重载，即喂狗操作。
- 支持中断回调函数。

2.26.2 如何使用AON WDT驱动

AON WDT HAL驱动使用方法如下：

1. 声明一个aon_wdt_handle_t句柄结构体，例如：aon_wdt_handle_t hwdt。
2. 配置p_aon_wdt句柄的初始化结构体init成员中的计数初值和复位模式。
3. 调用hal_aon_wdt_init()初始化AON WDT外设。

4. 开发者需要在AON WDT向下计数到0前调用hal_aon_wdt_refresh()重载计数初值，否则AON WDT会进行系统复位。

5. AON WDT向下计数到alarm_counter时，中断回调函数hal_wdt_period_elapsed_callback()会被调用，开发者可根据需要重写该API。

2.26.3 AON WDT驱动的结构体

2.26.3.1 aon_wdt_init_t

AON WDT驱动的初始化结构体aon_wdt_init_t的定义如下：

表 2-483 aon_wdt_init_t结构体

数据域	域段描述	取值
uint32_t counter	计数初值。	0x0000_0000 ～ 0xFFFF_FFFF
uint32_t alarm_counter	复位警告初始值。	取值范围为0 ～ 20。

2.26.3.2 aon_wdt_handle_t

AON WDT驱动的句柄结构体aon_wdt_handle_t的定义如下：

表 2-484 aon_aon_wdt_handle_t结构体

数据域	域段描述	取值
aon_wdt_init_t init	初始化结构体。	参考aon_wdt_init_t结构体。
__IO hal_lock_t lock	AON WDT锁（无需开发者初始化）。	N/A

2.26.4 AON WDT驱动API描述

AON WDT驱动的API主要如下：

表 2-485 AON WDT驱动的APIs

API类别	API名称	描述
初始化	hal_aon_wdt_init()	初始化AON WDT外设，配置计数初值等参数。
	hal_aon_wdt_deinit()	反初始化AON WDT外设。
IO操作	hal_aon_wdt_refresh	重载计数初值。
中断处理及回调函数	hal_aon_wdt_irq_handler	中断处理函数。
	hal_aon_wdt_alarm_callback	复位警告回调函数。

下面章节将对各API进行详细描述。

2.26.4.1 hal_aon_wdt_init

表 2-486 hal_aon_wdt_init接口

函数原型	hal_status_t hal_aon_wdt_init(aon_wdt_handle_t *p_aon_wdt)
功能说明	根据AON WDT句柄中的指定参数初始化AON WDT。
输入参数	p_aon_wdt：指向aon_wdt_handle_t结构体变量的指针，该结构体变量包含指定的AON WDT模块的配置信息。
返回值	HAL状态。
备注	

2.26.4.2 hal_aon_wdt_deinit

表 2-487 hal_aon_wdt_deinit接口

函数原型	hal_status_t hal_aon_wdt_deinit(aon_wdt_handle_t *p_aon_wdt)
功能说明	将AON WDT外设寄存器反初始化为它们的默认值。
输入参数	p_aon_wdt：指向aon_wdt_handle_t结构体变量的指针，该结构体变量包含指定的AON WDT模块的配置信息。
返回值	HAL状态。
备注	

2.26.4.3 hal_aon_wdt_refresh

表 2-488 hal_aon_wdt_refresh接口

函数原型	hal_status_t hal_aon_wdt_refresh(aon_wdt_handle_t *p_aon_wdt)
功能说明	刷新看门狗计数。
输入参数	p_aon_wdt：指向aon_wdt_handle_t结构体变量的指针，该结构体变量包含指定的AON_WDT模块的配置信息。
返回值	HAL状态。
备注	

2.26.4.4 hal_aon_wdt_irq_handler

表 2-489 hal_aon_wdt_irq_handler接口

函数原型	void hal_aon_wdt_irq_handler(aon_wdt_handle_t *p_aon_wdt)
功能说明	处理AON WDT中断请求。
输入参数	p_aon_wdt：指向aon_wdt_handle_t结构体变量的指针，该结构体变量包含指定的AON WDT模块的配置信息。
返回值	无

备注	

2.26.4.5 hal_aon_wdt_alarm_callback

表 2-490 hal_aon_wdt_alarm_callback接口

函数原型	void hal_aon_wdt_alarm_callback(aon_wdt_handle_t *p_aon_wdt)
功能说明	中断回调函数，AON WDT计数到alarm count时被调用。
输入参数	p_aon_wdt：指向aon_wdt_handle_t结构体变量的指针，该结构体变量包含指定的AON WDT模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.27 HAL WDT通用驱动

2.27.1 WDT驱动功能

WDT（WatchDog Timer）外设的HAL驱动主要实现了以下功能：

- 支持复位模式的使能及禁止。复位模式使能时，WDT向下计数到0后触发中断；看门狗模式下，第一次向下计数到0后触发中断，第二次向下计数到0后进行系统复位。
- 支持计数初值重载，即喂狗操作。
- 支持中断回调函数。

2.27.2 如何使用WDT驱动

WDT HAL驱动使用方法如下：

1. 重写hal_wdt_msp_init()接口，在该接口中调用hal_nvic_set_priority()及hal_nvic_enable_irq()使能WDT的NVIC中断。
2. 声明一个wdt_handle_t句柄结构体，例如：wdt_handle_t hwdt，并设置“p_instance”成员为WDT实例。
3. 配置hwdt句柄的初始化结构体init成员中的计数初值和复位模式。
4. 调用hal_wdt_init()初始化WDT外设。
5. 若初始化结构体中复位模式为WDT_RESET_ENABLE，则开发者需要在WDT第二次向下计数到0前调用hal_wdt_refresh()进行计数初值重载，否则WDT会进行系统复位。
6. WDT在第一次向下计数到0时会调用中断回调函数hal_wdt_period_elapsed_callback()，开发者可根据应用场景重写该API。

2.27.3 WDT驱动的结构体

2.27.3.1 wdt_init_t

WDT驱动的初始化结构体wdt_init_t的定义如下：

表 2-491 wdt_init_t结构体

数据域	域段描述	取值
uint32_t counter	计数初值	0x0000_0000 ～ 0xFFFF_FFFF
uint32_t reset_mode	计数模式	该参数的取值可以是下列值中的任意一个： • WDT_RESET_DISABLE（复位模式禁用） • WDT_RESET_ENABLE（复位模式启用）

2.27.3.2 wdt_handle_t

WDT驱动的句柄结构体wdt_handle_t的定义如下：

表 2-492 wdt_handle_t结构体

数据域	域段描述	取值
wdt_regs_t *p_instance	WDT外设实例	该参数的取值可以是WDT。
wdt_init_t init	初始化结构体	参考wdt_init_t结构体。
__IO hal_lock_t lock	WDT锁（无需开发者初始化）	N/A

2.27.4 WDT驱动API描述

WDT驱动的API主要如下：

表 2-493 WDT驱动的APIs

API类别	API名称	描述
初始化	hal_wdt_init()	初始化WDT外设，配置计数初值等参数。
	hal_wdt_deinit()	反初始化WDT外设。
	hal_wdt_msp_init()	初始化WDT外设所使用的NVIC中断。
	hal_wdt_msp_deinit()	反初始化WDT外设所使用的NVIC中断。
IO操作	hal_wdt_refresh	重载计数初值。
中断处理及回调函数	hal_wdt_irq_handler	中断处理函数。
	hal_wdt_period_elapsed_callback	计数完成回调函数。

下面章节将对各API进行详细描述。

2.27.4.1 hal_wdt_init

表 2-494 hal_wdt_init接口

函数原型	hal_status_t hal_wdt_init(wdt_handle_t *p_wdt)
功能说明	根据WDT句柄中的指定配置参数初始化WDT。
输入参数	p_wdt：指向wdt_handle_t结构体变量的指针，该结构体变量包含指定的WDT模块的配置信息。
返回值	HAL状态。
备注	

2.27.4.2 hal_wdt_deinit

表 2-495 hal_wdt_deinit接口

函数原型	hal_status_t hal_wdt_deinit(wdt_handle_t *p_wdt)
功能说明	将WDT外设寄存器反初始化为它们的默认重置值。
输入参数	p_wdt：指向wdt_handle_t结构体变量的指针，该结构体变量包含指定的WDT模块的配置信息。
返回值	HAL状态。
备注	

2.27.4.3 hal_wdt_msp_init

表 2-496 hal_wdt_msp_init接口

函数原型	void hal_wdt_msp_init(wdt_handle_t *p_wdt)
功能说明	初始化WDT外设所使用的NVIC中断。
输入参数	p_wdt：指向wdt_handle_t结构体变量的指针，该结构体变量包含指定的WDT模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需要重写该API以进行NVIC中断的初始化。

2.27.4.4 hal_wdt_msp_deinit

表 2-497 hal_wdt_msp_deinit接口

函数原型	void hal_wdt_msp_deinit(wdt_handle_t *p_wdt)
功能说明	反初始化WDT外设所使用的NVIC中断。
输入参数	p_wdt：指向wdt_handle_t结构体变量的指针，该结构体变量包含指定的WDT模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需要重写该API以进行NVIC中断的反初始化。

2.27.4.5 hal_wdt_refresh

表 2-498 hal_wdt_refresh接口

函数原型	hal_status_t hal_wdt_refresh(wdt_handle_t *p_wdt)
功能说明	刷新看门狗计数。
输入参数	p_wdt：指向wdt_handle_t结构体变量的指针，该结构体变量包含指定的WDT模块的配置信息。
返回值	HAL状态。
备注	

2.27.4.6 hal_wdt_irq_handler

表 2-499 hal_wdt_irq_handler接口

函数原型	void hal_wdt_irq_handler(wdt_handle_t *p_wdt)
功能说明	处理WDT中断请求。
输入参数	p_wdt：指向wdt_handle_t结构体变量的指针，该结构体变量包含指定的WDT模块的配置信息。
返回值	无
备注	

2.27.4.7 hal_wdt_period_elapsed_callback

表 2-500 hal_wdt_period_elapsed_callback接口

函数原型	void hal_wdt_period_elapsed_callback(wdt_handle_t *p_wdt)
功能说明	WDT计数到0的中断回调函数。
输入参数	p_wdt：指向wdt_handle_t结构体变量的指针，该结构体变量包含指定的WDT模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.28 HAL COMP通用驱动

2.28.1 COMP驱动功能

COMP（比较器）的HAL驱动主要实现了以下功能：

- 支持输入源、参考源可配置。
- 支持结果中断触发以及中断回调函数。

2.28.2 如何使用COMP驱动

COMP HAL驱动使用方法如下：

1. 重写hal_comp_msp_init()接口，在该接口中调用hal_nvic_set_priority()及hal_nvic_enable_irq()使能COMP的NVIC中断。
2. 重写hal_comp_trigger_callback()接口。
3. 声明一个comp_handle_t句柄结构体，例如：comp_handle_t g_comp_handle。
4. 配置g_comp_handle句柄的初始化结构体init成员中的输入源、参考源和参考值。
5. 调用hal_comp_init()初始化COMP模块。
6. 调用hal_comp_start开始比较器功能。
7. 调用hal_comp_stop停止比较器功能。

说明:

- GR5515芯片采用单电源供电，不支持负电压输入。
- 在实际使用中，需要将外部的物理量作为输入，IO不能为悬空态。

2.28.3 COMP驱动的结构体

2.28.3.1 comp_init_t

COMP驱动的初始化结构体comp_init_t的定义如下：

```
typedef ll_comp_init_t comp_init_t
```

详细请参考ll_comp_init_t。

2.28.3.2 comp_handle_t

COMP驱动的句柄结构体comp_handle_t的定义如下：

表 2-501 comp_handle_t结构体

数据域	域段描述	取值
comp_init_t init	初始化结构体	参考comp_init_t结构体。
__IO hal_lock_t lock	COMP锁（无需开发者初始化）	N/A
__IO hal_comp_state_t state	COMP状态	该参数的取值可以是下面中的任意一个： • HAL_COMP_STATE_RESET（未初始化） • HAL_COMP_STATE_READY（已初始化且空闲） • HAL_COMP_STATE_BUSY（忙） • HAL_COMP_STATE_ERROR（错误）
__IO uint32_t error_code	COMP错误码	该参数的取值可以是下面中的任意一个： • HAL_COMP_ERROR_NONE • HAL_COMP_ERROR_TIMEOUT • HAL_COMP_ERROR_INVALID_PARAM

数据域	域段描述	取值
uint32_t retention[1]	保存COMP寄存器信息（驱动负责管理，无需开发者初始化）	N/A

2.28.4 COMP驱动API描述

COMP驱动的API主要如下：

表 2-502 COMP驱动的APIs

API类别	API名称	描述
初始化	hal_comp_init()	初始化COMP模块，配置输入源、参考源、参考值等参数，同时调用hal_comp_msp_init完成中断相关配置，初始化COMP状态以及错误码。
	hal_comp_deinit()	重置相关COMP相关寄存器、COMP状态、错误码等参数。
	hal_comp_msp_init()	初始化COMP模块所使用的NVIC中断。
	hal_comp_msp_deinit()	反初始化COMP模块所使用的NVIC中断。
IO操作	hal_comp_start()	开始比较器功能。
	hal_comp_stop()	停止比较器功能。
中断处理及回调函数	hal_comp_irq_handler()	中断处理函数。
	hal_comp_trigger_callback()	中断回调函数。
状态及错误	hal_comp_get_state()	获取驱动运行状态。
	hal_comp_get_error()	获取错误码。
睡眠相关	hal_comp_suspend_reg()	睡眠之前挂起和COMP配置相关的寄存器。
	hal_comp_resume_reg()	唤醒时恢复和COMP配置相关的寄存器。

下面章节将对各API进行详细描述。

2.28.4.1 hal_comp_init

表 2-503 hal_comp_init接口

函数原型	hal_status_t hal_comp_init(comp_handle_t *p_comp)
功能说明	根据COMP句柄中的指定配置参数初始化COMP模块。
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	HAL状态
备注	

2.28.4.2 hal_comp_deinit

表 2-504 hal_comp_deinit接口

函数原型	hal_status_t hal_comp_deinit(comp_handle_t *p_comp);
功能说明	将COMP相关寄存器初始化为它们的默认重置值，同时重置比较器的状态和错误码。
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	HAL状态。
备注	

2.28.4.3 hal_comp_msp_init

表 2-505 hal_comp_msp_init接口

函数原型	void hal_comp_msp_init(comp_handle_t *p_comp)
功能说明	初始化COMP模块所使用的NVIC中断。
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需要重写该API以完成相应的功能。

2.28.4.4 hal_comp_msp_deinit

表 2-506 hal_comp_msp_deinit接口

函数原型	void hal_comp_msp_deinit(comp_handle_t *p_comp)
功能说明	反初始化COMP模块所使用的NVIC中断。
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，开发者需要重写该API以完成相应的功能。

2.28.4.5 hal_comp_start

表 2-507 hal_comp_start接口

函数原型	hal_status_t hal_comp_start(comp_handle_t *p_comp)
功能说明	开始比较器功能，当输入电压高于参考电压，产生COMP中断。
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	HAL状态。

备注	

2.28.4.6 hal_comp_stop

表 2-508 hal_comp_stop接口

函数原型	hal_status_t hal_comp_stop(comp_handle_t *p_comp)
功能说明	停止比较器功能。
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	HAL状态。
备注	

2.28.4.7 hal_comp_irq_handler

表 2-509 hal_comp_irq_handler接口

函数原型	void hal_comp_irq_handler(comp_handle_t *p_comp)
功能说明	处理COMP中断请求。
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	无
备注	

2.28.4.8 hal_comp_trigger_callback

表 2-510 hal_comp_trigger_callback接口

函数原型	void hal_comp_trigger_callback(comp_handle_t *p_comp)
功能说明	中断回调函数。
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	无
备注	该函数为weak类型的空函数，当开发者需要使用该回调函数时，可重写该API。

2.28.4.9 hal_comp_get_state

表 2-511 hal_comp_get_state接口

函数原型	hal_comp_state_t hal_comp_get_state(comp_handle_t *p_comp)
功能说明	获取比较器运行状态。

输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	COMP状态，值可以是下面中的任意一个： • HAL_COMP_STATE_RESET（未初始化） • HAL_COMP_STATE_READY（已初始化且空闲） • HAL_COMP_STATE_BUSY（忙） • HAL_COMP_STATE_ERROR（错误）
备注	

2.28.4.10 hal_comp_get_error

表 2-512 hal_comp_get_error接口

函数原型	uint32_t hal_comp_get_error(comp_handle_t *p_comp)
功能说明	获取比较器错误码。
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	COMP错误码，值可以是下面中的任意一个： • HAL_COMP_ERROR_NONE • HAL_COMP_ERROR_TIMEOUT • HAL_COMP_ERROR_INVALID_PARAM
备注	

2.28.4.11 hal_comp_suspend_reg

表 2-513 hal_comp_suspend_reg接口

函数原型	hal_status_t hal_comp_suspend_reg(comp_handle_t *p_comp);
功能说明	睡眠之前挂起和COMP配置相关的寄存器
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。
返回值	HAL状态
备注	

2.28.4.12 hal_comp_resume_reg

表 2-514 hal_comp_resume_reg接口

函数原型	hal_status_t hal_comp_resume_reg(comp_handle_t *p_comp);
功能说明	唤醒时恢复和COMP配置相关的寄存器
输入参数	p_comp：指向comp_handle_t结构体变量的指针，该结构体变量包含指定的COMP模块的配置信息。

返回值	HAL状态
备注	

3 LL驱动

3.1 简介

本节将对LL驱动中各个外设模块所使用的公共资源及如何使用LL驱动进行简单介绍。

说明:

此文档主要介绍LL层的初始化类API。更多API信息，请参考《GR551x API Reference》。

3.1.1 LL公共资源

在GR551x的LL驱动中，各个外设模块使用的公共枚举、结构体、宏定义在*gr55xx.h*中，主要如下：

1. 标志位状态/中断状态：用于表示相应的标志位或中断标志位是否置1，定义如下：

```
typedef enum
{
    RESET = 0,
    SET = !RESET
} flag_status, it_status;
```

2. 功能状态：用于表示相应的功能是否启用，其定义如下：

```
typedef enum
{
    DISABLE = 0,
    ENABLE = !DISABLE
} functional_state;
```

3. 公共宏：主要为寄存器相关的宏，可用于实现寄存器的直接读写及按位读写，定义如下：

```
#define SET_BIT(REG, BIT)       ((REG) |= (BIT))
#define CLEAR_BIT(REG, BIT)     ((REG) &= ~(BIT))
#define READ_BIT(REG, BIT)      ((REG) & (BIT))

#define CLEAR_REG(REG)           ((REG) = (0x0))
#define WRITE_REG(REG, VAL)     ((REG) = (VAL))
#define READ_REG(REG)           ((REG))
#define  MODIFY_REG(REG,CLEARMASK,SETMASK)  WRITE_REG((REG),(((READ_REG(REG))&(~(CLEARMASK)))|
(SETMASK)))

#define POSITION_VAL(VAL)       (__CLZ(__RBIT(VAL)))
```

3.1.2 如何使用LL驱动

LL驱动提供了各外设各个寄存器操作的API接口，其使用方法如下：

1. 如果外设的LL驱动提供初始化接口ll_ppp_init()，则先调用该接口对外设进行初始化配置。
2. 根据需要的功能调用相应的API接口，完成相应的操作。

3. 如果外设的LL驱动提供反初始化接口ll_ppp_deinit()，则可调用该接口对外设进行反初始化。

3.2 LL GPIO通用驱动

3.2.1 GPIO驱动的结构体

3.2.1.1 ll_gpio_init_t

GPIO外设LL层初始化结构体ll_gpio_init_t的定义如下：

表 3-1 ll_gpio_init_t结构体

数据域	域段描述	取值
uint32_t pin	指定要配置的GPIO引脚。	该参数的取值可以是下列值的组合： • LL_GPIO_PIN_0（引脚0） • LL_GPIO_PIN_1（引脚1） • LL_GPIO_PIN_2（引脚2） • LL_GPIO_PIN_3（引脚3） • LL_GPIO_PIN_4（引脚4） • LL_GPIO_PIN_5（引脚5） • LL_GPIO_PIN_6（引脚6） • LL_GPIO_PIN_7（引脚7） • LL_GPIO_PIN_8（引脚8） • LL_GPIO_PIN_9（引脚9） • LL_GPIO_PIN_10（引脚10） • LL_GPIO_PIN_11（引脚11） • LL_GPIO_PIN_12（引脚12） • LL_GPIO_PIN_13（引脚13） • LL_GPIO_PIN_14（引脚14） • LL_GPIO_PIN_15（引脚15） • LL_GPIO_PIN_ALL（所有引脚）
uint32_t mode	指定所选引脚的操作模式，开发者也可通过ll_gpio_set_pin_mode()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_GPIO_MODE_INPUT （输入模式） • LL_GPIO_MODE_OUTPUT（输出模式） • LL_GPIO_MODE_MUX（复用模式）
uint32_t pull	指定所选引脚上拉或下拉电阻类型，开发者也可通过ll_gpio_set_pin_pull()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_GPIO_PULL_NO（没有上拉或下拉电阻激活） • LL_GPIO_PULL_UP（激活上拉电阻） • LL_GPIO_PULL_DOWN（激活下拉电阻）
uint32_t mux	指定所选引脚的复用功能模式，开发者也可通过ll_gpio_set_mux_pin_0_7()以及接	该参数的取值可以是下列值中的任意一个： • LL_GPIO_MUX_0（复用模式0） • LL_GPIO_MUX_1（复用模式1） • LL_GPIO_MUX_2（复用模式2） • LL_GPIO_MUX_3（复用模式3）

数据域	域段描述	取值
	口ll_gpio_set_mux_pin_8_15()设置该参数。	• LL_GPIO_MUX_4（复用模式4） • LL_GPIO_MUX_5（复用模式5） • LL_GPIO_MUX_6（复用模式6） • LL_GPIO_MUX_7（复用模式7）
uin32_t trigger	指定所选引脚的中断触发类型，开发者也可通过ll_gpio_enable_falling_trig()、ll_gpio_enable_rising_trig()、ll_gpio_enable_high_trig()、ll_gpio_enable_low_trig()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_AON_GPIO_TRIGGER_NONE（无中断触发） • LL_AON_GPIO_TRIGGER_RISING（上升沿触发） • LL_AON_GPIO_TRIGGER_FALLING（下降沿触发） • LL_AON_GPIO_TRIGGER_HIGH（高电平触发） • LL_AON_GPIO_TRIGGER_LOW（低电平触发）

3.2.2 GPIO驱动API描述

GPIO驱动的API主要包括：

表 3-2 GPIO驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_gpio_init()	初始化GPIO外设。
	ll_gpio_deinit()	反初始化GPIO外设，恢复初始设置。
	ll_gpio_struct_init()	初始化结构体变量ll_gpio_init_t为默认值。

下面章节将对各API进行详细描述。

3.2.2.1 ll_gpio_init

表 3-3 ll_gpio_init接口

函数原型	error_status_t ll_gpio_init(gpio_regs_t *GPIOx, ll_gpio_init_t *p_gpio_init)
功能说明	根据ll_gpio_init_t指定参数初始化GPIO外设。
输入参数	GPIOx：x可以是0或1，x用于确定GR551x家族中被操作的GPIO端口。 p_gpio_init：指向ll_gpio_init_t结构体变量的指针，该结构体变量包含指定的GPIO外设实例的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：GPIO外设寄存器已被成功初始化 • ERROR：不可用
备注	

3.2.2.2 ll_gpio_deinit

表 3-4 ll_gpio_deinit接口

函数原型	error_status_t ll_gpio_deinit(gpio_regs_t *GPIOx)
功能说明	将GPIO外设寄存器初始化为它们的默认重置值。
输入参数	GPIOx：x可以是0或1，x用于确定GR551x家族中被操作的GPIO端口。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：GPIO外设寄存器已被成功反初始化 • ERROR：GPIOx变量错误
备注	

3.2.2.3 ll_gpio_struct_init

表 3-5 ll_gpio_struct_init接口

函数原型	void ll_gpio_struct_init(ll_gpio_init_t *p_gpio_init)
功能说明	将ll_gpio_init_t结构体变量初始化为默认值。
输入参数	p_gpio_init：指向待重置的结构体变量的指针。
返回值	无
备注	

3.3 LL AON GPIO通用驱动

3.3.1 AON GPIO驱动的结构体

3.3.1.1 ll_aon_gpio_init_t

AON GPIO外设LL层初始化结构体ll_aon_gpio_init_t的定义如下：

表 3-6 ll_aon_gpio_init_t结构体

数据域	域段描述	取值
uint32_t pin	要配置的AON GPIO引脚。	该参数的取值可以是下面中的值的组合： • LL_AON_GPIO_PIN_0（引脚0） • LL_AON_GPIO_PIN_1（引脚1） • LL_AON_GPIO_PIN_2（引脚2） • LL_AON_GPIO_PIN_3（引脚3） • LL_AON_GPIO_PIN_4（引脚4） • LL_AON_GPIO_PIN_5（引脚5） • LL_AON_GPIO_PIN_6（引脚6） • LL_AON_GPIO_PIN_7（引脚7） • LL_AON_GPIO_PIN_ALL（所有引脚）

数据域	域段描述	取值
uint32_t mode	指定所选引脚的操作模式，开发者也可通过ll_aon_gpio_set_pin_mode()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_AON_GPIO_MODE_INPUT（输入模式） • LL_AON_GPIO_MODE_OUTPUT（输出模式） • LL_AON_GPIO_MODE_MUX（复用模式）
uint32_t pull	所选引脚上拉或下拉电阻类型，开发者也可通过ll_aon_gpio_set_pin_pull()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_AON_GPIO_PULL_NO（没有上拉或下拉电阻激活） • LL_AON_GPIO_PULL_UP（激活上拉电阻） • LL_AON_GPIO_PULL_DOWN（激活下拉电阻）
uin32_t trigger	所选引脚的中断触发类型，开发者也可通过ll_aon_gpio_enable_falling_trig()、ll_aon_gpio_enable_rising_trig()、ll_aon_gpio_enable_high_trig()、ll_aon_gpio_enable_low_trig()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_AON_GPIO_TRIGGER_NONE（无中断触发） • LL_AON_GPIO_TRIGGER_RISING（上升沿触发） • LL_AON_GPIO_TRIGGER_FALLING（下降沿触发） • LL_AON_GPIO_TRIGGER_HIGH（高电平触发） • LL_AON_GPIO_TRIGGER_LOW（低电平触发）

3.3.2 AON GPIO驱动API描述

AON GPIO驱动的API主要包括：

表 3-7 AON GPIO驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_aon_gpio_init()	初始化AON GPIO外设。
	ll_aon_gpio_deinit()	反初始化AON GPIO外设，恢复初始设置。
	ll_aon_gpio_struct_init()	初始化结构体aon_gpio_init为默认值。

下面章节将对各API进行详细描述。

3.3.2.1 ll_aon_gpio_init

表 3-8 ll_aon_gpio_init接口

函数原型	error_status ll_aon_gpio_init(ll_aon_gpio_init_t *p_aon_gpio_init)
功能说明	根据ll_aon_gpio_init_t指定参数初始化AON GPIO外设。
输入参数	p_aon_gpio_init：指向ll_aon_gpio_init_t结构体变量的指针，该结构体变量包含指定的AON GPIO引脚的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：GPIO外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.3.2.2 ll_aon_gpio_deinit

表 3-9 ll_aon_gpio_deinit接口

函数原型	error_status_t ll_aon_gpio_deinit(void)
功能说明	将GPIO外设寄存器反初始化为它们的默认重置值。
输入参数	无
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：AON GPIO外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.3.2.3 ll_aon_gpio_struct_init

表 3-10 ll_aon_gpio_struct_init接口

函数原型	void ll_aon_gpio_struct_init(ll_aon_gpio_init_t *p_aon_gpio_init)
功能说明	将ll_aon_gpio_init_t结构体变量初始化为默认重置值。
输入参数	p_aon_gpio_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.4 LL ADC通用驱动

3.4.1 ADC驱动的结构体

3.4.1.1 ll_adc_init_t

ADC外设LL层初始化结构体ll_adc_init_t的定义如下：

表 3-11 ll_adc_init_t结构体

数据域	域段描述	取值
uint32_t channel_p	通道P的输入源。	该参数的取值可以是下列值中的任意一个： • LL_ADC_INPUT_SRC_IO0（MSIO0输入） • LL_ADC_INPUT_SRC_IO1（MSIO1输入） • LL_ADC_INPUT_SRC_IO2（MSIO2输入） • LL_ADC_INPUT_SRC_IO3（MSIO3输入） • LL_ADC_INPUT_SRC_IO4（MSIO4输入） • LL_ADC_INPUT_SRC_TMP（温度传感器输入） • LL_ADC_INPUT_SRC_BAT（电池电压输入） • LL_ADC_INPUT_SRC_REF（参考电压输入）
uint32_t channel_n	通道N的输入源。	该参数的取值可以是下列值中的任意一个： • LL_ADC_INPUT_SRC_IO0（MSIO0输入）

数据域	域段描述	取值
		• LL_ADC_INPUT_SRC_IO1（MSIO1输入） • LL_ADC_INPUT_SRC_IO2（MSIO2输入） • LL_ADC_INPUT_SRC_IO3（MSIO3输入） • LL_ADC_INPUT_SRC_IO4（MSIO4输入） • LL_ADC_INPUT_SRC_TMP（温度传感器输入） • LL_ADC_INPUT_SRC_BAT（电池电压输入） • LL_ADC_INPUT_SRC_REF（参考电压输入）
uint32_t input_mode	采样方式。	该参数的取值可以是下列值中的任意一个： • LL_ADC_INPUT_SINGLE（单端输入采样） • LL_ADC_INPUT_DIFFERENTIAL（差分输入采样）
uin32_t ref_source	参考源类型。	该参数的取值可以是下列值中的任意一个： • LL_ADC_REF_SRC_BUF_INT（内部Buffered参考源） • LL_ADC_REF_SRC_IO0（MSIO0输入电压） • LL_ADC_REF_SRC_IO1（MSIO1输入电压） • LL_ADC_REF_SRC_IO2（MSIO2输入电压） • LL_ADC_REF_SRC_IO3（MSIO3输入电压）
uin32_t ref_value	内部参考电压。	该参数的取值可以是下列值中的任意一个： • LL_ADC_REF_VALUE_0P8（0.85 V） • LL_ADC_REF_VALUE_1P2（1.28 V） • LL_ADC_REF_VALUE_1P6（1.6 V） **说明：** 外部输入信号的量程是0 ～ 2*ref_value，用户可按照实际的使用场景进行参数配置
uin32_t clock	采样时钟。	该参数的取值可以是下列值中的任意一个： • LL_ADC_CLK_16M（16 M时钟） • LL_ADC_CLK_1P6M（1.6 M时钟） • LL_ADC_CLK_8M（8 M时钟） • LL_ADC_CLK_4M（4 M时钟） • LL_ADC_CLK_2M（2 M时钟） • LL_ADC_CLK_1M（1 M时钟）

3.4.2 ADC驱动API描述

ADC驱动的API主要包括：

表 3-12 ADC驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_adc_init()	初始化ADC外设。
	ll_adc_deinit()	反初始化ADC外设，恢复初始设置。
	ll_adc_struct_init()	初始化结构体ll_adc_init_t为默认值。

下面章节将对各API进行详细描述。

3.4.2.1 ll_adc_init

表 3-13 ll_adc_init接口

函数原型	error_status_t ll_adc_init(ll_adc_init_t *p_adc_init)
功能说明	根据ll_adc_init_t指定参数初始化ADC外设。
输入参数	p_adc_init：指向ll_adc_init_t结构体变量的指针，该结构体变量包含指定的ADC引脚的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：ADC外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.4.2.2 ll_adc_deinit

表 3-14 ll_adc_deinit接口

函数原型	error_status_t ll_adc_deinit(void)
功能说明	将ADC外设寄存器反初始化为它们的默认值。
输入参数	无
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：ADC外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.4.2.3 ll_adc_struct_init

表 3-15 ll_adc_struct_init接口

函数原型	void ll_adc_struct_init(ll_adc_init_t *p_adc_init)
功能说明	将ll_adc_init_t结构体变量初始化为默认值。
输入参数	p_adc_init：指向待重置的结构体变量的指针。
返回值	无
备注	

3.5 LL DMA通用驱动

3.5.1 DMA驱动的结构体

3.5.1.1 ll_dma_init_t

DMA外设LL层初始化结构体ll_dma_init_t的定义如下：

表 3-16 ll_dma_init_t结构体

数据域	域段描述	取值
uint32_t src_address	源地址，开发者也可通过ll_dma_set_source_address()设置该参数。	0x0000_0000 ～ 0xFFFF_FFFF
uint32_t dst_address	目的地址，开发者也可通过ll_dma_set_destination_address()设置该参数。	0x0000_0000 ～ 0xFFFF_FFFF
uint32_t direction	传输方向，开发者也可通过ll_dma_set_data_transfer_direction()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DMA_DIRECTION_MEMORY_TO_MEMORY（内存到内存） • LL_DMA_DIRECTION_MEMORY_TO_PERIPH（内存到外设） • LL_DMA_DIRECTION_PERIPH_TO_MEMORY（外设到内存） • LL_DMA_DIRECTION_PERIPH_TO_PERIPH（外设到外设）
uint32_t mode	传输模式，开发者也可在初始化后通过ll_dma_set_mode()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DMA_MODE_SINGLE_BLOCK（单块传输） • LL_DMA_MODE_MULTI_BLOCK_SRC_RELOAD（多块传输、源地址自动重载） • LL_DMA_MODE_MULTI_BLOCK_DST_RELOAD（多块传输、目的地址自动重载） • LL_DMA_MODE_MULTI_BLOCK_ALL_RELOAD（多块传输、源地址及目的地址均自动重载）
uint32_t src_increment_mode	源地址增量模式，开发者也可在初始化后通过ll_dma_set_mode()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DMA_SRC_INCREMENT（源地址递增） • LL_DMA_SRC_DECREMENT（源地址递减） • LL_DMA_SRC_NO_CHANGE（源地址不变）
uint32_t dst_increment_mode	目的地址增量模式，开发者也可在初始化后通过ll_dma_set_mode()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DMA_DST_INCREMENT（目的地址递增） • LL_DMA_DST_DECREMENT（目的地址递减） • LL_DMA_DST_NO_CHANGE（目的地址不变）

数据域	域段描述	取值
uint32_t src_data_width	源数据突发传输宽度，开发者也可在初始化后通过ll_dma_set_mode()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DMA_SRC_BURST_LENGTH_1（单字节） • LL_DMA_SRC_BURST_LENGTH_4（4字节） • LL_DMA_SRC_BURST_LENGTH_8（8字节）
uint32_t dst_data_width	目的数据突发传输宽度，开发者也可在初始化后通过ll_dma_set_mode()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DMA_DST_BURST_LENGTH_1（单字节） • LL_DMA_DST_BURST_LENGTH_4（4字节） • LL_DMA_DST_BURST_LENGTH_8（8字节）
uint32_t block_size	数据传输长度，开发者也可在初始化后通过ll_dma_set_block_size()设置该参数。	0～4095
uint32_t src_peripheral	源外设，开发者也可在初始化后通过ll_dma_set_source_peripheral()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DMA_PERIPH_SPIM_TX（SPIM发送） • LL_DMA_PERIPH_SPIM_RX（SPIM接收） • LL_DMA_PERIPH_SPIS_TX（SPIS发送） • LL_DMA_PERIPH_SPIS_RX（SPIS接收） • LL_DMA_PERIPH_QSPI0_TX（QSPI0发送） • LL_DMA_PERIPH_QSPI0_RX（QSPI0接收） • LL_DMA_PERIPH_I2C0_TX（I2C0发送） • LL_DMA_PERIPH_I2C0_RX（I2C0接收） • LL_DMA_PERIPH_I2C1_TX（I2C1发送） • LL_DMA_PERIPH_I2C1_RX（I2C1接收） • LL_DMA_PERIPH_I2S_S_TX（I2SS发送） • LL_DMA_PERIPH_I2S_S_RX（I2SS接收） • LL_DMA_PERIPH_UART0_TX（UART0发送） • LL_DMA_PERIPH_UART0_RX（UART0接收） • LL_DMA_PERIPH_QSPI1_TX（QSPI1发送） • LL_DMA_PERIPH_QSPI1_RX（QSPI1接收） • LL_DMA_PERIPH_I2S_M_TX（I2SM发送） • LL_DMA_PERIPH_I2S_M_RX（I2SM接收） • LL_DMA_PERIPH_SNSADC（Sense ADC） • LL_DMA_PERIPH_MEM（内存）
uint32_t dst_peripheral	目的外设，开发者也可在初始化后通过ll_dma_set_destination_peripheral()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DMA_PERIPH_SPIM_TX（SPIM发送） • LL_DMA_PERIPH_SPIM_RX（SPIM接收） • LL_DMA_PERIPH_SPIS_TX（SPIS发送） • LL_DMA_PERIPH_SPIS_RX（SPIS接收） • LL_DMA_PERIPH_QSPI0_TX（QSPI0发送） • LL_DMA_PERIPH_QSPI0_RX（QSPI0接收） • LL_DMA_PERIPH_I2C0_TX（I2C0发送） • LL_DMA_PERIPH_I2C0_RX（I2C0接收） • LL_DMA_PERIPH_I2C1_TX（I2C1发送） • LL_DMA_PERIPH_I2C1_RX（I2C1接收）

数据域	域段描述	取值
		• LL_DMA_PERIPH_UART0_TX（UART0发送） • LL_DMA_PERIPH_UART0_RX（UART0接收） • LL_DMA_PERIPH_QSPI1_TX（QSPI1发送） • LL_DMA_PERIPH_QSPI1_RX（QSPI1接收） • LL_DMA_PERIPH_SNSADC（Sense ADC） • LL_DMA_PERIPH_MEM（内存）
uint32_t priority	通道优先级，开发者也可在初始化后通过ll_dma_set_channel_priority_level()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DMA_PRIORITY_0（优先级0，最低） • LL_DMA_PRIORITY_1（优先级1） • LL_DMA_PRIORITY_2（优先级2） • LL_DMA_PRIORITY_3（优先级3） • LL_DMA_PRIORITY_4（优先级4） • LL_DMA_PRIORITY_5（优先级5） • LL_DMA_PRIORITY_6（优先级6） • LL_DMA_PRIORITY_7（优先级7，最高）

3.5.2 DMA驱动API描述

DMA驱动的API主要包括：

表 3-17 DMA驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_dma_init()	初始化DMA的指定通道。
	ll_dma_deinit()	反初始化DMA外设，恢复初始设置。
	ll_dma_struct_init()	初始化结构体dma_init为默认值。

下面章节将对各API进行详细描述。

3.5.2.1 ll_dma_init

表 3-18 ll_dma_init接口

函数原型	error_status_t ll_dma_init(dma_regs_t *DMAx, uint32_t channel, ll_dma_init_t *p_dma_init)
功能说明	根据ll_dma_init_t指定参数初始化DMA的指定通道。
输入参数	DMAx：DMA外设实例。 channel：指定需要初始化的DMA通道，该参数可以是下列值中的任意一个： • LL_DMA_CHANNEL_0 • LL_DMA_CHANNEL_1 • LL_DMA_CHANNEL_2 • LL_DMA_CHANNEL_3 • LL_DMA_CHANNEL_4 • LL_DMA_CHANNEL_5 • LL_DMA_CHANNEL_6

	• LL_DMA_CHANNEL_7 p_dma_init：指向ll_dma_init_t结构体变量的指针，该结构体变量包含指定的DMA通道的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：DMA外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.5.2.2 ll_dma_deinit

表 3-19 ll_dma_deinit接口

函数原型	error_status_t ll_dma_deinit(dma_regs_t *DMAx, uint32_t channel)
功能说明	将DMA指定通道的寄存器反初始化为它们的默认重置值。
输入参数	DMAx：指向DMA外设实例的指针。 channel：指定需要初始化的DMA通道，该参数可以是下列值中的任意一个： • LL_DMA_CHANNEL_0 • LL_DMA_CHANNEL_1 • LL_DMA_CHANNEL_2 • LL_DMA_CHANNEL_3 • LL_DMA_CHANNEL_4 • LL_DMA_CHANNEL_5 • LL_DMA_CHANNEL_6 • LL_DMA_CHANNEL_7
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：DMA外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.5.2.3 ll_dma_struct_init

表 3-20 ll_dma_struct_init接口

函数原型	void ll_dma_struct_init(ll_dma_init_t *p_dma_init)
功能说明	将ll_dma_init_t结构体变量初始化为默认重置值。
输入参数	p_dma_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.6 LL DUAL TIMER通用驱动

3.6.1 DUAL TIMER驱动的结构体

3.6.1.1 ll_dual_timer_init_t

DUAL TIMER外设LL层初始化结构体ll_dual_timer_init_t的定义如下：

表 3-21 ll_dual_timer_init_t结构体

数据域	域段描述	取值
uint32_t prescaler	分频系数，开发者也可通过ll_dual_timer_set_prescaler()设置该参数。	参数的取值可以是下列值中的任意一个： • LL_DUAL_TIMER_PRESCALER_DIV0（无分频） • LL_DUAL_TIMER_PRESCALER_DIV16（16分频） • LL_DUAL_TIMER_PRESCALER_DIV256（256分频）
uint32_t counter_size	计数器位宽，开发者也可通过ll_dual_timer_set_counter_size()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DUAL_TIMER_COUNTERSIZE_16（16位） • LL_DUAL_TIMER_COUNTERSIZE_32（32位）
uint32_t counter_mode	计数模式，开发者也可通过ll_dual_timer_set_counter_mode()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_DUAL_TIMER_FREERUNNING_MODE（自由计数模式） • LL_DUAL_TIMER_PERIODIC_MODE（周期模式）
uint32_t auto_reload	计数初值，开发者也可通过ll_dual_timer_set_auto_reload()设置该参数。	0x0000_0000 ～ 0xFFFF_FFFF

3.6.2 DUAL TIMER驱动API描述

DUAL TIMER驱动的API主要包括：

表 3-22 DUAL TIMER驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_dual_timer_init()	初始化DUAL TIMER外设。
	ll_dual_timer_deinit()	反初始化DUAL TIMER外设，恢复初始设置。
	ll_dual_timer_struct_init()	初始化结构体dual_timer_init为默认值。

下面章节将对各API进行详细描述。

3.6.2.1 ll_dual_timer_init

表 3-23 ll_dual_timer_init接口

函数原型	error_status_t ll_dual_timer_init(dual_timer_regs_t *DUAL_TIMERx, ll_dual_timer_init_t *p_dual_timer_init)
功能说明	根据ll_dual_timer_init_t指定参数初始化DUAL TIMER外设。
输入参数	DUAL_TIMERx：DUAL TIMER外设实例。 p_dual_timer_init：指向ll_dual_timer_init_t结构体变量的指针，该结构体变量包含指定的DUAL TIMER的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：DUAL TIMER外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.6.2.2 ll_dual_timer_deinit

表 3-24 ll_dual_timer_deinit接口

函数原型	error_status_t ll_dual_timer_deinit(dual_timer_regs_t *DUAL_TIMERx)
功能说明	将DUAL_TIMERx外设的寄存器反初始化为它们的默认重置值。
输入参数	DUAL_TIMERx：DUAL TIMER外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：DUAL TIMER外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.6.2.3 ll_dual_timer_struct_init

表 3-25 ll_dual_timer_struct_init接口

函数原型	void ll_dual_timer_struct_init(ll_dual_timer_init_t *p_dual_timer_init)
功能说明	将ll_dual_timer_init_t结构体变量初始化为默认重置值。
输入参数	p_dual_timer_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.7 LL I2C通用驱动

3.7.1 I2C驱动的结构体

3.7.1.1 ll_i2c_init_t

I2C外设LL层初始化结构体ll_i2c_init_t的定义如下：

表 3-26 ll_i2c_init_t结构体

数据域	域段描述	取值
uint32_t speed	传输速率，开发者也可通过ll_i2c_configure_speed()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_I2C_SPEED_100K（100 Kbps） • LL_I2C_SPEED_400K（400 Kbps） • LL_I2C_SPEED_1000K（1.0 Mbps） • LL_I2C_SPEED_2000K（2.0 Mbps）
uint32_t own_address	本机设备地址（用于从设备模式），开发者也可通过ll_i2c_set_own_address()设置该参数。	7位地址： 0x08 ～ 0x77 10位地址：0x008 ～ 0x077, 0x080 ～ 0x3FE
uint32_t own_addr_size	本机设备地址格式（用于从设备模式），开发者也可通过ll_i2c_set_own_address()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_I2C_OWNADDRESS_7BIT（7位地址） • LL_I2C_OWNADDRESS_10BIT（10位地址）

3.7.2 I2C驱动API描述

I2C驱动的API主要包括：

表 3-27 I2C驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_i2c_init()	初始化I2C外设。
	ll_i2c_deinit()	反初始化I2C外设，恢复初始设置。
	ll_i2c_struct_init()	初始化结构体i2c_init为默认值。

下面章节将对各API进行详细描述。

3.7.2.1 ll_i2c_init

表 3-28 ll_i2c_init接口

函数原型	error_status_t ll_i2c_init(i2c_regs_t *I2Cx, ll_i2c_init_t *p_i2c_init)
功能说明	根据ll_i2c_init_t指定参数初始化I2C外设。
输入参数	I2Cx：I2C外设实例。

	p_i2c_init：指向ll_i2c_init_t结构体变量的指针，该结构体变量包含指定的I2C外设实例的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：I2C外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.7.2.2 ll_i2c_deinit

表 3-29 ll_i2c_deinit接口

函数原型	error_status_t ll_i2c_deinit(i2c_regs_t *I2Cx)
功能说明	将I2C外设寄存器反初始化为它们的默认重置值。
输入参数	I2Cx：I2C外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：I2C外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.7.2.3 ll_i2c_struct_init

表 3-30 ll_i2c_struct_init接口

函数原型	void ll_i2c_struct_init(ll_i2c_init_t *p_i2c_init)
功能说明	将ll_i2c_init_t结构体变量初始化为默认重置值。
输入参数	p_i2c_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.8 LL MSIO通用驱动

3.8.1 MSIO驱动的结构体

3.8.1.1 ll_msio_init_t

MSIO外设LL层初始化结构体ll_msio_init_t的定义如下：

表 3-31 ll_msio_init_t结构体

数据域	域段描述	取值
uint32_t pin	要配置的MSIO引脚。	该参数的取值可以是下面中的值的组合： • LL_MSIO_PIN_0（引脚0） • LL_MSIO_PIN_1（引脚1）

数据域	域段描述	取值
		• LL_MSIO_PIN_2（引脚2） • LL_MSIO_PIN_3（引脚3） • LL_MSIO_PIN_4（引脚4） • LL_MSIO_PIN_ALL（所有引脚）
uint32_t direction	所选引脚的方向（输入/输出），开发者也可通过ll_msio_set_pin_direction()设置该参数。	该参数的取值可以是下面的值： • LL_MSIO_DIRECTION_NONE（禁止输入输出） • LL_MSIO_DIRECTION_INPUT（使能输入） • LL_MSIO_DIRECTION_OUTPUT（使能输出） • LL_MSIO_DIRECTION_INOUT（使能输入输出）
uint32_t mode	指定所选引脚的操作模式，开发者也可通过ll_msio_set_pin_mode()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_MSIO_MODE_ANALOG（模拟模式） • LL_MSIO_MODE_DIGITAL（数字模式）
uint32_t pull	所选引脚上拉或下拉电阻类型，开发者也可通过ll_msio_set_pin_pull()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_MSIO_PULL_NO（没有上拉或下拉电阻激活） • LL_MSIO_PULL_UP（激活上拉电阻） • LL_MSIO_PULL_DOWN（激活下拉电阻）

3.8.2 MSIO驱动API描述

MSIO驱动的API主要包括：

表 3-32 MSIO驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_msio_init()	初始化MSIO外设。
	ll_msio_deinit()	反初始化MSIO外设，恢复初始设置。
	ll_msio_struct_init()	初始化结构体msio_init为默认值。

下面章节将对各API进行详细描述。

3.8.2.1 ll_msio_init

表 3-33 ll_msio_init接口

函数原型	error_status_t ll_msio_init(ll_msio_init_t *p_msio_init)
功能说明	根据ll_msio_init_t指定参数初始化MSIO外设。
输入参数	p_msio_init：指向ll_msio_init_t结构体变量的指针，该结构体变量包含指定的MSIO引脚的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：MSIO外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.8.2.2 ll_msio_deinit

表 3-34 ll_msio_deinit接口

函数原型	error_status_t ll_msio_deinit(void)
功能说明	将MSIO外设寄存器反初始化为它们的默认重置值。
输入参数	无
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：MSIO外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.8.2.3 ll_msio_struct_init

表 3-35 ll_msio_struct_init接口

函数原型	void ll_msio_struct_init(ll_msio_init_t *p_msio_init)
功能说明	将ll_msio_init_t结构体变量初始化为默认重置值。
输入参数	p_msio_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.9 LL AES通用驱动

3.9.1 AES驱动的结构体

3.9.1.1 ll_aes_init_t

AES结构体ll_aes_init_t的定义如下：

表 3-36 ll_aes_init_t结构体

数据域	域段描述	取值
uint32_t key_size	AES KEY长度。	该参数的取值可以是下列值中的任意一个： • LL_AES_KEY_SIZE_128（128 bits） • LL_AES_KEY_SIZE_192（192 bits） • LL_AES_KEY_SIZE_256（256 bits）
uint32_t *p_key	AES密钥。	该参数的取值由开发者指定。
uint32_t *p_init_vector	用于CBC模式的初始化向量。	该参数的取值由开发者指定。
uint32_t *p_key	随机种子。	该参数的取值由开发者指定。

3.9.2 AES驱动API描述

AES驱动的API主要包括：

表 3-37 AES驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_aes_init()	初始化AES外设。
	ll_aes_deinit()	反初始化AES外设，恢复初始设置。
	ll_aes_struct_init(ll_aes_init_t *aes_init)	初始化结构体aes_init为默认值。

下面章节将对各API进行详细描述。

3.9.2.1 ll_aes_init

表 3-38 ll_aes_init接口

函数原型	error_status_t ll_aes_init(aes_regs_t *AESx, ll_aes_init_t *p_aes_init)
功能说明	初始化AES外设，使能外设及小端模式。
输入参数	AESx：AES外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：AES外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.9.2.2 ll_aes_deinit

表 3-39 ll_aes_deinit接口

函数原型	error_status_t ll_aes_deinit(aes_regs_t *AESx)
功能说明	将AES外设寄存器反初始化为它们的默认值。
输入参数	AESx：AES外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：AES外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.9.2.3 ll_aes_struct_init

表 3-40 ll_aes_struct_init接口

函数原型	void ll_aes_struct_init(ll_aes_init_t *p_aes_init)
功能说明	将ll_aes_init_t结构体变量初始化为默认值。
输入参数	p_aes_init：指向待重置的结构体变量的指针。
返回值	无
备注	

3.10 LL PKC通用驱动

3.10.1 PKC驱动的结构体

3.10.1.1 ll_ecc_point_t

ECC点坐标结构体ll_ecc_point_t的定义如下：

表 3-41 ll_ecc_point_t结构体

数据域	域段描述	取值
uint32_t X[ECC_U32_LENGTH]	ECC点x轴坐标。	该参数的取值由开发者指定。
uint32_t Y[ECC_U32_LENGTH]	ECC点y轴坐标。	该参数的取值由开发者指定。

3.10.1.2 ll_ecc_curve_init_t

ECC曲线参数结构体ll_ecc_curve_init_t的定义如下：

表 3-42 ll_ecc_curve_init_t结构体

数据域	域段描述	取值
uint32_t A[ECC_U32_LENGTH]	操作数A（与操作数B确定一条椭圆曲线）。	该参数的取值由开发者指定。
uint32_t B[ECC_U32_LENGTH]	操作数B（与操作数A确定一条椭圆曲线）。	该参数的取值由开发者指定。
uint32_t P[ECC_U32_LENGTH]	质数P。	该参数的取值由开发者指定。
uin32_t PRSquare[ECC_U32_LENGTH]	对P的模参数。	该参数的取值由开发者指定。
uin32_t ConstP	蒙哥马利乘法常量P。	该参数的取值由开发者指定。
uint32_t N[ECC_U32_LENGTH]	质数N。	该参数的取值由开发者指定。
uin32_t NRSquare[ECC_U32_LENGTH]	对N的模参数。	该参数的取值由开发者指定。
uin32_t ConstN	蒙哥马利乘法常量N。	该参数的取值由开发者指定。
uint32_t H	系数H。	该参数的取值由开发者指定。
ll_ecc_point_t G	椭圆曲线基点。	该参数的取值由开发者指定。

3.10.1.3 ll_pkc_init_t

PKC外设LL层初始化结构体ll_pkc_init_t的定义如下：

表 3-43 ll_pkc_init_t结构体

数据域	域段描述	取值
ll_ecc_curve_init_t *ecc_curve	要配置的椭圆曲线参数。参考ll_ecc_curve_init_t	该参数的取值由开发者指定。

数据域	域段描述	取值
uint32_t data_bits	数据位宽。	256 ～ 2048

3.10.2 PKC驱动API描述

PKC驱动的API主要包括：

表 3-44 PKC驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_pkc_init()	初始化PKC外设。
	ll_pkc_deinit()	反初始化PKC外设，恢复初始设置。
	ll_pkc_struct_init(ll_pkc_init_t *pkc_init)	初始化结构体pkc_init为默认值。

下面章节将对各API进行详细描述。

3.10.2.1 ll_pkc_init

表 3-45 ll_pkc_init接口

函数原型	error_status_t ll_pkc_init(pkc_regs_t *PKCx, ll_pkc_init_t *p_pkc_init)
功能说明	初始化PKC外设，使能外设及小端模式。
输入参数	PKCx：PKC外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：PKC外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.10.2.2 ll_pkc_deinit

表 3-46 ll_pkc_deinit接口

函数原型	error_status_t ll_pkc_deinit(pkc_regs_t *PKCx)
功能说明	将PKC外设寄存器反初始化为它们的默认重置值。
输入参数	PKCx：PKC外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：PKC外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.10.2.3 ll_pkc_struct_init

表 3-47 ll_pkc_struct_init接口

函数原型	void ll_pkc_struct_init(ll_pkc_init_t *p_pkc_init)

功能说明	将ll_pkc_init_t结构体变量初始化为默认重置值。
输入参数	p_pkc_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.11 LL PWM通用驱动

3.11.1 PWM驱动的结构体

3.11.1.1 ll_pwm_channel_init_t

PWM外设LL层通道初始化结构体ll_pwm_channel_init_t的定义如下：

表 3-48 ll_pwm_channel_init_t结构体

数据域	域段描述	取值
uint32_t duty	占空比，开发者也可通过ll_pwm_set_compare_a0()等接口设置该参数。	0 ～ 100
uint32_t drive_polarity	驱动极性，开发者也可通过ll_pwm_enable_positive_drive_channel_a()等接口设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_PWM_DRIVEPOLARITY_NEGATIVE（反向驱动） • LL_PWM_DRIVEPOLARITY_POSITIVE（正向驱动）

3.11.1.2 ll_pwm_init_t

PWM外设LL层初始化结构体ll_pwm_init_t的定义如下：

表 3-49 ll_pwm_init_t结构体

数据域	域段描述	取值
uint32_t mode	输出模式，开发者也可通过ll_pwm_set_mod()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_PWM_FLICKER_MODE（正常模式） • LL_PWM_BREATH_MODE（呼吸模式）
uint32_t align	对齐方式，开发者需要在初始化时设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_PWM_EDGE_ALIGNED（左边缘对齐） • LL_PWM_CENTER_ALIGNED（中心对齐）
uint32_t prescaler	输出周期，开发者也可通过ll_pwm_set_prescaler()设置该参数。	0x0000_0000 ～ 0xFFFF_FFFF，建议为128的倍数。
uint32_t bprescaler	呼吸周期（占空比由0逐渐增大到100所用时间），开发者也可通	0x0000_0000 ～ 0xFFFF_FFFF，建议为period*128的倍数。

数据域	域段描述	取值
	过ll_pwm_set_breath_prescaler()设置该参数。	
uint32_t hprescaler	呼吸保持周期（两个呼吸周期之间输出保持的时间），开发者也可通过ll_pwm_set_hold_prescaler()设置该参数。	0x0000_0000 ～ 0xFFFF_FFFF，建议为period的倍数。
ll_pwm_channel_init_t channel_a	通道A的初始化结构体。	参考ll_pwm_channel_init_t。
ll_pwm_channel_init_t channel_b	通道B的初始化结构体。	参考ll_pwm_channel_init_t。
ll_pwm_channel_init_t channel_c	通道C的初始化结构体。	参考ll_pwm_channel_init_t。

3.11.2 PWM驱动API描述

PWM驱动的API主要包括：

表 3-50 PWM驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_pwm_init()	初始化PWM外设。
	ll_pwm_deinit()	反初始化PWM外设，恢复初始设置。
	ll_pwm_struct_init()	初始化结构体变量ll_pwm_channel_init_t为默认值。

下面章节将对各API进行详细描述。

3.11.2.1 ll_pwm_init

表 3-51 ll_pwm_init接口

函数原型	error_status_t ll_pwm_init(pwm_regs_t *PWMx, ll_pwm_init_t *p_pwm_init)
功能说明	根据ll_pwm_init_t指定参数初始化PWM外设。
输入参数	PWMx：PWM外设实例。 p_pwm_init：指向ll_pwm_init_t结构体变量的指针，该结构体变量包含指定的PWM外设实例的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：PWM外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.11.2.2 ll_pwm_deinit

表 3-52 ll_pwm_deinit接口

函数原型	error_status_t ll_pwm_deinit(pwm_regs_t *PWMx)
功能说明	将PWM外设寄存器初始化为它们的默认重置值。
输入参数	PWMx：PWM外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：PWM外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.11.2.3 ll_pwm_struct_init

表 3-53 ll_pwm_struct_init接口

函数原型	void ll_pwm_struct_init(ll_pwm_init_t *p_pwm_init)
功能说明	将ll_pwm_init_t结构体变量初始化为默认重置值。
输入参数	p_pwm_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.12 LL SPI通用驱动

3.12.1 SPI驱动的结构体

3.12.1.1 ll_spim_init_t

SPIM外设LL层初始化结构体ll_spim_init_t的定义如下：

表 3-54 ll_spim_init_t结构体

数据域	域段描述	取值
uint32_t transfer_direction	数据传输方向，开发者也可通过ll_spi_set_transfer_direction()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_FULL_DUPLEX（全双工） • LL_SSI_SIMPLEX_TX（单工发送） • LL_SSI_SIMPLEX_RX（单工接收） • LL_SSI_READ_EEPROM（读EEPROM）
uint32_t data_size	数据传输位宽，开发者也可通过ll_spi_set_data_size()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_DATASIZE_4BIT（4位） • LL_SSI_DATASIZE_5BIT（5位） • LL_SSI_DATASIZE_6BIT（6位） • LL_SSI_DATASIZE_7BIT（7位） • LL_SSI_DATASIZE_8BIT（8位）

数据域	域段描述	取值
		• LL_SSI_DATASIZE_9BIT（9位） • LL_SSI_DATASIZE_10BIT（10位） • LL_SSI_DATASIZE_11BIT（11位） • LL_SSI_DATASIZE_12BIT（12位） • LL_SSI_DATASIZE_13BIT（13位） • LL_SSI_DATASIZE_14BIT（14位） • LL_SSI_DATASIZE_15BIT（15位） • LL_SSI_DATASIZE_16BIT（16位） • LL_SSI_DATASIZE_17BIT（17位） • LL_SSI_DATASIZE_18BIT（18位） • LL_SSI_DATASIZE_19BIT（19位） • LL_SSI_DATASIZE_20BIT（20位） • LL_SSI_DATASIZE_21BIT（21位） • LL_SSI_DATASIZE_22BIT（22位） • LL_SSI_DATASIZE_23BIT（23位） • LL_SSI_DATASIZE_24BIT（24位） • LL_SSI_DATASIZE_25BIT（25位） • LL_SSI_DATASIZE_26BIT（26位） • LL_SSI_DATASIZE_27BIT（27位） • LL_SSI_DATASIZE_28BIT（28位） • LL_SSI_DATASIZE_29BIT（29位） • LL_SSI_DATASIZE_30BIT（30位） • LL_SSI_DATASIZE_31BIT（31位） • LL_SSI_DATASIZE_32BIT（32位）
uint32_t clock_polarity	时钟极性，开发者也可通过ll_spi_set_clock_polarity()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_SCPOL_LOW（时钟空闲状态为低电平） • LL_SSI_SCPOL_HIGH（时钟空闲状态为高电平）
uint32_t clock_phase	时钟相位，开发者也可通过ll_spi_set_clock_phase()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_SCPHA_1EDGE（在时钟线的第一个跳变处采样数据） • LL_SSI_SCPHA_2EDGE（在时钟线的第二个跳变处采样数据）
uint32_t slave_select	选择的从设备，开发者也可通过ll_spi_enable_ss()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_SLAVE0（从设备0） • LL_SSI_SLAVE1（从设备1）
uint32_t baud_rate	波特率分频数，开发者也可通过ll_spi_set_baud_rate_prescaler()设置该参数。	2 ～ 65534的偶数

3.12.1.2 ll_spis_init_t

SPIS外设LL层初始化结构体ll_spis_init_t的定义如下：

表 3-55 ll_spis_init_t结构体

数据域	域段描述	取值
uint32_t data_size	数据传输位宽，开发者也可通过ll_spi_set_data_size()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_DATASIZE_4BIT（4位） • LL_SSI_DATASIZE_5BIT（5位） • LL_SSI_DATASIZE_6BIT（6位） • LL_SSI_DATASIZE_7BIT（7位） • LL_SSI_DATASIZE_8BIT（8位） • LL_SSI_DATASIZE_9BIT（9位） • LL_SSI_DATASIZE_10BIT（10位） • LL_SSI_DATASIZE_11BIT（11位） • LL_SSI_DATASIZE_12BIT（12位） • LL_SSI_DATASIZE_13BIT（13位） • LL_SSI_DATASIZE_14BIT（14位） • LL_SSI_DATASIZE_15BIT（15位） • LL_SSI_DATASIZE_16BIT（16位） • LL_SSI_DATASIZE_17BIT（17位） • LL_SSI_DATASIZE_18BIT（18位） • LL_SSI_DATASIZE_19BIT（19位） • LL_SSI_DATASIZE_20BIT（20位） • LL_SSI_DATASIZE_21BIT（21位） • LL_SSI_DATASIZE_22BIT（22位） • LL_SSI_DATASIZE_23BIT（23位） • LL_SSI_DATASIZE_24BIT（24位） • LL_SSI_DATASIZE_25BIT（25位） • LL_SSI_DATASIZE_26BIT（26位） • LL_SSI_DATASIZE_27BIT（27位） • LL_SSI_DATASIZE_28BIT（28位） • LL_SSI_DATASIZE_29BIT（29位） • LL_SSI_DATASIZE_30BIT（30位） • LL_SSI_DATASIZE_31BIT（31位） • LL_SSI_DATASIZE_32BIT（32位）
uint32_t clock_polarity	时钟极性，开发者也可通过ll_spi_set_clock_polarity()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_SCPOL_LOW（时钟空闲状态为低电平） • LL_SSI_SCPOL_HIGH（时钟空闲状态为高电平）
uint32_t clock_phase	时钟相位，开发者也可通过ll_spi_set_clock_phase()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_SCPHA_1EDGE（在时钟线的第一个跳变处采样数据）

数据域	域段描述	取值
		• LL_SSI_SCPHA_2EDGE（在时钟线的第二个跳变处采样数据）

3.12.1.3 ll_qspi_init_t

QSPI外设LL层初始化结构体ll_qspi_init_t的定义如下：

表 3-56 ll_qspi_init_t结构体

数据域	域段描述	取值
uint32_t transfer_direction	数据传输方向，开发者也可通过ll_spi_set_transfer_direction()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_FULL_DUPLEX（全双工） • LL_SSI_SIMPLEX_TX（单工发送） • LL_SSI_SIMPLEX_RX（单工接收） • LL_SSI_READ_EEPROM（读EEPROM）
uint32_t instruction_size	指令位宽，开发者也可通过ll_spi_set_instruction_size()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_INSTSIZE_0BIT（0位） • LL_SSI_INSTSIZE_4BIT（4位） • LL_SSI_INSTSIZE_8BIT（8位） • LL_SSI_INSTSIZE_16BIT（16位）
uint32_t address_size	地址位宽，开发者也可通过ll_spi_set_address_size()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_ADDRSIZE_0BIT（0位） • LL_SSI_ADDRSIZE_4BIT（4位） • LL_SSI_ADDRSIZE_8BIT（8位） • LL_SSI_ADDRSIZE_12BIT（12位） • LL_SSI_ADDRSIZE_16BIT（16位） • LL_SSI_ADDRSIZE_20BIT（20位） • LL_SSI_ADDRSIZE_24BIT（24位） • LL_SSI_ADDRSIZE_28BIT（28位） • LL_SSI_ADDRSIZE_32BIT（32位）
uint32_t inst_addr_transfer_format	指令及地址传输格式，开发者也可通过ll_spi_set_add_inst_transfer_format()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_INST_ADDR_ALL_IN_SPI（指令和地址以SPI方式传输） • LL_SSI_INST_IN_SPI_ADDR_IN_SPIFRF（指令以SPI方式传输，地址以Dual/Quad SPI方式传输） • LL_SSI_INST_ADDR_ALL_IN_SPIFRF（指令和地址以Dual/Quad SPI方式传输）
uint32_t wait_cycles	等待时钟周期（Dual/Quad SPI单工接收时有效），开发者也可通过ll_spi_set_wait_cycles()设置该参数。	0 ～ 31
uint32_t data_size	数据传输位宽，开发者也可通过ll_spi_set_data_size()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_DATASIZE_4BIT（4位） • LL_SSI_DATASIZE_5BIT（5位）

数据域	域段描述	取值
		• LL_SSI_DATASIZE_6BIT（6位） • LL_SSI_DATASIZE_7BIT（7位） • LL_SSI_DATASIZE_8BIT（8位） • LL_SSI_DATASIZE_9BIT（9位） • LL_SSI_DATASIZE_10BIT（10位） • LL_SSI_DATASIZE_11BIT（11位） • LL_SSI_DATASIZE_12BIT（12位） • LL_SSI_DATASIZE_13BIT（13位） • LL_SSI_DATASIZE_14BIT（14位） • LL_SSI_DATASIZE_15BIT（15位） • LL_SSI_DATASIZE_16BIT（16位） • LL_SSI_DATASIZE_17BIT（17位） • LL_SSI_DATASIZE_18BIT（18位） • LL_SSI_DATASIZE_19BIT（19位） • LL_SSI_DATASIZE_20BIT（20位） • LL_SSI_DATASIZE_21BIT（21位） • LL_SSI_DATASIZE_22BIT（22位） • LL_SSI_DATASIZE_23BIT（23位） • LL_SSI_DATASIZE_24BIT（24位） • LL_SSI_DATASIZE_25BIT（25位） • LL_SSI_DATASIZE_26BIT（26位） • LL_SSI_DATASIZE_27BIT（27位） • LL_SSI_DATASIZE_28BIT（28位） • LL_SSI_DATASIZE_29BIT（29位） • LL_SSI_DATASIZE_30BIT（30位） • LL_SSI_DATASIZE_31BIT（31位） • LL_SSI_DATASIZE_32BIT（32位）
uint32_t clock_polarity	时钟极性，开发者也可通过ll_spi_set_clock_polarity()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_SCPOL_LOW（时钟空闲状态为低电平） • LL_SSI_SCPOL_HIGH（时钟空闲状态为高电平）
uint32_t clock_phase	时钟相位，开发者也可通过ll_spi_set_clock_phase()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_SSI_SCPHA_1EDGE（在时钟线的第一个跳变处采样数据） • LL_SSI_SCPHA_2EDGE（在时钟线的第二个跳变处采样数据）
uint32_t baud_rate	波特率分频数，开发者也可通过ll_spi_set_baud_rate_prescaler()设置该参数。	2 ～ 65534的偶数
uint32_t rx_sample_delay	RX延时采集参数	0x0 ～ 0x7

3.12.2 SPI驱动API描述

SPI驱动的API主要包括：

表 3-57 SPI驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_spim_init()	初始化SPIM外设。
	ll_spim_deinit()	反初始化SPIM外设，恢复初始设置。
	ll_spim_struct_init()	初始化结构体变量ll_spim_init_t为默认值。
	ll_spis_init()	初始化SPIS外设。
	ll_spis_deinit()	反初始化SPIS外设，恢复初始设置。
	ll_spis_struct_init()	初始化结构体变量ll_spis_init_t为默认值。
	ll_qspi_init()	初始化QSPI外设。
	ll_qspi_deinit()	反初始化QSPI外设，恢复初始设置。
	ll_qspi_struct_init()	初始化结构体变量ll_qspi_init_t为默认值。

下面章节将对各API进行详细描述。

3.12.2.1 ll_spim_init

表 3-58 ll_spim_init接口

函数原型	error_status_t ll_spim_init(ssi_regs_t *SPIx, ll_spim_init_t *p_spi_init)
功能说明	根据ll_spim_init_t指定参数初始化SPIM外设。
输入参数	SPIx：SPIM外设实例。 p_spi_init：指向ll_spim_init_t结构体变量的指针，该结构体变量包含指定的SPI外设实例的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：SPI外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.12.2.2 ll_spim_deinit

表 3-59 ll_spim_deinit接口

函数原型	error_status_t ll_spim_deinit(ssi_regs_t *SPIx)
功能说明	将SPI外设寄存器初始化为它们的默认重置值。
输入参数	SPIx：SPIM外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：SPI外设寄存器已被成功反初始化 • ERROR：未成功反初始化

备注	

3.12.2.3 ll_spim_struct_init

表 3-60 ll_spim_struct_init接口

函数原型	void ll_spim_struct_init(ll_spim_init_t *p_spi_init)
功能说明	将ll_spim_init_t结构体变量初始化为默认重置值。
输入参数	p_spi_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.12.2.4 ll_spis_init

表 3-61 ll_spis_init接口

函数原型	error_status_t ll_spis_init(ssi_regs_t *SPIx, ll_spis_init_t *p_spi_init)
功能说明	根据ll_spis_init_t指定参数初始化SPIS外设。
输入参数	SPIx：SPIS外设实例。 p_spi_init：指向ll_spis_init_t结构体变量的指针，该结构体变量包含指定的SPI外设实例的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：SPI外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.12.2.5 ll_spis_deinit

表 3-62 ll_spis_deinit接口

函数原型	error_status_t ll_spis_deinit(ssi_regs_t *SPIx)
功能说明	将SPI外设寄存器初始化为它们的默认重置值。
输入参数	SPIx：SPIS外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：SPI外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.12.2.6 ll_spis_struct_init

表 3-63 ll_spis_struct_init接口

函数原型	void ll_spis_struct_init(ll_spis_init_t *p_spi_init)

功能说明	将ll_spis_init_t结构体变量初始化为默认重置值。
输入参数	p_spi_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.12.2.7 ll_qspi_init

表 3-64 ll_qspi_init接口

函数原型	error_status_t ll_qspi_init(ssi_regs_t *SPIx, ll_qspi_init_t *p_spi_init)
功能说明	根据ll_qspi_init_t指定参数初始化QSPI外设。
输入参数	SPIx：QSPI外设实例。 p_spi_init：指向ll_qspi_init_t结构体变量的指针，该结构体变量包含指定的SPI外设实例的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：SPI外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.12.2.8 ll_qspi_deinit

表 3-65 ll_qspi_deinit接口

函数原型	error_status_t ll_qspi_deinit(ssi_regs_t *SPIx)
功能说明	将SPI外设寄存器初始化为它们的默认重置值。
输入参数	SPIx：QSPI外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：SPI外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.12.2.9 ll_qspi_struct_init

表 3-66 ll_qspi_struct_init接口

函数原型	void ll_qspi_struct_init(ll_qspi_init_t *p_spi_init)
功能说明	将ll_qspi_init_t结构体变量初始化为默认重置值。
输入参数	p_spi_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.13 LL TIMER通用驱动

3.13.1 TIMER驱动的结构体

3.13.1.1 ll_timer_init_t

TIMER外设LL层初始化结构体ll_timer_init_t的定义如下：

表 3-67 ll_timer_init_t结构体

数据域	域段描述	取值
uint32_t auto_reload	计数初值，开发者也可通过ll_tim_set_auto_reload()设置该参数。	0x0000_0000 ～ 0xFFFF_FFFF

3.13.2 TIMER驱动API描述

TIMER驱动的API主要包括：

表 3-68 TIMER驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_timer_init()	初始化TIMER外设。
	ll_timer_deinit()	反初始化TIMER外设，恢复初始设置。
	ll_timer_struct_init()	初始化结构体变量ll_timer_init_t为默认值。

下面章节将对各API进行详细描述。

3.13.2.1 ll_timer_init

表 3-69 ll_timer_init接口

函数原型	error_status_t ll_timer_init(timer_regs_t *TIMERx, ll_timer_init_t *p_timer_init)
功能说明	根据ll_timer_init_t指定参数初始化TIMER外设。
输入参数	TIMERx：TIMER外设实例。 p_timer_init：指向ll_timer_init_t结构体变量的指针，该结构体变量包含指定的TIMER的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：TIMER外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.13.2.2 ll_timer_deinit

表 3-70 ll_timer_deinit接口

函数原型	error_status_t ll_timer_deinit(tim_regs_t *TIMERx)

功能说明	将TIMERx外设的寄存器反初始化为它们的默认重置值。
输入参数	TIMERx：TIMER外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：TIMER外设寄存器已被成功反初始化 • ERROR：未成功反初始化
备注	

3.13.2.3 ll_timer_struct_init

表 3-71 ll_timer_struct_init接口

函数原型	void ll_timer_struct_init(ll_timer_init_t *p_timer_init)
功能说明	将ll_timer_init_t结构体变量初始化为默认重置值。
输入参数	p_timer_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.14 LL UART通用驱动

3.14.1 UART驱动的结构体

3.14.1.1 ll_uart_init_t

UART外设LL层初始化结构体ll_uart_init_t的定义如下：

表 3-72 ll_uart_init_t结构体

数据域	域段描述	取值
uint32_t baud_rate	波特率，开发者也可通过ll_uart_set_baud_rate()设置该参数。	9600 ～ 921600
uint32_t data_bits	数据位位宽，开发者也可通过ll_uart_set_data_bits_length()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_UART_DATABITS_5B（5位） • LL_UART_DATABITS_6B（6位） • LL_UART_DATABITS_7B（7位） • LL_UART_DATABITS_8B（8位）
uint32_t stop_bits	停止位位宽，开发者也可通过ll_uart_set_stop_bits_length()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_UART_STOPBITS_1（1位） • LL_UART_STOPBITS_1_5（1.5位） • LL_UART_STOPBITS_2（2位）
uint32_t parity	校验位，开发者也可通过ll_uart_set_parity()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_UART_PARITY_NONE（无校验） • LL_UART_PARITY_ODD（奇校验）

数据域	域段描述	取值
		• LL_UART_PARITY_EVEN（偶校验） • LL_UART_PARITY_SP0（校验位始终为0） • LL_UART_PARITY_SP1（校验位始终为1）
uint32_t hw_flow_ctrl	流控，开发者也可通过ll_uart_set_hw_flow_ctrl()设置该参数。	该参数的取值可以是下列值中的任意一个： • LL_UART_HWCONTROL_NONE（无流控） • LL_UART_HWCONTROL_RTS_CTS（自动流控）

3.14.2 UART驱动API描述

UART驱动的API主要包括：

表 3-73 URAT驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_uart_init()	初始化URAT外设。
	ll_uart_deinit()	反初始化URAT外设，恢复初始设置。
	ll_uart_struct_init()	初始化结构体变量ll_uart_init_t为默认值。

下面章节将对各API进行详细描述。

3.14.2.1 ll_uart_init

表 3-74 ll_uart_init接口

函数原型	error_status_t ll_uart_init(uart_regs_t *UARTx, ll_uart_init_t *p_uart_init)
功能说明	根据ll_uart_init_t指定参数初始化UART外设。
输入参数	UARTx：UART外设实例。 p_uart_init：指向ll_uart_init_t结构体变量的指针，该结构体变量包含指定的UART外设实例的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：UART外设寄存器已被成功初始化 • ERROR：未成功初始化
备注	

3.14.2.2 ll_uart_deinit

表 3-75 ll_uart_deinit接口

函数原型	error_status_t ll_uart_deinit(uart_regs_t *UARTx)
功能说明	将UART外设寄存器初始化为它们的默认重置值。
输入参数	UARTx：UART外设实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：UART外设寄存器已被成功反初始化

	• ERROR：未成功反初始化
备注	

3.14.2.3 ll_uart_struct_init

表 3-76 ll_uart_struct_init接口

函数原型	void ll_uart_struct_init(ll_uart_init_t *p_uart_init)
功能说明	将ll_uart_init_t结构体变量初始化为默认重置值。
输入参数	p_uart_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.15 LL I2S通用驱动

3.15.1 I2S驱动的结构体

3.15.1.1 ll_i2s_init_t

I2S LL层初始化结构体ll_i2c_init_t的定义如下：

表 3-77 ll_i2s_init_t结构体

数据域	域段描述	取值
uint32_t rxdata_size	接收数据长度。	该参数的取值可以是下列值中的任意一个： • LL_I2S_DATASIZE_IGNORE • LL_I2S_DATASIZE_12BIT • LL_I2S_DATASIZE_16BIT • LL_I2S_DATASIZE_20BIT • LL_I2S_DATASIZE_24BIT • LL_I2S_DATASIZE_32BIT **说明：** • data_size = I2S_DATASIZE_12BIT（12 bits），传输的数据以16 bit地址对齐存放，高4 bits数据被忽略；硬件使用的WSS（字采样长度）为16 sclk cycles，高4 bit被忽略； • data_size = I2S_DATASIZE_20BIT（20 bits），传输的数据以32 bit地址对齐存放，高12 bits数据被忽略；硬件使用的WSS（字采样长度）为24 sclk cycles，高4 bit被忽略； • data_size = I2S_DATASIZE_24BIT（24 bits），传输的数据以32 bit地址对齐存放，高8 bits数据被忽略；硬件使用的WSS（字采样长度）为24 sclk cycles。
uint32_t txdata_size	发送数据长度。	该参数的取值可以是下列值中的任意一个： • LL_I2S_DATASIZE_IGNORE • LL_I2S_DATASIZE_12BIT • LL_I2S_DATASIZE_16BIT

数据域	域段描述	取值
		• LL_I2S_DATASIZE_20BIT • LL_I2S_DATASIZE_24BIT • LL_I2S_DATASIZE_32BIT
uint32_t rx_threshold	接收FIFO阈值。	该参数的取值可以是下列值中的任意一个： • LL_I2S_THRESHOLD_1FIFO • LL_I2S_THRESHOLD_2FIFO • LL_I2S_THRESHOLD_3FIFO • LL_I2S_THRESHOLD_4FIFO • LL_I2S_THRESHOLD_5FIFO • LL_I2S_THRESHOLD_6FIFO • LL_I2S_THRESHOLD_7FIFO • LL_I2S_THRESHOLD_8FIFO • LL_I2S_THRESHOLD_9FIFO • LL_I2S_THRESHOLD_10FIFO • LL_I2S_THRESHOLD_11FIFO • LL_I2S_THRESHOLD_12FIFO • LL_I2S_THRESHOLD_13FIFO • LL_I2S_THRESHOLD_14FIFO • LL_I2S_THRESHOLD_15FIFO • LL_I2S_THRESHOLD_16FIFO
uint32_t tx_threshold	发送FIFO阈值。	该参数的取值可以是下列值中的任意一个： • LL_I2S_THRESHOLD_1FIFO • LL_I2S_THRESHOLD_2FIFO • LL_I2S_THRESHOLD_3FIFO • LL_I2S_THRESHOLD_4FIFO • LL_I2S_THRESHOLD_5FIFO • LL_I2S_THRESHOLD_6FIFO • LL_I2S_THRESHOLD_7FIFO • LL_I2S_THRESHOLD_8FIFO • LL_I2S_THRESHOLD_9FIFO • LL_I2S_THRESHOLD_10FIFO • LL_I2S_THRESHOLD_11FIFO • LL_I2S_THRESHOLD_12FIFO • LL_I2S_THRESHOLD_13FIFO • LL_I2S_THRESHOLD_14FIFO • LL_I2S_THRESHOLD_15FIFO • LL_I2S_THRESHOLD_16FIFO
uint32_t clock_source	时钟源。	该参数的取值可以是下列值中的任意一个： • LL_I2S_CLOCK_SRC_96M • LL_I2S_CLOCK_SRC_32M
uint32_t audio_freq	音频频率。	audio_freq = fsck/(2 * wss)，fsck是I2S的串行时钟频率，最大取值为3027 kHz；WSS依赖于位宽参数可取值16、24、32，比如

数据域	域段描述	取值
		位宽配置为16 bit，WSS取值为16，audio_freq最大可配置为96 kHz。

3.15.2 I2S驱动API描述

I2S驱动的API主要包括：

表 3-78 I2S驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_i2s_init()	初始化I2S。
	ll_i2a_deinit()	反初始化I2S，恢复初始设置。
	ll_i2s_struct_init()	初始化结构体i2s_init为默认值。

下面章节将对各API进行详细描述。

3.15.2.1 ll_i2s_init

表 3-79 ll_i2s_init接口

函数原型	error_status_t ll_i2s_init(i2s_regs_t *I2Sx, ll_i2s_init_t *p_i2s_init)
功能说明	根据ll_i2s_init_t指定参数初始化I2S外设。
输入参数	I2Sx：I2S实例。 p_i2s_init：指向ll_i2s_init_t结构体变量的指针，该结构体变量包含指定的I2S实例的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：I2S成功初始化 • ERROR：未成功初始化
备注	

3.15.2.2 ll_i2s_deinit

表 3-80 ll_i2s_deinit接口

函数原型	error_status_t ll_i2s_deinit(i2s_regs_t *I2Sx)
功能说明	将I2S反初始化为它们的默认值。
输入参数	I2Sx：I2S实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：I2S已被成功反初始化 • ERROR：未成功反初始化
备注	

3.15.2.3 ll_i2s_struct_init

表 3-81 ll_i2s_struct_init接口

函数原型	void ll_i2s_struct_init(ll_i2s_init_t *p_i2s_init)
功能说明	将ll_i2s_init_t结构体变量初始化为默认值。
输入参数	p_i2s_init：指向待重置的结构体变量的指针。
返回值	无
备注	

3.16 LL RNG通用驱动

3.16.1 RNG驱动的结构体

3.16.1.1 ll_rng_init_t

RNG LL层初始化结构体ll_rng_init_t的定义如下：

表 3-82 ll_rng_init_t结构体

数据域	域段描述	取值
uint32_t seed	指定LFSR的种子方式	该参数的取值可以是下列值中的任意一个： • LL_RNG_SEED_FR0_S0（LFSR种子来自开关振荡器s0） • LL_RNG_SEED_USER（LFSR种子由用户配置）
uint32_t lfsr_mode	LFSR配置模式	该参数的取值可以是下列值中的任意一个： • LL_RNG_LFSR_MODE_59BIT（59bit LFSR） • LL_RNG_LFSR_MODE_128BIT（128bit LFSR）
uint32_t out_mode	随机数输出模式	该参数的取值可以是下列值中的任意一个： • LL_RNG_OUTPUT_FR0_S0（数字RNG直接输出） • LL_RNG_OUTPUT_CYCLIC_PARITY（LFSR和RNG循环采样和奇偶校验生成） • LL_RNG_OUTPUT_CYCLIC（LFSR和RNG循环采样） • LL_RNG_OUTPUT_LFSR_RNG（LFSR ⊕ RNG.） • LL_RNG_OUTPUT_LFSR（LFSR直接输出） 说明: 当seed_mode选择为LL_RNG_SEED_USER时，out_mode不能选择为LL_RNG_OUTPUT_FR0_S0。
uint32_t post_mode	后处理模式	该参数的取值可以是下列值中的任意一个： • LL_RNG_POST_PRO_NOT（不进行处理） • LL_RNG_POST_PRO_SKIPPING（跳位处理） • LL_RNG_OUTPUT_CYCLIC（位计数处理） • LL_RNG_OUTPUT_LFSR_RNG（冯纽曼处理）
uint32_t interrupt	中断	该参数的取值可以是下列值中的任意一个：

数据域	域段描述	取值
		• LL_RNG_IT_DISABLE • LL_RNG_IT_ENABLE

3.16.2 RNG驱动API描述

RNG驱动的API主要包括：

表 3-83 RNG驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_rng_init()	初始化RNG。
	ll_rng_deinit()	反初始化RNG，恢复初始设置。
	ll_rng_struct_init()	初始化结构体rng_init为默认值。

下面章节将对各API进行详细描述。

3.16.2.1 ll_rng_init

表 3-84 ll_rng_init接口

函数原型	error_status_t ll_rng_init(rng_regs_t *RNGx, ll_rng_init_t *p_rng_init)
功能说明	根据ll_rng_init_t指定参数初始化RNG外设。
输入参数	RNGx：RNG实例。 p_rng_init：指向ll_rng_init_t结构体变量的指针，该结构体变量包含指定的RNG实例的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：RNG成功初始化 • ERROR：未成功初始化
备注	

3.16.2.2 ll_rng_deinit

表 3-85 ll_rng_deinit接口

函数原型	error_status_t ll_rng_deinit(rng_regs_t *RNGx)
功能说明	将RNG反初始化为它们的默认重置值。
输入参数	RNGx：RNG实例。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：RNG已被成功反初始化 • ERROR：未成功反初始化
备注	

3.16.2.3 ll_rng_struct_init

表 3-86 ll_rng_struct_init接口

函数原型	void ll_rng_struct_init(ll_rng_init_t *p_rng_init);
功能说明	将ll_rng_init_t结构体变量初始化为默认值。
输入参数	p_rng_init：指向要重置的结构体变量的指针。
返回值	无
备注	

3.17 LL COMP通用驱动

3.17.1 COMP驱动的结构体

3.17.1.1 ll_comp_init_t

COMP LL层初始化结构体ll_comp_init_t的定义如下：

表 3-87 ll_comp_init_t结构体

数据域	域段描述	取值
uint32_t input_source	输入源	该参数的取值可以是下列值中的任意一个： • COMP_INPUT_SRC_IO0（MSIO0） • COMP_INPUT_SRC_IO1（MSIO1） • COMP_INPUT_SRC_IO2（MSIO2） • COMP_INPUT_SRC_IO3（MSIO3） • COMP_INPUT_SRC_IO4（MSIO4）
uint32_t ref_source	参考源	该参数的取值可以是下列值中的任意一个： • COMP_REF_SRC_IO0（MSIO0） • COMP_REF_SRC_IO1（MSIO1） • COMP_REF_SRC_IO2（MSIO2） • COMP_REF_SRC_IO3（MSIO3） • COMP_REF_SRC_IO4（MSIO4） • COMP_REF_SRC_VBAT • COMP_REF_SRC_VREF
uint32_t ref_value	参考值	• 如果ref_source是COMP_REF_SRC_VBAT，ref_value取值范围是0 ～ 7。 • 如果ref_source是COMP_REF_SRC_VREF，ref_value取值范围是0 ～ 63。 • 如果ref_source是COMP_REF_SRC_IOX（X为0 ～ 4），由外部输入决定，ref_value无效。

3.17.2 COMP驱动API描述

COMP驱动的API主要包括：

表 3-88 RNG驱动的APIs

API类别	API名称	描述
初始化/反初始化	ll_comp_init()	初始化COMP。
	ll_comp_deinit()	反初始化COMP，恢复初始设置。
	ll_comp_struct_init()	初始化结构体ll_comp_init_t为默认值。

下面章节将对各API进行详细描述。

3.17.2.1 ll_comp_init

表 3-89 ll_comp_init接口

函数原型	error_status_t ll_comp_init(ll_comp_init_t *p_comp_init)
功能说明	根据ll_comp_init_t指定参数初始化COMP。
输入参数	p_comp_init：指向ll_comp_init_t结构体变量的指针，该结构体变量包含指定的COMP的配置信息。
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：COMP成功初始化 • ERROR：未成功初始化
备注	

3.17.2.2 ll_comp_deinit

表 3-90 ll_comp_deinit接口

函数原型	error_status_t ll_comp_deinit(void)
功能说明	将COMP反初始化为它们的默认重置值。
输入参数	无
返回值	error_status_t枚举类型的一个值，可以是： • SUCCESS：COMP已被成功反初始化 • ERROR：未成功反初始化
备注	

3.17.2.3 ll_comp_struct_init

表 3-91 ll_comp_struct_init接口

函数原型	void ll_comp_struct_init(ll_comp_init_t *p_comp_init)
功能说明	将ll_comp_init_t结构体变量初始化为默认值。
输入参数	p_comp_init：指向要重置的结构体变量的指针。
返回值	无
备注	

4 术语和缩略语

表 4-1 术语和缩略语

名称	描述
ADC	Analog-to-digital Converter，模数转换器
AES	Advanced Encryption Standard，高级加密标注
AON GPIO	Always-on GPIO，不间断电源的GPIO
AON WDT	Always-on WDT，不间断电源的WDT
API	Application Programming Interface，应用程序接口
BLE	Bluetooth Low Energy，低功耗蓝牙
DMA	Direct Memory Access，直接内存访问
GPIO	General Purpose I/O，通用I/O口
HAL	Hardware Abstraction Layer，硬件抽象层
HMAC	Hash Message Authentication Code，哈希消息身份验证代码
I2C	Inter-integrated Circuit，内置集成电路
I2S	Inter-IC Sound，集成电路内置音频总线
LL	Low Layer，底层
MSIO	Mixed Signal I/O，混合信号I/O口
MSP	MCU Specific Package，MCU操作的封装
NVIC	Nested Vectored Interrupt Controller，嵌套向量中断控制器
PKC	Public Key Cipher，公钥密码体系
PWM	Pulse Width Modulation，脉冲宽度调制
PWR	Power Controller，电源控制器
RNG	Random Number Generator，随机数生成器
SPI	Serial Peripheral Interface，串行外设接口
SysTick	System Tick Timer，系统节拍定时器
TIM	Timer，定时器
UART	Universal Asynchronous Receiver/Transmitter，通用非同步收发传输器
WDT	Watchdog Timer，看门狗定时器