

 SCICS	Title: SC2D007-S9070x_Peripheral_API_Reference
	Document No. 文件编号: SCDD007
	Version No. 版本号: V1.0
	Effective Date 生效日期: 2019-06-15
	Page 页码: 1

S9070x Peripheral API Reference

Document Revision: V1.0

Document Release: 2019/06/15

SmartChip Integration Inc.

9B, Science Plaza, International
Science Park, 1355 Jinjihu Avenue,
Suzhou Industrial Park, Suzhou,
Jiangsu, China.
ZIP: 215021

Telephone: +86-512-62620006

Fax: +86-512-62620002

E-mail: sales@sci-inc.com.cn

Website: <http://www.sci-inc.com.cn>



目录

1 概述.....	8
1.1 简介	8
1.2 特别说明	8
1.2.1 msp 说明.....	8
1.2.2 外设 msp 函数列表.....	8
1.2.3 msp 函数实现示例.....	9
2 API 介绍	10
2.1 UART.....	10
2.1.1 hal_status_e s907x_hal_uart_init(uart_hdl_t *uart);	11
2.1.2 hal_status_e s907x_hal_uart_deinit(uart_hdl_t *uart);	12
2.1.3 int s907x_hal_uart_tx(uart_hdl_t *uart, u8 *pbuf, uint16_t size, uint32_t timeout);	
13	
2.1.4 int s907x_hal_uart_rx(uart_hdl_t *uart, u8 *pbuf, uint16_t size, uint32_t timeout);	
14	
2.1.5 hal_status_e s907x_hal_uart_tx_it(uart_hdl_t *uart, u8 *pbuf, uint16_t size);.....	15
2.1.6 hal_status_e s907x_hal_uart_rx_it(uart_hdl_t *uart, u8 *pbuf, uint16_t size);	16
2.1.7 hal_status_e s907x_hal_uart_rx_it_to(uart_hdl_t *uart, u8*pbuf);	17
2.1.8 hal_status_e s907x_hal_uart_tx_dma(uart_hdl_t *uart, u8 *pbuf, uint16_t size);...19	
2.1.9 hal_status_e s907x_hal_uart_rx_dma(uart_hdl_t *uart, u8 *pbuf, uint16_t size);...20	
2.1.10 u32 s907x_hal_uart_rx_dma_address(uart_hdl_t *uart);	22
2.1.11 hal_status_e s907x_hal_uart_dma_txstop(uart_hdl_t *uart);	22
2.1.12 hal_status_e s907x_hal_uart_dma_rxstop(uart_hdl_t *uart);	23
2.2 GPIO	24
2.2.1 hal_status_e s907x_hal_gpio_init(u32 gpio_pin, gpio_init_t *init);	25
2.2.2 hal_status_e s907x_hal_gpio_deinit(u32 gpio_pin);.....	25



2.2.3 hal_status_e s907x_hal_gpio_write(u32 gpio_pin, gpio_status_e status);	26
2.2.4 gpio_status_e s907x_hal_gpio_read(u32 gpio_pin);.....	26
2.2.5 hal_status_e s907x_hal_gpio_togglepin(u32 gpio_pin);.....	27
2.2.6 void s907x_hal_gpio_it_start(u32 gpio_pin, hal_int_cb cb, void *context);.....	28
2.2.7 void s907x_hal_gpio_it_stop(u32 gpio_pin);	28
2.2.8 hal_status_e s907x_hal_gpio_set_io(u32 gpio_pin, u8 io);	29
2.2.9 hal_status_e s907x_hal_gpio_set_pull(u32 gpio_pin, u8 pull);.....	30
2.3 FLASH	30
2.3.1 void s907x_hal_flash_write(u32 addr, u8 *pbuff, int len);	31
2.3.2 void s907x_hal_flash_read(u32 addr, u8 *pbuff, int len);	32
2.3.3 void s907x_hal_flash_erase(int erase_type, u32 addr);.....	32
2.4 SPI.....	33
2.4.1 hal_status_e s907x_hal_spi_init(spi_hdl_t *spi);.....	34
2.4.2 hal_status_e s907x_hal_spi_deinit(spi_hdl_t *spi);.....	35
2.4.3 int s907x_hal_spi_master_txrx(spi_hdl_t *spi, void *txbuf, void *rxbuf, u16 xfer_size, uint32_t ms);.....	36
2.4.4 int s907x_hal_spi_slaver_tx(spi_hdl_t* spi, void *txbuf, u16 xfer_size, uint32_t timeout);	37
2.4.5 int s907x_hal_spi_slaver_rx(spi_hdl_t* spi, void *rxbuf, u16 xfer_size, uint32_t timeout);	38
2.4.6 hal_status_e s907x_hal_spi_master_xfer_interrupt(spi_hdl_t *spi, u8 *pbuff, u16 xfer_size);	39
2.4.7 hal_status_e s907x_hal_spi_master_recv_interrupt(spi_hdl_t *spi, void *pbuff, u16 xfer_size);	40
2.4.8 hal_status_e s907x_hal_spi_slaver_xfer_interrupt(spi_hdl_t *spi, u8 *pbuff, u16 xfer_size);	41
2.4.9 hal_status_e s907x_hal_spi_slaver_recv_interrupt(spi_hdl_t *spi, u8 *pbuff, u16	



xfer_size);	42
2.4.10 hal_status_e s907x_hal_spi_master_xfer_dma(spi_hdl_t *spi, void *pbuf, u16 xfer_size);	43
2.4.11 hal_status_e s907x_hal_spi_master_recv_dma(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size);	44
2.4.12 hal_status_e s907x_hal_spi_slaver_xfer_dma(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size);	45
2.4.13 hal_status_e s907x_hal_spi_slaver_recv_dma(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size);	46
2.5 I2C	46
2.5.1 hal_status_e s907x_hal_i2c_init(i2c_hdl_t *i2c);	48
2.5.2 hal_status_e s907x_hal_i2c_deinit(i2c_hdl_t *i2c);	49
2.5.3 hal_status_e s907x_hal_i2c_master_xfer(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms);	50
2.5.4 hal_status_e s907x_hal_i2c_master_recv(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms);	51
2.5.5 hal_status_e s907x_hal_i2c_slavor_xfer(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms);	52
2.5.6 hal_status_e s907x_hal_i2c_slavor_recv(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms);	53
2.5.7 hal_status_e s907x_hal_i2c_master_xfer_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);	54
2.5.8 hal_status_e s907x_hal_i2c_master_recv_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);	55
2.5.9 hal_status_e s907x_hal_i2c_slavor_xfer_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);	56
2.5.10 hal_status_e s907x_hal_i2c_slavor_recv_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16	



xfer_size);	57
2.5.11 hal_status_e s907x_hal_i2c_master_xfer_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);	57
2.5.12 hal_status_e s907x_hal_i2c_master_recv_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);	58
2.5.13 hal_status_e s907x_hal_i2c_slavor_xfer_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);	59
2.5.14 hal_status_e s907x_hal_i2c_slavor_recv_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);	60
2.6 I2S.....	61
2.7 ADC.....	61
2.7.1 hal_status_e s907x_hal_adc_initadc_hdl_t *adc);	62
2.7.2 hal_status_e s907x_hal_adc_deinitadc_hdl_t *adc);	63
2.7.3 hal_status_e s907x_hal_adc_start_itadc_hdl_t *adc, u32 mode);	64
2.7.4 hal_status_e s907x_hal_adc_stop_itadc_hdl_t *adc, u32 mode);	64
2.7.5 hal_status_e s907x_hal_adc_poll_oneshotadc_hdl_t *adc, u32 timeout);	65
2.7.6 hal_status_e s907x_hal_adc_poll_continousadc_hdl_t *adc);	67
2.7.7 hal_status_e s907x_hal_adc_interrupt_oneshotadc_hdl_t *adc, hal_int_cb cb, void *arg);	68
2.7.8 hal_status_e s907x_hal_adc_interrupt_continousadc_hdl_t *adc, hal_int_cb cb, void *arg);	69
2.7.9 hal_status_e s907x_hal_adc_get_mvadc_hdl_t *adc);	70
2.8 TIMER	72
2.8.1 u32 s907x_hal_timer_get_counter(timer_hdl_t *tim);	73
2.8.2 hal_status_e s907x_hal_timer_base_inittimer_hdl_t *tim);	74
2.8.3 hal_status_e s907x_hal_timer_base_deinittimer_hdl_t *tim);	75
2.8.4 hal_status_e s907x_hal_timer_start_base(timer_hdl_t *tim);	75



2.8.5 hal_status_e s907x_hal_timer_stop(timer_hdl_t *tim);.....	76
2.8.6 hal_status_e s907x_hal_timer_set_period(timer_hdl_t *tim, u32 period);..	77
2.8.7 hal_status_e s907x_hal_timer_pwm_init(timer_hdl_t *tim);	78
2.8.8 hal_status_e s907x_hal_timer_pwm_deinit(timer_hdl_t *tim);	79
2.8.9 hal_status_e s907x_hal_timer_start_pwm(timer_hdl_t *tim);	80
2.8.10 hal_status_e s907x_hal_timer_stop_pwm(timer_hdl_t *tim);.....	81
2.8.11 hal_status_e s907x_hal_timer_start_pwm_dma(timer_hdl_t *tim);	81
2.8.12 hal_status_e s907x_hal_timer_stop_pwm_dma(timer_hdl_t *tim);	82
2.8.13 hal_status_e s907x_hal_timer_pwm_set_ccr(timer_hdl_t *tim, u32 ccr, u8 channel);	83
2.8.14 hal_status_e s907x_hal_timer_pwm_deinit(timer_hdl_t *tim);	84
2.1.15 hal_status_e s907x_hal_timer_capture_init(timer_hdl_t *tim);	84
2.8.16 hal_status_e s907x_hal_timer_start_capture(timer_hdl_t *tim);.....	86
2.8.17 hal_status_e s907x_hal_timer_start_capture_dma(timer_hdl_t *tim);	86
2.8.18 hal_status_e s907x_hal_timer_stop_capture_dma(timer_hdl_t *tim);.....	87
2.9 RTC	88
2.9.1 hal_status_e s907x_hal_rtc_init(rtc_hdl_t *rtc);.....	89
2.9.2 hal_status_e s907x_hal_rtc_deinit(rtc_hdl_t *rtc);	90
2.9.3 void s907x_hal_rtc_get_time(rtc_hdl_t *rtc);	90
2.9.4 void s907x_hal_rtc_get_alarm(rtc_hdl_t *rtc);	91
2.9.5 void s907x_hal_rtc_set_unixtime(rtc_hdl_t *rtc);	92
2.9.6 void s907x_hal_rtc_set_basictime(rtc_hdl_t *rtc);	93
2.9.7 void s907x_hal_rtc_alarm_init(rtc_hdl_t *rtc);	94
2.9.8 void s907x_hal_rtc_alarm_deinit(rtc_hdl_t *rtc);.....	95
2.9.9 void s907x_hal_rtc_alarm_cmd(rtc_hdl_t *rtc, u8 enable);	96
2.10 WDG	96
2.10.1 hal_status_e s907x_hal_wdg_init(wdg_hdl_t *wdg);	97



2.10.2 hal_status_e s907x_hal_wdg_deinit(wdg_hdl_t *wdg);	98
2.10.3 void s907x_hal_wdg_start(wdg_hdl_t *wdg);	99
2.10.4 void s907x_hal_wdg_refresh(wdg_hdl_t *wdg);	99
2.10.5 void s907x_hal_wdg_start_it(wdg_hdl_t *wdg);	100
2.10.6 void s907x_hal_wdg_stop(wdg_hdl_t *wdg);	101
3 附录.....	102
3.1 UART.....	102
3.2 GPIO	103
3.3 SPI.....	103
3.4 I2C	106
3.5 ADC.....	108
3.6 TIMER	108
3.7 RTC	111
3.8 WDG	112
3.9 OTHERS.....	112
4 版本信息.....	114



1 概述

1.1 简介

S907A 目前支持 UART, GPIO, FLASH, SPI, I2C, ADC, TIMER, RTC, WDG 等外设，其中 TIMER 支持 8 通道 PWM，脉宽捕获等功能；相关外设均提供友好，易用的 API；用户基于这些 API 可以很方便的，高效的开发应用；本文档主要是对外设 API 的介绍，同时会附着一些简短使用示例，以便帮助开发者更快捷的理解 API，提高应用开发效率。

1.2 特别说明

1.2.1 msp 说明

开发者在使用 s907A 外设时，特别要注意一下 `hal_pinmux.h` 这个文件，这个文件中的内容是引脚定义和复用问题；在调用相关外设初始化函数时候，IO 引脚在涉及到是否复用，复用成什么功能时，需要在相关的 `msp` 函数中做好配置，忽略这一点很有可能导致初始化失败。`msp` 函数我们已经定义好了，用户只需结合 PCB 原理图和应用需求在 `msp` 函数中做好配置就行；还要说明一点，`msp` 函数实现为用户根据需求而定，所以 `msp` 函数中的实现并不局限引脚复用问题。

1.2.2 外设 msp 函数列表

- ◆ `void s907x_hal_uart_msp_init(uart_hdl_t *uart)`
- ◆ `void s907x_hal_uart_msp_deinit(uart_hdl_t *uart)`
- ◆ `void s907x_hal_spi_msp_init(spi_hdl_t *spi)`
- ◆ `void s907x_hal_spi_msp_deinit(spi_hdl_t *spi)`
- ◆ `void s907x_hal_i2c_msp_init(i2c_hdl_t *i2c)`
- ◆ `void s907x_hal_i2c_msp_deinit(i2c_hdl_t *i2c)`
- ◆ `void s907x_hal_time_msp_init(void *context)`
- ◆ `void s907x_hal_time_msp_deinit(void *context)`



注: 在 timer 用于 pwm 输出或脉宽捕获时, hal_time_msp_init 和 hal_time_msp_deinit 必须要做好引脚配置。

1.2.3 msp 函数实现示例

以 uart 为例: 应用需要用到 uart0, 查看 PCB 原理图及模组管脚功能表得知, uart0 的 rx 对应芯片引脚 GPIO18, tx 对应芯片引脚 GPIO23, 结合 hal_pinmux.h 文件, 那么 uart 的 msp 函数实现如下示例

```
void s907x_hal_uart_msp_init(uart_hdl_t *uart)
{
    UART0_RX_SEL1(HAL_ON);
    UART0_TX_SEL1(HAL_ON);
}

void s907x_hal_uart_msp_deinit(uart_hdl_t *uart)
{
    UART0_RX_SEL1(HAL_OFF);
    UART0_TX_SEL1(HAL_OFF);
}
```



2 API 介绍

2.1 UART

◆ `hal_status_e s907x_hal_uart_init(uart_hdl_t *uart)`

对串口设备初始化

◆ `hal_status_e s907x_hal_uart_deinit(uart_hdl_t *uart)`

将指定串口配置恢复为缺省值

◆ `int s907x_hal_uart_tx(uart_hdl_t *uart, u8 *pbuf, uint16_t size, uint32_t timeout)`

查询的方式发送固定字节的数据

◆ `int s907x_hal_uart_rx(uart_hdl_t *uart, u8 *pbuf, uint16_t size, uint32_t timeout)`

查询的方式接收固定字节的数据

◆ `hal_status_e s907x_hal_uart_tx_it(uart_hdl_t *uart, u8 *pbuf, uint16_t size)`

中断的方式发送固定字节的数据

◆ `hal_status_e s907x_hal_uart_rx_it(uart_hdl_t *uart, u8 *pbuf, uint16_t size)`

中断的方式接收固定字节的数据

◆ `hal_status_e s907x_hal_uart_rx_it_to(uart_hdl_t *uart, u8*pbuf)`

采用串口硬件中断超时机制完成数据接收

◆ `hal_status_e s907x_hal_uart_tx_dma(uart_hdl_t *uart, u8 *pbuf, uint16_t size);`

`dma` 的方式发送固定字节的数据

◆ `hal_status_e s907x_hal_uart_rx_dma(uart_hdl_t *uart, u8 *pbuf, uint16_t size)`

`dma` 的方式接收固定字节的数据

◆ `u32 s907x_hal_uart_rx_dma_address(uart_hdl_t *uart)`

获取当前 `dma` 接收缓冲区的地址

◆ `hal_status_e s907x_hal_uart_dma_txstop(uart_hdl_t *uart)`

使指定串口停止 `dma` 发送

◆ `hal_status_e s907x_hal_uart_dma_rxstop(uart_hdl_t *uart)`



使指定串口停止dma接收

2.1.1 hal_status_e s907x_hal_uart_init(uart_hdl_t *uart);

功能: 对串口设备初始化

参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 对 uart_0 初始化

```
/*声明实例句柄*/
uart_hdl_t  uart0_hd;
/*声明一个 uart_hdl_t 类型指针*/
uart_hdl_t  *p_uart0_hdl;
/*指向 uart0 实例句柄*/
p_uart0_hdl = &uart0_hd;
memset((void*)p_uart0_hdl, 0, sizeof(uart_hdl_t));
/*初始化的对象为 UART_0*/
p_uart0_hdl->config.idx = UART_0;
/*波特率 115200*/
p_uart0_hdl->config.baud = 115200;
/*无奇偶校验位*/
p_uart0_hdl->config.parity = UART_PARITY_NONE;
/*一个停止位*/
p_uart0_hdl->config.stopbits = UART_STOPBITS_1;
```

```
/*8个数据位*/  
p_uart0_hdl->config.datalen = UART_DATALENGTH_8B;  
  
/*串口低功耗模式失能*/  
  
p_uart0_hdl->config.lpmode = UART_LP_DISABLE;  
  
/*fifo threshold*/  
  
uart->config.rx_thd = 0;  
  
uart->config.tx_thd = 0;  
  
/*将 uart0 配置信息写入寄存器*/  
s907x_hal_uart_init(p_uart0_hdl);  
  
/**以上信息配置完毕以后即可通过查询的方式收发数据**/
```

2.1.2 hal_status_e s907x_hal_uart_deinit(uart_hdl_t *uart);

功能: 将指定串口配置恢复为缺省值

参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 将上述 uart_0 配置恢复为缺省值

```
/*声明一个 uart_hdl_t 类型指针*/  
uart_hdl_t *p_uart0_hdl;  
  
/*指向 uart0 实例句柄*/  
p_uart0_hdl = &uart0_hd;
```



```
/*将 uart0 配置恢复为缺省值*/  
s907x_hal_uart_deinit(p_uart0_hdl);
```

2.1.3 int s907x_hal_uart_tx(uart_hdl_t *uart, u8 *pbuf, uint16_t size, uint32_t timeout);

功能: 查询的方式发送固定字节的数据

注意: 调用此函数前需要已经完成对所操作串口的初始化

参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针
u8	pbuf: 指向待发送数据缓冲区的指针
uint16_t	size: 发送数据的长度
uint32_t	timeout: 查询等待的时间; 单位: ms

返回:

类型	描述
int	返回值为整型, 表示成功发送的数据长度

示例: 通过串口 uart0, 以查询的方式发送"hello,s907a!"

假设 uart0 已经配置完毕, 且实例句柄为: uart0_hd

```
/*声明一个 uart_hdl_t 类型指针*/
```

```
uart_hdl_t *p_uart0_hdl;
```

```
/*指向 uart0 实例句柄*/
```

```
p_uart0_hdl = &uart0_hd;
```

```
/*准备要发送的数据*/
```



```
u8 txbuffer[] = "hello,s907a!";  
/*查询的方式将准备好的数据发送出去，查询时间 1s*/  
s907x_hal_uart_tx(p_uart0_hdl, txbuffer, sizeof(txbuffer), 1000);
```

2.1.4 int s907x_hal_uart_rx(uart_hdl_t *uart, u8 *pbuf, uint16_t size, uint32_t timeout);

功能: 查询的方式接收固定字节的数据

注意: 调用此函数前需要已经完成对所操作串口的初始化

参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针
u8	pbuf: 指向待接收数据缓冲区指针
uint16_t	size: 接收指定长度的数据
uint32_t	timeout: 查询等待的时间; 单位: ms

返回:

类型	描述
int	返回值为整型, 表示成功接收的数据长度

示例: 通过串口 uart0, 以查询的方式接收 10 个字节的数据

假设 uart0 已经配置完毕, 且实例句柄为: uart0_hd

```
/*声明一个 uart_hdl_t 类型指针*/
```

```
uart_hdl_t *p_uart0_hdl;
```

```
/*指向 uart0 实例句柄*/
```

```
p_uart0_hdl = &uart0_hd;
```



```
/*准备好接收数据缓冲区*/  
u8 rxbuffer[10] = {0};  
  
/*通过查询方式接收串口 10 个字节的数据，查询时间 1s*/  
s907x_hal_uart_rx(p_uart0_hdl, rxbuffer, 10, 1000);
```

2.1.5 hal_status_e s907x_hal_uart_tx_it(uart_hdl_t *uart, u8 *pbuff, uint16_t size);

功能: 中断的方式发送固定字节的数据

注意: 调用此函数前需要已经完成对所操作串口的初始化，并已配置中断回调函数

参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针
u8	pbuff: 指向待发送数据缓冲区指针
uint16_t	size: 发送指定数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: 通过串口 uart0，以中断的方式发送"hello,s907a!"

假设 uart0 已经配置完毕，且实例句柄为: uart0_hd

```
/*声明一个 uart_hdl_t 类型指针*/  
uart_hdl_t *p_uart0_hdl;  
  
/*指向 uart0 实例句柄*/  
  
p_uart0_hdl = &uart0_hd;  
  
/*配置中断回调函数 begin*/  
  
p_uart0_hdl->it.tx_complete.func= txdone_cb; /*中断发送完成回调函数*/
```

```
p_uart0_hdl->it.tx_complete.context = p_uart0_hdl;  
p_uart0_hdl->it.rx_complete.func= rxdone_cb; /*中断接收完成回调函数*/  
p_uart0_hdl->it.rx_complete.context = p_uart0_hdl;  
p_uart0_hdl->it.trx_error.func = trx_error_cb; /*中断发送接收异常回调函数*/  
p_uart0_hdl->it.trx_error.context = p_uart0_hdl;  
/*配置中断回调函数 end*/  
/*准备好将要发送的数据*/  
u8 txbuffer[] = "hello,s907a!";  
/*通过中断的方式将准备好的数据发送出去*/  
s907x_hal_uart_tx_it(p_uart0_hdl, txbuffer, sizeof(txbuffer));
```

2.1.6 hal_status_e s907x_hal_uart_rx_it(uart_hdl_t *uart, u8 *pbuf, uint16_t size);

功能: 中断的方式接收固定字节的数据

注意: 调用此函数前需要已经完成对所操作串口的初始化，并已配置中断回调函数

参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针
u8	pbuf: 指向待接收数据缓冲区指针
uint16_t	size: 接收指定字节的数据

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: 通过串口 uart0，以中断的方式接收 10 个字节的数据

假设 uart0 已经配置完毕，且实例句柄为: uart0_hd



```
/*声明一个 uart_hdl_t 类型指针*/  
uart_hdl_t *p_uart0_hdl;  
  
/*指向 uart0 实例句柄*/  
p_uart0_hdl = &uart0_hd;  
  
/*配置中断回调函数 begin*/  
p_uart0_hdl->it.tx_complete.func= txdone_cb; /*中断发送完成回调函数*/  
p_uart0_hdl->it.tx_complete.context = p_uart0_hdl;  
p_uart0_hdl->it.rx_complete.func= rxdone_cb; /*中断接收完成回调函数*/  
p_uart0_hdl->it.rx_complete.context = p_uart0_hdl;  
p_uart0_hdl->it.trx_error.func = trx_error_cb; /*中断发送接收异常回调函数*/  
p_uart0_hdl->it.trx_error.context = p_uart0_hdl;  
  
/*配置中断回调函数 end*/  
  
/*准备接收数据缓冲区*/  
u8 rxbuffer[10] = {0};  
  
/*通过中断的方式接收 10 个字节的数据*/  
s907x_hal_uart_rx_it(p_uart0_hdl, rxbuffer, 10);
```

2.1.7 hal_status_e s907x_hal_uart_rx_it_to(uart_hdl_t *uart, u8*pbuf);

功能: 采用串口硬件中断超时机制完成数据接收(每一帧数据长度不可以大于 16)

注意: 调用此函数前需要已经完成对所操作串口的初始化，并已配置超时回调函数

参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针
u8	pbuf: 指向待接收数据缓冲区指针

返回:



类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 通过串口 uart0, 以硬件中断超时的方式接收数据帧, 且持续接收

```
/*超时回调函数实现*/
static void rxtimeout_cb(void *context)
{
    /*固定的格式 begin*/
    uart_hdl_t *uart = (uart_hdl_t *)context;
    while (uart_ll_rx_allow(uart)){
        *uart->it.rdbuf++ = uart_ll_recv_byte(uart);
        uart->it.rxlen++;
    }
    /*固定的格式 end*/
    /*对收到的数据帧处理, 这里只将收到的数据帧原样返回*/
    s907x_hal_uart_tx(uart, rdbuf, uart->it.rxlen, 1000);
    /*清除并重新接收*/
    uart->it.rxlen = 0;
    s907x_hal_uart_rx_it_to(uart, rdbuf);
}
```

假设 uart0 已经配置完毕, 且实例句柄为: uart0_hd

```
u8 rdbuf[256] = {0};
/*声明一个 uart_hdl_t 类型指针*/
uart_hdl_t *p_uart0_hdl;
/*指向 uart0 实例句柄*/
p_uart0_hdl = &uart0_hd;
/*配置中断超时回调函数 begin*/
```

```
/*中断超时回调函数*/  
p_uart0_hdl->it.rx_timeout.func= rxtimeout_cb;  
p_uart0_hdl->it.rx_timeout.context = p_uart0_hdl;  
/*配置中断超时回调函数 end*/  
/*开启中断超时接收数据功能*/  
s907x_hal_uart_rx_it_to(p_uart0_hdl, rxbuffer);
```

2.1.8 hal_status_e s907x_hal_uart_tx_dma(uart_hdl_t *uart, u8 *pbuff, uint16_t size);

功能: dma 的方式发送固定字节的数据

注意: 调用此函数前需要已经完成对所操作串口的初始化，并已配置 dma 回调函数
参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针
u8	pbuff: 指向待发送数据缓冲区指针
uint16_t	size: 发送指定字节的数据

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 通过串口 uart0, 以 dma 的方式发送"hello,s907a!"

假设 uart0 已经配置完毕, 且实例句柄为: uart0_hd

```
/*声明一个 uart_hdl_t 类型指针*/  
uart_hdl_t *p_uart0_hdl;
```

```
/*指向 uart0 实例句柄*/  
p_uart0_hdl = &uart0_hd;  
  
/*配置 dma 回调函数 begin*/  
  
/*dma tx rx burst size 设置 4*/  
p_uart0_hdl->dma.rx_burst_size = 4;  
p_uart0_hdl->dma.tx_burst_size = 4;  
  
/*配置 dma 接收完成回调函数*/  
p_uart0_hdl->dma.rx_complete.func= rxdone_dma_cb;  
p_uart0_hdl->dma.rx_complete.context = p_uart0_hdl;  
  
/*配置 dma 发送完成回调函数*/  
p_uart0_hdl->dma.tx_complete.func= txdone_dma_cb;  
p_uart0_hdl->dma.tx_complete.context = p_uart0_hdl;  
  
/*配置 dma 回调函数 end*/  
  
/*准备好将要发送的数据*/  
u8 txbuffer[] = "hello,s907a!";  
  
/*通过 dma 的方式将准备好的数据发送出去*/  
s907x_hal_uart_tx_dma(p_uart0_hdl, txbuffer, sizeof(txbuffer));
```

2.1.9 hal_status_e s907x_hal_uart_rx_dma(uart_hdl_t *uart, u8 *pbuf, uint16_t size);

功能: dma 的方式接收固定字节的数据

注意: 调用此函数前需要已经完成对所操作串口的初始化，并已配置 dma 回调函数
参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针



u8	pbuf: 指向待接收数据缓冲区指针
uint16_t	size: 接收指定长度的数据

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 通过串口 uart0, 以 dma 的方式接收 10 个字节的数据

假设 uart0 已经配置完毕, 且实例句柄为: uart0_hdl

```
/*声明一个 uart_hdl_t 类型指针*/
uart_hdl_t *p_uart0_hdl;

/*指向 uart0 实例句柄*/
p_uart0_hdl = &uart0_hd;

/*配置 dma 回调函数 begin*/
/*dma tx rx burst size 设置 4*/
p_uart0_hdl->dma.rx_burst_size = 4;
p_uart0_hdl->dma.tx_burst_size = 4;

/*配置 dma 接收完成回调函数*/
p_uart0_hdl->dma.rx_complete.func= rxdone_dma_cb;
p_uart0_hdl->dma.rx_complete.context = p_uart0_hdl;

/*配置 dma 发送完成回调函数*/
p_uart0_hdl->dma.tx_complete.func= txdone_dma_cb;
p_uart0_hdl->dma.tx_complete.context = p_uart0_hdl;

/*配置 dma 回调函数 end*/
/*准备接收数据缓冲区*/
u8 rxbuffer[10] = {0};

/*通过 dma 的方式接收 10 个字节的数据*/
s907x_hal_uart_rx_dma(p_uart0_hdl, rxbuffer, 10);
```



2.1.10 u32 s907x_hal_uart_rx_dma_address(uart_hdl_t *uart);

功能: dma 接收数据时, 获取当前 dma 接收缓冲区的地址

参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针

返回:

类型	描述
u32	32 位地址

示例: 获取当前 dma 的地址

假设 uart0 已经配置完毕, 且实例句柄为: uart0_hd, 且已经配置 dma 信息

```
/*声明一个 uart_hdl_t 类型指针*/\n\nuart_hdl_t *p_uart0_hdl;\n\n/*指向 uart0 实例句柄*/\n\np_uart0_hdl = &uart0_hd;\n\n/*获取 dma 的地址*/\n\nu32 addr = s907x_hal_uart_rx_dma_address(p_uart0_hdl);
```

2.1.11 hal_status_e s907x_hal_uart_dma_txstop(uart_hdl_t *uart);

功能: 使指定串口停止 dma 发送

参数:

类型	描述



uart_hdl_t	uart: 指向串口实例句柄指针
------------	------------------

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 停止串口 uart0 的 dma 发送

假设 uart0 已经配置完毕, 且实例句柄为: uart0_hd, 且已经配置 dma 信息

```
/*声明一个 uart_hdl_t 类型指针*/  
uart_hdl_t *p_uart0_hdl;  
  
/*指向 uart0 实例句柄*/  
p_uart0_hdl = &uart0_hd;  
  
/*停止串口 uart0 的 dma 发送*/  
s907x_hal_uart_dma_txstop(p_uart0_hdl);
```

2.1.12 hal_status_e s907x_hal_uart_dma_rxstop(uart_hdl_t *uart);

功能: 使指定串口停止 dma 接收

参数:

类型	描述
uart_hdl_t	uart: 指向串口实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 停止串口 uart0 的 dma 接收

假设 uart0 已经配置完毕, 且实例句柄为: uart0_hd, 且已经配置 dma 信息

```
/*声明一个 uart_hdl_t 类型指针*/  
uart_hdl_t *p_uart0_hdl;  
/*指向 uart0 实例句柄*/  
p_uart0_hdl = &uart0_hd;  
/*停止串口 uart0 的 dma 接收*/  
s907x_hal_uart_dma_rxstop(p_uart0_hdl);
```

2.2 GPIO

- ◆ `hal_status_e s907x_hal_gpio_init(u32 gpio_pin, gpio_init_t *init)`
对 `gpio` 口的初始化
- ◆ `hal_status_e s907x_hal_gpio_deinit(u32 gpio_pin)`
将指定 `gpio` 口恢复为缺省值状态
- ◆ `hal_status_e s907x_hal_gpio_write(u32 gpio_pin, gpio_status_e status)`
设定指定 `gpio` 口的输出的电平状态
- ◆ `gpio_status_e s907x_hal_gpio_read(u32 gpio_pin)`
读取指定 `gpio` 口的输入的电平状态
- ◆ `hal_status_e s907x_hal_gpio_togglepin(u32 gpio_pin)`
翻转指定 `gpio` 口输出的电平状态
- ◆ `void s907x_hal_gpio_it_start(u32 gpio_pin, hal_int_cb cb, void *context)`
开启指定 `gpio` 外部中断，并指定中断回调函数
- ◆ `void s907x_hal_gpio_it_stop(u32 gpio_pin)`
关闭指定 `gpio` 外部中断
- ◆ `hal_status_e s907x_hal_gpio_set_io(u32 gpio_pin, u8 io)`
设定指定的 `gpio` 工作模式
- ◆ `hal_status_e s907x_hal_gpio_set_pull(u32 gpio_pin, u8 pull)`
设定指定 `gpio`, 输出模式时的默认电平状态



2.2.1 hal_status_e s907x_hal_gpio_init(u32 gpio_pin, gpio_init_t *init);

功能: 对 gpio 口的初始化

参数:

类型	描述
u32	gpio_pin: 指定的 gpio 端口
gpio_init_t	init: 配置参数的指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 将 gpio7 配置为输入模式, 且下降沿触发中断

```
u32 pin = BIT(7);           /*表示 gpio7*/
gpio_init_t init;           /*定义一个参数配置的容器*/
init.mode = GPIO_MODE_INT_RISING; /*配置为输入, 且下降沿中断模式*/
init.pull = GPIO_PULLUP; /*配置初始电平状态为上拉*/
s907x_hal_gpio_init(pin, &init); /*将配置好的信息写入寄存器*/
```

2.2.2 hal_status_e s907x_hal_gpio_deinit(u32 gpio_pin);

功能: 将指定 gpio 口恢复为缺省值状态

参数:

类型	描述
u32	gpio_pin: 指定的 gpio 端口

返回:

类型	描述

hal_status_e	HAL_OK: 操作成功, 其他值均为失败
--------------	-----------------------

示例: 将 gpio7 恢复为缺省值状态

```
u32 pin = BIT(7);           /*表示 gpio7*/  
s907x_hal_gpio_deinit(pin); /*将 gpio7 寄存器状态恢复为缺省值*/
```

2.2.3 hal_status_e s907x_hal_gpio_write(u32 gpio_pin, gpio_status_e status);

功能: 设定指定 gpio 口的输出的电平状态

注意: 调用此函数前, 必须保证所操作的 IO 已被初始化为输出模式

参数:

类型	描述
u32	gpio_pin: 指定的 gpio 端口
gpio_status_e	status: 设定的状态; GPIO_PIN_RESET or GPIO_PIN_SET

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 假设 gpio6 已经为输出模式, 且当前输出低电平状态; 尝试用 hal_gpio_write, 改变 gpio6 的输出状态

```
u32 pin = BIT(6);           /*表示 gpio6*/  
s907x_hal_gpio_write(pin, GPIO_PIN_SET); /*让 gpio6 输出高电平*/
```

2.2.4 gpio_status_e s907x_hal_gpio_read(u32 gpio_pin);

功能: 读取指定 gpio 口的输入的电平状态

注意: 调用此函数前, 必须保证所操作的 IO 已被初始化为输入模式

参数:



类型	描述
u32	gpio_pin: 指定的 gpio 端口

返回:

类型	描述
gpio_status_e	返回 GPIO_PIN_RESET or GPIO_PIN_SET

示例: 读取 gpio7 的电平状态

```
gpio_status_e sta;           /*表示 gpio 状态*/  
u32 pin = BIT(7);           /*表示 gpio7*/  
sta = s907x_hal_gpio_read(pin);    /*读取 gpio7 当前输入的电平状态*/
```

2.2.5 hal_status_e s907x_hal_gpio_togglepin(u32 gpio_pin);

功能: 翻转指定 gpio 口输出的电平状态

注意: 调用此函数前, 必须保证所操作的 IO 已被初始化为输出模式

参数:

类型	描述
u32	gpio_pin: 指定的 gpio 端口

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 假设 gpio6 已经为输出模式, 且当前输出低电平状态; 尝试用 hal_gpio_togglepin, 改变 gpio6 的输出状态

```
u32 pin = BIT(6);           /*表示 gpio6*/  
s907x_hal_gpio_togglepin(pin);    /*翻转 gpio6 输出的电平状态*/
```



2.2.6 void s907x_hal_gpio_it_start(u32 gpio_pin, hal_int_cb cb, void *context);

功能: 开启指定 gpio 外部中断，并指定中断回调函数

注意: 调用此函数前，必须保证所操作的 IO 已被初始化为外部中断模式

参数:

类型	描述
u32	gpio_pin: 指定的 gpio 端口
hal_int_cb	cb: 中断回调函数
void	context: 任意类型的指针，指向回调函数的参数

返回: 无

示例: 假设 gpio7 已经为输入模式，且为下降沿触发中断模式

```
void pin7_isr_cb(void *context)                                /*gpio7 中断回调实现*/  
{  
    .....  
}  
u32 pin = BIT(7);                                              /*表示 gpio7*/  
s907x_hal_gpio_it_start(pin , pin7_isr_cb, NULL);   /*开启 gpio7 外部中断*/  
                                                       /*pin7_isr_cb 为中调回调入口*/  
                                                       /*NULL 表示中调回调不需任何参数*/
```

2.2.7 void s907x_hal_gpio_it_stop(u32 gpio_pin);

功能: 关闭指定 gpio 外部中断

注意: 所操作的 IO 已被初始化为外部中断模式，且已经使能中断，调用该函数失能中断

参数:

类型	描述

u32	gpio_pin: 指定的 gpio 端口
-----	-----------------------

返回: 无

示例: 假设 gpio7 已经为输入模式, 且为下降沿触发中断模式, 并已经开启

```
u32 pin = BIT(7);           /*表示 gpio7*/  
s907x_hal_gpio_it_stop(pin);    /*失能 gpio7 外部中断*/
```

2.2.8 hal_status_e s907x_hal_gpio_set_io(u32 gpio_pin, u8 io);

功能: 设定指定的 gpio 工作模式

参数:

类型	描述
u32	gpio_pin: 指定的 gpio 端口
u8	0: 普通输入模式 1: 输出模式 2: 输入模式, 上升沿触发中断 3: 输入模式, 下降沿触发中断 4: 输入模式, 高电平触发中断 5: 输入模式, 低电平触发中断

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 设定 gpio6 为输出模式

```
u32 pin = BIT(6);           /*表示 gpio6*/  
s907x_hal_gpio_set_io(pin, 1);    /*设定 gpio6 为输出模式*/
```

2.2.9 hal_status_e s907x_hal_gpio_set_pull(u32 gpio_pin, u8 pull);

功能：设定指定 gpio,输出模式时的默认电平状态

参数：

类型	描述
u32	gpio_pin: 指定的 gpio 端口
u8	0: 浮空 1: 上拉 2: 下拉

返回：

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例：设定 gpio6 默认的电平为上拉状态

```
u32 pin = BIT(6); /*表示 gpio6*/  
s907x_hal_gpio_set_io(pin, 1); /*设定 gpio6 为输出模式*/  
s907x_hal_gpio_set_pull(pin, 1); /*表示 gpio6 默认为上拉状态*/
```

2.3 FLASH

◆ `int s907x_hal_flash_write(u32 addr, u8 *pbuf, int len)`

flash 写函数, 在 *flash* 指定的地址开始写入指定长度的数据

◆ `int s907x_hal_flash_read(u32 addr, u8 *pbuf, int len)`

flash 读函数, 在 *flash* 指定的地址开始读出指定长度的数据

◆ `void s907x_hal_flash_erase(int erase_type, u32 addr)`

flash 擦除函数



2.3.1 void s907x_hal_flash_write(u32 addr, u8 *pbuff, int len);

功能: flash 写函数, 在 flash 指定的地址开始写入指定长度的数据

注意: 1,写入的地址必须为不影响系统的, 空闲的 FLASH 地址

2,flash 的写入以 sector 单位写入的, 用户想要在 flash 合法区域连续写入数据
需要自封装

参数:

类型	描述
u32	addr: 要写入数据 flash 的地址
u8	pbuff: 指向待写入数据的指针
int	len: 写入数据的长度

返回:

类型	描述
void	无

示例: 从 flash 地址 0x18002500 开始写入"hello,s907a!"

```
/*准备数据*/  
u8 txbuffer[] = "hello,s907a!";  
/*写入*/  
s907x_hal_flash_write(0x18002500, txbuffer, sizeof(txbuffer));  
/*
```

注意: 如果这样操作的话, 将导致 0x18002000-0x18002500 之间的数据丢失。
。如何保证数据不丢失的按照地址递增连续写入, 需要用户自行处理, 可参考
S9070x FLASH READ&WRITE Reference 中的示例程序。
*/



2.3.2 void s907x_hal_flash_read(u32 addr, u8 *pbuff, int len);

功能: flash 读函数, 在 flash 指定的地址开始读出指定长度的数据

参数:

类型	描述
u32	addr: 要读出数据 flash 的地址
u8	pbuff: 指向保存读出数据缓冲区的地址
int	len: 读出指定长度的数据

返回:

类型	描述
void	无

示例: 从 flash 地址 0x18002500 开始读出 12 个字节的数据

```
/*准备缓冲区*/  
u8 rxbuffer[12] = {0};  
  
/*从 0x18002500 开始读出 12 个字节数据保存在 rxbuffer 中*/  
s907x_hal_flash_read(0x18002500, rxbuffer, 12);
```

2.3.3 void s907x_hal_flash_erase(int erase_type, u32 addr);

功能: flash 擦除函数

参数:

类型	描述
int	erase_type: flash 擦除类型选择 EraseChip: 0 代表芯片全擦除 EraseBlock: 1 代表 block 方式擦除



	EraseSector: 2 代表 sector 方式擦除
u32	addr: 指要擦除区域的起始地址

示例: 以 EraseSector 方式, 将 flash 的 0x18002000 - 0x18003000 擦除

```
s907x_hal_flash_erase(EraseSector, 0x18002000);
```

2.4 SPI

◆ `hal_status_e s907x_hal_spi_init(spi_hdl_t *spi)`

对指定 spi 初始化

◆ `hal_status_e s907x_hal_spi_deinit(spi_hdl_t *spi)`

将指定 spi 配置信息恢复为缺省值

◆ `int s907x_hal_spi_master_txrx(spi_hdl_t *spi, void *txbuf, void *rxbuf, u16 xfer_size, uint32_t ms)`

spi 主设备通过查询的方式发和和接收指定长度的数据

◆ `int s907x_hal_spi_slaver_tx(spi_hdl_t *spi, void *txbuf, u16 xfer_size, uint32_t timeout)`

spi 从设备通过查询的方式发送指定长度的数据

◆ `int s907x_hal_spi_slaver_rx(spi_hdl_t *spi, void *rxbuf, u16 xfer_size, uint32_t timeout)`

spi 从设备通过查询的方式接收指定长度的数据

◆ `hal_status_e s907x_hal_spi_master_xfer_interrupt(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size)`

通过中断方式 spi 主设备发送固定字节数据

◆ `hal_status_e s907x_hal_spi_master_recv_interrupt(spi_hdl_t *spi, void *pbuf, u16 xfer_size)`

通过中断方式 spi 主设备接收固定字节数据

◆ `hal_status_e s907x_hal_spi_slaver_xfer_interrupt(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size)`

通过中断方式 spi 从设备发送固定字节数据

◆ `hal_status_e s907x_hal_spi_slaver_recv_interrupt(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size)`

通过中断方式 spi 从设备接收固定字节数据

◆ `hal_status_e s907x_hal_spi_master_xfer_dma(spi_hdl_t *spi, void *pbuf, u16 xfer_size)`

通过 dma 方式 spi 主设备发送固定字节数据



◆ `hal_status_e s907x_hal_spi_master_recv_dma(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size)`

通过 `dma` 方式 `spi` 主设备接收固定字节数据

◆ `hal_status_e s907x_hal_spi_slaver_xfer_dma(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size)`

通过 `dma` 方式 `spi` 从设备发送固定字节数据

◆ `hal_status_e s907x_hal_spi_slaver_recv_dma(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size)`

通过 `dma` 方式 `spi` 从设备接收固定字节数据

2.4.1 `hal_status_e s907x_hal_spi_init(spi_hdl_t *spi);`

功能: 对指定 `spi` 模块初始化

参数:

类型	描述
<code>spi_hdl_t</code>	<code>spi</code> : 指向 <code>spi</code> 实例句柄指针

返回:

类型	描述
<code>hal_status_e</code>	<code>HAL_OK</code> : 操作成功, 其他值均为失败

示例: 对 `spi1` 初始化, 配置为 `master`

```
spi_hdl_t spi_master_hd;           /* 声明 spi 实例句柄 */  
spi_hdl_t *p_spi_hdl;             /* 声明一个 spi_hdl_t 类型指针 */  
p_spi_hdl = &spi_master_hd;        /* 指向声明的 spi 句柄 */  
p_spi_hdl->config.idx = SPI_IDX_1; /* 配置对象 spi1 */  
p_spi_hdl->config.spi_master = HAL_MASTER_SEL; /* 配置为主设备 */  
p_spi_hdl->config.datalen = SPI_DATASIZE_8BIT; /* datasize 配置 8bit */  
p_spi_hdl->config.clk_phase = SPI_PHASE_1EDGE; /* 配置同步时钟相位 */  
p_spi_hdl->config.clk_polarity = SPI_POLARITY_LOW; /* 配置同步时钟极性 */
```



```
p_spi_hdl->config.clk_speed = 2500000; /*配置传输速率 2.5M*/  
p_spi_hdl->config.mode = SPI_MODE_TXRX/*配置当前设备的收发模式*/
```

如果应用中选择中断或者 dma 方式，那么还要配置回调信息，这里已中断为例：

```
spi_hdl_t->it.tx_complete.func= master_tx_int;  
/*master_tx_int 为发送完成回调函数*/  
spi_hdl_t->it.tx_complete.context = spi_hdl_t;  
spi_hdl_t->it.rx_complete.func = master_rx_int;  
/*master_rx_int 为接收完成回调函数*/  
spi_hdl_t->it.rx_complete.context = spi_hdl_t;  
s907x_hal_spi_init(p_spi_hdl); /*将配置信息写入寄存器*/
```

2.4.2 hal_status_e s907x_hal_spi_deinit(spi_hdl_t *spi);

功能： 将指定 spi 配置信息恢复为缺省值

参数：

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针

返回：

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例： 将上述示例中 spi1 恢复为缺省状态

```
spi_hdl_t *p_spi_hdl; /*声明一个 spi_hdl_t 类型指针*/  
p_spi_hdl = &spi_master_hd;  
s907x_hal_spi_deinit(p_spi_hdl); /*将 spi1 恢复为缺省值*/
```

2.4.3 int s907x_hal_spi_master_txrx(spi_hdl_t *spi, void *txbuf, void *rdbuf, u16 xfer_size, uint32_t ms);

功能: spi 主设备通过查询的方式发和和接收指定长度的数据

注意: 调用此函数前必须已经将 spi 初始化为主设备

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
void	txbuf: 指向待发送数据缓冲区的指针
void	rdbuf: 指向待接收数据缓冲区的指针
u16	xfer_size: 传输数据的大小
uint32_t	ms: poll 等待的最大时间

返回:

类型	描述
int	本次传输中实际传输完成的数据量

示例: spi 主设备向从设备发送"hello";

spi 主设备的句柄假设为 spi_master_hd; (这里初始化过程就省略了)

```
/*声明一个 spi_hdl_t 类型指针*/  
spi_hdl_t *p_spi_hdl;  
  
/*指向 spi 主设备句柄*/  
p_spi_hdl = &spi_master_hd;  
  
/*待发送的数据*/  
u8 txbuff[] = "hello";  
  
/*spi 主设备发送数据*/  
s907x_hal_spi_master_txrx(p_spi_hdl, txbuff, NULL, sizeof(txbuff), 0xffffffff);
```



2.4.4 int s907x_hal_spi_slaver_tx(spi_hdl_t* spi, void *txbuf, u16 xfer_size, uint32_t timeout);

功能: spi 从设备通过查询的方式发送指定长度的数据

注意: 调用此函数前必须已经将 spi 初始化为从设备

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
void	txbuf: 指向待发送数据缓冲区的指针
u16	xfer_size: 要发送数据的长度
uint32_t	timeout: poll 查询的最长等待时间

返回:

类型	描述
int	本次传输中实际传输完成的数据量

示例: spi 从设备向主设备发送"hello";

spi 从设备的句柄假设为 spi_slaver_hd; (这里初始化过程就省略了)

```
/*声明一个 spi_hdl_t 类型指针*/  
spi_hdl_t *p_spi_hdl;  
/*指向 spi 从设备句柄*/  
p_spi_hdl = &spi_slaver_hd;  
/*待发送的数据*/  
u8 txbuffer[] = "hello";  
/*spi 从设备发送数据*/  
s907x_hal_spi_slaver_tx(p_spi_hdl, txbuffer, sizeof(txbuffer), 0xffffffff);
```

2.4.5 int s907x_hal_spi_slaver_rx(spi_hdl_t* spi, void *rxbuf, u16 xfer_size, uint32_t timeout);

功能: spi 从设备通过查询的方式接收指定长度的数据

注意: 调用此函数前必须已经将 spi 初始化为从设备

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
void	rxbuf: 指向接收数据缓冲区的指针
u16	xfer_size: 要接收数据的长度
uint32_t	timeout: poll 查询的最长等待时间

返回:

类型	描述
int	本次传输中实际传输完成的数据量

示例: spi 从设备接收主设备发送 10 字节数据;

spi 从设备的句柄假设为 spi_slaver_hd; (这里初始化过程就省略了)

```
/*声明一个 spi_hdl_t 类型指针*/  
spi_hdl_t *p_spi_hdl;  
/*指向 spi 从设备句柄*/  
p_spi_hdl = &spi_slaver_hd;  
/*待接收数据缓冲区*/  
u8 rxbuffer[10];  
/*spi 从设备接收数据*/  
s907x_hal_spi_slaver_rx(p_spi_hdl, rxbuffer, 10,0xffff);
```



2.4.6 hal_status_e s907x_hal_spi_master_xfer_interrupt(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size);

功能: 通过中断方式 spi 主设备发送固定字节数据

注意: 调用此函数前必须已经将 spi 初始化为主设备，并且已经配置中断回调函数

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
u8	pbuf: 指向待发送数据缓冲区指针
u16	xfer_size: 发送数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: 发送"hello"给从设备

spi 主设备的句柄假设为 spi_master_hd; (这里初始化过程就省略了)

```
/*声明一个 spi_hdl_t 类型指针*/  
spi_hdl_t *p_spi_hdl;  
/*指向 spi 主设备句柄*/  
p_spi_hdl = &spi_master_hd;  
/*待发送的数据*/  
u8 pbuffer[] = "hello";  
/*spi 主设备发送数据*/  
s907x_hal_spi_master_xfer_interrupt(p_spi_hdl, pbuffer, sizeof(pbuffer));
```

2.4.7 hal_status_e s907x_hal_spi_master_recv_interrupt(spi_hdl_t *spi, void *pbuf, u16 xfer_size);

功能: 通过中断方式 spi 主设备接收固定字节数据

注意: 调用此函数前必须已经将 spi 初始化为主设备，并且已经配置中断回调函数

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
void	pbuf: 指向待接收缓冲区的指针
u16	xfer_size: 待接收数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: 接收 10 字节数据

spi 主设备的句柄假设为 spi_master_hd; (这里初始化过程就省略了)

```
/* 声明一个 spi_hdl_t 类型指针 */
spi_hdl_t *p_spi_hdl;
/* 指向 spi 主设备句柄 */
p_spi_hdl = &spi_master_hd;
/* 定义接收数据缓冲区 */
u8 rebuffer[10];
/* spi 主设备接收 10 个字节的数据 */
s907x_hal_spi_master_recv_interrupt(p_spi_hdl, rebuffer, 10);
```



2.4.8 hal_status_e s907x_hal_spi_slaver_xfer_interrupt(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size);

功能: 通过中断方式 spi 从设备发送固定字节数据

注意: 调用此函数前必须已经将 spi 初始化为从设备，并且已经配置中断回调函数

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
u8	pbuf: 指向待发送缓冲区指针
u16	xfer_size: 要发送的数据量

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: 向 spi 主设备发送“hello”

spi 从设备的句柄假设为 spi_slaver_hd; (这里初始化过程就省略了)

```
/* 声明一个 spi_hdl_t 类型指针 */
spi_hdl_t *p_spi_hdl;

/* 指向 spi 从设备句柄 */
p_spi_hdl = &spi_slaver_hd;

/* 指向 spi 主设备句柄 */
u8 txbuffer = "hello";
/* spi 从设备发送数据给主设备 */
s907x_hal_spi_slaver_xfer_interrupt(p_spi_hdl, txbuffer, sizeof(txbuffer));
```



2.4.9 hal_status_e s907x_hal_spi_slaver_recv_interrupt(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size);

功能: 通过中断方式 spi 从设备接收固定字节数据

注意: 调用此函数前必须已经将 spi 初始化为从设备，并且已经配置中断回调函数

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
u8	pbuf: 指向待接收数据缓冲区的指针
u16	xfer_size: 接收数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: spi 从设备接收来自自主设备的 10 个字节数据

spi 从设备的句柄假设为 spi_slaver_hd; (这里初始化过程就省略了)

```
/* 声明一个 spi_hdl_t 类型指针 */  
spi_hdl_t *p_spi_hdl;  
  
/* 指向 spi 从设备句柄 */  
p_spi_hdl = &spi_slaver_hd;  
  
/* 待接收数据缓冲区 */  
u8 rxbuffer[10];  
  
/* 接收 10 个字节的数据 */  
s907x_hal_spi_slaver_recv_interrupt(p_spi_hdl, rxbuffer, 10);
```



2.4.10 hal_status_e s907x_hal_spi_master_xfer_dma(spi_hdl_t *spi, void *pbuf, u16 xfer_size);

功能: 通过 dma 方式 spi 主设备发送固定字节数据

注意: 调用此函数前必须已经将 spi 初始化为主设备，并且已经配置 dma 回调函数

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
void	pbuf: 指向待发送数据缓冲区的指针
u16	xfer_size: 待发送数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: spi 主设备发送"hello"给从设备

spi 主设备的句柄假设为 spi_master_hd; (这里初始化过程就省略了)

```
/*声明一个 spi_hdl_t 类型指针*/  
spi_hdl_t *p_spi_hdl;  
  
/*指向 spi 主设备句柄*/  
p_spi_hdl = &spi_master_hd;  
  
/*定义待发送的数据*/  
u8 txbuffer[] = "hello";  
  
/*通过 dma 方式， spi 主设备将数据发送出去*/  
s907x_hal_spi_master_xfer_dma(p_spi_hdl, txbuffer, sizeof(txbuffer));
```

2.4.11 hal_status_e s907x_hal_spi_master_recv_dma(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size);

功能: 通过 dma 方式 spi 主设备接收固定字节数据

注意: 调用此函数前必须已经将 spi 初始化为主设备，并且已经配置 dma 回调函数

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
u8	pbuf: 指向待接收数据缓冲区的指针
u16	xfer_size: 待接收数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: spi 主设备接收从设备 10 个字节的数据

spi 主设备的句柄假设为 spi_master_hd; (这里初始化过程就省略了)

```
/*声明一个 spi_hdl_t 类型指针*/\n\nspi_hdl_t *p_spi_hdl;\n\n/*指向 spi 主设备句柄*/\n\np_spi_hdl = &spi_master_hd;\n\n/*待接收数据缓冲区*/\n\nu8 rxbuffer[10];\n\n/*spi 主设备接收从机数据*/\ns907x_hal_spi_master_recv_dma(p_spi_hdl, rxbuffer, 10);
```

2.4.12 hal_status_e s907x_hal_slaver_xfer_dma(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size);

功能: 通过 dma 方式 spi 从设备发送固定字节数据

注意: 调用此函数前必须已经将 spi 初始化为从设备，并且已经配置 dma 回调函数

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
u8	pbuf: 指向待发送数据缓冲区指针
u16	xfer_size: 要发送数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: spi 从设备发送"hello"给主设备

spi 从设备的句柄假设为 spi_slaver_hd; (这里初始化过程就省略了)

```
/*声明一个 spi_hdl_t 类型指针*/  
spi_hdl_t *p_spi_hdl;  
  
/*指向 spi 从设备句柄*/  
p_spi_hdl = &spi_slaver_hd;  
  
/*要发送出去的数据*/  
u8 txbuffer[] = "hello";  
  
/*spi 从设备发送数据*/  
s907x_hal_slaver_xfer_dma(p_spi_hdl, txbuffer, sizeof(txbuffer));
```



2.4.13 hal_status_e s907x_hal_slaver_recv_dma(spi_hdl_t *spi, u8 *pbuf, u16 xfer_size);

功能: 通过 dma 方式 spi 从设备接收固定字节数据

注意: 调用此函数前必须已经将 spi 初始化为从设备，并且已经配置 dma 回调函数

参数:

类型	描述
spi_hdl_t	spi: 指向 spi 实例句柄指针
u8	pbuf: 指向待接收数据缓冲区指针
u16	xfer_size: 要接收数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: spi 从设备接收主设备 10 字节数据

spi 从设备的句柄假设为 spi_slaver_hd; (这里初始化过程就省略了)

```
/*声明一个 spi_hdl_t 类型指针*/\n\nspi_hdl_t *p_spi_hdl;\n\n/*指向 spi 从设备句柄*/\n\np_spi_hdl = &spi_slaver_hd;\n\n/*定义接收数据缓冲区*/\n\nu8 rxbuffer[10];\n\n/*spi 从设备接收固定字节数据*/\ns907x_hal_slaver_recv_dma(p_spi_hdl, rxbuffer, 10);
```

2.5 I2C

- ◆ [hal_status_e s907x_hal_i2c_init\(i2c_hdl_t *i2c\)](#)



对指定 *i2c* 设备初始化

- ◆ `hal_status_e s907x_hal_i2c_deinit(i2c_hdl_t *i2c)`

将指定 *i2c* 设备配置恢复为缺省值

- ◆ `hal_status_e s907x_hal_i2c_master_xfer(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms)`

i2c 主设备通过查询的方式发送指定长度的数据

- ◆ `hal_status_e s907x_hal_i2c_master_recv(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms)`

i2c 主设备通过查询的方式接收指定长度的数据

- ◆ `hal_status_e s907x_hal_i2c_slavor_xfer(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms)`

i2c 从设备通过查询的方式发送指定长度的数据

- ◆ `hal_status_e s907x_hal_i2c_slavor_recv(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms)`

i2c 从设备通过查询的方式接收主设备指定长度的数据

- ◆ `hal_status_e s907x_hal_i2c_master_xfer_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size)`

i2c 主设备通过中断的方式发送给从设备指定长度的数据

- ◆ `hal_status_e s907x_hal_i2c_master_recv_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size)`

i2c 主设备通过中断的方式接收来自从设备指定长度的数据

- ◆ `hal_status_e s907x_hal_i2c_slavor_xfer_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size)`

i2c 从设备通过中断的方式发送指定长度的数据给主设备

- ◆ `hal_status_e s907x_hal_i2c_slavor_recv_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size)`

i2c 从设备通过中断的方式接收主设备指定长度的数据

- ◆ `hal_status_e s907x_hal_i2c_master_xfer_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size)`

i2c 主设备通过 *dma* 的方式发送指定长度的数据给从设备

- ◆ `hal_status_e s907x_hal_i2c_master_recv_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size)`

i2c 主设备通过 *dma* 的方式接收从设备指定长度的数据

- ◆ `hal_status_e s907x_hal_i2c_slavor_xfer_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size)`

i2c 从设备通过 *dma* 的方式发送指定字节的数据给主设备

- ◆ `hal_status_e s907x_hal_i2c_slavor_recv_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size)`



i2c 从设备通过 dma 的方式接收主设备指定字节的数据

2.5.1 hal_status_e s907x_hal_i2c_init(i2c_hdl_t *i2c);

功能: 对指定 i2c 设备初始化

参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 对 i2c0 初始化, 配置为 master

```
/* 声明 i2c 实例句柄 */
i2c_hdl_t i2c_master_hd;

/* 声明一个 i2c_hdl_t 类型的指针 */
i2c_hdl_t *p_i2c_master;

/* 指向 i2c 句柄 */
p_i2c_master = &i2c_master_hd;

/* 配置对象为 i2c0 */

p_i2c_master->config.idx = I2C_IDX_0;

/* 配置地址为 7bit 模式 */
p_i2c_master->config.addr_mode = I2C_ADDR_MODE_7BIT;

/* 配置速率为 400k 模式 */

p_i2c_master->config.clock = I2C_MASTER_CLK_400K;

/* 配置传输方向 */

p_i2c_master->config.dir = HAL_DIR_TX;

/* 配置是否 general call, 0 代表否 */
```

```
p_i2c_master->config.general_call = 0;  
/*配置为 master 身份*/  
  
p_i2c_master->config.i2c_master = HAL_MASTER_SEL;  
/*配置主设备自己地址为 0x55*/  
  
p_i2c_master->config.own_addr = 0x55;  
/*配置通信目标设备的地址为 0x66*/  
  
p_i2c_master->config.target_addr = 0x66;  
  
/*  
如果应用中选择中断方式，那么还要配置回调信息  
  
p_i2c_master->it.rx_complete.func = rx_interrupt_hdl;  
/*中断方式接收完成回调函数*/  
  
p_i2c_master->it.tx_complete.func = tx_interrupt_hdl;  
/*中断方式发送完成回调函数*/  
  
p_i2c_master->it.rx_complete.context = p_i2c_master;  
p_i2c_master->it.tx_complete.context = p_i2c_master;  
  
*/  
/*将配置信息写入寄存器*/  
  
s907x_hal_i2c_init(p_i2c_master);
```

2.5.2 hal_status_e s907x_hal_i2c_deinit(i2c_hdl_t *i2c);

功能: 将指定 i2c 设备配置恢复为缺省值

参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 将 i2c0 恢复为缺省值状态

```
/*声明一个 i2c_hdl_t 类型指针*/
i2c_hdl_t *p_i2c_master;

/*该指针指向操作的实例句柄*/
p_i2c_master = &i2c_master_hd;

/*将 i2c0 恢复为缺省状态*/
s907x_hal_i2c_deinit(p_i2c_master);
```

2.5.3 hal_status_e s907x_hal_i2c_master_xfer(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms);

功能: i2c 主设备通过查询的方式发送指定长度的数据

注意: 调用此函数前必须已经完成 i2c 主设备的初始化

参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待发送数据缓冲区指针
u16	xfer_size: 待发送数据的长度
uint32_t	ms: 超时等待的时间

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: i2c 主设备向从设备发送"hello"

I2c 主设备的句柄假设为 i2c_master_hd; (这里初始化过程就省略了)

```
/*声明一个 i2c_hdl_t 类型的指针*/  
i2c_hdl_t *p_i2c_master;  
  
/*指向 I2C 主设备句柄*/  
p_i2c_master = &i2c_master_hd;  
  
/*准备好待发送的数据*/  
u8 txbuffer[] = "hello";  
  
/*通过查询的方式将数据发送出去*/  
s907x_hal_i2c_master_xfer(p_i2c_master, txbuffer, sizeof(txbuffer), 1000);
```

2.5.4 hal_status_e s907x_hal_i2c_master_recv(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms);

功能: I2C 主设备通过查询的方式接收指定长度的数据

注意: 调用此函数前必须已经完成 I2C 主设备的初始化

参数:

类型	描述
i2c_hdl_t	i2c: 指向 I2C 实例句柄指针
u8	pbuf: 指向待接收数据缓冲区的指针
u16	xfer_size: 指定接收数据的长度
uint32_t	ms: 超时等待的时间

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: I2C 主设备接收从设备 10 字节的数据

I2C 主设备的句柄假设为 i2c_master_hd; (这里初始化过程就省略了)

```

/*声明一个 i2c_hdl_t 类型的指针*/
i2c_hdl_t *p_i2c_master;

/*指向 i2c 主设备句柄*/
p_i2c_master = &i2c_master_hd;

/*准备好接受数据缓冲区*/
u8 rxbuffer[10];

/*通过查询的方式 i2c 主设备接收来自从设备 10 字节数据*/
s907x_hal_i2c_master_recv(p_i2c_master, rxbuffer, 10, 1000);

```

2.5.5 hal_status_e s907x_hal_i2c_slavor_xfer(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms);

功能: i2c 从设备通过查询的方式发送指定长度的数据

注意: 调用此函数前必须已经完成 i2c 从设备的初始化

参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待发送数据缓冲区指针
u16	xfer_size: 指定发送数据的长度
uint32_t	ms: 超时等待的时间

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: i2c 从设备发送"hello"给主设备

i2c 从设备的句柄假设为 i2c_slaver_hd; (这里初始化过程就省略了)

```

/*声明一个 i2c_hdl_t 类型的指针*/

```

```
i2c_hdl_t *p_i2c_slaver;  
/*指向 i2c 从设备句柄*/  
p_i2c_slaver = &i2c_slaver_hd;  
/*准备好要发送的数据*/  
u8 txbuffer[] = "hello";  
/*通过查询的方式 i2c 从设备向主设备发送"hello"*/  
s907x_hal_i2c_slavor_xfer(p_i2c_slaver, txbuffer, sizeof(txbuffer), 1000);
```

2.5.6 hal_status_e s907x_hal_i2c_slavor_recv(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size, uint32_t ms);

功能: i2c 从设备通过查询的方式接收主设备指定长度的数据

注意: 调用此函数前必须已经完成 i2c 从设备的初始化

参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待接收数据缓冲区指针
u16	xfer_size: 指定接收数据的长度
uint32_t	ms: 超时等待的时间

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: i2c 从设备接收主设备 10 字节数据

i2c 从设备的句柄假设为 i2c_slaver_hd; (这里初始化过程就省略了)

```
/*声明一个 i2c_hdl_t 类型的指针*/
```

```
i2c_hdl_t *p_i2c_slaver;
```



```
/*指向 i2c 从设备的句柄*/
p_i2c_slaver = &i2c_slaver_hd;
/*准备好待接收数据缓冲区*/
u8 rxbuffer[10];
/*通过查询的方式 i2c 从设备接收来自主设备的 10 字节数据*/
s907x_hal_i2c_slavor_recv(p_i2c_slaver, rxbuffer, 10, 1000);
```

2.5.7 hal_status_e s907x_hal_i2c_master_xfer_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);

功能: i2c 主设备通过中断的方式发送给从设备指定长度的数据

注意: 调用此函数前必须已经完成 i2c 主设备的初始化，并且已经完成中断回调函数配置

参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待发送数据缓冲区指针
u16	xfer_size: 指定发送数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: i2c 主设备发送"hello"给从设备

i2c 主设备的句柄假设为 i2c_master_hd (这里初始化过程就省略了)

```
/*声明一个 i2c_hdl_t 类型指针*/
```

```
i2c_hdl_t *p_i2c_master;
```

```
/*指向 i2c 主设备句柄*/
```

```
p_i2c_master = &i2c_master_hd;
```

```
/*准备好要发送出去的数据*/  
u8 txbuffer[] = "hello";  
  
/*通过中断的方式 i2c 主设备向从设备发送 hello*/  
s907x_hal_i2c_master_xfer_interrupt(p_i2c_master, txbuffer, sizeof(txbuffer));
```

2.5.8 hal_status_e s907x_hal_i2c_master_recv_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);

功能: i2c 主设备通过中断的方式接收来自从设备指定长度的数据

注意: 调用此函数前必须已经完成 i2c 主设备的初始化, 并且已经完成中断回调函数配置

参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待接收数据缓冲区指针
u16	xfer_size: 指定接收数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: i2c 主设备接收从设备 10 字节数据

i2c 主设备的句柄假设为 i2c_master_hd (这里初始化过程就省略了)

```
/*声明一个 i2c_hdl_t 类型的指针*/  
i2c_hdl_t *p_i2c_master;  
  
/*指向 i2c 主设备句柄*/  
  
p_i2c_master = &i2c_master_hd;  
  
/*准备好接收数据缓冲区*/  
  
u8 rxbuffer[10];  
  
/*通过中断的方式 i2c 主设备接收从设备 10 字节数据*/
```

```
s907x_hal_i2c_master_recv_interrupt(p_i2c_master, rxbuffer, 10);
```

2.5.9 hal_status_e s907x_hal_i2c_slavor_xfer_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);

功能: i2c 从设备通过中断的方式发送指定长度的数据给主设备

注意: 调用此函数前必须已经完成 i2c 从设备的初始化，并且已经完成中断回调函数配置
参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待发送数据缓冲区指针
u16	xfer_size: 指定发送数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: i2c 从设备发送"hello"给主设备

i2c 从设备的句柄假设为 i2c_slaver_hd (这里初始化过程就省略了)

```
/*声明一个 i2c_hdl_t 类型的指针*/  
i2c_hdl_t *p_i2c_slaver;  
/*指向 i2c 从设备句柄*/  
p_i2c_slaver = &i2c_slaver_hd;  
/*准备好要发送的数据*/  
u8 txbuffer[] = "hello";  
/*通过中断方式 i2c 从设备发送"hello"给主设备*/  
s907x_hal_i2c_slavor_xfer_interrupt(p_i2c_slaver, txbuffer, sizeof(txbuffer));
```



2.5.10 hal_status_e s907x_hal_i2c_slavor_recv_interrupt(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);

功能: i2c 从设备通过中断的方式接收主设备指定长度的数据

注意: 调用此函数前必须已经完成 i2c 从设备的初始化，并且已经完成中断回调函数配置

参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待接收数据缓冲区指针
u16	xfer_size: 指定接收数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例: i2c 从设备接收来自主设备 10 字节数据

i2c 从设备的句柄假设为 i2c_slaver_hd (这里初始化过程就省略了)

```
i2c_hdl_t *p_i2c_slaver;  
p_i2c_slaver = &i2c_slaver_hd;  
u8 rxbuffer[10];  
s907x_hal_i2c_slavor_recv_interrupt(p_i2c_slaver, rxbuffer, 10);
```

2.5.11 hal_status_e s907x_hal_i2c_master_xfer_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);

功能: i2c 主设备通过 dma 的方式发送指定长度的数据给从设备

注意: 调用此函数前必须已经完成 i2c 主设备的初始化，并已完成 dma 回调函数配置



参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待发送数据缓冲区指针
u16	xfer_size: 指定发送数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: i2c 主设备发送"hello"给从设备

i2c 主设备的句柄假设为 i2c_master_hd (这里初始化过程就省略了)

```
/*声明一个 i2c_hdl_t 类型指针*/\n\ni2c_hdl_t *p_i2c_master;\n\n/*指向 i2c 主设备句柄*/\n\np_i2c_master = &i2c_master_hd;\n\n/*准备要发送的数据*/\n\nu8 txbuffer[] = "hello";\n\n/*通过 dma 方式 i2c 主设备发送"hello"给从设备*/\n\nhal_i2c_master_xfer_dma(p_i2c_master, txbuffer, sizeof(txbuffer));
```

2.5.12 hal_status_e s907x_hal_i2c_master_recv_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);

功能: i2c 主设备通过 dma 的方式接收从设备指定长度的数据

注意: 调用此函数前必须已经完成 i2c 主设备的初始化, 并已完成 dma 回调函数配置

参数:



类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待接收数据缓冲区指针
u16	xfer_size: 指定发送数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: i2c 主设备接收从设备 10 字节数据

i2c 主设备的句柄假设为 i2c_master_hd (这里初始化过程就省略了)

```
/*声明一个 i2c_hdl_t 类型的指针*/
i2c_hdl_t *p_i2c_master;
/*指向 i2c 主设备句柄*/
p_i2c_master = &i2c_master_hd;
/*准备待接收数据缓冲区*/
u8 rxbuffer[10];
/*通过 dma 方式主设备接收从设备 10 字节数据*/
s907x_hal_i2c_master_recv_dma(p_i2c_master, rxbuffer ,10);
```

2.5.13 hal_status_e s907x_hal_i2c_slavor_xfer_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);

功能: i2c 从设备通过 dma 的方式发送指定字节的数据给主设备

注意: 调用此函数前必须已经完成 i2c 从设备的初始化, 并已完成 dma 回调函数配置

参数:

类型	描述
----	----



i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待发送数据缓冲区指针
u16	xfer_size: 指定发送数据的长度

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: i2c 从设备发送"hello"给主设备

i2c 从设备的句柄假设为 i2c_slaver_hd (这里初始化过程就省略了)

```
/*声明一个 i2c_hdl_t 类型的指针*/  
i2c_hdl_t *p_i2c_slaver;  
  
/*指向 i2c 从设备句柄*/  
  
p_i2c_slaver = &i2c_slaver_hd;  
  
/*准备好要发送的数据*/  
  
u8 txbuffer[] = "hello";  
  
/*通过 dma 方式 i2c 从设备给主设备发送"hello"*/  
s907x_hal_i2c_slavor_xfer_dma(p_i2c_slaver, txbuffer, sizeof(txbuffer));
```

2.5.14 hal_status_e s907x_hal_i2c_slavor_recv_dma(i2c_hdl_t *hi2c, u8 *pbuf, u16 xfer_size);

功能: i2c 从设备通过 dma 的方式接收主设备指定字节的数据

注意: 调用此函数前必须已经完成 i2c 从设备的初始化, 并已完成 dma 回调函数配置

参数:

类型	描述
i2c_hdl_t	i2c: 指向 i2c 实例句柄指针
u8	pbuf: 指向待接收数据缓冲区指针



u16	xfer_size: 指定接收数据的长度
-----	----------------------

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: i2c 从设备接收主设备 10 字节数据

i2c 从设备的句柄假设为 i2c_slaver_hd (这里初始化过程就省略了)

```
/*声明一个 i2c_hdl_t 类型的指针*/  
i2c_hdl_t *p_i2c_slaver;  
  
/*指向 i2c 从设备句柄*/  
p_i2c_slaver = &i2c_slaver_hd;  
  
/*准备接收数据缓冲区*/  
u8 rxbuffer[10];  
  
/*通过 dma 方式 i2c 从设备接收主设备 10 字节数据*/  
s907x_hal_i2c_slavor_recv_dma(p_i2c_slaver, rxbuffer, 10);
```

2.6 I2S

2.7 ADC

◆ `hal_status_e s907x_hal_adc_init(adc_hdl_t *adc)`

`adc` 功能初始化

◆ `hal_status_e s907x_hal_adc_deinit(adc_hdl_t *adc)`

 将 `adc` 配置恢复为缺省值

◆ `hal_status_e s907x_hal_adc_start_it(adc_hdl_t *adc, u32 mode);`

 开启 `adc` 采样

◆ `hal_status_e s907x_hal_adc_stop_it(adc_hdl_t *adc, u32 mode);`

 停止 `adc` 采样



◆ `hal_status_e s907x_hal_adc_poll_oneshot(adc_hdl_t *adc, u32 timeout)`

查询式一次性读取一组 `adc` 采样数据

◆ `hal_status_e s907x_hal_adc_poll_continuous(adc_hdl_t *adc)`

查询式持续进行 `adc0` 和 `adc1` 采样

◆ `hal_status_e s907x_hal_adc_interrupt_oneshot(adc_hdl_t *adc, hal_int_cb cb, void *arg)`

中断 `adc` 采样一次

◆ `hal_status_e s907x_hal_adc_interrupt_continuous(adc_hdl_t *adc, hal_int_cb cb, void *arg)`

`adc` 采样一次，采样完成后进中断回调函数处理数据

◆ `hal_status_e s907x_hal_adc_get_mv(adc_hdl_t *adc)`

`adc` 采样电压值，单位 `mv`

2.7.1 `hal_status_e s907x_hal_adc_init(adc_hdl_t *adc);`

功能: `adc` 功能初始化

参数:

类型	描述
<code>adc_hdl_t</code>	<code>adc</code> : 指向 <code>adc</code> 实例句柄指针

返回:

类型	描述
<code>hal_status_e</code>	<code>HAL_OK</code> : 操作成功，其他值均为失败

示例: 对 `adc` 初始化

```
/*定义 adc 实例句柄*/
adc_hdl_t adc_hdl;
/*声明一个 adc_hdl_t 类型指针*/
adc_hdl_t *p_adc_hd;
/*指向 adc 实例句柄*/
```

```
p_adc_hd = &adc_hdl;  
/*失能 oneshot 功能*/  
p_adc_hd->config.oneshot.enable = FALSE;  
/*adc 初始化，将配置信息写入寄存器*/  
s907x_hal_adc_init(p_adc_hd);
```

2.7.2 hal_status_e s907x_hal_adc_deinit(adc_hdl_t *adc);

功能: 将 adc 配置恢复为缺省值

参数:

类型	描述
adc_hdl_t	adc: 指向 adc 实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 将 adc 配置恢复为缺省值

```
/*定义 adc 实例句柄*/  
adc_hdl_t adc_hdl;  
/*声明一个 adc_hdl_t 类型指针*/  
adc_hdl_t *p_adc_hd;  
/*指向 adc 实例句柄*/  
p_adc_hd = &adc_hdl;  
/*将 adc 恢复缺省状态*/  
s907x_hal_adc_deinit(p_adc_hd);
```



2.7.3 hal_status_e s907x_hal_adc_start_it(adc_hdl_t *adc, u32 mode);

功能: 开启中断 adc 采样

参数:

类型	描述
adc_hdl_t	adc: 指向 adc 实例句柄指针
u32	mode:模式 - ADC_IT_ONESHOT oneshot 模式下 - ADC_IT_CONTINOUS continues 模式下 - ADC_IT_DMA dma 模式下

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 开启 adc 采样(假设 adc 采样处于 disable 状态)

```
/*定义 adc 实例句柄*/  
adc_hdl_t adc_hdl;  
  
/*声明一个 adc_hdl_t 类型指针*/  
adc_hdl_t *p_adc_hd;  
  
/*指向 adc 实例句柄*/  
p_adc_hd = &adc_hdl;  
  
/*例如在 interrupt continuos 模式下使能 adc 采样, */  
s907x_hal_adc_start_it(p_adc_hd, ADC_IT_CONTINUOUS);
```

2.7.4 hal_status_e s907x_hal_adc_stop_it(adc_hdl_t *adc, u32 mode);

功能: 停止中断模式下 adc 采样



参数:

类型	描述
adc_hdl_t	adc: 指向 adc 实例句柄指针
u32	mode:模式 - ADC_IT_ONESHOT oneshot 模式下 - ADC_IT_CONTINOUS continuos 模式下 - ADC_IT_DMA dma 模式下

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 停止 adc 采样(假设 adc 采样处于 enable 状态)

```
/*定义 adc 实例句柄*/  
adc_hdl_t adc_hdl;  
/*声明一个 adc_hdl_t 类型指针*/  
adc_hdl_t *p_adc_hd;  
/*指向 adc 实例句柄*/  
p_adc_hd = &adc_hdl;  
/*例如 adc 工作在 interrupt continuos 模式下, 失能 adc 采样*/  
s907x_hal_adc_stop_it(p_adc_hd, ADC_IT_CONTINOUS);
```

2.7.5 hal_status_e s907x_hal_adc_poll_oneshot(adc_hdl_t *adc, u32 timeout);

功能: 查询式一次性读取一组 adc 采样数据

注意: 调用此函数前需要完成 adc 初始化, 如下示例

参数:

类型	描述
----	----



adc_hdl_t	adc: 指向 adc 实例句柄指针
u32	timeout: 超时等待的时间, 单位 ms

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 采样一组 adc 数据, 首先需要 adc 配置, 如下

```
/*定义 adc 实例句柄*/
adc_hdl_t adc_hdl;
/*声明一个 adc_hdl_t 类型指针*/
adc_hdl_t *p_adc_hd;
/*指向 adc 实例句柄*/
p_adc_hd = &adc_hdl;
/*使能 oneshot 功能*/
p_adc_hd->config.oneshot.enable = TRUE;
/*采样多组数据, 每组采样之间的时间间隔*/
p_adc_hd->config.oneshot.delay = 200;
/*一组采集 16 个样本, adc0, adc1 各 8 个样本*/
p_adc_hd->config.oneshot.read_nums = ADC_FIFO;
/*将配置信息写入到寄存器中*/
s907x_hal_adc_init(p_adc_hd);
    .
    .
    .
/*循环采样*/
While(1){
    s907x_hal_adc_poll_oneshot(p_adc_hd, HAL_MAX_DELAY);
    /*对采样的数据处理, 这里只是简单的打印出采样值*/
    for(i = 0; i < 8; i++) {
```



```
    HAL_TEST_DBG("cnt = %d adc0  = %x adc1  =%x\n", i,  
                 p_adc_hd->data[i].chn[0], p_adc_hd->data[i].chn[1]);  
}  
  
s907x_hal_adc_stop_it(p_adc_hd);  
}
```

2.7.6 hal_status_e s907x_hal_adc_poll_continuousadc_hdl_t *adc);

功能: 查询式持续进行 adc0 和 adc1 采样

注意: 调用此函数前需要完成对 adc 的初始化

参数:

类型	描述
adc_hdl_t	adc: 指向 adc 实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 查询方式 adc0, adc1 持续采样

```
/*定义 adc 实例句柄*/  
adc_hdl_t adc_hdl;  
  
/*声明一个 adc_hdl_t 类型指针*/  
adc_hdl_t *p_adc_hd;  
  
/*指向 adc 实例句柄*/  
p_adc_hd = &adc_hdl;  
  
/*失能 adc 的 oneshot 功能*/  
p_adc_hd->config.oneshot.enable = FALSE;  
  
/*将配置信息写入寄存器*/  
s907x_hal_adc_init(p_adc_hd );
```

```
•  
•  
•  
/*循环采样*/  
  
While(1){  
  
    s907x_hal_adc_poll_continuous(p_adc_hd);  
  
    /*对采样的数据处理，这里只是简单的打印出采样值*/  
  
    HAL_TEST_DBG("adc0 = %x adc1 =%x\n", adc->data[0].chn[0],  
    adc->data[0].chn[1]);  
  
    /*间隔 1s*/  
  
    wl_os_mdelay(1000);  
  
}
```

2.7.7 hal_status_e s907x_hal_adc_interrupt_oneshotadc_hdl_t *adc, hal_int_cb cb, void *arg);

功能: 中断 adc 采样一次

注意: 调用此函数前需要完成 adc 的初始化

参数:

类型	描述
adc_hdl_t	adc: 指向 adc 实例句柄指针
hal_int_cb	cb: adc 采样完成中断回调函数
void	arg: 指向将要传递到 cb 回调函数中的参数的指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 用中断方式 adc 采样一组数据



```
/*定义 adc 实例句柄*/  
adc_hdl_t adc_hdl;  
  
/*声明一个 adc_hdl_t 类型指针*/  
adc_hdl_t *p_adc_hd;  
  
/*指向 adc 实例句柄*/  
p_adc_hd = &adc_hdl;  
  
/*使能 oneshot 功能*/  
p_adc_hd->config.oneshot.enable = TRUE;  
  
/*表示调用 hal_adc_interrupt_oneshot 后等待 2s 采样*/  
p_adc_hd->config.oneshot.delay = 200;  
  
/*adc0, adc1 各采样 8 个样本数据*/  
p_adc_hd->config.oneshot.read_nums = ADC_FIFO;  
  
/*将配置信息写入寄存器*/  
s907x_hal_adc_init(p_adc_hd);  
.  
.  
.  
  
/*adc 采样开启, 用户可以在中断服务函数 adc_oneshot_poll_isr 中, 对数据进行  
处理, 并失能 adc 采样*/  
s907x_hal_adc_interrupt_oneshot(p_adc_hd, adc_oneshot_poll_isr, p_adc_hd);
```

2.7.8 hal_status_e s907x_hal_adc_interrupt_continious(adc_hdl_t *adc, hal_int_cb cb, void *arg);

功能: adc 采样一次, 采样完成后进中断回调函数处理数据

注意: 调用此函数前需要完成 adc 初始化

参数:



类型	描述
adc_hdl_t	adc: 指向 adc 实例句柄指针
hal_int_cb	cb: adc 采样完成中断回调函数
void	arg: 指向将要传递到 cb 回调函数中的参数的指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: adc0,adc1 各采样一次

```
/*定义 adc 实例句柄*/
adc_hdl_t adc_hdl;
/*声明一个 adc_hdl_t 类型指针*/
adc_hdl_t *p_adc_hd;
/*指向 adc 实例句柄*/
p_adc_hd = &adc_hdl;
/*失能 oneshot 功能*/
p_adc_hd->config.oneshot.enable = FALSE;
/*将配置信息写入寄存器*/
s907x_hal_adc_init(p_adc_hd);
/*开启采样, 采样完成进中断回调函数 adc_read_isr 中,可以对 adc0, 和 adc1 采样
到的数据进行处理*/
s907x_hal_adc_interrupt_continuous(p_adc_hd, adc_read_isr, p_adc_hd);
```

2.7.9 hal_status_e s907x_hal_adc_get_mv(adc_hdl_t *adc);

功能: adc 采样电压值, 单位 mv

注意: 调用此函数前需要完成 adc 初始化



参数:

类型	描述
adc_hdl_t	adc: 指向 adc 实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: adc0, adc1 采样电压值

```
/*定义 adc 实例句柄*/
adc_hdl_t adc_hdl;
/*声明一个 adc_hdl_t 类型指针*/
adc_hdl_t *p_adc_hd;
/*指向 adc 实例句柄*/
p_adc_hd = &adc_hdl;
/*失能 oneshot 功能*/
p_adc_hd->config.oneshot.enable = FALSE;
/*将配置信息写入寄存器*/
s907x_hal_adc_init(p_adc_hd);
.
.
.
While(1){
    /*采样*/
    s907x_hal_adc_get_mv(p_adc_hd);
    /*用户对采样的数据处理 begin*/
    .....
    /*用户对采样的数据处理 end*/
    /*每 1s 采样一次*/
    wl_os_mdelay(1000);
```



{}

2.8 TIMER

◆ `u32 s907x_hal_timer_get_counter(timer_hdl_t *tim)`

获取定时器的当前计数值

◆ `hal_status_e s907x_hal_timer_base_init(timer_hdl_t *tim)`

调用此函数完成定时器基本配置初始化

◆ `hal_status_e s907x_hal_timer_base_deinit(timer_hdl_t *tim)`

将指定定时器的配置信息恢复为缺省状态

◆ `hal_status_e s907x_hal_timer_start_base(timer_hdl_t *tim)`

开启指定定时器

◆ `hal_status_e s907x_hal_timer_stop(timer_hdl_t *tim)`

关闭指定定时器

◆ `hal_status_e s907x_hal_timer_set_period(timer_hdl_t *tim, u32 period)`

对指定定时器设置预装值

◆ `hal_status_e s907x_hal_timer_pwm_init(timer_hdl_t *tim)`

对定时器 PWM 功能初始化

◆ `hal_status_e s907x_hal_timer_pwm_deinit(timer_hdl_t *tim)`

将 pwm 配置信息恢复为缺省状态

◆ `hal_status_e s907x_hal_timer_start_pwm(timer_hdl_t *tim)`

开启 PWM

◆ `hal_status_e s907x_hal_timer_stop_pwm(timer_hdl_t *tim)`

关闭 PWM

◆ `hal_status_e s907x_hal_timer_start_pwm_dma(timer_hdl_t *tim)`

开启 pwm 的 dma 功能

◆ `hal_status_e s907x_hal_timer_stop_pwm_dma(timer_hdl_t *tim)`



关闭 *pwm* 的 *dma* 功能

◆ *hal_status_e s907x_hal_timer_pwm_set_ccr(timer_hdl_t *tim, u32 ccr, u8 channel)*

设定 *pwm* 的 *pulse* 值和输出 *channel*

◆ *hal_status_e s907x_hal_timer_pwm_deinit(timer_hdl_t *tim)*

pwm 功能恢复为缺省状态

◆ *hal_status_e s907x_hal_timer_capture_init(timer_hdl_t *tim)*

脉宽捕获初始化

◆ *hal_status_e s907x_hal_timer_start_capture(timer_hdl_t *tim)*

开启脉宽捕获

◆ *hal_status_e s907x_hal_timer_start_capture_dma(timer_hdl_t *tim)*

开启脉宽捕获 *dma* 接收功能

◆ *hal_status_e s907x_hal_timer_stop_capture_dma(timer_hdl_t *tim)*

关闭脉宽捕获 *dma* 接收功能

2.8.1 u32 s907x_hal_timer_get_counter(timer_hdl_t *tim);

功能: 获取定时器的当前计数值

注意: 调用此函数前需要完成 *tim* 初始化

参数:

类型	描述
<i>timer_hdl_t</i>	<i>tim</i> : 指向 <i>timer</i> 的实例句柄指针

返回:

类型	描述
u32	返回值为定时器的即时的计数值

示例: 无

2.8.2 hal_status_e s907x_hal_timer_base_init(timer_hdl_t *tim);

功能: 调用此函数完成定时器基本配置初始化

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 初始化用户定时器 TIM1(用户可用通用定时器有 TIM1,TIM2,TIM3)

```
/*定义 TIM1 实例句柄*/
timer_hdl_t tim1_hdl;

/*声明 timer_hdl_t 类型指针*/
timer_hdl_t *p_tim1_hd;

/*指向 TIM1 实例句柄*/
p_tim1_hd = &tim1_hdl;

/*idx 选择 TIM1*/
p_tim1_hd->config.idx = TIM1;

/*不分频*/
p_tim1_hd->config.prescaler = 0x0;
/*预装值 32215*/
p_tim1_hd->config.period = 32215; //1hz
/*中断使能*/
p_tim1_hd->config.int_enable = 1;
/*中断回调函数 timer_basic_interrupt 注册*/
p_tim1_hd->it.basic_user_cb.func = timer_basic_interrupt;
/*中断回调函数 timer_basic_interrupt 的参数*/
```



```
p_tim1_hd->it.basic_user_cb.context = p_tim1_hd;  
/*将配置信息写入寄存器*/  
s907x_hal_timer_base_init(p_tim1_hd);
```

2.8.3 hal_status_e s907x_hal_timer_base_deinit(timer_hdl_t *tim);

功能: 将指定定时器的配置信息恢复为缺省状态

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 将定时器 TIM1 恢复为缺省状态

假设 TIM1 的实例句柄为 tim1_hdl

```
/*声明 timer_hdl_t 类型指针*/  
timer_hdl_t *p_tim1_hd;  
/*指向 TIM1 实例句柄*/  
p_tim1_hd = &tim1_hdl;  
/*将 TIM1 恢复为缺省状态*/  
s907x_hal_timer_base_deinit(p_tim1_hd);
```

2.8.4 hal_status_e s907x_hal_timer_start_base(timer_hdl_t *tim);

功能: 开启指定定时器

注意: 调用此函数开启指定定时器前, 需要完成对该定时器的初始化

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 开启定时器 TIM1

假设 TIM1 已经完成初始化, 实例句柄为 tim1_hdl

```
/*声明 timer_hdl_t 类型指针*/\n\ntimer_hdl_t *p_tim1_hd;\n\n/*指向 TIM1 实例句柄*/\n\np_tim1_hd = &tim1_hdl;\n\n/*开启 TIM1*/\n\ns907x_hal_timer_start_base(p_tim1_hd);
```

2.8.5 hal_status_e s907x_hal_timer_stop(timer_hdl_t *tim);

功能: 关闭指定定时器

注意: 调用此函数前, 所操作定时器已经完成初始化, 且为开启状态

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败



示例: 关闭定时器 TIM1

假设 TIM1 已经完成初始化, 实例句柄为 tim1_hdl, 且为开启状态

```
/*声明 timer_hdl_t 类型指针*/  
timer_hdl_t *p_tim1_hd;  
  
/*指向 TIM1 实例句柄*/  
p_tim1_hd = &tim1_hdl;  
  
/*关闭 TIM1*/  
s907x_hal_timer_stop(p_tim1_hd);
```

2.8.6 hal_status_e s907x_hal_timer_set_period(timer_hdl_t *tim, u32 period);

功能: 对指定定时器设置预装值

注意: 调用此函数前, 所操作定时器已经完成初始化

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针
u32	period: 要设定预装值

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 设定 TIM1 预装值为 10000

假设 TIM1 已经完成初始化, 实例句柄为 tim1_hdl

```
/*声明 timer_hdl_t 类型指针*/  
timer_hdl_t *p_tim1_hd;  
  
/*指向 TIM1 实例句柄*/  
p_tim1_hd = &tim1_hdl;
```

```
/*设定 TIM1 的预装值为 10000*/  
s907x_hal_timer_set_period(p_tim1_hd, 10000);
```

2.8.7 hal_status_e s907x_hal_timer_pwm_init(timer_hdl_t *tim);

功能: 对定时器 PWM 功能初始化(PWM 唯一定时器 TIM_PWM)

注意: 调用此函数前需要完成定时器的初始化

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: PWM 功能初始化

```
/*定义定时器实例句柄*/  
timer_hdl_t tim_pwm_hdl;  
  
/*声明 timer_hdl_t 类型指针*/  
  
timer_hdl_t *p_tim_pwm_hd;  
  
/*指向定时器实例句柄*/  
p_tim_pwm_hd = &tim_pwm_hdl;  
/*定时器初始化**/  
/*pwm 定时器只能选用 TIM_PWM*/  
p_tim_pwm_hd->config.idx = TIM_PWM;  
/*预分频设置*/  
p_tim_pwm_hd->config.prescaler = 199;
```

```
/*预装值设置*/  
p_tim_pwm_hd->config.period = 99;  
  
/*PWM 不需要定时器中断回调函数*/  
p_tim_pwm_hd->it.basic_user_cb.func = NULL;  
p_tim_pwm_hd->it.basic_user_cb.context = p_tim_pwm_hd;  
  
/*定时器初始化*/  
s907x_hal_timer_base_init(p_tim_pwm_hd);  
  
/*PWM 的 channel 选择 0(支持 channel0 - channel7 共 8 通道)*/  
p_tim_pwm_hd->pwm.channel = 0;  
  
/*PWM 输出极性高*/  
p_tim_pwm_hd->pwm.polarity = PWM_POLARITY_HIGH;  
  
/*PWM 脉宽设定为 50*/  
p_tim_pwm_hd->pwm.pulse = 50;  
  
/*PWM 初始化*/  
s907x_hal_timer_pwm_init(p_tim_pwm_hd);
```

2.8.8 hal_status_e s907x_hal_timer_pwm_deinit(timer_hdl_t *tim);

功能: 将 pwm 配置信息恢复为缺省状态

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 将 PWM 恢复为缺省状态



假设 PWM 已经初始化，且触发 PWM 的定时器的实例句柄为 tim_pwm_hdl

```
/*声明 timer_hdl_t 类型指针*/  
timer_hdl_t *p_tim_pwm_hd;  
/*指向定时器实例句柄*/  
p_tim_pwm_hd = &tim_pwm_hdl;  
/*PWM 恢复为缺省状态*/  
s907x_hal_timer_pwm_deinit(p_tim_pwm_hd);
```

2.8.9 hal_status_e s907x_hal_timer_start_pwm(timer_hdl_t *tim);

功能: 开启 PWM

注意: 调用此函数前需要完成定时器初始化, pwm 初始化

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 开启 PWM

假设已经完成定时器初始化, PWM 初始化, 且定时器实例句柄为 tim_pwm_hdl

```
/*声明 timer_hdl_t 类型指针*/  
timer_hdl_t *p_tim_pwm_hd;  
/*指向定时器实例句柄*/  
p_tim_pwm_hd = &tim_pwm_hdl;  
/*开启 PWM*/  
s907x_hal_timer_start_pwm(p_tim_pwm_hd);
```



2.8.10 hal_status_e s907x_hal_timer_stop_pwm(timer_hdl_t *tim);

功能: 关闭 PWM

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 关闭 PWM

假设已经完成定时器初始化, PWM 初始化并开启, 定时器实例句柄为 tim_pwm_hdl

```
/* 声明 timer_hdl_t 类型指针 */
timer_hdl_t *p_tim_pwm_hd;
/* 指向定时器实例句柄 */
p_tim_pwm_hd = &tim_pwm_hdl;
/* 关闭 PWM */
s907x_hal_timer_stop_pwm(p_tim_pwm_hd);
```

2.8.11 hal_status_e s907x_hal_timer_start_pwm_dma(timer_hdl_t *tim);

功能: 开启 pwm 的 dma 功能

注意: 调用此函数前必须完成定时器初始化, pwm 初始化

参数:

类型	描述



timer_hdl_t	tim: 指向 timer 的实例句柄指针
-------------	-----------------------

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 开启 dma 输出 pwm

假设已经完成定时器初始化, PWM 初始化, 且定时器实例句柄为 tim_pwm_hdl

```
/*声明 timer_hdl_t 类型指针*/\n\ntimer_hdl_t *p_tim_pwm_hd;\n\n/*指向定时器实例句柄*/\n\np_tim_pwm_hd = &tim_pwm_hdl;\n\n/*dma 配置*/\n\np_tim_pwm_hd->dma.txbuf = (u8*)pwm_dma_buffer;\n\np_tim_pwm_hd->dma.txlen = sizeof(pwm_dma_buffer);\n\n/*根据 dma 配置输出 pwm*/\n\ns907x_hal_timer_start_pwm_dma(p_tim_pwm_hd);
```

2.8.12 hal_status_e s907x_hal_timer_stop_pwm_dma(timer_hdl_t *tim);

功能: 关闭 pwm 的 dma 功能

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 关闭 dma 输出 pwm

假设已经完成定时器初始化， PWM 初始化， 且定时器实例句柄为 tim_pwm_hdl
, pwm 的 dma 功能当前状态为开启

```
/*声明 timer_hdl_t 类型指针*/  
timer_hdl_t *p_tim_pwm_hd;  
/*指向定时器实例句柄*/  
p_tim_pwm_hd = &tim_pwm_hdl;  
/*关闭 pwm 的 dma 功能*/  
s907x_hal_timer_stop_pwm_dma(p_tim_pwm_hd);
```

2.8.13 hal_status_e s907x_hal_timer_pwm_set_ccr(timer_hdl_t *tim, u32 ccr, u8 channel);

功能： 设定 pwm 的 pulse 值和输出 channel

参数：

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针
u32	ccr: 脉宽设定值
u8	channel: pwm 输出通道选择(0-7 代表 8 个 channel)

返回：

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例： 设定 pwm 输出为 channel2, 脉宽为 30

假设已经完成定时器初始化， PWM 初始化， 且定时器实例句柄为 tim_pwm_hdl

```
/*声明 timer_hdl_t 类型指针*/  
timer_hdl_t *p_tim_pwm_hd;  
/*指向定时器实例句柄*/
```



```
p_tim_pwm_hd = &tim_pwm_hdl;  
/*设定 pwm 输出为 channel2, 脉宽为 30*/  
s907x_hal_timer_pwm_set_ccr(p_tim_pwm_hd, 30, 2);
```

2.8.14 hal_status_e s907x_hal_timer_pwm_deinit(timer_hdl_t *tim);

功能: pwm 功能恢复为缺省状态

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 将 pwm 恢复为缺省值状态

假设已经完成定时器初始化, PWM 初始化, 且定时器实例句柄为 tim_pwm_hdl

```
/*声明 timer_hdl_t 类型指针*/  
timer_hdl_t *p_tim_pwm_hd;  
/*指向定时器实例句柄*/  
p_tim_pwm_hd = &tim_pwm_hdl;  
/*将 pwm 恢复为缺省值状态*/  
s907x_hal_timer_pwm_deinit(p_tim_pwm_hd);
```

2.1.15 hal_status_e s907x_hal_timer_capture_init(timer_hdl_t *tim);

功能: 脉宽捕获初始化(专用定时器为 TIM_CAP)

参数:



类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 脉宽捕获初始化

```
/*定义定时器实例句柄*/
timer_hdl_t tim_cap_hdl;

/*声明 timer_hdl_t 类型指针*/
timer_hdl_t *p_tim_cap_hd;

/*指向定时器实例句柄*/
p_tim_cap_hd = &tim_cap_hdl;

/*脉宽捕获使用专用定时器 TIM_CAP*/
p_tim_cap_hd->config.idx = TIM_CAP;

/*预分频设置*/
p_tim_cap_hd->config.prescaler = 2;

/*预装值设置*/
p_tim_cap_hd->config.period = 0xFFFF;

/*定时器中断回调不需要注册*/
p_tim_cap_hd->it.basic_user_cb.func = NULL;
p_tim_cap_hd->it.basic_user_cb.context = p_tim_cap_hd;

/*定时器初始化信息写入寄存器*/
s907x_hal_timer_base_init(p_tim_cap_hd);

/*捕获模式为脉宽捕获*/
p_tim_cap_hd->capture.mode = CAPTURE_PULSE;

/*捕获通道为 channel0*/
p_tim_cap_hd->capture.channel = CAPTURE_CHANNEL_0;
```

```
/*脉宽捕获初始化*/  
s907x_hal_timer_capture_init(p_tim_cap_hd);
```

2.8.16 hal_status_e s907x_hal_timer_start_capture(timer_hdl_t *tim);

功能: 开启脉宽捕获

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 无

2.8.17 hal_status_e s907x_hal_timer_start_capture_dma(timer_hdl_t *tim);

功能: 开启脉宽捕获 dma 接收功能

注意: 调用此函数前需要完成定时器脉宽捕获功能的初始化

参数:

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 开启脉宽捕获 dma 接收功能

假设已经完成定时器脉宽捕获功能的初始化，且定时器实例句柄为 tim_cap_hdl

```
/*声明 timer_hdl_t 类型指针*/  
timer_hdl_t *p_tim_cap_hd;  
/*指向定时器实例句柄*/  
p_tim_cap_hd = &tim_cap_hdl;  
/*注册 dma 接收完成回调函数 timer_capture_dma_cb  
用户可以在回调函数中对收到的数据处理*/  
p_tim_cap_hd->dma.rx_complete.func = timer_capture_dma_cb;  
/*回调函数的参数*/  
p_tim_cap_hd->dma.rx_complete.context = p_tim_cap_hd;  
/*注册 dma 接收缓冲区*/  
p_tim_cap_hd->dma.rxbuf = (u8*)capture_dma_buffer;  
/*定义 dma 接收数据长度*/  
p_tim_cap_hd->dma.rxlen = sizeof(capture_dma_buffer);  
/*开启脉宽捕获 dma 接收*/  
s907x_hal_timer_start_capture_dma(p_tim_cap_hd);
```

2.8.18 hal_status_e s907x_hal_timer_stop_capture_dma(timer_hdl_t *tim);

功能： 关闭脉宽捕获 dma 接收功能

参数：

类型	描述
timer_hdl_t	tim: 指向 timer 的实例句柄指针

返回：

类型	描述
hal_status_e	HAL_OK: 操作成功，其他值均为失败

示例： 关闭脉宽捕获 dma 接收功能



假设已经完成定时器脉宽捕获功能的初始化，且定时器实例句柄为 `tim_cap_hdl`，且处于脉宽捕获 `dma` 接收状态

```
/*声明 timer_hdl_t 类型指针*/  
timer_hdl_t *p_tim_cap_hd;  
/*指向定时器实例句柄*/  
p_tim_cap_hd = &tim_cap_hdl;  
/*关闭脉宽捕获 dma 接收*/  
s907x_hal_timer_stop_capture_dma(p_tim_cap_hd);
```

2.9 RTC

◆ `hal_status_e s907x_hal_rtc_init(rtc_hdl_t *rtc)`

`rtc` 初始化

◆ `hal_status_e s907x_hal_rtc_deinit(rtc_hdl_t *rtc)`

`rtc` 恢复为缺省值状态

◆ `void s907x_hal_rtc_get_time(rtc_hdl_t *rtc)`

获取 `rtc` 当前时间

◆ `void s907x_hal_rtc_get_alarm(rtc_hdl_t *rtc)`

获取 `alarm` 设定时间

◆ `void s907x_hal_rtc_set_unixtime(rtc_hdl_t *rtc)`

通过 `unixtime` 设置本地 `rtc` 时间

◆ `void s907x_hal_rtc_set_basictime(rtc_hdl_t *rtc)`

设置本地 `rtc` 时间

◆ `void s907x_hal_rtc_alarm_init(rtc_hdl_t *rtc)`

`rtc` 的 `alarm` 初始化

◆ `void s907x_hal_rtc_alarm_deinit(rtc_hdl_t *rtc)`



将 rtc 的 alarm 恢复为缺省状态

◆ `void s907x_hal_rtc_alarm_cmd(rtc_hdl_t *rtc, u8 enable)`

rtc 的 alarm 使能/失能控制

2.9.1 hal_status_e s907x_hal_rtc_init(rtc_hdl_t *rtc);

功能: rtc 初始化

参数:

类型	描述
<code>rtc_hdl_t</code>	rtc: 指向 rtc 的实例句柄指针

返回:

类型	描述
<code>hal_status_e</code>	HAL_OK: 操作成功, 其他值均为失败

示例: rtc 初始化

```
/*定义 rtc 实例句柄*/
rtc_hdl_t rtc_hdl;
/*声明 rtc_hdl_t 类型指针*/
rtc_hdl_t *p_rtc_hd;
/*指向 rtc 实例句柄*/
p_rtc_hd = &rtc_hdl;
/*rtc 时钟源选择为内部 32k*/
p_rtc_hd->config.clk_sel = RTC_CLOCKSEL_I32K;
/*时区配置为东 8 区*/
p_rtc_hd->config.zone = 8;
/*将 rtc 配置信息写入寄存器*/
s907x_hal_rtc_init(p_rtc_hd);
```



2.9.2 hal_status_e s907x_hal_rtc_deinit(rtc_hdl_t *rtc);

功能: rtc 恢复为缺省值状态

参数:

类型	描述
rtc_hdl_t	rtc: 指向 rtc 的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 将 rtc 恢复为缺省值状态

假设 rtc 已经被初始化, rtc 实例句柄为 rtc_hdl

```
/*声明 rtc_hdl_t 类型指针*/\n\nrtc_hdl_t *p_rtc_hd;\n\n/*指向 rtc 实例句柄*/\n\np_rtc_hd = &rtc_hdl;\n\n/*将 rtc 恢复为缺省值*/\n\ns907x_hal_rtc_deinit(p_rtc_hd);
```

2.9.3 void s907x_hal_rtc_get_time(rtc_hdl_t *rtc);

功能: 获取 rtc 当前时间

注意: 调用此函数前需要完成 rtc 的初始化, 并已设置过初始时间

参数:

类型	描述



rtc_hdl_t	rtc: 指向 rtc 的实例句柄指针
-----------	---------------------

返回:

类型	描述
void	无

示例: 获取当前 rtc 的时间, 并打印输出

假设 rtc 已经被初始化, 并且已经设置过 rtc 时间, rtc 实例句柄为 rtc_hdl

```
/*声明 rtc_hdl_t 类型指针*/\n\nrtc_hdl_t *p_rtc_hd;\n\n/*指向 rtc 实例句柄*/\n\np_rtc_hd = &rtc_hdl;\n\n/*获取当前 rtc 的时间*/\n\ns907x_hal_rtc_get_time(p_rtc_hd);\n\n/*打印输出当前 rtc 时间*/\n\nprintf("year %d month %d days = %d week = %s hour = %d min %d\nsec %d\\n", p_rtc_hd->real_time.year, p_rtc_hd->real_time.month,\n\np_rtc_hd->real_time.day, week[p_rtc_hd->real_time.week],\n\np_rtc_hd->real_time.hour, p_rtc_hd->real_time.min,\n\np_rtc_hd->real_time.sec);
```

2.9.4 void s907x_hal_rtc_get_alarm(rtc_hdl_t *rtc);

功能: 获取 alarm 设定时间

注意: 调用此函数前需要完成 rtc 的初始化, 并已设置过 alarm 时间

参数:

类型	描述
rtc_hdl_t	rtc: 指向 rtc 的实例句柄指针



返回:

类型	描述
void	无

示例: 无

2.9.5 void s907x_hal_rtc_set_unixtime(rtc_hdl_t *rtc);

功能: 通过 unixtime 设置本地 rtc 时间

注意: wifi 需要连接正常

参数:

类型	描述
rtc_hdl_t	rtc: 指向 rtc 的实例句柄指针

返回:

类型	描述
void	无

示例: 通过 unixtime 设置本地 rtc 时间

```
/* 定义 rtc 实例句柄 */
rtc_hdl_t rtc_hdl;

/* 声明 rtc_hdl_t 类型指针 */
rtc_hdl_t *p_rtc_hd;
/* 指向 rtc 实例句柄 */
p_rtc_hd = &rtc_hdl;
/* 时区配置东 8 区 */
p_rtc_hd->config.zone = 8;
/* 时钟源配置 */
p_rtc_hd->config.clk_sel = RTC_CLOCKSEL_NORMAL;
```

```
/*通过 unixtime 设置本地 rtc 时间*/  
/*hal_rtc_set_unixtime 会在 sntp_setup 中被调用，这里不展开*/  
sntp_setup(p_rtc_hd->config.zone, "ntp1.aliyun.com", "ntp2.aliyun.com",  
           "ntp3.aliyun.com")) {  
  
    /*给出时间让设备获取 ntp 时间并设置到 rtc*/  
    wl_os_mdelay(500);  
  
    /*获取 rtc 当前时间*/  
    s907x_hal_rtc_get_time(p_rtc_hd);  
  
    /*打印输出当前 rtc 时间*/  
    printf("year %d month %d days = %d  week = %s hour = %d min %d  
sec %d\n", p_rtc_hd->real_time.year, p_rtc_hd->real_time.month,  
p_rtc_hd->real_time.day, week[p_rtc_hd->real_time.week],  
p_rtc_hd->real_time.hour, p_rtc_hd->real_time.min,  
p_rtc_hd->real_time.sec);
```

2.9.6 void s907x_hal_rtc_set_basictime(rtc_hdl_t *rtc);

功能: 设置本地 rtc 时间

注意: 调用此函数前需要完成 rtc 初始化

参数:

类型	描述
rtc_hdl_t	rtc: 指向 rtc 的实例句柄指针

返回:

类型	描述
void	无

示例: 设置本地 rtc 初始时间为 2019.4.16 17: 12: 45



假设 rtc 已经被初始化， rtc 实例句柄为 rtc_hdl

```
/*声明 rtc_hdl_t 类型指针*/  
rtc_hdl_t *p_rtc_hd;  
  
/*指向 rtc 实例句柄*/  
  
p_rtc_hd = &rtc_hdl;  
  
/*设置时间*/  
  
p_rtc_hd->config.init_time.year = 2017;  
p_rtc_hd->config.init_time.month = 4;  
p_rtc_hd->config.init_time.day = 16;  
p_rtc_hd->config.init_time.hour = 17;  
p_rtc_hd->config.init_time.min = 12;  
p_rtc_hd->config.init_time.sec = 45;  
  
/*设置 rtc 初始时间为 2019.4.16 17: 12: 45*/  
s907x_hal_rtc_set_basictime(p_rtc_hd);
```

2.9.7 void s907x_hal_rtc_alarm_init(rtc_hdl_t *rtc);

功能: rtc 的 alarm 初始化

注意: 调用此函数前需要完成 rtc 初始化; alarm 模式有 ABSOLUTE_TIME 和
ALARM_BY_PASS_TIME 两种

参数:

类型	描述
rtc_hdl_t	rtc: 指向 rtc 的实例句柄指针

返回:

类型	描述
void	无



示例：这里以 ABSOLUTE_TIME 模式为例初始化 alarm

假设 rtc 已经被初始化， rtc 实例句柄为 rtc_hdl

```
/*表示 alarm 编号，支持 0-11 共 12 个 alarm*/  
u8 alarmID = 1;  
  
/*声明 rtc_hdl_t 类型指针*/  
rtc_hdl_t *p_rtc_hd;  
  
/*指向 rtc 实例句柄*/  
p_rtc_hd = &rtc_hdl;  
  
/*alarm 模式 ABSOLUTE_TIME*/  
p_rtc_hd->alarm.alarm_mode = ALARM_BY_ABSOLUTE_TIME;  
  
/*使能 alarm1*/  
p_rtc_hd->alarm.abs_time[alarmID].enable = TRUE;  
  
/*alarm 设置的时间为 6: 00*/  
p_rtc_hd->alarm.abs_time[id].hour = 6;  
p_rtc_hd->alarm.abs_time[id].min = 00;  
  
/*注册 alarm 触发后的中断回调函数 alarm_timeout_isr*/  
p_rtc_hd->alarm.event.func = alarm_timeout_isr;  
p_rtc_hd->alarm.event.context = p_rtc_hd;  
  
/*配置信息写入寄存器*/  
s907x_hal_rtc_alarm_init(p_rtc_hd);  
  
/*使能 alarm*/  
s907x_hal_rtc_alarm_cmd(rtc, ENABLE);
```

2.9.8 void s907x_hal_rtc_alarm_deinit(rtc_hdl_t *rtc);

功能：将 rtc 的 alarm 恢复为缺省状态

参数：



类型	描述
rtc_hdl_t	rtc: 指向 rtc 的实例句柄指针

返回:

类型	描述
void	无

示例: 无

2.9.9 void s907x_hal_rtc_alarm_cmd(rtc_hdl_t *rtc, u8 enable);

功能: rtc 的 alarm 使能/失能控制

参数:

类型	描述
rtc_hdl_t	rtc: 指向 rtc 的实例句柄指针
u8	enable: ENABLE 使能 / DISABLE 失能

返回:

类型	描述
void	无

示例: 见 2.9.7 示例

2.10 WDG

◆ [hal_status_e hal_wdg_init\(wdg_hdl_t *wdg\)](#)

看门狗初始化

◆ [hal_status_e hal_wdg_deinit\(wdg_hdl_t *wdg\)](#)

将看门狗恢复为缺省状态



◆ `void hal_wdg_start(wdg_hdl_t *wdg)`

开启看门狗

◆ `void hal_wdg_refresh(wdg_hdl_t *wdg)`

喂狗

◆ `void hal_wdg_start_it(wdg_hdl_t *wdg)`

开启看门狗中断

◆ `void hal_wdg_stop(wdg_hdl_t *wdg)`

关闭看门狗

2.10.1 `hal_status_e s907x_hal_wdg_init(wdg_hdl_t *wdg);`

功能: 看门狗初始化

参数:

类型	描述
<code>wdg_hdl_t</code>	wdg: 指向看门狗的实例句柄指针

返回:

类型	描述
<code>hal_status_e</code>	<code>HAL_OK</code> : 操作成功, 其他值均为失败

示例: 初始化看门狗

```
/*定义看门狗实例句柄*/  
wdg_hdl_t wdg_hdl;  
/*声明 wdg_hdl 指针*/  
wdg_hdl *p_wdg_hd;  
/*指向看门狗实例句柄*/
```

```
p_wdg_hd = &wdg_hdl;  
/*在 1s 的时间内及时喂狗，不会引起复位*/  
p_wdg_hd->time_ms = 1000;  
/*初始化看门狗*/  
s907x_hal_wdg_init(p_wdg_hd);
```

2.10.2 hal_status_e s907x_hal_wdg_deinit(wdg_hdl_t *wdg);

功能: 将看门狗恢复为缺省状态

参数:

类型	描述
wdg_hdl_t	wdg: 指向看门狗的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 将看门狗恢复配置恢复为缺省值

假设看门狗已初始化, 实例句柄为 wdg_hdl

```
/*声明 wdg_hdl 指针*/  
wdg_hdl *p_wdg_hd;  
/*指向看门狗实例句柄*/  
p_wdg_hd = &wdg_hdl;  
/*看门狗恢复为缺省状态*/  
s907x_hal_wdg_deinit(p_wdg_hd);
```



2.10.3 void s907x_hal_wdg_start(wdg_hdl_t *wdg);

功能: 开启看门狗

注意: 调用此函数前需要完成看门狗的初始化

参数:

类型	描述
wdg_hdl_t	wdg: 指向看门狗的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 将看门狗开启

假设看门狗已初始化, 实例句柄为 wdg_hdl

```
/*声明 wdg_hdl 指针*/\n\nwdg_hdl *p_wdg_hd;\n\n/*指向看门狗实例句柄*/\n\np_wdg_hd = &wdg_hdl;\n\n/*看门狗开启*/\n\ns907x_hal_wdg_start(p_wdg_hd);
```

2.10.4 void s907x_hal_wdg_refresh(wdg_hdl_t *wdg);

功能: 喂狗

注意: 调用此函数前, 看门狗必须已经初始化, 且开启状态

参数:

类型	描述
wdg_hdl_t	wdg: 指向看门狗的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 喂狗

假设看门狗已初始化, 实例句柄为 wdg_hdl, 看门狗已开启

```
/*声明 wdg_hdl 指针*/\n\nwdg_hdl *p_wdg_hd;\n\n/*指向看门狗实例句柄*/\n\np_wdg_hd = &wdg_hdl;\n\n/*喂狗*/\n\ns907x_hal_wdg_refresh(p_wdg_hd);
```

2.10.5 void s907x_hal_wdg_start_it(wdg_hdl_t *wdg);

功能: 开启看门狗中断

参数:

类型	描述
wdg_hdl_t	wdg: 指向看门狗的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 初始化看门狗并开启中断

```
/*定义看门狗实例句柄*/\n\nwdg_hdl_t wdg_hdl;\n\n/*声明 wdg_hdl 指针*/\n\nwdg_hdl *p_wdg_hd;
```



```
/*指向看门狗实例句柄*/  
p_wdg_hd = &wdg_hdl;  
  
/*未及时喂狗触发中断的时间*/  
p_wdg_hd->time_ms = 1000;  
  
/*注册中断回调函数，用户可以在回调函数保存一些重要的数据*/  
p_wdg_hd->it.func = wdg_isr_cb;  
p_wdg_hd->it.context = p_wdg_hd;  
  
/*初始化看门狗*/  
s907x_hal_wdg_init(p_wdg_hd);  
  
/*开启看门狗中断*/  
s907x_hal_wdg_start_it(p_wdg_hd);
```

2.10.6 void s907x_hal_wdg_stop(wdg_hdl_t *wdg);

功能: 关闭看门狗

参数:

类型	描述
wdg_hdl_t	wdg: 指向看门狗的实例句柄指针

返回:

类型	描述
hal_status_e	HAL_OK: 操作成功, 其他值均为失败

示例: 无



3 附录

3.1 UART

```
u8 tx_burst_size;           /*!<每次 DMA 发送 transfer size */
hal_cb_t rx_complete;      /*!<接收完成回调功能 */
hal_cb_t tx_complete;      /*!<发送完成回调功能 */
u8 *rdbuf;                 /*!<指向接收缓冲区 */
u32 rxlen;                /*!<接收数据的总长度 */
u32 rx_complete_len;       /*!<已经接收到的长度 */
u8 *last_rx_addr;          /*!<指向最新的接收缓冲区指针 */
u8 *txbuf;                /*!<指向发送缓区指针 */
u32 txlen;                /*!<发送数据的总长度 */
u32 tx_complete_len;       /*!<已成功发送数据的长度 */
u32 rx_timeout_cnt;        /*!<串口 DMA 定时接收时的计数器 */

}uart_dma_t;
```

3.2 GPIO

```
typedef struct
{
    uint32_t mode;           /*!<指定 io 口的工作模式 */
    uint32_t pull;           /*!<指定 io 口的上下拉状态 */
}gpio_init_t;

typedef enum
{
    GPIO_PIN_RESET = 0,      /*!<gpio 为除位状态 */
    GPIO_PIN_SET       /*!<gpio 为置位状态 */
}gpio_status_e;
```

3.3 SPI

```
typedef struct spi_hdl_
{
    spi_config_t config;     /*!<spi 配置信息 */
    spi_it_t it;             /*!<spi 的中断控制器 */
    spi_dma_t dma;           /*!<spi 的 dma 控制器 */
    u32 error_code;          /*!<描述的错误信息 */
    u32 status;              /*!<描述的设备状态信息 */

}spi_hdl_t;
```



```
typedef struct spi_config_
{
    u8   idx;                      /*!<描述spi设备编号 */
    u8   spi_master;               /*!<描述spi设备的主/从属性 */
    u16 dir;                      /*!<表示“仅读”“仅写”“读写” */
    u32 mode;                     /*!<表示传输速率 */
    u32 clk_phase;                /*!<同步时钟相位配置 */
    u32 clk_polarity;             /*!<同步时钟极性配置 */
    u32 datalen;                  /*!<表示spi数据位传输模式，支持4-16位 */
}spi_config_t;

typedef struct spi_it_
{
    void *txbuf;                  /*!<指向发送缓冲区指针 */
    u16 txlen;                   /*!<已发送的数据的长度 */
    void *rxbuf;                  /*!<指向接收缓冲区指针 */
    u16 rxlen;                   /*!<已接收的数据的长度 */
    u16 rxreq;                   /*!< */
    hal_cb_t tx_complete;         /*!<发送完成回调功能 */
    hal_cb_t rx_complete;         /*!<接收完成回调功能 */
    hal_cb_t error_cb;            /*!<传输异常回调功能 */
    void *hk;
}spi_it_t;

typedef struct spi_dma_
{
    u8 rx_burst_size;             /*!<dma接收时burst size */
    u8 tx_burst_size;             /*!<dma发送时burst size */
    hal_cb_t rx_complete;          /*!<dma接收完成回调功能 */
    hal_cb_t tx_complete;          /*!<dma发送完成回调功能 */
    u8 *rxbuf;                   /*!<指向接收数据缓冲区指针 */
    u32 rxlen;                   /*!<待接收数据的长度 */
    u32 rx_complete_len;          /*!<已接收数据的长度 */
    u8 *txbuf;                   /*!<指向发送数据缓冲区指针 */
    u32 txlen;                   /*!<待发送数据的长度 */
    u32 tx_complete_len;          /*!<已发送数据的长度 */
}spi_dma_t;
```

idx

```
[  
    SPI_SLAVE_IDX          /*!<spi0*/  
    SPI_MASTER_IDX          /*!<spi1*/  
]  
  
mode  
[  
    SPI_MODE_TX            /*!<仅写*/  
    SPI_MODE_RX            /*!<仅读*/  
    SPI_MODE_TXRX          /*!<读写*/  
]  
  
datalen  
[  
    SPI_DATASIZE_4BIT      /*!< SPI Datasize = 4bits */  
    SPI_DATASIZE_5BIT      /*!< SPI Datasize = 5bits */  
    SPI_DATASIZE_6BIT      /*!< SPI Datasize = 6bits */  
    SPI_DATASIZE_7BIT      /*!< SPI Datasize = 7bits */  
    SPI_DATASIZE_8BIT      /*!< SPI Datasize = 8bits */  
    SPI_DATASIZE_9BIT      /*!< SPI Datasize = 9bits */  
    SPI_DATASIZE_10BIT     /*!< SPI Datasize = 10bits */  
    SPI_DATASIZE_11BIT     /*!< SPI Datasize = 11bits */  
    SPI_DATASIZE_12BIT     /*!< SPI Datasize = 12bits */  
    SPI_DATASIZE_13BIT     /*!< SPI Datasize = 13bits */  
    SPI_DATASIZE_14BIT     /*!< SPI Datasize = 14bits */  
    SPI_DATASIZE_15BIT     /*!< SPI Datasize = 15bits */  
    SPI_DATASIZE_16BIT     /*!< SPI Datasize = 16bits */  
]  
  
clk_phase  
[  
    SPI_PHASE_1EDGE        /*!< SPI Phase 1EDGE */  
    SPI_PHASE_2EDGE        /*!< SPI Phase 2EDGE */  
]  
  
clk_polarity  
[  
    SPI_POLARITY_LOW       /*!< SPI polarity Low */  
    SPI_POLARITY_HIGH      /*!< SPI polarity High */  
]
```



3.4 I2C

```
typedef struct i2c_hdl_
{
    u32 status;                      /*!<描述 i2c 设备状态信息 */
    u32 error_code;                  /*!<描述 i2c 设备错误信息 */
    i2c_config_t config;             /*!<描述 i2c 设备配置信息 */
    i2c_it_t it;                     /*!<i2c 中断控制器 */
    i2c_dma_t dma;                  /*!<i2c dma 控制器 */
}i2c_hdl_t;

typedef struct i2c_config_
{
    u8 idx;                          /*!<描述 i2c 设备编号 */
    u8 i2c_master;                  /*!<表示 i2c 设备主/从属性 */
    u16 dir;                        /*!<表示传输的方向 HAL_DIR_TX / HAL_DIR_RX */
    u16 clock;                      /*!<描述同步时钟的速率 */
    u16 addr_mode;                 /*!<描述 i2c 通信的地址模式 */
    u16 own_addr;                  /*!<描述 i2c 本设备的地址 */
    u16 target_addr;                /*!<描述 i2c 目标设备的地址 */
    u16 general_call;              /*!<置 0 表示不需要 general call */
}i2c_config_t;

typedef struct i2c_it_
{
    u8 *txbuf;                      /*!<指向发送数据缓冲区指针 */
    u16 txlen;                      /*!<发送数据的长度 */
    u8 *rxbuf;                      /*!<指向接收数据缓冲区指针 */
    u16 rxlen;                      /*!<接收数据的长度 */
    u16 rxreq;                      /*!< */
    u32 rst;                         /*!< */
    u32 stop;                        /*!< */
    hal_cb_t gc;                    /*!< */
    hal_cb_t tx_complete;           /*!<发送完成回调功能 */
    hal_cb_t rx_complete;           /*!<接收完成回调功能 */
    hal_cb_t error_cb;              /*!<通信异常回调功能 */
    void *hk;
}i2c_it_t;
```

```
typedef struct i2c_dma_
{
    u8 rx_burst_size;           /*!<dma 接收burst size */
    u8 tx_burst_size;           /*!<dma 发送burst size */
    hal_cb_t rx_complete;       /*!<接收完成回调功能 */
    hal_cb_t tx_complete;       /*!<发送完成回调功能 */
    u8 *rxbuf;                 /*!<指向接收数据缓冲区指针 */
    u32 rxlen;                  /*!<待接收数据的长度 */
    u32 rx_complete_len;        /*!<已接收数据的长度 */
    u8 *txbuf;                 /*!<指向发送数据缓冲区指针 */
    u32 txlen;                  /*!<待发送数据的长度 */
    u32 tx_complete_len;        /*!<已发送数据的长度 */
}i2c_dma_t;
```

idx

```
[  
    I2C_IDX_0           /*!<i2c0 */  
    I2C_IDX_1           /*!<i2c1 */  
]
```

clock

```
[  
    I2C_MASTER_CLK_20K   /*!<同步时钟速率 20khz */  
    I2C_MASTER_CLK_100K  /*!<同步时钟速率 100khz */  
    I2C_MASTER_CLK_400K  /*!<同步时钟速率 400khz */  
    I2C_MASTER_CLK_1000K /*!<同步时钟速率 1000khz */  
]
```

addr_mode

```
[  
    I2C_ADDR_MODE_7BIT   /*!<7 位地址模式 */  
    I2C_ADDR_MODE_10BIT  /*!<10 位地址模式 */  
]
```

3.5 ADC

```
typedef struct adc_hdl_
{
    adc_config_t config;           /*!<adc 配置信息 */
    adc_data_t   data[ADC_FIFO];   /*!<adc 采样fifo */
    hal_cb_t     it;               /*!<中断回调功能 */
}adc_hdl_t;

typedef struct adc_config_
{
    u32 mode;                     /*!<oneshort 功能控制器 */
    adc_oneshot_t oneshot;
}adc_config_t;

typedef struct adc_oneshot_
{
    u8 enable;                   /*!<oneshort 功能使能位 */
    u8 read_nums;                /*!<oneshort 读取一组, adc 采样的次数, 最大 16 */
    u16 rsvd;
    u32 delay;
}adc_oneshot_t;

typedef struct adc_data_
{
    u16 chn[2];                  /*!<表示 adc 的两个内部通道 */
    u16 temperature;             /*!<表示内部温度传感器的采样值 */
    double voltage[2];            /*!<内部电压传感器采集的电压值 */
    u16 rsvd;
}adc_data_t;
```

3.6 TIMER

```
typedef struct timer_hdl_
{
    u32          msp_mode;        /*!<描述 msp mode */
    timer_config_t config;        /*!<描述 time 配置信息 */
    timer_pwm_t   pwm;            /*!<timer 的 pwm 功能控制器 */
    timer_capture_t capture;      /*!<timer 的捕获功能控制器 */
}
```

```
timer_it_t          it;           /*!<time 中斷控制器 */
timer_dma_t         dma;          /*!<time dma 控制器 */
}timer_hdl_t;

typedef struct timer_config_
{
    u8   idx;                  /*!<TIM 的编号 */
    u32 prescaler;            /*!<预分频 */
    u32 period;               /*!<预装值 */
    u32 int_enable;           /*!<中斷使能标志 */
}timer_config_t;

typedef struct timer_pwm_
{
    u8   channel;              /*!<描述 pwm 的channel, 支持0-7 共8 通道 */
    u32 pulse;                /*!<描述 PWM 脉宽 */
    u32 polarity;              /*!<描述 PWM 的输出极性 */
}timer_pwm_t;

typedef struct timer_capture_
{
    u8   channel;              /*!<描述捕获模式不同对应不同的channel */
    u16 mode;                 /*!<表示捕获的模式 脉宽捕获或脉冲数量捕获 */
}timer_capture_t;

typedef struct timer_it_
{
    hal_cb_t basic_user_cb;    /*!<定时器中断回调函数 */
    void     *object;           /*!<用于中断回调函数的输入参数传入 */
}timer_it_t;

typedef struct timer_dma_
{
    hal_cb_t rx_complete;      /*!<dma 接收完成回调函数 */
    hal_cb_t tx_complete;      /*!<dma 发送完成回调函数 */
    u8     *rxbuf;              /*!<指向接收数据缓冲区指针 */
    u32    rxlen;               /*!<要接收数据的长度 */
    u32    rx_complete_len;     /*!<已接收完成数据的长度 */
    u8     *txbuf;              /*!<指向发送数据缓冲区指针 */
    u32    txlen;               /*!<要发送数据的长度 */
    u32    tx_complete_len;     /*!<已发送完成数据的长度 */
}
```



```
void *resource;  
}timer_dma_t;  
  
msp_mode  
[  
    TIM_MSP_BASIC           /*!<定时器作为基本功能时, msp 模式 */  
    TIM_MSP_PWM              /*!<定时器作为PWM 功能时, msp 模式 */  
    TIM_MSP_CAPTURE          /*!<定时器作为捕获功能时, msp 模式 */  
]  
  
idx  
[  
    TIM_PWM                 /*!<pwm timer */  
    TIM_CAP                 /*!<capture time */  
    TIM0                    /*!<system timer */  
    TIM1                    /*!<user timer 1 */  
    TIM2                    /*!<user timer 2 */  
    TIM3                    /*!<user timer 3 */  
]  
  
Polarity  
[  
    PWM_POLARITY_HIGH      /*!<pwm 输出极性高 */  
    PWM_POLARITY_LOW        /*!<pwm 输出极性低 */  
]  
  
channel  
[  
    CAPTURE_CHANNEL_0       /*!<脉宽捕获通道 */  
    CAPTURE_CHANNEL_NUMS    /*!<脉宽数量捕获通道 */  
]  
  
mode  
[  
    CAPTURE_PULSE           /*!<脉宽捕获模式 */  
    CAPTURE_NUMBER          /*!<脉宽数量捕获模式 */  
]
```



3.7 RTC

```
typedef struct rtc_hdl_
{
    rtc_config_t    config;           /*!<描述 rtc 配置信息 */
    alarm_config_t  alarm;            /*!<描述 rtc, alarm 配置信息 */
    system_time_t   real_time;        /*!<表示 rtc 当前的实际时间 */
}rtc_hdl_t;

typedef struct rtc_config_
{
    u8 clk_sel;                     /*!<rtc 驱动时钟选择 */
    int zone;                       /*!<描述时区, 8 代表东8区 */
    system_time_t init_time;        /*!<用于配置 rtc 时间的init time */
}rtc_config_t;

typedef struct alarm_config_
{
    u8  alarm_mode;                /*!<rtc alarm 模式选择, 定时或延时 */
    u8  rsvd;
    struct aram_time
    {
        u8 enable;                  /*!<alarm enable */
        u8 rsvd;                    /*!<指定小时 */
        u8 hour;                    /*!<指定分钟 */
        u8 min;                     /*!<定时 alarm 控制器ALRAM_MAX_NUMS=12*/
    }abs_time[ALRAM_MAX_NUMS];
    u32 pass_time_min;             /*!<延时 alarm 分钟值*/
    hal_cb_t      event;           /*!<alarm 回调事件函数 */
}alarm_config_t;

typedef struct system_time_
{
    u16 year;                      /*!<年 , from 1900 */
    u8 month;                      /*!<月, 1 - 12 */
    u8 day;                        /*!<日, 1 - 31 */
    u8 hour;                        /*!<时, 0 - 23 */
    u8 min;                         /*!<分, 0 - 59 */
    u8 sec;                          /*!<秒, 0 - 59 */
    u8 week;                        /*!<周 */
}
```



```
u32 hw_time;           /*!<unix time */  
}system_time_t;  
  
clk_sel  
[  
    RTC_CLOCKSEL_I32K          /*!<内部32k */  
    RTC_CLOCKSEL_NORMAL         /*!<正常的通过系统时钟分频得到的32k */  
    RTC_CLOCKSEL_EXT32K         /*!<内部32k */  
]  
  
alarm_mode  
[  
    ALARM_BY_ABSOLUTE_TIME    /*!<定时闹钟模式 */  
    ALARM_BY_PASS_TIME         /*!<延时模式 */  
]
```

3.8 WDG

```
typedef struct wdg_hdl_  
{  
    u32 time_ms;  
  
    hal_cb_t it;           /*!<描述规定最迟的喂狗时间, 未在规定时间内  
                           喂狗, 系统会复位 或产生中断 */  
}wdg_hdl_t;               /*!<中断回调函数 */
```

3.9 OTHERS

```
typedef struct hal_cb_  
{  
    hal_int_cb    func;      /*!<callback 回调函数*/  
    void         *context;   /*!<任意类型指针, 用于和callback 的连接*/  
}hal_cb_t;  
  
typedef void (*hal_int_cb)(void *);
```

hal_status_e 返回值为如下几种形态, *HAL_OK* 代表正常返回, 其他值表示异常



[
 HAL_OK (0)
 HAL_ERROR (-1)
 HAL_BUSY (-2)
 HAL_TIMEOUT (-3)
 HAL_NO_MEMORY (-4)
]

SCI CONFIDENTIAL



4 版本信息

日期	版本	更新内容	作者
2019-6-15	1.0	文档草案	Virty
2019-8-6	1.1	API 名称更新, 部分 API 函数 调整	Virty

SCI CONFIDENTIAL



SCI CONFIDENTIAL