BEKEN WiFi-SOC FREERTOS SDK API Reference

Better Life with Wireless
美好生活尽在无线

Version 3.0.1

Version History

Version	Date	Description
3.0.0	2021.07	First Release
3.0.1	2021.08	Add pwm/gpio/flash api
3.0.2	2021.08	Add wifi /TLS/MQTT/TCP-IP API
3.0.3	2021.08	Add PowerSave API

目录

1 UART

1.1 UART简介

UART（Universal Asynchronous Receiver/Transmitter）通用异步收发传输器，UART 作为异步串口通信协议的一种，工作原理是将传输数据的每个字符一位接一位地传输。是在应用程序开发过程中使用频率最高的数据总线。

UART 串口的特点是将数据一位一位地顺序传送，只要2根传输线就可以实现双向通信，一根线发送数据的同时用另一根线接收数据。UART串口通信有几个重要的参数，分别是波特率、起始位、数据位、停止位和奇偶检验位，对于两个使用UART 串口通信的端口，这些参数必须匹配，否则通信将无法正常完成。

1.2 UART Related API

Freertos中的uart接口位于/beken378/func/user_driver目录下，相关api接口如下：

函数	描述
bk_uart_initialize ()	uart初始化
bk_uart_send()	uart发送数据
bk_uart_recv()	uart接收数据

bk_uart_set_rx_callback()	接收回调函数
bk_uart_recv_prefetch()	uart接收预取
bk_uart_get_length_in_buffer()	获取uart中当前buffer长度

1.2.1 uart通用结构体说明

配置uart参数的结构体：

bk_uart_t：

结构体类型	成员
BK_UART_1	uart1
BK_UART_2	uart2

bk_uart_config_t：

结构体类型	成员
data_width	速率
parity	校验位
stop_bits	停止位
flow_control	流控
flags	标志位

uart 相关接口如下：

1.2.2 uart 初始化

```
OSStatus bk_uart_initialize( bk_uart_t uart, const bk_uart_config_t *config, ring_buffer_t *optional_rx_buffer );
```

参数	描述
uart	串口设备号
config	串口设置结构体
optional_rx_buffer	串口操作的buffer
返回	0: 函数执行成功 非0： 执行失败

1.2.3 uart 发送数据

```
OSStatus bk_uart_send( bk_uart_t uart, const void *data, uint32_t size );
```

参数	描述

uart	串口设备号
data	串口发送的数据
size	串口发送数据大小
返回	0：函数执行成功 非0： 执行失败

1.2.4 uart 接收数据

```
OSStatus bk_uart_recv( bk_uart_t uart, const void *data, uint32_t size );
```

参数	**描述**
uart	串口设备号
data	串口接收的数据
size	串口接收数据大小
返回	0：函数执行成功 非0： 执行失败

1.2.5 uart 接收回调函数

```
bk_uart_set_rx_callback(bk_uart_t uart, uart_callback callback, void *param);
```

参数	**描述**
uart	串口设备号
callback	串口接收回调函数
param	接收参数
返回	0：函数执行成功 非0： 执行失败

1.2.6 获取uart 当前缓存值

```
bk_uart_recv_prefetch( bk_uart_t uart, void *data, uint32_t size, uint32_t timeout );
```

参数	**描述**
uart	串口设备号
data	串口接收buffe地址
size	接收buffer大小
timeout	超时时间
返回	0：函数执行成功 非0： 执行失败

1.2.7 获取uart当前buffer长度

```
bk_uart_get_length_in_buffer( bk_uart_t uart );
```

参数	描述
uart	串口设备号
返回	length: 当前buffer长度

1.3 UART示例代码

1.3.1 关键说明

- **UART宏定义**

#define	BAUD_RATE_115200	115200	波特率
#define	DATA_BITS_8	8	数据位
#define	STOP_BITS_1	1	停止位
#define	PARITY_NONE	0	奇偶校验位
#define	BIT_ORDER_LSB	0	高位在前或者低位在前
#define	NRZ_NORMAL	0	模式
#define	RT_SERIAL_RB_BUFSZ	64	接收数据缓冲区大小

设置波特率：

#define	BAUD_RATE_2400	2400
#define	BAUD_RATE_4800	4800
#define	BAUD_RATE_9600	9600
#define	BAUD_RATE_19200	19200
#define	BAUD_RATE_38400	38400
#define	BAUD_RATE_57600	57600
#define	BAUD_RATE_115200	115200
#define	BAUD_RATE_203400	203400
#define	BAUD_RATE_460800	460800
#define	BAUD_RATE_921600	921600
#define	BAUD_RATE_2000000	2000000
#define	BAUD_RATE_3000000	3000000

设置数据位：		
#define	**DATA_BITS_5**	5
#define	**DATA_BITS_6**	6
#define	**DATA_BITS_7**	7
#define	**DATA_BITS_8**	8
#define	**DATA_BITS_9**	9
设置停止位：		
#define	**STOP_BITS_1**	0
#define	**STOP_BITS_2**	1
#define	**STOP_BITS_3**	2
#define	**STOP_BITS_4**	3
设置奇偶校验位：		
#define	**PARITY_NONE**	0
#define	**PARITY_ODD**	1
#define	**PARITY_EVEN**	2
设置高位在前：		
#define	**BIT_ORDER_LSB**	0 高位在前
#define	**BIT_ORDER_MSB**	1 高位在后
模式选择		
#define	**NRZ_NORMAL**	0 normal mode
#define	**NRZ_INVERTED**	1 inverted mode

1.3.2 示例代码

```
struct uart_message
{
        UINT32 send_len;
        UINT32 recv_len;
        UINT8 *send_buf;
        UINT16 *recv_buf;
};
const bk_uart_config_t uart1_config[] =
{
      [0] =
      {
            .baud_rate         = 115200,
            .data_width        = BK_DATA_WIDTH_8BIT,
            .parity    = BK_PARITY_NO,
            .stop_bits = BK_STOP_BITS_1,
```

```
            .flow_control = FLOW_CTRL_DISABLED,
            .flags     = 0,
        },

        [1] =
        {
            .baud_rate        = 19200,
            .data_width       = BK_DATA_WIDTH_8BIT,
            .parity    = BK_PARITY_NO,
            .stop_bits = BK_STOP_BITS_1,
            .flow_control = FLOW_CTRL_DISABLED,
            .flags     = 0,
        },
        [2] =
        {
            .baud_rate        = 115200,
            .data_width       = BK_DATA_WIDTH_8BIT,
            .parity    = BK_PARITY_EVEN,
            .stop_bits = BK_STOP_BITS_1,
            .flow_control = FLOW_CTRL_DISABLED,
            .flags     = 0,
        },
};
ring_buffer_t *ring_buf;
struct uart_message uart_msg;
void uart_test_send(void)
{
        struct uart_message msg;
        int i,ret = 0;
        msg.recv_len = UART_TX_BUFFER_SIZE;
        msg.send_len = UART_TX_BUFFER_SIZE;
        msg.recv_buf =os_malloc(UART_TX_BUFFER_SIZE * sizeof(msg.recv_buf[0]));
        if(msg.recv_buf == 0)
        {
                os_printf("msg.recv_buf malloc failed\r\n");
                return;
        }
        msg.send_buf = os_malloc(UART_TX_BUFFER_SIZE * sizeof(msg.send_buf[0]));
        if(msg.send_buf == 0)
```

```
    {
        os_printf("msg.send_buf malloc failed\r\n");
        return;
    }
    ring_buf->buffer = msg.send_buf;

    for(i=0; i<UART_TX_BUFFER_SIZE; i++)
    {
        msg.send_buf[i] = 0x01 + i;
    }
    ret =   bk_uart_initialize(UART_TEST_POART1, &uart1_config[0], ring_buf);
    if (ret != kNoErr)
    {
        os_printf("init failed\r\n");
    }
    bk_uart_send(UART_TEST_POART1, msg.send_buf, UART_TX_BUFFER_SIZE);
    for(i=0; i<UART_TX_BUFFER_SIZE; i++)
    {
        os_printf("send_buf[%d] =0x%x\r\n",i,msg.send_buf[i]);
    }
}
void uart_test_recv(void)
{
    struct uart_message msg;
    int i,ret = 0;
    msg.recv_len = UART_TX_BUFFER_SIZE;
    msg.send_len = UART_TX_BUFFER_SIZE;
    msg.recv_buf =os_malloc(UART_TX_BUFFER_SIZE * sizeof(msg.recv_buf[0]));
    if(msg.recv_buf == 0)
    {
        os_printf("msg.recv_buf malloc failed\r\n");
        return;
    }
    msg.send_buf = os_malloc(UART_TX_BUFFER_SIZE * sizeof(msg.send_buf[0]));
    if(msg.send_buf == 0)
    {
        os_printf("msg.send_buf malloc failed\r\n");
        return;
    }
```

```
    ring_buf->buffer = msg.send_buf;
    ret =   bk_uart_initialize(UART_TEST_POART1, &uart1_config[0], ring_buf);
    if (ret != kNoErr)
    {
        os_printf("init failed\r\n");
    }
    for(i=0; i<UART_RX_BUFFER_SIZE; i++)
    {
        os_printf("send_buf[%d] =0x%x\r\n",i,msg.recv_buf[i]);
    }
}
```

1.4 注意事项

接收数据缓冲区大小默认64字节。若一次性数据接收字节数很多，没有及时读取数据，那么缓冲区的数据将会被新接收到的数据覆盖，造成数据丢失，建议调大缓冲区。

2 Timer定时器

2.1 定时器简介

BEKEN WiFi Soc具有6路timer定时器，其中timer0/1/2为时钟源为26M，timer3/4/5时钟源为32K，对应的通道为。

通道	描述
0	timer0
1	timer1
2	timer2
3	timer3
4	timer4
5	timer5

2.2 Timer Related API

timer相关接口参考beken378\func\user_driver\BkDriverTimer.h，相关接口如下：

函数	描述
bk_timer_initialize ()	timer初始化
bk_timer_initialize_us()	timer初始化为单位为us的定时器
bk_get_timer_cnt()	获取定时器当前计数值
bk_timer_stop()	停止定时器

2.2.1 timer枚举类型说明

BKTIMER0	timer0
BKTIMER1	timer1
BKTIMER2	timer2
BKTIMER3	timer3
BKTIMER4	timer4
BKTIMER5	timer5

2.2.2 初始化timer

OSStatus bk_timer_initialize(uint8_t timer_id, uint32_t time_ms, void *callback);

参数	描述
timer_id	选择的定时器通道：0 ~ 5
time_ms	定时器设置时间
callback	定时器中断回调函数
返回	0：成功；-1：错误

2.2.3 初始化单位为us的定时器

OSStatus bk_timer_initialize_us(uint8_t timer_id, uint32_t time_us, void *callback);

参数	描述
timer_id	选择的定时器通道：0 ~ 5
time_us	定时器设置时间(单位为us)
callback	定时器中断回调函数
返回	0：成功；-1：错误

2.2.4 获取当前定时器计数值

UINT32 bk_get_timer_cnt(uint8_t timer_id);

参数	描述
timer_id	选择的定时器通道：0 ~ 5
返回	当前选择定时器计数值（其值为寄存器的实际计数值）

2.2.4 停止定时器

OSStatus bk_timer_stop(uint8_t timer_id)

参数	描述
timer_id	选择的定时器通道：0 ~ 5
返回	0：成功；-1：错误

2.3 定时器示例代码

定时器示例代码参考beken378\func\user_driver\BkDriverTimer.c

```
/static void bk_timer_test_isr_cb(UINT8 arg)
{
    bk_printf("%s %d rtos-time: %d mS\r\n",__FUNCTION__,__LINE__,rtos_get_time());
}

void bk_timer_test_start(void)
{
    bk_timer_initialize(BKTIMER5,1000,bk_timer_test_isr_cb);
}

void user_main(void)
{
    bk_printf("%s %s\r\n",__FILE__,__FUNCTION__);
    bk_timer_test_start();
}
```

3 看门狗

3.1 看门狗简介

看门狗时钟用于复位系统，避免系统乱序跑飞。当 MCU 停止运行或者断电时，看门狗也会停止运行。开启看门狗模块后需要设置一个定时喂狗的机制，方便软件调试以及重启问题分析。

3.2 看门狗 Related API

看门狗相关接口参考beken378\func\user_driver\BkDriverWdg.h，相关接口如下：

函数	描述
bk_wdg_initialize ()	初始化看门狗
bk_wdg_reload ()	喂狗
bk_wdg_finalize ()	结束看门狗

3.2.1 看门狗初始化

```
OSStatus bk_wdg_initialize( uint32_t timeout );
```

参数	描述

timerout	看门狗重启时间，单位ms，最大值0xFFFF，即约65s
返回	0：成功；-1：错误

3.2.2 喂狗

void bk_wdg_reload(void)；

参数	**描述**
void	空
返回	无

3.2.3 结束看门狗

OSStatus bk_wdg_finalize(void)；

参数	**描述**
void	空
返回	无

3.3 看门狗示例代码

看门狗使用比较简单，本次版本暂不提供示例代码。

4 通用SPI

4.1 通用SPI简介

BEKEN wifi soc 有硬件spi模块，特性如下：

a) 数据交换长度可配，常以byte为单位，MSB先发，LSB后发；
b) 支持主机模式，时钟可配置，最大速率30MHZ；
c) 支持从机模式，能承受的最大速率10MHZ；
d) 时针极性（CPOL）和时针相位（CPHA）可配置；
e) 支持四线全双工（MOSI、MISO、CSN、CLK）和三线半双工（DATA、CS、CLK）。

4.2 通用SPI Related API

目前freertos中实现都是通过spi+dma当时传输数据，所以相关api接口如下：

函数	描述
bk_spi_dma_init ()	初始化spi模块
bk_spi_dma_transfer ()	spi数据传输（接收/发送）

4.2.1 spi结构体说明

spi_message：	
send_buf	spi 发送数据buffer
send_len	spi 发送数据长度
recv_buf	spi 接收数据buffer
recv_len	spi 接收数据长度

4.2.2 spi模块初始化

在使用spi接口前，需配置spi接口：

```
int bk_spi_dma_init(UINT32 mode, UINT32 rate, struct spi_message *spi_msg);
```

参数	描述
mode	spi工作模式设置（见如下说明）
rate	spi工作频率
spi_msg	spi传输数据构体
返回	RT_EOK(0)：成功； 其他：出错

4.2.3 spi发送/接收数据

```
int bk_spi_dma_transfer(UINT32 mode, struct spi_message *spi_msg);
```

参数	描述
mode	spi工作模式设置（见如下说明）
spi_msg	spi传输数据构体
返回	RT_EOK(0)：成功； 其他：出错

主模式下，发送完所有数据后，立即返回。从模式下，可能会挂起，直到与之通信的SPI主发起spi时序，并且所有数据都发。

4.3 通用SPI示例代码

示例代码参考\beken378\fun\wlan_ui\bk_peripheral_test.c。打开宏定义：CFG_SUPPORT_SPI_TEST，开启通用spi_dma功能测试。

4.3.1 关键说明

• 通用SPI宏定义

#define	CFG_USE_SPI_DMA	开启spi + dma模式
#define	CFG_USE_SPI_MASTER	开启master
#define	CFG_USE_SPI_SLAVE	开启slave

• SPI工作模式:

#define	SPI_CPHA	(1<<0)	sck第二个边沿采样数据
#define	SPI_CPOL	(1<<1)	sck空闲时处于高电平
#define	SPI_LSB	(0<<2)	0-LSB
#define	SPI_MSB	(1<<2)	1-MSB
#define	SPI_MASTER	(0<<3)	master模式
#define	SPI_SLAVE	(1<<3)	slave模式
#define	SPI_MODE_0	(0 | 0)	CPOL = 0, CPHA = 0
#define	SPI_MODE_1	(0 | SPI_CPHA)	CPOL = 0, CPHA = 1
#define	SPI_MODE_2	(SPI_CPOL | 0)	CPOL = 1, CPHA = 0
#define	SPI_MODE_4	(SPI_CPOL | SPI_CPHA)	CPOL = 1, CPHA = 1

4.3.2 示例代码

```
/*
* 程序清单： 这是通用spi dma的使用例程.
* 命令调用格式： gspi_test slave_dma_rx/slave_dma_tx/master_dma_tx/master_dma_rx len rate
* 程序功能： 配置spi接口为主/从，传输速率rate，完成发送/接收
*/
void gspi_test(char *pcWriteBuffer, int xWriteBufferLen, int argc, char **argv)
{
    struct spi_message  msg;
    UINT32 max_hz;
    UINT32 mode;
    /* SPI Interface with Clock Speeds Up to 30 MHz */
    if (argc == 5)
    {
        max_hz = SPI_BAUDRATE ;//atoi(argv[3]);
    } else
    {
        max_hz = SPI_BAUDRATE; //master/slave
    }
    if (os_strcmp(argv[1], "master") == 0)
    {
        mode = SPI_MODE_0 | SPI_MSB | SPI_MASTER;
        //bk_spi_master_init (cfg->max_hz, cfg->mode);
    } else if (os_strcmp(argv[1], "slave") == 0)
    {
        mode = SPI_MODE_0 | SPI_MSB | SPI_SLAVE;
        bk_spi_slave_init(max_hz, mode);
    }
#if CFG_USE_SPI_DMA
    else if (os_strcmp(argv[1], "slave_dma_rx") == 0)
    {
        UINT8 *buf;
        int rx_len, ret;
        if (argc < 2)
            rx_len = SPI_RX_BUF_LEN;
        else
            rx_len = atoi(argv[2]);
        bk_printf("spi dma rx: rx_len:%d\n", rx_len);
        buf = os_malloc(rx_len * sizeof(UINT8));
```

```
        if (!buf) {
            bk_printf("spi test malloc buf fail\r\n");
            return ;
        }
        os_memset(buf, 0, rx_len);
        msg.send_buf = NULL;
        msg.send_len = 0;
        msg.recv_buf = buf;
        msg.recv_len = rx_len;
        mode = SPI_MODE_0 | SPI_MSB | SPI_SLAVE;
        max_hz = atoi(argv[3]);
        bk_spi_dma_init(mode, max_hz, &msg);
        ret = bk_spi_dma_transfer(mode, &msg);
        if (ret)
            bk_printf("spi dma recv error%d\r\n", ret);
        else {
            for (int i = 0; i < rx_len; i++) {
                bk_printf("%02x,", buf[i]);
                if ((i + 1) % 32 == 0)
                    bk_printf("\r\n");
            }
            bk_printf("\r\n");
            os_free(buf);
        }
    } else if ((os_strcmp(argv[1], "slave_dma_tx") == 0))
    {
        UINT8 *buf;
        int tx_len, ret;
        if (argc < 2)
            tx_len = SPI_RX_BUF_LEN;
        else
            tx_len = atoi(argv[2]);
        bk_printf("spi dma tx: tx_len:%d,%d\n", tx_len, max_hz);
        buf = os_malloc(tx_len * sizeof(UINT8));
        if (!buf) {
            bk_printf("spi test malloc buf fail\r\n");
            return ;
        }
        os_memset(buf, 0, tx_len);
```

```
        for (int i = 0; i < tx_len; i++)
            buf[i] = i & 0xFF;
        msg.send_buf = buf;
        msg.send_len = tx_len;
        msg.recv_buf = NULL;
        msg.recv_len = 0;
        mode = SPI_MODE_0 | SPI_MSB | SPI_SLAVE;
        max_hz = atoi(argv[3]);
        bk_spi_dma_init(mode, max_hz, &msg);
        ret = bk_spi_dma_transfer(mode, &msg);
        if (ret)
            bk_printf("spi dma send error%d\r\n", ret);
        else {
            for (int i = 0; i < tx_len; i++) {
                bk_printf("%02x,", buf[i]);
                if ((i + 1) % 32 == 0)
                    bk_printf("\r\n");
            }
            bk_printf("\r\n");
            os_free(buf);
        }
    } else if ((os_strcmp(argv[1], "master_dma_tx") == 0))
    {
        UINT8 *buf;
        int tx_len, ret;
        if (argc < 2)
            tx_len = SPI_RX_BUF_LEN;
        else
            tx_len = atoi(argv[2]);
        max_hz = atoi(argv[3]);//SPI_BAUDRATE;
        bk_printf("spi master    dma tx: tx_len:%d max_hz:%d\r\n", tx_len, max_hz);
        buf = os_malloc(tx_len * sizeof(UINT8));
        if (!buf) {
            bk_printf("spi test malloc buf fail\r\n");
            return ;
        }
        os_memset(buf, 0, tx_len);
        for (int i = 0; i < tx_len; i++)
            buf[i] = i & 0xFF;
```

```
        msg.send_buf = buf;
        msg.send_len = tx_len;
        msg.recv_buf = NULL;
        msg.recv_len = 0;
        mode = SPI_MODE_0 | SPI_MSB | SPI_MASTER;
        bk_spi_dma_init(mode, max_hz, &msg);
        ret = bk_spi_dma_transfer(mode,&msg);
        if (ret)
            bk_printf("spi dma send error%d\r\n", ret);
        else {
            for (int i = 0; i < tx_len; i++) {
                bk_printf("%02x,", buf[i]);
                if ((i + 1) % 32 == 0)
                    bk_printf("\r\n");
            }
            bk_printf("\r\n");
            os_free(buf);
        }
    } else if ((os_strcmp(argv[1], "master_dma_rx") == 0))
    {
        UINT8 *buf;
        int rx_len, ret;
        if (argc < 2)
            rx_len = SPI_RX_BUF_LEN;
        else
            rx_len = atoi(argv[2]) + 1;  //slave tx first send 0x72 so must send one more
        max_hz = atoi(argv[3]);//SPI_BAUDRATE;
        bk_printf("spi master    dma rx: rx_len:%d max_hz:%d\r\n\n", rx_len, max_hz);
        buf = os_malloc(rx_len * sizeof(UINT8));
        if (!buf) {
            bk_printf("spi test malloc buf fail\r\n");
            return ;
        }
        os_memset(buf, 0, rx_len);
        msg.send_buf = NULL;
        msg.send_len = 0;
        msg.recv_buf = buf;
        msg.recv_len = rx_len;
        mode = SPI_MODE_0 | SPI_MSB | SPI_MASTER;
```

```
        bk_spi_dma_init(mode, max_hz, &msg);
        ret = bk_spi_dma_transfer(mode,&msg);
        if (ret)
            bk_printf("spi dma recv error%d\r\n", ret);
        else {
            for (int i = 1; i < rx_len; i++) {
                bk_printf("%02x,", buf[i]);
                if ((i + 1) % 32 == 0)
                    bk_printf("\r\n");
            }
            bk_printf("\r\n");
            os_free(buf);
        }
    } else if ((os_strcmp(argv[1], "master_tx_loop") == 0))
    {
        UINT8 *buf;
        int tx_len, ret;
        UINT32 cnt = 0;
        if (argc < 2)
            tx_len = SPI_RX_BUF_LEN;
        else
            tx_len = atoi(argv[2]);
        max_hz = atoi(argv[3]); //SPI_BAUDRATE;
        bk_printf("spi master    dma tx: tx_len:%d max_hz:%d\r\n", tx_len, max_hz);
        buf = os_malloc(tx_len * sizeof(UINT8));
        if (!buf) {
            bk_printf("buf malloc fail\r\n");
            return;
        }
        os_memset(buf, 0, tx_len);
        for (int i = 0; i < tx_len; i++)
            buf[i] = i + 0x60;
        msg.send_buf = buf;
        msg.send_len = tx_len;
        msg.recv_buf = NULL;
        msg.recv_len = 0;
        mode = SPI_MODE_0 | SPI_MSB | SPI_MASTER;
        bk_spi_dma_init(mode, max_hz, &msg);
        while (1) {
```

```
                if (cnt >= 0x1000)
                    break;
                ret = bk_spi_dma_transfer(mode,&msg);
                if (ret)
                    bk_printf("spi dma send error%d\r\n", ret);
                else
                    bk_printf("%d\r\n", cnt++);
                rtos_delay_milliseconds(80);
            }
        }
#endif
        else
            bk_printf("gspi_test master/slave      tx/rx      rate    len\r\n");
        //CLI_LOGI("cfg:%d, 0x%02x, %d\r\n", cfg->data_width, cfg->mode, cfg->max_hz);
        if (os_strcmp(argv[2], "tx") == 0)
        {
            UINT8 *buf;
            int tx_len;
            if (argc < 4)
                tx_len = SPI_TX_BUF_LEN;
            else
                tx_len = atoi(argv[4]);
            bk_printf("spi init tx_len:%d\n", tx_len);
            buf = os_malloc(tx_len * sizeof(UINT8));
            if (buf) {
                os_memset(buf, 0, tx_len);
                for (int i = 0; i < tx_len; i++)
                    buf[i] = i & 0xff;
                msg.send_buf = buf;
                msg.send_len = tx_len;
                msg.recv_buf = NULL;
                msg.recv_len = 0;
                bk_spi_slave_xfer(&msg);
                for (int i = 0; i < tx_len; i++) {
                    bk_printf("%02x,", buf[i]);
                    if ((i + 1) % 32 == 0)
                        bk_printf("\r\n");
                }
                bk_printf("\r\n");
```

```
                os_free(buf);
            }
        } else if (os_strcmp(argv[2], "rx") == 0)
        {
            UINT8 *buf;
            int rx_len;
            if (argc < 4)
                rx_len = SPI_RX_BUF_LEN;
            else
                rx_len = atoi(argv[4]);
            bk_printf("SPI_RX: rx_len:%d\n", rx_len);
            buf = os_malloc(rx_len * sizeof(UINT8));
            if (buf) {
                os_memset(buf, 0, rx_len);
                msg.send_buf = NULL;
                msg.send_len = 0;
                msg.recv_buf = buf;
                msg.recv_len = rx_len;
                //CLI_LOGI("buf:%d\r\n", buf);
                rx_len = bk_spi_slave_xfer(&msg);
                bk_printf("rx_len:%d\r\n", rx_len);
                for (int i = 0; i < rx_len; i++) {
                    bk_printf("%02x,", buf[i]);
                    if ((i + 1) % 32 == 0)
                        bk_printf("\r\n");
                }
                bk_printf("\r\n");
                os_free(buf);
            }
        } else
        {
            //CLI_LOGI("gspi_test master/slave tx/rx rate len\r\n");
        }
}
```

5 PWM

5.1 PWM简介

BEKEN WiFi Soc具有6路PWM输出，每一路的周期及占空比都可以单独配置，BK7231N channel 如下。

通道	描述
0	对应gpio6引脚
1	对应gpio7引脚
2	对应gpio8引脚
3	对应gpio9引脚
4	对应gpio24引脚
5	对应gpio26引脚

Note: 此表格是BK7231N为例，wifi 系列 soc 中pwm通道对应GPIO需要以对应的wifi soc gpio映射表为准，请注意查看相应芯片手册。

5.2 PWM Related API

7231u/7251 pwm共有的相关接口参考beken378\func\user_driver\BkDriverPwm.h，相关接口如下：

函数	描述
bk_pwm_initialize()	PWM初始化
bk_pwm_start()	启动PWM功能
bk_pwm_stop()	停止PWM功能
bk_pwm_update_param()	更新pwm channel 频率和占空比

7231n pwm api 如下:

函数	描述
bk_pwm_cw_initialize()	初始化一组互斥的pwm channel参数
bk_pwm_cw_update_param()	更新一组互斥pwm channel的占空比和频率
bk_pwm_cw_start()	启动pwm 互斥channel
bk_pwm_cw_stop()	停止pwm 互斥channel
bk_pwm_initlevl_set_low()	设置pwm channel 初始电平为低电平
bk_pwm_initlevl_set_high()	设置pwm channel 初始电平为高电平

5.2.1 pwm枚举类型说明

bk_pwm_t：

BK_PWM_0	pwm0

BK_PWM_1	pwm1
BK_PWM_2	pwm2
BK_PWM_3	pwm3
BK_PWM_4	pwm4
BK_PWM_5	pwm5

5.2.2 初始化pwm

```
OSStatus bk_pwm_initialize(bk_pwm_t pwm, uint32_t cycle, uint32_t duty_cycle);
```

参数	描述
pwm	选择的pwm通道：0 ~ 5
cycle	设置pwm的方波周期
duty_cycle	设置pwm的占空值
返回	0：成功；-1：错误

5.2.3 启动pwm功能

```
OSStatus bk_pwm_start(bk_pwm_t pwm)；
```

参数	描述
pwm	选择的pwm通道：0 ~ 5
返回	0：成功；-1：错误

5.2.4 停止pwm功能

```
OStatus bk_pwm_stop(bk_pwm_t pwm)；
```

参数	描述
pwm	选择的pwm通道：0~5
返回	0：成功；-1：错误

5.2.5 调节pwm参数

```
OSStatus bk_pwm_update_param(bk_pwm_t pwm, uint32_t frequency, uint32_t duty_cycle)；
```

参数	描述
pwm	pwm通道
frequency	pwm频率

duty_cycle	pwm的占空比
返回	0：成功；-1：错误

bk7231n pwm api 说明

5.2.6 初始化2个互斥pwm通道参数

void bk_pwm_cw_initialize(bk_pwm_t pwm1, bk_pwm_t pwm2,uint32_t frequency, uint32_t duty_cycle1,uint32_t duty_cycle2,uint32_t dead_band);

参数	描述
pwm1	pwm互斥通道1
pwm2	pwm互斥通道2
frequency	2个互斥pwm频率
duty_cycle1	pwm通道1的占空比
duty_cycle2	pwm通道2的占空比
dead_band	2个pwm互斥死区时间
返回	空

5.2.7 调节互斥pwm参数

OSStatus bk_pwm_cw_update_param (bk_pwm_t pwm1, bk_pwm_t pwm2, uint32_t frequency, uint32_t duty_cycle1, uint32_t duty_cycle2, uint32_t dead_band);

参数	描述
pwm	pwm通道
frequency	2个互斥pwm channel频率
duty_cycle1	互斥pwm的占空比
duty_cycle2	pwm第2次翻转时间,默认一般为0
duty_cycle3	pwm第3次翻转时间,默认一般为0
返回	0：成功；-1：错误

5.2.8 启动pwm互斥功能

void bk_pwm_cw_start(bk_pwm_t pwm1, bk_pwm_t pwm2);

参数	描述
pwm1	选择的pwm通道：0~5
pwm2	选择的pwm通道：0~5

返回	空

5.2.8 停止pwm互斥功能

```
void bk_pwm_cw_stop(bk_pwm_t pwm1, bk_pwm_t pwm2);
```

参数	描述
pwm1	选择的pwm通道：0~5
pwm2	选择的pwm通道：0~5
返回	空

5.2.9 设置pwm初始电平为低电平

```
OSStatus bk_pwm_initlevl_set_low(bk_pwm_t pwm);
```

参数	描述
pwm	选择的pwm通道：0~5
返回	0：成功；-1：错误

5.2.10 设置pwm初始电平为高电平

```
OSStatus bk_pwm_initlevl_set_high(bk_pwm_t pwm);
```

参数	描述
pwm	选择的pwm通道：0~5
返回	0：成功；-1：错误

5.3 PWM示例代码

freertos中示例代码参考bk_peripheral_test.c中pwm_Command命令，串口输入命令 ：pwm single/update/cw channel1 duty_cycle1 cycle （channel2 duty_cycle1 dead_band）,即可在对应的pin脚上检测到波形。

```
/*
* 程序清单： 这是一个简单PWM使用例程
* 命令调用格式： pwm_test single 1 8000 16000
* 程序功能： 输入命令可以检测到对应的PWM通道上输出PWM波形。
*/
```

```
static void pwm_Command(char *pcWriteBuffer, int xWriteBufferLen, int argc, char **argv)
{
	UINT8 channel1;
	UINT32 duty_cycle1, cycle, cap_value;
#if (CFG_SOC_NAME == SOC_BK7231N) || (CFG_SOC_NAME == SOC_BK7236)
	UINT8 channel2;
	UINT32 duty_cycle2;
	UINT32 dead_band;
#endif

	/*get the parameters from command line*/
	channel1  = atoi(argv[2]);
	duty_cycle1	= atoi(argv[3]);
	cycle		= atoi(argv[4]);
#if (CFG_SOC_NAME == SOC_BK7231N) || (CFG_SOC_NAME == SOC_BK7236)
	channel2  = atoi(argv[5]);
	duty_cycle2	= atoi(argv[6]);
	dead_band	= atoi(argv[7]);
#endif

	if (cycle < duty_cycle1)
	{
		PERI_LOGW(TAG, "pwm param error: end < duty\r\n");
		return;
	}

	if (os_strcmp(argv[1], "single") == 0)
	{
		if (5 != argc) {
			PERI_LOGW(TAG, "pwm single test usage: pwm
[single][channel][duty_cycle][freq]\r\n");
			return;
		}
		PERI_LOGI(TAG, "pwm channel %d: duty_cycle: %d    freq:%d \r\n", channel1,
duty_cycle1, cycle);

		bk_pwm_initialize(channel1, cycle, duty_cycle1,0,0);
		bk_pwm_start(channel1);			/*start single pwm channel once */
	} else if (os_strcmp(argv[1], "stop") == 0)
```

```
            bk_pwm_stop(channel1);
#if ((CFG_SOC_NAME == SOC_BK7231N) || (CFG_SOC_NAME == SOC_BK7236)
||(CFG_SOC_NAME == SOC_BK7271))
        else if (os_strcmp(argv[1], "update") == 0)
        {
            if (5 != argc) {
                PERI_LOGW(TAG, "pwm update usage: pwm
[update][channel1][duty_cycle][freq]\r\n");
                return;
            }

            PERI_LOGI(TAG, "pwm %d update: %d\r\n", duty_cycle1);
            bk_pwm_update_param(channel1, cycle, duty_cycle1);          /*updata pwm freq and
duty_cycle */
        } else if (os_strcmp(argv[1], "cap") == 0)
        {
            uint8_t cap_mode = duty_cycle1;

            if (5 != argc) {
                PERI_LOGW(TAG, "pwm cap usage: pwm [cap][channel1][mode][freq]\r\n");
                return;
            }

            bk_pwm_capture_initialize(channel1, cap_mode);              /*capture pwm value */
            bk_pwm_start(channel1);
        } else if (os_strcmp(argv[1], "capvalue") == 0)
        {
            cap_value = bk_pwm_get_capvalue(channel1);
            PERI_LOGI(TAG, "pwm : %d cap_value=%x \r\n", channel1, cap_value);
        }
        #if ((CFG_SOC_NAME == SOC_BK7231N) || (CFG_SOC_NAME == SOC_BK7236))
        else if (os_strcmp(argv[1], "cw") == 0)
        {

            if (8 != argc) {
                PERI_LOGW(TAG, "pwm cw test usage: pwm
[cw][channel1][duty_cycle1][freq][channel2][duty_cycle2][dead_band]\r\n");
                return;
            }
```

```
            PERI_LOGI(TAG, "pwm : %d / %d cw pwm test \r\n", channel1, channel2);

            bk_pwm_cw_initialize(channel1, channel2, cycle, duty_cycle1, duty_cycle2, dead_band);
            bk_pwm_cw_start(channel1, channel2);
        } else if (os_strcmp(argv[1], "updatecw") == 0)
        {
            if (8 != argc) {
                PERI_LOGW(TAG, "pwm cw test usage: pwm
[cw][channel1][duty_cycle1][freq][channel2][duty_cycle2][dead_band]\r\n");
                return;
            }

            PERI_LOGI(TAG, "pwm : %d / %d cw updatw pwm test \r\n", channel1, channel2);

            bk_pwm_cw_update_param(channel1, channel2, cycle, duty_cycle1, duty_cycle2,
dead_band);
        } else if (os_strcmp(argv[1],    "loop") == 0)
        {
            uint16_t cnt = 1000;

            PERI_LOGI(TAG, "pwm : %d / %d pwm update loop test \r\n", channel1, channel2);

            while (cnt--) {
                duty_cycle1 = duty_cycle1 - 100;

                bk_pwm_cw_update_param(channel1, channel2, cycle, duty_cycle1, duty_cycle2,
dead_band);
                rtos_delay_milliseconds(10);

                if (duty_cycle1 == 0)
                    duty_cycle1 = cycle;
            }
        }
        #endif
        #endif
}
```

5.4 操作说明

5.4.1 打开配置

bk7231n示例代码参考bk_peripheral_test.c中cli 命令串口，打开宏定义：CFG_PERIPHERAL_TEST，编译完成后，将固件下载至设备。

5.4.2 运行现象

串口输入命令 ：pwm single/update/cw channel1 duty_cycle1 cycle（channel2 duty_cycle1 dead_band）即可在对应的gpio上检测到波形。分别输入channel:1~5的命令，即可用逻辑分析仪看到相应的gpio输出周期性的方波。如下图所示

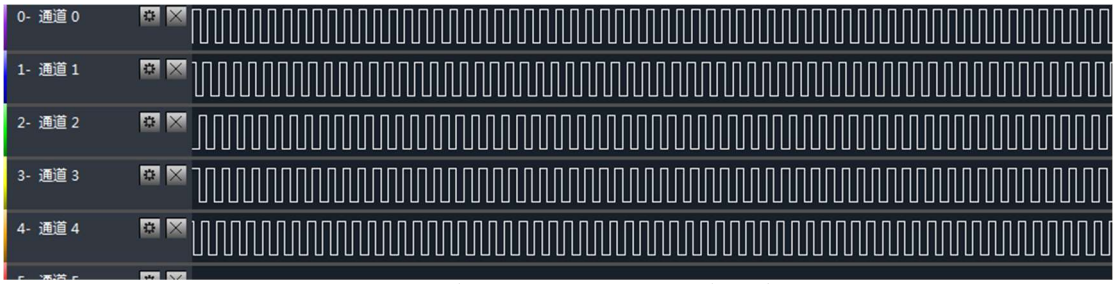

图5.4.2-1：5个pwm channel同时输出波形图

5.5 注意事项

- pwm channel0 可能已经被cpu用来做timer，所以其pwm功能不能使用。
- pwm输出的时候，其时钟源选择的是26M。

6 GPIO

6.1 GPIO简介

BEKEN wifi soc 的引脚一般分为4 类：电源、时钟、模拟/控制与I/O，I/O 口在使用模式上又分为General Purpose Input Output（通用输入/ 输出），简称GPIO，与功能复用I/O（如SPI/I2C/UART 等）。

6.2 GPIO Related API

Freertos中gpio api 位于beken378\func\user_driver\BkDriverGpio.h相关接口如下：

函数	描述
BkGpioInitialize ()	gpio初始化
BkGpioFinalize ()	终止使用gpio
BkGpioOutputHigh ()	gpio输出高电平
BkGpioOutputLow ()	gpio输出低电平
BKGpioOp ()	设置gpio模式
BkGpioInputGet ()	获取gpio输入值
BkGpioEnableIRQ ()	使能gpio中断
BkGpioDisableIRQ()	关闭gpio中断

6.2.1 gpio枚举类型说明

bk_gpio_t： gpio 0-31引脚编号

bk_gpio_config_t 枚举类型说明

INPUT_PULL_UP	下拉输入模式
INPUT_PULL_DOWN	上拉输入模式
INPUT_NORMAL	正常输入模式（既不上拉也不下拉）
OUTPUT_NORMAL	正常输出模式
GPIO_SECOND_FNNC	第二功能使能模式

bk_gpio_irq_trigger_t 枚举类型说明

IRQ_TRIGGER_LOW_LEVEL	低电平触发中断
IRQ_TRIGGER_HGIH_LEVEL	高电平触发中断
IRQ_TRIGGER_RISING_EDGE	上升沿触发中断
IRQ_TRIGGER_FALLING_EDGE	下降沿触发中断

6.2.2 设初始化gpio，设置gpio模式

引脚在使用前需要先设置好输入或者输出模式，通过如下函数完成：

```
OSStatus BkGpioInitialize( bk_gpio_t gpio, bk_gpio_config_t configuration );
```

参数	描述
gpio	gpio引脚编号
configuration	gpio引脚工作模式
返回	0：成功；-1：错误

6.2.3 终止使用gpio

```
OSStatus BkGpioFinalize ( bk_gpio_t gpio );
```

参数	描述
gpio	gpio引脚编号
返回	0：成功；-1：错误

6.2.4 设置gpio输出高电平

```
OSStatus BkGpioOutputHigh( bk_gpio_t gpio );
```

参数	描述
gpio	gpio引脚编号
返回	0：成功；-1：错误

6.2.5 设置gpio输出低电平

```
OSStatus BkGpioOutputLow( bk_gpio_t gpio );
```

参数	描述
gpio	gpio引脚编号
返回	0：成功；-1：错误

6.2.6 设置gpio模式

```
OSStatus BKGpioOp(char cmd, uint32_t id, char mode);
```

参数	描述
cmd	gpio设置命令
id	gpio引脚
mode	gpio设置模式
返回	0：成功；-1：错误

6.2.7 获取gpio输入电平

```
OSStatus BkGpioInputGet ( bk_gpio_t gpio );
```

参数	描述
gpio	gpio引脚编号
返回	0：低电平；1：高电平

6.2.8 使能gpio中断触发功能

```
OSStatus BkGpioEnableIRQ( bk_gpio_t gpio, bk_gpio_irq_trigger_t trigger,
bk_gpio_irq_handler_t handler, void *arg );
```

参数	描述
gpio	gpio引脚编号
trigger	gpio触发类型
handler	gpio注册的中断回调函数
返回	0：成功；-1：错误

6.2.9 停止gpio中断触发功能

```
OSStatus BkGpioDisableIRQ( bk_gpio_t gpio );
```

参数	描述
gpio	gpio引脚编号
返回	0：低电平；1：高电平

6.3 GPIO示例代码

freertos中示例代码参考wlan_cli.c中 Gpio_op_Command 和 Gpio_int_Command命令；

6.3.1 关键说明

通过发送Gpio_op_Command中的命令是对应的gpio处于同的模式，通过发送Gpio_int_Command中的命令是测试gpio中断可以使用不用的模式触发。

6.3.2 示例代码

```
/*
CMD FORMAT: GPIO CMD index PARAM
exmaple:GPIO 0 18 2                    (config GPIO18 input & pull-up)
*/
static void Gpio_op_Command(char *pcWriteBuffer, int xWriteBufferLen, int argc, char **argv)
{
    uint32_t ret, id, mode, i;
    char cmd0 = 0;
    char cmd1 = 0;
    char cmd;

    for(i = 0; i < argc; i++)
    {
        os_printf("Argument %d = %s\r\n", i + 1, argv[i]);
    }

    if(argc == 4)
    {
        cmd = argv[1][0];
        mode = argv[3][0];

        cmd0 = argv[2][0] - 0x30;
        cmd1 = argv[2][1] - 0x30;

        id = (uint32_t)(cmd0 * 10 + cmd1);
        os_printf("---%x,%x----\r\n", id, mode);
        ret = BKGpioOp(cmd, id, mode);
        os_printf("gpio op:%x\r\n", ret);
    }
    else
        os_printf("cmd param error\r\n");
}
```

```
void test_fun(char para)
{
    os_printf("---%d---\r\n", para);
}
/*
cmd format: GPIO_INT cmd index    triggermode
enable: GPIO_INT 1 18 0
*/
static void Gpio_int_Command(char *pcWriteBuffer, int xWriteBufferLen, int argc, char **argv)
{
    uint32_t id, mode;
    char cmd0 = 0;
    char cmd1 = 0;
    char cmd;

    if(argc == 4)
    {
        cmd = argv[1][0] - 0x30;
        mode = argv[3][0] - 0x30;

        cmd0 = argv[2][0] - 0x30;
        cmd1 = argv[2][1] - 0x30;

        id = (uint32_t)(cmd0 * 10 + cmd1);
        BKGpioIntcEn(cmd, id, mode, test_fun);
    }
    else
        os_printf("cmd param error\r\n");

}
```

7 Flash

7.1 Flash简介

Flash是存储数据和代码的存储模块，支持写平衡和掉电保护，保证了产品在后期升级时拥有更好的扩展性。

7.2 Flash Related API

Flash相关接口参考beken378\func\user_driver\BkDriverFlash.h，相关接口如下：

函数	描述
bk_flash_erase()	擦除flash
bk_flash_write ()	写flash
bk_flash_read ()	读取flash
bk_flash_enable_security()	保护flash

7.2.1 flash枚举类型说明

bk_partition_t：flash分配区域；

BK_PARTITION_BOOTLOADER	bootloader 分区
BK_PARTITION_APPLICATION	应用分区
BK_PARTITION_OTA	ota分区
BK_PARTITION_RF_FIRMWARE	rf firmware分区
BK_PARTITION_NET_PARAM	网络参数分区
BK_PARTITION_USR_CONFIG	用户配置分区
BK_PARTITION_MAX	最大分区

PROTECT_TYPE ：flash 保护类型

FLASH_PROTECT_NONE	不保护
FLASH_PROTECT_ALL	全保护
FLASH_PROTECT_HALF	半保护
FLASH_UNPROTECT_LAST_BLOCK	最后一个block不保护

7.2.1 擦除flash

OSStatus bk_flash_erase(bk_partition_t inPartition, uint32_t off_set, uint32_t size);

参数	描述
inPartition	分区位置
off_set	偏移地址
size	擦除大小
返回	0：成功；其他：失败

7.2.2 写入 flash

OSStatus bk_flash_write(bk_partition_t inPartition, volatile uint32_t off_set, uint8_t *inBuffer , uint32_t inBufferLength);

参数	描述
inPartition	分区位置
off_set	偏移地址
inBuffer	写入的数据buffer
inBufferLength	擦除大小
返回	0：成功；其他：失败

7.2.3 读取 flash

OSStatus bk_flash_read(bk_partition_t inPartition, volatile uint32_t off_set, uint8_t *outBuffer, uint32_t inBufferLength);

参数	描述
inPartition	分区位置
off_set	偏移地址
inBuffer	读取的数据buffer
inBufferLength	擦除大小
返回	0：成功；其他：失败

7.2.4 保护/解保护 flash

OSStatus bk_flash_enable_security(PROTECT_TYPE type);

参数	描述

void	空
返回	0：成功；其他：失败

7.3 Flash示例代码

freertos中示例代码参考wlan_cli.c中flash_command_test命令：

7.3.1 关键说明

通过发送flash_command_test中的flash擦除，flash的写入和读取，检查flash的写入和读取操作是否正常。

7.3.2 示例代码

```
/*
format: FLASH    E/R/W    0xABCD
example:         FLASH    R    0x00100

*/

extern OSStatus test_flash_write(volatile uint32_t start_addr, uint32_t len);
extern OSStatus test_flash_erase(volatile uint32_t start_addr, uint32_t len);
extern OSStatus test_flash_read(volatile uint32_t start_addr, uint32_t len);
extern OSStatus test_flash_read_time(volatile uint32_t start_addr, uint32_t len);

static void flash_command_test(char *pcWriteBuffer, int xWriteBufferLen, int argc, char **argv)
{
    char cmd = 0;
    uint32_t len = 0;
    uint32_t addr = 0;

    if(argc == 4)
    {
        cmd = argv[1][0];
          addr = atoi(argv[2]);
          len = atoi(argv[3]);

          switch(cmd)
```

```
        {
            case 'E':
                bk_flash_enable_security(FLASH_PROTECT_NONE);
                test_flash_erase(addr,len);
                bk_flash_enable_security(FLASH_UNPROTECT_LAST_BLOCK);
             break;

          case 'R':
                test_flash_read(addr,len);
              break;
            case 'W':
                bk_flash_enable_security(FLASH_PROTECT_NONE);
                test_flash_write(addr,len);
                bk_flash_enable_security(FLASH_UNPROTECT_LAST_BLOCK);
             break;
    //to check whether protection mechanism can work
             case 'N':
                test_flash_erase(addr,len);
                break;
             case 'M':
                test_flash_write(addr,len);
                break;
            case 'T':
                test_flash_read_time(addr,len);
                break;
          default:
            break;
        }
    }
     else
     {
        os_printf("FLASH <R/W/E/M/N/T> <start_addr> <len>\r\n");
     }
}
```

8 网络接口

8.1 网络接口简介

BKEN wifi soc的SDK给上层应用提供的网络接口用于：

1.启动STATION模式，去连接指定的网络。
2.关闭STATION模式。
3.启动AP模式，供其他设备连接。
4.关闭AP模式。
5.启动监听模式，供上层配网。
6.关闭监听模式。
7.获取状态，如连接状态，加密方式，当前使用的信道等等。
8.设置状态，如设置信道，IP地址等等。
9.启动scan，并获取scan结果。

8.2 网络接口 Related API

网络接口相关接口参考\beken378\func\include\wlan_ui_pub.h，应用程序可通过以下APIs控制网络，相关接口如下所示：

函数	描述
bk_wlan_start()	启动网络，包括STATION和AP
bk_wlan_start_sta_adv()	启动STATION快速连接
bk_wlan_stop()	关闭网络，包括STATION和AP
bk_wlan_start_scan()	启动scan
bk_wlan_scan_ap_reg_cb()	注册scan结束后的回调函数
bk_wlan_start_assign_scan()	scan特定的网络
bk_wlan_start_monitor()	启动监听模式
bk_wlan_stop_monitor()	关闭监听模式
bk_wlan_register_monitor_cb()	注册监听回调函数
bk_wlan_get_ip_status()	获取当前的网络状态
bk_wlan_get_link_status()	获取当前的连接状态
bk_wlan_get_channel()	获取当前的信道
bk_wlan_set_channel()	设置信道

8.2.1 启动网络

上层应该获得ssid与password之后，可以启动网络。通过如下函数完成：

OSStatus bk_wlan_start(network_InitTypeDef_st *inNetworkInitPara);

参数	描述
inNetworkInitPara	传入需要配置信息
返回	kNoErr：成功；其他：失败

参数类型		
network_InitTypeDef_st：		
char	wifi_mode	WiFi模式
char	wifi_ssid[33]	需要连接或建立的网络SSID
char	wifi_key[64]	需要连接或建立的网络密码
char	local_ip_addr[16]	静态IP地址，在DHCP关闭时有效
char	net_mask[16]	静态子网掩码，在DHCP关闭时有效
char	gateway_ip_addr[16]	静态网关地址，在DHCP关闭时有效
char	dns_server_ip_addr[16]	静态DNS地址，在DHCP关闭时有效
char	dhcp_mode	DHCP模式
char	reserved[32]	保留
Int	wifi_retry_interval	重连间隔，单位是毫秒

8.2.2 启动STATION快速连接

OSStatus bk_wlan_start_sta_adv(network_InitTypeDef_adv_st *inNetworkInitParaAdv);

参数	描述
inNetworkInitParaAdv	需要传入的网络参数
返回	kNoErr：成功；其他：失败

参数类型		
network_InitTypeDef_adv_st:		
apinfo_adv_t	ap_info	需要快速连接的网络信息
char	key[64]	需要快速连接的网络密码
Int	key_len	网络密码长度
char	local_ip_addr[16]	静态IP地址，在DHCP关闭时有效
char	net_mask[16]	静态子网掩码，在DHCP关闭时有效
char	gateway_ip_addr[16]	静态网关地址，在DHCP关闭时有效
char	dns_server_ip_addr[16]	静态DNS地址，在DHCP关闭时有效

char	**dhcp_mode**	DHCP模式
char	**reserved[32]**	保留
int	**wifi_retry_interval**	重连时间，单位是毫秒

apinfo_adv_st:

char	**ssid[32]**	需要快速连接的网络信息
char	**bssid[6]**	需要快速连接的网络密码
uint8_t	**channel**	网络密码长度
wlan_sec_type_t	**local_ip_addr[16]**	静态IP地址，在DHCP关闭时有效 typedef uint8_t wlan_sec_type_t

8.2.3 关闭网络

```
int bk_wlan_stop(char mode);
```

参数	描述
mode	需要关闭的模式，见枚举类型中关于mode的说明
返回	kNoErr：成功；其他：失败

8.2.4 启动scan

```
void bk_wlan_start_scan(void);
```

参数	描述
void	无
返回	无

8.2.5 注册scan结束后的回调函数

```
void bk_wlan_scan_ap_reg_cb(FUNC_2PARAM_PTR ind_cb);
```

参数	描述
ind_cb	scan结束后回调的函数。函数定义： typedef void (*FUNC_2PARAM_PTR)(void *arg, uint8_t vif_idx);
返回	无

8.2.6 scan特定的网络

```
void bk_wlan_start_assign_scan(UINT8 **ssid_ary, UINT8 ssid_num);
```

参数	描述
ssid_ary	指定网络的SSID
ssid_num	指定网络的数量
返回	无

8.2.7 启动监听模式

```
int bk_wlan_start_monitor(void);
```

参数	描述
void	无
返回	kNoErr：成功；其他：失败

8.2.8 关闭监听模式

```
int bk_wlan_stop_monitor(void);
```

参数	描述
void	无
返回	kNoErr：成功；其他：失败

8.2.9 注册监听回调函数

```
void bk_wlan_register_monitor_cb(monitor_data_cb_t fn);
```

参数	描述
fn	注册的回调函数。函数定义： typedef void (*monitor_data_cb_t)(uint8_t *data, int len, hal_wifi_link_info_t *info);
返回	无

8.2.10 获取当前的网络状态

```
OSStatus bk_wlan_get_ip_status(IPStatusTypedef *outNetpara, WiFi_Interface inInterface);
```

参数	描述
outNetpara	保存获取的网络状态。
inInterface	需要获取网络状态的模式。

返回	kNoErr：成功；其他：失败

参数类型		
IPStatusTypedef:		
uint8_t	**dhcp**	获取的DHCP模式
char	**ip[16]**	获取的IP地址
char	**gate[16]**	获取的网关IP地址
char	**mask[16]**	获取的子网掩码
char	**dns[16]**	DNS服务IP地址
char	**mac[16]**	获取的mac地址
char	**broadcastip[16]**	获取的广播IP地址

#define WiFi_Interface wlanInterfaceTypedef

```
typedef enum
{
    SOFT_AP,                    /*AP模式*/
    STATION                     /*STATION模式*/
} wlanInterfaceTypedef;
```

8.2.11 获取当前的连接状态

OSStatus bk_wlan_get_link_status(LinkStatusTypeDef *outStatus);

参数	描述
outStatus	保存获取的连接状态。具体参考该结构体的说明。
返回	kNoErr：成功；其他：失败

参数类型		
LinkStatusTypeDef:		
msg_sta_states	**conn_state**	当前连接状态
int	**wifi_strength**	当前的信号强度
uint8_t	**ssid[32]**	当前网络的SSID
uint8_t	**bssid[6]**	当前网络的BSSID
int	**channel**	当前网络的信道
wlan_sec_type_t	**security**	当前网络的加密方式 Typedef uint8_t wlan_sec_type_t

```
typedef enum {
    MSG_IDLE = 0,                   /*未任何连接状态*/
```

```
    MSG_CONNECTING,          /*正在连接中*/
    MSG_PASSWD_WRONG,        /*密码错误*/
    MSG_NO_AP_FOUND,         /*未找到要连接的网络*/
    MSG_CONN_FAIL,           /*连接失败*/
    MSG_CONN_SUCCESS,        /*连接成功*/
    MSG_GOT_IP,              /*获得IP*/
}msg_sta_states;
```

8.2.12 获取当前的信道

int bk_wlan_get_channel(void);

参数	描述
void	无
返回	channel

8.2.13 设置信道

int bk_wlan_set_channel(int channel);

参数	描述
channel	传入的信道数值
返回	0：成功；其他：失败

8.3 网络接口使用示例

8.3.1 关键说明

- **DHCP宏定义说明**

#define DHCP_DISABLE	**(0)**	/*DHCP关闭*/
#define DHCP_CLIENT	**(1)**	/*DHCP客户端模式*/
#define DHCP_SERVER	**(2)**	/* DHCP服务端模式*/

8.3.2 代码示例

启动一个STATION连接:

```
void demo_sta_app_init(char *oob_ssid,char *connect_key)
{
    /*定义一个结构体，用于传入参数*/
    network_InitTypeDef_st wNetConfig;
```

```
    int len;
    /*把这个结构体置空*/
    os_memset(&wNetConfig, 0x0, sizeof(network_InitTypeDef_st));

    /*检查SSID的长度，不能超过32字节*/
    len = os_strlen(oob_ssid);
    if(SSID_MAX_LEN < len)
    {
        bk_printf("ssid name more than 32 Bytes\r\n");
        return;
    }

    /*将SSID跟密码传入结构体*/
     os_strcpy((char *)wNetConfig.wifi_ssid, oob_ssid);
     os_strcpy((char *)wNetConfig.wifi_key, connect_key);

    /*当前为客户端模式*/
     wNetConfig.wifi_mode = STATION;
    /*采用DHCP CLIENT的方式获得，从路由器动态获取IP地址*/
     wNetConfig.dhcp_mode = DHCP_CLIENT;
     wNetConfig.wifi_retry_interval = 100;

     bk_printf("ssid:%s key:%s\r\n", wNetConfig.wifi_ssid, wNetConfig.wifi_key);
    /*启动WiFi连接*/
     bk_wlan_start(&wNetConfig);
}
```

启动AP模式，提供其他客户端连接：

```
void demo_softap_app_init(char *ap_ssid,char *ap_key)
{
    /*定义一个结构体，用于传入参数*/
     network_InitTypeDef_adv_st    wNetConfigAdv;
      int len;
    /*将结构体置空*/
     os_memset( &wNetConfigAdv, 0x0, sizeof(network_InitTypeDef_adv_st) );
     len = os_strlen(ap_ssid);
      if(SSID_MAX_LEN < len)
      {
          bk_printf("ssid name more than 32 Bytes\r\n");
```

```
        return;
    }
    /*传入要连接的ap ssid  和  ap key*/
    os_strcpy((char *)wNetConfig.wifi_ssid, ap_ssid);
    os_strcpy((char *)wNetConfig.wifi_key, ap_key);

    /*当前为ap模式*/
    wNetConfig.wifi_mode = SOFT_AP;
    /*采用DHCP SERVER模式，需要将静态地址分配为本地地址*/
    wNetConfig.dhcp_mode = DHCP_SERVER;
    wNetConfig.wifi_retry_interval = 100;
    os_strcpy((char *)wNetConfig.local_ip_addr, WLAN_DEFAULT_IP);
    os_strcpy((char *)wNetConfig.net_mask, WLAN_DEFAULT_MASK);
    os_strcpy((char *)wNetConfig.dns_server_ip_addr, WLAN_DEFAULT_IP);

    bk_printf("ssid:%s    key:%s\r\n", wNetConfig.wifi_ssid, wNetConfig.wifi_key);
    /*启动ap*/
    bk_wlan_start(&wNetConfig);}
```

启动STATION的快速连接：

```
void demo_sta_adv_app_init(char *oob_ssid,char *connect_key)
{
    /*定义一个结构体，用于传入参数*/
    network_InitTypeDef_adv_st    wNetConfigAdv;
    /*将结构体置空*/
    os_memset( &wNetConfigAdv, 0x0, sizeof(network_InitTypeDef_adv_st) );
    /*传入要连接的SSID*/
    os_strcpy((char*)wNetConfigAdv.ap_info.ssid, oob_ssid);
    /*传入要连接的网络的bssid，下面这个bssid仅供参考*/
    hwaddr_aton("12:34:56:00:00:01", wNetConfigAdv.ap_info.bssid);
    /*要连接网络的加密方式。具体参数参考该结构体说明。*/
    wNetConfigAdv.ap_info.security = SECURITY_TYPE_WPA2_MIXED;
    /*要连接的网络的信道*/
    wNetConfigAdv.ap_info.channel = 11;
    /*要连接的网络密码以及密码长度*/
    os_strcpy((char*)wNetConfigAdv.key, connect_key);
    wNetConfigAdv.key_len = os_strlen(connect_key);
    /*通过DHCP的方式获取IP地址等网络信息*/
    wNetConfigAdv.dhcp_mode = DHCP_CLIENT;
```

```
    wNetConfigAdv.wifi_retry_interval = 100;
    /*启动快速连接*/
    bk_wlan_start_sta_adv(&wNetConfigAdv);
}
```

启动scan，并分析scan的结果：

```
/*回调函数，用于scan结束后解析scan结果*/
static void scan_cb(void *ctxt, uint8_t param)
{
    /*指向scan结果的指针*/
    struct scanu_rst_upload *scan_rst;
    /*保存解析结果的结构体*/
    ScanResult apList;
    int i;

    apList.ApList = NULL;
    /*启动scan*/
    scan_rst = sr_get_scan_results();
    /*如果什么都没有scan到，返回；否则记录scan到的网络数量*/
    if( scan_rst == NULL )
    {
        apList.ApNum = 0;
        return;
    }
    else
    {
        apList.ApNum = scan_rst->scanu_num;
    }
    if( apList.ApNum > 0 )
    {
        /*申请对应的内存，用于保存scan的结果*/
        apList.ApList = (void *)os_malloc(sizeof(*apList.ApList) * apList.ApNum);
        for( i = 0; i < scan_rst->scanu_num; i++ )
        {
            /*将scan到的网络ssid与rssi记录下来*/
            os_memcpy(apList.ApList[i].ssid, scan_rst->res[i]->ssid, 32);
            apList.ApList[i].ApPower = scan_rst->res[i]->level;
        }
    }
```

```
    if( apList.ApList == NULL )
    {
        apList.ApNum = 0;
    }
     /*打印scan的结果*/
     bk_printf("Got ap count: %d\r\n", apList.ApNum);
     for( i = 0; i < apList.ApNum; i++ )
     {
          if(os_strlen(apList.ApList[i].ssid) >= SSID_MAX_LEN)
          {
               char temp_ssid[33];
               os_memset(temp_ssid, 0, 33);
               os_memcpy(temp_ssid, apList.ApList[i].ssid, 32);
               bk_printf("       %s, RSSI=%d\r\n", temp_ssid, apList.ApList[i].ApPower);
          }
          else
          {
               bk_printf("       %s, RSSI=%d\r\n", apList.ApList[i].ssid, apList.ApList[i].ApPower);
          }
     }
     bk_printf("Get ap end.....\r\n\r\n");

    /*结束后释放申请的内存*/
    if( apList.ApList != NULL )
    {
        os_free(apList.ApList);
        apList.ApList = NULL;
    }

#if CFG_ROLE_LAUNCH
     rl_pre_sta_set_status(RL_STATUS_STA_LAUNCHED);
#endif
    sr_release_scan_results(scan_rst);
}

void demo_scan_app_init(void)
{
    /*注册scan回调函数*/
     mhdr_scanu_reg_cb(scan_cb, 0);
```

```
    /*开始scan*/
     bk_wlan_start_scan();
}
```

连接成功后，获取连接后的网络状态

```
void demo_ip_app_init(void)
{
    /*定义一个用于保存网络状态的结构体*/
    IPStatusTypedef ipStatus;
    /*将该结构体置空*/
     os_memset(&ipStatus, 0x0, sizeof(IPStatusTypedef));
    /*获取网络状态，并保存在该结构体中*/
     bk_wlan_get_ip_status(&ipStatus, STATION);
    /*打印获取的网络状态*/
     bk_printf("dhcp=%d ip=%s gate=%s mask=%s mac=" MACSTR "\r\n",
                        ipStatus.dhcp, ipStatus.ip, ipStatus.gate,
                        ipStatus.mask, MAC2STR((unsigned char*)ipStatus.mac));
}
```

连接成功后，获取连接状态：

```
void demo_state_app_init(void)
{
    /*定义结构体用于保存连接状态*/
     LinkStatusTypeDef linkStatus;
    network_InitTypeDef_ap_st ap_info;
    char ssid[33] = {0};
    #if CFG_IEEE80211N
        bk_printf("sta: %d, softap: %d, b/g/n\r\n",sta_ip_is_start(),uap_ip_is_start());
    #else
        bk_printf("sta: %d, softap: %d, b/g\r\n",sta_ip_is_start(),uap_ip_is_start());
    #endif

    /*STATION模式下的连接状态*/
    if( sta_ip_is_start() )
    {
    /*将用于保存状态的结构体置空*/
     os_memset(&linkStatus, 0x0, sizeof(LinkStatusTypeDef));
    /*获取连接状态*/
     bk_wlan_get_link_status(&linkStatus);
```

```
/*打印连接状态*/
os_memcpy(ssid, linkStatus.ssid, 32);

 bk_printf("sta:rssi=%d,ssid=%s,bssid=" MACSTR ",channel=%d,cipher_type:",
        linkStatus.wifi_strength, ssid, MAC2STR(linkStatus.bssid), linkStatus.channel);
      switch(bk_sta_cipher_type())
      {
          case SECURITY_TYPE_NONE:
                bk_printf("OPEN\r\n");
                break;
           case SECURITY_TYPE_WEP :
                bk_printf("WEP\r\n");
                break;
           case SECURITY_TYPE_WPA_TKIP:
                bk_printf("TKIP\r\n");
                break;
           case SECURITY_TYPE_WPA2_AES:
                bk_printf("CCMP\r\n");
                break;
           case SECURITY_TYPE_WPA2_MIXED:
                bk_printf("MIXED\r\n");
                break;
           case SECURITY_TYPE_AUTO:
                bk_printf("AUTO\r\n");
                break;
           default:
                bk_printf("Error\r\n");
                break;
      }
}
/*AP模式下的连接状态*/
if( uap_ip_is_start() )
{
/*将用于保存连接状态的结构体置空*/
 os_memset(&ap_info, 0x0, sizeof(network_InitTypeDef_ap_st));
/*获取连接状态*/
 bk_wlan_ap_para_info_get(&ap_info);
/*打印出获取的连接状态值*/
os_memcpy(ssid, ap_info.wifi_ssid, 32);
```

```
    bk_printf("softap:ssid=%s,channel=%d,dhcp=%d,cipher_type:",
    ssid, ap_info.channel,ap_info.dhcp_mode);
    switch(ap_info.security)
        {
            case SECURITY_TYPE_NONE:
                bk_printf("OPEN\r\n");
                break;
             case SECURITY_TYPE_WEP :
                bk_printf("WEP\r\n");
                break;
             case SECURITY_TYPE_WPA_TKIP:
                bk_printf("TKIP\r\n");
                break;
             case SECURITY_TYPE_WPA2_AES:
                bk_printf("CCMP\r\n");
                break;
             case SECURITY_TYPE_WPA2_MIXED:
                bk_printf("MIXED\r\n");
                break;
             case SECURITY_TYPE_AUTO:
                bk_printf("AUTO\r\n");
                break;
             default:
                bk_printf("Error\r\n");
                break;
        }
        bk_printf("ip=%s,gate=%s,mask=%s,dns=%s\r\n",
                ap_info.local_ip_addr, ap_info.gateway_ip_addr, ap_info.net_mask,
ap_info.dns_server_ip_addr);
    }
}

/* monitor  回调函数*/
void bk_demo_monitor_cb(uint8_t *data, int len, hal_wifi_link_info_t *info)
{
    os_printf("len:%d\r\n", len);

    //Only for reference
    /*
```

```
        User can get ssid and key by prase monitor data,
        refer to the following code, which is the way airkiss
        use monitor get wifi info from data
        */
#if 0
        int airkiss_recv_ret;
        airkiss_recv_ret = airkiss_recv(ak_contex, data, len);
#endif
}
/*  程序清单： 这是一个简单网络接口程序使用例程
 * 命令调用格式： wifi_demo  sta oob_ssid  connect_key
 * 程序功能： 输入相关命令可以启动网络，连接网络等。
*/
int wifi_demo(int argc, char **argv)
{
        char *oob_ssid = NULL;
        char *connect_key;

      if (strcmp(argv[1], "sta") == 0)
      {
              os_printf("sta_Command\r\n");
              if (argc == 3)
              {
                      oob_ssid = argv[2];
                      connect_key = "1";
              }
              else if (argc == 4)
              {
                      oob_ssid = argv[2];
                      connect_key = argv[3];
              }
              else
              {
                      os_printf("parameter invalid\r\n");
                      return -1;
              }
              if(oob_ssid)
              {
                      demo_sta_app_init(oob_ssid, connect_key);
```

```
        }
        return 0;
    }
    if(strcmp(argv[1], "adv") == 0)
    {
        os_printf("sta_adv_Command\r\n");
        if (argc == 3)
        {
            oob_ssid = argv[1];
            connect_key = "1";
        }
        else if (argc == 4)
        {
            oob_ssid = argv[1];
            connect_key = argv[2];
        }
        else
        {
            os_printf("parameter invalid\r\n");
            return -1;
        }
        if(oob_ssid)
        {
            demo_sta_adv_app_init(oob_ssid, connect_key);
        }
        return 0;
    }
    if(strcmp(argv[1], "softap") == 0)
    {
        os_printf("SOFTAP_COMMAND\r\n\r\n");
        if (argc == 3)
        {
            oob_ssid = argv[1];
            connect_key = "1";
        }
        else if (argc == 4)
        {
            oob_ssid = argv[1];
            connect_key = argv[2];
```

```
            }
            else
            {
                os_printf("parameter invalid\r\n");
                return -1;
            }
            if(oob_ssid)
            {
                demo_softap_app_init(oob_ssid, connect_key);
            }
            return 0;
        }
        if(strcmp(argv[1], "monitor") == 0)
        {
            if(argc != 3)
            {
                os_printf("parameter invalid\r\n");
            }
            if(strcmp(argv[2], "start") == 0)
            {
                bk_wlan_register_monitor_cb(bk_demo_monitor_cb);
                bk_wlan_start_monitor();
            }
            else if(strcmp(argv[2], "stop") == 0)
            {
                bk_wlan_stop_monitor();
            }
            else
            {
                os_printf("parameter invalid\r\n");
            }
        }
        return 0;
}
MSH_CMD_EXPORT(wifi_demo, wifi_demo command);
```

8.4 操作说明

本节的示例代码均位于\beken378\demo\ieee802_11_demo.c,系统默认已经打开此功能，设备上电后，在调试串口输入相应指令即可运行不同程序。

8.4.1 启动STATION连接

设备上电后，调试串口输入wifi_demo sta your_ssid your_key,设备开始连接路由器。

```
wifi_demo sta your_ssid your_key
sta_Command
ssid: your_ssid key: your_key
rl_sta_start
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
hapd_intf_add_vif,type:2, s:0, id:0
 [wlan_connect]:start tick =  0, connect done tick = 22379, total = 22379
[wlan_connect]:start tick =  0, connect done tick = 22385, total = 22385
[WLAN_MGNT]wlan sta connected evenew dtim period:2
nt callback
IP UP: 192.168.44.27
[ip_up]:start tick =  0, ip_up tick = 25797, total = 25797
```

8.4.2 启动STATION快速连接

设备上电后，在调试串口输入wifi_demo adv your_ssid your_key,设备开始连接路由器，设备log如下：

```
wifi_demo adv your_ssid your_key
sta_adv_Command
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
bssid 48-ee-0c-48-93-12
security2cipher 2 3 24 8 security=6
cipher2security 2 3 24 8
---------SM_CONNECT_IND_ok
wpa_driver_assoc_cb
Cancelling scan request
hapd_intf_add_key CCMP
add sta_mgmt_get_sta
sta:1, vif:0, key:0
```

```
sta_mgmt_add_key
add hw key idx:25
add TKIP
add is_broadcast_ether_addr
sta:255, vif:0, key:1
add hw key idx:1
ctrl_port_hdl:1
[wlan_connect]:start tick =    0, connect done tick = 31898, total = 31898
[wlan_connect]:start tick =    0, connect done tick = 31904, total = 31904
[WLAN_MGNT]wlan sta connected event callback
sta_ip_start
configuring interface mlan (with DHCP client)
dhcp_check_status_init_timer
IP UP: 192.168.44.49
[ip_up]:start tick =    0, ip_up tick = 35292, total = 35292
```

8.4.3 STATION模式获取状态

- 获取网络状态

连接路由器，方法参考16.4.1小节，然后串口输入wifi_demo status net获取设备当前网络状态，设备log如下：

```
msh />
msh />wifi_demo status net
dhcp=0 ip=192.168.44.52 gate=192.168.44.119 mask=255.255.255.0 mac=c8:47:8c:2f:4b:d2
```

图5.4.3-1

- 获取连接状态

连接路由器，方法参考16.4.1小节，然后串口输入wifi_demo status link获取设备当前连接状态，设备log如下：

```
msh />
msh />wifi_demo status link
sta: 1, softap: 0, b/g/n
sta:rssi=-65,ssid=wifi-team,bssid=c0:3f:0e:c7:91:4c ,channel=11,cipher_type:MIXED
```

图5.4.3-2

8.4.4 启动AP

设备上电后，在调试串口输入wifi_demo softap beken 12345678,设备开始连接路由器，设备log如下：

```
wifi_demo softap beken 12345678
SOFTAP_COMMAND

ssid:beken  key:12345678
rl_ap_start
Soft_AP_start
[saap]MM_RESET_REQ
[saap]ME_CONFIG_REQ
[saap]ME_CHAN_CONFIG_REQ
[saap]MM_START_REQ
hapd_intf_add_vif,type:3, s:0, id:0
apm start with vif:0
------beacon_int_set:100 TU
update_ongoing_1_bcn_update
vif_idx:0, ch_idx:0, bcmc_idx:2
update_ongoing_1_bcn_update
```

图5.4.4-1

8.4.5 AP模式获取状态

- **获取网络状态**

连接路由器，方法参考16.4.4小节，然后串口输入wifi_demo status net获取设备当前网络状态，设备log如下：

```
msh />
msh />wifi_demo status net
dhcp=0 ip=20.240.159.229 gate=20.240.159.229 mask=20.240.159.229 mac=00:00:00:00:00:00
```

图5.4.5-1

- **获取连接状态**

连接路由器，方法参考16.4.4小节，然后串口输入wifi_demo status link获取设备当前连接状态，设备log如下：

```
msh />wifi_demo status link
sta: 0, softap: 1, b/g/n
softap:ssid=beken,channel=11,dhcp=1,cipher_type:CCMP
ip=192.168.0.1,gate=255.255.255.255,mask=255.255.255.0,dns=0.0.0.0
```

图5.4.5-2

8.4.6 启动SCAN

- **扫描WIFI热点**

设备上电后，在调试串口输入wifi_demo scan,设备开始扫描附近WIFI热点，设备log如下：

```
msh />wifi_demo scan
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
scan_start_req_handler
msh />ethernetif_input no netif found 255
Got ap count: 13
    Xiaomi_E21E, RSSI=210
    labast, RSSI=206
    HUAWEI-EZ7HKY, RSSI=204
    FAST_5AC4, RSSI=200
    ssid-tcj, RSSI=197
    B-LINK_F11566, RSSI=196
    antbang_195F4C, RSSI=193
    bk7252_smart, RSSI=193
    wifi-team, RSSI=192
    beken_airport, RSSI=189
    Honor Magic 2, RSSI=189
    Bekencorp-Guest, RSSI=187
    Bekencorp-WIFI, RSSI=177
Get ap end.......
```

图5.4.6-1

- **扫描指定WIFI热点**

设备上电后，在调试串口输入wifi_demo scan Bekencorp-WIFI,设备开始扫描Bekencorp-WIFI，设备log如下：

```
msh />wifi_demo scan Bekencorp-WIFI
scan for ssid:Bekencorp-WIFI
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
scan_start_req_handler
msh />Got ap count: 1
    Bekencorp-WIFI, RSSI=186
Get ap end.......
```

8.4.7 启动混杂包监听

设备上电后，在调试串口输入wifi_demo monitor start,设备开始监听混杂包，输入wifi_demo monitor stop,停止监听，设备log如下：

```
wifi_demo adv your_ssid your_key
msh />wifi_demo monitor
parameter invalid
parameter invalid
msh />wifi_demo monitor start
net_wlan_add_netif not vif idx found
Soft_AP_start
[saap]MM_RESET_REQ
[saap]ME_CONFIG_REQ
[saap]ME_CHAN_CONFIG_REQ
[saap]MM_START_REQ
apm start with vif:0
------beacon_int_set:100 TU
update_ongoing_1_bcn_update
hal_machw_enter_monitor_mode
msh />len:136
len:260
len:166
len:173
len:225
len:136
len:270
len:260
len:166
len:270
```

```
len:173
len:136
len:225
len:260
len:225
wifi_demo monitor stop
msh />
```

9.TLS/SSL

9.1.TLS/SSL简介

SSL(Secure Sockets Layer 安全套接字协议),及其继任者传输层安全（Transport Layer Security，TLS）是为网络通信提供安全及数据完整性的一种安全协议。TLS与SSL在传输层与应用层之间对网络连接进行加密。
mbedtls 库提供了一组可单独使用和编译的加密组件，还可以使用单个配置头文件加入或排除这些组件。

从功能角度来看，该mbedtls分为三个主要部分：

1. SSL/TLS 协议实现。
2. 一个加密库。
3. 一个 X.509 证书处理库。

9.2 TLS/SSL Related API

TLS/SSl接口相关接口参考beken378\func\mbedtls\mbedtls_ui\sl_tls.h，应用程序可通过以下APIs来使用，相关接口如下所示：

函数	描述
ssl_create()	创建一个TLS/SSL连接句柄
ssl_txdat_sender ()	发送数据
ssl_read_data()	获取接收到的数据
ssl_close()	关闭TLS/TLS连接句柄

9.2.1创建一个TLS/SSL连接

在网络可达时，使用者需传入有效的url或IP地址字符串，目标端口，即可以创建一个TLS/SSL句柄。通过如下函数完成：

```
MbedTLSSession * ssl_create(char *url,char *port)
```

参数	描述
url	有效的url或IP地址字符串
port	目标端口字符串
返回	TLS/SSL句柄，该句柄包含该连接的相关信息； 若返回NULL，则创建失败

9.2.2发送数据

```
int ssl_txdat_sender(MbedTLSSession *tls_session,int len,char *data)
```

参数	描述
MbedTLSSession *tls_session	TLS/SSL句柄
int len	待发送的数据长度
char *data	待发送的数据地址
返回	大于0，发送的字节数； 小于0，连接异常

9.2.3获取接收到的数据

```
int ssl_read_data(MbedTLSSession *session, unsigned char *msg, unsigned int mlen, unsigned int timeout_ms)
```

参数	描述
MbedTLSSession *tls_session	TLS/SSL句柄
unsigned char *msg	用于存放接收数据的buffer
unsigned int mlen	用于存放接收数据的buffer的大小
unsigned int timeout_ms	接收数据等待的（阻塞的）时间
返回	大于0，接收到的字节数； 小于0，连接异常

9.2.4关闭TLS/TLS连接句柄

```
int ssl_close(MbedTLSSession *session)
```

参数	描述
MbedTLSSession *tls_session	TLS/SSL句柄
返回	成功

9.3 操作说明

本节的示例代码均位于demos\components\tls_demo\src\tls_demo.c，系统默认不打开此功能，使用者需在wlan_cli功能启动后，调用tls_demo.c中的app_demo_init函数，将实例添加到CLI命令行中，通过命令行来测试此功能。

使用者需注意保证此时网络数据包相对于TLS\SSL服务端的网络是可达的。

使用如下CLI命令可以测试相关API：

命令	说明
tls create 192.168.19.101 443	创建一个TLS/SSL，连接”192.168.19.101”的443端口服务端；若创建成功，启动一个thread接收此连接的数据。

tls sender test_send_data_str	向服务端发送数据，“test_send_data_str”为向服务端发送的数据，请注意不要使用带空格的测试数据，免得被cli命令解析成参数。
tls create	创建一个TLS/SSL，连接” www.baidu.com”的443端口服务端；若创建成功，启动一个thread接收此连接的数据。
tls sender	向服务端发送数据，发送的数据见实例代码，为一个GET网页的请求。

代码示例

在 demos\components\tls_demo\src\tls_demo.c 的文件中添加如下代码，已将 TLS/SSL 测试命令添加到 CLI 中。

```
void user_main(void)
{
    extended_app_waiting_for_launch();
    app_demo_init();
}
```

9.4 TLS/SSL认证

mbedTLS 建立安全通信连接需要经过以下几个步骤：

- 初始化 SSL/TLS 上下文
- 建立 SSL/TLS 握手
- 发送、接收数据
- 交互完成，关闭连接

其中，最关键的步骤就是 SSL/TLS 握手 连接的建立，这里需要进行证书校验（当然也可以不）。

TLS/SSL认证方式有不认证、单向认证、双向认证。

单向认证指的是只有一个对象校验对端的证书合法性。

双向认证指的是相互校验，服务器需要校验每个client,client也需要校验服务器。

对此功能可以采用如下配置

beken378\func\mbedtls\mbedtls-port\inc\tls_client.h 中开启相应的宏。

```
#define CFG_USE_CA_CERTIFICATE                  0
#define CFG_USE_CA_CERTIFICATE_VERIFY           0
```

CFG_USE_CA_CERTIFICATE宏设置为1后，TLS/SSL将会加载ca证书，解释性地解析；同时，会设置验证对等证书所需的数据。解析 buf 中一个或多个证书并将其添加到根证书链接列表中。如果可以解析某些证书，则结果是它遇到的失败证书的数量。 如果没有正确完成，则返回第一个错误。其中ca证书存放在

beken378\func\mbedtls\mbedtls-port\src\ tls_certificate.c 中。CFG_USE_CA_CERTIFICATE_VERIFY宏设置为1后，TLS/SSL将设置为MBEDTLS_SSL_VERIFY_REQUIRED证书验证模式；即证书验证失败，不继续通讯。否则，采用MBEDTLS_SSL_VERIFY_NONE证书验证模式，即不验证证书。

9.5其它

9.5.1.动态窗口

根据TLS\SSL使用的情况，增加窗口的buff大小，若窗口buff始终达不到TLS\SSL的设定值MBEDTLS_SSL_MAX_CONTENT_LEN，则此时可以减少heap的使用，但当需要使用比当前更大的窗口且小于MBEDTLS_SSL_MAX_CONTENT_LEN时，则从heap中申请更大的内存。其由tls_config.h 中的MBEDTLS_SSL_DYNAMIC_CONTENT_LEN控制，该文件的具体位置为：beken378\func\mbedtls\mbedtls-port\inc\tls_config.h。默认打开此功能。

9.5.2.tls\ssl调试信息

调试信息可以通过CFG_OUT_PUT_MBEDTLS_DEBUG_INFO来开启，若其被设置为1，则将通过bk_printf函数输出日志。您可以重定义beken378\func\mbedtls\mbedtls-port\src\tls_client.c里的my_debug函数，以获得您感兴趣的日志信息。

10.MQTT

10.1简述

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输协议），是一种基于发布/订阅（publish/subscribe）模式的"轻量级"通讯协议(ISO 标准 ISO/IEC PRF 20922)，该协议构建于TCP/IP协议上，由IBM在1999年发布。MQTT最大优点在于，可以以极少的代码和有限的带宽，为连接远程设备提供实时可靠的消息服务。作为一种低开销、低带宽占用的即时通讯协议，使其在物联网、小型设备、移动应用等方面有较广泛的应用。

MQTT是一个基于客户端-服务器的消息发布/订阅传输协议。MQTT协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。

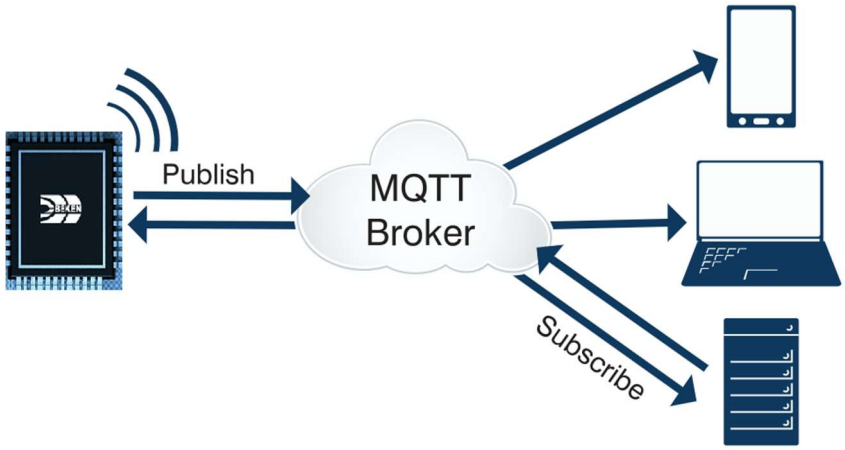

图 7-1 MQTT 的应用

MQTT有三种消息发布服务质量：

"至多一次"，消息发布完全依赖底层TCP/IP网络。会发生消息丢失或重复。这一级别可用于如下情况，环境传感器数据，丢失一次读记录无所谓，因为不久后还会有第二次发送。这一种方式主要普通APP的推送，倘若你的智能设备在消息推送时未联网，推送过去没收到，再次联网也就收不到了。

"至少一次"，确保消息到达，但消息重复可能会发生。

"只有一次"，确保消息到达一次。在一些要求比较严格的计费系统中，可以使用此级别。在计费系统中，消息重复或丢失会导致不正确的结果。这种最高质量的消息发布服务还可以用于即时通讯类的APP的推送，确保用户收到且只会收到一次。

10.2 MQTT客户端

一个使用MQTT协议的应用程序或者设备，它总是建立到服务器的网络连接。客户端可以：

（1）发布其他客户端可能会订阅的信息；

（2）订阅其它客户端发布的消息；
（3）退订或删除应用程序的消息；
（4）断开与服务器连接。

10.2 MQTT Related API

MQTT接口相关接口参考beken378\func\paho-mqtt\mqtt_ui\mqtt_client_core.h，应用程序可通过以下APIs来使用，相关接口如下所示：

函数	描述
mqtt_client_session_init ()	初始化mqtt上下文结构体
tcp_mqtt_client_api_register ()	注册MQTT TCP接口
mqtt_net_connect ()	创建MQTT 网络连接
matt_client_connect ()	MQTT连接
mqtt_client_session_deinit()	摧毁MQTT上下文结构体
mqtt_client_publish()	发布消息
mqtt_client_disconnect()	断开MQTT连接
mqtt_net_disconnect()	断开网络连接

10.2.1初始化mqtt上下文结构体

在创建MQTT之前需先定义一个mqtt_client_session，然后对其初始化。通过如下函数完成：

```
int mqtt_client_session_init(mqtt_client_session* cs)
```

参数	描述
mqtt_client_session* cs	待初始化mqtt上下文结构体
返回	初始化结果

10.2.2注册MQTT TCP接口

```
int tcp_mqtt_client_api_register(tmqtt_client_netport *np)
```

参数	描述
tmqtt_client_netport *np	网络回调接口函数
返回	返回注册结果

10.2.3创建MQTT 网络连接

```
int mqtt_net_connect(mqtt_client_session* cs,char *host,int port)
```

参数	描述
mqtt_client_session* cs	mqtt上下文结构体
char *host	有效的url或IP地址字符串
int port	服务端端口号
返回	返回结果

10.2.4 MQTT连接

```
int matt_client_connect(mqtt_client_session* cs, MQTTPacket_connectData* options)
```

参数	描述
mqtt_client_session* cs	mqtt上下文结构体
MQTTPacket_connectData* options	连接参数
返回	返回连接结果

10.2.5摧毁MQTT上下文结构体

```
int mqtt_client_session_deinit(mqtt_client_session* cs)
```

参数	描述
mqtt_client_session* cs	待摧毁MQTT上下文结构体
返回	成功

10.2.6发布消息

```
int mqtt_client_publish(mqtt_client_session *client, enum QoS qos, const char *topic, const char *msg_str)
```

参数	描述
mqtt_client_session *client	mqtt上下文结构体
enum QoS qos	消息发布服务质量
const char *topic	发布的主题
const char *msg_str	消息内容
返回	返回发布结果

10.2.7断开MQTT连接

```
int mqtt_client_disconnect(mqtt_client_session* cs)
```

参数	描述
mqtt_client_session *client	mqtt上下文结构体
返回	返回注册结果

10.2.8断开网络连接

```
int mqtt_net_disconnect(mqtt_client_session* cs)
```

参数	描述
mqtt_client_session *client	mqtt上下文结构体
返回	返回注册结果

10.3 操作说明

本节的示例代码位于demos\components\mqtt_demo\src\mqtt_demo.c,系统默认不打开此功能，使用者需在wlan_cli功能启动后，调用mqtt_demo.c中的mqtt_app_demo_init函数，将实例添加到CLI命令行中，通过命令行来测试此功能。此文件默认不参与编译，因此请将修改application.mk文件，具体添加方式参考代码示例。

使用者需注意保证此时网络数据包相对于MQTT服务端的网络是可达的。

使用如下CLI命令可以测试相关API：

命令	说明
MQTT con	创建一个MQTT连接，并定时发布消息，然后断开此MQTT连接
MQTT con 192.168.19.39	创建一个MQTT连接，并定时发布消息，然后断开此MQTT连接；其目标服务器的地址是192.168.19.39，端口为1883.

代码示例

1.在demos\components\mqtt_demo\src\mqtt_demo.c的文件中添加如下代码，已将MQTT测试命令添加到CLI中。

```
void user_main(void)
{
    extended_app_waiting_for_launch();
    mqtt_app_demo_init();
```

```
}
```

2.在.\ application.mk 里面增加 SRC_C += ./demos/components/mqtt_demo/src/mqtt_demo.c 使得 mqtt_demo.c 参与编译。

```
ifeq ("${CFG_MBEDTLS}", "1")
SRC_C += ./demos/components/tls_demo/src/tls_demo.c
endif
SRC_C += ./demos/components/mqtt_demo/src/mqtt_demo.c
```

11.TCP\IP

BK72xx SDK的TCP/IP 协议栈是LWIP，lwIP支持BSD Sockets API的所有常见用法。

11.1 BSD Sockets API

BSD套接字API是一个通用的跨平台TCP/IP套接字API，起源于UNIX的伯克利标准发行版，但现在已是POSIX规范的一部分。BSD套接字有时被称为POSIX套接字或Berkeley套接字。

BSD Sockets API的参考资料：
https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html

11.2 支持的BSD Sockets API

支持以下BSD Sockets API函数。详细信息请参见beken378\func\lwip_intf\lwip-2.0.2\src\include\lwip\sockets.h。
BSD Sockets API:

函数	描述
accept()	https://pubs.opengroup.org/onlinepubs/007908799/xns/accept.html
bind()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
shutdown()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
socket()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
getpeername()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
getsockopt()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
setsockopt()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
connect()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
listen()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
close()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
read()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
recv()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
recvfrom()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
write()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
writev()	
send()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
sendmsg()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
sendto()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
select()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
fcntl()	https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html
ioctl()	

closesocket()
ioctlsocket()

11.3 示例

本示例使用CLI命令行控制，需定义宏TCP_CLIENT_DEMO或宏CFG_TCP_SERVER_TEST，使得相关代码参与编译。

定义并使能宏CFG_TCP_SERVER_TEST，其相关代码实现如beken378\func\lwip_intf\tcp_server.c所示，将创建一个端口号20000的、绑定IP为INADDR_ANY的TCP server。其CLI命令为tcp_server。

定义并使能宏TCP_CLIENT_DEMO，其相关代码实现如demos\net\tcp_client\tcp_client_demo.c所示。其CLI命令为tcp_cont ip port，连接一个指定端口号和IP的TCP连接。

CLI命令行

tcp_server\r\n	创建TCP Server
tcp_cont ip port\r\n	创建一个TCP Client

建议在beken378\app\config下的sys_config_bk7231.h、sys_config_bk7231n.h、sys_config_bk7231u.h、sys_config_bk7236.h、sys_config_bk7251.h或sys_config_bk7271.h定义宏，其确保宏的作用域。

注：请保证网络可达。

12. 低功耗

12.1 低功耗简介

BK72xx低功耗模式包括有连接睡眠和无连接睡眠，有连接睡眠有MCU睡眠，MAC睡眠，无连接睡眠有低压睡眠和深度睡眠。另外有ble睡眠和rf睡眠默认已经启动不需要API。低压睡眠和深度睡眠的区别是，深度睡眠唤醒会重启，低压睡眠可以保持内存值。

12.2 低功耗 Related API

低功耗相关接口参考\beken378\func\include\wlan_ui_pub.h 和manual_ps_pub.h，相关接口如下：

函数	描述
bk_wlan_dtim_rf_ps_mode_enable ()	MAC睡眠启用
bk_wlan_dtim_rf_ps_mode_disable ()	MAC睡眠停止
bk_wlan_mcu_ps_mode_enable ()	MCU睡眠启用
bk_wlan_mcu_ps_mode_disable ()	MCU睡眠停止

bk_wlan_instant_lowvol_sleep ()	进入低压睡眠
bk_enter_deep_sleep_mode()	进入深度睡眠

12.2.1 进入有连接低功耗模式

有连接低功耗模式指设备处于STA模式，和AP保持DTIM1连接的WIFI低功耗。可以分为3个等级：

Level 0：mcu,mac睡眠都不启用

Level 1：只有mac睡眠启用

Level 2：mcu,mac睡眠都启用

12.2.1.1 MAC睡眠启用

int bk_wlan_dtim_rf_ps_mode_enable (void);

参数	描述
无	
返回	RT_EOK(0)：成功；其他：出错

12.2.1.2 MAC睡眠停止

int bk_wlan_dtim_rf_ps_mode_disable (void);

参数	描述
无	
返回	RT_EOK(0)：成功；其他：出错

12.2.1.3 MCU睡眠启用

int bk_wlan_mcu_ps_mode_enable (void);

参数	描述
无	
返回	RT_EOK(0)：成功；其他：出错

12.2.1.4 MCU睡眠停止

int bk_wlan_mcu_ps_mode_disable (void);

参数	描述

无	
返回	RT_EOK(0)：成功；其他：出错

12.2.2 低压睡眠

立即进入低压睡眠的函数如下所示：

```
void bk_wlan_instant_lowvol_sleep (PS_DEEP_CTRL_PARAM *lowvol_param);
```

参数	描述
PS_DEEP_CTRL_PARAM *lowvol_param	进入低压睡眠的参数设置
返回	空

参数类型	
PS_DEEP_CTRL_PARAM：	
PS_DEEP_WAKEUP_WAY lowvol_param	唤醒模式的枚举类型
UINT32 gpio_index_map	每个bit位对应gpio0-gpio31，0：不设置； 1：相应的gpio可以唤醒睡眠。
UINT32 gpio_edge_map	每个bit位对应gpio0-gpio31唤醒模式, 0：上升沿唤醒； 1：下降沿唤醒。
UINT32 gpio_last_index_map	低8位bit位对应gpio32-gpio39, 0：不设置； 1：相应的gpio可以唤醒睡眠。
UINT32 gpio_last_edge_map	低8位bit位对应gpio32-gpio39唤醒模式, 0：上升沿醒； 1：下降沿唤醒。
UINT32 gpio_stay_lo_map	每个bit位对应gpio0-gpio31，设置为1将在sleep中保持gpio状态不变，为0将被设置为高阻模式。
UINT32 gpio_stay_hi_map	低8位bit位对应gpio32-gpio39，设置为1将在sleep中保持gpio状态不变，为0将被设置为高阻模式。
UINT32 lpo_32k_src	睡眠32k时钟源选择，默认为0表示ROSC，不需要修改。 如果需要修改为外部32k时钟源则设置为1。
UINT32 sleep_time	rtc timer唤醒周期

12.2.3 深度睡眠模式

进入深度睡眠模式的函数如下所示：

```
void bk_enter_deep_sleep_mode(PS_DEEP_CTRL_PARAM *deep_param);
```

参数	描述
PS_DEEP_CTRL_PARAM *deep_param	进入deep_sleep的参数设置
返回	空

参数类型	
PS_DEEP_CTRL_PARAM：	
PS_DEEP_WAKEUP_WAY lowvol_param	唤醒模式的枚举类型
UINT32 gpio_index_map	每个bit位对应gpio0-gpio31，0：不设置； 1：相应的gpio可以唤醒睡眠。
UINT32 gpio_edge_map	每个bit位对应gpio0-gpio31唤醒模式, 0：上升沿唤醒； 1：下降沿唤醒。
UINT32 gpio_last_index_map	低8位bit位对应gpio32-gpio39, 0：不设置； 1：相应的gpio可以唤醒睡眠。
UINT32 gpio_last_edge_map	低8位bit位对应gpio32-gpio39唤醒模式, 0：上升沿醒； 1：下降沿唤醒。
UINT32 gpio_stay_lo_map	每个bit位对应gpio0-gpio31，设置为1将在sleep中保持gpio状态不变，为0将被设置为高阻模式。
UINT32 gpio_stay_hi_map	低8位bit位对应gpio32-gpio39，设置为1将在sleep中保持gpio状态不变，为0将被设置为高阻模式。
UINT32 lpo_32k_src	睡眠32k时钟源选择，默认为0表示ROSC，不需要修改。 如果需要修改为外部32k时钟源则设置为1。
UINT32 sleep_time	rtc timer唤醒周期

12.3 低功耗示例代码

低功耗的代码示例参考wlan_cli.c中的测试命令，示例代码如下：

14.3.1 有连接睡眠示例代码

```
/*
* 命令格式：首先sta连接：sta wifiname password  连接网络。然后：
            开启level 1 ： ps rfdtim 1
                           ps rf_timer 1
            退出level 1 ： ps rfdtim 0
                           ps rf_timer 0
            开启level 2 ： ps rfdtim 1
                           ps rf_timer 1
                           ps mcudtim 1
            退出level 2 ： ps rfdtim 0
                           ps rf_timer 0
                           ps mcudtim 0
```

```
   命令代码如下
*/
…
#if CFG_USE_MCU_PS
      else if(0 == os_strcmp(argv[1], "mcudtim"))
      {
            if(argc != 3)
            {
                  goto IDLE_CMD_ERR;
            }

            dtim = os_strtoul(argv[2], NULL, 10);
            if(dtim == 1)
            {
                  bk_wlan_mcu_ps_mode_enable();
            }
            else if(dtim == 0)
            {
                  bk_wlan_mcu_ps_mode_disable();
            }
            else
            {
                  goto IDLE_CMD_ERR;
            }
      }
#endif
#if CFG_USE_STA_PS
      else if(0 == os_strcmp(argv[1], "rfdtim"))
      {
            if(argc != 3)
            {
                  goto IDLE_CMD_ERR;
            }

            dtim = os_strtoul(argv[2], NULL, 10);
            if(dtim == 1)
            {
                  if (bk_wlan_dtim_rf_ps_mode_enable())
```

```
                os_printf("dtim enable failed\r\n");
        }
        else if(dtim == 0)
        {
            if (bk_wlan_dtim_rf_ps_mode_disable())
                os_printf("dtim disable failed\r\n");
        }
        else
        {
            goto IDLE_CMD_ERR;
        }
    }
    …
```

14.3.2无连接睡眠示例代码

- wake_up_way枚举型说明

```
typedef enum {
    PS_DEEP_WAKEUP_GPIO = 1,
    PS_DEEP_WAKEUP_RTC = 2,
    PS_DEEP_WAKEUP_USB = 4,
} PS_DEEP_WAKEUP_WAY;
```

并且这三种唤醒方式可以组合使用。

```
/*
* 程序清单： 以下是一个低压睡眠和深度睡眠的函数以及测试命令

* 可以使用命令测试：
* 例如测试deep sleep命令为：deep_sleep  1c  1c  0  0  5  1  0  0  ，设置为gpio唤醒，并且gpio2，3，4可以用下降沿唤醒。
* 同理：如果测试低压睡眠的命令为： lowvol_sleep  1c  c  0  0  5  3  0  0  ，设置为gpio和rtc都可以唤醒，并且gpio2，3可以用下降沿唤醒，gpio4可以用上升沿唤醒，rtc在5秒后会唤醒。
*
* wake_up_way选择唤醒模式，可参考结构体类型说明。

* 程序功能： 实现低功耗和deep_sleep睡眠命令
*/
void lowvol_Sleep_Command(char *pcWriteBuffer, int xWriteBufferLen, int argc, char **argv)
```

```
{
	PS_DEEP_CTRL_PARAM deep_sleep_param;

	deep_sleep_param.gpio_index_map = os_strtoul(argv[1], NULL, 16);
	deep_sleep_param.gpio_edge_map = os_strtoul(argv[2], NULL, 16);
	deep_sleep_param.gpio_last_index_map = 0;
	deep_sleep_param.gpio_last_edge_map = 0;
	deep_sleep_param.sleep_time = os_strtoul(argv[3], NULL, 16);
	deep_sleep_param.wake_up_way = os_strtoul(argv[4], NULL, 16);

	if(argc == 5)
	{
		os_printf("---lowvol sleep test param : 0x%0X 0x%0X %d s    %d\r\n",
				deep_sleep_param.gpio_index_map,
				deep_sleep_param.gpio_edge_map,
				deep_sleep_param.sleep_time,
				deep_sleep_param.wake_up_way);

		bk_wlan_instant_lowvol_sleep(&deep_sleep_param);

		os_printf("wake by %d\r\n",bk_misc_wakeup_get_gpio_num());
	}
	else
	{
		os_printf("---argc error!!! \r\n");
	}
}


static void Deep_Sleep_Command(char *pcWriteBuffer, int xWriteBufferLen, int argc, char **argv)
{
	PS_DEEP_CTRL_PARAM deep_sleep_param;

	deep_sleep_param.wake_up_way		= 0;

	deep_sleep_param.gpio_index_map		= os_strtoul(argv[1], NULL, 16);
	deep_sleep_param.gpio_edge_map		= os_strtoul(argv[2], NULL, 16);
	deep_sleep_param.gpio_last_index_map	= os_strtoul(argv[3], NULL, 16);
	deep_sleep_param.gpio_last_edge_map	= os_strtoul(argv[4], NULL, 16);
```

```
    deep_sleep_param.sleep_time                 = os_strtoul(argv[5], NULL, 16);
    deep_sleep_param.wake_up_way                = os_strtoul(argv[6], NULL, 16);
    deep_sleep_param.gpio_stay_lo_map           = os_strtoul(argv[7], NULL, 16);
    deep_sleep_param.gpio_stay_hi_map           = os_strtoul(argv[8], NULL, 16);

    if(argc == 9)
    {
        os_printf("---deep sleep test param : 0x%0X 0x%0X 0x%0X 0x%0X %d %d\r\n",
                        deep_sleep_param.gpio_index_map,
                        deep_sleep_param.gpio_edge_map,
                        deep_sleep_param.gpio_last_index_map,
                        deep_sleep_param.gpio_last_edge_map,
                        deep_sleep_param.sleep_time,
                        deep_sleep_param.wake_up_way);

    #if (CFG_SOC_NAME != SOC_BK7271)
        bk_enter_deep_sleep_mode(&deep_sleep_param);
    #endif
    }
    else
    {
        os_printf("---argc error!!! \r\n");
    }
}
```

12.4 操作说明

12.4.1 连接万用表

测量低功耗模式下的电流，需要将电源接到vbat引脚上以及串联万用表，如图：

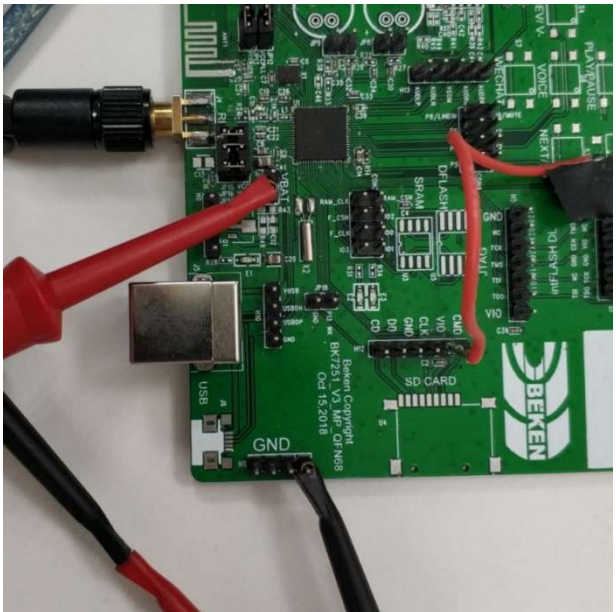

图12.4.1-1

12.4.2 运行现象

• mcu睡眠，mac睡眠示例

设备上电后，输入命令：sta wifiname password 连接网络，输入命令 ps rfdtim 1/0，ps mcudtim 1/0，可以看到电流表上显示芯片的电流值会发生变化。

• 深度睡眠模式

设备上电后，输入命令：deep_sleep 1c 1c 0 0 5 1 0 0 进入深度睡眠，电流可以达到8uA左右。

• 低压睡眠模式

设备上电后，输入命令：lowvol_sleep 1c 1c 0 0 5 1 0 0 进入低压睡眠，电流可以达到100uA左右。

13 BLE

13.1 BLE简介

BLE部分有4.2和5.x两套API，其中7231u、7251使用4.2的API，7231N使用5.x的API。BLE API分为无连接、有连接、其他三类，无连接包含advertising、scanning、initiating三类，有连接包含ble连接成功后，更改mtu size，连接参数等接口，其他包含ble ps功能打开关闭，设置事件回调，新建服务等功能

13.2 BLE Related API

BLE 4.2相关接口参考\beken378\driver\include\ble_api.h和\beken378\driver\ble\ble.h，BLE 5.x相关接口参考\beken378\driver\include\ble_api_5_x.h，相关接口如下：

BLE 4.2 API:

BLE 5.x API:

函数	描述
ble_set_notice_cb()	设置事件回调
bk_ble_create_db()	创建service
app_ble_get_idle_actv_idx_handle()	获取一个未使用句柄
bk_ble_create_advertising()	创建一个广播

bk_ble_set_adv_data()	设置广播数据
bk_ble_set_scan_rsp_data()	设置scan response数据
bk_ble_start_advertising()	开始广播
bk_ble_stop_advertising()	停止广播
bk_ble_delete_advertising()	删除广播
bk_ble_update_param()	更新连接参数
bk_ble_disconnect()	断开连接
bk_ble_gatt_mtu_change()	更新mtu size
bk_ble_get_conn_mtu()	获取mtu size
bk_ble_create_scan()	创建scan
bk_ble_start_scan()	开始scan
bk_ble_stop_scan()	停止scan
bk_ble_delete_scan()	删除scan

13.2.1 BLE 4.2 API

13.2.2 BLE 5.x API

13.2.2.1 无连接

13.2.2.1.1 Adv事件

创建广播

```
ble_err_t bk_ble_create_advertising(uint8_t actv_idx, unsigned char chnl_map, uint32_t intv_min, uint32_t intv_max, ble_cmd_cb_t callback);
```

参数	描述
actv_idx	广播事件的句柄(通过app_ble_get_idle_actv_idx_handle接口获得)
chnl_map	广播事件的channel map(bit 0,1,2分别代表37,38,39channel)
intv_min	最小广播间隔(单位0.625ms)
intv_max	最大广播间隔(单位0.625ms)
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

设置广播数据：

```
ble_err_t bk_ble_set_adv_data (uint8_t actv_idx, unsigned char *adv_buff, unsigned char adv_len, ble_cmd_cb_t callback);
```

参数	描述
actv_idx	广播事件的句柄
adv_buff	广播数据
adv_len	广播数据长度
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

设置scan response数据：

```
ble_err_t bk_ble_set_scan_rsp_data (uint8_t actv_idx, unsigned char *scan_buff, unsigned char scan_len, ble_cmd_cb_t callback);
```

参数	描述
actv_idx	广播事件的句柄
adv_buff	scan response数据
adv_len	scan response数据长度
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

开始广播

```
ble_err_t bk_ble_start_advertising(uint8_t actv_idx, uint16_t duration, ble_cmd_cb_t callback);
```

参数	描述
actv_idx	广播事件的句柄
duration	广播持续时间(0代表continue模式)
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

停止广播

```
ble_err_t bk_ble_stop_advertising(uint8_t actv_idx, ble_cmd_cb_t callback);
```

参数	描述
actv_idx	广播事件的句柄
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

删除广播

```
ble_err_t bk_ble_delete_advertising(uint8_t actv_idx, ble_cmd_cb_t callback);
```

参数	描述
actv_idx	广播事件的句柄
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

13.2.2.1.2 Scan事件

创建scan

```
ble_err_t bk_ble_create_scaning (uint8_t actv_idx, ble_cmd_cb_t callback);
```

参数	描述
actv_idx	scan事件的句柄(通过app_ble_get_idle_actv_idx_handle接口获得)
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

开始scan

```
ble_err_t bk_ble_start_scaning (uint8_t actv_idx, uint16_t scan_intv, uint16_t scan_wd,
ble_cmd_cb_t callback);
```

参数	描述
actv_idx	scan事件的句柄
scan_intv	scan interval(单位0.625ms)
scan_wd	scan window(单位0.625ms)
callback	命令执行成功的回调

返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

停止scan

```
ble_err_t bk_ble_stop_scaning (uint8_t actv_idx, ble_cmd_cb_t callback)；
```

参数	描述
actv_idx	scan事件的句柄
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

删除scan

```
ble_err_t bk_ble_delete_scaning (uint8_t actv_idx, ble_cmd_cb_t callback)；
```

参数	描述
actv_idx	scan事件的句柄
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

13.2.2.1.3 Init事件

获取一个链接句柄

```
uint8_t app_ble_get_idle_conn_idx_handle(void)；
```

参数	描述
返回	小于BLE_CONNECTION_MAX的为句柄，否则失败

注：该接口分配一个句柄号给用户，用于区分不同的连接，从连接将不在使用这个句柄号

主连接初始化

```
ble_err_t bk_ble_create_init(uint8_t con_idx,
                             unsigned short con_interval,
                             unsigned short con_latency,
```

```
unsigned short sup_to,
ble_cmd_cb_t callback);
```

参数	描述
con_idx,	连接句柄
con_interval	BLE 连接 interval参数
con_latency	BLE 连接 latency参数
sup_to	BLE 连接超时参数
callback	命令执行成功的回调
返回	ERR_SUCCESS接口调用成功，其他失败

注：此接口通常只需要调用一次即可

配置主连接句柄的待连接对方的地址和地址类型

```
ble_err_t bk_ble_init_set_connect_dev_addr(unsigned char connidx,struct bd_addr
*bdaddr,unsigned char addr_type);
```

参数	描述
connidx	连接句柄
bdaddr	对端BLE 地址
addr_type	BLE地址类型
返回	ERR_SUCCESS接口调用成功，其他失败

注：设置的参数值将被保存，该函数在bk_ble_init_start_conn调用均可生效

主连接发起连接API

```
ble_err_t bk_ble_init_start_conn(uint8_t con_idx,ble_cmd_cb_t callback);
```

参数	描述
con_idx	连接句柄
callback	命令执行成功的回调
返回	ERR_SUCCESS接口调用成功，其他失败

注：在之前结束之前，不可多次调用

停止当前正在进行连接的API

```
ble_err_t bk_ble_init_stop_conn(uint8_t con_idx,ble_cmd_cb_t callback);
```

参数	描述
con_idx	连接句柄
callback	命令执行成功的回调
返回	ERR_SUCCESS接口调用成功，其他失败

注：调用此接口，停止bk_ble_init_start_conn的连接过程，即此连接句柄未连接上时才能生效。

13.2.2.2 有连接

更新连接参数

```
ble_err_t bk_ble_update_param (uint8_t conn_idx, uint16_t intv_min, uint16_t intv_max, uint15_t latency, uint16_t sup_to, ble_cmd_cb_t callback);
```

参数	描述
conn_idx	连接的句柄(连接成功后，通过注册的notice callback得到)
intv_min	最小连接间隔(单位1.25ms)
intv_max	最大连接间隔(单位1.25ms)
latency	slave latency
sup_to	连接的超时时间(单位10ms)
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

断开连接

```
ble_err_t bk_ble_disconnect (uint8_t conn_idx, ble_cmd_cb_t callback);
```

参数	描述
conn_idx	连接的句柄(连接成功后，通过注册的notice callback得到)
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

更改mtu_size

```
ble_err_t bk_ble_gatt_mtu_change(uint8_t conn_idx, ble_cmd_cb_t callback);
```

参数	描述
conn_idx	连接的句柄(连接成功后，通过注册的notice callback得到)
callback	命令执行成功的回调
返回	ERR_SUCCESS成功，其他失败

注：接口为异步处理，返回值仅代表命令是否下发，具体命令是否执行成功，在callback中获取

获取mtu_size

```
int bk_ble_gatt_mtu_change(uint8_t conn_idx)；
```

参数	描述
conn_idx	连接的句柄(连接成功后，通过注册的notice callback得到)
返回	<0:失败 >0:mtu_size

交换MTU-SIZE

```
ble_err_t app_ble_gatt_mtu_changes(uint8_t conn_idx,uint16_t mtu_size)
```

参数	描述
conn_idx	连接的句柄(连接成功后，通过注册的notice callback得到)
mtu_size	用户自定义的MTU size
返回	<0:失败 >0:mtu_size

ATT-client读操作

```
ble_err_t bk_ble_read_service_data_by_handle_req(uint8_t conidx,uint16_t handle,ble_cmd_cb_t callback)
```

参数	描述
conn_idx	连接的句柄
handle	将要读取的ATT句柄
callback	命令执行成功的回调
返回	ERR_SUCCESS接口调用成功，其他失败

ATT-client写操作

```
ble_err_t bk_ble_write_service_data_req(uint8_t conidx,uint16_t handle,uint16_t data_len,uint8_t *data,ble_cmd_cb_t callback)
```

参数	描述
conn_idx	连接的句柄(连接成功后，通过注册的notice callback得到)
handle	将要写的ATT句柄
data_len	数据长度
data	数据指针

callback	命令执行成功的回调
返回	ERR_SUCCESS接口调用成功，其他失败

注册主连接过滤 Server-uuid

```
void register_app_sdp_service_tab(unsigned char
service_tab_nb,app_sdp_service_uuid *service_tab)
```

参数	描述
service_tab_nb	过滤的server-uuid表的个数
service_tab	过滤的server-uuid表
返回	无

注：对所有主连接均生效

使能注册的主连接过滤 Server-uuid 表

```
void app_sdp_service_filtration(int en)
```

参数	描述
en	0：关闭，1：使能
返回	无

注：对所有主连接均生效

13.2.2.3 其他

设置ble事件回调

```
void ble_set_notice_cb(ble_notice_cb_t func);
```

参数	描述
func	ble事件回调处理函数
返回	无

创建服务

```
ble_err_t bk_ble_create_db(struct bk_ble_db_cfg *ble_db_cfg);
```

参数	描述
ble_db_cfg	ble服务的详细信息
返回	ERR_SUCCESS成功，其他失败

获取未使用句柄

```
uint8_t app_ble_get_idle_actv_idx_handle(void);
```

参数	描述
返回	0xFF:失败

13.3 BLE示例代码

```
#include "app_ble.h"
#include "app_sdp.h"
void ble_notice_cb(ble_notice_t notice, void *param)
{
	switch (notice) {
	case BLE_5_STACK_OK:
		bk_printf("ble stack ok");
		break;
	case BLE_5_WRITE_EVENT:
	{
		write_req_t *w_req = (write_req_t *)param;
		bk_printf("write_cb:conn_idx:%d, prf_id:%d, add_id:%d, len:%d, data[0]:%02x\r\n",
			w_req->conn_idx, w_req->prf_id, w_req->att_idx, w_req->len, w_req->value[0]);
		break;
	}
	case BLE_5_READ_EVENT:
	{
		read_req_t *r_req = (read_req_t *)param;
		bk_printf("read_cb:conn_idx:%d, prf_id:%d, add_id:%d\r\n",
			r_req->conn_idx, r_req->prf_id, r_req->att_idx);
		r_req->value[0] = 0x12;
		r_req->value[1] = 0x34;
		r_req->value[2] = 0x56;
		r_req->length = 3;
		break;
	}
	case BLE_5_REPORT_ADV:
	{
		recv_adv_t *r_ind = (recv_adv_t *)param;
		bk_printf("r_ind:actv_idx:%d, adv_addr:%02x:%02x:%02x:%02x:%02x:%02x\r\n",
			r_ind->actv_idx, r_ind->adv_addr[0], r_ind->adv_addr[1], r_ind->adv_addr[2],
			r_ind->adv_addr[3], r_ind->adv_addr[4], r_ind->adv_addr[5]);
		break;
```

```
        }
        case BLE_5_MTU_CHANGE:
        {
            mtu_change_t *m_ind = (mtu_change_t *)param;
            bk_printf("m_ind:conn_idx:%d, mtu_size:%d\r\n", m_ind->conn_idx, m_ind->mtu_size);
            break;
        }
        case BLE_5_CONNECT_EVENT:
        {
            conn_ind_t *c_ind = (conn_ind_t *)param;
            bk_printf("c_ind:conn_idx:%d, addr_type:%d,
peer_addr:%02x:%02x:%02x:%02x:%02x:%02x\r\n",
                c_ind->conn_idx, c_ind->peer_addr_type, c_ind->peer_addr[0], c_ind->peer_addr[1],
                c_ind->peer_addr[2], c_ind->peer_addr[3], c_ind->peer_addr[4],
c_ind->peer_addr[5]);
            break;
        }
        case BLE_5_DISCONNECT_EVENT:
        {
            discon_ind_t *d_ind = (discon_ind_t *)param;
            bk_printf("d_ind:conn_idx:%d,reason:%d\r\n", d_ind->conn_idx,d_ind->reason);
            break;
        }
        case BLE_5_ATT_INFO_REQ:
        {
            att_info_req_t *a_ind = (att_info_req_t *)param;
            bk_printf("a_ind:conn_idx:%d\r\n", a_ind->conn_idx);
            a_ind->length = 128;
            a_ind->status = ERR_SUCCESS;
            break;
        }
        case BLE_5_CREATE_DB:
        {
            create_db_t *cd_ind = (create_db_t *)param;
            bk_printf("cd_ind:prf_id:%d, status:%d\r\n", cd_ind->prf_id, cd_ind->status);
            break;
        }
        case BLE_5_INIT_CONNECT_EVENT:
        {
```

```
            conn_ind_t *c_ind = (conn_ind_t *)param;
            bk_printf("BLE_5_INIT_CONNECT_EVENT:conn_idx:%d, addr_type:%d,
peer_addr:%02x:%02x:%02x:%02x:%02x:%02x\r\n",
                    c_ind->conn_idx, c_ind->peer_addr_type, c_ind->peer_addr[0], c_ind->peer_addr[1],
                    c_ind->peer_addr[2], c_ind->peer_addr[3], c_ind->peer_addr[4],
c_ind->peer_addr[5]);
            break;
        }
        case BLE_5_INIT_DISCONNECT_EVENT:
        {
            discon_ind_t *d_ind = (discon_ind_t *)param;
            bk_printf("BLE_5_INIT_DISCONNECT_EVENT:conn_idx:%d,reason:%d\r\n",
d_ind->conn_idx,d_ind->reason);
            break;
        }
        case BLE_5_SDP_REGISTER_FAILED:
            bk_printf("BLE_5_SDP_REGISTER_FAILED\r\n");
            break;
        default:
            break;
        }
}

void ble_cmd_cb(ble_cmd_t cmd, ble_cmd_param_t *param)
{
        bk_printf("cmd:%d idx:%d status:%d\r\n", cmd, param->cmd_idx, param->status);
}
#if BLE_SDP_CLIENT
static void ble_app_sdp_characteristic_cb(unsigned char conidx,uint16_t chars_val_hdl,unsigned
char uuid_len,unsigned char *uuid)
{
        bk_printf("[APP]characteristic
conidx:%d,handle:0x%02x(%d),UUID:0x",conidx,chars_val_hdl,chars_val_hdl);
        for(int i = 0; i< uuid_len; i++)
        {
            bk_printf("%02x ",uuid[i]);
        }
        bk_printf("\r\n");
}
```

```
void app_sdp_charac_cb(CHAR_TYPE type,uint8 conidx,uint16_t hdl,uint16_t len,uint8 *data)
{
	bk_printf("[APP]type:%x conidx:%d,handle:0x%02x(%d),len:%d,0x",type,conidx,hdl,hdl,len);
	for(int i = 0; i< len; i++)
	{
		bk_printf("%02x ",data[i]);
	}
	bk_printf("\r\n");
}
static const app_sdp_service_uuid service_tab[] ={
	{
		.uuid_len = 0x02,
		.uuid[0] = 0x00,
		.uuid[1] = 0x18,
	},
	{
		.uuid_len = 0x02,
		.uuid[0] = 0x01,
		.uuid[1] = 0x18,
	},
};

#endif
#define BLE_VSN5_DEFAULT_MASTER_IDX          0

static void ble_command(char *pcWriteBuffer, int xWriteBufferLen, int argc, char **argv)
{
	uint8_t adv_data[31];
	uint8_t actv_idx;

	if (os_strcmp(argv[1], "dut") == 0) {
		ble_dut_start();
	}
	if (os_strcmp(argv[1], "active") == 0) {
		ble_set_notice_cb(ble_notice_cb);
		bk_ble_init();
	}
	if (os_strcmp(argv[1], "create_adv") == 0) {
```

```
        actv_idx = app_ble_get_idle_actv_idx_handle();
        bk_ble_create_advertising(actv_idx, 7, 160, 160, ble_cmd_cb);
    }
    if (os_strcmp(argv[1], "set_adv_data") == 0) {
        adv_data[0] = 0x02;
        adv_data[1] = 0x01;
        adv_data[2] = 0x06;
        adv_data[3] = 0x0A;
        adv_data[4] = 0x09;
        memcpy(&adv_data[5], "Quec_BLE", 9);
        bk_ble_set_adv_data(os_strtoul(argv[2], NULL, 10), adv_data, 0xE, ble_cmd_cb);
    }
    if (os_strcmp(argv[1], "set_rsp_data") == 0) {
        adv_data[0] = 0x0A;
        adv_data[1] = 0x08;
        memcpy(&adv_data[2], "Quec_BLE", 9);
        bk_ble_set_scan_rsp_data(os_strtoul(argv[2], NULL, 10), adv_data, 0xB, ble_cmd_cb);
    }
    if (os_strcmp(argv[1], "start_adv") == 0) {
        bk_ble_start_advertising(os_strtoul(argv[2], NULL, 10), 0, ble_cmd_cb);
    }
    if (os_strcmp(argv[1], "stop_adv") == 0) {
        bk_ble_stop_advertising(os_strtoul(argv[2], NULL, 10), ble_cmd_cb);
    }
    if (os_strcmp(argv[1], "delete_adv") == 0) {
        bk_ble_delete_advertising(os_strtoul(argv[2], NULL, 10), ble_cmd_cb);
    }
    if (os_strcmp(argv[1], "create_scan") == 0) {
        actv_idx = app_ble_get_idle_actv_idx_handle();
        bk_ble_create_scaning(actv_idx, ble_cmd_cb);
    }
    if (os_strcmp(argv[1], "start_scan") == 0) {
        bk_ble_start_scaning(os_strtoul(argv[2], NULL, 10), 100, 30, ble_cmd_cb);
    }
    if (os_strcmp(argv[1], "stop_scan") == 0) {
        bk_ble_stop_scaning(os_strtoul(argv[2], NULL, 10), ble_cmd_cb);
    }
    if (os_strcmp(argv[1], "delete_scan") == 0) {
        bk_ble_delete_scaning(os_strtoul(argv[2], NULL, 10), ble_cmd_cb);
```

```
}
if (os_strcmp(argv[1], "update_conn") == 0) {
        bk_ble_update_param(os_strtoul(argv[2], NULL, 10), 50, 50, 0, 800, ble_cmd_cb);
}
if (os_strcmp(argv[1], "dis_conn") == 0) {
        bk_ble_disconnect(os_strtoul(argv[2], NULL, 10), ble_cmd_cb);
}
if (os_strcmp(argv[1], "mtu_change") == 0) {
        bk_ble_gatt_mtu_change(os_strtoul(argv[2], NULL, 10), ble_cmd_cb);
}else if (os_strcmp(argv[1], "mtu_changes") == 0) {
        bk_ble_gatt_mtu_changes(os_strtoul(argv[2], NULL, 10),128, ble_cmd_cb);
}else if (os_strcmp(argv[1], "get_mtu") == 0) {
        bk_printf("mtu:%d\r\n",bk_ble_get_conn_mtu(os_strtoul(argv[2], NULL, 10)));
}
if (os_strcmp(argv[1], "init_adv") == 0) {
        struct adv_param adv_info;
        adv_info.channel_map = 7;
        adv_info.duration = 0;
        adv_info.interval_min = 160;
        adv_info.interval_max = 160;
        adv_info.advData[0] = 0x02;
        adv_info.advData[1] = 0x01;
        adv_info.advData[2] = 0x06;
        adv_info.advData[3] = 0x0A;
        adv_info.advData[4] = 0x09;
        memcpy(&adv_info.advData[5], "Quec_BLE", 9);
        adv_info.advDataLen = 0xE;
        adv_info.respData[0] = 0x0A;
        adv_info.respData[1] = 0x08;
        memcpy(&adv_info.respData[2], "Quec_BLE", 9);
        adv_info.respDataLen = 0xB;
        actv_idx = app_ble_get_idle_actv_idx_handle();
        bk_ble_adv_start(actv_idx, &adv_info, ble_cmd_cb);
}
if (os_strcmp(argv[1], "deinit_adv") == 0) {
        bk_ble_adv_stop(os_strtoul(argv[2], NULL, 10), ble_cmd_cb);
}
if (os_strcmp(argv[1], "init_scan") == 0) {
        struct scan_param scan_info;
```

```
            scan_info.channel_map = 7;
            scan_info.interval = 100;
            scan_info.window = 30;
            actv_idx = app_ble_get_idle_actv_idx_handle();
            bk_ble_scan_start(actv_idx, &scan_info, ble_cmd_cb);
        }
        if (os_strcmp(argv[1], "deinit_scan") == 0) {
            bk_ble_scan_stop(os_strtoul(argv[2], NULL, 10), ble_cmd_cb);
        }
#if CFG_BLE_INIT_NUM
        if (os_strcmp(argv[1], "con_create") == 0)
        {
            ble_set_notice_cb(ble_notice_cb);
        #if BLE_SDP_CLIENT

        register_app_sdp_service_tab(sizeof(service_tab)/sizeof(app_sdp_service_uuid),service_tab);
            app_sdp_service_filtration(0);
            register_app_sdp_characteristic_callback(ble_app_sdp_characteristic_cb);
            register_app_sdp_charac_callback(app_sdp_charac_cb);
        #endif
            actv_idx = app_ble_get_idle_conn_idx_handle();
            bk_printf("---------->actv_idx:%d\r\n",actv_idx);
            ///actv_idx = BLE_VSN5_DEFAULT_MASTER_IDX;
            ///appm_create_init(actv_idx, 0, 0, 0);
            bk_ble_create_init(actv_idx, 0, 0, 0,ble_cmd_cb);
        }
        else if ((os_strcmp(argv[1], "con_start") == 0) && (argc >= 3))
        {
            struct bd_addr bdaddr;
            unsigned char addr_type = ADDR_PUBLIC;
            int addr_type_str = atoi(argv[3]);
            int actv_idx_str = atoi(argv[4]);
            bk_printf("idx:%d,addr_type:%d\r\n",actv_idx_str,addr_type_str);
            if((addr_type_str > ADDR_RPA_OR_RAND)||(actv_idx_str >= 0xFF)){
                return;
            }
            actv_idx = actv_idx_str;
            hexstr2bin(argv[2], bdaddr.addr, GAP_BD_ADDR_LEN);
            addr_type = addr_type_str;
```

```
            bk_ble_init_set_connect_dev_addr(actv_idx,&bdaddr,addr_type);
            bk_ble_init_start_conn(actv_idx,ble_cmd_cb);
        }
        else if ((os_strcmp(argv[1], "con_stop") == 0) && (argc >= 3))
        {
            int actv_idx_str = atoi(argv[2]);
            bk_printf("idx:%d\r\n",actv_idx_str);
            if(actv_idx_str >= 0xFF){
                return;
            }
            actv_idx = actv_idx_str;
            bk_ble_init_stop_conn(actv_idx,ble_cmd_cb);
        }
        else if ((os_strcmp(argv[1], "con_dis") == 0) && (argc >= 3))
        {
            int actv_idx_str = atoi(argv[2]);
            bk_printf("idx:%d\r\n",actv_idx_str);
            if(actv_idx_str >= 0xFF){
                return;
            }
            actv_idx = actv_idx_str;
            app_ble_master_appm_disconnect(actv_idx);
        }
#if BLE_SDP_CLIENT
        else if (os_strcmp(argv[1], "con_read") == 0)
        {
            if(argc < 4){
                bk_printf("param error\r\n");
                return;
            }
            int actv_idx_str = atoi(argv[3]);
            bk_printf("idx:%d\r\n",actv_idx_str);
            if(actv_idx_str >= 0xFF){
                return;
            }
            actv_idx = actv_idx_str;
            int handle = atoi(argv[2]);
            if(handle >=0 && handle <= 0xFFFF){
                bk_ble_read_service_data_by_handle_req(actv_idx,handle,ble_cmd_cb);
```

```
///appm_read_service_data_by_handle_req(BLE_VSN5_DEFAULT_MASTER_IDX,handle);
        }
        else{
            bk_printf("handle(%x) error\r\n",handle);
        }
}
else if (os_strcmp(argv[1], "con_write") == 0)
{
        if(argc < 4){
            bk_printf("param error\r\n");
            return;
        }
        int handle = atoi(argv[2]);
        int actv_idx_str = atoi(argv[3]);
        bk_printf("idx:%d\r\n",actv_idx_str);
        if(actv_idx_str >= 0xFF){
            return;
        }
        actv_idx = actv_idx_str;
        unsigned char test_buf[4] = {0x01,0x02,0x22,0x32};
        if(handle >=0 && handle <= 0xFFFF){
            bk_ble_write_service_data_req(actv_idx,handle,4,test_buf,ble_cmd_cb);

///appc_write_service_data_req(BLE_VSN5_DEFAULT_MASTER_IDX,handle,4,test_buf);
        }else{
            bk_printf("handle(%x) error\r\n",handle);
        }
}
else if (os_strcmp(argv[1], "con_rd_sv_ntf_int_cfg") == 0)
{
        if(argc < 4){
            bk_printf("param error\r\n");
            return;
        }
        int actv_idx_str = atoi(argv[3]);
        bk_printf("idx:%d\r\n",actv_idx_str);
        if(actv_idx_str >= 0xFF){
            return;
```

```
            }
            actv_idx = actv_idx_str;
            int handle = atoi(argv[2]);
            if(handle >=0 && handle <= 0xFFFF){
                appm_read_service_ntf_ind_cfg_by_handle_req(actv_idx,handle);
            }else{
                bk_printf("handle(%x) error\r\n",handle);
            }
        }
        else if (os_strcmp(argv[1], "con_rd_sv_ud_cfg") == 0)
        {
            if(argc < 4){
                bk_printf("param error\r\n");
                return;
            }
            int actv_idx_str = atoi(argv[3]);
            bk_printf("idx:%d\r\n",actv_idx_str);
            if(actv_idx_str >= 0xFF){
                return;
            }
            actv_idx = actv_idx_str;
            int handle = atoi(argv[2]);
            if(handle >=0 && handle <= 0xFFFF){
                appm_read_service_userDesc_by_handle_req(actv_idx,handle);
            }else{
                bk_printf("handle(%x) error\r\n",handle);
            }
        }
        else if(os_strcmp(argv[1], "svc_filt") == 0)
        {
            if(argc < 3){
                bk_printf("param error\r\n");
                return;
            }
            int en = atoi(argv[2]);
            bk_printf("svc_filt en:%d\r\n",en);
            app_sdp_service_filtration(en);
        }
#endif
```

```
#endif ///CFG_BLE_INIT_NUM
}
```