BEKEN WiFi-SOC FREERTOS SDK API Reference

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

contents

Page 8

1 UART

1.1 Introduction to UART

UART (Universal Asynchronous Receiver/Transmitter) Universal Asynchronous Receiver/Transmitter
As a kind of asynchronous serial communication protocol, UART works on the principle of transferring each character of the data
Transmit bit by bit. It is the most frequently used data bus in the application development process.

The characteristic of UART serial port is to transmit data sequentially, one by one, as long as two transmission lines can be realized.
Now two-way communication, one wire sends data while using another wire to receive data. How many UART serial communication
Two important parameters are baud rate, start bit, data bit, stop bit and parity bit. For two
For a port that uses UART serial communication, these parameters must match, otherwise the communication will not be completed normally.

1.2 UART Related API

The uart interface in Freertos is located in the /beken378/func/user_driver directory, and the related api interface
as follows:

function	describe
bk_uart_initialize ()	uart initialization
bk_uart_send()	uart send data
bk_uart_recv()	uart receive data

bk_uart_set_rx_callback()	Receive callback function
bk_uart_recv_prefetch()	uart receive prefetch
bk_uart_get_length_in_buffer()	Get the current buffer length in uart

1.2.1 Description of uart general structure

Configure the structure of uart parameters:

bk_uart_t:

Structure type	member
BK_UART_1	uart1
BK_UART_2	uart2

bk_uart_config_t:

Structure type	member
data_width	rate
parity	Check Digit
stop_bits	Stop bit
flow_control	Flow Control
flags	Flag bit

uart related interfaces are as follows:

1.2.2 uart initialization

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

parameter	describe
uart	Serial device number
config	Serial port setting structure
optional_rx_buffer	Buffer for serial port operation
return	0: The function is executed successfully
	Non-zero: execution failed

1.2.3 uart send data

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

parameter	describe

uart	Serial device number
data	Data sent by the serial port
size	Serial port send data size
return	0: The function is executed successfully
	Non-zero: execution failed

1.2.4 uart receive data

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

parameter	describe
uart	Serial device number
data	Data received by the serial port
size	Serial port receive data size
return	0: The function is executed successfully
	Non-zero: execution failed

1.2.5 uart receive callback function

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

parameter	describe
uart	Serial device number
callback	Serial port receive callback function
param	Receive parameters
return	0: The function is executed successfully
	Non-zero: execution failed

1.2.6 Get the current cache value of uart

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

parameter	describe
uart	Serial device number

data	The serial port receives the buffer address
size	Receive buffer size
timeout	overtime time
return	0: The function is executed successfully
	Non-zero: execution failed

1.2.7 Get the current buffer length of uart

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

parameter	describe
uart	Serial device number
return	length: current buffer length

1.3 UART sample code

1.3.1 Key instructions

·UART macro definition

#define	BAUD_RATE_115200	115200	Baud rate
#define	DATA_BITS_8	8	Data bit
#define	STOP_BITS_1	1	Stop bit
#define	PARITY_NONE	0	Parity bit
#define	BIT_ORDER_LSB	0	High bit first or low bit first
#define	NRZ_NORMAL	0	model
#define	RT_SERIAL_RB_BUFSZ 64		Receive data buffer size

Set the baud rate:

#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

Set the data bit:

#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

Set stop bit:

#define	STOP_BITS_1	0
#define	STOP_BITS_2	1
#define	STOP_BITS_3	2
#define	STOP_BITS_4	3

Set the parity bit:

#define	PARITY_NONE	0
#define	PARITY_ODD	1
#define	PARITY_EVEN	2

Set high order first:

#define	BIT_ORDER_LSB	0 high order first
#define	BIT_ORDER_MSB	1 High post

Mode selection

#define	NRZ_NORMAL	0 normal mode
#define	NRZ_INVERTED	1 inverted mode

1.3.2 Sample code

```
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 Matters needing attention

The default size of the receive data buffer is 64 bytes. If the number of bytes received in one-time data is too large, it is not timely Read the data, then the data in the buffer will be overwritten by the newly received data, causing data loss. It is recommended Increase the buffer zone.

2 Timer

2.1 Introduction to Timer

BEKEN WiFi Soc has 6 timer timers, among which timer0/1/2 is the clock source of 26M, The clock source of timer3/4/5 is 32K, and the corresponding channel is.

aisle	describe
0	timer0
1	timer1
2	timer2
3	timer3

4	timer4
5	timer5

2.2 Timer Related API

For timer related interfaces, refer to beken378\func\user_driver\BkDriverTimer.h, related interfaces as follows:

function	describe
bk_timer_initialize ()	timer initialization
bk_timer_initialize_us()	The timer is initialized as a timer with a unit of us
bk_get_timer_cnt()	Get the current count value of the timer
bk_timer_stop()	Stop timer

2.2.1 Timer enumeration type description

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

2.2.2 Initialize the timer

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

parameter	describe
timer_id	Selected timer channel: 0 ~ 5
time_ms	Timer setting time
callback	Timer interrupt callback function
return	0: success; -1: error

2.2.3 Timer with initialization unit of us

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

parameter	describe
timer_id	Selected timer channel: 0 ~ 5
time_us	Timer setting time (unit is us)
callback	Timer interrupt callback function
return	0: success; -1: error

2.2.4 Get the current timer count value

```
UINT32 bk_get_timer_cnt(uint8_t timer_id);
```

Page 17

parameter	describe
timer_id	Selected timer channel: 0 ~ 5
return	The currently selected timer count value (its value is the actual count value of the register)

2.2.4 Stop the timer

```
OSStatus bk_timer_stop(uint8_t timer_id)
```

parameter	describe
timer_id	Selected timer channel: 0 ~ 5
return	0: success; -1: error

2.3 Timer sample code

Timer sample code refer to 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();
}
```


Page 18

3 watchdog

3.1 Introduction to Watchdog

The watchdog clock is used to reset the system to prevent the system from running out of order. When the MCU stops running or is powered
The watchdog will also stop running at the time. After the watchdog module is turned on, it is necessary to set up a mechanism to feed the dog regu
Facilitate software debugging and restart problem analysis.

3.2 Watchdog Related API

Watchdog related interfaces refer to beken378\func\user_driver\BkDriverWdg.h, related interfaces
as follows:

function	describe
bk_wdg_initialize ()	Initialize the watchdog
bk_wdg_reload ()	Feed the dog
bk_wdg_finalize ()	End watchdog

3.2.1 Watchdog initialization

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

parameter	describe

timerout	Watchdog restart time, the unit is ms, the maximum value is 0xFFFF, which is about 65s
return	0: success; -1: error

3.2.2 Feeding the dog

```
void bk_wdg_reload( void );
```

parameter	describe
void	null
return	without

3.2.3 End watchdog

```
OSStatus bk_wdg_finalize( void );
```

parameter	describe
void	null
return	without

3.3 Watchdog sample code

The watchdog is relatively simple to use, and no sample code is provided in this version.

4 General SPI

4.1 Introduction to General SPI

BEKEN wifi soc has hardware spi module, the characteristics are as follows:

a) The data exchange length can be configured, usually in bytes, the MSB is sent first, and the LSB is sent later;

b) Support host mode, configurable clock, maximum rate 30MHZ;

c) Support the slave mode, and the maximum rate that can be sustained is 10MHZ;

d) Hour hand polarity (CPOL) and hour hand phase (CPHA) can be configured;

e) Support four-wire full-duplex (MOSI, MISO, CSN, CLK) and three-wire half-duplex (DATA, CS, CLK).

4.2 General SPI Related API

At present, the implementation in FreeRTOS is to transmit data at the time through spi+dma, so the relevant api interfaces are as follows:

function	describe
bk_spi_dma_init ()	Initialize the spi module
bk_spi_dma_transfer ()	spi data transmission (receive/send)

4.2.1 Description of spi structure

spi_message:

send_buf	spi send data buffer
send_len	spi send data length
recv_buf	spi receive data buffer
recv_len	spi received data length

4.2.2 SPI module initialization

Before using the spi interface, you need to configure the spi interface:

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

parameter	describe
mode	Spi working mode setting (see the description below)
rate	spi working frequency
spi_msg	spi transmission data structure
return	RT_EOK(0): success; others: error

4.2.3 SPI send/receive data

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

parameter	describe
mode	Spi working mode setting (see the description below)
spi_msg	spi transmission data structure
return	RT_EOK(0): success; others: error

In master mode, after sending all data, return immediately. In slave mode, it may hang until the The SPI master of the communication initiates the spi sequence, and all data is sent.

4.3 Generic SPI sample code

The sample code refers to \beken378\fun\wlan_ui\bk_peripheral_test.c. Open the macro definition: CFG_SUPPORT_SPI_TEST, enable general spi_dma function test.

4.3.1 Key instructions

·General SPI macro definition

#define	CFG_USE_SPI_DMA	Open spi + dma mode
#define	CFG_USE_SPI_MASTER	Open master
#define	CFG_USE_SPI_SLAVE	Open slave

·SPI working mode:

#define SPI_CPHA	(1<<0)	sck second edge sampling data
#define SPI_CPOL	(1<<1)	sck is at high level when idle
#define SPI_LSB	(0<<2)	0-LSB
#define SPI_MSB	(1<<2)	1-MSB
#define SPI_MASTER	(0<<3)	master mode
#define SPI_SLAVE	(1<<3)	slave mode
#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

Page 22

4.3.2 Sample code

```
/*
* Program list: This is a routine for the use of universal spi dma.
* Command call format: gspi_test slave_dma_rx/slave_dma_tx/master_dma_tx/master_dma_rx len rate
* Program function: Configure spi interface as master/slave, transmission rate rate, complete sending/receiving
*/
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));
```


Page 23


```
                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;
```


Page 26


```
        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) {
```

Page 27


```
                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");
```


Page 28


```
                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 Introduction to PWM

BEKEN WiFi Soc has 6 PWM outputs, and the period and duty cycle of each channel can be individually configured Set, BK7231N channel is as follows.

aisle	describe
0	Corresponding to gpio6 pin
1	Corresponding to gpio7 pin
2	Corresponding to gpio8 pin
3	Corresponding to gpio9 pin
4	Corresponding to gpio24 pin
5	Corresponding to gpio26 pin

Note: This table is BK7231N as an example. The pwm channel corresponding to the GPIO in the wifi series soc needs to be The corresponding wifi soc gpio mapping table shall prevail, please pay attention to check the corresponding chip manual.

5.2 PWM Related API

7231u/7251 pwm common related interface reference

beken378\func\user_driver\BkDriverPwm.h, the related interfaces are as follows:

function	describe
bk_pwm_initialize()	PWM initialization
bk_pwm_start()	Start the PWM function
bk_pwm_stop()	Stop PWM function
bk_pwm_update_param()	Update pwm channel frequency and duty cycle

7231n pwm api is as follows:

function	describe
bk_pwm_cw_initialize()	Initialize a set of mutually exclusive pwm channel parameters
bk_pwm_cw_update_param()	Update the duty cycle and frequency of a set of mutually exclusive pwm channels
bk_pwm_cw_start()	Start pwm mutex channel
bk_pwm_cw_stop()	Stop pwm mutex channel
bk_pwm_initlevl_set_low()	Set the initial level of pwm channel to low level
bk_pwm_initlevl_set_high()	Set the pwm channel initial level to high

5.2.1 pwm enumeration type description

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 Initialize pwm

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

parameter	describe
pwm	Selected pwm channel: 0 ~ 5
cycle	Set the square wave period of pwm
duty_cycle	Set the duty value of pwm
return	0: success; -1: error

5.2.3 Start pwm function

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

parameter	describe
pwm	Selected pwm channel: 0 ~ 5
return	0: success; -1: error

5.2.4 Stop pwm function

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

parameter	describe
pwm	Selected pwm channel: 0~5
return	0: success; -1: error

5.2.5 Adjust pwm parameters

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

parameter	describe
pwm	pwm channel
frequency	pwm frequency

duty_cycle	pwm duty cycle
return	0: success; -1: error

bk7231n pwm api description

5.2.6 Initialize two mutually exclusive pwm channel parameters

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);

parameter	describe
pwm1	pwm mutex channel 1
pwm2	pwm mutex channel 2
frequency	2 mutually exclusive PWM frequencies
duty_cycle1	pwm channel 1 duty cycle
duty_cycle2	Duty cycle of pwm channel 2
dead_band	2 pwm mutually exclusive dead time
return	null

5.2.7 Adjusting mutually exclusive pwm parameters

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);

parameter	describe
pwm	pwm channel
frequency	2 mutually exclusive pwm channel frequencies
duty_cycle1	Mutually exclusive pwm duty cycle
duty_cycle2	pwm second rollover time, the default is generally 0
duty_cycle3	pwm third rollover time, the default is generally 0
return	0: success; -1: error

5.2.8 Start pwm mutual exclusion function

void bk_pwm_cw_start(bk_pwm_t pwm1, bk_pwm_t pwm2);

parameter	describe
pwm1	Selected pwm channel: 0~5
pwm2	Selected pwm channel: 0~5

return	null

5.2.8 Stop pwm mutual exclusion function

void bk_pwm_cw_stop(bk_pwm_t pwm1, bk_pwm_t pwm2);

parameter	describe
pwm1	Selected pwm channel: 0~5
pwm2	Selected pwm channel: 0~5
return	null

5.2.9 Set pwm initial level to low level

OSStatus bk_pwm_initlevl_set_low(bk_pwm_t pwm);

parameter	describe
pwm	Selected pwm channel: 0~5
return	0: success; -1: error

5.2.10 Set pwm initial level to high level

OSStatus bk_pwm_initlevl_set_high(bk_pwm_t pwm);

parameter	describe
pwm	Selected pwm channel: 0~5
return	0: success; -1: error

5.3 PWM sample code

The sample code in freertos refers to the pwm_Command command in bk_peripheral_test.c, serial port Enter the command: pwm single/update/cw channel1 duty_cycle1 cycle (channel2 duty_cycle1 dead_band), the waveform can be detected on the corresponding pin.

```
/*
* Program list: This is a simple example of using PWM
* Command call format: pwm_test single 1 8000 16000
* Program function: Input command can detect the output PWM waveform on the corresponding PWM channel.
*/
```



```
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 Operating instructions

5.4.1 Open configuration

For the bk7231n sample code, refer to the cli command serial port in bk_peripheral_test.c, and open the macro definition: CFG_PERIPHERAL_TEST, after compiling, download the firmware to the device.

5.4.2 Operating Phenomenon

Serial input command: pwm single/update/cw channel1 duty_cycle1 cycle
(Channel2 duty_cycle1 dead_band) The waveform can be detected on the corresponding gpio. Lose separately
Enter the command of channel:1~5, you can use the logic analyzer to see the corresponding gpio output periodic square wave. As shown below

Figure 5.4.2-1: Simultaneous output waveform of 5 pwm channels

5.5 Matters needing attention

·Pwm channel0 may have been used by the cpu as a timer, so its pwm function cannot be used.
·When pwm is outputting, the clock source is 26M.

6 GPIO

6.1 Introduction to GPIO

The pins of BEKEN wifi soc are generally divided into 4 categories: power supply, clock, analog/control and I/O, I/O port In the usage mode, it is divided into General Purpose Input Output (General Purpose Input/Output), referred to as GPIO, and function multiplexing I/O (such as SPI/I2C/UART, etc.).

6.2 GPIO Related API

The gpio api in Freertos is located at beken378\func\user_driver\BkDriverGpio.h related interface
The mouth is as follows:

function	describe
BkGpioInitialize ()	gpio initialization
BkGpioFinalize ()	Terminate the use of gpio
BkGpioOutputHigh ()	gpio output high level
BkGpioOutputLow ()	gpio output low level
BKGpioOp ()	Set gpio mode
BkGpioInputGet ()	Get gpio input value
BkGpioEnableIRQ ()	Enable gpio interrupt
BkGpioDisableIRQ()	Turn off gpio interrupt

6.2.1 gpio enumeration type description

bk_gpio_t: gpio 0-31 pin number

bk_gpio_config_t enumeration type description

INPUT_PULL_UP	Drop-down input mode
INPUT_PULL_DOWN	Pull up input mode
INPUT_NORMAL	Normal input mode (neither pull-up nor pull-down)
OUTPUT_NORMAL	Normal output mode
GPIO_SECOND_FNNC	Second function enable mode

bk_gpio_irq_trigger_t enumeration type description

IRQ_TRIGGER_LOW_LEVEL	Low level trigger interrupt
IRQ_TRIGGER_HGIH_LEVEL	High level trigger interrupt
IRQ_TRIGGER_RISING_EDGE	Rising edge trigger interrupt
IRQ_TRIGGER_FALLING_EDGE	Falling edge trigger interrupt

6.2.2 Initialize gpio, set gpio mode

Before using the pin, you need to set the input or output mode, which is completed by the following function:

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

parameter	describe
gpio	gpio pin number
configuration	GPIO pin working mode
return	0: success; -1: error

6.2.3 Terminate the use of gpio

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

parameter	describe
gpio	gpio pin number
return	0: success; -1: error

6.2.4 Set gpio output high level

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

parameter	describe
gpio	gpio pin number
return	0: success; -1: error

6.2.5 Set gpio output low level

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

parameter	describe
gpio	gpio pin number
return	0: success; -1: error

6.2.6 Set gpio mode

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


parameter	describe
cmd	gpio setting command
id	gpio pin
mode	gpio settings mode
return	0: success; -1: error

6.2.7 Get gpio input level

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

parameter	describe
gpio	gpio pin number
return	0: low level; 1: high level

6.2.8 Enable gpio interrupt trigger function

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

parameter	describe
gpio	gpio pin number
trigger	gpio trigger type
handler	Interrupt callback function registered by gpio
return	0: success; -1: error

6.2.9 Stop gpio interrupt trigger function

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

parameter	describe
gpio	gpio pin number
return	0: low level; 1: high level

6.3 GPIO sample code

The sample code in freertos refers to Gpio_op_Command and wlan_cli.c
Gpio_int_Command command;

6.3.1 Key instructions

By sending the command in Gpio_op_Command, the corresponding gpio is in the same mode,
By sending the command in Gpio_int_Command to test the gpio interrupt, you can use different mode triggers.
send.

6.3.2 Sample code

```
/*
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");
}
```


Page 41


```
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 Introduction to Flash

Flash is a storage module that stores data and codes, supports write balance and power-down protection, and ensures the product Have better scalability in later upgrades.

7.2 Flash Related API

Flash related interfaces refer to beken378\func\user_driver\BkDriverFlash.h, related interfaces as follows:

function	describe
bk_flash_erase()	Erase flash
bk_flash_write ()	Write flash
bk_flash_read ()	Read flash
bk_flash_enable_security()	Protect flash

7.2.1 Description of flash enumeration type

bk_partition_t: flash allocation area;

BK_PARTITION_BOOTLOADER	bootloader partition
BK_PARTITION_APPLICATION	Application Partition
BK_PARTITION_OTA	ota partition
BK_PARTITION_RF_FIRMWARE	rf firmware partition
BK_PARTITION_NET_PARAM	Network parameter partition
BK_PARTITION_USR_CONFIG	User configuration partition
BK_PARTITION_MAX	Largest partition

PROTECT_TYPE: flash protection type

FLASH_PROTECT_NONE	Unprotected
FLASH_PROTECT_ALL	Full protection
FLASH_PROTECT_HALF	Semi-protected
FLASH_UNPROTECT_LAST_BLOCK	The last block is not protected

7.2.1 Erase flash

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

parameter	describe
inPartition	Partition location
off_set	Offset address
size	Erase size
return	0: success; others: failure

7.2.2 Write to flash

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

parameter	describe
inPartition	Partition location
off_set	Offset address
inBuffer	Data buffer written
inBufferLength	Erase size
return	0: success; others: failure

7.2.3 Read flash

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

parameter	describe
inPartition	Partition location
off_set	Offset address
inBuffer	Data buffer read
inBufferLength	Erase size
return	0: success; others: failure

7.2.4 Protect/unprotect flash

OSStatus bk_flash_enable_security(PROTECT_TYPE type);

parameter	describe

void	null
return	0: success; others: failure

7.3 Flash sample code

The sample code in freertos refers to the flash_command_test command in wlan_cli.c:

7.3.1 Key instructions

Check the flash by sending the flash erase, flash writing and reading in flash_command_test

Whether the write and read operations are normal.

7.3.2 Sample code

```
/*
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 network interface

8.1 Introduction to Network Interface

The network interface provided by the SDK of BKEN wifi soc to the upper application is used for:

1. Start STATION mode to connect to the specified network.
2. Turn off STATION mode.
3. Start AP mode for other devices to connect.
4. Turn off AP mode.
5. Start the monitoring mode for the upper-level network distribution.
6. Turn off the monitoring mode.
7. Get status, such as connection status, encryption method, currently used channel, etc.
8. Set the status, such as setting the channel, IP address and so on.
9. Start scan and get scan results.

8.2 Network Interface Related API

Network interface related interface reference \beken378\func\include\wlan_ui_pub.h, the application can The network is controlled through the following APIs, and the related interfaces are as follows:

function	describe
bk_wlan_start()	Start the network, including STATION and AP
bk_wlan_start_sta_adv()	Start STATION quick connection
bk_wlan_stop()	Close the network, including STATION and AP
bk_wlan_start_scan()	Start scan
bk_wlan_scan_ap_reg_cb()	Callback function after registering scan
bk_wlan_start_assign_scan()	scan specific network
bk_wlan_start_monitor()	Start listening mode
bk_wlan_stop_monitor()	Turn off monitoring mode
bk_wlan_register_monitor_cb()	Register the listener callback function
bk_wlan_get_ip_status()	Get the current network status

bk_wlan_get_link_status()	Get the current connection status
bk_wlan_get_channel()	Get the current channel
bk_wlan_set_channel()	Set up channel

8.2.1 Start the network

After the upper layer should obtain the ssid and password, the network can be started. Completed by the following function:

```
OSStatus bk_wlan_start(network_InitTypeDef_st *inNetworkInitPara);
```

parameter	describe
inNetworkInitPara	Incoming need configuration information
return	kNoErr: success; others: failure

Parameter Type

network_InitTypeDef_st:

char	wifi_mode	WiFi mode
char	wifi_ssid[33]	SSID of the network that needs to be connected or established
char	wifi_key[64]	Network password required to connect or establish
char	local_ip_addr[16]	Static IP address, valid when DHCP is turned off
char	net_mask[16]	Static subnet mask, effective when DHCP is turned off
char	gateway_ip_addr[16]	Static gateway address, valid when DHCP is turned off
char	dns_server_ip_addr[16]	Static DNS address, valid when DHCP is turned off
char	dhcp_mode	DHCP mode
char	reserved[32]	Reserve
Int	wifi_retry_interval	Reconnection interval, in milliseconds

8.2.2 Start STATION quick connection

```
OSStatus bk_wlan_start_sta_adv(network_InitTypeDef_adv_st *inNetworkInitParaAdv);
```

parameter	describe
inNetworkInitParaAdv	Network parameters that need to be passed in
return	kNoErr: success; others: failure

Parameter Type

network_InitTypeDef_adv_st:

apinfo_adv_t	ap_info	Network information that needs fast connection
char	key[64]	Network password for fast connection
Int	key_len	Network password length
char	local_ip_addr[16]	Static IP address, valid when DHCP is turned off
char	net_mask[16]	Static subnet mask, effective when DHCP is turned off
char	gateway_ip_addr[16]	Static gateway address, valid when DHCP is turned off
char	dns_server_ip_addr[16]	Static DNS address, valid when DHCP is turned off

char	dhcp_mode	DHCP mode
char	reserved[32]	Reserve
int	wifi_retry_interval	Reconnection time, in milliseconds

apinfo_adv_st:

char	ssid[32]	Network information that needs fast connection
char	bssid[6]	Network password for fast connection
uint8_t	channel	Network password length
wlan_sec_type_t	local_ip_addr[16]	Static IP address, valid when DHCP is turned off
		typedef uint8_t wlan_sec_type_t

8.2.3 Turn off the network

int bk_wlan_stop(char mode);

parameter	describe
mode	The mode that needs to be closed, see the description of mode in the enumeration type
return	kNoErr: success; others: failure

8.2.4 Start scan

void bk_wlan_start_scan(void);

parameter	describe
void	without
return	without

8.2.5 Callback function after registering scan

void bk_wlan_scan_ap_reg_cb(FUNC_2PARAM_PTR ind_cb);

parameter	describe
ind_cb	The function to call back after the scan is over. Function definition: typedef void (*FUNC_2PARAM_PTR)(void *arg, uint8_t vif_idx);
return	without

8.2.6 scan specific network

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

parameter	describe
ssid_ary	Specify the SSID of the network
ssid_num	Specify the number of networks
return	without

8.2.7 Start listening mode

int bk_wlan_start_monitor(void);

parameter	describe
void	without
return	kNoErr: success; others: failure

8.2.8 Turn off monitoring mode

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

parameter	describe
void	without
return	kNoErr: success; others: failure

8.2.9 Register listener callback function

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

parameter	describe
fn	Registered callback function. Function definition: typedef void (*monitor_data_cb_t)(uint8_t *data, int len, hal_wifi_link_info_t *info);
return	without

8.2.10 Get the current network status

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

parameter	describe
outNetpara	Save the acquired network status.
inInterface	Need to get the mode of the network status.

return	kNoErr: success; others: failure

Parameter Type

IPStatusTypedef:

uint8_t	dhcp	Obtained DHCP mode
char	ip[16]	Obtained IP address
char	gate[16]	Obtained gateway IP address
char	mask[16]	Get the subnet mask
char	dns[16]	DNS service IP address
char	mac[16]	Obtained mac address
char	broadcastip[16]	Obtained broadcast IP address

```
#define WiFi_Interface wlanInterfaceTypedef
typedef enum
{
    SOFT_AP,                    /*AP mode*/
    STATION                     /*STATION mode*/
} wlanInterfaceTypedef;
```

8.2.11 Get the current connection status

```
OSStatus bk_wlan_get_link_status(LinkStatusTypeDef *outStatus);
```

parameter	describe
outStatus	Save the obtained connection status. Refer to the description of the structure for details.
return	kNoErr: success; others: failure

Parameter Type

LinkStatusTypeDef:

msg_sta_states	conn_state	Current connection status
int	wifi_strength	Current signal strength
uint8_t	ssid[32]	SSID of the current network
uint8_t	bssid[6]	BSSID of the current network
int	channel	Channel of current network
wlan_sec_type_t	security	Encryption method of the current network Typedef uint8_t wlan_sec_type_t

```
typedef enum {
    MSG_IDLE = 0,                       /*No connection status*/
```



```
    MSG_CONNECTING,                     /*Connecting*/
    MSG_PASSWD_WRONG,                   /*wrong password*/
    MSG_NO_AP_FOUND,                    /*The network to connect to was not found*/
    MSG_CONN_FAIL,                      /*Connection failed*/
    MSG_CONN_SUCCESS,                   /*connection succeeded*/
    MSG_GOT_IP,                         /*Get IP*/
}msg_sta_states;
```

8.2.12 Get the current channel

```
int bk_wlan_get_channel(void);
```

parameter	describe
void	without
return	channel

8.2.13 Set channel

```
int bk_wlan_set_channel(int channel);
```

parameter	describe
channel	Incoming channel value
return	0: success; others: failure

8.3 Examples of network interface usage

8.3.1 Key instructions

·DHCP macro definition description

```
#define DHCP_DISABLE (0)                    /*DHCP is off*/
#define DHCP_CLIENT  (1)                    /*DHCP client mode*/
#define DHCP_SERVER  (2)                    /* DHCP server mode*/
```

8.3.2 Code Example

Start a STATION connection:

```
void demo_sta_app_init(char *oob_ssid,char *connect_key)
{
    /*Define a structure for passing in parameters*/
    network_InitTypeDef_st wNetConfig;
```

51

Page 52


```
    int len;
    /* leave this structure empty*/
    os_memset(&wNetConfig, 0x0, sizeof(network_InitTypeDef_st));

    /*Check the length of the SSID, it cannot exceed 32 bytes*/
    len = os_strlen(oob_ssid);
    if(SSID_MAX_LEN <len)
    {
        bk_printf("ssid name more than 32 Bytes\r\n");
        return;
    }

    /*Pass the SSID and password into the structure*/
     os_strcpy((char *)wNetConfig.wifi_ssid, oob_ssid);
     os_strcpy((char *)wNetConfig.wifi_key, connect_key);

    /*Currently in client mode*/
     wNetConfig.wifi_mode = STATION;
    /* Obtained by DHCP CLIENT, and dynamically obtain an IP address from the router*/
     wNetConfig.dhcp_mode = DHCP_CLIENT;
     wNetConfig.wifi_retry_interval = 100;

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

Start AP mode and provide other client connections:

```
void demo_softap_app_init(char *ap_ssid,char *ap_key)
{
    /*Define a structure for passing in parameters*/
     network_InitTypeDef_adv_st wNetConfigAdv;
      int len;
    /* leave the structure empty*/
     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");
```

52

Page 53

```
        return;
    }
    /*Pass in the ap ssid and ap key to be connected*/
    os_strcpy((char *)wNetConfig.wifi_ssid, ap_ssid);
    os_strcpy((char *)wNetConfig.wifi_key, ap_key);

    /*Currently in ap mode*/
    wNetConfig.wifi_mode = SOFT_AP;
    /*Using the DHCP SERVER mode, the static address needs to be assigned as a local address*/
    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);
    /*Start ap*/
    bk_wlan_start(&wNetConfig);}
```

Start the quick connection of STATION:

```
void demo_sta_adv_app_init(char *oob_ssid,char *connect_key)
{
    /*Define a structure for passing in parameters*/
    network_InitTypeDef_adv_st wNetConfigAdv;
    /* leave the structure empty*/
    os_memset( &wNetConfigAdv, 0x0, sizeof(network_InitTypeDef_adv_st) );
    /*Incoming the SSID to be connected*/
    os_strcpy((char*)wNetConfigAdv.ap_info.ssid, oob_ssid);
    /*Pass in the bssid of the network to be connected, the following bssid is for reference only*/
    hwaddr_aton("12:34:56:00:00:01", wNetConfigAdv.ap_info.bssid);
    /*The encryption method to connect to the network. Refer to the structure description for specific parameters. */
    wNetConfigAdv.ap_info.security = SECURITY_TYPE_WPA2_MIXED;
    /*The channel of the network to be connected*/
    wNetConfigAdv.ap_info.channel = 11;
    /*The network password to be connected and the length of the password*/
    os_strcpy((char*)wNetConfigAdv.key, connect_key);
    wNetConfigAdv.key_len = os_strlen(connect_key);
    /*Get network information such as IP address through DHCP*/
    wNetConfigAdv.dhcp_mode = DHCP_CLIENT;
```



```
    wNetConfigAdv.wifi_retry_interval = 100;
    /*Start quick connection*/
    bk_wlan_start_sta_adv(&wNetConfigAdv);
}
```

Start scan and analyze the results of scan:

```
/*Callback function, used to parse the scan result after the scan ends*/
static void scan_cb(void *ctxt, uint8_t param)
{
```

```
/*Pointer to scan result*/
struct scanu_rst_upload *scan_rst;
/*Save the structure of the analysis result*/
ScanResult apList;
int i;

apList.ApList = NULL;
/*Start scan*/
scan_rst = sr_get_scan_results();
/*If nothing has been scanned, return; otherwise, record the number of networks scanned*/
if( scan_rst == NULL)
{
        apList.ApNum = 0;
        return;
}
else
{
        apList.ApNum = scan_rst->scanu_num;
}
if( apList.ApNum> 0)
{
        /*Apply for the corresponding memory to save the results of the scan*/
        apList.ApList = (void *)os_malloc(sizeof(*apList.ApList) * apList.ApNum);
        for( i = 0; i <scan_rst->scanu_num; i++)
        {
                /*Record the scanned network ssid and rssi*/
                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;
}
 /*Print the result of scan*/
 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");
```

```
    /*Release the requested memory after the end*/
    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)
{
    /*Register scan callback function*/
     mhdr_scanu_reg_cb(scan_cb, 0);
```



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

After the connection is successful, get the connected network status

```
void demo_ip_app_init(void)
{
    /*Define a structure used to save the network state*/
    IPStatusTypedef ipStatus;
    /*Empty the structure*/
     os_memset(&ipStatus, 0x0, sizeof(IPStatusTypedef));
    /*Get the network status and save it in this structure*/
     bk_wlan_get_ip_status(&ipStatus, STATION);
    /*Print the obtained network status*/
     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));
}
```

After the connection is successful, get the connection status:

```
void demo_state_app_init(void)
{
    /*Define the structure to save the connection state*/
     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

    /*Connection status in STATION mode*/
```

```
if( sta_ip_is_start())
{
/*Empty the structure used to save the state*/
 os_memset(&linkStatus, 0x0, sizeof(LinkStatusTypeDef));
/*Get connection status*/
 bk_wlan_get_link_status(&linkStatus);
```



```
/*Print connection status*/
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;
        }
}
/*Connection status in AP mode*/
if( uap_ip_is_start())
{
/*Empty the structure used to save the connection state*/
 os_memset(&ap_info, 0x0, sizeof(network_InitTypeDef_ap_st));
/*Get connection status*/
 bk_wlan_ap_para_info_get(&ap_info);
/*Print out the obtained connection status value*/
os_memcpy(ssid, ap_info.wifi_ssid, 32);
```

Page 58


```
        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 callback function*/
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
         /*
```


Page 59


```
         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
}
/* Program list: This is a simple network interface program using routines
  * Command call format: wifi_demo sta oob_ssid connect_key
  * Program function: Enter related commands to start the network, connect to the network, and so on.
*/
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];
```


Page 61


```
        }
        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 Operating instructions

The sample codes in this section are located in \beken378\demo\ieee802_11_demo.c, and the system defaults to
After turning on this function, after the device is powered on, you can run different programs by inputting the corresponding commands in the debu

8.4.1 Start STATION connection

After the device is powered on, input wifi_demo sta your_ssid your_key in the debug serial port, and the device starts to connect router.

```
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 Start STATION quick connection

After the device is powered on, enter wifi_demo adv your_ssid your_key in the debugging serial port, and the device starts Connect the router, the device log is as follows:

```
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
```

Page 63


```
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 Get status in STATION mode

·Get network status

Connect to the router, refer to section 16.4.1 for the method, and then enter wifi_demo status net in the serial port to obtain The current network status of the device, the device log is as follows:

Figure 5.4.3-1

·Get connection status

Connect to the router, refer to section 16.4.1, then enter wifi_demo status link in the serial port to get The current connection status of the device, the device log is as follows:

Figure 5.4.3-2

8.4.4 Start AP

After the device is powered on, enter wifi_demo softap beken 12345678 in the debugging serial port, and the device starts Connect the router, the device log is as follows:

Page 64

Figure 5.4.4-1

8.4.5 Access status in AP mode

·Get network status

Connect to the router, refer to section 16.4.4, and then enter wifi_demo status net in the serial port to get
The current network status of the device, the device log is as follows:

Figure 5.4.5-1

·Get connection status

Connect to the router, refer to section 16.4.4, and then enter wifi_demo status link in the serial port to get
The current connection status of the device, the device log is as follows:

Figure 5.4.5-2

8.4.6 Start SCAN

·Scan WIFI hotspots

After the device is powered on, enter wifi_demo scan in the debugging serial port, and the device starts to scan for nearby WIFI hotspots.
The equipment log is as follows:

Figure 5.4.6-1

·Scan designated WIFI hotspots

After the device is powered on, enter wifi_demo scan Bekencorp-WIFI in the debugging serial port, and the device starts to scan
Tracing Bekencorp-WIFI, the device log is as follows:

Figure 5.4.6-2

8.4.7 Start promiscuous packet monitoring

After the device is powered on, enter wifi_demo monitor start in the debugging serial port, and the device starts to monitor mixed packets.
Enter wifi_demo monitor stop to stop monitoring, the device log is as follows:

```
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_l_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
```


Page 66


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

9.TLS/SSL

9.1. Introduction to TLS/SSL

SSL (Secure Sockets Layer), and its successor transport layer security
(Transport Layer Security, TLS) is a kind of security and data integrity for network communication
Security Protocol. TLS and SSL encrypt the network connection between the transport layer and the application layer.
The mbedtls library provides a set of encryption components that can be used and compiled separately, and a single configuration header can also b
Add or exclude these components.

From a functional point of view, the mbedtls is divided into three main parts:

1. SSL/TLS protocol implementation.
2. An encryption library.
3. An X.509 certificate processing library.

9.2 TLS/SSL Related API

TLS/SSl interface related interface refer to beken378\func\mbedtls\mbedtls_ui\sl_tls.h, should
The user program can be used through the following APIs, the related interfaces are as follows:

function	describe
ssl_create()	Create a TLS/SSL connection handle
ssl_txdat_sender ()	send data
ssl_read_data()	Get the received data
ssl_close()	Close the TLS/TLS connection handle

9.2.1 Create a TLS/SSL connection

When the network is reachable, the user needs to input a valid url or IP address string, destination port, and then
To create a TLS/SSL handle. Completed by the following function:

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

parameter	describe
url	A valid URL or IP address string
port	Destination port string
return	TLS/SSL handle, the handle contains information about the connection;
	If it returns NULL, the creation fails

9.2.2 Send data

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

Page 68

parameter	describe
MbedTLSSession *tls_session	TLS/SSL handle
int len	Length of data to be sent
char *data	Data address to be sent
return	Greater than 0, the number of bytes sent; Less than 0, abnormal connection

9.2.3 Get the received data

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

parameter	describe
MbedTLSSession *tls_session	TLS/SSL handle
unsigned char *msg	Buffer used to store received data
unsigned int mlen	The size of the buffer used to store received data
unsigned int timeout_ms	Waiting (blocking) time to receive data
return	Greater than 0, the number of bytes received; Less than 0, abnormal connection

9.2.4 Close the TLS/TLS connection handle

int ssl_close(MbedTLSSession *session)

parameter	describe
MbedTLSSession *tls_session	TLS/SSL handle
return	success

9.3 Operating instructions

The sample codes in this section are located in demos\components\tls_demo\src\tls_demo.c, system
This function is not turned on by default. The user needs to call the tls_demo.c after the wlan_cli function is activated
The app_demo_init function adds an instance to the CLI command line, and tests this function through the command line.

Users need to pay attention to ensure that the network data packets are reachable relative to the TLS\SSL server network at this time.

Use the following CLI commands to test related APIs:

Order	illustrate
tls create 192.168.19.101 443	Create a TLS/SSL, connect to "192.168.19.101" 443 Port server; if the creation is successful, start a thread to receive this Connected data.

Page 69

tls sender test_send_data_str	Send data to the server, "test_send_data_str" is to For the data sent by the server, please be careful not to use tests with spaces Data, so as not to be parsed into parameters by cli commands.
tls create	Create a TLS/SSL, connect to "www.baidu.com" 443

	Port server; if the creation is successful, start a thread to receive this
	Connected data.
tls sender	Send data to the server, see the example code for the sent data, which is one
	A request for a GET page.

Code example

Add the following code to the file in demos\components\tls_demo\src\tls_demo.c,
The TLS/SSL test command has been added to the CLI.

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

9.4 TLS/SSL certification

mbedTLS needs to go through the following steps to establish a secure communication connection:

➢ Initialize SSL/TLS context
➢ Establish SSL/TLS handshake
➢ Send and receive data
➢ The interaction is complete, close the connection

Among them, the most critical step is the establishment of the SSL/TLS handshake connection, where certificate verification is required.
Experience (of course not).
TLS/SSL authentication methods include non-authentication, one-way authentication, and two-way authentication.
One-way authentication means that only one object verifies the validity of the peer's certificate.
Two-way authentication refers to mutual verification, the server needs to verify each client, and the client also needs to verify the service
Device.
The following configuration can be used for this function
Open the corresponding macro in beken378\func\mbedtls\mbedtls-port\inc\tls_client.h.

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

After the CFG_USE_CA_CERTIFICATE macro is set to 1, TLS/SSL will load the CA certificate, explanatory
Analysis; At the same time, the data required to verify the peer certificate will be set. Parse one or more certificates in buf and
Add it to the list of root certificate links. If some certificates can be parsed, the result is the failure it encountered
The number of certificates. If it is not completed correctly, the first error is returned. Where the ca certificate is stored in

beken378\func\mbedtls\mbedtls-port\src\ tls_certificate.c.
After the CFG_USE_CA_CERTIFICATE_VERIFY macro is set to 1, TLS/SSL will be set to
MBEDTLS_SSL_VERIFY_REQUIRED certificate verification mode; that is, the certificate verification fails and communication is not continued
Otherwise, the MBEDTLS_SSL_VERIFY_NONE certificate verification mode is adopted, that is, the certificate is not verified.

9.5 Other

9.5.1. Dynamic windows

According to the usage of TLS\SSL, increase the window's buff size, if the window's buff always fails to reach
The setting value of TLS\SSL MBEDTLS_SSL_MAX_CONTENT_LEN can reduce the use of heap at this time.
But when you need to use a larger window than the current one and smaller than MBEDTLS_SSL_MAX_CONTENT_LEN, then from
Apply for larger memory in the heap. It is determined by the tls_config.h
MBEDTLS_SSL_DYNAMIC_CONTENT_LEN control, the specific location of the file is:
beken378\func\mbedtls\mbedtls-port\inc\tls_config.h. This feature is turned on by default.

9.5.2.tls\ssl debugging information

Debugging information can be opened by CFG_OUT_PUT_MBEDTLS_DEBUG_INFO, if it is set to 1, the log will be output through the bk_printf function. You can redefine The my_debug function in beken378\func\mbedtls\mbedtls-port\src\tls_client.c, To get the log information you are interested in.

10.MQTT

10.1 Brief description

MQTT (Message Queuing Telemetry Transport, Message Queuing Telemetry Transport Protocol), Is a "lightweight" communication protocol (ISO standard) based on the publish/subscribe model Quasi-ISO/IEC PRF 20922), this protocol is built on the TCP/IP protocol and was released by IBM in 1999. MQTT The biggest advantage is that it can provide real-time reliability for connecting remote devices with very few codes and limited bandwidth. Messaging service. As a low-overhead, low-bandwidth instant messaging protocol, it is used in the Internet of Things, small It has a wide range of applications in terms of equipment, mobile applications, etc.

MQTT is a client-server based message publish/subscribe transmission protocol. The MQTT protocol is lightweight, Simple, open and easy to implement, these characteristics make it a very wide range of applications. In many cases, the package Including restricted environments, such as: machine-to-machine (M2M) communication and the Internet of Things (IoT).

Figure 7-1 Application of MQTT

MQTT has three types of message publishing service quality:

"At most once", the news release completely relies on the underlying TCP/IP network. Message loss or duplication can occur. This level can be used in the following situations, the environmental sensor data, it does not matter if you lose a reading record, because soon There will be a second sending later. This method mainly pushes ordinary apps. If your smart device is consuming

The information is not connected to the Internet when it is pushed, and the push has not been received in the past, and it will not be received again

"At least once" to ensure that the message arrives, but message duplication may occur.

"Only once" ensures that the message arrives once. In some billing systems with stricter requirements, you can

Use this level. In the billing system, duplication or loss of messages can lead to incorrect results. This highest quality

A large number of message publishing services can also be used to push instant messaging apps to ensure that users receive and only receive

To once.

10.2 MQTT client

An application or device using the MQTT protocol always establishes a network connection to the server.

The client can:

(1) Publish information that other clients may subscribe to;

(2) Subscribe to the news published by other clients;

(3) Unsubscribe or delete the message of the application;

(4) Disconnect from the server.

10.2 MQTT Related API

MQTT interface related interface reference beken378\func\paho-mqtt\mqtt_ui\

mqtt_client_core.h, the application can be used through the following APIs, the related interfaces are as follows:

function	describe
mqtt_client_session_init ()	Initialize the mqtt context structure
tcp_mqtt_client_api_register ()	Register MQTT TCP interface
mqtt_net_connect ()	Create MQTT network connection
matt_client_connect ()	MQTT connection
mqtt_client_session_deinit()	Destroy the MQTT context structure
mqtt_client_publish()	make an announcement
mqtt_client_disconnect()	Disconnect MQTT connection
mqtt_net_disconnect()	Disconnect from the network

10.2.1 Initialize the mqtt context structure

Before creating MQTT, you need to define a mqtt_client_session and then initialize it. Pass

Completed by the following function:

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

parameter	describe
mqtt_client_session* cs	To be initialized mqtt context structure
return	Initialization result

10.2.2 Register MQTT TCP interface

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

parameter	describe
tmqtt_client_netport *np	Network callback interface function
return	Return to registration result

Page 73

10.2.3 Create MQTT network connection

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

parameter	describe
mqtt_client_session* cs	mqtt context structure
char *host	A valid URL or IP address string
int port	Server port number
return	Return result

10.2.4 MQTT connection

int matt_client_connect(mqtt_client_session* cs, MQTTPacket_connectData* options)

parameter	describe
mqtt_client_session* cs	mqtt context structure
MQTTPacket_connectData* options	Connection parameters
return	Return the connection result

10.2.5 Destroy the MQTT context structure

int mqtt_client_session_deinit(mqtt_client_session* cs)

parameter	describe
mqtt_client_session* cs	MQTT context structure to be destroyed
return	success

10.2.6 Publish a message

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

parameter	describe
mqtt_client_session *client	mqtt context structure
enum QoS qos	Message publishing service quality
const char *topic	Published topic
const char *msg_str	Message content
return	Back to publishing results

Page 74

10.2.7 Disconnect MQTT connection

int mqtt_client_disconnect(mqtt_client_session* cs)

parameter	describe
mqtt_client_session *client	mqtt context structure
return	Return to registration result

10.2.8 Disconnect from the network

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

parameter	describe
mqtt_client_session *client	mqtt context structure
return	Return to registration result

10.3 Operating instructions

The sample code in this section is located in demos\components\mqtt_demo\src\mqtt_demo.c,
This function is not turned on by default. The user needs to call the mqtt_demo.c after the wlan_cli function is activated.
mqtt_app_demo_init function, add the instance to the CLI command line, and test this function through the command line
can. This file does not participate in compilation by default, so please modify the application.mk file and add it specifically
Reference code example.

The user needs to pay attention to ensure that the network data packet is reachable relative to the network of the MQTT server at this time.
Use the following CLI commands to test related APIs:

Order	illustrate
MQTT con	Create an MQTT connection, publish messages regularly, and then disconnect this MQTT connection
MQTT con 192.168.19.39	Create an MQTT connection, publish messages regularly, and then disconnect this MQTT connection; the address of the target server is 192.168.19.39, The port is 1883.

Code example

1. Add the following code in the file demos\components\mqtt_demo\src\mqtt_demo.c,
The MQTT test command has been added to the CLI.

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



```
}
```

2. Add SRC_C in .\application.mk
+= ./demos/components/mqtt_demo/src/mqtt_demo.c makes mqtt_demo.c participate
Compile.

```
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

The TCP/IP protocol stack of BK72xx SDK is LWIP, lwIP supports all common BSD Sockets API usage.

11.1 BSD Sockets API

The BSD socket API is a general cross-platform TCP/IP socket API, which originated from Berkeley in UNIX Standard release, but is now part of the POSIX specification. BSD sockets are sometimes called POSIX sockets Word or Berkeley socket.

Reference materials for BSD Sockets API:
https://pubs.opengroup.org/onlinepubs/007908799/xnsix.html

11.2 Supported BSD Sockets API

The following BSD Sockets API functions are supported. For details, see beken378\func\lwip_intf\lwip-2.0.2\src\include\lwip\sockets.h.
BSD Sockets API:

function	describe
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 Example

This example uses CLI command line control, need to define macro TCP_CLIENT_DEMO or macro CFG_TCP_SERVER_TEST makes relevant code participate in compilation.

Define and enable the macro CFG_TCP_SERVER_TEST, and its related code implementation is as follows As shown in beken378\func\lwip_intf\tcp_server.c, a binding IP with port number 20000 will be created It is the TCP server of INADDR_ANY. The CLI command is tcp_server.

Define and enable the macro TCP_CLIENT_DEMO, and its related code is implemented as demos\net\tcp_client\tcp_client_demo.c as shown. The CLI command is tcp_cont ip port, connect Connect a TCP connection with a specified port number and IP.

CLI command line

tcp_server\r\n	Create TCP Server
tcp_cont ip port\r\n	Create a TCP Client

It is recommended to sys_config_bk7231.h under beken378\app\config, sys_config_bk7231n.h, sys_config_bk7231u.h, sys_config_bk7236.h, sys_config_bk7251.h or sys_config_bk7271.h defines a macro, which ensures the scope of the macro.

Note: Please ensure that the network is reachable.

12. Low power consumption

12.1 Introduction to Low Power Consumption

BK72xx low power consumption modes include connected sleep and non-connected sleep, with connected sleep and MCU sleep, MAC sleep, connectionless sleep has low-pressure sleep and deep sleep. In addition, there are ble sleep and rf sleep by default. No API is required upon startup. The difference between low-pressure sleep and deep sleep is that wake-up from deep sleep will restart, and low-pı Sleep can keep the memory value.

12.2 Low Power Related API

Low-power related interface reference \beken378\func\include\wlan_ui_pub.h and manual_ps_pub.h, the related interfaces are as follows:

function	describe
bk_wlan_dtim_rf_ps_mode_enable ()	MAC sleep enabled
bk_wlan_dtim_rf_ps_mode_disable ()	MAC sleep stops
bk_wlan_mcu_ps_mode_enable ()	MCU sleep enable

bk_wlan_mcu_ps_mode_disable ()	MCU sleep stop

bk_wlan_instant_lowvol_sleep ()	Enter low-pressure sleep
bk_enter_deep_sleep_mode()	Enter deep sleep

12.2.1 Enter connected low power consumption mode

Connected low power consumption mode means that the device is in STA mode, and the WIFI low power that keeps DTIM1 connected to the Consumption. It can be divided into 3 levels:

Level 0: mcu and mac sleep are not enabled

Level 1: Only mac sleep is enabled

Level 2: Both mcu and mac sleep are enabled

12.2.1.1 MAC sleep enable

```
int bk_wlan_dtim_rf_ps_mode_enable (void);
```

parameter	describe
without	
return	RT_EOK(0): success; others: error

12.2.1.2 MAC sleep stop

```
int bk_wlan_dtim_rf_ps_mode_disable (void);
```

parameter	describe
without	
return	RT_EOK(0): success; others: error

12.2.1.3 MCU sleep enable

```
int bk_wlan_mcu_ps_mode_enable (void);
```

parameter	describe
without	
return	RT_EOK(0): success; others: error

12.2.1.4 MCU sleep stop

```
int bk_wlan_mcu_ps_mode_disable (void);
```

parameter	describe

without	

return	RT_EOK(0): success; others: error

12.2.2 Low-pressure sleep

The function to enter low-pressure sleep immediately is as follows:

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

parameter	describe
PS_DEEP_CTRL_PARAM *lowvol_param	Parameter settings for entering low-pressure sleep
return	null

Parameter Type	
PS_DEEP_CTRL_PARAM:	
PS_DEEP_WAKEUP_WAY lowvol_param	Enumerated type of wake-up mode
UINT32 gpio_index_map	Each bit corresponds to gpio0-gpio31, 0: not set; 1: The corresponding gpio can wake up to sleep.
UINT32 gpio_edge_map	Each bit corresponds to gpio0-gpio31 wake-up mode, 0: wake up on rising edge; 1: Wake up on falling edge.
UINT32 gpio_last_index_map	The lower 8 bits correspond to gpio32-gpio39, 0: do not set; 1: The corresponding gpio can wake up to sleep.
UINT32 gpio_last_edge_map	The lower 8 bits correspond to gpio32-gpio39 wake-up mode, 0: wake up on rising edge; 1: Wake up on falling edge.
UINT32 gpio_stay_lo_map	Each bit corresponds to gpio0-gpio31, set to 1 will keep in sleep The gpio state remains unchanged, and 0 will be set to high-impedance mode.
UINT32 gpio_stay_hi_map	The lower 8 bits correspond to gpio32-gpio39, set to 1 will be in sleep Keep gpio status unchanged, 0 will be set to high impedance mode.
UINT32 lpo_32k_src	Sleep 32k clock source selection, the default is 0 means ROSC, no need to modify. If you need to modify it to an external 32k clock source, set it to 1.
UINT32 sleep_time	rtc timer wake-up cycle

12.2.3 Deep sleep mode

The function to enter deep sleep mode is as follows:

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

parameter	describe
PS_DEEP_CTRL_PARAM *deep_param	Enter the parameter setting of deep_sleep
return	null

Parameter Type	
PS_DEEP_CTRL_PARAM:	
PS_DEEP_WAKEUP_WAY lowvol_param	Enumerated type of wake-up mode
UINT32 gpio_index_map	Each bit corresponds to gpio0-gpio31, 0: not set; 1: The corresponding gpio can wake up to sleep.
UINT32 gpio_edge_map	Each bit corresponds to gpio0-gpio31 wake-up mode, 0: wake up on rising edge; 1: Wake up on falling edge.
UINT32 gpio_last_index_map	The lower 8 bits correspond to gpio32-gpio39, 0: do not set; 1: The corresponding gpio can wake up to sleep.
UINT32 gpio_last_edge_map	The lower 8 bits correspond to gpio32-gpio39 wake-up mode, 0: wake up on rising edge; 1: Wake up on falling edge.
UINT32 gpio_stay_lo_map	Each bit corresponds to gpio0-gpio31, set to 1 will keep in sleep

	The gpio state remains unchanged, and 0 will be set to high-impedance mode.
UINT32 gpio_stay_hi_map	The lower 8 bits correspond to gpio32-gpio39, set to 1 will be in sleep
	Keep gpio status unchanged, 0 will be set to high impedance mode.
UINT32 lpo_32k_src	Sleep 32k clock source selection, the default is 0 means ROSC, no need to modify.
	If you need to modify it to an external 32k clock source, set it to 1.
UINT32 sleep_time	rtc timer wake-up cycle

12.3 Low-power sample code

Refer to the test command in wlan_cli.c for the low power consumption code example, the sample code is as follows:

14.3.1 Sample code for connected sleep

```
/*
* Command format: first sta connection: sta wifiname password to connect to the network. Then:
        Turn on level 1: ps rfdtim 1
                ps rf_timer 1
        Exit level 1: ps rfdtim 0
                ps rf_timer 0
        Turn on level 2: ps rfdtim 1
                ps rf_timer 1
                ps mcudtim 1
        Exit level 2: ps rfdtim 0
                ps rf_timer 0
                ps mcudtim 0
```



```
    The command code is as follows
*/
…
#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())
```


Page 82


```
                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 No connection sleep sample code

· Wake_up_way enumeration description

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

And these three wake-up methods can be used in combination.

```
/*
* Program list: The following is a function and test command for low-voltage sleep and deep sleep

* You can use the command to test:
* For example, the test deep sleep command is: deep_sleep 1c 1c 0 0 5 1 0 0, set to gpio wake up, and
gpio2, 3, 4 can be awakened by the falling edge.
* The same reason: if the command to test low-voltage sleep is: lowvol_sleep 1c 1c 0 0 5 3 0 0, set to gpio and rtc
Both can wake up, and gpio2, 3 can wake up with a falling edge, gpio4 can wake up with a rising edge, and rtc will wake up after 5 seconds.
*
* Wake_up_way selects the wake-up mode, please refer to the structure type description.

* Program function: Realize low power consumption and deep_sleep sleep command
```

```
*/
void lowvol_Sleep_Command(char *pcWriteBuffer, int xWriteBufferLen, int argc, char **argv)
```


Page 83


```
{
    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 %ds %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);
```


Page 84

```
    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 Operating instructions

12.4.1 Connecting a multimeter

To measure the current in low power consumption mode, you need to connect the power supply to the vbat pin and connect a multimeter in se
picture:

84

Figure 12.4.1-1

12.4.2 Operating Phenomenon

·Mcu sleep, mac sleep example

After the device is powered on, enter the command: sta wifiname password to connect to the network, enter the command `ps rfdtim 1/0, ps mcudtim 1/0` , you can see that the current value of the chip displayed on the ammeter will change.

·Deep sleep mode

After the device is powered on, enter the command: `deepsleep 5 1 4 0` Into deep sleep,

The current can reach about 8uA.

·Low-pressure sleep mode

After the device is powered on, enter the command: low_vol_sleep Into low-pressure sleep,
The current can reach about 100uA.

13 BLE

13.1 Introduction to BLE

The BLE part has two sets of APIs, 4.2 and 5.x, of which 7231u and 7251 use the 4.2 API, and the 7231N uses 5.x API. BLE API is divided into three categories: no connection, connection, and others. No connection includes advertising, There are three types of scanning and initiating. There are connections including ble. After the connection is successful, change the mtu size and cc Interfaces such as parameters, others include functions such as turning on and off the ble ps function, setting event callbacks, and creating new serv

13.2 BLE Related API

BLE 4.2 related interface reference \beken378\driver\include\ble_api.h and \beken378\driver\ble\ble.h, BLE 5.x related interface reference \beken378\driver\include\ble_api_5_x.h, the related interfaces are as follows:

BLE 4.2 API:

BLE 5.x API:

function	describe
ble_set_notice_cb()	Set event callback
bk_ble_create_db()	Create service
app_ble_get_idle_actv_idx_handle()	Get an unused handle
bk_ble_create_advertising()	Create a broadcast

bk_ble_set_adv_data()	Set up broadcast data
bk_ble_set_scan_rsp_data()	Set scan response data
bk_ble_start_advertising()	Start broadcasting
bk_ble_stop_advertising()	Stop broadcasting
bk_ble_delete_advertising()	Delete broadcast
bk_ble_update_param()	Update connection parameters
bk_ble_disconnect()	Disconnect
bk_ble_gatt_mtu_change()	Update mtu size
bk_ble_get_conn_mtu()	Get mtu size
bk_ble_create_scan()	Create scan
bk_ble_start_scan()	Start scan
bk_ble_stop_scan()	Stop scan
bk_ble_delete_scan()	Delete scan

13.2.1 BLE 4.2 API

13.2.2 BLE 5.x API

13.2.2.1 No connection

13.2.2.1.1 Adv event

Create broadcast

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);

parameter	describe
actv_idx	The handle of the broadcast event (via App_ble_get_idle_actv_idx_handle interface obtained)
chnl_map	The channel map of the broadcast event (bit 0, 1, 2 respectively represent 37,38,39channel)
intv_min	Minimum broadcast interval (unit 0.625ms)
intv_max	Maximum broadcast interval (unit 0.625ms)
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma Get in callback

Set broadcast data:

Page 87

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);

parameter	describe
actv_idx	Handle of broadcast event
adv_buff	Broadcast data
adv_len	Broadcast data length
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma Get in callback

Set scan response data:

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);

parameter	describe
actv_idx	Handle of broadcast event
adv_buff	scan response data
adv_len	scan response data length
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma Get in callback

Start broadcasting

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

parameter	describe
actv_idx	Handle of broadcast event
duration	Broadcast duration (0 means continue mode)
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma Get in callback

Stop broadcasting

ble_err_t bk_ble_stop_advertising(uint8_t actv_idx, ble_cmd_cb_t callback);

parameter	describe
actv_idx	Handle of broadcast event
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma
Get in callback

Delete broadcast

ble_err_t bk_ble_delete_advertising(uint8_t actv_idx, ble_cmd_cb_t callback);

parameter	describe
actv_idx	Handle of broadcast event
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma
Get in callback

13.2.2.1.2 Scan event

Create scan

ble_err_t bk_ble_create_scaning (uint8_t actv_idx, ble_cmd_cb_t callback);

parameter	describe
actv_idx	The handle of the scan event (via App_ble_get_idle_actv_idx_handle interface obtained)
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma
Get in callback

Start 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);

parameter	describe
actv_idx	Handle of scan event
scan_intv	scan interval (unit 0.625ms)
scan_wd	scan window (unit 0.625ms)
callback	Callback for successful command execution

return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma
Get in callback

Stop scan

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

parameter	describe
actv_idx	Handle of scan event
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma
Get in callback

Delete scan

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

parameter	describe
actv_idx	Handle of scan event
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma
Get in callback

13.2.2.1.3 Init event

Get a link handle

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

parameter	describe
return	Handle less than BLE_CONNECTION_MAX, otherwise fail

Note: This interface assigns a handle number to the user to distinguish between different connections. The slave connection will no longer use this
Handle number

Primary connection initialization

```
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);
```

parameter	describe
con_idx,	Connection handle
con_interval	BLE connection interval parameter
con_latency	BLE connection latency parameter
sup_to	BLE connection timeout parameters
callback	Callback for successful command execution
return	ERR_SUCCESS interface call is successful, other failures

Note: This interface usually only needs to be called once

Configure the address and address type of the host to be connected to the main connection handle

ble_err_t bk_ble_init_set_connect_dev_addr(unsigned char connidx,struct bd_addr

*bdaddr,unsigned char addr_type);

parameter	describe
connidx	Connection handle
bdaddr	Peer BLE address
addr_type	BLE address type
return	ERR_SUCCESS interface call is successful, other failures

Note: The set parameter value will be saved, and the function can be effective when calling bk_ble_init_start_conn

The main connection initiates the connection API

ble_err_t bk_ble_init_start_conn(uint8_t con_idx,ble_cmd_cb_t callback);

parameter	describe
con_idx	Connection handle
callback	Callback for successful command execution
return	ERR_SUCCESS interface call is successful, other failures

Note: before the previous end, it cannot be called multiple times

Stop the currently connected API

ble_err_t bk_ble_init_stop_conn(uint8_t con_idx,ble_cmd_cb_t callback);

parameter	describe
con_idx	Connection handle
callback	Callback for successful command execution
return	ERR_SUCCESS interface call is successful, other failures

Note: Call this interface to stop the connection process of bk_ble_init_start_conn, that is, the connection handle can only take effect when it is not connected.

13.2.2.2 There is a connection

Update connection parameters

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);

parameter	describe
conn_idx	Connection handle (after the connection is successful, through the registered notice callback)
intv_min	Minimum connection interval (unit 1.25ms)
intv_max	Maximum connection interval (unit 1.25ms)
latency	slave latency
sup_to	Connection timeout time (unit: 10ms)
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma
Get in callback

Disconnect

ble_err_t bk_ble_disconnect (uint8_t conn_idx, ble_cmd_cb_t callback);

parameter	describe
conn_idx	Connection handle (after the connection is successful, through the registered notice callback)
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma Get in callback

Change mtu_size

ble_err_t bk_ble_gatt_mtu_change(uint8_t conn_idx, ble_cmd_cb_t callback);

parameter	describe
conn_idx	Connection handle (after the connection is successful, through the registered notice callback)
callback	Callback for successful command execution
return	ERR_SUCCESS succeeded, others failed

Note: The interface is processed asynchronously, and the return value only represents whether the command is issued, whether the specific comma Get in callback

Get mtu_size

int bk_ble_gatt_mtu_change(uint8_t conn_idx);

parameter	describe
conn_idx	Connection handle (after the connection is successful, through the registered notice callback)
return	<0:Failed>0:mtu_size

Exchange MTU-SIZE

ble_err_t app_ble_gatt_mtu_changes(uint8_t conn_idx,uint16_t mtu_size)

parameter	describe
conn_idx	Connection handle (after the connection is successful, through the registered notice callback)
mtu_size	User-defined MTU size
return	<0:Failed>0:mtu_size

ATT-client read operation

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

parameter	describe
conn_idx	Connection handle
handle	ATT handle to be read
callback	Callback for successful command execution
return	ERR_SUCCESS interface call is successful, other failures

ATT-client write operation

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)

parameter	describe

conn_idx	Connection handle (after the connection is successful, through the registered notice callback)
handle	ATT handle to be written
data_len	Data length
data	Data pointer

callback	Callback for successful command execution
return	ERR_SUCCESS interface call is successful, other failures

Register the main connection filter Server-uuid

void register_app_sdp_service_tab(unsigned char service_tab_nb,app_sdp_service_uuid *service_tab)

parameter	describe
service_tab_nb	Number of filtered server-uuid tables
service_tab	Filtered server-uuid table
return	without

Note: valid for all main connections

Enable registered primary connection filtering Server-uuid table

void app_sdp_service_filtration(int en)

parameter	describe
en	0: disable, 1: enable
return	without

Note: valid for all main connections

13.2.2.3 Other

Set ble event callback

void ble_set_notice_cb(ble_notice_cb_t func);

parameter	describe
func	ble event callback processing function
return	without

Create service

ble_err_t bk_ble_create_db(struct bk_ble_db_cfg *ble_db_cfg);

parameter	describe
ble_db_cfg	ble service details
return	ERR_SUCCESS succeeded, others failed

Get unused handle

uint8_t app_ble_get_idle_actv_idx_handle(void);

parameter	describe
return	0xFF: failure

13.3 BLE sample code

```
#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) {
```


Page 98


```
        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);
```

Page 99


```
}
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;
```


Page 100


```
        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
```

Page 104


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