BK3432-BLE SDK

编程指南

V1.0
2018.12.21

深圳市集贤科技有限公司
T:0755-82571152 F:0755-88373753
http://www.uascent.com
深圳市南山区朗山路同方信息港A栋4楼

目录

1. 版本记录及免责申明和版权公告

1.1 版本记录

Version	Date	Author	Description
V1.0	2018/12/26	Eatun	初始版本

1.2 免责申明和版权公告

本文档中所有信息均按产品现状提供，如有变更，恕不另行通知。

本文档内容不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不对使用本文档内信息产生的任何侵犯专利权的行为负责。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。

文中提到的所有商标均属其各自所有者的财产，特此声明。

注 意

由于产品版本升级或其他原因，本手册内容有可能变更，深圳市集贤科技有限公司保留在没有任何通知或者提示的情况下对本手册的内容进行修改的权利，使用者如需获取最新产品信息，请与本公司申请最终文档。本手册仅作为使用指导，深圳市集贤科技有限公司尽力在本手册中提供最新的信息，但不确保手册内容完全准确。本手册中的所有陈述、信息和建议也不构成任何明示或暗示的担保。

一 驱动源码篇

开发前准备及说明：硬件工具及烧录流程请参考我司相关开发工具及烧录流程。代码编译请使用keil MDK5.12版本。

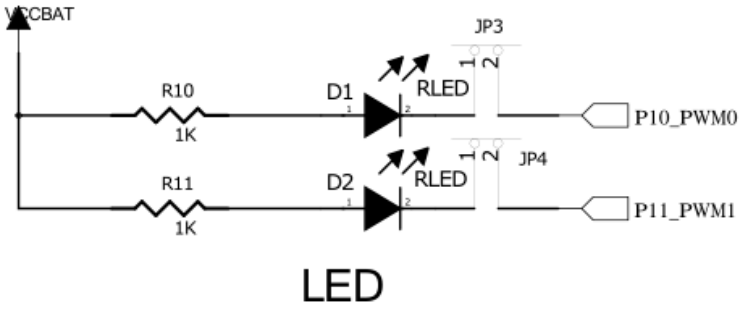
找到“BK3432资料包V1.1\ble_3432_sdk_ext_12_0203_SDK\bk3432_project\ble_app_gatt”这一工程路径，打开工程后，找到arch_main.c文件下的rw_main(void)这个函数，该函数就是工程的main函数入口。

（一）GPIO口输入输出实验

BK3431Q/BK3432内部集成了一颗arm内核的MCU，所以其本身与我们一般所使用的单片机无异，就如“Hellow world”一样，点亮LED毫无疑问一直是嵌入式开发之旅的第一步，下面我们以按键控制LED亮灭为切入点来介绍BK3431Q/BK3432工程模板的使用。

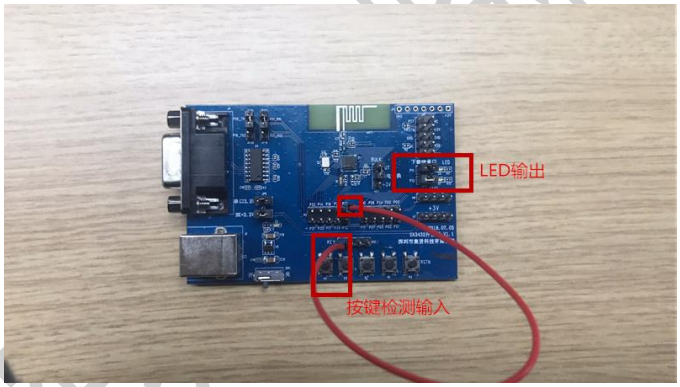
1硬件外设说明

首先查看开发板原理图，如下，我们需要将开发板上对应的LED跳线帽短路，可以看到对应IO为低电平点亮；



LED

同时开发板上需要将PB2口接到按键上，如下图：



3功能解说

本节代码实现对PB2作为IO口检测输入，当PB2输入为低电平时，PB1输出低，PB0输出高。当PB2输入为高电平时，PB1输出高，PB0输出低。

4 源码讲解

4. 1编程思路：

1）初始化 GPIO 输出的引脚；
2）检测PB2的IO口状态；
3）根据PB2的IO口状态对PB1跟PB0进行设置。

4. 1源码赏析：

```
/**************************************************
*作用
io口输入输出功能测试
*参数
无
*返回值
无
*其他说明
本节代码实现对PB2作为io口检测输入，
当PB2输入为低电平时，PB1输出低，PB0输出高。
当PB2输入为高电平时，PB1输出高，PB0输出低。
**************************************************/
void ZB_GPIOTest(void)
{
    uint8_t        Beiley_TestGPIO_Data;
    gpio_config(GPIOB_0, OUTPUT, PULL_NONE);//将PB0 设置为输出状态，无上下拉。
    gpio_config(GPIOB_1, OUTPUT, PULL_NONE);//将PB1 设置为输出状态，无上下拉。
    gpio_config(GPIOB_2, INPUT, PULL_HIGH); //将PB2 设置为输入上拉状态。
    while(1)
    {
        Beiley_TestGPIO_Data = gpio_get_input(GPIOB_2);//读取PB2的io口状态
        if(Beiley_TestGPIO_Data == 0) //如果为低，则将PB1 置低，PB0置高，且串口将PB2的值打印出来。
    {
          UART_PRINTF("intv_min = %d\r\n",Beiley_TestGPIO_Data);
          Delay_ms(10);
            gpio_set(GPIOB_1,0);
      gpio_set(GPIOB_0,1);
        }
    else //如果为高，则将PB1置高，将PB0 置低，且串口将PB2的值打印出来。
    {
            gpio_set(GPIOB_1,1);
      gpio_set(GPIOB_0,0);
            UART_PRINTF("intv_min = %d\r\n",Beiley_TestGPIO_Data);
            Delay_ms(10);
        }
    }
} ? end ZB_GPIOTest ?
```

下载程序

代码编写完成后，在../bk3432_project\ble_app_gatt\output\app目录下找到bk3432_ble_app_merge.bin文件下载到开发板。至此，GPIO口输入输出教程叙述完毕。

（二）PWM应用实验

硬件外设说明

开发板上需要将GPIOB3接到示波器上，在示波器上可以看到有PWM波形输出。

功能解说

本节代码实现将GPIOB3使能为PWM功能。

源码讲解

4. 1编程思路：

1）定义一个结构体存储PWM初始化数据：static PWM_DRV_DESC　ZB_PWMTest;

2）初始化PWM：pwm_init(&ZB_PWMTest);

```
/*************************************************************
*作用          :PWM功能测试
*参数          :无
*返回值        :无
*其他说明      :
周期计算公式=(end_value/时钟频率)*2
*************************************************************/
static PWM_DRV_DESC ZB_PWMTest;//定义一个结构体
void ZB_BLePwmTest(void)
{
    rwip_prevent_sleep_set(BK_DRIVER_TIMER_ACTIVE);
    ZB_PWMTest.channel          =   3;      //选择GPIOB3作为PWM输出
    ZB_PWMTest.mode             =   0x01;   //选择PWM的模式
    ZB_PWMTest.pre_divid        =   0;
    ZB_PWMTest.end_value        =   64;     //修改PWM输出频率
    ZB_PWMTest.duty_cycle       =   64/2;   //修改PWM占空比
    ZB_PWMTest.p_Int_Handler    =   NULL;
    pwm_init(&ZB_PWMTest);
}
```

3）将PWM_DRIVER宏定义设置为1

```
#define UART_DRIVER                     1
#define GPIO_DRIVER                     1
#define AUDIO_DRIVER                    0
#define RTC_DRIVER                      0
#define ADC_DRIVER                      1
#define I2C_DRIVER                      0
#define PWM_DRIVER                      1
```

5）在rw_main()函数里调用ZB_BLePwmTest(void)。

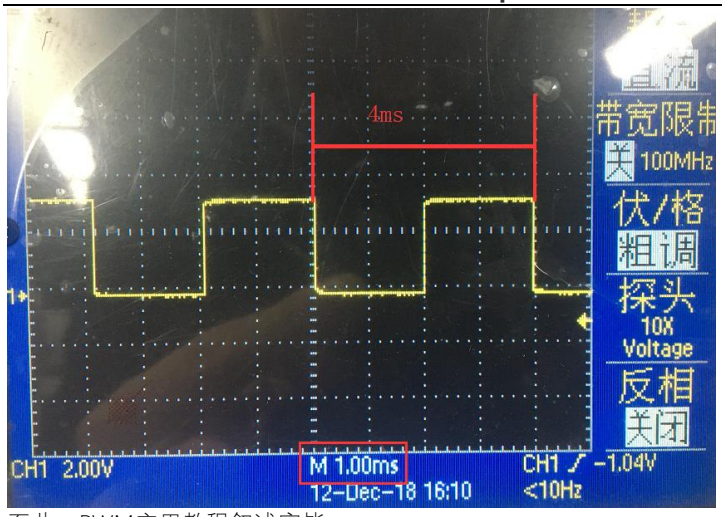
```
GLOBAL_INT_START();// finally start interrupt handling
#if 1 //1.GPIOB_3中断初始化 2.模拟iic初始化 3.PWM初始化4.定时器初始化
gpio_int_init();
iic_init();
ZB_BLePwmTest();
ZB_BLeTimerTest();
#endif
UART_PRINTF("bk3432..\r\n");
#if 1 //正常广播透传程序
gpio_config(GPIOB_0, OUTPUT, PULL_NONE);//初始化GPIO
while(1)
{
    rwip_schedule();//schedule all pending events
    GLOBAL_INT_DISABLE();// Checks for sleep have to be done with interru
```

下载程序

代码编写完成后，在../bk3432_project\ble_app_gatt\output\app目录下找到bk3432_ble_app_merge.bin文件下载到开发板。

实验现象

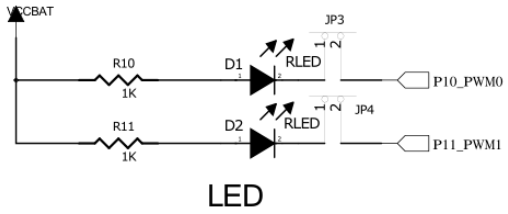
该实验PWM信号输出周期是4ms，占空比是50%。



至此，PWM应用教程叙述完毕。

（三）PWM定时器应用实验

硬件外设说明



LED

功能解说

本节代码实现PWM定时功能。开发板上D1和D2隔1秒循环闪灯并且串口会有信息打印。

源码讲解

4. 1编程思路：

1）定义一个结构体存储PWM初始化数据：PWM_DRV_DESC ZB_TimerTest;

2）初始化PWM：ZB_BLeTimerTest(void)

```
/*************************************************************
*作用          :PWM定时器功能测试
*参数          :无
*返回值        :无
*其他说明      :
周期计算公式=(end_value/时钟频率)
*************************************************************/
PWM_DRV_DESC ZB_TimerTest;//定义结构体
void ZB_BLeTimerTest(void)//每4ms进一次中断
{
    gpio_config(GPIOB_0, OUTPUT, PULL_NONE);
    gpio_config(GPIOB_1, OUTPUT, PULL_NONE);
    rwip_prevent_sleep_set(BK_DRIVER_TIMER_ACTIVE);
    ZB_TimerTest.channel         =   1;          //选择PWM1作为定时器时钟源1
    ZB_TimerTest.mode            =   0x17;       //PWM模式选择
    ZB_TimerTest.pre_divid       =   0;
    ZB_TimerTest.end_value       =   32000;      //设置定时器初值
    ZB_TimerTest.duty_cycle      =   6400/2;
    ZB_TimerTest.p_Int_Handler   =   ZB_TimerIrq;//定时器中断回调函数。
    pwm_init(&ZB_TimerTest);
    //选择时钟16M   注意：所有PWM时钟选择都共用0x00800020寄存器的位1
    REG_AHB0_ICU_PWMCLKCON |= 1<<1;
//    REG_AHB0_ICU_PWMCLKCON &= 1<<1;//清零
    REG_AHB0_ICU_PWMCLKCON |= 8<<12;//2分频
}
```

3）中断回调函数

```
int Zb_TimerFlag;
//   P10,P11    循环闪灯，1s周期
void ZB_TimerIrq(unsigned char ucChannel)
{
    Zb_TimerFlag++;
    if(Zb_TimerFlag == 250)
    {
        gpio_set(GPIOB_0,0);
        gpio_set(GPIOB_1,1);
    }
    else if(Zb_TimerFlag == 500)
    {
        gpio_set(GPIOB_0,1);
        gpio_set(GPIOB_1,0);
        Zb_TimerFlag=0;
    }
}
```

4）将PWM_DRIVER宏定义设置为1

```
#define UART_DRIVER                1
#define GPIO_DRIVER                1
#define AUDIO_DRIVER               0
#define RTC_DRIVER                 0
#define ADC_DRIVER                 1
#define I2C_DRIVER                 0
#define PWM_DRIVER                 1
```

5）查看中断入口函数IRQ_Exception(void)是否有调用PWM中断函数pwm_isr()

```
void IRQ_Exception(void)
 {
     uint32_t IntStat;
     uint32_t irq_status;
     IntStat = intc_status_get();
     #if (SYSTEM_SLEEP) ...
     #endif
     #if (UART_DRIVER)     ...
     #endif
     #if (GPIO_DRIVER) ...
     #endif
     #if (PWM_DRIVER)
     if(IntStat & INT_STATUS_PWM1_bit)
     {
         irq_status |= INT_STATUS_PWM1_bit;
         pwm_isr();
     }
     if(IntStat & INT_STATUS_PWM2_bit) ...
     if(IntStat & INT_STATUS_PWM3_bit) ...
     if(IntStat & INT_STATUS_PWM4_bit) ...
 #endif
 #if (ADC_DRIVER) ...
 #endif
 #if RTC_DRIVER ...
 #endif
     intc_status_clear(irq_status);
 }
```

6）在rw_main()函数里调用ZB_BLeTimerTest(void)。

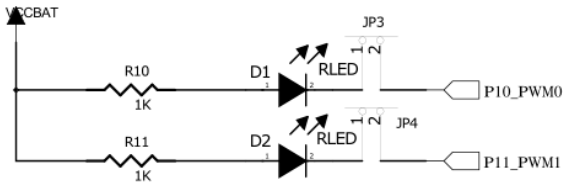
```
    GLOBAL_INT_START();// finally start interrupt handling
    #if 1 //1.GPIOB_3中断初始化 2.模拟iic初始化 3.PWM初始化4.定时器初始化
    gpio_int_init();
    iic_init();
    ZB_BLeTimerTest();
    ZB_BLePWMTest();
    #endif
    UART_PRINTF("bk3432...\r\n");
    #if 1 //正常广播透传程序
    gpio_config(GPIOB_0, OUTPUT, PULL_NONE);//初始化GPIO
    while(1)
    {
        rwip_schedule();//schedule all pending events
        GLOBAL_INT_DISABLE();// Checks for sleep have to be done with inter
        oad_updating_user_section_pro();
        if (!test_cnt)//只会进入一次
```

下载程序

代码编写完成后，在../bk3432_project\ble_app_gatt\output\app目录下找到bk3432_ble_app_merge.bin文件下载到开发板。至此，PWM定时器应用教程叙述完毕。

（四）软件定时器应用实验

硬件外设说明



LED

功能解说

本节代码实现利用蓝牙底层产生的10毫秒为基准的软件定时器来做一个500毫秒的闪灯程序。开发板上D1和D2隔500毫秒循环闪灯并且串口会有信息打印。

源码讲解

4. 1编程思路：

1）定义一个500毫秒的软件定时器函数：Ble_Timer500msTest。在函数里可以添加用户需要处理的各种驱动程序并且必须再次回调软件定时器ke_timer_set

```
/*************************************************************
*作用        :500毫秒软件定时器
*参数        :无
*返回值      :KE_MSG_CONSUMED=0
*其他说明    :
*************************************************************/
static int Ble_Timer500msTest(ke_msg_id_t const msgid,
                              struct fff0s_fff1_level_ntf_cfg_ind const *param,
                              ke_task_id_t const dest_id,
                              ke_task_id_t const src_id)
{
    static  unsigned char Led_sta=0;
    Led_sta=~Led_sta;
    gpio_set(GPIOB_0,Led_sta);
    gpio_set(GPIOB_1,~Led_sta);
    ke_timer_set(APP_TIMER_TEST,dest_id ,50);//50*10ms回调一次
    return (KE_MSG_CONSUMED);
}
```

2）在appm_msg添加一个任务消息

```
/// APP Task messages
enum appm_msg
{
    APPM_DUMMY_MSG = TASK_FIRST_MSG(TASK_ID_APP),
    APP_PARAM_UPDATE_REQ_IND,
    APP_PERIOD_TIMER,
    APP_TIMER_TEST,
    READ_WRITE_NVR_FLASH,
    READ_WRITE_AT24C02,
};
```

3）在appm_default_state[ ]结构体数组里调用函数Ble_Timer500msTest

```
/* Default State handlers definition. */
const struct ke_msg_handler appm_default_state[] =
{
    // Note: first message is latest message checked by kernel so default is put on top.
    {KE_MSG_DEFAULT_HANDLER,        (ke_msg_func_t)appm_msg_handler},
    {GAPM_DEVICE_READY_IND,         (ke_msg_func_t)gapm_device_ready_ind_handler},
    {GAPM_CMP_EVT,                  (ke_msg_func_t)gapm_cmp_evt_handler},
    {GAPC_GET_DEV_INFO_REQ_IND,     (ke_msg_func_t)gapc_get_dev_info_req_ind_handler},
    {GAPC_SET_DEV_INFO_REQ_IND,     (ke_msg_func_t)gapc_set_dev_info_req_ind_handler},
    {GAPC_CONNECTION_REQ_IND,       (ke_msg_func_t)gapc_connection_req_ind_handler},
    {GAPC_CMP_EVT,                  (ke_msg_func_t)gapc_cmp_evt_handler},
    {GAPC_DISCONNECT_IND,           (ke_msg_func_t)gapc_disconnect_ind_handler},
    {GAPM_PROFILE_ADDED_IND,        (ke_msg_func_t)gapm_profile_added_ind_handler},
    {GAPC_LE_PKT_SIZE_IND,          (ke_msg_func_t)gapc_le_pkt_size_ind_handler},
    {GAPC_PARAM_UPDATED_IND,        (ke_msg_func_t)gapc_param_updated_ind_handler},
    {GATTC_MTU_CHANGED_IND,         (ke_msg_func_t)gattc_mtu_changed_ind_handler},
    {GAPC_PARAM_UPDATE_REQ_IND,     (ke_msg_func_t)gapc_param_update_req_ind_handler},
    {APP_PARAM_UPDATE_REQ_IND,      (ke_msg_func_t)gapc_update_conn_param_req_ind_handler},
    {APP_PERIOD_TIMER,              (ke_msg_func_t)app_period_timer_handler},
    {APP_TIMER_TEST,                (ke_msg_func_t)Ble_Timer500msTest   },
    {READ_WRITE_NVR_FLASH,          (ke_msg_func_t)Read_Write_NVR_Flash   },
    {READ_WRITE_AT24C02,            (ke_msg_func_t)Read_Write_AT24C02   },
};
```

4）在rw_main()里调用ke_timer_set(APP_TIMER_TEST,TASK_APP,50)，只需要调用一次并且需要蓝牙的功能设置完成后才可以调用。

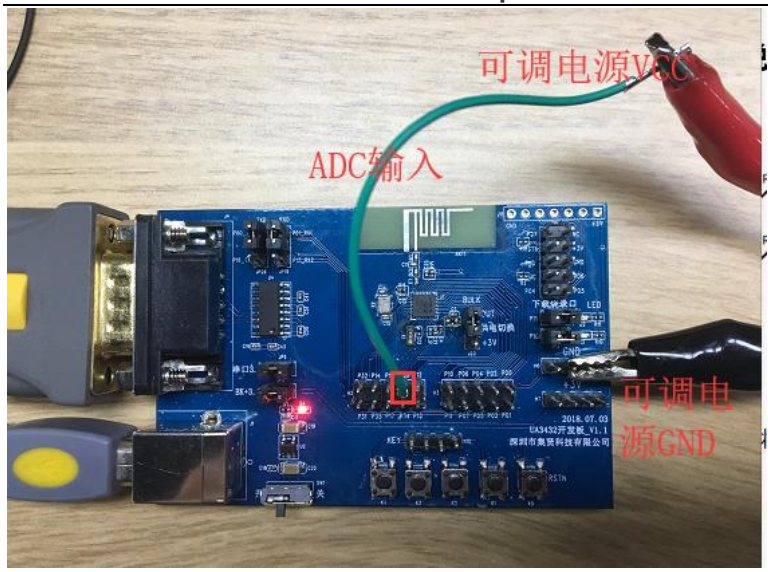
```
UART_PRINTF("bk3432...\r\n");
#if 1 //正常广播透传程序
gpio_config(GPIOB_0, OUTPUT, PULL_NONE);//初始化GPIO
gpio_config(GPIOB_1, OUTPUT, PULL_NONE);//初始化GPIO
while(1)
{
    rwip_schedule();//schedule all pending events
    GLOBAL_INT_DISABLE();// Checks for sleep have to be done with interrupt disabled
    oad_updating_user_section_pro();
    if (!test_cnt)//只进入一次
    {
        ke_timer_set(APP_TIMER_TEST,TASK_APP,50);
        test_cnt = 1;
    }
    if(wdt_disable_flag==1)
    {
        wdt_disable();
    }
    #if 0//SYSTEM_SLEEP// Check if the processor clock can be gated
    sleep_type = rwip_sleep();
    if((sleep_type & RW_MCU_DEEP_SLEEP) == RW_MCU_DEEP_SLEEP)
```

下载程序

代码编写完成后，在../bk3432_project\ble_app_gatt\output\app目录下找到bk3432_ble_app_merge.bin文件下载到开发板。至此，软件定时器应用教程叙述完毕。

（五）ADC应用实验

硬件外设说明



功能解说

本节代码将GPIOC3复用为ADC功能，将此GPIO口的电压值转换成数字量并且串口将打印的数据打印出来。

源码讲解

4. 1编程思路：

1）初始化ADC：void adc_init(uint8_t chanle,uint8_t mode)

```
/*******************************************************************************
*作用      :初始化ADC
*参数      :
*           chanle:选择ADC通道
*           mode   :ADC模式选择 00:休眠模式01:单步模式10:软件控制11:连续模式
*返回值    :
*其他说明  :选择模式时一般选择单步模式
*******************************************************************************/

void adc_init(uint8_t chanle,uint8_t mode)
{
    uint32_t cfg;
    //REG_AHB0_ICU_ANALOG1_PWD |= (0x01 << 9);
    REG_AHB0_ICU_ADCCLKCON &= ~(0x01 << 0);//enable adc clk
    REG_AHB0_ICU_ADCCLKCON = (0x5 << 1);//adc div
    //set special as peripheral func
    gpio_config(GPIOD_0 + chanle,FLOAT,PULL_NONE);
    //set adc mode/channel/wait clk
    cfg  = ( (mode << BIT_ADC_MODE ) | (chanle << BIT_ADC_CHNL) | (0x01 << BIT_ADC_WAIT_CLK_SETTING));
    REG_APB7_ADC_CFG =  cfg;
    //set adc sample rate/pre div
    cfg |= ((18 << BIT_ADC_SAMPLE_RATE) | (3 << BIT_ADC_PRE_DIV)|(0x0 << BIT_ADC_DIV1_MODE));
    REG_APB7_ADC_CFG =  cfg;
    cfg |= (0x0 << BIT_ADC_FILTER);
    REG_APB7_ADC_CFG =  cfg;
    REG_APB7_ADC_CFG |= (0x01 << BIT_ADC_INT_CLEAR);
    REG_APB7_ADC_CFG |= (0x01 << BIT_ADC_EN);
    //不能先使能ADC，不然ADC FIFO满时没有读出再次启动ADC就不会有中断
    REG_AHB0_ICU_INT_ENABLE |= (0x01 << 8);
}
```

2）获取ADC值函数

```
/*****************************************************************
*作用      :获取ADC转换后的数字量
*参数      :无
*返回值    :ADC转换后的数字量
*其他说明 :
*****************************************************************/
uint16_t adc_get_value(void)
{
    uint16_t adc_cnt;
    adc_cnt=0;
    REG_APB7_ADC_CFG |= SET_ADC_EN+(0x01 << BIT_ADC_MODE );
    UART_PRINTF("adc_flag=%d\r\n",adc_flag);
    while (!adc_flag)
    {
        adc_cnt++;
        if(adc_cnt>300)
        {
            break;
        }
        Delay_us(10);
    }
    if(adc_flag==1)//进入ADC中断时 置一
    {
        g_adc_value = (REG_APB7_ADC_DAT & 0xfff);//获取ADC的值
        UART_PRINTF("g_adc_value=%d\r\n",g_adc_value);
        adc_flag =0;
    }
    //ADC值读取完成后必须把使能位清除
    REG_APB7_ADC_CFG &= ~(SET_ADC_EN+(0x03 << BIT_ADC_MODE ));
    return g_adc_value;
}
```

3）将ADC_DRIVER宏定义设置为1

```
//DRIVER CONFIG
#define UART_DRIVER                     1
#define GPIO_DRIVER                     1
#define AUDIO_DRIVER                    0
#define RTC_DRIVER                      0
#define ADC_DRIVER                      1
#define I2C_DRIVER                      1
#define PWM_DRIVER                      1
```

4）查看中断入口函数IRQ_Exception(void)是否有调用PWM中断函数adc_isr()

```
void IRQ_Exception(void)
{
    uint32_t IntStat;
    uint32_t irq_status;
    IntStat = intc_status_get();
    #if (SYSTEM_SLEEP) ...
    #endif
    #if (UART_DRIVER)     ...
    #endif
    #if (GPIO_DRIVER) ...
    #endif
    #if (PWM_DRIVER) ...
    #endif
    #if (ADC_DRIVER)
    if(IntStat & INT_STATUS_ADC_bit)
    {
        irq_status |= INT_STATUS_ADC_bit;
//        UART_PRINTF("adc_isr.=%x\r\n",irq_status);
        adc_isr();
    }
    #endif
#if RTC_DRIVER ...
#endif
    intc_status_clear(irq_status);
}
```

5）利用软件定时器500毫秒定时调用adc_get_value() (有关软件定时器的使用请参考软件定时器应用实验篇)。

```
static int ADC_GetValue(ke_msg_id_t const msgid,
                                        struct fff0s_fff1_level_ntf_cfg_ind const *param,
                                        ke_task_id_t const dest_id,
                                        ke_task_id_t const src_id)
{
    adc_get_value();
    ke_timer_set(ADC_GET_VALUE,TASK_APP,50);
    return (KE_MSG_CONSUMED);
}
```

6）在rw_main()里调用ke_timer_set(ADC_GET_VALUE,TASK_APP,50);只需要调用一次

```
UART_PRINTF("bk3432..\r\n");
#if 1 //正常广播透传程序
gpio_config(GPIOB_0, OUTPUT, PULL_NONE);//初始化GPIO
while(1)
{
    rwip_schedule();//schedule all pending events
    GLOBAL_INT_DISABLE();// Checks for sleep have to be done with interrupt disabled
    oad_updating_user_section_pro();
    if (!test_cnt)//只进入一次
    {
        ke_timer_set(ADC_GET_VALUE,TASK_APP, 50);
        test_cnt = 1;
    }
    if(wdt_disable_flag==1)
    {
        wdt_disable();
    }
    #if 1//SYSTEM_SLEEP// Check if the processor clock can be gated
    sleep_type = rwip_sleep();
      UART_PRINTF("sleep_type=%d \r\n",sleep_type);
    if((sleep_type & RW_MCU_DEEP_SLEEP) == RW_MCU_DEEP_SLEEP)
    {
        if(icu_get_sleep_mode())// 1:idel  0:reduce voltage
```

下载程序

代码编写完成后，在../bk3432_project\ble_app_gatt\output\app目录下找到bk3432_ble_app_merge.bin文件下载到开发板。至此，ADC应用教程叙述完毕。

（六）读写内部Flash NVR区应用实验

硬件外设说明

无

功能解说

本节代码读写操作内部Flash NVR区域。连续往NVR区域写入60个字节0~55 、A3、56和最后两位校验值，然后读出数据在串口打印出来。

源码讲解

4. 1编程思路：

1）初始化flash：void flash_init(void)

```
/*********************************************************************
*作用      :初始化flash
*参数      :无
*返回值    :无
*其他说明  :
*********************************************************************/
void flash_init(void)
{
    // Init flash environment
    flash_env.length[0]     =  FLASH_MAIN_SIZE;
    flash_env.space_type[0]  = FLASH_SPACE_TYPE_MAIN;
    flash_env.length[1]     =  FLASH_NVR_SIZE;
    flash_env.space_type[1]  = FLASH_SPACE_TYPE_NVR;
}
```

2）Flash读写测试函数

```
/*************************************************************
*作用      :读写内部Flash NVR区域
*参数      :无
*返回值    :无
*其他说明  :连续往NVR区域写入60个字节0~55 、A3、56和最后两位校验值
*************************************************************/
uint8_t writedata_one[56];
uint8_t readdata_one[60];
void flash_readwrite_test()//读写flash测试
{
    uint8 i;
    UART_PRINTF("writedata:\r\n");
    Delay_ms(1);
    for(i =0;i<56;i++)//对writedata_one进行赋值
    {
        writedata_one[i]=i;
        UART_PRINTF("%02x ",writedata_one[i]);
        Delay_ms(1);
    }
    UART_PRINTF("\r\n");
    flash_erase_sector(FLASH_SPACE_TYPE_NVR,0x8000);
    //连续往NVR区域写入60个字节0~55 、A3、56和最后两位校验值
    write_lmecc_pointq(&writedata_one[0],&writedata_one[28]);
    Delay_ms(200);
    if(read_lmecc_pointq_status())
    {
        read_lmecc_pointq(&readdata_one[0],&readdata_one[30]);
        UART_PRINTF("readdata\r\n:");
        Delay_ms(1);
        for(i=0;i<60;i++)
        {
            UART_PRINTF("%02x ",readdata_one[i]);
            Delay_ms(1);
        }
        UART_PRINTF("\r\n");
    }
    else
    {
        flash_read(FLASH_SPACE_TYPE_NVR,0x8000,58,readdata_one);
        UART_PRINTF("\r\nelse readdata\r\n:");
        Delay_ms(1);
        for(i=0;i<60;i++)
        {
            UART_PRINTF("%02x ",readdata_one[i]);
            Delay_ms(1);
        }
        UART_PRINTF("\r\n");
    }
}
```

3）利用软件定时器500毫秒定时调用flash_readwrite_test() (有关软件定时器的使用请参考软件定时器应用实验篇)。

```
static int Read_Write_NVR_Flash(ke_msg_id_t const msgid,
                                struct fff0s_fff1_level_ntf_cfg_ind const *param,
                                ke_task_id_t const dest_id,
                                ke_task_id_t const src_id)
{
    flash_readwrite_test();
    ke_timer_set(READ_WRITE_NVR_FLASH,dest_id ,50);
    return (KE_MSG_CONSUMED);
}
```

6）在rw_main()里调用ke_timer_set(ADC_GET_VALUE,TASK_APP,50);只需要调用一次

```
UART_PRINTF("bk3432..\r\n");
#if 1 //正常广播透传程序
gpio_config(GPIOB_0, OUTPUT, PULL_NONE);//初始化GPIO
while(1)
{
    rwip_schedule();//schedule all pending events
    GLOBAL_INT_DISABLE();// Checks for sleep have to be done with interrupt disabled
    oad_updating_user_section_pro();
    if (!test_cnt)//只进入一次
    {
        ke_timer_set(READ_WRITE_NVR_FLASH,TASK_APP,50);//软件定时器调用读写flash
        test_cnt = 1;
        UART_PRINTF("READ_WRITE_NVR_FLASH....\r\n");
    }
    if(wdt_disable_flag==1)
    {
        wdt_disable();
    }
    #if SYSTEM_SLEEP// Check if the processor clock can be gated
    sleep_type = rwip_sleep();
    if((sleep_type & RW_MCU_DEEP_SLEEP) == RW_MCU_DEEP_SLEEP)
    {
```

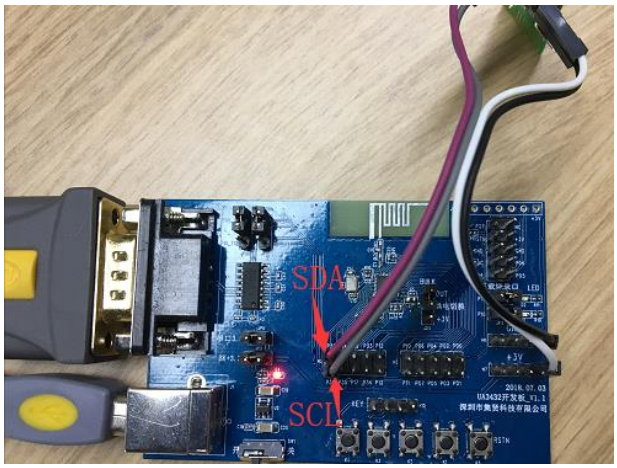
下载程序

代码编写完成后，在../bk3432_project\ble_app_gatt\output\app目录下找到bk3432_ble_app_merge.bin文件

下载到开发板。至此，读写内部Flash NVR区应用教程叙述完毕。

（七）模拟IIC读写AT24C02应用实验

硬件外设说明：用户需要外接AT24C02



功能解说

本节代码使用模拟IIC读写AT24C02的例程，500毫秒读写一次AT24C02，串口会打印出写入和读取出来的数据。

源码讲解

4. 1编程思路：

1）初始化IIC：初始化IIC的SDA和SCL

```
void iic_init()
{
    gpio_config(SCLK,OUTPUT,PULL_NONE);
    gpio_config(SDAT,OUTPUT,PULL_NONE);
}
```

2）AT24C02读写函数

```
uint8_t readbuf[5];
uint8_t writebuf[5]={0,1,2,3,4};
void readwrite_AT24C02()
{
    iic_tx_data(0Xa0, 0X01,&writebuf[1],1);
    UART_PRINTF("writebuf[1] = %x\r\n",writebuf[1]);
    DelayMs(1);
    writebuf[1]=writebuf[1]+1;
    iic_rx_data(0Xa0, 0X01,&readbuf[0],1);
    UART_PRINTF("readbuf[0] = %x\r\n",readbuf[0]);
//    Delay_ms(200);
}
```

3）利用软件定时器500毫秒定时调用readwrite_AT24C02() (有关软件定时器的使用请参考软件定时器应用实验篇)。

```
static int Read_Write_AT24C02(ke_msg_id_t const msgid,
                                                struct fff0s_fff1_level_ntf_cfg_ind const *param,
                                                ke_task_id_t const dest_id,
                                                ke_task_id_t const src_id)

{
    readwrite_AT24C02();
    ke_timer_set(READ_WRITE_AT24C02,dest_id ,50);
    return (KE_MSG_CONSUMED);
}
```

4）在rw_main()里调用ke_timer_set(READ_WRITE_NVR_FLASH,TASK_APP,50)只需要调用一次和调用初始化函数。

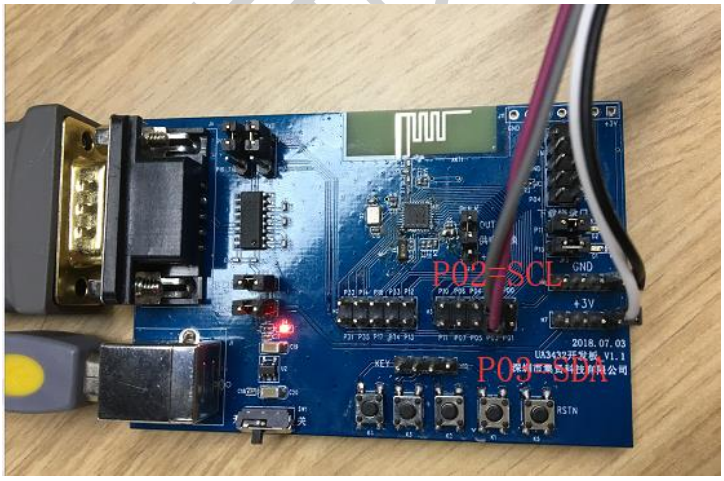
```
    iic_init();
    #endif
    UART_PRINTF("bk3432..\r\n");
    #if 1 //正常广播透传程序
    gpio_config(GPIOB_0, OUTPUT, PULL_NONE);//初始化GPIO
    while(1)
    {
        rwip_schedule();//schedule all pending events
        GLOBAL_INT_DISABLE();// Checks for sleep have to be done with interrupt disabled
        oad_updating_user_section_pro();
        if (!test_cnt)//只进入一次
        {
            ke_timer_set(READ_WRITE_NVR_FLASH,TASK_APP ,50);//读写内部flash_NVR
            test_cnt = 1;
        }
        if(wdt_disable_flag==1)
        {
            wdt_disable();
        }
        #if SYSTEM_SLEEP// Check if the processor clock can be gated
        sleep_type = rwip_sleep();
//        UART_PRINTF("sleep_type=%d \r\n",sleep_type);
        if((sleep_type & RW_MCU_DEEP_SLEEP) == RW_MCU_DEEP_SLEEP)
        {
            if(icu_get_sleep_mode())// 1:idel  0:reduce voltage
            {
                cpu_idle_sleep();
```

下载程序

代码编写完成后，在../bk3432_project\ble_app_gatt\output\app目录下找到bk3432_ble_app_merge.bin文件下载到开发板。至此，模拟IIC读写AT24C02应用教程叙述完毕

（八）硬件IIC读写AT24C02应用实验

硬件外设说明：用户需要外接AT24C02



功能解说

本节代码使用硬件IIC读写AT24C02的例程，500毫秒读写一次AT24C02，串口会打印出写入和读取出来的数据。

源码讲解

4.1编程思路：

1）初始化硬件IIC：i2c_init(0X50,0)

```
/*******************************************************************************
*作用     :初始化硬件IIC
*参数     :
         slaveAddr:从机地址
         baudRate :分频 选择0
*返回值   :
*其他说明:从机地址由器件的实际从机地址右移一位获得，
          例如：AT24C02的A0、A1、A2接地的情况下的从机地址是0xA0，
          但是slaveAddr要等于0x50
*******************************************************************************/
void i2c_init(uint32_t slaveAddr, uint32_t baudRate)
{
    uint32_t freq_div;
    if (baudRate == 0)
    {
        freq_div = I2C_CLK_DIVID_SET;
    }
    else
    {
        freq_div = NUMBER_ROUND_UP(NUMBER_ROUND_UP(I2C_DEFAULT_CLK, baudRate) - 6, 3) - 1;
    }
    if (REG_APB4_I2C_CN & I2C_CONFIG_I2C_ENABLE_MASK)
        {
            REG_APB4_I2C_STAT |= 0x200;
            REG_APB4_I2C_CN = 0;
            ICU_I2C_CLK_PWD_SET();
            REG_AHB0_ICU_INT_ENABLE &= ~(0x1 << 7);
        }
        ICU_I2C_CLK_PWD_CLEAR();
        // Enable GPIO P0.2, P0.3 peripheral function for I2C
        REG_APB5_GPIOA_CFG = (REG_APB5_GPIOA_CFG & (~((0x1<<2) | (0x1<<3) | (0x1<<26) | (0x1<<27))))
                           | ((0x1<<10) | (0x1<<11) | (0x1<<18) | (0x1<<19));
        REG_APB5_GPIOA_DATA   = REG_APB5_GPIOA_DATA & (~((0x1<<18) | (0x1<<19)));
         //Close JTAG to release GPIO to normal function
        CLOSE_JTAG_MODE();
        REG_APB4_I2C_STAT = 0x00000040;// 0100 0000; RXINT_MODE = 0x01, slvstop_stre_scl_en = 0x0
        REG_APB4_I2C_CN =  (I2C_CONFIG_I2C_ENABLE_SET)
                         | (I2C_CONFIG_INH_CLEAR)
                         | (I2C_CONFIG_SMBFTE_SET)
                         | (I2C_CONFIG_SMBTOE_SET)
                         | (I2C_CONFIG_CLOCK_SEL_FREQ_DIV)
                         | ((slaveAddr & 0x03FFUL) << I2C_CONFIG_SLAVE_ADDR_POSI)
                         | ((freq_div & 0x03FFUL) << I2C_CONFIG_FREQ_DIV_POSI)
                         | (0x04UL     << I2C_CONFIG_SCL_CR_POSI)
                         | (0x03UL     << I2C_CONFIG_IDLE_CR_POSI);
    REG_AHB0_ICU_INT_ENABLE |= (0x1 << 7);
    i2c_msg_reset();
}
```

2）将I2C_DRIVER宏定义设置为1

```
//DRIVER CONFIG
#define UART_DRIVER                     1
#define GPIO_DRIVER                     1
#define AUDIO_DRIVER                    0
#define RTC_DRIVER                      0
#define ADC_DRIVER                      1
#define I2C_DRIVER                      1
#define PWM_DRIVER                      1
```

3）查看中断入口函数IRQ_Exception(void)是否有调用IIC中断函数i2c_isr()

```
void IRQ_Exception(void)
{
    uint32_t IntStat;
    uint32_t irq_status;
    IntStat = intc_status_get();
    #if (SYSTEM_SLEEP) ...
    #endif
    #if (I2C_DRIVER)
    if(IntStat & INT_STATUS_I2C_bit)
    {
        irq_status |= INT_STATUS_I2C_bit;
        i2c_isr();
    }
    #endif
    #if (UART_DRIVER)
    if(IntStat & INT_STATUS_UART_bit) ...
    #endif
    #if (GPIO_DRIVER) ...
    #endif
    #if (PWM_DRIVER) ...
    #endif
    #if (ADC_DRIVER) ...
    #endif
    #if RTC_DRIVER ...
    #endif
    intc_status_clear(irq_status);
}
```

3）AT24C02读写函数。

```
uint8_t iic_readbuf[5];
uint8_t iic_writebuf[5];
void hardware_iic_AT24C02()//测试硬件iic
{
    iic_int_init(0X50);
    i2c_write(0x50,0x00,&iic_writebuf[1],1);
    UART_PRINTF("iic_writebuf[1] = %x\r\n",iic_writebuf[1]);
    Delay_ms(20);
    iic_writebuf[1]=iic_writebuf[1]+1;
    i2c_read(0x50,0x00,&iic_readbuf[0],1);
    UART_PRINTF("iic_readbuf[0]..= %x\r\n",iic_readbuf[0]);
}
```

4）利用软件定时器500毫秒定时调用hardware_iic_AT24C02() (有关软件定时器的使用请参考软件定时器应用实验篇)。

```
static int Hardwave_iic_AT24C02(ke_msg_id_t const msgid,
                                struct fff0s_fff1_level_ntf_cfg_ind const *param,
                                ke_task_id_t const dest_id,
                                ke_task_id_t const src_id)
{
    //ke_timer_set(LED_TIME_TEST,dest_id , 50);
    hardware_iic_AT24C02();
    ke_timer_set(HARDWARE_IIC_AT24C02,dest_id , 50);
    return (KE_MSG_CONSUMED);
}
```

5）在rw_main()里调用ke_timer_set(HARDWARE_IIC_AT24C02,TASK_APP,50);只需要调用一次和调用初始化函数。

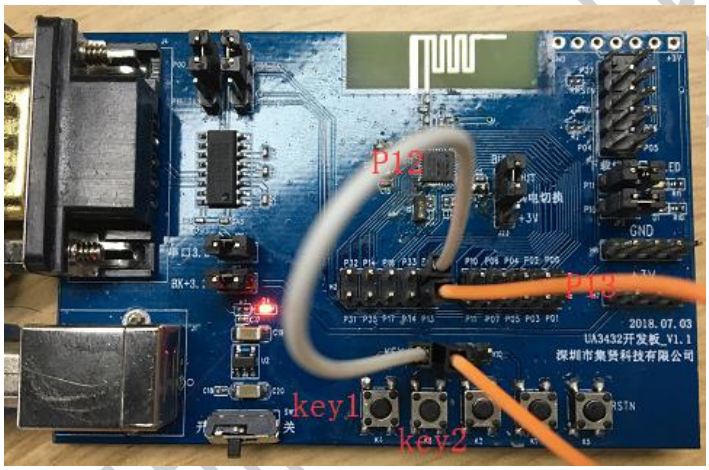
```
    i2c_init(0X50,0);//P02 P03
    #endif
    UART_PRINTF("bk3432..\r\n");
    #if 1 //正常广播透传程序
    gpio_config(GPIOB_0, OUTPUT, PULL_NONE);//初始化GPIO
    while(1)
    {
        rwip_schedule();//schedule all pending events
        GLOBAL_INT_DISABLE();// Checks for sleep have to be done with interrupt disabled
        oad_updating_user_section_pro();
        if (!test_cnt)//只进入一次
        {
            ke_timer_set(HARDWARE_IIC_AT24C02,TASK_APP,50);//硬件iic读写AT24C02
            test_cnt = 1;
        }
        if(wdt_disable_flag==1)
        {
            wdt_disable();
        }
        #if SYSTEM_SLEEP// Check if the processor clock can be gated
        sleep_type = rwip_sleep();
//        UART_PRINTF("sleep_type=%d \r\n",sleep_type);
        if((sleep_type & RW_MCU_DEEP_SLEEP) == RW_MCU_DEEP_SLEEP)
        {
            if(icu_get_sleep_mode())// 1:idel  0:reduce voltage
            {
```

下载程序

代码编写完成后，在../bk3432_project\ble_app_gatt\output\app目录下找到bk3432_ble_app_merge.bin文件下载到开发板。至此，硬件IIC读写AT24C02应用教程叙述完毕

（九）按键睡眠唤醒应用实验

硬件外设说明：



功能解说

本节代码实现按键睡眠唤醒功能。板子上电后正常运行，当key1按下时，进入睡眠模式，蓝牙不广播；当key2按下时，进入正常运行模式，蓝牙正常广播。

源码讲解

4. 1编程思路：

1）GPIO唤醒使能函数：set_wake_enble()设置用于唤醒的GPIO

```
void set_wake_enble(void)//设置唤醒的GPIO
{
    REG_APB5_GPIO_WUATOD_ENABLE = 0x00080800;    //允许GPIOB3唤醒
    REG_APB5_GPIO_WUATOD_STAT   = 0x00080800;    //GPIOB3唤醒有效
    REG_APB5_GPIO_WUATOD_TYPE = 0x00000000;      //设置唤醒条件:0:上升沿1:下降沿
    REG_AHB0_ICU_INT_ENABLE |= (0x01 << 9);      //开启GPIO中断使能
    REG_AHB0_ICU_DEEP_SLEEP0 = 0x00000800;       //GPIOB3 唤醒使能
}
```

2）将GPIO_DRIVER宏定义设置为1

```
//DRIVER CONFIG
#define UART_DRIVER                     1
#define GPIO_DRIVER                     1
#define AUDIO_DRIVER                    0
#define RTC_DRIVER                      0
#define ADC_DRIVER                      1
#define I2C_DRIVER                      1
#define PWM_DRIVER                      1
```

3）查看中断入口函数IRQ_Exception(void)是否有调用IIC中断函数gpio_isr()

```
void IRQ_Exception(void)
{
    uint32_t IntStat;
    uint32_t irq_status;
    IntStat = intc_status_get();
    #if (SYSTEM_SLEEP) ...
    #endif
    #if (UART_DRIVER)    ...
    #endif
    #if (GPIO_DRIVER)
    if(IntStat & INT_STATUS_GPIO_bit)
    {
        irq_status |= INT_STATUS_GPIO_bit;
        UART_PRINTF("gpio_isr.=%x\r\n",irq_status);
        gpio_isr();
    }
    #endif
    #if (PWM_DRIVER) ...
    #endif
    #if (ADC_DRIVER) ...
    #endif
    #if RTC_DRIVER ...
    #endif
    intc_status_clear(irq_status);
}
```

4）在rw_main()添加按键睡眠唤醒控制程序

```
gpio_config(GPIOB_2, INPUT, PULL_HIGH);
gpio_config(GPIOB_3, INPUT, PULL_HIGH); //初始化GPIO
while(1)
{
    Get_GPIO_Status = gpio_get_input(GPIOB_2);
    if(!Get_GPIO_Status) //进入睡眠
    {
        wdt_disable();
        while(!Get_GPIO_Status)
        {
            Get_GPIO_Status=gpio_get_input(GPIOB_2);
            UART_PRINTF("Get_GPIO_Status..\r\n");
        }
        wdt_disable();              //关闭看门狗
        appm_disconnect();          //断开连接
        appm_stop_advertising();//关闭广播
        set_wake_enble();           //设置唤醒
        icu_set_sleep_mode(0);   //选择睡眠模式
        rwip_prevent_sleep_clear(BK_DRIVER_TIMER_ACTIVE);
        UART_PRINTF("goto sleep\r\n");
        Sleep_Idle_Flag=0;
    }
    if(WakeFlag) //设置唤醒的GPIO有跳变沿时置一，芯片唤醒
    {
        WakeFlag = 0;
        wdt_enable(0x3fff);
        icu_set_sleep_mode(0);
        appm_start_advertising();
        UART_PRINTF("appm_start_advertising....\r\n");
    }
    //schedule all pending events
    rwip_schedule();
    // Checks for sleep have to be done with interrupt disabled
    GLOBAL_INT_DISABLE();
    oad_updating_user_section_pro();
```

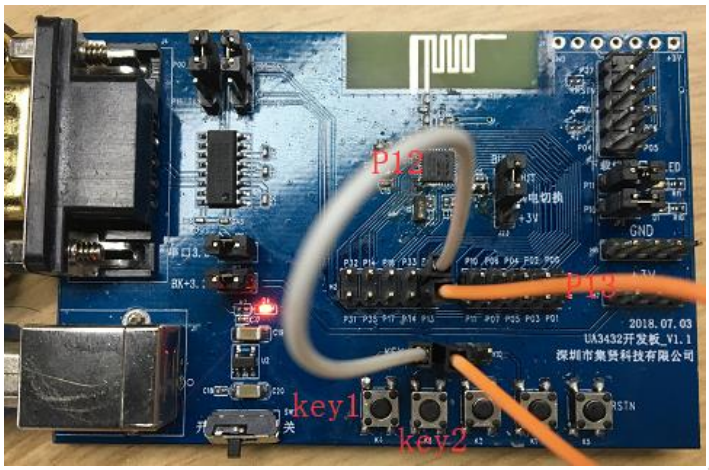
下载程序

代码编写完成后，在../bk3432_project\ble_app_gatt\output\app目录下找到bk3432_ble_app_merge.bin文件下载到开发板。至此，按键睡眠唤醒应用教程叙述完毕

二 BLE蓝牙应用篇

（九）BLE 双向透传

硬件外设说明：



功能解说

本节代码实现双向透传功能。app给蓝牙芯片发数据，在串口助手打印出来，串口助手给蓝牙芯片发数据，数据会在app上显示出来。

源码讲解

4. 1编程思路：

1）在uart_isr函数调用app_fff1_send_lvl函数

```
void uart_isr(void)
{
    uint32_t IntStat;
    #if BLE_TESTER
    void (*callback) (void*, uint8_t) = NULL;
    void* data =NULL;
    #endif
    UART_PRINTF("uart_isr \r\n");
    IntStat = uart_isr_stat_get();
    if(uart_rx_fifo_need_rd_isr_getf() || uart_rx_end_isr_getf()|| uart_rxd_wakeup_isr_getf())
    {
        while((REG_APB3_UART_FIFO_STAT & (0x01 << 21)))
        {
            uart_rx_buf[uart_rx_index++] = UART_READ_BYTE();
            if( UART0_RX_FIFO_MAX_COUNT == uart_rx_index )
            {
                uart_rx_index = 0;
            }
        }
        app_fff1_send_lvl(uart_rx_buf, uart_rx_index);
        uart_rx_done = 1;
        if(usrt_rx_cb)
        {
            (*usrt_rx_cb)(uart_rx_buf, uart_rx_index);
        }
        uart_rx_index = 0;
    }
    #if BLE_TESTER
    {
        callback = uart_env.rx.callback;
        data = uart_env.rx.dummy;
        if(callback != NULL)
        {
            uart_env.rx.callback = NULL;// Clear callback pointer
            uart_env.rx.dummy    = NULL;
            UART_PRINTF("UART callback 0x%08x\r\n",callback);// Call handler
            callback(data, RWIP_EIF_STATUS_OK);
        }
    }
    #endif // BLE_TESTER
    uart_isr_stat_set(IntStat);
}
```

2）将UART_DRIVER宏定义设置为1

```
//DRIVER CONFIG
#define UART_DRIVER                     1
#define GPIO_DRIVER                     1
#define AUDIO_DRIVER                    0
#define RTC_DRIVER                      0
#define ADC_DRIVER                      1
#define I2C_DRIVER                      1
#define PWM_DRIVER                      1
```

3）查看中断入口函数FIQ_Exception(void)是否有调用IIC中断函数uart_isr()

```
void IRQ_Exception(void)
{
    uint32_t IntStat;
    uint32_t irq_status;
    IntStat = intc_status_get();
    #if (SYSTEM_SLEEP) ...
    #endif
    #if (UART_DRIVER)
    if(IntStat & INT_STATUS_UART_bit)
    {
        irq_status |= INT_STATUS_UART_bit;
        uart_isr();
    }
    #endif
    #if (GPIO_DRIVER) ...
    #endif
    #if (PWM_DRIVER) ...
    #endif
    #if (ADC_DRIVER) ...
    #endif
    #if RTC_DRIVER ...
    #endif
    intc_status_clear(irq_status);
}
```

4）在rw_main()添加按键睡眠唤醒控制程序

```
gpio_config(GPIOB_2, INPUT, PULL_HIGH);
gpio_config(GPIOB_3, INPUT, PULL_HIGH); //初始化GPIO
while(1)
{
    Get_GPIO_Status = gpio_get_input(GPIOB_2);
    if(!Get_GPIO_Status) //进入睡眠
    {
        wdt_disable();
        while(!Get_GPIO_Status)
        {
            Get_GPIO_Status=gpio_get_input(GPIOB_2);
            UART_PRINTF("Get_GPIO_Status..\r\n");
        }
        wdt_disable();              //关闭看门狗
        appm_disconnect();          //断开连接
        appm_stop_advertising();//关闭广播
        set_wake_enble();           //设置唤醒
        icu_set_sleep_mode(0);  //选择睡眠模式
        rwip_prevent_sleep_clear(BK_DRIVER_TIMER_ACTIVE);
        UART_PRINTF("goto sleep\r\n");
        Sleep_Idle_Flag=0;
    }
    if(WakeFlag) //设置唤醒的GPIO有跳变沿时置一，芯片唤醒
    {
        WakeFlag = 0;
        wdt_enable(0x3fff);
        icu_set_sleep_mode(0);
        appm_start_advertising();
        UART_PRINTF("appm_start_advertising....\r\n");
    }
    //schedule all pending events
    rwip_schedule();
    // Checks for sleep have to be done with interrupt disabled
    GLOBAL_INT_DISABLE();
    oad_updating_user_section_pro();
```

下载程序

代码编写完成后，在../bk3432_project\ble_app_gatt\output\app目录下找到bk3432_ble_app_merge.bin文件下载到

开发板。至此，按键睡眠唤醒应用教程叙述完毕