

SDK3编程手册

1、概述

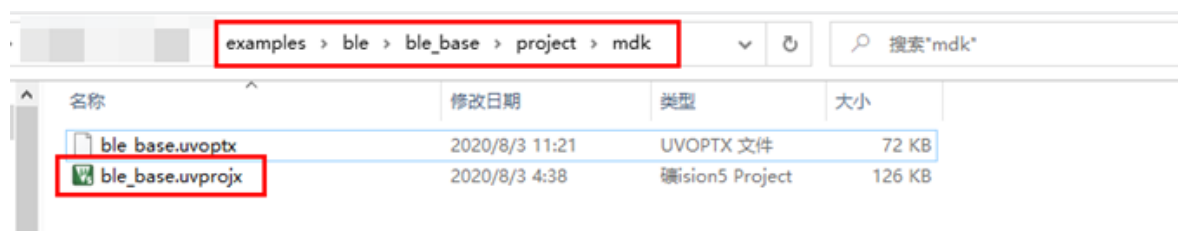
本文主要介绍在SDK3的基础上进行开发，主要内容如下：

- 基础应用，在此章节介绍bluex提供的基础应用，本文后续使用的应用编程，都是基于此应用来实现的
- LED点灯，第一个MCU应用，介绍如何使用SDK3，4行代码就实现LED的闪烁
- 驱动模型，基于LED应用，来说明SDK3的整体驱动模型，提出“服务（BXS）”的概念，介绍如何使用和操作服务
- BUTTON，基于驱动模型，，加深对“BXS”概念的理解和使用
- 第一个ble应用，基于“BXS”，实现一个功能应用

文档中涉及的代码，将全部保存在[SDK Demo](#)中

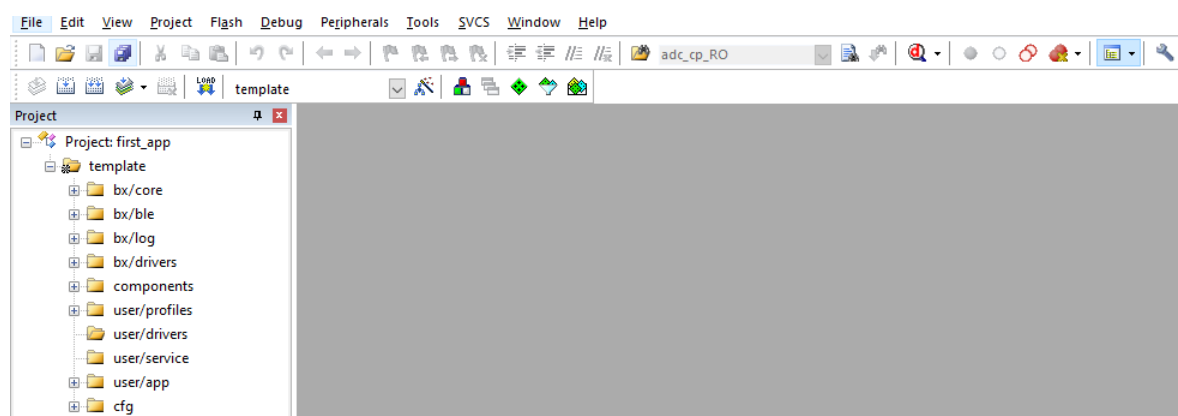
2、基础应用

2.1 打开工程



2.2 工程目录

可以看到左侧工程目录：



每个分类的说明如下：

名称	描述	备注
bx/core	bluex的主体代码部分	
bx/ble	bluex的ble协议栈部分	
bx/log	bluex的调试输出，支持使用RTT和UART两种	
bx/drivers	bluex提供的基础驱动代码	
components	某些组件，第三方的或者bluex提供的	
user/profiles	用户的BLEprofile，默认存在三个profile，设备信息、OTA务，和用户自定义	
user/drivers	存放用户代码需要的驱动	
user/service	存放用户自己实现的BXS服务	BXS的概念可以查看《SDK3编程手册》
user/app	用户的应用代码	
cfg	SDK3的配置	

2.3 main.c

main.c放在bx/core下，主要代码如下：

```

int main( void )
{
    ble_init();                //ble蓝牙的初始化
    bx_kernel_init();          //bluex内核初始化
    bxsh_init();               //shell脚本的初始化

    struct bx_service svc;      //定义一个bx_service变量，用于处理用
    户的属性、消息等
    svc.prop_set_func = NULL;    //设置用户属性的入口，此处为空，可以自
    行修改，该接口的含义具体在下文有介绍
    svc.prop_get_func = NULL;   //获取用户属性的入口，此处为空，可以自
    行修改，该接口的含义具体在下文有介绍
    svc.msg_handle_func = user_msg_handle_func; //用户消息的处理入口，默认为
    user_msg_handle_func，可以自行修改，该接口的含义具体下文有介绍
    svc.name = "user service";  //bx_service的名称
    user_service_id = bx_register(&svc); //将bx_service注册到内核

    user_init();                //用户的初始化
    user_app();                 //用户的app操作

    while( 1 ) {
        ble_schedule();         //ble内核调度
        bx_kernel_schedule();   //blux 内核调度
        bxsh_run();             //shell脚本的调度
    }
}

```

关于bluex内核和bx_service(BXS)服务，下文会有详细说明。

2.4 user_app.c

此.c文件，主要就是用户应用要填写的代码，默认代码如下：

```
/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
 *-----*/
void user_init( void )
{

}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
 *-----*/
void user_app( void )
{

}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
 *-----*/
bx_err_t user_msg_handle_func(s32 id, u32 msg,u32 param0,u32 param1 )
{

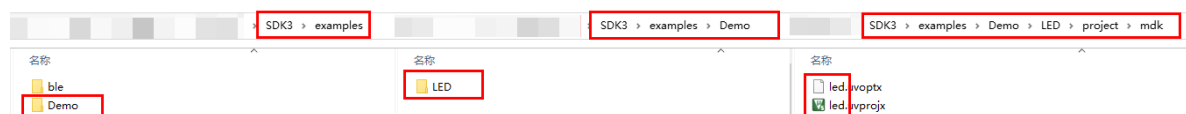
    return BX_OK;
}
```

仅仅定义了几个函数，并没有什么实际的实现，具体代码根据实际应用来编写。

3、LED点灯

3.1 新建应用

- 打开SDK3/examples路径
- 直接复制文件夹ble，把名称改为Demo
- 把ble_base改为LED， ble_base改为led，最后效果如下：



3.2 编写代码

只需要编写user_app.c一个文件即可。

因为需要使用到GPIO部分，因此，需要引用bx_service_gpio.h，在第文件开头处 include 即可：

```

/* includes -----*/

#include "bx_kernel.h"
#include "user_app.h"

#include "bx_service_gpio.h"

/* private define -----*/

```

- user_init()

```

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

void user_init( void )
{
    bxs_gpio_register();    //注册gpio服务
}

```

- user_app()

```

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

void user_app( void )
{
    bx_call( bxs_gpio_a_id(), BXM_OPEN, 0, 0 );
    bx_set( bxs_gpio_a_id(), BXP_MODE, 2, BX_GPIO_MODE_OUTPUT );
    bx_repeat( bxs_gpio_a_id(), BXM_TOGGLE, 2, 0, 1000 );
}

```

代码中使用了bx_call、bx_set、bx_repeat几个函数，同时出现一些新的单词BXM_OPEN, BMP_MODE, BXM_TOGGLE, BX_GPIO_MODE_OUTPUT，这些东西在下文都会有讲到，目前暂时可以不用了解，仅从字面上解释代码：

```

void user_app( void )
{
    /* 初始化gpioa */
    bx_call( bxs_gpio_a_id(), BXM_OPEN, 0, 0 );
    /* 将gpioa中的第2个引脚设置为输出模式 */
    bx_set( bxs_gpio_a_id(), BXP_MODE, 2, BX_GPIO_MODE_OUTPUT );
    /* 让gpioa中的第2个引脚重复做翻转操作，重复间隔为1000ms */
    bx_repeat( bxs_gpio_a_id(), BXM_TOGGLE, 2, 0, 1000 );
}

```

3.3 演示



















编译以上代码，把代码烧录到设备，即可看到pin2对应的LED灯，每隔1S闪烁一次。

4、驱动模型

SDK3驱动模型提供基础驱动层和驱动服务层，可以使用任意一层来实现代码开发，基础驱动与一般MCU开发流程无疑，驱动服务则整合了多个特性，服务有属性和消息，将基础驱动、设备驱动、业务逻辑、应用等组成部分分离开，同时把接口统一起来，而基础驱动则无此功能，下文主要是对服务层的介绍。

SDK3所有例程都是使用服务层来编写代码，我们鼓励用户使用服务层来开发应用，这样做的好处就是屏蔽了底层驱动的具体实现，若应用某一天更换芯片，直接适配底层驱动即可，而上层应用代码并不需要做任何修改。

4.1 基础驱动



















 bx_drv_adc.c	 bx_drv_adc.h
 bx_drv_gpio.c	 bx_drv_gpio.h
 bx_drv_iic.c	 bx_drv_iic.h
 bx_drv_io_mux.c	 bx_drv_io_mux.h
 bx_drv_pwm.c	 bx_drv_pwm.h
 bx_drv_spim.c	 bx_drv_spim.h
 bx_drv_tim.c	 bx_drv_tim.h
 bx_drv_uart.c	 bx_drv_uart.h
 bx_drv_wdt.c	 bx_drv_wdt.h

基础驱动与一般MCU的驱动无差异，直接使用即可。

4.2 驱动服务

4.2.1 简介

- 服务层是对基础驱动、外围IC、外围模块统一接口的一层代码
- 每个服务都有通用的属性和消息，也有自定义的属性和消息
- 把MCU一般的初始化配置，统一变为设置属性
- 把MCU一般的执行动作（函数调用），统一变为向内核提交/呼叫消息
- 提供原子操作，也提供常用的快捷配置

 bx_service.c	 bx_service.h
 bx_service_adc.c	 bx_service_adc.h
 bx_service_gpio.c	 bx_service_gpio.h
 bx_service_iic.c	 bx_service_iic.h
 bx_service_pwm.c	 bx_service_pwm.h
 bx_service_spi.c	 bx_service_spi.h
 bx_service_tim.c	 bx_service_tim.h
 bx_service_uart.c	 bx_service_uart.h
 bx_service_wdt.c	 bx_service_wdt.h

4.2.2 定义

服务的定义在 bx_kernel.h中可以看到：

```

typedef bx_err_t( *msg_handle_f )( s32 svc, u32 msg, u32 param0, u32 param1 );
//消息处理接口
typedef bx_err_t( *prop_handle_f )( s32 svc, u32 prop, u32 param0, u32 param1 );
//属性处理接口

struct bx_service {
    prop_handle_f    prop_set_func;        //服务属性的设置入口
    prop_handle_f    prop_get_func;       //服务属性的获取入口
    msg_handle_f     msg_handle_func;     //服务消息的处理入口
    char             *    name;           //服务的名称
};

```

- msg_handle_f

参数	含义	备注
svc	service 的缩写，代表具体某一个服务	
msg	message 的缩写，代表具体某一个消息	
param0	参数0，具体含义根据msg来定义	类似windows消息
param1	参数1，具体含义根据msg来定义	

- prop_handle_f

参数	含义	备注
svc	service 的缩写，代表具体某一个服务	
prop	property 的缩写，代表具体某一个属性	
param0	参数0，具体含义根据prop来定义	
param1	参数1，具体含义根据prop来定义	

4.2.3 属性

初始化的配置，就是属性，比如UART的波特率、数据位之类的，这些就属于uart服务的属性，按键的数量，按键按下的间隔，也是属性，大部分属性是可以量化的。

- 每个服务有通用属性，也有自己专属的属性

```

//通用属性
enum bx_property {
    BXP_HANDLE,

    BXP_STATE,
    BXP_MODE,
    BXP_LOCK,

    BXP_TIMEOUT,
    BXP_SPEED,
    BXP_ADDR,
    BXP_SUB_ADDR,
    BXP_PIN,
    BXP_ADDR_BIT,

```

```

    BXP_DATA_BIT,

    BXP_CHANNEL,
    BXP_VALUE,

    BXP_VERSION,
    BXP_FIRST_USER_PROP,
};

```

- GPIO专有的属性

```

//GPIO的属性
enum bx_property_gpio {
    BXP_GPIO_FIRST = BXP_FIRST_USER_PROP,

    BXP_GPIO_PULL,
};

```

注意在自定义服务的自定义属性时，**第一个属性的值，必须为 BXP_FIRST_USER_PROP**

- 属性可以定义原子操作，也可以定义快捷常用。如UART可以单独设置数据位、校验、波特率等属性，也可以直接定义一个快捷方式是115200_8_1_N的常用设置

```

enum bx_property_uart {
    BXP_UART_FIRST = BXP_FIRST_USER_PROP,

    BXP_UART_TX_PIN,
    BXP_UART_RX_PIN,
    BXP_UART_CTS_PIN,
    BXP_UART_RTS_PIN,

    BXP_UART_PARITY,
    BXP_UART_STOP_BIT,

    BXP_UART_115200_8_1_N,

};

```

4.2.4 消息

消息是就是要做的动作，比如uart开始读，开始写，这个读写就是消息，按键的单击、双击，也是消息，大部分消息是不能量化的。

- 每个服务有通用消息，也有自己专属的属性

```

//通用消息
enum bx_message {
    BXM_OPEN,
    BXM_CLOSE,

    BXM_READ,
    BXM_WRITE,
    BXM_TOGGLE,

    BXM_START,
    BXM_STOP,
};

```

```

    BXM_PREPARE_SLEEP,
    BXM_WAKEUP,

    BXM_READ_DONE,
    BXM_WRITE_DONE,
    BXM_DATA_UPDATE,

    BXM_FIRST_USER_MSG,
};

```

- GPIO专有的属性

```

enum bx_msg_gpio {
    BXM_GPIO_FIRST = BXM_FIRST_USER_MSG,

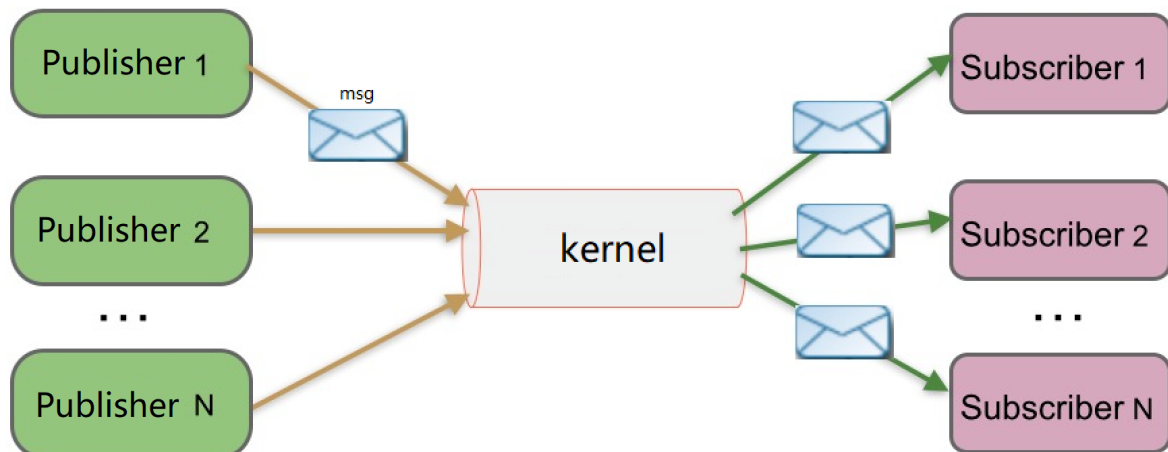
    BXM_GPIO_INTR,
    BXM_GPIO_EXT_INTR,
};

```

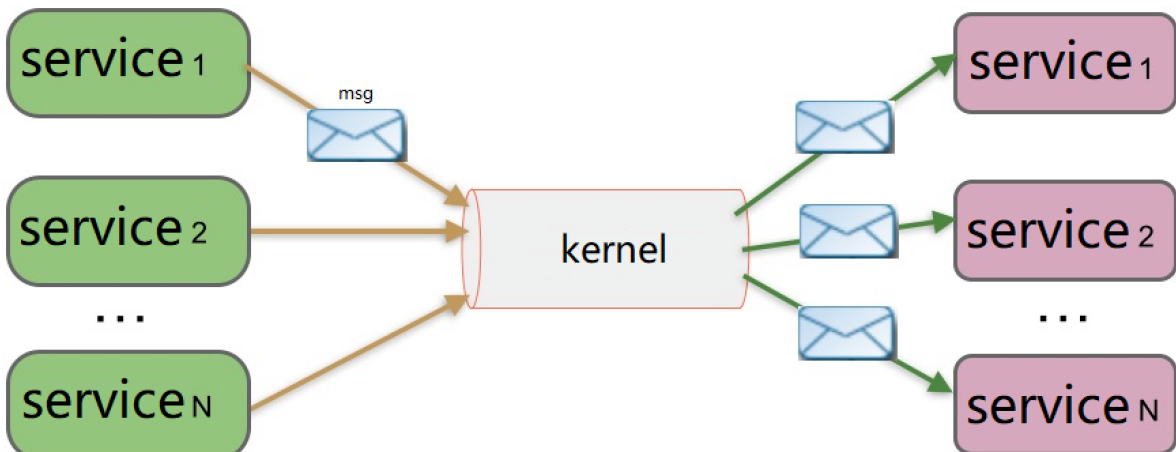
注意在自定义服务的自定义消息时，**第一个消息的值，必须为 BXM_FIRST_USER_MSG**

4.3 订阅发布模型

发布定于模型常见的框架如下图：



发布者发送到内核的消息，只有订阅了该消息的订阅者才会收到消息，而在驱动服务层，发布者和订阅者，都可以是某一个服务，因此可以换成下图：



当A服务发布某个消息，如果这个消息被B服务订阅了，那么内核就会将A服务的消息，转发到到B服务上，B服务就可以在消息处理函数里面执行向应的动作。

每一个服务都有自己产生的消息以及可以接收的消息，具体每个外设驱动接收的消息，以及产生的消息，消息对应的参数，会有一个专门的文档介绍。

4.4 常用API

API	介绍	备注
bx_register	先内核注册一个服务	可以配置最大可注册的数量
bx_set	设置某个服务的属性，同步操作，与一般函数调用一样	
bx_get	获取某个服务的属性，同步操作，与一般函数调用一样	
bx_call	呼叫某个服务的某个消息，同步操作，与一般函数调用一样	
bx_post	立即向某个服务发送某个消息，异步操作	发送完消息之后，消息的处理时间由内核决定
bx_defer	延迟发送，等待一段时间后再向某个服务发送某个消息	
bx_repeat	重复向某个服务发送某个消息，无限循环	
bx_repeatn	重复向某个服务发送某个消息，有循环次数	
bx_cancel	取消向某个服务发送某个消息	
bx_subscribe	订阅某个服务的某个消息	订阅的发起者是当前内核环境运行中的服务
bx_subscribeex	订阅某个服务的某个消息	显式表明订阅的发起者是哪个服务
bx_public	发布某个服务的某个消息	
bx_msg_source	获取当前消息是来自于哪一个服务	

call跟post消息的区别在于，call是不需要把消息发到内核，而是直接根据指针调用函数，post是把消息提交给内核，然后在适当的时候，内核在后台调用函数

4.5 延迟执行

一般MCU延迟执行的代码会是如下形式：

```
void funcA(void)
{
    ...
}
```

```

}

void funcB(void)
{
    ...
}

void delay(uint32_t time_ms)
{
    ...
}

void main(void)
{
    ...
    funcA();
    delay(1000);
    funcB();
    delay(1000);
    ...
}

```

执行了funcA后，需要等待1000ms，然后再执行funcB，再等待1000ms，这1000ms的延时，完全是浪费CPU运算能力，也浪费功耗，SDK3提供一个释放CPU，同时又兼备低功耗的延迟操作。

以上代码，换成SDK3的执行方式如下，首先定义两个消息：（可以在user_app.h中定义）

```

enum bx_msg_ble{
    USM_FIRST = BXM_FIRST_USER_MSG,
    USM_FUNCA_EXCUTE,
    USM_FUNCB_EXCUTE,
};

```

然后再通过发消息的方式延迟执行：

```

void funcA(void)
{
    ...
}

void funcB(void)
{
    ...
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

void user_app( void )
{
    //先发送 USM_FUNCA_EXCUTE 消息
    bx_post( user_service_id, USM_FUNCA_EXCUTE, 0, 0 );
}

```

```

//延迟1000ms 发送 USM_FUNCB_EXCUTE 消息
bx_defer( user_service_id, USM_FUNCB_EXCUTE, 0, 0, 1000 );
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
bx_err_t user_msg_handle_func( s32 svc, u32 msg, u32 param0, u32 param1 )
{
    s32 msg_src = bx_msg_source();
    if( msg_src == user_service_id ) {
        switch( msg ) {
            case USM_FUNCA_EXCUTE:
                funcA();
                break;
            case USM_FUNCB_EXCUTE:
                funcB();
                break;
            default:
                break;
        }
    }
    return BX_OK;
}

```

以上代码，在 user_app函数 中，首先发送 USM_FUNCA_EXCUTE ，内核会先收到该消息，然后通过调度到达 user_msg_handle_func 函数中，此时判断到消息是USM_FUNCA_EXCUTE，就会调用 funcA 函数，1000ms后，内核会收到 USM_FUNCB_EXCUTE 消息，通过调度到达 user_msg_handle_func 函数中，此时判断到消息是USM_FUNCB_EXCUTE，就会调用 funcB函数。

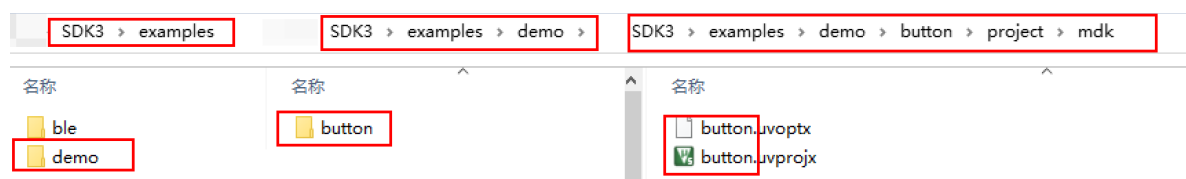
4.6 特别说明

每一个服务，都会有各自的消息和属性，每个属性和消息对应的参数的含义不一样，详情请参考[驱动参数说明](#)

5、BUTTON

5.1 新建应用

- 打开SDK3/examples路径
- 直接复制文件夹ble，把名称改为demo
- 把ble_base改为button，ble_base改为button，最后效果如下：



5.2 编写代码

只需要编写user_app.c一个文件即可。

因为需要使用到GPIO部分，因此，需要引用bx_service_gpio.h，在第文件开头处 include 即可：

```

/* includes -----*/

#include "bx_kernel.h"
#include "user_app.h"

#include "bx_service_gpio.h"
#include "bx_shell.h"
/* private define -----*/

```

- user_init()

```

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

void user_init( void )
{
    bxs_gpio_register();    //注册gpio服务
}

```

- user_app()

```

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

void user_app( void )
{
    s32 id = bxs_gpio_a_id();
    bx_call( id, BXM_OPEN, 0, 0 );    //启动GPIO服务
    bx_set(id, BXP_MODE, 2, BX_GPIO_MODE_OUTPUT);    //设置PIN2为输出

    bx_set(id, BXP_GPIO_PULL, 15, BX_GPIO_PULLUP);    //把P15上拉
    bx_set(id, BXP_MODE, 15, BX_GPIO_MODE_EIT_RISING);    //把P15设置为外部中断
    bx_subscribe( id, BXM_GPIO_EXT_INTR, 0, 0 );    //订阅GPIO服务的
    BXM_GPIO_EXT_INTR消息
}

```

- user_msg_handle_func

```

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

bx_err_t user_msg_handle_func( s32 id, u32 msg, u32 param0, u32 param1 )
{
    if( bx_msg_source() == bxs_gpio_a_id() ) {    //判断消息的来源是否为
    GPIO服务
        switch( msg ) {    //判断接收到的消息

```

```

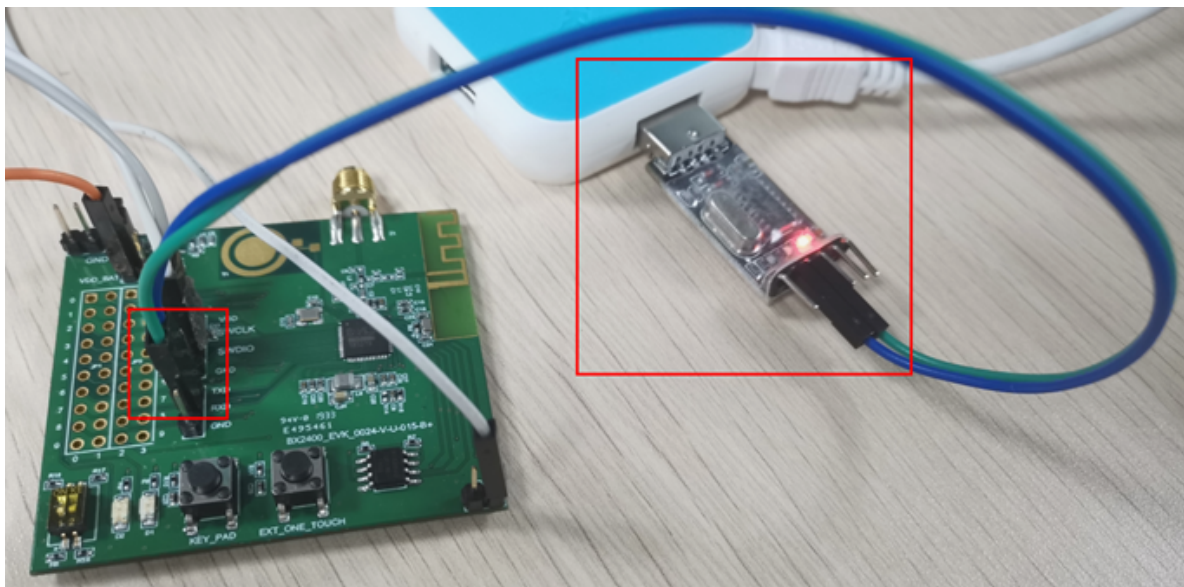
        case BXM_GPIO_EXT_INTR: //接收到的消息是
            BXM_GPIO_EXT_INTR
            bxsh_logln( "BXM_GPIO_EXT_INTR" ); //串口输出
            bx_post( bxs_gpio_a_id(),BXM_TOGGLE,2,0 ); //向GPIO服务发送翻转
            pin2的消息
            break;

        default:
            break;
    }
}
return BX_OK;
}

```

5.3 演示

- 硬件连接



编译以上代码，把代码烧录到设备，当单击P15对应的button时，可以看到PIN2对应的LED被翻转了，同时串口打印出“BXM_GPIO_EXT_INTR”

串口配置	
COM14	[16:18:51.455] Welcome To Use Bluex Shell
115200	[16:18:51.456] # BXM_GPIO_EXT_INTR
8	[16:18:58.290] BXM_GPIO_EXT_INTR
1	[16:18:59.149] BXM_GPIO_EXT_INTR
None	[16:18:59.973] BXM_GPIO_EXT_INTR
GB2312	[16:19:00.936] BXM_GPIO_EXT_INTR
	[16:19:01.836] BXM_GPIO_EXT_INTR
	[16:19:02.670] BXM_GPIO_EXT_INTR
	[16:19:03.465] BXM_GPIO_EXT_INTR
	[16:19:04.170] BXM_GPIO_EXT_INTR
	[16:19:04.821] BXM_GPIO_EXT_INTR

以上代码仅为button的基础演示，加深对服务层属性消息的理解，以及对实际使用button的时候，会有单击、双击、长按、短按，以及功耗等要求。

6、第一个BLE应用

在编写应用时，建议用户先对应用做一个模块的区分，或者硬件的区分，然后对模块/硬件抽象成一个service，然后使用驱动模型中常用的API来实现自己的应用程序，因此在编写应用时，首先要做到的就是编写自己的service，下文会以按键点灯的方式，一步一步来说明怎么编写一个应用。

6.1 功能简介

本应用会实现一个通过蓝牙点灯的功能，主要功能如下：

- 通过向指定蓝牙属性写入数据，熄灭/点亮LED
- 通过板载的button，熄灭/点亮LED，并向指定属性上报数据

6.2 编写 bx_service

根据功能简介，不难发现，实际上只有3个服务：

- LED模块
- BUTTON模块
- BLE通讯模块

其中BLE通讯模块，是本芯片自带的，已经写好，可以直接使用，因此需要对LED和BUTTON模块抽象为service层

6.2.1 led

- 新建文件：user_service_led.c、user_service_led.h

直接复制模板：user_service_XXX.c、user_service_XXX.h，修改文件名为 user_service_led 即可



然后把文件中的"XXX"修改为LED即可。

- 确定led模块的专有的属性和消息（通用的消息已经定义好，不需要重新定义）

由于LED暂时没有专用的属性和消息，因此不需要定义，最总得到的.h文件：

```
/**
 *
 * *****
 * @file    :   .h
 * @version:
 * @author :
 * @brief  :
 *
 * *****
 * @attention
 *
 * <h2><center>&copy; Copyright(c) . BLUEX Microelectronics.
 * All rights reserved.</center></h2>
 *
 *
 * *****
 */

/* Define to prevent recursive inclusion -----*/
```

```

#ifndef __USER_SERVICE_LED_H__
#define __USER_SERVICE_LED_H__

#ifdef __cplusplus
extern "C" {
#endif

/* includes -----*/
#include "bx_type_def.h"
#include "bx_msg_type_def.h"
#include "bx_property_type_def.h"

/* exported paras -----*/

/* exported types -----*/
enum user_property_led {
    USP_LED_FIRST = BXP_FIRST_USER_PROP,

};

enum user_msg_led{
    USM_LED_FIRST = BXM_FIRST_USER_MSG,

};
/* exported variables -----*/

/* exported constants -----*/

/* exported macros -----*/

/* exported functions -----*/

bool    us_led_register( void );
s32     us_led_id( void );

#ifdef __cplusplus
}
#endif

#endif /* __USER_SERVICE_LED_H__ */

/***** (C) COPYRIGHT BLUEX *****/

```

- 在user_service_led.c中，修改“xxx”为LED，实现LED模块的OPEN、CLOSE、WRITE、TOGGLE等消息，最终得到的代码：

```

/**
*****
* @file    :   .c
* @version:
* @author  :
* @brief   :
*****
* @attention
*
* <h2><center>&copy; Copyright(c) . BLUEX Microelectronics.

```

```

* All rights reserved.</center></h2>
*
*
*****
*/

/* includes -----*/

#include "bx_kernel.h"
#include "user_service_led.h"

#include "bx_service_gpio.h"

/* config -----*/
#define LED_PIN 2
/* private define -----*/

/* private typedef -----*/
struct us_led_service {
    s32 id;

};

/* private variables -----*/
static struct us_led_service led_svc = {0};

/* exported variables -----*/

/* private macros -----*/
#define GET_LED_SERVICE_BY_ID( p_svc, svc_id )      \
do{                                                  \
    if( ( svc_id ) == led_svc.id ) {                \
        p_svc = &led_svc;                          \
    } else {                                         \
        return BX_ERR_NOTSUP;                       \
    }                                               \
}while(0)

/*===== private function =====*/
/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
static void led_open( void )
{
    s32 id = bxs_gpio_a_id();
    bx_call( id, BXM_OPEN, 0, 0 );
    bx_set( id, BXP_MODE, LED_PIN, BX_GPIO_MODE_OUTPUT );
}
/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
static void led_close( void )

```



```

{
    s32 id = bxs_gpio_a_id();
    bx_call( id, BXM_CLOSE, 0, 0 );
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

static int32_t led_write( bool is_light )
{
    s32 id = bxs_gpio_a_id();
    if( is_light ) {
        bx_call( id, BXM_WRITE, LED_PIN, 0 );
    } else {
        bx_call( id, BXM_WRITE, LED_PIN, 1 );
    }
    return 0;
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

static int32_t led_toggle( )
{
    s32 id = bxs_gpio_a_id();
    bx_call( id, BXM_TOGGLE, LED_PIN, 0 );
    return 0;
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

static bx_err_t led_msg_handle(s32 svc_id, u32 msg, u32 param0, u32 param1 )
{
    // struct us_led_service * p_svc;
    // GET_LED_SERVICE_BY_ID( p_svc, svc_id );

    switch( msg ) {

        case BXM_OPEN:
            led_open();
            break;

        case BXM_CLOSE:
            led_close();
            break;

        case BXM_WRITE:
            led_write( param0 );
            break;
    }
}

```

```

        case BXM_TOGGLE:
            led_toggle( );
            break;

        default:
            return BX_ERROR;
    }
    return BX_OK;
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
static bx_err_t led_property_set(s32 svc_id, u32 property, u32 param0, u32
param1 )
{
    // struct us_led_service * p_svc;
    // GET_LED_SERVICE_BY_ID( p_svc, svc_id );

    switch( property ) {
        default:
            return BX_ERR_NOTSUP;
    }
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
static bx_err_t led_property_get(s32 svc_id, u32 property, u32 param0, u32
param1 )
{
    // struct us_led_service * p_svc;
    // GET_LED_SERVICE_BY_ID( p_svc, svc_id );

    switch( property ) {
        default:
            return BX_ERR_NOTSUP;
    }
}

/*===== end of private function =====*/

/*===== exported function =====*/

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :

```

```

-----*/
bool us_led_register( void )
{
    struct bx_service svc;
    svc.name = "led service";
    svc.msg_handle_func = led_msg_handle;
    svc.prop_get_func = led_property_get;
    svc.prop_set_func = led_property_set;
    led_svc.id = bx_register( &svc );
    if( led_svc.id == -1 ) {
        return false;
    }
    return true;
}
/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

s32 us_led_id( void )
{
    return led_svc.id;
}

/*===== end of exported function =====*/

/*===== import function =====*/

/*===== end of import function =====*/

/*===== interrupt function =====*/

/*===== end of interrupt function =====*/

/***** (C) COPYRIGHT BLUEX *****/END OF FILE****/

```

在以上代码，LED只实现了对一个pin引脚（PIN2）的操作，可以考虑一下多个pin引脚的实现方式。

- 在user_app.c中，写入代码，测试LED模块是否正常：

```

/**
*****
 * @file : main.c
 * @version:
 * @author :
 * @brief :
*****
 * @attention
 *
 * <h2><center>&copy; Copyright(c) . BLUEX Microelectronics.
 * All rights reserved.</center></h2>

```

```

*
*
*****
*/

/* includes -----*/

#include "bx_kernel.h"
#include "user_app.h"
#include "bx_service_gpio.h"
#include "user_service_led.h"

/* private define -----*/

/* private typedef -----*/

/* private variables -----*/

/* exported variables -----*/

/*===== private function =====*/

/*===== end of private function =====*/

/*===== exported function =====*/
/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
void user_init( void )
{
    bxs_gpio_register();
    us_led_register();
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
void user_app( void )
{
    bx_call( us_led_id(), BXM_OPEN, 0, 0);
    bx_repeat( us_led_id(), BXM_TOGGLE, 0, 0, 1000);
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
bx_err_t user_msg_handle_func(s32 svc, u32 msg, u32 param0, u32 param1 )
{

```

```

        return BX_OK;
    }
    /*===== end of exported function =====*/

    /*===== import function =====*/

    /*===== end of import function =====*/

    /*===== interrupt function =====*/

    /*===== end of interrupt function =====*/

    /***** (C) COPYRIGHT BLUEX *****/END OF FILE*****/

```

6.2.2 button

- 新建文件：user_service_button.c、user_service_button.h

直接复制模板：user_service_xxx.c、user_service_xxx.h，修改文件名为 user_service_button即可



然后把文件中的"XXX"修改为BUTTON即可。

- 确定button模块的专有的属性和消息（通用的属性消息已经定义好，不需要重新定义）

```

enum user_msg_btn{
    USM_BTN_FIRST = BXM_FIRST_USER_MSG,

    USM_BTN_CLICK,
};

```

此处btn暂无属性，仅定义一个单击消息：USM_BTN_CLICK。最终得到代码：

```

/**
*****
* @file   :   .h
* @version:

```

```

* @author :
* @brief :
*****

* @attention
*
* <h2><center>&copy; Copyright(c) . BLUEX Microelectronics.
* All rights reserved.</center></h2>
*
*
*****

*/

/* Define to prevent recursive inclusion -----*/
#ifndef __USER_SERVICE_BUTTON_H__
#define __USER_SERVICE_BUTTON_H__

#ifdef __cplusplus
extern "C" {
#endif

/* includes -----*/
#include "bx_type_def.h"
#include "bx_msg_type_def.h"
#include "bx_property_type_def.h"

/* exported paras -----*/

/* exported types -----*/
enum user_property_btn {
    USP_BTN_FIRST = BXP_FIRST_USER_PROP,

};

enum user_msg_btn{
    USM_BTN_FIRST = BXM_FIRST_USER_MSG,

    USM_BTN_CLICK,
};
/* exported variables -----*/

/* exported constants -----*/

/* exported macros -----*/

/* exported functions -----*/

bool    us_btn_register( void );
s32     us_btn_id( void );

#ifdef __cplusplus
}
#endif

#endif /* __USER_SERVICE_BUTTON_H__ */

/***** (C) COPYRIGHT BLUEX *****/END OF FILE****/

```

在user_service_button.c中，实现OPEN、CLOSE消息，以及订阅gpio_service产生的BXM_GPIO_EXT_INTR 消息：

```
/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
static void button_open( void )
{
    s32 id = bxs_gpio_a_id();
    bx_call( id, BXM_OPEN, 0, 0 );
    /* 引脚配置 */
    bx_set( id, BXP_GPIO_PULL, 15, BX_GPIO_PULLUP );
    bx_set( id, BXP_MODE, 15, BX_GPIO_MODE_EIT_RISING );
    /* button服务订阅gpio_a服务的BXM_GPIO_EXT_INTR消息，此处暗含订阅的发起者为button服务 */
    /*
    bx_subscribe(id,BXM_GPIO_EXT_INTR,0,0);

    /* button服务订阅gpio_a服务的BXM_GPIO_EXT_INTR消息 */
    /*此处直接表明订阅的发起者为button服务，与上一行代码等效 */
    //bx_subscribeex(btn_svc.id,id,BXM_GPIO_EXT_INTR,0,0);
    */
}
/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
static void button_close( void )
{
    s32 id = bxs_gpio_a_id();
    bx_call( id, BXM_CLOSE, 0, 0 );
}
```

在 btn_msg_handle 中处理button本身的消息，以及button订阅了的消息：

```
/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/
static bx_err_t btn_msg_handle(s32 svc_id, u32 msg, u32 param0, u32 param1 )
{
    /* 判断消息的来源是GPIO服务 */
    if( bx_msg_source() == bxs_gpio_a_id() ){
        /* 判断消息是前面订阅的消息 BXM_GPIO_EXT_INTR */
        if( msg == BXM_GPIO_EXT_INTR ) {
            /* BXM_GPIO_EXT_INTR消息的参数0，代表触发中断的引脚 */
            if( param0 & ( 0x01 << BTN_PIN ) ) {
                return bx_public( btn_svc.id, USM_BTN_CLICK, BTN_PIN, 0 );
            }
        }
    }
}
```

```

switch( msg ) {
    case BXM_OPEN:
        button_open();
        break;

    case BXM_CLOSE:
        button_close();
        break;

    default:
        return BX_ERR_NOTSUP;
}
}

```

- 在user_app.c中，写入代码，测试button模块是否正常：

```

/**
*****
* @file    :   main.c
* @version:
* @author  :
* @brief   :
*****
* @attention
*
* <h2><center>&copy; Copyright(c) . BLUEX Microelectronics.
* All rights reserved.</center></h2>
*
*
*****
*/

/* includes -----*/

#include "bx_kernel.h"
#include "user_app.h"
#include "bx_service_gpio.h"
#include "user_service_led.h"
#include "user_service_button.h"
/* private define -----*/

/* private typedef -----*/

/* private variables -----*/

/* exported variables -----*/

/*===== private function =====*/

/*===== end of private function =====*/

/*===== exported function =====*/
/** -----
* @brief   :

```



```

* @note      :
* @param     :
* @retval    :
-----*/
void user_init( void )
{
    bxs_gpio_register();
    us_led_register();
    us_btn_register();
}

/** -----
* @brief     :
* @note      :
* @param     :
* @retval    :
-----*/
void user_app( void )
{
    bx_post( us_led_id(), BXM_OPEN, 0, 0);
    //bx_repeat( us_led_id(), BXM_TOGGLE, 0, 0, 1000);
    bx_post( us_btn_id(), BXM_OPEN, 0, 0);

    /* user服务订阅button服务的USM_BTN_CLICK消息，此处暗含订阅的发起者为user服务 */
    bx_subscribe(us_btn_id(), USM_BTN_CLICK, 0, 0);

    /* user服务订阅button服务的USM_BTN_CLICK消息 */
    /*此处直接表明订阅的发起者为button服务，与上一行代码等效 */
    bx_subscribeex(user_service_id, us_btn_id(), USM_BTN_CLICK, 0, 0);
}

/** -----
* @brief     :
* @note      :
* @param     :
* @retval    :
-----*/
bx_err_t user_msg_handle_func(s32 svc, u32 msg, u32 param0, u32 param1 )
{
    /* 消息的来源是btn服务 */
    if( bx_msg_source() == us_btn_id() ) {
        if( msg == USM_BTN_CLICK ) {
            return bx_post( us_led_id(), BXM_TOGGLE, 0, 0);
        }
    }
    return BX_OK;
}

/*===== end of exported function =====*/

/*===== import function =====*/

/*===== end of import function =====*/

/*===== interrupt function =====*/

/*===== end of interrupt function =====*/

```

```
/****** (C) COPYRIGHT BLUEX *****END OF FILE*****/
```

注意在 user_app.c 中，需要使用 bx_post 来提交动作给内核，实现异步启动。

6.2.3 ble

ble 的驱动已经编写完成，直接使用即可，首先需要在 bx_service_ble.h 中，定义几个应用需要的消息，比如：

```
enum bx_msg_ble{
    BXM_BLE_FIRST = BXM_FIRST_USER_MSG,
    BXM_BLE_ADV_START,
    BXM_BLE_ADV_STOP,
    BXM_BLE_SCAN_START,
    BXM_BLE_SCAN_STOP,
    BXM_BLE_DIS_LINK,
    BXM_BLE_NOTIFY,
    BXM_BLE_NOTIFY_ENABLED,
    BXM_BLE_ADVERTISING,
    BXM_BLE_CONNECTED,
    BXM_BLE_DISCONNECT,
    /*****分割线*****/
    BXM_BLE_LED_ON,
    BXM_BLE_LED_OFF,
    BXM_BLE_LED_TOGGLE,
};
```

以上代码，分割线以上的是定义号的消息，分割线以下是自定义的消息，此处定义了3个消息：

```
BXM_BLE_LED_ON,
BXM_BLE_LED_OFF,
BXM_BLE_LED_TOGGLE,
```

然后在 user_profile_task.c 中，修改 gattc_write_req_ind_handler 函数：

```
static int gattc_write_req_ind_handler(ke_msg_id_t const msgid,
                                       struct gattc_write_req_ind *param,
                                       ke_task_id_t const dest_id,
                                       ke_task_id_t const src_id)
{
    // bxsh_logln("user gattc_write_req_ind_handler");
    uint8_t state = ke_state_get(dest_id);
    if(state == USER_PROFILE_IDLE)
    {
        uint8_t status = GAP_ERR_NO_ERROR;
        struct gattc_write_cfm * cfm;

        //Send write response
        cfm = KE_MSG_ALLOC(GATTC_WRITE_CFM, src_id, dest_id, gattc_write_cfm);
        cfm->handle = param->handle;
        cfm->status = status;
        ke_msg_send(cfm);
    }
}
```

```

/*****分割线*****/
if( param->value[0] == 0 ) {
    bx_public( bxs_ble_id(),BXM_BLE_LED_ON,0,0 );
}else if ( param->value[0] == 1 ){
    bx_public( bxs_ble_id(),BXM_BLE_LED_OFF,0,0 );
}else{
    bx_public( bxs_ble_id(),BXM_BLE_LED_TOGGLE,0,0 );
}
}
return (KE_MSG_CONSUMED);
}

```

分割线以下是新增的代码，根据代码可知：

- 当接收到手机的数据为0时，发布 BXM_BLE_LED_ON 的消息
- 当接收到手机的数据为1时，发布 BXM_BLE_LED_OFF 的消息
- 当接收到手机的数据为其他值时，发布 BXM_BLE_LED_TOGGLE 的消息

6.3 应用代码

根据功能可知，通过向指定蓝牙属性写入数据，熄灭/点亮LED，通过板载的button，熄灭/点亮LED，并向指定属性上报数据。

```

/**
 *
 * *****
 * @file    :   main.c
 * @version:
 * @author  :
 * @brief   :
 * *****
 * @attention
 *
 * <h2><center>&copy; Copyright(c) . BLUEX Microelectronics.
 * All rights reserved.</center></h2>
 *
 *
 * *****
 */

/* includes -----*/

#include "bx_kernel.h"
#include "user_app.h"
#include "bx_service_gpio.h"
#include "user_service_led.h"
#include "user_service_button.h"
#include "bx_service_ble.h"
#include "bx_shell.h"
/* private define -----*/

/* private typedef -----*/

/* private variables -----*/

/* exported variables -----*/

```

```

/*===== private function =====*/

/*===== end of private function =====*/


/*===== exported function =====*/
/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

void user_init( void )
{
    bxs_gpio_register();    //注册GPIO服务
    us_led_register();      //注册led服务
    us_btn_register();      //注册button服务
    bxs_ble_register();     //注册ble服务
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

void user_app( void )
{
    /* 启动led和button服务，BLE服务在代码运行后自动启动，无须再次启动 */
    bx_post( us_led_id(), BXM_OPEN, 0, 0 );
    bx_post( us_btn_id(), BXM_OPEN, 0, 0 );

    /* 订阅按键的单击消息，订阅BLE的LED_ON消息，订阅BLE的LED_OFF消息和订阅BLE的
    LED_TOGGLE消息 */
    bx_subscribe(us_btn_id(), USM_BTN_CLICK, 0, 0 );
    bx_subscribe(bxs_ble_id(), BXM_BLE_LED_ON, 0, 0 );
    bx_subscribe(bxs_ble_id(), BXM_BLE_LED_OFF, 0, 0 );
    bx_subscribe(bxs_ble_id(), BXM_BLE_LED_TOGGLE, 0, 0 );
}

/** -----
 * @brief :
 * @note :
 * @param :
 * @retval :
-----*/

bx_err_t user_msg_handle_func(s32 svc, u32 msg, u32 param0, u32 param1 )
{
    static struct ble_notify_data data;
    static u8 value = 0;
    /* 判断消息的来源是按键服务 */
    if( bx_msg_source() == us_btn_id() ) {
        /* 收到按键的单击消息 */
        if( msg == USM_BTN_CLICK ) {
            data.hdl = 36;
            data.len = 1;
            data.data = &value;
            value +=1;
        }
    }
}

```

```

        /* 向BLE服务发送一个NOTIFY消息，消息的参数0是(u32)&data，参数1无含义，可填任意值 */
        bx_post( bxs_ble_id(), BXM_BLE_NOTIFY, (u32)&data, 0);
        /* 向LED服务发送一个翻转消息，参数0和1都无含义，可填任意值 */
        bx_post( us_led_id(), BXM_TOGGLE, 0, 0);
    }
}

/* 判断消息的来源是BLE服务 */
if( bx_msg_source() == bxs_ble_id() ) {
    switch( msg ) {
        /* 收到BLE_LED_ON消息 */
        case BXM_BLE_LED_ON:
            bxsh_logln("BXM_BLE_LED_ON");
            /* 向LED服务发送一个写消息，消息的参数0代表写高还是低，此处1表示写高，参数1无含义，可填任意值 */
            bx_post( us_led_id(), BXM_WRITE, 1, 0);
            break;

            case BXM_BLE_LED_OFF:
                bxsh_logln("BXM_BLE_LED_OFF");
                bx_post( us_led_id(), BXM_WRITE, 0, 0);
                break;

            case BXM_BLE_LED_TOGGLE:
                bxsh_logln("BXM_BLE_LED_TOGGLE");
                bx_post( us_led_id(), BXM_TOGGLE, 0, 0);
                break;

            default:
                break;
    }
}

return BX_OK;
}

/*===== end of exported function =====*/

/*===== import function =====*/

/*===== end of import function =====*/

/*===== interrupt function =====*/

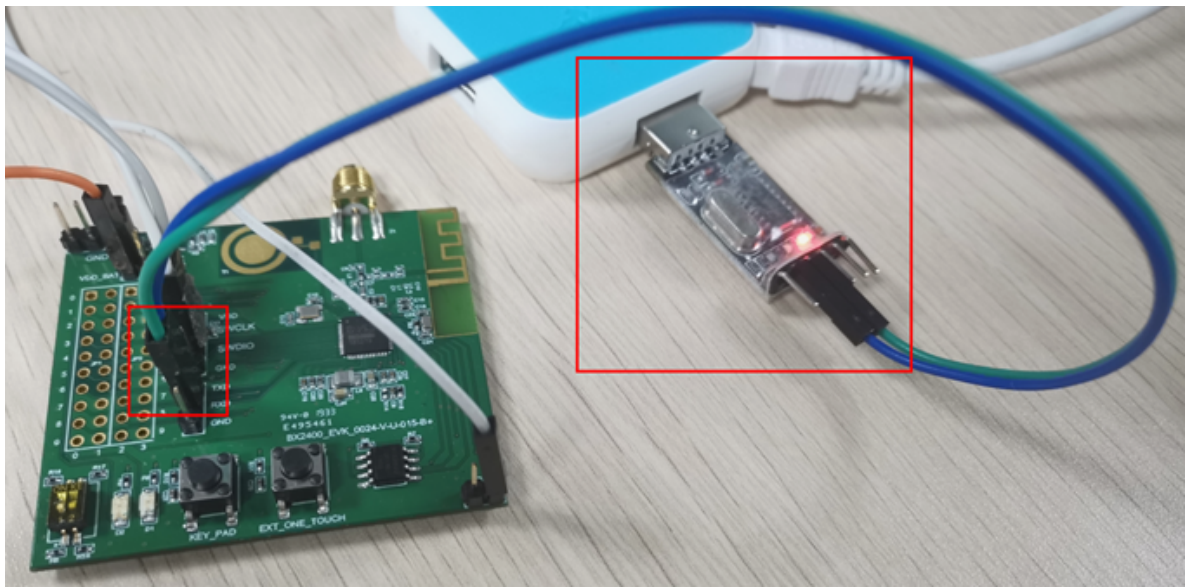
/*===== end of interrupt function =====*/

/***** (C) COPYRIGHT BLUEX *****/END OF FILE*****/

```

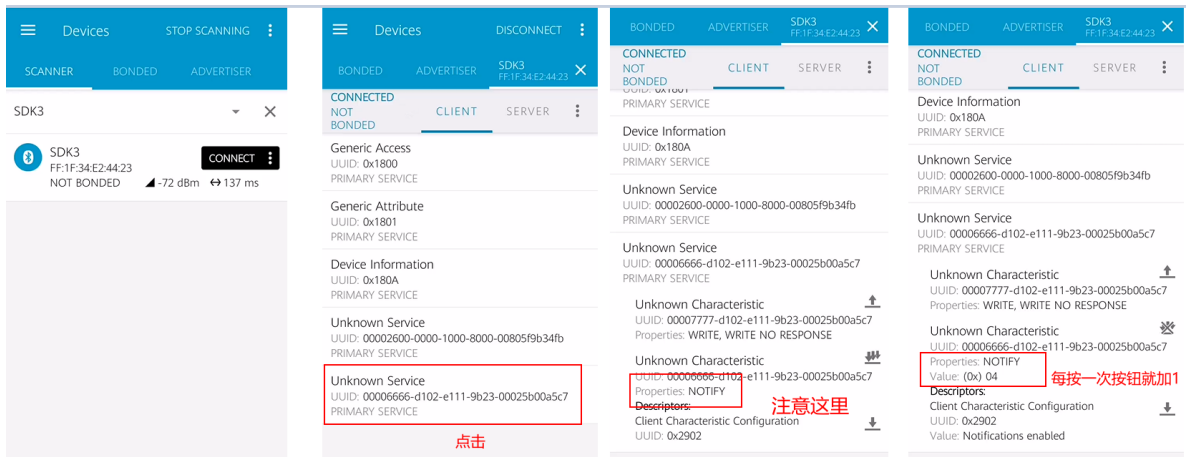
6.4 演示

- 硬件连接

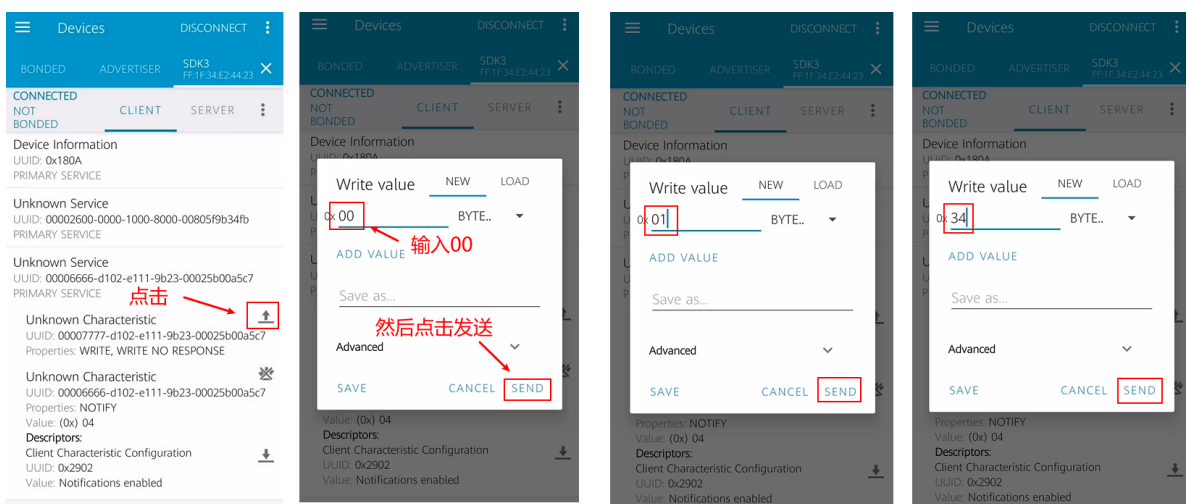


编译以上代码，按图示硬件连接，把代码烧录到设备

- 打开nrf connect 软件，扫描设备，连接之后，按下按钮，可以看到上报的数据加1了，同时LED灯在切换亮灭：



- 输入数据，可以控制LED的亮灭，同时串口输出消息：



串口配置

COM14

115200

8

1

None

GB2312

[23:34:09.578] Welcome To Use Bluex Shell

[23:34:11.137] # 05 08 53 44 4b 33

[23:34:16.584] connected

[23:34:27.549] BXM_BLE_LED_ON

[23:34:33.799] BXM_BLE_LED_OFF

[23:34:38.550] BXM_BLE_LED_TOGGLE

[23:34:45.800] BXM_BLE_LED_TOGGLE

[23:34:52.649] BXM_BLE_LED_TOGGLE

手机发送00

手机发送01

手机发送其它值时