LENZE_ST17H66_SDK_V5.0.6_介绍_V1

目录

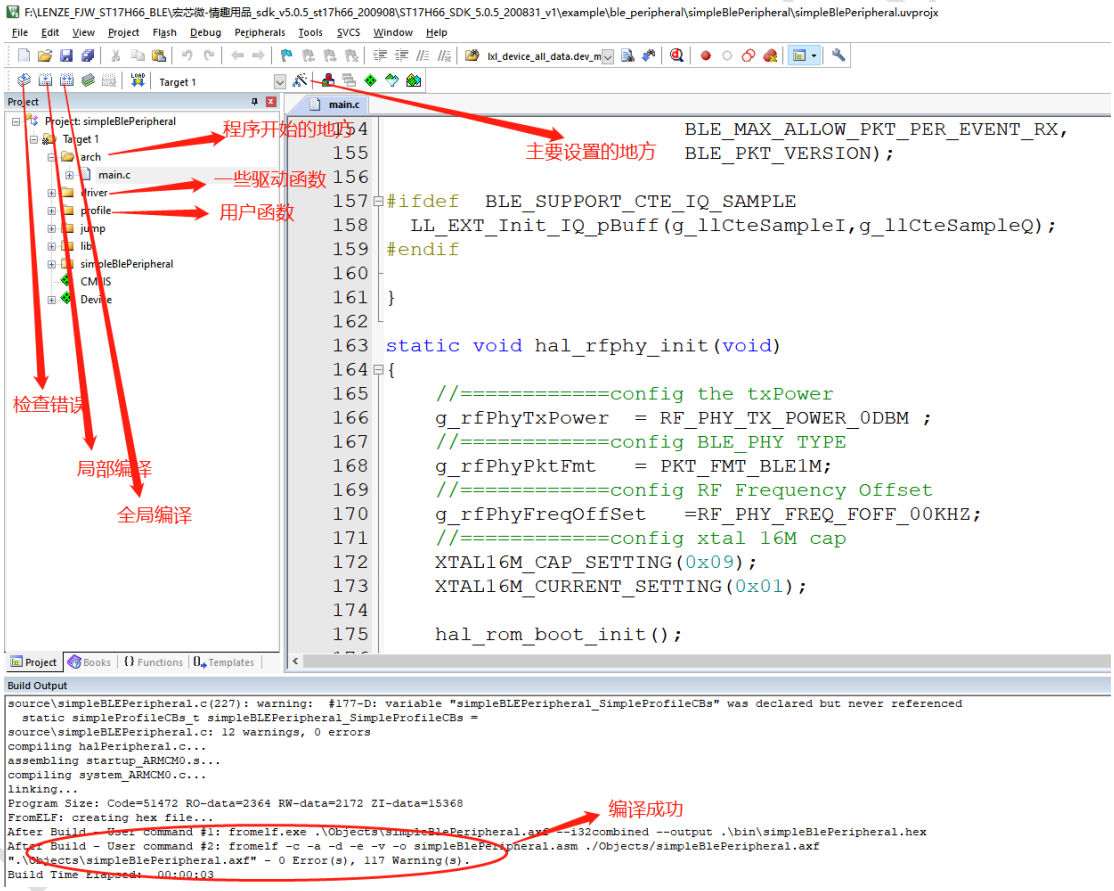
Chapter 1 SDK 开发包快速使用说明

1.1、IDE 开发工具的安装

第一步：先在网络上下载 ide 安装包：“keil5 版本”
第二步：安装软件前先关闭 360 安全软件以及其他安全软件。
第三步：选项都使用默认选项无需更改，安装路径最好不用修改。如果在必要的条件下可以去修改安装路径。
第四步：打开软件，软件设置可以在网上搜索，这些教程网络上比较齐全设置都一样。

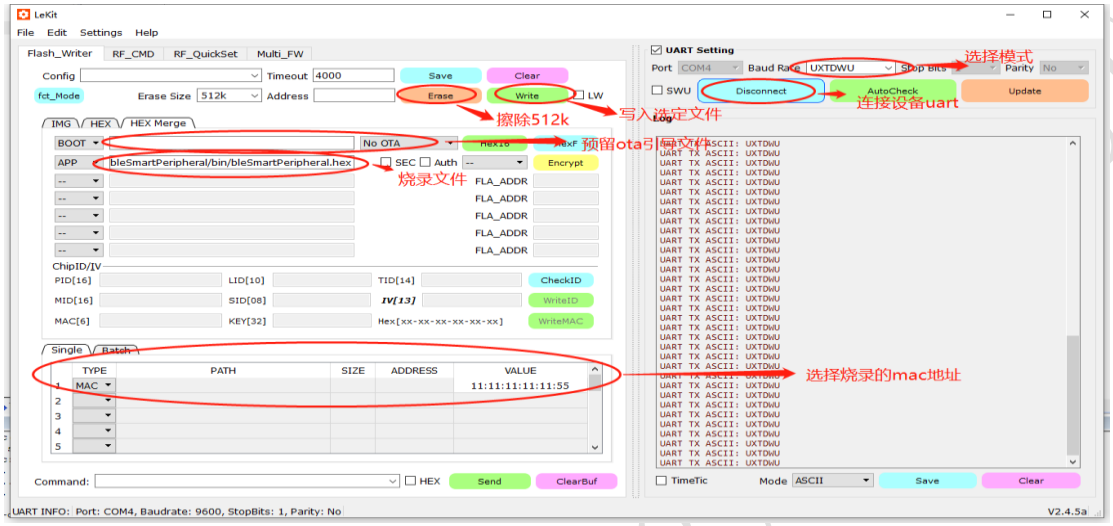
1.2、ST17H66_SDK_V5.06 工程的打开

第一步：打开软件得到工作界面如下：

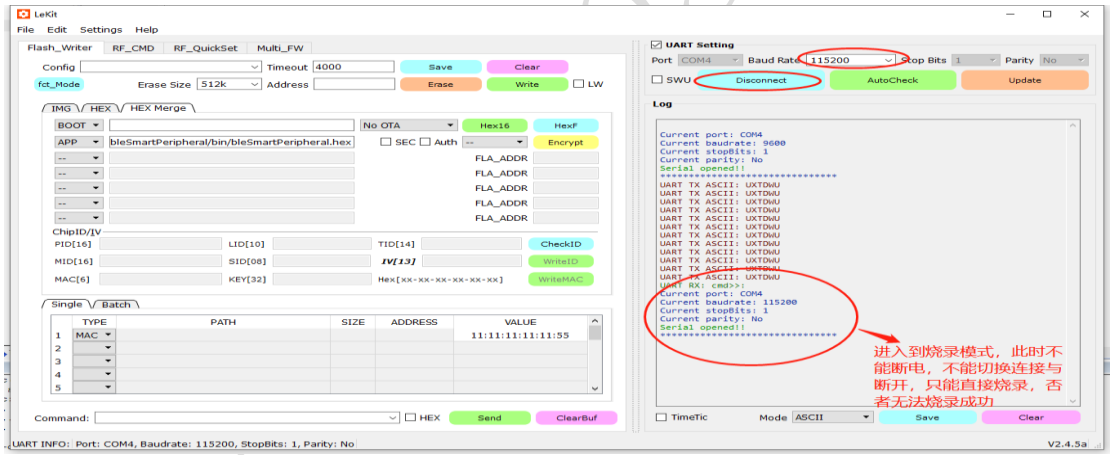


1.3、程序下载说明

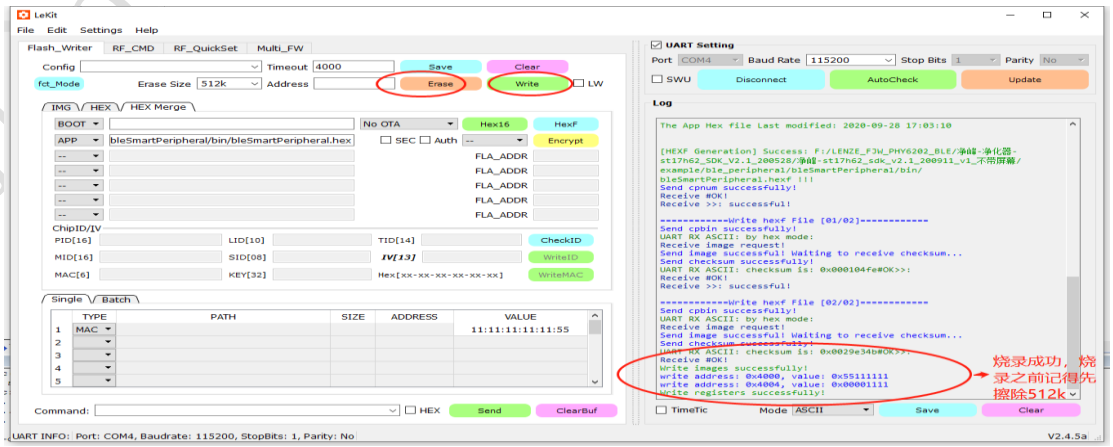
第一步：设置烧录器，按下开发板上“RST”按键，详细说明可以参考，烧录器介绍文档



第二步：进入烧录模式



第三步：开始烧录软件

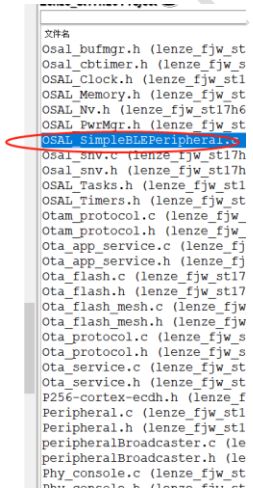


Chapter 2 蓝牙开发说明

1.1、怎么找到蓝牙初始化设置

当前蓝牙初始化函数，不同的 sdk 会以不同的名字命名，请查找。“osalInitTasks(void)”这个函数。

```
    HCI_Init( taskID++ );

#if defined ( OSAL_CBTIMER_NUM_TASKS )
    /* Callback Timer Tasks */
    osal_CbTimerInit( taskID );
    taskID += OSAL_CBTIMER_NUM_TASKS;
#endif

    /* L2CAP Task */
    L2CAP_Init( taskID++ );

    /* SM Task */
    SM_Init( taskID++ );

    /* GAP Task */
    GAP_Init( taskID++ );

    /* GATT Task */
    GATT_Init( taskID++ );

    /* Profiles */
    GAPRole_Init( taskID++ );
    GAPBondMgr_Init( taskID++ );          // 2017-11-15

    GATTServApp_Init( taskID++ );
    /* Application */

    SimpleBLEPeripheral_Init( taskID++ );   当前蓝牙初始化、
    lxl_ui_init_task_func(taskID++) ;
    //lxl_uart_init_task_func(taskID++) ;
    //lxl_sensor_init_task_func(taskID++) ;
    lxl_adc_init_task_func(taskID++) ;
```



1.2、设置广播包

在当前只是设置数组，当前数据在传入底层之前都可以改变或者改变数据后重新初始化底层广播数据，但是当前在广播状态下或者连接状态下不能一直更换广播数据，当前一直调用广播数据会导致无法搜索到广播设备，修改广播包必须在停止广播的状态下修改完成后在开启广播包。

当前更新数据函数：

GAPRole_SetParameter(GAPROLE_SCAN_RSP_DATA, sizeof (scanRspData), scanRspData);

GAPRole_SetParameter(GAPROLE_ADVERT_DATA, sizeof(advertData), advertData);

```
uint8 adv_mac_data[6]= {0x00,0x00,0x00,0x00,0x00,0x00} ;    //设备的mac地址
// GAP - SCAN RSP data (max size = 31 bytes)
uint8 scanRspData[] =
{
    // complete name
    0x0B, 0x09,'H','X','W','-','M','O','T','O','-','1',   //代表当前设备的设备名
};

// advert data for iBeacon
uint8 advertData[] =
{
    0x02,   // length of this data
    GAP_ADTYPE_FLAGS,
    DEFAULT_DISCOVERABLE_MODE | GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED,

    0x03, 0x02, 0xB0, 0xFF,
    0x0E, 0xFF,
    0xFD,                                       //代表当前设备：防丢器
    0x02,                                       //代表当前设备：伦茨公版
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff,         //代表当前设备的mac地址
    0x66,                                       // 设备类型-高
    0x02,                                       // 设备类型-低 01：宏芯微-情趣用品
    0x01,                                       // 版本号高
    0x03,                                       // 版本号低
    0x00,                                       // crc校验和

};
```

1.3、设置通信 uuid

一个数据通信最主要的是他们的 uuid 也代表 master 能不能跟他通信或者当前数据定义在哪儿，怎么传送，在哪儿传送。

```
00429:  *
00430:  * @return  Success or Failure
00431:  */
00432: bStatus_t SimpleProfile_AddService( uint32 services )
00433: {
00434:   uint8 status = SUCCESS;
00435:
00436:   // Initialize Client Characteristic Configuration attributes
00437:   GATTServApp_InitCharCfg( INVALID_CONNHANDLE, simpleProfileChar6Config );
00438:
00439:   // Register with Link DB to receive link status change callback
00440:   VOID linkDB_Register( simpleProfile_HandleConnStatusCB );
00441:
00442:   if ( services & SIMPLEPROFILE_SERVICE )
00443:   {
00444:     // Register GATT attribute list and CBs with GATT Server App
00445:     status = GATTServApp_RegisterService( simpleProfileAttrTbl,
00446:                                           GATT_NUM_ATTRS( simpleProfileAttrTbl ),
00447:                                           &simpleProfileCBs );
00448:   }
00449:
00450:   return ( status );
00451: } ? end SimpleProfile_AddService ?
```

1.4、设置广播以及通信间隔

当前广播间隔设定函数：

```
// Set advertising interval
{
    uint16 advInt = 400;//2400;//1600;//1600;//800;//1600;   // actual time = advInt * 625us

    GAP_SetParamValue( TGAP_LIM_DISC_ADV_INT_MIN, advInt );
    GAP_SetParamValue( TGAP_LIM_DISC_ADV_INT_MAX, advInt );
    GAP_SetParamValue( TGAP_GEN_DISC_ADV_INT_MIN, advInt );
    GAP_SetParamValue( TGAP_GEN_DISC_ADV_INT_MAX, advInt );
}
```

当前通信间隔设定函数：

```
GAPRole_SetParameter( GAPROLE_PARAM_UPDATE_ENABLE, sizeof( uint8 ), &enable_update_request );
GAPRole_SetParameter( GAPROLE_MIN_CONN_INTERVAL, sizeof( uint16 ), &desired_min_interval );
GAPRole_SetParameter( GAPROLE_MAX_CONN_INTERVAL, sizeof( uint16 ), &desired_max_interval );
GAPRole_SetParameter( GAPROLE_SLAVE_LATENCY, sizeof( uint16 ), &desired_slave_latency );
GAPRole_SetParameter( GAPROLE_TIMEOUT_MULTIPLIER, sizeof( uint16 ), &desired_conn_timeout );
}
```

当前广播间隔不能大于 2.5s，广播间隔可以设置成变化的方便 master 找寻设备。当前通信间隔不能大于 1.25s，当前设备大于 1.25 设备会断开连接。当前的低功耗睡眠间隔也是根据当前广播和通信间隔来做的。请注意配合好当前广播或者通信时长。

1.5、读写数据、

读取数据函数是自我定义函数：

```
00364:
00365: /*********************************************************************
00366:  * PROFILE CALLBACKS
00367:  */
00368: // Simple Profile Service Callbacks
00369: CONST gattServiceCBs_t simpleProfileCBs =
00370: {
00371:   simpleProfile_ReadAttrCB,  // Read callback function pointer
00372:   simpleProfile_WriteAttrCB, // Write callback function pointer
00373:   NULL                       // Authorization callback function pointer
00374: };
00375:
00376: /*********************************************************************
```

接收数据函数是用户自定义：

```
00744:  * @param   offset - offset of the first octet to be written
00745:  *
00746:  * @return  Success or Failure
00747:  */
00748: // // TODO: test this function
00749: static bStatus_t simpleProfile_WriteAttrCB( uint16 connHandle, gattAttribute_t *pAttr,
00750:                                             uint8 *pValue, uint8 len, uint16 offset )
00751: {
00752:   bStatus_t status = SUCCESS;
00753:   uint8 notifyApp = 0xFF;
00754:
00755:   // If attribute permissions require authorization to write, return error
00756:   if ( gattPermitAuthorWrite( pAttr->permissions ) )
00757:   {
00758:     // Insufficient authorization
00759:     return ( ATT_ERR_INSUFFICIENT_AUTHOR );
00760:   }
00761:
00762:   if ( pAttr->type.len == ATT_BT_UUID_SIZE )
00763:   {
00764:     // 16-bit UUID
```

写入数据函数是用户定义函数：

```
54:
55: bStatus_t simpleProfile_Notify( uint8 param, uint8 len, void *value )     //发送数据函数
56: {
57:     bStatus_t ret = SUCCESS;
58:     uint16 notfEnable;
59:     switch ( param )
70:     {
71:         case SIMPLEPROFILE_CHAR1:
72:           notfEnable = GATTServApp_ReadCharCfg( 0, simpleProfileChar6Config );
73:
74:           // If notifications enabled
75:           if ( notfEnable & GATT_CLIENT_CFG_NOTIFY )
76:           {
77:             VOID osal_memcpy( simpleProfileChar1, value, len );
78:             d2phone_length = len  ;
79:               //for SAR test copy the seqNum in pkt tail
80:               //simpleProfileChar1[ATT_GetCurrentMTUSize(0)-4-1]=simpleProfileChar1[0];
81:               //simpleProfileChar1[ATT_GetCurrentMTUSize(0)-3-1]=simpleProfileChar1[1];
82:
83:               ret=GATTServApp_ProcessCharCfg( simpleProfileChar6Config, simpleProfileChar1, FAL
84:                                               simpleProfileAttrTbl, GATT_NUM_ATTRS( simpleProfileAttrTb
85:                                               INVALID_TASK_ID );
86:
87:           }else{
88:               ret = bleNotReady;
89:           }
90:           break;
91: #if 0
```

读取函数可以去查看当前传入数据的结构体内部数据，对比查看当前数据。写入数据的函数需要注意的就是第一个参数，这个参数是根据服务列表获取到，当前传输数据的服务在列表中是第几行就填当前的行数。

Chapter 3 mcu 开发说明

1.1、设置 io

```
hal_gpio_pin_init(MY_GPIO_KEY_1,IE) ;                    //设置为输入
hal_gpio_pull_set(MY_GPIO_KEY_1,STRONG_PULL_UP) ;        //设置当前 io 的上下拉
hal_gpioin_enable(MY_GPIO_KEY_1) ;                       //设置 io 唤醒，请在中断中关闭唤醒
hal_gpioin_register(MY_GPIO_KEY_1, lxl_pin_event_handler_key, lxl_pin_event_handler_key);
//注册按键上升沿和下降沿回调函数

hal_gpio_fmux(MY_GPIO_BACK_LED_1,Bit_ENABLE) ;           //设置 gpio 为普通 io 状态
hal_gpio_pin_init(MY_GPIO_BACK_LED_1,OEN) ;              //设置为输出
hal_gpio_write(MY_GPIO_BACK_LED_1,0) ;                   //设置为输出为 0

hal_gpio_cfg_analog_io(MY_GPIO_ADC_MOTO_1, Bit_ENABLE); //设置为 adc 状态，模拟口
```

1.2、设置唤醒

当前所有 io 口都有唤醒和中断以及上下拉功能

```
// hal_gpio_fmux(MY_GPIO_KEY_1,Bit_DISABLE) ;              //上电设置为gpio口
 hal_gpio_pin_init(MY_GPIO_KEY_1,IE) ;                   //设置为输入
 //hal_gpio_cfg_analog_io(MY_GPIO_KEY_1,Bit_DISABLE) ;
 hal_gpio_pull_set(MY_GPIO_KEY_1,STRONG_PULL_UP) ;       //设置当前io的上下拉
 hal_gpioin_enable(MY_GPIO_KEY_1) ;
 hal_gpioin_register(MY_GPIO_KEY_1, lxl_pin_event_handler_key, lxl_pin_event_handler_key);

 hal_gpio_pin_init(MY_GPIO_ADC_5V_TEST,IE) ;                    //设置为输入
 hal_gpio_pull_set(MY_GPIO_ADC_5V_TEST,PULL_DOWN) ;      //设置当前io的上下拉
 hal_gpioin_enable(MY_GPIO_ADC_5V_TEST) ;
 hal_gpioin_register(MY_GPIO_ADC_5V_TEST, lxl_pin_event_handler_key, lxl_pin_event_handler_k
```

```
void lxl_pin_event_handler_key(GPIO_Pin_e pin,IO_Wakeup_Pol_e type)
{
    static u8 key_dowm_tick_temp = 0 ;
    switch(pin)
    {
        case MY_GPIO_KEY_1:    //按键检测
            if(type == NEGEDGE){
                lxl_key_all_data.key_dowm_tick_0 |= BIT(0);
                if(lxl_device_all_data.dev_job_tick){
                    lxl_device_all_data.dev_sleep_tick = 1 ;
                    osal_start_timerEx(lxl_ui_count_TaskID, MY_UI_10MS_EVENT_TICK, 10);
                }else {
                    osal_start_timerEx(lxl_ui_count_TaskID, MY_UI_POWER_EVENT_TICK, 10);
                }
                LOG("4:\n") ;
            }else {
                lxl_key_all_data.key_dowm_tick_0 &= (!BIT(0)) ;
                LOG("5:\n") ;
            }
            osal_start_timerEx(lxl_key_count_TaskID, MY_KEY_1_EVENT_TICK, 10);
        break;
        case MY_GPIO_ADC_5V_TEST:    //充电检测
            if(type == NEGEDGE){
                osal_start_timerEx(lxl_ui_count_TaskID, MY_UI_POWER_EVENT_TICK, 10);
            }else {
                //进入充电状态，其他关闭只闪烁灯，以及充满判断
            }
        break;
        default:
            LOG("MY_key_dowm_0000:\n") ;
        break;
```

在这个版本之上的开关机比较难设置，请参考示例代码。

1、在程序上请保证开启低功耗不管使用还是不使用，必须要开启低功耗，不需要低功耗时，请在程序上关闭低功耗，关机时开启低功耗

2、关机前一定要注册低功耗函数标注开启还是关闭低功耗

```
    hal_pwrmgr_register(MOD_USR8, NULL, NULL);          //进入低功耗函数
    hal_pwrmgr_lock(MOD_USR8) ;                         //退出低功耗

    lxl_proc_power_onoff(0,50) ;                        //短按开机
    lxl_led_enter_mode_1(1) ;
```

3、全局需要一个开关机标志位，标志位为零时一定要关闭任何事件回调，包括初始化的事件回调。

```
{
    lxl_ui_count_TaskID = task_id;
    my_gpio_all_init_func() ;
    if(lxl_device_all_data.dev_job_tick){          → 开机才能回调，避免出错，而且所有事件一样
        osal_start_timerEx(lxl_ui_count_TaskID, MY_UI_10MS_EVENT_TICK, 20);    //跳入UI处理第一次
        LOG("MY_INIT_GPIO_111111:\n") ;
    }
}
```

4、关机，只需要关闭所有事件的回调即可。无需去调用其他关机函数。

```
    hal_pwrmgr_unlock(MOD_USR8) ;     //进入低功耗
    //pwroff_cfg_t temp_gpio_wakeup[2] ;
    //temp_gpio_wakeup[0].pin = MY_GPIO_KEY_1 ;
    //temp_gpio_wakeup[0].type = NEGEDGE ;
    //temp_gpio_wakeup[1].pin = MY_GPIO_ADC_5V_TEST ;
    //temp_gpio_wakeup[1].type = POSEDGE ;
    if(data_tick){
        while(!hal_gpio_read(MY_GPIO_KEY_1))
                WaitUs(10*1000);
    }
    #if 0
    hal_pwrmgr_poweroff(temp_gpio_wakeup,2) ;
    #else
    uint8 initial_advertising_enable = FALSE;
    GAPRole_SetParameter( GAPROLE_ADVERT_ENABLED, sizeof( uint8 ), &initial_advertising_enable
    u32 data_taskid_tick = 1 ;
    for(u8 i=0; i<15; i++){
        //LOG("TaskID_STOP_DATA: %d\n",data_taskid_tick) ;
        osal_stop_timerEx(lxl_ui_count_TaskID,data_taskid_tick) ;
        osal_stop_timerEx(lxl_sensor_count_TaskID,data_taskid_tick) ;
        osal_stop_timerEx(lxl_uart_count_TaskID,data_taskid_tick) ;
        osal_stop_timerEx(lxl_key_count_TaskID,data_taskid_tick) ;
        osal_stop_timerEx(lxl_adc_count_TaskID,data_taskid_tick) ;
        data_taskid_tick = data_taskid_tick<<1 ;
    }
    #endif
```

5、每次唤醒都会进入按键中断，在按键中断函数中判断是否开机。

```
void lxl_pin_event_handler_key(GPIO_Pin_e pin,IO_Wakeup_Pol_e type)
{
    static u8 key_dowm_tick_temp = 0 ;
    switch(pin)
    {
        case MY_GPIO_KEY_1:    //按键检测
            if(type == NEGEDGE){
                lxl_key_all_data.key_dowm_tick_0 |= BIT(0);
                if(lxl_device_all_data.dev_job_tick){          → 判断当前是否是开机
                    lxl_device_all_data.dev_sleep_tick = 1 ;
                    osal_start_timerEx(lxl_ui_count_TaskID, MY_UI_10MS_EVENT_TICK, 10);
                }else {
                    osal_start_timerEx(lxl_ui_count_TaskID, MY_UI_POWER_EVENT_TICK, 10);
                }                                              ↓ 调用重启函数
                LOG("4:\n") ;
            }else {
                lxl_key_all_data.key_dowm_tick_0 &= (!BIT(0)) ;
                LOG("5:\n") ;
            }
            osal_start_timerEx(lxl_key_count_TaskID, MY_KEY_1_EVENT_TICK, 10);
        break;
        case MY_GPIO_ADC_5V_TEST:    //充电检测
            if(type == NEGEDGE){
                osal_start_timerEx(lxl_ui_count_TaskID, MY_UI_POWER_EVENT_TICK, 10);
            }else {
                //进入充电状态，其他关闭只闪烁灯，以及充满判断
            }
        break;
        default:
            LOG("MY_key_dowm_0000:\n") ;
        break;
```

6、从新启动去判断开关机条件。

```
if(events&MY_UI_POWER_EVENT_TICK){
    if(lxl_device_all_data.dev_job_tick == 0){    //关机状态下,重新初始化io，以及时间状态
        NVIC_SystemReset() ;
    }
    return (events ^ MY_UI_POWER_EVENT_TICK);
}
```

7、可以在当前步骤上加 log 查看一次，跟着 log 看一遍。思路会比较清晰。

1.3、设置 pwm

当前 pwm 口可以随意映射。

```
void lxl_pwm_timeout_handle(void)
{
      lxl_pwm_value_light[0] = 0;
      lxl_pwm_value_light[1] = 0;
      lxl_pwm_value_light[2] = 0;
      hal_pwm_close_channel(PWM_CH0);
      hal_pwm_destroy(PWM_CH0);
      hal_pwm_close_channel(PWM_CH1);
      hal_pwm_destroy(PWM_CH1);
      //hal_pwm_close_channel(PWM_CH2);
      //hal_pwm_destroy(PWM_CH2);
      hal_pwm_stop();
}

int lxl_pwm_ctrl(uint8_t ch, uint16_t value)
{
    if(ch >2 || (value > MY_PWM_CNTTOP_VALUE))
        return PPlus_ERR_INVALID_PARAM;
    switch (ch)
    {
        case 0:
            if(value==0){
                hal_gpio_write(MY_GPIO_PWM_M0_1, 0);
            }else{
                lxl_pwm_on(ch) ;
                hal_pwm_open_channel(PWM_CH0, MY_GPIO_PWM_M0_1);
                lxl_pwm_value_light[0] = value;
            }
            hal_pwm_set_count_val(PWM_CH0, lxl_pwm_value_light[0], MY_PWM_CNTTOP_VALUE);
        break;
        case 1:
            if(value==0){
                hal_gpio_write(MY_GPIO_BACK_LED_1, 0);
            }else{
                lxl_pwm_on(ch) ;
                hal_pwm_open_channel(PWM_CH1, MY_GPIO_BACK_LED_1);
                lxl_pwm_value_light[0] = value;
            }
            hal_pwm_set_count_val(PWM_CH1, lxl_pwm_value_light[1], 255);
        break;
        #if 0

int lxl_pwm_on(uint8_t ch)
{
    if(ch >3)
        return PPlus_ERR_INVALID_PARAM;
    lxl_pwm_value_light[ch] = (uint16_t)(MY_PWM_CNTTOP_VALUE>>1);
    switch(ch){
        case 0:
            hal_pwm_init(PWM_CH0, PWM_CLK_DIV_8, PWM_CNT_UP, PWM_POLARITY_RISING);    //16M/8 = 2M
            //hal_pwm_set_count_val(PWM_CH0, lxl_pwm_value_light[0], MY_PWM_CNTTOP_VALUE);  // 1
            hal_pwm_open_channel(PWM_CH0, MY_GPIO_PWM_M0_1);
        break;
        case 1:
            hal_pwm_init(PWM_CH1, PWM_CLK_DIV_8, PWM_CNT_UP, PWM_POLARITY_RISING);
            //hal_pwm_set_count_val(PWM_CH1, lxl_pwm_value_light[1], 255);
            hal_pwm_open_channel(PWM_CH1, 255);
        break;
        #if 0

int lxl_pwm_off(uint8_t ch)
{
    if(ch >2)
        return PPlus_ERR_INVALID_PARAM;

    lxl_pwm_value_light[ch] = 0;
    switch(ch){
        case 0:
            lxl_pwm_value_light[0] = 0;
            hal_pwm_close_channel(PWM_CH0);
            hal_pwm_destroy(PWM_CH0);
            hal_gpio_write(MY_GPIO_PWM_M0_1, 0);
        break;
        case 1:
            lxl_pwm_value_light[1] = 0;
            hal_pwm_close_channel(PWM_CH1);
            hal_pwm_destroy(PWM_CH1);
            hal_gpio_write(MY_GPIO_BACK_LED_1, 0);
        break;
        #if 0

            }
            //LOG("value: %d\n",lxl_BACK_LED_1_value) ;
            lxl_pwm_ctrl(1,lxl_BACK_LED_1_value) ;
        } ? end else ?
    } ? end if onOff ?
    else {
        lxl_pwm_off(1) ;
        hal_gpio_write(MY_GPIO_BACK_LED_1, 0);
    }
```

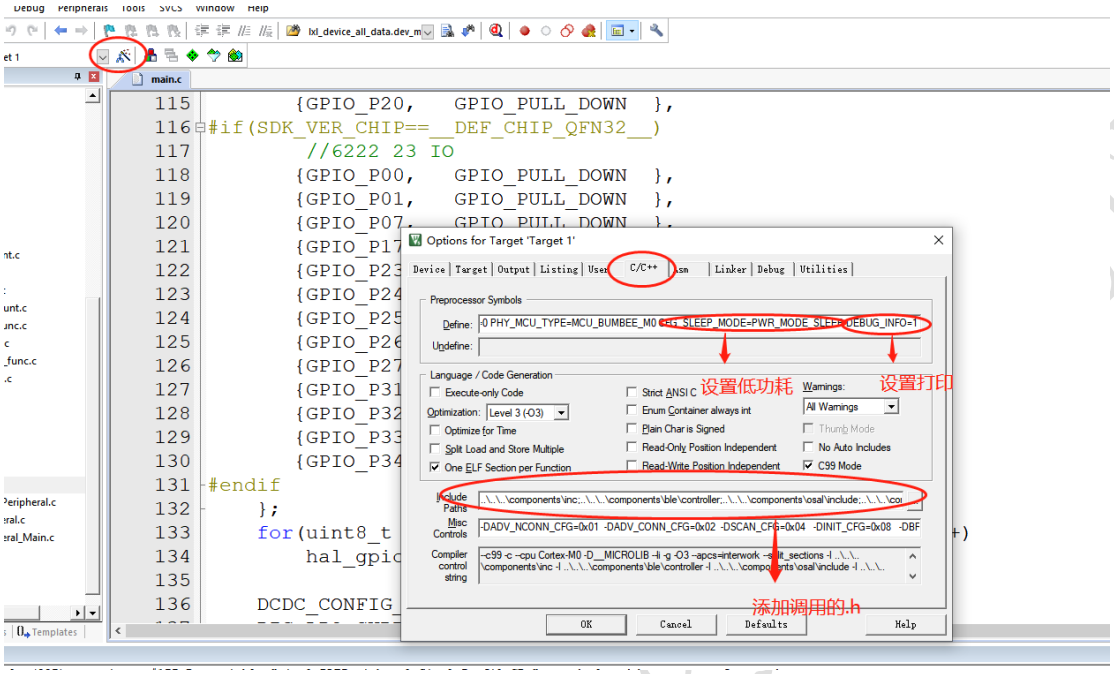
1.4、设置 adc

```
adc_Cfg_t normal_adc_cfg = {
      //.channel = ADC_BIT(ADC_CH1N_P11)|ADC_BIT(ADC_CH1P_P12)|ADC_BIT(ADC_CH2N_P13)|ADC_BIT(ADC
      .channel = ADC_BIT(ADC_CH3P_P20)|ADC_BIT(ADC_CH3N_P15),
      .is_continue_mode = FALSE,
      .is_differential_mode = 0x00,
      .is_high_resolution = 0xff,
};


int normal_adc_init(void)
{
    int ret;
    ret = hal_adc_config_channel(normal_adc_cfg, normal_adc_handle_evt);
    if(ret)
    return ret;
    return PPlus_SUCCESS;
}


void normal_adc_handle_evt(adc_Evt_t* pev)
{
    if(pev->type == HAL_ADC_EVT_DATA){
        if(pev->ch==ADC_CH3N_P15)
        {
            is_high_resolution = (normal_adc_cfg.is_high_resolution & ADC_BIT(ADC_CH3N_P15))?TR
            is_differential_mode = (normal_adc_cfg.is_differential_mode & ADC_BIT(ADC_CH3N_P15)
            adc_result = hal_adc_value_cal((adc_CH_t)(ADC_CH3N_P15),pev->data, pev->size, is_hi

            int ntc_volt =(int)(adc_result*1000);
            uint32 Ressis=(uint32)(((adc_result*1000)*470*1000/(3300-ntc_volt)-1300)/10);
            check_flag|=0x01;

            LOG("NTC_Temp=%d \n",lxl_adc_all_data.adc_ntc_value);
        }else if(pev->ch==ADC_CH3P_P20){
            is_high_resolution = (normal_adc_cfg.is_high_resolution & ADC_BIT(ADC_CH3P_P20))?TR
            is_differential_mode = (normal_adc_cfg.is_differential_mode & ADC_BIT(ADC_CH3P_P20)
            adc_result = hal_adc_value_cal((adc_CH_t)(ADC_CH3P_P20),pev->data, pev->size, is_hi
            lxl_adc_all_data.adc_diff_value = (int)(adc_result*1000) ;

            check_flag|=0x02;
            LOG("bat Volt=%d  ",lxl_adc_all_data.adc_diff_value);
            //osal_start_timerEx(uint8 task_id, uint16 event_id, uint32 timeout_value)
        }

        if(check_flag==0x03){    //计算adc数值
            check_flag=0;
            osal_start_timerEx(lxl_adc_count_TaskID, MY_ADC_22_EVENT_TICK, 10);
        }
    } ? end if pev->type==HAL_ADC_EV... ?
```

```
if(MY_ADC_11_EVENT_TICK&events){          //开始
    hal_adc_init() ;
    normal_adc_init() ;
    hal_adc_start() ;
    //LOG("8") ;

    osal_start_timerEx(lxl_adc_count_TaskID, MY_ADC_11_EVENT_TICK, 3*1000);
    return (events ^ MY_ADC_11_EVENT_TICK);
}
if(MY_ADC_22_EVENT_TICK&events){          //停止
    hal_adc_stop() ;

    return (events ^ MY_ADC_22_EVENT_TICK);
}
```

Chapter 4 低功耗开发说明

1.1、工作设置




```
hal_pwrmgr_register(MOD_USR8, NULL, NULL);      //进入低功耗函数初始化调用一次
hal_pwrmgr_lock(MOD_USR8) ;                     //退出低功耗
hal_pwrmgr_unlock(MOD_USR8) ;                   //进入低功耗
if(lxl_device_all_data.dev_sleep_tick==0){
    hal_pwrmgr_unlock(MOD_USR8) ;      //进入低功耗
}else{
    hal_pwrmgr_lock(MOD_USR8) ;        //退出低功耗
}
```

当前低功耗状态 ble、uart、pwm 等等一些底层函数中的已经设置 ok，当前低功耗状态是串联关系一个不进入低功耗状态，整个系统都处于全速工作，所以用户只需要定义一个低功耗回调来控制整个系统的低功耗状态即可。