旧的 SDK 可用的函数参考

```
#include "../../proj/tl_common.h"
#include "../../proj_lib/blt_ll/blt_ll.h"
#include "../../proj/mcu_spec/gpio_17H26.h"
#include "ui.h"

#if 1
////////////////////////////////////////////////////////////////////////////////
//  本地变量 Local Variables 用于显示系统当前状态的 led 以及 buzzer 结构定义
//
////////////////////////////////////////////////////////////////////////////////
ui_type_t buzzer_ui_buffer_MS[] = {
        //off           on      count   next mode
        {{0,            0},     0,      0}, //normal mode
        {{40,           200},    0x02,  0}, // power on 1S/50ms,adjust power mode
        {{180,          60},    0xff,   0}, // alert mode
        {{0,            60},    0x01,   0}, // button mode
};//由于控制 buzzer 的 GPIO 口开启了 PWM 功能，当用到 alert mode 时，buzzer 设备循环 0xff
次执行：每经过 180ms 时间的停顿，使能 pwm 输出从而使得 buzzer 以某一设定频率响持续 60ms。
当用到 button mode 时，设备执行一次：使能 pwm 输出，令 buzzer 以一设定频率响持续 60ms
ui_type_t led_ui_buffer_MS[] = {
        //off           on      count   next mode
        {{0,            0},     0,      0},         //power off  and connect state
        {{0,            2000},  0x01,   2},     //power on:0ms /2S: 1time
        {{2950,         60},    0xff,   2},     //adv : 3S  /50ms  :2950 + 50
        {{0,            60},    1,      2},         //button
        {{180,          60},    0xff,   0},     // alert mode
};//由于控制 led 的 GPIO 口开启了普通 GPIO 功能，当用到 alert mode 时，led 设备循环 0xff//
次执行：每经过 180ms 时间的停顿，开启对应 GPIO 输出从而使得 LED 亮灯 60ms。
//当用到 button mode 时，设备执行一次：使能 pwm 输出，令 LED 亮灯 60ms.

ui_param_t led_param ={led_ui_buffer_MS};
ui_param_t buzzer_param = {buzzer_ui_buffer_MS};

u8  ui_suspend_en = 0;
////////////////////////////////////////////////////////////////////////////////
//                      External Variable  外部变量
////////////////////////////////////////////////////////////////////////////////
extern u8   blt_suspend_mask;
extern u8   power_mode;//0->power off ; 1->power on
extern u8   is_power_switch_exist;
////////////////////////////////////////////////////////////////////////////////
///////
```

```
//                       常用函数 led_beep() 用于控制 LED 对应亮灭状态
////////////////////////////////////////////////////////////////////////////////
static inline void led_beep(u8 onOff){
#if LED_USE_PWM
    write_reg8 (0x780, (buzzer_param.cur_state << 1) | onOff);
    led_param.cur_state = onOff;
#else
    gpio_write(GPIO_GP7,onOff);
#endif
}
////////////////////////////////////////////////////////////////////////////////
//                       常用函数 buzzer_beep() 用于控制 Buzzer 对应启停状态
////////////////////////////////////////////////////////////////////////////////
static inline void buzzer_beep(u8 onOff){
#if LED_USE_PWM
    write_reg8 (0x780, led_param.cur_state | (onOff << 1));
    buzzer_param.cur_state = onOff;
#else
    if(onOff){
        write_reg8(0x780,0x02);
    }else{
        write_reg8(0x780,0x00);
    }
#endif
}

////////////////////////////////////////////////////////////////////////////////
////////        常用函数    calculate next tick and change led/buzzer mode
/////// 用于控制 led,或 buzzer 外设对应启停状态切换，其返回值 next_wakeup_tick 用于
////PM 电源控制的省电模式
////////////////////////////////////////////////////////////////////////////////
 u32 calc_led_ui(ui_param_t *ui_bl)
{
    if(ui_bl->cur_mode == 0) return 0;

    if(ui_bl->next_wakeup_tick - (clock_time() + 2*CLOCK_SYS_CLOCK_1MS) <
BIT(30)){
        return ui_bl->next_wakeup_tick;
    }

    if(ui_bl->cur_state && ui_bl->cur_cnt && ui_bl->cur_cnt != 0xff){//!=0 !=0xff
==on
        ui_bl->cur_cnt --;
    }
    if(ui_bl->cur_cnt == 0 ){
        ui_bl->cur_mode = ui_bl->ui_type[ui_bl->cur_mode].next_mode;
```

```
        ui_bl->cur_cnt = ui_bl->ui_type[ui_bl->cur_mode].offOn_cnt;
    }
    ui_bl->cur_state = ui_bl->cur_state ? 0: 1;
    ui_bl->next_wakeup_tick = clock_time() +
ui_bl->ui_type[ui_bl->cur_mode].offOn_Ms[ui_bl->cur_state] *
CLOCK_SYS_CLOCK_1MS;

    return ui_bl->next_wakeup_tick; //   calculate next tick

}

////////////////////////////////////////////////////////////////////////////////
//       常用函数 buzzer_led_ui() 用于控制外设 Buzzer 以及 led 对应的启停状态
//       其返回值 next_wakeup_tick 用于 PM 电源控制的睡眠时间判定
////////////////////////////////////////////////////////////////////////////////

u32 buzzer_led_ui ( )
{
    u32 next_led_wakeup_tick = calc_led_ui(&led_param);

    u32 next_buzzer_wakeup_tick = calc_led_ui(&buzzer_param);

    led_beep(led_param.cur_state);
    buzzer_beep(buzzer_param.cur_state);
// return the min of(next_buzzer_wakeup_tick,next_led_wakeup_tick except 0) or
//return 0;
#if LED_USE_PWM
    if(buzzer_param.cur_state || led_param.cur_state){
#else
    if(buzzer_param.cur_state){
#endif
        blt_suspend_mask = 0;
    }else{
        blt_suspend_mask = ui_suspend_en;
    }
    if((next_led_wakeup_tick - next_buzzer_wakeup_tick < BIT(30)  &&
next_buzzer_wakeup_tick) || next_led_wakeup_tick == 0){
        return next_buzzer_wakeup_tick;
    }
    return next_led_wakeup_tick;
}
////////////////////////////////////////////////////////////////////////////////
// ui_enter_mode 用于控制外设 LED 或 BUZZER 对应要进入的模式,并根据结构体设定的
on,off,count,next_mode 来实现具体的启停特征
////////////////////////////////////////////////////////////////////////////////
```

```
void ui_enter_mode(ui_param_t *ui_param,u8 mode){
    if(ui_param->cur_mode == 1 || ui_param->cur_mode == mode){
        return;
    }
    ui_param->cur_cnt = ui_param->ui_type[mode].offOn_cnt;
    ui_param->cur_mode = mode;
    ui_param->cur_state = 0;
    ui_param->next_wakeup_tick = clock_time();
}
//////////////////////////////////////////////////////////////////////////////
// led_enter_mode 用于控制 LED 要进入的模式，和对应的数据结构
//////////////////////////////////////////////////////////////////////////////
void led_enter_mode(u8 mode){
    ui_enter_mode(&led_param, mode);
    return;
}
//////////////////////////////////////////////////////////////////////////////
// buzzer_enter_mode 用于控制 buzzer 要进入的模式，和对应的数据结构
//////////////////////////////////////////////////////////////////////////////
void buzzer_enter_mode(u8 mode){
    ui_enter_mode(&buzzer_param, mode);
    return;
}
//////////////////////////////////////////////////////////////////////////////
// test_init_tmp 用于完成初始化系统输入输出端口的功能
//////////////////////////////////////////////////////////////////////////////
void test_init_tmp()
{
    blt_set_wakeup_source(PM_WAKEUP_CORE);//初始化系统为硬件唤醒
    write_reg8 ( 0x597, 0x02); //GPIO 初始化时指定 GPIO17 作为中断唤醒源
    write_reg8 ( 0x594, 0x02);//GPIO 初始化时指定 GPIO17 为低电平输入时唤醒
    gpio_set_interrupt(GPIO_GP18, 0);  // rising edge
    /*set gpio18 wakeup deepsleep*/
    analog_write (0x16, 0x01);//set enable gp17
    analog_write (0x14, analog_read (0x14) & 0xef);// set polarity

    gpio_set_func(GPIO_GP7, AS_GPIO);
    gpio_set_output_en(GPIO_GP7, 1);
    gpio_set_input_en(GPIO_GP7, 0);

    gpio_set_func(GPIO_GP22, AS_GPIO);
    gpio_set_output_en(GPIO_GP22, 0);
    gpio_set_input_en(GPIO_GP22, 1);

    gpio_set_func(GPIO_GP18, AS_GPIO);
    gpio_set_output_en(GPIO_GP18, 0);
```

```
    gpio_set_input_en(GPIO_GP18, 1);

    gpio_set_func(GPIO_GP17, AS_GPIO);
    gpio_set_output_en(GPIO_GP17, 0);
    gpio_set_input_en(GPIO_GP17, 1);
}
////////////////////////////////////////////////////////////////////////////////
//函数 task_connection_terminated
// This event is returned once connection is terminated
//ex:notify application connection terminated; reset connection para
// callback format  task_connection_terminated(&conn_terminate);
//当连接终止的时候，SDK 会自动调用这个函数，用户可以根据需要传入一个参数单独调用它，亦可以在其中加入自定义的 led ,buzzer 以及定时，来完成项目需要的状态显示功能
////////////////////////////////////////////////////////////////////////////////
void task_connection_terminated(rf_packet_connect_t* p){
    u8 is_terminate = *(u8*)p;
    if ( is_terminate == 1){// master send terminate
        proshutter_disconnect_state = 0;
        //start led mode
        buzzer_enter_mode(2);
        led_enter_mode(2);

        //change led next mode
        led_ui_buffer_MS[3].next_mode = 2;
        led_ui_buffer_MS[1].next_mode = 2;
        led_ui_buffer_MS[4].next_mode = 2;

        //change alert mode: alert frequency and button next mode
        buzzer_ui_buffer_MS[2].offOn_Ms[0] = 1000;
        buzzer_ui_buffer_MS[3].next_mode = 0;//next mode = 0 while disconnected
    }else{
        proshutter_disconnect_state = 1;
    }

    //clear and init flag
    selfie_adv_mode_start_tick =  clock_time ();
    blt_is_reconnection = 0;
//  button_need_send_pkt = 0;
//  shutter_mode_start_tick = 0;
    return;
}
////////////////////////////////////////////////////////////////////////////////
// proc_power_onoff 用于控制芯片开启或关闭
////////////////////////////////////////////////////////////////////////////////
 void proc_power_onoff(u8 cur_state,u32 poweron_start_tick)
{
```

```
//未使用开关开启按键时， 默认设备开机即时启动
    if(is_power_switch_exist == 0){
        power_mode = Mode_Power_On;
        return;
    }
//当前设备处于关机状态时，判断离最近一次调用本函数的时间差是否超过 2.4s，若未超过，则读取 GPIO 按键状态以确定 PM 电源管理模式是定时唤醒还是硬件唤醒。若时间差超过 2.4s，则开机
    if(!cur_state){//power off state
        while(!clock_time_exceed(poweron_start_tick,2400*1000)){
            if(gpio_read(GPIO_GP18)){
//按键按下时，芯片进入 suspend ，并每 1s 定时醒来一次
                blt_sleep_wakeup(0,PM_WAKEUP_TIMER,clock_time() +
100*CLOCK_SYS_CLOCK_1MS);
            }else{
//无按键按下时，芯片进入 deepsleep 模式并开启按键唤醒使能,需在 GPIO 初始化时指定一个特定
//IO 口作为唤醒源
                blt_sleep_wakeup(1,PM_WAKEUP_PAD,0);
            }
        }
        power_mode = Mode_Power_On;
    }
//当前设备处于开机状态时，判断离最近一次调用本函数的时间差是否超过 2.4s，若超过 2.4s，则关机
      else{
        if(clock_time_exceed(poweron_start_tick,2400*1000)){
            power_mode = Mode_Power_Off;
        }
    }
}
////////////////////////////////////////////////////////////////////////////////
// powerOff handle 用于控制芯片进入关闭状态时，buzzer 响一声，led 以 0.1s 间隔闪 3 下
////////////////////////////////////////////////////////////////////////////////
void proc_powerOff_handle()
{
    buzzer_beep(1);
    u8 i =0 ;
    for(i=1; i<7; ++i) //power down ui_led
    {
        led_beep (i&0x01);
        sleep_us (100*1000);
    }
    sleep_us(100*1000);

    //recover gp17 and gp18 to pulldn 100k
    while(gpio_read(GPIO_GP18))//按键按下时，芯片进入 suspend ，并每 1s 定时醒来一次
         blt_sleep_wakeup(0,PM_WAKEUP_TIMER,clock_time() + CLOCK_SYS_CLOCK_1S);
```

```
    blt_sleep_wakeup(1,PM_WAKEUP_PAD,0);//无按键按下时，芯片进入 deepsleep 模式并开
//启按键唤醒使能
}
/////////////////////////////////////////////////////////////////////////////////
// blt_set_ui_suspend_en 用于控制芯片是否使能 suspend 省电模式
/////////////////////////////////////////////////////////////////////////////////
void blt_set_ui_suspend_en( u8 suspend_en){
    ui_suspend_en = suspend_en;
}
#endif
/////////////////////////////////////////////////////////////////////////////////
// public_loop 用于控制芯片广播以及连接状态下的数据包发送，进入 suspend PM 电源管理模式，
//详情查看<Lenze 17H26 BLE SDK User Guide_v1.01>中的第 3,4 章 BLE 以及 PM 工作方式
/////////////////////////////////////////////////////////////////////////////////
static inline void public_loop()
{
    tick_app_wakeup = buzzer_led_ui () ;
    blt_brx_sleep (tick_app_wakeup);
    if(blt_state!=BLT_LINK_STATE_ADV)
    {
        blt_brx ();

        if(blt_conn_inst > 20 && os_check < 2)
        {
           os_check = 2;
           vr_autoSetMode();
           hid_setting_flag(1);    //  android set ccc at default
        }
    }else{
        // Must be on the final
        blt_send_adv (BLT_ENABLE_ADV_ALL);
        //blt_send_adv (BLT_ENABLE_ADV_38);
    }
}
/////////////////////////////////////////////////////////////////////////////////
// main_loop 用于综合控制芯片的广播状态，连接状态，外设以及调用内置的省电接口函数，完成
//一个完整的系统功能
/////////////////////////////////////////////////////////////////////////////////
void main_loop()
{
  {//
    if (blt_state == BLT_LINK_STATE_ADV)
    {
        shutdown_cnt = 90 ;
//PM 电源管理，开机后 60s 刷新 selfie_adv_mode_start_tick 的值，之后再 60s 进入关机模式
        if(clock_time_exceed(selfie_adv_mode_start_tick,60*1000*1000)){
```

```
            if(mle_15_mode && proximity_le_mode){
                power_mode = Mode_Power_Off;
            }
            if(proximity_le_mode == 0 ){
                proximity_le_mode = 1;
                selfie_adv_mode_start_tick = clock_time();
            }
        }
//PM 电源管理，开机后 60s 若 proximity_le_mode=1,更新广播间隔参数为 2000ms
        if(proximity_le_mode){
            blt_adv_interval = ((rand()%80) +2000)*CLOCK_SYS_CLOCK_1MS;
            led_enter_mode(0);
        }
    else {//若 proximity_le_mode=0，则每满足更新广播间隔参数的条件，就更新参数一次
            if(clock_time_exceed(selfie_adv_mode_start_tick,60*100*1000)) {
//若开机后超过 60s，且 proshutter_disconnect_state 标志位被置 1，则主动断开连接
                if(proshutter_disconnect_state) {
                    u8 disconnect = 1;
                    task_connection_terminated(&disconnect);
                }
            }
            if(clock_time_exceed(selfie_adv_mode_start_tick,6*1000*1000)) {
//若开机后超过 6s，则更新广播间隔参数为 600ms
                blt_adv_interval = ((rand()%20) + 600)*CLOCK_SYS_CLOCK_1MS;
            }
            else if(clock_time_exceed(selfie_adv_mode_start_tick,4*1000*1000)) {
//若开机后超过 4s，则更新广播间隔参数为 50ms
                blt_adv_interval = ((rand()%20) + 50)*CLOCK_SYS_CLOCK_1MS;
            }
            else if(clock_time_exceed(selfie_adv_mode_start_tick,2*1000*1000)) {
//若开机后超过 2s，则更新广播间隔参数为 500ms
                blt_adv_interval = ((rand()%20) + 500)*CLOCK_SYS_CLOCK_1MS;
            }
            else {
//否则，更新广播间隔参数为 15ms
                blt_adv_interval = ((rand()%5) + 15)*CLOCK_SYS_CLOCK_1MS;
            }
        }
    }else{
//若“设备处于连接状态，且完成连接之后的时间超过了 6s” ，blt_fifo_num()<3，则请求更新参数
    if((device_status_tmp==CONNECTED_DEVICE_STATUS) &&
clock_time_exceed(tick_connected_timer_tmp,6*1000*1000))// 6s timer out
        {
            if(blt_fifo_num()<3) {
               device_status_tmp = AFTER_CONNECTED_DEVICE_STATUS;
```

```
//void blt_update_connPara_request (u16 min_interval, u16 max_interval, u16
//latency,u16 timeout);  用于更新需要发送的关于更新连接包的参数。注意 interval min 和
//interval max 的值是实际 interval 时间值除以 1.25 ms（如申请 7.5ms 的连接，该值为 6），
//timeout 的值为实际 supervision timeout 时间值除以 10ms（如 1 s 的 timeout 该值为
//100）。
               blt_update_connPara_request(16,32,4,600);
            }
        }
//若不满足”设备处于连接状态，且完成连接之后的时间超过了 6s”,则对 proximity_le_mode 置
0，令 led 进入模式 1
        proximity_le_mode = 0;
        led_enter_mode(1);     }
    }
 //若当前设备状态为关机，且处于广播模式，蓝牙发送区为空，那么进入关机模式
    if(Mode_Power_Off == power_mode && (blt_state == BLT_LINK_STATE_ADV ||
blt_fifo_empty())){       //must be after send button pkt.
        proc_powerOff_handle();
    }
/*******************************public area******************/
    public_loop();
}
////////////////////////////////////////////////////////////////////////////////
// GPIO 模拟串口 uart  用于 输出 串口数据并与其他设备通信
  //////////////////////////////////////////////////////////////////////////////

 u32 simulation_BaudRate = 22;
////////////////////////////////////////////////////////////////////////////////
// GPIO 模拟串口 uart 初始化用于设定 gpio 初始电平和每个 bit 输出到切换到下一 bit 的时间
  //////////////////////////////////////////////////////////////////////////////
void my_uart_init(void)
{
    gpio_write(GPIO_GP17, 1);
//  simulation_BaudRate = 53;//// baud rate: 57600   17.361us
    simulation_BaudRate = 22;//// baud rate:  115200   8.56us
}
////////////////////////////////////////////////////////////////////////////////
// GPIO 模拟串口延时程序 : 每个 bit 输出到切换到下一 bit 的时间
  //////////////////////////////////////////////////////////////////////////////
_attribute_ram_code_ _attribute_no_inline_ void delay_uart_Tx(void)
{
    volatile int i =0;
    while((i++)<simulation_BaudRate);
}

////////////////////////////////////////////////////////////////////////////////
// GPIO 模拟串口 uart , 用于从当前 GPIO 口输出单字节数据
```

```
//////////////////////////////////////////////////////////////////////////////
void uart_tx_byte(u8 temp_data){
    for(u8 bit=0;bit<10;bit++){
        if(bit == 0){
            gpio_write(GPIO_GP17, 0);
        }
        else if(bit == 9){
            gpio_write(GPIO_GP17, 1);
        }
        else {
            if(temp_data & (1<<(bit-1))){
                gpio_write(GPIO_GP17, 1);
            }
            else {
                gpio_write(GPIO_GP17, 0);
            }
        }
        delay_uart_Tx();
    }
}
//////////////////////////////////////////////////////////////////////////////
// GPIO 模拟串口 uart ，用于从当前 GPIO 口输出任意长度的数组
//////////////////////////////////////////////////////////////////////////////
void uart_tx_array(u8 *uart_tx_array,u8 length)
{
    for(u8 i=0;i<length;i++){
        uart_tx_byte(uart_tx_array[i]);
    }
}
```