17H26 的 GPIO 使用说明

本文是基于“v0110”版本的 SDK -H26SDKCommon_GATT_GPIO_TEST 的使用介绍，读者在看本文档时，可以直接打开对应名称的 SDK，加以实验。下面将从程序的起步和执行顺序讲起。

程序执行顺序 如下 main 中所示，在进入 while（1）之前，程序先后顺序运行 main 函数中的子函数，完成 17H26 的 cpu 唤醒初始化，cpu 时钟频率初始化,程序运行内存设定（flash 或 otp）初始化，adc 模块初始化，用户定义模块初始化，使能可编程看门狗等功能。

```
int main (void) {
    cpu_wakeup_init();

    //clock_init
    write_reg8(0x66, 0x26); // 32M pll

    gpio_init();

   // reg_irq_mask = FLD_IRQ_ZB_RT_EN;
//    rf_drv_init(CRYSTAL_TYPE);
#if(DEBUG_FROM_FLASH)
    rf_drv_1M_init_flash();
#else
    rf_drv_1M_init_OTP();
#endif


#if(MODULE_ADC_ENABLE)
        adc_init();
#endif
    user_init ();

    watch_dog_en();
    while (1) {
        main_loop ();
        clr_watch_dog();
    }
}
```

主循环 main_loop();的作用包含 1.维持蓝牙正常工作的子函数 public_loop()；2.设置与功耗关系密切的广播状态的广播间隔 blt_adv_interval 和连接状态的连接间隔参数(在下图截图中的 blt_update_connPara_request()函数中设置)；3.使得用户能够向其中添加特殊需求的子函数 user_ui_process()，本例中着重讲在此函数中如何添加按键点灯。主从端的数据传输部分则在“数传 ibeacon”一文中详细说明。

```
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 ();
    }
    else {
        // Must be on the final
        blt_send_adv (BLT_ENABLE_ADV_ALL);
        //blt_send_adv (BLT_ENABLE_ADV_38);
    }
}

void main_loop()
{
//  extern u8 start_ota_flag;
//  if(start_ota_flag==0)
//  {
//      user_ui_process();
//  }
    user_ui_process();
    if (blt_state == BLT_LINK_STATE_ADV)
    {
        blt_adv_interval = ((rand()% 10) + LOW_ADV_INTERVAL-1)*CLOCK_SYS_CLOCK_1MS;
    }
    else {
        if((device_status_tmp==CONNECTED_DEVICE_STATUS)&& clock_time_exceed(tick_connected_timer_tmp,1*1000*1000))
        {
            device_status_tmp=AFTER_CONNECTED_DEVICE_STATUS;
            blt_update_connPara_request(160,180,4,400);
        }
    }
/*****************************public area*****************/
    public_loop();
}
```

IO 口初始化和使用：

在 user_init() 中调用 下面的配置命令：

1）、设置 GP18 为按键输入 GPIO 口，并使用内部下拉 100k 欧姆电阻(analog_write(0x08,0x0f);)的配置方式。这个 GPIO 的设置要查看《DS_ST17H2628283038-E11_Datasheet for LENZE ultra-low cost BLE SoC》第 78 页介绍。

2）、设置 GP7 为 LED 灯输出 GPIO 口，并使用下拉 100k 欧姆的配置方式,对于这个 GPIO 的设置要查看《DS_ST17H2628283038-E11_Datasheet for LENZE ultra-low cost BLE SoC》第 32 页介绍。

3）、

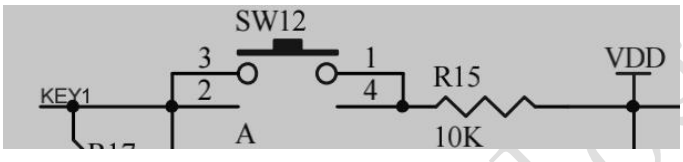
```
//  power_mode = Mode_Power_On;

    gpio_set_func(GPIO_GP18, AS_GPIO);
    gpio_set_output_en(GPIO_GP18,0);
    gpio_set_input_en(GPIO_GP18,1);
    analog_write (0x08, 0x0f);//G17  G18****100K
    gpio_set_func(GPIO_GP7, AS_GPIO);
    gpio_set_output_en(GPIO_GP7,1);
    gpio_set_input_en(GPIO_GP7,0);

    led_enter_mode(1);
    buzzer_enter_mode(1);

    /*set PWM Param*/
```

3)、设置 GP10 为输出 PWM 口，并使用无上下拉电阻的配置方式，并对对应的 pwm 频率加以设置。

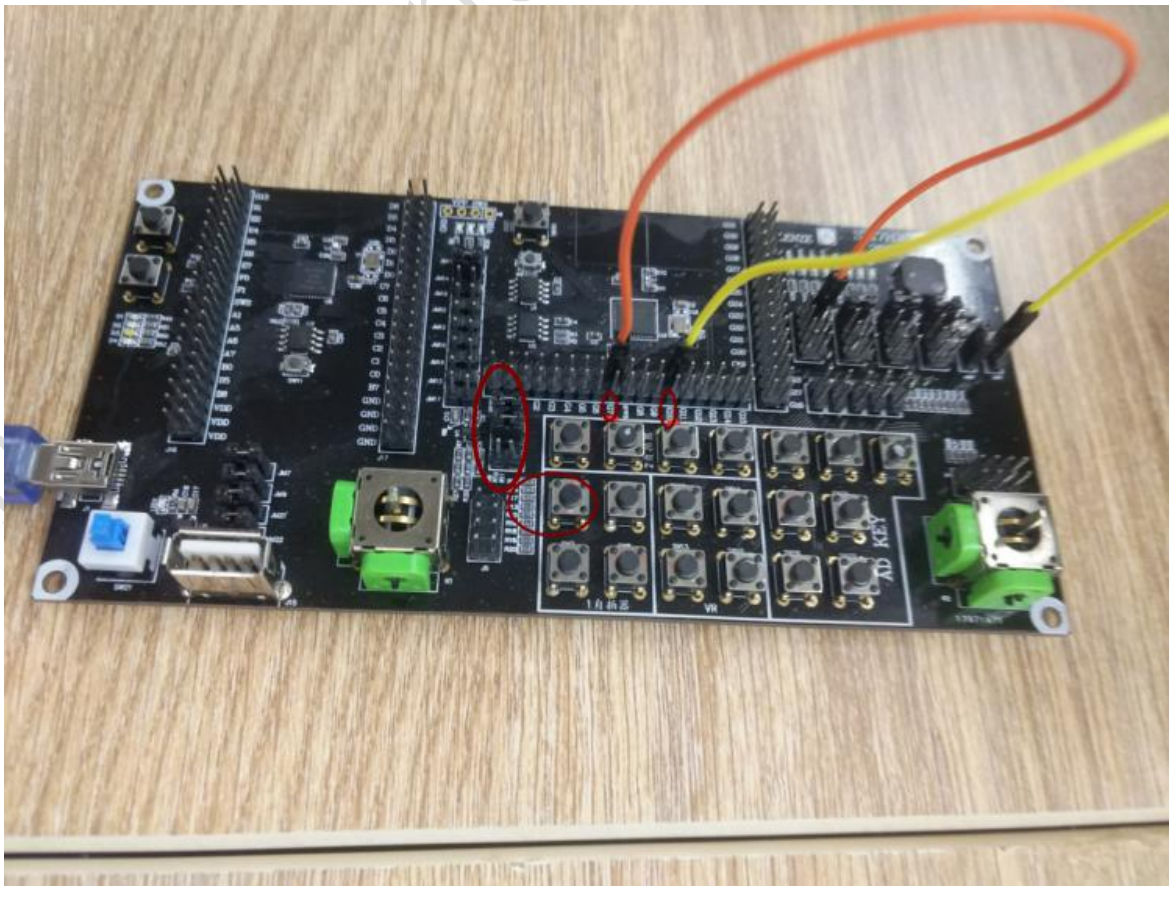
```
        u16 buzzer_freq = 727;      ///2.7K = 2M/740
//  }

    write_reg8(0x781,0x0f);//32M/(15+1) = 2M
    /* set buzzer pwm */
//  write_reg16(0x79a,727);//set pwm3 max cycle 2.7K = 2M/741
//  write_reg16(0x798,363);//set pwm3 duty_cycle = 50% = 370/1000
    write_reg16(0x79a,buzzer_freq);//set pwm3 max cycle 2.7K = 2M/741
    write_reg16(0x798,(buzzer_freq>>1));//set pwm3 duty_cycle = 50% = 370/1000

    gpio_set_output_en(GPIO_GP10,1);
    gpio_set_input_en(GPIO_GP10,0);
    gpio_set_func(GPIO_GP10,AS_PWM);
```

按键点灯子函数 user_ui_process()：

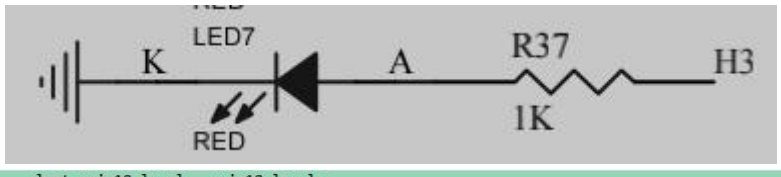
GPIO18 接到原理图上的 KEY1，再上拉 10k 的电阻到电源端，下图程序中可以看到，在按键未按下时，gpio18 口为低电平，按键程序会跳转到 else 内的部分（下图中的第二个方框），做判断。在按键按下时，gpio18 口为高电平，按键程序会跳转到 if(button_press)（下图中的第一个方框）内的部分，设置 led 的闪灯模式为 3。



即在开发板上按下图所示的方法接线，最大的椭圆红圈中圈住的部分是使用 GPIO18 时，必须接的跳线：



GPIO7 接到下图中 H3 的位置，当输出高电平时，灯亮，否则灭灯。




```
    last_gpio18_level = gpio18_level;

    u8 button_pressed = /*gpio17_level_inv || gpio18_level_inv || gpio22_level || gpio17_level ||*/
    if(button_pressed){
        if((!last_button_pressed)&&(0==btnClock)){
            btnClock=1;
            fisTime=clock_time();
            }
        if(!last_button_pressed && (!button_check_time || clock_time_exceed(last_button_release_tim
            buzzer_enter_mode(3);
            led_enter_mode(3);
            FFE1_value[0] = 0x01;
//          blt_push_notify(10,FFE1_value[0],1);
            last_button_pressed = 1;
            last_button_press_time = clock_tick;
            blt_disable_latency();
            //blt_push_notify(30, 0x01, 1);
            proximity_le_mode = 0;
            selfie_adv_mode_start_tick = clock_tick;
        }
         if(clock_time_exceed(last_button_press_time,900*1000)){  //last_button_press_time,1500ms
                FFE1_value[0] = 0x02;
//              blt_push_notify(10,FFE1_value[0],1);
         }
         if(!last_button_pressed &&(clock_time_exceed(last_button_release_time,65*1000))){
                FFE1_value[0] = 0x03;
                last_button_pressed = 1;
//              blt_push_notify(10,FFE1_value[0],1);
         }
    }
    else {
            if(last_button_pressed && clock_time_exceed(last_button_press_time, 80*1000)){
                last_button_pressed = 0;
                last_button_release_time = clock_tick;
            }
    }
    if((btnClock==1)&&clock_time_exceed(fisTime,1000*1000)){
            btnClock=2;
            blt_push_notify(10,FFE1_value[0],1);
            btnClock=0;
    }
```

长按，短按，连按按键的判断方法：

下图所示的程序是一个长短按的判定程序框架。FFE1_value 值 01 代表单击按键的情况，FFE1_value 值 02 代表长按按键的情况，FFE1_value 值 03 代表双击按键的情况。只要此次按键按下的时间和上次的时间间隔超过 500ms，我们就认为发生一次单击事件（实际的间隔用户可以自己调整）。当按键按下的时间超过 900ms 了，我们就认为是长按事件。当此次按键按下的时间和上次放开按键的时间超过 65ms 了，我们就认为 是双按事件。需要注意的是，从按键按下到从从端向主端的发送按键数据，我们要等够一秒，才能确定实际是长按还是短按。最终通过 blt_push_notify(15，FFE0_value，1)；函数的对应 Handle =15 的 0xFFE1 UUID 发送键值（截图中的数值不对，以文字描述为准）。如果实际需要长按的时间超过 5s ，那么就需要对按键判定的时间节点(下图黄圈中的数)加以变通。

```
u8 button_pressed = /*gpio17_level_inv || gpio18_level_inv || gpio22_level || gpio17_level ||*/ gpio18_lev
if(button_pressed){
    if((!last_button_pressed)&&(0==btnClock)){
        btnClock=1;
        fisTime=clock_time();
        }
    if(!last_button_pressed && (!button_check_time || clock_time_exceed(last_button_release_time, 500*1000
        buzzer_enter_mode(3);
        led_enter_mode(3);
        FFE1_value[0] = 0x01;
//          blt_push_notify(10,FFE1_value[0],1);
        last_button_pressed = 1;
        last_button_press_time = clock_tick;
        blt_disable_latency();
        //blt_push_notify(30, 0x01, 1);
        proximity_le_mode = 0;
        selfie_adv_mode_start_tick = clock_tick;
    }
     if(clock_time_exceed(last_button_press_time,900*1000)){  //last_button_press_time,1500ms  last_button
          FFE1_value[0] = 0x02;
//          blt_push_notify(10,FFE1_value[0],1);
     }
     if(!last_button_pressed &&(clock_time_exceed(last_button_release_time,65*1000) ){
          FFE1_value[0] = 0x03;
          last_button_pressed = 1;
//          blt_push_notify(10,FFE1_value[0],1);
     }
}
else {
        if(last_button_pressed && clock_time_exceed(last_button_press_time, 80*1000)){
            last_button_pressed = 0;
            last_button_release_time = clock_tick;
        }
}
if((btnClock==1)&&clock_time_exceed(fisTime,1000*1000)){
        btnClock=2;
        blt_push_notify(10,FFE1_value[0],1);
        btnClock=0;
}
```

用户如果想实现变量观测，请回到 Ui.c 的首部，按如下说明修改。这个方法对于所有蓝牙 sdk 均适用。

```
#include "ui.h"
#include "../../proj_lib/ble_l2cap/ble_ll_ota.h"

#include "../../proj/drivers/flash.h"
#include "yj_ui.h"
//static inline void send_databuf_tmp();
#define  TAIL_BOOT_CODE_PRESET      0
#if(TAIL_BOOT_CODE_PRESET)
volatile static u8 test_boot_code = 0;
_attribute_custom_code_  volatile u8 const boot_code[] = {
0x02,0x00,0x00,0xC5,0xFF,0xFF,0xFF,0xFF,0xFE,0xFF,0xFF,0xFF,0xFF,0xFF,0x0F,0xFC,
0xBF,0x98,0x02,0x06,0xBF,0x01,0x03,0x06,0x3F,0xF8,0x00,0x00,0xFF,0xFF,0xFF,0xFF
};
#endif


////////////////////////////////cfg address  //////////////

#if (1)
#define     SUSPEND_STATE           SUSPEND_CONN | SUSPEND_ADV
#else
#define     SUSPEND_STATE           0
#endif

#if(OTA_ENABLE)
#define TEST_OTA_1 0
#endif

#define  TEST_SUSPEND_TIME_ENABLE 0
```

要实现wtcdb 中的变量观测，首先要关闭低功耗模式，将箭头所指位置改为#if（0）！！！