HiSpark-WiFi-IoT 按键功能编程指南

文档版本　00B01

发布日期　2020/8/3

商标声明

 、HISILICON、海思和其他海思商标均为上海海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址：　　上海市青浦区金泽镇（西岑）水秀路 318 号 101 室　邮编：201718

网址：　　http://www.hisilicon.com

前　言

概述

本文档主要介绍基于海思 WiFi 芯片 Hi3861 开发的 HiSpark-WiFi-IoT 套件演示指导书。

产品版本

与本文档相对应的主芯片版本如下。

产品名称	产品版本
Hi3861	V100R001C00SPC021

读者对象

本文档（本指南）主要适用于以下工程师：

- 软件开发工程师
- 硬件开发工程师

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2020-08-3	00B01	第一次临时版本发布。

目 录

1 WIFI-IoT 开发板按键功能实现

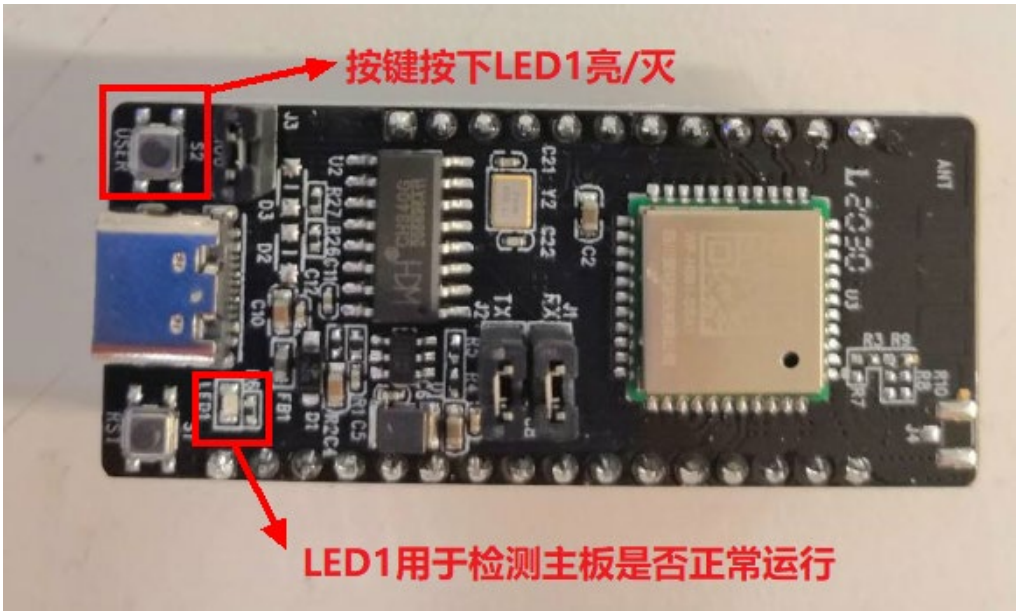
1.1 WIFI-IOT 开发板套件——按键功能介绍

WIFI-IOT 开发板套件，名称 HiSpark，以下简称 HiSpark 开发板套件。

HiSpark 开发板的硬件上有 5 个按键，其中一个为复位键，作用：复位重启；其余的 4 个都有各自的功能。

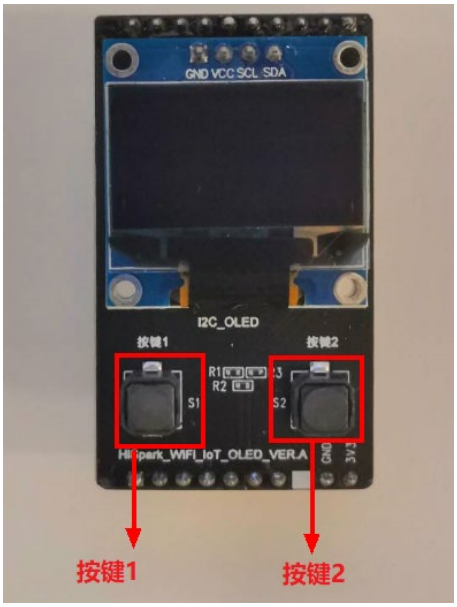
1. HiSpark 主板上的按键作用：在固件烧录后，按下该按键，主板上的 LED1 灯长亮，再次按下熄灭，检测 HiSpark 主板是否正常工作。

图 1.1-1



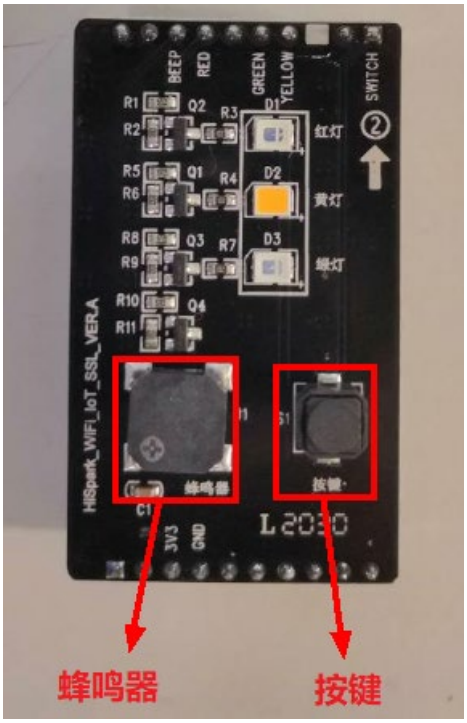
2. I2C_OLED 屏板上的按键作用：**按键 1 的作用**是选择功能和模式菜单，其中包括选择进入哪个功能菜单，如炫彩灯功能/交通灯功能/环境监测功能/NFC 功能，以及在每个功能下的模式选择，如在进入炫彩灯功能时，炫彩灯有 7 中模式，通过按键 1 选择这 7 种模式下的功能，并显示在 OLED 屏上；**按键 2 的作用**是在每一种功能下的类型选择，比如选择炫彩灯功能下的 pwm 模式，在 pwm 模式下有 5 种类型，这时通过按键 2 来选择进入哪种类型，并显示在 OLED 屏上。

图 1.1-2



3. 交通灯板上的按键作用：选择蜂鸣器是否开启，程序启动后，蜂鸣器默认关闭。当按下按键后，蜂鸣器开启，再次按下关闭。

图 1.1-2



1.2 按键功能实现逻辑

1. HiSpark 主板上的按键（USER）和 I2C_OLED 屏板上的 2 个按键（S1、S2）共用一个 GPIO5 pin 口，GPIO5 pin 口外设硬件采用 ADC 中断的方式，通过采集按键按下的电压不同判断是哪个按键按下。中断采用下降沿、边沿触发方式，按键防抖采用软件防抖设计。

2. 交通灯板上的按键（S1）通过 GPIO8 pin 口控制，采用 GPIO 外部中断，下降沿、边沿触发方式，按键挡都采用软件防抖设计。

图 1.2-1

```
hi_void gpio8_interrupt(hi_void)
{
    hi_u32 tick_interval =0;
    hi_u32 current_gpio8_tick =0;

    current_gpio8_tick = hi_get_tick();
    tick_interval = current_gpio8_tick - g_gpio8_tick;
    printf("gpio8 interrupt \r\n");

    if (tick_interval < KEY_INTERRUPT_PROTECT_TIME) {
        printf("gpio8 interrupt return  \r\n");
        return HI_NULL;
    }
    g_gpio8_current_type++;

    if (g_gpio8_current_type %2 == 0) {
        printf("beep off\r\n", g_gpio8_current_type);
        oc_beep_status = BEEP_OFF;
    } else {
        printf("beep on %d\r\n", g_gpio8_current_type);
        oc_beep_status = BEEP_ON;
    }

    if (g_gpio8_current_type >= 255)  {
        g_gpio8_current_type=0;
    }
}
```

按键防抖

1.3 按键功能软件实现

1. LiteOS 打开 WIFI-IOT 的 SDK 后，首先初始化 GPIO 配置

hi_void hi_switch_init(hi_io_name id, hi_u8 val, hi_gpio_idx idx, hi_gpio_dir dir, hi_io_pull pval);调用这个代码里封装好的接口进行 GPIO 口配置；

其中 hi_switch_init 接口参数含义如下：

hi_io_name id：gpio 硬件管脚编号，对应的是硬件管脚上的 GPIO pin 脚，枚举类型如下：

typedef enum {

HI_IO_NAME_GPIO_0, /**< GPIO0 */

HI_IO_NAME_GPIO_1, /**< GPIO1 */

HI_IO_NAME_GPIO_2, /**< GPIO2 */

```
    HI_IO_NAME_GPIO_3,      /**< GPIO3 */
    HI_IO_NAME_GPIO_4,      /**< GPIO4 */
    HI_IO_NAME_GPIO_5,      /**< GPIO5 */
    HI_IO_NAME_GPIO_6,      /**< GPIO6 */
    HI_IO_NAME_GPIO_7,      /**< GPIO7 */
    HI_IO_NAME_GPIO_8,      /**< GPIO8 */
    HI_IO_NAME_GPIO_9,      /**< GPIO9 */
    HI_IO_NAME_GPIO_10,     /**< GPIO10 */
    HI_IO_NAME_GPIO_11,     /**< GPIO11 */
    HI_IO_NAME_GPIO_12,     /**< GPIO12 */
    HI_IO_NAME_GPIO_13,     /**< GPIO13 */
    HI_IO_NAME_GPIO_14,     /**< GPIO14 */
    HI_IO_NAME_SFC_CSN,      /**< SFC_CSN */
    HI_IO_NAME_SFC_IO1,     /**< SFC_IO1 */
    HI_IO_NAME_SFC_IO2,     /**< SFC_IO2 */
    HI_IO_NAME_SFC_IO0,     /**< SFC_IO0 */
    HI_IO_NAME_SFC_CLK,      /**< SFC_CLK */
    HI_IO_NAME_SFC_IO3,     /**< SFC_IO3 */
    HI_IO_NAME_MAX,
} hi_io_name;
```

hi_u8 val :对应 GPIO 引脚功能，如果该引脚复用，可以配置为其他的复用功能，如：UART/PWM/I2C/SPI/SDIO 等功能；

hi_gpio_idx idx：选择 GPIO 引脚，枚举类型如下：

```
typedef enum {
    HI_GPIO_IDX_0,      /**< GPIO0*/
    HI_GPIO_IDX_1,      /**< GPIO1*/
    HI_GPIO_IDX_2,      /**< GPIO2*/
    HI_GPIO_IDX_3,      /**< GPIO3*/
    HI_GPIO_IDX_4,      /**< GPIO4*/
    HI_GPIO_IDX_5,      /**< GPIO5*/
    HI_GPIO_IDX_6,      /**< GPIO6*/
    HI_GPIO_IDX_7,      /**< GPIO7*/
    HI_GPIO_IDX_8,      /**< GPIO8*/
    HI_GPIO_IDX_9,      /**< GPIO9*/
```

HI_GPIO_IDX_10, /**< GPIO10*/

HI_GPIO_IDX_11, /**< GPIO11*/

HI_GPIO_IDX_12, /**< GPIO12*/

HI_GPIO_IDX_13, /**< GPIO13*/

HI_GPIO_IDX_14, /**< GPIO14*/

HI_GPIO_IDX_MAX, /**< Maximum value, which cannot be used.CNcomment:最大值，不可输入使用 CNend*/

} hi_gpio_idx;

hi_gpio_dir dir:将选择的 GPIO 引脚设置方向，设置 GPIO 输入/输出；枚举类型如下：

typedef enum {

HI_GPIO_DIR_IN = 0, /**< Input.CNcomment:输入方向 CNend*/

HI_GPIO_DIR_OUT /**< Output.CNcomment:输出方向 CNend*/

} hi_gpio_dir;

hi_io_pull pval：选择将 GPIO 口拉高/拉低，枚举类型如下：

typedef enum {

HI_IO_PULL_NONE, /**< Disabled.CNcomment:无拉 CNend */

HI_IO_PULL_UP, /**< Pull-up enabled.CNcomment:上拉 CNend */

HI_IO_PULL_DOWN, /**< Pull-down enabled.CNcomment:下拉 CNend */

HI_IO_PULL_MAX, /**< Invalid.CNcomment:无效值 CNend */

} hi_io_pull;

2. HiSpark 开发板上按键初始化配置如下：

接口定义：

图 1.3-1

```
/* gpio key init*/
hi_void hi_switch_init(hi_io_name id, hi_u8 val,hi_gpio_idx idx, hi_gpio_dir dir, hi_io_pull pval)
{
    hi_io_set_func(id, val);
    hi_gpio_set_dir(idx, dir);
    hi_io_set_pull(id,pval);
}
```

接口调用：

图 1.3-2

```
hi_void test_gpio_init(hi_void)
{
    hi_switch_init(HI_IO_NAME_GPIO_5,HI_IO_FUNC_GPIO_5_GPIO,HI_GPIO_IDX_5,HI_GPIO_DIR_IN,HI_IO_PULL_UP);//switch2
    hi_switch_init(HI_IO_NAME_GPIO_8,HI_IO_FUNC_GPIO_8_GPIO,HI_GPIO_IDX_8,HI_GPIO_DIR_IN,HI_IO_PULL_UP);//switch2

    pwm_init(HI_IO_NAME_GPIO_10,HI_IO_FUNC_GPIO_10_PWM1_OUT,HI_PWM_PORT_PWM1);
    pwm_init(HI_IO_NAME_GPIO_11,HI_IO_FUNC_GPIO_11_PWM2_OUT,HI_PWM_PORT_PWM2);
    pwm_init(HI_IO_NAME_GPIO_12,HI_IO_FUNC_GPIO_12_PWM3_OUT,HI_PWM_PORT_PWM3);
}
```

3. HiSpark 开发板上按键中断实现和 ADC 电压采集，接口如图所示：

图 1.3-3

```
hi_void app_multi_sample_demo(hi_void)
{
    hi_u32 ret =HI_ERR_SUCCESS;

    (hi_void)hi_gpio_init();
    g_gpio5_tick = hi_get_tick();
    ret = hi_gpio_register_isr_function(HI_GPIO_IDX_5, HI_INT_TYPE_EDGE,HI_GPIO_EDGE_FALL_LEVEL_LOW, get_gpio5_voltage, HI_NULL);
    if (ret == HI_ERR_SUCCESS) {
        printf(" register gpio 5\r\n");
    }

    g_gpio8_tick = hi_get_tick();
    ret = hi_gpio_register_isr_function(HI_GPIO_IDX_8, HI_INT_TYPE_EDGE,HI_GPIO_EDGE_FALL_LEVEL_LOW, gpio8_interrupt, HI_NULL);
    if (ret == HI_ERR_SUCCESS) {
        printf(" register gpio8\r\n");
    }
}
```

中断软件实现步骤：

（1） 中断 GPIO 初始化：hi_gpio_init();

（2） 调用接口 hi_gpio_register_isr_function(),其中 HI_CPIO_IDX_5 表示设置 GPIO5 为外部中断引脚

（3） get_gpio5_voltage()为触发中断后的处理函数。（**注意：中断处理函数里面不应处理浮点运算、while 循环、pwm、也不能在里面等待信号量等操作，中断应短而高效，循环处理、PWM 等操作可以使用信号量去同步**）。

（4） get_gpio5_voltage()具体处理函数如下：

图 1.3-4

```
/*get gpio5 Voltage*/
hi_u8 get_gpio5_voltage(hi_void *param)
{
    int i;
    hi_u16 data;
    hi_u32 ret;
    hi_u16 vlt;
    float voltage;
    float vlt_max = 0;
    float vlt_min = VLT_MIN;

    hi_unref_param(param);
    memset_s(g_gpio5_adc_buf, sizeof(g_gpio5_adc_buf), 0x0, sizeof(g_gpio5_adc_buf));
    for (i = 0; i < ADC_TEST_LENGTH; i++) {
        ret = hi_adc_read(HI_ADC_CHANNEL_2, &data, HI_ADC_EQU_MODEL_4, HI_ADC_CUR_BAIS_DEFAULT, 0xF0); //ADC_Channal_2  自动识别模式  CNcomment:4次平均算法模式 CNend */
        if (ret != HI_ERR_SUCCESS) {
            printf("ADC Read Fail\n");
            return  HI_NULL;
        }
        g_gpio5_adc_buf[i] = data;
    }

    for (i = 0; i < ADC_TEST_LENGTH; i++) {
        vlt = g_gpio5_adc_buf[i];
        voltage = (float)vlt * 1.8 * 4 / 4096.0;  /* vlt * 1.8* 4 / 4096.0为将码字转换为电压 */
        vlt_max = (voltage > vlt_max) ? voltage : vlt_max;
        vlt_min = (voltage < vlt_min) ? voltage : vlt_min;
    }
    if (vlt_max > 0.6 && vlt_max < 1.0) {
        gpio5_isr_func_mode();
    } else if (vlt_max > 1.0 && vlt_max < 1.5) {
        gpio7_isr_func_type();
    } else if (vlt_max <0.6) {
        printf("gpio9_led_light:vlt_max=%0.2f, vlt_min=%0.2f\r\n", vlt_max, vlt_min);
        gpio9_led_light_func();
    }
    printf("key_5:vlt_max=%0.2f, vlt_min=%0.2f\r\n", vlt_max, vlt_min);
}
```

根据采集电压阈值判断哪个按键按下

（5） gpio8_interrupt()具体处理函数如下:

图 1.3-5

```
hi_void gpio8_interrupt(hi_void)
{
    hi_u32 tick_interval =0;
    hi_u32 current_gpio8_tick =0;

    current_gpio8_tick = hi_get_tick();
    tick_interval = current_gpio8_tick - g_gpio8_tick;
    printf("gpio8 interrupt \r\n");

    if (tick_interval < KEY_INTERRUPT_PROTECT_TIME) {
        printf("gpio8 interrupt return  \r\n");
        return HI_NULL;
    }
    g_gpio8_current_type++;

    if (g_gpio8_current_type %2 == 0) {
        printf("beep off\r\n", g_gpio8_current_type);
        oc_beep_status = BEEP_OFF;
    } else {
        printf("beep on %d\r\n", g_gpio8_current_type);
        oc_beep_status = BEEP_ON;
    }

    if (g_gpio8_current_type >= 255)  {
        g_gpio8_current_type=0;
    }
}
```