按键值
获取逻辑

淘宝店铺：

PC 端：

http://n-xytrt8gqu585po94mwj5atokcyd4.taobao.com/index.htm

手机端：

https://shop.m.taobao.com/shop/shop_index.htm?sellerId=755668508&shopId=104493595&inShopPageId=423890608&pathInfo=shop/index2

资料下载地址：

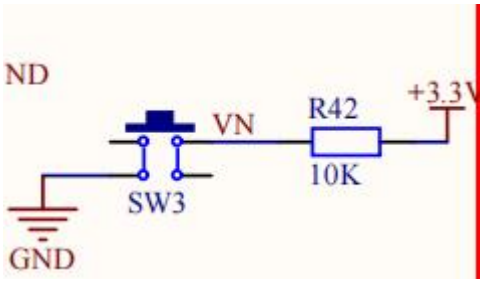
链接：https://pan.baidu.com/s/1kCjD8yktZECSGmHomx_veg?pwd=q8er
提取码：q8er

源码下载地址：

https://gitee.com/vi-iot/esp32-board.git

一、按键原理

大家在使用一些智能家居家电时，可能都会有那么几个按键，然后看操作说明，按键的短按长按功能是不一样的，短按的话基本上是执行标准功能，比如说开灯关灯，那么长按可能就是执行一些不常用的功能了，比如说配网。因此按键的处理对物联网开发来说是必备的基础技能，要说简单，也很简单，要玩出花样，也能玩出很多花样。本例程中会对单个按键实现长按和短按功能的划分，同时也可以扩展出多个按键的应用。原理图如下



按键在没按下的时候，GPIO 口被上拉电阻拉高，读取引脚的电平就是高电平 1，按键按下时，GPIO 本短接到地，读取引脚的电平就是低电平 0，原理图非常简单，我们只需在程序中对 GPIO 引脚设置成输入模式，实时读取电平即可。

二、第一个例程

这个例程实现的功能如下，我们实时的判断外部按键状态，当检测到低电平时，我们就把 LED 点亮，当检测到高电平时，我们就把 LED 熄灭。代码如下，代码位于 esp32-board/button

```
void app_main(void)
{
   //初始化 LED 的 GPIO 管脚，设置成输出
    gpio_config_t gpio_t =
    {
        .intr_type = GPIO_INTR_DISABLE,     //禁止中断
        .mode = GPIO_MODE_OUTPUT,           //输出模式
        .pin_bit_mask = (1ull<<LED_GPIO),   //GPIO 引脚号
        .pull_down_en = GPIO_PULLDOWN_DISABLE,//下拉禁止
        .pull_up_en = GPIO_PULLUP_DISABLE,   //上拉禁止
    };
    ESP_ERROR_CHECK(gpio_config(&gpio_t));

//简单按键例程
    xTaskCreatePinnedToCore(simple_btn_test,"btn1",2048,NULL,3,NULL,1);
}
```

```
/** 简单例程
 * @param 无
 * @return 无
*/
void simple_btn_test(void* arg)
{
    //初始化按键 GPIO
```

```
    gpio_config_t gpio_t =
    {
        .intr_type = GPIO_INTR_DISABLE,     //禁止中断
        .mode = GPIO_MODE_INPUT,            //输入模式
        .pin_bit_mask = (1ull<<BTN_GPIO),   //GPIO 引脚号
        .pull_down_en = GPIO_PULLDOWN_DISABLE,  //禁止下拉
        .pull_up_en = GPIO_PULLUP_ENABLE,   //使能上拉
    };
    ESP_ERROR_CHECK(gpio_config(&gpio_t));
    while(1)
    {
        //检测到按键按下，就亮，否则熄灭
        if(gpio_get_level(BTN_GPIO) == 0)
            gpio_set_level(LED_GPIO,1);
        else
            gpio_set_level(LED_GPIO,0);
        vTaskDelay(pdMS_TO_TICKS(50));
    }
}
```

在 app_main()函数中，初始化了 LED 的 GPIO 口，设置成输出模式，同时启用了一个任务 simple_btn_test，在这个新任务中处理 LED 和按键状态

simple_btn_test 函数中，对按键的 GPIO 口进行了初始化，并添加了内部上拉电阻（如果有外部上拉电阻，这里可以不设置）。然后在循环中不断读取按键状态，按键有按下，就输出 LED 的管脚为高电平点亮 LED，按键没按下就输出 LED 的管脚为低电平熄灭 LED。

三、第二个例程

现在我们来考虑更复杂的场景，需求如下：

1) 按键按下之后，LED 只闪亮一下，然后熄灭，不能一直亮
2) 按键长按超过 3 秒后，LED 闪亮 8 下，然后熄灭，提示用户已经检测到长按事件。
3) 在处理完长按后，没有释放按键之前，不做任何动作

对于按键事件的处理很简单，看如下代码

```
 EventBits_t ev;
    while(1)
    {
        //等待按键按下事件
        ev =
xEventGroupWaitBits(s_pressEvent,SHORT_EV|LONG_EV,pdTRUE,pdFALSE,portMA
X_DELAY);
        if(ev & SHORT_EV)
        {
            //短按事件，亮一下熄灭
            gpio_set_level(LED_GPIO,1);
            vTaskDelay(pdMS_TO_TICKS(200));
```

```
            gpio_set_level(LED_GPIO,0);
        }
        if(ev & LONG_EV)
        {
            //长按事件，闪烁 8 下，然后熄灭
            for(int i = 0;i < 8;i++)
            {
                gpio_set_level(LED_GPIO,1);
                vTaskDelay(pdMS_TO_TICKS(200));
                gpio_set_level(LED_GPIO,0);
                vTaskDelay(pdMS_TO_TICKS(200));
            }
        }
        xEventGroupClearBits(s_pressEvent,SHORT_EV);
        xEventGroupClearBits(s_pressEvent,LONG_EV);
    }
```

我们通过一个事件组来标识短按和长按事件，然后在任务循环中实时监控这个事件，短按事件则控制 LED 点亮，200ms 后熄灭。长按事件则用 for 循环来执行 8 次 led 的亮灭即可。在最后清除掉事件防止 LED 闪亮没处理完，又有事件重复触发。

那么接下来关键的是如何检测到这两个事件。这边思路是建立一个定时器，定时器中实时检测按键的状态，当检测到按键按下时，跳转到按键按下状态，同时触发按键短按回调函数，然后开始计数，当计数值到达预设的长按值时，就触发长按回调函数，同时把状态转入 LONG_HOLD 状态，不做任何处理，直到检测到按键释放。上面说的短按回调函数和长按回调函数里面，主要实现就是设置了相应的短按和长按事件位，由于代码较多，我只贴一部分关键代码。大家要看详细代码，到目录 esp32-board/button 中可以看到所有代码。

```
/** 定时器回调函数，本例中是 5ms 执行一次
 * @param cfg   配置结构体
 * @return ESP_OK or ESP_FAIL
*/
static void button_handle(void *param)
{
    int increase_cnt = (int)param;  //传入的参数是 5，表示定时器运行周期是 5ms
    button_dev_t* btn_target = s_button_head;
    for(;btn_target;btn_target = btn_target->next)
    {
        switch(btn_target->state)
        {
            case BUTTON_RELEASE:              //按键没有按下
                if(gpio_get_level(btn_target->btn_cfg.gpio_num) ==
btn_target->btn_cfg.active_level)
                {
                    btn_target->press_cnt += increase_cnt;
                    btn_target->state = BUTTON_PRESS;   //调转到按下状态
                }
```

```
                break;
            case BUTTON_PRESS:                  //按键按下了，等待一点延时（消抖），然后触发短按回调事件，进入 BUTTON_HOLD
                if(gpio_get_level(btn_target->btn_cfg.gpio_num) == btn_target->btn_cfg.active_level)
                {
                    btn_target->press_cnt += increase_cnt;
                    if(btn_target->press_cnt >= FILITER_TIMER)  //过了滤波时间，执行短按回调函数
                    {
                        ESP_LOGI(TAG,"short press detect");
                        if(btn_target->btn_cfg.short_cb)
                            btn_target->btn_cfg.short_cb();
                        btn_target->state = BUTTON_HOLD;    //状态转入按下状态
                    }
                }
                else
                {
                    btn_target->state = BUTTON_RELEASE;
                    btn_target->press_cnt = 0;
                }
                break;
            case BUTTON_HOLD:                   //按住状态，如果时间长度超过设定的超时计数，将触发长按回调函数，进入 BUTTON_LONG_PRESS_HOLD
                if(gpio_get_level(btn_target->btn_cfg.gpio_num) == btn_target->btn_cfg.active_level)
                {
                    btn_target->press_cnt += increase_cnt;
                    if(btn_target->press_cnt >= btn_target->btn_cfg.long_press_time)  //已经检测到按下大于预设长按时间,执行长按回调函数
                    {
                        ESP_LOGI(TAG,"long press detect");
                        if(btn_target->btn_cfg.long_cb)
                            btn_target->btn_cfg.long_cb();
                        btn_target->state = BUTTON_LONG_PRESS_HOLD;
                    }
                }
                else
                {
                    btn_target->state = BUTTON_RELEASE;
                    btn_target->press_cnt = 0;
                }
```

```
                break;
            case BUTTON_LONG_PRESS_HOLD:     //此状态等待电平消失，回到
BUTTON_RELEASE 状态
                if(gpio_get_level(btn_target->btn_cfg.gpio_num) !=
btn_target->btn_cfg.active_level)    //检测到释放，就回到初始状态
                {
                    btn_target->state = BUTTON_RELEASE;
                    btn_target->press_cnt = 0;
                }
                break;
            default:break;
        }
    }
}
```

button_handle 是定时器，每隔 5ms 执行一次， button_dev_t 结构体是对于单个按键的描述，本例程考虑了有多个按键，将这些按键描述通过链表串起来，然后循环遍历，对应配套教程的开发板只有一个按键，因此这个循环只执行一次，button_dev_t 链表只有一个内容。在这个循环中，会判断按键当前的状态，btn_target->state 状态类型分别有 4 个

1) BUTTON_RELEASE,//此状态表示按键没有按下，但会一直检测，如果检测到有按键按下，会进入到 BUTTON_PRESS 状态

2) BUTTON_PRESS,//按键按下了，如果计数值超过滤波时间，触发短按回调事件，然后进入 BUTTON_HOLD

3) BUTTON_HOLD, /按住状态，如果计数值超过预设长按超时，将触发长按回调函数，进入 BUTTON_LONG_PRESS_HOLD

4)BUTTON_LONG_PRESS_HOLD,   //此状态等待电平消失，回到 BUTTON_RELEASE 状态

btn_target->press_cnt 成员就是记录按键按下的时间，如果按键按下，每个定时器周期都会增加(本例中是 5ms)。

再看看 button 的相关初始化代码

```
//按键配置结构体
typedef struct
{
    int gpio_num;           //gpio 号
    int active_level;       //按下的电平
    int long_press_time;    //长按时间
    button_press_cb short_cb;   //短按回调函数
    button_press_cb long_cb;    //长按回调函数
}button_config_t;

/** 设置按键事件
 * @param cfg   配置结构体
 * @return ESP_OK or ESP_FAIL
```

```
*/
esp_err_t button_event_set(button_config_t *cfg);
```

这里的初始化需要新建一个按键配置，设定 gpio_num、按下时的电平、长按时间、短按和长按的回调函数，然后通过 button_event_set 把按键配置添加到按键配置链表中。看看在主函数中，是如果应用这个函数的。

```
/** 短按按键回调函数
 * @param 无
 * @return 无
*/
void short_press_handle(void)
{
    xEventGroupSetBits(s_pressEvent,SHORT_EV);
}

/** 长按按键回调函数
 * @param 无
 * @return 无
*/
void long_press_handle(void)
{
    xEventGroupSetBits(s_pressEvent,LONG_EV);
}
/** 完整的按键+LED 演示程序
 * @param 无
 * @return 无
*/
void complete_btn_test(void* arg)
{
    s_pressEvent = xEventGroupCreate();
    button_config_t btn_cfg =
    {
        .gpio_num = BTN_GPIO,       //gpio 号
        .active_level = 0,          //按下的电平
        .long_press_time = 3000,    //长按时间
        .short_cb = short_press_handle,          //短按回调函数
        .long_cb = long_press_handle            //长按回调函数
    };
    button_event_set(&btn_cfg);     //添加按键响应事件处理
    ……
```

上述我们定义了短按和长按回调函数，然后填充了 button_config_t 结构体，最后通过 button_event_set 把按键处理添加到链表中。

这样我们对于按键的处理就比较完善了。再次强烈建议大家去阅读一下源码，源码中的注释都比较详细。