SmartConfig

一键配网

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：

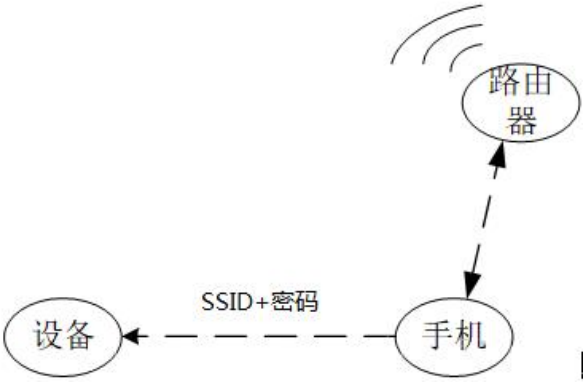
https://gitee.com/vi-iot/esp32-board.git

一、SmartConfig 知识扫盲

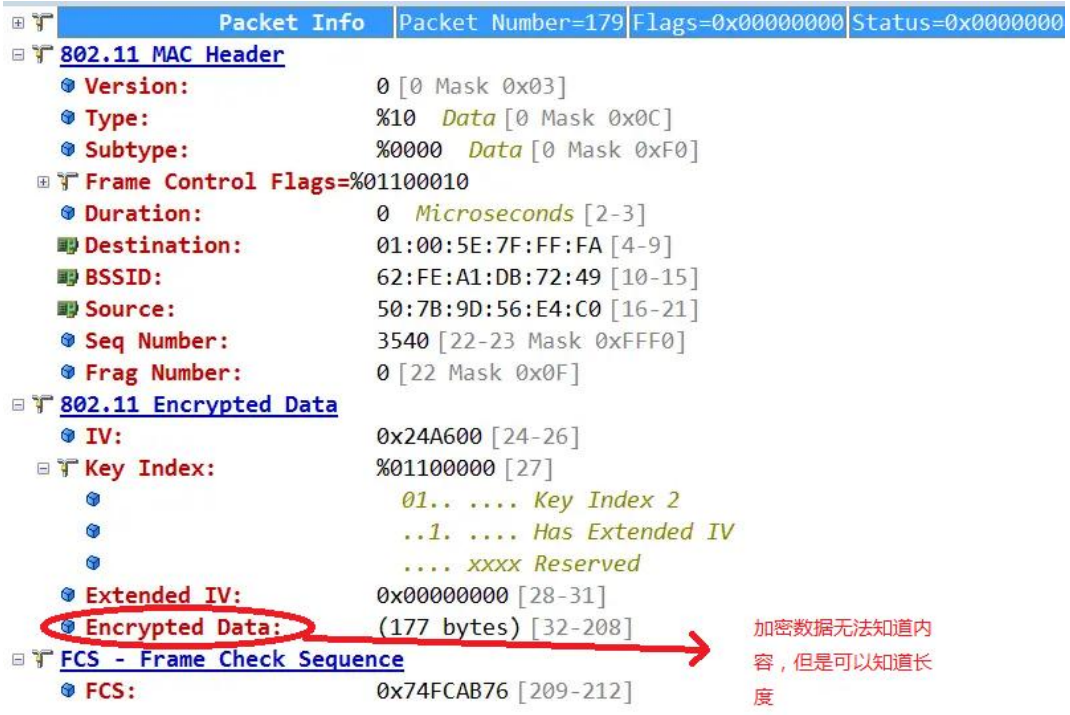
在讲 STA 课程的时候，我们用的是代码里面固定的 SSID 和密码去连接热点，但实际应用中不可能这么弄，我们得有办法把家里的 WiFi SSID 和密码输入到设备里面去，对于带屏带输入设备还好，因为可以人为手动输入，但很多 IOT 设备都不具备这种能力，因此我们需要其他方法。把 SSID 和密码告诉给设备，让设备能正确连接 WiFi 热点接入到物联网的过程，称为配网。

配网方式有很多种，比如 AP 配网、蓝牙配网，还有本课介绍的 SmartConfig 一键配网，SmartConfig 对用户来说操作是最简单的配网方式，其配网原理比较巧妙，我们来看下 SmartConfig 的基本原理到底如何。

首先要让 WiFi 芯片处于混杂模式下,监听网络中的所有报文;手机 APP 将 SSID 和密码编码到 UDP 报文中,通过广播包或组播报发送,智能硬件接收到 UDP 报文后解码,得到正确的 SSID 和密码,然后主动连接指定 SSID 的路由完成连接。



具体是如何接收报文的？另外在 802.11 协议中，MAC 帧数据域是加密的，设备没有连上 WiFi 是无法读取这部分内容的。具体帧格式如下图，我们只关注 MAC 帧中的数据域（MSDU）



对于 UDP 广播包，MAC 数据帧中数据部分（MSDU），我们无法知道内容，但是通过对帧的

解析我们知道这部分数据的长度，这部分数据是由 20 字节 IPv4 头部+8 字节 UDP 报文头部+UDP 内容组成的 IP 报文，假如 IP 报文长度为 500 字节，则 UDP 内容长度为 500-20-8=472 字节，这里我们定义，500 字节称为明文长度。

我们再制定一个定义，密文长度=明文长度+算法常量，算法常量往往是一个固定值，由 APP 和 WIFI 设备默认。

假如算法常量是 10。现在手机 APP 要传输 1234 这个数据，只需要在 UDP 报文内容中填充（1234-20(Ipv4 头)-8(UDP 报头)-10（算法常量））个字节即可（内容任意），也就是

IP 报文总长 1224

Ipv4 头：20 字节

UDP 头：8 字节

UDP 内容：1196 字节

也就是说我们通过 UDP 广播发送 1196 个字节就行，内容任意。

当设备收到这个 UDP 报文后需要解码，先得到 IP 报文长度 1224，然后我们要加上算法常量 10，得到 1234，因此设备最终获得了 1234 这个数据。

那么对于设备来说，如何知道这个 UDP 广播包就是 SmartConfig 发出的呢？这里涉及到一个前导码的概念，当设备 WIFI 开启混杂模式时，会在所处环境中快速切换各条信道来抓取每个信道中的数据包，当遇到正在发送前导码数据包的信道时，锁定该信道并继续接收广播数据，直到收到足够的数据来解码出其中的 WiFi 密码然后连接 WiFi，因此前导码一般由几个特殊的字节组成，方便和其他 UDP 包区分。

假设手机 APP 要发送”test”四个字符，算法常量为 16，流程如下：

1）APP 连续发送 3 个 UDP 广播包，数据为均为前导码。

2）APP 发送 1 个 UDP 广播包，IP 报文数据长度为’t’-16。

3）APP 发送 1 个 UDP 广播包，IP 报文数据长度为’e’-16。

4）APP 发送 1 个 UDP 广播包，IP 报文数据长度为’s’-16。

5）APP 发送 1 个 UDP 广播包，IP 报文数据长度为’t’-16。

6）APP 切换 WIFI 信道重复上述步骤

上述是数据传输的基本原理，但由于每一家厂商的算法常量、传输内容格式、前导码等都不一样，因此不同厂家之间的 SmartConfig 一般无法通用。

二、ESP32 中的 SmartConfig

通过查看 esp-idf 的源码，发现 ESP32 上的 SmartConfig 实现是看不到源码的，但不妨碍我们使用，而使用方式也比较简单，当然需要配合 APP 来使用，乐鑫官方也提供了 demo 版本的 APP，这个是开源的，我们可以集成到自己的 APP 应用中，下载地址是：

安卓：

https://github.com/EspressifApp/EsptouchForAndroid/releases/tag/v2.0.0/esptouch-v2.0.0.apk

IOS：

https://apps.apple.com/cn/app/espressif-esptouch/id1071176700

接下来看下 ESP32 源码，源码位于 esp32-board/wifi_smartconfig

由于在实际应用工程中，进入 SmartConfig 一般都是长按某个按键，因此这个例程中也

把之前按键短按长按处理的例程搬过来用了，长按 3 秒触发 SmartConfig。

app_main()如下

```
//按键事件组
static EventGroupHandle_t s_pressEvent;
#define SHORT_EV    BIT0    //短按
#define LONG_EV     BIT1    //长按
#define BTN_GPIO    GPIO_NUM_39

/** 长按按键回调函数
 * @param 无
 * @return 无
*/
void long_press_handle(void)
{
    xEventGroupSetBits(s_pressEvent,LONG_EV);
}
void app_main(void)
{
    nvs_flash_init();           //初始化 NVS
    initialise_wifi();          //初始化 wifi
    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 = NULL,           //短按回调函数
        .long_cb = smartconfig_start            //长按回调函数
    };
    button_event_set(&btn_cfg);     //添加按键响应事件处理
    EventBits_t ev;
    while(1)
    {
        ev =
xEventGroupWaitBits(s_pressEvent,LONG_EV,pdTRUE,pdFALSE,portMAX_DELAY);
        if(ev & LONG_EV)
        {
            smartconfig_start();    //检测到长按事件，启动 smartconfig
        }
    }
}
```

app_main()中注册了长按按键事件，主循环中检测到了长按事件，执行 smartconfig_start 函数，启动 SmartConfig。

接下来看下 main/wifi_smartconfig.c 文件对 SmartConfig 的处理

```
/** 启动 smartconfig
 * @param 无
 * @return 无
*/
void smartconfig_start(void)
{
    if(!s_is_smartconfig)
    {
        s_is_smartconfig = true;
        esp_wifi_disconnect();
        xTaskCreate(smartconfig_example_task, "smartconfig_example_task",
4096, NULL, 3, NULL);
    }
}
```

smartconfig_start 函数里面，会断开 wifi 连接然后启用一个 smartconfig 任务，s_is_smartconfig 标志是 SmartConfig 运行标志，防止重复执行 SmartConfig。

```
/** smartconfig 处理任务
 * @param 无
 * @return 无
*/
static void smartconfig_example_task(void * parm)
{
    EventBits_t uxBits;
    ESP_ERROR_CHECK( esp_smartconfig_set_type(SC_TYPE_ESPTOUCH_V2) );
       //设定 SmartConfig 版本
    smartconfig_start_config_t cfg = SMARTCONFIG_START_CONFIG_DEFAULT();
    ESP_ERROR_CHECK( esp_smartconfig_start(&cfg) ); //启动 SmartConfig
    while (1) {
        uxBits = xEventGroupWaitBits(s_wifi_event_group, CONNECTED_BIT |
ESPTOUCH_DONE_BIT, true, false, portMAX_DELAY);
        if(uxBits & CONNECTED_BIT) {
            ESP_LOGI(TAG, "WiFi Connected to ap");
        }
        if(uxBits & ESPTOUCH_DONE_BIT) {    //收到 smartconfig 配网完成通知
            ESP_LOGI(TAG, "smartconfig over");
            esp_smartconfig_stop();         //停止 smartconfig 配网
            write_nvs_ssid(s_ssid_value);   //将 ssid 写入 NVS
            write_nvs_password(s_password_value);   //将 password 写入 NVS
            s_is_smartconfig = false;
            vTaskDelete(NULL);              //退出任务
```

```
        }
    }
}
```

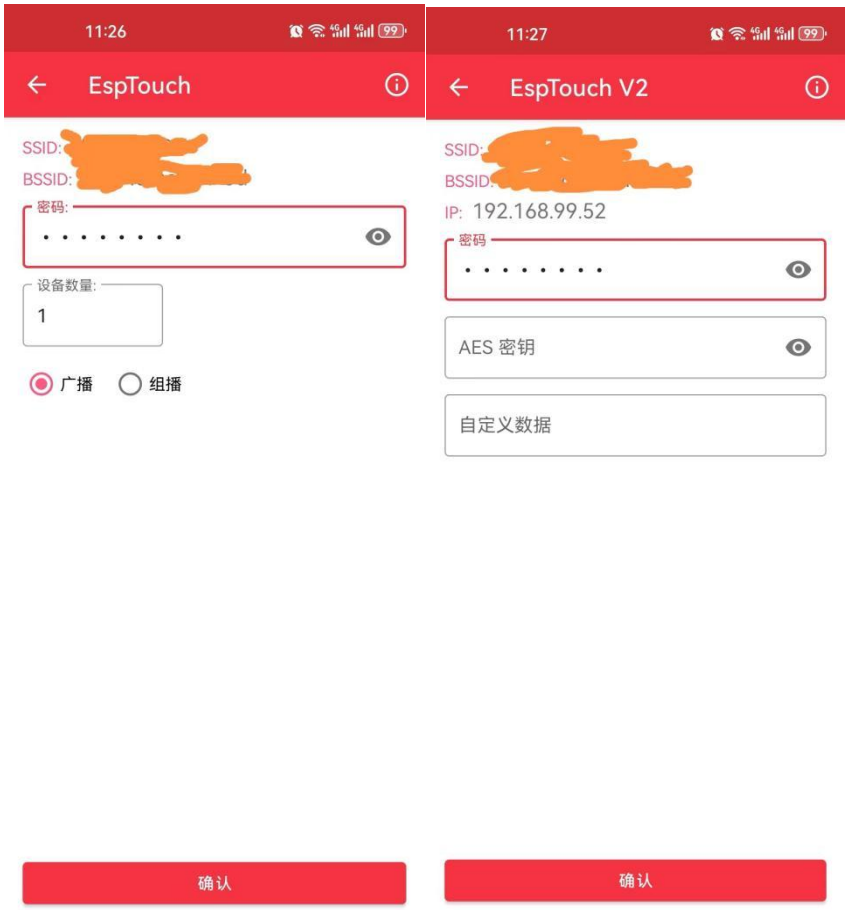
在 smartconfig_example_task 中会设定 SmartConfig 的版本，可以选 V1、V2、AirKiss（微信用的），版本之间不兼容，我这边选用了 V2，然后 esp_smartconfig_start(&cfg)启动 SmartConfig，后面会监听两个事件，一个是 WiFi 连接成功事件，一个是 SmartConfig 完成事件，当 SmartConfig 完成后，我们把 SSID 和密码保存到 NVS 中，然后退出任务，结束整个 SmartConfig 流程。

```
/** 各种网络事件的回调函数
 * @param arg 自定义参数
 * @param event_base 事件类型
 * @param event_id 事件标识 ID，不同的事件类型都有不同的实际标识 ID
 * @param event_data 事件携带的数据
*/
static void event_handler(void* arg, esp_event_base_t event_base,
                                int32_t event_id, void* event_data)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        if(s_ssid_value[0] != 0)
            esp_wifi_connect();
    } else if (event_base == WIFI_EVENT && event_id ==
WIFI_EVENT_STA_DISCONNECTED) {
        //WIFI 断开连接后，再次发起连接
        if(!s_is_smartconfig)
            esp_wifi_connect();
        //清除连接标志位
        xEventGroupClearBits(s_wifi_event_group, CONNECTED_BIT);
    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP)
{
        //获取到 IP，置位连接事件标志位
        xEventGroupSetBits(s_wifi_event_group, CONNECTED_BIT);
    } else if (event_base == SC_EVENT && event_id == SC_EVENT_SCAN_DONE) {
        //smartconfig 扫描完成
        ESP_LOGI(TAG, "Scan done");
    } else if (event_base == SC_EVENT && event_id == SC_EVENT_FOUND_CHANNEL)
{
        //smartconfig 找到对应的通道
        ESP_LOGI(TAG, "Found channel");
    } else if (event_base == SC_EVENT && event_id == SC_EVENT_GOT_SSID_PSWD)
{
        //smartconfig 获取到 SSID 和密码
        ESP_LOGI(TAG, "Got SSID and password");
```

```
        smartconfig_event_got_ssid_pswd_t *evt =
(smartconfig_event_got_ssid_pswd_t *)event_data;
        wifi_config_t wifi_config;
        uint8_t ssid[33] = { 0 };
        uint8_t password[65] = { 0 };
        //从 event_data 中提取 SSID 和密码
        bzero(&wifi_config, sizeof(wifi_config_t));
        memcpy(wifi_config.sta.ssid, evt->ssid,
sizeof(wifi_config.sta.ssid));
        memcpy(wifi_config.sta.password, evt->password,
sizeof(wifi_config.sta.password));
        wifi_config.sta.bssid_set = evt->bssid_set;
        if (wifi_config.sta.bssid_set == true) {
            memcpy(wifi_config.sta.bssid, evt->bssid,
sizeof(wifi_config.sta.bssid));
        }
        memcpy(ssid, evt->ssid, sizeof(evt->ssid));
        memcpy(password, evt->password, sizeof(evt->password));
        ESP_LOGI(TAG, "SSID:%s", ssid);
        ESP_LOGI(TAG, "PASSWORD:%s", password);
        snprintf(s_ssid_value,33,"%s",(char*)ssid);
        snprintf(s_password_value,65,"%s",(char*)password);
        //重新连接 WIFI
        ESP_ERROR_CHECK( esp_wifi_disconnect() );
        ESP_ERROR_CHECK( esp_wifi_set_config(WIFI_IF_STA,
&wifi_config) );
        esp_wifi_connect();
    } else if (event_base == SC_EVENT && event_id == SC_EVENT_SEND_ACK_DONE)
{
        //smartconfig 已发起回应
        xEventGroupSetBits(s_wifi_event_group, ESPTOUCH_DONE_BIT);
    }
}
```

在事件处理回调函数中，包含了对 SmartConfig 的处理，比较关键的是 SC_EVENT_GOT_SSID_PSWD 和 SC_EVENT_SEND_ACK_DONE 事件，SC_EVENT_GOT_SSID_PSWD 事件表示已经获取到 SSID 和密码了，接下来我们可以发起连接。SC_EVENT_SEND_ACK_DON 事件表示 SmartConfig 完成，配网可以结束了，通知 SmartConfig 任务退出。

由此可见，esp-idf 对 SmartConfig 功能进行了高度封装，我们基本不用做复杂的处理就可以使用，十分方便，完整的代码请看 esp32-board/wifi_smartconfig，idf.py build+idf.py flash 烧录到开发板后，就可以运行。另外大家可以看下官方的 demo APP



左边是 V1 版本，右边是 V2 版本，在进入 APP 的时候可以选择，使用的时候，需要手机连接当前的 2.4G WiFi，然后输入密码，点击确定的时候就开始了，开发板上长按按键 3 秒，查看串口打印开始了 SmartConfig 即可松手，过一会就会自动的完成配网。