ESP-TOUCH
使用指南

版本 2.0

关于本手册

发布说明

日期	版本	发布说明
2015.12	V1.0	首次发布。
2016.04	V1.1	更新第 2 章和第 3 章。
2018.06	V2.0	更新第 3 章，增加对 ESP32 的支持。

文档变更通知

用户可通过 乐鑫官网 订阅技术文档变更的电子邮件通知。

证书下载

用户可以通过 乐鑫官网 下载产品证书。

目录

1. ESP-TOUCH 技术原理简介

乐鑫自主研发的 ESP-TOUCH 协议采用的是 Smart Config（智能配置）技术，帮助用户将采用 ESP8266EX 和 ESP32 的设备（以下简称“设备”）连接至 Wi-Fi 网络。用户只需在手机上进行简单操作即可实现智能配置。整个过程如下图所示：

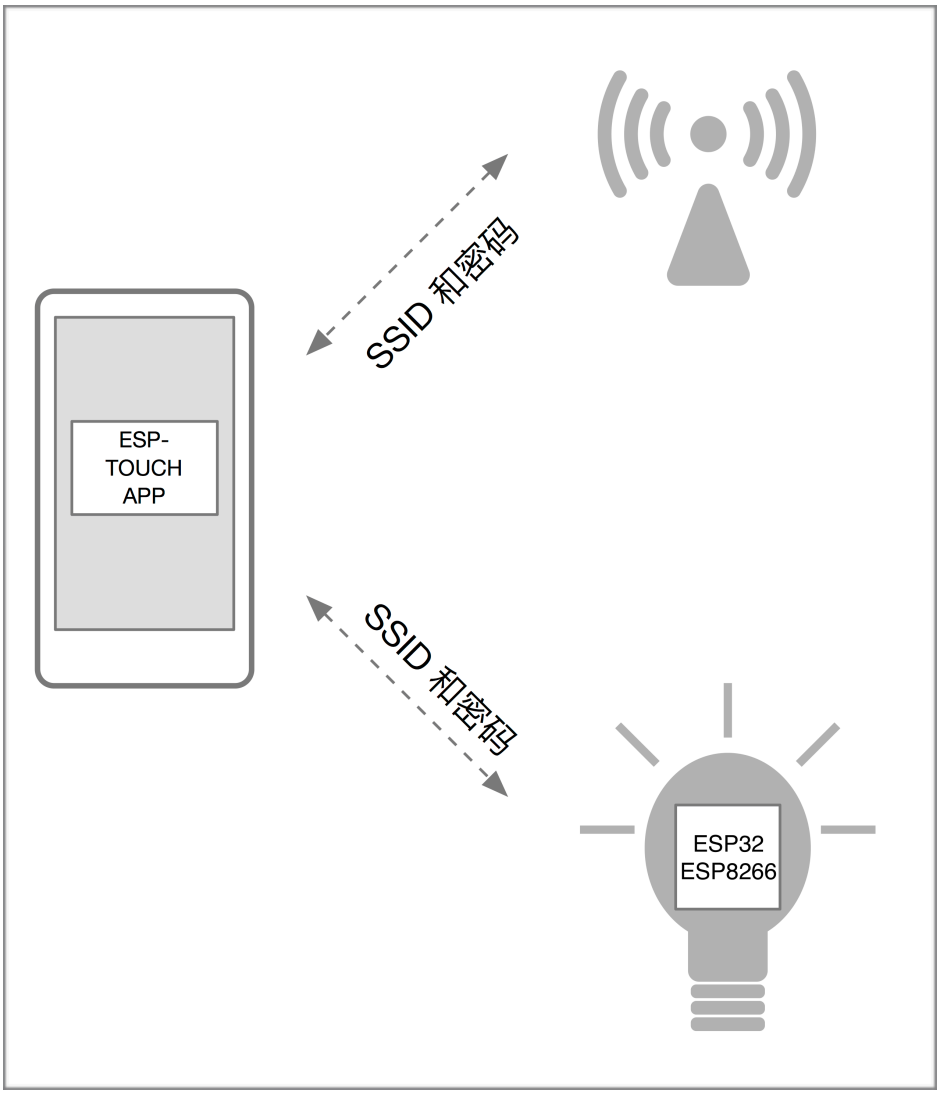

图 1-1. ESP-TOUCH 技术信息传输过程

由于设备在一开始并未连接至网络，ESP-TOUCH App 无法直接向设备发送信息。通过 ESP-TOUCH 通信协议，具备 Wi-Fi 网络接入能力的设备（例如智能手机）就可以向接入点 (AP) 发送一系列 UDP 包，其中每一包的长度（即 Length 字段）都按照 ESP-TOUCH 通信协议进行编码，SSID 和密码就包含在 Length 字段中，随后设备便可以获得并解析出所需的信息。数据包结构如图 1-2 所示：


6
6
2
3
5
Variable
4
DA
SA
Length
LLC
SNAP
DATA
FCS
包含设备可以获得的
SSID 和密钥信息

图 1-2. 数据包结构

2. ESP-TOUCH 使用方式

2.1. ESP-TOUCH 功能概述

ESP8266 OS 版和非 OS 版 SDK 以及 ESP32 IDF 均支持 ESP-TOUCH。

上述 SDK 和 IDF 还集成微信 AirKiss 协议，用户可使用 ESP-TOUCH App 或微信客户端配置设备。

> 📖 **说明：**
>
> *ESP-TOUCH App* 源码的下载地址为：*https://github.com/espressifAPP*。

2.2. ESP-TOUCH 操作说明

1. 准备一台支持 ESP-TOUCH 的设备，开启配置功能；
2. 将手机连接至路由器；
3. 打开安装在手机上的 ESP-TOUCH App；
4. 在 App 界面输入路由器的 SSID 和密码（若路由不加密则密码为空），开始连接。

> 📖 **说明：**
>
> - 设备与手机建立链路所需的时间和两者之间的距离有关，若距离很近，仅需数秒即可完成。
> - 在开启设备端 *ESP-TOUCH* 智能配置功能之前，请确保路由器已经开启。由于开启此功能后，设备会先扫描周围的 *AP* 信息，如果路由器没有开启，则无法获取周围的 *AP* 列表。
> - *ESP-TOUCH App* 端发送序列有超时限制，即如果没有在规定的时间配置上路由，*App* 会返回配置失败信息（参考 *App* 源码）。设备端也会有超时限制，从设备锁定通道开始计时到获取 *SSID* 和密码信息，如果达到超时限制但并未获取到 *SSID* 和密码信息，设备会自动重新开始一次 *Smart Config*。用户可以通过 `esptouch_set_timeout(uint8 time_s)` 或 `esp_smartconfig_set_timeout(uint8 time_s)` 设置超时时间。
> - *Smart Config* 过程中设备需要开启 *Sniffer* 模式，所以在 *ESP8266* 上 *Station* 和 *soft-AP* 模式都是关闭的，不要调用其他相关 *API*。*ESP32* 不受此限制，可以同时开启 *Sniffer* 和 *Station* 模式。
> - 配置结束后，发送端获取设备的 *IP*，设备端也会返回发送端的 *IP*，如果用户需要自定义发送端和设备端的信息交互，可以使用此 *IP* 信息进行局域网通信。
> - 如果路由器设置了*"AP* 隔离*"*，会出现设备连接路由器成功，但 *App* 未收到连接成功的提示的现象。
> - 支持手机同时配置多个设备到同一路由器上，配置时 *App* 端选择接收多个返回结果。
> - *ESP8266* 和 *ESP32* 不支持 *5G* 和 *11AC* 模式，因此不支持 *Smart Config* 连接 *5G* 或 *11AC* 模式的路由器。

3. API 开发

调用如下 API 可实现 ESP-TOUCH 配置，请尽量使用最新的 App 和固件版本。SDK 和 IDF 工程中提供使用 ESP-TOUCH 的操作示例供参考。

3.1. ESP8266 API

3.1.1. smartconfig_start

功能：配置设备连接 AP。

⚠ **注意：**

- 调用本接口前，设备需被设置成 *Station* 模式。
- `smartconfig_start` 未完成之前不可重复执行 `smartconfig_start` 函数。如果需要停止 *Smart Config* 过程，则需调用 `smartconfig_stop` 函数。

函数定义：`bool smartconfig_start(sc_callback_t cb, uint8 log)`

参数：

`sc_callback_t cb`	设备在状态发生改变时会产生回调，传入回调函数的参数为 sc_status status 和 struct station_config 类型的指针变量。 传入回调函数的参数 status 表示 Smart Config 状态: • 当 status 为 SC_STATUS_GETTING_SSID_PSWD 时，参数 void *pdata 为 sc_type 类型的指针变量，用户可以根据此参数知道此次配置过程是 AirKiss 还是 ESP-TOUCH。 • 当 status 为 SC_STATUS_LINK 时，参数 void *pdata 为 struct station_config 类型的指针变量。 • 当 status 为 SC_STATUS_LINK_OVER 时，参数 void *pdata 是移动端的 IP 地址的指针，4 个字节。（仅支持在 ESP-TOUCH 方式下，其他方式则为 NULL）。 • 当 status 为其他状态时，参数 void *pdata 为 NULL。

uint8 log	“1”, UART 打印连接过程；否则 UART 仅打印连接结果。 例如： • smartconfig_start(smartconfig_done,1)：串口将打印连接过程中的 DEBUG 信息。 • smartconfig_start(smartconfig_done)：串口不打印 DUBUG 信息，只打印连接结果。

返回：

TRUE	成功
FALSE	失败

示例：

```
void ICACHE_FLASH_ATTR
smartconfig_done(sc_status status, void *pdata)
{
    switch(status) {
        case SC_STATUS_WAIT:
            os_printf("SC_STATUS_WAIT\n");
            break;
        case SC_STATUS_FIND_CHANNEL:
            os_printf("SC_STATUS_FIND_CHANNEL\n");
            break;
        case SC_STATUS_GETTING_SSID_PSWD:
            os_printf("SC_STATUS_GETTING_SSID_PSWD\n");
            sc_type *type = pdata;
            if (*type == SC_TYPE_ESPTOUCH) {
                os_printf("SC_TYPE:SC_TYPE_ESPTOUCH\n");
            } else {
                os_printf("SC_TYPE:SC_TYPE_AIRKISS\n");
            }
            break;
        case SC_STATUS_LINK:
            os_printf("SC_STATUS_LINK\n");
            struct station_config *sta_conf = pdata;
            wifi_station_set_config(sta_conf);
            wifi_station_disconnect();
            wifi_station_connect();
            break;
```

```
                case SC_STATUS_LINK_OVER:
                    os_printf("SC_STATUS_LINK_OVER\n");
                    if (pdata != NULL) {
                        uint8 phone_ip[4] = {0};
                        memcpy(phone_ip, (uint8*)pdata, 4);
                        os_printf("Phone ip: %d.%d.%d.
                            %d\n",phone_ip[0],phone_ip[1],phone_ip[2],phone_ip[3]);
                    }
                    smartconfig_stop();
                    break;
            }
        }
        smartconfig_start(smartconfig_done);
```

3.1.2. smartconfig_stop

功能：停止配置设备，并且释放调用 smartconfig_start 函数所申请的内存。

> 📖 **说明：**
>
> 连上 *AP* 后，可调用本接口释放内存。

函数定义：`bool smartconfig_stop(void)`

参数：Null

返回：

TRUE	成功
FALSE	失败

3.1.3. smartconfig_set_type

功能：设置 smartconfig_start 模式的协议类型。

> 📖 **说明：**
>
> 如需调用本接口，请在 *smartconfig_start* 之前调用。

函数定义：`bool smartconfig_set_type(sc_type type)`

参数：

```
    typedef enum {
        SC_TYPE_ESPTOUCH = 0,
        SC_TYPE_AIRKISS,
        SC_TYPE_ESPTOUCH_AIRKISS,
```

```
    } sc_type;
```

返回：

TRUE	成功
FALSE	失败

3.1.4. 结构体

```
typedef enum {
    SC_STATUS_WAIT = 0,
    SC_STATUS_FIND_CHANNEL = 0,
    SC_STATUS_GETTING_SSID_PSWD,
    SC_STATUS_LINK,
    SC_STATUS_LINK_OVER,
} sc_status;
```

> ⚠ **注意：**
>
> *SC_STATUS_FIND_CHANNEL* 状态：设备处于扫描信道的过程，这时才可开启 *App* 进行配对连接。

```
typedef enum {
    SC_TYPE_ESPTOUCH = 0,
    SC_TYPE_AIRKISS,
    SC_TYPE_ESPTOUCH_AIRKISS,
} sc_type;
```

3.2. ESP32 API

3.2.1. esp_smartconfig_start

功能：配置设备连接 AP。

> ⚠ **注意：**
>
> - 调用本接口前，设备需被设置成 *Station* 或者 *Station + soft-AP* 模式。
> - `esp_smartconfig_start` 未完成之前不可重复执行 `esp_smartconfig_start` 函数。如果需要停止 *Smart Config* 过程，则需调用 `esp_smartconfig_stop` 函数。

函数定义：`esp_err_t esp_smartconfig_start(sc_callback_t cb, ...)`

参数：

sc_callback_t cb	设备在状态发生改变时会产生回调，传入回调函数的参数为 smartconfig_status_t status 和 wifi_config_t 类型的指针变量。 传入回调函数的参数 status 表示 Smart Config 状态: • 当 status 为 SC_STATUS_GETTING_SSID_PSWD 时，参数 void *pdata 为 smartconfig_type_t 类型的指针变量，用户可以根据此参数知道此次配置过程是 AirKiss 还是 ESP-TOUCH。 • 当 status 为 SC_STATUS_LINK 时，参数 void *pdata 为 wifi_config_t 类型的指针变量。 • 当 status 为 SC_STATUS_LINK_OVER 时，参数 void *pdata 是移动端的 IP 地址的指针，4 个字节。（仅支持在 ESP-TOUCH 方式下，其他方式则为 NULL）。 • 当 status 为其他状态时，参数 void *pdata 为 NULL。
...	“1”, UART 打印连接过程；否则 UART 仅打印连接结果。 例如： • esp_smartconfig_start(smartconfig_done,1)：串口将打印连接过程中的 DEBUG 信息。 • esp_smartconfig_start(smartconfig_done)：串口不打印 DUBUG 信息，只打印连接结果。

返回：

ESP_OK	成功
others	失败

示例：

```
void sc_callback(smartconfig_status_t status, void *pdata)
{
    switch(status) {
        case SC_STATUS_WAIT:
            printf("SC_STATUS_WAIT\n");
            break;
        case SC_STATUS_FIND_CHANNEL:
            printf("SC_STATUS_FIND_CHANNEL\n");
            break;
        case SC_STATUS_GETTING_SSID_PSWD:
            printf("SC_STATUS_GETTING_SSID_PSWD\n");
```

```
                    smartconfig_type_t *type = pdata;
                    if (*type == SC_TYPE_ESPTOUCH) {
                        printf("SC_TYPE:SC_TYPE_ESPTOUCH\n");
                    } else {
                        printf("SC_TYPE:SC_TYPE_AIRKISS\n");
                    }
                    break;
                case SC_STATUS_LINK:
                    printf("SC_STATUS_LINK\n");
                    wifi_config_t *wifi_conf = pdata;
                    esp_wifi_disconnect();
                    esp_wifi_set_config(ESP_IF_WIFI_STA, wifi_conf);
                    esp_wifi_connect();
                    break;
                case SC_STATUS_LINK_OVER:
                    printf("SC_STATUS_LINK_OVER\n");
                    if (pdata != NULL) {
                        uint8 phone_ip[4] = {0};
                        memcpy(phone_ip, (uint8*)pdata, 4);
                        printf("Phone ip: %d.%d.%d.
                            %d\n",phone_ip[0],phone_ip[1],phone_ip[2],phone_ip[3]);
                    }
                    esp_smartconfig_stop();
                    break;
            }
        }
        esp_smartconfig_start(sc_callback);
```

3.2.2. esp_smartconfig_stop

功能：停止配置设备，并且释放调用 esp_smartconfig_start 函数所申请的内存。

> 📖 **说明：**
>
> 连上 *AP* 后，可调用本接口释放内存。

函数定义：`esp_err_t esp_smartconfig_stop(void)`

参数：Null

返回：

ESP_OK	成功
others	失败

3.2.3. esp_smartconfig_set_timeout

功能：设置 Smart Config 的超时时间。

> 📖 **说明：**
>
> - 如需调用本接口，请在 *esp_smartconfig_start* 之前调用。
> - 超时时间是从 *SC_STATUS_FIND_CHANNEL* 状态开始计算，超时之后重新开始 *Smart Config*。

函数定义： `esp_err_t esp_smartconfig_set_timeout(uint8_t time_s)`

参数：

time_s	超时时间，范围 15 ~ 255s。

返回：

`ESP_OK`	成功
`others`	失败

3.2.4. esp_smartconfig_set_type

功能：设置 esp_smartconfig_start 模式的协议类型。

> 📖 **说明：**
>
> 如需调用本接口，请在 *esp_smartconfig_start* 之前调用。

函数定义： `esp_err_t esp_smartconfig_set_type(smartconfig_type_t type)`

参数：

```
typedef enum {
    SC_TYPE_ESPTOUCH = 0,
    SC_TYPE_AIRKISS,
    SC_TYPE_ESPTOUCH_AIRKISS,
} smartconfig_type_t;
```

返回：

`ESP_OK`	成功
`others`	失败

3.2.5. esp_smartconfig_fast_mode

功能：设置 Smart Config 快速模式是否使能。

> 📖 **说明：**
>
> - 如需调用本接口，请在 *esp_smartconfig_start* 之前调用；
> - 快速模式需要配合对应的 *App* 使用。

函数定义：`esp_err_t esp_smartconfig_fast_mode(bool enable)`

参数：

enable	True：使能快速模式，False：禁用快速模式。

返回：

`ESP_OK`	成功
`others`	失败

3.2.6. 类型

```
Typedef void (*sc_callback_t)(smartconfig_status_t status, void *pdata)
```

功能：传入 `esp_smartconfig_start` 的回调函数。

参数：

smartconfig_status_t	设备的状态。
Void*	• 当 status 为 SC_STATUS_GETTING_SSID_PSWD 时，参数 void *pdata 为 smartconfig_type_t 类型的指针变量，用户可以根据此参数知道此次配置过程是 AirKiss 还是 ESP-TOUCH。 • 当 status 为 SC_STATUS_LINK 时，参数 void *pdata 为 wifi_config_t 类型的指针变量。 • 当 status 为 SC_STATUS_LINK_OVER 时，参数 void *pdata 是移动端的 IP 地址的指针，4 个字节。（仅支持在 ESP-TOUCH 方式下，其他方式则为 NULL）。 • 当 status 为其他状态时，参数 void *pdata 为 NULL。

3.2.7. 结构体

```
typedef enum {
    SC_STATUS_WAIT = 0,
    SC_STATUS_FIND_CHANNEL = 0,
    SC_STATUS_GETTING_SSID_PSWD,
    SC_STATUS_LINK,
    SC_STATUS_LINK_OVER,
} smartconfig_status_t;
```

> ⚠ **注意：**
>
> *SC_STATUS_FIND_CHANNEL* 状态：设备处于扫描信道的过程，这时才可开启 *App* 进行配对连接。

```
typedef enum {
    SC_TYPE_ESPTOUCH = 0,
    SC_TYPE_AIRKISS,
    SC_TYPE_ESPTOUCH_AIRKISS,
} smartconfig_type_t;
```

4. ESP-TOUCH 通信协议性能分析

ESP-TOUCH 技术中的通信模型可以抽象为某种错误率的单向的信道，但这种错误率又根据带宽的不同而有所不同。20M 带宽的包错误率为 0 ~ 5%，40M 带宽的包错误率为 0 ~ 17%。假设所需要传递信息的最大长度为 104 Byte。在这种情况下，如果不采用纠错算法，就很难保证在有限次数内完成信息的发送。

ESP-TOUCH 采用了累积纠错算法来保证在有限次内完成传输过程。累积纠错算法的理论基础为：多轮数据发送过程中，在同一位数据上发生错误的概率是很低的。因此可以累积多轮的数据传递结果进行分析，其中一轮中某一位错误数据有很大的概率能其它轮中找到其对应的正确值，这样就能保证在有限次内完成信息的发送。信息的成功率可抽象为公式：$[1-[1-p]^k]^l$（p：包的成功率，k：循环发送的轮数，l：发送信息的长度）。

假定需要传递信息的长度为 104 Byte 和 72 Byte，我们计算了在最坏的情况下，即 p =0.95 (20M) 和 p =0.83 (40M)。

使用累积纠错算法与不使用累积纠错算法信息发送成功的概率与发送次数的关系，结果如下表所示：

表 4-1. ESP-TOUCH 纠错能力分析（20 MHz 宽带）

发送轮数	信息长度：104 Byte		信息长度：72 Byte	
	发送时间 (s)	成功率	发送时间 (s)	成功率
1	4.68	0.0048	3.24	0.0249
2	9.36	0.771	6.48	0.835
3	14.04	0.987	9.72	0.991
4	18.72	0.9994	12.9	0.9996
5	23.40	0.99997	16.2	0.99998
6	28.08	0.999998	19.4	0.99999

表 4-2. ESP-TOUCH 纠错能力分析（40 MHz 宽带）

发送轮数	信息长度：104 Byte		信息长度：72 Byte	
	发送时间 (s)	成功率	发送时间 (s)	成功率
1	4.68	3.84e-9	3.24	1.49e-6
2	9.36	0.0474	6.48	0.121
3	14.04	0.599	9.72	0.701
4	18.72	0.917	12.9	0.942
5	23.40	0.985	16.2	0.989
6	28.08	0.997	19.4	0.998

乐鑫 IoT 团队
www.espressif.com

免责申明和版权公告

本文中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。