WS63V100 特性开发

说明书

文档版本　02

发布日期　2024-06-27

前言

概述

本文档详细的描述了 WS63V100 相关特性的应用场景、实现原理及接口说明，方便读者了解并使用相关特性。

读者对象

本文档主要适用于以下工程师：

- 软件开发工程师
- 技术支持工程师
- 硬件开发工程师

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
WS63	V100

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
02	2024-06-27	• 更新“5.1 概述”、“5.2 开发流程”小节内容。 • 更新“6.5 使用示例”小节内容。
01	2024-04-10	第一次正式版本发布。 更新“9.4 接口说明”小节内容。
00B02	2024-03-29	• 更新“2.5 使用示例”小节内容。 • 更新“6.5 使用示例”小节内容。
00B01	2024-03-15	第一次临时版本发布

目 录

1 混杂模式特性说明

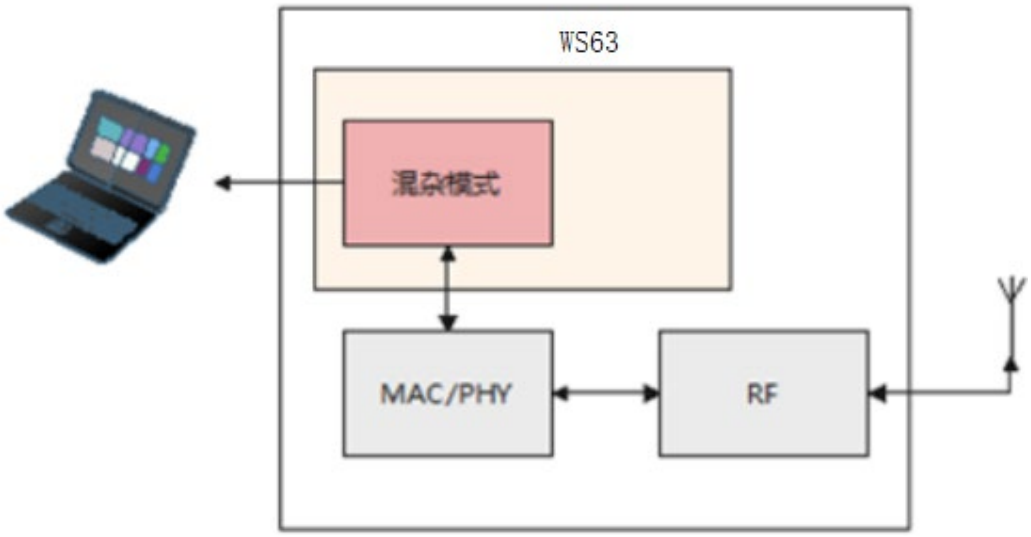
1.1 概述

混杂模式特性用于抓取所有经过本设备的管理帧/数据帧，并将抓取到的报文保存到文件中，可通过抓包软件例如：WireShark、OmniPeek 打开并查看报文信息。

1.2 应用场景

WiFi 混杂模式开启后，会抓取周边 AP 和 STA 之间的单播/组播的管理帧/数据帧，典型应用场景之一是支持作为抓包网卡。如下图，WS63 芯片作为抓包网卡，通过串口与 PC 连接，抓取 PC 周边报文。

图1-1 混杂模式进行空口抓包

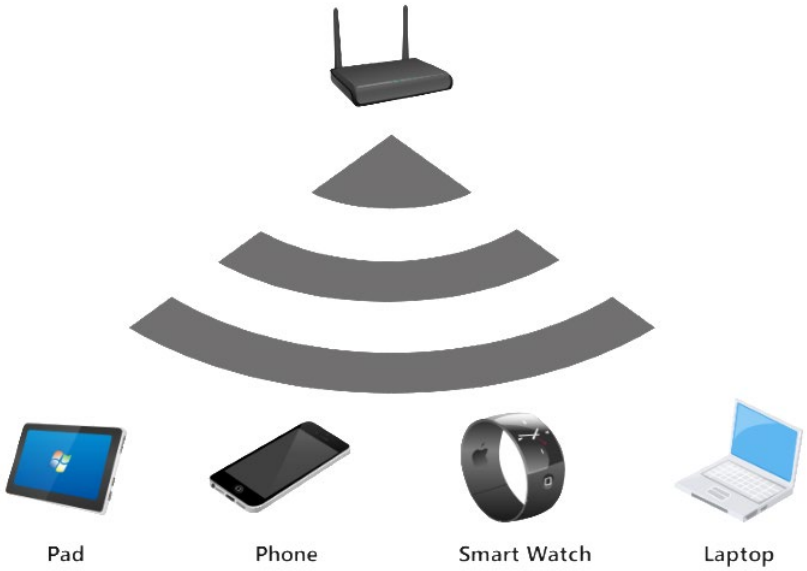


1.3 实现原理

无线网络信号在传播过程中是以发射点为中心，像波纹一样往外辐射。理论上来讲，如果一个接收器处于无线信号经过的地方，它可以“听到”任何经过它的信号，只是它可能“听不懂”(无法解析报文内容)。

以下图为例，Phone 与 Smart Watch 通信，Laptop 完全有能力从空口监听他们的通信。空口抓包就是基于这个原理工作的。如果我们想要抓某个嵌入式设备的无线报文，只需在它附近运行一个具有监听功能的无线设备。

图1-2 混杂模式



1.4 接口说明

1.4.1 混杂接口声明

```
/**
 * @if Eng
 * @brief   Type of WiFi interface.
 * @else
 * @brief   Type of WiFi interface。
 * @endif
 */
typedef enum {
    IFTYPE_STA,             /*!< @if Eng STAION.
                                 @else STAION。 @endif */
    IFTYPE_AP,              /*!< @if Eng HOTSPOT.
                                 @else HOTSPOT。 @endif */
    IFTYPE_P2P_CLIENT,   /*!< @if Eng P2P CLIENT.
                                 @else P2P CLIENT。 @endif */
    IFTYPE_P2P_GO,       /*!< @if Eng P2P GO.
                                 @else P2P GO。 @endif */
```

```
    IFTYPE_P2P_DEVICE,   /*!< @if Eng P2P DEVICE.
                                @else P2P DEVICE。 @endif */
    IFTYPES_BUTT
} wifi_if_type_enum;

/**
 * @if Eng
 * @brief   Struct of frame filter config in monitor mode.
 * @else
 * @brief   混杂模式报文接收过滤设置。
 * @endif
 */
typedef struct {
    int8_t mdata_en   : 1;    /*!< @if Eng get multi-cast data frame flag.
                                @else 使能接收组播(广播)数据包。 @endif */
    int8_t udata_en   : 1;    /*!< @if Eng get single-cast data frame flag.
                                @else 使能接收单播数据包。 @endif */
    int8_t mmngt_en   : 1;    /*!< @if Eng get multi-cast mgmt frame flag.
                                @else 使能接收组播(广播)管理包。 @endif */
    int8_t umngt_en   : 1;    /*!< @if Eng get single-cast mgmt frame flag.
                                @else 使能接收单播管理包。 @endif */
    int8_t custom_en : 1;    /*!< @if Eng get beacon/probe response flag.
                                @else 使能接收beacon/probe request包。 @endif */
    int8_t resvd       : 3;    /*!< @if Eng reserved bits.
                                @else 保留字段。 @endif */
} wifi_ptype_filter_stru;

/**
 * @if Eng
 * @brief   Set monitor mode.
 * @param   [in]   iftype Interface type.
 * @param   [in]   enable Enable(1) or disable(0).
 * @param   [in]   filter Filtered frame type enum.
 * @retval EXT_WIFI_OK            Execute successfully.
 * @retval EXT_WIFI_FAIL          Execute failed.
 * @else
 * @brief   设置混杂模式。
 * @param   [in]   iftype 接口类型。
```

```
 * @param   [in]   enable 开启/关闭。
 * @param   [in]   filter 过滤列表。
 * @retval EXT_WIFI_OK    成功。
 * @retval EXT_WIFI_FAIL 失败。
 * @endif
 */
errcode_t wifi_set_promis_mode(wifi_if_type_enum iftype, int32_t enable, const
wifi_ptype_filter_stru *filter);
```

1.4.2 开启和关闭混杂

命令格式：AT+CCPRIV=$vap,set_monitor,$switch,$val1,$val2,$val3,$val4

参数说明：

- $vap：表示需要维测的 vap 名字，通常为 wlan0。
- $switch：表示功能开、关、或者暂停，对应 1、0、2。
- $val1：表示广播/组播数据帧过滤开关，对应 0、1，0 表示过滤，1 表示不过滤。
- $val2：表示单播数据帧过滤开关，对应 0、1，0 表示过滤，1 表示不过滤。
- $val3：表示广播/组播管理帧过滤开关，对应 0、1，0 表示过滤，1 表示不过滤。
- $val4：表示单播管理帧过滤开关，对应 0、1，0 表示过滤，1 表示不过滤。

命令示例：

- 开启所有帧上报：AT+CCPRIV=wlan0,set_monitor,1,1,1,1,1
- 查看混杂收包统计：AT+CCPRIV=wlan0,set_monitor,2 （PS：命令下发成功后，会在 DebugKits 工具中打印收包统计信息。）
- 关闭混杂：AT+CCPRIV=wlan0,set_monitor,0

1.5 使用示例

步骤 1 执行 1.4 接口说明相关命令后，按需求开启对应帧过滤开关，例如所有帧上报：AT+CCPRIV=wlan0,set_monitor,1,1,1,1,1

步骤 2 若想查看开启混杂后的收包计数统计，可输入命令 AT+CCPRIV=wlan0,set_monitor,2 后，在 DebugKits 工具中查看。

----结束

2 动态国家码特性说明

2.1 概述

动态国家码特性用于支持全球发货场景，根据设备预制的国家信息，自动调整发射功率表，以符合全球各地区对发射功率的法律规范。

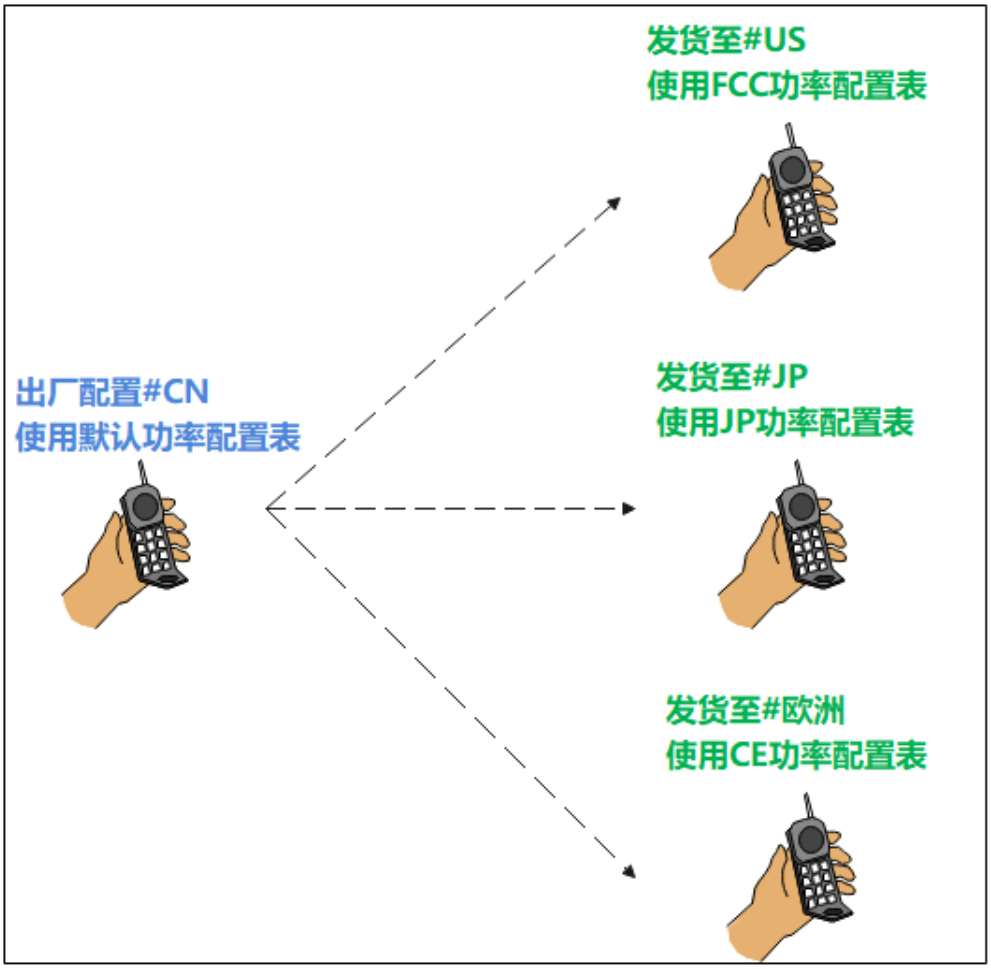
2.2 应用场景

在上网问题处理中，经常会碰到来自异国的设备出现扫描/连接/协商速率异常的情况，很多情况与 802.11d 协议（/国家码设置）相关。

国家码用来标识无线设备所在的国家，不同国家码规定了不同的无线设备射频特性，包括 AP 的发送功率、支持的信道等。配置国家码是为了使无线设备的射频特性符合不同国家或区域的法律法规要求。在第一次配置 WLAN 设备时，必须配置正确的国家码，以确保不违反当地的法律法规。

动态国家码提供一种配置方式，客户能够通过设置国家码，调整到国家对应大区（中国、亚太、北美、欧洲）的发射功率，从而符合法律规范

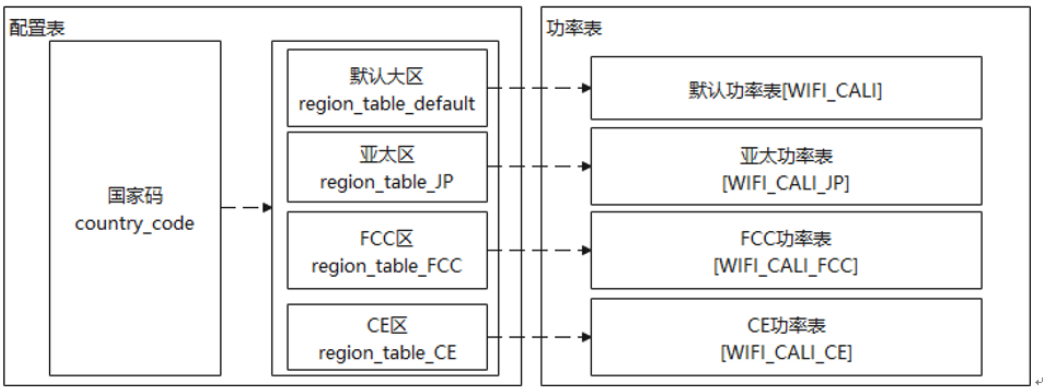
图2-1 动态国家码应用场景



2.3 实现原理

通过配置文件映射国家与大区的关系，每个国家码对应一个大区，每个大区对应一套功率表，国家码变动时，根据重新配置对应的功率表。

图2-2 动态国家码设计原理图



2.4 接口说明

2.4.1 国家码接口声明

```
/**
* @ingroup  soc_wifi_basic
* @brief  Set country code.CNcomment:设置国家码.CNend
*
* @par Description:
*              Set country code(two uppercases).CNcomment:设置国家码，由两个大写字符组成.CNend
*
* @attention  1.Before setting the country code, you must call uapi_wifi_init to complete the initialization.
*              CNcomment:设置国家码之前，必须调用uapi_wifi_init初始化完成.CNend\n
*              2.cc_len should be greater than or equal to 3.CNcomment:cc_len应大于等于3.CNend
* @param  cc                  [IN]     Type  #const char *, country code.CNcomment:国家码.CNend
* @param  cc_len              [IN]     Type  #unsigned char, country code length.CNcomment:国家码长度.CNend
*
* @retval #EXT_WIFI_OK  Excute successfully
* @retval #Other        Error code
```

```
* @par Dependency:
*                  @li soc_wifi_api.h: WiFi API
* @see   NULL
* @since
*/
td_s32 uapi_wifi_set_country(const td_char *cc, td_u8 cc_len);

/**
* @ingroup   soc_wifi_basic
* @brief   Get country code.CNcomment:获取国家码.CNend
*
* @par Description:
*                  Get country code.CNcomment:获取国家码，由两个大写字符组成.CNend
*
* @attention   1.Before getting the country code, you must call uapi_wifi_init to complete the initialization.
*                  CNcomment:获取国家码之前，必须调用uapi_wifi_init初始化完成.CNend
* @param   cc                      [OUT]       Type   #char *, country code.CNcomment:国家码.CNend
* @param   len                     [IN/OUT]   Type   #int *, country code length.CNcomment:国家码长度.CNend
*
* @retval #EXT_WIFI_OK   Excute successfully
* @retval #Other                 Error code
* @par Dependency:
*                  @li soc_wifi_api.h: WiFi API
* @see   NULL
* @since
*/
td_s32 uapi_wifi_get_country(td_char *cc, td_u8 *len);
```

2.4.2 设置和读取国家码

```
AT+CC=&country
AT+CC?
```

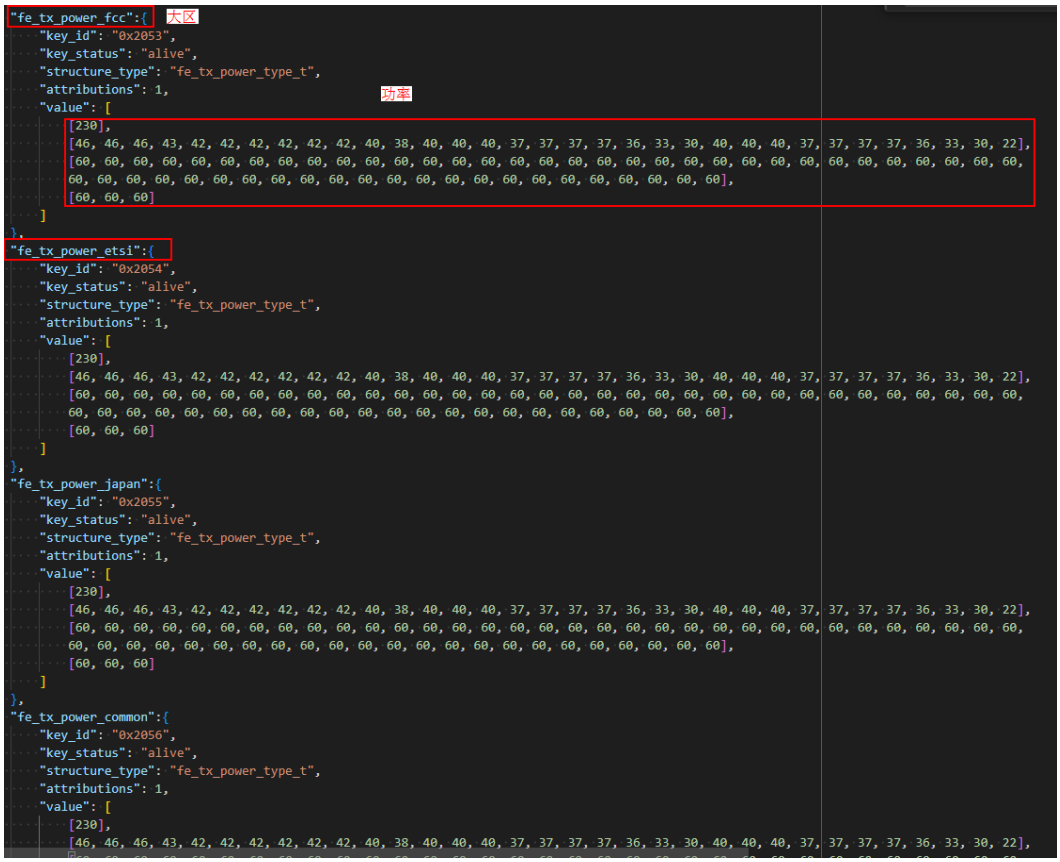
说明

- $COUNTRY：国家码，可配置范围：CN,JP,US,CA,KHRU,AU,MY,ID,TR,PL,FR,PT,IT,DE,ES,AR,ZA,MA,PH,TH,GB,CO,MX,EC,PE,CL,SA,EG,AE.

2.5 使用示例

加载驱动时，根据 nv 配置文件配置大区功率，如图 2-3 所示。

图2-3 nv 国家码配置示例



3 中继特性说明

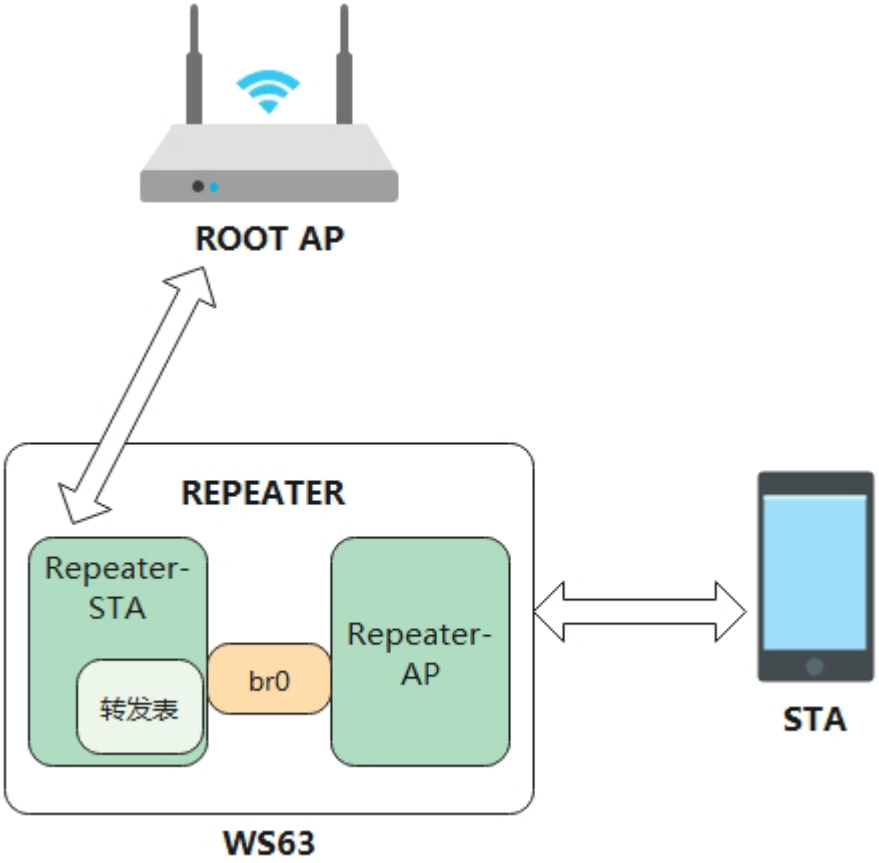
3.1 概述

中继特性主要用于无线网络连接中，通过对报文的转发，实现远距离无线通信，达到扩大网络覆盖范围、降低网络部署成本的目的。

3.2 应用场景

中继特性通过创建一个 Repeater-STA 端口与 ROOT AP 建立连接，创建一个 Repeater-AP 端口为其它 STA 提供服务，其工作原理为：Repeater-STA 和 Repeater-AP 之间支持报文转发，从而进一步实现 ROOT AP 与 STA 的报文交互、完成网络业务的功能。Repeater-STA 支持与一个 ROOT AP 接入，Repeater-AP 支持最多 5 个设备同时接入。

图3-1 中继场景



3.3 实现原理

STA 发给 ROOT AP 的报文经过 Repeater 时，报文中的源 MAC 地址与源 IP 地址的映射关系会被记录进入转发表，并且将报文中的源 MAC 修改为 Repeater-STA 的 MAC 发送给 ROOT AP。

ROOT AP 发送给 STA 的报文经过 Repeater 时， 会根据目的 IP 在转发表中查询，更改报文中的目的 MAC 地址，将报文转发到正确的 STA 中。

图3-2 Repeater-STA 上行数据场景

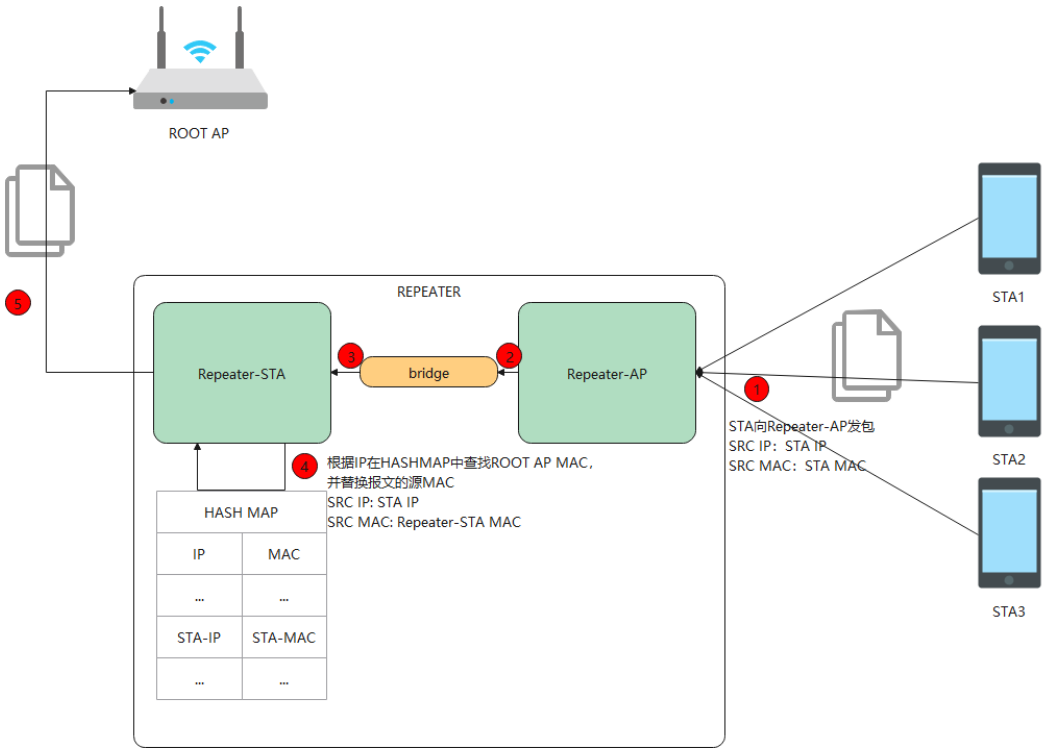
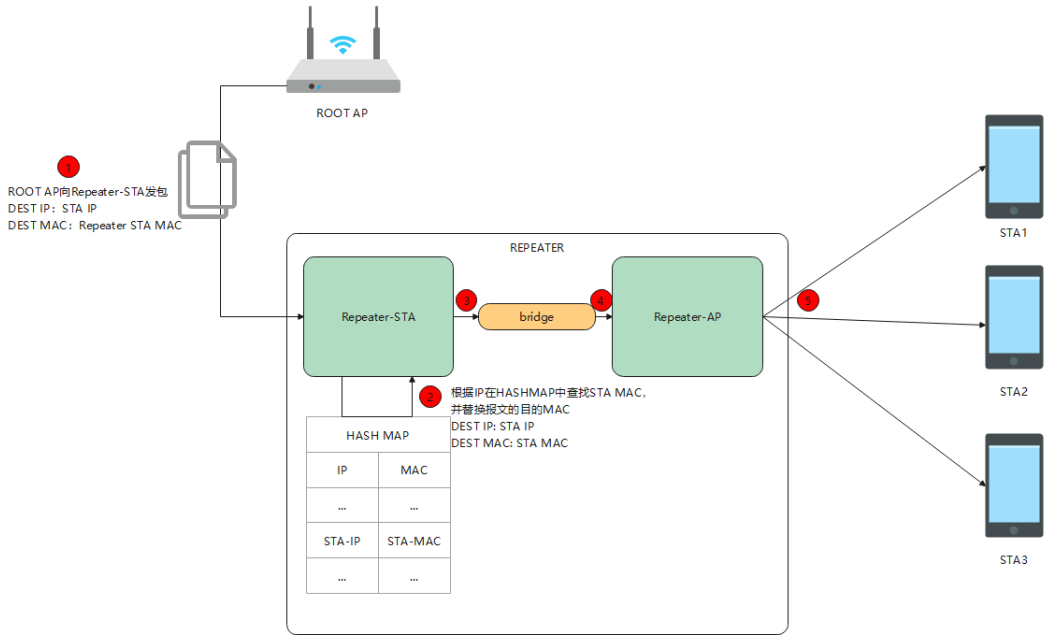


图3-3 Repeater-STA 下行数据场景



3.4 接口说明

暂无

3.5 使用示例

步骤 1 创建 ap0 端口。

```
AT+STARTAP="my_ap",13,2,"12345678"
```

步骤 2 创建 wlan0 端口并关联上 ROOT AP。

```
AT+STARTSTA
AT+SCAN
AT+CONN="ROOTAP",,"12345678"
```

步骤 3 使能 repeater 功能。

```
AT+BRCTL=addbr
```

步骤 4 将 ap0 端口、wlan0 端口分别加入 repeater，分别作为 Repeater-AP 和 Repeater-STA。

```
AT+BRCTL=addif,wlan0
AT+BRCTL=addif,ap0
```

步骤 5 陪测 STA 关联 Repeater-AP 并获取 IP 地址，进行 ping 测试、Iperf 测试等。

步骤 6 测试完成，将 wlan0 端口、ap0 端口从 repeater 删除，并关闭 repeater 功能。

```
AT+BRCTL=delif,wlan0
AT+BRCTL=delif,ap0
AT+BRCTL=delbr
```

----结束

4 CSI 特性说明

4.1 概述

CSI 属于链路层数据，CSI 在 WiFi 研究领域指 Channel State Information，也就是通过接收到的 WiFi 信号来估计 WiFi 信号的传播信道的状况。

4.2 应用场景

目前人们可以从 CSI 里提取到很多信息，比如室内人数，人的走动，(如上图)，还可以扩大到分析人的心跳，敲击键盘，抽烟、喝水等动作，有研究机构甚至可以做到支持每个 WiFi 包的频偏，均衡星座图，时间标签等，继而做到从 CSI 到 CSI”，指的是从 Channel State Information 到 Chip State Information!

CSI 能以更细粒度表征信号，通过对不同子信道信号传输情况分别进行分析，CSI 可以尽可能避免多径效应与噪声的影响从而实现人体感知、精细的定位及细粒度动作的识别。

而在本文场景中，PHY 层将 L-LTF（non-HT)、HT-LTF、VHT-LTF 或 HE-LTF 的符号的信道状态信息（CSI）通过 MAC、HAL 传递给 HOST，HOST 过滤后上报软件应用层用于定位功能。

CSI 上报主要场景：由 MAC 获取 CSI 信息后，经过 HAL、DMAC、HMAC 不断抛事件最后到达 WAL 层，再经过 WAL 层上报数据进行解析分类最后传递到应用层，供应用层的算法（AoA 定位算法等）使用；

4.3 实现原理

CSI 定位方法是根据接收端多天线间收到讯号的相位差计算出收到讯号的入射角与飞行时间来进行定位，利用讯号相位的差异值与波长的比例来计算出讯号由传送端发射到接收端的到达角度，利用讯号相位的差异值与频率差异值的比例来计算出讯号由传送端发射到接收端的飞行时间。

藉由多天线接收到的讯号相位差计算讯号到达角度，使用不同天线之间的相位差进行 AoA 的计算。无线讯号中不同子窄波之间的频率差为已知的信息，利用子窄波之间的相位差进行换算则可以计算讯号的飞行时间。CSI 定位方法根据手机或 Tag 每个传送讯号来计算到达角度与飞行时间，利用讯号直接路径会有较小的讯号飞行时间的想法，筛选出直接路径的到达角度当作定位的依据，此外，根据 WiFi Sniffer 接收到手机或 Tag 的讯号强度来换算手机与基地台之间的距离，当 WiFi Sniffer 知道手机或 Tag 的相对角度与距离时，即可以进行定位。为了提升定位的准确率，CSI 定位方法分别使用三台以上的 WiFi Sniffer，根据每个 WiFi Sniffer 所计算的可能位置，导入权重分析的概念分别计算 Likelihood 值，利用三台以上的 WiFi Sniffer 所计算的可能位置进行校正，减少定位的误差而提升准确率。

4.4 接口说明

1. 添加用户命令

```
/**
 * @if Eng
 * @brief   Config CSI.
 * @param   [in]   ifname Interface which enable CSI, wlan0 or ap0.
 * @param   [in]   config CSI's configuration.
 * @retval ERRCODE_SUCC             Execute successfully.
 * @retval Other                    Execute failed.
 * @else
 * @brief   配置CSI。
 * @param   [in]   ifname 使能CSI，wlan0，ap的接口。
 * @param   [in]   config CSI配置。
```

```
 * @retval ERRCODE_SUCC             成功。
 * @retval Other                    失败。
 * @endif
 */
errcode_t wifi_set_csi_config(const int8_t *ifname, const csi_config_stru *config);

typedef struct {
    uint8_t user_index;                 /*!< @if Eng user index.
                                            @else  用户ID,取值范围0~3，最多4个用户。
@endif */
    uint8_t enable;                     /*!< @if Eng enable.
                                            @else CSI白名单用户开关。 @endif */
    uint8_t match_ta_ra_select;         /*!< @if Eng match_ta_ra_select.
                                            @else CSI白名单地址过滤类型 0 RA 1 TA。
@endif */
    uint8_t resv;                       /*!< @if Eng resv.
                                            @else  保留1字节对齐。 @endif */
    uint8_t mac_addr[WIFI_MAC_LEN]; /*!< @if Eng Mac address.
                                            @else MAC地址。 @endif */
    uint8_t frame_filter_bitmap;        /*!< @if Eng frame_filter_bitmap.
                                            @else  帧类型过滤具体参数。 @endif */
    uint8_t sub_type_filter_enable; /*!< @if Eng subTypeFilterEnable.
                                            @else  帧子类型过滤开关。 @endif */
    uint8_t sub_type_filter;            /*!< @if Eng subTypeFilter.
                                            @else  帧子类型过滤具体参数。 @endif */
    uint8_t ppdu_filter_bitmap;         /*!< @if Eng ppduFilterBitmap.
                                            @else ppdu format过滤具体参数。 @endif */
    uint16_t period;                    /*!< @if Eng period.
                                            @else CSI上报时间间隔。 @endif */
} csi_config_stru;
```

帧子类型过滤具体参数 4 位二进制数对应的十进制结果（如 1100 即为 12）

```
/* 管理帧子类型 */
typedef enum {
    WLAN_ASSOC_REQ              = 0,    /* 0000 */
    WLAN_ASSOC_RSP              = 1,    /* 0001 */
    WLAN_REASSOC_REQ            = 2,    /* 0010 */
    WLAN_REASSOC_RSP            = 3,    /* 0011 */
    WLAN_PROBE_REQ              = 4,    /* 0100 */
    WLAN_PROBE_RSP              = 5,    /* 0101 */
    WLAN_TIMING_AD              = 6,    /* 0110 */
    WLAN_MGMT_SUBTYPE_RESV1     = 7,    /* 0111 */
    WLAN_BEACON                 = 8,    /* 1000 */
    WLAN_ATIM                   = 9,    /* 1001 */
    WLAN_DISASOC                = 10,   /* 1010 */
    WLAN_AUTH                   = 11,   /* 1011 */
    WLAN_DEAUTH                 = 12,   /* 1100 */
    WLAN_ACTION                 = 13,   /* 1101 */
    WLAN_ACTION_NO_ACK          = 14,   /* 1110 */
    WLAN_MGMT_SUBTYPE_RESV2     = 15,   /* 1111 */

    WLAN_MGMT_SUBTYPE_BUTT      = 16    /* 一共16种管理帧子类型 */
} wlan_frame_mgmt_subtype_enum;
```

```
/* 数据帧子类型 */
typedef enum {
    WLAN_DATA                   = 0,
    WLAN_DATA_CF_ACK            = 1,
    WLAN_DATA_CF_POLL           = 2,
    WLAN_DATA_CF_ACK_POLL       = 3,
    WLAN_NULL_FRAME             = 4,
    WLAN_CF_ACK                 = 5,
    WLAN_CF_POLL                = 6,
    WLAN_CF_ACK_POLL            = 7,
    WLAN_QOS_DATA               = 8,
    WLAN_QOS_DATA_CF_ACK        = 9,
    WLAN_QOS_DATA_CF_POLL       = 10,
    WLAN_QOS_DATA_CF_ACK_POLL   = 11,
    WLAN_QOS_NULL_FRAME         = 12,
    WLAN_DATA_SUBTYPE_RESV1     = 13,
    WLAN_QOS_CF_POLL            = 14,
    WLAN_QOS_CF_ACK_POLL        = 15,

    WLAN_DATA_SUBTYPE_MGMT      = 16,
} wlan_frame_data_subtype_enum;
```

ppdu format 过滤具体参数取值范围 0~63

bit[0]:non-HT

bit[1]:HE_(ER)SU

bit[2]:HE_MU_MIMO
bit[3]:HE_MU_OFDMA
bit[4]:HT
bit[5]:VHT

2. CSI 开关

打开 CSI

```
/**
 * @if Eng
 * @brief   Start CSI report.
 * @retval ERRCODE_SUCC          Execute successfully.
 * @retval Other                 Execute failed.
 * @else
 * @brief   开启CSI上报。
 * @retval ERRCODE_SUCC          成功。
 * @retval Other                 失败。
 * @endif
 */
errcode_t wifi_csi_start(void);
```

关闭 CSI

```
/**
 * @if Eng
 * @brief   Close CSI report.
 * @retval ERRCODE_SUCC          Execute successfully.
 * @retval Other                 Execute failed.
 * @else
 * @brief   关闭CSI上报。
 * @retval ERRCODE_SUCC          成功。
 * @retval Other                 失败。
 * @endif
 */
errcode_t wifi_csi_stop(void);
```

3. CSI 回调接口

```
/**
 * @if Eng
 * @brief   Register report callback of CSI.
 * @param   [in]   data_cb Callback pointer.
 * @retval ERRCODE_SUCC          Execute successfully.
 * @retval Other                 Error code.
```

```
 * @else
 * @brief   注册CSI数据上报回调函数。
 * @param   [in]   data_cb 回调函数。
 * @retval ERRCODE_SUCC            成功。
 * @retval Other                   失败。
 * @endif
 */
errcode_t wifi_register_csi_report_cb(wifi_csi_data_cb data_cb);

/**
 * @if Eng
 * @brief   CSI data report callback.
 * @param   [in]   csi_data 4 bytes extend timestamp + 758 bytes 64bit big endian data.
 * @param   [in]   len data length.
 * @retval void
 * @else
 * @brief   用户注册的回调函数，用于处理CSI上报的数据。
 * @param   [in]   csi_data 4字节扩展时间戳+758字节64位小端存储格式的CSI数据.
 * @param   [in]   len 数据长度,固定为762字节。
 * @retval void
 * @endif
 */
typedef void (*wifi_csi_data_cb)(uint8_t *csi_data, int32_t len);
```

5 雷达特性说明

5.1 概述

雷达特性为周期性地收发雷达信号以检测运动目标的功能特性，用户可以调用雷达 APIs 接口使用该特性，详细内容请参见《WS63V100 雷达快速入门指南》。

5.2 开发流程

5.2.1 数据结构

雷达状态设置枚举定义：

```
typedef enum {
    RADAR_STATUS_STOP = 0,   /* 雷达状态配置停止 */
    RADAR_STATUS_START,      /* 雷达状态配置启动 */
    RADAR_STATUS_RESET,      /* 雷达状态配置复位 */
    RADAR_STATUS_RESUME,     /* 雷达状态配置状态恢复 */
} radar_set_sts_t;
```

雷达状态查询枚举定义：

```
typedef enum {
    RADAR_STATUS_IDLE = 0,   /* 雷达状态未工作 */
    RADAR_STATUS_RUNNING,    /* 雷达状态工作 */
} radar_get_sts_t;
```

雷达结果上报结构体定义：

```
typedef struct {
    uint32_t lower_boundary;      /* 雷达结果靠近检测下边界 */
    uint32_t upper_boundary;      /* 雷达结果靠近检测上边界 */
    uint8_t is_human_presence;    /* 雷达结果有无人体存在 */
    uint8_t reserved_0;
    uint8_t reserved_1;
    uint8_t reserved_2;
} radar_result_t;
```

结果回调函数数据结构定义：

typedef void (*radar_result_cb_t)(radar_result_t *result);

5.2.2 APIs

雷达 APIs 接口如下表所示。

接口名称	描述	参数说明	返回信息说明
uapi_radar_set_status	设置雷达状态	*sts*:: 雷达状态	接口返回值：错误码。
uapi_radar_get_status	获取雷达状态	*sts: 雷达状态	接口返回值：错误码。
uapi_radar_register_result_cb	雷达结果回调注册函数	*cb*: 回调函数	接口返回值：错误码。
uapi_radar_set_delay_time	设置退出延迟时间	*time*: 退出延迟时间	接口返回值：错误码。
uapi_radar_get_delay_time	获取退出延迟时间	*time: 退出延迟时间	接口返回值：错误码。

接口名称	描述	参数说明	返回信息说明
uapi_radar_get_isolation	获取天线隔离度信息	*iso: 天线隔离度信息	接口返回值：错误码。

5.2.3 错误码

序号	定义	实际数值	描述
1	ERRCODE_SUCC	0	执行成功错误码。
2	ERRCODE_FAIL	0xFFFFFFFF	执行失败错误码。

5.3 注意事项

雷达特性需要在 WiFi 信道上进行工作，所以需注意打开雷达前，需要确保 WiFi 信道有配置，WiFi 进入 softAP 或 STA 模式即可。

5.4 编程实例

```
typedef void (*radar_result_cb_t)(radar_result_t *result);

#define WIFI_IFNAME_MAX_SIZE                16
#define WIFI_MAX_SSID_LEN                   33
#define WIFI_SCAN_AP_LIMIT                  64
#define WIFI_MAC_LEN                        6
#define WIFI_INIT_WAIT_TIME                 500 // 5s
#define WIFI_START_STA_DELAY                100 // 1s

#define RADAR_STATUS_SET_START              1
#define RADAR_STATUS_QUERY_DELAY            1000 // 10s

// WiFi启动STA模式实现样例
td_s32 radar_start_sta(td_void)
{
```

```
    (void)osDelay(WIFI_INIT_WAIT_TIME); /* 500: 延时0.5s, 等待wifi初始化完毕 */
    PRINT("STA try enable.\r\n");
    /* 创建STA接口 */
    if (wifi_sta_enable() != 0) {
        PRINT("sta enbale fail !\r\n");
        return -1;
    }

    /* 连接成功 */
    PRINT("STA connect success.\r\n");
    return 0;
}

// 雷达结果回调函数实现样例
static void radar_print_res(radar_result_t *res)
{
    PRINT("[RADAR_SAMPLE] lb:%u, hb:%u, hm:%u\r\n", res->lower_boundary, res->upper_boundary, res->is_human_presence);
}

int radar_demo_init(void *param)
{
    PRINT("[RADAR_SAMPLE] radar_demo_init sta!\r\n");

    param = param;
    // WiFi启动STA模式
    radar_start_sta();

    // 注册雷达结果回调函数
    uapi_radar_register_result_cb(radar_print_res);

    // 启动雷达
    (void)osDelay(WIFI_START_STA_DELAY);
    uapi_radar_set_status(RADAR_STATUS_SET_START);

    // 雷达查询接口示例
    while(1) {
        (void)osDelay(RADAR_STATUS_QUERY_DELAY);
        uint8_t sts;
```

```
        uapi_radar_get_status(&sts);
        uint16_t time;
        uapi_radar_get_delay_time(&time);
        uint16_t iso;
        uapi_radar_get_isolation(&iso);
    }

    return 0;
}
```

6 BLE 配网特性说明

6.1 概述

BLE 配网是指通过 BLE 辅助 WIFI 入网。

6.2 应用场景

在 BLE 配网模式下，设备在无网模式下通过 BLE 发送配网广播，附近的手机扫描到该广播，可通过多种方式与用户交互：

1）手机应用主动弹框询问手机用户是否允许该设备加入家庭网络。

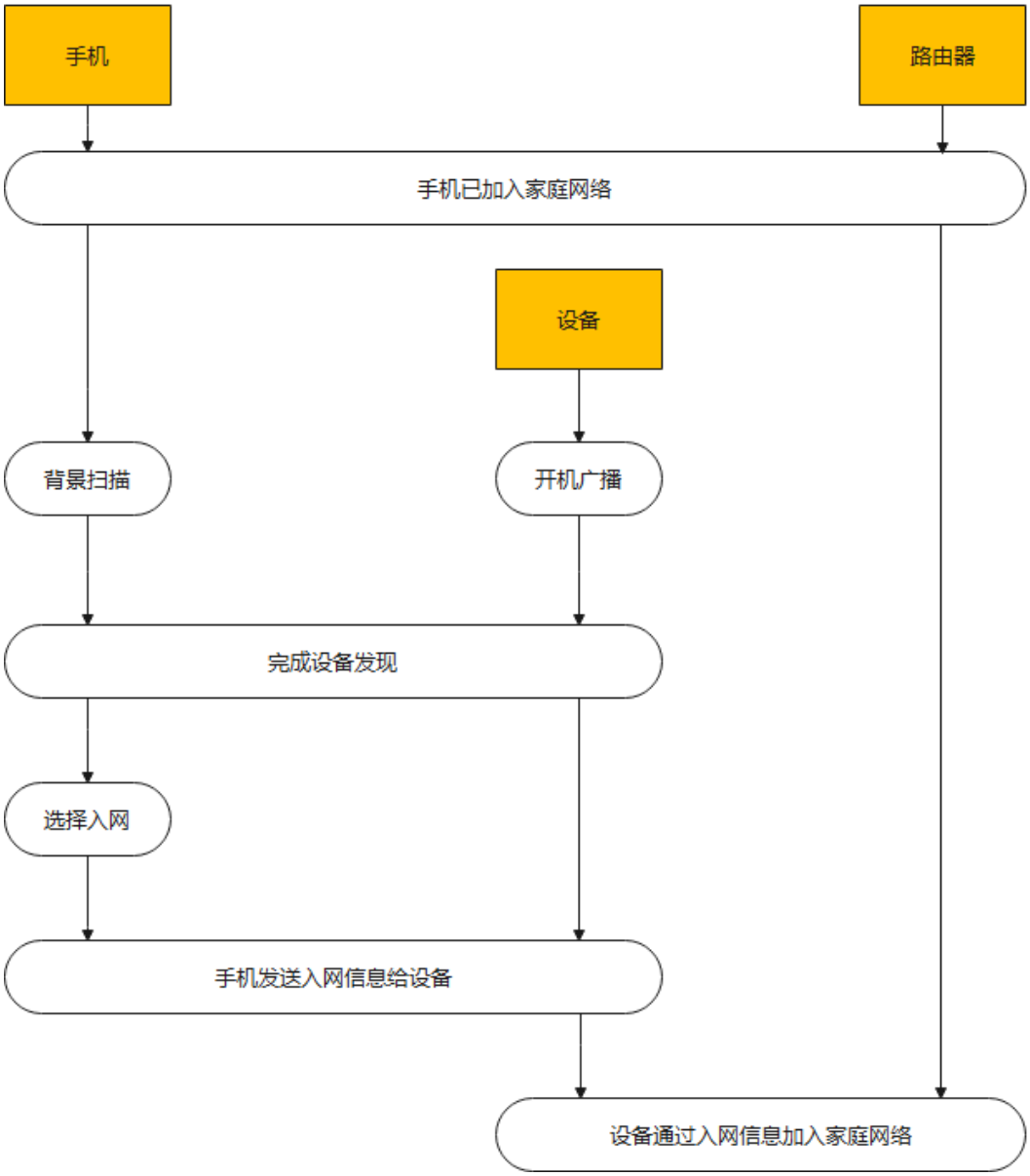
2）手机应用主动扫描后显示在可用设备列表，用户手动选择将该设备加入家庭网络。

如用户选择加入，手机与设备建立 BLE 连接，再将 WIFI 入网信息（SSID、密码等）传输给设备。与 SoftAP 配网相比，用户无需切换 WIFI AP 与 STA 模式，可以大大提升终端用户体验。

6.3 实现原理

BLE 配网参考流程如下：

图6-1 BLE 配网流程



使用 WS63 芯片的设备，在无网模式下启动，自动发送配网广播，用户可以调整广播持续时间、间隔以及重新广播的触发条件。

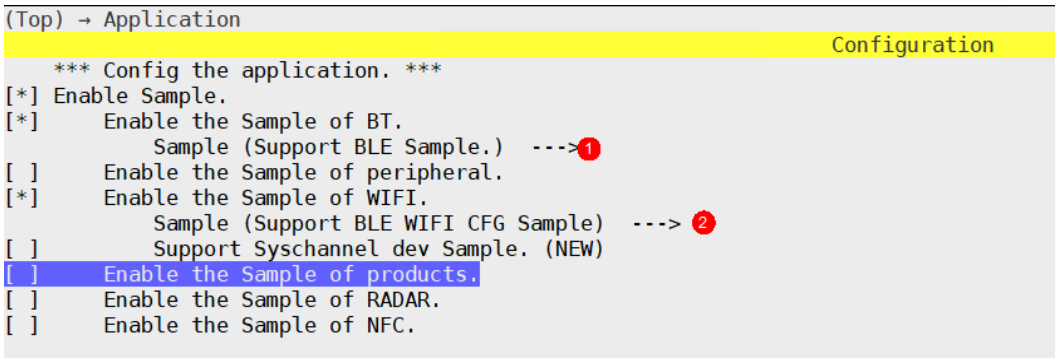
6.4 接口说明

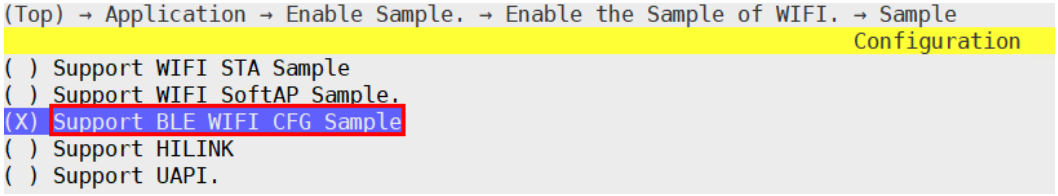
参考《WS63V100 软件开发指南》BLE 开发流程章节。

6.5 使用示例

步骤 1 在 SDK 根目录下执行命令“python3 build.py ws63-liteos-app menuconfig”，并按下图配置对应编译选项进行配置。

图6-2 BLE Demo 配置选项

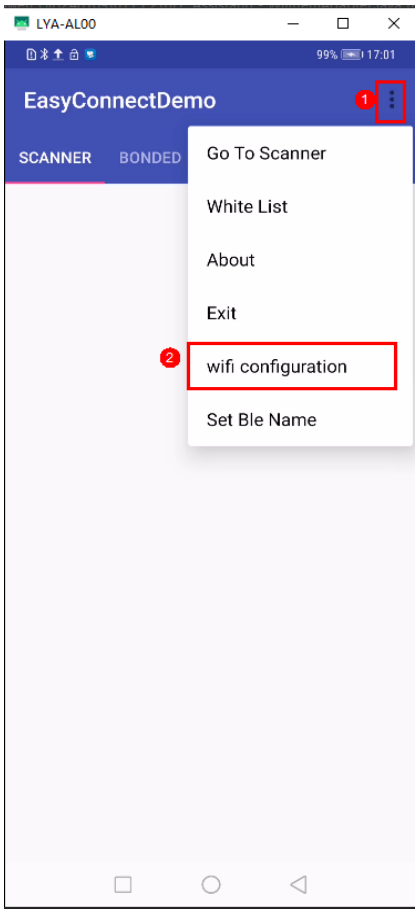




步骤 2 完成配置后执行命令 python3 build.py ws63-liteos-app，将生成的镜像通过 BurnTool 烧录进单板中。

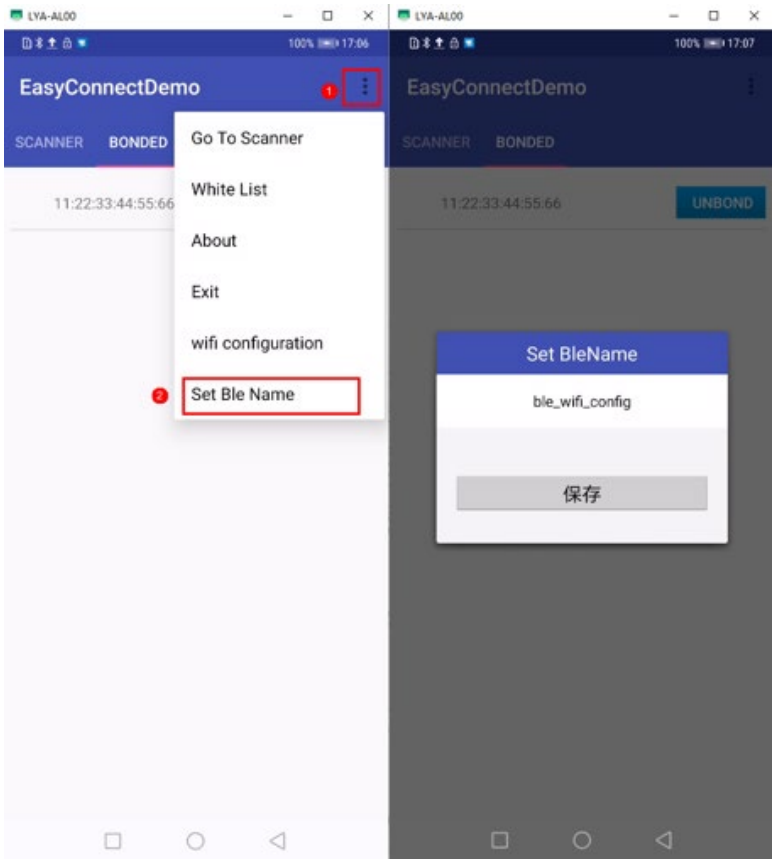
步骤 3 在 Android 手机安装“EasyConnect”软件，点击“wifi configuration”，配置待连接的 Wi-Fi 参数;

图6-3 配置 Wi-Fi 参数



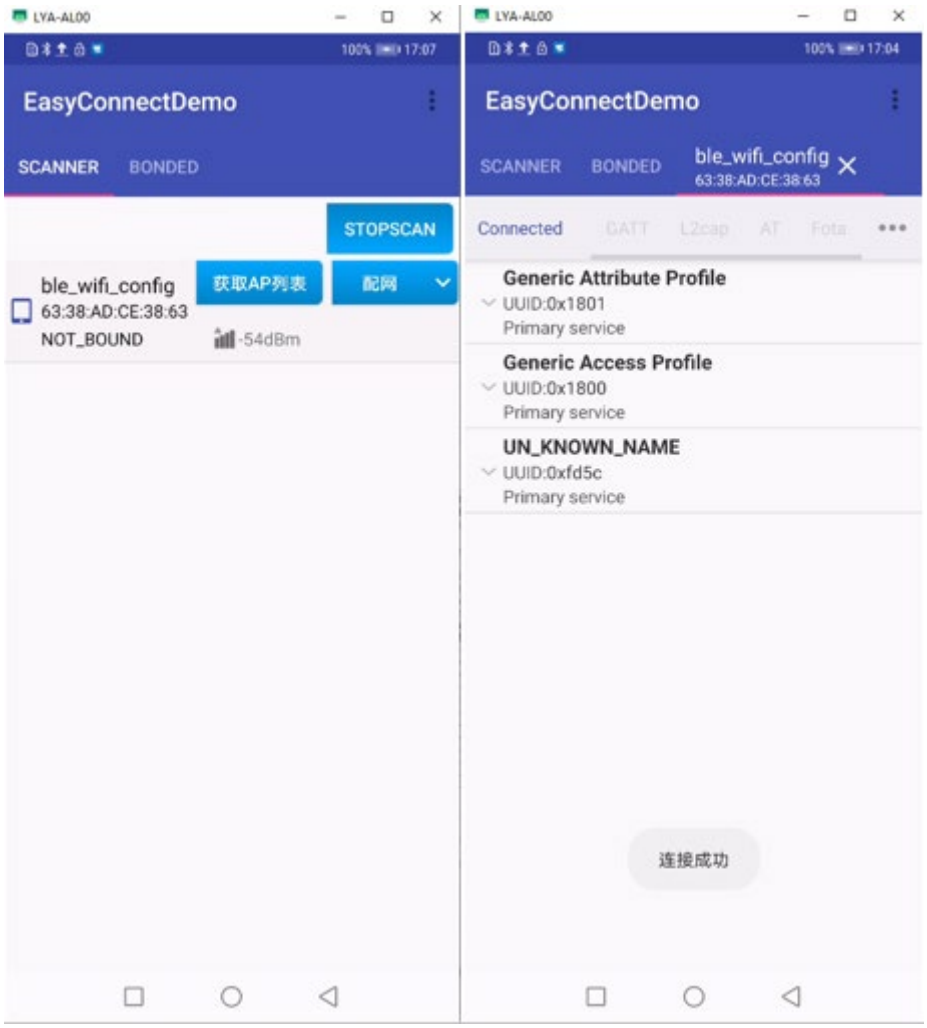
步骤 4 点击“Set Ble Name”，默认配置为 ble_wifi_config；

图6-4 配置蓝牙名称



步骤 5 配置完成后，点击“配网”，待 Android 手机和 63 模组连接成功后，会显示连接成功并播报。

图6-5 WS63 模组配网成功



----结束

说明

配网软件“EasyConnect”可通过技术支持获取。

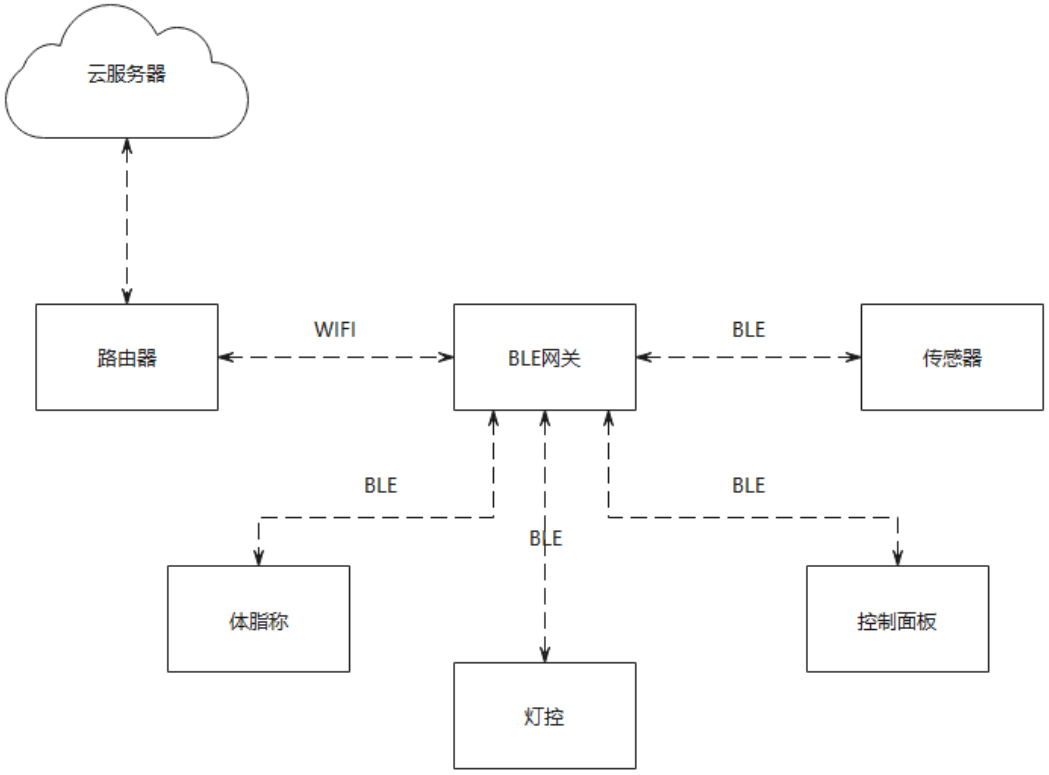
7 BLE 网关特性说明

7.1 概述

BLE 网关做为中继，帮助无法直接与服务器通信的 BLE 设备上云。

7.2 应用场景

BLE 网关实现多个 BLE 子设备的认证、连接管理，以及路由转发功能。从云发送的命令通过网关下发到 BLE 设备；BLE 设备的状态也可以通过网关上报给服务器。



7.3 实现原理

BLE 网关的工作流程参考：

1. 网关开启背景扫描，BLE 子设备发送广播，网关识别支持的服务 ID 主动建立 BLE 连接，完成认证和数据传输；
2. 网关服务程序将收到的 BLE 数据，通过 WIFI 路由器上传到服务器；
3. 服务器将 BLE 设备的信息在手机应用或者控制中枢显示屏显示，控制指令也可通过 WIFI 由服务器传给网关，再由网关通过 BLE 传输给设备。

7.4 接口说明

参考《WS63V100 软件开发指南》BLE 开发流程章节。

8 安全启动特性说明

8.1 概述

安全启动是指在系统启动时，使用 efuse 中预先写入的秘钥对镜像逐级校验的功能。

8.2 应用场景

安全启动用于保证镜像的完整性和安全性，防止镜像被破解和篡改。

须知

安全启动功能在产测阶段通过烧录 efuse 打开，开启该功能后仅能使用 efuse 中对应的秘钥组对镜像签名才能正常启动。

8.3 实现原理

预先在 efuse 中写入根公钥 HASH 值和安全启动使能位；

系统复位后从 bootrom 启动，

步骤 1 在 bootrom 中计算镜像中的根公钥 HASH 值与 efuse 中的进行校验；

步骤 2 根公钥校验通过后，使用根公钥验签 ssb 镜像；

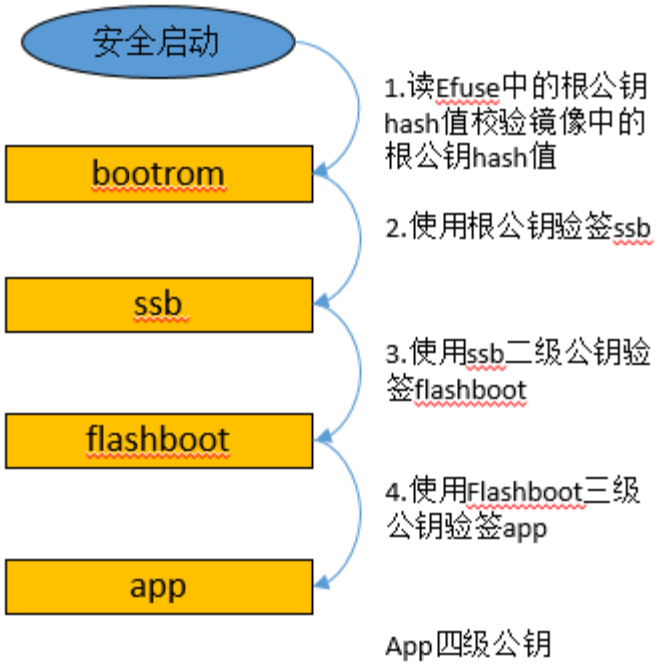
步骤 3 Ssb 镜像校验通过后，使用 ssb 镜像中的二级公钥校验 Flashboot 镜像；

步骤 4 Flashboot 镜像校验通过后，使用 flashboot 中的三级公钥校验 App 镜像；

步骤 5 App 镜像校验通过后，跳转到 app 镜像启动完成。

----结束

图 7.1 安全启动镜像逐级校验流程



8.4 接口说明

1.配置 efuse 安全启动使能位写 1

440088F0[0]	440088F0[0]	NA	NA	60	60	15	0	960	960	PG7	1	sec_verify_enable	secure boot verify enable. 0x0: not enable; 0x1: enable.

2.配置根公钥 HASH 值烧写到 efuse

440088E4[15]	440088A8[0]	NA	NA	57	42	15	0	927	672	PG5	256	Hash root public key	根公钥HASH
440088EC[15]	440088E8[0]	NA	NA	59	58	15	0	959	928	PG6	32	MSID	市场ID
440088F0[0]	440088F0[0]	NA	NA	60	60	15	0	960	960	PG7	1	sec_verify_enable	secure boot verify enable. 0x0: not enable; 0x1: enable.

说明

可参照《WS63V100 二次开发网络安全 注意事项》中安全启动配置章节进行配置

9 FLASH 在线解密特性说明

9.1 概述

FLASH 在线解密是指 CPU 在访问 FLASH 上的加密镜像时在线解密后运行。

9.2 应用场景

FLASH 在线解密特性主要用于对加密存储在 FLASH 上的 APP 镜像进行解密运行。

须知

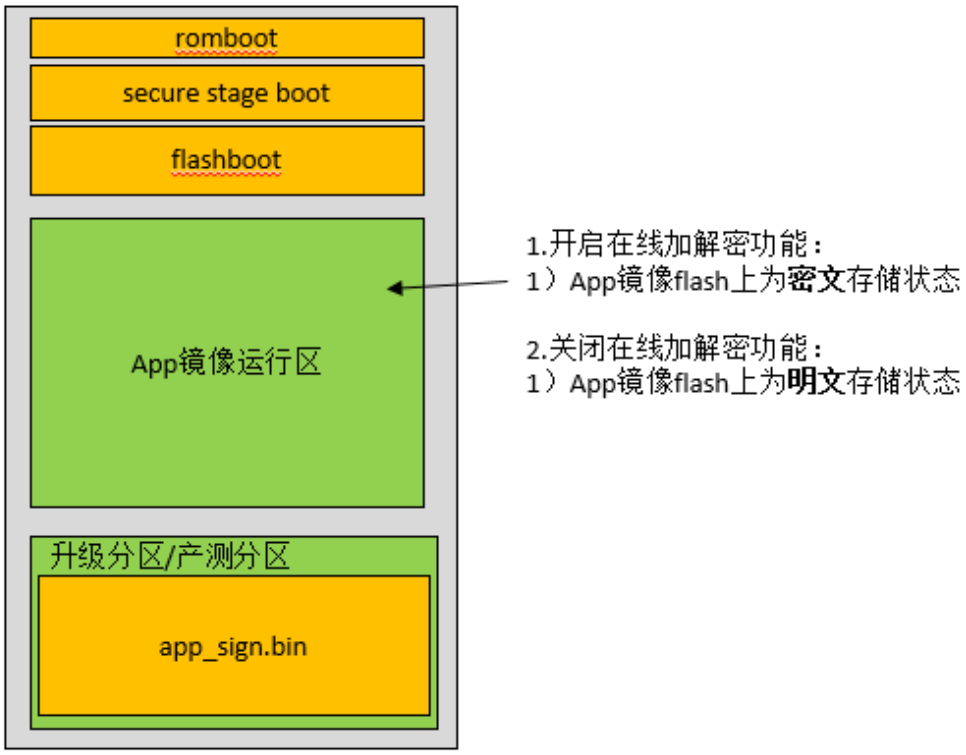
FLASH 在线解密功能需要在编译前打开镜像加密，并将对应的秘钥派生参数写入 efuse 中。

9.3 实现原理

开启 FLASH 在线解密功能后：

1. 在编译阶段签名后会对镜像加密。
2. 镜像烧写到 flash 上为加密存储状态。
3. 启动时会自动配置解密区域。
4. cpu 运行时通过配置由硬件自动解密。

图9-1 FLASH 在线解密示意图



9.4 接口说明

1. 在配置文件 hi3863/boards/HI3863/sdk/build/config/target_config/ws63/sign_config/liteos_app_bin_ecc.cfg 中将 SignSuite 配置为 1，并配置秘钥派生参数对应的 IV 和 PlainKey。
2. 将秘钥派生参数写入 efuse 对应位置

44008850[15]	44008850[9]	175	169					335	329		7	reserved	预留
44008854[15]	44008854[0]	191	176	21	21	15	0	351	336	PG2	16	obfu_mrk1_owner_id（KM）	Specific Identifier of Obfu_MRK1.
44008874[15]	44008858[0]	319	192	29	22	15	0	479	352	PG3	128	otp_oem_mrk（KM）	Obufuscated MRK1, owned by OEM
44008894[15]	44008878[0]	447	320	37	30	15	0	607	480	PG4	128	obfu_rusk（KM）	Obuscated Random Device RootKey.

说明

可参照《WS63V100 二次开发网络安全 注意事项》中镜像加密配置章节进行配置