



**Hi3861V100 / Hi3861LV100 Wi-Fi 软件**

## **开发指南**

文档版本 03

发布日期 2020-07-06

版权所有 © 上海海思技术有限公司2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 商标声明



**HISILICON**、海思和其他海思商标均为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：            深圳市龙岗区坂田华为总部办公楼    邮编：518129

网址：            <https://www.hisilicon.com/cn/>

客户服务邮箱：  [support@hisilicon.com](mailto:support@hisilicon.com)



# 前言

## 概述

本文档详细介绍了Hi3861V100、Hi3861LV100 Wi-Fi软件STA、SoftAp的接口功能以及开发流程。

## 产品版本

与本文档对应的产品版本如下。

产品名称	产品版本
Hi3861	V100
Hi3861L	V100



## 读者对象

本文档主要适用以下工程师：



- 技术支持工程
- 软件开发工程师

## 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
 <b>危险</b>	表示如不可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。
 <b>警告</b>	表示如不可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。



符号	说明
 注意	表示如不可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。如不可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
03	2020-07-06	新增“ <a href="#">9 软件重传功能</a> ”章节。
02	2020-06-22	更新“ <a href="#">6.3 注意事项</a> ”的 <a href="#">表6-3</a> 内容。



文档版本	发布日期	修改说明
01	2020-04-30	<p>第一次正式版本发布。</p> <ul style="list-style-type: none"> <li>在“<a href="#">1 概述</a>”中删除关于HSO维测工具的说明，更新WPA SUPPLICANT的说明；新增关于各个模块API接口的说明。</li> <li>在“<a href="#">2.3 注意事项</a>”中更新关于合理配置初始化的资源数的注意说明。</li> <li>在“<a href="#">2.4 编程实例</a>”中更新示例1、示例2的代码示例。</li> <li>在“<a href="#">3.2 开发流程</a>”的表3-1中新增 hi_wifi_set_bandwidth、hi_wifi_sta_set_reconnect_policy、hi_wifi_config_callback、hi_wifi_register_event_callback、hi_wifi_sta_advance_scan、netifapi_dhcp_stop接口的描述，更新 netifapi_dhcp_start接口的描述；更新<a href="#">开发流程</a>。</li> <li>在“<a href="#">3.3 注意事项</a>”中新增关于STA支持 5M/10M窄带模式、带指定参数的扫描、已知待连接网络的参数、注册事件回调函数的注意说明。</li> <li>在“<a href="#">3.4 编程实例</a>”中更新代码示例。</li> <li>在“<a href="#">4.1 概述</a>”中更新SoftAp功能的描述。</li> <li>在“<a href="#">4.2 开发流程</a>”的表4-1中新增 hi_wifi_set_bandwidth、netifapi_dhcps_stop接口的描述；更新<a href="#">开发流程</a>。</li> <li>在“<a href="#">4.3 注意事项</a>”中新增关于SoftAp支持 5M/10M窄带模式、SoftAp的网络参数在关闭SoftAp时不会重置、SoftAp模式下最大关联用户数限制的注意说明。</li> <li>在“<a href="#">4.4 编程实例</a>”中更新代码示例。</li> <li>在“<a href="#">5.3 注意事项</a>”中新增关于混杂模式会消耗大量CPU和内存资源的注意说明。</li> <li>在“<a href="#">5.4 编程实例</a>”中更新代码示例。</li> <li>在“<a href="#">6.1 概述</a>”中删除关于PHY层的描述。</li> <li>在“<a href="#">6.3 注意事项</a>”中新增关于配置CSI采样和上报周期、CSI上报条件的注意说明。</li> <li>在“<a href="#">6.4 编程实例</a>”中更新代码示例。</li> <li>新增“<a href="#">7 STA&amp;SoftAp共存</a>”章节。</li> <li>新增“<a href="#">8 Wi-Fi&amp;蓝牙共存</a>”章节。</li> <li>将本文中错误码改为返回值。</li> </ul>
00B03	2020-03-25	<ul style="list-style-type: none"> <li>新增“<a href="#">5 混杂模式</a>”章节。</li> <li>新增“<a href="#">6 CSI数据采集</a>”章节。</li> </ul>



文档版本	发布日期	修改说明
00B02	2020-03-19	更新“ <a href="#">4.4 编程实例</a> ”的代码示例。
00B01	2020-01-15	第一次临时版本发布。



## 目录

前言.....	i
1 概述.....	1
2 Wi-Fi 驱动加载与卸载.....	3
2.1 概述.....	3
2.2 开发流程.....	3
2.3 注意事项.....	4
2.4 编程实例.....	4
3 STA 功能.....	7
3.1 概述.....	7
3.2 开发流程.....	7
3.3 注意事项.....	9
3.4 编程实例.....	9
4 SoftAp 功能.....	12
4.1 概述.....	12
4.2 开发流程.....	12
4.3 注意事项.....	14
4.4 编程实例.....	14
5 混杂模式.....	16
5.1 概述.....	16
5.2 开发流程.....	16
5.3 注意事项.....	17
5.4 编程实例.....	17
6 CSI 数据采集.....	19
6.1 概述.....	19
6.2 开发流程.....	19
6.3 注意事项.....	20
6.4 编程实例.....	22
7 STA&SoftAp 共存.....	24
7.1 概述.....	24
7.2 开发流程.....	24



7.3 注意事项.....	25
7.4 编程实例.....	25
<b>8 Wi-Fi&amp;蓝牙共存.....</b>	<b>26</b>
8.1 概述.....	26
8.2 开发流程.....	26
8.3 注意事项.....	27
8.4 编程实例.....	27
<b>9 软件重传功能.....</b>	<b>29</b>
9.1 概述.....	29
9.2 开发流程.....	29
9.3 注意事项.....	30
9.4 编程实例.....	30

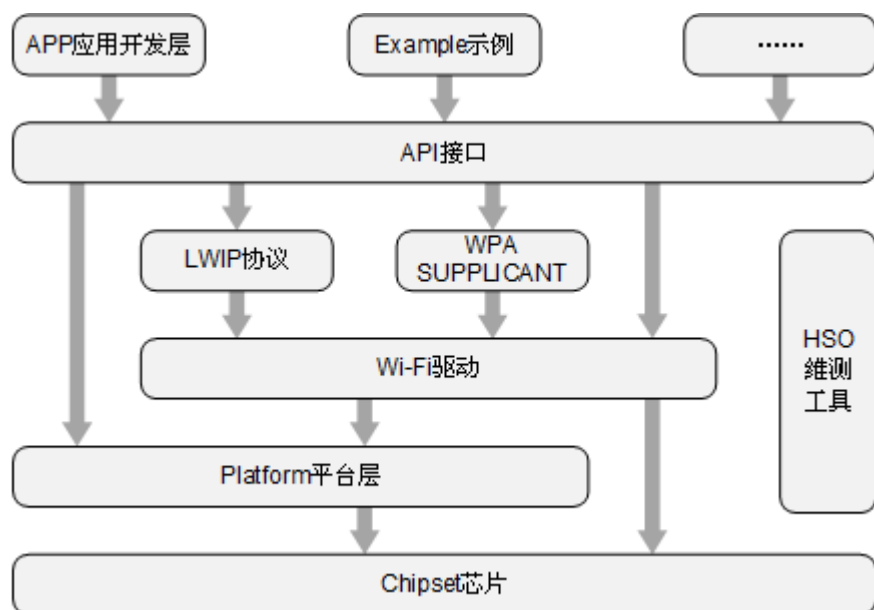




# 1 概述

Hi3861V100、Hi3861LV100 通过API（Application Programming Interface）面向开发者提供Wi-Fi功能的开发和应用接口，包括芯片初始化、资源配置、Station创建和配置、扫描、关联以及去关联、状态查询等一系列功能，框架结构如图1-1所示。

图 1-1 Hi3861/Hi3861L API 接口控制流框图



各功能模块说明如下：

- APP应用开发层：用户基于API接口的二次开发。
- Example示例：SDK提供的功能开发示例。
- API接口：提供基于SDK的通用接口。
- LWIP协议栈：网络协议栈。
- WPA SUPPLICANT（含HOSTAPD）：Wi-Fi管理模块。
- Wi-Fi驱动：802.11协议实现模块。
- Platform平台：提供SoC系统板级支持包（包括：芯片和外围设备驱动、操作系统以及系统管理）。



## 说明

该文档描述各个模块功能的基本流程，API接口的详细说明请参见《Hi3861V100 / Hi3861LV100 API 开发参考》。



# 2 Wi-Fi 驱动加载与卸载

## 2.1 概述

## 2.2 开发流程

## 2.3 注意事项

## 2.4 编程实例

## 2.1 概述

在完成芯片上电后，驱动加载实现对芯片寄存器的初始配置、校准参数读取与写入、软件资源的申请和配置；驱动卸载实现软件资源的释放。

## 2.2 开发流程

### 使用场景

Wi-Fi驱动初始化为Wi-Fi功能提供基本资源配置和芯片初始化，是Wi-Fi功能实现的第一步。当需要配置Wi-Fi功能时，必须先完成驱动的初始化，Wi-Fi功能使用完成后，可以使用去初始化完成资源释放也可以使用软复位来完成资源释放。

### 功能

Wi-Fi驱动加载与卸载提供的接口如表2-1所示。

表 2-1 Wi-Fi 驱动加载与卸载接口描述

接口名称	描述
hi_wifi_init	Wi-Fi驱动初始化。
hi_wifi_deinit	Wi-Fi驱动去初始化。



## 开发流程

使用驱动加载与卸载的典型流程：

- 步骤1 调用hi\_wifi\_init，完成Wi-Fi驱动初始化。
  - 步骤2 参考“[3 STA功能](#)”或“[4 SoftAp功能](#)”配置Wi-Fi功能。
  - 步骤3 调用hi\_wifi\_deinit，完成Wi-Fi驱动去初始化。
- 结束

## 返回值

Wi-Fi驱动加载与卸载的返回值如[表2-2](#)所示。

表 2-2 Wi-Fi 驱动加载与卸载返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	初始化成功。
2	HISI_FAIL	-1	初始化失败。

## 2.3 注意事项

- 驱动资源配置不支持运行中修改，须先卸载驱动再进行修改，修改后重新初始化。
- 驱动为了保证Wi-Fi业务的连续性，会在启动时根据VAP数量和用户数量预申请内存，其中1个VAP资源预申请约5K内存，1个用户资源预申请约7K内存，请根据场景需要合理配置初始化的资源数。目前仅配网时需要用到SoftAp和STA共存，一般建议配置为2个VAP和2个用户；如果可以实现配网时先关闭SoftAp再启动STA去关联，VAP和用户数量均可配置为1。

## 2.4 编程实例

示例1：基于LiteOS的app\_main函数，在系统初始化时自动完成Wi-Fi驱动的加载，此加载方式无须进行卸载开发，系统reboot时自动完成驱动卸载和加载。

### 代码示例

```
#define APP_INIT_VAP_NUM 2
#define APP_INIT_USR_NUM 2

hi_void app_main(hi_void)
{
    hi_u32 ret;
    const hi_uchar wifi_vap_res_num = APP_INIT_VAP_NUM;
    const hi_uchar wifi_user_res_num = APP_INIT_USR_NUM;

    ret = hi_wifi_init(wifi_vap_res_num, wifi_user_res_num);
    if (ret != 0) {
        printf("fail to init wifi\n");
    } else {
```



```
    printf("wifi init success\n");  
}  
}
```

### 结果验证

```
#reboot  
ready to OS start  
system init status:0x0  
FileSystem mount ok.  
wifi init success  
#
```

示例2：基于shell命令，在系统启动后，通过手动下发shell命令完成Wi-Fi驱动的加载和卸载

### 代码示例

```
#define APP_INIT_VAP_NUM  2  
#define APP_INIT_USR_NUM  2  
  
void cmd_wifi_init(int argc, const char* argv[])  
{  
    hi_u32 ret;  
    const hi_uchar wifi_vap_res_num = APP_INIT_VAP_NUM;  
    const hi_uchar wifi_user_res_num = APP_INIT_USR_NUM;  
  
    ret = hi_wifi_init(wifi_vap_res_num, wifi_user_res_num);  
    if (ret != 0) {  
        printf("fail to init wifi\n");  
    } else {  
        printf("wifi init success\n");  
    }  
}  
  
void cmd_wifi_deinit(int argc, const char* argv[])  
{  
    int ret;  
  
    ret = hi_wifi_deinit();  
    if (ret != 0) {  
        printf("fail to deinit wifi\n");  
    } else {  
        printf("deinit wifi success\n");  
    }  
}  
  
void hisi_wifi_shell_cmd_register(void)  
{  
    OsCmdReg(CMD_TYPE_EX, "wifi_init", XARGS, (CmdCallBackFunc)cmd_wifi_init);  
    OsCmdReg(CMD_TYPE_EX, "wifi_deinit", XARGS, (CmdCallBackFunc)cmd_wifi_deinit);  
}  
  
hi_void app_main(hi_void)  
{  
    hisi_wifi_shell_cmd_register();  
}
```

### 结果验证

```
#wifi_init  
wifi init success
```



```
#wifi_deinit  
deinit wifi success  
#
```



# 3 STA 功能

[3.1 概述](#)

[3.2 开发流程](#)

[3.3 注意事项](#)

[3.4 编程实例](#)

## 3.1 概述

STA功能即NON-AP Station功能，实现驱动STA VAP的创建、扫描、关联以及DHCP，完成通信链路的建立。开发STA功能前，须完成驱动加载。

## 3.2 开发流程

### 使用场景

当需要接入某个网络并与该网络通信时，需要启动STA功能。

### 功能

驱动STA功能提供的接口如[表3-1](#)所示。

表 3-1 驱动 STA 功能接口描述

接口名称	描述
hi_wifi_sta_start	启动STA接口。
hi_wifi_set_bandwidth	设置STA的带宽。
hi_wifi_sta_set_reconnect_policy	设置STA接口自动重连配置。
hi_wifi_config_callback	配置事件回调函数的调用方式。



接口名称	描述
hi_wifi_register_event_callback	注册STA接口的事件回调函数。
hi_wifi_sta_scan	触发STA扫描。
hi_wifi_sta_advance_scan	执行带特定参数的扫描。
hi_wifi_sta_scan_results	获取STA扫描结果。
hi_wifi_sta_connect	触发STA连接Wi-Fi网络。
hi_wifi_sta_get_connect_info	获取STA连接的网络状态。
netifapi_dhcp_start	启动DHCP客户端，获取IP地址。
netifapi_dhcp_stop	停止DHCP客户端。
hi_wifi_sta_disconnect	触发STA离开当前网络。
hi_wifi_sta_stop	关闭STA接口。

开发流程

STA功能开发的典型流程：

- 步骤1 调用hi\_wifi\_sta\_start，启动STA。
  - 步骤2 调用hi\_wifi\_set\_bandwidth，设置STA带宽模式，20M带宽可不用配置。
  - 步骤3 （可选，根据需要配置）调用hi\_wifi\_sta\_set\_reconnect\_policy，设置自动重连。
  - 步骤4 调用hi\_wifi\_sta\_scan（或调用hi\_wifi\_sta\_advance\_scan，执行带参数扫描），触发STA扫描。
  - 步骤5 调用hi\_wifi\_sta\_scan\_results，获取扫描结果。
  - 步骤6 根据接入网络需求，自定义筛选扫描结果，调用hi\_wifi\_sta\_connect，进行连接。
  - 步骤7 调用hi\_wifi\_sta\_get\_connect\_info，查询Wi-Fi连接状态。
  - 步骤8 连接成功后，调用netifapi\_dhcp\_start，启动DHCP客户端，获取IP地址。
  - 步骤9 调用hi\_wifi\_sta\_disconnect，离开当前连接的网络。
  - 步骤10 调用netifapi\_dhcp\_stop，停止DHCP客户端。
  - 步骤11 调用hi\_wifi\_sta\_stop，关闭STA。
- 结束

返回值

STA功能的返回值如表3-2所示。





表 3-2 STA 功能返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	执行成功。
2	HISI_FAIL	-1	执行失败。

## 3.3 注意事项

- STA支持5M/10M窄带模式，需要时可调用接口设置，不调用则默认启动20M带宽STA。
- 扫描为非阻塞式接口，扫描命令下发成功后需要延迟一段时间后再获取扫描结果，全信道扫描延迟时间建议设置为1s。
- 可通过指定SSID、BSSID、信道等带指定参数的扫描，实现更精准地扫描，缩短扫描时间。
- 已知待连接网络的参数时，可省去扫描过程，直接发起连接。
- 连接为非阻塞式接口，连接命令下发成功后，需要通过命令获取连接状态。
- 注册事件回调函数后，Wi-Fi相关的事件会通过该回调上报用户，用户可根据事件执行后续动作。
- 不支持重复启动STA，再次启动STA时须先执行关闭STA。
- 关闭STA步骤为可选，设备所处的网络地位不变，不需要执行关闭STA。

## 3.4 编程实例

示例：实现STA功能启动、扫描、关联以及获取IP地址。

### 代码示例

```
#include "lwip/netifapi.h"
#include "hi_wifi_api.h"
#include "hi_types.h"
#include "hi_timer.h"

#define WIFI_IFNAME_MAX_SIZE      16
#define WIFI_SCAN_AP_LIMIT      64

hi_s32 example_get_match_network(hi_wifi_assoc_request *expected_bss)
{
    hi_s32 ret;
    hi_u32 num = 0; /* 扫描到的Wi-Fi网络数量 */
    hi_char expected_ssid[] = "my_wifi";
    hi_char key[] = "my_password"; /* 待连接的网络接入密码 */
    hi_bool find_ap = HI_FALSE;
    hi_u8 bss_index;
    /* 获取扫描结果 */
    hi_u32 scan_len = sizeof(hi_wifi_ap_info) * WIFI_SCAN_AP_LIMIT;
    hi_wifi_ap_info *pst_results = malloc(scan_len);
    if (pst_results == HI_NULL) {
        return HISI_FAIL;
    }
    memset_s(pst_results, scan_len, 0, scan_len);
```



```
ret = hi_wifi_sta_scan_results(pst_results, &num);
if (ret != HISI_OK) {
    free(pst_results);
    return HISI_FAIL;
}
/* 筛选扫描到的Wi-Fi网络, 选择待连接的网络 */
for (bss_index = 0; bss_index < num; bss_index++) {
    if (strlen(expected_ssid) == strlen(pst_results[bss_index].ssid)) {
        if (memcmp(expected_ssid, pst_results[bss_index].ssid, strlen(expected_ssid)) ==
HISI_OK) {
            find_ap = HI_TRUE;
            break;
        }
    }
}
/* 未找到待连接AP,可以继续尝试扫描或者退出 */
if (find_ap == HI_FALSE) {
    free(pst_results);
    return HISI_FAIL;
}
/* 找到网络后复制网络信息和接入密码 */
if (memcpy_s(expected_bss->ssid, (HI_WIFI_MAX_SSID_LEN+1), expected_ssid,
strlen(expected_ssid)) != HISI_OK) {
    free(pst_results);
    return HISI_FAIL;
}
expected_bss->auth = pst_results[bss_index].auth;
/* hidden_ssid 固定写成1 */
expected_bss->hidden_ssid = 1;
/* pairwise mode 默认设置为0 */
expected_bss->pairwise = 0;
if (memcpy_s(expected_bss->key, (HI_WIFI_MAX_KEY_LEN+1), key, strlen(key)) != HISI_OK) {
    free(pst_results);
    return HISI_FAIL;
}
free(pst_results);
return HISI_OK;
}

hi_bool example_check_connect_status(hi_void)
{
    hi_u8 index;
    hi_wifi_status wifi_status;
    /* 获取网络连接状态, 共查询5次, 每次间隔500ms */
    for (index = 0; index < 5; index++) {
        hi_udelay(500000); /* 延迟500ms */
        memset_s(&wifi_status, sizeof(hi_wifi_status), 0, sizeof(hi_wifi_status));
        if (hi_wifi_sta_get_connect_info(&wifi_status) != HISI_OK) {
            continue;
        }
        if (wifi_status.status == HI_WIFI_CONNECTED) {
            return HI_TRUE; /* 连接成功退出循环 */
        }
    }
    return HI_FALSE;
}

hi_s32 example_sta_function(hi_void)
{
    hi_char ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0}; /* 创建的STA接口名 */
    hi_s32 len = WIFI_IFNAME_MAX_SIZE + 1; /* STA接口名长度, +1为了存放字符串结束符 */
    hi_wifi_assoc_request expected_bss = {0}; /* 连接请求信息 */
}
```



```
struct netif *netif_p = HI_NULL;

/* 创建STA接口 */
if (hi_wifi_sta_start(ifname, &len) != HISI_OK) {
    return HISI_FAIL;
}
/* 启动STA扫描 */
if (hi_wifi_sta_scan() != HISI_OK) {
    return HISI_FAIL;
}
hi_udelay(1000000); /* 延迟1s等待扫描完成 */
/* 获取待连接的网络 */
if (example_get_match_network(&expected_bss) != HISI_OK) {
    return HISI_FAIL;
}
/* 启动连接 */
if (hi_wifi_sta_connect(&expected_bss) != HISI_OK) {
    return HISI_FAIL;
}
/* 检查网络是否连接成功 */
if (example_check_connect_status() == HI_FALSE) {
    return HISI_FAIL;
}
/* DHCP获取IP地址 */
netif_p = netif_find(ifname);
if (netif_p == HI_NULL) {
    return HISI_FAIL;
}
if (netifapi_dhcp_start(netif_p) != HISI_OK) {
    return HISI_FAIL;
}
/* 连接成功 */
printf("STA connect success.\n");
return HISI_OK;
}
```

### 结果验证

```
#STA connect success.
#
```



# 4 SoftAp 功能

[4.1 概述](#)

[4.2 开发流程](#)

[4.3 注意事项](#)

[4.4 编程实例](#)

## 4.1 概述

SoftAp功能提供网络接入点供其他STA接入，并对接入的STA提供DHCP Server服务。

## 4.2 开发流程

### 使用场景

当需要创建一个网络接入点，供其他设备接入并共享网络内的数据时，需要使用SoftAp功能。

### 功能

驱动SoftAp功能提供的接口如[表4-1](#)所示。

表 4-1 驱动 SoftAp 功能接口描述

接口名称	描述
hi_wifi_softap_start	启动SoftAp接口。
hi_wifi_softap_set_protocol_mode	设置SoftAp协议模式。
hi_wifi_softap_get_protocol_mode	获取SoftAp协议模式。



接口名称	描述
hi_wifi_softap_set_beacon_period	设置SoftAp的beacon周期。
hi_wifi_softap_set_dtim_period	设置SoftAp的dtim周期。
hi_wifi_set_bandwidth	设置SoftAp的带宽模式。
netifapi_netif_set_addr	设置SoftAp的DHCP 服务器的IP地址、子网掩码和网关参数。
netifapi_dhcps_start	启动SoftAp的DHCP服务器。
netifapi_dhcps_stop	停止SoftAp的DHCP服务器。
hi_wifi_softap_get_connected_sta	获取当前接入的STA信息。
hi_wifi_softap_deauth_sta	断开指定STA的连接。
hi_wifi_softap_stop	关闭SoftAp接口。

## 开发流程

SoftAp功能开发的典型流程：

**步骤1** 配置SoftAp的网络参数：

- 调用hi\_wifi\_softap\_set\_protocol\_mode，设置协议模式。
- 调用hi\_wifi\_softap\_set\_beacon\_period，设置beacon周期。
- 调用hi\_wifi\_softap\_set\_dtim\_period，设置dtim周期。

**步骤2** 调用hi\_wifi\_softap\_start，启动SoftAp。

**步骤3** 调用hi\_wifi\_set\_bandwidth，设置SoftAp的带宽，20M带宽可不用配置。

**步骤4** 调用netifapi\_netif\_set\_addr，配置DHCP服务器。

**步骤5** 调用netifapi\_dhcps\_start，启动DHCP服务器。

**步骤6** 调用netifapi\_dhcps\_stop，停止DHCP服务器。

**步骤7** 调用hi\_wifi\_softap\_stop，关闭SoftAp。

----结束

## 返回值

SoftAp功能的返回值如[表4-2](#)所示。



表 4-2 SoftAp 功能返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	执行成功。
2	HISI_FAIL	-1	执行失败。

## 4.3 注意事项

- SoftAp的网络参数为可选配置，无特殊要求均可使用初始默认值。
- SoftAp支持5M/10M窄带模式，需要时可调用接口设置，不调用则默认启动20M带宽SoftAp。
- SoftAp的网络参数在关闭SoftAp时不会重置，会继续沿用上一次配置，重启单板可恢复至初始默认值。
- SoftAp模式下最大关联用户数限制：
  - 小于初始化时配置的用户数量。
  - 最大关联用户不超过2个。

## 4.4 编程实例

示例：实现将SoftAp功能的beacon周期配置为200ms，并启动SoftAp，最后将DHCP服务器的IP地址配置为192.168.43.1。

### 代码示例

```
#include "lwip/netifapi.h"
#include "hi_wifi_api.h"
#include "hi_types.h"

#define WIFI_IFNAME_MAX_SIZE      16

hi_s32 example_softap_function(hi_void)
{
    /* SoftAp接口的信息 */
    hi_wifi_softap_config hapd_conf = {
        "my_wifi", "my_passwork", 1, 0, HI_WIFI_SECURITY_WPA2PSK,
        HI_WIFI_PARIWISE_UNKNOWN};
    hi_char ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0}; /* 创建的SoftAp接口名 */
    hi_s32 len = WIFI_IFNAME_MAX_SIZE + 1; /* SoftAp接口名长度 +1为了存放字符串结束符 */
    struct netif *netif_p = HI_NULL;
    ip4_addr_t st_gw;
    ip4_addr_t st_ipaddr;
    ip4_addr_t st_netmask;
    IP4_ADDR(&st_gw, 192, 168, 43, 1);
    IP4_ADDR(&st_ipaddr, 192, 168, 43, 1);
    IP4_ADDR(&st_netmask, 255, 255, 255, 0);

    /* 配置SoftAp网络参数， beacon周期修改为200ms */
    if (hi_wifi_softap_set_beacon_period(200) != HISI_OK) {
        return HISI_FAIL;
    }

    /* 启动SoftAp接口 */
    if (hi_wifi_softap_start(&hapd_conf, ifname, &len) != HISI_OK) {
```



```
        return HISI_FAIL;
    }
    /* 配置DHCP服务器 */
    netif_p = netif_find(ifname);
    if (netif_p == HI_NULL) {
        (hi_void)hi_wifi_softap_stop();
        return HISI_FAIL;
    }
    if (netifapi_netif_set_addr(netif_p, &st_ipaddr, &st_netmask, &st_gw) != HISI_OK) {
        (hi_void)hi_wifi_softap_stop();
        return HISI_FAIL;
    }
    if (netifapi_dhcps_start(netif_p, NULL, 0) != HISI_OK) {
        (hi_void)hi_wifi_softap_stop();
        return HISI_FAIL;
    }
    printf("SoftAp start success.\n");
    return HISI_OK;
}
```

### 结果验证

```
#SoftAp start success.
#
```



# 5 混杂模式

## 5.1 概述

## 5.2 开发流程

## 5.3 注意事项

## 5.4 编程实例

## 5.1 概述

混杂模式下，Wi-Fi将收到的符合要求的报文通过设置的回调接口上报给用户，实现应用层收取空口数据，以达到发现对端和配网功能。

## 5.2 开发流程

### 使用场景

需要收取相应的空口报文时，将对应的网络接口设置为混杂模式，并配置需要收取的报文类型。

### 功能

混杂功能提供的接口如表5-1所示。

表 5-1 混杂模式功能接口描述

接口名称	描述
hi_wifi_promis_set_rx_callback	混杂模式数据上报回调接口注册。
hi_wifi_promis_enable	混杂模式使能接口。





## 开发流程

混杂模式功能开发的典型流程：

- 步骤1** 创建网络接口（详细内容请参见“[3 STA功能](#)”或“[4 SoftAp功能](#)”，关联或非关联状态均可）。
- 步骤2** 开发实现混杂模式接收回调接口，并调用hi\_wifi\_promis\_set\_rx\_callback，注册回调接口。
- 步骤3** 调用hi\_wifi\_promis\_enable配置帧过滤参数并配置enable参数为1，开启混杂模式。
- 步骤4** 使用完成后，调用hi\_wifi\_promis\_enable配置enable参数为0，关闭混杂模式。

----结束

## 返回值

混杂模式功能的返回值如[表5-2](#)所示。

表 5-2 混杂模式功能返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	执行成功。
2	HISI_FAIL	-1	执行失败。

## 5.3 注意事项

- 混杂模式需先创建网络接口，STA接口无论在关联或非关联状态均可使用混杂模式。
- 混杂模式会消耗大量的CPU和内存资源，特别是在网络较多的环境下，对当前已连接的网络通信影响很大，请根据需要开启。

## 5.4 编程实例

示例：实现开启混杂模式并在串口将收到的报文信息打印出来。

### 代码示例

```
#include "hi_wifi_api.h"
#include "hi_types.h"

#define WIFI_IFNAME_MAX_SIZE      16
int g_recv_cnt = 0;
char ifname[WIFI_IFNAME_MAX_SIZE + 1] = ""; /* +1为了存放字符串结束符 */
*****
功能描述：混杂模式下收包上报并打印
*****
int hi_promis_recv(void* recv_buf, int frame_len, signed char rssi)
{
    unsigned int ret;
    hi_wifi_ptype_filter filter = {0}; /* 关闭混杂模式，过滤值无意义置0处理 */
```



```
g_rcv_cnt ++;
printf("resv buf: %u , len: %d , rssi: %c, cnt:%d.\r\n", (unsigned int)rcv_buf, frame_len, rssi,
g_rcv_cnt);
/* 接收1000个报文后关闭混杂模式 */
if (g_rcv_cnt == 1000) {
    ret = hi_wifi_promis_enable(ifname, 0, &filter);
    if (ret != HI_ERR_SUCCESS) {
        printf("hi_wifi_promis_enable:: set error!\r\n");
        return ret;
    }
    printf("stop promis SUCCESS!\r\n");
}
return HI_ERR_SUCCESS;
}
```

\*\*\*\*\*

功能描述：创建STA并开启混杂模式，接收所有的管理和数据帧，再收到1000个报文后，关闭混杂模式

\*\*\*\*\*/

```
int main(hi_void)
{
    int len = WIFI_IFNAME_MAX_SIZE + 1;
    unsigned int ret;
    hi_wifi_ptype_filter filter = {0};

    /* 创建STA接口 */
    if (hi_wifi_sta_start(ifname, &len) != HISI_OK) {
        return HISI_FAIL;
    }
    /* 开启混杂模式，上报单播/组播管理帧和数据帧 */
    filter.mdata_en = 1;
    filter.udata_en = 1;
    filter.mmngt_en = 1;
    filter.umngt_en = 1;
    hi_wifi_promis_set_rx_callback(hi_promis_rcv);
    ret = hi_wifi_promis_enable(ifname, 1, &filter);
    if (ret != HI_ERR_SUCCESS) {
        printf("hi_wifi_promis_enable:: set error!\r\n");
        return ret;
    }
    printf("start promis SUCCESS!\r\n");
    return HI_ERR_SUCCESS;
}
```



# 6 CSI 数据采集

- 6.1 概述
- 6.2 开发流程
- 6.3 注意事项
- 6.4 编程实例

## 6.1 概述

CSI ( Channel State Information ) 是衡量信道情况的信道状态信息，属于PHY层中OFDM系统下解码的子载波。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。

## 6.2 开发流程

### 使用场景

应用层需要提取CSI数据用于相关应用时，开启CSI功能。

### 功能

CSI功能提供的接口如表6-1所示。

表 6-1 CSI 功能接口描述

接口名称	描述
hi_wifi_csi_set_config	配置CSI数据上报功能的基本参数。
hi_wifi_csi_register_data_recv_func	注册CSI数据上报回调函数。
hi_wifi_csi_start	开启CSI数据上报功能。



接口名称	描述
hi_wifi_csi_stop	关闭CSI数据上报功能。

## 开发流程

CSI功能开发的典型流程：

- 步骤1 创建对应的网络接口（详细内容请参见“[3 STA功能](#)”或“[4 SoftAp功能](#)”）。
- 步骤2 实现CSI数据处理函数，并调用hi\_wifi\_csi\_register\_data\_rcv\_func，注册回调函数。
- 步骤3 调用hi\_wifi\_csi\_set\_config，配置CSI数据上报参数。
- 步骤4 调用hi\_wifi\_csi\_start，开启CSI数据上报。
- 步骤5 调用hi\_wifi\_csi\_stop，关闭CSI数据上报。

----结束

## 返回值

CSI功能的返回值如[表6-2](#)所示。

表 6-2 CSI 功能返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	执行成功。
2	HISI_FAIL	-1	执行失败。

## 6.3 注意事项

- CSI功能由于需要校验发送端MAC地址，因此需要在关联状态下使用。
- 如果当前网络接口已经开启了低功耗，调用hi\_wifi\_csi\_set\_config接口配置CSI后会关闭低功耗，暂时不启用CSI可通过调用一次hi\_wifi\_csi\_stop接口重新开启低功耗。
- 配置CSI采样和上报周期越小，对系统性能影响越大，由于软件上报周期最小值为50ms，建议配置10以上的采样周期，上报周期请根据需要配置。
- CSI上报条件：
  - 物理层完成采样，成功上报CSI数据。
  - 软件两次上报间隔大于配置的上报周期。
- CSI数据长度固定为188byte，储存格式为：4byte扩展时间戳+184byte 64bit小端存储数据，详细的数据解析规则如[表6-3](#)所示。



表 6-3 CSI 数据解析说明

顺序 DWORD序号	位域	名称	说明
DWORD 0	bit[31:0]	csi_timestamp[63:32]	CSI采集时间戳，在csi_start位置锁存的timestamp信号高32bit。
DWORD 1	bit[31:0]	csi_timestamp[31:0]	CSI采集时间戳，在csi_start位置锁存的timestamp信号低32bit。
DWORD 2	bit[31:30]	reserved	预留。
	bit[29:25]	rx_vap_index	当前帧由哪个VAP接收标记。 0: AP0; 4: STA0; 5: STA1。
	bit[24:16]	天线agc code	天线AGC code。
	bit[15:10]	csi_nss_mcs_rate	采集的CSI对应帧的MCS速率。
	bit[9:6]	csi_frame_type	采集的CSI对应帧的子类型（详细内容请参见802.11子帧类型定义）。
	bit[5:4]	csi_frame_sub_type	采集的CSI对应帧的类型。 00: 管理帧; 01: 控制帧; 10: 数据帧。
	bit[3:0]	csi_protocol_mode	采集的CSI对应帧的协议（高2bit为0）。
DWORD 3	bit[31:0]	csi_ta	采集的CSI对应帧的发送端MAC地址低32bit。
DWORD 4	bit[29:17]	csi_phase_incr_ant	天线频偏。
	bit[16:8]	csi_snr_ant	天线SNR。
	bit[7:0]	csi_rssi_ant	天线RSSI。
DWORD 5	bit[31:0]	csi_ra	采集的CSI对应帧的接收端MAC地址低32bit。
DWORD 6	bit[31:16]	csi_ta[47:32]	采集的CSI对应帧的发送端MAC地址高16bit。
	bit[15:0]	csi_ra[47:32]	采集的CSI对应帧的接收端MAC地址高16bit。



顺序 DWORD序号	位域	名称	说明
DWORD N(N=0 ~ 12)	bit[31:0]	csi_h_iq	[23:0]: 第4×N+0个H矩阵信息 bit[23:0]。 [31:24]: 第4×N+1个H矩阵信息 bit[7:0]。
DWORD N(N=0 ~ 12)	bit[31:0]	csi_h_iq	[15:0]: 第4×N+1个H矩阵信息 bit[23:8]。 [31:16]: 第4×N+2个H矩阵信息 bit[15:0]。
DWORD N(N=0 ~ 12)	bit[31:0]	csi_h_iq	[7:0]: 第4×N+2个H矩阵信息 bit[23:16]。 [31:8]: 第4×N+3个H矩阵信息 bit[23:0]。

## 6.4 编程实例

示例：实现开启CSI数据上报并在串口将收到的报文信息打印出来。

### 代码示例

```

/*****
功能描述：打印上报的CSI数据
*****/
void at_hi_wifi_csi_cb(unsigned char *csi_data, int len)
{
    printf("=====csi data=====start=====\\n");
    for (int i=0; i<len; i++) {
        if(i%23 == 0 && i != 0){
            printf("\\n");
        }
        printf("%02x ", csi_data[i]);
    }
    printf("\\n=====csi data=====end=====\\n");
}

/*****
功能描述：STA关联后开启CSI数据上报功能
*****/
int main(hi_void)
{
    hi_wifi_csi_register_data_rcv_func(at_hi_wifi_csi_cb);
    char ifname[] = "wlan0"; /* STA */
    unsigned int interval = 100; /* report interval:100ms */
    hi_wifi_csi_entry csi_entry = {{00,00,00,00,00,00}, 7, 12}; /* mac need change to ap's mac */
    if(hi_wifi_csi_set_config(ifname, interval, &csi_entry, 1) != 0) {

        printf("set csi config failed.\\r\\n");
        return -1;
    }
    hi_wifi_csi_start();
}

```



```
    return 0;  
}
```



# 7 STA&SoftAp 共存

## 7.1 概述

## 7.2 开发流程

## 7.3 注意事项

## 7.4 编程实例

## 7.1 概述

STA&SoftAp共存即STA功能和SoftAp功能同时工作，根据工作的信道和带宽配置分为：

- 同频同带宽共存
- 同频异带宽共存
- 异频同带宽共存
- 异频异带宽共存

同频同带宽共存下，STA&SoftAp根据业务的先后使用device资源，STA和SoftAp全时在线，属于全时共存模式，其余三种共存模式均需要启动共存算法，通过算法调度时隙分时使用STA和SoftAp功能，两种功能在各自的时隙内工作，属于分时共存模式。目前，分时共存模式仅支持平均分时分存，即STA和SoftAp工作时隙相等，两者轮流工作。

## 7.2 开发流程

### 使用场景

配网时，产品先启动SoftAp，手机关联SoftAp后发送家居网络的SSID和密码给产品，产品获取到家居网络的连接参数后启动STA关联家居网络，完成产品联网，产品联网成功后，关闭SoftAp，只保留STA作为端侧长期保持连接。其他共存场景可视产品形态和需求自行使用。

### 功能

共存功能分别使用STA功能和SoftAp功能的API接口，无额外新增API接口。





## 开发流程

配网模式下共存功能开发的典型流程：

- 步骤1** 创建SoftAp网络接口（详细内容请参见“[4 SoftAp功能](#)”）。
- 步骤2** 手机关联SoftAp，并通过手机APP发送家居网络SSID和密码。
- 步骤3** 创建STA网络接口，并根据SSID和密码完成关联（详细内容请参见“[3 STA功能](#)”）。
- 步骤4** 关闭SoftAp（详细内容请参见“[4 SoftAp功能](#)”）。

----结束

## 返回值

返回值请参见对应模块功能的返回值说明。

## 7.3 注意事项

分时共存下，由于STA和SoftAp各自轮流使用时隙，即使另一个模块无数据也会得到相等的时隙，因此分时共存下性能表现会较差。对于长期业务使用的场景，建议勿使用分时共存模式，将SoftAp启动到STA工作的信道，使用全时共存模式。

## 7.4 编程实例

请参考STA和SoftAp功能的编程实例（详细内容请参见“[3 STA功能](#)”或“[4 SoftAp功能](#)”）。



# 8 Wi-Fi&蓝牙共存

## 8.1 概述

## 8.2 开发流程

## 8.3 注意事项

## 8.4 编程实例

## 8.1 概述

蓝牙（BT，Bluetooth）和Wi-Fi均可能工作在2.4G ISM，因此可能互相干扰。分时是利用蓝牙和Wi-Fi间的握手信号，使蓝牙和Wi-Fi分时在2.4G工作，这样可以避免噪音干扰和阻塞干扰。

802.15.2规定仲裁方式和信号（PTA，Packet Traffic Arbitration）的框架，在蓝牙或Wi-Fi有收发业务时，提交申请给PTA controller（集成在Wi-Fi中），由PTA controller进行许可。

蓝牙和Wi-Fi间的握手信号定义如下：

- 二线共存：wlan\_active、bt\_status
  - Wi-Fi给PTA信号wl\_active：Wi-Fi有收发业务。
  - 蓝牙给PTA信号bt\_status：蓝牙有高优先级业务。蓝牙低优先级业务延时访问。
- 三线共存：wlan\_active、bt\_active、bt\_status
  - Wi-Fi给PTA信号wl\_active：Wi-Fi有收发业务。
  - 蓝牙给PTA信号bt\_active：蓝牙有业务。
  - 蓝牙给PTA信号bt\_status：蓝牙业务状态，bt\_status复用为蓝牙优先级。

## 8.2 开发流程

### 使用场景

需要同时使用Wi-Fi和蓝牙时，开启Wi-Fi&BT共存功能。



## 功能

Wi-Fi&BT共存功能提供的接口如表8-1所示。

表 8-1 Wi-Fi&BT 共存功能接口描述

接口名称	描述
hi_wifi_btcoex_enable	配置Wi-Fi&BT共存开启/关闭的基本参数。

## 返回值

Wi-Fi&BT共存功能的返回值如表8-2所示。

表 8-2 Wi-Fi&BT 共存功能返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	执行成功。
2	HISI_FAIL	-1	执行失败。

## 开发流程

Wi-Fi&BT共存功能开发的典型流程：

- 步骤1** 创建STA网络接口（详细内容请参见“3 STA功能”）。
- 步骤2** 调用hi\_wifi\_btcoex\_enable()，开启Wi-Fi&BT共存，同时使用Wi-Fi和蓝牙功能。
- 步骤3** 调用hi\_wifi\_btcoex\_enable()，开启/关闭Wi-Fi&BT共存，仅使用Wi-Fi功能。
- 结束

## 8.3 注意事项

- Wi-Fi与BT共存仅支持STA模式，不支持SoftAp模式。
- 模组或产品内同时集成了Wi-Fi芯片和蓝牙芯片，建议常开Wi-Fi&BT共存功能，如果需要根据业务动态开启共存功能，请根据业务场景设计调用API接口的策略。

## 8.4 编程实例

示例：通过main函数打开或关闭蓝牙共存功能

```
unsigned int main()
{
    const char *ifname = "wlan0"; /* Wi-Fi STA设备名称 */
    unsigned char en      = 1; /* 开启Wi-Fi&BT共存功能 */
    unsigned char mode    = 2; /* 三线共存模式 */
    unsigned char share_ant = 1; /* Wi-Fi和BT共用天线 */
    unsigned char preempt = 1; /* 发送Wi-Fi&BT共存NULL帧 */
}
```



```
unsigned int ret;

/* 开启Wi-Fi&BT共存功能 */
ret = hi_wifi_btcoex_enable(ifname, en, mode, share_ant, preempt);
if (ret != HISI_OK) {
    printf("enable btcoex failed.\n");
} else {
    printf("enable btcoex success.\n");
}
return ret;
}

unsigned int main()
{
    const char *IFNAME = "wlan0";
    /* 关闭Wi-Fi&BT共存功能 */
    return hi_wifi_btcoex_enable(ifname, 0, 0, 0, 0);
}
```

### 结果验证

```
#enable btcoex success.
#
```



# 9 软件重传功能

## 9.1 概述

## 9.2 开发流程

## 9.3 注意事项

## 9.4 编程实例

## 9.1 概述

重传分为软件重传和硬件重传。

- 硬件重传是指在空口发送中，由于空口干扰，硬件会调整发送速率等级重新发送多次。当硬件重传多次后仍然发送失败，会触发驱动软件层面重传。
- 软件重传会将发送失败的帧重新放入软件发送队列中，通过硬件重新竞争发送窗口进行发送。

## 9.2 开发流程

### 使用场景

由于软件重传会消耗一些资源，因此数据帧只针对vip帧进行软件重传。vip数据帧有DHCP帧、EAPOL帧、ICMP帧、PPPOE帧、ARP帧、ND DHCPV6帧、MESH RPL帧和VO/VI队列中数据帧。管理帧的软件重传只限定AUTH、(RE)ASSOC REQ、(RE)ASSOC RSP和ACTION帧。

用户可在应用层修改IP TOS字段，使数据帧的TID成为VO/VI，同时设置软件重传次数或重传时间，获取业务数据软件多次重传效果。

### 功能

软件重传默认使用重传次数，重传次数默认值是数据帧5次，管理帧3次。

软件重传提供的接口如[表9-1](#)所示。



表 9-1 软件重传功能接口描述

接口名称	描述
hi_wifi_set_retry_params	软件重传次数或重传时间设置

## 开发流程

软件重传功能开发的典型流如下：

- 步骤1** 创建对应的网络接口（详细内容请参见“[3 STA功能](#)”或“[4 SoftAp功能](#)”）。
  - 步骤2** 在应用层修改需要软件重传的数据IP TOS字段值，使数据帧的TID成为VO/VI。使用lwip协议栈建立socket连接时，调用setsockopt接口函数将tos值作为参数传入，VO/VI对应的tos值范围是[128,255]。
  - 步骤3** 调用hi\_wifi\_set\_retry\_params，设置软件重传次数或重传时间。
- 结束

## 返回值

软件重传功能的返回值如[表9-2](#)所示。

表 9-2 软件重传返回值说明

序号	定义	实际数值	描述
1	HISI_OK	0	执行成功。
2	HISI_FAIL	-1	执行失败。

## 9.3 注意事项

- 重传次数的阈值是[0,15]次，重传时间的阈值是[0,200]个时间粒度,时间粒度10ms。
- 软件重传只支持软件重传次数和软件重传时间生效一种。
  - 在设置软件重传次数时，先将软件重传时间置为0。
  - 在设置软件重传时间时，先将软件重传次数置为0。

## 9.4 编程实例

示例：应用层修改ip tos字段值，使数据tid成为vo/vi。

### 代码示例

```
/*  
功能描述：修改ip tos字段  
*/  
static int32_t SendDataSetOption(hi_void)  
{
```



```
/* 将ip tos字段值设为VO/VI */
int32_t tos = 128;
if (setsockopt(context->trafficSock, SOL_IP, IP_TOS, &tos, sizeof(tos)) < 0)
{
    printk(("set reuse failed %d\r\n", errno));
    return HI_FAIL;
}
return HI_SUCCESS;
}

/*****
功能描述：STA启动后设置软件重传时间
*****/
int main(hi_void)
{
    char ifname[] = "wlan0"; /* STA */
    unsigned int retry_time = 5; /* retry time:50ms */
    /* 先将软件重传次数置为0 */
    if(hi_wifi_set_retry_params(ifname, 0, 0) != 0) {
        return -1;
    }
    /* 再设置软件重传时间为5*10ms */
    if(hi_wifi_set_retry_params(ifname, 2, retry_time) != 0) {
        return -1;
    }
    return 0;
}
```