全志科技
Allwinner Technology

# H6 WiFi 模块

# 使用说明书



0.8

2017.02.24

# 文档履历

| 版本号 | 日期 | 制/修订人 | 内容描述 |
| --- | --- | --- | --- |
| 0.8 | 2017.02.24 | | 初始版本 |
| | | | |

# 目录

 

# 1. 前言

## 1.1 编写目的

介绍 WiFi 模组移植配置方法，介绍 SDK 中 WiFi 的 Station、Softap 和 WiFi PPPoE 三个功能的软件结构和 API 函数，目的是让 WiFi 模块的开发和使用人员可以根据该文档可以完成一些 WiFi 的常规工作，解决常见问题。

## 1.2 适用范围

适用于 H6 Android4.4/AndroidN 平台

## 1.3 相关人员

WiFi 模块的开发和使用人员

## 1.4 相关术语

- Station：WiFi 的一种工作模式，处于 Station 模式的盒子可以去连接无线路由器，并通过 WiFi 上网；
- Softap：WiFi 的一种工作模式，处于 Softap 模式的盒子作为无线热点，供其他设备连接，将网络共享给手机、平板等设备；
- WiFi PPPoE: 基于 WiFi 的 PPPoE 宽带拨号，一般盒子与无线猫连接时需要使用该模式；
- WiFi 自适应：自动识别出 WiFi 模组型号，不需要配置来指定 WiFi 模组型号；

# 2. WiFi 模块框架概述

WiFi 模块软件结构框图如下图所示：

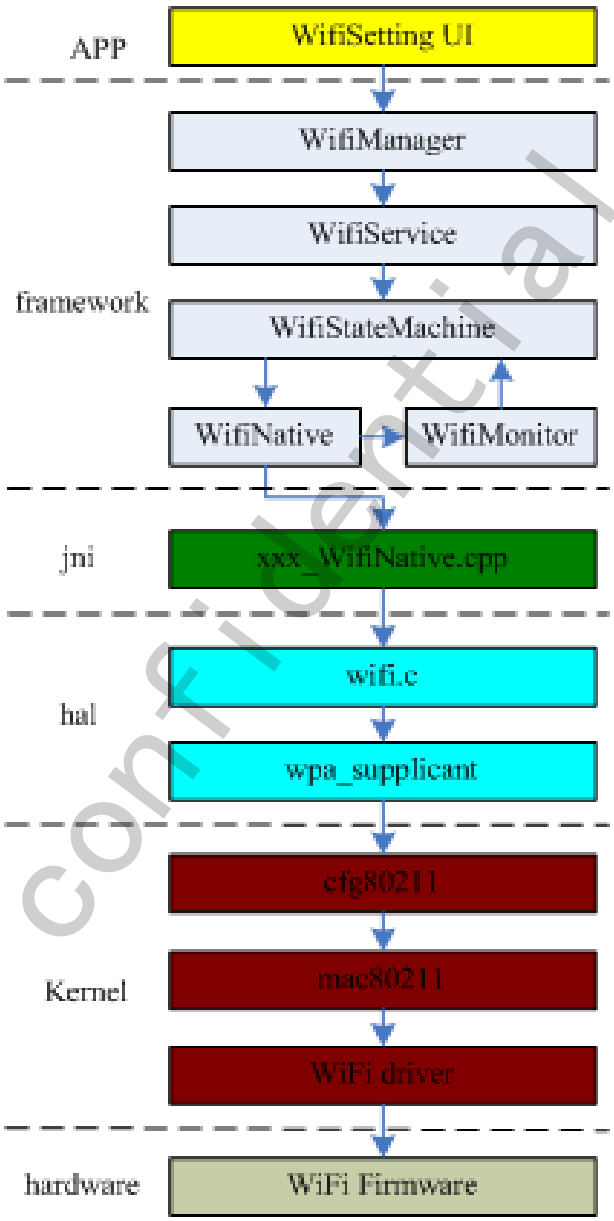


图 1: WiFi 模块软件框图



 

# 2.1 代码分布

Android4.4 和 AndroidN WiFi 相关的代码分布有一下差异，下面分开进行说明：

## 2.1.1 Android4.4 WiFi 代码分布

| 层次 | 路径 |
| --- | --- |
| APP | packages/apps/Settings/src/com/android/settings/wifi |
| framework | frameworks/base/wifi |
| jni | frameworks/base/core/jni/android_net_wifi_WifiNative.cpp |
| hal | hardware/libhardware_legacy/wifi |
| wpa_supplicant | external/wpa_supplicant_8 |
| driver | linux-3.10/driver/net/wireless |
| firmware | hardware/broadcom/wlan/bcmdhd/firmware |

## 2.1.2 AndroidN WiFi 代码分布

| 层次 | 路径 |
| --- | --- |
| APP | packages/apps/Settings/src/com/android/settings/wifi |
| framework | frameworks/base/wifi |
| framework | frameworks/base/opt/net/wifi |
| jni | frameworks/base/opt/net/wifi/service/jni/com_android_server_wifi_WifiNative.cpp |
| hal | hardware/libhardware_legacy/wifi |
| wpa_supplicant | external/wpa_supplicant_8 |
| driver | linux-3.10/driver/net/wireless |
| firmware | hardware/aw/wlan/firmware |

## 2.1.3 WiFi 框架介绍

1. Wifisetting ui 是 WiFi 的界面运用 APP，通过界面的操作，调用到 Framework 层的 API，此 API 由 WifiManager 提供；
2. WifiManger 提供 API 给应用层用，然后接口函数通过异步通道 mAsyncChannel 发送信息给 WifiService 或是通过包含直接调用 WifiService 里面的方法；
3. WifiStateMachine 是 WiFi framework 的逻辑控制中心，管理 WiFi 的各种状态；
4. WifiNative 和 jni 是 frameworks 与 hal 层的沟通桥梁，将 WiFi 的操作命令传递给 wpa_supplicant；
5. WifiMonitor 通过阻塞等到 wpa_supplicant 反馈的命令；
6. wpa_supplicant 是 WiFi 驱动和 Android 层的中转站，wpa_supplicant 通过 nl80211 将命令传递给驱动，同时也负责对协议和加密认证的支持；



 

# 3. WiFi 模块移植配置说明

## 3.1 已支持 WiFi 模组配置说明

SDK 已支持的 WiFi 模组型号请参考《wifi_module_support_list》，对于已支持的 WiFi 模组，分成自适应自测和非自适应支持，《wifi_module_support_list》中有说明；

### 3.1.1 sys_config.fex 配置

对于自适应支持的 WiFi 模组，只需要修改 lichee/tools 方案目录下的 sys_config.fex 即可；

```
[wlan]
wlan_used          = 1
wlan_busnum        = 1
wlan_usbnum        = 3
;wlan_power         = "vcc-wifi"
wlan_io_regulator   = "axp806_bldo3"
wlan_en            = port:PL08<1><default><default><0>
wlan_regon         = port:PM03<1><default><default><0>
wlan_hostwake      = port:PM00<0><default><default><0>
```

说明：

1. ";" 符号起注释作用，如果不需要则将其注释；
2. "wlan_busnum" 表示 WiFi 所使用的 SDIO 控制器号；
3. "wlan_usbnum" 表示 WiFi 所使用的 USB 控制器号；
4. "wlan_power" 表示给 WiFi 模组的 VBAT 供电的 regulator 名称；
5. "wlan_io_regulator" 表示给 WiFi 模组的 GPIO 供电的 regulator 名称；
6. "wlan_en" 表示 wlan_power 给 WiFi 模组的 VBAT 供电，wlan_en 控制供电的通断；
7. "wlan_regon" 表示 WiFi 的功能使用 GPIO；
8. "wlan_hostwake" 表示 WiFi 唤醒主控的 GPIO；



 

9. 以上所有项必须参看原理图进行配置，配置与原理图实际使用的资源保持一致；

以原型机的 AP6356s 为例进行说明，如下图所示：

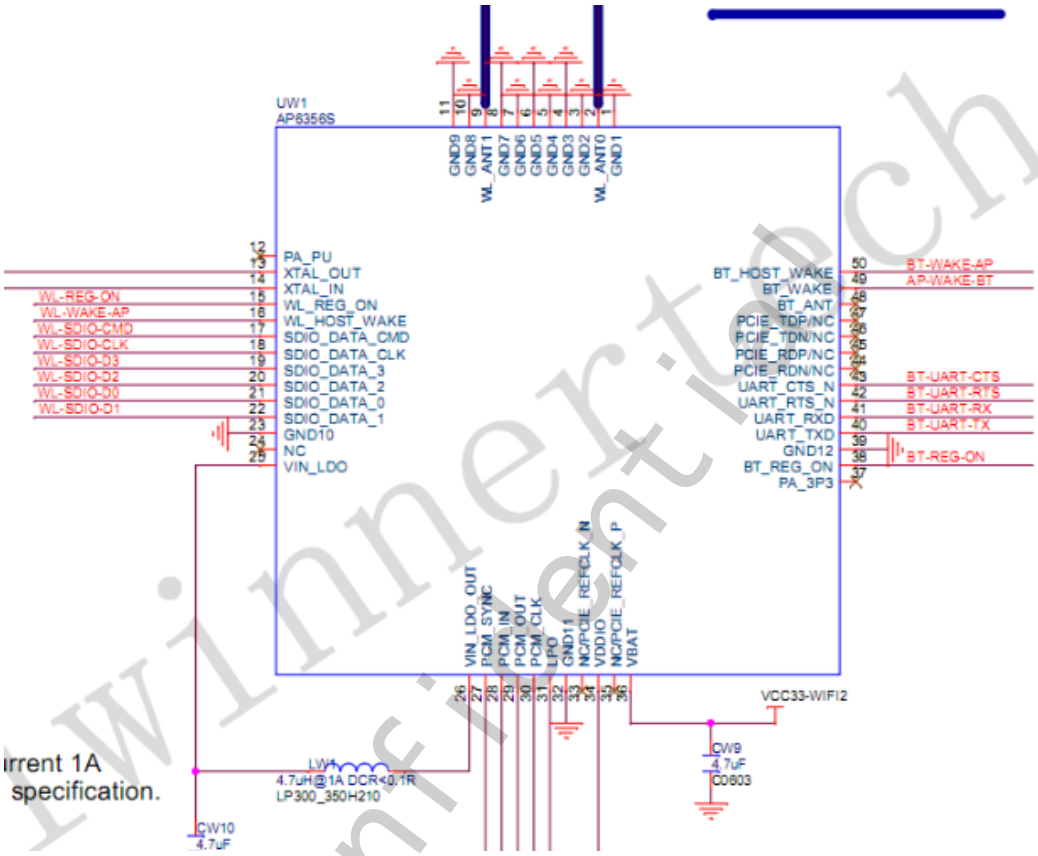


图 2: ap6356s

1. WiFi 使用的是 sdio 接口，且模组的 SDIO_DATA_xxx 连接到了主控的 SDC1-xxx，所有配置"wlan_busnum = 1"，"wlan_usbnum = 3" 可以将其注释掉，也可以留着不做处理；
2. WiFi 模组的 VBAT 是从 VCC-5V 直接转过来的，而 VCC-5V 是一直供电的，所有 VBAT 不需要额外的 regulator 供电，可将"wlan_power" 这一行注释掉；
3. VCC-5V 转成 VCC33-WIFI2 是由 PL08(原理图上连接到 BT-WIFI-ON) 控制通断的，所有配置 wlan_en = port:PL08<1><default><default><0>；
4. WIFi 模组的 VDDIO 连接到了 BLDO3，BLDO3 在 sys_config 中的名称是"axp806_bldo3"，所有配置 wlan_io_regulator = "axp806_bldo3"；
5. WiFi 模组的 WiFi 功能使能脚 WL_REG_ON 连接到主控的 PM03 脚上，所有配置 wlan_regon = port:PM03<1><default><default><0>；

6. WiFi 模组的唤醒主控脚 WL_HOST_WAKE 连接到主控的 PM00 脚上，所有配置 wlan_hostwake = port:PM00<0><default><default><0>；

### 3.1.2 BoardConfig.mk 配置

如果是非自适应支持的 WiFi，除了需要修改 lichee/tools 方案目录下的 sys_config.fex 外，还需要修改 Android 方案目录下 BoardConfig.mk；现以 AP6181 为例进行说明，如需要支持 AP6181，则需要在 BoardConfig.mk 中添加如下配置：

```
WIFI_VENDOR_NAME := broadcom
WIFI_MODULE_NAME := ap6181
WIFI_DRIVER_NAME := bcmdhd
WIFI_DRIVER_FW_PATH_STA := fw_bcm40181a2.bin
WIFI_DRIVER_FW_PATH_AP := fw_bcm40181a2_apsta.bin
WIFI_DRIVER_FW_PATH_P2P := fw_bcm40181a2_p2p.bin
```

说明：

1. "WIFI_VENDOR_NAME" 表示模组的厂商名称，常见的有 broadcom、realtek、xradio；
2. "WIFI_MODULE_NAME" 表示模组的名称，根据实际情况设置；
3. "WiFi_DRIVER_NAME" 表示该模组的驱动名称，broadcom 的 APxxx 系列 WiFi 的驱动是 bcmdhd.ko，所以该处设置成 bcmdhd 即可，realtek 的各个 WiFi 的驱动名称不一样，需根据实际情况设置；xradio 的驱动名称一般设置成 xradio_wlan；
4. "WIFI_DRIVER_FW_PATH_STA" 表示 Station 模式的 firmware 名字，broadcom 的 WiFi 一般需要根据实际使用的 firmware 名字进行设置，realtek 和 xradio 的 WiFi 一般设置成"WIFI_DRIVER_FW_PATH_STA := STA" 即可；
5. "WIFI_DRIVER_FW_PATH_AP" 表示 Softap 模式的 firmware 名字，broadcom 的 WiFi 一般需要根据实际使用的 firmware 名字进行设置，realtek 和 xradio 的 WiFi 一般设置成"WIFI_DRIVER_FW_PATH_STA := AP" 即可；
6. "WIFI_DRIVER_FW_PATH_P2P" 表示 Softap 模式的 firmware 名字，broadcom 的 WiFi 一般需要根据实际使用的 firmware 名字进行设置，realtek 和 xradio 的 WiFi 一般设置成"WIFI_DRIVER_FW_PATH_P2P := P2P" 即可；



 

# 3.2 如何添加支持一款尚未支持的 WiFi 模组

目前市面上使用较多的 WiFi 模组是 broadcom 和 realtek 两家的，下面就以 broadcom 和 realtek 为例来说明如何添加支持一款尚未支持的 WIFi 模组；

## 3.2.1 添加支持一款新的 broadcom WiFi 模组

SDK 中已经集成了 broadcom WiFi 驱动 bcmdhd，顾无需再移植驱动；需要做的移植工作有如下几部分：

1. firmware 移植如果是 Android4.4 系统，则在 android/hardware/broadcom/wlan/bcmdhd/firmware 目录下建立对应的文件夹，将 firmware 拷贝到该文件夹下，参考 ap6356s 在该文件夹下新建 device-bcm.mk，修改 android/hardware/broadcom/wlan/bcmdhd/firmware/firmware-bcm.m，将新建的 device-bcm.mk 包含编译；
2. sys_config.fex 配置参考 3.1.1 节配置 sys_config.fex；
3. BoardConfig.mk 配置参考 3.1.2 节配置 BoardConfig.mk

## 3.2.2 添加支持一款新的 realtek WiFi 模组

realtek WiFi 一般需要移植驱动，而不需要移植 firmware；需要做的移植工作有如下几部分：

### 3.2.2.1 驱动移植

首先需要向 WiFi 原厂申请 WiFi 驱动代码，将驱动放到 SDK 的 lichee/linux/driver/net/wireless 目录下面；修改 lichee/linux-3.10/driver/net/wireless/Kconfig，添加：

```
source "drivers/net/wireless/rtlxxx/Kconfig"
```

修改 lichee/linux-3.10/driver/net/wireless/Makefile，添加：



 

```
obj-S(CONFIG_RTLxxx) += rtlxxx/
```

修改驱动的 Makefile 文件，realtek WiFi 驱动一般选择默认选择 I386 平台，即在 Makefile 中定义"CONFIG_PLATFORM_I386_PC = y"，需要修改成如下：

```
CONFIG_PLATFORM_I386_PC = n
...
CONFIG_PLATFORM_ARM_SUNxI = y
```

如果新添加的这款 WiFi 是 sdio 接口的，则对比 rtl8723bs/platform/platform_ARM_SUNnI_sdio.c 修改驱动代码目录下面的 platform/ platform_ARM_SUNxI_sdio.c，修改后两者代码内容一样；如果新添加的这款 WiFi 是 usb 接口的，则对比 rtl8723bs/ platform/ platform_ARM_SUNxI_usb.c 修改驱动代码目录下面的 platform/platform_ARM_SUNxI_usb.c，修改后两者代码内容一样；

#### 3.2.2.2 sys_config.fex 配置

参考 3.1.1 节配置 sys_config.fex；

#### 3.2.2.3 BoardConfig.mk 配置

参考 3.1.2 节配置 BoardConfig.mk



 

# 4. WiFi 模块使用说明

## 4.1 Station 相关使用说明

WiFi Station 功能的主要 API 函数介绍如下：

boolean setWifiEnabled(boolean enabled)
作 用：打开或者关闭WiFi功能，Settings中的WiFi开关按钮便是调用该函数；
参 数：true：打开；false：关闭；
返回值：true：操作成功；flase：操作失败；

boolean isWifiEnabled()
作 用：判断当前WiFi是否已经打开；
参 数：无
返回值：true：已打开；flase：未打开；

int getWifiState()
作 用：获取WiFi的当前状态；
参 数：无
返回值：WIFI_STATE_DISABLED：已关闭
WIFI_STATE_DISABLING：正在关闭
WIFI_STATE_ENABLED：已打开
WIFI_STATE_ENABLING：正在打开
WIFI_STATE_UNKNOWN：未知状态

boolean startScan()
作 用：请求扫描，该函数立即反馈，扫描结果异步通知，通过getScanResults获得扫描结果；
参 数：无
返回值：true：操作成功；flase：操作失败



 

List<ScanResult> getScanResults()
作 用：获取最近一次扫描结果
参 数：无
返回值：最近一次扫描得到的AP列表；

int addNetwork(WifiConfiguration config)
作 用：将一个AP的信息配置到wpa_supplicant.conf的network中去，连接网络前需要先添加配置；
参 数：AP的信息，包括SSID，password，加密方式等；
返回值：返回一个网络ID；

boolean enableNetwork(int netId, boolean attemptConnect)
作 用：addNetwork添加一个AP的配置后，默认状态是DISABLED的，需要将其状态变成ENABLED才可以完成连接；
参 数：netid：addNetwork返回的网络号；
attempConnect：true：自动完成连接；false：不自动连接，需要调用reconnect才可以连接；

更多更详细的 WiFi API 函数说明请到网页"https://developer.android.google.cn" 上查阅；使用范例请参考 android/packages/app/Settings/src/com/android/settings/wifi 中的代码。

## 4.2 Softap 相关使用说明

WiFi Softap 功能相关的 API 函数介绍如下：

boolean setWifiApEnabled(WifiConfiguration wifiConfig, boolean enabled)
作 用：打开或者关闭Softap功能；
参 数：wifiConfig：指定Softap的SSID、security和channel等信息；
enable：true：打开；false：关闭；
返回值：true：操作成功；false：操作失败；



 

boolean isWifiApEnabled()
作 用：判断是否Softap已打开；
参 数：无
返回值：true：已打开；false：未打开；

int getWifiApState()
作 用：获取当前Softap状态；
参 数：无
返回值：WIFI_AP_STATE_DISABLED：已关闭
WIFI_AP_STATE_DISABLING：正在关闭
WIFI_AP_STATE_ENABLED：已打开
WIFI_AP_STATE_ENABLING：正在打开
WIFI_AP_STATE_FAILED：失败状态

boolean setWifiApConfiguration(wifiConfiguration wifiConfig)
作 用：配置Softap的信息；
参 数：wifiConfig：指定Softap的SSID、security和channel等信息；
返回值：true：设置成功；false：设置失败

使用范例请参考 android/packages/app/Settings/src/com/android/settings/wifi 中的代码 WifiApDialog.java 和 WifiApEnabler.java 这两个文件。



## 4.3 WiFi PPPoE 相关使用说明

Android 原生不支持 WiFi PPPoE，WiFi PPPoE 功能是由 Allwinnertech 添加支持的，用于基于 WiFi 向无线猫进行 PPPoE 拨号上网；Android4.4 和 AndroidN 上 WiFi PPPoE 的实现方式不一样，顾下面分开介绍；

## 4.3.1 Android4.4 WiFi PPPoE 功能介绍

1. 实现方式说明 Android4.4 上实现的 WiFi PPPoE 与 DHCP 平级，作为 WiFi 连接过程中获取 IP 的一种方式，当选择 WiFi PPPoE 时，则获取 IP 阶段不执行 DHCP，而是改成执行 PPPoE 拨号；WiFi PPPoE 的连接流程如下图所示：

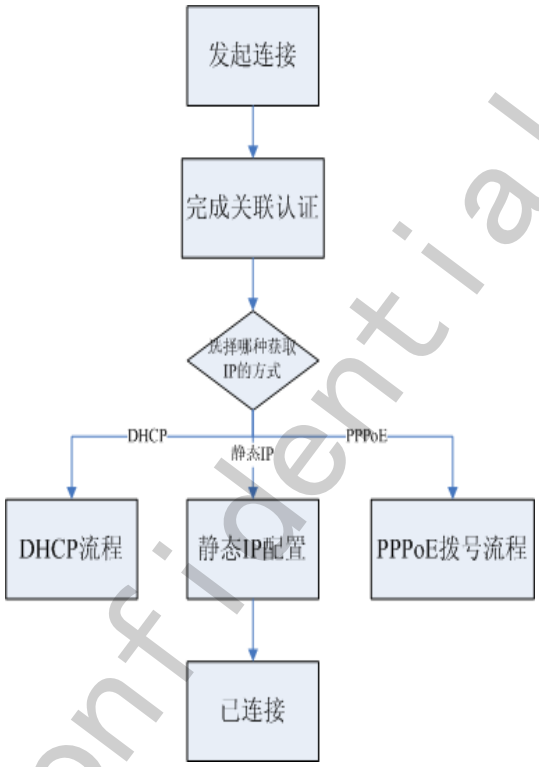


图 3: Android4.4 WiFi PPPoE 拨号流程

2. SDK 自带 Settings 操作说明在扫描列表中选中需要去进行 WiFi PPPoE 拨号的 SSID，弹出连接页面，输入正确的 WiFi 密码，选中" 显示高级选项" 复选框，"IP 设置" 选择"PPPoE"，如下图所示：



 

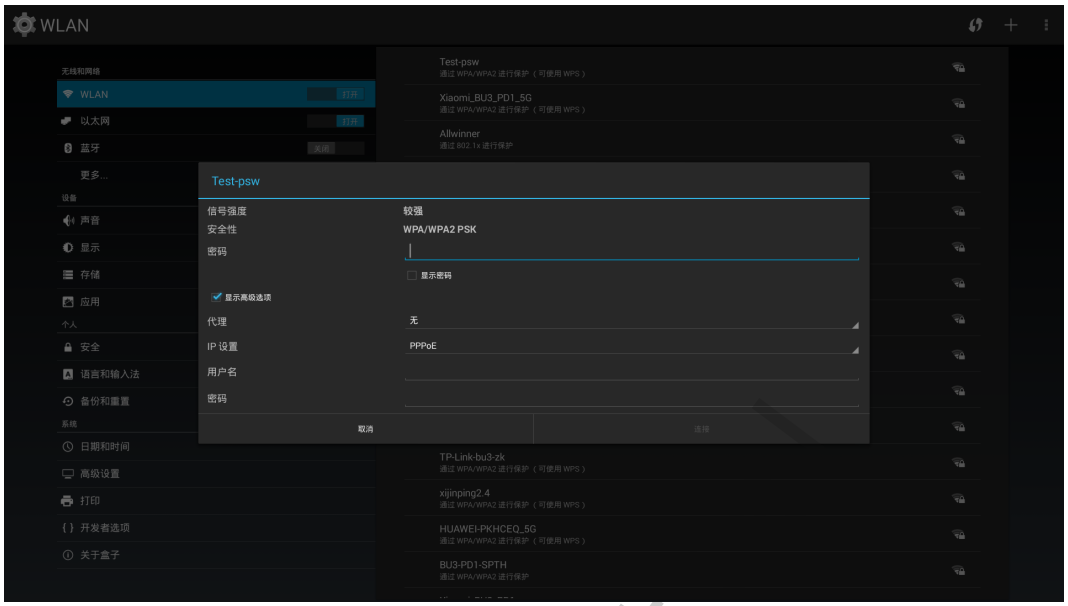


图 4: Android4.4 WiFi PPPoE 拨号设置

输入正确的 PPPoE 用户名和密码，点击连接开启 WiFi PPPoE 拨号;

### 4.3.2 AndroidN WiFi PPPoE 功能介绍

1. 实现方式说明 AndroidN 上实现的 WiFi PPPoE 作为一种新的网络存在，TYPE_PPPOE 与 TYPE_WIFI 平级存在，且 TYPE_PPPOE 的优先级高于 TYPE_WIFI；WiFi PPPoE 的连接流程如下图所示：



 

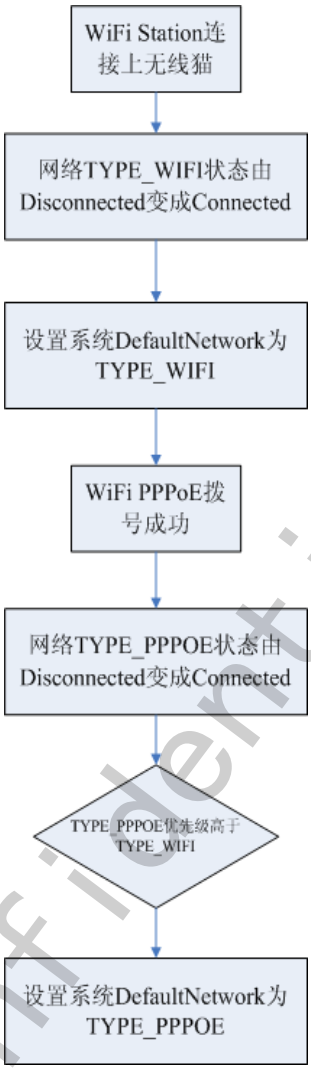


图 5: AndroidN WiFi PPPoE 拨号流程

2. SDK 自带 Settings 操作说明 AndroidN 的 WiFi PPPoE 拨号前，需要先将 WiFi Station 连接到无线猫，对于不支持 DHCP Server 的无线猫，是无法完成 WiFi PPPoE 拨号的；SDK 自带 Settings 中点击 PPPoE 选项，进入 PPPoE 设置界面，PPPoE 设置界面如下图所示：



 

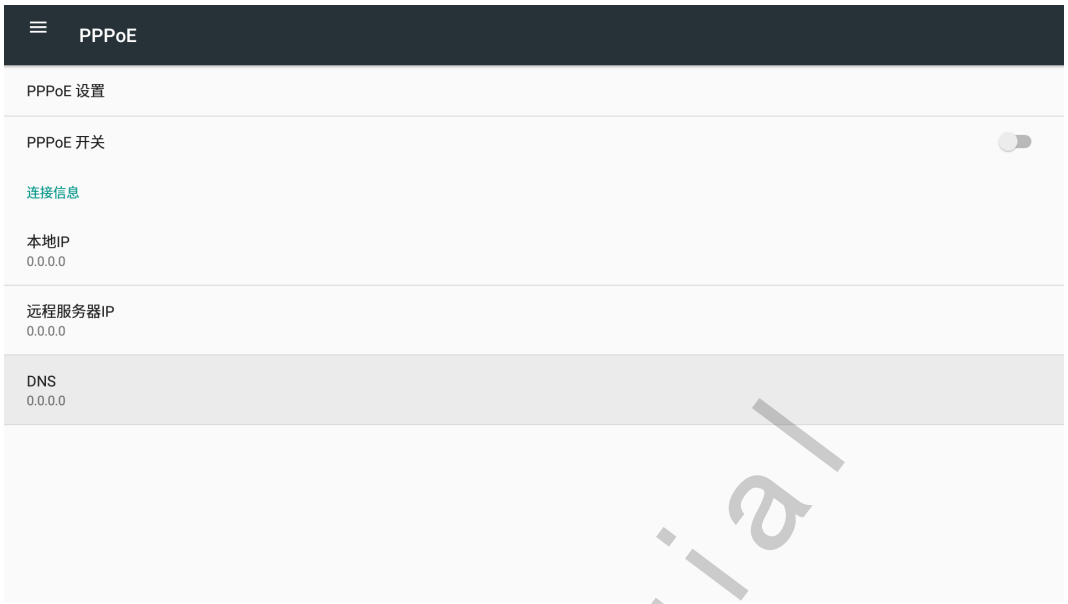


图 6: AndroidN WiFi PPPoE 拨号设置 1

点击"PPPoE 设置" 按钮，弹出"PPPoE 配置" 框，如下图所示，将" 接口名称" 选择成 wlan0，设置 PPPoE 的" 用户名" 和" 密码"，点击确定。

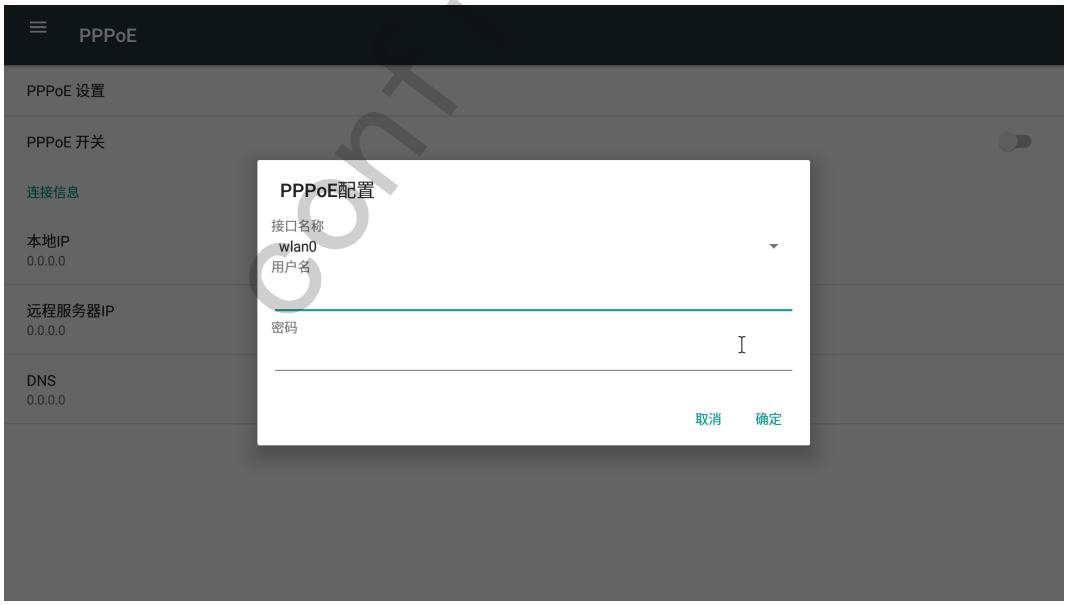


图 7: AndroidN WiFi PPPoE 拨号设置 2

点击"PPPoE 开关"，打开开关后即可进行 PPPoE 拨号；



 

3. 对接第三方 Settings

(PppoeManager)getSystemService(Context.ETHERNET_SERVICE)
获取PppoeManager对象

mPppoeManager.setupPppoe(String ifname, String user, String password)
配置PPPoE拨号的网口名称(WiFi PPPoE应该设为wlan0)、PPPoE的用户名、PPPoE的密码

mPppoeManager.connectPppoe(String ifname)
开启拨号

mPppoeManager.getPppoelinkProperties()
获取PPPoE的IP地址、默认网关、dns等信息;

mPppoeManager.disconnectPppoe()
断开拨号

mPppoeManager.getPppoeState(ifname)
获取当前PPPoE的状态，有:PPPOE_STATE_DISCONNECTED,
PPPOE_STATE_CONNECTING,
PPPOE_STATE_CONNECTED,
PPPOE_STATE_DISCONNECTING

详细的 API 函数请阅读 sdk 中的 android/frameworks/base/core/java/android/net/PppoeManager.java 使用范例请看到 android/packages/apps/Settings/src/com/android/settings/pppoe。



 

# 5. FAQ

## 5.1 修改 BoardConfig.mk 后如何编译才生效

由于 BoardConfig.mk 术语 Makefile 文件，仅仅修改了 Makefile 文件而没有修改源代码文件，编译是不会生效的；修改 BoardConfig.mk 后编译的方法有两个：

1. 整个 Android 目录"make clean" 后，再使用"make -j8" 完整编译 Android，该方法比较耗时，推荐使用方法 2，如果方法 2 不起效再使用方法 1；
2. 进到 android/hardware/libhardware_legacy/目录执行"mm -B -j8"；如果是 android4.4 进入 android/hardware/broadcom/wlan/bcmdhd/wpa_supplicant_8_lib/目录下执行"mm -B -j8"，如果是 AndroidN 进入 android/hardware/aw/wlan/wpa_supplicant_8_lib/目录下执行"mm -B -j8"；进到 android/external/wpa_supplicant_8/目录执行"mm -B -j8"；最后回到 android 主目录下使用"make -j8" 编译整个 Android；

## 5.2 WiFi 模组调试过程中常见问题排查

WiFi 模组调试过程中，经常会遇到 Settings 中 WiFi 开关按钮无法使能，或者 WiFi 开关按钮使能后，一直无法搜索到 AP 等问题；可以按照如下步骤进行排查；

1. 查看串口中的 log 信息，如果看到很多 sunxi-mmc 相关的 RTO 相关的错误打印，而没有看到"SDIO card at address 0001" 相关的打印，则说明 sdio 扫卡失败，需要检查模组上电是否 ok；"echo 1 > /sys/devices/virtual/misc/sunxi-wlan/rf-ctrl/power_state"，软件上给模组上电，然后使用万用表测量模组的 WL_REG_ON 引脚是否为高电平，测试 VCC-WIFI 和 VCC-WIFI-IO 是否都上电，以原型机上的 AP6356s 为例，WL_REG_ON 脚是否为高电平，VBAT 脚是否有 3.3v，VDDIO 脚是否有 3.3v 或者 1.8v，如果上电不正常，则参考 2.1 节说明配置 sys_config；
2. 如果 sdio 扫卡成功，可以通过"ifconfig wlan0" 查看是否已经生成了网口，如果"ifconfig wlan0" 返回"No such device"，且正在调试的 WiFi 并非自适应支持，则很有可能是 BoardConfig.mk 中的驱动名字 (WIFI_DRIVER_NAME) 配置错误，如果返回 wlan0 的状态为 down，则很有可能是 BoardConfig.mk 中的



 

firmware 名 字 (WIFI_DRIVER_FW_PATH_STA、WIFI_DRIVER_FW_PATH_AP、WIFI_DRIVER_FW_PATH_P2P) 配置错误，请重新确认 BoardConfig.mk 的配置；

3. 如果网口 wlan0 状态是 up，使用命令"ps | grep supplicant" 发现 wpa_supplicant 并没有运行起来，则很可能存在软件问题，可反馈给 Allwinnertech 解决；
4. 如果网口 wlan0 状态是 up，使用命令"ps | grep supplicant" 发现 wpa_supplicant 已经运行起来，但是还是搜索不到 AP，此时很可能是 24M 或者 26M 晶振存在问题，需要对晶振电路做检查并做频偏测试；

## 5.3 WiFi 吞吐量偏低怎么办

WiFi 吞吐量一般受环境影响较大，为了测试得到比较准确的吞吐量性能，建议在屏蔽房或者空旷干扰较少的环境下测试；其次，吞吐量也和路由器的性能相关，如果路由器的性能较低，则测试吞吐量低的瓶颈可能在路由器端，所以建议测试 WiFi 性能时选用较好的路由器，而不是随便找一个；WiFi 吞吐量性能一般和天线以及匹配电路的关系很大，如果吞吐量偏低，建议先做 RF 指标测试和天线匹配测试，测试方法可参看《RF 测试说明文档》；

## 5.4 如何获取 WiFi 的 MAC 地址、IP 地址、Gateway 地址

1. 获取 WiFi 的 MAC 地址

```
WifiInfo wifiInfo = wifiManager.getConnectonInfo();
String mac = wifiInfo.getMacAddress();
```

2. 获取 WiFi 的 IP 地址

```
ConnectivityManager cm = (ConnectivityManager) context
    .getSystemService(Context.CONNECTIVITY_SERVICE);
LinkProperties prop = cm.getActivityLinkProperties();
String ipAddress = formatIpAddress(prop);
private static String formatIpAddress(LinkProperties prop) {
```



 

```
if (prop == null) return null;
Iterator<InetAddress> iter = prop.getAllAddress().iterator();
if (!iter.hasNext()) return null;
String addresses = "";
while (iter.hasNext()) {
    addresses += iter.next().getHostAddress();
    if (iter.hasNext()) addresses += "\n";
}
return addresses;
}
```

3. 获取 WiFi 的 Gateway 地址

```
DhcpInfo mDhcpInfo = wifiManager.getDhcpInfo();
String gateway = NetworkUtils.intToInetAddress(mDhcpInfo.gateway)
    .getHostAddress();
```



 

# 6. Declaration

This document is the original work and copyrighted property of Allwinner Technology ( “Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.



 