

密级状态：绝密( ) 秘密( ) 内部( ) 公开( ☒ )

## Rockhip android 7.1 WIFI/BT 配置说明

(技术部，系统产品一部)

文件状态：  [ ] 正在修改  [ <input checked="" type="checkbox"/> ] 正式发布	当前版本：	V1.4
	作 者：	许学辉
	完成日期：	2017-03-08
	审 核：	胡卫国
	完成日期：	

福州瑞芯微电子有限公司

Fuzhou Rockchips Semiconductor Co., Ltd

(版本所有, 翻版必究)

## 版本历史

版本号	作者	修改日期	修改说明	备注
V1.0	许学辉	2017-01-23	初稿	
V1.1	许学辉	2017-02-14	添加新模块调试方法	
V1.2	许学辉	2017-02-15	添加 wifi 兼容软硬件注意事项	
V1.3	许学辉	2017-02-16	添加编译 wifi ko 注意事项	
V1.4	许学辉	2017-03-08	wifi 驱动 build in 方式编译说明	

## 目 录

<b>1</b>	<b>目的.....</b>	<b>2</b>
1.1	内核注意事项.....	2
1.2	ANDROID 注意事项 .....	3
<b>2</b>	<b>WIFI/BT 兼容原理简要说明.....</b>	<b>4</b>
2.1	WIFI 芯片识别流程 .....	4
<b>3</b>	<b>新 WIFI 模块调试 .....</b>	<b>5</b>
3.1	WIFI 驱动移植.....	5
3.2	WIFI.C 添加 WIFI 兼容 .....	5
<b>4</b>	<b>WIFI 兼容软硬件注意事项 .....</b>	<b>7</b>
<b>5</b>	<b>WIFI KO 编译注意事项 .....</b>	<b>8</b>
<b>6</b>	<b>WIFI 无法打开问题排查 .....</b>	<b>9</b>

# 1 目的

明确 android 7.1 平台上 wifi、bt 自动兼容原理和注意事项,按照 RK 提供的编译固件方法生成固件后,默认就可以支持相应的 wifi 模块,并且一套固件可以支持多个 WIFI 模块。

目前 android 7.1 平台 wifi、bt 模块 android 和 kernel 无需做任何配置。

## 1.1 内核注意事项

wifi 全自动兼容方案 AP6xxx 系列 wifi 和 Realtek 系列 wifi 驱动必须采用 module 方式,不能 build in 到内核 kernel.img 中。android 7.1 各个平台上内核使用的 config 需要最先确认。确认使用的 defconfig 是将 wifi 驱动编译成 ko modules 的配置,defconfig 参考如下配置:

CONFIG\_WIFI\_LOAD\_DRIVER\_WHEN\_KERNEL\_BOOTUP=n

CONFIG\_RTL8188EU=m

CONFIG\_RTL8189ES=m

CONFIG\_RTL8723BS=m

CONFIG\_RTL8188FU=m

CONFIG\_RTL8822BS=m

CONFIG\_RTL8822BU=m

CONFIG\_RKWIFI=m

Device Drivers --->

[\*] Network device support --->

[\*] Wireless LAN --->

```
--- Realtek wireless Device Driver Support
<M> Realtek 8188E USB WiFi
<M> Realtek 8189ES/ETV SDIO WiFi support
< > Realtek 8192CU USB WiFi support
< > Realtek 8192DU USB WiFi support
< > Realtek 8723AU USB WiFi support
< > Realtek 8723BU USB WiFi
<M> Realtek 8723B SDIO or SPI WiFi
< > Realtek 8723BS_VQ0 WiFi
< > Realtek 8723C SDIO or SPI WiFi
< > Realtek 8723D SDIO or SPI WiFi
< > Realtek 8812AU USB WiFi support
< > Realtek 8189F SDIO WiFi
<M> Realtek 8188F USB WiFi
<M> Realtek 8822BS SDIO WiFi
<M> Realtek 8822B USB WiFi
```

板级 dts 无需配置 WIFI 芯片类型(配置了也可以),因为加载 wifi 驱动不依赖 wifi\_chip\_type 节点,如果 WIFI 没有根据 RK 发布的硬件参考设计,板级 dts 先确认如下信息:

```
wireless-wlan {

    compatible = "wlan-platdata";

    wifi_chip_type = "ap6354"; //或者配置为 wifi_chip_type = "";

    WIFI,poweren_gpio = <&gpio2 GPIO_D2 GPIO_ACTIVE_HIGH>; //wifi power

    WIFI,host_wake_irq = <&gpio0 GPIO_D4 GPIO_ACTIVE_HIGH>;

    status = "okay";

};
```

说明：目前 WIFI 完全兼容方案，基于 RK 发布的 WIFI 参考设计，WIFI 上电管脚默认高电平有效，具体项目需要确认 WIFI 供电管脚和高低有效情况。

如果一套固件要做到全部兼容 RK support list 中的 WIFI，硬件板型的 WIFI 供电管脚(所有板硬件型要保持一致)以及 SDIO 电平都需要提前确认，在本文第四章节有详细硬件注意事项。

## 1.2 Android 注意事项

Android 根目录已经集成了 wifi ko 的编译和拷贝。

脚本位置:device/rockchip/common/build\_wifi\_ko.sh，该脚本在 android 源码根目录执行 build.sh 的时候会自动进行。在 build.sh 脚本如下位置：

```
# build wifi ko
echo "start build wifi ko"
lunch rk3328_box-userdebug
source device/rockchip/common/build_wifi_ko.sh

#!/bin/bash
. build/envsetup.sh >/dev/null && setpaths
export PATH=$ANDROID_BUILD_PATHS:$PATH
TARGET_PRODUCT=`get_build_var TARGET_PRODUCT`
TARGET_ARCH=`get_build_var TARGET_ARCH`
ANDROID_BUILD_TOP=`get_build_var ANDROID_BUILD_TOP`
echo ANDROID_BUILD_TOP=$ANDROID_BUILD_TOP
echo TARGET_PRODUCT=$TARGET_PRODUCT
if [ "$TARGET_PRODUCT" == "rk3228h" ];then
    TARGET_ARCH="arm64"
fi
echo TARGET_ARCH = $TARGET_ARCH
echo "---- make wifi ko ----"
make -C kernel ARCH=$TARGET_ARCH modules -j18
find $ANDROID_BUILD_TOP/kernel/drivers/net/wireless/rockchip_wlan/* -name "*.ko" | \
xargs -n1 -i cp {} $ANDROID_BUILD_TOP/vendor/rockchip/common/wifi/modules
```

编译WIFI ko

拷贝到android源码vendor目录

## 2 wifi/bt 兼容原理简要说明

### 2.1 wifi 芯片识别流程

1. 开机对 wifi 模块上电，并自动进行扫描 sdio 操作
2. 系统启动打开 wifi 操作时，分别对系统 sys/bus/sdio (sdio wifi)， sys/bus/usb(usb wifi)， sys/bus/pic (pcie wifi )文件系统下的 uevent 进行读取
3. 获取到 wifi 芯片 vid pid 加载相应的 wifi ko 驱动
4. 识别到相应的 wifi 模块后，即可知道相应的 bt 型号，走不同的 bluedroid 协议栈

**核心代码目录： android /hardware/libhardware\_legacy/wifi**

**kernel/net/rfkill/rfkill-wlan.c**

**packages/apps/Bluetooth**

**hardware/realtek/rtkbt**

## 3 新 wifi 模块调试

目前对外发布的 android 7.1 SDK，WIFI 自动兼容框架已经搭建完毕，如果客户需要自行调试其他模块，只需按照本章节提到的修改地方修改即可。

### 3.1 wifi 驱动移植

RK 平台上所有的 WIFI 模块驱动都是放到内核 kernel/drivers/net/wireless/rockchip\_wlan 目录，一般移植新的 WIFI 驱动，需要在 kernel/drivers/net/wireless 目录添加相应的 wifi 模块的 Kconfig 和 Makefile，有的模块还需要修改 wifi 驱动的 Kconfig 和 Makefile（根据特定的 wifi 模块驱动），如果采用 Realtek 的模块，可以参考 RealTek wifi 驱动移植说明\_V1.1.pdf 文档。

内核能正确编译出 wifi ko 驱动文件后，参考本文 V1.2 章节将 ko 在编译 android 阶段放到相应源码目录。

**注意：**由于目前 wifi 驱动是采用 ko 方式，如果有修改内核网络相关配置，一定要重新编译 ko，否则很可能导致 wifi ko 和内核网络协议栈不匹配。

### 3.2 wifi.c 添加 wifi 兼容

#### 1. 添加 wifi 名称和 wifi vid pid

source\_code/hardware/libhardware\_legacy/wifi/rk\_wifi\_ctrl.c 代码 **supported\_wifi\_devies[]** 结构体中添加 wifi 模块的名称和对应 vid pid，vid pid 可以根据下面章节手动读取 uevent 进行查看，以 AP6356S 为例：AP6356S 为模块名称，02d0:4356 为 vid pid，如下列表中已经添加了几款 wifi 的兼容，参考如下格式添加：

```
typedef struct _wifi_devices
{
    char wifi_name[64];
    char wifi_vid_pid[64];
}wifi_device;

static wifi_device supported_wifi_devices[] = {
    {"RTL8188EU", "0bda:0179"},
    {"RTL8723BU", "0bda:b720"},
    {"RTL8723BS", "024c:b723"},
    {"RTL8822BS", "024c:b822"},
    {"RTL8188FU", "0bda:f179"},
    {"RTL8822BU", "0bda:b82c"},
    {"RTL8189ES", "024c:8179"},
    {"RTL8189FS", "024c:f179"},
    {"RTL8192DU", "0bda:8194"},
    {"RTL8812AU", "0bda:8812"},
    {"SSV6051", "3030:3030"},
    {"ESP8089", "6666:1111"},
    {"AP6354", "02d0:4354"},
    {"AP6330", "02d0:4330"},
    {"AP6356S", "02d0:4356"},
    {"AP6335", "02d0:4335"}
};
```

## 2. 添加 wifi 驱动 ko 文件存放路径

hardware/libhardware\_legacy/wifi/wifi.c 中 **wifi\_ko\_file\_name module\_list[]** 结构体存放的是 wifi 模块的 ko 驱动存放路径和加载 wifi ko 驱动所需的参数，wifi ko 存放路径统一采用 XXXX\_DRIVER\_MODULE\_PATH 的命名方式。

如果 ismod wifi ko 不需要带参数，那么可以使用 UNKKOWN\_DRIVER\_MODULE\_ARG，如果需要额外参数请根据 wifi 模块的移植文档进行相应处理。

**注意：**wifi 名称要与 supported\_wifi\_devices[] 结构体中定义的名称一样。

```
wifi_ko_file_name module_list[] =
{
    {"RTL8723BU", RTL8723BU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"RTL8188EU", RTL8188EU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"RTL8192DU", RTL8192DU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"RTL8822BU", RTL8822BU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"RTL8822BS", RTL8822BS_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"RTL8188FU", RTL8188FU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"RTL8189ES", RTL8189ES_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"RTL8723BS", RTL8723BS_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"RTL8812AU", RTL8812AU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"RTL8189FS", RTL8189FS_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"SSV6051", SSV6051_DRIVER_MODULE_PATH, SSV6051_DRIVER_MODULE_ARG},
    {"ESP8089", ESP8089_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"AP6335", BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"AP6330", BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"AP6354", BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"AP6356S", BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"APXXX", BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
    {"UNKNOW", DRIVER_MODULE_PATH_UNKNOW, UNKKOWN_DRIVER_MODULE_ARG}
};
```



## 4 wifi 兼容软硬件注意事项

目前发布的 SDK 一套固件可以兼容 sdio 2.0 和 sdio 3.0 wifi, sdio2.0 clk 最高跑 50M, sdio 3.0 clk 最高跑 150M

wifi 自动兼容方案可以做到一套固件兼容多个 wifi, 特别需要注意: SDIO wifi io 参考电压, sdio 2.0 wifi 和 sdio 3.0 wifi 在硬件设计有下面两种选择: (**强烈推荐第一种参考设计**)

### 1. 硬件对于 sdio wifi 动态设计的板型

也即是支持 sdio3.0 wifi 模块 wccio\_wl 设计为 1.8v, 支持 sdio2.0 的 wifi 模块 wccio\_wl 设计为 3.3v

### 2. 硬件上所有 wifi 的 wccio\_wl 统一为 1.8v, 但内核软件需要如下修改(注意红色字体部分):

```
kernel/drivers/mmc/host/rk_sdmmc.c
void dw_mci_sdio_switch_iovel(struct mmc_host *mmc, int enable)
{
    struct dw_mci_slot *slot = mmc_priv(mmc);
    struct dw_mci *host = slot->host;
    const struct dw_mci_rockchip_priv_data *priv = host->priv;

    switch (priv->ctrl_type) {
    case DW_MCI_TYPE_RK322XH:
        regmap_write(host->grf, RK322XH_GRF_SOC_CON4,
-           (enable << 3) | ((1 << 3) << 16));
+           (1 << 3) | ((1 << 3) << 16));
        if (enable)
            host->bus_hz = 150000000;
        else
            host->bus_hz = 50000000;
        break;
    default:
        pr_info("%s not switch iovel.\n", mmc_hostname(host->mmc));
    }
}
```

## 5 wifi ko 编译注意事项

本章节主要说明内核网络相关配置修改，对应 wifi ko 驱动的编译方法。

wifi ko 要跟内核网络配置编译出的 kernel.img 一致，如果内核有修改网络配置，以 RK3328 为例，若需要支持 VLAN，内核 rk322xh\_android-7.1\_defconfig 需要打开内核如下配置：

```
CONFIG_VLAN_8021Q=y
```

```
CONFIG_VLAN_8021Q_GVRP=y
```

```
CONFIG_VLAN_8021Q_MVRP=y
```

如上修改后 wifi ko 必须重新编译，可以按照如下两种方法进行编译 ko，**建议使用方法 1**，方法 1 执行脚本后会先 make wifi ko，再将 wifi ko 拷贝到源码 vendor/rockchip/common/wifi/modules 目录，最后编译 android 并生成固件（参考本文 V1.2 章节）；方法 2 只会重新编译 ko,并拷贝到源码 vendor/rockchip/common/wifi/modules 目录。

**方法 1:** android 根目录执行：

```
souce build.sh
```

**方法 2:** android 根目录执行：

```
lunch rk3328_box-userdebug && souce device/rockchip/common/build_wifi_ko.sh
```

注意：**每次修改了内核网络配置，都需要重新编译 ko**，如果没有修改网络配置（使用 RK 发布 SDK 默认的网路配置），编译一次 ko 后，以后系统可以通用这些 ko，采用以上方法系统也会编译出 ko， git status 同样会看到新编译的 ko， git status 查看结果如下：

```
xxh@RD-DEP1-SERVER-163:~/work/rk3328/vendor/rockchip/common/wifi/modules$ cd ../
xxh@RD-DEP1-SERVER-163:~/work/rk3328/vendor/rockchip/common/wifi$ git status
on branch xxh_develop
Your branch is up-to-date with 'rk/rk32/mid/7.0/develop'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    modules/8188eu.ko
    modules/8723bs.ko
    modules/8822bs.ko
    modules/8822bu.ko
    modules/bcmdhd.ko

nothing added to commit but untracked files present (use "git add" to track)
```

红色方框部分表明是新编译出来的 wifi ko，**建议用 git add 命令合入到工程。**

## 6 Wifi 无法打开问题排查

1. 首先确认开机后系统是否有 USB WIFI 或者 SDIO wifi 设备, 正常开机后, 可以首先通过内核 log 进行确认, 如果是 sdio wifi 内 log 会有如下 sdio 识别成功 log:

```
mmc2: new ultra high speed SDR104 SDIO card at address 0001
```

如果是 usb wifi 会有类似如下 usb 信息:

```
usb 2-1: new high-speed USB device number 2 using ehci-platform
```

```
usb 2-1: New USB device found, idVendor=0bda, idProduct=b82c
```

```
usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
```

```
usb 2-1: Product: 802.11ac NIC
```

如果 usb 信息和 sdio 扫卡成功 log 信息都没有, 那说明 wifi 模块没有正常上电或者 sdio 扫卡异常, 需要再次确认硬件是否有问题以及软件 dts 里面 wifi 管脚是否正确配置。因为开机对模块成功上电是检测 wifi 芯片 id 的前提, 在本文 v2.1 章节中已经提到。

另外也可以 cat 如下路径下的 uevent 文件进行确认:

```
sys/bus/sdio/devices
```

```
sys/bus/usb/devices
```

以 ap6356s 为例, 打开 wifi 正常识别 wifi id 的信息如下

```
01-01 00:05:20.079 5037 5037 D WifiHW : uevent path:/sys/bus/sdio/devices/. /uevent
01-01 00:05:20.079 5037 5037 D WifiHW : uevent path:/sys/bus/sdio/devices/. /uevent
01-01 00:05:20.079 5037 5037 D WifiHW : uevent path:/sys/bus/sdio/devices/mmc2:0001:1/uevent
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : pid:vid : 02d0:4356
01-01 00:05:20.079 5037 5037 D WifiHW : SDIO detectd return ret:0
01-01 00:05:20.079 5037 5037 D WifiHW : sdio WIFI identify sucess
01-01 00:05:20.079 5037 5037 D WifiHW : check_wifi_chip_type_string: AP6356S
```

2. 如果模块正常上电，也可以识别到 wifi 模块的 vid pid，再确认是否正确加载 wifi ko 驱动文件

以 AP6356s wifi 模块为例，如下是正常加载 ko 的 log 信息：

```
01-01 00:00:08.991 389 509 D WifiHW : SDIO detectd return ret:0
01-01 00:00:08.991 389 509 D WifiHW : sdio WIFI identify sucess
01-01 00:00:08.991 389 509 D WifiHW : check_wifi_chip_type_string: AP6356S
01-01 00:00:08.991 389 509 D WifiHW : wifi_load_driver matched ko file path /system/lib/modules/bcmdhd.ko
```

Wifi 模块与相应 ko 相应的对应关系：

模块名称	ko 名称
AP6xxx(SDIO)	bcmdhd.ko
RTL8822BS(SDIO)	8822bs.ko
RTL8188EU(USB)	8188eu.ko
RTL8723BS(SDIO)	8723bs.ko
RTL8189ES(SDIO)	8189es.ko
RTL8822BU(USB)	8822bu.ko
RTL8822BE(PCI-E)	8822be.ko

详细的 ko 和模块对应关系详见 `android/hardware/libhardware_legacy/wifi/wifi.c` 文件，

**RTL8822BE PCIE 模块目前只在 rk3399 平台上才支持**

3. 如果 wifi ko 匹配错误或者没有相应的 ko 文件，请确认按照 v1.1 和 v1.2 章节进行确认，正常

系统 ko 文件存放路径如下：

```
/system/bin/sh: ll: not found
127|rk3328_box:/system/lib/modules # ls -l
total 36392
-rw-r--r-- 1 root root 1505088 2017-01-23 07:36 8188eu.ko
-rw-r--r-- 1 root root 2158864 2017-01-23 07:36 8723bs.ko
-rw-r--r-- 1 root root 2589024 2017-01-23 07:36 8822bs.ko
-rw-r--r-- 1 root root 10471968 2017-01-23 07:36 bcmdhd.ko
```

**4. 注意：由于目前 wifi 驱动是采用 ko 方式，如果有修改内核网络相关配置，一定要重新编译 ko，否则很可能加载 wifi ko 引起内核 panic，请详细阅读本文第 5 章节**

5. 如果客户不需要 wifi 自动兼容方案，希望将 wifi 驱动 build in 到内核，可以按照如下说明进行  
注意 Realtek 系列 wifi 驱动只能选择一款型号编译到内核，以 RK3328 平台为例，defconfig 如下配置，SDK 默认配置为 RTL8188EU

```
CONFIG_WIFI_LOAD_DRIVER_WHEN_KERNEL_BOOTUP=y
CONFIG_RTL_WIRELESS_SOLUTION=y
```

CONFIG\_RTL8188EU=y

CONFIG\_RKWIFI=y

Device Drivers --->

[\*] Network device support --->

[\*] Wireless LAN --->

```
[*] wifi load driver when kernel bootup
[*] Realtek Wireless Device Driver Support --->
[*] Espressif 8089 sdio Wi-Fi
<*> RK901/RK903/BCM4330/AP6XXX wireless cards support
iComm WLAN support --->
Select the wifi module (AP6335) --->
Select the wifi module crystal freq (37_4M) --->
```

```
--- Realtek Wireless Device Driver Support
<*> Realtek 8188E USB WiFi
< > Realtek 8189ES/ETV SDIO WiFi Support
< > Realtek 8192CU USB WiFi Support
< > Realtek 8192DU USB WiFi Support
< > Realtek 8723AU USB WiFi Support
< > Realtek 8723BU USB WiFi
< > Realtek 8723B SDIO or SPI WiFi
< > Realtek 8723BS_VQ0 WiFi
< > Realtek 8723C SDIO or SPI WiFi
< > Realtek 8723D SDIO or SPI WiFi
< > Realtek 8812AU USB WiFi Support
< > Realtek 8189F SDIO WiFi
< > Realtek 8188F USB WiFi
< > Realtek 8822BS SDIO WiFi
< > Realtek 8822B USB WiFi
```