Rockchip Linux Wi-Fi/BT 开发

文件标识：RK-KF-YF-381

发布版本：V6.2.0

日期：2022-06-05

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

Rockchip Linux平台 Wi-Fi/BT 开发移植调试

产品版本

芯片名称	内核版本
RK356X/3399/3326/3288/3308/RV1109/RV1126/PX30	Linux 4.4/4.19
RK3588/356X/3399/RV1106/RV1103	Linux 5.10

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

硬件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V6.0.1	xy	2020-08	增加新模组移植说明、编译说明、RF测试示例、P2P桥接功能、更详细的排查说明等；
V6.0.2	xy	2021-09	增加更详细的配置及排除说明；
V6.0.3	xy	2021-11	增加USB Wi-Fi/BT配置说明；
V6.1.0	xy	2021-12	增加Debian/麒麟/UOS等系统适配说明，增加BT 5.0功能说明，增加CY芯片的低功耗模式；
V6.1.1	xy	2022-01	增加Wi-Fi SDIO/USB/PCIE接口识别流程、SDIO Timing简述；增加网络性能类问题排查说明；增加Buildroot系统开源包的更新方法；
V6.2.0	xy	2022-06	支持RV1106/1103 IPC SDK的配置说明；细化PCIE WiFi的配置说明；优化WIFIBT错误排查说明；增加第三方WiFi移植说明；增加低功耗WiFi的说明及开发指导；增加WiFi/网络裁减相关的说明；优化USBWiFi的配置说明；区分正基/CYPRESS RF测试说明；增加解决IOS设备连接设备AP热点异常问题；增加RKWIFIBT-APP使用说明；

目录

1. 快速指南

- 对于WiFi/BT不能识别，请务必先仔细检查异常**log**，对照第7章节依次排查是哪一类问题！
- Wi-Fi/BT异常参考2/3/7章节；
- Wi-Fi/BT文件/编译/更新说明参考3章节；
- Wi-Fi 驱动 KO加载时机及相关说明参考3.4章节；
- Wi-Fi/BT驱动更新 或 新模块的驱动移植参考第8章节；
- Wi-Fi/BT应用开发参考第12章节；
- RV1126/1109 Wi-Fi 参考7.2.4章节，**RV1106/1103** Wi-Fi参考13章节；
- USB接口Wi-Fi/BT参考8.1章节进行移植，然后参考2/3章节进行配置，参考7章节排查问题；
- PCIE Wi-Fi 参考2.2.5章节；
- Wi-Fi性能问题参考5/7章节；
- Debian/UOS/麒麟系统Wi-Fi/BT适配说明参考第9章节；
- 2.2章节的DTS、内核配置跟Buildroot、Debian、IPC SDK等具体的系统无关，都是通用的。
- Wi-Fi 无法连接路由器或断线或连接不稳定或连接时间慢，参考7.2.7章节。

2. Wi-Fi/BT配置

2.1 Buildroot SDK编译配置指南

配置Wi-Fi蓝牙之前，首先要配置板级文件，否则会导致Wi-Fi/BT配置不生效或其它问题；配置文件位于device/rockchip/rkxx(芯片平台)/目录，下面以**RV1126**为例**(**其它平台都是相同的规则**)**：

```
#选择板级配置，比如选择BoardConfig.mk(此文件需要根据实际情况进行选择，可以参考SDK的Quick
Start文档)
#另外比较旧的SDK不支持这条命令，则不需执行这步。
./build.sh device/rockchip/rv1126_rv1109/BoardConfig.mk

#查看BoardConfig.mk文件的内容包括（重要）：
#Kernel defconfig 对应内核使用的defconfig文件，对应到
kernel/arch/arm/configs/rv1126_defconfig
export RK_KERNEL_DEFCONFIG=rv1126_defconfig
#Kernel dts  对应内核使用的DTS
export RK_KERNEL_DTS=rv1126-evb-ddr3-v13
#Buildroot的defconfig，对应到这个文件
buildroot/configs/rockchip_rv1126_rv1109_defconfig
export RK_CFG_BUILDROOT=rockchip_rv1126_rv1109

#选择Buildroot的defconfig，就是上面的RK_CFG_BUILDROOT配置，执行这步之后才可以单独编译
rkwifibt等模块，比如 make rkwifibt/deviceio_release等指令
source envsetup.sh rockchip_rv1126_rv1109

#Buildroot配置
make menuconfig
#选择相应的配置并保存到rootfs配置文件
#./buildroot/configs/rockchip_rv1126_rv1109_defconfig
make savedefconfig
```

```
#内核配置，这里以32位为例，注意arm64目录是不一样的
cd kernel
make ARCH=arm rv1126_defconfig #就是上面的RK_KERNEL_DEFCONFIG
make ARCH=arm menuconfig
make ARCH=arm savedefconfig
cp defconfig arch/arm/configs/rv1126_defconfig #更新内核配置
```

2.2 DTS 配置

2.2.1 SDIO Wi-Fi 配置

Wi-Fi/BT硬件管脚的配置主要有以下几点：

切记一定要对照原理图进行配置，且确保使用的dts/dtsi里面包含以下节点！

SDIO 接口Wi-Fi：WL_REG_ON由**sdio_pwrseq**节点进行管理控制，不需要在wireless-wlan节点下面重复添加WIFI,poweren_gpio配置;

```
/* SDIO接口Wi-Fi专用配置： WIFI_REG_ON: Wi-Fi的电源使能PIN脚 */
sdio_pwrseq: sdio-pwrseq {
    compatible = "mmc-pwrseq-simple";
    pinctrl-names = "default";
    pinctrl-0 = <&wifi_enable_h>;
    /* 特别注意：WIFI_REG_ON GPIO_ACTIVE 配置跟使能状态恰好是相反的，
     * 高有效为LOW，低有效则为HIGH
     * 切记：这个配置跟下面的WIFI,poweren_gpio是互斥的，不能同时配置！！！
     */
    reset-gpios = <&gpio0 RK_PA2 GPIO_ACTIVE_LOW>;
};

/* SDIO接口Wi-Fi专用配置：WIFI_REG_ON脚的pinctrl的配置 */
&pinctrl {
    sdio-pwrseq {
        wifi_enable_h: wifi-enable-h {
            rockchip,pins =
            /* 对应上面的WIFI_REG_ON，关掉上下拉，防止不能拉高或拉低 */
            <0 RK_PA2 RK_FUNC_GPIO &pcfg_pull_none>;
        };
    };
};

/* SDIO接口Wi-Fi专用配置：SDIO接口节点 */
&sdio {
    max-frequency = <150000000>;    /* sdio接口的最大频率,可调整 */
    bus-width = <4>;        /* 4线模式，可调整1线模式 */
    sd-uhs-sdr104;          /* 支持SDIO3.0 */
     … …
    status = "okay";
};

/* Wi-Fi节点 */
wireless-wlan {
    compatible = "wlan-platdata";
    rockchip,grf = <&grf>;
```

```
    /* 注意：如果排查发现Wi-Fi模块没有32.768K波形，且硬件上是有
     * RK PMU供给的，则打开下面的clock属性，按照实际使用的PMU型号填写，
     * 否在SDIO/Wi-Fi无法使用.
     */
    clocks = <&rk809 1>;  //如果使用RK809，只能配置一个
    clocks = <&hym8563>;  //如果使用hym8563，只能配置一个
    clock-names = "ext_clock";

    /* 按实际名字填写 */
    wifi_chip_type = "ap6255";

    /* WIFI_WAKE_HOST: Wi-Fi中断通知主控的PIN脚。
     * 特别注意：确认下这个Wi-Fi pin脚跟主控的pin的
     * 硬件连接关系，直连的话就是GPIO_ACTIVE_HIGH;
     * 如果中间加了一个反向管就要改成低电平GPIO_ACTIVE_LOW触发
     */
    WIFI,host_wake_irq = <&gpio0 RK_PA0 GPIO_ACTIVE_HIGH>;

    //SDIO Wi-Fi 无需此配置，除非有动态加载协助驱动ko的需求，参考11.8章节
    //WIFI,poweren_gpio = <&gpio0 RK_PA2 GPIO_ACTIVE_HIGH>;
    status = "okay";
};

/* WIFI_WAKE_HOST脚的pinctrl的配置 */
wireless-wlan {
    /omit-if-no-ref/
    wifi_wake_host: wifi-wake-host {
        /* 注意一般Wi-Fi的wake host pin都是高电平触发，
         * 所以默认这里要配置为下拉；如果客户的硬件设计
         * 是反向的则要改为上拉，总之要初始化为与触发电平
         * 相反的状态
         */
        rockchip,pins = <0 RK_PA0 0 &pcfg_pull_down>;
    };
};
```

2.2.2 蓝牙配置

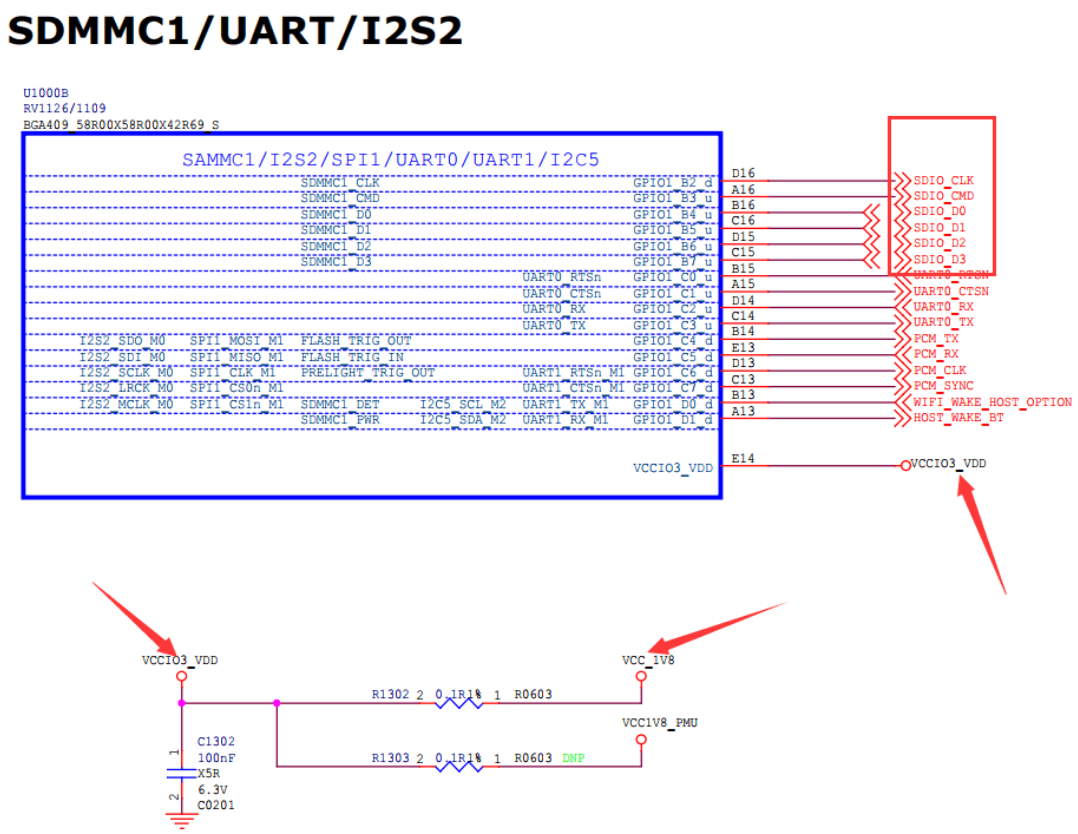
以下UART相关的都要配置为实际使用的**UART**口的所对应**PIN**，注意RTS/CTS pin一定要按照SDK设计接**(**具体接法参考**7.3**章节的**UART**描述**)**，很多客户反馈的异常都是因为这两个PIN脚没有接导致初始化异常，下面假设蓝牙使用UART4：

```
/* 蓝牙节点 */
/* 注意下面关于UART的配置：uart4_xfer/uart4_rts/uart4_ctsn
 * 每个平台的名字可能不一样，要在对应芯片平台的dts/dtsi里面找下对
 * 应的uart写法，比如uart4_ctsn有些平台的名字为uart4_cts.
 */
wireless-bluetooth {
    compatible = "bluetooth-platdata"

    /* 这里要配置对应主控UART的RTS脚 */
    uart_rts_gpios = <&gpio4 RK_PA7 GPIO_ACTIVE_LOW>;
    pinctrl-names = "default", "rts_gpio";
    pinctrl-0 = <&uart4_rts>;
    pinctrl-1 = <&uart4_rts_gpio>;
```

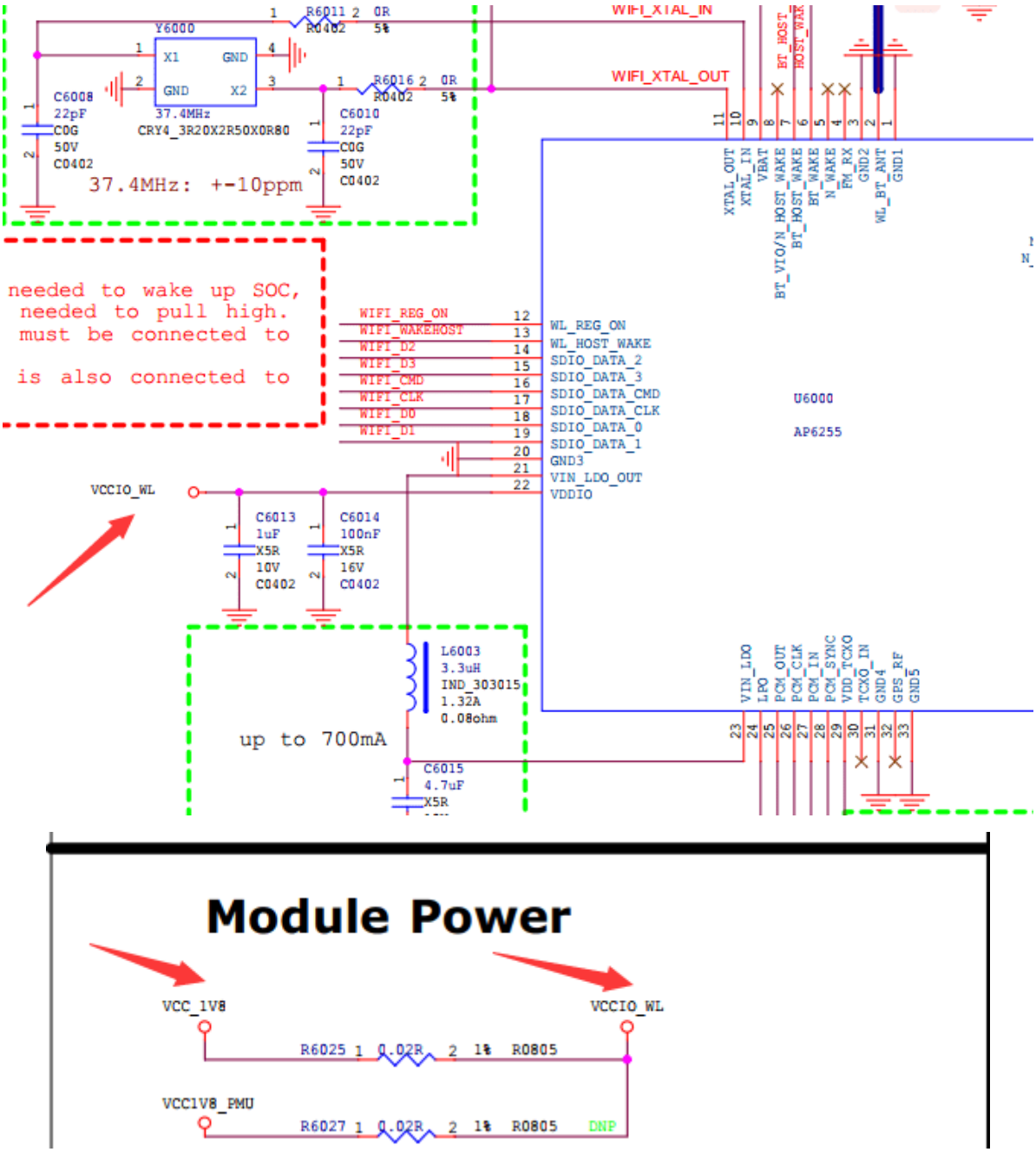
```
    /* BT_REG_ON 蓝牙电源的开关 */
    BT,power_gpio     = <&gpio4 RK_PB3 GPIO_ACTIVE_HIGH>;

    /* Linux平台：下面两个配置无需配置 */
    //BT,wake_host_irq = <&gpio4 RK_PB4 GPIO_ACTIVE_HIGH>; /* BT_WAKE_HOST */
    //BT,wake_gpio = <&gpio4 31 GPIO_ACTIVE_HIGH>; /* HOST_WAKE_BT */
    status = "okay";
};

/* 打开对应的UART配置 */
&uart4 {
    pinctrl-names = "default";
    /* 这里配置对应主控UART的TX/RX/CTS PIN ，不要配置RTS PIN*/
    pinctrl-0 = <&uart4_xfer &uart4_ctsn>;
    status = "okay";
};
```

2.2.3 IO 电源域的配置

Wi-Fi 部分共有两部分供电组成，一个是主控端的IO：SDIO_CLK/CMD/D0~D3，它需要外部供电，比如下图的VCCIO3_VDD可以看到它由VCC_1V8给它供电；



另一个是Wi-Fi模块的IO的供电，比如下图的VCCIO_WL它由也由VCC_1V8供电，两部分供电必须一致否则会导致**Wi-Fi**异常；对于支持**SDIO3.0** 超高速模式（**UHS**）的Wi-Fi模块，必须供电**1.8V**，但对于仅支持SDIO2.0高速模式的Wi-Fi模块，供电1.8/3.3V都是可以的，切记必须保持一致；



上面描述的都是硬件要求，下面对软件配置做下说明：从上图可以看到给主控IO供电的VCCIO3_VDD是1.8V，则对应dts/dtsi的io_domains需要做下配置，VDDIO3_VDD对应软件的vccio3-supply，而vccio3-supply由vcc_1v8供电则对应配置为：

```
//注意每个平台的io_domains名字不一样，有的写做&pmu_io_domains；
&io_domains {
//或
&pmu_io_domains {
    /* VDDIO3_VDD引用vcc_1v8的电压，如果硬件接的是3.3v，则改为vcc_3v3，
     * 注意：vcc_1v8/vcc_3v3名字要根据实际dts/dtsi有的进行调整。
     */
    vccio3-supply = <&vcc_1v8>;
};

vcc_1v8: vcc_1v8: vcc-1v8 {
    compatible = "regulator-fixed";
    regulator-name = "vcc_1v8";
    regulator-always-on;
    regulator-boot-on;
    /* vcc_1v8供电1.8v */
```

```
    regulator-min-microvolt = <1800000>;
    regulator-max-microvolt = <1800000>;
    vin-supply = <&vcc_io>;
};
```

2.2.4 32.768K的配置

正基/海华的模组都是要外供**32.768k**，而**realtek**的模组基本都是内部封装好的，只有**COB**芯片才会外供。

一般情况下32k都是由RK8XX型号的PMU给Wi-Fi提供，一般PMU默认打开32k，如果没有打开则需要添加如下配置：

```
wireless-wlan {
    compatible = "wlan-platdata";
    rockchip,grf = <&grf>;
    /* rk809要改为实际使用的型号，添加后如果还是没有，
     * 则需要提个pmu 输出32 clk的redmine
     */
+   clocks = <&rk809 1>;
+   clock-names = "clk_wifi";
};
```

注意：如果用的不是RK的PMU提供，则不需要这个配置；

2.2.5 PCIE Wi-Fi 配置

首先务必对照原理图阅读SDK目录如下文档：

docs\Common\PCIe\Rockchip_RK3399_Developer_Guide_PCIe_CN.pdf

docs\Common\PCIe\Rockchip_RK356X_Developer_Guide_PCIe_CN.pdf

docs\Common\PCIe\Rockchip_Developer_Guide_PCIe_CN.pdf

PCIe配置有两个关键点：

一是对照原理图找到如下关键点：

PCIe接口的WiFi基本有两种：贴在板子上模块及M2接口，但它们关键点是一样的：

VBAT/VCC3V3 ：总的3.3V电源；

WIFI_REG_ON ：芯片的复位/使能管脚；

PCIE_PERST_L/PERSTO：芯片PCIe部分的复位管脚；

二是根据原理图正确配置**Controller**及**phy**，参考上面提到的三个文档！

下面是个简单的示例：

```
//PHY的节点，不同芯片名字不一样，有的叫做&combphy2_psq，实际名称参考上面的三个文档；
&pcieXXphy {
    status = "okay";
};
```

```
//控制器节点，具体接到哪个控制器节点
&pcixxx {
    /* 此项是设置 PCIe 接口的 PERST#复位信号；不论是插槽还是
     * 焊贴的设备，请在原理图上找到该引脚，并正确配
     * 置，否则将无法完成链路建立。
     */
    ep-gpios = <&gpio3 13 GPIO_ACTIVE_HIGH>;  //RK3399平台
    reset-gpios = <&gpio4 RK_PA5 GPIO_ACTIVE_HIGH>; //RK356X/3588平台

    /* 下面三个配置是可选项，用于配置PCIe外设的1V8/3V3供电；
     * 是板级针对PCIe外设供电需要控制使能的配置项
     */
    vpcie3v3-supply = <&vdd_pcie3v3>;

    num-lanes = <4>;
    pinctrl-names = "default";
    pinctrl-0 = <&pcie_clkreqn_cpm>;  //无需修改，按照平台默认即可；
    status = "okay";
};

//Wi-Fi节点配置
wireless_wlan: wireless-wlan {
    compatible = "wlan-platdata";
    wifi_chip_type = "ap6275p";
    ... ...
    //配置：WIFI_HOST_WAKE，如果有的话
    WIFI,host_wake_irq = <&gpio0 RK_PA0 GPIO_ACTIVE_HIGH>;
    status = "okay";
};

/* VBAT/VCC3V3 ：总的3.3V电源，请注意如果这个电源是长供电的，则无需配置，
 * 如果是被GPIO控制则需添加如下配置：
 */
vcc3v3_wifi: vcc3v3-wifi {
    compatible = "regulator-fixed";
    regulator-name = "vcc3v3_wifi";
    regulator-boot-on;
    regulator-min-microvolt = <3300000>;
    regulator-max-microvolt = <3300000>;
    enable-active-high; //LEVEL高有效，如果是低有效改为LOW
    gpio = <&gpioX RK_PXX GPIO_ACTIVE_HIGH>; //修改为对应的GPIO，LEVEL同上
    startup-delay-us = <5000>;
};

//添加WIFI_REG_ON的配置
wifi_regon: wifi-regon {
    compatible = "regulator-fixed";
    regulator-name = "wifi_regon";
    regulator-boot-on;
    regulator-min-microvolt = <1800000>;
    regulator-max-microvolt = <1800000>;
    enable-active-high; //LEVEL高有效，如果是低有效改为LOW
    gpio = <&gpioX RK_PXX GPIO_ACTIVE_HIGH>; //修改为对应的GPIO，LEVEL同上
    startup-delay-us = <5000>;
};
```

2.2.6 USB Wi-Fi 配置

```
//Wi-Fi节点配置，分两种情况：
//情况一：内核kernel目录的.config文件有打开CONFIG_RFKILL/COFIG_RFKILL_RK的配置选项：
wireless_wlan: wireless-wlan {
    compatible = "wlan-platdata";
    wifi_chip_type = "ap6275p";
    //配置：WIFI_REG_ON 管脚
    WIFI,poweren_gpio = <&gpio0 RK_PA2 GPIO_ACTIVE_HIGH>;
    //配置：WIFI_HOST_WAKE，一般只有Realtek/broadcom模块需要配置
    WIFI,host_wake_irq = <&gpio0 RK_PA0 GPIO_ACTIVE_HIGH>;
    status = "okay";
};

//情况二：内核kernel目录的.config文件*没有*打开CONFIG_RFKILL/COFIG_RFKILL_RK的配置选项：
//一般针对RV1106/1103芯片
//配置：WIFI_REG_ON 管脚
wifi_usb: wifi-usb {
    compatible = "regulator-fixed";
    regulator-name = "wifi_usb";
    regulator-min-microvolt = <3300000>;
    regulator-max-microvolt = <3300000>;
    regulator-boot-on;
    enable-active-high;  //如果低有效则改为low
    gpio = <&gpio0 RK_PA2 GPIO_ACTIVE_HIGH>; //WIFI_REG_ON PIN,如果低有效则改为LOW
};

/* USB相关的参考如下文档，打开对应的控制器及PHY
 * docs\Common\USB\Rockchip_Developer_Guide_USB_CN.pdf
 * 关于USB部分RV1126/1109配置参考如下，其它芯片请根据实际情况修改，
 */
&u2phy_host {
    status = "okay";
};

&u2phy1 {
    status = "okay";
};

&usb_host0_ehci {
    status = "okay";
};

&usb_host0_ohci {
    status = "okay";
};
```

2.3 SDMMC接口接Wi-Fi芯片

有时由于特殊要求，需要把Wi-Fi芯片接到SDMMC接口，则配置做如下修改：

找到如下两个配置，把&sdio改为&sdmmc，并disabled掉没有使用的节点；

```
&sdmmc {
    bus-width = <4>;
    cap-mmc-highspeed;
    cap-sd-highspeed;
    card-detect-delay = <200>;
    rockchip,default-sample-phase = <90>;
    supports-sd;
    sd-uhs-sdr12;
    sd-uhs-sdr25;
    sd-uhs-sdr104;
    vqmmc-supply = <&vccio_sd>;
-   status = "okay";
+   status = "disabled";
};

+&sdmmc {
-&sdio {
    max-frequency = <200000000>;
    bus-width = <4>;
    cap-sd-highspeed;
    cap-sdio-irq;
    keep-power-in-suspend;
    non-removable;
    rockchip,default-sample-phase = <90>;
    sd-uhs-sdr104;
    supports-sdio;
    mmc-pwrseq = <&sdio_pwrseq>; //sdio_pwrseq只能被一个节点引用，不能sdmmc、sdio同时引用的！
    status = "okay";
};

#RK3328平台：
&sdmmc {
    bus-width = <4>;
    cap-mmc-highspeed;
    cap-sd-highspeed;
    disable-wp;
    max-frequency = <150000000>;
    pinctrl-names = "default";
    pinctrl-0 = <&sdmmc0_clk &sdmmc0_cmd &sdmmc0_dectn &sdmmc0_bus4>;
    vmmc-supply = <&vcc_sd>;
    supports-sd;
-   status = "okay";
+   status = "disabled";
};

+&sdmmc {
-&sdio  {
    bus-width = <4>;
    cap-sd-highspeed;
```

```
    cap-sdio-irq;
    keep-power-in-suspend;
    max-frequency = <150000000>;
    supports-sdio;
    mmc-pwrseq = <&sdio_pwrseq>;
    non-removable;
    pinctrl-names = "default";
    pinctrl-0 = <&sdmmc1_bus4 &sdmmc1_cmd &sdmmc1_clk>;
    status = "okay";
};
```

2.4 内核配置

```
#kernel目录
make menuconfig ARCH=armXX   #使用的defconfig参考2.1章节
```

2.4.1 Wi-Fi 配置

```
CONFIG_WL_ROCKCHIP:

Enable compatible Wifi drivers for Rockchip platform.

Symbol: WL_ROCKCHIP [=y]
Type  : boolean
Prompt: Rockchip Wireless LAN support
  Location:
    -> Device Drivers
      -> Network device support (NETDEVICES [=y])
        -> Wireless LAN (WLAN [=y])
  Defined at drivers/net/wireless/rockchip_wlan/Kconfig:2
  Depends on: NETDEVICES [=y] && WLAN [=y]
  Selects: WIRELESS_EXT [=y] && WEXT_PRIV [=y] && CFG80211 [=y] && MAC80211 [=y]
```

```
--------------------------------------------------------------------------------
            -*- Rockchip Wireless LAN support
            [ ]   build wifi ko modules
            [*]   Wifi load driver when kernel bootup
            < >   ap6xxx wireless sdio cards support
            <*>     Cypress wireless sdio cards support
            [ ]   Realtek Wireless Device Driver Support  ----
            < >   Realtek 8723B SDIO or SPI WiFi
            < >   Realtek 8723C SDIO or SPI WiFi
            < >   Realtek 8723D SDIO or SPI WiFi
            < >   Marvell 88W8977 SDIO WiFi
```

Wi-Fi驱动可编译到内核或者ko方式，切记下面两个配置必须二选一，否则**Wi-Fi**无法加载！

```
# KO 配置如下，下面两个互斥
[*]   build wifi ko modules
[ ]   Wifi load driver when kernel bootup

# buildin 配置如下，下面两个互斥
[ ]   build wifi ko modules
[*]   Wifi load driver when kernel bootup
```

- buildin 只能选择一个型号，realtek 模组和 ap6xxx 模组不能同时选择为y，且realtek的也只能选择其中一个；
- ap6xxx 和 cypress 也是互斥的，只能选择一个且如果选择ap6xxx，cypress的配置自动消失，去掉ap配置，cypress自动出现；

- ko方式则可以选择多个Wi-Fi；

配置完成后要保存到对应的**defconfig**，可以参考**2.1**章节。

2.4.2 蓝牙配置

正基和海华的模块使用内核的默认**CONFIG_BT_HCIUART** 驱动，而Realtek使用自己的hci uart驱动，源码目录为: `external\rkwifibt\realtek\bluetooth_uart_driver`，且使用ko方式加载，所以使用Realtek时一定要把内核的**CONFIG_BT_HCIUART**配置去掉！

```
CONFIG_BT_HCIUART:

Bluetooth HCI UART driver.
This driver is required if you want to use Bluetooth devices with
serial port interface. You will also need this driver if you have
UART based Bluetooth PCMCIA and CF devices like Xircom Credit Card
adapter and BrainBoxes Bluetooth PC Card.

Say Y here to compile support for Bluetooth UART devices into the
kernel or say M to compile it as module (hci_uart).

Symbol: BT_HCIUART [=y]
Type  : tristate
Prompt: HCI UART driver
  Location:
    -> Networking support (NET [=y])
      -> Bluetooth subsystem support (BT [=y])
        -> Bluetooth device drivers
  Defined at drivers/bluetooth/Kconfig:77
  Depends on: NET [=y] && BT [=y] && (SERIAL_DEV_BUS [=n] || !SERIAL_DEV_BUS [=n]) && TTY [=y]
```

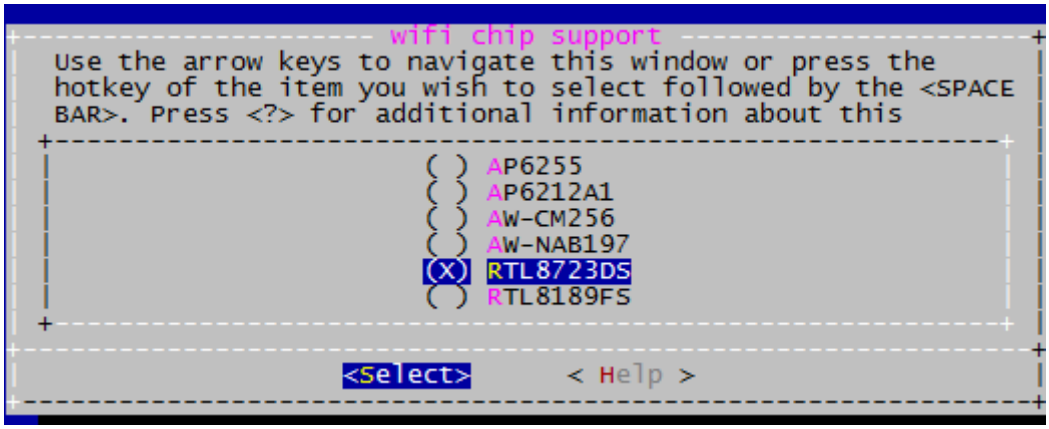
注意：配置完成后要保存到对应的**defconfig**，可以参考**2.1**章节。

2.5 Buildroot配置

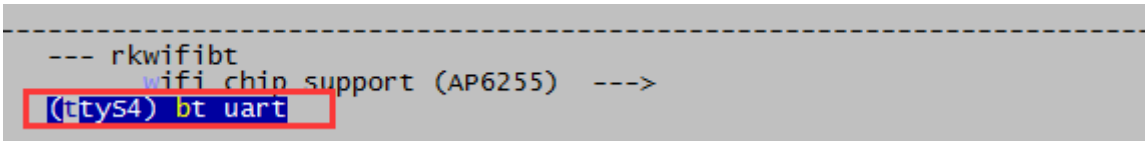
根据实际**Wi-Fi**选择对应配置，要跟内核配置的型号一致！

根目录执行：make menuconfig (编译环境参考2.1章节)，然后搜索rkwifibt进入到如下配置界面：

```
There is no help available for this option.
Prompt: wifi chip support
  Location:
    -> Target packages
      -> rockchip BSP packages (BR2_PACKAGE_ROCKCHIP [=y])
        -> rkwifibt (BR2_PACKAGE_RKWIFIBT [=y])
  Defined at package/rockchip/rkwifibt/Config.in:5
  Depends on: BR2_PACKAGE_ROCKCHIP [=y] && BR2_PACKAGE_RKWIFIBT [=y]
  Selected by: BR2_PACKAGE_ROCKCHIP [=y] && BR2_PACKAGE_RKWIFIBT [=y] && m
```



对于有带蓝牙的模组，要配置对应的ttySX，对应实际硬件接的的uart口：



注意：配置完成后要保存到对应的**defconfig**(参考**2.1**章节)，切记保存后要编译更新：参考**3.5**章节；

3. Wi-Fi/BT的文件及其编译更新说明（Buildroot系统）

3.1 编译涉及的文件

Wi-Fi驱动所对应的文件目录：

```
kernel/drivers/net/wireless/rockchip_wlan/
kernel/drivers/net/wireless/rockchip_wlan/rkwifibt/ #正基模组通用驱动
kernel/drivers/net/wireless/rockchip_wlan/cywdhd/    #cypress海华模组通用驱动
kernel/drivers/net/wireless/rockchip_wlan/rtlxxx     #realtek模组不通用，每个型号都有单独的驱动
```

Wi-Fi蓝牙所涉及的文件对应的目录为：

`external/rkwifibt/`

正基海华的Wi-Fi/BT的Firmware文件存放位置：

`external/rkwifibt/firmware/broadcom/`

正基海华**Wi-Fi**各个型号所对应的**Wi-Fi/BT**的**firmware**名字：

```
├── AP6212A1
│   ├── bt                          #蓝牙firmware
│   │   └── BCM43430A1.hcd
│   └── wifi                        #Wi-Fi firmware
│       ├── fw_bcm43438a0.bin
│       ├── fw_bcm43438a1.bin
│       ├── fw_bcm43438a1_mfg.bin
│       ├── nvram_ap6212a.txt
│       └── nvram_ap6212.txt
├── AP6236
│   ├── bt
│   │   └── BCM43430B0.hcd
│   └── wifi
│       ├── fw_bcm43436b0.bin
│       ├── fw_bcm43436b0_mfg.bin
│       └── nvram_ap6236.txt
├── AW-CM256
│   ├── bt
│   │   └── BCM4345C0.hcd
│   └── wifi
│       ├── fw_cyw43455.bin
│       └── nvram_azw256.txt
└── AW-NB197
    ├── bt
```

```
    │   └── BCM43430A1.hcd
    └── wifi
        ├── fw_cyw43438.bin
        └── nvram_azw372.txt
```

Realtek BT UART/USB驱动及**Firmware**：(注意：**Realtek Wi-Fi**不需要**firmware**文件，只有蓝牙需要)

```
external/rkwifibt/realtek/bluetooth_uart_driver/   #蓝牙uart驱动
external/rkwifibt/realtek/bluetooth_usb_driver/    #蓝牙usb驱动
external/rkwifibt/realtek/rtk_hciattach/        #蓝牙初始化程序

external/rkwifibt/realtek/RTL8723DS/
rtl8723d_config                      #8723DS的蓝牙config
rtl8723d_fw                       #8723DS的蓝牙fw
```

3.2 编译规则

对应的编译规则文件：

```
buildroot/package/rockchip/rkwifibt/Config.in     # 跟常规的Kconfig一样的规则
buildroot/package/rockchip/rkwifibt/rkwifibt.mk # 类似Makefile文件
```

由于各个芯片平台的SDK版本更新进度不一致，你们手上的in/mk文件的内容可能不同，但总的原则是一致的.

```
Config.in：将menuconfig rkwifibt配置的Wi-Fi/BT型号传递给mk文件；

rkwifibt.mk：将对应Wi-Fi/BT所需的文件、firmware等拷贝到文件系统里面去；
# 指定rkwifibt的源码目录
rkwifibt_SITE = $(TOPDIR)/../external/rkwifibt
#是构建过程函数，给源代码传递编译和链接选项，调用源代码的Makefile执行编译
RKWIFIBT_BUILD_CMDS
# 编译完之后，自动执行安装，让Buildroot把编译出来库和bin文件安装到指定的目录
RKWIFIBT_INSTALL_TARGET_CMDS
```

请仔细阅读这两个rkwifibt.mk、Config.in文件，这两个文件主的核心工作就是：

- 编译模块蓝牙驱动KO文件，比如 Realtek的uart/usb的蓝牙驱动，以及一些厂家私有可执行二进制比如正基的wl，realtek的rtwpriv 等工具;
- 根据配置Wi-Fi/BT的型号，把对应的firmware/驱动KO/可执行文件拷贝安装到指定目录，参考3.1章节的位置及对应目录;

所以开发人员必须熟悉编译规则，这对于调试非常重要**!!!**

注意：要学会看**make rkwifibt-rebuild**编译时打印**log**的输出，里面包含上面**mk**文件的编译/拷贝过程，有助于分析解决编译报错/拷贝出错等问题。

3.3 Wi-Fi/BT运行时所需的文件及其位置

开发人员要了解Wi-Fi蓝牙工作时要用的文件及位置，当遇到Wi-Fi/BT启动异常的问题时，需确认Wi-Fi蓝牙的firmware/config文件是否存在，以及是否跟蓝牙型号相匹配，参考**3.1**章节查看对应关系，如果不对应则参考**2.5**章节检查配置问题。

- 正基/海华模组以**AP6255**为例：

Wi-Fi/BT的firmware在SDK中的位置：

```
external/rkwifibt/firmware/broadcom/AP6255/
├── bt
│   └── BCM4345C0.hcd
└── wifi
    ├── fw_bcm43455c0_ag.bin
    ├── fw_bcm43455c0_ag_mfg.bin
    └── nvram_ap6255.txt
```

经过**2.2**章节的编译规则后，对应的文件被拷贝到工程的output目录：**(kernel4.19**内核由**system**变更为**vendor**目录**)**

```
buildroot/output/rockchip_rk3xxxx/target/
/system(vendor)/lib/modules/bcmdhd.ko            #驱动ko（如果是ko编译的话）
/system(vendor)/etc/firmware/fw_bcm43455c0_ag.bin    #驱动firmware文件存放位置
/system(vendor)/etc/firmware/nvram_ap6255.txt        #驱动nvram文件存放位置
/system(vendor)/etc/firmware/BCM4345C0.hcd       #蓝牙firmware文件（如果有蓝牙功能）
```

最终烧录到机器中后，Wi-Fi运行时所需的文件及存放位置：

```
/system(vendor)/lib/modules/bcmdhd.ko            #驱动ko（如果是ko编译的话）
/system(vendor)/etc/firmware/fw_bcm43455c0_ag.bin    #驱动firmware文件存放位置
/system(vendor)/etc/firmware/nvram_ap6255.txt        #驱动nvram文件存放位置
/system(vendor)/etc/firmware/BCM4345C0.hcd       #蓝牙firmware文件（如果有蓝牙功能）
```

- **Realtek**模组，以**RTL8723DS/RTL8821CU**为例：

Wi-Fi/BT的firmware在SDK中的位置：

```
external/rkwifibt/realtek$ tree
├── RTL8723DS
│   ├── mp_rtl8723d_config #蓝牙测试用的config, 需要找厂家获取
│   ├── mp_rtl8723d_fw     #蓝牙测试用的fw, 需要找厂家获取
│   ├── rtl8723d_config    #蓝牙config
│   └── rtl8723d_fw    #蓝牙fw
├── RTL8821CU
│   ├── rtl8821cu_config   #蓝牙config
│   └── rtl8821cu_fw       #蓝牙fw
├── bluetooth_uart_driver  #RTL8723DS 的蓝牙UART驱动，被编译成hci_uart.ko
│   ├── hci_h4.c
│   ├── hci_ldisc.c
│   ├── hci_rtk_h5.c
│   ├── hci_uart.h
│   ├── Kconfig
│   ├── Makefile
│   ├── rtk_coex.c
```

```
│   └── rtk_coex.h
├── bluetooth_usb_driver    #RTL8821CU的蓝牙USB驱动，被编译成rtk_btusb.ko
│   ├── Makefile
│   ├── rtk_bt.c
│   ├── rtk_bt.h
│   ├── rtk_coex.c
│   ├── rtk_coex.h
│   ├── rtk_misc.c
│   └── rtk_misc.h
├── rtk_hciattach       #蓝牙初始化程序rtk_hciattach,只有UART接口的BT才会使用，USB不需要
│   ├── fix_mac.patch
│   ├── hciattach.c
│   ├── hciattach.h
│   ├── hciattach_h4.c
│   ├── hciattach_h4.h
│   ├── hciattach_rtk.c
│   ├── Makefile
│   ├── rtb_fwc.c
│   └── rtb_fwc.h
```

经过**3.2**章节的编译规则后，对应的文件被拷贝到工程的output目录：

```
#前缀目录
buildroot/output/rockchip_rk3xxxx/target/

#  如果有蓝牙功能的话
/system(vendor)/lib/modules/8723ds.ko    #驱动ko（如果是ko编译的话）
/system(vendor)/lib/modules/8821cu.ko    #驱动ko（如果是ko编译的话）
/usr/lib/rtk_hciattach           #蓝牙初始化程序
/usr/bin/modules/hci_uart.ko         #蓝牙ko
#切记UART接口蓝牙firmware文件会拷贝到/lib/firmware/rtlbt/目录
/lib/firmware/rtlbt/rtl8723d_config #蓝牙正常功能的fw/config
/lib/firmware/rtlbt/rtl8723d_fw
#切记USB接口蓝牙firmware文件会拷贝到/lib/firmware/目录
/lib/firmware/rtl8821cu_config       #蓝牙正常功能的fw/config
/lib/firmware/rtl8821cu_fw
```

最终烧录到机器中后，Wi-Fi运行时所需的文件及存放位置：(kernel4.19内核由system变更为vendor目录)

```
/system(vendor)/lib/modules/8723ds.ko    #驱动ko（如果是ko编译的话）
/system(vendor)/lib/modules/8821cu.ko    #驱动ko（如果是ko编译的话）
/usr/bin/rtk_hciattach           #蓝牙初始化程序(仅uart接口使用)
/usr/lib/modules/hci_uart.ko         #UART蓝牙ko
/usr/lib/modules/rtk_btusb.ko        #USB蓝牙ko
#切记UART接口蓝牙firmware文件会拷贝到/lib/firmware/rtlbt/目录
/lib/firmware/rtlbt/rtl8723d_config #蓝牙正常功能的fw/config
/lib/firmware/rtlbt/rtl8723d_fw
#切记USB接口蓝牙firmware文件会拷贝到/lib/firmware/目录
/lib/firmware/rtl8821cu_config       #蓝牙正常功能的fw/config
/lib/firmware/rtl8821cu_fw
```

3.4 开机自动加载Wi-Fi驱动KO的规则

Wi-Fi加载驱动的规则如下：

```
# 原始文件位置如下，它经过./build.sh 或者make rkwifibt-rebuild编译后会被拷贝
到/etc/init.d/目录下（每次修改后需要重新编译打包）
cat external/rkwifibt/SXXload_wifi_xxx_modules
...
insmod WIFI_KO # 内容为通用的名字，在编译期间会根据rkwifibt的配置进行替换
...

# 通过menuconfig的配置及buildroot/package/rockchip/rkwifibt/Config.in & rkwifibt.mk
的编译规则.
# rkwifibt.mk 内容节选如下，WIFI_KO被替换成实际配置的ko名字
$(SED) 's/WIFI_KO/\/$(FIRMWARE_DIR)\/lib\/modules\ ... ...

# 查看编译后的文件内容
cat buildroot/output/rockchip_rk3xxxx/target/etc/init.d/SXXload_wifi_xxx_modules
... ...
#可以看到Wi-Fi KO被替换成实际配置的ko，比如RTL8723DS：
insmod 8723ds.ko
#它由make menuconfig配置时通过buildroot/package/rockchip/rkwifibt/Config.in文件传入
... ...

# 最后 etc/init.d/目录的文件在系统启动时会依次去调用它们，所以Wi-Fi的驱动在此期间被自动加载。

# 经常遇到的问题：比如使用RTL8723DS，但是有时开发人员忘记配置Buildroot里面的Wi-Fi型号导致
SXXload_wifi_xxx_modules文件里面insmod 错误的ko型号：insmod bcmdhd.ko（正常应该为
insmod 8723ds.ko），解决办法：参考2.5章节Buildroot的修改！
```

3.5 编译更新

首先参考**2.1**章节确认编译环境

- 对于kernel Wi-Fi配置的修改

`make menuconfig`选择完成后，一定要保存对应的defconfig文件，比如我使用的是：`kernel/arch/arm/configs/rockchip_xxxx_defconfig`

要把对应的修改更新到这个文件，否则会出现更新不生效的问题

```
# 内核目录
make menuconfig
# 修改相应的配置
make savedefconfig
cp defconfig arch/arm/configs/rockchip_xxxx_defconfig
```

- 对于Buildroot配置的修改，make menuconfig选择完成后，在根目录执行make savedeconfig进行保存。

```
# 顶层目录
source xxxx                #首先选择对应工程的配置，这部分参考SDK的开发配置文档
make menuconfig
# 选择对应的Wi-Fi型号
make savedeconfig          #保存配置
make rkwifibt-dirclean         #清除掉之前的
make rkwifibt-rebuild          #重新编译
./build.sh             #重新打包生成固件
```

注意：一定要`make savedefconfig`，不然编译时会被原来覆盖掉，导致修改未生效。

4. Wi-Fi/BT功能验证

4.1 Wi-Fi STA测试

4.1.1 打开关闭Wi-Fi

如果没有出现wlan0节点，请先检查dts/驱动配置部分是否配置正确，驱动是否已加载(ko还是buildin)，如果正确则请参考第7章节进行排查。

打开**Wi-Fi**

```
echo 1 > /sys/class/rfkill/rfkill1/state  # 当蓝牙节点没有打开时是rfkill0
ifconfig wlan0 up
wlan0     Link encap:Ethernet  HWaddr F0:85:C1:0F:9C:02
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

然后查看**Wi-Fi**的服务进程启动： **ps**检查是否有 **wpa_supplicant**进程, 如果没启动可以手动开启：

注意下面的 **-c** 选项，它指定配置文件的路径！不要配错！

`wpa_supplicant -B -i wlan0 -c /data/cfg/wpa_supplicant.conf`

配置文件解析：

```
#ctrl_interface接口配置
#如果有修改的话对应wpa_cli命令-p参数要相应进行修改，wpa_cli -i wlan0 -p <ctrl_interface>
xxx
$ vi /data/cfg/wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant  #默认不建议修改!
ap_scan=1
update_config=1 #这个配置使wpa_cli命令配置的热点保存到conf文件里面（wpa_cli
save_config）

#AP配置项
network={
    ssid="WiFi-AP"       # Wi-Fi名字
    psk="12345678"       # Wi-Fi密码
    key_mgmt=WPA-PSK     # 加密配置，不加密则改为：key_mgmt=NONE
}
```

注意：`wpa_supplicant.conf`文件请根据实际平台的存放位置进行修改。

关闭**Wi-Fi**

`ifconfig wlan0 down`

`killall wpa_supplicant`

4.1.2 扫描周边的AP

wpa_cli 指令跟wpa_supplicant进程通信，所以确保wpa_supplicant进程有运行起来。

`wpa_cli -i wlan0 -p /var/run/wpa_supplicant scan`

`wpa_cli -i wlan0 -p /var/run/wpa_supplicant scan_results`

```
/ #
/ # wpa_cli -i wlan0 -p /var/run/wpa_supplicant scan_results
bssid / frequency / signal level / flags / ssid
dc:ef:09:a7:77:53       2437    -30     [WPA2-PSK-CCMP][ESS]    fish1
10:be:f5:1d:a3:74       2447    -34     [WPA-PSK-CCMP+TKIP][WPA2-PSK-CCMP+TKIP][ESS]    DLink8808
d4:ee:07:5b:81:80       2432    -35     [WPA-PSK-CCMP][WPA2-PSK-CCMP][ESS]      Fang-HiWiFi
76:7d:24:51:39:d0       2422    -35     [WPA-PSK-CCMP+TKIP][WPA2-PSK-CCMP+TKIP][ESS]    @PHICOMM_CE
2c:b2:1a:3a:7f:d6       2412    -42     [WPA-PSK-CCMP+TKIP][WPA2-PSK-CCMP+TKIP][ESS]    RK_0101
74:05:a5:29:5f:cc       2412    -42     [WPA-PSK-CCMP][WPA2-PSK-CCMP][ESS]      TP-LINK_5FJK
24:69:68:98:aa:42       2437    -42     [WPA-PSK-CCMP][WPA2-PSK-CCMP][ESS]      ZainAP
d4:ee:07:1c:2d:18       2427    -43     [WPA-PSK-CCMP][WPA2-PSK-CCMP][ESS]      ROCKROOM
9c:21:6a:c8:6f:7c       2462    -43     [WPA-PSK-CCMP][WPA2-PSK-CCMP][ESS]      TP-LINK_HKH
```

注意：要看扫描到的热点的个数是否匹配你周围的路由器大概个数，跟手机扫描的**Wi-Fi**对比（如果客户的模组不支持**5G**，只对比**2.4G**的个数）；还有检查离测试者最近的路由器的的信号强度，如果路由器距离测试者很近，但信号强度却非常弱（正常情况下：**-20**到**-65**；偏弱：**-65**到**-70**；差**-70**到**-90**），这时就要检查的**Wi-Fi**模组是否有接天线，模组的**RF**指标是否合格等等（参考第**5**章节**Wi-Fi/BT**的硬件测试）。

4.1.3 连接路由器

方法一：通过修改配置文件

```
#在wpa_supplicant.conf里面添加network配置项
network={
    ssid="WiFi-AP"       # Wi-Fi名字
    psk="12345678"       # Wi-Fi密码
    key_mgmt=WPA-PSK     # 加密配置，不加密则改为：key_mgmt=NONE
}

#让wpa_supplicant进程重新读取上述配置，命令如下：
wpa_cli -i wlan0 -p /var/run/wpa_supplicant reconfigure
#发起连接：
wpa_cli -i wlan0 -p /var/run/wpa_supplicant reconnect
```

方法二：简单的脚本

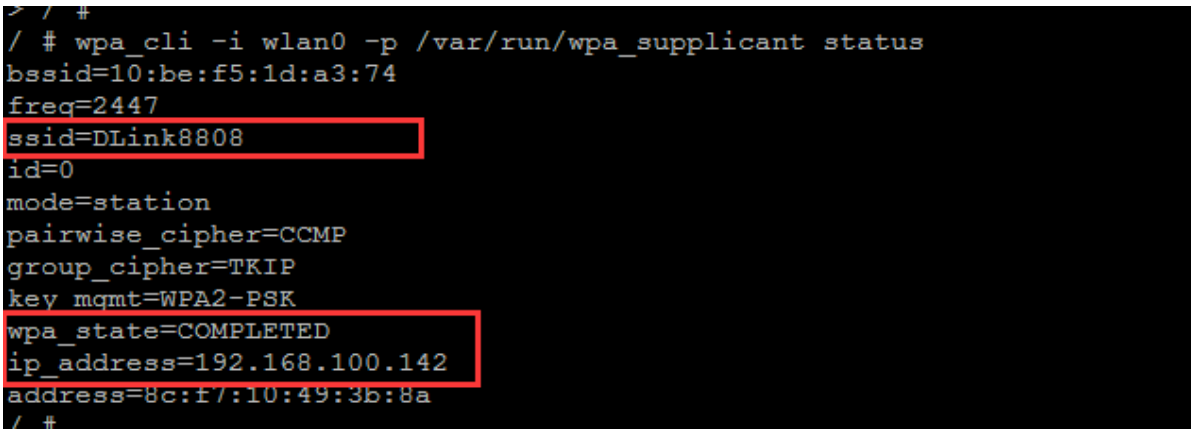
最新sdk集成wifi_start.sh脚本，如果有这个脚本，则可以直接在脚本后面跟 ssid和password

```
wifi_start.sh fanxing 12345678
```

方法三：**wpa_cli**工具

```
#加密：
wpa_cli -i wlan0 -p /var/run/wpa_supplicant remove_network 0
wpa_cli -i wlan0 -p /var/run/wpa_supplicant ap_scan 1
wpa_cli -i wlan0 -p /var/run/wpa_supplicant add_network
wpa_cli -i wlan0 -p /var/run/wpa_supplicant set_network 0 ssid "dlink"
wpa_cli -i wlan0 -p /var/run/wpa_supplicant set_network 0 key_mgmt WPA-PSK
wpa_cli -i wlan0 -p /var/run/wpa_supplicant set_network 0 psk '"12345678"'
wpa_cli -i wlan0 -p /var/run/wpa_supplicant select_network 0
wpa_cli -i wlan0 -p /var/run/wpa_supplicant save_config # 保存上述配置到conf文件

#不加密：
wpa_cli -i wlan0 -p /var/run/wpa_supplicant remove_network 0
wpa_cli -i wlan0 -p /var/run/wpa_supplicant ap_scan 1
wpa_cli -i wlan0 -p /var/run/wpa_supplicant add_network
wpa_cli -i wlan0 -p /var/run/wpa_supplicant set_network 0 ssid "dlink"
wpa_cli -i wlan0 -p /var/run/wpa_supplicant set_network 0 key_mgmt NONE
wpa_cli -i wlan0 -p /var/run/wpa_supplicant select_network 0
wpa_cli -i wlan0 -p /var/run/wpa_supplicant save_config
```

连接成功：



如果出现`wpa_state=COMPLETED`但没有出现有效的ip_address, 则检查**dhcpcd**获取ip地址的进程是否有启动；如果wpa_state不为COMPLETED，则先排查**2.1.1**扫描章节。

4.2 Wi-Fi AP热点验证

SDK集成了相关程序，执行：softapDemo apName （开启名为apName默认无加密的热点）即可开启热点模式.
代码及编译文件位置：

```
/external/softapDemo/src/main.c
buildroot/package/rockchip/softap/Config.in softap.mk
make softap-dirclean
make softap
```

如果没有找到softapDemo源码，则从下面地址下载到external目录：

```
https://github.com/rockchip-linux/softapServer
```

RTL模组：使用p2p0作为softap功能，通过内核驱动的配置生成p2p0，如果没有p2p0节点请检查这里的配置

```
+++ b/drivers/net/wireless/rockchip_wlan/rtl8xxx/Makefile
@@ -1593,7 +1593,7 @@ endif
ifeq ($(CONFIG_PLATFORM_ARM_RK3188), y)
EXTRA_CFLAGS += -DCONFIG_PLATFORM_ANDROID
+EXTRA_CFLAGS += -DCONFIG_CONCURRENT_MODE
```

AP/海华模组： 使用wlan1作为softap功能，且使用iw命令去生成wlan1节点

`iw phy0 interface add wlan1 type managed`

调试及定制化修改：

```
//这里可以添加加密、修改IP地址及dns等相关信息，可自行修改
int wlan_accesspoint_start(const char* ssid, const char* password)
{
    //创建softap热点配置
    create_hostapd_file(ssid, password);

    //softap_name: wlan1/p2p0
    sprintf(cmdline, "ifconfig %s up", softap_name);
    //设置自定义IP地址
    sprintf(cmdline, "ifconfig %s 192.168.88.1 netmask 255.255.255.0",
softap_name);

    //创建dhcp 地址池的配置文件
    creat_dnsmasq_file();
    int dnsmasq_pid = get_dnsmasq_pid();
    if (dnsmasq_pid != 0) {
        memset(cmdline, 0, sizeof(cmdline));
        sprintf(cmdline, "kill %d", dnsmasq_pid);
        console_run(cmdline);
    }
    memset(cmdline, 0, sizeof(cmdline));
    //使用dnsmasq做dhcp服务器，给设备分配ip地址
    sprintf(cmdline, "dnsmasq -C %s --interface=%s", DNSMASQ_CONF_DIR,
softap_name);
```

```
    console_run(cmdline);

    memset(cmdline, 0, sizeof(cmdline));
    //启动softap热点模式
    sprintf(cmdline, "hostapd %s &", HOSTAPD_CONF_DIR);
    console_run(cmdline);
    return 1;
}

//创建dhcp配置文件，要与你自定义的IP地址一致，否则会出现手机无法获取IP
bool creat_dnsmasq_file()
{
    FILE* fp;
    fp = fopen(DNSMASQ_CONF_DIR, "wt+");
    if (fp != 0) {
        fputs("user=root\n", fp);
        fputs("listen-address=", fp);
        fputs(SOFTAP_INTERFACE_STATIC_IP, fp);
        fputs("\n", fp);
        fputs("dhcp-range=192.168.88.50,192.168.88.150\n", fp);
        fputs("server=/google/8.8.8.8\n", fp);
        fclose(fp);
        return true;
    }
    DEBUG_ERR("---open dnsmasq configuarion file failed!!---");
    return true;
}

//创建AP热点配置文件，具体可咨询Wi-Fi厂家，这里的参数跟芯片规格有很大关系
int create_hostapd_file(const char* name, const char* password)
{
    FILE* fp;
    char cmdline[256] = {0};

    fp = fopen(HOSTAPD_CONF_DIR, "wt+");

    if (fp != 0) {
        sprintf(cmdline, "interface=%s\n", softap_name);
        fputs(cmdline, fp);
        fputs("ctrl_interface=/var/run/hostapd\n", fp);
        fputs("driver=nl80211\n", fp);
        fputs("ssid=", fp);
        fputs(name, fp);
        fputs("\n", fp);
        fputs("channel=6\n", fp);  // 信道设置
        fputs("hw_mode=g\n", fp);  // 2.4/5G配置
        fputs("ieee80211n=1\n", fp);
        fputs("ignore_broadcast_ssid=0\n", fp);
#if 0  //如果选择加密，则修改这里
        fputs("auth_algs=1\n", fp);
        fputs("wpa=3\n", fp);
        fputs("wpa_passphrase=", fp);
        fputs(password, fp);
        fputs("\n", fp);
        fputs("wpa_key_mgmt=WPA-PSK\n", fp);
        fputs("wpa_pairwise=TKIP\n", fp);
        fputs("rsn_pairwise=CCMP", fp);
#endif
```

```
        fclose(fp);
        return 0;
    }
    return -1;
}

int main(int argc, char **argv)
    //根据Wi-Fi型号设置对应的热点接口，4.4内核通过"/sys/class/rkwifi/chip"节点自动获取Wi-
Fi型号，
    //但4.19及其之后的内核版本无此节点，可通过下面方式确定Wi-Fi型号
    //正基/海华： sys/bus/sdio/drivers/bcmsdh_sdmmc 判断有无此目录
    //realtek: sys/bus/sdio/drivers/rtl8723ds 判断有无此目录
    if (!strncmp(wifi_type, "RTL", 3))
        strcpy(softap_name, "p2p0"); //realtek使用p2p0
    else
        strcpy(softap_name, "wlan1"); //正基海华使用wlan1

    ... ...
    if (!strncmp(wifi_type, "RTL", 3)) {
        //Realtek模组打开共存模式后自动生成p2p0节点
        console_run("ifconfig p2p0 down");
        console_run("rm -rf /userdata/bin/p2p0");
        wlan_accesspoint_start(apName, NULL);
    } else {
        console_run("ifconfig wlan1 down");
        console_run("rm -rf /userdata/bin/wlan1");
        console_run("iw dev wlan1 del");
        console_run("ifconfig wlan0 up");
        //AP模组需要iw 命令生成wlan1节点做softap使用
        console_run("iw phy0 interface add wlan1 type managed");
        wlan_accesspoint_start(apName, NULL);
    }
```

执行完命令后可以在手机的**setting Wi-Fi**界面下看到对应的**AP**，如果没有请排查：

- 首先确保上面提到的几个配置文件是否有配置正确；
- 确认ifconfig是否有看到wlan1或p2p0节点；
- hostapd/dnsmasq进程是否有启动成功；

4.3 BT 验证测试

首先确保dts/Buildroot要配置正确，参考第2/3/7.2章节，这里对常用的两类BT模组进行说明，在配置正确情况下，系统会生成一个**bt_pcba_test**(或者最新SDK会生成一个**bt_init.sh**)的脚本程序

REALTEK模组

```
#UART 接口：
/ # cat usr/bin/bt_pcba_test (bt_init.sh)
#!/bin/sh

killall rtk_hciattach

echo 0 > /sys/class/rfkill/rfkill0/state #下电
sleep 1
echo 1 > /sys/class/rfkill/rfkill0/state #上电
```

```
sleep 1
insmod /usr/lib/modules/hci_uart.ko          # realtek模组需要加载uart驱动
rtk_hciattach -n -s 115200 /dev/ttyS4 rtk_h5 &  # 蓝色指的是蓝牙使用哪个uart口

#注意：每次启动测试时要先kill掉rtk_hciattach进程

#注意：USB接口蓝牙没有sh脚本，手动执行如下：
echo 0 > /sys/class/rfkill/rfkill0/state #下电
sleep 1
echo 1 > /sys/class/rfkill/rfkill0/state #上电
sleep 1
insmod /usr/lib/modules/rtk_btusb.ko     #realtek模组需要加载usb驱动
```

正基/海华模组

```
/ # cat usr/bin/bt_pcba_test (bt_init.sh)
#!/bin/sh

killall brcm_patchram_plus1

echo 0 > /sys/class/rfkill/rfkill0/state  # 下电
sleep 2
echo 1 > /sys/class/rfkill/rfkill0/state  # 上电
sleep 2

brcm_patchram_plus1 --bd_addr_rand --enable_hci --no2bytes --
use_baudrate_for_download  --tosleep  200000 --baudrate 1500000 --patchram
 /system/etc/firmware/bcm43438a1.hcd /dev/ttyS4 &

#注意：每次启动测试时要先kill掉brcm_patchram_plus1进程
```

bcm43438a1.hcd表示**BT** 对应型号**Firmware**文件，**/dev/ttyS4**是蓝牙使用哪个**UART**口。

注意：`rtk_hciattach 、hci_uart.ko、bcm43438a1.hcd`等文件都是第2章节Buildroot配置选择正确的Wi-Fi/BT模组的前提下才会生成，如果没有这些文件请检查上述配置(参考第3章节)。

执行该脚本后，执行：

注意：如果没有**hciconfig**命令，请在**Buildroot**配置选择**BR2_PACKAGE_BLUEZ5_UTILS** 编译并更新测试

`hciconfig hci0 up`
`hciconfig -a`

正常的情况下可以看到：

```
/ # hciconfig -a
hci0:   Type: Primary  Bus: UART
        BD Address: 2A:CB:74:E5:DF:92  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING
        RX bytes:1224 acl:0 sco:0 events:60 errors:0
        TX bytes:796 acl:0 sco:0 commands:60 errors:0
        Features: 0xbf 0xfe 0xcf 0xfe 0xdb 0xff 0x7b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH SNIFF
        Link mode: SLAVE ACCEPT
        Name: 'BCM43438A1 26MHz AP6212A1_CL1 BT4.0 OTP-BD-0058'
        Class: 0x000000
        Service Classes: Unspecified
        Device Class: Miscellaneous,
        HCI Version: 4.0 (0x6)  Revision: 0xf9
        LMP Version: 4.0 (0x6)  Subversion: 0x2209
        Manufacturer: Broadcom Corporation (15)
```

BT扫描：`hcitool scan`

```
/ # hcitool scan
Scanning ...
        D0:C5:D3:92:D9:04               -A11-0308
        2C:57:31:50:B3:09       E2
        EC:D0:9F:B4:55:06       xing_mi6
        5C:07:7A:CC:22:22               _AUDIO
        18:F0:E4:E7:17:E2       小米手机123
        AC:C1:EE:18:4C:D3       红米手机
        B4:0B:44:E2:F7:0F       n/a
```

异常情况请参考**7.2**章节进行排查；

4.4 Wi-Fi 休眠唤醒

目前Wi-Fi支持网络唤醒功能，例如：设备连接AP并获取IP地址，则当设备休眠后我们可以通过无线网络包（ping）唤醒系统，一般规则为：只要是发给本设备网络包都可唤醒系统。

修改wpa_supplicant.conf文件，添加如下配置：

```
wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
update_config=1
ap_scan=1
+wowlan_triggers=any # 添加这个配置
```

Realtek Wi-Fi请检查对应驱动的Makefile里面是否有如下配置：

```
kernel/drivers/net/wireless/rockchip_wlan/rtl8xxx/Makefile
+CONFIG_WOWLAN = y
+CONFIG_GPIO_WAKEUP = y
```

DTS 配置：查看原理图确保 WIFI_WAKE_HOST (或 WL_HOST_WAKE) PIN脚有接到主控，然后检查dts的如下配置是否正确：

```
WIFI,host_wake_irq = <&gpio0 RK_PA0 GPIO_ACTIVE_HIGH>;
```

测试系统及Wi-Fi休眠：

```
dhd_priv setsuspendmode 1   # 仅针对正基海华模组，Realtek无需此命令
echo mem > /sys/power/state
```

此时同一局域网内的设备可以ping此设备，正常情况下可以观察到系统被唤醒，注意系统唤醒后需要恢复Wi-Fi正常的工作状态：

```
dhd_priv setsuspendmode 0 # 仅针对正基海华模组，Realtek无需此命令
```

排查：如果系统没有预期内唤醒，则请排查wake pin脚是否配置正确及电平状态是否正确，32.768k是否被关掉等等.

4.5 Wi-Fi MONITOR模式

正基/海华**Wi-Fi**

```
#设置监听信道：
dhd_priv channel 6  //channal numbers

#打开 monitor模式：
dhd_priv monitor 1

#关闭 monitor模式：
dhd_priv monitor 0
```

Realtek Wi-Fi

```
#驱动Makefile需要打开：
+ CONFIG_WIFI_MONITOR = y

#打开wlan0并关闭p2p0
ifconfig wlan0 up
ifconfig p2p0 down

#打开监听模式
iwconfig wlan0 mode monitor
or
iw dev wlan0 set type monitor

#切换信道
echo "<chan> 0 0" > /proc/net/<rtk_module>/wlan0/monitor // <rtk_module> is the
realtek Wi-Fi module name, such like rtl8812au, rtl8188eu ..etc
```

4.6 Wi-Fi P2P验证

```
#新建配置文件：p2p_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
update_config=1
device_name=p2p_name
device_type=10-0050F204-5
config_methods=display push_button keypad virtual_push_button physical_display
```

```
p2p_add_cli_chan=1
pmf=1

#启动：（先kill掉之前的）
wpa_supplicant -B -i wlan0 -c /tmp/p2p_supplicant.conf
wpa_cli
> p2p_find
>

#此时手机端打开p2p,可以搜索到上面的device_name=p2p_name，点击连接

#此时设备端显示 //下面是测试手机
> <3>P2P-PROV-DISC-PBC-REQ 26:31:54:8e:14:e7 p2p_dev_addr=26:31:54:8e:14:e7
pri_dev_type=10-0050F204-5 name='www' config_methods=0x188 dev_capab=0x25
group_capab=0x0

#设备端响应并下发连接命令，注意mac地址要跟上面的一致
>p2p_connect 26:31:54:8e:14:e7 pbc go_intent=1

> p2p_connect 26:31:54:8e:14:e7 pbc go_intent=1
OK
<3>P2P-FIND-STOPPED
<3>P2P-GO-NEG-SUCCESS role=client freq=5200 ht40=0 peer_dev=26:31:54:8e:14:e7
peer_iface=26:31:54:8e:94:e7 wps_method=PBC
<3>P2P-GROUP-FORMATION-SUCCESS
<3>P2P-GROUP-STARTED p2p-wlan0-2 client ssid="DIRECT-24-www" freq=5200
psk=3d67671b71f7a171118c1ace34ae5e4bcc8e17394394e258be91f55b7ab63748
go_dev_addr=26:31:54:8e:14:e7 [PERSISTENT]
> #此时连接成功
> quit

ifconfig
p2p-wlan0-2 Link encap:Ethernet  HWaddr 82:C5:F2:2E:7F:89
          inet addr:192.168.49.220  Bcast:192.168.49.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:470 errors:0 dropped:0 overruns:0 frame:0
          TX packets:344 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:71779 (70.0 KiB)  TX bytes:33829 (33.0 KiB)
#可以看到设备连接到手机，可以互相ping通则表示正常。
```

4.7 桥接功能

场景：Wi-Fi启动wlan0连接可以上网的AP，且配合4.2章节启动wlan1或p2p0做热点，使手机连接热点上网，配置如下，内核打开如下配置：

```
+CONFIG_NETFILTER=y
+CONFIG_NF_CONNTRACK=y
+CONFIG_NF_TABLES=y
+CONFIG_NF_TABLES_INET=y
+CONFIG_NF_CONNTRACK_IPV4=y
+CONFIG_IP_NF_IPTABLES=y
+CONFIG_IP_NF_NAT=y
+CONFIG_IP_NF_TARGET_MASQUERADE=y
+CONFIG_BRIDGE=y
```

执行下面两条命令，下面IP地址为启动softap时配置的地址

```
iptables -t nat -A POSTROUTING -s 192.168.43.0/24 -o wlan0 -j MASQUERADE
echo "1" > /proc/sys/net/ipv4/ip_forward
```

5. Wi-Fi/BT硬件RF指标

5.1 测试项目

Wi-Fi/BT部分

比如：发射功率、**EVM**、晶体频偏、接收灵敏度等等

示例：(b/g/n/ac)

Test mode	802.11b RF report				
传导功率	802.11b bandwidth_20MHz (dBm)				
	调制方式 Modulate model	CH1 Antenna 0	CH7 Antenna 0	CH13 Antenna 0	满足规格/spec
	11Mbps	17.17	16.86	17.21	<20dbm
频谱模板 /Transmit spectrum mask	802.11b bandwidth_20MHz (GHz)				
	调制方式 Modulate model	CH1 Antenna 0	CH7 Antenna 0	CH13 Antenna 0	满足规格 /spec
	11Mbps	pass	pass	pass	
发射调制精度测试 /Transmit modulation accuracy	802.11b bandwidth_20MHz (dB)				
	调制方式 Modulate model	CH1 Antenna 0	CH7 Antenna 0	CH13 Antenna 0	满足规格/spec (峰值检波)
	11Mbps	6.65%	5.91%	4.26%	<35%
中心频率容限 /Transmit center frequency tolerance	802.11b bandwidth_20MHz (ppm)				
	调制方式 Modulate model	CH1 Antenna 0	CH7 Antenna 0	CH13 Antenna 0	满足规格/spec (10ppm余量)
	11Mbps	6.65	5.91	5.27	(+/-10ppm)
接收灵敏度/ Receiver minimum input level sensitivity	802.11b bandwidth_20MHz (dB)				
	调制方式 Modulate model	CH1 Antenna 0	CH7 Antenna 0	CH13 Antenna 0	满足规格 (2dB余量)
	11Mbps	-84	-83	-82	-78
最大接收电平/ Receiver maximum input level	802.11b bandwidth_20MHz (dBm)				
	调制方式 Modulate model	CH1 Antenna 0	CH7 Antenna 0	CH13 Antenna 0	满足规格/spec
	11Mbps				》 -10dBm

天线部分

无源 **S11**，有源**Wi-Fi**天线整机的**OTA**测试

示例：

80211b：11MBps					80211g：54MBps			
Test	Wi-Fi 2G TRP				Test	Wi-Fi 2G TRP		
Channel	1	7	13		Channel	1	7	13
Frequency(MHz)	2412	2442	2472		Frequency(MHz)	2412	2442	2472
Txp Ave(dBm)	11.12	14.39	13.79		Txp Ave(dBm)	12.4	15.07	14.45
Sens Ave(dBm)	-80.8	-80.62	-80.54		Sens Ave(dBm)	-67.25	-66.49	-65.66

5.2 测试工具及方法

下面提到的**PDF/TXT**文档在**docs/linux/wifibt**目录下查找，且如果客户没有专业测试仪器或者有测试疑问，请直接找模组厂协助。

5.2.1 Realtek 测试

一般分为两种COB和模组，模组一般都是经过模组厂严格测试并出厂默认烧录校准好的数据到内部efuse，客户只需测试验证指标是否合格；而COB则自行设计Wi-Fi外围电路及添加器件，所以需要跟realtek合作进行完整的rf校准测试，并集成校准好的数据到芯片的efuse里面或由驱动进行加载，具体请直接咨询模组厂。

Wi-Fi测试

```
参考 Quick_Start_Guide_V6.txt, 特别注意里面的命令iwpriv统一换成rtwpriv。
```

BT测试

参考 MP tool user guide for linux20180319.pdf（里面的具体测试内容请咨询模组厂或者原厂），特别注意测试需要模组厂提供专门用于测试的蓝牙firmware文件，并放到指定目录下才可以测试蓝牙指标。

```
# 注：执行测试前请先打开 bt 电源
echo 0 > sys/class/rfkill/rfkill0/state
sleep 1
echo 1 > sys/class/rfkill/rfkill0/state

# 特别注意：rtk_hciattach这个进程不能跑，如果有请杀掉killall rtk_hciattach

////////////
/ #
/ # rtlbtmp
::::::::::::::::::::::::::::::::::::::::::::::::
:::::::: Bluetooth MP Test Tool Starting ::::::::

>
>
>enable uart:/dev/ttyS4    # 注意ttySX 对应实际硬件接的uart口
>
>> > enable[Success:0]
```

当使用**Realtek**的**cob**方案时，如果需要把测试得到的校准数据**map**文件集成到驱动里面时，方法如下：

```
drivers/net/wireless/rockchip_wlan/rtl8xxx/core/efuse/rtw_efuse.c
#ifdef CONFIG_EFUSE_CONFIG_FILE
u32 rtw_read_efuse_from_file(const char *path, u8 *buf, int map_size)
{
    u32 i;
    u8 c;
    u8 temp[3];
    u8 temp_i;
    u8 end = _FALSE;
    u32 ret = _FAIL;

    u8 *file_data = NULL;
    u32 file_size, read_size, pos = 0;
    u8 *map = NULL;

    if (rtw_is_file_readable_with_size(path, &file_size) != _TRUE) {
        RTW_PRINT("%s %s is not readable\n", __func__, path);
        goto exit;
    }

    file_data = rtw_vmalloc(file_size);
    if (!file_data) {
        RTW_ERR("%s rtw_vmalloc(%d) fail\n", __func__, file_size);
        goto exit;
    }

    #if 0  //屏蔽掉这里
    read_size = rtw_retrieve_from_file(path, file_data, file_size);
    if (read_size == 0) {
        RTW_ERR("%s read from %s fail\n", __func__, path);
        goto exit;
    }
    ... ...
    RTW_PRINT("efuse file:%s, 0x%03x byte content read\n", path, i);
    #endif

    //把模组厂提供的校准 “map文件” 改成数组形式，赋值给map即可。
    _rtw_memcpy(buf, map, map_size); //这里是map最终赋值给buf操作

    ret = _SUCCESS;

exit:
    if (file_data)
        rtw_vmfree(file_data, file_size);
    if (map)
        rtw_vmfree(map, map_size);

    return ret;
}
```

5.2.2 AP/CY测试

Wi-Fi测试

首先要更换为测试firmware：每一个AP模组对应的测试firmware都是不一样的，比如：

```
AP6236  -> fw_bcm43436b0_mfg.bin
AP6212A -> fw_bcm43438a1_mfg.bin
```

fw_bcmxxx_mfg.bin 和 APxxxx 要根据你的模组型号进行匹配，否则无法测试！所以先确认是否有对应型号的测试firmware，如果没有找模组厂提供即可。

最新SDK内置了支持型号的测试firmware，所以直接使用内置的测试脚本，使Wi-Fi进入RF测试模式：

```
external\rkwifibt\wifi_ap6xxx_rftest.sh
#!/bin/sh
killall ipc-daemon netserver connmand wpa_supplicant
echo "Pull BT_REG_ON to Low"
echo 0 > /sys/class/rfkill/rfkill0/state
echo "Pull WL_REG_ON to Up"
echo 1 > /sys/class/rfkill/rfkill1/state
sleep 1
echo "update wifi test fw"
echo /vendor/etc/firmware/fw_bcmdhd_mfg.bin >
/sys/module/bcmdhd/parameters/firmware_path
sleep 1
ifconfig wlan0 down
ifconfig wlan0 up
sleep 1
echo "wl ver"
wl ver
```

旧sdk没有内置测试firmware，下面举例说明下：

```
# AP6236
# 把fw_bcm43436b0_mfg.bin push到data或其他可写分区，然后执行如下命令：(注意下面的路径)
mount --bind /data/fw_bcm43436b0_mfg.bin /system/etc/firmware/fw_bcm43436b0.bin
ifconfig wlan0 down
ifconfig wlan0 up
wl ver

# CYW43438
echo /data/cyw43438-7.46.58.25-mfgtest.bin >
/sys/module/cywdhd/parameters/firmware_path
ifconfig wlan0 down
ifconfig wlan0 up
wl ver
```

正常的话执行wl ver会打印一串字符，里面有WL_TEST字样，表示进入测试模式，具体的测试参考：

```
Wi-Fi RF Test Commands for Linux-v03.pdf
```

蓝牙测试

执行bt_init.sh脚本（SDK自带脚本，参考3.3章节）后，执行：**(**注意：如果没有**hciconfig**命令，请在**Buildroot**配置选择**BR2_PACKAGE_BLUEZ5_UTILS** 编译并更新测试**)**

```
hciconfig hci0 up
hciconfig -a
```

如果有出现hci0节点就表示初始化完成，如果没有出现有两种可能：

1. 蓝牙dts配置异常或硬件异常或者uart口配置错误，导致初始化失败；
2. 蓝牙firmware文件配置错误或者没有该文件；

以上两个请参考2/3章节的BT相关的进行排查；

具体测试指令请参考：

```
BT RF Test Commands for Linux-v05.pdf #文档里面测试1/2步都在脚本里面执行过了，无需再执行）
```

5.3 报告

请硬件确认完以上测试后，请整理一份测试报告出来，在遇到性能或稳定性时问题时需要提供报告给我们。

6. Wi-Fi 性能测试

使用**iperf**测试性能

注意两个影响性能的点：

- 一定要完成Wi-Fi的RF测试及天线的OTA测试后，确保指标没有问题再测试性能，否在无意义；
- 如果发现数据波动较大，请到空旷或地下室等干扰小的地方确认（最好在屏蔽室测试）；

测试环境: 由于开放环境下干扰因素比较大，建议在屏蔽室环境下测试，首先确保Wi-Fi可以正常连接到AP并获取到IP地址；
测试点: 吞吐率的大小，以及稳定性、是否有上下波动等等；
路由器信道: 要选择低、中、高信道分别进行测试， 比如1/6/11信道：

```
# TCP
下行：
    板子端：iperf -s -i 1
    电脑：   iperf -c xxxxxxx(板子的ip地址) -i 1 -w 2M -t 120

上行：
    电脑：iperf -s -i 1
    板子端：iperf -c xxxxxxx(电脑的ip地址) -i 1 -w 2M -t 120

# UDP
下行：
    板子端：iperf -s -u -i 1
    电脑：iperf -c xxxxxxx(板子的ip地址) -u -i 1 -b 100M -t 120
```

```
上行：
    电脑：iperf -s -u -i 1
    板子端：iperf -c xxxxxxx(电脑的ip地址) -u -i 1 -b 100M -t 120

# 注意：板子的iperf命令要在Buildroot端进行配置：BR2_PACKAGE_IPERF = y
```

7. Wi-Fi/BT问题排查

7.1 Wi-Fi识别流程简述

SDIO 接口：开机时kernel MMC框架会去初始化SDIO WiFi设备，首先会去解析dts里面的**sdio_pwrseq**节点的**reset-gpios**属性配置的GPIO，也就是**WL_REG_ON**然后拉高它，并通过SDIO_CLK/CMD/DATA发初始化指令给模块，首先主控会以400/300/200K的低频去访问模块，询问它基本的信息：SDIO2.0 (CLK最大50M)还是3.0 (CLK最大208M)、支持4线还是1线等信息，然后根据支持的规格提升**CLK**频率到高频，此时初始化基本完成，可以看到如下log：

```
# 注意 mmc0: 0的数字是不固定的，也可能是0/1/2; ff4a0000：表示控制器的地址，不同平台也是不一样的；
dwmmc_rockchip ff4a0000.dwmmc: allocated mmc-pwrseq
# 低频
mmc_host mmc0: Bus speed (slot 0) = 400000Hz (... actual 400000HZ div = 0)
# 提频
mmc_host mmc0: Bus speed (slot 0) = 50000000Hz (... actual 50000000HZ div = 0)
mmc0: new high speed SDIO card at address 0001  #SDIO 2.0
# 或
mmcx: new ultra high speed SDR104 SDIO card at address 0001  #SDIO 3.0
```

USB/PCIE接口：这两个接口识别过程比较复杂，请参考doc目录下的USB/PCIE相关文档，当识别完成后：

USB接口：正常识别的话执行**lsusb**可以看到如下信息：

```
Bus 001 Device 002: ID 0bda:f179 Realtek Semiconductor Corp. RTL8188FTV
802.11b/g/n 1T1R 2.4G WLAN Adapter
```

或 dmesg | grep usb #看下是否有识别到usb device设备。

PCIE接口：正常识别的话执行**lspci**可以看到如下信息：

```
0002:21:00.0 Network controller: Broadcom Inc. and subsidiaries Device 449d (rev
02)
```

注意：以上识别过程都是在开机过程中识别的，只有识别到正确的设备后，**WiFi**驱动才会被正确加载！

7.2 Wi-Fi问题

Wi-Fi是SDIO接口：一般有两种情况

- 首先确认在 kernel log找下如下LOG，没有则表示SDIO卡没有被识别到，这类问题参考**7.1.1**章节；

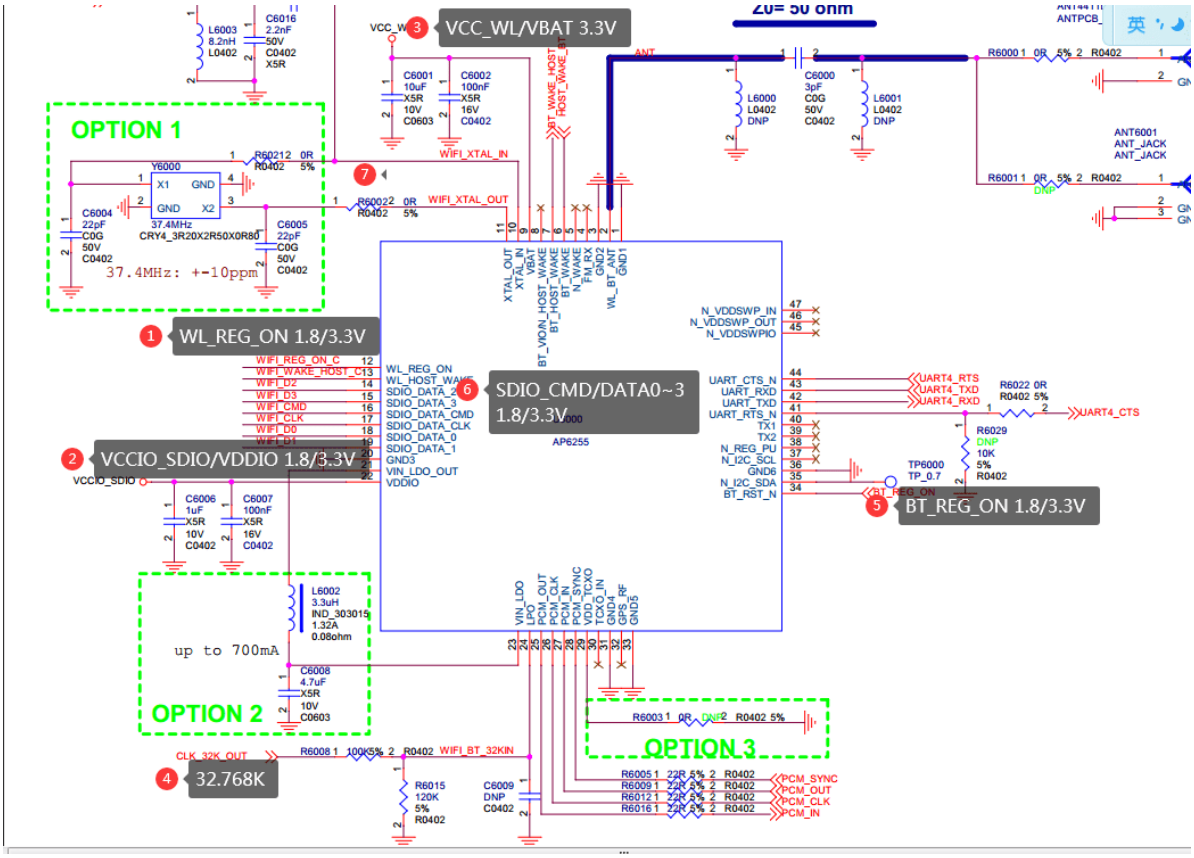
```
mmc0: new high speed SDIO card at address 0001
# 或
mmcx: new ultra high speed SDR104 SDIO card at address 0001
```

- 如果有看到正常识别SDIO的LOG，但wlan0没有或者wlan0 up失败；参考**7.1.2/7.1.3/7.1.4**章节；

7.2.1 Wi-Fi 异常SDIO识别不到

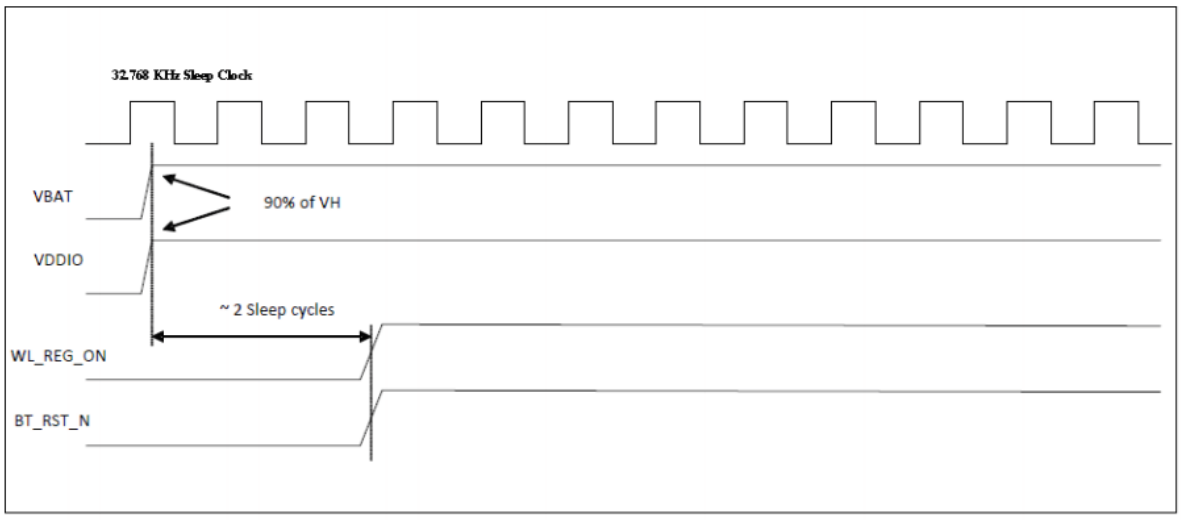
则按照以下步骤对照内核**LOG**依次排查：

- 首先强烈建议再次详细检查下第**2**章节的**DTS/KERNEL**配置！！！
- 对照下图，按照标号依次测量对应PIN脚的电平状态及CLK频率是否正确（注：**SDIO3.0**模式必须为**1.8V**）



- 用示波器测量**VBAT/VDDIO/WL_REG_ON**的时序，确保符合下面的时序要求：

很多客户都是因为VDDIO上电时序异常导致SDIO无法识别。



- **WL_REG_ON: Enable pin for WLAN device ON: pull high ; OFF: pull low [Voltage: VDDIO]**

DTS的WL_REG_ON 配置错误，导致Wi-Fi使能脚拉不高；

方法一：可用示波器测试它的波形，看下是否有被拉高拉低，电压幅值**(1.8/3.3V)**是否符合要求；如果测到连续拉高拉低几次则是正常情况，因为**SDIO**初始化时会**retry**几次，当识别成功：**WL_REG_ON**会被拉高，识别失败则会被拉低；如果一直为低则表示**DTS**配置错误；

方法二：硬件上直接把**WL_REG_ON**拉高验证，如果可以识别也表示DTS WL_REG_ON配置错误；

典型异常LOG如下：（**mmcX: X**的数字不固定）

```
mmc_host mmc1: Bus speed (slot 0) = 300000Hz (slot req 300000Hz, actual 300000HZ
div = 0)
mmc_host mmc1: Bus speed (slot 0) = 200000Hz (slot req 200000Hz, actual 200000HZ
div = 0)
mmc_host mmc1: Bus speed (slot 0) = 100000Hz (slot req 100000Hz, actual 100000HZ
div = 0)
```

- **DTS WL_REG_ON 配置错误**

sdio_pwrseq 跟 **wireless-wlan** 里面都配置了**WL_REG_ON**，导致重复了，回看**DTS**配置说明**!**

```
[WLAN_RFKILL]: Enter rfkill_wlan_init
[WLAN_RFKILL]: Enter rfkill_wlan_probe
[WLAN_RFKILL]: wlan_platdata_parse_dt: wifi_chip_type = ap6255
[WLAN_RFKILL]: wlan_platdata_parse_dt: enable wifi power control.
[WLAN_RFKILL]: wlan_platdata_parse_dt: wifi power controled by gpio.
of_get_named_gpiod_flags: parsed 'WIFI,poweren_gpio' property of node '/wireless-
wlan[0]' - status (0)
[WLAN_RFKILL]: wlan_platdata_parse_dt: WIFI,poweren_gpio = 6 flags = 0.
of_get_named_gpiod_flags: can't parse 'WIFI,vbat_gpio' property of node
'/wireless-wlan[0]'
of_get_named_gpiod_flags: can't parse 'WIFI,reset_gpio' property of node
'/wireless-wlan[0]'
of_get_named_gpiod_flags: parsed 'WIFI,host_wake_irq' property of node
'/wireless-wlan[0]' - status (0)
[WLAN_RFKILL]: wlan_platdata_parse_dt: WIFI,host_wake_irq = 8, flags = 0.
[WLAN_RFKILL]: wlan_platdata_parse_dt: The ref_wifi_clk not found !
[WLAN_RFKILL]: rfkill_wlan_probe: init gpio
gpio-6 (reset): gpiod_request: status -16
[WLAN_RFKILL]: Failed to get rkwifi_wlan_poweren gpio.
wlan-platdata: probe of wireless-wlan failed with error -1
```

- **VDDIO/SDIO: I/O Voltage supply input**

PIN12脚的VDDIO供电要为3.3/1.8V，且SDIO_CMD/SDIO_DATA0~3对应的也必须为3.3v或1.8V。

- **IO**电源域

VDDIO：供电电压跟**DTS**电源域配置不匹配（参考**2.2.3**章节**IO**电源域配置）；

- **SDIO_CLK** 没有波形

SDIO_CLK：IO电源域设置错误可能会导致CLK测不到波形（参考2.2.3章节IO电源域配置）；

SDIO_CLK：是否有加上拉电阻，去掉测试；

- 电源供电不足或纹波太大

VCC_WL/VBAT/VDDIO_SDIO：电源供电不足或纹波太大

```
# Realtek模块
RTL871X: ### rtw_hal_ops_check - Error : Please hook HalFunc.read_chip_version
###
RTL871X: ### rtw_hal_ops_check - Error : Please hook HalFunc.init_default_value
###
RTL871X: ### rtw_hal_ops_check - Error : Please hook HalFunc.intf_chip_configure
###
RTL871X: ### rtw_hal_ops_check - Error : Please hook HalFunc.read_adapter_info
###
RTL871X: ### rtw_hal_ops_check - Error : Please hook HalFunc.hal_power_on ###
RTL871X: ### rtw_hal_ops_check - Error : Please hook HalFunc.hal_power_off ###
```

- **32.768K**

External clock reference (External LPO signal characteristics)

Parameter	**Specification**	**Units**
Nominal input frequency	32.768	kHz
Frequency accuracy	± 30	ppm
Duty cycle	30 - 70	%
Input signal amplitude	400 to 1800	mV, p-p
Signal type	Square-wave	-
Input impedance	>100k <5	Ω pF
Clock jitter (integrated over 300Hz – 15KHz)	<1	Hz
Output high voltage	0.7Vio - Vio	V

CLK_32K_OUT：**32.768K**没有波形、或者波形精度不满足上面表格的要求！

```
#正基/海华模块，无32k的异常log：
[   11.068180] dhdsdio_htclk: HT Avail timeout (1000000): clkctl 0x50
[   11.074372] dhd_bus_init: clock state is wrong. state = 1
[   12.078468] dhdsdio_htclk: HT Avail timeout (1000000): clkctl 0x50
[   12.086051] dhd_net_bus_devreset: dhd_bus_devreset: -1
```

- **PCB**走线质量/电容/电感不合适/**IO_DOMAIN**电压设置错误

情况一：SDIO_CLK/CMD/DATAX：**PCB**走线异常、电容电感不符合要求、接触不良、焊接不良导致等问题导致初始化异常或跑着不了高频，可以适当降低频率确认（修改**&sdio**节点下的**max-frequency** ），如果降频可以，则要找硬件工程师测波形确认是否符合**WiFi**模块的**datasheet**里面关于**CLK/CMD/DATA Timing**的要求。

SDIO2.0: SDIO **High Speed** Mode Timing Diagram **(CLK <= 50M)**;

SDIO3.0: SDIO Bus Timing Specifications **in SDR Modes (SDR104/DDR50) (50M < CLK <= 208M)**;

情况二：Wi-Fi和主控sdio电压不匹配，请检查软件io_domian配置是否和硬件一致（参考2.2.3章节）

```
# 异常log1
mmc_host mmc1: Bus speed(slot0)=100000000Hz(slotreq 100000000Hz,actual
100000000HZ div=0)
dwmmc_rockchip ff0d0000.dwmmc: All phases bad!
mmc1: tuning execution failed
mmc1: error -5 whilst initialising SDIO card

# 异常log2，比如下载firmware失败等类似的数据通信异常的log：
sdioh_buffer_tofrom_bus: TX FAILED ede95000, addr=0x08000, pkt_len=1968, ERR=-84
_dhdsdio_download_firmware: dongle image file download failed
dhd_bus_devreset Failed to download binary to the donglesdio

# 异常log3
dwmmc_rockchip 30120000.rksdmmc: Busy; trying anyway
sdioh_buffer_tofrom_bus: RX FAILED c52ce000, addr=0x0f154, pkt_len=3752, ERR=-5
dhdsdio_membytes: membytes transfer failed
bcmsdh_sdmmc: Failed to Write byte F1:@0x1000a=00, Err: -5
bcmsdh_sdmmc: Failed to Write byte F1:@0x1000a=00, Err: -5
bcmsdh_sdmmc: Failed to Write byte F1:@0x1000a=00, Err: -5
dhdsdio_membytes: FAILED to set window back to 0x18100000
```

- **WL_HOST_WAKE: WLAN to wake-up HOST [Voltage: VDDIO]**
  WL_HOST_WAKE PIN 脚或中断电平配置错误（参考2.2.1章节），或者虚焊，导致如下异常或系统卡住：

```
# 正基海华模块，异常log：
dhd_bus_rxctl: resumed on timeout, INT status=0x208000C0
dhd_bus_rxctl: rxcnt_timeout=1, rxlen=0
```

修改频率的方法：

```
&sdio {
+    max-frequency = <10000000>;    # 修改这里，限制频率
```

- **SDIO_D1~3**某根线焊接不良

如果出现如下类似log，且降低max-frequency频率到低于10M都还不行：

```
[   22.430412] mmc_host mmc3: Bus speed (slot 0) = 375000Hz (slot req 400000Hz,
actual 375000HZ div = 0)
[   22.447549] mmc_host mmc3: Bus speed (slot 0) = 375000Hz (slot req 375000Hz,
actual 375000HZ div = 0)
[   22.476779] mmc3: queuing unknown CIS tuple 0x80 (2 bytes)
[   22.478881] mmc3: queuing unknown CIS tuple 0x80 (3 bytes)
[   22.480874] mmc3: queuing unknown CIS tuple 0x80 (3 bytes)
[   22.484301] mmc3: queuing unknown CIS tuple 0x80 (7 bytes)
[   22.598965] mmc_host mmc3: Bus speed (slot 0) = 148500000Hz (slot req
150000000Hz, actual 148500000HZ div = 0)
[   23.466816] dwmmc_rockchip fe000000.dwmmc: Unexpected xfer timeout, state 3
[   24.370310] dwmmc_rockchip fe000000.dwmmc: All phases bad!
[   24.370391] mmc3: tuning execution failed: -5
```

则修改为1线模式测试：

```
&sdio {
+   bus-width = <1>;        /* 调整1线模式测试 */
```

如果有效果，则表示硬件的SDIOD1~3某根线硬件有问题，虚焊或者接错；

- 模组/芯片为不良品

有小概率情况发现主控芯片或Wi-Fi模块为不良品，可以多找几台机器做验证；

- **iomux** 复用异常

```
rockchip-pinctrl pinctrl: pin gpio3-4 already requested by ff120000.serial;
cannot claim for ff5f0000.dwmmc
rockchip-pinctrl pinctrl: pin-100 (ff5f0000.dwmmc) status -22
rockchip-pinctrl pinctrl: could not request pin 100 (gpio3-4) from group
sdmmc0ext-bus4  on device rockchip-pinctrl
dwmmc_rockchip ff5f0000.dwmmc: Error applying setting, reverse things
```

- 其它

如果以上排查不出问题，请在**redmine**系统上传：**dts/dtsi**配置，内核完整**log dmesg**，**pdf**原理图等文件，并提供：开机时**WIFI_REG_ON**和**Wi-Fi_CLK/Wi-Fi CMD**三根线的上电波形图；

7.2.2 USB Wi-Fi 排查

USB Wi-Fi正常情况下会有类似如下usb信息，如果没有则**Wi-Fi**驱动不会加载！

```
#识别到USB设备
usb 2-1: new high-speed USB device number 2 using ehci-platform
#厂家PID/VID
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
```

如果没有上述信息则参考**2.2**章节，排查模块使能脚**VBAT/WL_REG_ON**是否有拉高及**USB**相关的是否配置正确！！！

注意：RV1106/1103有时需要手动输入如下命令才可以识别：

echo 1 > /sys/devices/platform/ff3e0000.usb2-phy/otg_mode

7.2.3 Realtek Wi-Fi 特别注意点

当上面的章节排查完毕后都正常，但Realtek Wi-Fi还是识别不到SDIO，则确认以下两点：

7.2.3.1 wlan0有识别到但扫描异常

- 小瑞 欧智通**Wi-Fi**模组：PIN25（对应COB 芯片的22 PIN）脚，它是芯片内部复用管脚， 一个是作为PCM_OUT 连接 RK芯片的PCM_IN，用作蓝牙 PCM 通话功能； 另外一个作为 Wi-Fi IC LDO_SPS_SEL 功能(SPS 或者 LDO 模式选择)， 当这个管脚为低电平表示 SPS 模式， 如果为高电平为 LDO 模式， 所以这个**PIN**要么高电平要么低电平，如果出现半电平， 会导致 Wi-Fi 无法正常使用， 比如无法扫描 SDIO 或者扫描不到 AP 问题。如果测到半电平硬件上要做上拉处理。
- 必联模块：PIN7 必须接下拉。

7.2.3.2 Realtek 支持**SDIO3.0**

Realtek Wi-Fi SDIO 3.0异常
当使用高端的比如：RTL8821CS等支持3.0模块时，概率初始化不过，异常log如下：

```
mmc_host mmc1: Bus speed (slot 0) = 400000Hz (slot req 400000Hz, actual 400000HZ
div = 0)
mmc_host mmc1: Voltage change didn't complete
mmc1: error -5 whilst initialising SDIO card
```

请打上以下补丁：

```
diff --git a/drivers/mmc/core/sdio.c b/drivers/mmc/core/sdio.c
index 2046eff..6626752 100644
--- a/drivers/mmc/core/sdio.c
+++ b/drivers/mmc/core/sdio.c
@@ -646,7 +646,7 @@ static int mmc_sdio_init_card(struct mmc_host *host, u32 ocr,
* try to init uhs card. sdio_read_cccr will take over this task
* to make sure which speed mode should work.
*/
- if (!powered_resume && (rocr & ocr & R4_18V_PRESENT)) {
+ /*if (!powered_resume && (rocr & ocr & R4_18V_PRESENT)) {
        err = mmc_set_uhs_voltage(host, ocr_card);
        if (err == -EAGAIN) {
            mmc_sdio_resend_if_cond(host, card);
@@ -655,7 +655,10 @@ static int mmc_sdio_init_card(struct mmc_host *host, u32
ocr,
        } else if (err) {
            ocr &= ~R4_18V_PRESENT;
        }
- }
+ }*/
+
+ ocr &= R4_18V_PRESENT;
```

```
#方法二
    if (host->ops->card_busy && !host->ops->card_busy(host)) {
+#if 0 /* SDIO 3.0 patch for Realtek 88x2BS */
        err = -EAGAIN;
        goto power_cycle;
+#else
+       pr_warning("%s: Ignore checking low after cmd11\n",
+               mmc_hostname(host));
+#endif
    }
```

7.2.4 RV1109/1126 平台wlan0 无法up

请务必确认：**busybox ps:** 没有出现**ipc_daemon/connmand/netserver**等进程！

```
# ifconfig -a   #有wlan0接口，但是ifconfig wlan0 报这个错误
# ifconfig: SIOCSIFFLAGS: Operation not possible due to RF-kill

#Buildroot配置修改如下配置并保存：
BR2_PACKAGE_IPC_DAEMON = n
BR2_PACKAGE_NETSERVER = n
BR2_PACKAGE_CONNMAN = n
BR2_PACKAGE_DHCPCD = y
#删掉中间文件：
buildroot/output/rockchip_rv1126_rv1109_xxx/target/etc/init.d/S45connman
buildroot/output/rockchip_rv1126_rv1109_xxx/target/usr/bin/connmanctl
buildroot/output/rockchip_rv1126_rv1109_xxx/target/usr/sbin/connmand
buildroot/output/rockchip_rv1126_rv1109_xxx/target/usr/sbin/ipc_daemon
buildroot/output/rockchip_rv1126_rv1109_xxx/target/usr/sbin/netserver
#重新编译确保开机后 busybox ps: 没有出现ipc_daemon/connmand/netserver等进程即可
```

7.2.5 Wi-Fi SDIO卡识别到但wlan0 up失败

kmesg log出现如下log表示SDIO正常识别到了，但Wi-Fi还是无法使用。

```
mmcx: new high speed SDR104 SDIO card at address 0001
#或
mmcx: new ultra high speed SDR104 SDIO card at address 0001
```

- Wi-Fi 驱动ko模式：insmod ko加载错误，参考**2.4**章节；
- Wi-Fi 驱动buildin模式：内核配置错误，参考**2.4**章节；
- Firmware 文件不存在或文件名跟模组型号不匹配（只针对正基和海华模组），参考**3.3**章节；

```
[dhd] dhd_conf_set_path_params : Final
clm_path=/vendor/etc/firmware/clm_bcm43438a1.blob
[dhd] dhd_conf_set_path_params : Final conf_path=/vendor/etc/firmware/config.txt
  dhdsdio_download_code_file: Open firmware file failed
/vendor/etc/firmware/fw_bcm43438a1.bin
_dhdsdio_download_firmware: dongle image file download failed
```

- Wi-Fi 模块供电电源不稳定

```
[ 14.059448] RTW: ERROR sd_write8: FAIL!(-110) addr=0x10080 val=0x00
[ 14.059602] RTW: ERROR _sd_cmd52_read: FAIL!(-110) addr=0x00086
[ 14.059615] RTW: ERROR sdio_chk_hci_resume((null)) err:-110
```

- RTL模块扫描异常，比如扫描不到AP

参考7.2.3.1章节

```
[ 43.859911] RTW: sdio_power_on_check: val_mix:0x0000063f, res:0x0000063f
[ 43.859932] RTW: sdio_power_on_check: 0x100 the result of cmd52 and cmd53 is the
same.
[ 43.860022] RTW: sdio_power_on_check: 0x1B8 test Pass.
[ 43.860115] RTW: _InitPowerOn_8723DS: Test Mode
[ 43.860156] RTW: _InitPowerOn_8723DS: SPS Mode
```

- 内核配置的**buildin**和**ko**模式同时选或者配置不匹配

1. Wi-Fi驱动加载过早导致异常，从log可以看到驱动在0点几秒就被加载了，此时sdio/usb设备还没枚举到，检查内核配置章节；
2. 系统同时加载了两个ko，一个bcmdhd.ko，另一个是88xx.ko导致异常，原因是参考第3章节的编译说明；

7.2.6 Wi-Fi 跑一段时间后异常（SDIO接口）

情况一：SDIO_CLK/CMD/DATAX：**PCB**走线异常、电容电感不符合要求、接触不良、焊接不良导致等问题导致初始化异常或跑着不了高频，可以适当降低频率确认（修改**&sdio**节点下的**max-frequency** ），如果降频可以，则要找硬件测波形确认是否符合**WiFi**模块的**datasheet**里面关于**CLK/CMD/DATA Timing**的要求。

SDIO2.0: SDIO **High Speed** Mode Timing Diagram (CLK <= 50M);

SDIO3.0: SDIO Bus Timing Specifications **in SDR Modes (SDR104/DDR50) (50M < CLK <= 208M)**;

```
&sdio {
+    max-frequency = <10000000>;    # 修改这里，限制频率
```

情况二：Wi-Fi和主控sdio电压不匹配，请检查软件io_domian配置是否和硬件一致（参考2.2.3章节）

7.2.7 Wi-Fi 无法连接路由器或断线或连接不稳定或连接时间慢

这类问题基本都是**Wi-Fi**芯片或者模块的**RF**、天线指标不合格导致的，确认方法如下：

软件简单连接测试参考**4.1**章节！

- RF指标

首先找硬件工程师要Wi-Fi RF指标及天线整机OTA测试合格的报告，确保硬件指标正常（发射功率、**EVM**、晶体频偏、接收灵敏度）；

- 扫描对比

做个基本的扫描对比：测试设备和手机放在距离路由器的同一位置，通过扫描的AP的个数及其信号强度跟手机（或相同规格的竞品）做对比 来初步判定硬件指标是否正常，扫描方法参考4.1章，这里说明如何跟手机做对比，找台Android手机，从Setting里面进到开发者选项，开启WLAN详细日志记录的选项，回到WLAN设置界面可以看到AP的详细信息包括RSSI，通过对比跟手机扫描的热点数量及信号强度来初步判定硬件指标是否合格；

- 干扰

排除干扰因素，比如当前环境有非常多2.4/5G的无线设备在同时连接，使得当前环境干扰很大时，则可以把测试设备和对比手机（或相同规格的竞品）放在同一位置，如果都会出现异常，则可判断为干扰，如果手机正常，则可判定是硬件指标异常；

- 距离

排除距离因素，距离太远导致信号弱（通过扫描wpa_cli scan/scan_r，连接的AP信号强度为-70～-90之间），进而导致通信失败，可以拉近两者的距离来确认；

- 路由器兼容性

排除路由器兼容性问题，可以更换不同厂家型号的路由器确认；

- 一致性

排除单板异常问题，可以拿二到三台机器做对比测试；

- 找模组厂协助

可以直接找模组厂或Wi-Fi原厂协助，他们有专业的抓包仪器抓空中包，能快速定位问题；

- 带环境去我们深圳办公室确认

7.2.8 吞吐率不符合预期

- 如果是SDIO接口，RK平台接口最大只支持到150M，协议最大支持208M，所以硬件本身会损失一部分性能；
- 其次是RF指标先测试通过，必要时提供测试报告及天线OTA测试报告；
- 然后找个屏蔽室或干净环境如地下停车场进行测试，排除环境干扰，先确保干净的环境是正常的；
- 最后可以做CPU/DDR定频验证；

```
#DDR
cat /sys/devices/platform/dmc/devfreq/dmc/available_frequencies
echo userspace > /sys/devices/platform/dmc/devfreq/dmc/governor
echo 1560000000 > /sys/devices/platform/dmc/devfreq/dmc/min_freq
cat /sys/devices/platform/dmc/devfreq/dmc/cur_freq

#CPU
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo 1992000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq

#将dw-mmc 中断放到其它cpu core 进行验证

参考：
如切换CPU core 中断验证。
cat /proc/interrupts 查看相应中断号
echo 5 > /proc/irq/38/smp_affinity_list //把中断放到cpu2上执行
```

```
cat /proc/interrupts //查看记数
```

7.2.9 IP异常

获取不到IP地址或IP地址冲突，请确认dhcpcd或udhcpc进程是否有启用；

dhcpcd：SDK默认使用，随系统启动而启动，它是功能比较完整的dhcp客户端；

udhcpcd：是busybox的精简的dhcp客户端；

注意：这两个进程一定不要同时启用，只能使用其中一个！

7.2.10 休眠唤醒异常

休眠后被Wi-Fi 频繁唤醒，排除以下几种情况：

- 休眠后Wi-Fi模组的32.768k被关闭；
- WIFI_REG_ON被拉低；
- WIFI_WAKE_HOST PIN硬件不稳定，电平抖动（正常情况下都是低电平，触发为高电平）；
- AP/CY模组Wi-Fi休眠前后没有执行如下特定命令来过滤广播或组播包：

```
#休眠前执行：
dhd_priv setsuspendmode 1
#唤醒后执行：
dhd_priv setsuspendmode 0
```

7.2.11 PING 异常

PING 不通或概率延迟很大

- RF指标没测过；
- 大概率是Wi-Fi在执行扫描操作，导致ping延迟很大；
- 路由器或者PC防火墙打开导致禁ping操作；

7.2.12 客户自定义修改

经常有收到一些奇怪的问题，排查发现客户自己修改了一些Wi-Fi/BT的驱动配置导致异常，所以请先检查自己是否有修改过原始代码，如果有先回退确认，修改的地方及原因可以通过redmine发给我们确认。

7.2.13 wlan0正常，但扫描不到任何AP

- 检查模组对应的晶振是否跟芯片的要求的一致，比如Wi-Fi要求24M，但接了一个37M的；
- 32.768k的精度不符合要求；

7.2.14 双Wi-Fi AP + RTL 异常

接两个Wi-Fi，一个是sdio接口的AP6xxx，另一个是USB接口的RTL8xxxu。kernel启动后，两个初始化都正常，但操作RTLxxxbu做接口down的时候，kernel挂掉.

```
diff --git a/drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/wl_cfg80211.c
b/drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/wl_cfg80211.c
index f4838a8..ceb2a00 100644
--- a/drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/wl_cfg80211.c
+++ b/drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/wl_cfg80211.c
@@ -14640,6 +14640,9 @@ wl_cfg80211_netdev_notifier_call(struct notifier_block *
nb,
    if (!wdev || !cfg || dev == bcmcfg_to_prmry_ndev(cfg))
        return NOTIFY_DONE;

+   if(strncmp(dev->name, "wlan0",strlen("wlan0"))) {
+       return NOTIFY_DONE;
+   }
    switch (state) {
        case NETDEV_DOWN:
        {
```

7.2.15 南方硅谷Wi-Fi异常

在RV1109/1126平台发现能识别到SDIO，但是读写出错，尝试如下修改：

```
&sdio {
    //rockchip,default-sample-phase = <90>; #删掉这个配置选项
```

7.2.16 IOS 系统问题

- iOS 系统的iPhone开热点连不上

```
+Realtek模块修改
sdk_project@aaaaa:~/poco/3399/10/kernel/drivers/net/wireless/rockchip_wlan/rtl818
8eu$ git diff .
diff --git a/os_dep/linux/ioctl_cfg80211.c b/os_dep/linux/ioctl_cfg80211.c
index 2fe9c2b..ba24cb7 100755
--- a/os_dep/linux/ioctl_cfg80211.c
+++ b/os_dep/linux/ioctl_cfg80211.c
@@ -10114,7 +10114,7 @@ static int rtw_cfg80211_init_wiphy(_adapter *adapter,
struct wiphy *wiphy)
 #endif

 #if (KERNEL_VERSION(3, 8, 0) <= LINUX_VERSION_CODE)
-       wiphy->features |= NL80211_FEATURE_SAE;
+       //wiphy->features |= NL80211_FEATURE_SAE;
 #endif

+博通模块修改：
```

```
diff --git a/drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/Makefile
b/drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/Makefile
index 39c1984..1f156d8 100644
--- a/drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/Makefile
+++ b/drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/Makefile
@@ -157,7 +157,7 @@ ifneq ($(CONFIG_CFG80211),)
     DHDCFLAGS += -DWLTDLS -DMIRACAST_AMPDU_SIZE=8
     DHDCFLAGS += -DWL_VIRTUAL_APSTA
     DHDCFLAGS += -DPNO_SUPPORT -DEXPLICIT_DISCIF_CLEANUP
-    DHDCFLAGS += -DWL_SAE
+    #DHDCFLAGS += -DWL_SAE
```

- iOS连接设备的热点后，iOS设备会在10s左右网络会断开然后自动重连的异常

原因：当iOS连上设备的热点后，设备会给iOS设备生产一个DHCP租约文件，这个租约文件会被存放在临时的位置，重启后会丢失；当使用dnsmasq分配IP时，则需在启动命令指定如下参数：

```
-l, --dhcp-leasefile=<path>  #path路径必须掉电不丢失的
```

其他应用实现参考这个即可。

7.2.17 正基模块问题

无非就是如下几个问题，请务必仔细检查：

1. WIFI_REG_ON配置错误；
2. 32.768K 没有出来或者信号不达标；
3. WIFI_WAKE_HOST：管脚配置错误或电平配置错误；
4. 休眠时32.768K被关掉，或WIFI_REG_ON被关掉；
5. firmware/nvram固件不存在，在log的很明显能看到下载文件时出错；
6. 正基(bcmdhd.ko)/海华(cywdhd.ko)模块驱动不能通用，不要配错驱动；

7.2.18 PCIe WiFi识别不到

请参考2.2.5章节，检查所有提到的配置，并务必测量相关的电源电压是否正常！

7.3 蓝牙问题

第**4.3**章节蓝牙测试项失败 或 **deviceio_test bluetooth**蓝牙功能异常，无非就下面几种情况，请按照如下步骤仔细排查：

- Realtek模组必须要关掉内核这个配置：**CONFIG_BT_HCIUART=n** 参考**2.4.2**章节；
- DTS BT (**BT_RST_N**)电源使能PIN脚配置不对，参考**2.2.2**章节；

```
#可以通过以下指令来确认：
echo 0 > sys/class/rfkill/rfkill0/state #拉低BT_REG_ON, 用万用表测量对应的PIN
echo 1 > sys/class/rfkill/rfkill0/state #拉高BT_REG_ON, 用万用表测量对应的PIN
```

- UART配置错误，参考**2.2.2/2.4.2/2.5**及章节；

- Realtek模组需要rtk_hciattach / hci_uart.ko / rtk_btusb.ko，确认下没有正确编译或内核配置错误，参考**2.5/3.3**章节；

```
#初始化蓝牙，仅UART模块使用
/usr/bin/rtk_hciattach

#有没有去掉内核的CONFIG_BT_HCIUART配置，以及有没有生成对应的ko文件
/usr/lib/modules/hci_uart.ko

#USB接口是否有生成对应的usb驱动
/usr/lib/modules/rtk_btusb.ko
```

- 蓝牙的**Firmware/config** 文件不存在或着与蓝牙模组型号不匹配，参考**3.3**章节；

```
#正基海华模块  例如：AP6212A：对应的文件为bcm43438a1.hcd
/system(vendor)/etc/firmware/BCMxxxx.hcd

#RTL SDIO接口 RTL8723DS对应的文件
/lib/firmware/rtlbt/rtl8723d_fw
/lib/firmware/rtlbt/rtl8723d_config

#RTL USB接口  RTL8821CU对应的文件
/lib/firmware/rtl8821cu_fw
/lib/firmware/rtl8821cu_config
```

- BT UART RTS/CTS 硬件接法错误，导致初始化识别异常;

```
#正基、海华模块，4线都要跟主控连接
host tx  - controller rx
host rx  - controller tx
host rts - controller cts
host cts - controller rts

#Realtek模块
#对于直接使用Realtek蓝牙的COB芯片：UART接口的硬件接法如下：
host tx  - controller rx
host rx  - controller tx
host rts - controller cts
host cts - ground   #主控cts要接地
#For RTL8822C 这个芯片比较特殊，4线都要跟主控连接
host tx  - controller rx
host rx  - controller tx
host rts - controller cts
host cts - controller rts

#对于模组而言，模组厂一般会把controller rts内部接地，所以主控这边就不用接地，直接连接到
controller
#rts即可；特别注意：realtek有大量的代理商每家模组做的可能都不一样，所以要跟模组厂确认，如果
#controller rts没有接地，则需要主控接地。
host tx  - controller rx
host rx  - controller tx
host rts - controller cts
host cts - controller rts
```

- deviceio_release 编译错误，参考**10.2/3.5**章节配置章节！

- deviceio_test bluetooth异常，基本也是由于上面那几种问题导致；

```
#打开蓝牙时打印如下log
#hcd_file = /system/etc/firmware/BCM4343A1.hcd 这个文件是否跟蓝牙模组匹配
#ttys_dev = /dev/ttyS1 UART口是否跟硬件匹配
Which would you like: bt_test_bluetooth_init_thread: BT BLUETOOTH INIT
++++++++++ RK_BT_STATE_TURNING_ON ++++++++++
hcd_file = /system/etc/firmware/BCM4343A1.hcd
ttys_dev = /dev/ttyS1
killall: brcm_patchram_plus1: no process killed
killall: bsa_server: no process killed
/usr/bin/bsa_server.sh: line 41: check_not_exist.sh: not found
start broadcom bluetooth server bsa_sever
|----- bluetooth bsa server is open ------
```

8. 新模块移植或旧模块驱动更新

特别注意：

- 务必熟悉并理解第**3**章节的文件存放规则及编译规则，这对移植至关重要务；
- 客户拿到的**SDK**可能比较旧，所以**Config.in**和**rkwifibt.mk**的规则可能有变化但原理都是一样的；
- 对更新驱动的情况，下面有些步骤可以根据实际情况省略；

8.1 Realtek模块

8.1.1 Wi-Fi 部分

下面以RTL8821CS为例：

首先找模组厂或原厂拿到对应的移植包类似下面：

```
20171225_RTL8821CS_WiFi_linux_v5.2.18_25830_BT_ANDROID_UART_COEX_8821CS-
B1d1d_COEX20170908-1f1f
```

Wi-Fi驱动部分，进到如下目录：

```
WiFi\RTL8821CS_WiFi_linux_v5.2.18_25830_COEX20170908-1f1f.20171225\driver
```

- 添加对应编译配置（更新驱动的情况无需此操作）

拷贝驱动文件到drivers/net/wireless/rockchip_wlan/，并重命名为rtl8821cs，并修改Makefile/Kconfig添加对应配置：

```
drivers/net/wireless/rockchip_wlan/Makefile
+ obj-$(CONFIG_RTL8821CS) += rtl8821cs/

drivers/net/wireless/rockchip_wlan/Kconfig
+ source "drivers/net/wireless/rockchip_wlan/rtl8821cs/Kconfig"
```

- 修改Makefile

```
#改为RK平台：
CONFIG_PLATFORM_I386_PC = n
CONFIG_PLATFORM_ARM_RK3188 = y

#RK平台需要去掉以下配置：
ifeq ($(CONFIG_PLATFORM_ARM_RK3188), y)
-EXTRA_CFLAGS += -DRTW_ENABLE_WIFI_CONTROL_FUNC #删掉这个配置，如果有的话
#修改ko的名字，跟下面Buildroot config.in里面的BR2_PACKAGE_RKWIFIBT_WIFI_KO配置要保持一致！！！
MODULE_NAME := 8821cs

#如果有Wi-Fi休眠保持连接(WOWLAN)需求，打开如下配置：
CONFIG_WOWLAN = y
CONFIG_GPIO_WAKEUP = y
```

- 如果有WOWLAN需求，则增加WL_HOST_WAKE脚的irq获取函数

```
#修改 platform\platform_ops.c
+#include <linux/rfkill-wlan.h>
+extern unsigned int oob_irq;
int platform_wifi_power_on(void)
{
    int ret = 0;

+   oob_irq = rockchip_wifi_get_oob_irq();  //这里对应dts的WIFI_WAKE_HOST PIN脚

    return ret;
}
```

- 如果有自定义mac地址需求

```
#修改：core\rtw_ieee80211.c
+include <linux/rfkill-wlan.h> //添加头文件
#找到rtw_macaddr_cfg函数
void rtw_macaddr_cfg(u8 *out, const u8 *hw_mac_addr)
-/* Use the mac address stored in the Efuse */
-if (hw_mac_addr) {
-   rtw_memcpy(mac, hw_mac_addr, ETH_ALEN);
-   goto err_chk;
-}

+ /* Use the mac address stored in the Efuse */
+ if (hw_mac_addr) {
+   _rtw_memcpy(mac, hw_mac_addr, ETH_ALEN);
+ }

+ if (!rockchip_wifi_mac_addr(mac)) {
+   printk("get mac address from flash=[%02x:%02x:%02x:%02x:%02x:%02x]\n",
mac[0], mac[1],
+       mac[2], mac[3], mac[4], mac[5]);
+   }
+ }
```

- 增加驱动加载入口

```
/* SDIO接口：os_dep\linux\sdio_intf.c */
/* USB接口：  os_dep\linux\usb_intf.c */
/* PCIE接口：os_dep\linux\pci_intf.c */

//在该文件的最后加上如下代码：
#include "rtw_version.h"
#include <linux/rfkill-wlan.h>
extern int get_wifi_chip_type(void);
extern int rockchip_wifi_power(int on);
extern int rockchip_wifi_set_carddetect(int val);

int rockchip_wifi_init_module_rtkwifi(void)
{
    printk("\n");
    printk("=======================================================\n");
    printk("==== Launching Wi-Fi driver! (Powered by Rockchip) ====\n");
    printk("=======================================================\n");
    printk("Realtek 8XXX SDIO Wi-Fi driver (Powered by Rockchip,Ver %s) init.\n",
DRIVERVERSION);

    rockchip_wifi_power(1);
    rockchip_wifi_set_carddetect(1);

    return rtw_drv_entry();
}

void rockchip_wifi_exit_module_rtkWi-Fi(void)
{
    printk("\n");
    printk("=======================================================\n");
    printk("==== Dislaunching Wi-Fi driver! (Powered by Rockchip) ====\n");
    printk("=======================================================\n");
    printk("Realtek 8XXX SDIO Wi-Fi driver (Powered by Rockchip,Ver %s) init.\n",
DRIVERVERSION);

    rtw_drv_halt();

    rockchip_wifi_set_carddetect(0);
    rockchip_wifi_power(0);
}

//注意内核的配置，选择KO或者buildin对应的配置不一样
#ifdef CONFIG_WIFI_BUILD_MODULE
//KO模式走这里
module_init(rockchip_wifi_init_module_rtkwifi);
module_exit(rockchip_wifi_exit_module_rtkwifi);
#else
#ifdef CONFIG_WIFI_LOAD_DRIVER_WHEN_KERNEL_BOOTUP
//buildin模式走这里
late_initcall(rockchip_wifi_init_module_rtkwifi); //late_initcall延迟加载，等待文件
系统准备好
module_exit(rockchip_wifi_exit_module_rtkwifi);
#else
EXPORT_SYMBOL(rockchip_wifi_init_module_rtkwifi);
EXPORT_SYMBOL(rockchip_wifi_exit_module_rtkwifi);
#endif
#endif
```

```
//这里屏蔽掉下面，注意把下面两个函数的入口__init __exit 删掉
//module_init(rtw_drv_entry);
//module_exit(rtw_drv_halt);
```

- 4.4版本内核需添加型号的识别函数（更新驱动的情况无需此操作，且**4.19**内核无需添加下面的内容）

```
diff --git a/drivers/net/wireless/rockchip_wlan/wifi_sys/rkwifi_sys_iface.c
b/drivers/net/wireless/rockchip_wlan/wifi_sys/rkwifi_sys_iface.c
index 88db4de..2e3679a 100755
--- a/drivers/net/wireless/rockchip_wlan/wifi_sys/rkwifi_sys_iface.c
+++ b/drivers/net/wireless/rockchip_wlan/wifi_sys/rkwifi_sys_iface.c
@@ -133,6 +133,11 @@ static ssize_t wifi_chip_read(struct class *cls, struct
class_attribute *attr, c
             printk("Current wifi chip is RTL8189FS.\n");
         }
+       if(type == WIFI_RTL8821CS) {
+           count = sprintf(_buf, "%s", "RTL8821CS");
+           printk("Current Wi-Fi chip is RTL8821CS.\n");
+       }
+
     if(type == WIFI_ESP8089) {
         count = sprintf(_buf, "%s", "ESP8089");
         printk("Current Wi-Fi chip is ESP8089.\n");
diff --git a/include/linux/rfkill-wlan.h b/include/linux/rfkill-wlan.h
index 4218b84..698b685 100755
--- a/include/linux/rfkill-wlan.h
+++ b/include/linux/rfkill-wlan.h
@@ -73,6 +73,7 @@ enum {
    WIFI_RTL8189ES,
    WIFI_RTL8189FS,
    WIFI_RTL8812AU,
+   WIFI_RTL8821CS,
    WIFI_RTL_SERIES,
    WIFI_ESP8089,
    WIFI_MVL88W8977,
diff --git a/net/rfkill/rfkill-wlan.c b/net/rfkill/rfkill-wlan.c
index a17810d..7bbce01 100755
--- a/net/rfkill/rfkill-wlan.c
+++ b/net/rfkill/rfkill-wlan.c
@@ -156,6 +156,8 @@ int get_WIFI_chip_type(void)
         type = WIFI_RTL8189FS;
     } else if (strcmp(wifi_chip_type_string, "rtl8812au") == 0) {
         type = WIFI_RTL8812AU;
+   } else if (strcmp(wifi_chip_type_string, "rtl8821cs") == 0) {
+       type = WIFI_RTL8821CS;
     } else if (strcmp(wifi_chip_type_string, "esp8089") == 0) {
         type = WIFI_ESP8089;
     } else if (strcmp(wifi_chip_type_string, "mvl88w8977") == 0) {
```

- 可能的编译报错处理

```
rtl8xxx\os_dep\linux\rtw_android.c //屏蔽如下两条语句
#if (LINUX_VERSION_CODE >= KERNEL_VERSION(2, 6, 39)) ||
defined(COMPAT_KERNEL_RELEASE)
void *wifi_get_country_code(char *ccode)
{
    RTW_INFO("%s\n", __FUNCTION__);
    if (!ccode)
        return NULL;
-   if (wifi_control_data && wifi_control_data->get_country_code)
-       return wifi_control_data->get_country_code(ccode);
    return NULL;
}
#endif /* (LINUX_VERSION_CODE >= KERNEL_VERSION(2, 6, 39)) */
```

Buildroot系统部分（更新驱动的情况无需此操作）

```
buildroot\package\rockchip\rkwifibt\Config.in
依次添加如下内容：
+config BR2_PACKAGE_RKWIFIBT_RTL8821CS
+   bool "RTL8821CS"

config BR2_PACKAGE_RKWIFIBT_CHIPNAME
+   default "RTL8821CS" if BR2_PACKAGE_RKWIFIBT_RTL8821CS

config BR2_PACKAGE_RKWIFIBT_VENDOR
+   default "REALTEK" if BR2_PACKAGE_RKWIFIBT_RTL8821CS

config BR2_PACKAGE_RKWIFIBT_WIFI_KO
+   default "8821cs.ko" if BR2_PACKAGE_RKWIFIBT_RTL8821CS
#这个8821cs.ko名字跟驱动的Makefile里面的对应，make rkwifibt编译时会根据这个名字从内核驱动目录拷贝到文件系统里面去。
```

8.1.2 BT蓝牙部分

8.1.2.1 UART接口

如果对应的模组支持蓝牙，则需要找原厂提供类似如下软件包

```
20201130_ANDROID_BT_DRIVER_RTL8822C_COEX_v1c1c.tar.xx
```

下面以RTL8821CS为例（更新驱动的情况只需替换对应文件即可）：

```
#把蓝牙的firmware及配置文件，放置或更新到如下目录：
ls external/rkwifibt/realtek/
#结构如下：
external/rkwifibt/realtek/RTL8821CS #如果没有则创建新目录
├── rtl8821cs_config
└── rtl8821cs_fw

# 注意：RTL8821CS 目录的名字要跟 Buildroot rkwifibt Config.in里面的配置一致：
# BR2_PACKAGE_RKWIFIBT_"RTL8821CS"
```

Buildroot部分：使能蓝牙（更新驱动的情况无需此操作）

```
config BR2_PACKAGE_RKWIFIBT_BT_EN
+   default "ENABLE" if BR2_PACKAGE_RKWIFIBT_RTL8821CS
```

8.1.2.2 USB 接口

如果使用USB接口的蓝牙，则原厂会提供如下类似移植包，以RTL8821CU为例：

```
# tree
Linux_BT_USB_v3.10_20190430_8821CU_BTCOEX_20190509-4139.tar.xx
或
20201130_ANDROID_BT_DRIVER_RTL8821CU_COEX_v1c1c.tar.xx
|-8821CU                #firmware文件
|-bluetooth_usb_driver  #usb驱动

#拷贝或更新到external/rkwifibt/realtek/目录，结果如下：
# external/rkwifibt/realtek$ tree
├── RTL8821CU
│   ├── rtl8821cu_config    #蓝牙config
│   └── rtl8821cu_fw        #蓝牙fw
├── bluetooth_usb_driver    #RTL8821CU的蓝牙USB驱动，被编译成rtk_btusb.ko
│   ├── Makefile
│   ├── rtk_bt.c
│   ├── rtk_bt.h
│   ├── rtk_coex.c
│   ├── rtk_coex.h
│   ├── rtk_misc.c
│   └── rtk_misc.h
```

Buildroot系统部分（跟Wi-Fi部分是一样的，如果Wi-Fi有配置则无需再配置，注意名字匹配）：

```
buildroot\package\rockchip\rkwifibt\Config.in
依次添加如下内容：
+config BR2_PACKAGE_RKWIFIBT_RTL8821CU
+   bool "RTL8821CU"

config BR2_PACKAGE_RKWIFIBT_CHIPNAME
+   default "RTL8821CU" if BR2_PACKAGE_RKWIFIBT_RTL8821CU

config BR2_PACKAGE_RKWIFIBT_VENDOR
+   default "REALTEK" if BR2_PACKAGE_RKWIFIBT_RTL8821CU

config BR2_PACKAGE_RKWIFIBT_WIFI_KO
+   default "8821cu.ko" if BR2_PACKAGE_RKWIFIBT_RTL8821CU

config BR2_PACKAGE_RKWIFIBT_BT_EN
+   default "ENABLE" if BR2_PACKAGE_RKWIFIBT_RTL8821CU
```

在rkwifibt.mk里面增加编译ko选项及安装install命令，如下：

```
+++ b/package/rockchip/rkwifibt/rkwifibt.mk
@@ -73,6 +73,7 @@ define RKWIFIBT_REALTEK_BT_INSTALL
```

```
    $(INSTALL) -D -m 0644 $(@D)/realtek/$(BR2_PACKAGE_RKWIFIBT_CHIPNAME)/mp_*
$(TARGET_DIR)/lib/firmware/

#USB接口蓝牙需要把firmware文件拷贝到/lib/firmware/目录
+   $(INSTALL) -D -m 0644 $(@D)/realtek/$(BR2_PACKAGE_RKWIFIBT_CHIPNAME)/
$(TARGET_DIR)/lib/firmware/
    $(INSTALL) -D -m 0755 $(@D)/bt_realtek* $(TARGET_DIR)/usr/bin/
    $(INSTALL) -D -m 0644 $(@D)/realtek/bluetooth_uart_driver/hci_uart.ko
$(TARGET_DIR)/usr/lib/modules/hci_uart.ko

#拷贝USB接口的ko到指定目录usr/lib/modules/
+   $(INSTALL) -D -m 0644 $(@D)/realtek/bluetooth_usb_driver/rtk_btusb.ko
$(TARGET_DIR)/usr/lib/modules/rtk_btusb.ko
    $(INSTALL) -D -m 0755 $(@D)/bt_load_rtk_firmware $(TARGET_DIR)/usr/bin/
    $(SED) 's/BT_TTY_DEV/\/dev\/$(BT_TTY_DEV)/g'
$(TARGET_DIR)/usr/bin/bt_load_rtk_firmware
    $(INSTALL) -D -m 0755 $(TARGET_DIR)/usr/bin/bt_load_rtk_firmware
$(TARGET_DIR)/usr/bin/bt_pcba_test
@@ -92,8 +93,10 @@ define RKWIFIBT_BUILD_CMDS
    $(TARGET_CC) -o $(@D)/brcm_tools/brcm_patchram_plus1
$(@D)/brcm_tools/brcm_patchram_plus1.c
    $(TARGET_CC) -o $(@D)/brcm_tools/dhd_priv $(@D)/brcm_tools/dhd_priv.c
    $(TARGET_CC) -o $(@D)/src/rk_wifi_init $(@D)/src/rk_wifi_init.c
    $(MAKE) -C $(@D)/realtek/rtk_hciattach/ CC=$(TARGET_CC)
    $(TARGET_CONFIGURE_OPTS) $(MAKE) -C $(TOPDIR)/../kernel/
M=$(@D)/realtek/bluetooth_uart_driver ARCH=$(RK_ARCH)

#编译USB接口的ko，注意下面的ARCH=$(RK_ARCH)，不同SDK可能不一样，参考上面一行的写法
+   $(TARGET_CONFIGURE_OPTS) $(MAKE) -C $(TOPDIR)/../kernel/
M=$(@D)/realtek/bluetooth_usb_driver ARCH=$(RK_ARCH)
```

8.2 AP正基模组

下面以AP6256为例，找模组厂要AP6256的Wi-Fi和BT的firmware文件包（更新驱动的情况只需替换对应文件即可）

```
external\rkwifibt\firmware\broadcom\
//在该目录下创建名为AP6256的文件夹，并把里面的文件按照如下结构存放
external/rkwifibt/firmware/broadcom/AP6256$ tree
├── bt
│   └── BCM4345C5.hcd
└── wifi
    ├── fw_bcm43456c5_ag.bin
    ├── fw_bcm43456c5_ag_mfg.bin
    └── nvram_ap6256.txt
//注意 AP6256目录名字，要跟下面wifi的Config.in里面的配置一致：
BR2_PACKAGE_RKWIFIBT_AP6256
```

Buildroot系统配置（更新驱动的情况无需此操作）

```
buildroot\package\rockchip\rkwifibt\Config.in
依次添加如下内容：
+config BR2_PACKAGE_RKWIFIBT_AP6256
+   bool "AP6256"
```

```
config BR2_PACKAGE_RKWIFIBT_CHIPNAME
+   default "AP6256" if BR2_PACKAGE_RKWIFIBT_AP6256

config BR2_PACKAGE_RKWIFIBT_VENDOR
+   default "BROADCOM" if BR2_PACKAGE_RKWIFIBT_AP6256

config BR2_PACKAGE_RKWIFIBT_WIFI_KO
+   default "bcmdhd.ko" if BR2_PACKAGE_RKWIFIBT_AP6256

config BR2_PACKAGE_RKWIFIBT_BT_FW
+   default "BCM4345C5.hcd" if BR2_PACKAGE_RKWIFIBT_AP6256  //"BCM4345C5.hcd"注意
这个名字，要改成对应型号的

config BR2_PACKAGE_RKWIFIBT_BT_EN
+   default "ENABLE" if BR2_PACKAGE_RKWIFIBT_AP6256
```

内核驱动部分无需改动，基本兼容所有**AP**模组，默认都使用**CONFIG_AP6XXX**的配置即可。

8.3 第三方Wi-Fi驱动移植指导

针对SDK不支持的WiFi模块，可参考如下说明进行porting：

DTS配置参考**2.2**章节，主要配置的是：

- WIFI_REG_ON：WIFI模块的电源脚；
- WIFI_WAKE_HOST：WiFi模块的唤醒主控的脚（可选，一般Broadcom/Realtek模块使用）；

以上配置的GPIO会被kernel的net/rfkill/rfkill-wlan.c解析，并给WIFI驱动提供如下接口：

```
/* SDIO 扫卡函数 */
int rockchip_wifi_set_carddetect(int val)

/* WiFi驱动获取唤醒主控的中断的GPIO口，可选 */
int rockchip_wifi_get_oob_irq(void)
/* 中断的触发电平，跟上面的API配合使用 */
int rockchip_wifi_get_oob_irq_flag(void)

/* WIFI的上下电函数 */
int rockchip_wifi_power(int on)
```

WiFi驱动根据实际情况调用上面的接口即可；

驱动的编译：需指定kernel目录及对应的RK编译器：

```
#下面的变量名根据实际情况进行修改：
diff --git a/Makefile b/Makefile
index 649da3c..d0b128d 100644
--- a/Makefile
+++ b/Makefile

+KDIR=/home/rk/kernel/
+ARCH=arm or arm64
+CROSS_COMPILE=SDK/prebuilts/gcc/linux-x86/arm/gcc-linaro-6.3.1-2017.05-
x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-
```

9. Debian及其他第三方系统Wi-Fi/BT适配说明

9.1 系统适配概述

- **Debian/麒麟/UOS/统信系统**

这类系统上层都有完整的Wi-Fi/BT应用(NetworkManager/blumeman/gnome-bluetooth)，我们的工作只需在系统启动前把接口初始化好即可；比如，对于Wi-Fi： ifconfig 可以看到wlan0节点；对于蓝牙：hciconfig可以看到hci0节点；注意**Wi-Fi/BT**的**DTS**/内核配置是跟具体的系统无关，它们是通用配置，都直接参照第**2**章节进行基础**DTS**及内核配置，下面章节会对接口初始化进行说明：

- 对于**Yocto** 等类**Buildroot**系统

这类系统如果也有上层WiFiBT的应用，则流程跟上面一致；如果没有：

对于**Wi-Fi**：一般都使用**wpa_supplicant**应用做WiFi的扫描连接等无线管理，且这类系统基本都有包wpa_supplicant应用包，在系统里面选择相应配置并编译，然后参考第4.1章节进行基础的验证；

对于蓝牙：一般都使用**bluez5**协议栈做蓝牙的扫描连接等基本操作，且这类系统基本都有包含bluez5应用包，在系统里面选择相应配置并编译，启动测试步骤：

```
# 编译完成后会生成 bluetoothd/bluetoothctl/hciconfig 等二进制
$ bluetoothd -ndC & # 启动 bluetoothd
$ bluetoothctl      # 进入交换模式
[bluetooth]# power on
[bluetooth]# scan on
[bluetooth]# help   # 查看更多命令
```

Note：上面提到的应用工具及编译请参考对应系统的文档及说明。

9.2 正基模块适配示例

```
### 下面以AP6275P为例，三个文件找正基模组厂获取
# 蓝牙初始化文件：brcm_patchram_plus1.c，然后用系统使用的编译器编译成可执行文件
brcm_patchram_plus1并放到系统里面去
external/rkwifibt/brcm_tools/brcm_patchram_plus1.c # 如果有RKSDK，可以从这个目录获取

# BT firmware文件：根据实际使用系统的需求存放，没有特殊要求，下面的brcm_patchram_plus1蓝牙
初始化程序会要求指定firmware的路径；
BCM4362A2.hcd

# Wi-Fi firmware文件：根据实际使用系统的需求存放
clm_bcm43752a2_pcie_ag.blob
fw_bcm43752a2_pcie_ag.bin
fw_bcm43752a2_pcie_ag_apsta.bin
nvram_AP6275P.txt

### 配置
# 检查内核Wi-Fi配置，打开如下几个配置：
CONFIG_WL_ROCKCHIP=y
CONFIG_WIFI_BUILD_MODULE=y
CONFIG_BCMDHD=y
CONFIG_AP6XXX=m
```

```
CONFIG_BCMDHD_PCIE=y #PCIE接口，与SDIO互斥，不是PCIE可不配
CONFIG_BCMDHD_SDIO=y #SDIO接口，与PCIE互斥

### Wi-Fi接口初始化
# make 编译完会生成ko，这个文件根据你们实际需求存放到对应位置，打开Wi-Fi加载这个ko即可；
drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/bcmdhd.ko

# 打开Wi-Fi：需先加载ko，并在insmod时参数指定firmware/nvram的路径，下面xx_path改成实际使用的：
insmod /ko_path/bcmdhd.ko firmware_path=/fw_path/ nvram_path=/nvram_path/
ifconfig -a #正常的话可以看到wlan0，如果没有参考第2章及第7章节进行排查

### 蓝牙接口初始化
# 打开蓝牙，先复位BT电源：
echo 0 > /sys/class/rfkill/rfkill0/state    #关闭BT电源，等同于rfkill block操作
sleep 0.2
echo 1 > /sys/class/rfkill/rfkill0/state    #打开BT电源，等同于rfkill unblock操作
sleep 0.2
# 初始化蓝牙命令，--patchram指定蓝牙firmware文件的路径(根据实际情况修改)，/dev/ttyS8是对应硬件的串口号(根据实际情况修改)
brcm_patchram_plus1 --bd_addr_rand --enable_hci --no2bytes --
use_baudrate_for_download  --tosleep  200000 --baudrate 1500000 --patchram
/system/etc/firmware/BCM4362A2.hcd /dev/ttyS8 &
# 如果系统有安装bluez协议栈，则使用hciconfig指令
hciconfig -a #可以看到hci0节点，如果没有参考第2章及第7章节进行排查

# 关闭蓝牙：
echo 0 > /sys/class/rfkill/rfkill0/state    #关闭BT电源，等同于rfkill block操作
killall brcm_patchram_plus1                 #务必要杀掉brcm_patchram_plus1进程，因为打开时会再次执行，否则会冲突；

#以上打开关闭操作根据实际情况移植到你们系统中去；

# 注意：如果应用层开关蓝牙有调用到rfkill block关闭蓝牙电源，则再次unblock打开蓝牙电源时，必须再次执行brcm_patchram_plus1蓝牙初始化命令，否则蓝牙无法使用；如果上层仅是hciconfig hci0 down/up，则无需调用重复初始化；
```

9.3 Realtek模块适配示例

9.3.1 适配说明

```
### 下面以RTL8822CS为例，首先找模组厂获取对应模块的驱动包
# Wi-Fi的：RTL8822CS_WiFi_linux_v5.12.1.5-1-g0e1519e_COEX20210504-2323.20210527
# BT的：20201202_LINUX_BT_DRIVER_RTL8822C_COEX_v1c1c

### Wi-Fi适配
# 参考8.1移植章节适配Wi-Fi驱动到RK平台，并参考第2章节进行基础的dts及内核配置
# make 编译完会生成ko，这个文件根据你们实际需求存放到对应位置，打开Wi-Fi加载这个ko即可；
drivers/net/wireless/rockchip_wlan/rkwifi/rtl8822cs/8822cs.ko

insmod /ko_path/88xxxx.ko # realtek无需firmware/nvram文件，insmod执行时机根据系统要求进行调整；
ifconfig -a #正常的话可以看到wlan0，如果没有参考第2章及第7章节进行排查
```

```
### 蓝牙适配
# fw/config 文件说明：
# 只有蓝牙才需要fw/config(文件从驱动包里面找)文件，存放位置跟接口有关
# RTL UART接口，RTL8822CS对应的文件放到如下位置
/lib/firmware/rtlbt/rtl8822cs_fw
/lib/firmware/rtlbt/rtl8822cs_config
# Copy the right FW file and config file to the correct path. (拷贝
firmware/config文件)
$ sudo mkdir -p /lib/firmware/rtlbt/
$ sudo cp rtkbt-firmware/lib/firmware/rtlbt/rtl8xxxx_fw /lib/firmware/rtlbt/
$ sudo cp rtkbt-firmware/lib/firmware/rtlbt/rtl8xxxx_config /lib/firmware/rtlbt/

# RTL USB接口  RTL8822CU对应的文件（把对应的fw/config文件拷贝到系统对应的位置）
/lib/firmware/rtl8822cu_fw
/lib/firmware/rtl8822cu_config
# Copy the right FW file and config file to the correct path.
$ sudo cp rtkbt-firmware/lib/firmware/rtl8xxxxx_fw /lib/firmware/
$ sudo cp rtkbt-firmware/lib/firmware/rtl8xxxxx_config /lib/firmware/

# rtk_hciattach/hci_uart/usb.ko文件说明
# 参考9.2.2章节工具编译说明

# hci_uart/usb.ko文件说明，realtek不使用内核自带的接口驱动，内核必须先去掉如下两个配置：
CONFIG_BT_HCIBTUSB
CONFIG_BT_HCIUART

### 初始化说明
# UART 接口：
killall rtk_hciattach #首先必须确保先关掉此进程(如果之前有打开)
echo 0 > /sys/class/rfkill/rfkill0/state #下电
sleep 0.5
echo 1 > /sys/class/rfkill/rfkill0/state #上电
sleep 0.5
insmod /usr/lib/modules/hci_uart.ko              # realtek模组需要加载uart驱动
rtk_hciattach -n -s 115200 /dev/ttyS4 rtk_h5 &   # ttySX指的是蓝牙使用哪个uart口
# 如果系统有安装bluez协议栈，则使用hciconfig指令
hciconfig -a #正常的话可以看到hci0节点，如果没有参考第2章及第7章节进行排查

# USB 接口：
echo 0 > /sys/class/rfkill/rfkill0/state #下电
sleep 0.5
echo 1 > /sys/class/rfkill/rfkill0/state #上电
sleep 0.5
insmod /usr/lib/modules/rtk_btusb.ko             # realtek模组需要加载usb驱动
# 如果系统有安装bluez协议栈，则使用hciconfig指令
hciconfig -a #正常的话可以看到hci0节点，如果没有参考第2章及第7章节进行排查
```

9.3.2 蓝牙驱动/rtk_hciattach工具编译说明

```
### Realtek UART/USB 蓝牙驱动 ko驱动编译：
$ make -C /home/rk/rk3xxx/kernel/
 CROSS_COMPILE=/home/rk/rk3xxx/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-
2021.07-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu- ARCH=arm64
M=/home/rk/rk3xxx/usb(uart)/bluetooth_usb(uart)_driver/
# -C 指定kernel目录
# CROSS_COMPILE 指定交叉编译工具链路径
# ARCH 指定系统平台
# M 指定uart/usb driver路径
# 注意路径必须为绝对路径
# 编译成功后会生成

# rtk_hciattach UART 初始化程序编译：
$ make CROSS_COMPILE=/home/rk/rk3xxx/prebuilts/gcc/linux-x86/aarch64/gcc-arm-
10.3-2021.07-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu- -C
/home/rk/rk3xxx/uart/rtk_hciattach/
# -C 指定kernel目录
# CROSS_COMPILE 指定交叉编译工具链路径
```

9.4 自动安装说明

上面的适配说明都是通过手动编译或者push等操作来完成，当调试完成后如何简化步骤自动安装，建议如下：

- 如果是成熟的商业系统：比如UOS/麒麟/统信，他们有自己的一套方法，请咨询系统提供商；
- 如果是使用免费的Debian，则需要把上述提到的各种文件制作成deb包，进行安装使用；制作deb包及安装文档参考: https://www.debian.org/doc/
- 如果使用Yocto类Buildroot系统，则参考对应系统的使用的编译规则及方法；

10. 蓝牙拓展功能

10.1 蓝牙低功耗

目前仅针对CY模块+BSA协议栈，AP模块在验证中，Realtek模块默认支持；实现效果：打开BT时功耗从2~3mA 左右降低到0.5mA左右；

```
# 修改脚本添加lpm参数，bsa版本需更新
external/broadcom_bsa/bsa_server.sh
bsa_server -r 12 -pp $hcd_file -d $ttys_dev -all=0 -lpm &

# 内核需更新rfkill-bt.c文件
# 以上文件可通过redmine获取；
```

10.2 蓝牙5.0功能验证(目前仅RK3588平台支持，后续支持更多平台)

- PHY选择: 如果选用支持BT5.0的模块，则有以下PHY可以选择：在速率及传输距离都有很大提升。

PHY	Data Rate	Theoretical Relative Range	Advantage
2 Mbps	2 Mb/s	~0.8x	Speed (Throughput)
1 Mbps	1 Mb/s	1x	Compatilbility (balanced)
Coded S2	500 kbps	2x	Range
Coded S8	125 kbps	4x	Range x2

```
+/* 根据实际需求进行修改：*/
diff --git a/net/bluetooth/hci_core.c b/net/bluetooth/hci_core.c
index 2ad66f64879f..a80d921c66ed 100644
--- a/net/bluetooth/hci_core.c
+++ b/net/bluetooth/hci_core.c
@@ -3632,8 +3632,8 @@ struct hci_dev *hci_alloc_dev(void)
    hdev->le_max_rx_time = 0x0148;
    hdev->le_max_key_size = SMP_MAX_ENC_KEY_SIZE;
    hdev->le_min_key_size = SMP_MIN_ENC_KEY_SIZE;
-   hdev->le_tx_def_phys = HCI_LE_SET_PHY_1M;
-   hdev->le_rx_def_phys = HCI_LE_SET_PHY_1M;
//2M
+   hdev->le_tx_def_phys = HCI_LE_SET_PHY_2M;
+   hdev->le_rx_def_phys = HCI_LE_SET_PHY_2M;
//CODED
+   hdev->le_tx_def_phys = HCI_LE_SET_PHY_CODED;
+   hdev->le_rx_def_phys = HCI_LE_SET_PHY_CODED;
    hdev->le_num_of_adv_sets = HCI_MAX_ADV_INSTANCES;
    hdev->def_multi_adv_rotation_duration = HCI_DEFAULT_ADV_DURATION;
    hdev->def_le_autoconnect_timeout = HCI_LE_AUTOCONN_TIMEOUT;
```

- 扩展广播，支持两个新特性：

  广播数据量：从原来的31byte 增大到 251byte；

  同时进行多个广播：4.2只能同一时间广播一个，5.0最大支持6个广播同时进行；

```
# 测试：
# 广播数据量测试：
$ btmgmt
> privacy on
> power on
> add-adv -u 180d -u 180f -d
240954657374204C4554657374204C4554657374204C4554657374204C4554657374204C45 -c -P
1M 1

# 多个广播实列测试
$ btmgmt
> privacy on
> power on
> add-adv -u 180d -u 180f -d 080954657374204C45 -c -P 1M 1
> add-adv -u 180d -u 180f -d 080954657374204C46 -c -P 1M 2
```

```
> add-adv -u 180d -u 180f -d 080954657374204C47 -c -P 1M 3
> add-adv -u 180d -u 180f -d 080954657374204C48 -c -P 1M 3

# 同时用支持5.0的手机安装nrf connect apk，可以看到多个广播实列；
```

内核打上如下补丁：

```
diff --git a/net/bluetooth/hci_request.c b/net/bluetooth/hci_request.c
index 33dc78c24b73..6194fdcbad86 100644
--- a/net/bluetooth/hci_request.c
+++ b/net/bluetooth/hci_request.c
@@ -2054,7 +2060,8 @@ int __hci_req_setup_ext_adv_instance(struct hci_request
*req, u8 instance)

    hci_req_add(req, HCI_OP_LE_SET_EXT_ADV_PARAMS, sizeof(cp), &cp);

-   if (own_addr_type == ADDR_LE_DEV_RANDOM &&
+   if ((own_addr_type == ADDR_LE_DEV_RANDOM ||
+        own_addr_type == ADDR_LE_DEV_RANDOM_RESOLVED) &&
            bacmp(&random_addr, BDADDR_ANY)) {
```

11. 其他杂项功能及配置说明

11.1 RV1126/1109 Connmand

connmand控制Wi-Fi (不推荐，已经没有维护)

RV1109/1126平台默认使用connman管理Wi-Fi，而且Wi-Fi的核心进程wpa_supplicant的启动方法由它启动：

```
# ps
//可以看到如下两个进程
conmand //它使用dbus跟wpa_supplicant进行通信
wpa_supplicant -u //打开支持dbus通信
```

标准使用方法：通过**1109**的**web**界面进行**Wi-Fi**操作，参考**1109/1126**平台的相关文档：

```
docs/RV1126_RV1109/ApplicationNote/Rockchip_Instructions_Linux_Web_Configuration_
CN.pdf
```

终端简单测试方法如下：

```
/ # killall ipc-daemon netserver #kill掉上层冲突应用
/ # connmanctl
connmanctl> enable wifi
connmanctl> scan wifi    #可以多次扫描
connmanctl> scan wifi    #可以多次扫描
connmanctl> agent on
connmanctl> services  #列出扫描到的Wi-Fi列表
```

```
connmanctl>              *AO yyz123
wifi_c0847daf6f42_79797a313233_managed_psk
    NETGEAR75-5G          wifi_c0847daf6f42_4e45544745415237352d3547_managed_psk
    aaabbb                wifi_c0847daf6f42_616161626262_managed_psk
    Hiwifi-Free          wifi_c0847daf6f42_204869576946692d46726565_managed_none
    Fang-Hiwifi           wifi_c0847daf6f42_46616e672d486957694669_managed_psk
    yyz123                wifi_c0847daf6f42_79797a313233_managed_psk

connmanctl> connect wifi_c0847daf6f42_79797a313233_managed_psk     #假如要连接上面
yyz123，则connect的参数为后面的wifi_xxx_psk
Connected wifi_c0847daf6f42_79797a313233_managed_psk #如果连接成功则会有这个打印
connmanctl> quit #退出连接模式
/ # ifconfig wlan0  #可以看到ip地址
```

如果不用**connman**，而使用传统的**wpa_supplicant/wpa_cli**的方式，则**Buildroot**去掉**connman**的配置：

```
BR2_PACKAGE_CONNMAN
```

且删掉之前生成的相关文件：

```
buildroot/output/rockchip_rv1126_rv1109(根据实际情况调整目录
名)/target/etc/init.d/S45connman
buildroot/output/rockchip_rv1126_rv1109/target/usr/bin/connmanctl
buildroot/output/rockchip_rv1126_rv1109/target/usr/sbin/connmand
```

步骤参考第4.1/2章节进行之前旧的方式的开发。

AP热点功能：

wifi 默认作为主网卡

```
# 下载https://github.com/rockchip-linux/softapdemo到external目录
# 88端口可以改为任意跟你们应用不冲突的即可，其他定制化修改才开wifibt开发文档的3.2章节
external/softapDemo$ git diff
diff --git a/src/main.c b/src/main.c
index 0aad306..d713be3 100644
--- a/src/main.c
+++ b/src/main.c
@@ -170,7 +170,7 @@ int wlan_accesspoint_start(const char* ssid, const char*
password)
        console_run(cmdline);
    }
    memset(cmdline, 0, sizeof(cmdline));
-   sprintf(cmdline, "dnsmasq -C %s --interface=%s", DNSMASQ_CONF_DIR,
softap_name);
+   sprintf(cmdline, "dnsmasq -p 88 -C %s --interface=%s", DNSMASQ_CONF_DIR,
softap_name);
    console_run(cmdline);
```

编译运行

```
$ make menuconfig
选上
$ make savedefconfig
```

```
$ make softap
$ ./build.sh

# 开机执行
//关闭以太网
dbus-send --system --print-reply --dest=rockchip.dbserver /
rockchip.dbserver.net.Cmd \string:"{ \"table\": \"NetworkPower\",\"key\":
{\"sType\": \"ethernet\"}, \"data\": {\"iPower\": 0}, \"cmd\": \"Update\" }"

//打开Wi-Fi
dbus-send --system --print-reply --dest=rockchip.dbserver /
rockchip.dbserver.net.Cmd \string:"{ \"table\": \"NetworkPower\",\"key\":
{\"sType\": \"wifi\"}, \"data\": {\"iPower\": 1}, \"cmd\": \"Update\" }"

//执行softapDemo:
$ softapDemo AP_NAME

手机搜索AP_NAME，并连接，然后打开浏览器输入：192.168.88.1即可访问web
```

11.2 开机自动设置静态IP等参数

```
//Buildroot系统默认标准的Linux ifupdown命令，对应的启动脚本在如下目录：
buildroot/package/ifupdown-scripts
> S40network //由/etc/init.d/rcS启动
//而S40network会调用ifup命令去读取默认的配置脚本：/etc/network/interfaces
//所以可以把相关的默认配置写到/etc/network/interfaces文件里面，而修改interfaces文件方法如
下：
buildroot/package/ifupdown-scripts/ifupdown-scripts.mk
define IFUPDOWN_SCRIPTS_LOCALHOST
( \
    echo "# interface file auto-generated by buildroot"; \
    echo ; \
    echo "auto lo"; \
    echo "iface lo inet loopback"; \
    echo "auto wlan0"; \
    echo "iface wlan0 inet static"; \
    echo "address 192.168.1.111"; \
    echo "gateway 192.168.1.1"; \
    echo "netmask 255.255.255.0"; \
    echo "broadcast 192.168.1.0"; \
) > $(TARGET_DIR)/etc/network/interfaces  // 需要添加更多配置，可请自行查询相关资料，都
是linux标准的
endef
//编译升级：make ifupdown-scripts-dirclean && make ifupdown-scripts-rebuild

//对于默认dns的修改，Buildroot系统没有预编译配置，在没有运行DHCP进程的情况下，只能手动去添
加：
echo 'nameserver 114.114.114.114' >> /etc/resolv.conf // 添加定制的dns配置

//动态设置
//设置ip地址
ifconfig wlan0 xx.xx.xx.xx netmask xx.xx.xx.xx
//设置默认路由
route add default gw xx.xx.xx.xx
//添加dns
```

```
echo 'nameserver xx.xx.xx.xx' > /etc/resolv.conf
```

11.3 DHCP客户端

dhcpcd： SDK默认使用，随系统启动而启动，它是功能比较完整的dhcp客户端；

udhcpc：是busybox的精简的dhcp客户端；

注意：这两个进程一定不要同时启用，只能使用其中一个！

使用dhcpcd客户端，如果需要加快获取IP地址的速度，改动如下：

```
#修改S41dhcpcd文件
index a2e87ca054..f8b924ab0f 100755
/buildroot/package/dhcpcd/S41dhcpcd
@@ -13,7 +13,7 @@ PIDFILE=/var/run/dhcpcd.pid
 case "$1" in
    start)
        echo "Starting dhcpcd..."
-       start-stop-daemon -S -x "$DAEMON" -p "$PIDFILE" -- -f "$CONFIG"
+       start-stop-daemon -S -x "$DAEMON" -p "$PIDFILE" -- -AL -f "$CONFIG"
        ;;

//重新打包固件
make dhcpcd-dirclean
make dhcpcd-rebuild
```

11.4 Wi-Fi/BTMAC地址

一般情况下wifibt的mac地址都是芯片内置的，如果需要自定义mac地址，需要使用RK专用工具写到Flash自定义的vendor分区(方法请参考相应文档，这里不做说明)；

正基/海华Wi-Fi模组：

修改 Makefile，增加如下配置：

```
+    -DGET_CUSTOM_MAC_ENABLE
-    -DGET_OTP_MAC_ENABLE #如果有的话

正基：drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/Makefile
海华：drivers/net/wireless/rockchip_wlan/cywdhd/bcmdhd/Makefile
```

正基模组需要额外的修改**Wi-Fi**才能正常工作，**MAC**地址的前**3**个字节表示称为**OUI**，每一个**OUI**对应一组**macpad**，如果要修改**OUI**则需向正基申请对应的**macpad**，然后修改如下：

注意：正基新的firmware已去掉此限制，可以找正基要最新的firmware。

```
drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/dhd_gpio.c
static int dhd_wlan_get_mac_addr(unsigned char *buf)
{
    int err = 0;
    printf("======== %s ========\n", __FUNCTION__);
#ifdef EXAMPLE_GET_MAC
```

```
    /* EXAMPLE code */
    {
        struct ether_addr ea_example = {{0x00, 0x11, 0x22, 0x33, 0x44, 0xFF}};
        bcopy((char *)&ea_example, buf, sizeof(struct ether_addr));
    }
#endif /* EXAMPLE_GET_MAC */

    //方法一：如果使用我们的vendor方案，把自定义的mac要烧录到vendor分区，下面函数会从vendor
分区读取.
    err = rockchip_wifi_mac_addr(buf);
    //这个函数的目的就是把 mac 地址的填充在 buf 的前 6 个位置，参考上面 ea_example.

    // 方法二：如果 mac 地址存放在你们自定义的位置，则需要自行实现读取函数
    // TODO: 把 mac 地址的填充在 buf 的前 6 个位置，参考上面 ea_example

    //#ifdef EXAMPLE_GET_MAC_VER2 //定义或屏蔽掉这个宏，使下面代码生效
    /* EXAMPLE code */
    {
        char macpad[56]= { //这里替换成原厂提供的 macpad
            0x43,0xdf,0x6c,0xb3,0x06,0x3e,0x8e,0x94,
            0xc7,0xa9,0xd3,0x41,0xc8,0x6f,0xef,0x67,
            0x05,0x30,0xf1,0xeb,0x4b,0xa9,0x0a,0x05,
            0x41,0x73,0xbc,0x8c,0x30,0xe5,0x74,0xc6,
            0x88,0x36,0xad,0x0c,0x34,0x7d,0x5b,0x60,
            0xe7,0xd7,0x98,0x64,0xd0,0xfa,0xe3,0x83,
            0x76,0x35,0x1a,0xc8,0x2b,0x0b,0x65,0xb1};
        bcopy(macpad, buf+6, sizeof(macpad));
    }
    //#endif /* EXAMPLE_GET_MAC_VER2 */
    return err;
}
```

11.5 正基模组兼容版本(Debian/Ubuntu)

SDK提供一个正基常用芯片的兼容配置Buildroot rkwifibt：

```
BR2_PACKAGE_RKWIFIBT_AMPAKALL
```

这个配置可以兼容SDK支持的模组，它开机自动判断Wi-Fi/BT芯片型号并加载对应firmware，源码目录为：

```
external/rkwifibt/src/rk_wifi_init.c
```

注意几点：

- 此配置会拷贝所有正基模组的firmware导致rootfs文件系统变大，需要更大的flash；
- 此配置不支持bsa（正基蓝牙私有协议栈）兼容，导致bsa无法使用；

所以只建议使用**Debian/Ubuntu**系统使用此配置，这些开源系统都有自己的蓝牙协议栈，我们只需完成初始化即可（**rk_wifi_init**）。

11.6 修改Realtek Wi-Fi的扫描时间

```
//realtek wifi scan在每个信道停留时间，修改include/rtw_mlme_ext.h
#define SURVEY_TO                   (100)   //单位是ms, 按需求修改
```

11.7 Wi-Fi 国家码

Realtek芯片：修改驱动Makefile, 在平台编译项加入：

```
+++ b/drivers/net/wireless/rockchip_wlan/rtlxxx/Makefile
@@ -1270,6 +1270,7 @@ EXTRA_CFLAGS += -DCONFIG_LITTLE_ENDIAN -
DCONFIG_PLATFORM_ANDROID -DCONFIG_PLATFO
 # default setting for Android 4.1, 4.2, 4.3, 4.4
 EXTRA_CFLAGS += -DCONFIG_IOCTL_CFG80211 -DRTW_USE_CFG80211_STA_EVENT
 EXTRA_CFLAGS += -DCONFIG_CONCURRENT_MODE
+EXTRA_CFLAGS += -DCONFIG_RTW_IOCTL_SET_COUNTRY
 # default setting for Power control
 #EXTRA_CFLAGS += -DRTW_ENABLE_WIFI_CONTROL_FUNC
 #EXTRA_CFLAGS += -DRTW_SUPPORT_PLATFORM_SHUTDOWN
```

1. 透过proc方式`echo X > /proc/net/rtlxxx/wlan0/country_code`, 如: `echo CN > /proc/net/rtlxxx/wlan0/country_code`
2. wpa_supplicant.conf 配置参数 country=X，如果是softap，则hostapd.conf配置参数country_code=X
   注：如何确认country code X 可通过网址查询，比如https://countrycode.org/ 看ISO CODES 两位大写字母组合；

正基/海华芯片：

`dhd_priv country XX`

11.8 Wi-Fi 动态加载卸载KO模式

如果在Wi-Fi编译为ko模式下，有多次加载/卸载操作的情况下，需要打上下面的patch

```
--- a/arch/arm64/boot/dts/rockchip/rk3xxx.dts
+++ b/arch/arm64/boot/dts/rockchip/rk3xxx.dts
@@ -112,6 +112,7 @@
    wireless-wlan {
        rockchip,grf = <&grf>;
        wifi_chip_type = "ap6354";
        sdio_vref = <1800>;
+       WIFI,poweren_gpio = <&gpio1 18 GPIO_ACTIVE_HIGH>;  //配置对应WIFI_REG_ON的
PIN
        WIFI,host_wake_irq = <&gpio1 19 GPIO_ACTIVE_HIGH>;
        status = "okay";
    };

sdio_pwrseq: sdio-pwrseq {
+   status = "disabled"; //关闭这个节点
};
```

```
&sdio { //去掉下面两个配置
    disable-wp;
    keep-power-in-suspend;
    max-frequency = <150000000>;
-   mmc-pwrseq = <&sdio_pwrseq>;
-   non-removable;
    num-slots = <1>;
}

diff --git a/drivers/mmc/host/dw_mmc.c b/drivers/mmc/host/dw_mmc.c
index 8730e2e..04b9cb8 100644
--- a/drivers/mmc/host/dw_mmc.c
+++ b/drivers/mmc/host/dw_mmc.c
@@ -1518,6 +1518,9 @@ static int dw_mci_get_cd(struct mmc_host *mmc)
    struct dw_mci *host = slot->host;
    int gpio_cd = mmc_gpio_get_cd(mmc);
+   if (mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO)
+       return test_bit(DW_MMC_CARD_PRESENT, &slot->flags);
+
    /* Use platform get_cd function, else try onboard card detect */
    if ((brd->quirks & DW_MCI_QUIRK_BROKEN_CARD_DETECTION) ||
        (mmc->caps & MMC_CAP_NONREMOVABLE))
@@ -2755,6 +2758,9 @@ static int dw_mci_init_slot(struct dw_mci *host, unsigned
int id)
    dw_mci_get_cd(mmc);
+   if (mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO)
+       clear_bit(DW_MMC_CARD_PRESENT, &slot->flags);
+
    ret = mmc_add_host(mmc);
    if (ret)
        goto err_host_allocated;

--- a/drivers/mmc/core/sdio.c
+++ b/drivers/mmc/core/sdio.c
@@ -996,9 +996,7 @@ static int mmc_sdio_resume(struct mmc_host *host)
        }

        /* No need to reinitialize powered-resumed nonremovable cards */
-       if (mmc_card_is_removable(host) || !mmc_card_keep_power(host)) {
-           err = mmc_sdio_reinit_card(host, mmc_card_keep_power(host));
-       } else if (mmc_card_keep_power(host) && mmc_card_wake_sdio_irq(host)) {
+       if (mmc_card_keep_power(host) && mmc_card_wake_sdio_irq(host)) {
            /* We may have switched to 1-bit mode during suspend */
            err = sdio_enable_4bit_bus(host->card);
        }
```

11.9 Wi-Fi或者以太网UDP测试丢包率异常

```
# 可做如下修改验证：
# 每一个TCP socket最大的用于数据发送的缓存，字节；
echo 1048576 > /proc/sys/net/core/rmem_max
echo 12582912 > /proc/sys/net/core/wmem_max              #修改成12M
echo 2097152 > /proc/sys/net/core/wmem_default       #修改成2M
echo 65535 >  /proc/sys/net/core/netdev_max_backlog
#netdev_max_backlog    表示网卡设备的backlog，因为网卡接收数据包的速度远大于内核处理这些数据包的速度，所以，就出现了网卡设备的backlog.
```

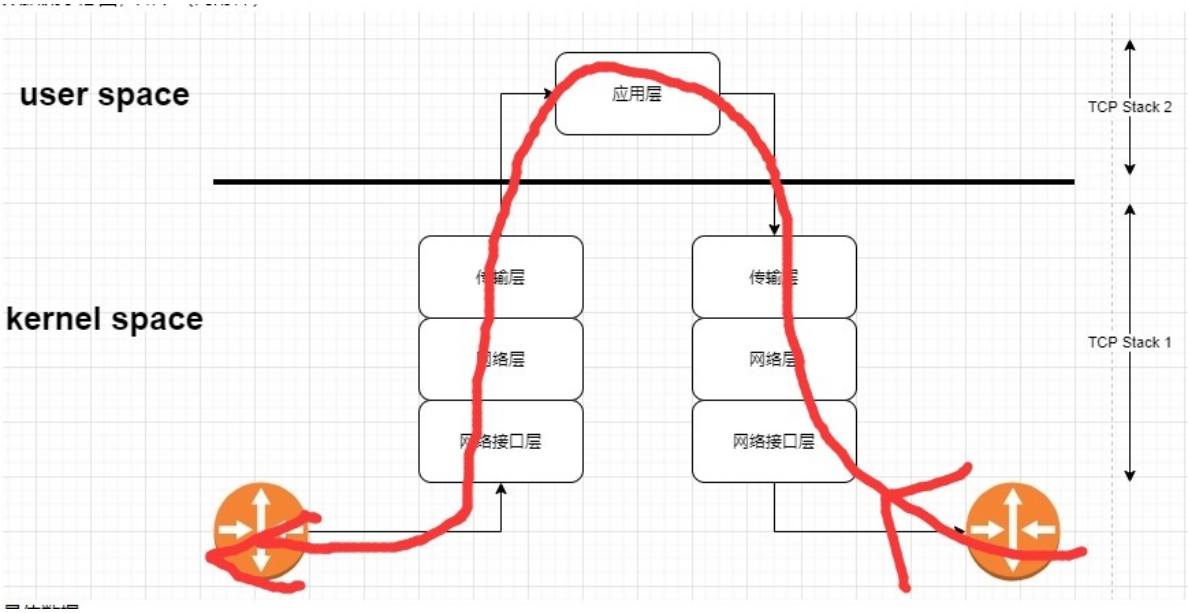
11.10 网络类排查步骤

11.10.1 卡顿丢帧类问题排查

```
# 如果使用以太网换网线确认

# 通过tcpdump抓报文确认是否有丢包，
# 如果是UDP+RTP的话，可以通过RTP序号来确认；

# 如果没有丢包，则可能是应用层收包不及时造成丢包的，可通过netstat查看tcp/ip层缓存数据情况来确认
netstat -n -t -u

# 关掉gro确认：
ethtool -K eth0 gro off

# cpu频率提到最高确认
echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

# 增大tcp/ip缓存buf大小来确认
# 单位是字节 接收缓存区大小，缓存从对端接收的数据，后续会被应用程序读取
echo "524288 1048576 2097152" > /proc/sys/net/ipv4/tcp_rmem
echo 2097152 > /proc/sys/net/core/rmem_max #
echo 2097152 > /proc/sys/net/core/rmem_default

# busybox ifconfig看是否存在丢包统计

# 视频播放类：TCP之后导致卡顿、UDP会导致花屏；
```

11.10.2 网络协议栈处理包时间简单验证

编写一个标准的echo网络程序，通过tcpdump抓包可以在下图的两个黄色圆圈处抓包可以计算出协议栈处理包的时间。



11.11 wpa_supplicant/hostapd更新版本

```
# 目前SDK的版本为2.6版本 + 解决WPA2的密钥重装漏洞
# 补丁对应说明及地址为：
# https://w1.fi/security/2017-1/wpa-packet-number-reuse-with-replayed-
messages.txt

# 如果需要更新到其他新版本，则方法如下：
# 到github官网找Buildroot仓库，进到如下目录：
https://github.com/buildroot/buildroot/tree/master/package/
... ...
# 分别找到wpa_supplicant和hostapd两个目录，并下载到本地：
wpa_supplicant
hostapd
... ...

# 进到SDK目录：
# RK3XXX_SDK: buildroot/package/
# 分别用前面下载的新版本目录直接替换掉wpa_supplicant和hostapd两个目录；

# 编译更新方法：
make wpa_supplicant-dirclean && make wpa_supplicant
make hostapd-dirclean && make hostapd
./build.sh rootfs
```

11.12 驱动应用DEBUG 调试配置

11.12.1 Wi-Fi 驱动DEBUG

有时需要更加详细的log来debug问题，**Realtek**芯片请打开如下选项使内核打印更加完整的驱动log：

```
#修改目录：kernel/drivers/net/wireless/rockchip_wlan/rtl8xxx
#较新的驱动：
Makefile
+CONFIG_RTW_DEBUG = y
+CONFIG_RTW_LOG_LEVEL = 2 #默认为2，debug时改为4可以打印更完整的log信息

有些旧的驱动没有这个配置，使能如下配置：
include/autoconf.h
+#define CONFIG_DEBUG /* DBG_871X, etc... */
```

11.12.2 TCPDUMP 抓包|

网络应用问题有时需要抓包确认问题，打开配置，编译生成tcpdump可执行程序，抓包命令：

```
Buildroot 选择对应配置 BR2_PACKAGE_TCPDUMP
tcpdump -h
tcpdump -i wlan0 -w /data/xxxx.pcap
```

11.12.3 wpa_supplicant 调试

- 有时需要wpa_supplicant的log调试问题

```
#Buildroot打开如下配置
#切记make savedefconfig保存起来
+ BR2_PACKAGE_WPA_SUPPLICANT_DEBUG_SYSLOG

#重新编译
make wpa_supplicant-dirclean
make wpa_supplicant-rebuild

#在启动wpa_supplicant时，要加个-s参数，这样log会输出到/var/log/messages文件里面
"  -s = log output to syslog instead of stdout"

#-d 打印更多log
"  -d = increase debugging verbosity (-dd even more)"

#由于wpa的log非常多，而messages文件大小很小，所以可以修改如下配置，增大文件大小
buildroot/package/busybox/S01logging
@@ -3,7 +3,7 @@
# Start logging
#
-SYSLOGD_ARGS=-n
+SYSLOGD_ARGS="-n -s 8192"

#重新编译busybox
make busybox-dirclean
make busybox-rebuild

#最后重新打包
./build.sh

#启动wpa_supplicant，并添加-s -d等debug选项
```

```
wpa_supplicant -B -i wlan0 -c /xxx/wpa_supplicant.conf -s -ddd
```

11.12.4 SDIO驱动DEBUG

```
#注意：kernel版本不同，结构体有所变化，根据实际的内核版本合下补丁
diff --git a/drivers/mmc/host/dw_mmc.c b/drivers/mmc/host/dw_mmc.c
old mode 100644
new mode 100755
index 0ed1854..3019413
--- a/drivers/mmc/host/dw_mmc.c
+++ b/drivers/mmc/host/dw_mmc.c
@@ -410,6 +410,9 @@ static void dw_mci_start_command(struct dw_mci *host,
        "start command: ARGR=0x%08x CMDR=0x%08x\n",
        cmd->arg, cmd_flags);
+   if (host->slot[0]->mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO) //kernel 4.4
+   if (host->slot->mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO) //kernel 4.19
+       pr_err("start command: ARGR=0x%08x CMDR=0x%08x\n", cmd->arg, cmd_flags);
+
    mci_writel(host, CMDARG, cmd->arg);
@@ -2529,6 +2532,9 @@ static irqreturn_t dw_mci_interrupt(int irq, void *dev_id)
    pending = mci_readl(host, MINTSTS); /* read-only mask reg */
+   if (host->slot[0]->mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO)//kernel 4.4
+   if (host->slot->mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO) //kernel 4.19
+       pr_err("=== dw_mci_interrupt: pending: 0x%x ===\n", pending);
+
    /*
     * DTO fix - version 2.10a and below, and only if internal DMA
     * is configured.
@@ -2558,6 +2564,8 @@ static irqreturn_t dw_mci_interrupt(int irq, void *dev_id)
    }
    if (pending & DW_MCI_CMD_ERROR_FLAGS) {
+       if (host->slot[0]->mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO)//kernel
4.4
+       if (host->slot->mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO)//kernel
4.19
+           pr_err("=== CMD ERROR: 0x%x ===\n", pending);
        spin_lock_irqsave(&host->irq_lock, irqflags);
        del_timer(&host->cto_timer);
@@ -2573,6 +2581,8 @@ static irqreturn_t dw_mci_interrupt(int irq, void *dev_id)
        }
    if (pending & DW_MCI_DATA_ERROR_FLAGS) {
+       if (host->slot[0]->mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO)//kernel
4.4
+       if (host->slot->mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO)//kernel
4.19
+       pr_err("=== DATA ERROR: 0x%x ===\n", pending);
        /* if there is an error report DATA_ERROR */
        mci_writel(host, RINTSTS, DW_MCI_DATA_ERROR_FLAGS);
        host->data_status = pending;
@@ -2582,6 +2592,8 @@ static irqreturn_t dw_mci_interrupt(int irq, void *dev_id)
    }
    if (pending & SDMMC_INT_DATA_OVER) {
+       if (host->slot[0]->mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO)//kernel
4.4
```

```
+       if (host->slot->mmc->restrict_caps & RESTRICT_CARD_TYPE_SDIO)//kernel
4.19
+           pr_err("=== SDMMC_INT_DATA_OVER: 0x%x ===\n", pending);
```

12. 应用开发

下面介绍的是一个应用开发库，它屏蔽了底层复杂的**Wi-Fi-BT**操作，比如**wpa_supplicant/bluez/bsa**等，向上提供友好的应用开发接口，针对**Buildroot**原生系统，其它像**Debian/**麒麟**/UOS**的第三方系统都有完整的上层**WIFI-BT**应用，所有无需使用此接口。

12.1 旧版本Deviceio应用开发介绍

Deviceio(external/deviceio_release)是一个应用开发库，它屏蔽了底层复杂的Wi-FiBT操作比如wpa_supplicant/bluez/bsa等，向上提供友好的应用开发接口，详细请参考 docs\Linux\Wi-Fi/BT目录下的文档:

```
Rockchip_Developer_Guide_DeviceIo_WIFI_CN.pdf        #Wi-Fi开发
Rockchip_Developer_Guide_DeviceIo_Bluetooth_CN.pdf   #蓝牙开发
Rockchip_Developer_Guide_Network_Config_CN.pdf       #配网开发（BLE)
```

12.1.1 配置编译

Realtek蓝牙使用Linux标准的bluez协议栈： `buildroot/packages/bluez5_utils`；

正基和海华使用私有的bsa协议栈：`external/broadcom_bsa 和 external/bluetooth_bsa`；

不同模块的对应配置：

- Realtek模块（例如rtl8723ds）配置如下：

```
#内核配置
CONFIG_BT_HCIUART=n #切记关掉这个配置

#Buildroot
BR2_PACKAGE_RKWIFIBT_RTL8723DS=y
BR2_PACKAGE_DEVICEIO_RELEASE=y
BR2_PACKAGE_BLUEZ_ALSA=y
BR2_PACKAGE_SBC=y
BR2_PACKAGE_BLUEZ5_UTILS=y
```

- 正基模组例如ap6255

```
#内核配置
CONFIG_BT_HCIUART=y #打开这个配置

#Buildroot
BR2_PACKAGE_RKWIFIBT_AP6255=y
BR2_PACKAGE_BROADCOM_BSA=y      #external/broadcom_bsa
BR2_PACKAGE_DEVICEIO_RELEASE=y
```

- 海华模组比如AW-CM256

```
#内核配置
CONFIG_BT_HCIUART=y #打开这个配置

#Buildroot
BR2_PACKAGE_RKWIFIBT_AWCM256=y
BR2_PACKAGE_CYPRESS_BSA=y       #external/bluetooth_bsa`
BR2_PACKAGE_DEVICEIO_RELEASE=y
```

编译更新：

make menuconfig 修改完配置后，需要make savedefconfig 保存配置，参考第2.1章节。

正基模组：

```
#严格按照下面顺序编译：
make rkwifibt-dirclean && make rkwifibt-rebuild
make broadcom_bsa-dirclean && make broadcom_bsa-rebuild
make deviceio_release-dirclean && make deviceio_release
```

海华模组：

```
#严格按照下面顺序编译：
make rkwifibt-dirclean && make rkwifibt-rebuild
make cypress_bsa-dirclean && make cypress_bsa-rebuild
make deviceio_release-dirclean && make deviceio_release
```

Realtek模组：

```
#严格按照下面顺序编译：
make rkwifibt-dirclean && make rkwifibt-rebuild
make bluez5_utils-dirclean && make bluez5_utils-rebuild
make bluez-alsa-dirclean && make bluez-alsa-rebuild
make deviceio_release-dirclean && make deviceio_release
#注：bluez-alsa是配合bluez支持：A2DP和HFP功能
```

12.2 RKWIFIBT-APP应用开发

RK3588 平台buildroot系统使用rkwifibt-app进行应用开发，相关接口跟deviceio保持一致。

12.3 应用接口开发最新说明

详细开发指导从如下FTP地址获取：

公共参考资料（*WIFI*、蓝牙、以太网相关资料 / 补丁 / *APK*源码 / 芯片规格书 ） - *FAE* 项目 - Rockchip Redmine (rock-chips.com)

路径：**/11-Linux**平台**/WIFIBT**编程接口**/**最新**WIFIBT**接口说明**.txt**

13. RV1106/1103/RK3588 IPC SDK Wi-Fi说明

13.1 配置说明

系统配置如下：

打开Wi-Fi功能，则配置(参考：IPC SDK配置及编译方法参考RV1106/1103 SDK的doc目录的相关说明)修改如下，假定使用BoardConfig-EMMC-NONE-EVB1_V10-IPC.mk这个配置：

```
# project/cfg$
BoardConfig-EMMC-NONE-EVB1_V10-IPC.mk

# Kernel defconfig fragment
# rv1106-xxx.config为所选配置的默认配置，确定WiFi的接口类型，根据类型添加如下配置：
# USB接口WIFI添加如下配置：rv1106-mini-usbwifi.config
+export RK_KERNEL_DEFCONFIG_FRAGMENT="rv1106-xxx.config rv1106-mini-
usbwifi.config"
# SDIO接口WIFI添加如下配置：rv1106-mini-sdiowifi.config
+export RK_KERNEL_DEFCONFIG_FRAGMENT="rv1106-xxx.config rv1106-mini-
sdiowifi.config"

# 使能WIFI功能
+export RK_ENABLE_WIFI=y
```

注意：usb/sdio wifi config文件从WiFi FTP服务器获取：

FTP地址：

公共参考资料（WIFI、蓝牙、以太网相关资料 / 补丁 / APK源码 / 芯片规格书 ） - FAE 项目 - Rockchip Redmine (rock-chips.com)

路径：*/11-Linux平台/RV1106_03平台/rv1106-mini-usbwifi.config*。

DTS配置：参考2.2章节

然后./build.sh进行编译，编译完成后会生成支持WiFi的ko，及wpa_supplicant/wpa_cli程序；对应运行目录为：

/oem/usr/ko/wifi_xxx.ko

/usr/bin/wpa_supplicant

/usr/bin/wpa_cli

13.2 WiFi相关文件说明

Wi-Fi驱动源码目录为：

```
# ls sysdrv/drv_ko/wifi/
hichannel        #海思Hi3861L，低功耗WiFi
rtl8188ftv       #RTL8188FU
rtl8189fs        #RTL8189FS/FTV
ssv6x5x          #SSV6155P
insmod_wifi.sh   #驱动ko加载脚本
```

目前支持RTL8188FU/8189FS、Hi3861L、南方硅谷SSV6155P；

Wi-Fi联网程序wpa_supplicant的源码目录：

```
#ls sysdrv/tools/board/wifi_app/
hisi_tools            #海思低功耗WiFi的应用配置源码
Makefile              #编译规则
wpa_supplicant_hostapd-0.8_rtw-2-ga8ef7c824.20200911.tar.gz #默认的 WiFi wpa精简守护进程
wpa_supplicant-2.6   #默认的完整功能守护进程，跟上面的二选一
wpa_supplicant.conf #默认的配置文件
```

SDK自带联网脚本：insmod_wifi.sh

根据VID:PID来判断WiFi型号，进而自动加载对应的WiFi驱动及对应的依赖模块：

```
#!/bin/sh

#hi3861l
cat /sys/bus/sdio/devices/*/uevent | grep "0296:5347"
if [ $? -eq 0 ];then
    insmod $(pwd)/hichannel.ko
    sleep 0.5
    vlink_hichannel_main &
fi

#rtl8189fs
cat /sys/bus/sdio/devices/*/uevent | grep "024C:F179"
if [ $? -eq 0 ];then
    insmod $(pwd)/8189fs.ko
fi

#usb wifi
echo 1 > /sys/devices/platform/ff3e0000.usb2-phy/otg_mode
sleep 0.8

#rtl18188fu
cat /sys/bus/usb/devices/*/uevent | grep "bda\/f179"
if [ $? -eq 0 ];then
    insmod  $(pwd)/8188fu.ko
fi

#ssv6x5x
cat /sys/bus/usb/devices/*/uevent | grep "8065\/6000"
if [ $? -eq 0 ];then
    insmod  $(pwd)/cfg80211.ko
    insmod  $(pwd)/libarc4.ko
    insmod  $(pwd)/mac80211.ko
    insmod  $(pwd)/ctr.ko
    insmod  $(pwd)/ccm.ko
    insmod  $(pwd)/libaes.ko
    insmod  $(pwd)/aes_generic.ko
    insmod  $(pwd)/ssv6x5x.ko
fi
```

注意：南方硅谷**WiFi**驱动必需要加载**cfg/mac80211**相关驱动；而**Realtek WiFi**默认使用传统的**WEXT**方式，这样可以省空间，适用一些**flash**容量比较小的情况。

13.3 第三方应用移植说明

SDK默认只提供了wpa_supplicant/wpa_cli/udhcpc等必要的程序，如果需要使用第三方程序，则需要使用SDK提供的交叉编译工具链：arm-rockchip830-linux-uclibcgnueabihf- 进行交叉编译，编译好二进制可以放到如下目录：sysdrv\tools\board\wifi_app\bin且修改为可执行的权限，并修改sysdrv/Makefile文件：

```
ifeq ($(ENABLE_WIFI),y)
    $(MAKE) -C $(SYSDRV_DIR)/tools/board/wifi_app;
    @pushd $(SYSDRV_DIR_OUT_BOARD); \
+   cp -af xxx \
+       $(SYSDRV_DIR_OUT_ROOTFS)/usr/bin; \
```

重新编译二进制会被自动拷贝到/usr/bin/目录。

13.4 新驱动移植说明

假如需要使用SDK还未支持的WiFi，则参考如下步骤进行移植：以高拓ATBM6301为例：

首先拷贝原厂提供的驱动atbm到SDK目录：sysdrv/drv_ko/wifi/目录下：

```
ls sysdrv\drv_ko\wifi
atbm
```

然后修改驱动的Makefile文件，主要修改三个系统环境变量：

```
ARCH=$(ARCH)    //RV1106都是32位
CROSS_COMPILE=$(CROSS_COMPILE)   //交叉编译工具链
$(KERNEL_DIR) //内核目录
```

找到make相关的行进行修改：

```
install:
    @echo "make PLATFORM_CROSS=$(platform)"
-   $(MAKE) ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE) -C $(KERNEL_DIR)
M=$(shell pwd) modules -j12
+   $(MAKE) all -f $(MAKEFILE_SUB) ARCH=$(ARCH)  CROSS_COMPILE=$(CROSS_COMPILE)
KDIR=$(KERNEL_DIR) SYS=$(sys) PLAT=$(platform) -j8
+   #strip ko减少ko大小
+   $(CROSS_COMPILE)strip --strip-debug $(shell pwd)/driver_install/atbm603x_.ko
+   #拷贝到out目录，系统自动会拷贝到oem/usr/ko/目录
+   cp $(shell pwd)/driver_install/atbm603x_.ko $(M_OUT_DIR)
```

注意：需要原厂提供支持5.10内核的WiFi驱动！

最后修图insmod_wifi.sh文件，参考里面的其它项目进行添加对应的insmod内容。

13.5 问题排查说明

参考第**7.2**章节.

13.6 RF测试

参考第5章节，注意：需要WiFi原厂或模组厂提供测试之类及测试程序，测试程序最好是静态二进制，否则会出现无法执行的情况，此时则需要提供SDK的交叉编译工具链：arm-rockchip830-linux-uclibcgnueabihf- 给原厂或模组厂对测试程序的重新编译。

14. WiFi相关的裁减说明

对于小容量Flash/RAM设备场景下需要使用WiFi功能的，则可从如下几个方面进行裁减：

14.1 wpa_supplicant裁减

wpa_supplicant是WiFi的应用管理进程，包括station、hostapd、p2p、display、hostpot、wps等功能，我们只需选择需要的功能即可；修改wpa_supplicant目录下的.config文件：

```
#必选
BR2_PACKAGE_WPA_SUPPLICANT=y

#与内核无线框架交互的机制，最常用的有NL80211、WEXT(默认支持)
#如果选择的WiFi支持WEXT机制，则可以去掉这个配置
BR2_PACKAGE_WPA_SUPPLICANT_NL80211=y

#SOFTAP功能
BR2_PACKAGE_WPA_SUPPLICANT_AP_SUPPORT=y

#WIFI DISPLAY功能
BR2_PACKAGE_WPA_SUPPLICANT_WIFI_DISPLAY=y

#WiFi mesh功能
BR2_PACKAGE_WPA_SUPPLICANT_MESH_NETWORKING=y

#加密可选项，非必要
BR2_PACKAGE_WPA_SUPPLICANT_EAP=y

#hostpot功能
BR2_PACKAGE_WPA_SUPPLICANT_HOTSPOT=y

#debug log功能
BR2_PACKAGE_WPA_SUPPLICANT_DEBUG_SYSLOG=y

#wps功能
BR2_PACKAGE_WPA_SUPPLICANT_WPS=y

#wpa_cli功能，建议保留
```

```
BR2_PACKAGE_WPA_SUPPLICANT_CLI=y
BR2_PACKAGE_WPA_SUPPLICANT_WPA_CLIENT_SO=y

#密码生成工具，非必要
BR2_PACKAGE_WPA_SUPPLICANT_PASSPHRASE=y

#DBUG接口，与传统桌面系统应用对接，可去掉
BR2_PACKAGE_WPA_SUPPLICANT_DBUS_OLD=y
BR2_PACKAGE_WPA_SUPPLICANT_DBUS_NEW=y
```

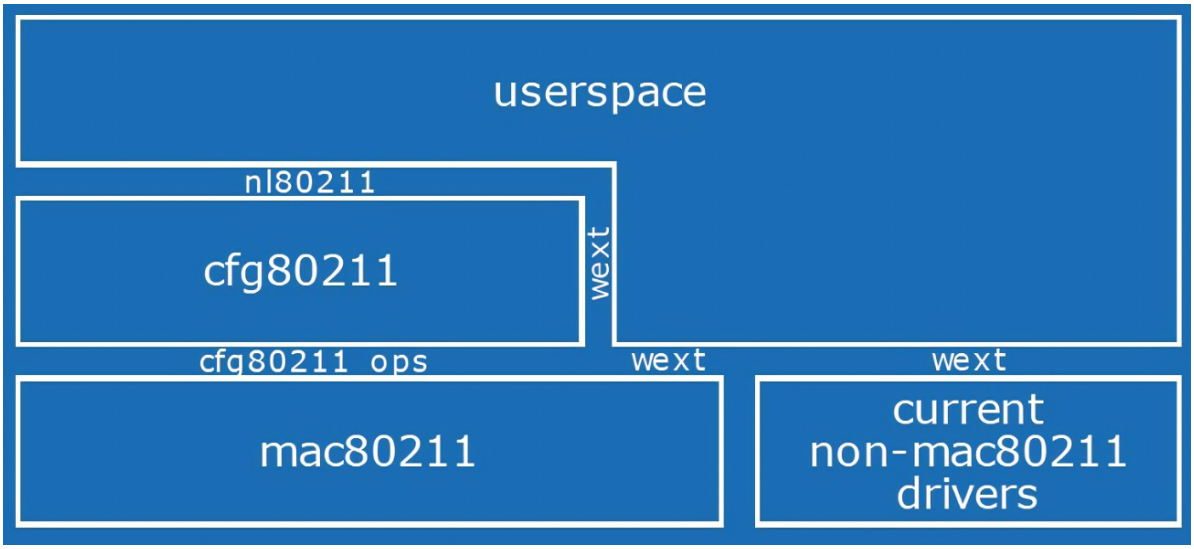
14.2 内核网络框架裁减

内核网络框架包括：tcp/ip协议栈、ipv4协议(默认支持)、ipv6、 ip隧道、Netfilter 包过滤/跟踪、NAT网络地址转换、iptables 网络防火墙系统等等，同样只需选择需要的功能即可。

```
#网络必要配置项目，默认支持IPV4
CONFIG_NET=y
CONFIG_COMPAT_NETLINK_MESSAGES=y
CONFIG_INET=y
CONFIG_OF_NET=y
CONFIG_NETDEVICES=y
CONFIG_NET_CORE=y
CONFIG_ETHERNET=y
CONFIG_GENERIC_NET_UTILS=y

#网络可选配置项目
#网络隧道技支持
CONFIG_NET_IP_TUNNEL=m
CONFIG_INET_TUNNEL=m

#IPv6配置
CONFIG_IPV6=m

#Netfilter 包过滤/跟踪、iptables网络防火墙
CONFIG_NETFILTER=m
CONFIG_BPFILTER=m
CONFIG_NF_TABLES=m
CONFIG_NF_CONNTRACK=m
CONFIG_NF_TABLES_INET=m
CONFIG_NF_CONNTRACK_IPV4=m
CONFIG_IP_NF_IPTABLES=m

#NAT网络地址转换
CONFIG_IP_NF_NAT=m
CONFIG_IP_NF_TARGET_MASQUERADE=m

#网络桥接
CONFIG_BRIDGE=m

#ethtool以太网交互交互配置
CONFIG_ETHTOOL_NETLINK=m
```

14.3 内核无线网络框架裁减

框架



概念

- cfg80211：用于对无线设备进行配置管理；
- mac80211：是为SoftMAC无线设备写驱动的框架；
- MLME：即MAC(Media Access Control) Layer Management Entity，它管理物理层MAC状态机；
- SoftMAC： 其MLME由软件实现，mac80211为SoftMAC实现提供了一个driver API。 即：SoftMAC设备允许对硬件执行更好地控制，允许用软件实现对802.11的帧管理，包括解析和产生802.11无线帧；
- FullMAC：其MLME由硬件管理，当写FullMAC无线驱动时，不需要使用mac80211；

目前Linux无线驱动接口有两种标准接口：wext（Wireless Extensions无线扩展接口）和nl80211接口：

- WEXT（Wireless Extension）：使用WEXT的工具通过ioctl和驱动通信，典型工具ifconfig等;
- nl80211：基于Netlink对无线设备进行配置管理提供接口；

所以目前有如下几种架构图组合：

- Wext ＋私有Driver
- NL80211 + Fullmac
- NL80211 + SoftMac

下面分别上面组合进行相关配置说明:

14.3.1 WEXT架构

支持模块：Realtek系列；

```
#Wireless必要配置项目
CONFIG_RFKILL=y
CONFIG_RFKILL_RK=y
CONFIG_WLAN=y
CONFIG_WIRELESS=y

#WEXT模式必要配置
CONFIG_WIRELESS_EXT=y
CONFIG_WEXT_CORE=y
CONFIG_WEXT_PROC=y
CONFIG_WEXT_PRIV=y
```

14.3.2 SoftMac

支持模块：RK915、南方硅谷/高拓等模块等；

```
#Wireless必要配置项目
CONFIG_RFKILL=y
CONFIG_RFKILL_RK=y
CONFIG_WLAN=y
CONFIG_WIRELESS=y

#SoftMac模式必要配置
CONFIG_CFG80211=y
CONFIG_MAC80211=y
```

14.3.3 FullMac

支持模块Realtek、broadcom系列；

```
#Wireless必要配置项目
CONFIG_RFKILL=y
CONFIG_RFKILL_RK=y
CONFIG_WLAN=y
CONFIG_WIRELESS=y

#FullMac模式必要配置
CONFIG_CFG80211=y
```

14.4 驱动接口裁减

针对USB接口的WiFi，USB模块的代码量很大，我们仅需留下host功能即可，相关的核心配置存放在WiFi FTP服务器：/11-Linux平台/RV1106_03平台/rv1106-mini-usbwifi.config。

15. 低功耗WiFi开发指导

15.1 简介

低功耗WiFi主要用于电池类产品，它有如下几个特点：

1. 功耗要低，休眠情况下一般不超过3.3v/500uA；
2. 可以独立于主芯片单独运行，可以独自跟外部网络建立TCP/UDP连接；
3. 可以控制主芯片的电源的开关，通过WiFi网络唤醒是主芯片开机；
4. 主芯片被网络唤醒开机后快速联网；
5. 内部有独立的MCU及相关接口进行定制化功能开发，比如PIR/按键；(可选功能)；

目前市面上有两种类型的低功耗WiFi：

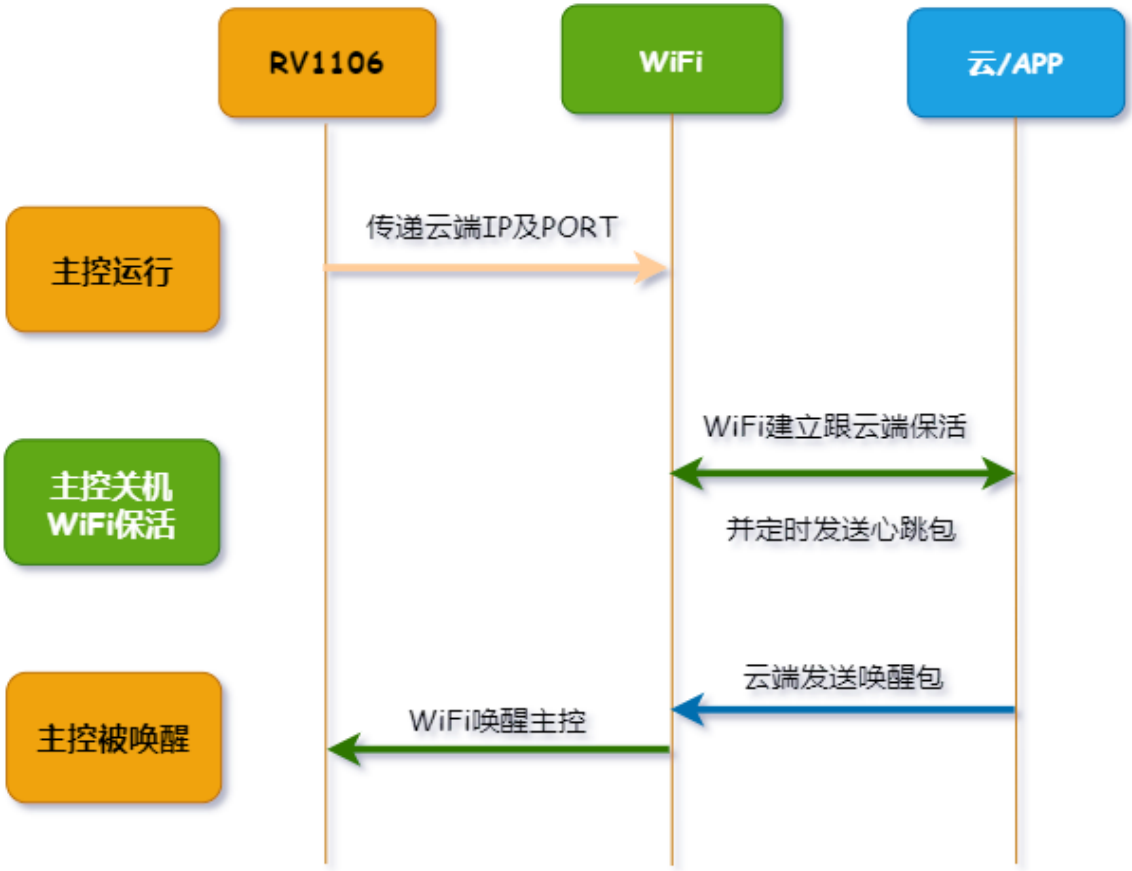
- 一种是WiFi模块自带MCU并包括独立的TCP/IP协议栈，可编程并独立运行网络相关的程序；
- 一种WiFi是不带MCU，但可独立运行网络相关程序；

下面分别对两种类型的低功耗WiFi方案进行进行说明：

15.2 WiFi带MCU方案

代表芯片：**ATBM6441/Hi3861L**

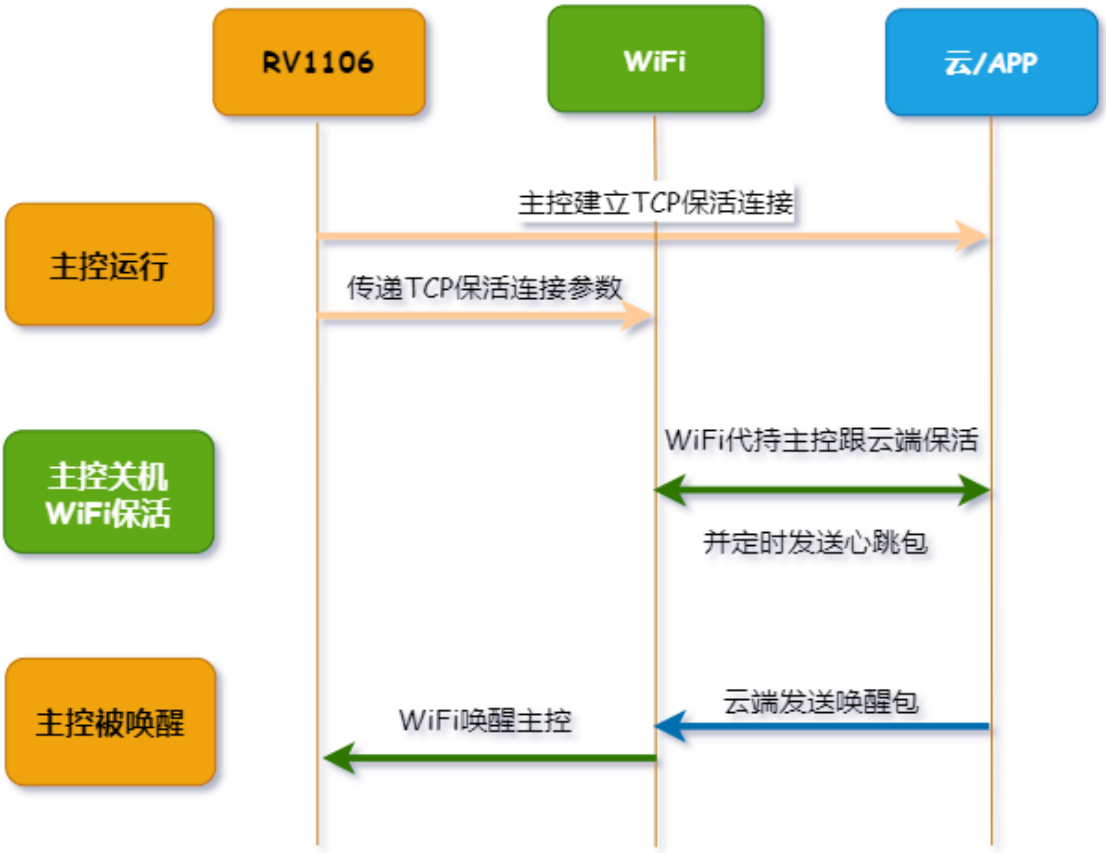
1. 正常状态：IPC设备首先与云端建立一个视频推流连接，当推完录像流，或云端主动断开直播流后，设备把云云端保活的IP地址及端口号发给WiFi设备；
2. 触发休眠进入低功耗：由IPC摄像头设备物理按键、摄像头程序逻辑(检测空闲时)等方式触发休眠或WiFi控制IPC设备下电或休眠；
3. 进入休眠前：WiFi芯片通过前面收到的保活地址及端口号，独立连接一个保活连接，并定期发送心跳包，让云端一直知道设备端在线；
4. 设备休眠：IPC摄像头设备关机，此时仅由WiFi芯片在运行并维持和云端的网络心跳；
5. 唤醒设备：云端通过保活连接发送唤醒包给WiFi，WiFi通过GPIO的方式唤醒IPC设备；
6. 退出休眠：IPC设备被触发开机，重新建立跟云端视频推流连接，回到第1步的状态；



15.3 WiFi不带MCU方案

代表芯片：**AP6203/6201BM**

1. 正常状态：IPC设备首先与云端建立一个视频推流连接，当推完录像流，或云端主动断开直播流后，设备本身将建立一个与云端的保活连接，并定期发送心跳包给云端，让云端一直知道设备端在线；
2. 触发休眠进入低功耗：由IPC摄像头设备物理按键、摄像头程序逻辑(检测空闲时)等方式触发休眠或WiFi控制RK主芯片下电或休眠；
3. 进入休眠前：IPC设备把第1步建立保活连接参数发给WiFi，由WiFi继续维持这个保活TCP连接，并定期发送心跳包；
4. 设备休眠：IPC摄像头设备关机，此时仅由WiFi芯片在运行并维持和云端的TCP心跳；
5. 唤醒设备：云端通过保活连接发送唤醒包给WiFi，WiFi通过GPIO的方式唤醒IPC设备；
6. 退出休眠：IPC设备被触发开机，重新建立跟云端视频推流连接，回到第1步的状态；



15.4 开发指导

详细开发指导从如下FTP地址获取：

公共参考资料（WIFI、蓝牙、以太网相关资料 / 补丁 / APK源码 / 芯片规格书 ） - FAE 项目 - Rockchip Redmine (rock-chips.com)

路径：**/11-Linux**平台**/WIFI**低功耗方案