Rockchip Linux Wi-Fi/BT 开发

文件标识：RK-KF-YF-381

发布版本：V7.0.1

日期：2024-04-16

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WiFi/BT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；
V6.3.1	xy	2022-09	增加WiFi/BT开发流程说明； 增加新的WiFi/BT的编译框架说明； 增加RV1106/03的低功耗WiFi开发说明； 增加Debian系统WiFi/BT的ko及firmware文件的简化安装说明； 增加RV1106/03 BLE开发说明；
V6.3.2	xy	2023-02	WiFi/BT的编译框架说明--补充说明； WiFi/BT接口支持低功耗接口； rkweb_ui配网新SDK无法配网问题； 增加PCIE的配置说明； 增加USB的配置说明；
V7.0.1	xy	2024-16	移除过时的相关说明，包括编译方式及说明等等 新SDK框架配置编译说明； PCIE配置更新说明； 新增RKWIFIBT-APP 新API说明及编程指南

目录

1. 前言

文档主要提供WiFi/BT的配置/调试/开发以及应用的指导，包括Buildroot/IPC低功耗/Debian等SDK。

2. Wi-Fi/BT 配置

2.1 SDK 配置指导

首先配置芯片和板子型号

`./build.sh chip`

可以修改默认配置，假定使用如下deconfig：

```
SDK_ROOT/device/rockchip$
+++ b/rk3588/rockchip_rk3588_evb1_lp4_v10_defconfig
@@ -1,4 +1,5 @@
 RK_YOCTO_CFG="rockchip-rk3568-evb"
+RK_WIFIBT_CHIP="ALL_AP"   //Note the colon ""
 RK_KERNEL_DTS_NAME="rk3568-evb1-ddr4-v10-linux" //The specific dts file used
 RK_PARAMETER="parameter-buildroot-fit.txt"
```

- 通过make menuconfig 配置WiFi型号：RK_WIFIBT_CHIP（输入"/"搜索RK_WIFIBT_CHIP ）
  - **ALL_AP**：支持SDK所有的新思（正基）和Realtek模块
- **ALL_CY**：支持SDK所有的英飞凌和Realtek模块
- **指定具体的型号：**

> AP6275/AP6358S/AP6212/AP6236 ... ...
>
> RTL8723DS/RTL8822CS ... ...
>
> CYW4354/CYW43438/CYW5557X_PCIE/CYW5557X_PCIE

- 支持的所有具体型号如下

```
device/rockchip/common/scripts/post-wifibt.sh
156:    if [[ "$RK_WIFIBT_CHIP" = "AP6275_PCIE" ]];then
167:    if [[ "$RK_WIFIBT_CHIP" = "CYW4354" ]];then
174:    if [[ "$RK_WIFIBT_CHIP" = "CYW4373" ]];then
181:    if [[ "$RK_WIFIBT_CHIP" = "CYW43438" ]];then
188:    if [[ "$RK_WIFIBT_CHIP" = "CYW43455" ]];then
195:    if [[ "$RK_WIFIBT_CHIP" = "CYW5557X" ]];then
202:    if [[ "$RK_WIFIBT_CHIP" = "CYW5557X_PCIE" ]];then
209:    if [[ "$RK_WIFIBT_CHIP" = "CYW54591" ]];then
216:    if [[ "$RK_WIFIBT_CHIP" = "CYW54591_PCIE" ]];then
223:    if [[ "$RK_WIFIBT_CHIP" = "RTL8188FU" ]];then
228:    if [[ "$RK_WIFIBT_CHIP" = "RTL8189FS" ]];then
233:    if [[ "$RK_WIFIBT_CHIP" = "RTL8723DS" ]];then
```

```
237:    if [[ "$RK_WIFIBT_CHIP" = "RTL8821CS" ]];then
241:    if [[ "$RK_WIFIBT_CHIP" = "RTL8822CS" ]];then
245:    if [[ "$RK_WIFIBT_CHIP" = "RTL8852BS" ]];then
249:    if [[ "$RK_WIFIBT_CHIP" = "RTL8852BE" ]];then
280:    if [[ "$RK_WIFIBT_CHIP" = "ALL_CY" ]];then
299:    if [[ "$RK_WIFIBT_CHIP" = "ALL_AP" ]];then
```

- 其它型号的支持：比如爱科微(AIC)、高拓(ATBM)等国产模块，可以从FTP上获取支持

2.2 DTS 配置

务必对照原理图进行配置

2.2.1 SDIO Wi-Fi 配置

支持两种方式，主要是**电源控制**和**识别卡的时机**的区别

- 标准框架
  - 原理参考：The purpose of the simple MMC power sequence provider is to supports a set of common properties between various SOC designs. It thus enables us to use the same provider for several SOC designs.mmc-pwrseq-simple
  - WIFI的REG_ON由mmc框架提供支持，参考代码 `kernel/drivers/mmc/core/pwrseq_simple.c`
  - 优点：所有的WiFi的电源由mmc框架统一管理
  - 缺点：不支持SDIO card 重新识别

```
/* System-on-Chip designs may specify a specific
 * MMC power sequence. To successfully detect an
 * SDIO card, that power sequence must be maintained
 * while initializing the card.
 */
sdio_pwrseq: sdio-pwrseq {
    compatible = "mmc-pwrseq-simple";
    pinctrl-names = "default";
    pinctrl-0 = <&wifi_enable_h>;

    /* reset-gpios:
     * contains a list of GPIO specifiers. The reset GPIOs are asserted
     * at initialization and prior we start the power up procedure of the
     * card. They will be de-asserted right after the power has been
     * provided to the card.
     *
     * Alias: WIFI_REG_ON/WL_REG_ON etc (schematic diagram)
     * Notes:
     * GPIO_ACTIVE_LOW: Active HIGH
     * GPIO_ACTIVE_LOW: Active LOW
     */
    reset-gpios = <&gpio0 RK_PA2 GPIO_ACTIVE_LOW>;

    /*
     * Default: disabled
     */
    //clocks = <&clk_32768_ck>;  //Handle for the entry in clock-names.
```

```
    //clock-names = "ext_clock"; //32.768K: External clock provided to the
card.

    /* It's reused as a tunable delay waiting for I/O signalling
     * and card power supply to be stable, regardless of whether
     * pwrseq-simple is used. Default to 10ms if no available.
     *
     * Delay in ms after powering the card and de-asserting the
     * reset-gpios (if any).
     *
     * Default: disabled
     */
    //post-power-on-delay-ms = <200>;

    /* Delay in us after asserting the reset-gpios (if any)
     * during power off of the card.
     *
     * Default: disabled
     */
    //power-off-delay-us = <200>;
};

/* Related to reset-gpio above */
&pinctrl {
    sdio-pwrseq {
        wifi_enable_h: wifi-enable-h {
            rockchip,pins =
            <0 RK_PA2 RK_FUNC_GPIO &pcfg_pull_none>;
        };
    };
};

/* SDIO
 * https://elixir.bootlin.com/linux/latest/source/
 * Documentation/devicetree/bindings/mmc/mmc-controller.yaml
 */
&sdio {
    /*
     * Description:
     * Maximum operating frequency of the bus:
     *  - for SD/SDIO cards the SDR104 mode has a max supported
     *    frequency of 150 or 200MHz,
     * So, lets keep the maximum supported value here.
     */
    max-frequency = <150000000>;

    /* Number of data lines. */
    bus-width = <4>;

    /* SD high-speed timing is supported. */
    cap-sd-highspeed;

    /* SD UHS SDR104 speed is supported. */
    sd-uhs-sdr104;

    /*
     * Controller is limited to send SD commands during initialization.
     * Controller is limited to send MMC commands during initialization.
```

```
     */
    no-sd;
    no-mmc;

    /* Non-removable slot (like SDIO WiFI); assume always present. */
    non-removable;

    /* enable SDIO IRQ signalling on this interface */
    cap-sdio-irq;

    /* SDIO only. Preserves card power during a suspend/resume cycle. */
    keep-power-in-suspend;

    /* Enable pwrseq*/
    mmc-pwrseq = <&sdio_pwrseq>;

    status = "okay";
};

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

    /* Attention: If the Wi-Fi module requires 32.768K and is supplied
     * by the Power Management Integrated Circuit (PMIC) as indicated
     * in the schematic, open the following properties and fill in the
     * relevant attributes according to the actual PMIC model used.
     */
    clocks = <&rk809 1>;  //RK809
    clocks = <&hym8563>;  //or hym8563
    clock-names = "ext_clock";

    /* The name of the Wi-Fi, without any specific constraints,
     * serves merely as a suggestive indicator.
     */
    wifi_chip_type = "ap6255";

    /* WIFI_WAKE_HOST: The Wi-Fi chip issues an interrupt notification
     * to the System on Chip (SOC) through this pin for further operations.
     * Special attention: Please verify the connection relationship between
     * this pin and the main controller(SOC); the configuration for a direct
     * connection is GPIO_ACTIVE_HIGH. If an inverted tube is introduced
     * in between, the configuration should be set to GPIO_ACTIVE_LOW.
     */
    WIFI,host_wake_irq = <&gpio0 RK_PA0 GPIO_ACTIVE_HIGH>;

    status = "okay";
};

wireless-wlan {
    /omit-if-no-ref/
    wifi_wake_host: wifi-wake-host {
        /* The initial state of the interrupt pin.*/
        rockchip,pins = <0 RK_PA0 0 &pcfg_pull_none>;
    };
};
```

- 传统框架
  - WiFi的REG_ON由`net/rfkill/rfkill-wlan.c`控制
  - 可以多次rmmod/insmod WiFi驱动，需要打补丁： KO模式的补丁

```
/* Wi-Fi */
wireless-wlan {
    compatible = "wlan-platdata";
    rockchip,grf = <&grf>;
    clocks = <&rk809 1>;  //RK809
    clocks = <&hym8563>;  //or hym8563
    clock-names = "ext_clock";
    wifi_chip_type = "ap6255";
    WIFI,host_wake_irq = <&gpio0 RK_PA0 GPIO_ACTIVE_HIGH>;
    WIFI,poweren_gpio = <&gpio0 RK_PA2 GPIO_ACTIVE_HIGH>;
    status = "okay";
};

&sdio {
    max-frequency = <150000000>;
    bus-width = <4>;
    cap-sd-highspeed;
    sd-uhs-sdr104;
    no-sd;
    no-mmc;
    cap-sdio-irq;
    keep-power-in-suspend;

    /* delete-property */
    /delete-property/ non-removable;
    /delete-property/ mmc-pwrseq = <&sdio_pwrseq>;

    status = "okay";
};

wireless-wlan {
    /omit-if-no-ref/
    wifi_wake_host: wifi-wake-host {
        rockchip,pins = <0 RK_PA0 0 &pcfg_pull_none>;
    };
};
```

2.2.2 蓝牙配置

UART相关的都要配置为**实际使用的UART口的所对应PIN脚**，注意RTS/CTS PIN脚一定要按照SDK设计去接！参考蓝牙注意事项 。

> 特别注意：很多异常都是因为这两个**PIN**脚没有接或者没有按照要去导致初始化异常！

下面假设蓝牙使用UART4：

```
/* 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"

    /* 务必要配置，且确保配置正确
     * 这里要配置对应主控UART的RTS脚
     * 特殊配置的原因是有些蓝牙芯片CTS脚上电前要保持为低电平，
     * 否则会进入测试模式或者其它非正常的工作模式，所以驱动会在上
     * 电前，通过pinctrl-1拉低，上电后，通过pinctrl-0切回RTS功能；
     */
    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";
};

/* uart4_rts_gpio */
&pinctrl {
    wireless-bluetooth {
        uart4_rts_gpio: uart4-rts-gpio {
            rockchip,pins = <4 RK_PA7 RK_FUNC_GPIO &pcfg_pull_none>;
        };
    };
};
```

kernel 6.1可选的Upstream配置

broadcom-bluetooth

```
# SPDX-License-Identifier: (GPL-2.0 OR BSD-2-Clause)
%YAML 1.2
---
$id: http://devicetree.org/schemas/net/broadcom-bluetooth.yaml#
$schema: http://devicetree.org/meta-schemas/core.yaml#

title: Broadcom Bluetooth Chips

maintainers:
  - Linus Walleij <linus.walleij@linaro.org>

description:
```

```
  This binding describes Broadcom UART-attached bluetooth chips.

properties:
  compatible:
    enum:
      - brcm,bcm20702a1
      - brcm,bcm4329-bt
      - brcm,bcm4330-bt
      - brcm,bcm4334-bt
      - brcm,bcm43430a0-bt
      - brcm,bcm43430a1-bt
      - brcm,bcm43438-bt
      - brcm,bcm4345c5
      - brcm,bcm43540-bt
      - brcm,bcm4335a0
      - brcm,bcm4349-bt
      - cypress,cyw4373a0-bt
      - infineon,cyw55572-bt

  shutdown-gpios:
    maxItems: 1
    description: GPIO specifier for the line BT_REG_ON used to
      power on the BT module

  reset-gpios:
    maxItems: 1
    description: GPIO specifier for the line BT_RST_N used to
      reset the BT module. This should be marked as
      GPIO_ACTIVE_LOW.

  device-wakeup-gpios:
    maxItems: 1
    description: GPIO specifier for the line BT_WAKE used to
      wakeup the controller. This is using the BT_GPIO_0
      pin on the chip when in use.

  host-wakeup-gpios:
    maxItems: 1
    deprecated: true
    description: GPIO specifier for the line HOST_WAKE used
      to wakeup the host processor. This is using he BT_GPIO_1
      pin on the chip when in use. This is deprecated and replaced
      by interrupts and "host-wakeup" interrupt-names

  clocks:
    minItems: 1
    maxItems: 2
    description: 1 or 2 clocks as defined in clock-names below,
      in that order

  clock-names:
    description: Names of the 1 to 2 supplied clocks
    oneOf:
      - const: extclk
        deprecated: true
        description: Deprecated in favor of txco

      - const: txco
```

```
        description: >
          external reference clock (not a standalone crystal)

      - const: lpo
        description: >
          external low power 32.768 kHz clock

      - items:
          - const: txco
          - const: lpo

  vbat-supply:
    description: phandle to regulator supply for VBAT

  vddio-supply:
    description: phandle to regulator supply for VDDIO

  brcm,bt-pcm-int-params:
    $ref: /schemas/types.yaml#/definitions/uint8-array
    minItems: 5
    maxItems: 5
    description: |-
      configure PCM parameters via a 5-byte array:
       sco-routing: 0 = PCM, 1 = Transport, 2 = Codec, 3 = I2S
       pcm-interface-rate: 128KBps, 256KBps, 512KBps, 1024KBps, 2048KBps
       pcm-frame-type: short, long
       pcm-sync-mode: slave, master
       pcm-clock-mode: slave, master

  brcm,requires-autobaud-mode:
    type: boolean
    description:
      Set this property if autobaud mode is required. Autobaud mode is required
      if the device's initial baud rate in normal mode is not supported by the
      host or if the device requires autobaud mode startup before loading FW.

  interrupts:
    items:
      - description: Handle to the line HOST_WAKE used to wake
          up the host processor. This uses the BT_GPIO_1 pin on
          the chip when in use.

  interrupt-names:
    items:
      - const: host-wakeup

  max-speed: true
  current-speed: true

required:
  - compatible

dependencies:
  brcm,requires-autobaud-mode: [ shutdown-gpios ]

if:
  not:
    properties:
```

```
      compatible:
        contains:
          enum:
            - brcm,bcm20702a1
            - brcm,bcm4329-bt
            - brcm,bcm4330-bt
then:
  properties:
    reset-gpios: false

additionalProperties: false

examples:
  - |
    #include <dt-bindings/gpio/gpio.h>
    #include <dt-bindings/interrupt-controller/irq.h>

    uart {
        uart-has-rtscts;

        bluetooth {
            compatible = "brcm,bcm4330-bt";
            max-speed = <921600>;
            brcm,bt-pcm-int-params = [01 02 00 01 01];
            shutdown-gpios = <&gpio 30 GPIO_ACTIVE_HIGH>;
            device-wakeup-gpios = <&gpio 7 GPIO_ACTIVE_HIGH>;
            reset-gpios = <&gpio 9 GPIO_ACTIVE_LOW>;
            interrupt-parent = <&gpio>;
            interrupts = <8 IRQ_TYPE_EDGE_FALLING>;
        };
    };
```

2.2.3 IO 电源域的配置

重要： 参考对应SDK的电源域配置说明

参考如下，需要指出SDIO3.0的设备务必要1.8v供电。

```
//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 PCIE Wi-Fi 配置

首先强烈建议阅读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

对照原理图找到如下三个关键点：

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，一般是长供电；

WIFI_REG_ON ：芯片的复位/使能管脚，开机保持高电平；

PCIE_PERST_L/PERSTO：芯片PCIe部分的复位管脚，DTS对应的PCIE节点下务必配置正确；

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";
};

/* 如果使用pcie2x1l2，sata0与pcie2x1l2复用了combphy0_ps，需确保禁用 */
&sata0 {
    status = "disabled";
}

//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。.
     * 二选一
     */
    ep-gpios = <&gpio3 13 GPIO_ACTIVE_HIGH>;  //RK3399平台
    reset-gpios = <&gpio4 RK_PA5 GPIO_ACTIVE_HIGH>; //RK356X/3588平台

    num-lanes = <4>;
    rockchip,skip-scan-in-resume;
    rockchip,perst-inactive-ms = <500>; /* 参考Wi-Fi模组手册，查询所需#PERST复位时间
*/
    vpcie3v3-supply = <&vcc3v3_pcie20_wifi>;
    status = "okay";
};
```

```
//配置PCIe WiFi的WIFI REG ON，目的是为了在开机识别阶段拉高
vcc3v3_pcie20_wifi: vcc3v3-pcie20-wifi {
    compatible = "regulator-fixed";
    regulator-name = "vcc3v3_pcie20_wifi";
    regulator-min-microvolt = <3300000>;
    regulator-max-microvolt = <3300000>;
    enable-active-high;
    /*
     * wifi_reg_on 是在vbat、vddio稳定后才使能，在reset-gpios前拉高，所以
     * 放到PCIe的电源节点中才满足模块时序，引用wifi_poweron_gpio。
     */
    pinctrl-0 = <&wifi_poweron_gpio>
    startup-delay-us = <5000>;
    vin-supply = <&vcc12v_dcin>;
};

//Wi-Fi节点配置
wireless_wlan: wireless-wlan {
    compatible = "wlan-platdata";
    wifi_chip_type = "ap6275p";
    //配置：WIFI_REG_ON，给WiFi驱动使用
    WIFI,poweren_gpio = <&gpio0 RK_PA2 GPIO_ACTIVE_HIGH>;
    //配置：WIFI_HOST_WAKE，如果有的话
    WIFI,host_wake_irq = <&gpio0 RK_PA0 GPIO_ACTIVE_HIGH>;
    status = "okay";
};

&pinctrl {
    wireless-wlan {
        wifi_poweren_gpio: wifi-poweren-gpio {
            //PCIE REG ON:  务必配置成上拉
            rockchip,pins = <0 RK_PA2 RK_FUNC_GPIO &pcfg_pull_up>;
        };
    };
};
```

2.2.4.1 RK3588 原理图跟PCIE20的dtsi对应关系

```
//PCIE20_0 -> pcie2x1l2
PCIE20_0_REFCLKP
PCIE20_0_REFCLKN
PCIE20_0_TXP
PCIE20_0_TXN
PCIE20_0_RXP
PCIE20_0_RXN

&combphy0_ps {
    status = "okay";
};

&pcie2x1l2 {   //pcie@fe190000
    reset-gpios = <&gpio1 RK_PB4 GPIO_ACTIVE_HIGH>;
    status = "okay";
};
```

```
//PCIE20_1 -> pcie2x1l0
PCIE20_1_REFCLKP
PCIE20_1_REFCLKN
PCIE20_1_TXP
PCIE20_1_TXN
PCIE20_1_RXP
PCIE20_1_RXN

&combphy1_ps {
    status = "okay";
};

&pcie2x1l0 {   //pcie@fe170000
    reset-gpios = <&gpio1 RK_PB4 GPIO_ACTIVE_HIGH>;
    status = "okay";
};

//PCIE20_2 -> pcie2x1l1
PCIE20_2_REFCLKP
PCIE20_2_REFCLKN
PCIE20_2_TXP
PCIE20_2_TXN
PCIE20_2_RXP
PCIE20_2_RXN

&combphy2_psu {
    status = "okay";
};

&pcie2x1l1 { //pcie@fe180000
    reset-gpios = <&gpio1 RK_PB4 GPIO_ACTIVE_HIGH>;
    status = "okay";
};
```

2.2.5 USB Wi-Fi 配置

首先强烈建议读下**USB**相关的参考如下文档：

docs\Common\USB\Rockchip_Developer_Guide_USB_CN.pdf.

Wi-Fi节点配置，分以下两种情况：

```
//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 = "rtl8188fu";
    //配置：WIFI_REG_ON 管脚：
    WIFI,poweren_gpio = <&gpio0 RK_PA2 GPIO_ACTIVE_HIGH>;
    //配置：WIFI_HOST_WAKE，如果有的话；
    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：
//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节点配置参考如下

```
/* 打开对应的控制器及PHY
 * 以下是RV1126/1109配置参考如下，其它芯片请根据实际情况修改；
 * 注意USB 有host/device/otg等模式；
 * 如果不知道如何配置，请在RK的redmine提一个问题：RKxxx芯片的USBXX口如何配置为host模式
 * （usb模块做device）；
 */

/* USB接在芯片的HOST脚 */
&u2phy_host {
    status = "okay";
};

&u2phy1 {
    status = "okay";
};

&usb_host0_ehci {
    status = "okay";
};

&usb_host0_ohci {
    status = "okay";
};

/* 另一种是接在主控的OTG脚 */
&u2phy1 {
    status = "okay";
};

&usbdrd {
    status = "okay";
};

&usbdrd_dw3 {
    status = "okay";
};

&u2phy_otg {
    status = "okay";
};
```

2.3 SDMMC(SD卡接口)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，找到如下两个配置，把&sdio改为&sdmmc，并disabled掉没有使用的节点；

特别注意：SDMMC_DET脚硬件上要拉低。

```
&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

2.4.1 WiFi内核配置

WiFi 驱动都已经移动到**external/rkwifibt/drivers/**目录下，参考SDK编译配置指南。

必要内核配置（根据自己的实际情况可以修改为module）

```
CONFIG_RFKILL=y
CONFIG_RFKILL_RK=y

CONFIG_MMC=y
CONFIG_PWRSEQ_SIMPLE=y

CONFIG_MMC_DW=y
CONFIG_MMC_DW_PLTFM=y
CONFIG_MMC_DW_ROCKCHIP=y

CONFIG_WIRELESS=y
CONFIG_WIRELESS_EXT=y
CONFIG_WEXT_CORE=y
CONFIG_WEXT_PROC=y
CONFIG_WEXT_PRIV=y
CONFIG_CFG80211=y
CONFIG_CFG80211_REQUIRE_SIGNED_REGDB=y
CONFIG_CFG80211_USE_KERNEL_REGDB_KEYS=y
CONFIG_CFG80211_DEFAULT_PS=y
CONFIG_CFG80211_CRDA_SUPPORT=y
# CONFIG_CFG80211_WEXT is not set
CONFIG_MAC80211=y
CONFIG_MAC80211_HAS_RC=y
CONFIG_MAC80211_RC_MINSTREL=y
CONFIG_MAC80211_RC_DEFAULT_MINSTREL=y
CONFIG_MAC80211_RC_DEFAULT="minstrel_ht"
CONFIG_MAC80211_STA_HASH_MAX_SIZE=0
```

2.4.2 蓝牙内核配置

- 新思(正基)和英飞凌(海华/CY)的模块使用内核的默认**CONFIG_BT_HCIUART** 驱动；
- Realtek使用自己的hci uart/hci_usb驱动，对应的驱动源码目录为:

```
external\rkwifibt\realtek\bluetooth_uart_driver
external\rkwifibt\realtek\bluetooth_usb_driver
```

使用ko方式加载，因此使用Realtek蓝牙时一定要把内核的**CONFIG_BT_HCIUART 配置关掉！**

`#CONFIG_BT_HCIUART is not set`

- 内核config配置

```
CONFIG_BT=y
CONFIG_BT_BREDR=y
CONFIG_BT_RFCOMM=y
# CONFIG_BT_RFCOMM_TTY is not set
# CONFIG_BT_BNEP is not set
CONFIG_BT_HIDP=y
# CONFIG_BT_HS is not set
CONFIG_BT_LE=y
CONFIG_BT_DEBUGFS=y
CONFIG_BT_HCIUART=y
CONFIG_BT_HCIUART_H4=y
```

3. Wi-Fi/BT框架及编译运行说明

3.1 编译涉及的文件

Wi-Fi/BT所涉及的文件说明如下：

```
external/rkwifibt/drivers/
bcmdhd                              #新思/正基模组通用驱动
infineon                            #英飞凌（cypress海华）模组通用驱动
rtlxxx                              #realtek模组不通用，每个型号都有单独的驱动
bluetooth_uart_driver               #RTK的蓝牙uart接口驱动
bluetooth_usb_driver                #RTK的蓝牙USB接口驱动
bin/arm*/                           #预置厂商的bin工具，wl正基模块配置工具，rtwpriv rtk模块
的测试工具
conf/*conf                          #预置的wpa_supplicant.conf/dnsmasq.conf用于sta/ap模
式
tools/brcm_tools rtk_hciattach      #各家的蓝牙初始化工具
debian wifibt-init.service          #debian系统的组件
S36wifibt-init.sh script/*sh        #Buildroot系统wifibt初始化脚本，包括连接测试等等
firmware/*                          #每个厂家对应的各个型号的WiFi/BT的firmware文件
```

正基/英飞凌各个型号所对应的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/drivers/bluetooth_uart_driver/     #蓝牙uart驱动
external/rkwifibt/drivers/bluetooth_usb_driver/      #蓝牙usb驱动
external/rkwifibt/tools/rtk_hciattach/               #蓝牙初始化程序

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

3.2 编译规则

> 建议开发人员仔细阅读下面几个编译脚本！

对应的编译规则文件，分两部分：

- 驱动编译

```
/*
 * 包括驱动的编译，WiFiBT的ko及firmware拷贝：
 * 1. ./build.sh 或 ./build.sh wifibt会调用到下面的脚本
 * 2. 脚本会去编译external/rkwifibt/drivers/目录下的驱动，并拷贝相应的firmware文件到文件
系统
 */
device/rockchip/common/scripts/mk-wifibt.sh
device/rockchip/common/scripts/post-wifibt.sh
```

- 工具编译（包括**brcm_patchram_plus1/rtk_hciattach**）

```
#编译规则
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

#蓝牙初始化工具编译及拷贝，通过./build.sh 或 make rkwifibt-rebuild更新
external/rkwifibt/meson.build
```

> 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涉及的文件及其位置

> 开发人员要熟悉**Wi-Fi/BT**使用的的文件及位置，当遇到**Wi-Fi/BT**启动异常的问题时，需确认**WiFiBT**的**firmware/config**文件是否存在，以及是否跟蓝牙型号相匹配。

- **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
```

经过上面章节的编译规则，对应的文件被拷贝到工程的output目录：**(system/vendor是同一个目录的链接)**

```
buildroot/output/rockchip_rk3xxxx/target/
#实际烧录到设备后，可以看到如下文件，否则排查配置及编译问题!!!
$ ls
/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/firmware/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
├── bluetooth_usb_driver    #RTL8821CU的蓝牙USB驱动，被编译成rtk_btusb.ko
├── 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
```

经过编译，对应的文件被拷贝到工程的output目录：

```
buildroot/output/rockchip_rk3xxxx/target/

#实际烧录到设备后，可以看到如下文件，否则排查配置及编译问题!!!
$ ls
/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 编译更新

参考SDK配置指南，配置正确的型号，编译指令如下：

```
#规则：优先编译命令行指定的参数，如果没用指定参数则会从BoardXXX.mk获取
$ ./build.sh

#或者
$./build.sh wifibt

#打包
$ ./build.sh rootfs && ./build.sh firmware
```

> 注意：观察./build.sh wifibt的输出，可以看到整个编译过程。

3.5 Wi-Fi/BT运行说明

开发人员必须要熟悉下面三个脚本的内容，对排查问题至关重要！

开机运行Wi-Fi/BT规则相关规则如下：

```
#启动脚本流程：
#/etc/init.d/S36wifibt-init.sh
#  -> wifibt-init.sh
#    -> wifibt-util.sh
external/rkwifibt/scripts/S36wifibt-init.sh

external/rkwifibt/scripts/
#根据VID/PID自动识别Wi-Fi/BT的型号
wifibt-util.sh

#根据上面识别的型号加载WiFi的ko并初始化蓝牙
wifibt-init.sh #start_wifibt/start_bt_brcm/start_bt_rtk_usb&uart etc.
```

4. Wi-Fi/BT软硬件功能测试

4.1 Buildroot系统

4.1.1 Wi-Fi STA测试

加载驱动，正常情况下ifconfig -a 会看到wlan0

`insmod bcmdhd.ko/RTL8723DS.ko`

打开WiFi

`ifconfig wlan0 up`

启动wpa_supplicant

`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，好：-20到-50，偏弱但可以接受：-50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Wi-Fi/BT硬件RF指标。

4.1.3 连接路由器

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

#连接信息
wpa_cli -i wlan0 -p /var/run/wpa_supplicant status
bssid=04:42:1a:b4:e7:7c
freq=5805
ssid=wwc3
id=0
mode=station
wifi_generation=6
pairwise_cipher=CCMP
group_cipher=CCMP
key_mgmt=SAE
pmf=2
mgmt_group_cipher=BIP
sae_group=19
sae_h2e=0
sae_pk=0
wpa_state=COMPLETED
ip_address=192.168.50.169
address=d4:9c:dd:f5:52:c4
uuid=47ed7266-da34-5c0b-884a-4fcc17648d87
ieee80211ac=1
```

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
```

正基/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
echo 0 > /proc/bluetooth/sleep/btwrite
sleep 1
echo 1 > /sys/class/rfkill/rfkill0/state         #上电
echo 1 > /proc/bluetooth/sleep/btwrite
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
echo 0 > /proc/bluetooth/sleep/btwrite
sleep 1
echo 1 > /sys/class/rfkill/rfkill0/state         #上电
echo 1 > /proc/bluetooth/sleep/btwrite
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
echo 0 > /proc/bluetooth/sleep/btwrite
sleep 2
echo 1 > /sys/class/rfkill/rfkill0/state  # 上电
```

```
echo 1 > /proc/bluetooth/sleep/btwrite
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               -All-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(WOWLAN)

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 网卡桥接功能

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
```

4.8 Wi-Fi/BT硬件RF指标

4.8.1 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余里)
	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余里)
	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

并整理上述测试到一份正式的报告。

4.8.2 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。
- 其它如南方硅谷/高拓/爱科微平台测试工具和方法请找对应的模块厂或原厂支持。

4.8.2.1 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
echo 0 > /proc/bluetooth/sleep/btwrite
sleep 1
echo 1 > sys/class/rfkill/rfkill0/state
echo 1 > /proc/bluetooth/sleep/btwrite

# 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]
```

4.8.2.2 新思/英飞凌芯片测试

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

#复位
echo 0 > /sys/class/rfkill/rfkill0/state
sleep 1
echo 1 > /sys/class/rfkill/rfkill1/state
sleep 1

#设置更新测试firmware
```

```
echo /vendor/etc/firmware/fw_bcmdhd_mfg.bin >
/sys/module/bcmdhd/parameters/firmware_path
sleep 1

#重启wlan0
ifconfig wlan0 down
ifconfig wlan0 up
sleep 1

#确认是否进入测试模式
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）
```

4.9 Wi-Fi 性能测试

使用iperf测试性能，请注意：

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；

测试指令

```
TCP下行：
    RK: iperf -s -i 1
    PC: iperf -c xxxxxxx(RK ipaddr) -i 1 -w 2M -t 120

TCP上行：
    PC: iperf -s -i 1
    RK: iperf -c xxxxxxx(PC ipddr) -i 1 -w 2M -t 120

UDP下行：
    PC: iperf -s -u -i 1
    RK: iperf -c xxxxxxx(RK ipaddr) -u -i 1 -b 100M -t 120

UDP上行
    PC: iperf -s -u -i 1
    RK: iperf -c xxxxxxx(PC ipddr) -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
```

5. Wi-Fi/BT问题排查

重要：务必先看log，根据log的异常进行排查！！！

WiFi/BT大概分为3类问题：

- SDIO/USB/PCIE设备识别不到；
- 识别到SDIO卡，但firmware加载失败；
- 性能问题，连不上/扫描个数少/概率断开等等；

5.1 Wi-Fi识别流程简述

SDIO 接口：

开机时kernel MMC框架会去识别SDIO 卡，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SDIO_CLK会以400/300/200K的低频给WiFi模块发命令，获取WiFi SDIO基本的信息，比如是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**提高SDIO_CLK频率到相应的高频50M/150M**，然后初始化基本完成，**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
# 解析dts的mmc-pwrseq节点，获取WL_REG_ON
dwmmc_rockchip ff4a0000.dwmmc: allocated mmc-pwrseq
# 低频初始化
mmc_host mmc0: Bus speed (slot 0) = 400000Hz (... actual 400000HZ div = 0)
# 高频工作模式
mmc_host mmc0: Bus speed (slot 0) = 50000000Hz (... actual 50000000HZ div = 0)
# SDIO 2.0
mmc0: new high speed SDIO card at address 0001
# SDIO 3.0
mmcx: new ultra high speed SDR104 SDIO card at address 0001
```

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！

5.2 Wi-Fi问题排查

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。

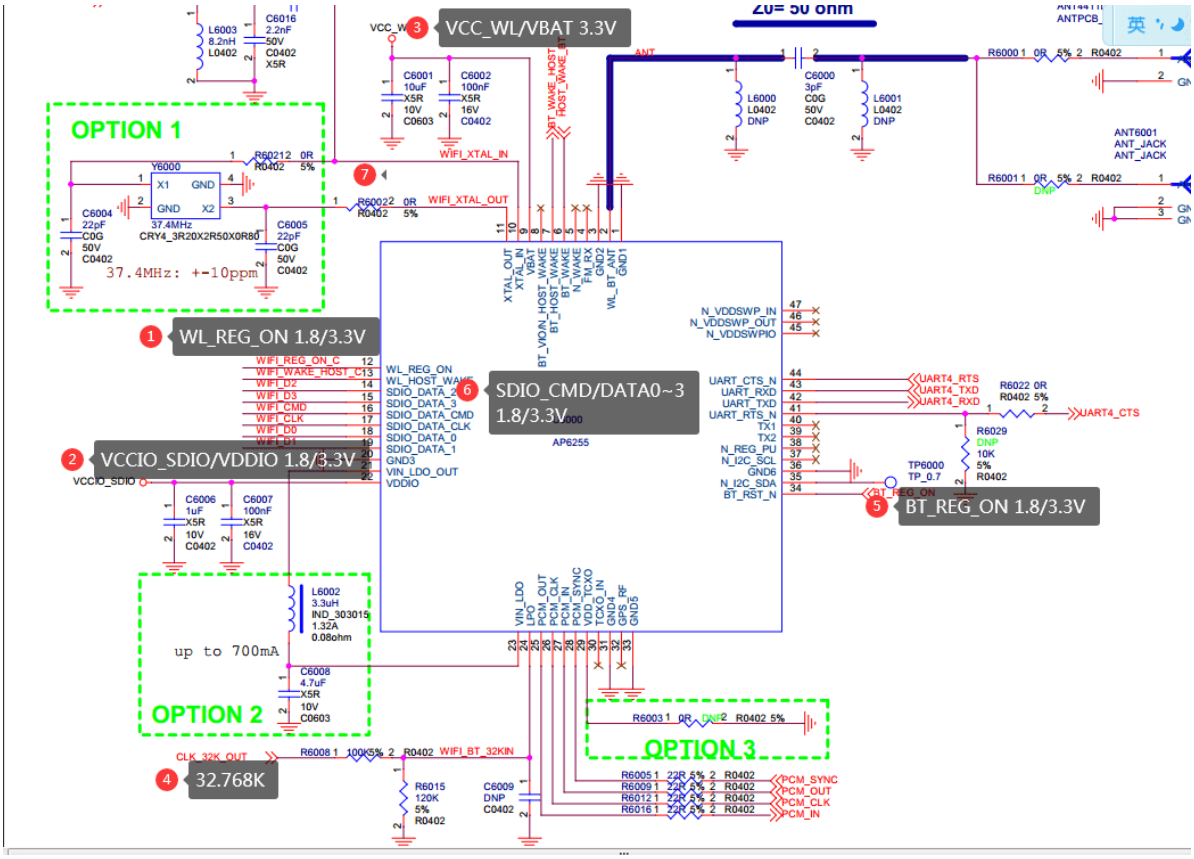
5.2.1 Wi-Fi 异常SDIO识别不到

首先强烈建议再次详细检查下第2章节的DTS/KERNEL配置！！！

按照以下步骤对照内核LOG依次排查：

5.2.1.1 测电压

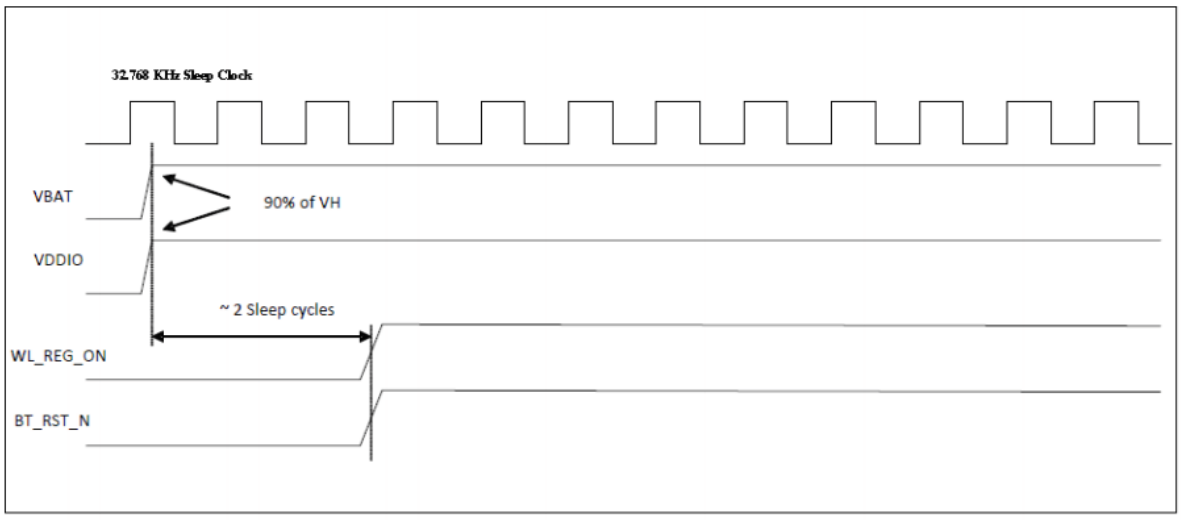
按照标号依次测量对应PIN脚的电平状态及CLK频率是否正确（注：SDIO3.0模式必须为1.8V）



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：

5.2.1.2 测时序

注意：下面是AP正基模块的时序要求，要看下实际使用的模块的规格书是时序图是如何要求的，二是特别注意下VDDIO的时序。



5.2.1.3 DTS的WL_REG_ON 配置错误

导致Wi-Fi使能脚拉不高

- 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；
- 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；

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

5.2.1.4 DTS的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!

5.2.1.5 VDDIO/SDIO/IO电源域

- 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。
- VDDIO：供电电压跟DTS电源域配置不匹配（参考IO电源域配置）；

5.2.1.6 SDIO_CLK 没有波形

- IO电源域设置错误可能会导致CLK测不到波形；
- CLK脚是否有加上拉电阻，去掉测试；
- 由于CLK波形需要设置触发模式进行抓取‘；

5.2.1.7 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
###
```

5.2.1.8 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
```

5.2.1.9 PCB走线质量/电容/电感不合适

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。

修改频率的方法：

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

5.2.1.10 io_domian配置

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
dhdsdio_membytes: FAILED to set window back to 0x18100000
```

5.2.1.11 WL_HOST_WAKE

WLAN to wake-up RK HOST

WL_HOST_WAKE PIN 脚或中断电平配置错误或者虚焊，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
```

5.2.1.12 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：

```
[   22.484301] mmc3: queuing unknown CIS tuple 0x80 (7 bytes)
[   22.598965] xxx 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；

5.2.1.13 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；

5.2.1.14 iomux 复用异常

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

5.2.1.15 其它

如果以上排查不出问题，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.

5.2.2 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WiFi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

5.2.3 Realtek Wi-Fi 注意事项

5.2.3.1 wlan0有识别到但扫描异常

- **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。

5.2.3.2 Realtek 支持SDIO3.0

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;
```

5.2.4 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
```

- 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）

```
[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
```

- Realtek模块扫描异常，比如扫描不到AP

```
[ 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**
  - Wi-Fi驱动加载过早导致异常，从log可以看到驱动在1秒内就被加载了，此时sdio/usb设备还没枚举到，检查内核配置章节；
  - 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；

5.2.5 SDIO Wi-Fi 运行一段时间后异常

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）

5.2.6 Wi-Fi 无法连接AP/断线/连接不稳定/连接时间慢

首先了解下连接流程，包含以下过程：

1. scan配置的ssid，如果一直扫描不到，则会重复扫描，直到扫描到对应的ssid；
2. 发起了连接及4次握手；
3. 获取dhcp；

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

首先提供板子的rf指标测试报告，如果没有请先测试基础的指标；其次如果是整机包含外壳，天线的OTA整机指标是否有测试，没有也请优先测试下。

确保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

5.2.7 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
```

5.2.8 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！

5.2.9 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
```

5.2.10 PING 不通或概率延迟很大

- RF/天线指标没测过；
- 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；

5.2.11 客户自定义修改

如果自己有修改驱动代码，请务必告知，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。

5.2.12 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;

5.2.13 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
}
```

5.2.14 iPhone问题

问题：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：

解决方式：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`

5.2.15 正基/英飞凌/海华/新思/模块问题

常见问题：

WIFI_REG_ON配置错误；

32.768K 没有出来或者信号不达标；

WIFI_WAKE_HOST：管脚配置错误或中断电平配置错误，导致初始化失败或dhd_dpc线程cpu占有率高；参考DTS配置

```
/* 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>;
    };
};
```

休眠时32.768K被关掉，或WIFI_REG_ON被关掉；

firmware/nvram固件不存在，在log的很明显能看到下载文件时出错；

正基(bcmdhd.ko)/海华(cywdhd.ko)模块驱动不能通用，不要配错驱动；

5.2.16 PCIe WiFi识别不到

请参考PCIE WiFi配置章节，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！

5.3 蓝牙问题

蓝牙测试项失败或 **rkwifibt-app-test BT**蓝牙功能异常，有下面几种情况，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** 参考蓝牙内核配置章节；
- DTS BT (**BT_RST_N**)电源使能PIN脚配置不对，参考蓝牙配置；
- 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 0 > /proc/bluetooth/sleep/btwrite

echo 1 > sys/class/rfkill/rfkill0/state #拉高BT_REG_ON, 用万用表测量对应的PIN
echo 1 > /proc/bluetooth/sleep/btwrite
```

- 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
```

6. Wi-Fi/BT新模块移植及驱动更新

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；**
- **对更新驱动的情况，下面有些步骤可以根据实际情况省略；**

6.1 Realtek模块

6.1.1 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

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，更新驱动的情况无需此操作。

```
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;
}
```

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)
{
    rockchip_wifi_power(1);
    rockchip_wifi_set_carddetect(1);
    return rtw_drv_entry();
}

void rockchip_wifi_exit_module_rtkwifi(void)
{
    rtw_drv_halt();
    rockchip_wifi_set_carddetect(0);
    rockchip_wifi_power(0);
}

//KO模式走这里
module_init(rockchip_wifi_init_module_rtkwifi);
module_exit(rockchip_wifi_exit_module_rtkwifi);E

//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);
```

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)) */
```

6.1.2 BT蓝牙部分

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"
```

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
```

6.2 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
```

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。

6.3 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-
```

7. 常见功能及配置说明

7.1 Debian及其他第三方系统Wi-Fi/BT适配说明

Debian/麒麟/UOS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，需要开启wpa_supplicant/bluez协议栈来支持蓝牙功能

```
# 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。

7.1.1 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
echo 0 > /proc/bluetooth/sleep/btwrite
sleep 0.2
echo 1 > /sys/class/rfkill/rfkill0/state   #打开BT电源，等同于rfkill unblock操作
echo 1 > /proc/bluetooth/sleep/btwrite
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,可以看到hci0节点
hciconfig -a

# 关闭蓝牙
echo 0 > /sys/class/rfkill/rfkill0/state   #关闭BT电源，等同于rfkill block操作
echo 0 > /proc/bluetooth/sleep/btwrite

#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；
killall brcm_patchram_plus1

#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；
```

7.1.2 Realtek模块适配示例

WiFi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

# realtek无需firmware/nvram文件，insmod执行时机根据系统要求进行调整；
insmod /ko_path/88xxxx.ko
#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
ifconfig -a

### 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

# 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
echo 0 > /proc/bluetooth/sleep/btwrite
sleep 0.5
echo 1 > /sys/class/rfkill/rfkill0/state #上电
echo 1 > /proc/bluetooth/sleep/btwrite
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
echo 0 > /proc/bluetooth/sleep/btwrite
sleep 0.5
echo 1 > /sys/class/rfkill/rfkill0/state #上电
echo 1 > /proc/bluetooth/sleep/btwrite
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,正常的话可以看到hci0
hciconfig -a
```

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
```

7.1.3 相关文件自动安装集成说明

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：

  方法一：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/

  方法二：直接修改**RK**平台的**Debian**打包脚本

  ```
  #Debain的版本
  debian/mk-rootfs-bullseye.sh
  或
  debian/mk-rootfs-buster.sh

  #新建相关文件夹
  sudo mkdir -p $TARGET_ROOTFS_DIR/system/lib/modules/
  sudo mkdir -p $TARGET_ROOTFS_DIR/lib/firmware/rtl/
  #拷贝相关二进制
  sudo cp /实际存放的bin目录/rtk_hcixxx or brcmxxx $TARGET_ROOTFS_DIR/usr/bin/
  #拷贝ko
  sudo cp /实际存放的ko目录/wifi.ko $TARGET_ROOTFS_DIR/system/lib/modules/
  #拷贝firmware
  sudo cp /实际存放的firmware的目录/fwXXX $TARGET_ROOTFS_DIR/vendor/etc/firmware/

  #最后重新打包即可
  ```

- 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；

7.2 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
```

7.3 Wi-Fi/BT MAC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)；

海华Wi-Fi模组，修改 Makefile，增加如下配置：

```
+    -DGET_CUSTOM_MAC_ENABLE
-    -DGET_OTP_MAC_ENABLE
```

7.4 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
```

- 透过proc方式`echo X > /proc/net/rtlxxx/wlan0/country_code`, 如: `echo CN > /proc/net/rtlxxx/wlan0/country_code`
- 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`

7.5 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：

kernel 4.19

```
--- 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);
        }
```

7.6 网络类排查步骤

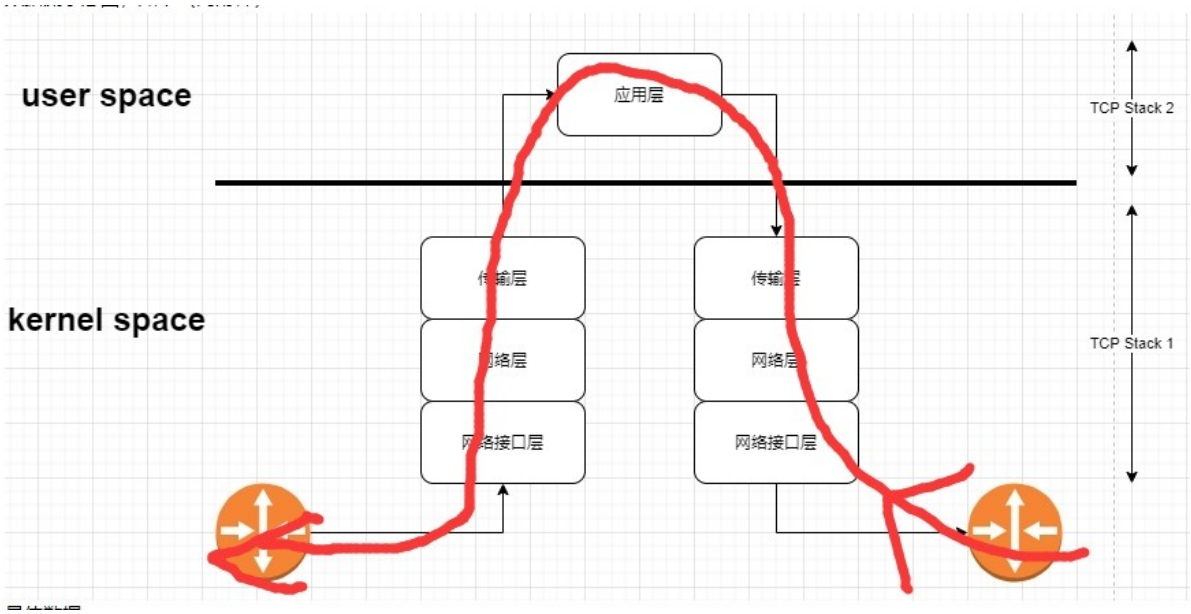
7.6.1 网络卡顿/速率不达标

```
1. 以太网：以太网换网线确认
2. 通过tcpdump抓报文确认是否有丢包
3. 如果是UDP+RTP的话，可以通过RTP序号来确
4. 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
5. 关掉gro确认
ethtool -K eth0 gro off
ethtool -S eth0/wlan0
6 cpu频率提到最高确认
echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
7. 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
8. busybox ifconfig看是否存在丢包统计
9.视频播放类：TCP丢包导致卡顿、UDP丢包会花屏
10.Wi-Fi或者以太网UDP测试丢包率异常，可做如下修改验证：
注意：每一个socket最大的用于数据发送的缓存(单位：字节)，HRTIMER打开以后，cpu经常被timer打断，相当于udp发送变慢了，buf就不会溢出了（可以看到1ms就会被timer打断3-4次）
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 1048576 > /proc/sys/net/core/rmem_max

11. 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.
echo 65535 >  /proc/sys/net/core/netdev_max_backlog
```

```
12. Apply network parameters for high data performance.
echo 327680 > /proc/sys/net/core/rmem_default
echo 8388608 > /proc/sys/net/core/rmem_max
echo 327680 > /proc/sys/net/core/wmem_default
echo 8388608  /proc/sys/net/core/wmem_max

echo 20480 > /proc/sys/net/core/optmem_max
echo 10000 > /proc/sys/net/core/netdev_max_backlog
echo "2097152 4194304 8388608" > /proc/sys/net/ipv4/tcp_rmem
echo "262144 524288 8388608" > /proc/sys/net/ipv4/tcp_wmem
echo "44259 59012 88518" > /proc/sys/net/ipv4/tcp_mem
echo "88518 118025 177036" > /proc/sys/net/ipv4/udp_mem
```

7.6.2 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。



7.7 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
```

7.8 驱动应用DEBUG 调试配置

7.8.1 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
```

7.8.2 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
SYSLOGD_ARGS=-n
改成
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
```

8. RV系列 IPC SDK Wi-Fi说明

8.1 配置说明

系统配置

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-usbwifi.config
+export RK_KERNEL_DEFCONFIG_FRAGMENT="rv1106-xxx.config rv1106-usbwifi.config"
# SDIO接口WIFI添加如下配置：rv1106-sdiowifi.config
+export RK_KERNEL_DEFCONFIG_FRAGMENT="rv1106-xxx.config rv1106-sdiowifi.config"

# 使能WIFI功能，支持SSV6X5X/RTL8189FS/RTL8188FTV/ATBM603X/ATBM6441/HI3861L
+export RK_ENABLE_WIFI_CHIP="RTL8189FS"
+export RK_ENABLE_WIFI=y
```

DTS 配置参考DTS 配置章节

注意：USB模块一般都是长供电，无需电源控制；对应SDIO模块一般都会有个WL_REG_ON（不同模块叫法不同），需要按照dts配置具体的gpio，特别要注意的是kernel的config需要打开如下两个配置：

> CONFIG_RFKILL=y
>
> CONFIG_RFKILL_RK=y

编译

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；对应实际运行目录为：

> /oem/usr/ko/wifi_xxx.ko
>
> /usr/bin/wpa_supplicant
>
> /usr/bin/wpa_cli

8.2 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 USB
rtl8189fs         #RTL8189FS/FTV SDIO
ssv6x5x           #SSV6155P USB
atbm              #atbm620x SDIO
atbm6441          #atbm6441低功耗WiFi SDIO
insmod_wifi.sh    #驱动ko加载脚本
```

目前支持RTL8188FU/8189FS、Hi3861L、SSV6155P、ATBM6XX、ATBM6441等；

wpa_supplicant的源码目录：

```
#wpa_supplicant
wpa_supplicant-2.6

#默认的配置文件
wpa_supplicant.conf
```

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 （低功耗WiFi参考低功耗章节）
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。

8.3 应用开发

RK默认集成了简化WiFi应用的Demo：`project/app/wifi_app/wifi/`

注意：**Demo**仅供参考保证稳定性，如需产品化需自行完善并增删相关功能。

使用方法：

```
//首先ps看下是否有rkwifi_server是否启动，如果没有首先要启动后台服务
# rkwifi_server start &

//启动客户端
# rkwifi_server -h
[Usage]:
        "rkwifi_server close"
        "rkwifi_server scan"
        "rkwifi_server connect ssid password"
        "rkwifi_server disconnect"
        "rkwifi_server getinfo"
        "rkwifi_server deepsleep [ONLY LOW POWER]"
        "rkwifi_server reconnect ssid"
        "rkwifi_server forget ssid"
        "rkwifi_server getSavedInfo"
        "rkwifi_server cancel"
        "rkwifi_server reset"
        "rkwifi_server onoff_test"
        "rkwifi_server ap_cfg"
        "rkwifi_server ap_cfg_clr"
        "rkwifi_server setpir <0/1> = disable/enable PIR"
        "rkwifi_server ota"
        "rkwifi_server start_kp <ip:port:expire>"
        "rkwifi_server stop_kp"
```

8.4 第三方应用移植说明

参考IPC SDK的相关文档，里面有详细说明。

8.5 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makefile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。

8.6 问题排查说明/RF测试

参考上面的WiFi/BT问题排查章节和RF指标章节。

RF测试请注意：

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。

9. RV系列低功耗WiFi开发指导

详细文档参考：https://docs.qq.com/doc/p/ad70d285112c58d5aa09b08eb2d93cf96d85c591

10. 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。

RKWIFIBT-APP应用开发及测试指导参考如下：

https://docs.qq.com/doc/p/a9dd00348639929fc080d20dbbdc4feea89c4983

11. FTP地址

FAE资料

> FTP地址：ftp://www.rockchip.com.cn
> 账号名：rkwifi
> 密码：Cng9280H8t
>
> /11-Linux平台/WIFIBT 开发文档/
>
> /11-Linux平台/WIFIBT 编程接口/
>
> /11-Linux平台/RV1106_03平台/