Rockchip Linux Wi-Fi/BT Developer Guide

ID: RK-KF-YF-381

Release Version: V6.1.1

Release Date: 2022-01-11

Security Level: □Top-Secret □Secret □Internal ■Public

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

This document is going to introduce Wi-Fi/BT development, porting and debugging on Rockchip Linux platform.

Product Version

Chipset	Kernel Version
RK3566/RK3568/RK3399/RK3326/RK3288/RK3308/RV1109/RV1126/PX30	Linux 4.4/4.19
RK3588	Linux 5.10

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Hardware development engineers

Revision History

Version	Author	Date	Change Description
V6.0.1	XY	2020-08-25	Added new module porting instructions, building instructions, RF test examples, P2P bridge function, more detailed troubleshooting instructions, etc.
V6.0.2	XY	2021-01-15	Add more detailed configuration and debug introduction;
V6.0.3	XY	2021-11	Add USB Wi-Fi/BT introduction.
V6.1.0	XY	2021-12-27	Add Debian/Kylin and other system adaptation introduction; add BT 5.0 function introduction, added CY chip low power mode.
V6.1.1	XY	2022-01-11	Added Wi-Fi SDIO/USB/PCIE interface identification process and SDIO timing brief introduction; Added network performance troubleshooting introduction; Added Buildroot system open source package update method;

Contents

1. Quick Start Guide

- If WiFi/BT cannot be recognized, **please check the abnormal log carefully first**, and check the type of problem according to Chapter 7 in order!
- About Wi-Fi/BT abnormalities please refer to Chapter 2/3/7;
- About Wi-Fi/BT files/compilation/update introduction, please refer to Chapter 3;
- About Wi-Fi driver KO loading, please refer to Chapter 3.4;
- About Wi-Fi/BT porting/driver update, please refer to Chapter 8;
- Please refer to Chapter 12 for Wi-Fi/BT application development;
- About Wi-Fi country code settings, please refer to Chapter 11.7;
- About SDK compilation configuration, please refer to Chapter 2.1;
- About RV1126/RV1109 Wi-Fi, please refer to chapter 7.2.4;
- About USB interface Wi-Fi/BT, please refer to Chapter 8.1 for porting, Chapter 2/3 for configuration, and Chapter 7 for troubleshooting;
- About PCIE Wi-Fi, please refer to Chapter 2.2.5;
- About performance issues, please refer to Chapter 5/7;
- For Debian/UOS/Kylin system Wi-Fi/BT adaptation introduction, please refer to Chapter 9
- **Note: in the Wi-Fi/BT configuration chapter , the DTS and kernel configurations have nothing to do with Buildroot or Debian systems, it can be used in both of the two system!**

2. Wi-Fi/BT Configuration

2.1 Buildroot SDK Compilation and Configuration Guide

Before configuring Wi-Fi/BT, firstly, you must configure the board-level files, otherwise the Wi-Fi/BT configuration will not take effect or cause other problems; the configuration file is located in the device/rockchip/rkxx (rkxx represents the chip platform)/ directory, **the following will take RV1126 for example (other platforms can follow the same rules):**

```
#Select board-level configuration, such as selecting BoardConfig.mk (this file
should be selected according to your actual situation, you can refer to the quick
start document of the SDK for details)
#In addition, the earlier version SDKs do not support this command, so this step
is not required.
./build.sh device/rockchip/rv1126_rv1109/BoardConfig.mk

#Checking the contents of the BoardConfig.mk file, it includes(important):
#Kernel defconfig corresponds to the defconfig file used by the kernel,
corresponding to kernel/arch/arm/configs/rv1126_defconfig
export RK_KERNEL_DEFCONFIG=rv1126_defconfig
#Kernel dts corresponds to the DTS used by kernel
export RK_KERNEL_DTS=rv1126-evb-ddr3-v13
#defconfig of Buildroot, corresponding to this file
buildroot/configs/rockchip_rv1126_rv1109_defconfig
export RK_CFG_BUILDROOT=rockchip_rv1126_rv1109
```

```
#Select the defconfig of buildroot, which is the RK_CFG_BUILDROOT configuration
above. After this step, rkwifibt and other modules can be compiled
separately,such as make rkwifibt/deviceio_release and other command
source envsetup.sh rockchip_rv1126_rv1109

#buildrootConfiguration
make menuconfig
#Select the corresponding configuration and save it to the rootfs configuration
file
#./buildroot/configs/rockchip_rv1126_rv1109_defconfig
make savedefconfig

#Kernel configuration, take 32-bit as an example, note that directories of arm64
are different
cd kernel
make ARCH=arm rv1126_defconfig # it is the RK_KERNEL_DEFCONFIG mentioned above
make ARCH=arm menuconfig
make ARCH=arm savedefconfig
cp defconfig arch/arm/configs/rv1126_defconfig #Update kernel configuration
```

2.2 DTS Configuration

2.2.1 Wi-Fi Configuration

The following items are included in Wi-Fi/BT hardware pin configurations:

Remember to configure according to the schematic, and make sure that the dts/dtsi contains the following nodes!

Note:

- For Wi-Fi with **SDIO interface**: WL_REG_ON is managed and controlled by the **sdio_pwrseq** node, **no need** to add WIFI, poweren_gpio configuration under the wireless-wlan node **repeatedly**;
- For Wi-Fi with **USB/PCIE** interface: **need** to add WIFI, poweren_gpio corresponding configuration of WL_REG_ON GPIO under the wireless-wlan node;

```
/* Only used for SDIO interface Wi-Fi configuration: WIFI_REG_ON: power enble PIN
of Wi-Fi */
sdio_pwrseq: sdio-pwrseq {
        compatible = "mmc-pwrseq-simple";
        pinctrl-names = "default";
        pinctrl-0 = <&wifi_enable_h>;
        /* Special attention: WIFI_REG_ON GPIO_ACTIVE configuration is exactly
the opposite of enable state: LOW is High effective, High is low effective;
Remember that: this configuration is mutually exclusive with the following WIFI,
poweren_gpio, and cannot be configured at the same time! ! !*/
        reset-gpios = <&gpio0 RK_PA2 GPIO_ACTIVE_LOW>;
};
/* Only used for SDIO interface Wi-Fi configuration: pinctrl configuration of
WIFI_REG_ON pin */
&pinctrl {
        sdio-pwrseq {
                wifi_enable_h: wifi-enable-h {
                        rockchip,pins =
```

```
                        /* Corresponds to the WIFI_REG_ON above, turn off pull-up
and pull-down to prevent it from being pulled high or low*/
                        <0 RK_PA2 RK_FUNC_GPIO &pcfg_pull_none>;
                };
        };
};

/* Only used for SDIO interface Wi-Fi configuration: SDIO interface node */
&sdio {
        max-frequency = <150000000>; /* The maximum frequency of the sdio
interface, adjustable */
        bus-width = <4>;             /* 4-line mode,can be changed to 1-line mode
*/
        sd-uhs-sdr104;                      /* Support SDIO3.0 */
        ……
        status = "okay";
};

/* WIFI node*/
wireless-wlan {
        compatible = "wlan-platdata";
        rockchip,grf = <&grf>;
        /* Note: If you finds that the Wi-Fi module does not have 32.768K
waveform when debugging, and the hardware is provided by RK PMU, open the clock
property below and fill in according to the actual PMU model used. Otherwise the
SDIO/Wi-Fi can not work.*/
        //clocks = <&rk809 1>; //IF RK809 is used, only one can be configured
        clocks = <&hym8563>;  //IF hym8563 is used, only one can be configured
        clock-names = "ext_clock";
        /* Fill in according to the model you actually used */
        wifi_chip_type = "ap6255";
        /* WIFI_WAKE_HOST: The PIN of the Wi-Fi interrupt notification to
controller. Special attention: please check the hardware connection between the
Wi-Fi pin and the controller pin. If they are directly connected, it is
GPIO_ACTIVE_HIGH; if a reverse tube is added between them, it should be changed
to low-level GPIO_ACTIVE_LOW trigger*/
        WIFI,host_wake_irq = <&gpio0 RK_PA0 GPIO_ACTIVE_HIGH>;
        /* Note that the Wi-Fi with USB/PCIE interface needs to add this
configuration, and the corresponding WIFI PIN should be enable,and no need to
configure nodes such as sdio_pwrseq/sdio*/
       //WIFI,poweren_gpio = <&gpio0 RK_PA2 GPIO_ACTIVE_HIGH>;
       status = "okay";
};

/* pinctrl configuration of WIFI_WAKE_HOST pin*/
wireless-wlan {
       /omit-if-no-ref/
       wifi_wake_host: wifi-wake-host {
              /* Note that the wake host pin of regular Wi-Fi is triggered by a
high level, so it must be configured as a pull-down by default. If the customer's
hardware design is reversed, it must be changed to a pull-up. In short, it must
be initialized with the opposite state with trigger circuit.*/
              rockchip,pins = <0 RK_PA0 0 &pcfg_pull_down>;
       };
};
```

```
/* USB Wi-Fi: please refer to the following for the USB part configurations, and
modify according to the actual situation. Pleas refer to the USB related
documents in the doc/ directory for configuration */
&u2phy_host {
        status = "okay";
};

&u2phy1 {
        status = "okay";
};

&usb_host0_ehci {
        status = "okay";
};

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

2.2.2 Bluetooth Configuration

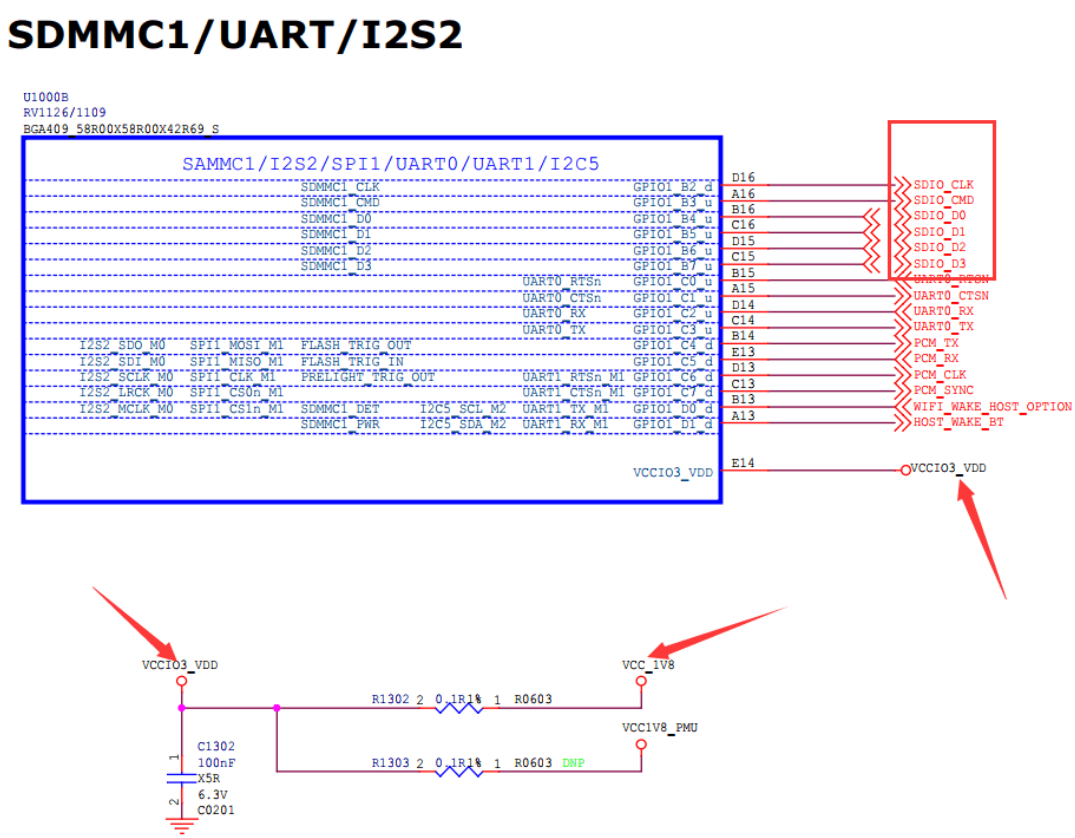
The following UART related items should be configured as **the corresponding PIN of actual used UART port**. Note that RTS/CTS pin must be connected according to SDK design **(Please refer to the UART description in Chapter 7.2 for details)**. So many abnormalities reported by customers are all because these two pins are not connected, causing initialization abnormal, the following supposes that Bluetooth uses UART4:

```
/* Bluetooth node: pay attention to the following UART configuration: the name of
uart4_xfer/uart4_rts/uart4_ctsn may be different between different platform, you
should find the corresponding uart in the dts/dtsi of the corresponding chip
platform, such as uart4_ctsn of some platforms are named uart4_cts. */
wireless-bluetooth {
        compatible = "bluetooth-platdata";
        /* It is used to configure the RTS pin of UART of correspoinding
controller */
        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 is Bluetooth power switch */
        BT,power_gpio    = <&gpio4 RK_PB3 GPIO_ACTIVE_HIGH>;

        /* Linux platform: the following two configurations do not need to be
configured */
        //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";
};
/* Open the corresponding UART configuration. */
&uart4 {
        pinctrl-names = "default";
        /*The Configuration here corresponding to the TX/RX/CTS PIN of the UART
of the controller, and the RTS PIN is not needed to configure*/
        pinctrl-0 = <&uart4_xfer &uart4_ctsn>;
        status = "okay";
```

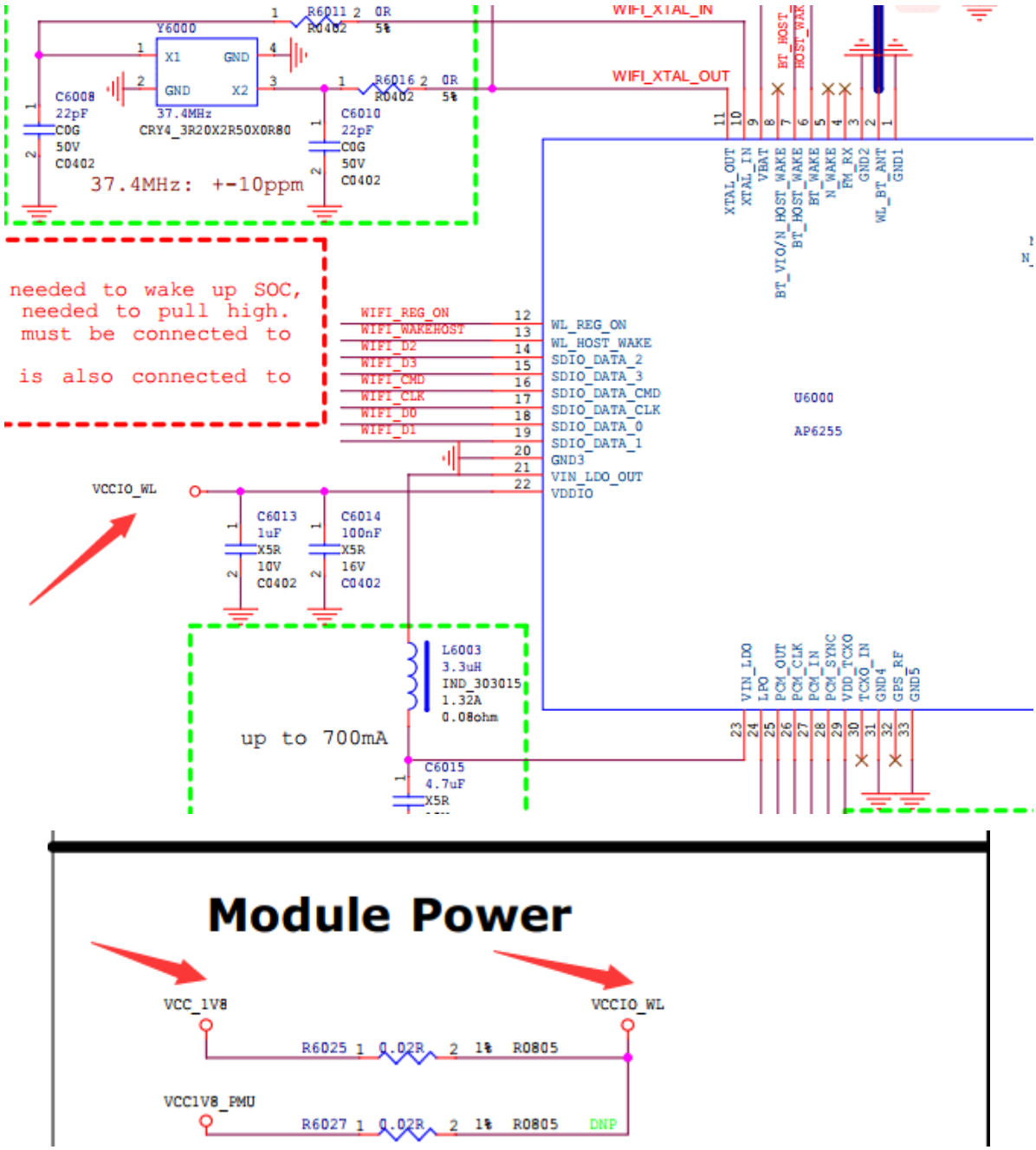
```
};
```

2.2.3 IO Power Domain Configuration

The Wi-Fi circuit consists of two parts of power supply, one is the IO of the controller: SDIO_CLK/CMD/D0~D3, which requires external power supply, such as VCCIO3_VDD in the figure below, you can see that it is powered by VCC_1V8;



The other is the power supply of the IO of the Wi-Fi module. For example, the VCCIO_WL in the figure below is also powered by VCC_1V8. **The power supply of the two parts must be the same, otherwise it will lead to Wi-Fi abnormal**; for support **SDIO3.0 Ultra-High speed Mode (UHS)**, **it must be powered by 1.8V**, but for Wi-Fi modules that only support SDIO2.0 high-speed mode, both power supply of 1.8 or 3.3V can be used, remember to keep matching ;



The above descriptions are all hardware requirements. The following is going to introduce the software configuration: From the above figure, it can be seen that the VCCIO3_VDD supplying power to the controller IO is 1.8V, then the io_domains corresponding to dts/dtsi need to be configured, and VDDIO3_VDD corresponds to the software vccio3-supply, while vccio3-supply is powered by vcc_1v8, the corresponding configuration is:

```
//Note that the names of io_domains for each platform are different, and some are named as &pmu_io_domains;
&io_domains {
// Or
&pmu_io_domains {
        /* VDDIO3_VDD refers to the voltage of vcc_1v8, if the hardware is connected to 3.3v, change it to vcc_3v3. Note: The names of vcc_1v8/vcc_3v3 should be adjusted according to the actual dts/dtsi */
        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 with power supply of 1.8v */
        regulator-min-microvolt = <1800000>;
        regulator-max-microvolt = <1800000>;
        vin-supply = <&vcc_io>;
};
```

2.2.4 The 32.768K Configuration

Modules of Azurewave/AMPAK should be supplied with external 32.768k, while Realtek's modules are packaged internally, so only COB chips need supplied externally.

Generally, 32k is powered by the RK8XX PMU for Wi-Fi, and 32k is turned on by default in the PMU. If it is not turned on, the following configuration should be added:

```
wireless-wlan {
        compatible = "wlan-platdata";
        rockchip,grf = <&grf>;
        /* rk809 must be changed to the actual model you used,If it still can't
be found after adding it, you need to commit a redmine with pmu output 32 clk */
+        clocks = <&rk809 1>;
+        clock-names = "clk_wifi";
};
```

Note: if RK's PMU are not used, this configuration is not needed.

2.2.5 PCIE Wi-Fi Configuration

```
// Which phy node is connected
&pcixxx {
        /* This item is to set the PERST# signal of the PCIe interface; whether
it is a slot or a soldered device, please find this pin on the schematic diagram
and configure it correctly, otherwise the link will not be created. */
        ep-gpios = <&gpio3 13 GPIO_ACTIVE_HIGH>; //RK3399 platform
        reset-gpios = <&gpio4 RK_PA5 GPIO_ACTIVE_HIGH>; //RK356X/3588 platform

        /* The following three configurations are optional, they are used to
configure the 1V8/3V3 power supply of PCIe peripherals; they are a board-level
configuration items that need to be controlled and enabled for PCIe peripheral
power supply */
        vpcie3v3-supply = <&vdd_pcie3v3>;
        vpcie1v8-supply = <&vdd_pcie1v8>;
        vpcie0v9-supply = <&vdd_pcie1v8>;

        num-lanes = <4>;
         pinctrl-names = "default";
         pinctrl-0 = <&pcie_clkreqn_cpm>;
        status = "okay";
};

//phy version
&pcieXXphy {
    status = "okay";
```

```
};

//Wi-Fi node configuration
wireless_wlan: wireless-wlan {
        compatible = "wlan-platdata";
        wifi_chip_type = "ap6275p";
        ... ...
        WIFI,host_wake_irq = <&gpio0 RK_PA0 GPIO_ACTIVE_HIGH>;
        WIFI,poweren_gpio = <&gpio0 RK_PC7 GPIO_ACTIVE_HIGH>;
        status = "okay";
};

// Pull up a PIN by default, such as GPIO3D5
// Method one:
&pinctrl {
+       pinctrl-names = "default";
+       pinctrl-0 = <&pcie_3v3>;
+       pcie_vbat {
+               pcie_3v3: pcie-vbat {
+                       rockchip,pins = <3 RK_PD5 RK_FUNC_GPIO &pcfg_pull_up>;
+               };
+       };
+
};
```

For detailed configuration of PCIE, please refer to related documents in the SDK directory docs\Common\PCIe.

2.3 SDMMC Interface Connected to Wi-Fi Chip

For some special requirements, the Wi-Fi chip needs to be connected to SDMMC interface, so the configuration will be modified as follows:

Find the following two configurations: change &sdio to &sdmmc, and disable unused nodes;

```
&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 can only be referenced by one
node, should not be referenced by sdmmc and sdio at the same time!
        status = "okay";
};

#RK3328 platform:
&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 Kernel Configuration

```
#kernel directory
make menuconfig ARCH=armXX  # please refer to Chapter 2.1 for defconfig usage
```

2.4.1 Wi-Fi Configuration

```
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 driver can be built into kernel or ko mode,**Remember that only one of the following two configurations can be selected, otherwise Wi-Fi cannot be loaded!**

```
# KO configuration is as follows, the following two are mutually exclusive
[*]   build wifi ko modules
[ ]   Wifi load driver when kernel bootup

# buildin configuration is as follows, the following two are mutually exclusive
[ ]   build wifi ko modules
[*]   Wifi load driver when kernel bootup
```

- Only one model of buildin can be selected, Realtek modules and ap6xxx modules cannot be selected as y at the same time, you can only choose one of them;
- ap6xxx and cypress are also mutually exclusive, you can only choose one and if you choose ap6xxx, the cypress configuration will disappear automatically, and if you remove the ap configuration, cypress will appear automatically;

  You can select multiple Wi-Fi in ko mode.

After the configuration, the corresponding defconfig should be saved, please refer to Chapter 2.1 for details.

2.4.2 Bluetooth Configuration

Both of Azurewave/AMPAK modules use the CONFIG_BT_HCIUART driver of kernel by default, while Realtek uses its own hci uart driver. The source code directory is as follows: `external\rkwifibt\realtek\bluetooth_uart_driver`, and they are loaded in ko mode, so when Realtek is used, don't forget to remove the **CONFIG_BT_HCIUART** configuration of kernel!

```
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]
```

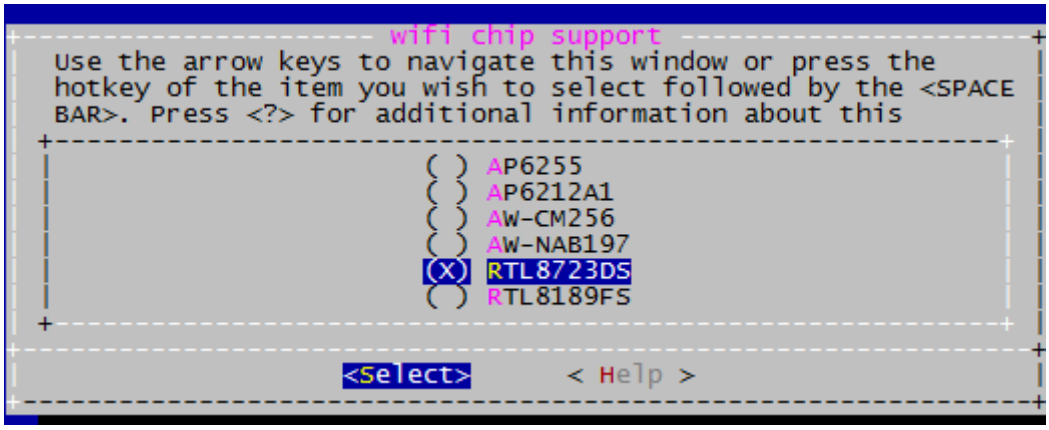
After the configuration, the corresponding defconfig should be saved, please refer to Chapter 2.1 for details.

2.5 Buildroot Configuration

Choose the corresponding configuration according to the Wi-Fi used actually, which should be consistent with the kernel configuration:

Execute in the root directory: make menuconfig (refer to Chapter 2.1 for compilation environment), and then search for rkwifibt to enter the following configuration interface:

```
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
```



For modules with Bluetooth, the corresponding ttySX and the hardware uart port should be configured:

```
--- rkwifibt
      wifi chip support (AP6255)  --->
(ttyS4) bt uart
```

Note: after the configuration, the corresponding defconfig should be saved, please refer to Chapter 2.1 for details, remember to compile and update after saving: please refer to Chapter 3.5 for details;

3. Wi-Fi/BT Files and Compilation Update Introduction

3.1 Compilation Files

Files directory corresponding to different Wi-Fi drivers:

```
kernel/drivers/net/wireless/rockchip_wlan/
kernel/drivers/net/wireless/rockchip_wlan/rkwifibt/ #AMPAK modules commonly used
driver
kernel/drivers/net/wireless/rockchip_wlan/cywdhd/ #Cypress/Azurewave modules
commonly used driver
kernel/drivers/net/wireless/rockchip_wlan/rtlxxx #Realtek module is not commonly
used, each model has a separate driver
```

Wi-Fi and Bluetooth files are located in:

`external/rkwifibt/`

AMPAK and Cypress Wi-Fi/BT Firmware files are located in:

`external/rkwifibt/firmware/broadcom/`

The firmware name of the Wi-Fi/BT corresponding to each model of AMPAK and Cypress Wi-Fi:

```
├── AP6212A1
│   ├── bt                          #Bluetooth 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 driver and Firmware: (**Note: Realtek Wi-Fi does not require firmware files, only Bluetooth requires**)

```
external/rkwifibt/realtek/bluetooth_uart_driver/  #Bluetooth uart driver
external/rkwifibt/realtek/bluetooth_usb_driver/   #Bluetooth usb driver
external/rkwifibt/realtek/rtk_hciattach/        #Bluetooth initialization code

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

3.2 Compilation Rules

The corresponding compilation rule files:

```
buildroot/package/rockchip/rkwifibt/Config.in    # The same rules as regular
Kconfig
buildroot/package/rockchip/rkwifibt/rkwifibt.mk # Similar to Makefile
```

As the progress and time of SDK version of each chip platform are inconsistent, the content of the in/mk file obtained by customers may be different, but the general rules are the same.

```
Config.in: pass the Wi-Fi/BT model configured by menuconfig rkwifibt to the mk
file;

rkwifibt.mk: Copy the files, firmware, etc. required for corresponding Wi-Fi/BT
into the file system;
# Specify the source directory of rkwifibt
rkwifibt_SITE = $(TOPDIR)/../external/rkwifibt
#Is the function of building process, pass the building and link options to the
source code, and call the source code Makefile to execute the building
RKWIFIBT_BUILD_CMDS
# After building, install automatically, Buildroot will install the built
libraries and bin files to the specified directory
RKWIFIBT_INSTALL_TARGET_CMDS
```

Please read these two rkwifibt.mk and Config.in files carefully. The main work of these two files is:

- Build module bluetooth driver KO files, such as Realtek's uart/usb bluetooth driver, and some manufacturers' private executable binaries such as AMPAK's wl, Realtek's rtwpriv and other tools;
- According to the Wi-Fi/BT configuration model, copy and install the corresponding firmware/driver KO/executable file to the specified directory, please refer to the corresponding directory in Chapter 3.1;

So developers must be familiar with compilation rules, which is very important for debugging!!!

Note: learn to read the output log printed by make rkwifibt-rebuild when compiling, which includes the compilation/copying process of the above mk file, which is helpful to analyze and solve problems such as compilation errors/copy errors.

3.3 Required Files and Their Paths during Wi-Fi/BT Running

Developers need to understand the files and locations used when Wi-Fi Bluetooth works. When encountering the problem of abnormal Wi-Fi/BT startup, they need to confirm whether the firmware/config file of Wi-Fi Bluetooth exists and whether it matches the Bluetooth model. **Please refer to Chapter 3.1 to check the corresponding relationship, if they are not matched, refer to Chapter 2.5 to check the configuration problems.**

- **For AMPAK/Azurewave modules, take AP6255 as an example:**

The path of the Wi-Fi/BT firmware in the SDK:

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

After the compilation rules in Chapter 2.2, the corresponding files are copied to the output directory of the project: **(kernel4.19 is changed from system to vendor directory)**

```
buildroot/output/rockchip_rk3xxxx/target/
/system(vendor)/lib/modules/bcmdhd.ko                    #Driver ko (if it is
compiled in ko mode)
/system(vendor)/etc/firmware/fw_bcm43455c0_ag.bin        #Driver firmware file
storage path
/system(vendor)/etc/firmware/nvram_ap6255.txt            #Driver nvram file
storage path
/system(vendor)/etc/firmware/BCM4345C0.hcd                #Bluetooth firmware
file (if there is Bluetooth function)
```

After flashing into the device, the files and storage locations required for Wi-Fi running are as follows:

```
/system(vendor)/lib/modules/bcmdhd.ko                    #Driver ko (if it is
compiled in ko mode)
/system(vendor)/etc/firmware/fw_bcm43455c0_ag.bin        #Driver firmware file
storage path
/system(vendor)/etc/firmware/nvram_ap6255.txt            #Driver nvram file
storage path
/system(vendor)/etc/firmware/BCM4345C0.hcd                #Bluetooth firmware
file (if there is Bluetooth function)
```

- **For Realtek modules, take RTL8723DS/RTL8821CU as an example:**

The path of the Wi-Fi/BT firmware in the SDK:

```
external/rkwifibt/realtek$ tree
├── RTL8723DS
│   ├── mp_rtl8723d_config  #the config used by Bluetooth test, which needs to be
obtained from the vendor
│   ├── mp_rtl8723d_fw      #The fw for Bluetooth testing, which needs to be
obtained from the vendor
│   ├── rtl8723d_config     #Bluetooth config
│   └── rtl8723d_fw         #Bluetooth fw
├── RTL8821CU
│   ├── rtl8821cu_config    #Bluetooth config
│   └── rtl8821cu_fw        #Bluetooth fw
```

```
├── bluetooth_uart_driver #RTL8723DS Bluetooth UART driver, which is compiled
into 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 Bluetooth USB driver, which is compiled into
rtk_btusb.ko
│ ├── Makefile
│ ├── rtk_bt.c
│ ├── rtk_bt.h
│ ├── rtk_coex.c
│ ├── rtk_coex.h
│ ├── rtk_misc.c
│ └── rtk_misc.h
├── rtk_hciattach      #Bluetooth initialization program rtk_hciattach, only BT
with UART interface will be used, BT with USB interface does not need
│ ├── fix_mac.patch
│ ├── hciattach.c
│ ├── hciattach.h
│ ├── hciattach_h4.c
│ ├── hciattach_h4.h
│ ├── hciattach_rtk.c
│ ├── Makefile
│ ├── rtb_fwc.c
│ └── rtb_fwc.h
```

After the compilation rules in Chapter 2.2, the corresponding files are copied to the output directory of the project:

```
# prefix directory
buildroot/output/rockchip_rk3xxxx/target/

# if there is a bluetooth function
/system(vendor)/lib/modules/8723ds.ko    #Drive ko (if it is compiled in ko mode)
/system(vendor)/lib/modules/8821cu.ko    #Drive ko (if it is compiled in ko mode)
/usr/lib/rtk_hciattach                   #Bluetooth initialization program
/usr/bin/modules/hci_uart.ko             #Bluetooth ko
#Remember that the firmware file of Bluetooth wtih UART interface will be copied
to the /lib/firmware/rtlbt/ directory
/lib/firmware/rtlbt/rtl8723d_config        #The fw/config of normal Bluetooth
function
/lib/firmware/rtlbt/rtl8723d_fw
#Remember that the firmware file of Bluetooth with USB interface will be copied
to the /lib/firmware/ directory
/lib/firmware/rtl8821cu_config            #The fw/config of normal Bluetooth
function
/lib/firmware/rtl8821cu_fw
```

After flashing into the device, the files and storage paths required for Wi-Fi running is as follows: (kernel 4.19 is changed from system to vendor directory)

```
/system(vendor)/lib/modules/8723ds.ko    #Drive ko (if it is compiled in ko mode)
/system(vendor)/lib/modules/8821cu.ko    #Drive ko (if it is compiled in ko mode)
/usr/bin/rtk_hciattach                   #Bluetooth initialization program (only
used by uart interface)
/usr/lib/modules/hci_uart.ko             #UART bluetooth ko
/usr/lib/modules/rtk_btusb.ko            #USB Bluetooth ko
#Remember that the firmware file of Bluetooth with UART interface will be copied
to the /lib/firmware/rtlbt/ directory
/lib/firmware/rtlbt/rtl8723d_config      #The fw/config of normal Bluetooth
function
/lib/firmware/rtlbt/rtl8723d_fw
#Remember that the firmware file of Bluetooth with USB interface will be copied
to the /lib/firmware/ directory
/lib/firmware/rtl8821cu_config            #The fw/config for normal Bluetooth
function
/lib/firmware/rtl8821cu_fw
```

3.4 The Rules for Auto Loading Wi-Fi Driver KO when Startup

The rules for Wi-Fi loading drivers are as follows:

```
# The original file location is as follows, it will be copied to the /etc/init.d/
directory after being compiled by ./build.sh or make rkwifibt-rebuild (remember
to recompile and package after each modification)
cat external/rkwifibt/SXXload_wifi_xxx_modules
...
insmod WIFI_KO # The content is a generic name, which will be replaced according
to the configuration of rkwifibt during compilation
...

# Through the configuration of menuconfig and the compilation rules of
buildroot/package/rockchip/rkwifibt/Config.in & rkwifibt.mk.
# Part of the content of rkwifibt.mk is showed as follows, WIFI_KO is replaced
with the actual configured ko name
$(SED) 's/WIFI_KO/\/$(FIRMWARE_DIR)\/lib\/modules\ ... ...

# To check the contents of file after compilation
cat buildroot/output/rockchip_rk3xxxx/target/etc/init.d/SXXload_wifi_xxx_modules
 …
#You can see that the Wi-Fi KO is replaced with the actual configured ko, such as
RTL8723DS:
insmod 8723ds.ko
#It is passed in through the buildroot/package/rockchip/rkwifibt/Config.in file
when it is configured by make menuconfig
 …

# Finally, the files in the etc/init.d/ directory will be called in turn when the
system starts, so the Wi-Fi driver is automatically loaded during this period.

# Frequently encountered problems: For example, when using RTL8723DS, but
sometimes developers forget to configure the Wi-Fi model in the buildroot,
resulting in the wrong ko model of insmod in the SXXload_wifi_xxx_modules file:
insmod bcmdhd.ko (normally, it should be insmod 8723ds.ko), the solution: please
refer to Chapter 2.5 for modification of buildroot!
```

3.5 Compilation Update

Firstly, refer to Chapter 2.1 to confirm the compilation environment

- For the modification of kernel Wi-Fi configuration,

  after selecting `make menuconfig`, be sure to save the corresponding defconfig file. For example, If the following is used: `kernel/arch/arm/configs/rockchip_xxxx_defconfig`, and the corresponding modification should be updated to this file, otherwise the update will not take effect.

```
# Kernel directory
make menuconfig
# Modify the corresponding configuration
make savedefconfig
cp defconfig arch/arm/configs/rockchip_xxxx_defconfig
```

- For the modification of Buildroot configuration, after selecting `make menuconfig`, execute `make savedeconfig` in the root directory to save.

```
# Top directory
source xxxx                    #First select the configuration of the
corresponding project, please refers to the SDK development configuration
document
make menuconfig
# Select the corresponding Wi-Fi model
make savedeconfig               #Save configuration
make rkwifibt-dirclean          #Clear the previous
make rkwifibt-rebuild           #Rebuild
./build.sh                      #Repackage to generate firmware
```

Note: Be sure to `make savedefconfig`, otherwise it will be overwritten by the original when building, resulting in the modification not taking effect.

4. Wi-Fi/BT Function Verification

4.1 Wi-Fi STA Test

4.1.1 Turn Wi-Fi On and Off

If there is no wlan0 node, please first check whether the dts/driver is configured correctly and whether the driver is loaded (ko or buildin). If it is correct, please refer to Chapter 6 for troubleshooting.

Turn on Wi-Fi

```
echo 1> /sys/class/rfkill/rfkill1/state # it is rfkill0 when the Bluetooth node
is not turned on
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)
```

Then check whether the Wi-Fi service process is started: check whether there is a wpa_supplicant process, if it is not started, you can start it manually:

Note the -c option below is used to specify the path of the configuration file! Make sure it should be correct!

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

Configuration file analysis:

```
#ctrl_interface interface configuration
#If there is any modification, the -p parameter of the wpa_cli command should be
modified accordingly, wpa_cli -i wlan0 -p <ctrl_interface> xxx
$ vi /data/cfg/wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant  #It is not recommended to modify by
default!
ap_scan=1
update_config=1  #This configuration saves the hotspot configured by the wpa_cli
command to the conf file (wpa_cli save_config)

#AP configuration items
network={
        ssid="WiFi-AP" # Wi-Fi name
        psk="12345678" # Wi-Fi password
        key_mgmt=WPA-PSK # Encryption configuration, if it is not encrypted,
change to: key_mgmt=NONE
}
```

Note: The `wpa_supplicant.conf` file should be modified according to the storage path of the platform used actually.

Turn off Wi-Fi:

`ifconfig wlan0 down`

`killall wpa_supplicant`

4.1.2 Scan APs Nearby

The wpa_cli command communicates with the wpa_supplicant process, so make sure the wpa_supplicant process is running.

`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
```

Note: check whether the number of scanned hotspots matches the number of routers around you similarly, you can compare it with Wi-Fi number scanned by your mobile phone (if your module does not support 5G, only compare 2.4G Wi-Fi numbers); also check the signal strength of the router closest to you, if the router is very close to you, but the signal strength is very weak (regular: -20 to -65; weak: -65 to -70 ; Poor -70 to -90), then check whether your Wi-Fi module is connected to antenna, whether the module's RF index is qualified, etc. (please refer to Chapter 5 Wi-Fi/BT Hardware Test) .

4.1.3 Connect to Router

The first way: **By modifying the configuration file**:

```
#Add network configuration items in wpa_supplicant.conf
network={
        ssid="WiFi-AP"                  #The name of the Wi-Fi
        psk="12345678"                  #The password of the Wi-Fi
        key_mgmt=WPA-PSK        #Encryption configuration; change to
key_mgmt=NONE if not encrypted
}
# Use the wpa_supplicant process to read the above configuration again by the
following command:
wpa_cli -i wlan0 -p /var/run/wpa_supplicant reconfigure
#Send a connection:
wpa_cli -i wlan0 -p /var/run/wpa_supplicant reconnect
```

The second way: **using simple script**:

The latest SDK integrates the wifi_start.sh script, you can directly add ssid and password behind the script if it exists:

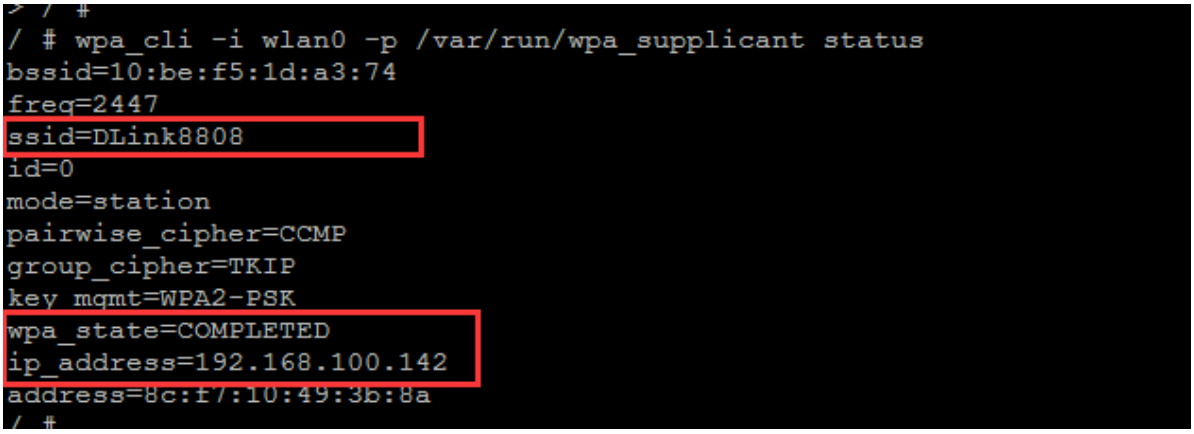
```
wifi_start.sh fanxing 12345678
```

The third way: **using wpa_cli tool**:

```
#encryption:
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 # Save the above
configurations to the conf file

#No encryption:
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
```

Successful connection:



If there is `wpa_state=COMPLETED` but no valid ip_address, check whether the process of obtaining IP address by **dhcpcd** is started; if wpa_state is not COMPLETED, please **check following 2.1.1 scanning chapter first**.

4.2 Wi-Fi AP Hotspot Verification

SDK integrates related programs, execute: softapDemo apName (to open the hotspot with the name of apName without encryption by default) to enable hotspot mode.
Code and building file path:

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

If the softapDemo source code is not found, download it to the external directory from the following address:

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

RTL module: take p2p0 as softap function to generate p2p0 through kernel driver configuration. If there is no p2p0 node, please check the configuration below:

```
+++ 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/Azurewave module: wlan1 is used as softap function, and generate wlan1 nodes by iw command:

`iw phy0 interface add wlan1 type managed`

Debug and customized modification:

```
//You can add encryption, modify IP address and dns and other related information
here by yourself
int wlan_accesspoint_start(const char* ssid, const char* password)
{
        //Configuration of creating a softap hotspot
        create_hostapd_file(ssid, password);

        //softap_name: wlan1/p2p0
        sprintf(cmdline, "ifconfig %s up", softap_name);
        //Set a customize IP address
        sprintf(cmdline, "ifconfig %s 192.168.88.1 netmask 255.255.255.0",
softap_name);
        //Create a configuration file of the dhcp address pool
        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));
        //Use dnsmasq as a dhcp server and assign an ip address to the device
        sprintf(cmdline, "dnsmasq -C %s --interface=%s", DNSMASQ_CONF_DIR,
softap_name);
        console_run(cmdline);

        memset(cmdline, 0, sizeof(cmdline));
        //Start softap hotspot mode
        sprintf(cmdline, "hostapd %s &", HOSTAPD_CONF_DIR);
        console_run(cmdline);
        return 1;
}

//Create a dhcp configuration file, which must be consistent with your customized
IP address, otherwise your phone will not obtain 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;
}

//Create AP hotspot configuration file, please consult Wi-Fi vendors for details,
the parameters here have a close relationship with chip specifications
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); // Channel settings
                fputs("hw_mode=g\n", fp); // 2.4/5G settings
                fputs("ieee80211n=1\n", fp);
                fputs("ignore_broadcast_ssid=0\n", fp);
#if 0 //If you choose encryption, modify here
                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)
        //Set the corresponding hotspot interface according to the Wi-Fi model,
the 4.4 kernel can obtain the Wi-Fi model automatically through the
"/sys/class/rkwifi/chip" node, but there is no this node in the kernel version
4.19 and later, and the Wi-Fi model can be checked by the following way;
        //AMPAK/Azurewave: sys/ bus/sdio/drivers/bcmsdh_sdmmc to check whether
this directory exists
        //Realtek: sys/bus/sdio/drivers/rtl8723ds to check whether this directory
exists
        if (!strncmp(wifi_type, "RTL", 3))
                strcpy(softap_name, "p2p0");//For Realtek modules, use p2p0
        else
                strcpy(softap_name, "wlan1");//For AMPAK/Azurewave modules, use
wlan1

    ... ...
        if (!strncmp(wifi_type, "RTL", 3)) {
                //Realtek module will generate p2p0 node automatically after
opening the coexistence mode
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 module needs iw command to generate wlan1 node as softap
                console_run("iw phy0 interface add wlan1 type managed");
                wlan_accesspoint_start(apName, NULL);
    }
```

After executing the command, you will see the corresponding AP under the setting Wi-Fi interface of the phone. If not, please troubleshoot:

- First: ensure that the configuration files mentioned above are configured correctly;
- Second: confirm whether there is wlan1 or p2p0 node in ifconfig;
- Third: whether the hostapd/dnsmasq process has started successfully;

4.3 BT Verification Test

First of all, make sure that dts/ Buildroot configuration are correct by referring to Chapter 2/3/7.2. Here are two types of commonly used BT modules. If the configuration is correct, the system will generate a **bt_pcba_test** (or the latest SDK will generate a **bt_init.sh**) script program.

Realtek modules:

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

killall rtk_hciattach

echo 0> /sys/class/rfkill/rfkill0/state #Power off
sleep 1
echo 1> /sys/class/rfkill/rfkill0/state #Power on
sleep 1
insmod /usr/lib/modules/hci_uart.ko            # Realtek modules need to load a
specific driver
rtk_hciattach -n -s 115200 /dev/ttyS4 rtk_h5 & # The blue refers to which uart
port is used by Bluetooth

#Note: every time you start a test, you have to kill the rtk_hciattach process
firstly

#Note: There is no sh script for Bluetooth with USB interface, and execute
manually as follows:
echo 0 > /sys/class/rfkill/rfkill0/state #Power off
sleep 1
echo 1 > /sys/class/rfkill/rfkill0/state #Power on
sleep 1
insmod /usr/lib/modules/rtk_btusb.ko     #realtek module needs to load usb driver
```

Azurewave/AMPAK Modules:

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

killall brcm_patchram_plus1

echo 0> /sys/class/rfkill/rfkill0/state # Power off
sleep 2
echo 1> /sys/class/rfkill/rfkill0/state # Power on
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 &

#Note: every time you start a test, you have to kill the brcm_patchram_plus1
process firstly
```

bcm43438a1.hcd represents the firmware file corresponding to the BT model, and /dev/ttyS4 is the UART port Bluetooth used.

Note: `rtk_hciattach, hci_uart.ko, bcm43438a1.hcd` and other files are generated only when the correct Wi-Fi/BT modules are selected in Buildroot configuration in Chapter 2. If these files are not available, please check the above configurations (please refer to the building configuration file in Chapter 3).

After executing the script, execute:

Note: If there is no hciconfig command, please select BR2_PACKAGE_BLUEZ5_UTILS in Buildroot configuration to build and update test

`hciconfig hci0 up`
`hciconfig -a`

Normally, you will see:

```
/ # 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 scanning: `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
```

When abnormal, please refer to Chapter 7.2 for troubleshooting;

4.4 Wi-Fi Suspend and Resume

At present, Wi-Fi supports the network resume function. For example, when the device connects to an AP and obtains an IP address, when the device suspends, we can resume the system through a wireless network packet (ping). Generally, any network packet sent to the device can resume the system.

Modify the wpa_supplicant.conf file and add the following configuration:

```
wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
update_config=1
ap_scan=1
+wowlan_triggers=any # Add this configuration
```

For Realtek Wi-Fi, please check whether the following configuration is in the Makefile of the corresponding driver:

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

DTS configuration: check the schematic diagram to ensure that WIFI_WAKE_HOST (or WL_HOST_WAKE) PIN is connected to the controller, and then check whether the following configuration of dts is correct:

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

System and Wi-Fi suspend testing:

```
dhd_priv setsuspendmode 1 # Only used for AMPAK/Azurewave modules, Realtek does
not need this command
echo mem> /sys/power/state
```

At this time, devices in the same local area network can ping this device. Normally, you can find that the system is resumed. Note to return to normal Wi-Fi working state after the system is resumed:

```
dhd_priv setsuspendmode 0 # Only for AMPAK and AzureWave modules, Realtek's do
not need this command
```

Troubleshooting: If the system does not wake up as expected, please check whether the wake pin is configured correctly and the level status is correct, whether 32.768k is turned off, etc.

4.5 Wi-Fi Monitor Mode

AMPAK or AzureWave Wi-Fi modules:

```
#Set up monitoring channel:
dhd_priv channel 6 //channal numbers

#Open monitor mode:
dhd_priv monitor 1

#Close monitor mode:
dhd_priv monitor 0
```

Realtek Wi-Fi modules:

```
#Driver Makefile should be opened:
+ CONFIG_WIFI_MONITOR = y

#Open wlan0 and close p2p0
ifconfig wlan0 up
ifconfig p2p0 down

#Open monitor mode
iwconfig wlan0 mode monitor
or
iw dev wlan0 set type monitor

#Switch channels
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 Verification

```
#New configuration file: 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

#Start: (kill the previous firstly)
wpa_supplicant -B -i wlan0 -c /tmp/p2p_supplicant.conf
wpa_cli
> p2p_find
>

#At this time, open p2p on the mobile phone, you wll search for the above
device_name=p2p_name, click to connect

#At this time it will display on the device:  //The following is my phone
> <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

#The device responds and send a connection command, and pay attention to the MAC
address to be consistent with the above
```

```
>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:14e:14e:14
> #At this time the connection is successful
> 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)
#You can see that the device is connected to the phone, and it is normal if you
can ping each other.
```

4.7 Connection Function

Case: Wi-Fi enables wlan0 to connect to an AP that can access Internet, together with opening wlan1 or p2p0 as a hotspot in Chapter 4.2, so that the mobile phone can connect to the hotspot for Internet access. The configuration is as follows, and open the following configuration in the kernel :

```
+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
```

Execute the following two commands to start the connection function, the following IP address is the address configured when softap is started:

```
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 Hardware RF Target

5.1 Test Items

Wi-Fi/BT test items:

For example: transmit power, EVM, crystal frequency offset, receiving sensitivity, etc.

Example: (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

Antenna test items:

Passive S11, OTA test of the whole active Wi-Fi antenna

Example:

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

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

5.2 Test Tools and Methods

The PDF/TXT documents mentioned below can be found in the docs/linux/wifibt directory, and if customers have test problems or without professional test equipment, please directly contact module vendors for assistance.

5.2.1 Realtek Test

Generally, there are two types of COB and module. Modules are generally strictly tested by the module factory and flashed calibrated data to internal efuse by default. Customers only need to test and verify whether the indicators are qualified; while COB need to design Wi-Fi peripherals circuits and additional components, so you need to complete RF calibration test with Realtek, and integrate the calibrated data into efuse of the chip or load it by driver. Please contact module vendor directly for details.

Wi-Fi test:

```
Please refer to Quick_Start_Guide_V6.txt, pay special attention to replace the
command iwpriv with rtwpriv.
```

BT test:

Please refer to MP tool user guide for linux20180319.pdf (please contact module vendors or manufacturers for detailed test items), pay special attention to the test that the module factory needs to provide a Bluetooth firmware file specially used for testing, and put it in the specified directory to test the Bluetooth indicators..

```
# Note: please turn on the BT power before testing
echo 0> sys/class/rfkill/rfkill0/state
sleep 1
echo 1> sys/class/rfkill/rfkill0/state

# Special attention: the process rtk_hciattach cannot be run, please kill killall
rtk_hciattach

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

>
>
>enable uart:/dev/ttyS4 # Note that ttySX corresponds to connected hardware uart
port actually
>
>>> enable[Success:0]
```

If it is Realtek's COB solution, and you need to integrate the test calibration data map file into driver, please refer to the following way:

```
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 //Block out here
    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

//Change the calibration "map file" provided by the module vendor into an array
form and assign it to map.
    _rtw_memcpy(buf, map, map_size); //It is the operation that final assignment
of map to buf

    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 Test

Wi-Fi Test

Firstly, replace it with test firmware: the test firmware of each AP module is different, such as:

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

The fw_bcmxxx_mfg.bin and APxxxx should be matched according to your module model, otherwise it cannot be tested! So confirm whether there is a test firmware of the corresponding model firstly, if you don't find it, please ask module vendor to provide it.

The latest SDK has built-in test firmware for supporting models, so use the built-in test script to let Wi-Fi enter RF test mode directly:

```
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
```

The previous SDK does not have a built-in test firmware. The following takes AP6236 as an example:

```
# Push fw_bcm43436b0_mfg.bin to data or other writable partitions, and then
execute the following command: (note the path below)
mount --bind /data/fw_bcm43436b0_mfg.bin /system/etc/firmware/fw_bcm43436b0.bin
ifconfig wlan0 down
ifconfig wlan0 up
wl ver
```

Normally, executing `wl ver` will print a string of characters with the word WL_TEST in it, indicating that it has entered the test mode. Please refer to the following document for detailed test:

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

Bluetooth Test

After executing the bt_init.sh script (it is in SDK by default, please refer to Chapter 3.3), execute :**(Note: If there is no hciconfig command, please select BR2_PACKAGE_BLUEZ5_UTILS in the Buildroot configuration to build and update the test)**:

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

It does not finish the initialization until hci0 node appears. But when it does not appear, there are two possibilities:

1. The Bluetooth dts configuration is abnormal or the hardware is abnormal or the uart port is configured incorrectly, causing the failed initialization;
2. The Bluetooth firmware file is configured incorrectly or there is no such file;

Please refer to the BT related troubleshooting in Chapter 1or 2;

For detailed test instructions, please refer to:

```
BT RF Test Commands for Linux-v05.pdf #Test 1/2 steps in the document have been
executed in the script, no need to execute)
```

5.3 Report

After confirming the above hardware tests, please output a test report, which should be provided to us when you encounter performance or stability problems.

6. Wi-Fi Performance Test

Pleas test performance by iperf

Pay attention to the following two items that affect performance:

- After finishing Wi-Fi RF test and OTA test of antenna, and make sure that there is no problem with the indicators before testing performance, otherwise it is meaningless;
- If you find the data fluctuates greatly, please go to an open space or basement or other places with little interference to confirm again (it is best to test in a shielded room);

Test environment: **due to the large interference factors in an open environment, it is recommended to test in a shielded room**. First, ensure that Wi-Fi can connect to an AP normally and obtain an IP address;
Test points: the size of throughput rate, and stability, whether there is up and down fluctuations, etc.;
Router channel: choose low, medium, and high channels to test separately, such as 1/6/11 channel:

```
# TCP
Down:
        Board: iperf -s -i 1
        Computer:        iperf -c xxxxxxx(IP address of the board) -i 1 -w 2M -t
120

Up:
        Compute: iperf -s -i 1
        Board: iperf -c xxxxxxx(IP address of the computer) -i 1 -w 2M -t 120

# UDP
Down:
        Board: iperf -s -u -i 1
        Compute: iperf -c xxxxxxx(IP address of the board) -u -i 1 -b 100M -t 120

Up:
        Compute: iperf -s -u -i 1
        Board: iperf -c xxxxxxx(IP address of the computer) -u -i 1 -b 100M -t
120

# Note: The iperf command of the board should be configured in the Buildroot:
BR2_PACKAGE_IPERF = y
```

7. Wi-Fi/BT Troubleshooting

7.1 Brief Description of Wi-Fi Identification Process

SDIO interface: kernel MMC framework will initialize the SDIO WiFi device when booting,. First, it will parse the GPIO configured by the **reset-gpios** attribute of the **sdio_pwrseq** node in the dts, that is, **WL_REG_ON** and then pull it High, and send an initialization command to the module through SDIO_CLK/CMD/DATA. Firstly, the controller will access the module at a low frequency of 400/300/200K and ask its basic information: SDIO2.0 (maximum CLK is 50M) or 3.0 (maximum CLK is 208M), support 4-line or 1-line and other information, and then **increase the CLK frequency to high frequency** according to the supported specifications. At this time, the initialization is basically completed, and you will see the following log:

```
# Note that mmc0: The number of 0 is not fixed, it may be 0/1/2; ff4a0000:
indicates the address of the controller, and different platforms are also
different;
dwmmc_rockchip ff4a0000.dwmmc: allocated mmc-pwrseq
# low frequency
mmc_host mmc0: Bus speed (slot 0) = 400000Hz (... actual 400000HZ div = 0)
# increase frequency
mmc_host mmc0: Bus speed (slot 0) = 50000000Hz (... actual 50000000HZ div = 0)
mmc0: new high speed SDIO card at address 0001 #SDIO 2.0
# or
mmcx: new ultra high speed SDR104 SDIO card at address 0001 #SDIO 3.0
```

USB/PCIE interface: The identification process of these two interfaces is complicated. Please refer to the USB/PCIE related documents in the doc/ directory. After identification:

USB interface: If it is recognized normally, execute **lsusb**, you will see the following information:

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

PCIE interface: If it is recognized normally, execute **lspci**, you will see the following information:

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

Note: The above identification processes are all identified during the boot process. Only after the correct device is identified, Wi-Fi driver will be loaded correctly!

7.2 Wi-Fi Issues

Wi-Fi with SDIO interface: There are generally two cases

- First, make sure to find the following LOG in the kernel log. If not, it means that the SDIO card is not recognized. For such issues, please **refer to Chapter 7.1.1**;

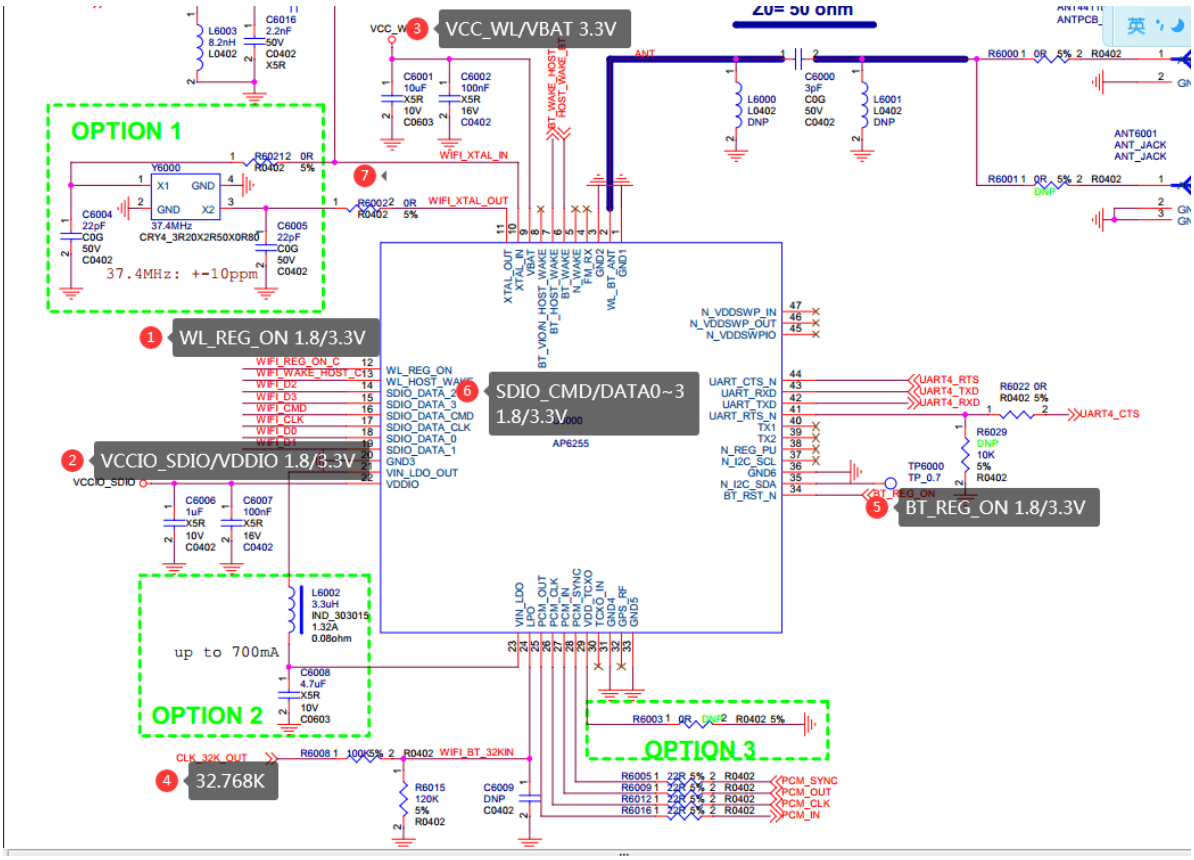
```
mmc0: new high speed SDIO card at address 0001
# or
mmcx: new ultra high speed SDR104 SDIO card at address 0001
```

- If you see the LOG that recognizes SDIO normally, but without wlan0 or wlan0 up fails; **Refer to Chapter 7.1.2/7.1.3/7.1.4;**

7.2.1 Wi-Fi Abnormal: SDIO Can Not Be Recognized

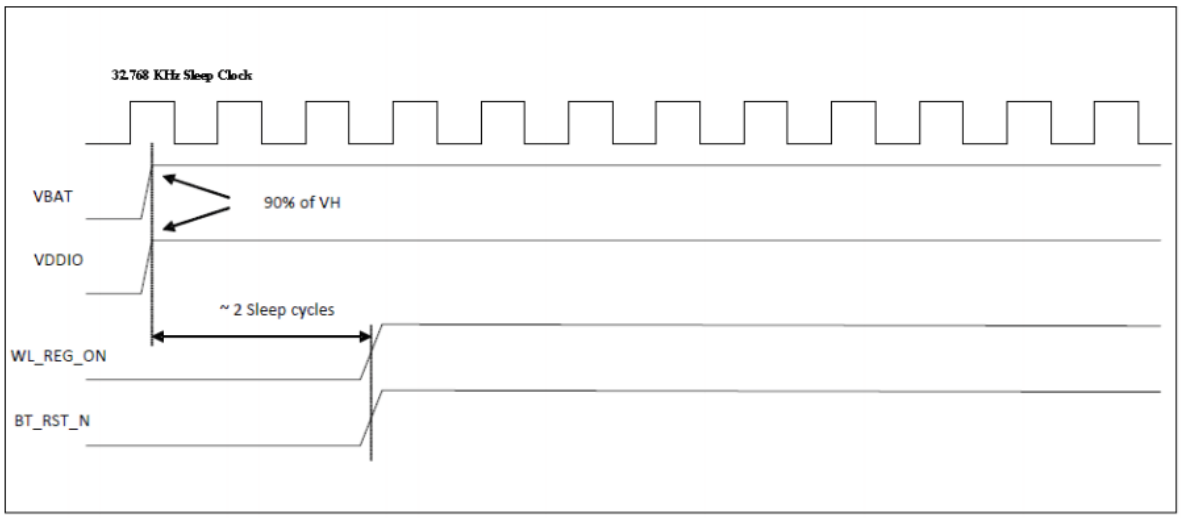
Follow the steps below to check the kernel LOG in turn:

- **First of all, it is strongly recommended to double check the DTS/KERNEL configuration in Chapter 2 in detail! ! !**
- Check the figure below, measure the level status of the corresponding pins and whether the CLK frequency is correct according to the label (**Note: SDIO3.0 mode must be 1.8V**)



- **Measure the timing of VBAT/VDDIO/WL_REG_ON with an oscilloscope to ensure that the following timing requirements are met:**

Many customers fail to recognize SDIO because of abnormal VDDIO power-on sequence.



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

The WL_REG_ON configuration of DTS is incorrect, resulting in the Wi-Fi enable pin can not being pulled high;

Method 1: **You can use an oscilloscope to test its waveform to see if it is pulled high or low, and whether the voltage amplitude (1.8/3.3V) meets the requirements; Because SDIO will retry several times during initialization, when the identification is successful: WL_REG_ON will be pulled high, and if the identification fails, it will be pulled low; if it is always low, it means DTS configuration error**;

Method 2: **Pull up hardware WL_REG_ON directly to verify**, if it can be recognized, it means that the DTS WL_REG_ON configuration is incorrect;

The typical abnormal LOG is as follows: **(mmcX: The number of X is not fixed)**

```
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 configuration error**

Both of sdio_pwrseq and wireless-wlan are configured with WL_REG_ON, which leads to repetition, please double check the DTS configuration introduction!

```
[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**

The VDDIO power supply of pin 12 must be 3.3/1.8V, and the corresponding SDIO_CMD/SDIO_DATA0~3 must also be 3.3v or 1.8V.

- **IO Power Domain**

VDDIO: **The power supply voltage does not match the DTS power domain configuration (please refer to Chapter 2.2.3 IO power domain configuration)**;

- **SDIO_CLK without waveform**

SDIO_CLK: The wrong IO power domain setting may cause the CLK waveform can not to be measured (please refer to Chapter 2.2.3 IO Power Domain Configuration);

SDIO_CLK: Whether there is a pull-up resistor, if so, remove and then test;

- **Insufficient power supply or too much ripple**

  VCC_WL/VBAT/VDDIO_SDIO: **Insufficient power supply or too much ripple**

```
# Realtek modules
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: **There is no 32.768K waveform, or the waveform amplitude or accuracy is not meet the requirement of the above table**

```
  #Without 32k, abnormal log of AMPAK/Azurewave module::
[   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 trace quality/inappropriate capacitance or inductance/IO_DOMAIN voltage setting error**

  Case 1：SDIO_CLK/CMD/DATAX：**PCB layout abnormality or capacitance and inductance do not meet requirements, or poor connection and poor soldering lead to fail to initialize or run high frequency, you can reduce the frequency appropriately to confirm whether it meets the timing requirements of CLK/CMD/DATA in the datasheet of the WiFi module:**

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

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

Case 2:Wi-Fi does not match the controller's sdio voltage, please check whether the software io_domian configuration is consistent with the hardware (please refer to Chapter 2.2.3)

```
# Abnormal 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

# Abnormal log2, For example, the log of data communication abnormality such as
failling to download firmware and so on:
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

# Abnormal 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 **or interrupt level configuration error** (please refer to Chapter 2.2.1), or virtual soldering, resulting in **the following abnormality or system stuck:**

```
# Abnormal log of AMPAK and Azurewave module:
dhd_bus_rxctl: resumed on timeout, INT status=0x208000C0
dhd_bus_rxctl: rxcnt_timeout=1, rxlen=0
```

The way to modify the frequency:

```
&sdio {
+ max-frequency = <10000000>; # Modify here to limit the frequency
```

- **A certain line of SDIO_D1~3 is poorly soldered**

If there is a log similar to the following, and it is still abnormal to reduce the max frequency to less than 10M:

```
[   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
```

Then modify it to 1-line mode to test:

```
&sdio {
+        bus-width = <1>;       /* change to 1-wire mode and test */
```

If it takes effect, it means that there is a hardware problem with one of SDIOD1~3 line of the hardware, such as virtual soldering or wrong connection;

- **Module/chip is defective product**

There is a small probability that the controller chip or Wi-Fi module is found to be defective, and you can find a few more devices for verification;

- **Abnormal iomux multiplexing**

```
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
```

- **Others**

If the above troubleshooting fails, please upload to RK redmine system: dts/dtsi configuration, complete kernel log dmesg, pdf schematic diagram and other files, and provide: WIFI_REG_ON and Wi-Fi_CLK/Wi-Fi CMD three lines on the power-on waveform diagram;

Abnormal log:

```
[    3.842211] dhd_module_init: in Dongle Host Driver, version 1.579.77.41.10 (r)
[    3.842219] ======== dhd_wlan_init_plat_data ========
[    3.842224] [WLAN_RFKILL]: rockchip_wifi_get_oob_irq: Enter
[    3.842236] dhd_wlan_init_gpio: WL_HOST_WAKE=-1, oob_irq=71,
oob_irq_flags=0x414
[    3.842240] dhd_wlan_init_gpio: WL_REG_ON=-1
[    3.842244] dhd_wifi_platform_load: Enter
[    3.842272] Power-up adapter 'DHD generic adapter'
[    3.842321] wifi_platform_set_power = 1
[    3.842326] ======== PULL WL_REG_ON(-1) HIGH! ========
[    3.842332] [WLAN_RFKILL]: rockchip_wifi_power: 1
[    3.842338] [WLAN_RFKILL]: wifi turn on power. -1
```

```
[    3.852660] mmc_host mmc2: Bus speed (slot 0) = 300000Hz (slot req 300000Hz,
actual 300000HZ div = 0)
[    3.905554] mmc2: queuing unknown CIS tuple 0x80 (2 bytes)
[    3.907592] mmc2: queuing unknown CIS tuple 0x80 (3 bytes)
[    3.909632] mmc2: queuing unknown CIS tuple 0x80 (3 bytes)
[    3.913294] mmc2: queuing unknown CIS tuple 0x80 (7 bytes)
[    4.010193] mmc_host mmc2: Bus speed (slot 0) = 150000000Hz (slot req
150000000Hz, actual 150000000HZ div = 0)
[    4.142591] wifi_platform_bus_enumerate device present 1
[    4.142597] ======== Card detection to detect SDIO card! ========
[    4.142603] mmc2:mmc host rescan start!
[    4.266632] dwmmc_rockchip fe310000.dwmmc: Successfully tuned phase to 216
[    4.271163] mmc2: new ultra high speed SDR104 SDIO card at address 0001
[    4.288408] bcmsdh_register: register client driver
[    4.288816] bcmsdh_sdmmc_probe: Enter num=1
[    4.289045] bcmsdh_sdmmc_probe: Enter num=2
[    4.289050] bus num (host idx)=2, slot num (rca)=1
[    4.289056] found adapter info 'DHD generic adapter'
[    4.289179] sdioh_attach: set sd_f2_blocksize 256
[    4.289285] sdioh_attach: sd clock rate = 0
[    4.289947] dhdsdio_probe : no mutex held. set lock
[    4.290183] F1 signature read @0x18000000=0x1042aae8
[    4.300297] F1 signature OK, socitype:0x1 chip:0xaae8 rev:0x2 pkg:0x4
[    4.300304] dhdsdio_probe_attach: unsupported chip: 0xaae8
[    4.300311] dhdsdio_probe: dhdsdio_probe_attach failed
[    4.300318] dhdsdio_probe : the lock is released.
[    4.300323] bcmsdh_probe: device attach failed
[    4.300327] sdioh_probe: bcmsdh_probe failed
[    4.300557] bcmsdh_sdmmc: probe of mmc2:0001:2 failed with error -12
```

7.2.2 USB Wi-Fi Troubleshooting

USB Wi-Fi will print the information similar to the following, if not, the Wi-Fi driver will not be loaded:

```
usb 2-1: new high-speed USB device number 2 using ehci-platform
usb 2-1: New USB device found, idVendor=0bda, idProduct=b82c #PID/VID of vendor
usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 2-1: Product: 802.11ac NIC
```

If you do not refer to Chapter 2.2.1, please check whether the module enable pin WL_REG_ON is pulled high and whether the USB-related configuration is correct.

7.2.3 Special Notice of Realtek Wi-Fi

When you finish troubleshoot in Chapter 6.1, but Realtek Wi-Fi still cannot recognize SDIO, then confirm the following two points:

7.2.3.1 wlan0 Has Identified but Scan Abnormality

- **Rf-Link/Fn-Link Wi-Fi module**: PIN25 (corresponding to the 22 PIN of the COB chip) is the internal multiplexing pin of the chip, one is connected to the PCM_IN of RK chip as PCM_OUT, used for Bluetooth PCM calls function; the other is as Wi-Fi IC LDO_SPS_SEL function (SPS or LDO mode selection), when this pin is low, it means SPS mode, when it is high, it is LDO mode, **so this pin is either High or Low level, if there is a half-level** , it will cause the Wi-Fi to fail to work normally, such as failing to scan SDIO or failing to scan AP. **If a half-level hardware is detected, it is required to pull-up**
- **Required module:** PIN7 must be pulled down.

7.2.3.2 Realtek Supports SDIO3.0

Realtek Wi-Fi SDIO 3.0 abnormality

When using high-end modules such as RTL8821CS that support 3.0, initialization failure with some probability, and the abnormal log is as follows:

```
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
```

Please add the following patches:

```
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;

#The other way
        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 Wlan0 of RV1109/1126 Platform Cannot Be Up

```
# ifconfig -a   #There is a wlan0 interface, but ifconfig wlan0 reports this
error
# ifconfig: SIOCSIFFLAGS: Operation not possible due to RF-kill

#Buildroot configuration, modify the following configuration and save:
BR2_PACKAGE_IPC_DAEMON = n
BR2_PACKAGE_NETSERVER = n
BR2_PACKAGE_CONNMAN = n
BR2_PACKAGE_DHCPCD = y
#Delete intermediate files:
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
#Rebuild to ensure busybox ps after booting: there is no processes such as
ipc_daemon/connmand/netserver appear
```

7.2.5 Wi-Fi SDIO Card Is Recognized but Wlan0 Up Failed

The following log appears in the kmesg log, indicating that SDIO is recognized normally, but Wi-Fi is still unavailable.

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

- Wi-Fi driver ko mode: insmod ko loading error, **please refer to Chapter 2.4**;
- Wi-Fi driver buildin mode: kernel configuration error, **please refer to Chapter 2.4**;
- The Firmware file does not exist or the file name does not match the module model (only for AMPAK and Azurewave modules), **please refer to Chapter 3.3**;

```
[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
```

- The power supply of the Wi-Fi module is unstable

```
[ 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
```

- The RTL module scans abnormally, for example, the AP cannot be scanned

Please refer to Chapter 7.2.3.1

```
[ 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
```

- **The buildin and ko modes of the kernel configuration are selected at the same time or the configuration does not match**

1. The Wi-Fi driver is loaded too early, causing an exception. From the log, it can be seen that the driver is loaded more than 0 points. At this time, the sdio/usb device has not been enumerated. Pleas check the kernel configuration chapter;
2. The system loads two kos at the same time, one is bcmdhd.ko and the other is 88xx.ko, which causes an abnormality. Please refer to the compilation instructions in Chapter 3 for details;

7.2.6 Wi-Fi with SDIO Interface Runs Abnormally After A Period of Time

Case 1: SDIO_CLK/CMD/DATAX: **PCB trace is abnormal, capacitor and inductance does not meet the requirements, poor connection, poor soldering and other problems lead to abnormal initialization or can not run high frequency, you can reduce the frequency to confirm (modify the max-frequency under &sdio node) If the frequency can be reduced, you need to find the hardware measurement waveform to confirm whether it meets the timing requirements of CLK/CMD/DATA in the datasheet of the Wi-Fi module.**

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>;   # Modify here to limit the frequency
```

Case 2: Wi-Fi and the controller's sdio voltage do not match, please check whether the software io_domian configuration is consistent with the hardware (please refer to Chapter 2.2.3)

7.2.7 Wi-Fi Unable to Connect to Router for Disconnection or Unstable Connection

This kind of problem is usually caused by the unqualified RF and antenna indicators of the Wi-Fi chip or module. The confirmation method is as follows:

- RF indicators

First, get a report on the Wi-Fi RF indicators and the antenna OTA test of the whole device from a hardware engineer to ensure that the hardware indicators are normal (**transmit power, EVM, crystal frequency offset, receiving sensitivity**);

- Scan comparison

Do a basic scan comparison: place the test device and the mobile phone at the same location away from the router, and preliminarily check whether the hardware indicator is normal by comparing the number of APs scanned and their signal strength with the mobile phone (or a competitor of the same specification). Please refer to Chapter 4.1 for the scanning method. It is going to introduce how to compare it with a mobile phone. Find an Android phone, go to the developer option from settings, enable the option of WLAN detailed logging, and go back to the WLAN setting interface to see the AP's detailed information includes RSSI, and the hardware indicators are preliminarily checked by comparing the number of hotspots and signal strengths scanned by the mobile phone;

- Interference

Troubleshooting interference factors. For example, there are a lot of 2.4/5G wireless devices connected at the same time in the current environment, so that the current environment has a lot of interference, you can put the test device and the comparison mobile phone (or a competitor of the same specification) at the same location, if both are abnormal, it can be regarded as interference; if the mobile phone is normal, it can be regarded as an abnormal hardware indicator of the device;

- Distance

Troubleshooting distance factors, the signal is weak due to too long distance (by scanning wpa_cli scan/scan_r, the signal strength of the connected AP is between -70 ~ -90), which leads to communication failure, you can reduce the distance to confirm;

- Router compatibility

Troubleshooting router compatibility issues, you can replace routers of different manufacturers and models to confirm;

- Consistency

Troubleshooting abnormal problem of the single board, you can use two or three devices for comparative testing;

- Ask the module vendor for assistance

You can directly ask the module vendor or the original Wi-Fi manufactory for assistance. They have professional packet capture instruments to capture air packets, which can quickly fix the problem;

- Bring the issue scene to our Shenzhen office to confirm

7.2.8 Throughput Not As Expected

- If it is SDIO interface, RK platform interface only supports a maximum of 150M, and the protocol supports a maximum of 208M, so the hardware itself will lose some performance;
- The second is to pass the RF indicators test first, and provide test reports and antenna OTA test reports if necessary;
- Then find a shielded room or a clean environment such as an underground parking lot for testing, eliminate environmental interference, and ensure that the clean environment is normal at first;
- Finally, CPU/DDR fixed frequency verification can be done;

```
#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

#Put the dw-mmc interrupt to other cpu cores for verification

Reference:
Such as switching CPU core interrupt to verify.
cat /proc/interrupts  // check the corresponding interrupt number
echo 5 > /proc/irq/38/smp_affinity_list // put the interrupt on cpu2 for running
cat /proc/interrupts //check the count
```

7.2.9 IP Abnormal

If the IP address cannot be obtained or the IP address conflicts, please confirm whether the dhcpcd or udhcpc process is enabled;

dhcpcd: is used by the SDK by default and starts with the system start. It is a dhcp client with relatively complete functions;

udhcpcd: is a simplified dhcp client of busybox;

Note: The two processes should not be enabled at the same time, only one of them can be used!

7.2.10 Resume and Suspend Abnormal

Frequent resume by Wi-Fi after suspending, troubleshoot the following situations:

- The 32.768k of the Wi-Fi module is turned off after suspending;
- WIFI_REG_ON is pulled low;
- WIFI_WAKE_HOST PIN hardware is unstable, and the level is jittering (normally it is low level, and the trigger is high level);
- The following specific commands are not executed before and after the AP/CY module Wi-Fi suspending to filter broadcast or multicast packets:

```
#Execute before suspending:
dhd_priv setsuspendmode 1
#Execute after Resuming:
dhd_priv setsuspendmode 0
```

7.2.11 PING Abnormal

PING fails or delay is very large with some probability:

- RF indicators have not been tested;
- There is a high probability that Wi-Fi is performing a scanning operation, which will cause a large ping delay;
- The router or PC firewall is turned on, so the ping operation is forbidden;

7.2.12 Customized Modification

We often receive some strange issues. After troubleshooting, it is found that some customers have modified some Wi-Fi/BT driver configurations, which leads to abnormality, so please check whether you have modified the original code. The modification code and the reason can be sent to us via redmine for confirmation.

7.2.13 Wlan0 Is Normal, but No AP Can Be Scanned

- Check whether the crystal oscillator corresponding to the module is consistent with the requirements of the chip, for example, Wi-Fi requires 24M, but a 37M is connected;
- The accuracy of 32.768k does not meet the requirements;

7.2.14 Dual Wi-Fi AP+RTL Abnormal

Connect two Wi-Fi, one is AP6xxx with sdio interface and the other is RTL8xxxu with USB interface; after the kernel is started, both initializations are normal, but when doing "down" operation of RTLxxxbu module interface, kernel hangs.

```
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 iComm Wi-Fi Abnormal

On the RV1109/1126 platform, it is found that SDIO can be recognized, but there is an error in reading and writing. Try the following modifications:

```
&sdio {
        //rockchip,default-sample-phase = <90>; #Delete this configuration option
```

7.2.16 Hotspot of iPhone Can't be Connected in iOS15 System

```
#For Realtek modules, modified like this
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

#Broadcom modules:
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
```

7.3 Bluetooth Issues

The Bluetooth test items fail in Chapter 4.3 or the deviceio_test bluetooth function is abnormal, in must be one of the following situations, please follow the steps below to check carefully:

- The kernel configuration of Realtek modules must be turned off : **CONFIG_BT_HCIUART=n, please refer to Chapter 2.4.2**;
- DTS BT (**BT_RST_N**) power enable pin configuration is incorrect, **please refer to Chapter 2.2.2**;

```
#Can be confirmed by the following command:
echo 0 > sys/class/rfkill/rfkill0/state  #Pull down BT_REG_ON and measure the
corresponding PIN with a multimeter
echo 1 > sys/class/rfkill/rfkill0/state  #Pull BT_REG_ON high and measure the
corresponding PIN with a multimeter
```

- UART configuration error, **please refer to Chapter 2.2.2/2.4.2/2.5**;
- The Realtek module needs rtk_hciattach / hci_uart.ko / rtk_btusb.ko, make sure whether is is compiled correctly or the kernel configuration is right, **please refer to Chapter 2.5/3.3**;

```
#Initialize bluetooth, only used by UART module
/usr/bin/rtk_hciattach

#Whether CONFIG_BT_HCIUART configuration of the kernel is removed, and whether
the corresponding ko file has been generated?
/usr/lib/modules/hci_uart.ko

#Whether there is a corresponding usb driver generated for the USB interface
/usr/lib/modules/rtk_btusb.ko
```

- **Bluetooth Firmware/config file does not exist or does not match the Bluetooth module version, please refer to Chapter 3.3;**

```
#AMPAK and Azurewave module, For example: AP6212A: The corresponding file is
bcm43438a1.hcd
/system(vendor)/etc/firmware/BCMxxxx.hcd

#The corresponding file of RTL8723DS with SDIO interface
/lib/firmware/rtlbt/rtl8723d_fw
/lib/firmware/rtlbt/rtl8723d_config

#The corresponding file of RTL8821CU with USB interface:
/lib/firmware/rtl8821cu_fw
/lib/firmware/rtl8821cu_config
```

- BT UART RTS/CTS **hardware connection error**, resulting in abnormal initialization recognition;

```
#AMPAK and Azurewave module, 4 lines must be connected to the controller
host tx - controller rx
host rx - controller tx
host rts - controller cts
host cts - controller rts

#Realtek Modules
#For COB chips that use Realtek Bluetooth directly: The hardware connection of
the UART interface is as follows:
host tx - controller rx
host rx - controller tx
host rts - controller cts
host cts - ground #The cts of the controller should be grounded
#RTL8822C chip is special, all 4 lines must be connected to the controller
host tx - controller rx
host rx - controller tx
host rts - controller cts
host cts - controller rts

#For modules, the module vendors generally grounds the controller rts internally,
so the controller does not need to be grounded, and is directly connected to the
controller rts; Special attention: realtek has a large number of agents and each
modules may be different, so please confirm with the module vendor, if the
controller rts is not grounded,  it must be grounded on the controller side.
host tx - controller rx
host rx - controller tx
host rts - controller cts
host cts - controller rts
```

- deviceio_release compilation error, **Please refer to Chapter 10.2/3.5 for configuration!**
- The deviceio_test bluetooth abnormality is basically caused by the above problems;

```
#Print the following log when turning on bluetooth
#hcd_file = /system/etc/firmware/BCM4343A1.hcd, does this file match the
Bluetooth module?
#ttys_dev = /dev/ttyS1, Whether the UART port matches the hardware
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. New Module Porting or Old Module Driver Update

Pay special attention to the following items:

- **It is strongly recommended to read and understand the files storage rules and building rules in Chapter 3, which is very important for porting.**
- **The SDK obtained by customers may be outdated, so the rules of Config.in and rkwifibt.mk may be updated but the theory are the same;**
- **For updating driver cases, you can skip some steps accordingly.**

8.1 Realtek Modules

8.1.1 Wi-Fi Modules

Take RTL8821CS as an example:

Get the corresponding porting package from module vendors or the manufacturers:

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

For Wi-Fi driver part , go to the following directory:

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

- Add the corresponding compilation configuration (**This operation is not required for case of updating driver**)

Copy the driver file to drivers/net/wireless/rockchip_wlan/, rename it to rtl8821cs, and modify Makefile/Kconfig to add the corresponding configuration:

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

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

- Modify Makefile

```
#Change to RK platform:
CONFIG_PLATFORM_I386_PC = n
CONFIG_PLATFORM_ARM_RK3188 = y

#The following configuration should be removed on RK platform:
ifeq ($(CONFIG_PLATFORM_ARM_RK3188), y)
-EXTRA_CFLAGS += -DRTW_ENABLE_WIFI_CONTROL_FUNC #Remove this configuration,if it
existed
MODULE_NAME := 8821cs

#If there is a requirement for WiFi keep in connection when in sleep(WOWLAN),
open the following configuration
CONFIG_WOWLAN = y
CONFIG_GPIO_WAKEUP = y
```

- If there is a WOWLAN requirement, add the irq obtainning function of the WL_HOST_WAKE pin

```
#Modify 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();  //corresponding to WIFI_WAKE_HOST
PIN of dts

        return ret;
}
```

- If there is a customized MAC address requirements

```
# Modify: core\rtw_ieee80211.c
+include <linux/rfkill-wlan.h> //Add the header file
# Find the rtw_macaddr_cfg function
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]);
+       }
+ }
```

- Add driver loading entrance

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

//Add the following code at the end of the file:
#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);
}

//Pay attention to the configuration of the kernel,the corresponding
configuration of KO or buildin is different
#ifdef CONFIG_WIFI_BUILD_MODULE
//KO mode following this way
module_init(rockchip_wifi_init_module_rtkwifi);
module_exit(rockchip_wifi_exit_module_rtkwifi);
#else
```

```
#ifdef CONFIG_WIFI_LOAD_DRIVER_WHEN_KERNEL_BOOTUP
//buildin mode following this way
late_initcall(rockchip_wifi_init_module_rtkwifi); //late_initcall delay loading,
waiting for the file system to be ready
module_exit(rockchip_wifi_exit_module_rtkwifi);
#else
EXPORT_SYMBOL(rockchip_wifi_init_module_rtkwifi);
EXPORT_SYMBOL(rockchip_wifi_exit_module_rtkwifi);
#endif
#endif

//comment out the followings, pay attention to delete the entry __init __exit of
the following two functions
//module_init(rtw_drv_entry);
//module_exit(rtw_drv_halt);
```

- The 4.4 version kernel needs to add the model identification function (**This operation is not required for the case of updating driver, and the 4.19 kernel does not need to add the following content**)

```
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) {
```

- Deal with the possible compilation errors:

```
rtl8xxx\os_dep\linux\rtw_android.c //Comment out the following two commands
#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)) */
```

For Buildroot system part: (This operation is not necessary in the case of updating driver)

```
buildroot\package\rockchip\rkwifibt\Config.in
Add the followings in turn:
+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
#The name of 8821cs.ko corresponds to the driver's Makefile. When do make
rkwifibt compilation, it will be copied from the kernel driver directory to the
file system according to this name.
```

8.1.2 BT Modules

8.1.2.1 UART Interface

If modules support Bluetooth, you have to ask the vendors to provide a software package similar to the following:

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

Take RTL8821CS as an example (you only need to replace the corresponding file in the case of updating driver):

```
#Place or update the Bluetooth firmware and configuration files to the following
directory:
ls external/rkwifibt/realtek/
#The structure is as follows:
external/rkwifibt/realtek/RTL8821CS #Create a new directory if it didn't exist
├── rtl8821cs_config
└── rtl8821cs_fw

# Note: The name of the RTL8821CS directory should be consistent with the
configuration in Buildroot rkwifibt Config.in:
# BR2_PACKAGE_RKWIFIBT_"RTL8821CS"
```

For Buildroot part: enable the Bluetooth, (no need to do this in the case of updating driver)

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

8.1.2.2 USB Interface

For Bluetooth with USB interface, the vendor will provide a porting package similar to the following, take RTL8821CU as an example:

```
# tree
Linux_BT_USB_v3.10_20190430_8821CU_BTCOEX_20190509-4139.tar.xx
or
20201130_ANDROID_BT_DRIVER_RTL8821CU_COEX_v1c1c.tar.xx
|-8821CU                #firmware file
|-bluetooth_usb_driver  #usb driver

#Copy or update to the external/rkwifibt/realtek/ directory, the results are as
follows:
# external/rkwifibt/realtek$ tree
├── RTL8821CU
│   ├── rtl8821cu_config   #Bluetooth config
│   └── rtl8821cu_fw       #Bluetooth fw
├── bluetooth_usb_driver   #Bluetooth USB driver of RTL8821CU, which is compiled
into rtk_btusb.ko
│   ├── Makefile
│   ├── rtk_bt.c
│   ├── rtk_bt.h
│   ├── rtk_coex.c
│   ├── rtk_coex.h
│   ├── rtk_misc.c
│   └── rtk_misc.h
```

For Buildroot system part (The same as the Wi-Fi part, if the Wi-Fi is configured, no need to configure it again, pay attention to match the name):

```
buildroot\package\rockchip\rkwifibt\Config.in
Add the following in sequence:
+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
```

Add the compilation ko option and the install command in rkwifibt.mk, as follows:

```
+++ 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 interface Bluetooth needs to copy the firmware file to the /lib/firmware/
directory
+        $(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

#Copy the ko of the USB interface to the specified directory 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)

#Compile the ko of the USB interface, pay attention to the following
ARCH=$(RK_ARCH),it may be different in different SDKs, please refer to the
writing way of the above line
+        $(TARGET_CONFIGURE_OPTS) $(MAKE) -C $(TOPDIR)/../kernel/
M=$(@D)/realtek/bluetooth_usb_driver ARCH=$(RK_ARCH)
```

8.2 AMPAK Modules

Take AP6256 as an example, obtain the Wi-Fi and BT firmware file package of AP6256 from the module vendors (if it is updating driver case, just replace the corresponding file):

```
external\rkwifibt\firmware\broadcom\
//Create a folder named AP6256 in this directory, and store the files inside
according to the following structure
external/rkwifibt/firmware/broadcom/AP6256$ tree
├── bt
│   └── BCM4345C5.hcd
└── wifi
    ├── fw_bcm43456c5_ag.bin
    ├── fw_bcm43456c5_ag_mfg.bin
    └── nvram_ap6256.txt
//Pay attention to the directory name of AP6256, which should be consistent with
the configuration in the following Config.in of WiFi: BR2_PACKAGE_RKWIFIBT_AP6256
```

Buildroot configuration (no need to do this in updating driver case):

```
buildroot\package\rockchip\rkwifibt\Config.in
Add the followings in turn:
+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  //pay attention
to this name of "BCM4345C5.hcd" , which shoud be changed to the corresponding
model

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

There is no need to change the kernel driver part, and it is basically compatible with all AP modules. the configuration of CONFIG_AP6XXX can be used by default.

8.3 HiSilicon Wi-Fi Porting

For HiSilicon Hi3881/3861 modules:

DTS modification:

```
//DTS configuration
wireless-wlan {
```

```
    wifi_chip_type = "Hi3861L";
    WIFI,poweren_gpio = <&gpio1 RK_PC7 GPIO_ACTIVE_LOW>; //Configure the
corresponding Wi-Fi enable pin
    status = "okay";
};

sdio_pwrseq: sdio-pwrseq {
    status = "disabled"; // disable this node
};

&sdio { //Turn off the following properties
    //non-removable;
    //sd-uhs-sdr104;
    //mmc-pwrseq = <&sdio_pwrseq>;
};
```

RK code modification

```
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);
                }
```

Specify the kernel directory and the corresponding RK compiler:

```
Hi3881V100R001C00SPC021$ git diff .
diff --git a/Makefile b/Makefile
index 649da3c..d0b128d 100644
--- a/Makefile
+++ b/Makefile
@@ -4,9 +4,9 @@
 # Author      Date          Version
 # hisilion    2020-3-29     V1.0
 ###############################################################################
-KDIR=/home/shell/linux-4.9.y
+KDIR=/home/hcq/1126/kernel/
 ARCH=arm
-CROSS_COMPILE=arm-himix100-linux-
+CROSS_COMPILE=../prebuilts/gcc/linux-x86/arm/gcc-linaro-6.3.1-2017.05-
x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-
 HISILICON_PLATFORM?=

 CURDIR := $(shell if [ "$$PWD" != "" ]; then echo $$PWD; else pwd; fi)
```

Add the RK platform code to the HiSilicon driver:

```
diff --git a/driver/oal/oal_sdio_host.c b/driver/oal/oal_sdio_host.c
index 1b27742..e964b20 100644
--- a/driver/oal/oal_sdio_host.c
+++ b/driver/oal/oal_sdio_host.c
@@ -2263,10 +2263,13 @@ static  struct sdio_driver oal_sdio_driver = {
         }
 #endif
 };
-
+#include <linux/rfkill-wlan.h>
+extern int __maybe_unused rockchip_wifi_power(int on);
+extern int rockchip_wifi_set_carddetect(int val);
 hi_void sdio_card_detect_change(hi_s32 val)
 {
         printk("sdio_card_detect_change. val: %d \n", val);
+#if 0
 #if (_PRE_OS_VERSION_LINUX == _PRE_OS_VERSION)
         hisi_sdio_rescan(val);
 #else
@@ -2275,6 +2278,15 @@ hi_void sdio_card_detect_change(hi_s32 val)
                 oam_error_log0(0, 0, "sdio rescan failed.");
         }
 #endif
+
+#endif
+       printk("call rockchip carddetect api\n");
+       rockchip_wifi_set_carddetect(0);
+       rockchip_wifi_power(0);
+       msleep(50);
+       rockchip_wifi_power(1);
+       msleep(100);
+       rockchip_wifi_set_carddetect(1);
 }
```

```
#if (_PRE_OS_VERSION_LITEOS == _PRE_OS_VERSION)
```

9. Wi-Fi/BT of Debian and Other Third-Party Systems Adaptation Introduction

9.1 System Adaptation Overview

- **Debian/Kylin/UOS/Tongxin System**

There are complete Wi-Fi/BT applications on the upper layer of such systems, and our work only needs to initialize the interface before the system starts; for example, for Wi-Fi: the wlan0 node can be found from ifconfig; for Bluetooth: the hci0 node can be found from hciconfig; **Note that the DTS/kernel configuration of Wi-Fi/BT has nothing to do with the specific system, they are general configurations, and please directly refer to Chapter 2 for basic DTS and kernel configuration,** The following is going to introduce the interface initialization:

- **For Buildroot-like systems, such as Yocto**

If such systems also have upper-layer WiFiBT applications, the process is the same as above; if not:

For Wi-Fi: Generally, the **wpa_supplicant** application is used for wireless management such as WiFi scanning connection, and this type of system usually has the wpa_supplicant application package, select the corresponding configuration in the system and compile, and then refer to Chapter 4.1 for basic verification;

For Bluetooth: Generally, **bluez5 protocol stack** is used for basic operations such as scanning and connection of Bluetooth, and these systems usually include bluez5 application packages. Select the corresponding configuration and compile in the system, and start the test steps. :

```
# Binaries such as bluetoothd/bluetoothctl/hciconfig will be generated after
compilation
$ bluetoothd -ndC & # start bluetoothd
$ bluetoothctl      # enter swap mode
[bluetooth]# power on
[bluetooth]# scan on
[bluetooth]# help # To see more commands
```

Note: For the application tools and compilation mentioned above, please refer to the documentation and instructions of the corresponding system.

9.2 AMPAK Modules Adaptation Example

```
### The following takes AP6275P as an example, and the three files are obtained
from the AMPAK module vendor.
# Bluetooth initialization file: brcm_patchram_plus1.c, then compile it into an
executable file brcm_patchram_plus1 with the compiler of the system and put it in
the system
external/rkwifibt/brcm_tools/brcm_patchram_plus1.c # If there is a RKSDK, you can
get it from this directory
```

```
# BT firmware file: Stored according to the system actually used, no special requirements, the following brcm_patchram_plus1 Bluetooth initialization program will require to specify firmware path;
BCM4362A2.hcd

# Wi-Fi firmware file: stored according to the actual used system
clm_bcm43752a2_pcie_ag.blob
fw_bcm43752a2_pcie_ag.bin
fw_bcm43752a2_pcie_ag_apsta.bin
nvram_AP6275P.txt

### Configuration
# Check the kernel Wi-Fi configuration and enbale the following configurations:
CONFIG_WL_ROCKCHIP=y
CONFIG_WIFI_BUILD_MODULE=y
CONFIG_BCMDHD=y
CONFIG_AP6XXX=m
CONFIG_BCMDHD_PCIE=y #PCIE interface, mutually exclusive with SDIO, if it is not PCIE, no need to config
CONFIG_BCMDHD_SDIO=y #SDIO interface, mutually exclusive with PCIE

### Wi-Fi interface initialization
# After make compiling, ko will be generated. This file is stored in the corresponding location according to your actual needs, and you can turn on Wi-Fi to load this ko;
drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/bcmdhd.ko

# Turn on Wi-Fi: you need to load ko first, and specify the path of firmware/nvram in the parameter when in insmod. The following xx_path is changed to the actual one used:
insmod /ko_path/bcmdhd.ko firmware_path=/fw_path/ nvram_path=/nvram_path/
ifconfig -a        #Normally, you can see wlan0. If you can't, refer to Chapter 2 and Chapter 7 for troubleshooting

### Bluetooth interface initialization
# Turn on Bluetooth and reset the BT power first:
echo 0 > /sys/class/rfkill/rfkill0/state   #Turn off the BT power, which is be equal to the rfkill block operation
sleep 0.2
echo 1 > /sys/class/rfkill/rfkill0/state   #Turn on the BT power, which is be equal to the rfkill unblock operation
sleep 0.2
# Initialize Bluetooth command, --patchram specifies the path of the Bluetooth firmware file (modified according to the actual situation), /dev/ttyS8 is the serial port number of the corresponding hardware (modified according to the actual situation)
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 &
# If the system has installed the bluez protocol stack, use the hciconfig command
hciconfig -a #You can see the hci0 node, if you can't, please refer to Chapter 2 and Chapter 7 for troubleshooting

# Turn off bluetooth:
echo 0 > /sys/class/rfkill/rfkill0/state   #Turn off the BT power, which is equal to the rfkill block operation
```

```
killall brcm_patchram_plus1                #Be sure to kill the
brcm_patchram_plus1 process, because it will be executed again when it is opened,
otherwise it will conflict;

#The above enable and disable operations can be porting to your system according
to the actual situation;

# Note: If the application layer switch bluetooth calls the rfkill block to turn
off the bluetooth power, when unblock to turn on the bluetooth power again, you
must execute the brcm_patchram_plus1 bluetooth initialization command again,
otherwise bluetooth cannot be used; if the upper layer is only hciconfig hci0
down/up, no need to Call repeated initialization;
```

9.3 Realtek Module Adaptation Example

9.3.1 Adaptation Introduction

```
### The following takes RTL8822CS as an example, first get the driver package of
the corresponding module from module vendor
# Wi-Fi: RTL8822CS_WiFi_linux_v5.12.1.5-1-g0e1519e_COEX20210504-2323.20210527
# BT: 20201202_LINUX_BT_DRIVER_RTL8822C_COEX_v1c1c

### Wi-Fi Adaptation
# Please refer to chapter 8.1 to adapt Wi-Fi driver to RK platform, and refer to
chapter 2 for basic dts and kernel configuration
# After make compilation, ko will be generated. This file is stored in the
corresponding location according to your actual requirements, and you can turn on
Wi-Fi to load this ko;
drivers/net/wireless/rockchip_wlan/rkwifi/rtl8822cs/8822cs.ko

insmod /ko_path/88xxxx.ko # realtek does not need the firmware/nvram file, and
the execution timing of insmod is adjusted according to system requirements;
ifconfig -a        #Normally, you can see wlan0. If you can't, please refer to
Chapter 2 and Chapter 7 for troubleshooting

### Bluetooth adapter
# fw/config file introduction:
# Only bluetooth needs the fw/config file(the file is found in the driver
package), the storage location is related to the interface
# RTL UART interface, the file of RTL8822CS is placed in the following location
/lib/firmware/rtlbt/rtl8822cs_fw
/lib/firmware/rtlbt/rtl8822cs_config
# Copy the right FW file and config file to the correct path. (copy the
firmware/config file)
$ 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 interface: RTL8822CU corresponding file (copy the corresponding
fw/config file to the corresponding location of the system)
/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 file introduction
# Please refer to Chapter 9.2.2 Tool Compilation Introduction

# The introduction to hci_uart/usb.ko file: realtek does not use the interface
driver that comes with the kernel. The kernel must first remove the following two
configurations:
CONFIG_BT_HCIBTUSB
CONFIG_BT_HCIUART

### Initialization introduction
# UART interface:
killall rtk_hciattach #First make sure to close this process (if it was opened
before)
echo 0 > /sys/class/rfkill/rfkill0/state #Power off
sleep 0.5
echo 1 > /sys/class/rfkill/rfkill0/state #Power on
sleep 0.5
insmod /usr/lib/modules/hci_uart.ko             # The realtek module needs to
load the uart driver
rtk_hciattach -n -s 115200 /dev/ttyS4 rtk_h5 &    # ttySX refers to which uart
port Bluetooth used
# If the system has installed the bluez protocol stack, use the hciconfig command
hciconfig -a #Normally, you can see the hci0 node. If you can't, please refer to
Chapter 2 and Chapter 7 for troubleshooting

# USB interface:
echo 0 > /sys/class/rfkill/rfkill0/state #Power off
sleep 0.5
echo 1 > /sys/class/rfkill/rfkill0/state #Power on
sleep 0.5
insmod /usr/lib/modules/rtk_btusb.ko            # The realtek modules need to
load the usb driver
# If the system has installed the bluez protocol stack, use the hciconfig command
hciconfig -a #Normally, you can see the hci0 node. If you can't, please refer to
Chapter 2 and Chapter 7 for troubleshooting
```

9.3.2 Bluetooth Driver /rtk_hciattach Tool Compilation Introduction

```
### Realtek UART/USB Bluetooth driver ko driver compilation:
$ 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 specifies the kernel directory
# CROSS_COMPILE specifies the cross-compilation toolchain path
# ARCH specifies the system platform
# M specifies the uart/usb driver path
# Note that the path must be an absolute path
# Generated after successful compilation

# rtk_hciattach UART initialization program build:
$ 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 specifies the kernel directory
# CROSS_COMPILE specifies the cross-compile toolchain path
```

9.4 Automatic Installation Introduction

The above adaptation introduction are all done by manual compilation or push operations. When debugging is completed, how to simplify the steps to install automatically, the suggestions are as follows:

- If it is a mature commercial system: such as UOS/Kylin/Tongxin, they have their own methods, please consult the system provider;
- If you are using free Debian, you need to make various files mentioned above into deb packages for installation and use; please refer to the following link for making deb packages and installation documentation: https://www.debian.org/doc/
- If you use the Yocto-like Buildroot system, please refer to the compilation rules and methods used by the corresponding system;

10. Bluetooth Extension Functions

10.1 Bluetooth with Low Power Consumption

Currently only suitable for CY module + BSA protocol stack , and the AP module is being verified, and the Realtek module is supported by default; The performance achieved so far are as follows: the power consumption is reduced from about 2~3mA to about 0.5mA when the BT is turned on;

```
# Modify the script to add lpm parameters, the bsa version needs to be updated
external/broadcom_bsa/bsa_server.sh
bsa_server -r 12 -pp $hcd_file -d $ttys_dev -all=0 -lpm &

# Kernel needs to update the rfkill-bt.c file
# The above files can be obtained through redmine;
```

10.2 Bluetooth 5.0 Functional Verification (Currently only supported by RK3588 platform, more platforms will be supported in the future)

- PHY selection: If you choose a module that supports BT5.0, you can choose the following PHYs: the speed and transmission distance are greatly improved.

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

```
/* Modify according to actual requirements: */
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;
```

- Extended broadcast to support two new features:

  Broadcast data size: increased from the original 31byte to 251byte;

  Multiple broadcasts at the same time: 4.2 can only broadcast one at the same time, and 5.0 supports a maximum of 6 broadcasts at the same time;

```
# test:
# Broadcast data size test:
$ btmgmt
> privacy on
> power on
> add-adv -u 180d -u 180f -d
240954657374204C4554657374204C4554657374204C4554657374204C4554657374204C45 -c -P
1M 1

# Multiple broadcast test example
$ 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

# At the same time, install nrf connect apk with a mobile phone that supports
5.0, and you can see multiple broadcast test examples
```

Add the patchs in kernel as follows:

```
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. Other Functions and Configurations Introduction

11.1 RV1126 /RV1109 Connmand

Control Wi-Fi by Connmand (not recommended, it is no longer being maintained)

RV1109/1126 platform uses connman to manage WiFi by default, and the start way of the core process wpa_supplicant of Wi-Fi is started by:

```
# ps
//You will see the following two processes
conmand //It uses dbus to communicate with wpa_supplicant
wpa_supplicant -u //Open support for dbus communication
```

Standard usage: Wi-Fi can be operated through RV1109 web interface, please refer to the related documents of the RV1109/RV1126 platform;

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

The simple way to test terminal is as follows:

```
/ # killall ipc-daemon netserver #kill conflicting applications in upper-level
/ # connmanctl
connmanctl> enable wifi
```

```
connmanctl> scan wifi #Can scan multiple times
connmanctl> scan wifi #Can scan multiple times
connmanctl> agent on
connmanctl> services  #List the scanned Wi-Fi list
connmanctl>             *AO yyz123
wifi_c0847daf6f42_79797a313233_managed_psk
        NETGEAR75-5G
wifi_c0847daf6f42_4e455447454152373352d3547_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 #If you want to
connect to yyz123 above, the connect parameter is the following wifi_xxx_psk
connmanctl> Connected wifi_c0847daf6f42_79797a313233_managed_psk #If the
connection is successful, there will be this print
connmanctl> quit #Exit connection mode
/ # ifconfig wlan0 #You will see the IP address
```

If you want to use the traditional wpa_supplicant/wpa_cli method instead of connman, then remove the connman configuration in Buildroot:

```
BR2_PACKAGE_CONNMAN
```

And delete the related files generated before:

```
buildroot/output/rockchip_rv1126_rv1109(Adjust the directory name
actually)/target/etc/init.d/S45connman
buildroot/output/rockchip_rv1126_rv1109/target/usr/bin/connmanctl
buildroot/output/rockchip_rv1126_rv1109/target/usr/sbin/connmand
```

The steps refer to Chapter 4.1/2 for the development of the previous way.

Functions of AP hotspot:

Use Wi-Fi as the main network card by default:

```
# Download from https://github.com/rockchip-linux/softapdemo to the external
directory
# 88 port can be changed to any one that does not conflict with your application,
and for other customized modifications, please refer to chapter 3.2 of the Wf-Fi
development document
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);
```

Build and run:

```
$ make menuconfig
choose
$ make savedefconfig
$ make softap
$ ./build.sh

# Execute at boot
//Close ethernet
dbus-send --system --print-reply --dest=rockchip.dbserver /
rockchip.dbserver.net.Cmd \string:"{ \"table\": \"NetworkPower\",\"key\":
{\"sType\": \"ethernet\"}, \"data\": {\"iPower\": 0}, \"cmd\": \"Update\" }"

//Open 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\" }"

//Execute softapDemo:
$ softapDemo AP_NAME

Search for AP_NAME on the mobile phone, connect it, then open the browser and
enter: 192.168.88.1 to access the web
```

11.2 Set Static IP and Other Parameters at Boot Automatically

```
//The standard Linux ifupdown command is used in Buildroot system by default, the
corresponding boot script is in the following directory:
buildroot/package/ifupdown-scripts
> S40network //Started by /etc/init.d/rcS
//But S40network will call the ifup command to read the default configuration
script: /etc/network/interfaces
//So the default configuration can be written into the /etc/network/interfaces
file, and the way to modify the interfaces file is as follows:
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 // when more configuration needs to be
added, please check the related materials by yourself, which are all linux
standard endef
//Build upgrade: make ifupdown-scripts-dirclean && make ifupdown-scripts-rebuild

//For the modification of the default dns, Buildroot system does not have a pre-
build configuration. You have to add it manually when the DHCP process is not
running:
echo 'nameserver 114.114.114.114' >> /etc/resolv.conf // Add customized dns
configuration

//Dynamic setting
//Set IP address
ifconfig wlan0 xx.xx.xx.xx netmask xx.xx.xx.xx
//Set the default router
route add default gw xx.xx.xx.xx
//Add dns
echo 'nameserver xx.xx.xx.xx' > /etc/resolv.conf
```

11.3 DHCP Client

dhcpcd: it is used in the SDK by default and starts when the system is started. It is a dhcp client with relatively complete functions;

udhcpcd: is a compact dhcp client of busybox;

Note: these two processes must not be enabled at the same time, only one of them can be used!

If you need to get IP addresses faster when using dhcpcd client, modify as follows:

```
#Modify S41dhcpcd file
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"
        ;;

//Repackage a firmware
make dhcpcd-dirclean
make dhcpcd-rebuild
```

11.4 Wi-Fi/BT MAC Address

Generally, MAC address of Wi-Fi is built-in in the chip. If you need to customize the MAC address, you need to use RK special tool to write to customize vendor partition of Flash (Please refer to the related documents of vendor storage operation for details, we won't go into much detail here.);

AzureWave/AMPAK Wi-Fi Modules

Modify the Makefile and add the following configuration:

```
+    -DGET_CUSTOM_MAC_ENABLE
-    -DGET_OTP_MAC_ENABLE #if it exists

AMPAK: drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/Makefile
AzureWave: drivers/net/wireless/rockchip_wlan/cywdhd/bcmdhd/Makefile
```

Note: **AMPAK modules need additional modification of Wi-Fi to work normally. The first 3 bytes of the MAC address are called OUI. Each OUI corresponds to a group of macpad. If you want to modify the OUI, you need to apply to AMPAK for corresponding macpad**, then modify as follows:

Note: The new firmware of AMPAK has removed this restriction, you can ask AMPAK for the latest 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 */

        //Method 1: If you use our vendor solution and flash the customized MAC
to the vendor partition, the following function will be read from the vendor
partition.
        err = rockchip_wifi_mac_addr(buf);
        //The function is used to fill the MAC address into the first 6 positions
of buf, please refer to the above ea_example.

        // Method 2: If the MAC address is stored in your customized place, you
need to implement the read function by yourself
        // TODO: Fill the MAC address into the first 6 positions of buf, please
refer to ea_example above

        //#ifdef EXAMPLE_GET_MAC_VER2 //Define or comment out this macro to make
the following code effective
/* EXAMPLE code */
        {
                char macpad[56]= {//Replace with the macpad provided by the
vendors
                        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 AMPAK Module Compatible Version (Debian/Ubuntu)

A compatible configuration Buildroot rkwifibt is provided in the SDK for regular AMPAK chips:

```
BR2_PACKAGE_RKWIFIBT_AMPAKALL
```

This configuration can be compatible with the modules supported by SDK. It will automatically detect the Wi-Fi/BT chip model and load the corresponding firmware when it is turned on. The source code directory is:

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

The following points should be noted:

- This configuration will copy the firmware of all AMPAK modules and cause the rootfs file system to become larger, a larger flash is needed;
- This configuration does not support BSA (AMPAK Bluetooth proprietary protocol stack) compatibility, resulting in BSA unusable;

So only Debian/Ubuntu systems are recommended to use this configuration, these open source systems have their own Bluetooth protocol stack, we only need to do the initialization (rk_wifi_init).

11.6 Modify Realtek Wi-Fi Scan Time

```
//Realtek wifi scan time during each channel, modify include/rtw_mlme_ext.h
#define SURVEY_TO                   (100)  //The unit is ms, modify as required
```

11.7 Wi-Fi Country Code

Realtek modules: modify the Makefile of driver and add the following items in platform building:

```
+++ 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. By the way of proc `echo X> /proc/net/rtlxxx/wlan0/country_code`, such as:
   `echo CN> /proc/net/rtlxxx/wlan0/country_code`

2. wpa_supplicant.conf configuration parameter country=X, if it is softap, configuration parameter of hostapd.conf is country_code=X;
   Note: the way to confirm country code X can be searched through the website, such as https://countrycode.org/ to see the combination of two capital letters in ISO CODES.

   AzureWave/AMPAK Wi-Fi Modules:

   `dhd_priv country XX`

11.8 Load and Unload Wi-Fi KO Mode Dynamically

If there are multiple loading/unloading operations in Wi-Fi built in KO mode, the following patch needs to be added:

```
--- 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>; //Configure the
PIN corresponding to WIFI_REG_ON
                WIFI,host_wake_irq = <&gpio1 19 GPIO_ACTIVE_HIGH>;
                status = "okay";
            };
sdio_pwrseq: sdio-pwrseq {
+        status = "disabled"; //Disabled this node
};

&sdio { //remove the following two configurations
        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 or Ethernet UDP Packet Loss Rate Test Is Abnormal

```
# Verify by the following modifications:
# The maximum buffer for data transmission of each TCP socket, in bytes;
echo 1048576 > /proc/sys/net/core/rmem_max
echo 12582912 > /proc/sys/net/core/wmem_max            #Modify to 12M
echo 2097152 > /proc/sys/net/core/wmem_default         #Modify to 2M
echo 65535 > /proc/sys/net/core/netdev_max_backlog
#netdev_max_backlog represents the backlog of the network card device, because
the speed of network card receives packets is much faster than the speed of
kernel processes these packets, so the backlog of the network card device
appears.
```

11.10 Network Issues Troubleshooting Steps

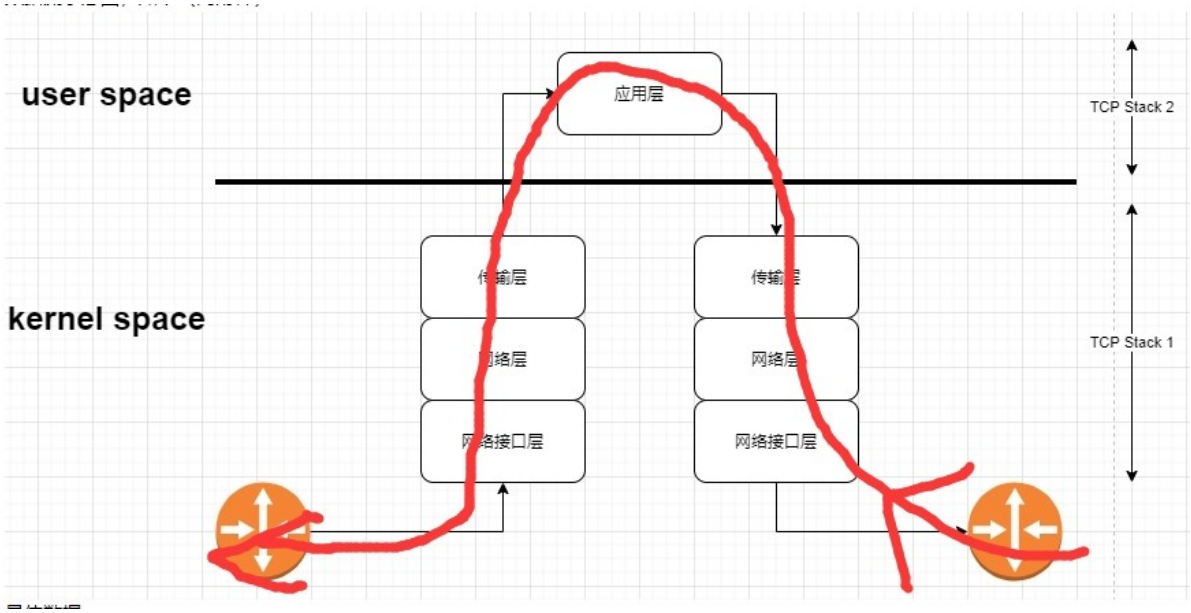
11.10.1 Stuck or Frame Loss Issue Troubleshooting

```
#Using an Network cable to check when using Ethernet

# Use tcpdump to capture packets to confirm whether there is packet loss,
# If it is UDP+RTP, it can be confirmed by the RTP sequence number;

# If there is no packet loss, the packet loss may be caused by the application
layer not receiving packets in time. You can use netstat to check the data cached
at the tcp/ip layer to confirm
netstat -n -t -u

# Turn off gro to confirm:
ethtool -K eth0 gro off
```

```
# Increase cpu frequency to the highest to confirm
echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

# Increase the tcp/ip cache buf size to confirm
#With the unit of bytes, the size of the receiving buffer area, cache data
received from the other side, and will be read by the application later
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 to check whether there is packet loss statistics

# Video playback issues: after TCP may be stuck, and UDP may cause a blurry
screen;
```

11.10.2 Simple Verification of Network Protocol Stack Processing Packet Time

Write a standard echo network program, capture packets through tcpdump, and capture packets at the two yellow circles in the figure below to calculate the time for the protocol stack to process packets.



11.11 wpa_supplicant/hostapd Version Updated

```
# The current SDK version is 2.6 + Solve the key reinstallation vulnerability of
WPA2
# The corresponding introduction and address of the patch is in:
# https://w1.fi/security/2017-1/wpa-packet-number-reuse-with-replayed-
messages.txt

# If you need to update to other new versions, the way is as follows:
# Go to the github official website to find the Buildroot repository, and go to
the following directory:
https://github.com/buildroot/buildroot/tree/master/package/
 …
# Find the two directories: wpa_supplicant and hostapd respectively, and download
them locally:
```

```
wpa_supplicant
hostapd
 …

# Go to the SDK directory:
# RK3XXX_SDK: buildroot/package/
# Directly replace the two directories wpa_supplicant and hostapd with the new
version directory downloaded above;

# The way to update the buid:
make wpa_supplicant-dirclean && make wpa_supplicant
make hostapd-dirclean && make hostapd
./build.sh rootfs
```

11.12 Debug Configuration of Driver Application

11.12.1 Wi-Fi Driver Debug

Sometimes, a more detailed log is needed to debug issues. **For Realtek chips**, please enable the following options to enable the kernel to print a more complete driver log:

```
#Modify directory: kernel/drivers/net/wireless/rockchip_wlan/rtl8xxx
#Newer driver:
Makefile
+CONFIG_RTW_DEBUG = y
+CONFIG_RTW_LOG_LEVEL = 2 #It is 2 by defaul, and it can be changed to 4 during
debugging to print more complete log information

#There is no this configuration in some old drivers , and enable the following
configuration:
include/autoconf.h
+#define CONFIG_DEBUG /* DBG_871X, etc... */
```

11.12.2 TCPDUMP Capture Packet

Network Application issues: Sometimes you need to capture packet to confirm problems, open the configuration, build and generate the tcpdump executable program, the packet capture command:

```
Choose the corresponding configuration BR2_PACKAGE_TCPDUMP in buildrooot:
tcpdump -h
tcpdump -i wlan0 -w /data/xxxx.pcap
```

11.12.3 wpa_supplicant Debugging

- Sometimes the log of wpa_supplicant is needed to debug problems:

```
#Open the following configuration in Buildroot
#Remember to save by "make savedefconfig"
+ BR2_PACKAGE_WPA_SUPPLICANT_DEBUG_SYSLOG
```

```
#Rebuild
make wpa_supplicant-dirclean
make wpa_supplicant-rebuild

#When starting wpa_supplicant, add the -s parameter so that the log will be
output to the /var/log/messages file
"  -s = log output to syslog instead of stdout"

#-d print more logs
"  -d = increase debugging verbosity (-dd even more)"

#Because there are many logs of wpa, but the size of the messages file is small,
you can change the following configuration to increase the file size
buildroot/package/busybox/S01logging
@@ -3,7 +3,7 @@
# Start logging
#
-SYSLOGD_ARGS=-n
+SYSLOGD_ARGS="-n -s 8192"

#Rebuild busybox
make busybox-dirclean
make busybox-rebuild

#Package again at last
./build.sh

#Enable wpa_supplicant and add debug options such as -s -d
wpa_supplicant -B -i wlan0 -c /xxx/wpa_supplicant.conf -s -ddd
```

11.12.4 SDIO Driver Debugging

```
#Note: Different kernel versions, the structure has changed, add the patch
according to the actual kernel version
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. Application Development

Only applicable for Buildroot original systems!

12.1 Deviceio Introduction

Deviceio (external/deviceio_release) is an application development library, which eliminating underlying complex Wi-Fi/BT operations such as wpa_supplicant/bluez/bsa, etc., and provides friendly application development interfaces. Please refer to documents in the docs\Linux\Wifibt directory for details:

```
Rockchip_Developer_Guide_DeviceIo_WIFI_CN.pdf       #Wi-Fi development
Rockchip_Developer_Guide_DeviceIo_Bluetooth_CN.pdf  #Bluetooth development
Rockchip_Developer_Guide_Network_Config_CN.pdf      #Network Config
development(BLE)
```

12.2 Configuration Build

Linux standard bluez protocol stack is used by Realtek Bluetooth: `buildroot/packages/bluez5_utils`;

AzureWave/AMPAK use the proprietary bsa protocol stack: `external/broadcom_bsa and external/bluetooth_bsa`

Configurations of different modules:

- Realtek modules (such as rtl8723ds) are configured as follows:

```
#kernel configuration
CONFIG_BT_HCIUART=n #Remember to turn off this configuration

#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
```

- AMPAK modules such as ap6255

```
#kernel configuration
CONFIG_BT_HCIUART=y #Open this configuration

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

- AzureWave modules such as AW-CM256

```
#kernel configuration
CONFIG_BT_HCIUART=y #Open this configuration

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

Compilation update:

`make menuconfig`, after modifying the configuration, you need to `make savedefconfig` to save the configuration, please refer to Chapter 2.1.

AMPAK modules:

```
# Build in the following order strictly:
make rkwifibt-dirclean && make rkwifibt-rebuild
make broadcom_bsa-dirclean && make broadcom_bsa-rebuild
make deviceio_release-dirclean && make deviceio_release
```

AzureWave modules:

```
# Build in the following order strictly:
make rkwifibt-dirclean && make rkwifibt-rebuild
make cypress_bsa-dirclean && make cypress_bsa-rebuild
make deviceio_release-dirclean && make deviceio_release
```

Realtek modules:

```
# Build in the following order strictly:
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
#Note: bluez-alsa is supported by bluez: A2DP and HFP functions
```