密级状态： 绝密(　　) 秘密(　　) 内部资料(　　) 公开(　√)
Security Class: Top-Secret (　　) Secret (　　) Internal (　　) Public (√)

Rockchip_Android8.1_WIFI 配置说明

Rockchip_Android8.1_WIFI_Configuration_Introduction

（第二系统产品部）

(Technical Department, R & D Dept. II）

文件状态： **Status:** [　] 草稿 [　] Draft [　] 正在修改 [　] Modifying [√] 正式发布 [√] Released	**文件标识：** **File No.:**	RK-SM-YF-247
	当前版本： **Current Version：**	V1.3
	作　　者： **Author:**	赵子初 Zhao Zichu
	完成日期： **Finish Date:**	2018-02-25
	审　　核： **Auditor:**	
	审核日期： **Finish Date:**	

福州瑞芯微电子股份有限公司
Fuzhou Rockchip Electronics Co., Ltd.

免责声明

本文档按“现状”提供，福州瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

Disclaimer

This document is provided “as is” and Fuzhou Rockchip Electronics Co. Ltd (“the company”) makes no express or implied statement or warranty as to the accuracy, reliability, completeness, merchantability, specific purpose and non-infringement of any statement, information and contents of the document. This document is for reference only.

This document may be updated without any notification due to product version upgrades or other reasons.

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

Brand Statement

Rockchip, Rockchip™ icon, Rockchip and other Rockchip trademarks are trademarks of Fuzhou Rockchip Electronics Co., Ltd., and are owned by Fuzhou Rockchip Electronics Co., Ltd.

All other trademarks or registered trademarks mentioned in this document are owned by their respective owners.

福州瑞芯微电子股份有限公司
Fuzhou Rockchip Electronics Co., Ltd.
地址： 福建省福州市铜盘路软件园 A 区 18 号
网址： www.rock-chips.com
客户服务电话： +86-4007-700-590
客户服务传真： +86-591-83951833
客户服务邮箱： fae@rock-chips.com

Fuzhou Rockchip Electronics Co., Ltd.
Address: 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

版 本 历 史 Revision History

版本号 Version no.	作者 Author	修改日期 Revision Date	修改说明 Revision description	备注 Remark
V1.0	ZZC	2017-11-15	发布初稿 Initial version release	
V1.1	ZZC	2017-12-05	编译 kernel 时自动编译 wifi ko Automatically compile wifi ko when compiling kernel	
V1.2	ZZC	2017-12-15	增加 wifi 问题排查 Add wifi issues analysis	
V1.3	ZZC	2018-02-25	增加 Android 注意事项 Add Android notices	

目 录 Contents

1 目的 Purpose

明确 android 8.1 平台上 wifi 自动兼容原理和注意事项，按照本文档 wifi 提供的完全自动兼容说明生成固件后，即可支持相应的 wifi 模块，并且一套固件可以支持多个 WIFI 模块。

To introduce wifi auto compatibility principle and notices based on Android8.1 platform. After generating images according to the wifi complete auto compatibility instruction provided in this document, it can support the corresponding wifi module and one set of images can support multiple WIFI modules.

按照本文提供的方法，android 8.1 平台 wifi 可实现完全自动兼容，android 和 kernel 无需任何额外配置。

Using the method provided in this document, Android8.1 platform Wi-Fi can achieve complete auto compatibility without any additional configuration of Android and kernel.

1.1 Kernel 注意事项 Kernel notices

wifi 完全自动兼容方案，AP6xxx 系列 wifi 和 Realtek 系列 wifi 驱动必须采用 module 方式，不能 build in 到内核 kernel.img 中；如果希望采用 build in 驱动到内核，参考第 7 章节进行；采用自动兼容方案需要确认各 andorid 8.1 平台内核使用的 conifig；**确认 defconfig 是将 wifi 驱动编译成 ko modules 的配置，defconfig 参考如下配置，如下配置基于 SDK 对外最新内核代码：**

For Wi-Fi complete auto compatibility solution, the drivers of AP6xxx series wifi and Realtek series wifi must use module method and cannot be built in kernel.img. If want to build in the driver to kernel, perform referring to chapter 7. Using the auto compatibility solution requires to confirm the conifig used by kernel on Android8.1 platforms. **Confirm defconfig is the configuration compiling the wifi driver as ko modules. Defconfig refers to the following configuration which is based on SDK latest kernel code released:**

CONFIG_WL_ROCKCHIP=y
CONFIG_WIFI_BUILD_MODULE=y
 # CONFIG_WIFI_LOAD_DRIVER_WHEN_KERNEL_BOOTUP is not set
CONFIG_AP6XXX=m
CONFIG_RTL_WIRELESS_SOLUTION=y
CONFIG_RTL8188EU=m
CONFIG_RTL8188FU=m
CONFIG_RTL8189ES=m
CONFIG_RTL8189FS=m
CONFIG_RTL8723BS=m
CONFIG_RTL8723BU=m
CONFIG_RTL8723CS=m
CONFIG_RTL8723DS=m

Device Drivers --->

[*] Network device support --->

[*] Wireless LAN --->

[*] Rockchip Wireless LAN support --->

```
--- Rockchip Wireless LAN support
[*]   build wifi ko modules
[*]   Wifi load driver when kernel bootup
<M>   ap6xxx wireless sdio cards support
[*]   Realtek Wireless Device Driver Support  ----
<M>   Realtek 8188E USB WiFi
<M>   Realtek 8188F USB WiFi
<M>   Realtek 8189E SDIO WiFi
<M>   Realtek 8189F SDIO WiFi
<M>   Realtek 8723B SDIO or SPI WiFi
<M>   Realtek 8723B USB WiFi
<M>   Realtek 8723C SDIO or SPI WiFi
<M>   Realtek 8723D SDIO or SPI WiFi
```

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

Board level dts doesn't need to configure WIFI chip type (also ok if configured), because wifi driver loading is not dependent on wifi_chip_type node. If WIFI is not designed according to the hardware reference design published by RK, board level dts should confirm the following information first:

(注意 kernel4.4 的内核 wifi power 脚改到 sdio 中配置 **reset-gpios**)

(Note: for kernel4.4, wifi power pin should be configured in sdio and then filled to **reset-gpios**)

```
sdio_pwrseq: sdio-pwrseq {
                compatible = "mmc-pwrseq-simple";
                clocks = <&rk818 1>;
                clock-names = "ext_clock";
                pinctrl-names = "default";
                pinctrl-0 = <&wifi_enable_h>;
                /*
                 * On the module itself this is one of these (depending
                 * on the actual card populated):
                 * - SDIO_RESET_L_WL_REG_ON
                 * - PDN (power down when low)
                 */
                reset-gpios = <&gpio3 4 GPIO_ACTIVE_LOW>; //wifi power
        };
wireless-wlan {
        compatible = "wlan-platdata";
```

```
        wifi_chip_type ="ap6255"; //或者配置为 or configured as wifi_chip_type = "";
        WIFI,host_wake_irq = <&gpio0 GPIO_D4 GPIO_ACTIVE_HIGH>;
        status = "okay";
    };
```

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

Note: current WIFI complete compatibility solution, based on WIFI reference design published by RK, WIFI power pin is high active by default. Need to confirm the WIFI power pin and high/low active configuration for the specific projects.

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

If one set of images should be compatible with all WIFI modules in RK support list, WIFI power pin of the hardware board (the hardware type of all boards should keep consistent) and SDIO level should be confirmed in advance. There is elaboration on the hardware notices in chapter 4.

1.2 Android 注意事项 Android notices

编译 kernel 时会自动编译 wifi ko 文件，生成固件的时候会自动将 wifi ko 文件拷贝到 vendor.img 中，存放在/vendor/lib/modules/wifi/目录下。

It will automatically compile wifi ko file when compiling kernel. When generating the images, it will automatically copy wifi ko file to vendor.img and save in the directory of /vendor/lib/modules/wifi/.

执行脚本 mkimage.sh

Execute the script mkimage.sh

```
if [ `grep "CONFIG_WIFI_BUILD_MODULE=y" $KERNEL_CONFIG` ]; then
    echo "Install wifi ko to $TARGET_OUT_VENDOR/lib/modules/wifi/"
    mkdir -p $TARGET_OUT_VENDOR/lib/modules/wifi
    find kernel/drivers/net/wireless/rockchip_wlan/*  -name "*.ko" | xargs -n1 -i cp {} $TARGET_OUT_VENDOR/lib/modules/wifi/
fi
```

2 WIFI 兼容原理简要说明 Brief introduction of WIFI compatibility principle

2.1 WIFI 芯片识别流程 WIFI chip recognition process

1. 开机对 wifi 模块上电，并自动进行扫描 sdio 操作。

   Power up wifi module, and automatically scan sdio.

2. 系统启动打开 wifi 操作时，分别对系统 sys/bus/sdio（sdio wifi）， sys/bus/usb(usb wifi)，

sys/bus/pic (pcie wifi)文件系统下的 uevent 进行读取。

When starting wifi, the system separately reads uevent from the file system of sys/bus/sdio(sdio wifi), sys/bus/usb(usb wifi) and sys/bus/pic (pcie wifi).

3. 获取到 wifi 芯片 vid pid 加载相应的 wifi ko 驱动。

   Load the corresponding wifi ko driver after acquiring vid pid of the wifi chip.

4. 识别到 wifi 类型后加载不同的 wpa_supplicant 参数启动 wifi。

   Load different wpa_supplicant parameter to start wifi after recognizing the wifi type.

核心代码目录：

The directory of core code:

android /frameworks/opt/net/wifi

kernel/net/rfkill/rfkill-wlan.c

hardware/broadcom

external/wpa_supplicant_8

3 如何添加新 WIFI 模块的支持 How to add new wifi module support

目前对外发布的 andriod8.1 SDK，WIFI 自动兼容框架已经搭建完毕，如果客户需要自行调试其他模块，只需按照本章节提到的修改地方进行修改即可。

Currently released android8.1 SDK already builds in the WIFI auto compatibility framework. If customers need to debug other module by self, only need to modify the parts mentioned in this chapter.

3.1 WIFI 驱动移植 WIFI driver porting

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

All WIFI module drivers on RK platforms are put in the directory of kernel/drivers/net/wireless/rockchip_wlan. Generally for porting new WIFI driver, need to add Kconfig and Makefile corresponding to the wifi module in the directory of kernel/drivers/net/wireless. Some module also requires to modify Kconfig and Makefile of wifi driver (according to the specific wifi module driver). If using Realtek module, you can refer to the document RealTek wifi 驱动移植说明_V1.1.pdf.

内核能正确编译出 wifi ko 驱动文件，wifi ko 文件在编译 kernel 的时候会自动编译。

Kernel can correctly compile wifi ko driver file. The wifi ko file will be automatically compiled when compiling kernel.

注意：由于目前 wifi 驱动是采用 ko 方式，如果有修改内核网络相关配置，一定要重新编译

ko，否则很可能导致 wifi ko 和内核网络协议栈不匹配。

Note: Because current wifi driver is using ko method, if kernel network related configuration is modified, remember to re-compile ko, otherwise it will probably lead to the mismatch of wifi ko with kernel network protocol stack.

3.2 添加 wifi 兼容 Add wifi compatibility

1. 添加 wifi 名称和 wifi vid pid

   Add wifi name and wifi vid pid

   源码路径：frameworks/opt/net/wifi/libwifi_hal/rk_wifi_ctrl.cpp 代码 **supported_wifi_devices[]**结构体中添加 wifi 模块的名称和对应 vid pid，vid pid 可以根据下面章节手动读取 uevent 进行查看；以 AP6255 为例：AP6255 为模块名称，02d0:09bf 为 vid pid，如下列表中已经添加了几款 wifi 的兼容，参考如下格式添加：

   Source code path: Add wifi module name and corresponding vid pid in **supported_wifi_devices[]** struct in frameworks/opt/net/wifi/libwifi_hal/rk_wifi_ctrl.cpp. Manually read uevent according to the following chapter can check vid pid. Take AP6255 as example: AP6255 is the module name, 02d0:09bf is vid pid, the following table already adds several wifi modules referring to the following format:

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

static wifi_device supported_wifi_devices[] = {
        {"RTL8188EU",    "0bda:8179"},
        {"RTL8188EU",    "0bda:0179"},
        {"RTL8723BU",    "0bda:b720"},
        {"RTL8723BS",    "024c:b723"},
        {"RTL8822BS",    "024c:b822"},
        {"RTL8723CS",    "024c:b703"},
        {"RTL8723DS",    "024c:d723"},
        {"RTL8188FU",    "0bda:f179"},
        {"RTL8822BU",    "0bda:b82c"},
        {"RTL8189ES",    "024c:8179"},
        {"RTL8189FS",    "024c:f179"},
        {"RTL8192DU",    "0bda:8194"},
        {"RTL8812AU",    "0bda:8812"},
        {"SSV6051",      "3030:3030"},
        {"ESP8089",      "6666:1111"},
        {"AP6354",       "02d0:4354"},
        {"AP6330",       "02d0:4330"},
        {"AP6356S",      "02d0:4356"},
        {"AP6335",       "02d0:4335"},
        {"AP6255",       "02d0:a9bf"},
        {"RTL8822BE",    "10ec:b822"},
};
```

2. 添加 wifi 驱动 ko 文件存放路径

   Add the path to save the ko file of wifi driver

   frameworks/opt/net/wifi/libwifi_hal/wifi_hal_common.cpp 中 **wifi_ko_file_name module_list**[]结构体存放的是 wifi 模块的 ko 驱动存放路径和加载 wifi ko 驱动所需的参数，wifi ko 存放路径统一采用 XXXX_DRIVER_MODULE_PATH 的命名方式。

   The ko driver of wifi module and the parameters required for loading wifi ko driver are saved in

wifi_ko_file_name module_list[] struct in frameworks/opt/net/wifi/libwifi_hal/wifi_hal_common.cpp. The path of wifi ko is named as XXXX_DRIVER_MODULE_PATH.

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

If ismod wifi ko doesn't have parameter, you can use UNKKOWN_DRIVER_MODULE_ARG. If need additional parameter, please perform relative operation according to wifi module porting document.

注意：wifi 名称要与 supported_wifi_devies[]结构体中定义的名称一样。

Note: wifi name should be the same as the name defined in supported_wifi_devies[] struct.

```
#define RTL8188EU_DRIVER_MODULE_PATH      "/vendor/lib/modules/8188eu.ko"
#define RTL8723BU_DRIVER_MODULE_PATH      "/vendor/lib/modules/8723bu.ko"
#define RTL8723BS_DRIVER_MODULE_PATH      "/vendor/lib/modules/8723bs.ko"
#define RTL8723BS_VQ0_DRIVER_MODULE_PATH "/vendor/lib/modules/8723bs-vq0.ko"
#define RTL8723CS_DRIVER_MODULE_PATH      "/vendor/lib/modules/8723cs.ko"
#define RTL8723DS_DRIVER_MODULE_PATH      "/vendor/lib/modules/8723ds.ko"
#define RTL8188FU_DRIVER_MODULE_PATH      "/vendor/lib/modules/8188fu.ko"
#define RTL8822BU_DRIVER_MODULE_PATH      "/vendor/lib/modules/8822bu.ko"
#define RTL8822BS_DRIVER_MODULE_PATH      "/vendor/lib/modules/8822bs.ko"
#define RTL8189ES_DRIVER_MODULE_PATH      "/vendor/lib/modules/8189es.ko"
#define RTL8189FS_DRIVER_MODULE_PATH      "/vendor/lib/modules/8189fs.ko"
#define RTL8192DU_DRIVER_MODULE_PATH      "/vendor/lib/modules/8192du.ko"
#define RTL8812AU_DRIVER_MODULE_PATH      "/vendor/lib/modules/8812au.ko"
#define RTL8822BE_DRIVER_MODULE_PATH      "/vendor/lib/modules/8822be.ko"
#define SSV6051_DRIVER_MODULE_PATH        "/vendor/lib/modules/ssv6051.ko"
#define ESP8089_DRIVER_MODULE_PATH        "/vendor/lib/modules/esp8089.ko"
#define BCM_DRIVER_MODULE_PATH            "/vendor/lib/modules/bcmdhd.ko"
#define DRIVER_MODULE_PATH_UNKNOW         ""
```

```
#define RTL8188EU_DRIVER_MODULE_NAME      "8188eu"
#define RTL8723BU_DRIVER_MODULE_NAME      "8723bu"
#define RTL8723BS_DRIVER_MODULE_NAME      "8723bs"
#define RTL8723BS_VQ0_DRIVER_MODULE_NAME "8723bs-vq0"
#define RTL8723CS_DRIVER_MODULE_NAME      "8723cs"
#define RTL8723DS_DRIVER_MODULE_NAME      "8723ds"
#define RTL8188FU_DRIVER_MODULE_NAME      "8188fu"
#define RTL8822BU_DRIVER_MODULE_NAME      "8822bu"
#define RTL8822BS_DRIVER_MODULE_NAME      "8822bs"
#define RTL8189ES_DRIVER_MODULE_NAME      "8189es"
#define RTL8189FS_DRIVER_MODULE_NAME      "8189fs"
#define RTL8192DU_DRIVER_MODULE_NAME      "8192du"
#define RTL8812AU_DRIVER_MODULE_NAME      "8812au"
#define RTL8822BE_DRIVER_MODULE_NAME      "8822be"
#define SSV6051_DRIVER_MODULE_NAME        "ssv6051"
#define ESP8089_DRIVER_MODULE_NAME        "esp8089"
#define BCM_DRIVER_MODULE_NAME            "bcmdhd"
#define DRIVER_MODULE_NAME_UNKNOW         ""
```

```
wifi_ko_file_name module_list[] =
{
        {"RTL8723BU", RTL8723BU_DRIVER_MODULE_NAME, RTL8723BU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8188EU", RTL8188EU_DRIVER_MODULE_NAME, RTL8188EU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8192DU", RTL8192DU_DRIVER_MODULE_NAME, RTL8192DU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8822BU", RTL8822BU_DRIVER_MODULE_NAME, RTL8822BU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8822BS", RTL8822BS_DRIVER_MODULE_NAME, RTL8822BS_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8188FU", RTL8188FU_DRIVER_MODULE_NAME, RTL8188FU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8189ES", RTL8189ES_DRIVER_MODULE_NAME, RTL8189ES_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8723BS", RTL8723BS_DRIVER_MODULE_NAME, RTL8723BS_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8723CS", RTL8723CS_DRIVER_MODULE_NAME, RTL8723CS_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8723DS", RTL8723DS_DRIVER_MODULE_NAME, RTL8723DS_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8812AU", RTL8812AU_DRIVER_MODULE_NAME, RTL8812AU_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8189FS", RTL8189FS_DRIVER_MODULE_NAME, RTL8189FS_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"RTL8822BE", RTL8822BE_DRIVER_MODULE_NAME, RTL8822BE_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"SSV6051",     SSV6051_DRIVER_MODULE_NAME,   SSV6051_DRIVER_MODULE_PATH, SSV6051_DRIVER_MODULE_ARG},
        {"ESP8089",     ESP8089_DRIVER_MODULE_NAME,   ESP8089_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"AP6335",          BCM_DRIVER_MODULE_NAME,       BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"AP6330",          BCM_DRIVER_MODULE_NAME,       BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"AP6354",          BCM_DRIVER_MODULE_NAME,       BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"AP6356S",         BCM_DRIVER_MODULE_NAME,       BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"AP6255",          BCM_DRIVER_MODULE_NAME,       BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"APXXX",           BCM_DRIVER_MODULE_NAME,       BCM_DRIVER_MODULE_PATH, UNKKOWN_DRIVER_MODULE_ARG},
        {"UNKNOW",       DRIVER_MODULE_NAME_UNKNOW,    DRIVER_MODULE_PATH_UNKNOW, UNKKOWN_DRIVER_MODULE_ARG}

};
```

3.3 添加 wpa_supplicant 启动参数 Add wpa_supplicant start parameter

init.connectivity.rc 中启动 wpa_supplicant 参数从/vendor/etc/wifi/wpa_config.txt 文件中读取。

Read the parameter starting wpa_supplicant in init.connectivity.rc from /vendor/etc/wifi/wpa_config.txt file.

service wpa_supplicant /vendor/bin/hw/wpa_supplicant \

/vendor/etc/wifi/wpa_config.txt

class main

socket wpa_wlan0 dgram 660 wifi wifi

disabled

oneshot

wpa_config.txt 代码中存放在 deivice/rockchip/common/wpa_config.txt

wpa_config.txt code is saved in deivice/rockchip/common/wpa_config.txt

wpa_supplicant/main.c 中 main 函数入口会根据芯片类型来选择读取不同的 wpa_supplicant 参数。

The entry of main function in wpa_supplicant/main.c will select to read different wpa_supplicant parameter according to the chip type.

以 broadcom 模块为例：

Take Broadcom module as example:

当读取到芯片是 APxxx 系列的，就会根据 BROADCOM_MODULE_NAME 在/vendor/etc/wifi/wpa_config.txt 文件中查找 broadcom 模块的参数。

When reading the chip is APxxx series, it will look for the parameter of Broadcom module in /vendor/etc/ wifi/wpa_config.txt file according to BROADCOM_MODULE_NAME.

```
int main(int argc, char *argv[])
{
        int ret = -1;
        char module_type[20]={0};

        wpa_printf(MSG_INFO,"argc = %d\n",argc);
        if(argc == 2) {
                if (wifi_type[0] == 0) {
                        check_wifi_chip_type_string(wifi_type);
                }
                wpa_printf(MSG_INFO,"Current wifi chip is %s\n",wifi_type);
                if (0 == strncmp(wifi_type, "RTL", 3)) {
                        wpa_printf(MSG_INFO,"Start rtl_wpa_supplicant\n");
                        ret = read_wpa_param_config(REALTEK_MODULE_NAME,argv[1]);
                } else if (0 == strncmp(wifi_type, "AP", 2)) {
                        wpa_printf(MSG_INFO,"Start bcm_wpa_supplicant\n");
                        ret = read_wpa_param_config(BROADCOM_MODULE_NAME,argv[1]);
                } else if (0 == strncmp(wifi_type, "SSV", 3)) {
                        wpa_printf(MSG_INFO,"Start ssv_wpa_supplicant\n");
                        ret = read_wpa_param_config(SSV_MODULE_NAME,argv[1]);
                } else if (0 == strncmp(wifi_type, "ESP", 3)) {
                        wpa_printf(MSG_INFO,"Start esp_wpa_supplicant\n");
                        ret = read_wpa_param_config(ESP_MODULE_NAME,argv[1]);
                } else {
                        wpa_printf(MSG_INFO,"Start wpa_supplicant\n");
                        sprintf(module_type,"[%s]",wifi_type);
                        ret = read_wpa_param_config(module_type,argv[1]);
                }
        } else {
                wpa_printf(MSG_INFO,"Start wpa_supplicant\n");
                ret = main_loop(argc, argv);
        }
        return ret;
}
```

```
#define BROADCOM_MODULE_NAME "[broadcom]"
```

wpa_config.txt 中找到参数如下：

The parameter found in wpa_config.txt is as below:

```
[broadcom]
/vendor/bin/hw/wpa_supplicant
-O/data/vendor/wifi/wpa/sockets
-puse_p2p_group_interface=1
-g@android:wpa_wlan0
```

目前 wpa_config.txt 文件中已添加：

Currently wpa_config.txt file already adds:

[broadcom] [realtek] [ssv] [esp]四个厂家的启动参数，如新增模块不在列表内，请按照以上格式进行添加。

[broadcom] [realtek] [ssv] [esp] four vendors' start parameter. If new module is not in the list, please add according to the above format.

4 WIFI 兼容软硬件注意事项 WIFI compatibility software/hardware notices

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

clk 最高跑 150M。

Currently released SDK can be compatible with sdio 2.0 and sdio 3.0 wifi using one set of images, supporting sdio2.0 clk up to 50M, and sdio 3.0 clk up to 150M.

SDIO 配置请参考《Rockchip SDMMC SDIO eMMC 开发指南 V1.0-20160630.pdf》

For SDIO configuration, please refer to 《Rockchip SDMMC SDIO eMMC 开发指南 V1.0-20160630.pdf》.

5 WIFI ko 编译注意事项 WIFI ko compiling notices

本章节主要说明内核网络相关配置修改，对应 wifi ko 驱动的编译方法。

This chapter mainly describes the modification of kernel network related configuration and compiling method corresponding to wifi ko driver.

wifi ko 要跟内核网络配置编译出的 kernel.img 一致。当调试需要**单独烧写 kernel.img 时**，如果内核有修改网络配置，需要执行 mkimage.sh 后重新生成 vendor.img，kernel.img 和 vendor.img 并烧写。

Wifi ko should be consistent with kernel.img compiled by kernel network configuration. When need to separately flash kernel.img for debugging, if kernel modifies the network configuration, need to re-generate vendor.img after executing mkimage.sh, both kernel.img and vendor.img should be flashed.

6 WIFI 驱动加载方式说明 WIFI driver loading method introduction

如果**不需要 wifi 自动兼容方案**，可以将 wifi 驱动 build in 到内核，由客户自行决定。

If you don’t need wifi auto compatibility solution, you can build wifi driver into kernel. Customers can make decision by themselves.

以 AP6255 为例，配置如下：

Take AP6255 as example, the configuration is as below:

WIFI_BUILD_MODULE = n

CONFIG_AP6XXX = y

```
--- Rockchip Wireless LAN support
[ ]   build wifi ko modules
[*]   Wifi load driver when kernel bootup
<*>   ap6xxx wireless sdio cards support
[ ]   Realtek Wireless Device Driver Support  ----
< >   Realtek 8188E USB WiFi
< >   Realtek 8188F USB WiFi
< >   Realtek 8189E SDIO WiFi
< >   Realtek 8189F SDIO WiFi
< >   Realtek 8723B SDIO or SPI WiFi
< >   Realtek 8723B USB WiFi
< >   Realtek 8723C SDIO or SPI WiFi
< >   Realtek 8723D SDIO or SPI WiFi
< >   Realtek 8822B PCIE WiFi
```

7 WIFI 无法打开问题排查 Issue analysis of failing to open WIFI

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

First confirm if system has USB WIFI or SDIO wifi device after power up, after power up normally, you can confirm first through kernel log, if it is sdio, wifi will have the sdio recognition success log as below:

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

如果是 usb，wifi 会有类似如下 usb 信息:

If it is usb, wifi will have the usb information similar as below:

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

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

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

usb 2-1: Product: 802.11ac NIC

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

If there is no usb information and sdio recognition success log information, it means wifi module is not powered up normally or sdio recognition fails, need to double check if the hardware has problem and wifi pin in software dts is configured correctly. Because powering up the module successfully is the precondition to detect wifi chip id as described in chapter 2.1.

另外也可以 cat 如下路径下的 uevent 文件进行确认：

Besides, you can also cat the uevent file in the following path to confirm:

sys/bus/sdio/devices

sys/bus/usb/devices

以 AP6255 为例， cat 对应的 uevent 可以看到 ap6255 对应的 pid 和 vid（02D0：A9BF）

Take AP6255 as example, cat the corresponding uevent can see the pid and vid corresponding to ap6255 (02D0：A9BF):

```
rk3368:/ #
rk3368:/ # cat /sys/bus/sdio/devices/mmc1\:0001\:3/uevent
SDIO_CLASS=02
SDIO_ID=02D0:A9BF
MODALIAS=sdio:c02v02D0dA9BF
rk3368:/ #
```

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

If the module is powered up normally and it can also recognize vid pid of wifi module, then confirm if wifi ko driver file is correctly loaded.

以 AP6255 wifi 模块为例，如下是正常加载 ko 的 log 信息：

Take AP6255 wifi module as example, the following is log information when normally loading ko:

```
130:rk3368:/ # logcat |  grep rk_wifi_hal
01-05 01:39:21.860   268   268 E /vendor/bin/hw/rk_wifi_hal: Kernel version is 4.4.: No such file or directory
01-05 01:39:21.860   268   268 E /vendor/bin/hw/rk_wifi_hal: Wifi driver is not ready.: No such file or directory
01-05 01:39:21.870   268   268 E /vendor/bin/hw/rk_wifi_hal: found device pid:vid :02d0:a9bf: Permission denied
01-05 01:39:21.870   268   268 E /vendor/bin/hw/rk_wifi_hal: check_wifi_chip_type_string : AP6255: Permission denied
01-05 01:39:21.874   268   268 E /vendor/bin/hw/rk_wifi_hal: matched ko file path /vendor/lib/modules/wifi/bcmdhd.ko: No such file or directory
01-05 01:39:22.298   268   268 E /vendor/bin/hw/rk_wifi_hal: Kernel version is 4.4.: No such file or directory
01-05 01:39:22.298   268   268 E /vendor/bin/hw/rk_wifi_hal: Wifi driver is not ready.: No such file or directory
01-05 01:39:22.500   268   268 E /vendor/bin/hw/rk_wifi_hal: Kernel version is 4.4.: No such file or directory
01-05 01:39:22.501   268   268 E /vendor/bin/hw/rk_wifi_hal: Wifi driver is not ready.: No such file or directory
01-05 01:39:22.703   268   268 E /vendor/bin/hw/rk_wifi_hal: Kernel version is 4.4.: No such file or directory
01-05 01:39:22.703   268   268 E /vendor/bin/hw/rk_wifi_hal: Wifi driver is ready for now...: No such file or directory
```

Wifi 模块与相应 ko 的对应关系：

The relationship between wifi module and corresponding ko:

模块名称 Module name	ko 名称 ko name
AP6xxx(SDIO)	bcmdhd.ko
RTL8723BU(USB)	8723bu.ko
RTL8188EU(USB)	8188eu.ko
RTL8188EU(USB)	8188fu.ko
RTL8723BS(SDIO)	8723bs.ko
RTL8723CS(SDIO)	8723cs.ko
RTL8723DS(SDIO)	8723ds.ko
RTL8189ES(SDIO)	8189es.ko
RTL8189FS(SDIO)	8189fs.ko

详细的 ko 和模块对应关系详见 **android/frameworks/opt/net/wifi/libwifi_hal/wifi_hal_common.cpp 文件。**

The detailed relationship between ko and the module refers to **android/frameworks/opt/net/wifi/libwifi_hal/wifi_hal_common.cpp file.**

3. 如果 wifi ko 匹配错误或者没有相应的 ko 文件，请确认按照 v1.1 和 v1.2 章节进行确认，正常系统 ko 文件存放路径如下：

If wifi and ko don’t match or there is no corresponding ko file, please confirm according to chapter 1.1 and 1.2. The normal path of system ko file is as below:

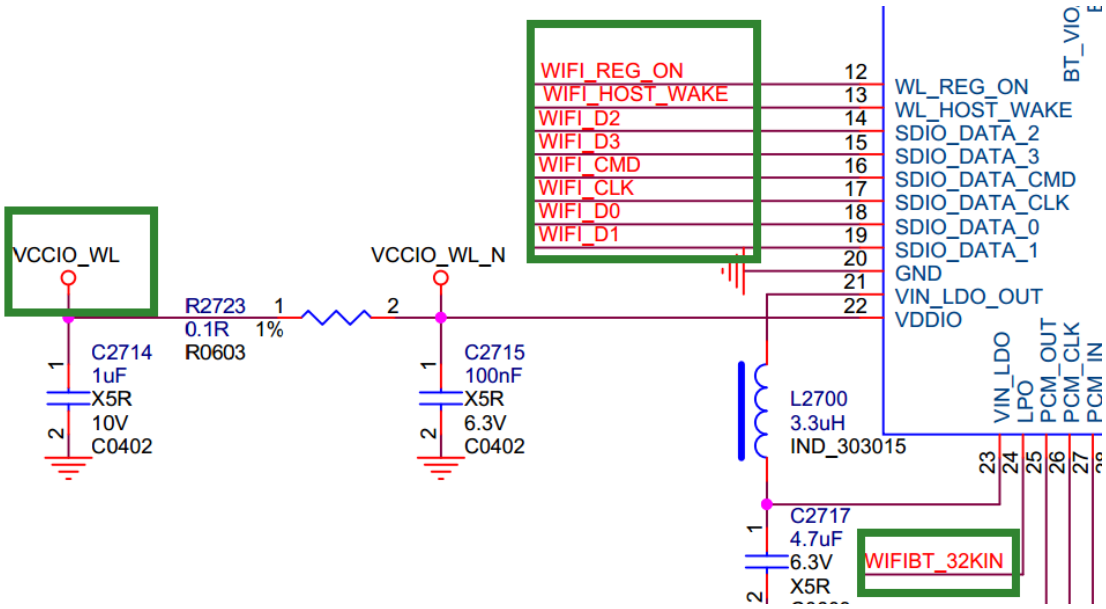

```
rk3368:/vendor/lib/modules/wifi # ls -al
total 39196
drwxr-xr-x 2 root shell     4096 2017-12-10 21:52 .
drwxr-xr-x 3 root shell     4096 2017-12-10 21:53 ..
-rw-r--r-- 1 root root    3002696 2017-12-14 01:06 8188eu.ko
-rw-r--r-- 1 root root    2414232 2017-12-14 01:06 8188fu.ko
-rw-r--r-- 1 root root    2175416 2017-12-14 01:06 8189es.ko
-rw-r--r-- 1 root root    2507328 2017-12-14 01:06 8189fs.ko
-rw-r--r-- 1 root root    2023896 2017-12-14 01:06 8723bs.ko
-rw-r--r-- 1 root root    2691448 2017-12-14 01:06 8723bu.ko
-rw-r--r-- 1 root root    3037080 2017-12-14 01:06 8723cs.ko
-rw-r--r-- 1 root root    3215960 2017-12-14 01:06 8723ds.ko
-rw-r--r-- 1 root root    3578928 2017-12-14 01:06 8822be.ko
-rw-r--r-- 1 root root   15451976 2017-12-14 01:06 bcmdhd.ko
rk3368:/vendor/lib/modules/wifi #
```

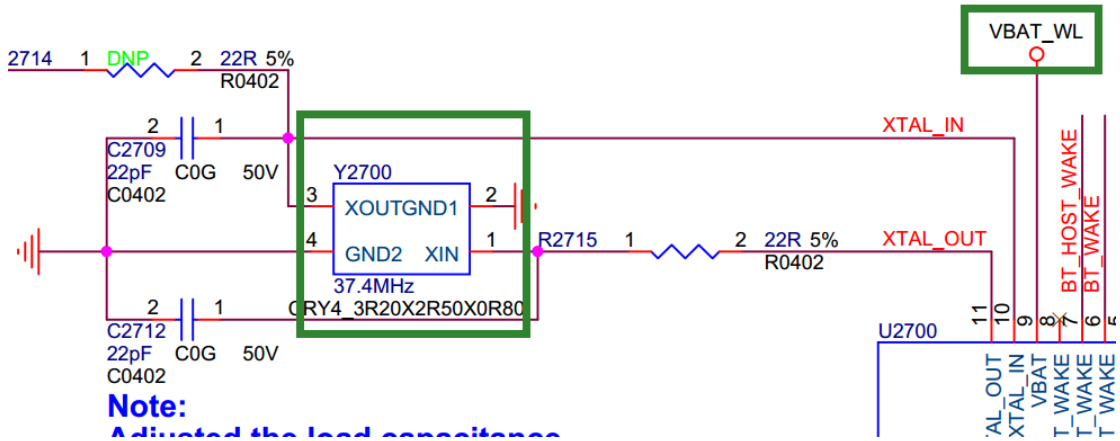
4. 注意：由于目前 wifi 驱动是采用 ko 方式，如果有修改内核网络相关配置，一定要重新编译 ko，否则很可能加载 wifi ko 引起内核 panic，请详细阅读本文第 5 章节。

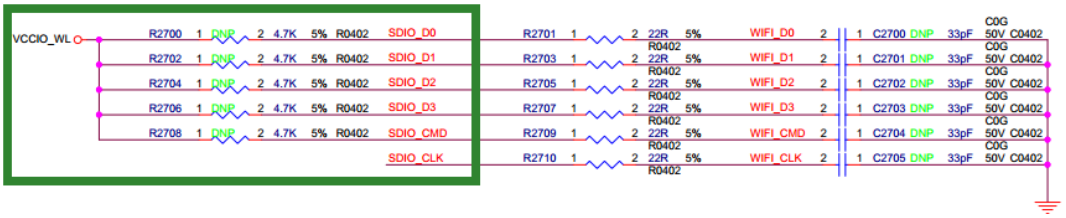
Note: Because currently wifi driver is using ko method, if kernel network related configuration is modified, remember to re-compile ko, otherwise it will probably lead to kernel panic when loading wifi ko. Please read chapter 5 carefully.

8 SDIO 问题排查 SDIO issue analysis

8.1 硬件部分 Hardware part







首先看下硬件：主要的部分都在绿色方框内

Firstly check the hardware: main parts are marked as the green boxes.

WIFI_D0~3：数据线，平时为高，电压取决于 VCCIO_WL 的电压；

WIFI_D0~3: data line, usually high, the voltage depends on VCCIO_WL.

WIFI_CMD：命令线，平时为高，电压取决于 VCCIO_WL 的电压；

WIFI_CMD: command line, usually high, the voltage depends on VCCIO_WL.

WIFI_CLK：时钟，平时为低，电压取决于 VCCIO_WL 的电压；

WIFI_CLK: clock, usually low, the voltage depends on VCCIO_WL.

VBAT_WL：WIFI 模组供电电源，一直都为高，供电需打印 3.3v；

VBAT_WL: power supply of WIFI module, always high, need to print 3.3V for power supply.

VCCIO_WL：给 DATA/CMD/CLK 的 IO 供电电源，可以为 3.3 或者 1.8v，但 SDIO3.0 必须为 1.8v；

VCCIO_WL: supply power for IO of DATA/CMD/CLK, can be 3.3V or 1.8V, but SDIO3.0 must be 1.8V.

WIFI_REG_ON：正常工作时为 3.3v，WiFi 关闭时为 0v；

WIFI_REG_ON: 3.3V for normal working, 0V when Wi-Fi is disabled.

两个晶振：32K 和 26M/37.4M,正常工作时都会有波形输出；

Two crystal oscillators: 32K and 26M/37.4M, there will be waveform output for normal working.

遇到 WIFI SDIO 问题时：

When encountering WIFI SDIO issues:

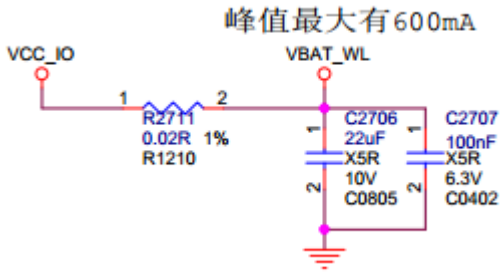
1、 首先以上的电压以及晶振问题；

Firstly check the above voltage and crystal oscillator issue.

2、 VBAT_WL 和 VCCIO_WL 的电源是长供电，但有时候会因为这两路电源和其他模块共用一个

电源，可能会出现电压塌陷问题，导致 WIFI 模组异常；比如下图，有非常多外设都会挂在 VCC_IO 上面；

The power supply of VBAT_WL and VCCIO_WL is always on, but sometimes because they share the same power with other modules, the voltage collapse issue may occur and lead to the abnormality of WIFI module. As shown in below picture, there are many peripherals connected to VCC_IO.



8.2 软件部分 Software part

首先介绍下 SDIO 波形基本组成：

Firstly introduce the basic component of SDIO waveform:

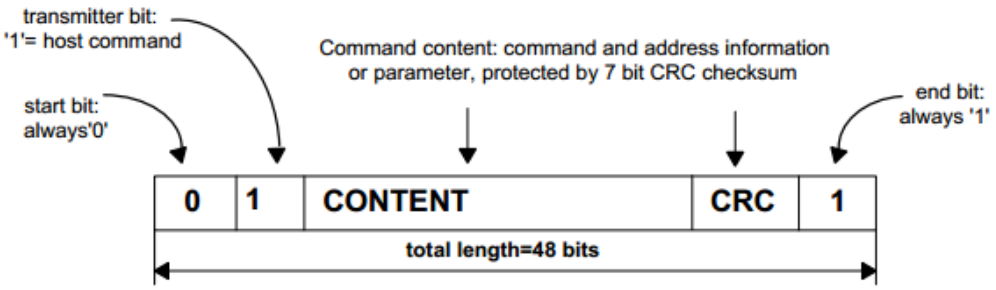


Figure 3-7: Command Token Format

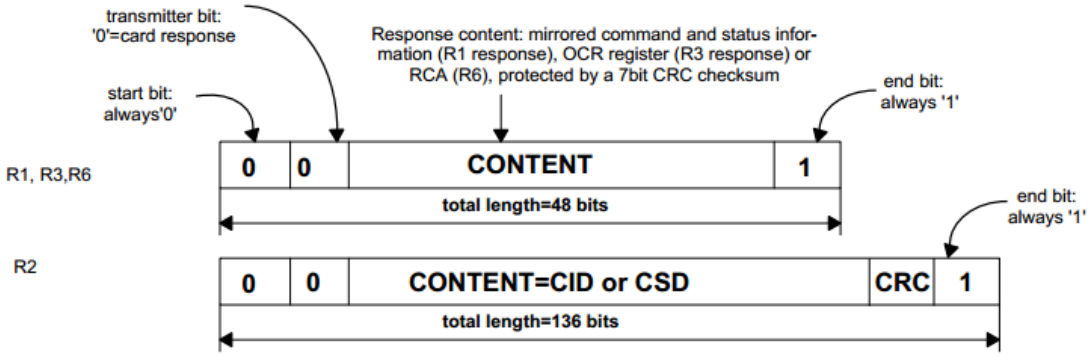
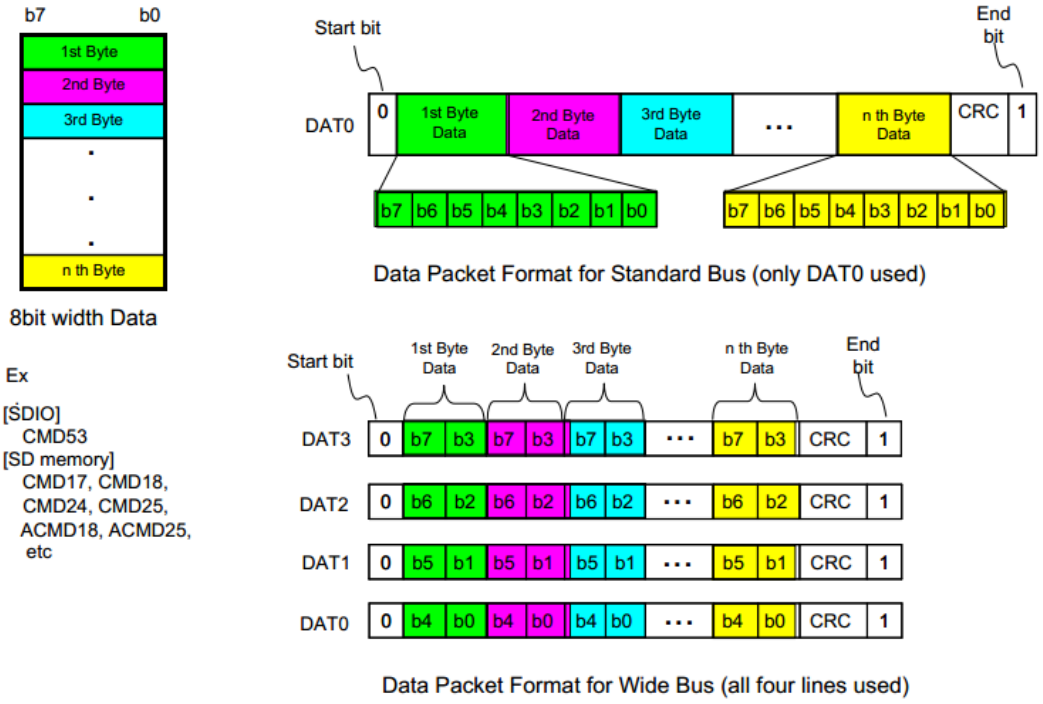


Figure 3-8: Response Token Format

1. Data Packet Format for Usual Data (8-bit width)



下图是 WIFI SDIO 识别模式时的典型的波形时序图：

The following picture shows the typical waveform timing sequence of WIFI SDIO recognition mode:

简单说一下识别 SDIO 的方式：主控发出 48clk 并携带 48bit 的数据发给 SDIO，而 SDIO 要回应给主控 48clk 加 48bit 的数据，如下图：

Briefly introduce the method to recognize SDIO: SoC outputs 48clk and 48bit data to SDIO, and SDIO should respond 48clk and 48bit data to SoC, as below picture:

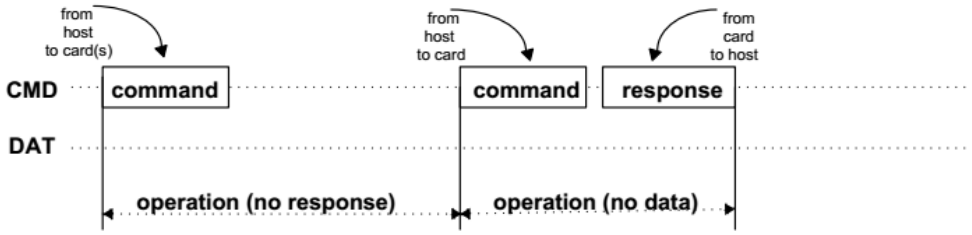


Figure 3-4: "no response" and "no data" Operations

下图是 SDIO 数据传输阶段的时序图：

The following picture shows the timing sequence of SDIO data transmission stage:

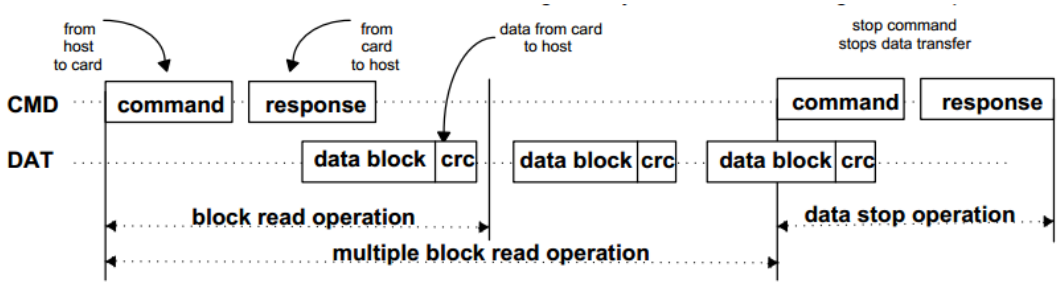


Figure 3-5: (Multiple) Block Read Operation

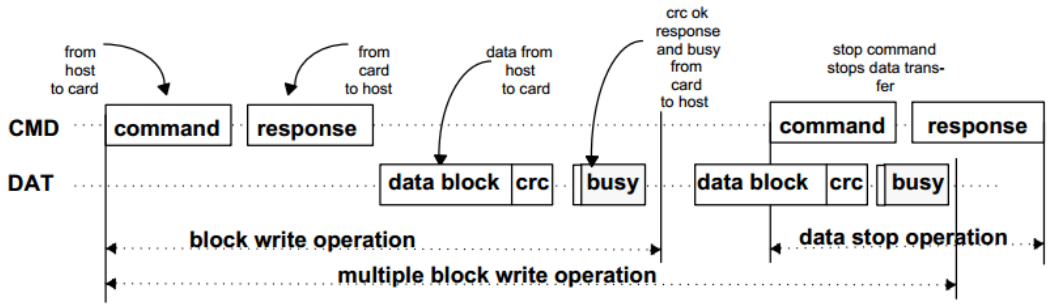


Figure 3-6: (Multiple) Block Write Operation

实例：

For example:

绿色：SDMMC_CLK

Green：SDMMC_CLK

黄色：SDMMC_CMD SDMMC_CMD 空闲时一直处于高电平；

Yellow：SDMMC_CMD keep high level when SDMMC_CMD is idle.

主控发出的波形： 当最开始的两个电平有一高一低时，是主控发出去的命令；

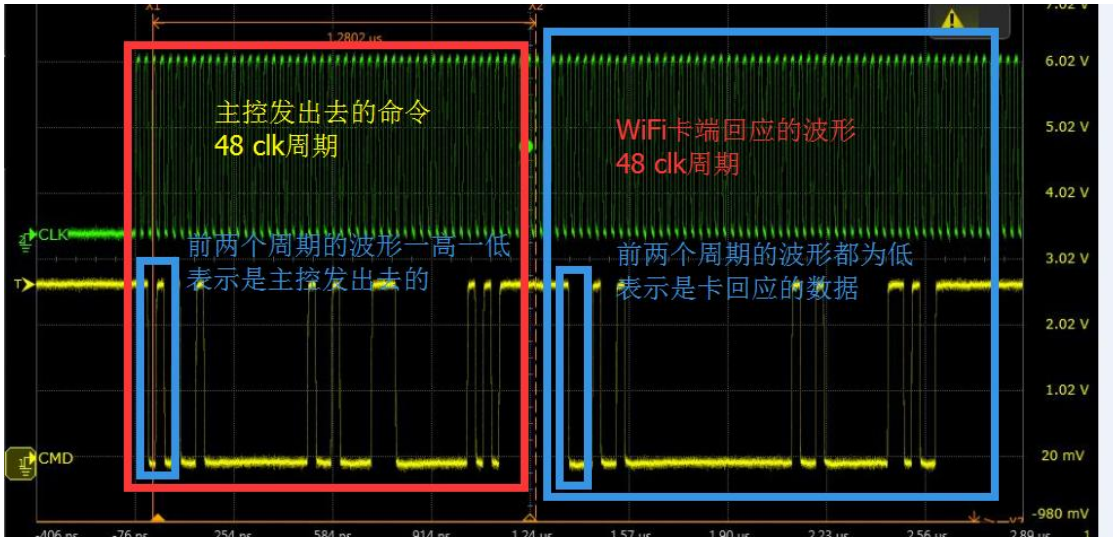
The waveform SoC outputs: when there is one high level and one low level at the beginning, it is the command output from SoC.

SD 卡响应的波形： 当最开始的两个电平有连续的两个低电平时，表示卡端有响应；

The waveform SD responds: when there are two continuous low levels at the beginning, it means there is response from card side.

其次主控和响应一般包含 48 个 bit 的数据，所以 48 个 clk 为一个完整的包。要确认的就是：主控发出去命令包后,SD 卡端是否有响应。

Secondly, SoC and response generally contain 48bit data, so 48 clk is a complete package. **Need to confirm that: if there is response from SD card side after SoC sending out the command package.**



（注意：dts 中的相关节点 sdio 一定要 okay，其次是 io 管脚不要被复用）

(Note: sdio of relative nodes in dts must be okay, and io pin should be not reused)

8.3 错误典型示例 Typical error example

（1） 电压问题，排查 WIFI_DATA 线的电压以及 VCCIO_WL 电压

Voltage problem, analyze the voltage of WIFI_DATA line and VCCIO_WL voltage

[100.086615] slot->flags = 3

[100.086628] CPU: 0 PID: 113 Comm: kworker/u8:3 Tainted: P W O 3.10.0 #250

[100.086644] Workqueue: kmmcd mmc_rescan

[100.086663] [<c0013e04>] (unwind_backtrace+0x0/0xe0) from [<c0011720>] (show_stack+0x10/0x14)

[100.086677] [<c0011720>] (show_stack+0x10/0x14) from [<c050d3b4>] (dw_mci_set_ios+0x9c/0x21c)

[100.086689] [<c050d3b4>] (dw_mci_set_ios+0x9c/0x21c) from [<c04fabd4>] (mmc_power_up+0x60/0xa0)

[100.086700] [<c04fabd4>] (mmc_power_up+0x60/0xa0) from [<c04faeac>] (mmc_rescan_try_freq+0x14/0xd0)

[100.086710] [<c04faeac>] (mmc_rescan_try_freq+0x14/0xd0) from [<c04fb7b4>] (mmc_rescan+0x204/0x298)

[100.086722] [<c04fb7b4>] (mmc_rescan+0x204/0x298) from [<c00471f0>] (process_one_work+0x29c/0x458)

[100.086734] [<c00471f0>] (process_one_work+0x29c/0x458) from [<c0047540>] (worker_thread+0x194/0x2d4)

[100.086745] [<c0047540>] (worker_thread+0x194/0x2d4) from [<c004ca28>] (kthread+0xa0/0xac)

[100.086759] [<c004ca28>] (kthread+0xa0/0xac) from [<c000da98>] (ret_from_fork+0x14/0x3c)

[100.086767] 1400..dw_mci_set_ios: wait for unbusy timeout..... STATUS = 0x206 [mmc2]

[102.546615] 1009..mci_send_cmd: wait for unbusy timeout.....[mmc2]

[102.595819] mmc_host mmc1: Timeout sending command (cmd 0x202000 arg 0x0 status 0x80202000)

（2）有两种可能：

a) **WiFi 异常，排查 VBAT_WL 和 WIFI_REG_ON 以及晶振是否正常**
   Wi-Fi abnormality, analyze if VBAT_WL, WIFI_REG_ON and crystal oscillator are normal or not

b) **走线太长导致波形质量很差，降频或者修改硬件**
   The trace is too long which will cause the poor quality of waveform, reduce the frequency or modify the hardware

dwmmc_rockchip 10218000.rksdmmc: req failed (CMD52): error = 110, timeout = 8000ms

[mmc2] - Timeout recovery procedure start --

rk_sdmmc: BOOT dw_mci_setup_bus: argue clk_mmc workaround out 800000Hz for init[mmc2]

mmc_host mmc2: Bus speed (slot 0) = 37500000Hz (slot req 400000Hz, actual 398936HZ div = 47)

rk_sdmmc: BOOT dw_mci_setup_bus: argue clk_mmc workaround out 800000Hz for init[mmc2]

[mmc2] -- Timeout recovery procedure finished --

（3）走线太长导致波形质量很差，降频或者修改硬件

The trace is too long which will cause the poor quality of waveform, reduce the frequency or modify the hardware

[200.731403] [mmc2] Data transmission error !!!! MINTSTS: [0x00000080]

[200.731426] [mmc2] host was already tuning, Don't need to retry tune again ignore 0.

（4）WIFI_REG_ON 异常

WIFI_REG_ON is abnormal

rk_sdmmc: BOOT dw_mci_setup_bus: argue clk_mmc workaround out 800000Hz for init[mmc2]

rk_sdmmc: BOOT dw_mci_setup_bus: argue clk_mmc workaround out 400000Hz for init[mmc2]

rk_sdmmc: BOOT dw_mci_setup_bus: argue clk_mmc workaround out 600000Hz for init[mmc2