



D1 Tina Linux Wi-Fi 开发指南

版本号: 1.0
发布日期: 2021.04.06

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.04.06	AWA1381	1. 建立初始版本。



目 录

1 前言	1
1.1 文档简介	1
1.2 目标读者	1
1.3 适用范围	1
2 Wi-Fi 简介	2
2.1 Wi-Fi 工作的几种模式	2
2.2 Tina Wi-Fi 软件结构	2
2.3 Wi-Fi 常用命令介绍	3
2.3.1 station 模式	3
2.3.2 ap 模式	3
3 Wi-Fi 模组移植	4
3.1 模组移植的步骤	6
3.2 XR829	7
3.3 RTL8723DS	9
3.4 验证	11
3.5 模组移植总结	11
3.6 Tina 平台已经移植的模组	11
4 Wi-Fi manager 介绍	13
4.1 sdk 代码目录	13
4.2 框架结构	14
4.3 编译配置	14
4.4 Wi-Fi daemon API 说明	15
4.4.1 准备	15
4.4.1.1 头文件与动态库	15
4.4.1.2 示例代码	15
4.4.2 Wi-Fi daemon API	16
4.4.2.1 连接网络	16
4.4.2.2 扫描网络	17
4.4.2.3 列出网络	17
4.4.2.4 移除网络	17
4.4.2.5 获取连接状态	17
4.4.2.6 Wi-Fi daemon 打开	18
4.4.2.7 Wi-Fi daemon 关闭	18
4.5 Wi-Fi API 说明	18
4.5.1 准备	18
4.5.1.1 头文件与动态库	18
4.5.1.2 示例代码	19
4.5.2 Wi-Fi 打开和关闭	20
4.5.2.1 Wi-Fi 打开	20

4.5.2.2	Wi-Fi 状态切换 (回调函数)	20
4.5.2.3	Wi-Fi 操作接口	20
4.5.2.4	Wi-Fi 关闭	21
4.5.3	添加事件回调接口	21
4.5.4	获取 Wi-Fi 信息	21
4.5.5	扫描 AP	22
4.5.6	连接与断开 AP	22
4.5.6.1	connect_ap	22
4.5.6.2	connect_ap_auto	22
4.5.6.3	connect_ap_with_netid	23
4.5.6.4	disconnect_ap	23
4.5.7	获取 IP 地址	23
4.5.8	获取配置信息	23
4.5.9	删除 network 记录	24
4.5.10	打印 log 控制	24
4.5.10.1	设置打印级别	24
4.5.10.2	获取打印级别	25
4.5.10.3	将打印重定向到 syslog 中	25
4.5.10.4	关闭打印信息重定向到 syslog	25
4.5.10.5	打印信息重定向到指定文件中	25
4.5.10.6	关闭打印信息重定向到文件中	26
4.5.11	编程建议	26
4.5.11.1	wifi_on	26
4.5.11.2	事件回调	26
4.5.11.3	wifi off	26
5	Softap 介绍	27
5.1	sdk 代码目录	27
5.2	编译配置	27
5.2.1	内核配置	27
5.2.2	Tina 配置	28
5.3	API 编写说明	29
5.3.1	导入接口文件	29
5.3.2	动态链接库	29
5.3.3	示例代码	29
5.4	Wi-Fi 打开和关闭	29
5.4.1	Wi-Fi 打开	29
5.4.2	Wi-Fi 服务关闭	30
5.5	Softap API 说明	30
5.5.1	SoftAP 初始化和配置	30
5.5.1.1	wifi firmware 切换	30
5.5.1.2	Softap 初始化	30
5.5.1.3	Softap 反初始化	31

5.5.1.4	配置 Softap	31
5.5.1.5	保存配置	32
5.5.2	建立 SoftAP 热点	32
5.5.2.1	启动 Softap	32
5.5.2.2	设置 ip 和子网掩码	32
5.5.2.3	启动 udhcpd 和 dns 服务	32
5.5.2.4	使能数据转发	33
5.5.3	关闭 Softap	33
5.5.3.1	关闭 Softap	33
5.5.4	获取 SoftAP 状态	33
5.5.4.1	获取 SoftAP 状态	33
5.6	使用说明	33
5.6.1	关于 Station 和 SoftAP 共存模式的说明	33
5.6.2	Tina Softap 中 firmware 参数设置	34
5.7	Softap demo	35
5.8	Softap 使用	35
5.8.1	使用流程	35
5.8.2	测试 log	36
6	常见问题	37
6.1	编译问题	37
6.1.1	找不到 wowlan 变量	37
6.1.2	找不到 xxx.ko	37
6.1.3	mmc_xxx undefined	38
6.1.4	缺少依赖库	38
6.2	驱动加载问题	39
6.2.1	XR829 模组 ifconfig 显示: No such device	39
6.2.2	XR829 can't open /etc/wifi/xr_wifi.conf, failed	39
6.2.3	驱动加载问题总结	40
6.2.3.1	配置问题	40
6.2.3.2	供电问题	40
6.2.3.3	sdio 问题	40
6.3	supplicant 服务问题	41
6.3.1	找不到 wpa_supplicant.conf 文件	41
6.4	wifimanager 使用问题	41
6.4.1	联网时出现: network not exist!	41
6.5	上层网络应用服务问题	42
6.5.1	XR829 ping 压力测试: poll time out	42

1 前言

1.1 文档简介

介绍 Allwinner 平台上 Wi-Fi 驱动移植，介绍 Tina Wi-Fi 管理框架，包括 Station，Ap 以及 Wi-Fi 常见问题。

1.2 目标读者

适用 Tina 平台的广大客户和对 Tina Wi-Fi 感兴趣的同事。

1.3 适用范围

Allwinner 软件平台 Tina。

Allwinner 硬件平台 D1。

2 Wi-Fi 简介

2.1 Wi-Fi 工作的几种模式

目前 Tina 平台上的 Wi-Fi 一般可处于 3 种工作模式，分别是 STATION，AP，MONITOR。

- STATION：连接无线网络的终端，大部分无线网卡默认都处于该模式，也是常用的一种模式。
- AP：无线接入点，常称热点，比如路由器功能。
- MONITOR：也称为混杂设备监听模式，所有数据包无过滤传输到主机。

2.2 Tina Wi-Fi 软件结构

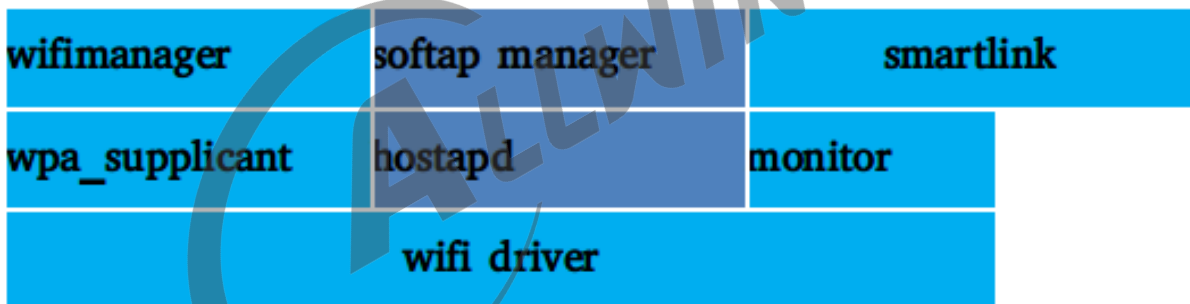


图 2-1: Tina 软件结构图

- wifimanager：主要用于 STATION 模式，提供 Wi-Fi 连接扫描等功能。
- softap manager：提供启动 AP 的功能。
- smartlink：对于 NoInput 的设备，通过借助第三方设备（如手机）实现透传配网的功能，包括 softap/soundwave/xconfig/airkiss/等多种配网方式。
- wpa_supplicant：开源的无线网络配置工具，主要用来支持 WEP，WPA/WPA2 和 WAPI 无线协议和加密认证的，实际上的工作内容是通过 socket 与驱动交互上报数据给用户。
- hostapd：是一个用户态用于 AP 和认证服务器的守护进程。
- monitor：Wi-Fi 处于混杂设备监听模式的处理应用。

2.3 Wi-Fi 常用命令介绍

2.3.1 station 模式

详情配置请看第 4.3 节。

wifi_scan_results_test <level>	扫描周围网络
wifi_connect_ap_test ssid passwd	连接指定网络
wifi_disconnect_ap_test	断开已经连接的网络
wifi_list_networks_test	列出保存的网络
wifi_reconnect_ap_test	重连断开的网络
wifi_get_connection_info_test	获取已连接网络的信息(循环获取)
wifi_connect_ap_with_netid_test <netid> <level>	连接保存的网络,netid是保存网络号
wifi_on_test	打开Wi-Fi测试
wifi_off_test	关闭Wi-Fi测试
wifi_on_off_test	Wi-Fi开关测试
wifi_remove_network_test <ssid> <key_mgmt> <level>	移除保存的指定网络
wifi_remove_all_networks_test <level>	移除所有保存的网络
wifi_longtime_test <ssid> <passwd> <test_times> <level>	保持长连测试
wifi_longtime_scan_test <test_times> <level>	网络扫描测试

注:

ssid 网络名
 passwd 密钥
 netid 保存网络列表中的id号可以用wifi_list_networks_test查看
 level 调试等级d0-d5,所有命令最后都可以加该参数

2.3.2 ap 模式

softap_up ssid passwd 起一个热点

注:

ap模式和station模式在不同模组上不一定能共存,详情看第5节介绍。

3 Wi-Fi 模组移植

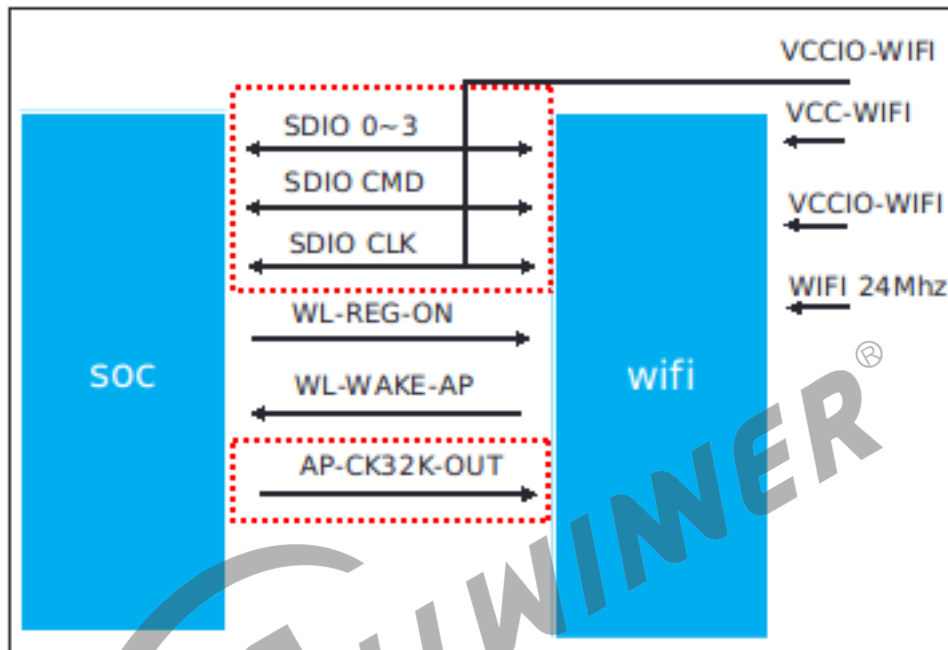


图 3-1: 主控与 Wi-Fi 硬件连接简图

Wi-Fi 模组工作的条件，如上图，需要满足以下几个条件：

- 供电：一般有两路供电，其中 VCC-Wi-Fi 为主电源，VCCIO-Wi-Fi 为 IO 上拉电源。
- 使能：要能正常工作，需要 WL-REG-ON 给高电平。
- SDIO：与 SOC 的通信有通过 USB，SDIO 等，这里以 SDIO 为例，其中 SDIO 0~3 为 SDIO 的 4 条数据线。
- 唤醒主控：当系统休眠时，Wi-Fi 模组可通过 WL-WAKE-AP 通过中断的方式唤醒主控，有些模组也通过该引脚来作为主控接收数据的中断。
- 24/26Mhz 时钟信号。
- 32.768Khz 信号：根据模组而定，有些模组内部通过（5）中的输入的 clk 进行分频得到，有些需要外部单独输入该信号。

对于 Wi-Fi 模组移植，重点围绕以上的几个条件进行开展，对于以上几个工作条件 allwinner 已经提供了对应的 driver，根据总线设备驱动模型，只需要根据各个平台配置 device 即可。allwinner device 除了可以通过 dts 外，还可通过 sys_config.fex 的方式，sys_config.fex 的优先级高于 dts。

说明：

```
- sys_config.fex的路径: tina/device/config/chips/d1/configs/nezha/sys_config.fex
- board.dts的路径: tina/device/config/chips/d1/configs/nezha/linux/board.dts
```

board.dts 配置

sdcl引脚功能配置

```
106     sdcl_pins_a: sdcl@0 {
107         pins = "PG0", "PG1", "PG2",
108             "PG3", "PG4", "PG5";
109         function = "sdcl";
110         drive-strength = <30>;
111         bias-pull-up;
112     };
```

wlan时钟引脚配置

```
140     wlan_pins_a:wlan@0 {
141         pins = "PG11";
142         function = "clk_fanout1";
143     };
```

sdcl属性配置

```
639 &sdcl {
640     bus-width = <4>;
641     no-mmc;
642     no-sd;
643     cap-sd-highspeed;
644     /*sd-uhs-sdr12*/
645     /*sd-uhs-sdr25*/
646     /*sd-uhs-sdr50*/
647     /*sd-uhs-ddr50*/
648     /*sd-uhs-sdr104*/
649     /*sunxi-power-save-mode*/
650     /*sunxi-dis-signal-vol-sw*/
651     cap-sdio-irq;
652     keep-power-in-suspend;
653     ignore-pm-notify;
654     max-frequency = <150000000>;
655     ctl-spec-caps = <0x8>;
656     status = "okay";
657 };
```

rf驱动配置

```
535     rfkill: rfkill@0 {
536         compatible = "allwinner,sunxi-rfkill";
537         chip_en;
538         power_en;
539         status = "okay";
540         /*wlan配置*/
541         wlan: wlan@0 {
542             compatible = "allwinner,sunxi-wlan";
543             pinctrl-0 = <&wlan_pins_a>;
544             pinctrl-names = "default";
545             clock-names = "32k-fanout1";
546             clocks = <&ccu CLK_FANOUT1_OUT>;
547             wlan_busnum = <0x1>;
548             wlan_regon = <&pio PG 12 GPIO_ACTIVE_HIGH>;
549             wlan_hostwake = <&pio PG 10 GPIO_ACTIVE_HIGH>;
```

```
550             /*wlan_power   = "VCC-3V3";*/
551             /*wlan_power_vol = <3300000>;*/
552             /*interrupt-parent = <&pio>;
553             interrupts = < PG 10 IRQ_TYPE_LEVEL_HIGH>;*/
554             wakeup-source;
555
556         };
557         /*bt配置*/
558         bt: bt@0 {
559             compatible = "allwinner,sunxi-bt";
560             pinctrl-0 = <&wlan_pins_a>;
561             pinctrl-names = "default";
562             clock-names = "32k-fanout1";
563             clocks = <&ccu CLK_FANOUT1_OUT>;
564             /*bt_power_num = <0x01>;*/
565             /*bt_power       = "axp803-dldo1";*/
566             /*bt_io_regulator = "axp803-dldo1";*/
567             /*bt_io_vol      = <3300000>;*/
568             /*bt_power_vol   = <3300000>;*/
569             bt_rst_n        = <&pio PG 18 GPIO_ACTIVE_LOW>;
570             status          = "okay";
571         };
572     };
573     /*bt低功耗配置*/
574     btlpm: btlpm@0 {
575         compatible = "allwinner,sunxi-btlpm";
576         uart_index = <0x1>;
577         bt_wake    = <&pio PG 16 GPIO_ACTIVE_HIGH>;
578         bt_hostwake = <&pio PG 17 GPIO_ACTIVE_HIGH>;
579         status     = "okay";
580     };
581     /*mac地址管理驱动配置*/
582     addr_mgt: addr_mgt@0 {
583         compatible = "allwinner,sunxi-addr_mgt";
584         type_addr_wifi = <0x0>;
585         type_addr_bt   = <0x0>;
586         type_addr_eth  = <0x0>;
587         status         = "okay";
588     };
589 };
```

linux driver 路径，详情请参考以下代码路径

```
wlan属性配置解析驱动: tina/lichee/linux-5.4/drivers/misc/sunxi-rf/
mac地址配置解析驱动: tina/lichee/linux-5.4/drivers/misc/sunxi-addr/
xr829 wifi驱动: tina/lichee/linux-5.4/drivers/net/wireless/xr829/
```

3.1 模组移植的步骤

下面总结一款新模组移植到 Tina 平台的步骤。以 XR829 和 RTL8723ds 为例：（当前 D1:SDK 默认支持了 XR829）

模组厂提供过来的 driver，适配到 Tina 平台，主要修改的地方是调用 Tina 平台提供的有上下电，扫卡函数，修改 firmware 的 download 路径，配置 Kconfig 和 Makefile 等。

linux 5.4

```
#include <linux/mmc/host.h>
#include <linux/sunxi-gpio.h>
#include <linux/power/aw_pm.h>

/*
 * 函数功能： 获取所使用的sdio卡号，对应sysconfig.fex中的wlan_busnum
 * 返回值： sdio 卡号
 */
extern int sunxi_wlan_get_bus_index(void);

/*
 * 函数功能： sdio 扫卡
 * 参数 id: 卡号，(sdio 0 or 1 ...)
 * 返回值： 无
 */
extern void sunxi_mmc_rescan_card(unsigned ids);

/*
 * 函数功能： Wi-Fi模组上电，使能。
 * 参数 on: 0, 上电; 1, 掉电。
 * 返回值： 无
 */
extern void sunxi_wlan_set_power(bool on);

/*
 * 函数功能： 获取gpio wlan hostwake pin的申请的中断号
 * 参数 : void
 * 返回值： irq number
 * 说明： 部分模组，主控接收数据通过hostwake pin产生中断来触发，
 * 所以需要主控这边提供获取到中断号。
 */
extern int sunxi_wlan_get_oob_irq(void);

/*
 * 函数功能： 获取host wake pin设置中断的标志位
 * 参数 : void
 * 返回值： irq flag
 */
extern int sunxi_wlan_get_oob_irq_flags(void);
```

3.2 XR829

1. 首先是将 Wi-Fi driver 放到 tina/linux-5.4/drivers/net/wireless，并重命名为 xr829 即 tina/lichee/linux-5.4/drivers/net/wireless/xr829
2. 其次是增加内核的 menuconfig 配置以及编译，只需要修改以下地方即可。

```
tina/lichee/linux-5.4/drivers/net/wireless/Kconfig
example:
+source "drivers/net/wireless/xr829/Kconfig"
```

```
tina/lichee/linux-5.4/drivers/net/wireless/Makefile
example:
+obj-$(CONFIG_XR829_WLAN) += xr829/
```

3. 配置完成后，可执行 `make kernel_menuconfig` 中选中，编译。

```
Device Drivers --->
[*] Network device support --->
  [*] Network device support --->
    [*] Wireless LAN --->
      <M> XR829 WLAN support
```

4. 驱动正常编译过后，添加 `make munconfig` 的配置

该步骤主要将 `kernel` 中编译的 `ko` 文件以及 `firmware` 拷贝到跟文件系统中。

4.1 首先是配置 `firmware`。`firmware` 文件一般以模组文件名存放在如下，并需要新增一个 `mk` 文件，使其在 `make munconfig` 中可见。

```
tina/package/firmware/linux-firmware/xr829
tina/package/firmware/linux-firmware/xr829/xr829.mk
example:
make munconfig
  Firmware --->
    < > xr829-firmware..... Xradio xr829 firmware
```

4.2 其次是配置 `ko`。

`tina/target/allwinner/d1-common/modules.mk`(D1 上默认焊接的是 40M 的晶振)

```
84 define KernelPackage/net-xr829-40M
85   SUBMENU:=$(WIRELESS_MENU)
86   TITLE:=xr829 support (staging)
87   DEPENDS:= +xr829-firmware +@IPV6 +@XR829_USE_40M_SDD +@USES_XRADIO +@PACKAGE_xr829-
  rftest
88   KCONFIG:=\
89     CONFIG_XR829_WLAN=m \
90     CONFIG_PM=y\
91     CONFIG_BT=y \
92     CONFIG_BT_BREDR=y \
93     CONFIG_BT_RFCOMM=y \
94     CONFIG_BT_RFCOMM_TTY=y \
95     CONFIG_BT_DEBUGFS=y \
96     CONFIG_XR_BT_LPM=y \
97     CONFIG_XR_BT_FDI=y \
98     CONFIG_BT_HCIUART=y \
99     CONFIG_BT_HCIUART_H4=y \
100    CONFIG_HFP_OVER_PCM=y \
101    CONFIG_RFKILL=y \
102    CONFIG_RFKILL_PM=y \
103    CONFIG_RFKILL_GPIO=y
104
105   #FILES:=$(LINUX_DIR)/drivers/net/wireless/xr829/wlan/xradio_core.ko
```

```

106 #FILES+=$(LINUX_DIR)/drivers/net/wireless/xr829/wlan/xradio_wlan.ko
107 #FILES+=$(LINUX_DIR)/drivers/net/wireless/xr829/umac/xradio_mac.ko
108 #AUTOLOAD=$(call AutoProbe, xradio_mac xradio_core xradio_wlan)
109
110 FILES=$(LINUX_DIR)/drivers/net/wireless/xr829/xr829.ko
111 AUTOLOAD=$(call AutoProbe, xr829)
112 endif
113
114 define KernelPackage/net-xr829-40M/description
115 Kernel modules for xr829 support
116 endif
117
118 $(eval $(call KernelPackage,net-xr829-40M))

```

```

make mnuconfig
Kernel modules --->
Wireless Drivers --->
  <*> kmod-net-xr829-40M..... xr829 support (staging)

```

5. 配置 sys_config.fex 或者 board.dts

```

rfkill: rfkill@0 {
    compatible = "allwinner,sunxi-rfkill";
    chip_en;
    power_en;
    status = "okay";

    wlan: wlan@0 {
        compatible = "allwinner,sunxi-wlan";
        pinctrl-0 = <&wlan_pins_a>;
        pinctrl-names = "default";
        clock-names = "32k-fanout1";
        clocks = <&ccu CLK_FANOUT1_OUT>;
        wlan_busnum = <0x1>;
        wlan_region = <&pio PE 17 GPIO_ACTIVE_HIGH>;
        wlan_hostwake = <&pio PG 10 GPIO_ACTIVE_HIGH>;
        /*wlan_power = "VCC-3V3";*/
        /*wlan_power_vol = <3300000>;*/
        /*interrupt-parent = <&pio>;
        interrupts = < PG 10 IRQ_TYPE_LEVEL_HIGH>;*/
        wakeup-source;

    };
    ...
}

```

6. 整体编译打包烧写

3.3 RTL8723DS

1. 获取资料

1.1 建议从RTL原厂获取最新版本的完整资料，包括驱动，文档，工具。（也可以从其他内核已适配版本获取驱动）

2. 内核适配

```

2.1将整个驱动SDK拷贝到tina/lichee/linux-xxx/drivers/net/wireless/
2.2驱动重命名为rtl8723ds
2.3在tina/lichee/linux-xxx/drivers/net/wireless/目录修改Kconfig和Makefile
Kconfig:
+source "drivers/net/wireless/rtl8723ds/Kconfig"
Makefile:
+obj-$(CONFIG_RTL8723DS) += rtl8723ds/ (注意: 这里命名一定要匹配)
2.4修改驱动原生代码
2.4.1驱动的Makefile(tina/lichee/linux-xxx/drivers/net/wireless/rtl8723ds/Makefile)
+CONFIG_RTW_ANDROID = 0 (# CONFIG_RTW_ANDROID - 0: no Android, 4/5/6/7/8/9/10 : Android
version)
+CONFIG_PLATFORM_I386_PC = n
+CONFIG_PLATFORM_ARM_SUNXI = y
2.4.2替换适配sunxi的平台文件(tina/lichee/linux-xxx/drivers/net/wireless/rtl8723ds/platform)
可以从已经适配过的其他模组获取: platform_ARM_SUNXI_sdio.c

```

3. Tina module 适配

```

3.1. 从其他任意已经支持的IC方案中拷贝module的配置
define KernelPackage/net-rtl8723ds
    SUBMENU:=$(WIRELESS_MENU) //make menuconfig的菜单位置, 一般不更改。
    TITLE:=RTL8723DS support (staging) //make menuconfig的提示
    DEPENDS:= +r8723ds-firmware +@IPV6 +@USES_REALTEK +@PACKAGE_realtek-rftest +
    @PACKAGE_rtk_hciattach //添加tina依赖, 可以理解为select
    FILES:=$(LINUX_DIR)/drivers/net/wireless/rtl8723ds/8723ds.ko
    KCONFIG:=\ //添加内核依赖可以理解为select
    ...
    AUTOLOAD:=$(call AutoProbe,8723ds)
endef
define KernelPackage/net-rtl8723ds/description //make menuconfig的描述
Kernel modules for RealTek RTL8723DS support
endef
$(eval $(call KernelPackage,net-rtl8723ds))
一个完整的module
注: 建议直接添加在平台的通用配置中: tina/target/allwinner/xxx-common/modules.mk

3.2. firmware的配置
/package/firmware/linux-firmware/rtl8723ds/ //更细驱动时更新firmware文件(如果有最新的)

3.3. sys_config.fex/board.dts的配置
rfkill: rfkill@0 {
    compatible = "allwinner,sunxi-rfkill";
    chip_en;
    power_en;
    status = "okay";

    wlan: wlan@0 {
        compatible = "allwinner,sunxi-wlan";
        pinctrl-0 = <&wlan_pins_a>;
        pinctrl-names = "default";
        clock-names = "32k-fanout1";
        clocks = <&ccu CLK_FANOUT1_OUT>;
        wlan_busnum = <0x1>;
        wlan_regon = <&pio PE 17 GPIO_ACTIVE_HIGH>;
        wlan_hostwake = <&pio PG 10 GPIO_ACTIVE_HIGH>;
        /*wlan_power = "VCC-3V3";*/
        /*wlan_power_vol = <3300000>;*/
    }
}

```

```

        /*interrupt-parent = <&pio>;
        interrupts = < PG 10 IRQ_TYPE_LEVEL_HIGH>;*/
        wakeup-source;

    };
    ...
}

```

4. 整体编译打包烧写

3.4 验证

按照前面的配置好，make kernel_menuconfig 选上对应模块，make menuconfig 选项对应 firmware 和模块，同时，make munconfig 新增选上如下，即可进行验证了。

```

make menuconfig
Allwinner --->
  wifimanager..... Tina wifimanager --->
    <*> wifimanager-demo..... Tina wifimanager app demo

```

ps:这部分的详情说明见后面章节

验证命令：

```

查看模块是否加载: lsmod
模块卸载: rmmmod

连接路由命令: wifi_connect_ap_test ssid passwd
扫描周围热点: wifi_scan_results_test

```

3.5 模组移植总结

主要就是以下几点：

- 修改模组厂提供的 driver，填充相应的上电，扫卡等函数。
- 增加 make kernel_menuconfig 和 make menuconfig 选项，涉及到 firmware,makefile,ko。
- 配置 sys_config.fex 或 board.dts。
- 验证。

3.6 Tina 平台已经移植的模组

Tina 平台上已经移植过多款 Wi-Fi 模组，支持列表如下：


```
BCM      : AP6212,AP6212A,AP6255,AP6256,AP6335等。
Realtek: RTL8723DS (linux 3.4/4.9/5.4),RTL8821cs(linux 4.9/5.4),RTL8822cs(linux 4.9),
          RTL8189FTV(linux4.9)
Xradio  : XR819(linux 3.4/4.4/4.9),xr829(linux 3.4/4.4/4.9/5.4)
Esp      : esp8089 (linux 3.4)
.....
```

对于以上已经移植的模组，用户大多情况只需要在 `kernel_menuconfig` 和 `menuconfig` 选上对应的配置即可。如果想要在 D1 上支持新的模组，请参考 3.1-3.3 章节。



4 Wi-Fi manager 介绍

wifimanager 用于 station, wpa_supplicant 进行通信。实现包括打开/关闭, 连接/断开 AP, 获取连接过程中的状态信息等功能。

4.1 sdk 代码目录

sdk 中 wifimanager 相关代码目录为 package/allwinner/wifimanager。

```
.
├── daemon-demo          #基于wifi daemon API的demo
├── demo                 #基于wifi API的demo
├── files
├── Makefile
├── src
│   ├── core
│   └── daemon
```

wifimanager 提供了两套 API 接口。其中一套需要经过 Wi-Fi daemon(后面简称 Wi-Fi daemon API), 另外一套则不经过 Wi-Fi daemon(后面简称 Wi-Fi API)。

(1) 关于 Wi-Fi daemon API 与 Wi-Fi daemon: Wi-Fi daemon API 和 Wi-Fi daemon(可执行程序) 是基于 Wi-Fi API 再次封装。Wi-Fi daemon 开机自启动, 如果网络配置中存在有效历史信息, 会自动进行联网。当已经连接上的网络异常断开时, 会自动搜索历史配置信息进行切换自动连接。可通过 Wi-Fi daemon API 编写的程序对 Wi-Fi daemon 进行控制 (如添加网络连接, 清除网络, 扫描网络, 获取当前网络状态等)。

(2) 关于 Wi-Fi API 基于该接口的调用, 需要客户自己实现开机自启动进行连接。另外, 说明../wifimanager/demo 编译生成的可执行程序如: wifi_connect_ap_test, wifi_scan_results_test 等只是根据 Wi-Fi API 的接口编译的 demo 供用户参考, 用户不能直接使用该 demo 应用到产品中。(如回连机制是需要客户调用接口自己去实现一套逻辑的。)

4.2 框架结构

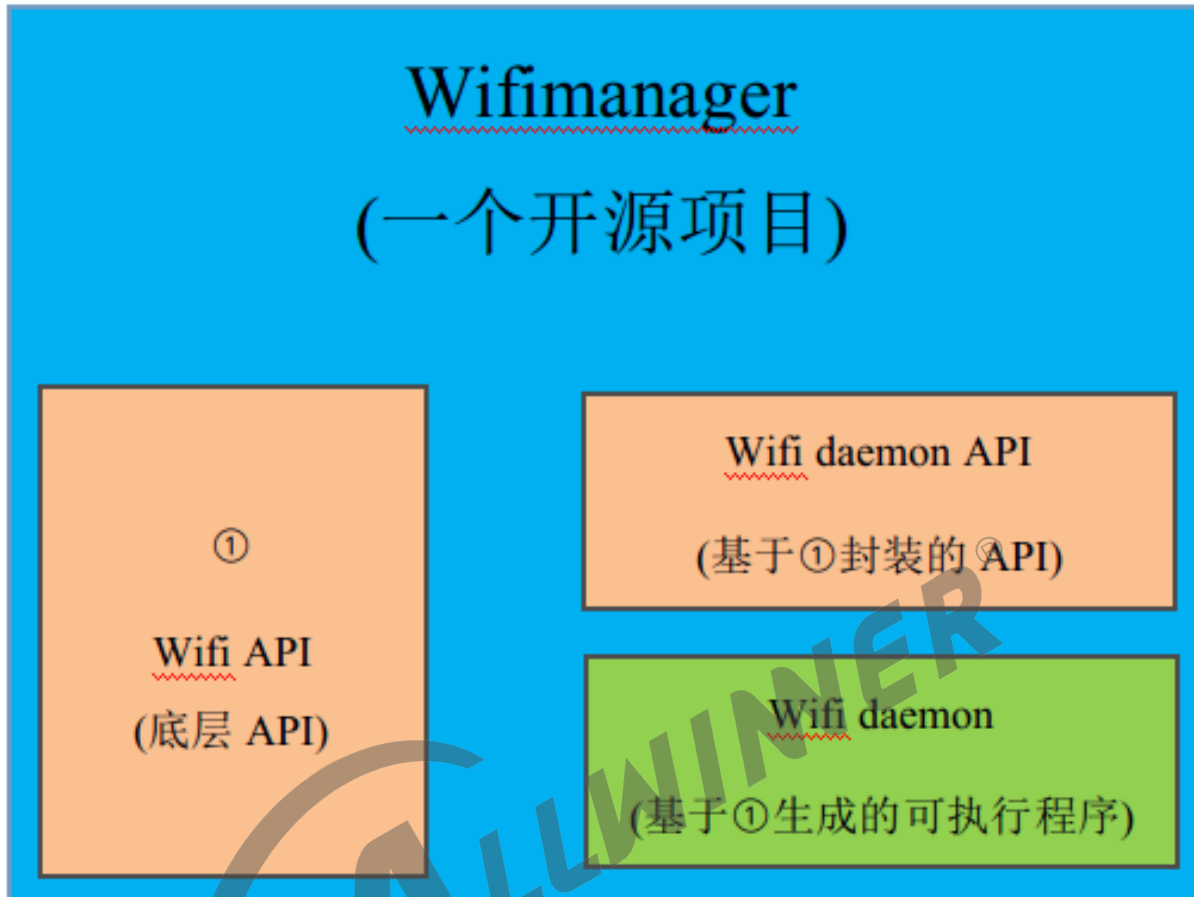


图 4-1: wifimanager 框架结构图

4.3 编译配置

Wi-Fi sdk 相关 menuconfig 配置如下：

```
tina根目录下, 输入make menuconfig
Allwinner --->
  wifimanager --->
    [ ] Enable wifimanager daemon support 见a
    < > wifimanager-daemon-demo..... Tina wifimanager daemon demo 见b
    < > wifimanager-demo..... Tina wifimanager app demo 见c
```

- a: 是否使能 wifimanager daemon
- b: wifimanager daemon API 示例
- c: wifimanager core API 示例
- 一般情况下, 选中了 a 和 b, 就不要选择 c 了. WiFi daemon demo 和 wifimanager demo 这两套是相互独立且排斥的。

4.4 Wi-Fi daemon API 说明

4.4.1 准备

4.4.1.1 头文件与动态库

(1) 导入的头文件

```
#include "wifid_cmd.h"
```

(2) 链接动态库

```
libwifid.so
libwifimg.so
```

具体操作如下

A. make menuconfig 按照如下方式选择。

```
Allwinner --->
  wifimanager --->
    [*] Enable wifimanager daemon support
    <*> wifimanager-daemon-demo..... Tina wifimanager daemon demo 按
    需选择
    < > wifimanager-demo..... Tina wifimanager app demo
```

B. package Makefile 依赖上:DEPENDS +=wifimanager

C. 源码编译的 Makefile 中加入选项-lwifid -lwifimg

4.4.1.2 示例代码

(1) 示例代码

```
tina/package/allwinner/wifimanager/daemon-demo
```

编译出来的可执行程序为 wifid。

参数说明	解释	例子
-h,-help	print this help and exit	wifid -h
-c,-connect	connect AP	wifid -c aw-test 12345678
-s,-scan	scan AP	wifid -s
-l,-list_network	list network	wifid -l
-t,-status	get wifi status	wifid -s

参数说明	解释	例子
-r,-remove_net	remove network in config	wifid -r aw-test
-o,-open	open wifi daemon	wifid -o
-d,-close	close wifi daemon	wifid -d
-debug	set debug log	wifid -debug 5 -c aw-test 12345678

4.4.2 Wi-Fi daemon API

Wi-Fi daemon API 是经过 Wi-Fi_daemon 可执行程序, 通过调用不同的接口来控制 Wi-Fi_daemon。Wi-Fi daemon 应用本质上也是调用的 Wi-Fi API 接口。使用 Wi-Fi daemon API 接口, 用户不需要关心过多网络的状态以及事件, 也不用自行处理有效网络信息中途异常断开, 又重新连接的问题, 这些由 Wi-Fi daemon 已经处理。Wi-Fi daemon API 较 Wi-Fi API 简单, 如用户不想自行处理 Wi-Fi 内部的连接状态, 可使用这套接口。

4.4.2.1 连接网络

```
* aw_wifid_connect_ap
* 【函数原型】: int aw_wifid_connect_ap(const char *ssid, const char *passwd, enum cn_event *
ptrEvent)
* 【功能描述】: 用于连接网络, 如果连接成功, 信息将会被保存到/etc/wifi/wpa_supplicant.conf中。
+ 拓展:
- (1) 开机自启动, Wi-Fi daemon 会自动检查存放在/etc/wifi/wpa_supplicant.conf中的配
置信息, 如果网络信息有效, 将会自动进行连接。
- (2) 已经连接的网络, 如果异常断开, wifi daemon 从/etc/wifi/wpa_supplicant.conf中
自动寻找可用网络进行连接。如果/etc/wifi/wpa_supplicant.conf只存在一个网络信息, 当该
网络断开时(比如拔掉了路由器), wifi daemon 会定时监听, 当该网络再此有效时(再次接上路由器),
会进行自动连接。
* 【参数说明】: 大于等于0: 表示执行成功; 小于0: 表示执行失败。
* 【返回说明】: enum cn_event *ptrEvent: 反馈的事件, 如下
+ DA_CONNECTED,          连接成功
+ DA_PASSWORD_INCORRECT, 密码错误
+ DA_NETWORK_NOT_FOUND,  网络不存在
+ DA_CONNECTED_TIMEOUT,  连接超时
+ DA_AP_ASSOC_REJECT,    路由器拒绝连接
+ DA_OBTAINED_IP_TIMEOUT, 获取ip超时
+ DA_DEV_BUSING,         设备忙碌
+ DA_CMD_OR_PARAMS_ERROR, 参数错误
+ DA_KEYMT_NO_SUPPORT,   加密方式不支持
+ DA_UNKNOWN,
```

4.4.2.2 扫描网络

```
* aw_wifid_get_scan_results
* 【函数原型】：int aw_wifid_get_scan_results(char *results,int len);
* 【功能描述】：用于扫描周围的网络。
* 【参数说明】：
  + char *result: 存放scan结果。
  + int len: len 为result buf长度。
* 【返回说明】：大于等于0:表示执行成功;小于0:表示执行失败。
```

4.4.2.3 列出网络

```
* aw_wifid_list_networks
* 【函数原型】：int aw_wifid_list_networks(char *reply, size_t len);
* 【功能描述】：列出保存在wpa_supplicant配置文件中(/etc/wifi/wpa_supplicant.conf)所有的network信息。
* 【参数说明】：
  + reply: 用来保存结果。
  + reply_len: reply buf的大小。
* 【返回说明】：大于等于0:调用成功;小于0:调用失败;
```

4.4.2.4 移除网络

```
* aw_wifid_remove_networks
* 【函数原型】：int aw_wifid_remove_networks(char *ssid,int len);
* 【功能描述】：删除保存在wpa_supplicant配置文件中(/etc/wifi/wpa_supplicant.conf)指定的network信息。
* 【参数说明】：
  + ssid: 需要删除的AP的ssid;
  + int len: ssid的长度
* 【返回说明】：大于等于0:调用成功;小于0:调用失败;
```

4.4.2.5 获取连接状态

```
* aw_wifid_get_status
* 【函数原型】：int aw_wifid_get_status(struct wifi_status *sptr);
* 【功能描述】：获取网络状态,当连接成功的时候,已连接的网络名称存储在wifi_status 中的ssid中。
* 【参数说明】：
```

```
struct wifi_status *sptr:存储Wi-Fi的状态信息,结构体如下:
struct wifi_status {
  enum wmgState state;      网络状态,参考enum wmgState
  char ssid[SSID_MAX];     如果已经连接,存储其网络名称
};
enum wmgState {
  NETWORK_CONNECTED,      已经连接
```

```

CONNECTING,           正在连接
OBTAINING_IP,        正在获取ip地址
DISCONNECTED,       断开连接
CONNECTED,           连接上ip,但是未分配到ip地址
STATE_UNKDOWN,      未知
};
* 【返回说明】：大于等于0:调用成功；小于0:调用失败；

```

4.4.2.6 Wi-Fi daemon 打开

```

* aw_wifid_open
* 【函数原型】：void aw_wifid_open(void);
* 【功能描述】：用于打开Wi-Fi daemon,正常情况下,Wi-Fi daemon会开机自启动,主要是配合Wi-Fi daemon
  close使用
* 【参数说明】：无
* 【返回说明】：无

```

4.4.2.7 Wi-Fi daemon 关闭

```

* aw_wifid_close
* 【函数原型】：void aw_wifid_close(void);
* 【功能描述】：关闭Wi-Fi(Wi-Fi将断开连接)
* 【参数说明】：无
* 【返回说明】：无

```

4.5 Wi-Fi API 说明

4.5.1 准备

4.5.1.1 头文件与动态库

(1) 导入的头文件

```

#include "wifi_intf.h"
#include "wifi_udhcpc.h"

```

(2) 链接动态库

```

libwifimg.so

```

具体操作如下

- A. make menuconfig 按照如下方式选择。

```

Allwinner --->
  wifimanager --->
    [ ] Enable wifimanager daemon support                不选
    < > wifimanager-daemon-demo..... Tina wifimanager daemon demo 不选
    <*> wifimanager-demo..... Tina wifimanager app demo 按需
    选择
    
```

- B. packge Makefile 依赖上:DEPENDS :=+wifimanager
- C. 源码编译的 Makefile 中加入选项 -lwifimg

4.5.1.2 示例代码

wifimanager app demo 代码目录为：package/allwinner/wifimanager/demo。

示例程序	含义
wifi_connect_ap.cpp	连接AP --重点参考
wifi_scan_results.c	扫描周围网络
wifi_get_netid.c	获取对应SSID的id号
wifi_connect_ap_with_netid.c	指定id号连接
wifi_remove_all_networks.c	移除所有网络配置
wifi_remove_network.c	移除指定网络
wifi_on_off_test.c	打开与关闭测试
wifi_longtime_test.c	多次连接AP测试

本节简要说明 API 接口使用, 如果接口与实际代码有出入, 请以实际代码为准, 具体参照 demo。

Tina 平台 Wi-Fi 包括打开/关闭, 连接/断开 AP, 获取连接过程中的状态信息。Wi-Fi 存在以下几个状态, 调用到相应的接口会激活响应状态的切换。

Wi-Fi状态	含义
CONNECTING	Wi-Fi正在连接状态
CONNECTED	Wi-Fi已经连接AP(还为分配到IP地址)状态
OBTAINING_IP	Wi-Fi正在获取IP地址状态
NETWORK_CONNECTED	Wi-Fi已经获取到IP地址状态
DISCONNECTED	Wi-Fi断开状态
STATE_UNKDOWN	Wi-Fi状态未知

导致 DISCONNATED 的原因, 我们称为事件。

事件	含义
WSE_PASSWORD_INCORRECT	密码不正确
WSE_NETWORK_NOT_EXIST	网络不存在
WSE_AP_ASSOC_REJECT	AP拒绝连接
WSE_WPA_TERMINATING	wpa_supplicant退出
WSE_OBTAINED_IP_TIMEOUT	获取IP超时
WSE_CONNECTED_TIMEOUT	连接AP超时
WSE_DEV_BUSING	设备忙碌
WSE_CMD_OR_PARAMS_ERROR	传入参数不正确
WSE_KEYMT_NO_SUPPORT	加密方式不支持

WSE_ACTIVE_DISCONNECT	激活断开
WSE_AUTO_DISCONNECTED	异常自动断开

4.5.2 Wi-Fi 打开和关闭

4.5.2.1 Wi-Fi 打开

```

* aw_wifi_on
* 【函数原型】：const aw_wifi_interface_t *aw_wifi_on(tWifi_event_callback pcb, int
  event_label)
* 【功能描述】：打开Wi-Fi, 并获取操作wifi interface的句柄。
* 【参数说明】：
  + tWifi_event_callback pcb: Wi-Fi状态切换回调函数地址。
  + int event_label: 事件标签, tWifi_event_callback回调时返回, 用来标明是Wi-Fi on的回调事件。
* 【返回说明】：
  + 非NULL: 指向aw_wifi_interface_t结构指针, 是操作wifi interface的句柄。详见3.1.3 Wi-Fi操作接口
  + NULL: 失败。

```

4.5.2.2 Wi-Fi 状态切换 (回调函数)

```

* tWifi_event_callback
* 【函数原型】：typedef void (*tWifi_event_callback) (struct Manager *wmg,void *buf, int
  event_label);
* 【功能描述】：当Wi-Fi状态切换的时候, 会回调这个函数。
* 【参数说明】：
  + struct Manager *wmg: Wi-Fi的状态, 反馈事件, Wi-Fi是否使能结构体。
  + int event_label: 事件标签, 用来标明是哪次调用的回调事件。
* 【返回说明】：无。

```

4.5.2.3 Wi-Fi 操作接口

```

* aw_wifi_interface_t
* 【函数原型】：
typedef struct{
  int (*add_event_callback)(tWifi_event_callback pcb);
  int (*is_ap_connected)(char *ssid, int *len);
  int (*get_scan_results)(char *result, int *len);
  int (*connect_ap)(const char *ssid, const char *passwd, int event_label);
  int (*connect_ap_key_mgmt)(const char *ssid, tKEY_MGMT key_mgmt, const char *passwd, int
    event_label);
  int (*connect_ap_auto)(int event_label);
  int (*add_network)(const char *ssid, tKEY_MGMT key_mgmt, const char *passwd, int
    event_label);
  int (*disconnect_ap)(int event_label);
  int (*remove_all_networks)(void);
}aw_wifi_interface_t;
* 【功能描述】：Wi-Fi操作接口结构体指针, 需先调用aw_wifi_on获取操作wifi interface的句柄, 才可借助于句柄

```

调用对应函数，各函数功能见下文。

- * 【参数说明】：无。
- * 【返回说明】：无。

4.5.2.4 Wi-Fi 关闭

- * aw_wifi_off
- * 【函数原型】：int aw_wifi_off(const aw_wifi_interface_t *p_wifi_interface_t)
- * 【功能描述】：关闭Wi-Fi。
- * 【参数说明】：const aw_wifi_interface_t *p_wifi_interface_t: 打开Wi-Fi时获得的操作句柄。
- * 【返回说明】：int 0:成功; 非0:失败。

4.5.3 添加事件回调接口

- * add_event_callback
- * 【函数原型】：int (*add_event_callback)(tWifi_event_callback pcb);
- * 【功能描述】：添加Wi-Fi事件回调函数。用户不可直接调用，需借助于aw_wifi_on返回的Wi-Fi操作句柄。
- * 【参数说明】：tWifi_event_callback pcb。
- * 【返回说明】：int 0:成功; 非0:失败。

4.5.4 获取 Wi-Fi 信息

- * aw_wifi_get_wifi_state
 - * 【函数原型】：enum wmgState aw_wifi_get_wifi_state();
 - * 【功能描述】：获取Wi-Fi此刻的状态
 - * 【参数说明】：无
 - * 【返回说明】：返回Wi-Fi的状态,见第三小节。
-

- * aw_wifi_get_wifi_event
 - * 【函数原型】：enum wmgState aw_wifi_get_wifi_event();
 - * 【功能描述】：获取Wi-Fi此刻的事件
 - * 【参数说明】：无
 - * 【返回说明】：返回Wi-Fi的事件,见第三小节。
-

- * is_ap_connected
 - * 【函数原型】：int (*is_ap_connected)(char *ssid, int *len);
 - * 【功能描述】：判断当前是否连接网络，并获取当前连接网络的ssid信息与其对应协议（IPv4/IPv6）。用户不可直接调用，需借助于aw_wifi_on返回的Wi-Fi操作句柄。
 - * 【参数说明】：
 - + char *ssid: 存放当前连接AP的ssid
 - + int *len: len调用前为ssid长度，调用后为当前连接AP ssid长度。
 - * 【返回说明】：大于等0:表示执行成功;小于0:表示执行失败。
 - + int -1: 当前Wi-Fi状态为WIFIMG_WIFI_DISABLED, 即Wi-Fi不可用
 - + int 0: 当前未连接AP

- + int 1: 当前连接AP为IPv4网络
- + int 2: 当前连接AP为IPv6网络

4.5.5 扫描 AP

```
* get_scan_results
* 【函数原型】：int (*get_scan_results)(char *result, int *len);
* 【功能描述】：返回scan结果。用户不可直接调用，需借助于aw_wifi_on返回的Wi-Fi操作句柄。
* 【参数说明】：
  + char *result: 存放scan结果。
  + int *len: len调用前为result buf长度，调用后为scan result长度。
* 【返回说明】：int 0:调用成功;非0:调用失败。
```

4.5.6 连接与断开 AP

4.5.6.1 connect_ap

```
* connect_ap
* 【函数原型】：int (*connect_ap)(const char *ssid, const char *passwd, int event_label);
* 【功能描述】：连接AP
* 【参数说明】：
  + ssid : 连接指定AP的ssid。
  + passwd: 连接指定AP的密码，当AP无密码时，为NULL。
  + event_label: 事件标签，tWifi_event_callback回调时返回，用来标明是connect_ap的回调事件。
* 【返回说明】：大于等于0:表示执行成功;小于0:表示执行失败。
```

4.5.6.2 connect_ap_auto

```
* aw_wifid_get_scan_results
* 【函数原型】：int (*connect_ap_auto)(int event_label);
* 【功能描述】：自动重连wpa_supplicant已保存的ap。用户不可直接调用，需借助于aw_wifi_on返回的Wi-Fi操作句柄。
* 【参数说明】：
  + event_label: 事件标签，tWifi_event_callback回调时返回，用来标明是connect_ap_auto的回调事件。
* 【返回说明】：大于等于0:表示执行成功;小于0:表示执行失败。
```

4.5.6.3 connect_ap_with_netid

```
* connect_ap_with_netid
* 【函数原型】：int (*connect_ap_with_netid)(const char *net_id, int event_label);
* 【功能描述】：使用netid连接wpa_supplicant已保存的ap。用户不可直接调用，需借助于aw_wifi_on返回的Wi-Fi操作句柄。
* 【参数说明】：
+ net_id: 需要连接的AP的ID，可以通过list_networks查看，并通过get_netid获得。
+ event_label: 事件标签，tWifi_event_callback回调时返回，用来标明是connect_ap_with_netid的回调事件。
* 【返回说明】：大于等于0:表示执行成功;小于0:表示执行失败。
```

4.5.6.4 disconnect_ap

```
* disconnect_ap
* 【函数原型】：int (*disconnect_ap)(int event_label);
* 【功能描述】：断开与当前ap的连接。用户不可直接调用，需借助于aw_wifi_on返回的Wi-Fi操作句柄。
* 【参数说明】：
+ event_label: 事件标签，tWifi_event_callback回调时返回，用来标明是disconnect_ap的回调事件。
+ 断开成功，会发送断开连接消息(WIFIMG_NETWORK_DISCONNECTED)。
* 【返回说明】：大于等于0:表示执行成功;小于0:表示执行失败。
```

4.5.7 获取 IP 地址

```
* start_udhcpd
* 【函数原型】：void start_udhcpd();
* 【功能描述】：启动udhcpd获取ip地址，示例代码中，在状态切换为CONNECTED的时候调用，获取ip地址后，将状态切换为NETWORK_CONNECTED。用户可自定义该函数。
* 【参数说明】：无。
* 【返回说明】：无。
```

4.5.8 获取配置信息

```
* list_networks
* 【函数原型】：int (*list_networks)(char *reply, size_t reply_len, int event_label);
* 【功能描述】：列出保存在wpa_supplicant配置文件中所有的network信息。用户不可直接调用，需借助于aw_wifi_on返回的Wi-Fi操作句柄。
* 【参数说明】：
+ reply: 用来保存结果;
+ reply_len: 调用前reply的大小;
+ event_label: 事件标签，tWifi_event_callback回调时返回，用来标明是list_networks的回调事件。
* 【返回说明】：大于等于0:表示执行成功;小于0:表示执行失败。
```



```
* get_netid
* 【函数原型】：int (*get_netid)(const char *ssid, tKEY_MGMT key_mgmt, char *net_id, int *
```

```
length);
```

- * 【功能描述】：获取保存在wpa_supplicant配置文件中指定的network的netid。用户不可直接调用，需借助于aw_wifi_on返回的Wi-Fi操作句柄。
- * 【参数说明】：
 - + ssid: AP的ssid。
 - + key_mgmt: AP的加密方式。
 - + net_id: 用于存放指定AP的netid。
 - + length: 值—结果参数。通过此参数传入保存netid数组的大小；返回获取到的netid的长度。
- * 【返回说明】：大于等于0:表示执行成功;小于0:表示执行失败。

4.5.9 删除 network 记录

```
* remove_network
* 【函数原型】：int (*remove_network)(char *ssid, tKEY_MGMT key_mgmt);
* 【功能描述】：删除保存在wpa_supplicant配置文件中指定的network信息。用户不可直接调用，需借助于aw_wifi_on返回的Wi-Fi操作句柄。
* 【参数说明】：
  + ssid: 需要删除的AP的ssid。
  + Key_mgmt: 需要删除的AP的加密方式。
* 【返回说明】：大于等于0:表示执行成功;小于0:表示执行失败。

<br/>

* remove_all_networks
* 【函数原型】：int (*remove_all_networks)();
* 【功能描述】：删除wpa_supplicant配置文件所有保存的network信息，即重置配置文件。用户不可直接调用，需借助于aw_wifi_on返回的Wi-Fi操作句柄。
* 【参数说明】：无。
* 【返回说明】：大于等于0:表示执行成功;小于0:表示执行失败。
```

4.5.10 打印 log 控制

头文件

```
#include "wmg_debug.h"
```

4.5.10.1 设置打印级别

```
* wmg_set_debug_level
* 【函数原型】：void wmg_set_debug_level(int level);
* 【功能描述】：设置打印级别。
* 【参数说明】：无
```

```
int level:打印级别,取值如下:
enum {
  MSG_ERROR=0,
  MSG_WARNING,
  MSG_INFO,
```

```
MSG_DEBUG,  
MSG_MSGDUMP,  
MSG_EXCESSIVE  
};  
* 【返回说明】：无
```

4.5.10.2 获取打印级别

```
* wmg_get_debug_level  
* 【函数原型】：int wmg_get_debug_level();  
* 【功能描述】：获取打印级别。  
* 【参数说明】：无  
* 【返回说明】：(0~5)见4.5.1
```

4.5.10.3 将打印重定向到 syslog 中

```
* wmg_debug_open_syslog  
* 【函数原型】：void wmg_debug_open_syslog(void);  
* 【功能描述】：关闭打印信息重定向到syslog  
* 【参数说明】：无  
* 【返回说明】：无
```

4.5.10.4 关闭打印信息重定向到 syslog

```
* wmg_debug_close_syslog  
* 【函数原型】：oid wmg_debug_close_syslog(void);  
* 【功能描述】：关闭打印信息重定向到syslog  
* 【参数说明】：无  
* 【返回说明】：无
```

4.5.10.5 打印信息重定向到指定文件中

```
* wmg_debug_open_file  
* 【函数原型】：int wmg_debug_open_file(const char *path);  
* 【功能描述】：打印信息重定向到指定文件中，释放时需要调用wmg_debug_close_file函数  
* 【参数说明】：const char *path:文件路径,需保证改路径文件系统可读写。  
* 【返回说明】：大于等0:表示执行成功;小于0:表示执行失败。
```

4.5.10.6 关闭打印信息重定向到文件中

```
* wmg_debug_close_file
* 【函数原型】：void wmg_debug_close_file(void);
* 【功能描述】：关闭打印信息重定向到文件中。
* 【参数说明】：无
* 【返回说明】：无
```

4.5.11 编程建议

4.5.11.1 wifi_on

在一个进程中，aw_wifi_on 只能调用一次。wifi_on 打开 wifi，返回 Wi-Fi 操作句柄 aw_wifi_interface_t。该进程第二次及以后打开 Wi-Fi，返回 NULL，表示失败。假设在一个进程的主线程 A 中打开了 Wi-Fi，获得了 aw_wifi_interface 句柄，如果同进程的其它线程 B 想操作 Wi-Fi，由主线程 A 传递句柄给该线程 B。

4.5.11.2 事件回调

如果主线程 A 想监听包括 wifi on 时的所有事件，必须在 wifi on 时将回调函数传入。如果主线程 A 只想监听 wifi on 之后的事件，可以在 wifi on 之后调用 add_event_callback 接口添加事件回调函数。如果线程 B 想监听 Wi-Fi 的事件，同理，调用 add_event_callback 接口添加事件回调函数。

4.5.11.3 wifi_off

wifi_off 只能调用一次。第二次及以后调用直接返回。wifi_off 关闭 Wi-Fi，完全关闭前一次调用 aw_wifi_on 以及后续调用的其他 Wi-Fi 接口的影响，关闭后所有 Wi-Fi 停止工作，不能收到任何事件了。

5 Softap 介绍

softap 部分代码为 Tina 平台管理 wifi softap 模式的模块。主要功能包括打开/配置/启动/关闭 softap, 获取 softap 的状态等。

5.1 sdk 代码目录

sdk 中 softap 相关代码目录为 tina/package/allwinner/softap。包括源码和 demo 程序。

5.2 编译配置

5.2.1 内核配置

Tina 根目录下, 输入:

```
make kernel_menuconfig
```

选择如下 softap 所需要的内核组件:

```
[*] Networking support --->
    Networking options --->
        [*] Network packet filtering framework (Netfilter) --->
            [*] Advanced netfilter configuration
                Core Netfilter Configuration --->
                    <*> Netfilter connection tracking support
                        [*] Connection mark tracking support
                        [*] Connection tracking security mark support
                        [*] Connection tracking events
                        [*] Connection tracking timeout
                        [*] Connection tracking timestamping
                        <M> Connection tracking netlink interface
                        <M> Connection tracking timeout tuning via Netlink
                    -* Netfilter Xtables support (required for ip_tables)
                        <*> "conntrack" connection tracking match support
                        <*> "state" match support
                IP: Netfilter Configuration --->
                    <*> IPv4 connection tracking support (required for NAT)
                    <*> IP tables support (required for filtering/masq/NAT)
                    <*> Packet filtering
                    <*> IPv4 NAT
                        <*> MASQUERADE target support
                        <*> NETMAP target support
```



```
<*> REDIRECT target support
<*> Packet mangling
```

5.2.2 Tina 配置

Softap 需要用到的应用包括：hostapd、iptables、dnsmasq，在编译之前需要配置并选中这些功能。在 Tina 根目录下，输入：

```
make menuconfig
```

配置选中 softap

```
Allwinner --->
  <*> softap..... Tina softap manager --->
    <*> softap-demo..... Tina softap app demo
      wifi module (xr819) --->
```

配置 softap 软件包时需注意以下两点内容：

- 1. 如果要参考 softap app demo 代码，需要先选择 softap，再选择 softap-demo 包，表示 softap app demo 程序。
- 2. 选择 softap 软件包时，为了适配不同的平台，应根据平台模组型号选择对应的 Wi-Fi 驱动模块，如上示例选中的是 xr819 模块。

配置选中 hostapd

```
Network --->
  <*> hostapd..... IEEE 802.1x Authenticator (full)
  *- hostapd-common..... hostapd/wpa_supplicant common support files
```

配置选中 iptables

```
Network --->
  Firewall --->
    <*> iptables..... IP firewall administration tool
  --->
```

配置选中 dnsmasq

```
Base system --->
  <*> dnsmasq..... DNS and DHCP server
```

另外，使用 softap 功能前需要先装载平台 Wi-Fi 驱动。若想在开机启动之后即可使用 softap 应用，需实现开机自动装载 Wi-Fi 驱动（建议）；若无上述需求，也可在开机之后手动装载 Wi-Fi 驱动。Wi-Fi 驱动自加载相关配置如下所示：

- 1. 对于 busybox init 的情况，需配置 busybox-init-base-file 内核模块自加载选项，详情参考《Tina System init 使用说明文档.pdf》
- 2. Wi-Fi 驱动自加载是借助 kmodloader，由 ubox 软件包提供，因此 menuconfig 配置时需选中该软件包：

```
Base system --->
<*> ubox..... OpenWrt system helper toolbox
```

5.3 API 编写说明

5.3.1 导入接口文件

```
#include <aw_softap_intf.h>
```

5.3.2 动态链接库

```
libsoftap.so
```

5.3.3 示例代码

softap app demo 代码目录为：package/allwinner/softap/demo。

5.4 Wi-Fi 打开和关闭

5.4.1 Wi-Fi 打开

Wi-Fi 打开主要完成如下工作：

- 1. 启动 wpa_supplicant 服务 (如果没有启动)；
- 2. 连接 wpa_supplicant (Wi-Fi driver 由系统启动时完成加载，wpa_supplicant 服务可以在系统启动过程中启动)

5.4.2 Wi-Fi 服务关闭

Wi-Fi 关闭主要完成如下工作：

- 1. 断开与 wpa_supplicant 的连接
- 2. kill 掉 wpa_supplicant 服务
- 3. disable wlan0 网口，Wi-Fi 不再可用。

5.5 Softap API 说明

Tina 平台 softap 包括初始化 softAP，配置 softAP，打开/关闭 softAP，获取 softAP 的状态信息等。

5.5.1 SoftAP 初始化和配置

5.5.1.1 wifi firmware 切换

```
* aw_softap_reload_firmware
* 【函数原型】：int aw_softap_reload_firmware(char *ap_sta);
* 【功能描述】：切换Wi-Fi模式对应的firmware。
* 【参数说明】：ap_sta: 需要切换的firmware对应的Wi-Fi模式。可以输入的参数有“STA”、“AP”、“P2P”，
  代表目前支持Wi-Fi的三种模式：
  + a、“STA”——station模式；
  + b、“AP”——softAP模式；
  + c、“P2P”——P2P模式；
* 【返回说明】：0：切换指定firmware成功；非0：切换失败。
* 【注意事项】：
  + 需要注意的是：切换firmware并不会立即生效让Wi-Fi处于对应的模式，而是在调用aw_softap_enable()时生效。
  + 目前Tina SDK支持的Wi-Fi模组中，broadcom AP6212/AP6212A等系列模组需要切换firmware；
  而其他厂家（例如realtek）的模组则无需/无法切换firmware。具体情况请咨询使用的模组原厂技术支持人员或代理商。
  + firmware的参数设置详见4.2部分。
```

5.5.1.2 Softap 初始化

```
* aw_softap_init
* 【函数原型】：int aw_softap_init();
* 【功能描述】：初始化softap内部数据结构和默认基本配置。默认的基本配置如下：
  + a、使用wlan0接口启动softap；
  + b、ssid为Smart-AW-HOSTAPD；
  + c、psk为wifil1111；
  + d、AP可见（broadcast）；
```

- + e、使用通道6;
- + f、加密方式为wpa2-psk。
- * 【参数说明】：无
- * 【返回说明】：0：初始化成功；非0：初始化失败。
- * 【注意事项】：
 - + softap支持的加密方式只有三种：SOFTAP_NONE、SOFTAP_WPA_PSK、SOFTAP_WPA2_PSK。
 - + 配置的通道不一定生效。例如，对于broadcom的模组，如果是同一模组同时开启了station和softap模式，则softap使用的通道随station变动而变动。
 - + 关于以上两点更多说明见aw_softap_config() API的介绍。

5.5.1.3 Softap 反初始化

- * aw_softap_deinit
- * 【函数原型】：int aw_softap_deinit();
- * 【功能描述】：释放softap内部数据结构。
- * 【参数说明】：无
- * 【返回说明】：0：初始化成功；非0：初始化失败。
- * 【注意事项】：退出softap相关进程时，必须要调用此接口释放内部数据结构。根据使用场景的不同，分为两种场景：
 - + 启动softAP以后不关闭softAP。退出进程前调用此接口释放softap内部数据结构，并不会影响已经保存或生效的softAP相关配置。
 - + 启动softAP以后关闭softAP。执行完softap关闭的所有动作后，退出进程前调用此接口释放softap内部数据结构。

5.5.1.4 配置 Softap

- * aw_softap_config
- * 【函数原型】：int aw_softap_config(char *ssid, char *psk, tSOFTAP_KEY_MGMT key_mgmt, char *interface, char *channel, char *broadcast_hidden);
- * 【功能描述】：配置softAP的各项参数。
- * 【参数说明】：
 - ssid：设置softAP的ssid。
 - psk：设置softAP的psk。 (详见注意事项1)
 - key_mgmt：设置softAP的加密方式。softap支持的加密方式只有三种：SOFTAP_NONE（无密）、SOFTAP_WPA_PSK、SOFTAP_WPA2_PSK。 (详见注意事项1)
 - interface：设置softAP使用的网络接口。
 - channel：设置softAP使用的通信信道。 (详见注意事项2)
 - broadcast_hidden：设置softAP是否隐藏。不隐藏：参数为字符串broadcast；隐藏：参数为字符串hidden。
- * 【返回说明】：0：初始化成功；非0：初始化失败。
- * 【注意事项】：退出softap相关进程时，必须要调用此接口释放内部数据结构。根据使用场景的不同，分为两种场景：
 1. 如果设置加密方式为SOFTAP_NONE，即使psk带入的参数不为空字符串，则仍认为该AP设置为开放（无密）的AP。
 2. 配置的通道不一定生效。例如，对于broadcom的模组，如果是同一模组同时开启了station和softap模式，则softap使用的通道随station变动而变动。

5.5.1.5 保存配置

```
* aw_softap_save_config
* 【函数原型】：int aw_softap_enable();
* 【功能描述】：应用保存到配置文件的配置，启动softAP。
* 【参数说明】：无
* 【返回说明】：0：初始化成功；非0：初始化失败。
* 【注意事项】：如果是使用broadcom的Wi-Fi模组，设置interface为wlan1，开启station和softAP兼容模式（wlan0作为staion，wlan1作为softAP），需要在aw_softap_intf.h中将#define IW_UP_BROADCOM_WLAN1 0修改为#define IW_UP_BROADCOM_WLAN1 1。
```

5.5.2 建立 SoftAP 热点

5.5.2.1 启动 Softap

```
* aw_softap_deinit
* 【函数原型】：int aw_softap_deinit();
* 【功能描述】：释放softap内部数据结构。
* 【参数说明】：无
* 【返回说明】：0：初始化成功；非0：初始化失败。
* 【注意事项】：退出softap相关进程时，必须要调用此接口释放内部数据结构。根据使用场景的不同，分为两种场景：
+ 启动softAP以后不关闭softAP。退出进程前调用此接口释放softap内部数据结构，并不会影响已经保存或生效的softAP相关配置。
+ 启动softAP以后关闭softAP。执行完softap关闭的所有动作后，退出进程前调用此接口释放softap内部数据结构。
```

5.5.2.2 设置 ip 和子网掩码

```
* aw_softap_router_config
* 【函数原型】：int aw_softap_router_config(char *ip, char *netmask)
* 【功能描述】：设置建立的softAP的IP和子网掩码。
* 【参数说明】：
+ ip: IP地址。
+ netmask: 子网掩码。
* 【返回说明】：0：初始化成功；非0：初始化失败。
```

5.5.2.3 启动 udhcpd 和 dns 服务

```
* aw_softap_start_udhcp_dns_server
* 【函数原型】：int aw_softap_start_udhcp_dns_server();
* 【功能描述】：启动udhcpd和dns中转、缓存服务。
* 【参数说明】：无
* 【返回说明】：0：初始化成功；非0：初始化失败。
* 【注意事项】：只有启动了udhcpd和dns服务，其他设备才能正常连接至该softAP并自动获取IP地址。
```

5.5.2.4 使能数据转发

```
* aw_softap_enable_data_forward
* 【函数原型】：int aw_softap_enable_data_forward(char *interface);
* 【功能描述】：使能数据转发。如果设备带有以太网卡等网络接口，想实现外部网络接入和数据互通，可以使用此调用开启数据转发。
* 【参数说明】：interface：需要转发数据的网络接口。（详见注意事项1）
* 【返回说明】：0：初始化成功；非0：初始化失败。
* 【注意事项】：1. 常见的网络接口如以太网eth0。
```

5.5.3 关闭 Softap

5.5.3.1 关闭 Softap

```
* aw_softap_disable
* 【函数原型】：int aw_softap_disable();
* 【功能描述】：关闭softAP。
* 【参数说明】：无
* 【返回说明】：0：初始化成功；非0：初始化失败。
* 【注意事项】：如果是使用broadcom的Wi-Fi模组，设置interface为wlan1，开启station和softAP兼容模式（wlan0作为staion，wlan1作为softAP），需要在aw_softap_intf.h中将
#define IW_UP_BROADCOM_WLAN1 0修改为#define IW_UP_BROADCOM_WLAN1 1。
```

5.5.4 获取 SoftAP 状态

5.5.4.1 获取 SoftAP 状态

```
* aw_is_softap_started
* 【函数原型】：int aw_is_softap_started();
* 【功能描述】：用于获取softAP状态，从返回值获知softAP是否建立。
* 【参数说明】：无
* 【返回说明】：0：未建立；1：已建立。
* 【注意事项】：成功调用aw_softap_enable()后，调用此接口即返回1。
```

5.6 使用说明

5.6.1 关于 Station 和 SoftAP 共存模式的说明

如果使用的模组（如 realtek 某些型号的模组）驱动能同时生成两个虚拟接口，两个接口相互独立，可以使用一个接口作为 station(如 wlan0)，使用另一个接口作为 softAP。具体详情请咨询

模组原厂或代理商相关技术人员。

如使用的为 broadcom 支持 station 和 AP 共存的模组，需要在 wlan0 启动的基础上添加 wlan1 接口。原则上，常见的所有 broadcom 模组均支持 station 和 AP 模式共存，但需要使用 broadcom 原生工具 dhd_priv 添加新的 interface 并建立 softAP。

但目前只有部分模组支持使用 hostapd 在 wlan1 建立 softAP。这些 broadcom 模组有 AP6255、AP6356S 等，且驱动需升级为 1.363.59.144.10 以上版本。wlan1 的启动必须先 ifconfig wlan0 up (Tina softap 内部已经进行相应处理)。

Tina softap 目前仅支持基于 hostapd 进行 broadcom 模组。

station 和 softAP 共存，使用时需要将 package/allwinner/softap/src/include/aw_softap_intf.h 中：

```
#define IW_UP_BROADCOM_WLAN1 0  
改为：  
#define IW_UP_BROADCOM_WLAN1 1
```

如使用的模组不支持基于 hostapd 在 wlan1 上建立 softAP，可以自行使用命令行添加 wlan1，建立热点（由于未使用 hostapd 标准组件，因此未将此方式兼容进 Tina softap）：

- 1、启动 wlan0：ifconfig wlan0 up
- 2、添加 wlan1：dhd_priv iapsta_init mode apsta ifname wlan1
- 3、设置 softAP 参数：

```
dhd_priv iapsta_config ifname wlan1 ssid tttv chan 6 amode [open/wpa2psk] emode [none/aes]  
key 12345678  
设置softAP参数项为：  
ssid: 示例为tttv  
channel: softAP通道, 示例为6  
amode: 加密方式, open(无密)/wpa2psk  
emode: 加密算法, none/aes  
key: 密钥, 示例为12345678
```

- 4、启动 softAP：dhd_priv iapsta_enable ifname wlan1
- 5、使用 wlan0 连接其他 AP
- 6、关闭 softAP：dhd_priv iapsta_disable ifname wlan1

5.6.2 Tina Softap 中 firmware 参数设置

如果使用的模组不需要加载 firmware(如 realtek 的常见模组)，则无需修改。如果使用的模组在启动时需要加载 firmware，可能需要修改 firmware 的相关参数。目前 Tina softap 默认支持的模组为 AP6212，如果使用的为其他模组，需要修改 package/allwinner/softap/src/include/wifi.h 中如下宏：

```
/*path of firmware for Wi-Fi in different mode*/
#ifndef WIFI_DRIVER_FW_PATH_STA
#define WIFI_DRIVER_FW_PATH_STA "/lib/firmware/fw_bcm43455c0_ag.bin"
#endif

#ifndef WIFI_DRIVER_FW_PATH_AP
#define WIFI_DRIVER_FW_PATH_AP "/lib/firmware/fw_bcm43455c0_ag_apsta.bin"
#endif

#ifndef WIFI_DRIVER_FW_PATH_P2P
#define WIFI_DRIVER_FW_PATH_P2P "/lib/firmware/fw_bcm43438a0_p2p.bin"
#endif

#ifndef WIFI_DRIVER_FW_PATH_PARAM
#define WIFI_DRIVER_FW_PATH_PARAM "/sys/module/bcmdhd/parameters/firmware_path"
#endif
```

其中，WIFI_DRIVER_FW_PATH_STA、WIFI_DRIVER_FW_PATH_AP、WIFI_DRIVER_FW_PATH_P2P 分别为 Wi-Fi 模组 station 模式、softAP 模式、P2P 模式使用的模组在设备上的放置路径；WIFI_DRIVER_FW_PATH_PARAM 为 firmware 参数设置节点，由内核 Wi-Fi 驱动指定。

5.7 Softap demo

Demo 部分，softAP 启动部分示例了 softAP 启动的主要流程；softAP 长时耐久性测试，作为内部测试用例，一般不为用户所使用。但它作为一个完整的 softAP 启动、关闭的流程，可以作为读者编程的重要参考。

5.8 Softap 使用

5.8.1 使用流程

1. 两块板子分别通过串口连接PC与开发板，系统起来，进入Linux shell；
2. 启动softap并查看ip地址：
 - softap_up fly 00000000 //ssid和passwd可以自己定
 - ifconfig
3. 用另外一块板子或是手机连接起的ap fly；
4. 用连接的板子或是手机ping ip(ap模式的)；

5.8.2 测试 log

```
root@TinaLinux:/# softap_up fly 00000000
*****
Start hostapd test!
*****
wpa2-psk!
Start to set softap!
Message is: OK
Set softap finished!
[ 43.988675] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
Start to start softap!
SoftAP started successfullyHaving catch sig_chld!
Message is: OK
Start softap finished!
```



6 常见问题

6.1 编译问题

6.1.1 找不到 wowlan 变量

1. 现象：

```
drivers/net/wireless/xr829/umac/main.c:870:17: error: 'struct wiphy' has no member named '
    wowlan'

if ((hw->wiphy->wowlan->flags || hw->wiphy->wowlan->n_patterns)
```

2. 原因：

wowlan成员变量受CONFIG_PM控制，没有打开导致的。休眠唤醒的依赖。

3. 解决方案：

在内核配置

```
-Power management options ---->
    Device power management core functionality
```

6.1.2 找不到 xxx.ko

1. 现象：

```
sunxi_wlan_get_bus_index...xradio_core.ko undefined!
```

2. 原因：

缺少配置misc。

3. 解决方案：

在内核配置

```
m kernel_menuconfig-->
    Device drivers-->
        Misc devices-->
            Allwinner rfkill driver
```

6.1.3 mmc_xxx undefined

1. 现象：

```
drivers/built-in.o: In function scan_device_store':
lichee/linux-5.4/drivers/misc/sunxi-rf/sunxi-wlan.c:309: undefined reference
  to sunxi_mmc_rescan_card'
lichee/linux-5.4/drivers/misc/sunxi-rf/sunxi-wlan.c:309:(.text+0x5fc40): relocation
  truncated to fit:
  R_AARCH64_CALL26 against undefined symbol `sunxi_mmc_rescan_card'
```

2. 原因：

没有配置mmc。

3. 解决方案

```
Device Drivers --->
  <*> MMC/SD/SDIO card support --->
  <*> Allwinner sunxi SD/MMC Host Controller support
```

6.1.4 缺少依赖库

1. 现象：

```
Package kmod-net-xr829 is missing dependencies for the following libraries:
  cfg80211.ko
  mmc_core.ko
  sunxi-wlan.ko
```

2. 原因：

依赖库需要编译进内核，不能以模块方式编译进去。

3. 解决方案。

```
在内核配置如下模块时，配置成y
CONFIG_RFKILL =y
CONFIG_CFG80211=y
CONFIG_MMC=y
CONFIG_MAC80211=y
```

6.2 驱动加载问题

6.2.1 XR829 模组 ifconfig 显示：No such device

1. 现象：

```
ifconfig: SIOCGIFFLAGS: No such device
```

2. 原因：

```
firmware选择不匹配。  
- lsmod查看驱动已经正常加载。  
- dmesg 查看加载log发现：  
  
[ 195.966066] [XRADIO_ERR] xradio_load_firmware: Wait for wakeup:device is not responding.  
XR829换了40M晶振。
```

3. 解决方案

```
tina配置选择40M晶振的firmware  
firmware --->  
[*] xr829 with 40M sdd
```

6.2.2 XR829 can't open /etc/wifi/xr_wifi.conf, failed

1. 现象：

```
lsmod驱动没有正常加载。
```

2. 原因：

```
- dmesg查看log:  
[ 6.802331] [XRADIO_ERR] can't open /etc/wifi/xr_wifi.conf, failed(-30)  
[ 6.802338] [XRADIO_ERR] Access_file failed, path:/etc/wifi/xr_wifi.conf!  
[ 6.914044] sunxi-mmc sdcl: no vqmmc,Check if there is regulator  
[ 7.028376] [XRADIO_ERR] xradio_load_firmware: Wait_for_wakeup: can't read control  
register.  
busnum配置错误，原理图上使用的是sdc0。
```

3. 解决方案

```
board.dts中配置wlan时  
busnum = 0;
```

6.2.3 驱动加载问题总结

6.2.3.1 配置问题

- 1.内核驱动, Tina modules, Tina firmware三者必须正确对应同一个模组。
- 2.注意common下的modules.mk的编写。
- 3.Sdio的配置一定要根据原理图选择对应busnum。

可能导致：

- 1.扫卡失败。
- 2.下载firmware失败。

最终导致驱动加载失败。

6.2.3.2 供电问题

检查VCC_WIFI和VCC_IO_WIFI两路电。

不同模组对供电时序有一定要求，比如RTL8723ds需要两路电同时供电，针对有AXP的方案，一定要注意供电的配置，特别是enable的时间。

1. 硬件方面：主要排查两路电的供电方案，是否是同一路供电，若是分开供电，要考虑两路供电的时序，例如DCDC1--->VCC_WIFI, LD0A--->VCC_IO_WIFI, 那么DCDC1和LD0A的时序就得考虑。

2. 软件方面主要是sysconfig.fex或者board.dts的配置，分开供电的是否需要单独配置。

如：R818硬件设计是两路电分开供电。

可能导致：

- 1.扫卡失败。
- 2.下载firmware失败。
- 3.sdio_clk没有时钟。
- 4.32k晶振不起振。

最终导致驱动加载失败。

6.2.3.3 sdio 问题

1.sdio busnum配置错误。

2.驱动WL-REG-ON的方式不对。例如：

```
[SBUS_ERR] sdio probe timeout!
```

```
[XRADIO_ERR] sbus_sdio_init_failed
```

这个问题主要是sdio扫卡失败，跟sdio上电时序有关，可在drivers/net/wireless/xradio/wlan/platform.c中

xradio_wlan_power函数sunxi_wlan_set_power(on)后面加上一段延时。

RLT8723ds需要先高一低一高的方式。

可能导致：

- 1.扫卡失败。
- 2.下载firmware失败。
3. sdio_clk没有时钟。
4. 32k竞争不起振。
5. WL-REG-ON无法正常被拉高。

最终导致驱动加载失败。

6.3 supplicant 服务问题

6.3.1 找不到 wpa_supplicant.conf 文件

1. 现象：

```
起supplicant失败
- ps发现没有supplicant进程。
- 于是手动执行wpa_supplicant -D nl80211 -i wlan0 -c /etc/wpa_supplicant.conf -B
提示：
Failed to open config file '/etc/wpa_supplicant.conf', error: No such file or directory
Failed to read or parse configuration '/etc/wpa_supplicant.conf'.
```

2. 原因：

路径错误。

3. 解决方案

tina正常的路径一般在/etc/wifi/wpa_supplicant.conf
在wifimanage包下面配置正确的路径，保持和启动脚本一致。

6.4 wifimanager 使用问题

6.4.1 联网时出现：network not exist!

1. 现象：

```
wifi_connect_ap_test ssid passwd
network not exist!
```

2. 原因：

```
- lsmod查看驱动已经正常加载。
- ifconfig查看wlan0已经正常up。
- ps查看supplicant服务已经正常启动。
- 使用wifi_scan_results_test扫描网络
root@TinaLinux:/# wifi_scan_results_test
*****
***Start scan!***
*****
bssid / frequency / signal level / flags / ssid
*****
Wifi get_scan_results: Success!
*****
没有任何网络扫描到。
```

3. 解决方案

一般是信号太多，没有板载天线，尝试外加一根天线。

6.5 上层网络应用服务问题

6.5.1 XR829 ping 压力测试: poll time out

1. 现象:

ping 压力测试，一段时间后出现poll time out。

2. 原因:

ping的网络性能不好。连接的公司内网可能存在一些未知的限制。

3. 解决方案

尝试连接另外的路由器测试。






著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

、 **全志科技** （不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。