TinaLinux

Wifi 开发指南

1.0

2019.03.03

文档履历

版本号	日期	制/修订人	内容描述
1.0	2019.03.03	AWA1423	创建

目录

1. 概述

1.1 编写目的

介绍 Allwinner 平台上 wifi 驱动移植，介绍 Tina wifi 管理框架，包括 Station，Ap 以及 smartlink（透传配网），

1.2 适用范围

Allwinner 软件平台 Tina v3.0 版本以上 Allwinner 硬件平台 R 系列（R6，R11，R16，R18，R30，R40，R328，R331…）

1.3 相关人员

适用 Tina 平台的广大客户。

2. wifi 简介

2.1 wifi 工作的几种模式

目前 tina 平台上的 wifi 一般可处于 3 种工作模式，分别是 STATION，AP，MONITOR。

- STATION：连接网线网络的终端，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，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 wifi 软件结构

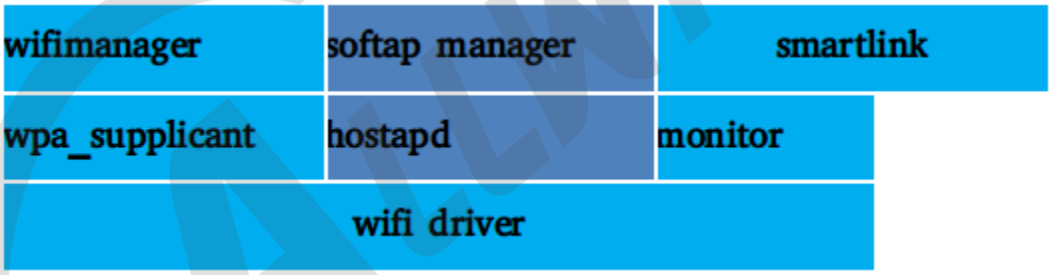

图 1: Tina 软件结构图

- wifimanger：主要用于 STATION 模式，提供 wifi 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

3. 3. wifi 模组移植

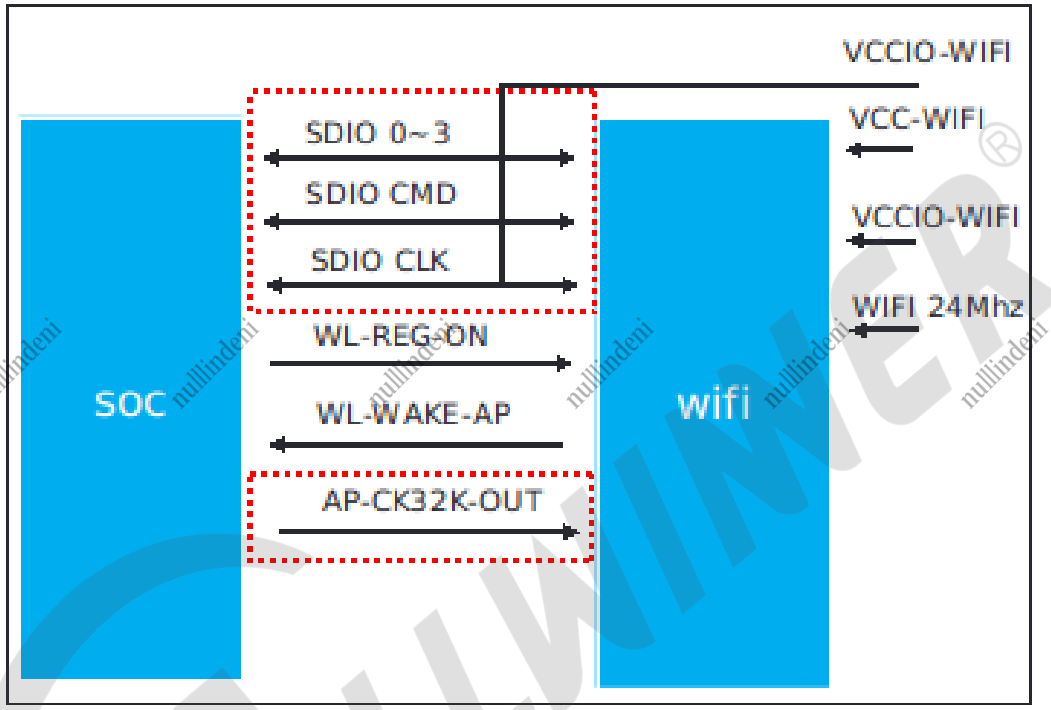

图 2: 主控与 wifi 硬件连接简图

wifi 模组工作的条件，如上图，需要满足以下几个条件。

- （1）供电：一般有两路供电，其中 VCC-WIFI 为主电源，VCCIO-WIFI 为 IO 上拉电源。
- （2）使能：要能正常工作，需要 WL-REG-ON 给高电平。
- （3）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 3 条数据线。
- （4）唤醒主控：当系统休眠时，wifi 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。
- （5）24/26Mhz 时钟信号
- （6）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 wifi 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的配置 device 即可，allwinner device

除了可以 dts 外 (linux-3.4 内核无 dts)，还可通过 sysconfig.fex 的方式，sysconfig.fex 的优先级高于 dts，一般情况下，直接配置 sysconfig.fex 即可。

linux 3.4 device（sysconfig.fex 配置）

```
[rf_para]
module_num = 10 /*用于区分模组型号：默认8~10*/
module_power1 = "axp22_dldo1" /*power1使用的电源树：dld01*/
module_power1_vol = 3300000 /*power1使用的电源树电压设置为3.3v*/
module_power2 = "axp22_dldo2" /*power2使用的电源树：dld02*/
module_power2_vol = 3300000 /*power2使用的电源树电压设置为3.3v*/
module_power3 = "axp22_aldo3" /*power3使用的电源树：dld03*/
module_power3_vol = 3300000 /*power3使用的电源树电压设置为3.3v*/

/*以上，是使用pmu供电的配置，如果直接供电，可省略*/

power_switch =
chip_en = /*电源使能，备用于使用gpio高低点平打开电源*/
lpo_use_apclk = "losc_out" /*32khz的时钟使能*/

[wifi_para]
wifi_used = 1 /*是否使用wifi*/
wifi_sdc_id = 1 /*wifi所使用的sdio 卡号*/
wl_reg_on = port:PL06<1><default><default><0> /*使能脚所使用的gpio*/
wl_host_wake = port:PL07<4><default><default><0> /*唤醒主控所使用的gpio*/

[mmc1_para]
sdc_used = 1
sdc_detmode = 4
sdc_buswidth = 4
sdc_clk = port:PG00<2><1><1><default> /*sdio所使用的gpio号*/
sdc_cmd = port:PG01<2><1><1><default>
sdc_d0 = port:PG02<2><1><1><default>
sdc_d1 = port:PG03<2><1><1><default>
sdc_d2 = port:PG04<2><1><1><default>
sdc_d3 = port:PG05<2><1><1><default>
sdc_det =
sdc_use_wp = 0
sdc_wp =
sdc_isio = 1
sdc_regulator = "none"
```

linux 3.4 driver 路径，详情请参考以下代码路径

```
tina/lichee/linux-3.4/drivers/misc/rf_pm
```

linux3.4 以上，device（sysconfig.fex 配置）

```
[sdc1]
sdc1_used = 1
bus-width = 4
sdc1_clk = port:PG00<2><1><3><default>
sdc1_cmd = port:PG01<2><1><3><default>
sdc1_d0 = port:PG02<2><1><3><default>
sdc1_d1 = port:PG03<2><1><3><default>
sdc1_d2 = port:PG04<2><1><3><default>
sdc1_d3 = port:PG05<2><1><3><default>
sd-uhs-sdr50 =
sd-uhs-ddr50 =
sd-uhs-sdr104 =
cap-sdio-irq =
keep-power-in-suspend =
ignore-pm-notify =
max-frequency = 150000000 /*sdio所使用的最大扫卡频率*/
mix-frequency = 150000000

[wlan]
wlan_used = 1
compatible = "allwinner,sunxi-wlan"
clocks = "losc_out"

wlan_power_num = 2 /*使用pmu供电，所使用的电源数量*/
wlan_power1 = "vcc-wifi1" /*使用pmu供电，使用的power1标示*/
wlan_power2 = "vcc-wifi2" /*使用pmu供电，使用的power2标示*/
wlan_io_regulator = "vcc-io-wifi" /*使用pmu供电，使用的io标示*/

/*上面4行表示使用pmu供电的配置项，对应后面的regulator选项。如果不使用pmu供电，可忽略。*/

wlan_busnum = 1
wlan_regon = port:PE06<1><1><1><0>
wlan_hostwake = port:PE05<6><default><default><default>

[regulator0]
compatible = "axp221s-regulator"
regulator_count = 20
......
regulator2 = "axp221s_dcdc2 none vdd-cpua"
regulator3 = "axp221s_dcdc3 none vdd-sys vdd-gpu"
regulator4 = "axp221s_dcdc4 none"
regulator5 = "axp221s_dcdc5 none vcc-dram"
regulator6 = "axp221s_rtc none vcc-rtc"
regulator7 = "axp221s_aldo1 none vcc-25 csi-avdd"
regulator8 = "axp221s_aldo2 none vcc-ephy0"
regulator9 = "axp221s_aldo3 none avcc vcc-pll"
regulator10 = "axp221s_dldo1 none vcc-io-wifi vcc-pg " /*io power 挂在dld01上*/
regulator11 = "axp221s_dldo2 none vcc-wifi1" /*power1 挂在dld02上*/
regulator12 = "axp221s_dldo3 none vcc-wifi2" /*power2 挂在dld03上*/
regulator13 = "axp221s_dldo4 none vdd-sata-25 vcc-pf"
```

```
regulator14 = "axp221s_eldo1 none vcc-pe csi-iovcc"
.....
```

linux 3.4 以上 driver，详情请参考以下代码路径

```
tina/lichee/linux-XXX/drivers/misc/sunxi-rf
```

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。

3.1.1 修改模组厂提供的 wifi driver

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。

下面先说明 tina 平台提供给 driver 的函数，其中 linux 3.4 跟其他内核稍微有些区别。

linux 3.4

```
#include <mach/sys_config.h>
#include <linux/gpio.h>

/*
*函数功能： sdio扫卡
*参数 id： 卡号，（sdio 0 or 1 ...）
*参数 insert： 0，卡插入，进行扫卡； 1，卡拔出。
*返回值：无
*/
extern void sunxi_mci_rescan_card(unsigned id, unsigned insert);

/*
*函数功能： wifi模组上电，使能。
*参数 on： 0，上电；1，掉电。
*返回值： 无
*/
```

```
extern void wifi_pm_power(int on);
```

linux 3.4 以上

```
#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： wifi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);
```

首先是将 wifi driver 放到 linux-4.9/drivers/net/wireless，填充对应的上电，扫卡等函数。linux 3.4 的驱动请参考：

esp8089模组：
tina/lichee/linux-3.4/drivers/net/wireless/esp8089/sdio_stub.c

xr819模组：
tina/lichee/linux-3.4/drivers/net/wireless/xradio/wlan/platform.c

linux 3.4 以上的驱动请参考：

tina/lichee/linux-4.9/drivers/net/wireless/rtl8723ds/platform/platform_ARM_SUNnI_sdio.c

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-4.9/drivers/net/wireless/Kconfig

example：
+source "drivers/net/wireless/xr829/Kconfig"

tina/lichee/linux-4.9/drivers/net/wireless/Makefile
example：
+obj-$(CONFIG_XR829_WLAN) += xr829/

配置完成后，可执行 make kernel_menuconfig 中选上，编译的时候，就会把指定的 driver 编译。

```
Device Drivers --->
  [*] Network device support --->
    [*] Network device support --->
      [*] Wireless LAN --->
        [] xxx模块
```

3.1.2 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。

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/XXX模组

tina/package/firmware/linux-firmware/XXX模组/XXX.mk

example:
tina/package/firmware/linux-firmware/xr829

make munuconfig
    Firmware --->
        < > xr829-firmware..... Xradio xr829 firmware
```

其次是配置 ko。

```
tina/target/allwinner/xxx方案/modules.mk

example:
tina/target/allwinner/cowbell-perf1/modules.mk

make munuconfig
    Kernel modules --->
        Wireless Drivers --->
            XXX 模块
```

3.1.3 配置 sysconfig.fex

前面已经阐述，见第 3 章节开头描述。

3.1.4 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： rmmod

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

3.1.5 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。
- 配置 sysconfig.fex。
- 验证。

目前 tina 平台的 linux 内核版本有 linux 3.4,linux 4.4,linux 4.9，由于历史原因，很有可能内核版本之间的配置有些不一样，主要体现在 device：sysconfig.fex 以及 driver：sunxi-rf。用户在模组移植时，可参考对应各个内核版本进行参考。

方案	内核版本	sysconfig.fex 路径
astar_parrot	linux 3.4	tina/target/allwinner/astar-parrot/configs/sysconfig.fex

方案	内核版本	sysconfig.fex 路径
tulip_d1	linux 4.4	tina/target/allwinner/tulip_d1/configs/sysconfig.fex
cowbell_demo	linux 4.9	tina/target/allwinner/cowbell_demo/configs/sysconfig.fex

3.2 tina 平台已经移植的模组

tina 平台上已经移植过多款 wifi 模组，支持列表如下：

```
Ampak： AP6212,AP6212A,AP6255,AP6256,AP6335等。
Realtek： RTL8723DS（linux 3.4/4.9），RTL8822cs（linux 4.9）
Xradio： XR819，xr829
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。如果按照在对应的 menuconfig 中选上，还是不能工作，就按照 3.1 小节的步骤依次排查原因。同时，如果有些方案可能在 make menuconfig 中无法显示相应的 Kernel modules，这是因为在方案下的 modules.mk 文件中没有添加，可按照 3.1.2 小节的方式进行添加。

以下列出各个模组 kernel_menuconfig 以及 menuconfig 的选项。

（1）AP 系列的模组

make kernel_menuconfig 配置

```
Device Drivers --->
  Network device support --->
    Wireless LAN --->
      <M> Broadcom FullMAC wireless cards support
        (/lib/firmware/fw_bcmdhd.bin) Firmware path
        (/lib/firmware/nvram.txt) NVRAM path

PS： AP系列的模组，如果AP6212，AP6255，AP6256..都是用的同一份driver
```

make menuconfig 配置

```
Kernel modules--->
      Wireless Drivers--->
            <*> kmod-net-broadcom
Firmware--->
      <*> ap6212-firmware. ---根据模组型号选择
```

（2）XR819

make kernel_menuconfig 配置

```
Device Drivers --->
  Network device support --->
    Wireless LAN --->
      <M> XR819 WLAN support --->
/*or*/<M> XRadio WLAN support --->
```

make menuconfig 配置

```
Kernel modules--->
  Wireless Drivers--->
    <*> kmod-net-broadcom
Firmware--->
      <*> xr819-firmware..... Xradio xr819 firmware
```

（3）XR829

make kernel_menuconfig 配置

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

make menuconfig 配置

```
Kernel modules--->
  Wireless Drivers--->
    <*> kmod-net-xr829..... xr829 support (staging)
Firmware--->
    <*> xr829-firmware..... Xradio xr829 firmware
```

（3）RTK8723DS

make kernel_menuconfig 配置

```
Device Drivers --->
  Network device support --->
    Wireless LAN --->
      <M> Realtek 8723D SDIO or SPI WiFi
```

make menuconfig 配置

```
Kernel modules--->
  Wireless Drivers--->
    <*> kmod-net-rtl8723ds..... RTL8723DS support (staging)
Firmware--->
      <*> r8723ds-firmware..... RealTek RTL8723DS firmware
```

（4）RTK8822CS

make kernel_menuconfig 配置

```
Device Drivers --->
  Network device support --->
    Wireless LAN --->
      <M> Realtek 8822C SDIO WiFi
```

make menuconfig 配置

```
Kernel modules--->
  Wireless Drivers--->
    <*> kmod-net-rtl8822cs..... RTL8723CS support (staging)
Firmware--->
  <*> rtl8821cs-firmware..... RealTek RTL8821CS firmware
```

（4）ESP8089

make kernel_menuconfig 配置

```
Device Drivers --->
  Network device support --->
    Wireless LAN --->
      <*> Eagle WLAN driver
```

make menuconfig 配置

```
Kernel modules--->
  Wireless Drivers--->
    <*> kmod-esp8089..... esp8089 support (staging)
Firmware--->
  <*> esp8089-firmware..... esp8089 firmware
```

4. wifi 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 wifi daemon(后面简称 wifi daemon API), 另外一套则不经过 wifi daemon(后面简称 wifi API).

（1）关于 wifi daemon API 与 wifi daemon: Wifi daemon API 和 wifi daemon(可执行程序) 是基于 wifi API 再次封装. Wifi 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 wifi daemon API 编写的程序对 wifi 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 wifi 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 wifi 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。

4.2 编译配置

WIFI sdk 相关 menuconfig 配置如下：

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

- 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.

4.3 Wifi daemon API 说明

4.3.1 准备

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

4.3.1.2 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		例子
-h, --help	print this help and exit	wifid -h
-c, --connect	connect AP,-c <ssid> <password>	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
-r, --remove_net	remove network in config,-r <ssid>	wifid -r aw-test
-o, --open	open wifi daemon	wifid -o

参数说明		例子
-d, --close wifi daemon	close	wifid -d
--debug, debug log level,--debug <num>	set	wifid --debug 5 -c aw-test 12345678

4.3.2 WIFI daemon API

Wifi daemon API 是经过 wifi_daemon 可执行程序, 通过调用不同的接口来控制 wifi_daemon. wifi daemon 应用本质上也是调用的 wifi API 接口. 使用 wifi 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 wifi daemon 已经处理.Wifi daemon API 较 wifi API 简单, 如用户不想自行处理 wifi 内部的连接状态, 可使用这套接口.

4.3.2.1 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,wifi 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, 当改网络再此有效时 (再次接上路由器), 会进行自动连接。
- 【参数说明】: 大于等 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,

4.3.2.2 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 0: 表示执行成功; 小于 0: 表示执行失败。

4.3.2.3 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；

4.3.2.4 移除网络

- aw_wifid_remove_networks
- 【函数原型】: int aw_wifid_remove_networks(char *pssid,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;

4.3.2.5 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wifi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;

4.3.2.6 Wifi daemon 打开

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

4.3.2.7 Wifi daemon 关闭

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

4.4 Wifi 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 "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

4.4.1.2 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 wifi 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.

```
Wifi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 DISCONNETED 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
```

4.4.2 wifi 打开和关闭

4.4.2.1 wifi 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 wifi，并获取操作 wifi interface 的句柄。
- 【参数说明】:
  - tWifi_event_callback pcd：wifi 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 wifi 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 wifi

操作接口
– NULL：失败。

4.4.2.2 wifi 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 wifi 状态切换的时候, 会回调这个函数。
- 【参数说明】:
  – struct Manager *wmg: wifi 的状态, 反馈事件,wifi 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

4.4.2.3 wifi 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_mgnt, const char *passwd, int event_label);
    int (*connect_ap_auto)(int event_label);
    int (*add_network)(const char *ssid, tKEY_MGMT key_mgnt, 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】: wifi 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。

4.4.2.4 wifi 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 wifi。
- 【参数说明】:const aw_wifi_interface_t *p_wifi_interface_t：打开 wifi 时获得的操作句柄。
- 【返回说明】: int 0: 成功； 非 0: 失败；

4.4.3 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 wifi 事件回调函数。用户不可直接调用，需借助于 aw_wifi_on 返回的 wifi 操作句柄。
- 【参数说明】:tWifi_event_callback pcb
- 【返回说明】: int 0: 成功； 非 0: 失败。

4.4.4 获取 wifi 信息

- aw_wifi_get_wifi_state
- 【函数原型】: enum wmgState aw_wifi_get_wifi_state();
- 【功能描述】: 获取 wifi 此刻的状态
- 【参数说明】: 无
- 【返回说明】: 返回 wifi 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 wifi 此刻的事件
- 【参数说明】: 无
- 【返回说明】: 返回 wifi 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 wifi 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 WiFi 状态为 WIFIMG_WIFI_DISABLED，即 WiFi 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

4.4.5 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 wifi 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。

4.4.6 连接与断开 AP

4.4.6.1 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 0: 表示执行成功; 小于 0: 表示执行失败。

4.4.6.2 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 wifi 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 0: 表示执行成功; 小于 0: 表示执行失败。

4.4.6.3 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 wifi 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 0: 表示执行成功; 小于 0: 表示执行失败。

4.4.6.4 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 wifi 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 0: 表示执行成功; 小于 0: 表示执行失败。

4.4.7 获取 IP 地址

- start_udhcpc
- 【函数原型】: void start_udhcpc();
- 【功能描述】: 启动 udhcpc 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

4.4.8 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 wifi 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 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 wifi 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：值一结果参数。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 0: 表示执行成功; 小于 0: 表示执行失败。

4.4.9 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 wifi 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 0: 表示执行成功; 小于 0: 表示执行失败。

- 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 wifi 操作句柄。
- 【参数说明】：无
- 【返回说明】：大于等 0: 表示执行成功; 小于 0: 表示执行失败。

4.4.10 打印 log 控制

头文件

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

4.4.10.1 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】:

```
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

4.4.10.2 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

4.4.10.3 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

4.4.10.4 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

4.4.10.5 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。

4.4.10.6 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

4.4.11 编程建议

4.4.11.1 wifi_on

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

4.4.11.2 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 wifi 的事件，同理，调用 add_event_callback 接口添加事件回调函数。

4.4.11.3 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 wifi，完全关闭前一次调用 aw_wifi_on 以及后续调用的其他 wifi 接口的影响，关闭后所有 wifi 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 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 targer 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，再选择 softap-demo 包，表示 softap app demo 程序。

```
<*> softap..... .....Tina softap manager --->
```

- 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 wifi 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 wifi 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 wifi 驱动（建议）；若无上述需求，也可在开机之后手动装载 wifi 驱动。wifi 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. wifi 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 APP 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 wifi 打开和关闭

5.4.1 wifi 打开

wifi 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（wifi 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 wifi 服务关闭

wifi 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，wifi 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 WIFI 模式对应的 firmware。
- 【参数说明】:ap_sta：需要切换的 firmware 对应的 WIFI 模式。可以输入的参数有``STA"、``AP"、``P2P"，代表目前支持 WIFI 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 WIFI 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 WIFI 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 wifi1111;
  - 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 WIFI 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 WIFI 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 staion(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 WIFI 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 WIFI 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 WIFI 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。

6. smartlink

Tina 中目前支持的 wifi 模组有全志 Xradio，Broadcom AP 系列模组。支持的配网方式有 airkiss、soundwave（声波）、softap（热点），另外 Xradio 模组还支持 smartconfig，Broadcom 模组还支持 cooee。

6.1 sdk 代码目录

源码路径为/packge/allwinner/smartlinkd。包括 smartlink 服务进程、配网协议和 demo 程序。

```
├── demo #存放demo
├── files #配置文件
│   ├── smartlinkd.init #开机自启动配置文件
│   └── usrapp #手机APP目录
│   ├── README
│   ├── source #APP源码，详见后说明
│   └── testapp #Android 测试apk，详见后说明
├── Makefile
├── README
└── src
├── sm_link_manager.c
├── smg_log.c
├── smg_event.c
└── protocol
    ├── dependent #模组相关协议
    │   ├── broadcom #broadcom模组底层协议
    │   ├── realtek #Realtek模组底层协议
    │   └── xr819 #xr819模组底层协议
    └── independent #模组无关协议（适用于所有模组）
        ├── softAP #热点配网协议
        └── soundwave #声波配网协议
```

```
souce目录下
.
├── CooeeDemoAndroid-v1.8.0.zip #broadcom cooee协议Android端源码
├── CooeeDemoIOS-v1.5.0.zip #broadcom cooee 协议IOS端源码
├── SoundWaveAndroid.zip #声波Android端源码
├── SoftApAndroid.zip #Softap Android端源码
└── XConfig.zip #XR819 smartconfig协议Android 端源码

test目录下的apk，安装到Android手机中可直接使用，方便测试
```

```
.
├── AirKissDebugger.apk #airkiss apk，xr819与broadcom通用
├── CooeeDemo.apk #broadcom cooee Android端测试apk
├── smatconfig.apk #XR819 smartconfig Android端测试apk
├── softAP.apk #softap Android端测试apk
└── soundwave.apk #声波配网 Android端测试apk
```

6.2 编译配置

xradio menuconfig 配置如下：

```
make menuconfig
  Allwinner --->
    smartlinkd --->
      <*> smartlinkd-demo..... smartlink demo
      -*- smartlinkd-lib..... smartlinkd-lib
        [*] enable xradio airkiss support for lib
        [*] enable xradio smartconfig support for lib
        [*] enable soft ap support for lib
        [*] enable sound wave support for lib
```

- smartlinkd-demo：smartlinkd 实例程序
- smartlinkd-lib：demo 所使用的声波配网协议库

6.3 APP 编写说明

6.3.1 准备

(1) 导入接口头文件

```
#include "sm_link_manager.h"
```

(2) 链接动态库

libsm_mg.so

(3) 示例代码

<tina>/package/allwinner/smartlinkd/demo/main.c

6.3.2 API

- sm_link_init
- 【函数原型】: int sm_link_init(int protocol_num);
- 【功能描述】: 初始化 smartlink，注意是分配资源
- 【参数说明】:
  - protocol_num: 需要启动的配网协议数量，支持同时启动多个协议，前提是模组能够支持多种模式共存。
- 【返回说明】: 0：初始化成功；非 0：初始化失败。

- sm_link_protocol_enable
- 【函数原型】: int sm_link_protocol_enable(int type,struct proto_params *p,int protocol_num);
- 【功能描述】: 协议使能
- 【参数说明】:
  - type: 需要启动的协议类型
  - p：启动的协议需要传递的参数
  - protocol_num：启动的协议数量
- 【返回说明】: 0：成功；非 0：初始化失败。

- sm_link_wait_get_results
- 【函数原型】: int sm_link_wait_get_results(int type,struct net_info *info);
- 【功能描述】: 协议使能
- 【参数说明】:
  - type: 需要启动的协议类型

– info：存放返回的网络信息

- 【返回说明】：0：成功；小于 0：初始化失败。

- sm_link_deinit
- 【函数原型】：int sm_link_deinit();
- 【功能描述】：smartlink 资源销毁，配合 sm_link_init 使用
- 【参数说明】：无
- 【返回说明】：0：成功；小于 0：初始化失败。

关于启动多个协议：目前 tina smartlink 同时支持启动多个 protocol，前提是 wifi 模组能够支持多种模式共存。声波能配合其他协议同时启动。如同时启动 softap 和 soundwave，或者同时启动 airkiss 和 soundwave。另外，需要注意的时，在同时启动多个协议的时候，struct proto_params 参数需要按照指定顺序赋值。以下是协议的顺序。

```
SM_LINK _AP_COOEE->
SM_LINK_AP_NEEZE->
SM_LINK_AP_AIRKISS->
SM_LINK_XR_AIRKISS->
SM_LINK_XR_SMARTCONFIG->
SM_LINK_SOUND_WAVE->SM_LINK_SOFTAP
.....
```

例子：同时启动 xradio airkiss 和 soundwave

```
void *sound_wave_argv[4] = {
        "default", //录音声卡设备
        "0", //频率类型：0-LOW，1-MID，2-HIGH
        "16000", //录音采样率
        "60", //等待超时时间
    };

    void *xr_airkiss_argv[3] = {
        "60", //等待超时时间
        "1234567890123456", //AES KEY
        "wlan0" //使用的网卡设备
    };

#define WORK_PROTOCOL_NUM 2
```

```
#define COM_PROTOCOL (SM_LINK_XR_AIRKISS|SM_LINK_SOUND_WAVE)
struct proto_params params_com[WORK_PROTOCOL_NUM];
params_com[0].argv = xr_airkiss_argv; //第一个参数必须为xradio airkiss的参数
params_com[1].argv = sound_wave_argv;
```

例子：同时启动 softap 和 soundwave

```
#define WORK_PROTOCOL_NUM 2
#define COM_PROTOCOL (SM_LINK_SOFTAP|SM_LINK_SOUND_WAVE)
struct proto_params params_com[WORK_PROTOCOL_NUM];
params_com[0].argv = sound_wave_argv; //第一个参数必须为声波的参数
params_com[1].argv = NULL; //softap不需要传参数
```

6.4 demo 使用说明

```
example: smartlink_demo V

V:take the following value
        0-softap
        1-soundwave
        2-xradio smartconfig
        3-xradio airkiss
        4-ampark (ap6212...) airkiss
        5-ampark (ap6212...) cooee
        6-composite (softap & soudwave)
```

7. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.