全志科技
Allwinner Technology

Tina

wifimanager API v2.4

文档履历

版本号	日期	制/修订人	制/修订记录
V1.0	2016/01/28		初始版本
V2.0	2016/05/18		更新接口名称，修改格式
V2.1	2016/09/23		更新接口名称
V2.2	2016/12/01		增加新的接口，修改一些接口的回调信息，修改格式
V2.3	2017/09/05		更新接口功能与使用说明
V2.4	2017/09/30		更新 get netid 接口

目　录

1. 概述

1.1. 编写目的

介绍 Tina wifi 管理开发接口和 Demo 代码。

1.2. 适用范围

Allwinner 软件平台 Tina v1.0 版本以上
Allwinner 硬件平台 R6 R11 R16 R18 R30 R40

1.3. 相关人员

适用 Tina 平台的广大客户。

2. Wi-Fi manager 相关说明

wifimanager 部分代码是 Tina 平台管理 wifi 与 AP 连接模块。主要功能包括打开/关闭，连接/断开 AP，获取连接过程中的状态信息。

2.1. sdk 代码目录

sdk 中 wifimanager 相关代码目录为 package\allwinner\wifimanager。包括 wifi 连接管理，事件监听和 demo 程序。

2.2. 编译配置

WIFI sdk 相关 menuconfig 配置如下：

tina 根目录下，输入 make menuconfig

选择： Allwinner --->

└> <*> wifimanager..... Tina wifi manager --->

└> < > wifimanager-demo..... Tina wifi app demo

注意:如果要参考 wifi app demo 代码，需要先选择

<*> wifimanager..... .....Tina wifi manager --->

再选择 wifimanager-demo 包，表示 wifi app demo 程序。

2.3. APP 编写说明

2.3.1. 导入接口文件

#include <wifi_intf.h>

2.3.2. 链接动态库

libwifimg.so

2.3.3. 示例代码

wifimanager app demo 代码目录为：

package\allwinner\wifimanager\demo。

2.4. wifi 打开和关闭

2.4.1. 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）

2.4.2. 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。

3. API 说明

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。
部分 API 的执行结果以事件的形式上报给 wifi 事件监听接口，参考 第 5 章 wifi 事件上报对照表

3.1. WiFi 打开与关闭 wifi on

3.1.1. wifi 打开

函数原型	const aw_wifi_interface_t *aw_wifi_on(tWifi_event_callback pcb, int event_label)
参数说明	tWifi_event_callback pcd：wifi 事件监听接口，详见 3.1.2 wifi 事件监听与处理。
	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。
功能描述	打开 wifi，并获取操作 wifi interface 的句柄。

成功返回事件	含义
WIFIMG_WIFI_ON_SUCCESS	Wi-Fi 打开成功
WIFIMG_NO_NETWORK_CONNECTING	Wi-Fi 打开过程中没有连接任何 ap
WIFIMG_NETWORK_CONNECTED	Wi-Fi 打开过程中连接上某个 ap
WIFIMG_PASSWORD_FAILED	Wi-Fi 打开过程中连接 ap 密码错误
WIFIMG_CONNECT_TIMEOUT	Wi-Fi 打开过程中连接 ap 超时(未分配到 ip 地址)
失败返回事件	含义
WIFIMG_WIFI_ON_FAILED	Wi-Fi 打开失败

3.1.2. wifi 事件监听与处理接口

函数原型	typedef void (*tWifi_event_callback) (tWIFI_STATE wifi_state, void *buf, int event_label);
参数说明	① tWIFI_STATE wifi_state：底层传递给 APP 的 wifi 事件 事件类型见 第 5 章 wifi 事件上报对照表 ② void *buf：存放状态信息。 ③ 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
功能描述	Wifi 事件监听与处理，只规定函数结构，具体由 app 定义与实现，并于打开 wifi 时作函数指针参数传递。

3.1.3. wifi 操作接口

函数原型	typedef struct{ int (*add_event_callback)(tWifi_event_callback pcb); int (*is_ap_connected)(char *ssid, int *len); int (*start_scan)(int event_label); 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;
参数说明	无
返回说明	无
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。

3.1.4. wifi 关闭

函数原型	int aw_wifi_off(const aw_wifi_interface_t *p_wifi_interface_t)
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；
功能描述	关闭 wifi。
成功返回事件	含义
WIFIMG_WIFI_OFF_SUCCESS	wifi 关闭成功
失败返回事件	含义
WIFIMG_WIFI_OFF_FAILED	wifi 关闭失败

3.2. 添加事件回调接口

函数原型	int (*add_event_callback)(tWifi_event_callback pcb);
参数说明	tWifi_event_callback pcb
返回说明	int 0:成功；非 0:失败。
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。

3.3. 获取 wifi 信息

3.3.1. aw_wifi_get_wifi_state

函数原型	int aw_wifi_get_wifi_state(void);
参数说明	void
返回说明	wifi manager 当前状态：详见此表格下部分。
功能描述	获取 wifi 状态信息。用户可直接调用，无需通过 aw_wifi_on 获取 wifi 操作句柄。
wifi 状态返回宏	含义
WIFIMG_WIFI_DISCONNECTED	WiFi 已断开(未与任何 ap 连接)
WIFIMG_WIFI_BUSING	WiFi 设备忙
WIFIMG_WIFI_CONNECTED	WiFi 已连接(已连接上 ap)
注意	
此状态宏只用于此函数的返回，用于主动查询 wifi 状态，区别于 wifi 上报事件状态宏。	

3.3.2. is_ap_connected

函数原型	int (*is_ap_connected)(char *ssid, int *len);
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	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
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。

3.4. 扫描 AP

3.4.1. start_scan

函数原型	int (*start_scan)(int event_label);
参数说明	int event_label：事件标签，tWifi_event_callback 回调时返回，用来标明是 start_scan 的回调事件。
返回说明	int 0:调用成功；非 0:调用失败。 调用成功，直接返回。 调用失败，返回事件：WIFIMG_DEV_BUSING_EVENT　　设备忙
功能描述	发送 scan 命令，扫描周围 ap。用户不可直接调用，需借助于 aw_wifi_on 返回的 wifi 操作句柄。

3.4.2. get_scan_results

函数原型	int (*get_scan_results)(char *result, int *len);
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。
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。

3.5. 连接与断开 AP

3.5.1. connect_ap

函数原型	int (*connect_ap)(const char *ssid, const char *passwd, int event_label);
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	int 0:调用成功； 非 0:调用失败；
功能描述	连接不知道加密方式的 ap，因连接时间长，不建议使用。 此接口只能连接当前可扫描得到的 ap。 用户不可直接调用，需借助于 aw_wifi_on 返回的 wifi 操作句柄。

成功返回事件	含义
WIFIMG_NETWORK_CONNECTED	连接成功
WIFIMG_CONNECT_TIMEOUT	连接超时
失败返回事件	**含义**
WIFIMG_CMD_OR_PARAMS_ERROR	命令或参数错误
WIFIMG_PASSWORD_FAILED	密码错误
WIFIMG_KEY_MGMT_NOT_SUPPORT	加密方式不支持
WIFIMG_OPT_NO_USE_EVENT	操作无效
WIFIMG_NETWORK_NOT_EXIST	网络不存在
WIFIMG_DEV_BUSING_EVENT	设备忙

3.5.2. connect_ap_key_mgmt

函数原型	int (*connect_ap_key_mgmt)(const char *ssid, tKEY_MGMT key_mgnt, const char *passwd, int event_label);
参数说明	ssid： 连接指定 AP 的 ssid key_mgnt： 连接指定 AP 的加密方式 passwd： 连接指定 AP 的密码 event_label： 事件标签，tWifi_event_callback 回调时返回，用来标明是 connect_ap_key_mgmt 的回调事件
返回说明	int 0:成功； 非 0:失败；
功能描述	连接指定加密方式的 AP，建议采用此方式连接。 此接口只能连接当前可扫描得到的 ap。 用户不可直接调用，需借助于 aw_wifi_on 返回的 wifi 操作句柄。

成功返回事件	含义
WIFIMG_NETWORK_CONNECTED	连接成功
WIFIMG_CONNECT_TIMEOUT	连接超时
失败返回事件	**含义**
WIFIMG_CMD_OR_PARAMS_ERROR	命令或参数错误
WIFIMG_PASSWORD_FAILED	密码错误
WIFIMG_KEY_MGMT_NOT_SUPPORT	加密方式不支持
WIFIMG_OPT_NO_USwifi onE_EVENT	操作无效
WIFIMG_NETWORK_NOT_EXIST	网络不存在
WIFIMG_DEV_BUSING_EVENT	设备忙

3.5.3. add_network

函数原型	int (*add_network)(const char *ssid, tKEY_MGMT key_mgnt, const char *passwd, int event_label);
参数说明	ssid: 连接指定 AP 的 ssid key_mgnt: 连接指定 AP 的加密方式 passwd: 连接指定 AP 的密码 event_label: 事件标签，tWifi_event_callback 回调时返回，用来标明是 add_network 的回调事件
返回说明	int 0:成功； 非 0:失败；
功能描述	添加指定 AP。此接口可用于连接匿名 AP。用户不可直接调用，需借助于 aw_wifi_on 返回的 wifi 操作句柄。

成功返回事件	含义
WIFIMG_NETWORK_CONNECTED	连接成功
WIFIMG_CONNECT_TIMEOUT	连接超时
失败返回事件	**含义**
WIFIMG_CMD_OR_PARAMS_ERROR	命令或参数错误
WIFIMG_PASSWORD_FAILED	密码错误
WIFIMG_KEY_MGMT_NOT_SUPPORT	加密方式不支持
WIFIMG_OPT_NO_USE_EVENT	操作无效
WIFIMG_NETWORK_NOT_EXIST	网络不存在
WIFIMG_DEV_BUSING_EVENT	设备忙

3.5.4. connect_ap_auto

函数原型	int (*connect_ap_auto)(int event_label);
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	int 0:成功； 非 0:失败；
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。

成功返回事件	含义
WIFIMG_NETWORK_CONNECTED	连接成功
WIFIMG_PASSWORD_FAILED	密码错误
WIFIMG_CONNECT_TIMEOUT	连接超时
失败返回事件	**含义**
WIFIMG_CMD_OR_PARAMS_ERROR	命令或参数错误
WIFIMG_KEY_MGMT_NOT_SUPPORT	加密方式不支持
WIFIMG_OPT_NO_USE_EVENT	操作无效
WIFIMG_NETWORK_NOT_EXIST	网络不存在
WIFIMG_DEV_BUSING_EVENT	设备忙

3.5.5. connect_ap_with_netid

函数原型	int (*connect_ap_with_netid)(const char *net_id, int event_label);
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	int 0:成功；

	非 0:失败；
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。
成功返回事件	含义
WIFIMG_NETWORK_CONNECTED	连接成功
WIFIMG_CONNECT_TIMEOUT	连接超时
失败返回事件	含义
WIFIMG_CMD_OR_PARAMS_ERROR	命令或参数错误
WIFIMG_KEY_MGMT_NOT_SUPPORT	加密方式不支持
WIFIMG_PASSWORD_FAILED	密码错误
WIFIMG_NETWORK_NOT_EXIST	网络不存在
WIFIMG_DEV_BUSING_EVENT	设备忙

3.5.6. disconnect_ap

函数原型	int (*disconnect_ap)(int event_label);
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
返回说明	int 0:成功；非 0:失败。 断开成功，会发送断开连接消息(WIFIMG_NETWORK_DISCONNECTED)
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。
成功返回事件	含义
WIFIMG_NETWORK_DISCONNECTED	断开连接

3.6. 获取配置信息

3.6.1. list_networks

函数原型	int (*list_networks)(char *reply, size_t reply_len, int event_label);
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	int 0:成功； 非 0:失败；
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。
失败返回事件	含义
WIFIMG_CMD_OR_PARAMS_ERROR	命令或参数错误
WIFIMG_DEV_BUSING_EVENT	设备忙

3.6.2. get_netid

函数原型	int (*get_netid)(const char *ssid, tKEY_MGMT key_mgmt, char *net_id, int *length);
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	int 0:成功； 非 0:失败；
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。

3.7. 删除 network 记录

3.7.1. remove_network

函数原型	int (*remove_network)(char *ssid, tKEY_MGMT key_mgmt);
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	int 0:成功； 非 0:失败；
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。

3.7.2. remove_all_networks

函数原型	int (*remove_all_networks)();
参数说明	void
返回说明	int 0:成功； 非 0:失败；
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。

4. 使用说明

4.1. 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。

4.2. 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。

4.3. 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。

4.4. 编程建议

wifi 操作时，需先调用 aw_wifi_on 开启 wifi，并获取 wifi 操作句柄。调用 aw_wifi_on 会在底层开启一个长时存在的扫描线程，进行 15s 一次 scan 动作。频繁的 scan 动作会占用较多的系统资源，并使 WIFI 模组处于繁忙状态，进而影响 WIFI 吞吐量。有如下编程建议供参考：

1、除非用户程序有频繁/经常调用 wifi 操作接口的需求，建议用户开启独立子进程来进行联网操作。联网操作结束后，直接退出子进程（不调用 aw_wifi_off，此时网络将保持连接状态，不会断开）；
2、若用户需查询网络连接状态，可在主进程中直接调用 aw_wifi_get_wifi_state(无需调用 aw_wifi_on)；
3、若用户需关闭网络时，先调用 aw_wifi_on，使用获得的 wifi 操作句柄作为参数执行 aw_wifi_off。

5. wifi 事件上报对照表

注意：按事件名称升序排列

事件	含义
WIFIMG_CMD_OR_PARAMS_ERROR	命令或者参数错误，此时函数返回为-1
WIFIMG_CONNECT_TIMEOUT	连接超时(未分配到 ip 地址)
WIFIMG_DEV_BUSING_EVENT	设备忙，此时函数返回值为-1
WIFIMG_KEY_MGMT_NOT_SUPPORT	加密类型不支持，此时函数返回值为-1
WIFIMG_NETWORK_CONNECTED	连接上 AP
WIFIMG_NETWORK_DISCONNECTED	断开连接
WIFIMG_NETWORK_NOT_EXIST	网络不存在，此时函数返回值为-1
WIFIMG_NO_NETWORK_CONNECTING	Wi-Fi 打开过程中没有连接任何 ap
WIFIMG_OPT_NO_USE_EVENT	操作无效，此时函数返回值为-1
WIFIMG_PASSWORD_FAILED	连接过程中密码出错
WIFIMG_WIFI_OFF_FAILED	WiFi 关闭失败
WIFIMG_WIFI_OFF_SUCCESS	WiFi 关闭成功
WIFIMG_WIFI_ON_FAILED	Wi-Fi 打开失败
WIFIMG_WIFI_ON_SUCCESS	Wi-Fi 打开成功

6. Demo

6.1. wifi 打开和关闭

```
#define TAG "wifi"
#include <tina_log.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <wifi_intf.h>
#include <pthread.h>

#define WIFI_ON_OFF_TEST_CNTS    1000

static int event = WIFIMG_NETWORK_DISCONNECTED;

static void wifi_event_handle(tWIFI_EVENT wifi_event, void *buf, int event_label)
{
    printf("event_label 0x%x\n", event_label);
    switch(wifi_event)
    {
        case WIFIMG_WIFI_ON_SUCCESS:
        {
            printf("WiFi on success!\n");
            event = WIFIMG_WIFI_ON_SUCCESS;
            break;
        }

        case WIFIMG_WIFI_ON_FAILED:
        {
            printf("WiFi on failed!\n");
            event = WIFIMG_WIFI_ON_FAILED;
            break;
        }

        case WIFIMG_WIFI_OFF_FAILED:
        {
            printf("wifi off failed!\n");
            event = WIFIMG_WIFI_OFF_FAILED;
            break;
        }

        case WIFIMG_WIFI_OFF_SUCCESS:
        {
            printf("wifi off success!\n");
            event = WIFIMG_WIFI_OFF_SUCCESS;
            break;
        }

        case WIFIMG_NETWORK_CONNECTED:
        {
            printf("WiFi connected ap!\n");
            event = WIFIMG_NETWORK_CONNECTED;
```

```
        break;
    }

    case WIFIMG_NETWORK_DISCONNECTED:
    {
        printf("WiFi disconnected!\n");
        event = WIFIMG_NETWORK_DISCONNECTED;
        break;
    }

    case WIFIMG_PASSWORD_FAILED:
    {
        printf("Password authentication failed!\n");
        event = WIFIMG_PASSWORD_FAILED;
        break;
    }

    case WIFIMG_CONNECT_TIMEOUT:
    {
        printf("Connected timeout!\n");
        event = WIFIMG_CONNECT_TIMEOUT;
        break;
    }

    case WIFIMG_NO_NETWORK_CONNECTING:
    {
       printf("It has no wifi auto connect when wifi on!\n");
       event = WIFIMG_NO_NETWORK_CONNECTING;
       break;
    }

    case WIFIMG_CMD_OR_PARAMS_ERROR:
    {
        printf("cmd or params error!\n");
        event = WIFIMG_CMD_OR_PARAMS_ERROR;
        break;
    }

    case WIFIMG_KEY_MGMT_NOT_SUPPORT:
    {
        printf("key mgmt is not supported!\n");
        event = WIFIMG_KEY_MGMT_NOT_SUPPORT;
        break;
    }

    case WIFIMG_OPT_NO_USE_EVENT:
    {
        printf("operation no use!\n");
        event = WIFIMG_OPT_NO_USE_EVENT;
        break;
    }

    case WIFIMG_NETWORK_NOT_EXIST:
    {
        printf("network not exist!\n");
        event = WIFIMG_NETWORK_NOT_EXIST;
        break;
```

```
            }

            case WIFIMG_DEV_BUSING_EVENT:
            {
                printf("wifi device busing!\n");
                event = WIFIMG_DEV_BUSING_EVENT;
                break;
            }

            default:
            {
                printf("Other event, no care!\n");
            }
        }
}

/*
 *argc[1]     ap ssid
 *argc[2]     ap passwd
*/
int main(int argv, char *argc[]){
    int ret = 0;
    int i = 0, event_label = 0;

    const aw_wifi_interface_t *p_wifi_interface = NULL;

    event_label=rand();
    for(i=0; i<WIFI_ON_OFF_TEST_CNTS;i++)
    {
        printf("\n**************************\n");
        printf("Do wifi on off test %d times\n", i);
        printf("**************************\n");

        event_label++;
        /* turn on wifi */
        p_wifi_interface = aw_wifi_on(wifi_event_handle, event_label);
        if(p_wifi_interface == NULL)
        {
        printf("Test failed: wifi on failed with event 0x%x\n", event);
            return -1;
        }

        /* test */
        usleep(10000);

        /* turn off wifi */
        ret = aw_wifi_off(p_wifi_interface);
        if(ret < 0)
        {
            printf("Test failed: wifi off error!\n");
            return -1;
        }

        /* wait event */
        while(event != WIFIMG_WIFI_OFF_SUCCESS)
        {
            usleep(200);
```

```
            }
        }

        printf("********************************\n");
        printf("Test completed: Success!\n");
        printf("********************************\n");

        return 0;
}
```

6.2. wifi 连接和断开

```
#define TAG "wifi"
#include <tina_log.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <wifi_intf.h>
#include <pthread.h>
static pthread_t    app_scan_tid;
static int event = WIFIMG_NETWORK_DISCONNECTED;

static void wifi_event_handle(tWIFI_EVENT wifi_event, void *buf, int event_label)
{
    printf("event_label 0x%x\n", event_label);
    switch(wifi_event)
    {
        case WIFIMG_WIFI_ON_SUCCESS:
        {
            printf("WiFi on success!\n");
            event = WIFIMG_WIFI_ON_SUCCESS;
            break;
        }

        case WIFIMG_WIFI_ON_FAILED:
        {
            printf("WiFi on failed!\n");
            event = WIFIMG_WIFI_ON_FAILED;
            break;
        }

        case WIFIMG_WIFI_OFF_FAILED:
        {
            printf("wifi off failed!\n");
            event = WIFIMG_WIFI_OFF_FAILED;
            break;
        }

        case WIFIMG_WIFI_OFF_SUCCESS:
        {
            printf("wifi off success!\n");
            event = WIFIMG_WIFI_OFF_SUCCESS;
            break;
        }

        case WIFIMG_NETWORK_CONNECTED:
        {
            printf("WiFi connected ap!\n");
            event = WIFIMG_NETWORK_CONNECTED;
            break;
        }

        case WIFIMG_NETWORK_DISCONNECTED:
        {
```

```
        printf("WiFi disconnected!\n");
        event = WIFIMG_NETWORK_DISCONNECTED;
        break;
    }

    case WIFIMG_PASSWORD_FAILED:
    {
        printf("Password authentication failed!\n");
        event = WIFIMG_PASSWORD_FAILED;
        break;
    }

    case WIFIMG_CONNECT_TIMEOUT:
    {
        printf("Connected timeout!\n");
        event = WIFIMG_CONNECT_TIMEOUT;
        break;
    }

    case WIFIMG_NO_NETWORK_CONNECTING:
    {
        printf("It has no wifi auto connect when wifi on!\n");
        event = WIFIMG_NO_NETWORK_CONNECTING;
        break;
    }

    case WIFIMG_CMD_OR_PARAMS_ERROR:
    {
        printf("cmd or params error!\n");
        event = WIFIMG_CMD_OR_PARAMS_ERROR;
        break;
    }

    case WIFIMG_KEY_MGMT_NOT_SUPPORT:
    {
        printf("key mgmt is not supported!\n");
        event = WIFIMG_KEY_MGMT_NOT_SUPPORT;
        break;
    }

    case WIFIMG_OPT_NO_USE_EVENT:
    {
        printf("operation no use!\n");
        event = WIFIMG_OPT_NO_USE_EVENT;
        break;
    }

    case WIFIMG_NETWORK_NOT_EXIST:
    {
        printf("network not exist!\n");
        event = WIFIMG_NETWORK_NOT_EXIST;
        break;
    }

    case WIFIMG_DEV_BUSING_EVENT:
    {
        printf("wifi device busing!\n");
```

```
                event = WIFIMG_DEV_BUSING_EVENT;
                break;
            }

            default:
            {
                printf("Other event, no care!\n");
            }
        }
}
void *app_scan_task(void *args)
{
    const aw_wifi_interface_t *p_wifi = (aw_wifi_interface_t *)args;
    char scan_results[4096];
    int len = 0;
    int event_label = 0;

    while(1){
        event_label++;
        p_wifi->start_scan(event_label);
        len = 4096;
        p_wifi->get_scan_results(scan_results, &len);
    }
}
/*
 *argc[1]     ap ssid
 *argc[2]     ap passwd
*/
int main(int argv, char *argc[]){
    int ret = 0, len = 0;
    int times = 0, event_label = 0;;
    char ssid[256] = {0}, scan_results[4096] = {0};
    int    wifi_state = WIFIMG_WIFI_DISABLED;
    const aw_wifi_interface_t *p_wifi_interface = NULL;

    printf("\n**********************************\n");
    printf("***Start wifi connect ap test!***\n");
    printf("**********************************\n");

    event_label = rand();
    p_wifi_interface = aw_wifi_on(wifi_event_handle, event_label);
    if(p_wifi_interface == NULL){
        printf("wifi on failed event 0x%x\n", event);
        return -1;
    }

    while(aw_wifi_get_wifi_state() == WIFIMG_WIFI_BUSING){
        printf("wifi state busing,waiting\n");
        usleep(2000000);
    }

    //pthread_create(&app_scan_tid, NULL, &app_scan_task,(void *)p_wifi_interface);

    event_label++;
    p_wifi_interface->connect_ap(argc[1], argc[2], event_label);

    while(aw_wifi_get_wifi_state() == WIFIMG_WIFI_BUSING){
```

```
            printf("wifi state busing,waiting\n");
            usleep(2000000);
        }

        printf("*****************************\n");
        printf("Wifi connect ap test: Success!\n");
        printf("*****************************\n");

        return 0;
}
```

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 acknowledgment 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.