XR871 Wi-Fi Developer Guide

Revision 1.0

May 9, 2017

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE INFORMATION FURNISHED BY XRADIO IS BELIEVED TO BE ACCURATE AND RELIABLE. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE. XRADIO 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 XRADIO. THIS DATASHEET NEITHER STATES NOR IMPLIES WARRANTY OF ANY KIND, INCLUDING FITNESS FOR ANY PARTICULAR APPLICATION.

THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Revision History

Version	Data	Summary of Changes
1.0	2017-5-9	Initial Version
	2017-7-25	增加 Station 功能描述
	2017-8-7	增加 AP 功能描述

Table 1-1 Revision History

Contents

Tables

Figures

1 概述

XR871 系列 IC 是基于 Wi-Fi 技术的无线 MCU，其内置 Wi-Fi 协议栈，并于 SDK 中集成 LwIP，因此可以应用于物联网中的各类 Wi-Fi 联网设备。此文档用以说明 XR871 的 SDK 中的 WLAN 的功能描述和使用方法，进而指导网络应用开发者能够有效的使用正确的 API 完成应用功能实现。

此文档将指引你实现以下功能指导：

- 初始化 Wi-Fi 系统进入 STA 模式或者 AP 模式
- 配置 STA 或 AP 模式下的 Wi-Fi 系统参数
- 扫描，连接可用是 AP 节点
- 创建 socket，使用上层网络协议

1.1 网络系统架构

XR871 软件系统架构分为四层结构：驱动层，OSAL 层，服务层及应用层，如下图所示。

- 驱动层：提供芯片级功能访问 API，提供外设数据传输通道，实现完整的设备管理。
- OSAL 层：提供操作系统抽象，对接各操作系统内核。
- 服务层：提供系统接口，功耗管理，网络协议实现等，WLAN 系统及网络相关内容都属于服务层。
- 应用层：提供应用程序实现，网络配置管理，产品集成等。

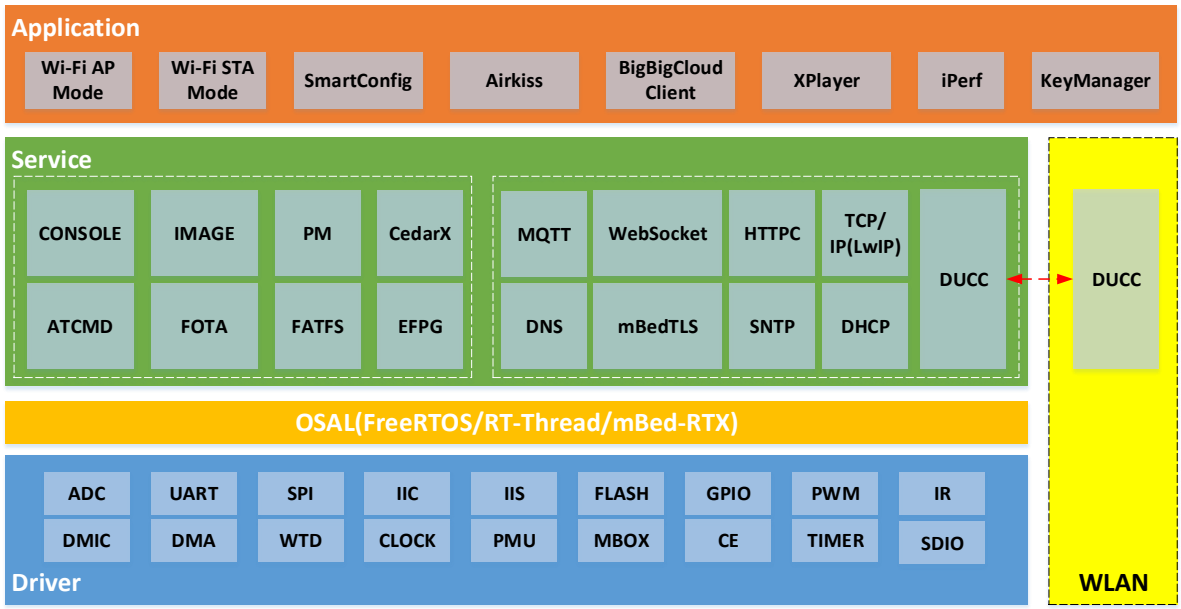

图 1-1 软件架构框图

1.2 网络通信实现描述

XR871 硬件实现 Application 子系统和 WLAN 子系统物理隔离的架构，从而为应用提供更加灵活稳定的系统运行环境，也为开发者提供了更加丰富的系统资源。在此系统中，应用子系统与网络子系统通过 MailBox 进行通信，其实现架构如下图。

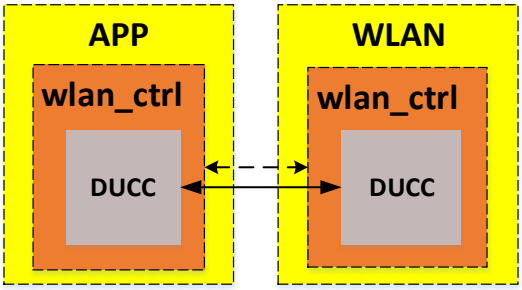

图 1-2 网络系统通信示意图

该系统在 MailBox 的基础上实现了一套通用的传输协议 DUCC，通过 DUCC 命令实现对网络子系统的基本访问，并通过 DUCC 命令之一 WLAN_WPA_CTRL_REQUEST 实现 supplicant 功能扩展中，具体命令如下：

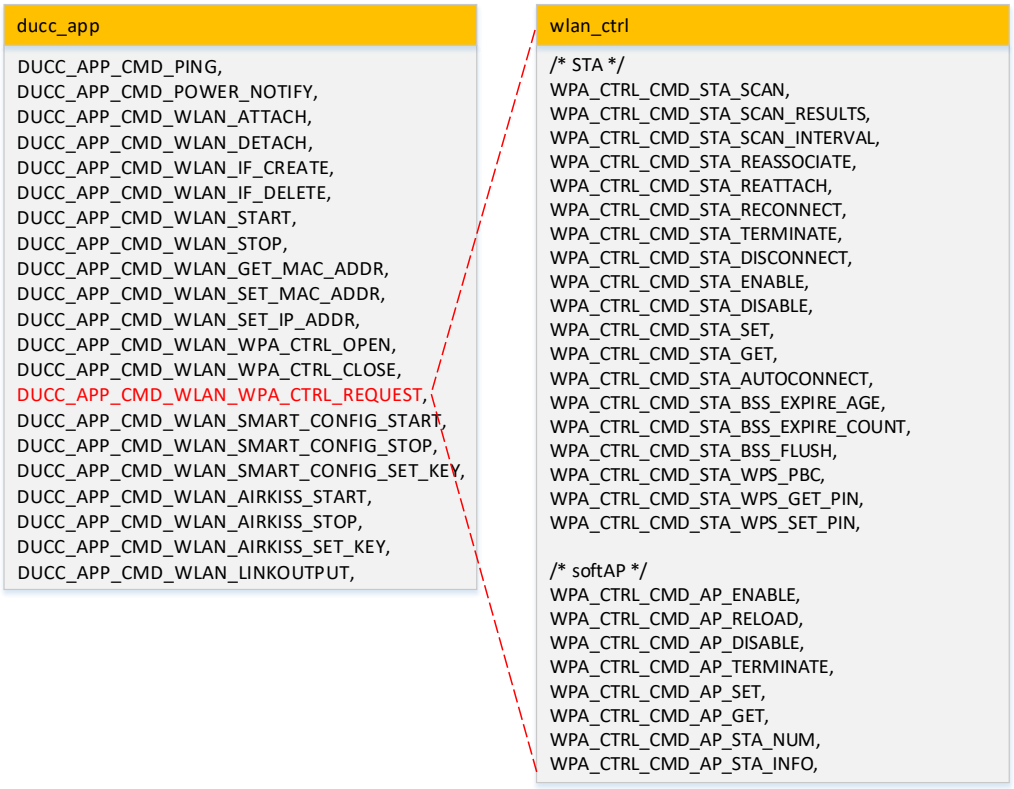

图 1-3 网络通信命令集

相关文件请参考：

sdk-code/include/sys/ducc sdk-code/src/sys/ducc

sdk-code/include/net/wlan sdk-code/src/net/wlan

2 WLAN 功能描述

2.1 示例工程

请参考 sdk-code/project/wlan_demo。

2.2 WLAN 系统启动

在 wlan_demo 工程中，网络系统加载流程如下：

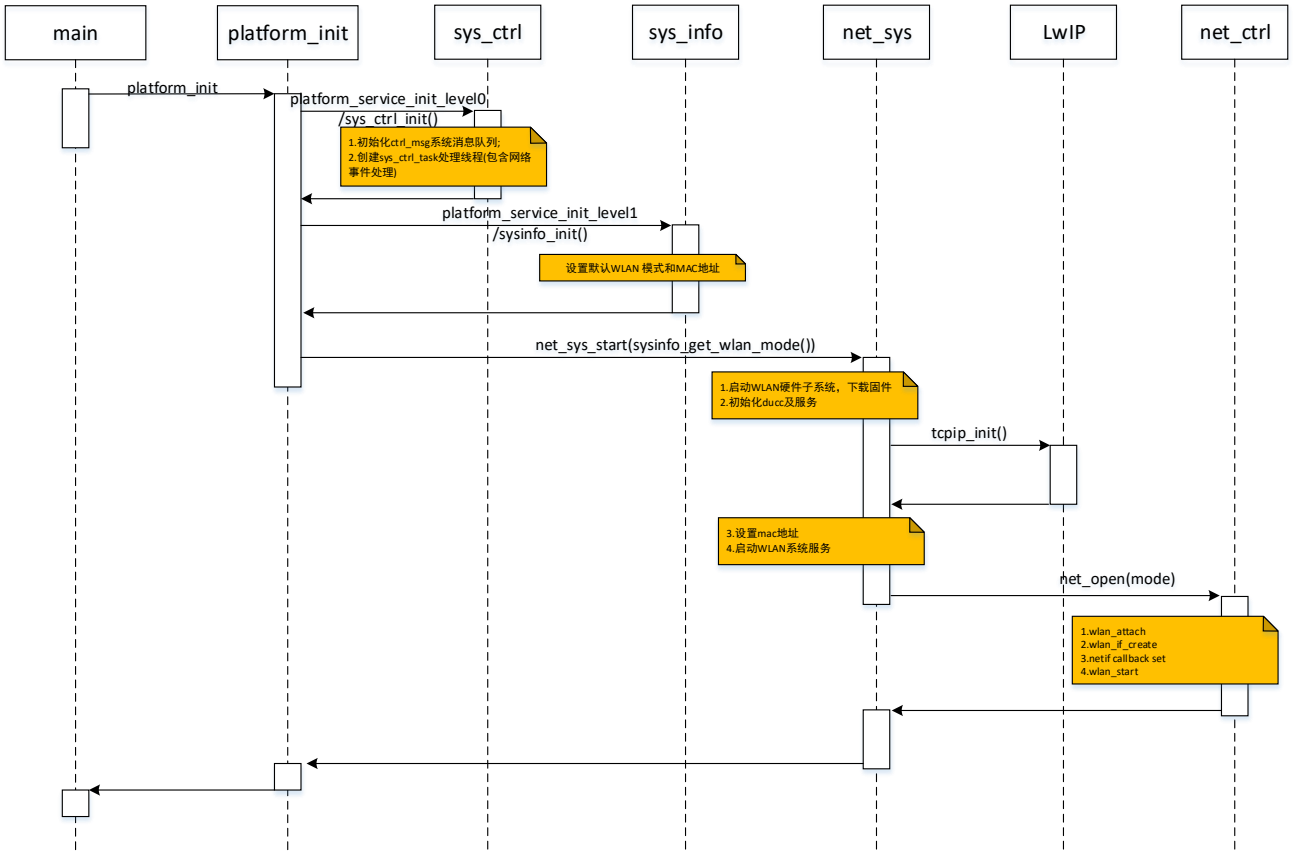

图 2-1 网络系统启动流程

网络系统加载过程中依次完成了以下几个重要的过程：

1. 初始化系统消息队列，创建系统消息处理线程；
2. 配置网络子系统硬件，加载网络子系统固件；
3. 初始化 DUCC 服务，注册回调函数；
4. 初始化 TCP/IP 协议栈；
5. 创建 netif 网络接口并启动 WLAN 子系统任务；

2.3 Station 模式操作

2.3.1 初始化 STA 模式

上一章节已在网络启动流程说明通过调用 net_sys_start(mode)的方式来启动网络子系统，并初始化到相关的模式，下面通过示例说明配置 Station 模式下的网络参数。

1. 连接一个工作在开放模式的 AP，其 SSID 为 TEST_AP

   - 方式一，通过傻瓜匹配

```
#include "net/wlan/wlan.h"

wlan_sta_set("\"TEST_AP\"", NULL); #注意引号使用
wlan_sta_enable();
```

   - 方式二，通过精确配置

```
include "net/wlan/wlan.h"

wlan_sta_config_t config = {0};
char* ssid = "\"TEST_AP\"";

/* ssid */
memset(&config, 0, sizeof(config));
config.field = WLAN_STA_FIELD_SSID;
strlcpy((char *)config.u.ssid, ssid, sizeof(config.u.ssid));
if (wlan_sta_set_config(&config) != 0) {
    NET_WARN("set ssid failed\n");
    return -1;
}

/* auth_alg: OPEN */
memset(&config, 0, sizeof(config));
config.field = WLAN_STA_FIELD_AUTH_ALG;
config.u.auth_alg = WPA_AUTH_ALG_OPEN;
if (wlan_sta_set_config(&config) != 0)
    return -1;

memset(&config, 0, sizeof(config));
config.field = WLAN_STA_FIELD_KEY_MGMT;
config.u.key_mgmt = WPA_KEY_MGMT_NONE;
if (wlan_sta_set_config(&config) != 0) {
    NET_WARN("set key_mgmt failed\n");
    return -1;
}

if (wlan_sta_enable()!= 0) {
    NET_WARN("enable sta failed\n");
    return -1;
}
```

2. 连接工作在 WEP 模式的 AP，SSID 为 TEST_AP，WEP_KEY0 为 1234567890

```
#include "net/wlan/wlan.h"
```



```
wlan_sta_config_t config = {0};
char* ssid = "\"TEST_AP\"";
char* key = "\"1234567890\"";

/* ssid */
config.field = WLAN_STA_FIELD_SSID;
strlcpy((char *)config.u.ssid, ssid, sizeof(config.u.ssid));
if (wlan_sta_set_config(&config) != 0) {
    NET_WARN("set ssid failed\n");
    return -1;
}

/* auth_alg: OPEN */
config.field = WLAN_STA_FIELD_AUTH_ALG;
config.u.auth_alg = WPA_AUTH_ALG_SHARED;
if (wlan_sta_set_config(&config) != 0)
    return -1;

config.field = WLAN_STA_FIELD_WEP_KEY_INDEX;
config.u.wep_tx_keyidx = 0;
if (wlan_sta_set_config(&config) != 0) {
    NET_WARN("set key_mgmt failed\n");
    return -1;
}

config.field = WLAN_STA_FIELD_WEP_KEY0;
strlcpy((char *)config.u.wep_key, key, sizeof(config.u.wep_key));
if (wlan_sta_set_config(&config) != 0) {
    NET_WARN("set wep key0 failed\n");
    return -1;
}

if (wlan_sta_enable()!= 0) {
    NET_WARN("enable sta failed\n");
    return -1;
}
```

3. 连接工作在 WPA 模式的 AP，SSID 为 TEST_AP，PSK 为 12345678

- 方式一，通过傻瓜匹配

```
#include "net/wlan/wlan.h"

wlan_sta_set("\"TEST_AP\"", "\"12345678\""); #注意引号使用
wlan_sta_enable();
```

- 方式二，通过精确配置

```
#include "net/wlan/wlan.h"

wlan_sta_config_t config = {0};
char* ssid = "\"TEST_AP\"";
char* key = "\"12345678\"";

/* ssid */
config.field = WLAN_STA_FIELD_SSID;
strlcpy((char *)config.u.ssid, ssid, sizeof(config.u.ssid));
if (wlan_sta_set_config(&config) != 0) {
```



```
        NET_WARN("set ssid failed\n");
        return -1;
    }

    /* auth_alg: OPEN */
    config.field = WLAN_STA_FIELD_AUTH_ALG;
    config.u.auth_alg = WPA_AUTH_ALG_OPEN;
    if (wlan_sta_set_config(&config) != 0)
        return -1;

    config.field = WLAN_STA_FIELD_PSK;
    strlcpy((char *)config.u.psk, (char *)psk, sizeof(config.u.psk));
    if (wlan_sta_set_config(&config) != 0) {
        NET_WARN("set psk failed\n");
        return -1;
    }

    /* proto: WPA | RSN */
    config.field = WLAN_STA_FIELD_PROTO;
    config.u.proto = WPA_PROTO_WPA | WPA_PROTO_RSN;
    if (wlan_sta_set_config(&config) != 0)
        return -1;

    /* key_mgmt: PSK */
    config.field = WLAN_STA_FIELD_KEY_MGMT;
    config.u.key_mgmt = WPA_KEY_MGMT_PSK;
    if (wlan_sta_set_config(&config) != 0)
        return -1;

    /* pairwise: CCMP | TKIP */
    config.field = WLAN_STA_FIELD_PAIRWISE_CIPHER;
    config.u.pairwise_cipher = WPA_CIPHER_CCMP | WPA_CIPHER_TKIP;
    if (wlan_sta_set_config(&config) != 0)
        return -1;

    /* group: CCMP | TKIP | WEP40 | WEP104 */
    config.field = WLAN_STA_FIELD_GROUP_CIPHER;
    config.u.pairwise_cipher = WPA_CIPHER_CCMP | WPA_CIPHER_TKIP
                | WPA_CIPHER_WEP40 | WPA_CIPHER_WEP104;
    if (wlan_sta_set_config(&config) != 0)
        return -1;

    if (wlan_sta_enable()!= 0) {
        NET_WARN("enable sta failed\n");
        return -1;
    }
```

4. 连接隐藏 SSID 的 AP，其工作在 WPA 模式，SSID 为 TEST_AP，PSK 为 12345678

当 AP 的 SSID 隐藏时，在配置完相关参数需要明确的指明扫描此 SSID，如下面代码所示：

- 方式一，通过傻瓜匹配

```
#include "net/wlan/wlan.h"

wlan_sta_set("\"TEST_AP\"", "\"12345678\""); #注意引号使用
wlan_sta_enable();
```

- 方式二，通过精确配置


```
#include "net/wlan/wlan.h"

wlan_sta_config_t config = {0};
char* ssid = "\"TEST_AP\"";
char* key = "\"12345678\"";

/* ssid */
config.field = WLAN_STA_FIELD_SSID;
strlcpy((char *)config.u.ssid, ssid, sizeof(config.u.ssid));
if (wlan_sta_set_config(&config) != 0) {
    NET_WARN("set ssid failed\n");
    return -1;
}

/* auth_alg: OPEN */
…

/* psk:*/
…

/* proto: WPA | RSN */
…

/* key_mgmt: PSK */
…

/* pairwise: CCMP | TKIP */
…

/* group: CCMP | TKIP | WEP40 | WEP104 */
…

/* scan_ssid: 1 */
config.field = WLAN_STA_FIELD_SCAN_SSID;
config.u.scan_ssid = 1;
if (wlan_sta_set_config(&config) != 0)
    return -1;

if (wlan_sta_enable()!= 0) {
    NET_WARN("enable sta failed\n");
    return -1;
}
```

2.3.2 获取已设置的配置参数

当设置完 WLAN 相关参数后，可以通过 net_sta_config_get(&config)函数来获取所对应的配置项，在 get 配置项之前需要设置所要获取的配置项的类别，示例如下：

```
#include "net/wlan/wlan.h"

wlan_sta_config_t config = {0};
config.field = WLAN_STA_FIELD_SSID;
if (wlan_sta_get_config(&config) != 0) {
    NET_ERR("%s: get config failed\n", __func__);
    return -1;
```



```
    }
    NET_MSG("ssid: %s\n", config.u.ssid);
```

2.3.3 扫描可用的 AP 列表

扫描 AP 列表并获取结果可以分别使用 wlan_sta_scan_once()和 wlan_sta_scan_result(&result)来实现，在获取结果之前需要为扫描结果分配空间，并指定希望得到的 AP 数目，那么返回结果将优先提供信号最好的 AP 节点信息，AP 节点信息结构 wlan_sta_scan_ap_t 定义如下：

```
    typedef struct wlan_sta_scan_ap {
        uint8_t bssid[6];
        uint8_t ssid[65];
        int    freq;
        int    level;
        int    wpa_flags;
        int    wpa_cipher;
        int    wpa_key_mgmt;
        int    wpa2_cipher;
        int    wpa2_key_mgmt;
    }wlan_sta_scan_ap_t;
```

SCAN 操作示例如下：

- 设置一次 scan

```
    #include "net/wlan/wlan.h"

    wlan_sta_scan_once();
```

- 获取 scan 结果，以获取 10 个 AP 节点为例

```
    #include "net/wlan/wlan.h"

    wlan_sta_scan_results_t results = {0};
    results.ap = (wlan_sta_scan_ap_t *)malloc(size *
                                              sizeof(wlan_sta_scan_ap_t));
    if (results.ap == NULL) {
        NET_ERR("%s: malloc failed\n", __func__);
        return -1;
    }
    results.size = 10; // 获取 10 个结果信息
    ret = wlan_sta_scan_result(&results);
    if (ret == 0) {
        ...
    }
    free(results.ap);
```

2.3.4 通过 WPS 连接 AP

该系统支持通过 WPS 方式连接目标 AP，并支持 PBC(BUTTON)模式和 PIN 码模式连接，其对应的 API 如下：

```
    int wlan_sta_wps_pbc(void);
    int wlan_sta_wps_pin_get(wlan_sta_wps_pin_t *wps);
    int wlan_sta_wps_pin_set(wlan_sta_wps_pin_t *wps);
```


2.4 AP 模式操作

2.4.1 启动一个 AP 节点

上一章节讲述了通过调用 net_sys_start(mode)的方式来启动工作在 Station 模式下的网络子系统，同样的使用 net_sys_start(mode)并设置 mode 为 WLAN_MODE_HOSTAP 即可将 WLAN 设置为 AP 模式，如果系统先启动在 STA 模式然后切换到 HOSTAP 模式，需要先 stop 当前网络，如下：

```
net_sys_stop();
net_sys_start(WLAN_MODE_HOSTAP);
```

AP 模式启动之后需要通过一系列操作配置 AP 模式的网络参数，下面一一说明。

1. 配置工作在 OPEN 模式的 AP，其 SSID 为 TEST_AP，注意系统默认 AP 为 AP-XR871，修改时先 disable 再 enable 完成配置更新。

```
wlan_ap_set("\"TEST_AP\"", NULL);
wlan_ap_disable();
wlan_ap_enable();
```

2. 设置工作在加密模式的 AP，其 SSID 为 TEST_AP，PSK 为 12345678（此处只选择设置最通用的模式，WEP 模式不推荐不做描述说明）

```
wlan_ap_set("\"TEST_AP\"", "\"12345678\"");
wlan_ap_disable();
wlan_ap_enable();
```

2.4.2 Set/Get AP 节点配置

可以通过 wlan_ap_set_config()接口和 wlan_ap_get_config()来设置或获取相关参数，在 SET/GET 的调用中，通过制定 wlan_ap_config 的 field 参数来指定所要设置或者获取的参数值，目前能够配置的参数类型如下：

```
typedef enum wlan_ap_field {
    WLAN_AP_FIELD_SSID = 0,
    WLAN_AP_FIELD_PSK,
    WLAN_AP_FIELD_KEY_MGMT,
    WLAN_AP_FIELD_WPA_CIPHER,
    WLAN_AP_FIELD_RSN_CIPHER,
    WLAN_AP_FIELD_PROTO,
    WLAN_AP_FIELD_AUTH_ALG,
    WLAN_AP_FIELD_GROUP_REKEY,
    WLAN_AP_FIELD_STRICT_REKEY,
    WLAN_AP_FIELD_GMK_REKEY,
    WLAN_AP_FIELD_PTK_REKEY,
    WLAN_AP_FIELD_HW_MODE,
    WLAN_AP_FIELD_IEEE80211N,
    WLAN_AP_FIELD_CHANNEL,
    WLAN_AP_FIELD_BEACON_INT,
    WLAN_AP_FIELD_DTIM,
    WLAN_AP_FIELD_MAX_NUM_STA,

    WLAN_AP_FIELD_NUM,
};
```


此处以 SET/GET channel 为例说明如下：

1. Set Channel

```
    wlan_ap_config_t config = {0};
    config.field = WLAN_AP_FIELD_CHANNEL;
    config.u.channel = 6;
    wlan_ap_set_config(&config);
```

2. Get Channel

```
    wlan_ap_config_t config = {0};
    config.field = WLAN_AP_FIELD_CHANNEL;
    wlan_ap_get_config(&config);
```

2.5 SmartConfig 使用

SmartConfig 机制让设备在没有输入接口的情况下得以连接到无线网络上，连接设备（通常是手机）通过广播所要连接的 AP 的加密的 SSID 和 Password，而设备端通过监听并解密信息来获得 SSID 和 Password，从而发起连接请求给指定的 AP，解密使用的 key 可以通过 API 调用进行设置。

该平台提供 SmartConfig 库及 Android APK 示例作为参考。SmartConfig 接口如下：

```
    int wlan_smart_config_start(struct netif *nif);
    int wlan_smart_config_stop(void);
    int wlan_smart_config_set_key(char *key);
```

在调用 wlan_smart_config_start 之后，设备监听 AP 的配置信息，一旦获取成功，WLAN 系统会返回 NET_CTRL_MSG_WLAN_SMART_CONFIG_RESULT 消息，在 net_ctrl.c 中处理如下：

```
static int net_ctrl_process_smart_config_result(struct wlan_smart_config_result
*result)
{
    wlan_sta_config_t config;
    memset(&config, 0, sizeof(config));

    if (!result->valid) {
        NET_DBG("invalid smart config result\n");
        return 0;
    }

    NET_DBG("smart config ssid: %s\n", result->ssid);
    NET_DBG("smart config psk: %s\n", result->psk);
    config.field = WLAN_STA_FIELD_SSID;
    strlcpy((char *)config.u.ssid, (char *)result->ssid, sizeof(config.u.ssid));
    if (wlan_sta_set_config(&config) != 0) {
        NET_WRN("set ssid failed\n");
        return -1;
    }

    config.field = WLAN_STA_FIELD_PSK;
    strlcpy((char *)config.u.psk, (char *)result->psk, sizeof(config.u.psk));
    if (wlan_sta_set_config(&config) != 0) {
        NET_WRN("set psk failed\n");
        return -1;
    }
```



```
    if (wlan_sta_enable()!= 0) {
        NET_WRN("enable sta failed\n");
        return -1;
    }

    return wlan_sta_connect();
}
```

在调用完 wlan_smart_config_start()之后，应用程序需要监控网络节点 NIF 的状态变化，如果网络节点状态变为 CONNECTED 或者获取到 IP 地址，则可以通过调用 wlan_smart_config_stop()来停止监听行为。

2.6 Airkiss 使用

Airkiss 是微信提供的配网模式，使用和 SmartConfig 类似，在此平台上，通过将原有微信的 API 进行整合封装，简化流程，相关的 API 如下：

```
    /* airkiss */
    int wlan_airkiss_start(struct netif *nif);
    int wlan_airkiss_stop(void);
    int wlan_airkiss_set_key(char *key);
    int wlan_airkiss_ack_start(struct wlan_smart_config_result *result, struct
netif *netif);
    void wlan_airkiss_online_ack_start();
    void wlan_airkiss_online_ack_stop();
```

