WLAN Framework Application Notes

RT-THREAD Document Center

WWW.RT-THREAD.ORG

Friday 14th December, 2018

Table of contents

1 Purpose and structure of this paper. 1
1.1 Purpose and background of this paper. 1
1.2 Structure of this paper. 1
2 Problem Statement 1
3 Problem Solved 1
3.1 Introduction to the WLAN Framework. 1
3.2 WLAN Framework Composition 2
3.3 WLAN Framework Functionality 2
3.4 WLAN framework configuration and initialization. 3
3.4.1. WLAN framework configuration. 3
3.4.2. WLAN device initialization. 5
3.5 WLAN usage. 5
3.5.1. Shell operation WiFi 5
3.5.1.1. WiFi Scanning. 5
3.5.1.2. WiFi connection. 6
3.5.1.3. WiFi is disconnected. 7
3.5.2. WiFi Scanning. 7
3.5.3. WiFi connection and disconnection. 8
3.5.4. Enable automatic reconnection on WiFi. 11
4 References. 13
4.1 API related to this article. 13
4.2 API List 13
4.3 Core API Detailed Explanation. 14
4.3.1. rt_wlan_set_mode() 14

4.4 Functions 14
4.5 Function prototypes 14
4.6 Function parameters 14
4.7 Return Value. . . . 14
4.7.1. rt_wlan_prot_attach() 15
4.8 Functions 15
4.9 Function prototypes 15
4.10 Function parameters 15
4.11 Return Value. . . . 15
4.11.1. rt_wlan_scan_sync() 15
4.12 Functions 15
4.13 Function prototypes 15
4.14 Function parameters 15
4.15 Return value. . . . 16
4.15.1. rt_wlan_connect() 16
4.16 Functions 16
4.17 Function prototypes 16
4.18 Function parameters 16
4.19 Return value. . . . 16
4.19.1. rt_wlan_disconnect() 17
4.20 Functions 17
4.21 Function prototypes 17
4.22 Function parameters 17
4.23 Return value. . . . 17
4.23.1. rt_wlan_config_autoreconnect() 17
4.24 Functions 17
4.25 Function prototypes 17
4.26 Function parameters 17
4.27 Return value. . . . 18

1 Purpose and structure of this paper

1.1 Purpose and Background of this Paper

With the rapid development of the Internet of Things (IoT), more and more embedded devices are equipped with Wi-Fi wireless networking equipment. To manage Wi-Fi network devices, RT-Thread introduces a WLAN device management framework. This framework provides numerous functions for controlling and managing Wi-Fi, providing developers with many convenient ways to use Wi-Fi devices.

This article will help developers learn how to use this framework to control and manage WIFI. It will introduce the relevant aspects of the WLAN framework from the perspectives of concepts, examples, etc. It provides relevant knowledge and demonstrates WIFI related functions through Shell and code, such as controlling WIFI scanning, connection, disconnection, etc.

1.2 Structure of this paper

- Introduction to WLAN framework
- WLAN framework configuration
- WLAN framework usage

2 Problem Description

This application note mainly introduces the WLAN framework around the following issues:

- What is the WLAN framework and what functions does it have?
- How to configure the WLAN framework?
- How to use the WLAN framework?

To solve the above problems, we need to understand the composition principle of WLAN framework and learn how to use various functions in WLAN framework. Gradually begin to introduce the composition of the WLAN framework and the use of related functions.

3. Problem Solving

3.1 Introduction to the WLAN Framework

The WLAN framework is a set of middleware developed by RT-Thread for managing WIFI. It connects to specific WIFI drivers at the bottom and controls WIFI connection, disconnection, scanning and other operations. It carries different applications at the top and provides WIFI control, events, data diversion and other operations for applications, providing a unified WIFI control interface for upper-level applications. The WLAN framework is mainly composed of three parts. The DEV driver interface layer provides a unified calling interface for the WLAN framework. The Manage management layer provides users with specific functions such as WIFI scanning, connection, disconnection and reconnection. The Protocol protocol is responsible for processing the data stream generated on WIFI, and can mount different communication protocols such as LWIP according to different usage scenarios. It has the characteristics of simple use, complete functions, easy docking and strong compatibility.

3.2 WLAN Framework Composition

The following figure is the structural framework of the WIFI framework

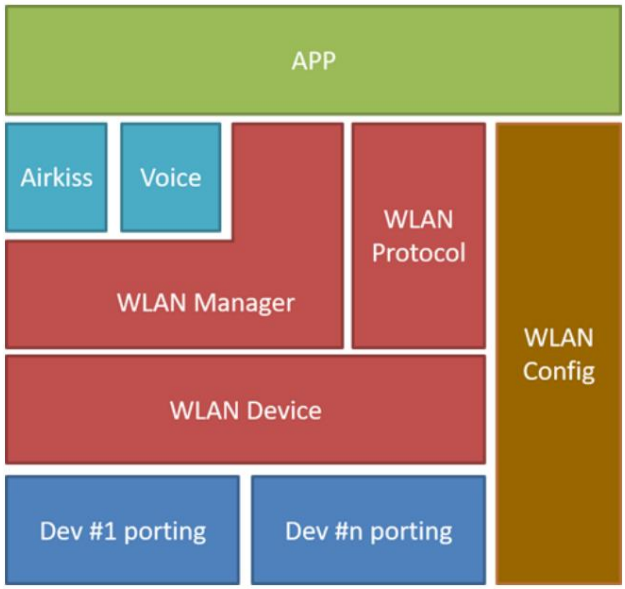

Figure 1: *Wi-Fi* frame

The first part, app, is the application layer, which is a specific application based on the WLAN framework, such as WiFi-related Shell commands.

The second part, airkiss and voice, is the network configuration layer, providing functions such as wireless network configuration and acoustic network configuration.

The third section, WLAN Manager, is the WLAN management layer. It controls and manages WLAN devices. It offers WLAN control functions such as setting the mode, connecting to a hotspot, disconnecting from a hotspot, starting a hotspot, and scanning for hotspots. It also provides management features such as disconnection reconnection and automatic hotspot switching.

The fourth part, WLAN protocol, is the protocol layer. It submits the data stream to a specific protocol for parsing. Users can specify different protocols for communication.

The fifth part, WLAN config, is the parameter management layer. It manages the hotspot information and passwords that are successfully connected and writes them to non-volatile storage media. middle.

The sixth part, WLAN dev, is the driver interface layer, which connects to specific WLAN hardware and provides a unified calling interface for the management layer.

3.3 WLAN Framework Functions

- Auto-connect: When auto-connect is enabled, any time your Wi-Fi connection is disconnected, it automatically reads the hotspot information of the previously connected connection and connects to it. If a hotspot connection fails, it switches to the next hotspot and continues to connect until a connection is successful. Hotspots used for automatic connection are tried in the order in which they were successfully connected, with the most recently connected hotspot being prioritized. After a successful connection, the hotspot is cached and used first the next time the connection is disconnected.

- Parameter storage stores Wi-Fi parameters after a successful connection. A copy of these parameters is cached in memory. If an external non-volatile storage interface is configured, a copy is stored in external storage media. Users can implement **the struct** rt_wlan_cfg_ops structure to store parameters anywhere they choose. The cached parameters primarily provide hotspot information for automatic connection. When the Wi-Fi connection is not connected, the cached parameters are read to attempt a connection.

- WIFI control provides a complete WIFI control interface, including scanning, connection, hotspot, etc. Provides WIFI related status callback events, disconnection, Connection, connection failure, etc. Provide users with a simple and easy-to-use WIFI management interface.
- Shell command can be used to input commands in Msh to control WIFI to perform scan, connect, disconnect and other actions. Print WIFI status and other debugging information interest.

3.4 WLAN Framework Configuration and Initialization

This article will be based on the Zhengdian Atom STML4 IOT board development board, which has an AP6181 WiFi chip onboard and WiFi driver Already implemented. Suitable for learning WLAN management framework.

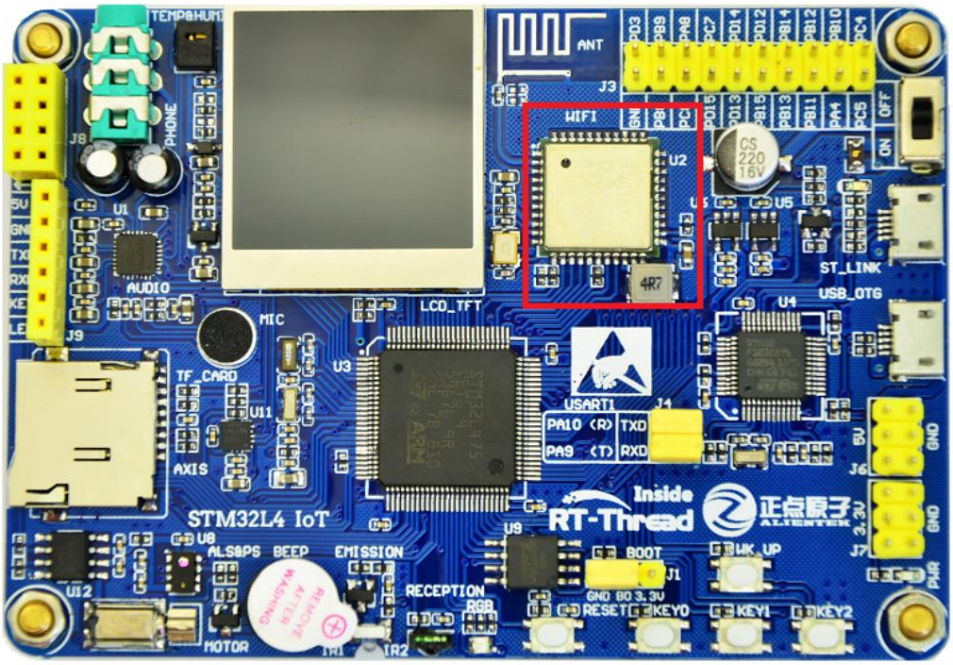

Figure 2: *WIFI* frame

WLAN framework configuration mainly includes the following aspects

- Enable the WLAN framework and configure
- Initialize the WLAN device and specify the protocol to use

3.4.1. WLAN Framework Configuration

Here we will introduce the configuration related to the WIFI framework. Use the env tool to enter the IOT board directory and enter in the env command line menuconfig command to open the graphical configuration interface

- In the menuconfig configuration interface, select RT-Thread Components -> Device Drivers -> Using WiFi - > As shown in the figure below

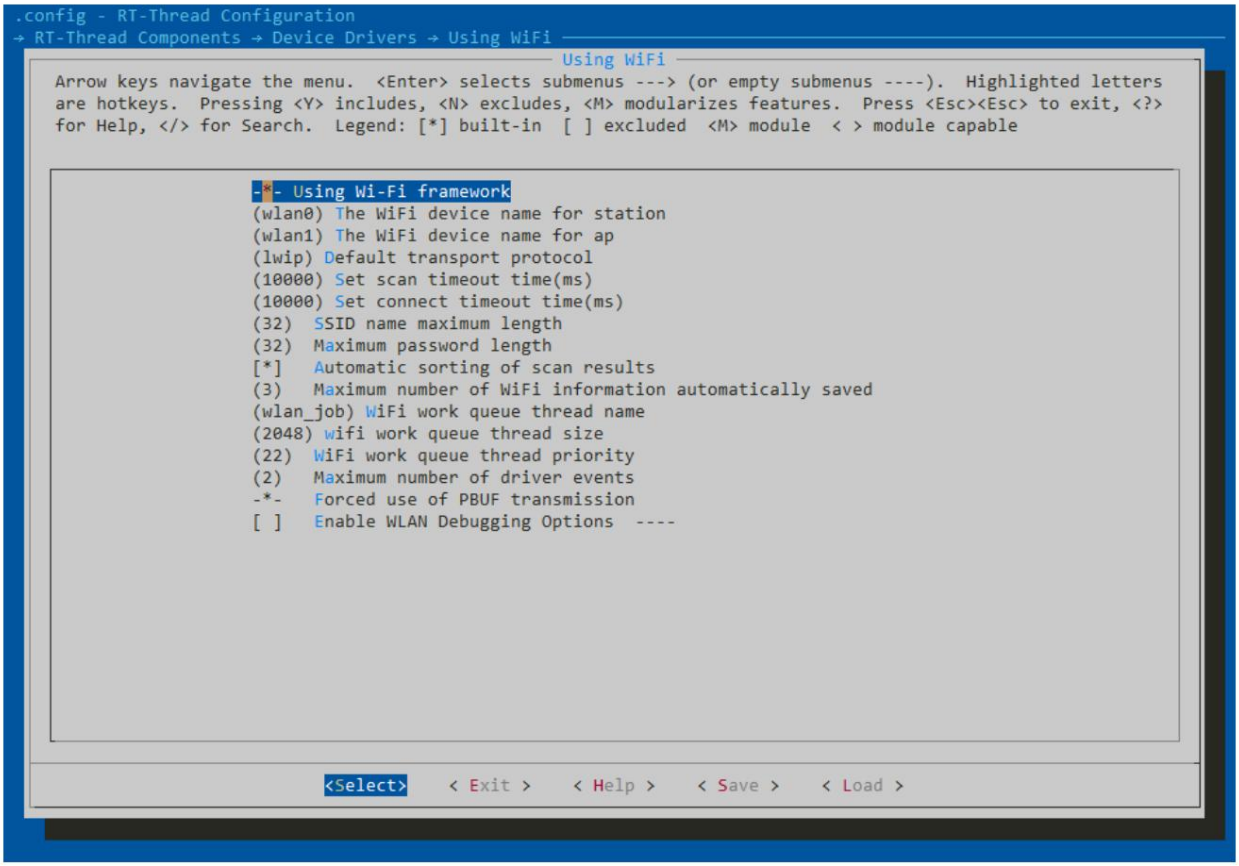

Figure 3: *WLAN* Configuration

These configuration items are described below.

- Using Wi-Fi framework: Using WLAN management framework
- the WiFi device name for station: Station device default name
- the WiFi device name for ap: ap device default name
- Default transport protocol: default protocol
- Scan timeout time: Scan result timeout time
- connect timeout time: Connection timeout time
- SSID name maximum length: SSID maximum length
- Maximum password length: Maximum password length
- Automatic sorting of scan results: Automatic sorting of scan results
- Maximum number of WiFi information automatically saved: Maximum number of entries automatically saved
- WiFi work queue thread name: WiFi background thread name
- wifi work queue thread size: WIFI background thread stack size
- WiFi work queue thread priority: WiFi background thread priority

- Maximum number of driver events: The maximum number of events registered at the dev layer
- Forced use of PBUF transmission: Forced use of PBUF to exchange data
- Enable WLAN Debugging Options: Enable debug log

After configuring as shown above, save and exit

3.4.2. WLAN Device Initialization

The WLAN framework needs to specify a working mode for initializing the WLAN device. This needs to be implemented in code, so the following code is needed Initialize. The code is as follows:

```
int wifi_init(void)
{
    rt_wlan_set_mode(RT_WLAN_DEVICE_STA_NAME, RT_WLAN_STATION);    //Configure WLAN device
        Working Mode
    return 0;
}
```

After writing the above code, be sure to call and execute it once, and then you can use the WLAN framework to manage the device.

3.5 WLAN Usage

Note: Before performing the following operations, you must first perform WLAN device initialization. For details, see the WLAN device initialization section .

3.5.1. Shell WiFi Operation

Using shell commands can help us quickly debug WiFi related functions. Just enter the corresponding command in msh and WiFi will be Execute the corresponding action. The following three commands will show how to operate WiFi using shell commands.

The wifi-related shell commands are as follows:

```
Wi-Fi                                   : Print Help
wifi help wifi                          : View help
join SSID [PASSWORD] wifi ap            ÿ Connect to wifi, SSDI is empty, use configuration to automatically connect
SSID [PASSWORD]                         : Create a hotspot
Wi-Fi Scan                              ÿ Scan all hotspots
wifi disc                               : Disconnect
wifi ap_stop                            : Stop hotspot
Wi-Fi status                            ÿ Print wifi status sta + ap
wifi smartconfig                        ÿ Start the network configuration function
```

3.5.1.1. WiFi Scan After executing the WiFi Scan command, the surrounding hotspot information will be printed on the terminal. Through the printed hotspot information, You can see multiple attributes such as SSID, MAC address, etc.

- The wifi scan command format is as follows

```
Wi-Fi Scan
```

Command Description

Field	describe
Wi-Fi	All wifi commands start with wifi
scan	Wi-Fi scan action

Enter this command in msh to perform wifi command scanning. The scanning results are shown in the figure below.

```
Wi-Fi Scan
SSID                             MAC            security     rssi chn Mbps
--------------------------------
rtt_test_ssid_1                  c0:3d:46:00:3e:aa OPEN      -14  8 300
test_ssid                        3c:f5:91:8e:4c:79 WPA2_AES_PSK -18 ec:88:8f:88:aa:9a  6 72
rtt_test_ssid_2                  WPA2_MIXED_PSK -47 c0:3d:46:00:41:ca  6 144
rtt_test_ssid_3                  WPA2_MIXED_PSK -48           3 300
```

3.5.1.2. WiFi connection After executing the WiFi connection command, if the hotspot exists and the password is correct, the development board will connect to the hotspot and obtain After the network connection is successful, you can use sockets for network communication.

- The wifi connection command format is as follows

```
wifi join rtt-SSID0 12345678
```

- Command parsing

Field	describe
Wi-Fi	All wifi commands start with wifi
join	Wi-Fi connection action
ssid	Hotspot name
123456789	Hotspot password. If you don't have a password, you don't need to enter this item.

- After the connection is successful, the obtained IP address will be printed on the terminal, as shown below

```
wifi join ssid_test 12345678
[I/WLAN.mgnt] wifi connect success ssid:ssid_test
[I/WLAN.lwip] Got IP address : 192.168.1.110
```

3.5.1.3. WiFi Disconnect After executing the WiFi Disconnect command, the development board will disconnect from the hotspot.

- The wifi disconnect command format is as follows

```
wifi disc
```

- Command parsing

Field	describe
Wi-Fi	All wifi commands start with wifi
disc	Wi-Fi disconnect action

- After successful disconnection, the following information will be printed on the terminal, as shown below

```
wifi disc
[I/WLAN.mgnt] disconnect success!
```

3.5.2. WiFi Scanning

The following code demonstrates a synchronous Wi-Fi scan and prints the results to the terminal. First, Wi-Fi initialization is performed, followed by the Wi-Fi scan function rt_wlan_scan_sync. This function is synchronous and returns the number of scans and the results. In this example, the names of the scanned hotspots are printed.

```
#include <rthw.h>
#include <rtthread.h>

#include <wlan_mgnt.h> #include
<wlan_prot.h> #include
<wlan_cfg.h>

void wifi_scan(void) {

    struct rt_wlan_scan_result *result; int i = 0;


    /* Configuring WLAN device working mode */
    rt_wlan_set_mode(RT_WLAN_DEVICE_STA_NAME, RT_WLAN_STATION);
    /* WiFi scan */
    result = rt_wlan_scan_sync();
    /* Print scan results */ rt_kprintf("scan
    num:%d\n", result->num); for (i = 0; i < result->num; i++) {


        rt_kprintf("ssid:%s\n", result->info[i].ssid.val);
```

```
        }
}

int scan(int argc, char *argv[]) {

        wifi_scan(); return
        0;
}
MSH_CMD_EXPORT(scan, scan test.);
```

The results are as follows:

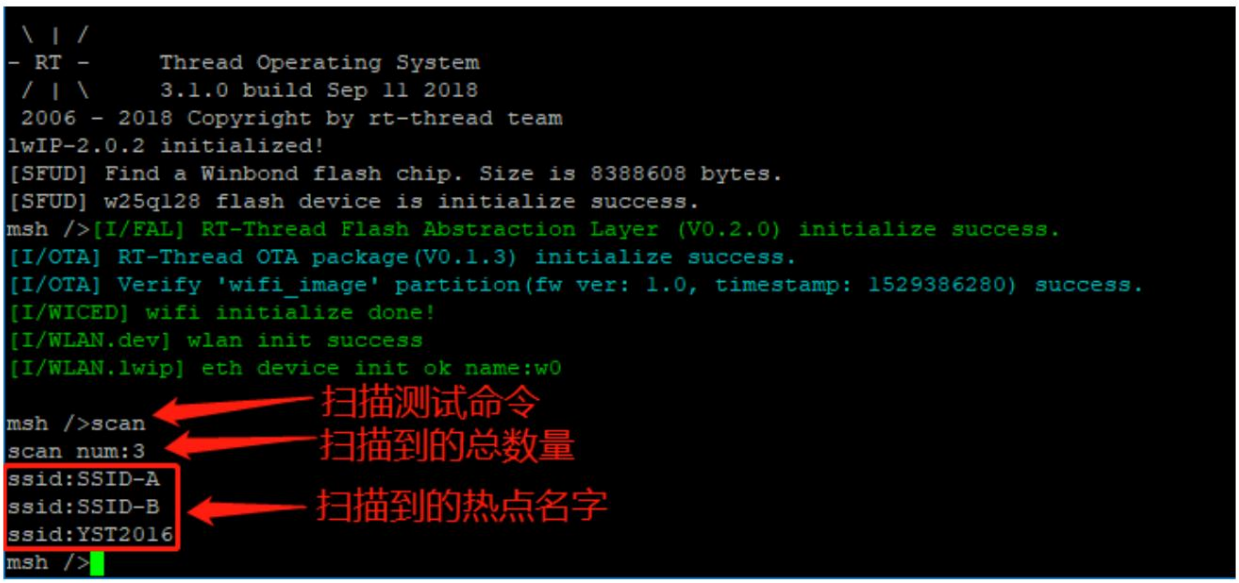

Figure **4:** scanning

3.5.3. WiFi connection and disconnection

The following code snippet will demonstrate a simultaneous WiFi connection.

First, you need to initialize Wi-Fi and create a semaphore to wait for the RT_WLAN_EVT_READY event. You then register callback functions for the desired events and execute the rt_wlan_connect Wi-Fi connection function. This function returns whether the connection is successful. However, even if the connection is successful, communication is not yet possible; you still need to wait for the network to acquire an IP address. Using the previously created semaphore, you wait for the network to become ready. Once the network is ready, communication can resume normally.

After connecting to Wi-Fi, wait for a while and then execute the rt_wlan_disconnect function to disconnect. The disconnect operation is blocking, and the return value indicates whether the disconnection is successful.

```
#include <rthw.h>
#include <rtthread.h>

#include <wlan_mgnt.h>
#include <wlan_prot.h> #include
<wlan_cfg.h>

#define WLAN_SSID                       "SSID-A"
```

```
#define WLAN_PASSWORD                   "12345678"
#define NET_READY_TIME_OUT              (rt_tick_from_millisecond(15 * 1000))

static rt_sem_t net_ready = RT_NULL;

static void
wifi_ready_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__);
    rt_sem_release(net_ready);
}

static void
wifi_connect_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

static void
wifi_disconnect_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

static void
wifi_connect_fail_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

rt_err_t wifi_connect(void) {

    rt_err_t result = RT_EOK;

    /* Configuring WLAN device working mode */
    rt_wlan_set_mode(RT_WLAN_DEVICE_STA_NAME, RT_WLAN_STATION);
    /* station connect */
```

```
    rt_kprintf("start to connect ap ...\n"); net_ready =
    rt_sem_create("net_ready", 0, RT_IPC_FLAG_FIFO);
    rt_wlan_register_event_handler(RT_WLAN_EVT_READY,
                wifi_ready_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_CONNECTED,
                wifi_connect_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_DISCONNECTED,
                wifi_disconnect_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_CONNECTED_FAIL,
                wifi_connect_fail_callback, RT_NULL);

    /* connect wifi */
    result = rt_wlan_connect(WLAN_SSID, WLAN_PASSWORD);

    if (result == RT_EOK) {

        /* waiting for IP to be got successfully */ result =
        rt_sem_take(net_ready, NET_READY_TIME_OUT); if (result == RT_EOK) {


            rt_kprintf("networking ready!\n");
        }
        else
        {
            rt_kprintf("wait ip got timeout!\n");

        } rt_wlan_unregister_event_handler(RT_WLAN_EVT_READY);
        rt_sem_delete(net_ready);

        rt_thread_delay(rt_tick_from_millisecond(5 * 1000)); rt_kprintf("wifi
        disconnect test!\n");
        /* disconnect */
        result = rt_wlan_disconnect(); if (result !=
        RT_EOK) {

            rt_kprintf("disconnect failed\n"); return result;


        } rt_kprintf("disconnect success\n");
    }
    else
    {
        rt_kprintf("connect failed!\n");

    } return result;
}

int connect(int argc, char *argv[]) {
```

```
    wifi_connect();
    return 0;
}
MSH_CMD_EXPORT(connect, connect test.);
```

The results are as follows

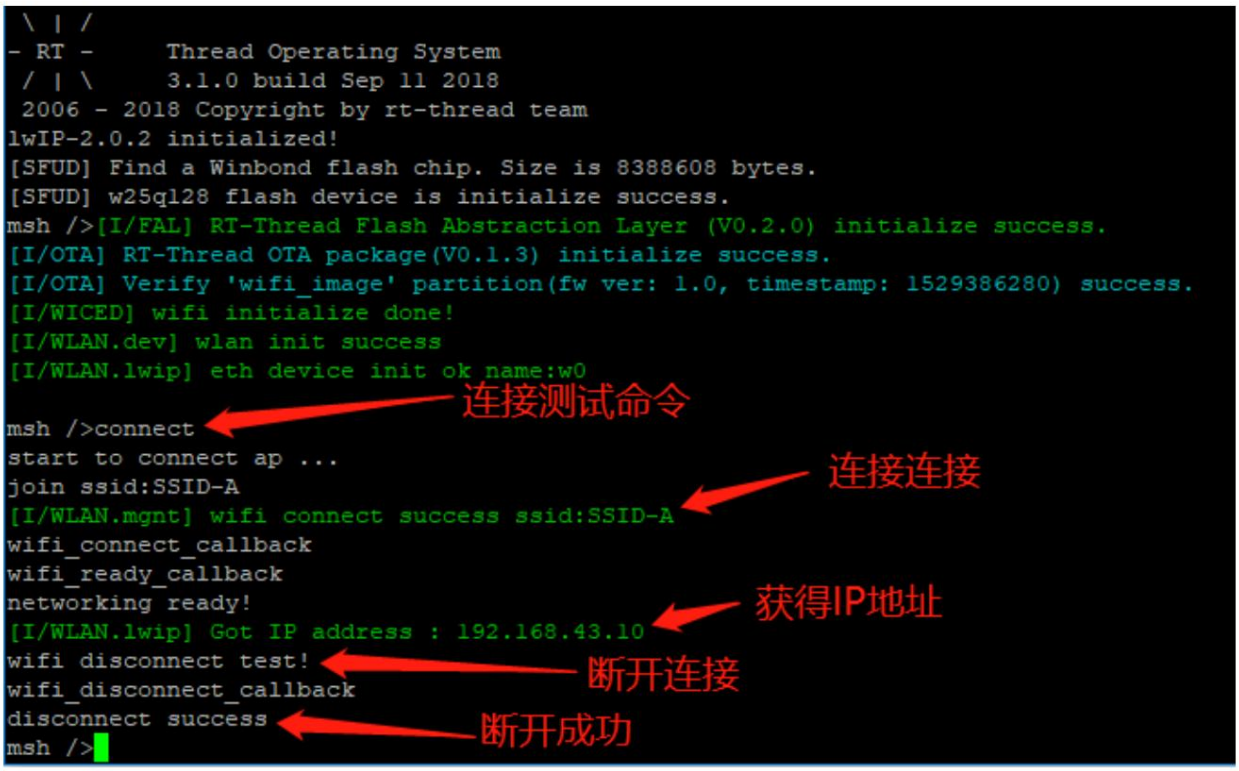

Figure **5:** Disconnection

3.5.4. Enable automatic reconnection on WiFi

First, enable the automatic reconnection function. Use the command line to connect to hotspot A, then connect to hotspot B. After waiting for a few seconds, power off hotspot B. The system will automatically retry connecting to hotspot B. If hotspot B fails to connect, the system will automatically switch to hotspot A. After successfully connecting to hotspot A, the system will stop connecting.

```
#include <rthw.h>
#include <rtthread.h>

#include <wlan_mgnt.h>
#include <wlan_prot.h>
#include <wlan_cfg.h>

static void
wifi_ready_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__);
}
```

```
static void
wifi_connect_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

static void
wifi_disconnect_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

static void
wifi_connect_fail_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

int wifi_autoconnect(void) {

    /* Configuring WLAN device working mode */
    rt_wlan_set_mode(RT_WLAN_DEVICE_STA_NAME, RT_WLAN_STATION);
    /* Start automatic connection */
    rt_wlan_config_autoreconnect(RT_TRUE); /* register
    event */
    rt_wlan_register_event_handler(RT_WLAN_EVT_READY,
            wifi_ready_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_CONNECTED,
            wifi_connect_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_DISCONNECTED,
            wifi_disconnect_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_CONNECTED_FAIL,
            wifi_connect_fail_callback, RT_NULL);
    return 0;
}

int auto_connect(int argc, char *argv[])
```

```
{
    wifi_autoconnect();
    return 0;
}
MSH_CMD_EXPORT(auto_connect, auto connect test.);
```

The results are as follows:

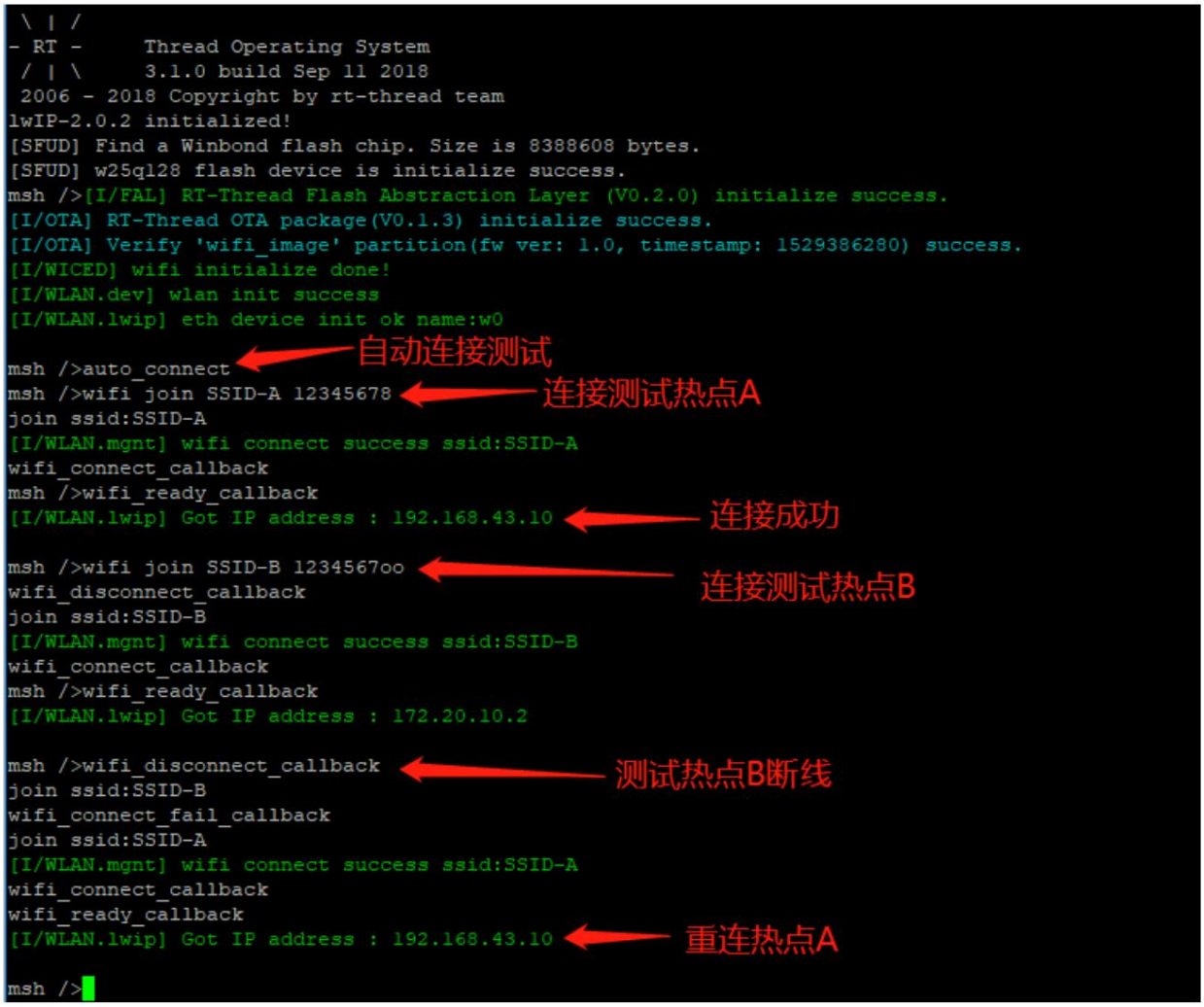
\ | /
- RT - Thread Operating System
/ | \ 3.1.0 build Sep 11 2018
2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
[SFUD] Find a Winbond flash chip. Size is 8388608 bytes.
[SFUD] w25q128 flash device is initialize success.
msh />[I/FAL] RT-Thread Flash Abstraction Layer (V0.2.0) initialize success.
[I/OTA] RT-Thread OTA package(V0.1.3) initialize success.
[I/OTA] Verify 'wifi_image' partition(fw ver: 1.0, timestamp: 1529386280) success.
[I/WICED] wifi initialize done!
[I/WLAN.dev] wlan init success
[I/WLAN.lwip] eth device init ok name:w0
msh />auto_connect
自动连接测试
msh />wifi join SSID-A 12345678
连接测试热点A
join ssid:SSID-A
[I/WLAN.mgnt] wifi connect success ssid:SSID-A
wifi_connect_callback
msh />wifi_ready_callback
[I/WLAN.lwip] Got IP address : 192.168.43.10
连接成功
msh />wifi join SSID-B 1234567oo
连接测试热点B
wifi_disconnect_callback
join ssid:SSID-B
[I/WLAN.mgnt] wifi connect success ssid:SSID-B
wifi_connect_callback
msh />wifi_ready_callback
[I/WLAN.lwip] Got IP address : 172.20.10.2
msh />wifi_disconnect_callback
测试热点B断线
join ssid:SSID-B
wifi_connect_fail_callback
join ssid:SSID-A
[I/WLAN.mgnt] wifi connect success ssid:SSID-A
wifi_connect_callback
wifi_ready_callback
[I/WLAN.lwip] Got IP address : 192.168.43.10
重连热点A
msh />
```

Figure 6: Automatic connection

## 4References

### 4.1 APIs related to this article

## 4.2 API List

| API | Location |
|---|---|
| rt_wlan_set_mode | wlan_mgnt.c |
```

API	Location
rt_wlan_prot_attach	wlan_prot.c
rt_wlan_scan_sync	wlan_mgnt.c
rt_wlan_connect	wlan_mgnt.c
rt_wlan_disconnect	wlan_mgnt.c
rt_wlan_config_autoreconnect	wlan_mgnt.c

4.3 Detailed explanation of core API

4.3.1. rt_wlan_set_mode()

4.4 Function

Register the Wi-Fi device to the Wi-Fi device framework

4.5 Function Prototype

```
rt_err_t rt_wlan_set_mode(const char *dev_name, rt_wlan_mode_t mode);
```

4.6 Function Parameters

parameter	describe
dev_name	Wi-Fi device name
mode	Wi-Fi device operating mode

4.7 Return Value

Return Value	describe
-RT_EINVAL	Parameter error
-RT_EIO	Device not found
-RT_ERROR	Execution failed
RT_EOK	Execution successful

wlan mode can take one of the following values RT_WLAN_NONE clear the working mode RT_WLAN_STATION work in STATION mode RT_WLAN_AP works in AP mode

4.7.1. rt_wlan_prot_attach()

4.8 Function

Specify the protocol used by the WLAN device

4.9 Function Prototype

```
rt_err_t rt_wlan_prot_attach(const char *dev_name, const char *prot_name);
```

4.10 Function Parameters

parameter	describe
name	Wi-Fi device name
prot_name	Protocol Name

4.11 Return Value

Return Value	describe
-RT_ERROR	Execution failed
RT_EOK	Execution successful

type can be one of the following values RT_WLAN_PROT_LWIP protocol type is LWIP

4.11.1. rt_wlan_scan_sync()

4.12 Function

Simultaneous scanning of hotspots

4.13 Function Prototype

```
struct rt_wlan_scan_result *rt_wlan_scan_sync(void);
```

4.14 Function Parameters

none

4.15 Return Value

Return Value	describe
rt_wlan_scan_result	Scan Results

The scan result is a structure with the following members

num: info number info: info pointer

```
struct rt_wlan_scan_result
{
    rt_int32_t num;
    struct rt_wlan_info *info;
};
```

4.15.1. rt_wlan_connect()

4.16 Function

Simultaneous connection hotspot

4.17 Function Prototype

```
rt_err_t rt_wlan_connect(const char *ssid, const char *password);
```

4.18 Function Parameters

parameter	describe
ssid	Wi-Fi Name
password	WIFI password

4.19 Return Value

Return Value	describe
-RT_EINVAL	Parameter error
-RT_EIO	Unregistered device
-RT_ERROR	Connection failed
RT_EOK	Connection successful

Return Value	describe

4.19.1. rt_wlan_disconnect()

4.20 Function

Synchronize disconnect hotspot

4.21 Function Prototype

```
rt_err_t rt_wlan_disconnect(void);
```

4.22 Function Parameters

none

4.23 Return Value

Return Value	describe
-RT_EIO	Unregistered device
-RT_ENOMEM	Out of memory
-RT_ERROR	Disconnection failed
RT_EOK	Disconnect successfully

4.23.1. rt_wlan_config_autoreconnect()

4.24 Function

Configuring automatic reconnection mode

4.25 Function Prototype

```
void rt_wlan_config_autoreconnect(rt_bool_t enable);
```

4.26 Function Parameters

parameter	describe
enable	enable/disable automatic reconnection

4.27 Return Value

none