MT793X IoT SDK for

Wi-Fi Developer’s Guide

Version:	0.7.3
Release date:	2022-08-19

Use of this document and any information contained therein is subject to the terms and conditions set forth in Exhibit 1. This document is subject to change without notice.

Version History

Version	Date	Description
0.1	2021-04-08	Initial release of STA get started guide
0.5	2021-04-15	Add more information on STA Mode
0.6	2021-05-30	Add AP usage
0.6.1	2021-06-12	Add COEX chapter
0.6.2	2021-06-15	Add low power chapter
0.7	2022-08-10	Add antenna diversity chapter
0.7.1	2022-08-18	Add Sub-SYS reset
0.7.2	2022-08-18	Add agile multiband
0.7.3	2022-08-19	Add roaming chapter

Table of Contents

List of Figures

List of Tables

1 Overview

MT793X series chipsets are based on Wi-Fi System-on-Chip (SoC) with embedded TCP/IP stack for Internet of Things (IoT) devices that can connect to other smart devices or to cloud applications and services directly. The IoT SDK (Software Development Kit) provides API and example applications for the Wi-Fi module.
This document guides you through the following:

- Initializing the Wi-Fi module in Station (STA) modes.
- Configuring the module to operate in STA mode
- Scanning for the available stations.

1.1 Supported Features

The Wi-Fi module supports the following features and modes.

- Support 1T1R IEEE802.11 a/b/g/n/ac/ax
- Support 20MHz bandwidth only until MCS8 (256-QAM) in 2.4G/5GHz
- Individual Target Wake Time (TWT) under AX mode
- Orthogonal Frequency Division Multiple Access (OFDMA) UL/DL under AX mode
- STA Open, Open Wired Equivalent Privacy (WEP)
- STA Wi-Fi protected access: (WPA)/WPA2/WPA3 personal
- STA Wi-Fi power-save mode and Delivery Traffic Indication Message (DTIM) configuration
- Wi-Fi aggregated MAC protocol data unit (A-MPDU) support

1.2 Software Architecture of the Wi-Fi Module

There are three software layers: **Driver**, **Middleware** and **Application**, as shown Figure 1 shows. The Wi-Fi module is located in **Middleware**.

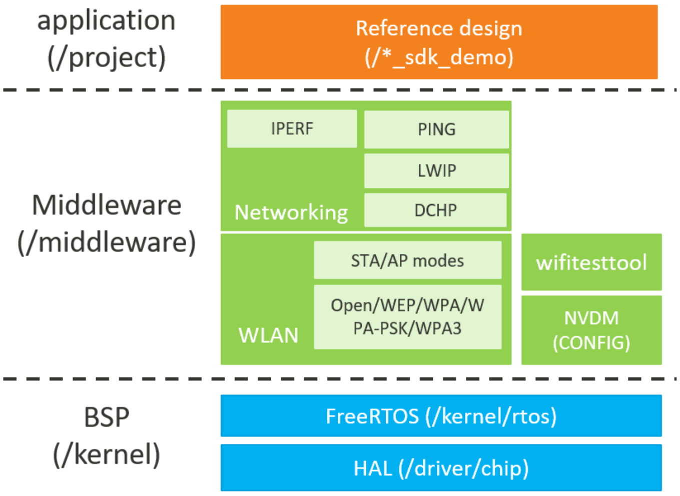


Figure 1. Wi-Fi SYS Architecture

A brief description of the layers is as follows:

- BSP
  - FreeRTOS. An OS (Operating System) with the open source software for Middleware components and Application.
  - Hardware Abstraction Layer (HAL). Provides the driver Application Programming Interface (API) encapsulating the low-level functions of peripheral drivers for the operating system, Middleware features and Application
- Middleware
  - WLAN. Provides OS dependent function calls, including Wi-Fi APIs that control the bridge supplicant and network processor messages.
  - wifitesttool. Provides factory test function for manufacturers
  - Network. Provides OS dependent features such as PING and IPERF
  - LwIP. A widely used open source TCP/IP stack designed for embedded systems. The goal of the LwIP TCP/IP implementation is to reduce resource usage while still having a full-scale TCP
  - Non-Volatile Data Management (NVDM). Store the configuration of Wi-Fi such as MAC address
- Application
  - Pre-configured projects using Middleware components such as Wi-Fi station

The features, configuration options, and module paths are listed as below:

Feature	**Feature Option (/project/*_sdk_demo/GCC/feature.mk)**	**Module Path**
WLAN	MTK_MT7933_CONSYS_ENABLE MTK_MT7933_CONSYS_WIFI_ENABLE	middleware\MTK\minisupp middleware\MTK\connectivity\wlan
wifitesttool	MTK_WLAN_SERVICE_ENABLE MTK_WIFI_TEST_TOOL_ENABLE	middleware\MTK\connectivity wlan_tool\wifi_test_tool
DHCP	Makefile: include dhcpd	middleware\MTK\dhcpd
LwIP	MTK_LWIP_ENABLE	middleware\third_party\lwip
ping	MTK_PING_OUT_ENABLE	middleware\third_party\ping
IPERF	MTK_IPERF_ENABLE	middleware\MTK\iperf
NVDM (Config)	MTK_NVDM_ENABLE	project*_sdk_demo\ src\ network_default_config.c wifi_cfg_default_config.c

2 Start with CLI

There are several ways to use SDK, 1) from CLI command line through UART command line, and 2) by API from user application. This chapter goes through 1), while 2) is introduced in Section 4 Using the Module through API.

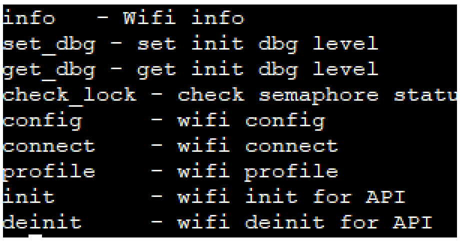
The functions can be performed under CLI shown as follows. This chapter goes through the Wi-Fi connection procedure, which includes 1) how to set up Wi-Fi connection with different security modes, and 2) how to use iperf/ping to test network. For basic HDK usage guide, please refer to MT793x_Hands_on_Training.pdf. For detailed information on Wi-Fi hardware performance evaluation, please refer to MT793x_ATE_Cal_Flow.pdf.

Table 1. SDK Demo Supported Functions

Functions	Description
Wi-Fi connection	Perform Wi-Fi connection with different security mode and get IP address through DHCP
Wi-Fi hardware performance evaluation	Perform Wi-Fi test with tool. Please refer to tool document
Wi-Fi iperf/ping test	Perform Wi-Fi performance/connection test when connected to Wi-Fi router

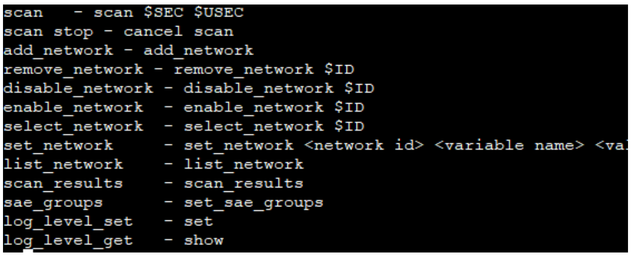
2.1 Get Started with Wi-Fi CLI Commands

With "wifi" and "wpa_cli" CLI commands, you can perform several functions, including initialize Wi-Fi, scan, connection etc.



```
info    - Wifi info
set_dbg - set init dbg level
get_dbg - get init dbg level
check_lock - check semaphore statu
config      - wifi config
connect     - wifi connect
profile     - wifi profile
init        - wifi init for API
deinit      - wifi deinit for API
```

Figure 2. Supported Wi-Fi CLI Commands



```
scan    - scan $SEC $USEC
scan stop - cancel scan
add_network - add_network
remove_network - remove_network $ID
disable_network - disable_network $ID
enable_network  - enable_network $ID
select_network  - select_network $ID
set_network     - set_network <network id> <variable name> <va
list_network    - list_network
scan_results    - scan_results
sae_groups      - set_sae_groups
log_level_set   - set
log_level_get   - show
```

Figure 3. Supported wpa_cli Commands

This section provides the description of each command and Section 2.2 introduces good practices for how to use different commands.

Commands	Description
wifi info	Show the status of Wi-Fi STA connection such as connected channel and IP.
wifi set/get_dbg	Set Wi-Fi internal debug log levels, ranging from 0x1 to 0x3f.
wifi config	Configure Wi-Fi connection parameters such as SSID and security mode. Will be introduced in the following section.
wifi connect	Query status under connection.
wifi profile	Configure MAC address.
wifi init/deinit	Initialize/De-initialize Wi-Fi. Note: The deinitialization command disassociates AP and stops scan.

2.2 Good Practices for Wi-Fi CLI

2.2.1 Wi-Fi Initialization and Scan/Stop Scan

```
$ wifi init # init Wi-Fi and it will start scan after init
[wlan]wifi_init(), ret:Success, Code=0
wlan0: State: DISCONNECTED -> SCANNING
...
wlan0: Event SCAN_RESULTS (3) received
[wlan]result[1] b0:2a:43:e9:4d:c@freq:2412 Reset_a_p2mcc_ch1 ie_len 42
[wlan]result[2] 3c:37:86:93:9d:1f@freq:2417 NETGEAR94-Vick ie_len 331
[wlan]result[3] a8:5e:45:af:b0:30@freq:2442 AX6000_2G ie_len 263
[wlan]result[4] 4:a1:51:13:ee:a@freq:2462 NET-2G ie_len 315
[wlan]result[5] 0:c:43:26:46:30@freq:2462 MTK_7915_AP_boforn ie_len 151
[wlan]result[6] 4:a1:51:13:ee:9@freq:5745 NET-5G ie_len 214
...

$ wifi connect set scan 0 # 0: stop scan, 1, 0, 0: start scan
```

2.2.2 Connect to AP and Disconnect from AP

open mode:

```
wifi config set ssid 0 <ssid>
wifi config set sec 0 0 0
wifi config set reload (connected)
wifi config get ssid 0
wifi config get sec 0
```

wep mode:

```
wifi config set ssid 0 <ssid>
wifi config set sec 0 0 0
wifi config set wep 0 <key_idx> <key>
wifi config set reload
wifi config get ssid 0
wifi config get sec 0
wifi config get wep 0
```

wpa_psk mode:

```
wifi config set ssid 0 <ssid>
wifi config set sec 0 9 4
wifi config set psk 0 <passphrase>
wifi config set reload (connected)
wifi config get ssid 0
wifi config get sec 0
wifi config get psk 0
```

wpa2_psk mode:

```
wifi config set ssid 0 <ssid>
wifi config set sec 0 7 6
wifi config set psk 0 <passphrase>
wifi config set reload (connected)
wifi config get ssid 0
wifi config get sec 0
wifi config get psk 0
```

wpa3_psk:

```
wifi config set ssid 0 <ssid>
wifi config set sec 0 11 6
wifi config set psk 0 <passphrase>
wifi config set reload (connected)
wifi config get ssid 0
wifi config get sec 0
wifi config get psk 0
```

disconnection:

```
wifi connect set disconap
```

2.2.3 Use PING/IPERF to Test

```
Ping: ping <addr> <count> <pkt_len>
Example:
ping 10.10.10.254 1 500
$ 00-00 00:46:20.549 102 I [ping]: ping: send seq(0x0074) 10.10.10.254
00-00 00:46:20.551 102 I [ping]: ping: recv seq(0x0074) 10.10.10.254, 1 ms
00-00 00:46:21.551 102 I [ping]: 10.10.10.254, Packets: Sent = 1, Received =1, Lost = 0 (0% loss)
00-00 00:46:21.552 102 I [ping]: Packets: min = 1, max =1, avg = 1
Iperf:
Client/Server
  -u,       use UDP rather than TCP
  -p,   #   server port to listen on/connect to (default 5001)
  -n,   #[kmKM]   number of bytes to transmit
  -b,   #[kmKM]   for UDP, bandwidth to send at in bits/sec
  -i,       10 seconds between periodic bandwidth reports
Server specific:
  -s,       run in server mode
  -B,   <ip>   bind to <ip>, and join to a multicast group (only Support UDP)
  -r,       for UDP, run iperf in tradeoff testing mode, connecting back to client
Client specific:
  -c,   <ip>   run in client mode, connecting to <ip>
  -w,   #[kmKM]   TCP window size
  -l,   #[kmKM]   UDP datagram size
  -t,   #   time in seconds to transmit for (default 10 secs)
  -S,   #   the type-of-service of outgoing packets\n
Example:
Iperf TCP Server: iperf -s
Iperf UDP Server: iperf -s -u
Iperf TCP Client: iperf -c <ip> -w <window size> -t <duration> -p <port>
Iperf UDP Client: iperf -c <ip> -u -l <datagram size> -t <duration> -p <port>
```

3 Wi-Fi Configuration

3.1 Configuration Options

You can change the setting of STA through options configured in NVDM. Reboot is needed to make the setting take effect. The configuration options and their descriptions are shown as below.

Table 2. Configuration Options for Wi-Fi

NVDM Group	Configuration Options	Description
STA	MacAddr	Configure STA MAC address.
wifi	ExtendedRange	Enable extended range for HE mode
wifi	Sta*Bfee	Enable Beamformee in n, ac, ax modes
wifi	*BaSize	Configure BA size for different modes
wifi	EfuseBufferModeCal	Apply calibration data in eFuse or buffer (for manufacturers)

3.2 Region and Supported Channel List

Please refer to Table 8 for list of the supported region and channel. By using the following API, you can change the channel list and power table to apply new region code.

API. (os/freertos/include/wifi_api.h)	Description
`int32_t wifi_config_set_country_code(wifi_country_code_t *wifi_country_code)`	Set country code to Wi-Fi

`typedef struct {`
`    uint8_t country_code[4];`
`    /**< Country code string.*/`
`...`
`} wifi_country_code_t;`
	Only country_code[4] takes effect in the design. For supported region string, please refer to Table 8 for the supported region

Sample code

```
int32_t ret = 0;
wifi_country_code_t country_code;

memzero(country_code, sizeof(wifi_country_code_t));
memcpy(country_code, “US”, strlen(“US”));

ret = wifi_config_set_country_code(country_code);
```

Test with SDK CLI

```
iwpriv wlan0 driver country US
iwpriv wlan0 driver get_country
------------------------------------------------------------
```

```
(210105_16:19:32.654)iwpriv wlan0 driver get_country
(210105_16:19:32.654)[wlan]Command success.
(210105_16:19:32.654)[wlan]wlan0 driver:
(210105_16:19:32.654)[wlan]
(210105_16:19:32.654)Country Code: US (0x5355)
```

4 Using the Module through API

Initialize the Wi-Fi settings by calling the `wifi_init()` function, which also initiates the IP module (LwIP). Most of the Wi-Fi APIs in wifi_api.h require FreeRTOS to be running. These APIs should be invoked in a task triggered by vTaskStartScheduler().

Table 3. APIs

API. (os/freertos/include/wifi_api.h)	Description
wifi_init()	Initialize Wi-Fi
wifi_connection_register_event_handler() wifi_connection_unregister_event_handler ()	Register event notification
wifi_config_get_mac_address ()	Get MAC address of the Wi-Fi interface

You can register an event handler to complete the initialization and call the Wi-Fi APIs, as shown in the example code below.

```
/* Register wifi initialization event handler in main() function */
wifi_connection_register_event_handler(WIFI_EVENT_IOT_INIT_COMPLETE, user_wifi_init_callback);

/* User-defined wifi initialization callback function */
bool g_wifi_init_ready = false;
void user_wifi_init_callback() { g_wifi_init_ready = true; }
void user_wifi_init_qurey_status(){
        while(g_wifi_init_ready == false) {
                vTaskDelay(20);
        }
}

/* User-defined task */
user_task() {
        wifi_config_t config = {0};

        config.opmode = WIFI_MODE_STA_ONLY;
        wifi_init(&config, NULL);
        user_wifi_init_query_status();
        /* Call APIs to connect only after Wi-Fi is initialized. open mode: */
        wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
        wifi_config_set_ssid(WIFI_PORT_STA, (uint8_t *)"SS2_EA8500_2G", kalStrLen("SS2_EA8500_2G"));
        wifi_config_set_security_mode(WIFI_PORT_STA, WIFI_AUTH_MODE_OPEN,
WIFI_ENCRYPT_TYPE_WEP_ENABLED);

        wifi_config_reload_setting(); //connect to the AP

}
```

Table 4. Supported Events

Event for wifi_connection_un/register_event	Description
WIFI_EVENT_IOT_CONNECTED	**Connected event** The event is triggered when authentication and association are complete. Event payload • MAC address (6 bytes) and port number (1 byte). • MAC address. The MAC address of a remote AP that the device is connected to or the MAC address of a client connected to the device when it is in AP mode. • Port number indicates from which port the event initiates. • 0, STA port
WIFI_EVENT_IOT_SCAN_COMPLETE	Scan complete event Triggered when the scan process is complete. Event payload NULL
WIFI_EVENT_IOT_DISCONNECTED	Disconnected event
WIFI_EVENT_IOT_PORT_SECURE	Port secure event This event is triggered after 4-way handshake
WIFI_EVENT_IOT_CONNECTION_FAILED	Connection failed event This event is triggered when connection to an AP fails, including disconnecting from an AP, connecting to an AP with wrong password.
WIFI_EVENT_IOT_INIT_COMPLETE	Initialization complete event for the Wi-Fi module. Triggered when supplicant initialization is complete. Event payload • Zero data (6 bytes) and port number (1 byte). • Port number indicates from which port the event initiates • 0, STA port

4.1 Using the Wi-Fi Module in STA Mode

In most cases, using the configuration to initialize the Wi-Fi module in STA mode takes effect immediately. However, some settings can only take effect by calling the wifi_config_reload_setting() after the following APIs are called:

- wifi_config_set_pmk(uint8_t port, uint8_t *pmk)

- wifi_config_set_security_mode(uint8_t port, wifi_auth_mode_t auth_mode, wifi_encrypt_type_t encrypt_type)
- wifi_config_set_ssid(uint8_t port, uint8_t *ssid, uint8_t ssid_length)
- wifi_config_set_wep_key()
- wifi_config_set_wpa_psk_key()

The supported modes are as listed below:

Table 5. Supported Auth and SEC Modes

Supported Mode	
Open mode	
WPA_PSK and AES mode	
WPA_PSK and TKIP mode	
WPA_PSK and AES+TKIP mode	
WPA2_PSK and TKIP mode	
WPA2_PSK and AES+TKIP mode	
WPA_PSK_WPA2_PSK and AES mode	
WPA_PSK_WPA2_PSK and TKIP mode	
WPA_PSK_WPA2_PSK and AES+TKIP mode	
WEP Open, shared, and auto switch mode	
WPA2_PSK_WPA3_PSK and AES mode	
WPA2_PSK_WPA3_PSK and AES+TKIP mode	
WPA3_PSK and AES mode	Group 19: 256-bit random ECP group (NIST) (Mandatory)
	Group 20: 384-bit random ECP group (NIST)
	Group 21: 512-bit random ECP group (NIST)

```
/* API use example for different AUTH & SEC mode */

/* Include header file */

#include “wifi_api_ex.h”
```

open mode:

```
wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
wifi_config_set_ssid(WIFI_PORT_STA, (uint8_t *)"EA8500_2G", kalStrLen("EA8500_2G"));
wifi_config_set_security_mode(WIFI_PORT_STA, WIFI_AUTH_MODE_OPEN,
WIFI_ENCRYPT_TYPE_WEP_ENABLED);
wifi_config_reload_setting(); //reload setting, then auto scanning to connect to AP
```

wpa_psk mode:

```
wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
wifi_config_set_ssid(WIFI_PORT_STA, (uint8_t *)"EA8500_2G", kalStrLen("EA8500_2G"));
wifi_config_set_security_mode(WIFI_PORT_STA, WIFI_AUTH_MODE_WPA_PSK,
WIFI_ENCRYPT_TYPE_TKIP_ENABLED);
wifi_config_set_wpa_psk_key(WIFI_PORT_STA, "12345678", kalStrLen("12345678"));
wifi_config_reload_setting(); //reload setting, then auto scanning to connect to AP
```

wpa2_psk mode:

```
wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
wifi_config_set_ssid(WIFI_PORT_STA, (uint8_t *)"EA8500_2G", kalStrLen("EA8500_2G"));
wifi_config_set_security_mode(WIFI_PORT_STA, WIFI_AUTH_MODE_WPA2_PSK,
WIFI_ENCRYPT_TYPE_AES_ENABLED);
wifi_config_set_wpa_psk_key(WIFI_PORT_STA, "12345678", kalStrLen("12345678"));
wifi_config_reload_setting(); //reload setting, then auto scanning to connect to AP
```

wpa3_psk mode:

```
wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
wifi_config_set_ssid(WIFI_PORT_STA, (uint8_t *)"EA8500_2G", kalStrLen("EA8500_2G"));
wifi_config_set_security_mode(WIFI_PORT_STA, WIFI_AUTH_MODE_WPA3_PSK,
WIFI_ENCRYPT_TYPE_AES_ENABLED);
wifi_config_set_wpa_psk_key(WIFI_PORT_STA, "12345678", kalStrLen("12345678"));
wifi_config_reload_setting(); //reload setting, then auto scanning to connect to AP
```

wep open mode:

```
wifi_wep_key_t wep_key = {{{0}, {0} }, {0}, 0};

wep_key.wep_tx_key_index = 0;

wep_key.wep_key_length[wep_key.wep_tx_key_index] = 5;

AtoH((char *)”1234567890”,(char *)&wep_key.wep_key[wep_key.wep_tx_key_index],
(int)wep_key.wep_key_length[wep_key.wep_tx_key_index]);

wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
wifi_config_set_ssid(WIFI_PORT_STA, (uint8_t *)"EA8500_2G", kalStrLen("EA8500_2G"));
wifi_config_set_security_mode(WIFI_PORT_STA, WIFI_AUTH_MODE_OPEN,
WIFI_ENCRYPT_TYPE_WEP_ENABLED);
wifi_config_set_wep_key(WIFI_PORT_STA, &wep_keys);

wifi_config_reload_setting(); //reload setting, then auto scanning to connect to AP
```

wep shared/auto switch mode:

```
wifi_wep_key_t wep_key = {{{0}, {0} }, {0}, 0};

wep_key.wep_tx_key_index = 0;

wep_key.wep_key_length[wep_key.wep_tx_key_index] = 5;

AtoH((char *)”1234567890”,(char *)&wep_key.wep_key[wep_key.wep_tx_key_index],
(int)wep_key.wep_key_length[wep_key.wep_tx_key_index]);

wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
wifi_config_set_ssid(WIFI_PORT_STA, (uint8_t *)"EA8500_2G", kalStrLen("EA8500_2G"));
wifi_config_set_security_mode(WIFI_PORT_STA, WIFI_AUTH_MODE_SHARED,
WIFI_ENCRYPT_TYPE_WEP_ENABLED);
wifi_config_set_wep_key(WIFI_PORT_STA, &wep_keys);

wifi_config_reload_setting(); //reload setting, then auto scanning to connect to AP
```

The configuration APIs use in-band mechanism for Wi-Fi driver and Wi-Fi firmware communication; thus, these APIs must be called after the OS task scheduler has started, to make sure an in-band task is running.

4.2 Using the Wi-Fi Module in AP Mode

This section introduces how to initialize and configure the Wi-Fi module in AP mode. A wireless AP is a device that allows wireless devices to connect to a wired network through Wi-Fi or related standards. The AP usually connects to a router as a standalone device, but it can also be an integral component of the router. An AP is different from a hotspot, which is the physical space where the wireless service is provided. The development board can be configured as an AP. The supported modes are as listed below:

- Open mode
- WPA2_PSK and AES modes

Table 6 lists the supporting APIs to set the platform to AP mode.

Table 6. Configuration and Connection APIs in the AP mode

API	Description
wifi_config_set_opmode()	This function sets the Wi-Fi operation mode and it takes effect immediately.
wifi_config_set_ssid()	This function sets the SSID and SSID length that Wi-Fi driver uses for a specific wireless port.
wifi_config_set_channel()	This function sets the channel number that the Wi-Fi driver uses for a specific wireless port.
wifi_config_set_security_mode()	This function sets the authentication and encryption modes used in the Wi-Fi driver for a specific wireless port.

API	Description
wifi_config_set_wpa_psk_key()	This function sets the password of WPA-PSK or WPA2-PSK encryption type used in the Wi-Fi driver.
wifi_config_set_dtim_interval()	This function sets the DTIM interval used in the Wi-Fi driver.
wifi_config_set_bcn_interval()	This function sets the Beacon interval used in the Wi-Fi driver.
wifi_config_reload_setting()	This function applies the new network configurations used in the Wi-Fi driver.
wifi_connection_get_sta_list()	This function gets the lisf of Wi-Fi associated stations.
wifi_connection_disconnect_sta()	This function disconnects the connected Wi-Fi station.

You can find more information on the APIs in the Wi-Fi API Reference Manual.
To use the device in the AP mode, apply any of the two examples described below.

4.3 Use the Device in AP Open Mode

1) In this example, initialize the device in AP open mode on channel 6 with the SSID, "MTK_SOFT_AP”, as shown below.
   a) Define the operation mode (config.opmode), SSID (config.ap_config.ssid), channel (config.ap_config.channel), authentication mode (config.ap_config.auth_mode) and encryption type (config.ap_config.encrypt_type) in the wifi_config_t structure as shown below.

```
wifi_config_t config = {0};
config.opmode = WIFI_MODE_AP_ONLY;
strcpy((char *)config.ap_config.ssid, "MTK_SOFT_AP");
config.ap_config.ssid_length = strlen((char *)config.ap_config.ssid);
config.ap_config.auth_mode = WIFI_AUTH_MODE_OPEN;
config.ap_config.encrypt_type = WIFI_ENCRYPT_TYPE_WEP_DISABLED;
config.ap_config.channel = 6;
```

   b) Call the `wifi_init()` function to initialize the Wi-Fi driver.

```
wifi_init(&config, NULL);
```

2) Use configuration APIs to set the device in AP mode that operates on channel 6 in open mode with the SSID, “MTK_SOFT_AP”.

   a) Call the function wifi_config_set_opmode(opmode) to set the opmode to `WIFI_MODE_AP_ONLY`,

```
wifi_config_set_opmode(WIFI_MODE_AP_ONLY);
```

as shown below.

b) Call the function wifi_config_set_ssid(port, ssid_name, strlen(ssid_name)) to set the SSID ("MTK_SOFT_AP") of a given port (WIFI_PORT_AP), as shown below.

```
wifi_config_set_ssid(WIFI_PORT_AP, "MTK_SOFT_AP", strlen("MTK_SOFT_AP"));
```

c) Call the function wifi_config_set_security_mode(port, auth, encrypt) to set the security mode of the AP router, as shown below.

```
wifi_config_set_security_mode(WIFI_PORT_AP, WIFI_AUTH_MODE_OPEN,
WIFI ENCRYPT TYPE WEP DISABLED);
```

d) Call the function wifi_config_set_channel(port, channel) to set the channel of a given port, as shown below.

```
wifi_config_set_channel(WIFI_PORT_AP, 6);
```

e) Call the function wifi_config_set_bcn_interval(interval) to set the Beacon interval in ms, as shown below.

```
wifi_config_set_bcn_interval(100);
```

f) Call the function wifi_config_set_dtim_interval(interval) to set the DTIM interval and the unit is beacon interval, as shown below.

```
wifi_config_set_dtim_interval(10);
```

g) Apply the configuration by calling the function wifi_config_reload_setting(), as shown below.

```
wifi_config_reload_setting();
```

4.4 Use the Device in AP Mode with WPA2PSK Method

1) In this example, the device is in AP mode and operates on channel 6. The authentication mode is WPA2PSK (WIFI_AUTH_MODE_WPA2_PSK) with AES encryption type (WIFI_ENCRYPT_TYPE_AES_ENABLED). The password is "12345678" and the SSID is "MTK_SOFT_AP". Initialize the module in AP mode, as shown below.

a) Define the operation mode (config.opmode), SSID (config.ap_config.ssid), channel (config.ap_config.channel), authentication mode (config.ap_config.auth_mode), encryption type (config.ap_config.encrypt_type) and password (config.ap_config.password) in the wifi_config_t structure, as shown below.

```
wifi_config_t config = {0};
config.opmode = WIFI_MODE_AP_ONLY;
strcpy((char *)config.ap_config.ssid, "MTK_SOFT_AP");
config.ap_config.ssid_length = strlen("MTK_SOFT_AP");
config.ap_config.auth_mode = WIFI_AUTH_MODE_WPA2_PSK;
config.ap_config.encrypt_type = WIFI_ENCRYPT_TYPE_AES_ENABLED;
strcpy((char *)config.ap_config.password, "12345678");
config.ap_config.password_length = strlen("12345678");
```

b) Call the wifi_init() function to initialize the Wi-Fi driver.

```
wifi_init(&config, NULL);
```

2) Use configuration APIs to set the device in AP mode that operates in WPA2PSK mode with a given port, channel, password and SSID.

a) Call the function wifi_config_set_opmode(opmode) to set the opmode to WIFI_MODE_AP_ONLY, as shown below.

```
wifi_config_set_opmode(WIFI_MODE_AP_ONLY);
```

b) Call the function wifi_config_set_ssid(port, ssid_name, strlen(ssid_name)) to set the SSID ("MTK_SOFT_AP") of a given port (WIFI_PORT_AP), as shown below.

```
wifi_config_set_ssid(WIFI_PORT_AP, "MTK_SOFT_AP", strlen("MTK_SOFT_AP"));
```

c) Call the function wifi_config_set_security_mode(port, auth, encrypt) to set the security mode of the AP router, as shown below.

```
wifi_config_set_security_mode(WIFI_PORT_AP, WIFI_AUTH_MODE_WPA2_PSK,
WIFI ENCRYPT TYPE AES ENABLED);
```

d) Call the function wifi_config_set_wpa_psk_key(port, password, strlen(password)) to set the WPA2PSK key, as shown below.

```
wifi_config_set_wpa_psk_key(WIFI_PORT_AP, "12345678", strlen("12345678"));
```

e) Call the function wifi_config_set_channel(port, channel) to set the channel of a given port, as shown below.

```
wifi_config_set_channel(WIFI_PORT_AP, 6);
```

f) Call the function wifi_config_set_bcn_interval(interval) to set the Beacon interval in ms, as shown below.

```
wifi_config_set_bcn_interval(100);
```

g) Call the function wifi_config_set_dtim_interval(interval) to set the DTIM interval and the unit is beacon interval, as shown below.

```
wifi_config_set_dtim_interval(10);
```

h) Apply the configuration by calling the function wifi_config_reload_setting(), as shown below.

```
wifi_config_reload_setting();
```

4.5 Using the Wi-Fi Module in Router Mode

This section introduces how to initialize and configure the Wi-Fi module in router mode. Router mode contains two interfaces: STA interface and SAP interface. The API function can work on corresponding interface according to the input port. Refer to 4.1 and 4.2 to initialize the STA and SAP respectively. If STA and the SAP stay in the same channel, SCC is applied. Otherwise, MCC is applied. STA and SAP can use different encryption modes.

4.6 Use the Device in Router Mode with WPA2PSK Method

1) In this example, the device is in router mode and the SAP operates on channel 6. The authentication mode of both interfaces is WPA2PSK (WIFI_AUTH_MODE_WPA2_PSK) with AES encryption (WIFI_ENCRYPT_TYPE_AES_ENABLED). On the SAP side, the password is "12345678" and the SSID is "MTK_SOFT_AP". On the STA side, the SSID of the target AP is "REF_AP" and the password is "12345678". Initialize the module in router mode, as shown below.

   a) Define the operation mode (config.opmode), STA SSID (config.sta_config.ssid), STA channel (config.sta_config.channel), STA WEP presentation (config_ext.sta_wep_key_index_present) and STA password (config.sta_config.password), SAP SSID (config.ap_config.ssid), SAP channel (config.ap_config.channel), SAP authentication mode (config.ap_config.auth_mode), SAP encryption type (config.ap_config.encrypt_type) and SAP password (config.ap_config.password) in the wifi_config_t structure, as shown below.

```
wifi_config_t config = {0};
wifi_config_ext_t config_ext = {0};
config.opmode = WIFI_MODE_REPEATER;
strcpy((char *)config.sta_config.ssid, "REF_AP");
config.sta_config.ssid_length = strlen("REF_AP");
config.sta_config.channel = 6;
strcpy((char *)config.sta_config.password, "12345678");
config_ext.sta_wep_key_index_present = 0;
config.sta_config.password_length = strlen("12345678");
strcpy((char *)config.ap_config.ssid, "MTK_SOFT_AP");
config.ap_config.ssid_length = strlen("MTK_SOFT_AP");
config.ap_config.auth_mode = WIFI_AUTH_MODE_WPA2_PSK;
config.ap_config.encrypt_type = WIFI_ENCRYPT_TYPE_AES_ENABLED;
strcpy((char *)config.ap_config.password, "12345678");
config.ap_config.password_length = strlen("12345678");
config.ap_config.channel = 6;
```

b) Call the wifi_init() function to initialize the Wi-Fi driver.

```
wifi_init(&config, &config_ext);
```

2) Use configuration APIs to set the device in AP mode that operates in WPA2PSK mode with a given port, channel, password and SSID.

a) Call the function wifi_config_set_opmode(opmode) to set the opmode to WIFI_MODE_AP_ONLY, as shown below.

```
wifi_config_set_opmode(WIFI_MODE_REPEATER);
```

b) Call the function wifi_config_set_ssid(port, ssid_name, strlen(ssid_name)) to set the SSID ("REF_AP") of a given port (WIFI_PORT_STA) and the SSID ("MTK_SOFT_AP") of a given port (WIFI_PORT_AP), as shown below.

```
wifi_config_set_ssid(WIFI_PORT_AP, "MTK_SOFT_AP", strlen("MTK_SOFT_AP"));
wifi_config_set_ssid(WIFI_PORT_STA, "REF_AP", strlen("REF_AP"));
```

c) Call the function wifi_config_set_security_mode(port, auth, encrypt) to set the security mode of the STA and the SAP, as shown below.

```
wifi_config_set_security_mode(WIFI_PORT_AP, WIFI_AUTH_MODE_WPA2_PSK,
WIFI_ENCRYPT_TYPE_AES_ENABLED);
wifi_config_set_security_mode(WIFI_PORT_STA, WIFI_AUTH_MODE_WPA2_PSK,
WIFI_ENCRYPT_TYPE_AES_ENABLED);
```

d) Call the function wifi_config_set_wpa_psk_key(port, password, strlen(password)) to set the WPA2PSK key, as shown below.

```
wifi_config_set_wpa_psk_key(WIFI_PORT_AP, "12345678", strlen("12345678"));
wifi_config_set_wpa_psk_key(WIFI_PORT_STA, "12345678", strlen("12345678"));
```

e) Call the function wifi_config_set_channel(port, channel) to set the channel of a given port, as shown below.

```
wifi_config_set_channel(WIFI_PORT_AP, 6);
wifi_config_set_channel(WIFI_PORT_STA, 6);
```

f) Call the function wifi_config_set_bcn_interval(interval) to set the Beacon interval in ms, as shown below.

```
wifi_config_set_bcn_interval(100);
```

g) Call the function wifi_config_set_dtim_interval(interval) to set the DTIM interval and the unit is beacon interval, as shown below.

```
wifi_config_set_dtim_interval(10);
```

h) Apply the configuration by calling the function wifi_config_reload_setting(), as shown below.

```
wifi_config_reload_setting();
```

4.7 Wi-Fi Connection Support

The API is for connection support in AP mode:

1) Get the list of stations associated with the device in AP mode.

Call the function wifi_connection_get_sta_list() to get the information of stations associated with the AP. An example implementation is shown below.

```
uint8_t i;
uint8_t status = 0;
wifi_sta_list_t list[WIFI_MAX_NUMBER_OF_STA];
uint8_t size = 0;
status = wifi_connection_get_sta_list(&size, list);
printf("stalist size=%d\n", size);
for (i = 0; i < size; i++)
{
printf("%d\n", i);
printf("last_tx_rate: MCS=%d, LDPC=%d, MODE=%d\n",
(list[i].last_tx_rate.field.mcs),
(list[i].last_tx_rate.field.ldpc),
(list[i].last_tx_rate.field.mode));
printf("last_rx_rate: MCS=%d, LDPC=%d, MODE=%d\n",
(list[i].last_rx_rate.field.mcs),
(list[i].last_rx_rate.field.ldpc),
(list[i].last_rx_rate.field.mode));
printf("rssi_sample.LastRssi0)=%d\n",
(int)(list[i].rssi_sample.last_rssi));
printf("rssi_sample.AvgRssi0X8=%d\n",
(int)(list[i].rssi_sample.average_rssi));
printf("addr=%02x:%02x:%02x:%02x:%02x:%02x\n", list[i].mac_address[0],
list[i].mac_address[1], list[i].mac_address[2],
list[i].mac_address[3], list[i].mac_address[4],
list[i].mac_address[5]);
printf("power_save_mode=%d\n",
(unsigned int)(list[i].power_save_mode));
printf("bandwidth=%d\n", (unsigned int)(list[i].bandwidth));
printf("keep_alive=%d\n", (unsigned int)(list[i].keep_alive));
}
```

2) Disconnect a station associated with the device in AP mode.

Call the function wifi_connection_disconnect_sta(address) to disconnect a given station, as shown below.

```
unsigned char addr[WIFI_MAC_ADDRESS_LENGTH] = {0};
wifi_conf_get_mac_from_str((char *)addr, "de:86:59:6c:cf:06");
wifi_connection_disconnect_sta(addr);
```

4.8 Scan

Scan modes and scan options are selected when the scan function is executed. The modes and options are shown in Table 7 and Table 8.

Table 7.Scan Mode

Value	Description
0	Full scan
1	Reserved

Table 8.Channels Supported in 2.4G and 5G

Region	Channels	
00	2.4G	CH1-14 active scan
	5G	CH36-64 active scan CH100-165 active scan
US	2.4G	CH1-11 active scan
	5G	CH36-48 active scan CH52-64 passive scan, DFS CH100-140 passive scan, DFS CH149-165 active scan
CN	2.4G	CH1-13 active scan
	5G	CH36-48 active scan CH52-64 passive scan, DFS CH149-165 active scan
JP	2.4G	CH1-14 active scan
	5G	CH36-48 active scan CH52-64 passive scan, DFS CH100-140 passive scan, DFS

4.9 Scan APIs

The following table shows the APIs to set the platform to AP mode.

Table 9. Scan APIs

API	Description
wifi_connect_scan_init (wifi_scan_list_item_t *ap_list, uint32_t max_count)	This function should be called before the function wifi_connection_start_scan() is called, and it should be called only once to initialize the scan to store the scanned AP list in the ap_list buffer.
wifi_connection_start_scan (uint8_t *ssid, uint8_t ssid_length, uint8_t *bssid, uint8_t scan_mode, uint8_t scan_option)	This function starts the Wi-Fi scanning based on scan mode and scan option (see Table 7).
wifi_connection_stop_scan (void)	This function stops the Wi-Fi scanning triggered by wifi_connection_start_scan().
wifi_connection_scan_deinit (void)	This function de-initializes the scan table. When the scan is finished, wifi_connection_scan_deinit() should be called to unload the buffer from the driver. After that, the data in the ap_list can be processed by user applications safely, and then another scan can be initialized by calling wifi_connection_scan_init(). Not calling the wifi_connection_scan_deinit() function may cause failure if another task attempts to call wifi_connection_scan_init().
wifi_config_set_multi_channel(uint8_t port, uint8_t *channel, uint8_t channel_len)	This function starts the Wi-Fi scanning based on input channel list. It only scans the spcific channels defined in the list.

4.10 Using the wifi_connection_scan_init() API

1) Define a buffer g_ap_list to store the scan results. The Maximum of scan list is 8. Currently, support 8 BSS by compile time

```
uint8_t size = 8;
wifi_scan_list_item_t g_ap_list[size] = {0};
```

2) Call the function wifi_connection_scan_init() to initialize the buffer in the Wi-Fi module. The scan result is recorded in g_ap_list in descending order of the RSSI. The number of APs to detect is limited by size, which is 30 in this example.

```
wifi_connection_scan_init(g_ap_list, size);
```

3) Call the function wifi_connection_start_scan() to start the scan.

```
wifi_connection_start_scan(NULL, 0, NULL, 0, 0);
```

4) Call the function wifi_connection_stop_scan() to stop the scan. The parameters are SSID, length of ssid, BSSID, scan mode and scan option respectively, and only scan option is supported.

```
wifi_connection_stop_scan();
```

5) Call the function wifi_connection_scan_deinit() to de-initialize the scan and to unload the buffer from the driver

```
wifi_connection_scan_deinit();
```

4.11 Wi-Fi Connection Support

The connection APIs are used to manage the link status, such as disconnect from the AP, disconnect the station, get the link status, get the station list, start/stop the scan and register an event handler for scan, connect, or disconnect events. The connection APIs use an in-band mechanism and must be called after the OS task scheduler has started, to ensure the in-band task is running. In the STA mode, the device can disconnect from the AP, get the link status, start or stop the scan and register an event handler for scan, connect or disconnect events.

- Get the link status.

  Call the function wifi_connection_get_link_status() to get the link status in STA mode, as shown below.

```
uint8_t status = 0;
uint8_t link = 0;
status = wifi_connection_get_link_status(&link);
if (link == 1) {
    printf("link=%d, the station is connecting to an AP router.\n", link);
}else if (link == 0) {
    printf("link=%d, the station doesn’t connect to an AP router.\n", link);
```

- Disconnect from the AP.

  Call the function wifi_connection_disconnect_ap() to disconnect the station from the AP router, as shown below.

```
uint8_t status = 0;
status = wifi_connection_disconnect_ap();
```

- Register or unregister an event handler. The event handlers to register or unregister events are listed in Table 4. The table shows the supported events generated by the Wi-Fi driver. The events will be sent to handlers registered by the upper layer.

```
status = wifi_connection_register_event_handler(WIFI_EVENT_IOT_CONNECTED,(wifi_event_handler_t)
user_event_callback);
status = wifi_connection_register_event_handler(WIFI_EVENT_IOT_SCAN_COMPLETE,
(wifi_event_handler_t) user_event_callback);
status = wifi_connection_register_event_handler(WIFI_EVENT_IOT_DISCONNECTED, (wifi_event_handler_t)
user_event_callback);


/* Unregister event handler */
status = wifi_connection_unregister_event_handler(WIFI_EVENT_IOT_CONNECTED, (wifi_event_handler_t)
user_event_callback);
status = wifi_connection_unregister_event_handler(WIFI_EVENT_IOT_SCAN_COMPLETE,
(wifi_event_handler_t) user_event_callback);
status = wifi_connection_unregister_event_handler(WIFI_EVENT_IOT_DISCONNECTED,
(wifi_event_handler_t) user_event_callback);
```

5 Wi-Fi Onboarding with Smart Connection through BLE GATT

The MT793X IoT SDK provides the smart connection feature, which allows user to configure the device through BLE GATT connection to connect to the wireless network. The process and demonstration of Wi-Fi onboarding through BLE GATT are described as follows.

1) Components required to accomplish the onboarding process:
   - a. One MT793X device as target to demonstrate
   - b. Peer device for user to set information of Wi-Fi connection, for example, SSID and password of the AP router. The peer device is installed with nRF, an application package with generic BLE function and transmission ability, so that the IoT device built with MediaTek MT793X IoT SDK receives the information to connect to the specified wireless network through Wi-Fi.

2) Onboarding process, demonstrated with application powered by Android:
   - a. The BLE advertising is first started on MT793X side.
   - b. User start the connection and set the information of Wi-Fi connection through the nRF application. Here the APK of Android system is used for the demonstration, where the BLE connection setup and parameter setup are included.
   - c. The information of Wi-Fi connection is received by smart connection application where the application is integrated with Wi-Fi connection APIs introduced in the previous section. Once Wi-Fi is connected, you are ready to use the wireless function. The code for smart connection through BLE GATT is under /project/apps/*_sdk_demo/src/ble_smtcn.c.

3) Configuration for project
   - a. Please enable “MTK_BLE_SMTCN_ENABLE” in /project/apps/*_sdk_demo/GCC/feature.mk

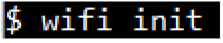
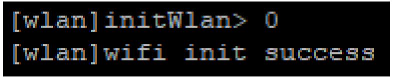
5.1 Procedure of Using Wi-Fi Onboarding with HDK

An application powered by Android is used to demonstrate the procedure, which involves the following steps.

1. Setting up before performing onboarding
2. Completing the connection of BLE GATT, whose process includes connecting without bonding, bonding, and then connecting
3. Filling in the Wi-Fi information in user interface
4. Checking the results of Wi-Fi onboarding

5.1.1 Set up before Performing Wi-Fi Onboarding

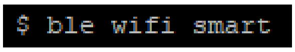
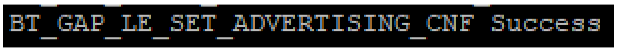
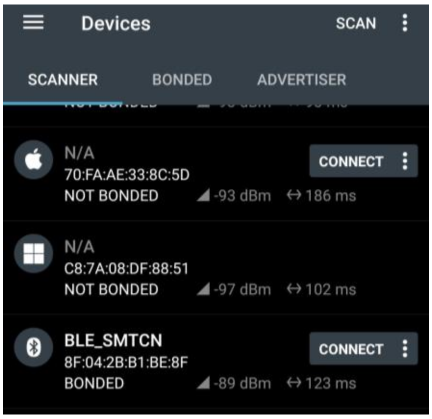
Step	Operation	Success Keyword
1	Make sure you set MTK_BLE_SMTCN_ENABLE as enabled	
2	Use the command "ble init" to initialize BLE `$ ble init`	BT_POWER_ON_CNF Success `BT_POWER_ON_CNF Success`
3	Use the command "wifi init" to start Wi-Fi and supplicant	wifi init success

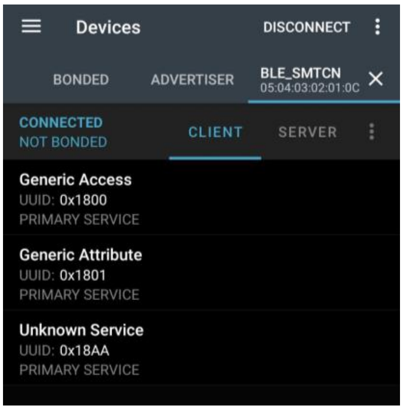
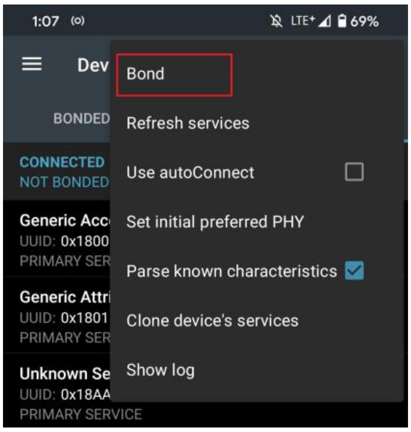
Step	Operation	Success Keyword
	 `$ wifi init`	 `[wlan]initWlan> 0` `[wlan]wifi init success`
4	Download and install the "nRF Connect" app on your phone	

5.1.2 Connect and Bond

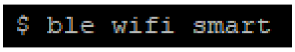
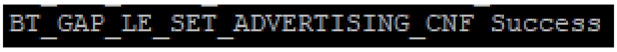
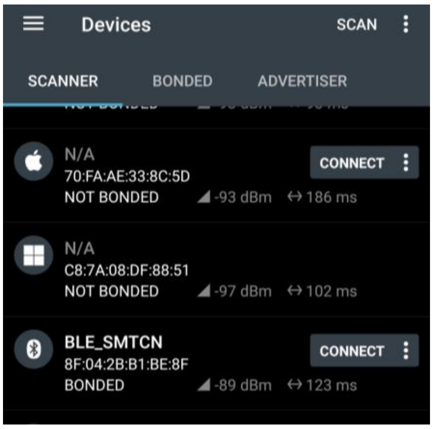
No matter the peer device is with the MT793X built-in or not, the Wi-Fi onboarding service can operate. There are two kinds of operations. Choose either one of the operations to connect to the MT793X**.** If reconnection is required, complete the whole process of Connect without Bonding again.

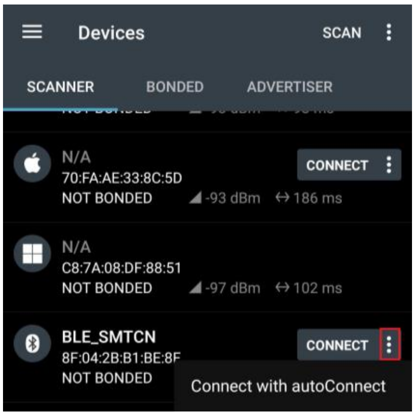
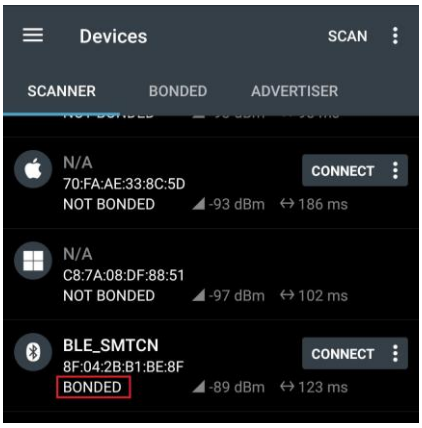
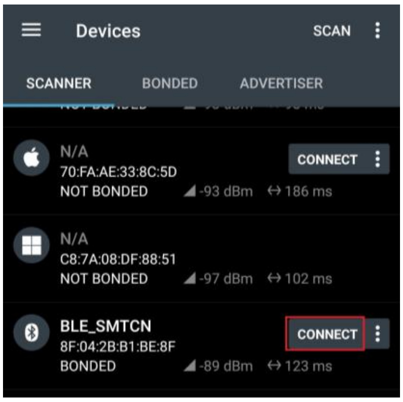
5.1.3 Connect without Bonding

Step	Operations	Description
1	Use the command "ble wifi smart" to start advertising  `$ ble wifi smart`	Keyword of success: BT_GAP_LE_SET_ADVERTISING_CNF Success  `BT_GAP_LE_SET_ADVERTISING_CNF Success` The default name of BLE device is 'BLE_SMTCN'
2	Scan for the BLE device by nRF Connect and then connect it	 Devices SCAN SCANNER BONDED ADVERTISER N/A 70:FA:AE:33:8C:5D NOT BONDED -93 dBm 186 ms CONNECT N/A C8:7A:08:DF:88:51 NOT BONDED -97 dBm 102 ms BLE_SMTCN 8F:04:2B:B1:BE:8F BONDED -89 dBm 123 ms CONNECT

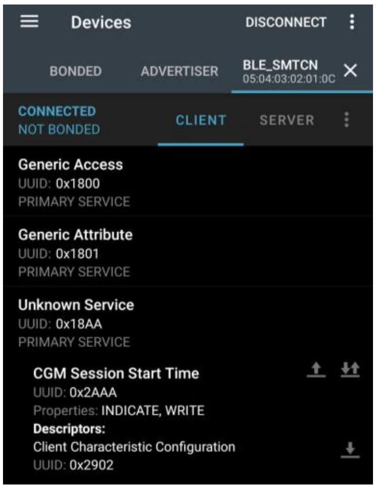
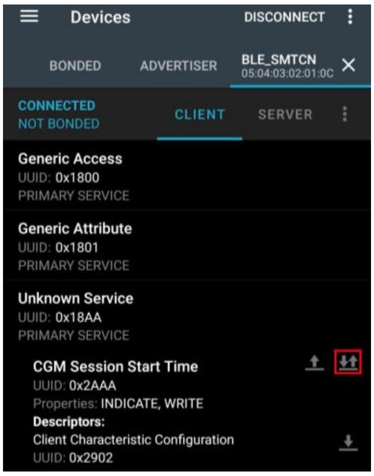
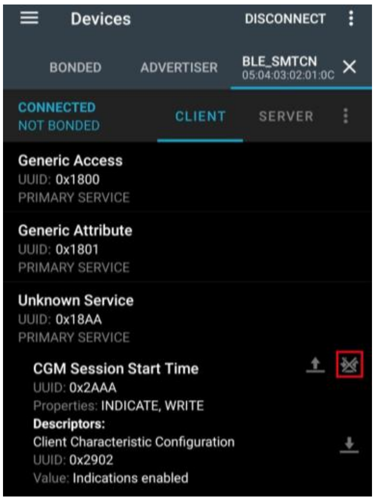
Step	Operations	Description
3	After the connection succeeds, you can find the supported service list of the device	 Devices DISCONNECT BONDED ADVERTISER BLE_SMTCN 05:04:03:02:01:0C CONNECTED NOT BONDED CLIENT SERVER Generic Access UUID: 0x1800 PRIMARY SERVICE Generic Attribute UUID: 0x1801 PRIMARY SERVICE Unknown Service UUID: 0x18AA PRIMARY SERVICE
4 (Optional)	You can select the button at the upper right corner. Then, you can find the Bond button. Select the button to bond with the MT793X.	 1:07 LTE+ 69% Dev BONDED CONNECTED NOT BONDED Generic Acc UUID: 0x1800 PRIMARY SER Generic Attri UUID: 0x1801 PRIMARY SER Unknown Se UUID: 0x18AA PRIMARY SERVICE Bond Refresh services Use autoConnect □ Set initial preferred PHY Parse known characteristics ☑ Clone device's services Show log

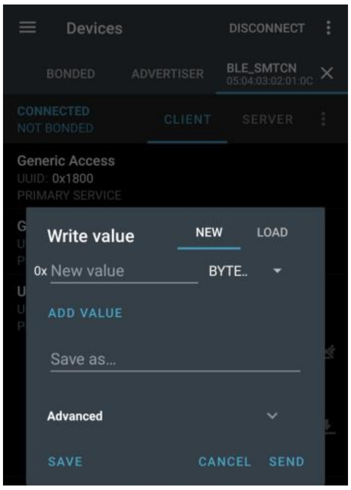
5.1.4 Bond and Then Connect

Step	Operations	Description
1	Use the command "ble wifi smart" to start advertising  `$ ble wifi smart`	Keyword of success: BT_GAP_LE_SET_ADVERTISING_CNF Success  `BT_GAP_LE_SET_ADVERTISING_CNF Success` The default name of BLE device is 'BLE_SMTCN'
2	Scan for the BLE device by nRF Connect and then connect it	 Devices SCAN SCANNER BONDED ADVERTISER N/A 70:FA:AE:33:8C:5D NOT BONDED -93 dBm ↔ 186 ms CONNECT N/A C8:7A:08:DF:88:51 NOT BONDED -97 dBm ↔ 102 ms BLE_SMTCN 8F:04:2B:B1:BE:8F BONDED -89 dBm ↔ 123 ms CONNECT

Step	Operations	Description
3	Bond with the device	 Devices SCAN SCANNER BONDED ADVERTISER CONNECT CONNECT Connect with autoConnect N/A 70:FA:AE:33:8C:5D NOT BONDED -93 dBm 186 ms N/A C8:7A:08:DF:88:51 NOT BONDED -97 dBm 102 ms BLE_SMTCN 8F:04:2B:B1:BE:8F NOT BONDED  Devices SCAN SCANNER BONDED ADVERTISER CONNECT CONNECT N/A 70:FA:AE:33:8C:5D NOT BONDED -93 dBm 186 ms N/A C8:7A:08:DF:88:51 NOT BONDED -97 dBm 102 ms BLE_SMTCN 8F:04:2B:B1:BE:8F BONDED -89 dBm 123 ms
4	Connect the device	 Devices SCAN SCANNER BONDED ADVERTISER CONNECT CONNECT N/A 70:FA:AE:33:8C:5D NOT BONDED -93 dBm 186 ms N/A C8:7A:08:DF:88:51 NOT BONDED -97 dBm 102 ms BLE_SMTCN 8F:04:2B:B1:BE:8F BONDED -89 dBm 123 ms
5	After the connection succeeds, you can find the supported service list of the device	

5.1.5 Configure Wi-Fi Onboarding Parameter by Enabling Indication and Writing Value

Step	Operations	Description
1	Select "Unknown Service". There is only one characteristic value of the service	 Devices DISCONNECT BONDED ADVERTISER BLE_SMTCN 05:04:03:02:01:0C CONNECTED NOT BONDED CLIENT SERVER Generic Access UUID: 0x1800 PRIMARY SERVICE Generic Attribute UUID: 0x1801 PRIMARY SERVICE Unknown Service UUID: 0x18AA PRIMARY SERVICE CGM Session Start Time UUID: 0x2AAA Properties: INDICATE, WRITE Descriptors: Client Characteristic Configuration UUID: 0x2902
2	Select the button to enable indication	 Devices DISCONNECT BONDED ADVERTISER BLE_SMTCN 05:04:03:02:01:0C CONNECTED NOT BONDED CLIENT SERVER Generic Access UUID: 0x1800 PRIMARY SERVICE Generic Attribute UUID: 0x1801 PRIMARY SERVICE Unknown Service UUID: 0x18AA PRIMARY SERVICE CGM Session Start Time UUID: 0x2AAA Properties: INDICATE, WRITE Descriptors: Client Characteristic Configuration UUID: 0x2902  Devices DISCONNECT BONDED ADVERTISER BLE_SMTCN 05:04:03:02:01:0C CONNECTED NOT BONDED CLIENT SERVER Generic Access UUID: 0x1800 PRIMARY SERVICE Generic Attribute UUID: 0x1801 PRIMARY SERVICE Unknown Service UUID: 0x18AA PRIMARY SERVICE CGM Session Start Time UUID: 0x2AAA Properties: INDICATE, WRITE Descriptors: Client Characteristic Configuration UUID: 0x2902 Value: Indications enabled

Step	Operations	Description
3	Write value by selecting this button	 Devices DISCONNECT BONDED ADVERTISER BLE_SMTCN 05:04:03:02:01:0C CONNECTED NOT BONDED CLIENT SERVER Generic Access UUID: 0x1800 PRIMARY SERVICE Write value NEW LOAD 0x New value BYTE.. ADD VALUE Save as... Advanced SAVE CANCEL SEND
4	Configure three values for onboarding	Please refer to Table 10. Wi-Fi Onboarding Content Configuration through BLE.

User need to configure three values through BLE, which is composed in TLV format {TYPE, LEN, DATA}. All three types of configurations are required. The type of configuration and its detailed format are shown in the following table:

Table 10. Wi-Fi Onboarding Content Configuration through BLE

Content	Format	Keyword for Success
SSID	01XXYYYYYYYYYYYY	If write SSID is successful, you find the keyword below. ble_smtcn_parse_data, ssid = "SSID you set"
PASSWD	02AABBBBBBBB	If write PASSWD is successful, you find the keyword below. ble_smtcn_parse_data, pw = "PASSWD you set"
SEC Mode	0302CCDD	If write secure mode is successful, you find the keyword below. ble_smtcn_parse_data, auth = x, encryt = x

Table 11. Wi-Fi Onboarding: SSID Configuration

Field	Description	Remark
01	Represent SSID	N/A
XX	Length of SSID in binary format	N/A
YYYYYYYYYYYY	SSID in binary format	N/A

- An example of SSID configuration is shown below:
    - For SSID of AP as "Lawrance_Test_2.4G"
    - The binary format of "Lawrance_Test_2.4G" is "4c 61 77 72 61 6e 63 65 5f 54 65 73 74 5f 32 2e 34 47" with length of 18 bytes, where hex value of 18 is 12.
    - The input for SSID is "01**12**4c617772616e63655f546573745f322e3447" in application.

Table 12. Wi-Fi Onboarding: PASSWD Configuration

Field	Description	Remark
02	Represent PASSWD	N/A
AA	Length of PASSWD in binary format	N/A
BBBBBBBB	PASSWD in binary format	N/A

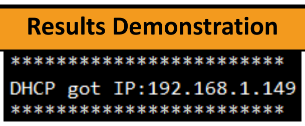
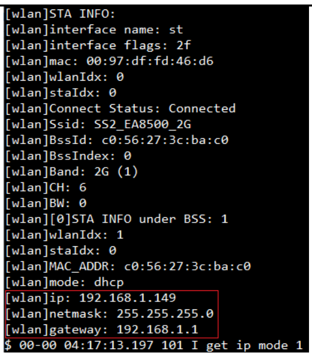
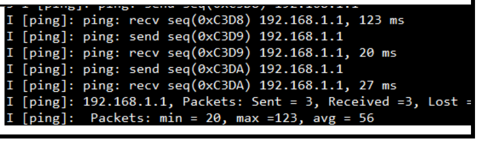
- An example of PASSWD configuration is shown below:
    - PASSWD is "12345678".
        - Binary format of "12345678" is "31 32 33 34 35 36 37 38", with length of 8. "8" in hex is "08".
        - The input for PASSWD is "02083132333435363738" in application.
    - No PASSWD for open connection
        - No password; length is 0.
        - The input for PASSWD is "0200" in application

Table 13. Wi-Fi Onboarding: SEC Mode Configuration

Field	Description	Remark
03	Represent the secure mode	N/A
02	Length	N/A
CC	Authentication mode	0x00 : WIFI_AUTH_MODE_OPEN 0x01 : WIFI_AUTH_MODE_SHARED 0x02 : WIFI_AUTH_MODE_AUTO_WEP 0x03 : WIFI_AUTH_MODE_WPA 0x04 : WIFI_AUTH_MODE_WPA_PSK 0x05 : WIFI_AUTH_MODE_WPA_None 0x06 : WIFI_AUTH_MODE_WPA2 0x07: WIFI_AUTH_MODE_WPA2_PSK 0x08 : WIFI_AUTH_MODE_WPA_WPA2 0x09 : WIFI_AUTH_MODE_WPA_PSK_WPA2_PSK
DD	Encryption type	0x00 : WIFI_ENCRYPT_TYPE_WEP_ENABLED 0x01 : WIFI_ENCRYPT_TYPE_WEP_DISABLED 0x02 : WIFI_ENCRYPT_TYPE_WEP_KEY_ABSENT 0x03 : WIFI_ENCRYPT_TYPE_WEP_NOT_SUPPORTED 0x04 :WIFI_ENCRYPT_TYPE_TKIP_ENABLED 0x06 : WIFI_ENCRYPT_TYPE_AES_ENABLED 0x07 : WIFI_ENCRYPT_TYPE_AES_KEY_ABSENT 0x08 : WIFI_ENCRYPT_TYPE_TKIP_AES_MIX 0x09 : WIFI_ENCRYPT_TYPE_TKIP_AES_KEY_ABSENT 0x0A : WIFI_ENCRYPT_TYPE_GROUP_WEP40_ENABLED 0x0B : WIFI_ENCRYPT_TYPE_GROUP_WEP104_ENABLED

- An example of SEC Mode configuration is shown below:
  - For AP of "Lawrance_Test_2.4G", which supports WPA2 (0x07) and uses AES encryption (0x06).
    - The input for SEC Mode is "03020706".
  - For AP that only supports open mode.
    - The input for SEC mode is "03020000"

5.1.6 Check Results of Wi-Fi Onboarding

Step	Operation	Results Demonstration
1	After you write the three values, Wi-Fi is automatically connected. After the connection succeeds, the log shows the DHCP IP.	 ************************ DHCP got IP:192.168.1.149 ************************
2 (optional)	Use "wifi info" to check DHCP IP and other details of the Wi-Fi connection.	 [wlan]STA INFO: [wlan]interface name: st [wlan]interface flags: 2f [wlan]mac: 00:97:df:fd:46:d6 [wlan]wlanIdx: 0 [wlan]staIdx: 0 [wlan]Connect Status: Connected [wlan]Ssid: SS2_EA8500_2G [wlan]BssId: c0:56:27:3c:ba:c0 [wlan]BssIndex: 0 [wlan]Band: 2G (1) [wlan]CH: 6 [wlan]BW: 0 [wlan][0]STA INFO under BSS: 1 [wlan]wlanIdx: 1 [wlan]staIdx: 0 [wlan]MAC_ADDR: c0:56:27:3c:ba:c0 [wlan]mode: dhcp [wlan]ip: 192.168.1.149 [wlan]netmask: 255.255.255.0 [wlan]gateway: 192.168.1.1 $ 00-00 04:17:13.197 101 I get ip mode 1
3 (optional)	Use the command "ping <gateway>" to send packets to connected AP.	 I [ping]: ping: recv seq(0xC3D8) 192.168.1.1, 123 ms I [ping]: ping: send seq(0xC3D9) 192.168.1.1 I [ping]: ping: recv seq(0xC3D9) 192.168.1.1, 20 ms I [ping]: ping: send seq(0xC3DA) 192.168.1.1 I [ping]: ping: recv seq(0xC3DA) 192.168.1.1, 27 ms I [ping]: 192.168.1.1, Packets: Sent = 3, Received =3, Lost = I [ping]: Packets: min = 20, max =123, avg = 56

6 Wi-Fi and Bluetooth Coexistence

Wi-Fi and Bluetooth coexistence mode is time-division duplexing (TDD) mode. On hardware side, Wi-Fi and Bluetooth use one shared antenna to transmit or receive alternately, so there is a switch component called SPDT, which is used to switch antenna controlled by Bluetooth PTA grant signal. By default the antenna parks on Wi-Fi side. In the software flow, Bluetooth takes charge of time division schedule, while time domain is divided to Wi-Fi time and Bluetooth time.

6.1 Configure the TDD Mode

To configure the TDD mode.
First, set the SPDT control pin (antsel ctrl) and coex mode setting in the eFuse component.
The following table shows the eFuse values in different types of hardware.

Hardware type	eFuse value
BGA	ANTSEL_CTRL0 0x3D8 = 8'h8D ANTSEL_CTRL1 0x3D9 = 8'h03 ANTSEL_CTRL2 0x3DA = 8'h00
QFN	ANTSEL_CTRL0 0x3D8 = 8'h8D ANTSEL_CTRL1 0x3D9 = 8'h04 ANTSEL_CTRL2 0x3DA = 8'h00

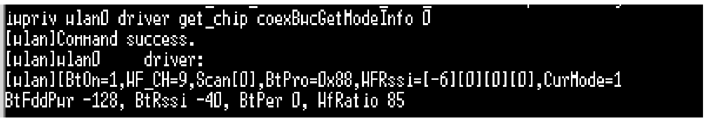
Second, set the driver to the eFuse mode, please refer section 3.1 Configuration Option.

ANTSEL CTRL information:

ANTSEL	Bitmap
ANTSEL_CTRL0	ANT_CTRL_COEX_TDD_EN BIT(0) ANT_CTRL_COEX_TDD_PIN_SEL BIT(1) // 0: 1 pin, 1: 2 pins ANT_CTRL_COEX_TDD_SHARE_WITH_BT0_EN BIT(2) ANT_CTRL_COEX_TDD_POL_SWP BIT(3) ANT_CTRL_VALID_BIT BIT(7)
ANTSEL_CTR1	ANT_CTRL_COEX_TDD_P_ANT_NO BITS(0,3) ANT_CTRL_COEX_TDD_N_ANT_NO BITS(4,7)

6.2 Common Command

Command	Field description

iwpriv wlan0 driver "get_chip coexBwcGetModeInfo 0"

`iwpriv wlan0 driver get_chip coexBwcGetModeInfo 0`
`[wlan]Command success.`
`[wlan]wlan0     driver:`
`[wlan][BtOn=1,WF_CH=9,Scan[0],BtPro=0x88,WFRssi=[-6][0][0][0],CurMode=1`
`BtFddPwr -128, BtRssi -40, BtPer 0, WfRatio 85`
	BtOn = Bluetooth state
	WF_CH = Wi-Fi channel
	BtPro = Bluetooth profile
	CurMode = coex mode

Field	Value
BtOn	OFF : 0 ON : 1
BtPro	SCO BIT(0) A2DP BIT(1) LINK_CONNECTED BIT(2) HID BIT(3) PAGE BIT(4) INQUIRY BIT(5) ESCO BIT(6) MULTI_HID BIT(7) BLE_SCAN BIT(8) ADV BIT(9) ADV_DIRECT BIT(10) A2DP_SINK BIT(11)
CurMode	Non coex : 0 TDD : 1

6.3 TDD Scenarios

The trigger condition of coexistence mode is Wi-Fi 2.4G connection and BLE on. this section shows how to setup/check BLE profile.

For BLE mesh mode, DUT needs to switch to mesh mode via the following command. Then reboot DUT.

- $ ble boot mode mesh

Related BLE parameters are listed as follows

- BLE Scan window 30ms, interval 50ms
- ADV length 1.875ms, interval 10ms
- BLE link length 1.25ms, interval 7.5ms

Scenario	BLE Setup step	Final Bluetooth profile
Wi-Fi + BLE on (mesh mode)	1. BLE initialization $ ble init	0x300/0x100
Wi-Fi + Provisioner prov and config 1* DUT by PB-GATT, bubble on/off	1. Follow Wi-Fi + BLE on setup step 2. In provisioner, use MeshProv APP to establish network (GATT bearer) 3. In provisioner, use MeshProv APP to connect to DUT	0x308/0x108
Wi-Fi + Provisioner prov and config 1* DUT by PB-ADV, bubble on/off	1. Follow Wi-Fi + BLE on setup step 2. In provisioner, use MeshProv APP to establish network (ADV bearer) 3. In provisioner, use MeshProv APP to connect to DUT	0x300/0x100

For BLE GATT mode, DUT needs to switch to GATT mode via the following command. Then, reboot DUT.

- $ ble boot mode gatts

Related BLE parameters are listed as follows

- BLE ADV length 3.75, interval 122.5ms
- BLE link length 1.875ms, interval 11.25ms

Scenario	Setup step	Final Bluetooth profile

Wi-Fi + BLE on (GATT mode with ADV)	1. BLE initialization
	$ ble init
	2. Start ADV
	$ ble wifi smart
		Bluetooth profile 0x200

1 GATT Link-LE 1M	1. Follow Wi-Fi + BLE on setup steps 2. Use the nRF Connect app to connect to DUT 3. Update to LE 1M PHY rate using the app “Set Prefer Phy”	Bluetooth profile 0x008
1 GATT Link-LE 2M	1. Follow Wi-Fi + BLE on setup steps 2. Use the nRF Connect app to connect to DUT 3. Update to LE 2M PHY rate using the app “Set Prefer Phy”	Bluetooth profile 0x008

In BLE GATT mode, DUT needs to switch to GATT mode via the following command. Then, reboot DUT.

- $ ble boot mode gatts

Related BLE parameters are listed as follows

- BLE Scan window 200ms, interval 200ms
- BLE link interval 30ms

Scenario	Setup step	Final Bluetooth profile

Wi-Fi + BLE on (GATT mode with BLE scan)	1. BLE initialization
	$ ble init
	2. Enable BLE scan with full scan
	$ ble gap start_scan 1 0140 0140 0 0 2
		0x100

Wi-Fi + 6 BLE	1. Follow Wi-Fi + BLE on setup steps
	2. Disable BLE full scan
	$ ble gap stop_scan
	3. The 6 BLE devices initialize BLE and start ADV
	$ ble init
	$ ble gatts add jitter srv
	$ ble wifi smart
	4. Connect to the 6 BLE devices in order with addr and interval
	$ ble gap advanced_conn 0010 0010 0 0 <addr> 0 0018 0018 0000 012C 0004 0004
	5. Check connection list
	$ ble list connection
	6. Reconfigure data length of server
	$ ble gattc jitter auto setup
	7. Request server to send 128 bytes indication every 3 seconds
	$ ble gattc jitter auto run
	8. Show jitter information of each link
	$ ble gattc show jitter info
	9. Request server to stop sending data (when test is done)
	$ ble gattc jitter auto stop
		0x088

7 Wi-Fi System Low Power Mode

This chapter introduces Wi-Fi system low power mode of the MT793X.

7.1 Legacy Wi-Fi Power Saving

The MT793X supports 3 Wi-Fi power management modes in STA mode:

- Constantly Awake Mode (**CAM**)

  The 802.11 station never turns the radio off. The 802.11 station will usually be in this mode if the portable device is operating from an AC power source.
- Max Power Save Polling (**MAXPSP**)

  This PS mode requires that the 802.11 station turn off the radio for as long as possible without losing BSS network connectivity, resulting in the greatest power savings at the sake of network performance.

  STA always stays in PS state, and send PS-Poll to AP if there is buffered data at AP side.
- Fast Power Save Polling (**FastPSP**)

  This PS mode requires that the 802.11 station turn off the radio for small periods in order to provide optimal network performance.

  STA will switch between PS and active state. STA will notify AP the STA's power state by the PM bit in the frame control.

After connected to AP, the MT793X enters power mode according to STA NVDM keyword: **PSMode**.
In FastPSP mode, the MT793X starts a decision window to check Wi-Fi's data sending/receiving counter and decides whether to enter PS state or not. If there is no any sending and receiving during the decision window, the MT793X will enter PS state. Otherwise, MT793X will keep at active state and start a new decision window.

API for Legacy Wi-Fi Power Saving protocol

CLI Commands	API	Description
wifi config set ps_mode <0/1/2> wifi config get ps_mode	int32_t **wifi_config_set_power_save_mode**(uint8_t ps_mode)	Configure Wi-Fi power saving mode. uint8_t ps_mode ● 0: CAM ● 1: MAXPSP ● 2: FastPSP

NVDM profile for Legacy Wi-Fi Power Saving protocol

Group	Data item	Description
STA	PSMode	0 to 2, apply to STA mode after connecting to AP. ● 0: CAM ● 1: MAXPSP ● 2: FastPSP

Group	Data item	Description
wifi	EnterPmInterval_lp EnterPmInterval_conn	Decision window of fastPSP mode. If there is no TX and RX during the decision window, MT793X enters power saving mode. Otherwise, MT793X will keep at active state and start a new decision window.

7.2 Wi-Fi Auto Suspend

When there is no Wi-Fi activity, including sending/receiving data and sending command to firmware, for a long time, Wi-Fi driver will enter suspend mode and apply settings to reduce more power consumption.

Auto suspend mechanism

Driver will start a monitor window for 400ms after Wi-Fi driver idle. If no Wi-Fi activity happens during this window, Wi-Fi driver will enter suspend mode.

If any activity happens during the monitor window, driver will stop monitoring and re-start a new window when the next time Wi-Fi driver idle.

Driver will apply the following setting when entering suspend mode.

Setting	Config method	Description
EnterPmInterval_lp	NVDM	Driver will apply a shorter FastPSP decision window: EnterPmInterval_lp(default = 5ms) when Wi-Fi driver enters suspend mode. MT793X can enter power saving mode as soon as possible if only EAPOL, ARP request, or Null frame are received. ARP request and Null frame will be offloaded to firmware to avoid wake up CM33 too frequently.
listen interval	API	Driver will apply listen interval configured by user. The value of listen interval should be a multiple of AP's DTIM period.

Driver will apply the following setting when leaving suspend mode.

Setting	Config method	Description
EnterPmInterval_conn	NVDM	When leaving suspend mode, Wi-Fi driver will apply a proper FastPSP decision window: EnterPmInterval_conn(default = 200ms). Wi-Fi driver set a longer duration for FastPSP's decision window to provide a better performance and response latency.

Setting	Config method	Description
Set listen interval to one beacon interval	N/A, Length of beacon interval is decided by associated AP	Driver will apply a beacon interval as listen interval when leaving suspend mode to provide a better response latency when Wi-Fi active.

Related API

CLI Commands	API	Description
wifi config set listen <1~255> wifi config get listen	int32_t wifi_config_set_listen_interval(uint8_t interval) int32_t wifi_config_get_listen_interval(uint8_t *interval)	Set or Get Wi-Fi Listen Interval uint8_t interval • Default = 1. (follow AP's DTIM) • Range from 1 to 255 • Must be multiple of AP's DTIM

7.3 Packet Filter (PF)

User can configure TCP or UDP port number according to the services running on the MT793X. For example:

- DHCP (UDP port 68)
- DNS (UDP port 53)
- MQTT (TCP port 1883/8883)

Note that the MT793X supports total 5 ports for UDP + TCP protocol.

User can configure 0 ~ 32 multicast MAC address to Multicast whitelist by API.
After configuring packet filters above, user can enable WOW mode by API, When WOW is enabled, only

1. management frame.
2. offload feature (ARP, EAPOL).
3. matched PF data frame.

can be received by the MT793X. Any other UC or B/MC data frame will be dropped.

- If above packet filters are not configured before user enable WOW, the MT793X can only receive
  1. management frame.
  2. offload feature (ARP, EAPOL).
  3. WOW magic packet.

  (The magic packet is a frame that is most often sent as a broadcast and that contains anywhere within its payload 6 bytes of all 255 (FF FF FF FF FF FF in hexadecimal), followed by sixteen repetitions of the target computer's 48-bit MAC address, a total of 102 bytes.)

API for Packet Filter

CLI Commands	API	Description
wifi config set wow <0/1> wifi config get wow	int32_t **wifi_config_set_wow**(uint8_t enable) int32_t **wifi_config_get_wow**(uint8_t *mode)	Configure Wi-Fi WOW mode. **uint8_t enable / uint8_t *mode** • 0: Disable • 1 (default): Enable
wifi config set wow_udp <port1> <port2> ... wifi config set wow_udp_del wifi config get wow_udp wifi config set wow_tcp <port1> <port2> ... wifi config set wow_tcp_del wifi config get wow_tcp	int32_t **wifi_config_wow_port**(struct wifi_wow_ports_t *prWowPorts, uint8_t op)	**struct wifi_wow_ports_t *prWowPorts** **struct wifi_wow_ports_t {** **uint8_t len;** **#define MAX_TCP_UDP_PORT 5** **uint16_t ports[MAX_TCP_UDP_PORT];** **};** • support in total 5 UDP and TCP ports. **uint8_t op** • BIT(0) = Protocol ○ 0: UDP port ○ 1: TCP port • BIT(1) reserved • BITS(2,3): Operation ○ 0: set (add UDP or TCP ports to WOW list) ○ 1: unset (delete all UDP or all TCP ports on WOW)
Enable: ● wifi config set mc_address 01-00-5e-00-00-00 01-00-5e-00-00-02 ... Disable: ● wifi config set mc_address (Only support 8 MCast address lists from CLI)	int32_t **wifi_config_set_mc_address** (uint32_t len, void *AddrList)	Configure multicast MAC address whitelist to allow frames with these SA pass the filter. **uint32_t len** • number of MCast. • 0~32 **void *AddrList** (= uint8_t arAddress[len][6]) • array of MCast address list **AddrList** can be NULL if len == 0

CLI Commands	API	Description
wifi config set arp_offload <0/1>	int32_t **wifi_cofig_set_arp_offload**(uint8_t enable)	Enable/Disable ARP offload by API (default enable) When ARP offload is enable, firmware will auto reply ARP request without wakeup CM33 When ARP offload is disable, ARP request will be treated as a normal data frame. In this way, ARP request may be dropped if WOW enable.

Example sequence of API

Scenario	CLI	Result
WOW enable. Drop multicast data frame.	Enable: wifi config set mc_address 00-00-00-00-00-00 wifi config set wow 1 Disable: wifi config set mc_address wifi config set wow 0	User can set a dummy address "00-00-00-00-00-00" to drop all MCdata. (User can also set a specific MC address to receive the MC data frame from this MC address) User can set a null MC address to disable this filter. After WOW enable, MT793X can only receive the following packets: 1. Management / control frame. 2. BC/MC or UC magic packet. 3. The MC data with the correct MC transmit address 4. ARP request if ARP offload is enable. 5. EAPOL (GTK rekey frame).
WOW enable. Drop multicast data frame. Set port wakeup. (DHCP port = 68)	Enable: wifi config set mc_address 00-00-00-00-00-00 wifi config set wow_udp 68 wifi config set wow 1 Disable: wifi config set mc_address wifi config set wow 0	User can set the specific port number of UDP or TCP to allow these data Rx to Host when WOW enable.

8 Wi-Fi System Antenna Diversity Mode

This chapter introduces Wi-Fi system antenna diversity mode of the MT793X.

8.1 Background

The concept of antenna diversity is to add an extra antenna and choose the best one. As Figure 4 shows, the device may receive better signal from AP either on antenna 1 or antenna 2. This relates to its location and environment and may change when it moves. The target is to improve the RX performance in middle and long range, where the RSSI is typically below -62 dBm.

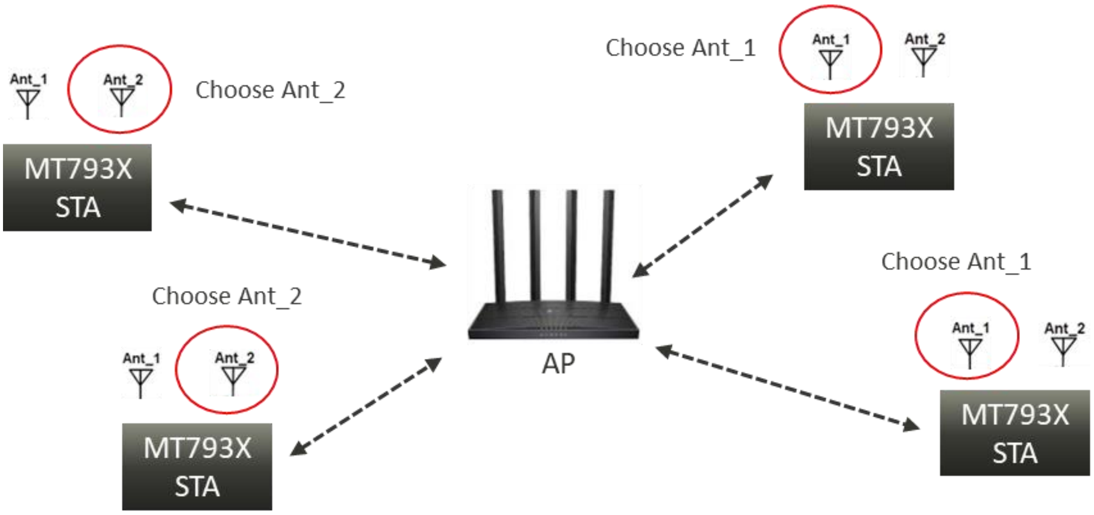


Figure 4. Antenna Diversity Based on The Situation

8.2 RF Configuration

Figure 5 shows the RF configuration required to support antenna diversity. A dedicated pin (GPIO40) is used to control the external SPDT to select antenna 1 or 2 for data transmission. The MT793X automatically controls the signal upon an incoming Wi-Fi packet and determines which is the better antenna to receive the packet.

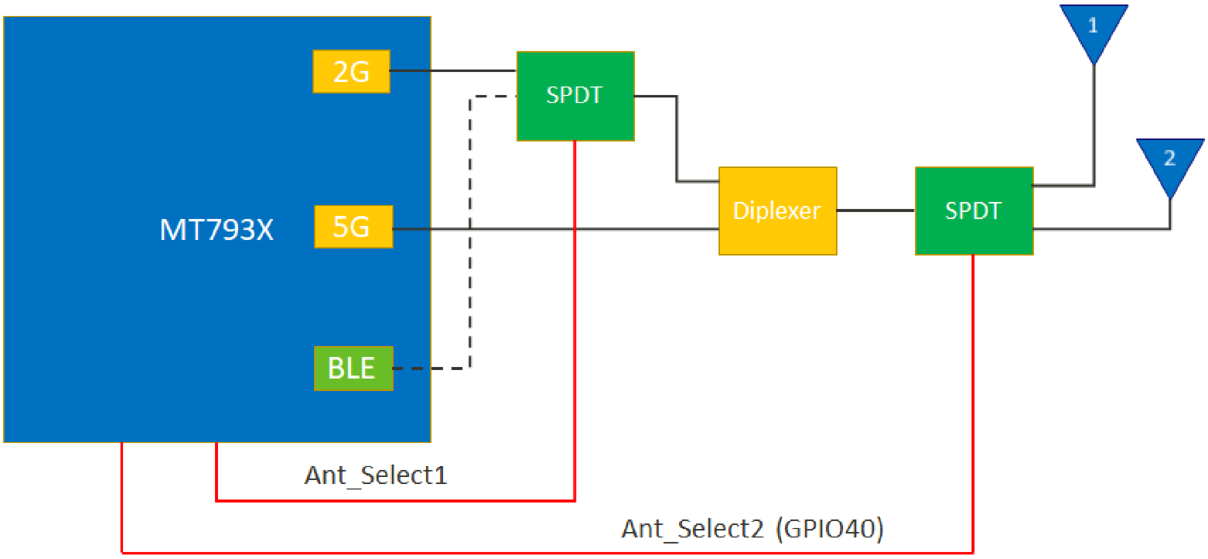


Figure 5. RF Configuration Required

8.3 eFuse Value

Software checks if the hardware supports antenna diversity or not by the value in eFuse offset 0x3DA. If the device is crafted to support antenna diversity, a suitable value should be programmed in the eFuse. Please refer to Figure 6 and the document "MT793X IoT SDK for Wi-Fi Test Tool" to read/write the eFuse value.

Offset Address	Hex Value	Description	Write owner	Value Type
0x3DA	00	ANTSEL_CTRL2	Customer	Option
Offset	**Bit-field**	**Description**		
0x3DA	0:2	SW diversity control with BT signal bit 0x0: WF0 diversity disable 0x1: WF0 diversity enable with BT 0x2: WF0 diversity enable without BT 0x3~0x7: reserved		
0x3DA	3	Antenna config identification 0:Antenna diversity (2 ANT) 1:1 antenna only		
0x3DA	4:7	Reserved		

Figure 6. eFuse Definition for Antenna Diversity

8.4 Antenna Diversity Mode

The antenna diversity works only when the STA is connected to an AP. The antenna diversity mode can be configured either by NVDM option at bootup or the Wi-Fi API at run-time.

8.4.1 NVDM Option

Antenna diversity includes three modes that can be configured as Table 14 lists.

Table 14. NVDM Option for Antenna Diversity

NVDM Group	Configuration Option	Description
wifi	AntDivMode	Antenna diversity mode 0: Force antenna 0 1: Force antenna 1 2: Auto selection

If the NVDM setting is preferred, please configure the value before Wi-Fi initialization. Otherwise, a system reboot like below is required to make the setting take effect.

```
$ config write wifi AntDivMode 2
wifi-AntDivMode = 2
write data item ok
$
$ config read wifi AntDivMode
AntDivMode = 2
$
$ reboot
Reboot Bye Bye Bye!!!!
```

8.4.2 Wi-Fi API

If a run-time control is needed, the Wi-Fi APIs in Table 15 can be used to configure and get the status.

Table 15. Wi-Fi APIs for Antenna Diversity

Wi-Fi API	Description	Wi-Fi CLI
int32_t wifi_config_set_antdiv_mode(uint8_t mode);	Set antenna diversity mode 0: Force antenna 0 1: Force antenna 1 2: Auto selection	wifi config set antmode <value>
int32_t wifi_config_get_antdiv_mode(uint8_t *mode);	Get antenna diversity mode 0: Force antenna 0 1: Force antenna 1 2: Auto selection	wifi config get antmode
int32_t wifi_config_get_antdiv_cur_idx (uint8_t *index)	Set antenna diversity index 0: Antenna 0 1: Antenna 1	wifi config get antidx

8.4.3 Relation between The Settings

The relation between above HW and SW settings are show as Figure 7. If HW does not support or the feature is not enabled by eFuse value, the antenna diversity mode will never take effect. If HW does support, SW will apply the NVDM option first during the initialization. Once the antenna diversity mode is changed by Wi-Fi API at run-time, it will be applied to overwrite the initial setting immediately.

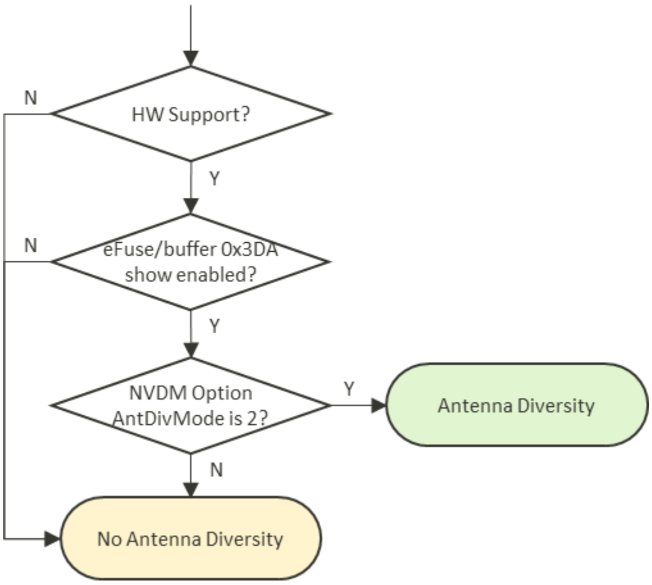

HW Support?
N
Y
eFuse/buffer 0x3DA show enabled?
N
Y
NVDM Option AntDivMode is 2?
Y
N
Antenna Diversity
No Antenna Diversity

Figure 7. Flow to Determine the Antenna Diversity Mode

9 Wi-Fi Sub-SYS reset

Purpose:

If Wi-Fi sub-system encounters a hardware or firmware issue, driver can reset Wi-Fi Sub-SYS only without impacting other Sub-SYS automatically.

Scenario:

- Wi-Fi firmware ASSERT or Exception
- Chip no response (maybe bus hang or WFDMA hang)
- I/O request command or firmware command timeout.

To prevent triggering reset too frequently, driver only allows reset 3 times in 30 seconds. If reset time exceeds the threshold, driver will be blocked after N10 coredump.

10 Wi-Fi Agile Multiband

This chapter introduces Wi-Fi agile multiband feature of the MT793X.

10.1 Background

Wi-Fi agile multiband facilitates efficient use of different frequency bands. This technology provides better management of Wi-Fi network environments and enables Wi-Fi devices to better respond to network changes. Agile multiband leverages existing 802.11kvr features with additional MBO elements to indicate the capability.

- Partial leverage
  - 802.11k: neighbor report and beacon report
  - 802.11v: BSS transition management (BTM) and WNM notification
  - 802.11r: Fast BSS Transition over-the-air (FT OTA)
- MBO original
  - STA non-preferred channel
  - MBO IE

10.2 BSS Transition

There are types of roaming in agile multiband. By sharing the beacon reports, AP can trigger BTM request to the STA with the neighbor report list. STA also can trigger AP to send BTM request by sending BTM query. Besides, STA can set channel preference in advance. The following APIs can help the MT793X in MBO environment.

10.3 Wi-Fi API

If runtime control is needed, the Wi-Fi APIs in Table 16 can be used to trigger the specific function.

Table 16. Wi-Fi APIs for Agile Multiband

Wi-Fi API	Description	Wi-Fi CLI
int32_t wifi_wnm_bss_query(uint8_t value);	Send BTM query value: 80211v transition and transition query reasons	wifi wnm_bss_query <value>
int32_t wifi_neighbor_rep_request(uint8_t *ssid, uint8_t ssid_length);	Send neighbor report request. Give an SSID to indicate a request for the specified SSID. NULL ssid parameter to indicate neighbor report for the current ESS.	wifi neighbor_rep_request <ssid>

Wi-Fi API	Description	Wi-Fi CLI
int32_t wifi_config_set_non_pref_chan (uint8_t port, uint8_t *non_pref_chan, uint8_t non_pref_chan_length)	Send MBO non-preferred channels to the associated AP. non_pref_chan=<oper_class>:<chan>:<preference>:<reason>	wifi config set non_pref_chan 0 <oper_class>:<chan>:<preference>:<reason>

11 Wi-Fi STA Roaming

This chapter introduces Wi-Fi STA Roaming feature of the MT793X.

11.1 Features

Features and limitations of roaming include:

- Triggered by RSSI threshold or beacon miss count
- Same SSID and security mode for APs
- Background scan for full channel or fixed channel
- Over-the-Air FT and Non-FT roaming
- API to adjust RSSI threshold, block time, retry time, retry limit, beacon miss threshold, and beacon timeout period

11.2 Roaming APIs

Table 17. Wi-Fi APIs for Roaming

Wi-Fi API	Description
int32_t wifi_config_set_autoroam(uint8_t port, int32_t value)	Enable or disable auto-roaming 0: Disable 1: Enable
int32_t wifi_config_get_autoroam(uint8_t port, int32_t *value)	Get auto-roaming setting 0: Disable 1: Enable
int32_t wifi_config_set_roam_by_rssi(uint8_t port, int32_t value)	Set roaming to be triggered by RSSI 0: Disable 1: Enable
int32_t wifi_config_get_roam_by_rssi(uint8_t port, int32_t *value)	Get roaming to be triggered by RSSI 0: Disable 1: Enable
int32_t wifi_config_set_roam_by_bcnmiss(uint8_t port, int32_t value)	Set roaming to be triggered by beacon lost 0: Disable 1: Enable
int32_t wifi_config_get_roam_by_bcnmiss(uint8_t port, int32_t *value)	Get roaming to be triggered by beacon lost 0: Disable 1: Enable
int32_t wifi_config_set_roam_rssithreshold(uint8_t port, int32_t value)	Set threshold of RSSI to trigger roaming Default: 0 Value: -127 to 127

Wi-Fi API	Description
int32_t wifi_config_get_roam_rssithreshold(uint8_t port, int32_t *value)	Check the threshold of RSSI to trigger roaming
int32_t wifi_config_set_roam_by_bcnmissthreshold (uint8_t port, int32_t value)	Set counter of beacon lost to trigger roaming Default: 10 Value: 0 to 30
int32_t wifi_config_get_roam_by_bcnmissthreshold (uint8_t port,int32_t *value)	Check the beacon lost counter to trigger roaming
int32_t wifi_config_set_roam_delta(uint8_t port, uint8_t value)	Set delta between two APs to trigger roaming. This would be checked after RSSI delta. Default: 10 Value: 0 to 255
int32_t wifi_config_get_roam_delta(uint8_t port, uint8_t *value)	Check the delta between two APs to trigger roaming
int32_t wifi_config_set_roam_block_time(uint8_t port, int32_t value)	Set the amount of time to avoid ping-pong roaming and trigger roaming again Default: 30 sec Value: 0 to 65535
int32_t wifi_config_get_roam_block_time(uint8_t port, int32_t *value)	Check the time for avoiding ping-pong roaming and for triggering roaming again
int32_t wifi_config_set_roam_maxlock_count(uint8_t port, int32_t value)	Set the number of retries to trigger roaming again when scan does not get SSID of AP Default: 1 Value: 0 to 10
int32_t wifi_config_get_roam_maxlock_count(uint8_t port, int32_t *value)	Check the retry count for triggering roaming again when scan does not get SSID of AP
int32_t wifi_config_set_roam_lock_time(uint8_t port, int32_t value)	Set the amount of time to trigger roaming again while roaming had already been triggered. Default: 2 sec Value: 0 to 180
int32_t wifi_config_get_roam_lock_time(uint8_t port, int32_t *value)	Check the time for triggering roaming again while roaming had already been triggered.
int32_t wifi_config_set_roam_scan_channel(uint8_t port, uint8_t value)	Set full or current channel scan 0: Current channel 1: Full channel
int32_t wifi_config_get_roam_scan_channel(uint8_t port, uint8_t *value)	Get full or current channel scan 0: Current channel 1: Full channel

Wi-Fi API	Description
int32_t wifi_config_set_bto_time(uint8_t port, int32_t value)	Set beacon timeout period to trigger disconnect Default: 5 sec Value: 0 to 180
int32_t wifi_config_get_bto_time(uint8_t port, int32_t *value)	Get beacon timeout period to trigger disconnect
int32_t wifi_config_get_roam_type(uint8_t port, uint8_t *value)	Get roaming type 1: RSSI trigger 2: Beacon timeout trigger
int32_t wifi_config_is_connect_ft_ap(uint8_t port, int *value)	Check whether connected AP is FT or not 0: Non-FT AP 1: FT AP
int32_t wifi_config_get_roam_statistic(uint8_t port, struct roam_statistic_t *roam_stat)	Get the number of successful roaming by RSSI and beacon miss
int32_t wifi_config_clear_roam_statistic(uint8_t port)	Clear the number of successful roaming by RSSI and beacon miss

11.3 Trigger Roaming with RSSI Threshold

This section introduces how to use API to trigger Roaming with RSSI threshold. Refer to the following APIs to set roaming parameters, and these APIs can be set before or after connection.

1) In this example, enable roaming and set RSSI threshold to -60dB, as shown below.
   a) Call `wifi_config_set_autoroam` () to enable roaming feature.

```
wifi_config_set_autoroam(0, 1);
```

   b) Call `wifi_config_set_roam_by_rssi()` to trigger roaming by RSSI threshold.

```
wifi_config_set_roam_by_rssi(0, 1);
```

   c) Use the API to set RSSI threshold (-60dB). If RSSI value is lower than RSSI threshold(-60dB), the device will trigger roaming flow.

```
wifi_config_set_roam_rssithreshold(0, -60);
```

   d) Use the API to set block time (20s). If roaming succeeds or fails, the device cannot trigger roaming during block time (20s).

```
wifi_config_set_roam_block_time(0, 20);
```

   e) Use the API to set max lock count (1 time). If roaming scan can’t find the target AP, the device will try one more time.

```
wifi_config_set_roam_maxlock_count(0,1);
```

f) Use the API to set lock time (2s). If roaming scan can’t find the target AP, the device will try one more time after lock time (2s).

```
wifi_config_set_roam_lock_time(0, 2);
```

g) Use the API to set scan channel type (full channel). In this example, roaming scan type is full channel.

```
wifi_config_set_roam_scan_channel(0, 1);
```

h) Use the API to set RSSI delta (10dB). If roaming scan finds the target AP and the RSSI of target AP is larger than the original AP 10dB, the device will roam to the target AP.

```
wifi_config_set_roam_delta(0, 10);
```

11.4 Trigger Roaming with Beacon Miss Threshold

This section introduces how to use API to trigger roaming with beacon miss threshold. Refer to the following APIs to set roaming parameters, and these APIs can be set before or after connection.

1) In this example, enable roaming and set beacon miss threshold to 10, as shown below.
   a) Call `wifi_config_set_autoroam` () to enable roaming feature.

```
wifi_config_set_autoroam(0, 1);
```

   b) Call `wifi_config_set_roam_by_bcnmiss()` to trigger roaming by beacon miss threshold.

```
wifi_config_set_roam_by_bcnmiss(0, 1);
```

   c) Use the API to set beacon miss threshold (10). If beacon miss count is lower than beacon miss threshold (10), the device will trigger roaming flow.

```
wifi_config_set_roam_by_bcnmissthreshold (0, 10);
```

   d) Use the API to set block time (0s). If roaming succeeds or fails, the device cannot trigger roaming during block time (0s).

```
wifi_config_set_roam_block_time(0, 0);
```

   e) Use the API to set max lock count (1 time). If roaming scan can't find the target AP, the device will try one more time.

```
wifi_config_set_roam_maxlock_count(0,1);
```

   f) Use the API to set lock time (2s). If roaming scan can't find the target AP, the device will try one more time after lock time (2s).

```
wifi_config_set_roam_lock_time(0, 2);
```

   g) Use the API to set scan channel type (full channel). In this example, roaming scan type is full channel.

```
wifi_config_set_roam_scan_channel(0, 1);
```

   h) Use the API to set beacon timeout period (5s). If roaming flow is not finished after beacon timeout period (5s), the device will trigger disconnect flow.

```
wifi_config_set_bto_time(0, 5);
```

Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this “Document”) is subject to your (including the corporation or other legal entity you represent, collectively “You”) acceptance of the terms and conditions set forth below (“T&C”). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don’t agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively “MediaTek”) or its licensors and is provided solely for Your internal use with MediaTek’s chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek’s suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. MediaTek does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MediaTek makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does MediaTek assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek’s product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.