Airoha IoT SDK for RTOS MT5932 Wi-Fi Developer's Guide

Version: 1.0

Release date: 21 May 2018

Document Revision History

Revision	Date	Description
1.0	21 May 2018	Initial release

Table of contents

Lists of tables and figures

1. Introduction

1.1. Platform

Airoha IoT SDK for RTOS development platform offers comprehensive technology including hardware development kit (HDK) and software development kit (SDK) to design and develop Wearables and IoT applications.

MT5932 HDK is a Wi-Fi only chipset that connects to the host device with SDIO interface. The host device used as an example host HDK is MT2523.

MT5932 provides the following features:

- Station mode Wi-Fi Connection (under development)
- Wi-Fi scan

This document guides you through:

- The hardware requirements and system architecture.
- Getting started with the environment setup, prerequisites and HDK configuration.
- Building and running an application that connects MT5932 to the host using Wi-Fi connectivity.

1.2. Architecture

Figure 1 provides detailed communication between host (MT2523 is used in current setup) and MT5932 communication chipset. WFC/WFCM is a Wi-Fi communication protocol between MT5932 and the host. Currently this platform supports Wi-Fi scan, but network and Wi-Fi connectivity features are not supported.

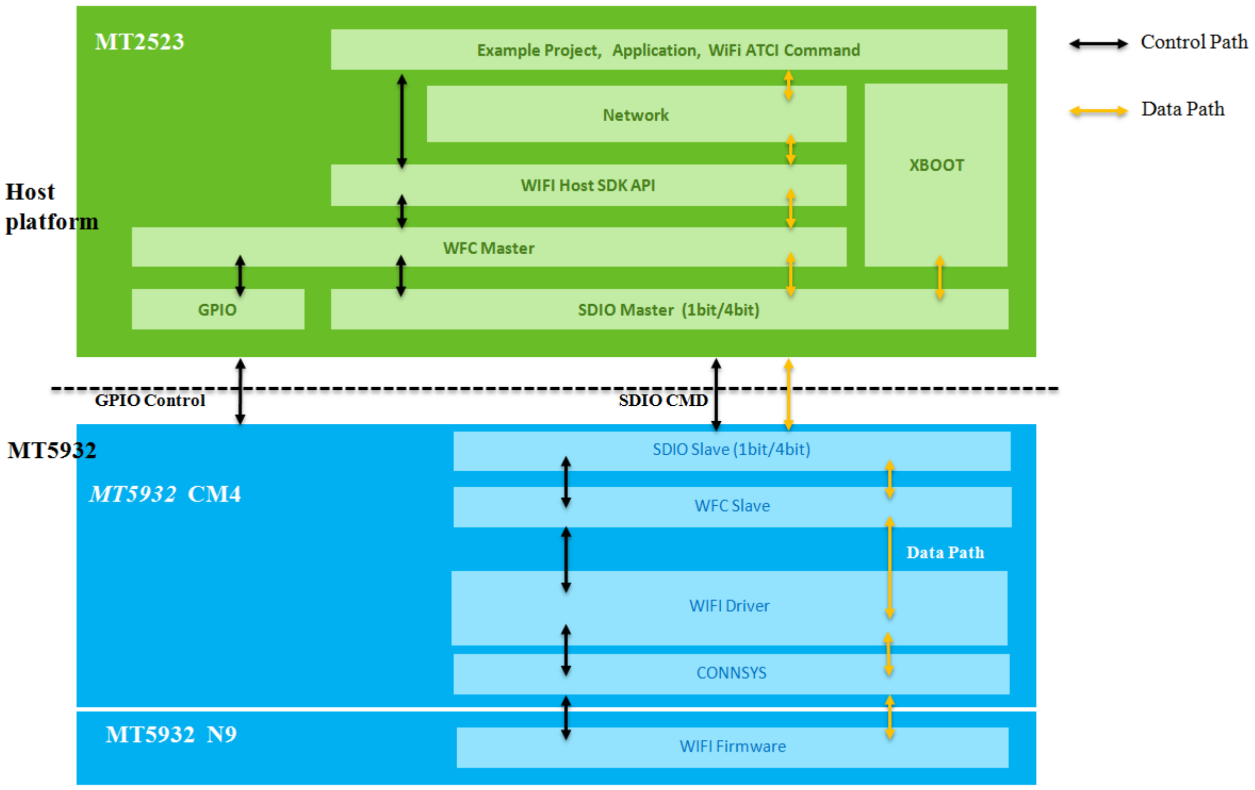

Figure 1. Wi-Fi architecture

Major module introduction:
Wi-Fi Host SDK API: A serial Wi-Fi APIs to control MT5932 for user;
WFC Master: Flow control in host for command and data path;
XBOOT: Used to download MT5932 image to slave;
SDIO Master: SDIO driver interface (necessary for running WFC Master and XBOOT; It supports 1-bit and 4-bit modes);
WFC Slave: Flow control in slave for command and data path;
SDIO Slave: SDIO driver interface (it works with SDIO Master and also supports 1-bit and 4-bit modes);

2. Getting Started

2.1. Environment

A GCC compiler is necessary for building the project.

2.2. Hardware configuration

2.2.1. SDIO interface configuration

An example of the pin configuration between MT5932 and MT2523 using the SDIO interface is shown in Table 1**.**

Table 1*. *SDIO interface configuration

MT2523		MT5932		Function	Description
GPIO	Direction	GPIO	Direction		
GPIO30	Output	GPIO11	Input	SDIO interface	SDIO_MC0_CK
GPIO31	I/O	GPIO12	I/O	SDIO interface	SDIO_MC0_CM0
GPIO32	I/O	GPIO13	I/O	SDIO interface	SDIO_MC0_DA0
GPIO33	I/O	GPIO14	I/O	SDIO interface	SDIO_MC0_DA1
GPIO34	I/O	GPIO15	I/O	SDIO interface	SDIO_MC0_DA2
GPIO35	I/O	GPIO16	I/O	SDIO interface	SDIO_MC0_DA3
GPIO24	Output	GPIO3	Input		Host wake up/notify slave
GPIO6	Input	GPIO4	Output		Slave wake up/notify host
GPIO48	Output	-		Chip_en	enable/disable slave

2.3. Your first project

Airoha IoT SDK for RTOS version 4.7.0 provides a reference application based on MT2523 and MT5932 under "`<sdk_root>/project/mt2523_hdk/app/wifi5932_ref_design`".

1) Enable the Wi-Fi feature

Navigate to the project folder and find the following macros to enable the Wi-Fi feature through the SDIO interface in feature.mk file, as shown below:

```
MTK_WIFI_CHIP_USE_MT5932 = y
MTK_WIFI_STUB_CONF_ENABLE = y
MTK_WIFI_STUB_CONF_SDIO_MSDC_ENABLE = y
```

2) Build and run the project

Navigate to the SDK root directory. Enter the following command to build the project on the MT2523 HDK:

```
./build.sh mt2523_hdk wifi5932_ref_design
```

3) Use the host supported Flash Tool to download to binary images to the MT2523 HDK's flash. The files are in "`<sdk_root>/out/mt2523_hdk/wifi5932_ref_design`".

```
flash.bin
iot_sdk_demo.bin
```

   a) `flash.bin` is the image of the wifi5932_ref_design project.

   b) `iot_sdk_demo.bin` is the image of the MT5932 project. This image is copied from

      `<sdk_root>/prebuilt/driver/chip/mt5932/wifi_image/mt5932_image_sdio.bin.`

4) Reboot the MT2523 HDK. MT2523 uses the XBOOT mechanism to send iot_sdk_demo`.bin` to the MT5932 system RAM. MT5932 starts running in the system RAM code.

2.3.1. API description

This section describes the APIs supported by the host for configuring MT5932. For more details, please refer to the WiFi_MT5932 chapter in host device's API reference manual in `<sdk_root>/doc`.

Syntax	`int32_t wifi_host_connection_scan_init(wifi_scan_list_item_t *ap_list, uint32_t max_count);`			
Parameters	**Mode**	**Name**	**Type**	**Description**
	Input	`ap_list`	`wifi_scan_list_item_t`	A buffer to store scanned data.
	Input	`max_count`	`uint8_t`	The count of APs to scan. The maximum is 100.
Return Value	() return values: • = 0, the operation completed successfully; • < 0, the operation failed.			

Syntax	`int32_t wifi_host_connection_scan_deinit(void);`			
Parameters	**Mode**	**Name**	**Type**	**Description**
	N/A	N/A	N/A	N/A
Return Value	() return values: • = 0, the operation completed successfully; • < 0, the operation failed.			

Syntax	`int32_t wifi_host_connection_start_scan(uint8_t *ssid, uint8_t ssid_length, uint8_t *bssid, uint8_t scan_mode, uint8_t scan_option);`			
Parameters	**Mode**	**Name**	**Type**	**Description**
	Input	`ssid`	`uint8_t`	Specifies the SSID to include in the probe request packet for scanning hidden APs. It can be NULL.
	Input	`ssid_length`	`uint8_t`	The length of the SSID. The maximum length is 32.
	Input	`bssid`	`uint8_t`	If the BSSID is specified, a unicast probe request is sent. If the BSSID is NULL, the broadcast probe request is sent.
	Input	`scan_mode`	`uint8_t`	Scan mode can be either full scan or partial scan.

	Input	`scan_option`	`uint8_t`	Selects the scan options based on one of the following: active scan; passive scan; or force active scan.
Return Value	() return values: • = 0, the operation completed successfully; • < 0, the operation failed.			

Syntax	`int32_t wifi_host_connection_stop_scan(void);`			
Parameters	**Mode**	**Name**	**Type**	**Description**
	N/A	N/A	N/A	N/A
Return Value	() return values: • = 0, the operation completed successfully; • < 0, the operation failed.			

Syntax	`int32_t wifi_host_connection_register_event_handler(wifi_event_t event, wifi_event_handler_t handler);`			
Parameters	**Mode**	**Name**	**Type**	**Description**
	Input	`event`	`wifi_event_t`	Event identification. Please refer to `wifi_event_t` in host API reference manual for more information.
	Input	`handler`	`wifi_event_handler_t`	Event handler. Please refer to `wifi_event_handler_t` in host API reference manual for more information.
Return Value	() return values: • = 0, the operation completed successfully; • < 0, the operation failed.			

Syntax	`int32_t wifi_host_connection_unregister_event_handler(wifi_event_t event, wifi_event_handler_t handler);`			
Parameters	**Mode**	**Name**	**Type**	**Description**
	Input	`event`	`wifi_event_t`	Event identification. Please refer to `wifi_event_t` in host API reference manual for more information.
	Input	`handler`	`wifi_event_handler_t`	Event handler. Please refer to `wifi_event_handler_t` in host API reference manual for more information.
Return Value	() return values: • = 0, the operation completed successfully; • < 0, the operation failed.			

Syntax	`int32_t wifi_host_config_set_wifi_start(void);`

Parameters	**Mode**	**Name**	**Type**	**Description**
	N/A	N/A	N/A	Starts MT5932.
Return Value	() return values: • = 0, the operation completed successfully; • < 0, the operation failed.			

Syntax	`int32_t wifi_host_config_set_wifi_stop(void);`			
Parameters	**Mode**	**Name**	**Type**	**Description**
	N/A	N/A	N/A	Stops MT5932. Nearly the same as powering off MT5932.
Return Value	() return values: • = 0, the operation completed successfully; • < 0, the operation failed.			

2.4. Host project configuration

MT2523 is the host platform used with the MT5932 HDK. To configure a MT2523 based application with Wi-Fi capabilities, add the following code flow to the project:

1) Initialize the Wi-Fi

Call wifi_host_config_set_wifi_start () and lwip_host_main_init () in "`main()`" function of your host application to initialize WFCM in the host and activate the pins for MT5932.

```
wifi_host_config_set_wifi_start();
lwip_host_main_init();
```

2) Scan and get the scan result

Call `wifi_host_connection_start_scan()` to trigger the scan and get the scan result, as shown below.

a) We strongly recommend that you define an event handler function to parse the scan result (payload).

```
static int scan_event_handler_sample(wifi_event_t event_id, unsigned
char *payload, unsigned int len)
{
    /*
        Scan complete, the scan result will be stored in scan_ap_list[]
        Do the scan result analysis.
    */

    wifi_host_connection_scan_deinit();
}
```

b) Register the event handler function and call `wifi_host_connection_start_scan()` to start scanning.

```
#define SCAN_AP_LIST_MAX_COUNT  30
wifi_scan_list_item_t scan_ap_list[SCAN_AP_LIST_MAX_COUNT];

wifi_host_connection_register_event_handler(WIFI_EVENT_IOT_SCAN_COMPLETE
, (wifi_event_handler_t) scan_event_handler_sample);

wifi_host_connection_scan_init(scan_ap_list, SCAN_AP_LIST_MAX_COUNT);
```

```
wifi_host_connection_start_scan(NULL, 0, NULL, 0, 0);
```

Note, always call `wifi_host_connection_scan_deinit()` after calling `wifi_host_connection_scan_init()`.

2.5. Customize TX power settings

2.5.1. How to customize TX power

The host project provides a method for setting the TX power during the initial stage. The TX power message is stored in the g_tx_power_bin[TX_POWER_BIN_SIZE] parameter in <sdk_root>/middleware/MTK/wifi_host/common/src/wifi_host_init.c

```
uint8_t g_tx_power_bin[TX_POWER_BIN_SIZE] =
{0x76,0x82,0x76,0x86,0xC5,0xC5,0xC4,0xC3,0xC1,0xC0,0xC0,0xC2,0xC2,0xC0,0
xC0,0x82,0x82,0x00,0x00,0x00,0x00,0x83,0x83,0x00};
```

The user can customize this parameter value before using it.

2.5.2. Introduction of TX power value

Table 2 shows a description of the power values in g_tx_power_bin.

Table 2. TX power value introduction

Offset	Value	Description	Write owner	Value
0x00	76	**must be kept as in file**	Customer	Default by Airoha
0x01	82		Customer	Default by Airoha
0x02	76	**must be kept as in file**	Customer	Default by Airoha
0x03	86		Customer	Default by Airoha
0x04	C5	2.4GHz TX power for CCK 1M/2M (delta, dB)	Customer	Option
0x05	C5	2.4GHz TX power for CCK 5.5M/11M (delta, dB)	Customer	Option
0x06	C4	2.4GHz TX power for OFDM 6M/9M (delta, dB)	Customer	Option
0x07	C3	2.4GHz TX power for OFDM 12M/18M (delta, dB)	Customer	Option
0x08	C1	2.4GHz TX power for OFDM 24M/36M (delta, dB)	Customer	Option
0x09	C0	2.4GHz TX power for OFDM 48M (delta, dB)	Customer	Option
0x0A	C0	2.4GHz TX power for OFDM 54M (delta, dB)	Customer	Option
0x0B	C2	2.4G TX power for HT20 MCS=0 (delta, dB)	Customer	Option
0x0C	C2	2.4G TX power for HT20 MCS=1,2 (delta, dB)	Customer	Option
0x0D	C0	2.4G TX power for HT20 MCS=3,4 (delta, dB)	Customer	Option
0x0E	C0	2.4G TX power for HT20 MCS=5 (delta, dB)	Customer	Option
0x0F	82	2.4G TX power for HT20 MCS=6 (delta, dB)	Customer	Option
0x10	82	2.4G TX power for HT20 MCS=7 (delta, dB)	Customer	Option

0x11	00	2.4G TX power for HT40 MCS=0 (delta, dB)	Customer	Option
0x12	00	2.4G TX power for HT40 MCS=1,2 (delta, dB)	Customer	Option
0x13	00	2.4G TX power for HT40 MCS=3,4 (delta, dB)	Customer	Option
0x14	00	2.4G TX power for HT40 MCS=5 (delta, dB)	Customer	Option
0x15	83	2.4G TX power for HT40 MCS=6 (delta, dB)	Customer	Option
0x16	83	2.4G TX power for HT40 MCS=7 (delta, dB)	Customer	Option
0x17	00	2.4G TX power for HT40 MCS=32 (delta, dB)	Customer	Option

2.5.3. 2.4G TX rate power configuration (0x04h~0x17h)

Default value=0x00, 6-bit signed 2's complement value. (1 step=0.5dB) 0x04~0x17 are used as the TX rate power configuration in the customer production line. Customers can set a different TX rate power value according to their RF power requirement.

Offset	Field	Description
04h ~17h	5:0	Tx per-rate power setting
	7:6	Bit[7]: enable Bit[6]: 0: decrease, 1 :increase

The 1 step=0.5dB.

Example:
In 2.4G, all TX rate power configurations 0x04 ~ 0x17 are referred to the value of OFDM 54M TX power level. The default value of OFDM 54M TX power is 16dBm.

0x04: 0xC5 ➔ 2.4G TX power of CCK 1M/2M is 2.5dB higher than OFDM-54M
0x06: 0xC4 ➔ 2.4G TX power of OFDM 6/9M is 2dB higher than OFDM-54M
0x07: 0xC3 ➔ 2.4G TX power of OFDM 12/18M is 1.5dB higher than OFDM-54M
0x0B: 0xC2 ➔ 2.4G TX power of HT20 MCS=0 is 1dB higher than OFDM-54M