MediaTek LinkIt™ Connect 7681 Developer's Guide

Version: 1.2
Release date: 12th May 2016

Specifications are subject to change without notice.

Document Revision History

Revision	Date	Description
1.0	3rd January 2015	Initial release
1.0	31st January 2015	Update to code in 2.4.1, "Code your firmware" Correction to path names in 3.1, "make" Typo in FOTA FW description Appendix A
1.1	23rd January 2016	Updated instructions in 2.2.1, "Installing Cygwin" Updated instructions in 2.4.3.1 "Uploading the firmware from Windows" Correction to the code in 5.3.1, "Creating a TCP client that connects to a remote TCP server" and 5.3.2, "Create a TCP server and allow others to connect".
1.2	12th May 2016	Updated commands with long dash with "-" throughout.

Table of Contents

Lists of tables and figures

1.   Introduction

There is an increasing trend towards connecting every imaginable electrical or electronic device found in the home. For many of these applications developers simply want to add the ability to remotely control a device — turn on a table lamp, adjust the temperature setting of an air-conditioner or unlock a door.

The MediaTek MT7681 chipset is designed to fulfil this need.

1.1.   MediaTek MT7681

The MediaTek MT7681 is a compact Wi-Fi System-on-Chip (SoC) with embedded TCP/IP stack for IoT devices. By adding the MT7681 to an IoT device it can connect to other smart devices or to cloud applications and services.

Connectivity on the MT7681 is achieved using Wi-Fi, in either Wi-Fi station or access point (AP) mode.

In Wi-Fi station mode, MT7681 connects to a wireless AP and can then communicate with web services or cloud servers. A typical use of this option would be to enable a user to control the heating in their home from a home automation website.

To simplify the connection of an MT7681 chip to a wireless AP in Wi-Fi station mode, the MediaTek Smart Connection APIs are provided. These APIs enable a smart device app to remotely provision a MT7681 chip with AP details (SSID, authentication mode and password).

In AP mode, an MT7681 chip acts as an AP, enabling other wireless devices to connect to it directly. Using this mode, for example, the developer of a smart light bulb could offer users a smartphone application that enables the bulbs to be controlled from within the home.

To control the device an MT7681 is incorporated into, the chip provides five GPIO pins and one UART port. In addition PWM is supported in software, for applications such as LED dimming.

A block diagram illustrating the components of the chipset is shown in Figure 1.
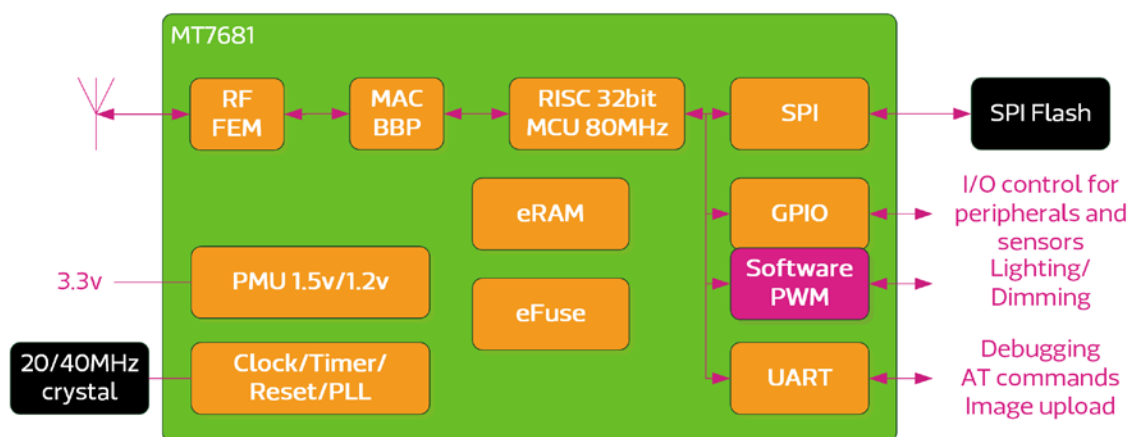
Figure 1 Block diagram of the MT7681

Key features of the MT7681 chipset:

- Wi-Fi station and access point (AP) modes

- 802.11b/g/n (in station mode) and 802.11b/g (in AP mode)

- TCP/IP stack

- 5 GPIO pins and 1 UART port

- software PWM emulation for LED dimming

- firmware upgrade over UART, APIs for FOTA implementation

1.2.   MediaTek LinkIt™ Connect 7681 Development Platform

To enable developers and makers to take advantage of the features of the MT7681 chipset, MediaTek Labs offers the MediaTek LinkIt Connect 7681 development platform, consisting of an SDK, HDK and related documentation.

For software development the SDK is provided for Microsoft Windows and Ubuntu Linux. Based on the Andes Development Kit, the MediaTek LinkIt Connect 7681 SDK enables developers to compile and upload firmware to control an IoT device in response to instructions received wirelessly.

For IoT device prototyping, the LinkIt Connect 7681 HDK delivers the LinkIt Connect 7681 development board. The development board consists of a LinkIt Connect 7681 module, micro-USB port and pins for each of the I/O interfaces of the MT7681 chipset. This enables you to quickly connect external hardware and peripherals to create device prototypes. The LinkIt Connect 7681 module, which measures just 15 x 18mm, is designed to easily mount on a PCB as part of production versions of the IoT device.

1.3.   MediaTek LinkIt Connect 7681 SDK

To enable the development of software for MediaTek LinkIt Connect 7681 development platform an SDK is provided for Microsoft Windows and Ubuntu Linux. The SDKs can be downloaded from the MediaTek website.

The MediaTek LinkIt Connect 7681 SDK, based on the Andes Development Kit, enables you to create firmware to control an IoT device in response to instructions received wirelessly over a WiFi connection.

The key features of the MediaTek LinkIt Connect 7681 SDK are:

- libraries for all the MT7681 APIs, including MediaTek Smart Connection and FOTA firmware updates

- C-like language

- command line compiler, based on Andes Development Kit:

  o   Linux tool chain package: `Andes_Linux_BSP320_Toolchains.tgz`

  o   Windows tool chain package: `Andes_Windows_Toolchains.rar`

- firmware upload tool

- source and binary code for MT7681 firmware

- MediaTek Smart Connection app (IoT Manager) with examples for Android and iOS, including source code

- example source code : IoT Server, AT Command Parser, Data Command Parser and X-Modem

- versions available for Microsoft Windows (requires Cygwin) and Ubuntu Linux

1.3.1.   Software Architecture

The software architecture of the MT7681 provides for access to its hardware features through four APIs — Wi-Fi API, Security API, Interface API and Control API — as shown in Figure 2.
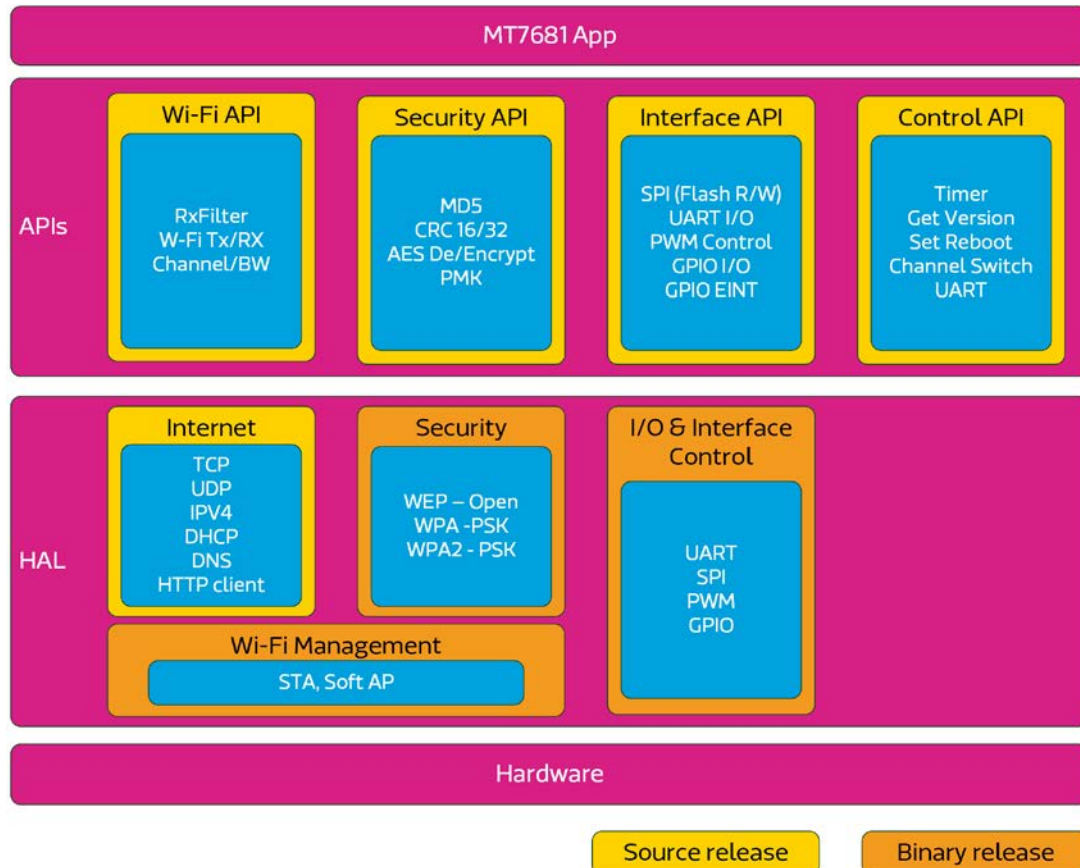
Figure 2 The software architecture of the MT7681

The SDK includes this software stack as:

- binary releases for the Wi-Fi protocol stack (including station mode, AP mode and authentication protocols) and the I/O interface control.

- source code release for all the other elements in the architecture, including TCP/IP stack, UART data parser and alike.

1.3.2. API Function Overview

The functions provided in each API are outlined in Table 1, Table 2 and Table 3. For full API and function descriptions please refer to the MediaTek LinkIt Connect 7681 API Reference.

Interface API	
SPI read/write	Read and write data from and to the built-in Flash memory through a SPI interface.
UART	Read and write data from and to the UART interface.
GPIO	Control the mode and state of five GPIO pins.
PWM	Provide simple PWM functionality on all five GPIO pins.

Table 1 The interface APIs of MT7681

Internet API	
TCP/IP	TCP/IP stack functions.
UDP	UDP protocol functions.
FOTA	Provide Firmware Over The Air (FOTA) features, enabling wireless firmware upgrades.

Table 2 The Internet APIs of MT7681

Wi-Fi Functions	
WEP/WPA	Wi-Fi security authentication functions.

Table 3 The Wi-Fi functions of MT7681

1.4. LinkIt Connect 7681 HDK

The LinkIt Connect 7681 development board, co-designed with Seeed Studio, provides easy access to the five GPIO pins and one UART port — each with an LED — of the MT7681, to quickly connect peripheral controllers and electronic circuits. The MT7681 is provided on the LinkIt Connect 7681 module, which was co-designed with AcSiP. This module, which is only 15 x 18mm, will also be available separately for direct integration into your product's PCB.

Key features of the LinkIt Connect 7681 development board:

- access to the five GPIO pins and one UART port provided by MT7681, with attached LEDs

- access t0 the UART port from the USB port.

- Wi-Fi antenna integrated on LinkIt Connect 7681 module

- power over USB (Micro USB port)

- open source hardware board reference design, including schematic, layouts, and pin details.

The LinkIt Connect 7681 development board will be available from Seeed Bazaar.

The key specifications of the LinkIt Connect 7681 development board are shown in Table 4.

Category	Feature	Spec
PCB Size		50 x 31 mm
Power		USB 5V Input (micro USB)
GPIO Pins	Pin Count Voltage	5 3.3v
GPIO LEDs		5
UART	Set Count Voltage	1 3.3v
UART LEDs		2 (TX/RX)

Table 4 Key specifications of the LinkIt Connect 7681 development board

The key specifications of the LinkIt Connect 7681 module are shown in Table 5.

Category	Feature	Spec
Microcontroller	Chipset Core Clock Speed	MT7681 ANDES N9 80MHz
Module Size		15 x 18 mm
Memory	Flash RAM	1MB 64KB
Power	Operate voltage	3.3v
GPIO Pins	Pin Count Voltage	5 3.3v
PWM Output Pins	Pin Count Voltage PWM levels	5 3.3v 0~20
UART	Set Count Voltage	1 3.3v
Wi-Fi Spec	Station mode Access point (AP) mode	802.11 b/g/n 802.11 b/g

Table 5 Key specifications of the LinkIt Connect 7681 module

1.5.   Join Our Ecosystem

Wearables and Internet of Things are the next wave in the consumer gadget revolution. MediaTek is a key player in this field, combining the best of two worlds —the existing MediaTek ecosystem of phone manufacturers, electronic device manufacturers, and telecom operators combined with the open, vibrant maker community world.

No matter whether you're a maker, device manufacturer, student, DIY hobbyist, or programmer, you can use this powerful yet simple platform to create something innovative. You can join the MediaTek LinkIt ecosystem by registering on labs.mediatek.com, we look forward to you joining our ecosystem and creating something great together.

2.   Getting started

This section provides a guide to getting started with the MediaTek LinkIt Connect 7681 development platform and covers:

- the supported environments for development.
- installing and configuring the MediaTek LinkIt Connect 7681 SDK.
- creating your first project.

2.1.   Environment

Currently MediaTek LinkIt Connect 7681 SDK supports Microsoft Windows and Ubuntu Linux platforms.

2.1.1.   Microsoft Windows

To run MediaTek LinkIt Connect 7681 SDK for Windows the Cygwin environment is needed. Cygwin is a large collection of GNU and Open Source tools that provide functionality similar to a Linux distribution on Microsoft Windows. Cygwin is available for all recent, commercially released x86 32-bit and 64-bit versions of Windows, starting with Windows XP SP3. Using the latest version of Cygwin is recommended.

In addition, a tool to unpack RAR archives is required.

2.1.2.   Ubuntu Linux

MediaTek LinkIt Connect 7681 SDK for Linux requires Ubuntu Linux. Any version can be used, however the latest LTS version is recommended.

2.2.   Installing MediaTek LinkIt Connect 7681 SDK for Windows

2.2.1.   Installing Cygwin

Download the appropriate Cygwin installation package to match your Windows environment from the Cygwin website: namely `setup-x86.exe` for 32-bit machines or `setup-x86-64.exe` for 64-bit machines.

Open the downloaded installation package. In the welcome screen, see Figure 3, click **Next**.
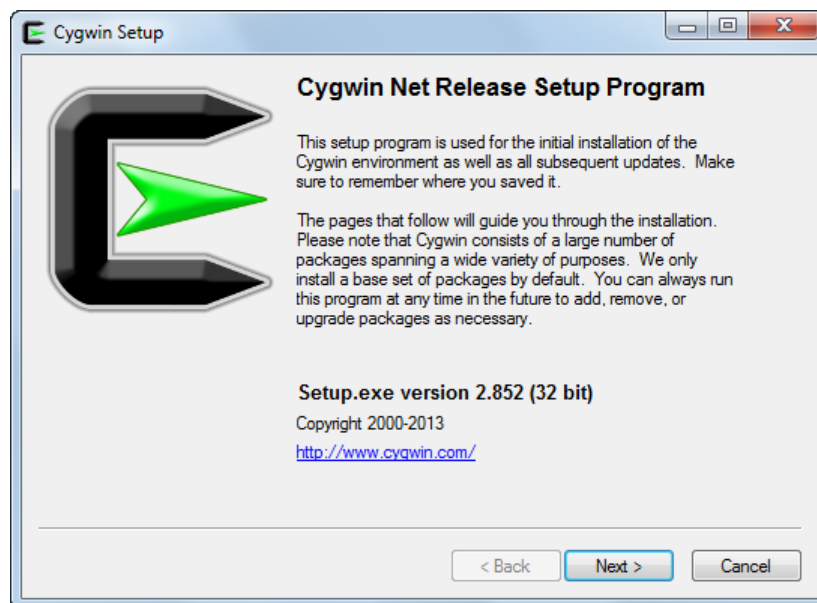
Figure 3 Cygwin Net Release Setup Program welcome screen

In the **Choose A Download Source** screen, see Figure 4, choose **Install from Internet** and click **Next**,
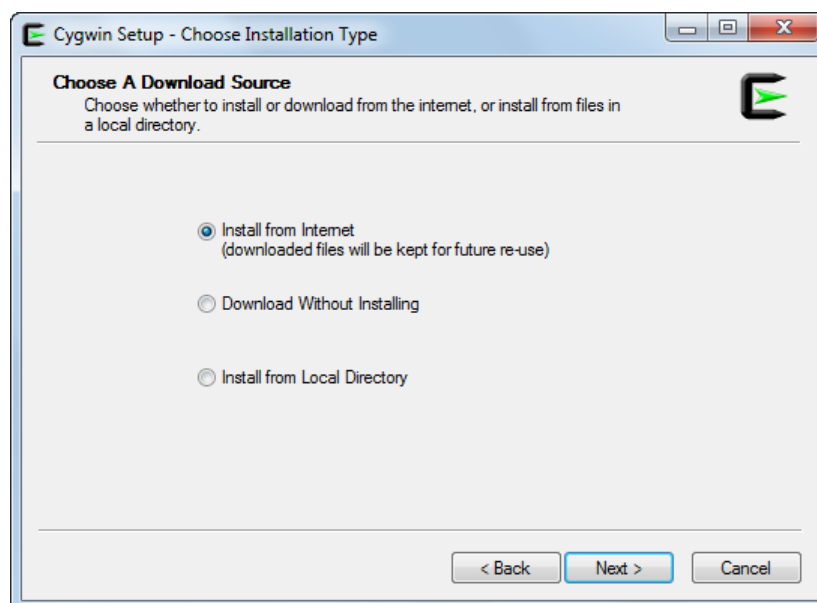
Figure 4 Choosing the installation type in the Cygwin setup dialog

In the **Select Root Install Directory** screen, see Figure 5, click **Next** to accept the default setting.
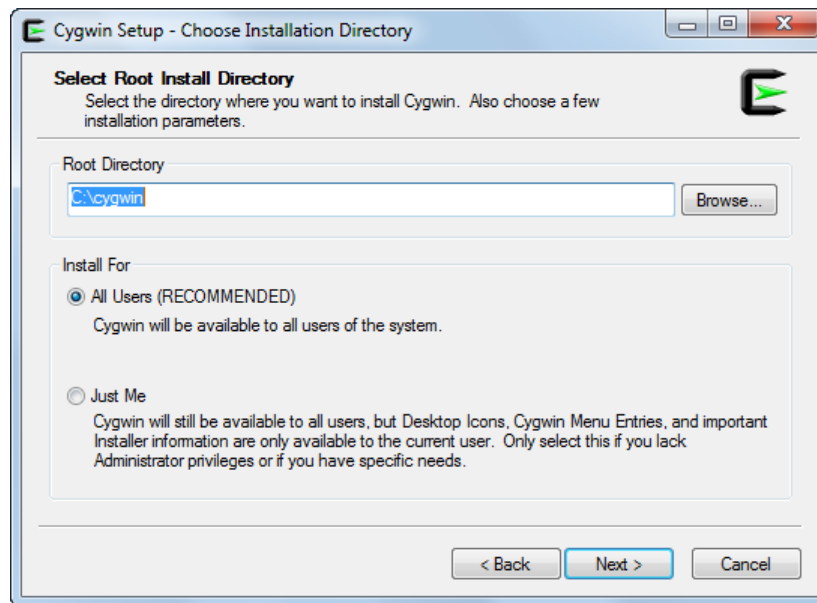
Figure 5 Accept the default installation settings

In the **Select Local Package Directory** screen, see Figure 6, click **Next** to accept the default location.
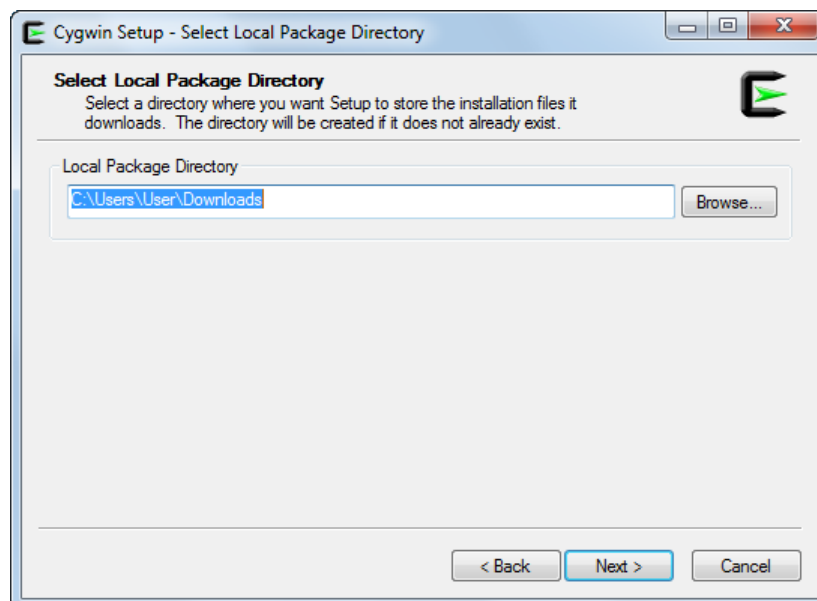
Figure 6 Define where the downloads packages will be stored

In the **Select Your Internet Connection** screen, see Figure 7, define your proxy if you use one otherwise use **Direct Connection** and click **Next**.
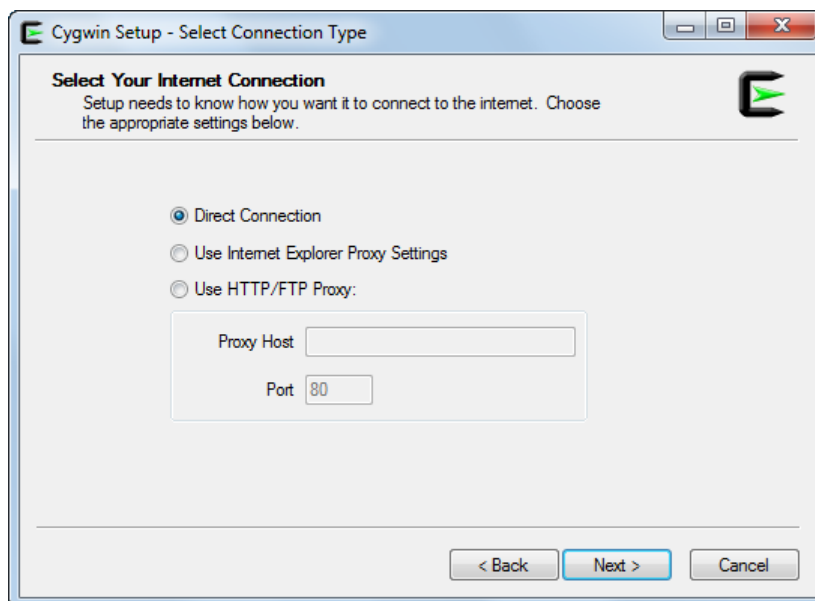
Figure 7 Use the direct connection unless you access the Internet using a proxy

In the **Choose A Download Site** screen, see Figure 8, select a mirror site: for the best download speed select a site that you think is nearby. Click **Next**.
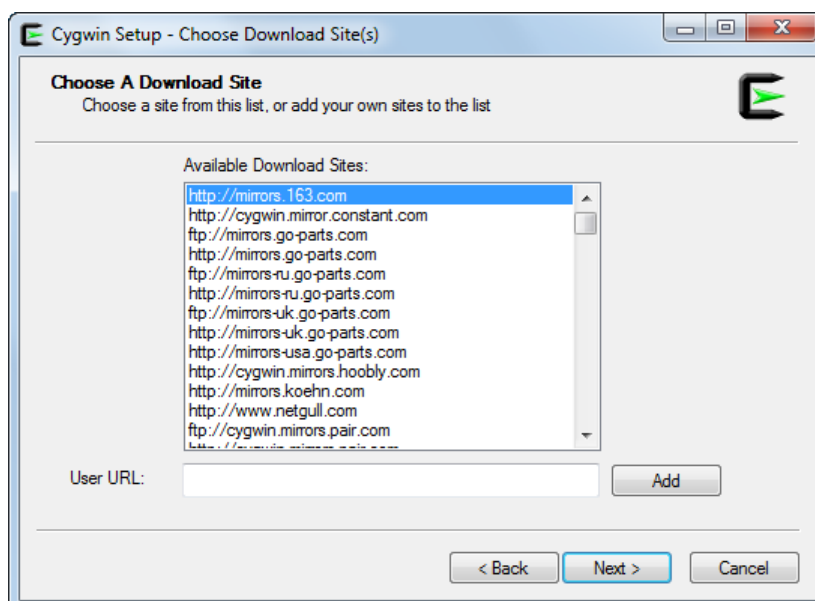
Figure 8 Select a nearby mirror site from which to download

The installer now downloads details and displays progress in the **Progress** screen, see Figure 9.
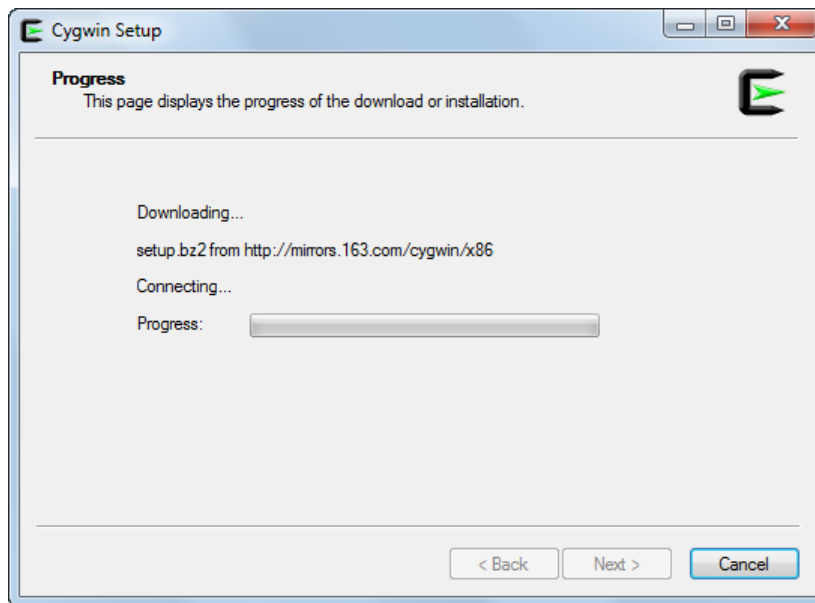
Figure 9 The installer downloads information about Cygwin

In the **Select Package** screen, select the **make** and **gcc-core** packages from the **Devel** category leaving other items at their default settings.
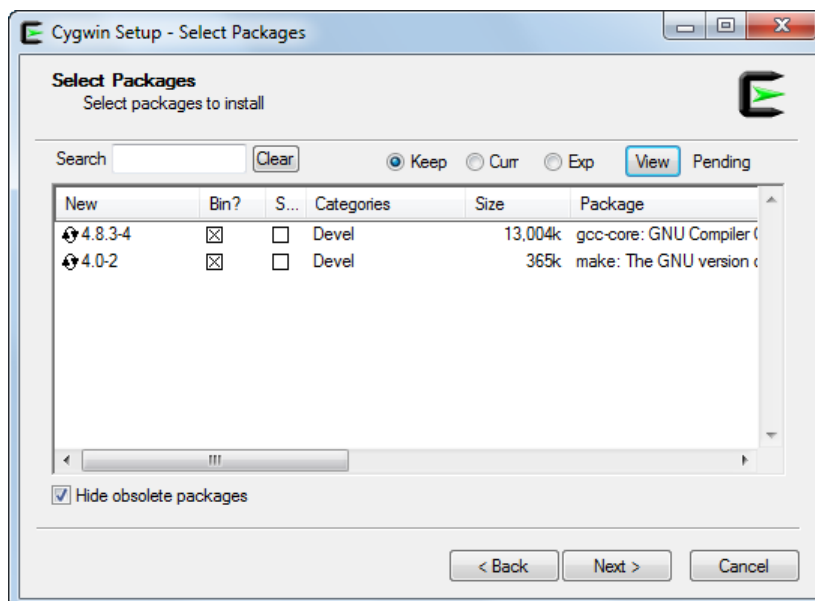
Figure 10 Select the make and gcc-core packages

The easiest way to select **make** and **gcc-code** is by expanding **Devel** and scrolling down to find **gcc-core** first and then **make** (the packages are listed alphabetically) as shown in Figure 11. Don't use the search filter.
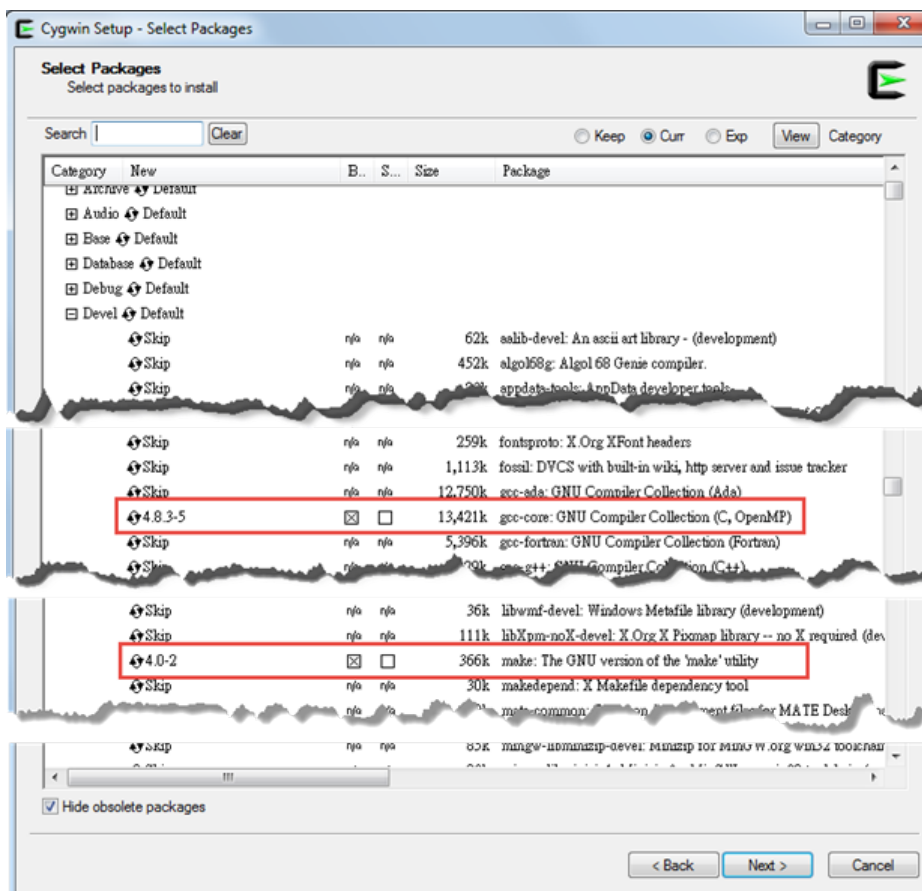
Figure 11 Selecting make and gcc-core packages

Click **Next**. The **Resolving Dependencies** screen, see Figure 12, displays showing details of any dependent packages for your selection, click **Next**.
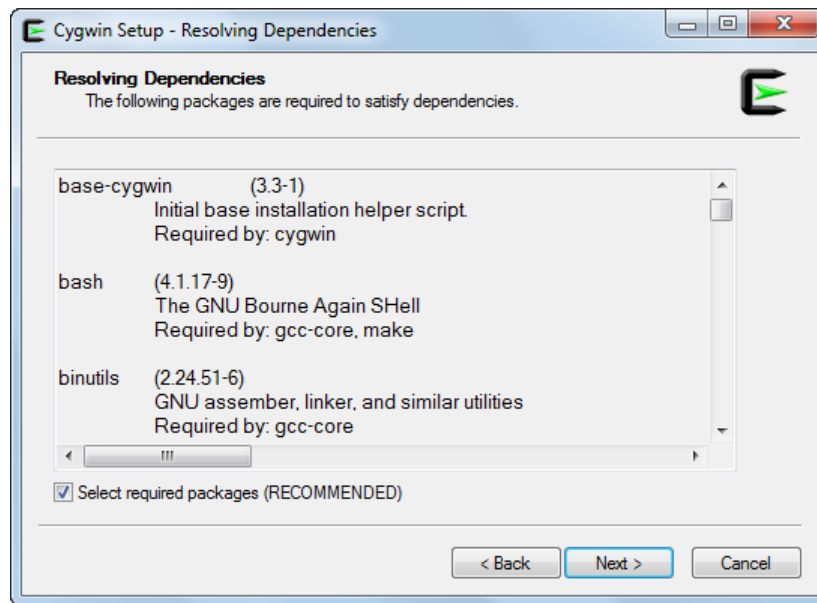
Figure 12 Details of any dependent packages are shown

The **Progress** screen, see Figure 13, shows the progress of the download and installation of your selected components. When installation has completed, click **Next**.
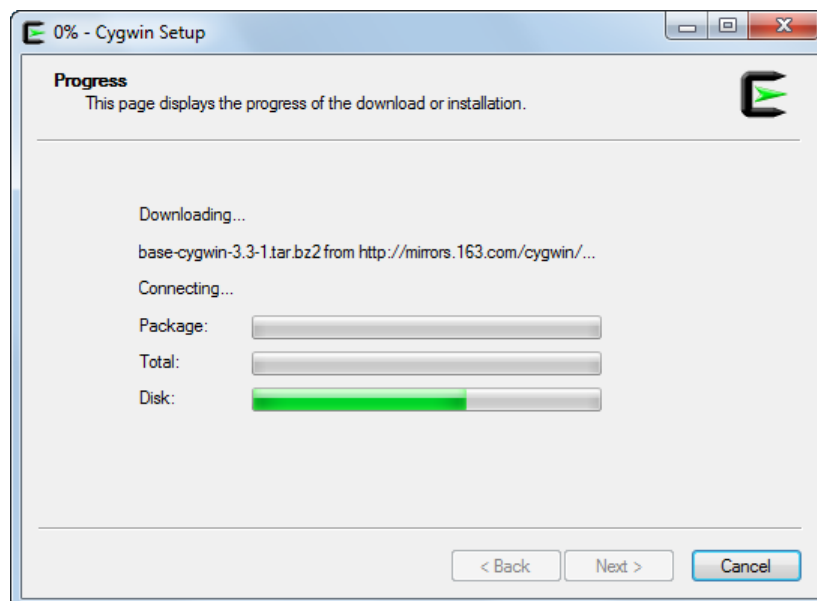
Figure 13 The progress of download and installation is displayed

The **Postinstall Script Errors** screen, see Figure 14, may display with details of any issues with the installation. Review any issues and determine what if any action you need to take, if any. Click **Next**.
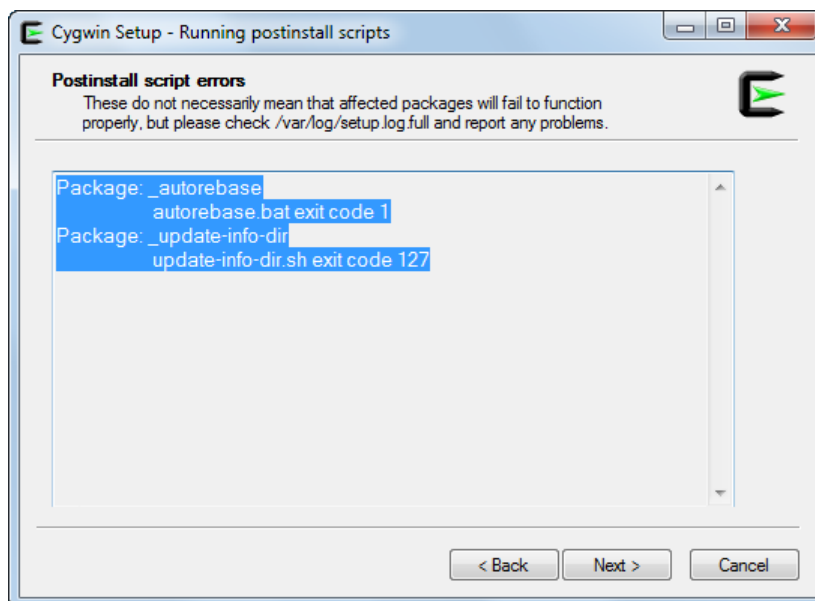
Figure 14 Any post install script errors are displayed

In the **Create Icons** screen, see Figure 15, click **Finish**.
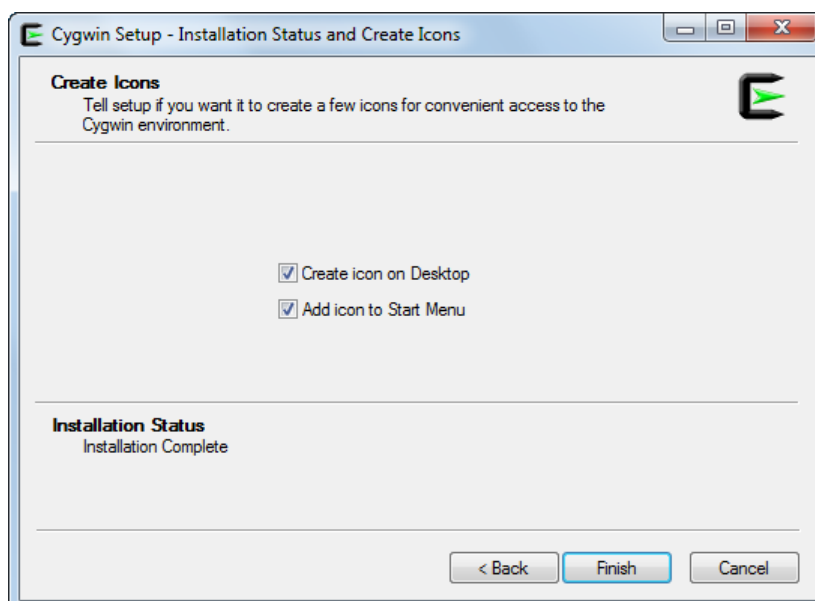
Figure 15 Click Finish to complete the installation

Cygwin is now installed. You can now continue to install the MediaTek LinkIt Connect 7681 SDK.

Adding the `cygwin\bin` and `cygwin\lib` folders to the system path is recommended. To do this, at the Windows command prompt (`cmd`) execute `path %path%;C:\cygwin\bin;C:\cygwin\lib`.

2.2.2. Installing the MediaTek LinkIt Connect 7681 SDK for Windows

Download the MediaTek LinkIt Connect 7681 SDK for Windows [from the MediaTek Labs website](#) and extract its content to a suitable drive.

Locate the tool chain file `Andes_Windows_Toolchains.rar` in the ToolChain folder, and extract its contents (folder `nds32le-elf-newlib-v2j`) to `C:\cygwin`, the root folder of the Cygwin installation.

If you've chosen to install Cygwin in a folder other than the default, you'll need to:

1) modify the tool chain's configuration file, which is located at `C:\<chosen folder>\nds32le-elf-newlib-v2j\cygwin-andes.bat`, so that the highlighted code in the following code example matches the folder you installed Cygwin into:

```
@echo off

@rem Batch file for Andes Technology Product to launch Cygwin

set TOP=C:\cygwin
set TOP=%TOP:\=/%
set TMP=C:\cygwin\cygwin_tmp
set PATH=C:\cygwin\bin;%CD%\bin
set HOME=%CD%\bin

IF EXIST "C:\cygwin\bin\bash.exe" set SHELL=/bin/bash

"C:\cygwin\bin\bash.exe" --login -i

:END
```

Save the configuration file.

2) modify the compiler's configuration file, which is located in the SDK under `Src\mak\MT7681\compiler.mk`. Modify the highlighted section of the `TOOLCHAINS` setting to map to the location of the `nds32le-elf-newlib-v2j` folder in the Cygwin installation folder, as shown below.

```
#****************************************************************************
# TOOLCHAINS DEFINE
#****************************************************************************

#For Windows TS
TOOLCHAINS  =   /cygdrive/C/cygwin/nds32le-elf-newlib-v2j/bin
```

Save the `compiler.mk` file.

You're ready to start developing code for the MediaTek LinkIt Connect 7681 development platform.

2.3. Installing MediaTek LinkIt Connect 7681 SDK for Linux

Download the MediaTek LinkIt Connect 7681 SDK for Linux [from the MediaTek Labs website](#) and extract its content to a suitable location.

Locate the tool chain file `Andes_Linux_BSP320_Toolchains.tgz` in the `ToolChain` folder, and extract its contents (folder `nds32le-elf-newlib-v2j`) to the folder `AndesTools`.

Now locate the `Src` folder and make sure the three shell script files (`flash_merge.sh`, `flash_merge_xip164.sh` and `header.sh`) are executable, by running `chmod` as follows:

```
linux:~/Src$chmod 755 *.sh
linux:~/Src$ls -l *.sh
-rwxr-xr-x flash_merge.sh
-rwxr-xr-x flash_merge_xip164.sh
-rwxr-xr-x header.sh
linux:~/Src$
```

Now locate `Src\mak\MT7681\compiler.mk` and modify the `TOOLCHAINS` setting to map to the location of the `nds32le-elf-newlib-v2j` folder in your Linux user folders, as shown below.

```
#**************************************************************************
*****
# TOOLCHAINS DEFINE
#**************************************************************************
*****

#For Linux TS
TOOLCHAINS  =   /home/xxxx/AndesTools/nds32le-elf-newlib-v2j/bin
```

Where 'xxxx' is your Linux account name.

Save the `compiler.mk` file and you're ready to start developing your code for the MediaTek LinkIt Connect 7681 development platform.

2.4. Your First Project

This section describes how to create, compile, and upload to the LinkIt Connect 7681 development board a simple application that blinks the GPIO2 LED every second.

2.4.1.   Code your firmware

Custom firmware code for a MT7681 is stored in the `*:\LinkIt_Connect_7681_SDK\Src\` folder. For this example the following code needs to be added to the `IoT_Cust_SubTask_1` method of `iot_custom.c`:

```
void iot_cust_subtask1(void)
{
    /* example: make GPIO 2 blinking every 1 second */
    static uint32 subPreTimer = 0;
    static uint32 subCurTimer = 0;
    uint8 input, polarity;

    /* get current counter value (unit:1ms) */
    subCurTimer =  iot_get_ms_time();

    /* if the time difference is 1000ms, change gpio output value*/
    if (subCurTimer - subPreTimer >= 1000)
    {
        subPreTimer = subCurTimer;
        iot_gpio_read(2, &input, &polarity); /*get gpio mode and value*/
        iot_gpio_output(2, (input+1)%2); /*change gpio to output mode with
the reversed value*/
    }
}
```

Simply locate the `iot_custom.c` file, open it in a suitable editor, add the code above, and save the file.

See section 4.1, "Creating software projects" for details on how to use the content of `*:\LinkIt_Connect_7681_SDK\Src\` to create your own firmware projects.

2.4.2.   Compiling the application

This section describes how to compile the `iot_customer.c` file into Wi-Fi station mode firmware for the LinkIt Connect 7681 development board.

2.4.2.1.   Source Code Compilation on Windows

In File Explorer open `C:\Cygwin` and double-click `cygwin-andes.bat` to start the Cygwin shell.

Enter the root directory of MT7681 source project, that is, the `Src` folder of the extracted SDK. In this case `*:\LinkIt_Connect_7681_SDK\Src\`, noting that Cygwin uses the Linux / to delineate the folder tree.

Enter `make clean; make` and press return to compile station mode firmware, as shown below.

```
~
$ cd /cygdrive/C/LinkIt_Connect_7681_SDK/Src

/cygdrive/C/LinkIt_Connect_7681_SDK/Src
$ make clean;make
...
...
merge flash bin file:MT7681_all.bin ok,size is 465212 bytes
out/sta/MT7681_sta.bin 4 out/sta/MT7681_sta_header.bin

$
```
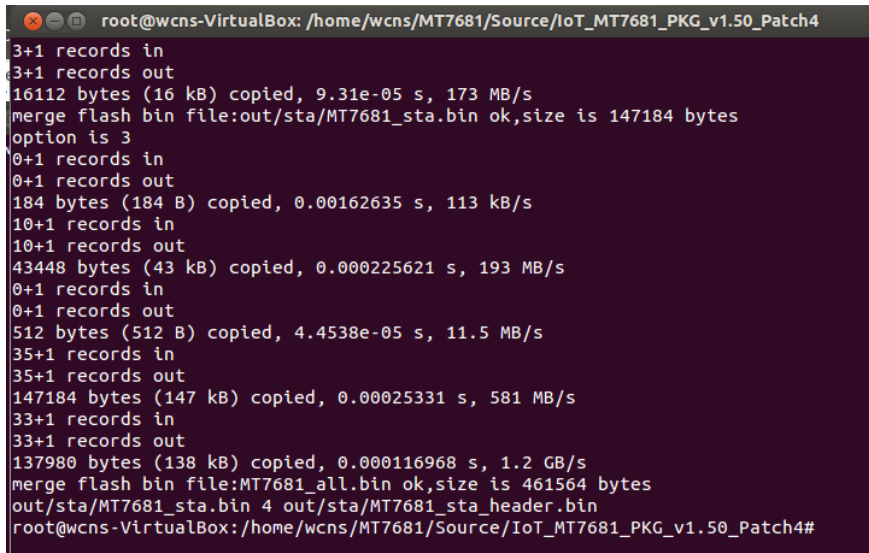
For more details on the commands available for `make`, see section 3.1, "make".

2.4.2.2. Source Code compilation on Linux

To compile the source code on Ubuntu Linux, open Terminal and switch to the `/Project/Src` folder in which `iot_custom.c` is stored. Enter `make clean; make` and press enter to compile station mode firmware as shown in Figure 16.

Figure 16 Compiling the MT7681 firmware on Linux

For more details on the commands available for `make` see section 3.1, "make".

2.4.3. Uploading the compiled binary to LinkIt Connect 7681 development board

This section describes how to upload the compiled firmware to the LinkIt Connect 7681 development board.

2.4.3.1.   Uploading the firmware from Windows

To upload the firmware from a Windows PC:

1) connect the LinkIt Connect 7681 development board to your computer using a USB cable, as shown in Figure 17.
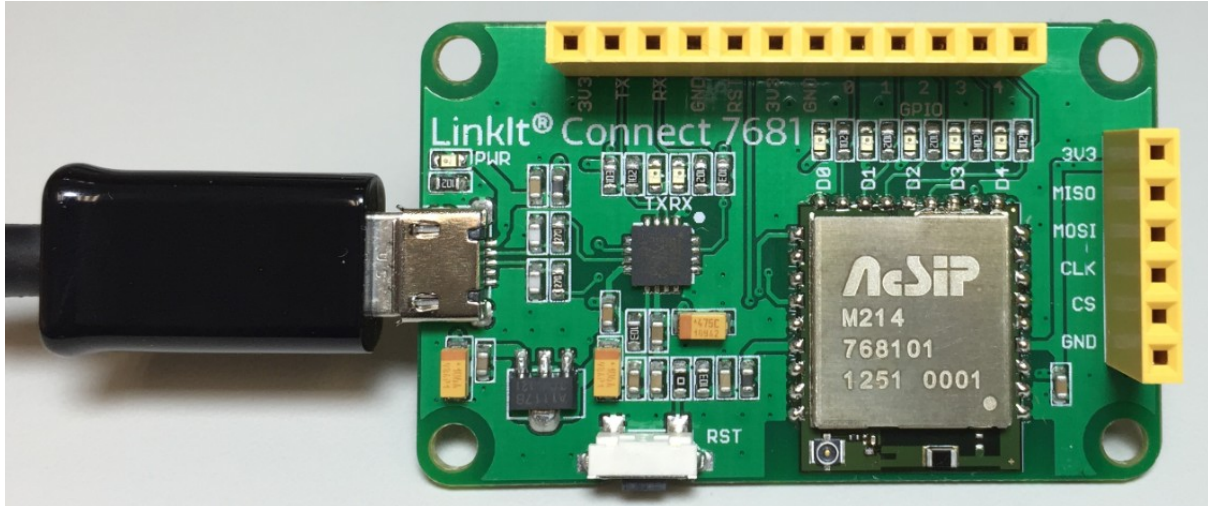
Figure 17 Connect the LinkIt Connect 7681 development board to a USB cable

2) identify the COM port on which the development board is connected, by opening **Device Manager** from **Control Panel**, see Figure 18.
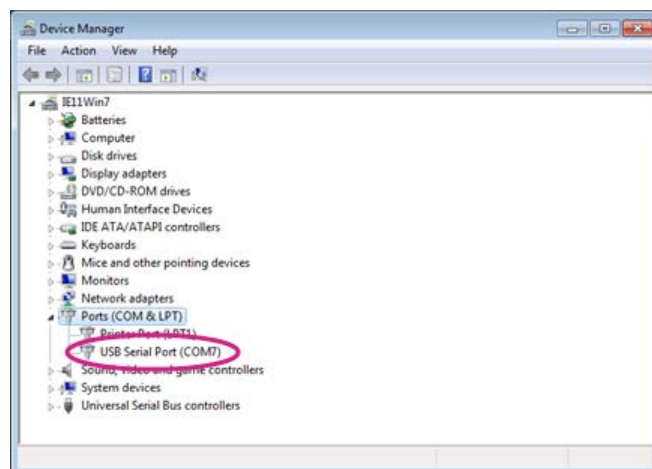
Figure 18 The connected COM port identified in Device Manager

3) open the Windows command prompt and:

   a) add the uploader folder to the path by executing
   `path %path%;*:\LinkIt_Connect_7681_SDK \Uploader`
   where * is the drive you installed the SDK (see section 2.2.2, "Installing the MediaTek LinkIt Connect 7681 SDK for Windows").

   b) switch to the `Scr` folder by executing `CD *:\LinkIt_Connect_7681_SDK\Src\` where * is again the drive where you installed the SDK.

4) upload the firmware using `mt7681_uploader.exe`, setting `-f` to specify the firmware generated in the previous step, and `-c` to identify the new COM port, as shown below:

```
> mt7681_uploader.exe -f out\sta\MT7681_sta_header.bin -c COM7
```

5) successful upload of the firmware will result in the following being displayed:

```
Recovery Mode entered, Starting to upload...
138084/138084 bytes (100%) ... Update done! Bye
```

If this method fails, you can run the uploader using the Python script, as follows.

- If you don't have Python installed, get it from the downloads section at python.org and install.

- Open `cmd` (using the Cygwin shell might not work) and execute `mt7681_uploader.py -f out\sta\MT7681_sta_header.bin -c COM7`

If the COM port is not shown in the PC's **Device Manager** and an error message states "`FT230x Basic UART driver not recognized`", install the driver software from Future Technology Devices International Limited. Download the executable version corresponding to your OS from the **Currently Supported VCP Drivers** list. After installation, the COM port will be correctly identified in **Device Manager**.

For more details on using `mt7681_uploader.exe` see section 3.2, 'Firmware uploader'.

2.4.3.2. Uploading the firmware from Linux

To upload the firmware from a Linux computer:

1) connect the LinkIt Connect 7681 development board to your computer using a USB cable, as shown in Figure 17.

2) identify the COM port on which the development board is connected by running `dmesg`, as shown below, and identify the port from the highlighted `ttyUSBx` statement:

```
shell$ dmesg
[   77.228120] usb 2-2: new full-speed USB device number 3 using ohci-pci
[   77.460318] usb 2-2: New USB device found, idVendor=0403, idProduct=6001
[   77.460326] usb 2-2: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[   77.460331] usb 2-2: Product: FT232R USB UART
[   77.460336] usb 2-2: Manufacturer: FTDI
[   77.460341] usb 2-2: SerialNumber: A602IU80
[   77.526011] usbcore: registered new interface driver usbserial
[   77.526379] usbcore: registered new interface driver usbserial_generic
[   77.527024] usbserial: USB Serial support registered for generic
[   77.530284] usbcore: registered new interface driver ftdi_sio
[   77.530726] usbserial: USB Serial support registered for FTDI USB Serial
Device
[   77.530861] ftdi_sio 2-2:1.0: FTDI USB Serial Device converter detected
[   77.531049] usb 2-2: Detected FT232RL
[   77.531052] usb 2-2: Number of endpoints 2
[   77.531056] usb 2-2: Endpoint 1 MaxPacketSize 64
[   77.531059] usb 2-2: Endpoint 2 MaxPacketSize 64
[   77.531062] usb 2-2: Setting MaxPacketSize 64
[   77.541287] usb 2-2: FTDI USB Serial Device converter now attached to
ttyUSB0
```

3) upload the firmware using `mt7681_uploader.py`, setting `-f` to specify the firmware generated in the previous step, and `-c` to identify the new COM port, as shown below:

```
> python mt7681_uploader.py -f out/sta/MT7681_sta_header.bin -c
/dev/ttyUSB0
```

4) successful upload of the firmware will result in the following being displayed:

```
Recovery Mode entered, Starting to upload...
138084/138084 bytes (100%) ... Update done! Bye
```

For more details on using `mt7681_uploader.py` see section 3.2, 'Firmware uploader'.

2.4.4.   Run the application on the LinkIt Connect 7681 development board

The LinkIt Connect 7681 development board will automatically reboot once the update process finishes successfully. You'll see the LED2 (mapped to GPIO2) start to blink every second.

2.4.5.   Troubleshooting

If you encounter issues with compiling or running your first firmware, please consult section 6, "Troubleshooting".

3. Tools guide

This section provides details on the use of the tools included in the MediaTek LinkIt Connect 7681 SDK.

3.1. make

`make` provides for building a collection of source code files into firmware for MT7681. The utility provides for four types of builds:

- `make -b=0` to build recovery mode firmware.

- `make -b=1` to build station mode firmware (the default build).

- `make -b=2` to build AP mode firmware.

- `make clean;make` to perform a full rebuild after cleaning all object files, useful particularly after making configurations in `makefile` or header files.

`make` should be run from a project's source root folder (for example `*:\LinkIt_Connect_7681_SDK\Src\`). Each build will create a log file, as follows:

- `out\sta\build_sta.log` for a Wi-Fi station image build.

- `out\sta\build_ap.log` for an access point (AP) image build.

3.2. Firmware uploader

The firmware uploader is provided as an executable on Windows (`mt7681_uploader.exe`) and source python script on Windows and Linux (`mt7681_uploader.py`). To execute the python script on Windows python 2.x is required — Linux provides native python support.

The syntax of the uploader for Windows are:

```
> mt7681_uploader.exe -f out\sta\MT7681_sta_header.bin -c COM7

Or

> python mt7681_uploader.py -f out\sta\MT7681_sta_header.bin -c COM7
```

While on Linux its:

```
> python mt7681_uploader.py -f out/sta/MT7681_sta_header.bin -c
/dev/ttyUSB0
```

The supported command switches are shown in Table 6.

Switch	Function
`-h: --help`	Displays a help message and exits the command
`-f: BIN_PATH`	Defines the path of the bin file to be uploaded
`-c: COM_PORT`	Defines the COM port the bin file will be sent to in the format COMx, for example COM1, COM2, and alike
`-b: BAUD_RATE`	Defines the Baud rate of the connection. Can be any valid baud rate, such as 9600 or 115200 (default)

Table 6 uploader command switches

Uploader works as follows:

1) sends an `AT#Reboot` to reboot MT7681 and place it in recovery mode.

2) sends `AT#UpdateFW`.

3) sends the specified firmware over the X-modem protocol to MT7681.

4) once the firmware transmission is complete, MT7681 will reboot itself automatically to run the new firmware.

If issues are encountered with uploads, please refer to the troubleshooting guide section 6.2, "Firmware upload errors".

3.3.   Additional tools

The following additional tools may be useful:

- a text/code editor or IDE suitable for editing C-style code.

- a terminal emulator for Windows, such as HyperTerminal, to issue AT commands. (In Ubuntu Linux you can use PuTTY.)

- a network analyzer, such as OmniPeek Network Analyzer, which will be helpful in debugging connection issues by sniffing the network traffic between MT7681 and a wireless AP or device connecting to MT7681 in AP mode.

- a Flash Programmer, such as Xilinx Platform Cable USB or Open Programmer to restore firmware should the uploader fail. For more information on Flash Programmers, see this Stack Exchange article.

4. Programming Guide

This section provides the basic information you need to understand how to create software for the MT7681. Included in this section are details on:

- creating your own software project source code library from the code provided.

- the structure of the source code files, including where the files you would modify are.

- the boot flow of the MT7681 and where your software is run.

- the user callbacks available for your custom software.

- the Wi-Fi state machine and its various modes.

- the API interfaces.

- MediaTek Smart Connect and the example apps provided.

4.1. Creating software projects

The coding style of MediaTek LinkIt Connect 7681 SDK is to modify existing source files stored in `*:\LinkIt_Connect_7681_SDK\Src\` to create custom firmware. Therefore when you want to create a new software project, you simply take a copy of the `Src` folder and make the necessary modifications to file in the copy.

4.2.   Source File Structure

The source files for MT7681 firmware must be stored in specific locations within a source folder (see Figure 19). This section describes how the files should be stored.
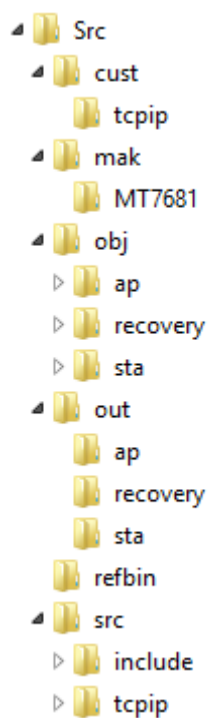
Figure 19 Example of the source files folder structure on Windows

4.2.1. Root Folder

The files stored in the root folder of a firmware source project are shown in Table 7

File	Purpose
`Makefile`	MT7681 Makefile
`libandesrecovery.a`	Binary library for recovery mode
`libandessta.a`	Binary library for station mode
`libandesap.a`	Binary library for AP mode
`header.sh`	Script used to append headers to compiled firmware. Firmware with header can then be uploaded to MT7681 using `mt7681_uploader`. The following files updates are made by this script: • `MT7681_sta.bin` → `MT7681_sta_header.bin` • `MT7681_ap .bin` → `MT7681_ap_header.bin` • `MT7681_recovery.bin` → `MT7681_recovery_header.bin`
`flash_merge.sh`	Script used to merge Station, AP, and Recovery image into a single full image. The following file updates are made by this script: • `MT7681_station.bin + MT7681_ap.bin + MT7681_recovery_old.bin` → `MT7681_all.bin` This full image can be written to MT7681 using a hardware flash programmer (burner) through the SPI interface. This is useful when the firmware uploader doesn't work. For more information see section 6.4, "Unresponsive board".

Table 7 Source files stored in the root folder

4.2.2. Customizable code folder

All source code that you may want to modify is stored in the `/cust` folder, as shown in Table 8.

File	Purpose
`iot_at_cmd.c`	AT command parsing and processing
`iot_at_cmd_utility.c`	AT command parsing utility
`iot_parse.c`	Process data commands received from Wi-Fi channel
`iot_parse_conn_mgmt.c`	Data command management
`iot_custom.c`	User callback, for more details see section 4.4, "User callbacks"
`iot_uart_rb.c`	UART Rx/Tx callback
`iot_custom_uart2wifi.c`	Example code for parsing data from UART and sending them through Wi-Fi
`iot_at_cmd_tcpip.c`	Example code for parsing AT commands and creating TCP/IP connections
`iot_xmodem.c`	X-modem protocol implementation
`iot_aes_pub.c`	Example code for AES
`iot_uplink.c`	Example code for packing data and sending them via UDP
`iot_crc16.c`	CRC16 source file
`main_pub.c`	Main function of MT7681 system

File	Purpose
`rtmp_data_pub.c`	Wi-Fi Rx packet handler
`wifi_task_pub.c`	Wi-Fi Task function
`spi-flash_pub.c`	Implementation for serial flash reading and writing
`misc_pub.c`	Implementation for delay functions
`mt7681_ChSw_pub.c`	Implementation for setting Wi-Fi channels
`stdlib.c`	Implementation for `malloc()`, `free()` and `atoi()`
`printf.c`	Implementation for log function: `printf_high()`
`ap_pub.c`	AP mode configuration initialization and PMK setting
`tcpip\*.c`	Source files and sample code for TCP/UDP

Table 8 Content of the customizable folder

4.2.3. Header files

Header files are stored in the `/src` folder in files with the format `libandesxxx.a`.

4.2.4. Build settings folder

Build configuration setting files are stored in the `/mak/MT7681` folder and include the files shown in Table 9.

File	Purpose
`compiler.mk`	Tool chain directory configuration
`flags_ap.mk`	Configuration for AP mode
`flags_sta.mk`	Configuration for Station mode
`flags_recovery.mk`	Configuration for Recovery mode

Table 9 The build configuration settings files

4.2.5. Object folder

Generated object code from the compiler is stored in the `/obj` folder.

4.2.6. Output folder

The final binaries generated by the compiler are stored in the `/out` folder. For example, the station mode firmware is located at `/out/sta/MT7681_sta_header.bin`. For more information, see section 3.1, "make".

4.2.7. Reference firmware binaries folder

Reference firmware binaries, guarantee to work original firmware binaries, are stored in the `/refbin` folder.

4.3. Firmware boot up flow

When source code is compiled the compiler generates `MT7681_all.bin`. — the software for MT7681. This software consists of four images that provide features related to Loader, Recover/Calibration, Station, and AP as described in Table 10.

Function	Image name	Purpose
Loader	`MT7681_loader.bin`	Responsible for loading images from Flash to SRAM
Recovery/Calibration	`MT7681_recovery_old.bin`	Responsible for firmware upgrade by UART and power and frequency calibration on the factory production line.
Station	`MT7681_sta.bin`	Image for Station mode.
Access point (AP)	`MT7681_ap.bin`	Image for AP mode.

Table 10 The four MT7681 software images

4.3.1. Boot up Sequence

The boot up flow for the MT7681, as shown in Figure 20, is as follows:

1) Power on
   MT7681 powers on and loads "Loader image"

2) Loader
   "Loader image" loads "Recovery Image" from Flash to SRAM. MT7681 then jumps to SRAM address 0 and starts to run.

3) Recovery/Calibration Mode
   "Recovery/Calibration Image" will start a 4 seconds timer and wait for a command from UART.

   o If no command is received before the timer expires, MT7681 proceeds to determining whether to load Station or AP mode.

   o If `AT#UpdateFW` is received on the UART, MT7681 goes to recovery mode and starts the X-modem protocol to wait new firmware. Using the firmware uploader, you can write new firmware with header content (`MT7681_sta_header.bin` or `MT7681_ap_header.bin`) to MT7681. For more details, see section 3.2, "Firmware uploader".

4) Determined Station or AP mode
   "Loader image" reads the "Firmware Boot Index" setting stored in Flash (stored in the Flash offset: 0x18001):, and load "station image" or "AP image" to SRAM:

   o If the value is 0x00 or 0xFF, "station image" is loaded to SDRAM.

   o If the value is 0x01, "AP image" is loaded to SDRAM.

   It is possible to change the value of "Firmware Boot Index" during Recovery mode by sending the AT command `AT#FLASH -s0x18001 -vX` where X is the new index value.

5) Station mode
   When "Station Image" is loaded, MT7681 is in station mode, it can connect to an AP-router, and create TCP/UDP connection with other devices in the LAN or WLAN

6) AP mode
   When "AP Image" is loaded, MT7681 is in AP mode. Other station devices (phone, tablet, or

PC) can connect to it, and it can create TCP/UDP connection with other devices in the LAN or WLAN
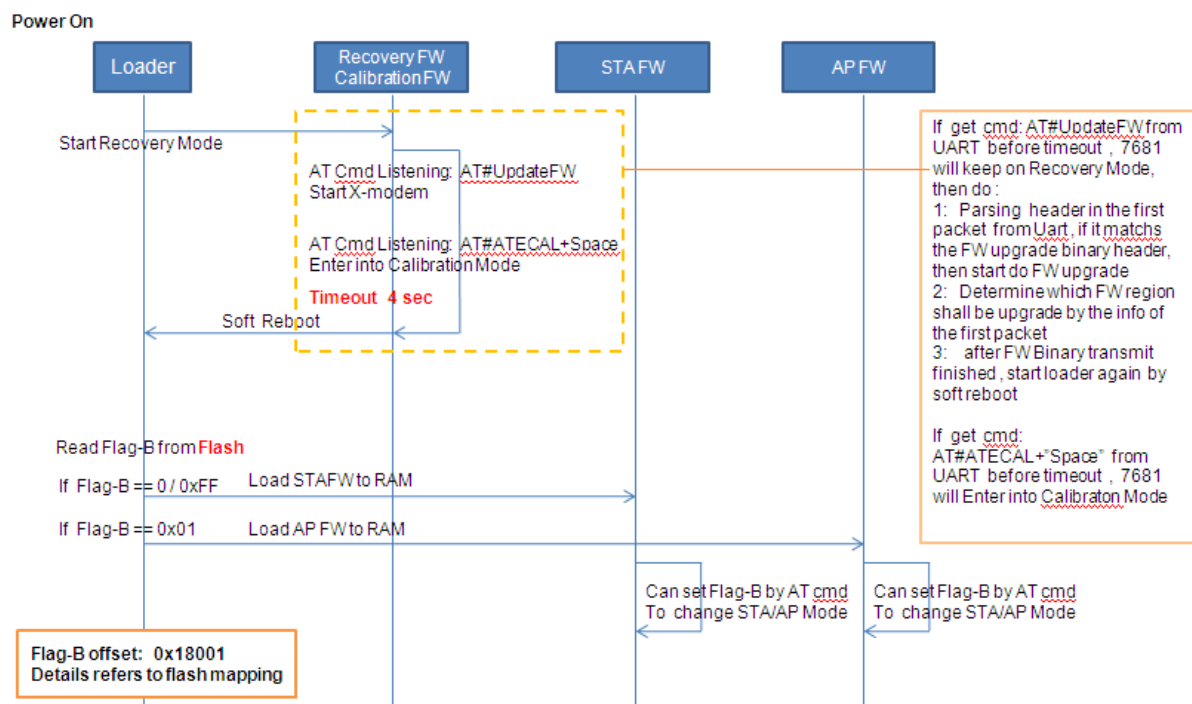
Figure 20 The boot sequence of MT7681

4.3.2. UART log at Boot up stage

During the boot sequence a log is written to UART, this is an example of the log for Station mode:

```
==> Recovery Mode  /*start to running recovery/Calibration image*/
<== Recovery Mode  /* end to running recovery/Calibration image*/
(-)                /*start to running station image*/
SM=0, Sub=0
SM=1, Sub=0
```

While this is an example of the log for AP mode:

```
==> Recovery Mode  /*start to running recovery/Calibration image*/
<== Recovery Mode  /* end to running recovery/Calibration image*/
(-)                /*start to running AP image*/
load_ap_cfg
store_ap_cfg
===> APStartUp
PMK Updating ...
AP SETTING: SSID[MT7681_AP1], AuthMode[9], WepStatus[8], Channel[8]
APStartUp ...  OK
[WTask]53637
```

4.4.   User callbacks

MT7681 firmware provides callback functions that will be called when specific events occur. The main callback functions you'll use are:

- `iot_cust_preinit()`

- `iot_cust_init()`

- `iot_cust_subtask1()`

Details on registering, the process flow, and introduction to these main callback are provided in this section.

4.4.1.   Registering callbacks

Any callbacks used must be registered in `iot_cust_ops()`, as shown in the code below:

```c
void iot_cust_ops(void)
{
    /*Callback in system Boot state, do not print msg as Uart is not
ready*/
    IoTCustOp.IoTCustPreInit = iot_cust_preinit;
    /*The callback in system initial state*/
    IoTCustOp.IoTCustInit    = iot_cust_init;

    ...

    /* The callback in  the wifi main task , it will be called
       every cycle of the wifi main task*/
    IoTCustOp.IoTCustSubTask1 = iot_cust_subtask1;

    ...
}
```

4.4.2.   Callback flowchart

The flowchart in Figure 21 shows the event timing and triggers for MT7681 callbacks.
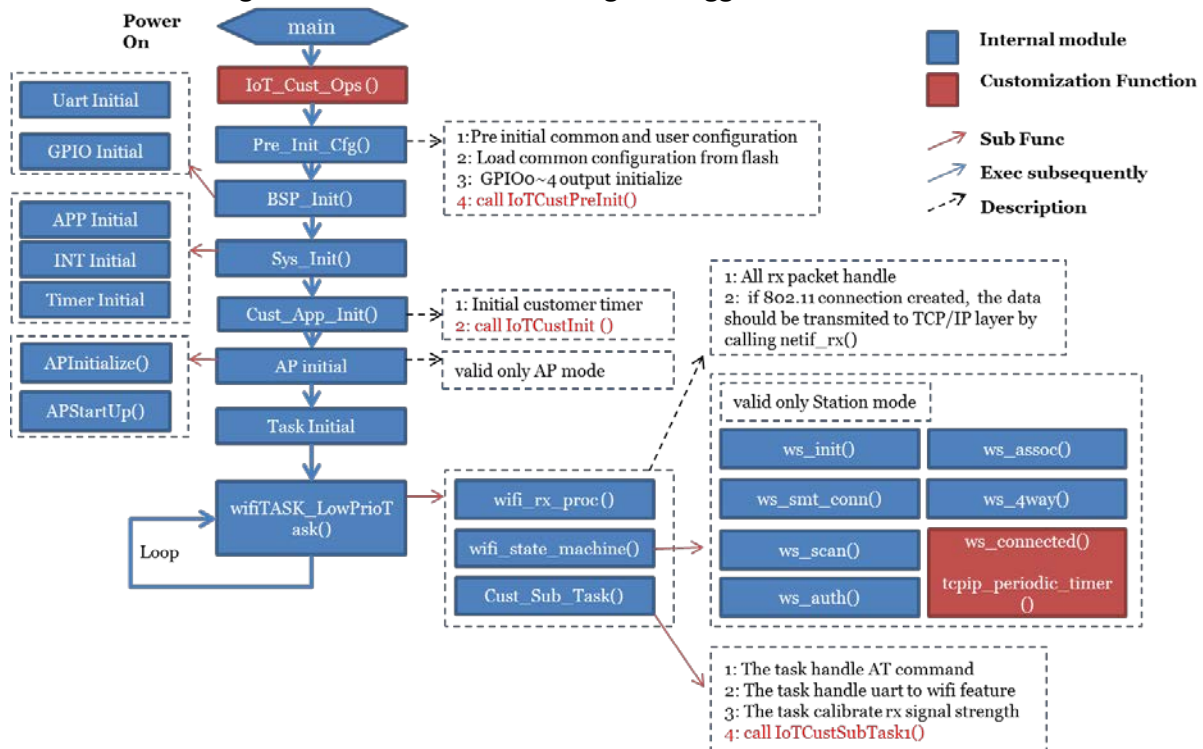
Figure 21 The flow of callback function in MT7681

4.4.3.   iot_cust_preinit ()

`iot_cust_preinit()` is called once at system start-up. Code should NOT do anything related to UART or GPIO here, because they are not initialized at this stage. It's recommended you place structure or global parameter initialization here.

4.4.4.   iot_cust_init()

`iot_cust_init()` is called once after module initialization has finished. It's recommended that UART and GPIO or customer software initialization is done here. The following is an example of code that could be placed in this callback.

```
void iot_cust_init(void)
{
    /* run customer initial function */
    printf_high("Hello world \n");
}
```

This code will result in the message "Hello world" being output from UART when MT7681 starts up, as follows:

```
==> Recovery Mode
<== RecoveryMode
(-)
SM=0, Sub=0
Hello world
[WTask]5001
[WTask]10001
```

4.4.5.　iot_cust_subtask1()

`iot_cust_subtask1()` is called as a sub-task of `wifiTASK_LowPrioTask()`, which in turn executes approximately once every 1ms or less, but this will depend on how long this callback takes to execute. You can add your own logic here, for example:

```
void iot_cust_subtask1(void)
{
    static int32 i=0;
    printf_high("Hello world [%d]\n", i++);
}
```

This code will result in the message "Hello world" being output from UART approximately once every 1ms.

```
==> Recovery Mode
<== RecoveryMode
(-)
SM=0, Sub=0
SM=1, Sub=0
Hello world [0]
Hello world [1]
Hello world [2]
Hello world [3]
Hello world [4]
```

4.4.6.　UART Tx and Rx callback

MT7681 registers a UART ISR handler callback for URT Tx and Rx where:

- `UART_Rx_Cb()` is the callback for the UART Rx data handler and the interrupt assert triggers as Rx FIFO gets close to full.

- `UART_Tx_Cb()` is the callback for the UART Tx data handler and the interrupt asserts as `IoT_uart_output()` gets close to filling the Tx buffer.

Smart Connection provides the callback `STARxDoneInterruptHandle()` that can check the contents of the Smart Connection Rx packet.

4.5. Wi-Fi State Machine (Station mode)

When MT7681 is in Station mode it provides a Wi-Fi state machine for the Wi-Fi connection. The entry function for this state machine is `wifi_state_machine()`.

4.5.1. States

There are in total 7 states, as shown in Figure 22:

- `WIFI_STATE_INIT`

  Initializes the Wi-Fi state machine and loads the target AP settings (SSID, Password, AuthMode, and PMK) which are stored in Flash. If the settings are valid, the machine goes to state `WIFI_STATE_SCAN`, otherwise it goes to `WIFI_STATE_SMTCNT`.

- `WIFI_STATE_SMTCNT`

  In this state the MT7681 will listen on Wi-Fi for a Smart Connection packet containing target AP settings (SSID, Password, AuthMode, and PMK):

  o if a Smart Connection packet is received, the machine goes to `WIFI_STATE_SCAN`.

  o otherwise the machine goes back to `WIFI_STATE_INIT`.

  If the AT command `AT#Smnt` is received, MT7681 will go into this state immediately.

- `WIFI_STATE_SCAN`

  When MT7681 gets target AP settings from `WIFI_STATE_INIT` or `WIFI_STATE_SMTCNT`, it starts to switch channels and scan for the target AP.

  o if a probe response is received from the target AP, MT7681 will stay in the current channel and check the AuthMode in the AP's Probe response frame, and update AuthMode setting to local. Then go to `WIFI_STATE_AUTH`.

  o if the expected probe response isn't received within 10 seconds, MT7681 goes back to `WIFI_STATE_INIT`.

- `WIFI_STATE_AUTH`

  MT7681 sends a Management frame (Auth Request) to the AP, then:

  o if an Auth Response is received with the status code `0` (Auth successful), MT7681 goes to `WIFI_STATE_ASSOC`.

  o if an Auth Response is received with status code not equal to `0`, a Deauth frame is received from the target AP, or an Auth Response frame isn't received within 300ms after retrying 5 times, MT7681 goes to `WIFI_STATE_INIT`.

  There are sub-states for the authentication process:

  o Sub-State = `0`, MT7681 is preparing to send Auth Request.

  o Sub-State = `1`, MT7681 is waiting for Auth Response.

- `WIFI_STATE_ASSOC`

  MT7681 sends a Management frame (Assoc Request) to AP and waits for a response:

  o if the Assoc Response is received with the status code =0 (Assoc successful), then:

    ▪ if AuthMode is 0 (Open mode), MT7681 goes to `WIFI_STATE_CONNED`.

    ▪ if the AuthMode is 4 (WPAPSK mode), 7 (WPA2PSK mode), or 9 (WPA/WPA2PSK mixed mode), MT7681 goes to `WIFI_STATE_4WAY`.

  o if Assoc Response is received with status code not equal 0, a Deassoc frame is received from the target AP, or an Assoc response frame isn't received within 300ms after retrying 5 times, MT7681 goes to `WIFI_STATE_INIT`.

  There are sub-states for the Assoc process:

  o Sub-state = 0, MT7681 is preparing to send an Assoc request.

  o Sub-state = 1, MT7681 is waiting for an Assoc response.

- `WIFI_STATE_4WAY`

  MT7681 and the target AP undertake a 4 Way Handshake process and negotiate the PTK (Pairwise Key) and GTK (Group Key) used for encrypting/decrypting unicast and broadcast frames when MT7681 and AP communicate:

  o if the 4 Way Handshake is successful, MT7681 goes to `WIFI_STATE_CONNED`.

  o if 4 Way Handshake fails or the 4 Way Handshake process times out (>10s), MT7681 goes to `WIFI_STATE_INIT`.

- `WIFI_STATE_CONNED`

  When MT7681 enters into this state from `WIFI_STATE_ASSOC` or `WIFI_STATE_4WAY` it sends a DHCP discovery request to apply for an IP address from the AP and the AP should assign an IP address to MT7681.

  Once the IP address has been acquired MT7681 should stay in this state until powered down, however it will go to `WIFI_STATE_INIT` if:

  o Beacon is not received from target AP within 20 seconds.

  o Deauth or Deassoc is received through a unicast from the AP.

  o IP address is not received from DHCP and a retry times out (1 minute).

  o AT command `AT#Reboot` is received.
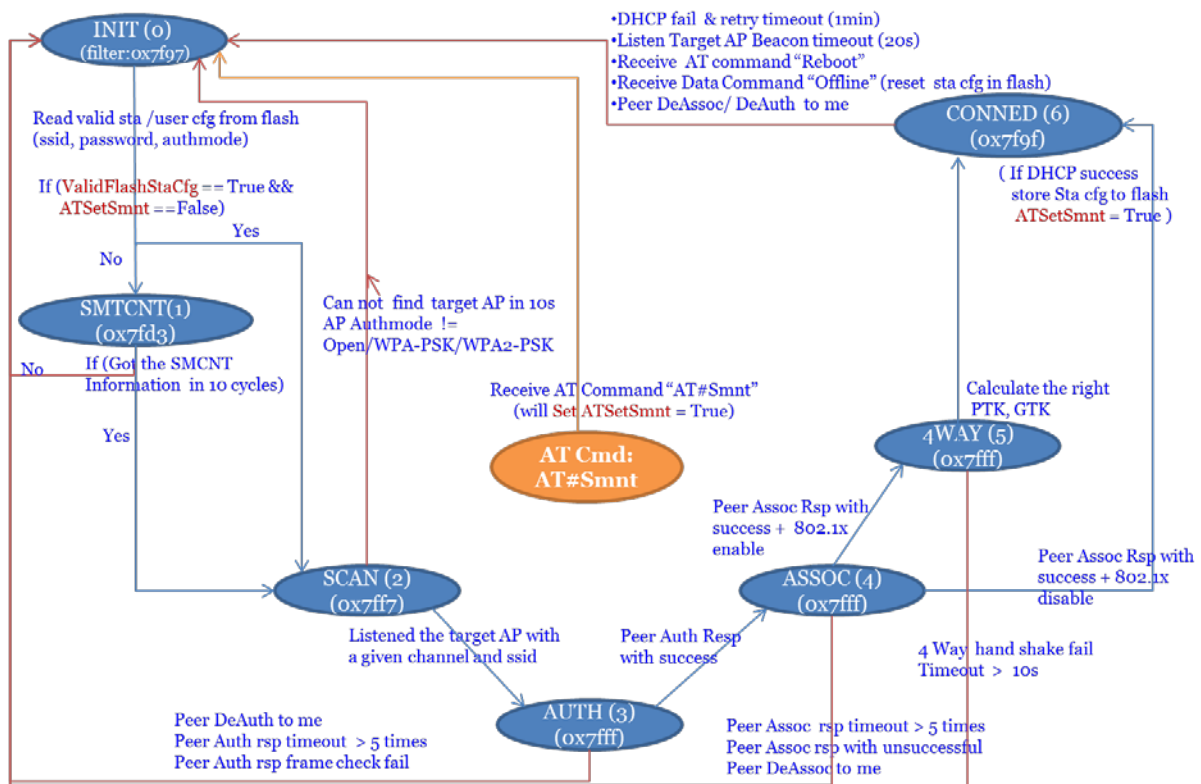
Figure 22 The Wi-Fi state machine

A log is written to UART by the state machine, as shown in this example:

```
==> Recovery Mode   /* start to running recovery/Calibration image*/
<== Recovery Mode   /* end to running recovery/Calibration image*/
(-)                 /* start to running station image*/
SM=0, Sub=0         /* Change to "WIFI_STATE_INIT" */
SM=1, Sub=0         /* Change to "WIFI_STATE_SMTCNT" */
[WTask]5001         /*--output timer log period to notice system alive--*/
SM=2, Sub=0         /* Change to "WIFI_STATE_SCAN" */
SM=3, Sub=0         /* Change to "WIFI_STATE_AUTH", is going to send Auth
Request*/
Auth with:ssid = belkin, auth mode = 9,  /*--output the target AP's
information "SSID, AuthMode"-- */
SM=3, Sub=1         /* Change to "WIFI_STATE_AUTH" is waiting Auth
Response*/
SM=4, Sub=0         /* Change to "WIFI_STATE_ASSOC", is going to send Assoc
Request*/
SM=4, Sub=1         /* Change to "WIFI_STATE_ASSOC", is waiting Assoc
Response */
SM=5, Sub=0         /* Change to "WIFI_STATE_4WAY" */
[WTask]10005
SM=6, Sub=0         /* Change to "WIFI_STATE_CONNED", start to get IP*/
[WTask]15006
Got IP:192.168.2.12/* Got IP from target AP*/
[WTask]20009
```

4.6.   API Interface Guide

This section provides an overview to the APIs available in MT7681, for a complete guide to these APIs please refer to the MediaTek LinkIt Connect 7681 API Reference.

4.6.1. Access Point (AP) mode

MT7681 can be set in access point (AP) mode. In this mode stations are able to connect to MT7681 and get an IP address from MT7681.

MT7681 Station mode supports 4 AuthModes, they are: Open, WPA-PSK, WPA2-PSK, and WPA/WPA2-PSK.

MT7681 supports AP mode by software emulation, so there are some limitations:

- a maximum of 3 stations can connect to an MT7681 AP at the same time.
- MT7681 AP mode does NOT support "frame buffering", it will keep stations alive by sending them a beacon with `TIM=1`. Frame Buffering means AP can buffer frames for a station when that station is in power save mode and transmit data later. It's more power efficient if a station can turn transmitter and receiver on only when necessary. MT7681 has a small RAM, so it cannot buffer frames for stations.
- MT7681 AP mode only supports 802.11b/g.

4.6.2. Station mode

MT7681 can be set in station mode. In this mode it's able to connect to an AP and get an IP address from the AP. When MT7681 is connecting to an AP, it can establish a TCP/IP socket connection to send or receive data between remote servers.

Note the following about Station mode, it:

- supports 4 authentication modes: Open, WPA-PSK, WPA2-PSK, and WPA/WPA2-PSK.
- supports 802.11b/g/n.
- does NOT support QoS (Quality of Service) and A-MPDU (Aggregate MAC Protocol Data Unit) protocols.

4.6.3. PWM Interface

MT7681 offers 5 GPIO pins, each GPIO pin can be set to PWM mode with specific frequency and duty cycle.

Note the following about the PWM interface:

- the highest PWM frequency is 1 KHz, but in this case there are only 2 duty cycle options: 0%, 100%.
- for a PWM frequency of 50Hz, there are 20 duty cycle options: 0% to 100% in 5% increments.
- the frequency and duty cycle are defined by setting `PWM_HIGHEST_LEVEL` in the source file. Please refer to PWM section in the MediaTek LinkIt Connect 7681 API Reference for more detail.

4.6.4. Timer

MT7681 has 2 hardware timers.

The counter precision of the hardware timers is 1ms.

4.6.5. TCP/IP Stack

MT7681 uses uIP (micro IP), which is an open source TCP/IP stack used in many embedded systems. It supports TCP, UDP, DNS, DHCP, and HTTP client.

> The maximum number of TCP/UDP connection supported by MT7681 is 4. The maximum TCP/UDP packet length is 600 Bytes. This configuration is best suited to MT7681's RAM.
>
> These limits are defined in `uip-conf.h`, details are as below:
>
> ```
> #define UIP_CONF_MAX_CONNECTIONS 4
> #define UIP_CONF_UDP_CONNS       4
> #define UIP_CONF_BUFFER_SIZE     600
> ```

4.7. Smart Connection Guide

As devices containing an MT7681 have no user interface there is no mechanism for users to enter IP configuration details and connect to an AP. To overcome this issue MediaTek Smart Connection is provided. It provides a mechanism whereby a wireless network's information (SSID, password, and AuthMode) is broadcast. When powered up the device containing an MT6781 listens to detect the broadcast, and can consume it to obtain its wireless network settings. The transmissions are also encrypted with a pre-defined key to ensure security.

The SDK provides the Smart Connection library for Android and iOS. There is also an example app for both included in the SDK. This section describes the APIs in that library and how to use them.

4.7.1. Smart Connection API for Android

4.7.1.1. StartSmartConnection

This API packages SSID, Password, and AuthMode into the MediaTek Smart Connection format and asks the Wi-Fi driver to broadcast to the air. If an MT7681 in Smart Connection mode is listening, it can receive the information and connect to the defined network. The syntax of this API are as follows:

```
int StartSmartConnection(const char *SSID, const char *Password, char
AuthMode);
```

The parameters used in this API are listed in Table 11.

Parameter	Description
SSID	SSID of AP-router that the MT7681 should connect to
Password	Password of AP-router that the MT7681 should connect to
AuthMode	Authentication mode. The value can be: 0: Open 4: WPA-PSK 7: WPA2-PSK 9: WPA/WPA2-PSK

Table 11 Parameters of StartSmartConnection API

The return values from the API are listed in Table 12.

Returned value	Meaning
0	Success
Other	Error (reserved for future use).

Table 12 Return value of StartSmartConnection API

4.7.1.2. StopSmartConnection

This API stops the Wi-Fi broadcast of the Smart Connection packet. The syntax of this API are as follows:

```
int StopSmartConnection(void);
```

The return values from the API are listed in Table 13.

Returned value	Meaning
0	Success
Other	Error (reserved for future use).

Table 13 Return value of StopSmartConnection API

4.7.1.3. Example app

The SDK folder `APK\Android` contains an example APK to demonstrate MediaTek Smart Connection. The APK contains IoT Manager, which provides a simple UI, shown in Figure 23, enabling the user to input SSID, Password, and AuthMode. There are also buttons to start and stop the broadcast of the Smart Connection package.
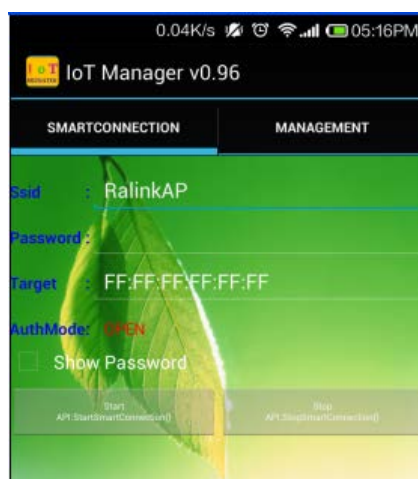
Figure 23 The UI of the IoT Manager app

When the user taps **Start**, the app runs to JNI_StartSmartConnection() that calls StartSmartConnection to begin broadcasting. The code for this is shown below:

```
File: IoTManage_jni.cpp

static jint JNI_StartSmartConnection(JNIEnv *env, jobject thiz, jstring
nSSID, jstring nPassword, jbyte nAuth)
{
    int iRst = 0;
    const char *pSSID = NULL;
    const char *pPassword = NULL;

    pSSID = env->GetStringUTFChars(nSSID, 0);
    pPassword = env->GetStringUTFChars(nPassword, 0);

    iRst = StartSmartConnection(pSSID, pPassword, (char)nAuth);
    if (iRst != 0)
    {
        HWTEST_LOGD("StartSmartConnection error.");
    }
    HWTEST_LOGD("Leave JNI_StartSmartConnection.");
    return iRst;
}
```

When the device containing the MT7681 is turned on, it will enter Smart Connection mode, receive the broadcast packets and attempt to connect to the defined AP.

During this process the MT7681 writes a log to UART, here is an example:

```
==> Recovery Mode  /*start to running recovery/Calibration image*/
<== Recovery Mode  /* end to running recovery/Calibration image*/
(-)                /* start to running station image*/
SM=0, Sub=0        /* Change to "WIFI_STATE_INIT" */
SM=1, Sub=0        /* Change to "WIFI_STATE_SMTCNT" */
[WTask]5001        /*--output timer log period to notice system alive--*/
SM=2, Sub=0        /* Change to "WIFI_STATE_SCAN" */
SM=3, Sub=0        /* Change to "WIFI_STATE_AUTH", is going to send Auth
Request*/
Auth with:ssid = RalinkAP, auth mode = 0,  /*--output the target AP's
information "SSID, AuthMode"-- */
SM=3, Sub=1        /* Change to "WIFI_STATE_AUTH" is waiting Auth
Response*/
SM=4, Sub=0        /* Change to "WIFI_STATE_ASSOC", is going to send Assoc
Request*/
SM=4, Sub=1        /* Change to "WIFI_STATE_ASSOC", is waiting Assoc
Response */
[WTask]10005
SM=6, Sub=0        /* Change to "WIFI_STATE_CONNED", start to get IP*/
[WTask]15006
Got IP:192.168.2.12/* Got IP from target AP*/
[WTask]20009
```

When MT7681 completes the Smart Connection process (it has connected to the target AP and obtained an IP address), it will store the configuration to Flash. Then, when MT7681 restarts it will load the setting from Flash and directly go to Scan state, instead of Smart Connection state.

To force MT7681 to redo the Smart Connection process, either use AT command `AT#Smnt`" to switch MT7681 to Smart Connection state or use AT command `AT#Default` to perform a full reset and restart.

4.7.2.　　Smart Connection API for iOS

A similar interface to that provided for Android is offered for iOS. Compared to coding a Smart Connection app for Android, iOS requires an additional step `Init` before `Start`, as shown below:

```
#ifndef _SMTIOT_H_
#define _SMTIOT_H_

int InitSmartConnection(void);
int StartSmartConnection(const char *SSID, const char *Password,
                         const char *Target, char AuthMode);
int StopSmartConnection(void);

#endif
```

5.   Using the APIs

This section provides information on how to use the MT7681 APIs. For details of the API please refer to the MediaTek LinkIt Connect 7681 API Reference.

The topics covered in this section include:

- connecting MT7681 to an AP.

- configuring MT7681 as an AP.

- using uIP for TCP/IP operations, including:

    o   creating a TCP client that connects to a remote TCP server.

    o   creating a TCP server and allowing others to connect to it.

    o   increasing the available TCP/UDP connections.

5.1.   Connecting MT7681 to an AP

This section describes how to code MT7681 so that it will always connect to the specified AP. This is useful when you don't want to use Smart Connection but have a specific AP you want to connect to. The steps to achieve this are as follows:

1)   disable default Smart Connection

Open `mak\MT7681\flags_stak.mk` (this is the make file for Station mode), scroll to the bottom and add following lines before `#END-OF-FILE#`.

```
...
FUNCFLAGS += -DCFG_SUPPORT_MTK_SMNT=0  # add this line to enable MTK
SMNT
#END-OF-FILE#
```

2)   add the specific AP setting to `IoT_Cust_SM_Smnt()`.

3) `IoT_Cust_SM_Smnt()` is located in `cust\Iot_custom.c`. Modify the default value of SSID and Passphrase variables to match AP you wish to connect to, as follows:

```c
void IoT_Cust_SM_Smnt(void)
{
    /* Example for customer's smart connection implementation*/

    /* Step1:  create parameter to store the smnt information */
    /* or it is better to create a structure*/
    uint8  Ssid[MAX_LEN_OF_SSID+1] = "myap";        <-- Sepecify SSID here
    uint8  Passphase[CIPHER_TEXT_LEN] ="12345678";  <-- Sepecify Key here
    uint8  PMK[CIPHER_TEXT_LEN];

    ...

    /*Step3:  After smart connection done */
    /* it need set smart connection information , then start to scan*/
    /*The Authmode shall be detected in scan state again, if
AuthMode>WPA, PMK will be calculated in scan state*/
    pIoTStaCfg->AuthMode     = Ndis802_11AuthModeOpen;
    pIoTStaCfg->SsidLen      = strlen(Ssid);
    pIoTStaCfg->PassphaseLen = strlen(Passphase); //sizeof(Passphase);
    memcpy(pIoTStaCfg->Ssid,      Ssid,      pIoTStaCfg->SsidLen);
    memcpy(pIoTStaCfg->Passphase, Passphase, pIoTStaCfg->PassphaseLen);
    memcpy(pIoTStaCfg->PMK,  PMK, strlen(PMK));

    wifi_state_chg(WIFI_STATE_SCAN, 0);  /* Change wifi state to SCAN*/
}
```

AuthMode can be left unchanged, MT7681 will automatically determine authentication during scanning.

`MAX_LEN_OF_SSID` and `CIPHER_TEXT_LEN` are both 32. Don't specify strings longer than 31 characters.

4) run `make` to rebuild the project and use uploader to upload `MT7681_sta_header.bin` to the device or development board.

Running the code on the device or development board will connect to the specified AP and the following log will be sent to UART:

```
SM=0, Sub=0
SM=1, Sub=0
SM=2, Sub=0
SM=3, Sub=0
Auth with:ssid = myap, auth mode = 9, /* SSID specified */
SM=3, Sub=1
SM=4, Sub=0
SM=4, Sub=1
SM=5, Sub=0
[WTask]9301450
SM=6, Sub=0
Got IP:192.168.2.19
```

5.2. Configuring MT7681 as an AP

This section describes how to code MT7681 so that it acts as an AP and lets other devices (such as mobile phones) connect to it and retrieve data. The steps to achieve this are as follows:

1) Set the default configuration of MT7681 to AP mode.

   Set the default configuration (SSID, password, and authentication mode) in the macro ap_pub.c, as follows:

```
#define AP_MODE_OPEN                    0
#define AP_MODE_WPAPSK_TKIP             1
#define AP_MODE_WPA2PSK_AES             2
#define AP_MODE_WPA1WPA2PSK_TKIPAESMIX  3

/*MT7681's Default Authtication mode in AP mode*/
#define DEFAULT_AP_MODE     AP_MODE_WPA1WPA2PSK_TKIPAESMIX
/*MT7681's Default SSID in AP mode*/
#define Default_Ssid        "MT7681_AP1"
/*MT7681's Default Password in AP mode*/
#define Default_Password    "12345678"
```

   If there are no valid AP settings in Flash MT7681 uses these default settings, which are stored to Flash by load_ap_cfg() when MT7681 starts up.

   The position of the AP setting in the Flash is shown in Figure 24.

Flash Layout						AP Mode Config/Setting			
Offest	Section	Size (KB)	HEX (Byte)	DEC Offset		Offest	Section	Size (Byte)	DEC Offset
0x0000	Loader	20	0x5000	0		0x1A000	AP Info Stored Flag (1 Byte)	1	0
0x5000	reserved 1	4	0x1000	20480		0x1A001	BSSID (6 Byte) /*not used, but use MAC in EEPROM region*/	6	1
0x6000	Recovery Mode FW	64	0x10000	24576		0x1A007	SSID (32 Byte)	32	7
0x16000	reserved 2	4	0x1000	90112		0x1A027	SSID Len (1 Byte)	1	39
0x17000	EEPROM	4	0x1000	94208		0x1A028	Physical Mode(1 Byte)	1	40
0x18000	Common config	4	0x1000	98304		0x1A029	AP Channel (1 Byte)	1	41
0x19000	Station Mode Config		0x1000	102400		0x1A02A	Auth Mode (1Byte)	1	42
0x1A000	AP Mode Config	4	0x1000	106496		0x1A02B	AP Password (32 Byte)	32	43
0x1B000	User Config	4	0x1000	110592		0x1A04B	AP Password Len (1 Byte)	1	75
						0x1A04C	Beacon Interval (2 Byte)	2	76
						0x1A04E	DTIM Count (1Byte)	1	78
						0x1A04F	DTIM Interval (1Byte)	1	79
						0x1A050	Capability Info (2 Byte)	2	80
						0x1A052	fgIsHidden_ssid(1 Byte)	1	82
						0x1A053	Reseved 1	x	83

Figure 24 The position of the AP setting in Flash

> The AP configuration can be changed while MT7681 is running using the AT command AT#SoftApConf -s[SSID] -p[Password] -c[Channel] -a[AuthMode]. This command will not store the modified AP settings to Flash, to do this the AT command AT#SoftApConf -m0 is used.

2) Boot MT7681 to AP mode

   During startup the boot loader reads the Firmware Boot Index setting from Flash to determine whether the Station image or AP image is to be loaded to SRAM and run. For more information see 4.3, "Firmware boot up flow".

   Changing the value of Firmware Boot Index during Recovery mode by sending AT command AT#FLASH -s0x18001 -v1 will instruct MT7681 to boot to AP mode.

This is the UART log for MT7681 AP mode after boot up:

```
==> Recovery Mode  /*start to running recovery/Calibration image*/
<== Recovery Mode  /*end to running recovery/Calibration image*/
(-)                /*start to running AP image*/
load_ap_cfg
store_ap_cfg
===> APStartUp
PMK Updating ...
AP SETTING: SSID[MT7681_AP1], AuthMode[9], WepStatus[8], Channel[8]
APStartUp ...  OK
[WTask]53637
```

MT7681 is now ready to accept Wi-Fi client connections.

Refer to section 5.3.1, "Creating a TCP client that connects to a remote TCP server" for detail log of client connection.

> In AP mode only 3 clients may connect to a MT7681 at the same time.

5.3. Using uIP for TCP/IP operations

This section provides an introductory guide to using uIP (micro IP), the TCP/IP stack included in the MT7681. For more information on this stack refer to the uIP Reference Manual (uip-refman.pdf) available from the source repository.

The design principle of uIP is that applications must provide an `APP_CALL` callback function that uIP calls whenever an event occurs. Several APIs are provided to check these events:

- if `uip_newdata()` returns a non-zero value, a remote host has sent new data that are stored in the `uip_appdata` global variable.

- if `uip_poll()` returns a non-zero value, the connection is idle, and you can send new data to the remote host. For more information, see chapter 1.6 of the uIP Reference Manual.

In MT7681 SDK this callback is `iop_tcp_appcall()`, which is in `cust\tcpip\iot_tcp_app.c`.

5.3.1.   Creating a TCP client that connects to a remote TCP server

This section describes how to create a TCP client, connect to a remote TCP server and exchange data with that remote server.

1) Initialize the connection

Use `uip_connect()` to create the connection. In the example below, a connection to port 9999 of 192.168.1.1 with the local port is set as 8888 is defined:

```c
File: iot_tcp_app.c

void app_init_connection(void)
{
    UIP_CONN *tcp_conn=NULL;
    uip_ipaddr_t raddr;
    UINT8 iot_srv_ip[MAC_IP_LEN] = {192, 168, 1, 1};

    uip_ipaddr(raddr, iot_srv_ip[0],iot_srv_ip[1],
iot_srv_ip[2],iot_srv_ip[3]);

    /* Specify remote address and port here. */
    tcp_conn = uip_connect(&raddr, HTONS(9999));

    if(tcp_conn) {
        tcp_conn->lport = HTONS(8888);
    } else {
        printf_high("connect fail\n");
    }
}
```

This step can be actioned at any time, including before MT7681 has connected to a wireless network.

2) Exchange data with Remote Server

uIP calls `iot_tcp_appcall()` whenever an event occurs, so this is where you write the code to handle events.

First check to see if this event is related to the connection defined above, by checking whether the local port matches the one defined. If it does, use `uip_newdata()` to check for received data and `uip_poll()` to check if there is any data to be sent.

In this case the app prints the received data to UART and includes a timer that sends "hello" to UART every 5 seconds also.

```
File: iot_tcp_app.c

void app_handle_connection(void)
{
    static struct timer user_timer; //create a timer;
    static bool app_init = FALSE;

    if(uip_newdata()) // there are data to be received and stored in
uip_appdata
    {
        printf_high("TCP Client RX [%d] bytes\n", uip_datalen());
        iot_uart_output(uip_appdata, uip_datalen());
    }

    if (uip_poll())
    {
        if((app_init == FALSE) || timer_expired(&user_timer))
        {
            printf_high("TCP Client timer is expired, TX\n");
            uip_send("hello\n", 6);
            timer_set(&user_timer, 5*CLOCK_SECOND);
            app_init = TRUE;
        }
    }
}

iot_tcp_appcall(void)
{
    u16_t lport = HTONS(uip_conn->lport);

    if (lport == 8888) // if local port is 8888, it is our event
    {
        app_handle_connection();
    }

    ...
}
```

To test the code, first make sure that MT7681 has network access to the remote server. Second, have the remote server listen to port 9999. You can use nc (netcat), which is built-in to Linux and available for Windows, to listen on the port.

```
> nc -l 9999
```

The shell window at the remote server will start displaying "hello" periodically. You can also type characters back, these should be output on UART of MT7681.

5.3.2.   Create a TCP server and allow others to connect

This section describes how to create a TCP server, listen to a port and exchange data with a connection to that port.

1) Listening to the Port

Use `uip_listen()` to listen on a specified port. The following code listens to port 9999:

```
File: iot_tcp_app.c

void app_init_connection(void)
{
    uip_listen(HTONS(9999));
}


void iot_tcp_app_init (void)
{
    ...

    app_init_connection();
}
```

2) Exchange data with TCP Client

The logic and source code in `iot_tcp_appcall` is the same as used in section5.3.1. It prints out whenever data are received and sends back "hello" once every 5 seconds:

```c
File: iot_tcp_app.c

void app_handle_connection(void)
{
    static struct timer user_timer; //create a timer;
    static bool app_init = FALSE;

    if(uip_newdata()) // there are data to be received and stored in
uip_appdata
    {
        printf_high("TCP Server RX [%d] bytes\n", uip_datalen());
        iot_uart_output(uip_appdata, uip_datalen());
    }

    if (uip_poll())
    {
        if((app_init == FALSE) || timer_expired(&user_timer))
        {
            printf_high("TCP Server timer is expired, TX\n");
            uip_send("hello\n", 6);
            timer_set(&user_timer, 5*CLOCK_SECOND);
            app_init = TRUE;
        }
    }
}

iot_tcp_appcall(void)
{
    u16_t lport = HTONS(uip_conn->lport);

    if (lport == 9999) // if local port is 9999, it is our event
    {
        app_handle_connection();
    }

    ...
}
```

To test the code, first make sure that MT7681 has network access. Second, have the remote client connect to the IP of MT7681 (the IP address is output to UART during the acquisition process). You can use nc (netcat), which is built-in to Linux and available for Windows, to do this:

```
> nc [IP of MT7681] 9999      // Ex. nc 192.168.1.1 9999
```

The shell window at the remote client will start displaying "hello" periodically. You can also type characters back, these should be output on UART of MT7681.

> Placing the client and server (MT7681) under the same network will avoid potential issues with local IP and port forwarding.

5.3.3. Increasing available TCP/UDP connections

The maximum number of connections that can be made at the same time is defined, as shown below, in `uip-confg.h`.

While it isn't recommended, the number of connections can be increased, however, be aware that doing so will use more memory and may cause run-time problems.

```
File: uip-conf.h

/**
 * Maximum number of TCP connections.
 *
 * \hideinitializer
 */
#define UIP_CONF_MAX_CONNECTIONS 4

/**
 * Maximum number of listening TCP ports.
 *
 * \hideinitializer
 */
#define UIP_CONF_MAX_LISTENPORTS 4

/**
 * Maximum number of UDP connections..
 *
 * \hideinitializer
 */
#define UIP_CONF_UDP_CONNS  4
```

6. Troubleshooting

This section provides details on methods of troubleshooting issues with code on MT7681.

6.1. Compile Errors

If you're unable to compile the firmware:

1) locate the build logs as follows

   o if you are building a Wi-Fi station image (`make` or `make b=1`), the log is located at `out/sta/build_sta.log`.

   o if you are building an access point (AP) image (`make b=2`), the log is located at `out/stad/build_ap.log`.

2) review the log file to determine the cause.

6.1.1. GCC not found

An error where the GCC was not found will be indicated by:

```
In File :  out/sta/build_sta.log

Compiling src/wifi/wifi_task_pub.c ...
/bin/sh: /mtktools/Andestech/BSPv320/toolchains/nds32le-elf-newlib-
v2j/bin/nds32le-elf-gcc: No such file or directory
```

The most likely issues are that:

- the toolchain is setup incorrectly. Refer to section 2.2, "Installing MediaTek LinkIt Connect 7681 SDK for Windows" or section 2.3, "Installing MediaTek LinkIt Connect 7681 SDK for Linux" as appropriate and confirm the setup is correct.

- there is something wrong in the makefile (`mak/compiler.mk`). Check the makefile content.

6.1.2. Compile error in specified c file

A compile error in a c file will be indicated by output similar to:

```
Compiling cust/iot_custom.c ...
echo Compiling cust/iot_custom.c ... >> out/sta/build_sta.log
Makefile:264: recipe for target 'obj/sta/iot_custom.o' failed
make: *** [obj/sta/iot_custom.o] Error 1
```

6.1.2.1. Syntax error

A syntax error will be identified in the log file as shown in this example:

```
cust/iot_custom.c:187: error: expected ',' or ';' before 'if'
```

Which indicates that a ";" was missed from the end of the statement.

6.1.2.2. Missing variable

A missing variable error will be identified in the log file as shown in this example:

```
cust/iot_custom.c:187: error: 'curr_time_2' undeclared (first use in this
function)
cust/iot_custom.c:187: error: (Each undeclared identifier is reported only
once
cust/iot_custom.c:187: error: for each function it appears in.)
```

The root cause is either an undeclared variable or a miss spelt variable name.

6.1.2.3. Undefined reference to 'xxxxx'

An undefined reference (API not found) error will be identified in the log file as shown in this example:

```
obj/sta/iot_custom.o: In function `iot_cust_subtask1':
(.text.iot_cust_subtask1+0x6): undefined reference to `iot_get_ms__time'
obj/sta/iot_custom.o: In function `iot_cust_subtask1':
(.text.iot_cust_subtask1+0xa): undefined reference to `iot_get_ms__time'
Makefile:284: recipe for target 'out/sta/MT7681_sta.elf' failed
make: *** [out/sta/MT7681_sta.elf] Error 1
```

The root cause is a spelling error in the function name, correct the function spelling (remembering that function names are case sensitive).

6.2. Firmware upload errors

Uploading firmware may not work as expected occasionally. This may be indicated by:

- the firmware uploader displaying the message **Failed to enter 'Recovery Mode' automatically, please press reset key manually**.

- the board not responding as expected after a firmware update.

In these cases, use the reset switch to put the LinkIt Connect 7681 development board into recovery mode, as shown in Figure 25. The board should restart and run the firmware as expected. If the board still isn't behaving as expected, consult section 6.4, "Unresponsive board".
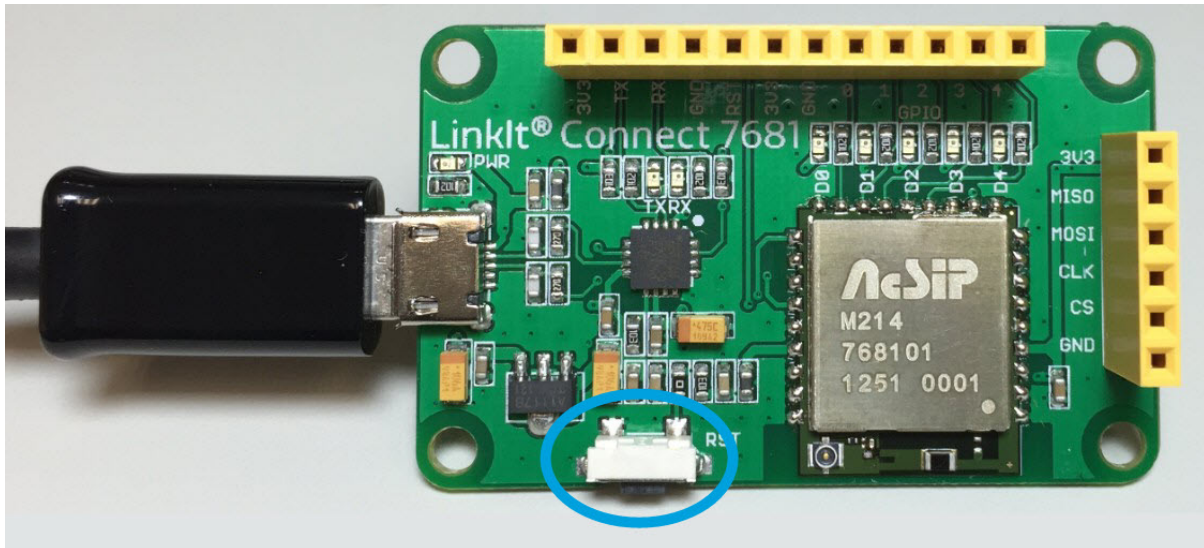
Figure 25 The reset switch on the LinkIt Connect 7681 development board

6.3.   Use Log to debug

MT7681 provides features for outputting debug messages to a log on UART. To enable this feature, set `PRINT_FLAG` in `iot_custom.c` to `TRUE` as follows:

```
bool PRINT_FLAG = TRUE;
```

6.3.1.   Using log to debug in Station mode

If MT7681 is in station mode with `PRINT_FLAG=TRUE` and is connected to a wireless AP defined by Smart Connection, the following is a typical debug log:

```
==> Recovery Mode   /* start to running recovery/Calibration image*/
<== Recovery Mode   /* end to running recovery/Calibration image*/
(-)                 /* start to running station image*/
SM=0, Sub=0         /* Change to "WIFI_STATE_INIT" */
SM=1, Sub=0         /* Change to "WIFI_STATE_SMTCNT" */
[WTask]5001         /*--output timer log period to notice system alive--*/
SM=2, Sub=0         /* Change to "WIFI_STATE_SCAN" */
SM=3, Sub=0         /* Change to "WIFI_STATE_AUTH", going to send Auth
Request*/
Auth with:ssid = RalinkAP, auth mode = 0, /* target AP's SSID and AuthMode
*/
SM=3, Sub=1         /* Change to "WIFI_STATE_AUTH" waiting for Auth
Response*/
SM=4, Sub=0         /* Change to "WIFI_STATE_ASSOC", going to send Assoc
Request*/
SM=4, Sub=1         /* Change to "WIFI_STATE_ASSOC", waiting Assoc Response
*/
[WTask]10005
SM=5, Sub=0         /* Change to "WIFI_STATE_4WAY", start to 4 way
handshake*/
SM=6, Sub=0         /* Change to "WIFI_STATE_CONNED", start to get IP*/
[WTask]15006
Got IP:192.168.2.12/* Got IP from target AP*/
[WTask]20009
```

However, if there are issues with the connection they may be identified as follows:

- if the state switches from `SM=1, Sub=0` to `SM=0, Sub=0` Smart Connection has failed, possibly because the user did not click the button that fires `start API: SartSmartConnection` in the Android app.

- if the state switches from `SM=2, Sub=0` to `SM=0, Sub=0` the scan operation failed. Likely reasons for this are:

  o the wireless AP with the stated SSID isn't available.

  o the wireless AP information received from Smart Connection is not correct.

  In the second case, you can use a network analyzer, such as OmniPeek Network Analyzer, to capture the wireless scan packet. Then check whether a valid SSID is contained in the probe request packet that is sent from MT7681 and whether the wireless AP responds with valid probe response packet.

  If there is no valid SSID in the probe request packet, recheck the SSID defined in the Smart Connection Android or iOS app. You can also print out the following variables to UART in `wifi_state_machine()` when MT7681 is in `WIFI_STATE_SCAN` mode, to check if the input parameters are correct:

```
void wifi_state_machine(void)
{
    ...
    if(pIoTMlme->CurrentWifiState == WIFI_STATE_SCAN)
    {
        printf_high("%d,%d,%d,%s,%s,%s", pIoTStaCfg->AuthMode,
            pIoTStaCfg->SsidLen, pIoTStaCfg->PassphaseLen,
            pIoTStaCfg->Ssid, pIoTStaCfg->Passphase,
            pIoTStaCfg->PMK);
    }
    ...
}
```

- if the state switches from `SM=3, Sub=0` to `SM=0, Sub=0`" authentication failed.

- If the state switches from `SM=4, Sub=0` to `SM=0, Sub=0` association failed.

  ior the two cases above, use a network analyzer to check the `Status Code` in the `Auth Response` or `Assoc Response` packet from the AP or `Reason Code` in the `Deauth` or `Deassoc` packet from MT7681. The meaning of the codes in these packets can be found in the 802.11 specifications, sections 8.3, "Format of individual frame types" and 8.4.1.9, "Status Code field".

- if the state switches from `SM=5, Sub=0` to `SM=0, Sub=0` the 4 way handshake failed. Check whether the password is correct.

- if the state switches from `SM=6, Sub=0` to `SM=0, Sub=0` DHCP failed and MT7681 cannot get an IP from the wireless AP. Use a network analyzer to capture the wireless packet and check whether there is a problem in the DHCP handshake (DHCP discover, DHCP offer, DHCP ACK, and/or DHCP Request).

- if no response packet is received, double check the antenna on MT7681 and on AP-router.

6.3.2. Using log to debug in AP mode

If MT7681 is in AP mode with PRINT_FLAG=TRUE the following is a typical debug log:

```
==> Recovery Mode   /*start to running recovery/Calibration image*/
<== Recovery Mode   /* end to running recovery/Calibration image*/
(-)                 /* start to running station image*/
load_ap_cfg         /* Load AP configuration from flash, if no valid
content,
                    use default value defined in source files*/
===> APStartUp
PMK Updating ...    /* if AP is WPA or WPA2-PSK mode, calculate PMK
according to
                    SSID and password */

/*print the final used AP configurations*/
AP SETTING: SSID[MT7681_AP1], AuthMode[9], WepStatus[8], Channel[8]

APStartUp ...  OK
[WTask]10140

/*A station which has MAC address "***.f4", send Auth Req packet to MT7681
AP*/
APPeerAuthReqAction: MAC-69:df:dd:29:62:f4
MacTableInsertEntry,280,  pEntryWcid=0
APHandleRxMgmtFrame,260, SUBTYPE_AUTH:/* MT7681 AP deal with the received
Auth
                                      Req packet , send back Auth
Response
APAssocReqActionAndSend : MAC-69:df:dd:29:62:f4  /* Station send Assoc
Request
                                      packet to MT7681 AP*/
APHandleRxMgmtFrame,273, SUBTYPE_ASSOC_REQ: /* MT7681 AP deal with the
received
                                      Assoc Req packet, send back
                                      Assoc Response */
WpaEAPOLStartAction [pEntry->Aid= 1] ===> : /* MT7681 AP send EAPOL start
                                      packet to station, to start
4
                                      way handshake*/
===> WPAStart4WayHS, pEntry->Aid=1
<=== WPAStart4WayHS: send Msg1 of 4-way
===> PeerPairMsg2Action [pEntry->Aid=1]
<=== PeerPairMsg2Action: send Msg3 of 4-way
===> PeerPairMsg4Action [pEntry->Aid=1]

/*4 way handshake finished*/
AP SETKEYS DONE - WPA2, AuthMode(7), WepStatus(6), GroupWepStatus(4)

Client IP: 192.168.81.2       /*MT7681 AP assign a IP address to station*/

/*MT7681 AP dump all DHCP client list*/
DhcpNO[0] Flag[1] IP[192.168.81.2]   MAC[69-df-dd-29-62-f4]
DhcpNO[1] Flag[0] IP[0.0.0.0]    MAC[0-0-0-0-0-0]
DhcpNO[2] Flag[0] IP[0.0.0.0]    MAC[0-0-0-0-0-0]
DhcpNO[3] Flag[0] IP[0.0.0.0]    MAC[0-0-0-0-0-0]
DhcpNO[4] Flag[0] IP[0.0.0.0]    MAC[0-0-0-0-0-0]
```

If MT7681 is set to open mode (no WEP/WPA in use), `PMK Update…` and the log related to 4 way handshake should not be listed.

You can check the AP setting from `AP SETTING: SSID[****]`, `AuthMode[*]`, `WepStatus[*]`, `Channel[*]`. Noting that valid values of:

- `AuthMode` are 0, 4, 7 and 9.

- `WepStatus` are 4, 6 and 8.

- `Channel` are in the range 1 to 14.

The meaning of `AuthMode` and `WepStatus` can be found in `iot_api.h` as follows:

```c
File : iot_api.h
typedef enum _NDIS_802_11_WEP_STATUS
{
    Ndis802_11WEPEnabled,
    Ndis802_11Encryption1Enabled = Ndis802_11WEPEnabled,
    Ndis802_11WEPDisabled,
    Ndis802_11EncryptionDisabled = Ndis802_11WEPDisabled,
    Ndis802_11WEPKeyAbsent,
    Ndis802_11Encryption1KeyAbsent = Ndis802_11WEPKeyAbsent,
    Ndis802_11WEPNotSupported,
    Ndis802_11EncryptionNotSupported = Ndis802_11WEPNotSupported,
    Ndis802_11Encryption2Enabled,    /* TKIP mode*/
    Ndis802_11Encryption2KeyAbsent,
    Ndis802_11Encryption3Enabled,    /* AES mode*/
    Ndis802_11Encryption3KeyAbsent,
    Ndis802_11Encryption4Enabled,    /* TKIP-AES mix mode*/
    Ndis802_11Encryption4KeyAbsent,
    Ndis802_11GroupWEP40Enabled,
    Ndis802_11GroupWEP104Enabled
} NDIS_802_11_WEP_STATUS, *PNDIS_802_11_WEP_STATUS,
NDIS_802_11_ENCRYPTION_STATUS, *PNDIS_802_11_ENCRYPTION_STATUS;

typedef enum _NDIS_802_11_AUTHENTICATION_MODE
{
    Ndis802_11AuthModeOpen,
    Ndis802_11AuthModeShared,
    Ndis802_11AuthModeAutoSwitch,
    Ndis802_11AuthModeWPA,
    Ndis802_11AuthModeWPAPSK,
    Ndis802_11AuthModeWPANone,         /*for ad-hoc*/
    Ndis802_11AuthModeWPA2,
    Ndis802_11AuthModeWPA2PSK,
    Ndis802_11AuthModeWPA1WPA2,
    Ndis802_11AuthModeWPA1PSKWPA2PSK,
    Ndis802_11AuthModeMax              // Not a real mode, defined as upper
bound
} NDIS_802_11_AUTHENTICATION_MODE, *PNDIS_802_11_AUTHENTICATION_MODE;
```

However, if there are issues with the connection they may be identified as follows:

- if a station fails to connect to MT7681 AP the log will stop before completing. Confirm that:
  - the password setting on the station side is correct.
  - the station antenna is connected and working.
  - there are not 3 stations already connected to the MT7681 AP. A fourth station trying to connect will be rejected by MT7681.

  If none of these are the cause, use a network analyzer tool to capture the wireless packets and check the codes they contain. Check the documentation with the network analyzer tool for details on the meaning of the codes.

6.4.   Unresponsive board

If your LinkIt Connect 7681 development board becomes unresponsive:

1) attempt to reboot the board using either AT#Reboot or the hardware reset button (see Figure 25).

2) reload the recovery firmware using the uploader.

3) reload the recovery firmware using a Flash Programmer (for more information on Flash Programmers see section 3.3, "Additional tools").

4) contact Seeed Studio support.

Remember also to check the Labs Forums for posts on possible causes and cures.

Appendix A Flash Layout

MT7681 firmware and configuration data are stored in external Flash. There are two types of Flash layout used:

- Flash layout with XIP region size of 60KB.

- Flash layout with XIP region size of 164KB.

It's acceptable to use external serial Flash of 512KB, but 1MB is recommended.

The FOTA FW region is optional. It's used to store an image download from the cloud or a server over wireless, if the device is using FOTA.

The offset for each region in the flash layout cannot be changed, if user data is to be stored in Flash, it must be added to the User Config region without changing the defined items.

The Flash layout setting is managed in the header file `src\include\flash_map` as follow:

```
#if (FLASH_XIP_164_SUPPORT == 1)
/****************************************************************
*
  *     Flash Mapping

****************************************************************/
#define FLASH_OFFSET_BASE               (0x0000)
#define FLASH_OFFSET_LOADER_START       (0x0000)            //20KB
#define FLASH_OFFSET_RESERVE_1_START    (0x5000)            //4KB

#define FLASH_OFFSET_UPG_FW_START       (0x6000)            //64KB
(RecoveryFW)
#define FLASH_OFFSET_RESERVE_2_START    (0x16000)           //4KB


.......

#define FLASH_OFFSET_WRITE_BUF_START    (0xB3000)           //4KB
#define FLASH_OFFSET_RESERVE_8_START    (0xB4000)           //4KB

#define FLASH_OFFSET_OTA_START          (0xB5000)           //292KB
#define FLASH_OFFSET_RESERVE_9_START    (0xFE000)           //4KB
#define FLASH_OFFSET_END                (0xFF000)           //0KB

#else
/****************************************************************
*
  *     Flash Mapping

****************************************************************/
#define FLASH_OFFSET_BASE               (0x0000)
#define FLASH_OFFSET_LOADER_START       (0x0000)            //20KB
#define FLASH_OFFSET_RESERVE_1_START    (0x5000)            //4KB

#define FLASH_OFFSET_UPG_FW_START       (0x6000)            //64KB
(RecoveryFW)
#define FLASH_OFFSET_RESERVE_2_START    (0x16000)           //4KB
.......
#define FLASH_OFFSET_AP_FW_START        (0x4F000)           //64KB
#define FLASH_OFFSET_RESERVE_6_START    (0x5F000)           //4KB
#define FLASH_OFFSET_AP_XIP_FW_START    (0x60000)           //120KB
(AP:XIP+OVL)
#define FLASH_OFFSET_RESERVE_7_START    (0x7E000)           //4KB

#define FLASH_OFFSET_WRITE_BUF_START    (0x7F000)           //4KB
#define FLASH_OFFSET_RESERVE_8_START    (0x80000)           //0KB
#define FLASH_OFFSET_END                (0x80000)           //0KB

#endif /* FLASH_XIP_164_SUPPORT */
```

Setting the specific Flash layout for XIP region (60KB or 164KB) is done in the Makefile, as follows:

```
# After changing FLASH_XIP_164,
# It is must compile all Firmwares (Recovery, STA, AP)
# Then upgrade loader, recovery, ap, sta firmwares in order by uart,
# or upgrade MT7681_all.bin / MT7681_all_xip164.bin by Flash writer
FLASH_XIP_164 = 0

ifeq ($(FLASH_XIP_164),1)
XIPSIZE        = _xip164
endif


ifeq ($(FLASH_XIP_164),1)
FUNCFLAGS += -DFLASH_XIP_164_SUPPORT=1
endif
```

The Flash layout with XIP region size set to 60KB is shown in Table 14.

	Offset (Hex)	Section	Size (KB)	Size (Byte) Hex	Notes
1	0x0000	Loader	20	0x5000	Loader image `MT7681_loader.bin`
	0x5000	reserved 1	4	0x1000	
2	0x6000	Recovery Mode FW	64	0x10000	Recovery mode image `MT7681_recovery_old.bin`
	0x16000	reserved 2	4	0x1000	
3	0x17000	EEPROM	4	0x1000	Factory Calibration image `MT7681E2_EEPROM layout_20140330.bin`
	0x18000	Common Config	4	0x1000	
	0x19000	Station Mode Config	4	0x1000	
	0x1A000	AP Mode Config	4	0x1000	
	0x1B000	User Config	4	0x1000	
	0x1C000	reserved 3	12	0x3000	
4	0x1F000	STA Mode RAM FW	64	0x10000	Station mode image `MT7681_sta.bin`
	0x2F000	reserved 4	4	0x1000	
	0x30000	STA Mode-XIP FW	60	0xF000	
	0x3F000	STA Mode-OVL FW	60	0xF000	
	0x4E000	reserved 5	4	0x1000	
6	0x4F000	AP Mode FW	64	0x10000	AP mode image `MT7681_ap.bin`
	0x5F000	reserved 6	4	0x1000	
	0x60000	AP Mode-XIP FW	60	0xF000	
	0x6F000	AP Mode-OVL FW	60	0xF000	
	0x7E000	reserved 7	4	0x1000	
	0x7F000	Flash Write Buffer	4	0x1000	
	0x80000	reserved 8	4	0x1000	
	0x81000	FOTA FW	188	0x2F000	
	0xB0000	reserved 9	4	0x1000	

Table 14 MT7681 Flash Layout (XIP=60KB)

The Flash layout with XIP region size set to 164KB is shown in Table 15.

	Offset	Section	Size (KB)	HEX (Byte)	Notes
1	0x0000	Loader	20	0x5000	Loader image `MT7681_loader_xip164.bin`
	0x5000	reserved 1	4	0x1000	
2	0x6000	Recovery Mode FW	64	0x10000	Recovery mode image `MT7681_recovery_old.bin`
	0x16000	reserved 2	4	0x1000	
3	0x17000	EEPROM	4	0x1000	Factory Calibration image `MT7681E2_EEPROM layout_20140330.bin`
	0x18000	Common Config	4	0x1000	
	0x19000	Station Mode Config	4	0x1000	
	0x1A000	AP Mode Config	4	0x1000	
	0x1B000	User Config	4	0x1000	
	0x1C000	reserved 3	12	0x3000	
4	0x1F000	STA Mode RAM FW	64	0x10000	Station mode image `MT7681_sta.bin`
	0x2F000	reserved 4	4	0x1000	
	0x30000	STA Mode-XIP FW	164	0x29000	
	0x59000	STA Mode-OVL FW	60	0xF000	
	0x68000	reserved 5	4	0x1000	
6	0x69000	AP Mode FW	64	0x10000	AP mode image `MT7681_ap.bin`
	0x79000	reserved 6	4	0x1000	
	0x7A000	AP Mode-XIP FW	164	0x29000	
	0xA3000	AP Mode-OVL FW	60	0xF000	
	0xB2000	reserved 7	4	0x1000	
	0xB3000	Flash Write Buffer	4	0x1000	
	0xB4000	reserved 8	4	0x1000	
	0xB5000	FOTA FW	292	0x49000	
	0xFE000	reserved 9	4	0x1000	

Table 15 MT7681 Flash Layout (XIP Region=164KB)

The layout for the Common Config region in Flash is shown in Table 16. These items are:

- loaded when MT7681 calls `load_com_cfg()`.

- reset by calling `reset_com_cfg()`.

Offset	Section	Size (Byte)	DEC Offset
0x18000	Common Info Stored Flag	1	0
0x18001	Boot Firmware Index: (0=station mode 1=AP mode 2=Recovery mode)	1	1
0x18002	Firmware Update Status	1	2
0x18003	I/O Mode select	1	3
0x18004	Reserved 1	20	4
0x18018	UART Baud rate	4	24
0x1801C	UART Data bits	1	28
0x1801D	UART Parity bits	1	29
0x1801E	UART Stop bits	1	30
0x1801F	Reserved 2	20	31
0x18033	TCP/UDP, Server/Client Select (Bitmap)	1	51
0x18034	TCP Server Port (2Bytes)	2	52
0x18036	TCP Client Port (2Bytes)	2	54
0x18038	UDP Server Port (2Bytes)	2	56
0x1803A	UDP Client Port (2Bytes)	2	58
0x1803C	IP Type select (0:Static / 1: Dynamic)	1	60
0x1803D	Static IP	4	61
0x18041	Subnet Mask (4 Bytes)	4	65
0x18045	DNS Server IP (4 Bytes)	4	69
0x18049	Gateway IP (4 Bytes)	4	73
0x1804D	IoT Server IP (4 Bytes)	4	77
0x18051	IoT Sever Domain Name (128 Bytes)	128	81
0x180D1	Reserved 3	20	209
0x180E5	Cmd_Password (4 Byte)	4	229
0x180E9	Reserved 4	x	233

Table 16 Common Config (0x11000)

The Station Mode Config region in Flash is shown in Table 17. These items are:

- loaded when MT7681 calls `load_sta_cfg()`.

- reset by calling `reset_sta_cfg()`.

- stored from RAM by calling `store_sta_cfg()`.

Offset	Section	Size (Byte)	DEC Offset
0x19000	Station Info Stored Flag	1	0
0x19001	BSSID	6	1
0x19007	SSID	32	7
0x19027	SSID Len	1	39
0x19028	AP Password	32	40
0x19048	AP Password Len	1	72
0x19049	AuthMode	1	73
0x1904A	4-Way PMK	32	74
0x1906A	Reserved 1	x	106

Table 17 Station Mode Config/Setting

The layout for the AP Mode Config region in Flash is shown in Table 18. These items are:

- loaded when MT7861 calls `load_ap_cfg()`.
- reset by calling `reset_ap_cfg()`.
- stored from RAM by calling `store_ap_cfg()`.

Offset	Section	Size (Byte)	DEC Offset
0x1A000	AP Info Stored Flag	1	0
0x1A001	BSSID (Not used, use MAC in EEPROM region)	6	1
0x1A007	SSID	32	7
0x1A027	SSID Len	1	39
0x1A028	Physical Mode	1	40
0x1A029	AP Channel	1	41
0x1A02A	AuthMode	1	42
0x1A02B	AP Password	32	43
0x1A04B	AP Password Len	1	75
0x1A04C	Beacon Interval	2	76
0x1A04E	DTIM Count	1	78
0x1A04F	DTIM Interval	1	79
0x1A050	Capability Info	2	80
0x1A052	fgIsHidden_ssid	1	82
0x1A053	Reserved 1	x	83

Table 18 AP Mode Config/Setting

The layout for the User Config region in Flash is shown in Table 19. These items are:

- loaded when MT7681 calls `load_usr_cfg()`.
- reset by calling `reset_usr_cfg()`.

Offset	Section	Size (Byte)	DEC Offset
0x1B000	Product Info Stored Flag	1	0
0x1B001	Vendor Name	32	1
0x1B021	Product Type	32	33
0x1B041	Product Name	32	65
0x1B061	Transport Frame Size	2	97
0x1B063	Transport Frame Timeout	4	99
0x1B067	Reserved 1	x	103

Table 19 User Config/Setting