MAVID-3M SDK

Getting Started Guide

Revision: 1.5

Libre Wireless Technologies Private Limited

librewireless.com

Circuit diagrams and other information relating to Libre Wireless Technologies products are included as a means of illustrating typical applications. Consequently, complete information sufficient for construction purposes is not necessarily given. Although the information has been checked and is believed to be accurate, no responsibility is assumed for inaccuracies. Libre Wireless Technologies reserves the right to make changes to specifications and product descriptions at any time without notice. Contact your local Libre Wireless Technologies sales office to obtain the latest specifications before placing your product order. The provision of this information does not convey to the purchaser of the described semiconductor devices any licenses under any patent rights or other intellectual property rights of Libre Wireless Technologies or others. All sales are expressly conditional on your agreement to the terms and conditions of the most recently dated version of Libre Wireless Technologies standard Terms of Sale Agreement dated before the date of your order (the "Terms of Sale Agreement"). The product may contain design defects or errors known as anomalies which may cause the product's functions to deviate from published specifications. Anomaly sheets are available upon request. Libre Wireless Technologies products are not designed, intended, authorized or warranted for use in any life support or other application where product failure could cause or contribute to personal injury or severe property damage. Any and all such uses without prior written approval of an Officer of Libre Wireless Technologies and further testing and/or modification will be fully at the risk of the customer. Copies of this document or other Libre Wireless Technologies literature, as well as the Terms of Sale Agreement, may be obtained by visiting Libre Wireless Technologies website.

LIBRE WIRELESS TECHNOLOGIES DISCLAIMS AND EXCLUDES ANY AND ALL WARRANTIES, INCLUDING WITHOUT LIMITATION ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND AGAINST INFRINGEMENT AND THE LIKE, AND ANY AND ALL WARRANTIES ARISING FROM ANY COURSE OF DEALING OR USAGE OF TRADE. IN NO EVENT SHALL LIBRE WIRELESS TECHNOLOGIES BE LIABLE FOR ANY DIRECT, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES; OR FOR LOST DATA, PROFITS, SAVINGS OR REVENUES OF ANY KIND; REGARDLESS OF THE FORM OF ACTION, WHETHER BASED ON CONTRACT; TORT; NEGLIGENCE OF LIBRE WIRELESS TECHNOLOGIES OR OTHERS; STRICT LIABILITY; BREACH OF WARRANTY; OR OTHERWISE; WHETHER OR NOT ANY REMEDY OF BUYER IS HELD TO HAVE FAILED OF ITS ESSENTIAL PURPOSE, AND WHETHER OR NOT LIBRE WIRELESS TECHNOLOGIES HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES

Table of Contents

Table of Figures

1. Document Information

1.1. Abstract

Libre MAVID-3M is low cost, low power module targeted primarily for IoT applications. In-built Voice front end and Alexa Voice Service integrated platform extends its capacity to design Voice AI solutions. This document explains how to set up the software development environment for the hardware based on MAVID-3M EVK.

1.2. Document Convention

Icon	Meaning	Description
	Note	Provides information good to know
	Caution	Indicates situation that might result in loss of data or hardware damage

1.3. Document Revision History

Revision	Date	Description of change	Author
1.5	Nov 12, 2021	Final Draft	Sachin and Simbu
1.4	July 18, 2021	Final Draft	Jay, Sachin and Manoj
1.3	May 26, 2021	Final Draft	Jay, Sachin and Manoj
1.2	May 25, 2021	Final Draft	Jay, Sachin and Manoj
1.1	May 6, 2021	Final Draft	Jay and Sachin
1.0	May 3, 2021	Final Draft	Jay and Sachin
0.3	May 1, 2021	Initial Draft	Jay and Sachin
0.2	April 29, 2021	Initial Draft	Jay and Sachin
0.1	April 20, 2021	Initial Draft	Jay and Sachin

2. Get Started

This chapter provides the complete installation procedures followed to set up the MAVID-3M EVK and develop custom applications.

2.1. Install Pre-requisites

This section provides detailed guideline on how to set up the SDK build environment with default GCC on Linux OS and on Microsoft Windows using MinGW cross-compilation tool.

The default GCC compiler provided in the SDK is based on the 32-bit architecture, Windows/Linux with 64-bit or 32-bit operating system is acceptable.

To Download the MAVID-3M_SDK follow section2.2.

To Install the build environment, follow section 2.3.

2.2. Get MAVID-3M SDK

To build applications for the MAVID-3M_EVK, Libre provides software libraries as part of software development package.

The SDK package consists of source code for custom application development, supporting tools, documents and mobile application for device configuration.

The MAVID-3M_SDK can be obtained from the Libre GitHub. The MAVID-3M_SDK repository is private and can only be accessed by authorized users. To get access to MAVID-3M_SDK repository, generate the SSH keys and share the public key to SDKsupport@librewireless.com. For more information on SSH Key generation refer to SSH Key Generation Guide.

Once the access is permitted, the SDK can be cloned from the GitHub using the following command:

```
git clone git@github.com:LibreWireless/MAVID-3_SDK.git
```

2.3. Set up the Tools

2.3.1. Installing the SDK Build Environment on Linux

Default GCC compiler provided in the SDK is required to setup the build environment on Linux OS.

Before building the project, verify that the required toolchain is installed for the build environment, as shown in the table:

Item	Description
OS	Linux OS
Make	GNU make 3.81
Compiler	Linaro GCC Toolchain for ARM Embedded Processors 4.8.4

The following command downloads and installs the basic building tools in Ubuntu:

```
sudo apt-get install build-essential
```

A compilation error occurs when building the IoT SDK with the default GCC cross compiler on a 64-bit system without installing the package to support the 32-bit executable binary, as shown below:

/bin/sh: 4: tools/gcc/gcc-arm-none-eabi/bin/arm-none-eabi-gcc: not found

The commands to install the basic build tools and the package for supporting 32-bit binary executable in the Ubuntu are shown below:

```
sudo dpkg --add-architecture i386

sudo apt-get update

sudo apt-get install libc6-i386
```

The default installation path of the GCC compiler is <sdk_root>/tools/gcc, and the compiler settings are in the <sdk_root>/.config configuration file.

Setup the BINPATH in the .config file as given below:

```
BINPATH = $(SOURCE_DIR)/tools/gcc/gcc-arm-none-eabi/bin
```

2.3.2. Installing the SDK Build Environment on Microsoft Windows

To build the project on Windows OS, install MinGW cross-compiler and integrate ARM GCC toolchain for Windows with the MAVID-3_SDK.

Preparing the cross-compiler tool:

1. Download mingw-get-setup.exe from here.
2. Launch the installer, and click **Install:**

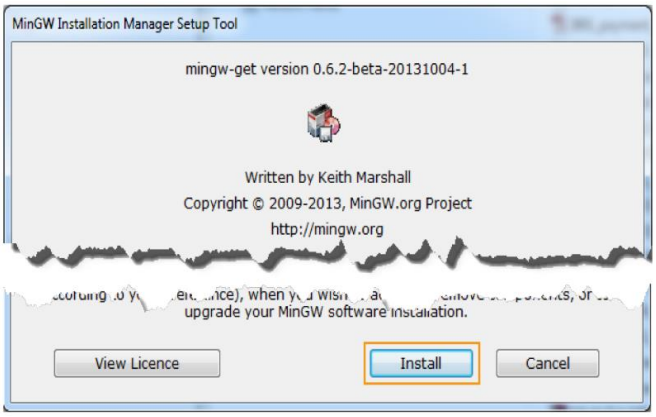


Figure 2.3.2-1: MinGW Installation Manager Setup Tool

3. Follow the on-screen instructions and keep the default settings, then click **Continue** to download the tool to C:\MinGW installation directory.

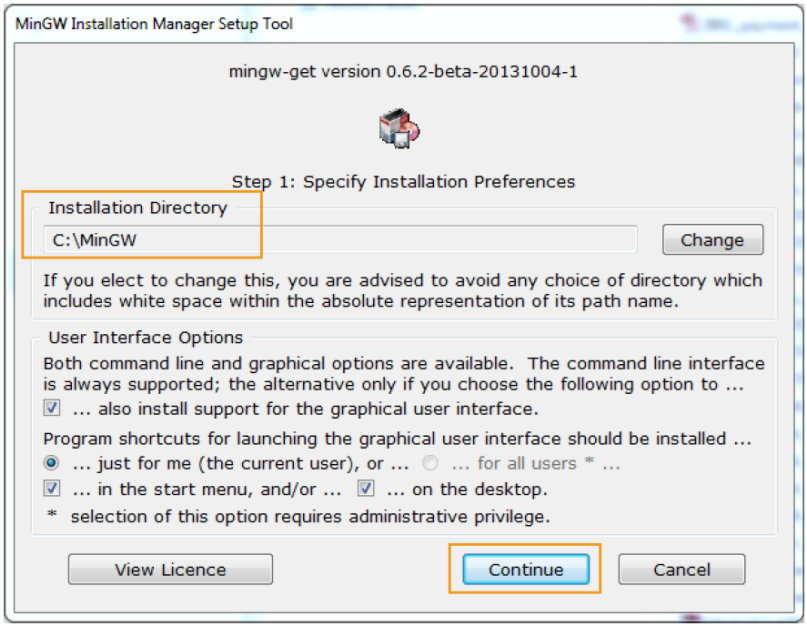


Figure 2.3.2-2: Keep default installation preferences

4. Click **Continue** on the MinGW Installation Manager Setup Tool after the download is complete:

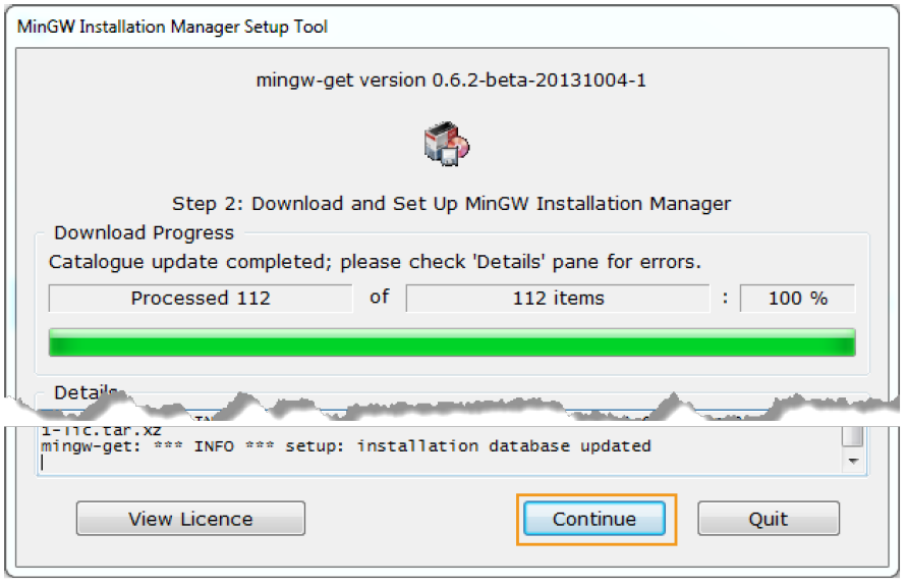


Figure 2.3.2-3: Download and set up MinGW Installation Manager

5. Select msys-base and mingw32-base from Basic Setup package list, and right click to bring up the menu options. Click **Mark for Installation** from the menu:

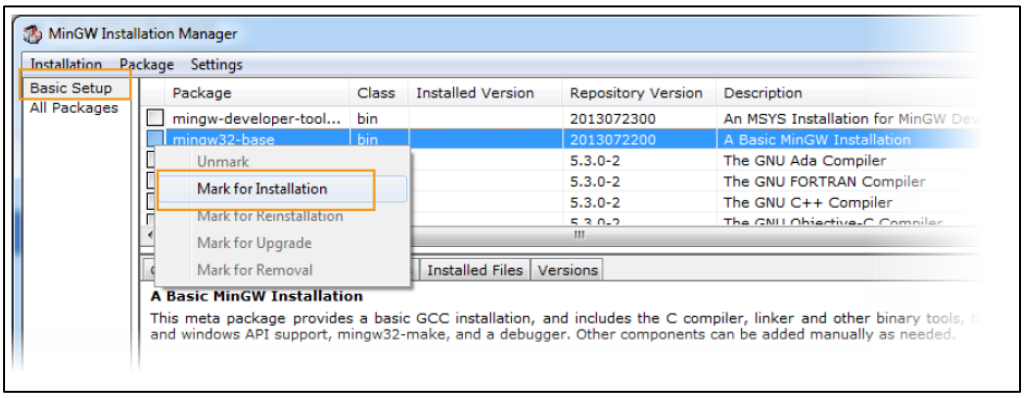


Figure 2.3.2-4: Basic setup on MinGW Installation Manager

6. Click **Apply Changes** from the Installation menu:

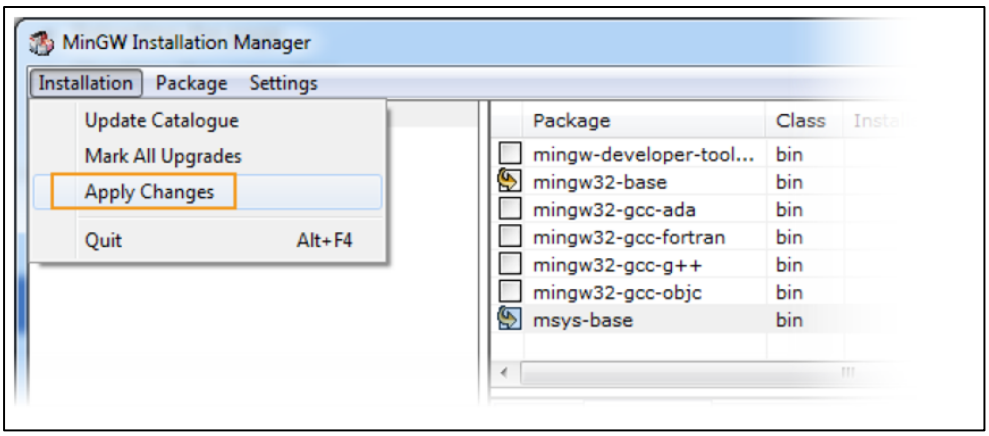


Figure 2.3.2-5: A basic MinGW installation

7. Click **Apply** on the pop-up dialog window:

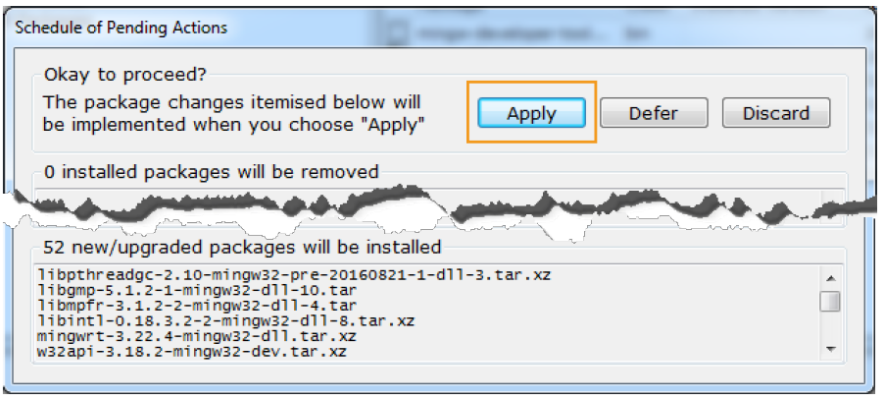


Figure 2.3.2-6: Schedule of pending actions

8. Click **Close** to close the dialog window once the operation is complete:

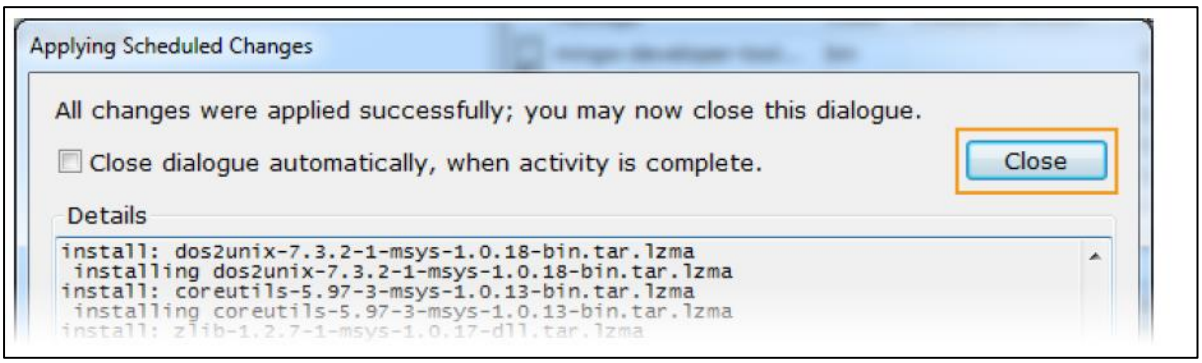


Figure 2.3.2-7: Applying scheduled changes

9. Navigate to C:/MinGW/msys/1.0 folder and launch the MinGW terminal by running msys.bat to create home/<user_name> folder.

10. Copy the SDK to MinGW home/<user_name> folder, as shown:

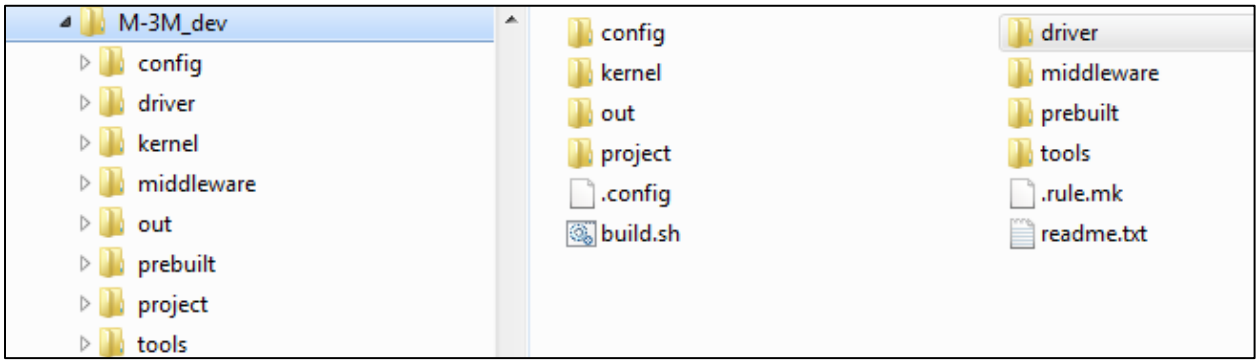


Figure 2.3.2-8: SDK folder structure

11. Download ARM-GCC-win32 from here.

a) Create a new folder named win under <sdk_root>/tools/gcc/.

b) Unzip the content of gcc-arm-none-eabi-4_8-2014q3-20140805-win32.zip to <sdk_root>/tools/gcc/win/ folder.

c) Rename the unzipped gcc-arm-none-eabi-4_8-2014q3-20140805-win32 folder to gcc-arm-none-eabi, as shown in figure:

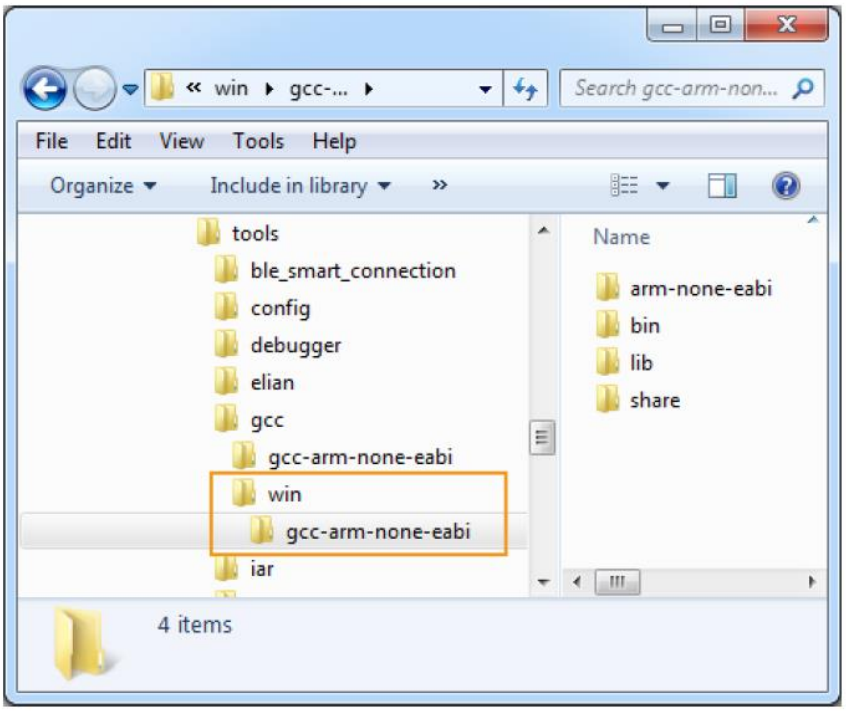


Figure 2.3.2-9: tools/gcc folder structure

2.3.3. Troubleshooting

Note the following caveats when building the project with MinGW on Windows OS.

- The folder name and file name in the IoT SDK should not contain ' ', '[' or ']' characters.
- The project name should be less than 30 characters. Otherwise, a build error similar to the one shown below may occur due to the long path.

arm-none-eabi-gcc.exe: error: ../../../../../out/mt2523_hdk/i2c_communication_with_EEPROM_dma/obj/project/mt2523_hdk/hal_examples/i2c_communication_with_EEPROM_dma/src/system_mt2523.o: No such file or directory

- The makefile in your project should not use any platform dependent commands or files, such as stat or /proc/cpuinfo.
- MinGW installation directory should be C:\MinGW. Otherwise, build errors may occur if MinGW installation path is very deep.
- To build an httpd project, export MinGW/bin path for gcc command, then install msys-vim package for xxd command by launching the MinGW Installation Manager (mingw-get-setup.exe) and choosing the reinstall, as shown in figure below:

export PATH=$PATH:/c/MinGW/bin:

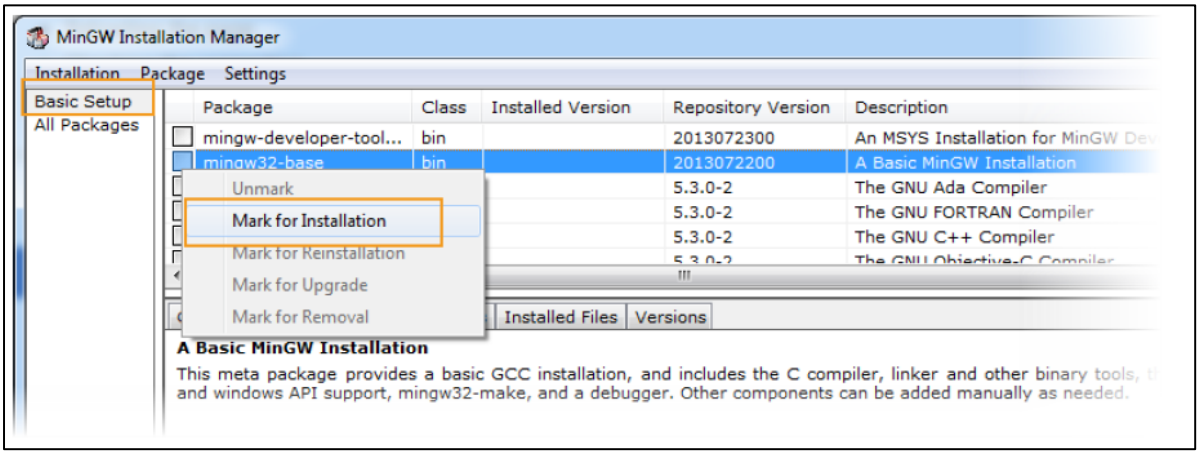


Figure 2.3.3-1: MinGW Installation Manager

- By default, the parallel build feature is enabled in build.sh to speed up the compilation. Disable the parallel build feature, if any unreasonable build error or system exception occurs.
    - To disable the parallel build feature for all modules, change the value "-j" of EXTRA_VAR to "-j1" in build.sh, as shown below:

```
platform=$(uname)
if [[ "$platform" =~ "MINGW"]]; then
export EXTRA_VAR=-j1
else
export EXTRA_VAR=-j`cat /proc/cpuinfo |grep ^processor|wc -l`
fi...
```

    - If build errors are due to a particular module's parallel build, set the value <module>_EXTRA as "-j1" in .rule.mk to disable the parallel build for that particular module, as shown below:

```
OS_VERSION:= $(shell uname)
ifneq ($(filter MINGW%,$(OS_VERSION)),)
$(DRV_CHIP_PATH)_EXTRA := -j1
$(MID_MBEDTLS_PATH)_EXTRA := -j1
$(<module>)_EXTRA := -j1
endif...
```

2.4. Setting up Serial Connection

Define device manager rules using Udev rules.

The mtk-usb.rules file can be found on <sdk_root>/Flash_tool_files directory, copy that file and paste it to /etc/udev/rules.d/ directory.

```
cp -f mtk-usb.rules /etc/udev/rules.d/mtk-usb.rules
```

Sign back in or reboot the computer to enable the Udev rules.

Trouble shooting:

In case the device is not detected please try the following.

- Configure the serial port permissions for a given user.

  ls -l /dev/ttyUSBx

  /dev/ttyUSBx, owner: root, group: dialout

- Add the account <example_user> to the group.

  sudo usermod -a -G dialout <example_user>

2.5. Building Project with MAVID-3M SDK

Build the project using the script at <sdk_root>/build.sh. To find out more about the usage of the script, navigate to the SDK's root directory and execute the following command:

```
cd <sdk_root>

./build.sh
```

Build Project

Usage: ./build.sh <board> <project_name> <argument>
Example:
./build.sh aw7698_evk <project_name>
./build.sh clean (clean folder: out)
./build.sh aw7698_evk clean (clean folder: out/aw7698_evk)
./build.sh aw7698_evk <project_name> clean (clean folder: out/aw7698_evk / <project_name>)

Argument:
-f=<feature makefile> or --feature=<feature makefile>

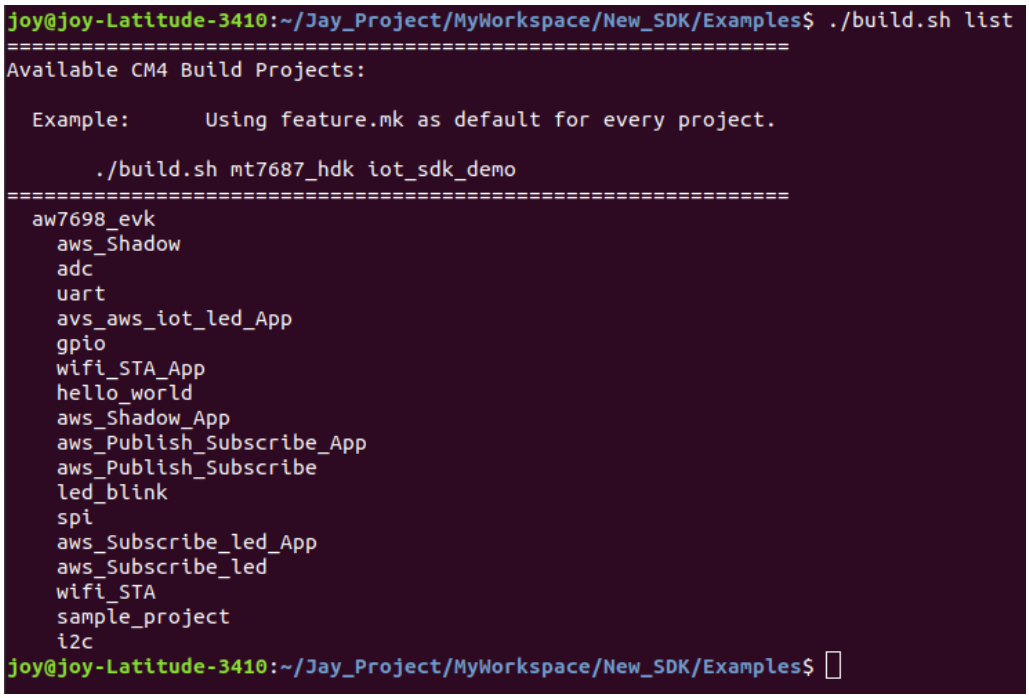
Replace feature.mk with another makefile. For example,
the feature_example.mk is under project folder, -f=feature_example.mk
will replace feature.mk with feature_example.mk.
-o=<make option> or --option=<make option>
Assign additional make options. For example,
to compile module sequentially, use -o=-j1;
to turn on specific feature in feature makefile, use -o=<feature_name>=y;
to assign more than one options, use -o=<option_1> -o=<option_2>.

2.5.1. List all Available Boards and Projects

Run the command to show all available boards and projects:

```
./build.sh list
```

Output:



2.5.2. Build the Project

To build a specific project, run the following command:

```
./build.sh aw7698_evk <project_name>

./build.sh aw7698_evk hello_world -f=feature.mk (for customized feature file)
```

The output files will be placed under <sdk_root>/out/<board>/<project>/

For example, to build a project hello_world, run the following build command:

```
./build.sh aw7698_evk hello_world

./build.sh aw7698_evk hello_world -f=feature.mk (for customize feature file)
```

The output files will be placed under <sdk_root>/out/aw7698_evk/ hello_world /

Output:

```
joy@joy-Latitude-3410:~/Jay_Project/MyWorkspace/New_SDK/Examples$ ./build.sh aw7698_evk hello_world
UE BUILD BOARD: aw7698_evk
UE BUILD PROJECT: hello_world
platform=Linux
FEATURE = feature.mk
make: Entering directory '/home/joy/Jay_Project/MyWorkspace/New_SDK/Examples/project/aw7698_evk/apps/hello_world/GCC'
trigger by build.sh, skip cleanlog
Build... verno.o
Build... verno.o PASS
Linking... ../../../../../libraries/LibreSDK.a ../../../../../libraries/libbc_smtcn_protected.a ../../../../../libraries/libhal_core_CM4_GCC.a
../../../../../libraries/libhal_protected_CM4_GCC.a ../../../../../libraries/libminicli.a ../../../../../libraries/libwifi.a ../../../../../lib
raries/libwifi_aw7698_ram.a ../../../../../libraries/libble.a ../../../../../libraries/libbtdriver_7698.a ../../../../../libraries/libmp3dec.a
../../../../../libraries/libheaac_decoder.a ../../../../../libraries/libflash_manager_CM4_GCC.a
Done
   text    data     bss     dec     hex filename
 878524    3980 2727456 3609960  371568 /home/joy/Jay_Project/MyWorkspace/New_SDK/Examples/out/aw7698_evk/hello_world/hello_world.elf
Generate Assembly from elf:
copy_firmware.sh....
BOARD=aw7698_evk
cp /home/joy/Jay_Project/MyWorkspace/New_SDK/Examples/project/aw7698_evk/apps/hello_world/GCC/aw7698_hdk.cmm to /home/joy/Jay_Project/MyWorkspa
ce/New_SDK/Examples/out/aw7698_evk/hello_world/
make: Leaving directory '/home/joy/Jay_Project/MyWorkspace/New_SDK/Examples/project/aw7698_evk/apps/hello_world/GCC'
============================================================
./build.sh aw7698_evk hello_world
Start Build: Thu Apr 15 17:15:50 IST 2021
End Build: Thu Apr 15 17:16:38 IST 2021
TOTAL BUILD: PASS
joy@joy-Latitude-3410:~/Jay_Project/MyWorkspace/New_SDK/Examples$
```

2.5.3. Cleaning the Project

Usage: 1. ./build.sh <board> <project> clean

2. ./build.sh <board> <project> [clean | -f=feature.mk]

Example:

```
./build.sh aw7698_evk hello_world clean

./build.sh aw7698_evk hello_world clean -f=feature.mk
```

For more details on feature file follow section 2.5.4.

2.5.4. Feature File

Find the feature.mk file from
<sdk_root>/project/aw7698_evk/apps/<Example_name>/GCC/

Make changes in feature.mk file according to requirements. Also, create multiple feature files and provide the name as a feature_<name>.mk, after creating feature file follow the procedure to build it as shown in section 2.5.

Example feature.mk file:

project > aw7698_evk > apps > led_blink > GCC > feature.mk

```
IC_CONFIG                               = aw7698
BOARD_CONFIG                            = aw7698_evk
# debug level: none, error, warning, and info
MTK_DEBUG_LEVEL                       = info
# System service debug feature for internal use
MTK_SUPPORT_HEAP_DEBUG                = n
MTK_HEAP_SIZE_GUARD_ENABLE            = n
MTK_OS_CPU_UTILIZATION_ENABLE         = y
#SWLA
MTK_SWLA_ENABLE                       = n

#NVDM
MTK_NVDM_ENABLE                       = y

#WIFI features
#If this feature is off, than other Wi-Fi and LWIP feature options won't take effect
MTK_WIFI_ENABLE                       = y
MTK_WIFI_TGN_VERIFY_ENABLE            = n
MTK_WIFI_WPS_ENABLE                   = n
MTK_WIFI_DIRECT_ENABLE                = n
MTK_WIFI_REPEATER_ENABLE              = n
MTK_WIFI_PROFILE_ENABLE               = y
MTK_SMTCN_V5_ENABLE                   = n
MTK_CM4_WIFI_TASK_ENABLE              = y
MTK_WIFI_ROM_ENABLE                   = y
MTK_WIFI_CLI_ENABLE                   = y
#Wi-Fi&BT coex
MTK_BWCS_ENABLE                       = y

#Bluetooth feature
MTK_BT_ENABLE                         = y
MTK_BLE_ONLY_ENABLE                   = y
```

2.6. Flashing the Firmware

Tool used for updating the firmware: CODA flash tool

CODA package is provided in <sdk_root>/Flash_tool_files directory.

2.6.1. Connect your Device

Now connect MAVID-3M_EVK board to the computer and check under what serial port the board is visible as shown below:

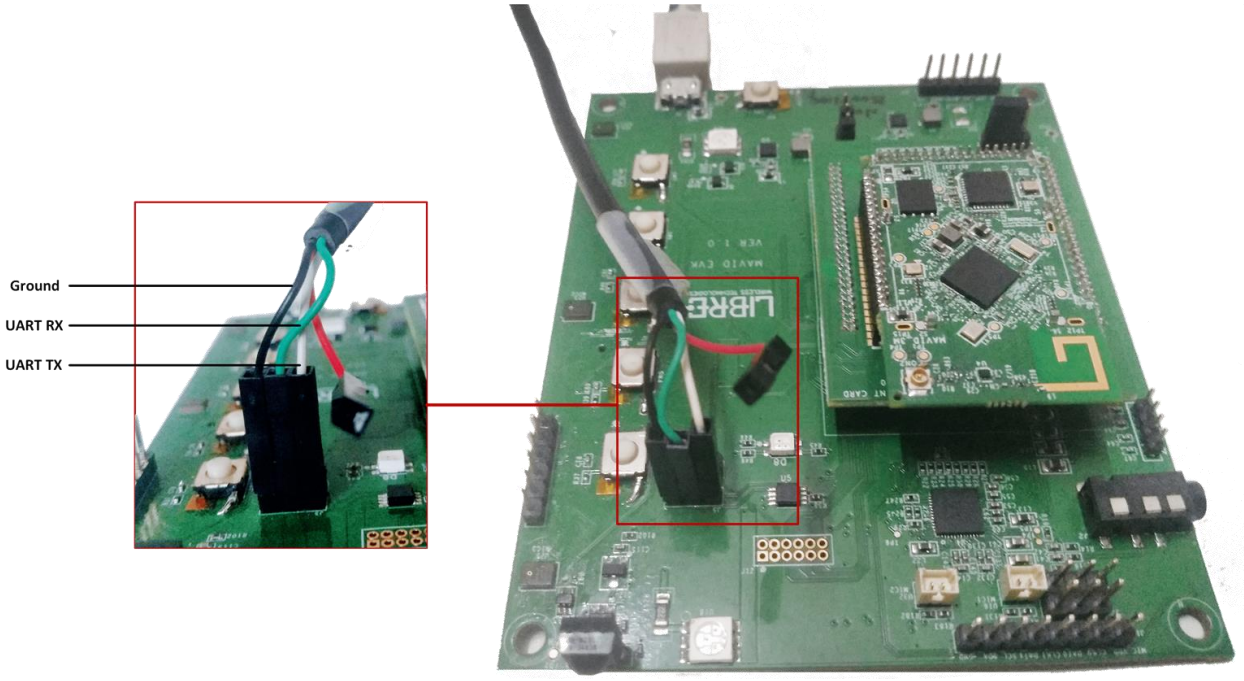


Figure 2.6.1-1: MAVID-3M EVK Setup

Serial ports have the following patterns in their names:

- **Linux/Windows**: Starting with */dev/tty**

Apply the command 'ls /dev/tty*' and find the port as a ttyUSBx.

> **Note:** Keep the port name ready as it is required in the next steps.

2.6.2. Load the Firmware using Linux

CODA package for Linux contains below displayed files:

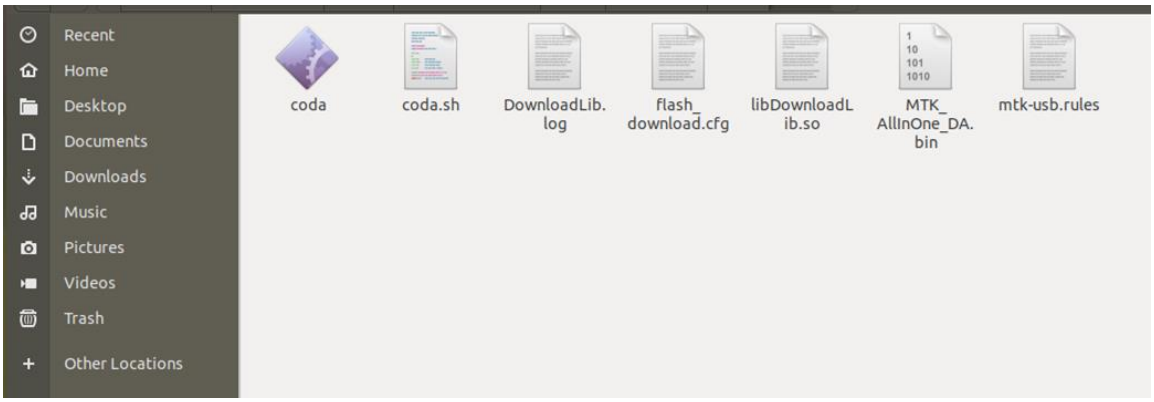


Figure 2.6.2-1: IoT_Falsh_Tool linux

Follow the below steps to flash the firmware:

a) H/W connection

1. Power Cable
2. TTL cable to the device for logs and flash

b) Find the CODA package files at given path:

<sdk_root>/Flash_tool_files/

1. coda
2. coda.sh
3. DownloadLib.log
4. libDownloadLib.so
5. MTK_AllInOne_DA.bin

These files will be copied at <sdk_root>/out/<board>/<project>/ directory during build time as the make file contains copy command. If not copied during build, copy the given files from <sdk_root>/Flash_tool_files/ directory to <sdk_root>/out/<board>/<project>/ directory.

c) Open the command prompt and switch to <sdk_root>/out/<board>/<project>/

Run the below command:

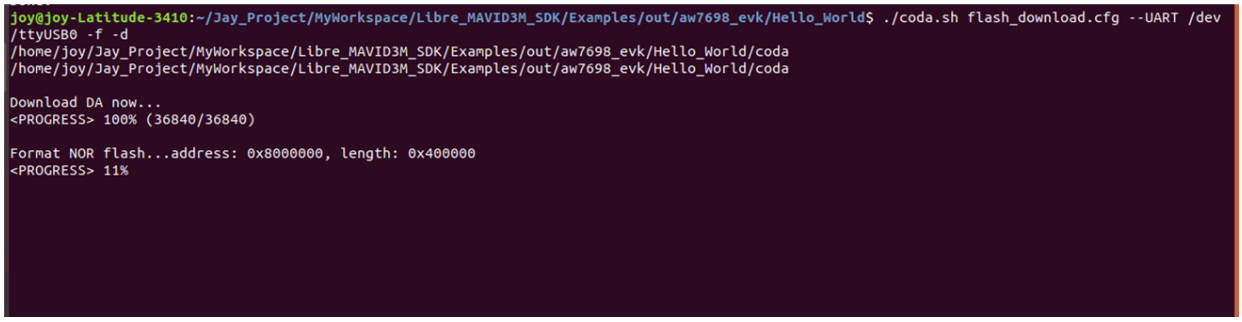
```
$ ./coda.sh flash_download.cfg --UART /dev/ttyUSBx -f -d
```

- Find appropriate USB port with command $ ls/dev/tty*
- -f option will erase the contents of flash and saved ENV. Do not use it unless required.

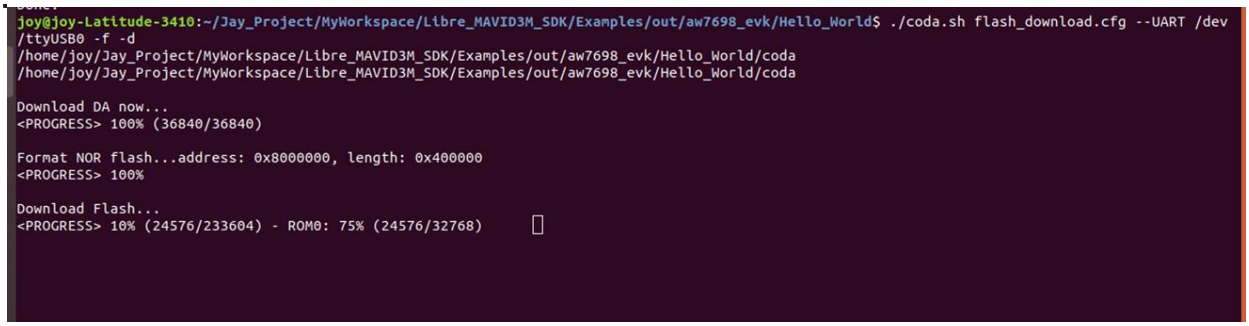
d) Reset the device. It will start progressing.

Normal Operation

During flashing, the below output log will be displayed:



If there are no issues by the end of the flash process, the board will reboot and start up the "hello_world" application:



2.6.3. Load the Firmware using Windows

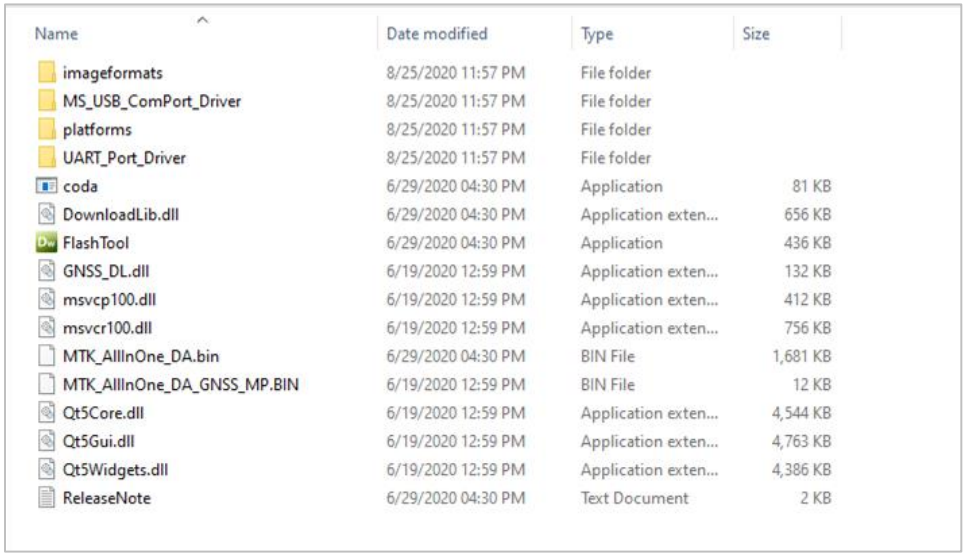
2.6.3.1. Installing IoT Flash Tool

Coda package is part of MAVID-3M_SDK release SDK and can be found under

<sdk_root>\tools\FOTA_Packaging_Tool\win

To install the IoT Flash Tool, copy the package folder to the Windows computer. No further steps are required.

CODA package for windows will contain below given files:



Name	Date modified	Type	Size
imageformats	8/25/2020 11:57 PM	File folder	
MS_USB_ComPort_Driver	8/25/2020 11:57 PM	File folder	
platforms	8/25/2020 11:57 PM	File folder	
UART_Port_Driver	8/25/2020 11:57 PM	File folder	
coda	6/29/2020 04:30 PM	Application	81 KB
DownloadLib.dll	6/29/2020 04:30 PM	Application exten...	656 KB
FlashTool	6/29/2020 04:30 PM	Application	436 KB
GNSS_DL.dll	6/19/2020 12:59 PM	Application exten...	132 KB
msvcp100.dll	6/19/2020 12:59 PM	Application exten...	412 KB
msvcr100.dll	6/19/2020 12:59 PM	Application exten...	756 KB
MTK_AllInOne_DA.bin	6/29/2020 04:30 PM	BIN File	1,681 KB
MTK_AllInOne_DA_GNSS_MP.BIN	6/19/2020 12:59 PM	BIN File	12 KB
Qt5Core.dll	6/19/2020 12:59 PM	Application exten...	4,544 KB
Qt5Gui.dll	6/19/2020 12:59 PM	Application exten...	4,763 KB
Qt5Widgets.dll	6/19/2020 12:59 PM	Application exten...	4,386 KB
ReleaseNote	6/29/2020 04:30 PM	Text Document	2 KB

Figure 2.6.3-1: IoT_Flash_Tool contain for windows

Using the IoT Flash Tool

The IoT Flash Tool is used to download, format and readback images on the flash memory of the target device.

2.6.3.2. Launching the IoT Flash Tool

Run **FlashTool.exe** from IoT Flash Tool package. (refer Installing IoT Flash Tool)

The main GUI of the tool is shown in the below figure. Each item on the main GUI will be described in detail in the following sections.

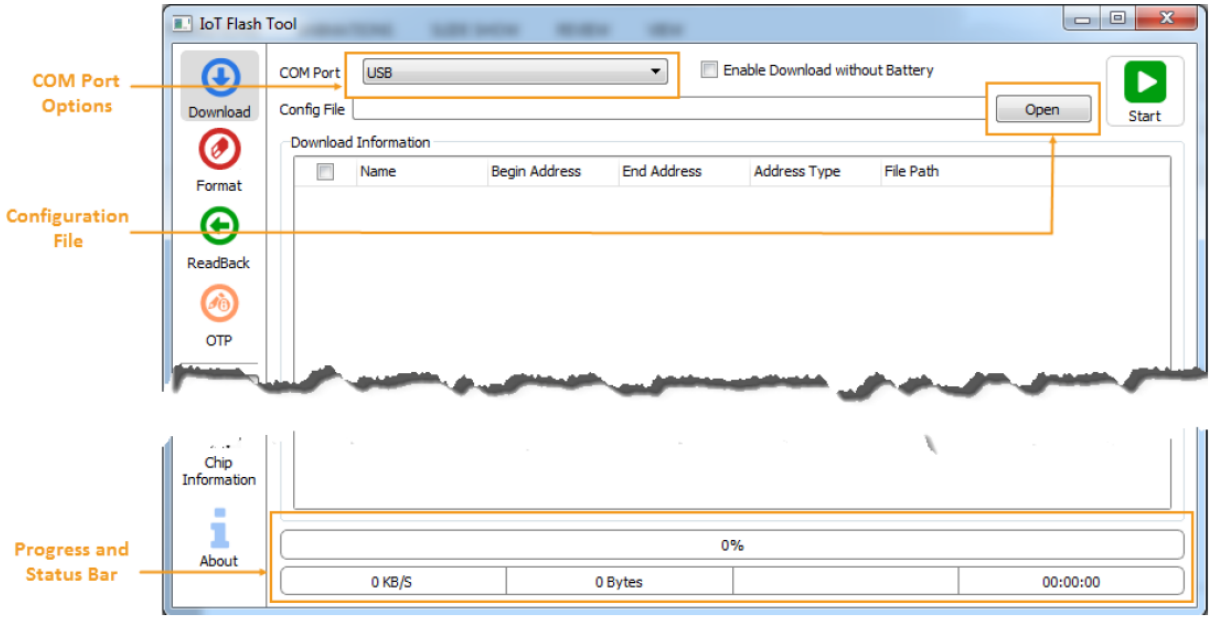


Figure 2.6.3-2: IoT Flash Tool's main GUI

2.6.3.3. Downloading the Firmware

Follow the steps to download the firmware to the target device over UART interface.

Step 1. Plug in the UART cable.

Step 2. Click **Download** on the left panel of the main GUI.

Step 3. Select UART port from the **COM Port** drop down menu.

Step 4. Click **Open** to provide the configuration file.

<sdk_root>\out\<board>\<project>\flash_download.cfg. Download Information will be displayed, including Name, Begin Address, End Address, Address Type and File Path of the firmware binary, as shown in the figure below:

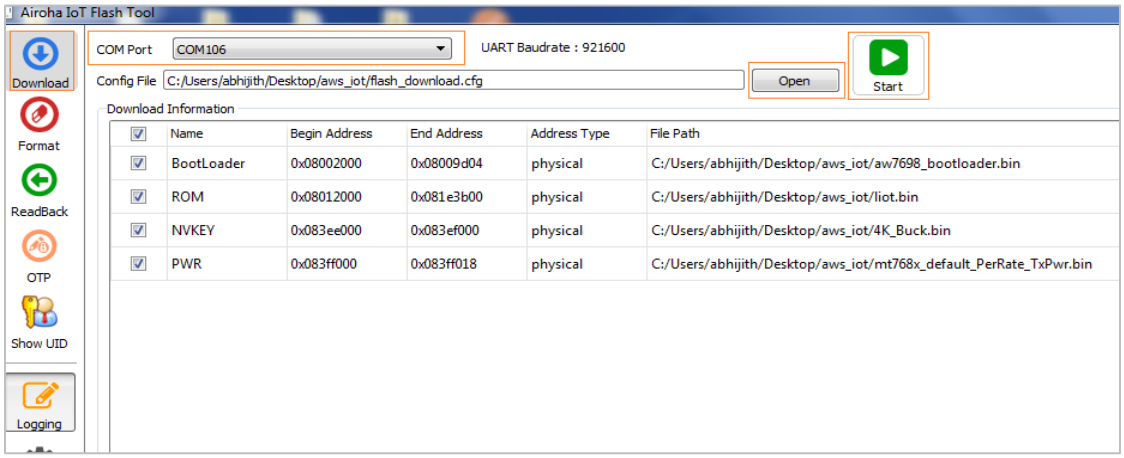


Figure 2.6.3-3: Download the firmware to a target device using UART connection

Step 5. Click **Start** to start downloading.

Step 6. Power on the device or press reset button on the device and then the process will start automatically.

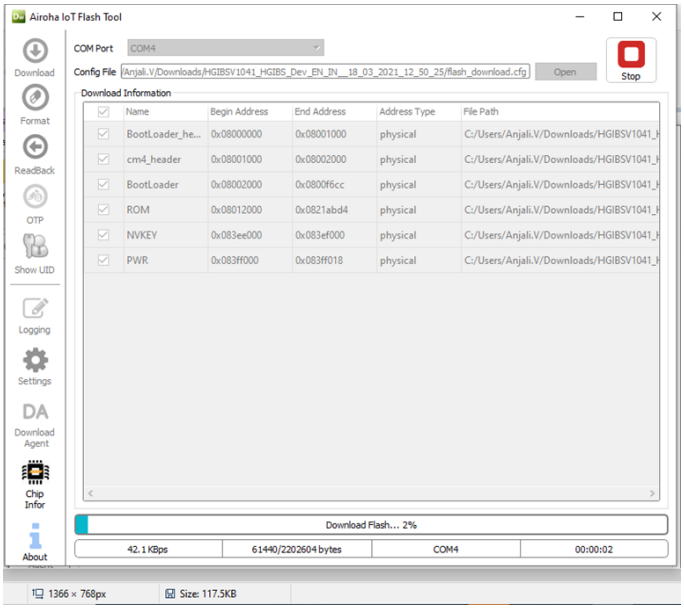


Figure 2.6.3-4: Firmware Flashing Screen

Step 7. After successful MAVID-3M EVK program, the following image message will be displayed:

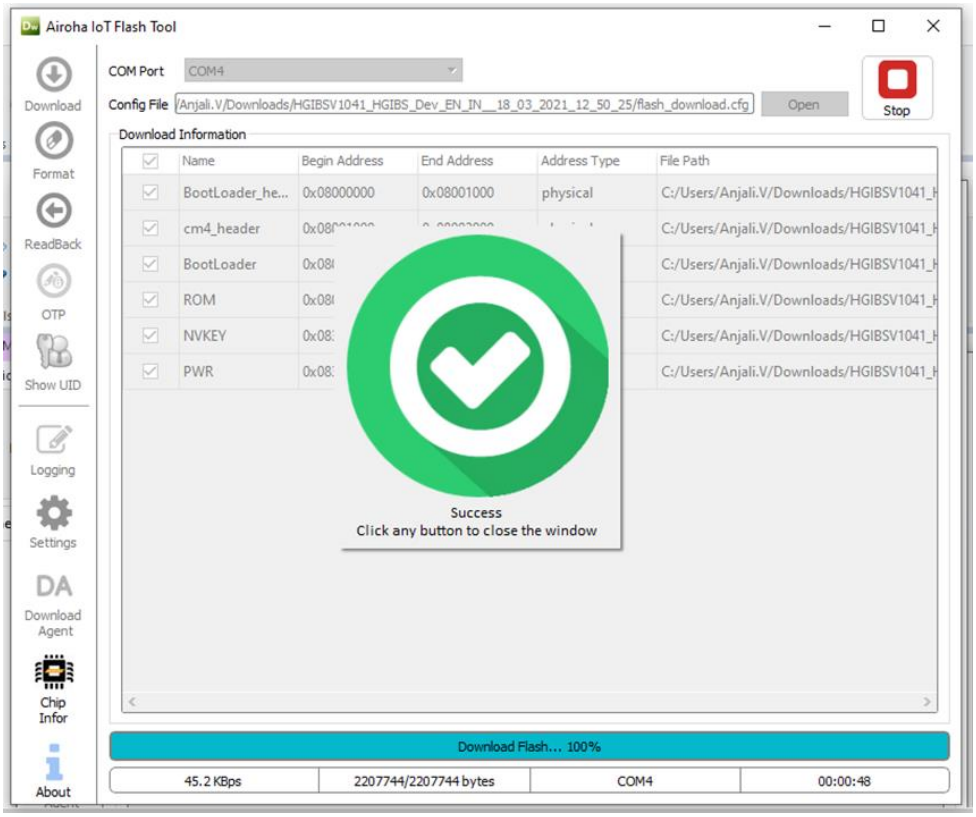


Figure 2.6.3-5: Firmware Flash Success

Step 8. If the program must be erased click on the Format tab on left, select the correct COM port, click on **Start** and reboot the board.

Note:	The status bar is in green color.

The following image displays Erasing flash/program:

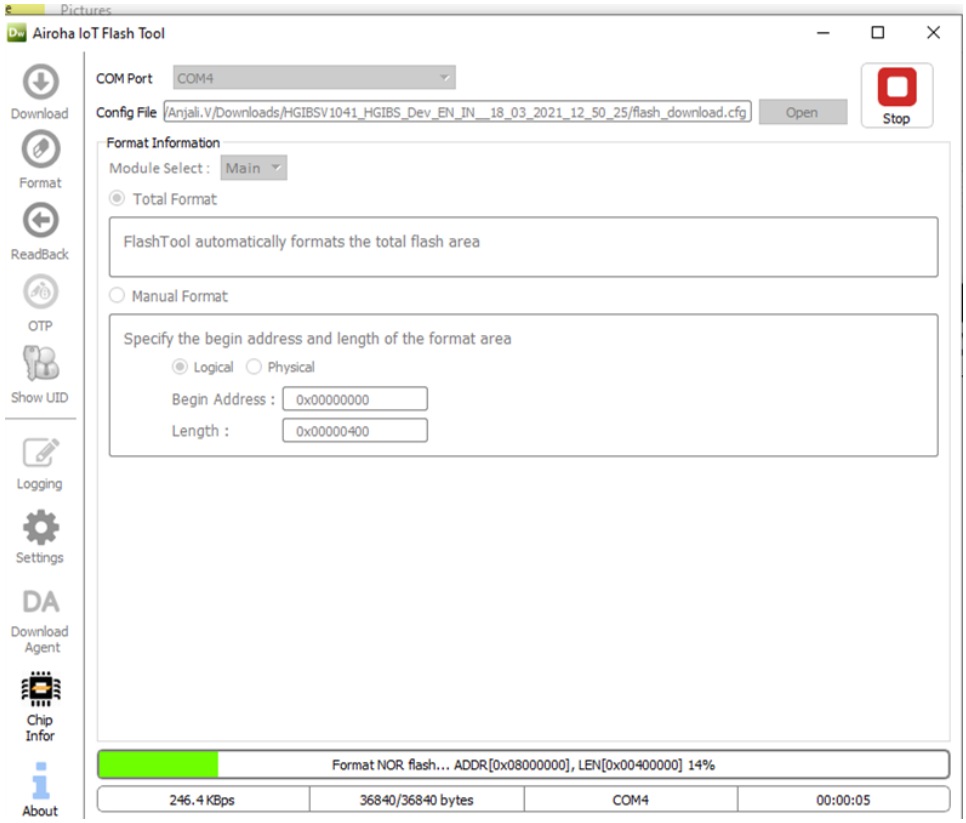


Figure 2.6.3-6: Erasing Flash

2.7. Monitor

To check if project is indeed running, open the terminal like gtkTerm and set the credentials.

1. Install any serial terminal.

   Example: Teraterm, putty, gtkterm, etc.

2. UART connector is pre-wired as shown in below diagram. Connect the serial cable to the device as shown below:

Hardware connection for UART cable is displayed below:

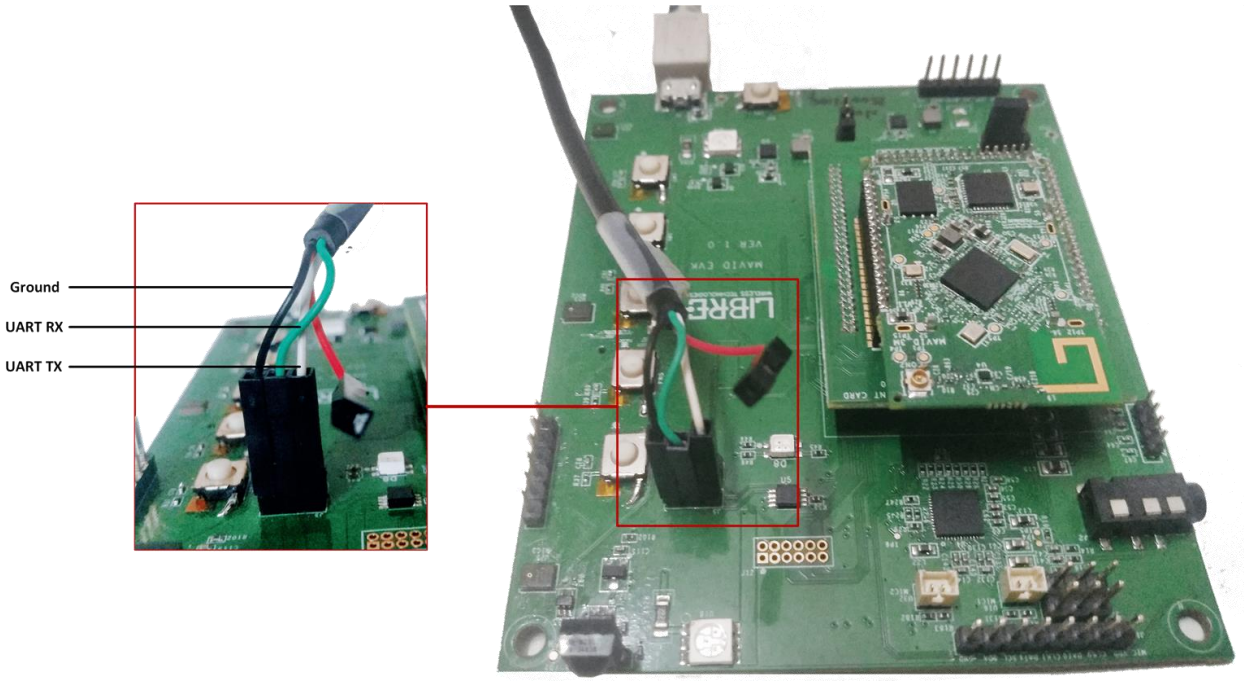


Figure 2.7-1: MAVID-3M EVK Setup

3. Open the serial terminal and configure the serial port as shown:

Baud Rate	115200
Data	8 bits
Parity	None
Stop Bits	1 bit
Flow Control	None
Enable LF on the serial receive	

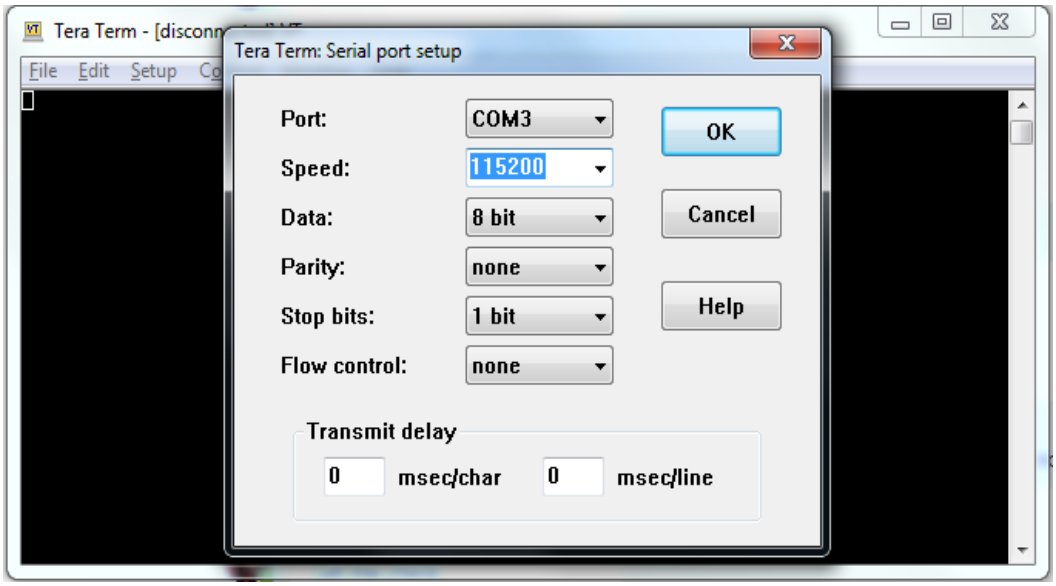


Figure 2.7-2: Serial port configuration

4. Reset the device.

Command Line Interpreter (CLI)

The CLI provides various useful debugging and system status commands.

Enter "?" to view the available commands and explore the options under each section.

```
?
en      - enter test mode
ip      - ip config
ipc     - ip mode/addr control
stat    - show statistics
ping    - ping <domain_name/addr> <count> <pkt_len>
iperf   - iperf
config  - user config read/write/reset/show
wifi    - wifi api
smart   - smart connection
log     - log control
reboot  - reboot
ver     - f/w ver
fw      - fw function
```

Figure 2.7-3: CLI command screen

Updating MAVID-3M_SDK

User should update MAVID-3M_SDK from time to time, as newer versions fix bugs and provide new features. The simplest way to do the update is to delete the existing MAVID-3M_SDK folder and follow the initial installation already described in section 2.2.

3. Sample Projects

3.1. Peripheral Interface

This project demonstrates the example projects of Peripherals of MAVID-3M_EVK.

MAVID-3M_EVK board contains following user-friendly Peripheral interfaces:

- 1x UART (currently used for debugging)
- 2x SPI (QSPI for Flash, SPI for Voice Front End)
- 1x I2C
- Supports up to 16 GPIOs
- 1x I2S Serial audio interface
- 1x RGB LED

For more details refer ***MAVID-3M Data Sheet*** document.

3.1.1. GPIO Example

This project demonstrates complete procedure to interface externals with GPIOs of MAVID-3M_EVK. User can interface the external LEDs with 100ohm resistor or according to LED voltage capacity and perform the task. User can also monitor in digital oscilloscope or in multimeter.

Some external pins are available to perform GPIO operations as listed below:

Use these GPIOs to perform the operations:

GPIO_5 ===> I2C_SCL

GPIO_6 ===> I2C_SDA

GPIO_14 ===> SPI_MISO

GPIO_15 ===> SPI_MOSI

GPIO_16 ===> SPI_CLK

GPIO_17 ===> SPI_CS

Find the GPIO example project at "<sdk_root>/project/aw7698_evk/apps/gpio".

Follow the given steps to customize the project:

Step 1. The program main can be found at "<sdk_root>/project/aw7698_evk/apps/gpio/src/main.c".

Step 2. Call **system_init** function for clock configuration and required peripherals initialization.

Step 3. Create the tasks to perform the example project.

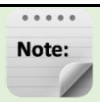
Step 4. Configure the structures/unions to set the parameters.

Step 5. Call the respective Peripheral HAL_init for peripheral configuration and initialization with structures/unions or enum values found in hal_gpio.h file.

Step 6. Call the respective GPIO Peripheral API functions to perform the task, which is found in hal_gpio.h file.

Step 7. Call the respective HAL_deinit Peripheral functions for deinitialization found in hal_gpio.h file.

Step 8. Schedule the scheduler to start the tasks.



Set without_app =1 in main.c.

Initialize CLI task to enable user input CLI command from UART port.

Step 9. Follow the procedure to build the project as given in section 2.5.

Step 10. Follow the procedure to flash the code on board as given in section 2.6.

Step 11. Follow the procedure to debug and monitor as given in section 2.7.

3.1.2. UART Example

This project demonstrates complete procedure for UART communication of 7698 EVK.

A Universal Asynchronous Receiver/Transmitter (UART) is a hardware feature that handles communication (i.e., timing requirements and data framing) using widely adapted asynchronous serial communication interfaces. A UART provides widely adopted method to realize full-duplex or half-duplex data exchange among different devices.

UART controller is independently configurable with parameters such as baud rate, data bit length, bit ordering, number of stop bits, parity bit, etc. All the controllers are compatible with UART-enabled devices from various manufactures, use that devices to perform UART communication.

Find the UART example project from "<sdk_root>/project/aw7698_evk/apps/uart".

Use **hal_uart_config_t** structure to configure UART parameters like baud rate, parity bits, stop bits and word length.

```
hal_uart_config_t uart_config;
uart_config.baudrate = HAL_UART_BAUDRATE_115200;
uart_config.parity = HAL_UART_PARITY_NONE;
uart_config.stop_bit = HAL_UART_STOP_BIT_1;
uart_config.word_length = HAL_UART_WORD_LENGTH_8;
```

Use Preconfigured UART port **HAL_UART_0** for UART communication. This port is also connected for CLI.

Follow the given steps to customize the project:

Step 1. The program main can be found at "<sdk_root>/project/aw7698_evk/apps/uart/src/main.c".

Step 2. Call **system_init** function for clock configuration and required peripherals initialization.

Step 3. Create the tasks to perform the example project.

Step 4. Configure the structures/unions to set the parameters.

Step 5. Call the respective Peripheral HAL_init for peripheral configuration and initialization with structures/unions or enum values found in hal_uart.h file.

Step 6. Call the respective UART Peripheral API functions to perform the task found in hal_uart.h file.

Step 7. Call the respective HAL_deinit Peripheral functions for deinitialization found in hal_uart.h file.

Step 8. Schedule the scheduler to start the tasks.

Set without_app = 1 in main.c.

Step 9. Follow the procedure to build the project as given in section 2.5.

Step 10. Follow the procedure to flash the code on board as given in section 2.6.

Step 11. Follow the procedure to debug and monitor as given in section 2.7.

3.1.3. SPI Example

This project demonstrates complete procedure to interface externals with SPI of 7698 EVK.

SPI is a synchronous, full duplex master-slave-based interface. The data from the master or the slave is synchronized on the rising or falling clock edge. Both master and slave can transmit data at the same time. The SPI interface can be either 3-wire or 4-wire. This section focuses on the popular 4-wire SPI interface.

Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device that the user wishes to talk to.

External pins are available to perform SPI operations as listed below:

GPIO_14 ===> SPI_MISO

GPIO_15 ===> SPI_MOSI

GPIO_16 ===> SPI_CLK

GPIO_17 ===> SPI_CS

Find the SPI example project from "<sdk_root>/project/aw7698_evk/apps/spi".

Use **hal_spi_master_config_t** structure to configure SPI parameters like bit order slave port, clock frequency, clock phase and clock polarity.

```
hal_spi_master_config_t SPI_config;
spi_config.bit_order = HAL_SPI_MASTER_LSB_FIRST;
spi_config.slave_port = HAL_SPI_MASTER_SLAVE_0;
spi_config.clock_frequency = 1000000;
spi_config.phase = HAL_SPI_MASTER_CLOCK_PHASE0;
spi_config.polarity = HAL_SPI_MASTER_CLOCK_POLARITY0;
```

Use **hal_spi_master_send_and_receive_config_t** structure to configure receive and send parameters like receive data length, send data length, send data buffer and receive data buffer.

```
spi_send_and_receive_config.receive_length = 2;
spi_send_and_receive_config.send_length = 1;
spi_send_and_receive_config.send_data = &status_cmd;
spi_send_and_receive_config.receive_buffer = status_receive;
```

Follow the given steps to customize the project:

Step 1. The program main can be found at "<sdk_root>/project/aw7698_evk/apps/spi/src/main.c".

Step 2. Call **system_init** function for clock configuration and required peripherals initialization.

Step 3. Create the tasks to perform the example project.

Step 4. Configure the structures/unions to set the parameters.

Step 5. Call the respective Peripheral HAL_init for peripheral configuration and initialization with structures/unions or enum values found in hal_spi_master.h file.

Step 6. Call the respective SPI Peripheral API functions to perform the task found in hal_spi_master.h file.

Step 7. Call the respective HAL_deinit Peripheral functions for deinitialization found in hal_spi_master.h file.

Step 8. Schedule the scheduler to start the tasks.

Set without_app =1 in main.c.

Initialize CLI task to enable user input CLI command from UART port.

Step 9. Follow the procedure to build the project as given in section 2.5.

Step 10. Follow the procedure to flash the code on board as given in section 2.6.

Step 11. Follow the procedure to debug and monitor as given in section 2.7.

3.1.4. I2C Example

This project demonstrates complete procedure to interface externals with I2C of 7698 EVK. The Inter-Integrated Circuit (I2C) Protocol is a protocol intended to allow multiple "peripheral" digital integrated circuits ("chips") to communicate with one or more "controller" chips. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. It only requires two signal wires to exchange information.

External pins are available to perform I2C operations as listed below:

GPIO_5 ===> I2C_SCL

GPIO_6 ===> I2C_SDA

Find the I2C example project from "<sdk_root>/project/aw7698_evk/apps/i2c".

Use **hal_i2c_config_t** structure to configure I2C frequency setting:

```
Hal_i2c_config_t i2c_config;
```

i2c_config.frequency = HAL_I2C_FREQUENCY_400K;

Follow the given steps to customize the project:

Step 1. The program main can be found at "<sdk_root>/project/aw7698_evk/apps/i2c/src/main.c".

Step 2. Call **system_init** function for clock configuration and required peripherals initialization.

Step 3. Create the tasks to perform the example projects.

Step 4. Configure the structures/unions to set the parameters.

Step 5. Call the respective Peripheral HAL_init for peripheral configuration and initialization with structures/unions or enum values found in hal_i2c_master.h file.

Step 6. Call the respective I2C Peripheral API functions to perform the task found in hal_i2c_master.h file.

Step 7. Call the respective HAL_deinit Peripheral functions for deinitialization found in hal_i2c_master.h file.

Step 8. Schedule the scheduler to start the tasks.

Set without_app =1 in main.c.

Initialize CLI task to enable user input CLI command from UART port.

Step 9. Follow the procedure to build the project as given in section 2.5.

Step 10. Follow the procedure to flash the code on board as given in section 2.6.

Step 11. Follow the procedure to debug and monitor as given in section 2.7.

3.1.5. LED_BLINK/LED_BREATH

This project demonstrates complete procedure to interface RGB LED of 7698 EVK.

Use the given preconfigured RGB LED, which is connected with BT chip with GPIOs **EX_GPIO_18, EX_GPIO_1 and EX_GPIO_15** to perform led_blink/led_breath operations as listed below:

BSP_LED_0

BSP_LED_1

BSP_LED_2

Find the led_blink example project from "<sdk_root>/project/aw7698_evk/apps/led_blink".

Use **bsp_led_config_t** structure to configure ON, OFF time and repetition of RGB LED.

Follow the given steps to customize the project:

Step 1. The program main can be found at "<sdk_root>/project/aw7698_evk/apps/led_blink/src/main.c".

Step 2. Call **system_init** function for clock configuration and required peripherals initialization.

Step 3. Call **bt_create_task** function to start BT module.

Step 4. Create the tasks to perform the example project.

Step 5. Configure the structures/unions to set the parameters found in bsp_led.h file.

Step 6. Schedule the scheduler to start the tasks.

Set without_app =1 in main.c.

Initialize CLI task to enable user input CLI command from UART port.

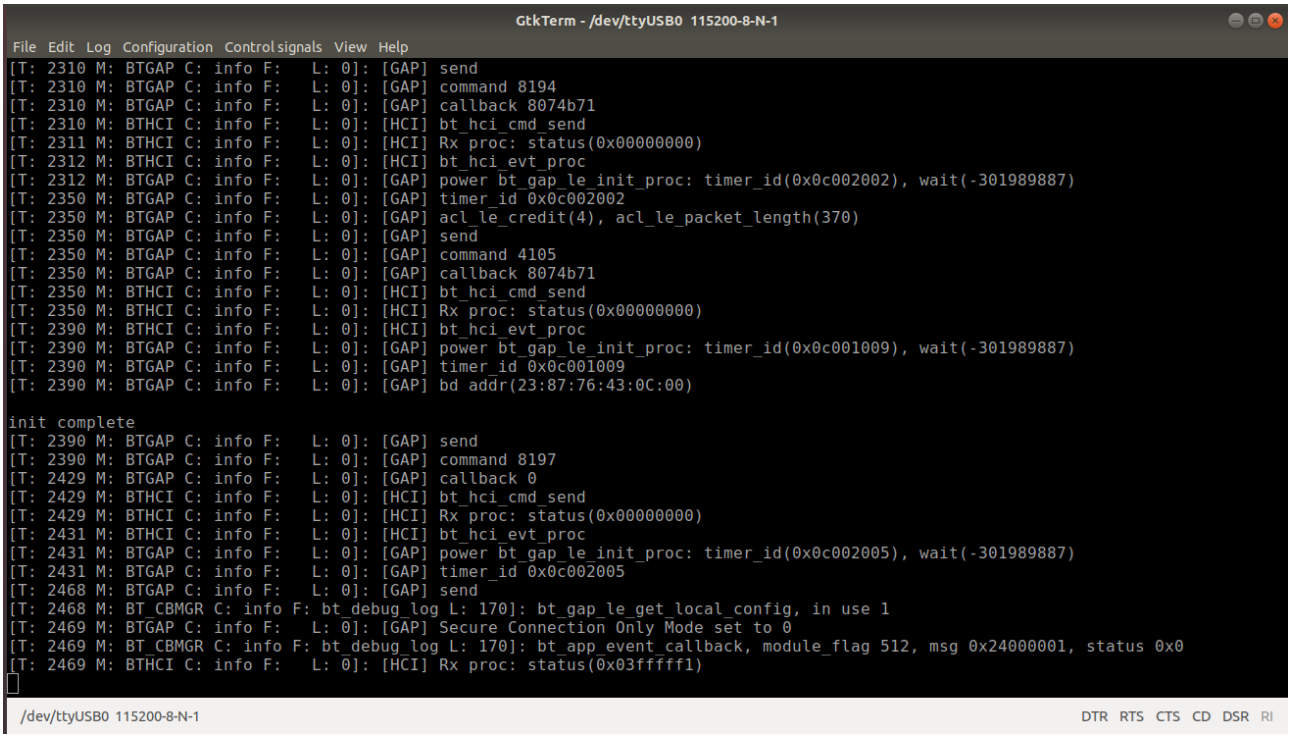
For LED control Project call **bt_create_task,** RGB led is interfaced with BT chip.

Step 7. Follow the procedure to build the project as given in section 2.5.

Step 8. Follow the procedure to flash the code on board as given in section 2.6.

Step 9. Follow the procedure to debug and monitor as given in section 2.7.

While running the code BT will start and it will display the logs as shown below:



3.2. Networking

3.2.1. Wi-Fi Setup Example (wifi_STA)

This project demonstrates Wi-Fi configuration of MAVID-3M EVK. The Wi-Fi credentials are configured within the code at compile time or using CLI commands at run time.

The example project is available at "<sdk_root>/project/aw7698_evk/apps/wifi_STA".

Step 1. To configure the Wi-Fi at compile time set **PRE_CONFIG** to **1** and set the credentials at "<sdk_root>/project/aw7698_evk/apps/wifi_STA/src/main.c".

```
#define WIFI_SSID ("ssid")
#define WIFI_PASSWORD ("12345678")
int PRE_CONFIG=1;
```

Step 2. To configure the Wi-Fi at run time using CLI commands set **PRE_CONFIG** to **0** at "<sdk_root>/project/aw7698_evk/apps/wifi_STA/src/main.c" (refer to section 3.2.1.1 for CLI commands).

```
int PRE_CONFIG=0;
```

Step 3. Call the **system_init** for clock configuration and required peripherals initialization.

Step 4. Follow the procedure to build the project as given in section 2.5.

Step 5. Follow the procedure to flash the code on board as given in section 2.6.

Step 6. Follow the procedure to debug and monitor as given in section 2.7.

Step 7. Once the device is rebooted, Wi-Fi can be configured using CLI commands (refer to section 3.2.1.1 for CLI commands).

Initialize CLI task to enable user input CLI command from UART port.

3.2.1.1. CLI Commands to Configure Wi-Fi

The following are example CLI commands to configure the Wi-Fi:

1. config write <group_name> <data_item_name> <value>

Write value to data_item_name in specified group_name

e.g.:

config write STA Ssid <ssid_name>

config write STA WpaPsk <password>

2. config read <group_name> <data_item_name>

Read value of data_item_name in specified group_name

e.g.:

config read STA Ssid
config read STA WpaPsk

3. config show

It will display all group data items with its value

e.g.:

config show

[common]Opmode: 1

[STA]Ssid: <ssid_name>

4. config show <group_name>

It will display specified group data items with it's value

e.g.:

config show STA

[STA]Ssid: <ssid_name>

5. config reset

It will reset all group data items with default value provided to it

e.g.:

config reset

6. config reset <group_name>

It will reset specified group data items with default value provided to it

e.g.:

config reset STA

7. To reboot use reboot

e.g.: reboot

The following list demonstrate the CLI commands Group names and Data item names:

<group_name>	**<data_item_name>**
common	OpMode
common	CountryCode
common	CountryRegion
common	CountryRegionABand
common	RadioOff

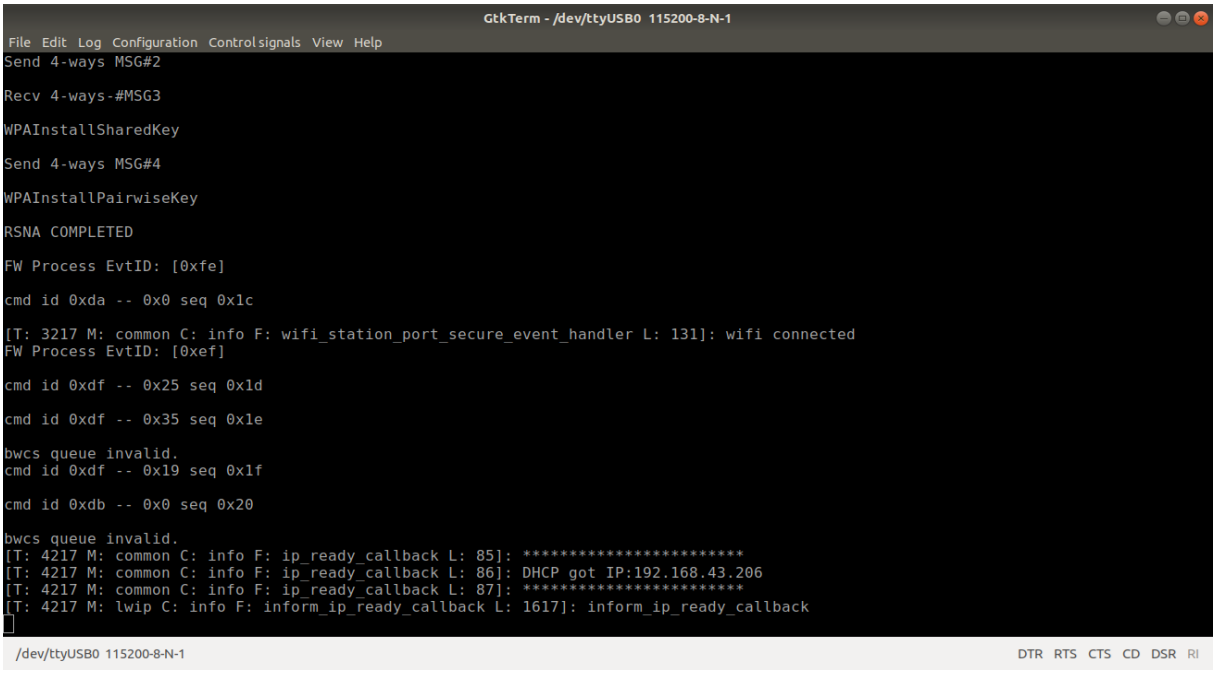
<group_name>	<data_item_name>
common	DbgLevel
common	RTSThreshold
common	FragThreshold
common	BGChannelTable
common	AChannelTable
common	syslog_filters
common	WiFiPrivilegeEnable
common	StaFastLink
STA	LocalAdminMAC
STA	MacAddr
STA	Ssid
STA	SsidLen
STA	BssType
STA	Channel
STA	BW
STA	wirelessMode
STA	BADecline
STA	AutoBA
STA	HT_MCS
STA	HT_BAWinSize
STA	HT_GI
STA	HT_PROTECT
STA	HT_EXTCHA

<group_name>	<data_item_name>
STA	WmmCapable
STA	ListenInterval
STA	AuthMode
STA	EncrypType
STA	WpaPsk
STA	WpaPskLen
STA	PMK_INFO
STA	PairCipher
STA	GroupCipher
STA	DefaultKeyId
STA	SharedKey
STA	SharedKeyLen
STA	PSMode
STA	KeepAlivePeriod
STA	BeaconLostTime
STA	ApcliBWAutoUpBelow
STA	StaKeepAlivePacket
AP	LocalAdminMAC
AP	MacAddr
AP	Ssid
AP	SsidLen
AP	Channel
AP	BW

<group_name>	<data_item_name>
AP	WirelessMode
AP	AutoBA
AP	HT_MCS
AP	HT_BAWinSize
AP	HT_GI
AP	HT_PROTECT
AP	HT_EXTCHA
AP	WmmCapable
AP	DtimPeriod
AP	AuthMode
AP	EncrypType
AP	WpaPsk
AP	WpaPskLen
AP	PairCipher
AP	GroupCipher
AP	DefaultKeyId
AP	SharedKey
AP	SharedKeyLen
AP	HideSSID
AP	RekeyInterval
AP	AutoChannelSelect
AP	BcnDisEn
network	IpAddr

<group_name>	<data_item_name>
network	IpNetmask
network	IpGateway
network	IpMode

CLI Wi-Fi Config Example:



3.2.2. Wi-Fi Setup with Phone App (wifi_STA_App)

This project demonstrates the Wi-Fi configuration of MAVID-3M EVK using mobile application and CLI commands.

Find the wifi_STA_App example project at "<sdk_root>/project/aw7698_evk/apps/wifi_STA_App".

Follow the given steps to customize the project:

Step 1. The program main can be found at "<sdk_root>/project/aw7698_evk/apps/wifi_STA_App/src/main.c".

Step 2. Call **system_init** function for clock configuration and required peripherals initialization.

Step 3. Create the **LibreNetWorkInit** task for Mobile Phone Application.

Step 4. Create the **ip_ready_task** task to get IP for connection.

Step 5. Schedule the scheduler to start the tasks.

Step 6. Follow the procedure to build the project as given in section 2.5.

Step 7. Follow the procedure to flash the code on board as given in section 2.6.

Step 8. Follow the procedure to debug and monitor as given in section 2.7.

Step 9. Once the device is rebooted, the device will enter into setup mode if Wi-Fi credentials are not pre-configured.

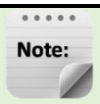
Step 10. If the device has pre-configured credentials, the device can be forced into setup mode with the following CLI command.

```
libre keys setup
```

The device will reboot and enter into setup mode.

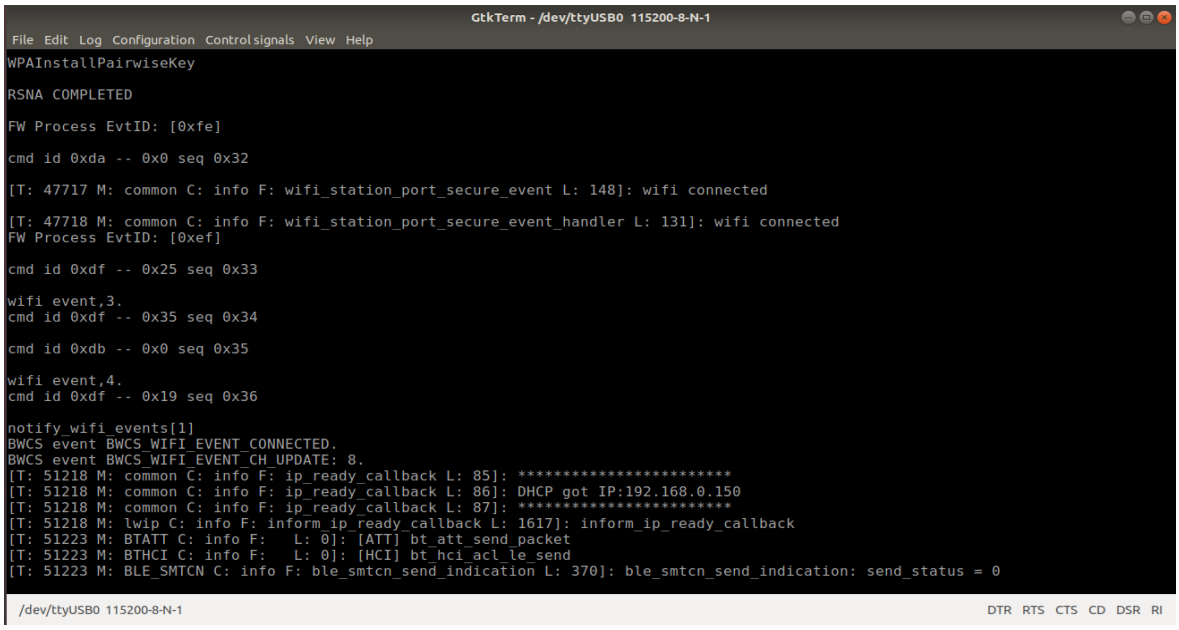
Step 11. Once the device is in setup mode, follow the procedure as mentioned in the *section 4* in **MAVID-3M EVK - User Guide** document to configure Wi-Fi using mobile application. The mobile application is available at <sdk_root>/tools/apps.

Step 12. Wi-Fi can also be configured using CLI commands (refer to section 3.2.1.1 for CLI commands).

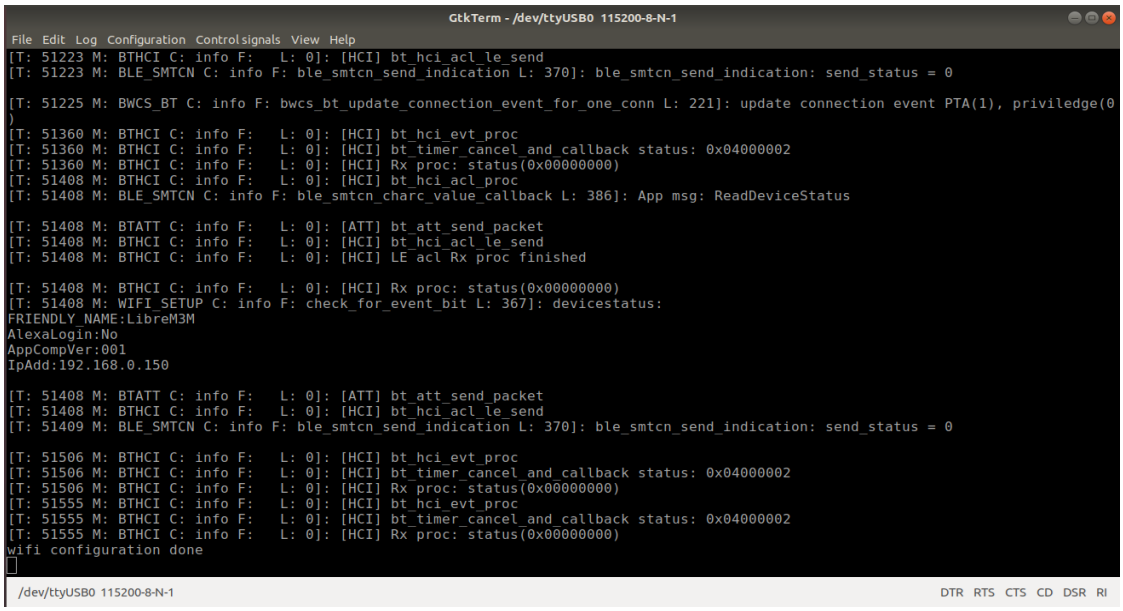


Initialize CLI task to enable user input CLI command from UART port.

The example output is as displayed below:



Example output screen:



3.2.3. My First IoT Example (aws_Publish_Subscribe)

This project demonstrates the AWS Publish Subscribe Example project of MAVID-3M_EVK. This application is a simple demonstration program which shows how to use the AWS IoT APIs and start a service running AWS IoT MQTT client.

This application explain user to how to:

1. Connect to AWS IoT proxy via MQTT over TLS.
2. Publish topic to MQTT server.
3. Subscribe topic to MQTT server.
4. Stop connection, disconnect MQTT.

Find the AWS_Publish_Subscribeexample project at "<sdk_root>/project/aw7698_evk/apps/ AWS_Publish_Subscribe".

Step 1. The program main can be found at "<sdk_root>/project/aw7698_evk/apps/aws_Publish_Subscribe/src/main.c".

Step 2. Get AWS IoT certificates from AWS IoT Console and insert the certificates into "<sdk_root>/project/common/certs_protected/aws_iot_cert_rtos.h".

Step 3. To get the AWS IoT certificate follow section 4.

Step 4. Enable macro **#define PUBLISH** and **#define SUBSCRIBE** in <sdk_root>\project\aw7698_evk\apps\aws_Publish_Subscribe\src\aws_iot_pub_sub.c, according to requirement, at a time both can be enabled to perform loopback Publish and Subscribe operation.

Step 5. Update the endpoint URL, thing name, client ID and topic name to AWS_IOT_MQTT_HOST, AWS_IOT_MY_THING_NAME, INTEGRATION_TEST_CLIENT_ID and INTEGRATION_TEST_TOPIC macros found in <sdk-root>\middleware\third_party\aws_iot\include\aws_iot_config.h.

Step 6. Use **IoT_Client_Init_Params** structure to initialize MQTT parameters.

Step 7. Use **IoT_Client_Connect_Params** structure to initialize connection parameters.

Step 8. Use **IoT_Publish_Message_Params** structure to create QOS0 and QOS1 to Publish message to console.

> **Note:** Initialize CLI task to enable user input CLI command from UART port.

Step 9. Follow the procedure to build the project as given in section 2.5.

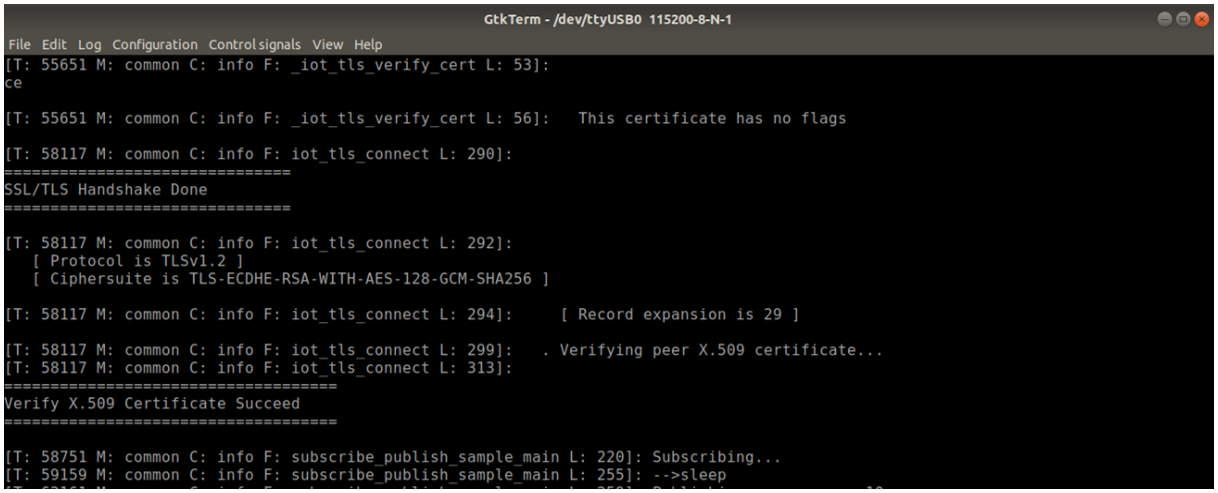
Step 10. Follow the procedure to flash the code on board as given in section 2.6.

Step 11. Follow the procedure to debug and monitor as given in section 2.7.

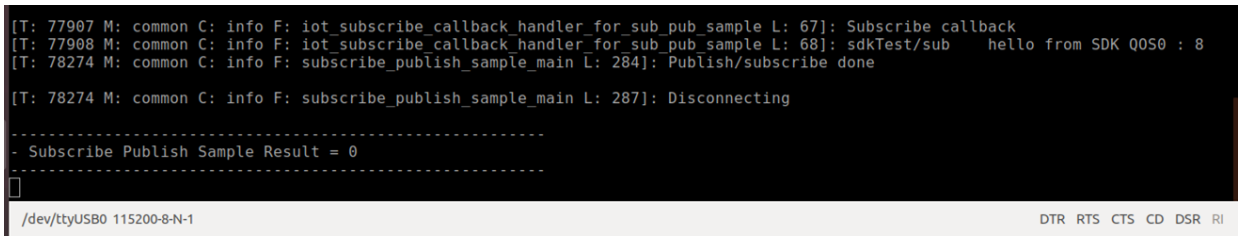
Step 12. Once the device is rebooted, configure Wi-Fi by following the procedure as given in ***Step 11*** of section 3.2.2.

Subscribe the topic:

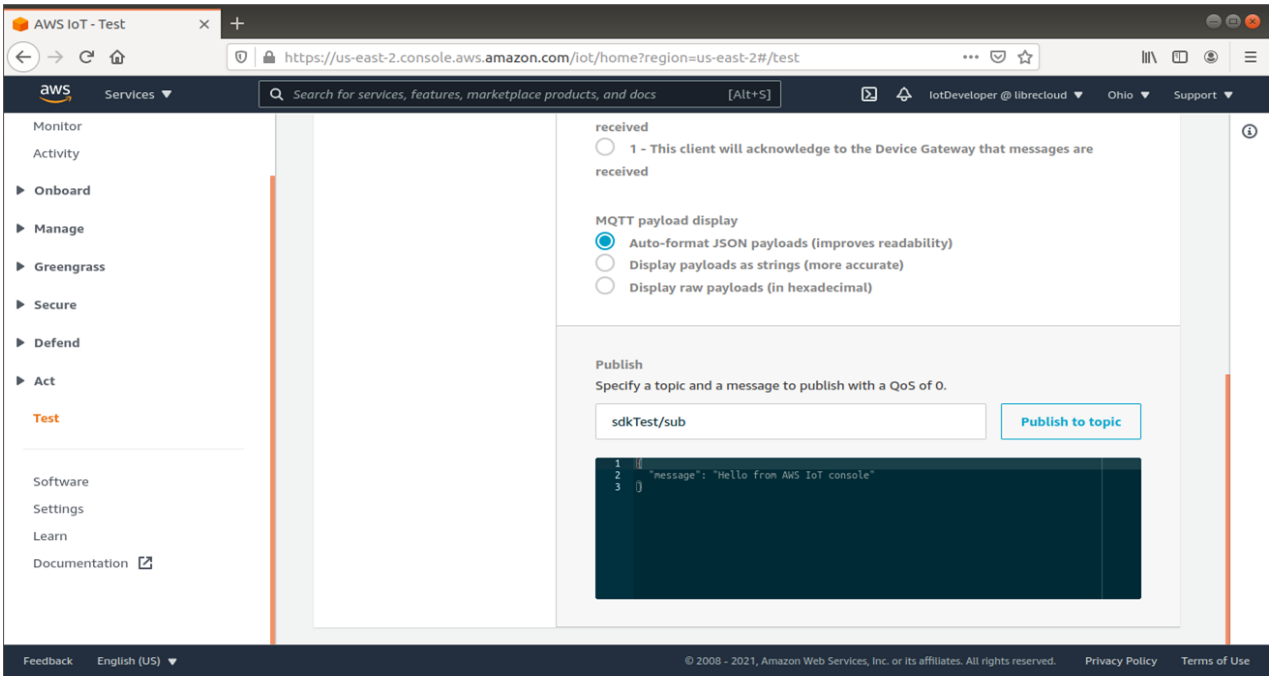
Run the Application, the below logs will be displayed:



Output logs are as shown below:

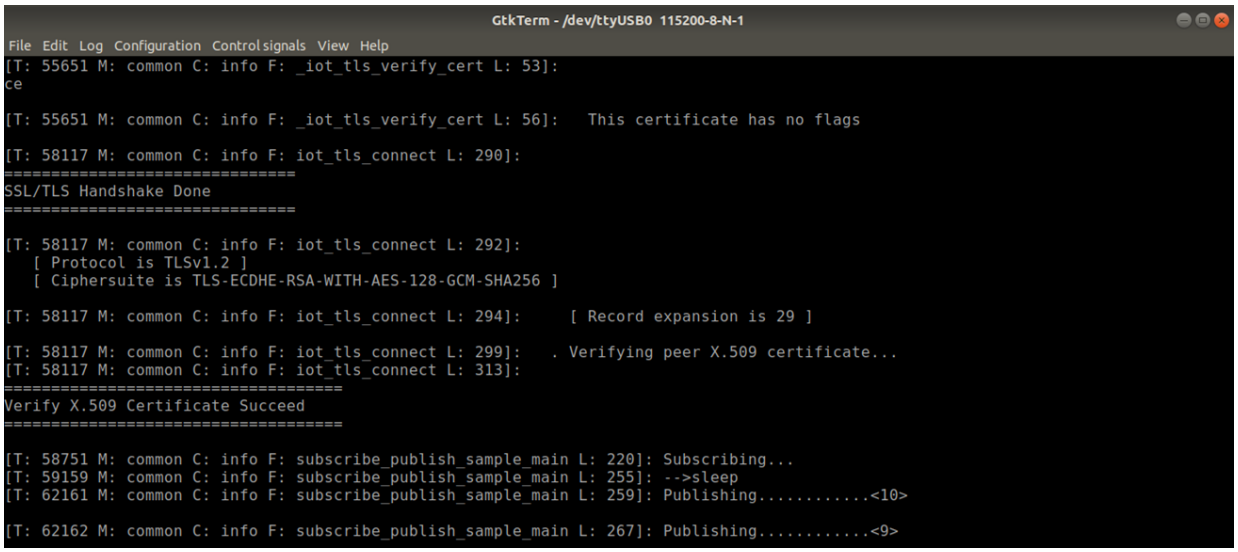


From AWS IoT console publish the topic from test as shown below:

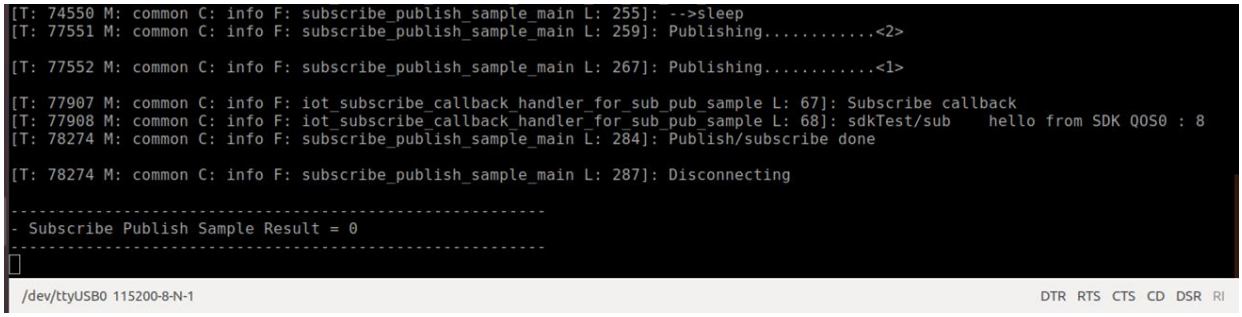


Publish:

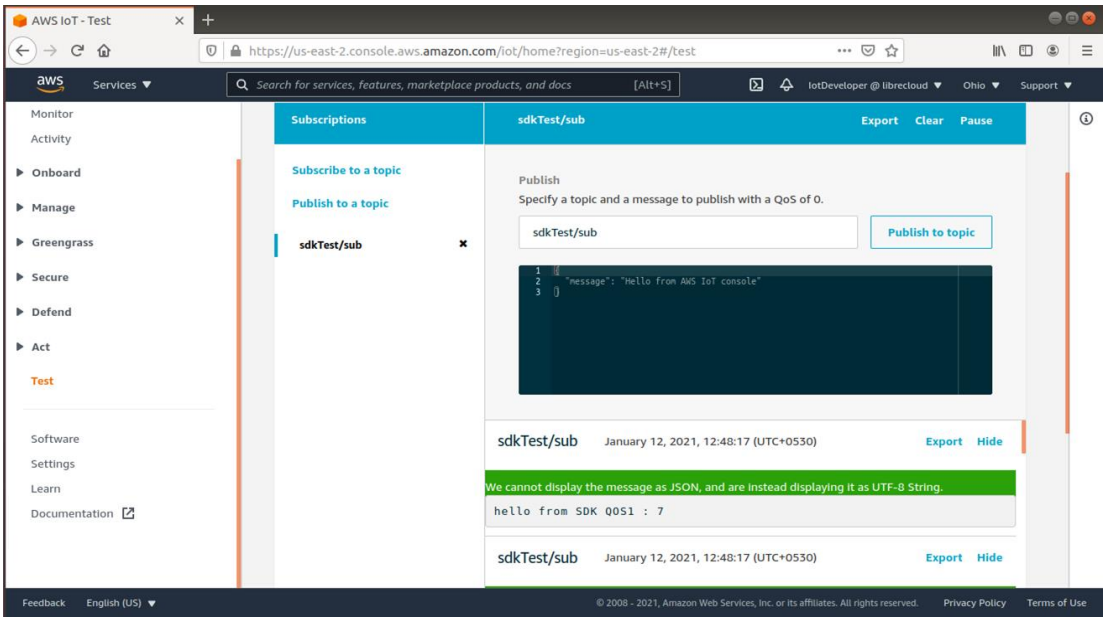
Run the Application, the below logs will be displayed:



Output logs are as shown below:



From AWS IoT console subscribe the topic from test as shown below:



3.2.4. IoT Shadow Example (aws_Shadow)

This project demonstrates the AWS SHADOW example project of the MAVID-3M_EVK. This application is a simple demonstration program which shows how to use the AWS IoT APIs and start a service running AWS IoT MQTT client.

This application explain user to how to:

1. Connect to AWS IoT proxy via MQTT over TLS.
2. Create shadow in AWS IoT server via publish some MQTT message.

3. Update shadow status (five times).
4. Stop connection, disconnect MQTT.

Find the AWS_Shadow example project at "<sdk_root>/project/aw7698_evk/apps".

Step 1. The program main can be found at "<sdk_root>/project/aw7698_evk/apps/aws_Shadow/src/main.c".

Step 2. Please get AWS IoT certificates from AWS IOT Console, then insert the certificates into "<sdk_root>/project/common/certs_protected/aws_iot_cert_rtos.h".

Step 3. To get the AWS IoT certificate follow section 4.

Step 4. Update the endpoint URL, thing name, client ID and topic name to AWS_IOT_MQTT_HOST, AWS_IOT_MY_THING_NAME, INTEGRATION_TEST_CLIENT_ID and INTEGRATION_TEST_TOPIC macros found in <sdk-root>\middleware\third_party\aws_iot\include\aws_iot_config.h .

Step 5. Use **ShadowInitParameters_t** structure to initialize shadow parameters.

Step 6. Use **ShadowConnectParameters_t** structure to configure shadow connect parameters.

Step 7. Use **jsonStruct_t** structure to configure JSON string.

Initialize CLI task to enable user input CLI command from UART port.

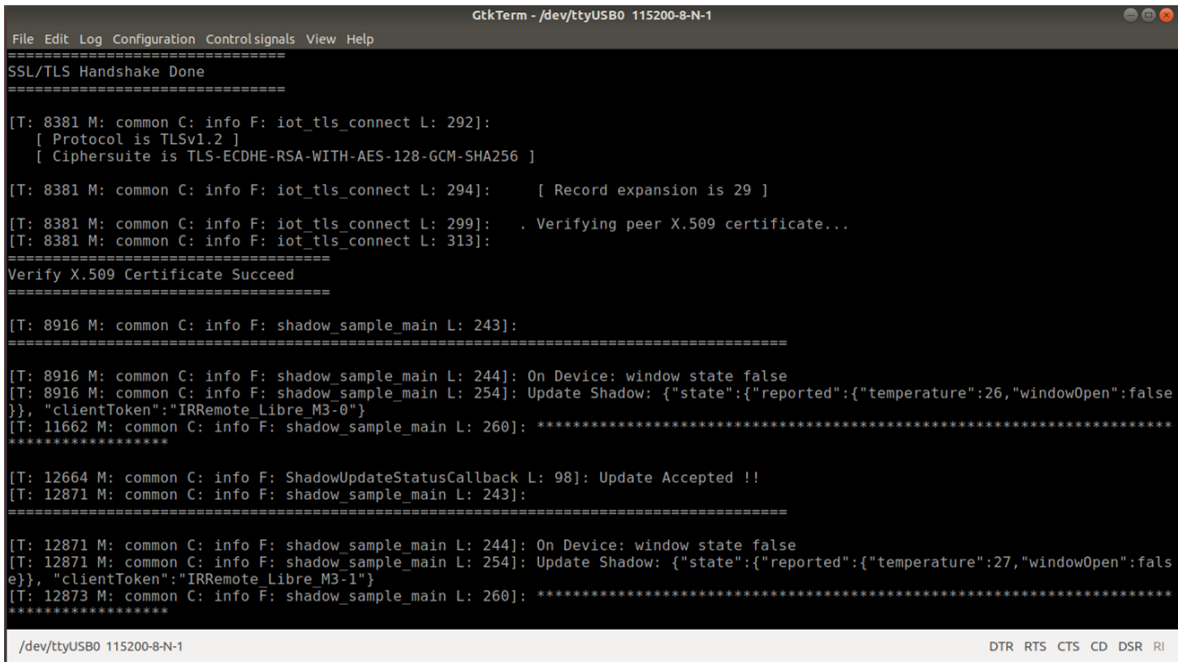
Step 8. Follow the procedure to build the project as given in section 2.5.

Step 9. Follow the procedure to flash the code on board as given in section 2.6.

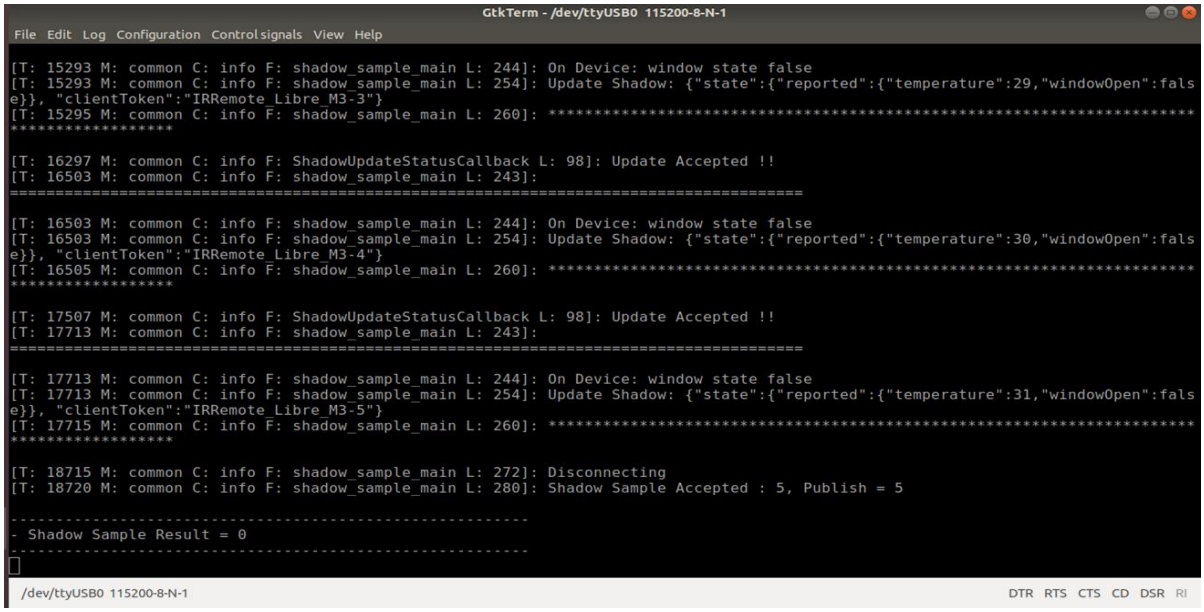
Step 10. Follow the procedure to debug and monitor as given in section 2.7.

Step 11. Once the device is rebooted, configure Wi-Fi by following the procedure as given in ***Step 11*** of section 3.2.2.

Run the Application, the below logs will be displayed:

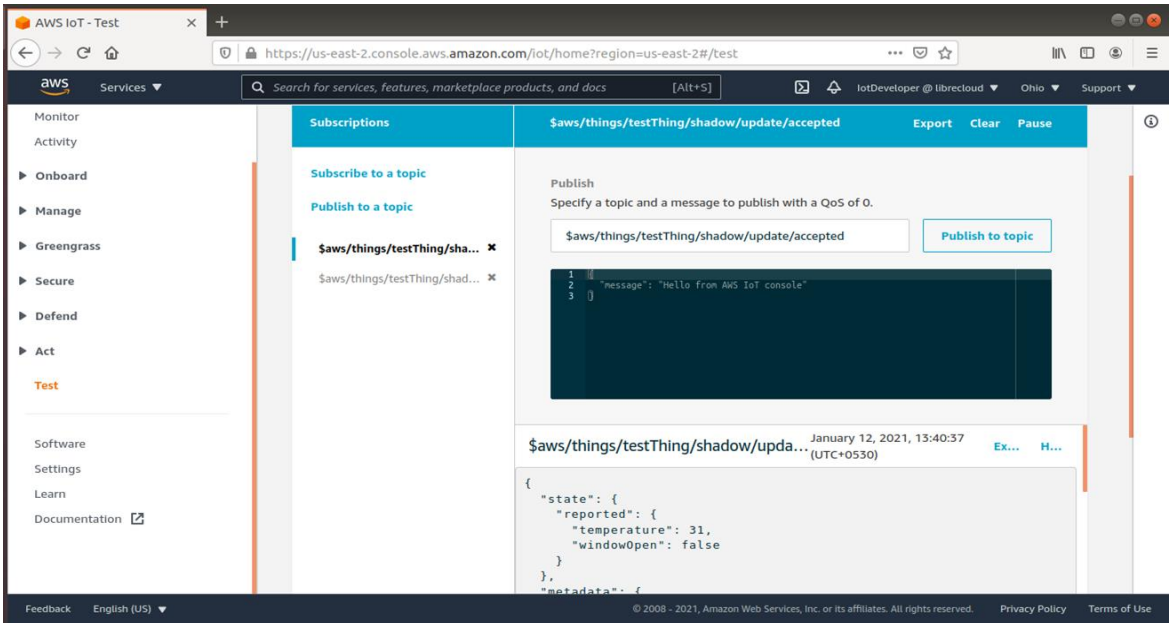


Final output screen:

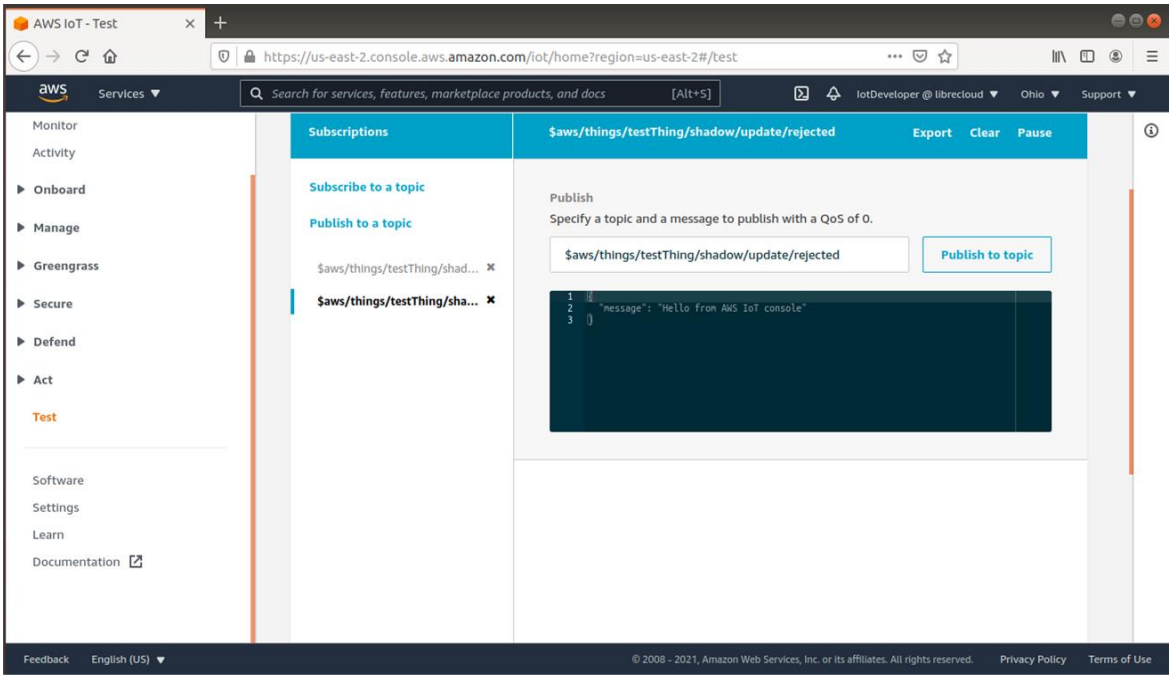


From AWS IoT Console subscribe as per the command given below:

$aws/things/testThing/shadow/update/accepted

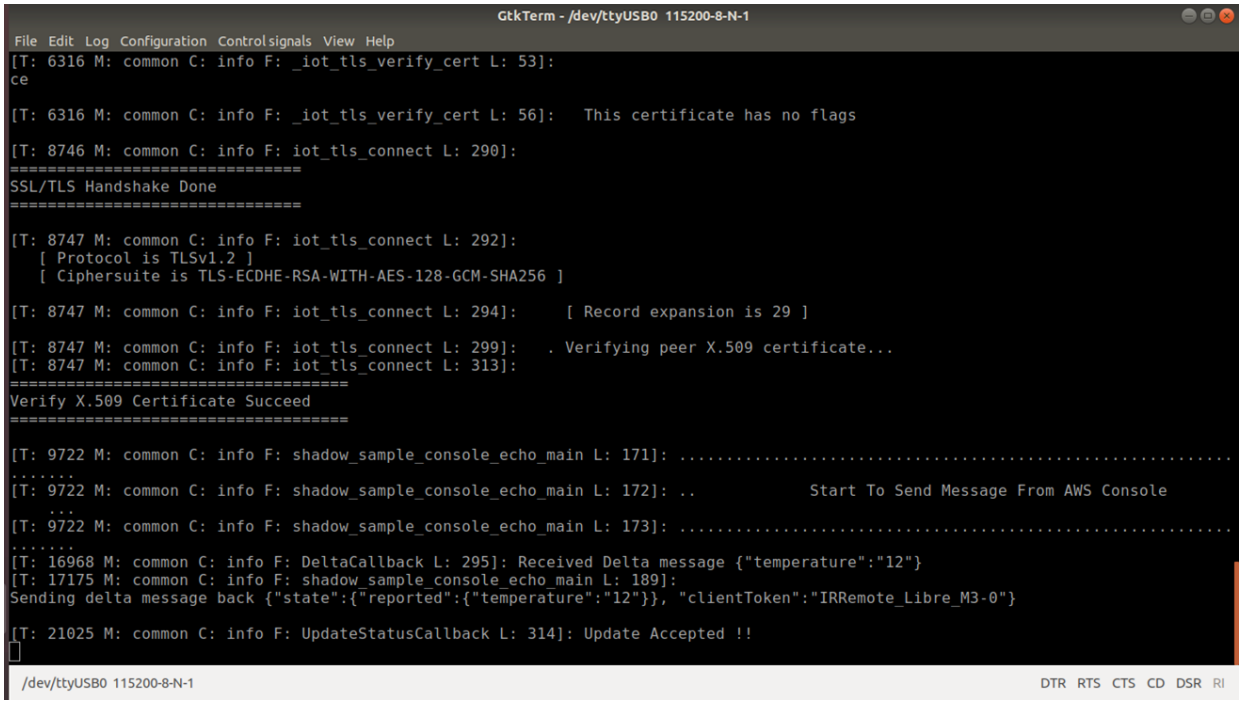


$aws/things/testThing/shadow/update/rejected



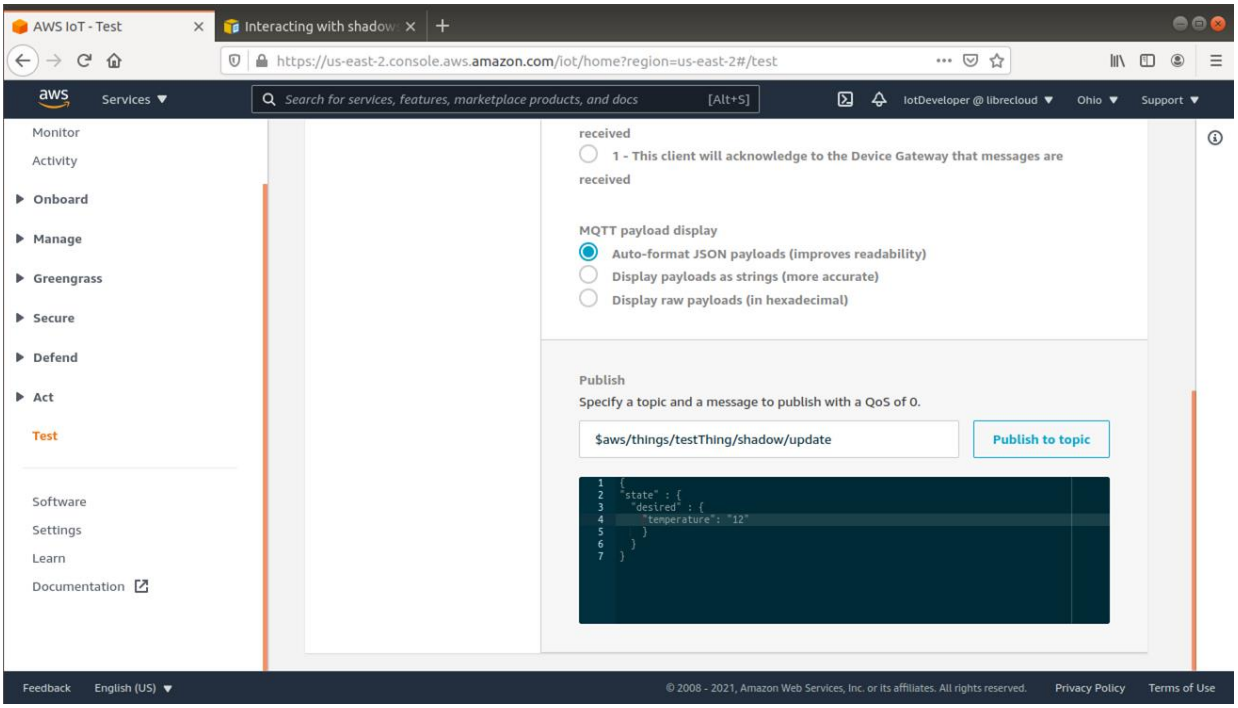
Subscribe:

Run the Application, the below logs will be displayed:



From AWS IoT Console publish the topic using the command given below:

$aws/things/testThing/shadow/update and enter the message format as given below:



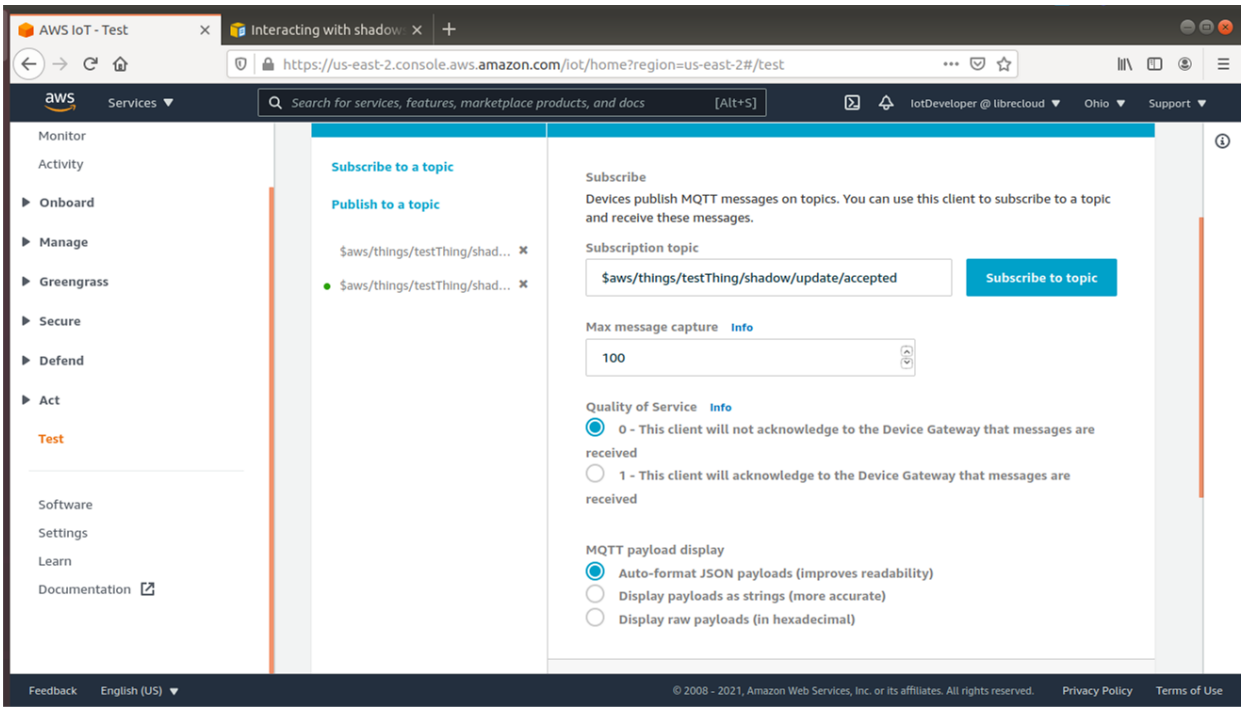
Example 1:

```
{
        “state” : {
                “desired” : {
                        “temprature” : “12”
                        “dooropen” : “false”
                        }
                }
}
```

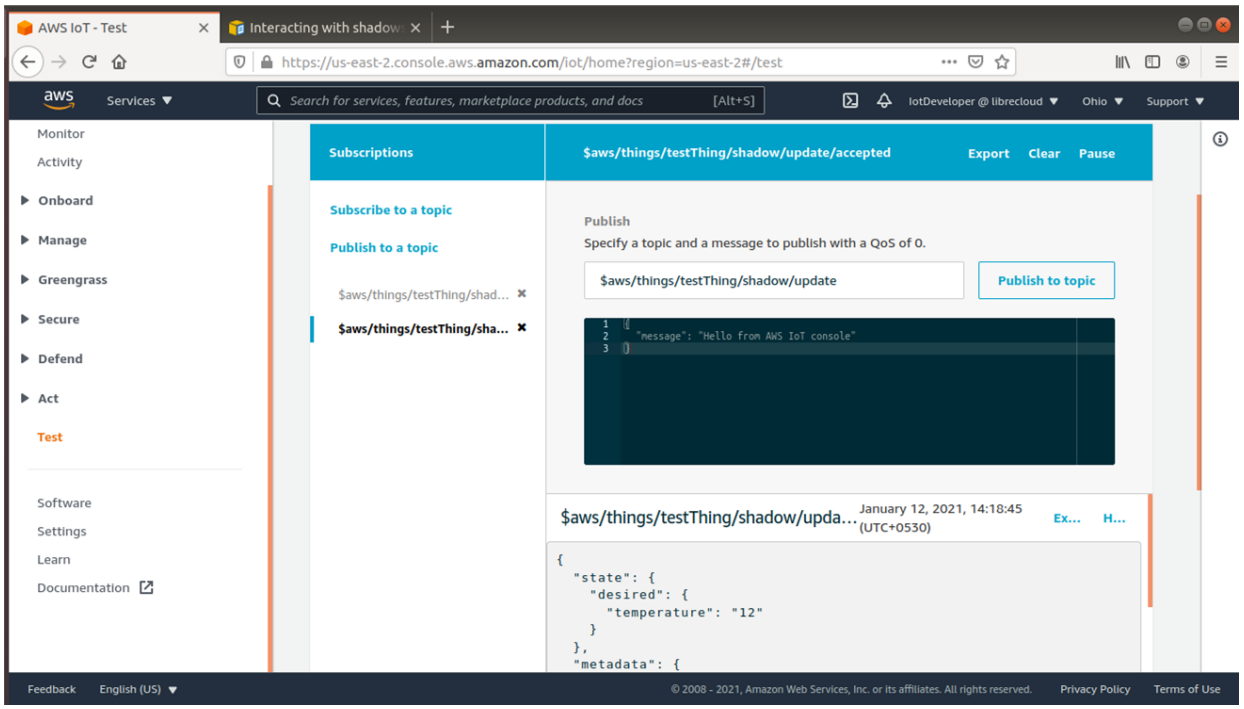
Example 2:

```
{
        "state": {
          "desired": {
              "switch": "on"
           }
   }
}
```

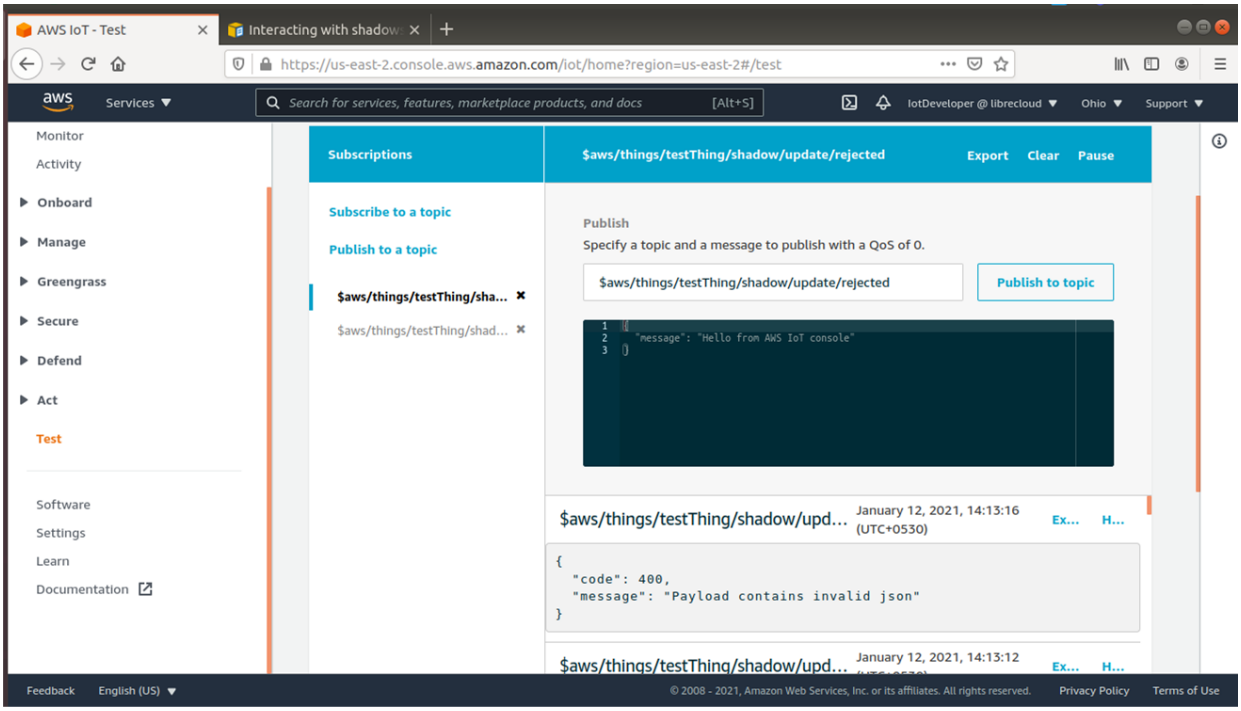
Subscribe the topic from AWS IoT Console side, it will provide the notification about the shadow update accepted or rejected:



The below image indicates how to subscribe update for accepted:



The below image indicates subscribe update for rejected:



3.2.5. LED Control with IoT (aws_Subscribe_led)

This project demonstrates the AWS IoT with LED interface project of MAVID-3M_EVK with AWS IoT APIs and start a service running AWS IOT MQTT client.

This application explain user to how to:

1. Connect to AWS IoT proxy via MQTT over TLS.
2. Publish topic to MQTT server.
3. Subscribe topic to MQTT server.
4. Stop connection, disconnect MQTT.

Find the AWS_iot_Subscribe_led example project at "<sdk_root>/project/aw7698_evk/apps".

Step 1. The program main can be found at "<sdk_root>/project/aw7698_evk/apps/aws_Subscribe_led/src/main.c".

Step 2. Get AWS IOT certificates from AWS IOT Console, then insert the certificates into "<sdk_root>/project/common/certs_protected/aws_iot_cert_rtos.h".

Step 3. To get the AWS IoT certificate follow section 4.

Step 4. Update the endpoint URL, thing name, client ID and topic name to AWS_IOT_MQTT_HOST, AWS_IOT_MY_THING_NAME, INTEGRATION_TEST_CLIENT_ID and INTEGRATION_TEST_TOPIC macros found in <sdk-root>\middleware\third_party\aws_iot\include\aws_iot_config.h.

Step 5. Use **IoT_Client_Init_Params** to initialize MQTT parameters.

Step 6. Use **IoT_Client_Connect_Params** to initialize connection parameters.

Initialize cli task to enable user input cli command from UART port.

Step 7. Follow the procedure to build the project as given in section 2.5.

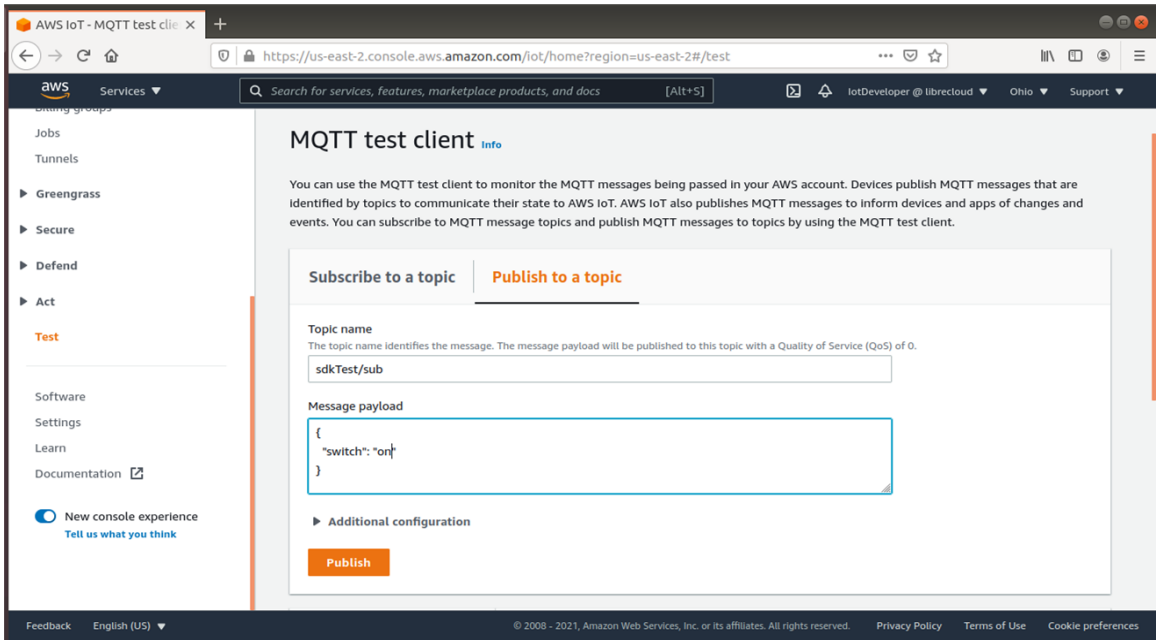
Step 8. Follow the procedure to flash the code on board as given in section 2.6.

Step 9. Follow the procedure to debug and monitor as given in section 2.7.

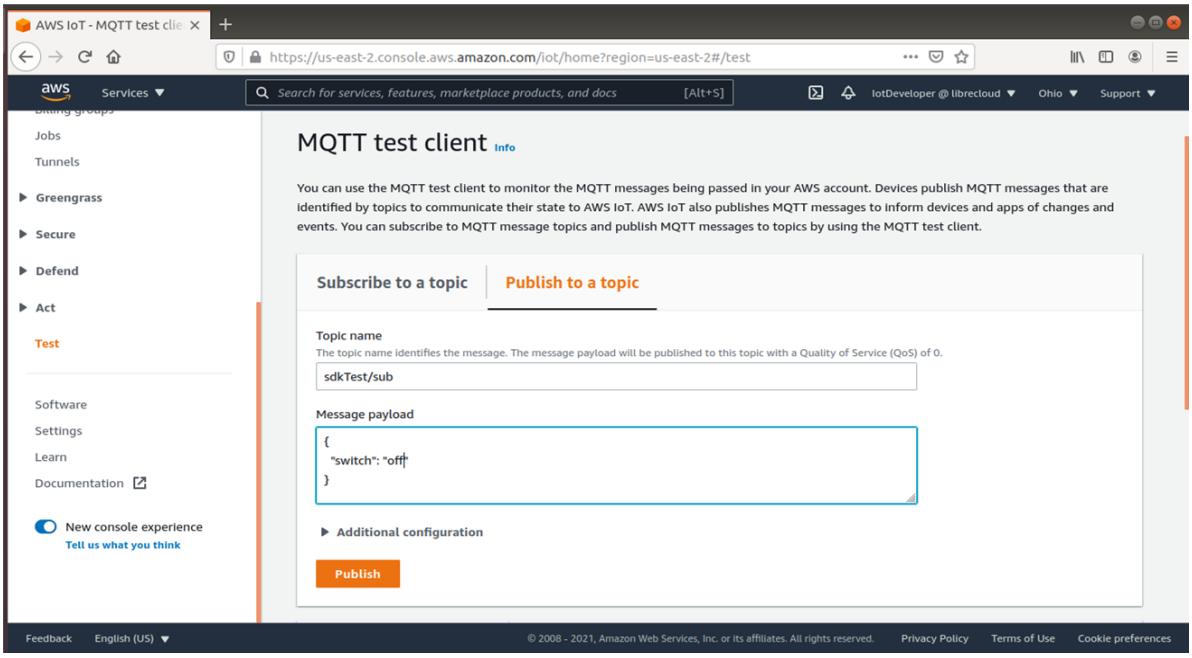
Step 10. Once the device is rebooted, configure Wi-Fi by following the procedure as given in ***Step 11*** of section 3.2.2.

Publish as **INTEGRATION_TEST_TOPIC** for **AWS_IOT_MY_THING_NAME** topic and send a message as a payload to turn ON and turn OFF the LED using AWS IoT console as shown below.

Payload turn ON screen:

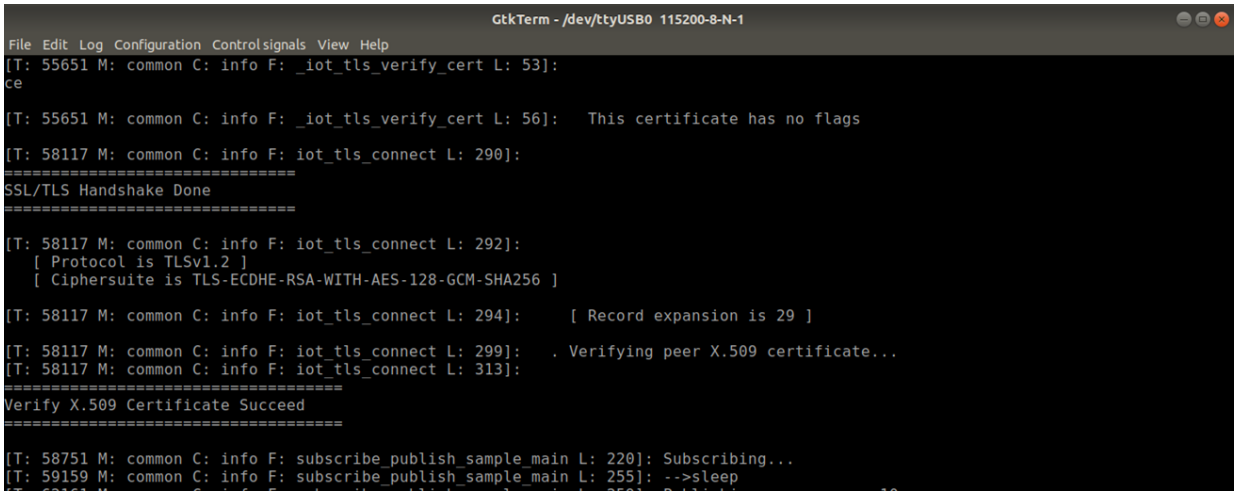


Payload turn OFF screen:

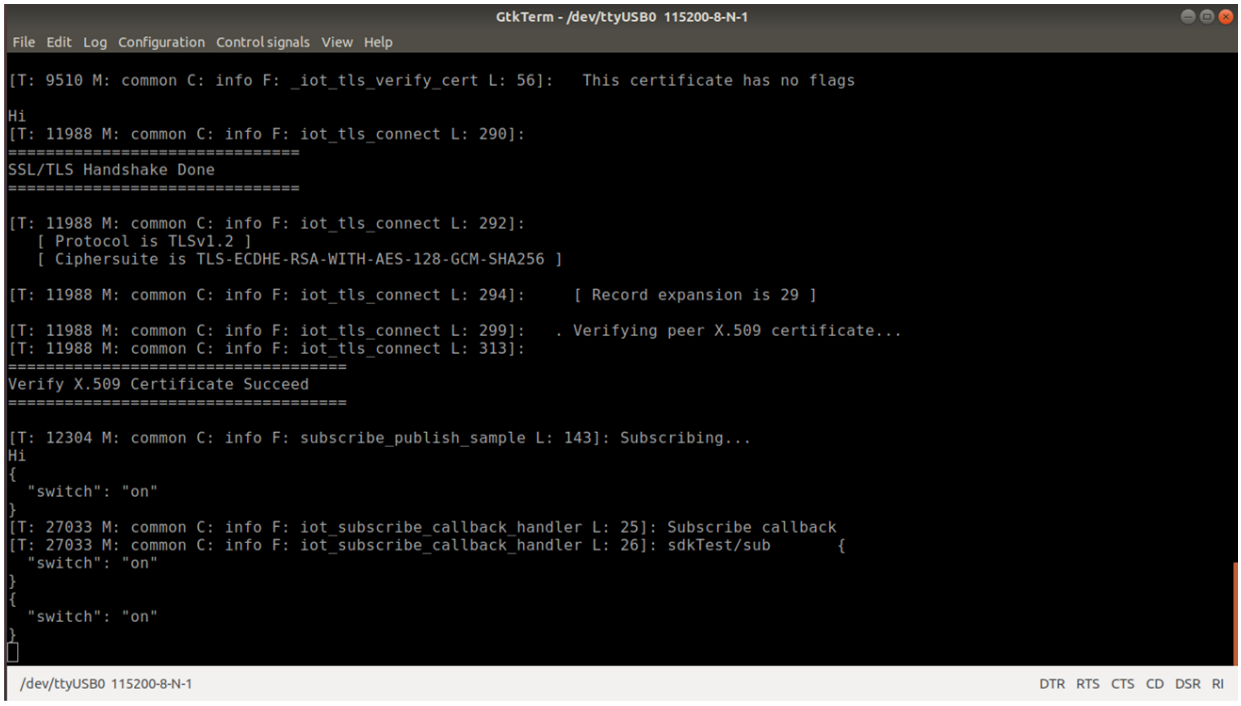


Subscribe the Topic:

Run the Application, the below logs will be displayed:



Final output screen:



3.2.6. Alexa Voice LED Control Example (avs_ledcontrol)

This project demonstrates control of LED using Alexa Voice. It uses Alexa Voice service and AWS Cloud services (IoT Core, Alexa Smart Home Skills, etc.) to control a LED light using voice commands.

Find the **avs_ledcontrol** example project from "<sdk_root>/project/aw7698_evk/apps".

Step 1. The program main can be found at "<sdk_root>/project/aw7698_evk/apps/avs_ledcontrol/src/main.c".

Step 2. Follow section 5 to setup necessary Cloud Infrastructure.

Step 3. Program the device certificates (refer section 5.3 step 4) onto the device as shown in section 4.1.

Step 4. Update the AWS_IOT_MQTT_HOST, AWS_IOT_MY_THING_NAME, and INTEGRATION_TEST_TOPIC macros in <sdk-root>\middleware\third_party\aws_iot\include\aws_iot_config.h, to the values as in Skill Lambda (Section 5.2)

```
#define AWS_IOT_MQTT_HOST             "xxxxxxxxxxxx-ats.iot.us-east-2.amazonaws.com" ///< Customer specific MQTT HOST.
#define AWS_IOT_MY_THING_NAME         "DemoThing"
#define INTEGRATION_TEST_TOPIC        "switch/bulb"
```

Step 5. Use **IoT_Client_Init_Params** to initialize MQTT parameters.

Step 6. Use **IoT_Client_Connect_Params** to initialize connection parameters.

Initialize CLI task to enable user input cli command from UART port.

Step 7. Follow the procedure to build the project as given in section 2.5.

Step 8. Follow the procedure to flash the code on board as given in section 2.6.

Step 9. Follow the procedure to debug and monitor as given in section 2.7.

Step 10. To configure Wi-Fi with mobile application, follow the procedure as given in the section 3.2.2.

Step 11. Login into your Alexa Account and follow the steps to login with Alexa account given in *section 4* in **MAVID-3M EVK - User Guide** document.

Step 12. Once the device is logged in, utter;

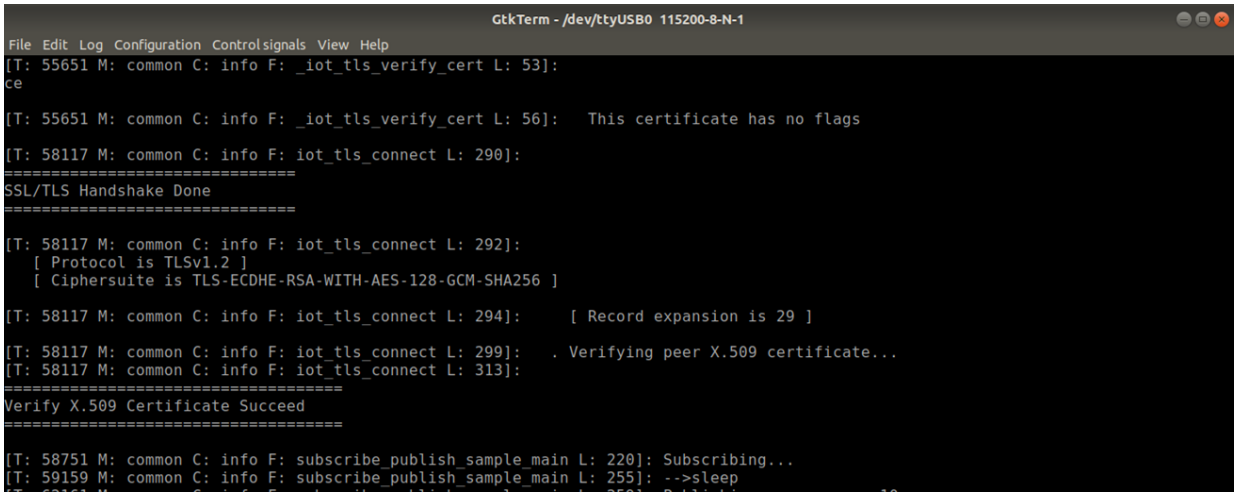
"Alexa, Turn ON LED"

"Alexa, Turn OFF LED"

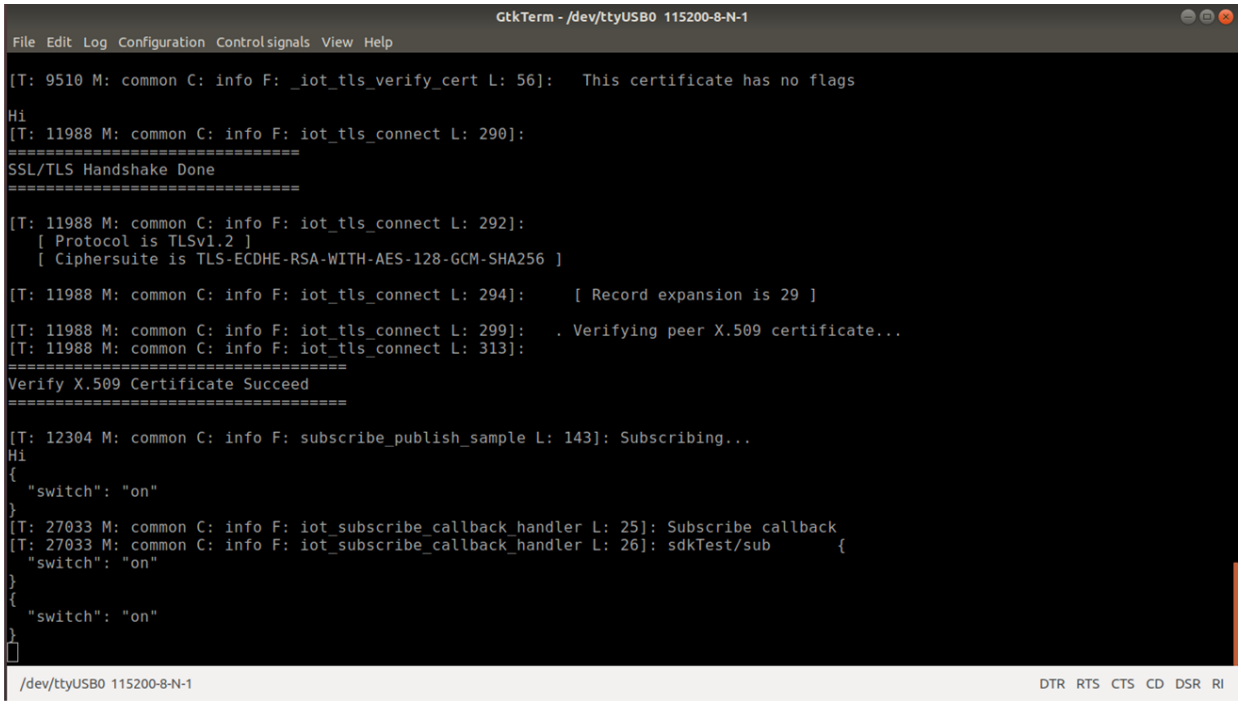
Step 13. To control the LED using voice commands.

Subscribe the Topic:

Run the Application, the below logs will be displayed:



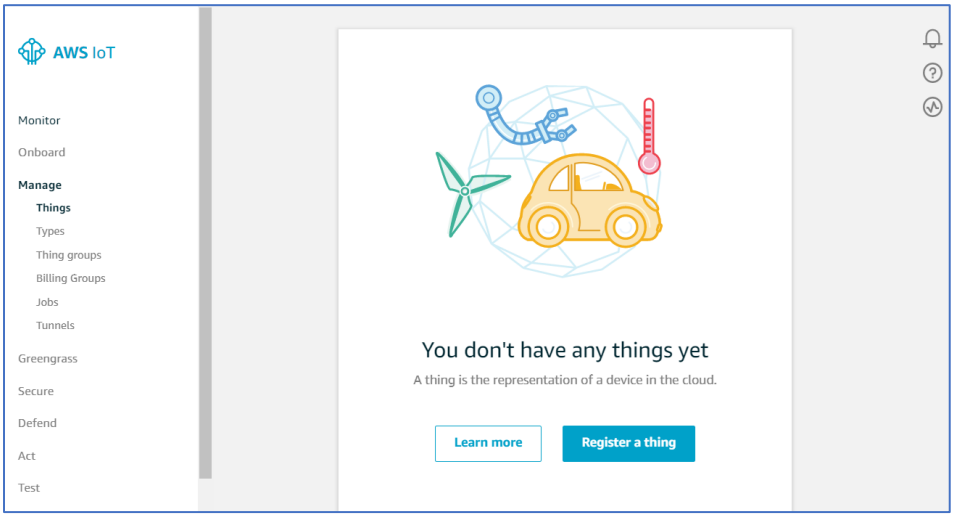
Final output screen:



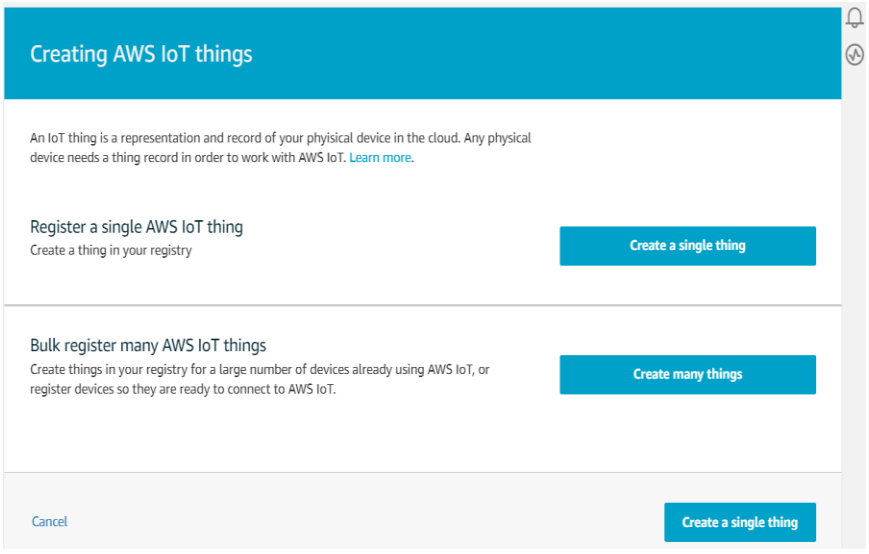
4. IoT Device Provisioning

Follow the steps to provision the device on AWS management console. This includes creating IoT thing, generating certificates and policies, attaching policies and certificates to the IoT thing.

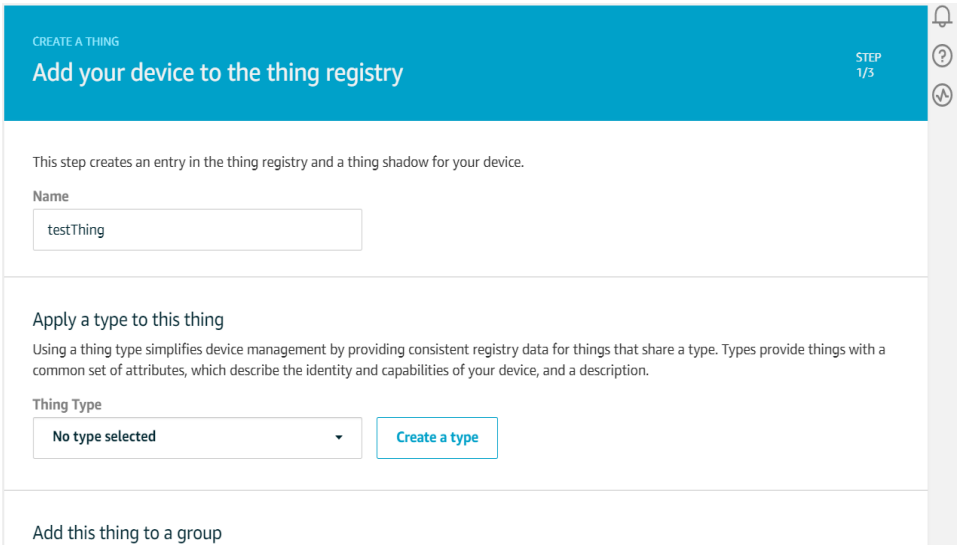
Step 1. Visit AWS IoT Core Console and select **Manage Things** from the left menu, click **Register a thing:**



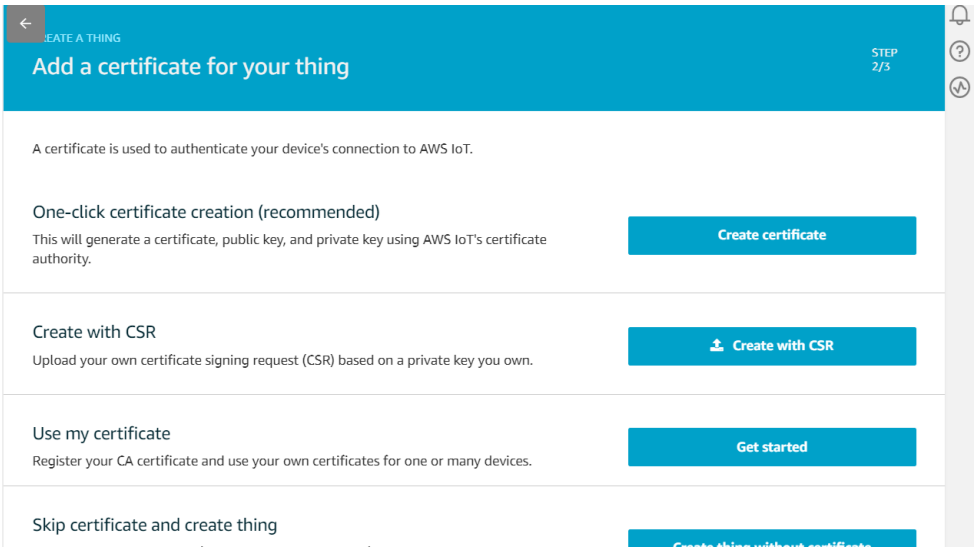
Step 2. Select **Create a single thing**:



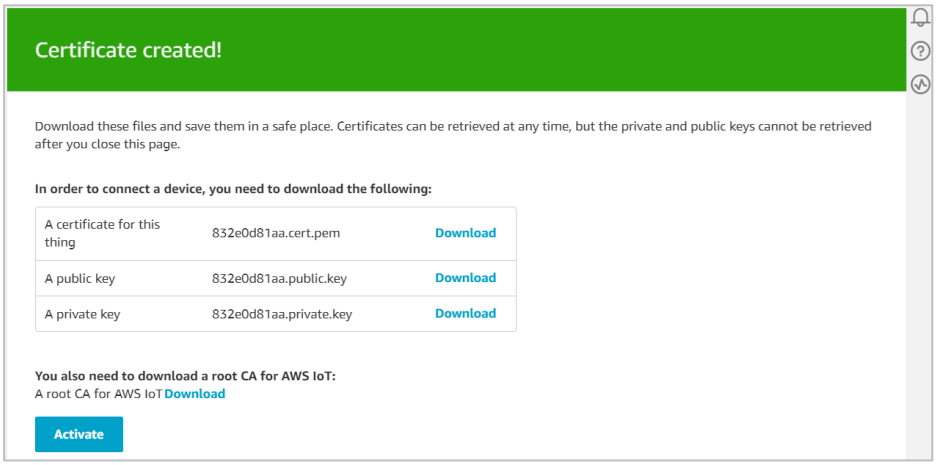
Step 3. Enter your thing's name in the Name field, scroll down and click **Next**:



Step 4. Click **Create certificate:**



Step 5. Download the certificate and both the keys and save them in a safe place. The user must download the private key and public key as they cannot be retrieved after you close this page.



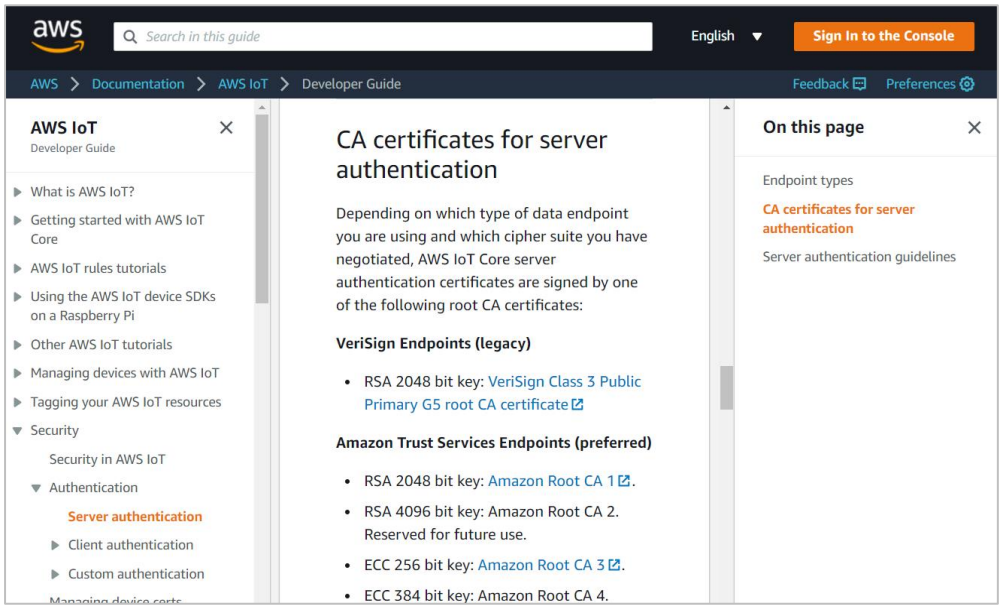
Step 6. Download the root CA.

Step 7. Choose Download and select the appropriate one:

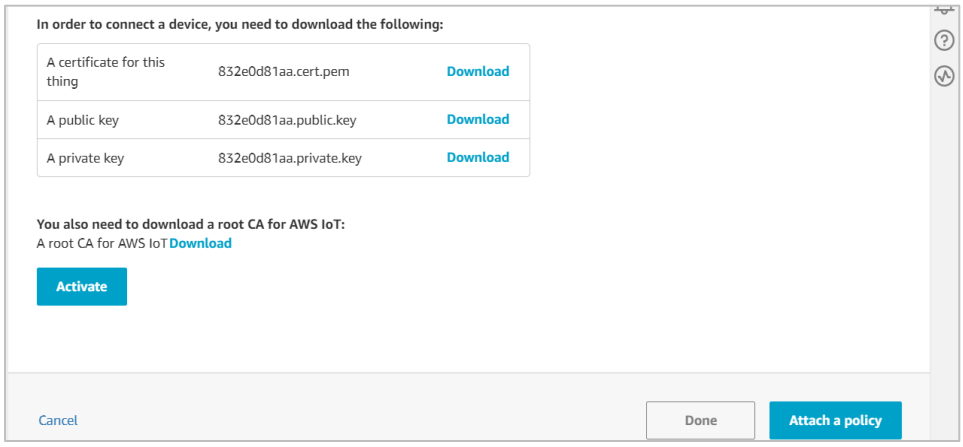
Step 8. RSA 2048 bit key: Amazon Root CA 1 (shown in this example),

OR

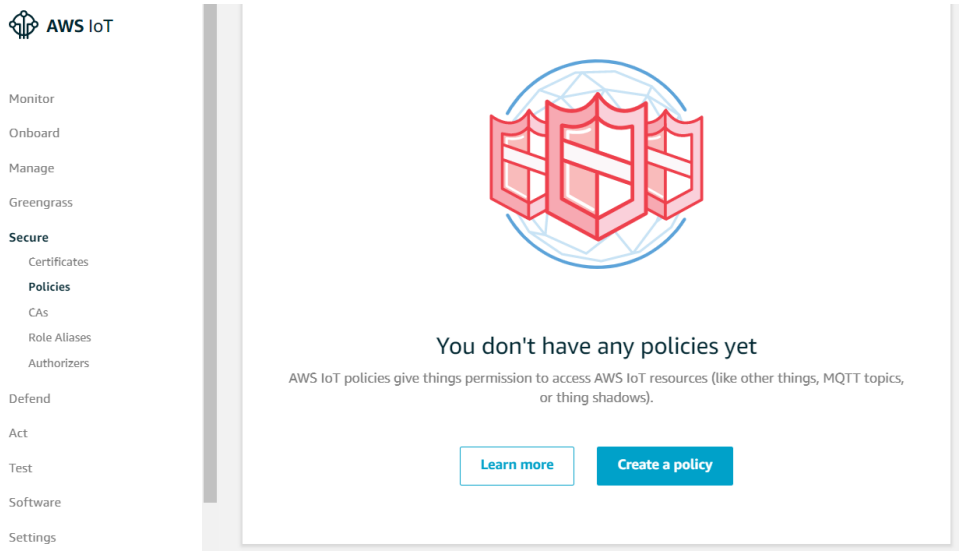
ECC 256 bit key: Amazon Root CA 3



Step 9. After making sure you have downloaded private key and public key (certificate and Amazon root CA), scroll down and click **Activate**, later click **Done** button:

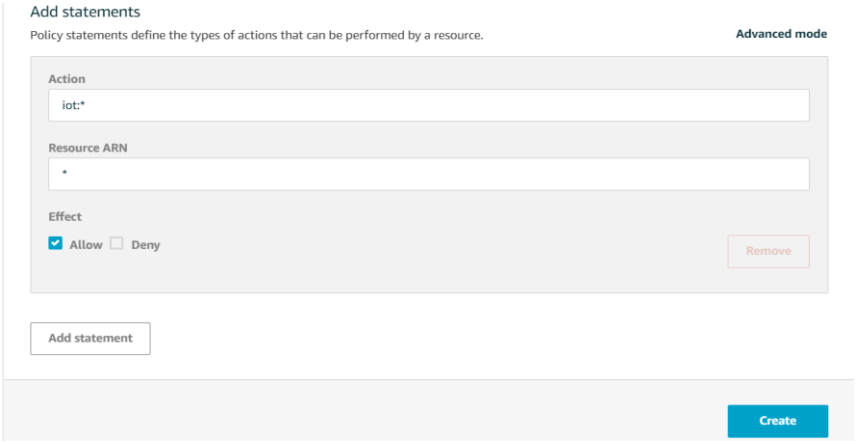


Step 10. Select **Secure Policies** from left menu and click **Create a policy** button.



Step 11. Enter a name for this policy. For example, here we used testPolicy.

Step 12. In Add Statements, choose Advanced mode, then paste the following policy:



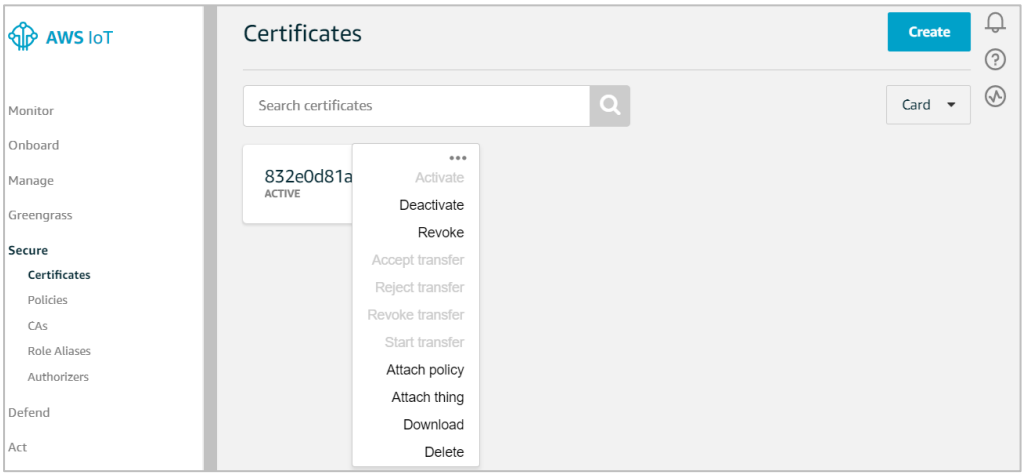

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
    "Effect": "Allow",
    "Action": "iot:*",
    "Resource": "*"
   }
 ]
}
```

This policy grants unrestricted access for all IoT operations and is to be used only in a development environment. For non-dev environments, all devices in your fleet must have credentials with privileges that authorize intended actions only, which include (but not limited to) AWS IoT MQTT actions such as publishing messages or subscribing to topics with specific scope and context. The specific permission policies can vary for your use cases. Identify the permission policies that best meet your business and security requirements. For sample policies, refer to https://docs.aws.amazon.com/iot/latest/developerguide/example-iot-policies.html

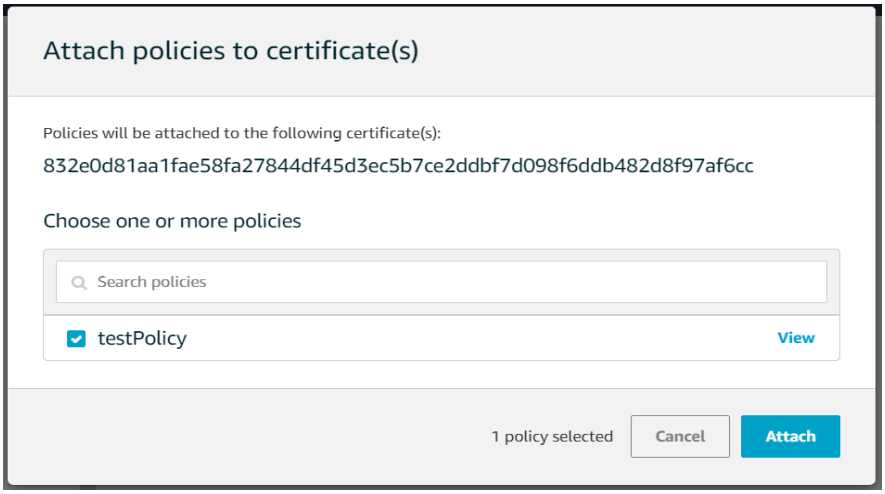
Also refer to https://docs.aws.amazon.com/iot/latest/developerguide/security-best-practices.html

Step 13. Choose Add Statement and then Create.

Step 14. Click **Secure Certificates** from left menu. Click three dots menu on your certificate and click **Attach policy**:



Step 15. Select the testPolicy you just created and click **Attach**:

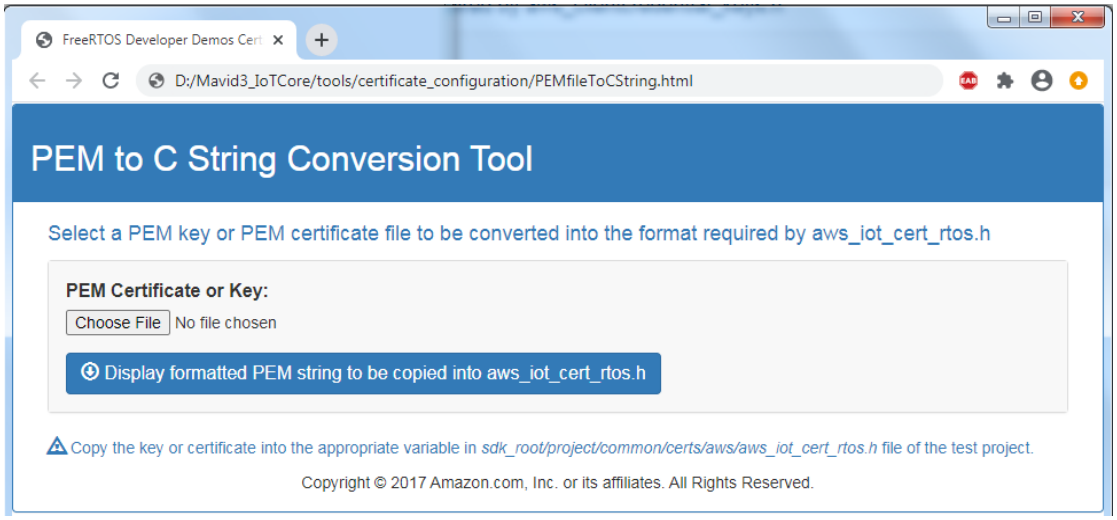


Step 16. Attach the thing to the certificate. Click **Secure Certificates** from left menu then click three dots menu on your certificate, and then click **Attach thing**.

Step 17. Select testThing and click **Attach**. At this point, your device has been provisioned with AWS IoT and can begin to communicate.

4.1. Update Device Certificates

Step 1. Open <sdk_root>\tools\certificate_configuration\PEMfileToCString .html in web browser.



Step 2. Choose the public key pem file downloaded and click on **Display** formatted PEM string to be copied into aws_iot_cert_rtos.h button.

Step 3. Copy the string contents to AWS_IOT_DEVICE_CERT macro in <sdk_root>/project/common/certs/aws/aws_iot_cert_rtos.h file.

```
#define AWS_IOT_DEVICE_CERT     "-----BEGIN CERTIFICATE-----\n"\
"MIIDWjCCAkKgAwIBAgIVAJuanh/8yusM+UiTJU5ZPPbXCF6nMA0GCSqGSIb3DQEB\n"\
"CwUAME0xSzBJBgNVBAsMQkFtYXpvbiBXZWIgU2VydmljZXMgTz1BbWF6b24uY29t\n"\
```

Step 4. Choose the private key pem file downloaded and click on **Display** formatted PEM string to be copied into aws_iot_cert_rtos.h button.

Step 5. Copy the string contents to AWS_IOT_PRIVATE_KEY macro in <sdk_root>/project/common/certs/aws/aws_iot_cert_rtos.h file.

```
#define AWS_IOT_PRIVATE_KEY     "-----BEGIN RSA PRIVATE KEY-----\n"\
"MIIEpAIBAAKCAQEAsYNzQxUFaM6h7oR3k3nRfXU7ZvEGHgsXERa8FbJsmZ/nb+VT\n"\
```

Step 6. Choose the rootCA pem file downloaded and click on **Display** formatted PEM string to be copied into aws_iot_cert_rtos.h button.

Step 7. Copy the string contents to AWS_IOT_ROOT_CA_CERT macro in <sdk_root>/project/common/certs/aws/aws_iot_cert_rtos.h file.

```
#define AWS_IOT_ROOT_CA_CERT    "-----BEGIN CERTIFICATE-----\n"\
"MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF\n"\
"ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6\n"\
"b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL\n"\
```

5. AWS Deployment Services

This section explains the process of setting up AWS cloud infrastructure required for Voice based sample applications. This includes setting up of reference cloud, deployment of a Smart Home skill and configuring the IoT.

Pre-requisites:

- **AWS CDK toolkit**

5.1. Deploying Libre Reference Architecture in AWS

This section walks through the usage of AWS CDK (Cloud Development Kit) for deploying AWS services such as AWS Lambda, AWS IoT Core and AWS Cognito.

Please follow here on how to run **cdworkshop_stack.py** python code given in **<sdk-root>/aws_infrastructure** directory to deploy the necessary infrastructure as per the reference architecture for voice-based sample application provided in MAVID-3M SDK.

Find test.json and my-lambda-handler.zip files from <sdk-root>/aws-infrastructure and add it into cdkworkshop directory, which is created for new project in CDK workshop.Replace cdworkshop_stack.py file with cdworkshop_stack.py file given in <sdk-root>/aws-infrastructure, and

Add this to setup.py file:

```
install_requires=[
    "aws-cdk.core==1.107.0",
    "aws-cdk.aws_iam==1.107.0",
    "aws-cdk.aws_sqs==1.107.0",
    "aws-cdk.aws_sns==1.107.0",
    "aws-cdk.aws_sns_subscriptions==1.107.0",
    "aws-cdk.aws_s3==1.107.0",
```

//lines to Add

"aws-cdk.aws_iot==1.107.0",

"aws-cdk.aws_cognito==1.107.0",

"aws-cdk.aws_dynamodb==1.107.0",

"aws-cdk.aws_lambda==1.107.0",

"aws-cdk.aws_apigateway==1.107.0",

]

On successful deployment, user should be able to see the below AWS services deployed in his/her account:

Step 1. AWS Lambda service with reference code:

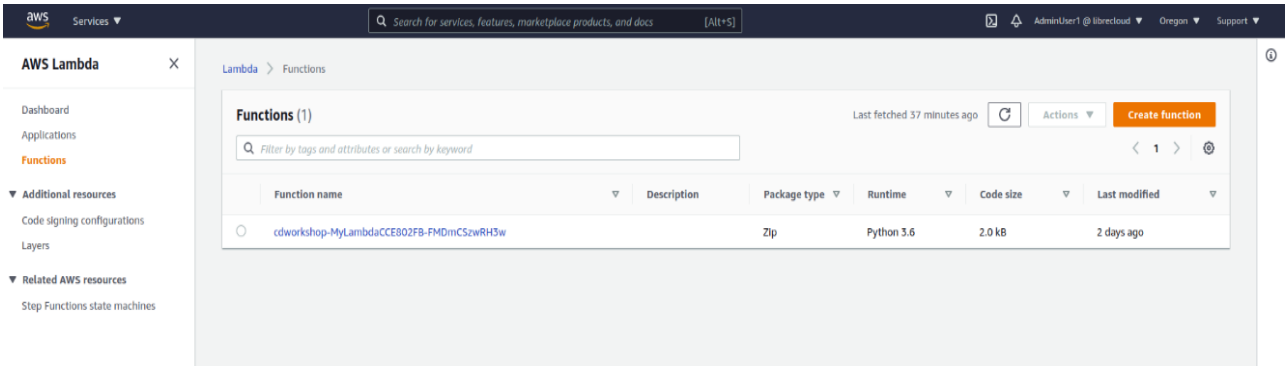


Figure 5.1-1: AWS Lambda

Step 2. AWS Cognito User pool:

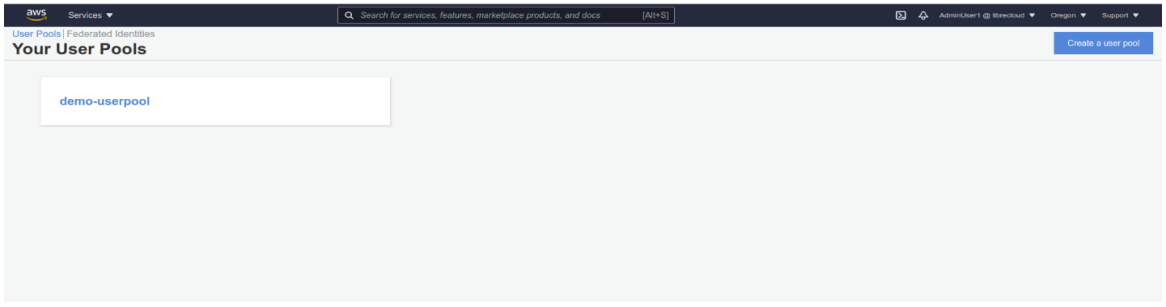


Figure 5.1-2: AWS Cognito

Step 3. AWS IoT Things and Things policy:

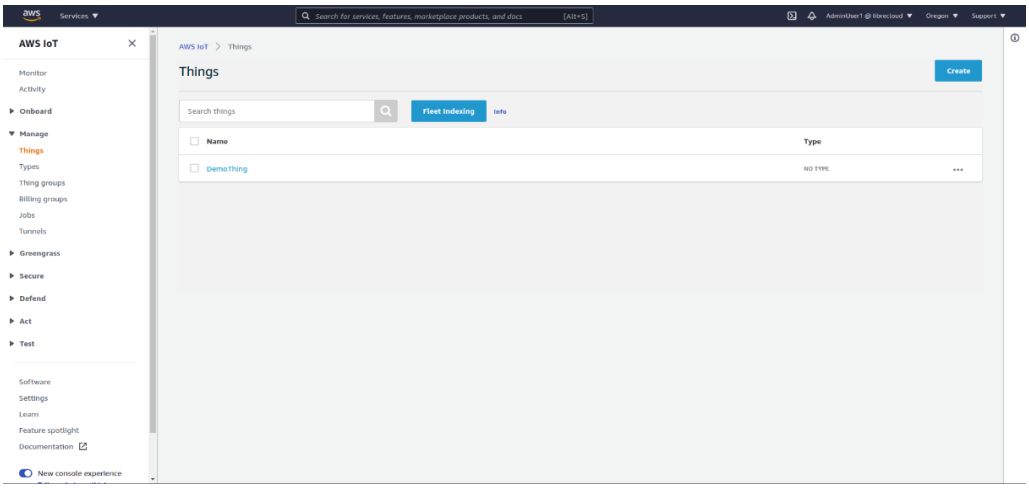


Figure 5.1-3: AWS IoT Things

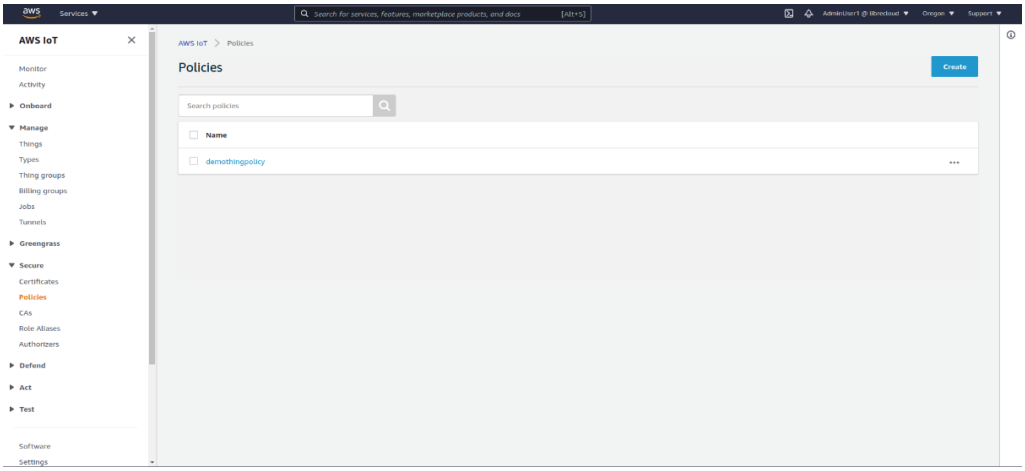


Figure 5.1-4: AWS IoT Policies

5.2. Configuration of Smart Home Skill

a) To create smart home skill, follow here

b) Configuring Smart Home Skill:

Step 1. Copy the Smart home Skill ID:

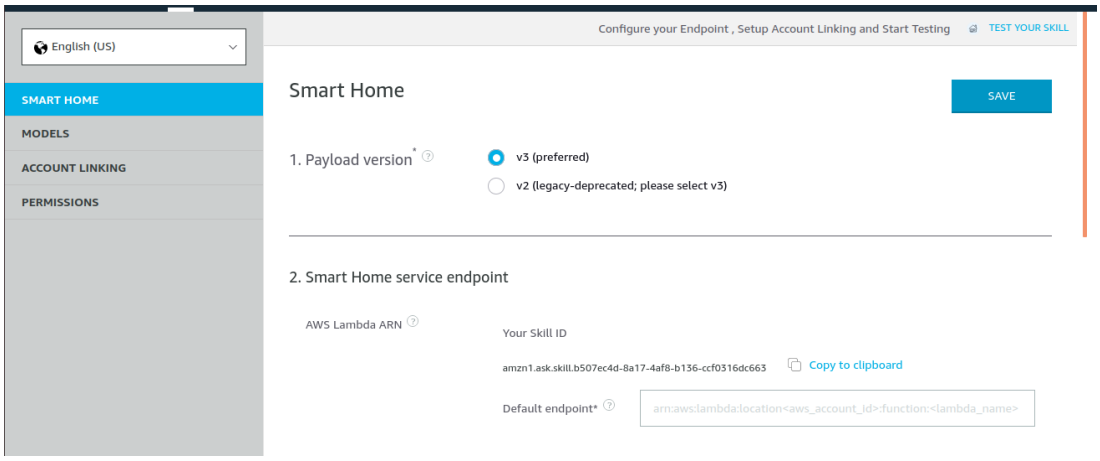


Figure 5.2-1: Smart Home Screen

Step 2. From the Smart Home Skill Console, Select ACCOUNT LINKING.

Under Security Provider Information, copy Alexa Redirect URLs:

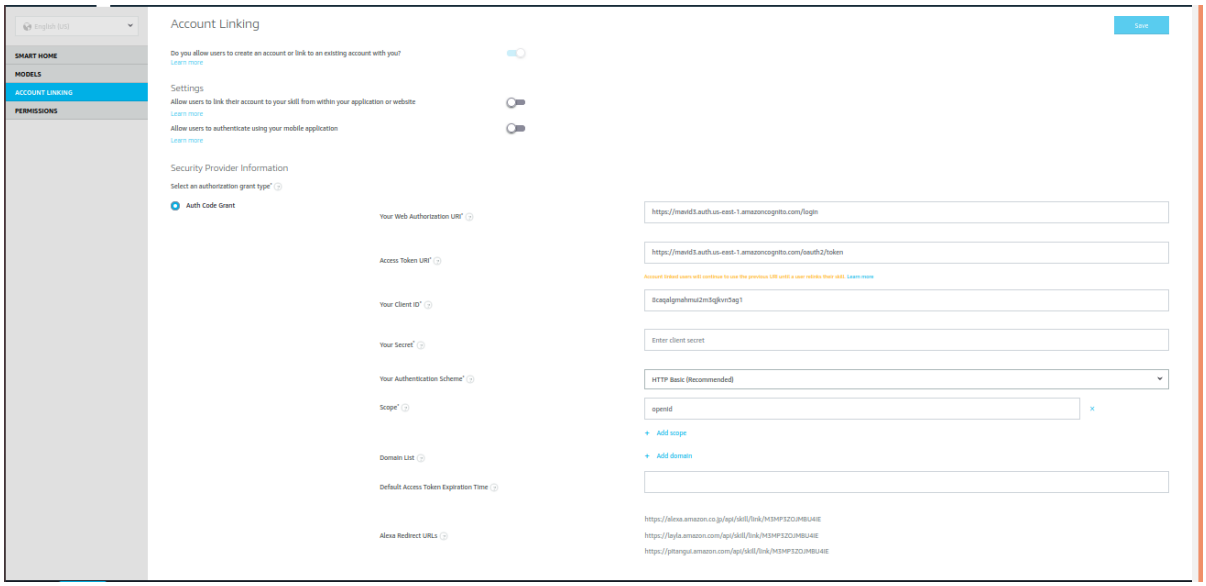


Figure 5.2-2: Smart Home Skill Console

Step 3. Navigate to AWS Lambda. Click **Add Trigger**:

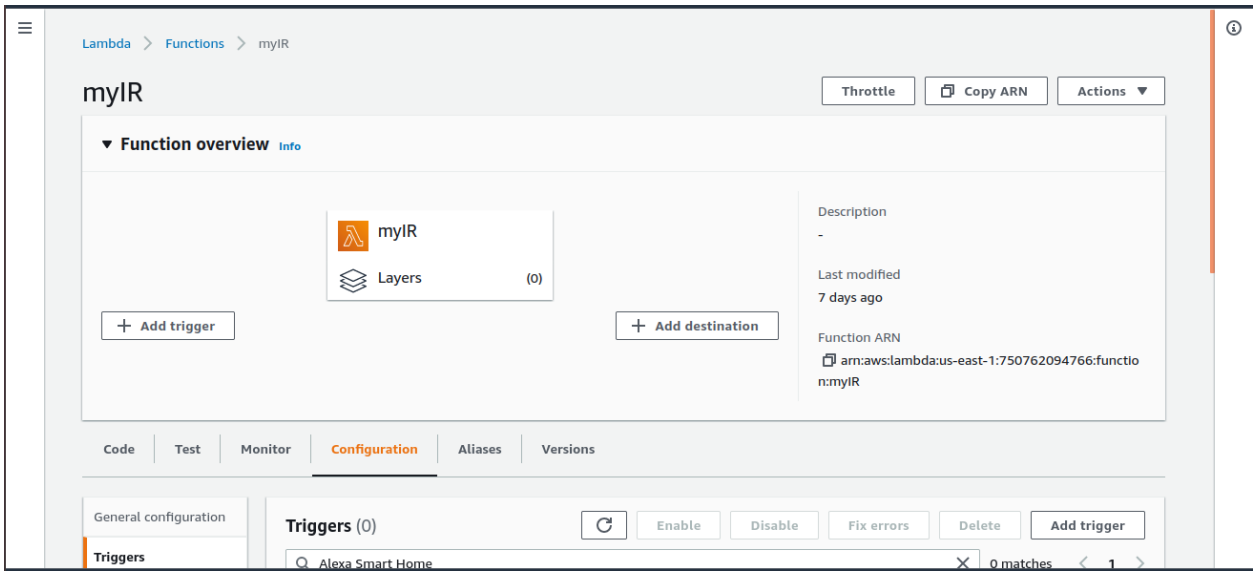


Figure 5.2-3: AWS Lambda

Step 4. Select Alexa smart home kit form the drop down:

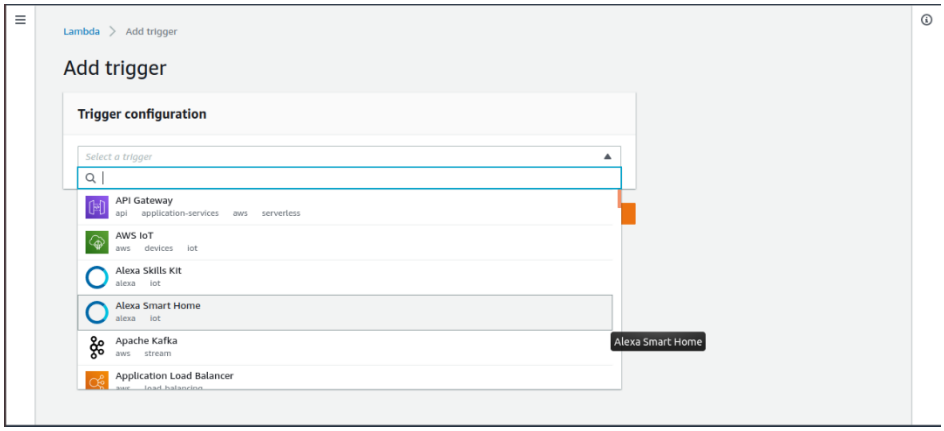


Figure 5.2-4: Add Trigger Screen

Step 5. Paste the skill ID in the Application ID and click **ADD**:

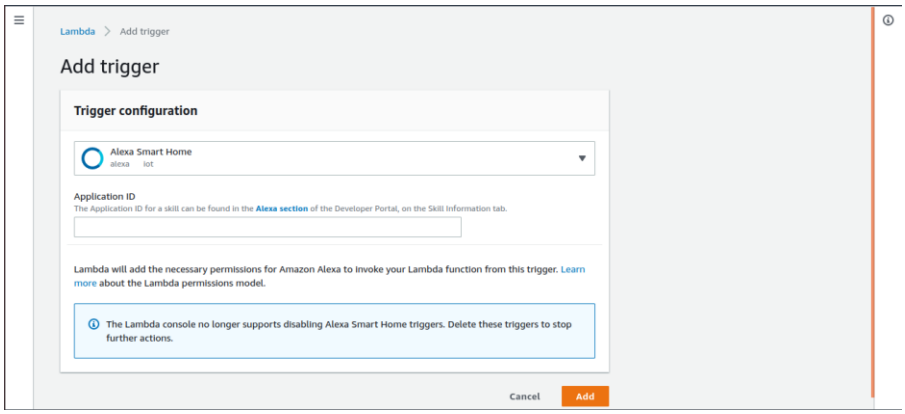


Figure 5.2-5: Add Trigger Screen

Step 6. Navigate to AWS Cognito, select Domain Name from the APP Integration menu. Copy Amazon Cognito domain:

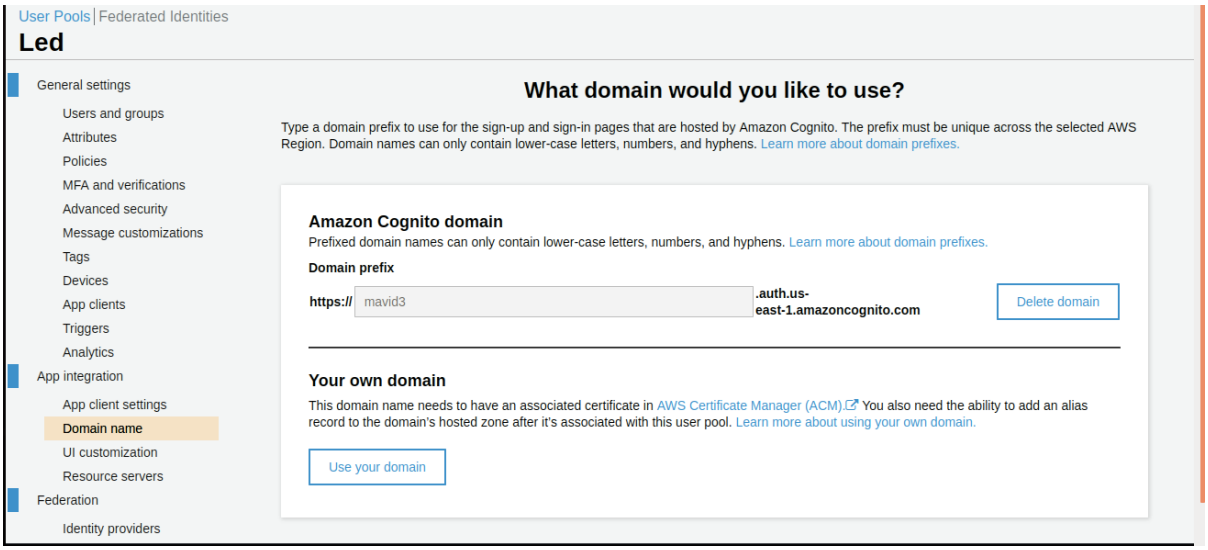


Figure 5.2-6: AWS Cognito Screen

Step 7. Navigate to AWS Cognito, select App Clients from General settings menu. Copy App Client Id and App Client Secret:

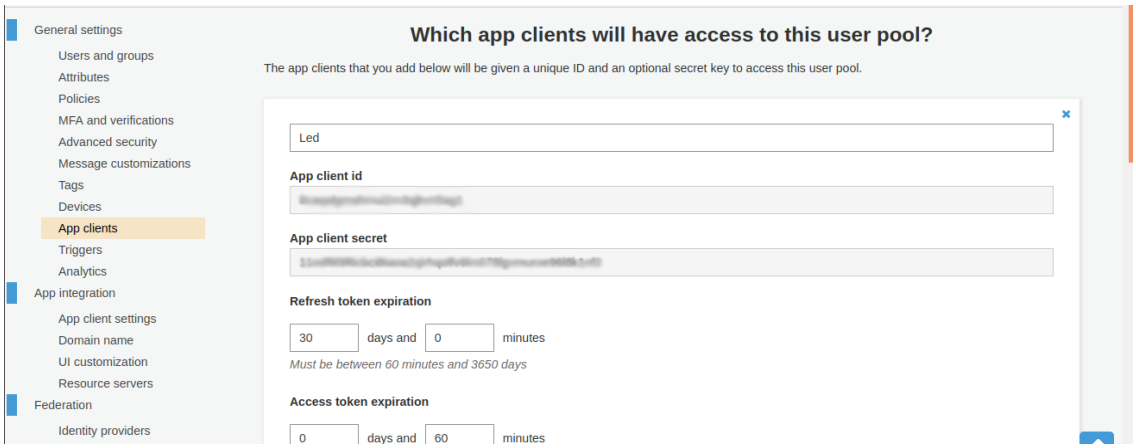


Figure 5.2-7: AWS Cognito Screen

Step 8. Navigate to AWS Cognito, select App Client settings menu from App Integration. Under callback URLs paste Alexa Redirect URL's from step 2 with comma separation between the URLs.

In Signout URLs: Append /logout to the domain name that was copied in step 6

Example: https://mavid3.auth.us-east-1.amazoncognito.com/logout

Under Allowed OAuth Scope tick/select the openid checkbox:

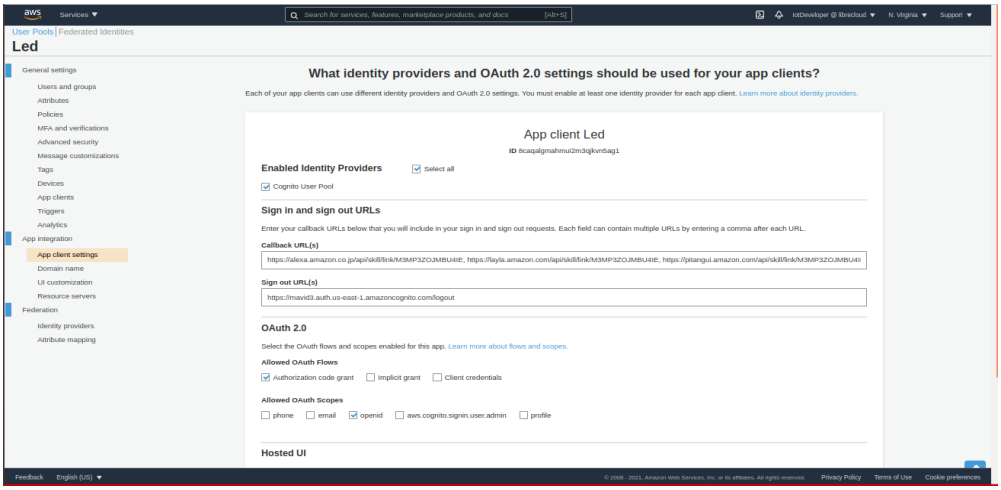


Figure 5.2-8: App Client Settings

Step 9. From the Smart Home Skill Console, select ACCOUNT LINKING.

In Security Provider Information fill the following details:

a) Your Web Authorization URI

Append/login to the domain name that was copied in step 5

Example: https://mavid3.auth.us-east-1.amazoncognito.com/login

b) Access Token URI

Append /oauth2/token to the domain name that was copied in step 5

Example: https://mavid3.auth.us-east-1.amazoncognito.com/oauth2/token

c) Your Client ID

Copy the Client ID that was copied in step 5

d) Your Secret

Copy the Client Secret that was copied in step 5

e) Scope

Give outh2 scope as openid

Fill the above details as shown below:

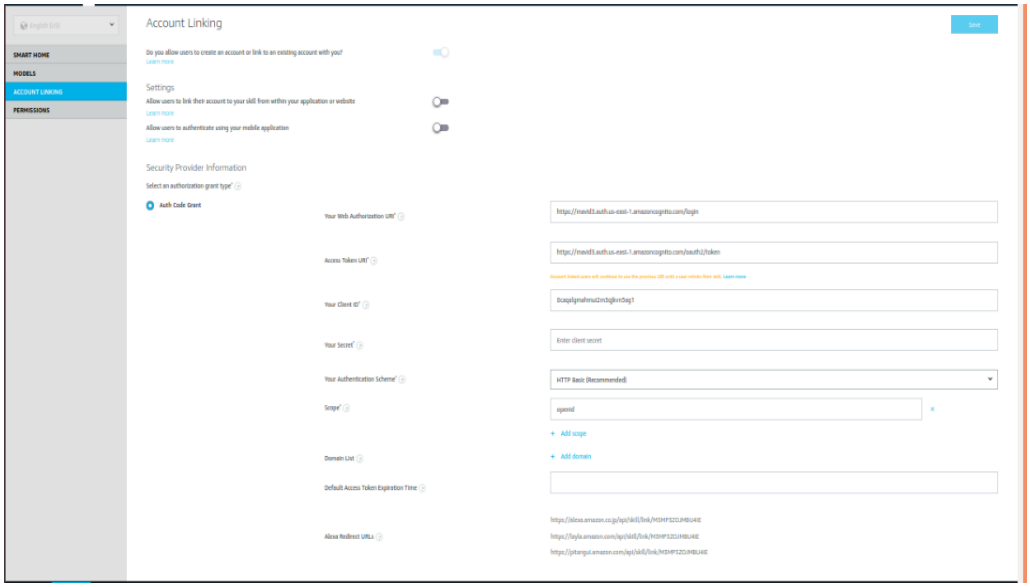


Figure 5.2-9: Account Linking

5.3. Configuring Device to AWS IoT Core

Step 1. Login to user AWS account and navigate to AWS IoT core.

Step 2. Click **Manage** and select DemoThing.

Step 3. On the left Tab select Security.

Step 4. Click on **Create Certificate** and download certificates.

Step 5. Click **Activate** and **Attach a policy** respectively:

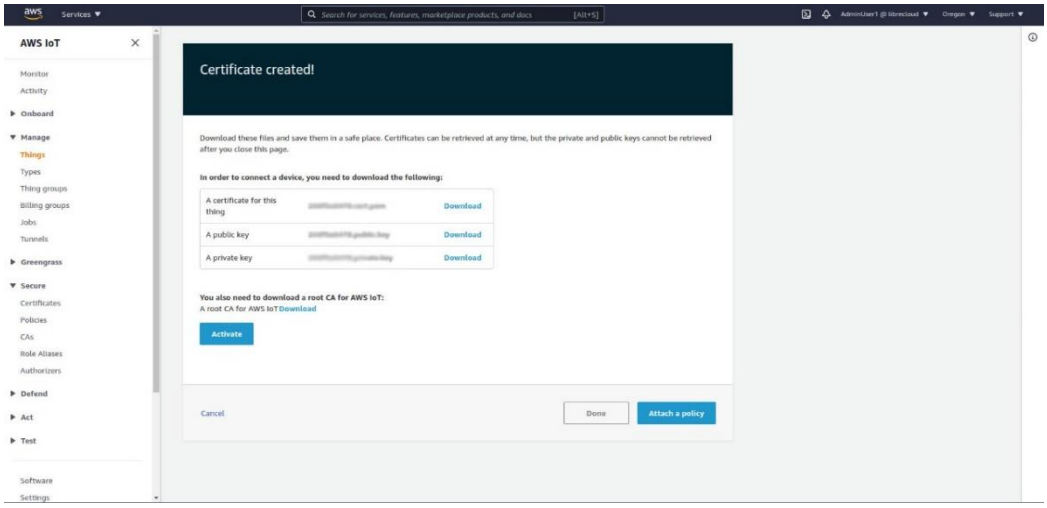


Figure 5.3-1: Attach a Policy

Step 6. Select demothingpolicy and click **Done**:

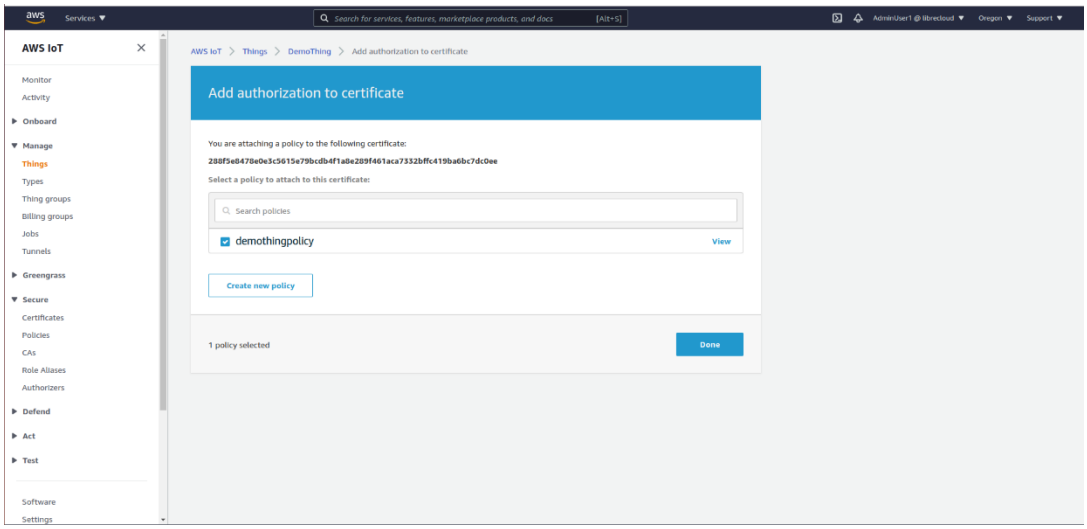


Figure 5.3-2: Add Authorization