­­­

|  |  |
| --- | --- |
| Azure\_iot\_kit\_on\_CEC1x02DevBoard | |
| User manual | |
| Rev 04 | 03/14/2018 |

|  |
| --- |
| Copyright © 2018 Microchip or its subsidiaries. All rights reserved.  The information contained herein is confidential and proprietary to Microchip, shall be used solely in accordance with the agreement pursuant to which it is provided, and shall not be reproduced or disclosed to others without the prior written consent of Microchip. Although the information is believed to be accurate, no responsibility is assumed for inaccuracies. Microchip reserves the right to make changes to this document and to specifications and product descriptions at any time without notice. Neither the provision of this information nor the sale of the described semiconductor devices conveys any licenses under any patent rights or other intellectual property rights of Microchip or others. The product may contain design defects or errors known as anomalies, including but not necessarily limited to any which may be identified in this document, which may cause the product to deviate from published specifications. Microchip 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 Microchip will be fully at the risk of the customer.    MICROCHIP 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 SMSC 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 SMSC 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 SMSC HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. |

Table of Contents

[1. Introduction 3](#_Toc508795689)

[2. Prerequisites 4](#_Toc508795690)

[2.1 Development Environment 4](#_Toc508795691)

[2.2 Hardware components 4](#_Toc508795692)

[2.3 Bootrom revision 4](#_Toc508795693)

[3. Executing the demo 5](#_Toc508795694)

[3.1 Enroll device using Test UDS Certificate and debug 5](#_Toc508795695)

[3.1.1 Prepare Device 6](#_Toc508795696)

[3.1.2 Build and Run the sample 7](#_Toc508795697)

[3.1.3 Connect and send messages to Azure IoT Hub 10](#_Toc508795698)

[3.1.4 Receive messages from Azure IoT Hub 16](#_Toc508795699)

[3.2 Enroll device using Efuse UDS Certificate and debug 18](#_Toc508795700)

[3.2.1 UDS Generation 18](#_Toc508795701)

[3.2.2 Efuse memory map file generation 19](#_Toc508795702)

[3.2.3 Efuse programming using MPLABX and ICD4 19](#_Toc508795703)

[3.2.4 Run the demo using ICD4 using efuse UDS 20](#_Toc508795704)

[3.2.5 Deriving the X.509 certificate and the thumbprint 20](#_Toc508795705)

[3.3 Enroll device using Efuse UDS Certificate and program SPI image 21](#_Toc508795706)

[3.4 Deriving the X.509 certificate and the thumbprint 22](#_Toc508795707)

[4. Debugging connection failure 24](#_Toc508795708)

[5. Next Steps 25](#_Toc508795709)

# Introduction

Microchip CEC1x02 IoT Starter Kit is an CEC1702 (32-bit ARM® Cortex®-M4 Controller with Integrated Crypto Accelerators) based development kit that integrates LCD display, Serial Quad I/O (SQI) flash, ATWINC1510 Clicker and Thermo5 Clicker. Using the CEC1x02 Development Board you can evaluate and program keys used for authentication.

**About this document**

This document describes how to connect Microchip CEC1x02 board along with Wifi Clicker Board (ATWINC1500) to Microsoft Azure IoT Hub, by using the X.509 based MQTT application sample, based on Azure IoT Device SDK. Also we will make use of Azure DPS (Device Provisioning Service) to register and provision our device. This multi-step process includes:

* Configuring Azure IoT Hub
* Provisioning your CEC1x02 device using Azure Device Provisioning Service (DPS)
* Build and deploy Azure IoT SDK on CEC1x02 Development Board
* Send and Receive messages from Cloud

# Prerequisites

You should have the following items ready before beginning the process

## Development Environment

* ***MPLAB X IDE*** - For project build, downloading and debugging code
* **xc32 compiler** – For compiling and linking the code
* ***mikroProg Suite For ARM*** – For efuse programming and loading spi image in flash
* ***ICD4 debugger –*** For code debugging
* ***ComXDBG.exe*** (provided with package) or any other serial terminal installed in your PC – for viewing trace messages from CEC1x02 Devboard
* [Configure cloud resource for device provisioning](https://docs.microsoft.com/en-us/azure/iot-dps/tutorial-set-up-cloud)

## Hardware components

The Kit contents are as follows:

* CEC1x02 Development Board
* ATWINC1510 Clicker Board
* Thermo5 Click Board
* mikroProg for CEC

## Bootrom revision

The CEC1702 device should have B0\_340 revision of bootrom. This can be verified by calling the ROM API rom\_version() in the code and ensuring the return value is 0x00400232

## 

# Executing the demo

The CEC1x02 DevBoard has CEC1702 device which **NOT** programmed with efuse UDS during production and **are blank**. User need to program the euse memory to use the Azure DPS demo. There are 3 ways to run the demo which is explained below. **It is highly recommended** to follow the below sequence **when using the CEC1x02 DevBoard for the first time** to successfully test Azure DPS demo

* Enroll device using Test UDS certificate and run the demo using ICD4 debugger
* Enroll device using Efuse UDS certificate and run the demo using ICD4 debugger
* Enroll device using Efuse UDS certificate and run the demo using SPI image programmed on the on-board spi flash

## Enroll device using Test UDS Certificate and debug

CEC1702 uses X.509 based authentication for connecting to DPS and IoT Hub. The X.509 certificate is generated for the device through DICE and RIoT process. See [Device Identity with DICE and RIoT: Keys and Certificates](https://www.microsoft.com/en-us/research/publication/device-identity-dice-riot-keys-certificates/) for more details.

In the current DICE-RIoT implementation; the DICE, RIoT and Azure application code are built together as one firmware image; they are organized separately in memory by using linker options.

Please refer [IoT Hub DPS Security Concepts](https://docs.microsoft.com/en-us/azure/iot-dps/concepts-security) and for more details on X.509 security in DPS.

For testing purpose the UDS (Unique Device Secret) for the device uses a test value. Later when you want to use the UDS value from efuse, you can uncomment the macro USE\_TEST\_UDS in dps\_hsm\_riot.c. See section 3.5 for more details.

Follow the below steps to provision your device in DPS using the test alias certificate.

1. Download the [devBoard\_Azure\_IoT\_build](http://www.microchip.com/SWLibraryWeb/product.aspx?product=CEC1702_AZURE_IOT) package. This package contains all the required drivers to use with the CEC1x02 Development board with Winc1500 clicker board, together with pre-integrated Microsoft Azure IoT C SDK.
2. The test alias certificate for the device is provided in the package as test\_alias\_certificate.pem

dice\_RIoT\_mplabx/test\_alias\_certificate.pem

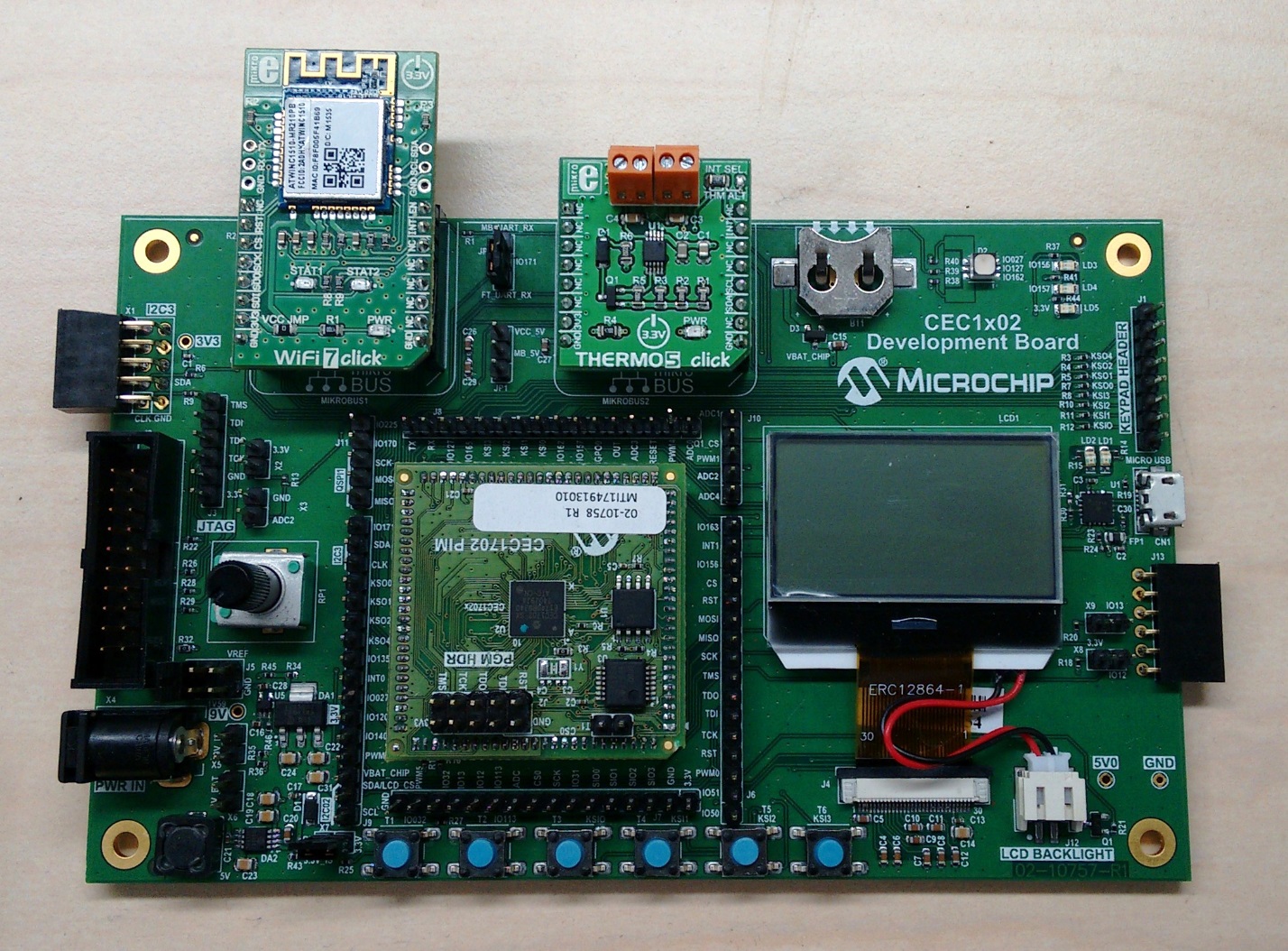
1. Provision this alias certificate .pem file in your DPS dashboard in Azure Portal

See [Provision device to Hub tutorial](https://docs.microsoft.com/en-us/azure/iot-dps/tutorial-provision-device-to-hub) for provisioning your device using enrollment lists. Follow instructions for X.509 based devices using Individual Enrollment.

## Prepare Device

On the CEC1x02 Development Board attach the ATWINC1510 Clicker Board and Thermo5 clicker board as follows:

* ATWINC1510 Clicker Board on mikroBUS slot 1
* Thermo5 Clicker Board on mikroBUS slot 2



## Build and Run the sample

1. Download the [devBoard\_Azure\_IoT\_build](http://www.microchip.com/SWLibraryWeb/product.aspx?product=CEC1702_AZURE_IOT) package. This package contains all the required drivers to use with the CEC1x02 devBoard board with Winc1500 clicker board, together with pre-integrated Microsoft Azure IoT C SDK.

The project uses mbedTLS as the TLS stack, which has been added to the project as a library:

Src/APP/framework/mbedTLS/mbedtls.X/dist/default/production/mbedtls.X.a

1. Extract the contents of the downloaded zip file: **CEC1702\_azure\_mplabx\_build\_0100.zip**

Clear the ‘Readonly’ attribute for the project folder

* + Right click the project folder and Select ‘Properties’
  + In the General tab, clear the Read-Only attribute and select Apply
  + Select ‘*Apply changes to this folder, subfolder and files*’ and click OK

1. Open MPAB X Project

* In MPLAB X IDE, Select ‘Open Project’ from the File Menu
* Navigate to the downloaded CEC1702 DICE-RIoT project and open the MPLABX project **devBoard\_diceRIoT\_MPLABX.X**

1. The DPS client application is **prov\_dev\_client\_ll\_sample.c.** It uses HTTPS as the transport for communicating to DPS server and MQTT as transport for communicating with AZURE IOT hub.

Src\APP\apps\ prov\_dev\_client\_ll\_sample.c

The certificates and keys required for authentication is generated through DICE (Device Identification Composition Engine)

For testing purpose the UDS (Unique Device Secret) for the device, uses a test value hardcoded in dps\_hsm\_riot.c

static unsigned char g\_uds\_seed[DICE\_UDS\_LENGTH] = {

0x54, 0x10, 0x5D, 0x2E, 0xCD, 0x07, 0xF9, 0x01,

0x99, 0xB3, 0x95, 0xC7, 0x42, 0x61, 0xA0, 0x8C,

0xFF, 0x27, 0x1A, 0x0D, 0xF6, 0x6F, 0x1F, 0xE0,

0x00, 0x34, 0xBB, 0x11, 0xF7, 0x98, 0x9A, 0x12 };

Src\APP\framework\azure\dps\_client\adapters\dps\_hsm\_riot.c

Later when you want to use the UDS value from efuse, you can uncomment the macro USE\_TEST\_UDS in dps\_hsm\_riot.c.

See CEC1702.ld in project folder for the linker file memory layouts

devBoard\_diceRIoT\_MPLABX.X/CEC1702.ld

1. Update your Device Provisioning Service ID Scope in **prov\_dev\_client\_ll\_sample.c** line 37

For example, if Device Provisioning Service ID is 0ne000000A9

static const char\* id\_scope = "0ne000000A9";

The ID scope value is used to identify Registration IDs. This value is retrieved from your DPS dashboard in Azure portal.

1. Wifi Configuration - Currently the code is configured to connect through WPA-PSK. The SSID and password are set statically in ***winc1500\_connect.c***

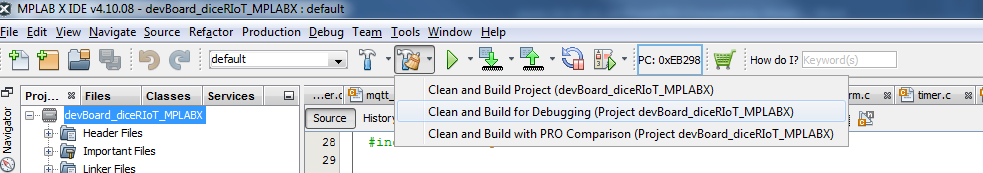
Src\APP\platform\winc\winc1500\_connect.c

#define CONN\_SSID "TP-LINK\_9746"

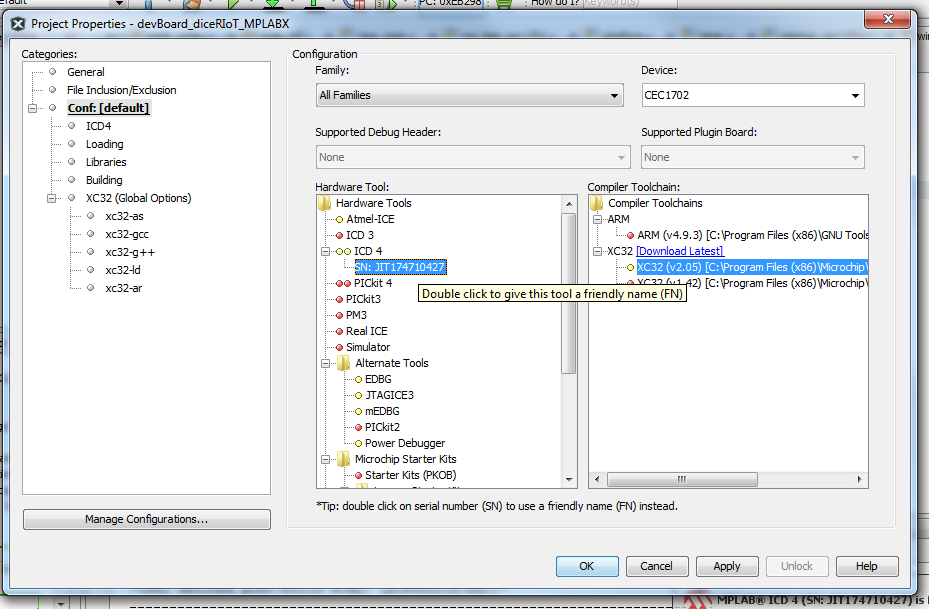
#define CONN\_PSK\_PWD "69651946"

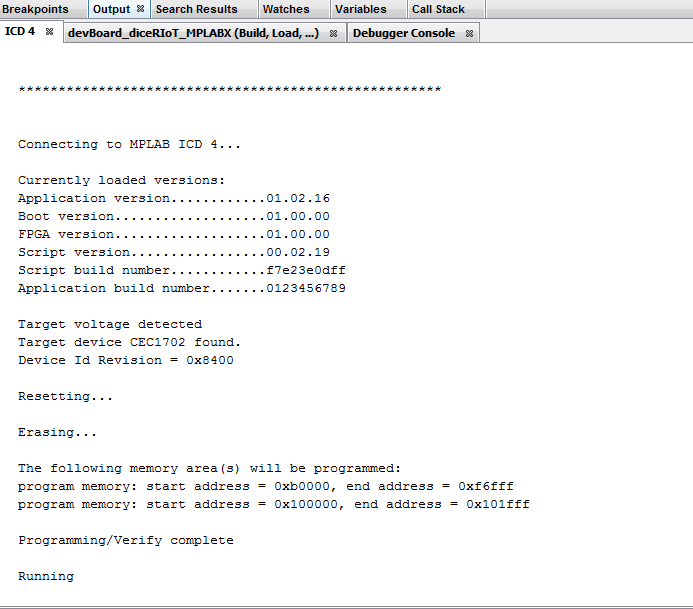
Modify the above values to match your wifi router.

1. Select ‘Clean and Build for Debugging’ option



1. Connect the ICD4 using JTAG cable and power up the board. Next select *Debug* and Click Debug Project. MPLABX would detect the ICD4 automatically and start the debug session





## Connect and send messages to Azure IoT Hub

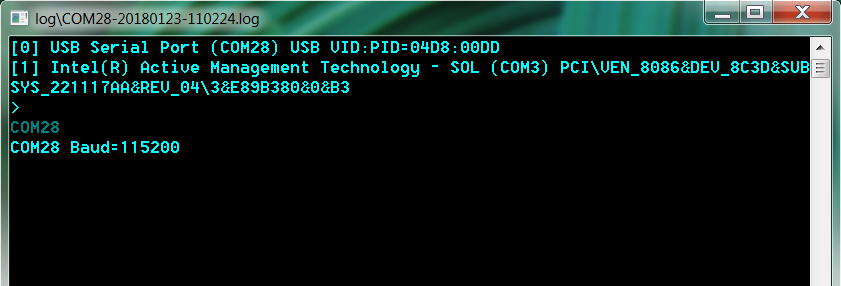
To view the UART traces, it is recommended to download and install [MCP2221 driver for USB to UART](http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2221%20Windows%20Driver%202014-10-09.zip)

Follow the below steps to use ComXDBG.exe

1. connect USB cable between devBoard board and Windows host
2. After driver installation, start ComXDBG.exe

utilities\ ComEDBG \ ComXDBG.exe

1. Select port with **VID:PID** as **04D8:00DD**. For example; for the displayed options (in below snapshot) we would enter 0



1. You should be able to view UART traces from the devBoard board

Alternatively, you can use your serial terminal (e.g. TeraTerm for Windows) with the following parameters:

* BaudRate : 115200
* Data : 8-bit
* Parity : None
* Stop : 1 bit
* Flow Control : None

Sample log

[12:54:21.565] AZURE IoT Node

[12:54:21.565] Microchip CEC1702 devBoard 01

[12:54:21.565] Firmware Version : CEC1702\_devBoard\_diceRIOT\_MPLABX\_build\_0600

[12:54:21.565] Mar 9 2018 12:41:14

[12:54:21.565] ------------------------------------------------

[12:54:22.513] winc1500\_wifi\_cb: 2c

[12:54:22.513] M2M\_WIFI\_RESP\_CON\_STATE\_CHANGED: CONNECTED

[12:54:23.025] winc1500\_wifi\_cb: 32

[12:54:23.025] M2M\_WIFI\_REQ\_DHCP\_CONF: IP is 192.168.0.101

[12:54:23.076] WINC is connected to TP-LINK\_9746 successfully!

[12:54:23.076] winc1500\_wifi\_init: Done

[12:54:23.076] Initializing rando.

[12:54:23.076] DPS\_HSM: hsm\_client\_x509\_init:

[12:54:23.076] Provisioning API Version: 1.1.01

[12:54:23.076] Iothub API Version: 1.1.27

[12:54:23.076]

[12:54:23.076] RIOT: CDI

[12:54:23.076] ce1c95020a4902a87fde4dbf8977b68362a944eac301c4c6c25828c92bd80780

[12:54:24.455]

[12:54:24.455] RIOT: Device Certificate

[12:54:24.455] -----BEGIN CERTIFICATE-----

[12:54:24.507] 

[12:54:24.507] -----END CERTIFICATE-----

[12:54:24.507]

[12:54:24.507] RIOT: Alias Certificate

[12:54:24.507] -----BEGIN CERTIFICATE-----

[12:54:24.559] 

[12:54:24.559] -----END CERTIFICATE-----

[12:54:24.559]

[12:54:24.559] RIOT: Alias key

[12:54:24.560] -----BEGIN EC PRIVATE KEY-----

[12:54:24.611] MHcCAQEEINXd3WSS7LqDEFGdpbQi0V51w01XkpMVPUYlG9V7/ldUoAoGCCqGSM49AwEHoUQDQgAEXl8k/JM5Trrd+T2hCzH8J8FzdMo+nqEBctNE0wUQWesM+QbsF1PBmEHxb/FpqegLlleJW4niMxaN8z3J7T/imA==

[12:54:24.611] -----END EC PRIVATE KEY-----

[12:54:24.611]

[12:54:24.611] tlsio\_mbedtls\_create: host:global.azure-devices-provisioning.net port:443

[12:54:24.611]

[12:54:24.611] socketio\_create:

[12:54:24.611] TLSIO MBEDTLS: setoption: ignore\_server\_name\_check

[12:54:24.611] TLSIO MBEDTLS: setoption: x509EccCertificate

[12:54:24.611] TLSIO MBEDTLS: setoption: x509EccAliasKey

[12:54:24.663] TLSIO MBEDTLS: setoption: TrustedCerts

[12:54:24.663]

[12:54:24.663] tlsio\_mbedtls\_open:

[12:54:24.663]

[12:54:24.663] socketio\_open:

[12:54:24.663] winc1500\_connect:

[12:54:24.714] winc1500\_wifi\_cb: 20

[12:54:25.026] SERVER IP is 52.163.212.39

[12:54:25.126] winc1500\_connect: socket: 0

[12:54:28.453] mbedtls\_connect: sts 0

[12:54:28.453]

[12:54:28.453] indicate\_open\_complete:

[12:54:28.453] Provisioning Status: PROV\_DEVICE\_REG\_STATUS\_CONNECTED

[12:54:34.705]

[12:54:34.705] PUT /0ne000000A5/registrations/riot-device-test-cert/register?api-version=2017-08-31-preview HTTP/1.1

[12:54:34.705] UserAgent: prov\_device\_client/1.0

[12:54:34.705] Accept: application/json

[12:54:34.705] Connection: keep-alive

[12:54:34.705] Content-Type: application/json; charset=utf-8

[12:54:34.705] Host: global.azure-devices-provisioning.net:443

[12:54:34.705] Content-Length: 44

[12:54:34.705]

[12:54:34.705]

[12:54:34.816]

[12:54:34.816] len: 44

[12:54:34.816] .\*s

[12:54:37.218]

[12:54:37.218] HTTP Status: 202

[12:54:37.218]

[12:54:37.218] Date: Fri, 09 Mar 2018 07:24:35 GMT

[12:54:37.218]

[12:54:37.269] Content-Type: application/json; charset=utf-8

[12:54:37.269]

[12:54:37.269] Transfer-Encoding: chunked

[12:54:37.269]

[12:54:37.269] Location: https://global.azure-devices-provisioning.net:443/0ne000000A5/registrations/riot-device-test-cert/register

[12:54:37.269]

[12:54:37.269] x-ms-request-id: fe7ba465-cf81-48f7-804e-5613eca39add

[12:54:37.269]

[12:54:37.269] Strict-Transport-Security: max-age=31536000; includeSubDomains

[12:54:37.269]

[12:54:37.269] .\*s

[12:54:39.298] Provisioning Status: PROV\_DEVICE\_REG\_STATUS\_ASSIGNING

[12:54:41.421]

[12:54:41.471] GET /0ne000000A5/registrations/riot-device-test-cert/operations/2.8efab11d183aaab6.d8771ecf-445c-46b9-a752-97dd79d47b09?api-version=2017-08-31-preview HTTP/1.1

[12:54:41.471] UserAgent: prov\_device\_client/1.0

[12:54:41.471] Accept: application/json

[12:54:41.471] Connection: keep-alive

[12:54:41.471] Content-Type: application/json; charset=utf-8

[12:54:41.471] Host: global.azure-devices-provisioning.net:443

[12:54:41.471] Content-Length: 0

[12:54:41.471]

[12:54:41.471]

[12:54:42.689]

[12:54:42.689] HTTP Status: 200

[12:54:42.689]

[12:54:42.689] Date: Fri, 09 Mar 2018 07:24:40 GMT

[12:54:42.689]

[12:54:42.689] Content-Type: application/json; charset=utf-8

[12:54:42.689]

[12:54:42.689] Transfer-Encoding: chunked

[12:54:42.689]

[12:54:42.689] x-ms-request-id: b25085bb-6860-4c9e-8e71-4b6ba3d507c4

[12:54:42.689]

[12:54:42.689] Strict-Transport-Security: max-age=31536000; includeSubDomains

[12:54:42.689]

[12:54:42.689] .\*s

[12:54:44.736] Registration Information received from service: azure-iothub-mchp-ny-1.azure-devices.net!

[12:54:44.801] winc1500\_close: socket: 0

[12:54:44.852]

[12:54:44.852] RIOT: CDI

[12:54:44.852] ce1c95020a4902a87fde4dbf8977b68362a944eac301c4c6c25828c92bd80780

[12:54:46.207]

[12:54:46.207] RIOT: Device Certificate

[12:54:46.207] -----BEGIN CERTIFICATE-----

[12:54:46.262] 

[12:54:46.262] -----END CERTIFICATE-----

[12:54:46.262]

[12:54:46.262] IoTHubTransport\_MQTT\_Common\_SetOption: option logtrace

[12:54:46.262] IoTHubTransport\_MQTT\_Common\_SetOption: option TrustedCerts

[12:54:46.262] tlsio\_mbedtls\_create: host:azure-iothub-mchp-ny-1.azure-devices.net port:8883

[12:54:46.262]

[12:54:46.262] socketio\_create:

[12:54:46.281]

[12:54:46.281] IoTHubClient\_Auth\_Set\_xio\_Certificate

[12:54:46.281]

[12:54:46.281] RIOT: Alias Certificate

[12:54:46.281] -----BEGIN CERTIFICATE-----

[12:54:46.386] 

[12:54:46.386] -----END CERTIFICATE-----

[12:54:46.386]

[12:54:46.386] RIOT: Alias key

[12:54:46.386] -----BEGIN EC PRIVATE KEY-----

[12:54:46.386] MHcCAQEEINXd3WSS7LqDEFGdpbQi0V51w01XkpMVPUYlG9V7/ldUoAoGCCqGSM49AwEHoUQDQgAEXl8k/JM5Trrd+T2hCzH8J8FzdMo+nqEBctNE0wUQWesM+QbsF1PBmEHxb/FpqegLlleJW4niMxaN8z3J7T/imA==

[12:54:46.386] -----END EC PRIVATE KEY-----

[12:54:46.386] TLSIO MBEDTLS: setoption: x509EccCertificate

[12:54:46.386] TLSIO MBEDTLS: setoption: x509EccAliasKey

[12:54:46.405] TLSIO MBEDTLS: setoption: TrustedCerts

[12:54:46.458] Sending 1 messages to IoTHub every 2 seconds for 5 messages (Send any message to stop)

[12:54:46.458]

[12:54:46.458] tlsio\_mbedtls\_open:

[12:54:46.458]

[12:54:46.458] socketio\_open:

[12:54:46.458] winc1500\_connect:

[12:54:46.771] SERVER IP is 40.76.71.185

[12:54:47.032] winc1500\_connect: socket: 0

[12:54:50.809] mbedtls\_connect: sts 0

[12:54:50.809]

[12:54:50.809] indicate\_open\_complete:

[12:54:51.064]

[12:54:51.064] -> CONNECT | VER: 4 | KEEPALIVE: 25 | FLAGS: zu | USERNAME: azure-iothub-mchp-ny-1.azure-devices.net/riot-device-test/api-version=2016-11-14&DeviceClientType=iothubclient%2f1.1.27%20(CEC1702) | CLEAN: 0

[12:54:52.134]

[12:54:52.134] <- CONNACK | SESSION\_PRESENT: true | RETURN\_CODE: 0x0

[12:54:54.245]

[12:54:54.295] IoTHubClient\_LL\_SendEventAsync accepted message [1] for transmission to IoT Hub.

[12:54:54.497]

[12:54:54.497] -> SUBSCRIBE | PACKET\_ID: 2 | TOPIC\_NAME: devices/riot-device-test/messages/devicebound/# | QOS: 1

[12:54:55.556]

[12:54:55.556] <- SUBACK | PACKET\_ID: 2 | RETURN\_CODE: 1

[12:54:57.687]

[12:54:57.687] IoTHubClient\_LL\_SendEventAsync accepted message [2] for transmission to IoT Hub.

[12:55:00.161]

[12:55:00.466] -> PUBLISH | IS\_DUP: false | RETAIN: 0 | QOS: DELIVER\_AT\_LEAST\_ONCE | TOPIC\_NAME: devices/riot-device-test/messages/events/ | PACKET\_ID: 3 | PAYLOAD\_LEN: 57

[12:55:00.466] -> PUBLISH | IS\_DUP: false | RETAIN: 0 | QOS: DELIVER\_AT\_LEAST\_ONCE | TOPIC\_NAME: devices/riot-device-test/messages/events/ | PACKET\_ID: 4 | PAYLOAD\_LEN: 57

[12:55:01.489]

[12:55:01.489] <- PUBACK | PACKET\_ID: 3

[12:55:01.489] <- PUBACK | PACKET\_ID: 4

[12:55:03.665]

[12:55:03.665] IoTHubClient\_LL\_SendEventAsync accepted message [3] for transmission to IoT Hub.

[12:55:03.917]

[12:55:03.917] -> PUBLISH | IS\_DUP: false | RETAIN: 0 | QOS: DELIVER\_AT\_LEAST\_ONCE | TOPIC\_NAME: devices/riot-device-test/messages/events/ | PACKET\_ID: 5 | PAYLOAD\_LEN: 57

[12:55:04.970]

[12:55:04.970] <- PUBACK | PACKET\_ID: 5

[12:55:07.073]

[12:55:07.123] IoTHubClient\_LL\_SendEventAsync accepted message [4] for transmission to IoT Hub.

[12:55:07.325]

[12:55:07.325] -> PUBLISH | IS\_DUP: false | RETAIN: 0 | QOS: DELIVER\_AT\_LEAST\_ONCE | TOPIC\_NAME: devices/riot-device-test/messages/events/ | PACKET\_ID: 6 | PAYLOAD\_LEN: 57

[12:55:08.399]

[12:55:08.399] <- PUBACK | PACKET\_ID: 6

[12:55:10.538]

[12:55:10.538] IoTHubClient\_LL\_SendEventAsync accepted message [5] for transmission to IoT Hub.

[12:55:10.798]

[12:55:10.798] -> PUBLISH | IS\_DUP: false | RETAIN: 0 | QOS: DELIVER\_AT\_LEAST\_ONCE | TOPIC\_NAME: devices/riot-device-test/messages/events/ | PACKET\_ID: 7 | PAYLOAD\_LEN: 57

[12:55:11.821]

[12:55:11.821] <- PUBACK | PACKET\_ID: 7

[12:55:13.961]

[12:55:14.011] IoTHubClient\_LL\_SendEventAsync accepted message [6] for transmission to IoT Hub.

[12:55:14.268]

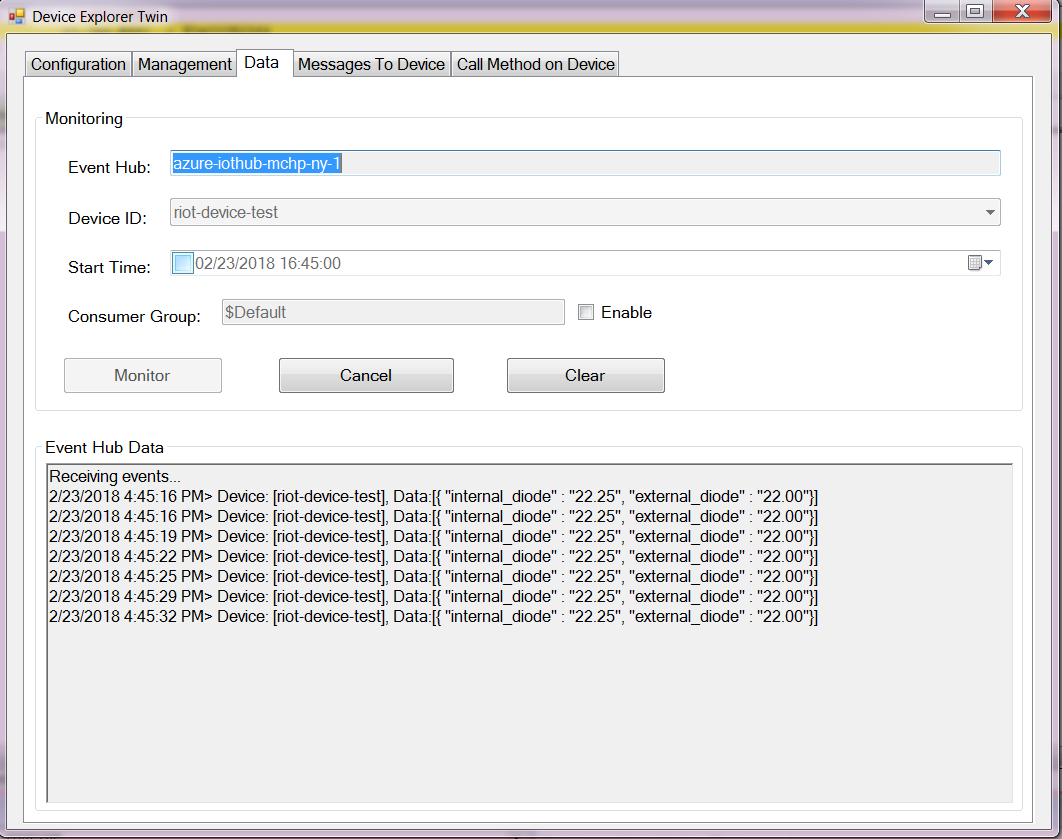
[12:55:14.268] -> PUBLISH | IS\_DUP: false | RETAIN: 0 | QOS: DELIVER\_AT\_LEAST\_ONCE | TOPIC\_NAME: devices/riot-device-test/messages/events/ | PACKET\_ID: 8 | PAYLOAD\_LEN: 57

[12:55:15.277]

[12:55:15.277] <- PUBACK

See [Device Explorer](https://github.com/fsautomata/azure-iot-sdks/blob/master/tools/DeviceExplorer/doc/how_to_use_device_explorer.md) to learn how to observe the messages IoT Hub receives from the device.

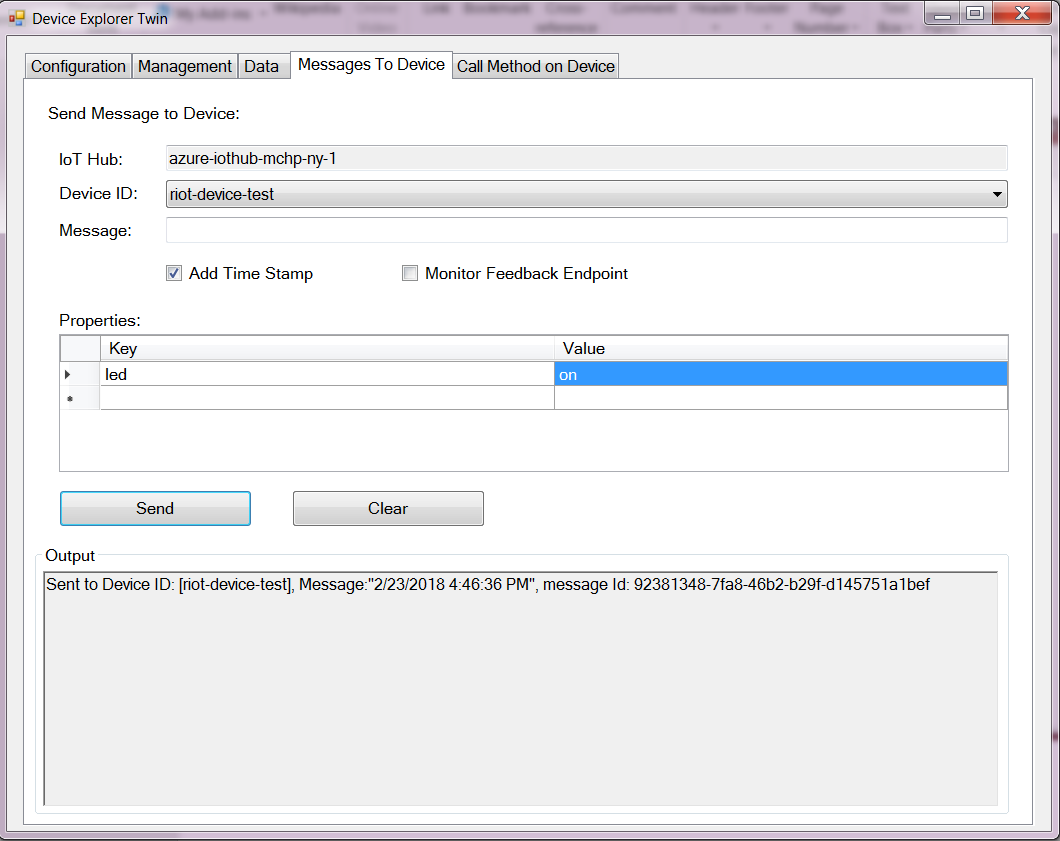
You should be able to see the temperature readings from the Thermo5 Clicker board.



## Receive messages from Azure IoT Hub

See [Device Explorer](https://github.com/fsautomata/azure-iot-sdks/blob/master/tools/DeviceExplorer/doc/how_to_use_device_explorer.md) to learn how to send cloud-to-device messages from IoT hub.

* Select the ‘**Messages to Device**’ tab in Device Explorer tool to send messages to your device
* Select your device in drop-down and type the message you want to send as key-value pair
* Click on **Send**
* Example message: *{“led":“on”}*



This message should turn ON the RGB led on the board. The received messages are displayed in the serial terminal.

Example :

[20:44:03.366] <- PUBLISH | IS\_DUP: false | RETAIN: 0 | QOS: DELIVER\_AT\_LEAST\_ONCE = 0x01 | TOPIC\_NAME: devices/dice-device-1/messages/devicebound/%24.mid=22a4184a-ecbb-473b-b816-3d5fae0a63e5&%24.to=%2Fdevices%2Fdice-device-1%2Fmessages%2FdeviceBound&iothub-ack=full&led=green | PACKET\_ID: 3 | PAYLOAD\_LEN: 0

[20:44:03.366] Received Message [0]

[20:44:03.366] Message ID: 22a4184a-ecbb-473b-b816-3d5fae0a63e5

[20:44:03.366] Correlation ID: <null>

[20:44:03.377] Data: <<<>>> & Size=0

[20:44:03.377] Message Properties:

[20:44:03.377] Key: led Value: green

[20:44:05.754] IoTHubClient\_LL\_SendEventAsync accepted message [77] for transmission to IoT Hub.

[20:44:06.106] -> PUBLISH | IS\_DUP: false | RETAIN: 0 | QOS: DELIVER\_AT\_LEAST\_ONCE | TOPIC\_NAME: devices/dice-device-1/messages/events/ | PACKET\_ID: 79 | PAYLOAD\_LEN: 83

[20:44:07.107] <- PUBACK | PACKET\_ID: 79

## Enroll device using Efuse UDS Certificate and debug

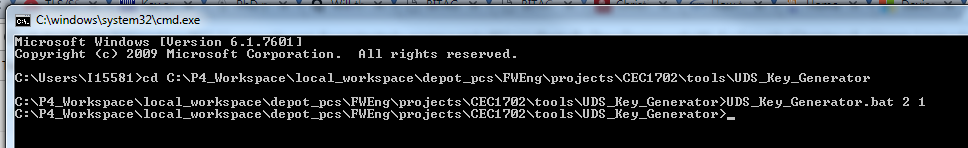
To use the efuse based UDS user needs to first program the blank CEC1702 device on CEC1x02 DevBoard or he should already have efuse programmed device

## UDS Generation

1. Go to UDS\_Key\_Generator folder in UTILITIES directory and read the readme file
2. Set the openssl path and key size in config.ini
3. Open a command prompt
4. Go to UDS\_Key\_Generator folder in tools directory and run the batch file as below

>Uds\_Key\_Generator.bat 2 1

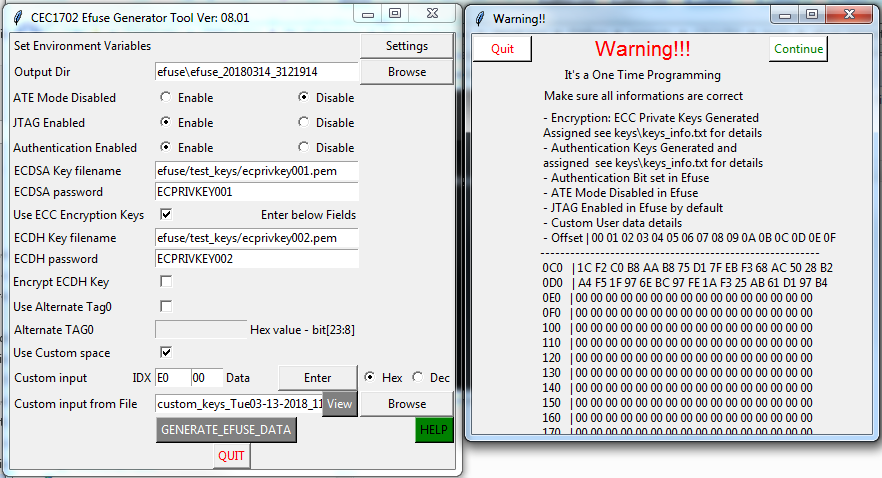
Where 1 is for number of keys (UDS) to be generated



The UDS will be generated in a hex file in UDS\_Key\_Generator\Outputs\Custom\_Keys\_raw directory

## Efuse memory map file generation

1. Run the CEC1702\_efuseGEN.exe from efuse generator downloaded separately (not part of this release package)
   1. Make sure to select DISABLE for ATE mode
   2. Make sure to select ENABLE for JTAG
   3. If Authentication is required select ENABLE and provide the filename for the key and corresponding password
   4. If Encryption is required select checkbox for “Use ECC Encryption Keys” and provide the filename for the key and corresponding password
   5. To program the UDS generated in 3.2.1 select the checkbox “Use Custom space” and browse for the hex file generated in step 3.2.1
   6. Click GENERATE\_EFUSE\_DATA, the tool would display the selected configuration
   7. Press Continue to generate the efuse memory map in hex format



## Efuse programming using MPLABX and ICD4

Ensure to have the required jumper for reference voltage for efuse programming. For cec17x2 dev board this is jumper J5 where pins 3-4 needs to be shorted

1. Open MPLABX and Under MPLAB X Options/Preferences -> Embedded -> Generic Settings
2. Set Debug Reset @ = Reset vector
3. Debug startup = Halt at Reset vector
4. Copy the hex file from step 3.2.2 to a new location and create a new project as a Prebuilt (Hex) Project with the production hex file. Set this project as the main project.
5. Debug Main project. The debugger should automatically stop after “Programming/Verify complete”.
6. In the Execution Memory window, go to the address of the Reset\_Handler() which is 0xE0349. Right click on this line and select “Set PC at Cursor”.
7. Debug continue and wait for 10-15 seconds for programming to complete. Please note there will not be any LED indication for successful efuse programing but only on failures you will see LD3 & LD4 blinking alternatively.
8. Power cycle the board

## Run the demo using ICD4 using efuse UDS

1. In the diceRIOT project Disable (comment) the macro USE\_TEST\_UDS in dps\_hsm\_riot.c
2. Compile the code and start debug session as mentioned before. You can pause the debug session once you see the below certificate on serial terminal, even if you continue to run the code the connection to Azure server will fail since the certificate is not yet provisioned which is OK at this point.

## Deriving the X.509 certificate and the thumbprint

1. The alias certificate is displayed on the serial terminal, after the following the text:

RIOT: Alias Certificate

Example:

*-----BEGIN CERTIFICATE-----*

*[15:11:14.359] *

*[15:11:14.359] -----END CERTIFICATE-----*

1. Copy the text from ‘BEGIN CERTIFICATE’ to ‘END CERTIFICATE’ to a text editor file.

Remove the time stamps and leading spaces; and save with .pem extension.

1. Provision this alias certificate .pem file in your DPS dashboard in Azure Portal

See [Provision device to Hub tutorial](https://docs.microsoft.com/en-us/azure/iot-dps/tutorial-provision-device-to-hub) for provisioning your device using enrollment lists. Follow instructions for X.509 based devices using Individual Enrollment.

Note: When you enroll your device, your device enrollment entry name will be **riot-device-3-cert.** This is changeable using the **DEVICE\_COMMON\_NAME** macro in dps\_hsm\_riot.c (line 28)

#define DEVICE\_COMMON\_NAME "riot-device-3-cert"

#ifndef USE\_TEST\_UDS

#define RIOT\_COMMON\_NAME DEVICE\_COMMON\_NAME

#else

#define RIOT\_COMMON\_NAME "riot-device-test-cert"

#endif

**Power cycle the device**

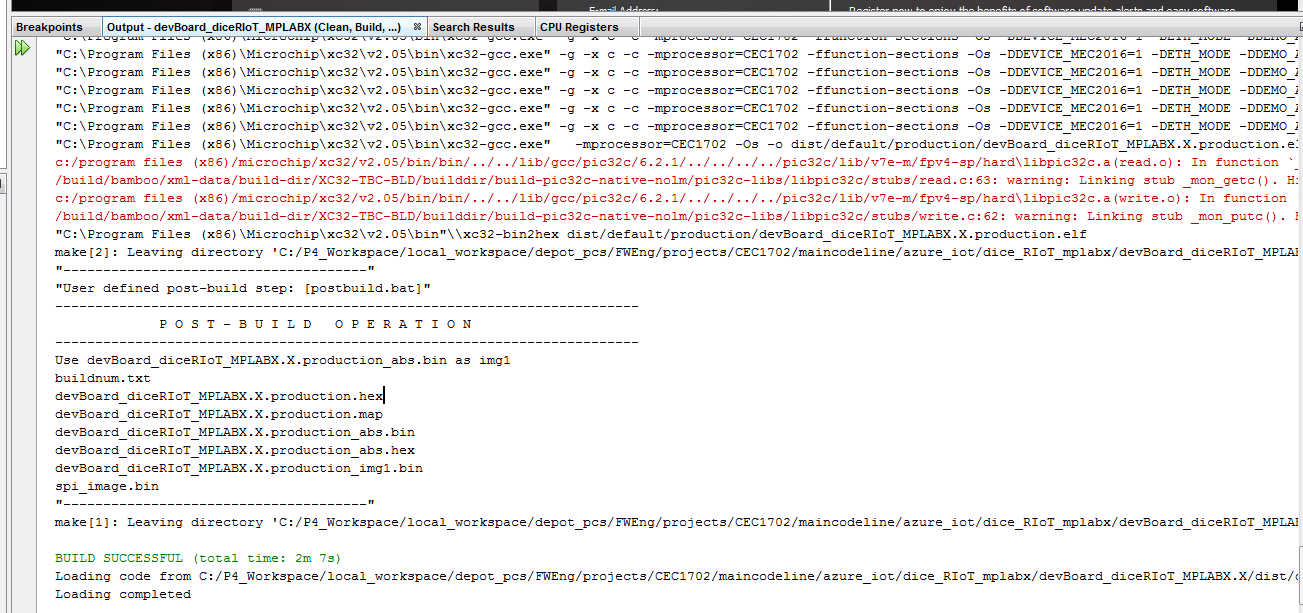
Once the alias certificate is updated for the device; on the next power-on and debug session, the device should be able to connect to the Azure IoT hub

## Enroll device using Efuse UDS Certificate and program SPI image

It is assumed that the device efuse has been programmed in step 3.2, if not please follow instructions in section 3.2 and then proceed to below steps

To use the UDS value from eFuse follow the below steps:

1. Disable (comment) the macro USE\_TEST\_UDS in dps\_hsm\_riot.c
2. Clean and Build the project . After build is complete you should have the SPI image binary ***spi\_image.bin*** generated in Target folder



1. Program the utilities\SPI\_utility\_0600\spi\_image.bin

Note: Once you program this image to SPI; the UDS will be locked after the bootrom loads and executes this application. Hence later if you run any application through JTAG, it will always read the UDS as 0.

## Deriving the X.509 certificate and the thumbprint

1. On powering the board, the alias certificate is displayed on the serial terminal, after the following the text:

RIOT: Alias Certificate

Example:

*-----BEGIN CERTIFICATE-----*

*[15:11:14.359] *

*[15:11:14.359] -----END CERTIFICATE-----*

1. Copy the text from ‘BEGIN CERTIFICATE’ to ‘END CERTIFICATE’ to a text editor file.

Remove the time stamps and leading spaces; and save with .pem extension.

1. Provision this alias certificate .pem file in your DPS dashboard in Azure Portal

See [Provision device to Hub tutorial](https://docs.microsoft.com/en-us/azure/iot-dps/tutorial-provision-device-to-hub) for provisioning your device using enrollment lists. Follow instructions for X.509 based devices using Individual Enrollment.

**Power cycle the device**

Once the alias certificate is updated for the device; on the next power-on, the device should be able to connect to the Azure IoT hub.

# Debugging connection failure

1. Ensure the you have updated the wifi details correctly as mentioned in section 3.1.2 point 6
2. Ensure the MQTT port 8883 is not blocked in your network
3. Ensure the device certificate is captured correctly from serial terminal and provisioned in Azure dashboard
4. For using efuse UDS ensure the device is programmed with efuse. The devices are blank (no efuse) by default

# Next Steps

You have now learned how to run a sample application that collects sensor data and sends it to your IoT hub. To explore how to store, analyze and visualize the data from this application in Azure using a variety of different services, please click on the following lessons:

* [Manage cloud device messaging with iothub-explorer](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-explorer-cloud-device-messaging)
* [Save IoT Hub messages to Azure data storage](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-store-data-in-azure-table-storage)
* [Use Power BI to visualize real-time sensor data from Azure IoT Hub](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-live-data-visualization-in-power-bi)
* [Use Azure Web Apps to visualize real-time sensor data from Azure IoT Hub](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-live-data-visualization-in-web-apps)
* [Weather forecast using the sensor data from your IoT hub in Azure Machine Learning](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-weather-forecast-machine-learning)
* [Remote monitoring and notifications with Logic Apps](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-monitoring-notifications-with-azure-logic-apps)

-------------------------------------------------- END OF DOCUMENT -----------------------------------------------------------