Getting Started with M.2 Modules and i.MX RT

Embedded Artists

Embedded Artists AB
Jörgen Ankersgatan 12
SE-211 45 Malmö
Sweden

http://www.EmbeddedArtists.com

Disclaimer

Embedded Artists AB makes no representation or warranties with respect to the contents hereof and specifically disclaim any implied warranties or merchantability or fitness for any particular purpose. Information in this publication is subject to change without notice and does not represent a commitment on the part of Embedded Artists AB.

Feedback

We appreciate any feedback you may have for improvements on this document. Send your comments by using the contact form: www.embeddedartists.com/contact.

Trademarks

All brand and product names mentioned herein are trademarks, services marks, registered trademarks, or registered service marks of their respective owners and should be treated as such.

Table of Contents

1 Document Revision History

Revision	*Date*	*Description*
PA1	2019-04-30	First release.
PA2	2019-05-03	Updated screen shots and some pictures.
PA3	2019-09-10	Updated screen shots in 3.1. Updated debug troubleshooting in 3.3.6. Added Bluetooth example in section 3.4
PA4	2020-01-14	Updated to match SDK 2.7.0
PA5	2020-01-31	Added note in section 5 about broken instructions

2 Introduction

This document describes how to add wireless functionality with M.2 modules to an *iMX RT1062 Developer's Kit*. The kit will be referred to as the *iMX RT Developer's Kit* for the rest of the document.

Additional documentation you need is:

- *iMX RT1052/1062 Developer's Kit Program Development Guide* - called *Program Development Guide* for short in this document
- *iMX RT1062 Developer's Kit User's Guide*
- *iMX RT1052/1062 OEM Board Datasheet*
- *M.2 Module Datasheet* for the specific M.2 module you are using

2.1 Conventions

A number of conventions have been used throughout to help the reader better understand the content of the document.

`Constant width text` – is used for file system paths and command, utility and tool names.

```
$ This field illustrates user input in a terminal running on the
development workstation, i.e., on the workstation where you edit,
configure and build Linux
```

```
# This field illustrates user input on the target hardware, i.e.,
input given to the terminal attached to the COM Board
```

```
This field is used to illustrate example code or excerpt from a
document.
```

This field is used to highlight important information

3 QuickStart Guide

This chapter is a step-by-step guide to get Wi-Fi up and running in shortest possible time:

1. The first step is to download and install the SDK.
2. The second step describes how to physically mount the M.2 module.
3. The third step describes how to setup, patch and run the example project.

Above are the three simple steps to get up-and-running immediately!

There are a couple of more sections describing different aspects, like debugging and a performance patch for iperf.

iMX RT1052
Note that as of January 2020 the NXP SDK (2.7.0) includes Wi-Fi support for the 1DX and 1LV M.2 modules and Bluetooth support only for the 1DX M.2 module. Additional Bluetooth support is scheduled for a coming release as well as support for the 1MW and 1LV M.2 modules. Exact release date is TBD.

iMX RT1062
Note that as of January 2020 the NXP SDK (2.7.0) includes Wi-Fi support for the 1DX and 1LV M.2 modules and Bluetooth support only for the 1DX M.2 module. Additional Bluetooth support is scheduled for a coming release as well as support for the 1MW and 1LV M.2 modules. Exact release date is TBD.

NXP's SDK supports several IDEs but this guide is focusing on the free MCUXpresso IDE from NXP. Combining this document and the more general *Program Development Guide* should make it easy to use another IDE as well.

3.1 Step #1: Downloading and Installing MCUXpresso SDK

This section will walk you through the installation of the MCUXpresso SDK, which is a package of sample software projects (with device drivers and peripheral examples and demos) that will get you started immediately with your i.MX RT software development. Note that even though the name of the SDK suggests the code package is for the MCUXpresso IDE, the sample projects have project files for other IDEs as well, such as Keil uVision/MDK, IAR Embedded Workbench, and more.

First download the MCUXpresso SDK by following this URL: https://mcuxpresso.nxp.com/en/welcome. This is the online SDK builder that makes sure you will get the latest version of the software.

Note that some details in the screen dumps in this section might change over time, but the basic walkthrough steps are the same.

You will need to login to your NXP account. Then click on *Select Development Board*.

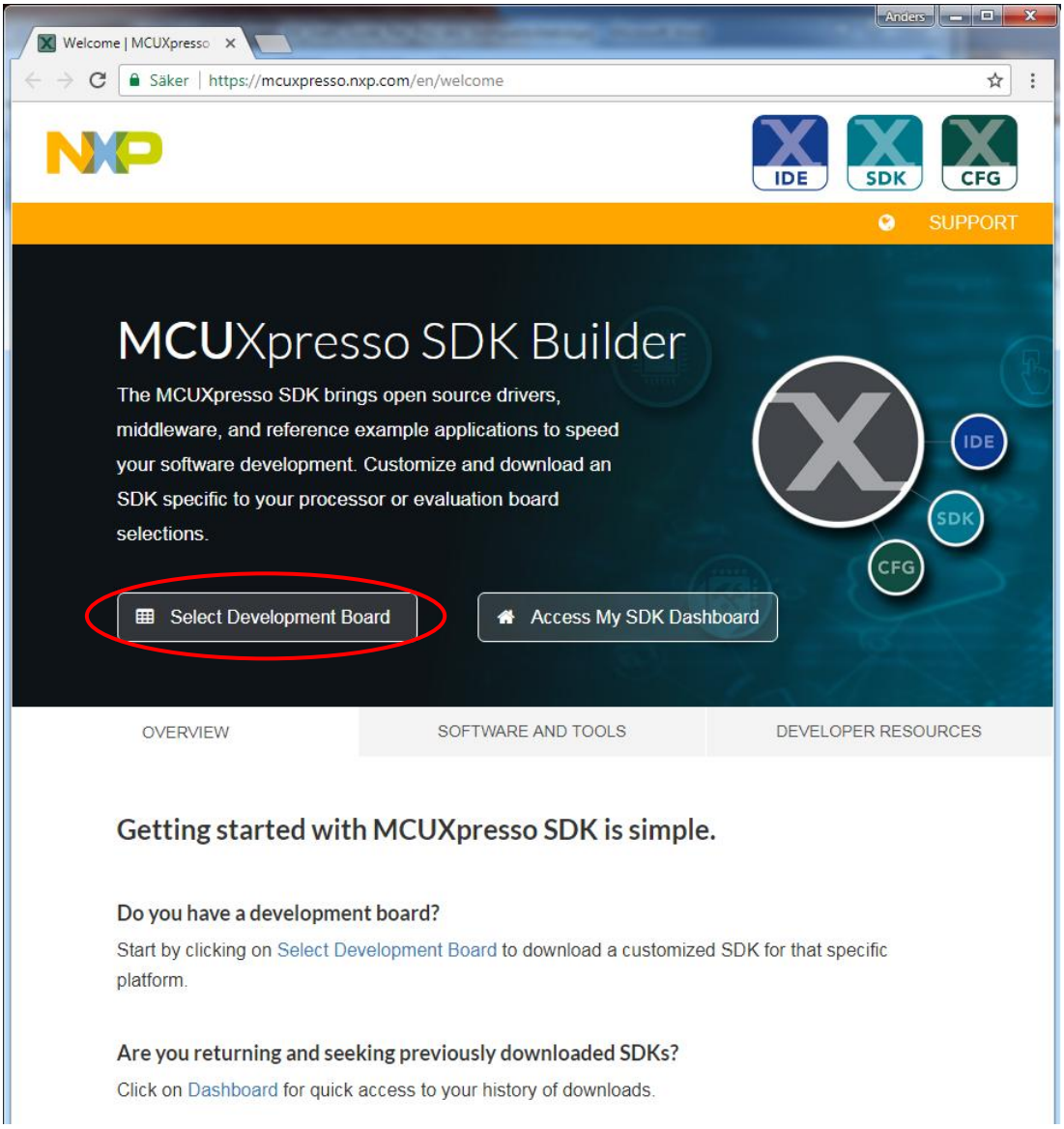


Figure 1 – MCUXpresso SDK Builder

Then select the Boards → i.MX → EVK-MIMXRT1060. This is the NXP evaluation board, but most of the samples will work unmodified on the *iMX RT Developer's Kit*.

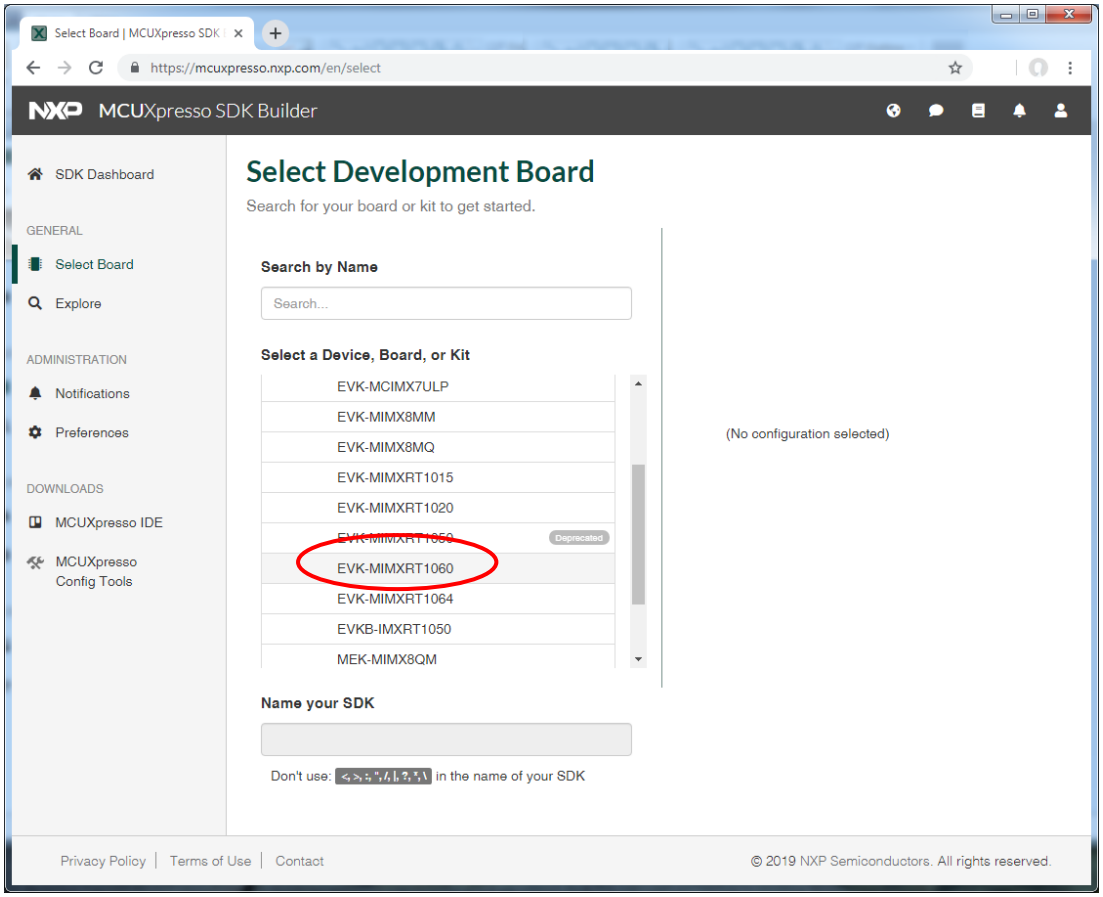


Figure 2 – MCUXpresso SDK Builder - Select Development Board

After selecting the NXP EVK, click on the *Build MCUXpresso SDK* button.

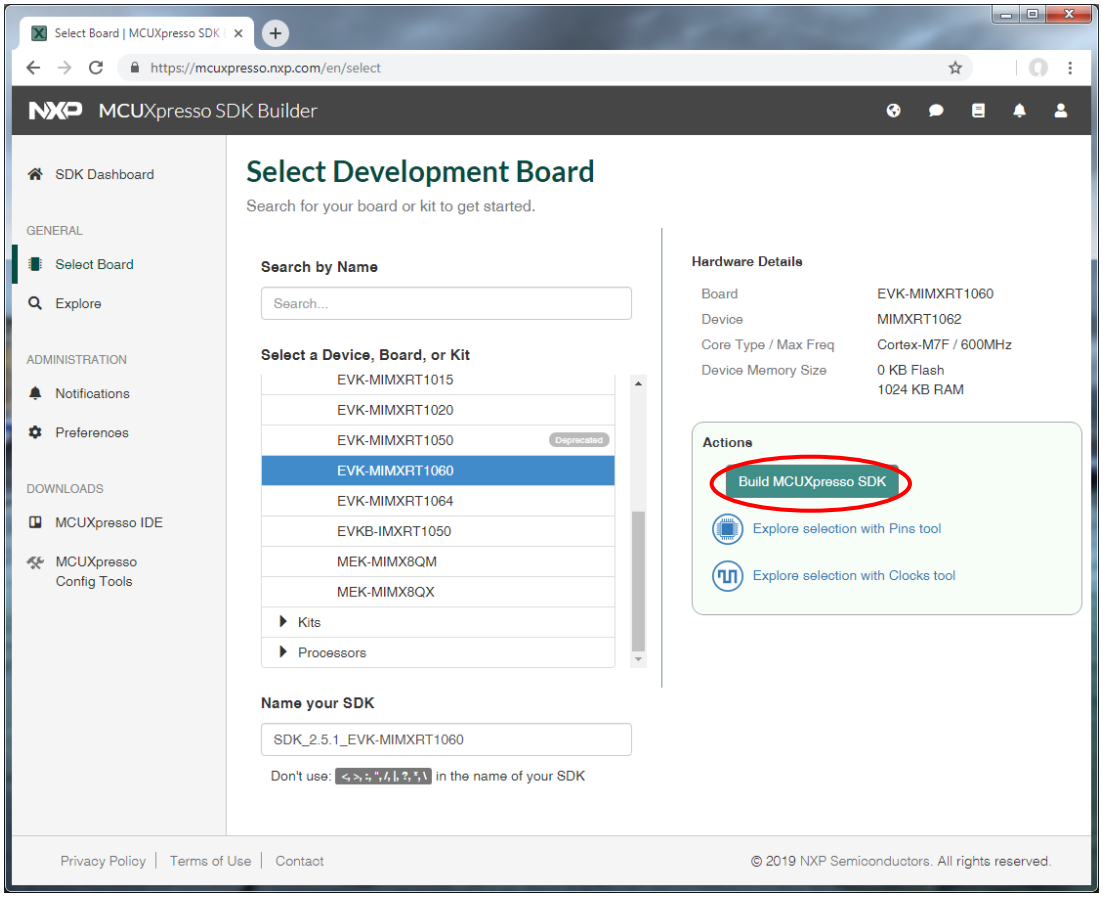


Figure 3 – MCUXpresso SDK Builder - Build MCUXpresso SDK

Select *All Toolchains* under the *Toolchain/IDE* dropdown list. Select SDK version *2.7.0 2019-12-19*.

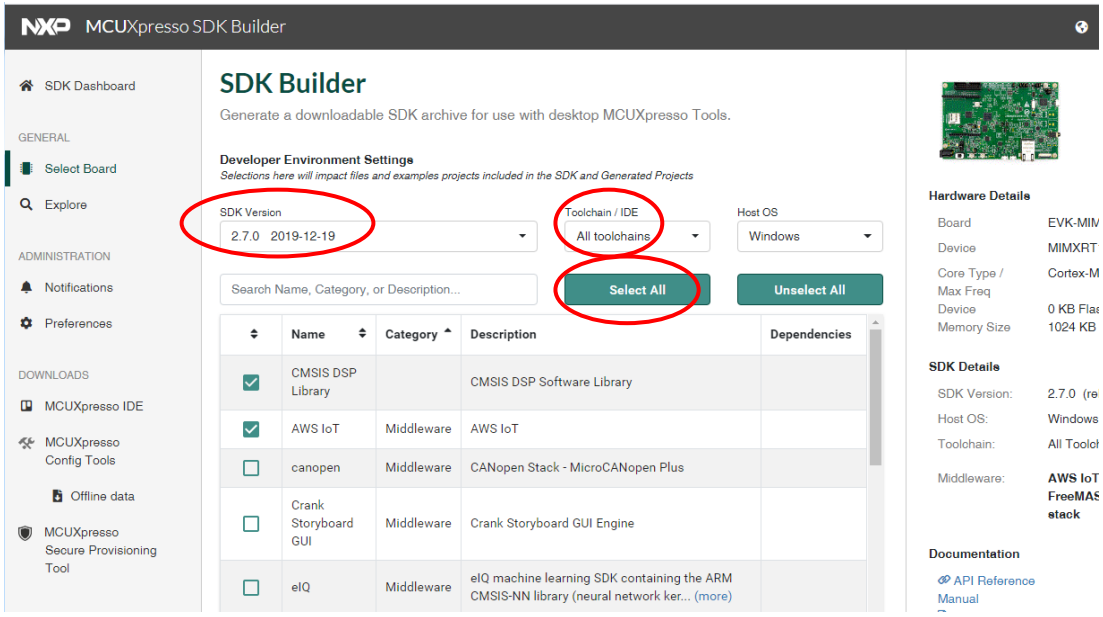


Figure 4 – MCUXpresso SDK Builder - Select Toolchain

Click the *Select* All button to select all optional software components.

Scroll to the bottom of the page and press the *Download SDK* button. This will give you a complete package with all software.

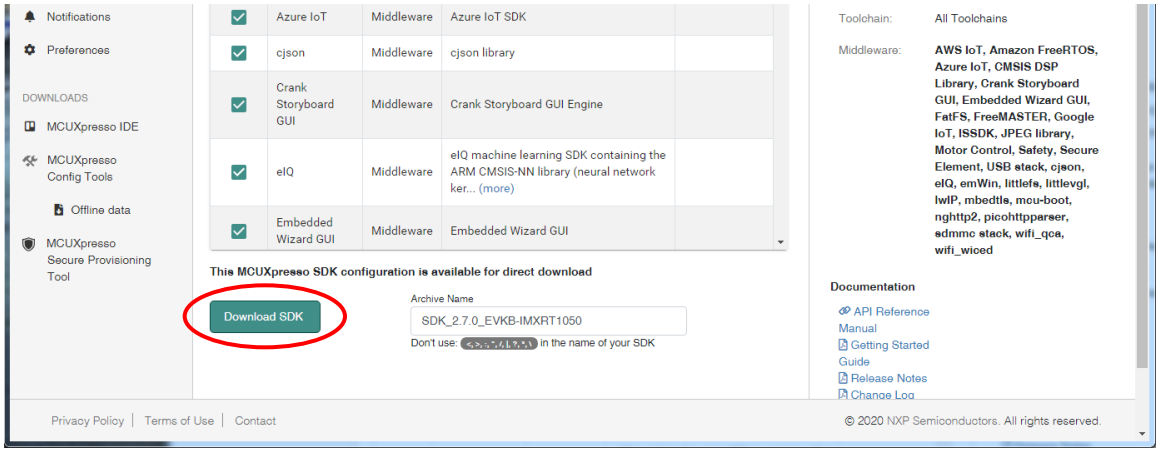


Figure 5 – MCUXpresso SDK Builder - SDK Builder

Also agree to the *Software Terms and Conditions* by clicking on the *I Agree* button.

Download of a (about) 170MByte zip-file will begin.

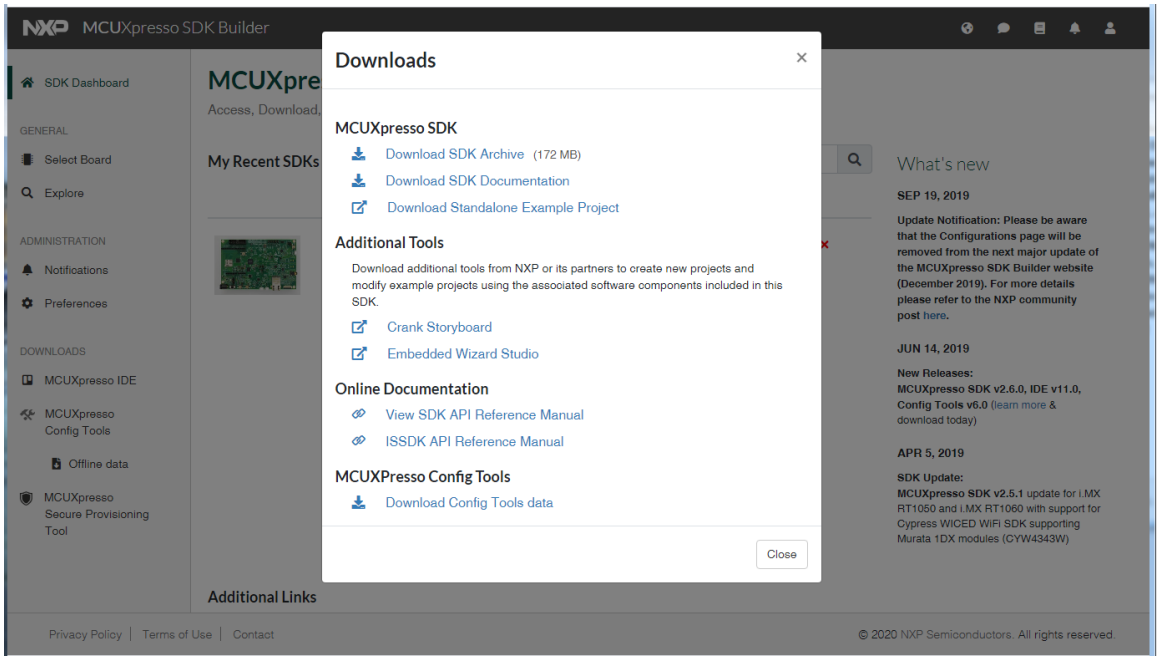
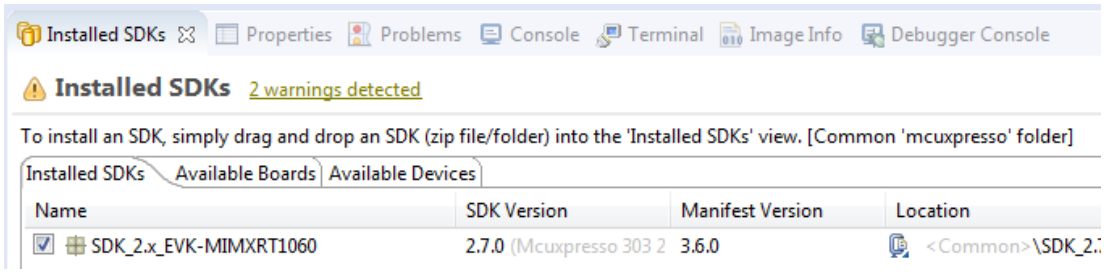


Figure 6 – MCUXpresso SDK Builder - SDK Downloads

Save the file on your computer. Note the download location. As of January 2020, the file name is **`SDK_2.7.0_EVK-MIMXRT1060.zip`** but the version number (i.e., 2.7.0) will likely increment over time.

The SDK needs to be installed in MCUXpresso before it can be used. To do that start MCUXpresso and then drag-n-drop the SDK archive (**`SDK_2.7.0_EVK-MIMXRT1060.zip`**) on to the "Installed SDKs" tab:



During the installation MCUXpresso will make a copy of the archive that you drag-n-dropped so it is ok to delete the **`SDK_2.7.0_EVK-MIMXRT1060.zip`** afterwards if you don't need it for one of the other IDEs.

3.2 Step #2: Mount M.2 Module

Make sure the *iMX RT Developer's Kit* is **powered off** and then mount the M.2 Module, as illustrated in the picture below:

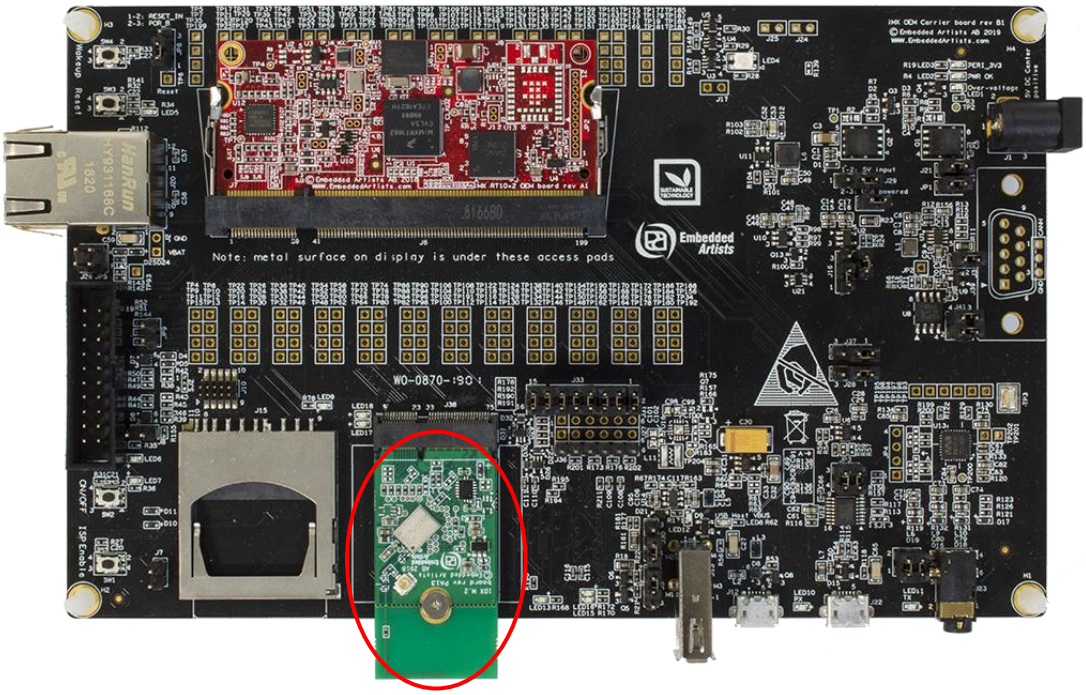


Figure 1 – M.2 Module on iMX OEM Carrier Board

Do not put too much force/pressure on the M2 screw (into the M.2 connector stand-off) so that the PCB is bent. Bending the PCB too much will damage the board!
Use your fingers on the bottom side (while screwing) to give a counter-force, keeping the PCB straight.

The picture below illustrates the typical angle (about 25 degrees) to use when inserting the M.2 module into the connector. The pictures illustrate another carrier board, but the principle is exactly the same, regardless of carrier board.

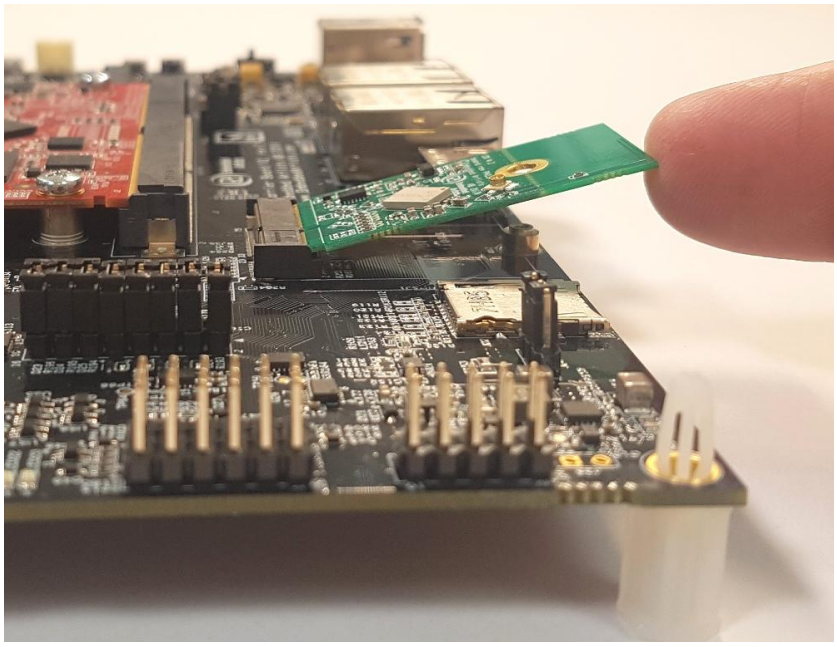


Figure 2 – M.2 Module on Carrier Board

The picture below illustrates how to use two fingers, placed under the grounding stand-off, to avoid bending the board. Make sure to always use this method to avoid damaging the board.

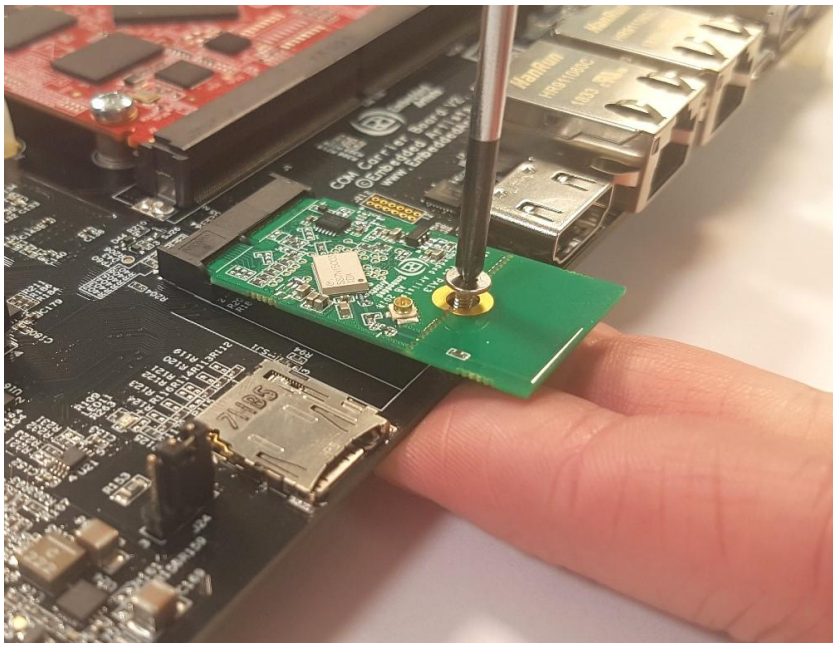


Figure 3 – M.2 Module on Carrier Board

3.3 Step #3: Example Application - iperf

As of January 2020 there are five Wi-Fi example applications in the SDK:

- wiced_iperf_43012 - this is a performance test application for the 1LV M.2 module
- wiced_iperf_4343W - this is a performance test application for the 1DX M.2 module
- wiced_iperf3_4343W - this is a performance test application for the 1DX M.2 module. This example is only available in the SDK for the iMX RT1052 and the functionality/configuration is the same as for the two other iperf examples (except that it is for the newer iperf3 protocol) so it will not be covered by this document
- wiced_mfg_test_43012 - this is a special application that is used for RF measurements and it will not be covered by this document
- wiced_mfg_test_4343W - this is a special application that is used for RF measurements and it will not be covered by this document

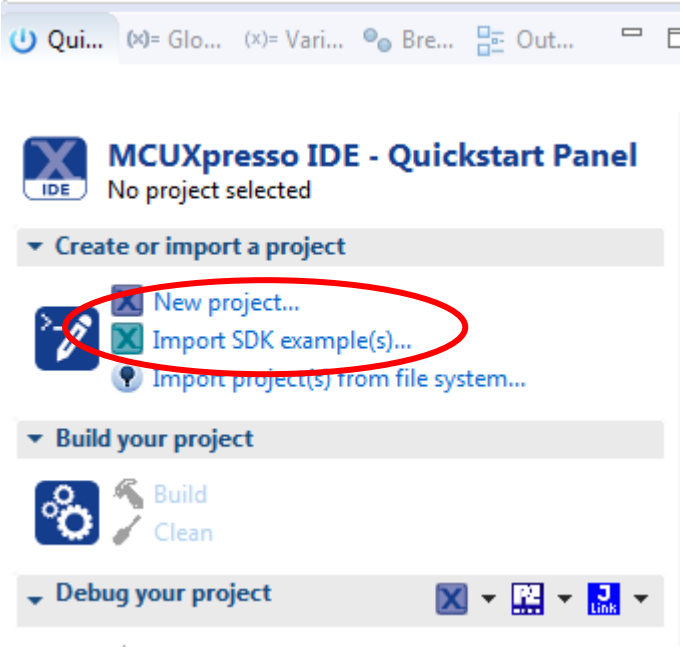
Running one of the Wi-Fi examples requires the following steps:

1. Import the example application and change flash configuration
2. Patch the example code
3. Configure the example for your environment (e.g. SSID + password for your network)
4. Compile and debug/run

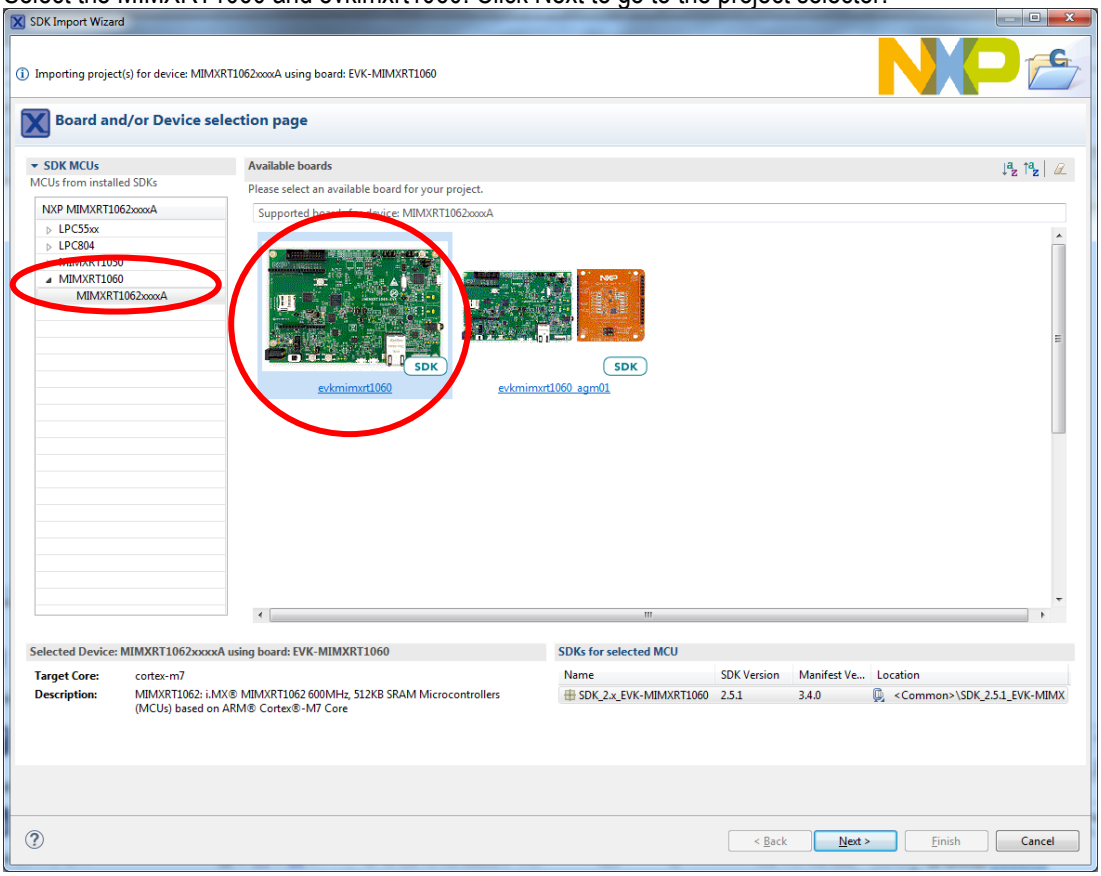
3.3.1 Import wiced-iperf

The following steps will guide you through opening the wiced-iperf application from the SDK.

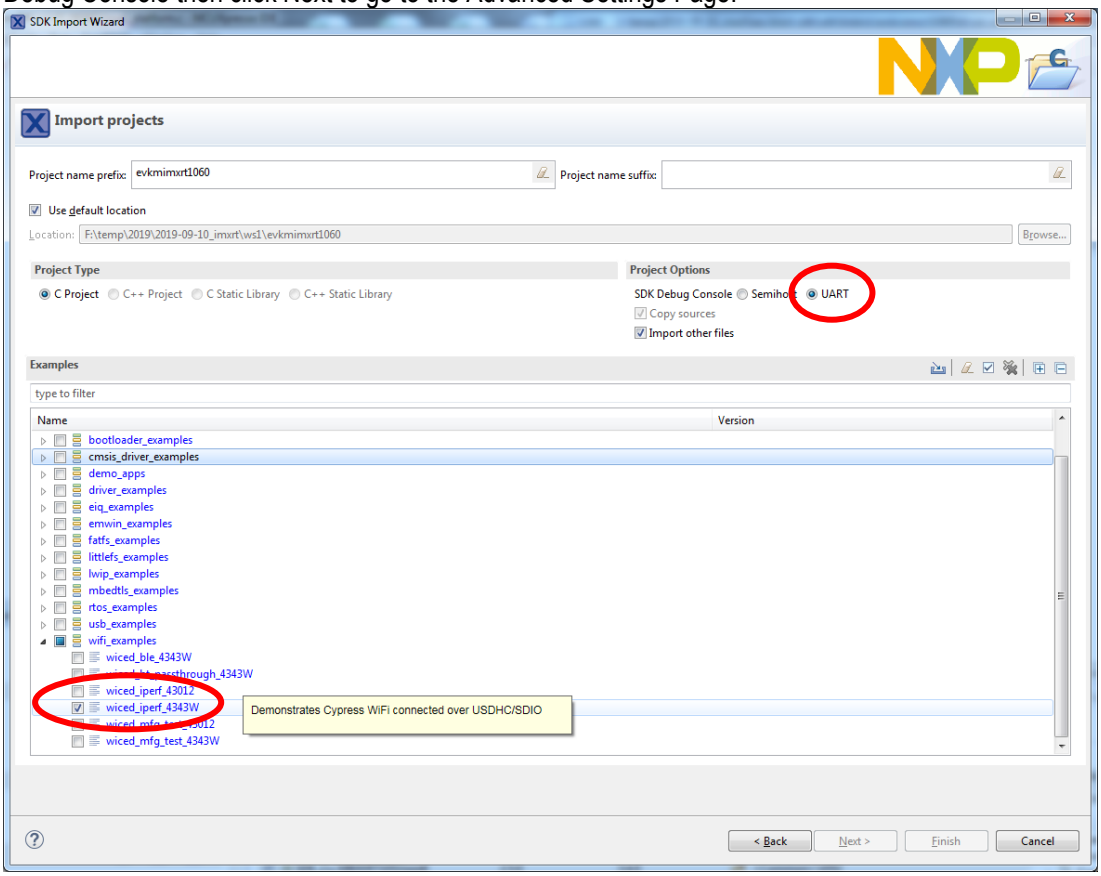
1. Install the SDK as described in section 3.1 if you have not done so already
2. Click the "Import SDK example(s)…" link in the Quickstart Panel



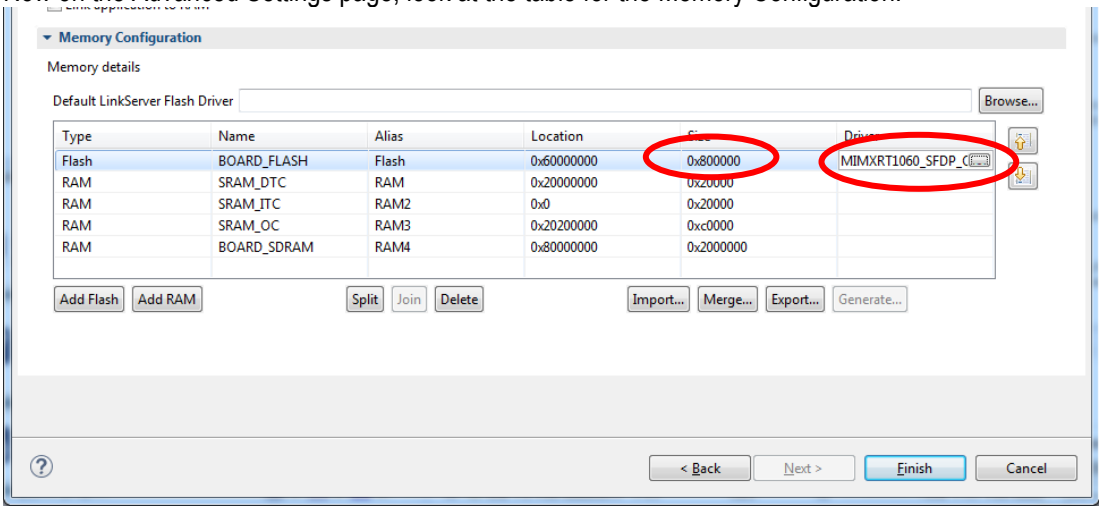
3. Select the MIMXRT1060 and evkimxrt1060. Click Next to go to the project selector.



4. Select the wiced_iperf example and make sure to switch from Semihost to UART for the SDK Debug Console then click Next to go to the Advanced Settings Page.



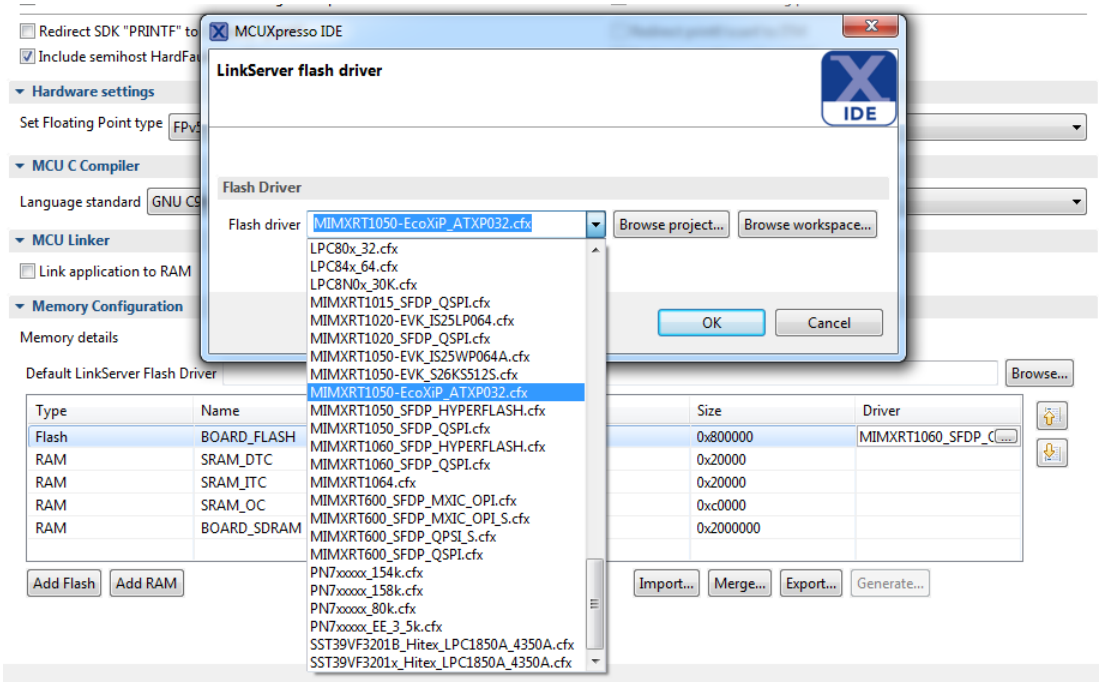
5. Now on the Advanced Settings page, look at the table for the Memory Configuration:



6. Click and change the Size for BOARD_FLASH to 0x00400000 (both iMX RT1052 and RT1062 have 4MB flash)

7. Click in the table cell for the driver and then on the small button that appears. Change the Driver for BOARD_FLASH to **MIMXRT1050-EcoXiP_ATXP032.cfx** using the dropdown menu.

The same flash driver is used for both i.MX RT1052 and RT1062.



8. With all the changes made, click ok and then finish to have MCUXpresso complete the project setup

3.3.2 Patch

This section describes how to modify the project to be compatible with the features of the *iMX RT Developer's Kit*, like the EcoXiP.

The SDK supports hyper flash but the *iMX RT Developer's Kit* has EcoXip flash from Adesto. Ideally the EcoXip flash driver should be placed in a new file with _ecoxip in the name but that would require changes to every project for every IDE in the SDK and that is a huge task. The solution is instead to replace the file content but keep the filename. This way none of the projects have to be modified.

Download the zip-file from the *Resources* tab on the product page on Embedded Artists' website. The name of the file is **`imxrt1062_ea_files_<date>.zip`**

Unpack the downloaded zip-file in a temporary directory, for example: **`c:/temp/ea_files`**, and then copy the following files from that directory into the project in MCUXpresso. It is typically easiest to do this outside of the IDE using for example Windows Explorer.

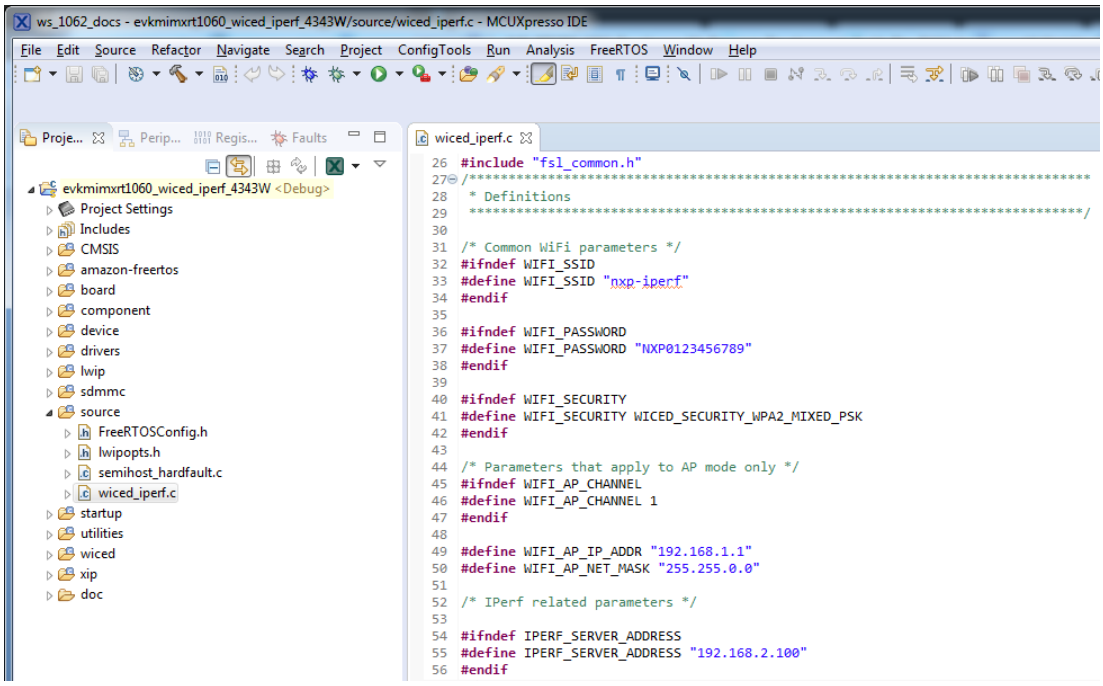
File to copy	Destination	Comment
pin_mux.c	board/	Replace existing file
fsl_lpi2c.c	drivers/	New file
fsl_lpi2c.h	drivers/	New file
pca6416.c	source/	New file
pca6416.h	source/	New file
wwd_platform.c	wiced/43xxx_Wi-Fi/WICED/platform/MCU/LPC/WWD/	Replace existing file
wwd_SDIO.c	wiced/43xxx_Wi-Fi/WICED/platform/MCU/LPC/WWD/	Replace existing file

evkmimxrt1060_flexspi_nor_config.c	xip/	Replace existing file

After copying the files into the project the IDE must be told to look for the new files. In MCUXpresso this is done by right clicking the project and selecting *Refresh* in the menu.

3.3.3 Configure the Example

Before the wiced_iperf example can be used some settings must be changed to allow it to find and connect to your network. These settings are found near the top of the wiced_iperf.c file and looks like this:



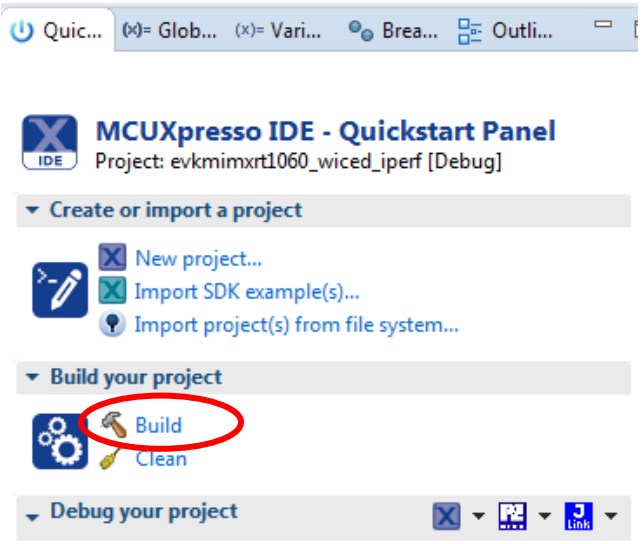
Change the WIFI_SSID, WIFI_PASSWORD and WIFI_SECURITY to match your network.

Change the IPERF_SERVER_ADDRESS to the server you want to test against. This option will be explained more in detail later but it is good to know where the setting is in the code.

3.3.4 Compile and Run/Debug

To download and run the application, perform these steps:

1. Click Build in the Quickstart Panel

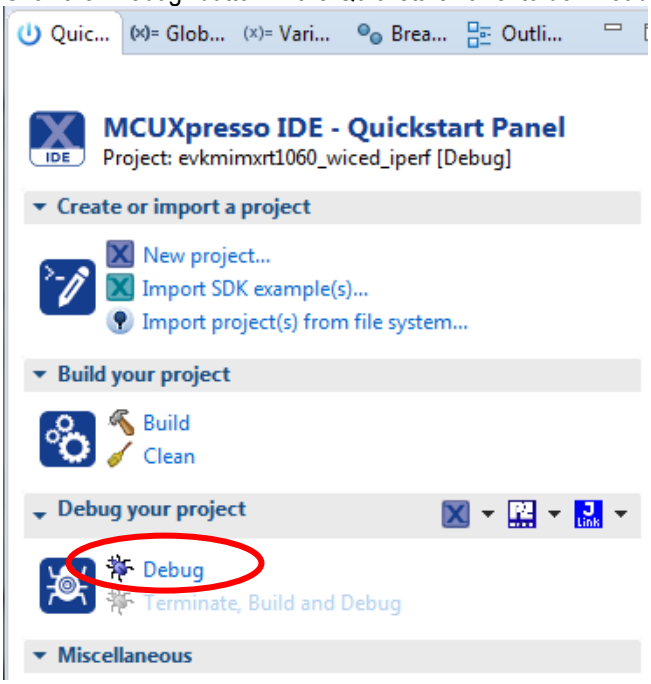


2. The program builds without errors

3. Connect the debug probe (LPC-Link2, ULINK2, J-LINK, etc.) to development platform to your PC via USB cable. See chapter **4** for details how to connect the debug probe.

4. Open the terminal application on the PC, such as TeraTerm or PuTTY, and connect to the debug serial port number. Configure the terminal with 115200 baud, 8N1.
   You can alter the baud rate be searching for the reference BOARD_DEBUG_UART_BAUDRATE variable in file: **`board.h`**

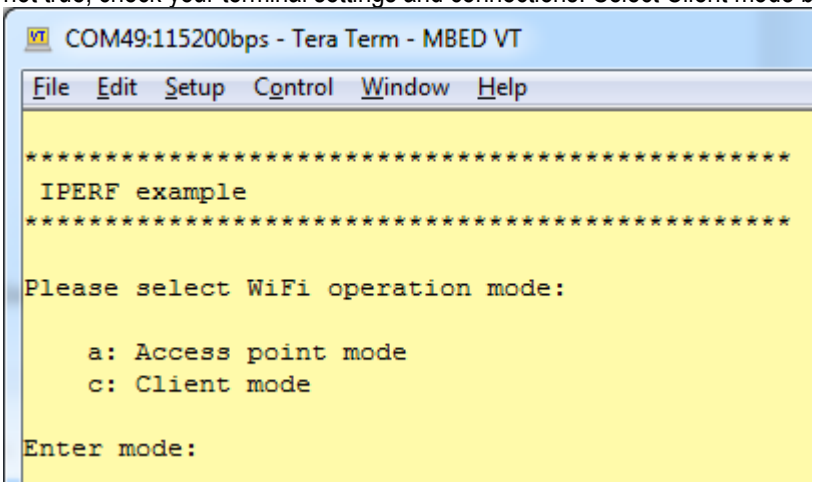
5. Click the "Debug" button in the Quickstart Panel to download the application to the target.



6. The application is then downloaded to the target and automatically runs to the main() function.

7. Run the code by clicking the "Resume" button to start the application.

8. The wiced_iperf application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections. Select Client mode by typing c.



3.3.5 Explanation of the example

Ping is a great way to test if the hardware is connected to the network, or not, but to really test the network interface it is better to use a program like `iperf`. Iperf requires both a server and a client to run the tests. The client is typically run on the iMX RT1062 board and the server software can either be installed on a computer on the local network (https://iperf.fr/iperf-download.php) or one of the online servers can be used (https://iperf.fr/iperf-servers.php). It is also possible to run the server on the iMX RT1062 and the client on a PC in the same local network.

> Make sure to select the latest version of the 2.0.* branch (2.0.9 as of May 2019) when downloading the iperf software from https://iperf.fr/iperf-download.php. Version 3.* is not supported by the demo code and it will not work as iperf and iperf3 are completely different protocols.

The wiced_iperf example implements eight different modes of testing:

```
Successfully joined: VSG1
Getting IP address from DHCP server
 IPv4 Address got from DHCP  : 192.168.50.212

Please select one of the following modes to run IPERF with:

    1: TCP server mode (RX only test)
    2: TCP client mode (TX only test)
    3: TCP client dual mode (TX and RX in parallel)
    4: TCP client tradeoff mode (TX and RX sequentially)
    5: UDP server mode (RX only test)
    6: UDP client mode (TX only test)
    7: UDP client dual mode (TX and RX in parallel)
    8: UDP client tradeoff mode (TX and RX sequentially)

Enter mode number:
```

Mode 1 - Server RX Only

Type a "1" to start server mode. Check the IP number that was assigned (in this case 192.168.50.164 as shown in the image above) and then run the following command on the PC (the program exists for most common operating systems but is shown here for Windows):

```
C:\temp> iperf -c 192.168.50.164 -i 2 -t 20 -P 4
```

The parameters specifies the IP number of the iMX RT1062 board, that a report should be printed every 2 seconds, that the test will run for 20 seconds and use 4 threads/connections.

After the test completes it will print something like this on the PC:

```
[SUM]  0.0-20.0 sec  92.5 MBytes   37.3 Mbits/sec
```

The log from the iMX RT1062 will show something like this for each of the 4 connections after the test completes:

```
------------------------------------------------
 TCP_DONE_SERVER (RX)
 Local address : 192.168.50.164  Port 5001
 Remote address : 192.168.50.2  Port 43036
 Bytes Transferred 24248344
 Duration (ms) 27485
 Bandwidth (kbitpsec) 7056
------------------------------------------------
```

The information shown on the target is not entirely correct. It shows the correct number of bytes transferred but does not calculate the time correctly. As seen above the time is 27485ms for a test that takes 20 seconds. The wiced_iperf code measures time from the connection is established and not from the first byte is sent over the connection. Depending on the PC software it may take extra time (in this case roughly 7.5 seconds) to setup the test and this should not be included in the test result but it is. When checking the result look at the PC instead of the target (i.e., the iMX RT1062).

Press SPACE to return to the main menu to run this or another test again.

Mode 2 - Client TX Only

Before running this test the PC must be running the server:

```
C:\temp> iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  208 KByte (default)
------------------------------------------------------------
```

Type a "2" to start client mode. It will run the tests against the server IP number configured in section 3.3.3 . Make sure that matches the IP number of the PC running the server.

After the test completes (it takes 10 seconds) it will print the following:

```
------------------------------------------------
 TCP_DONE_CLIENT (TX)
 Local address : 192.168.50.164  Port 49153
```

```
 Remote address : 192.168.50.2  Port 5001
 Bytes Transferred 23600924
 Duration (ms) 10000
 Bandwidth (kbitpsec) 18880
```

Press SPACE to return to the main menu to run this or another test again.

Mode 3 - Client RX+TX in Parallel

Before running this test the PC must be running the server:

```
C:\temp> iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  208 KByte (default)
------------------------------------------------------------
```

Type a "3" to start in client mode. It will run the tests against the server IP number configured in section 3.3.3 . Make sure that matches the IP number of the PC running the server.

After the test completes (it takes 10 seconds) it will print the following:

```
------------------------------------------------
 TCP DONE CLIENT (TX)
 Local address : 192.168.50.164  Port 49153
 Remote address : 192.168.50.2  Port 5001
 Bytes Transferred 16591464
 Duration (ms) 10000
 Bandwidth (kbitpsec) 13272
------------------------------------------------
 TCP_DONE_SERVER (RX)
 Local address : 192.168.50.164  Port 5001
 Remote address : 192.168.50.2  Port 43052
 Bytes Transferred 19004004
 Duration (ms) 10520
 Bandwidth (kbitpsec) 14448
------------------------------------------------
```

Press SPACE to return to the main menu to run this or another test again.

Mode 4 - Client RX+TX in Sequence

Before running this test the PC must be running the server:

```
C:\temp> iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  208 KByte (default)
------------------------------------------------------------
```

Type a "4" to start in client mode. It will run the tests against the server IP number configured in section 3.3.3 . Make sure that matches the IP number of the PC running the server.

After the test completes (it takes 10 seconds) it will print the following:

```
------------------------------------------------
 TCP_DONE_CLIENT (TX)
 Local address : 192.168.50.164  Port 49153
 Remote address : 192.168.50.2  Port 5001
 Bytes Transferred 23441784
 Duration (ms) 10000
 Bandwidth (kbitpsec) 18752
------------------------------------------------
 TCP_DONE_SERVER (RX)
 Local address : 192.168.50.164  Port 5001
 Remote address : 192.168.50.2  Port 43054
 Bytes Transferred 30276044
 Duration (ms) 10005
 Bandwidth (kbitpsec) 24208
```

Press SPACE to return to the main menu to run this or another test again.

Modes 5 to 8 - UDP

The first four modes were for TCP and modes 5 through 8 are for UDP. The tests are run in the same way except for one thing: the -u option must be used on the PC side to force iperf into UDP mode.

```
C:\temp> iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
```

Or

```
C:\temp> iperf -u -c 192.168.50.164 -i 2 -t 20 -P 4
```

3.3.6 Troubleshooting

Some things to check if the application is not working as expected:

- **Program not starting after being flashed**.
  This is typically because either the flash algorithm/flash size was not changed when importing the project or the project files were not patched.

- **The iperf tests fail**.
  Check that there in actual connection to the (wireless) network. Look for these prints in the boot log:

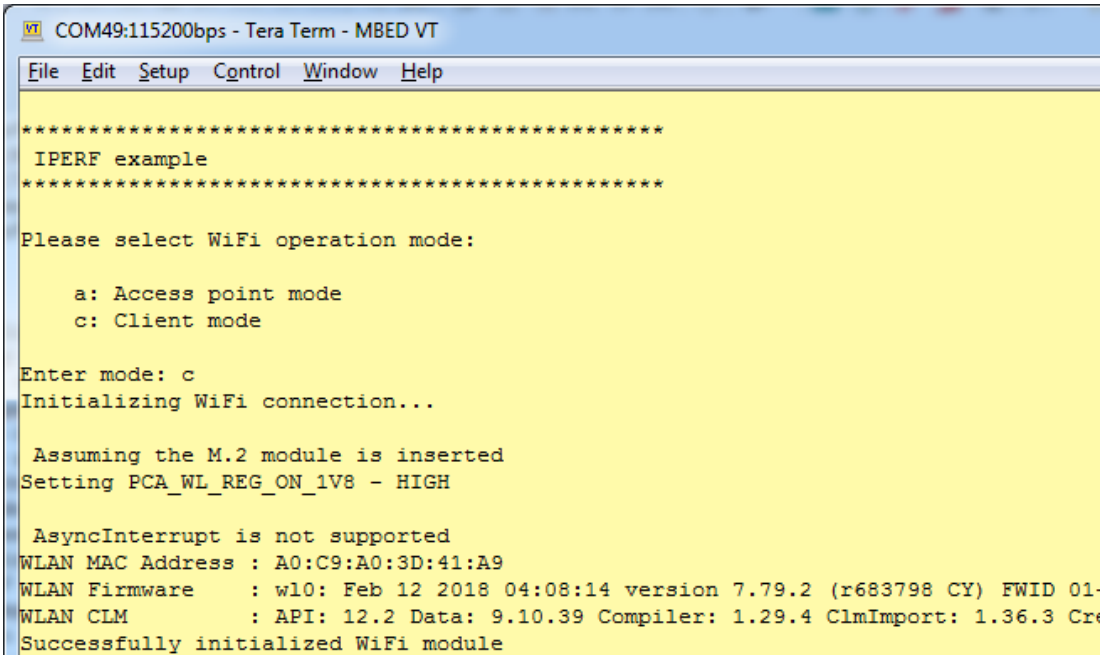
```
Joining: VSG1
Successfully joined: VSG1
Getting IP address from DHCP server
 IPv4 Address got from DHCP  : 192.168.50.212

Please select one of the following modes to run IPERF with:

    1: TCP server mode (RX only test)
```

In order for the tests to work the "Successfully joined" message must be shown and the board must have received an IP address from DHCP. If one of those is missing check the SSID information again (see 3.3.3).

- **Cannot find M.2 module**.
  A successful detection of the M.2 module looks like this:



  If the M.2 module cannot be detected, check that the module is correctly inserted and that all patches were applied as specified in section 3.3.2 Make sure that the project matches the M.2 module - the wiced_iperf_43012 is for 1LV and wiced_iperf4343W is for 1DX.

- **Firewalls**
  As with all network tests make sure that the firewall on the test PC is not stopping the traffic from the iMX RT1062 board. This is probably not a problem when running the iperf client on the PC as it is outgoing traffic, however when running the iperf server it expects incoming connections and those may be blocked by a firewall.

- **Throughput**
  The exact speed that can be achieved during testing is heavily dependent on the test environment including but not limited to:
  - Use of onboard antenna vs external antenna
  - Wireless router performance
  - Distance between iMX RT1062 and router
  - PC that is running the software
  - Usage of the 2.4GHz/5GHz band by others

3.4 Step #4: Example Application - Bluetooth

As of January 2020 there are two Bluetooth example applications in the SDK:

- wiced_ble_4343W - this is a Bluetooth Low Energy (BLE) + Wi-Fi application for the 1DX M.2 module
- wiced_bt_passthrough_4343W - this application allows direct communication over UART between the 1DX M.2 module and a PC. It requires the CyBluetool from Cypress to work and it will not be covered by this document

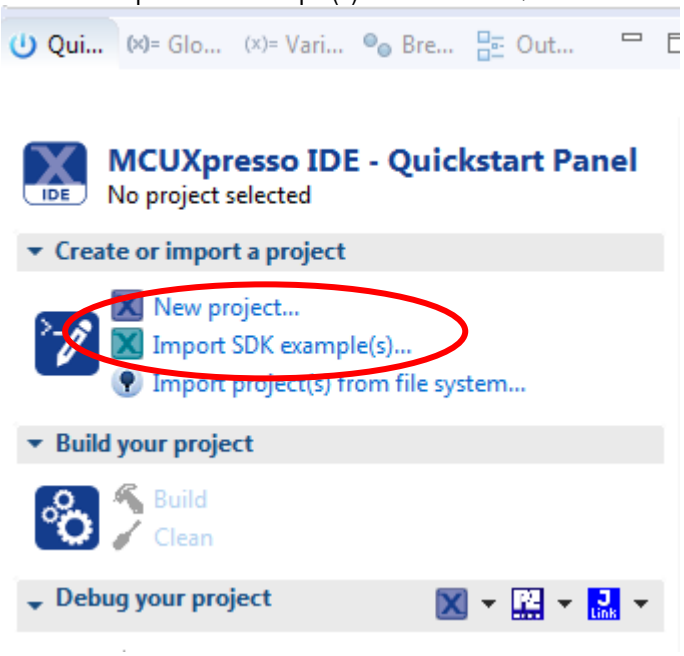
Running one of the examples requires the following steps:

1. Import the example application and change flash configuration
2. Patch the example code
3. Configure the example for your environment (e.g. SSID + password for your network)
4. Compile and debug/run

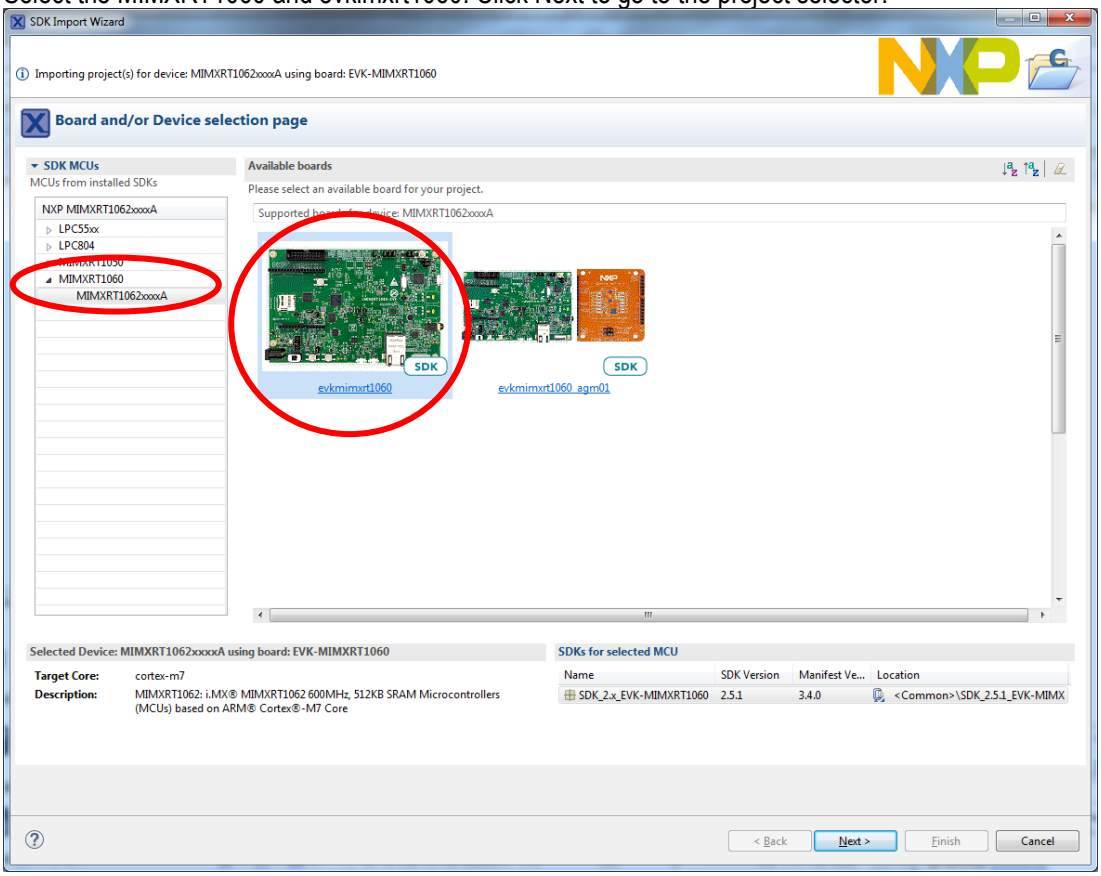
3.4.1 Import wiced_ble_4343W

The following steps will guide you through opening the wiced_ble_4343W application from the SDK.

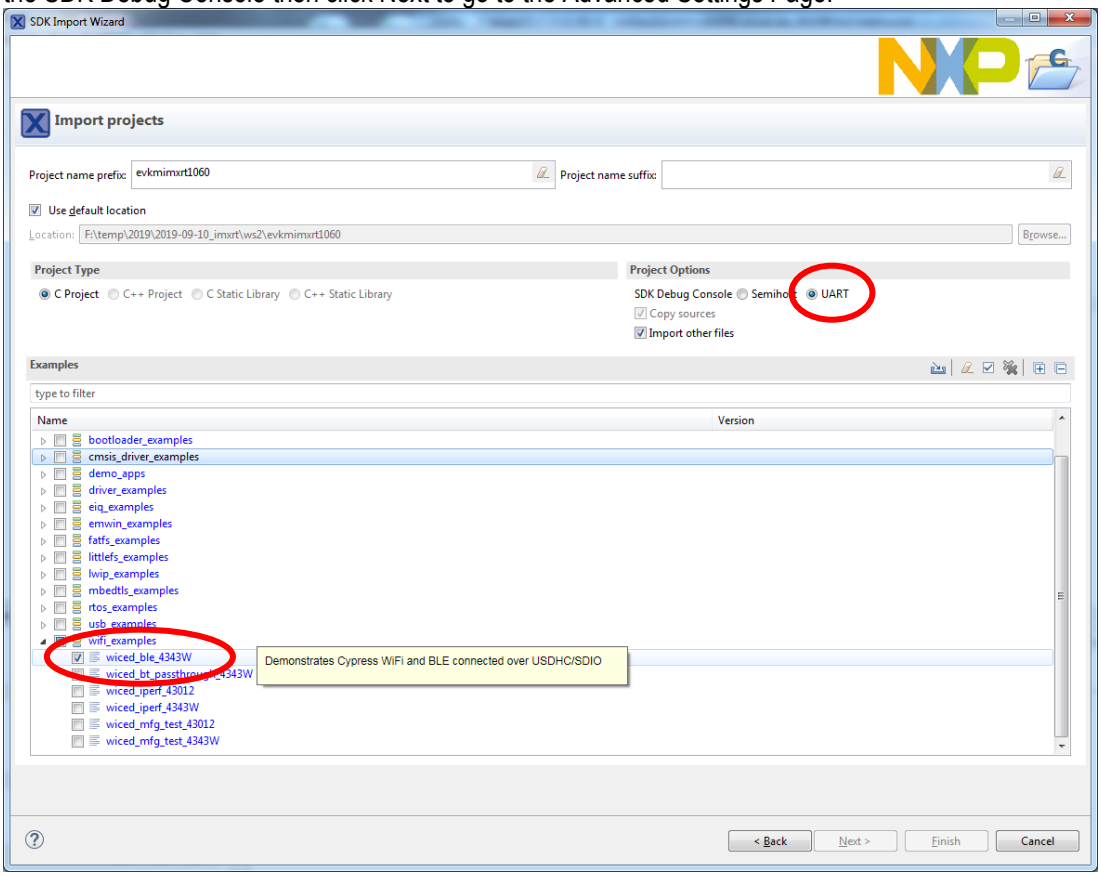
1. Install the SDK as described in section 3.1 if you have not done so already
2. Click the "Import SDK example(s)…" link in the Quickstart Panel



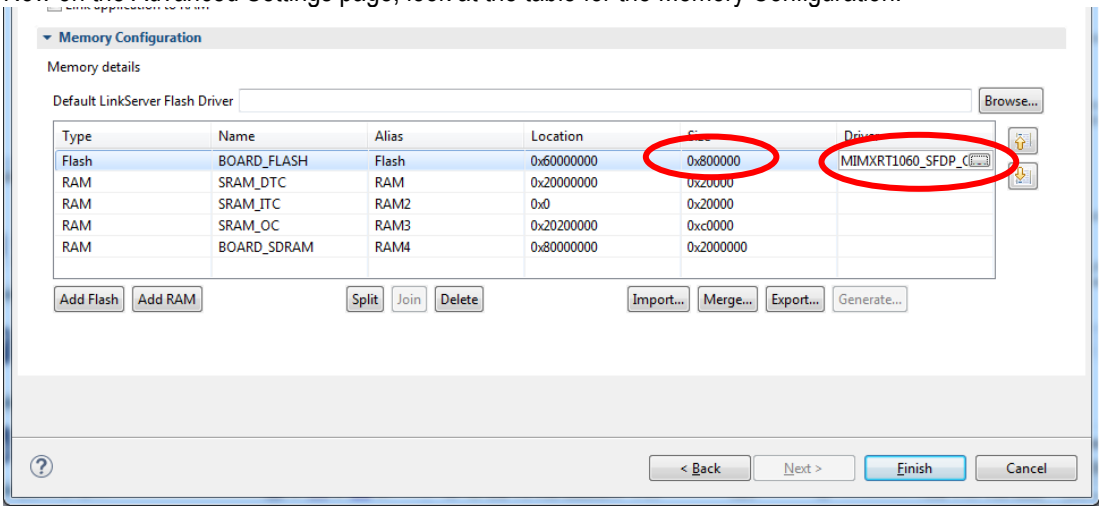
3. Select the MIMXRT1060 and evkimxrt1060. Click Next to go to the project selector.



4. Select the wiced_ble_4343W example and make sure to switch from Semihost to UART for the SDK Debug Console then click Next to go to the Advanced Settings Page.



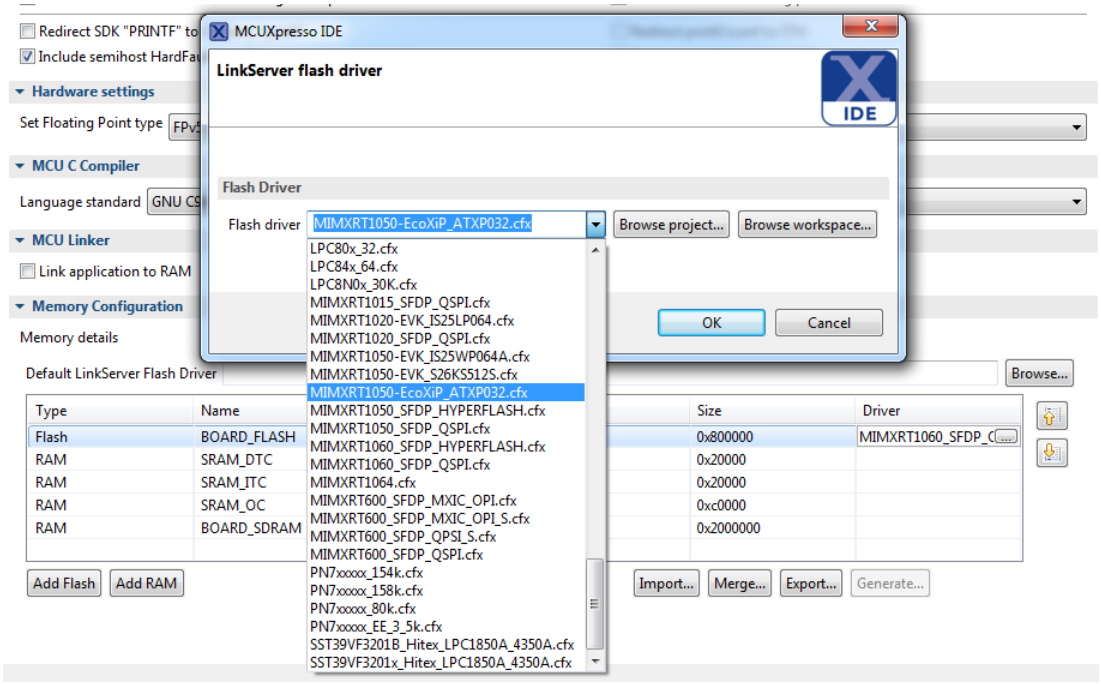
5. Now on the Advanced Settings page, look at the table for the Memory Configuration:



6. Click and change the Size for BOARD_FLASH to 0x00400000 (both iMX RT1052 and RT1062 have 4MB flash)

7. Click in the table cell for the driver and then on the small button that appears. Change the Driver for BOARD_FLASH to **MIMXRT1050-EcoXiP_ATXP032.cfx** using the dropdown menu.

The same flash driver is used for both i.MX RT1052 and RT1062.



8. With all the changes made, click ok and then finish to have MCUXpresso complete the project setup

3.4.2 Patch

This section describes how to modify the project to be compatible with the features of the *iMX RT Developer's Kit*, like the EcoXiP.

The SDK supports hyper flash but the *iMX RT Developer's Kit* has EcoXip flash from Adesto. Ideally the EcoXip flash driver should be placed in a new file with _ecoxip in the name but that would require changes to every project for every IDE in the SDK and that is a huge task. The solution is instead to replace the file content but keep the filename. This way none of the projects have to be modified.

Download the zip-file from the *Resources* tab on the product page on Embedded Artists' website. The name of the file is **`imxrt1062_ea_files_<date>.zip`**

Unpack the downloaded zip-file in a temporary directory, for example: **`c:/temp/ea_files`**, and then copy the following files from that directory into the project in MCUXpresso. It is typically easiest to do this outside of the IDE using for example Windows Explorer.

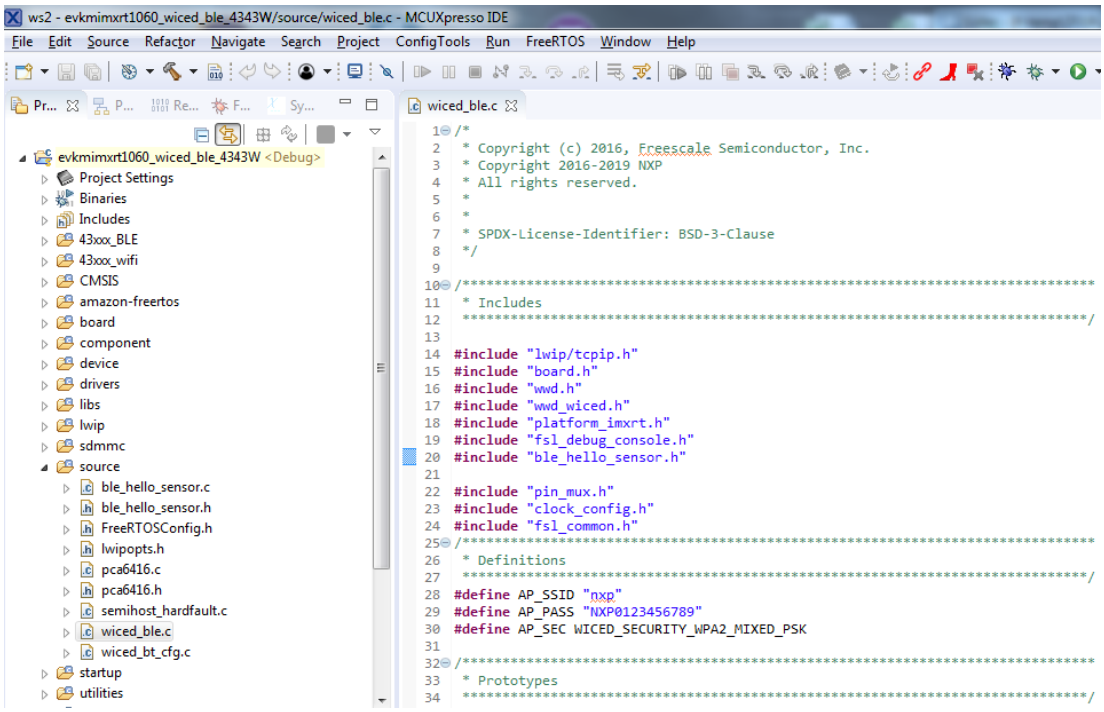
File to copy	Destination	Comment
pin_mux_ble.c	board/pin_mux.c	Replace existing file
fsl_lpi2c.c	drivers/	New file
fsl_lpi2c.h	drivers/	New file
pca6416.c	source/	New file
pca6416.h	source/	New file
wwd_platform.c	wiced/43xxx_Wi-Fi/WICED/platform/MCU/LPC/WWD/	Replace existing file
wwd_SDIO.c	wiced/43xxx_Wi-Fi/WICED/platform/MCU/LPC/WWD/	Replace existing file

platform_imxrt.c	wiced/43xxx_BLE/wiced_bt/imxrt_port/	Replace existing file
evkmimxrt1060_flexspi_nor_config.c	xip/	Replace existing file

After copying the files into the project the IDE must be told to look for the new files. In MCUXpresso this is done by right clicking the project and selecting *Refresh* in the menu.

3.4.3 Configure the Example

Before the wiced_ble_4343W example can be used some settings must be changed to allow it to find and connect to your network. These settings are found near the top of the wiced_ble.c file and looks like this:

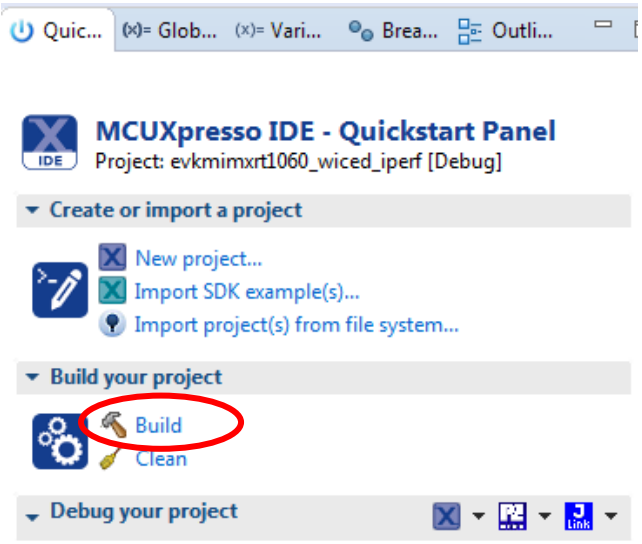


Change the AP_SSID, AP_PASS and AP_SEC to match your network.

3.4.4 Compile and Run/Debug

To download and run the application, perform these steps:

1. Click Build in the Quickstart Panel

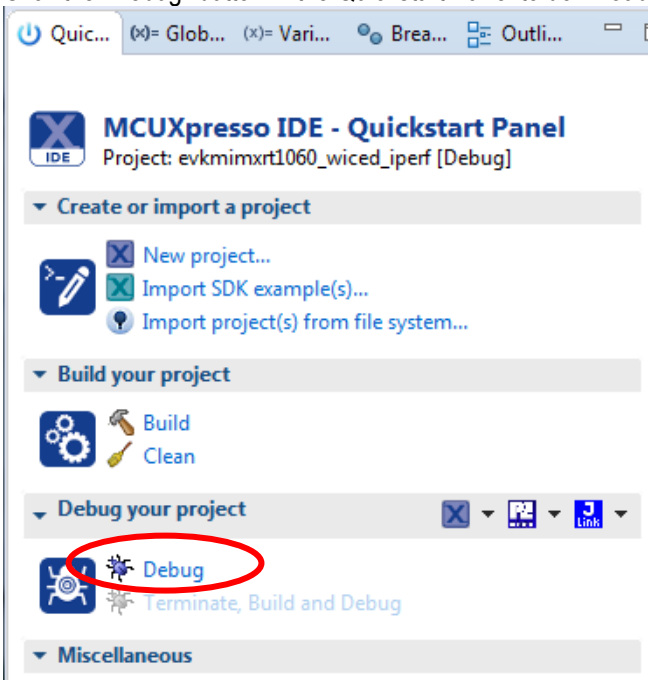


2. The program builds without errors

3. Connect the debug probe (LPC-Link2, ULINK2, J-LINK, etc.) to development platform to your PC via USB cable. See chapter **4** for details how to connect the debug probe.

4. Open the terminal application on the PC, such as TeraTerm or PuTTY, and connect to the debug serial port number. Configure the terminal with 115200 baud, 8N1.
   You can alter the baud rate be searching for the reference BOARD_DEBUG_UART_BAUDRATE variable in file: **`board.h`**

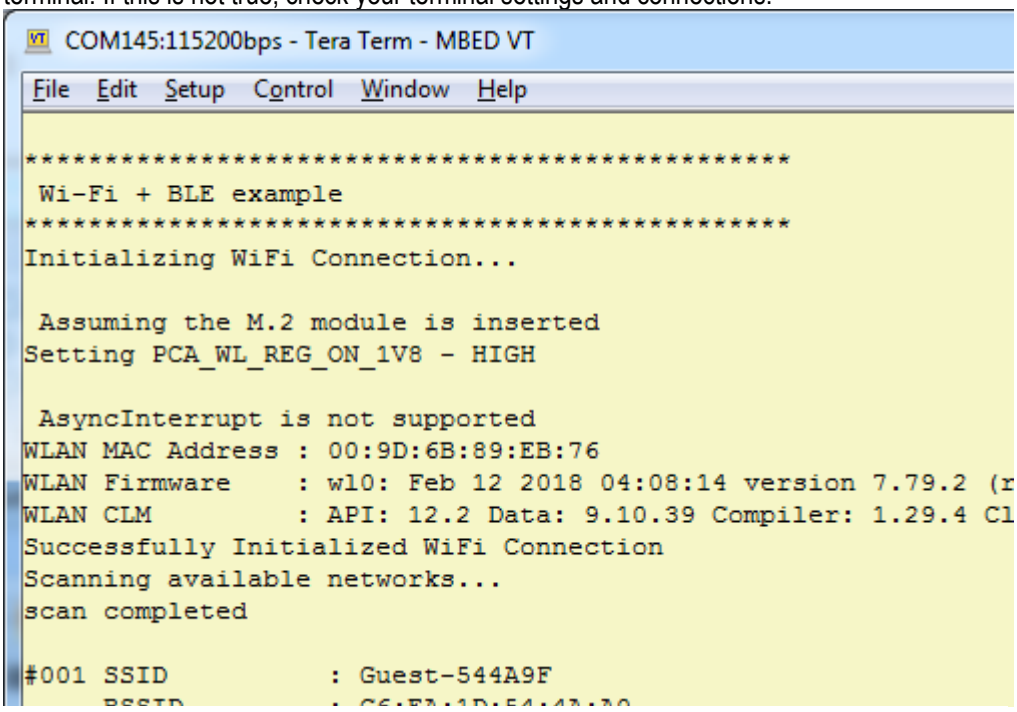
5. Click the "Debug" button in the Quickstart Panel to download the application to the target.

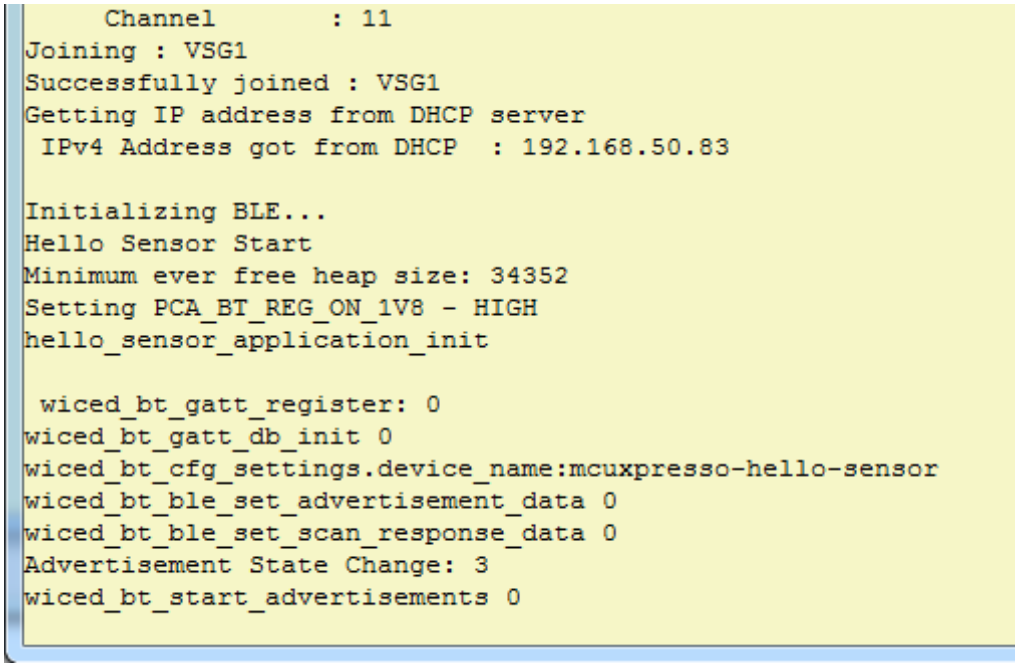


6. The application is then downloaded to the target and automatically runs to the main() function.

7. Run the code by clicking the "Resume" button to start the application.

8. The wiced_ble_4343W application is now running and some information is displayed on the terminal. If this is not true, check your terminal settings and connections.





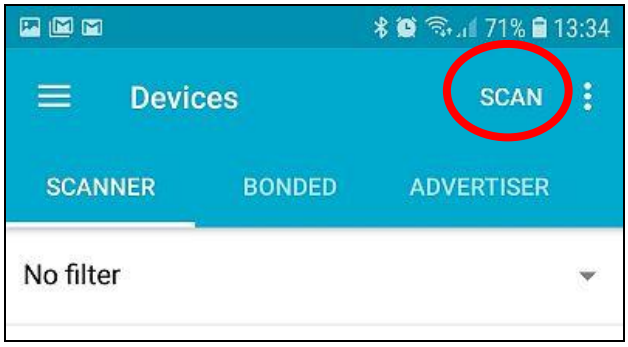
3.4.5 Explanation of the example

The example will start by scanning for nearby Wi-Fi networks and show some information about each network that is found. It will then connection to the Wi-Fi network specified in section 3.4.3 above.

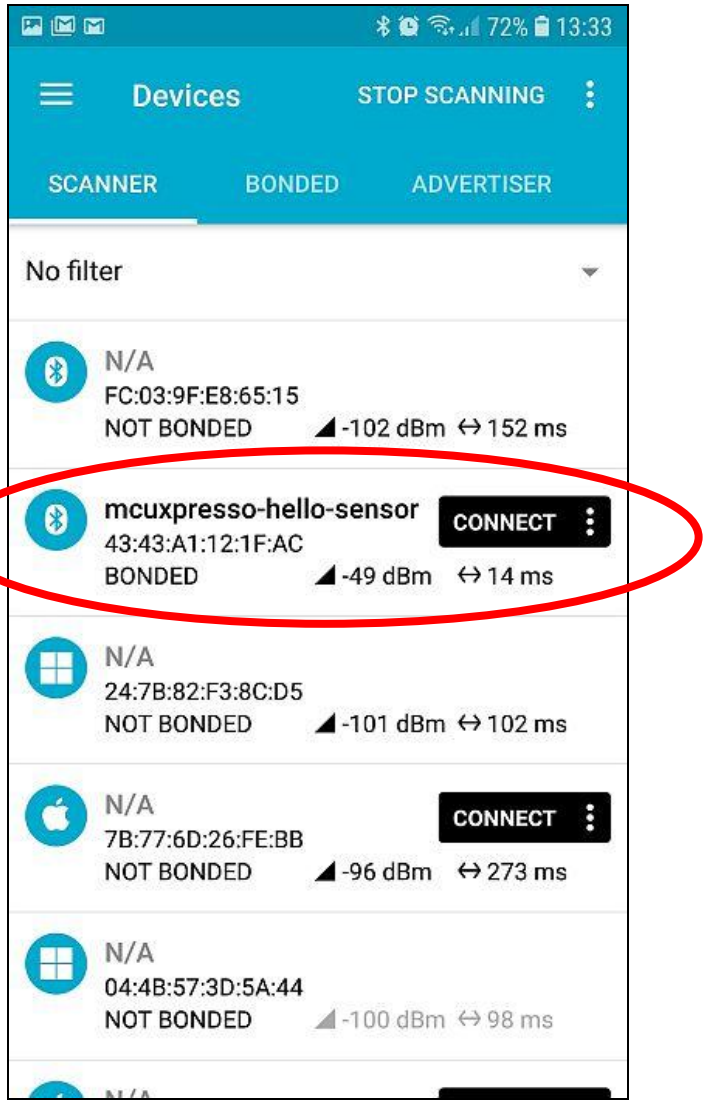
When the network connection has been established it will start BLE advertising with the device name "mcuxpresso-hello-sensor". Download and install one of the following apps on your phone:

1. nRF Connect: https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp
2. LightBlue: https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer

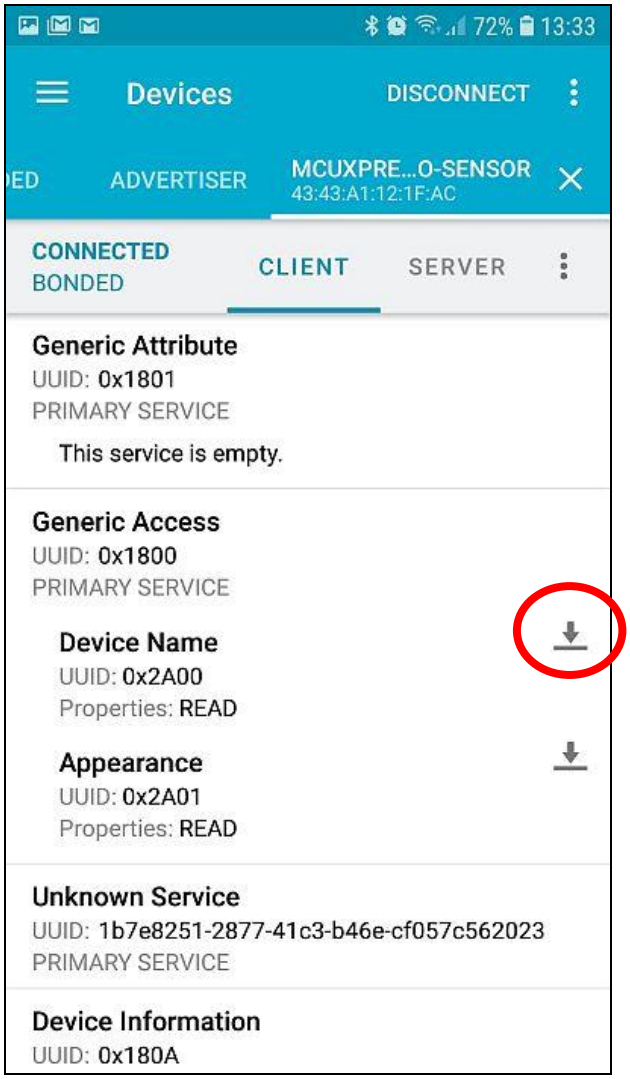
Start the nRF Connect app and press the SCAN button:



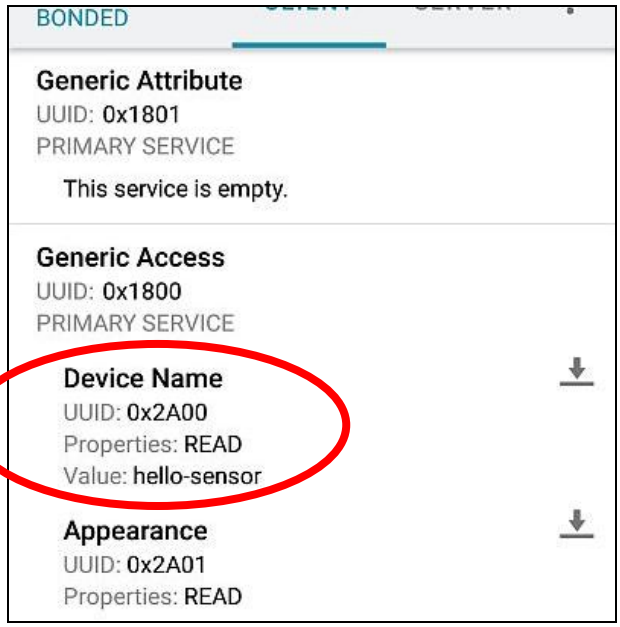
The Bluetooth device should appear like this:



Click CONNECT to get a list of attributes and information like this:



Click the marked button to read the device name. It will look like this:

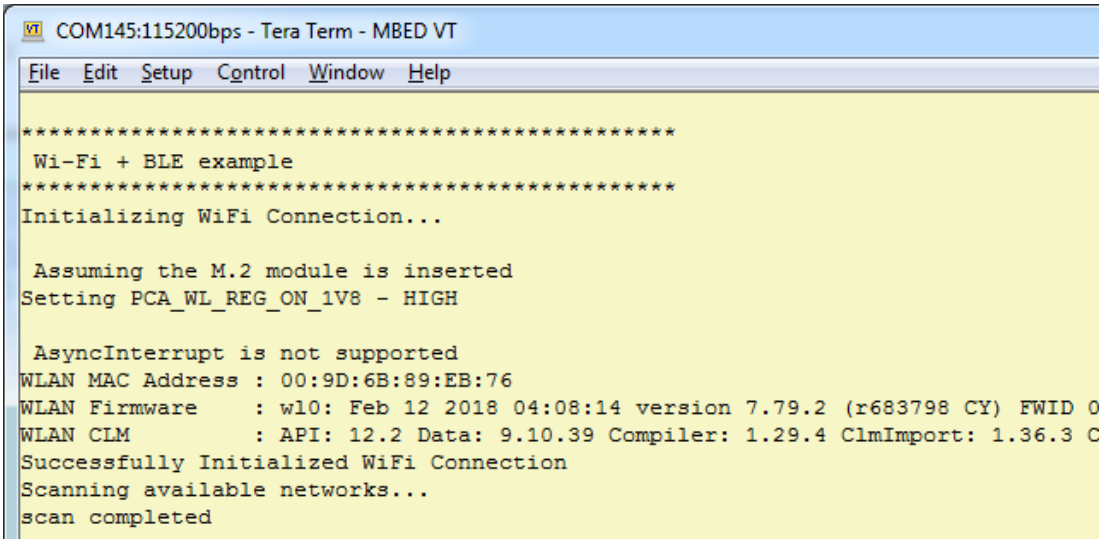


3.4.6 Troubleshooting

Some things to check if the application is not working as expected:

- **Program not starting after being flashed**.
  This is typically because either the flash algorithm/flash size was not changed when importing the project or the project files were not patched.

- **Cannot find M.2 module**.
  A successful detection of the M.2 module looks like this:



  If the M.2 module cannot be detected, check that the module is correctly inserted and that all patches were applied as specified in section 3.3.2 Make sure that the project matches the M.2 module - the wiced_ble_4343W is for 1DX.

- **Cannot find Bluetooth device when searching with the app**.
  A successful initialization of the M.2 module looks like this:

```
Initializing BLE...
Hello Sensor Start
Minimum ever free heap size: 34352
Setting PCA_BT_REG_ON_1V8 - HIGH
hello_sensor_application_init

 wiced_bt_gatt_register: 0
wiced_bt_gatt_db_init 0
wiced_bt_cfg_settings.device_name:mcuxpresso-hello-sensor
wiced_bt_ble_set_advertisement_data 0
wiced_bt_ble_set_scan_response_data 0
Advertisement State Change: 3
wiced_bt_start_advertisements 0
```

  If Bluetooth cannot be initialized, check that the M.2 module is correctly inserted and that all patches were applied as specified in section 3.3.2 Make sure that the project matches the M.2 module - the wiced_ble_4343W is for 1DX.

4 Debug Interface

It is strongly recommended to use a debug/JTAG probe during program development. The low-cost LPC-Link2 is an excellent choice. Keil ULINK2 and ULINKplus, as well as Segger J-LINK, are also excellent debug probes.

There are two debug interface connectors available on the *iMX OEM Carrier board*:

- J10 – this is a Cortex Debug connector. It is a 2x5 pos, 50 mil pitch connector without a shroud. Be careful when inserting the debug probe cable. Position 1 is in specifically marked on the PCB silkscreen. It is located in the lower right corner, see Figure 7 below.
  The connector supports both the SWD and JTAG interfaces.
- J11 – this is an ARM Debug connector. It is a 2x10 pos, 100 mil pitch connector with shroud.

Both connector are defined and supports both the SWD and JTAG type of debug interfaces.

Note that in order to enable the JTAG/SWD interface on the i.MX RT, JP5 shall **not** be shorted/inserted.

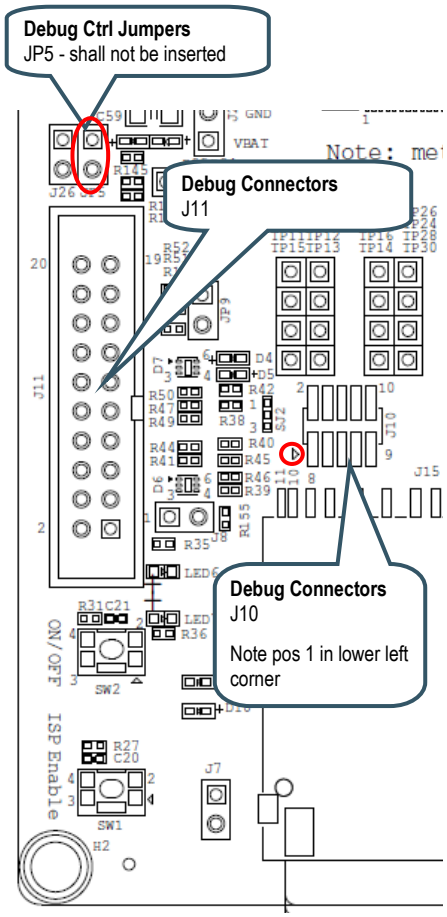


Figure 7 – Debug Interfaces on rev B1 iMX OEM Carrier board

Note that due to the powering sequencing requirements on the i.MX RT family, the debug probe I/O voltage **MUST** follow the i.MX RT I/O voltage.

The debug adapter must not drive any output higher than the Vcc/Vref voltage (and if that voltage is zero, then the debug adapter must not drive any output signal). Vcc/Vref is pin 1 on both J10 and J11.

Make sure the debug probe does not have a fixed output voltage, but rather follow Vcc/Vref. If using LPC-Link2 as debug interface, make sure there is **NO** jumper inserted in JP2 on the LPC-Link2.

4.1 J-LINK/J-TRACE Support

This section describes the steps necessary to get the Segger J-TRACE to work with NXP MCUXpresso and Keil uVision. The same instructions are likely to work for Segger J-LINK as well, but it has not been verified.

4.1.1 Install J-LINK Software

The software for the J-TRACE/J-LINK is installed in a central location (i.e. outside of the IDEs). The latest version (v6.60d) as of January 2020 have support for the iMX RT1062 but needs a configuration change to support the EcoXiP flash memory:

1. Download and install the latest version of Segger's software: https://www.segger.com/downloads/jlink/JLink_Windows.exe (v6.42, or later). The rest of the steps will use `<jdir>` as abbreviation for the folder that the driver was installed in, for example c:\Program Files (x86)\SEGGER\JLink_V642b\
2. Open `<jdir>\JLinkDevices.xml`
3. Search for MCIMXRT1062 to find <Device> entries
4. In each of the <Device> entries, change

   `Loader="Devices/NXP/iMXRT106x/NXP_iMXRT106x_HyperFlash.elf"`

   to

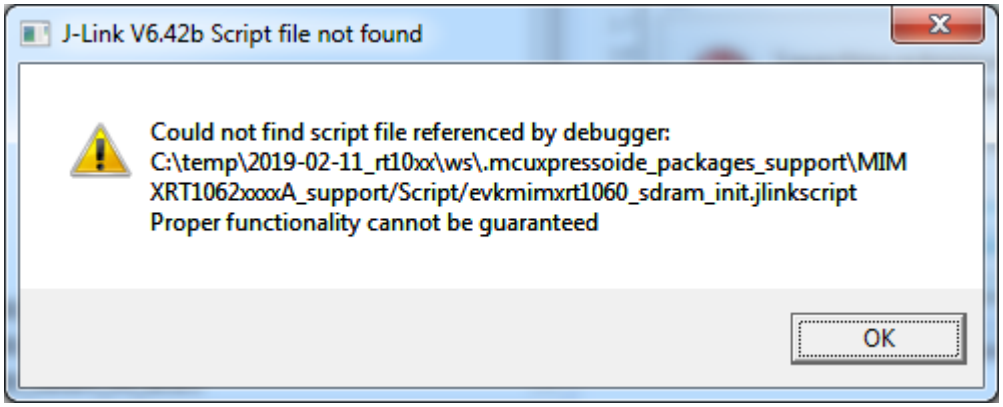
   `Loader="Devices/NXP/iMXRT106x/NXP_iMXRT106x_QSPI.elf"`

   Change the `MaxSize="0x04000000"` to `MaxSize="0x00400000"` and then save the file.

4.1.2 MCUXpresso 10.3.0

MCUXpresso will detect the J-LINK / J-TRACE and configure itself correctly.

If this dialog appears:



Then copy the missing file from the folder that was created in see section 3.3.2 (`c:/temp/ea_files/`)

4.2 Debug Troubleshooting

In some cases the IDE complains about not being able to connect to the target. This is most likely because the program already running on the target is interfering. The solution is to put the hardware in ISP mode before starting the flash/debug operation in the IDE. To do this:

1. Push and hold down the ISP enable button
2. Press the Reset button
3. Release the Reset button
4. Wait 1 seconds
5. Release the ISP enable button

If the LPC-Link2 debugger is used then there are some additional things to note:

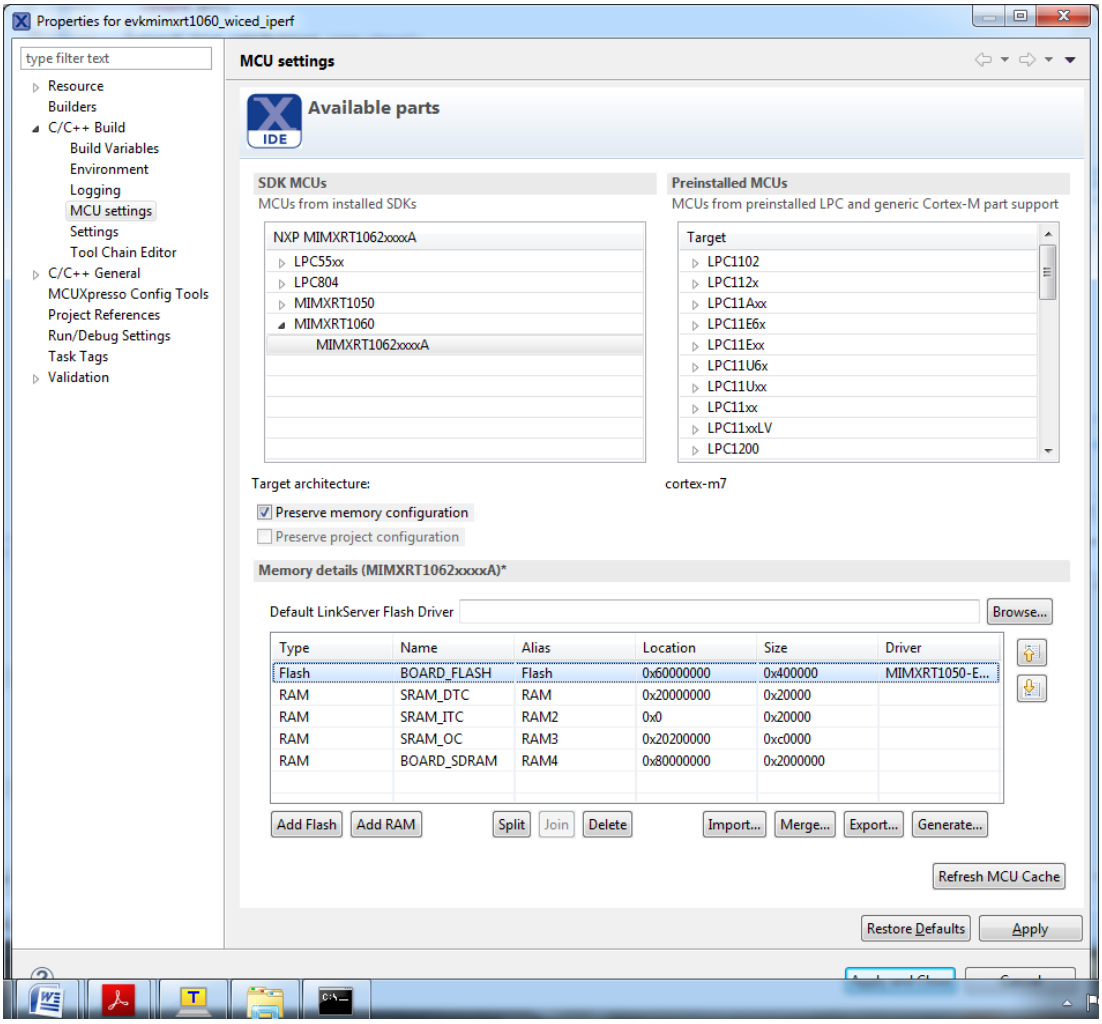
1. Make sure that the J2 jumper on the LPC-Link2 is not inserted. If the jumper is inserted/closed then the target will be powered by the LPC-Link2 which might be too much power for the usb port that the LPC-Link2 is connected to.
2. If the LPC-Link2 is not found by the IDE and you are working on a laptop then try using a powered usb hub instead.
3. The troubleshooting section in this forum post has a couple of additional things to try: https://community.nxp.com/thread/388964
4. There is a *Using and troubleshooting LPC-Link2* in the *Appendix - Additional Hints and Tips* of the User Guide for MCUXpresso IDE. The location of the document is c:\nxp\MCUXpressoIDE_11.0.0_2516\MCUXpresso_IDE_User_Guide.pdf if the IDE was installed with the default settings.

5 Improved IPERF performance

There are some improvements that can be made to the wiced_iperf example to more than double the throughput. The changes include rearranging the memory layout, increasing the TCP receive window and increasing the number of buffers.

> The instructions below were written for an older version of the SDK. As of 2.7.0 the memory ranges and buffers have been modified and as a result the instructions below will result in a **too big binary that cannot be used**. The instructions are left here as pointers for the reader that wants to experiment with performance tweaking themselves.

1) Setup the wiced_iperf example as explained in 3.3 and verify that it works
2) Right click the project and select *Properties…* from the menu. Expand the tree and locate the MCU Settings node:



Delete the SRAM_ITC and then change the sizes of SRAM_DTC and SRAM_OC so that it

looks like below and then save the changes and close the dialog

Type	Name	Alias	Location	Size	Driver
Flash	BOARD_FLASH	Flash	0x60000000	0x400000	MIMXRT1050-E...
RAM	SRAM_DTC	RAM	0x20000000	0x80000	
RAM	SRAM_OC	RAM2	0x20200000	0x80000	
RAM	BOARD_SDRAM	RAM3	0x80000000	0x2000000	

3) Open board/board.c and locate these lines that control the size of the SRAM_DTC (region 5):

```
377
378     /* Region 5 setting: Memory with Normal type, not shareable, outer/inner write back */
379     MPU->RBAR = ARM_MPU_RBAR(5, 0x20000000U);
380     MPU->RASR = ARM_MPU_RASR(0, ARM_MPU_AP_FULL, 0, 0, 1, 1, 0, ARM_MPU_REGION_SIZE_128KB);
```

Replace the ARM_MPU_REGION_SIZE_128KB to ARM_MPU_REGION_SIZE_**512**KB. This aligns the size with the 0x80000 value entered in the previous dialog for SRAM_DTC.

4) Open source/lwipopts.h and locate these lines:

```
109 /* ---------- Pbuf options ---------- */
110 /* PBUF_POOL_SIZE: the number of buffers in the pbuf pool. */
111 #ifndef PBUF_POOL_SIZE
112 #define PBUF_POOL_SIZE 9
113 #endif
```

Change from 9 to 44 buffers in the pool. This is what the increase of SRAM_DTC was used for.

5) In the same file (source/lwipopts.h) locate these lines:

```
155 /* TCP receive window. */
156 #ifndef TCP_WND
157 #define TCP_WND (9 * TCP_MSS)
158 #endif
159
```

Replace the constant 9 with 44

6) Open startup/startup_mimxrt1062.c and locate these lines:

```
424 __attribute__ ((used, section(".isr_vector")))
425 void (* const g_pfnVectors[])(void) = {
426     // Core Level - CM7
427     &_vStackTop,                    // The initial stack pointer
428     ResetISR,                       // The reset handler
429     NMI_Handler,                    // The NMI handler
430     HardFault_Handler,              // The hard fault handler
431     0,                              // Reserved
```

Replace the &_vStackTop with (0x20200000 + 0x1000U) to move the initial stack pointer to the end of the first block of SRAM_OC. It should look like this after the change:

```
424 __attribute__ ((used, section(".isr_vector")))
425 void (* const g_pfnVectors[])(void) = {
426     // Core Level - CM7
427     (0x20200000 + 0x1000U),           // The initial stack pointer
428     ResetISR,                         // The reset handler
429     NMI_Handler,                      // The NMI handler
430     HardFault_Handler,                // The hard fault handler
431     0,                                // Reserved
```

7) In the same file (startup/startup_mimxrt1062.c) locate the ResetISR function:

```
646 __attribute__ ((section(".after_vectors.reset")))
647 void ResetISR(void) {
648
649     // Disable interrupts
650     __asm volatile ("cpsid i");
651
652 #if defined (  USE CMSIS)
```

Insert the following lines after the _asm… line:

```
*((volatile unsigned int *)0x400ac044) = 0x5aaaaaaa;
*((volatile unsigned int *)0x400ac040) = 0x00200006;   // DTC RAM with flexram config
*((volatile unsigned int *)0x400ac038) = 0x00A00000;   // DTC - 512, ITC - 0
```

so that it now looks like this:

```
646 __attribute__ ((section(".after_vectors.reset")))
647 void ResetISR(void) {
648
649     // Disable interrupts
650     __asm volatile ("cpsid i");
651
652     *((volatile unsigned int *)0x400ac044) = 0x5aaaaaaa;
653     *((volatile unsigned int *)0x400ac040) = 0x00200006;   // DTC RAM with flexram config
654     *((volatile unsigned int *)0x400ac038) = 0x00A00000;   // DTC - 512, ITC - 0
655
656 #if defined (  USE CMSIS)
```

8) Compile and run. Compare iperf throughput results with, and without, this patch.