MULTI-MODE WIFI, BLUETOOTH, ZIGBEE

QCA4020: Multi-Mode Dual Band WiFi, Bluetooth 5, and ZigBee (802.15.4)

Purpose of the Document

The purpose of this document is to explain the QCA4020 which is multi-mode dual band WiFi, Bluetooth 5, and ZigBee (802.15.4). This document contains the features of the QCA4020 and how to configure it.

Document History

Version	Author	Date	Description
A	5G HUB	8.16.2021	Initial Document

Table of Contents

1 Introduction

This is a miniPCIe card that supports multi-mode intelligent connectivity. It supports dual-band WiFi, Bluetooth 5, and ZigBee. It is based on Qualcomm QCA4020 System-On-Chip (SoC). It has low power SoC that integrates a Cortex M4F for application processing, Cortex M0 for network stack processing, and a separate processor for Wi-Fi stack designed to enable a highly concurrent multiple radio solution.

The QCA4020 SDK pre-integrates support for advanced security features and multiple software and cloud ecosystems.

Designed to address IoT fragmentation and support for interoperability, this solution is ideal for multiple IoT industries from home control and automation, networking, home entertainment and smart cities.

Feature Highlights

- Multi-mode SoC supporting dual band Wi-Fi, Bluetooth 5, and IEEE 802.15.4 concurrently
- Dedicated processor for Bluetooth LE LC and 15.4 MAC
- Dedicated processor for 802.11 a/b/g/n
- Zigbee 3.0 and OpenThread support
- Isolated power islands for low power operation
- Advanced hardware-based security featuring secure boot, trusted execution environment, encrypted storage, key provisioning and application-level security
- Comprehensive set of peripherals and interfaces: SPI, I2C, UART, HS-UART, ADC and GPIOs
- Integrated sensor hub for post-processing designed to enable low power sensor use cases
- Small package size allows for optimized form factors
- 300+KB RAM reserved for applications
- Bluetooth radio details: v5.0 with PA =+4dBm/+10dBm (for Long Range)
- 802.15.4 radio details: 2006 compliant, 15.4e, 2.4GHz DSSS +4dBm/+21dBm (for Long Range)

Specifications

Wi-Fi

Standards: 802.11a/b/g, 802.11n
Wi-Fi Spectral Bands: 2.4 GHz, 5 GHz
Peak Speed: 150 Mbps
MIMO Configuration: 1x1 (1-stream)

Bluetooth

Bluetooth Specification Version: Bluetooth 5.0
Bluetooth Technology: Qualcomm Bluetooth mesh, Bluetooth Low Energy

802.15.4

LR-WPAN Protocol: Thread, Zigbee

USB

USB Version: USB 2.0

CPU

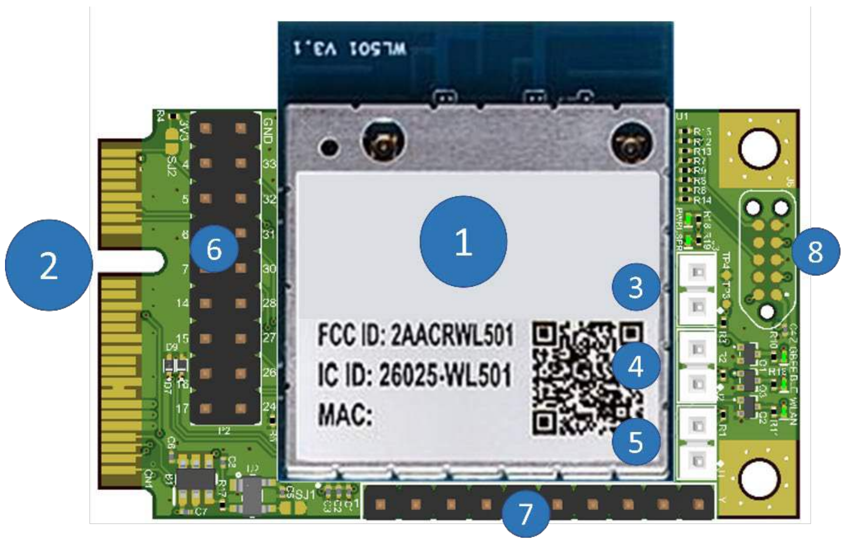
CPU Clock Speed: Up to 128 MHz
CPU Cores: Arm Cortex-M4F CPU, Arm Cortex-M0 CPU, Tensilica Xtensa

Security Support

Security Features: Application-level Security, Hardware-based Crypto Engine, Key Provisioning Security, Qualcomm® Trusted Execution Environment (TEE), Secure Boot, Secure Storage, Software Image Encryption, True Random Number Generator
Wi-Fi Security: WPS Interface

2 QCA4020 miniPCIe Layout

The following figure explains the QCA4020 miniPCIe. It has two headers P1 and P2 which exposes many of the GPIOs and interface of the QCA4020. In addition, it has a JTAG interface for debugging and flashing image. In addition, it has Emergency Download Mode (EDL) jumper header.



1- QCA402 M20 module
2- miniPCIe Interface
3- J3 (EDL)
4- J2 (configure JTAG interface)
5- J1 (Force JTAG) 5- miniPCIe socket
6- P2 Header
7- P1 Header
8- JTAG Interface

Figure 1: QCA4020 miniPCIe Layout.

The following figure shows the size and dimension of the QCA4020 miniPCIe.

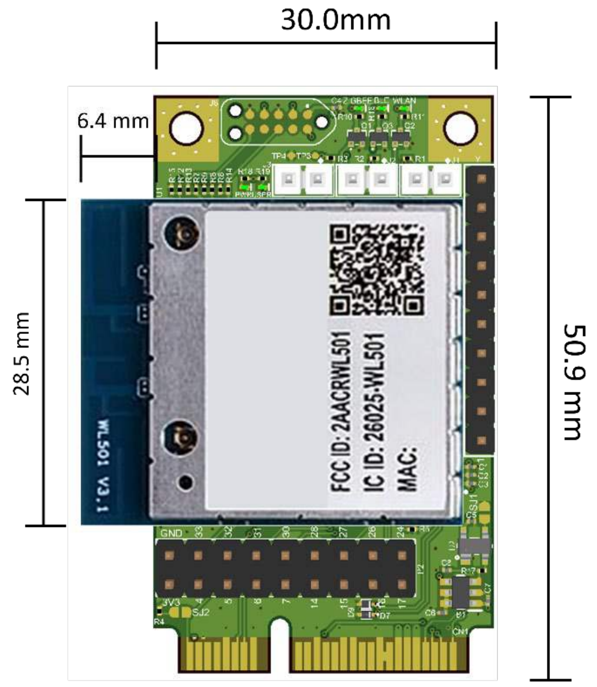


Figure 2: QCA4020 miniPCIe Physical Dimension.

3 QCA4020 miniPCIe Pin Out

The QCA4020 is miniPCIe card and interface. The following figure shows the pin out of the miniPCIe:

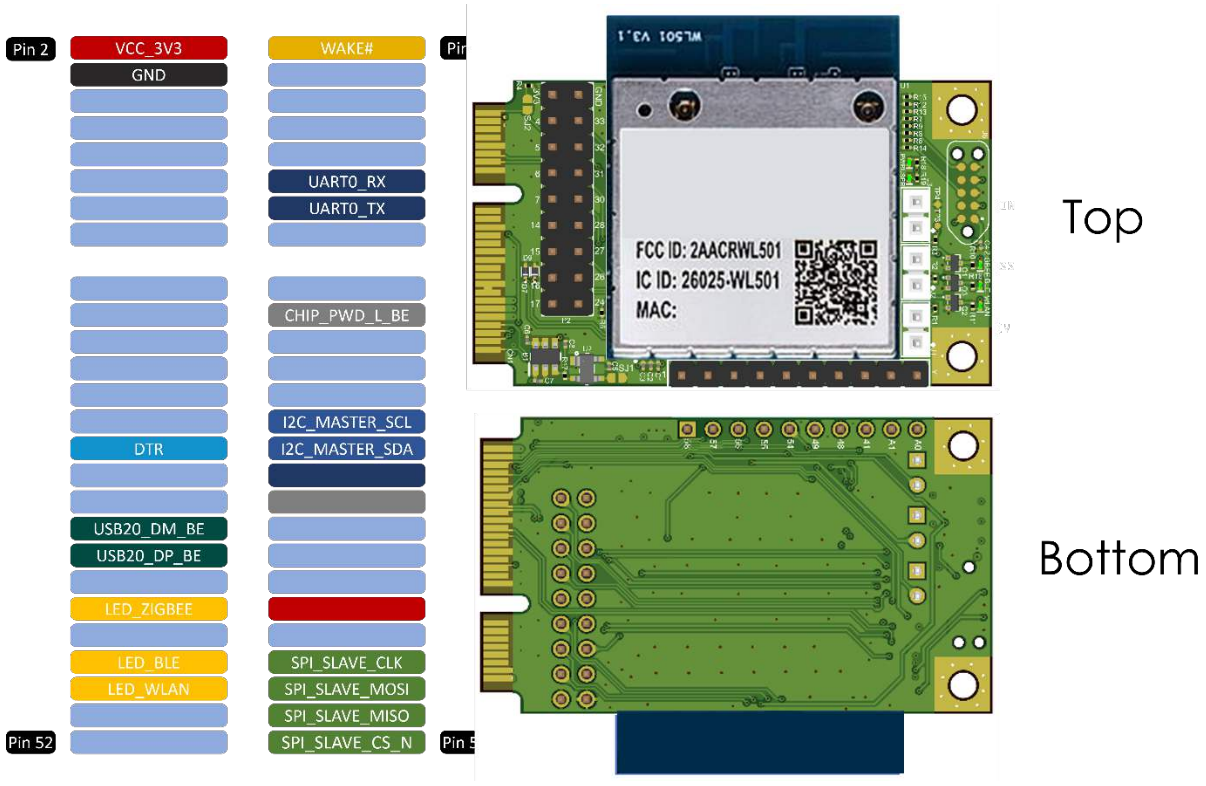


Figure 3: QCA4020 miniPCIe Pin out.

To use the miniPCIe module, you can use USB adaptor (also know as Raspberry PI HAT). The miniPCIe pin mapping as in the table below. If the miniPCIe is used with he Raspberry PI HAT, miniPCIe pins are mapped to Raspberry PI for additional functionality and interface to Raspberry PI.

Table 1: QCA4020 miniPCIe Interface.

PIN #	QCA4020	QCA4020 GPIO PIN#	Raspberry PI PIN#
1	WAKE-UP	GPIO29_BE	
2	VCC_3V3		
3			
4			
5			
6			
7			
8			
9			
10			
11	M0&M4_UART0_RX	GPIO8_BE	GPIO 14 (8)
12			
13	M0&M4_UART0_TX	GPIO9_BE	GPIO 15 (10)
14			

15			
16			
17			
18			
19			
20			
21			
22	CHIP_PWD_L_BE	T4	
23			
24			
25			
26			
27			
28			
29			
30	I2C0_MASTER_SCL	GPIO10_BE	GPIO 19 (35)
31	DTR		33
32	I2C0_MASTER_SDA	GPIO11_BE	GPIO 18 (12)
33			
34			
35			
36	USB20_DM_BE	USB20_DM_BE	
37			
38	USB20_DP_BE	USB20_DP_BE	
39			
40			
41			
42	LED_ZIGBEE	GPIO13_BE	
43			
44			
45	SPI_SLAVE_CLK	GPIO18_BE	GPIO 11 (23)
46	LED_BLE	GPIO60_BE	
47	SPI_SLAVE_MOSI	GPIO23_BE	GPIO 9 (21)
48	LED_WLAN	GPIO12_BE	
49	SPI_SLAVE_MISO	GPIO20_BE	GPIO 10 (19)
50			
51	SPI_SLAVE_CS_N	GPIO19_BE	GPIO 8 (24)
52			

4 How to flash Image to QCA4020 miniPCIe

Flashing the image o the QCA4020 miniPCIe can be done using either of the following two methods:

4.1 Method 1: Through Emergency Download Mode (EDL)

In this step up, user can do the following:

1- Connect J3 (PIN 1 and PIN 2). This is the GPIO22_BE used for EDL.
2- Connect the USB cable between the PC and the USB port of M20.

4.2 Method 2: Through JTAG

In this step up, user can do the following:

1- Connect J1 (PIN 1 and PIN 2). This is the GPIO20_BE which used to force JTAG mode
2- Connect JTAG cable to the JTAG 10-PIN header which includes the signals **TCK, TDI, TDO, TMS**

Follow normal procedure to flash the *.elf file over the JTAG interface.

QCA4020 JTAG pins used as in the following Table:

Table 2: JTAG Interface.

QCA4020 PIN#	JTAG Signal
GPIO_50	JTAG3_BE_TCK
GPIO_51	JTAG3_BE_TDO
GPIO_52	JTAG3_BE_TMS
GPIO_53	JTAG3_BE_TDI

J2 on the QCA4020 is used to configure JTAG. Connecting J2 (PIN 1 and PIN2), force the following JTAG configuration:

Table 3: J2 Setting for JTAG Interface.

Boot_Configure_BE_0GPIO_9_BE	Boot_Configure_BE_1GPIO_25_BE	Boot_Configure_BE_2GPIO_18_BE	JTAG Interface for M4
0	0	1	JTAG in GPIO[53:50]

5 QCA4020 Pin Out

The QCA4020 has abundant number of pins and GPIOs and functionalities which are exposed through P1 and P2 jumper headers. The following table summarizes all pins exposed through P1 and P2 jumper headers and their hardware functionalities.

Table 4: P1 and P2 PIN Mapping.

Header and PIN	GPIO #	SPI or I2C orQSPI	UART	PWMADC/ SenseADC	PTA
P2 - PIN 1	3.3V				
P2 - PIN 2	GPIO4_BE				
P2 - PIN 3	GPIO5_BE				BT_ ACTIVE
P2 - PIN 4	GPIO6_BE				WLAN_ACTIVE
P2 - PIN 5	GPIO7_BE				BT_ PRIORITY
P2 - PIN 6	GPIO14_BE		HS_UART0_DM_CTS		
P2 - PIN 7	GPIO15_BE		HS_UART0_DM_TXD		
P2 - PIN 8	GPIO16_BE	I2C1_Master_SCL	HS_UART0_DM_RFR		BT_ ACTIVE
P2 - PIN 9	GPIO17_BE	I2C1_Master_SDA	HS_UART0_DM_RXD		WLAN_ACTIVE
P2 - PIN 10	GPIO24_BE				
P2 - PIN 11	GPIO26_BE				
P2 - PIN 12	GPIO27_BE				

P2 - PIN 13	GPIO28_BE				
P2 - PIN 14	GPIO30_BE				
P2 - PIN 15	GPIO31_BE				
P2 - PIN 16	GPIO32_BE				
P2 - PIN 17	GPIO33_BE				
P2 - PIN 18	GND				
P1 - PIN 10	A0 (SENSEADC_1_BE)				
P1 - PIN 9	A1 (SENSEADC_0_BE)				
P1 - PIN 8	GPIO_41_BE				
P1 – PIN 7	GPIO48_BE				
P1 - PIN 6	GPIO49_BE				
P1 - PIN 5	GPIO54_BE			SENSEADC2	
P1 - PIN 4	GPIO55_BE			SENSEADC3	
P1 - PIN 3	GPIO56_BE			SENSEADC4	
P1 - PIN 2	GPIO57_BE			SENSEADC5	
P1 - PIN 1	GPIO58_BE			SENSEADC6	

6 Software Tools to be Installed

Please install the following tools

- **Python 2.7.X** (Download Python | Python.org)
- **Eclipse IDE for C/C++** (Eclipse IDE for C/C++ Developers | Eclipse Packages)
  This is a GUI-based integrated development environment
  Supported Version: Oxygen version - Release 4.7.2
- **Java**:
  Eclipse IDE has dependency on Java, JDK 8 or higher
- **OpenOCD** (Download OpenOCD for Windows (gnutoolchains.com))
  version 0.10.0 [2017-06-09]
- **GNU Arm Embedded Toolchain** (GNU ARM Toolchain)
  version 6.x
- **Qualcomm SDK for QCA4020:** The SDK contains sample demo applications with source code to demonstrate different features and technologies that QCA4020 supports.

7 Setting Up the Software Development Environment

7.1 Python

After installing Python, add the path to python.

Example:

If python.exe is in the folder

```
C:\CRMApps\Apps\Python276-64
```

set path as follows:

```
%PATH%=%PATH%:C:\CRMApps\Apps\Python276-64
```

7.2 Java

After installing Java, add path to Java.

Example:

If Java.exe is in the folder

```
C:\ProgramData\Oracle\Java\javapath
```

set path as follows:

```
%PATH%=%PATH%: C:\Program Files\Java\jdk1.8.0_161\bin
```

7.3 OpenOCD

OpenOCD plugin is required to establish the connection between Eclipse IDE and onboard FTDI JTAG debugger. After installation, add the path to OpenOCD.

Example:

If openocd.exe is in the folder

```
C:\Program Files\OpenOCD-20170609\bin
```

set path as follows:

```
set %PATH%=%PATH%:C:\Program Files\OpenOCD-20170609\bin
```

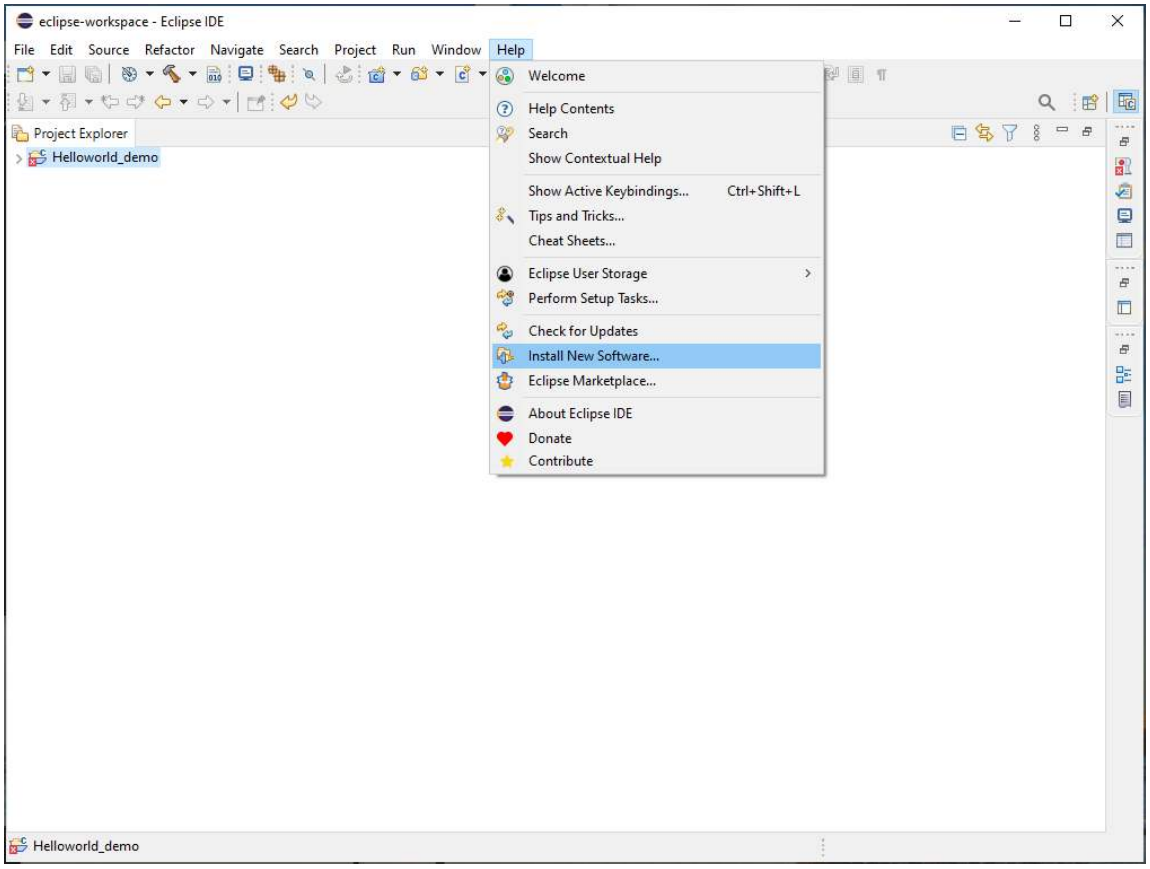
7.4 GNU ARM Toolchain

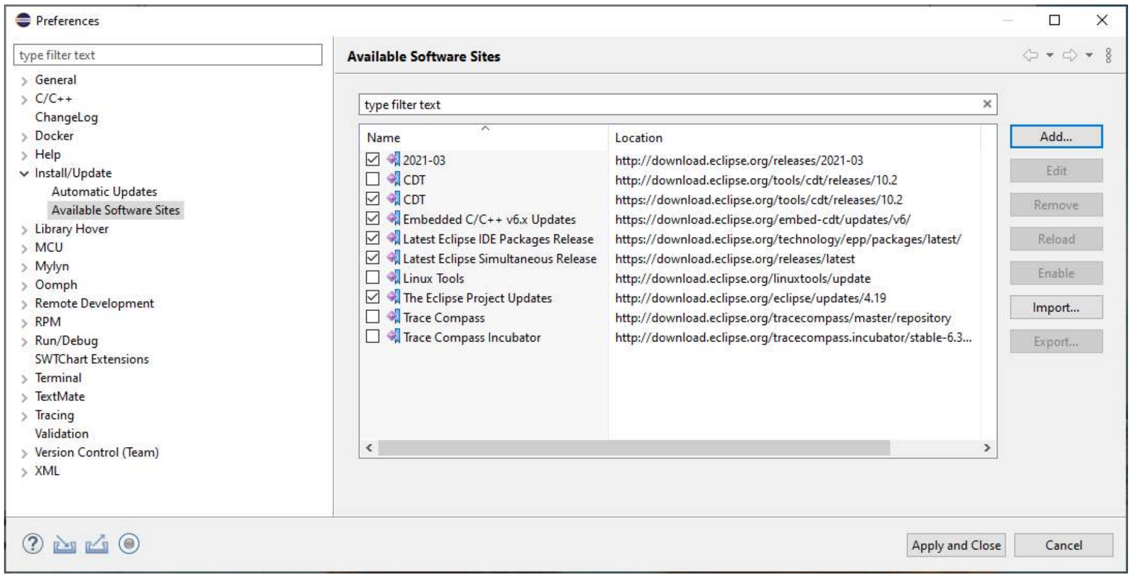
Install the ARM toolchain by running the ".exe" file and make sure you select the option to "Add path to environment variables" during the final step.

7.5 Setup OpenOCD Plugin Usage with Eclipse

Do the following steps:

1- Go to **Help > Install new software** in EclipseIDE and make sure the following plugin are installed and enabled by default.

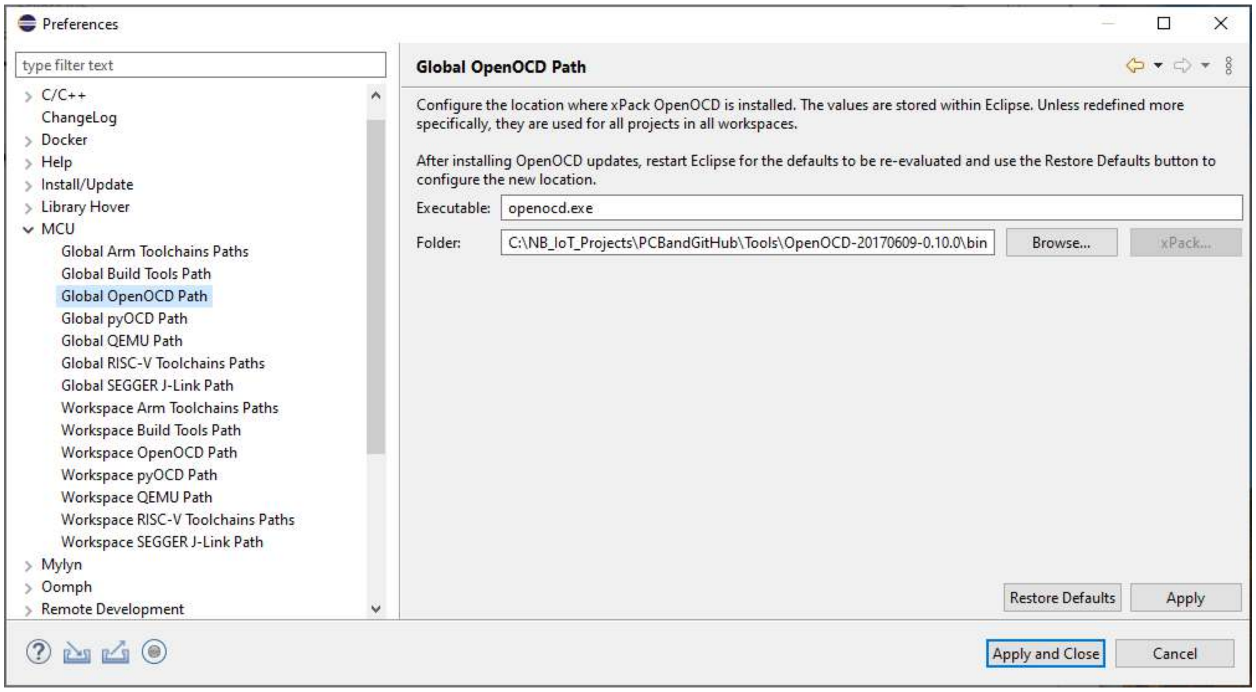




Available Software Sites

Name	Location
☑ 2021-03	http://download.eclipse.org/releases/2021-03
☐ CDT	http://download.eclipse.org/tools/cdt/releases/10.2
☑ CDT	https://download.eclipse.org/tools/cdt/releases/10.2
☑ Embedded C/C++ v6.x Updates	https://download.eclipse.org/embed-cdt/updates/v6/
☑ Latest Eclipse IDE Packages Release	https://download.eclipse.org/technology/epp/packages/latest/
☑ Latest Eclipse Simultaneous Release	https://download.eclipse.org/releases/latest
☐ Linux Tools	http://download.eclipse.org/linuxtools/update
☑ The Eclipse Project Updates	http://download.eclipse.org/eclipse/updates/4.19
☐ Trace Compass	http://download.eclipse.org/tracecompass/master/repository
☐ Trace Compass Incubator	http://download.eclipse.org/tracecompass.incubator/stable-6.3...

2- Set path to openOCD. Restart the Eclipse IDE and under **Window -> preferences**. Set path to openOCD as seen below:



Global OpenOCD Path

Configure the location where xPack OpenOCD is installed. The values are stored within Eclipse. Unless redefined more specifically, they are used for all projects in all workspaces.

After installing OpenOCD updates, restart Eclipse for the defaults to be re-evaluated and use the Restore Defaults button to configure the new location.

Executable: openocd.exe

Folder: C:\NB_IoT_Projects\PCBandGitHub\Tools\OpenOCD-20170609-0.10.0\bin

3- After installing the Qualcomm SDK, **QCA4020 OEM SDK+CDB,** Demos samples are in the following folder:

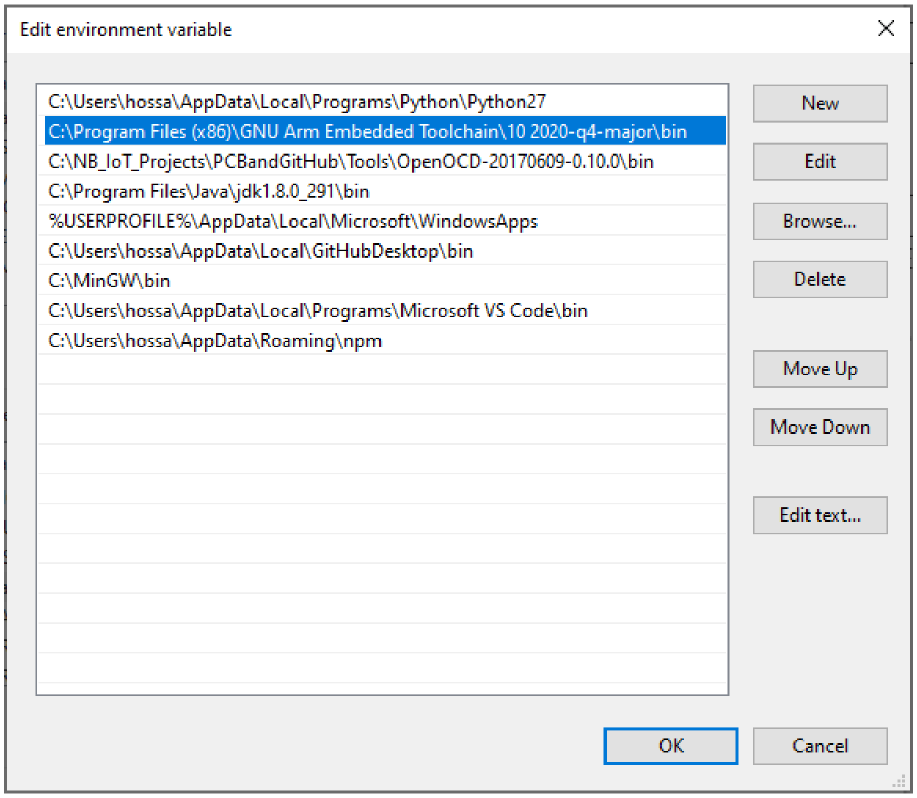
```
target\quart\demo\
```

4- Install the QCA plugin jar file available at

```
<SDK_source>/target/quartz/demo/EclipseSupportFiles
```

- Copy the jar file (QCA402x_plugin.jar) to the “dropin” folder under the Eclipse IDE installed folder.
- Restart the Eclipse IDE if running. To restart Eclipse, click on the **File->Restart** after the plug-in is installed.

5- Make sure all environment variable are set up correctly



```
C:\Users\hossa\AppData\Local\Programs\Python\Python27
C:\Program Files (x86)\GNU Arm Embedded Toolchain\10 2020-q4-major\bin
C:\NB_IoT_Projects\PCBandGitHub\Tools\OpenOCD-20170609-0.10.0\bin
C:\Program Files\Java\jdk1.8.0_291\bin
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\hossa\AppData\Local\GitHubDesktop\bin
C:\MinGW\bin
C:\Users\hossa\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\hossa\AppData\Roaming\npm
```

8 Importing “Hello World” Application

1- Install Eclipse project files for sample demo applications.
To install there is eclipseSupport.bat for Windows and eclipseSupport.sh for Linux in the following folder:

```
<SDK_source>/target
```

2- Open a terminal window and navigate to

```
<SDK_source>/target
```

3- Run the following command

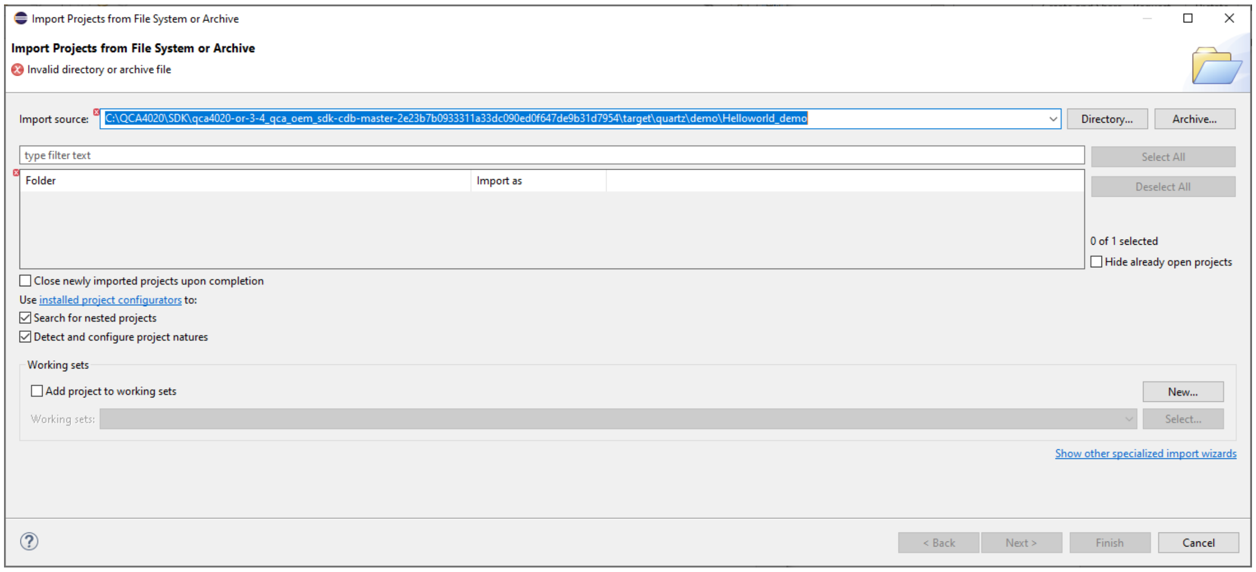
```
sh eclipseSupport.sh
```

Or

```
eclipseSupport.bat
```

After executing the script, the Eclipse project files ***.cproject, .project*** and ***.settings*** folder are updated in the respective folders of the demo application.

4- Open Eclipse application and Go to **File->Open Project** and choose the folder of the "Hello World" application and check the box beside the application. Click **Finish**



Import Projects from File System or Archive

Import Projects from File System or Archive

Invalid directory or archive file

Import source: C:\QCA4020\SDK\qca4020-or-3-4_qca_oem_sdk-cdb-master-2e23b7b0933311a33dc090ed0f647de9b31d7954\target\quartz\demo\Helloworld_demo

0 of 1 selected

☐ Hide already open projects

☐ Close newly imported projects upon completion

Use installed project configurators to:

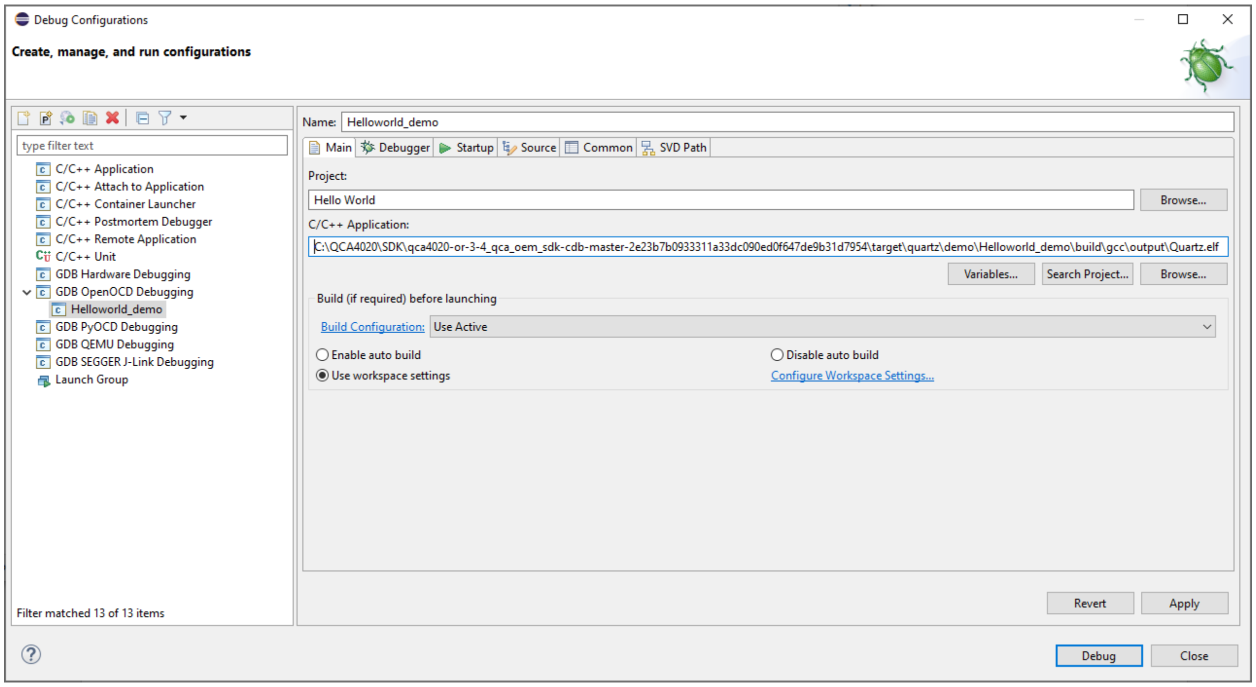
☑ Search for nested projects

☑ Detect and configure project natures

Working sets

☐ Add project to working sets

5- To debug the project, set the Debug Config. Go to **Run->Debug Configuration->GDB OpenOCD**. Set the Application and OpenOCD options as show in the images below. Select Quartz.elf as the C/C++ application. If you have already built the image, select Disable auto build. If image is not built, select Use workspace settings.


Debug Configurations

Create, manage, and run configurations

Name: Helloworld_demo

Project: Hello World

C/C++ Application: C:\QCA4020\SDK\qca4020-or-3-4_qca_oem_sdk-cdb-master-2e23b7b0933311a33dc090ed0f647de9b31d7954\target\quartz\demo\Helloworld_demo\build\gcc\output\Quartz.elf

Build (if required) before launching

Build Configuration: Use Active

Enable auto build | Disable auto build

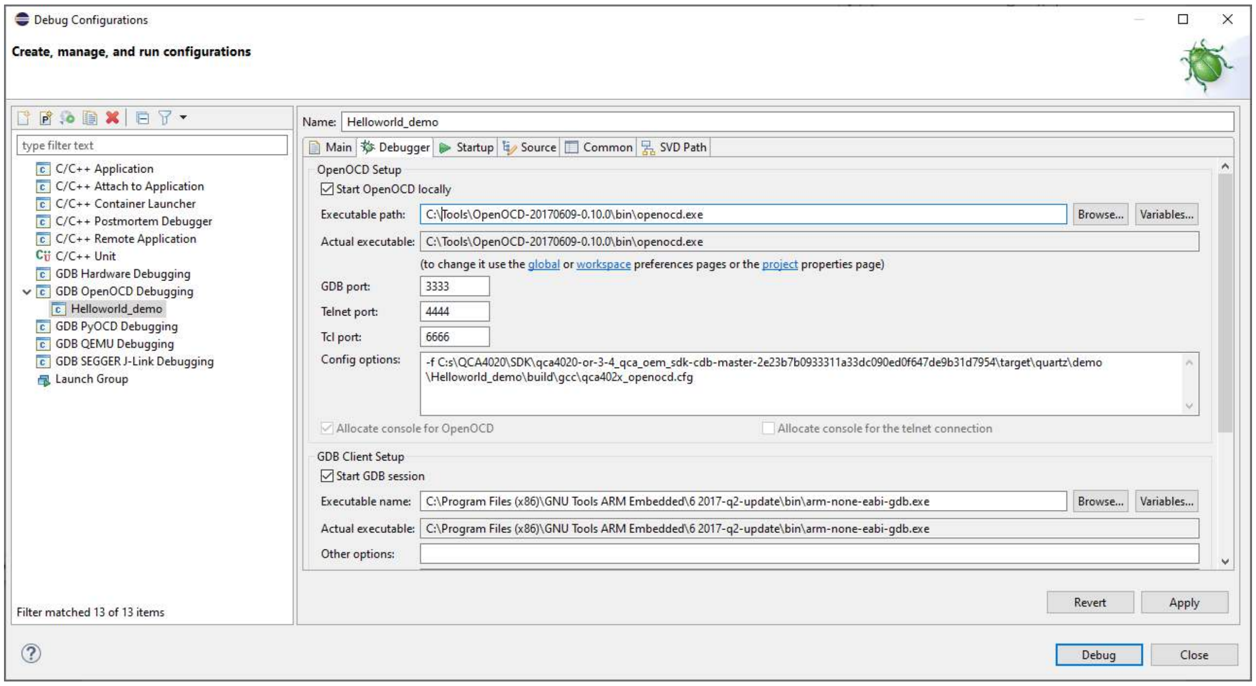
Use workspace settings

Filter matched 13 of 13 items

6- In the Debugger window, provide the openocd executable path with config option

```
-f ${project_loc}\build\gcc\qca402x_openocd.cfg
```

7- Provide the arm-none-eabi-dgb executable path for the GDB client as seen below.


Debug Configurations

Create, manage, and run configurations

Name: Helloworld_demo

OpenOCD Setup

Start OpenOCD locally

Executable path: C:\Tools\OpenOCD-20170609-0.10.0\bin\openocd.exe

Actual executable: C:\Tools\OpenOCD-20170609-0.10.0\bin\openocd.exe

(to change it use the global or workspace preferences pages or the project properties page)

GDB port: 3333

Telnet port: 4444

Tcl port: 6666

Config options: -f C:s\QCA4020\SDK\qca4020-or-3-4_qca_oem_sdk-cdb-master-2e23b7b0933311a33dc090ed0f647de9b31d7954\target\quartz\demo \Helloworld_demo\build\gcc\qca402x_openocd.cfg

Allocate console for OpenOCD | Allocate console for the telnet connection

GDB Client Setup

Start GDB session

Executable name: C:\Program Files (x86)\GNU Tools ARM Embedded\6 2017-q2-update\bin\arm-none-eabi-gdb.exe

Actual executable: C:\Program Files (x86)\GNU Tools ARM Embedded\6 2017-q2-update\bin\arm-none-eabi-gdb.exe

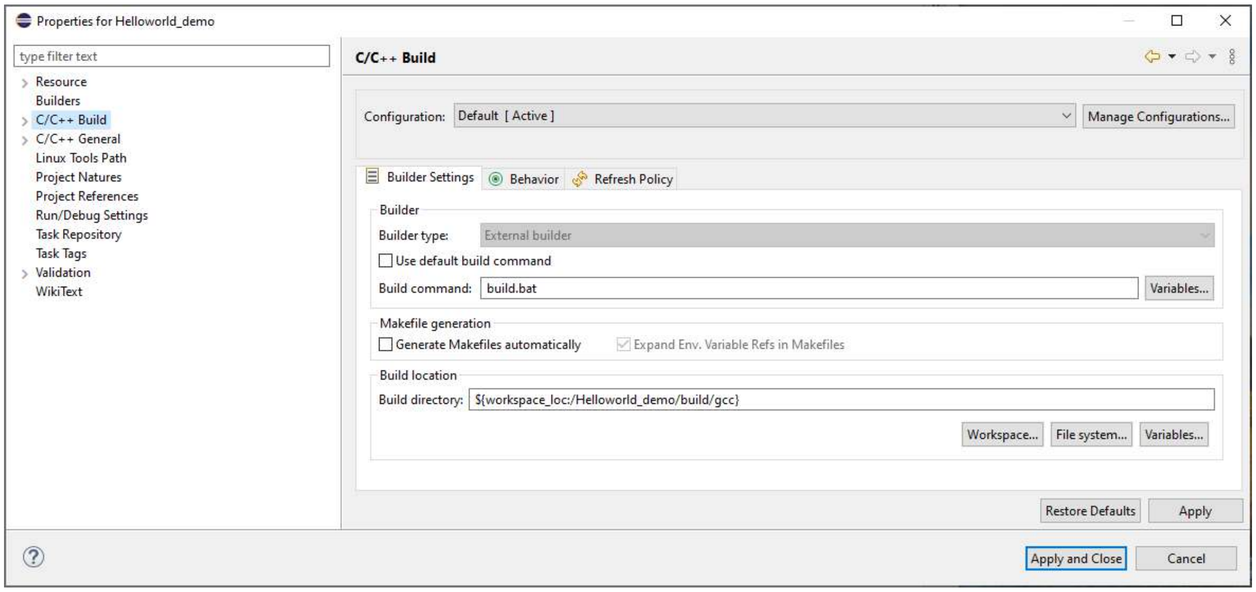
Other options:

Filter matched 13 of 13 items

9 Building "Hello World" Application

To build the application, do the following:

1. Go to Project **properties->Configure build.** Right-click on the project name in project explorer and go to **Properties**
2. Verify the build command and the build directory in the Builder settings tab.
3. Go to C/C++ build and set the build command to "build.bat", and the build directory to "path to source" as in the following screenshots.
4. Click **Apply and Close**



Configuration: Default [Active]

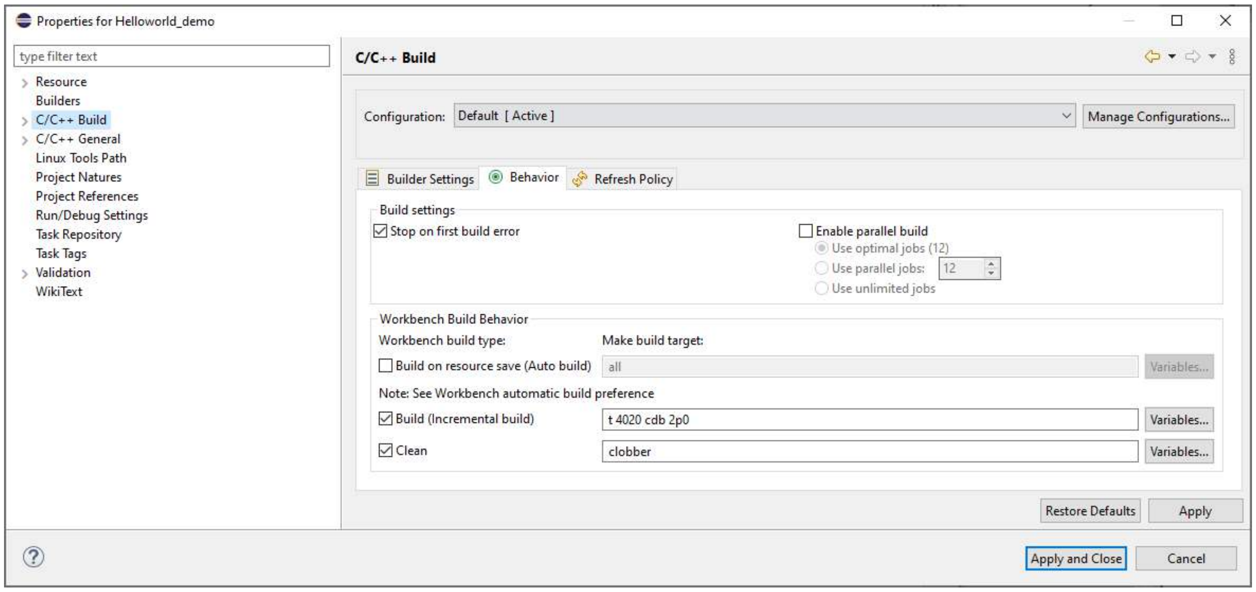
Builder type: External builder

☐ Use default build command

Build command: build.bat

☐ Generate Makefiles automatically ☑ Expand Env. Variable Refs in Makefiles

Build directory: ${workspace_loc:/Helloworld_demo/build/gcc}



Configuration: Default [Active]

☑ Stop on first build error

☐ Enable parallel build

Use optimal jobs (12)

Use parallel jobs: 12

Use unlimited jobs

Workbench build type: Make build target:

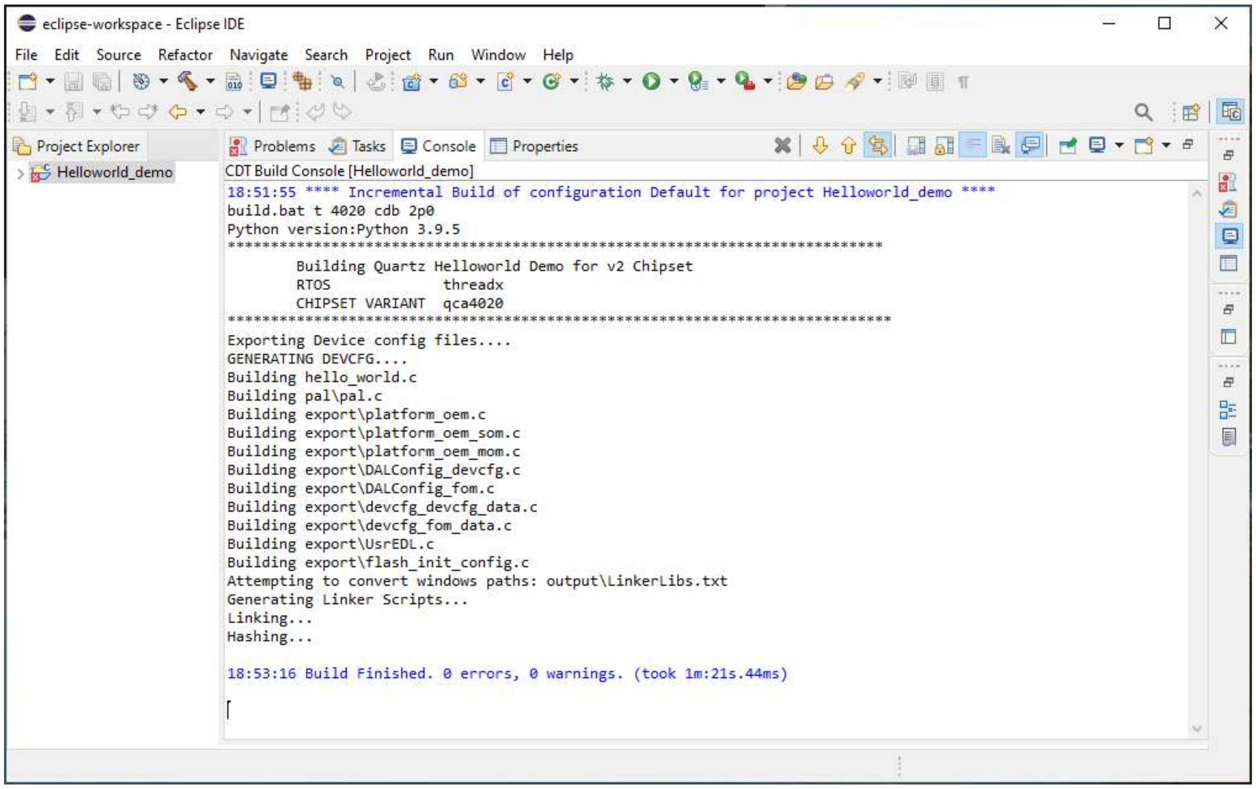
☐ Build on resource save (Auto build) all

Note: See Workbench automatic build preference

☑ Build (Incremental build) t 4020 cdb 2p0

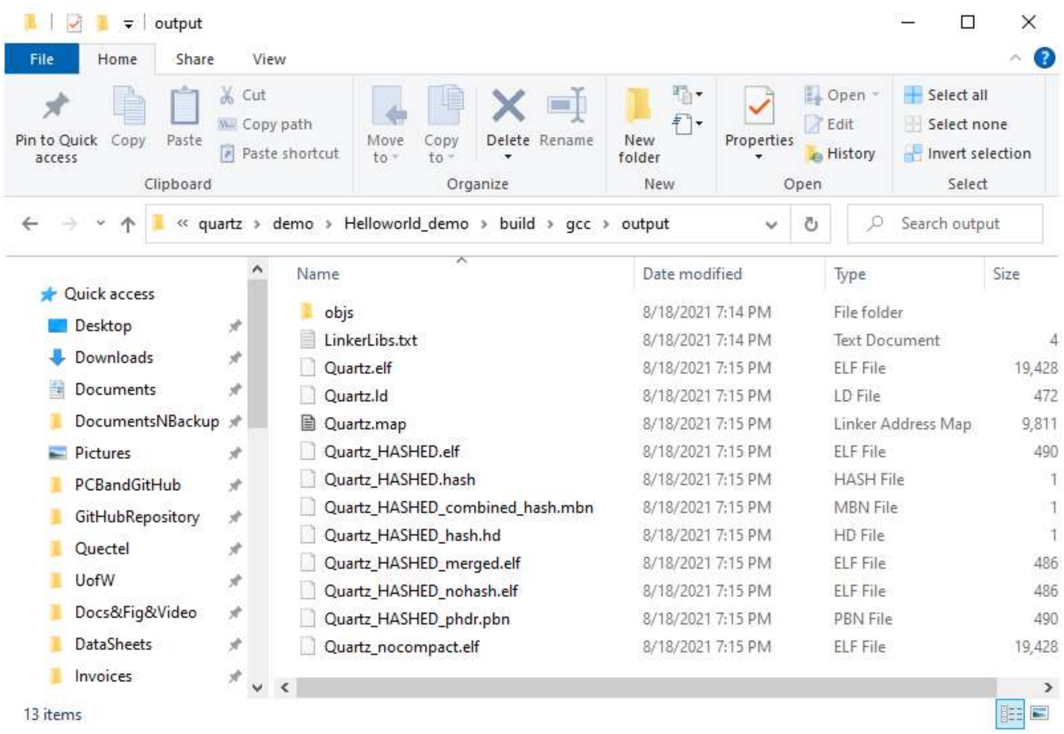
☑ Clean clobber

5. Build the application and you will see the following message in the Console Window


eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Helloworld_demo
Problems Tasks Console Properties
CDT Build Console [Helloworld_demo]

```
18:51:55 **** Incremental Build of configuration Default for project Helloworld_demo ****
build.bat t 4020 cdb 2p0
Python version:Python 3.9.5
*****************************************************************************
        Building Quartz Helloworld Demo for v2 Chipset
        RTOS             threadx
        CHIPSET VARIANT  qca4020
*****************************************************************************
Exporting Device config files....
GENERATING DEVCFG....
Building hello_world.c
Building pal\pal.c
Building export\platform_oem.c
Building export\platform_oem_som.c
Building export\platform_oem_mom.c
Building export\DALConfig_devcfg.c
Building export\DALConfig_fom.c
Building export\devcfg_devcfg_data.c
Building export\devcfg_fom_data.c
Building export\UsrEDL.c
Building export\flash_init_config.c
Attempting to convert windows paths: output\LinkerLibs.txt
Generating Linker Scripts...
Linking...
Hashing...

18:53:16 Build Finished. 0 errors, 0 warnings. (took 1m:21s.44ms)
```

The output and ***.elf** files are generated and available under the **output** folder:



« quartz › demo › Helloworld_demo › build › gcc › output

Name	Date modified	Type	Size
objs	8/18/2021 7:14 PM	File folder	
LinkerLibs.txt	8/18/2021 7:14 PM	Text Document	4
Quartz.elf	8/18/2021 7:15 PM	ELF File	19,428
Quartz.ld	8/18/2021 7:15 PM	LD File	472
Quartz.map	8/18/2021 7:15 PM	Linker Address Map	9,811
Quartz_HASHED.elf	8/18/2021 7:15 PM	ELF File	490
Quartz_HASHED.hash	8/18/2021 7:15 PM	HASH File	1
Quartz_HASHED_combined_hash.mbn	8/18/2021 7:15 PM	MBN File	1
Quartz_HASHED_hash.hd	8/18/2021 7:15 PM	HD File	1
Quartz_HASHED_merged.elf	8/18/2021 7:15 PM	ELF File	486
Quartz_HASHED_nohash.elf	8/18/2021 7:15 PM	ELF File	486
Quartz_HASHED_phdr.pbn	8/18/2021 7:15 PM	PBN File	490
Quartz_nocompact.elf	8/18/2021 7:15 PM	ELF File	19,428

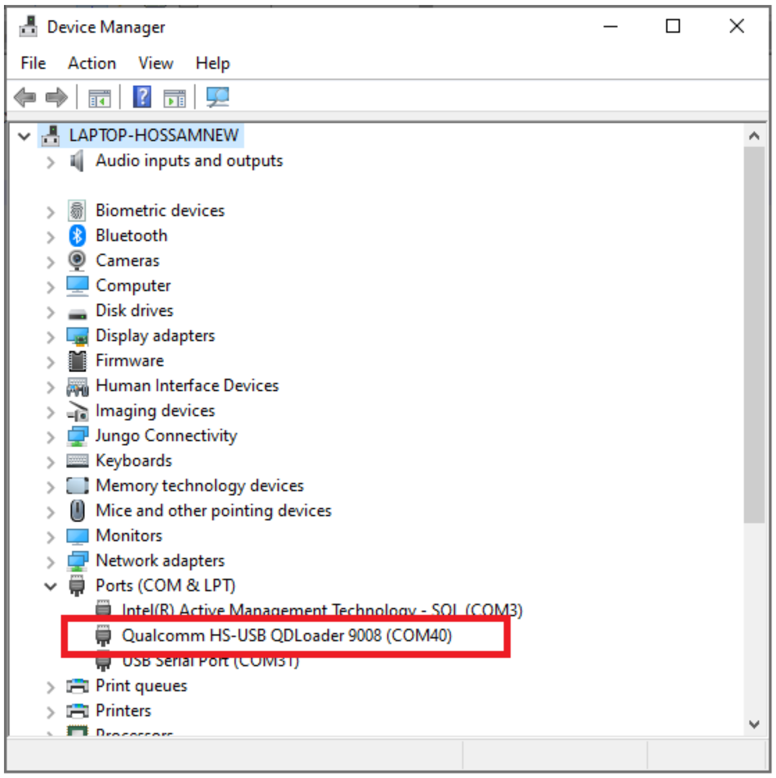
13 items

10 Flash the Application to the QCA4020 miniPCIe using USB

You can flash an application to the QCA4020 miniPCIe over the USB port. To do so, follow the following steps:

1- Connect a jumper on J3 to short-circuit it. This puts the QCA4020 in EDL (Emergency Download Mode) which allows to download an image to the QCA4020 flash.
2- Connect a Micro USB cable between USB port on the USB adaptor (HAT) and a USB port on a Windows PC.

Open the device manager and you shall see Qualcomm DLoader is enumerated on a serial port (i.e., COM40 in this case).



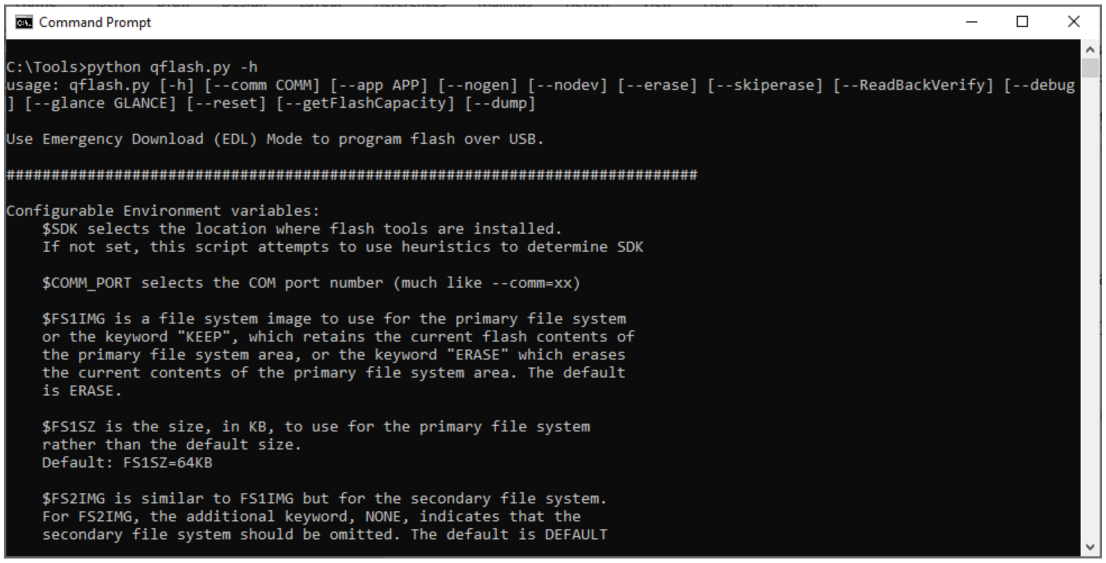
3- Use the python-based tool called **qflash.py** (which is included in the QCA4020 SDK). The tool allows flashing images over USB. The tool is available at the following folder:

```
target\build\tools\flash
```

If the tool is invoked without any optional parameters, it does he following:

a) Generate a default firmware descriptor table.
b) Generate a default partition table.
c) Flash the default sample application ***.elf** files to the qca4020 flash.

When you run this tool with **-h** parameter, you get the following as in this screenshot:



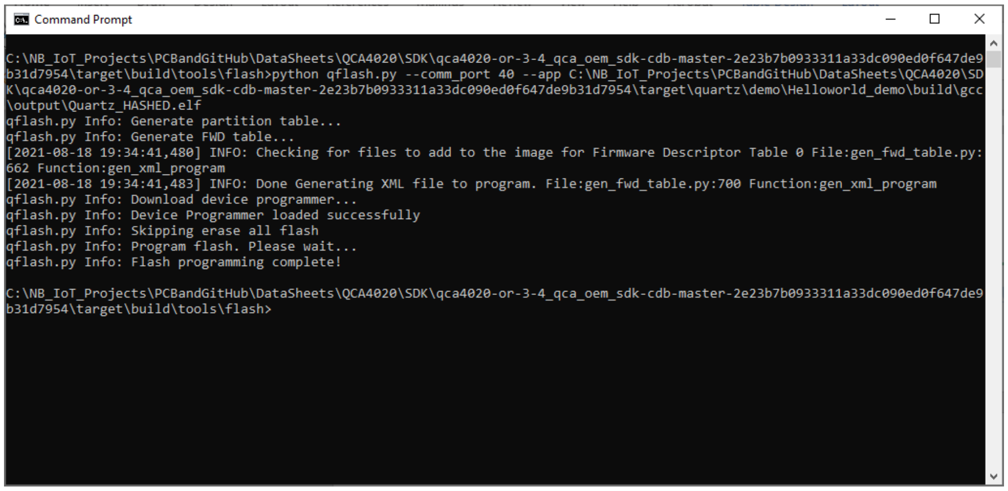
```
C:\Tools>python qflash.py -h
usage: qflash.py [-h] [--comm COMM] [--app APP] [--nogen] [--nodev] [--erase] [--skiperase] [--ReadBackVerify] [--debug
] [--glance GLANCE] [--reset] [--getFlashCapacity] [--dump]

Use Emergency Download (EDL) Mode to program flash over USB.

#############################################################################

Configurable Environment variables:
    $SDK selects the location where flash tools are installed.
    If not set, this script attempts to use heuristics to determine SDK

    $COMM_PORT selects the COM port number (much like --comm=xx)

    $FS1IMG is a file system image to use for the primary file system
    or the keyword "KEEP", which retains the current flash contents of
    the primary file system area, or the keyword "ERASE" which erases
    the current contents of the primary file system area. The default
    is ERASE.

    $FS1SZ is the size, in KB, to use for the primary file system
    rather than the default size.
    Default: FS1SZ=64KB

    $FS2IMG is similar to FS1IMG but for the secondary file system.
    For FS2IMG, the additional keyword, NONE, indicates that the
    secondary file system should be omitted. The default is DEFAULT
```

4- Run the following command to flash the application to the QCA4020

```
python qflash.py --comm_port 40 --app C:\QCA4020\SDK\qca4020-or-3-4_qca_oem_sdk-cdb-master-2e23b7b0933311a33dc090ed0f647de9b31d7954\target\quartz\demo\Helloworld_demo\build\gcc\output\Quartz_HASHED.elf
```

Noe the image to be flashed is the **Quartz_HASHED.elf**.

You shall see the output as in the below screen shoot which indicates the image is successfully flashed to QCA4020.



```
C:\NB_IoT_Projects\PCBandGitHub\DataSheets\QCA4020\SDK\qca4020-or-3-4_qca_oem_sdk-cdb-master-2e23b7b0933311a33dc090ed0f647de9
b31d7954\target\build\tools\flash>python qflash.py --comm_port 40 --app C:\NB_IoT_Projects\PCBandGitHub\DataSheets\QCA4020\SD
K\qca4020-or-3-4_qca_oem_sdk-cdb-master-2e23b7b0933311a33dc090ed0f647de9b31d7954\target\quartz\demo\Helloworld_demo\build\gcc
\output\Quartz_HASHED.elf
qflash.py Info: Generate partition table...
qflash.py Info: Generate FWD table...
[2021-08-18 19:34:41,480] INFO: Checking for files to add to the image for Firmware Descriptor Table 0 File:gen_fwd_table.py:
662 Function:gen_xml_program
[2021-08-18 19:34:41,483] INFO: Done Generating XML file to program. File:gen_fwd_table.py:700 Function:gen_xml_program
qflash.py Info: Download device programmer...
qflash.py Info: Device Programmer loaded successfully
qflash.py Info: Skipping erase all flash
qflash.py Info: Program flash. Please wait...
qflash.py Info: Flash programming complete!

C:\NB_IoT_Projects\PCBandGitHub\DataSheets\QCA4020\SDK\qca4020-or-3-4_qca_oem_sdk-cdb-master-2e23b7b0933311a33dc090ed0f647de9
b31d7954\target\build\tools\flash>
```

11 Flash the Application to the QCA4020 miniPCIe using JTAG

You can use JTAG interface on the QCA4020 miniPCIe to flash the image. A SEGGER J-link hardware is used wit connect and interface to the JTAG.

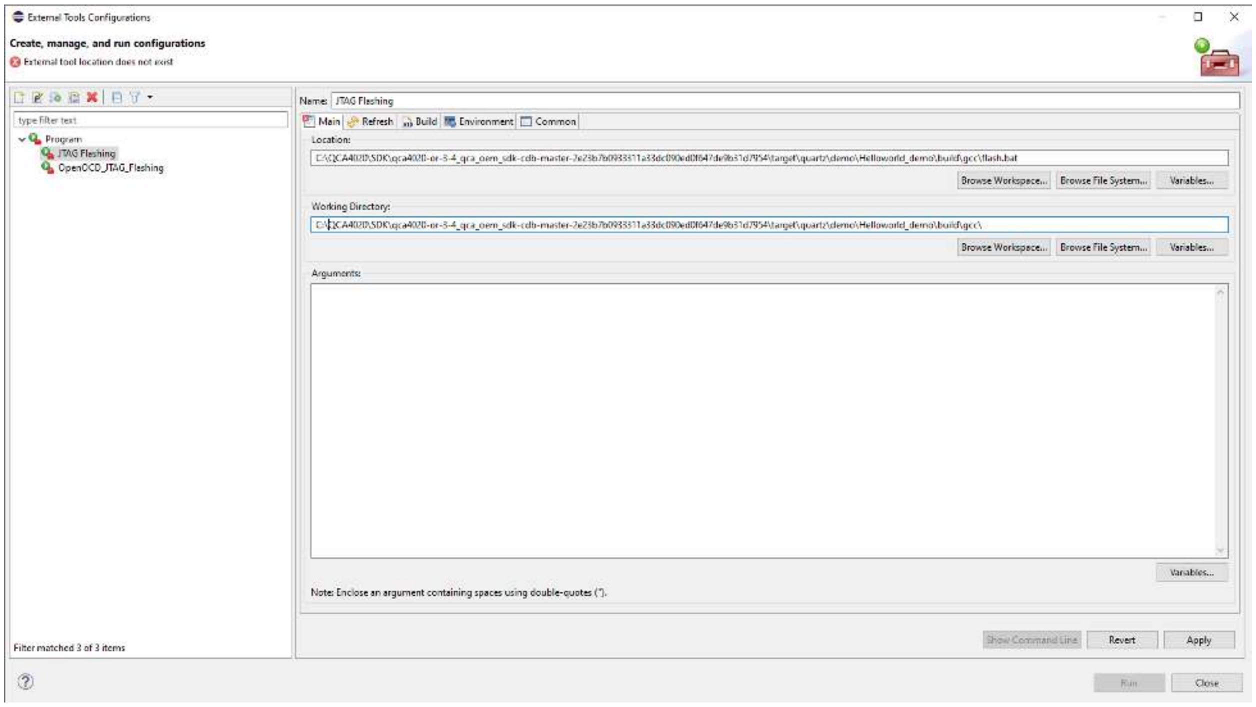
On the QCA4020 miniPCIe, do the following

- Connect J1 (PIN1 and PIN2). This force JTAG mode
- Connect J3 (PIN1 and PIN2). This enable JTAG interface

On Windows environment variables, add the following environment variable:

- **JLINK_PATH** set it to J-link GDB server executable file (e.g., **C:\Program Files (x86)\SEGGER\JLink**)
- **CLIENT_PATH** set it to ARM GNU tools (e.g., **C:\Program Files (x86)\GNU Arm Embedded Toolchain\10 2020-q4-major\bin**)

Using Eclipse IDE, go to **Run->External Tools->External Tools Configurations** and add a new configuration. Point to the **flash.bat** file in the SDK which is used to configure and launch GDB server. Click **Apply** and then **Run**.



External tool location does not exist

Name: JTAG Flashing

Location:
C:\QCA4020\SDK\qca4020-or-3-4_qca_oem_sdk-cdb-master-2e23b7b0933311a33dc090ed0f647de9b31d7954\target\quartz\demo\Helloworld_demo\build\gcc\flash.bat

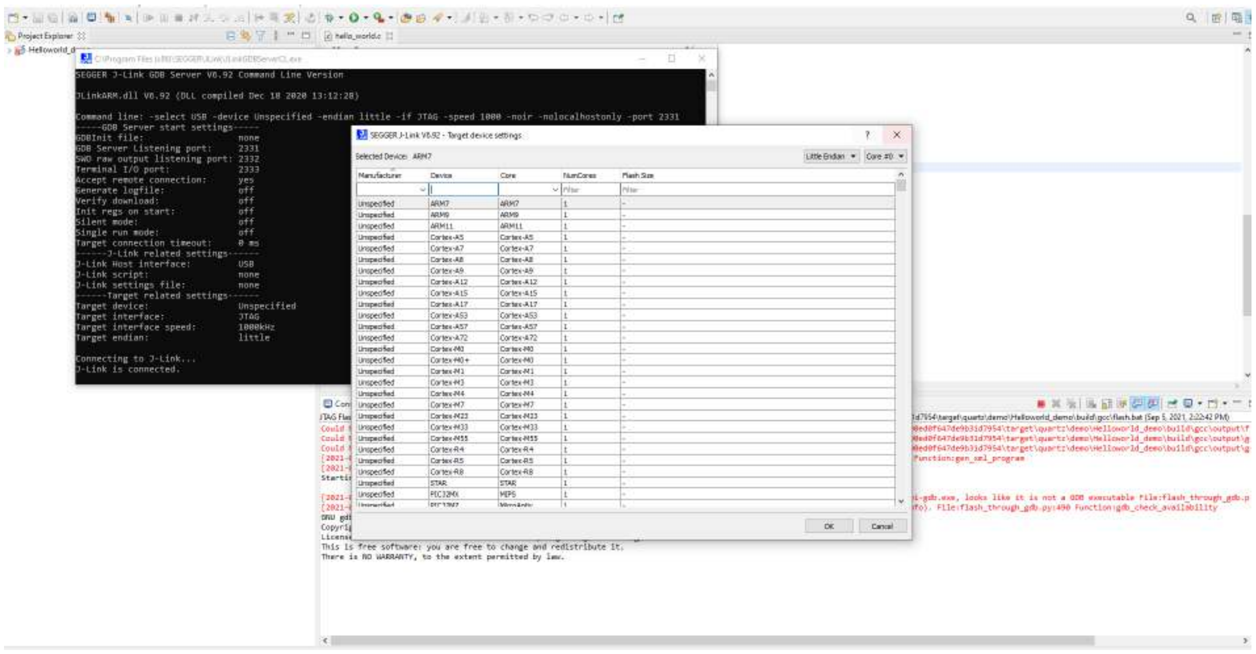
Working Directory:
C:\QCA4020\SDK\qca4020-or-3-4_qca_oem_sdk-cdb-master-2e23b7b0933311a33dc090ed0f647de9b31d7954\target\quartz\demo\Helloworld_demo\build\gcc\

Arguments:

Note: Enclose an argument containing spaces using double-quotes (").

Filter matched 3 of 3 items

You will see the following screenshot and prompted to select device or core. Choose **Cortex-M4**. Click **OK**.



```
SEGGER J-Link GDB Server V6.92 Command Line Version

JLinkARM.dll V6.92 (DLL compiled Dec 18 2020 13:12:28)

Command line: -select USB -device Unspecified -endian little -if JTAG -speed 1000 -noir -nolocalhostonly -port 2331
-----GDB Server start settings-----
GDBInit file:                  none
GDB Server Listening port:     2331
SWO raw output listening port: 2332
Terminal I/O port:             2333
Accept remote connection:      yes
Generate logfile:              off
Verify download:               off
Init regs on start:            off
Silent mode:                   off
Single run mode:               off
Target connection timeout:     0 ms
------J-Link related settings------
J-Link Host interface:         USB
J-Link script:                 none
J-Link settings file:          none
------Target related settings------
Target device:                 Unspecified
Target interface:              JTAG
Target interface speed:        1000kHz
Target endian:                 little

Connecting to J-Link...
J-Link is connected.
```

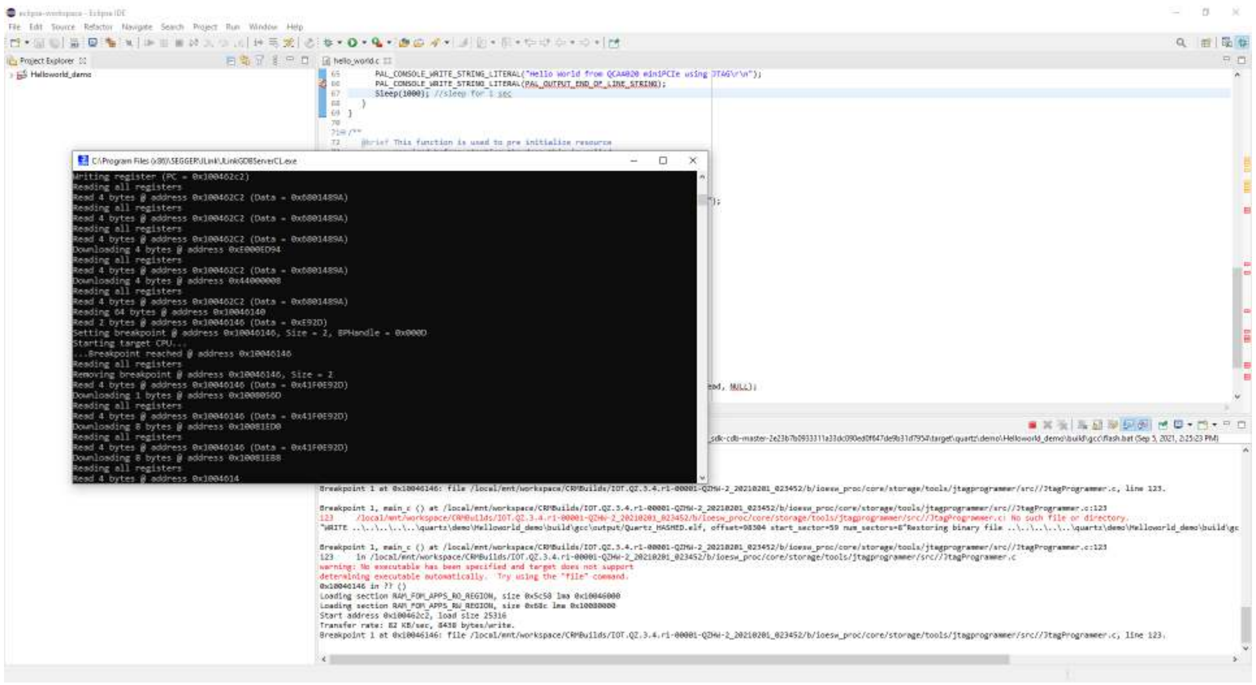
SEGGER J-Link V6.92 - Target device settings

Selected Device: ARM7

Manufacturer	Device	Core	NumCores	Flash Size
Unspecified	ARM7	ARM7	1	-
Unspecified	ARM9	ARM9	1	-
Unspecified	ARM11	ARM11	1	-
Unspecified	Cortex-A5	Cortex-A5	1	-
Unspecified	Cortex-A7	Cortex-A7	1	-
Unspecified	Cortex-A8	Cortex-A8	1	-
Unspecified	Cortex-A9	Cortex-A9	1	-
Unspecified	Cortex-A12	Cortex-A12	1	-
Unspecified	Cortex-A15	Cortex-A15	1	-
Unspecified	Cortex-A17	Cortex-A17	1	-
Unspecified	Cortex-A53	Cortex-A53	1	-
Unspecified	Cortex-A57	Cortex-A57	1	-
Unspecified	Cortex-A72	Cortex-A72	1	-
Unspecified	Cortex-M0	Cortex-M0	1	-
Unspecified	Cortex-M0+	Cortex-M0	1	-
Unspecified	Cortex-M1	Cortex-M1	1	-
Unspecified	Cortex-M3	Cortex-M3	1	-
Unspecified	Cortex-M4	Cortex-M4	1	-
Unspecified	Cortex-M7	Cortex-M7	1	-
Unspecified	Cortex-M23	Cortex-M23	1	-
Unspecified	Cortex-M33	Cortex-M33	1	-
Unspecified	Cortex-M55	Cortex-M55	1	-
Unspecified	Cortex-R4	Cortex-R4	1	-
Unspecified	Cortex-R5	Cortex-R5	1	-
Unspecified	Cortex-R8	Cortex-R8	1	-
Unspecified	STAR	STAR	1	-
Unspecified	PIC32MX	MIPS	1	-

```
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Eclipse will start flashing the image to the QCA4020 miniPCIe over JTAG. You will see the following screenshot.



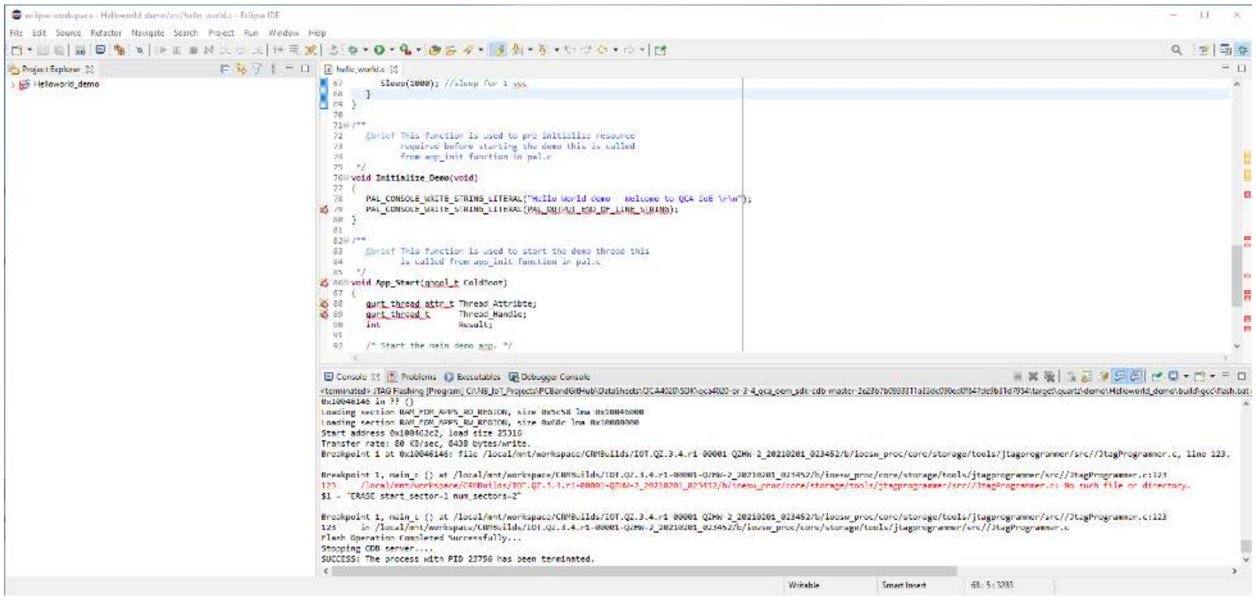
```
        PAL_CONSOLE_WRITE_STRING_LITERAL("Hello World from QCA4020 miniPCIe using JTAG\r\n");
        PAL_CONSOLE_WRITE_STRING_LITERAL(PAL_OUTPUT_END_OF_LINE_STRING);
        Sleep(1000); //sleep for 1 sec
    }
}

/**
   @brief This function is used to pre initialize resource
```

```
Writing register (PC = 0x100462c2)
Reading all registers
Read 4 bytes @ address 0x100462C2 (Data = 0x6801489A)
Reading all registers
Read 4 bytes @ address 0x100462C2 (Data = 0x6801489A)
Reading all registers
Read 4 bytes @ address 0x100462C2 (Data = 0x6801489A)
Downloading 4 bytes @ address 0xE000ED94
Reading all registers
Read 4 bytes @ address 0x100462C2 (Data = 0x6801489A)
Downloading 4 bytes @ address 0x44000008
Reading all registers
Read 4 bytes @ address 0x100462C2 (Data = 0x6801489A)
Reading 64 bytes @ address 0x10046140
Read 2 bytes @ address 0x10046146 (Data = 0xE92D)
Setting breakpoint @ address 0x10046146, Size = 2, BPHandle = 0x000D
Starting target CPU...
...Breakpoint reached @ address 0x10046146
Reading all registers
Removing breakpoint @ address 0x10046146, Size = 2
Read 4 bytes @ address 0x10046146 (Data = 0x41F0E92D)
Downloading 1 bytes @ address 0x1000056D
Reading all registers
Read 4 bytes @ address 0x10046146 (Data = 0x41F0E92D)
Downloading 8 bytes @ address 0x10081ED0
Reading all registers
Read 4 bytes @ address 0x10046146 (Data = 0x41F0E92D)
Downloading 8 bytes @ address 0x10081EB8
Reading all registers
Read 4 bytes @ address 0x1004614
```

```
Breakpoint 1 at 0x10046146: file /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgrammer.c, line 123.

Breakpoint 1, main_c () at /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgrammer.c:123
123     /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgrammer.c: No such file or directory.
"WRITE ..\..\..\..\..\quartz\demo\Helloworld_demo\build\gcc\output/Quartz_HASHED.elf, offset=98304 start_sector=59 num_sectors=8"Restoring binary file ..\..\..\..\..\quartz\demo\Helloworld_demo\build\gc

Breakpoint 1, main_c () at /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgrammer.c:123
123     in /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgramer.c
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x10046146 in ?? ()
Loading section RAM_FOM_APPS_RO_REGION, size 0x5c58 lma 0x10046000
Loading section RAM_FOM_APPS_RW_REGION, size 0x68c lma 0x10000000
Start address 0x100462c2, load size 25316
Transfer rate: 82 KB/sec, 8438 bytes/write.
Breakpoint 1 at 0x10046146: file /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgrammer.c, line 123.
```

When flashing is completed. You will see the following screen shoot with the output:

```
Flash Operation Completed Successfully...
Stopping GDB server....
SUCCESS: The process with PID 23756 has been terminated.
```



```
        Sleep(1000); //sleep for 1 sec
    }
}

/**
   @brief This function is used to pre initialize resource
          required before starting the demo this is called
          from app_init function in pal.c
 */
void Initialize_Demo(void)
{
   PAL_CONSOLE_WRITE_STRING_LITERAL("Hello World demo   Welcome to QCA IoE \r\n");
   PAL_CONSOLE_WRITE_STRING_LITERAL(PAL_OUTPUT_END_OF_LINE_STRING);
}

/**
   @brief This function is used to start the demo thread this
          is called from app_init function in pal.c
 */
void App_Start(qbool_t ColdBoot)
{
   qurt_thread_attr_t Thread_Attribte;
   qurt_thread_t      Thread_Handle;
   int                Result;

   /* Start the main demo app. */
```

```
<terminated> JTAG Flashing [Program] C:\NB_IoT_Projects\PCBandGitHub\DataSheets\QCA4020\SDK\qca4020-or-3-4_qca_oem_sdk-cdb-master-2e23b7b0983311a33dc090ed0f847de9b31d7954\target\quartz\demo\Helloworld_demo\build\gcc\flash.bat
0x10046146 in ?? ()
Loading section RAM_FOM_APPS_RO_REGION, size 0x5c58 lma 0x10046000
Loading section RAM_FOM_APPS_RW_REGION, size 0x60c lma 0x10000000
Start address 0x100462c2, load size 25316
Transfer rate: 80 KB/sec, 8438 bytes/write.
Breakpoint 1 at 0x10046146: file /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgrammer.c, line 123.

Breakpoint 1, main_c () at /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgrammer.c:123
123     /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgrammer.c: No such file or directory.
$1 = "ERASE start_sector=1 num_sectors=2"

Breakpoint 1, main_c () at /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgrammer.c:123
123     in /local/mnt/workspace/CRMBuilds/IOT.QZ.3.4.r1-00001-QZHW-2_20210201_023452/b/ioesw_proc/core/storage/tools/jtagprogrammer/src//JtagProgrammer.c
Flash Operation Completed Successfully...
Stopping GDB server....
SUCCESS: The process with PID 23756 has been terminated.
```

After finishing flashing the image, make sure to remove J1 & J3 on QCA4020 miniPCIe to exit JTAG mode.

12 Connecting QCA4020 to Serial Terminal Through UART

You can use the QCA4020 miniPCIe when mounted on USB adaptor (HAT) with any device that has UART (Tx/Rx) interface such as a Raspberry PI board, Arduino board or any other hardware board, with UART interface.

You can also use and connect the QCA4020 miniPCIe/USB adaptor (HAT) to a USB port in the computer through a USB-to-UART cable. You can use cable such as this one: USB TO UART TTL (Wires) Serial Cable (PL2303HX) MCP00102W Programmer Arduino Compatible in Elecrow bazaar!

Such a cable has four wires colored as follow:

Red: +5V

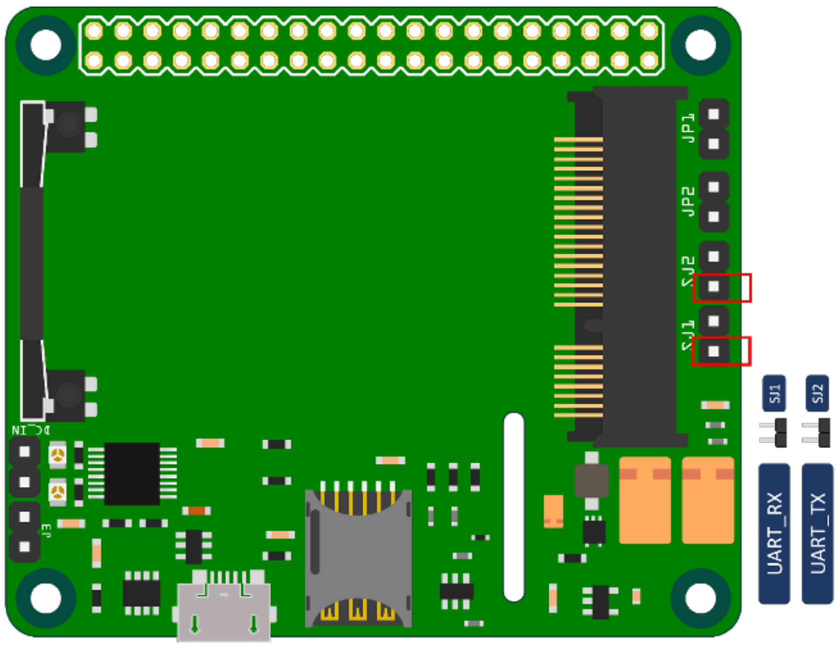
White: Tx

Green: Rx

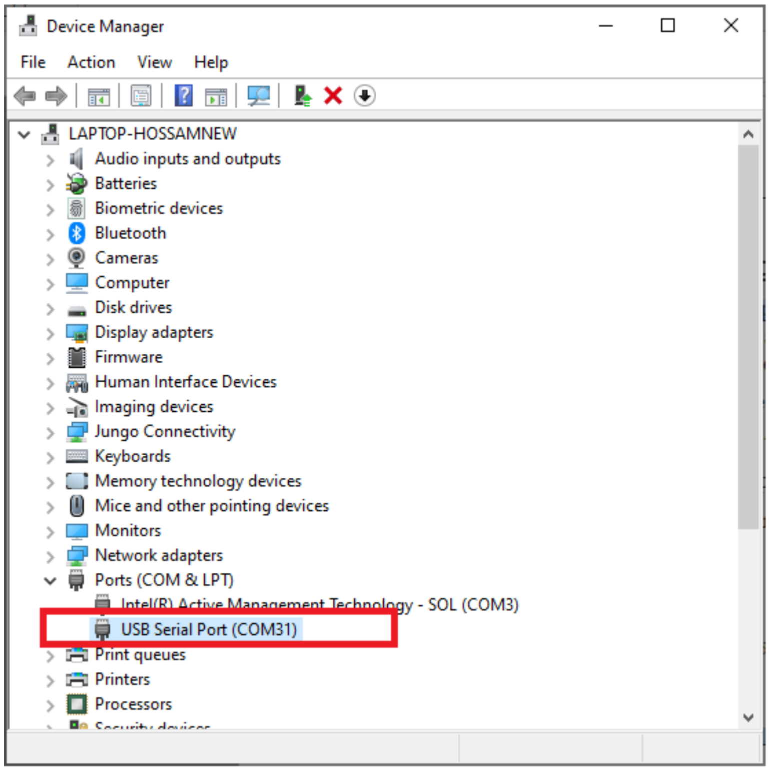
Black: GND

Connect the **White** cable to **UART_TX** and **Green** cable to **UART_RX**. Also connect the **Black** cable to the **GND**).

As in the picture below, connect **UART_TX to SJ2 (PIN2) AND UART_RX to SJ1 (PIN2)**.



Make sure when you connect the USB-to-UART cable, it shows correctly in Windows device manager and all its driver is installed as in this screenshot.

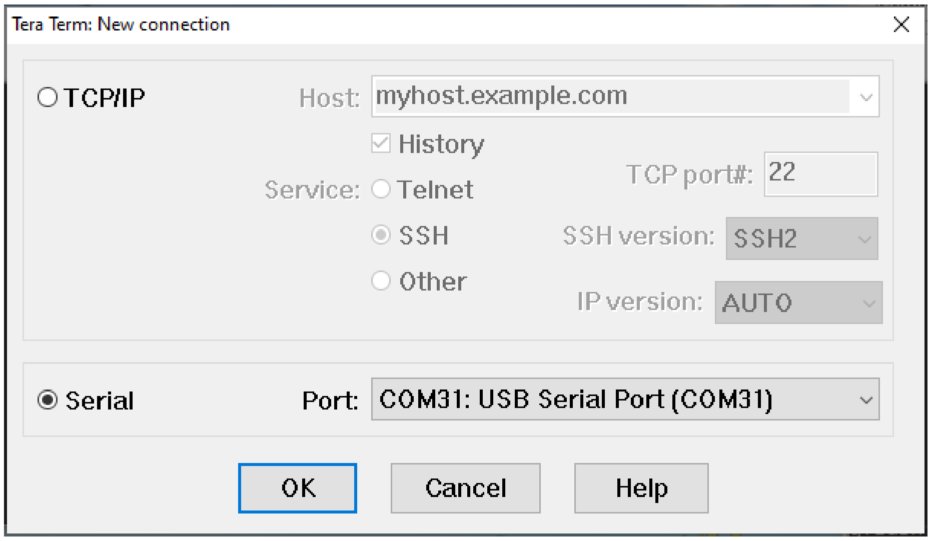


13 Using Serial Terminal

It is recommended to use Tera Term tool as the serial terminal. You can download it from here:

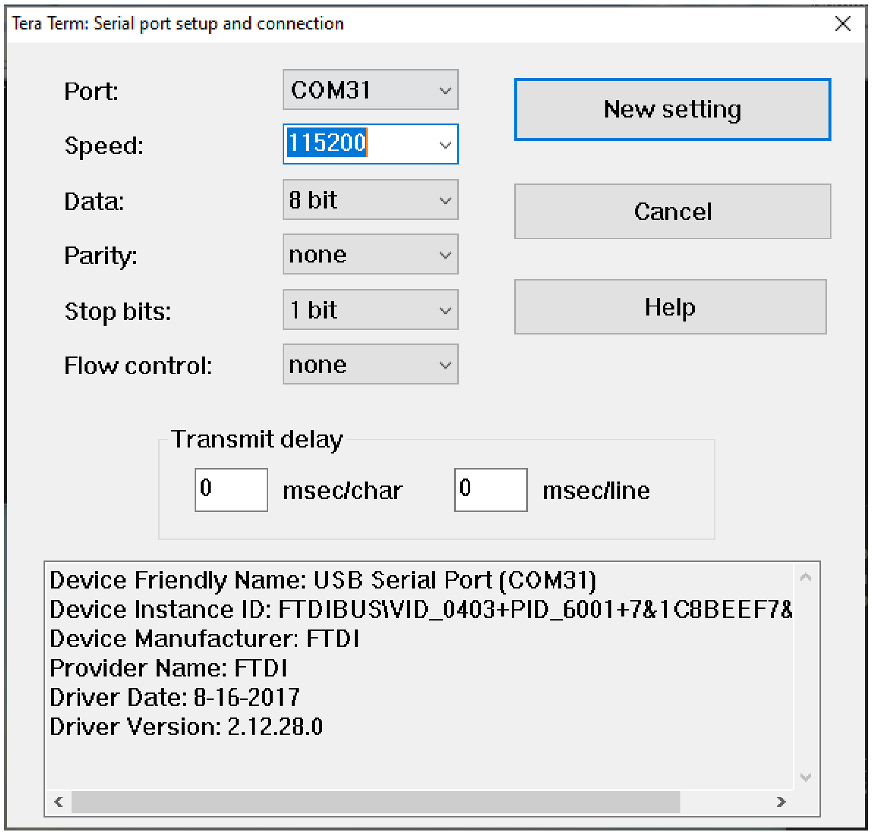
https://osdn.net/projects/ttssh2/downloads/54081/teraterm-4.72.exe/

Launch Tera Terminal and select the **Serial** option and select USB-to-Serial port.



TCP/IP Host: myhost.example.com
[✓] History
Service: Telnet [✓ SSH] Other
TCP port#: 22
SSH version: SSH2
IP version: AUTO
[✓ Serial] Port: COM31: USB Serial Port (COM31)

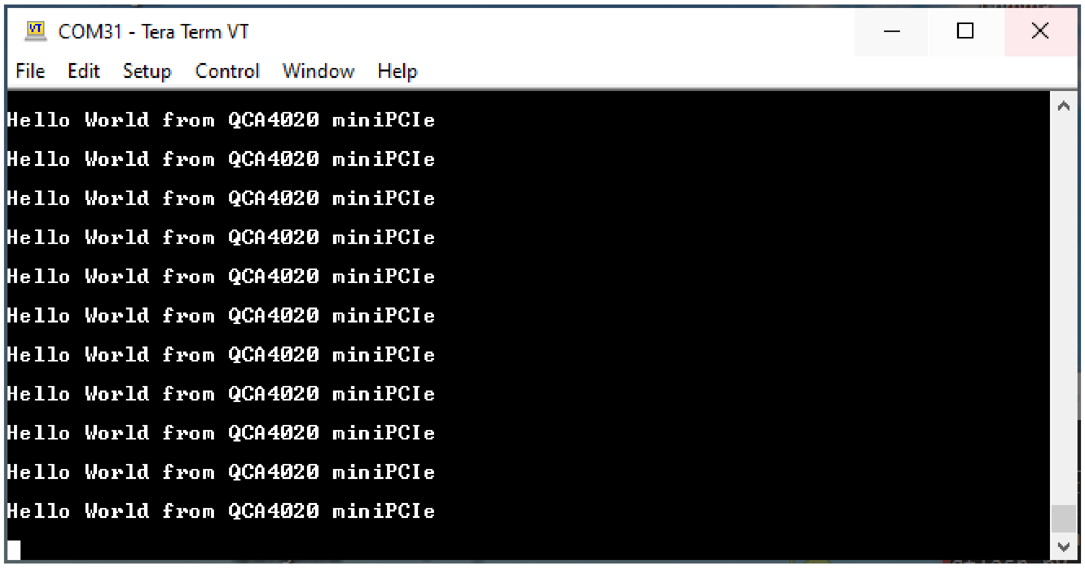
In Tera Term, choose **Setup->Serial port** and configure serial ports according to the following:



Port: COM31
Speed: 115200
Data: 8 bit
Parity: none
Stop bits: 1 bit
Flow control: none
0 msec/char
0 msec/line

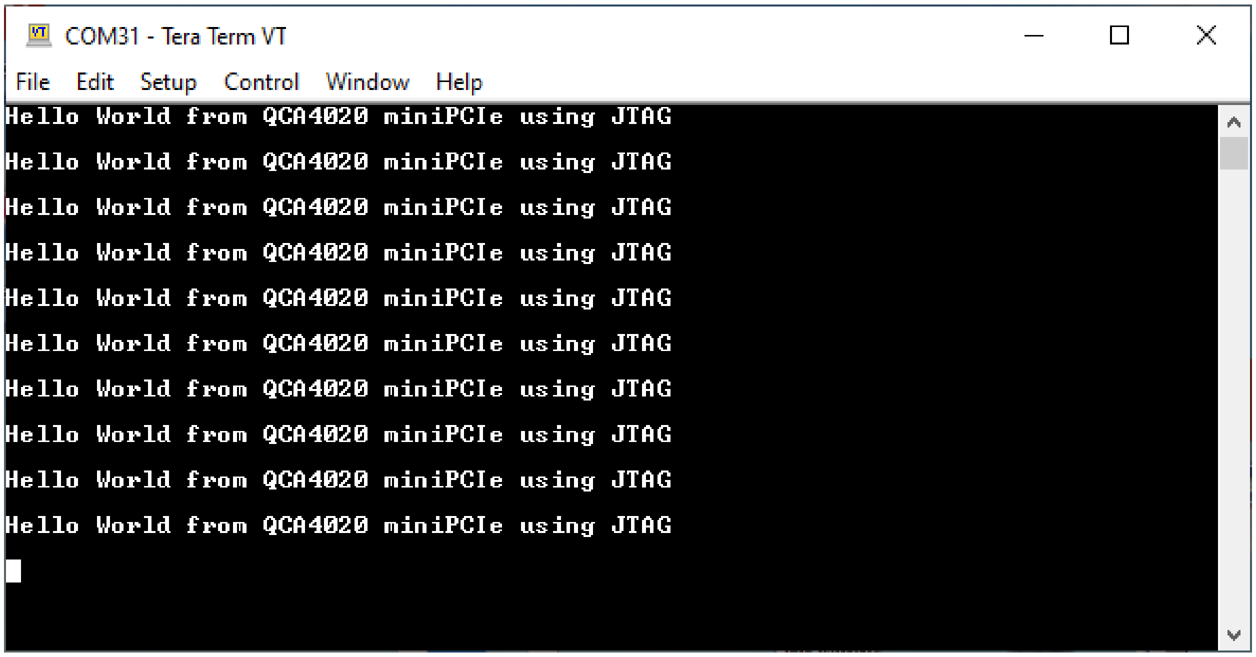
Device Friendly Name: USB Serial Port (COM31)
Device Instance ID: FTDIBUS\VID_0403+PID_6001+7&1C8BEEF7&
Device Manufacturer: FTDI
Provider Name: FTDI
Driver Date: 8-16-2017
Driver Version: 2.12.28.0

And now you are ready to see the output from "hello World" application on the serial monitor.



```
Hello World from QCA4020 miniPCIe
Hello World from QCA4020 miniPCIe
Hello World from QCA4020 miniPCIe
Hello World from QCA4020 miniPCIe
Hello World from QCA4020 miniPCIe
Hello World from QCA4020 miniPCIe
Hello World from QCA4020 miniPCIe
Hello World from QCA4020 miniPCIe
Hello World from QCA4020 miniPCIe
Hello World from QCA4020 miniPCIe
Hello World from QCA4020 miniPCIe
```

If you are using JTAG mode to flash the image to the QCA4020 miniPCIe, you shall see the following output on Tera terminal

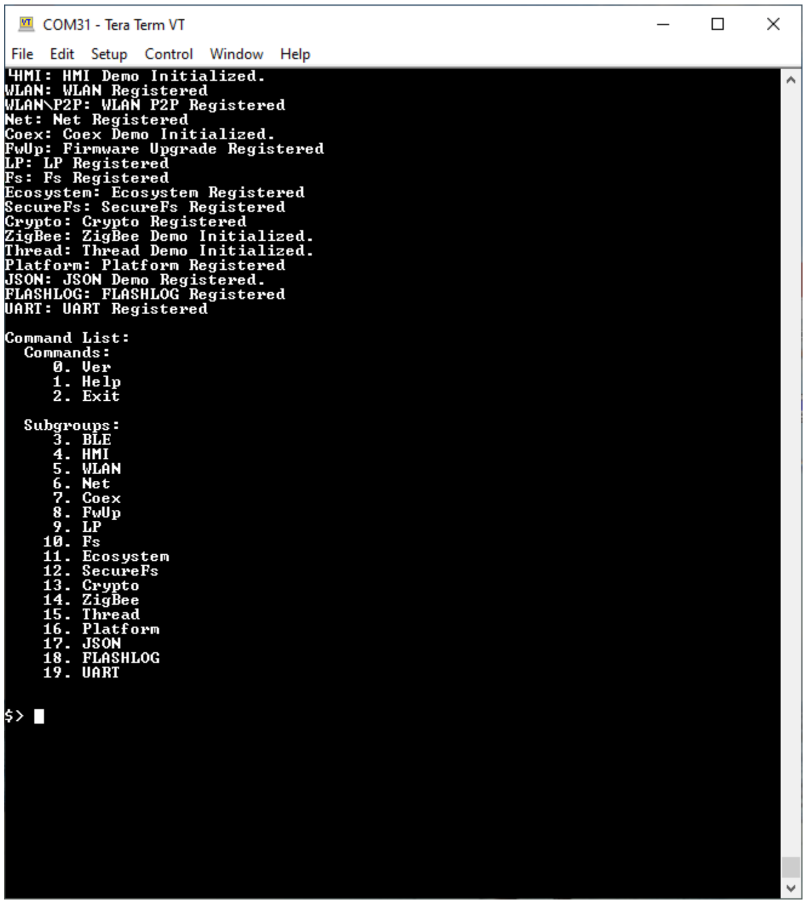


```
Hello World from QCA4020 miniPCIe using JTAG
Hello World from QCA4020 miniPCIe using JTAG
Hello World from QCA4020 miniPCIe using JTAG
Hello World from QCA4020 miniPCIe using JTAG
Hello World from QCA4020 miniPCIe using JTAG
Hello World from QCA4020 miniPCIe using JTAG
Hello World from QCA4020 miniPCIe using JTAG
Hello World from QCA4020 miniPCIe using JTAG
Hello World from QCA4020 miniPCIe using JTAG
Hello World from QCA4020 miniPCIe using JTAG
```

14 QCLI Demo Application

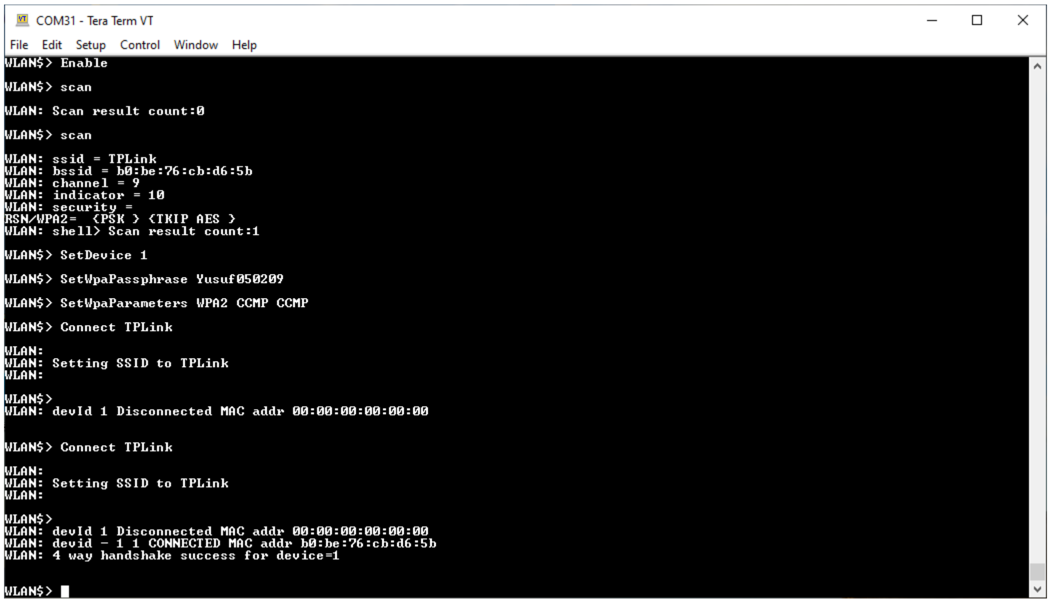
CLI demo application is a comprehensive demo that provides a mechanism to demonstrate different and all features and technologies that QCA4020 miniPCIe supports. It also provides reference implementation and usage of customer facing QAPIs.

The QCLI-demo application is available with the QCA4020 SDK and you can follow the same steps described before in Eclipse to build and flash it to the QCA4020 miniPCIe. Once flashed, you can see the output on the serial terminal as in the screenshot below. Also, you can exercise connecting to your home/office WLAN



```
HMI: HMI Demo Initialized.
WLAN: WLAN Registered
WLAN\P2P: WLAN P2P Registered
Net: Net Registered
Coex: Coex Demo Initialized.
FwUp: Firmware Upgrade Registered
LP: LP Registered
Fs: Fs Registered
Ecosystem: Ecosystem Registered
SecureFs: SecureFs Registered
Crypto: Crypto Registered
ZigBee: ZigBee Demo Initialized.
Thread: Thread Demo Initialized.
Platform: Platform Registered
JSON: JSON Demo Registered.
FLASHLOG: FLASHLOG Registered
UART: UART Registered

Command List:
  Commands:
     0. Ver
     1. Help
     2. Exit

  Subgroups:
     3. BLE
     4. HMI
     5. WLAN
     6. Net
     7. Coex
     8. FwUp
     9. LP
    10. Fs
    11. Ecosystem
    12. SecureFs
    13. Crypto
    14. ZigBee
    15. Thread
    16. Platform
    17. JSON
    18. FLASHLOG
    19. UART

$>
```

The following screenshot shows how to connect to an WLAN access point.



```
WLAN$> Enable

WLAN$> scan

WLAN: Scan result count:0

WLAN$> scan

WLAN: ssid = TPLink
WLAN: bssid = b0:be:76:cb:d6:5b
WLAN: channel = 9
WLAN: indicator = 10
WLAN: security =
RSN/WPA2=  <PSK > <TKIP AES >
WLAN: shell> Scan result count:1

WLAN$> SetDevice 1

WLAN$> SetWpaPassphrase Yusuf050209

WLAN$> SetWpaParameters WPA2 CCMP CCMP

WLAN$> Connect TPLink

WLAN:
WLAN: Setting SSID to TPLink
WLAN:

WLAN$>
WLAN: devId 1 Disconnected MAC addr 00:00:00:00:00:00


WLAN$> Connect TPLink

WLAN:
WLAN: Setting SSID to TPLink
WLAN:

WLAN$>
WLAN: devId 1 Disconnected MAC addr 00:00:00:00:00:00
WLAN: devid - 1 1 CONNECTED MAC addr b0:be:76:cb:d6:5b
WLAN: 4 way handshake success for device=1


WLAN$> 
```