Qualcomm Technologies, Inc.

Qualcomm Linux Build Guide

80-70018-254 AC

April 10, 2025

Contents

1 Introduction

Qualcomm recommends that you read the Qualcomm Linux Yocto Guide before starting your build. You can do any of the following:

- Download prebuilt images and flash the software
- Sync, build, and flash the software

1.1 Download prebuilt images and flash the software

- Download prebuilt flashable images (includes Platform eSDK) using the links in the table *Artifactory links to prebuilt flashable images and eSDK* of the Release Notes.
- The Platform eSDK is an installer generated from the Qualcomm Linux software. It provides a complete Yocto environment that allows you to sync, change, compile, and install applications and open-source plug-ins. For more information, see Download the Platform eSDK.
- To flash the prebuilt flashable images, see Flash software images.

1.2 Sync, build, and flash the software

Unregistered users

Unregistered users can sync and build Qualcomm Linux using the Build with GitHub for unregistered users. Qualcomm exclusively offers firmware components in the form of binary files to users who aren't registered.

Registered users

Note: To register, go to https://www.qualcomm.com/support/working-with-qualcomm.

Registered users can use any one of the following three methods to sync and build Qualcomm Linux. These methods use the Qualcomm Yocto layers and the supporting base Yocto layers. Registered users have the ability to access the source of certain firmware components and Qualcomm tools, which allows them to build and download the software.

Launcher	CLI	GitHub
Easy-to-use, GUI-based, Qualcomm® Software Center (QSC) Launcher.	Simple QSC command-line interface (CLI).	Instructions to use GitHub based workflow.
Build with QSC Launcher	Build with QSC CLI	Build with GitHub for registered users

Note:

- See hardware SoCs that are supported on Qualcomm Linux.
- For QCS615 build instructions, see QCS615.LE.1.0 Software User Guide.

2 Build with QSC Launcher

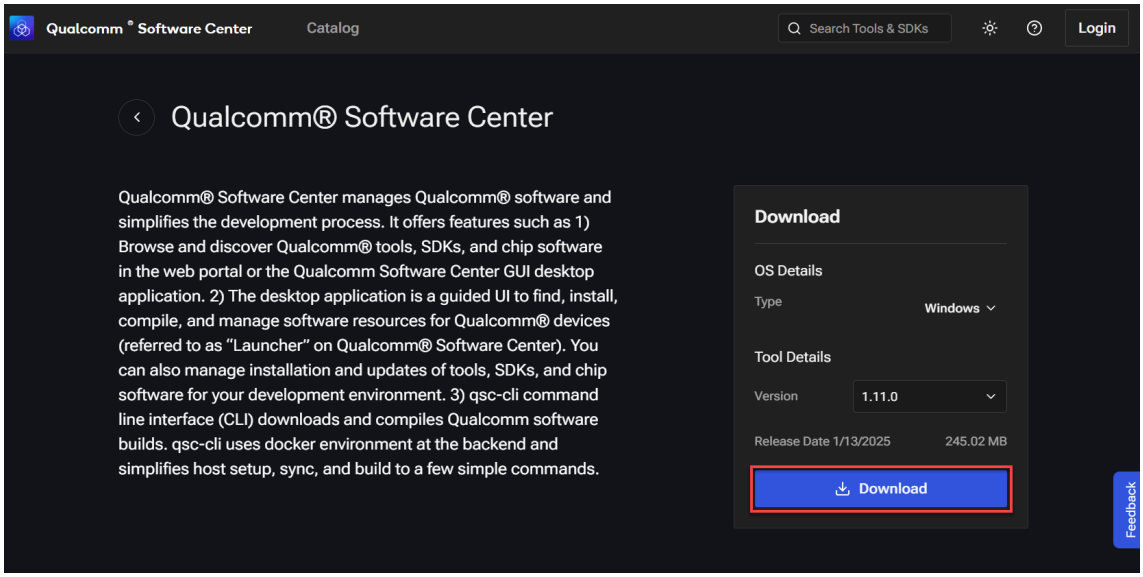
The following sections explain how to configure, download, compile, and flash Qualcomm Linux using the QSC Launcher.

2.1 Host computer requirements

Configuration	Permissions
x86 machine	A `sudo` permission is required to run a few commands
Quad-core CPU, for example, Intel i7-2600 at 3.4 GHz (equivalent or better)	
300 GB free disk space (swap partition > 32 GB)	
16 GB RAM	
Ubuntu 22.04	

2.2 Install QSC using a GUI

1. Download the latest Qualcomm Software Center, and then select *Download*.

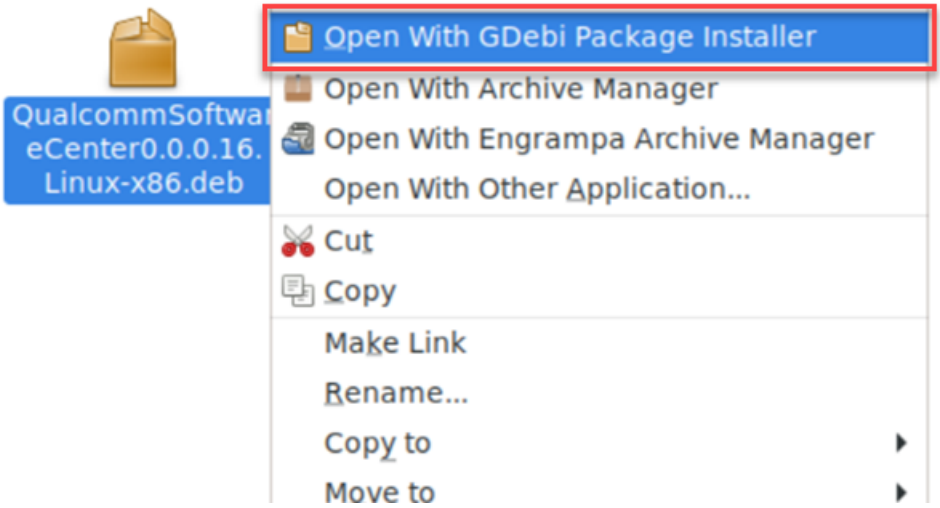


The QSC Debian package (`.deb`) is downloaded to your machine.

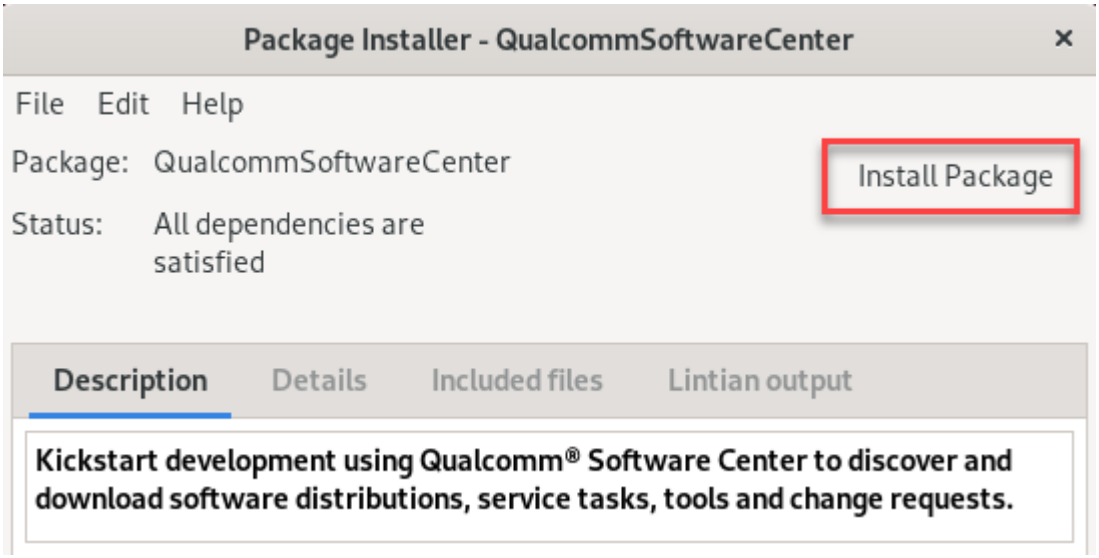
2. To install the QSC Debian package using the Ubuntu GNOME GUI, GDebi must be installed first. Run the following command on your Linux host computer to install GDebi:

```
sudo apt update
sudo apt install gdebi
```

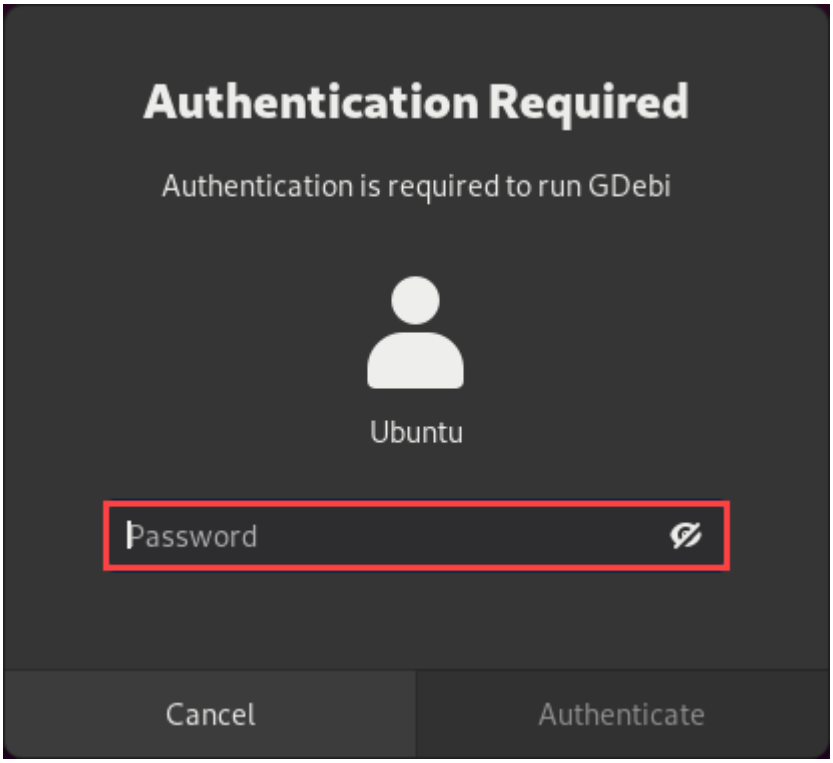
3. Right-click on the downloaded QSC Debian file and select *Open With GDebi Package Installer*.



4. Select *Package*.

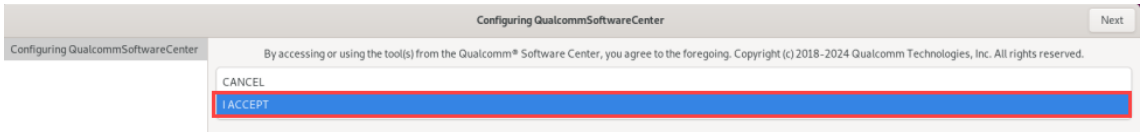


5. Enter the password for the current user. In the following image, the current user is Ubuntu.

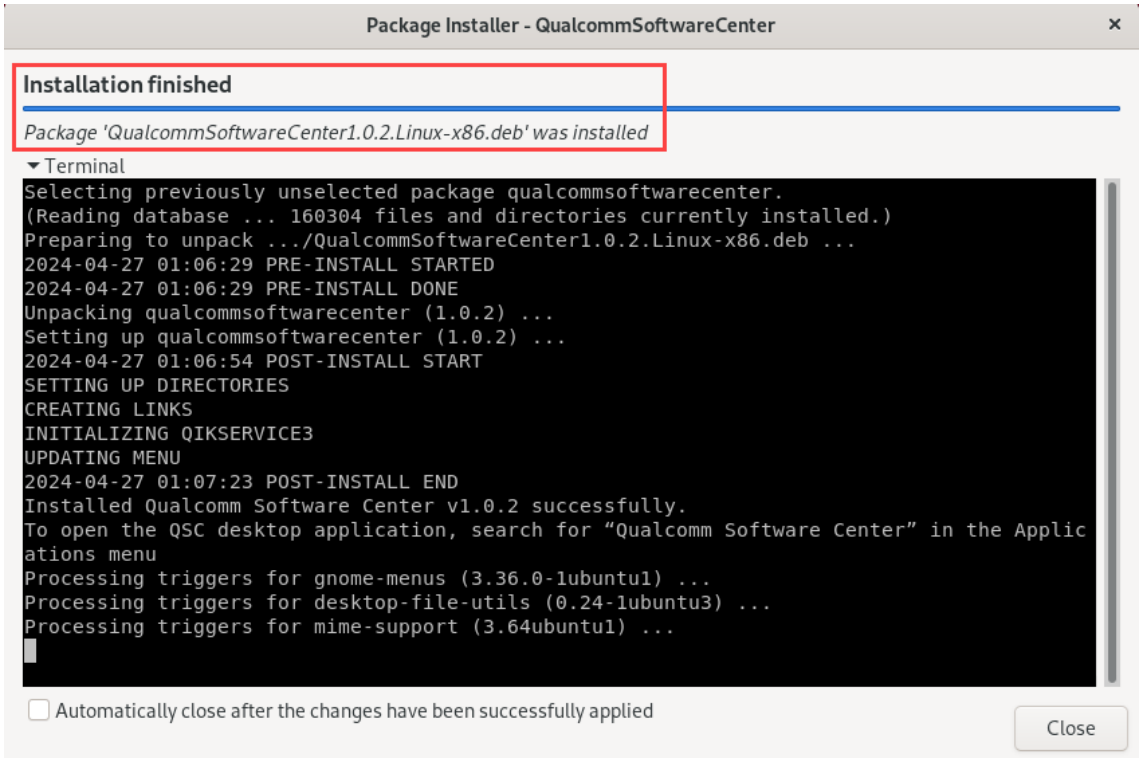


6. Accept the license agreement. Select *Next*.

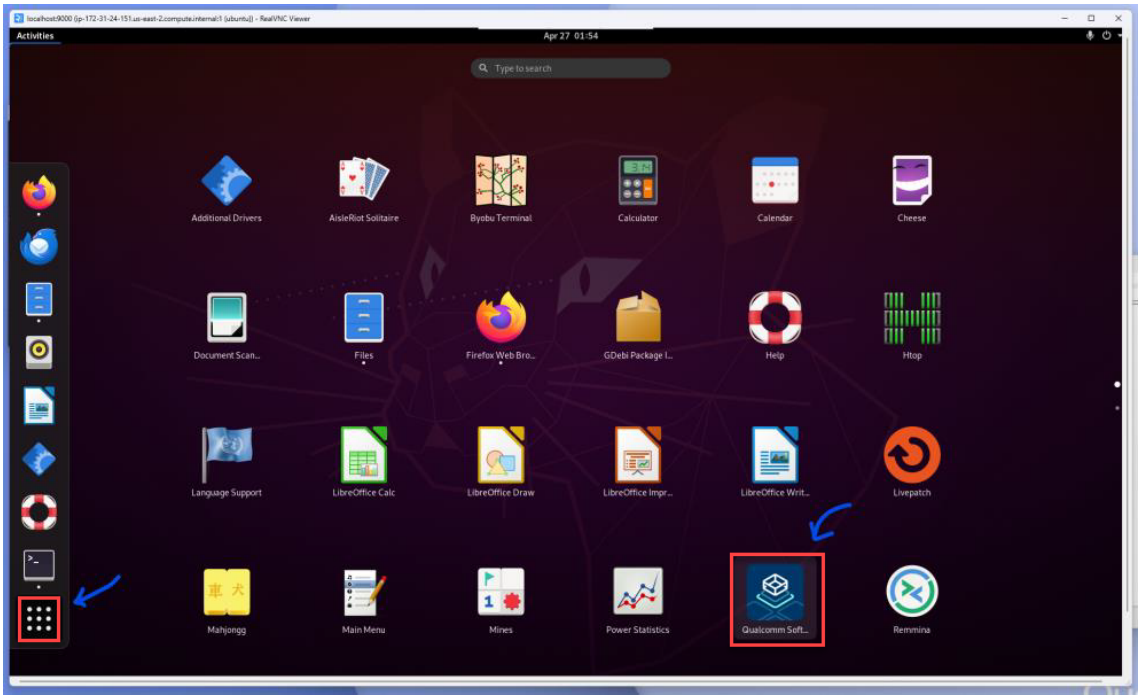
7. Select *I ACCEPT*.



Wait for the installation to complete.



8. To find the Qualcomm Software Center, select *Show Applications*.

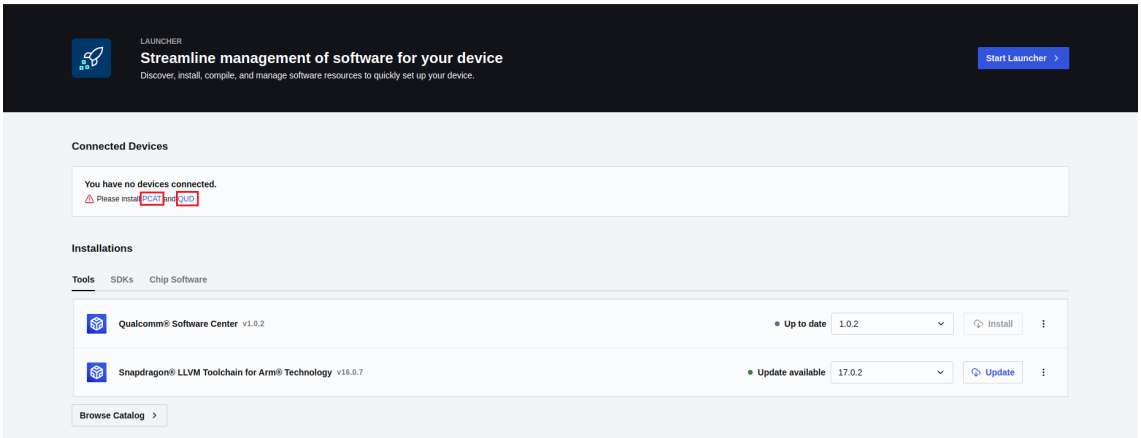


2.3 Use QSC Launcher

Software download

1. To open the QSC Launcher desktop application, launch *Qualcomm Software Center* from the *Applications* menu.

Note: For the Launcher workflow to detect connected devices and flash software builds, install the Qualcomm Product Configuration Assistant Tool (PCAT) and Qualcomm USB Driver (QUD) tools on the host computer. Select *PCAT* to install PCAT and *QUD* to install QUD as shown in the following image:



Or

Install PCAT and QUD using `qpm-cli`:

```
qpm-cli --login
qpm-cli --install quts --activate-default-license
qpm-cli --install qud --activate-default-license
qpm-cli --install pcat --activate-default-license
```

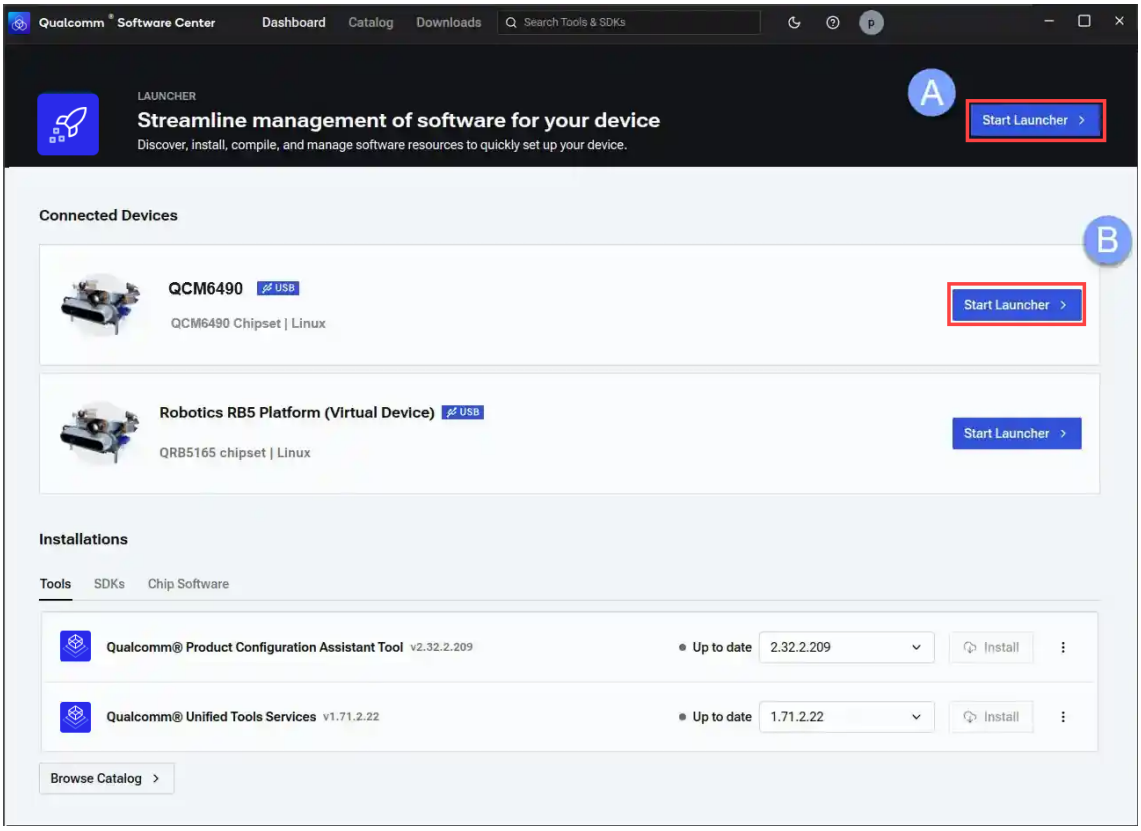
The `qpm-cli --help` command lists the help options.

For Ubuntu 22.04, you may see an issue while installing QUD where you must enroll the public key on your Linux host for a successful QUD installation. For more details, follow the steps provided in the `signReadme.txt` file available at the `/opt/QTI/sign/` directory.

---

2. Use your registered email ID to sign in to the QSC desktop application. The QSC Launcher dashboard page (for example, when you connect the QCS6490 development kit) appears as shown in the following image:

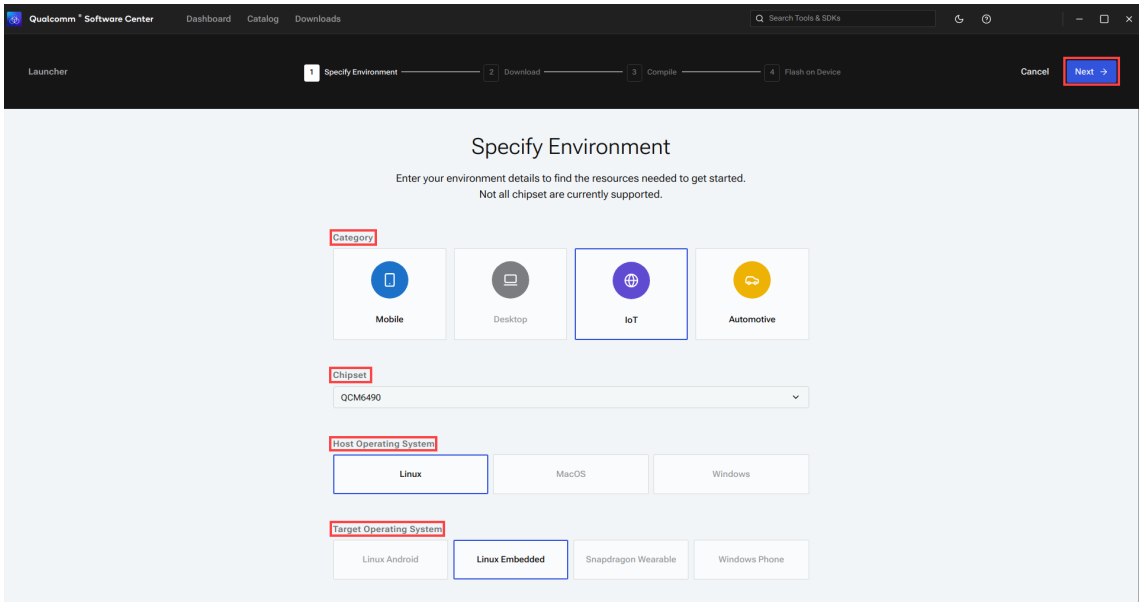
Note: To register, go to https://www.qualcomm.com/support/working-with-qualcomm.



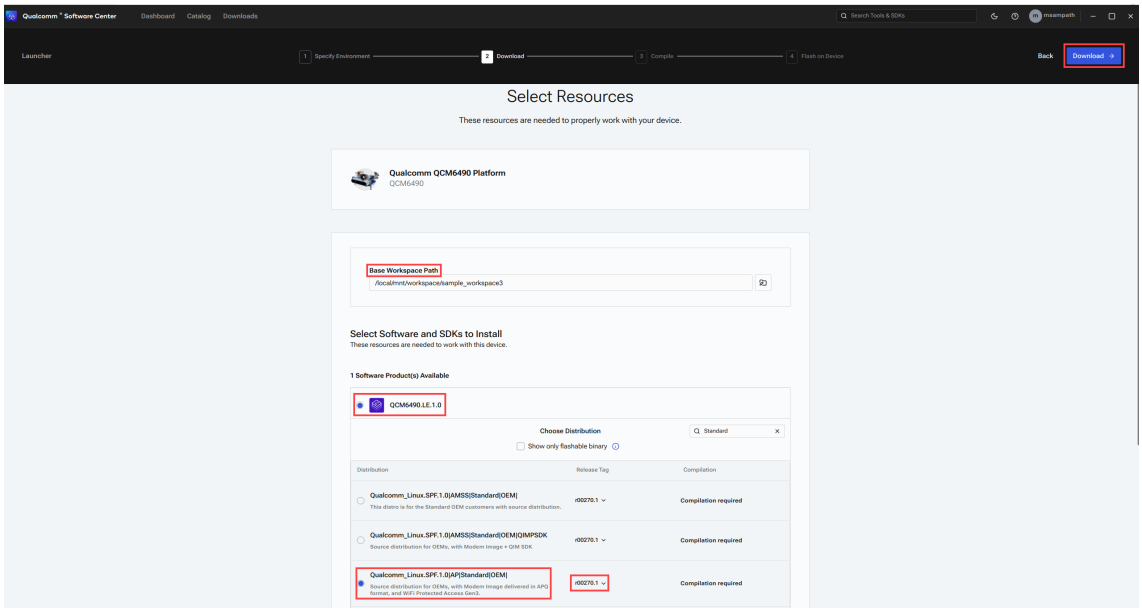
- If you don't have a connected device, click *Start Launcher* (A) on the top panel.
- If you have a connected device, select *Start Launcher* (B) for the appropriate device in the *Connected devices* panel.

3. On the *Specify Environment* page, select the appropriate values for *Category*, *Chipset*, *Host Operating System*, and *Target Operating System*.

Note: See chipsets (hardware SoCs) that are supported on Qualcomm Linux.



4. Select *Next*. The *Select Resources* page appears.



a. In the *Base Workspace Path* text box, specify a directory where you want to download the software.

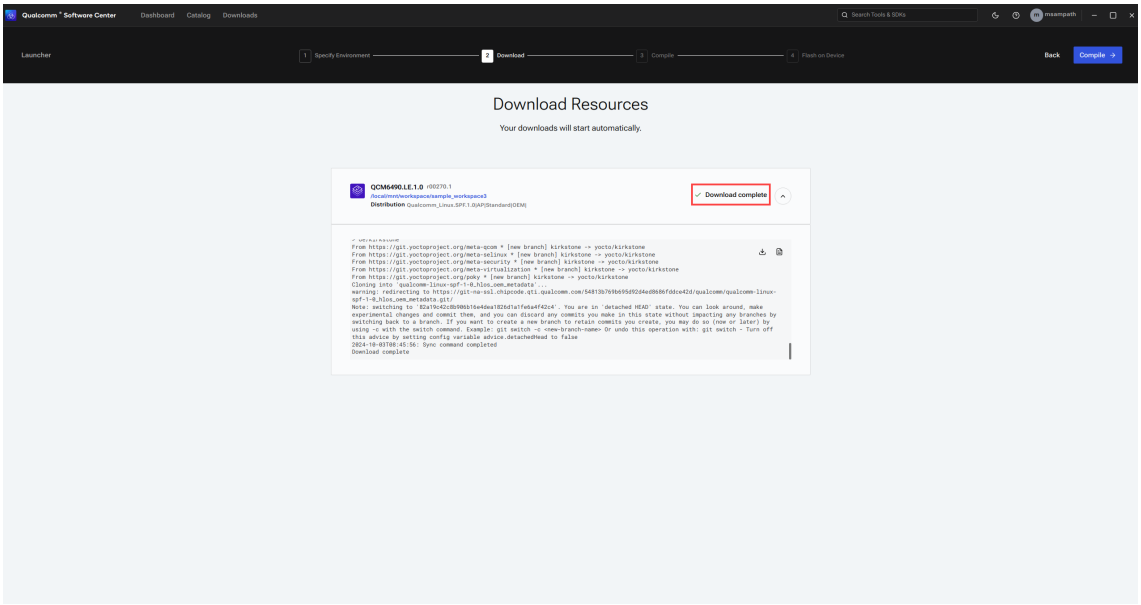
b. Select the *Software Product*.

c. Select the *Distribution* and the *Release Tag*.

Note:

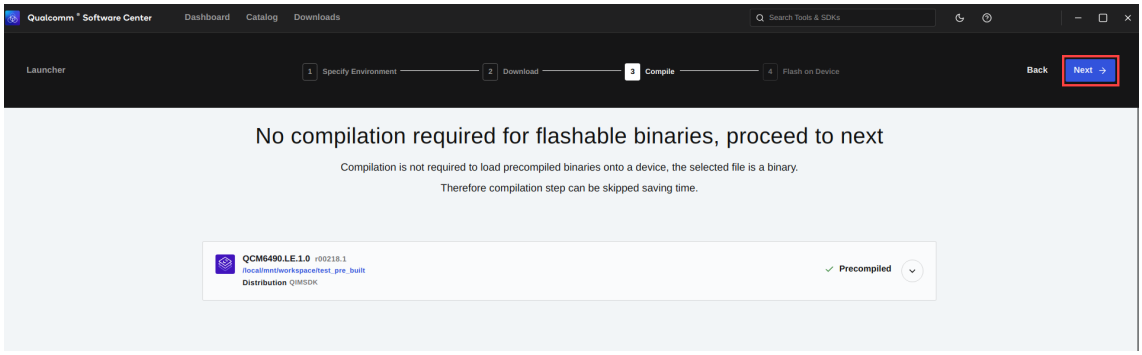
- For information on the supported distributions for your hardware SoCs, see the table *Access Controlled Distribution* in the Release Notes.
- For information on the Yocto layers, see Qualcomm Linux metadata layers.
- For information on the QIMP and QIRP SDKs, see the following guides:
  - QIMP SDK Quick Start Guide
  - QIRP SDK 2.0 User Guide

5. Select *Download* to download the selected compilable distribution or flashable binary. After the software download is successful, a *Download complete* status appears.



Note: You can also track the download progress through the *Downloads* option in the top menu bar.

You don't have to compile flashable binaries. If you have selected a flashable binary, follow the on-screen instructions to flash to a connected device.


Qualcomm Software Center
Dashboard
Catalog
Downloads
Search Tools & SDKs
Launcher
1 Specify Environment
2 Download
3 Compile
4 Flash on Device
Back
Next
No compilation required for flashable binaries, proceed to next
Compilation is not required to load precompiled binaries onto a device, the selected file is a binary.
Therefore compilation step can be skipped saving time.
QCM6490.LE.1.0 r00218.1
/local/mnt/workspace/test_pre_built
Distribution QIMSDK
Precompiled

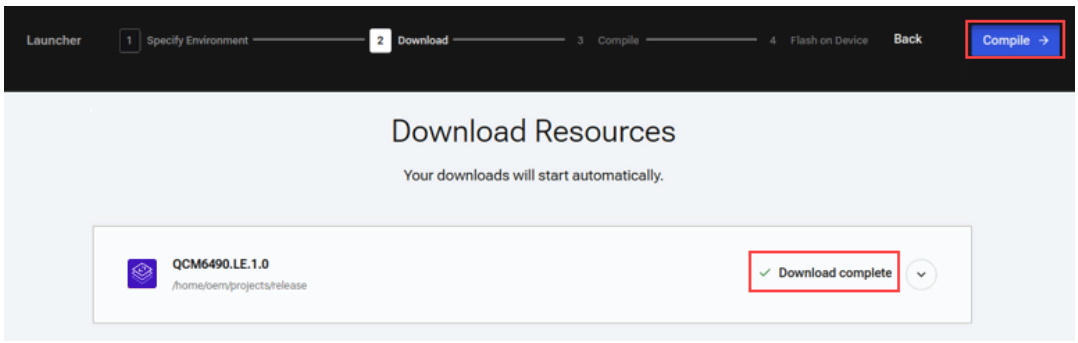
Build and flash default configuration

1. Compile the default build.

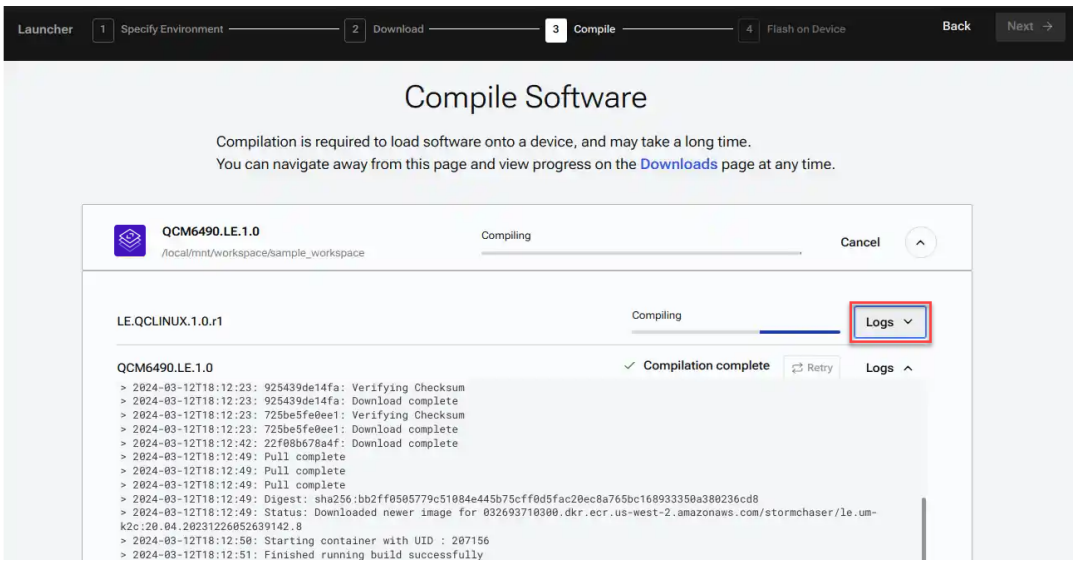
Note: For information on the default configurations, see the table *Default values of "MACHINE" and "QCOM_SELECTED_BSP" parameters for QSC* in the Release Notes.

---

   a. After the download is complete, select *Compile* to start compiling. Depending on the size of the downloaded software and the host computer configuration, the compilation process may take a few hours.



   b. To view the compilation progress, expand the logs panel.



Note: BitBake fetch errors are typically intermittent fetch failures. To resolve these errors, retry step 1a. If the issue persists, see BitBake Fetcher Error for a solution.

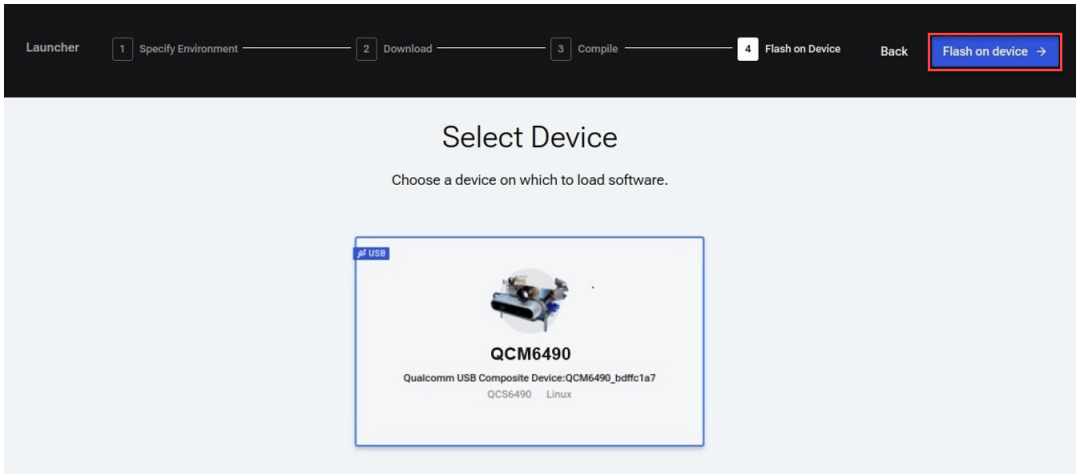
---

2. Flash the software with the default configuration.

Note: Before you flash the software, ensure the following:

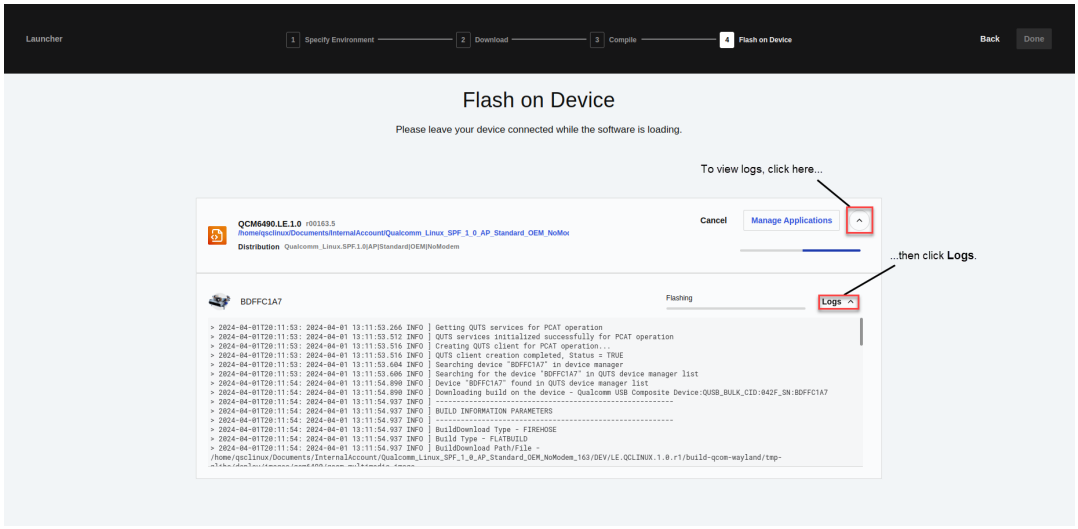
1. Device is in Emergency Download (EDL) mode.
2. Provision UFS.
3. Flash CDT.
4. Flash SAIL.

a. Flash the software by selecting the device on which you want to flash the compiled software. If you connected many devices, then select the correct target device.

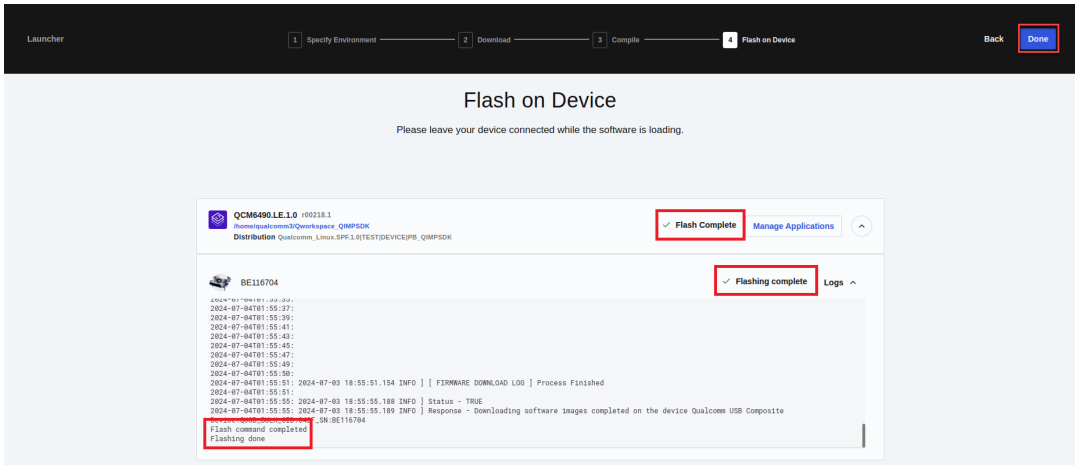


b. Select *Flash on device*. The page is updated and displays a progress bar as QSC Launcher begins flashing the software. Leave the device connected while you flash the software.

c. To view the compilation progress of individual software images, expand the logs panel.



d. When the process is complete, the page displays a *Flash Complete* message.



e. Select *Done*.

Note: The device reboots after the flashing procedure completes successfully. To verify the updated software version, see Verify the Qualcomm Linux version.

---

f. To establish UART and network connections, see Connect to UART shell and network.

Build your own configuration

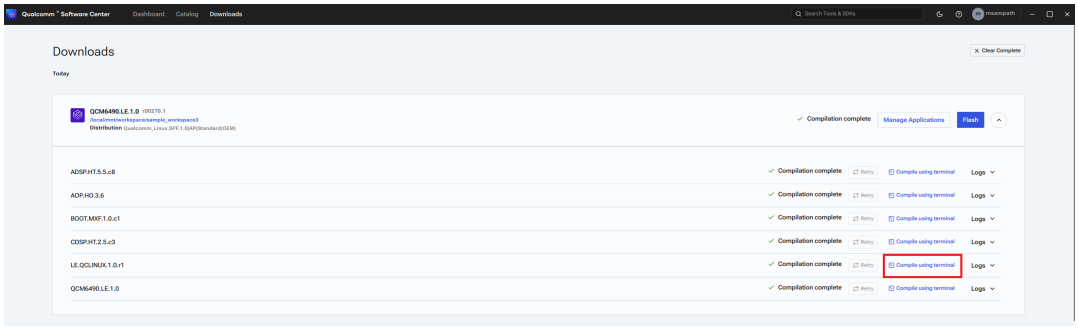
To build your own configuration, compile the build for default machine configuration and then compile the LE.QCLINUX.1.0.r1 image with your own MACHINE and QCOM_SELECTED_BSP parameter values.

1. Compile the build for the default machine configuration.
2. Compile the *LE.QCLINUX.1.0.r1* image with your own MACHINE and QCOM_SELECTED_BSP parameter values.

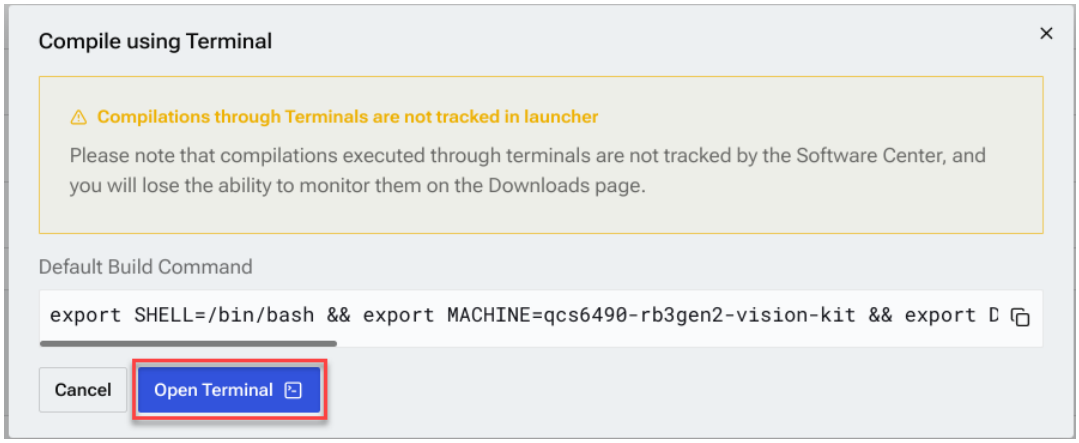
Note: For information on the supported machine configurations of the development kit, see the table *Default values of "MACHINE" and "QCOM_SELECTED_BSP" parameters for QSC* in the Release Notes.

---

a. To run the build commands for a specific configuration, select *Compile using terminal*.



Note: Compilations initiated through the terminal aren't tracked by the Qualcomm Software Center, making it impossible to track their progress on the Download page.

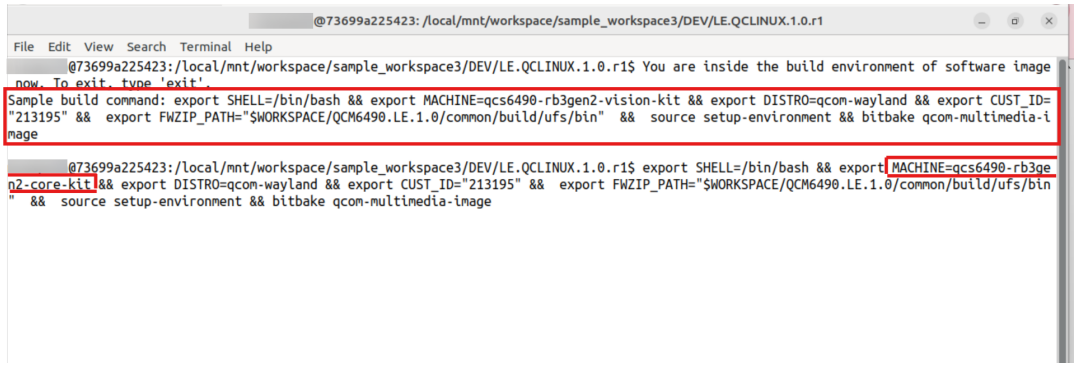


---

b. Select *Open Terminal*. A terminal window with the expanded default build command

appears.

c. Update the highlighted command with your own configuration and run it on the terminal:



For example, to build for Qualcomm® RB3 Gen 2 Core Development Kit, change the value of `MACHINE` in the preceding build command to `qcs6490-rb3gen2-core-kit`.

d. After a successful build, check that the `system.img` file is in the `<Base Workspace Path>/DEV/LE.QCLINUX.1.0.r1/build-<DISTRO>/tmp-glibc/deploy/images/<MACHINE>/qcom-multimedia-image` directory with an updated timestamp. For example:

```
cd <Base Workspace Path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-core-kit/qcom-multimedia-image
ls -al system.img
```

Note: When compiling a software image other than `LE.QCLINUX.1.0.r1`, ensure that you also compile both the software product and `LE.QCLINUX.1.0.r1` in the same order.

For example, if you compile `BOOT.MXF.1.0.c1`, ensure that you also compile the software product (such as `QCM6490.LE.1.0`) and then `LE.QCLINUX.1.0.r1`.

---

3. To flash your build, see Flash software images.

Note:

- Before flashing, update the build images path to the compiled build images workspace at `<Base_Workspace_Path>/DEV/LE.QCLINUX.1.0.r1/build-<DISTRO>/tmp-glibc/deploy/images/<MACHINE>/qcom-multimedia-image`.

  For example, `<Base Workspace Path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-core-kit/qcom-multimedia-image`.

---

4. To establish UART and network connections, see Connect to UART shell and network.

3 Build with QSC CLI

The following sections explain how to configure, download, compile, and flash Qualcomm Linux using the QSC CLI.

3.1 Host computer requirements

Configuration	Permissions
x86 machine	A `sudo` permission is required to run a few commands
Quad-core CPU, for example, Intel i7-2600 at 3.4 GHz (equivalent or better)	
300 GB free disk space (swap partition > 32 GB)	
16 GB RAM	
Ubuntu 22.04	

3.2 Install QSC CLI

1. Install curl (if not installed already):

```
sudo apt install curl
```

2. Download the Debian package for `qsc-cli`:

```
cd <workspace_path>
curl -L https://softwarecenter.qualcomm.com/api/download/
software/qsc/linux/latest.deb -o qsc_installer.deb
```

3. Install the `qsc-cli` Debian package:

```
sudo apt update
sudo apt install ./qsc_installer.deb
```

4. Sign in to `qsc-cli` using your registered email ID:

   Note: To register, go to https://www.qualcomm.com/support/working-with-qualcomm.

   ```
   qsc-cli login -u <username>
   ```

Note: For more information, see `qsc-cli` related topics in How to Sync.

3.3 Use QSC CLI

This section explains how to configure, download, compile, and flash Qualcomm Linux using the QSC CLI.

Software download

- Download a software release by specifying the absolute workspace path, product ID, distribution, and release ID as shown in the following example:

  ```
  qsc-cli download --workspace-path '<Base_Workspace_Path>' --
  product '<Product_ID>' --distribution '<Distribution>' --release
  '<Release_ID>'
  # Example, qsc-cli download --workspace-path '/local/mnt/
  workspace/sample_workspace' --product 'QCM6490.LE.1.0' --
  distribution 'Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem' --
  release 'r00270.1'
  ```

  Note:

  – If you are downloading more than one distribution, create a new workspace for each distribution that you download.

  – For the Product_ID, Distribution, and Release_ID values, see the table *QSC-CLI Input Parameters* in the Release Notes.

  – For more information on the Yocto layers, see Qualcomm Linux metadata layers.

Build default configuration

Compile

Note: For information on the default configurations, see the table *Default values of MACHINE and QCOM_SELECTED_BSP parameters for QSC* in the Release Notes.

Start the compilation process after the download is complete:

Note: Depending on the size of the software and the host computer configuration, the compilation process may take a few hours.

```
qsc-cli compile --workspace-path '<Base_Workspace_Path>'

# Example, qsc-cli compile --workspace-path '/local/mnt/workspace/
sample_workspace'
```

This process builds the necessary Qualcomm firmware and completes the Qualcomm Linux build.

Note: If you get a BitBake fetcher error, try recompiling to resolve the issue. If the issue persists, see BitBake Fetcher Error for a solution.

Recompile

To recompile after any modifications to the software release, use your existing workspace built using QSC CLI:

```
qsc-cli compile --image '<Software_Image_Name>' --workspace-path '
<Base_Workspace_Path>'

# Example, qsc-cli compile --image LE.QCLINUX.1.0.r1 --workspace-path
'/local/mnt/workspace/sample_workspace'
```

Note: For information on software image names (`--image`), see the table *QSC-CLI Input Parameters* in the Release Notes.

Flash

Note: For the QSC CLI to detect the connected devices and flash the software builds, install the Qualcomm Product Configuration Assistant Tool (PCAT) and Qualcomm USB Driver (QUD) on the host computer. Use the `qpm-cli` command to install PCAT and QUD:

```
qpm-cli --login
qpm-cli --install quts --activate-default-license
qpm-cli --install qud --activate-default-license
qpm-cli --install pcat --activate-default-license
```

The `qpm-cli --help` command lists the help options.

For Ubuntu 22.04, you may see an issue while installing QUD, where you must enroll the public key on your Linux host for a successful QUD installation. For more information, see the `signReadme.txt` file in the `/opt/QTI/sign/` directory.

Note: Before you flash the software, ensure the following:

1. Device is in Emergency Download (EDL) mode.
2. Provision UFS.
3. Flash CDT.
4. Flash SAIL.

1. Flash a device.

   ```
   qsc-cli flash --workspace-path <Base_Workspace_Path> --
   buildflavor "sa2150p_emmc" --serialnumber <serial number>

   # Example, qsc-cli flash --workspace-path '/local/mnt/workspace/
   sample_workspace' --serialnumber 'be116704'
   ```

   The `--buildflavor` argument is optional and only required for devices that have multiple flavors. To list the build flavors, run the following command on the host computer:

   ```
   qsc-cli flash --workspace-path <workspace path> --list-
   buildflavor
   ```

   Note:

   - To find the *<serial number>*, run the following command on the host computer:

```
pcat -devices
```

Sample output

```
Searching devices in Device Manager, please wait for a
moment...
ID | DEVICE TYPE | DEVICE STATE | SERIAL NUMBER | ADB SERIAL
NUMBER | DESCRIPTION
NA | NA          | EDL          | BE116704      | be116704
      | Qualcomm USB Composite Device:QUSB_BULK_CID:042F_SN:
BE116704
```

- The device reboots after the flashing procedure completes successfully. To verify the updated software version, see Verify the Qualcomm Linux version.

2. To establish UART and network connections, see Connect to UART shell and network.

Build your own configuration

To build your own configuration, you must compile the build for default machine configuration and compile the LE.QCLINUX.1.0.r1 image with your own MACHINE and QCOM_SELECTED_BSP parameter values.

1. Compile the build for the default machine configuration:
   a. Download the software.
   b. Compile the default build.
2. Compile the `LE.QCLINUX.1.0.r1` image with your own MACHINE and QCOM_SELECTED_BSP parameter values.

Note: For information on the supported machine configurations of the development kit, see the table *Default values of MACHINE and QCOM_SELECTED_BSP parameters for QSC* in the Release Notes.

   a. Run the build commands for a specific configuration:

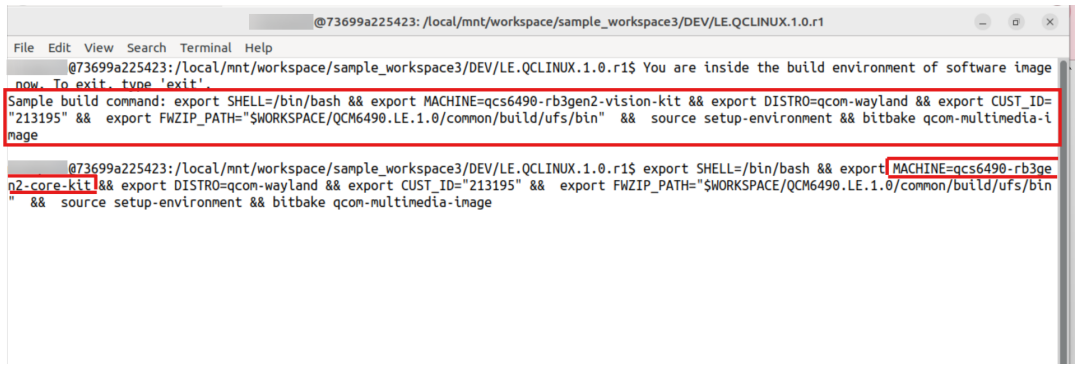
```
qsc-cli open-build-env --workspace-path <Base_Workspace_Path>
--image <Software_Image_Name>
# Example, qsc-cli open-build-env --workspace-path '/local/
mnt/workspace/sample_workspace' --image 'LE.QCLINUX.1.0.r1'
```

This command opens the terminal.

Note: An environment is set up to run your own build commands for a specific software image. QSC won't track the status of input workspaces in the future releases and flash using `qsc-cli` won't be supported for these workspaces.

---

b. Update the highlighted command according to your own machine configuration and run it on the terminal:



For example, to build for Qualcomm® RB3 Gen 2 Core Development Kit, change the value of `MACHINE` in the preceding build command to `qcs6490-rb3gen2-core-kit`.

c. After a successful build, check that the `system.img` file is in the `<Base_Workspace_Path>/DEV/LE.QCLINUX.1.0.r1/build-<DISTRO>/tmp-glibc/deploy/images/<MACHINE>/qcom-multimedia-image` directory with an updated timestamp. For example:

```
cd <Base Workspace Path>/build-qcom-wayland/tmp-glibc/deploy/
images/qcs6490-rb3gen2-core-kit/qcom-multimedia-image
ls -al system.img
```

Note: When compiling a software image other than `LE.QCLINUX.1.0.r1`, ensure that you also compile both the software product and `LE.QCLINUX.1.0.r1` in the same order.

For example, if you compile `BOOT.MXF.1.0.c1`, ensure that you also compile the software product (such as `QCM6490.LE.1.0`) and then `LE.QCLINUX.1.0.r1`.

---

3. To flash your build, see Flash software images.

Note:

- Before flashing, update the build images path to the compiled build images workspace

at `<Base_Workspace_Path>/DEV/LE.QCLINUX.1.0.r1/build-<DISTRO>/tmp-glibc/deploy/images/<MACHINE>/qcom-multimedia-image`.

For example, `<Base Workspace Path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-core-kit/qcom-multimedia-image`.

4. To establish UART and network connections, see Connect to UART shell and network.

4 Build with GitHub for unregistered users

The following sections explain how to use GitHub and make a build using the prebuilt proprietary binaries/images.

Note: For more details on the hardware SoCs supported in this build method, see the table *Sync and build methods* in the Release Notes.

4.1 Host computer requirements

Configuration	Tools	Permissions
x86 machine	Git 1.8.3.1 or later versions	A `sudo` permission is required to run a few commands
Quad-core CPU, for example, Intel i7-2600 at 3.4 GHz (equivalent or better)	Tar 1.28 or later versions	
300 GB free disk space (swap partition > 32 GB)	Python 3.10.2 or later versions	
16 GB RAM	GCC 8.0 or later versions	
Ubuntu 22.04	GNU Make 4.0 or later versions	

Note: To set up a virtual machine (VM) running Ubuntu 22.04 on Microsoft® Windows® or Apple® macOS®, see Qualcomm Linux Virtual Machine Setup Guide. Code compilation on a VM is a slow process and may take a few hours. Qualcomm recommends using an Ubuntu host computer for

compilation.

4.2 Build with standalone commands

Ubuntu host setup

Install and configure the required software tools on the Ubuntu host computer.

1. Install the following packages to prepare your host environment for the Yocto build:

```
sudo apt update
sudo apt install repo gawk wget git diffstat unzip texinfo gcc
build-essential chrpath socat cpio python3 python3-pip python3-
pexpect xz-utils debianutils iputils-ping python3-git python3-
jinja2 libegl1-mesa libsdl1.2-dev pylint xterm python3-subunit
mesa-common-dev zstd liblz4-tool locales tar python-is-python3
file libxml-opml-simplegen-perl vim whiptail g++ libacl1
```

2. Set up the locales (if not set up already):

```
sudo locale-gen en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

3. Update git configurations:

```
# Check if your identity is configured in .gitconfig
git config --get user.email
git config --get user.name

# Run the following commands if you don't have your account
identity set in .gitconfig
git config --global user.email <Your email ID>
git config --global user.name <"Your Name">

# Add the following UI color option for output of console
(optional)
git config --global color.ui auto

# Add the following git configurations to fetch large size
repositories and to avoid unreliable connections
git config --global http.postBuffer 1048576000
```

```
git config --global http.maxRequestBuffer 1048576000
git config --global http.lowSpeedLimit 0
git config --global http.lowSpeedTime 999999
```

Yocto layers	Manifest release tag	Reference distribution (DISTRO)

Sync

This section uses the Repo tool installed in Ubuntu host setup to download git repositories and other attributes from the Qualcomm manifest. The Repo tool downloads the manifests using the `repo init` command.

The following table shows an example mapping of the Yocto layers to the manifest release tags. Use this mapping to download and build Qualcomm Linux.

Mapping Yocto layers to manifest release tags

Yocto layers	Manifest release tag	Reference distribution (DISTRO)
• meta-qcom • meta-qcom-hwe • meta-qcom-distro	BSP build: High-level OS and prebuilt firmware (GPS only) qcom-6.6.65-QLI.1.4-Ver.1.1.xml	qcom-wayland
• meta-qcom • meta-qcom-hwe • meta-qcom-distro • meta-qcom-qim-product-sdk	BSP build + QIMP SDK build: qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2.xml	qcom-wayland
• meta-qcom • meta-qcom-hwe • meta-qcom-distro • meta-qcom-realtime	BSP build + Real-time kernel build: qcom-6.6.65-QLI.1.4-Ver.1.1_realtime-linux-1.1.xml	qcom-wayland

Yocto layers	Manifest release tag	Reference distribution (DISTRO)
• meta-qcom • meta-qcom-hwe • meta-qcom-distro • meta-ros • meta-qcom-robotics • meta-qcom-robotics-distro • meta-qcom-robotics-sdk • meta-qcom-qim-product-sdk	BSP build + QIRP SDK build: qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-1.0.xml	qcom-robotics-ros2-humble

Note:

- Syntax for the manifest release tag:

  qcom-<Linux LTS Kernel Version>-QLI.<version>-Ver.<release>.xml

  For example, the manifest release tag qcom-6.6.65-QLI.1.4-Ver.1.1.xml denotes the following:

  – 6.6.65: Qualcomm Linux kernel

  – QLI.1.4: Qualcomm Linux v1.4

  – 1.0: Milestone release

- Syntax for the additional productization manifest release tag:

  qcom-<Linux LTS Kernel version>-QLI.<version>-Ver.<milestone release>_<product/customization>-<patch release>.xml

  For example, the additional productization manifest release tag qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2.xml denotes the following:

  – 6.6.65: Qualcomm Linux kernel

  – QLI.1.4: Qualcomm Linux v1.4

  – qim-product-sdk-1.1.1: QIMP SDK release on top of QLI.1.4

    Other product/customization examples:

    ○ *realtime-linux-1.0*

    - *robotics-product-sdk-1.1*
  - 1.0: Milestone release
  - 1.1.1: Patch release associated with the milestone release
- For more information on the Yocto layers, see Qualcomm Linux metadata layers.

Build BSP image

Board support package (BSP) image build has software components for Qualcomm device support and value-added software features applicable to Qualcomm SoCs. It includes a reference distribution configuration for Qualcomm development kits.

For more details, see Qualcomm Linux metadata layers.

1. Download Qualcomm Yocto and the supporting layers:

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
1.xml
repo sync
```

Note: For the latest `<manifest release tag>`, see the section *Build-Critical Release Tags* in the Release Notes.

2. Set up the build environment:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=
<override> source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

Note: For various `<machine>` and `<override>` combinations, see Release Notes.

3. Build the software image:

Note: For supported image recipes, see Image recipes supported in the GitHub workflow.

```
bitbake <image recipe>
# Example, bitbake qcom-multimedia-image
```

After a successful build, check that the `system.img` file is in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-multimedia-image
ls -al system.img
```

Build QIMP SDK image

The Qualcomm® Intelligent Multimedia Product (QIMP) SDK is a collection of four standalone function SDKs, namely, Qualcomm® Intelligent Multimedia SDK (IM SDK), Qualcomm® Neural Processing SDK, Qualcomm® AI Engine direct SDK, and Lite Runtime (LiteRT). It also includes reference applications that you can use to develop use cases.

For more details, see QIMP SDK Quick Start Guide.

1. Download Qualcomm Yocto and the supporting layers:

Note:

- The `<manifest release tag>` for the QIMP SDK build is the same as the BSP build. Clone the QIMP SDK layer on top of the BSP build.
- For the latest `<manifest release tag>`, see the section *Build-Critical Release Tags* in the Release Notes.

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
1.xml
repo sync
```

2. Clone the QIMP SDK layer into the workspace:

```
git clone https://github.com/quic-yocto/meta-qcom-qim-product-
sdk -b <meta-qcom-qim-product-sdk release tag> layers/meta-qcom-
qim-product-sdk
# Example, <meta-qcom-qim-product-sdk release tag> is qcom-6.6.
65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2
```

   Build the QIMP SDK layer:

```
export EXTRALAYERS="meta-qcom-qim-product-sdk"
```

3. Set up the build environment:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

Note: To know the `MACHINE` parameter values, see Release Notes.

4. Build the software image:

```
bitbake qcom-multimedia-image
# Build SDK image
bitbake qcom-qim-product-sdk
```

   After a successful build, check that the `system.img` file is in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-multimedia-image
ls -al system.img
```

Build QIRP SDK image

The Qualcomm® Intelligent Robotics Product (QIRP) SDK 2.0 is a collection of components that enable you to develop robotic features on Qualcomm platforms. This SDK is applicable to the Qualcomm Linux releases.

For more details, see QIRP SDK 2.0 User Guide.

1. Download Qualcomm Yocto and the supporting layers:

Note: The <manifest release tag> for QIRP SDK build is the same as the BSP build. Clone the QIRP SDK layers on top of the BSP build.

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
1.xml
repo sync
```

Note: For the latest <manifest release tag>, see the section *Build-Critical Release Tags* in the Release Notes.

2. Download the QIRP SDK layers into the BSP build <WORKSPACE DIR> directory:

```
git clone https://github.com/ros/meta-ros -b scarthgap layers/
meta-ros && cd layers/meta-ros && git checkout
c560699e810e60a9526f4226c2c23f8d877280c8 && cd ../../
git clone https://github.com/quic-yocto/meta-qcom-robotics.git -
b qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-1.0 layers/
meta-qcom-robotics
git clone https://github.com/quic-yocto/meta-qcom-robotics-
distro.git -b qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-
1.0 layers/meta-qcom-robotics-distro
git clone https://github.com/quic-yocto/meta-qcom-robotics-sdk.
git -b qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-1.0
layers/meta-qcom-robotics-sdk
git clone https://github.com/quic-yocto/meta-qcom-qim-product-
sdk -b qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2 layers/
meta-qcom-qim-product-sdk
```

3. Set up the build environment:

```
ln -s layers/meta-qcom-robotics-distro/set_bb_env.sh ./setup-
robotics-environment
ln -s layers/meta-qcom-robotics-sdk/scripts/qirp-build ./qirp-
build
MACHINE=<machine> DISTRO=qcom-robotics-ros2-humble QCOM_
SELECTED_BSP=<override> source setup-robotics-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
robotics-ros2-humble QCOM_SELECTED_BSP=custom source setup-
robotics-environment
# source setup-robotics-environment: Sets the environment,
creates the build directory build-qcom-robotics-ros2-humble,
# and enters into build-qcom-robotics-ros2-humble directory.
```

Note: For various <machine> and <override> combinations, see Release Notes.

4. Build the robotics image and the QIRP SDK artifacts:

```
../qirp-build qcom-robotics-full-image
```

After a successful build, check that the QIRP SDK build artifacts are at the following paths:

```
QIRP SDK artifacts: <WORKSPACE DIR>/build-qcom-robotics-ros2-
humble/tmp-glibc/deploy/qirpsdk_artifacts/qirp-sdk_<version>.
tar.gz
# system.img is present in the following path
Robotics image: <WORKSPACE DIR>/build-qcom-robotics-ros2-humble/
tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-
robotics-full-image
```

Build real-time Linux image

The real-time layer provides recipes and configurations required to run the Qualcomm Linux kernel as a real-time kernel. The real-time kernel runs with preemption fully enabled through a configuration, `CONFIG_PREEMPT_RT=y`. This layer supports `linux-kernel-qcom-rt` recipe that fetches and builds the Qualcomm Linux kernel for the supported machine. This layer appends to the kernel and the upstream `PREEMPT_RT` patches based on the kernel version, and enables real-time configurations.

For more details, see Real-time kernel.

1. Download Qualcomm Yocto and the supporting layers:

   Note: The `<manifest release tag>` for real-time Linux image is the same as the BSP build. Clone the real-time Linux on top of the BSP build.

   ```
   # cd to directory where you have 300 GB of free storage space to
   create your workspaces
   mkdir <WORKSPACE_DIR>
   cd <WORKSPACE_DIR>
   repo init -u https://github.com/quic-yocto/qcom-manifest -b
   qcom-linux-scarthgap -m <manifest release tag>
   # Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
   1.xml
   repo sync
   ```

   Note: For the latest `<manifest release tag>`, see the section *Build-Critical Release Tags* in the Release Notes.

2. Clone the real-time Linux layer into the workspace:

   ```
   git clone https://github.com/quic-yocto/meta-qcom-realtime -b
   <meta-qcom-realtime release tag> layers/meta-qcom-realtime
   # Example, <meta-qcom-realtime release tag> is qcom-6.6.65-QLI.
   1.4-Ver.1.1_realtime-linux-1.1
   ```

   Build the real-time layer:

   ```
   export EXTRALAYERS="meta-qcom-realtime"
   ```

3. Set up the build environment:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=
<override> source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

Note: For various <machine> and <override> combinations, see Release Notes.

4. Build the software image:

```
bitbake qcom-multimedia-image
```

After a successful build, check that the system.img file is in the <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-image directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-multimedia-image
ls -al system.img
```

Flash

To flash the software images to the device, see Flash software images.

5 Build with GitHub for registered users

The following sections explain how to create a build using the prebuilt proprietary binaries/images available on GitHub.

5.1 Host computer requirements

Configuration	Tools	Permissions
x86 machine	Git 1.8.3.1 or later versions	A `sudo` permission is required to run a few commands
Quad-core CPU, for example, Intel i7-2600 at 3.4 GHz (equivalent or better)	Tar 1.28 or later versions	
300 GB free disk space (swap partition > 32 GB)	Python 3.10.2 or later versions	
16 GB RAM	GCC 8.0 or later versions	
Ubuntu 22.04	GNU Make 4.0 or later versions	

Note: To set up a virtual machine (VM) running Ubuntu 22.04 on Microsoft® Windows® or Apple® macOS®, see Qualcomm Linux Virtual Machine Setup Guide. Code compilation on a VM is a slow process and may take a few hours. Qualcomm recommends using an Ubuntu host computer for compilation.

5.2 Install QSC CLI

1. Install curl (if not installed already):

```
sudo apt install curl
```

2. Download the Debian package for `qsc-cli`:

```
cd <workspace_path>
curl -L https://softwarecenter.qualcomm.com/api/download/
software/qsc/linux/latest.deb -o qsc_installer.deb
```

3. Install the `qsc-cli` Debian package:

```
sudo apt update
sudo apt install ./qsc_installer.deb
```

4. Sign in to `qsc-cli` using your registered email ID:

Note: To register, go to https://www.qualcomm.com/support/working-with-qualcomm.

```
qsc-cli login -u <username>
```

Note: For more information, see `qsc-cli` related topics in How to Sync.

5.3 Workflow options

The two workflows for registered users to set up, sync, and build Qualcomm Linux with GitHub are described in the following table:

Standalone Commands	docker
Build public Qualcomm Yocto layers with standalone commands.	Build public Qualcomm Yocto layers with Dockerfile.
Build with standalone commands	Build with Dockerfile

5.4 Build with standalone commands

Ubuntu host setup

The Ubuntu host computer must be set up to install the required software tools and configure them for use.

1. Install the following packages to prepare your host environment for the Yocto build:

```
sudo apt update
sudo apt install repo gawk wget git diffstat unzip texinfo gcc
build-essential chrpath socat cpio python3 python3-pip python3-
pexpect xz-utils debianutils iputils-ping python3-git python3-
jinja2 libegl1-mesa libsdl1.2-dev pylint xterm python3-subunit
mesa-common-dev zstd liblz4-tool locales tar python-is-python3
file libxml-opml-simplegen-perl vim whiptail g++ libacl1
```

2. Add your Qualcomm Log in ID with Personalized Access Token (PAT) to the `~/.netrc` file in your home directory:

```
# Log in to qsc-cli to generate PAT
qsc-cli login -u <username>
# Run the following command to generate PAT
qsc-cli pat --get
# This command gives output as shown in the following note
# The last line in this output is the token, which can be used
to access
# Qualcomm Proprietary repositories. This token expires in two
weeks.
```

Note:

user@hostname:/local/mnt/workspace$ qsc-cli pat –get
[Info]: Starting qsc-cli version 0.0.0.9
5LThNIklb55mMVLB5C2KqUGU2jCF

3. Use your preferred text editor to edit the `~/.netrc` file and add the following entries:

Note: Create the `~/.netrc` file if it doesn't exist.

```
machine chipmaster2.qti.qualcomm.com
login <your Qualcomm login id>
password <your PAT token>
```

```
machine qpm-git.qualcomm.com
login <your Qualcomm login id>
password <your PAT token>
```

4. Set up the locales (if not set up already):

```
sudo locale-gen en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

5. Update git configurations:

```
# Check if your identity is configured in .gitconfig
git config --get user.email
git config --get user.name

# Run the following commands if you don't have your account
identity set in .gitconfig
git config --global user.email <Your email ID>
git config --global user.name <"Your Name">

# Add the following UI color option for output of console
(optional)
git config --global color.ui auto

# Add the following git configurations to fetch large size
repositories and to avoid unreliable connections
git config --global http.postBuffer 1048576000
git config --global http.maxRequestBuffer 1048576000
git config --global http.lowSpeedLimit 0
git config --global http.lowSpeedTime 999999

# Add the following git configurations to follow remote
redirects from http-alternates files or alternates
git config --global http.https://chipmaster2.qti.qualcomm.com.
followRedirects true
git config --global http.https://qpm-git.qualcomm.com.
followRedirects true
```

Sync

This section uses the Repo tool installed in Ubuntu host setup to download a list of git repositories and additional attributes from the Qualcomm manifest. The Repo tool downloads the manifests using the `repo init` command.

The following table shows an example mapping of the Yocto layers to the manifest release tags. Use this mapping to download and build Qualcomm Linux:

Mapping Yocto layers to manifest release tags

Yocto layers	Manifest release tag	Reference distribution (DISTRO)
• meta-qcom • meta-qcom-hwe • meta-qcom-distro	BSP build: High-level OS and prebuilt firmware (GPS only) qcom-6.6.65-QLI.1.4-Ver.1.1.xml	qcom-wayland
• meta-qcom • meta-qcom-hwe • meta-qcom-distro • meta-qcom-qim-product-sdk	BSP build + QIMP SDK build: qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2.xml	qcom-wayland
• meta-qcom • meta-qcom-hwe • meta-qcom-distro • meta-qcom-realtime	BSP build + Real-time kernel build: qcom-6.6.65-QLI.1.4-Ver.1.1_realtime-linux-1.1.xml	qcom-wayland

Yocto layers	Manifest release tag	Reference distribution (DISTRO)
• meta-qcom • meta-qcom-hwe • meta-qcom-distro • meta-ros • meta-qcom-robotics • meta-qcom-robotics-distro • meta-qcom-robotics-sdk • meta-qcom-qim-product-sdk	BSP build + QIRP SDK build: qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-1.0.xml	qcom-robotics-ros2-humble

Note:

- For more information on the Yocto layers, see Qualcomm Linux metadata layers.
- For information on building the meta-qcom-extras add-on layer and select firmware sources, see Build with GitHub using firmware and extras.

Build BSP image

BSP image build has software components for Qualcomm device support and value-added software features applicable to Qualcomm SoCs. It includes a reference distribution configuration for Qualcomm development kits.

For more details, see Qualcomm Linux metadata layers.

1. Download Qualcomm Yocto and the supporting layers:

```
# cd to directory where you have 300 GB of free storage space to create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
```

```
1.xml
repo sync
```

Note: For the latest <manifest release tag>, see the section *Build-critical release tags* in the Release Notes.

2. Set up the build environment:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=
<override> source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

Note: For various <machine> and <override> combinations, see Release Notes.

3. Build the software image:

Note: For supported image recipes, see Image recipes supported in the GitHub workflow.

```
bitbake <image recipe>
# Example, bitbake qcom-multimedia-image
```

After a successful build, you can verify if system.img is present in the <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-image directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-multimedia-image
ls -al system.img
```

Build QIMP SDK image

The QIMP SDK is a collection of four standalone function SDKs, namely, IM SDK, Qualcomm Neural Processing SDK, Qualcomm AI Engine direct SDK, and LiteRT. It also includes reference applications that you can use to develop use cases.

For more details, see QIMP SDK Quick Start Guide.

1. Download Qualcomm Yocto and the supporting layers:

   Note: The <manifest release tag> for QIMP SDK build is the same as the BSP build. Clone the QIMP SDK layer on top of the BSP build.

   ```
   # cd to directory where you have 300 GB of free storage space to
   create your workspaces
   mkdir <WORKSPACE_DIR>
   cd <WORKSPACE_DIR>
   repo init -u https://github.com/quic-yocto/qcom-manifest -b
   qcom-linux-scarthgap -m <manifest release tag>
   # Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
   1.xml
   repo sync
   ```

   Note: For the latest <manifest release tag>, see the section *Build-critical release tags* in the Release Notes.

2. Clone the QIMP SDK layer into the workspace:

   ```
   git clone https://github.com/quic-yocto/meta-qcom-qim-product-
   sdk -b <meta-qcom-qim-product-sdk release tag> layers/meta-qcom-
   qim-product-sdk
   # Example, <meta-qcom-qim-product-sdk release tag> is qcom-6.6.
   65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2
   ```

   To build a QIMP SDK layer, the following export is required:

   ```
   export EXTRALAYERS="meta-qcom-qim-product-sdk"
   ```

3. Set up the build environment:

   ```
   MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
   source setup-environment
   # Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
   ```

```
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

Note: For information about the `MACHINE` parameter values, see Release Notes.

4. Build the software image:

```
bitbake qcom-multimedia-image
# Build SDK image
bitbake qcom-qim-product-sdk
```

After a successful build, you can verify if `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-multimedia-image
ls -al system.img
```

Build QIRP SDK image

The Qualcomm® Intelligent Robotics Product (QIRP) SDK 2.0 is a collection of components that enable you to develop robotic features on Qualcomm Linux releases.

For more details, see QIRP SDK 2.0 User Guide.

1. Download Qualcomm Yocto and the supporting layers:

Note: The `<manifest release tag>` for QIRP SDK build is the same as the BSP build. Clone the QIRP SDK layer on top of the BSP build.

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
1.xml
repo sync
```

Note: For the latest <manifest release tag>, see the section *Build-critical release tags* in the Release Notes.

2. Download the QIRP SDK layers into the BSP build <WORKSPACE DIR> directory:

```
git clone https://github.com/ros/meta-ros -b scarthgap layers/
meta-ros && cd layers/meta-ros && git checkout
c560699e810e60a9526f4226c2c23f8d877280c8 && cd ../../
git clone https://github.com/quic-yocto/meta-qcom-robotics.git -
b qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-1.0 layers/
meta-qcom-robotics
git clone https://github.com/quic-yocto/meta-qcom-robotics-
distro.git -b qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-
1.0 layers/meta-qcom-robotics-distro
git clone https://github.com/quic-yocto/meta-qcom-robotics-sdk.
git -b qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-1.0
layers/meta-qcom-robotics-sdk
git clone https://github.com/quic-yocto/meta-qcom-qim-product-
sdk -b qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2 layers/
meta-qcom-qim-product-sdk
```

3. Set up the build environment:

```
ln -s layers/meta-qcom-robotics-distro/set_bb_env.sh ./setup-
robotics-environment
ln -s layers/meta-qcom-robotics-sdk/scripts/qirp-build ./qirp-
build
MACHINE=<machine> DISTRO=qcom-robotics-ros2-humble QCOM_
SELECTED_BSP=<override> source setup-robotics-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
robotics-ros2-humble QCOM_SELECTED_BSP=custom source setup-
robotics-environment
# source setup-robotics-environment: Sets the environment,
creates the build directory build-qcom-robotics-ros2-humble,
# and enters into build-qcom-robotics-ros2-humble directory.
```

Note: For various <machine> and <override> combinations, see Release Notes.

4. Build the robotics image and the QIRP SDK artifacts:

```
../qirp-build qcom-robotics-full-image
```

After a successful build, you can see the QIRP SDK build artifacts at the following paths:

```
QIRP SDK artifacts: <WORKSPACE DIR>/build-qcom-robotics-ros2-
humble/tmp-glibc/deploy/qirpsdk_artifacts/qirp-sdk_<version>.
tar.gz
# system.img is present in the following path
Robotics image: <WORKSPACE DIR>/build-qcom-robotics-ros2-humble/
tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-
robotics-full-image
```

Build real-time Linux image

The real-time layer provides recipes and configurations required to run the Qualcomm Linux kernel as a real-time kernel. The real-time kernel runs with preemption fully enabled through a configuration, `CONFIG_PREEMPT_RT=y`. This layer supports `linux-kernel-qcom-rt` recipe that fetches and builds the Qualcomm Linux kernel for the supported machine. This layer appends to the kernel and the upstream `PREEMPT_RT` patches based on the kernel version, and enables real-time configurations.

For more details, see Real-time kernel.

1. Download Qualcomm Yocto and the supporting layers:

Note: The `<manifest release tag>` for real-time Linux image is the same as the BSP build. Clone the Real-time Linux on top of the BSP build.

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
1.xml
repo sync
```

Note: For the latest `<manifest release tag>`, see the section *Build-critical release tags* in the Release Notes.

2. Clone the real-time Linux layer into the workspace:

```
git clone https://github.com/quic-yocto/meta-qcom-realtime -b
<meta-qcom-realtime release tag> layers/meta-qcom-realtime
# Example, <meta-qcom-realtime release tag> is qcom-6.6.65-QLI.
1.4-Ver.1.1_realtime-linux-1.1
```

To build a real-time layer, the following export is required:

```
export EXTRALAYERS="meta-qcom-realtime"
```

3. Set up the build environment:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=
<override> source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

Note: For various `<machine>` and `<override>` combinations, see Release Notes.

4. Build the software image:

```
bitbake qcom-multimedia-image
```

After a successful build, check that the `system.img` file is in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-multimedia-image
ls -al system.img
```

Flash

To flash the software images to the device, see Flash software images.

5.5 Build with Dockerfile

Ubuntu host setup

Install and configure the required software tools on the Ubuntu host computer.

- Install git:

```
# Install git if you haven't already installed
sudo apt install git
```

- Clone the `qcom-download-utils` git repository, which provides a Dockerfile for Qualcomm public Yocto layers and a few helper scripts:

```
mkdir <workspace_path>
cd <workspace_path>
git clone https://git.codelinaro.org/clo/le/qcom-download-utils.
git
cd qcom-download-utils
```

- Add the user to the Docker group:

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
# To check if you are part of a Docker group, run the following
command:
sudo grep /etc/group -e "docker"
```

Check the host computer configuration

- Configure your host computer:

```
bash utils/check_config.sh
# Resolve the errors and run this command until no errors show
up
```

- Install Docker:

```
bash docker/docker_setup.sh
```

Note: The following figure shows the directory structure of `qcom-download-utils` in relation to the Docker setup:

```
├── docker
│   ├── docker_build.sh
│   ├── dockerfiles
│   │   ├── Dockerfile_20.04
│   │   └── Dockerfile_22.04
│   ├── docker_run.sh
│   └── docker_setup.sh
├── LICENSE
├── qcom-6.6.00-QLI.1.0-Ver.1.1
│   └── config.sh
├── qcom-6.6.13-QLI.1.0-Ver.1.2
│   └── config.sh
├── qcom-6.6.13-QLI.1.0-Ver.1.3
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.3
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.3_qim-product-sdk-1.1
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.3_realtime-linux-1.0
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.4
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.4_qim-product-sdk-1.1
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.4_realtime-linux-1.0
│   └── config.sh
├── qcom-6.6.28-QLI.1.1-Ver.1.0
│   └── config.sh
├── qcom-6.6.28-QLI.1.1-Ver.1.0_qim-product-sdk-1.1.1
│   └── config.sh
├── qcom-6.6.28-QLI.1.1-Ver.1.1
│   └── config.sh
├── qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-sdk-1.1.3
│   └── config.sh
├── qcom-6.6.28-QLI.1.1-Ver.1.1_realtime-linux-1.0
│   └── config.sh
├── qcom-6.6.38-QLI.1.2-Ver.1.0
│   └── config.sh
├── qcom-6.6.38-QLI.1.2-Ver.1.0_qim-product-sdk-1.1.1
│   └── config.sh
├── qcom-6.6.38-QLI.1.2-Ver.1.0_realtime-linux-1.0
│   └── config.sh
├── README.md
└── utils
    ├── check_config.sh
    └── sync_build.sh
```

Build BSP image

Create a Yocto Docker image and build:

1. Run `docker_build.sh` to create the Docker image with Dockerfile (**Dockerfile_22.04**) and Dockertag (**qcom-6.6.65-qli.1.4-ver.1.1_22.04**). Use this Docker image to create the container environment and run the Yocto build.

   Dockertag: Use lowercase letters for the release folder followed by the Dockerfile OS version, to identify the release build with the Dockerfile. Docker doesn't allow uppercase letters in the Dockertag.

   Note: To troubleshoot Docker issues, see Troubleshoot Docker.

```
bash docker/docker_build.sh -f ./docker/dockerfiles/Dockerfile_
22.04 -t qcom-6.6.65-qli.1.4-ver.1.1_22.04
```

   If you face any issues while running `docker_build.sh`, see the following solution:

```
# Error 1: Cache-related issue.
   # If you are facing issues with the docker build command, try
using --no-cache option. This option forces rebuilding of the
layers that are already available
   bash docker/docker_build.sh -n --no-cache -f ./docker/
dockerfiles/Dockerfile_22.04 -t qcom-6.6.65-qli.1.4-ver.1.1_22.
04

# Error2: Response from daemon: Get "https://registry-1.docker.
io/v2/": http: server gave HTTP response to HTTPS client
   # Check with your IT administrator to acquire ``registry-
mirrors`` URL and replace <docker-mirror-host>`` in the following
solution
   # Using a tab instead of space and other invisible white-
space characters may break the proper functioning of the JSON
configuration files
   # and later may lead to the Docker service failing to start

   # Solution:
   sudo vim /etc/docker/daemon.json
   # Add an entry similar to the following in /etc/docker/
daemon.json:
   {
      "registry-mirrors": ["https://<docker-mirror-host>"]
   }
```

```
   # Restart the Docker service to take the new settings
   sudo systemctl restart docker
```

2. Sync and build the Yocto image in a Docker container with the Docker tag and release parameters:

```
bash docker/docker_run.sh -t qcom-6.6.65-qli.1.4-ver.1.1_22.04 -
r qcom-6.6.65-QLI.1.4-Ver.1.1 -M <machine> --build-override
<override> --alternate-repo true
# Example, bash docker/docker_run.sh -t qcom-6.6.65-qli.1.4-ver.
1.1_22.04 -r qcom-6.6.65-QLI.1.4-Ver.1.1 -M qcs6490-rb3gen2-
vision-kit --build-override custom --alternate-repo true
```

Note: For various `<machine>` and `<override>` combinations, see Release Notes.

The build workspace is available in `<qcom-download-utils download path>/<release>/build-qcom-wayland`. For example, `qcom-download-utils/qcom-6.6.65-QLI.1.4-Ver.1.1/build-qcom-wayland`.

Note:

- **# ERROR: error.GitError: git config (‘–replace-all’,‘color.ui’, ‘auto’): error: couldn’t write config file /home/$USER/.gitconfig: Device or resource busy**

  As Git 1.8.4 is enabled by default, you will see this error when the UI color configuration is not set in gitconfig. To enable color display in your account, run the following command: `git config --global color.ui auto`.

- If you see and fix a build error, run the commands in Rebuild.

Build QIMP SDK image

1. Build BSP image using Docker.
2. Build QIMP SDK on top of the base image using Docker:
   a. Run the `docker run` command:

```
# Run the following commands inside the base image build
location
cd <workspace_path>/qcom-download-utils/qcom-6.6.65-QLI.1.4-
Ver.1.1
bash
docker run -it -v "${HOME}/.gitconfig":"/home/${USER}/.
```

```
gitconfig" -v "${HOME}/.netrc":"/home/${USER}/.netrc" -v
$(pwd):$(pwd) -w $(pwd) qcom-6.6.65-qli.1.4-ver.1.1_22.04 /
bin/bash
```

b. Clone the QIMP SDK layer into the workspace:

```
git clone https://github.com/quic-yocto/meta-qcom-qim-
product-sdk -b <meta-qcom-qim-product-sdk release tag>
layers/meta-qcom-qim-product-sdk
# Example, <meta-qcom-qim-product-sdk release tag> is qcom-6.
6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2
```

Build the QIMP SDK layer:

```
export EXTRALAYERS="meta-qcom-qim-product-sdk"
```

c. Set up the build environment:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_
BSP=custom source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

Note: For the `MACHINE` parameter values, see Release Notes.

d. Build the software image:

```
bitbake qcom-multimedia-image
# Build SDK image
bitbake qcom-qim-product-sdk
```

Rebuild

To rebuild after any modifications to the software release, use your existing workspace built using Docker:

1. List the Docker images:

```
docker images
```

   This command generates the following output:

```
REPOSITORY                                                      TAG
                    IMAGE ID       CREATED        SIZE
qcom-6.6.65-qli.1.4-ver.1.1_22.04                               latest
                    8fcea388d8ca   2 days ago     1.47GB
```

2. Attach the container:

```
# Run the following commands outside the Docker container
cd <workspace_path>/qcom-download-utils/qcom-6.6.65-QLI.1.4-Ver.
1.1

# Run the following commands inside the base image build
location
bash
docker run -it -v "${HOME}/.gitconfig":"/home/${USER}/.gitconfig
" -v "${HOME}/.netrc":"/home/${USER}/.netrc" -v $(pwd):$(pwd) -w
$(pwd) qcom-6.6.65-qli.1.4-ver.1.1_22.04 /bin/bash

# Example
WORKSPACE=<workspace_path>/qcom-download-utils/qcom-6.6.65-QLI.
1.4-Ver.1.1
```

3. Set up the build environment:

```
# cd <release directory>
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=
<override> source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

Note: For various <machine> and <override> combinations, see Release Notes.

4. Build the software image:

```
bitbake qcom-multimedia-image
```

Note: Close Docker before you flash the images.

Flash

To flash the software images to the device, see Flash software images.

6 Build with GitHub using firmware and extras

Note: This information is applicable only for authorized users. To upgrade your access, go to http://www.qualcomm.com/support/working-with-qualcomm.

Use this information to build various Qualcomm Linux Yocto layers including the `meta-qcom-hwe`, `meta-qcom-extras`, `meta-qcom-qim-product-sdk`, and `meta-qcom-robotics-sdk` layers using selective proprietary sources and binaries/libraries.

6.1 Host computer requirements

Configuration	Tools	Permissions
x86 machine	Git 1.8.3.1 or later versions	A `sudo` permission is required to run a few commands
Quad-core CPU, for example, Intel i7-2600 at 3.4 GHz (equivalent or better)	Tar 1.28 or later versions	
300 GB free disk space (swap partition > 32 GB)	Python 3.10.2 or later versions	
16 GB RAM	GCC 8.0 or later versions	
Ubuntu 22.04	GNU Make 4.0 or later versions	

Note: To set up a virtual machine (VM) running Ubuntu 22.04 on Microsoft® Windows® or Apple® macOS®, see Qualcomm Linux Virtual Machine Setup Guide. Code compilation on a VM is a slow process and may take a few hours. Qualcomm recommends using an Ubuntu host computer for

compilation.

6.2 Install QSC CLI

1. Install curl (if not installed already):

```
sudo apt install curl
```

2. Download the Debian package for `qsc-cli`:

```
cd <workspace_path>
curl -L https://softwarecenter.qualcomm.com/api/download/
software/qsc/linux/latest.deb -o qsc_installer.deb
```

3. Install the `qsc-cli` Debian package:

```
sudo apt update
sudo apt install ./qsc_installer.deb
```

4. Sign in to `qsc-cli` using your registered email ID:

Note: To register, go to https://www.qualcomm.com/support/working-with-qualcomm.

```
qsc-cli login -u <username>
```

Note: For more information, see `qsc-cli` related topics in How to Sync.

6.3 Ubuntu host setup

The Ubuntu host computer must be set up to install the required software tools and configure them for use.

1. Install the following packages:

```
sudo apt update
sudo apt install repo gawk wget git diffstat unzip texinfo gcc
build-essential chrpath socat cpio python3 python3-pip python3-
pexpect xz-utils debianutils iputils-ping python3-git python3-
jinja2 libegl1-mesa libsdl1.2-dev pylint xterm python3-subunit
```

```
mesa-common-dev zstd liblz4-tool locales tar python-is-python3
file libxml-opml-simplegen-perl vim whiptail g++ libacl1
sudo apt-get install lib32stdc++6 libncurses5 checkinstall
libreadline-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-
dev libgdbm-dev libc6-dev libbz2-dev libffi-dev curl
```

2. Add your Qualcomm login ID with PAT to the `~/.netrc` file in your home directory:

```
# Log in to qsc-cli to generate PAT
qsc-cli login -u <username>
# Run the following command to generate PAT
qsc-cli pat --get
# This command gives output as shown in the following note
# The last line in this output is the token, which can be used
to access
# Qualcomm Proprietary repositories. This token expires in two
weeks.
```

Note:

user@hostname:/local/mnt/workspace$ qsc-cli pat –get
[Info]: Starting qsc-cli version 0.0.0.9
5LThNIkIb55mMVLB5C2KqUGU2jCF

3. Use your preferred text editor to edit the `~/.netrc` file and add the following entries:

Note: Create the `~/.netrc` file if it doesn't exist.

```
machine chipmaster2.qti.qualcomm.com
login <your Qualcomm login id>
password <your PAT token>

machine qpm-git.qualcomm.com
login <your Qualcomm login id>
password <your PAT token>
```

4. Set up the locales (if not set up already):

```
sudo locale-gen en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
```

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

5. Update git configurations:

```
# Check if your identity is configured in .gitconfig
git config --get user.email
git config --get user.name

# Run the following commands if you don't have your account
identity set in .gitconfig
git config --global user.email <Your email ID>
git config --global user.name <"Your Name">

# Add the following UI color option for output of console
(optional)
git config --global color.ui auto

# Add the following git configurations to fetch large size
repositories and to avoid unreliable connections
git config --global http.postBuffer 1048576000
git config --global http.maxRequestBuffer 1048576000
git config --global http.lowSpeedLimit 0
git config --global http.lowSpeedTime 999999

# Add the following git configurations to follow remote
redirects from http-alternates files or alternates
git config --global http.https://chipmaster2.qti.qualcomm.com.
followRedirects true
git config --global http.https://qpm-git.qualcomm.com.
followRedirects true
```

6. Set up Python 3.10.2:

Note: Skip the following instructions if you already have Python 3.10.2 or later versions.

```
python --version
# Download it in a directory of your choice
wget https://www.python.org/ftp/python/3.10.2/Python-3.10.2.tgz
tar -xvf Python-3.10.2.tgz
cd Python-3.10.2
# Use sudo if you need to access /opt
./configure --prefix=/opt/python3 --enable-optimizations
```

```
make
make install
ln -s /opt/python3/bin/python3  /opt/python3/bin/python
export PATH=/opt/python3/bin:$PATH
export PYTHONPATH=/opt/python3/lib:$PYTHONPATH
```

6.4 Build with firmware sources

Sync firmware

The following table describes the Qualcomm Yocto layers and release tags:

Qualcomm Yocto layers and manifest tags

Access level	Yocto layer	Release tag	Example
Public developers (unregistered)	meta-qcom-hwe	manifest release tag	qcom-6.6.65-QLI.1.4-Ver.1.1.xml
	meta-qcom-qim-product-sdk	manifest release tag	qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2.xml
	meta-qcom-robotics-sdk	manifest release tag	qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-1.0.xml
Licensed developers with authorized access	meta-qcom-extras	meta-qcom-extras release tag	r1.0_00077.0
See Mapping access levels to firmware distributions	NA	firmware release tag	r1.0_00075.0

The following tables describe the firmware distributions you can download according to the need and entitlements:

Mapping access levels to firmware distributions

Access level	Distribution	Yocto layers
Licensed developers with authorized access	BSP build: High-level OS and firmware source (GPS only) Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem	meta-qcom meta-qcom-hwe meta-qcom-distro meta-qcom-extras

Access level	Distribution	Yocto layers

	BSP build + QIMP SDK
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NM_QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-distro`
		`meta-qcom-extras`
		`meta-qcom-qim-product-sdk`

	BSP build + QIMP SDK + QIRP SDK
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NM_QIRPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-distro`
		`meta-qcom-extras`
		`meta-qcom-robotics-extras`
		`meta-ros`
		`meta-qcom-robotics`
		`meta-qcom-robotics-distro`
		`meta-qcom-robotics-sdk`
		`meta-qcom-qim-product-sdk`

Licensed developers (contact Qualcomm for access)	BSP build: High-level OS and firmware (GPS only) source
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-distro`
		`meta-qcom-extras`

	BSP build + QIMP SDK (GPS only)
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-distro`
		`meta-qcom-extras`
		`meta-qcom-robotics-extras`
		`meta-qcom-qim-product-sdk`

Access level	Distribution	Yocto layers

	BSP build: High-level OS and firmware (GPS and modem) source
	`Qualcomm_ Linux.SPF. 1.0|AMSS| Standard| OEM|`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-distro`
		`meta-qcom-extras`

	BSP build + QIMP SDK (GPS and modem)
	`Qualcomm_ Linux.SPF. 1.0|AMSS| Standard| OEM|QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-distro`
		`meta-qcom-extras`
		`meta-qcom-qim-product-sdk`

Note: For more information on the Yocto layers, see Qualcomm Linux metadata layers.

Mapping firmware distributions to git repositories

Firmware distribution	Git command	Directory into which firmware gets synced on git clone
Qualcomm_Linux.SPF.1.0| AP|Standard|OEM|NoModem	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem.git`	`qualcomm-linux-spf-1-0_ap_standard_oem_nomodem`

Firmware distribution	**Git command**	**Directory into which firmware gets synced on git clone**
Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NM_QIMPSDK	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk.git`	`qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk`
Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NM_QIRPSDK	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk.git`	`qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk`
Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem.git`	`qualcomm-linux-spf-1-0_ap_standard_oem`
Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|QIMPSDK	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem_qimpsdk.git`	`qualcomm-linux-spf-1-0_ap_standard_oem_qimpsdk`

Firmware distribution	Git command	Directory into which firmware gets synced on git clone
Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_amss_standard_oem.git`	`qualcomm-linux-spf-1-0_amss_standard_oem`
Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|QIMPSDK	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_amss_standard_oem_qimpsdk.git`	`qualcomm-linux-spf-1-0_amss_standard_oem_qimpsdk`

Note: Commands in the following sections are based on the binary and source for firmware images without modem and GPS (see the command in Mapping firmware distributions to git repositories). Hence, `qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk` is used. If you use any other distribution, then update the directory accordingly.

The **Git command** column (see Mapping firmware distributions to git repositories) provides information about the git repositories that contain the firmware sources. Qualcomm servers host these repositories. Clone the appropriate repositories based on your access profile and use case.

The following `git clone` command downloads the selected firmware components in source, except the modem:

```
mkdir -p <FIRMWARE_ROOT>
cd <FIRMWARE_ROOT>
git clone -b <firmware release tag> --depth 1 https://qpm-git.
qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_
oem_nm-qimpsdk.git
# Example, <firmware release tag> is r1.0_00075.0
```

Note:

- The `git clone` command clones the content into the `<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk` directory.
- For the latest `<firmware release tag>`, see the section *Build-critical release tags* in the Release Notes.

Build firmware

QCS6490/QCS5430

Prerequisites

- Ensure that the working shell is `bash`.

```
echo $0
```

  The expected output of the command should be `bash`. If not, enter the bash shell:

```
bash
```

- Install the libffi6 package using the following commands. This is required for the QAIC compiler, which generates the header and the source files from the IDL files:

```
curl -LO http://archive.ubuntu.com/ubuntu/pool/main/libf/libffi/
libffi6_3.2.1-8_amd64.deb
sudo dpkg -i libffi6_3.2.1-8_amd64.deb
```

- Install LLVM for AOP, Qualcomm® Trusted Execution Environment (TEE), and boot compilation:

```
cd <FIRMWARE_ROOT>
mkdir llvm

# Log in to qpm-cli and activate the license
qpm-cli --login
qpm-cli --license-activate sdllvm_arm

# LLVM requirement for boot compilation is 14.0.4
qpm-cli --install sdllvm_arm --version 14.0.4 --path <FIRMWARE_
ROOT>/llvm/14.0.4
chmod -R 777 <FIRMWARE_ROOT>/llvm/14.0.4

# LLVM requirement for Qualcomm TEE compilation is 16.0.7
qpm-cli --install sdllvm_arm --version 16.0.7 --path <FIRMWARE_
```

```
ROOT>/llvm/16.0.7
chmod -R 777 <FIRMWARE_ROOT>/llvm/16.0.7
```

- Export the `SECTOOLS` variable and compile the firmware builds (`<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk` is the top-level directory):

```
export SECTOOLS=<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_
standard_oem_nm-qimpsdk/QCM6490.LE.1.0/common/sectoolsv2/ext/
Linux/sectools
export SECTOOLS_DIR=<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_
standard_oem_nm-qimpsdk/QCM6490.LE.1.0/common/sectoolsv2/ext/
Linux
```

- Install and set up Qualcomm® Hexagon™ Processor:

```
qpm-cli --extract hexagon8.4 --version 8.4.07
export HEXAGON_ROOT=$HOME/Qualcomm/HEXAGON_Tools
echo $HEXAGON_ROOT
```

Note: Set the environment variable HEXAGON_ROOT to the path where the Hexagon SDK is installed. To change the install path when using `qpm-cli`, see Change the Hexagon tool install path.

Build cDSP

Tools required

- Compiler version: Hexagon 8.4.07
- Python version: Python 3.10.2
- libffi6 package

Build steps

1. Navigate to the following directory:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/CDSP.HT.2.5.c3/cdsp_proc/build/ms
```

2. Clean the build:

```
python ./build_variant.py kodiak.cdsp.prod --clean
```

3. Build the image:

```
python ./build_variant.py kodiak.cdsp.prod
```

Build aDSP

Tools required

- Compiler version: Hexagon 8.4.07
- Python version: Python 3.10.2
- libffi6 package

Nanopb integration (one-time setup)

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
ADSP.HT.5.5.c8/adsp_proc/qsh_api
curl https://jpa.kapsi.fi/nanopb/download/nanopb-0.3.9.5-linux-x86.
tar.gz -o nanopb-0.3.9.5-linux-x86.tar.gz
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
ADSP.HT.5.5.c8/adsp_proc/
python qsh_api/build/config_nanopb_dependency.py -f nanopb-0.3.9.5-
linux-x86
```

Build steps

1. Navigate to the following directory:

   ```
   cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
   qimpsdk/ADSP.HT.5.5.c8/adsp_proc/build/ms
   ```

2. Clean the build:

   ```
   python ./build_variant.py kodiak.adsp.prod --clean
   ```

3. Build the image:

   ```
   python ./build_variant.py kodiak.adsp.prod
   ```

Build Boot

Tools required

- Compiler version: LLVM version must be updated to 14.0.4

Note: To avoid build errors, ensure that there is a / at the end of the command.

```
export LLVM=<FIRMWARE_ROOT>/llvm/14.0.4/
```

- Python version: Python 3.10
- libffi6 package

Build steps

1. Install the device tree compiler:

```
sudo apt-get install device-tree-compiler
export DTC=/usr/bin
```

2. Navigate to the following directory:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/BOOT.MXF.1.0.c1/
```

3. Install the dependencies:

```
python -m pip install -r boot_images/boot_tools/dtschema_tools/
oss/requirements.txt
pip install json-schema-for-humans
```

4. Clean the build:

```
python -u boot_images/boot_tools/buildex.py -t kodiak,
QcomToolsPkg -v LAA -r RELEASE --build_flags=cleanall
```

5. Build the image:

```
python -u boot_images/boot_tools/buildex.py -t kodiak,
QcomToolsPkg -v LAA -r RELEASE
```

Note: For debug variant builds, replace `RELEASE` with `DEBUG`.

Qualcomm TEE firmware

Tools required

- Compiler version: LLVM 16.0.7
- Python version: Python 3.10

Build steps

1. Install LLVM:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/TZ.XF.5.29.1/trustzone_images/build/ms/
vi build_config_deploy_kodiak.xml
# Edit all the occurrences of /pkg/qct/software/llvm/release/
arm/16.0.7/ to <FIRMWARE_ROOT>/llvm/16.0.7/
```

2. Clean the build:

```
python build_all.py -b TZ.XF.5.0 CHIPSET=kodiak --cfg=build_
config_deploy_kodiak.xml --clean
```

3. Build the image:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/TZ.XF.5.29.1/trustzone_images/build/ms/
python build_all.py -b TZ.XF.5.0 CHIPSET=kodiak --cfg=build_
config_deploy_kodiak.xml
```

AOP firmware

Tools required

- Compiler version: LLVM 14.0.4
- Python version: Python 3.10

Build steps

1. Navigate to the following directory:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/AOP.HO.3.6/aop_proc/build/
```

2. Clean the build:

```
./build_kodiak.sh -c -l <FIRMWARE_ROOT>/llvm/14.0.4/
```

3. Build the image:

```
./build_kodiak.sh -l <FIRMWARE_ROOT>/llvm/14.0.4/
```

CPUCP firmware

The CPUCP firmware is released as a binary and build compilation isn't required.

CPUSYS.VM firmware

The CPUSYS.VM firmware is released as a binary and build compilation isn't required.

BTFM firmware

The BTFM firmware is released as a binary and build compilation isn't required.

WLAN firmware

The WLAN firmware is released as a binary and build compilation isn't required.

Generate firmware prebuilds (boot-critical and split-firmware binaries)

Create an integrated firmware binary from the individual components that you compiled:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
QCM6490.LE.1.0/common/build
python build.py --imf
```

Note: Firmware prebuild is successful if the following zip files are generated in the `<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/QCM6490.LE.1.0/common/build/ufs/bin` directory:

- `QCM6490_bootbinaries.zip`
- `QCM6490_dspso.zip`
- `QCM6490_fw.zip`

QCS9075

Prerequisites

- Ensure that the working shell is `bash`.

```
echo $0
```

  The expected output of the command should be `bash`. If not, enter the bash shell:

```
bash
```

- Install the libffi6 package using the following commands. This is required for the QAIC compiler, which generates the header and the source files from the IDL files:

```
curl -LO http://archive.ubuntu.com/ubuntu/pool/main/libf/libffi/
libffi6_3.2.1-8_amd64.deb
sudo dpkg -i libffi6_3.2.1-8_amd64.deb
```

- Install LLVM for AOP, Qualcomm TEE, and boot compilation:

```
cd <FIRMWARE_ROOT>
mkdir llvm

# Log in to qpm-cli and activate the license
qpm-cli --login
qpm-cli --license-activate sdllvm_arm

# LLVM requirement for boot compilation is 14.0.4
qpm-cli --install sdllvm_arm --version 14.0.4 --path <FIRMWARE_
ROOT>/llvm/14.0.4
chmod -R 777 <FIRMWARE_ROOT>/llvm/14.0.4

# LLVM requirement for Qualcomm TEE compilation is 16.0.7
qpm-cli --install sdllvm_arm --version 16.0.7 --path <FIRMWARE_
ROOT>/llvm/16.0.7
chmod -R 777 <FIRMWARE_ROOT>/llvm/16.0.7
```

- Export the `SECTOOLS` variable and compile the firmware builds (`<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk` is the top-level directory):

```
export SECTOOLS=<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_
standard_oem_nm-qimpsdk/QCS9100.LE.1.0/common/sectoolsv2/ext/
Linux/sectools
export SECTOOLS_DIR=<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_
```

```
standard_oem_nm-qimpsdk/QCS9100.LE.1.0/common/sectoolsv2/ext/
Linux
```

- Install and set up Hexagon:

```
qpm-cli --extract hexagon8.6 --version 8.6.05.2
export HEXAGON_ROOT=$HOME/Qualcomm/HEXAGON_Tools
echo $HEXAGON_ROOT
```

Note: Set the environment variable HEXAGON_ROOT to the path where the Hexagon SDK is installed. To change the install path when using `qpm-cli`, see Change the Hexagon tool install path.

Build DSP

Tools required

- Compiler version: Hexagon 8.6.05.2
- Python version: Python 3.8.2

Build steps

1. Install the device tree compiler:

```
sudo apt-get install device-tree-compiler
export DTC_PATH=/usr/bin
```

2. Install the dependencies:

```
pip install ruamel.yaml==0.17.17
pip install dtschema==2021.10
pip install jsonschema==4.0.0
```

3. Navigate to the following directory:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/DSP.AT.1.0/dsp_proc/build/ms
```

4. Clean the build:

```
python ./build_variant.py lemans.adsp.prod --clean
python ./build_variant.py lemans.cdsp0.prod --clean
python ./build_variant.py lemans.cdsp1.prod --clean
python ./build_variant.py lemans.gpdsp0.prod --clean
```

```
python ./build_variant.py lemans.gpdsp1.prod --clean
```

5. Build the image:

```
python ./build_variant.py lemans.adsp.prod && python ./build_
variant.py lemans.cdsp0.prod && python ./build_variant.py
lemans.cdsp1.prod && python ./build_variant.py lemans.gpdsp0.
prod && python ./build_variant.py lemans.gpdsp1.prod
```

Build Boot

Tools required

- Compiler version: LLVM version must be updated to 14.0.4

```
export LLVM=<FIRMWARE_ROOT>/llvm/14.0.4/
```

- Python version: Python 3.10
- libffi6 package

Build steps

1. Install the device tree compiler:

```
sudo apt-get install device-tree-compiler
export DTC=/usr/bin
```

2. Navigate to the following directory:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/BOOT.MXF.1.0.c1/
```

3. Install the dependencies:

```
python -m pip install -r boot_images/boot_tools/dtschema_tools/
oss/requirements.txt
pip install json-schema-for-humans
```

4. Clean the build:

```
python -u boot_images/boot_tools/buildex.py -t lemans,
QcomToolsPkg - v LAA -r RELEASE --build_flags=cleanall
```

5. Build the image:

```
python -u boot_images/boot_tools/buildex.py -t lemans,
QcomToolsPkg - v LAA -r RELEASE
```

Note: For debug variant builds, replace `RELEASE` with `DEBUG`.

Qualcomm TEE firmware

Tools required

- Compiler version: LLVM 16.0.7
- Python version: Python 3.10

Build steps

1. Install LLVM:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/TZ.XF.5.29.1/trustzone_images/build/ms/
vi build_config_deploy_lemans.xml
# Edit all the occurrences of /pkg/qct/software/llvm/release/
arm/16.0.7/ to <FIRMWARE_ROOT>/llvm/16.0.7/
```

2. Clean the build:

```
python build_all.py -b TZ.XF.5.0 CHIPSET=lemans --cfg=build_
config_deploy_lemans.xml --clean
```

3. Build the image:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/TZ.XF.5.29.1/trustzone_images/build/ms/
python build_all.py -b TZ.XF.5.0 CHIPSET=lemans --cfg=build_
config_deploy_lemans.xml
```

AOP firmware

Tools required

- Compiler version: LLVM 14.0.4
- Python version: Python 3.10

Build steps

1. Navigate to the following directory:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/AOP.HO.3.6.1/aop_proc/build/
```

2. Clean the build:

```
./build_lemansau.sh -c -l <FIRMWARE_ROOT>/llvm/14.0.4/
```

3. Build the image:

```
./build_lemansau.sh -l <FIRMWARE_ROOT>/llvm/14.0.4/
```

CPUCP firmware

The CPUCP firmware is released as a binary and build compilation isn't required.

CPUSYS.VM firmware

The CPUSYS.VM firmware is released as a binary and build compilation isn't required.

BTFM firmware

The BTFM firmware is released as a binary and build compilation isn't required.

WLAN firmware

The WLAN firmware is released as a binary and build compilation isn't required.

Generate firmware prebuilds (boot-critical and split-firmware binaries)

Create an integrated firmware binary from the individual components that you compiled:

Note: Apply all the changes from the section *Additional information* in the Release Notes.

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
QCS9100.LE.1.0/common/build
python build.py --imf
```

Note:

Firmware prebuild is successful if the following zip files are generated in the `<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/QCS9100.LE.1.0/common/build/ufs/bin` directory:

- `QCS9100_bootbinaries.zip`

- QCS9100_dspso.zip
- QCS9100_fw.zip

QCS8275

Prerequisites

- Ensure that the working shell is bash.

```
echo $0
```

  The expected output of the command should be bash. If not, enter the bash shell:

```
bash
```

- Install the libffi6 package using the following commands. This is required for the QAIC compiler, which generates the header and the source files from the IDL files:

```
curl -LO http://archive.ubuntu.com/ubuntu/pool/main/libf/libffi/
libffi6_3.2.1-8_amd64.deb
sudo dpkg -i libffi6_3.2.1-8_amd64.deb
```

- Install LLVM for AOP, Qualcomm TEE, and boot compilation:

```
cd <FIRMWARE_ROOT>
mkdir llvm

# Log in to qpm-cli and activate the license
qpm-cli --login
qpm-cli --license-activate sdllvm_arm

# LLVM requirement for boot compilation is 14.0.4
qpm-cli --install sdllvm_arm --version 14.0.4 --path <FIRMWARE_
ROOT>/llvm/14.0.4
chmod -R 777 <FIRMWARE_ROOT>/llvm/14.0.4

# LLVM requirement for Qualcomm TEE compilation is 16.0.7
qpm-cli --install sdllvm_arm --version 16.0.7 --path <FIRMWARE_
ROOT>/llvm/16.0.7
chmod -R 777 <FIRMWARE_ROOT>/llvm/16.0.7
```

- Export the SECTOOLS variable and compile the firmware builds (<FIRMWARE_ ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk is the

top-level directory):

```
export SECTOOLS=<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_
standard_oem_nm-qimpsdk/QCS8300.LE.1.0/common/sectoolsv2/ext/
Linux/sectools
export SECTOOLS_DIR=<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_
standard_oem_nm-qimpsdk/QCS8300.LE.1.0/common/sectoolsv2/ext/
Linux
```

- Install and set up Hexagon:

```
qpm-cli --extract hexagon8.6 --version 8.6.05.2
qpm-cli --extract hexagon8.7 --version 8.7.02.1
export HEXAGON_ROOT=$HOME/Qualcomm/HEXAGON_Tools
echo $HEXAGON_ROOT
```

Note: Set the environment variable HEXAGON_ROOT to the path where the Hexagon SDK is installed. To change the install path when using `qpm-cli`, see Change the Hexagon tool install path.

Build DSP

Tools required

- Compiler version: Hexagon 8.6.05.2 and 8.7.02.1
- Python version: Python 3.8.2

Build steps

1. Install the device tree compiler:

```
sudo apt-get install device-tree-compiler
export DTC_PATH=/usr/bin
```

2. Install the dependencies:

```
pip install ruamel.yaml==0.17.17
pip install dtschema==2021.10
pip install jsonschema==4.0.0
```

3. Navigate to the following directory:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/DSP.AT.1.0/dsp_proc/build/ms
```

4. Clean the build:

```
python ./build_variant.py lemans.adsp.prod --clean
python ./build_variant.py monaco.cdsp0.prod --clean
python ./build_variant.py lemans.gpdsp0.prod --clean
```

5. Build the image:

```
python ./build_variant.py lemans.adsp.prod && python ./build_
variant.py monaco.cdsp0.prod && python ./build_variant.py
lemans.gpdsp0.prod
```

Build Boot

Tools required

- Compiler version: LLVM version must be updated to 14.0.4

```
export LLVM=<FIRMWARE_ROOT>/llvm/14.0.4/
```

- Python version: Python 3.10
- libffi6 package

Build steps

1. Install the device tree compiler:

```
sudo apt-get install device-tree-compiler
export DTC=/usr/bin
```

2. Navigate to the following directory:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/BOOT.MXF.1.0.c1/
```

3. Install the dependencies:

```
python -m pip install -r boot_images/boot_tools/dtschema_tools/
oss/requirements.txt
pip install json-schema-for-humans
```

4. Clean the build:

```
python -u boot_images/boot_tools/buildex.py -t monaco,
QcomToolsPkg - v LAA -r RELEASE --build_flags=cleanall
```

5. Build the image:

```
python -u boot_images/boot_tools/buildex.py -t monaco,
QcomToolsPkg - v LAA -r RELEASE
```

Note: For debug variant builds, replace `RELEASE` with `DEBUG`.

Qualcomm TEE firmware

Tools required

- Compiler version: LLVM 16.0.7
- Python version: Python 3.10

Build steps

1. Install LLVM:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/TZ.XF.5.29.1/trustzone_images/build/ms/
vi build_config_deploy_monaco.xml
# Edit all the occurrences of /pkg/qct/software/llvm/release/
arm/16.0.7/ to <FIRMWARE_ROOT>/llvm/16.0.7/
```

2. Clean the build:

```
python build_all.py -b TZ.XF.5.0 CHIPSET=monaco --cfg=build_
config_deploy_monaco.xml --clean
```

3. Build the image:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/TZ.XF.5.29.1/trustzone_images/build/ms/
python build_all.py -b TZ.XF.5.0 CHIPSET=monaco --cfg=build_
config_deploy_monaco.xml
```

AOP firmware

Tools required

- Compiler version: LLVM 14.0.4
- Python version: Python 3.10

Build steps

1. Navigate to the following directory:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-
qimpsdk/AOP.HO.3.6.1/aop_proc/build/
```

2. Clean the build:

```
./build_monaco.sh -c -l <FIRMWARE_ROOT>/llvm/14.0.4/
```

3. Build the image:

```
./build_monaco.sh -l <FIRMWARE_ROOT>/llvm/14.0.4/
```

CPUCP firmware

The CPUCP firmware is released as a binary and build compilation isn't required.

CPUSYS.VM firmware

The CPUSYS.VM firmware is released as a binary and build compilation isn't required.

BTFM firmware

The BTFM firmware is released as a binary and build compilation isn't required.

WLAN firmware

The WLAN firmware is released as a binary and build compilation isn't required.

Generate firmware prebuilds (boot-critical and split-firmware binaries)

Create an integrated firmware binary from the individual components that you compiled:

```
cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
QCS8300.LE.1.0/common/build
python build.py --imf
```

Note: Firmware prebuild is successful if the following zip files are generated in the `<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/QCS8300.LE.1.0/common/build/ufs/bin` directory:

- `QCS8300_bootbinaries.zip`
- `QCS8300_dspso.zip`
- `QCS8300_fw.zip`

Build BSP image with extras

BSP image build has the software components for Qualcomm device support and value-added software features applicable to Qualcomm SoCs. It includes a reference distribution configuration for Qualcomm development kits. The `meta-qcom-extras` layer enables source compilation of select components, which are otherwise present as binary.

For more details, see Qualcomm Linux metadata layers.

1. Download Qualcomm Yocto and the supporting layers with extras:

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
1.xml
repo sync
git clone https://qpm-git.qualcomm.com/home2/git/qualcomm/
qualcomm-linux-spf-1-0_hlos_oem_metadata.git -b <meta-qcom-
extras release tag> --depth 1
# Example, <meta-qcom-extras release tag> is r1.0_00077.0
mkdir -p layers/meta-qcom-extras
cp -rf qualcomm-linux-spf-1-0_hlos_oem_metadata/<product>/
common/config/meta-qcom-extras/* layers/meta-qcom-extras/
# An example <product> is QCM6490.LE.1.0. For more information
on <product>, see the latest Release Notes (https://docs.
qualcomm.com/bundle/publicresource/topics/RNO-250403001134/).
```

Note: For the `<manifest release tag>` and `<meta-qcom-extras release tag>` information, see the section *Build-critical release tags* in the Release Notes.

2. Set up the Yocto build:

```
# Export additional meta layers to EXTRALAYERS. Location is
assumed under <WORKSPACE DIR>/layers.
export EXTRALAYERS="meta-qcom-extras"

# CUST_ID is used to clone the proprietary source repositories
downloaded by meta-qcom-extras.
# It allows source compilation for the corresponding binaries
present in meta-qcom-hwe.
```

```
# CUST_ID must be set to "213195" for no-modem based
distributions ("qualcomm-linux-spf-1-0_ap_standard_oem_nomodem",
# "qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk",
"qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk").
# For other modem based distributions, CUST_ID must be set based
on the "Customer ID".
# To find "Customer ID", sign in to your account at qualcomm.
com.
# Click the Profile icon, select Account Settings, and then
scroll down to the Company Information section.
# export CUST_ID using the following command.
export CUST_ID=<Customer ID>

# The firmware recipe is compiled when the Yocto build is
initiated. Firmware recipe expects the
# path of firmware. You have generated firmware prebuilts (boot-
critical and split-firmware binaries)
# using the steps described in the previous section.
# Example, for QCM6490, the directory path must contain QCM6490_
bootbinaries.zip, QCM6490_dspso.zip, and QCM6490_fw.zip.
# Set the environment variable to pick up the prebuilts:
export FWZIP_PATH="<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_
standard_oem_nm-qimpsdk/<product>/common/build/ufs/bin"
# An example <product> is QCM6490.LE.1.0. For more information
on <product>, see the latest Release Notes (https://docs.
qualcomm.com/bundle/publicresource/topics/RNO-250403001134/).
```

3. Set up the build environment:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

Note: To know the `MACHINE` parameter values, see Release Notes.

4. Compile the Yocto build:

```
bitbake qcom-multimedia-image
```

Note: Clean the Yocto build:

```
bitbake -fc cleansstate qcom-multimedia-image
bitbake -fc cleanall qcom-multimedia-image
```

After a successful build, check that the `system.img` file is in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-multimedia-image
ls -al system.img
```

5. Flash the generated build using Flash software images.

Build QIMP SDK image with extras

The QIMP SDK is a collection of four standalone function SDKs, namely, Qualcomm IM SDK, Qualcomm Neural Processing SDK, Qualcomm AI Engine direct SDK, and LiteRT. It also includes reference applications that you can use to develop use cases. The `meta-qcom-extras` layer enables source compilation of select components, which are otherwise present as binary.

For more details, see QIMP SDK Quick Start Guide.

1. Download the QIMP SDK layers, Qualcomm Yocto layer, and the supporting layers with extras:

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
1.xml
repo sync
git clone https://qpm-git.qualcomm.com/home2/git/qualcomm/
qualcomm-linux-spf-1-0_hlos_oem_metadata.git -b <meta-qcom-
extras release tag> --depth 1
# Example, <meta-qcom-extras release tag> is r1.0_0005.0
mkdir -p layers/meta-qcom-extras
cp -rf qualcomm-linux-spf-1-0_hlos_oem_metadata/<product>/
common/config/meta-qcom-extras/* layers/meta-qcom-extras/
# An example <product> is QCM6490.LE.1.0. For more information
```

```
on <product>, see the latest Release Notes (https://docs.
qualcomm.com/bundle/publicresource/topics/RNO-250403001134/).
git clone https://github.com/quic-yocto/meta-qcom-qim-product-
sdk -b qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2 layers/
meta-qcom-qim-product-sdk
```

Note: For the `<manifest release tag>` and `<meta-qcom-extras release tag>` information, see the section *Build-critical release tags* in the Release Notes.

2. Set up the Yocto build:

```
# Export additional meta layers to EXTRALAYERS. Location is
assumed under <WORKSPACE DIR>/layers.
export EXTRALAYERS="meta-qcom-extras meta-qcom-qim-product-sdk"

# CUST_ID is used to clone the proprietary source repositories
downloaded by meta-qcom-extras.
# It allows source compilation for the corresponding binaries
present in meta-qcom-hwe.
# CUST_ID must be set to "213195" for no-modem based
distributions ("qualcomm-linux-spf-1-0_ap_standard_oem_nomodem",
# "qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk",
"qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk").
# For other modem based distributions, CUST_ID must be set based
on the "Customer ID".
# To find "Customer ID", sign in to your account at qualcomm.
com.
# Click the Profile icon, select Account Settings, and then
scroll down to the Company Information section.
# export CUST_ID using the following command.
export CUST_ID=<Customer ID>

# The firmware recipe is compiled when the Yocto build is
initiated. Firmware recipe expects the
# path of firmware. You have generated firmware prebuilts (boot-
critical and split-firmware binaries)
# using the steps described in the previous section.
# Example, for QCM6490, the directory path must contain QCM6490_
bootbinaries.zip, QCM6490_dspso.zip, and QCM6490_fw.zip.
# Set the environment variable to pick up the prebuilts:
export FWZIP_PATH="<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_
standard_oem_nm-qimpsdk/<product>/common/build/ufs/bin"
# An example <product> is QCM6490.LE.1.0. For more information
```

```
on <product>, see the latest Release Notes (https://docs.
qualcomm.com/bundle/publicresource/topics/RNO-250403001134/).
```

3. Set up the build environment:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

Note: To know the `MACHINE` parameter values, see Release Notes.

4. Compile the QIMP SDK build:

```
bitbake qcom-multimedia-image
# Build SDK image
bitbake qcom-qim-product-sdk
```

Note: Clean the QIMP SDK build:

```
bitbake -fc cleansstate qcom-multimedia-image
bitbake -fc cleanall qcom-multimedia-image

bitbake -fc cleansstate qcom-qim-product-sdk
bitbake -fc cleanall qcom-qim-product-sdk
```

After a successful build, check that the `system.img` file is in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-multimedia-image
ls -al system.img
```

5. Flash the generated build using Flash software images.

Build QIRP SDK image with extras

The QIRP SDK 2.0 is a collection of components that enable you to develop robotic features on Qualcomm Linux releases. The `meta-qcom-extras` layer enables source compilation of select components, which are otherwise available as binaries.

For more details, see QIRP SDK 2.0 User Guide.

Note: Before you begin, clone the respective firmware for QIRP SDK, for example, `qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk`.

1. Download QIRP SDK layers, Qualcomm Yocto, and supporting layers with extras:

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.
1.xml
repo sync
git clone https://qpm-git.qualcomm.com/home2/git/qualcomm/
qualcomm-linux-spf-1-0_hlos_oem_metadata.git -b <meta-qcom-
extras release tag> --depth 1
# Example, <meta-qcom-extras release tag> is r1.0_00077.0
mkdir -p layers/meta-qcom-extras
mkdir -p layers/meta-qcom-robotics-extras
cp -rf qualcomm-linux-spf-1-0_hlos_oem_metadata/<product>/
common/config/meta-qcom-extras/* layers/meta-qcom-extras/
cp -rf qualcomm-linux-spf-1-0_hlos_oem_metadata/<product>/
common/config/meta-qcom-robotics-extras/* layers/meta-qcom-
robotics-extras/
# An example <product> is QCM6490.LE.1.0. For more information
on <product>, see the latest Release Notes (https://docs.
qualcomm.com/bundle/publicresource/topics/RNO-250403001134/).

git clone https://github.com/ros/meta-ros -b scarthgap layers/
meta-ros && cd layers/meta-ros && git checkout
c560699e810e60a9526f4226c2c23f8d877280c8 && cd ../../
git clone https://github.com/quic-yocto/meta-qcom-robotics.git -
b qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-1.0 layers/
meta-qcom-robotics
git clone https://github.com/quic-yocto/meta-qcom-robotics-
```

```
distro.git -b qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-
1.0 layers/meta-qcom-robotics-distro
git clone https://github.com/quic-yocto/meta-qcom-robotics-sdk.
git -b qcom-6.6.65-QLI.1.4-Ver.1.0_robotics-product-sdk-1.0
layers/meta-qcom-robotics-sdk
git clone https://github.com/quic-yocto/meta-qcom-qim-product-
sdk -b qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2 layers/
meta-qcom-qim-product-sdk
# Example, <meta-qcom-qim-product-sdk release tag> is qcom-6.6.
65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2
```

Note: For the <manifest release tag>, <meta-qcom-extras release tag>, and <meta-qcom-qim-product-sdk release tag> information, see the section *Build-critical release tags* in the Release Notes.

2. Set up the Yocto build:

```
# Export additional meta layers to EXTRALAYERS. Location is
assumed under <WORKSPACE DIR>/layers.
export EXTRALAYERS="meta-qcom-extras meta-qcom-robotics-extras"

# CUST_ID is used to clone the proprietary source repositories
downloaded by meta-qcom-extras.
# It allows source compilation for the corresponding binaries
present in meta-qcom-hwe.
# CUST_ID must be set to "213195" for no-modem based
distributions ("qualcomm-linux-spf-1-0_ap_standard_oem_nomodem",
# "qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk",
"qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk").
# For other modem based distributions, CUST_ID must be set based
on the "Customer ID".
# To find "Customer ID", sign in to your account at qualcomm.
com.
# Click the Profile icon, select Account Settings, and then
scroll down to the Company Information section.
# export CUST_ID using the following command.
export CUST_ID=<Customer ID>

# The firmware recipe is compiled when the Yocto build is
initiated. Firmware recipe expects the
# path of firmware. You have generated firmware prebuilts (boot-
critical and split-firmware binaries)
# using the steps described in the previous section.
```

```
# Example, for QCM6490, the directory path must contain QCM6490_
bootbinaries.zip, QCM6490_dspso.zip, and QCM6490_fw.zip.
# Set the environment variable to pick up the prebuilts:
export FWZIP_PATH="<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_
standard_oem_nm-qirpsdk/<product>/common/build/ufs/bin"
# An example <product> is QCM6490.LE.1.0. For more information
on <product>, see the latest Release Notes (https://docs.
qualcomm.com/bundle/publicresource/topics/RNO-250403001134/).
```

3. Compile the QIRP SDK build:

```
ln -s layers/meta-qcom-robotics-distro/set_bb_env.sh ./setup-
robotics-environment
ln -s layers/meta-qcom-robotics-sdk/scripts/qirp-build ./qirp-
build
MACHINE=<machine> DISTRO=qcom-robotics-ros2-humble QCOM_
SELECTED_BSP=custom source setup-robotics-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
robotics-ros2-humble QCOM_SELECTED_BSP=custom source setup-
robotics-environment
# source setup-robotics-environment: Sets the environment,
creates the build directory build-qcom-robotics-ros2-humble,
# and enters into build-qcom-robotics-ros2-humble directory.
../qirp-build qcom-robotics-full-image
```

Note: To know the `MACHINE` parameter values, see Release Notes.

After a successful build, check that the QIRP SDK build artifacts are at the following paths:

```
QIRP SDK artifacts: <workspace_path>/build-qcom-robotics-ros2-
humble/tmp-glibc/deploy/qirpsdk_artifacts/qirp-sdk_<version>.
tar.gz
# system.img is present in the following path
Robotics image: <workspace_path>/build-qcom-robotics-ros2-
humble/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-
robotics-full-image
```

4. Flash the generated build using Flash software images.

7 Flash software images

Follow these steps to flash the software images:

1. Update the `udev` rules (one-time prerequisite).
2. Move the device to Emergency Download (EDL) mode.
3. Provision UFS (one-time prerequisite).
4. Flash SAIL (one-time prerequisite).
5. Flash CDT.
6. Flash the software:
   - Using QDL
   - Using PCAT

7.1 Update `udev` rules

Configure the `udev` USB rules for the Qualcomm manufacturing vendor ID **05c6** on the Linux host:

1. Go to the `udev` USB rules directory:

   ```
   cd /etc/udev/rules.d
   ```

2. List the contents of the directory:

   ```
   ls
   ```

   - If the `51-qcom-usb.rules` file isn't present, use the `sudo vi 51-qcom-usb.rules` file to create it and add the following content to the file:

     ```
     SUBSYSTEMS=="usb", ATTRS{idVendor}=="05c6", ATTRS{idProduct}
     =="9008", MODE="0666", GROUP="plugdev"
     ```

   - If the file exists, then check for the earlier content:

```
cat 51-qcom-usb.rules
```

3. Restart `udev`:

```
sudo systemctl restart udev
```

If the USB cable is already connected to the host, unplug and reconnect the cable for the updated rules to take effect.

7.2 Move to EDL mode

The device must be in the EDL mode before you flash the software. The Qualcomm supported device by default enters the EDL mode if there is no image on the device after power up or if it's corrupted. To force the device into the EDL mode, use any one of the following methods.

Using UART

Note: Use UART only if the device has a preloaded build.

1. Connect the device to a UART shell.
2. Move the device to the EDL mode by running the following command on the UART shell:

```
reboot edl
```

3. Verify if the device has entered the Qualcomm Download mode (QDL mode) by running the following command on the host computer:

```
lsusb
```

Sample output

```
Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem (QDL mode)
```

Note: This procedure applies to the Ubuntu host environment.

Using ADB

Note: Use ADB only if the device has a preloaded build.

1. Install QUD on the host device.
2. Install ADB on the host device.
3. Connect ADB to the device.
4. Move the device to EDL mode by running the following command on the host computer:

```
adb shell reboot edl
```

5. Verify whether the device has entered the QDL mode by running the following command on the host computer:

```
lsusb
```

Sample output

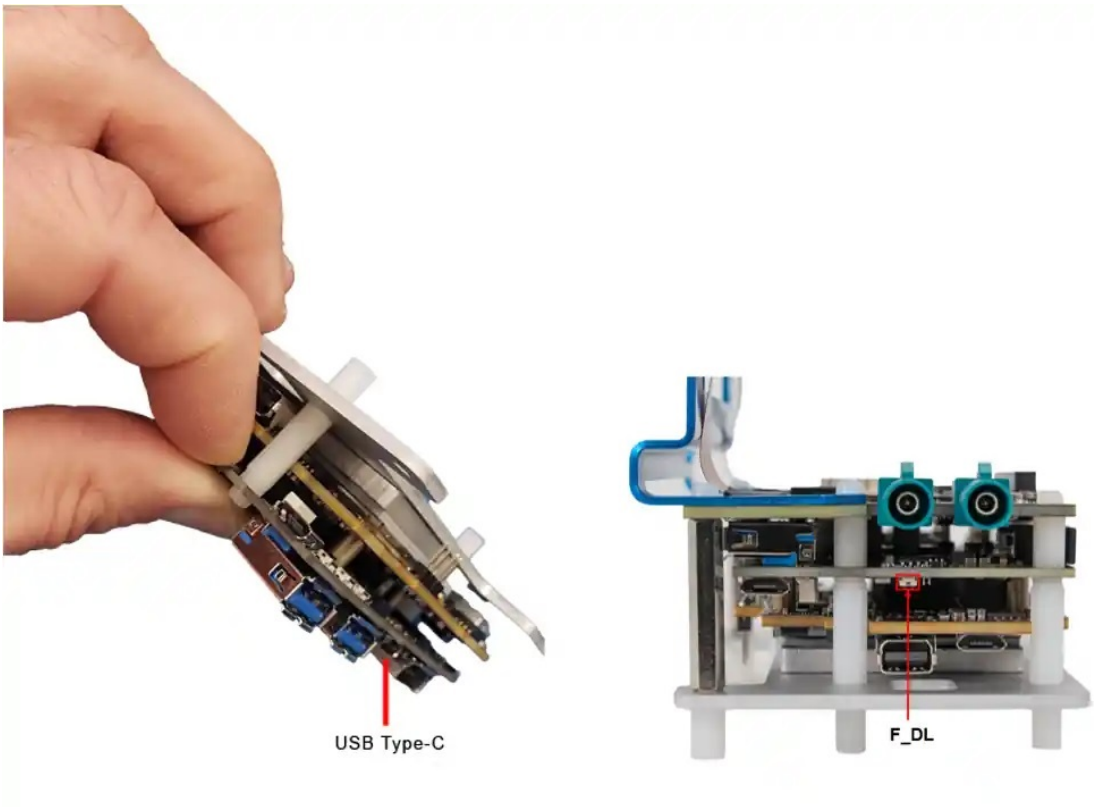
```
Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem (QDL
mode)
```

Note: This procedure applies to the Ubuntu host environment.

Manual

QCS6490/QCS5430

1. Press and hold the **F_DL** button.



2. Connect the device to a +12 V wall power supply.
3. Connect the device to the host system using a Type-C cable through the USB Type-C connector.
4. Release the **F_DL** button. The device should now be in the Qualcomm Download mode (QDL mode). For this task, QDL is used interchangeably with EDL.
5. Verify whether the device has entered the QDL mode:

```
lsusb
```

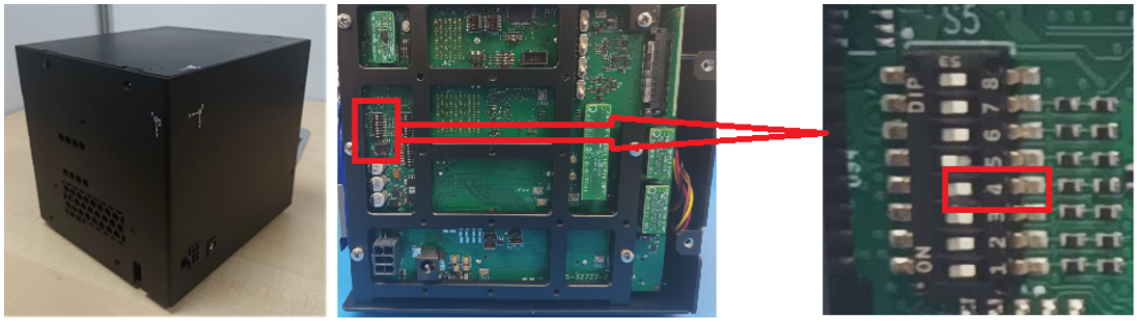
Sample output

```
Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem (QDL mode)
```

QCS9075

Qualcomm® IQ-9 Beta Evaluation Kit (EVK)

1. Switch on the dip switch S5-4 to put the device in the EDL mode.



2. Verify whether the device has entered the QDL mode:

```
lsusb
```

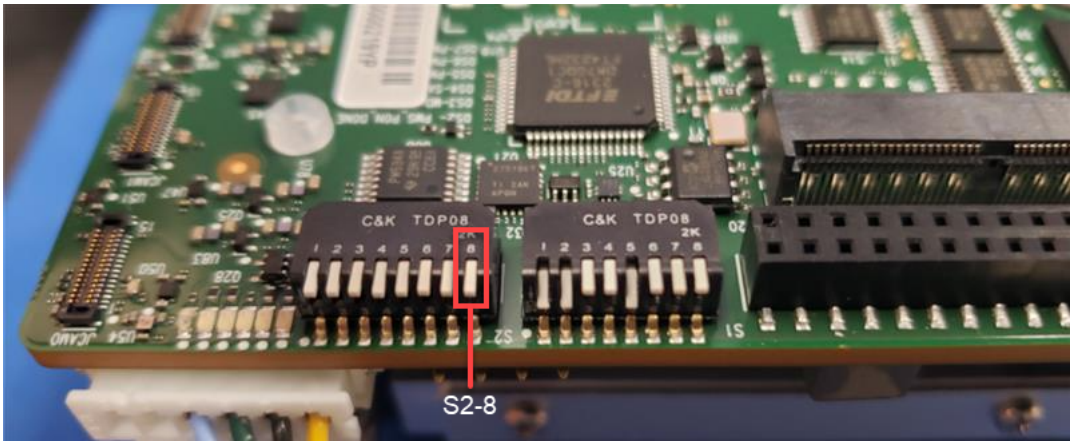
Sample output

```
Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem (QDL mode)
```

Note: Dip switch S5-4 must be turned off after the flashing is complete.

Qualcomm Dragonwing™ IQ-9075 EVK

1. Switch on the dip switch S2-8 to put the device in the EDL mode.



2. Verify whether the device has entered the QDL mode:

```
lsusb
```

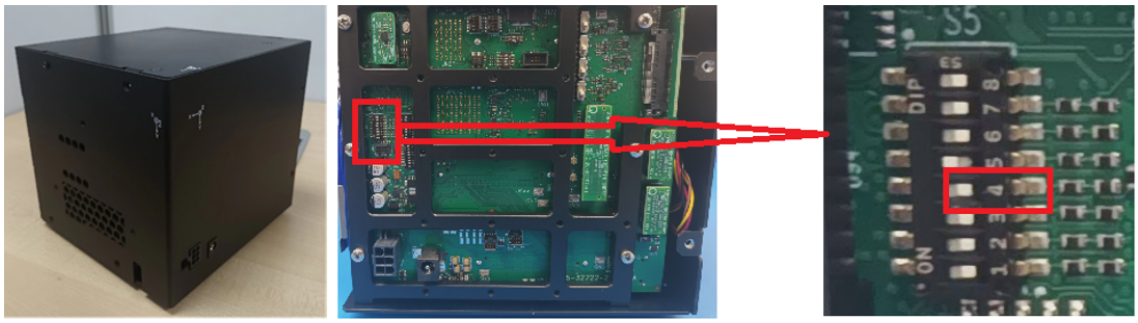
Sample output

```
Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem (QDL
mode)
```

Note: Dip switch S2-8 must be turned off after the flashing is complete.

QCS8275

1. Switch on the dip switch S5-4 to put the device in the EDL mode.



2. Verify whether the device has entered the QDL mode:

```
lsusb
```

Sample output

```
Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem (QDL
mode)
```

Note: Dip switch S5-4 must be turned off after the flashing is complete.

7.3 Provision UFS

Universal Flash Storage (UFS) provisioning helps to divide the storage into many LUNs, which stores different types of data separately. This improves access efficiency and system organization.

Note:

- UFS is provisioned by default. If there are any changes in LUNs, UFS must be re-provisioned. To download the provision XML file and to check the applicability of UFS provisioning for different SoCs, see the table *UFS Provision* in Release Specific Information.

1. Download the provision file.

   Based on the required SoC, download the respective 'provision' from the *UFS Provision* table of the Release Notes.

   ```
   wget <provision_download_link>
   unzip <downloaded_zip_file>
   ```

   Example:

   ```
   mkdir <provision_download_path>
   cd <provision_download_path>
   wget https://artifacts.codelinaro.org/artifactory/codelinaro-le/
   Qualcomm_Linux/QCS6490/provision.zip
   unzip provision.zip
   ```

2. Download the QDL tool.

   Qualcomm Device Loader (QDL) is a software tool that communicates with the Qualcomm USB devices to upload a flash loader and flash software images.

   Acquire the latest version of the QDL tool using one of the following methods:

   - Download the tool from https://softwarecenter.qualcomm.com/#/catalog/item/Qualcomm_Device_Loader and unzip the contents of the downloaded folder.
   - Run the following command to download using CLI:

     ```
     wget https://softwarecenter.qualcomm.com/api/download/
     software/tools/Qualcomm_Device_Loader/Windows/Latest.zip
     ```

   Run the following command to provide executable permission to QDL:

   ```
   chmod -R 777 <qdl_download_path>
   ```

3. Provision UFS:

   ```
   cd <provision_download_path>
   <qdl_download_path>/qdl_<version>/QDL_Linux_x64/qdl --storage
   ufs prog_firehose_ddr.elf <Provision file>
   ```

```
# Example, <qdl_download_path>/qdl_<version>/QDL_Linux_x64/qdl --storage ufs prog_firehose_ddr.elf provision_1_3.xml
```

Note: Use QDL binary based on the host computer architecture. For example, linux_x64 supported qdl binary is `qdl_<version>/QDL_Linux_x64/qdl`.

7.4 Flash SAIL

Note: Safety Island (SAIL) is applicable only for QCS9075 and QCS8275.

QCS9075

1. Download the QDL tool.

   QDL is a software tool that communicates with the Qualcomm USB devices to upload a flash loader and flash software images.

   Acquire the latest version of the QDL tool using one of the following methods:

   - Download the tool from https://softwarecenter.qualcomm.com/#/catalog/item/Qualcomm_Device_Loader and unzip the contents of the downloaded folder.
   - Run the following command to download using CLI:

     ```
     wget https://softwarecenter.qualcomm.com/api/download/software/tools/Qualcomm_Device_Loader/Windows/Latest.zip
     ```

2. Flash the SAIL.

   ```
   # SAIL image is under <workspace_path>/build-<DISTRO>/tmp-glibc/deploy/images/<MACHINE>/<IMAGE>/sail_nor
   # build_path: For DISTRO=qcom-wayland, it's build-qcom-wayland.
   #             For DISTRO=qcom-robotics-ros2-humble, it's build-qcom-robotics-ros2-humble
   # qdl --storage spinor <prog.mbn> [<program> <patch> ...]
   # Example: build_path is build-qcom-wayland
   cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs9075-rb8-core-kit/qcom-multimedia-image/sail_nor
   <qdl_download_path>/qdl_<version>/QDL_Linux_x64/qdl --storage
   ```

```
spinor prog_firehose_ddr.elf rawprogram0.xml patch0.xml
```

QCS8275

Note: This procedure is available for registered users only.

For QCS8275 SAIL flashing steps, see Section 4.4 (Flash SAIL) from the Qualcomm IQ-8 Beta Evaluation Kit Quick Start Guide.

7.5 Flash CDT

CDT provides platform/device-dependent data such as platform ID, subtype, version. Various Software (drivers/firmware) modules can use this information to perform dynamic detection and initialization of the platform. You can update CDT by flashing a CDT binary:

1. Download the CDT binary.

   Based on the required reference kit, download the respective CDT from the *CDT* table of the Release Notes.

   ```
   wget <CDT_download_link>
   unzip <downloaded_zip_file>
   ```

   Example:

   ```
   mkdir <cdt_download_path>
   cd <cdt_download_path>
   wget https://artifacts.codelinaro.org/artifactory/codelinaro-le/
   Qualcomm_Linux/QCS6490/cdt/rb3gen2-core-kit.zip
   unzip rb3gen2-core-kit.zip
   ```

2. Download the QDL tool.

   QDL is a software tool that communicates with the Qualcomm USB devices to upload a flash loader and flash software images.

   Acquire the latest version of the QDL tool using one of the following methods:

   - Download the tool from https://softwarecenter.qualcomm.com/#/catalog/item/Qualcomm_Device_Loader and unzip the contents of the downloaded folder.
   - Run the following command to download using CLI:

```
wget https://softwarecenter.qualcomm.com/api/download/
software/tools/Qualcomm_Device_Loader/Windows/Latest.zip
```

3. Flash the CDT:

```
cd <cdt_download_path>
<qdl_download_path>/qdl_<version>/QDL_Linux_x64/qdl prog_
firehose_ddr.elf rawprogram3.xml patch3.xml
```

Note: Use QDL binary based on the host computer architecture. For example, linux_x64 supported qdl binary is `qdl_<version>/QDL_Linux_x64/qdl`.

7.6 Flash software using QDL

1. Ensure that the ModemManager tool isn't running.

   Some Linux distributions include the ModemManager tool, which allows you to configure the mobile broadband. When you connect the device in the USB mode, it's identified as a Qualcomm modem, and the ModemManager tries to configure the device. As this interferes with the QDL flashing, you must disable the ModemManager before connecting your device.

   If you are using a Linux distribution with `systemd`, stop the ModemManager tool using the following command:

   ```
   sudo systemctl stop ModemManager
   ```

   If you need the ModemManager, you can restart it after the flashing is complete.

2. Download the QDL tool.

   QDL is a software tool that communicates with Qualcomm USB devices to upload a flash loader and flash software images.

   Acquire the latest version of the QDL tool using one of the following methods:

   - Download the tool from https://softwarecenter.qualcomm.com/#/catalog/item/Qualcomm_Device_Loader and unzip the contents of the downloaded folder.
   - Run the following command to download using CLI:

     ```
     wget https://softwarecenter.qualcomm.com/api/download/
     software/tools/Qualcomm_Device_Loader/Windows/Latest.zip
     ```

3. Flash the images:

```
# Built images are under <workspace_path>/build-<DISTRO>/tmp-
glibc/deploy/images/<MACHINE>/<IMAGE>
# build_path: For DISTRO=qcom-wayland, it's build-qcom-wayland.
#             For DISTRO=qcom-robotics-ros2-humble, it's build-
qcom-robotics-ros2-humble
# qdl <prog.mbn> [<program> <patch> ...]
# Example: build_path is build-qcom-wayland
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-multimedia-image
<qdl_download_path>/qdl_<version>/QDL_Linux_x64/qdl prog_
firehose_ddr.elf rawprogram*.xml patch*.xml
```

Note: Use QDL binary based on the host computer architecture. For example, linux_x64 supported qdl binary is `qdl_<version>/QDL_Linux_x64/qdl`.

Flashing is successful if you see *partition 1 is now bootable* on the terminal window as shown in the following message:

```
LOG: INFO: Calling handler for setbootablestoragedrive
LOG: INFO: Using scheme of value = 1
partition 1 is now bootable
LOG: INFO: Calling handler for power
LOG: INFO: Will issue reset/power off 100 useconds, if this hangs check if watchdog is
enabled
LOG: INFO: bsp_target_reset() 1
```

After a successful flashing operation, run the `lsusb` command to see the device information on the terminal window as shown in line 4 of the following message:

```
# Sample output for QCS6490
Bus 002 Device 003: ID 05c6:9135 Qualcomm, Inc. qcs6490-rb3gen2-vision-kit
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

To verify the updated software version, see Verify the Qualcomm Linux version.

Note: If flashing fails, perform the following steps and retry the flashing procedure:

1. Power off the device.
2. Disconnect from the host.

3. Restart the host.

To establish UART and network connections, see Connect to UART shell and network.

7.7 Flash software using PCAT

Note: This procedure is available for registered users only.

1. Install QSC CLI.
2. To detect the connected devices and flash the software builds, install the Qualcomm PCAT and QUD tools on the host computer. Run the following commands to use `qpm-cli` to install PCAT and QUD:

```
qpm-cli --login
qpm-cli --install quts --activate-default-license
qpm-cli --install qud --activate-default-license
qpm-cli --install pcat --activate-default-license
```

Note: For Ubuntu 22.04, you may see an issue while installing QUD, where you must enroll the public key on your Linux host for a successful QUD installation. For more details, follow the steps provided in the README file available in the `/opt/QTI/sign/signReadme.txt` directory.

3. Check if the `QTI_HS-USB_QDLoader` driver is available in the installed directory:

```
ls -la /dev/Q*
```

Sample output

```
crw-rw-rw- 1 root 242 0 Dec 10 10:51 /dev/QTI_HS-USB_QDLoader_9008_3-8:1.0
```

4. Verify if the device entered the QDL mode:

```
lsusb
```

Sample output

```
Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem (QDL mode)
```

5. Check if PCAT recognizes the device:

```
pcat -devices
```

Sample output

```
Searching devices in Device Manager, please wait for a moment...
ID | DEVICE TYPE | DEVICE STATE | SERIAL NUMBER | ADB SERIAL
NUMBER | DESCRIPTION
NA | NA          | EDL          | BE116704      | be116704
   | Qualcomm USB Composite Device:QUSB_BULK_CID:042F_SN:
BE116704
```

6. Flash the build:

```
PCAT -PLUGIN SD -DEVICE <device_serial_number> -BUILD "<build_
images_path>" -MEMORYTYPE UFS -FLAVOR asic

# Example, PCAT -PLUGIN SD -DEVICE be116704 -BUILD "<workspace_
path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-
rb3gen2-vision-kit/qcom-multimedia-image" -MEMORYTYPE UFS -
FLAVOR asic
```

If flashing the software is successful, you will see the following message:

```
xxxx INFO] [ FIRMWARE DOWNLOAD LOG ] Process Finished
xxxx INFO] Status - TRUE
xxxx INFO] Response - Downloading software images completed on the device
Qualcomm USB Composite Device:QUSB_BULK_CID:042F_SN:BE116704
```

After a successful flashing operation, run the `lsusb` command to see the device information on the terminal window as shown in line 4 of the following message:

```
# Sample output for QCS6490
Bus 002 Device 003: ID 05c6:9135 Qualcomm, Inc. qcs6490-rb3gen2-vision-kit
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

The device reboots after the flashing procedure completes successfully. To verify the updated software version, see Verify the Qualcomm Linux version.

7.8 Connect to UART shell and network

After flashing and booting the device, follow these steps to establish the connections:

- Connect to UART shell
- Connect to network
- Sign in using SSH

8 Troubleshoot

8.1 Docker

- **docker: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?**

  Run the following command to start Docker:

```
sudo systemctl start docker
```

- **Error response from daemon: Get "https://registry-1.docker.io/v2/": http: server gave HTTP response to HTTPS client**

  Add an internal Docker registry mirror (internal setting for Qualcomm network).

  Note: Don't include # comments in the JSON configuration file. Using a tab instead of space and other invisible whitespace characters may break the functionality of JSON configuration files and can also lead to `docker.service` failing to start.

```
sudo vim /etc/docker/daemon.json
# Add an entry similar to the following in /etc/docker/daemon.
json:
{
   "registry-mirrors": ["https://docker-registry.qualcomm.com"]
}
```

  Note: As an alternative, you can add the following entry in `/etc/docker/daemon.json`:

```
"registry-mirrors": ["https://ccr.ccs.tencentyun.com"]
```

  Restart the Docker service to take the new settings.

```
sudo systemctl restart docker
```

- **Failed to download from https://download.docker.com**

Note: If you are unable to access or download from https://download.docker.com, run the following commands to install docker:

```
sudo apt update
sudo apt install apt-transport-https ca-certificates curl gnupg
lsb-release
curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg
| sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] http://mirrors.aliyun.
com/docker-ce/linux/ubuntu $(lsb_release -cs) stable"
```

- **Docker failure due to Virtualization not enabled**

  Enable virtualization from the BIOS to resolve this error. Follow the specific instructions from the system provider to enable the virtualization. For example, the following steps can enable virtualization provided by a system provider:

  1. When the system is booting up, step into the BIOS. The *BIOS* window appears.
  2. Switch to the *Advanced* tab.
  3. In the *CPU Configuration* section, set *Virtualization Technology* to enabled.
  4. Save and exit.
  5. Restart the system.

- **Permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock**

  This happens when `qsc-cli` is already installed on the machine and you aren't part of the Docker group:

  ```
  sudo groupadd docker
  sudo usermod -aG docker $USER
  newgrp docker
  ```

  To confirm that you are part of the Docker group, run the following command:

  ```
  sudo grep /etc/group -e "docker"
  # This command shows a list of users who are part of the Docker
  group; must include your user ID
  ```

  Sign out and sign in again for the access to take effect.

  ```
  # You can run the following command to check if you are part of
  the Docker group
  ```

```
id -a
# This command returns an output string which should include
'docker'
```

8.2 Sync

- **repo init or sync failure with except ManifestInvalidRevisionError, e:**

  You may see this issue after installing the Repo package:

  – If you have redirection in your `/etc/gitconfig` or `~/.gitconfig` to an internal mirror.

  – If your internal mirror has a prefix to branches while mirroring. For example, if `/etc/gitconfig` redirects and the internal mirror has a stable branch from upstream git mirrored as `aosp/stable`, then the following error appears while performing `repo init`:

```
[url "ssh://<internal mirror>:<port>/tools/repo"]
insteadOf = https://android.googlesource.com/tools/repo
insteadOf = https://gerrit.googlesource.com/git-repo
```

```
File "/local/mnt/workspace/<userid>/test_repo/.repo/repo/main.py
", line 126
   except ManifestInvalidRevisionError, e:
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
SyntaxError: multiple exception types must be parenthesized
```

  The steps to resolve this error are as follows:

```
# Remove the older .repo folder. This will be in the directory
where you ran 'repo init' command earlier
rm -rf .repo

# Export and run the repo commands to fix the repo issues. The
REPO_REV must point to the mirrored
# branch from upstream 'stable' branch of https://gerrit.
googlesource.com/git-repo
export REPO_REV='aosp/stable'
```

- **Install repo “Server certificate verification failed”**

  If you see a certificate error such as ‘Server certificate verification failed. CAfile: none CRLfile: none’, configure git to disable SSL certificate verification with git configuration.

Discuss with your IT administration for further guidance. You can use either of the following commands to disable SSL:

```
export GIT_SSL_NO_VERIFY=1
git config --global http.sslverify false
```

If your region is blocking access to `android.googlesource`, try the following configuration to fetch Repo from CodeLinaro Mirror:

```
git config --global url.https://git.codelinaro.org/clo/la/tools/
repo.insteadOf https://android.googlesource.com/tools/repo
```

- **error.GitError: git config (‘–replace-all’, ‘color.ui’, ‘auto’): error: could not write config file /home/$USER/.gitconfig: Device or resource busy**

  This error occurs when your gitconfig doesn’t set the UI color configuration. This configuration is set by default in Git 1.8.4 and later versions. Run the following command to enable the color display in your account:

  ```
  git config --global color.ui auto
  ```

- **[Error]: Failed preparing build for compilation. Error: Error setting docker credentials. Error: “Error saving credentials: error closing temp file: close /usr2/<userid>/.docker/config.json3322274803: disk quota exceeded\n”**

  QSC CLI uses the home directory only for a few kilo bytes (kB). Clear a few MBs from your home directory.

- **[Error]: The “path” argument must be of type string. Received undefined**

  Error excerpt

  ```
  qsc-cli download --workspace-path '/local/mnt/workspace/<userid>
  /K2L/QSC_CLI_BUILD/build' --product 'QCM6490.LE.1.0' --release
  'r00270.1' --distribution 'Qualcomm_Linux.SPF.1.0|TEST|DEVICE|
  PUBLIC'
  [Info]: Starting qsc-cli version 0.0.0.7
  (node:2924765) ExperimentalWarning: The Fetch API is an
  experimental feature. This feature could change at any time
  (Use `qsc-cli --trace-warnings ...` to show where the warning was
  created)
  [Info]: Checking if Workspace already exists
  [Info]: Saved updated Workspace info
  [Info]: Workspace Setup Completed
  [Error]: The "path" argument must be of type string. Received
  undefined
  ```

  Solution

This error may occur if QSC CLI is incompatible with Qlauncher. Qlauncher will be deprecated and replaced with a new application from the QSC. If you have Qlauncher in the workspace, you can run the following commands:

```
# Find your workspace within the Qlauncher UI
# Take a backup of the following metadata file if you want to
preserve the older workspace created with Qlauncher.
# These will work only with Qlauncher app. You can reinstall the
app at a later time again to access. If you don't
# need the workspaces, you can delete this file using:
mv /var/lib/qcom/data/qualcomm_launcher/workspaces2.json /var/
lib/qcom/data/qualcomm_launcher/workspaces2.json.bak
# Uninstall Qlauncher with the following command:
qpm-cli --uninstall qualcomm_launcher
```

- **docker: Error response from daemon: error while creating mount source path '/usr2/<userid>/.netrc': mkdir /usr2/<userid>/.netrc: permission denied**

  Error excerpt

```
Updating files: 100% (64/64), done.
2024-02-29T07:58:00: Sync Command Completed
2024-02-29T07:58:01: Finished setup.
[Info]: Setting Docker Credential
2024-02-29T07:58:03: Authorization successful
2024-02-29T07:58:03: Sync Command Starting
2024-02-29T07:58:03: Running Sync Command for SyncOpenSourceCode
- DownloadOpenSource
docker: Error response from daemon: error while creating mount
source path '/usr2/ramevelp/.netrc': mkdir /usr2/ramevelp/.
netrc: permission denied.
2024-02-29T07:58:04: Sync Command Failed
[Error]: Failed SP Download with error: 2024-02-29T07:58:04: Sp
Download failed. ExitCode: 126 Signal: 0  with errorcode 4
[Error]: 2024-02-29T07:58:04: Sp Download failed. ExitCode: 126
Signal: 0
```

  Solution

  This could happen due to the way IT has set up your home directory. Work with your IT administrator for any further changes to your home directory.

- **fatal: couldn't find remote ref refs/heads/qcom-linuxSTXscarthgap**

  If you see any junk characters while copying commands from the PDF, remove or replace the junk characters with appropriate symbols and rerun the command. You can also open the guide in HTML mode and use the copy command option.

Example

```
# Replace the following command
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linuxSTXscarthgap -m qcom-6.6.65-QLI.1.4-Ver.1.1.xml
# with
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linux-scarthgap -m qcom-6.6.65-QLI.1.4-Ver.1.1.xml
```

- **pull access denied for 032693710300.dkr.ecr.us-west-2.amazonaws.com/stormchaser/ql-tool**

  This error may occur while running the `qsc-cli` download command.

  Error excerpt

```
Unable to find image '032693710300.dkr.ecr.us-west-2.amazonaws.
com/stormchaser/ql-tool:20.04.20231220102843864.9' locally
docker: Error response from daemon: pull access denied for
032693710300.dkr.ecr.us-west-2.amazonaws.com/stormchaser/ql-
tool, repository doesn't exist or may require 'docker login':
denied: Your authorization token has expired. Reauthenticate and
try again.
```

  Solution

```
rm -rf ~/.docker/config.json
```

  Rerun the `qsc-cli` command.

8.3 Build

- **ERROR: linux-kernel-qcom-6.6-r0 do_menuconfig: No valid terminal found, unable to open devshell**

  This error can trigger while running a `bitbake linux-kernel-qcom -c menuconfig` command.

  Error excerpt

```
ERROR: linux-kernel-qcom-6.6-r0 do_menuconfig: No valid terminal
found, unable to open devshell.
Tried the following commands:
      tmux split-window -c "{cwd}" "do_terminal"
      tmux new-window -c "{cwd}" -n "linux-kernel-qcom
Configuration" "do_terminal"
```

```
      xfce4-terminal -T "linux-kernel-qcom Configuration" -e
"do_terminal"
      terminology -T="linux-kernel-qcom Configuration" -e do_
terminal
```

Solution

```
sudo apt install screen
sudo apt install tmux
```

- **NOTE: No reply from server in 30s**

  If you are seeing this error during the build on the rerun of `qsc-cli compile` or `bitbake` commands, you can try to delete `bitbake.lock`, `bitbake.sock`, and `hashserve.lock` from your partially built workspace and retry the build. For example, if you are building with `qsc-cli`, then these files are found under `<absoute_workspace_path>/DEV/LE.QCLINUX.1.0.r1/build-qcom-wayland`.

- **do_fetch: BitBake Fetcher Error: FetchError('Unable to fetch URL from any source')**

  These are intermittent fetch failures. Check if there is a network/host issue at your end, else check if the server is creating this issue. You can increase `postBuffer` and `maxRequestBuffer` settings in your `.gitconfig` file if the errors occur while fetching the git repositories. If you are using `qsc-cli`, then these configurations are already taken care of by the `qsc-cli` tool:

  ```
  git config --global http.postBuffer 1048576000
  git config --global http.maxRequestBuffer 1048576000
  ```

  If these configurations don't work, you can retry the compile to get past these intermittent errors for the first time.

  A few large git projects may show this error. For such projects, a feasible option is to manually clone as follows:

  ```
  cd <workspace_path>/downloads/git2/
  git clone --bare --mirror https://<url>/<project-name>.git
  <workspace_path>/downloads/git2/<local-name>.git
  touch <workspace_path>/downloads/git2/<local-name>.git.done
  ```

  For example, when `do_fetch` fails for `qualcomm_linux-spf-1-0-le-qclinux-1-0-r1_api-linux_history_prebuilts.git`, run the following command:

  ```
  git clone --bare --mirror https://qpm-git.qualcomm.com/home2/
  git/revision-history/qualcomm_linux-spf-1-0-le-qclinux-1-0-r1_
  api-linux_history_prebuilts.git <workspace_path>/downloads/git2/
  ```

```
qpm-git.qualcomm.com.home2.git.revision-history.qualcomm_linux-
spf-1-0-le-qclinux-1-0-r1_api-linux_history_prebuilts.git
touch <workspace_path>/downloads/git2/qpm-git.qualcomm.com.
home2.git.revision-history.qualcomm_linux-spf-1-0-le-qclinux-1-
0-r1_api-linux_history_prebuilts.git.done
```

After creating the `.done` file, proceed with the `bitbake <image recipe>` command. After completing the initial build, it's recommended to set up your own download directory.

- **make[4]: /bin/sh: Argument list too long**

  This happens when the path of the workspace exceeds 90 characters. Reduce the workspace path length to avoid this failure.

- **kernel-source/arch/arm64/boot/dts/qcom/qcm6490-idp.dts:8:10: fatal error: dt-bindings/iio/qcom,spmi-adc7-pmk8350.h: No such file or directory**

  The file in question `qcom,spmi-adc7-pmk8350.h` is part of the kernel source `<kernel-src>/include/dt-bindings/iio/qcom,spmi-adc7-pmk8350.h`.

  Check the workspace for this file and initialize the environment to pick this file. While compiling dtbs, the kernel build system runs a GCC preprocessor to replace the macros in dts files by its definition. The mentioned path is one such place where several `includes` reside.

  Check if you have `core.symlinks` set to `false` in your git configuration. If yes, set it to true:

```
git config --global core.symlinks true
```

- **qpm-git.qualcomm.com.home2.git.revision-history.qualcomm_ linux-spf-1-0-le-qclinux-1-0-r1_api-linux_history_prebuilts.git –progress failed with exit code 128, no output**

  128 is a masking error and this error needs further triage as it can be a network issue at your end or a genuine issue accessing Qualcomm or upstream mirrors. As a workaround for this error, see BitBake Fetcher Error. You can triage it further by following the subsequent instructions to dump verbose logs during fetch.

  By default, verbose logging isn't enabled for Yocto git fetch. To enable the same for all git projects, edit the `local.conf` file and change the `BB_GIT_VERBOSE_FETCH` value to **1**. You can also enable verbose logging for each recipe. For example, to enable verbose logging and debug a `do_fetch()` failure in a diag recipe, perform the following steps:

  1. Edit `layers/meta-qcom-hwe/recipes-bsp/diag/daig_15.0.bb` and add the line `BB_GIT_VERBOSE_FETCH = "1"`.
  2. Clean the earlier downloaded artifacts using `bitbake -fc cleanall diag`.
  3. Fetch the source again using `bitbake -fc fetch diag`.

4. Fetch the log with verbose logging available under the diag recipe's working directory `build-qcom-wayland/tmp-glibc/work/qcm6490-qcom-linux/diag/15.0-r0/temp`.
5. Share `log.do_fetch` from this path with the Qualcomm support team.

Note: Enabling git verbose logging for all recipes can significantly increase the build time. It's recommended to enable it only in required recipes on a need basis.

- **Failed SP Download with error: <> Sp Download failed. ExitCode: 128 Signal: 0 with errorcode 4**

  Error excerpt

```
warning: redirecting to https://git-na-ssl.chipcode.qti.
qualcomm.com/57f0ec058e47f7a82b2de7b95111c74a/qualcomm/qualcomm-
linux-spf-1-0_ap_standard_oem_nomodem.git/
remote: Counting objects: 129803, done.
remote: Compressing objects: 100% (114948/114948), done.
fatal: write error: No space left on device5 GiB | 1.63 MiB/s
fatal: fetch-pack: invalid index-pack output
2024-03-02T14:32:18: Sync Command Failed
[Error]: Failed SP Download with error: 2024-03-02T14:32:18: Sp
Download failed. ExitCode: 128 Signal: 0 with errorcode 4
[Error]: 2024-03-02T14:32:18: Sp Download failed. ExitCode: 128
Signal: 0
```

  Solution

  Error log indicates that there is no space on the device **"fatal: write error: No space left on device"**.

  Clean up the space and retrigger.

- **File "/usr/lib/python3.10/locale.py", line 620, in setlocale return _setlocale(category, locale)locale.Error: unsupported locale setting**

  To resolve this error, run the following commands and recompile:

```
sudo locale-gen en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

- **layer directories do not exist build-qcom-wayland/conf/../../layers/meta-qcom-qim-product-sdk**

This error occurs due to one of the following reasons:

– Git clone of `meta-qcom-qim-product-sdk` didn't complete successfully.

– `meta-qcom-qim-product-sdk` layer isn't exported to EXTRALAYERS.

Error excerpt

```
xxxx@xxxx:~/github_un/build-qcom-wayland$ bitbake qcom-
multimedia-image
ERROR: The following layer directories do not exist:
ERROR: <workspace_path>/build-qcom-wayland/conf/../../layers/
meta-qcom-qim-product-sdk
ERROR: Please check BBLAYERS in <workspace_path>/build-qcom-
wayland/conf/bblayers.conf
```

Solution

– Remove the `build-qcom-wayland` directory.

– Rerun the commands in Build QIMP SDK image.

- **failed: database disk image is malformed. abort()ing pseudo client by server request**

  The Pseudo tool gets path mismatch and corrupt database issues when processing the file system operations. When Pseudo simulates the file system operations in a Yocto project, problems can occur while handling file paths and permissions.

  This is a known issue in the Yocto community.

  Solution

  Run the following commands:

```
rm -rf <workspace_path>/build-qcom-robotics-ros2-humble/tmp-
glibc
bitbake -c cleanall pseudo-native & bitbake pseudo-native
../qirp-build qcom-robotics-full-image
```

- **pyinotify.WatchManagerError: No space left on device (ENOSPC)**

  Compilation triggers this error.

  Solution

  Run the following commands:

```
sudo su
echo 1048576 > /proc/sys/fs/inotify/max_user_watches
```

8.4 Flash

No known errors.

9 How to

9.1 Sync

Alternative methods to install Repo

The latest Repo works with python3. If your default Python is python2, then install `python-is-python3` to make python3 as the default Python.

```
mkdir -p ~/bin
cd ~/bin
# If you already have a previous directory of repo_tool, you can
delete it
rm -rf ~/bin/repo_tool
git clone https://android.googlesource.com/tools/repo.git -b v2.41
repo_tool
cd repo_tool
git checkout -b v2.41
export PATH=~/bin/repo_tool:$PATH
```

If the earlier steps didn't work, install Repo using the following commands:

```
# Install curl (if it isn't installed)
sudo apt install curl bc

# Latest Repo version works with python3

mkdir -p ~/bin
curl https://raw.githubusercontent.com/GerritCodeReview/git-repo/v2.
41/repo -o ~/bin/repo && chmod +x ~/bin/repo
export PATH=~/bin:$PATH
```

How does QSC CLI work?

1. **Setup**

   QSC CLI installs Docker and configures git.

2. **Sync**

   QSC CLI downloads the firmware sources and the Qualcomm Yocto layers, based on the input parameters.

3. **Build**

   QSC CLI builds the necessary Qualcomm firmware and the Qualcomm Yocto layers.

4. Internally, QSC CLI implements the standalone commands covered in the Build with GitHub using firmware and extras and leverages the prebuilt Docker images for the respective Qualcomm style software images. For example, `LE.QCLINUX.1.0.r1`.

View information about QSC CLI commands

To see all the commands provided by QSC CLI, run the following commands:

```
qsc-cli -h
qsc-cli download -h
```

To see more details about a particular command, you can append `-h` to the command. For example:

```
qsc-cli compile -h
```

Manage workspaces using QSC CLI

List the workspaces using the following command:

```
qsc-cli workspace info --list
```

To delete a workspace, run the following command:

```
qsc-cli workspace delete --workspace-path <workspace_path>

# Example, qsc-cli workspace delete --workspace-path '/local/mnt/
worskspace/Qworkspace_QIMPSDK'
```

Find a Yocto workspace using QSC CLI

You can install the `tree` command and run it on your workspace. The Yocto workspace is under the `LE.QCLINUX.1.0.r1` directory. These directories stay the same for future releases.

- **QSC CLI workspace structure after ``Qualcomm_Linux.SPF.1.0|TEST|DEVICE|PB_QIMPSDK`` distribution build**

  The following is a sample view, in which:

  – `LE.QCLINUX.1.0.r1` has the Yocto workspace.

  – Ignore the remaining directories as all the necessary sources and binaries are encoded in the Yocto layer recipes synced under `LE.QCLINUX.1.0.r1/layers`.

```
├── about.html
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   │   ├── build_levm.sh
│   │   ├── prebuilt_HY22
│   │   ├── snap_release.xml
│   │   ├── syncbuild.sh
│   │   └── sync_snap_v2.sh
│   ├── build-qcom-wayland
│   │   ├── bitbake-cookerdaemon.log
│   │   ├── bitbake.lock
│   │   ├── bitbake.sock
│   │   ├── buildhistory
│   │   ├── cache
│   │   ├── conf
│   │   ├── qim-prod-sdk
│   │   ├── qim-sdk
│   │   ├── tflite-sdk
│   │   └── tmp-glibc
│   ├── layers
│   │   ├── meta-openembedded
│   │   ├── meta-qcom
│   │   ├── meta-qcom-distro
│   │   ├── meta-qcom-hwe
│   │   ├── meta-qcom-qim-product-sdk
│   │   ├── meta-rust
│   │   ├── meta-security
│   │   ├── meta-selinux
│   │   ├── meta-virtualization
│   │   └── poky
│   └── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
│       ├── qcom
│       └── snap_release.xml
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   └── ufs
│   │       └── bin
│   │           ├── QCM6490_bootbinaries.zip
│   │           ├── QCM6490_dspso.zip
│   │           └── QCM6490_fw.zip
│   │   ├── config
│   │   ├── core_qupv3fw
│   │   ├── dataipa.gsifw
│   │   └── sectoolsv2
│   └── contents.xml
```

- **QSC CLI workspace structure after ``Qualcomm_ Linux.SPF.1.0|AP|Standard|OEM|NoModem`` distribution build with firmware and extras**

  The following is a sample view, in which:

  – `LE.QCLINUX.1.0.r1` has the Yocto workspace.

  – Few additional directories are for the Qualcomm firmware. While building with extras:

    ○ The additional firmware is built.

    ○ The output binaries from these are taken from the firmware recipes in the Qualcomm Yocto layers.

    ○ For detailed sync and build instructions, see Build with GitHub using firmware and

```
├── about.html
├── ADSP.HT.5.5.c8
│   ├── adsp_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_kodiak.adsp.prodQ.json
├── AOP.HO.3.0.2
│   ├── aop_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── AOP.HO.3.6
│   ├── aop_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── BOOT.MXF.1.0.c1
│   └── boot_images
├── BTFW.HSP.2.1.2
│   └── btfw_proc
├── BTFW.MOSELLE.1.1.0
│   └── btfw_proc
├── CDSP.HT.2.5.c3
│   ├── BuildProducts.txt
│   ├── cdsp_proc
│   └── VariantImgInfo_kodiak.cdsp.prodQ.json
├── CPUCP.FW.1.0
│   └── cpucp_proc
├── DSP.AT.1.0
│   └── dsp_proc
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   ├── build-qcom-wayland
│   ├── downloads
│   ├── layers
│   ├── qualcomm-linux-spf-1-0_hlos_oem_metadata
│   ├── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
│   └── sstate-cache
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   └── ufs
│   │       └── bin
│   │           ├── QCM6490_bootbinaries.zip
│   │           ├── QCM6490_dspso.zip
│   │           └── QCM6490_fw.zip
│   └── contents.xml
├── QCS9100.LE.1.0
│   ├── common
│   └── contents.xml
├── SAIL.SI.1.0
│   └── sail_proc
├── TZ.APPS.1.0
│   ├── qtee_tas
│   ├── ta_size_info_kodiak.csv
│   └── ta_size_info_lemans.csv
├── TZ.XF.5.0
│   ├── BuildProducts.txt
│   ├── HY22
│   └── trustzone_images
├── WLAN.HSP.1.1
│   └── wlan_proc
└── WLAN.MSL.2.0.c2
    └── wlan_proc
```

extras.

Refresh the workspace with a new download using QSC CLI

This option is supported only for the LE.QCLINUX.1.0.r1 image, which syncs the Yocto layers and prepares to build the Yocto workspace. This includes the following steps:

Note: Get to a Docker shell as mentioned in Generate an eSDK.

1. Download a new release:

```
repo init -u https://github.com/quic-yocto/qcom-manifest -b
qcom-linux-scarthgap -m <release tag>
repo sync
```

Note: For the <manifest release tag> information, see the section *Build-critical release tags* in the Release Notes. An example <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.1.xml.

2. Set up the build environment:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
```

Note: To know the MACHINE parameter values, see Release Notes.

3. Build the software image:

```
bitbake qcom-multimedia-image
```

9.2 Build

Check if the build is complete

If your build instruction is `bitbake qcom-multimedia-image`, check if the `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-multimedia-image
ls -al system.img
```

Generate an eSDK

Get a Docker shell and shell prompt

1. List the Docker images:

   ```
   docker images
   ```

   The output for this command is as follows:

   ```
   REPOSITORY TAG IMAGE ID CREATED SIZE
   032693710300.dkr.ecr.us-west-2.amazonaws.com/stormchaser/ql-tool
   20.04.20231220102843864.9 864b345bd707 2 months ago 715MB
   032693710300.dkr.ecr.us-west-2.amazonaws.com/stormchaser/le.um-k2c
   20.04.20231215014450998.7 4678dda58a91 2 months ago 929MB
   ```

2. Attach the container:

   ```
   WORKSPACE=<WORKSPACE_PATH> && SRC_DIR=<SoftwareImage> && docker
   run --rm  -it -v ~/.qualcomm_launcher_workspace_config:/var/tmp/
   .docker_qualcomm_launcher_setup/ -v $WORKSPACE:$WORKSPACE -e
   LOCAL_USER_NAME=`id -u -n` -e LOCAL_USER_ID=`id -u` -e USER=`id -
   u` -e WORKSPACE=$WORKSPACE -w $WORKSPACE/$SRC_DIR <REPOSITORY:
   TAG> bash

   # Example
   WORKSPACE=/local/mnt/workspace/Qworkspace/DEV && SRC_DIR=LE.
   QCLINUX.1.0.r1 && docker run --rm  -it -v ~/.qualcomm_launcher_
   workspace_config:/var/tmp/.docker_qualcomm_launcher_setup/ -v
   $WORKSPACE:$WORKSPACE -e LOCAL_USER_NAME=`id -u -n` -e LOCAL_
   USER_ID=`id -u` -e USER=`id -u` -e WORKSPACE=$WORKSPACE -w
   $WORKSPACE/$SRC_DIR 032693710300.dkr.ecr.us-west-2.amazonaws.
   com/stormchaser/le.um-k2c:20.04.20231215014450998.7 bash
   ```

Check if you are in a workspace that has `.repo` in it.

Set up the environment and generate eSDK:

Note: When the eSDK generation is complete, you can see the images in the following directory: `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/sdk`.

1. After building the `meta-qcom-hwe` with QSC CLI:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
bitbake -c do_populate_sdk_ext <image>
# Example, bitbake -c do_populate_sdk_ext qcom-multimedia-image
```

   Note: To know the `MACHINE` parameter values, see Release Notes.

2. After building with `meta-qcom-extras` and firmware sources with QSC CLI:

   Note: This step isn't applicable for public developers (unregistered).

```
# Example
export EXTRALAYERS="meta-qcom-extras"
export CUST_ID="213195"
export FWZIP_PATH="/local/mnt/workspace/extras/DEV/QCM6490.LE.1.
0/common/build/ufs/bin"
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
bitbake -c do_populate_sdk_ext qcom-multimedia-image
```

3. After building standalone instructions within the same shell (shell where the build is successful):

```
bitbake -c do_populate_sdk_ext <image>

# Example
bitbake -c do_populate_sdk_ext qcom-multimedia-image
```

4. After building with standalone instructions and with a new shell (assuming the build workspace exists):

```
cd <workspace_path>
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
source setup-environment
bitbake -c do_populate_sdk_ext <image>

# Example
cd /local/mnt/workspace/LE.QCLINUX.1.0.r1
MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-wayland QCOM_
SELECTED_BSP=custom source setup-environment
bitbake -c do_populate_sdk_ext qcom-multimedia-image
```

5. After building with standalone instructions using Dockerfile.

   a. Move the control to the workspace directory:

   ```
   cd /local/mnt/workspace/qcom-download-utils/<release>

   # Example
   cd /local/mnt/workspace/qcom-download-utils/qcom-6.6.65-QLI.
   1.4-Ver.1.1
   ```

   b. Set up the environment and issue an eSDK build:

   ```
   MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_
   BSP=custom source setup-environment
   bitbake -c do_populate_sdk_ext <image>

   # Example
   MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-wayland QCOM_
   SELECTED_BSP=custom source setup-environment
   bitbake -c do_populate_sdk_ext qcom-multimedia-image
   ```

Troubleshoot eSDK generation – basehash mismatch

Error excerpt

ERROR: When reparsing /local/mnt/workspace/extras/DEV/LE.QCLINUX.1.0.r1/build-qcom-wayland/conf/../../layers/meta-qcom-distro/recipes-products/images/qcom-multimedia-image.bb:do_populate_sdk_ext, the basehash value changed from 7bce27b0510cb666f1bba1d03f055cfef48f9db2eabc17d490e14bbe4c632eba to 48ccd9d7370e0bf2435aa8b5067162932e07a3832adfa6ca037aa0ddb765c8de. The metadata isn't deterministic and this needs to be fixed.
ERROR: The following commands may help:
ERROR: $ bitbake qcom-multimedia-image -cdo_populate_sdk_ext -Snone
ERROR: Then:
ERROR: $ bitbake qcom-multimedia-image -cdo_populate_sdk_ext -Sprintdiff

Solution

Rebuild the image and generate the eSDK again.

Rebuild using a Docker environment

Run the commands to connect to Docker for your environment setup and then use the BitBake commands to rebuild:

```
cd <workspace_path>/DEV/<softwareimage>
# Example, cd /local/mnt/workspace/Qworkspace/DEV/LE.QCLINUX.1.0.r1
for making changes to Yocto layers
# Make code changes
```

Note: Get to a Docker shell as mentioned in Generate an eSDK.

- Rebuild using your source changes:

```
# Rebuild commands
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
bitbake qcom-multimedia-image
```

Note: To know the `MACHINE` parameter values, see Release Notes.

- Build image `qcom-multimedia-test-image`:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
bitbake qcom-multimedia-test-image
```

Build a standalone QDL

Prerequisites:

- The modules `make` and `gcc` must be available.
- Install the following dependent packages:

```
sudo apt-get install git libxml2-dev libusb-1.0-0-dev pkg-config
```

1. Download and compile the Linux flashing tool (QDL):

```
git clone --branch master https://github.com/linux-msm/qdl
cd qdl
git checkout cbd46184d33af597664e08aff2b9181ae2f87aa6
make
```

2. Flash using the generated QDL:

```
./qdl --storage ufs --include <workspace_path>/build-qcom-
wayland/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-
multimedia-image <workspace_path>/build-qcom-wayland/tmp-glibc/
deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-image/
prog_firehose_ddr.elf <workspace_path>/build-qcom-wayland/tmp-
glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-
image/rawprogram*.xml <workspace_path>/build-qcom-wayland/tmp-
glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-multimedia-
image/patch*.xml
```

Change the Hexagon tool install path

The HEXAGON_ROOT environment variable must point to the path where the Hexagon tools are installed. By default, the qpm-cli tool installs a HEXAGON_ROOT variable in the $HOME directory. You can also choose an alternate installation directory.

Use the --path option in qpm-cli command to install Hexagon tools in a directory of your choice and export the HEXAGON_ROOT variable to the same directory.

Provide an absolute path for <TOOLS_DIR> in qpm-cli and export commands as shown in the following example:

```
# Example

mkdir -p <TOOLS_DIR>
qpm-cli --extract hexagon8.4 --version 8.4.07 --path <TOOLS_DIR>/8.4.
07
export HEXAGON_ROOT=<TOOLS_DIR>
```

Image recipes supported in the GitHub workflow

Image recipe	Description
qcom-minimal-image	A minimal rootfs image that boots to shell
qcom-console-image	Boot to shell with package group to bring in all the basic packages
qcom-multimedia-image	Image recipe includes recipes for multimedia software components, such as, audio, Bluetooth®, camera, computer vision, display, and video.
qcom-multimedia-test-image	Image recipe that includes tests

Download the Platform eSDK

1. Check the host computer requirements.
2. Set up the Ubuntu host.
3. Download the Platform eSDK.

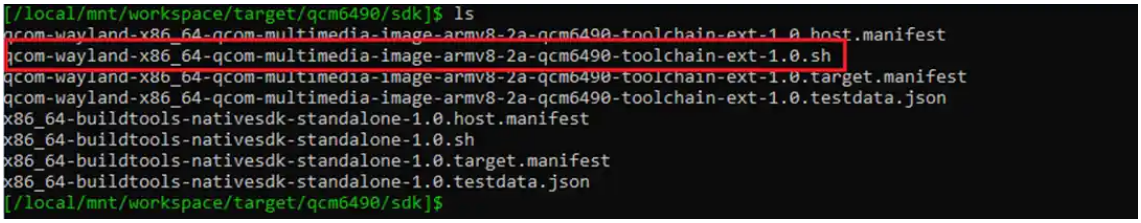
   Based on the required SoC, download the respective eSDK from the *Artifactory links to pre-built flashable images and eSDK* table of the Release Notes.

```
mkdir <esdk_download_path>
cd <esdk_download_path>
wget <flashable_binaries_download_link>
unzip <downloaded_zip_file>
```

Example:

```
mkdir <esdk_download_path>
cd <esdk_download_path>
wget https://artifacts.codelinaro.org/artifactory/qli-ci/
flashable-binaries/qimpsdk/qcs6490-rb3gen2-core-kit/x86-qcom-6.
6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2.zip
unzip x86-qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2.zip
```

After unzipping, you must see the eSDK installer at <esdk_download_path>:



If you don't have the necessary write permissions for the directory where you are trying to install the eSDK, the installer alerts you and then terminates. If this occurs, set up the permissions in the directory appropriately by using the following command and rerun the installer:

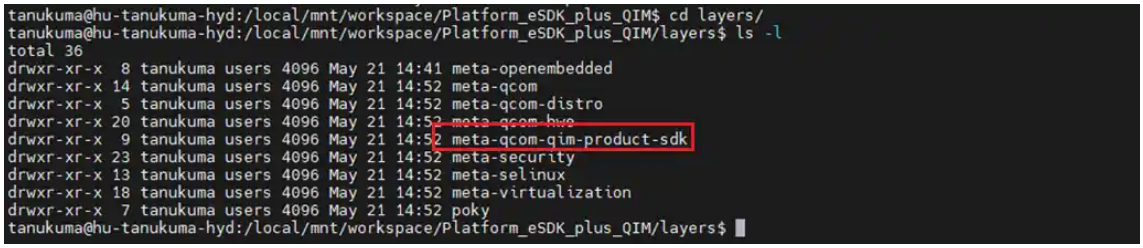
```
umask a+rx
```

4. Run the installer script to install the eSDK. For example:

```
sh ./qcom-wayland-x86_64-qcom-multimedia-image-armv8-2a-qcm6490-
toolchain-ext-1.0.sh
```

5. Follow the instructions on the console to install the eSDK in a convenient location on your host computer.
6. Ensure that the eSDK installation is successful.

   After installation, you can see the QIMP SDK layers under <workspace_path>/layers:



Note: Advanced developers can still build their own eSDK by following the steps mentioned in Advanced procedures.

7. Run the following command to set the `ESDK_ROOT` variable:

```
export ESDK_ROOT=<pathofinstallationdirectory>
```

For example:

```
export ESDK_ROOT=/local/mnt/workspace/Platform_eSDK_plus_QIM
```

The QIMP SDK installation is now complete. To develop an application for the device, see Develop your first application.

9.3 Developer workflow

Sync and build with real-time Linux

`meta-qcom-realtime` layer provides `PREEMPT_RT` patches for the Qualcomm Linux kernel. This layer is available in the Qualcomm GitHub and it's built on top of the BSP build image. For more information on this layer, see meta-qcom-realtime from the Qualcomm Linux Yocto Guide.

To sync and build real-time Linux, see Build real-time Linux image.

Migrate from the earlier release to the next release

The migration process depends on the development, branching, and integration workflows at your end. However, perform the following steps:

1. Integrate changes for the sources that you have branched:
   a. Skip this step if you haven't forked any underlying source code used by Qualcomm Yocto layers and are applying only `.patch` files.
   b. Qualcomm provides git history to all the source repositories. You can see a reference list of repositories in the Release Notes. For Qualcomm repositories branched and modified at your end, perform the following steps:

i. Compare the Qualcomm Yocto layers with your Yocto layers, and identify the repositories you must migrate.

ii. Check the Qualcomm Yocto layer recipes and identify the SRC_URI updates.

iii. Clone these repositories using `git clone` and check out the appropriate branch and SHA as pointed by the SRC_URI.

iv. Use `git merge` or the appropriate mechanism to bring the delta to your own fork according to your need.

v. Update your recipes as required.

2. Integrate the Yocto layer recipes – `git merge` the Yocto layer to pick up any changes.
3. Build and validate your resulting workspace.
4. Proceed with the next steps in your internal workflows.

Build a Qualcomm Linux kernel

See building kernel image.

Customize Qualcomm Yocto layers

See user customizations.

Download layers for the QIMP SDK build using the manifest release tag

Note: For the latest <manifest release tag>, see the section *Build-critical release tags* in the Release Notes.

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-
linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.1_qim-
product-sdk-1.1.2.xml
repo sync
```

Note: For the steps to set up environment and build software images, see Build QIMP SDK image.

Download layers for the QIRP SDK build by using the manifest release tag

Note: For the latest <manifest release tag>, see the section *Build-critical release tags* in the Release Notes.

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-
linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.0_
robotics-product-sdk-1.0.xml
repo sync
```

Note: For the steps to set up the environment and build software images, see Build QIRP SDK image.

Download layers for the real-time Linux build by using the manifest release tag

Note: For the latest <manifest release tag>, see the section *Build-critical release tags* in the Release Notes.

```
# cd to directory where you have 300 GB of free storage space to
create your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-
linux-scarthgap -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.65-QLI.1.4-Ver.1.1_
realtime-linux-1.1.xml
repo sync
```

Note: For the steps to set up the environment and build software images, see Build real-time Linux image.

Build a meta-qcom-qim-product-sdk layer as an add-on layer

If you have completed a base image build using GitHub standalone commands and already have an existing <WORKSPACE DIR>, follow these steps to build meta-qcom-qim-product-sdk:

1. Download the latest <meta-qcom-qim-product-sdk release tag>:

   ```
   cd <WORKSPACE DIR>
   rm -rf layers/meta-qcom-qim-product-sdk
   git clone https://github.com/quic-yocto/meta-qcom-qim-product-
   sdk -b <meta-qcom-qim-product-sdk release tag> layers/meta-qcom-
   qim-product-sdk
   ```

   Note: For more information on <meta-qcom-qim-product-sdk release tag>, see https://github.com/quic-yocto/meta-qcom-qim-product-sdk/tags. An example <meta-qcom-qim-product-sdk release tag> is qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-sdk-1.1.2.xml.

2. Set up the build environment:

```
MACHINE=<machine> DISTRO=qcom-wayland QCOM_SELECTED_BSP=custom
source setup-environment
# Example, MACHINE=qcs6490-rb3gen2-vision-kit DISTRO=qcom-
wayland QCOM_SELECTED_BSP=custom source setup-environment
# source setup-environment: Sets the environment, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory
```

Note: To know the `MACHINE` parameter values, see Release Notes.

3. Build the software image:

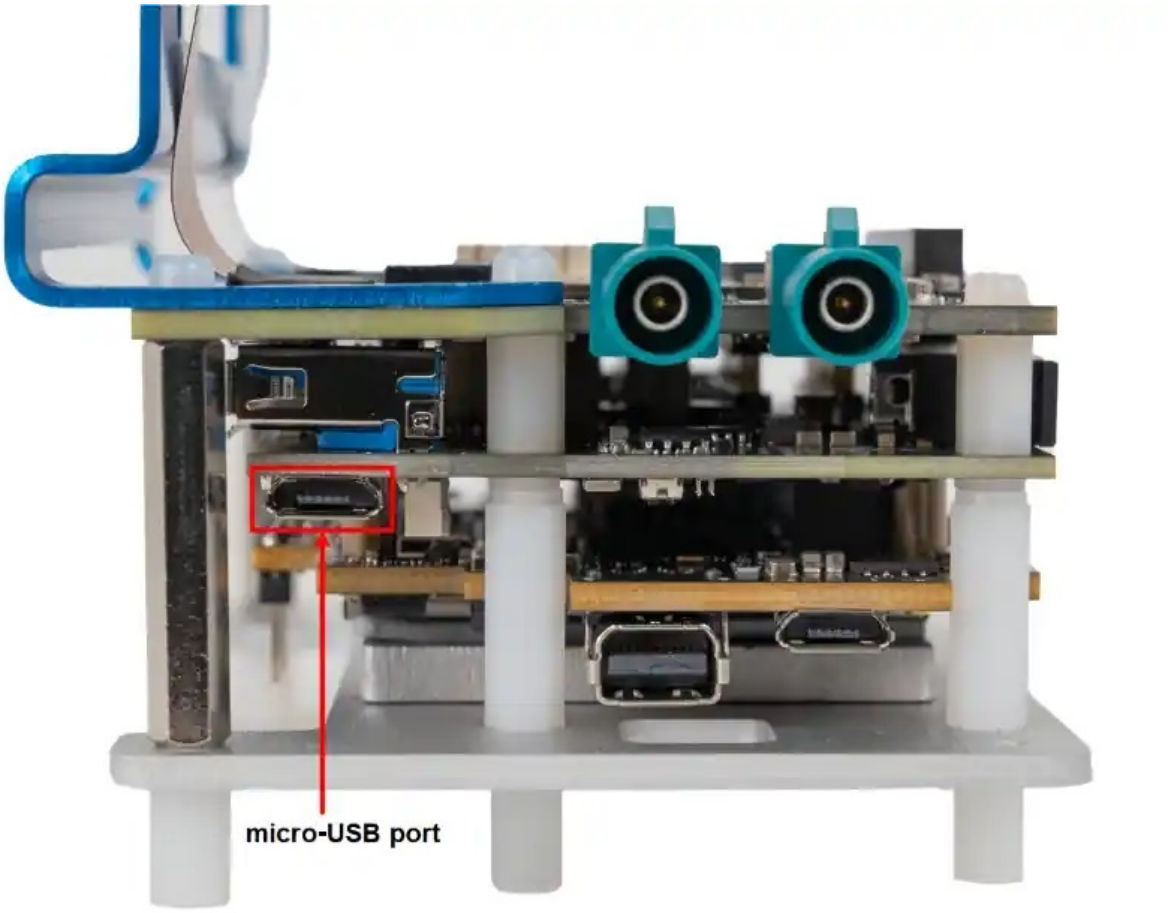
```
bitbake qcom-multimedia-image
# Build SDK image
bitbake qcom-qim-product-sdk
```

9.4 Setup

Connect to a UART shell

QCS6490/QCS5430

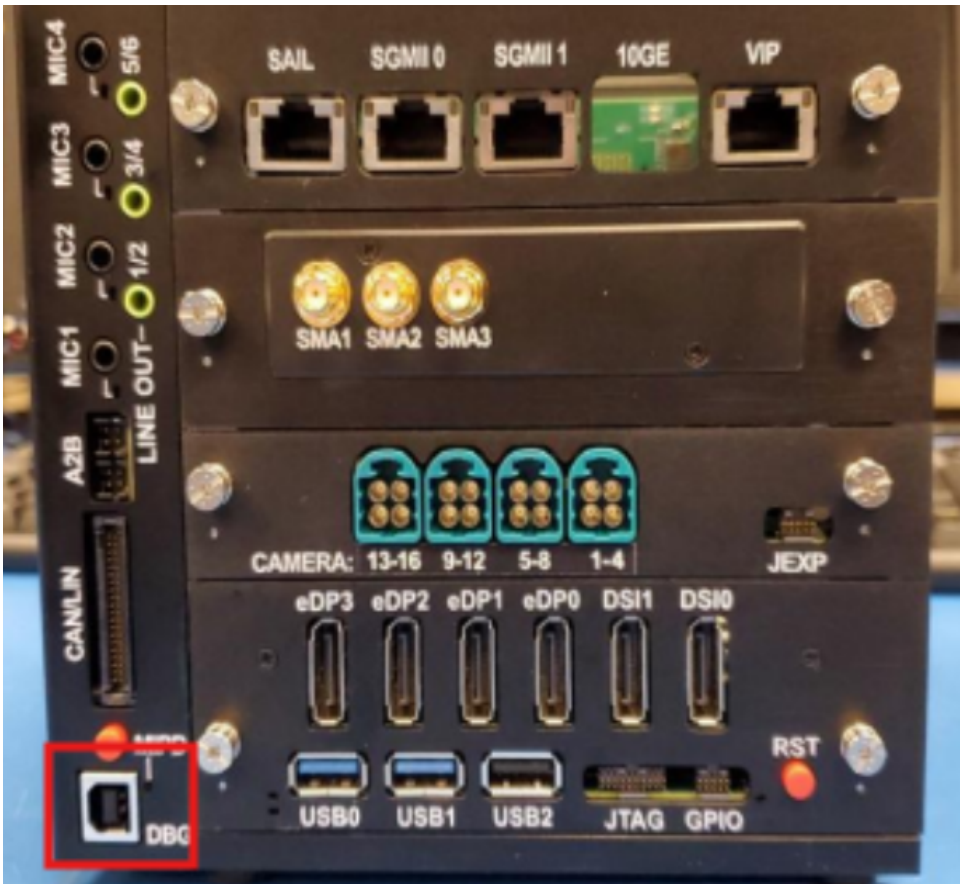
To set up the debug UART connection and view the diagnostic messages, connect the micro-USB cable from the micro-USB port on the RB3 Gen 2 device to the Linux host.



QCS9075

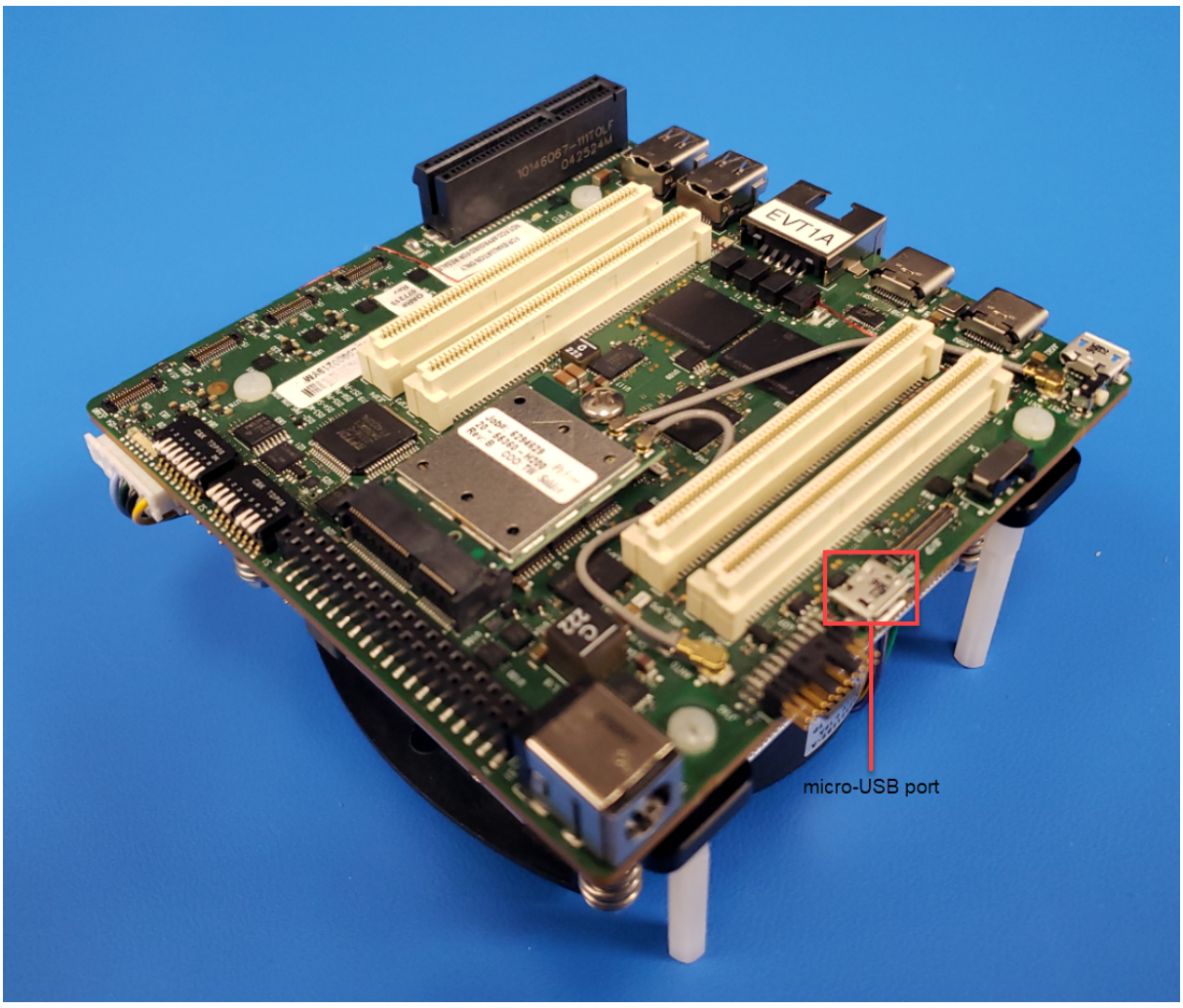
Qualcomm® IQ-9 Beta EVK

To set up the debug UART connection and view the diagnostic messages, connect the debug-USB cable from the debug-USB port on the Qualcomm® IQ-9 Beta Evaluation Kit device to the Linux host.



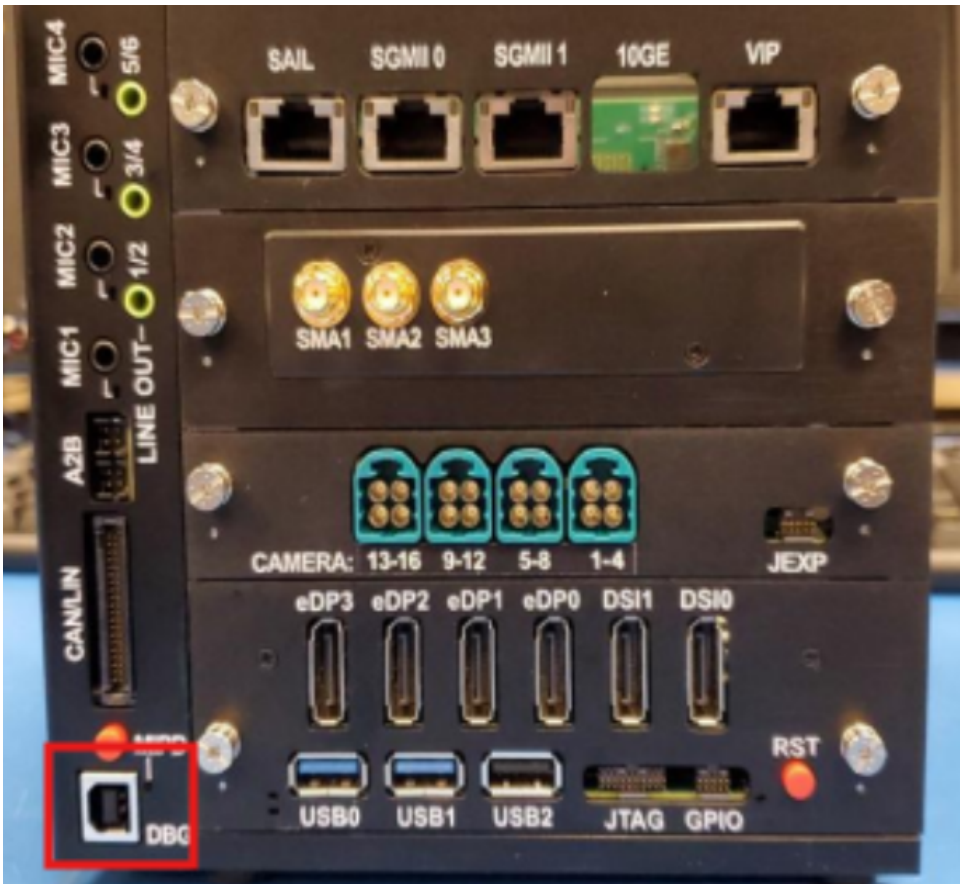
Dragonwing IQ-9075 EVK

To set up the debug UART connection and view the diagnostic messages, connect the debug-USB cable from the debug-USB port on the Qualcomm® Dragonwing™ IQ-9075 EVK device to the Linux host.



QCS8275

To set up the debug UART connection and view the diagnostic messages, connect the debug-USB cable from the debug-USB port on the Qualcomm® IQ-8 Beta Evaluation Kit device to the Linux host.



1. Install Minicom on the Linux host:

```
sudo apt update
sudo apt install minicom
```

2. Check the USB port:

```
ls /dev/ttyUSB*
```

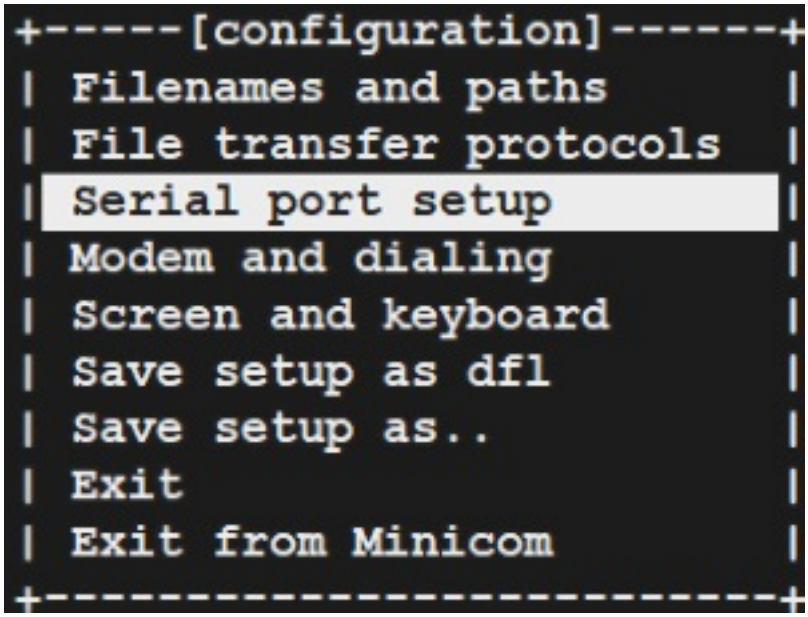
Sample output

```
/dev/ttyUSB0
```

3. Open Minicom:

```
sudo minicom -s
```

4. Use the Down arrow key to select the *Serial port setup* option. Use the Up and Down arrow keys to navigate through the menu.



5. Set up the serial device configuration:

Note: Ensure that the letters are in uppercase.

   a. Select *A* on your keyboard to set up the serial device name such as `/dev/ttyUSB0`.
   b. Select *Enter* to save the changes.
   c. Select *E* on your keyboard to set the baud rate and 8N1 configuration:
      i. Select the *E* key again if the baud rate isn't set to **115200**.
      ii. Select the *Q* key if the configuration isn't set to **8N1**.

```
+---------[Comm Parameters]----------+
|                                     |
|     Current: 115200 8N1             |
| Speed              Parity      Data |
| A: <next>          L: None     S: 5 |
| B: <prev>          M: Even     T: 6 |
| C:   9600          N: Odd      U: 7 |
| D:  38400          O: Mark     V: 8 |
| E: 115200          P: Space         |
|                                     |
| Stopbits                            |
| W: 1               Q: 8-N-1         |
| X: 2               R: 7-E-1         |
|                                     |
|                                     |
| Choice, or <Enter> to exit?         |
+-------------------------------------+
```

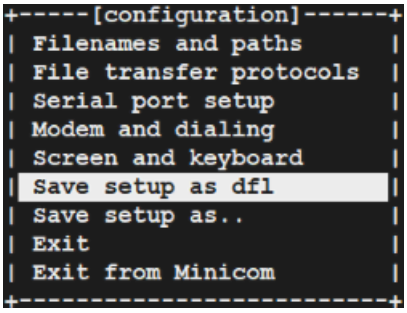
d. Select *Enter* to save the changes.

e. Select *F* on your keyboard to set the **Hardware Flow Control** to `No`.

```
+-----------------------------------------------+
| A -    Serial Device      : /dev/ttyUSB0      |
| B - Lockfile Location     : /var/lock         |
| C -   Callin Program      :                   |
| D -  Callout Program      :                   |
| E -    Bps/Par/Bits       : 115200 8N1        |
| F - Hardware Flow Control : No                |
| G - Software Flow Control : No                |
| H -     RS485 Enable      : No                |
| I -   RS485 Rts On Send   : No                |
| J -  RS485 Rts After Send : No                |
| K -  RS485 Rx During Tx   : No                |
| L -  RS485 Terminate Bus  : No                |
| M - RS485 Delay Rts Before: 0                 |
| N - RS485 Delay Rts After : 0                 |
|                                               |
|    Change which setting?                      |
+-----------------------------------------------+
```

f. Select *Enter* to save the changes.

6. Select the *Save setup as dfl* option and then select *Enter*.



7. Select *EXIT* to open the UART console and then select *Enter*.
8. Sign in to the UART console:
   - Login: `root`
   - Password: `oelinux123`

   Note: If the sign in console doesn't display as expected, verify the USB connection. If the issue persists, disconnect and then reconnect the micro-USB.

Note: If you want to run sample applications from the UART shell, remount the root file system with write permissions:

```
mount -o rw,remount /
```

Connect to ADB

See Install and connect to ADB.

Connect to the network

Set up Wi-Fi

Wi-Fi is operational in Station mode. The Wi-Fi host driver and the authentication for network management are initialized during the device boot up.

1. Connect to the Wireless Access Point (Wi-Fi Router):

   ```
   nmcli dev wifi connect <WiFi-SSID> password <WiFi-password>
   ```

   Example

   ```
   root@qcs6490-rb3gen2-vision-kit:~# nmcli dev wifi connect
   QualcommWiFi password 1234567890
   ```

```
Device 'wlan0' successfully activated with 'd7b990bd-3b77-4b13-b239-b706553abaf8'.
```

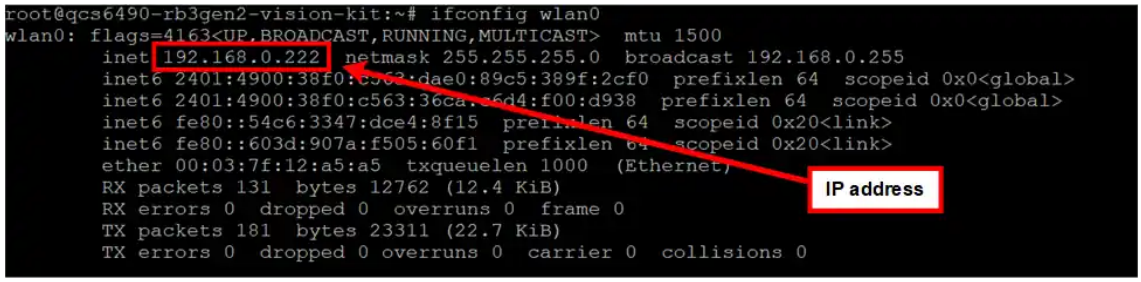
2. Check the connection and device status:

```
nmcli -p device
```

```
root@qcs6490-rb3gen2-vision-kit:~# nmcli -p device
=====================
  Status of devices
=====================
DEVICE  TYPE      STATE         CONNECTION
---------------------------------------------------------------------------
wlan0   wifi      connected     QualcommWiFi
eth0    ethernet  unavailable   --
eth1    ethernet  unavailable   --
can0    can       unmanaged     --
lo      loopback  unmanaged     --
root@qcs6490-rb3gen2-vision-kit:~#
```

3. Check the WLAN connection status and IP address:

```
ifconfig wlan0
```



4. Ensure that the connection is active by pinging any website:

```
ping google.com
```

Switching networks

If you are already connected to a network and need to reconnect to another network, do the following:

1. Disconnect from the current network:

```
nmcli c down QualcommWiFi
```

```
Connection 'QualcommWiFi' successfully deactivated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/1)
```

2. Check the disconnect status:

```
nmcli -p device
```

```
root@qcs6490-rb3gen2-vision-kit:~# nmcli -p device
=====================
  Status of devices
=====================
DEVICE  TYPE      STATE         CONNECTION
-------------------------------------------------------------------
wlan0   wifi      disconnected  --
eth0    ethernet  unavailable   --
eth1    ethernet  unavailable   --
can0    can       unmanaged     --
lo      loopback  unmanaged     --
```

3. Connect to a different Wi-Fi network:

```
nmcli dev wifi connect QualcommAP password XXXXXXXXX
```

```
Device 'wlan0' successfully activated with '6159ac7c-58c2-44fa-938f-45dcb544fac3'.
```

Sign in using SSH

Establish the network connectivity before connecting to SSH.

1. Locate the IP address of the RB3 Gen 2 device according to the type of network connection, using the UART console on the Linux host:

   For Ethernet:

   ```
   ifconfig eth2
   ```

   For Wi-Fi:

   ```
   ifconfig wlan0
   ```

2. Use the IP address from the Linux host to establish an SSH connection to the device:

   ```
   ssh root@ip-address
   ```

   Example

   ```
   ssh root@192.168.0.222
   ```

3. Connect to the SSH shell using the following password:

```
oelinux123
```

Note:

- Connect the remote host to the same Wi-Fi access point as the device.
- To create a non-root user account, see Create a non-root user account.

Configure Ethernet with RJ45 port

Ethernet/RJ45 port is enabled as a downstream port of PCIe to USB controller (`renesas`). Ensure that the `renesas_usb_fw.mem` file is available at the `var/usbfw` directory.

Note:

- If the `renesas_usb_fw.mem` firmware isn't available at the `var/usbfw` directory, then update USB and Ethernet controller firmware.

To check if the USB to Ethernet controller is enumerated, run the following command:

```
lsusb
```

Sample output:

```
Bus 002 Device 003: ID 0b95:1790 ASIX Electronics Corp. AX88179
Gigabit Ethernet
Bus 002 Device 002: ID 05e3:0625 Genesys Logic, Inc. USB3.2 Hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 05e3:0610 Genesys Logic, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Connect an RJ45 cable to the RB3 Gen 2 device.

To check the Ethernet IP address, run the following command:

```
ifconfig
```

Sample output:

Note: 10.219.0.106 is the IP address.

Update USB and Ethernet controller firmware

If you face USB or Ethernet connectivity issues on the RB3 Gen 2 device, consider updating the firmware for the USB controller.

Prerequisites

- Upgrade the software as described in Update the software before updating the Renesas firmware.
- Connect the device to the Linux host through the USB Type-C cable.

Note: The following procedure is applicable only to Ubuntu 22.04 host. If you are using a Windows or macOS host, set up an Ubuntu virtual machine by following the instructions described in the Virtual Machine Setup Guide.

1. Register and sign in to Renesas.
2. Download the firmware.
3. Create the `usb_fw.img` image and copy the USB firmware:

```
dd if=/dev/zero of=usb_fw.img bs=4k count=240
mkfs -t ext4 usb_fw.img
mkdir usb_fw
sudo mount -o loop usb_fw.img usb_fw/
sudo cp -rf renesas_usb_fw.mem usb_fw
sudo umount usb_fw
```

4. Start the device in Fastboot mode:

```
adb root
adb shell
reboot bootloader
```

5. Check if the device is in Fastboot mode:

```
fastboot devices
```

```
7dc85f5e fastboot
```

6. Flash the `usb_fw.img` image to the device:

```
fastboot erase usb_fw
fastboot flash usb_fw usb_fw.img
fastboot reboot
```

```
c:>fastboot erase usb_fw
Erasing 'usb_fw' FAILED (remote: 'Check device console.')
fastboot: error: Command failed
```

7. Verify if the firmware is successfully updated:

```
dmesg
```

Sample log after the firmware is successfully updated.

```
[    6.589462] usbcore: registered new device driver
onboard-usb-hub
[    6.653277] usb 2-1: new SuperSpeed USB device number 2 using
xhci_hcd
[    7.013061] usb 2-1.1: new SuperSpeed USB device number 3
using xhci_hcd
[    7.120657] ax88179_178a 2-1.1:1.0 eth0: register 'ax88179_
178a' at usb-0001:04:00.0-1.1, ASIX AX88179 USB 3.0 Gigabit
Ethernet, 3e:9e:5e:ff:d3:fb
[    7.120767] usbcore: registered new interface driver ax88179_
178a
```

10 References

10.1 Workspace view

This section provides sample workspace structures with `qsc-cli` and GitHub workflow standalone use cases, for the supported hardware SoCs.

Workspace structure with qsc-cli

- The following figure shows the directory structure before the `Qualcomm_ Linux.SPF.1.0|TEST|DEVICE|PB_QIMPSDK` distribution build (`LE.QCLINUX.1.0.r1` has the Yocto workspace):

```
├── about.html
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   │   ├── build_levm.sh
│   │   ├── prebuilt_HY22
│   │   ├── snap_release.xml
│   │   ├── syncbuild.sh
│   │   └── sync_snap_v2.sh
│   ├── layers
│   │   ├── meta-openembedded
│   │   ├── meta-qcom
│   │   ├── meta-qcom-distro
│   │   ├── meta-qcom-hwe
│   │   ├── meta-qcom-qim-product-sdk
│   │   ├── meta-rust
│   │   ├── meta-security
│   │   ├── meta-selinux
│   │   ├── meta-virtualization
│   │   └── poky
│   └── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
│       ├── qcom
│       └── snap_release.xml
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   └── ufs
│   │       └── bin
│   │           ├── QCM6490_bootbinaries.zip
│   │           ├── QCM6490_dspso.zip
│   │           └── QCM6490_fw.zip
│   │   ├── config
│   │   ├── core_qupv3fw
│   │   ├── dataipa.gsifw
│   │   └── sectoolsv2
│   └── contents.xml
```

- The following figure shows the directory structure after `Qualcomm_ Linux.SPF.1.0|TEST|DEVICE|PB_QIMPSDK` distribution build:

```
├── about.html
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   │   ├── build_levm.sh
│   │   ├── prebuilt_HY22
│   │   ├── snap_release.xml
│   │   ├── syncbuild.sh
│   │   └── sync_snap_v2.sh
│   ├── build-qcom-wayland
│   │   ├── bitbake-cookerdaemon.log
│   │   ├── bitbake.lock
│   │   ├── bitbake.sock
│   │   ├── buildhistory
│   │   ├── cache
│   │   ├── conf
│   │   ├── qim-prod-sdk
│   │   ├── qim-sdk
│   │   ├── tflite-sdk
│   │   └── tmp-glibc
│   ├── layers
│   │   ├── meta-openembedded
│   │   ├── meta-qcom
│   │   ├── meta-qcom-distro
│   │   ├── meta-qcom-hwe
│   │   ├── meta-qcom-qim-product-sdk
│   │   ├── meta-rust
│   │   ├── meta-security
│   │   ├── meta-selinux
│   │   ├── meta-virtualization
│   │   └── poky
│   └── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
│       ├── qcom
│       └── snap_release.xml
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   └── ufs
│   │       └── bin
│   │           ├── QCM6490_bootbinaries.zip
│   │           ├── QCM6490_dspso.zip
│   │           └── QCM6490_fw.zip
│   │   ├── config
│   │   ├── core_qupv3fw
│   │   ├── dataipa.gsifw
│   │   └── sectoolsv2
│   └── contents.xml
```

- The following figure shows the directory structure before `Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem` distribution build with firmware and extras:

```
├── about.html
├── ADSP.HT.5.5.c8
│   └── adsp_proc
├── AOP.HO.3.6
│   ├── aop_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── BOOT.MXF.1.0.c1
│   └── boot_images
├── BTFW.HSP.2.1.2
│   └── btfw_proc
├── BTFW.MOSELLE.1.1.0
│   └── btfw_proc
├── CDSP.HT.2.5.c3
│   └── cdsp_proc
├── CPUCP.FW.1.0
│   └── cpucp_proc
├── DSP.AT.1.0
│   └── dsp_proc
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   ├── layers
│   ├── qualcomm-linux-spf-1-0_hlos_oem_metadata
│   └── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
├── QCM6490.LE.1.0
│   ├── common
│   └── contents.xml
├── SAIL.SI.1.0
│   └── sail_proc
├── TZ.APPS.1.0
│   ├── qtee_tas
│   ├── ta_size_info_kodiak.csv
│   └── ta_size_info_lemans.csv
├── TZ.XF.5.0
│   └── trustzone_images
├── WLAN.HSP.1.1
│   └── wlan_proc
└── WLAN.MSL.2.0.c2
    └── wlan_proc
```

- The following figure shows the directory structure after `Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem` distribution build with firmware and extras:

```
├── about.html
├── ADSP.HT.5.5.c8
│   ├── adsp_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_kodiak.adsp.prodQ.json
├── AOP.HO.3.0.2
│   ├── aop_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── AOP.HO.3.6
│   ├── aop_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── BOOT.MXF.1.0.c1
│   └── boot_images
├── BTFW.HSP.2.1.2
│   └── btfw_proc
├── BTFW.MOSELLE.1.1.0
│   └── btfw_proc
├── CDSP.HT.2.5.c3
│   ├── BuildProducts.txt
│   ├── cdsp_proc
│   └── VariantImgInfo_kodiak.cdsp.prodQ.json
├── CPUCP.FW.1.0
│   └── cpucp_proc
├── DSP.AT.1.0
│   └── dsp_proc
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   ├── build-qcom-wayland
│   ├── downloads
│   ├── layers
│   ├── qualcomm-linux-spf-1-0_hlos_oem_metadata
│   ├── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
│   └── sstate-cache
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   └── ufs
│   │       └── bin
│   │           ├── QCM6490_bootbinaries.zip
│   │           ├── QCM6490_dspso.zip
│   │           └── QCM6490_fw.zip
│   └── contents.xml
├── QCS9100.LE.1.0
│   ├── common
│   └── contents.xml
├── SAIL.SI.1.0
│   └── sail_proc
├── TZ.APPS.1.0
│   ├── qtee_tas
│   ├── ta_size_info_kodiak.csv
│   └── ta_size_info_lemans.csv
├── TZ.XF.5.0
│   ├── BuildProducts.txt
│   ├── HY22
│   └── trustzone_images
├── WLAN.HSP.1.1
│   └── wlan_proc
└── WLAN.MSL.2.0.c2
    └── wlan_proc
```

Workspace structure with GitHub workflow standalone instructions

- The following figure shows the directory structure before the GitHub workflow QIMP SDK standalone build:

```
├── layers
│   ├── bitbake-cookerdaemon.log
│   ├── meta-openembedded
│   ├── meta-qcom
│   ├── meta-qcom-distro
│   ├── meta-qcom-hwe
│   ├── meta-qcom-qim-product-sdk
│   ├── meta-rust
│   ├── meta-security
│   ├── meta-selinux
│   ├── meta-virtualization
│   └── poky
└── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
```

- The following figure shows the directory structure after the GitHub workflow QIMP SDK standalone build:

```
├── build-qcom-wayland
│   ├── bitbake-cookerdaemon.log
│   ├── buildhistory
│   ├── cache
│   ├── conf
│   ├── qim-prod-sdk
│   ├── qim-sdk
│   ├── tflite-sdk
│   └── tmp-glibc
├── downloads
│   ├── a52dec-0.7.4.tar.gz
│   ├── a52dec-0.7.4.tar.gz.done
│   ├── acl-2.3.1.tar.gz
      --<ALL downloads Contents>
│   └── zlib-1.2.11.tar.xz.done
├── layers
│   ├── bitbake-cookerdaemon.log
│   ├── meta-openembedded
│   ├── meta-qcom
│   ├── meta-qcom-distro
│   ├── meta-qcom-hwe
│   ├── meta-qcom-qim-product-sdk
│   ├── meta-rust
│   ├── meta-security
│   ├── meta-selinux
│   ├── meta-virtualization
│   └── poky
├── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
└── sstate-cache
    ├── 00
    ├── 01
     --<sstate-cache-objects>
```

- The following figure shows the directory structure after building firmware of `qualcomm-linux-spf-1-0_ap_standard_oem_nomodem`:

Note:

– `qualcomm-linux-spf-1-0_ap_standard_oem_nomodem` contains the downloaded select firmware sources.

– `LE.QCLINUX.1.0.r1` has the built Yocto workspace.

```
├── about.html
├── ADSP.HT.5.5.c8
│   ├── adsp_proc
│   │   ├── avs
│   │   ├── build
│   │   ├── config
│   │   ├── core
│   │   ├── crashman
│   │   ├── dsp_kodiak.adsp.prodQ.elf
│   │   ├── eai_lib
│   │   ├── gpr
│   │   ├── image_info.json
│   │   ├── loc
│   │   ├── performance
│   │   ├── platform
│   │   ├── qdsp6
│   │   ├── qmimsgs
│   │   ├── qsh_algorithms
│   │   ├── qsh_api
│   │   ├── qsh_platform
│   │   ├── ssc_algorithms
│   │   ├── ssc_drivers
│   │   └── tools
│   ├── BuildProducts.txt
│   └── VariantImgInfo_kodiak.adsp.prodQ.json
├── AOP.HO.3.0
│   ├── aop_proc
│   │   ├── build
│   │   ├── core
│   │   ├── pack
│   │   └── tools
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── BOOT.MXF.1.0.c1
│   └── boot_images
│       ├── boot
│       ├── boot_tools
│       ├── Build
│       ├── BuildLogs
│       ├── edk2
│       ├── sdk
│       ├── sectools
│       └── ssg
├── BTFW.HSP.2.1.2
│   └── btfw_proc
│       ├── btfm
│       ├── build
│       └── out
├── BTFW.MOSELLE.1.1.0
│   └── btfw_proc
│       ├── btfm
│       ├── build
│       └── out
├── CDSP.HT.2.5.c3
│   ├── BuildProducts.txt
│   ├── cdsp_proc
│   │   ├── build
│   │   ├── ceng
│   │   ├── config
│   │   ├── core
│   │   ├── crashman
│   │   ├── cvp
│   │   ├── dsp_kodiak.cdsp.prodQ.elf
│   │   ├── image_info.json
│   │   ├── performance
│   │   ├── platform
│   │   ├── qdsp6
│   │   ├── qmimsgs
│   │   ├── tools
│   │   ├── vap
│   │   └── video_processing
│   └── VariantImgInfo_kodiak.cdsp.prodQ.json
├── CPUCP.FW.1.0
│   └── cpucp_proc
│       ├── kodiak
│       └── lemansau
├── DSP.AT.1.0
│   └── dsp_proc
│       ├── ams_spf
│       ├── avs
│       ├── awe
│       ├── build
│       ├── ceng
│       ├── config
│       ├── core
│       ├── crashman
│       ├── data
│       ├── eai_lib
│       ├── gpr
│       ├── image_info.json
│       ├── performance
│       ├── platform
│       ├── qdsp6
│       ├── qmimsgs
│       └── tools
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   │   ├── build_levm.sh
│   │   ├── prebuilt_HY11
│   │   ├── snap_release.xml
│   │   ├── sources
│   │   ├── syncbuild.sh
│   │   └── sync_snap_v2.sh
│   ├── build-qcom-wayland
│   │   ├── bitbake-cookerdaemon.log
│   │   ├── buildhistory
│   │   ├── cache
│   │   ├── conf
│   │   └── tmp-glibc
│   ├── downloads
│   │   ├── acl-2.3.1.tar.gz
│   │   ├── acl-2.3.1.tar.gz.done
│   │   │       ~~<ALL downloads Contents>
│   ├── layers
│   │   ├── meta-openembedded
│   │   ├── meta-qcom
│   │   ├── meta-qcom-distro
│   │   ├── meta-qcom-extras
│   │   ├── meta-qcom-hwe
│   │   ├── meta-rust
│   │   ├── meta-security
│   │   ├── meta-selinux
│   │   ├── meta-virtualization
│   │   └── poky
│   ├── qualcomm-linux-spf-1-0_hlos_oem_metadata
│   │   ├── about.html
│   │   ├── QCM6490.LE.1.0
│   │   ├── QCM8550.LE.2.0
│   │   └── QCS9100.LE.1.0
│   ├── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
│   └── sstate-cache
│       ├── 00
│       ├── 01
│       │       ~~<sstate-cache-objects>
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
│       ├── qcom
│       ├── snap_release.xml
│       └── sync_snap_v2.sh
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   ├── config
│   │   ├── core_qupv3fw
│   │   ├── dataipa_gsifw
│   │   └── sectoolsv2
│   └── contents.xml
├── QCS9100.LE.1.0
│   ├── common
│   │   ├── aurixfw
│   │   ├── build
│   │   ├── cdt
│   │   ├── config
│   │   ├── Core
│   │   └── sectoolsv2
│   └── contents.xml
├── SAIL.SI.1.0
│   └── sail_proc
│       ├── BSP
│       └── build
├── TZ.APPS.1.0
│   ├── qtee_tas
│   │   ├── apps
│   │   ├── build
│   │   └── sdk
│   ├── ta_size_info_kodiak.csv
│   └── ta_size_info_lemans.csv
├── TZ.XF.5.0
│   ├── BuildProducts.txt
│   ├── HY22
│   │   └── trustzone_images
│   └── trustzone_images
│       ├── build
│       ├── core
│       ├── scons_dep_tree_EACAANAA_2eb56721f.txt
│       ├── sectools
│       ├── signed_images.json
│       ├── ssg
│       └── tools
├── WLAN.HSP.1.1
│   └── wlan_proc
│       ├── build
│       ├── config
│       ├── core
│       ├── scripts
│       ├── tools
│       └── wlan
└── WLAN.MSL.2.0.c2
    └── wlan_proc
        ├── build
        ├── scripts
        ├── tools
        └── wlan
```

- The following figure shows the directory structure after building firmware of `qualcomm-linux-spf-1-0_amss_standard_oem_nomodem`:

Note:

– `qualcomm-linux-spf-1-0_amss_standard_oem_nomodem` contains the downloaded select firmware sources.

– `LE.QCLINUX.1.0.r1` has the built Yocto workspace.

```
├── about.html
├── ADSP.HT.5.5.c8
│   ├── adsp_proc
│   │   ├── avs
│   │   ├── build
│   │   ├── config
│   │   ├── core
│   │   ├── crashman
│   │   ├── dsp_kodiak.adsp.prodQ.elf
│   │   ├── eai_lib
│   │   ├── gpr
│   │   ├── image_info.json
│   │   ├── loc
│   │   ├── performance
│   │   ├── platform
│   │   ├── qdsp6
│   │   ├── qmimsgs
│   │   ├── qsh_algorithms
│   │   ├── qsh_api
│   │   ├── qsh_platform
│   │   ├── ssc_algorithms
│   │   ├── ssc_drivers
│   │   └── tools
│   ├── BuildProducts.txt
│   └── VariantImgInfo_kodiak.adsp.prodQ.json
├── ADSP.HT.5.8.c4
│   └── adsp_proc
│       ├── avs
│       ├── build
│       ├── config
│       ├── core
│       ├── crashman
│       ├── eai_lib
│       ├── enpu
│       ├── gpr
│       ├── image_info.json
│       ├── loc
│       ├── performance
│       ├── platform
│       ├── qdsp6
│       ├── qmimsgs
│       ├── qsh
│       ├── sdc_core
│       ├── sdc_sensors
│       ├── ssc
│       ├── ssc_api
│       ├── ssc_drivers
│       └── tools
├── AOP.HO.3.6
│   ├── aop_proc
│   │   ├── build
│   │   ├── core
│   │   ├── pack
│   │   └── tools
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── BOOT.MXF.1.0.c1
│   └── boot_images
│       ├── boot
│       ├── boot_tools
│       ├── Build
│       ├── Build.cgs
│       ├── edk2
│       ├── sdk
│       ├── sectools
│       └── ssg
├── BTFW.HSP.2.1.2
│   └── btfw_proc
│       ├── btfw
│       ├── build
│       └── out
├── BTFW.MOSELLE.1.1.0
│   └── btfw_proc
│       ├── btfw
│       ├── build
│       └── out
├── CDSP.HT.2.5.c3
│   ├── BuildProducts.txt
│   ├── cdsp_proc
│   │   ├── build
│   │   ├── ceng
│   │   ├── config
│   │   ├── core
│   │   ├── crashman
│   │   ├── cvp
│   │   ├── dsp_kodiak.cdsp.prodQ.elf
│   │   ├── image_info.json
│   │   ├── performance
│   │   ├── platform
│   │   ├── qdsp6
│   │   ├── qmimsgs
│   │   ├── tools
│   │   ├── vap
│   │   └── video_processing
│   └── VariantImgInfo_kodiak.cdsp.prodQ.json
├── CPUCP.FW.1.0
│   └── cpucp_proc
│       ├── kodiak
│       └── lemansau
├── LE.QCLINUX.1.0.r1
│   ├── appx_proc
│   │   ├── build_levm.sh
│   │   ├── prebuilt_HY11
│   │   ├── snap_release.xml
│   │   ├── sources
│   │   ├── syncbuild.sh
│   │   └── sync_snap_v2.sh
│   ├── build-qcom-wayland
│   │   ├── bitbake-cookerdaemon.log
│   │   ├── bitbake.lock
│   │   ├── bitbake.sock
│   │   ├── buildhistory
│   │   ├── cache
│   │   ├── conf
│   │   └── tmp-glibc
│   ├── downloads
│   │   ├── acl-2.3.1.tar.gz
│   │   ├── acl-2.3.1.tar.gz.done
│   │   │   ~~<ALL downloads Contents>
│   ├── layers
│   │   ├── meta-openembedded
│   │   ├── meta-qcom
│   │   ├── meta-qcom-distro
│   │   ├── meta-qcom-extras
│   │   ├── meta-qcom-hwe
│   │   ├── meta-rust
│   │   ├── meta-security
│   │   ├── meta-selinux
│   │   ├── meta-virtualization
│   │   └── poky
│   ├── qualcomm-linux-spf-1-0_hlos_oem_metadata
│   │   ├── about.html
│   │   ├── QCM6490.LE.1.0
│   │   ├── QCM8550.LE.2.0
│   │   └── QCS9100.LE.1.0
│   ├── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
│   └── sstate-cache
│       ├── 00
│       ├── 01
│       │   ~~<sstate-cache-objects>
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
│       ├── qcom
│       ├── snap_release.xml
│       └── sync_snap_v2.sh
├── MPSS.HI.4.3.3.c6.2
│   └── modem_proc
│       ├── build
│       ├── datamodem
│       ├── mcfg
│       └── rf
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   ├── config
│   │   ├── core_qupv3fw
│   │   ├── dataipa_gsifw
│   │   └── sectoolsv2
│   └── contents.xml
├── SAIL.SI.1.0
│   └── sail_proc
│       ├── BSP
│       └── build
├── TZ.APPS.1.0
│   ├── qtee_tas
│   │   ├── apps
│   │   ├── build
│   │   └── sdk
│   ├── ta_size_info_kodiak.csv
│   └── ta_size_info_lemans.csv
├── TZ.XF.5.0
│   ├── BuildProducts.txt
│   ├── HY22
│   │   └── trustzone_images
│   └── trustzone_images
│       ├── build
│       ├── core
│       ├── scons_dep_trew_EACAANAA_b11b39829.txt
│       ├── sectools
│       ├── signed_images.json
│       ├── ssg
│       └── tools
├── WLAN.HSP.1.1
│   └── wlan_proc
│       ├── build
│       ├── config
│       ├── core
│       ├── scripts
│       ├── tools
│       └── wlan
└── WLAN.MSL.2.0.c2
    └── wlan_proc
        ├── build
        ├── scripts
        ├── tools
        └── wlan
```

Images directory structure after successful build

- The following figure shows the Images directory after a successful build:

```
├── aop.mbn
├── cpucp.elf
├── devcfg.mbn
├── dtb.bin
├── efi.bin
├── gpt_backup0.bin
├── gpt_backup1.bin
├── gpt_backup2.bin
├── gpt_backup3.bin
├── gpt_backup4.bin
├── gpt_backup5.bin
├── gpt_main0.bin
├── gpt_main1.bin
├── gpt_main2.bin
├── gpt_main3.bin
├── gpt_main4.bin
├── gpt_main5.bin
├── hypvm.mbn
├── Image
├── imagefv.elf
├── kernel-modules.tgz
├── multi_image.mbn
├── patch0.xml
├── patch1.xml
├── patch2.xml
├── patch3.xml
├── patch4.xml
├── patch5.xml
├── prog_firehose_ddr.elf
├── prog_firehose_lite.elf
├── qdl
├── qupv3fw.elf
├── rawprogram0.xml
├── rawprogram1.xml
├── rawprogram2.xml
├── rawprogram3.xml
├── rawprogram4.xml
├── rawprogram5.xml
├── shrm.elf
├── system.img
├── tools.fv
├── tz.mbn
├── uefi.elf
├── uefi_sec.mbn
├── vmlinux
├── xbl_config.elf
├── xbl.elf
├── XblRamdump.elf
└── zeros_5sectors.bin
```

The following table describes the files in the images directory:

Filename	Description
.mbn and *.elf	Boot critical images
gpt_main*.bin	GUID partition table binaries for the primary partition table
gpt_backup*.bin	GUID partition table binaries for the secondary partition table
system.img	Rootfs image
rawprogram*.xml	Image lun and start sector lba values
efi.bin	EFI system partition image. For more information, see Qualcomm Linux Yocto Guide.
qdl	Flashing tool binary
dtb.bin	Binary image that bundles all DTB binaries generated during build
vmlinux	Compile kernel elf binary
Image	Linux kernel ARM64 boot executable image

10.2 Related documents

Document title	Document number
Qualcomm Technologies, Inc.	
Qualcomm Linux Yocto Guide	80-70018-27
Qualcomm Linux Release Notes	RNO-250403001134
Qualcomm Linux Kernel Guide	80-70018-3
RB3 Gen 2 Quick Start Guide	80-70018-253
Qualcomm Intelligent Multimedia Product (QIMP) SDK Quick Start Guide	80-70018-51
Qualcomm® Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Quick Start Guide	80-70018-265
Qualcomm Linux Virtual Machine Setup Guide	80-70018-41
Qualcomm Software Center User Guide	80-72780-2
QCS615.LE.1.0 Software User Guide	80-70018-75
Resources	
Qualcomm manifest	https://github.com/quic-yocto/qcom-manifest
Yocto central download directory	https://docs.yoctoproject.org/4.0.16/singleindex.html#term-DL_DIR

Document title	Document number
CodeLinaro patch file	https://artifacts.codelinaro.org/artifactory/codelinaro-le/0001-fetch2-git-Add-verbose-logging-support.patch
meta-qcom-realtime	https://github.com/quic-yocto/meta-qcom-realtime

10.3 Acronyms and terms

Acronym or term	Definition
aDSP	Advanced digital signal processor
cDSP	Compute digital signal processor
ADB	Android debug bridge
AOP	Always on processor
BSP	Board support package
CDT	Configuration data table
CLI	Command-line interface
EDL	Emergency download
EFI	Extensible firmware interface
LE	Linux embedded
LiteRT	Lite Runtime
PCAT	Product configuration assistant tool
QDL	Qualcomm Device Loader
QIMP	Qualcomm Intelligent Multimedia Product
QIRF	Qualcomm Intelligent Robotics Function
QIRP	Qualcomm Intelligent Robotics Product
QLI	Qualcomm Linux
QNN	Qualcomm® Neural Network
QPM	Qualcomm Package Manager
QSC	Qualcomm Software Center
QTEE	Qualcomm Trusted Execution Environment
QUD	Qualcomm USB drivers
SAIL	Safety Island
SDK	Software development kit
WSL	Windows Subsystem for Linux

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.