Qualcomm Technologies, Inc.

Qualcomm Linux Graphics Guide

80-70018-19 AB

May 9, 2025

Contents

1 Graphics overview

The Qualcomm® Adreno™ GPU powers the graphics subsystem and offers the following capabilities:

- Serves as a three-dimensional (3D) graphics accelerator with 64-bit addressing.
- Incorporates the graphics memory (GMEM) as a dedicated memory for the graphics subsystem and facilitates Fast Z, color, and stencil rendering.
- Supports Qualcomm® Universal Bandwidth Compression, enhancing the overall performance.

The Adreno GPU supports the following graphics and compute APIs:

- **OpenGL ES**: 3.0, 3.1, 3.2
- **OpenCL**: 1.2 FP, 2.0 FP, 3.0 FP
- **EGL**: 1.5
- **Vulkan**: 1.1

Note: See Hardware SoCs that are supported on Qualcomm® Linux®.

1.1 Graphics subsystem architecture

The graphics architecture consists of Qualcomm and open-source components, as shown in the following figure.

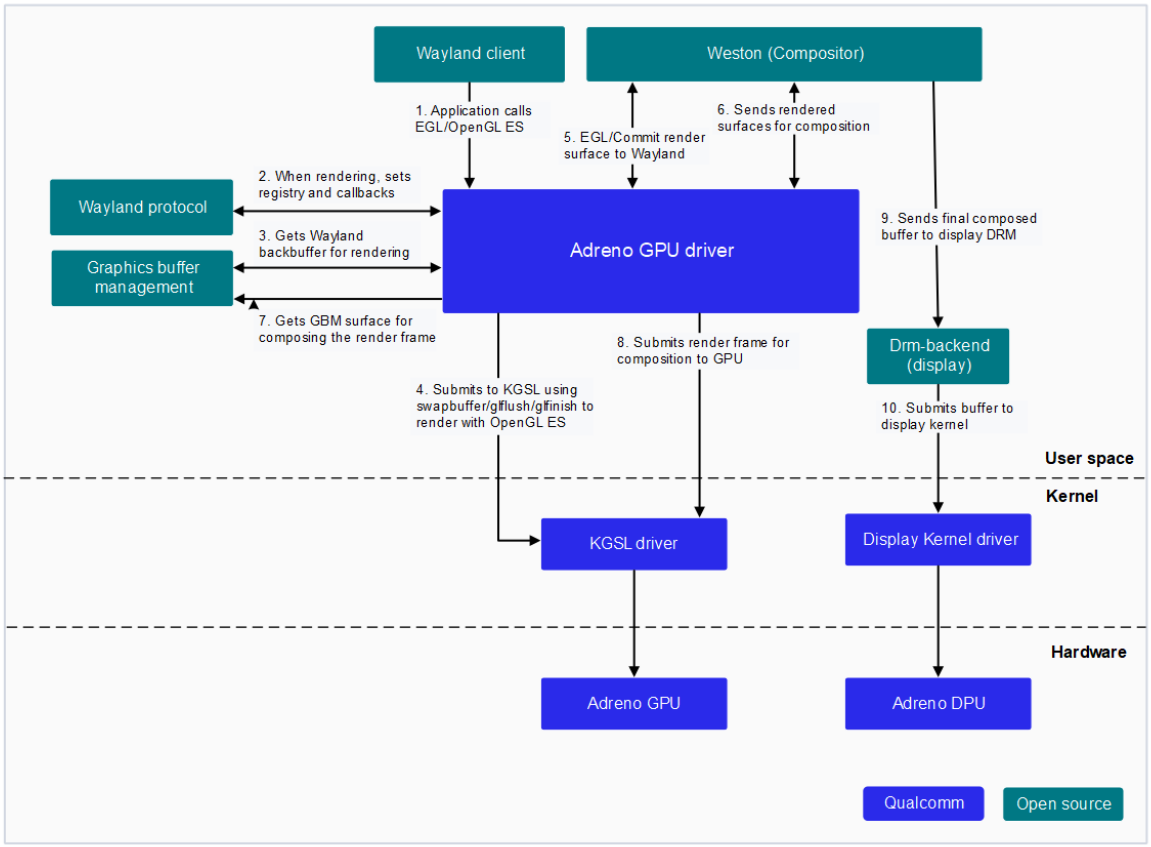

Figure : Graphics subsystem architecture

Adreno GPU driver

The Qualcomm Adreno GPU driver is a collection of user-mode precompiled libraries and firmware. The Adreno GPU driver can't be modified and must be used as-is.

Note: To use all the features supported by the Adreno GPU, ensure that the files listed in the Table: GPU firmware files and Table: GPU driver libraries are present on the device in the `/lib/firmware/` and `/usr/lib/` directories respectively.

Adreno GPU firmware

The following table lists the firmware files of the Adreno GPU hardware.

Table : GPU firmware files

Description	QCS5430/QCS6490 firmware	QCS8275 firmware	QCS9075 firmware
Microcode for command processor	a660_sqe.fw	a650_sqe.fw	a660_sqe.fw
Microcode for secure content support	a660_zap	a623_zap	a663_zap
Microcode for graphics management unit (GMU)	a660_gmu.bin	a663_gmu.bin	a663_gmu.bin

Kernel Graphics Support Layer (KGSL)

KGSL serves as the kernel-mode driver for the Adreno GPU. The primary function of the KGSL driver is to submit commands generated by the Adreno user-mode driver to the Adreno GPU for processing. Additionally, the KGSL driver communicates with the GMU to ensure proper state management.

The following table lists the supported GPU libraries and KGSL driver.

Table : GPU libraries and KGSL driver

Adreno GPU libraries

Library	Description
libCB.so	Driver for GPU computing
libEGL_adreno.so	EGL 1.x driver
libeglSubDriverWayland.so	EGL driver for Wayland subsystem
libadreno_utils.so	Utility library for Adreno GPU driver
libgsl.so	GSL library
libq3dtools_ adreno.so	Adreno profiler support layer
libGLESv1_CM_ adreno.so	OpenGL ES 1.x driver
libGLESv2_ adreno.so	OpenGL ES 2.x/3.x driver
libq3dtools_esx.so	OpenGL ES profiler support layer
libOpenCL.so	OpenCL library
libOpenCL_ adreno.so	OpenCL library for Adreno GPU
libllvm-qcom.so	OpenCL compiler library
libvulkan_adreno.so	Vulkan 1.x driver
libllvm-glnext.so	OpenGL ES and Vulkan program binary loader
libllvm-qgl.so	OpenGL ES and Vulkan core compiler library

KGSL driver

Linux kernel	6.6
Source code path	`<root_dir>/source/graphics-kernel/`
GPU DTSI files	
QCS5430	• `<root_dir>/source/graphics-devicetree/gpu/qcs5430-graphics.dtsi` • `<root_dir>/source/graphics-devicetree/gpu/qcs5430-graphics-pwrlevels.dtsi`
QCS6490	• `<root_dir>/source/graphics-devicetree/gpu/qcm6490-graphics.dtsi` • `<root_dir>/source/graphics-devicetree/gpu/qcm6490-graphics-pwrlevels.dtsi`
QCS8275	`<root_dir>/source/graphics-devicetree/gpu/qcs8300-graphics.dtsi`
QCS9075	`<root_dir>/source/graphics-devicetree/gpu/qcs9075-graphics.dtsi`

1.2 Adreno GPU specifications

The following table lists the specifications of the Adreno GPU.

Chipset	QCS5430	QCS6490	QCS8275	QCS9075
GPU	Adreno GPU 642L	Adreno GPU 643	Adreno GPU 623	Adreno GPU 663
GPU clock	315 MHz	812 MHz	877 MHz	800 MHz
On-Chip memory (GMEM)	512 kB	512 kB	512 kB	1.5 MB

To find the GPU information, run the following command using SSH on the device:

```
cat sys/class/kgsl/kgsl-3d0/gpu_model
```

For general information about the Adreno GPU, see Game Developer Guide.

2 Run sample applications

The following workflow shows how to get started with a graphics application on a Qualcomm device.

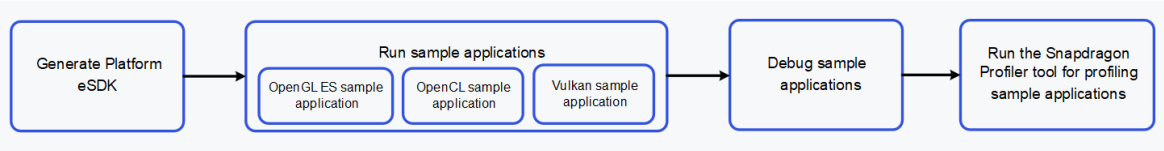

Figure : Workflow of graphics applications

2.1 Compile and run OpenGL ES-based applications

The Platform eSDK provides an OpenGL ES-based sample application called glmark2. This application renders a variety of scenes using OpenGL ES API.

To compile and run OpenGL ES-based applications, you must have an understanding of the OpenGL ES API.

Prerequisites

- Install the Platform eSDK using either of the following methods:
  - Using Qualcomm release archive
  - Manually compile the Qualcomm Linux SDK
- Establish an SSH connection from the Linux host computer to the device. For instructions, see Sign in using SSH.
- Ensure that the Weston application is running. If not, start the Weston application by running the following commands:

```
mount -o remount,rw /
export XDG_RUNTIME_DIR=/dev/socket/weston && mkdir -p
$XDG_RUNTIME_DIR
weston --idle-time=0 --continue-without-input
```

For more information about the Weston application, see Qualcomm Linux Display Guide.

For information about the OpenGL ES extensions supported on Qualcomm Linux devices, see EGL extensions.

Compile and run the glmark2 application

1. To set the SDK environment on the Linux host computer, run the following command:

```
source environment-setup-armv8-2a-qcom-linux
```

2. To compile and generate a binary, run the following commands:

```
devtool modify glmark2
devtool build glmark2
```

   If you encounter any umask error, set the umask to 022.

   After successful compilation, you can find the output files in the `<path-to-installed-eSDK>/workspace/sources/glmark2/oe-workdir/image/usr/bin` directory.

3. To push the binary to the device, run the following commands:

```
scp -r <path-to-installed-Platform-eSDK>/workspace/sources/
glmark2/oe-workdir/image/usr/bin/glmark2-es2-wayland root@[IP-
address-of-device]:/tmp
```

```
scp -r <path-to-installed-Platform-eSDK>/workspace/sources/
glmark2/oe-workdir/image/usr/share/glmark2 root@[IP-address-of-
device]:/tmp
```

4. To run the application on the device, open the SSH terminal using the IP address of the device and run the following commands:

```
export XDG_RUNTIME_DIR=/dev/socket/weston && export WAYLAND_
DISPLAY=wayland-1
chmod 777 /tmp/glmark2-es2-wayland
chmod -R 777 /tmp/glmark2
cd /tmp
./glmark2-es2-wayland --data-path /tmp/glmark2 -b jellyfish
```

Note: To run the sample application from the UART shell, remount the file system using the following command:

```
mount -o remount,rw /
```

2.2 Compile and run OpenCL-based applications

The Adreno OpenCL SDK has many sample applications. The steps to compile and run them are the same. You can develop several applications using the Adreno OpenCL SDK. However, consider the GPU capabilities before you compile and run applications.

Note: Currently, Qualcomm Linux supports OpenCL v2.1.

The steps to run the hello world application are as follows:

Prerequisites

- Install the Platform eSDK using either of the following methods:
  - Using Qualcomm release archive
  - Manually compile the Qualcomm Linux SDK
- Establish an SSH connection from the Linux host computer to the device. For instructions, see Sign in using SSH.
- Download the Adreno OpenCL SDK.

Note: You must be a registered user to download the Adreno OpenCL SDK.

For information about the OpenCL extensions supported on Qualcomm Linux devices, see OpenCL extensions.

Compile and run the hello world sample application

1. To compile the application, run the following commands on the Linux host computer:

```
cd opencl-sdk
```

```
source <path-to-installed-Platform-eSDK>/environment-setup-
armv8-2a-qcom-linux
```

```
cmake -B build -DCLSDK_OPENCL_LIBRARY=$OECORE_TARGET_SYSROOT/
usr/lib/libOpenCL.so -DCLSDK_DMABUFHEAP_LIBRARY=$OECORE_TARGET_
SYSROOT/usr/lib/libdmabufheap.so -DOPEN_EMBEDDED=1
cmake --build build
```

After you successfully compile the code, the system generates binary files in the `/build` directory.

2. To run the application, open the SSH terminal using the IP address of the device and run the following commands:

```
mkdir -p /opt/data
mkdir -p /opt/data/opencl
```

Note: To run the sample application from the UART shell, remount the file system using the following command:

```
mount -o remount,rw /
```

3. To push the binary to the device, run the following commands:

```
scp -r <opencl-sdk>/build/* root@[IP-address-of-the device]:/
opt/data/opencl/
```

```
scp -r <opencl-sdk>/example_images/* root@[IP-address-of-the
device]:/opt/data/opencl/
```

4. To start the application, run the following commands on the device using the SSH terminal:

```
cd /opt/data/opencl/
mkdir out
chmod 777 ./*
echo "run hello world 2.0 opencl sdk" > hello_world_input.txt
touch out/hello_world_output.txt
cat out/hello_world_output.txt
./cl_sdk_hello_world hello_world_input.txt out/hello_world_
output.txt
cat out/hello_world_output.txt
```

2.3 Compile and run Vulkan-based applications

The Adreno SDK has many sample applications based on Vulkan. The steps to compile and run any Vulkan-based application are the same. You can develop Vulkan-based applications using the Adreno SDK. To compile and run these applications, refer to the following procedures. As an example, this section describes the steps to run Sascha Willems and Khronos Vulkan-based applications.

Compile and run Sascha Willems Vulkan-based applications

You can find the Sascha Willems Vulkan sample applications at https://github.com/SaschaWillems/Vulkan.git. The steps to compile and run any Sascha Willems Vulkan application are the same.

For information about the Vulkan extensions supported on Qualcomm Linux devices, see Vulkan extensions.

Prerequisites

- Install the Platform eSDK using either of the following methods:
  - Using Qualcomm release archive
  - Manually compile the Qualcomm Linux SDK
- Establish an SSH connection from the Linux host computer to the device. For instructions, see Sign in using SSH.

Steps to compile and run the triangle bin sample application

1. To compile the application, run the following commands on the Linux host computer:

```
git clone --recurse-submodules https://github.com/SaschaWillems/
Vulkan.git
cd Vulkan
```

```
source <path-to-installed-Platform-eSDK>/environment-setup-
armv8-2a-qcom-linux
```

```
cmake -G "Unix Makefiles" -Bbuild/linux -DUSE_WAYLAND_WSI=ON -
DRESOURCE_INSTALL_DIR="/tmp/"
cmake --build build/linux --config Release  -j$(nproc)
```

   After successful compilation, you can find the binary files in the `build/linux/bin` directory.

2. To push the binary to the device, run the following commands:

```
cd Vulkan
```

```
scp -r assets root@[IP-address-of-device]:/tmp/
```

```
scp -r shaders root@[IP-address-of-device]:/tmp/
```

```
scp -r build/linux/bin/triangle root@[IP-address-of-device]:/
tmp/
```

3. To run the application, open the SSH terminal using the IP address of the device and run the following commands:

```
cd /tmp
chmod -R 777 assets/
chmod -R 777 shaders/
chmod 777 triangle
export XDG_RUNTIME_DIR=/dev/socket/weston && export WAYLAND_
DISPLAY=wayland-1
./triangle
```

Compile and run Khronos Vulkan-based applications

You can find the Khronos Vulkan-based applications at https://github.com/KhronosGroup/Vulkan-Samples.git. The steps to compile and run any Khronos Vulkan-based application are the same.

For information about the Vulkan extensions supported on Qualcomm Linux devices, see Vulkan extensions.

Prerequisites

- Install the Platform eSDK using either of the following methods:
  - Using Qualcomm release archive
  - Manually compile the Qualcomm Linux SDK
- Establish an SSH connection from the Linux host computer to the device. For instructions, see Sign in using SSH.

Steps to compile and run the vulkan_samples application

1. To compile the application, run the following commands on the Linux host computer:

```
sudo apt-get install -y libwayland-dev
git clone --recurse-submodules https://github.com/KhronosGroup/
Vulkan-Samples.git
cd Vulkan-Samples/
```

```
source <path-to-installed-Platform-eSDK>/environment-setup-
armv8-2a-qcom-linux
```

```
cmake -G "Unix Makefiles" -Bbuild/linux -DCMAKE_BUILD_
TYPE=Release -DVKB_WSI_SELECTION=WAYLAND -DGLFW_BUILD_X11=OFF
```

If the compilation fails, do the following:

a. Open the `Vulkan-Samples/build/linux/CmakeCache.txt` file.

b. Change `GLFW_BUILD_X11` to `OFF`.

2. To generate the binary files, run the following command:

```
cmake --build build/linux --config Release --target vulkan_
samples -j$(nproc)
```

After successful compilation, you can find the binary files in the `Vulkan-Samples/build/linux/app/bin/Release/aarch64` directory.

3. To push the binary to the device, run the following commands:

```
cd Vulkan-Samples
```

```
scp -r assets root@[IP-address-of-device]:/tmp/
```

```
scp -r shaders root@[IP-address-of-device]:/tmp/
```

```
scp -r build/linux/app/bin/Release/aarch64/vulkan_samples
root@[IP-address-of-device]:/tmp/
```

4. To run the application, open the SSH terminal using the IP address of the device and run the following commands:

```
cd /tmp
chmod -R 777 assets/
chmod -R 777 shaders/
chmod 777 vulkan_samples
export XDG_RUNTIME_DIR=/dev/socket/weston && export WAYLAND_
DISPLAY=wayland-1
./vulkan_samples sample swapchain_images
```

Note: If you are facing issues compiling the application or running it, use the following commands to checkout the specific version. Later, repeat the steps from 2 to 4 and recompile the application and run.

```
git checkout b3cb3822e8896ab650c4310f2c5f66a101469e9e
git submodule sync
git submodule update
```

If you are unable to run these sample applications from the `/tmp` directory due to storage issues, try alternative directories such as `/etc`.

2.4 Compile and run X11-based applications

The X11 applications are graphical programs designed for the X Window system, a widely used display protocol on Linux and other Unix-like OS.

Prerequisites

- Install the Platform eSDK using either of the following methods:
  - Using Qualcomm release archive
  - Manually compile the Qualcomm Linux SDK
- Establish an SSH connection from the Linux host computer to the device. For instructions, see Sign in using SSH.
- Install X11-related binaries from the following Gitlab sites:
  - Install libXt (X Toolkit Intrinsics)

```
git clone https://gitlab.freedesktop.org/xorg/lib/libxt.
git
cd libxt
```

```
source <path-to-installed-Platform-eSDK>/environment-
setup-armv8-2a-qcom-linux
```

```
./autogen.sh --host=aarch64-linux-gnu --prefix=<path-to-
installed-Platform-eSDK>/tmp/sysroots/qcs6490-rb3gen2-
core-kit/usr
```

```
./configure --host=aarch64-linux-gnu --prefix=<path-to-
installed-Platform-eSDK>/tmp/sysroots/qcs6490-rb3gen2-
core-kit/usr
```

```
make
make install
cd src/.libs/
```

```
cp -r libX* <path-to-installed-Platform-eSDK>/tmp/
sysroots/qcs6490-rb3gen2-core-kit/usr/lib
```

```
cd ../../..
```

  - Install libxmu (Miscellaneous utility functions for X11)

```
git clone https://gitlab.freedesktop.org/xorg/lib/
libxmu.git
cd libxmu
```

```
source <path-to-installed-Platform-eSDK>/environment-
setup-armv8-2a-qcom-linux
```

```
./autogen.sh --host=aarch64-linux-gnu --prefix=<path-to-
installed-Platform-eSDK>/tmp/sysroots/qcs6490-rb3gen2-
core-kit/usr
```

```
./configure --host=aarch64-linux-gnu --prefix=<path-to-
installed-Platform-eSDK>/tmp/sysroots/qcs6490-rb3gen2-
core-kit/usr
```

```
make
make install
cd src/.libs/
```

```
cp -r libX* <path-to-installed-Platform-eSDK>/tmp/
sysroots/qcs6490-rb3gen2-core-kit/usr/lib
```

```
cd ../../..
```

– Install X development libraries for host computer

```
sudo apt-get install xorg-dev
```

Compile and run the X11-based application

1. To compile the Xeyes application, run the following commands:

```
git clone https://gitlab.freedesktop.org/xorg/app/xeyes.
git
cd xeyes
```

```
source <path-to-installed-Platform-eSDK>/environment-
setup-armv8-2a-qcom-linux
```

```
./autogen.sh --host=aarch64-linux-gnu
make
```

2. Open new SSH terminal using the IP address of the device and run the following commands:

```
killall weston
export XDG_RUNTIME_DIR=/dev/socket/weston && mkdir -p
$XDG_RUNTIME_DIR && weston --continue-without-input --
idle-time=0 --backend=drm-backend.so --xwayland
```

3. Open the original terminal where Xeyes has been compiled and run the following commands:

```
scp -r xeyes/ root@[IP-address-of-device]:/etc/data
```

```
scp -r <path-to-installed-Platform-eSDK>/tmp/sysroots/qcs6490-
rb3gen2-core-kit/usr/lib/ root@[IP-address-of-device]:/etc/data
```

4. To run the application on the device, open the SSH terminal using the IP address of the device and run the following commands:

```
. /etc/profile
export XDG_RUNTIME_DIR=/dev/socket/weston
mkdir --parents $XDG_RUNTIME_DIR
chmod 0700 $XDG_RUNTIME_DIR
export WAYLAND_DISPLAY=wayland-1
export DISPLAY=:0
export LD_LIBRARY_PATH=/etc/data/lib
cd /etc/data/xeyes
./xeyes
```

3 Snapdragon Profiler tool

Qualcomm provides Snapdragon® Profiler as a performance profiling tool for the Adreno GPU.

The Snapdragon Profiler tool allows you to examine various aspects of the Adreno GPU and profile performance issues. The Snapdragon Profiler offers the following features:

- Real-time and Trace modes for the OpenGL ES, Vulkan, and OpenCL applications
- Snapshot mode for the OpenGL ES and Vulkan applications

Download

Download the latest version of the Snapdragon Profiler.

Analyze graphics issues

For information about how to analyze graphics issues using the Snapdragon Profiler, see Snapdragon Profiler Documentation.

Note: On Qualcomm Linux devices, the Snapdragon Profiler supports:

- Profiling of only the GPU and doesn't support profiling of other components such as the CPU and DSP.
- Vulkan Snapshot capture without ShaderStats or Metrics values.

---

4 Debug issues

You can use the Snapdragon Profiler and kernel commands for performance profiling of the applications. This section describes how to collect information for profiling and debugging using the kernel commands, and how to collect logs.

Caution: Use these commands with caution because these commands may impact the power consumption of the device.

4.1 Capture Weston/Wayland logs

You can use the Wayland/Weston logs to verify whether the display driver is loaded on the device and to find the supported OpenGL ES extensions. To capture the Weston/Wayland debug logs, run the following command before launching the Weston or any other application:

```
export WAYLAND_DEBUG=server/client
```

4.2 Debug Kernel/KGSL

You can evaluate the performance of your application by adjusting the GPU to different frequency levels and setting the GPU or CPU to performance mode. Based on these evaluations, you can optimize the performance of your application.

Set CPU to performance mode

1. Remount the file system using the following command:

```
mount -o remount,rw /
```

2. To set all CPU cores, replace `cpux` with `cpu0`, `cpu1`, and `cpun`.

```
echo 1 > /sys/devices/system/cpu/[cpux]/online
```

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_
governors
```

Sample output

```
ondemand userspace performance schedutil
```

3. If any CPU clock is running at a low frequency, set the CPU to performance mode and verify the issue by running the following command:

```
echo performance > /sys/devices/system/cpu/[cpux]/cpufreq/
scaling_governor
```

4. To validate the trial settings, run the following commands:

```
cat /sys/devices/system/cpu/[cpux]/online (cpux : cpu0,cpu1 ...)
```

```
cat /sys/devices/system/cpu/[cpux]/cpufreq/scaling_governor(cpux
: cpu0, cpu1 ...)
```

Set GPU frequency

1. Remount the file system using the following command:

```
mount -o remount,rw /
```

2. For the power level, check the `available_frequencies` sys node.

```
cat /sys/class/kgsl/kgsl-3d0/devfreq/available_frequencies
```

Sample output

```
812000000 700000000 608000000 550000000 450000000 315000000
```

From the sample output, you can interpret GPU power levels as:

Frequency	812000000	700000000	608000000	550000000	450000000	315000000
Power level	0	1	2	3	4	5

3. Increment the GPU power level by one and perform the GPU power-level test by substituting `<level>` with 0, 1, 2, 3, 4, and 5 in the following commands:

```
echo <level> > /sys/class/kgsl/kgsl-3d0/min_pwrlevel
```

```
echo <level> > /sys/class/kgsl/kgsl-3d0/max_pwrlevel
```

Note: For QCS5430:

- The Adreno GPU frequency is fixed at 315 MHz.
- The power levels specified here aren't supported and the output of the `available_frequencies` command displays 315000000.

Set GPU to performance mode

- To set the GPU to performance mode, run the following commands:

```
mount -o remount,rw /
echo 0 > /sys/class/kgsl/kgsl-3d0/min_pwrlevel
echo 0 > /sys/class/kgsl/kgsl-3d0/max_pwrlevel
```

Note: These commands force the GPU to always run at the maximum frequency.

- To ensure that the GPU doesn't enter the low-power state even if the device is idle for 10000000 ms, run the following command:

```
mount -o remount,rw /
echo 10000000 > /sys/class/kgsl/kgsl-3d0/idle_timer
```

- To set the GPU to performance mode while ensuring that the GPU doesn't enter the

low-power state, run the following commands:

```
mount -o remount,rw /
echo 0 > /sys/class/kgsl/kgsl-3d0/min_pwrlevel
echo 0 > /sys/class/kgsl/kgsl-3d0/max_pwrlevel
echo 10000000 > /sys/class/kgsl/kgsl-3d0/idle_timer
```

Note: QCS5430 doesn't support performance mode because the Adreno GPU frequency is fixed at 315 MHz.

Check other KGSL settings

- To enable the KGSL clock, run the following command:

```
echo 1 > /sys/class/kgsl/kgsl-3d0/force_clk_on
```

- To enable the KGSL rail, run the following command:

```
echo 1 > /sys/class/kgsl/kgsl-3d0/force_rail_on
```

- To enable the KGSL bus, run the following command:

```
echo 1 > /sys/class/kgsl/kgsl-3d0/force_bus_on
```

4.3 Debug OpenGL ES-based applications

You can use the Snapdragon Profiler tool to capture snapshots and real-time traces of OpenGL ES applications. For more information, see Snapdragon Profiler.

4.4 Debug OpenCL-based applications

You can use the Snapdragon Profiler tool to capture snapshots and real-time traces of OpenCL applications.

- For more information, see Snapdragon Profiler.
- For performance profiling, see **Performance profiling** in Qualcomm® Snapdragon™ Mobile Platform OpenCL General Programming and Optimization Guide.

Note: Some sections in the Qualcomm® Snapdragon™ Mobile Platform OpenCL General Programming and Optimization Guide apply to Android platforms but not to Qualcomm Linux

devices. However, some profiling options are purely based on the OpenCL API and are therefore supported on the Qualcomm Linux devices. For example, the `clGetEventProfilingInfo` API.

4.5 Debug Vulkan-based applications

Use the Snapdragon Profiler tool to capture snapshots and real-time traces of Vulkan applications. For more information, see Snapdragon Profiler. Also, you can use validation layers to debug Vulkan applications.

Debug Vulkan application using validation layers

To debug a Vulkan application using validation layers, do the following on the Linux host computer:

1. To set the source environment, run the following command:

```
source <path-to-installed-platform-eSDK>/environment-setup-armv8-2a-qcom-linux
```

2. To download the validation layers, run the following command:

```
git clone https://github.com/KhronosGroup/Vulkan-ValidationLayers
```

3. To build the validation layers, run the following commands:

```
cd Vulkan-ValidationLayers
cmake -S . -B build -D UPDATE_DEPS=ON -D BUILD_WSI_XLIB_SUPPORT=OFF -D BUILD_TESTS=OFF -D CMAKE_BUILD_TYPE=Release -D BUILD_WSI_XCB_SUPPORT=OFF -D BUILD_WSI_WAYLAND_SUPPORT=ON
cmake --build build --config Debug
```

```
scp -r build/layers/libVkLayer_khronos_validation.so root@[IP-address-of-device]:/tmp/local/debug/vulkan/
```

```
scp -r build/layers/VkLayer_khronos_validation.json root@[IP-address-of-device]:/tmp/local/debug/vulkan/
```

```
export VK_LAYER_PATH=/tmp/local/debug/vulkan:$VK_LAYER_PATH
```

Note: If the `/tmp/local/debug/vulkan` directory isn't available, create a similar

directory.

4. To enable the validation layers on Vulkan applications, set the `VK_INSTANCE_LAYERS` environment variable as follows:

```
export VK_INSTANCE_LAYERS=VK_LAYER_KHRONOS_validation
```

5. To edit or append the settings in the `layers/vk_layer_settings.txt` file, run the following commands:

```
khronos_validation.debug_action = VK_DBG_LAYER_ACTION_LOG_MSG
khronos_validation.report_flags = error,warn,perf
khronos_validation.file = true
khronos_validation.log_filename = vk_validatedump.txt
```

6. To push the `layers/vk_layer_settings.txt` file to the device, run the following command:

```
scp -r vk_layer_settings.txt root@[IP-address-of-device]:/tmp/
local/debug/vulkan/
```

7. To dump the log, add path of the validation layers by running the following command:

```
export VK_LAYER_SETTINGS_PATH=/tmp/local/debug/vulkan/vk_layer_
settings.txt
```

   This command generates the dumps in the same directory where the Vulkan application binary exists.

Note: If you are facing issues compiling the application or running it, use the following commands to checkout the specific version. Later, repeat the steps from 3 to 7 and recompile the application and run.

```
git checkout -b <local_branch_name> --track remotes/origin/vulkan-
sdk-1.3.275

example: git checkout -b 13275 --track remotes/origin/vulkan-sdk-1.3.
275
```

To avoid data flush in the `tmp` directory, save the Vulkan application binary to another location. If you change the location of the binary, ensure that you update the environment variables and verify the changes after rebooting.

5 Supported extensions

Qualcomm Linux devices support the following extensions:

- OpenGL ES
- OpenCL
- Vulkan

5.1 EGL extensions

EGL client extensions	GL extensions	
• EGL_EXT_client_ extensions • EGL_KHR_client_get_ all_proc_addresses • EGL_EXT_platform_ base • EGL_KHR_platform_ android • EGL_KHR_platform_ wayland • EGL_KHR_platform_ gbm • EGL_KHR_platform_ x11 • EGL_KHR_image • EGL_KHR_image_base • EGL_QCOM_create_ image • EGL_KHR_lock_ surface • EGL_KHR_lock_ surface2 • EGL_KHR_lock_ surface3 • EGL_KHR_gl_texture_ 2D_image • EGL_KHR_gl_texture_ cubemap_image • EGL_KHR_gl_texture_ 3D_image • EGL_KHR_gl_ renderbuffer_image	• GL_OES_EGL_image • GL_OES_EGL_image_ external • GL_OES_EGL_sync GL_OES_vertex_half_ float • GL_OES_framebuffer_ object GL_OES_rgb8_ rgba8 • GL_OES_compressed_ ETC1_RGB8_texture • GL_AMD_compressed_ ATC_texture • GL_KHR_texture_ compression_astc_ldr • GL_KHR_texture_ compression_astc_hdr • GL_OES_texture_ compression_astc • GL_EXT_texture_ compression_s3tc • GL_EXT_texture_ compression_s3tc_srgb • GL_EXT_texture_ compression_rgtc • GL_EXT_texture_ compression_bptc • GL_OES_texture_npot • GL_EXT_texture_filter_ anisotropic • GL_EXT_texture_ format_BGRA8888 • GL_EXT_read_format_ bgra	• GL_EXT_copy_image • GL_EXT_geometry_ shader • GL_EXT_tessellation_ shader • GL_OES_texture_ stencil8 • GL_EXT_shader_io_ blocks • GL_OES_shader_ image_atomic • GL_OES_sample_ variables • GL_EXT_texture_ border_clamp • GL_EXT_EGL_image_ external_wrap_modes • GL_EXT_ multisampled_render_ to_texture • GL_EXT_ multisampled_render_ to_texture2 • GL_OES_shader_ multisample_ interpolation • GL_EXT_texture_cube_ map_array • GL_EXT_draw_buffers_ indexed • GL_EXT_gpu_shader5 • GL_EXT_robustness • GL_EXT_texture_buffer

EGL client extensions	GL extensions	
• EGL_ANDROID_blob_cache • EGL_KHR_create_context • EGL_KHR_surfaceless_context • EGL_KHR_create_context_no_error • EGL_KHR_get_all_proc_addresses • EGL_QCOM_lock_image2 • EGL_KHR_no_config_context • EGL_EXT_surface_SMPTE2086_metadata • EGL_EXT_image_dma_buf_import • EGL_EXT_image_dma_buf_import_modifiers • EGL_EXT_yuv_surface • EGL_IMG_context_priority • EGL_WL_bind_wayland_display • EGL_WL_create_wayland_buffer_from_image • EGL_ANDROID_native_fence_sync • EGL_EXT_create_context_robustness • EGL_KHR_fence_sync • EGL_KHR_wait_sync • EGL_KHR_mutable_render_buffer • EGL_KHR_partial_update • EGL_KHR_surfaceless_context	• GL_OES_texture_3D • GL_EXT_color_buffer_float • GL_EXT_color_buffer_half_float • GL_EXT_float_blend • GL_QCOM_alpha_test • GL_OES_depth24 • GL_OES_packed_depth_stencil • GL_OES_depth_texture • GL_OES_depth_texture_cube_map • GL_EXT_sRGB • GL_OES_texture_float • GL_OES_texture_float_linear • GL_OES_texture_half_float • GL_OES_texture_half_float_linear • GL_EXT_texture_type_2_10_10_10_REV • GL_EXT_texture_sRGB_decode • GL_EXT_texture_compression_astc_decode_mode • GL_EXT_texture_mirror_clamp_to_edge • GL_EXT_texture_format_sRGB_override • GL_OES_element_index_uint	• GL_OES_sample_shading • GL_OES_get_program_binary • GL_EXT_debug_label • GL_KHR_blend_equation_advanced • GL_KHR_blend_equation_advanced_coherent • GL_QCOM_tiled_rendering • GL_ANDROID_extension_pack_es31a • GL_EXT_primitive_bounding_box • GL_OES_standard_derivatives • GL_OES_vertex_array_object • GL_EXT_disjoint_timer_query • GL_KHR_debug • GL_EXT_YUV_target • GL_EXT_sRGB_write_control • GL_EXT_texture_norm16 • GL_EXT_discard_framebuffer • GL_OES_surfaceless_context • GL_OVR_multiview • GL_EXT_shader_framebuffer_fetch • GL_ARM_shader_framebuffer_fetch_depth_stencil • GL_OES_texture_storage_multisample_2d_array

EGL client extensions	GL extensions

5.2 OpenCL extensions

• cl_khr_icd, cl_img_egl_image • cl_khr_3d_image_writes • cl_khr_byte_addressable_store • cl_khr_depth_images • cl_khr_egl_event • cl_khr_egl_image • cl_khr_fp16 • cl_khr_gl_sharing • cl_khr_global_int32_base_atomics • cl_khr_global_int32_extended_atomics • cl_khr_image2d_from_buffer • cl_khr_local_int32_base_atomics	• cl_khr_local_int32_extended_atomics • cl_khr_mipmap_image • cl_khr_srgb_image_writes • cl_khr_subgroups • cl_qcom_accelerated_image_ops • cl_qcom_compressed_image • cl_qcom_compressed_yuv_image_read • cl_qcom_create_buffer_from_image • cl_qcom_dot_product8 • cl_qcom_ext_host_ptr • cl_qcom_ext_host_ptr_iocoherent	• cl_qcom_extended_query_image_info • cl_qcom_extract_image_plane • cl_qcom_dmabuf_host_ptr • cl_qcom_other_image • cl_qcom_perf_hint • cl_qcom_priority_hint • cl_qcom_protected_context • cl_qcom_recordable_queues • cl_qcom_reqd_sub_group_size • cl_qcom_subgroup_shuffle • cl_qcom_vector_image_op

5.3 Vulkan extensions

• VK_KHR_get_physical_device_properties2 • VK_KHR_surface • VK_KHR_external_semaphore_capabilities • VK_KHR_external_memory_capabilities • VK_KHR_device_group_creation • VK_EXT_debug_utils • VK_KHR_wayland_surface • VK_KHR_external_fence_capabilities • VK_KHR_get_surface_capabilities2 • VK_EXT_debug_report • VK_EXT_subgroup_size_control • VK_KHR_external_memory • VK_EXT_pipeline_creation_feedback • VK_KHR_shader_float16_int8 • VK_KHR_get_memory_requirements2 • VK_KHR_copy_commands2	• VK_KHR_shader_terminate_invocation • VK_QCOM_fragment_density_map_offset • VK_EXT_scalar_block_layout • VK_KHR_sampler_ycbcr_conversion • VK_EXT_vertex_attribute_divisor • VK_KHR_variable_pointers • VK_QCOM_multiview_per_view_viewports • VK_KHR_push_descriptor • VK_KHR_timeline_semaphore • VK_EXT_device_memory_report • VK_KHR_imageless_framebuffer • VK_KHR_device_group • VK_EXT_device_fault • VK_KHR_relaxed_block_layout • VK_KHR_external_fence • VK_KHR_shader_non_semantic_info	• VK_QCOM_tile_properties • VK_KHR_image_format_list • VK_EXT_external_memory_dma_buf • VK_EXT_sampler_filter_minmax • VK_KHR_16bit_storage • VK_KHR_pipeline_executable_properties • VK_EXT_shader_demote_to_helper_invocation • VK_QCOM_render_pass_transform • VK_KHR_create_renderpass2 • VK_EXT_transform_feedback • VK_EXT_blend_operation_advanced • VK_EXT_provoking_vertex • VK_QCOM_multiview_per_view_render_areas • VK_KHR_depth_stencil_resolve • VK_KHR_shader_float_controls • VK_EXT_texture_compression_astc_hdr

• VK_KHR_spirv_1_4 • VK_EXT_fragment_density_map • VK_KHR_external_semaphore_fd • VK_KHR_swapchain • VK_QCOM_render_pass_store_ops • VK_EXT_astc_decode_mode • VK_KHR_shared_presentable_image • VK_KHR_external_memory_fd • VK_QCOM_render_pass_shader_resolve • VK_KHR_maintenance1 • VK_KHR_maintenance2 • VK_KHR_maintenance3 • VK_KHR_separate_depth_stencil_layouts • VK_EXT_image_robustness • VK_KHR_buffer_device_address • VK_EXT_extended_dynamic_state • VK_EXT_queue_family_foreign • VK_KHR_bind_memory2 • VK_KHR_external_semaphore	• VK_EXT_shader_atomic_float • VK_EXT_custom_border_color • VK_EXT_host_query_reset • VK_EXT_index_type_uint8 • VK_KHR_multiview • VK_KHR_storage_buffer_storage_class • VK_EXT_image_drm_format_modifier • VK_EXT_fragment_density_map2 • VK_QCOM_rotated_copy_commands • VK_KHR_shader_subgroup_extended_types • VK_EXT_private_data • VK_EXT_pipeline_creation_cache_control • VK_EXT_robustness2 • VK_EXT_shader_module_identifier • VK_EXT_global_priority_query • VK_EXT_separate_stencil_usage • VK_EXT_vertex_input_dynamic_state • VK_IMG_filter_cubic • VK_EXT_filter_cubic	• VK_EXT_global_priority • VK_KHR_shader_draw_parameters • VK_KHR_vulkan_memory_model • VK_EXT_descriptor_indexing • VK_EXT_depth_clip_enable • VK_KHR_synchronization2 • VK_EXT_line_rasterization • VK_KHR_fragment_shading_rate • VK_KHR_descriptor_update_template • VK_KHR_draw_indirect_count • VK_KHR_driver_properties • VK_KHR_uniform_buffer_standard_layout • VK_KHR_dedicated_allocation • VK_EXT_primitive_topology_list_restart • VK_KHR_global_priority • VK_EXT_sample_locations • VK_KHR_sampler_mirror_clamp_to_edge • VK_KHR_external_fence_fd

6 References

6.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
Qualcomm Linux Build Guide	80-70018-254
Qualcomm Linux Display Guide	80-70018-18
Resources	
Snapdragon Profiler Documentation	
Game Developer Guide	
Qualcomm® Snapdragon™ Mobile Platform OpenCL General Programming and Optimization Guide	
https://www.khronos.org/opengl/	
https://www.khronos.org/opencl/	
https://docs.vulkan.org/guide/latest/vulkan_spec.html	

6.2 Acronyms and terms

Acronym	Definition
DPU	Display processing unit
GMEM	Graphics memory
GMU	Graphics management unit
KGSL	Kernel graphics support layer

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.