Qualcomm Linux Camera Guide

80-70018-17 AB

April 10, 2025

Contents

1 Camera documentation

Connect camera sensors, enable camera use cases, and learn about available advanced features.

1.1 Camera overview

Camera components

Review the components of the camera subsystem.

Prerequisites

Set up your device for baseline camera functionality.

1.2 Stream cameras

Set up the camera

Connect camera sensors to your reference hardware platform.

Choose a stream API

Review the available stream API options.

Stream with the GStreamer API

Use GStreamer APIs to stream cameras.

Stream with the V4L2 API

Use the the V4L2 framework to stream cameras.

1.3 Enhance camera output

Enable high dynamic range

Enable the staggered high dynamic range (SHDR) feature.

Enable EIS and LDC

Enable the electronic image stabilization (EIS) and lens distortion correction (LDC) features.

Enable defog

Enable the defog feature.

1.4 Troubleshoot camera

Troubleshoot sensor streaming

Troubleshoot camera issues.

2 Camera overview

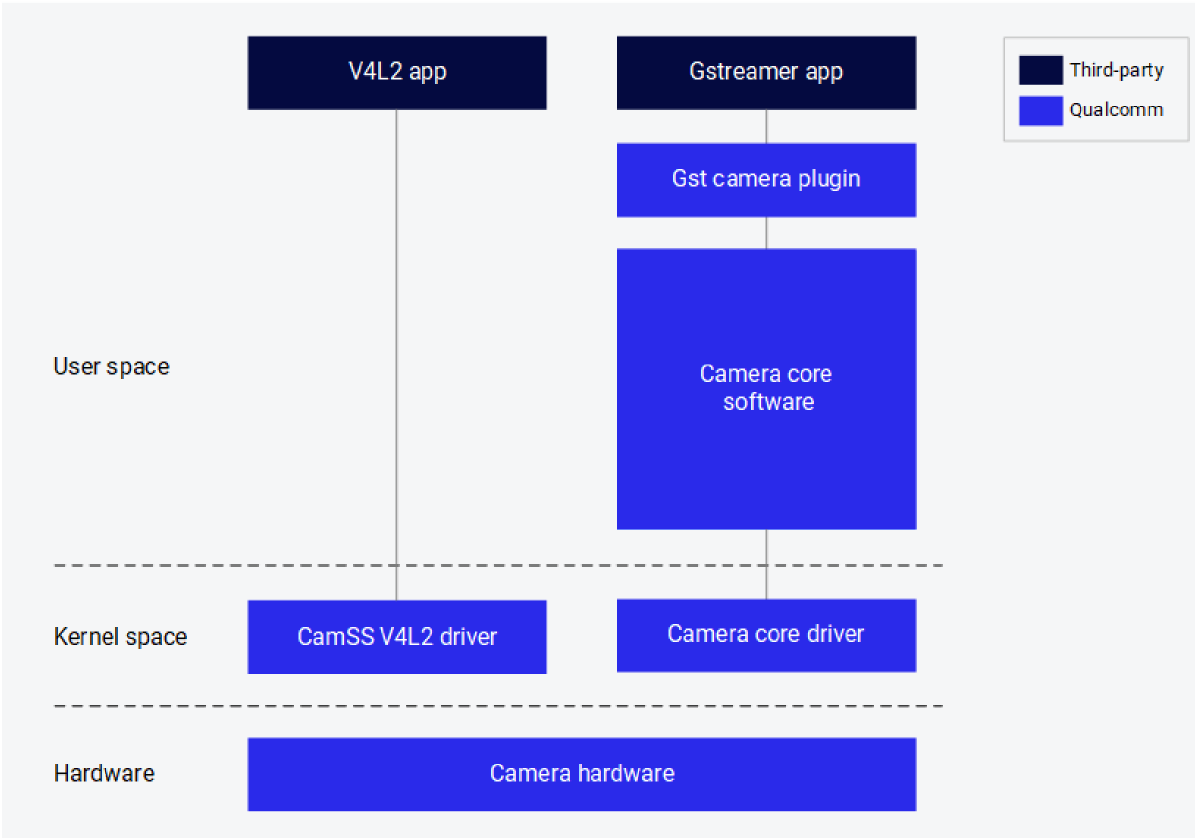
This document explains the camera subsystem that receives data through a MIPI CSI interface.

USB camera data is delivered though a USB interface, and the camera subsystem doesn't involve USB camera data transaction. See USB camera configuration for USB camera usage.

Network camera data is delivered though a network interface, and the camera subsystem doesn't involve network camera data transaction. The device can receive network camera data through the GStreamer rtspsrc plugin.

2.1 Camera components

The following diagram shows the components of Qualcomm Camera.



The following components are provided by Qualcomm:

Component	Description
Gst camera plugin (qtiqmmfsrc)	GStreamer plugin for Qualcomm's camera subsystem
Camera Core Software	Qualcomm's proprietary camera software that provides the interface to develop camera sensor drivers, camera tuning, and custom software nodes
Camera Core Driver	Qualcomm camera subsystem driver in the downstream Linux kernel
CamSS V4L2 driver	Qualcomm camera subsystem driver in the upstream Linux kernel

Qualcomm provides a GStreamer plug-in to enable application developers to interface with the Qualcomm camera subsystem. See GStreamer-camera-application for application development.

Qualcomm provides an interface for camera sensor driver developers and IQ tuning engineers to develop their own sensor drivers and perform custom IQ tuning. See Camera Sensor Driver Development and Tuning in the Camera Addendum document.

Access to the meta- qcom-extras layer is required for camera sensor driver development and custom IQ tuning. For the access level, refer to Mapping access levels.

2.2 Prerequisites

- Set up your infrastructure as described in the Qualcomm Linux Build Guide
- Flash the latest software release to the development board
- Set up SSH connection:
  1. Enable SSH in Permissive mode by performing the steps mentioned in Use SSH.
  2. Connect to the device by running the following command:

     ```
     ssh root@<device_IP_address>
     ```

     For example, if the IP address of the device is 10.92.160.222, run the following command:

     ```
     ssh root@10.92.160.222
     ```

3 Stream cameras

This page describes how to connect camera sensors to your reference hardware platform and provides information about available APIs.

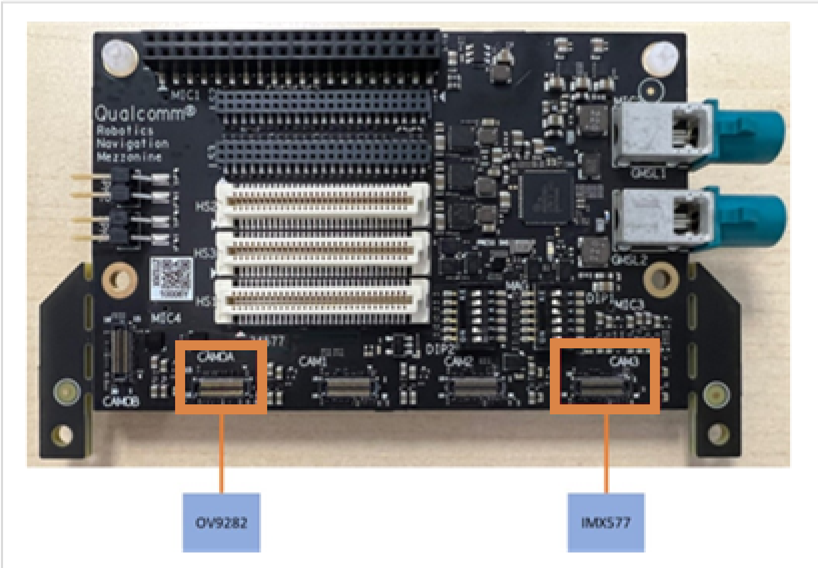
3.1 Set up the camera

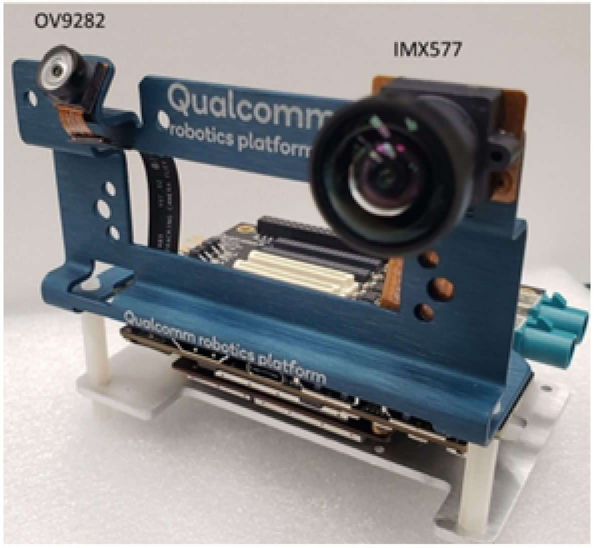
QCS6490

RB3G2 MIPI CSI connection on vision mezzanine board

OV9282 and IMX577 are the default camera sensors.

Connect the OV9282 module to CAM0A and the IMX577 module to CAM3 port as shown:





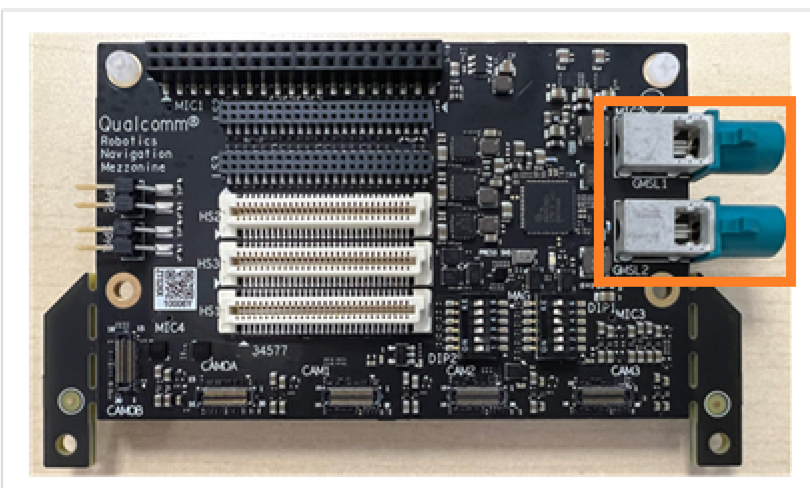
SW2300 DIP switch on interposer board

To use the CAM3 slot of the vision mezzanine board, switch 1 of the SW2300 DIP switch on the interposer board must be set to OFF. This switch is set to OFF by default.

To use the CAM1 and CAM2 slots of the interposer board:

RB2G2 GMSL connection on vision mezzanine board

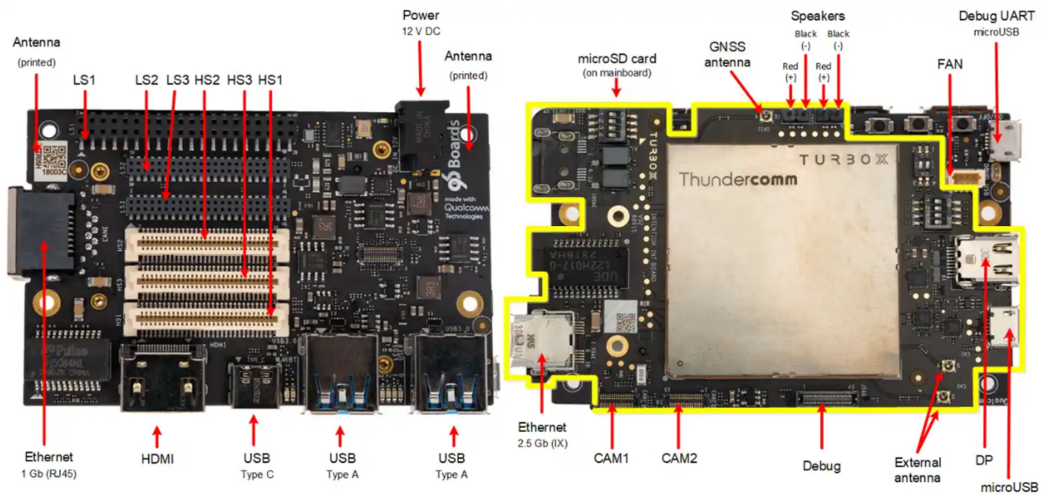
There are two GMSL slots (GMSL1 and GMSL2) on the mezzanine board of Qualcomm's RB3G2 Vision Kit platform. Two GMSL cameras can be connected to these slots.



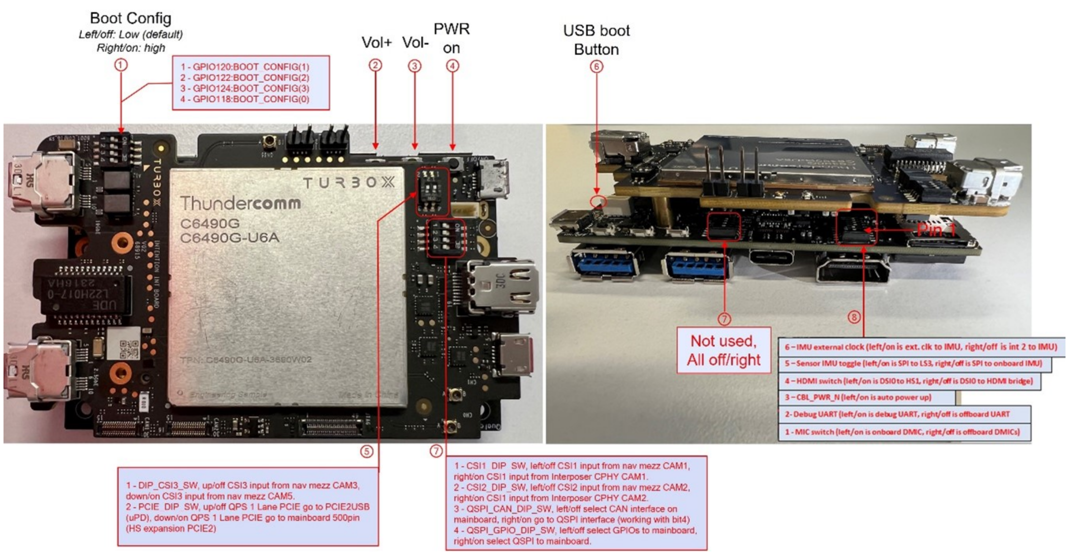
The GMSL1 slot can be used without any hardware setting. To use the GMSL2 slot, switch 1 of the SW2300DIP switch on the interposer board must be set to ON. This switch is set to OFF by default. See SW2300 Dip switch on interposer board.

RB3G2 MIPI CSI camera on interposer board

Connect OV9282 module to CAM1 and IMX577 module to CAM2 on the interposer board.



Set CSI1_DIP_SW and CSI2_DIP_SW to ON as shown in the right side image). As shown in #7 of the following image, the switch is set OFF by default. Change it to ON to use CAM/CAM2 on the Core Kit.



Supported resolutions and features

The following table shows the supported resolutions each camera module.

Resolution	Aspect Ratio	IMX577 (CAM3)	OV9282 (CAM0A)	AR0231 (GMSL)
4000 x 3000	4:3	Yes	No	No
3840 x 2160	16:9	Yes	No	No
2976 x 2976	1:1	Yes	No	No
2592 x 1940	4:3	Yes	No	No
2048 x 1536	4:3	Yes	No	No
1920 x 1440	4:3	Yes	No	No
1928 x 1208	16:10	Yes	No	Yes
1920 x 1080	16:9	Yes	No	Yes
1440 x 1080	4:3	Yes	No	Yes
1280 x 720	16:9	Yes	Yes	Yes
1024 x 768	4:3	Yes	Yes	Yes
800 x 600	4:3	Yes	Yes	Yes
640 x 480	4:3	Yes	Yes	Yes
640 x 360	16:9	Yes	Yes	Yes
320 x 240	4:3	Yes	Yes	Yes

The following table shows the supported features of each camera module.

Feature	IMX577 (CAM3)	OV9282 (CAM0A)	AR0231 (GMSL)
SHDR	Yes	No	No
LDC	Yes	No	No
EIS	Yes	No	No

QCS9075

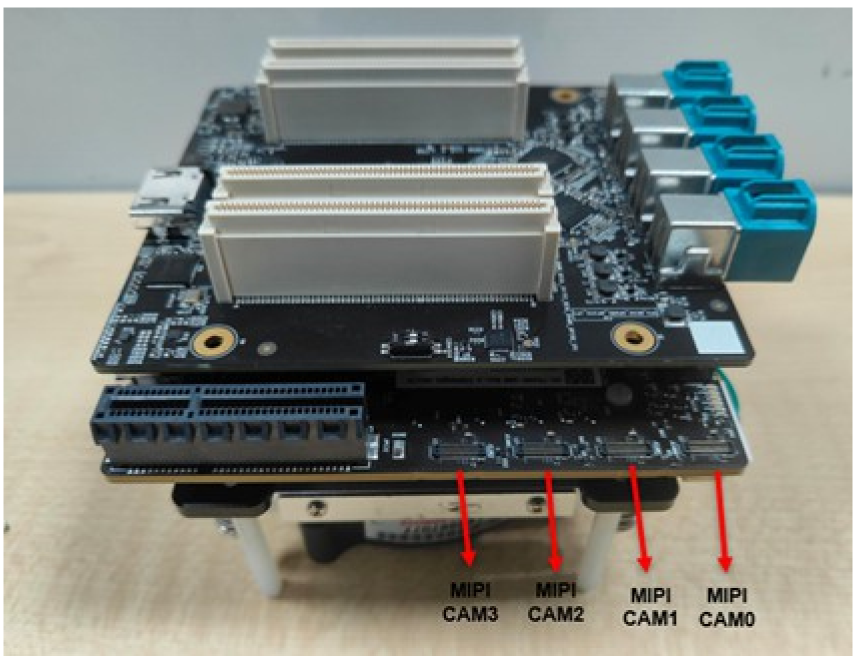
This section explains how to connect camera sensors on the QCS9075 reference hardware platform RB8 device.

Connect MIPI CSI cameras on RB8

RB8 will be offered in one kit version called the core kit. This core kit supports only MIPI CSI cameras. The GMSL cameras are supported using a separate add-on GMSL mezzanine board that needs to be connected to core kit.

In the RB8 EVT (core kit + GMSL mezzanine) hardware, there are four MIPI CSI (C/D-PHY) connectors present on the RB8 core kit/main board and four GMSL ports (0 to 3) present on the GMSL mezzanine board.

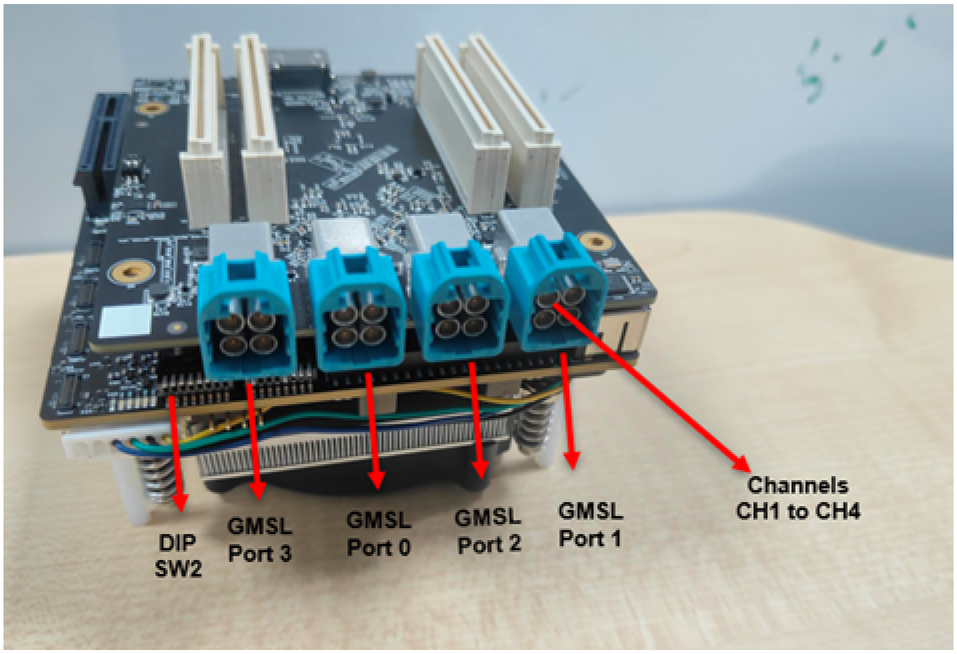
The following diagram shows the MIPI connectors on the RB8 core kit/main board.



Connect GMSL cameras on RB8

There are four GMSL ports (0 to 3) present on the GMSL mezzanine board. Each GMSL port connects with a MAX96724 quad GMSL deserializer. Each deserializer is connected to one CSI.

The following diagram shows the GMSL ports on the RB8 device. Note that the GMSL port numbering is not in sequence - It is defined based on the CSI index to which a GMSL port is connected.



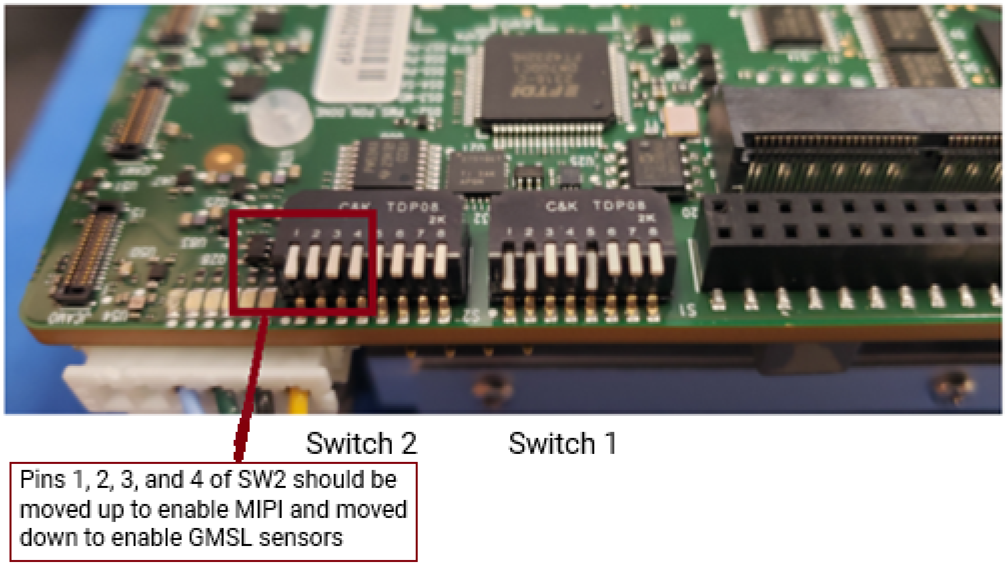
Note: In the current release, three GMSL ports (0, 2, and 3) are supported to connect GMSL cameras. Port 1 is not enabled due to a hardware issue.

- Each GMSL port contains four slots (1 to 4, also referred to as channels) and supports connection to four GMSL cameras on a single port
- With the existing software configuration in this release, a single GMSL port supports connection to only one GMSL camera on any of its four slots. GMSL Port-0 supports single Bayer camera, and Port-2 and Port-3 each support a single YUV camera. You can use any slot on a port to connect a GMSL camera.
- The current release supports OX03f10 Bayer GMSL and OX03f10 YUV GMSL cameras. You can connect one OX03F10 Bayer GMSL on GMSL Port-0 and two OX03F10 YUV GMSL cameras each on Port-2 and Port-3.

Set DIP switches

QCS9075 has four CSI PHYs. Each CSI PHY is routed to connect to either a MIPI camera port or a GMSL camera port controlled using DIP switch SW2 present on the core kit/main board.

The following diagram shows DIP switch SW2 present on the core kit/main board.



The following table explains the required DIP switch SW2 settings needed to use MIPI and GMSL camera ports.

SWITCH	OFF (default from factory)	ON	Connection when on	Connection when off (default from factory)
SW2 - 1	HIGH	LOW	CSI0 connected to GMSL mezzanine	CSI0 connected to main board
SW2 - 2	HIGH	LOW	CSI1 connected to GMSL mezzanine	CSI1 connected to main board
SW2 - 3	HIGH	LOW	CSI2 connected to GMSL mezzanine	CSI2 connected to main board

SW2 - 4	HIGH	LOW	CSI3 connected to GMSL mezzanine	CSI3 connected to main board

Supported resolutions and features

The following table shows the supported resolutions each camera module on the RB8 platform.

Resolution	Aspect Ratio	0X3F10 Bayer GMSL	0X3F10 YUV GMSL	OV9282 (CAM0A)
1920 x 1536	5:4	No	Yes	No
1920 x 1440	4:3	No	Yes	No
1928 x 1208	16:10	No	Yes	No
1920 x 1080	16:9	No	Yes	No
1824 x 1536	19:16	Yes	Yes	No
1440 x 1080	4:3	Yes	Yes	No
1280 x 720	16:9	Yes	Yes	Yes
1024 x 768	4:3	Yes	Yes	Yes
800 x 600	4:3	Yes	Yes	Yes
640 x 480	4:3	Yes	Yes	Yes
640 x 360	16:9	Yes	Yes	Yes
320 x 240	4:3	Yes	Yes	Yes

Advanced features such as SHDR, LDC, and EIS are not supported on QCS9075.

Concurrent camera support on RB8

The following tables explain the concurrent camera use cases which are supported in the current release.

- RB8 MIPI sensor

Sensor	CAM0	CAM1	CAM2	CAM3	Mode	Note
MIPI OV9282	TRUE	TRUE	TRUE	TRUE	Independent sensor testing	Concurrency testing of OV9282
MIPI OV9282	TRUE	TRUE	FALSE	FALSE	Concurrency testing of OV9282	A maximum of 2 MIPI sensor concurrency is supported

- RB8 GMSL sensor

Group	Sensor	Ch1	Ch2	Ch3	Ch4	Note
Des0	0x3F10 GMSL Bayer	TRUE	TRUE	TRUE	TRUE	A maximum of one sensor can be connected per deserializer
Des1	NA	FALSE	FALSE	FALSE	FALSE	Hardware issue can not be validated
Des2	0x3F10 GMSL YUV	TRUE	TRUE	TRUE	TRUE	A maximum of one sensor can be connected per deserializer
Des3	0x3F10 GMSL YUV	TRUE	TRUE	TRUE	TRUE	A maximum of one sensor can be connected per deserializer

- RB8 GMSL sensor + MIPI sensor

Group	Sensor	Ch1	Ch2	Ch3	Ch4	Note
CAM0	MIPI OV9282	TRUE	TRUE	TRUE	TRUE	
CAM1	MIPI OV9282	TRUE	TRUE	TRUE	TRUE	
Des2	0x3F10 GMSL YUV	TRUE	TRUE	TRUE	TRUE	A maximum of one sensor can be connected per deserializer
Des3	0x3F10 GMSL YUV	TRUE	TRUE	TRUE	TRUE	A maximum of one sensor can be connected per deserializer

QCS8275

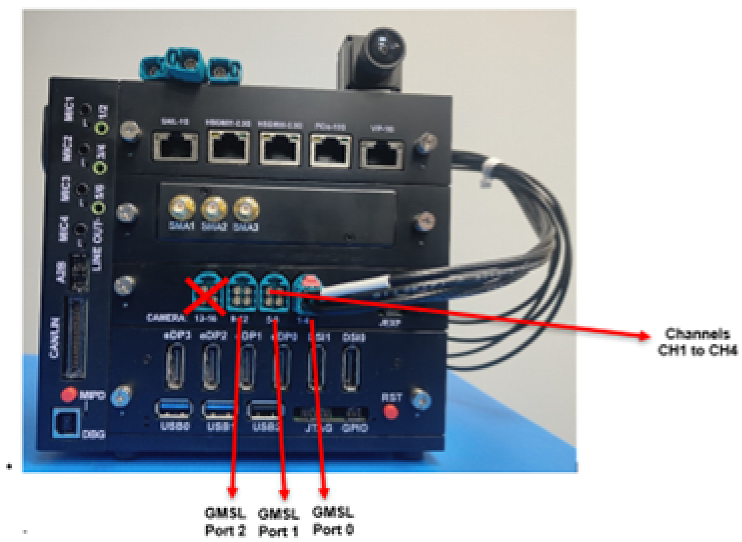
This section explains how to connect camera modules on the QCS8275 reference platform Ride SX hardware.

Connect GMSL camera module on Ride SX hardware

- In the QCS8275 reference platform Ride SX hardware, four GMSL ports (0 to 3) are present. Three GMSL ports (0 to 2) are supported to connect GMSL cameras. Each GMSL port connects with MAX96724 quad GMSL deserializer. Port 3 is a dummy port.
- There are no MIPI CSI camera ports available on this hardware.
- Each GMSL port contains four slots (1 to 4, also referred to as channels) and supports connection of four GMSL cameras to a single port.

- With the existing software configuration in this release, a single GMSL port supports connection to only one GMSL camera on any of its four slots. GMSL Port-0 supports single Bayer camera, and Port-1 and Port-2 each supports single YUV camera. Any slot can be used on these ports.
- The current release supports OX03f10 Bayer GMSL and OX03f10 YUV GMSL cameras. You can connect one OX03F10 Bayer GMSL on GMSL Port-0 and two OX03F10 YUV GMSL cameras each on Port-1 and Port-2.

The following diagram shows connection of a OX03F10 Bayer GMSL camera on GMSL Port-0.



Supported resolutions and features

The following table shows the supported resolutions each camera module on the Ride SX platform.

Resolution	Aspect Ratio	0X3F10 Bayer GMSL	0X3F10 YUV GMSL
1920 x 1536	5:4	No	Yes
1920 x 1440	4:3	No	Yes
1928 x 1208	16:10	No	Yes
1920 x 1080	16:9	No	Yes
1824 x 1536	19:16	Yes	Yes
1440 x 1080	4:3	Yes	Yes
1280 x 720	16:9	Yes	Yes
1024 x 768	4:3	Yes	Yes
800 x 600	4:3	Yes	Yes
640 x 480	4:3	Yes	Yes
640 x 360	16:9	Yes	Yes
320 x 240	4:3	Yes	Yes

Advanced features such as SHDR, LDC, and EIS are not supported on QCS8275.

3.2 Choose the stream API

Qualcomm Linux supports following APIs for camera.

- GStreamer API

  GStreamer is an open-source multimedia framework. Qualcomm provides a GStreamer plugin (qtiqmmfsrc) that allows developers to control the camera subsystem in applications. See Qualcomm GStreamer plugins for more information.

- V4L2 API (QCS6490 only)

  V4L2 is a framework within the Linux kernel that provides support for video capture, video output, and other multimedia devices. Developers can operate the camera using the V4L2 API. The V4L2 API, which uses the CAMSS driver, is suitable for developers who only need to obtain raw images from the camera.

3.3 Stream camera with the GStreamer API

gst-launch-1.0 is a command-line GStreamer utility used to build and run a GStreamer pipeline. The pipeline is specified as a collection of elements with properties separated by exclamation marks (!).

Prerequisites

To use gst-launch-1.0 and GStreamer plugins, QIM-SDK (meta-qcom-qim-product-sdk) must be installed on the device. See Qualcomm Linux Build Guide for QIM-SDK build and installation information.

Note: Connect to the device console using SSH. See How To SSH? for instructions.

Run the following command in an SSH terminal:

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

QCS6490

Note: Ensure that MIPI cameras are connected to CSI slots. The OV9282 MIPI camera should be connected to the CAM 0A slot and the IMX577 MPI camera should be be connected to the CAM3 slot.

Single camera stream start

1. Run the following command in the device terminal:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc camera=0 ! 'video/x-raw,
format=NV12,\
width=1280,height=720,framerate=30/1' ! fakesink
```

2. This command starts the camera with 720p at 30 FPS configuration. The frame coming from the camera sensor is thrown away by fakesink. If the gst pipeline status is changed to “PLAYING” as shown below, this indicates that the camera is running. Since this command dumps camera frames to fakesink, nothing will be saved on the device.

```
gbm_create_device(187): Info: backend name is: msm_drm
Setting pipeline to PAUSED ...
Pipeline is live and does not need PREROLL ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
```

To stop the camera, press **CTRL+C**.

Video encoding

1. Run the following command in the device terminal:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc camera=0 ! \
video/x-raw,format=NV12,width=1280,height=720,framerate=30/1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc \
capture-io-mode=4 output-io-mode=5 extra-controls="controls,video_
bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
mux_avc.mp4
```

This command starts the camera with 720p at 30 FPS configuration and saves it as a video file after h264 video encoding. If the gst pipeline status is changed to “PLAYING”, this indicates the camera is running.

To stop the camera, press **CTRL+C**.

2. `/opt/mux_avc.mp4` is generated on the device. The recorded content can be pulled from

the device by running the following scp command on the host PC:

```
$ scp -r root@[ip-addr]:/opt/mux_avc.mp4 .
```

Video encoding and snapshot

1. Run the following command in the device terminal:

```
gst-pipeline-app -e qtiqmmfsrc name=camsrc camera=0 ! \
video/x-raw,format=NV12,width=1280,height=720,framerate=30/1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc \
capture-io-mode=4 output-io-mode=5 extra-controls="controls,video_
bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
mux_avc.mp4 \
camsrc.image_1 ! "image/jpeg,width=1280,height=720,framerate=30/1" \
! multifilesink location=/opt/frame%d.jpg async=false sync=true
enable-last-sample=false
```

2. Press **Enter**. This command will print the following menu and wait for user input.

```
##################################### MENU #########################
###########

============================== Pipeline
Controls==============================
(0) NULL: Set the pipeline into NULL state
(1) READY: Set the pipeline into READY state
(2) PAUSED: Set the pipeline into PAUSED state
(3) PLAYING: Set the pipeline into PLAYING state

====================================
Other====================================
(p) Plugin Mode: Choose a plugin which to control
(q) Quit : Exit the application

Choose an option:
```

3. Use the following menu steps to take a snapshot while recording video.

```
(1) ready -> (3) Playing -> (p)Plugin Mode : Select (8)camerasrc ->
(37) capture-image -> (1): still - Snapshot ->(1) Snapshot count (
'guint' value for arg1)
```

4. To stop the camera, press **Enter**, press **b** (back), and then press **q** (quit). The recorded video file and snapshot images are saved in `/opt/`. The recorded content can be pulled from the

device by running the following scp command on the host PC:

```
$ scp -r root@[ip-addr]:/opt/<file name> .
```

QCS9075

Note: Ensure that the camera (MIPI or GMSL) sensor is connected to the RB8 device. You can connect OV9282 MIPI cameras on CSI slots. Connect OX03f10 Bayer GMSL camera to GMSL Port-0 and OX03f10 YUV GMSL cameras to Port-2 and Port-3.

Note: There are random CSI errors on GMSL Port-0 that stop Bayer GMSL camera streaming. This issue will be fixed in the next GA release.

Single camera stream start

Run the following command in the device terminal:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc camera=0 ! 'video/x-raw,
format=NV12,\
width=1280,height=720,framerate=30/1' ! fakesink
```

The following command starts the camera with 720p at 30 FPS configuration. The frame coming from the camera sensor is thrown away by fakesink. If the gst pipeline status is changed to “PLAYING” as shown below, this indicates that the camera is running. Since this command dumps camera frames to fakesink, nothing will be saved on the device.

```
gbm_create_device(187): Info: backend name is: msm_drm
Setting pipeline to PAUSED ...
Pipeline is live and does not need PREROLL ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
```

To stop the camera, press **CTRL+C**.

Video encoding

1. Run the following command in the device terminal:

   ```
   gst-launch-1.0 -e qtiqmmfsrc name=camsrc camera=0 ! \
   video/x-raw,format=NV12,width=1280,height=720,framerate=30/1,\
   interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc \
   capture-io-mode=4 output-io-mode=5 extra-controls="controls,
   video_bitrate=6000000,\
   ```

```
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink
location=/opt/mux_avc.mp4
```

This command starts the camera with 720p at 30 FPS configuration and saves it as a video file after h264 video encoding. If the gst pipeline status is changed to “PLAYING”, this indicates the camera is running.

To stop the camera, press **CTRL+C**.

2. `/opt/mux_avc.mp4` is generated on the device. The recorded content can be pulled from the device by running the following scp command on the host PC:

```
$ scp -r root@[ip-addr]:/opt/mux_avc. mp4 .
```

Video encoding and snapshot

The snapshot use case is not enabled on QCS9075 and will not work in the current release.

QCS8275

Note: Ensure the GMSL camera is connected to the Ride SX device. You can connect the OX03f10 Bayer GMSL camera to GMSL Port-0 and OX03f10 YUV GMSL cameras to Port-1 and Port-2.

Single camera stream start

Run the following command in the device terminal:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc camera=0 ! 'video/x-raw,
format=NV12,\
width=1280,height=720,framerate=30/1' ! fakesink
```

The following command starts the camera with 720p at 30 FPS configuration. The frame coming from the camera sensor is thrown away by fakesink. If the gst pipeline status is changed to “PLAYING” as shown below, this indicates that the camera is running. Since this command dumps camera frames to fakesink, nothing will be saved on the device.

```
gbm_create_device(187): Info: backend name is: msm_drm
Setting pipeline to PAUSED ...
Pipeline is live and does not need PREROLL ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
```

To stop the camera, press **CTRL+C**.

Video encoding

1. Run the following command in the device terminal:

   ```
   gst-launch-1.0 -e qtiqmmfsrc name=camsrc camera=0 ! \
   video/x-raw,format=NV12,width=1280,height=720,framerate=30/1,\
   interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc \
   capture-io-mode=4 output-io-mode=5 extra-controls="controls,
   video_bitrate=6000000,\
   video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink
   location=/opt/mux_avc.mp4
   ```

   This command starts the camera with 720p at 30 FPS configuration and saves it as a video file after h264 video encoding. If the gst pipeline status is changed to “PLAYING”, this indicates the camera is running.

   To stop the camera, press **CTRL+C**.

2. `/opt/mux_avc.mp4` is generated on the device. The recorded content can be pulled from the device by running the following scp command on the host PC:

   ```
   $ scp -r root@[ip-addr]:/opt/mux_avc. mp4 .
   ```

Video encoding and snapshot

The snapshot use case is not enabled on QCS8275 and will not work in the current release.

Other GStreamer Samples

The QIM SDK includes GStreamer sample applications for camera and sample applications for AI/ML and other multimedia applications.

Note: Before using the sample applications, ensure that the installation prerequisites for gst-launch-1.0 and GStreamer plugins are met.

See Multimedia use case examples for examples using gst-launch-1.0.

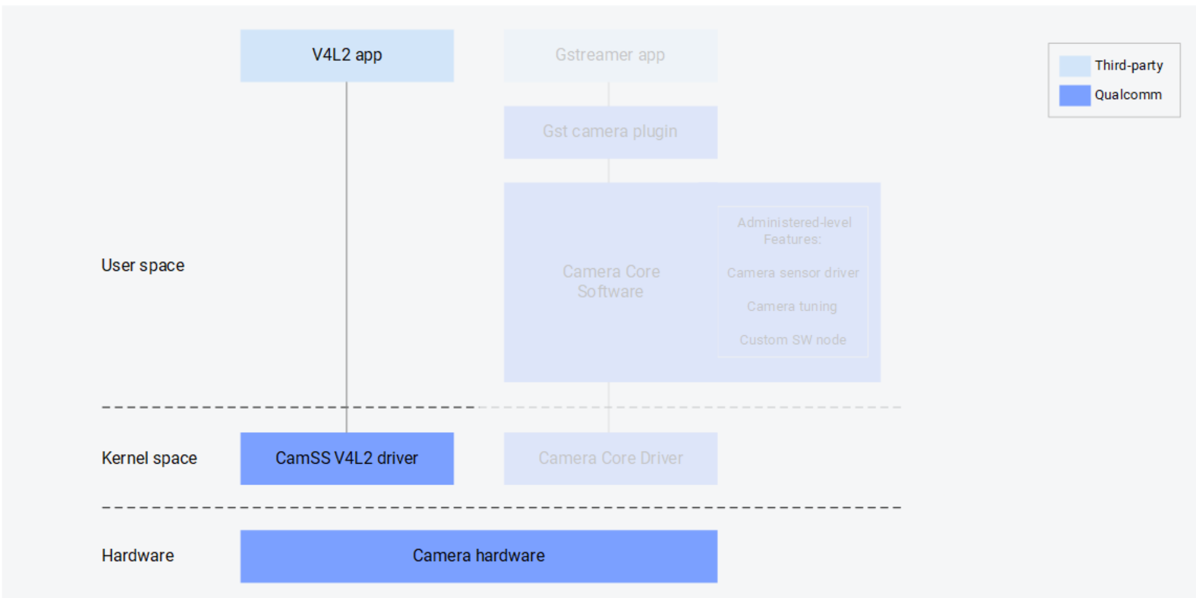
Stream camera with the V4L2 API

QCS6490

The V4L2 framework within the Linux kernel supports video devices. It provides an API that allows user space applications to interact with devices such as cameras and video capture cards.

More information on V4L2 is available from kernel.org.

Qualcomm supports the V4L2 interface camera ISP driver for raw frame dump functionality in the upstream kernel.



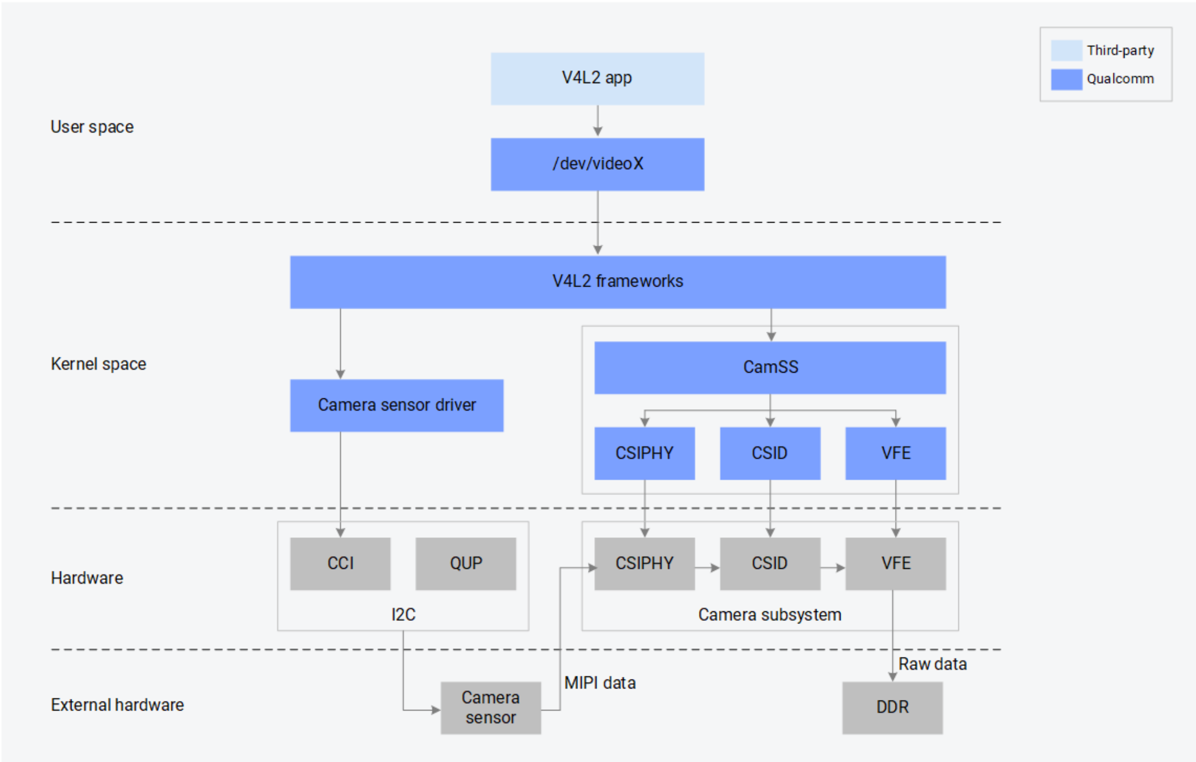
CamSS driver

The Qualcomm camera subsystem (CamSS) driver in the upstream kernel implements the V4L2, media controller, and V4L2 subdev interfaces.

Camera sensors using the V4L2 subdev interface in the kernel are supported.

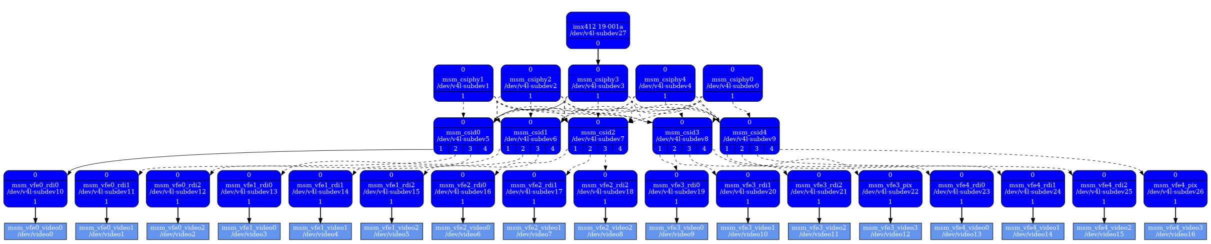
The CamSS driver consists of:

- CSIPHY module – Handles the physical layer of the CSI2 receivers. A separate camera sensor can be connected to each CSIPHY module.
- CSI Decoder (CSID) module – Handles the protocol and application layers of the CSI2 receivers. A CSID can decode a data stream from any CSIPHY.
- Video Front End (VFE) module – Represents the Image Front End (IFE) that contains Raw Dump Interface (RDI) input interfaces that bypass the image processing pipeline. The VFE also contains the AXI bus interface which writes output data to memory.



The CamSS driver implements the V4L2 interface. Each CSIPHY, CSID, and VFE module is represented by a single V4L2 sub-device.

As shown in the following diagram, each CSIPHY can connect to each CSID. Each CSID can connect to each VFE. Each RDI port has an individual video node.



CamSS V4L2 drivers are in `<kernel_root>/drivers/media/platform/qcom/camss`. `<kernel_root>` is the directory of the Linux kernel.

The following code snippet shows a v4l2_subdev_internal_ops for the CamSS driver.

```
static const struct v4l2_subdev_internal_ops csiphy_v4l2_internal_ops
= {
  .open = csiphy_init_formats,
};
...
static const struct v4l2_subdev_internal_ops csid_v4l2_internal_ops =
```

```
{
  .open = csid_init_formats,
};
...
static const struct v4l2_subdev_internal_ops vfe_v4l2_internal_ops =
{
  .open = vfe_init_formats,
};
```

The CamSS driver only uses the open interface to initialize the supported formats. It is called when the subdev device node is opened by an application.

The following code snippet shows a `v4l2_subdev_ops` for the CSID module. The CSIPHY and VFE modules have similar `v4l2_subdev_ops`.

```
static const struct v4l2_subdev_ops csid_v4l2_ops = {
   .core = &csid_core_ops,
   .video = &csid_video_ops,
   .pad = &csid_pad_ops,
};
...
static const struct v4l2_subdev_core_ops csid_core_ops = {
   .s_power = csid_set_power,
   .subscribe_event = v4l2_ctrl_subdev_subscribe_event,
   .unsubscribe_event = v4l2_event_subdev_unsubscribe,
};

static const struct v4l2_subdev_video_ops csid_video_ops = {
   .s_stream = csid_set_stream,
};

static const struct v4l2_subdev_pad_ops csid_pad_ops = {
   .enum_mbus_code = csid_enum_mbus_code,
   .enum_frame_size = csid_enum_frame_size,
   .get_fmt = csid_get_format,
   .set_fmt = csid_set_format,
};
```

These interfaces are called by the V4L2 framework or drivers in various contexts (for example, `ioctl`, `setup_link`, `start_streaming`). For example:

```
v4l2_subdev_call(subdev, pad, get_fmt, NULL, &fmt);``
```

The following code snippet shows a `media_device_ops` for the CamSS driver and `media_entity_operations` for the CSIPHY, CSID, and VFE modules.

```
static const struct media_device_ops camss_media_ops = {
   .link_notify = v4l2_pipeline_link_notify,
};
...
static const struct media_entity_operations csiphy_media_ops = {
   .link_setup = csiphy_link_setup,
   .link_validate = v4l2_subdev_link_validate,
};
...
static const struct media_entity_operations csid_media_ops = {
   .link_setup = csid_link_setup,
   .link_validate = v4l2_subdev_link_validate,
};
...
static const struct media_entity_operations vfe_media_ops = {
   .link_setup = vfe_link_setup,
   .link_validate = v4l2_subdev_link_validate,
};
```

`Camss_media_ops` is registered when the CamSS driver is registered (`camss_probe`). The CSIPHY, CSID, and VFE media entity operation. `link_setup` is called in the context of `media_device_setup_link`.

V4L2 sample application - Yavta

This section describes how to capture raw frame data using an application that supports the V4L2 interface.

Enable CamSS driver

1. Download upstream kernel source code using the following command.

```
devtool modify linux-qcom-custom
```

This downloads the kernel source code to
`<WORKSPACE>/build-qcom-wayland/workspace/sources/linux-qcom-custom/`.

2. Apply the following change to enable the CamSS driver in the device tree:

```
<WORKSPACE>/build-qcom-wayland/workspace/sources/linux-qcom-custom/
arch/ arm64/boot/dts/qcom/qcs6490-addons-rb3gen2.dtsi

&camss {
-        status = "disabled";
+        status = "okay";
         ports {
            #address-cells = <1>;
```

```
            #size-cells = <0>;
csiphy3_ep: endpoint {
};

&cci1 {
-        status = "disabled";
+        status = "okay";
};
```

3. Build the image following the Yocto build instructions in Build Guide.
4. Flash the image following the instructions in Flash Images.

Build and push the media controller utility and Yavta application

1. Build Yavta and the media controller.

```
bitbake yavta
```

2. Push the binaries to the device. In the following example, `<workspace>` is the directory of the Qualcomm software release.

Note: Connect to the device console using SSH. See How To SSH? for instructions.

```
scp <WORKSPACE>/build-qcom-wayland/tmp-glibc/deploy/ipk/armv8-2a/v4l-
utils_1.22.1-r0_armv8-2a.ipk root@[ip-addr]:/var/cache/camera/
scp <WORKSPACE>/build-qcom-wayland/tmp-glibc/deploy/ipk/armv8-2a/
yavta_0.0-r2_armv8-2a.ipk root@[ip-addr]:/var/cache/camera/
scp <WORKSPACE>/build-qcom-wayland/tmp-glibc/deploy/ipk/armv8-2a/
media-ctl_1.22.1-r0_armv8-2a.ipk root@[ip-addr]:/var/cache/camera/
scp <WORKSPACE>/build-qcom-wayland/tmp-glibc/deploy/ipk/armv8-2a/
libv4l_1.22.1-r0_armv8-2a.ipk root@[ip-addr]:/var/cache/camera/
```

3. Create a shell connection to the device:

```
ssh root@[ip-addr]
```

4. Disable the camera module.
   - The camera module cannot coexist with the CamSS driver.
   - Move the camera.ko module out from `/lib/modules/*` to make it not load automatically, then reboot the device.

```
# mount -o rw,remount /usr
# mv /lib/modules/*/updates/camera.ko /
```

```
# reboot
```

5. Create a shell connection to the device.

```
ssh root@[ip-addr]
```

6. Install the media-ctl, libv4l, v4l-utils, and yavta packages.

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

# opkg --nodeps install /var/cache/camera/media-ctl_1.22.1-r0_armv8-
2a.ipk --force-reinstall
# opkg --nodeps install /var/cache/camera/libv4l_1.22.1-r0_armv8-2a.
ipk --force-reinstall
# opkg --nodeps install /var/cache/camera/v4l-utils_1.22.1-r0_armv8-
2a.ipk --force-reinstall
# opkg --nodeps install /var/cache/camera/yavta_0.0-r2_armv8-2a.ipk -
-force-reinstall
```

7. Optionally, add the sensor driver and CamSS modules.

```
# modprobe imx412
# modprobe qcom-camss
```

This is an optional step since the imx412 and qcom-camss modules in /lib/modules are loaded automatically. Modprobe is used to add/remove modules from the Linux Kernel. The imx412 and qcom-camss modules are located in the following paths on the device:

- `/lib/modules/*/kernel/drivers/media/i2c/imx412.ko`
- `/lib/modules/*/kernel/drivers/media/platform/qcom/camss/qcom-camss.ko`

Loading of the qcom_camss and imx412 modules can be verified with the following lsmod command:

```
# lsmod | grep qcom_camss # lsmod | grep imx412
```

Check the media node number

Run the following command to print the media device node number for the CamSS driver.

If /dev/media0 does not list the qcom-camss driver, try with /dev/media1.

```
# media-ctl -p -d /dev/media0 | grep camss
driver       qcom-camss
bus info     platform:acaf000.camss
```

Find the sensor name

Run the following command to print the sensor name to the terminal:

```
# cat /sys/dev/char/81\:*/name | grep imx imx412 19-001a
imx577 17-001a
```

Configure the media controller

The media controller utility (media-ctrl) is a V4L2 utility used to configure camera subsystem subdevices. Use `media-ctl --help` to print usage information.

Note: Replace [x] with the number found via :ref. <Check the media node number>. For example, `# media-ctl -d /dev/media0 --reset`.

---

1. Reset all links to inactive:

```
# media-ctl -d /dev/media[x] --reset
```

2. Configure the camera sensor format and resolution on pipeline nodes:

```
# media-ctl -d /dev/media[x] -V '"imx577 17-001a":0[fmt:SRGGB10/
4056x3040 field:none]'
```

3. Configure CSIPHY with 4056x3040 resolution:

```
# media-ctl -d /dev/media[x] -V '"msm_csiphy3":0[fmt:SRGGB10/
4056x3040]' # media-ctl -d /dev/media[x] -V '"msm_csiphy3":1[fmt:
SRGGB10/4056x3040]'
```

4. Configure CSID with 4056x3040 resolution:

```
# media-ctl -d /dev/media[x] -V '"msm_csid0":0[fmt:SRGGB10/4056x3040]
' # media-ctl -d /dev/media[x] -V '"msm_csid0":1[fmt:SRGGB10/
4056x3040]'
```

5. Configure ISP with 3840x2160 resolution:

```
# media-ctl -d /dev/media[x] -V '"msm_vfe0_rdi0":0[fmt:SRGGB10/
4056x3040]' # media-ctl -d /dev/media[x] -V '"msm_vfe0_rdi0":1[fmt:
SRGGB10/4056x3040]'
```

6. Link the pipeline:

```
# media-ctl -d /dev/media[x] -l '"msm_csiphy3":1->"msm_csid0":0[1]'
# media-ctl -d /dev/media[x] -l '"msm_csid0":1->"msm_vfe0_rdi0":0[1]'
```

Capture images

The Yavta test application validates the camera using the V4L2 interface. Run Yavta to capture images:

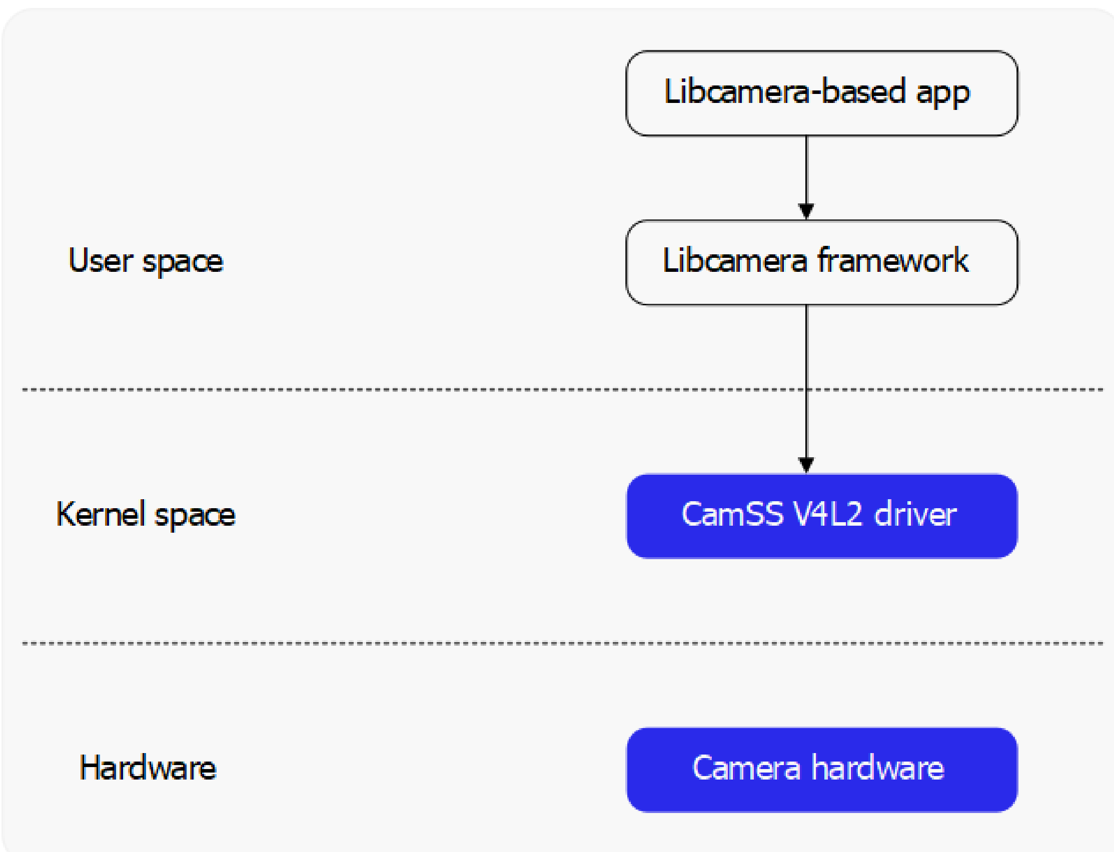
```
# yavta -B capture-mplane -c -I -n 5 -f SRGGB10P -s 4056x3040 -F /
dev/video0 --capture=5 --file='frame-#.raw'
```

V4L2 sample application - libcamera

libcamera framework and application

libcamera is an open-source software framework. It handles control of the V4L2 camera interface and exposes a native C++ API to upper layers. The applications and upper-level frameworks run based on the libcamera framework.

See libcamera architecture for more detail about the libcamera architecture.



libcamera is validated using the cam utility.

Capture raw frame data

The following steps describe how to capture raw frame data with the camera utility app using the V4L2 interface.

1. Enable the CamSS driver.

   a. Download upstream kernel source code using the following command.

```
devtool modify linux-qcom-custom
```

This downloads the kernel source code to `<WORKSPACE>/build-qcom-wayland/workspace/sources/linux-qcom-custom/`.

   b. Apply the following change to enable the CamSS driver in the device tree:

```
<WORKSPACE>/build-qcom-wayland/workspace/sources/linux-qcom-
custom/arch/ arm64/boot/dts/qcom/qcs6490-addons-rb3gen2.dtsi

&camss {
-        status = "disabled";
+        status = "okay";
         ports {
               #address-cells = <1>;
               #size-cells = <0>;
csiphy3_ep: endpoint {
};

&cci1 {
-        status = "disabled";
+        status = "okay";
};
```

2. Build the image following the Yocto build command in Build Guide.
3. Flash the image following the instructions in Flash images.
4. Disable the camera module.

The camera module cannot coexist with the CamSS driver. Move the camera.ko module out from `/lib/modules/*` to make it not load automatically, then reboot the device.

```
# mount -o rw,remount /usr
# mv /lib/modules/*/updates/camera.ko /
#reboot
```

5. Compile and push libcamera utilities.

```
bitbake libcamera
```

6. Push the binaries to the device. In the following example, `<workspace>` is the directory of the Qualcomm software release.

Note: Connect to the device console using SSH. See How To SSH? for instructions.

```
scp <WORKSPACE>/build-qcom-wayland/tmp-glibc/deploy/ipk/armv8-2a/
libcamera_202105+git0+acf8d028ed-r0_armv8-2a.ipk root@[ip-addr]:/var/
cache/camera/
scp <WORKSPACE>/build-qcom-wayland/tmp-glibc/deploy/ipk/armv8-2a/
libevent-pthreads-2.1-7_2.1.12-r0_armv8-2a.ipk root@[ip-addr]:/var/
cache/camera/
scp <WORKSPACE>/build-qcom-wayland/tmp-glibc/deploy/ipk/armv8-2a/
libevent-2.1-7_2.1.12-r0_armv8-2a.ipk root@[ip-addr]:/var/cache/
camera/
```

7. Create a shell connection to the device:

```
ssh root@[ip-addr]
```

8. Install libcamera utilities:

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

# opkg --nodeps install /var/cache/camera/libcamera_
202105+git0+acf8d028ed-r0_armv8-2a.ipk --force-reinstall
# opkg --nodeps install /var/cache/camera/libevent-pthreads-2.1-7_2.
1.12-r0_armv8-2a.ipk --force-reinstall
# opkg --nodeps install /var/cache/camera/libevent-2.1-7_2.1.12-r0_
armv8-2a.ipk --force-reinstall
```

9. Run the cam utility:

```
# cam -c 1 --capture=10 --file='frame-#.raw'
```

This runs the utility and captures 10 raw frames in the root directory and saves them into files.

Check the console log messages for information about the resolution and format of the dumped images.

```
Using camera /base/soc@0/cci@ac4b000/i2c-bus@1/camera@1a as cam0 [0:
15:01.067615799] [2155] INFO Camera camera.cpp:945 configuring
streams:
(0) 4056x3040-SRGGB10_CSI2P
cam0: Capture 10 frames
```

QCS9075

Note: This section is only applicable for QCS6490.

---

4 Enhance camera output

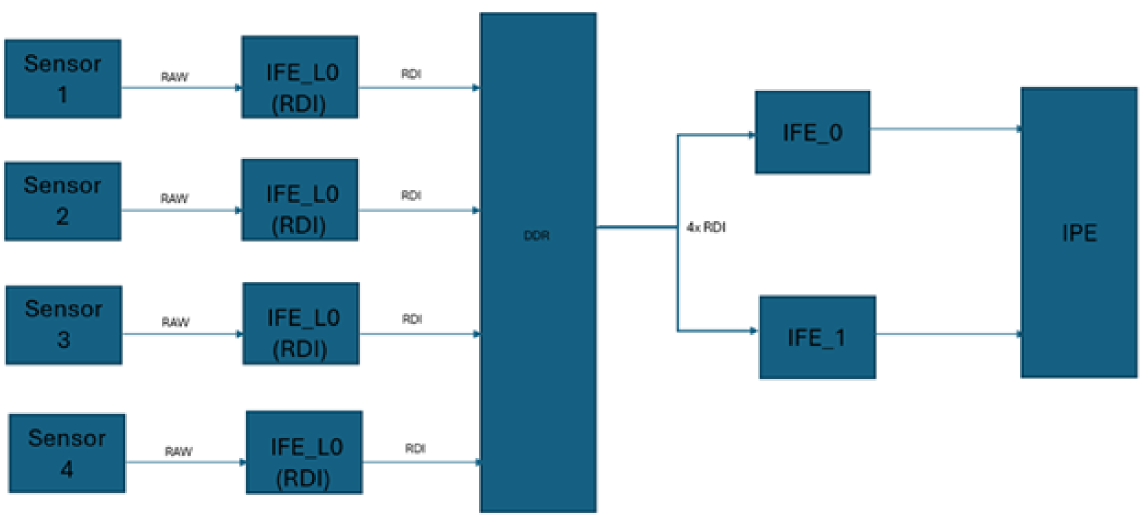
Note: This section is only applicable for QCS6490.

4.1 Support multi-camera using offline IFE

Note: This section is only applicable for QCS9075.

QCS9075 has two IFEs that support de-bayering (bayer-to-YUV processing) of the bayer camera images in real-time. This allows for support of concurrent streaming from two bayer cameras.

The Offline IFE feature allows the IFE to run in offline mode and supports de-bayering of two cameras using single IFE. This allows for support concurrent streaming from four bayer cameras.



Each Sensor is connected to one IFE-Lite hardware instance and data is dumped to DDR using the RDI port. A single IFE hardware is being used to read data from two IFE-Lites using bus fetch engine SFE Lite and processing data in Offline mode.

With this approach, instead of directly feeding the camera frames to IFE, frames are routed from IFE_LITE and dumped in DDR. Then they're provided to IFE using the fetch engine SFE Lite for bayer-to-YUV processing.

By enabling this feature, the following camera concurrency use cases are possible to run:

- Four OV9292 MIPI cameras connected to 4 MIPI CSI slots 0, 1, 2 ,and 3
- Single bayer OX03F10 GMSL camera connected to GMSL Port-0 and three OV9282 cameras connected to CSI slots 1, 2, and 3

Test procedure

Note: Connect to the device console using SSH. See How To SSH? for instructions.

To collect the user log, run the following command in the device:

```
# journalctl -f > /opt/user_log.txt
```

To collect the kernel log, run the following command in the device:

```
# dmesg -w  > /opt/kernel_log.txt
```

Run a GST command with `multicamera-hint=true` for each camera to enable this feature. For example:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc camera=0 multicamera-
hint=true ! \
video/x-raw,format=NV12,width=1280,height=720,framerate=30/1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc \
capture-io-mode=4 output-io-mode=5 extra-controls="controls,video_
bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
mux_avc.mp4
```

Similarly run other cameras with the `multicamera-hint=true` option.

Log verification

Check for the following prints in user logs:

```
CamX: [CORE_CFG]3509 3556 [CORE   ] camxpipeline.h:4222
SetPipelineStatus() RealTimeFeatureZSLPreviewRawOfflineIFE_0_cam_0
status is now PipelineStatus::STREAM_ON
```

Check for the following prints in kernel logs:

```
CAM_INFO: CAM-ISP: cam_ife_hw_mgr_print_acquire_info: 1733: 0:4:11.
835 Acquired Single IFE[0] SFE[0] OFFLINE: Y with [9 pix] [0 pd] [0
rdi] ports for ctx:1
```

4.2 Enable high dynamic range

Note: This section is only applicable for QCS6490.

Staggered high dynamic range (SHDR) mode is a sensor feature that outputs frames with different exposure times - Long exposure frame (LEF) and short exposure frame (SEF).

- The sensor outputs a pair of two lines as one unit. The frame of LEF and the frame of SEF are output alternately in the pair of these two lines.
- The rolling shutter readout is staggered (row interleaved) so that the short integration starts immediately (within the same frame) after sampling of the long integration. This is also called Digital Overlap (DOL) mode.
- The SHDR sensors are capable of outputting LEF and SEF frames with a single virtual channel (the single frame contains both LEF and SEF in an interleaved manner) or different virtual channels (LEF and SEF frames are output separately on different virtual channels)
- On Qualcomm chipsets, there are two solutions (SHDR v3 and SHDR v2). SHDR v3 works for dual virtual channels. SHDR v2 works for single virtual channel.

Note: Connect to the device console using SSH. See How To SSH? for instructions.

To collect the log, run the following command on the device:

```
# journalctl -f > /opt/log.txt
```

SHDR v3 - Dual virtual channel with two-frame exposure

The SHDR v3 solution uses sensors that can send the exposure frames (LEF, SEF) on different CSI virtual channels. Long and short exposure frames are transmitted on different virtual channels.

Use the following GStreamer command to enable SHDR v3:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc camera=0 vhdr=2 ! \
video/x-raw,format=NV12,width=3840,height=2160,framerate=30/1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc \
capture-io-mode=4 output-io-mode=5 extra-controls="controls,video_
bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
cam_4k.mp4
```

The video file is saved at `/opt/cam_4k.mp4`.

To verify the SHDR v3 use case is selected, check for the following log:

```
cam-server: CamX: [CORE_CFG]891 23288 [CORE  ] camxpipeline.h:3024
SetPipelineStatus() RealTimeYUVSHDR_0 status is now PipelineStatus::
STREAM_ON
```

SHDR v2 - Single virtual channel with two-frame exposure

The SHDR v2 solution requires the sensor to output both LEF and SEF on a single virtual channel in an interleaved manner (also referred to as DOL mode). The rolling shutter readout is staggered (row interleaved) so that the short integration starts immediately (within the same frame) after sampling of the long integration.

Use the following GStreamer command to enable SHDR v2:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc camera=0 vhdr=1 ! \
video/x-raw,format=NV12,width=3840,height=2160,framerate=30/1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc \
capture-io-mode=4 output-io-mode=5 extra-controls="controls,video_
bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
cam_4k.mp4
```

The video file is saved at `/opt/cam_4k.mp4`.

To verify the SHDR v2 use case is selected, check for the following log:

```
cam-server: CamX: [CORE_CFG]852 2070 [CORE] camxpipeline.h:3015
SetPipelineStatus()
RealTimeSHDR_0 status is now PipelineStatus::STREAM_ON
```

Defog feature

Defog is a fog detection and removal technique. This feature allows users to remove the foggy effect in poor weather conditions, such as rain, smog, haze, or fog. It provides a defogged image by improving the image quality in SHDR v2 and SHDR v3 use cases. The defog library performs defog operations with the statistics and interpolation data collected from IFE, BPS, and IPE and generates new tables for the IQ modules to apply in the next frame.

To validate the defog feature:

1. Use the `shdrModeType=5` override setting in `/var/cache/camera/camxoverridesettings.txt` to enable the defog feature in SHDR use cases.
2. Use the following override settings in `/var/cache/camera/camxoverridesettings.txt` to verify the defog feature is working:

```
logInfoMask=0x40000
logVerboseMask=0x40000
```

3. Test the defog feature.

   - Run the following command to test defog with a SHDR v2 use case:

```
gst-pipeline-app -e qtiqmmfsrc name=camsrc vhdr=1  ! \
video/x-raw,format=NV12,width=3840,height=2160,framerate=30/
1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc
capture-io-mode=4 \
output-io-mode=5  extra-controls="controls,video_
bitrate=6000000,video_bitrate_mode=0;" ! queue ! \
h264parse ! mp4mux ! queue ! filesink location=/opt/cam_4k.
mp4
```

   - Run the following command to test defog with a SHDR v3 use case:

```
gst-pipeline-app -e qtiqmmfsrc name=camsrc vhdr=2 ! \
video/x-raw,format=NV12,width=3840,height=2160,framerate=30/
1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc
capture-io-mode=4 \
output-io-mode=5 extra-controls="controls,video_
bitrate=6000000,video_bitrate_mode=0;" ! queue ! \
h264parse ! mp4mux ! queue ! filesink location=/opt/cam_4k.
mp4
```

4. Enable defog.
   a. Select **(1) READY**.
   b. Select **(3) PLAYING**.
   c. Press **Enter**.
   d. Select **(p) Plugin Mode**.
   e. Select **(8) camsrc**.
   f. Select **(24) defog-table**.
   g. Enter the following string to enable defog.

   ```
   org.quic.camera.defog, enable=true, strength=3, ates_
   strength=1;
   ```

   h. Close the camera by selecting **(b) BACK** and then **(q) QUIT**. The video file is saved at/opt/cam_4k.mp4.
5. Check for the following logs to verify the defog feature is enabled:

```
defog.cpp:5782 DefogInitialize() ===================== Defog lib
v2.5.0=====================
cam-server: CamX: [ VERB]793766 793830 [CHI ] camxchinodedefog.
cpp:1981 SetDeFog2ConfigParams() Previous ( defog_en = 0 defog_
strength = 0 ates_strength = 0
cam-server: CamX: [ VERB]793766 793830 [CHI ] camxchinodedefog.
cpp:1984 SetDeFog2ConfigParams() Current ( defog_en = 1 defog_
strength = 3 ates_strength = 1
```

4.3 Enable EIS and LDC

Note: This section is only applicable for QCS6490.

This section describes the EIS and LDC features and how to run their various use cases.

Electronic Image Stabilization (EIS) is an image enhancement technique using electronic processing. EIS minimizes blurring and compensates for device shake. EIS takes the motion data from an IMU sensor and generates the transformation matrix to compensate for device moment in all three directions.

A gyroscope sensor provides motion in pitch, yaw, and roll. The image warping library module that runs on the GPU applies warping to the image based on the generated transformation matrix. The iWarp library uses the OpenGL API for warping. Lens Distortion Correction (LDC) is the process to correct the distortion that's introduced due to fisheye lenses, which makes a straight line in a scene be captured as a curved line due to the distortion of the lens. A static mesh is generated using the calibration process of the lens.

The image warping library module that runs on the GPU applies warping to the image based on the generated static mesh. The iWarp library uses the OpenGL API for warping.

EIS and LDC can be run independently or at the same time. The following are the possible GST command options for EIS and LDC use cases:

GST command option	Description
`eis=1`	Enables EIS on first stream only
`eis=2`	Enables EIS on two streams
`ldc=1`	Enables LDC on all (1 or 2) streams
`eis=1, ldc=1`	Enables EIS and LDC on the first stream only
`eis=2, ldc=1`	Enables EIS and LDC on two streams

Note: Connect to the device console using SSH. See How To SSH? for instructions.

To collect the log, run the following command on the device:

```
# journalctl -f > /opt/log.txt
```

EIS single stream use case

Use the following GStreamer command to enable this use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc eis=1 ! \
video/x-raw,format=NV12,width=1920,height=1080,framerate=30/1 ! \
v4l2h264enc capture-io-mode=4 output-io-mode=5 extra-controls=
"controls,video_bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
cam_1080p.mp4
```

Verify this use case is selected using the following logs:

```
cam-server[926]: [INFO]: RecorderCameraContext : OpenCamera: EIS on
single stream is ON..
cam-server[908]: CamX: [ INFO]908 981 [CHI    ] camxchinodeeisdgv26.
cpp:1346 Initialize() m_nodeCaps 131072
cam-server[908]: CamX: [CORE_CFG]908 1796 [CORE      ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeFeatureZSLPreviewRawYuvEisIoT_0
status is now PipelineStatus::STREAM_ON
```

EIS enabled on first of two streams use case

Use the following GStreamer command to enable this use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc eis=1 video_0::type=preview
! \
video/x-raw,format=NV12,width=1920,height=1080,framerate=30/1 ! \
v4l2h264enc capture-io-mode=4 output-io-mode=5 \
extra-controls="controls,video_bitrate=6000000,video_bitrate_mode=0;"
! \
h264parse ! mp4mux ! filesink location=/opt/cam_prev.mp4 camsrc. ! \
video/x-raw,format=NV12,width=1280,height=720,framerate=30/1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc capture-
io-mode=4 \
output-io-mode=5 extra-controls="controls,video_bitrate=6000000,
video_bitrate_mode=0;" \
! h264parse ! mp4mux ! filesink location=/opt/cam_vid.mp4
```

Verify this use case is selected using the following logs:

```
cam-server[926]: [INFO]: RecorderCameraContext : OpenCamera: EIS on
single stream is ON..
cam-server[926]: CamX: [ INFO]926 988 [CHI    ] camxchinodeeisdgv26.
cpp:1346 Initialize() m_nodeCaps 131072
cam-server[926]: CamX: [CORE_CFG]926 2529 [CORE       ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeFeatureZSLPreviewRawYuvEisIoT_0
status is now PipelineStatus::STREAM_ON
```

EIS enabled on two streams

Use the following GStreamer command to enable this use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc eis=2 video_0::type=preview
! \
video/x-raw,format=NV12,width=1920,height=1080,framerate=30/1 ! \
v4l2h264enc capture-io-mode=4 output-io-mode=5 \
extra-controls="controls,video_bitrate=6000000,video_bitrate_mode=0;"
! \
h264parse ! mp4mux ! filesink location=/opt/cam_prev.mp4 camsrc. ! \
video/x-raw,format=NV12,width=1280,height=720,framerate=30/1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc capture-
io-mode=4 \
output-io-mode=5 extra-controls="controls,video_bitrate=6000000,
video_bitrate_mode=0;" \
! h264parse ! mp4mux ! filesink location=/opt/cam_vid.mp4
```

Verify this use case is selected using the following logs:

```
cam-server[914]: [INFO]: RecorderCameraContext : OpenCamera: EIS on
dual stream is ON..
cam-server[914]: CamX: [ INFO]914 960 [CHI   ] camxchinodeeisdgv26.
cpp:1346 Initialize() m_nodeCaps 131072
cam-server[914]: CamX: [CORE_CFG]914 2421 [CORE      ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeFeatureZSLPreviewRawYuvEisIoT_0
status is now PipelineStatus::STREAM_ON
```

LDC single stream use case

Use the following GStreamer command to enable this use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc ldc=1 ! \
video/x-raw,format=NV12,width=1920,height=1080,framerate=30/1 ! \
v4l2h264enc capture-io-mode=4 output-io-mode=5 extra-controls=
"controls,video_bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
cam_1080p.mp4
```

Verify this use case is selected using the following logs:

```
cam-server[926]: [INFO]: RecorderCameraContext : OpenCamera: EIS is
disabled cam-server[926]: [INFO]: RecorderCameraContext : OpenCamera:
LDC is ON..
cam-server[926]: CamX: [ INFO]926 990 [CHI   ] camxchinodeeisdgv26.
cpp:1346 Initialize() m_nodeCaps 524288
cam-server[926]: CamX: [CORE_CFG]926 2132 [CORE      ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeFeatureZSLPreviewRawYuvEisIoT_0
status is now PipelineStatus::STREAM_ON
```

LDC enabled on two streams use case

Use the following GStreamer command to enable this use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc ldc=1 video_0::type=preview
! \
video/x-raw,format= NV12,width=1920,height=1080,framerate=30/1 ! \
v4l2h264enc capture-io-mode=4 output-io-mode=5 extra-controls=
"controls,video_bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
cam_prev.mp4 \
```

```
camsrc. ! video/x-raw,format=NV12,width=1280,height=720,\
framerate=30/1,interlace-mode=progressive,colorimetry=bt601 !
v4l2h264enc capture-io-mode=4 \
output-io-mode=5 extra-controls="controls,video_bitrate=6000000,
video_bitrate_mode=0;" ! \
h264parse ! mp4mux ! filesink location=/opt/cam_vid.mp4
```

Verify this use case is selected using the following logs:

```
cam-server[914]: [INFO]: RecorderCameraContext : OpenCamera: EIS is
disabled cam-server[914]: [INFO]: RecorderCameraContext : OpenCamera:
LDC is ON..
cam-server[914]: CamX: [ INFO]914 966 [CHI    ] camxchinodeeisdgv26.
cpp:1346 Initialize() m_nodeCaps 524288
cam-server[914]: CamX: [CORE_CFG]914 1591 [CORE       ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeFeatureZSLPreviewRawYuvEisIoT_0
status is now PipelineStatus::STREAM_ON
```

EIS and LDC single stream use case

Use the following GStreamer command to enable this use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc eis=1 ldc=1 ! \
video/x-raw,format=NV12,width=1920,height=1080,framerate=30/1 \
! v4l2h264enc capture-io-mode=4 output-io-mode=5 extra-controls=
"controls,video_bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
cam_1080p.mp4
```

Verify this use case is selected using the following logs:

```
cam-server[914]: [INFO]: RecorderCameraContext : OpenCamera: EIS on
single stream is ON..
cam-server[914]: [INFO]: RecorderCameraContext : OpenCamera: LDC is
ON.. cam-server[914]: CamX: [CORE_CFG]914 2621 [CORE      ]
camxpipeline.h:3024 SetPipelineStatus()
RealTimeFeatureZSLPreviewRawYuvEisIoT_0 status is now PipelineStatus:
:STREAM_ON
```

EIS and LDC enabled on first of two streams use case

Use the following GStreamer command to enable this use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc eis=1 ldc=1 video_0::
type=preview ! \
video/x-raw,format= NV12,width=1920,height=1080,framerate=30/1 ! \
v4l2h264enc capture-io-mode=4 output-io-mode=5 extra-controls=
"controls,video_bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
cam_prev.mp4 camsrc. ! \
video/x-raw,format=NV12,width=1280,height=720,framerate=30/1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc capture-
io-mode=4 \
output-io-mode=5 extra-controls="controls,video_bitrate=6000000,
video_bitrate_mode=0;" ! \
h264parse ! mp4mux ! filesink location=/opt/cam_vid.mp4
```

Verify this use case is selected using the following logs:

```
cam-server[2661]: [INFO]: RecorderCameraContext : OpenCamera: EIS on
single stream is ON..
cam-server[2661]: CamX: [ INFO]2661 2663 [CHI        ]
camxchinodeeisdgv26.cpp:1346 Initialize() m_nodeCaps 262144
cam-server[2661]: CamX: [CORE_CFG]2661 2834 [CORE ] camxpipeline.h:
3024 SetPipelineStatus() RealTimeFeatureZSLPreviewRawYuvEisIoT_0
status is now PipelineStatus::STREAM_ON
```

EIS and LDC enabled on two streams use case

Use the following GStreamer command to enable this use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc eis=2 ldc=1 video_0::
type=preview ! \
video/x-raw,format=NV12,width=1920,height=1080,framerate=30/1 ! \
v4l2h264enc capture-io-mode=4 output-io-mode=5 extra-controls=
"controls,video_bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
cam_prev.mp4 camsrc. ! \
video/x-raw,format=NV12,width=1280,height=720,\
framerate=30/1,interlace-mode=progressive,colorimetry=bt601 !
v4l2h264enc capture-io-mode=4 \
output-io-mode=5 extra-controls="controls,video_bitrate=6000000,
video_bitrate_mode=0;" ! \
h264parse ! mp4mux ! filesink location=/opt/cam_vid.mp4
```

Verify this use case is selected using the following logs:

```
cam-server[882]: [INFO]: RecorderCameraContext : OpenCamera: EIS on
dual stream is ON..
cam-server[882]: CamX: [ INFO]882 977 [CHI    ] camxchinodeeisdgv26.
cpp:1346 Initialize() m_nodeCaps 262144
cam-server[882]: CamX: [CORE_CFG]882 1613 [CORE       ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeFeatureZSLPreviewRawYuvEisIoT_0
status is now PipelineStatus::STREAM_ON
```

4.4 Advanced feature concurrences

Note: This section is only applicable for QCS6490.

This section describes the possible concurrency use cases for running SHDR, EIS, and LDC.

The following are the combinations available to run SHDR, EIS, and LDC in concurrency:

GST command option	Description
`vhdr=2 eis=1`	SHDR v3 applied on all streams. EIS applied on first stream
`vhdr=2 eis=2`	SHDR v3 and EIS applied on two streams
`vhdr=2 ldc=1`	SHDR v3+ LDC applied on all (1 or 2) streams
`vhdr=1 ldc=1`	SHDR v2+ LDC applied on all (1 or 2) streams
`eis=1 vhdr=2 ldc=1`	SHDRV3 applied on two streams. EIS+LDC applied on first stream
`eis=2 vhdr=2 ldc=1`	SHDRV3+EIS+LDC applied on two streams

Note: The current release has a stability issue with SHDR v3 and EIS concurrency use cases. The GST command isn't terminating after pressing Ctrl+C. A device reboot is required to run the camera the next time.

Note: Connect to the device console using SSH. See How To SSH? for instructions.

To collect the log, run the following command on the device:

```
# journalctl -f > /opt/log.txt
```

SHDR v3 and EIS concurrency use case

SHDR and EIS concurrency is enabled for SHDR v3 only. This feature uses SHDR v3 sensor mode and does SHDR v3 processing first followed by EIS processing.

Use the following GStreamer command to enable the SHDR v3+EIS single stream use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc eis=1 vhdr=2 ! video/x-raw,
format=NV12,\
width=1920,height=1080,framerate=30/1 ! v4l2h264enc capture-io-mode=4
output-io-mode=5 \
extra-controls="controls,video_bitrate=6000000,video_bitrate_mode=0;"
! h264parse ! \
mp4mux ! filesink location=/opt/cam_1080p.mp4
```

Verify the SHDR v3 + EIS single stream use case using the following logs:

```
cam-server[882]: [INFO]: RecorderCameraContext : OpenCamera: EIS on
single stream is ON..
cam-server[882]: CamX: [ INFO]882 975 [CHI   ] camxchinodeeisdgv26.
cpp:1346 Initialize() m_nodeCaps 131072
cam-server[882]: CamX: [CORE_CFG]882 1877 [CORE      ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeYUVSHDREISIOT_0 status is now
PipelineStatus::STREAM_ON
```

Use the following GStreamer command to enable the SHDR v3 + EIS on two streams use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc eis=2 vhdr=2 video_0::
type=preview ! \
video/x-raw,format= NV12,width=1920,height=1080,framerate=30/1 ! \
v4l2h264enc capture-io-mode=4 output-io-mode=5 extra-controls=
"controls,video_bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
cam_prev.mp4 \
camsrc. ! video/x-raw,format=NV12,width=1280,height=720,\
framerate=30/1,interlace-mode=progressive,colorimetry=bt601 !
v4l2h264enc capture-io-mode=4 \
output-io-mode=5 extra-controls="controls,video_bitrate=6000000,
video_bitrate_mode=0;" ! \
h264parse ! mp4mux ! filesink location=/opt/cam_vid.mp4
```

Verify SHDR v3 + EIS on two streams use case using the following logs:

```
cam-server[882]: [INFO]: RecorderCameraContext : OpenCamera: EIS on
dual stream is ON..
cam-server[882]: CamX: [ INFO]882 952 [CHI   ] camxchinodeeisdgv26.
```

```
cpp:1346 Initialize() m_nodeCaps 131072
cam-server[882]: CamX: [CORE_CFG]882 2058 [CORE      ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeYUVSHDREISIOT_0 status is now
PipelineStatus::STREAM_ON
```

SHDR v3 and LDC concurrency use case

This feature uses SHDR v3 sensor mode and does SHDR v3 processing followed by LDC processing.

Use the following GStreamer command to enable this use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc vhdr=2 ldc=1 ! video/x-raw,\
format=NV12,width=1920,height=1080,framerate=30/1 ! v4l2h264enc
capture-io-mode=4 output-io-mode=5 \
extra-controls="controls,video_bitrate=6000000,video_bitrate_mode=0;"
! h264parse ! mp4mux ! \
filesink location=/opt/cam_1080p.mp4
```

Verify this use case is selected using the following logs:

```
cam-server[2089]: [INFO]: RecorderCameraContext : OpenCamera: EIS is
disabled cam-server[2089]: CamX: [ INFO]2089 2090 [CHI  ]
camxchinodeeisdgv26.cpp:1346 Initialize() m_nodeCaps 524288
cam-server[2089]: CamX: [CORE_CFG]2089 2324 [CORE    ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeYUVSHDREISIOT_0 status is now
PipelineStatus::STREAM_ON
```

SHDR v2 and LDC concurrency use case

This feature uses SHDR v2 sensor mode and does SHDR v2 processing followed by LDC processing.

Use the following GStreamer command to enable this use case:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc vhdr=1 ldc=1 ! video/x-raw,\
format=NV12,width=1920,height=1080,framerate=30/1 ! v4l2h264enc
capture-io-mode=4 output-io-mode=5 \
extra-controls="controls,video_bitrate=6000000,video_bitrate_mode=0;"
! h264parse ! mp4mux ! \
filesink location=/opt/cam_1080p.mp4
```

Verify this use case is selected using the following logs:

```
cam-server[2358]: [INFO]: RecorderCameraContext : OpenCamera: EIS is
disabled cam-server[2358]: CamX: [ INFO]2358 2366 [CHI  ]
camxchinodeeisdgv26.cpp:1346 Initialize() m_nodeCaps 524288
cam-server[2358]: CamX: [CORE_CFG]2358 2527 [CORE    ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeSHDRIOTLDC_0 status is now
PipelineStatus::STREAM_ON
```

SHDR v3, EIS, and LDC concurrency use case

SHDR, EIS, and LDC concurrency is enabled with SHDR v3. This feature uses SHDR v3 sensor mode and does SHDR v3 processing followed by EIS and LDC processing.

Use the following GStreamer command to enable the SHDR v3, EIS, and LDC concurrency use case on a single stream:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc eis=1 vhdr=2 ldc=1 ! video/
x-raw,\
format=NV12,width=1920,height=1080,framerate=30/1 ! v4l2h264enc
capture-io-mode=4 output-io-mode=5 \
extra-controls="controls,video_bitrate=6000000,video_bitrate_mode=0;"
! h264parse ! mp4mux ! \
filesink location=/opt/cam_1080p.mp4
```

Verify the SHDR v3, EIS, and LDC concurrency on a single stream use case using the following logs:

```
cam-server[2555]: [INFO]: RecorderCameraContext : OpenCamera: EIS on
single stream is ON..
cam-server[2555]: CamX: [ INFO]2555 2560 [CHI         ]
camxchinodeeisdgv26.cpp:1346 Initialize() m_nodeCaps 262144
```

```
cam-server[2555]: CamX: [CORE_CFG]2555 2764 [CORE    ] camxpipeline.
h:3024
SetPipelineStatus() RealTimeYUVSHDREISIOT_0 status is now
PipelineStatus::STREAM_ON
```

Use the following GStreamer command to enable the SHDR v3, EIS, and LDC concurrency use case on two streams:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc eis=2 vhdr=2 ldc=1 video_0::
type=preview ! \
video/x-raw,format=NV12,width=1920,height=1080,framerate=30/1 ! \
v4l2h264enc capture-io-mode=4 output-io-mode=5 extra-controls=
"controls,video_bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink location=/opt/
cam_prev.mp4 \
camsrc. ! video/x-raw,format=NV12,width=1280,height=720,\
framerate=30/1,interlace-mode=progressive,colorimetry=bt601 !
v4l2h264enc capture-io-mode=4 \
output-io-mode=5 extra-controls="controls,video_bitrate=6000000,
video_bitrate_mode=0;" ! \
h264parse ! mp4mux ! filesink location=/opt/cam_vid.mp4
```

Verify the SHDR v3, EIS, and LDC concurrency on two streams use case using the following logs:

```
cam-server[907]: [INFO]: RecorderCameraContext : OpenCamera: EIS on
dual stream is ON..
cam-server[907]: CamX: [ INFO]907 984 [CHI   ] camxchinodeeisdgv26.
cpp:1346 Initialize() m_nodeCaps 262144
cam-server[907]: CamX: [CORE_CFG]907 1586 [CORE      ] camxpipeline.
h:3024 SetPipelineStatus() RealTimeYUVSHDREISIOT_0 status is now
PipelineStatus::STREAM_ON
```

4.5 Enable multiple ROI streams

Note: This section is only applicable for QCS6490.

This is an advanced camera control feature that allows the user to get multiple ROI streams from a single camera with each stream showing a different ROI from the full FOV. It supports up to three cropped ROI streams and one full ROI stream.

In case of three ROIs, the app sends five streams to configure, with one full FOV output stream, one input stream, and three ROI output streams. The real-time pipeline produces the full FOV image. The application receives the full FOV image and submits the full FOV buffer back to the camera reprocess pipeline as an input image. The reprocess pipeline generates up to three ROI images from the full FOV image.

Note: Connect to the device console using SSH. See How To SSH? for instructions.

To collect the log, run the following command on the device:

```
# journalctl -f > /opt/log.txt
```

To verify the multiple ROI feature:

1. Connect an HDMI monitor to the device.
2. Run the following commands from an SSH terminal to set the display variables:

```
# export XDG_RUNTIME_DIR=/dev/socket/weston
# export WAYLAND_DISPLAY=wayland-1
```

3. Run the following GStreamer command:

```
gst-pipeline-app -e qtiqmmfsrc input-roi-enable=true video_0::
type=preview \
video_1::type=preview video_2::type=preview video_3::
type=preview video_0::reprocess-enable=true \
name=camsrc ! video/x-raw,format=NV12,width=1920,height=1080,
framerate=30/1 ! \
queue ! waylandsink x=0 y=0 width=959 height=540 qos=false
sync=false async=true \
camsrc. ! video/x-raw,format=NV12,width=1920,height=1080,
framerate=30/1 ! \
queue ! waylandsink x=960 y=0 width=960 height=540 qos=false
sync=false async=true \
camsrc. ! video/x-raw,format=NV12,width=1920,height=1080,
framerate=30/1 ! \
queue ! waylandsink x=0 y=540 width=960 height=540 qos=false
sync=false async=true \
camsrc. ! video/x-raw,format=NV12,width=1920,height=1080,
framerate=30/1 ! \
queue ! waylandsink x=960 y=540 width=960 height=540 qos=false
sync=false async=true
```

4. Select **(1) READY**.
5. Select **(3) PLAYING**. Press **Enter**.

6. Select **(p) Plugin Mode**.
7. Select (**13) camsrc**. You should see four preview streams with Full FOV.
8. Select **(32) input-roi-info**.
9. Enter the following input:

```
<1920, 0, 1920, 1080, 0, 1080, 1920, 1080, 0, 0, 1920, 1080>
```

You should see one full FOV and three ROI streams on HDMI.

To close the camera, select **(b) BACK** and then **(q) QUIT**.

Check the following UMD logs to verify this feature is applied on each stream:

```
cam-server[2810]: CamX: [ INFO]2810 2892 [PPROC ] camxipenode.cpp:
2928 FillFrameZoomWindow() ZDBG IPE[3] crop Window [0, 0, 1920, 1080]
full size 1920X1080 active 4056X3040, requestId 811 cropType 2 ROI
count 3
cam-server[2810]: CamX: [ INFO]2810 2895 [PPROC ] camxipenode.cpp:
2928 FillFrameZoomWindow() ZDBG IPE[2] crop Window [0, 1080, 1920,
1080] full size 1920X1080 active 4056X3040, requestId 811 cropType 1
ROI count 3
cam-server[2810]: CamX: [ INFO]2810 2891 [PPROC ] camxipenode.cpp:
2928
 FillFrameZoomWindow() ZDBG IPE[1] crop Window [1920, 0, 1920, 1080]
full size 1920X1080 active 4056X3040, requestId 811 cropType 0 ROI
count 3
```

Note: The total bandwidth for the Full FOV stream + ROI streams needs to be less than 4K, which is the max capability of the QCS6490 chipset.

4.6 Switch linear vs. SHDR mode automatically

Note: This section is only applicable for QCS6490.

This feature allows on-the-fly switching between Linear and SHDR v2 mode pipelines without stopping and starting the camera session.

This feature is helpful to prevent interrupting a video session when there is a SHDR vs. Linear mode switch needed based on lux value. It also helps reduce the pipeline switch latency as the pipelines (Linear and SHDR) are pre-initialized during session creation time. During mode switch, hardware resources of the pipeline are released (sensor, IFE, IPE) and then acquired and reconfigured for the new pipeline.

Note: This feature works only for SHDR v2 and Linear mode pipelines. It will be enabled for SHDR v3 and Linear mode in a future release.

Note: Connect to the device console using SSH. See How To SSH? for instructions.

To collect the log, run the following command on the device:

```
# journalctl -f > /opt/log.txt
```

Use `vhdr=3` in a GST command to enable this feature. To enable the feature and toggle between Linear and SHDR modes:

1. Run the gst-camera-metadata-example application, which can support setting the vendor tag to switch the use case during runtime. For example:

   ```
   gst-camera-metadata-example -p "qtiqmmfsrc name=camsrc camera=0
   vhdr=3 ! \
   video/x-raw,format=NV12,width=3840,height=2160,framerate=30/1,\
   interlace-mode=progressive,colorimetry=bt601 ! queue !
   v4l2h264enc capture-io-mode=4 \
   output-io-mode=5 ! queue ! h264parse ! mp4mux ! queue ! filesink
   location=/opt/mux4k.mp4"
   ```

2. In the application, select **(1) READY**.

3. Select **(3) PLAYING**. The camera starts in linear mode.

4. Select **(4) META**. The following menu is shown:

   ```
    ---------MENU---------
   (1)         video-metadata
   (2)         image-metadata
   (3)         static-metadata
   (4)         session-metadata
   ```

5. Select **(1) video-metadata**. The following menu is shown:

   ```
    ---------video-metadata---------
   (1)         List all available tags
   (2)         Dump all tags values in a file
   (3)         Dump custom tags values in a file
   ```

```
(4)         Get a tag
(5)         Set a tag
```

6. Select **(5) Set a tag**.
7. A prompt appears to enter the section name and tag name separated by spaces without quotes. Enter `org.quic.camera.videoHDRmode modeType`
8. A prompt appears to set the value. Enter 1 to switch to SHDR mode.

   To toggle between Linear and SHDR modes, repeat this process and enter 1 (SHDR mode) and 0 (Linear mode) as needed.

   To close the camera, select **(q) QUIT** or press **CTRL + C**.

   Verify this feature using the following logs:

   - Linear mode:

```
cam-server[1693]: CamX: [REQMAP]1693 2652 [CORE   ] camxsession.
cpp:4811
ProcessRequest() chiFrameNum: 0 <==> requestId: 1 <==>
sequenceId: 0
<==> CSLSyncId: 1 -- RealTimeFeatureZSLPreviewRawYUV_0
```

   - SHDR mode:

```
cam-server[1693]: CamX: [REQMAP]1693 2652 [CORE   ] camxsession.
cpp:4811
ProcessRequest() chiFrameNum: 296 <==> requestId: 297 <==>
sequenceId:
296 <==> CSLSyncId: 297 -- RealTimeSHDR_0
```

5 Troubleshoot camera

If the camera app doesn't work, check the following:

1. Check the camera module connection and DIP switch settings. See Getting started.
2. Restart the cam-server:

```
# systemctl restart cam-server

or:

# pkill cam-server
```

3. Run a single stream video recording use case:

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

gst-launch-1.0 -e qtiqmmfsrc name=camsrc camera=0 ! \
video/x-raw,format=NV12,width=1280,height=720,framerate=30/1,\
interlace-mode=progressive,colorimetry=bt601 ! v4l2h264enc \
capture-io-mode=4 output-io-mode=5 extra-controls="controls,
video_bitrate=6000000,\
video_bitrate_mode=0;" ! h264parse ! mp4mux ! filesink
location=/opt/mux_avc.mp4
```

4. Check the sensor probe.
   a. Collect logs using following command:

   ```
   # journalctl -f > /opt/log.txt
   ```

   b. Search for "probe success" in the log. Probe success means the camera module is powered up and responding to I2C control. If there is no 'probe success' log for a particular sensor, the flex cable connection or camera module may be the problem.

      The following log indicates one IMX577 (0x34), one OV9282 (0xc0), and two GMSL deserializer (0x90) are probed:

```
Jun 20 02:12:42 qcm6490 kernel: CAM_INFO: CAM-SENSOR:
cam_sensor_driver_cmd: 938: Probe success,slot:0,slave_addr:
0x34,sensor_id:0x577, is always on: 0 Jun 20 02:12:42 qcm6490
kernel: CAM_INFO: CAM-SENSOR:
cam_sensor_driver_cmd: 938: Probe success,slot:1,slave_addr:
0xc0,sensor_id:0x9281, is always on: 0 Jun 20 02:12:43
qcm6490 kernel: CAM_INFO: CAM-SENSOR:
cam_sensor_driver_cmd: 938: Probe success,slot:4,slave_addr:
0x90,sensor_id:0x94, is always on: 0 Jun 20 02:12:43 qcm6490
kernel: CAM_INFO: CAM-SENSOR:
cam_sensor_driver_cmd: 938: Probe success,slot:5,slave_addr:
0x90,sensor_id:0x94, is always on: 0
```

5. Check the camera sensor driver command.

   Collect logs and search for `cam_sensor_driver_cmd`. `CAM_START_DEV Success` indicates camera sensor streaming starts. `CAM_STOP_DEV Success` indicates camera sensor streaming stops. For example:

```
Jun 20 02:12:50 qcm6490 kernel: CAM_INFO: CAM-SENSOR:
cam_sensor_driver_cmd: 1017: CAM_ACQUIRE_DEV Success, sensor_id:
0x577,sensor_slave_addr:0x34, is always on: 0 Jun 20 02:12:50
qcm6490 kernel: CAM_INFO: CAM-SENSOR:
cam_sensor_driver_cmd: 1128: CAM_START_DEV Success, sensor_id:
0x577,sensor_slave_addr:0x34
Jun 20 02:12:53 qcm6490 kernel: CAM_INFO: CAM-SENSOR:
cam_sensor_driver_cmd: 1156: CAM_STOP_DEV Success,
sensor_id:0x577,sensor_slave_addr:0x34
Jun 20 02:12:53 qcm6490 kernel: CAM_INFO: CAM-SENSOR:
cam_sensor_driver_cmd: 1070: CAM_RELEASE_DEV Success, sensor_id:
0x577,sensor_slave_addr:0x34
```

6. Check sensor streaming.

   a. Enable CSID SOF/EOF IRQ logs:

```
# mount -o rw,remount /usr
# mount -t debugfs none /sys/kernel/debug/
# echo 0x8 > /sys/module/camera/parameters/debug_mdl
# echo 3 >/sys/kernel/debug/camera_ife/ife_csid_debug
# echo 1 > /sys/kernel/tracing/tracing_on
# echo 1 > /sys/kernel/tracing/events/camera/cam_log_debug/
enable # echo 2 > /sys/module/camera/parameters/debug_type
# cat /sys/kernel/tracing/trace_pipe > trace.txt
```

b. The captured logs can help provide the details about the SOF and EOF. Search for `irq_status_ipp` in the log (`trace.txt`). BIT12(0x1000) denotes an SOF packet and BIT9(0x200) denotes an EOF packet. The log should resemble the following:

```
<idle>-0       [000] d.h1. 19287.546764: cam_log_debug:
CAM_DBG: CAM-ISP: cam_ife_csid_irq: 4996: irq_status_ipp =
0x1110 cam-server-25604     [000] dNH.. 19287.561705: cam_
log_debug:
CAM_DBG: CAM-ISP: cam_ife_csid_irq: 4996: irq_status_ipp =
0xee8
```

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.