Qualcomm Linux Video Guide

80-70018-20 AA

April 9, 2025

Contents

1 Video overview

The Qualcomm® Adreno™ Video Processing Unit (VPU) is a hardware-accelerated video engine that supports video playback (decode) and recording (encode) for various video codecs. Applications can offload video operations to the Adreno VPU using the Qualcomm® Intelligent Multimedia SDK (IM SDK), or Video4Linux2 (V4L2) thereby freeing up the CPU and GPU for other operations.

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

- Video software architecture consists of user space, kernel space, and hardware modules
- User space consists of applications based on V4L2 or GStreamer
- Kernel space consists of V4L2 framework and Adreno VPU driver
- Hardware module consists of Adreno VPU firmware and hardware

The following figure shows the modules present in the video software architecture.

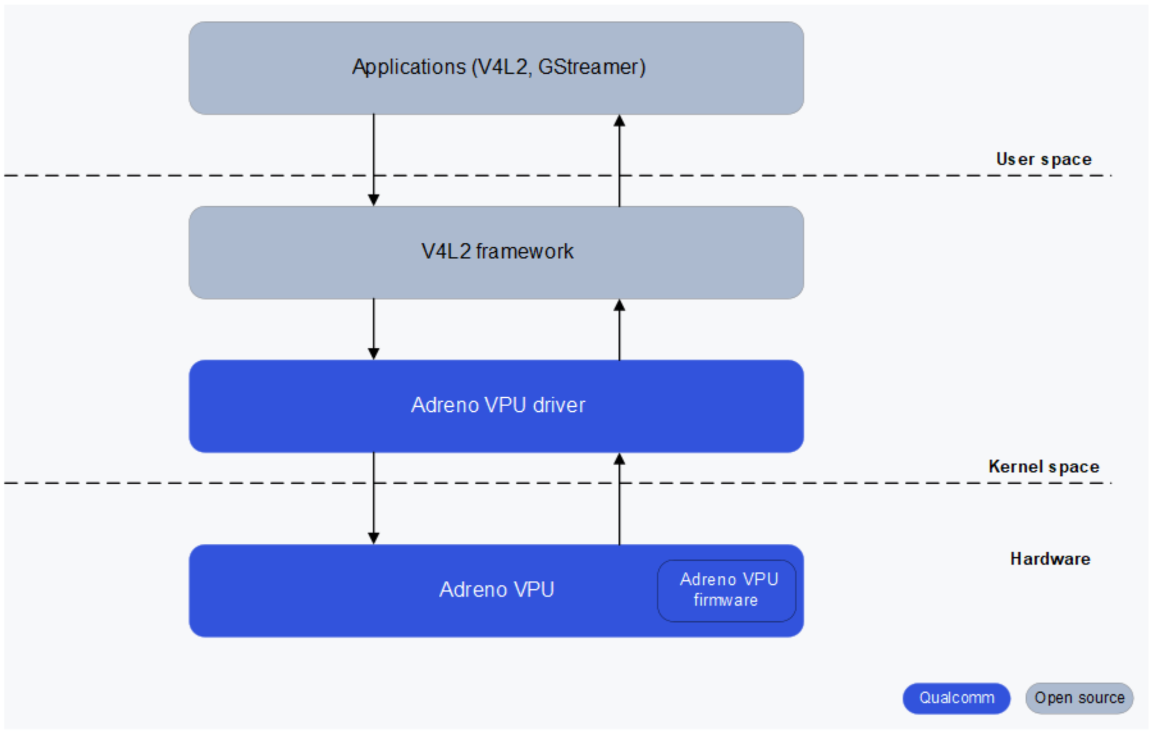

Figure : High-level architecture of the video software

2 Enable video

This section outlines the prerequisites for video functionality, steps to render video, and methods to validate playback and recording. The following workflow shows the sequence to get started with video functionality:

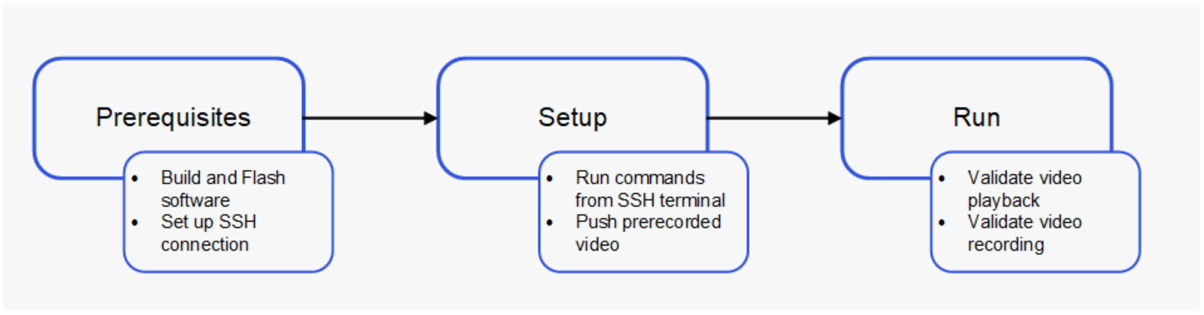

Figure : Getting started workflow for video

2.1 Prerequisites

- Build and flash the software. For information, see Qualcomm Linux Build Guide.
- Connect to device using SSH. For information, see Sign in using SSH.

2.2 Set up video rendering

The setup procedure helps you render the video content on the display, and transfer the video content to the device. Follow these steps from an SSH terminal:

1. Start Weston server manually in a shell. To start Weston server, see Launch Weston.
2. To render video on the display, run these commands in another shell:

```
mount -o rw,remount /
export XDG_RUNTIME_DIR=/dev/socket/weston
export WAYLAND_DISPLAY=wayland-1
```

3. For video playback, transfer a prerecorded H.264 video to the device using the following command:

```
scp -r <file_name> root@[DEVICE IP-ADDR]:/opt
```

Note: When prompted for a password, enter `oelinux123`.

2.3 Validate playback and video recording

You can run a set of commands on the device terminal to validate the video playback and recording functionality.

Validate video playback functionality

1. To validate the video playback functionality, run the command:

```
gst-launch-1.0 filesrc location=/opt/H264_480p_30fps.mp4 !
qtdemux ! h264parse ! v4l2h264dec capture-io-mode=4 output-io-
mode=4 ! video/x-raw,format=NV12 ! waylandsink fullscreen=true
```

2. To stop validating the use case, select **CTRL + C**.

Validate video recording functionality

1. To validate the video recording functionality, run the command:

```
gst-launch-1.0 -e qtiqmmfsrc ! video/x-raw, format=NV12,
width=1280,height=720,framerate=30/1,interlace-mode=progressive,
colorimetry=bt601 ! queue ! v4l2h264enc capture-io-mode=4
output-io-mode=4 extra-controls="controls,video_bitrate=1000000,
video_gop_size=29;" ! queue ! h264parse ! muxer. pulsesrc do-
timestamp=true provide-clock=false volume=10 ! audio/x-raw,
format=S16LE,channels=1,rate=48000 ! audioconvert ! queue !
lamemp3enc ! mp4mux name=muxer ! queue ! filesink location=/opt/
720p_AVC_MP3.mp4
```

2. To stop validating the use case, select **CTRL + C**.

3. Pull the recorded content out from the device using this SCP command:

```
scp -r root@[DEVICE IP-ADDR]:/opt/720p_AVC_MP3.mp4 user1@[HOST
IP-ADDR]:/workspace
```

4. Play the content on the Linux host computer.

3 Video architecture

The video software stack consists of a client application that interacts with the Adreno VPU using the V4L2 framework. The following figure shows the video software architecture for decoder and encoder. The image on the left shows the decoder architecture and the image on the right shows the encoder architecture.

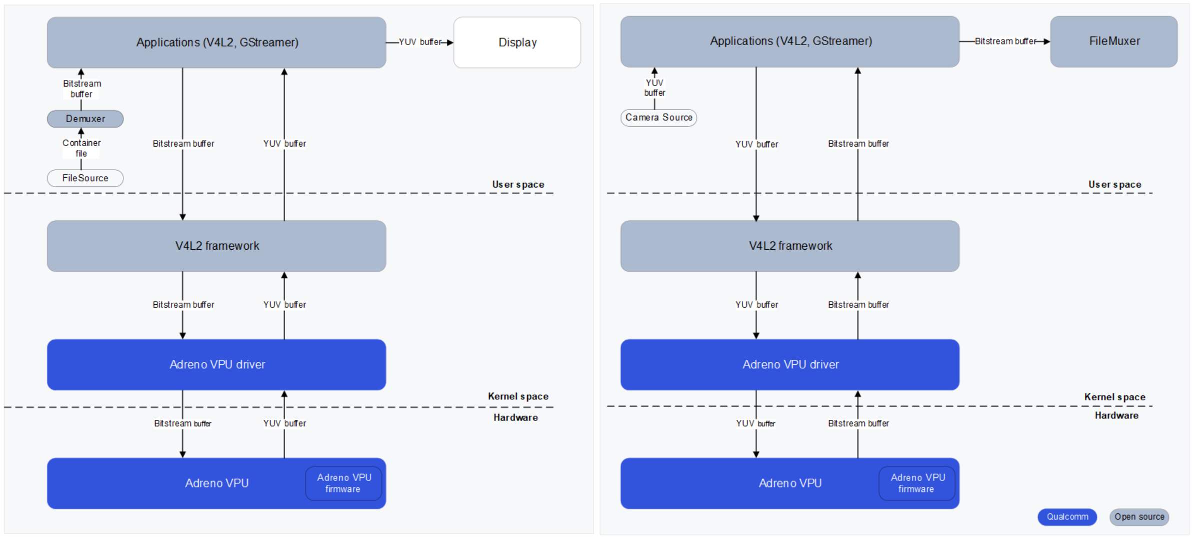

Figure : Video decoder and encoder software architecture

The following table lists the components of the video software architecture:

Component	**Description**
User application	Interacts with the FileSource for decoder and the Camera Source for encoder based on GStreamer/V4L2
V4L2 framework	• Interacts with user application and the Adreno VPU driver • Handles events, callbacks, and buffer management

Component	**Description**
Adreno VPU driver	• Acts as an entry point to the video kernel mode driver (KMD) • Common driver for the encoder and decoder that controls the Adreno VPU • Manages the video driver statemachine, buffer allocation, clock/bus management, and communication with Adreno VPU using the Host Firmware Interface (HFI)
Adreno VPU	Functions as a hardware accelerated engine that decodes and encodes a video with minimal power consumption

3.1 Video feature summary

The Adreno VPU provides the following video decoding and encoding features:

Table : Supported features

Use case	**Feature**	**QCS6490**	**QCS9075/QCS8275**
Codecs	H.264 (8-bit decoder)	✓	✓
	HEVC (8-bit decoder)	✓	✓
	HEVC (10-bit decoder)	✓	✓
	VP9 (8-bit decoder)	✓	✓
	VP9 (10-bit decoder)	✓	✓
	AV1 (8-bit decoder)	×	✓
	AV1 (10-bit decoder)	×	✓
	H.264 (8-bit encoder)	✓	✓
	HEVC (8-bit encoder)	✓	✓
Features	B-frame encode	✓	✓
	Encoder initial QP override	✓	✓
	Hierarchical-P encode	✓	×
	Slice encode	✓	×
	Intra-refresh	✓	×
	Preprocessing	✓	×
	Rate control	✓	✓
	Long term reference (LTR)	✓	×
	Dynamic properties	✓	✓
Concurrency	Multi-instance support	✓	✓

3.2 Video capabilities

QCS6490

The Adreno VPU on QCS6490 is a sixth-generation video processing unit that comes with the following capabilities:

Feature	**Capability**
Video decode	Up to 4096 × 2160 at 60 fps for H.264/HEVC/VP9
Video encode	Up to 4096 × 2160 at 30 fps for H.264/HEVC
Multichannel capability	• 8x of 1920 × 1088 at 30 fps decode • 16x of 1280 × 720 at 30 fps decode • 4x of 1920 × 1088 at 30 fps encode • 8x of 1280 × 720 at 30 fps encode
High frame rate capture	1280 × 720 at 480 fps or 1920 × 1088 at 240 fps
Maximum macroblocks per second	2088960 You can calculate the maximum macroblocks per second using the formula: (aligned width × aligned height × fps)/(macroblock size) For example, (4096 × 2176 × 60)/256=2088960

For more information on decode and encode capabilities for QCS6490, see Advanced video specifications.

QCS8275

The Adreno VPU on QCS8275 is a seventh-generation video processing unit that comes with the following capabilities:

Feature	**Capability**
Video decode	• Up to 3840 × 2160 at 120 fps for H.264/HEVC/VP9 • Up to 3840 × 2160 at 60 fps for AV1
Video encode	Up to 4096 × 2160 at 60 fps for H.264/HEVC
Multichannel capability	• 16x for 1920 × 1088 at 30 fps decode • 16x for 1280 × 720 at 30 fps encode
High framerate capture	1920 × 1088 at 240 fps or 3840 × 2160 at 60 fps
Maximum macroblocks per second	3916800 You can calculate the maximum macroblocks per second using the formula: (aligned width × aligned height × fps)/(macroblock size) For example, (3840 × 2176 × 120)/256 = 3916800

For more information on decode and encode capabilities for QCS8275, see Advanced video specifications.

QCS9075

The Adreno VPU on QCS9075 is a seventh-generation video processing unit that comes with the following capabilities:

Feature	**Capability**
Video decode	• Up to 7680 × 4320 at 60 fps or 3840 × 2160 at 240 fps for H.264/HEVC/AV1 • Up to 3840 × 2160 at 240 fps for VP9
Video encode	Up to 7680 × 4320 at 30 fps for H.264/HEVC
Multichannel capability	• 32x for 1920 × 1088 at 30 fps decode • 32x for 1280 × 720 at 30 fps encode • 24x for 1920 × 1088 at 30 fps encode • 8x for 3840 × 2160 at 30 fps decode
High frame rate capture	1920 x 1088 at 480 fps or 3840 × 2160 at 120 fps
Maximum macroblocks per second	7833600 You can calculate the maximum macroblocks per second using the formula: (aligned width × aligned height × fps)/(macroblock size) For example, (3840 × 2176 × 240)/256 = 7833600

For more information on decode and encode capabilities for QCS9075, see Advanced video specifications.

4 Build video source components

The video software stack consists of GStreamer plugins and V4L2 components.

- V4L2 source components are available in the `meta-qcom-multimedia` image as part of the entire software build package. For more information on the procedure to sync and build the `meta-qcom-multimedia` image, see Qualcomm Linux Build Guide.
- GStreamer plugins are part of Qualcomm IM SDK

The following sections provide information on GStreamer plugins and the V4L2 source build instructions.

4.1 V4L2 for video

The following is the information related to the source tree, VPU driver build instructions, and the commands that help push the video firmware.

Video source tree

The video software modules include the Adreno VPU driver, the video device tree, and the firmware binary.

- Adreno VPU driver BitBake recipe file.

Chip product	VPU driver BitBake recipe file path
QCS6490	`<workspacepace-path>/layers/meta-qcom-hwe/recipes-multimedia/video/qcom-videodlkm_1.0.bb`
QCS9075/QCS8275	

- Video device tree BitBake recipe file: The video device tree node is a part of the root device tree file.

Chip product	Device tree BitBake recipe file path
QCS6490	`<workspace-path>/layers/meta-qcom-hwe/recipes-multimedia/video/qcom-videodtb_1.0.bb`
QCS9075/QCS8275	

VPU driver build instructions

Use `devtool` to add the debug changes and compile the VPU driver (`iris_vpu.ko`). To build the VPU driver, follow the steps:

1. To access the source code for VPU drivers, run the BitBake commands:

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

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

   ```
   devtool modify qcom-videodlkm
   ```

2. Make the required debug changes to the VPU driver code available at: `<workspace-path>/build-qcom-wayland/workspace/sources/videodlkm`. After making the debug changes, build the VPU using the following command:

   ```
   devtool build qcom-videodlkm
   ```

3. After compilation, the corresponding `iris_vpu.ko` is available at: `<workspace-path>/build-qcom-wayland/workspace/sources/videodlkm/iris_vpu.ko`.

4. After you enter the SSH shell, transfer the compiled library to the device using this command:

   Note: You must start SSH to access your Linux host computer. For instructions, see Sign in using SSH.

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

5. Change the following command according to your `<workspace-path>`, and the `<IP address of the device>`, and run the command on the Linux host computer:

   ```
   scp <workspace-path>/build-qcom-wayland/workspace/sources/
   videodlkm/iris_vpu.ko root@<IP address of the device>/lib/
   modules/6.6.38-02695-g9d53679ea3d3/updates/
   ```

Note: When prompted for a password, enter `oelinux123`.

6. To see the updated debug changes, power OFF and then power ON the device.

Video device tree

The root device tree file includes the video device tree node. To access the VPU device tree, follow the steps:

1. To access the source code for the VPU device tree, use the following BitBake command:

```
devtool modify qcom-videodtb
```

2. Make the required debug changes to the VPU driver code available at: `<workspace-path>/build-qcom-wayland/workspace/sources/videodlkm`. After making the debug changes, build the VPU using the following command:

```
devtool build qcom-videodlkm
```

The VPU device tree code is available at:
`<workspace-path>/build-qcom-wayland/workspace/sources/videodtb`

Video firmware

Qualcomm provides the video firmware binary as a prebuilt image. Use the following commands to transfer the binary to your device:

Chip product	Command

QCS6490	`cd <workspace-path>/build-qcom-wayland/tmp-glibc/work/qcs6490_rb3gen2_vision_kit-qcom-linux/qcom-multimedia-image/1.0-r0/rootfs/lib/firmware/qcom/vpu-2.0`
	`scp vpu20_1v.mbn root@<IP address of the device>/lib/firmware/qcom/vpu-2.0/`
	`scp vpu20_1v_unsigned.mbn root@<IP address of the device>/lib/firmware/qcom/vpu-2.0/`

Chip product	Command

QCS9075/QCS8275	`cd <workspace-path>/build-qcom-wayland/tmp-glibc/work/qcs9100_ride_sx-qcom-linux/qcom-multimedia-image/1.0-r0/rootfs/lib/firmware/qcom/vpu-3.0`
	`scp vpu30_4v.mbn root@<IP address of the device>/lib/firmware/qcom/vpu-3.0/`
	`scp vpu30_4v_unsigned.mbn root@<IP address of the device>/lib/firmware/qcom/vpu-3.0/`

Note: When prompted for a password, enter `oelinux123`.

4.2 GStreamer plugins for video

The Qualcomm IM SDK includes GStreamer plugins for video decoder and encoder. Download the Qualcomm IM SDK to use the `v4l2h264dec`, `v4l2h265dec`, `v4l2h264enc`, `v4l2h265enc`, and the `v4l2vp9dec` GStreamer plugins.

The following table lists the GStreamer plugins and their corresponding reference links:

Plug-in	Description/reference link
`v4l2h264dec`	Decode the H.264 video stream
`v4l2h265dec`	Decode the H.265 video stream
`v4l2h264enc`	Encode the H.264 video stream
`v4l2h265enc`	Encode the H.265 video stream
`v4l2vp9dec`	Decode the VP9 video stream

For information on the steps to download and build, see Qualcomm Intelligent Multimedia Product (QIMP) SDK.

5 Explore video sample applications

You can use the video functionality at the native layer by using the Qualcomm IM SDK framework or by using the V4L2 interfaces. Qualcomm provides GStreamer and V4L2-based sample applications to run the video sample use cases.

5.1 GStreamer samples

The Qualcomm IM SDK includes sample GStreamer applications for video playback.

Note: For information on sample GStreamer applications supported on different SoCs, see Sample applications.

For information about the prerequisites for the GStreamer applications, see Run sample applications for video and audio.

The following table lists the methods and sample video test cases:

Table : Methods to test video use cases

Method	Use case
Using `gst-launch-1-0`	Video only playback
	Audio and video playback
	Video recording
Using the GStreamer application	Video playback and recording

5.2 V4L2 samples

The `iris_v4l2_test` sample video test application uses Linux V4L2 interfaces to simulate several basic encoder and decoder behaviors. For more information, see sample V4L2 application.

Qualcomm Linux supports the following video APIs:

- GStreamer APIs: The Qualcomm GStreamer plugins are open-source GStreamer framework compliant. For more information about the GStreamer based plugins, see Qualcomm IM SDK.

- V4L2 APIs: The Adreno VPU driver follows a standard V4L2-compliant call flow. The V4L2 APIs section provides information related to V4L2 APIs, commands, controls, and events.

5.3 Configure GStreamer and V4L2 APIs

- Qualcomm offers GStreamer-based plugins that you can integrate into your media pipeline. For information on the GStreamer-based plugins, see Configure Qualcomm GStreamer plugins.
- The Adreno VPU driver uses the V4L2 interface to support video encoding and decoding for all clients that follow the V4L2 guidance.

GStreamer APIs

For information on the GStreamer application development and pipeline creation, see Building an Application.

Configure GStreamer APIs

For information on configuring the GStreamer element property for different features and respective plugins, see Configure Qualcomm GStreamer plugins.

V4L2 APIs

This section describes V4L2 device nodes, file operations, commands, controls, and events for the decoder and encoder.

VPU driver nodes

The device registers the VPU driver during the boot-up process and assigns the device node number.

- VPU driver node for decoder: `/dev/video32`
- VPU driver node for encoder: `/dev/video33`

Operation	Description

V4L2 file operations

The Adreno VPU driver supports the following V4L2 file operations:

Operation	Description
`open`	Opens the video device file descriptor (FD) that's used to communicate with the VPU driver using the input/output control `ioctl` commands
`close`	Closes the video device upon the completion of a video use case, and releases all the VPU driver resources related to the video device
`poll`	Waits for the events from the VPU driver
`ioctl`	Sends commands or controls to the VPU device

V4L2 common commands for decoder and encoder

The VPU driver supports the following `ioctl` commands for the video decoder and encoder use cases:

- `VIDIOC_ENUM_FMT`
- `VIDIOC_TRY_FMT`
- `VIDIOC_G_FMT`
- `VIDIOC_S_FMT`
- `VIDIOC_ENUM_FRAMESIZES`
- `VIDIOC_REQBUFS`
- `VIDIOC_QUERY_BUF`
- `VIDIOC_CREATE_BUF`
- `VIDIOC_PREPARE_BUF`
- `VIDIOC_QBUF`
- `VIDIOC_DQBUF`
- `VIDIOC_STREAMON`
- `VIDIOC_STREAMOFF`
- `VIDIOC_QUERYCAP`
- `VIDIOC_QUERYCTRL`
- `VIDIOC_QUERYMENU`

- `VIDIOC_SUBSCRIBE_EVENT`
- `VIDIOC_UNSUBSCRIBE_EVENT`
- `VIDIOC_G_SELECTION`
- `VIDIOC_G_CTRL`
- `VIDIOC_S_CTRL`

V4L2 video decoder

This section provides information on the Adreno VPU decoder device nodes, supported controls, commands, and events. For the procedure to open a VPU decoder node, see VPU driver nodes.

V4L2 ioctl commands for decoder

For descriptions of the standard `ioctl` commands, see Video for Linux API Function Reference.

Apart from the common commands, the VPU driver also supports the following `ioctl` commands for the video decoder use cases:

- `VIDIOC_TRY_DECODER_CMD`
- `VIDIOC_DECODER_CMD`

As specified in the Memory-to-Memory Stateful Video Decoder Interface, the decoder supports all the mandatory commands required for the video decoder use cases.

V4L2 open-source controls for decoder

For descriptions of the standard open-source controls, see Codec Control Reference.

The VPU driver supports the following open-source controls for the video decoder use cases:

Control ID	**Purpose**	**Dynamic/Static**
`V4L2_CID_MPEG_VIDEO_H264_PROFILE`	H.264 decoder profile	Static
`V4L2_CID_MPEG_VIDEO_HEVC_PROFILE`	HEVC decoder profile	Static
`V4L2_CID_MPEG_VIDEO_VP9_PROFILE`	VP9 decoder profile	Static
`V4L2_CID_MPEG_VIDEO_H264_LEVEL`	H.264 decoder level	Static
`V4L2_CID_MPEG_VIDEO_HEVC_LEVEL`	HEVC decoder level	Static
`V4L2_CID_MPEG_VIDEO_VP9_LEVEL`	VP9 decoder level	Static
`V4L2_CID_MPEG_VIDEO_HEVC_TIER`	HEVC decoder tier	Static

The decoder supports the following V4L2 open-source events:

- `V4L2_EVENT_EOS`
- `V4L2_EVENT_SOURCE_CHANGE`
- `V4L2_EVENT_CTRL`

The decoder supports the following V4L2 commands:

- `V4L2_DEC_CMD_START`
- `V4L2_DEC_CMD_STOP`

V4L2 video encoder

This section provides the information on the Adreno VPU encoder device nodes, supported controls, commands, and events. For the procedure to open a VPU encoder node, see VPU driver nodes.

V4l2 ioctl commands for encoder

For descriptions of the standard `ioctl` commands, see Function Reference.

Apart from the common commands, the VPU driver also supports the following `ioctl` commands for the encode use cases:

- `VIDIOC_ENUM_FRAMEINTERVALS`
- `VIDIOC_S_SELECTION`
- `VIDIOC_S_PARM`
- `VIDIOC_G_PARM`
- `VIDIOC_TRY_ENCODER_CMD`
- `VIDIOC_ENCODER_CMD`

As specified in the Memory-to-Memory Stateful Video Encoder Interface, the encoder supports all the mandatory commands required for the video encoder use cases.

V4L2 open-source controls for encoder

For descriptions of the standard open-source controls, see Codec Control Reference.

The VPU driver supports the following open-source controls for the video encoder use cases:

Control ID	Purpose	Dynamic/Static
`V4L2_CID_MPEG_VIDEO_H264_PROFILE`	H.264 encoder profile	Static
`V4L2_CID_MPEG_VIDEO_HEVC_PROFILE`	HEVC encoder profile	Static
`V4L2_CID_MPEG_VIDEO_H264_LEVEL`	H.264 encoder level	Static
`V4L2_CID_MPEG_VIDEO_HEVC_LEVEL`	HEVC encoder level	Static

Control ID	Purpose	Dynamic/Static
`V4L2_CID_MPEG_VIDEO_HEVC_TIER`	H.264 encoder tier	Static
`V4L2_CID_HFLIP`	Horizontal flip	Both
`V4L2_CID_VFLIP`	Vertical flip	Both
`V4L2_CID_ROTATE`	Rotation	Static
`V4L2_CID_MPEG_VIDEO_HEADER_MODE`	Header mode	Static
`V4L2_CID_MPEG_VIDEO_PREPEND_SPSPPS_TO_IDR`	SPS/PPS with Instantaneous Decoder Refresh (IDR) frame	Static
`V4L2_CID_MPEG_VIDEO_FORCE_KEY_FRAME`	Request force sync frame	Both
`V4L2_CID_MPEG_VIDEO_BITRATE`	Average bit rate	Both
`V4L2_CID_MPEG_VIDEO_BITRATE_PEAK`	Peak bit rate	Both
`V4L2_CID_MPEG_VIDEO_BITRATE_MODE`	Rate control	Static
`V4L2_CID_MPEG_VIDEO_FRAME_SKIP_MODE`	Frame skip mode	Static
`V4L2_CID_MPEG_VIDEO_FRAME_RC_ENABLE`	RC enable/disable	Static
`V4L2_CID_MPEG_VIDEO_GOP_SIZE`	Group-of-pictures size	Both
`V4L2_CID_MPEG_VIDEO_B_FRAMES`	B-frames	Static
`V4L2_CID_MPEG_VIDEO_LTR_COUNT`	LTR frame count	Static
`V4L2_CID_MPEG_VIDEO_USE_LTR_FRAMES`	Use LTR	Both
`V4L2_CID_MPEG_VIDEO_FRAME_LTR_INDEX`	Mark LTR	Both
`V4L2_CID_MPEG_VIDEO_BASELAYER_PRIORITY_ID`	Set base layer ID for layer encode	Static
`V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD_TYPE`	Intra-refresh	Static
`V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD`	Intra-refresh period	Both
`V4L2_CID_MPEG_VIDEO_H264_MIN_QP`	H.264 minimum Quantization Parameter (QP)	Static

Control ID	Purpose	Dynamic/Static
`V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP`	HEVC minimum QP	Static
`V4L2_CID_MPEG_VIDEO_H264_MAX_QP`	H.264 maximum QP	Static
`V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP`	HEVC maximum QP	Static
`V4L2_CID_MPEG_VIDEO_H264_I_FRAME_MIN_QP`	H.264 I-frame minimum QP	Static
`V4L2_CID_MPEG_VIDEO_HEVC_I_FRAME_MIN_QP`	HEVC I-frame minimum QP	Static
`V4L2_CID_MPEG_VIDEO_H264_P_FRAME_MIN_QP`	H.264 P-frame minimum QP	Static
`V4L2_CID_MPEG_VIDEO_HEVC_P_FRAME_MIN_QP`	HEVC P-frame minimum QP	Static
`V4L2_CID_MPEG_VIDEO_H264_B_FRAME_MIN_QP`	H.264 B-frame minimum QP	Static
`V4L2_CID_MPEG_VIDEO_HEVC_B_FRAME_MIN_QP`	HEVC B-frame minimum QP	Static
`V4L2_CID_MPEG_VIDEO_H264_I_FRAME_MAX_QP`	H.264 I-frame maximum QP	Static
`V4L2_CID_MPEG_VIDEO_HEVC_I_FRAME_MAX_QP`	HEVC I-frame maximum QP	Static
`V4L2_CID_MPEG_VIDEO_H264_P_FRAME_MAX_QP`	H.264 P-frame maximum QP	Static
`V4L2_CID_MPEG_VIDEO_HEVC_P_FRAME_MAX_QP`	HEVC P-frame maximum QP	Static
`V4L2_CID_MPEG_VIDEO_H264_B_FRAME_MAX_QP`	H.264 B-frame maximum QP	Static
`V4L2_CID_MPEG_VIDEO_HEVC_B_FRAME_MAX_QP`	HEVC B-frame maximum QP	Static
`V4L2_CID_MPEG_VIDEO_HEVC_I_FRAME_QP`	HEVC I-frame QP	Both
`V4L2_CID_MPEG_VIDEO_H264_I_FRAME_QP`	H.264 I-frame QP	Both
`V4L2_CID_MPEG_VIDEO_HEVC_P_FRAME_QP`	HEVC P-frame QP	Both
`V4L2_CID_MPEG_VIDEO_H264_P_FRAME_QP`	H.264 P-frame QP	Both
`V4L2_CID_MPEG_VIDEO_HEVC_B_FRAME_QP`	HEVC B-frame QP	Both

Control ID	Purpose	Dynamic/Static
`V4L2_CID_MPEG_VIDEO_H264_B_FRAME_QP`	H.264 B-frame QP	Both
`V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_TYPE`	HEVC Hier coding type	Static
`V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_TYPE`	H.264 Hier coding type	Static
`V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING`	H.264 Hier coding	Static
`V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_LAYER`	HEVC Hier layer count	Both
`V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_LAYER`	H.264 Hier layer count	Both
`V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L0_BR`	H.264 layer 0-bit rate	Both
`V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L0_BR`	HEVC layer 0-bit rate	Both
`V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L1_BR`	H.264 layer 1-bit rate	Both
`V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L1_BR`	HEVC layer 1-bit rate	Both
`V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L2_BR`	H.264 layer 2-bit rate	Both
`V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L2_BR`	HEVC layer 2-bit rate	Both
`V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L3_BR`	H.264 layer 3-bit rate	Both
`V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L3_BR`	HEVC layer 3-bit rate	Both

Control ID	Purpose	Dynamic/Static
V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L4_BR	H.264 layer 4-bit rate	Both
V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L4_BR	HEVC layer 4-bit rate	Both
V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L5_BR	H.264 layer 5-bit rate	Both
V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L5_BR	HEVC layer 5-bit rate	Both
V4L2_CID_MPEG_VIDEO_H264_ENTROPY_MODE	H.264 entropy mode	Static
V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE	Encode slice mode	Static
V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_BYTES	Maximum bytes in slice	Static
V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_MB	Maximum macroblocks in slice	Static

Set all static configuration options after setting the formats on both the planes and before requesting buffers on either of the plane. Call the dynamic parameters after setting the formats on both planes.

For information on the V4L2 controls for encoder, see Codec Control Reference.

The supported open-source events for encoder are:

- `V4L2_EVENT_EOS`
- `V4L2_EVENT_CTRL`

The supported open-source commands for encoder are:

- `V4L2_ENC_CMD_START`
- `V4L2_ENC_CMD_STOP`

Configure V4L2 APIs

Use the V4L2 control IDs, accepted format, accepted ranges, and sample code information, provided in this section, to configure and manage different video features on your device.

B-frame encode

You can configure the VPU driver to set up B-frames in an encoder session using the `VIDIOC_S_CTRL` ioctl system call. Set the B-frame encode configuration before setting the `VIDIOC_STREAMON` on an output plane. The VPU driver supports B-frame encode only with Hierarchical coding. This feature doesn't support dynamic configuration.

Control ID for H.264 and HEVC codecs	**Accepted format**	**Range**
`V4L2_CID_MPEG_VIDEO_B_FRAMES`	Integer	• {0, 1} • 0: Disable B-frame • 1: Enable single B-frame

The following code sample shows the static configuration for a single B-frame:

```
v4l2_control control;
int ret = 0;
int bframes = 1;
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_B_FRAMES;
control.value = bframes;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_TYPE;
control.value = 0;// V4L2_MPEG_VIDEO_HEVC_HIERARCHICAL_CODING_B
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_LAYER;
control.value = 1;//Enable 1 layer
```

```
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
```

Encoder initial QP override

You can configure the VPU driver to override the initial frame Quantization Parameter (QP) for I-frames, P-frames, and B-frames to an intended value of encoded frames. You can override using the `VIDIOC_S_CTRL` ioctl system call. The QP override provides the flexibility to set the initial frame QP for I-frame only, or P-frame only, or B-frame only, or I-frame and P-frame only, or P-frame and B-frame only or combinations of these frames. Set the encoder initial QP override configuration before the `VIDIOC_STREAMON` on an output plane and is only applicable to an encoder session.

Control ID for H.264 codec	Accepted format	QP range
`V4L2_CID_MPEG_VIDEO_H264_I_FRAME_QP`	Integer	Example according to the codec standard: • 1 to 51 for 8-bit • 1 to 63 (-12 to 51) for 10-bit
`V4L2_CID_MPEG_VIDEO_H264_P_FRAME_QP`	Integer	
`V4L2_CID_MPEG_VIDEO_H264_B_FRAME_QP`	Integer	

Control ID for HEVC codec	Accepted format	QP range
`V4L2_CID_MPEG_VIDEO_HEVC_I_FRAME_QP`	Integer	Example according to the codec standard: • 1 to 51 for 8-bit • 1 to 63 (-12 to 51) for 10-bit
`V4L2_CID_MPEG_VIDEO_HEVC_P_FRAME_QPI`	Integer	
`V4L2_CID_MPEG_VIDEO_HEVC_B_FRAME_QP`	Integer	

The following code sample shows the static configuration parameter to choose:

- 40 as an initial QP for I-frame
- 35 as an initial QP for P-frame
- B-frame for H.264 session

```
v4l2_control control;
int ret = 0;
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_I_FRAME_QP;
control.value = 40;
```

```
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_P_FRAME_QP;
control.value = 35;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_B_FRAME_QP;
control.value = 35;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
```

Slice encode

You can configure the VPU driver to encode a single frame to many slices using the `VIDIOC_S_CTRL` ioctl system call. The number of bits per slice or the number of macroblocks per slice is a slice boundary. Set the slice encode configuration before the `VIDIOC_STREAMON` on an output plane. The slice encode feature doesn't support dynamic configuration, and this configuration is applicable only to an encoder session.

Control ID for H.264 and HEVC codecs	Accepted format	Range
`V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE`	enum `v4l2_mpeg_video_multi_slice_mode`	Values according to the codec standard: enum `v4l2_mpeg_video_multi_slice_mode`
`V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_BYTES`	Integer	512 - MAX_BITRATE/8 Unit: bits/second

Control ID for H.264 and HEVC codecs	Accepted format	Range
`V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_MB`	Integer	1 to 36864 Unit: Macroblocks

The following code sample shows the static configuration to encode frames with many slices and with the maximum number of macroblocks per slice set to 400:

```
v4l2_control control;
int ret = 0;
v4l2_mpeg_video_multi_slice_mode slicemode = V4L2_MPEG_VIDEO_MULTI_
SLICE_MODE_MAX_MB;
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE;
control.value = slicemode;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_MB;
control.value = 400;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
```

Intra-refresh

You can configure the VPU driver to activate the intra-refresh mode and intra-refresh period using the `VIDIOC_S_CTRL` ioctl system call. Set the intra-refresh configuration before the `VIDIOC_STREAMON` on an output plane. You can change the intra-refresh period dynamically during the session and this feature is applicable only to an encoder session.

Control ID for H.264 and HEVC codecs	**Accepted format**	**Range**
`V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD_TYPE`	enum `v4l2_mpeg_video_intra_refresh_period_type`	`{V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD_TYPE_RANDOM, V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD_TYPE_CYCLIC}` Random mode: `V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD_TYPE_RANDOM` Cyclic mode: `V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD_TYPE_CYCLIC`
`V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD`	Integer	Set the intra-refresh period to be less than an I-frame interval Unit: number of frames

The following code sample shows the static configuration to activate the random intra refresh for every 10 frames:

```
v4l2_control control;
int ret = 0;
v4l2_mpeg_video_intra_refresh_period_type refreshtype = V4L2_CID_
MPEG_VIDEO_INTRA_REFRESH_PERIOD_TYPE_RANDOM;

memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD_TYPE;
control.value = refreshtype;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD;
control.value = 10;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
```

Rotation

You can configure the VPU driver to rotate the input frame before encoding using the `VIDIOC_S_CTRL` ioctl system call. Set the rotation configuration before the `VIDIOC_STREAMON` on an output plane. The rotation feature doesn't support dynamic configuration and is applicable only to an encoder session.

Control ID for H264 and HEVC codecs	**Accepted format**	**Range**
`V4L2_CID_ROTATE`	Integer	{90, 180, 270} Unit: degrees

The following code sample shows the static configuration for rotating an input frame to 90 degrees:

```
v4l2_control control;
int ret = 0;
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_ROTATE;
control.value = 90;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
```

Frame QP range

You can configure the VPU driver to set the minimum and maximum frame QP range parameters for I-frames, P-frames, and B-frames to an intended value for encoded frames using the `VIDIOC_S_CTRL` ioctl system call. Set the frame QP range configuration before the `VIDIOC_STREAMON` on an output plane. The values are valid for an entire session. Dynamic configuration isn't supported, and the frame QP feature is applicable only to an encoder session.

Control ID for H.264 codec	Accepted format	Range
V4L2_CID_MPEG_VIDEO_H264_ I_FRAME_MIN_QP	Integer	Example according to the codec standard: 1 to 51 for 8-bit
V4L2_CID_MPEG_VIDEO_H264_ I_FRAME_MAX_QP		
V4L2_CID_MPEG_VIDEO_H264_ P_FRAME_MIN_QP		
V4L2_CID_MPEG_VIDEO_H264_ P_FRAME_MAX_QP		
V4L2_CID_MPEG_VIDEO_H264_ B_FRAME_MIN_QP		
V4L2_CID_MPEG_VIDEO_H264_ B_FRAME_MAX_QP		
V4L2_CID_MPEG_VIDEO_HEVC_ I_FRAME_MIN_QP		
V4L2_CID_MPEG_VIDEO_HEVC_ I_FRAME_MAX_QP		
V4L2_CID_MPEG_VIDEO_HEVC_ P_FRAME_MIN_QP		
V4L2_CID_MPEG_VIDEO_HEVC_ P_FRAME_MAX_QP		
V4L2_CID_MPEG_VIDEO_HEVC_ B_FRAME_MIN_QP		
V4L2_CID_MPEG_VIDEO_HEVC_ B_FRAME_MAX_QP		

The following code sample shows the static configuration parameter to choose 40 as the maximum QP and 10 as the minimum QP for an I-Frame, P-Frame, and B-Frame in a H.264 session:

```
v4l2_control control;
int ret = 0;
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_I_FRAME_MIN_QP;
control.value = 10;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_P_FRAME_MIN_QP;
control.value = 10;
```

```
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_B_FRAME_MIN_QP;
control.value = 10;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_I_FRAME_MAX_QP;
control.value = 40;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_P_FRAME_MAX_QP;
control.value = 40;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
memset(&control, 0, sizeof(control));
control.id = V4L2_CID_MPEG_VIDEO_H264_B_FRAME_MAX_QP;
control.value = 40;
ret = ioctl(mDriverFd, VIDIOC_S_CTRL, &control);// mDriverFd returned
from open() system call
if (ret) {
    return -1;
}
```

You can configure the following controls if you need to have the same minimum and maximum QP value for an I-frame, P-frame, and B-frame:

Control ID for H.264 codec	Accepted format	Range
`V4L2_CID_MPEG_VIDEO_H264_MIN_QP`	Integer	Example according to the code standard: 1 to 51 for 8-bit
`V4L2_CID_MPEG_VIDEO_H264_MAX_QP`		

Control ID for HEVC codec	Accepted format	Range
`V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP`	Integer	Example according to the code standard: 1 to 51 for 8-bit
`V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP`	Integer	

6 Troubleshoot video

This section outlines the process to verify the function of the VPU driver. The following workflow diagram shows the sequence to verify the VPU driver.

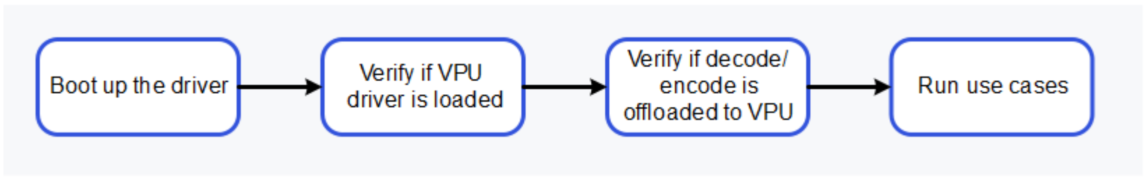
Boot up the driver
Verify if VPU driver is loaded
Verify if decode/ encode is offloaded to VPU
Run use cases

Figure : Workflow to verify the VPU driver

You can use the commands and samples provided in this section to verify the following:

- VPU driver is successfully loaded
- Decoding/encoding is offloaded to the VPU

6.1 Verify if VPU driver is loaded

The kernel image loads the VPU driver. You can use the `lsmod` or the `udevadm` commands to check if the VPU driver loads successfully.

Verify using the lsmod command

1. Run the `lsmod` command on the device shell:

```
lsmod | grep -i "iris"
```

2. To verify if the VPU driver is successfully loaded, review the sample output and ensure that an entry for `iris_vpu` is available under the Module column:

```
Module     Size     Used by

iris_vpu    585728     0
```

Verify using the udevadm command

1. Run the `udevadm` command on the device shell:

   ```
   udevadm info -n /dev/video*
   ```

2. To verify whether the VPU is successfully loaded, review if the following sample output shows `msm_vidc_decoder` and `msm_vidc_encoder` against the `ID_V4L_PRODUCT` tag:

Note: To identify the device nodes, check for the `DEVNAME` tag. In the earlier example, the `/dev/video32` is for the VPU decoder node and `/dev/video33` is for the VPU encoder node.

6.2 Verify if decoding/encoding is offloaded

To confirm if the video decoding or encoding is successfully offloaded to the VPU:

1. Run the command as a loop:

   ```
   cat /proc/interrupts | grep "msm-vidc"
   ```

   Sample output:

   ```
   253: 4        0         0         0         0         0         0
          0      GICv3 206 Level  msm-vidc
   ```

2. Verify if the interrupt count from `msm-vidc` increases across any CPU when the decoding/encoding use case is running.

6.3 Debug video

If you face an issue while running a video decode or encode use case, debugging can help identify the root cause. This information provides the methods to debug the VPU driver, firmware, and GStreamer video plugins.

Debug the VPU driver

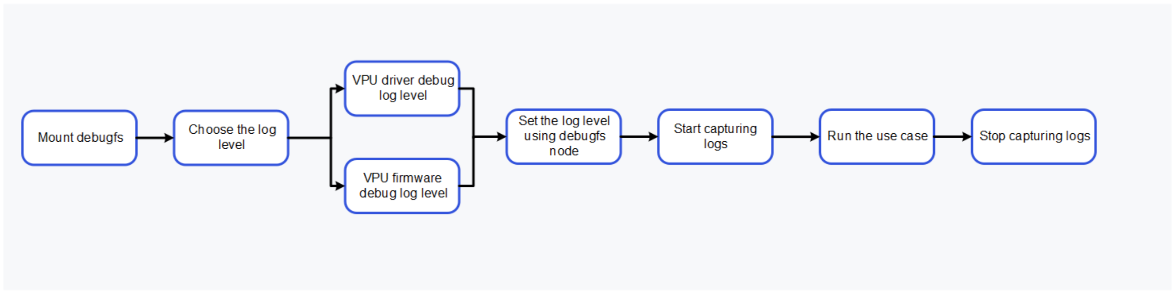
Mount debugfs
Choose the log level
VPU driver debug log level
VPU firmware debug log level
Set the log level using debugfs node
Start capturing logs
Run the use case
Stop capturing logs

Figure : Debug VPU workflow

Follow the steps to debug the VPU driver and VPU firmware:

1. Mount the `debugfs` node using the following command:

```
mount -t debugfs none /sys/kernel/debug/
```

2. Choose the VPU driver debug log level and debug log value as listed in the following table. You can also select many debug log levels by using the bitmask of debug log values.

Debug log level	Debug log value
VIDC_ERR	0x00000001
VIDC_HIGH	0x00000002
VIDC_LOW	0x00000004
VIDC_PERF	0x00000008
VIDC_PKT	0x00000010
VIDC_BUS	0x00000020
VIDC_STAT	0x00000040
VIDC_ENCODER	0x00000100
VIDC_DECODER	0x00000200
VIDC_PRINTK	0x10000000
VIDC_FTRACE	0x20000000

Note: The VPU driver sets the default debug level to `VIDC_ERR` and limits debug logs to error scenarios.

3. Set the VPU driver debug log value to the `/sys/module/iris_vpu/parameters/msm_vidc_debug` node.

   The sample command to activate all message levels:

```
echo 0x1000037F > /sys/module/iris_vpu/parameters/msm_vidc_debug
```

4. Choose the VPU firmware debug log level and debug log value as listed in the following table. You can also select many debug log levels by using the bitmask of debug log values.

Debug log level	Debug log value
FW_LOW	0x00000001
FW_MED	0x00000002
FW_HIGH	0x00000004
FW_ERROR	0x00000008
FW_FATAL	0x00000010
FW_PERF	0x00000020
FW_CACHE_LOW	0x00000100
FW_CACHE_MED	0x00000200
FW_CACHE_HIGH	0x00000400
FW_CACHE_ERROR	0x00000800
FW_CACHE_FATAL	0x00001000
FW_CACHE_PERF	0x00002000
FW_PRINTK	0x10000000
FW_FTRACE	0x20000000

Note: The VPU firmware debug logs are limited to error and fatal scenarios, and the default debug level is set to `FW_ERROR | FW_FATAL`.

5. Set the VPU firmware debug log value to the `/sys/module/iris_vpu/parameters/msm_fw_debug` node.

   The sample command to activate all message levels:

```
echo 0x1000037F > /sys/module/iris_vpu/parameters/msm_fw_debug
```

6. Start capturing kernel messages for both the VPU driver and the firmware using the following command and run the use case:

```
cat /proc/kmsg
```

Debug the GStreamer plugins

Qualcomm GStreamer video plugins are compliant with the standard GStreamer framework. For information on the debug methods for the GStreamer plugins, see Debug GStreamer Plugins.

7 Advanced video specifications

For more information on the video decode and encode capabilities such as the supported profiles, levels, resolutions, frame rates, bit rates, and limitations, select the following tabs:

Decoder standard	Supported profile and level	Minimum/Maximum resolution, Maximum frame rate, and bit rate	Maximum supported resolution, frame rate, and bit rate	Limitations/tools not supported

QCS6490

Table : Adreno VPU feature description for QCS6490

Feature	Description	Codecs	Remarks
Encoder input color formats	NV12 and QC08C	H.264 and HEVC	None
Decoder output color formats	NV12, QC08C, and QC10C	H.264, HEVC, and VP9	None
Rotation	Supports 90, 180, and 270-degree rotation before encoding the frame	H.264 and HEVC	Supports static rotation only
Flip	Supports horizontal and vertical flip before encoding the frame	H.264 and HEVC	Supports static and dynamic flip
B-frame encode	Up to 1920 × 1088 at 60 fps encode	H.264 and HEVC	The maximum number of B-frames supported between two P-frames is one
Hierarchical-P encode	Up to 5 layers	H.264 and HEVC	None
Initial QP override	Supports I-frames, P-frames, and B-frames	H.264 and HEVC	None
Slice encode	Yes	H.264 and HEVC	The number of bits per slice or the number of macroblocks per slice determines the slice boundary support
Intra-refresh	Random refresh mode	H.264 and HEVC	• Supported only in 8-bit encoding • Supported only in the CBR RC mode
Rate control	CBR, VBR, and MBR	H.264 and HEVC	None
LTR	2 frames	H.264 and HEVC	Supported in CBR RC mode
Dynamic properties for encoder	Sync frame, bit rate, and fps	H.264 and HEVC	Supported in CBR and VBR RC modes

Table : Adreno VPU decoder capabilities for QCS6490

Decoder standard	Supported profile and level	Minimum/Maximum resolution, Maximum frame rate, and bit rate	Maximum supported resolution, frame rate, and bit rate	Limitations/tools not supported
HEVC	• Main profile 8-bit up to level 5.1 • Main profile 10-bit, up to level 5.1, HLG schemes	• Minimum resolution: 96 × 96 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 480 fps • Maximum bit rate: 100 Mbps	• 1280 × 720 at 480 fps, 100 Mbps • 1920 × 1088 at 240 fps, 100 Mbps • 3840 × 2160 at 60 fps, 100 Mbps • 4096 × 2160 at 60 fps, 100 Mbps	• Maximum 128 slices per frame • Individual slice-based decoding
H.264	Constrained baseline, baseline, main, high, constrained high profiles; up to level 5.2	• Minimum resolution: 96 × 96 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 480 fps • Maximum bit rate: 100 Mbps	• 1280 × 720 at 480 fps, 100 Mbps • 1920 × 1088 at 240 fps, 100 Mbps • 3840 × 2160 at 60 fps, 100 Mbps • 4096 × 2160 at 60 fps, 100 Mbps	• Flexible macroblock order (FMO) • Arbitrary slice ordering (ASO) • Redundant slices (RS) • Data partition • Maximum 10 slices per frame • Interlaced content isn't supported
VP9	• Profile 0; 8-bit up to level 5.1 • Profile 2; 10-bit up to level 5.1 HLG/PQ schemes	• Minimum resolution: 96 × 96 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 480 fps • Maximum bit rate: 100 Mbps	• 1280 × 720 at 480 fps, 100 Mbps • 1920 × 1088 at 240 fps, 100 Mbps • 3840 × 2160 at 60 fps, 100 Mbps • 4096 × 2160 at 60 fps, 100 Mbps	Profile 2; 12-bit isn't supported

Table : Adreno VPU encoder capabilities for QCS6490

Encoder standard	Supported profile and level and RC modes	Minimum/Maximum resolution, maximum frame rate, and maximum bit rate	Supported resolution, frame rate, bit rate	Limitations/tools not supported
H.264	• Constrained baseline, baseline, main, high, constrained high profiles; up to level 5 • VBR, CBR, MBR	• Minimum resolution: 128 × 128 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 240 fps • Maximum bit rate: 100 Mbps	• 1280 × 720 at 240 fps, 100 Mbps • 1920 × 1088 at 120 fps, 100 Mbps • 3840 × 2160 at 30 fps, 100 Mbps • 4096 × 2160 at 30 fps, 100 Mbps	None

Encoder standard	**Supported profile and level and RC modes**	**Minimum/Maximum resolution, maximum frame rate, and maximum bit rate**	**Supported resolution, frame rate, bit rate**	**Limitations/tools not supported**
HEVC	• Main profile 8-bit, up to level 5.0 • Main/High tier VBR, CBR, MBR	• Minimum resolution: 128 × 128 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 240 fps • Maximum bit rate: 100 Mbps	• 1280 × 720 at 240 fps, 100 Mbps • 1920 × 1088 at 120 fps, 100 Mbps • 3840 × 2160 at 30 fps, 100 Mbps • 4096 × 2160 at 30 fps, 100 Mbps	Vertical tiling is only enabled for frame width 960

Feature	Description	Codecs	Remarks

QCS8275

Table : Adreno VPU feature description for QCS8275

Feature	Description	Codecs	Remarks
Encoder input color formats	NV12 and QC08C	H.264 and HEVC	None
Decoder output color formats	NV12, QC08C, and QC10C	H.264, HEVC, VP9 and AV1	None
B-frame encode	Up to 1920 × 1088 at 120 fps encode	H.264 and HEVC	The maximum number of B-frames supported between two P-frames is one
Initial QP override	Supports I-frames, P-frames, and B-frames	H.264 and HEVC	None
Rate control	CBR and VBR	H.264 and HEVC	None
Dynamic properties for encoder	Sync frame, bit rate, and fps	H.264 and HEVC	Supported in CBR and VBR RC modes

Note: End-to-end functionality using IM SDK is validated up to 3840 × 2160 resolution.

Table : Adreno VPU decoder capabilities for QCS8275

Decoder standard	Supported profile and level	Minimum/Maximum resolution, maximum frame rate, and maximum bit rate	Maximum supported resolution, frame rate, and bit rate	Limitations/tools not supported
HEVC	• Main profile 8-bit, up to level 5.1 Main tier, high tier • Main10, profile up to level 5.1 Main/High tier, HLG schemes	• Minimum resolution: 96 × 96 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 480 fps • Maximum bit rate: 160 Mbps	• 1280 × 720 at 480 fps, 160 Mbps (IBP/IPP) • 1920 × 1088 at 480 fps, 160 Mbps (IBP/IPP) • 3840 × 2160 at 120 fps, 160 Mbps (IBP/IPP) • 4096 × 2160 at 60 fps, 120 Mbps (IBP/IPP)	Individual slice-based decoding

Decoder standard	**Supported profile and level**	**Minimum/Maximum resolution, maximum frame rate, and maximum bit rate**	**Maximum supported resolution, frame rate, and bit rate**	**Limitations/tools not supported**
H.264	Constrained baseline, Baseline, main, high, constrained high profiles; up to level 5.2	• Minimum resolution: 96 × 96 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 480 fps • Maximum bit rate: 160 Mbps	• 1280 × 720 at 480 fps, 160 Mbps (IBP/IPP) • 1920 × 1088 at 480 fps, 160 Mbps (IBP/IPP) • 3840 × 2160 at 120 fps, 160 Mbps (IBP/IPP) • 4096 × 2160 at 60 fps, 120 Mbps (IBP/IPP)	• Flexible macroblock order (FMO) • Arbitrary slice ordering (ASO) • Redundant slices (RS) • Data partition • Individual slice-based decoding • Non-progressive-only content up to 1920 x 1088 • Best effort B-frame decode is: 3840 × 2160 at 120 fps

Decoder standard	**Supported profile and level**	**Minimum/Maximum resolution, maximum frame rate, and maximum bit rate**	**Maximum supported resolution, frame rate, and bit rate**	**Limitations/tools not supported**
VP9	• Profile 0; 8-bit, up to level 5.1 • Profile2, 10-bit, up to level 5.1, HLG/PQ schemes	• Minimum resolution: 96 × 96 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 120 fps • Maximum bit rate: 50 Mbps	• 1280 × 720 at 120 fps, 50 Mbps • 1920 × 1088 at 120 fps, 50 Mbps • 3840 × 2160 at 120 fps, 50 Mbps • 4096 × 2160 at 60 fps, 50 Mbps	• Profile 2, 12-bit isn't supported • Individual slice-based decoding

Decoder standard	**Supported profile and level**	**Minimum/Maximum resolution, maximum frame rate, and maximum bit rate**	**Maximum supported resolution, frame rate, and bit rate**	**Limitations/tools not supported**
AV1	• Main (Profile-0) • Maximum level: 5.1 • HLG/PQ schemes	• Minimum resolution: 96 × 96 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 240 fps • Maximum bit rate: 120 Mbps	• 1280 × 720 at 240 fps, 120 Mbps • 1920 × 1088 at 240 fps, 120 Mbps • 3840 × 2160 at 60 fps, 120 Mbps • 4096 × 2160 at 60 fps, 120 Mbps	Individual slice-based decoding

Table : Adreno VPU encoder capabilities for QCS8275

Encoder standard	**Supported profile, level, and RC mode**	**Minimum/Maximum resolution, maximum frame rate, and maximum bit rate**	**Maximum supported resolution, frame rate, and bit rate**	**Limitations/tools not supported**
H.264	• Constrained baseline, baseline, main, high, constrained high profiles; up to level 5.2 • VBR and CBR	• Minimum resolution: 128 × 128 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 480 fps • Maximum bit rate: 160 Mbps • Maximum operating rate: 480	• 1280 × 720 at 480 fps, 88 Mbps • 1920 × 1088 at 240 fps, 128 Mbps • 3840 × 2160 at 60 fps, 80 Mbps • 4096 × 2160 at 60 fps, 92 Mbps	

Encoder standard	**Supported profile, level, and RC mode**	**Minimum/Maximum resolution, maximum frame rate, and maximum bit rate**	**Maximum supported resolution, frame rate, and bit rate**	**Limitations/tools not supported**
HEVC	Main profile 8-bit, up to level 5.1 Main/High tier VBR and CBR	• Minimum resolution: 96 × 96 • Maximum resolution: 4096 × 2160 or 2160 × 4096 • Maximum frame rate: 480 fps • Maximum bit rate: 160 Mbps • Maximum operating rate: 480 • Maximum frame rate: 480 fps • Maximum bit rate: 160 Mbps	• 1280 × 720 at 480 fps, 62 Mbps • 1920 × 1088 at 240 fps, 90 Mbps • 3840 × 2160 at 60 fps, 56 Mbps • 4096 × 2160 at 60 fps, 64 Mbps	• Multislice is enabled

QCS9075

Table : Adreno VPU multichannel capabilities for QCS9075

Multichannel/Resolution/ fps/Codec	Use case combination	Recommended bit rate per session (Mbps)			
		H.264 (CAVLC)	H.264 (CABAC)	HEVC	AV1 (Decoder only)
24x for 1920 × 1088 at 30 fps, any supported codec combination	Encode only	9.17	7.92	7.92	5
32x for 1920 × 1088 at 30 fps, any supported codec combination	Decode only	6.88	5.94	5.94	–
32x for 1920 × 1088 at 30 fps decode + 1280 × 720 at 30 fps encode, any supported codec combination	Decode/Encode	7/5	7/5	7/5	–
32x for 1280 × 720 at 30 fps, any supported codec combination	Encode only	6.88	5.94	5.94	–

Table : Adreno VPU feature description for QCS9075

Feature	Description	Codecs	Remarks
Encoder input color formats	NV12 and QC08C	H.264 and HEVC	None
Decoder output color formats	NV12, QC08C, and QC10C	H.264, HEVC, VP9 and AV1	None
B-frame encode	Up to 3840 × 2160 at 60 fps encode	H.264 and HEVC	The maximum number of B-frames supported between two P-frames is one
Initial QP override	Supports for I-frames, P-frames, and B-frames	H.264 and HEVC	None
Rate control	CBR and VBR	H.264 and HEVC	None
Dynamic properties for encoder	Sync frame, bit rate, and fps	H.264 and HEVC	Supported in CBR and VBR RC modes

Note: End-to-end functionality using Qualcomm IM SDK is validated up to

3840 $\times$ 2160 resolution.

Decoder standard	Supported profile and level	Minimum/maximum resolution, Maximum frame rate, and bit rate	Maximum supported resolution, frame rate, and bit rate	Limitations/tools not supported

Table : Adreno VPU decoder capabilities for QCS9075

Decoder standard	Supported profile and level	Minimum/maximum resolution, Maximum frame rate, and bit rate	Maximum supported resolution, frame rate, and bit rate	Limitations/tools not supported
HEVC	• Main profile 8-bit, up to level 6.2, main tier, high tier • Main10 profile up to level 6.2 Main/High tier HLG schemes	• Minimum resolution: 96 × 96 • Maximum resolution: 8192 × 4320 or 4320 × 8192 • Maximum frame rate: 480 fps • Maximum bit rate: 160 Mbps	• 1280 × 720 at 480 fps, 160 Mbps (IBP/IPP) • 1920 × 1088 at 480 fps, 160 Mbps (IBP/IPP) • 3840 × 2160 at 240 fps, 80 Mbps (IBP/IPP) • 3840 × 2160 at 120 fps, 160 Mbps (IBP/IPP) • 4096 × 2160 at 120 fps, 80 Mbps (IBP/IPP) • 7680 × 4320 at 30 fps, 120 Mbps (IBP/IPP) • 7680 × 4320 at 60 fps, 80 Mbps (IBP/IPP) • 8192 × 4320 at 30 fps,	Individual slice-based decoding

Decoder standard	**Supported profile and level**	**Minimum/maximum resolution, Maximum frame rate, and bit rate**	**Maximum supported resolution, frame rate, and bit rate**	**Limitations/tools not supported**
H.264	Constrained baseline, baseline, main, high, constrained high profiles; up to level 6.1	• Minimum resolution: 96 × 96 • Maximum resolution: 8192 × 4320 or 4320 × 8192 • Maximum frame rate: 480 fps • Maximum bit rate: 220 Mbps (CAVLC), 160 Mbps (CABAC)	• 1280 × 720 at 480 fps, 160 Mbps (IBP/IPP) • 1920 × 1088 at 480 fps, 160 Mbps (IBP/IPP) • 3840 × 2160 at 240 fps, 80 Mbps (IPP) • 3840 × 2160 at 120 fps, 160 Mbps (IBP/IPP) • 4096 × 2160 at 120 fps, 80 Mbps (IPP) • 7680 × 4320 at 60 fps, 80 Mbps (IPP) • 7680 × 4320 at 30 fps, 120 Mbps (IBP/IPP) • 8192 × 4320 at 30 fps, 120 Mbps (IPP) • 8192 × 4320 at 48 fps, 80 Mbps (IPP)	• Flexible macroblock order (FMO) • Arbitrary slice ordering (ASO) • Redundant slices (RS) • Data partition • Individual slice-based decoding • Non-progressive-only content up to 1920 × 1088 • Best effort B-frame decodes are: – 3840 × 2160 at 240 fps – 4096 × 2160 at 120 fps – 7680 × 4320 at 60 fps – 8192 × 4320 at 30/48 fps

Decoder standard	**Supported profile and level**	**Minimum/maximum resolution, Maximum frame rate, and bit rate**	**Maximum supported resolution, frame rate, and bit rate**	**Limitations/tools not supported**
VP9	• Profile 0; 8-bit up to level 5.1 • Profile2, 10-bit, up to level 5.1, HLG/PQ schemes	• Minimum resolution: 96 × 96 • Maximum resolution: 4096 × 2304 or 2304 × 4096 • Maximum frame rate: 120 fps • Maximum bit rate: 100 Mbps	• 1280 × 720 at 120 fps, 100 Mbps • 1920 × 1088 at 120 fps, 100 Mbps • 3840 × 2160 at 60 fps, 100 Mbps • 3840 × 2160 at 240 fps, 30 Mbps • 4096 × 2160 at 60 fps, 100 Mbps • 4096 × 2304 at 60 fps, 100 Mbps	Profile 2, 12-bit isn't supported

Decoder standard	**Supported profile and level**	**Minimum/maximum resolution, Maximum frame rate, and bit rate**	**Maximum supported resolution, frame rate, and bit rate**	**Limitations/tools not supported**
AV1	• Main (Profile-0), 8-bit and 10-bit • Maximum level: 6.1 • HLG/PQ schemes	• Minimum resolution: 96 × 96 • Maximum resolution: 8192 × 4320 or 4320 × 8192 • Maximum frame rate: 480 fps • Maximum bit rate: 120 Mbps	• 1280 × 720 at 480 fps, 120 Mbps • 1920 × 1088 at 480 fps, 120 Mbps • 3840 × 2160 at 240 fps, 120 Mbps • 3840 × 2160 at 120 fps, 120 Mbps • 4096 × 2160 at 120 fps, 120 Mbps • 7680 × 4320 at 30 fps, 120 Mbps • 7680 × 4320 at 60 fps, 120 Mbps • 8192 × 4320 at 30 fps, 120 Mbps • 8192 × 4320 at 48 fps, 120 Mbps	Individual slice-based decoding

Table : Adreno VPU encoder capabilities for QCS9075

Encoder standard	Supported profile, level, and RC modes	Minimum/Maximum resolution, maximum frame rate, and maximum bit rate	Supported resolution, frame rate, bit rate	Limitations/tools not supported
H.264	• Constrained baseline, baseline, main, high, constrained high profiles; up to level 6.0 • VBR and CBR	• Minimum resolution: 128 × 128 • Maximum resolution: 8192 × 4320 or 4320 × 8192 • Maximum frame rate: 480 fps • Maximum bit rate: 220 Mbps (CAVLC), 160 Mbps (CABAC)	• 1280 × 720 at 480 fps, 160 Mbps • 1920 × 1088 at 480 fps, 160 Mbps • 3840 × 2160 at 120 fps, 160 Mbps • 7680 × 4320 at 30 fps, 160 Mbps	Individual encoded slice delivery per buffer

Encoder standard	Supported profile, level, and RC modes	Minimum/Maximum resolution, maximum frame rate, and maximum bit rate	Supported resolution, frame rate, bit rate	Limitations/tools not supported
HEVC	• Main profile 8bit, up to level 6.0, 6.1, Main/High tier • VBR and CBR	• Minimum resolution: 96 × 96 • Maximum resolution: 8192 × 4320 or 4320 × 8192 • Maximum frame rate: 480 fps • Maximum bit rate: 160 Mbps	• 1280 × 720 at 480 fps, 160 Mbps • 1920 × 1088 at 480 fps, 160 Mbps • 3840 × 2160 at 120 fps, 160 Mbps • 7680 × 4320 at 30 fps, 160 Mbps	• Individual encoded slice delivery per buffer • Multislice is enabled

7.1 Feature descriptions

The supported encoder feature descriptions are as follows:

B-frame encode

B-frame uses both the earlier and the future frames as reference data to obtain the highest amount of data compression. The Adreno VPU encodes frames with adaptive B type to obtain the highest possible compression without compromising video quality.

Encoder initial QP override

Video encoding compresses signal levels by mapping them to discrete values. Quantization is a lossy process, and the levels of quantization govern the quality compared to compression. Encoders start with a default Quantization Parameter (QP) at the beginning. Based on the configured bit rate and scene complexity, encoders determine the right QP value by continuously monitoring the complexity and redundancy across frames. The encoder takes a few seconds to reach a steady state and predict the correct QP value that matches the target bit rate, also known as rate convergence.

Hierarchical-P encode

In the Hierarchical-P (Hier-P) feature, the encoder organizes the frames into many layers, with frames of one layer referencing frames only from the lower layers. The lowest layer, also known as layer 0 or the base layer, is the only exception.

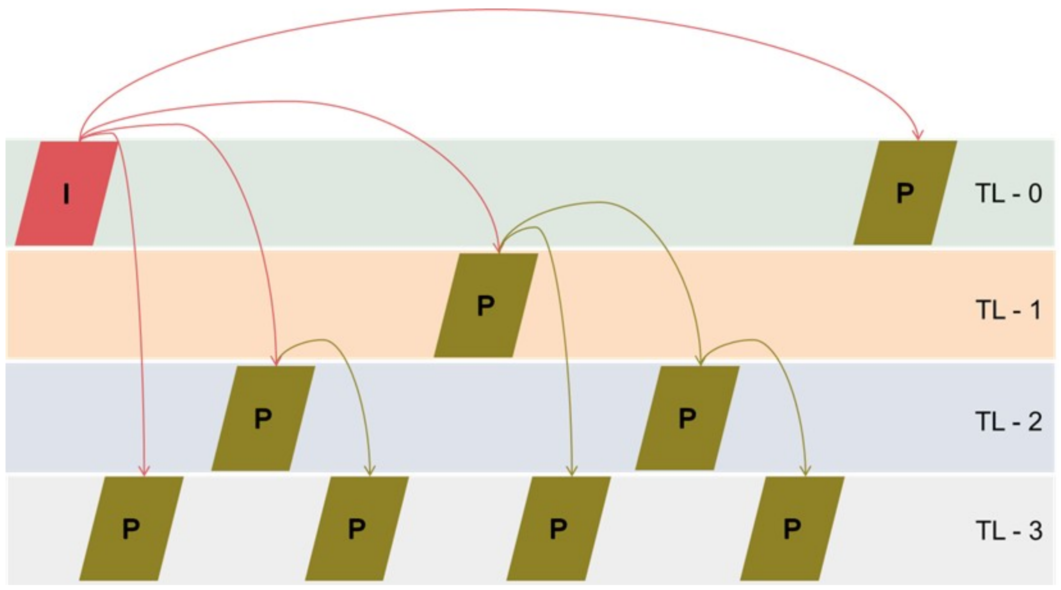

Figure : Hier-P layer encoding pattern

In this figure, TL-0 represents the base layer, and the remaining layers represent the enhancement layers. Hier-P improves error resilience and temporal scalability. The Hier-P feature is useful for video telephony (VT) or videoconferencing applications that involve channel errors. Hier-P allows you to control error propagation by selectively dropping the enhancement layers.

Slice encode

Encoders can compress a frame with an independently decodable Group-of-blocks (GOB), also known as slices. If there is a data loss or corruption, each slice is independently decodable and intends to be a unit of recovery. The advantages of introducing slices in an encoded frame are:

- Encoder ignores a corrupt slice and skips to the next slice, thus restricting the corruption to a part of the frame instead of the entire frame.
- Encoder sizes the slices to fit them within a network packet to help with transmission.
- Encoder retransmits erroneous slices instead of sending the entire frame.
- Applications use slices to reduce latency in real-time communication. Applications transmit and decode slices in parallel, eliminating the need to wait for the entire frame to encode.

Slices also work as resynchronization markers because the decoders can resume from the next slice (marker) when there are bit errors. The H.264 and HEVC encoders support slicing on Qualcomm Linux. The number of bits per slice or the number of macroblocks per slice is a slice boundary.

Intra-refresh

The intra-refresh feature reduces the channel loss in streaming and casting applications that favor a constant bit rate. The Adreno VPU supports random intra-refresh mode.

Video encoder preprocessing

Applications can use the Adreno VPU to rotate or flip a YUV frame before encoding it. The Adreno VPU rotates or flips the YUP frame without consuming extra power.

Rate control

The following table lists the supported rate control algorithms:

Rate control mode	Description
Variable bit rate (VBR)	• Minimizes the frame-by-frame video quality fluctuation • Camcorder and Wi-Fi display are the example use cases
Constant bit rate (CBR)	• Reduces bit rate fluctuation • Used for real-time communication with channel bandwidth limitation • Video telephony and streaming are the example use cases
Maximum bit rate (MBR)	• Limits the bit rate while maintaining flexibility and may bounce up and down within the set target • Bit rate increases when the activity in a scene increases within a maximum limit • Integrated with a smart bit allocation (SBA) feature to achieve better quality at a lower bit rate

Long-Term Reference (LTR) support

Video compression works by eliminating redundancies within the frame (intra-frame) and between the frames (inter-frame). Earlier, the encoded frames that used to serve as a basis to derive future frames were known as reference frames.

The following reference frames allow advanced encoding applications to control the way a reference frame is stored and referred:

- **Short-Term Reference (STR)**: The encoder maintains the recent frames in a reference buffer list from the newest to the oldest. The encoder automatically manages frames using STRs for reference, and deletes them from a stored list when they're no longer used.

- **Long-Term Reference (LTR)**: Frames that the application can save, use, and remove. The LTR frames help improve quality and ensure error resiliency in video communication. The maximum number of frames that can be marked with LTR frames depends on the device capability.

LTR frames are useful in error-prone channels. Referring to LTR in error-prone channels reduces the possibility of drift errors due to channel losses. The receiver must confirm that the LTR is received successfully and that it can request a new LTR when an error occurs. The network protocols have checksums that can confirm this behavior, along with the decoder corruption flags. New LTR frames are generated from the sender until the receiver confirms that a successful LTR is received.

LTR frames are also useful in videos with scene changes where an LTR with the earlier scene is preserved. If that scene comes back, then the LTR can be used effectively.

At the start of a new Group-of-pictures (GOP), the encoder automatically fills the LTR slots, with the first slot (slot number 0) containing an IDR frame. Applications can explicitly send an LTR mark request to mark the LTR frames in the appropriate slots. The following image shows the flow diagram for LTR usage, how to mark and use the LTR frames on an H.264 or HEVC encoder:

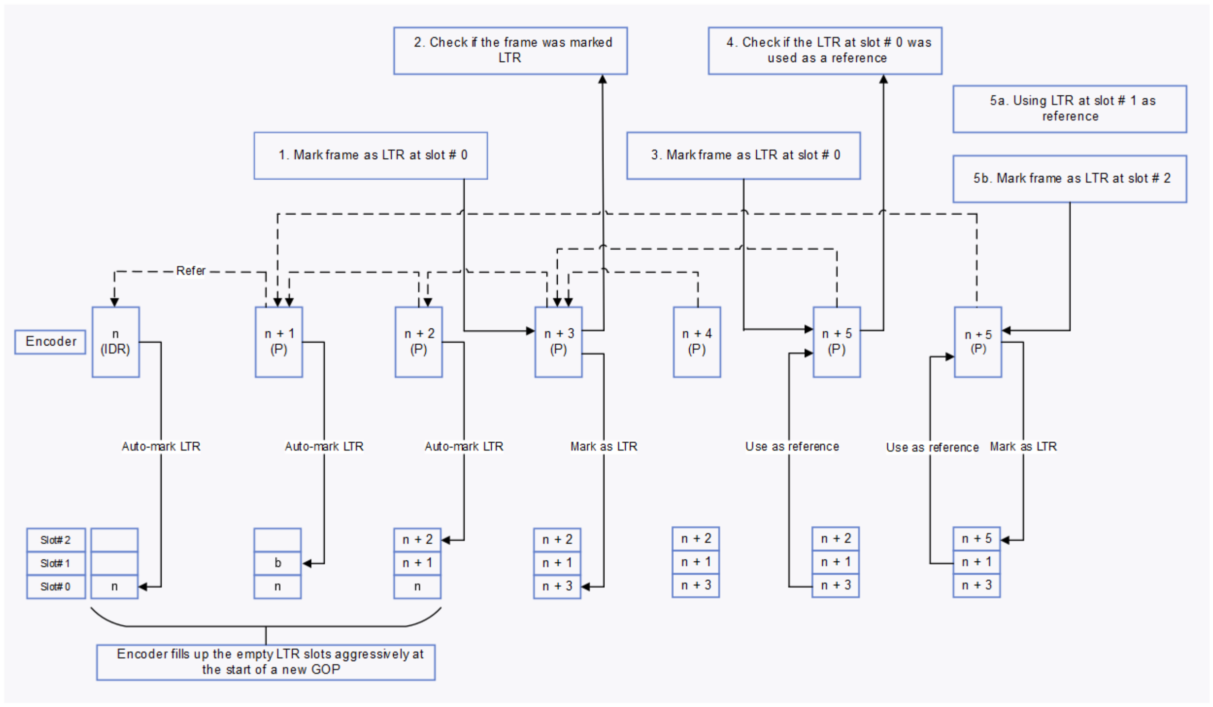
2. Check if the frame was marked LTR
4. Check if the LTR at slot # 0 was used as a reference
5a. Using LTR at slot # 1 as reference
1. Mark frame as LTR at slot # 0
3. Mark frame as LTR at slot # 0
5b. Mark frame as LTR at slot # 2
Refer
Encoder
n (IDR)
n + 1 (P)
n + 2 (P)
n + 3 (P)
n + 4 (P)
n + 5 (P)
n + 5 (P)
Auto-mark LTR
Auto-mark LTR
Auto-mark LTR
Mark as LTR
Use as reference
Use as reference
Mark as LTR
Slot# 2
Slot# 1
Slot# 0
n
b
n
n + 2
n + 1
n
n + 2
n + 1
n + 3
n + 2
n + 1
n + 3
n + 2
n + 1
n + 3
n + 5
n + 1
n + 3
Encoder fills up the empty LTR slots aggressively at the start of a new GOP

Figure : LTR encoding

Dynamic encoder properties

The Adreno VPU encoder supports dynamic change of properties such as bit rate, frame rate, and sync frame. This support allows the application to change the properties and helps in improved visual experience, video data adjusting to network conditions, and minimizing the loss of data during transmission.

8 References

8.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
Qualcomm Intelligent Multimedia Software Development Kit	80-70018-50
Qualcomm Intelligent Multimedia Product (QIMP) SDK	80-70018-51
Qualcomm Linux Build Guide	80-70018-254
Resources	
https://github.com/quic/v4l-video-test-app	
https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/v4l2.html	
https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/user-func.html	
https://www.kernel.org/doc/html/v5.5/media/uapi/v4l/ext-ctrls-codec.html	
https://www.kernel.org/doc/html/latest/userspace-api/media/v4l/dev-decoder.html	
https://www.kernel.org/doc/html/latest/userspace-api/media/v4l/dev-encoder.html	

8.2 Acronyms and terms

Acronym or term	Definition
ASO	Arbitrary slice ordering
CBR	Constant bit rate
FD	File descriptor
FMO	Flexible macroblock order
GOB	Group of blocks
GOP	Group of pictures
GPU	Graphics processing unit
HEVC	High efficiency video coding
HFI	Host firmware interface

Acronym or term	**Definition**
LTR	Long-term reference
MBR	Maximum bit rate
QP	Quantization parameter
RS	Redundant slices
STR	Short-term reference
VBR	Variable bit rate
VPU	Video processing unit
VT	Video telephony
YUV	Luminance, bandwidth, chrominance, also known as YCbCr and YPbPr