i.MX 8 GStreamer User Guide

Version 2.0, September, 2020

Table of Contents

Chapter 1. Introduction

This document is a user guide for the GStreamer version 1.0 based accelerated solution included in all the i.MX 8 family SoCs supported by NXP BSP L5.4.24_2.1.1.

Some instructions assume a host machine running a Linux distribution, such as Ubuntu, connected to an i.MX 8 device. These commands were tested using Ubuntu 18.04 LTD, and while Ubuntu is not required on the host machine, other distributions have not been tested.

These instructions are targeted for use with the following hardware:

- i.MX 8M Quad EVK
- i.MX 8M Mini EVK
- i.MX 8M Nano EVK
- i.MX 8QuadXPlus MEK B0
- i.MX 8QuadMax MEK B0

1.1. Release History

Table 1. Release history

Version	Release Data	Description
v1.0	Mar 2020	Initial release.
v2.0	Set 2020	Minor arrangements with some specific plugins, rearrange the chapters, and add the following content: - Mux/Demux Examples - Audio Examples - Image Examples - Transcode Examples - Streaming Examples - Multi Display Examples - Scaling and Rotation Examples - Zero-copy Examples - Debug Examples

Chapter 2. GStreamer-1.0 Plugin Reference

The NXP BSP L5.4.24_2.1.1 supports a huge list of GStreamer plugins. This section describes all the plugins used on this user guide, but it does not intent to keep the user limited by them, so feel free and encouraged to experience other options.

To check all the plugins available, enter with the following command at the device:

```
$ gst-inspect-1.0
```

2.1. Video Decoder Plugins

Video decoders are usually used to link a video source format to a raw format, which can be interpreted by the destination sink, such as a display.

Table 2. Video Decoder Plugins

Video Decoder	Package	Description
decodebin	gst-plugins-base	Autoplug and decode to raw media
v4l2mpeg4dec	gst-plugins-good	Decodes MPEG4 streams via V4L2 API
v4l2mpeg2dec	gst-plugins-good	Decodes MPEG2 streams via V4L2 API
v4l2h264dec	gst-plugins-good	Decodes H.264 streams via V4L2 API
v4l2h265dec	gst-plugins-good	Decodes H.265 streams via V4L2 API
v4l2vp6dec	gst-plugins-good	Decodes VP6 streams via V4L2 API
v4l2vp8dec	gst-plugins-good	Decodes VP8 streams via V4L2 API

2.2. Video Encoder Plugins

Working as an opposition from the decoders, video encoders can take raw data and turns into an encoded video format, such as H.264 format.

Table 3. Video Encoder Plugins

Video Encoder	Package	Description
encodebin	gst-plugins-base	Convenience encoding/muxing element
v4l2h264enc	gst-plugins-good	Encode H.264 video streams via V4L2 API

2.3. Video Sink Plugins

Video sink plugins are used to show the data consumed results through the display output.

Table 4. Video Sink Plugins

Video Sink	Package	Description
autovideosink	gst-plugins-good	Wrapper video sink for automatically detected video sink
kmssink	gst-plugins-bad	Video sink using the Linux Kernel mode setting API*
ximagesink	gst-plugins-base	A standard X based videosink**
glimagesink	gst-plugins-base	Infrastructure to process GL textures

Video Sink	Package	Description
waylandsink	gst-plugins-bad	Output to wayland surface
fbdevsink	gst-plugins-bad	Output a Linux framebuffer videosink
fpsdisplaysink	gst-plugins-bad	Video sink with current and average framerate

*In order to use the `kmssink` and `fbdevsink` plugins, stop the `weston` interface before: `$ systemctl stop weston`

**In order to use the `ximagesink` plugins, start the X server before: `$ export DISPLAY=:0`

2.4. Demux Plugins

Demuxers plugins are responsible to convert different video/audio formats into raw unparsed data. The most common are described in the table below.

Table 5. Demux Plugins

Video Demux	Package	Description
qtdemux	gst-plugins-good	Demux a .mov/.mp4 file to raw data
matroskademux	gst-plugins-good	Demux a .mkv file to raw data
flvdemux	gst-plugins-good	Demux a .flv file to raw data
avidemux	gst-plugins-good	Demux a .avi file to raw data
aiurdemux	imx-gst1.0-plugin	Unified parser for raw data

2.5. Mux Plugins

Muxers plugins are responsible to convert raw unparsed data into a specific video/audio data. The most common are described in the table below.

Table 6. Mux Plugins

Video Mux	Package	Description
qtmux	gst-plugins-good	Mux a raw data to a .mov file
matroskemux	gst-plugins-good	Mux a raw data to a .mkv file
flvmux	gst-plugins-good	Mux a raw data to a .flv file
avimux	gst-plugins-good	Mux a raw data to a .avi file
mp4mux	gst-plugins-good	Mux a raw data to a .mp4 file

2.6. i.MX Proprietary Plugins

The i.MX GStreamer support has the following proprietary plugins, which can help the user to reach some superior results by using it.

Table 7. i.MX Proprietary Plugins

i.MX Proprietary Plugin	Package	Description
vpudec	imx-gst1.0-plugin	Decodes compressed video to raw data
vpuenc_h264	imx-gst1.0-plugin	Encode raw data to compressed video

i.MX Proprietary Plugin	Package	Description
vpuenc_vp8	imx-gst1.0-plugin	Encode raw data to compressed video
imxcompositor_g2d	imx-gst1.0-plugin	Composite multiple video streams with HW acceleration
imxvideoconvert_g2d	imx-gst1.0-plugin	i.MX Video Convert Plugins with HW acceleration

2.7. Audio Plugins

Audio plugins are responsible to arrange the data from audio raw formats or specific audio data formats, such as WAV.

Table 8. Audio Plugins

Audio Plugin	Package	Description
mpg123audiodec	gst-plugins-good	MP3 decoding plugin based on the mpg123 library
vorbisdec	gst-plugins-base	Decodes raw vorbis streams to float audio
vorbisenc	gst-plugins-base	Encodes audio in Vorbis format
alsasink	gst-plugins-base	Output to a sound card via ALSA
pulsesink	gst-plugins-good	Plays audio to a PulseAudio server

2.8. Image Plugins

Image plugins are responsible to arrange the data from image raw formats or specific data formats, such as JPEG.

Table 9. Image Plugins

Image Plugins	Package	Description
jpegdec	gst-plugins-good	Decode images from JPEG format
v4l2jpegdec	gst-plugins-good	Decodes JPEG streams via V4L2 API
pngdec	gst-plugins-good	Decode a png video frame to a raw image
jpegenc	gst-plugins-good	Encode images in JPEG format
pngenc	gst-plugins-good	Decode a png video frame to a raw image
imagefreeze	gst-plugins-good	Generates a still frame stream from an image

2.9. Network Protocol Plugins

Network protocol plugins are responsible for establishing connections between devices over the network.

Table 10. Network Protocol Plugins

Network Plugins	Package	Description
udpsink	gst-plugins-good	Send data over the network via UDP
multiudpsink	gst-plugins-good	Send data over the network via UDP to one or multiple recipients
udpsrc	gst-plugins-good	Receive data over the network via UDP
tcpserversink	gst-plugins-base	Send data as a server over the network via TCP
tcpclientsrc	gst-plugins-base	Receive data as a client over the network via TCP

Network Plugins	Package	Description
rtspsrc	gst-plugins-good	Receive data over the network via RTSP

2.10. Payload/Depayload Plugins

Payload plugins are responsible for packing the data over the network. In order to received and unpacking it, depayload plugins are used in combination with these plugins.

Table 11. Payload/Depayload Plugins

Pay/Depayload	Package	Description
gdppay	gst-plugins-bad	Payloads GStreamer Data Protocol buffers
gdpdepay	gst-plugins-bad	Depayloads GStreamer Data Protocol buffers
rtpvrawpay	gst-plugins-good	Payload raw video as RTP packets
rtpvrawdepay	gst-plugins-good	Extracts raw video from RTP packets
rtph264pay	gst-plugins-good	Payload-encode H264 video into RTP packets
rtph264depay	gst-plugins-good	Extracts H264 video from RTP packets
rtpmpapay	gst-plugins-good	Payload MPEG audio as RTP packets
rtpmpadepay	gst-plugins-good	Extracts MPEG audio from RTP packets
rtpjitterbuffer	gst-plugins-good	A buffer that deals with network jitter and other transmission faults

Chapter 3. Decode Examples

This section shows how to perform video decode with some GStreamer pipelines examples and its supported devices.

3.1. i.MX 8M

The i.MX 8M family adopted the `Hantro VPU IP`. This VPU provides the following accelerated video decoder solutions.

3.1.1. H.264/H.265/VP8 Decode (HW Accelerated Decode)

```
$ gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=d d.video_0 ! queue ! h264parse ! vpudec ! \
  queue ! waylandsink
```

The `vpudec` provides support for more than one video format. In order to use it correctly, be sure to set the parser according:

- H.264: `h264parse`;
- H.265: `h265parse`;
- VP8: does not require parse plugin.

3.2. i.MX 8X/8QM

The i.MX 8QXP and i.MX 8QM SoCs are equipped with the `Amphion VPU IP`. This VPU provides the following accelerated video decoder solutions.

3.2.1. H.264 Decode (HW Accelerated Decode)

```
$ gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=d d.video_0 ! queue ! h264parse ! v4l2h264dec ! \
  imxvideoconvert_g2d ! queue ! waylandsink
```

The `Amphion VPU IP` uses a specific tiling format, so it requires the `imxvideoconvert_g2d` plugin usage.

3.2.2. H.265 Decode (HW Accelerated Decode)

```
$ gst-launch-1.0 filesrc location=<filename.mkv> ! \
  qtdemux name=d d.video_0 ! queue ! h265parse ! v4l2h265dec ! \
  imxvideoconvert_g2d ! queue ! waylandsink
```

3.2.3. MPEG-4 Decode (HW Accelerated Decode)

```
$ gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=d d.video_0 ! queue ! mpeg4videoparse ! v4l2mpeg4dec ! \
  imxvideoconvert_g2d ! queue ! waylandsink
```

3.2.4. MPEG-2 Decode (HW Accelerated Decode)

```
$ gst-launch-1.0 filesrc location=<filename.ts> ! \
  qtdemux name=d d.video_0 ! queue ! mpegvideoparse ! v4l2mpeg2dec ! \
  imxvideoconvert_g2d ! queue ! waylandsink
```

3.2.5. VP6 Decode (HW Accelerated Decode)

```
$ gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=d d.video_0 ! queue ! v4l2vp6dec ! \
  imxvideoconvert_g2d ! queue ! waylandsink
```

3.2.6. VP8 Decode (HW Accelerated Decode)

```
$ gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=d d.video_0 ! queue ! v4l2vp8dec ! \
  imxvideoconvert_g2d ! queue ! waylandsink
```

Chapter 4. Encode Examples

This section shows how to perform video encode with some GStreamer pipelines examples and its supported devices.

4.1. i.MX 8M

The i.MX 8M family adopted the `Hantro VPU IP`. This VPU provides the following accelerated video encode solutions.

4.1.1. H.264 Encode (HW Accelerated Encode)

```
$ gst-launch-1.0 videotestsrc ! \
  video/x-raw, format=I420, width=640, height=480 ! vpuenc_h264 ! \
  filesink location=test.mp4
```

The i.MX 8M Quad EVK and i.MX 8M Nano EVK do not have HW Accelerated encode support.

4.1.2. VP8 Encode (HW Accelerated Encode)

```
$ gst-launch-1.0 videotestsrc ! \
  video/x-raw, format=I420, width=640, height=480 ! vpuenc_vp8 ! \
  matroskamux ! filesink location=test.mkv
```

4.2. i.MX 8X/8QM

The i.MX 8QXP and i.MX 8QM SoCs are equipped with the `Amphion VPU IP`. This VPU provides the following accelerated video encoder solutions.

4.2.1. H.264 Encode (HW Accelerated Decode)

```
$ gst-launch-1.0 videotestsrc ! \
  video/x-raw, format=NV12, width=640, height=480 ! v4l2h264enc ! \
  filesink location=test.mp4
```

4.2.2. Additional control of v4l2

```
$ v4l2-ctl --list-ctrls -d /dev/video13
```

This should output something like:

```
User Controls
min_number_of_output_buffers (int) : min=1 max=32 step=1 default=3 value=3 flags=read-only, volatile
Codec Controls
video_b_frames (int) : min=0 max=4 step=1 default=2 value=2 flags=update
video_gop_size (int) : min=1 max=300 step=1 default=30 value=30 flags=volatile, execute-on-write
video_bitrate_mode (menu) : min=0 max=1 default=0 value=0 flags=update
video_bitrate (int) : min=16384 max=251658240 step=1024 default=2097152 value=2097152
video_peak_bitrate (int) : min=16384 max=251658240 step=1024 default=8388608 value=8388608
force_key_frame (button) : flags=write-only, execute-on-write
h264_i_frame_qp_value (int) : min=0 max=51 step=1 default=25 value=25
h264_p_frame_qp_value (int) : min=0 max=51 step=1 default=25 value=25
h264_b_frame_qp_value (int) : min=0 max=51 step=1 default=25 value=25
h264_i_frame_period (int) : min=1 max=300 step=1 default=30 value=30 flags=volatile, execute-on-write
h264_level (menu) : min=0 max=15 default=11 value=11
h264_profile (menu) : min=0 max=4 default=0 value=0
h264_arbitrary_slice_ordering (int) : min=0 max=1 step=1 default=1 value=1
```

Then with the provided information, it is possible to configure more encode parameters such as shown below:

```
$ gst-launch-1.0 v4l2src device=/dev/video1 ! video/x-raw,width=640,height=480 ! \
  v4l2h264enc extra-controls="controls,h264_entropy_mode=0,video_bitrate=245000;" ! \
  h264parse ! v4l2h264dec ! queue ! waylandsink sync=false
```

And to reduce CPU usage, use the following:

```
$ gst-launch-1.0 v4l2src device=/dev/video0 num-buffers=300 io-mode=dmabuf ! \
  'video/x-raw,format=(string)NV12,width=1920,height=1080,framerate=(fraction)30/1' ! \
  queue ! v4l2h264enc output-io-mode=dmabuf-import ! avimux ! filesink location=test.avi
```

Chapter 5. Mux/demux Examples

This section shows some most commonly used mux and demux plugins and some examples of how to use each one of them correctly.

5.1. Mux Plugins

5.1.1. qtmux

This type of muxer converts video and/or audio into `QuickTime (.mov)` files.

```
$ gst-launch-1.0 v4l2src num-buffers=500 ! video/x-raw,width=320,height=240 ! \
  videoconvert ! qtmux ! filesink location=video.mov
```

The pipeline above records a camera video to a `.mov` file.

5.1.2. matroskamux

This type of muxer converts video and/or audio into a `Matroska (.MKV)` file.

```
$ gst-launch-1.0 filesrc location=<AUDIO_FILE>.mp3 ! \
  mpegaudioparse ! matroskamux ! filesink location=test.mkv
```

The pipeline above muxes an `MP3` file into an `MKV` file.

5.1.3. mp4mux

This type of muxer converts video and/or audio into an `ISO MPEG-4 (.mp4)` file.

```
$ gst-launch-1.0 v4l2src num-buffers=50 ! queue ! vpudec ! mp4mux ! filesink location=video.mp4
```

The pipeline above describes the process of recording a video from a v4l2 device, encoding into an H.264 format and muxes into a `.mp4` file.

5.1.4. flvmux

This type of muxer converts video and/or audio into a `flash video (.FLV)` file.

```
$ gst-launch-1.0 filesrc location=<AUDIO_FILE>.mp3 ! decodebin ! queue ! \
  flvmux name=mux ! filesink location=test.flv \
  filesrc location=<VIDEO_FILE>.mp4 ! decodebin ! queue ! mux.
```

The pipeline above decodes a video and audio file into a `.FLV` file.

5.1.5. avimux

This type of muxer converts video and/or audio into an `.AVI` file.

```
$ gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw,format=I420,width=640,height=480,framerate=30/1' \
  avimux ! filesink location=test.avi
```

5.2. Demux Plugins

5.2.1. aiurdemux

This type of demuxer is a proprietary plugin from NXP that automatically detects the file encoding and turns into a `raw` unparsed file.

```
$ gst-launch-1.0 filesrc location=<VIDEO_FILE>.mp4 ! video/quicktime ! \
  aiurdemux ! queue max-size-time=0 ! vpudec ! autovideosink
```

5.2.2. qtdemux

This type of demuxer converts a `.mov` file into a `raw` unparsed file.

```
$ gst-launch-1.0 filesrc location=<VIDEO_FILE>.mov ! qtdemux name=demux  \
  demux.audio_0 ! queue ! decodebin ! audioconvert ! audioresample ! autoaudiosink \
  demux.video_0 ! queue ! decodebin ! videoconvert ! videoscale ! autovideosink
```

5.2.3. matroskademux

This type of demuxer converts a `.mkv` file into a `raw` unparsed file.

```
$ gst-launch-1.0 filesrc location=<VIDEO_FILE>.mkv ! matroskademux ! \
  vorbisdec ! audioconvert ! audioresample ! autoaudiosink
```

5.2.4. flvdemux

This type of demuxer converts a `.flv` file into a `raw` unparsed file.

```
$ gst-launch-1.0 filesrc location=<VIDEO_FILE>.flv ! flvdemux ! \
  audioconvert ! autoaudiosink
```

5.2.5. avidemux

This type of demuxer converts a `.avi` file into a `raw` unparsed file.

```
$ gst-launch-1.0 filesrc location=<VIDEO_FILE>.avi ! avidemux name=demux \
  demux.audio_0 ! decodebin ! audioconvert ! audioresample ! autoaudiosink \
  demux.video_0 ! decodebin ! videoconvert ! videoscale ! autovideosink
```

Chapter 6. Camera Examples

This section shows how to perform camera captures with some GStreamer pipelines examples and its supported devices. There are also some differences regarding each board model, which will be described at this chapter.

6.1. Camera Capture

First, locate the video outputs with the following command line:

```
$ gst-device-monitor-1.0
```

This should output something like:

```
Device found:
        name  : i.MX6S_CSI
        class : Video/Source
        caps  : video/x-raw, format=(string)YUY2, width=(int)2592, height=(int);
                video/x-raw, format=(string)YUY2, width=(int)1920, height=(int);
                video/x-raw, format=(string)YUY2, width=(int)1280, height=(int);
                video/x-raw, format=(string)YUY2, width=(int)720, height=(int)4;
                video/x-raw, format=(string)YUY2, width=(int)640, height=(int)4;
                video/x-raw, format=(string)YUY2, width=(int)320, height=(int)2;
        properties:
                udev-probed = true
                device.bus_path = platform-32e20000.csi1_bridge
                sysfs.path = /sys/devices/platform/32e20000.csi1_bridge/video4l0
                device.subsystem = video4linux
                device.product.name = i.MX6S_CSI
                device.capabilities = :capture:
                device.api = v4l2
                device.path = /dev/video0
                v4l2.device.driver = mx6s-csi
                v4l2.device.card = i.MX6S_CSI
                v4l2.device.bus_info = platform:32e20000.csi1_bridge
                v4l2.device.version = 267043 (0x00041323)
                v4l2.device.capabilities = 2216689665 (0x84200001)
                v4l2.device.device_caps = 69206017 (0x04200001)
        gst-launch-1.0 v4l2src ! ...
```

The described command line can output various important information such as camera resolution, framerate, and supported formats.

You can use the following command line to locate the video outputs as well:

```
$ v4l2-ctl --list-devices
```

This should output something like:

```
vpu B0 (platform:):
       /dev/video12
       /dev/video13
mxc-isi (platform:58100000.isi.0):
       /dev/video0
       /dev/video5
mxc-isi (platform:58110000.isi.1):
       /dev/video1
mxc-isi (platform:58120000.isi.2):
       /dev/video2
mxc-isi (platform:58130000.isi.3):
       /dev/video3
mxc-jpeg decoder (platform:58400000.jpegdec):
       /dev/video4
mxc-jpeg decoder (platform:58450000.jpegenc):
       /dev/video6
```

Then, to automatically output the CSI port into the screen, type the following pipeline:

```
$ gst-launch-1.0 v4l2src device=/dev/videoX ! autovideosink
```

6.2. Changing the Camera Resolution and Framerate

In order to change configurations such as resolution and framerate, enter with the following properties:

```
$ gst-launch-1.0 v4l2src ! video/x-raw, width=<WIDTH>, height=<HEIGHT>, framerate=<FRAMERATE> ! autovideosink
```

Which `WIDTH`, `FRAMERATE`, and `HEIGHT` are the parameters that you should change.

As an example, in order to set the resolution to HD, run the following pipeline:

```
$ gst-launch-1.0 v4l2src ! 'video/x-raw,framerate=30/1,width=1280,height=720' ! autovideosink
```

And to change the frame rate to 60 fps, run the following pipeline:

```
$ gst-launch-1.0 v4l2src ! 'video/x-raw,framerate=60/2,width=1280,height=720' ! autovideosink
```

6.3. Using Multiple Cameras

The i.MX 8QXP MEK C0 and i.MX 8QM MEK supports more than one camera.

In order to display all the cameras at the same monitor output, use the following GStreamer pipeline:

```
$ gst-launch-1.0 imxcompositor_g2d name=comp \
  sink_0::xpos=0 sink_0::ypos=0 sink_0::width=640 sink_0::height=480 \
  sink_1::xpos=0 sink_1::ypos=480 sink_1::width=640 sink_1::height=480 \
  sink_2::xpos=640 sink_2::ypos=0 sink_2::width=640 sink_2::height=480 \
  sink_3::xpos=640 sink_3::ypos=480 sink_3::width=640 sink_3::height=480 ! \
  video/x-raw,format=RGB16 ! waylandsink \
  v4l2src device=/dev/video0 ! video/x-raw,width=640,height=480 ! comp.sink_0 \
  v4l2src device=/dev/video1 ! video/x-raw,width=640,height=480 ! comp.sink_1 \
  v4l2src device=/dev/video2 ! video/x-raw,width=640,height=480 ! comp.sink_2 \
  v4l2src device=/dev/video3 ! video/x-raw,width=640,height=480 ! comp.sink_3
```

This pipeline enables the user to set up more than one camera to the same screen using the `imxcompositor_g2d` plugin. This is the unique solution available to create an interface over `Weston/Wayland` interface, i.e., in i.MX 8 devices we need to use GPU to handle the screen position.

Chapter 7. Audio Examples

This section describes some basic pipelines regarding audio output using GStreamer.

7.1. Audio Playback Example

Audio playback consists of the process of playing a determining audio file based on its determined file format.

In the examples shown below, the use of `audiotestsrc` plugin outputs standard audio to the audio jack:

```
$ gst-launch-1.0 audiotestsrc wave=5 ! alsasink device=plughw:1
```

if needed to change the device and confirm the number of each output, run the command *$ pactl list sinks*

7.2. Audio Decode Examples

The following described pipelines decodes a audio file located in the board using the `filesrc` plugin:

7.2.1. Play an MP3 format file:

```
$ gst-launch-1.0 filesrc location=<audio_file.mp3> ! mpegaudioparse ! mpg123audiodec ! audioconvert ! alsasink device=plughw:1
```

7.2.2. Play an Ogg Vorbis format file:

```
$ gst-launch-1.0 filesrc location=<audio_file.ogg> ! oggdemux ! vorbisdec ! audioconvert ! audioresample ! alsasink device=plughw:1
```

7.3. Audio Format Conversion

Audio conversion is the process to change the current format of the audio file to another desired format, for example changing `.wav` to `.aac`.

In the pipelines described below, some cases are used as an example.

7.3.1. Convert MP3 to the Ogg Vorbis format:

```
$ gst-launch-1.0 filesrc location=Sweet.mp3 ! audioconvert ! vorbisenc ! oggmux ! filesink location=output.ogg
```

7.3.2. Convert WAV to the MP3 format:

```
$ gst-launch-1.0 filesrc location=<audio_file.wav> ! wavparse ! avenc_mp2 ! filesink location=output.mp3
```

7.4. Audio Record

Besides audio playback and conversion of the file format, it's also possible to record audio provided by some external source such as a microphone attached to the jack input.

The pipeline described below shows the process to obtain this type of audio file and save it as an Ogg Vorbis file:

```
$ gst-launch-1.0 -v pulsesrc ! audioconvert ! vorbisenc ! oggmux ! filesink location=alsasrc.ogg
```

Chapter 8. Image Examples

This section describes some basic pipelines regarding image output using GStreamer.

8.1. Image Output

Image output consists on the process of showing on the desired screen or any other type of output source, the desired image file. The pipeline described below executes this process:

8.1.1. To display a PNG image file, use the following pipeline:

```
$ gst-launch-1.0 filesrc location=<output_image>.png ! pngdec ! imagefreeze ! videoconvert ! autovideosink
```

8.1.2. To display a JPEG image file, use the following pipeline:

```
$ gst-launch-1.0 -v filesrc location=<output_image>.jpeg ! jpegdec ! imagefreeze ! videoconvert ! autovideosink
```

8.2. Image Record

For image record, it is possible to use an image provider input, such as a camera, in order to execute the pipeline and obtain pictures from the camera. The pipelines described below executes this process:

8.2.1. Camera Raw to JPG

```
$ gst-launch-1.0 v4l2src num-buffers=1 ! jpegenc ! filesink location=capture.jpg
```

8.2.2. Camera Raw to PNG

```
$ gst-launch-1.0 v4l2src num-buffers=1 ! pngenc ! filesink location=capture.png
```

8.2.3. Camera Raw to JPEG

```
$ gst-launch-1.0 v4l2src num-buffers=1 ! jpegenc ! filesink location=capture.jpeg
```

8.3. JPEG VPU support

The i.MX 8QM and i.MX 8QXP support JPEG VPU encode thought `v4l2jpegenc` plugin. Check the pipeline below for an example:

```
$ gst-launch-1.0 v4l2src num-buffers=1 ! v4l2jpegenc ! filesink location=capture.jpeg
```

Chapter 9. Transcode Examples

This section shows how to perform some transcoding pipelines desired to general i.MX8 boards and how to properly run those pipelines in each one of them.

9.1. Video Transcoding

The example below transcodes MJPEG file obtained from the camera into a MKV file:

```
$ gst-launch-1.0  v4l2src device=/dev/video0 ! jpegparse ! v4l2jpegdec ! queue ! videoconvert ! v4l2h264enc ! h264parse ! matroskamux !
filesink location=out.mkv
```

In some cases, the pipeline uses a lot of the processing power from the board, in this case, it is desirable to use the `zero-copy` method with the pipeline, such as shown below:

```
$ gst-launch-1.0 v4l2src num-buffers=300 io-mode=dmabuf ! v4l2h264enc output-io-mode=dmabuf-import ! h264parse ! v4l2h264dec !
imxvideoconvert_g2d ! waylandsink
```

For more information on this type of process, check Zero-copy Pipelines chapter.

Chapter 10. Video Streaming

This section shows how to perform video, camera, and audio streaming with some GStreamer pipelines examples and its supported devices.

10.1. Video file Streaming

10.1.1. Video UDP Streaming

- SERVER

```
$ gst-launch-1.0 -v filesrc location=<filename.mp4> ! \
  qtdemux ! queue ! rtph264pay ! \
  udpsink host=<CLIENT_IP>
```

In order to perform UDP streaming, the `SERVER` pipeline must be the verbose enabled (-v). The output value should be something like the following:

```
Setting pipeline to PAUSED ...
Pipeline is PREROLLING ...
/GstPipeline:pipeline0/GstQueue:queue0.GstPad:sink: caps = video/x-h264, stream-format=(string)avc, alignment=(string)au, level=(string)4,
profile=(string)high,
codec_data=(buffer)01640028ffe1001e67640028acd940780227e5c05a808080a0000003002000000641e30632c001000568e93b2c8bfdf8f800, width=(int)1920,
height=(int)1080, framerate=(fraction)25/1, pixel-aspect-ratio=(fraction)1/1, colorimetry=(string)bt709
/GstPipeline:pipeline0/GstQueue:queue0.GstPad:sink: caps = video/x-h264, stream-format=(string)avc, alignment=(string)au, level=(string)4,
profile=(string)high,
codec_data=(buffer)01640028ffe1001e67640028acd940780227e5c05a808080a0000003002000000641e30632c001000568e93b2c8bfdf8f800, width=(int)1920,
height=(int)1080, framerate=(fraction)25/1, pixel-aspect-ratio=(fraction)1/1, colorimetry=(string)bt709
/GstPipeline:pipeline0/GstRtpH264Pay:rtph264pay0.GstPad:src: caps = application/x-rtp, media=(string)video, clock-rate=(int)90000,
encoding-name=(string)H264, packetization-mode=(string)1, profile-level-id=(string)640028, sprop-parameter-
sets=(string)"Z2QAKKzZQHgCJ+XAWoCAgKAAAAMAIAAABkHjBjLA\,aOk7LIs\=", payload=(int)96, ssrc=(uint)1622932748, timestamp-
offset=(uint)2591288474, seqnum-offset=(uint)8760, a-framerate=(string)25
/GstPipeline:pipeline0/GstUDPSink:udpsink0.GstPad:sink: caps = application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-
name=(string)H264, packetization-mode=(string)1, profile-level-id=(string)640028, sprop-parameter-
sets=(string)"Z2QAKKzZQHgCJ+XAWoCAgKAAAAMAIAAABkHjBjLA\,aOk7LIs\=", payload=(int)96, ssrc=(uint)1622932748, timestamp-
offset=(uint)2591288474, seqnum-offset=(uint)8760, a-framerate=(string)25
/GstPipeline:pipeline0/GstRtpH264Pay:rtph264pay0.GstPad:sink: caps = video/x-h264, stream-format=(string)avc, alignment=(string)au,
level=(string)4, profile=(string)high,
codec_data=(buffer)01640028ffe1001e67640028acd940780227e5c05a808080a0000003002000000641e30632c001000568e93b2c8bfdf8f800, width=(int)1920,
height=(int)1080, framerate=(fraction)25/1, pixel-aspect-ratio=(fraction)1/1, colorimetry=(string)bt709
/GstPipeline:pipeline0/GstRtpH264Pay:rtph264pay0: timestamp = 2591288474
/GstPipeline:pipeline0/GstRtpH264Pay:rtph264pay0: seqnum = 8760
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
```

By examining the output values, search for `caps = "application/x-rtp"` and copy its value.

- CLIENT

The caps value has to be used at the `CLIENT` pipeline, so export it as `CAPS` value:

```
export CAPS='application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264, packetization-mode=(string)1,
profile-level-id=(string)640028, sprop-parameter-sets=(string)"Z2QAKKzZQHgCJ+XAWoCAgKAAAAMAIAAABkHjBjLA\,aOk7LIs\=", payload=(int)96,
ssrc=(uint)1622932748, timestamp-offset=(uint)2591288474, seqnum-offset=(uint)8760, a-framerate=(string)25'
```

Then run the `udpsrc` pipeline:

```
$ gst-launch-1.0 udpsrc caps = $CAPS ! \
  rtpjitterbuffer latency=100 ! queue max-size-buffers=0 ! \
  rtph264depay ! decodebin ! autovideosink sync=false
```

10.1.2. Video Multi UDP Streaming

- SERVER

```
$ gst-launch-1.0 -v filesrc location=<filename.mp4> ! \
  qtdemux ! queue ! rtph264pay ! \
  multiudpsink clients=<CLIENT_IP_1>:5000,<CLIENT_IP_2>:5004
```

- CLIENT_1

```
$ gst-launch-1.0 udpsrc port=5000 caps = $CAPS ! \
  rtpjitterbuffer latency=100 ! queue max-size-buffers=0 ! \
  rtph264depay ! decodebin ! autovideosink sync=false
```

- CLIENT_2

```
$ gst-launch-1.0 udpsrc port=5004 caps = $CAPS ! \
  rtpjitterbuffer latency=100 ! queue max-size-buffers=0 ! \
  rtph264depay ! decodebin ! autovideosink sync=false
```

For more details at the `CAPS` value, check the Video UDP Streaming section.

10.1.3. Video TCP Streaming

- SERVER

```
$ gst-launch-1.0 filesrc location=<video_file>.mov ! \
  decodebin ! vpuenc_h264 gop-size=2 bitrate=10000 ! \
  rtph264pay pt=96 timestamp-offset=0 ! gdppay ! \
  tcpserversink blocksize=512000 host=<SERVER_IP> port=8554 sync=false
```

- CLIENT

```
$ gst-launch-1.0 tcpclientsrc host=<SERVER_IP> port=8554 ! gdpdepay ! queue max-size-buffers=0 ! \
  'application/x-rtp, framerate=(fraction)24/1, media=(string)video, clock10-rate=(int)90000, payload=(int)96' ! \
  rtpjitterbuffer latency=100 ! rtph264depay ! \
  decodebin ! autovideosink sync=false
```

10.1.4. Video RTSP Streaming

In order to enable RTSP protocol support on i.MX 8 devices, please check the i.MX 8 - RTSP Streaming Support documentation.

- SERVER

```
$ gst-variable-rtsp-server -p 9001 -u \
  "filesrc location=<filename.mp4> ! \
  qtdemux ! queue ! rtph264pay name=pay0 pt=96"
```

- CLIENT

```
$ gst-launch-1.0 rtspsrc location=rtsp://$SERVERIP:9001/stream ! \
  queue ! rtph264depay ! v4l2h264dec ! autovideosink
```

10.1.5. Video Streaming to PC/VLC

- SERVER

```
$ gst-launch-1.0 -v filesrc location=<video_file>.mov ! \
  qtdemux ! rtph264pay ! \
  udpsink host=<CLIENT_IP> port=5004
```

In the Linux PC, create a text file named `test_video.sdp` and copy the following content to it:

- CLIENT

```
v=0
m=video 5000 RTP/AVP 96
c=IN IP4 <SERVER_IP>
a=rtpmap:96 H264/90000
a=fmtp:96 sprop-parameter-sets=J01AHqkYGwe83gDUBAQG2wrXvfAQ=,KN4JyA=;
```

For more details at the `sprop-parameter-sets` content, check the `CAPS` value at the Video UDP Streaming section.

Then start the server and open the file with `VLC` application.

10.2. Camera Streaming

10.2.1. Camera UDP Streaming

- SERVER

```
$ gst-launch-1.0 -v v4l2src device=/dev/video0 ! 'video/x-raw,width=1280,height=720' ! \
  rtpvrawpay pt=96 timestamp-offset=0 ! queue max-size-buffers=0 ! \
  udpsink host=<CLIENT_IP>
```

- CLIENT

```
$ gst-launch-1.0 udpsrc caps = $CAPS ! \
  rtpjitterbuffer latency=1 ! queue max-size-buffers=0 ! \
  rtpvrawdepay ! autovideosink sync=false
```

For more details at the `CAPS` value, check the Video UDP Streaming section.

10.2.2. Camera TCP Streaming

- SERVER

```
$ gst-launch-1.0 v4l2src device=/dev/video0 ! 'video/x-raw,width=1280,height=720' ! \
  rtpvrawpay pt=96 timestamp-offset=0 ! queue max-size-buffers=0 ! gdppay ! \
  tcpserversink blocksize=512000 host=<SERVER_IP> port=8554 sync=false
```

- CLIENT

```
$ gst-launch-1.0 tcpclientsrc host=<SERVER_IP> port=8554 ! gdpdepay ! queue max-size-buffers=0 ! \
  'application/x-rtp, framerate=(fraction)24/1, media=(string)video, clock10-rate=(int)90000, payload=(int)96' ! \
  rtpjitterbuffer latency=100 ! rtpvrawdepay ! \
  decodebin ! autovideosink sync=false
```

10.2.3. Camera RTSP Streaming

In order to enable RTSP protocol support on i.MX 8 devices, please check the i.MX 8 - RTSP Streaming Support documentation.

- SERVER

```
$ gst-variable-rtsp-server -p 9001 -u \
  "v4l2src device=/dev/video0 ! video/x-raw,width=640,height=480 ! \
  v4l2h264enc ! rtph264pay name=pay0 pt=96"
```

- CLIENT

```
$ gst-launch-1.0 rtspsrc location=rtsp://$SERVERIP:9001/stream ! \
  queue ! rtph264depay ! v4l2h264dec ! autovideosink
```

10.3. Audio Streaming

10.3.1. Audio UDP Streaming

- SERVER

```
$ gst-launch-1.0 -v filesrc location=<audio_file>.mp3 ! \
  id3demux ! mpegaudioparse ! rtpmpapay ! \
  udpsink host=<CLIENT_IP> port=5004
```

- CLIENT

```
$ gst-launch-1.0 udpsrc port=5004 caps= $CAPS ! \
  rtpmpadepay ! queue ! \
  beepdec ! alsasink sync=false
```

For more details at the `CAPS` value, check the Video UDP Streaming section.

10.3.2. Audio Streaming to PC/VLC

- SERVER

```
$ gst-launch-1.0 filesrc location=<audio_file>.mp3 ! \
  id3demux ! mpegaudioparse ! rtpmpapay ! \
  udpsink host=<CLIENT_IP> port=5004
```

In the Linux PC, create a text file named `test_audio.sdp` and copy the following content to it:

- CLIENT

```
v=0
m=audio 5004 RTP/AVP 98
c=IN IP4 <SERVER_IP>
a=rtpmap:98 MP4A-LATM/48000
```

Then start the server and open the file with `VLC` application.

10.4. Video and Audio Streaming

10.4.1. Video and Audio Streaming to PC/VLC

- SERVER

```
$ gst-launch-1.0 -v filesrc location=<video_file>.mov ! qtdemux name=demux \
  demux. ! queue ! rtph264pay name=pay0 pt=96 ! udpsink host=<CLIENT_IP> port=5000 sync=false \
  demux. ! queue ! mpegaudioparse ! rtpmpapay ! udpsink host=<CLIENT_IP> port=5004 sync=false
```

In the Linux PC, create a text file named `test_audio.sdp` and copy the following content to it:

- CLIENT

```
v=0
m=video 5000 RTP/AVP 96
c=IN IP4 <SERVER_IP>
a=rtpmap:96 H264/90000
a=fmtp:96 sprop-parameter-sets=J01AHqkYGwe83gDUBAQG2wrXvfAQ=,KN4JyA=;
m=audio 5004 RTP/AVP 98
a=rtpmap:98 MP4A-LATM/48000
```

For more details at the `sprop-parameter-sets` content, check the `CAPS` value at the Video UDP Streaming section.

Then start the server and open the file with `VLC` application.

Chapter 11. Multi-Display Examples

Some i.MX 8 devices support more than one display output. This section describes how to enable them.

11.1. i.MX 8M Quad EVK

In order to enable dual display support on i.MX 8M Quad EVK, change the `.dtb` to `fsl-imx8mq-evk-dual-display.dtb`. The native HDMI will be handled by the DCSS controller and reaches up to 4k@60fps, while the MIPI-DSI will be controlled by the LCDIF and reaches up to 720@60fps.

11.1.1. Waylandsink + Kmssink

```
$ gst-launch-1.0 videotestsrc ! "video/x-raw,width=1920,height=1080" ! waylandsink \
  videotestsrc ! "video/x-raw,width=1280,height=720" ! kmssink
```

11.1.2. Kmsink Framebuffer + DRM

```
$ gst-launch-1.0 videotestsrc ! "video/x-raw,width=1280,height=720" ! \
  kmssink driver-name=mxsfb-drm \
  videotestsrc ! "video/x-raw,width=1920,height=1080" ! \
  kmssink driver-name=imx-drm
```

In order to use this pipeline, stop the `weston` interface before: `$ systemctl stop weston`

11.2. i.MX 8QM and i.MX 8QXP

The i.MX 8QM MEK can handle up to four monitors. However, just like the i.MX 8QXP, it requires the mouse navegation on these displays in order to enable it.

So in order to support more than one display at these devices, move the mouse to it, click at the screen, and then run any GStreamer pipeline.

Repeat this process to each monitor.

Chapter 12. Video Composition

The composition consists of a method of outputting multiple video displays with GStreamer. It is usefulness reaches a lot of applications and it is very common video output method for many necessities.

For i.MX 8 devices, the unique available solution to create a video composition over `Weston/Wayland` is by using GPU to handle the screen position. So all the GStreamer pipelines in this section use the `imxcompositor_g2d` plugin for it.

12.1. i.MX 8M

12.1.1. Video Composition Example

```
# gst-launch-1.0 -v imxcompositor_g2d name=comp \
  sink_0::xpos=0 sink_0::ypos=0 sink_0::width=640 sink_0::height=480 \
  sink_1::xpos=0 sink_1::ypos=480 sink_1::width=640 sink_1::height=480 ! \
  waylandsink \
  videotestsrc ! comp.sink_0 \
  videotestsrc ! comp.sink_1
```

This pipeline does not work with the i.MX 8M Nano EVK because it does not have the `imxcompositor_g2d` plugin support.

12.1.2. Video Decode Composition Example

This example shows how to display nine videos from a unique H.264 decode process:

```
# gst-launch-1.0 -v imxcompositor_g2d name=comp \
  sink_0::xpos=0 sink_0::ypos=0 sink_0::width=1280 sink_0::height=720 \
  sink_1::xpos=0 sink_1::ypos=720 sink_1::width=1280 sink_1::height=720 \
  sink_2::xpos=0 sink_2::ypos=1440 sink_2::width=1280 sink_2::height=720 \
  sink_3::xpos=1280 sink_3::ypos=0 sink_3::width=1280 sink_3::height=720 \
  sink_4::xpos=1280 sink_4::ypos=720 sink_4::width=1280 sink_4::height=720 \
  sink_5::xpos=1280 sink_5::ypos=1440 sink_5::width=1280 sink_5::height=720 \
  sink_6::xpos=2560 sink_6::ypos=0 sink_6::width=1280 sink_6::height=720 \
  sink_7::xpos=2560 sink_7::ypos=720 sink_7::width=1280 sink_7::height=720 \
  sink_8::xpos=2560 sink_8::ypos=1440 sink_8::width=1280 sink_5::height=720 \
  waylandsink sync=false filesrc location=<HD_video_file> ! decodebin ! imxvideoconvert_g2d ! tee name=t \
  t. ! queue ! comp.sink_0 \
  t. ! queue ! comp.sink_1 \
  t. ! queue ! comp.sink_2 \
  t. ! queue ! comp.sink_3 \
  t. ! queue ! comp.sink_4 \
  t. ! queue ! comp.sink_5 \
  t. ! queue ! comp.sink_6 \
  t. ! queue ! comp.sink_7 \
  t. ! queue ! comp.sink_8
```

This pipeline does not work with the i.MX 8M Nano EVK because it does not have the `imxcompositor_g2d` plugin support.

12.1.3. Two Camera Composition Example

At this example, it is using a `MINISAS-TO-CSI` daughter card and a USB web camera:

```
# gst-launch-1.0 -v imxcompositor_g2d name=comp \
  sink_0::xpos=0 sink_0::ypos=0 sink_0::width=640 sink_0::height=480 \
  sink_1::xpos=0 sink_1::ypos=480 sink_1::width=640 sink_1::height=480 ! \
  video/x-raw,format=RGB16 ! autovideosink \
  v4l2src device=/dev/video0 ! video/x-raw,width=640,height=480 ! comp.sink_0 \
  v4l2src device=/dev/video1 ! video/x-raw,width=640,height=480 ! comp.sink_1
```

This pipeline does not work with the i.MX 8M Nano EVK because it does not have the `imxcompositor_g2d` plugin support.

12.2. i.MX 8QXP/QM

12.2.1. Video Composition Example

```
# gst-launch-1.0 -v imxcompositor_g2d name=comp \
  sink_0::xpos=0 sink_0::ypos=0 sink_0::width=640 sink_0::height=480
  sink_1::xpos=0 sink_1::ypos=480 sink_1::width=640 sink_1::height=480 ! \
  waylandsink \
  videotestsrc ! comp.sink_0 \
  videotestsrc ! comp.sink_1
```

12.2.2. Nine Video Decode Composition Example

Different from i.MX 8M example, this example decodes nine different H.264 videos at the same time:

```
# gst-launch-1.0 -v imxcompositor_g2d name=comp \
  sink_0::xpos=0 sink_0::ypos=0 sink_0::width=1280 sink_0::height=720 \
  sink_1::xpos=0 sink_1::ypos=720 sink_1::width=1280 sink_1::height=720 \
  sink_2::xpos=0 sink_2::ypos=1440 sink_2::width=1280 sink_2::height=720 \
  sink_3::xpos=1280 sink_3::ypos=0 sink_3::width=1280 sink_3::height=720 \
  sink_4::xpos=1280 sink_4::ypos=720 sink_4::width=1280 sink_4::height=720 \
  sink_5::xpos=1280 sink_5::ypos=1440 sink_5::width=1280 sink_5::height=720 \
  sink_6::xpos=2560 sink_6::ypos=0 sink_6::width=1280 sink_6::height=720 \
  sink_7::xpos=2560 sink_7::ypos=720 sink_7::width=1280 sink_7::height=720 \
  sink_8::xpos=2560 sink_8::ypos=1440 sink_8::width=1280 sink_5::height=720 \
  ! waylandsink sync=false \
  filesrc location=<video_file> ! decodebin ! imxvideoconvert_g2d ! comp.sink_0 \
  filesrc location=<video_file> ! decodebin ! imxvideoconvert_g2d ! comp.sink_1 \
  filesrc location=<video_file> ! decodebin ! imxvideoconvert_g2d ! comp.sink_2 \
  filesrc location=<video_file> ! decodebin ! imxvideoconvert_g2d ! comp.sink_3 \
  filesrc location=<video_file> ! decodebin ! imxvideoconvert_g2d ! comp.sink_4 \
  filesrc location=<video_file> ! decodebin ! imxvideoconvert_g2d ! comp.sink_5 \
  filesrc location=<video_file> ! decodebin ! imxvideoconvert_g2d ! comp.sink_6 \
  filesrc location=<video_file> ! decodebin ! imxvideoconvert_g2d ! comp.sink_7 \
  filesrc location=<video_file> ! decodebin ! imxvideoconvert_g2d ! comp.sink_8
```

12.2.3. Eight Camera Composition Example

This following example uses two `MX8XMIPI4CAM2` daughter cards and 8 `MCIMXCAMERA1MP` cameras, being able to display 8 different images at the same output.

```
# gst-launch-1.0 -v imxcompositor_g2d name=comp \
  sink_0::xpos=0 sink_0::ypos=0 sink_0::width=320 sink_0::height=240 \
  sink_1::xpos=0 sink_1::ypos=240 sink_1::width=320 sink_1::height=240 \
  sink_2::xpos=320 sink_2::ypos=0 sink_2::width=320 sink_2::height=240 \
  sink_3::xpos=320 sink_3::ypos=240 sink_3::width=320 sink_3::height=240 \
  sink_4::xpos=640 sink_4::ypos=0 sink_4::width=320 sink_4::height=240 \
  sink_5::xpos=640 sink_5::ypos=240 sink_5::width=320 sink_5::height=240 \
  sink_6::xpos=960 sink_6::ypos=0 sink_6::width=320 sink_6::height=240 \
  sink_7::xpos=960 sink_7::ypos=240 sink_7::width=320 sink_7::height=240 ! \
  video/x-raw,format=RGB16 ! waylandsink \
  v4l2src device=/dev/video0 ! video/x-raw,width=320,height=240 ! comp.sink_0 \
  v4l2src device=/dev/video1 ! video/x-raw,width=320,height=240 ! comp.sink_1 \
  v4l2src device=/dev/video2 ! video/x-raw,width=320,height=240 ! comp.sink_2 \
  v4l2src device=/dev/video3 ! video/x-raw,width=320,height=240 ! comp.sink_3 \
  v4l2src device=/dev/video4 ! video/x-raw,width=320,height=240 ! comp.sink_4 \
  v4l2src device=/dev/video5 ! video/x-raw,width=320,height=240 ! comp.sink_5 \
  v4l2src device=/dev/video6 ! video/x-raw,width=320,height=240 ! comp.sink_6 \
  v4l2src device=/dev/video7 ! video/x-raw,width=320,height=240 ! comp.sink_7
```

The i.MX 8QXP only supports one `MX8XMIPI4CAM2` daughter card.

Chapter 13. Video Scaling and Rotation

This section shows how to perform video scaling and rotation with some GStreamer pipelines examples and its supported devices.

For i.MX 8 devices, the unique available solution to create an interface over Weston/Wayland is by using GPU to handle the screen position. So all the GStreamer pipelines in this sections use `glimagesink` for it.

13.1. i.MX 8

13.1.1. Video Scaling

In order to display different scaling results, uses the `glimagesink - render_rectangle` property:

For VGA resolution:

```
 # gst-launch-1.0 filesrc location=<name_of_the_video.mp4> ! qtdemux ! h264parse ! vpudec ! queue ! glimagesink render-rectangle='<0, 0,
720, 480>'
```

For Full HD resolution:

```
 #  gst-launch-1.0 filesrc location=<name_of_the_video.mp4> ! qtdemux ! h264parse ! vpudec ! queue ! glimagesink render-rectangle='<0, 0,
1920, 1080>'
```

13.1.2. Video Rotation

In order to rotate the video results, uses the `glimagesink - rotate-method` property:

To rotate 90 degrees:

```
 # gst-launch-1.0 filesrc location=<name_of_the_video.mp4> ! qtdemux ! h264parse ! vpudec ! queue ! glimagesink rotate-method=1
```

To rotate 180 degrees:

```
 # gst-launch-1.0 filesrc location=<name_of_the_video.mp4> ! qtdemux ! h264parse ! vpudec ! queue ! glimagesink rotate-method=2
```

To rotate 270 degrees:

```
 # gst-launch-1.0 filesrc location=<name_of_the_video.mp4> ! qtdemux ! h264parse ! vpudec ! queue ! glimagesink rotate-method=3
```

Chapter 14. Zero-copy Pipelines

This section approaches the `zero-copy` operations using GStreamer pipelines. The `zero-copy` support is an operation in which the CPU uses the data produced by one element but without requiring any type of transformation.

14.1. Pushing buffers

One way of using buffers to operate a `zero-copy` pipeline is to use property such as `io-mode`. This process can be very helpful in order to improve the execution speed of a video processing pipeline. Some of these types can be seen below:

14.1.1. Dmabuf

The `dmabuf` uses buffers of a hardware `DMA` in order to perform a `zero-copy` pipeline, as shown below:

```
$ gst-launch-1.0 v4l2src device=/dev/video0 num-buffers=300 io-mode=dmabuf ! \
  'video/x-raw,format=(string)NV12,width=1920,height=1080,framerate=(fraction)30/1' ! \
  queue ! v4l2h264enc output-io-mode=dmabuf-import ! avimux ! filesink location=test.avi
```

In this pipeline, the RAW format is stored by the `io-mode` property and then used further on the H.264 format encode process.

14.1.2. MMAP

The MMAP is a memory allocation process provided by the kernel that can perform the `zero-copy` procedure. One usage example is shown below:

```
$ gst-launch-1.0 v4l2src io-mode=2 device=/dev/video0 do-timestamp=true ! \
  'video/x-raw, width=1280, height=720, framerate=30/1, format=UYVY' ! autoconvert ! \
  'video/x-raw, width=1280, height=720, framerate=30/1, format=I420' ! autovideosink sync=false
```

14.2. CPU performance

In the pipelines described above, when comparing the use of CPU with each type of buffer usage, it is possible to see that the `zero-copy` adoption reduces the CPU usage, as shown in the table below:

Type of encode	Average CPU usage
Direct encode	102%
Encode with Dmabuf	15%
Encode with MMAP	70%

Therefore, you can see in some cases an average of 87% decrease in CPU usage with these pipelines.

For the test described in the table above, the board used was a i.MX 8QuadXPlus MEK B0 and the CPU usage was measured by the `top` command, while other boards can show different results for the pipelines.

Chapter 15. Debug Tools

This section describes some debug tools functionalities, how to use it, and when to use each one.

15.1. GStreamer standard debug

The most common GStreamer debug tool is the standard one. Use the following command to check all the debug options available:

```
$ --gst-debug-help
```

The example below shows the buffer movement:

```
$ GST_DEBUG=GST_BUFFER:5 gst-launch-1.0 -v filesrc location=cut_bbb_720.mov ! \
 decodebin ! imxvideoconvert_g2d ! waylandsink sync=false
```

As you may notice, the video performance was affected by the console log return. To avoid it, keep the debug values at a file:

```
$ GST_DEBUG_FILE=gst_debug.log GST_DEBUG=GST_BUFFER:5 gst-launch-1.0 -v filesrc location=cut_bbb_720.mov ! \
 decodebin ! imxvideoconvert_g2d ! waylandsink sync=false
```

15.2. Graphviz

One special way to debug the pipeline and its capabilities is over `.dot` files, which can be used to create a diagram of the pipeline.

For it, set a directory to save the `.dot` files:

```
 # export GST_DEBUG_DUMP_DOT_DIR=/tmp/
```

Then, run the pipeline:

```
gst-launch-1.0 -v videotestsrc num-buffers=300 ! waylandsink
```

Check the `.dot` files generated at the `/tmp` directory and copy the `PLAYING_PAUSED` one to the host machine.

At the **host machine**, install `graphviz`:

```
 # apt-get install graphviz
```

Still on the host machine, convert the `.dot` file to yours preferred image file format, in this case, `PNG`:

```
$ dot -Tpng <PLAYING_PAUSED_FILE>.dot > diagram.png
```

Open the image to check the results:

```
$ eog diagram.png
```