The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Section</th>
<th>Revision Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>06/19/2020 Version 1.0</td>
<td>Initial release.</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>
# Table of Contents

## Revision History

[2]

## Chapter 1: Document Scope

[5]

## Chapter 2: Introduction to Zynq UltraScale+ MPSoC: An Ideal System for Multimedia

- Zynq UltraScale+ MPSoC Processing System Highlights [7]
- Zynq Ultrascale+ MPSoC Multimedia Blocks [8]

## Chapter 3: Multimedia PL IP

- Video Capture [12]
- Video Display [18]
- PHY Controllers [22]
- Audio [23]
- Test Pattern Generators [25]

## Chapter 4: Video Pipelines

- HDMI-Rx Capture and Audio Pipeline [28]
- HDMI-Tx Pipeline [31]
- SDI-Rx Pipeline [33]
- SDI-Tx Pipeline [35]
- MIPI CSI-Rx Pipeline [37]
- TPG Pipeline [41]
- SDI Audio Pipeline [43]
- Streaming-Based SCD [45]
- Memory-Based SCD [49]

## Chapter 5: VCU Codec Features

- VCU Encoder Support [50]
- Rate Control Modes [51]
- Different Group of Picture Configuration [52]
- Parameter List [52]
Chapter 6: GStreamer Multimedia Framework................................. 55
  Install and Set Up Gstreamer.......................................................... 55
  Pixel Format Support........................................................................ 56
  GStreamer Plugins........................................................................... 56
  Raw Use Case.................................................................................... 58
  Decode/Encode Example.................................................................... 59
  Camera Interface and Support......................................................... 60
  Video Pipelines................................................................................ 60

Chapter 7: Debug............................................................................... 71
  Media Control for Capture Link Up.................................................. 71
  Modetest for Display Link Up......................................................... 73
  GStreamer Debugging Techniques Using GST-launch................... 75

Chapter 8: Performance and Optimization........................................ 78
  Performance Improvement from the GStreamer Perspective.......... 79

Appendix A: Additional Resources and Legal Notices..................... 82
  Xilinx Resources............................................................................. 82
  Documentation Navigator and Design Hubs................................. 82
  References....................................................................................... 82
  Please Read: Important Legal Notices....................................... 84
Chapter 1

Document Scope

This document describes the architecture and features of multimedia systems with PS + PL + VCU IP. Learning about this architecture can help you understand the complete multimedia system, and facilitates integration with third party IP to develop custom multimedia pipelines. This document lists several audio-video pipelines to demonstrate expanded and enhanced multimedia solutions. It covers the VCU codec parameters that you must fine-tune for various video use cases, as well as information for debugging the multimedia pipeline.

This document consists of the following chapters:

- **Chapter 2: Introduction to Zynq UltraScale+ MPSoC: An Ideal System for Multimedia:** Provides a brief introduction to the architecture and multimedia processing engines of the Zynq® UltraScale™ MPSoC.
- **Chapter 3: Multimedia PL IP:** Discusses the features of the Multimedia PL Fabric IP.
- **Chapter 4: Video Pipelines:** Showcases the example video pipelines created using Xilinx® IP. This document also discusses device tree changes required as part of Petalinux build, to bring up video pipelines for capture and display, and to enable audio support.
- **Chapter 5: VCU Codec Features:** Describes the VCU parameters in detail.
- **Chapter 6: GStreamer Multimedia Framework:** Reveals the GStreamer pipeline for audio-video. Explains pipeline configuration, and each GStreamer element required to run the audio-video pipeline.
- **Chapter 7: Debug:** Aids in debugging multimedia pipeline issues. The GST_shark tool measures the pipeline latency, and helps debug latency issues.
- **Chapter 8: Performance and Optimization:** Specifies the techniques to enhance the performance of the GStreamer pipeline.
Chapter 2

Introduction to Zynq UltraScale+ MPSoC: An Ideal System for Multimedia

The Zynq® UltraScale+™ MPSoC is a heterogeneous, multiprocessing SoC with both 64-bit and 32-bit processing subsystems, dedicated hardened engines for real-time graphics and video processing, advanced high-speed peripherals, and programmable logic, serving cost-sensitive and high-performance applications using a single platform, and industry-standard tools. The Zynq UltraScale+ MPSoC is created with complete flexibility at both the software and hardware levels, while integrating the most diverse number of specialized engines the embedded market has ever seen.

To achieve high-performance and low-latency, advanced multimedia systems must have the right processing engines and the capability to add custom logic. They must also support any-to-any connectivity required by the wide range of multimedia devices available today. Traditionally, these requirements suggest a multi-chip solution. While a multi-chip solution might provide the required multimedia and connectivity functions, it can also lead to high power consumption.

A single-chip solution like the Zynq UltraScale+ MPSoC is ideal for such scenarios. The existence of custom logic for hardware acceleration, or any-to-any connectivity on the same device can lead to significant power savings. In addition to the mix of processing engines, hardware codecs, and support for custom logic, the Zynq UltraScale+ MPSoC places these components in different power domains with independent power rails. You can use this configuration to design optimized power management schemes for the entire system. The Zynq UltraScale+ MPSoC is built upon the 16nm FinFET process node, resulting in greater performance and lower power consumption, enabling the design of power-efficient next-generation multimedia systems.

The Zynq UltraScale+ MPSoC combines a powerful processing system (PS) and user-programmable logic (PL) into the same device. The PS features the Arm® flagship Cortex™-A53 64-bit quad-core or dual-core processor (APU), Cortex™-R5F dual-core real-time processor (RPU), and Graphical Processor Unit (GPU).
The Zynq UltraScale+ MPSoC device variants include dual application processor (CG) devices, quad application processor and GPU (EG) devices, and video codec (EV) devices, creating unlimited possibilities for a wide variety of applications. The Zynq UltraScale+ MPSoC EV devices build on the powerful EG platform, and add an integrated H.264/H.265 video codec capable of simultaneous encode and decode up to 4Kx2K (60fps). Designed with high definition video in mind, EV devices are ideal for multimedia, Advanced Driver Assistance Systems (ADAS), surveillance, and other embedded vision applications.

Zynq UltraScale+ MPSoC Processing System Highlights

- Applications Processing Unit (APU) with either quad-core (EG and EV devices) or dual-core (CG devices) Arm® Cortex™-A53 processors:
  - Arm®v8 architecture supporting 32- or 64-bit modes
  - Ideal for Linux and bare-metal SMP/AMP application systems
- Real-time processing unit (RPU) with dual-core Arm® Cortex™-R5F processors:
  - Low-latency, highly deterministic performance
  - APU offloading
- Integrated hardened multimedia blocks:
  - Graphics processing unit (GPU) (Arm® Mali™ 400MP2)
  - 4Kx2K 60fps video encoder/decoder (VCU) (in select devices)
  - 4Kx2K 30fps DisplayPort interface
- Integrated high-speed peripherals:
  - PCIe® Gen2 root complex and integrated Endpoint block x4 lanes
  - USB 3.0/2.0 with host, device, and OTG modes
  - Gigabit Ethernet with jumbo frames and precision time protocol
  - SATA 3.1 host
  - Dedicated quad transceivers up to 6 Gb/s
- General and boot peripherals:
  - CAN, I2C, QSPI, SD, eMMC, and NAND flash interfaces
  - GPIO, UART, and trace ports
• 6-port DDR controller with ECC, supporting x32 and x64 DDR3, DDR3L, LPDDR3, LPDDR4, DDR4
• Integrated Platform Management Unit (PMU) supporting multiple power domains
• Integrated Configuration Security Unit (CSU)
• TrustZone support
• Peripheral and memory protection

For more information, see Zynq UltraScale+ Device Technical Reference Manual (UG1085).

Zynq Ultrascale+ MPSoC Multimedia Blocks

The Zynq® UltraScale+™ MPSoC has hardened, integrated multimedia video and graphics processing blocks that operate at up to 4K video rates, letting the CPUs focus on the applications, delivering greater system performance, and power efficiency across the device. These blocks provide a powerful graphics and video management pipeline that can help lower Bill Of Materials (BOM) costs by removing extraneous devices from the board. The following sections gives a brief description of multimedia blocks namely Graphics Processing Unit (GPU), DisplayPort (DP) interface, and Video Code Unit (VCU).

Graphics Processing Unit

The Graphics Processing Unit (GPU) is the Arm® Mali-400 MP2 and resides in the Zynq UltraScale+ MPSoC processing system (PS). The GPU can generate video information through its dedicated, parallel engines much faster than competing ASSPs that rely on the CPU to handle graphics processing, cheaper, and with less power consumption than solutions that rely on the designer to add an off-chip GPU engine.

The GPU accelerates both 2D and 3D graphics with a fully programmable architecture that provides support for both shader-based and fixed-function graphics APIs. It includes anti-aliasing for optimal image quality, with virtually no additional performance overhead.

Zynq UltraScale+ MPSoC GPU Highlights
• Arm® Mali-400 MP2
• Up to 667MHz performance in the fastest speed grade
• One geometry processor, two pixel processors
• Dedicated 64KB shared L2 cache
• Dedicated memory management unit
• OpenGL ES 2.0 and OpenGL ES 1.1 support
- OpenVG 1.1 API support
- GPU power gating on each of the three engines
- 1334 Mpixels/sec pixel fill rate
- 72.6 Mtriangles/sec
- 21.34 Gflops floating point shading


DisplayPort Interface

The Zynq UltraScale+ MPSoC also includes a hardened DisplayPort (DP) interface module. The DisplayPort interface is located in the PS and can be multiplexed to one of four dedicated high-speed serial transceivers operating at up to 6Gb/s. This eliminates the need for additional display chips to further reduce system BOM cost. The DisplayPort interface is based on the Video Electronics Standards Association (VESA) V-12a specification and provides multiple interfaces to process live audio/video feeds from either the PS or the PL, or stored audio/video from memory frame buffers. It simultaneously supports two audio/video pipelines, providing on-the-fly rendering features like alpha blending, chroma resampling, color space conversion, and audio mixing. This block also includes a dedicated video Phased-Lock Loop (PLL) for generating sync clocks.

Zynq UltraScale+ MPSoC DisplayPort Interface Highlights

- Up to 4K x 2k video @30 Hz video resolution
- Y-only, YCbCr444, YCbCr422, YCbCr420, RGB video formats
- 6, 8, 10, or 12 bits per color components
- 36-bit native video input interface for live video
- Captured video interface from frame buffers using built-in DMA
- Two-plane rendering pipeline
- Up to two channels of audio, 24-bit at 48 KHz
- Dedicated video PLL
- Controller to generate video timing for captured video
- System time clock (STC) compliant with ISO/IEC 13818-1

For more information on the DP Interface, see Chapter 33: Display Port Controller of the Zynq UltraScale+ Device Technical Reference Manual (UG1085).
**Video Codec Unit**

The H.264/H.265 Video Codec Unit (VCU) is an integrated block in the PL of Zynq UltraScale+ MPSoC EV devices. Unlike software codecs, the VCU in Zynq UltraScale+ MPSoC EV devices provides low-power, high-performance compression and decompression of H.264/H.265 video data. This makes it an ideal candidate for real-time streaming of Ultra High Definition (UHD) videos on the network, where it can save a considerable amount of storage and network bandwidth.

VCU support for both H.264 and H.265 standards provides a compelling capability to develop solutions in line with current market needs (H.264), as well as advance-generation requirements (H.265). The ability to encode and decode simultaneously with low latency makes it a perfect fit for video conferencing and transcoding from H.264 to H.265 or vice-versa. Multistream, multicodec encoding and decoding suits the requirement for video surveillance applications such as DVRs, video servers, and multistream IP camera head-ends.

**VCU Highlights**

- Multi-standard encoding/decoding support, including:
  - ISO MPEG-4 Part 10: Advanced Video Coding (AVC)/ITU H.264
  - ISO MPEG-H Part 2: High Efficiency Video Coding (HEVC)/ITU H.265
  - HEVC: Main, Main Intra, Main 10, Main10 Intra, Main 4:2:2 10, Main 4:2:2 10 Intra up to Level 5.1 High Tier
  - AVC: Baseline, Main, High, High10, High 4:2:2, High10 Intra, High 4:2:2 Intra up to Level 5.2

- Support simultaneous encoding and decoding of up to 32 streams with a maximum aggregated bandwidth of 3840x2160 at 60fps

- Low latency rate control

- Flexible rate control: Constant Bit Rate (CBR), Variable Bit Rate (VBR), and Constant Quantization Parameter (QP)

- Supports simultaneous encoding and decoding up to 4K UHD resolution at 60 Hz

**RECOMMENDED:** 4k (3840x2160) video, and lower resolutions are supported in all speed grades. However, 4K UHD (4096x2160) requires -2 or -3 speed grade.

- Supports 8K UHD video at reduced frame rate (~15 Hz)

- Progressive support for H.264/H.265 and Interlace support for H.265

- Video input: YCbCr 4:2:2, YCbCr 4:2:0, and Y-only (monochrome), 8- and 10-bit per color channel

For VCU Codec features, see Chapter 5: VCU Codec Features.
For more information, see *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide (PG252).*
Chapter 3

Multimedia PL IP

The Arm® Cortex™-A53 cores, along with the memory unit and many peripherals on the Zynq® UltraScale+™ MPSoC, play a strong role in managing and capturing the data from many different sources before making it available to the Video Codec Unit (VCU). Processing System (PS) peripherals such as USB and Ethernet can be used for streaming video devices such as camcorders, network cameras, and webcams. Custom logic can be designed in the Programmable Logic (PL) to capture raw video from live sources such as Serial Digital Interface (SDI) RX, High-Definition Multimedia Interface (HDMI) RX, and Mobile Industry Processor Interface (MIPI) Camera Serial Interface (CSI) IPs. The VCU can then encode the raw video into Advanced Video Coding (AVC) or High-Efficiency Video Coding (HEVC) compressed bitstream format. Similarly, decoded raw video can be transmitted to an external display unit. The video can then be displayed using the DisplayPort controller in the PS or by creating a relevant IP such as HDMI TX, SDI TX, or MIPI Display Serial Interface (DSI).

Xilinx provides a collection of multimedia IP, which are available in the Xilinx Vivado® IP catalog. The PL provides the required hardware for the IP, resulting in performance improvements suitable for next-generation technology.

This chapter describes the Xilinx LogiCORE™ PL IP that are commonly used in multimedia applications. These IP can be categorized into the following types based on their functionality:

1. Video Capture
2. Video Display
3. PHY Controllers
4. Audio
5. Test Pattern Generators

---

Video Capture

The IPs that are part of the Capture pipeline, capture video frames into DDR memory from either a HDMI source, MIPI CSI-2 image sensor, SDI source, or test pattern generator (TPG). Additionally, video can be sourced from a SATA drive, USB 3.0 device, or an SD card.
HDMI Receiver Subsystem

The HDMI 1.4/2.0 Receiver Subsystem is a hierarchical IP that bundles a collection of HDMI IP sub-cores and outputs them as a single IP. Xilinx Subsystem IPs are ready-to-use, and do not require the user to assemble sub-cores in order to produce a working system.

Features

- HDMI 2.0 and 1.4b compatible
- 2 or 4 symbol/pixel per clock input
- Supports resolutions up to 4,096 x 2,160 at 60 fps
- 8, 10, 12, and 16-bit deep color support
- Dynamic support for RGB, YUV 4:4:4, YUV 4:2:2, YUV 4:2:0 color formats
- Supports Advanced eXtensible Interface (AXI4)-Stream video input stream and native video input stream
- Audio support for up to 8 channels
- High bit rate (HBR) Audio
- Optional HDCP 2.2/1.4 encryption support
- Info frames
- Data Display Channel (DDC)
- Hot-Plug Detection
- 3D video support
- Optional video over AXIS compliant NTSC/PAL support
- Optional video over AXIS compliant YUV420 support
- Optional Hot Plug Detect (HPD) active polarity
- Optional cable detect active polarity

For more information, refer to the HDMI 1.4/2.0 Receiver Subsystem Product Guide (PG236).

SMPTE UHD-SDI Receiver Subsystem

The Society of Motion Picture and Television Engineers (SMPTE) UHD-SDI receiver subsystem implements an SDI receive interface in accordance with the SDI family of standards. The subsystem receives video from a native SDI and generates an AXI4-Stream video. The subsystem allows fast selection of the top-level parameters, and automates most of the lower level parameterization. The AXI4-Stream video interface allows a seamless interface to other AXI4-Stream-based subsystems.
Features

- Supports AXI4-Stream, native video and native SDI user interfaces
- Support for 2 pixel per sample
- 10-bit per color component
- Supports YUV 4:4:4, YUV 4:2:2, and YUV 4:2:0 color space
- AXI4-Lite interface for register access to configure different subsystem options
- Audio support
- Standards compliance:
  - SMPTE ST 259: SD-SDI at 270 Mb/s
  - SMPTE RP 165: EDH for SD-SDI
  - SMPTE ST 292: HD-SDI at 1.485 Gb/s and 1.485/1.001 Gb/s
  - SMPTE ST 372: Dual Link HD-SDI
  - SMPTE ST 424: 3G-SDI with data mapped by any ST 425-x mapping at 2.97 Gb/s and 2.97/1.001 Gb/s
  - SMPTE ST 2081-1: 6G-SDI with data mapped by any ST 2081-x mapping at 5.94 Gb/s and 5.94/1.001 Gb/s
  - SMPTE ST 2082-1: 12G-SDI with data mapped by any ST 2082-x mapping at 11.88 Gb/s and 11.88/1.001 Gb/s
  - Dual link and quad link 6G-SDI and 12G-SDI supported by instantiating two or four SMPTE UHD-SDI RX subsystems

For more information, see SMPTE UHD-SDI Receiver Subsystem Product Guide (PG290).

MIPI CSI-2 Receiver Subsystem

The Mobile Industry Processor Interface (MIPI) Camera Serial Interface (CSI-2) RX subsystem implements a CSI-2 receive interface according to the MIPI CSI-2 standard v2.0 with underlying MIPI D-PHY standard v1.2. The subsystem captures images from MIPI CSI-2 camera sensors and outputs AXI4-Stream video data ready for image processing. The subsystem allows fast selection of the top-level parameters and automates most of the lower level parameterization. The AXI4-Stream video interface allows a seamless interface to other AXI4-Stream-based subsystems.

Features

- Support for 1 to 4 D-PHY lanes
- Line rates ranging from 80 to 2500 Mb/s (Zynq UltraScale+ MPSoC EV devices)
• Multiple Data Type support (RAW, RGB, YUV)
• Filtering based on Virtual Channel Identifier
• Support for 1, 2, or 4 pixels per sample at the output
• AXI4-Lite interface for register access to configure different subsystem options
• Dynamic selection of active lanes within the configured lanes during subsystem generation
• Interrupt generation to indicate subsystem status information
• Internal D-PHY allows direct connection to image sources

For more information, see MIPI CSI-2 Receiver Subsystem Product Guide (PG232).

**Sensor Demosaic**

The Xilinx LogiCORE IP Sensor Demosaic core provides an optimized hardware block that reconstructs sub-sampled color data for images captured by a Bayer image sensor. CMOS and CCD images sensors leverage a Color Filter Array (CFA), which passes only the wavelengths associated with a single primary color to a given pixel on the substrate. Each pixel is therefore sensitive to only red, green, or blue. The Sensor Demosaic IP provides an efficient and low-footprint solution to interpolate the missing color components for every pixel.

**Features**

• RGB Bayer image sensor support
• One, two, four, or eight pixel-wide AXI4-Stream video interface
• Supports 8, 10, 12, and 16 bits per color component
• Supports spatial resolutions from 64 x 64 up to 8192 x 4320
• Supports 4K 60 fps in all supported device families

For more information, refer to the Sensor Demosaic LogiCORE IP Product Guide (PG286).

**Gamma LUT**

The Xilinx LogiCORE IP Gamma Look-up Table (LUT) core provides customers with an optimized hardware block for manipulating image data to match the response of display devices. This core is implemented using a look-up table structure that is programmed to implement a gamma correction curve transform on the input image data.

**Features**

• Programmable gamma table supports gamma correction or any user defined function
• Three channel independent look-up table structure
Video Processing Subsystem

The Video Processing Subsystem (VPSS) is a collection of video processing IP subcores, bundled together in hardware and software, abstracting the video processing pipe. It provides the end-user with an out of the box ready to use video processing core, without having to learn about the underlying complexities. The Video Processing Subsystem enables streamlined integration of various processing blocks including scaling, deinterlacing, color space conversion and correction, chroma resampling, and frame rate conversion.

Features

- One, two or four pixel-wide AXI4-Stream video interface
- Video resolution support up to UHD at 60 fps
- Run-time color space support for RGB, YUV 4:4:4, YUV 4:2:2, YUV 4:2:0
- 8, 10, 12, and 16 bits per component support
- Deinterlacing
- Scaling
- Color space conversion and correction
- Chroma resampling between YUV 4:4:4, YUV 4:2:2, YUV 4:2:0
- Frame rate conversion using dropped/ repeated frames

For more information, see Video Processing Subsystem Product Guide (PG231).

Video Scene Change Detection

The Video Scene Change Detection (SCD) IP provides a video processing block that implements the scene change detection algorithm. The IP core calculates a histogram on a vertically subsampled luma frame for consecutive frames. The histograms of these frames are then compared using the sum of absolute differences (SAD). This IP core is programmable through a comprehensive register interface to control the frame size, video format, and subsampling value.
Features

• Input streams can be read from memory mapped AXI4 interface or from AXI4-Stream interface
• Supports up to eight streams in the memory mapped mode and one stream in the stream mode
• Supports Y8 and Y10 formats for memory interface
• Supports RGB, YUV 444, YUV 422, and YUV 420 formats for stream interface
• Supports 8, 10, 12, and 16 bits per color component input and output on AXI4-Stream interface
• Supports one, two, or four-pixel width for stream mode, and one-pixel width for memory mode
• Supports spatial resolutions ranging from 64 × 64 up to 8,192 × 4,320
• Supports 4k 60 fps in all supported device families
• Supports 32-bit and 64-bit DDR memory address access

For more information, see Video Scene Change Detection LogiCORE IP Product Guide (PG322).

Video Frame Buffer Write

The Video Frame Buffer Write core provide high-bandwidth direct memory access between memory and AXI4-Stream video type target peripherals that support the AXI4-Stream Video protocol.

Features

• AXI4 Compliant
• Memory Video Format support for: RGBX8, BGRX8, YUVX8, YUYV8, UYVY8, RGBA8, BGRA8, YUVA8, RGBX10, YUVX10, Y_YUV8, Y_YUV8_420, RGB8, BGR8, YUV8, Y_YUV10, Y_YUV10_420, Y8, Y10
• Provides programmable memory video format
• Supports progressive and interlaced video
• Supports 8 and 10-bits per color component on stream interface and memory interface
• Supports spatial resolutions from 64 × 64 up to 8192 × 4320
• Supports 4K60 in all supported device families
For more information, refer to the *Video Frame Buffer Read and Video Frame Buffer Write LogiCORE IP Product Guide* (PG278).

**VCU Sync IP**

The VCU Sync IP core acts as a fence IP between the Video DMA IP and the VCU IP. It is used in Multimedia applications that need ultra-low latencies. The VCU Sync IP does a AXI transaction level tracking so that producer and consumer can be synchronized at the granularity of AXI transactions, instead of granularity of Video Buffer level.

The VCU Sync IP is responsible for synchronizing buffers between the Capture DMA and the VCU encoder. Capture hardware writes video buffers in raster scan order. The VCU Sync IP monitors the buffer level while capture element is writing into DRAM, and allows the encoder to read input buffer data if the requested data is already written by DMA. Otherwise, it blocks the encoder until the DMA completes its writes.

**Features**

- The VCU Sync IP core can track up to four producer transactions simultaneously
- Each channel can track up to three buffer sets
- Each buffer set has Luma and Chroma buffer features
- Each consumer port can hold 256 AXI transactions without back-pressure to the consumer
- In encoder mode, the Sync IP core supports the tracking and control of four simultaneous channels

For more information, see Chapter 7 of *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

**Video Display**

The IPs that are part of the display pipeline read video frames from memory and send them to a monitor through either the DisplayPort Tx controller inside the PS, the SDI transmitter subsystem through the PL, or the HDMI transmitter subsystem through the PL.

**HDMI Transmitter Subsystem**

The HDMI 1.4/2.0 Transmitter Subsystem is a hierarchical IP that bundles a collection of HDMI IP sub-cores and outputs them as a single IP. Xilinx Subsystem IP are ready-to-use, and do not require the user to assemble sub-cores in order to produce a working system.
Features

- HDMI 2.0 and 1.4b compatible
- 2 or 4 symbol/pixel per clock input
- Supports resolutions up to 4,096 x 2,160 @ 60 fps
- 8, 10, 12, and 16-bit deep color support
- Dynamic support for RGB, YUV 4:4:4, YUV 4:2:2, YUV 4:2:0 color formats
- Supports AXI4-Stream video input stream and native video input stream
- Audio support for up to 8 channels
- High bit rate (HBR) Audio
- Optional HDCP 2.2/1.4 encryption support
- Info frames
- Data Display Channel (DDC)
- Hot-Plug Detection
- 3D video support
- Optional video over AXIS compliant NTSC/PAL support
- Optional video over AXIS compliant YUV420 support
- Optional HPD active polarity

For more information, see *HDMI 1.4/2.0 Transmitter Subsystem Product Guide (PG235)*.

## SMPTE UHD-SDI Transmitter Subsystem

The SMPTE UHD-SDI transmitter subsystem implements a SDI transmit interface in accordance with the SDI family of standards. The subsystem accepts video from an AXI4-Stream video interface and outputs a native video stream. It allows fast selection of top-level parameters, and automates most of the lower level parameterization. The AXI4-Stream video interface allows a seamless interface to other AXI4-Stream-based subsystems.

### Features

- Supports AXI4-Stream, native video and native SDI user interfaces
- Supports 2 pixels per sample
- 10-bit per color component
- Supports YUV 4:4:4, YUV 4:2:2, and YUV 4:2:0 color space
- Provision to insert ancillary data
• AXI4-Lite interface for register access to configure different subsystem options

• Standards compliance:
  ○ SMPTE ST 259: SD-SDI at 270 Mb/s
  ○ SMPTE ST 292: HD-SDI at 1.485 Gb/s and 1.485/1.001 Gb/s
  ○ SMPTE ST 372: Dual Link HD-SDI
  ○ SMPTE ST 424: 3G-SDI with data mapped by any ST 425-x mapping at 2.97 Gb/s and 2.97/1.001 Gb/s
  ○ SMPTE ST 2081-1: 6G-SDI with data mapped by any ST 2081-x mapping at 5.94 Gb/s and 5.94/1.001 Gb/s
  ○ SMPTE ST 2082-1: 12G-SDI with data mapped by any ST 2082-x mapping at 11.88 Gb/s and 11.88/1.001 Gb/s
  ○ Dual link and quad link 6G-SDI and 12G-SDI are supported by instantiating two or four UHD-SDI transmitter subsystems

For more information, see SMPTE UHD-SDI Transmitter Subsystem Product Guide (PG289).

### Video Mixer

The Xilinx LogiCORE IP Video Mixer core provides the following features:

- Flexible video processing block for alpha blending and compositing multiple video and/or graphics layers
- Support for up to eight layers, with an optional logo layer, using a combination of video inputs from either frame buffer or streaming video cores through AXI4-Stream interfaces
- Programmable core through a comprehensive register interface to control frame size, background color, layer position, and the AXI4-Lite interface
- Comprehensive set of interrupt status bits for processor monitoring

### Features

- Supports (per pixel) alpha-blending of seventeen video/graphics layers
- Optional logo layer with color transparency support
- Layers can either be memory mapped AXI4 interface or AXI4-Stream
- Provides programmable background color
- Provides programmable layer position and size
- Provides upscaling of layers by 1x, 2x, or 4x
- Optional built-in color space conversion and chroma re-sampling
• Supports RGB, YUV 444, YUV 422, YUV 420
• Supports 8, 10, 12, and 16 bits per color component input and output on stream interface, 8-bit and 10-bit per color component on memory interface
• Supports semi-planar memory formats next to packed memory formats
• Supports spatial resolutions from 64 × 64 up to 8192 × 4320
• Supports 8K30 in all supported device families

For more information, see Video Mixer LogiCORE IP Product Guide (PG243).

Video Frame Buffer Read

The Video Frame Buffer Read core provides high-bandwidth direct memory access between memory and AXI4-Stream video type target peripherals that support the AXI4-Stream video protocol.

Features

• AXI4 Compliant
• Memory Video Format support for: RGBX8, BGRX8, YUVX8, YUYV8, UYVY8, RGBA8, BGRA8, YUVA8, RGBX10, YUVX10, Y_UV8, Y_UV8_420, RGB8, BGR8, YUV8, Y_UV10, Y_UV10_420, Y8, Y10
• Provides programmable memory video format
• Supports progressive and interlaced video
• Supports 8 and 10-bits per color component on stream interface and memory interface
• Supports spatial resolutions from 64 x 64 up to 8192 x 4320
• Supports 4K60 video in all supported device families

For more information, see Video Frame Buffer Read and Video Frame Buffer Write LogiCORE IP Product Guide (PG278).
PHY Controllers

Video PHY Controller

The Xilinx Video PHY Controller IP core is designed for enabling plug-and-play connectivity with Video (DisplayPort and HDMI technology) MAC transmit or receive subsystems. The interface between the video MAC and PHY layers is standardized to enable ease of use in accessing shared transceiver resources. The AXI4-Lite interface is provided to enable dynamic accesses of transceiver controls/status.

Features

- AXI4-Lite support for register accesses
- Protocol Support for DisplayPort and HDMI
- Full transceiver dynamic reconfiguration port (DRP) accesses and transceiver functions
- Independent TX and RX path line rates (device specific)
- Single quad support
- Phase-locked loop (PLL) switching support from software
- Transmit and receiver user clocking
- Protocol specific functions for HDMI
  - HDMI clock detector
  - Use of fourth GT channel as TX TMDS clock source
  - Non-integer data recovery unit (NI-DRU) support for lower line rates. NI-DRU support is for the HDMI protocol only
- Advanced clocking support

For more information, see Video PHY Controller LogiCORE IP Product Guide (PG230).

SMPTE UHD-SDI

The SDI family of standards from the SMPTE is widely used in professional broadcast video equipment. The SMPTE UHD-SDI core supports SMPTE SDI data rates from SD-SDI through 12G-SDI. The core supports both transmit and receive.

Features

- SMPTE ST 259, SMPTE RP 165, SMPTE ST 292, SMPTE ST 372, SMPTE ST 424, SMPTE ST 2081-1, SMPTE ST 2082-1, SMPTE ST 352
For more information, see *SMPTE UHD-SDI LogiCORE IP Product Guide (PG205).*

## Audio

### Audio Formatter

The Xilinx LogiCORE IP Audio Formatter core is a soft Xilinx IP core that provides high-bandwidth direct memory access between memory and AXI4-Stream target peripherals supporting audio data.

**Features**

- Supports 2, 4, 6, or 8 audio channels
- Supports 8, 16, 20, 24, and 32 bits PCM data width
- Independent S2MM (write to Memory) and MM2S (read from memory) operations
- Supports Interleaved and Non-interleaved modes of data packaging in memory
- Supported data formats for audio stream writes to memory (S2MM)
  - AES to AES
  - AES to PCM (includes AES Decoder)
  - PCM to PCM
- Supported data formats for audio stream reads from memory (MM2S)
  - AES to AES
  - AES to PCM
  - PCM to PCM
  - PCM to AES (Includes AES Encoder)
- Supports timeout feature in case of S2MM
- Interrupt generation on completion of every Period size of data, or when an error occurs
- Support to pad 0s if some channels in S2MM are missing samples
- Can handle random order of audio channels in incoming stream in S2MM
- Supports graceful halt and soft reset by completion of pending AXI4 transactions
- Supports selection of synchronous or asynchronous clocks through the IP configuration wizard
For more information, see Audio Formatter Product Guide (PG330).

**UHD SDI Audio**

The UHD SDI Audio core is configurable as an Audio Embedder or an Audio Extractor.

When configured as an Audio Embedder, it can embed up to 32 channels of AES3 audio data that is transmitted over an AXI4-Stream Audio Interface onto an SDI stream.

Similarly, when configured as an Audio Extractor, it can extract up to 32 channels of audio data from the incoming SDI stream, and output them in AES3 format on the AXI4-Stream Audio Interface. In both configurations, it supports multiple audio sample rates (32 KHz, 44.1 KHz, and 48 KHz).

The UHD SDI Audio core is designed in accordance with SMPTE ST 272 for SD-SDI, SMPTE ST 299-1 for HD-SDI, and SMPTE ST 299-1 & 2 for 3G/6G/12G-SDI.

**Features**

- Supports up to 32 channels of audio
- 20/24-bit audio at multiple sample rates (32 KHz, 44.1 KHz, and 48 KHz)
- Synchronous and asynchronous audio support
- Supports 192-bit AES3 channel status extraction
- Reports the presence and status of audio groups on the incoming SDI stream
- AXI4-Stream interface to carry audio samples in AES3 format
- AXI4-Lite and port based interface for configuration of the core

For more information, see UHDSI Audio LogiCORE IP Product Guide (PG309).

**I2S Transmitter and Receiver**

The Xilinx® LogiCORE™ IP I2S Transmitter and LogiCORE Receiver cores are soft Xilinx IP cores for use with the Xilinx Vivado® Design Suite, which makes it easy to implement the inter-IC sound (I2S) interface used to connect audio devices for transmitting and receiving PCM audio.

**Features**

- AXI4-Stream compliant
- Supports up to four I2S channels (up to eight audio channels)
- 16/24-bit data
- Supports master I2S mode
• Configurable FIFO depth
• Supports the AES channel status extraction/insertion

For more information, refer to the I2S Transmitter and Receiver LogiCORE IP Product Guide (PG308).

**Audio Clock Recovery Unit**

The Audio Clock Recovery Unit is a soft IP core that provides an easy mechanism to recover the audio sampling clock from a given reference clock. It can be used with HDMI or DisplayPort receivers to recover the audio sampling clock.

**Features**

• A sampling clock recovery from a reference clock
• Fixed audio sampling clock recovery
• Loop control-based audio sampling clock recovery
• Compatibility with HDMI and DisplayPort

For more information, see Audio Clock Recovery Unit Product Guide (PG335).

---

**Test Pattern Generators**

**Video Test Pattern Generator**

The Video Test Pattern Generator core generates test patterns for video system bring up, evaluation, and debugging. The core provides a wide variety of test patterns enabling you to debug and assess video system color, quality, edge, and motion performance. The core can be inserted in an AXI4-Stream video interface that allows user-selectable pass-through of system video signals and test pattern insertion.

**Features**

• Color bars
• Zone plate with adjustable sweep and speed
• Temporal and spatial ramps
• Moving box with selectable size and color over any available test pattern
• RGB, YUV 444, YUV 422, YUV 420 AXI4-Stream data interfaces
• AXI4-Lite control interface
• Supports 8, 10, 12, and 16-bits per color component input and output
• Supports spatial resolutions from 64 x 64 up to 10328 x 7760
  ◦ Supports 4K60 video in all supported device families

For more information, see Video Test Pattern Generator LogiCORE IP Product Guide (PG103).

### Video Timing Controller

The Video Timing Controller core is a general-purpose video timing generator and detector. The core is highly programmable through a comprehensive register set allowing control of various timing generation parameters. This programmability is coupled with a comprehensive set of interrupt bits that provide easy integration into a processor system for in-system control of the block in real-time. The Video Timing Controller is provided with an optional AXI4-Lite compliant interface.

### Features

- Support for progressive or interlaced video frame sizes up to 16384 x 16384
- Direct regeneration of output timing signals with independent timing and polarity inversion
- Automatic detection and generation of horizontal and vertical video timing signals
- Support for multiple combinations of blanking or synchronization signals
- Automatic detection of input video control signal polarities
- Support for detection and generation of horizontal delay of vertical blank/sync
- Programmable output video signal polarities
- Generation of up to 16 additional independent output frame synchronization signals
- Optional AXI4-Lite processor interface
- High number of interrupts and status registers for easy system control and integration

For more information, see Discrete Fourier Transform LogiCORE IP Product Guide (PG106).
Video Pipelines

This chapter explains the example video capture, the display and processing pipelines designs that can be generated using the Xilinx® Multimedia IPs, and how to build and generate bootable images using the Petalinux tool.

See chapter 5 in Zynq UltraScale+ MPSoC ZCU106 Video Codec Unit Targeted Reference Design User Guide (UG1250) for examples of capture, display, and video pipelines designs that can be generated on the ZCU106 board using the Xilinx Multimedia IPs.

PetaLinux is an Embedded Linux System Development Kit targeting Xilinx FPGA-based System-on-Chip designs. Refer to the PetaLinux Tools Documentation: Reference Guide (UG1144) on how to configure and build bootable images for the Vivado® generated designs using the Petalinux toolchain.

The following section explains the device tree (DT) and how the DT is generated using Petalinux. In addition, the section describes all the nodes and the properties that you need to manually update, to generate the bootable Linux images for video pipelines.

The DT is a data structure used during Linux configuration to describes the available hardware components of the embedded platform, including multimedia IP.

Embedded Linux uses the DT for platform identification, run-time configuration like bootargs, and the device node population.

**Device Tree Generation**

Generally for the SOCs, there are static dts/dtsi files, but when it comes to the FPGA there can be many complicated designs, in which the Programmable Logic (PL) IPs may vary or might have different configurations.

Device-Tree Generator (DTG), a part of the Xilinx Petalinux toolset, dynamically generates device tree file for FPGA components.

Once the device-tree is generated for a hardware design using Xilinx Petalinux tool, the components folder contains statically configured DT for the board PS files and DT files generated for FPGA components.

- **pl.dtsi**: Contains the DT node for FPGA PL components.
- **pcw.dtsi**: Contains dynamic properties of PS peripherals.
- **system-top.dts**: Contains the memory information, early console and the boot arguments.
- **zynqmp.dtsi**: Contains all the PS peripheral information and also the CPU information.
- **zynqmp-clk-ccf.dtsi**: Contains all the clock information for the peripheral IPs.
- **board.dtsi**: Based on the board, DTG generates this file under the same output directory.

In some cases, DTG cannot generate the input and output port information for the video pipelines as there is no information from the design/hdf. For such IPs, you need to manually update the port information inside the video pipeline IP nodes.

The following sections describe the video pipelines for which the DTG tool cannot generate complete pipeline information, and you need to manually update some of the device-tree properties and port information. The device tree properties and nodes to be updated are in the `system-user.dtsi` file. The `system-user.dtsi` file is part of the PetaLinux BSP in the path `BSP/project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`.

---

**HDMI-Rx Capture and Audio Pipeline**

*Figure 1: HDMI-Rx Capture and Audio Pipeline Block Diagram*

To create an HDMI-Rx Capture media pipeline, update the following pipeline DT nodes.
HDMI-Rx DT Node

To create a media graph on a HDMI-Rx subsystem DT node, use the following commands to update the clocks, Video PHY lanes, and ports.

```
&hdmi_input_v_hdmi_rx_ss_0 {
    clock-names = "s_axi_cpu_aclk", "s_axis_video_aclk";
    clocks = <&vid_s_axi_clk>, <&vid_stream_clk>; /* Provide optional
clocks information */
    phy-names = "hdmi-phy0", "hdmi-phy1", "hdmi-phy2";
    /* Provide Video phy interface lanes used by HDMI-Rx */
    phys = <&vphy_lane0 0 1 1 0>, <&vphy_lane1 0 1 1 0>, <&vphy_lane2 0 1 1 0>;
    /* HDMI Rx port connection information for creating media graph for
HDMI-Rx capture pipeline */
    hdmirx_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        hdmirx_port: port@0 {
            /* Fill the fields xlnx,video-format and xlnx,video-width based
on user requirement */
            reg = <0>;
            xlnx,video-format = <0>;
            xlnx,video-width = <10>;
            hdmi_rx_out: endpoint {
                remote-endpoint = <&vpss_scaler_in>; /* HDMI-Rx is connected
to input of the VPSS scalar */
            }
        };
    }
};
```

Scalar DT Node

On a Video Processing Sub System (VPSS) scalar node, you must update the information of the compatible string for selecting the v4l-based scalar driver, along with the reset-gpio and port information, for creating a media pipeline.

```
&hdmi_input_v_proc_ss_0 {
    reset-gpios = <&gpio 98 1>;
    compatible = "xlnx,v-vpss-scaler-2.2";
    vpss_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        vpss_port0: port@0 {
            /* For xlnx,video-format user needs to fill as per their
requirement */
            reg = <0>;
            xlnx,video-format = <3>;
            xlnx,video-width = <8>;
            vpss_scaler_in: endpoint {
                remote-endpoint = <&hdmi_rx_out>;
            }
        };
        vpss_port1: port@1 {
            /* For xlnx,video-format user needs to fill as per their
requirement */
            reg = <1>;
        }
    }
};
```
Framebuffer Write

On a Framebuffer Write DT node, you must update the reset-gpio, and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

Create HDMI-Rx Capture Video Pipeline

Create the v4l2 video capture pipeline by adding DMA and port information to the HDMI Capture DT node. The video pipeline is created based on this node.

Audio Formatter

On an Audio Formatter DT node, you must update the clocks, and input/output audio devices information.
HDMI-Tx Pipeline

To create an HDMI-Tx media pipeline, update the following pipeline DT nodes.

**HDMI-Tx DT Node**

On a HDMI-Tx subsystem DT node, you must update the clocks, Video PHY lanes, and ports information.

```c
&hdmi_output_v_hdmi_tx_ss_0 {
    clock-names = "s_axi_cpu_aclk", "s_axis_video_aclk", "txref-clk", "retimer-clk";
    clocks = <&vid_s_axi_clk>, <&vid_stream_clk>, <&si5319_0>, <&dp159>;/* Provide optional clocks information */
    phy-names = "hdmi-phy0", "hdmi-phy1", "hdmi-phy2";/* Provide Video phy interface lanes used by HDMI-Tx */
    phys = <&vphy_lane0 0 1 1 1>, <&vphy_lane1 0 1 1 1>, <&vphy_lane2 0 1 1 1>;
    hdmixtx_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        encoder_hdmi_port: port@0 {
            reg = <0>;
            hdmixtx_encoder: endpoint [
                remote-endpoint = <&mixer_crtc>:
            ];
        };
    };
};
```
Video Mixer DT Node

On a Video Mixer DT node, you must update the clocks, reset-gpio, ports, and overlay layers information for creating a media pipeline.

```c
&hdmi_output_v_mix_0 {
  reset-gpios = <&gpio 98 1>;
  /delete-property/ clock-names;
  clocks = <&si5319 0>;
  xlnx,dma-addr-width = <64>;

  crtc_mixer_port: port@0 {
    reg = <0>;
    mixer_crtc: endpoint {
      remote-endpoint = <&hdmi_encoder>;
    };
  };

  xx_mix_master: layer_0 {
    xlnx.layer-id = <0>;
    xlnx.layer-max-height = <2160>;
    xlnx.layer-max-width = <3840>;
    xlnx.layer-primary;
    xlnx,vformat = 'BG24';
  };

  xx_mix_overlay_1: layer_1 {
    xlnx.layer-id = <1>;
    xlnx.layer-max-width = <1920>;
    xlnx,vformat = 'NV12';
  };

  xx_mix_overlay_2: layer_2 {
    xlnx.layer-id = <2>;
    xlnx.layer-max-width = <1920>;
    xlnx,vformat = 'NV12';
  };

  xx_mix_overlay_3: layer_3 {
    xlnx.layer-id = <3>;
    xlnx.layer-max-width = <1920>;
    xlnx,vformat = 'NV12';
  };

  xx_mix_overlay_4: layer_4 {
    xlnx.layer-id = <4>;
    xlnx.layer-max-width = <1920>;
    xlnx,vformat = 'NV12';
  };

  xx_mix_overlay_5: layer_5 {
    xlnx.layer-id = <5>;
    xlnx.layer-max-width = <1920>;
    xlnx,vformat = 'NV12';
  };

  xx_mix_overlay_6: layer_6 {
    xlnx.layer-id = <6>;
    xlnx.layer-max-width = <1920>;
    xlnx,vformat = 'NV12';
  };

  xx_mix_overlay_7: layer_7 {
    xlnx.layer-id = <7>;
    xlnx.layer-max-width = <1920>;
    xlnx,vformat = 'NV12';
  };

  xx_mix_overlay_8: layer_8 {
    xlnx.layer-id = <8>;
  }
}
```
Framebuffer Read

On a Framebuffer Read DT node, you must update reset-gpio information.

```
&hdmi_output_v_frbuf_rd_0 {
    reset-gpios = &gpio 79 1;
};
```

SDI-Rx Pipeline

To create a Serial Digital Interface (SDI)-Rx Capture media pipeline, update the following pipeline DT nodes.

**SDI-Rx DT Node**

On a SDI-Rx subsystem DT node, you must update the clocks, reset gt gpio, and ports information, for creating a media pipeline.

```
&sdì_rx_input_v_smpte_uhdsdi_rx_ss {
    clock-names = 'video_out_clk', 'sdi_rx_clk', 's_axi_aclk';
    clocks = &zynqmp_clk 72, &si570_2, &zynqmp_clk 71;
    reset_gt-gpios = &axi_gpio_0 0 0 1;
```
Framebuffer Write

On a Framebuffer Write DT node, you must update the reset-gpio, and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

Create SDI-Rx Capture Video Pipeline

Create the v4l2 video capture pipeline by adding both DMA and port information to the SDI Capture DT node. A video pipeline is created based on this node.
For creating the SDI-Tx media pipeline, you must update the following pipeline DT nodes.

**SDI Tx DT Node**

On a SDI-Tx subsystem node, you must update the reset-gpio and SDI ports information.

```plaintext
&sdixoutput_v_smpte_uhdsdixss {
  sdix_ports: ports {
    #address-cells = <1>;
    #size-cells = <0>;
    //      reset-gpios = <&axi_gpio_0 1 0 1>;
    encoder_sdi_port: port@0 {
      reg = <0>;
      sdi_encoder: endpoint {
        remote-endpoint = <&pl_disp_crtc>;
      };
    };
  };
}
```

**Framebuffer Read**

On a Framebuffer Read DT node, you must update the reset-gpio information.

```plaintext
&sdixoutput_v_frmbuf_rd_0 {
  reset-gpios = <&gpio_resets_axi_gpio_resets 0 0 1>:
}
```
Display CRTC DT Node

You can create the video playback pipeline by adding the information of the compatible string for selecting the display driver, along with the DMA and port information.

```c
v_pl_disp: drm-pl-disp-drv {
    /* Fill the field xlnx.vformat based on user requirement */
    compatible = "xlnx.pl-disp";
    dma-names = "dma0";
    dmas = <&sdi_tx_output_v_frmbuf_rd_0 0>;
    xlnx.vformat = 'YUYV';
    pl_display_port: port@0 {
        reg = <0>;
        pl_disp_crtc: endpoint {
            remote-endpoint = <&sdi_encoder>;
        };
    };
}
```
To create a Mobile Industry Processor Interface (MIPI) Camera Serial Interface (CSI)-Rx Capture media pipeline, update the following pipeline DT nodes.

### MIPI CSI-2 SS DT Node

On a MIPI CSI-2 subsystem DT node, you must update the information of the compatible string for selecting the MIPI CSI 2 capture driver along with the CSI ports, for creating a media pipeline.

```
&mipi_csi2_rx_mipi_csi2_rx_subsystem_0 {
    compatible = "xlnx,mipi-csi2-rx-subsystem-2.0";
    csiss_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        csiss_port0: port0 {
```
Demosaic DT Node

On a Demosaic DT node, you must update the reset-gpio, and demosaic ports information.

```c
&mipi_csi2_rx_v_demosaic_0 {
  reset-gpios = &gpio 85 1;
  xlnx.max-height = <2160>;
  xlnx.max-width = <3840>;
  demosaic_ports: ports {
    #address-cells = <1>;
    #size-cells = <0>;
    demosaic_port0: port@0 {
      /* For cfa-pattern=rggb user needs to fill as per BAYER format */
      reg = <0>;
      xlnx.cfa-pattern = "rggb";
      xlnx.video-width = <8>;
      demosaic_in: endpoint {
        remote-endpoint = &csiss_out;
      };
    }
    demosaic_port1: port@1 {
      /* For cfa-pattern=rggb user needs to fill as per BAYER format */
      reg = <1>;
      xlnx.cfa-pattern = "rggb";
      xlnx.video-width = <8>;
      demosaic_out: endpoint {
      }
    }
  }
}
```
Gamma DT Node

On a Gamma DT node, you must update the reset-gpio, and gamma ports information.

```c
&mipi_csi2_rx_v_gamma_lut_0 {
    reset-gpios = <&gpio 86 1>;
    gamma_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        gamma_port0: port@0 {
            reg = <0>;
            xlnx,video-width = <8>;
            gamma_in: endpoint {
                remote-endpoint = <&demosaic_out>;
            }
        }
        gamma_port1: port@1 {
            reg = <1>;
            xlnx,video-width = <8>;
            gamma_out: endpoint {
                remote-endpoint = <&csc_in>;
            }
        }
    }
};
```

VPSS CSC DT Node

On a Video Processing Sub System (VPSS) Color Space Converter (CSC) DT node, you must update the information of the compatible string for selecting the v4l-based scalar driver, along with the reset-gpio, and SCS port information, for creating the media pipeline.

```c
&mipi_csi2_rx_v_proc_ss_csc {
    compatible = "xlnx,v-vpss-csc";
    reset-gpios = <&gpio 84 1>;
    csc_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        csc_port0: port@0 {
            /* For xlnx,video-format user needs to fill as per their requirement */
            reg = <0>;
            xlnx,video-format = <3>;
            xlnx,video-width = <8>;
            csc_in: endpoint {
                remote-endpoint = <&gamma_out>;
            }
        }
        csc_port1: port@1 {
            /* For xlnx,video-format user needs to fill as per their requirement */
```
Scalor DT Node

On a vproc subsystem scalar node, you must update the information of the compatible string for selecting the v4l-based scalar driver, along with the reset-gpio and port information, for creating the media pipeline.

```c
&mipi_csi2_rx_v_proc_ss_scaler {
    compatible = "xlnx,v-vpss-scaler-2.2";
    reset-gpios = <&gpio 82 1>; /* Reset GPIO connected to scalar IP */
    scaler_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        scaler_port0: port@0 {
            /* For xlnx,video-format user needs to fill as per their requirement */
            reg = <0>;
            xlnx,video-format = <3>;
            xlnx,video-width = <8>;
            scaler_in: endpoint {
                remote-endpoint = <&csc_out>;
            };
        };
        scaler_port1: port@1 {
            /* connect scaler output node to capture csi node */
            /* For xlnx,video-format user needs to fill as per their requirement */
            reg = <1>;
            xlnx,video-format = <3>;
            xlnx,video-width = <8>;
            scaler_out: endpoint {
                remote-endpoint = <&vcap_csi_in>;
            };
        };
    };
};
```

Framebuffer Write

On a Framebuffer Write DT node, you must update reset-gpio, and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

```c
&mipi_csi2_rx_v_frmbuf_wr_0 {
    reset-gpios = <&gpio 80 1>;
    xlnx,dma-align = <32>;
};
```
Create MIPI-CSI2 Rx Capture Video Pipeline

Add DMA and port information, to create a CSI video capture pipeline.

```c
vcap_csi {
    compatible = "xlnx,video";
    dma-names = "port0";
    dmas = <&mipi_csi2_rx_v_frmbuf_wr_0 0>;
    vcap_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        vcap_port: port@0 {
            direction = 'input';
            reg = <0>;
            vcap_csi_in: endpoint {
                remote-endpoint = <&scaler_out>;
            };
        };
    };
};
```

TPG Pipeline

Figure 6: TPG Pipeline Block Diagram

To create a Test Pattern Generator (TPG) media pipeline, update the following pipeline DT nodes.

TPG DT Node

On a TPG node, you must update the reset-gpio, and ports information.

```c
&tpg_input_v_tpg_1{
    reset-gpios = <&gpio 96 1>;
    xlnx.vtc = <&tpg_input_v_tc_1>;
    tpg_ports0: ports {
        #address-cells = <1>;
        #size-cells = <0>;
    };
};
```
tpg_port0: port@0 {
    /* Fill the field xlnx,video-format based on user requirement */
    reg = <0>;
    xlnx,video-format = <0x3>;
    xlnx,video-width = <8>;
    tpg_out0: endpoint {
        remote-endpoint = <&vcap_dev_in0>;
    };
};
}

**Framebuffer Write**

On a Framebuffer Write DT node, you must update the reset-gpio, and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

&tpg_input_v_frmbuf_wr_0 {
    reset-gpios = <&gpio 94 1>;
    xlnx,dma-align = <32>;
};

**Create TPG Pipeline**

Add the DMA and port information to create a TPG pipeline.

vcap_tp0 {
    compatible = "xlnx,video";
    dma-names = "port0";
    dmas = <&tpg_input_v_frmbuf_wr_0 0>;
    v_ports0: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        v_port0: port@0 {
            direction = 'input';
            reg = <0>;
            vcap_dev_in0: endpoint {
                remote-endpoint = <&tpg_out0>;
            };
        };
    };
};
To create a media pipeline for an SDI Audio capture pipeline, update the following pipeline DT nodes.

SDI Rx and Audio DT Node

On a SDI-Rx with Audio Capture DT node, update the clocks, reset gt gpions, and ports information.

```c
&sdid_output_v_smpyte_uhdssdi_rx_ss {
    clock-names = 'video_out_clk', 'sdid_rx_clk', "s_axi_aclk";
    clocks = <&zynqmp_clk 72>, <&si570_2>, <&zynqmp_clk 71>;
    reset_gt-gpios = <&axi_gpio_0 0 0 1>;
    sdirx_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        sdirx_port: port@0 {
            /* Fill the fields xlnx,video-format and xlnx,video-width based on user requirement */
            reg = <0>;
            xlnx,video-format = <0>;
            xlnx,video-width = <10>;
            sdi.rx.out: endpoint {
                remote-endpoint = <&vcap_sdirx_in>;
            };
        };
    };
};
```
Framebuffer Write

On a Framebuffer Write DT node, you must update the reset-gpios, and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

```c
&sd़i_rx_input_v_frbmbuf_wr_0 {
  reset-gpios = <&gpio_resets_axi_gpio_resets 1 0 1>;
  xlnx,dma-align = <32>;
};
```

Create SDI Video Capture Pipeline

Add the DMA and port information to create the v4l2 video capture pipeline.

```c
cap_sdirx {
  compatible = "xlnx,video";
  dma-names = 'port0';
  dmas = <&sd़i_rx_input_v_frbmbuf_wr_0 0>;
  vcap_ports: ports {
    #address-cells = <1>;
    #size-cells = <0>;
    vcap_port: port@0 {
      direction = "input";
      reg = <0>;
      vcap_sdirx_in: endpoint {
        remote-endpoint = <&sd़i_rx_out>;
      };
    };
  };
};
```

SDI Audio Pipeline

On an Audio Formatter node, you must update the input/output audio devices information.

```c
&audio_ss_audio_formatter_0 {
  xlnx,rx = <&audio_ss_v_uhdsdi_audio_1_extract>;
  xlnx,tx = <&audio_ss_v_uhdsdi_audio_0_embed>;
};
```

Audio Embed and Audio Extract IP nodes are required to embed/extract audio in SDI Tx and Rx protocol respectively. SDI embed contains an output port to remote endpoint of SDI video Tx node.
Audio Embed DT Node

&audio_ss_v_uhdsdi_audio_0_embed {
    xlnx,snd-pcm = &audio_ss_audio_formatter_0;
    sdi_av_port: port@0 {
        reg = <0>;
        sditx_audio_embed_src: endpoint {
            remote-endpoint = &sdi_audio_sink_port;
        };
    };
};

Audio Extract

&audio_ss_v_uhdsdi_audio_1_extract {
    xlnx,sdi-rx-video = &sdi_rx_input_v_smpte_uhdsdi_rx_ss;
    xlnx,snd-pcm = &audio_ss_audio_formatter_0;
};

Streaming-Based SCD

Scene Change Detection (SCD) is useful in various capture pipelines because a change of scene is a key transition of video, that if identified, could be useful to other processing elements, such as video encoders.

Using SCD, you might obtain improved video quality through processing elements such as video encoders.

The Video SCD IP is a configurable IP core that can read only one video stream in Stream mode. In Stream mode, input is read from the AXI4-Stream interface and the output stream is same as the received input stream.

Use the Video SCD core for the following applications:

- Video surveillance
- Video streaming
- Video conferencing
To create an HDMI streaming-based SCD media pipeline, update the following pipeline DT nodes.

**HDMI Rx Stream-based SCD Pipeline**

On a HDMI-Rx stream-based SCD node, you must update Video PHY lanes, audio formatter, and ports information.

```c
&hdmi_input_v_hdmi_rx_ss_0 {
    phy-names = "hdmi-phy0", "hdmi-phy1", "hdmi-phy2";
    phys = <&vphy_lane0 0 1 1 0>, <&vphy_lane1 0 1 1 0>, <&vphy_lane2 0 1 1 0>;
    xlnx.audio-enabled ;
    xlnx.edid-ram-size = <0x100>;
    xlnx.input-pixels-per-clock = <2>;
    xlnx.max-bits-per-component = <8>;
    xlnx snd-pcm = <&audio_ss_0_audio_formatter_1>;
    hdmirx_ports: ports {
        #address-cells = <1>;
    }
}```
VPROC SS DT Node

On a VPSS scaler node, to create the media pipeline, you must perform the following steps:

1. Update the information of the compatible string for selecting the v4l-based scalar driver.
2. Update the reset-gpio, and port information.

```c
&hdmi_input_v_proc_ss_0 {
    reset-gpios = <&gpio 98 1>; /* Reset GPIO connected to VPSS scalar IP */
    compatible = "xlnx,v-vpss-scaler-2.2"; /* V4L2 scalar selection using the compatible string */
    vpss_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        vpss_port0: port@0 { /* Connect the scalar input to HDMI-Rx output node */
            /* For xlnx,video-format user needs to fill as per their requirement */
            reg = <0>;
            xlnx,video-format = <3>;
            xlnx,video-width = <8>;
            vpss_scaler_in: endpoint {
                remote-endpoint = <&hdmi_rx_out>;
            };
        };
        vpss_port1: port@1 { /* connect VPSS output node to capture video node */
            /* For xlnx,video-format user needs to fill as per their requirement */
            reg = <1>;
            xlnx,video-format = <3>;
            xlnx,video-width = <8>;
            vpss_scaler_out: endpoint {
                remote-endpoint = <&scd_in>;
            };
        };
    };
};
```
SCD DT Node

On a SCD DT node, to create a media pipeline, you must update the reset gpio, and SCD ports information.

```c
&hdmi_input_v_scenechange_0 {
    reset-gpios = <&gpio 88 1>;
    scd_ports: scd {
        #address-cells = <1>;
        #size-cells = <0>;
        scd_port0: port@0 {
            reg = <0>;
            scd_in: endpoint {
                remote-endpoint = <&vpss_scaler_out>;
            };
        };
        scd_port1: port@1 {
            reg = <1>;
            scd_out: endpoint {
                remote-endpoint = <&scd_hdm_i_in>;
            };
        };
    };
}
```

Audio Formatter

On an Audio Formatter DT node, you must update the clocks, and input/output audio devices information.

```c
&audio_ss_0_audio_formatter_1 {
    clocks = <&zynqmp_clk 71>, <&si570_1>, <&si570_1>, <&zynqmp_clk 71>;
    xlnx,rx = <&hdmi_input_v_hdm_i_rx_ss_0>;
    xlnx,tx = <&hdmi_output_v_hdm_i_tx_ss_0>;
};
```

Framebuffer Write

On a Framebuffer Write DT node, you must update reset-gpio, and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

```c
&hdmi_input_v_frmbuf_wr_0 {
    reset-gpios = <&gpio 78 1>;
    xlnx,dma-align = <32>;
};
```
Create HDMI Stream-based SCD Video Pipeline

Add DMA and SCD port information to a HDMI capture DT node to create a v4l2 video capture pipeline with stream-based SCD video pipeline.

```diff
vcap_hdmi {
    compatible = "xlnx,video";
    dma-names = "port0";
    dmas = <&hdmi_input_v_frmbuf_wr_0 0>;
    scd_hdmi_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        scd_hdmi_port: port@0 {
            direction = 'input';
            reg = <0>;
            scd_hdmi_in: endpoint {
                remote-endpoint = <&scd_out>;
            };
        };
    };
};
```

Memory-Based SCD

The Video Scene Change Detection (SCD) IP is a configurable IP core that can read up to eight video streams in Memory mode. In Memory mode, input is read from the memory mapped AXI4 interface.

For memory-based SCD, there is nothing to update manually. The SCD node is in pl.dtsi, which is an autogenerated dtsi using PetaLinux.

You may need to add reset-gpio information in Scene Change node.

```diff
&v_scenechange_0 {
    reset-gpios = <&gpio 168 1>;
};
```

To implement memory-based SCD using static pipelines, add the xilinxscd element between source and encoder, as follows:

```diff
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! video/x-raw,
    format=NV12, width=3840, height=2160, framerate=60/1 !
xilinxscd io-mode=5 ! omxh265enc ...
```
Chapter 5

VCU Codec Features

The following sections list the features of the VCU codec.

VCU Encoder Support

The following table shows the list of supported multi-standard encoding in VCU.

*Table 1: Encoder Features*

<table>
<thead>
<tr>
<th>Video Coding Parameter</th>
<th>H.265 (HEVC)</th>
<th>H.264 (AVC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profiles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Main</td>
<td>• Baseline</td>
<td></td>
</tr>
<tr>
<td>• Main Intra</td>
<td>• Main</td>
<td></td>
</tr>
<tr>
<td>• Main10</td>
<td>• High</td>
<td></td>
</tr>
<tr>
<td>• Main10 Intra</td>
<td>• High10</td>
<td></td>
</tr>
<tr>
<td>• Main 4:2:2 10</td>
<td>• High 4:2:2</td>
<td></td>
</tr>
<tr>
<td>• Main 4:2:2 10 Intra</td>
<td>• High10 Intra</td>
<td></td>
</tr>
<tr>
<td>• High 4:2:2 Intra</td>
<td>• High 4:2:2 Intra</td>
<td></td>
</tr>
<tr>
<td>Levels</td>
<td>Up to 5.1 High Tier</td>
<td>Up to 5.2</td>
</tr>
<tr>
<td>Resolution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• 4096x2160 (DCI)</td>
<td>• 4096x2160 (DCI)</td>
<td></td>
</tr>
<tr>
<td>• 3840 x 2160</td>
<td>• 3840 x 2160</td>
<td></td>
</tr>
<tr>
<td>• 1920 x 1080</td>
<td>• 1920 x 1080</td>
<td></td>
</tr>
<tr>
<td>• 1420 x 576</td>
<td>• 1420 x 576</td>
<td></td>
</tr>
<tr>
<td>• 1280 x 720</td>
<td>• 1280 x 720</td>
<td></td>
</tr>
<tr>
<td>• 854 x 480</td>
<td>• 854 x 480</td>
<td></td>
</tr>
<tr>
<td>• 640 x 480</td>
<td>• 640 x 480</td>
<td></td>
</tr>
<tr>
<td>• 352 x 240</td>
<td>• 352 x 240</td>
<td></td>
</tr>
<tr>
<td>Frame Rate</td>
<td>Select 15, 30, 45 or 60</td>
<td>Select 15, 30, 45 or 60</td>
</tr>
<tr>
<td>Bit Depth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GStreamer</td>
<td>8-bit, 10-bit</td>
<td>8-bit, 10-bit</td>
</tr>
<tr>
<td>OMX</td>
<td>8-bit, 10-bit</td>
<td>8-bit, 10-bit</td>
</tr>
<tr>
<td>Chroma Format</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GStreamer</td>
<td>4:2:0, 4:2:2</td>
<td>4:2:0, 4:2:2</td>
</tr>
<tr>
<td>OMX</td>
<td>4:2:0, 4:2:2</td>
<td>4:2:0, 4:2:2</td>
</tr>
</tbody>
</table>
Rate Control Modes

The VCU supports the following rate control modes.

- Constant Quant Param (QP)
- Variable Bit Rate (VBR)
- Constant Bit Rate (CBR)
- Low-latency

The MCU firmware handles the rate control process. No signals (either in Software API or FPGA signals) are triggered during the rate control process.

VBR

When using VBR, the encoder buffer is allowed to underflow (be empty), and the maximum bitrate, which is the transmission bitrate used for the buffering model, can be higher than the target bitrate. So VBR relaxes the buffering constraints and allows to decrease the bitrate for simple content and can improve quality by allowing more bits on complex frames. VBR mode constrains the bitrate with a specified maximum while keeping it on the target bit rate where possible. Similar to CBR, it avoids buffer underflow by increasing the QP. However, the target bit rate can exceed up to the maximum bit rate. Therefore, the QP has to be increased by a smaller factor. A buffer overflow results in an unchanged QP and a lower bit rate.

CBR

The goal of CBR is to reach the target bitrate on average (at the level of one or a few GOPs) and to comply with the Hypothetical Reference Decoder (HRD) model, avoiding decoder buffer overflows and underflows. In CBR mode, a single bitrate value defines both the target stream bitrate and the output/transmission (leaky bucket) bitrate. The reference decoder buffer parameters are Coded Picture Buffer (CPBSizs) and Initial Delay. The CBR rate control mode tries to keep the bit rate constant whilst avoiding buffer overflows and underflows. If a buffer underflow happens, the QP is increased (up to MaxQP) to lower the size in bits of the next frames. If a buffer overflow occurs, the QP is decreased (down to MinQP) to increase the size in bits.

Low Latency

The frame is divided into multiple slices; the VCU encoder output, and decoder input are processed in slice mode. The VCU Encoder input, and Decoder output still work in frame mode.
Different Group of Picture Configuration

In video coding, a group of pictures (GoP) structure, specifies the order in which intra-and inter-frames are arranged. GoP Length is a length between two intra-frames. The GoP is a collection of successive pictures within a coded video stream. Each coded video stream consists of successive GoPs from which the visible frames are generated. The GoP range is from 1–1000. The GoP length must be a multiple of B-Frames+1.

- **DEFAULT_GOP**: IBBPBBP... (Display order)
- **LOW_DELAY_P**: GopPattern with a single I-picture at the beginning followed with P-pictures only. Each P-picture uses the picture just before as reference. IPPP....
- **LOW_DELAY_B**: GopPattern with a single I-picture at the beginning followed by B-pictures only. Each B-picture uses the picture just before it as first reference; the second reference depends on the Gop.Length parameter. IBBB...
- **PYRAMIDAL_GOP**: Advanced GOP pattern with hierarchical B-frame. The size of the hierarchy depends on the Gop.NumB parameter.
- **ADAPTIVE_GOP**: The encoder adapts the number of B-frames used in the GOP pattern based on heuristics on the video content.
- **DEFAULT_GOP_B**: IBBBBBB... (P frames replaced with B).
- **PYRAMIDAL_GOP_B**: Advanced GOP pattern with hierarchical B frame. Here, P frames are replaced with B.

Parameter List

The following table shows the list of VCU parameters and their description.

### Table 2: Parameter List

<table>
<thead>
<tr>
<th>VCU Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Target Bit Rate     | Target bitrate in kb/s
|                     | Default value: 64                                                          |
| GOP Length          | Distance between two consecutive Intra frames. Specify integer value between 0 and 1,000. Value 0 and 1 corresponds to Intra-only encoding
|                     | Default value: 30                                                          |
| Number of B-frames  | Number of B-frames between two consecutive P-frames. Used only when gop-mode is basic or pyramidal.
|                     | Range:
|                     |   - 0 - 4 (for gop-mode = basic)
|                     |   - 3, 5, or 7 (for gop-mode = pyramidal)
|                     | B-frames should be set to zero in low-latency and reduced-latency mode as there cannot be any frame reordering when B-frames are set.
|                     | Default value: 0                                                           |
### Table 2: Parameter List (cont’d)

<table>
<thead>
<tr>
<th>VCU Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inserting Key Frame (IDR)</td>
<td>Specifies the number of frames between consecutive instantaneous decoder refresh (IDR) pictures. Encoder inserts a Keyframe and restart a GOP upon IDR request.</td>
</tr>
<tr>
<td>Region of Interest Encoding</td>
<td>Region of Interest Encoding tags regions in a video frame to be encoded with user supplied quality (high, medium, low, and dont-care) relative to the picture background (untagged region). You provide the region of interest (ROI) location (top, left) in pixels and the width and height in pixels along with the quality index. Multiple and overlapped ROI regions within a frame are supported. The sample GStreamer application only adds one ROI region but users can attach multiple ROI meta data properties to the buffer.</td>
</tr>
<tr>
<td>Long Term Reference Picture</td>
<td>If enabled, encoder accepts dynamically inserting and using long-term reference picture events from upstream elements. Boolean: True or False Default value: False</td>
</tr>
<tr>
<td>Adaptive GOP</td>
<td>You specify the maximum number of B frames that can be used in a GOP. Set the GOP mode to adaptive using GopCtrlMode=ADAPTIVE_GOP in encoder configuration file at control software, and gop-mode=adaptive in GStreamer. The encoder adapts the number of B frames used in the GOP pattern based on heuristics on the video content. The encoder does not go higher than the maximum number of B frames that you specify.</td>
</tr>
<tr>
<td>SEI Insertion and Extraction</td>
<td>Adds SEI NAL to the stream. You are responsible for the SEI payload and have to write it as specified in Annex D.3 of ITU-T. The encoder does the rest including anti-emulation of the payload.</td>
</tr>
<tr>
<td>Dual Pass Encoding</td>
<td>Encode the video twice, the first pass collects information about the sequence. Statistics are used to improve the encoding of the second pass.</td>
</tr>
<tr>
<td>Scene Change Detection GDR Intra Refresh</td>
<td>The Xilinx® video scene change detection IP provides a video processing block that implements a scene change detection algorithm. The IP core calculates the histogram on a vertically subsampled HMA frame for consecutive frames. The histogram of these frames is then compared using the sum of absolute differences (SAD). This IP core is programmable through a comprehensive register interface to control the frame size, video format, and subsampling value. Gradual Decoder Refresh (GDR): When GOPCtrlMode is set to LOW_DELAY_P, the GDRMode parameter is used to specify whether GDR scheme should be used or not. When GDR is enabled (horizontal/vertical), the encoder inserts intra MBs row/column in the picture to refresh the decoder. The Gop.FreqIDR parameter specifies the frequency at which the refresh pattern should happen. To allow full picture refreshing, the Gop.FreqIDR parameter should be greater than the number of CTB/MB rows (GDR_HORIZONTAL) or columns (GDR_VERTICAL).</td>
</tr>
</tbody>
</table>
| Dynamic Resolution Change – VCU Encoder/Decoder | Dynamic Resolution Change is supported at the VCU for both the Encoder/Decoder.  
  • VCU Encoder: Input sources may contain several resolution.  
  • VCU Decoder: An input compressed stream can contain multiple resolutions. The VCU Decoder can decode pictures without re-creating a channel. All the streams should belong to same codec, chroma-format, and bit-depth. |
| Frameskip support for VCU Encoder     | When an encoded picture is too large and exceeds the CPB buffer size, the picture is discarded and replaced by a picture with all MB/CTB encoded as « Skip ». This feature is useful especially at low bitrates when the encoder must maintain a strict target-bitrate irrespective of video-complexity. Use this parameter only if control-rate=constant/variable and b-frames are less than 2. Default: FALSE |
Table 2: Parameter List (cont’d)

<table>
<thead>
<tr>
<th>VCU Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-Streams Support</td>
<td>The VCU supports simultaneous encoding and decoding up to 4K UHD resolution at 60 Hz. This can be divided into 32 smaller streams of up to 480p at 30 Hz. Several combinations of one to 32 streams can be supported with different resolutions, provided the cumulative throughput does not exceed 4K UHD at 60 Hz.</td>
</tr>
<tr>
<td>DCI-4k Encode/Decode</td>
<td>VCU is capable of encoding or decoding at 4096x2160p60, 422, provided the VCU Core-clk frequency is set to 712 MHz, keeping rest of the AXI and MCU frequency as recommend in the harware section.</td>
</tr>
</tbody>
</table>
| External QP Table | Along with each frame, it is possible to associate a QP Table specifying the QP to use for each encoding block of the frame. The QP value can be relative or absolute. The QP table buffer must contain a byte per MB/CTB, in raster scan format:  
  - In AVC, one byte per 16x16 MB  
  - In HEVC, one byte per 32x32 CTB |
| Temporal-ID support for VCU Encoder | The VCU encoder assigns a temporal layer ID to each frame based on its hierarchical layer as per the AVC/HEVC standard. This enables having a temporal-ID based QP and the Lambda table control for encode session. This is for Pyramidal GOP only. |
| Low latency support | Low-latency rate control (hardware RC) is the preferred control-rate for video streaming; it tries to maintain an equal amount of frame sizes for all pictures. |
| Dynamic-Bitrate | Dynamic-bitrate is the ability to change encoding bitrate (target-bitrate) while the encoder is active. |
| Decoder Meta-data Transfer Using 1-to-1 Relation between Input and Output Buffer | Each incoming buffer is copied into the decoder internal circular buffer, and the frame boundaries are (re-)detected afterwards by the decoder itself. This prevents it from keeping a true 1-to-1 relationship between the input buffer and decoded output frame. An application can try to retrieve the 1-to-1 relationship based on the decoding order but this is not reliable in case of error concealment. This new feature consists of bypassing the circular buffer stage for use cases where the incoming buffers are frame (or slice) aligned. In this case, the decoder can directly work on the input buffer (assuming DMA input buffer) and any associated metadata can be retrieved on the output callback. |
| DEFAULT_GOP_B and PYRAMIDAL_GOP_B GOP Control Modes | Patterns are identical to DEFAULT_GOP and PYRAMIDAL_GOP except that P frames are replaced with B Pictures. Omxh264enc/omxh265enc element gop-mode parameter supports these two new settings from 2019.2 onwards. “basic-b” corresponds to DEFAULT_GOP_B and “pyramidal-b” corresponds to PYRAMIDAL_GOP_B. |
| Adaptive Deblocking Filter Parameters Update | Loop filter beta and Tc offsets are configurable at frame level.  
Constraints: New offset values are applied on the chosen frame and on the following frames in the decoding order.  
Omxh264enc/omxh265enc elements support the following two mutable parameters. The values can be modified during run time.  
  - loop-filter-beta-offset: Beta offset for the deblocking filter; used only when loop-filter-mode is enabled.  
  - loop-filter-alpha-c0-offset / loop-filter-tc-offset: Alpha_C0 / TC offset for the deblocking filter; used only when loop-filter-mode is enabled. |
Chapter 6

GStreamer Multimedia Framework

GStreamer is a library for constructing graphs of media-handling components. The applications it supports range from simple playback, audio and video streaming, to complex audio (mixing) and video (non-linear editing) processing. Xilinx® has developed omx-based elements to enable users to easily create flexible audio and video processing pipelines in an open-source framework.

Install and Set Up Gstreamer

To install Gstreamer-1.0 using PetaLinux, build the Linux image and boot image using the PetaLinux build tool.

For the PetaLinux installation steps, see PetaLinux Tools Documentation: Reference Guide (UG1144).

Check Gstreamer Version

Check the Gstreamer-1.0 version with the following command:

```
gst-inspect-1.0 --version
```

The following table shows that list ofgst-omx video decoders, included in Gstreamer-1.0:

*Table 3: gst-omx Video Decoders*

<table>
<thead>
<tr>
<th>Decoder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omxh265dec</td>
<td>OpenMAX IL H.265 Video Decoder</td>
</tr>
<tr>
<td>omxh264dec</td>
<td>OpenMAX IL H.264 Video Decoder</td>
</tr>
</tbody>
</table>

The following table shows that list ofgst-omx video encoders, included in Gstreamer-1.0:

*Table 4: gst-omx Video Encoders*

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omxh265enc</td>
<td>OpenMAX IL H.265/HEVC Video Encoder</td>
</tr>
<tr>
<td>Omxh264enc</td>
<td>OpenMAX IL H.264/AVC Video Encoder</td>
</tr>
</tbody>
</table>
Pixel Format Support

VCU supports the following pixel formats. These formats signify the memory layout of pixels. Each format applies at the encoder input and the decoder output side.

If a format is not supported between two elements, the cap (capability) negotiation fails and Gstreamer returns an error. In that case, use a video conversion element to perform format conversion from one format to another.

Table 5: Supported Pixel Formats

<table>
<thead>
<tr>
<th>Input for VCU decoder (encoded data)</th>
<th>VCU decoded format</th>
<th>V4L2</th>
<th>Gstreamer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVC or HEVC 420, 8-bit</td>
<td>NV12</td>
<td>V4L2_PIX_FMT_NV12</td>
<td>GST_VIDEO_FORMAT_GRAY8_NV12</td>
</tr>
<tr>
<td>AVC or HEVC 422, 8-bit</td>
<td>NV16</td>
<td>V4L2_PIX_FMT_NV16</td>
<td>GST_VIDEO_FORMAT_NV16</td>
</tr>
<tr>
<td>AVC or HEVC 420, 10-bit</td>
<td>NV12_10LE32</td>
<td>V4L2_PIX_FMT_XV15</td>
<td>GST_VIDEO_FORMAT_NV12_10LE32</td>
</tr>
<tr>
<td>AVC or HEVC 422, 10-bit</td>
<td>NV16_10LE32</td>
<td>V4L2_PIX_FMT_XV20</td>
<td>GST_VIDEO_FORMAT_NV16_10LE32</td>
</tr>
</tbody>
</table>

Note: Native output of VCU Decoder is always in semi-planar format.

For more details, see H.264/H.265 Video Codec Unit LogiCORE IP Product Guide (PG252).

GStreamer Plugins

GStreamer is a library for constructing graphs of media-handling components. The applications it supports range from simple playback and audio/video streaming to complex audio (mixing) and video processing.

GStreamer uses a plug-in architecture which makes the most of GStreamer functionality implemented as shared libraries. The GStreamer base functionality contains functions for registering and loading plug-ins and for providing the fundamentals of all classes in the form of base classes. Plug-in libraries get dynamically loaded to support a wide spectrum of codecs, container formats, and input/output drivers.

The following table describes the plug-ins used in the GStreamer interface library.

Table 6: Verified GStreamer Plug-ins

<table>
<thead>
<tr>
<th>Plug-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v4l2src</td>
<td>Use v4l2src to capture video from V4L2 devices like Xilinx HDMI-RX and TPG.</td>
</tr>
<tr>
<td>Plug-in</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>kmssink</td>
<td>The kmssink is a simple video sink that renders raw video frames directly in a plane of a DRM device. Example pipeline:</td>
</tr>
<tr>
<td></td>
<td>\texttt{gst-launch-1.0 v4l2src ! &quot;video/x-raw, format=NV12, width=3840, height=2160&quot; ! kmssink}</td>
</tr>
<tr>
<td>h26xparse</td>
<td>Parses a H.26x encoded stream. Example pipeline:</td>
</tr>
<tr>
<td></td>
<td>\texttt{gst-launch-1.0 filesrc location=/media/card/abc.mp4 ! qtdemux ! h26xparse ! omxh26xdec ! kmssink}</td>
</tr>
<tr>
<td>omxh26xdec</td>
<td>The omxh26xdec is a hardware-accelerated video decoder that decodes encoded video frames. Example pipeline:</td>
</tr>
<tr>
<td></td>
<td>\texttt{gst-launch-1.0 filesrc location=/media/card/abc.mp4 ! qtdemux ! h26xparse ! omxh26xdec ! kmssink}</td>
</tr>
<tr>
<td></td>
<td>This pipeline shows a .mp4 multiplexed file where the encoded format is h26x encoded video. \textbf{Note:} Use the omxh264dec for H264 decoding, and the omxh265dec for H265 decoding. h264parse parses a H.264 encoded stream. h265parse parses a H.265 encoded stream.</td>
</tr>
<tr>
<td>omxh26xenc</td>
<td>The omxh26xenc is a hardware-accelerated video encoder that encodes raw video frames. Example pipeline:</td>
</tr>
<tr>
<td></td>
<td>\texttt{gst-launch-1.0 v4l2src ! omxh26xenc ! filesink location=out.h26x}</td>
</tr>
<tr>
<td></td>
<td>This pipeline shows the video captured from a V4L2 device that delivers raw data. The data is encoded to the h26x encoded video type, and stored to a file. \textbf{Note:} Use the omxh264enc for H264 encoding, and the omxh265enc for H265 encoding.</td>
</tr>
<tr>
<td>alsasrc</td>
<td>Use the alsasrc plug-in to capture audio from audio devices such as Xilinx HDMI-RX. Example pipeline:</td>
</tr>
<tr>
<td></td>
<td>\texttt{gst-launch-1.0 alsasrc device=hw:1,1 ! queue ! audioconvert ! audioresample ! audio/x-raw, rate=48000, channels=2, format=S24_32LE ! alsasink device=&quot;hw:1,0&quot;}</td>
</tr>
<tr>
<td></td>
<td>This pipeline shows that the audio captured from an ALSA source, plays on an ALSA sink.</td>
</tr>
<tr>
<td>alsasink</td>
<td>The alsasink is a simple audio sink that plays raw audio frames. Example pipeline:</td>
</tr>
<tr>
<td></td>
<td>\texttt{gst-launch-1.0 alsasrc device=hw:1,1 ! queue ! audioconvert ! audioresample ! audio/x-raw, rate=48000, channels=2, format=S24_32LE ! alsasink device='hw:1,0'}</td>
</tr>
<tr>
<td></td>
<td>This pipeline shows that the audio captured from the ALSA source, plays on an ALSA sink.</td>
</tr>
</tbody>
</table>
Table 6: Verified GStreamer Plug-ins (cont’d)

<table>
<thead>
<tr>
<th>Plug-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>faad</td>
<td>Decoder faad is an audio decoder that decodes encoded audio frames. Example pipeline:</td>
</tr>
<tr>
<td></td>
<td><code>gst-launch-1.0 filesrc location=out.ts ! tsdemux ! aacparse ! faad ! audioconvert ! audioresample ! audio/x-raw,rate=48000,channels=2,format=S24_32LE ! alsasink device=&quot;hw:1,0&quot;</code></td>
</tr>
<tr>
<td></td>
<td>This pipeline shows a .ts multiplexed file where the encoded format is aac encoded audio. The data is decoded and played on an ALSA sink device.</td>
</tr>
<tr>
<td>faac</td>
<td>The faac is an audio encoder that encodes raw audio frames. Example pipeline:</td>
</tr>
<tr>
<td></td>
<td><code>gst-launch-1.0 alsasrc device=hw:1,1 num-buffers=500 ! audio/x-raw, format=S24_32LE, rate=48000, channels=2 ! queue ! audioconvert ! audioresample ! faac ! aacparse ! mpegtsmux ! filesink location=out.ts</code></td>
</tr>
<tr>
<td></td>
<td>This pipeline shows the audio captured from an ALSA device that delivers raw data. The data is encoded to aac format and stored to a file.</td>
</tr>
<tr>
<td>xilinxscd</td>
<td>The xilinxscd is hardware-accelerated IP that enables detection of scene change in a video stream. This plugin generates upstream events whenever there is scene change in an incoming video stream so the encoder can insert an Intra frame to improve video quality. Example pipeline:</td>
</tr>
<tr>
<td></td>
<td><code>gst-launch-1.0 -v v4l2src ! video/x-raw, width=3840, height=2160, format=NV12, framerate=60/1 ! xilinxscdio-mode=5 ! omxh26xenc ! filesink location=/run/out.h26x</code></td>
</tr>
<tr>
<td></td>
<td>This pipeline shows the video captured from a V4L2 device that delivers raw data. This raw data is passed through the xilinxscd plugin which analyzes the stream in runtime and provides an event to the encoder that determines whether or not a scene change is detected in a video stream. The encoder uses this information to insert an I-frame in an encoded bit-stream.</td>
</tr>
<tr>
<td>Note: Use the omxh264enc for H264 encoding, and the omxh265enc for H265 encoding.</td>
<td></td>
</tr>
<tr>
<td>appsrc</td>
<td>The appsrc element can be used by applications to insert data into a GStreamer pipeline. Unlike most GStreamer elements, appsrc provides external API functions.</td>
</tr>
<tr>
<td>appsink</td>
<td>The appsink is a sink plugin that supports many different methods, enabling the application to manage the GStreamer data in a pipeline. Unlike most GStreamer elements, appsink provides external API functions.</td>
</tr>
<tr>
<td>queue</td>
<td>Queues data until one of the limits specified by the <code>max-size-buffers</code>, <code>max-size-bytes</code>, or <code>max-size-time</code> properties has been reached</td>
</tr>
</tbody>
</table>

For more details, see H.264/H.265 Video Codec Unit LogiCORE IP Product Guide (PG252).

Raw Use Case

For RAW video stream only:
Use the following pipeline to play a raw video stream captured from an input source device:

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! video/x-raw, width=3840, height=2160, format=NV12, framerate=60/1 ! queue ! kmssink bus-id='a0070000.v_mix'
```

In the preceding example, the live source device link is present under the `/dev` directory. Video stream resolution is 4kp and 60fps. Video stream color format is NV12.

For RAW video and audio stream:

Use the following pipeline to play raw video and audio captured from the input source device:

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! video/x-raw, width=3840, height=2160, format=NV12, framerate=60/1 ! queue max-size-bytes=0 ! kmssink bus-id='a0070000.v_mix' alsasrc device=hw:2,1 ! audio/x-raw, rate=48000, channels=2, format=S24_32LE ! audioconvert ! audiore只想 sample ! audio/x-raw, rate=48000, channels=2, format=S24_32LE ! queue ! alsasink device="hw:2,0"
```

The preceding example shows that video and audio can be captured using a single gstreamer pipeline. Audio capture device is hw:2,1 and playback device is hw:2,0. For video stream, the pipeline remains the same as the previous example.

---

## Decode/Encode Example

### Video Decode Using Gstreamer-1.0

The following example shows how to decode an input file with H.265 (HEVC) video format using Gstreamer-1.0.

The file is present in the SD Card, the format is mp4, and the encoded video format is H265.

```
gst-launch-1.0 filesrc location="/media/card/input-file.mp4" ! qtdemux name=demux demux.video_0 ! h265parse ! video/x-h265 ! omxh265dec ! queue max-size-bytes=0 ! fakevideosink
```

**Note:** To decode a file in H264(AVC) video format, replace the `h265` elements with `h264`.

### Video Encode Using Gstreamer-1.0

The following example shows how to encode a captured stream of input source device with H265 (HEVC) video format using Gstreamer-1.0.
The input stream is a live source (for example, HDMI-Rx or MIPI camera) and is present under the /dev directory. The encoded video format is H265, color format is NV12, and resolution is 4kp at 60fps.

```bash
gst-launch-1.0 v4l2src io-mode=4 device=/dev/video0 ! \
video/x-raw, width=3840, height=2160, framerate=60/1, format=NV12 ! \
omxh265enc ! video/x-h265 ! fakesink
```

**Note:** To encode a video stream in H264 (AVC) video format, replace the h265 elements with h264.

---

**Camera Interface and Support**

Gstreamer-1.0 supports the following camera interfaces.

**CSI Cameras**

- ZCU106 currently supports only 1 CSI Camera
- LI-IMX274MIPI-FMC image sensor daughter card
- CSI camera supports the following image resolutions:
  - 3840x2160
  - 1920x1080
  - 1280x720

---

**Video Pipelines**

A video pipeline consists of three elements:

1. A live-capture/file-src element receives frames either from an external source, or produces video frames internally. The captured video frames are written to memory.
2. A processing element reads video frames from memory, performs certain processing, and then writes the processed frames back to memory.
3. A display element reads video frames from memory and sends the frames to a sink. In cases where the sink is displayed, this pipeline is also referred to as display pipeline.

**File Playback**

Use the following static pipelines to perform local file playback using Gstreamer-1.0.
Static pipelines for file playback using Gstreamer-1.0

- To play a Transport Stream (TS) file with only the video stream:

```bash
gst-launch-1.0 filesrc location=/media/card/abc.ts ! tsdemux ! queue ! h265parse ! omxh265dec ! kmssink bus-id=a0070000.v_mix
```

In this example, the file is present in the SD card, the container format is TS, and the encoded video format is H265.

- To play a TS file with both the video and audio streams:

```bash
gst-launch-1.0 filesrc location=/media/card/abc.ts ! tsdemux name=demux ! queue ! h265parse ! omxh265dec ! queue max-size-bytes=0 ! kmssink bus-id=a0070000.v_mix sync=true demux. ! queue ! faad ! audioconvert ! audio/x-raw, rate=48000, channels=2, format=S24_32LE ! alsasink device=hw:2,0
```

In this example, the file is present in the SD Card, the container format is TS, and the encoded video format is H265. The encoded audio stream is AAC, with a sample rate of 48000 (48kHz) and audio format of S24_32LE. Audio playback device is hw:2,0.

Dynamic pipeline for file playback using Gstreamer-1.0

GStreamer also provides uridecodebin, a basic media-playback plugin that automatically takes care of most playback details. The following example shows how to play any file if the necessary demuxing and decoding plugins are installed.

- To play a TS file containing only the video stream:

```bash
gst-launch-1.0 uridecodebinuri=file:///media/card/abc.ts ! queue ! kmssink bus-id=a0070000.v_mix
```

This example shows that the file is present in the SD Card.

- To play a TS file containing both the video and the audio streams:

```bash
gst-launch-1.0 uridecodebinuri='file:///media/card/test.ts' name=decode ! queue max-size-bytes=0 ! kmssink bus-id='a0070000.v_mix' decode. ! audioconvert ! audioresample ! audio/x-raw, rate=48000, channels=2, format=S24_32LE ! kmssink bus-id=a0070000.v_mix sync=true decode. ! queue ! alsasink device=hw:2,0
```

Recording

Use the following pipelines to record video from an input source to the required file format.
• To record only the video stream:

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! \
video/x-raw,format=NV12,width=3840,height=2160,framerate=60/1 ! \
omxh265enc qp-mode=auto gop-mode=basic gop-length=60 b-frames=0 \ntarget-bitrate=60000 num-slices=8 control-rate=constant prefetch-buffe=true \nlow-bandwidth=false filler-data=true cpb-size=1000 initial-delay=500 ! \nqueue ! video/x-h265, profile=main, alignment=au ! mpegtsmux alignment=7 \name=mux ! \filesink location="/run/media/sda/test.ts"
```

This example shows that the live source device link is present under the /dev directory. Encoded video format is H265 and color format is NV12. Video stream resolution is 4k and 60fps. The record file test.ts is present in SATA drive in TS file format.

**Note:**

1. For input source with 1080p@60 resolution, replace width and height with 1920 and 1080 respectively. Frame rate can also be changed to 30.
2. To encode input stream into H264 video format, replace h265 in the above pipeline with h264.

• To record both the video and audio stream:

The following pipeline can be used to record video with audio from an input source to the required file format.

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! \
video/x-raw, format=NV12, width=3840, height=2160, framerate=60/1 ! 
omxh265enc qp-mode=auto gop-mode=basic gop-length=60 b-frames=0 \ntarget-bitrate=60000 num-slices=8 control-rate=constant prefetch-buffe=true \nlow-bandwidth=false filler-data=true cpb-size=1000 initial-delay=500 ! \nvideo/x-h265, profile=main, alignment=au ! queue ! mpegtsmux alignment=7 \name=mux ! \alsasrc device=hw:2,1 ! audio/x-raw, format=S24_32LE, rate=48000, channels=2 ! queue ! 
audioconvert ! audioremap ! faac ! aacparse ! mpegtsmux name=mux ! \filesink location="/run/media/sda/test.ts"
```

In this example, video and audio can be encoded using a single gstreamer pipeline to record into a single file. Encoded audio stream is AAC. Audio capture device is hw:2,1 and record file test.ts is present in SATA. For video encoding, the pipeline remains the same as the previous example.

**Streaming Out Using RTP Unicast Example**

• For only a video stream out:
Use the following pipeline to send a video stream of input source, from one device to another device on the same network.

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! \
  video/x-raw, format=NV12, width=3840, height=2160, framerate=60/1 ! \
  omxh265enc qp-mode=auto gop-mode=basic gop-length=60 b-frames=0 \ 
  target-bitrate=60000 num-slices=8 control-rate=constant prefetch- 
  buffer=true \ 
  low-bandwidth=false filler-data=true cpb-size=1000 initial-delay=500 \ 
  periodicity-idr=60 ! video/x-h265, profile=main, alignment=au ! \
  queue ! mpegtsmux alignment=7 name=mux ! rtpmp2tpay ! \
  udpsink host=192.168.25.89 port=5004
```

This example shows that video streamed out from one device (server) to another device (client) on the same network. Encoded video format is H265 and color format is NV12. Video stream resolution is 4kp with 60fps, and bitrate is 60 Mbps. Server sends the video stream to the client host device with IP Address 192.168.25.89 on port 5004.

**Note:** Replace host IP address as per IP configuration of the client device.

- For both video and audio stream out:

Use the following pipeline to send both video and audio stream of input source, from one device to another device on the same network.

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! \
  video/x-raw, format=NV12, width=3840, height=2160, framerate=60/1 ! \
  omxh265enc qp-mode=auto gop-mode=basic gop-length=60 b-frames=0 \ 
  target-bitrate=60000 num-slices=8 control-rate=constant prefetch- 
  buffer=true \ 
  low-bandwidth=false filler-data=true cpb-size=1000 initial-delay=500 \ 
  periodicity-idr=60 ! video/x-h265, profile=main, alignment=au ! \
  queue ! audioclip ! faac ! aacparse ! mpegtsmux alignment=7 name=mux ! rtpmp2tpay ! \
  udpsink host=192.168.25.89 port=5004
```

This example shows that video and audio can be streamed out using a single gstreamer pipeline. Encoded audio is in AAC. Audio and video is streamed out simultaneously. For video stream-out, the pipeline remains the same as in the previous example.

### Streaming In

- Static pipelines for stream-in using Gstreamer-1.0
  - For video stream in only:
The following pipeline can be used to receive the video stream from another device (server), to the host device on the same network.

```bash
gst-launch-1.0 udpsrc port=5004 buffer-size=60000000 caps='application/x-rtp, clock-rate=90000' ! rtpjitterbuffer latency=1000 ! rtpmp2tdepay ! tsparse ! video/mpegts ! tsdemux name=demux ! queue ! h265parse ! video/x-h265, profile=main, alignment=au ! omxh265dec internal-entropy-buffers=5 low-latency=0 ! queue max-size-bytes=0 ! kmssink bus-id="a0070000.v_mix"
```

In this example, the encoded video format is H265. Stream in at client device occurs on port 5004 over the UDP protocol.

- For video and audio stream in:

  Use the following pipeline to receive video and audio stream from another device (server), to the host device on the same network.

  ```bash
gst-launch-1.0 udpsrc port=5004 buffer-size=60000000 caps='application/x-rtp, clock-rate=90000' ! rtpjitterbuffer latency=1000 ! rtpmp2tdepay ! tsparse ! video/mpegts ! tsdemux name=demux ! queue ! h265parse ! video/x-h265, profile=main, alignment=au ! omxh265dec internal-entropy-buffers=5 low-latency=0 ! queue max-size-bytes=0 ! kmssink bus-id="a0070000.v_mix" demux. ! queue ! aacparse ! faad ! audioconvert ! audioresample ! audio/x-raw, rate=48000, channels=2, format=S24_32LE ! alsasink device="hw:2,0"
  ```

  In this example, the encoded video format is H265. Stream in at client device occurs on port 5004 over the UDP protocol. Audio playback device is hw:2,0. For video stream-in, the pipeline remains the same as in the previous example.

- Dynamic pipelines for stream-in using Gstreamer-1.0

  - For video stream-in:

    ```bash
gst-launch-1.0 uridecodebinuri=udp://192.168.25.89:5004 ! kmssink bus-id=a0070000.v_mix
  ```

  - For both video and audio stream-in:

    ```bash
gst-launch-1.0 uridecodebinuri=udp://192.168.25.89:5004 name=demuxdemux. ! queue ! kmssink bus-id=a0070000.v_mix demux. ! queue ! audioconvert ! audiostream ! audio/x-raw, rate=48000, channels=2, format=S24_32LE ! alsasink device='hw:2,0'
  ```
Low-Latency

The frame is divided into multiple slices; the VCU encoder output and decoder input are processed in slice mode. The VCU Encoder input and Decoder output still works in frame mode. The VCU encoder generates a slice done interrupt at every end of the slice and outputs stream buffer for slice, and is available immediately for next element processing. Therefore, with multiple slices it is possible to reduce VCU processing latency from one frame to one-frame/num-slices. In the low-latency mode, a maximum of four streams for the encoder and two streams for the decoder can be run.

The Low-Latency 4kp60 HEVC streaming pipeline is as follows:

- **Stream Out:**
  
  Use the following pipeline to stream-out (capture → encode → stream-out) NV12 video using a low-latency GStreamer pipeline. This pipeline demonstrates how to stream-out low-latency-encoded video from one device (server) to another device (client) on the same network. The pipeline is encoded with the NV12 color format, and the H265 video format. Video stream resolution is 4kp with 60fps and bitrate is 25Mbps. It sends the video stream to the client host device with an IP Address 192.168.25.89 on port 5004.

  ```bash
gst-launch-1.0 -v v4l2src device=/dev/video0 io-mode=4 ! 
video/x-raw,format=NV12,width=3840,height=2160,framerate=60/1 ! 
omxh265enc num-slices=8 periodicity-idr=240 cpb-size=500 
gdr-mode=horizontal initial-delay=250 control-rate=low-latency 
prefetch-buffer=true target-bitrate=25000 gop-mode=low-delay-p ! 
video/x-h265, alignment=nal ! rtph265pay ! 
udpsink buffer-size=60000000 host=192.168.25.89 port=5004 async=false 
max-lateness=-1 qos-dscp=60 max-bitrate=120000000 -v
```

**IMPORTANT! Replace the host IP address with the IP of the client device.**

- **Stream In:**
  
  Use the following pipeline to stream-in (stream-in → decode → display) NV12 video using a low-latency GStreamer pipeline. This pipeline demonstrates how low-latency stream-in data is decoded and displayed on the client device. The pipeline states that the encoded video format is H265, and streams-in on the client device on port 5004 - over UDP protocol.

  ```bash
gst-launch-1.0 udpsrc port=5004 buffer-size=60000000 caps='application/x-rtp, media=video, 
clock-rate=90000, payload=96, encoding-name=H265' ! rtpjitterbuffer
latency=7 ! rtph265depay ! h265parse ! video/x-h265, alignment=nal ! omxh265dec low-latency=1 ! 
video/x-raw ! queue max-size-bytes=0 ! fpsdisplaysink name=fpssink text-overlay=false
video-sink="kmssink
bus-id=a0070000.v_mix plane-id=33" sync=true -v
```
Xilinx Low-Latency

Xilinx Low-Latency: The frame is divided into multiple slices; the VCU encoder output and decoder input are processed in slice mode. The producer (Capture DMA) and the consumer (VCU Encoder) work on the same buffer by having the synchronization IP in place. There is no sync IP in between decoder and display. The decoder signals the display component when half of the input frame is ready.

- **Stream Out**

  Use the following pipeline to stream-out (capture → encode → stream-out) NV12 video using Xilinx ultra low-latency GStreamer pipeline. This pipeline demonstrates how to stream-out Xilinx ultra low-latency encoded video from one device (server) to another device (client) on the same network. The pipeline is encoded with NV12 color format, and H265 video format. Video stream resolution is 4k p with 60fps, and bitrate is 25Mbps. It sends video stream to client host device with an IP Address 192.168.25.89 on port 5004.

  ```
gst-launch-1.0 -v v4l2src device=/dev/video0 io-mode=4 ! video/x-raw\(memory:XLNXLL\), format=NV12,width=3840,height=2160,framerate=60/1 ! omxh265enc num-slices=8 periodicity-idr=240 cpb-size=500 gdr-mode=horizontal initial-delay=250 control-rate=low-latency prefetch-buffer=true target-bitrate=25000 gop-mode=low-delay-p ! video/x-h265, alignment=nal ! rtph265pay ! udpsink buffer-size=60000000 host=192.168.25.89 port=5004 async=false max-lateness=-1 qos-dscp=60 max-bitrate=120000000 -v
  ```

  IMPORTANT! Replace the host IP address with the IP of the client device.

- **Stream In**

  Use the following pipeline to stream-in (stream-in → decode → display) NV12 video using Xilinx ultra low-latency GStreamer pipeline. The pipeline demonstrates how Xilinx ultra low-latency stream-in data is decoded and displayed on the client device. The pipeline states that the encoded video format is H265 and streams-in on the client device on port 5004 - over UDP protocol.

  ```
gst-launch-1.0 udpsrc port=5004 buffer-size=60000000 caps='application/x-rtp, media=video, clock-rate=90000, payload=96, encoding-name=H265' ! rtpjitterbuffer latency=7 ! rtph265depay ! h265parse ! video/x-h265, alignment=nal ! omxh265dec low-latency=1 ! video/x-raw\(memory:XLNXLL\) ! queue max-size-bytes=0 ! fpsdisplaysink name=fpssink text-overlay=false video-sink="kmssink bus-id=a0070000.v_mix plane-id=33" sync=true -v
  ```
Transcoding

- Transcode from H.265 to H.264 using Gstreamer-1.0:

Use the following pipeline to convert a H.265 based input container format file into H.264 format.

```
gst-launch-1.0 filesrc location="/run/media/sda/input-h265-file.mp4" ! \ qtdemux name=demux demux.video_0 ! h265parse ! video/x-h265, alignment=au ! \ omxh265dec low-latency=0 ! omxh264enc ! video/x-h264, alignment=au ! \ filesink location="/run/media/sda/output.h264"
```
In this example, the file is present in the SATA drive, the H265 based input file format is MP4, and the file is transcoded into a H264 video format file.

- Transcode from H.264 to H.265 using Gstreamer-1.0:

Use the following pipeline to convert a H.264 based input container format file into H.265 format.

```
gst-launch-1.0 filesrc location="input-h264-file.mp4" ! \ qtdemux name=demux demux.video_0 ! h264parse ! video/x-h264, alignment=au ! \ omxh264dec low-latency=0 ! omxh265enc ! video/x-h265, alignment=au ! \ filesink location="output.h265"
```
In this example, the file is present in the SATA drive, the H264 based input file format is MP4, and is transcoded into a H265 video format file.

Multi-Stream

The following pipelines show that multiple streams can be played simultaneously using Gstreamer-1.0.

- Two simultaneous instances of 4k resolution:

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! \ video/x-raw, format=NV12, width=3840, height=2160, framerate=30/1 ! \ omxh265enc qp-mode=auto gop-mode=basic gop-length=60 b-frames=0 \ target-bitrate=30000 num-slices=8 control-rate=constant \ prefetch-buffer=true low-bandwidth=false \ filler-data=true cpb-size=1000 initial-delay=500 periodicity-idr=60 ! \ video/x-h265, profile=main, alignment=au ! queue ! \ mpegtsmux alignment=7 name=mux ! rtpmp2tpay ! \ udpsink host=192.168.25.89 port=5004
```
```
gst-launch-1.0 v4l2src device=/dev/video1 io-mode=4 ! \ video/x-raw, format=NV12, width=3840, height=2160, framerate=30/1 ! \ omxh265enc qp-mode=auto gop-mode=basic gop-length=60 b-frames=0 \ target-bitrate=30000 num-slices=8 control-rate=constant \ prefetch-buffer=true low-bandwidth=false filler-data=true \ cpb-size=1000 initial-delay=500 periodicity-idr=60 ! \ video/x-h265, profile=main, alignment=au ! queue ! \ mpegtsmux alignment=7 name=mux ! rtpmp2tpay ! \ udpsink host=192.168.25.89 port=5004
```
In this example, two instances of 4k resolution are streamed out. Maximum bit rate is 30 Mbps and 30fps. Encoded video format is H265.

**Note:**

1. Input source devices can be as per your choice and availability. In the preceding example, two input devices are used, video0 and video1.
2. To run multiple instances, it is recommended to execute pipelines in the background by adding `&` at the end of the pipeline.
3. To stream out both video streams, use two different ports for the same host device (that is, `port=5004`, and `port=5008`) to avoid incorrect data stream.

- Four simultaneous instances of 1080p60 resolution:

```bash
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! \
  video/x-raw, format=NV12, width=1920, height=1080, framerate=60/1 ! \
  omxh265enc qp-mode=auto gop-mode=basic gop-length=60 b-frames=0 \ 
  target-bitrate=15000 num-slices=8 control-rate=constant \ 
  prefetch-buffer=true low-bandwidth=false filler-data=true \ 
  cpb-size=1000 initial-delay=500 periodicity-idr=60 ! \
  video/x-h265, profile=main, alignment=au ! queue ! \
  mpegtsmux alignment=7 name=mux ! rtpmp2tpay ! \ 
  udpsink host=192.168.25.89 port=5004

gst-launch-1.0 v4l2src device=/dev/video1 io-mode=4 ! \
  video/x-raw, format=NV12, width=1920, height=1080, framerate=60/1 ! \
  omxh265enc qp-mode=auto gop-mode=basic gop-length=60 b-frames=0 \ 
  target-bitrate=15000 num-slices=8 control-rate=constant \ 
  prefetch-buffer=true low-bandwidth=false filler-data=true \ 
  cpb-size=1000 initial-delay=500 periodicity-idr=60 ! \
  video/x-h265, profile=main, alignment=au ! queue ! \
  mpegtsmux alignment=7 name=mux ! rtpmp2tpay ! \
  udpsink host=192.168.25.89 port=5008

gst-launch-1.0 v4l2src device=/dev/video2 io-mode=4 ! \
  video/x-raw, format=NV12, width=1920, height=1080, framerate=60/1 ! \
  omxh265enc qp-mode=auto gop-mode=basic gop-length=60 b-frames=0 \ 
  target-bitrate=15000 num-slices=8 control-rate=constant \ 
  prefetch-buffer=true low-bandwidth=false filler-data=true \ 
  cpb-size=1000 initial-delay=500 periodicity-idr=60 ! \
  video/x-h265, profile=main, alignment=au ! queue ! \
  mpegtsmux alignment=7 name=mux ! rtpmp2tpay ! \
  udpsink host=192.168.25.89 port=5012

gst-launch-1.0 v4l2src device=/dev/video3 io-mode=4 ! \
  video/x-raw, format=NV12, width=1920, height=1080, framerate=60/1 ! \
  omxh265enc qp-mode=auto gop-mode=basic gop-length=60 b-frames=0 \ 
  target-bitrate=15000 num-slices=8 control-rate=constant \ 
  prefetch-buffer=true low-bandwidth=false filler-data=true \ 
  cpb-size=1000 initial-delay=500 periodicity-idr=60 ! \
  video/x-h265, profile=main, alignment=au ! queue ! \
  mpegtsmux alignment=7 name=mux ! rtpmp2tpay ! \
  udpsink host=192.168.25.89 port=5016
```

In this example, you can stream out four instances of 1080p resolution. Maximum bit rate can be 15 Mbps and 60fps. Encoded video format is H265.
Note:

1. Input source devices can be as per your choice and availability. In the preceding example, four input devices are used, video0, video1, video2, and video3.

2. To run multiple instances, it is recommended to execute pipelines in background by adding & at the end of pipeline.

3. To stream out all four video streams, four different ports are used for same host device (that is, 5004, 5008, 5012, and 5016) to avoid incorrect data stream.

DCI 4K

DCI (Digital Cinema Initiatives) is the standards body formed by motion picture studios to establish architectures, and standards for the industry. DCI defines the Digital Cinema 4K video (4096 x 2160) format.

DCI Capture & Playback

Use the following pipeline to play a raw video stream captured from the input source device.

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! \
video/x-raw, width=4096, height=2160, format=NV12, framerate=60/1 ! \
queue ! kmssink bus-id='a0070000.v_mix'
```

In this example, the live source device link is present under the /dev directory. The video stream resolution is set at 4k for DCI with 60fps.

Interlaced Video Support

Use the following pipeline for video interlacing using gstreamer-1.0.

```
gst-launch-1.0 filesrc location=/media/card/file_1080i.h265 ! \
omxh265dec ! omxh265enc target-bitrate=10000 control-rate=2 ! \
queue max-size-bytes=-1 ! filesink location=file.h265
```

In this example, the file is present on the SD card, and the encoded video format is H265.

**IMPORTANT!** The Interlace pipeline supports only the HVEC/H265 mode of VCU.

Video Format Conversion and Scaling

Use the following pipeline to do video format conversion and scaling. The pipeline converts 1080p NV12 data to VGA BGR data.
Here, `v4l2video1convert` is a multi scalar GStreamer plugin that is created based on the `video1` node.

```bash
gst-launch-1.0 v4l2src device=/dev/video3 io-mode=4 ! \
video/x-raw, width=1920, height=1080, format=NV12 ! \
v4l2video1convert capture-io-mode=4 output-io-mode=4 ! \
video/x-raw, width=640, height=480, format=BGR ! \
filesink location=/run/Chan_1_640X480.rgb
```

**Memory-Based Scene Change Detection**

Use the following pipeline to play a captured video stream from an input source device with scene change detection (SCD), to a host device on the same network.

```bash
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! \
video/x-raw, format=NV12, width=3840, height=2160, framerate=60/1 ! \
xilinxscdio-mode=5 ! omxh265enc qp-mode=auto gop-mode=basic \ 
gop-length=60 b-frames=0 target-bitrate=60000 num-slices=8 \ 
control-rate=constant prefetch-buffer=true low-bandwidth=false \ 
filler-data=true target-cpb-size=1000 initial-delay=500 periodicity-idr=60 ! \
video/x-h265, profile=main, alignment=au ! queue ! 
mpegtsmux alignment=7 name=mux ! rtpmp2tpay ! 
udpsink host=192.168.25.89 port=5004
```

In the preceding example, the live source device link is present under the `/dev` directory. Resolution is 4k with 60fps and bitrate is 60 Mbps. Xilinx® SCD enables detecting a scene change in a video stream. The encoder can insert an I-frame to improve video quality.

For 1080p60 resolution, replace the width and height with 1920 and 1080 respectively.
Chapter 7

Debug

Use the following utilities to debug media issues.

• Media Control Utility (media-ctl)
• Modetest Utility

Media Control for Capture Link Up

The media-ctl application from the v4l-utils package is a userspace application that uses the Linux Media Controller API to configure video pipeline entities.

Run the following command to check the link up status and formats set for each of the source and sink pads.

```
$ media-ctl -d /dev/mediaX -p     -> Replace X with the corresponding media node
```

Media controller API version 5.4.0

If the capture device is connected, then the preceding command generates the following media graph:

```
Media device information
------------------------
driver          xilinx-video
model           Xilinx Video Composite Device
serial
bus info
hw revision     0x0
driver version  5.4.0

Device topology
- entity 1: vcap_hdmi output 0 (1 pad, 1 link)
  type Node subtype V4L flags 0
  device node name /dev/video0
  pad0: Sink
    <- 'a0080000.v_proc_ss':1 [ENABLED]
- entity 5: a0080000.v_proc_ss (2 pads, 2 links)
  type V4L2 subdev subtype Unknown flags 0
  device node name /dev/v4l-subdev21
  pad0: Sink
    [fmt:RGB888_1X24/1920x1080 field:none colorspace:srgb] ->
  VPSS scalar sink pad format
```
If a source is not connected to the HDMI-Rx port, then the media-ctl utility generates the following media graph:

```
$media-ctl -d /dev/mediaX -p
Media controller API version 5.4.0

Media device information
------------------------
driver          xilinx-video
model           Xilinx Video Composite Device
serial
bus info
hw revision     0x0
driver version  5.4.0

Device topology
- entity 1: vcap_hdmi output 0 (1 pad, 1 link)
  type Node subtype V4L flags 0
  device node name /dev/video0
  pad0: Sink
    [fmt:VYYUYY8_1X24/1280x720 field:none colorspace:srgb] -> 'a0080000.v_proc_ss':1 [ENABLED]

- entity 5: a0080000.v_proc_ss (2 pads, 2 links)
  type V4L2 subdev subtype Unknown flags 0
  device node name /dev/v4l-subdev15
  pad0: Sink
    [fmt:VYYUYY8_1X24/1280x720 field:none colorspace:srgb] -> 'a0000000.v_hdmi_rx_ss':0 [ENABLED]
  pad1: Source
    [fmt:VYYUYY8_1X24/1920x1080 field:none colorspace:srgb] -> 'vcap_hdmi output 0':0 [ENABLED]

- entity 8: a0000000.v_hdmi_rx_ss (1 pad, 1 link)
  type V4L2 subdev subtype Unknown flags 0
  device node name /dev/v4l-subdev16
  pad0: Source
    [fmt:RGB888_1X24/1280x720 field:none colorspace:srgb] -> 'a0000000.v_proc_ss':0 [ENABLED]
```
Modetest for Display Link Up

Use the `modetest` tool, provided by the libdrm library, to:

- List all display capabilities: CRTC, encoders & connectors (DP, HDMI, SDI ...), planes, modes...
- Perform basic tests: display a test pattern, display 2 layers, perform a vsync test
- Specify the video mode: resolution and refresh rate

**Note:** For information about the `modetest` tool, refer to [https://wiki.st.com/stm32mpu/wiki/DRM_KMS_overview#cite_note-mesa_libdrm-3](https://wiki.st.com/stm32mpu/wiki/DRM_KMS_overview#cite_note-mesa_libdrm-3).

Find the bus_id by running the following command:

```bash
cat /sys/kerne/debug/dri/<corresponding id>/name
```

Running the `modetest` command with the `-D` option and passing the bus_id provides HDMI connector status and maximum supported resolutions, frame rate, and supporting plane formats. The following is the example HDMI-Tx command.

```bash
$ modetest -D a0070000.v_mix
Encoders:
   id  crtc  type  possible crtc  possible clones
   43  42    TMDS  0x00000001  0x00000000

Connectors:
   id  encoder status  name           size (mm)  modes
   44  43     connected  HDMI-A-1       700x390  49 43
   modes:
   name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
   3840x2160 60.00 3840 3888 3920 4000 2160 2163 2168 2222 533250 flags:
   phsync, nvsync: type:
   preferred, driver
   3840x2160 60.00 3840 4016 4104 4400 2160 2168 2218 2250 594000 flags:
   phsync, pvsync: type: driver
   3840x2160 59.94 3840 4016 4104 4400 2160 2168 2218 2250 593407 flags:
   phsync, pvsync: type: driver
   3840x2160 50.00 3840 4896 4984 5280 2160 2168 2178 2250 594000 flags:
   phsync, pvsync: type: driver
   3840x2160 30.00 3840 4016 4104 4400 2160 2168 2178 2250 297000 flags:
   phsync, pvsync: type: driver
```

---

Chapter 7: Debug

**[dv.caps:BT.656/1120 min:0x0@25000000 max:4096x2160@297000000 stds:CEA-861,DMT,CVT, GTF caps:progressive,reduced-blanking,custom] [dv.query:no-link] -> "a0080000.v_proc_ss":0 [ENABLED]"**
The preceding command also shows information about the number of planes and formats, and the DRM properties of those particular planes.

<table>
<thead>
<tr>
<th>id</th>
<th>crtc</th>
<th>fb</th>
<th>CRTC x,y</th>
<th>x,y</th>
<th>gamma</th>
<th>size</th>
<th>possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>0</td>
<td>0</td>
<td>0,0</td>
<td>0,0</td>
<td>0</td>
<td>0x00000001</td>
<td></td>
</tr>
</tbody>
</table>

**Planes:**

**formats:** BG24

**props:**

8 **type:**

flags: immutable enum
enums: Overlay=0 Primary=1 Cursor=2
value: 1

17 **FB_ID:**

flags: object
value: 0

18 **IN_FENCE_FD:**

flags: signed range
values: -1 2147483647
value: -1

20 **CRTC_ID:**

flags: object
value: 0

13 **CRTC_X:**

flags: signed range
values: -2147483648 2147483647
value: 0

14 **CRTC_Y:**

flags: signed range
values: -2147483648 2147483647
value: 0

15 **CRTC_W:**

flags: range
values: 0 2147483647
value: 3840

16 **CRTC_H:**

flags: range
values: 0 2147483647
value: 2160

9 **SRC_X:**

flags: range
values: 0 4294967295
value: 0

10 **SRC_Y:**

flags: range
values: 0 4294967295
value: 0

11 **SRC_W:**

flags: range
values: 0 4294967295
value: 251658240

12 **SRC_H:**

flags: range
values: 0 4294967295
value: 141557760

To run the color pattern on the connected HDMI screen, run the following command:

```
$modetest -D <bus-id> -s <connector_id>[,<connector_id>]
[@<crtc_id>]:<mode>[,-<vrefresh>][@<format>]
$modetest -D a0070000.v_mix -s 41:3840x2160-60@BG2
```
GStreamer Debugging Techniques Using GST-launch

Use the following techniques to debug Gstreamer:

- GST_DEBUG
- GST_Shark
- GDB

GST_DEBUG

The first category is the Debug Level, which is a number specifying the amount of desired output:

Table 7: Debug Level

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>none</td>
<td>No debug information is output.</td>
</tr>
<tr>
<td>1</td>
<td>ERROR</td>
<td>Logs all fatal errors. These are errors that do not allow the core or elements to perform the requested action. The application can still recover if programmed to handle the conditions that triggered the error.</td>
</tr>
<tr>
<td>2</td>
<td>WARNING</td>
<td>Logs all warnings. Typically these are non-fatal, but user-visible problems are expected to happen.</td>
</tr>
<tr>
<td>3</td>
<td>FIXME</td>
<td>Logs all \texttt{fixme} messages - typically that a codepath that is known to be incomplete has triggered. Gstreamer may work in most cases, but may cause problems in specific instances.</td>
</tr>
<tr>
<td>4</td>
<td>INFO</td>
<td>Logs all informational messages. These are typically used for events in the system that only happen once, or are important and rare enough to be logged at this level.</td>
</tr>
<tr>
<td>5</td>
<td>DEBUG</td>
<td>Logs all debug messages. These are general debug messages for events that happen only a limited number of times during an object's lifetime; these include setup, teardown, and change of parameters.</td>
</tr>
<tr>
<td>6</td>
<td>LOG</td>
<td>Logs all log messages. These are messages for events that happen repeatedly during an object's lifetime; these include streaming and steady-state conditions. Use this level to log messages that happen on every buffer in an element, for example.</td>
</tr>
<tr>
<td>7</td>
<td>TRACE</td>
<td>Logs all trace messages. These are message that happen often. For example, each time the reference count of a \texttt{GstMiniObject}, such as a \texttt{GstBuffer} or \texttt{GstEvent}, is modified.</td>
</tr>
<tr>
<td>9</td>
<td>MEMDUMP</td>
<td>Logs all memory dump messages. This is the heaviest logging of all, and may include dumping the content of blocks of memory.</td>
</tr>
</tbody>
</table>

To enable debug output, set the GST_DEBUG environment variable to the desired debug level. All levels lower than the set level are also displayed. For example, if you set \texttt{GST_DEBUG=2}, you will see both ERROR and WARNING messages.
Furthermore, each plugin or part of the GStreamer defines its own category, so you can specify a debug level for each individual category. For example, `GST_DEBUG=2,v4l2src*:6, uses Debug Level 6 for the v4l2src element, and 2 for all the others.

The GST_DEBUG environment variable, then, is a comma-separated list of category:level pairs, with an optional level at the beginning, representing the default debug level for all categories.

The "*" wildcard is also available. For example, `GST_DEBUG=2,audio*:5` uses Debug Level 5 for all categories starting with the word audio. `GST_DEBUG=*:2` is equivalent to `GST_DEBUG=2`.

**GST_Shark**

Use `gst-shark` (a GStreamer-based tool) to verify performance and understand the element that is causing performance drops.

To check the instantaneous latencies, run the following command:

```
GST_DEBUG="GST_TRACER:7" GST_TRACERS="latency" GST_DEBUG_FILE=/run/server.txt
gst-launch-1.0 v4l2src io-mode=4 device=/dev/video0 ! video/x-raw, width=3840, height=2160, format=NV12, framerate=60/1 ! omxh265enc qp-mode=auto gop-mode=low-delay-p gop-length=60 periodicity-idd=60 b-frames=0 target-bitrate=60000 num-slices=8 control-rate=low-latency prefetch-buffer=TRUE low-bandwidth=false filler-data=0 cpb-size=1000 initial-delay=500 ! video/x-h265. alignment=nal ! queue max-size-buffers=0 ! rtph265pay ! udpsink host=192.168.25.89 port=5004 buffer-size=6000000 max-bitrate=12000000 max-lateness=-1 qos-dscp=60 async=false
```

The latency tracer module gives instantaneous latencies that may not be the same as the reported latencies. The latencies may be higher if the inner pipeline (element ! element) takes more time, or lower if the inner pipeline is running faster, but the GStreamer framework waits until the running time equals the reported latency.

Check for the time (in nanosecond for latency) marked in bold in the following logs. The initial few readings may be high due to initialization time, but become stable after initialization is complete. For example, the following logs shows ~12 ms of latency for stream-out pipeline.

```
0:00:21.633532492 20066 0x558abfb8a0 TRACE GST_TRACER :0:: latency, src-element-id=(string)0x558abea190, src-element=(string)v4l2src0, src=(string)src, sink-element-id=(string)0x558ac2b9e0, sink-element=(string)udpsink0, sink=(string)sink, time=(guint64)12399379, ts=(guint64)21633482297;
```

**GDB**

Use the `gdb` command for general debugging.
Run the application using the following command:

```
gdb -args gst-launch-1.0 v4l2src io-mode=4 device=/dev/video0
   ! video/x-raw, width=3840, height=2160, format=NV12, framerate=60/1 !
   omxh265enc
   qp-mode=auto gop-mode=low-delay-p gop-length=60 periodicity-idr=60 b-
   frames=0
   target-bitrate=60000 num-slices=8 control-rate=low-latency prefetch-
   buffers=TRUE
   low-bandwidth=false filler-data=0 cpb-size=1000 initial-delay=500 ! video/x-
   h265,
   alignment=nal ! queue max-size-buffers=0 ! rtph265pay ! udpsink
   host=192.168.25.89
   port=5004 buffer-size=6000000 max-bitrate=12000000 max-lateness=-1
   qos-dscp=60 async=false
```

The command provides the gdb shell. Type `run` to execute the application.

```
(gdb) run
```

To debug, use the backtrace function to view the last function flow.

```
(gdb) bt
```
Measuring pipeline performance

To measure pipeline performance, use the `fpsdisplaysink` in the pipeline. The following is a sample pipeline with `fpsdisplaysink` for file playback:

```
gst-launch-1.0 filesrc location="/media/card/input-file.mp4" ! qtdemux name=demux demux.video_0 ! h265parse ! omxh265dec ! queue max-size-bytes=0 ! fpsdisplaysink text-overlay=false video-sink='kmssink bus-id=a0070000.v_mix' -v
```

A sample performance log is as follows:

```
'GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 1079375, dropped: 6, current: 60.00, average: 59.99
'GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 1079406, dropped: 6, current: 60.00, average: 59.99
'GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 1079437, dropped: 6, current: 60.00, average: 59.99
'GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 1079468, dropped: 6, current: 60.00, average: 59.99
'GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 1079499, dropped: 6, current: 60.00, average: 59.99
'GstPipeline:pipeline0/GstFPSDisplaySink:fpsdisplaysink0: last-message = rendered: 1079529, dropped: 6, current: 60.00, average: 59.99
```

The log includes the following fields:

- **Current**: The instance frame rate
- **Average**: The average frame rate from the beginning
- **Dropped**: The number of frame counts that are dropped
- **Rendered**: The number of rendered frames
Performance Improvement from the GStreamer Perspective

Queue element

In the single threaded GStreamer pipeline, data starvation may occur. Use the queue element to improve the performance of a single threaded pipeline.

Data is queued until one of the limits specified by the “max-size-buffers”, “max-size-bytes” and/or “max-size-time” properties has been reached. Any attempt to push more buffers into the queue blocks the pushing thread until more space becomes available.

The queue element adds a thread boundary to the pipeline, and support for buffering. The queue creates a new thread on the source pad to decouple the processing on sink and source pad.

The default queue size limits are 200 buffers, 10MB of data, or one second worth of data, whichever is reached first.

Sample pipeline with queue element:

```bash
gst-launch-1.0 filesrc location="/media/card/input-file.mp4" ! qtdemux
name=demux
demux.video_0 ! h265parse ! omxh265dec ! queue max-size-bytes=0 !
fpsdisplaysink text-overlay=false
video-sink='kmssink bus-id=a0070000.v_mix' -v
```

Quality Of Service (QoS)

Quality of Service in GStreamer is about measuring and adjusting the real-time performance of a pipeline. The real-time performance is always measured relative to the pipeline clock and typically happens in the sinks when they synchronize buffers against the clock.

When buffers arrive late in the sink, that is, when their running-time is smaller than that of the clock, the pipeline has a quality of service problem. These are a few possible reasons:

- High CPU load, there is not enough CPU power to handle the stream, causing buffers to arrive late in the sink.
- Network problems
- Other resource problems such as disk load, and memory bottlenecks

The measurements result in QoS events that aim to adjust the data rate in one or more upstream elements. Two types of adjustments can be made:

- Short time emergency corrections based on latest observation in the sinks
- Long term rate corrections based on trends observed in the sinks
• **Short term correction:**

Use the timestamp and the jitter value in the QOS event to perform a short-term correction. If the jitter is positive, the previous buffer arrived late, and can be sure that a buffer with a timestamp < timestamp + jitter is also going to be late. Therefore, drop all buffers with a timestamp less than timestamp + jitter.

• **Long term correction:**

Long term corrections are a bit more difficult to perform. They rely on the value of the proportion in the QOS event. Elements should reduce the amount of resources they consume by the proportion field in the QoS message.

Here are some possible strategies to achieve this:

- Permanently drop frames or reduce the CPU or bandwidth requirements of the element.
- Switch to lower quality processing or reduce the algorithmic complexity. Care should be taken that this does not introduce disturbing visual or audible glitches.
- Switch to a lower quality source to reduce network bandwidth.
- Assign more CPU cycles to critical parts of the pipeline. This could, for example, be done by increasing the thread priority.

In all cases, elements should be prepared to go back to their normal processing rate, when the proportion member in the QOS event approaches the ideal proportion.

The Encoder and Decoder plugin also supports the QoS functionality.

- In the decoder, QoS is enabled by default and drops frames after decoding is finished, based on the QoS event from downstream.
- In the encoder, QoS is disabled by default and drops the input buffer while encoding, if the QoS condition is true, based on the QoS event from downstream.

**Sync**

In the GStreamer pipeline, the sync flag plays an important role. The sync flag is used for the synchronization of audio/video in the pipeline by checking the timestamp in the sink element. To know the best outcome of the pipeline, disable the sync flag in the sink element of the pipeline, keeping in mind that synchronization cannot be achieved by setting it to false. The sync flag is useful for a record pipeline to dump the data as soon as it receives it at the sink element.

Use the following example with sync=false for a record pipeline:

```bash
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! video/x-raw,
format=NV12,width=3840,height=2160,framerate=60/1 ! omxh265enc qp-
mode=auto gop-mode=basic gop-length=60 b-frames=0 target-bitrate=60000 num-slices=8 control-
rate=constant```

prefetch-buffer=true cpb-size=1000 initial-delay=500 ! queue ! video/x-h265, profile=main, alignment=au ! mpegtsmux alignment=7 name=mux ! filesink location="/media/card/test.ts" sync=false
Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select Help → Documentation and Tutorials.
- On Windows, select Start → All Programs → Xilinx Design Tools → DocNav.
- At the Linux command prompt, enter docnav.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the Design Hubs View tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents and links provide supplemental material useful with this guide:
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>H.264/H.265 Video Codec Unit LogiCORE IP Product Guide (PG252)</td>
</tr>
<tr>
<td>2.</td>
<td>Zynq UltraScale+ MPSoC ZCU106 Video Codec Unit Targeted Reference Design User Guide (UG1250)</td>
</tr>
<tr>
<td>6.</td>
<td>DisplayPort TX Subsystem Product Guide (PG199)</td>
</tr>
<tr>
<td>7.</td>
<td>DisplayPort RX Subsystem Product Guide (PG233)</td>
</tr>
<tr>
<td>8.</td>
<td>I2S Transmitter and Receiver LogiCORE IP Product Guide (PG308)</td>
</tr>
<tr>
<td>10.</td>
<td>Video Frame Buffer Read and Video Frame Buffer Write LogiCORE IP Product Guide (PG278)</td>
</tr>
<tr>
<td>11.</td>
<td>SMPTE UHD-SDI Transmitter Subsystem Product Guide (PG289)</td>
</tr>
<tr>
<td>12.</td>
<td>SMPTE UHD-SDI Receiver Subsystem Product Guide (PG290)</td>
</tr>
<tr>
<td>13.</td>
<td>Video Test Pattern Generator LogiCORE IP Product Guide (PG103)</td>
</tr>
<tr>
<td>14.</td>
<td>10 Gigabit Ethernet Subsystem Product Guide (PG157)</td>
</tr>
<tr>
<td>15.</td>
<td>AXI DMA LogiCORE IP Product Guide (PG021)</td>
</tr>
<tr>
<td>17.</td>
<td>MIPI CSI-2 Transmitter Subsystem Product Guide (PG260)</td>
</tr>
<tr>
<td>18.</td>
<td>MIPI CSI-2 Receiver Subsystem Product Guide (PG232)</td>
</tr>
<tr>
<td>20.</td>
<td>Gamma Look Up Table LogiCORE IP Product Guide (PG285)</td>
</tr>
<tr>
<td>22.</td>
<td>HDMI 1.4/2.0 Transmitter Subsystem Product Guide (PG235)</td>
</tr>
<tr>
<td>23.</td>
<td>HDMI 1.4/2.0 Receiver Subsystem Product Guide (PG236)</td>
</tr>
<tr>
<td>26.</td>
<td>Zynq UltraScale+ MPSoC VCU TRD Wiki</td>
</tr>
<tr>
<td>27.</td>
<td><a href="https://gstreamer.freedesktop.org/">https://gstreamer.freedesktop.org/</a></td>
</tr>
</tbody>
</table>
Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.
Copyright

© Copyright 2020 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.