Multimedia User Guide

UG1449 (v1.3) October 29, 2021

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
10/29/2021 Version 1.3	
General	Updated links throughout document.
Video Codec Unit	Added links to information on VCU Encoder only and Decoder only configuration.
Vitis Video Analytics SDK	New section.
06/16/2021 Version 1.2	
Basic Configuration Tab	New section.
Dynamic Insertion of IDR for Pyramidal GOP Mode	New section.
HLG Support	New section.
PCIe	New section.
HLG Pipeline	New section.
RTSP Streaming Pipeline	New section.
Monochrome Capture Pipeline	New section.
Media Control for Capture Link Up	Updated media graphs.
12/04/2020 Version 1.1	
Chapter 4: Video Pipelines	Updated chapter. Added links to pipelines.
Dynamic Features	Updated dynamic features in section.
AppSrc and AppSink Application	New section.
XAVC Record Pipeline	Updated pipeline.
Appendix A: Format Mapping	New appendix.
06/19/2020 Version 1.0	
Initial release.	N/A

Table of Contents

Revision History **2**

Chapter 1: Document Scope **5**

Navigating Content by Design Process 5

Chapter 2: Introduction to Zynq UltraScale+ MPSoC: An Ideal System for Multimedia **7**

Zynq UltraScale+ MPSoC Processing System Highlights 8

Zynq Ultrascale+ MPSoC Multimedia Blocks 9

Chapter 3: Multimedia PL IP **13**

Video Capture 13

Video Display 19

PHY Controllers 23

Audio 24

Test Pattern Generators 26

Chapter 4: Video Pipelines **28**

Chapter 5: VCU Codec Features **30**

VCU Encoder Support 30

Rate Control Modes 30

Different Group of Picture Configuration 31

Parameter List 32

HDR10 Support 34

Dynamic Features 35

GDR Support 37

XAVC 37

HLG Support 38

PCIe 38

Chapter 6: GStreamer Multimedia Framework **40**

Install and Set Up Gstreamer 40

Pixel Format Support.....41
GStreamer Plugins.....41
Raw Use Case.....43
Decode/Encode Example.....44
Camera Interface and Support.....45
AppSrc and AppSink Application.....45
Video Pipelines.....46
Audio Pipelines.....61
Vitis Video Analytics SDK.....62

Chapter 7: Debug.....64
Media Control for Capture Link Up.....64
Modetest for Display Link Up.....68
GStreamer Debugging Techniques Using GST-launch.....70

Chapter 8: Performance and Optimization.....73
Performance Improvement from the GStreamer Perspective 74

Appendix A: Format Mapping.....77

Appendix B: Additional Resources and Legal Notices.....78
Xilinx Resources.....78
Documentation Navigator and Design Hubs.....78
References.....78
Please Read: Important Legal Notices.....80

Chapter 1

Document Scope

This document describes the architecture and features of multimedia systems with Processing Subsystem (PS) + Programmable Logic (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.

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal® ACAP design process Design Hubs can be found on the Xilinx.com website. This document covers the following design processes:

- **System and Solution Planning:** Identifying the components, performance, I/O, and data transfer requirements at a system level. Includes application mapping for the solution to PS, PL, and AI Engine.
- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration.

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

For more information on GPU, refer to Chapter 5: Graphics Processing Unit of the *Zynq UltraScale + Device Technical Reference Manual* (UG1085).

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 Higlights

- 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

> **IMPORTANT!** *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 on VCU Encoder only and Decoder only configuration, see this link in *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

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. Similarly, 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.

All capture sources utilize the V4L2 framework. The V4L2 drivers can be a subset of the full LogiCORE™ PL IP specifications.

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

- One, two, four or eight pixel-wide AXI4-Stream video interface
- 8 and 10 bits per component support
- Supports spatial resolutions from 64 x 64 up to 8192 x 4320
- Supports 4K60 in all supported device families

For more information, refer to the *Gamma Look Up Table LogiCORE IP Product Guide* (PG285).

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
- 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 theAXI4-Stream Video protocol.

Features

- AXI4 Compliant
- Streaming Video Format support for: RGB, RGBA, YUV 4:4:4, YUVA 4:4:4, YUV 4:2:2, YUV 4:2:0
- 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 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.

All Display IP utilize the DRM/KMS driver framework. The available DRM / KMS driver can be a subset of the underlying LogiCORE IP.

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 otherAXI4-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 sixteen 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
- Streaming Video Format support for: RGB, RGBA, YUV 4:4:4, YUVA 4:4:4, YUV 4:2:2, YUV 4:2:0
- 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 of up to 32 channels of AES3 audio data that is transmitted over anAXI4-Stream Audio Interface on to 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 the 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 *UHDSDI 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 *Video Timing Controller LogiCORE IP Product Guide* (PG016) .

Chapter 4

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 describe 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.

AR# 75895 describes the DT changes that are not generated by the DTG tool, meaning`system-user.dtsi` needs to be updated manually. The `system-user.dtsi` file is a part of the PetaLinux BSP in the path `<petalinux-bsp>/project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`.

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**

Video Coding Parameter	H.265 (HEVC)	H.264 (AVC)
Profiles	• Main • Main Intra • Main10 • Main10 Intra • Main 4:2:2 10 • Main 4:2:2 10 Intra	• Baseline • Main • High • High10 • High 4:2:2 • High10 Intra • High 4:2:2 Intra
Levels	Up to 5.1 High Tier	Up to 5.2
Resolution	Supports up to 4KP60 (-1 speed grade) and up to DCI-4KP60 (-2 and -3 speed grade)	Supports up to 4KP60 (-1 speed grade) and up to DCI-4KP60 (-2 and -3 speed grade)
Bit Depth		
GStreamer	8-bit, 10-bit	8-bit, 10-bit
OMX	8-bit, 10-bit	8-bit, 10-bit
Chroma Format		
GStreamer	4:2:0, 4:2:2	4:2:0, 4:2:2
OMX	4:2:0, 4:2:2	4:2:0, 4:2:2

Rate Control Modes

The VCU supports the following rate control modes.

- 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 (CPBSize) 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. IPPPP....
- 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**

VCU Parameter	Description
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
Inserting Key Frame (IDR)	Specifies the number of frames between consecutive instantaneous decoder refresh (IDR) pictures. Encoder inserts a Keyframe and restart a GOP upon IDR request.
Region of Interest Encoding	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.

Table 2: **Parameter List** *(cont'd)*

VCU Parameter	Description
Long Term Reference Picture	If enabled, encoder accepts dynamically inserting and using long-term reference picture events from upstream elements. Boolean: True or False Default value: False
Adaptive GOP	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.
SEI Insertion and Extraction	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.
Dual Pass Encoding	Encode the video twice, the first pass collects information about the sequence. Statistics are used to improve the encoding of the second pass.
Scene Change Detection GDR Intra Refresh	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).
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
32-Streams Support	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.
DCI-4k Encode/Decode	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.

Table 2: **Parameter List** *(cont'd)*

VCU Parameter	Description
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.
Max Picture Size	Provide the maximum possible limit for encoded video frame, which should be a function of the input target bitrate. For more information, see the GStreamer Encoding Parameters table in *H.264/ H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252)

HDR10 Support

HDR10 is a high dynamic range standard for 10-bit video streams, allowing viewers better brightness and color contrast resulting in recreation of more realistic video frames. This HDR10 information is static metadata which is passed before the video frames to tell the display that incoming input is going to be HDR10. Xilinx® offers this solution as a better alternative to the HDMI-only connectivity solution.

Refer to *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252) for how exactly HDR10 metadata is parsed by the Xilinx VCU software stack.

For more information, see the PL DDR HDR10 HDMI Video Capture and Display wiki page.

Dynamic Features

The VCU encoder supports the following dynamic features.

Dynamic Feature Application

Use this application to set dynamic features.

Dynamic Bitrate

The Dynamic bitrate feature is useful in the scenario where you have limited network bandwidth. In network congestion scenarios, having a static bitrate can cause frame drops because the network connection is unable to keep up and will drop frames to improve the stability of your stream. This feature allows you to adjust the encoder bitrate at run time.

Dynamic bitrate is useful for Video on Demand and Video Conferencing use cases.

For more information, see Dynamic Bitrate in *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

Dynamic GOP

The Dynamic group of pictures (GOP) feature allows you to resize the GOP length, GOP structure, number of B-frames, or force IDR pictures at run time. This feature helps improve visual quality if video complexity varies throughout a single stream.

Dynamic GOP is beneficial for video recording and streaming use cases.

For more information, see Dynamic GOP in *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

Dynamic Insertion of B-Frames

Dynamic insertion of B-Frames is the ability to change the number of B-Frames in a GOP while the encoder is active. This feature improves visual quality and compression efficiency if video complexity changes within a single stream.

This is beneficial for video recording and streaming use cases.

For more information on Dynamic insertion of B-Frames, see Dynamic GOP in *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

Dynamic Insertion of Key-Frames

Dynamic insertion of key-frames is the ability to insert Instantaneous Decoder Refresh (IDR) pictures while the encoder is active. This immediately re-syncs the decoder to stop any error propagation.

For more information on dynamic insertion of key-frames, see the Dynamic GOP section in *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

Dynamic Insertion of Long Term Reference Frame

The long term reference picture is an important encoding feature that improves compression efficiency and video quality. In case of videos with a static background, it allows encoding the background scene with high quality for better motion compensation. This feature can be used in video conferences and surveillance applications. This feature can also be used in packet loss scenarios where P-frame can be used instead of traditional intra-frame (I-frame) to reconstruct the frame.

For more information, see Long Term Reference Picture in *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

Dynamic ROI

Dynamic ROI allows users to specify one or more region(s) of interest (ROI) encoding for a particular portion of a frame. This feature allows users to specify parameters which will affect visual quality of identified regions. These parameters can now be updated dynamically.

Dynamic ROI is useful for video conferencing and video surveillance.

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

Dynamic Scene Change

Dynamic scene change is the ability to restart a GOP when a scene change has been detected by an application. This improves visual quality and compression efficiency for videos with frequent scene changes.

This feature is useful in video recording and streaming use-cases.

For more information, see Dynamic Scene Change in *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

Dynamic Resolution Change

Dynamic Resolution Change is a feature that allows users to adjust the resolution of their video content while the encoder is being utilized in active pipelines. This feature allows users to reduce the encoded resolution relative to when the pipeline was first defined. Generally a reduction in encoded resolution can result in encoded bitrate savings. This can be used to adjust video resolution to suit fluctuating network bandwidths.

This feature is useful in video streaming and video conferencing use-cases.

For more information, see Dynamic Resolution Change in *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

Dynamic Insertion of IDR for Pyramidal GOP Mode

VCU encoder supports dynamically inserting IDR based on a user signal such as frame_number for Default and Low_delay_P/B gop modes. This feature handles scene_change detections better while in Pyramidal GOP mode.

GDR Support

Gradual decoder refresh (GDR) helps the decoder refresh or start decoding without a full Intra-Frame. In GDR mode, the encoder will insert intra-encoded MB strips (horizontal or vertical) into inter frames. This additional data allows for decoders to refresh, enabling the decode of compressed streams even after streams have already started or are recovering from streams due to unreliable network conditions.

For more information, see GDR Intra Refresh in *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

XAVC

This proprietary new AVC profile from SONY is mainly used for video recording.

The VCU encoder can produce xAVC Intra or xAVC Long GOP bitstreams.

To use this profile, you need to provide the information required to create the XAVC bitstream, as mentioned in the specifications. For instance, you must provide at least the resolution, the framerate, and the clock ratio.

XAVC is mainly used as a video recording option in cameras.

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

HLG Support

HLG is a HDR standard that provides backwards compatibility with non-HDR monitors while achieving a wider color gamut for HDR monitors. Because HLG EOTF is similar to standard gamma functions, HLG streams look correct when played on SDR monitors. Meanwhile PQ EOTF streams will not be displayed properly and usually look washed out when played on a SDR monitor because the gamma and PQ curves are different. Unlike HDR10, HLG does not require any extra metadata with the video data. HLG streams consist of the HLG transfer characteristics/EOTF and BT2020 color primaries and BT2020 color matrix.

For information on how HLG is processed by the software stack, see *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252).

PCIe

This design demonstrates the file-based VCU transcode, encode and decode capabilities over PCI Express® in Zynq UltraScale+ MPSoC EV devices.

This design supports AVC/HEVC video codec, the NV12, NV16, XV15, and XV20 video formats, and 4k and 1080p resolutions for encode/decode and transcode use cases.

Encode

The host application reads an input `.yuv` file from the host machine and sends it to the ZCU106 board, connected as an endpoint device to the PCIe slot of the host machine. The data received from the host is encoded with the provided encoder type. mpegtsmux is run using VCU hardware and it writes the encoded data back to the host machine in a `.ts` file.

Decode

The host application reads an input `.mp4` or `.ts` file from the host machine and sends it to the ZCU106 board, connected as an endpoint device to the PCIe slot of host machine. The data received from the host is decoded using VCU hardware. It then writes the decoded data back to the host machine in a `.yuv` file.

Transcode

The host application reads an input `.mp4` or `.ts` file from the host machine and sends it to the ZCU106 board, connected as an endpoint device to the PCIe slot of the host machine. The data received from the host is decoded, then again encoded with the provided encoder type and mpegtsmux using VCU hardware. Transcoded data is written back to the host machine in a `.ts` file.

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 of `gst-omx` video decoders, included in Gstreamer-1.0:

Table 3: **gst-omx Video Decoders**

Decoder	Description
omxh265dec	OpenMAX IL H.265 Video Decoder
omxh264dec	OpenMAX IL H.264 Video Decoder

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

Table 4: **gst-omx Video Encoders**

Encoder	Description
Omxh265enc	OpenMAX IL H.265/HEVC Video Encoder
Omxh264enc	OpenMAX IL H.264/AVC Video Encoder

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: **VCU Decoder Input and Output Format Table**

Input for VCU decoder (encoded data)	VCU decoded format
AVC or HEVC 420, 8-bit	NV12
AVC or HEVC 422, 8-bit	NV16
AVC or HEVC 420, 10-bit	NV12_10LE32
AVC or HEVC 422, 10-bit	NV16_10LE32

***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**

Plug-in	Description
v4l2src	Use v4l2src to capture video from V4L2 devices like Xilinx HDMI-RX and TPG.

Table 6: **Verified GStreamer Plug-ins** *(cont'd)*

Plug-in	Description
kmssink	The kmssink is a simple video sink that renders raw video frames directly in a plane of a DRM device. Example pipeline: `gst-launch-1.0 v4l2src ! “video/x-raw, format=NV12, width=3840, height=2160” !kmssink`
h26xparse	Parses a H.26x encoded stream. Example pipeline: `gst-launch-1.0 filesrc location=/media/card/abc.mp4 ! qtdemux ! h26xparse ! omxh26xdec ! kmssink`
omxh26xdec	The omxh26xdec is a hardware-accelerated video decoder that decodes encoded video frames. Example pipeline: `gst-launch-1.0 filesrc location=/media/card/abc.mp4 ! qtdemux ! h26xparse ! omxh26xdec ! kmssink` This pipeline shows a .mp4 multiplexed file where the encoded format is h26x encoded video. ***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.
omxh26xenc	The omxh26xenc is a hardware-accelerated video encoder that encodes raw video frames. Example pipeline: `gst-launch-1.0 v4l2src ! omxh26xenc ! filesink location=out.h26x` 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. ***Note*: Use the omxh264enc for H264 encoding, and the omxh265enc for H265 encoding.
alsasrc	Use the alsasrc plug-in to capture audio from audio devices such as Xilinx HDMI-RX. Example pipeline: `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"` This pipeline shows that the audio captured from an ALSA source, plays on an ALSA sink.
alsasink	The alsasink is a simple audio sink that plays raw audio frames. Example pipeline: `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"` This pipeline shows that the audio captured from the ALSA source, plays on an ALSA sink.

Table 6: **Verified GStreamer Plug-ins** *(cont'd)*

Plug-in	Description
faad[1]	Decoder faad is an audio decoder that decodes encoded audio frames. Example pipeline: `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="hw:1,0"` 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.
faac[1]	The faac is an audio encoder that encodes raw audio frames. Example pipeline: `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` 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.
xilinxscd	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: `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` 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. ***Note*:** Use the omxh264enc for H264 encoding, and the omxh265enc for H265 encoding.
appsrc	The appsrc element can be used by applications to insert data into a GStreamer pipeline. Unlike most GStreamer elements, appsrc provides external API functions.
appsink	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.
queue	Queues data until one of the limits specified by the `max-size-buffers`, `max-size-bytes`, or `max-size-time` properties has been reached

Notes:

1. The faac/faad plugin is not actively maintained in the community. For higher audio quality and less noise, Opus Codec (opusenc/opusdec) is an alternative.

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 ! audioresample ! \
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.

```
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

AppSrc and AppSink Application

Appsrc allows the application to feed buffers to a pipeline. Applications can use the `appsrc` element to insert data into a GStreamer pipeline. Unlike most GStreamer elements, Appsrc provides external API functions.

Appsink allows the application to get access to the raw buffer from the GStreamer pipeline. Appsink is a sink plugin that supports various methods for helping the application get a handle on the GStreamer data in a pipeline. Unlike most GStreamer elements, Appsink provides external API functions.

For an example, see this application.

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:

      ```
      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:

      ```
      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:

      ```
      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:

```
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 ! \
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 \
target-bitrate=60000 num-slices=8 control-rate=constant prefetch-
buffer=true \
low-bandwidth=false filler-data=true cpb-size=1000 initial-delay=500 ! \
queue ! 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 \
target-bitrate=60000 num-slices=8 control-rate=constant prefetch-
buffer=true \
low-bandwidth=false filler-data=true cpb-size=1000 initial-delay=500 ! \
video/x-h265, profile=main, alignment=au ! queue ! mux. alsasrc
device=hw:2,1 ! \
audio/x-raw, format=S24_32LE, rate=48000, channels=2 ! queue ! \
audioconvert ! audioresample ! 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 Mb/s. 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 ! mux. alsasrc device=hw:2,1 \
provide-clock=false ! audio/x-raw, format=S24_32LE, rate=48000, \
channels=2 ! queue ! audioconvert ! audioresample ! opusenc \
! opusparse ! mpegtsmux 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 Opus. 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.

    ```
    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.

    ```
    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 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 \
    ! opusparse ! opusdec ! 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:

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

  - For both video and audio stream-in:

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

Low Latency and Xilinx Low Latency

For a real-time experience, use low-latency and Xilinx low-latency use cases in video conferencing and broadcasting.

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 25 Mb/s. It sends the video stream to the 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,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.

```
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 internal-entropy-buffers=5
! video/x-raw ! queue max-size-bytes=0 ! fpsdisplaysink
name=fpssink text-overlay=false
'video-sink=kmssink bus-id=a0070000.v_mix hold-extra-sample=1
show-preroll-frame=false sync=true ' sync=true -v
```

Xilinx Low-Latency

Xilinx Low-Latency: In the low-latency mode, the VCU encoder and decoder work at subframe or slice level boundary but other components at the input of encoder and output of decoder namely capture DMA and display DMA still work at frame level boundary. This means that the encoder can read input data only when capture has completed writing the full frame.

In the Xilinx low-latency mode, capture and display also work at subframe level and therefore reduce the pipeline latency significantly. This is made possible by making the producer (Capture DMA) and the consumer (VCU encoder) work on the same input buffer concurrently, but maintaining the synchronization between the two such that consumer read request is unblocked only when the producer is done writing the data required for that read request.

- 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 4kp with 60fps, and bitrate is 25 Mb/s. 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 internal-entropy-buffers=5
! video/x-raw\(memory:XLNXLL\) ! queue max-size-bytes=0
! fpsdisplaysink name=fpssink text-overlay=false
'video-sink=kmssink bus-id=a0070000.v_mix hold-extra-sample=1
show-preroll-frame=false sync=true ' sync=true -v
```

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

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=5008
```

In this example, two instances of 4k resolution are streamed out. Maximum bit rate is 30 Mb/s and 30 fps. 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:

```
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/video2io-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/video3io-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=truelow-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 Mb/s and 60 fps. 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 is only supported with a speed grade of -2 and above.

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.

```
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.

```
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 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 Mb/s. 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.

HDR10 Pipeline

The HDR10 pipeline supports the reception and insertion of HDR10 static metadata. This HDR10 metadata contains critical information needed to support HDR and will be carried throughout the pipeline - from the source to the sink.

Run the following gst-launch-1.0 command to **display** the XV20 HDR10 video on HDMI-Tx using the GStreamer pipeline (capture (HDR10) → encode → decode → display(HDR10)):

```
gst-launch-1.0 v4l2src device=/dev/video0
io-mode=4 ! video/x-raw, width=3840, height=2160,
format=NV16_10LE32, 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 ! video/x-h265, profile=main-422-10,
alignment=au ! queue ! omxh265dec internal-entropy-buffers=5
low-latency=0 ! queue max-size-bytes=0 ! fpsdisplaysink
text-overlay=false
video-sink="kmssink bus-id="a00c0000.v_mix show-preroll-frame=false"
plane-id=34 sync=true" sync=true
```

Run the following gst-launch-1.0 command to record XV20 HDR10 video using the GStreamer pipeline:

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4
num-buffers=3600 ! video/x-raw, width=3840, height=2160,
format=NV16_10LE32, 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 ! video/x-h265, profile=main-422-10,
alignment=au ! h265parse ! queue ! mpegtsmux alignment=7
name=mux ! filesink location="/run/test.ts"
```

For more information, see *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252) and PL DDR HDR10 HDMI Video Capture and Display.

Videotestsrc

The `videotestsrc` element is an open source up-streamed GStreamer plugin. It is used to produce test video data in a wide variety of formats. You can use the `pattern` property to control the video test data that is produced.

By default, `videotestsrc` will generate data indefinitely, but if the num-buffers property is non-zero it will instead generate a fixed number of video frames and then send EOS.

To test the encode functionality, use the following pipeline:

```
gst-launch-1.0 videotestsrc \
! video/x-raw,format=NV12,width=1920, \
height=1080,framerate=60/1 \
!  videoconvert \
! omxh265enc prefetch-buffer=true \
! fakesink
```

***Note*:** `videotestsrc` does not support zero copy. The primary aim of this pipeline is to validate encoder functionality.

In the preceding example, the source device generates a video pattern with 1080p resolution and 60fps. The video stream color format is NV12.

To test the display functionality, use the following pipeline:

```
gst-launch-1.0 videotestsrc \
! video/x-raw,format=NV12,width=1920,\
height=1080,framerate=60/1 \
!  videoconvert \
! fpsdisplaysink \
video-sink="kmssink bus-id=a00c0000.v_mix" \
text-overlay=false sync=false -v
```

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

Processed Serial Pipeline

Use the following pipeline to play a serial processed pipeline where video stream is captured from an input source device:

```
gst-launch-1.0 v4l2src device=/dev/videoX io-mode=4 ! video/x-raw, \
width=3840, height=2160, format=NV12, framerate=60/1 ! xilinxscd io-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 cpb-size=1000 initial-delay=500 \
! video/x-h265, profile=main, alignment=au ! queue \
! omxh265dec internal-entropy-buffers=5 low-latency=0 \
! queue max-size-bytes=0 ! kmssink bus-id="a0070000.v_mix"
```

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

For more information, see *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252) and HDMI Video Capture and Display.

XAVC Record Pipeline

This is the SONY proprietary new AVC profile, mainly used for video recording.

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 num-buffers=1000 \
! video/x-raw, width=3840,height=2160, \
format=NV16_10LE32, framerate=60000/1001 \
! queue ! omxh264enc control-rate=constant target-bitrate=599545 \
xavc-max-picture-sizes-in-bits="<10002432,0,0>" gop-mode=basic \
gop-length=1 b-frames=0 entropy-mode=CAVLC num-slices=8 \
cpb-size=1000 initial-delay=500 prefetch-buffer=TRUE aspect-ratio=1 \
quant-i-frames=51 min-qp=31 max-qp=51 \
! video/x-h264 , profile=xavc-high-4:2:2-intra-cbg, level=\(string\)5.2, \
alignment=au ! queue max-size-buffers=0 ! fpsdisplaysink name=fpssink \
text-overlay=false 'video-sink=filesink \
location="/run/xavc_intra_QFHD_4k_c300_CBG.h264"' \
sync=true -v
```

In the preceding example, the live source device link is present under the `/dev` directory. The video stream resolution is UHD and framerate is 59.94fps. The video stream color format is XV20.

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

HLG Pipeline

Two HLG modes can be enabled for the VCU:

- **Backwards compatible (SDR EOTF + HLG SEI):** This mode uses the BT2020 value in the SPS/VUI parameters instead of the HLG transfer characteristics. The VCU encoder inserts alternative transfer characteristics (ATC) SEI with the HLG value. The following is a sample serial pipeline:

```
gst-launch-1.0 -v v4l2src device=/dev/video0 ! video/x-raw, width=3840, \
height=2160, framerate=60/1, format=NV16_10LE32 ! queue max-size-bytes=0 \
! omxh265enc control-rate=constant target-bitrate=25000 prefetch-
buffer=TRUE num-slices=8 \
! omxh265dec ! queue max-size-bytes=0 ! fpsdisplaysink name=fpssink text-
overlay=false \
'video-sink=kmssink connector-
properties="props,sdi_mode=5,sdi_data_stream=8,is_frac=0,\
sdi_420_in=0,c_encoding=1" show-preroll-frame=false sync=true'
```

- **HLG only (HLG EOTF):** This mode directly uses the HLG value in the SPS/VUI parameters. The following is a sample serial pipeline:

```
gst-launch-1.0 -v v4l2src device=/dev/video0 ! video/x-raw, width=3840, \
height=2160, framerate=60/1, format=NV16_10LE32 ! queue max-size-bytes=0 \
! omxh265enc control-rate=constant target-bitrate=25000 prefetch-
buffer=TRUE num-slices=8 \
! omxh265dec ! queue max-size-bytes=0 ! fpsdisplaysink name=fpssink text-
overlay=false \
'video-sink=kmssink connector-
properties="props,sdi_mode=5,sdi_data_stream=8,is_frac=0,\
sdi_420_in=0,c_encoding=1" show-preroll-frame=false sync=true'
```

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

RTSP Streaming Pipeline

gst-rtsp-server is a library on top of GStreamer for building an real-time streaming protocol server (RTSP). RTSP is a presentation-layer protocol that lets you command media servers via pause and play capabilities, whereas RTP is the transport protocol used to move the data. RTSP is a dependable technology used to control audio/video transmission between two endpoints and facilitate the transportation of low-latency streaming content across the Internet.

To build the open source rtsp_server_application, see AR76505.

To run the demo:

1. Connect the target board and host machine in loopback.

2. Run the following pipeline on the target board:

```
./rtsp_server_app "v4l2src io-mode=4 device=/dev/video0 ! \
video/x-raw,format=NV12,width=1920,height=1080,framerate=60/1 ! \
omxh265enc prefetch-buffer=true gop-length=60 control-rate=2 target-
bitrate=10000 \
filler-data=false ! h265parse ! rtph265pay name=pay0 pt=96"
"192.168.2.222" "50000"
```

3. On the host machine, open the following URL in the VLC player: rtsp://192.168.2.222:50000/test

Monochrome Capture Pipeline

Use the following pipeline to play GRAY8 and GRAY10_LE32 format.

The following example uses the video test source. The video stream resolution is 1080p and 30fps, and the video stream color format is GRAY8.

```
gst-launch-1.0 -v videotestsrc ! video/x-raw, width=1920, height=1080,
format=GRAY8, \
framerate=30/1 ! queue ! omxh265enc gop-mode=low-delay-p periodicity-
idr=240 \
target-bitrate=6250 num-slices=8 control-rate=low-latency prefetch-
buffer=TRUE \
filler-data=0 cpb-size=500 initial-delay=250 gdr-mode=horizontal ! video/x-
h265, \
alignment=nal ! queue max-size-buffers=0 ! omxh265dec low-latency=1 ! \
queue max-size-bytes=0 ! kmssink bus-id="fd4a0000.display"
```

The following example uses the video test source. The video stream resolution is 1080p and 30fps, and the video stream color format is GRAY10_LE32.

```
gst-launch-1.0 -v videotestsrc ! video/x-raw, width=1920, height=1080,
format=GRAY10_LE32, \
framerate=30/1 ! queue ! omxh265enc gop-mode=low-delay-p periodicity-
idr=240 \
target-bitrate=6250 num-slices=8 control-rate=low-latency filler-data=0 cpb-
size=500 \
initial-delay=250 gdr-mode=horizontal ! video/x-h265, alignment=nal ! queue
max-size-buffers=0 ! \
omxh265dec low-latency=1 ! queue max-size-bytes=0 ! kmssink bus-
id="fd4a0000.display"
```

For more information, see *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252) and HDMI Video Capture and Display.

Audio Pipelines

List of audio input devices:

File, HDMI-Rx, I2S-Rx and SDI-Rx

List of audio output devices:

File, HDMI-Tx, I2S-Tx, SDI-Tx and DP

Dual Channel Audio-Video Record Pipeline

In the followng example, the video source device generates video at 4kp resolution and 60fps. The video stream color format is NV12. The audio source device generates audio at 48KHz with S24_32LE format and dual channel from input device ID. The audio capture device ID is **hw:2,1**.

```
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 ! video/x-h265,
profile=main, alignment=au ! queue ! mux. alsasrc device=hw:2,1
provide-clock=false ! audio/x-raw, format= S24_32LE, rate=48000,
channels=2 ! queue ! audioconvert ! audioresample ! opusenc
! opusparse ! mpegtsmux name=mux
! filesink location = "/run/test.ts
```

Dual Channel Audio-Video Playback Pipeline

In the following example, the audio renderer renders the audio at 48KHz with S24_32LE format and dual channel at output device ID. The audio playback device ID is **hw:2,0**.

```
gst-launch-1.0 uridecodebin uri="file:///run/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 ! queue
! alsasink device="hw:2,0"
```

8-Channel Audio-Video Serial Pipeline

In the following example, the video source device generates video at 4kp resolution and 60fps. The video stream color format is XV20. The audio source device generates audio at 48KHz with S24_32LE format and 8 channels from input device ID. The audio capture device ID is **hw:1,1**, and the audio playback device ID is **hw:1,0**.

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4
! video/x-raw, width=3840, height=2160, format=NV16_10LE32,
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
! video/x-h265, profile=main-422-10, alignment=au
! queue max-size-bytes=0 ! omxh265dec internal-entropy-buffers=5
low-latency=0 ! queue max-size-bytes=0 ! fpsdisplaysink
text-overlay=false
video-sink="kmssink driver-name=xlnx async=false hold-extra-sample=true
show-preroll-frame=false"
alsasrc device=hw:1,1 provide-clock=false ! audio/x-raw, rate=48000,
channels=8, format=S24_32LE ! queue max-size-bytes=0
! alsasink device="hw:1,0"
```

For more information, see PL DDR SDI Audio Video Capture and Display.

Vitis Video Analytics SDK

The Vitis Video Analytics SDK (VVAS) is a framework to build transcoding and AI-powered solutions on Xilinx platforms. It takes input data - from USB/CSI camera, video from file or streams over RTSP, and uses Vitis™ AI to generate insights from pixels for various use cases. VVAS SDK can be the foundation layer for a number of video analytic solutions like understanding traffic and pedestrians in a smart city, health and safety monitoring in hospitals, self-checkout and analytics in retail, detecting component defects at a manufacturing facility and others. VVAS can also be used to build Adaptive Bitrate Transcoding solutions that may require re-encoding the incoming video at different bitrates, resolution, and encoding format.

The core SDK consists of several hardware accelerator plugins that use various accelerators such as multiscaler (for resize and color space conversion), and deep learning processing unit (DPU) for machine learning. By performing all the compute heavy operations in dedicated accelerators, VVAS can achieve highest performance for video analytics, transcoding and several other application areas.

Features

- Ships several hardware accelerators for various functions
- Provides highly optimized GStreamer plugins, which meets most of the requirements of the Video Analytics and transcoding solutions

- Provides an easy to use framework to integrate the hardware accelerators/kernels in Gstreamer framework based applications
- Provides AI model support for popular object detection and classification models like SSD, YOLO etc

Advantages

- Application developers can build seamless streaming pipelines for AI-based video and image analytics, complex Adaptive Bitrate Transcoding pipelines, and several other solutions using VVAS, without having any understanding about low level environment complexities.
- VVAS provides the flexibility for rapid prototyping to full production level solutions by significantly reducing the time to market for the solutions on Xilinx platforms.

For more information on VVAS and its applications, see the following:

- Vitis Video Analytics SDK Overview
- Zynq UltraScale+ MPSoC ZCU106 VCU HDMI Single-Stream ROI TRD 2021.1
- Defect Detection Accelerated Application
- Vitis Video Analytics SDK Source Code

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 controller API version 5.10.0

Media device information
------------------------
driver          xilinx-video
model           Xilinx Video Composite Device
serial
bus info
hw revision     0x0
driver version  5.10.0

Device topology
- entity 1: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video0
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":1 [ENABLED]

- entity 5: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video1
    pad0: Sink
```

```
        <- "amba_pl@0:axis_broadcasterhdmi_":2 [ENABLED]

- entity 9: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video2
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":3 [ENABLED]

- entity 13: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
             type Node subtype V4L flags 0
             device node name /dev/video3
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":4 [ENABLED]

- entity 17: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
             type Node subtype V4L flags 0
             device node name /dev/video4
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":5 [ENABLED]

- entity 21: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
             type Node subtype V4L flags 0
             device node name /dev/video5
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":6 [ENABLED]

- entity 25: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
             type Node subtype V4L flags 0
             device node name /dev/video6
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":7 [ENABLED]

- entity 29: amba_pl@0:axis_broadcasterhdmi_ (8 pads, 8 links)
             type V4L2 subdev subtype Unknown flags 0
             device node name /dev/v4l-subdev15
    pad0: Sink
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        <- "a0080000.v_proc_ss":1 [ENABLED]
    pad1: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad2: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad3: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad4: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad5: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad6: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad7: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]

- entity 38: a0080000.v_proc_ss (2 pads, 2 links)
             type V4L2 subdev subtype Unknown flags 0
             device node name /dev/v4l-subdev16
    pad0: Sink
        [fmt:RBG888_1X24/3840x2160 field:none]
        <- "a0000000.v_hdmi_rx_ss":0 [ENABLED]
    pad1: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "amba_pl@0:axis_broadcasterhdmi_":0 [ENABLED]

- entity 41: a0000000.v_hdmi_rx_ss (1 pad, 1 link)
             type V4L2 subdev subtype Unknown flags 0
             device node name /dev/v4l-subdev17
    pad0: Source
        [fmt:RBG888_1X24/3840x2160 field:none]
        [dv.caps:BT.656/1120 min:0x0@25000000 max:4096x2160@297000000
stds:CEA-861,DMT,CVT,GTF caps:progressive,reduced-blanking,custom]
        [dv.detect:BT.656/1120 3840x2160p60 (4400x2250) stds:CEA-861
flags:CE-video]
        -> "a0080000.v_proc_ss":0 [ENABLED]
```

If a source is not connected to the HDMI-Rx port, then the `media-ctl` utility generates the following media graph:

```
Media controller API version 5.10.0

Media device information
------------------------
driver          xilinx-video
model           Xilinx Video Composite Device
serial
bus info
hw revision     0x0
driver version  5.10.0

Device topology
- entity 1: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video0
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":1 [ENABLED]

- entity 5: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video1
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":2 [ENABLED]

- entity 9: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
            type Node subtype V4L flags 0
            device node name /dev/video2
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":3 [ENABLED]

- entity 13: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
             type Node subtype V4L flags 0
             device node name /dev/video3
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":4 [ENABLED]

- entity 17: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
             type Node subtype V4L flags 0
             device node name /dev/video4
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":5 [ENABLED]
```

```
- entity 21: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
             type Node subtype V4L flags 0
             device node name /dev/video5
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":6 [ENABLED]

- entity 25: vcapaxis_broad_out1hdmi_input_a (1 pad, 1 link)
             type Node subtype V4L flags 0
             device node name /dev/video6
    pad0: Sink
        <- "amba_pl@0:axis_broadcasterhdmi_":7 [ENABLED]

- entity 29: amba_pl@0:axis_broadcasterhdmi_ (8 pads, 8 links)
             type V4L2 subdev subtype Unknown flags 0
             device node name /dev/v4l-subdev15
    pad0: Sink
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        <- "a0080000.v_proc_ss":1 [ENABLED]
    pad1: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad2: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad3: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad4: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad5: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad6: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]
    pad7: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "vcapaxis_broad_out1hdmi_input_a":0 [ENABLED]

- entity 38: a0080000.v_proc_ss (2 pads, 2 links)
             type V4L2 subdev subtype Unknown flags 0
             device node name /dev/v4l-subdev16
    pad0: Sink
        [fmt:RBG888_1X24/3840x2160 field:none]
        <- "a0000000.v_hdmi_rx_ss":0 [ENABLED]
    pad1: Source
        [fmt:VYYUYY8_1X24/3840x2160 field:none]
        -> "amba_pl@0:axis_broadcasterhdmi_":0 [ENABLED]

- entity 41: a0000000.v_hdmi_rx_ss (1 pad, 1 link)
             type V4L2 subdev subtype Unknown flags 0
             device node name /dev/v4l-subdev17
    pad0: Source
        [fmt:RBG888_1X24/3840x2160 field:none]
        [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]
```

***Note*:** Media Graph and Entity Name can vary as per design. For exact media graph of specific design, refer to the relevant design wiki pages of the desired release.

Modetest for Display Link Up

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

- List all display capabilities: CRTCs, 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.

Find the bus_id by running the following command:

```
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.

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

Connectors:
id      encoder status          name            size (mm)       modes
encoders
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 2178 2250 594000 flags:
phsync, pvsync; type: driver
  3840x2160 59.94 3840 4016 4104 4400 2160 2168 2178 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
  3840x2160 30.00 3840 4016 4104 4400 2160 2168 2178 2250 297000 flags:
phsync, pvsync; type: driver
```

The preceding command also shows information about the number of planes and formats, and the DRM properties of those particular planes.

```
Planes:
id      crtc    fb      CRTC x,y        x,y     gamma size      possible
crtcs
41      0       0       0,0             0,0     0               0x00000001
  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**

Number	Name	Description
0	none	No debug information is output.
1	ERROR	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.
2	WARNING	Logs all warnings. Typically these are non-fatal, but user-visible problems are expected to happen.
3	FIXME	Logs all `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.
4	INFO	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.
5	DEBUG	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.
6	LOG	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.
7	TRACE	Logs all trace messages. These are message that happen often. For example, each time the reference count of a `GstMiniObject`, such as a `GstBuffer` or `GstEvent`, is modified.
9	MEMDUMP	Logs all memory dump messages. This is the heaviest logging of all, and may include dumping the content of blocks of memory.

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 `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-idr=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=60000000
max-bitrate=120000000 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-
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=60000000 max-bitrate=120000000 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
```

Performance and Optimization

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:

```
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:

```
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
```

Format Mapping

The following table shows the mapping of formats in GStreamer, V4L2, Media Control, and DRM.

Table 8: **Format Mapping**

S. No.	YUV Format	GStreamer	V4L2 Framework	Media Bus Format	DRM Format
1	YUV 4:2:0, 8 bit	GST_VIDEO_FORMAT_NV12	V4L2_PIX_FMT_NV12	MEDIA_BUS_FMT_VYYUYY8_1X24	DRM_FORMAT_NV12
2	YUV 4:2:2, 8 bit	GST_VIDEO_FORMAT_NV16	V4L2_PIX_FMT_NV16	MEDIA_BUS_FMT_UYVY8_1X16	DRM_FORMAT_NV16
3	YUV 4:2:0, 10 bit	GST_VIDEO_FORMAT_NV12_10LE32	V4L2_PIX_FMT_XV15	MEDIA_BUS_FMT_VYYUYY10_4X20	DRM_FORMAT_XV15
4	YUV 4:2:2, 10 bit	GST_VIDEO_FORMAT_NV16_10LE32	V4L2_PIX_FMT_XV20	MEDIA_BUS_FMT_UYVY10_1X20	DRM_FORMAT_XV20

Note: For information on the YUV 444 format, see AR 76939.

Appendix B

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:

1. *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252)
2. *Zynq UltraScale+ MPSoC ZCU106 Video Codec Unit Targeted Reference Design User Guide* (UG1250)
3. *PetaLinux Tools Documentation: Reference Guide* (UG1144)
4. *Video Scene Change Detection LogiCORE IP Product Guide* (PG322)
5. *Zynq UltraScale+ Device Technical Reference Manual* (UG1085)
6. *DisplayPort TX Subsystem Product Guide* (PG199)
7. *DisplayPort RX Subsystem Product Guide* (PG233)
8. *I2S Transmitter and I2S Receiver Product Guide* (PG308)
9. *Audio Formatter Product Guide* (PG330)
10. *Video Frame Buffer Read and Video Frame Buffer Write LogiCORE IP Product Guide* (PG278)
11. *SMPTE UHD-SDI Transmitter Subsystem Product Guide* (PG289)
12. *SMPTE UHD-SDI Receiver Subsystem Product Guide* (PG290)
13. *Video Test Pattern Generator LogiCORE IP Product Guide* (PG103)
14. *10 Gigabit Ethernet Subsystem Product Guide* (PG157)
15. *AXI DMA LogiCORE IP Product Guide* (PG021)
16. *Video PHY Controller LogiCORE IP Product Guide* (PG230)
17. *MIPI CSI-2 Transmitter Subsystem Product Guide* (PG260)
18. *MIPI CSI-2 Receiver Subsystem Product Guide* (PG232)
19. *Sensor Demosaic LogiCORE IP Product Guide* (PG286)
20.
21. *Video Processing Subsystem Product Guide* (PG231)
22. *HDMI 1.4/2.0 Transmitter Subsystem Product Guide* (PG235)
23. *HDMI 1.4/2.0 Receiver Subsystem Product Guide* (PG236)
24. *Video Mixer LogiCORE IP Product Guide* (PG243)
25. *Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications* (https://www.zynq-mpsoc-book.com/)
26. Zynq UltraScale+ MPSoC VCU TRD Wiki
27. https://gstreamer.freedesktop.org/

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright