H.264/H.265 Video Codec Unit v1.2 Solutions

LogiCORE IP Product Guide

Vivado Design Suite

PG252 (v2021.2) October 27, 2021

Xilinx is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this link for more information.

Table of Contents

XILINX

Section I

H.264/H.265 Video Codec Unit v1.2

Introduction

The Xilinx® LogiCORE™ IP H.264/H.265 Video Codec Unit (VCU) core for Zynq® UltraScale+™ MPSoCs is capable of compressing and decompressing video streams simultaneously at resolutions of up to 3840×2160 pixels at 60 frames per second (4K UHD at 60 Hz). H.264/H.265 functionality is implemented as an embedded hard IP inside Zynq UltraScale+ MPSoC EV devices. The VCU is suitable for applications including but not limited to video surveillance and video over IP connectivity. The applications supported include video conferencing, embedded vision, and biomedical instrumentation.

Features

The features of the Xilinx LogiCORE IP H.264/H.265 Video Codec Unit (VCU) core for Zynq UltraScale+ MPSoC devices are as follows:

- 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, Main10, 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@60fps
- Low latency rate control
- Flexible rate control: CBR, VBR, and Constant QP
- Supports simultaneous encoding and decoding up to 4K UHD resolution at 60 Hz

  Note: 4k (3840x2160) and below resolutions are supported in all speedgrades. However, 4K DCI (4096x2160) requires -2 or -3 speedgrade.
- Supports 8K UHD at reduced frame rate (~15 Hz)
- Progressive support for H.264 and H.265; Interlace support for H.265

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

IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq® UltraScale+™ MPSoC EV Devices
Supported User Interfaces	AXI4-Lite, AXI4
Resources	See Resource Utilization
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	Verilog
Constraints File	Xilinx Design Constraints File (XDC)
Simulation Model	Not Provided
Supported S/W Driver	Included in PetaLinux
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 76600
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Provided by Xilinx at the Xilinx Support web page	

Notes:
1. For a complete list of supported devices, see the Vivado® IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

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. Topics in this document that apply to this design process include:
  - Section I: H.264/H.265 Video Codec Unit v1.2
    - Product Specification
    - Core Architecture
    - Designing with the Core
    - Design Flow Steps
  - Section II: Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP v1.1
    - Product Specification
    - Core Architecture
    - Designing with the Core
    - Design Flow Steps
  - Section III: VCU Sync IP v1.0
    - Product Specification
    - Core Architecture
    - Designing with the Core
    - Software Flow Overview
    - Sync IP Software Programming Model
- **Embedded Software Development:** Creating the software platform from the hardware platform and developing the application code using the embedded CPU. Also covers XRT and Graph APIs. Topics in this document that apply to this design process include:
  - Section III: VCU Sync IP v1.0
    - Software Flow Overview
    - Sync IP Software Programming Model
  - Section V: Application Software Development
    - Encoder and Decoder Software Features
    - Preparing PetaLinux to Run VCU Applications
    - VCU Encoder Features
    - GStreamer Pipelines

    - OpenMax Integration Layer
    - Sync IP API
    - VCU Control Software
    - Driver
    - MCU Firmware
    - Encoder and Decoder Stacks
    - Encoder Flow
    - Decoder Flow

- **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. Topics in this document that apply to this design process include:
    - Section I: H.264/H.265 Video Codec Unit v1.2
        - Designing with the Core
        - Design Flow Steps
    - Section II: Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP v1.1
        - Designing with the Core
        - Design Flow Steps
    - Section III: VCU Sync IP v1.0
        - Designing with the Core
        - Design Flow Steps

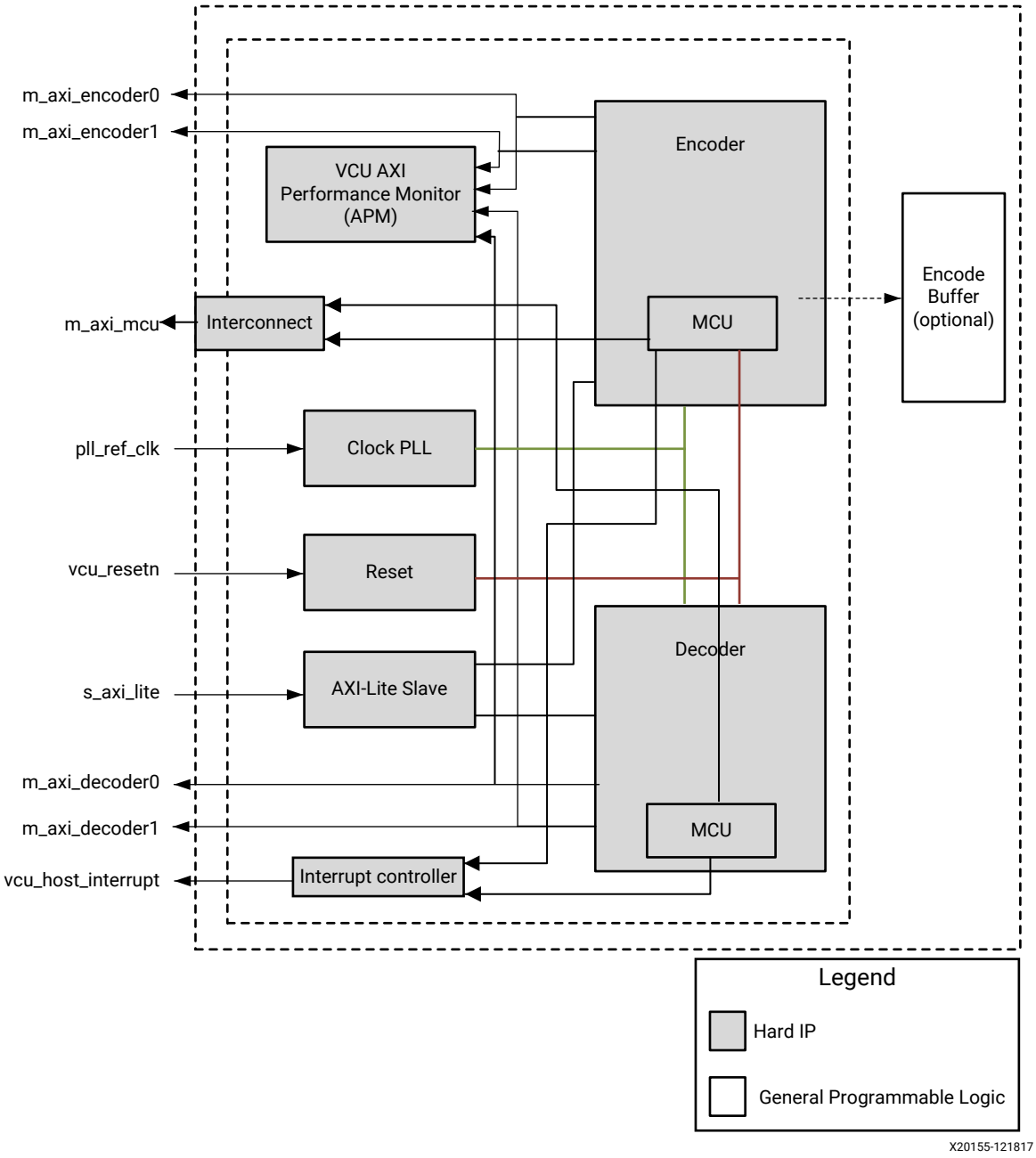
- **System Integration and Validation:** Integrating and validating the system functional performance, including timing, resource use, and power closure. Topics in this document that apply to this design process include:
    - Section IV: Performance and Debugging
        - Latency in the VCU Pipeline
        - Debugging
    - Section VI: Appendices
        - Codec Parameters for Different Use Cases
        - QoS Configurations
        - IRQ Balancing

Overview

The LogiCORE™ IP H.264/H.265 Video Codec Unit (VCU) core supports multi-standard video encoding and decoding, including support for the high-efficiency Video Coding (HEVC) and Advanced Video Coding (AVC) H.264 standards. The unit contains both encode (compress) and decode (decompress) functions, and is capable of simultaneous encode and decode.

The VCU is an integrated block in the programmable logic (PL) of selected Zynq UltraScale+ MPSoCs with no direct connections to the processing system (PS), and contains encoder and decoder interfaces. The VCU also contains additional functions that facilitate the interface between the VCU and the PL. VCU operation requires the application processing unit (APU) to service interrupts to coordinate data transfer. The encoder is controlled by the APU through a task list prepared in advance, and the APU response time is not in the execution critical path. The VCU has no audio support. Audio encoding and decoding can be done in software using the PS or through soft IP in the PL. The following figure shows the top-level block diagram with the VCU core.

Figure 1: **Top-level Block Diagram**



Encoder Block Overview

The encoder engine is designed to process video streams using the HEVC (ISO/IEC 23008-2 high-efficiency Video Coding) and AVC (ISO/IEC 14496-10 Advanced Video Coding) standards. It provides complete support for these standards, including support for 8-bit and 10-bit color, Y-only (monochrome), 4:2:0 and 4:2:2 Chroma formats, up to 4K UHD at 60 Hz performance. The encoder contains global registers, an interrupt controller, and a timer. The encoder is controlled by a microcontroller (MCU) subsystem. VCU applications running on the APU use the Xilinx® VCU Control Software library API to interact with the encoder microcontroller. The microcontroller firmware (MCU Firmware) is not user modifiable.

A 32-bit AXI4-Lite interface is used by the APU to control the MCU (to configure encoding parameters). Two 128-bit AXI4 master interfaces are used to move video data and metadata to and from the system memory. A 32-bit AXI4 master interface is used to fetch the MCU software (instruction cache interface) and load/store additional MCU data (data cache interface).

Decoder Block Overview

The Decoder block is capable of processing video streams using the HEVC (ISO/IEC 23008-2 High Efficiency Video Coding) and AVC (ISO/IEC 14496-10 Advanced Video Coding) standards. It provides a complete support for these standards, including support for 8-bit and 10-bit color depth, Y-only (monochrome), 4:2:0 and 4:2:2 Chroma formats, up to 4K UHD at 60 Hz performance. It also contains global registers, an interrupt controller, and a timer.

The VCU decoder is controlled by a microcontroller (MCU) subsystem. A 32-bit AXI4-Lite slave interface is used by the APU to control the MCU. Two 128-bit AXI4 master interfaces are used to move video data and metadata to and from the system memory. A 32-bit AXI4 master interface is used to fetch the MCU software (instruction cache interface) and load/store additional MCU data (data cache interface). VCU applications running on the APU use the Xilinx VCU Control Software library API to interact with the decoder microcontroller. The microcontroller firmware is not user modifiable.

The decoder includes control registers, a bridge unit and a set of internal memories. The bridge unit manages the request arbitration, burst addresses, and burst lengths for all external memory accesses required by the decoder.

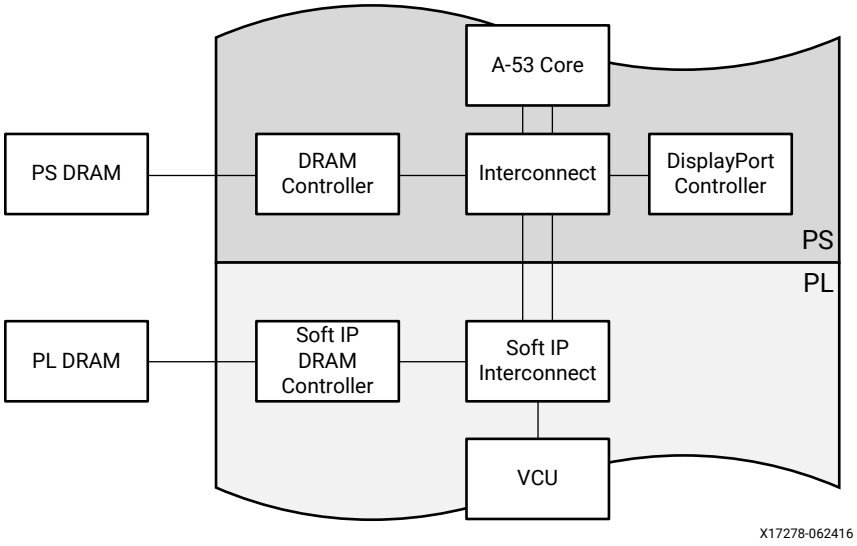
Microcontroller Unit Overview

The encoder and decoder blocks each implement a 32-bit microcontroller unit (MCU) to handle interaction with the hardware blocks. The MCU receives commands from the APU, parses the command into multiple slice- or tile-level commands, and executes them on the encoder and decoder blocks. After the command is executed, the MCU communicates the status to the APU and the process is repeated.

Applications

The VCU core is a dedicated circuitry located in the PL to enable maximum flexibility for a wide selection of use cases, memory bandwidth being a key driver. Whether the application requires simultaneous 4K UHD at 60 Hz encoding and decoding or a single SD stream to be processed, a system design and memory topology can be implemented that balances performance, optimization, and integration for the specific use case. The following figure shows the use case example where the VCU core works with the PS and the PL DDR external memory.

Figure 2: **VCU Application**



Unsupported Features

The following features are not supported:

- Scalable video coding in AVC/HEVC
  - Non-Annex B (AVCC)
- H.264 (AVC)
  - Interlace video format
  - Encoder:
    - Lossless mode (transform bypass, I_PCM)
  - Decoder:
    - Flexible macroblock ordering (FMO)

    - Arbitrary slice ordering (ASO)
    - Redundant slice (RS)
    - Dynamic chroma format/profile/level change within a single stream
- H.265 (HEVC)
  - Encoder:
    - Sample adaptive offset (SAO) filter
    - Asymmetric motion partition (AMP)
    - Lossless mode (transquant bypass, PCM)
    - Transform skip mode
    - Wavefront parallel processing (WPP)
  - Decoder:
    - Dynamic chroma format/profile/level change within a single stream

  See the *Zynq UltraScale+ MPSoC Production Errata* (EN285) for more information.

Licensing and Ordering

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

For more information, visit the Zynq UltraScale+ MPSoC product page.

Implementation of H.264 or H.265 video compression standards may require a license from third parties as well as the payment of royalties; further information may be obtained from individual patent holders and industry consortia such as MPEG LA and HEVC Advance.

Information about other Xilinx® LogiCORE™ IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

The Encoder and Decoder blocks are compatible with the following standards:

- **ISO/IEC 14496-10:2014(en) Information technology — Coding of audio-visual objects — Part 10:** Advanced Video Coding
- **Recommendation ITU — T H.264 | International Standard ISO/IEC 14496-10:** Advanced video coding for generic audiovisual services
- **Recommendation ITU — T H.265 | International Standard ISO/IEC 23008-2:** High efficiency video coding
- **ISO/IEC 23008-2:2017 Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2:** High efficiency video coding

Performance

The following sections detail the performance characteristics of the H.264/H.265 Video Codec Unit.

Maximum Frequencies

The typical clock frequencies for the target devices are described in the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925). The maximum achievable clock frequency of the system can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the device, using a different version of Xilinx tools and other factors.

Throughput

The VCU supports simultaneous encoding and decoding up to 4K UHD resolution at 60 Hz. This throughput can be a single stream at 4K UHD or can be divided into up to 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.

Resource Utilization

Streams of 4K UHD at 60 Hz consume significant amounts of the bandwidth of the external memory interfaces and significant amounts of the Arm® AMBA® AXI4 bus bandwidth between the Processing System and the Programmable Logic.

For simultaneous encoder and decoder operation (including transcode use cases), consider using both a Xilinx PS Memory Controller and dedicated a Xilinx VCU Memory Controller.

Table 1: **Resource Utilization**

Number of stream	Single stream	Two stream	Four stream
CLB LUT (230400)	16224	18302	21178
CLB Registers (460800)	21098	24175	30339
Carry8 (28800)	1316	1417	1620
F7 Muxes (11520)	105	105	119
F8 Muxes (57600)	0	0	12
BLOCK RAM TILE (312)	8	8	8
DSP's (1728)	0	0	0
HPIOBDIFFINBUF (192)	0	0	0
BITSLICE_RX_TX (416)	0	0	0
GLOBAL CLOCK BUFFERS (544)	0	0	0
PLL (16)	0	0	0
MMCM (8)	0	0	0

Related Information

DDR Memory Footprint Requirements
Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP v1.1

Core Interfaces

The core has the following interfaces:

- Four 128-bit AXI master interfaces to communicate with external memory

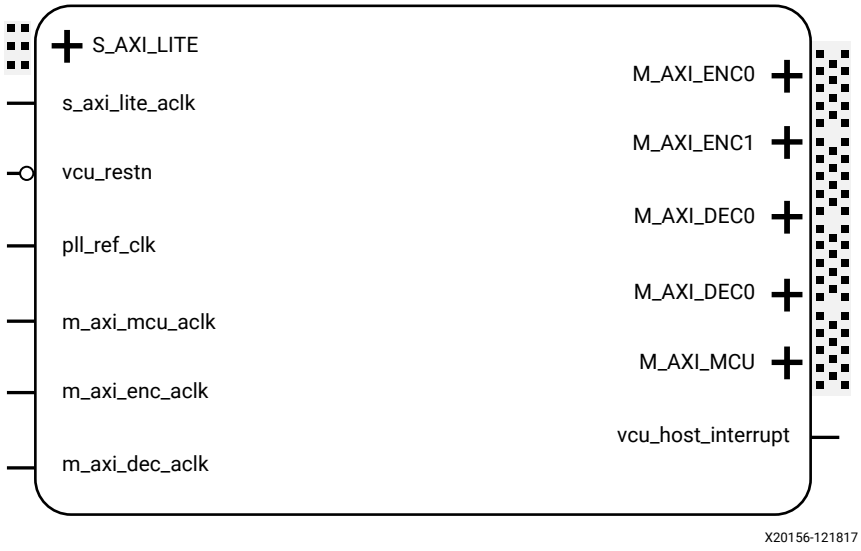
- One 32-bit AXI master interface for control communication
- An AXI4-Lite interface for communicating with the application processing unit (APU)

The four 128-bit AXI master interfaces are used for moving video data into and out of external memory through the PS memory and the PL memory interfaces. Two AXI interfaces are allocated to encoding and two are allocated to decoding.

Port Descriptions

The VCU core top-level signaling interface is shown in the following figure.

Figure 3: **VCU Core Top-Level Signaling Interface**



The following table summarizes the core interfaces.

Table 2: **VCU Interfaces**

Interface Name	Interface Type	Description
M_AXI_ENC0	Memory mapped AXI4 master interface	128-bit memory mapped interface for Encoder block.
M_AXI_ENC1	Memory mapped AXI4 master interface	128-bit memory mapped interface for Encoder block.
M_AXI_DEC0	Memory mapped AXI4 master interface	128-bit memory mapped interface for Decoder block.
M_AXI_DEC1	Memory mapped AXI4 master interface	128-bit memory mapped interface for Decoder block.
M_AXI_MCU	Memory mapped AXI4 master interface	32-bit memory mapped interface for MCU.

Table 2: **VCU Interfaces** *(cont'd)*

Interface Name	Interface Type	Description
S_AXI__LITE	Memory mapped AXI4-Lite slave interface	AXI4-Lite memory mapped interface for external master access.

Related Information

Overview

Common Interface Signals

The following table summarizes the signals which are either shared by, or not part of the dedicated AXI4 interfaces.

Table 3: **VCU Ports**

Port Name	Direction	Description
m_axi_enc_aclk	Input	AXI clock input for M_AXI_VCU_ENCODER0 and M_AXI_VCU_ENCODER1
s_axi_lite_aclk	Input	AXI clock input for S_AXI_PL_VCU_LITE
pll_ref_clk	Input	PLL reference clock input
vcu_resetn	Input	Active-Low reset input from PL
vcu_host_interrupt	Output	Active-High interrupt output from VCU. Can be mapped to PL-PS interrupt pin.
m_axi_dec_aclk	Input	AXI input clock for M_AXI_VCU_DECODER0 and M_AXI_VCU_DECODER1
m_axi_mcu_aclk	Input	Input clock for M_AXI_MCU interface

Register Space

The Zynq UltraScale+ MPSoC VCU soft IP implements registers in the programmable logic. The following table summarizes the soft IP registers. These registers are accessible from the PS via the AXI4-Lite bus.

Table 4: **Soft IP Registers (Video Configuration)**

Register	Address Offset	Width	Type	Definition
VCU_ENCODER_ENABLE	0x41000	32	Ro	1 = Enable 0 = Disable
VCU_DECODER_ENABLE	0x41004	32	Ro	1 = Enable 0 = Disable
VCU_MEMORY_DEPTH	0x41008	32	Ro	Number of entries in Encoder Buffer

Table 4: **Soft IP Registers (Video Configuration)** *(cont'd)*

Register	Address Offset	Width	Type	Definition
VCU_ENC_COLOR_DEPTH	0x4100C	32	Ro	0 = 8 bits per pixel 1 = 8 and 10 bits per pixel
VCU_ENC_VERTICAL_RANGE	0x41010	32	Ro	0 = Low 1 = Medium 2 = High
VCU_ENC_FRAME_SIZE_X	0x41014	32	Ro	Encoder horizontal pixel size
VCU_ENC_FRAME_SIZE_Y	0x41018	32	Ro	Encoder vertical pixel size
VCU_ENC_COLOR_FORMAT	0x4101C	32	Ro	0 = 4:2:0 1 = 4:2:2 2 = 4:0:0
VCU_ENC_FPS	0x41020	32	Ro	Denotes frames per second
VCU_ENC_VIDEO_STANDARD	0x41038	32	Ro	0 = H.265 (HEVC) 1 = H.264 (AVC)
VCU_STATUS	0x4103C	32	Ro	0 = INTERRUPT 1 = POWER_STATUS_VCUINT 2 = POWER_STATUS_VCUAUX 3 = PLL_LOCK_STATUS
VCU_DEC_VIDEO_STANDARD	0x4104C	32	Ro	0 = H.265 (HEVC) 1 = H.264 (AVC)
VCU_DEC_FRAME_SIZE_X	0x41050	32	Ro	Decoder horizontal pixel size
VCU_DEC_FRAME_SIZE_Y	0x41054	32	Ro	Decoder vertical pixel size
VCU_DEC_FPS	0x41058	32	Ro	Decoder frames per second
VCU_BUFFER_B_FRAME	0x4105C	32	Ro	B-frame usage 0 = B-frames disabled 1 = B-frames enabled .
ENC_NUM_CORE	0x4106C	32	Ro	Number of encoders core used for the provided configuration
VCU_PLL_CLK_HI	0x41034	32	Ro	Reports the integer value of PLL clock frequency as set in the Vivado block design. Each unit is 10 kHz. Default: 33.33 MHz
VCU_PLL_CLK_LO	0x41064	32	Ro	Reports the fractional value of PLL clock frequency as set in the Vivado block design. Each unit is 10 kHz. Default: 33.33 MHz

Table 4: **Soft IP Registers (Video Configuration)** *(cont'd)*

Register	Address Offset	Width	Type	Definition
VCU_GASKET_INIT	0x41074	32	Rw	Ensure there is no pending AXI transaction in VCU AXI bus/AXI4-Lite bus before accessing the register. Assert and de-assert `vcu_resetn` signal before accessing the register. Bit 0 is set to 1 to remove gasket isolation after VCUINT_VCU fully ramps, VCCAUX fully ramps, and the PL is programmed Bit 1 is set to 0 to assert reset to VCU. Software needs to de-assert it to 1 for out of resets.

***Note*:** For multi-stream use case, the registers in Table 4: Soft IP Registers (Video Configuration) represent blended values that are input in GUI.

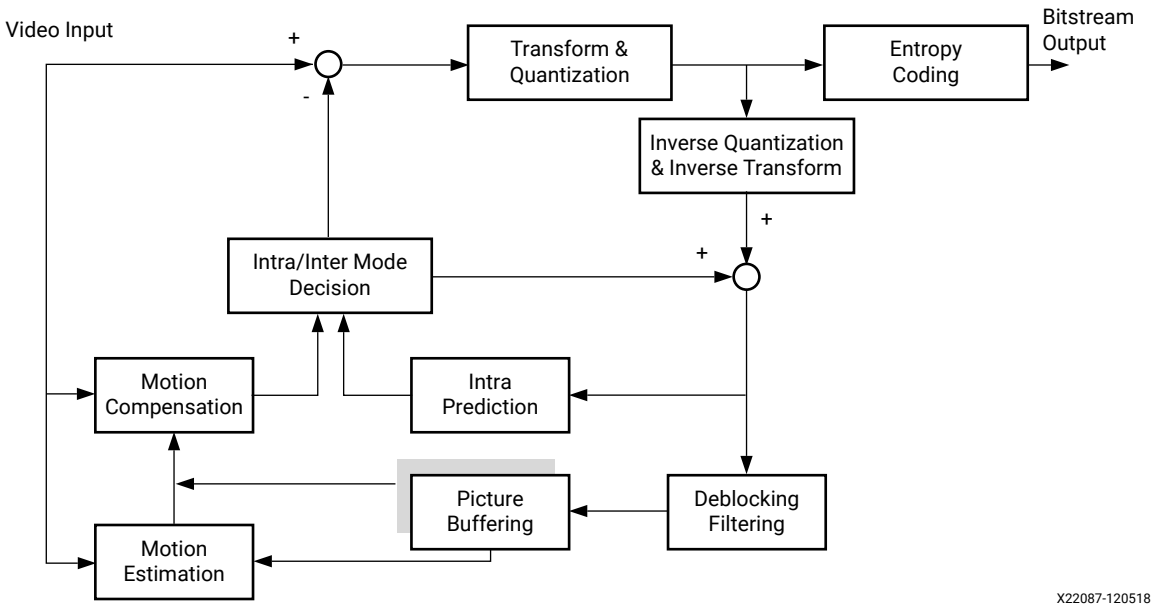
Core Architecture

Encoder Block

The encoder block of the Video Codec Unit (VCU) core is a video encoder engine for processing video streams using the H.265 (ISO/IEC 23008-2 high-efficiency video coding) and H.264 (ISO/IEC 14496-10 advanced video coding) standards. It provides complete support for these standards, including support for 8-bit and 10-bit color depth, 4:2:0, 4:2:2, and 4:0:0 Chroma formats and up to 4K UHD at 60 Hz performance.

The VCU encoder block is shown in the following figure.

Figure 4: **VCU Encoder Block**



Features

The following table describes the VCU Encoder block features.

Table 5: **Encoder Block Features**

Video Coding Feature	H.264	H.265
Performance		
Profiles	Baseline Main High High 10 High 4:2:2 High10 Intra High 4:2:2 Intra	Main Main Intra Main 10 Main 10 Intra Main 4:2:2 10 Main 4:2:2 10 Intra
Levels	Up to 5.2[1]	Up to 5.1 High Tier[1]
Performance at 667 MHz[2] 32 streams at 720×480p at 30 Hz Eight streams at 1920×1080p at 30 Hz Four streams at 1920×1080p at 60 Hz Two streams at 3840×2160p at 30 Hz One stream at 3840×2160p at 60 Hz One stream at 7680×4320p at 15 Hz	Supported	Supported
Configurable resolution: Picture width and height multiple of 8 Minimum size: 128×64 Maximum width or height: 16384 Maximum size: 33.5 megapixel	Supported	Supported
Configurable frame rate	Supported	Supported
Configurable bit rate	Supported	Supported
Coding Tools		
Sample bit depth: 8 bpc, 10 bpc	Supported	Supported
Chroma format: YCbCr 4:2:0, YCbCr 4:2:2, Y-only (monochrome)	Supported	Supported
Slice types: I, P, B	Supported	Supported
Progressive format only	Supported	Supported
Coding block size	16×16 macroblocks	LCU size: 32×32 CU size down to 8×8
Prediction size	Down to 4×4 for intra prediction, down to 8×8 inter prediction	Down to 4×4 for intra prediction, down to 8×8 inter prediction
Transform size	4×4, 8×8	4×4, 8×8, 16×16, 32×32
Intra prediction modes	All intra 4×4, intra 8×8, intra 16×16 modes	All 33 directional modes, planar, DC
Constrained Intra Pred support	Supported	Supported
Motion estimation: 1 reference picture for P slices or 2 reference pictures for B slices	Supported	Supported

Table 5: **Encoder Block Features** *(cont'd)*

Video Coding Feature	H.264	H.265
Motion estimation and compensation: quarter sample interpolation	Supported	Supported
Motion vector prediction modes	All motion vector prediction modes except spatial direct mode and direct_8×8_inference_flag=0	All motion vector prediction/ merge/skip modes
Weighted prediction	Supported	Supported
QP control	Constant per frame, configurable per MB or adaptive per MB	Constant per frame, configurable per LCU or adaptive per CU
Chroma QP offset	Supported	Supported
Scaling lists	Supported	Supported
In-loop deblocking filter	Supported	Supported
Entropy coding	CABAC, CAVLC	CABAC
Configurable CABAC initialization table	Supported	Supported
Slice support	Supported (slices required above 1080p60 performance)	Supported
Dependent slice support	N/A	Supported
Tile support	N/A	Supported (tiles required above 1080p60 performance)

Notes:

1. Support of 8K15 uses a subset of level 6.
2. AVC minimum picture resolution: 80×96; HEVC minimum picture resolution: 128×128.
3. In HEVC, the minimum coding unit is 8x8. As there is no padding in the hardware, some uninitialized pixels can be encoded at the bottom and at the right of a frame if the width and height are not multiple of eight. HVEC allows resolution multiple of 2 to handle interlaced sequence. But, the encoder still encodes up to the above 8x8 aligned resolution. The decoder will then crop down to the real resolution but uninitialized pixels can still generate/ propagate artifacts in the final cropped video. For nonaligned resolution, it is recommended to provide aligned buffer to 8x8 with bottom pixels properly initialized (either to black or with neighborhood pixel value).

The following table summarizes the maximum bit rate achievable for different profile/level combinations.

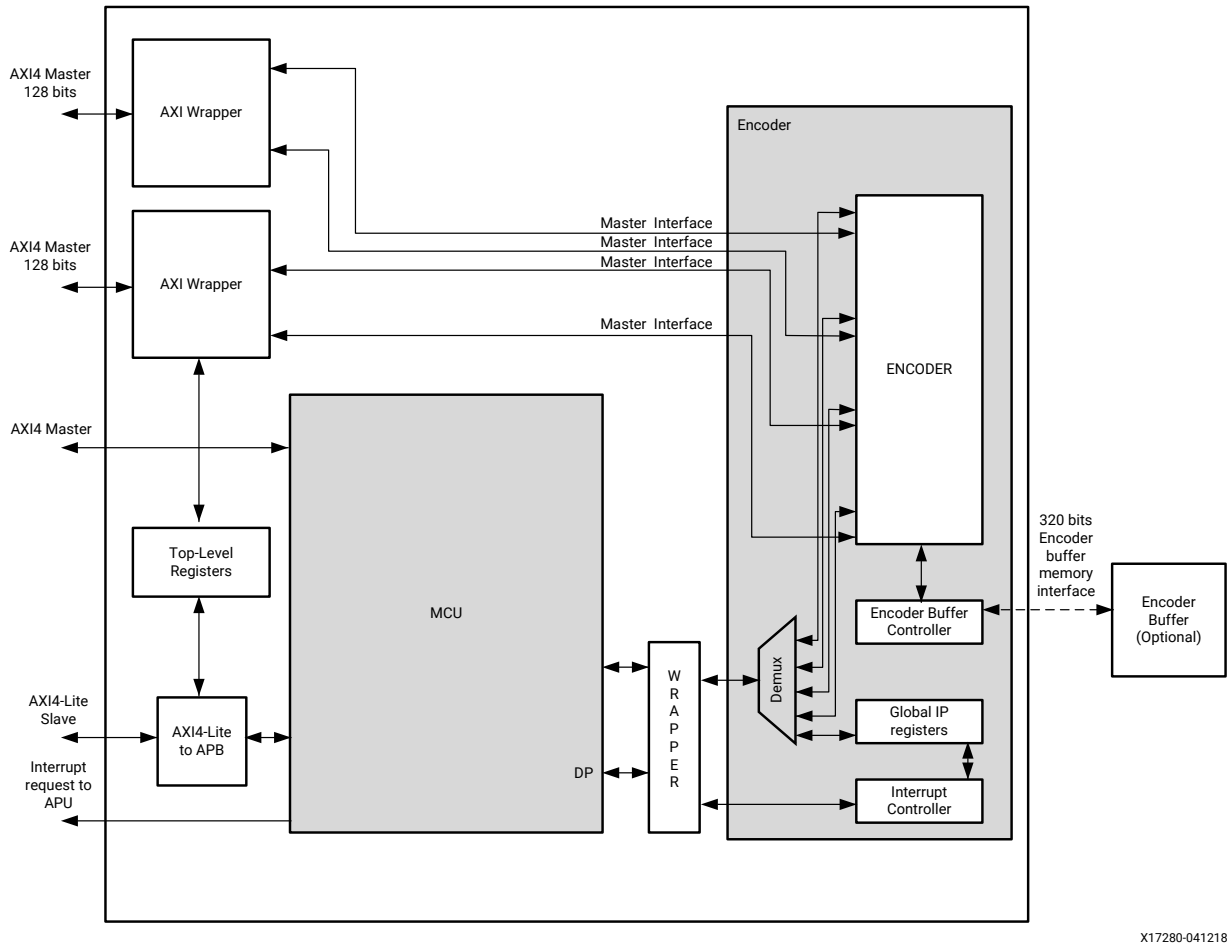
Table 6: **Maximum Bit Rate**

Standard	Level	Profile	Maximum Bit Rate (Mbits/s)
H.264 (AVC)	4.2 (1080p60)	Baseline, Main	50
		High	62.5
		High 10	150 (CAVLC or CABAC Intra only) 67 (CABAC non-Intra only)
		High 4:2:2	200 (CAVLC or CABAC Intra only) 67 (CABAC non-Intra only)
	5.2 (2160p60)	Baseline, Main	240
		High	300 (context-adaptive variable-length coding (CAVLC) or context-adaptive binary arithmetic coding (CABAC) Intra-only) 267 (CABAC non-Intra-only)
		High 10	720 (CAVLC or CABAC Intra-only) 267 (CABAC non-Intra-only)
		High 4:2:2	960 (CAVLC or CABAC Intra-only) 267 (CABAC non-Intra-only)
H.265 (HEVC)	4.1 (1080p60) High Tier	Main, Main 10	50
		Main 4:2:2 10	84
		Main 4:2:2 10 Intra	167
	5.1 (2160p60) High Tier	Main, Main 10	160
		Main 4:2:2 10	267
		Main 4:2:2 10 Intra	533

Functional Description

The following figure shows the top-level interfaces and detailed architecture of the encoder block.

Figure 5: **Detailed Architecture of the Encoder Block**



***Note*:** The AXI4 master interface from the MCU is multiplexed with the corresponding AXI-4 Master interface from the Decoder. The multiplexer output is available at the embedded VCU.

- The encoder block includes the compression engines, control registers, an interrupt controller, and an optional encoder buffer with a memory controller. The encoder buffer is connected to UltraRAM or block RAM in the programmable logic and enabled using registers.
- The encoder block is controlled by a microcontroller unit (MCU) subsystem, including a 32-bit MCU with a 32 KB instruction cache, a 1 KB data cache, and a 32 KB local SRAM.
- A 32-bit AXI4-Lite slave interface is used by the APU to control the MCU for the configuration of encoder parameters, to start/stop processing, to get status and to get results.
- Two 128-bit AXI4 master interfaces are used to fetch video input data, load and store intermediate data, store compressed data back to memory.
- A 32-bit AXI4 master interface is used to fetch the MCU software and load/store additional MCU data.

The VCU control software can change encoding parameters and even change between H.264 and H.265 encoding dynamically; however, the available memory and bandwidth must be selected to support the worst case needed by the application. Use the VCU GUI to explore bandwidth requirements.

Interfaces and Ports

Applications that use the encoder block must connect all encoder ports (ports with names beginning with `m_axi_enc`). The following table shows the list of ports/interfaces of the top-level encoder block.

Table 7: **Encoder Ports**

Name	Size (bits)	Dir	Description
Clocks and Resets			
pll_ref_clk	1	Input	Reference clock to the VCU PLL from PL
m_axi_enc_aclk	1	Input	Memory interface Encoder clock
m_axi_dec_aclk	1	Input	Memory interface Decoder clock
s_axi_lite_aclk	1	Input	AXI4-Lite clock
m_axi_mcu_aclk	1	Input	VCU MCU AXI interface clock from PL
vcu_resetn	1	Input	Active-Low. VCU reset from PL
VCU-Interrupt (s_axi_lite_aclk)			
vcu_host_interrupt	1	Output	Active-High. Interrupt from VCU to PS
VCU Encoder Block, 128-bit AXI Master Interface 0 (m_axi_enc_aclk domain)			
vcu_pl_enc_araddr0	44	Output	AXI Master read address bus for interface 0
vcu_pl_enc_arburst0	2	Output	AXI Master read burst type signal
vcu_pl_enc_arid0	4	Output	AXI Master read burst ID for interface 0
vcu_pl_enc_arlen0	8	Output	AXI Master read burst length for interface 0
pl_vcu_enc_arready0	1	Input	AXI Master read address ready for interface 0
vcu_pl_enc_arsize0	3	Output	AXI Master read interface size for interface 0
vcu_pl_enc_arvalid0	1	Output	AXI Master read address valid for interface 0
vcu_pl_enc_awaddr0	44	Output	AXI Master write address for interface 0
vcu_pl_enc_awburst0	2	Output	AXI Master write burst type for interface 0
vcu_pl_enc_awid0	4	Output	AXI Master write burst ID for interface 0
vcu_pl_enc_awlen0	8	Output	AXI Master write burst length for interface 0
pl_vcu_enc_awready0	1	Input	AXI Master write address ready for interface 0
vcu_pl_enc_awsize0	3	Output	AXI Master write burst size for interface 0
vcu_pl_enc_awvalid0	1	Output	AXI Master write address valid for interface 0
pl_vcu_enc_bresp0	2	Input	AXI Master write response for interface 0
vcu_pl_enc_bready0	1	Output	AXI Master write response ready for interface 0
pl_vcu_enc_bvalid0	1	Input	AXI Master write response valid for interface 0
pl_vcu_enc_bid0	4	Input	AXI Master write response ID for interface 0

Table 7: **Encoder Ports** *(cont'd)*

Name	Size (bits)	Dir	Description
pl_vcu_enc_rdata0	128	Input	AXI Master read data for interface 0
pl_vcu_enc_rid0	4	Input	AXI Master read ID signal for interface 0
pl_vcu_enc_rlast0	1	Input	AXI Master read last signal for interface 0
vcu_pl_enc_rready0	1	Output	AXI Master read ready signal for interface 0
Pl_vcu_enc_rresp0	2	Input	AXI Master read response signal for interface 0
pl_vcu_enc_rvalid0	1	Input	AXI Master read valid signal for interface 0
vcu_pl_enc_wdata0	128	Output	AXI Master write data for interface 0
vcu_pl_enc_wlast0	1	Output	AXI Master write last signal for interface 0
pl_vcu_enc_wready0	1	Input	AXI Master write ready signal for interface 0
vcu_pl_enc_wvalid0	1	Output	AXI Master write valid signal for interface 0
vcu_pl_enc_awprot0	1	Output	AXI Master write protection signal for interface 0, controlled from SLCR
vcu_pl_enc_arprot0	1	Output	AXI Master read protection signal for interface 0, controlled from SLCR
vcu_pl_enc_awqos0	4	Output	AXI Master write QOS signal for interface 0, controlled from SLCR
vcu_pl_enc_arqos0	4	Output	AXI Master read QOS signal for interface 0, controlled from SLCR
vcu_pl_enc_awcache0	4	Output	AXI Master write cache signal for interface 0, controlled from SLCR
vcu_pl_enc_arcache0	4	Output	AXI Master read cache signal for interface 0, controlled from SLCR
VCU Encoder Block, 128-bit AXI Master Interface 1 (m_axi_enc_aclk domain)			
vcu_pl_enc_araddr1	44	Output	AXI Master read address bus for interface 1
vcu_pl_enc_arburst1	2	Output	AXI Master read burst type signal
vcu_pl_enc_arid1	4	Output	AXI Master read burst ID for interface 1
vcu_pl_enc_arlen1	8	Output	AXI Master read burst length for interface 1
pl_vcu_enc_arready1	1	Input	AXI Master read address ready for interface 1
vcu_pl_enc_arsize1	3	Output	AXI Master read interface size for interface 1
vcu_pl_enc_arvalid1	1	Output	AXI Master read address valid for interface 1
vcu_pl_enc_awaddr1	44	Output	AXI Master write address for interface 1
vcu_pl_enc_awburst1	2	Output	AXI Master write burst type for interface 1
vcu_pl_enc_awid1	4	Output	AXI Master write burst ID for interface 1
vcu_pl_enc_awlen1	8	Output	AXI Master write burst length for interface 1
pl_vcu_enc_awready1	1	Input	AXI Master write address ready for interface 1
vcu_pl_enc_awsize1	3	Output	AXI Master write burst size for interface 1
vcu_pl_enc_awvalid1	1	Output	AXI Master write address valid for interface 1
Pl_vcu_enc_bresp1	2	Input	AXI Master write response for interface 1
vcu_pl_enc_bready1	1	Output	AXI Master write response ready for interface 1
pl_vcu_enc_bvalid1	1	Input	AXI Master write response valid for interface 1

Table 7: **Encoder Ports** *(cont'd)*

Name	Size (bits)	Dir	Description
pl_vcu_enc_bid1	4	Input	AXI Master write response ID for interface 1
pl_vcu_enc_rdata1	128	Input	AXI Master read data for interface 1
pl_vcu_enc_rid1	4	Input	AXI Master read ID signal for interface 1
pl_vcu_enc_rlast1	1	Input	AXI Master read last signal for interface 1
vcu_pl_enc_rready1	1	Output	AXI Master read ready signal for interface 1
Pl_vcu_enc_rresp1	2	Input	AXI Master read response signal for interface 1
pl_vcu_enc_rvalid1	1	Input	AXI Master read valid signal for interface 1
vcu_pl_enc_wdata1	128	Output	AXI Master write data for interface 1
vcu_pl_enc_wlast1	1	Output	AXI Master write last signal for interface 1
pl_vcu_enc_wready1	1	Input	AXI Master write ready signal for interface 1
vcu_pl_enc_wvalid1	1	Output	AXI Master write valid signal for interface 1
vcu_pl_enc_awprot1	1	Output	AXI Master write protection signal for interface 1, controlled from SLCR
vcu_pl_enc_arprot1	1	Output	AXI Master read protection signal for interface 1, controlled from SLCR
vcu_pl_enc_awqos1	4	Output	AXI Master write QOS signal for interface 1, controlled from SLCR
vcu_pl_enc_arqos1	4	Output	AXI Master read QOS signal for interface 1, controlled from SLCR
vcu_pl_enc_awcache1	4	Output	AXI Master write cache signal for interface 1, controlled from SLCR
vcu_pl_enc_arcache1	4	Output	AXI Master read cache signal for interface 1, controlled from SLCR
VCU Encoder- 32-bit AXI Master MCU Instruction and Data Cache Interface			
vcu_pl_mcu_m_axi_ic_dc_araddr	44	Output	AXI Master read address bus for MCU
vcu_pl_mcu_m_axi_ic_dc_arburst	2	Output	AXI Master read burst type signal
vcu_pl_mcu_m_axi_ic_dc_arcache	4	Output	AXI Master read cache for MCU
vcu_pl_mcu_m_axi_ic_dc_arid	3	Output	AXI Master read burst ID for MCU
vcu_pl_mcu_m_axi_ic_dc_arlen	8	Output	AXI Master read burst length for MCU
vcu_pl_mcu_m_axi_ic_dc_arlock	1	Output	AXI Master read lock for MCU
vcu_pl_mcu_m_axi_ic_dc_arprot	3	Output	AXI Master read protection signal for MCU
vcu_pl_mcu_m_axi_ic_dc_arqos	4	Output	AXI Master read QoS for MCU
pl_vcu_mcu_m_axi_ic_dc_arready	1	Input	AXI Master read address ready for MCU
vcu_pl_mcu_m_axi_ic_dc_arsize	3	Output	AXI Master read address size for MCU
vcu_pl_mcu_m_axi_ic_dc_arvalid	1	Output	AXI Master read address valid for MCU
vcu_pl_mcu_m_axi_ic_dc_awaddr	44	Output	AXI Master write address for MCU
vcu_pl_mcu_m_axi_ic_dc_awburst	2	Output	AXI Master write burst type for MCU
vcu_pl_mcu_m_axi_ic_dc_awcache	4	Output	AXI Master write cache for MCU
vcu_pl_mcu_m_axi_ic_dc_awid	3	Output	AXI Master write address ID for MCU
vcu_pl_mcu_m_axi_ic_dc_awlen	8	Output	AXI Master write burst length for MCU

Table 7: **Encoder Ports** *(cont'd)*

Name	Size (bits)	Dir	Description
vcu_pl_mcu_m_axi_ic_dc_awlock	1	Output	AXI Master write lock for MCU
vcu_pl_mcu_m_axi_ic_dc_awprot	3	Output	AXI Master write protection for MCU
vcu_pl_mcu_m_axi_ic_dc_awqos	4	Output	AXI Master write QoS for MCU
pl_vcu_mcu_m_axi_ic_dc_awready	1	Input	AXI Master write address ready signal for MCU
vcu_pl_mcu_m_axi_ic_dc_awsize	3	Output	AXI Master write burst size signal for MCU
vcu_pl_mcu_m_axi_ic_dc_awvalid	1	Output	AXI Master write address valid signal for MCU
pl_vcu_mcu_m_axi_ic_dc_bid	3	Input	AXI Master write response ID for MCU
vcu_pl_mcu_m_axi_ic_dc_bready	1	Output	AXI Master write response ready signal for MCU
pl_vcu_mcu_m_axi_ic_dc_bresp	2	Input	AXI Master write response for MCU
pl_vcu_mcu_m_axi_ic_dc_bvalid	1	Input	AXI Master write response valid signal for MCU
pl_vcu_mcu_m_axi_ic_dc_rdata	32	Input	AXI Master read data signal for MCU
pl_vcu_mcu_m_axi_ic_dc_rid	3	Input	AXI Master read ID signal for MCU
pl_vcu_mcu_m_axi_ic_dc_rlast	1	Input	AXI Master read last signal for MCU
vcu_pl_mcu_m_axi_ic_dc_rready	1	Output	AXI Master read ready signal for MCU
pl_vcu_mcu_m_axi_ic_dc_rresp	2	Input	AXI Master read response signal for MCU
pl_vcu_mcu_m_axi_ic_dc_rvalid	1	Input	AXI Master read valid signal for MCU
vcu_pl_mcu_m_axi_ic_dc_wdata	32	Output	AXI Master write data signal for MCU
vcu_pl_mcu_m_axi_ic_dc_wlast	1	Output	AXI Master write last signal for MCU
pl_vcu_mcu_m_axi_ic_dc_wready	1	Input	AXI Master write ready signal for interface 1
vcu_pl_mcu_m_axi_ic_dc_wstrb	4	Output	AXI Master write strobe signal
vcu_pl_mcu_m_axi_ic_dc_wvalid	1	Output	AXI Master write valid signal for MCU
AXI4-Lite Slave Interface (s_axi_lite_aclk domain)			
pl_vcu_awaddr_axi_lite	20	Input	AXI4-Lite write address bus
pl_vcu_awprot_axi_lite	3	Input	AXI4-Lite write protection signal
pl_vcu_awvalid_axi_lite	1	Input	AXI4-Lite write address valid signal
vcu_pl_awready_axi_lite	1	Output	AXI4-Lite write address ready signal
pl_vcu_wdata_axi_lite	32	Input	AXI4-Lite write data channel
pl_vcu_wstrb_axi_lite	4	Input	AXI4-Lite write strobe signal
pl_vcu_wvalid_axi_lite	1	Input	AXI4-Lite write data valid signal
vcu_pl_wready_axi_lite	1	Output	AXI4-Lite write ready signal
vcu_pl_bresp_axi_lite	2	Output	AXI4-Lite write response channel
vcu_pl_bvalid_axi_lite	1	Output	AXI4-Lite write response valid signal
pl_vcu_bready_axi_lite	1	Input	AXI4-Lite write response ready signal
pl_vcu_araddr_axi_lite	20	Input	AXI4-Lite read address channel
pl_vcu_arprot_axi_lite	3	Input	AXI4-Lite read channel protection signal
pl_vcu_arvalid_axi_lite	1	Input	AXI4-Lite read address valid signal
vcu_pl_arready_axi_lite	1	Output	AXI4-Lite read address ready signal
vcu_pl_rdata_axi_lite	32	Output	AXI4-Lite read data bus

Table 7: **Encoder Ports** *(cont'd)*

Name	Size (bits)	Dir	Description
vcu_pl_rresp_axi_lite	2	Output	AXI4-Lite read response signal
vcu_pl_rvalid_axi_lite	1	Output	AXI4-Lite read data valid signal
pl_vcu_rready_axi_lite	1	Input	AXI4-Lite read data ready signal
VCU Encoder Buffer Interface			
Vcu_pl_enc_al_l2c_rvalid	1	Output	Read data valid
Pl_vcu_enc_al_l2c_rready	1	Input	Read data ready
Vcu_pl_enc_al_l2c_addr	17	Output	Address
Pl_vcu_enc_al_l2c_rdata	320	Input	Read data
Vcu_pl_enc_al_l2c_wvalid	1	Output	Write data valid
Vcu_pl_enc_al_l2c_wdata	320	Output	Write data

Clocking

Refer to Clocking and Resets for more information on clocking.

Reset

Refer to Clocking and Resets for more information on resets.

MCU Subsystem

The Encoder block includes an embedded MCU that runs the MCU firmware and controls the hardware Encoder core. Refer to Microcontroller Unit Overview for more information on the MCU.

Data Path

The encoder block has two 128-bit AXI4 master interfaces to fetch video data from external DDR memory attached to either the Processing System (PS) or the Programmable Logic (PL).

The data fetched from memory includes:

- Source frame pixels
- Reference frame pixels
- Reference frame motion vectors
- Slice/frame parameters: lambda table, scaling lists, QP control, QP table
- Residual data (H.264 only)

The data written to memory includes:

- Residual data (H.264 only)
- Reconstructed frame pixels
- Reconstructed frame motion vectors
- Encoded bitstream

Control Path

The MCU slave interface is accessed once per frame by the APU, which sends a frame-level command to the IP core. This interface does not require a fast data path. Interrupts are triggered at frame level to wake up the APU at the end of each frame processing. These commands are processed by the embedded MCUs, which generate slice- and tile-level commands to the video encoder hardware. For more information, refer to Microcontroller Unit Overview.

Encoder Buffer

The buffer memory controller of the encoder block manages the read and write access to the encoder buffer, which stores pixel data from the reference frames. It pre-fetches data blocks from the reference frames in the system memory and stores them in the encoder buffer. The encoder buffer stores Luma and Chroma pixels from the reference frames so that they are present in the buffer when needed by the encoder. The encoder buffer must be one contiguous memory access (CMA) buffer and should be aligned to a 32-byte boundary. Refer to the *Zynq UltraScale+ MPSoC Data Sheet: Overview* (DS891) to see the device memory available per EV device.

Calculate the system bandwidth in total derated based on memory controller efficiency for the required access pattern. Enable the Encoder Buffer if the calculated bandwidth is insufficient.

The optional encoder buffer can be used to reduce the memory bandwidth. This option can slightly reduce the video quality. See the CacheLevel2 in Table 114: Encoder Settings Parameters for more information. Aside from the size, there are no user controls for tuning the Encoder buffer usage.

***Note*:** To enable the Encoder buffer, pass the prefetch-buffer parameter into the GStreamer pipeline that utilizes the hardware. For example:

```
gst-launch-1.0 videotestsrc ! omxh265enc prefetch-buffer=true ! fakesink
```

Table 8: **Available Device Memory**

	ZU4EV	ZU5EV	ZU7EV
Block RAM (MB)	0.56	0.63	1.37
UltraRam (MB)	1.68	2.25	3.37

IMPORTANT! *Using the Encoder block buffer reduces the external DDR memory bandwidth requirement. Refer to Customizing and Generating the Core for more information regarding memory bandwidth requirements.*

The required encoder buffer memory size for 4:2:0 and 4:2:2 sampling formats are shown in the following tables.

Table 9: **Encoder Buffer Memory Requirements for 4:2:0 Sampling**

Encoder Configuration	1920×1080	3840×2160	
	8 bpc	8 bpc	10 bpc
B-frame=NONE Motion Vector Range=LOW	105 KB	291 KB	364 KB
B-frame=STANDARD Motion Vector Range=MEDIUM	395 KB	1,128 KB	1,410 KB
B-frame=HIERARCHICAL Motion Vector Range=HIGH	761 KB	2,250 KB	2,813 KB

Table 10: **Encoder Buffer Memory Requirements for 4:2:2 Sampling**

Encoder Configuration	1920×1080	3840×2160	
	8 bpc	8 bpc	10 bpc
B-frame=NONE Motion Vector Range=LOW	139 KB	388 KB	485 KB
B-frame=STANDARD Motion Vector Range=MEDIUM	526 KB	1,504 KB	1,880 KB
B-frame=HIERARCHICAL Motion Vector Range=HIGH	1,014 KB	3,000 KB	3,750 KB

***Note*:** To find the most precise frame buffer size, use the **Advance Configuration → Resource Summary** menu option in the VCU GUI.

Encoder Buffer Requirements

The encoder input and output requirements are shown in the following table.

Table 11: **Encoder Buffer Requirements**

Requirement	Description
Input Buffer	
No of Buffers	For AVC: 2 + no. of B Frames(num-B). For HEVC: 1 + no. of B Frames(num-B). ***Note*:** In look-ahead mode, add this value by number of look-ahead frames.
Contiguous	Yes
Alignment	32

Table 11: **Encoder Buffer Requirements** *(cont'd)*

Requirement	Description
Size	stride * slice-height * chroma-mode. stride should be 32-aligned. slice-height should be 16-aligned for AVC and 32-aligned for HEVC. ***Note*:** It works with a slice-height 8-aligned, however hardware's harmless requests may be outside of the buffer.
Output Buffer	
No of Buffers	For AVC: 2 + no. of B-Frames(num-B). For HEVC: 1 + no. of B-Frames (num-B). ***Note*:** In the subframe mode, multiply this value by num-slices.
Contiguous	Yes
Alignment	32
Size	Use below API to get size. AL_GetMitigatedMaxNalSize (AL_TDimension tDim, AL_EChromaMode eMode, int iBitDepth)

***Note*:** Because the VCU uses multiple internal encoder engines, it is not possible to reduce the output buffer requirements.

Memory Requirements

The VCU software reads the total available encoder buffer size from LogiCORE registers (the values can be in the GUI, after the settings) along with the value of maximum number of cores based on the settings you select (resolution and fps). The memory allocated by the software is calculated using the total encoder buffer size divided by the maximum number of cores. If this value is inadequate, no channel is created.

***Note*:** The value of 4kp30 encoder buffer is not half of 4kp60.

The 4kp30 encoder buffer requires lesser space than the 4kp60 because it uses two cores and four cores, respectively. Two tiles are encoded in parallel for 4kp30 whereas for 4kp60, four tiles are encoded in parallel. To allocate encoder buffer, the most demanding use case is entered in the Vivado IDE, which computes and provides the maximum number of cores used in that use case (4 for 4K60) to the driver. For each core, the firmware allocates a static encoder buffer size equal to the total size divided by the maximum number of cores. When a channel is started, the firmware computes the required number of cores (for example, two cores for 4K30) and tries to use the available encoder buffer size for these cores.

The following example uses these values: HEVC, 4:2:0 8-bit, B-frames, Low MV range, and single stream. The requirement of encoder buffer is 696 KB for 4kp60 and 504 KB for 4kp30.

- Set the GUI settings to: 4:2:0 8-bit, B-frames, Low MV range, and 4Kp60. The VCU software gets the available encoder buffer size = 696 KB, max-number-of-cores = 4.
  - Running the 4kp30 use case with same design might not work. A channel creation error might occur because 4kp30 requires two encoder cores and therefore, the available encoder buffer size is calculated as 696/2=348 Kb but 348 Kb is not enough to run the 4kp30 use case.
- The same use case works if you override the setting number-cores=4, in the software or the application, because the encoder buffer attached to each core is used in that case.

Based on the use case, the two options to configure are:

- Two 4Kp30 streams are defined in the GUI as the case with highest usage.
- Number of cores for each 4Kp30 must be forced to four (this forces time sharing for the four cores and change the scheduling of the two channels while reducing the total amount of encoder buffer).

The first option is preferred to avoid exposing core management and to avoid additional encoding constraints. The second option may be preferred if the PL memory optimization is an important requirement. The following table shows the possible combinations:

Table 12: **HEVC 4:2:0, 8-bit Depth, with B-Frames enabled, Low MV Range**

Use Case	Codec	L2 Cache (KB)	Comments
1080p60	AVC/HEVC	228	1 enc core
1080p60 - 4 streams	AVC/HEVC	912	1 enc core per stream
4kp30 - 1 stream	AVC	453	2 enc cores
4kp30 - 1 stream	HEVC	504	2 enc cores
4kp30 - 2 streams	AVC	906	2 enc cores per stream
4kp30 - 2 streams	HEVC	1008	2 enc cores per stream
4kp60 - 1 stream	AVC	582	4 enc cores
4kp60 - 1 stream	HEVC	696	4 enc cores

- The automatic computation of the required size becomes cumbersome if you want to support multiple use cases that do not have the same max-number-of-cores, but in basic cases (for example, 4x1080p60 or 2x4k30 or 1x4k60), the worst case encoder buffer size and the worst case max-number-of-cores must be provided. Choose multi-stream use case in GUI to avoid such failures.

  Note: You cannot set the value of num-cores to maximum in the sub-frame latency mode. It is recommended that you leave num-cores calculation as Auto to VCU firmware and adjust GUI settings to support multiple use cases.

DDR Memory Footprint Requirements

DDR memory buffer size depends on the following:

- Video resolution
- Chroma sub-sampling
- Color depth
- Coding standard

The number of buffers depend on the following:

- Coding standard – H.264 or H.265
- Group of Picture (GOP) pattern

The following table shows the worst-case memory footprint for various encoding schemes.

Table 13: **Encoder Block Memory Footprint**

Examples with Two B-Frames	720p			1080p			2160p		
	Buffers	Per Buffer	Total	Buffers	Per Buffer	Total	Buffers	Per Buffer	Total
Source Frame	5	2.3 MB	12 MB	5	5.3 MB	27 MB	5	21.1 MB	106 MB
Reference Frames	3	2.2 MB	7 MB	3	5.0 MB	15 MB	3	19.9 MB	60 MB
Reconstructed Frame	1	2.2 MB	3 MB	1	5.0 MB	5 MB	1	19.9 MB	20 MB
Intermediate Buffers	2	0.0 MB	0 MB	2	0.0 MB	0 MB	2	41.0 MB	83 MB
Motion Vector Buffer	4	0.1 MB	1 MB	4	0.3 MB	1 MB	4	1.0 MB	4 MB
Bitstream Buffer	2	1.1 MB	3 MB	2	2.5 MB	5 MB	2	9.9 MB	20 MB
Other Buffers	1	0.0 MB	1 MB	1	0.0 MB	1 MB	1	0.1 MB	1 MB
Total			27 MB			54			294

CMA Size Requirements

The following table contains theoretical contiguous memory access (CMA) buffer requirements for the VCU encoder/decoder based on resolution and format. The sizes below correspond to one instance of the encoder or decoder. Multiply these by number of streams for multistream use cases. Other elements such as kmssink/v4l2src typically increase the CMA requirements by an additional 10 to 15%.

Table 14: **VCU Encoder CMA Requirements**

Resolution	4:2:2 10-bit AVC[1]	4:2:2 10-bit HEVC[2]	4:2:2: 8-bit AVC[1]	4:2:2: 8-bit HEVC[2]	4:2:0 10-bit AVC[1]	4:2:0 10-bit HEVC[2]	4:2:0 8-bit AVC[1]	4:2:0 8-bit HEVC[2]
3840×2160 (MB)[3]	294	199	248	155	243	151	208	117
1920×1080 (MB)	54	52	42	40	42	40	32	31

Table 14: **VCU Encoder CMA Requirements** *(cont'd)*

Resolution	4:2:2 10-bit AVC[1]	4:2:2 10-bit HEVC[2]	4:2:2: 8-bit AVC[1]	4:2:2: 8-bit HEVC[2]	4:2:0 10-bit AVC[1]	4:2:0 10-bit HEVC[2]	4:2:0 8-bit AVC[1]	4:2:0 8-bit HEVC[2]
1280×720 (MB)	27	26	21	20	20	19	17	16

Notes:

1. AVC requires extra intermediate buffers when it is using multiple-cores. Unlike HVEC, the AVC standard does not support Tile processing, so to exploit data processing parallelism it requires two intermediate buffers.
2. VCU AVC Encoder uses multiple cores when resolution is >= 1080p60, that is reason for having ~100 MB delta between HEVC and AVC CMA requirements for 3840x2160.
3. Includes memory for two intermediate buffers.

Memory Footprint Calculation

An approximate formula for each buffer type can be used to derive the total memory requirement per use case.

Table 15: **Buffer-Formula Mapping**

Buffer	Formula
Source Frame	height x width x bit depth (10-bit = 4/3, 8-bit = 1) x chroma format (For 4:2:2 = 2, 4:2:0 = 1.5, 4:0:0 = 1)
Reference Frame	height x width x bit depth (10-bit = 10/8, 8-bit = 1) x chroma format (For 4:2:2 = 2, 4:2:0 = 1.5, 4:0:0 = 1)
Reconstructed Frame	height x width x bit depth (10-bit = 10/8, 8-bit = 1) x chroma format (For 4:2:2 = 2, 4:2:0 = 1.5, 4:0:0 = 1)
Intermediate Buffers	height/16 x width/16 x 1328 (Requires by AVC when using multi-cores)
Motion Vector Buffer	height/16 x width/16 x Codec (For AVC = 32, For HEVC = 16)
Bitstream Buffer	height x width x bit depth (10-bit = 10/8, 8-bit = 1) x color format (For 4:2:2 = 2, 4:2:0 = 1.5, 4:0:0 = 1)/ Codec (For AVC = 2, For HEVC = 4)
Other Buffers	25664 + (height x width)/Codec (For AVC= 256, HEVC=1024)

The following example is for a multistream use case: 4 1080p30 AVC, 4:2:2 chroma format, 10-bit depth.

Table 16: **Multi-stream Use Case**

Buffers	No. of Buffers	Single Stream(1080p30)	Multi-Stream (4 1080p30)
Source Frame	5	27 MB	105 MB
Reference Frame	3	15 MB	60 MB
Reconstructed Frame	1	5 MB	20 MB
Intermediate Buffers	2	0 MB	0 MB
Motion Vector Buffer	4	1 MB	4 MB
Bitstream Buffer	2	5 MB	20 MB
Other Buffers	1	1 MB	4 MB

Table 16: **Multi-stream Use Case** *(cont'd)*

Buffers	No. of Buffers	Single Stream(1080p30)	Multi-Stream (4 1080p30)
Total		54 MB	217 MB

Memory Bandwidth

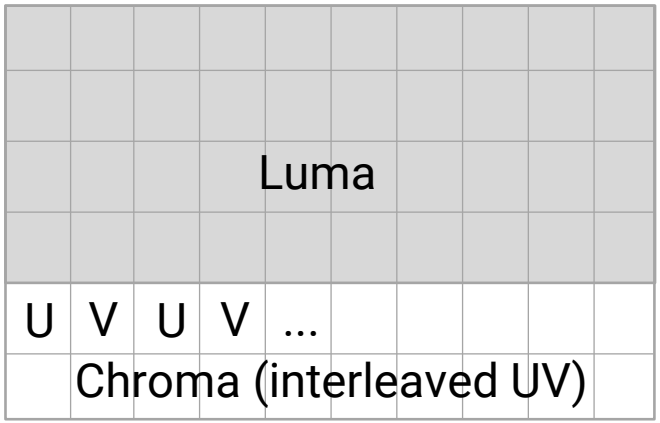
Xilinx recommends using the fastest DDR4 memory interface possible. Specifically, the 8x8-bit memory interface is more efficient than 4x16-bit memory interface because the x8 mode has four bank groups, whereas the x16 mode has only two; and DDR4 allows for simultaneous bank group access. For more information, see Xilinx Answer Record 71209.

Source Frame Format

The source frame buffer contains the input frame pixels. It contains two parts: luminance pixels (Luma) followed by chrominance pixels (Chroma). Luma pixels are stored in pixel raster scan order, shown in the following figure. Chroma pixels are stored in an U/V-interleaved pixel raster scan order. Therefore, the Chroma portion is half the size of the Luma portion when using a 4:2:0 format and the same size as the Luma portion when using a 4:2:2 format. The encoder picture buffer must be one contiguous memory region.

***Note*:** The VCU accepts semi-planar data.

Figure 6: **Frame Layout**



Two packing formats are supported in external memory: 8 bits per component or 10 bits per component, shown in the following tables. The 8-bit format can only be used for an 8-bit component depth and the 10-bit format can only be used for a 10-bit component depth.

255 248	...	31 24	23 16	15 8	7 0
$Y_{x+31,y}$	...	$Y_{x+3,y}$	$Y_{x+2,y}$	$Y_{x+1,y}$	$Y_{x,y}$
...(all Luma in pixel raster scan order)...					
255 248	247 240 ...	31 24	23 16	15 8	7 0
$V_{x+15,y}$	$U_{x+15,y}$ ...	$V_{x+1,y}$	$U_{x+1,y}$	$V_{x,y}$	$U_{x,y}$

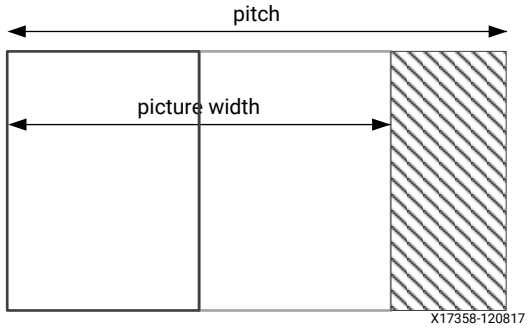
...(all interleaved Chroma in pixel raster scan order)...

255	254	253 244		31	30	29 20	19 10	9 0
0	0	$V_{x+23,y}$		0	0	$Y_{x+2,y}$	$Y_{x+1,y}$	$Y_{x,y}$
...(all Luma in pixel raster scan order)...								
255	254	253 244	... 63 62	61 52	51 42	41 32	31 30	29 20	19 10	9 0
0	0	$V_{x+11,y}$	... 0 0	$V_{x+2,y}$	$U_{x+2,y}$	$V_{x+1,y}$	0 0	$U_{x+1,y}$	$V_{x,y}$	$U_{x,y}$
...(all interleaved Chroma in pixel raster scan order)...								

The encoder buffer must be one contiguous memory region and should be aligned to a 32-byte boundary.

The frame buffer width (pitch) may be larger than the frame width. When the pitch is greater than the frame width, pixels in each line beyond the picture width are ignored, as illustrated in the following figure.

Figure 7: **Frame Buffer Pitch**



***Note*:** Encoder input buffers width and height should be in multiple of 32. Decoder output buffers width is in multiple of 256-byte. Height is in multiples of 64. For example, for 1920x1080 resolution, the decoder output is 2048x1088.

Encoder Block Register Overview

The following table lists the encoder block registers. For additional information, see *Zynq UltraScale+ Device Register Reference* (UG1087).

Table 19: **Encoder Registers**

Register	Offset	Width	Type	Reset Value	Description
MCU_RESET	0x9000	32	mixed[1]	0x00000000	MCU Subsystem Reset
MCU_RESET_MODE	0x9004	32	mixed[1]	0x00000001	MCU Reset Mode
MCU_STA	0x9008	32	mixed[1]	0x00000000	MCU Status
MCU_WAKEUP	0x900C	32	mixed[1]	0x00000000	MCU Wake-up
MCU_ADDR_OFFSET_IC0	0x9010	32	rw	0x00000000	MCU Instruction Cache Address Offset 0
MCU_ADDR_OFFSET_IC1	0x9014	32	rw	0x00000000	MCU Instruction Cache Address Offset 1
MCU_ADDR_OFFSET_DC0	0x9018	32	rw	0x00000000	MCU Data Cache Address Offset 0
MCU_ADDR_OFFSET_DC1	0x901C	32	rw	0x00000000	MCU Data Cache Address Offset 1
ITC_MCU_IRQ	0x9100	32	mixed[1]	0x00000000	MCU Interrupt Trigger
ITC_CPU_IRQ_MSK	0x9104	32	rw	0x00000000	CPU Interrupt Mask
ITC_CPU_IRQ_CLR	0x9108	32	mixed[1]	0x00000000	CPU Interrupt Clear
ITC_CPU_IRQ_STA	0x910C	32	mixed[1]	0x00000000	CPU Interrupt Status
AXI_BW	0x9204	32	rw	0x00000000	AXI Bandwidth Measurement Window
AXI_ADDR_OFFSET_IP	0x9208	32	rw	0x00000000	Video Data Address Offset
AXI_RBW0	0x9210	32	ro	0x00000000	AXI Read Bandwidth Status 0
AXI_RBW1	0x9214	32	ro	0x00000000	AXI Read Bandwidth Status 1
AXI_WBW0	0x9218	32	ro	0x00000000	AXI Write Bandwidth Status 0
AXI_WBW1	0x921C	32	ro	0x00000000	AXI Write Bandwidth Status 1
AXI_RBL0	0x9220	32	rw	0x00000000	AXI Read Bandwidth Limiter 0
AXI_RBL1	0x9224	32	rw	0x00000000	AXI Read Bandwidth Limiter 1

Notes:

1. *Mixed* registers are registers that have read only, write only, and read write bits grouped together.

Improving VCU Encoder Quality

The quality of encoded video is based on a function of the target-bitrate and type of video content. Several encoder parameters can be used to adjust the encoder quality, such as:

- The number of B-frames (Gop.NumB) that can be adjusted according to the amount of motion, for example, increase to two for static scenes or video conferencing or reduce to zero for sequences with a lot of motion or high frame rates.

- The VBR rate control mode can improve the average quality when some parts of the sequence have lower complexity or motion.
- For video conference or when random access is not needed, you can replace the IPP... GOP by the `LOW_DELAY_P` GOP and optionally enable the GDR intra refresh.
- If there are frequent scene changes, the `ScnChgResilience` setting can be enabled to reduce artifacts following scene change transitions.
- If scene changes can be detected by the system, the encoder's scene change signaling API should be called instead (with `ScnChgResilience` disabled, for example) for the encoder to dynamically adapt the encoding parameters and GOP pattern. The scene change information can be provided in a separate input file (`CmdFile`) when using the control software test application.
- If the highest PSNR figures are targeted instead of subjective quality, it is recommended to set QPCtrlMode = UNIFORM_QP and ScalingList = FLAT.
- If target bitrate is too low for complex video scenes, I frame beating or a sweeping low quality line (if in GDR mode) is noticeable. Target bitrate needs to be increased to avoid such visual artifacts.

Decoder Block

The decoder block is designed to process video streams using the H.265 (HEVC) and H.264 (AVC) standards. It provides a complete support for these standards, including support for 8-bit and 10-bit color depth, 4:0:0, 4:2:0, and 4:2:2 Chroma formats, up to 4K UHD at 60 Hz performance.

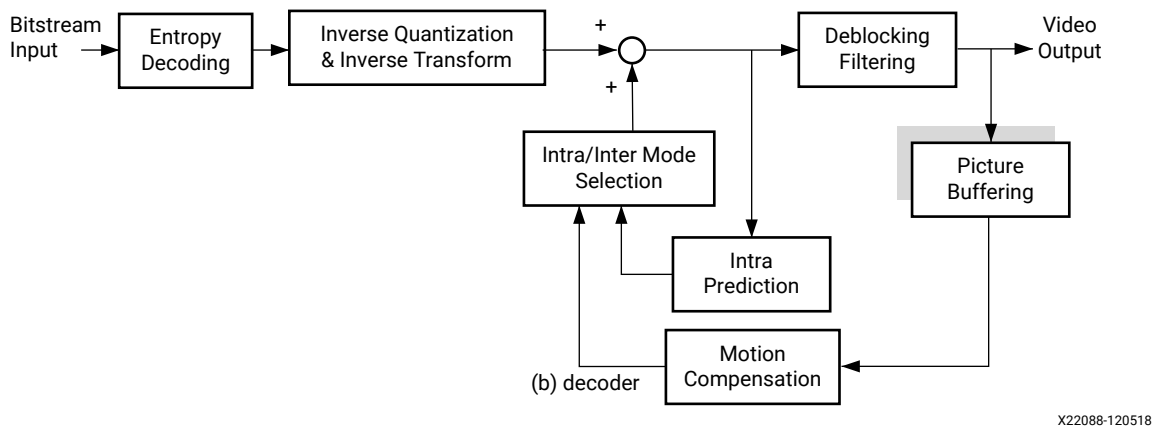
The decoder block efficiently performs video decompression.

The IP hardware has a direct access to the system data bus through a high-bandwidth master interface to transfer video data to and from an external memory.

The IP control software is partitioned into two layers. The VCU Control Software runs on the APU while the MCU firmware runs on an MCU, which is embedded in the hardware IP. The APU communicates with the embedded MCU through a slave interface, also connected to the system bus. The IP hardware is controlled by the embedded MCU using a register map to set decoding parameters through an internal peripheral bus.

The VCU decoder block is shown in the following figure.

Figure 8: **VCU Decoder Block**



Features

The following table describes the decoder block features.

Table 20: **VCU Features**

Video Coding Feature	H.264	H.265
Performance		
Profiles	Baseline (except FMO/ASO/RS) Constrained Baseline Main High High 10 High 4:2:2	Main Main Intra Main 10 Main 10 Intra Main 4:2:2 10 Main 4:2:2 10 Intra
Levels	up to 5.2[2]	up to 5.1 High Tier[1]
Performance at 667 MHz 32 streams at 720x480p at 30 Hz Eight streams at 1920x1080p at 30 Hz Four streams at 1920x1080p at 60 Hz Two streams at 3840x2160p at 30 Hz One stream at 3840x2160p at 60 Hz One stream at 7680x4320p at 15 Hz	Supported	Supported
Configurable resolution Picture width and height multiple of eight Maximum width or height: 8192 Maximum picture size of 33.5 MP	Supported Minimum size: 80×96 Maximum width: 8,192 Maximum height: 8,192	Supported Minimum size: 128×128 Maximum width: 8184 (limited to 4,096 in level 4/4.1 or when WPP is enabled) Maximum height: 8,192
Configurable frame rate	Supported	Supported

Table 20: **VCU Features** *(cont'd)*

Video Coding Feature	H.264	H.265
Coding Tools		
Sample bit depth: 8 bpc, 10 bpc	Supported	Supported
Chroma format: YCbCr 4:2:0, YCbCr 4:2:2, Y-only (monochrome)	Supported	Supported
Progressive format only	Supported	Supported

Notes:

1. Support of 8K15 uses a subset of level 6: maximum Luma picture size up to 2^{25} samples, other constraints of Level 5.1 apply (e.g., maximum of 200 slices and 11×10 tiles), WPP is not supported for widths above 4,096.
2. Support of 8K15 uses a subset of Level 6: maximum Luma picture size up to 2^{25} samples, other constraints of Level 5.2 apply, maximum slice size of 65,535 macroblocks so a minimum of two balanced slices must be used above 4K size.

The following table describes the VCU decoder block maximum supported bit rates.

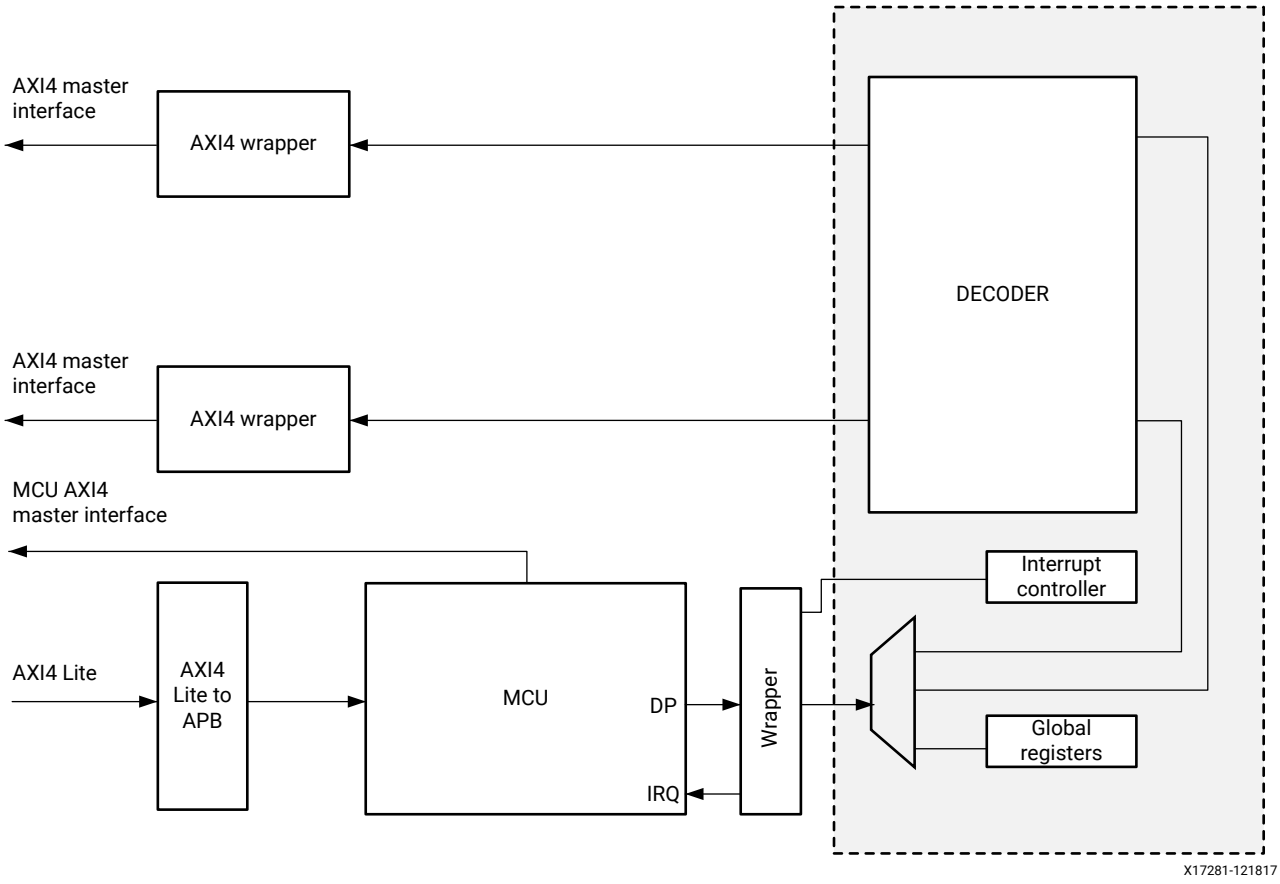
Table 21: **VCU Decoder Maximum Supported Bit Rates**

Standard	Level	Profile	Maximum Bit Rate (Mbits/s)
H.264	4.2 (1080p60)	Baseline, Main	50
		High	62.5
		High 10	150
		High 4:2:2	200
	5.2 (2160p60)	Baseline, Main	240
		High	300
		High 10	720
		High 4:2:2	960 (CAVLC) 720 (CABAC)
H.264	4.1 (1080p60) High Tier	Main, Main 10	50
		Main 4:2:2 10	84
		Main 4:2:2 10 Intra	167
	5.1 (2160p60) High Tier	Main, Main 10	160
		Main 4:2:2 10	267
		Main 4:2:2 10 Intra	533

Functional Description

The following figure shows the block diagram of the decoder block.

Figure 9: **Detailed Architecture of the Decoder Block**



The decoder block includes the H.265/H.264 decompression engine, control registers, and an interrupt controller block. The decoder block is controlled by an MCU subsystem. A 32-bit AXI4-Lite slave interface is used by the system CPU to control the MCU to configure decoder parameters, start processing of video frames and to get status and results. Two 128-bit AXI4 master interfaces are used to fetch video input data and store video output data from/to the system memory. An AXI4 master interface is used to fetch the MCU software and performs load/store operation on additional MCU data.

Interfaces and Ports

Applications that use the decoder must connect all the decoder ports (ports beginning with `m_axi_dec`). The following table shows the decoder block AXI4 master interface ports.

Table 22: **Decoder Ports**

Name	Width	Direction	Description
vcu_pl_dec_araddr0/1	44	Output	AXI4 ARADDR signal
vcu_pl_dec_arburst0/1	2	Output	AXI4 ARBURST signal

Table 22: **Decoder Ports** *(cont'd)*

Name	Width	Direction	Description
vcu_pl_dec_arid0/1	4	Output	AXI4 ARID signal
vcu_pl_dec_arlen0/1	8	Output	AXI4 ARLEN signal
pl_vcu_dec_arready0/1	1	Input	AXI4 ARREADY signal
vcu_pl_dec_arsize0/1	3	Output	AXI4 ARSIZE signal
vcu_pl_dec_arvalid0/1	1	Output	AXI4 ARVALID signal
vcu_pl_dec_awaddr0/1	44	Output	AXI4 AWADDR signal
vcu_pl_dec_awburst0/1	2	Output	AXI4 AWBURST signal
vcu_pl_dec_awid0/1	4	Output	AXI4 AWID signal
vcu_pl_dec_awlen0/1	8	Output	AXI4 AWLEN signal
pl_vcu_dec_awready0/1	1	Input	AXI4 AWREADY signal
vcu_pl_dec_awsize0/1	3	Output	AXI4 AWSIZE signal
vcu_pl_dec_awvalid0/1	1	Output	AXI4 AWVALID signal
pl_vcu_dec_bresp0/1	2	Input	AXI4 BRESP signal
vcu_pl_dec_bready0/1	1	Output	AXI4 BREADY signal
pl_vcu_dec_bvalid0/1	1	Input	AXI4 BVALID signal
pl_vcu_dec_bid0/1	4	Input	AXI4 BID signal
pl_vcu_dec_rdata0/1	128	Input	AXI4 RDATA signal
pl_vcu_dec_rid0/1	4	Input	AXI4 RID signal
pl_vcu_dec_rlast0/1	1	Input	AXI4 RLAST signal
vcu_pl_dec_rready0/1	1	Output	AXI4 RREADY signal
pl_vcu_dec_rresp0/1	2	Input	AXI4 RRESP signal
pl_vcu_dec_rvalid0/1	1	Input	AXI4 RVALID signal
vcu_pl_dec_wdata0/1	128	Output	AXI4 WDATA signal
vcu_pl_dec_wlast0/1	1	Output	AXI4 WLAST signal
pl_vcu_dec_wready0/1	1	Input	AXI4 WREADY signal
vcu_pl_dec_wvalid0/1	1	Output	AXI4 WVALID signal
vcu_pl_dec_awprot0/1	1	Output	AXI4 AWPROT signal, controlled from System Level Control Register (SLCR)
vcu_pl_dec_arprot0/1	1	Output	AXI4 ARPROT signal, controlled from SLCR
vcu_pl_dec_awqos0/1	4	Output	AXI4 AWQOS signal, controlled from SLCR
vcu_pl_dec_arqos0/1	4	Output	AXI4 ARQOS signal, controlled from SLCR
vcu_pl_dec_awcache0/1	4	Output	AXI4 AWCACHE signal, controlled from SLCR
vcu_pl_dec_arcache0/1	4	Output	AXI4 ARCACHE signal, controlled from SLCR

Clocking

Refer to Clocking and Resets for more information on clocking.

Reset

Refer to Clocking and Resets for more information on resets.

Data Path

The master interface inputs several types of video data from external memory:

- Bitstream
- Reference frame pixels
- Co-located picture motion vectors
- Headers and residual data

The master interface outputs:

- Decoded frame pixels
- Headers and residual data
- Decoded frame motion vectors, when the picture is later used as co-located picture

Control Path

The VCU slave interface is accessed once per frame by the APU, which sends a frame-level command to the IP. This interface therefore does not require a fast data path. An interrupt is generated on conclusion of each frame. These commands are processed by the embedded MCU, which generates tile- and slice-level commands to the Decoder block hardware.

Decoder Buffer Requirements

The Decoder input and output requirements are shown in the following table.

Table 23: **Decoder Buffer Requirements**

Requirement	Description
Input Buffer	
Number of Buffers	>=1
Contiguous	No
Alignment	0
Size	Any > 0

Table 23: **Decoder Buffer Requirements** *(cont'd)*

Requirement	Description
Output Buffer	
Number of Buffers	For AVC: AL_AVC_GetMinOutputBuffersNeeded (AL_TStreamSettings tStreamSettings, int iStack). For HEVC: AL_HEVC_GetMinOutputBuffersNeeded (AL_TStreamSettings tStreamSettings, int iStack). ***Note*:** Dimension, chroma-mode and bit-depth are in stream-settings.
Contiguous	Yes
Alignment	32
Size	stride * slice-height * chroma-mode. ***Note*:** stride and slice-height should be 64-aligned.

***Note*:** Because the VCU uses multiple internal decoder engines, it is not possible to reduce the output buffer requirements.

Memory Footprint Requirements

The Decoder block memory footprint depends on the decoding parameters.

The buffer size depends on the following:

- Video resolution
- Chroma sub-sampling
- Color depth
- Coding standard: H.264 or H.265

The number of buffers depend on the coding standard.

The following table shows the worst-case memory footprint required for different buffer sizes.

Table 24: **Decoder Block Memory Footprint**

Examples with Two B-Frames	720p			1080p			2160p		
	Buffers	Per Buffer	Total	Buffers	Per Buffer	Total	Buffers	Per Buffer	Total
Input Bitstream Buffer	2	1.9 MB	3.8 MB	2	4.0 MB	8.0 MB	2	16.0 MB	32.0 MB
Circular Bitstream Buffer	1	9.5 MB	9.5 MB	1	20.1 MB	20.1 MB	1	80.0 MB	80.0 MB
Reference Frames	23	2.5 MB	56.6 MB	23	5.3 MB	122.2 MB	12	21.1 MB	253.1 MB
Reconstructed Frame	1	2.5 MB	2.5 MB	1	5.3 MB	5.3 MB	1	21.1 MB	21.1 MB

Table 24: **Decoder Block Memory Footprint** *(cont'd)*

Examples with Two B-Frames	720p			1080p			2160p		
	Buffers	Per Buffer	Total	Buffers	Per Buffer	Total	Buffers	Per Buffer	Total
Intermediate Buffers	5	4.9 MB	24.4 MB	5	11.1 MB	55.4 MB	5	44.0 MB	220.0 MB
Motion vector Buffer	23	0.2 MB	5.1 MB	23	0.5 MB	11.5 MB	12	2.0 MB	23.7 MB
Slice Parameters	5	6.1 MB	30.7 MB	5	6.1 MB	30.7 MB	5	6.1 MB	30.7 MB
Other Buffers	1	3.9 MB	3.9 MB	1	3.9 MB	3.9 MB	1	3.9 MB	3.9 MB
Total			137 MB			258 MB			665 MB

CMA Size Requirements

The following table contains theoretical contiguous memory access (CMA) buffer requirements for the VCU decoder based on resolution and format. The sizes below correspond to one instance of the encoder or decoder. Multiply these by number of streams for multistream use cases. Other elements such as kmssink/v4l2src typically increase the CMA requirements by an additional 10 to 15%.

Table 25: **VCU Decoder CMA Requirements**

Resolution	4:2:2 10-bit AVC	4:2:2 10-bit HEVC	4:2:2: 8-bit s AVC	4:2:2: 8-bit s HEVC	4:2:0 10-bit AVC	4:2:0 10-bit HEVC	4:2:0 8-bit AVC	4:2:0 8-bit HEVC
3840×2160 (MB)	665	582	597	513	524	466	473	414
1920×1080 (MB)	258	214	232	190	208	167	188	148
1280×720 (MB)	137	104	120	87	144	82	101	69

Memory Bandwidth

The decoder memory bandwidth depends on frame rate, resolution, color depth, chroma format and Decoder profile. The LogiCORE IP provides an estimate of decoder bandwidth based on the video parameters selected in the GUI.

Xilinx recommends using the fastest DDR4 memory interface possible. Specifically, the 8x8-bit memory interface is more efficient than 4x16-bit memory interface because the x8 mode has four bank groups, whereas the x16 mode has only two and DDR4 allows for simultaneous bank group access. For more information, see Xilinx Answer: 71209.

Memory Format

The decoded picture buffer contains the decoded pixels. It contains two parts: luminance pixels (Luma) followed by chrominance pixels (Chroma). Luma pixels are stored in pixel raster scan order. Chroma pixels are stored in U/V-interleaved pixel raster scan order, hence the Chroma part is half the size of the Luma part when using a 4:2:0 format and the same size as the Luma part when using a 4:2:2 format. The decoded picture buffer must be one contiguous memory region.

***Note*:** Decoder output buffers width are in multiple of 256 byte. Height is in multiples of 64. For example: for 1920x1080 resolution, Decoder output is 2048x1088.

Two packing formats are supported in external memory: eight bits per component or 10 bits per component, shown in the following tables, respectively. The 8-bit format can only be used for an 8-bit component depth and the 10-bit format can only be used for a 10-bit component depth. The following tables show the raster scan format supported by the decoder block for 8-bit and 10-bit color depth.

255 248	...	31 24	23 16	15 8	7 0
$Y_{x+31,y}$	...	$Y_{x+3,y}$	$Y_{x+2,y}$	$Y_{x+1,y}$	$Y_{x,y}$
...(all Luma in pixel raster scan order)...					
255 248	247 240 ...	31 24	23 16	15 8	7 0
$V_{x+15,y}$	$U_{x+15,y}$ ...	$V_{x+1,y}$	$U_{x+1,y}$	$V_{x,y}$	$U_{x,y}$
...(all interleaved Chroma in pixel raster scan order)...					

255	254	253 244							31	30	29 20	19 10	9 0
0	0	$V_{x+23,y}$							0	0	$Y_{x+2,y}$	$Y_{x+1,y}$	$Y_{x,y}$
...(all Luma in pixel raster scan order)...													
255	254	253 244	...	63	62	61 52	51 42	41 32	31	30	29 20	19 10	9 0
0	0	$V_{x+11,y}$	...	0	0	$V_{x+2,y}$	$U_{x+2,y}$	$V_{x+1,y}$	0	0	$U_{x+1,y}$	$V_{x,y}$	$U_{x,y}$
...(all interleaved Chroma in pixel raster scan order)...													

The frame buffer width (pitch) may be larger than the frame width so that there are (pitch - width) ignored values between consecutive pixel lines.

VCU Decoder

The VCU encoded data and the decoded format is shown in the following table:

Table 28: **VCU Decoder 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.

Decoder Block Register Overview

The following table lists the decoder block registers. For additional information, see the *Zynq UltraScale+ Device Register Reference* (UG1087).

Table 29: **Decoder Registers**

Register Name	Offset	Width	Type	Reset Value	Description
MCU_RESET	0x9000	32	mixed[1]	0x00000000	MCU Subsystem Reset
MCU_RESET_MODE	0x9004	32	mixed[1]	0x00000001	MCU Reset Mode
MCU_STA	0x9008	32	mixed[1]	0x00000000	MCU Status
MCU_WAKEUP	0x900C	32	mixed[1]	0x00000000	MCU Wake-up
MCU_ADDR_OFFSET_IC0	0x9010	32	rw	0x00000000	MCU Instruction Cache Address Offset 0
MCU_ADDR_OFFSET_IC1	0x9014	32	rw	0x00000000	MCU Instruction Cache Address Offset 1
MCU_ADDR_OFFSET_DC0	0x9018	32	rw	0x00000000	MCU Data Cache Address Offset 0
MCU_ADDR_OFFSET_DC1	0x901C	32	rw	0x00000000	MCU Data Cache Address Offset 1
ITC_MCU_IRQ	0x9100	32	mixed[1]	0x00000000	MCU Interrupt Trigger
ITC_CPU_IRQ_MSK	0x9104	32	rw	0x00000000	CPU Interrupt Mask
ITC_CPU_IRQ_CLR	0x9108	32	mixed[1]	0x00000000	CPU Interrupt Clear
ITC_CPU_IRQ_STA	0x910C	32	mixed[1]	0x00000000	CPU Interrupt Status
AXI_BW	0x9204	32	rw	0x00000000	AXI Bandwidth Measurement Window
AXI_ADDR_OFFSET_IP	0x9208	32	rw	0x00000000	Video Data Address Offset
AXI_RBW0	0x9210	32	ro	0x00000000	AXI Read Bandwidth Status 0
AXI_RBW1	0x9214	32	ro	0x00000000	AXI Read Bandwidth Status 1

Table 29: **Decoder Registers** *(cont'd)*

Register Name	Offset	Width	Type	Reset Value	Description
AXI_WBW0	0x9218	32	ro	0x00000000	AXI Write Bandwidth Status 0
AXI_WBW1	0x921C	32	ro	0x00000000	AXI Write Bandwidth Status 1
AXI_RBL0	0x9220	32	rw	0x00000000	AXI Read Bandwidth Limiter 0
AXI_RBL1	0x9224	32	rw	0x00000000	AXI Read Bandwidth Limiter 1

Notes:

1. *Mixed* registers are registers that have read only, write only, and read write bits grouped together.

***Note*:** The VCU encoder and decoder output streams are stored in DDR memory (either PS or PL) and cannot be routed directly from encoder to decoder and vice versa, so it is required to use DDR (PS or PL) with the VCU encoder and decoder.

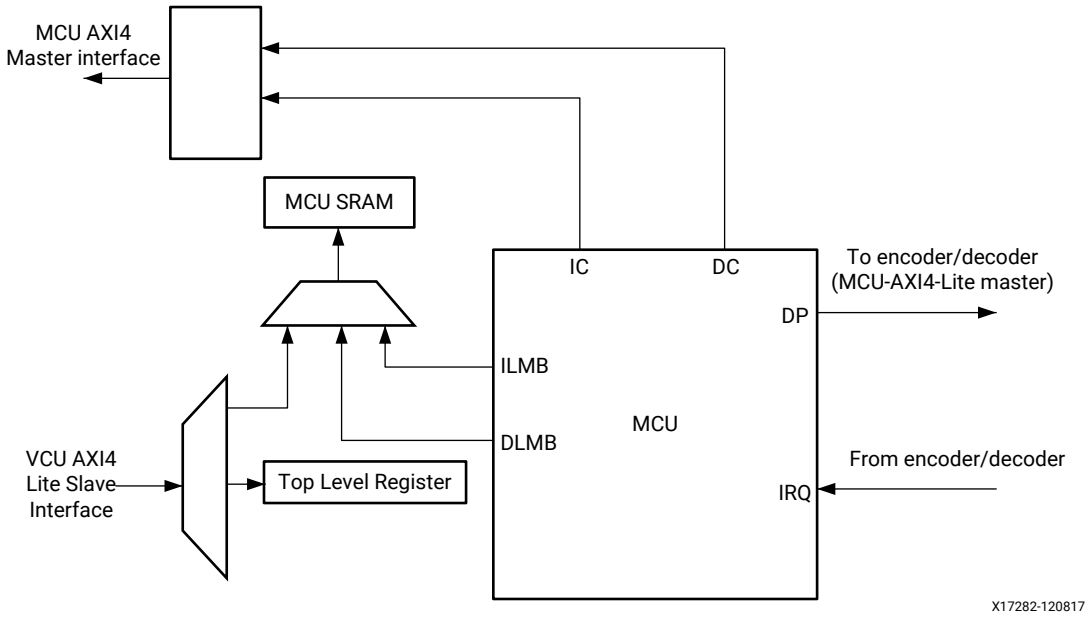
Microcontroller Unit Overview

The VCU core includes two MCU subsystems that run the MCU firmware and control the encoder and decoder blocks. The encoder and decoder blocks each have their own MCU to execute the firmware. The MCU has a 32-bit RISC architecture capable of executing pipelined transactions. The MCU has internal instruction, data cache, and AXI master interface to interface with the external memory.

Functional Description

The following figure shows the top-level interfaces and detailed architecture of the MCU.

Figure 10: **MCU (Top-level)**



The MCU interfaces to peripherals using a 32-bit AXI4-Lite master interface. It has a local memory bus, an AXI4 32-bit instruction, and data cache interfaces.

The MCU block has a 32 KB local memory for internal operations that is shared with the CPU for boot and mailbox communication. The MCU has a 32 KB instruction cache with 32-byte cache line width. It has a 4 KB data cache with 16-byte cache line width. The data cache has a write-through cache implementation.

Interfaces and Ports

The following table shows the AXI4 instruction and data cache interface ports of MCU.

Table 30: **MCU Ports**

Port	Size (bits)	Direction	Description
vcu_pl_mcu_m_axi_ic_dc_araddr	44	Output	AXI4 read address
vcu_pl_mcu_m_axi_ic_dc_arburst	2	Output	AXI4 read burst type
vcu_pl_mcu_m_axi_ic_dc_arcache	4	Output	AXI4 ARCACHE value
vcu_pl_mcu_m_axi_ic_dc_arid	3	Output	AXI4 read master ID
vcu_pl_mcu_m_axi_ic_dc_arlen	8	Output	AXI4 read burst size
vcu_pl_mcu_m_axi_ic_dc_arlock	1	Output	AXI4 ARLOCK signal
vcu_pl_mcu_m_axi_ic_dc_arprot	3	Output	AXI4 ARPROT signal
vcu_pl_mcu_m_axi_ic_dc_arqos	4	Output	AXI4 ARQOS signal
pl_vcu_mcu_m_axi_ic_dc_arready	1	Input	AXI4 ARREADY signal
vcu_pl_mcu_m_axi_ic_dc_arsize	3	Output	AXI4 ARSIZE signal

Table 30: **MCU Ports** *(cont'd)*

Port	Size (bits)	Direction	Description
vcu_pl_mcu_m_axi_ic_dc_arvalid	1	Output	AXI4 ARVALID signal
vcu_pl_mcu_m_axi_ic_dc_awaddr	44	Output	AXI4 AWADDR signal
vcu_pl_mcu_m_axi_ic_dc_awburst	2	Output	AXI4 AWBURST signal
vcu_pl_mcu_m_axi_ic_dc_awcache	4	Output	AXI4 AWCACHE signal
vcu_pl_mcu_m_axi_ic_dc_awid	3	Output	AXI4 AWID signal
vcu_pl_mcu_m_axi_ic_dc_awlen	8	Output	AXI4 AWLEN signal
vcu_pl_mcu_m_axi_ic_dc_awlock	1	Output	AXI4 AWLOCK signal
vcu_pl_mcu_m_axi_ic_dc_awprot	3	Output	AXI4 AWPROT signal
vcu_pl_mcu_m_axi_ic_dc_awqos	4	Output	AXI4 AWQOS signal
pl_vcu_mcu_m_axi_ic_dc_awready	1	Input	AXI4 AWREADY signal
vcu_pl_mcu_m_axi_ic_dc_awsize	3	Output	AXI4 AWSIZE signal
vcu_pl_mcu_m_axi_ic_dc_awvalid	1	Output	AXI4 AWVALID signal
pl_vcu_mcu_m_axi_ic_dc_bid	3	Input	AXI4 BID signal
vcu_pl_mcu_m_axi_ic_dc_bready	1	Output	AXI4 BREADY signal
pl_vcu_mcu_m_axi_ic_dc_bresp	2	Input	AXI4 BRESP signal
pl_vcu_mcu_m_axi_ic_dc_bvalid	1	Input	AXI4 BVALID signal
pl_vcu_mcu_m_axi_ic_dc_rdata	32	Input	AXI4 RDATA signal
pl_vcu_mcu_m_axi_ic_dc_rid	3	Input	AXI4 RID signal
pl_vcu_mcu_m_axi_ic_dc_rlast	1	Input	AXI4 RLAST signal
vcu_pl_mcu_m_axi_ic_dc_rready	1	Output	AXI4 RREADY signal
pl_vcu_mcu_m_axi_ic_dc_rresp	2	Input	AXI4 RRESP signal
pl_vcu_mcu_m_axi_ic_dc_rvalid	1	Input	AXI4 RVALID signal
vcu_pl_mcu_m_axi_ic_dc_wdata	32	Output	AXI4 WDATA signal
vcu_pl_mcu_m_axi_ic_dc_wlast	1	Output	AXI4 WLAST signal
pl_vcu_mcu_m_axi_ic_dc_wready	1	Input	AXI4 WREADY signal
vcu_pl_mcu_m_axi_ic_dc_wstrb	4	Output	AXI4 WSTRB signal
vcu_pl_mcu_m_axi_ic_dc_wvalid	1	Output	AXI4 WVALID signal

The following table summarizes the AXI4-Lite slave interface ports of the MCU subsystem.

Table 31: **AXI4-Lite Slave Ports**

Port	Width	Direction	Description
pl_vcu_awaddr_axi_lite_apb	20	Input	AXI4 AWADDR signal
pl_vcu_awprot_axi_lite_apb	3	Input	AXI4 AWPROT signal
pl_vcu_awvalid_axi_lite_apb	1	Input	AXI4 AWVALID signal
vcu_pl_awready_axi_lite_apb	1	Output	AXI4 AWREADY signal
pl_vcu_wdata_axi_lite_apb	32	Input	AXI4 WDATA signal
pl_vcu_wstrb_axi_lite_apb	4	Input	AXI4 WSTRB signal

Table 31: **AXI4-Lite Slave Ports** *(cont'd)*

Port	Width	Direction	Description
pl_vcu_wvalid_axi_lite_apb	1	Input	AXI4 WVALID signal
vcu_pl_wready_axi_lite_apb	1	Output	AXI4 WREADY signal
vcu_pl_bresp_axi_lite_apb	2	Output	AXI4 BRESP signal
vcu_pl_bvalid_axi_lite_apb	1	Output	AXI4 BVALID signal
pl_vcu_bready_axi_lite_apb	1	Input	AXI4 BREADY signal
pl_vcu_araddr_axi_lite_apb	20	Input	AXI4 ARADDR signal
pl_vcu_arprot_axi_lite_apb	3	Input	AXI4 ARPROT signal
pl_vcu_arvalid_axi_lite_apb	1	Input	AXI4 ARVALID signal
vcu_pl_arready_axi_lite_apb	1	Output	AXI4 ARREADY signal
vcu_pl_rdata_axi_lite_apb	32	Output	AXI4 RDATA signal
vcu_pl_rresp_axi_lite_apb	2	Output	AXI4 RRESP signal
vcu_pl_rvalid_axi_lite_apb	1	Output	AXI4 RVALID signal
pl_vcu_rready_axi_lite_apb	1	Input	AXI4 RREADY signal

Control Flow

The MCU is kept in sleep mode after applying the reset until the firmware boot code is downloaded by the kernel device driver into the internal memory of the MCU. After downloading the boot code and completing the MCU initialization sequence, the control software communicates with the MCU using a mailbox mechanism implemented in the internal SRAM of the MCU. The MCU sends an acknowledgment to the control software and performs the encoding/decoding operation. When the requested operation is complete, the MCU communicates the status to the control software.

For more details about control software and MCU firmware, refer to Section V: Application Software Development.

MCU Register Overview

The following table lists the MCU registers. For additional information, see the *Zynq UltraScale+ Device Register Reference* (UG1087) .

Table 32: **Encoder MCU Registers**

Register	Address	Width	Type	Reset Value	Description
MCU_RESET	0xA0009000	32	mixed[1]	0x00000000	MCU Subsystem Reset
MCU_RESET_MODE	0xA0009004	32	mixed[1]	0x00000001	MCU Reset Mode
MCU_STA	0xA0009008	32	mixed[1]	0x00000000	MCU Status
MCU_WAKEUP	0xA000900C	32	mixed[1]	0x00000000	MCU Wake-up
MCU_ADDR_OFFSET_IC0	0xA0009010	32	rw	0x00000000	MCU Instruction Cache Address Offset 0
MCU_ADDR_OFFSET_IC1	0xA0009014	32	rw	0x00000000	MCU Instruction Cache Address Offset 1

Table 32: **Encoder MCU Registers** *(cont'd)*

Register	Address	Width	Type	Reset Value	Description
MCU_ADDR_OFFSET_DC0	0xA0009018	32	rw	0x00000000	MCU Data Cache Address Offset 0
MCU_ADDR_OFFSET_DC1	0xA000901C	32	rw	0x00000000	MCU Data Cache Address Offset 1

Notes:

1. *Mixed* registers are registers that have read only, write only, and read write bits grouped together.

AXI Performance Monitor

Overview

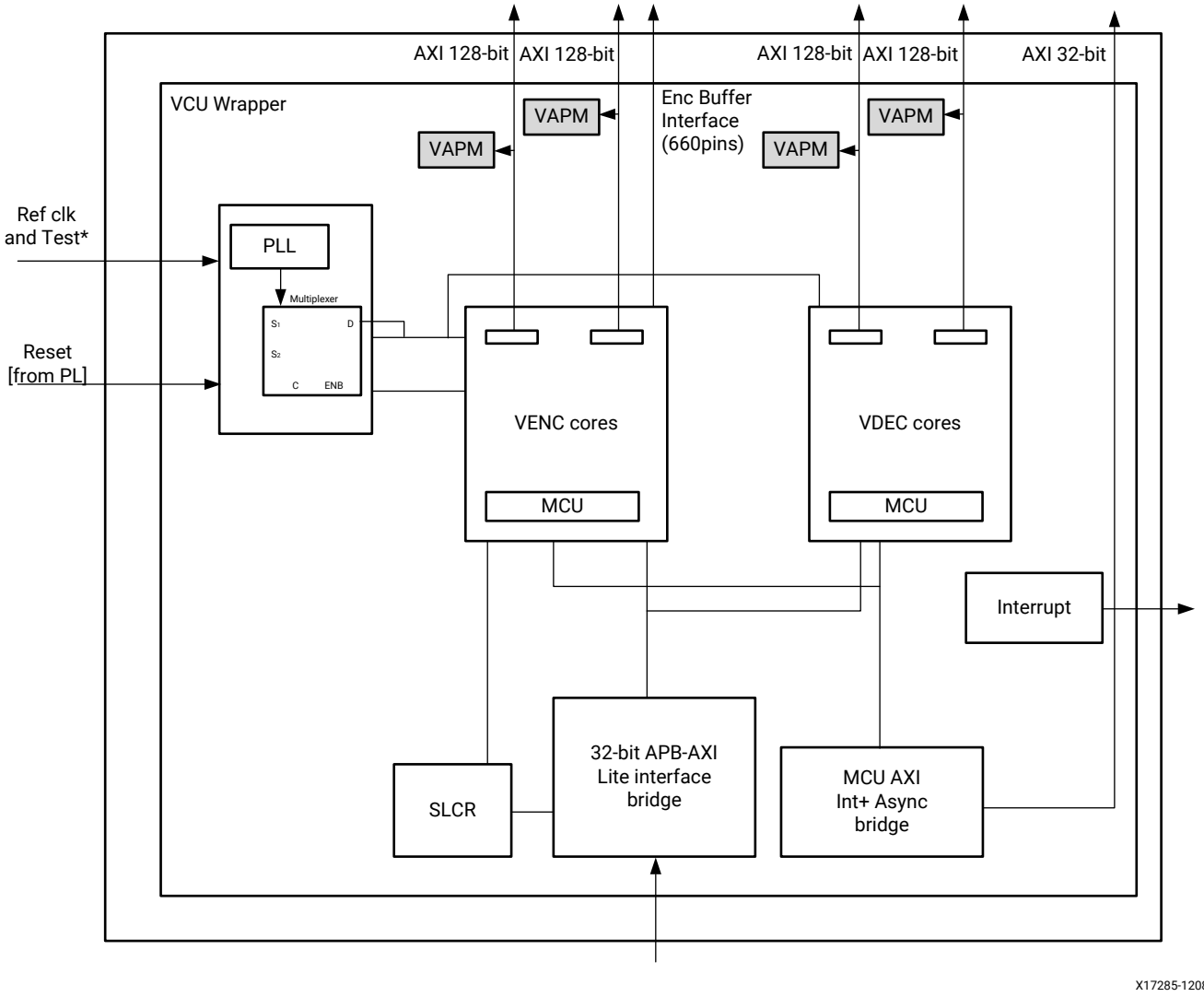
The AXI Performance Monitor (APM) is implemented inside the embedded Video Codec Unit (VCU). The VCU AXI Performance Monitor (VAPM) allows access to system level behavior in a non-invasive way and without burdening the design with additional soft IP.

The APM block is capable of measuring the number of read/write bytes and address based transactions within a measurement window on the AXI master bus from Encoder/Decoder blocks. The APM can additionally measure master ID based read and write latency within a measurement window. The APM supports cumulative latency value along with the number of outstanding transfers being considered for latency measurement. The APM has the ability to interrupt the host processor when the status registers are ready to be read.

Functional Description

The following figure shows the VAPM.

Figure 11: **VCU AXI Performance Monitor (VAPM)**



The following sections describe the different operating modes of VAPM.

Operating Timing Window Generation

The VAPM generates measurement parameters based on two user-selected operating modes.

Start/Stop Mode

In this mode, the measurement window is determined by the start/stop bit in `VCU_SLCR.APMn_TRG[start_stop] (n=0,1,2,3)` bit. A measurement is triggered when the start bit is set from 0 to 1 in this register and it is stopped when this bit is reset from 1 to 0.

Fixed Duration Timing Window

In this mode, a 32-bit counter is used to generate a fixed length measurement window. When the counter reaches the maximum value, it resets to a value specified in the VCU_SLCR.APM*n*_TIMER (n = 0, 1, 2, 3) register. The measurement is continued until the 32-bit counter reaches the value set in the APM*n*_TIMER register and a capture pulse is generated to store the measured values in the VCU_SLCR result registers.

The VAPM is capable of doing the following measurements:

- **AXI Read and Write Transaction Measurement:** Two 32-bit registers count number of read and write 128-bit AXI bus cycles transferred in a given timing window. The measured value is transferred to the VCU_SLCR result register when a capture pulse is generated based on the start/stop mode or the fixed duration timing window mode. To compute the number of bytes transferred, VCU_SLCR must be multiplied by 16.
- **AXI Read and Write Byte Count Measurement:** Two 32-bit registers are implemented to count number of read and write bytes transferred in a given timing window. The register content has to be multiplied by 16 to know the actual byte count transferred across AXI 128-bit master interface. The measured value is transferred to the VCU_SLCR result register when a capture pulse is generated based on the start/stop mode or fixed duration timing window mode.
- **AXI Transaction Latency Measurement:** Read and write latency can be measured based on AXI master ID. Read latency is defined as AXI read address acknowledged to last read data cycle. Write latency is defined as AXI write address acknowledged to write response handshaking between master and slave. A 13-bit counter is implemented to measure the latency on read and write bus. The timer is used to timestamp an event. The difference in the timestamp between two events is used to calculate the latency.

Latency can be calculated on transaction ID basis. It is possible to select a single ID or all IDs for latency calculation. For additional information, see the *Zynq UltraScale+ Device Register Reference* (UG1087).

APM Registers

Table 33: **APM Registers**

Register	Address	Width	Type	Reset Value	Description
APM_InputT_GBL_CNTL	0×A0040090	32	mixed	0x00000001	This register controls APM timing window completion interrupt.
APM0_CFG	0xA0040100	32	mixed	0x00000002	APM0_CFG
APM0_TIMER	0xA0040104	32	rw	0x00000000	APM0_TIMER
APM0_TRG	0xA0040108	32	mixed	0x00000000	APM0_TRG
APM0_RESULT0	0xA004010C	32	ro	0x00000000	APM0_RESULT0
APM0_RESULT1	0xA0040110	32	ro	0x00000000	APM0_RESULT1
APM0_RESULT2	0xA0040114	32	ro	0x00000000	APM0_RESULT2

Table 33: **APM Registers** *(cont'd)*

Register	Address	Width	Type	Reset Value	Description
APM0_RESULT3	0xA0040118	32	ro	0x00000000	APM0_RESULT3
APM0_RESULT4	0xA004011C	32	mixed	0x00000000	APM0_RESULT4
APM0_RESULT5	0xA0040120	32	mixed	0x00000000	APM0_RESULT5
APM0_RESULT6	0xA0040124	32	mixed	0x00000000	APM0_RESULT6
APM0_RESULT7	0xA0040128	32	mixed	0x00000000	APM0_RESULT7
APM0_RESULT8	0xA004012C	32	mixed	0x00000000	APM0_RESULT8
APM0_RESULT9	0xA0040130	32	mixed	0x00000000	APM0_RESULT9
APM0_RESULT10	0xA0040134	32	mixed	0x00000000	APM0_RESULT10
APM0_RESULT11	0xA0040138	32	mixed	0x00000000	APM0_RESULT11
APM0_RESULT12	0xA004013C	32	mixed	0x00000000	APM0_RESULT12
APM0_RESULT13	0xA0040140	32	mixed	0x00000000	APM0_RESULT13
APM0_RESULT14	0xA0040144	32	mixed	0x00000000	APM0_RESULT14
APM0_RESULT15	0xA0040148	32	mixed	0x00000000	APM0_RESULT15
APM0_RESULT16	0xA004014C	32	mixed	0x00000000	APM0_RESULT16
APM0_RESULT17	0xA0040150	32	mixed	0x00000000	APM0_RESULT17
APM0_RESULT18	0xA0040154	32	mixed	0x00000000	APM0_RESULT18
APM0_RESULT19	0xA0040158	32	mixed	0x00000000	APM0_RESULT19
APM0_RESULT20	0xA004015C	32	mixed	0x1FFF0000	APM0_RESULT20
APM0_RESULT21	0xA0040160	32	mixed	0x1FFF0000	APM0_RESULT21
APM0_RESULT22	0xA0040164	32	mixed	0x1FFF0000	APM0_RESULT22
APM0_RESULT23	0xA0040168	32	mixed	0x1FFF0000	APM0_RESULT23
APM0_RESULT24	0xA004016C	32	mixed	0x00000000	APM0_RESULT24
APM1_CFG	0xA0040200	32	mixed	0x00000002	APM1_CFG
APM1_TIMER	0xA0040204	32	rw	0x00000000	APM1_TIMER
APM1_TRG	0xA0040208	32	mixed	0x00000000	APM1_TRG
APM1_RESULT0	0xA004020C	32	ro	0x00000000	APM1_RESULT0
APM1_RESULT1	0xA0040210	32	ro	0x00000000	APM1_RESULT1
APM1_RESULT2	0xA0040214	32	ro	0x00000000	APM1_RESULT2
APM1_RESULT3	0xA0040218	32	ro	0x00000000	APM1_RESULT3
APM1_RESULT4	0xA004021C	32	mixed	0x00000000	APM1_RESULT4
APM1_RESULT5	0xA0040220	32	mixed	0x00000000	APM1_RESULT5
APM1_RESULT6	0xA0040224	32	mixed	0x00000000	APM1_RESULT6
APM1_RESULT7	0xA0040228	32	mixed	0x00000000	APM1_RESULT7
APM1_RESULT8	0xA004022C	32	mixed	0x00000000	APM1_RESULT8
APM1_RESULT9	0xA0040230	32	mixed	0x00000000	APM1_RESULT9
APM1_RESULT10	0xA0040234	32	mixed	0x00000000	APM1_RESULT10
APM1_RESULT11	0xA0040238	32	mixed	0x00000000	APM1_RESULT11
APM1_RESULT12	0xA004023C	32	mixed	0x00000000	APM1_RESULT12
APM1_RESULT13	0xA0040240	32	mixed	0x00000000	APM1_RESULT13

Table 33: **APM Registers** *(cont'd)*

Register	Address	Width	Type	Reset Value	Description
APM1_RESULT14	0xA0040244	32	mixed	0x00000000	APM1_RESULT14
APM1_RESULT15	0xA0040248	32	mixed	0x00000000	APM1_RESULT15
APM1_RESULT16	0xA004024C	32	mixed	0x00000000	APM1_RESULT16
APM1_RESULT17	0xA0040250	32	mixed	0x00000000	APM1_RESULT17
APM1_RESULT18	0xA0040254	32	mixed	0x00000000	APM1_RESULT18
APM1_RESULT19	0xA0040258	32	mixed	0x00000000	APM1_RESULT19
APM1_RESULT20	0xA004025C	32	mixed	0x1FFF0000	APM1_RESULT20
APM1_RESULT21	0xA0040260	32	mixed	0x1FFF0000	APM1_RESULT21
APM1_RESULT22	0xA0040264	32	mixed	0x1FFF0000	APM1_RESULT22
APM1_RESULT23	0xA0040268	32	mixed	0x1FFF0000	APM1_RESULT23
APM1_RESULT24	0xA004026C	32	mixed	0x00000000	APM1_RESULT24
APM2_CFG	0xA0040300	32	mixed	0x00000002	APM2_CFG
APM2_TIMER	0xA0040304	32	rw	0x00000000	APM2_TIMER
APM2_TRG	0xA0040308	32	mixed	0x00000000	APM2_TRG
APM2_RESULT0	0xA004030C	32	ro	0x00000000	APM2_RESULT0
APM2_RESULT1	0xA0040310	32	ro	0x00000000	APM2_RESULT1
APM2_RESULT2	0xA0040314	32	ro	0x00000000	APM2_RESULT2
APM2_RESULT3	0xA0040318	32	ro	0x00000000	APM2_RESULT3
APM2_RESULT4	0xA004031C	32	mixed	0x00000000	APM2_RESULT4
APM2_RESULT5	0xA0040320	32	mixed	0x00000000	APM2_RESULT5
APM2_RESULT6	0xA0040324	32	mixed	0x00000000	APM2_RESULT6
APM2_RESULT7	0xA0040328	32	mixed	0x00000000	APM2_RESULT7
APM2_RESULT8	0xA004032C	32	mixed	0x00000000	APM2_RESULT8
APM2_RESULT9	0xA0040330	32	mixed	0x00000000	APM2_RESULT9
APM2_RESULT10	0xA0040334	32	mixed	0x00000000	APM2_RESULT10
APM2_RESULT11	0xA0040338	32	mixed	0x00000000	APM2_RESULT11
APM2_RESULT12	0xA004033C	32	mixed	0x00000000	APM2_RESULT12
APM2_RESULT13	0xA0040340	32	mixed	0x00000000	APM2_RESULT13
APM2_RESULT14	0xA0040344	32	mixed	0x00000000	APM2_RESULT14
APM2_RESULT15	0xA0040348	32	mixed	0x00000000	APM2_RESULT15
APM2_RESULT16	0xA004034C	32	mixed	0x00000000	APM2_RESULT16
APM2_RESULT17	0xA0040350	32	mixed	0x00000000	APM2_RESULT17
APM2_RESULT18	0xA0040354	32	mixed	0x00000000	APM2_RESULT18
APM2_RESULT19	0xA0040358	32	mixed	0x00000000	APM2_RESULT19
APM2_RESULT20	0xA004035C	32	mixed	0x1FFF0000	APM2_RESULT20
APM2_RESULT21	0xA0040360	32	mixed	0x1FFF0000	APM2_RESULT21
APM2_RESULT22	0xA0040364	32	mixed	0x1FFF0000	APM2_RESULT22
APM2_RESULT23	0xA0040368	32	mixed	0x1FFF0000	APM2_RESULT23
APM2_RESULT24	0xA004036C	32	mixed	0x00000000	APM2_RESULT24

Table 33: **APM Registers** *(cont'd)*

Register	Address	Width	Type	Reset Value	Description
APM3_CFG	0xA0040400	32	mixed	0x00000002	APM3_CFG
APM3_TIMER	0xA0040404	32	rw	0x00000000	APM3_TIMER
APM3_TRG	0xA0040408	32	mixed	0x00000000	APM3_TRG
APM3_RESULT0	0xA004040C	32	ro	0x00000000	APM3_RESULT0
APM3_RESULT1	0xA0040410	32	ro	0x00000000	APM3_RESULT1
APM3_RESULT2	0xA0040414	32	ro	0x00000000	APM3_RESULT2
APM3_RESULT3	0xA0040418	32	ro	0x00000000	APM3_RESULT3
APM3_RESULT4	0xA004041C	32	mixed	0x00000000	APM3_RESULT4
APM3_RESULT5	0xA0040420	32	mixed	0x00000000	APM3_RESULT5
APM3_RESULT6	0xA0040424	32	mixed	0x00000000	APM3_RESULT6
APM3_RESULT7	0xA0040428	32	mixed	0x00000000	APM3_RESULT7
APM3_RESULT8	0xA004042C	32	mixed	0x00000000	APM3_RESULT8
APM3_RESULT9	0xA0040430	32	mixed	0x00000000	APM3_RESULT9
APM3_RESULT10	0xA0040434	32	mixed	0x00000000	APM3_RESULT10
APM3_RESULT11	0xA0040438	32	mixed	0x00000000	APM3_RESULT11
APM3_RESULT12	0xA004043C	32	mixed	0x00000000	APM3_RESULT12
APM3_RESULT13	0xA0040440	32	mixed	0x00000000	APM3_RESULT13
APM3_RESULT14	0xA0040444	32	mixed	0x00000000	APM3_RESULT14
APM3_RESULT15	0xA0040448	32	mixed	0x00000000	APM3_RESULT15
APM3_RESULT16	0xA004044C	32	mixed	0x00000000	APM3_RESULT16
APM3_RESULT17	0xA0040450	32	mixed	0x00000000	APM3_RESULT17
APM3_RESULT18	0xA0040454	32	mixed	0x00000000	APM3_RESULT18
APM3_RESULT19	0xA0040458	32	mixed	0x00000000	APM3_RESULT19
APM3_RESULT20	0xA004045C	32	mixed	0x1FFF0000	APM3_RESULT20
APM3_RESULT21	0xA0040460	32	mixed	0x1FFF0000	APM3_RESULT21
APM3_RESULT22	0xA0040464	32	mixed	0x1FFF0000	APM3_RESULT22
APM3_RESULT23	0xA0040468	32	mixed	0x1FFF0000	APM3_RESULT23
APM3_RESULT24	0xA004046C	32	mixed	0x00000000	APM3_RESULT24

Notes:

1. *Mixed* registers are registers that have read only, write only, and read write bits grouped together.

Designing with the Core

This section includes guidelines and additional information to facilitate designing with the core.

General Design Guidelines

The Video Codec Unit (VCU) core is a dedicated hardware block in the programming logic (PL). All interfaces are connected through AXI interconnect blocks in the PL. The VCU core is AXI4 compliant on its AXI master interfaces. It can be connected to the `S_AXI_HP_FPD` (or `S_AXI_LPD`, `S_AXI_HPC_FPD`) ports of the PS or AXI compliant interface of the PL memory controller. There are no direct (hardwired) connections from the VCU to the processing system (PS). HP0 to HP4 AXI ports are recommended for PS-DDR video application.

For high bandwidth VCU applications requiring simultaneous encoder and decoder operation, the encoder should be connected to the PS DDR and the decoder should be connected to the PL DDR. Refer to Section II: Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP v1.1 for more information. This approach makes most effective use of limited AXI4 read/write issuance capability in minimizing latency for the decoder. DMA buffer sharing requirements will determine how capture, display, and intermediate processing stages should be mapped to PS or PL DDR.

The register programming interface of the VCU core connects to PS General Purpose (GP) ports. The clock can be used from PL or through an internal PLL inside the VCU core.

Interrupts

There is one interrupt line from the VCU core to the PS (`vcu_host_interrupt`). This interrupt has to be connected to either `PL-PS-IRQ0[7:0]` or `PL-PS-IRQ1[7:0]`. If there are other interrupts in the design, the interrupt has to be concatenated along with the other interrupts and then connected to the PS.

Clocking and Resets

The Video Codec Unit (VCU) core supports one clocking topology, the internal phase locked loop (PLL). An internal VCU PLL drives the high frequency core (667 MHz) and MCU (444 MHz) clocks based on an input reference clock from the programmable logic (PL). The internal PLL generates a clock for the encoder and decoder blocks.

***Note*:** All AXI clocks are supplied with clocks from external PL sources. These clocks are asynchronous to core encoder and decoder block clocks. The encoder and decoder blocks handle asynchronous clocking in the AXI ports.

The VCU core is reset under the following conditions:

- Initially while the PL is in power-up/configuration mode, the VCU core is held in reset.
- After the PL is fully configured, a PL based reset signal can be used to reset the VCU for initialization and bring-up. Platform management unit (PMU) in the processing system (PS) can drive this reset signal to control the reset state of the VCU.
- During partial reconfiguration (PR), the VCU block is kept under reset if it is part of the dynamically reconfigurable module.

Functional Description

Clocking

The Decoder (VDEC) and Encoder (VENC) blocks work independently as separate units without any dependency on each other. The following table describes the clock domains in VCU core.

Table 34: **VCU Clock Domains**

Domain	Max Freq (MHz)	Description
Core clock	712	Processing core, most of the logic and memories
MCU clock	444	Internal micro controllers

AXI Master Port clock	333	`m_axi_enc_aclk`
		`m_axi_dec_aclk`
		`enc_buffer_clk`
		`pl_vcu_axi_mcu_clk`
		
		AXI master port for memory access, 128-bit, typically connected to PS AFI-FM (HP) port or to a soft memory controller in the PL.

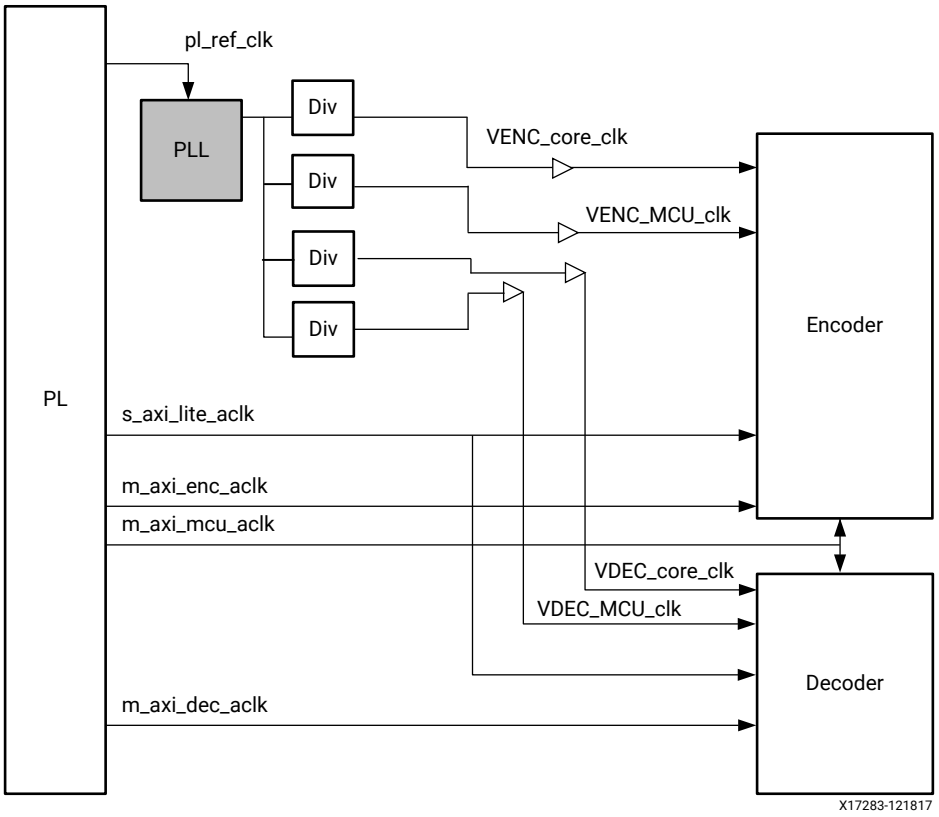
AXI4-Lite slave port clock	167	`s_axi_lite_aclk`, AXI4-Lite slave port (32-bit) for register programming

***Note*:** All AXI clocks are supplied with clocks from external PL sources. These clocks are asynchronous to core encoder, decoder, and MCU clocks. The VENC and VDEC cores are designed to handle asynchronous clocking in the AXI ports. The `m_axi_mcu_aclk` is asynchronous to all clocks used in VCU.

The following figure shows the clock generation options inside VCU block. Note that the following blocks work on a single clock domain:

- `pll_ref_clk` is sourced externally to the device, typically by a programmable clock integrated circuit.
- Video encoder and decoder blocks work under the `VENC_core_clk` domain generated by the VCU PLL.
- MCU for encoder and decoder work under the `VENC_MCU_clk` domain generated by the VCU PLL.
- `m_axi_enc_aclk` is the AXI clock input from the PL for the 128-bit AXI master interfaces for the encoder.
- `m_axi_dec_aclk` is the AXI clock input from the PL for the 128-bit AXI master interfaces for the decoder.
- `s_axi_lite_aclk` is the AXI4-Lite clock from the PL.
- `m_axi_mcu_aclk` is the MCU AXI master clock from the PL.

Figure 12: **Clock Generation Options**



The following clock frequency requirements must be met while providing clocks from PL:

- The AXI clock for encoder and decoder interface is limited to 333 MHz.
- The following ratio requirements need to be met:
  - $s\_axi\_lite\_aclk \leq 2 \times m\_axi\_enc\_aclk$
  - $s\_axi\_lite\_aclk \leq 2 \times m\_axi\_dec\_aclk$

  Refer to Microcontroller Unit Overview for more information on the MCU.

The `VENC_core_clk` is generated based on the VCU PLL.

The `VENC_MCU_clk` is generated based on the VCU PLL.

The `VDEC_core_clk` is generated based on the VCU PLL.

PLL Overview

The VCU core has a PLL for generating encoder and decoder block clocks. Typically, the PLL has an external source such as an Si570 XO programmable clock generator connected directly using an IBUFDS to the PLL reference clock. Alternatively, the IBUFDS may drive an MMCM to enable other modules to share an external clock source while meeting the sub-100ps jitter specification. It is not recommended to use the PS PLL as a clock source due to jitter requirements. The range of the PLL reference clock is 27 MHz to 60 MHz. The PLL generates a high frequency clock that can be divided down to generate various output clock frequencies. The divided clock can be supplied to the encoder block, decoder block, and MCU (separate MCU for video encoder and decoder).

Generation of Primary Clock

The PLL has a Voltage Controlled Oscillator (VCO) block which generates an output clock based on the input reference clock. The output clock from VCO is generated based on a frequency multiplier value. The output clock of the VCO is divided by an output divider to generate the final clock.

VCO Frequency and MF Value

The VCO operating frequency can be determined by using the following relationship:

$f_{vco} = f_{refclk} \times M$

and

$f_{clkout} = f_{vco} / O$

where, *M* corresponds to the integer feedback divide value and *O* corresponds to the value of output divide.

***Note*:** The PLL does not support fractional divider values.

IMPORTANT! *Select the PLL feedback multiplier value based on the supported VCO frequency range (f_{VCO}).*

Refer to the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) for more information on the operating range of f_{VCO}.

Select the output divider (O) based on the required core clock or MCU clock frequency.

CAUTION! *Sharing VCU clock inputs with other IP can result in clock jitter that may degrade VCU performance or image quality.*

Reset Sequence

The state of the VCU during PL power up and the initialization sequence for the VCU are as follows:

- The PL is not yet configured. In this condition, the VCU is held in reset.
- The VCU core is held in reset when the power supplies ramp up. A voltage detector present in the VCU core to PL interface keeps the core under reset while supplies ramp up.
- PL is fully configured. The PL is configured with AXI connectivity between a CPU in the PS or PL and the AXI slave port of the VCU core. The VCU core reset can be released so that the core is in a known state.

After the VCU core reset is de-asserted, use the software to program the VCU PLL for generating the clocks for VCU core and MCU blocks. When programming the VCU PLL, follow the steps described in PLL Integer Divider Programming for programming PLL configuration parameters. The PLL lock status is indicated by `VCU_SLCR`. For additional information, see the *Zynq UltraScale + Device Register Reference* (UG1087).

***Note*:** The VCU core clocks are available while the reset is released. The PL should be configured before releasing the raw reset, which can be controlled by the PMU from outside of the VCU core.

Additional initialization is done by software through programming the VCU core registers after the PL is configured and core is in a reset release state.

PLL Integer Divider Programming

To operate the VCU PLL, configure the `VCU_SLCR.VCU_PLL_CFG` register using the values in the following table. The following fields must be programmed.

- `VCU_SLCR.VCU_PLL_CFG[LOCK_DLY]`
- `VCU_SLCR.VCU_PLL_CFG[LOCK_CNT]`
- `VCU_SLCR.VCU_PLL_CFG[LFHF]`
- `VCU_SLCR.VCU_PLL_CFG[CP]`

- `VCU_SLCR.VCU_PLL_CFG[RES]`

The FBDIV value (or the PLL feedback multiplier value, M) depends on the output VCO frequency (f_{vco}). You must program `VCU_SRCR.VCU_PLL_CFG` based on the calculated FBDIV values in the following table.

Table 35: **PLL Programming for Integer Feedback Divider Values**

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
25	3	10	3	63	1000
26	3	10	3	63	1000
27	4	6	3	63	1000
28	4	6	3	63	1000
29	4	6	3	63	1000
30	4	6	3	63	1000
31	6	1	3	63	1000
32	6	1	3	63	1000
33	4	10	3	63	1000
34	5	6	3	63	1000
35	5	6	3	63	1000
36	5	6	3	63	1000
37	5	6	3	63	1000
38	5	6	3	63	975
39	3	12	3	63	950
40	3	12	3	63	925
41	3	12	3	63	900
42	3	12	3	63	875
43	3	12	3	63	850
44	3	12	3	63	850
45	3	12	3	63	825
46	3	12	3	63	800
47	3	12	3	63	775
48	3	12	3	63	775
49	3	12	3	63	750
50	3	12	3	63	750
51	3	2	3	63	725
52	3	2	3	63	700
53	3	2	3	63	700
54	3	2	3	63	675
55	3	2	3	63	675
56	3	2	3	63	650
57	3	2	3	63	650
58	3	2	3	63	625

Table 35: **PLL Programming for Integer Feedback Divider Values** *(cont'd)*

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
59	3	2	3	63	625
60	3	2	3	63	625
61	3	2	3	63	600
62	3	2	3	63	600
63	3	2	3	63	600
64	3	2	3	63	600
65	3	2	3	63	600
66	3	2	3	63	600
67	3	2	3	63	600
68	3	2	3	63	600
69	3	2	3	63	600
70	3	2	3	63	600
71	3	2	3	63	600
72	3	2	3	63	600
73	3	2	3	63	600
74	3	2	3	63	600
75	3	2	3	63	600
76	3	2	3	63	600
77	3	2	3	63	600
78	3	2	3	63	600
79	3	2	3	63	600
80	3	2	3	63	600
81	3	2	3	63	600
82	3	2	3	63	600
83	4	2	3	63	600
84	4	2	3	63	600
85	4	2	3	63	600
86	4	2	3	63	600
87	4	2	3	63	600
88	4	2	3	63	600
89	4	2	3	63	600
90	4	2	3	63	600
91	4	2	3	63	600
92	4	2	3	63	600
93	4	2	3	63	600
94	4	2	3	63	600
95	4	2	3	63	600
96	4	2	3	63	600
97	4	2	3	63	600

Table 35: **PLL Programming for Integer Feedback Divider Values** *(cont'd)*

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
98	4	2	3	63	600
99	4	2	3	63	600
100	4	2	3	63	600
101	4	2	3	63	600
102	4	2	3	63	600
103	5	2	3	63	600
104	5	2	3	63	600
105	5	2	3	63	600
106	5	2	3	63	600
107	3	4	3	63	600
108	3	4	3	63	600
109	3	4	3	63	600
110	3	4	3	63	600
111	3	4	3	63	600
112	3	4	3	63	600
113	3	4	3	63	600
114	3	4	3	63	600
115	3	4	3	63	600
116	3	4	3	63	600
117	3	4	3	63	600
118	3	4	3	63	600
119	3	4	3	63	600
120	3	4	3	63	600
121	3	4	3	63	600
122	3	4	3	63	600
123	3	4	3	63	600
124	3	4	3	63	600
125	3	4	3	63	600

***Note*:** The range for FBDIV is based on what is supported in silicon. A minimum value of 25 is determined based on the minimum VCO frequency and the maximum input clock frequency. The applicable values for FBDIV are determined by the f_{VCO}, VCO output frequency range. For VCU PLL, the range is 1500-3000 MHz. FBDIV should be determined based on the f_{VCO}, required output frequency and the input clock frequency. For example, for a 27 MHz input frequency and a 666 MHz VCU operating frequency, the possible FBDIV value is 2664/27= 99. Also, the maximum value of FBDIV for 27 MHz will be 3000/27=111. Similarly, the minimum value of FBDIV for 60 MHz will be 1500/60=25.

Reset

The VCU hard block can be held under reset under the following conditions:

- When external reset input `vcu_resetn` signal is asserted.
- During PL configuration.
- When the VCU to PL isolation is not removed.

The VCU reset signal must be asserted for, at least, two clock cycles of the VCU PLL reference clock (the slowest clock input to the VCU). The VCU registers can be accessed after the reset signal is de-asserted.

***Note*:**

- If software resets the VCU block in the middle of a frame, use the software to clear the physical memory allocated for the VCU.
- The reset does not need to be asserted between changes to the VCU configuration during run-time via the VCU control software.
- The `vcu_resetn` signal of Zynq UltraScale+ VCU should be tied to either AXI GPIO or ZynqMP GPIO(EMIO).

The software can program the `VCU_GASKET_INIT` register at offset `0x41074` in the `VCU_SLCR` to assert a reset pulse to the VCU block. Reset VCU using the following procedure:

1. Ensure there is no pending AXI transaction in VCU AXI bus/AXI4-Lite bus.
2. Assert `vcu_resetn` through an EMIO GPIO pin to VCU LogiCORE IP.
3. De-assert `vcu_resetn`.
4. Write 0 to VCU gasket isolation register `VCU_GASKET_INIT[1]` to assert reset to VCU.
5. Write 0 to VCU gasket isolation register `VCU_GASKET_INIT[0]` to enable VCU gasket isolation.
6. Power down VCU supply.

The PLL in the VCU core can be reset through `VCU_SLCR` register which is accessible through AXI4-Lite interface.

Each of the encoder and decoder blocks have register-based soft reset.

Related Information

Common Interface Signals

Clocking and Reset Registers

The following table lists the clocking and reset registers.

Table 36: **Clocking and Reset Registers**

Register	Address	Width	Type	Reset Value	Description
CRL_WPROT	0xA0040020	1	rw	0x00000000	CRL SLCR Write protection register

Table 36: **Clocking and Reset Registers** *(cont'd)*

Register	Address	Width	Type	Reset Value	Description
VCU_PLL_CTRL	0xA0040024	32	mixed[1]	0x0000510F	PLL Basic Control
VCU_PLL_CFG	0xA0040028	32	mixed[1]	0x00000000	Helper data
PLL_STATUS	0xA0040060	32	mixed[1]	0x00000008	Status of the PLLs

Notes:

1. *Mixed* registers are registers that have read only, write only, and read write bits grouped together.

Design Flow Steps

This section describes customizing and generating the core, constraining the core, and the simulation, synthesis, and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
- *Vivado Design Suite User Guide: Designing with IP* (UG896)
- *Vivado Design Suite User Guide: Getting Started* (UG910)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900)

Customizing and Generating the Core

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado® Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

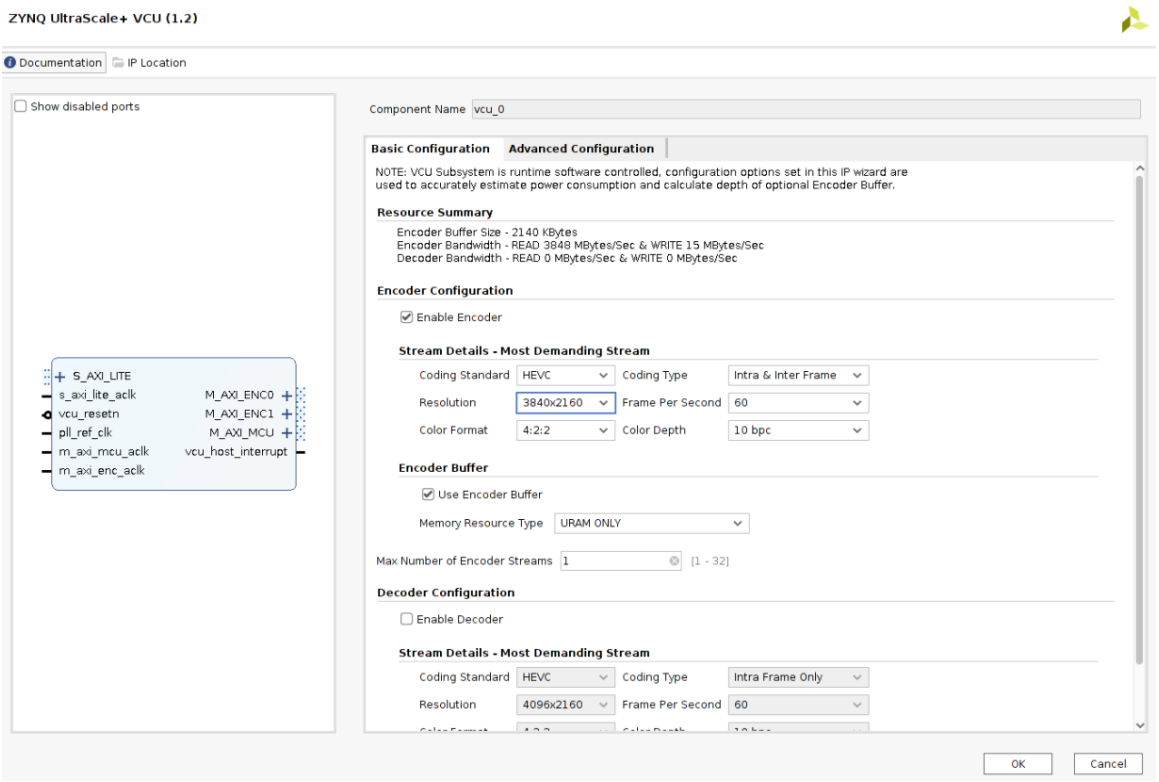
For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

Basic Configuration Tab

The Basic configuration tab, shown in the following figure, allows for the selection of video parameters used to calculate the encoder buffer size and total dynamic power used by encoder or decoder blocks.

Figure 13: **Basic Configuration Tab - Encoder Only Enabled**



The VCU subsystem is controlled by software at runtime. Configuration options set in the VCU GUI are used to estimate power consumption, estimate bandwidth, and calculate the encoder buffer size. See VCU Control Software Sample Applications and Xilinx VCU Control Software API for information about controlling configuration parameters at runtime.

The parameters on the basic configuration tab are as follows:

- **Component Name:** Component name is set automatically by IP integrator.
- **Resource Summary:** Reports the encoder buffer size. Reports bandwidth for the encoder and decoder.
- **Encoder Configuration:** The encoder has the following parameters:
  - **Enable Encoder:** Enables the encoder and related parameters.

- **Maximum Number of Encoder Streams:** Select one to 32 streams. Determines memory requirements.

  ***Note*:** The VCU support 32 streams, but Xilinx recommends choosing the closest combination using the available options in the GUI.

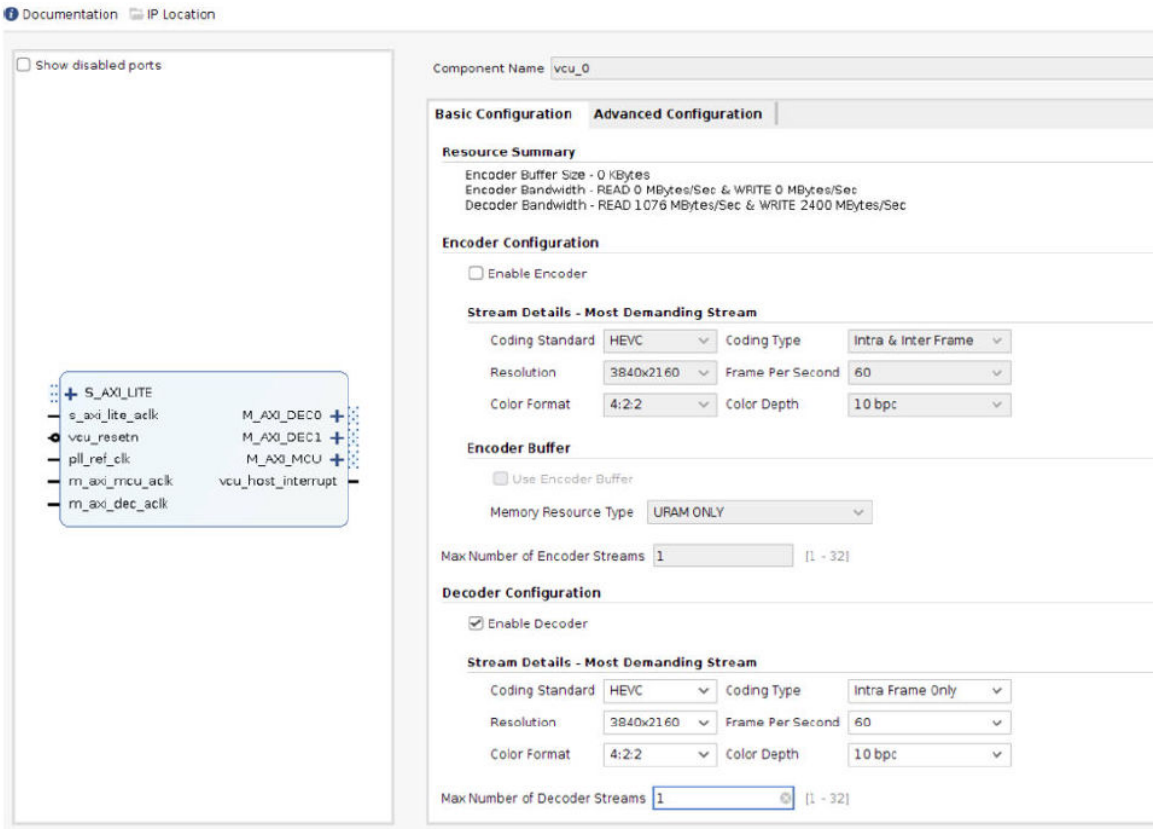
- **Coding Standard:** Select AVC or HEVC.
- **Coding Type:** Select the GOP structure to use for encoding:
  - Intra frame only - I-frame only
  - Intra and inter frame - I-frame, B-frame, and P-frames

  The encoder buffer can only be enabled if intra and inter frame is selected.

- **Resolution:** Select one of the following resolutions:
  - 854×480
  - 1280×720
  - 1420×576
  - 1920×1080
  - 3840×2160
  - 4096×2160
  - 7680×4320
- **Frames Per Second:** Select 15, 30, 45, or 60 fps. At 7680×4320 resolution, only 15 fps is available.
- **Color Format:** Select one of the following color formats:
  - 4:0:0 - monochrome
  - 4:2:0
  - 4:2:2
- **Color Depth:** Select 8 or 10 bits per channel.
- **Use Encoder Buffer:** Select whether or not to use the Encoder Buffer. The Encoder Buffer can only be enabled if intra and inter frame is selected. The Encoder Buffer reduces external memory bandwidth by buffering data in the Programmable Logic; however, it can slightly reduce the video quality.
- **Memory Resource Type:** Select of the following memory type options:
  - UltraRAM only
  - Block RAM only

- Combination of UltraRAM and block RAM

Figure 14: **Basic Configuration Tab - Decoder Only Enabled**



- **Decoder Configuration:** The decoder has the following parameters:
  - **Enable Decoder:** Enables the decoder and associated parameters.
  - **Maximum Number of Decoder Streams:** Select one to 32 streams. Determines memory requirements.

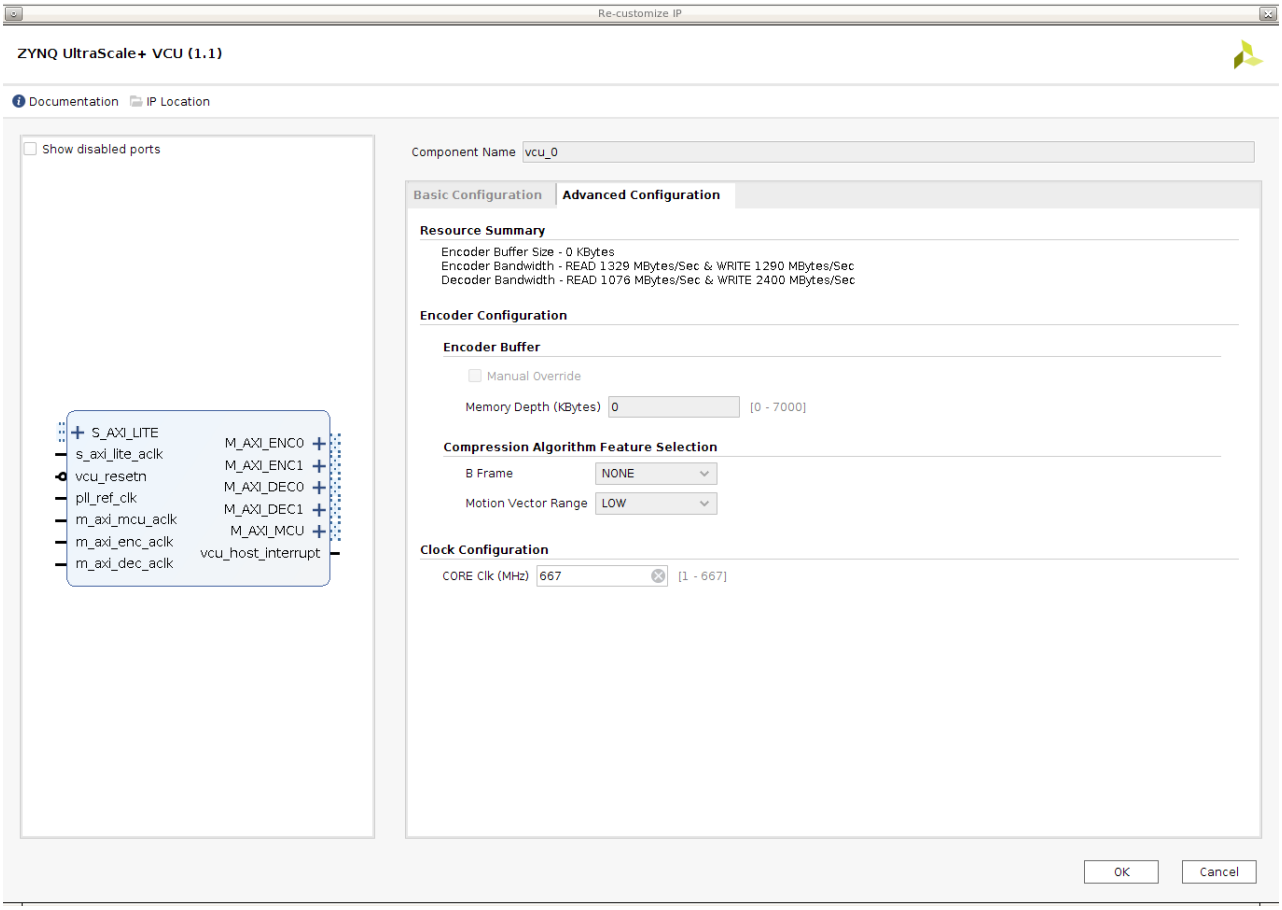
    ***Note*:** The VCU support 32 streams, but Xilinx recommends choosing the closest combination using the available options in the GUI.
  - **Coding Standard:** Select AVC or HEVC.
  - **Resolution:** Select one of the following resolutions:
    - 854×480
    - 1024×576
    - 1280×720
    - 1920×1080
    - 3840×2160

  - 4096×2160
  - 7680×4320
- **Frames Per Second:** Select 15, 30, 45, or 60 fps. At 7680×4320 resolution, only 15 fps is available.
- **Color Format:** Select one of the following color formats:
  - 4:0:0 - monochrome
  - 4:2:0
  - 4:2:2
- **Color Depth:** Select 8 or 10 bits per channel.

Advanced Configuration Tab

The Advanced Configuration tab, shown in the following figure, allows you to override the encoder buffer memory depth, compression features, and the encoder core clock.

Figure 15: **Advanced Configuration Tab**



The advanced configuration options for the encoder buffer are only enabled the Basic Configuration tab has the following settings:

- The Coding Type is Intra and Inter Frame
- Use Encoder Buffer is checked

The parameters on the advanced configuration tab are as follows:

- Manual Override: Select this to override the Encoder Buffer memory size calculated by the IP integrator.
- Memory Depth (Kbytes): If the Manual Override checkbox is selected, you can enter a memory size ranging from 0 to 7,000 Kbytes.
- B-Frame: Select one of the following:
  - NONE - lowest latency
  - STANDARD - GOP configuration IPPP with intra period of 30 ms or GOP configuration IPBBBBPBBBBP with num-b-frames=4.
  - HIERARCHICAL - Also known as pyramidal. Works with 3, 5, or 7 B-frames.

  Standard B-Frame use case has lower write bandwidth requirements because it does not write a reconstructed frame to memory as a reference for subsequent frame encoding.
- Motion Vector Range: Decides the encoder buffer size.:
  - LOW
  - MEDIUM
  - HIGH

  The exact encoder buffer size is reported in resource summary.
- CORE Clk (MHz): Select a clock frequency ranging from 1 to 667 MHz.

Decoder Configuration for Multi-Stream Use Cases

Use the Decoder Configuration tab for a multi-stream use-case of decoding three streams of 1080p60 resolution, for example. The max bandwidth of the VCU is: 1 Stream * 3840 x 2160 fps 60 or 8 Stream * 1920 x 1080 fps 30.

1. Set **Max. Number of Decoder Streams**.
2. Set **Resolution and Frames Per Second** such that the maximum resolution times the maximum number of streams represents the maximum bandwidth plan to use on your system.

   For the case listed above:

   - Set the **Max. Number of Decoded Streams** to three.

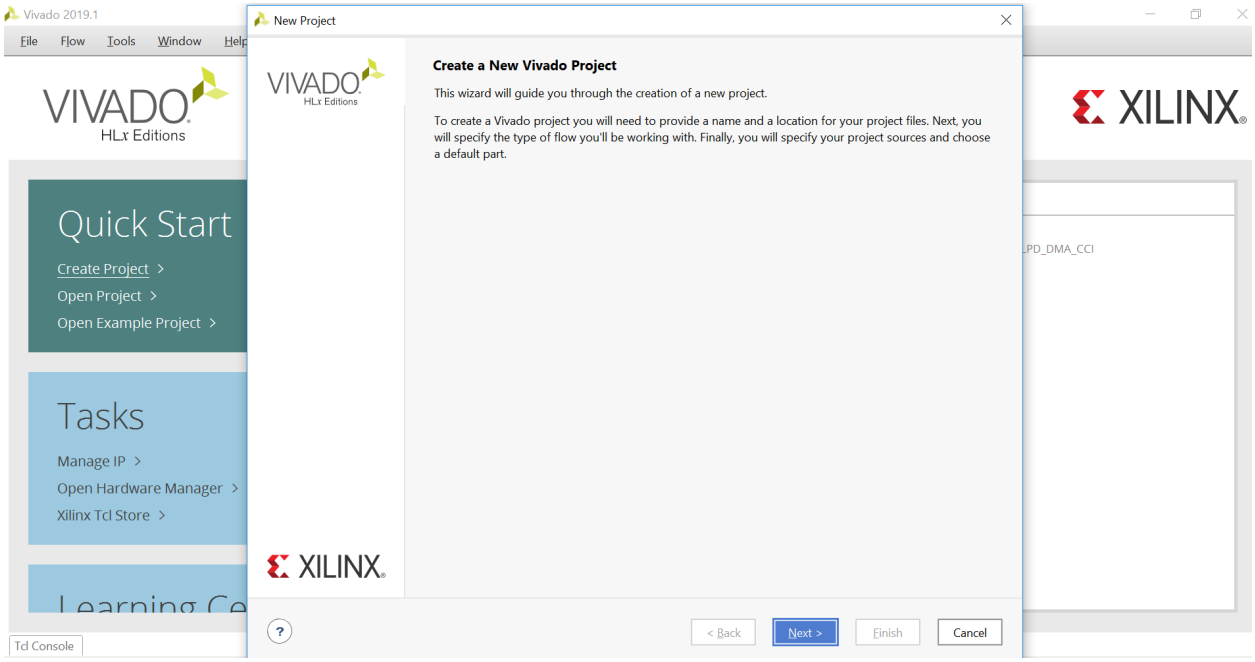
- Set the **Resolution** to 1920x1080.
- Set **Frames Per Second** to 60.

3. You must input the upper range of video parameters (for color format or color depth) if multiple streams use different color formats and color depth.
4. Encoder buffer option is enabled for intra and inter frame coding only. The Encoder buffer is used for motion estimation.
5. GUI configuration is primarily used to calculate the memory bandwidth. As you adjust the values here, you will see the memory requirement at the top of the GUI adjust. If you are using multiple resolutions, you need to make sure to select the maximum resolution you plan to support. For example, if you need to support six channels of 720p60 or four channels of 1080p60 for your application, you should set the GUI to 1080p60 x4 channels, as this is the higher bandwidth application. The resolution for six and four channels are as follows.
   - 720p60 x 6ch = 720*1280*60*6 = 331,776,000
   - 1080p60 x 4ch = 1920*1080*60*4 = 497,664,000

Interfacing the Core with Zynq UltraScale+ MPSoC Devices

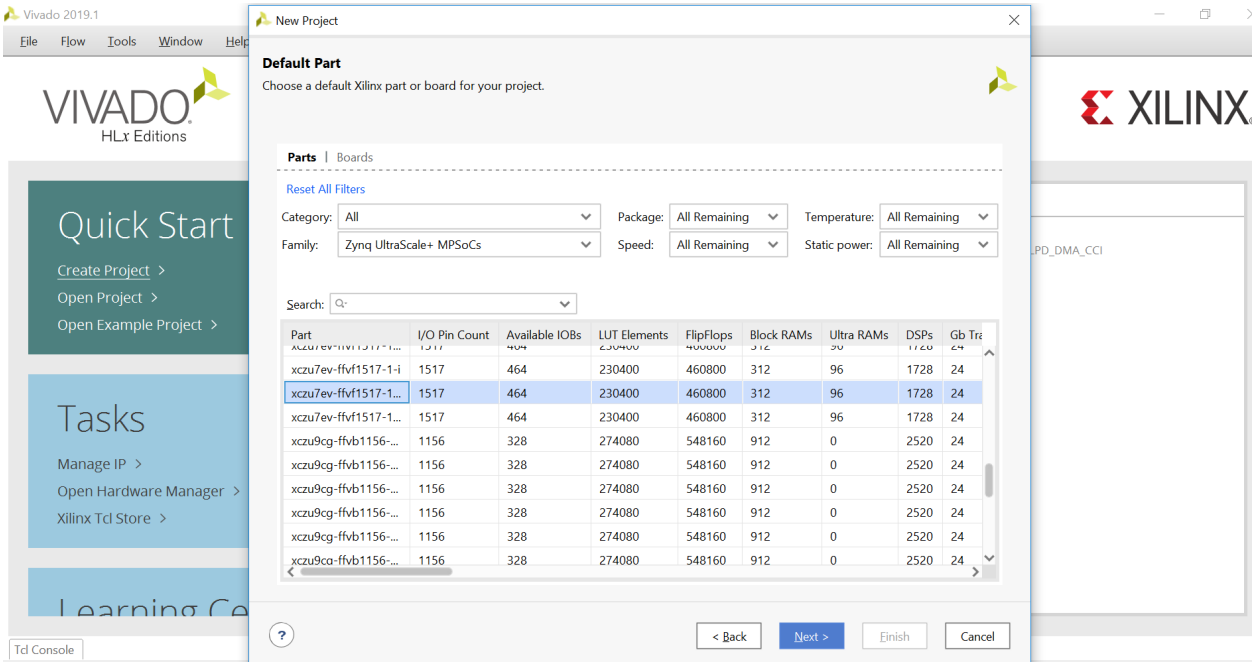
To integrate the VCU core into an IP integrator (IPI) block design, follow these steps:

1. Launch the Vivado IDE and create a new project.

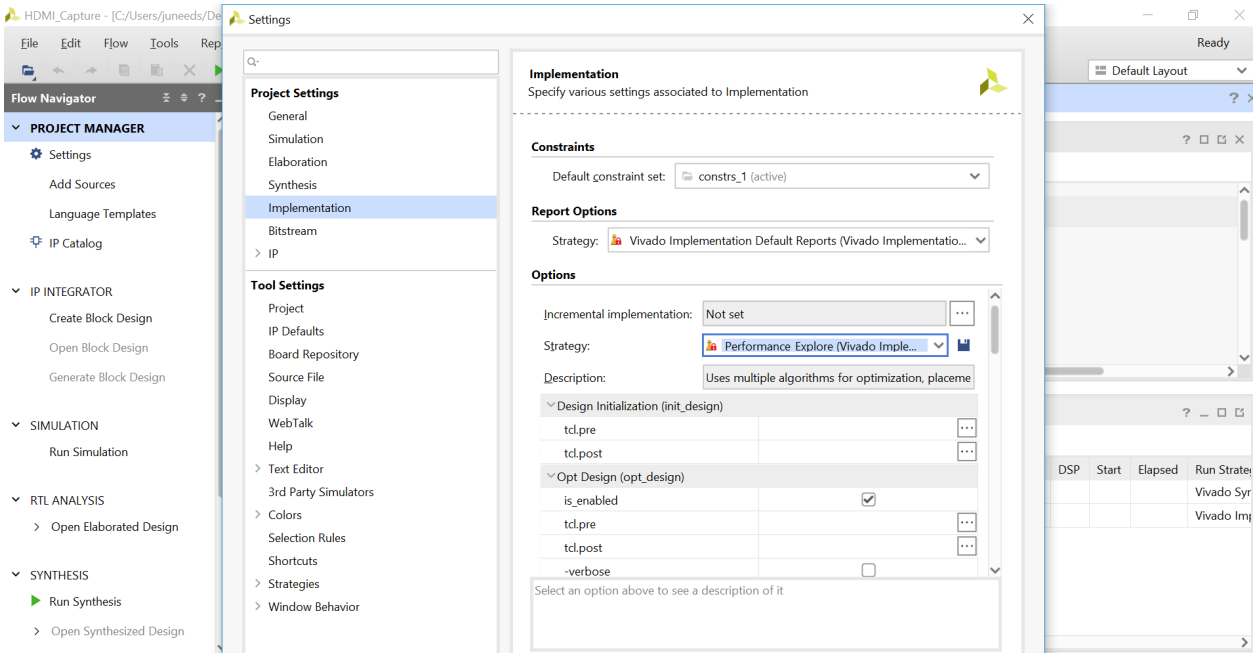


2. Click **Next** on **New Project** wizard until you reach the Family Selection window.

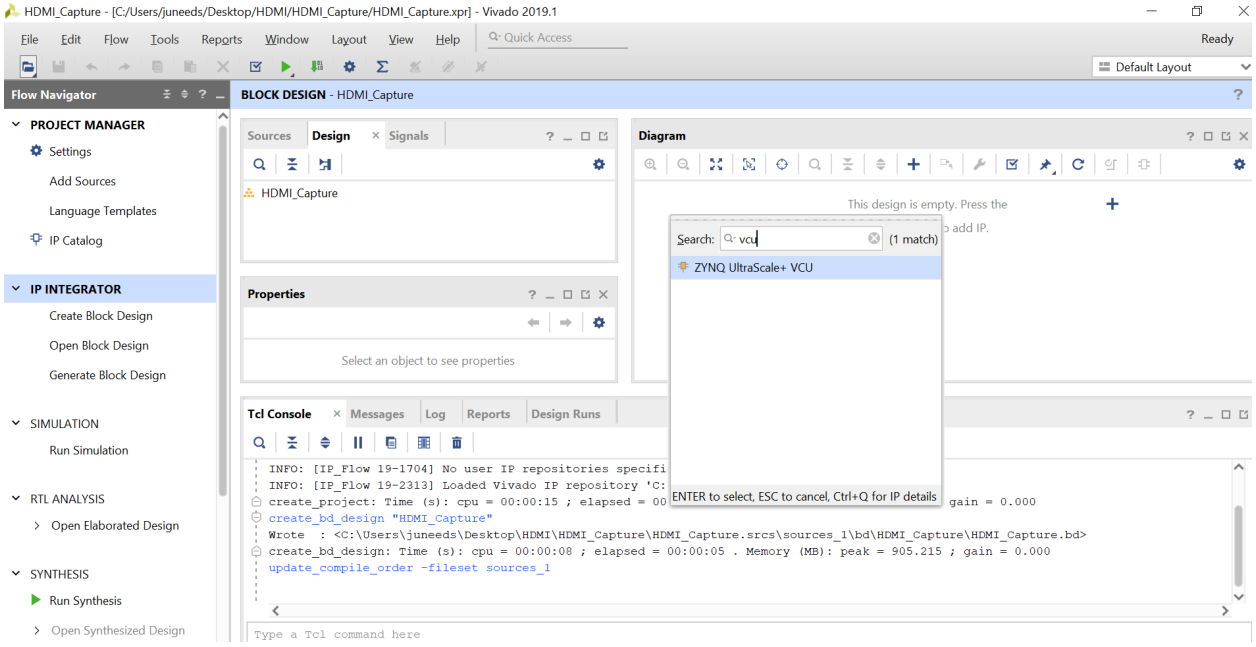
3. Select a target device for the VCU core.



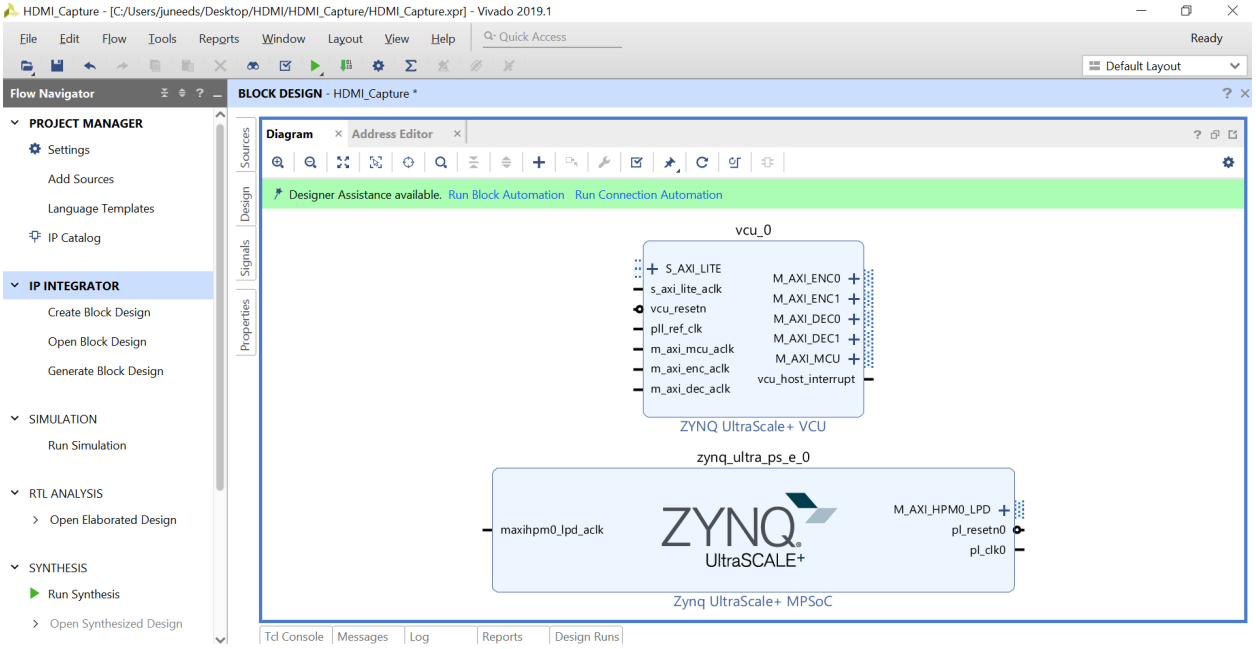
4. Click on the Project Settings window. Click **Implementation**.



5. In the Settings window, enable the Performance_Explore option by selecting **Settings → Implementation → Options → Strategy: Performance_Explore** See the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) for more information.
6. Click **Create Block Design**.
7. Click **Add IP** and type VCU. The following IP appears.



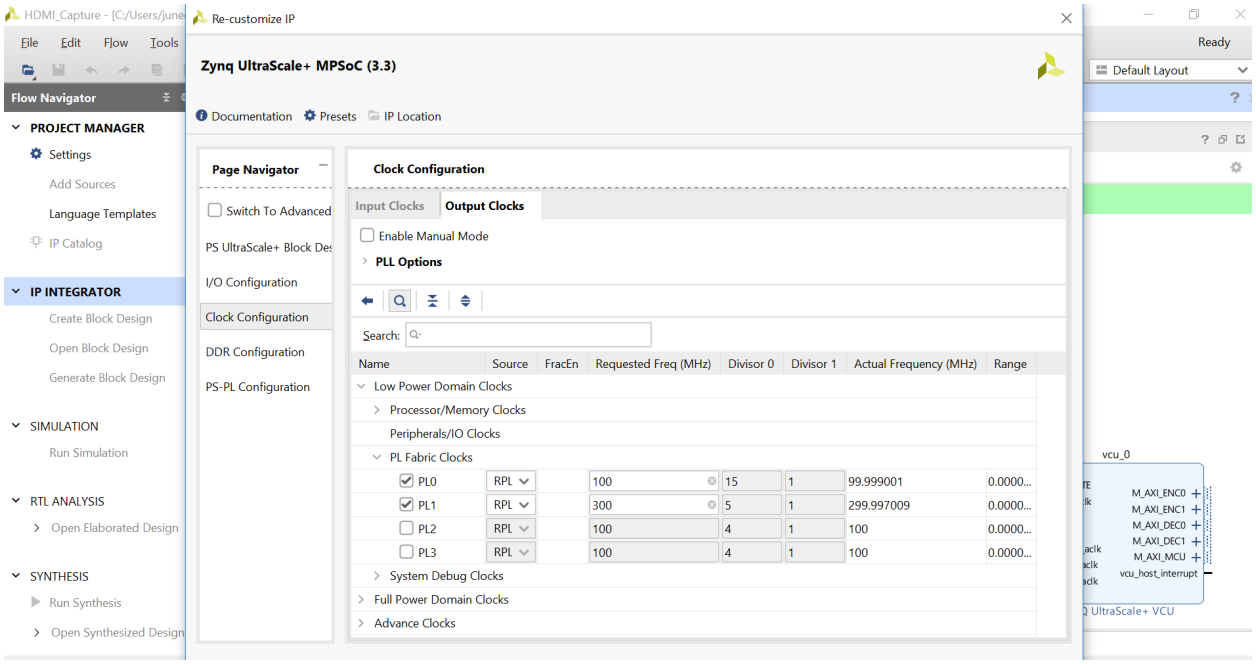
8. Add **Zynq UltraScale+** VCU to the block design.
9. Add **Zynq UltraScale+ MPSoC** IP to the block design as shown.



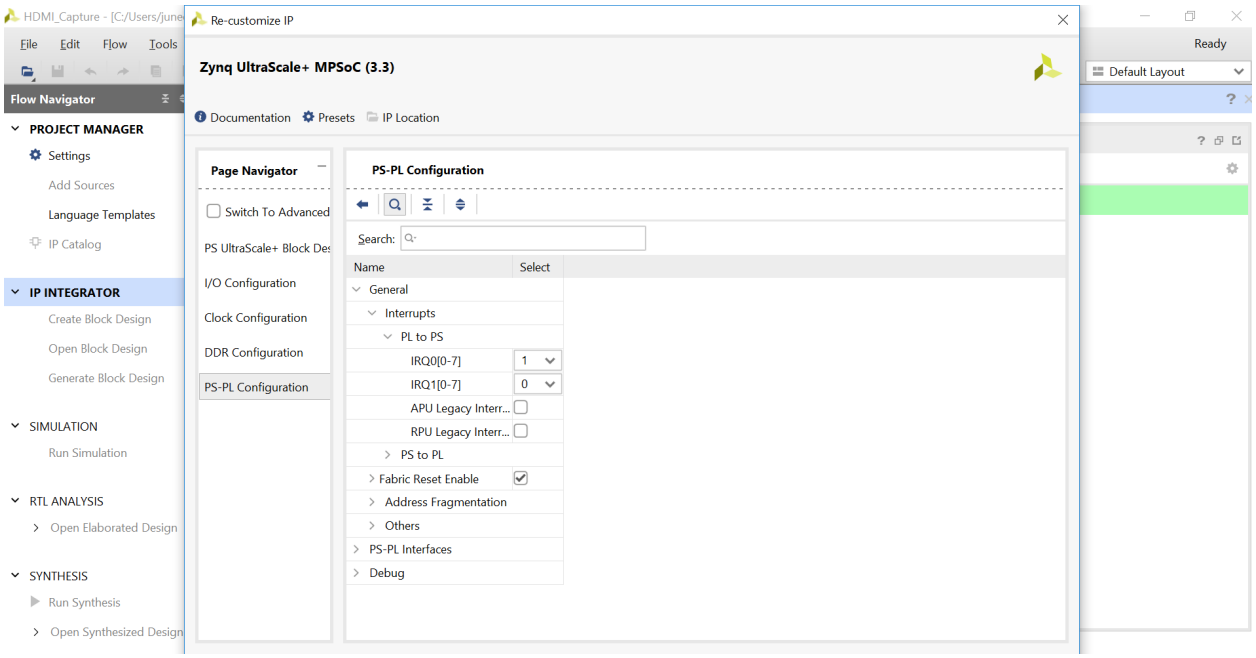
10. Configure Zynq UltraScale+ MPSoC to enable AXI slave interfaces, clocking, and PL-PS interrupt signal per your design requirements. Refer to the *Zynq UltraScale+ MPSoC Processing System LogiCORE IP Product Guide* (PG201) for configuration options of the Zynq UltraScale+ MPSoC IP.

    The following figure shows an example of configuring the PS-PL interface signals.

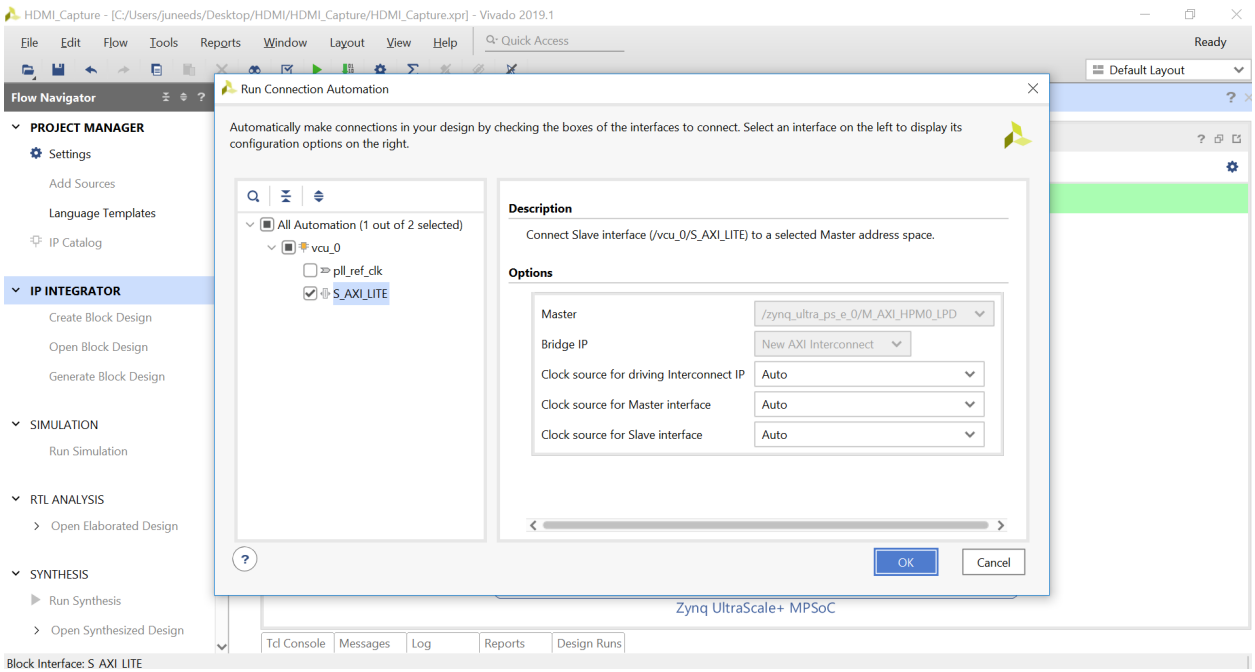
11. Select PL1 clock frequency as 300 MHz.



12. Enable IRQ0 [0-7] and HP0-3 ports.



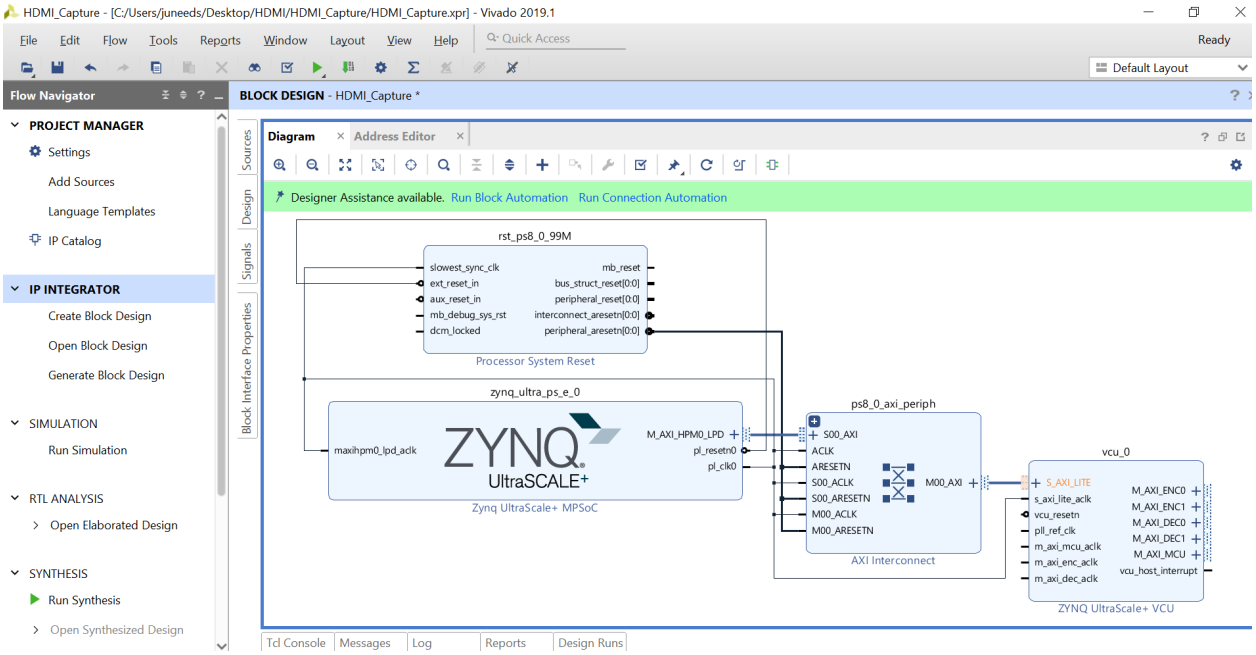
13. Use connection automation to connect the S_AXI_LITE interface of VCU IP to the M_AXI_HPM0_LPD interface.



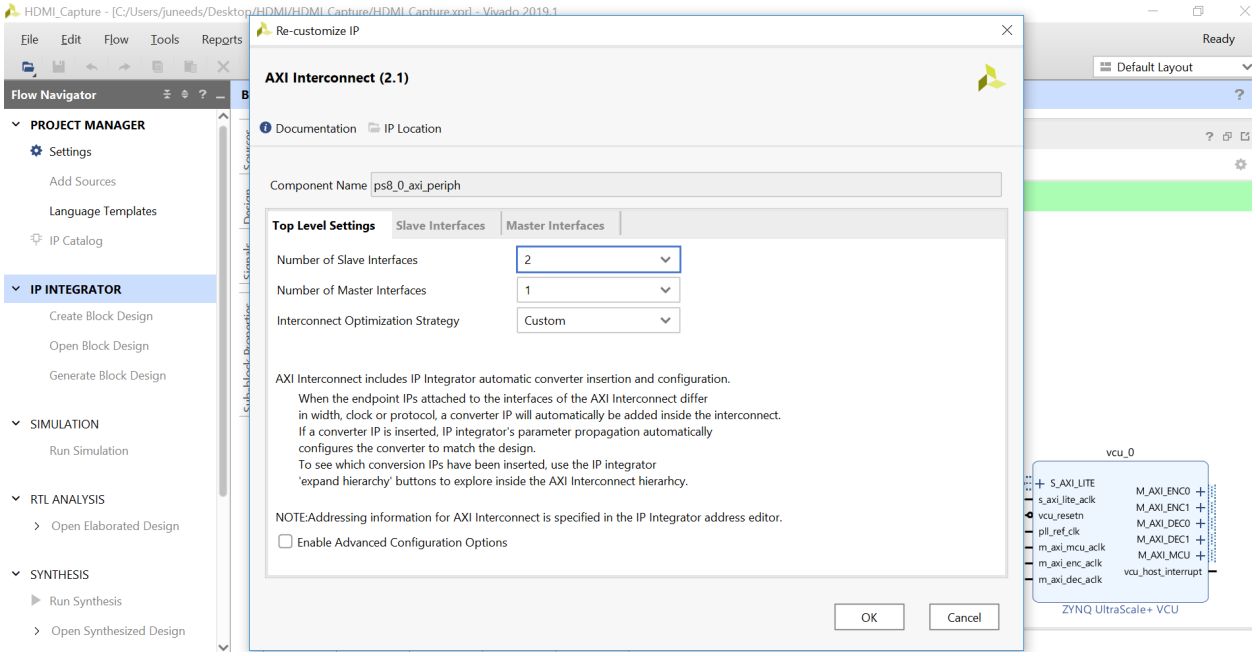
14. Connect the following interfaces manually:
    - Zynq UltraScale+ VCU.M_AXI_ENC1 to Zynq UltraScale+ MPSoC.S_AXI_HP1_FPD
    - Zynq UltraScale+ VCU.M_AXI_DEC0 to Zynq UltraScale+ MPSoC.S_AXI_HP0_FPD
    - Zynq UltraScale+ VCU.M_AXI_DEC1 to Zynq UltraScale+ MPSoC.S_AXI_HP3_FPD

    Note the selection of MPSoC.S_AXI_HP1_FPD, MPSoC.S_AXI_HP0_FPD, and MPSoC.S_AXI_HP3_FPD. These are non-coherent, high-performance DMA ports for large datasets. They support AXI FIFO QoS-400 traffic shaping. For each of these ports, there is an associated register set. The register addresses are needed for command line configuration of quality of service and issuing capability using the `devmem` command.

    The address and description of S_AXI_HP1_FPD can be found in *Zynq UltraScale+ Device Register Reference* (UG1087). The address is `0xFD390000`. The register is used to configure QoS and the FIFO. It is part of the AFIFM Module. The AFIFM Module documentation provides relative addresses and values for fields defining traffic priority and maximum number of read or write commands.

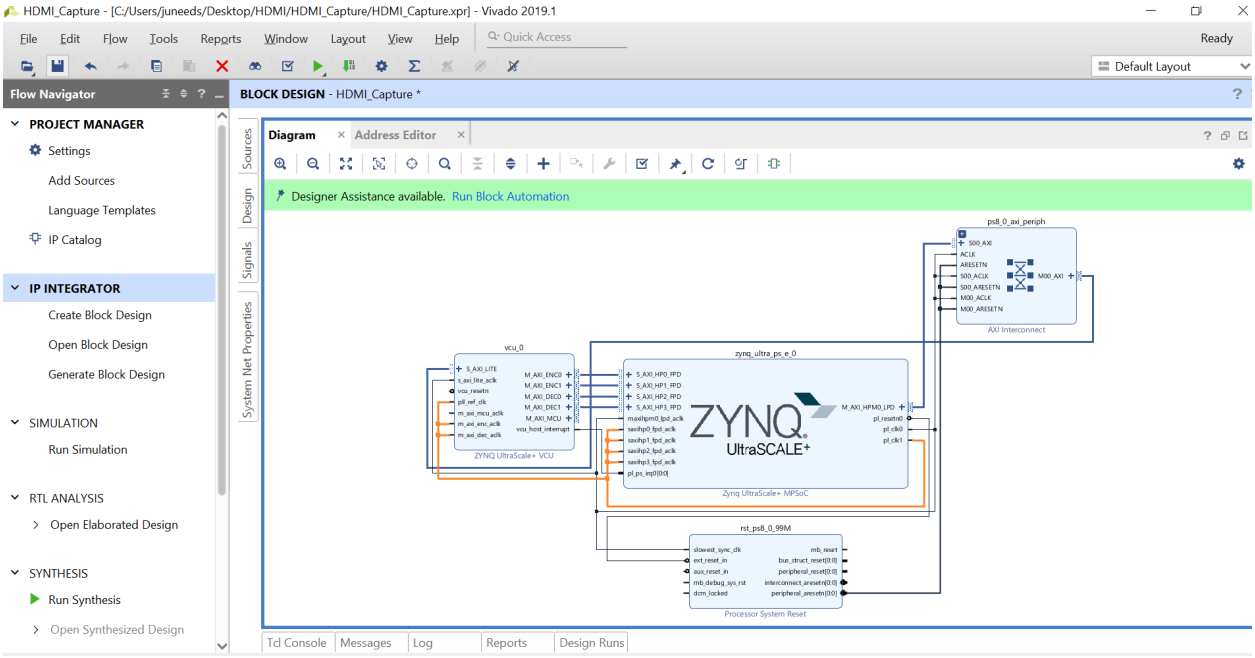


15. Add the AXI Interconnect IP and set number of slave interfaces to 2 and master interface to 1 as shown in the following figure.



16. Perform the following connections manually:
    - Instantiate the processor system reset IP. A second reset block is needed for the `pl_clk1` clock domain.
    - Connect the slowest sync clock to the `pl_clk1` port.

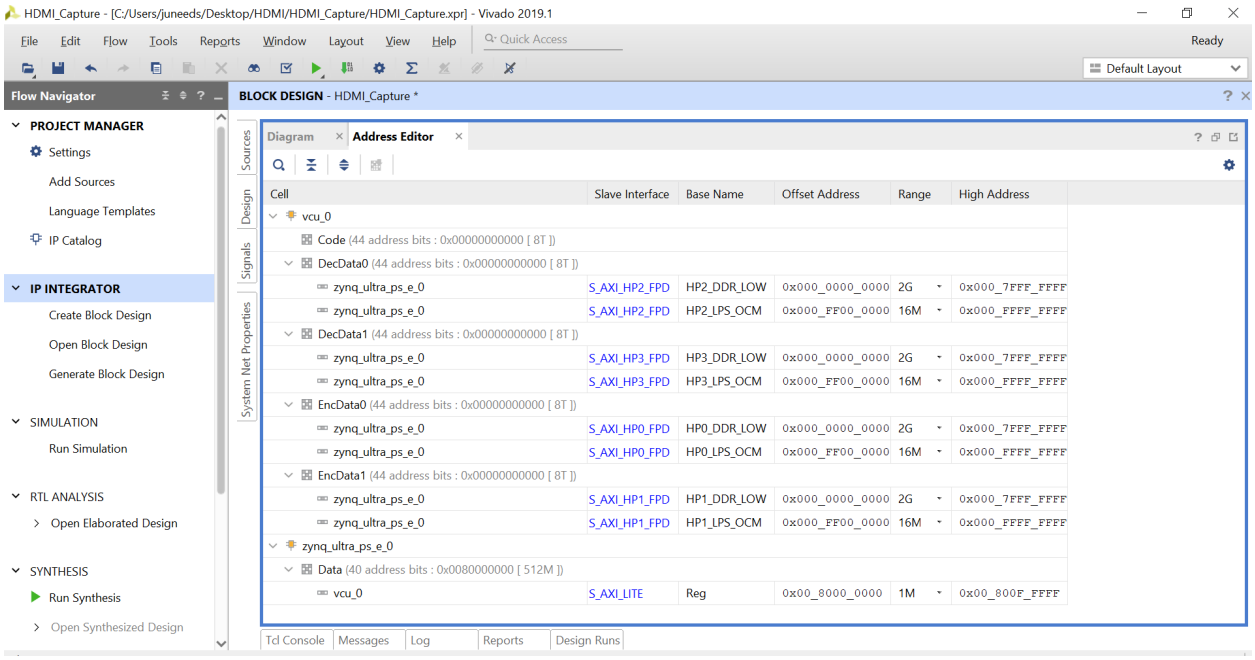
- Use the `interconnect_aresetn` port as the `ARESETN` input to the AXI Interconnect IP core.
- Use `peripheral_aresetn` port as a reset input to the `S00_ARESETN`, `S01_ARESETN`, and `M00_ARESETN` ports as shown in the following figure.



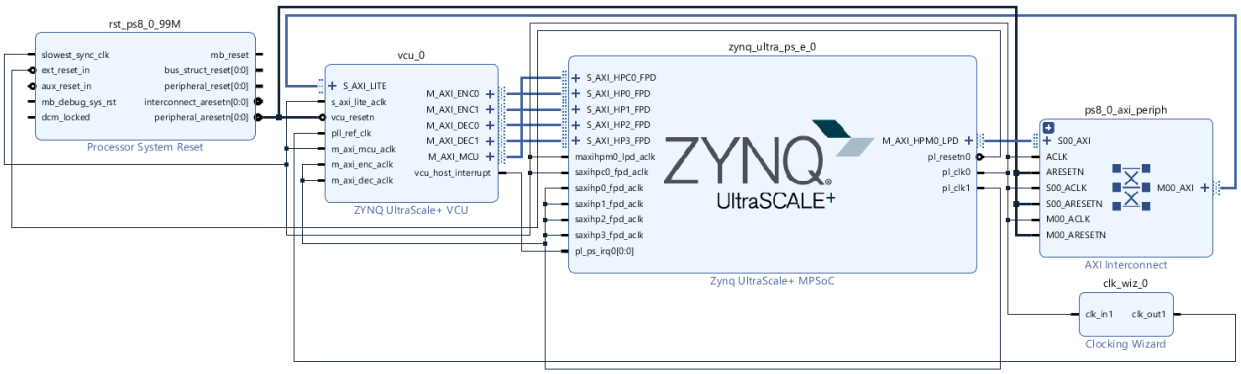
- Connect the `ext_reset_n` signal to the `pl_resetn0` signal of Zynq UltraScale+ MPSoC.
- Connect the `vcu_host_interrupt` to the `pl_ps_irq` port of Zynq UltraScale+ MPSoC IP.

17. Connect up the following clocks to the `pl_clk1` output of Zynq UltraScale+ MPSoC core:
    - AXI Interconnect: `aclk`
    - AXI Interconnect: `s00_aclk`
    - AXI Interconnect: `s01_aclk`
    - AXI Interconnect: `m01_aclk`
    - VCU: `m_axi_mcu_aclk`
    - VCU: `m_axi_enc_aclk`
    - VCU: `m_axi_dec_aclk`
    - Zynq UltraScale+ MPSoC: `saxihp0_fpd_aclk`
    - Zynq UltraScale+ MPSoC: `saxihp1_fpd_aclk`
    - Zynq UltraScale+ MPSoC: `saxihp2_fpd_aclk`
    - Zynq UltraScale+ MPSoC: `saxihp3_fpd_aclk`

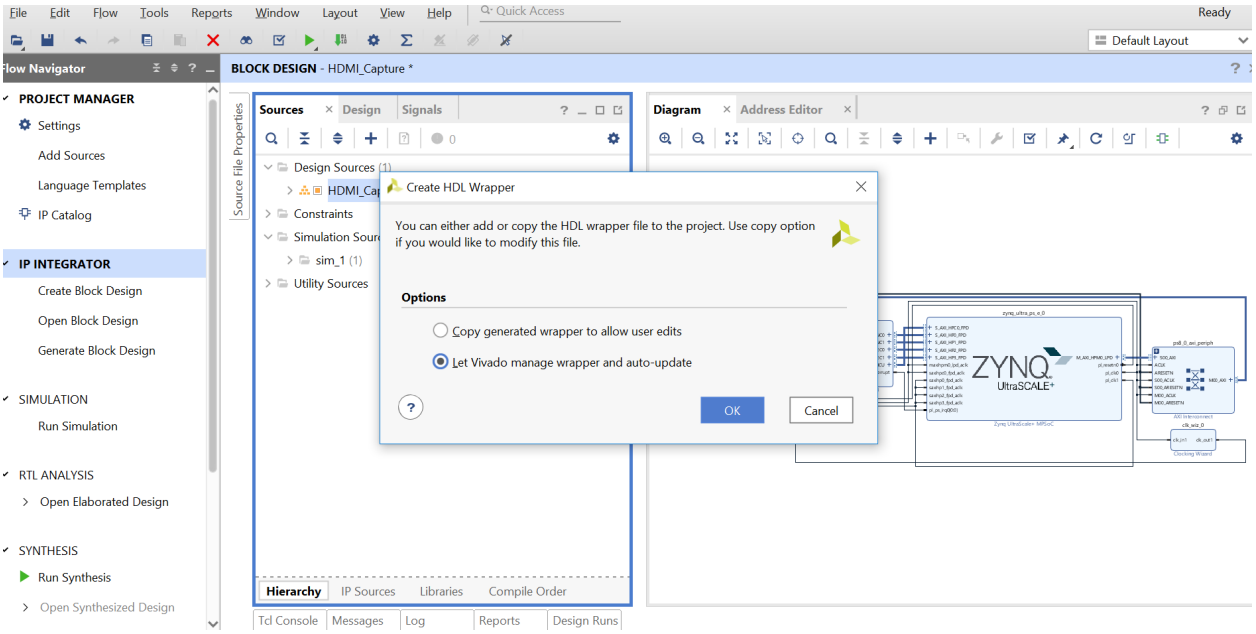
18. Connect `saxihp0_fpd_aclk`, `saxihp1_fpd_aclk`, `saxihp2_fpd_aclk` and `saxihp3_fpd_aclk` to `pl_clk1` output of Zynq UltraScale+ MPSoC core.
19. Tie off the `vcu_resetn` signal of Zynq UltraScale+ MPSoC VCU to either AXI GPIO or ZynqMP GPIO (EMIO).
20. Make `pll_ref_clk` signal as external.
21. In the Address Editor tab, expand EncData address segment and auto assign the addresses. The following table shows an example address map.



22. Click on Validate Block Design to validate the connections.



23. Create a top-level Vivado wrapper by right-clicking on Block Design and selecting Create HDL Wrapper option as shown in the following figure.



24. Add constraints file to the project.
25. Add the constraints file (.xdc) from the board support package if available. If no constraints file is available, several settings must be changed from their default values to enable error-free bitstream generation. In the I/O ports window, for `pll_ref_clk_0`, the I/O Std must be changed from LVCMOS18 (Default) to LVCMOS18.

    And on the same row, the Fixed checkbox must be checked. This corresponds to an XDC file containing the following:

    - `set_property IOSTANDARD LVCMOS18 [get_ports pll_ref_clk_0]`
    - `set_property PACKAGE_PIN AA2 [get_ports pll_ref_clk_0]`
26. Click on the Run Synthesis, Run Implementation, or Generate Bitstream option.

Enabling PL-DDR for VCU

Use-case 1 (UC1) refers to the multimedia pipeline, where decoder and encoder are using PS_DDR for buffer allocations and memory read/write operations of video processing. The current system is capable of encoding and decoding of 4k@60 fps and transcoding of 4k@30 fps with the available bandwidth of PS_DDR. The target is to achieve transcoding at 4k@60fps and it has been identified that the PS_DDR bandwidth is the bottleneck. A new design has been proposed to overcome PS_DDR bandwidth limitations.

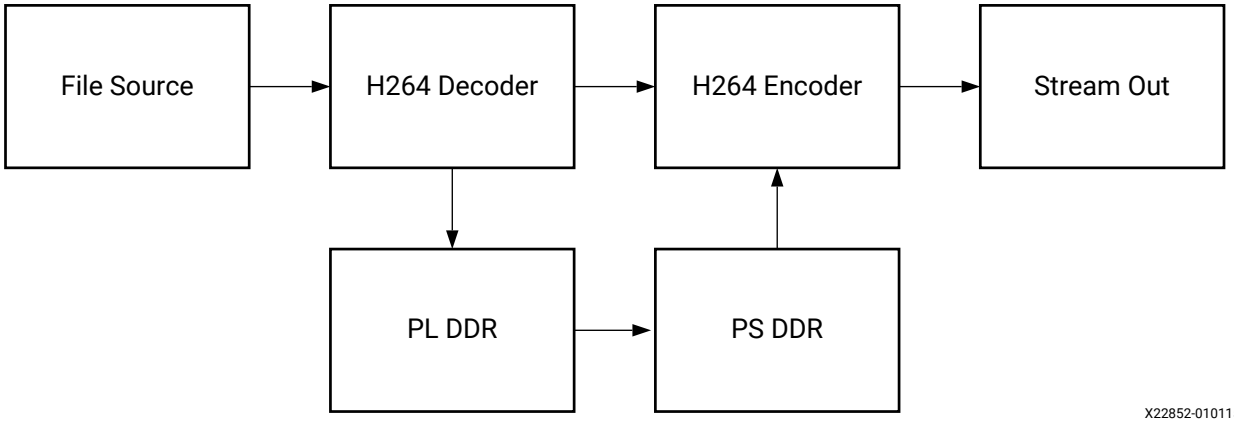
Use-case 2 (UC2) is the new design approach proposed to use PL_DDR for decoding and PS_DDR for encoding, so that the DDR bandwidth would be sufficient to achieve transcoding at 4k@60fps. The figure below explains the transcoding pipeline. The decoder completes the decoding process and writes the data to PL_DDR and the same is copied to PS_DDR from where the encoder consumed the data. The buffer copy from PL_DDR to PS_DDR is achieved by DMA transfers.

2 GB Access Limit

The following access limits apply:

- VCU IP can access 4 GB range from aligned 4 GB base address and MCU can access buffers within 2 GB range only from dcache offset.
- The MCU requires a certain buffer access during encode/decode process, so the buffers that are accessed by MCU should be within 2 GB range from dcache offset.

Figure 16: **Use Case 2**



Build Steps

To enable PL-DDR:

1. Extract the PetaLinux BSP.
2. Update required notes in `system-user.dtsi`:
   a. The UC2 design requires two Frame Buffer IPs for DMA transfers. The video_m2m device node constructs the video mem2mem pipeline. The dtsi changes are as follows:

```
&amba_pl {
    video_m2m {
    compatible = "xlnx,mem2mem";
    dmas = <&v_frmbuf_rd_0 0>, <&v_frmbuf_wr_0 0>;
    dma-names = "tx", "rx";
    };
};
```

b. Update the device tree node to have a dedicated memory for the VCU.

- <=19.2 dtsi changes:

```
&vcu_ddr4_controller_0
{
    compatible = "xlnx,ddr4-2.2";
    reg = <0x00000048 0x00000000 0x0 0x80000000>;
    ranges;
    #address-cells = <2>;
    #size-cells = <2>;
    plmem_vcu: pool@0
         {
        reg = <0x48 0x00000000 0x0 0x70000000>;
        };
};
```

- >=20.1 dtsi changes

```
/ {
    reserved-memory {
        #address-cells = <0x2>;
        #size-cells = <0x2>;
        ranges;

        plmem_vcu: vcu_dma_mem_region {
        compatible = "shared-dma-pool";
        no-map;
        reg = <0x48 0x0 0x0 0x70000000>;
        };
    };
};"
```

c. The VCU device tree node changes to allocate memory from PL_DDR.

- If decoder is connected to PL-DDR:
  - <=19.2 dtsi changes:

```
&decoder {
    xlnx,dedicated-mem = <&plmem_vcu>;
};
```

  - >=20.1 dtsi changes:

```
&decoder {
        memory-region = <&plmem_vcu>;
};
```

- If encoder is connected to PL-DDR:
  - <=19.2 dtsi changes:

```
&encoder {
    xlnx,dedicated-mem = <&plmem_vcu>;
};
```

- >=20.1 dtsi changes:

```
&encoder {
        memory-region = <&plmem_vcu>;
};
```

***Note*:** These changes are relevant to the ZCU106 and ZCU104 designs only. You should modify your changes based on the specific design that you are working with.

This generates the device tree in the path: `components/plnx_workspace/device-tree/device-tree`

`vi project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`

The contents of the file for >= 20.1 UC2 are:

```
/include/ "system-conf.dtsi"
/ {
                reserved-memory {
                                #address-cells = <0x2>;
                                #size-cells = <0x2>;
                                ranges;

                                plmem_vcu_dec: vcu_dma_mem_region {
                                                compatible = "shared-dma-
pool";
                                                no-map;
                                                reg = <0x48 0x0 0x0
0x70000000>;
                                };
                };
};
&amba_pl {
                video_m2m {
                                compatible = "xlnx,mem2mem";
                                dmas = <&v_frmbuf_rd_0 0>,
<&v_frmbuf_wr_0 0>;
                                dma-names = "tx", "rx";
                };
};
&vcu_ddr4_controller_0 {
                compatible = "xlnx,ddr4-2.2";
                reg = <0x00000048 0x00000000 0x0 0x80000000>;
                ranges;
                #address-cells = <2>;
                #size-cells = <2>;
};
&decoder {
                memory-region = <&plmem_vcu_dec>;
};
```

3. Build the images `petalinux-build`.
4. Package `Boot.bin` using the following command.

```
petalinux-package --boot --fsbl zynqmp_fsbl.elf --u-boot u-boot.elf --
pmufw pmufw.elf --fpga system.bit
```

Steps to Run The Commands

1. Boot using the above images and run the following commands for QoS settings:

```
devmem 0xfd3b0008 w 0x3 #DEC0, DEC1<->HP3_FPD
devmem 0xfd3b001c w 0x3
devmem 0xfd3b0004 w 0xf
devmem 0xfd3b0018 w 0xf
devmem 0xfd390008 w 0x3 #ENC1<->HP1_FPD
devmem 0xfd39001c w 0x3
devmem 0xfd390004 w 0xf
devmem 0xfd390018 w 0xf
devmem 0xfd3a0008 w 0x3 #ENC0<->HP2_FPD
devmem 0xfd3a001c w 0x3
devmem 0xfd3a0004 w 0xf
devmem 0xfd3a0018 w 0xf
devmem 0xfd360008 w 0x3 #MCU<->HPC0
devmem 0xfd36001c w 0x3
devmem 0xfd360004 w 0x7
devmem 0xfd360018 w 0x7
```

2. Set XV20 for mem2mem video node:

```
v4l2-ctl -d /dev/video0 --set-fmt-video=width=3840, height=2160,
pixelformat='XV20'
```

***Note*:** The format will vary based on the format that you are using. For example:

- XV20 when using 4:2:2 10-bit
- NV12 when using 4:2:0 8-bit
- **GStreamer Pipeline Example for Measuring Performance:**

```
gst-launch-1.0 filesrc location="/run/1600frames.h264" ! h264parse !
queue ! omxh264dec internal-entropy-buffers=7 ! queue max-size-
bytes=0 ! v4l2convert output-io-mode=5 capture-io-mode=4 disable-
passthrough=1 import-buffer-alignment=true! omxh265enc num-slices=8
prefetch-buffer=true ! fpsdisplaysink name=fpssink text-overlay=false
video-sink="fakesink sync=false" sync=false -v
```

- **GStreamer Pipeline to Run Transcode (Decode to Encode):**

```
gst-launch-1.0 filesrc location="/run/1600frames.h264"! h264parse!
queue! omxh264dec internal-entropy-buffers=7! queue max-size-bytes=0!
v4l2convert output-io-mode=5 capture-io-mode=4 disable-passthrough=1
import-buffer-alignment=true! omxh265enc num-slices=8 prefetch-
buffer=true! filesink location="op.h265"
```

***Note*:** A V4L2convert is required for the transcode usecase when encoder and decoder are using different DDRs. For example, the encoder is using PS and the decoder is using PL. These configuration of using separate DDR is used to achieve performance.

Constraining the Core

The necessary XDC constraints are delivered with the core generation in the Vivado Design Suite.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

***Note*:** 4K (3840x2160) and below is supported in all speedgrades and 4K DCI (4096x2160) requires -2 or -3 speedgrade.

Clock Frequencies

There is no restriction for speed grade. All speed grades support the maximum frequency of operation.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

Simulation

Simulation of the H.264/H.265 Video Codec Unit is not supported.

XILINX

Section II

Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP v1.1

Introduction

The Xilinx® Zynq® UltraScale+™ MPSoC architecture-based FPGAs VCU DDR4 IP v1.1 core is a combined pre-engineered controller and physical layer (PHY) for interfacing Zynq UltraScale+ MPSoC programmable logic (PL) user designs to DDR4 SDRAM. This DDR4 Controller is only for use with the Zynq UltraScale+ MPSoC EV products and not for us with any other Xilinx devices.

This chapter provides information about using, customizing, and simulating a LogiCORE™ IP DDR4 SDRAM for Zynq UltraScale+ MPSoCs. It also describes the core architecture and provides details on customizing and interfacing to the core.

IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq® UltraScale+™ MPSoC EV Devices
Supported User Interfaces	AXI4
Resources	See Performance and Resource Utilization
Provided with Core	
Design Files	RTL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Xilinx Constraints File (XDC)
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite
Synthesis	Vivado® Design Suite
Support	
Release Notes and Known Issues	Master Answer Record: 76600
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado® IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The Xilinx UltraScale+ architecture includes the DDR4 SDRAM cores. These cores provide solutions for interfacing with these SDRAM memory types. Both a complete Memory Controller and a physical layer only solution are supported. This controller is optimized for the VCU traffic patterns, specifically the decoder accesses to memory. The UltraScale+ architecture for the DDR4 cores are organized in the following high-level blocks:

- **Controller:** The controller accepts burst transactions from the user interface and generates transactions to and from the SDRAM. The controller takes care of the SDRAM timing parameters and refresh. It coalesces write and read transactions to reduce the number of dead cycles involved in turning the bus around. The controller also reorders commands to improve the usage of the data bus to the SDRAM.

- **Physical Layer:** The physical layer provides a high-speed interface to the SDRAM. This layer includes the hard blocks inside the FPGA and the soft blocks calibration logic necessary to ensure optimal timing of the hard blocks interfacing to the SDRAM. The application logic is responsible for all SDRAM transactions, timing, and refresh.

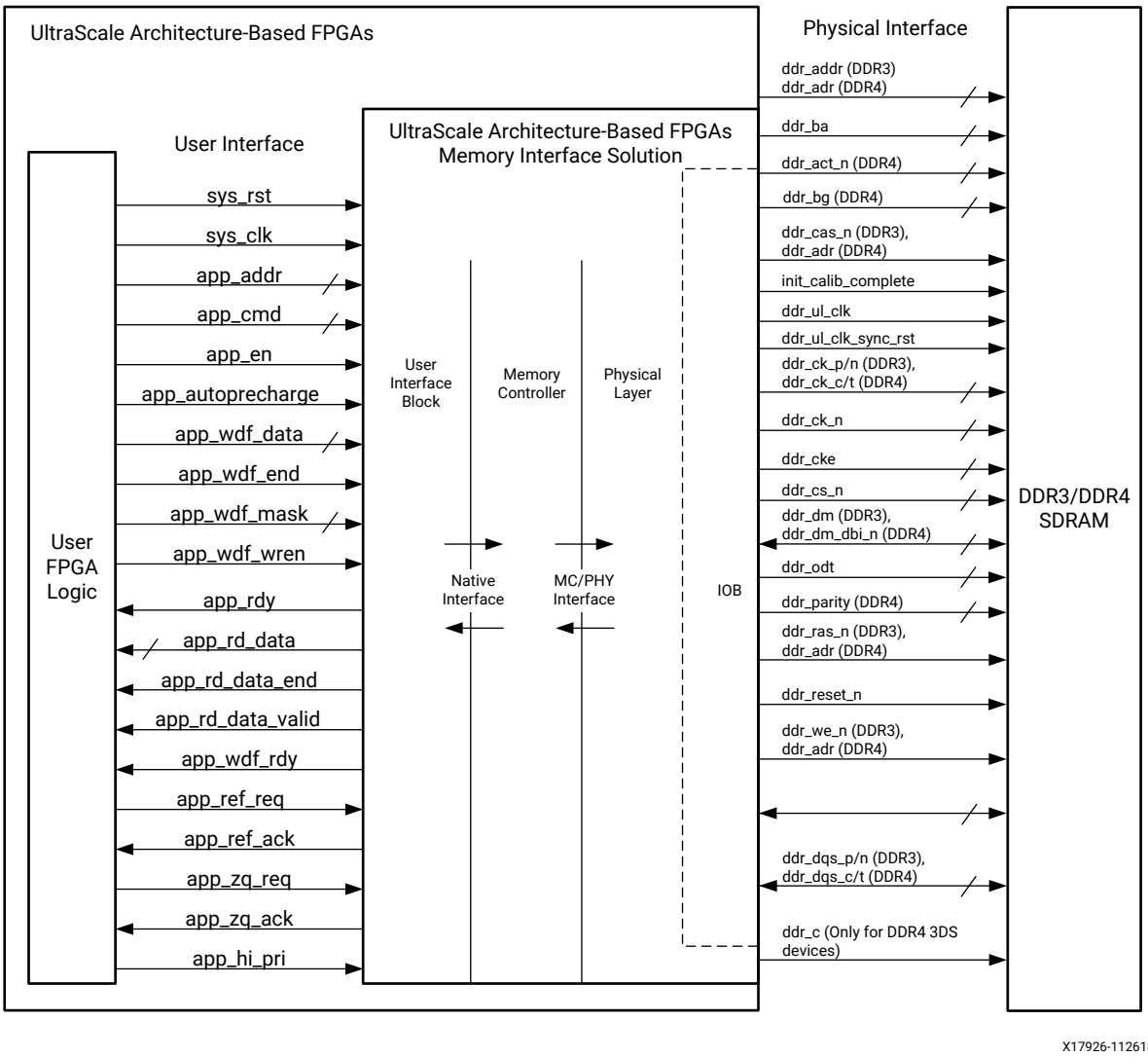
These hard blocks include:

- Data serialization and transmission
- Data capture and deserialization
- High-speed clock generation and synchronization
- Coarse and fine delay elements per pin with voltage and temperature tracking

The soft blocks include:

- **Memory Initialization:** The calibration modules provide a JEDEC®-compliant initialization routine for the particular memory type. The delays in the initialization process can be bypassed to speed up simulation time, if desired.
- **Calibration:** The calibration modules provide a complete method to set all delays in the hard blocks and soft IP to work with the memory interface. Each bit is individually trained and then combined to ensure optimal interface performance. Results of the calibration process are available through the Xilinx debug tools. After completion of calibration, the PHY layer presents raw interface to the SDRAM.

Figure 17: **UltraScale+ Architecture-Based FPGAs DDR4 Memory Interface Solution**



DDR4 DSDRAM Feature Summary

IMPORTANT! *Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP can only be used with the H.264/H.265 Video Codec Unit (VCU) core for Zynq UltraScale+ MPSoCs.*

- Supports five high performance AXI ports for connecting to decoder 0, decoder 1, MCU, PS, and display controller interface
- Component/SODIMM support for interface width of 64-bits
  - Supports speed bins of 2133, 2400, and 2667

- Support for Zynq-UltraScale+ EV series -1e , -2 , -3e parts and supported frequencies 2133 MT/s, 2400 MT/s, 2667 MT/s.
- Does not support speed grades –1L and –1LV.
- Supports AXI

See Table 50: Supported Configuration for a complete list of supported memories.

- Reorders FIFO for better efficiencyx8 and x16 device support
- 8-word burst support
- ODT support
- Write leveling support for DDR4 (fly-by routing topology required component designs)
- JEDEC-compliant DDR4 initialization support
- Encrypted source code delivery in Verilog/VHDL
- Open, closed, and transaction based pre-charge controller policy
- Interface calibration and training information available through the Vivado hardware manager
- Target technologies (physical interface):
  - Using MIG generated phy-only design
  - Zynq UltraScale+ MPSoC
  - DDR4 features are supported
- Controller:
  - High efficiency is achieved for multi-port and random access applications through
  - Reordering
  - Burst maximization
  - Deep lookahead
  - Ping pong ss
  - High frequency can be achieved through configurable pipes
  - Low resource count

Licensing and Ordering

This Xilinx LogiCORE IP module is provided at no additional cost for Zynq UltraScale+ MPSoC EV devices with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License. Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

This core supports DRAMs that are compliant to the JESD79-4, DDR4 SDRAM Standard, JEDEC Solid State Technology Association.

Performance

System Throughput

The following table shows memory controller performance for running a 4kp60, 4:2:2, 10-bit decode-display pipeline. These throughput numbers are based on x8 configuration of the controller at 2133 DRAM speed.

Table 37: **Memory Controller Performance**

Bandwidth	Decoder port 0	Decoder port 1	Display	Video traffic generator in PL
Read bandwidth	1879.78 MB/s	1913.62 MB/s	1328.09 MB/s	950.72 MB/s
Write bandwidth	895.32 MB/s	792.01 MB/s	N/A	N/A

The following table summarizes performance in decoded images/sec for decoding a 4kp60 video stream, 4:2:2, 10-bit, using four B-frames.

Table 38: **Decoding Performance**

Speed	Frame-rate Decode Only	Frame-rate Decode + Display
X8	93 frames/sec	74 frames/sec
X16	89 frames/sec	70 frames/sec

VCU DDR4 Controller Latency

***Note*:** These numbers do not include the latency information from the interconnect in the fabric. For more information, see the *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150).

Latency number for the access times in "VCU DDR4 Controller" is 130 ns on SODIMM MTA8ATF51264HZ-2G6B1 @ 2400.

Resource Utilization

The following table shows details about performance and resource utilization.

Table 39: **VCU DDR4 UTILIZATION**

Name	vcu_ddr4_controller_0 (vcu_llp2_trd_vcu_ddr4_controller_0_0)	vcu_0 (vcu_llp2_trd_vcu_0_0)
CLU LUTS	23066	434
CLB Registers	25270	1681
CARRY8	170	0
F7 MUXES	1147	0
BLOCK RAM TILE	123	0
DSPs	3	0
HPIOBDIFFINBUF	9	0
BITSLICE_RX_TX	105	0
GLOBAL CLOCK BUFFERS	5	0
PLL	3	0
MMCM	1	0

The following table illustrates the resource utilization for VCU DDR4 Controller based on the vcu_llp2_trd_vcu_ddr4_controller_0_0.

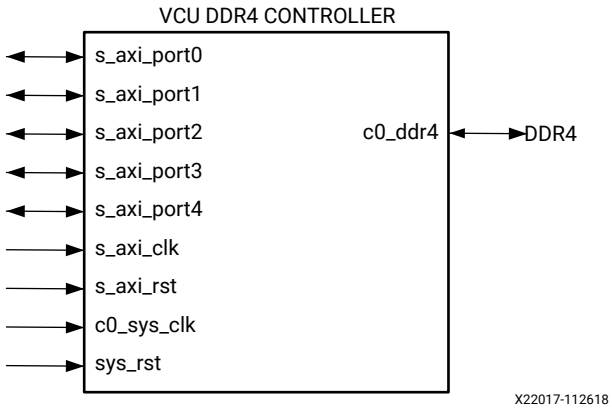
Table 40: **Resource Utilization for VCU DDR4 Controller Based on vcu_llp2_trd_vcu_ddr4_controller_0_0**

Memory	Usage
Block RAM only	Block RAM: 123
Block RAM + UltraRAM	Block RAM: 43, UltraRAM: 80
UltraRAM only	Not Supported

Port Descriptions

The following figure shows the block diagram of VCU DDR4 Controller which has five AXI ports, `s_axi_clk`, `s_axi_rst`, `c0_sys_clk`, and `sys_rst`.

Figure 18: **AXI Clock Port / Reset Port Connection**



S_AXI_CLK / S_AXI_RST: The AXI ports work with respect to this clock and reset. Active-Low reset is used.

C0_SYS_CLK / SYS_RST: This is the actual clock used for VCU DDR4 controller, which is of frequency 125 MHz for x16 configuration and 300 MHz for x8 configuration. Active-Low reset is used.

AXI Ports

The AXI specification is available (after registration) on http://www.arm.com/products/system-ip/amba/amba-open-specifications.php. The AXI ports of the BA317 match AXI4 specification. However, they do not perform reordering. They support all burst types (narrow transfer, incremental/wrapping/fixed bursts).

The following tables represents the AXI port numbers.

Table 41: **AXI Write Ports**

Signal	Direction	Description
Write Address Channel		
pl_cust_slot_awidX [15:0]	In	
pl_cust_slot_awaddrX [31:0]	In	Byte address
pl_cust_slot_awlenX [7:0]	In	Length of burst (number of transfer minus 1)
pl_cust_slot_awsizeX [2:0]	In	Transfer width: 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit

Table 41: **AXI Write Ports** *(cont'd)*

Signal	Direction	Description
pl_cust_slot_awburstX [1:0]	In	Burst type: 00: Fixed 01: Incremental 10: Wrapping 11: Reserved
pl_cust_slot_awvalidX	In	
cust_pl_slot_awreadyX	Out	
Write Data Channel		
pl_cust_slot_widX[15:0]	In	
pl_cust_slot_wdataX [127:0]	In	Write data
pl_cust_slot_wstrbX [15:0]	In	Byte enable
pl_cust_slot_wlastX	In	Not used by the port
pl_cust_slot_wvalidX	In	
cust_pl_slot_wreadyX	Out	
Write response channel		
cust_pl_slot_bidX[15:0]	Out	
cust_pl_slot_brespX[1:0]	Out	Tied to zero(OKAY).
cust_pl_slot_bvalidX	Out	
pl_cust_slot_breadyX	In	

Table 42: **AXI Read Ports**

Signal	Direction	Description
Read Address Channel		
pl_cust_slot_aridX[15:0]	In	
pl_cust_slot_araddrX [31:0]	In	Byte address
pl_cust_slot_arlenX[7:0]	In	Length of burst (number of transfer minus 1)
pl_cust_slot_arsizeX[2:0]	In	Transfer width: 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit
pl_cust_slot_arburstX [1:0]	In	Burst type: 00: Fixed 01: Incremental 10: Wrapping 11: Reserved
pl_cust_slot_arvalidX	In	

Table 42: **AXI Read Ports** *(cont'd)*

Signal	Direction	Description
cust_pl_slot_arreadyX	Out	
Read Data Channel		
cust_pl_slot_ridX[15:0]	Out	
cust_pl_slot_rdataX[127:0]	Out	Read data
cust_pl_slot_rrespX[1:0]	Out	Tied to zero (OKAY)
cust_pl_slot_rlastX	Out	
cust_pl_slot_rvalidX	Out	
pl_cust_slot_rreadyX	In	

Core Design Signals

The following table shows the signals to be used while designing the core and their descriptions:

Table 43: **Core Design Signals**

Signal	Description
phy_Clk	These are the clock rates that are set for each of the memory speed bins: For 2133 speed grade, the usrclk is 267 For 2400 speed grade, usrclk is 300 For 2667 speed grade, usrclk is 333 ***Note*:** phy_Clk was Usrclk in 2019.1 and earlier releases.
phy_sRst	This signal specifies that the DDR controller is out of reset (based on MMCM locked signal in VCU DDR4 controller) and it can be used to gate interconnect reset signal. ***Note*:** phy_sRst was sRst_out in 2019.1 and earlier releases.
InitDone	This signal specifies DDR4 calibration completion.

***Note*:** You should handle the domain crossing in the interconnect. You can use the `phy_clk` as master clock for the interconnect that interfaces with DDR4 memory ports.

Endianness

BA317 ports use normally little endian convention. The following table shows accesses to same data in memory from different kind of ports.

Table 44: **BA317 Ports**

Port	Endianness	Address	Data
64-bit buffered port	Little endian	0x0	0x0807060504030201

Table 44: **BA317 Ports** *(cont'd)*

Port	Endianness	Address	Data
32-bit buffered port	Little endian	0x0	0x04030201
		0x1	0x08070605
16-bit buffered port	Little endian	0x0	0x0201
		0x1	0x0403
		0x2	0x0605
		0x3	0x0807
8-bit buffered port	Little endian	0x0	0x01
		0x1	0x02
		0x2	0x03
		0x3	0x04
		0x4	0x05
		0x5	0x06
		0x6	0x07
		0x7	0x08
128-bit AXI port	Little endian	0x0	0x100F0E0D0C0B0A090807060504030201
32-bit AXI port	Little endian	0x0	0x04030201
		0x1	0x08070605

Physical Interface

The following table lists all available physical interfaces with the supported SDRAM and FPGA devices. Some additional FPGAs might be supported because they are compatible with the listed FPGAs.

Table 45: **Physical Interface**

Phy	SDRAM	FPGA	Rate
phyXilinxUltrascale	DDR4	Zynq-UltraScale+ EV	Quad

Vendor guidelines apply.

UltraScale/UltraScale+ Xilinx Phy (phyXilinxUltrascale)

Refer to the *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150) for details.

XILINX

Chapter 9

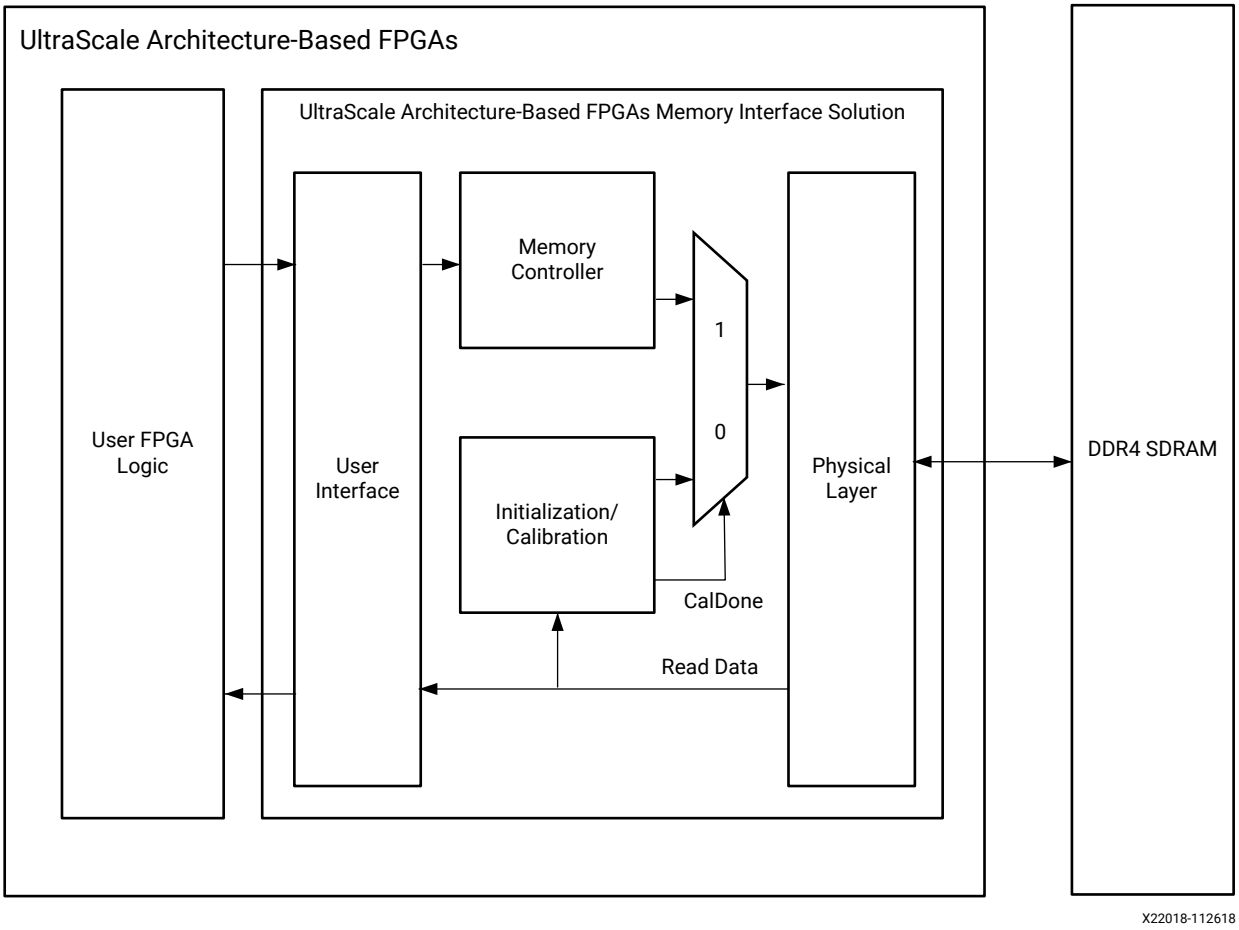
Core Architecture

This section describes the UltraScale architecture-based FPGAs Memory Interface Solutions core with an overview of the modules and interfaces.

Overview

The UltraScale architecture-based FPGAs Memory Interface Solutions is shown in the following figure.

Figure 19: **UltraScale Architecture-Based FPGAs Memory Interface Solution Core Architecture**



Memory Controller

The memory controller (MC) is designed to take Read, Write, and Read-Modify-Write transactions from the user interface (UI) block and issues them to memory efficiently with low latency, meeting all DRAM protocol and timing requirements while using minimal FPGA resources. The controller operates with a DRAM to system clock ratio of 4:1 and can issue one Activate, one CAS, and one Precharge command on each system clock cycle.

The controller supports an open page policy and can achieve very high efficiencies with workloads with a high degree of spatial locality. The controller also supports a closed page policy and the ability to reorder transactions to efficiently schedule workloads with address patterns that are more random. The controller also allows a degree of control over low-level functions with a UI control signal for AutoPrecharge on a per transaction basis as well as signals that can be used to determine when DRAM refresh commands are issued.

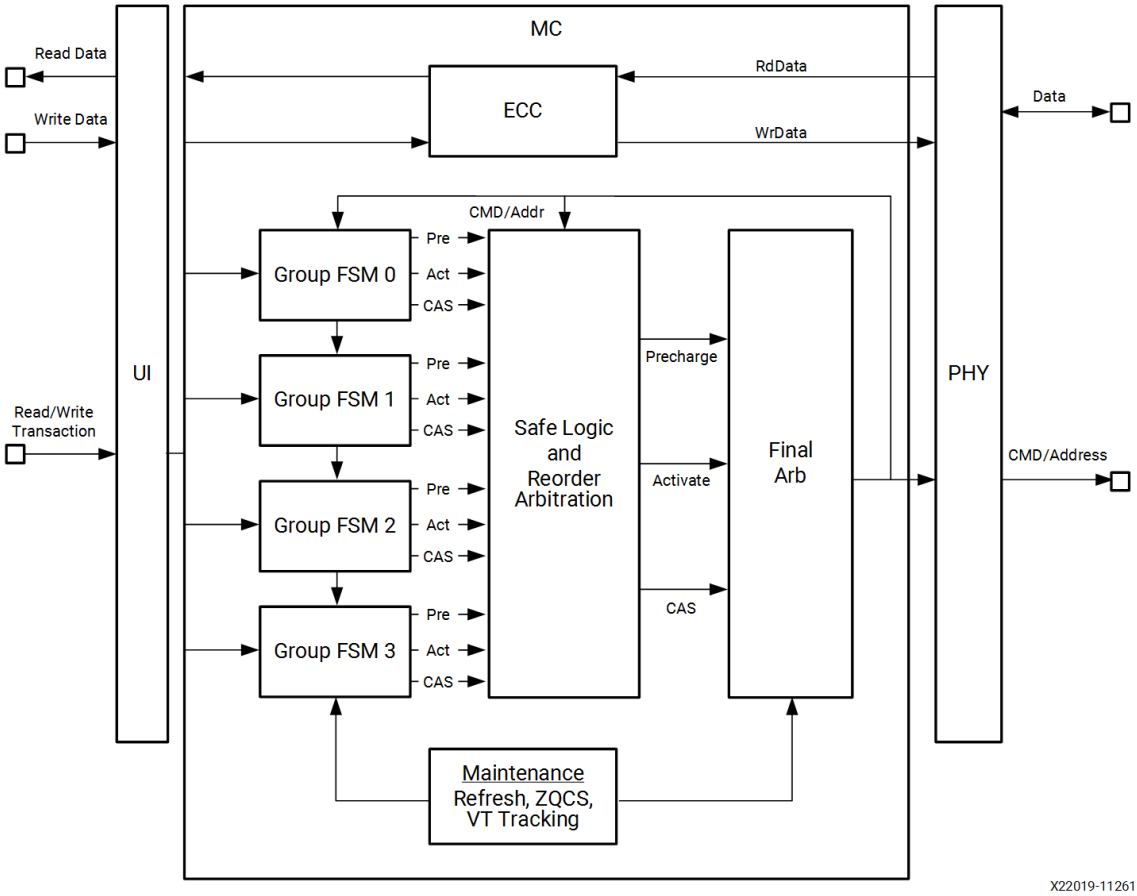
The key blocks of the controller command path include:

- The Group FSMs that queue up transactions, check DRAM timing, and decide when to request Precharge, Activate, and CAS DRAM commands.
- The "Safe" logic and arbitration units that reorder transactions between Group FSMs based on additional DRAM timing checks while also ensuring forward progress for all DRAM command requests.
- The Final Arbiter that makes the final decision about which commands are issued to the PHY and feeds the result back to the previous stages.

The maintenance blocks of the controller command path include:

- Blocks that generate refresh and ZQ Calibration Short commands
- Commands needed for voltage and temperature tracking

Figure 20: **Memory Controller Block Diagram**



Read and Write Coalescing

The controller prioritizes reads over writes when reordering is enabled. If both read and write CAS commands are safe to issue on the SDRAM command bus, the controller selects only read CAS commands for arbitration. When a read CAS issues, write CAS commands are blocked for several SDRAM clocks specified by parameter tRTW. This extra time is required for a write CAS to become safe after issuing a read CAS allows groups of reads to issue on the command bus without being interrupted by pending writes.

Reordering

Requests that map to the same mcGroup are never reordered. Reordering between the mcGroup instances is controlled with the ORDERING parameter. When set to "NORM," reordering is enabled and the arbiter implements a round-robin priority plan, selecting in priority order among the mcGroups with a command that is safe to issue to the SDRAM.

The timing of when it is safe to issue a command to the SDRAM can vary on the target bank or bank group and its page status. This often contributes to reordering.

When the ORDERING parameter is set to "STRICT," all requests have their CAS commands issued in the order in which the requests were accepted at the native interface. STRICT ordering overrides all other controller mechanisms, such as the tendency to coalesce read requests, and can therefore degrade data bandwidth utilization in some workloads.

When a new transaction is accepted from the UI, it is pushed into the stage 1 transaction FIFO. The page status of the transaction at the head of the stage 1 FIFO is checked and provided to the stage 1 transaction FSM. The FSM decides if a Precharge or Activate command needs to be issued, and when it is safe to issue them based on the DRAM timers.

When the page is open and not already scheduled to be closed due to a pending RDA or WRA in the stage 2 FIFO, the transaction is transferred from the stage 1 FIFO to the stage 2 FIFO. At this point, the stage 1 FIFO is popped and the stage 1 FSM begins processing the next transaction. In parallel, the stage 2 FSM processes the CAS command phase of the transaction at the head of the stage 2 FIFO. The stage 2 FSM issues a CAS command request when it is safe based on the tRCD timers. The stage 2 FSM also issues both a read and write CAS request for RMW transactions.

Read-Modify-Write Flow

When a `wr_bytes` command is accepted at the user interface it is eventually assigned to a group state machine like other write or read transactions. The group machine breaks the Partial Write into a read phase and a write phase. The read phase performs the following:

- First reads data from memory.
- Checks for errors in the read data.

- Corrects single bit errors.
- Stores the result inside the Memory Controller.

After read data is stored in the controller, the write phase begins as follows:

- Write data is merged with the stored read data based on the write data mask bits.
- Any multiple bit errors in the read phase results in the error being made undetectable in the write phase as new check bits are generated for the merged data.

When the write phase completes, the group machine becomes available to process a new transaction. The RMW flow ties up a group machine for a longer time than a simple read or write, and therefore might impact performance.

PHY

PHY is considered the low-level physical interface to an external DDR3 or DDR4 SDRAM device as well as all calibration logic for ensuring reliable operation of the physical interface itself. PHY generates the signal timing and sequencing required to interface to the memory device.

PHY contains the following features:

- Clock/address/control-generation logics
- Write and read datapaths
- Logic for initializing the SDRAM after power-up

In addition, PHY contains calibration logic to perform timing training of the read and write datapaths to account for system static and dynamic delays.

The PHY is included in the complete Memory Interface Solution core, but can also be implemented as a standalone PHY only block. A PHY only solution can be selected if you plan to implement a custom Memory Controller. For details about interfacing to the PHY only block. Refer to the *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150) for more information.

Designing with the Core

Clocking

PLDDR supports a wide range of clocks from the GUI drop down list.

The memory interface requires one MMCM, one TXPLL per I/O bank used by the memory interface, and two BUFGs. These clocking components are used to create the proper clock frequencies and phase shifts necessary for the proper operation of the memory interface.

There are two `TXPLL`s per bank. If a bank is shared by two memory interfaces, both `TXPLL`s in that bank are used.

***Note*:** DDR4 SDRAM generates the appropriate clocking structure and no modifications to the RTL are supported.

The DDR4 SDRAM tool generates the appropriate clocking structure for the desired interface. This structure must not be modified. The allowed clock configuration is as follows:

- Differential reference clock source connected to GCIO
- GCIO to MMCM (located in center bank of memory interface)
- MMCM to BUFG (located at center bank of memory interface) driving FPGA logic and all TXPLLs
- MMCM to BUFG (located at center bank of memory interface) divide by two mode driving 1/2 rate FPGA logic
- Clocking pair of the interface must be in the same SLR of memory interface for the SSI technology devices

Requirements

GCIO

- Must use a differential I/O standard
- Must be in the same I/O column as the memory interface

- Must be in the same SLR of memory interface for the SSI technology devices
- The I/O standard and termination scheme are system dependent. For more information, refer to the *UltraScale Architecture SelectIO Resources User Guide* (UG571).

MMCM

- MMCM is used to generate the FPGA logic system clock (1/4 of the memory clock)
- Must be located in the center bank of memory interface
- Must use internal feedback
- Input clock frequency divided by input divider must be 70 MHz (CLKINx / D = 70 MHz)

Table 46: **Memory Part Data Rate and Frequency**

Memory Part Data Rate (MT/s)	Phy_clk/User_clk
KVR21SE15S8/4 2133	267 MHz
KVR21SE15S8/4 2400	300 MHz
MT40A256M16GE-075E 2400	300 MHz
MT40A256M16GE-075E 2667	333 MHz
MT40A256M16H A-093 2133	267 MHz
MT40A512M8HX -093 2133	267 MHz
MTA4ATF51264H Z-2G6 2133	267 MHz
MT40A1G8SA-075:E 2400	300 MHz
MT40A1G8SA-075:E 2667	333 MHz
MT40A256M16LY-062E:F 2400	300 MHz

Input Clock Period Jitter

- Clock input period jitter must be 50 ps peak-to-peak

BUFGs and Clock Roots

- One BUFG is used to generate the system clock to FPGA logic and another BUFG is used to divide the system clock by two.
- BUFGs and clock roots must be located in center most bank of the memory interface.
  - For two bank systems, banks with more number of bytes selected is chosen as the center bank. If the same number of bytes is selected in two banks, then the top bank is chosen as the center bank.
  - For four bank systems, either of the center banks can be chosen. DDR4 SDRAM refers to the second bank from the top-most selected bank as the center bank.
  - Both the BUFGs must be in the same bank.

Resets

An asynchronous reset (`sys_rst`) input is provided. This is an active-Low reset and the `sys_rst` must assert for a minimum pulse width of 5 ns. The `sys_rst` can be an internal or external pin.

For more information on reset, see Reset Sequence.

Note: The best possible calibration results are achieved when the FPGA activity is minimized from the release of this reset input until the memory interface is fully calibrated as indicated by the `init_calib_complete` port (see the User Interface section of this document)

Connectivity with VCU DDR4 Controller IP

Figure 21: **Connecting to the VCU IP**

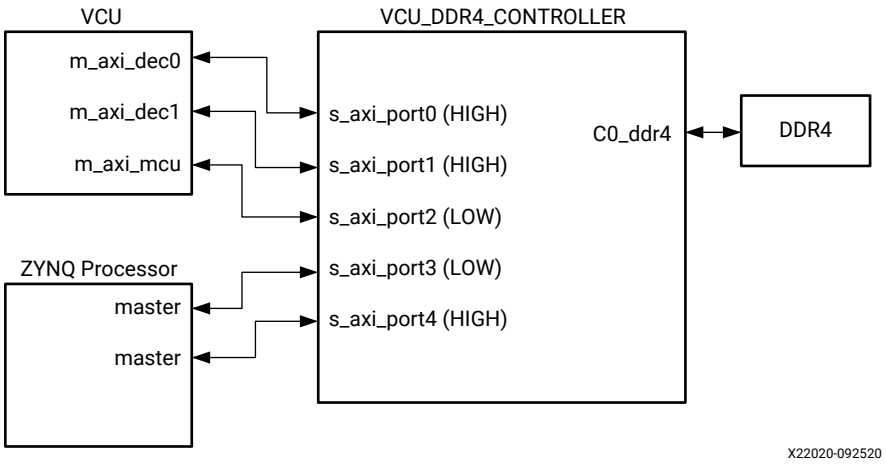


Table 47: **AXI Ports Map and Priority**

AXI Ports	Mapping	Port Priority
S_AXI_PORT_0	M_AXI_DEC_0 (VCU)	HIGH
S_AXI_PORT_1	M_AXI_DEC_1 (VCU)	HIGH
S_AXI_PORT_2	M_AXI_MCU (VCU)	LOW
S_AXI_PORT_3	Zynq processor	LOW
S_AXI_PORT_4	Zynq processor	HIGH

Port Priority Rationale

To achieve the best port performance and memory bandwidth usage, the following priority is provided and also hardcoded in the drop.

The ports consuming the highest bandwidth using the decoder ports and Zynq UltraScale+ MPSoC frame fetch are provided the highest priority. The other two ports that consume lower bandwidth are provided the lowest priority in the arbitration. This configuration is tested to provide the best performance.

Refer to the *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150) for information regarding PCB Guidelines for DDR4, Pin and Bank Rules, Protocol Description, Performance, DIMM Configurations, Setting Timing Options, and M and D Support for Reference Input Clock Speed.

Port connection recommendations for different use cases with VCU DDR Memory Controller

Capture – Encode – Stream @ 4Kp60

The current system can run Capture – Encode pipeline @ 4Kp60 with the available bandwidth of VCU DDR4 memory controller. Interconnect Encoder and Frame Buffer write using the following port recommendations for the VCU DDR Memory Controller.

Table 48: **Port Recommendations for 4Kp60 Streaming**

VCU DDR4 Controller port configuration	Slaves Connected through Interconnect
Port0	• Encoder_0 • Frame_buffer_write_0 • Frame_buffer_write_1
Port1	• Encoder_1 • Frame_buffer_write_2 • Frame_buffer_write_3
Port2	MCU
Port3	MPSOC – m_axi_hpm1_fpd
Port4	NC

Capture – Encode – Decode – Display @ 4Kp30

The current system can run Capture – Encode – Decode – Display pipeline with VCU DDR4 memory controller @ 4Kp30. Interconnect Encoder, Decoder and Frame Buffer write using the following port recommendations for the VCU DDR Memory Controller. This use case is tested and is working at 300MHz.

Table 49: **Port Recommendations for 4Kp30 Display**

VCU DDR4 Controller port configuration	Slaves Connected via Interconnect
Port0	• Encoder_0 • Decoder_0 • Frame_buffer_write_0 • Frame_buffer_write_1
Port1	• Encoder_1 • Decoder_0 • Frame_buffer_write_2 • Frame_buffer_write_3
Port2	MCU
Port3	MPSOC – m_axi_hpm1_fpd
Port4	HDMI – output (Video mixer and Frame buffer read)

Design Flow Steps

This section describes customizing and generating the core, constraining the core, and the simulation, synthesis, and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
- *Vivado Design Suite User Guide: Designing with IP* (UG896)
- *Vivado Design Suite User Guide: Getting Started* (UG910)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900)

Supported Configuration

Table 50: **Supported Configuration**

MEMORY PART	SUPPORTED DATA RATE MT/s	SUPPORTED DIMM Type	Ref CLK freq.	WIDTH OF DRAM	SUPPORTED RANKS	CLOCK CYCLES CL – tRCD - tRP
X16 (MT40A256M16GE-07)	DDR4 - 2400, 2667	COMPONENT	125 MHz	64-bit	1	2400 MT/s – 17-17-17 2667 MT/s – 19-19-19
X8 (KVR21SE15S8/4)	DDR4 - 2400, 2133	COMPONENT, SODIMM	300 MHz	64-bit	1	2400 MT/s – 17-17-17 2133 MT/s – 15-15-15
X16 (MT40A256M16HA-093)	DDR4 - 2133	COMPONENT	125 MHz	64-bit	1	2133 MT/s – 15-15-15
X8 (MT40A512M8HX-093)	DDR4 - 2133	COMPONENT	300 MHz	64-bit	1	2133 MT/s – 15-15-15
X16 (MTA4ATF51264HZ-2G6)	DDR4 - 2133	SODIMM	125 MHz	64-bit	1	2133 MT/s – 15-15-15
X8 (MT40A1G8SA-075:E)[1]	DDR4 - 2400, 2666	COMPONENT	100 MHz	64-bit	1	2400 MTs -17-17-17 2667 MT/s - 19-19-19
X16 (MT40A256M16LY-062E:F)	DDR4-2400	COMPONENT	125 MHz	64-bit	1	2400 MTs - 17-17-17

Notes:

1. Validation for this part performed on an alternative platform.

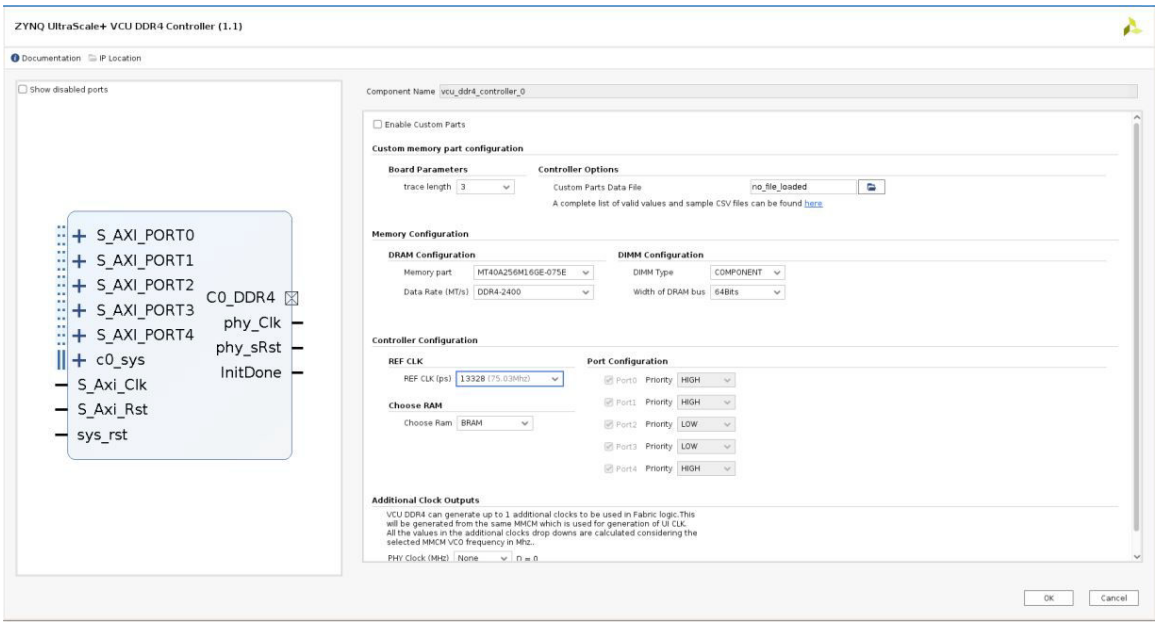
***Note*:** For the memory parts not listed in the previous table, compare the datasheet timing specifications (JEDEC) for compatibility. The key timing specifications are T~RCD~, T~RP~, T~AA~, T~FAW~, T~CCD~, Write CWL, and CL. If the datasheet specifications have similar or better timing, then the memory part should be compatible. Only the memory parts listed in the previous table have been validated by Xilinx.

Customizing the VCU DDR4 Controller

You can customize the GUI based on the following options:

- **Ref Clk (MHz):** This is a user defined clock that varies from 75MHz to 770MHz.
- **Choose RAM:** You can select from "BRAM" or "URAM + BRAM". Note that for 4EV and 5EV devices, the "URAM + BRAM" option is not valid. For 7EV devices, you should be aware that additional IPs in the design can consume URAMs and may result in a overutilization of URAMs in the 7EV device.
- **DRAM speed bin selection:** 2133 and 2400 for x8 selection. 2400 and 2667 for x16 selection.

Figure 22: **GUI Configuration**



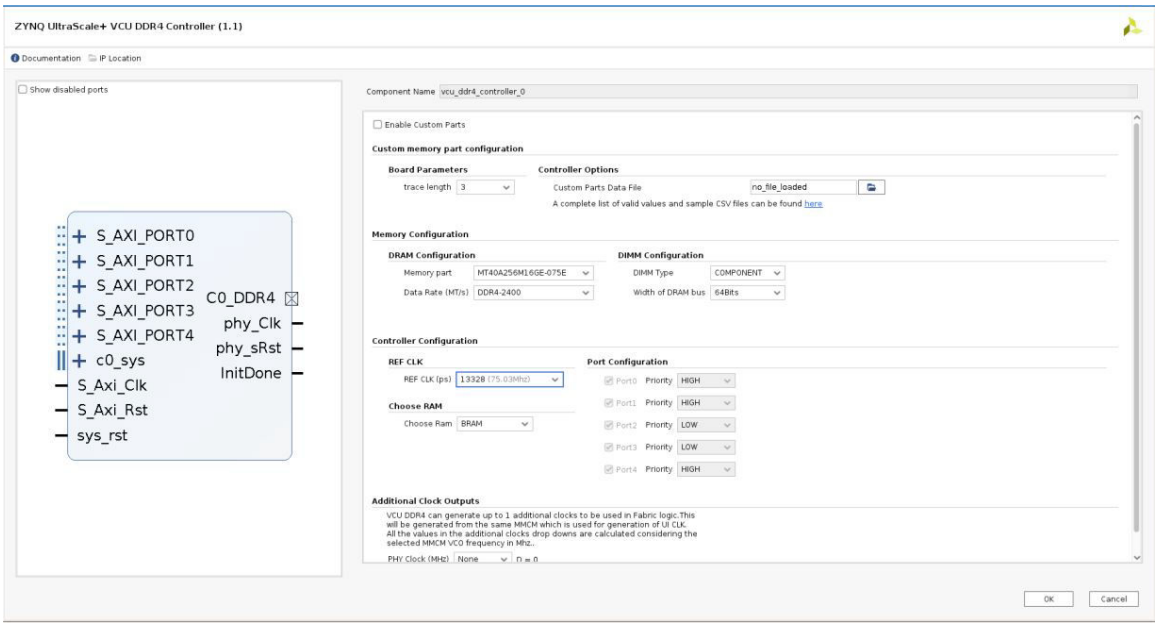
The port priority is fixed. Five ports are enabled in the IP. The high priority ports need to be connected to a decoder/display interface. The low priority ports need to be connected to a PS, Micro Controller Unit (MCU) interface.

Custom Flow

In the current VCU, DDR4 logic core IP supports only listed memories. and If additions are required, the user needs to get support from Xilinx team to add the new memories. To avoid this scenario, the VCU DDR4 controller is enhanced to support custom memory addition by getting the parameters and CSV format such as MIG input.

The following figure shows the GUI changes:

Figure 23: **Custom Flow GUI Configuration**



The preceding figure shows the GUI with updated DDR4 controller.

To add memory, perform the following steps:

1. Check the **Enable Custom Parts** option
2. Set the Reference clock (**Ref clk (MHZ)**). Currently, the Reference clock is user configurable from 75Mhz to 770Mhz.
3. Select the Data rate (**Data Rate (MT/s)**) from DDR4-2133, DDR4-2400 and DDR4 -2667.
4. Select the **DIMM Type** from SODIMM and COMPONENT
5. Select the populated CSV file in (**Custom Parts Data File**). (Details about the CSV file are explained in the following section.)
6. Select the Phy clock which is again user configurable. For better performance, select between 200 to 375 MHz.

Remaining parts are not required for the custom flow.

CSV File Detailss

Note that the CSV file needs to have only two rows - the first row contains the parameter names, and the second row contains only values.

The following are the valid values for all the parameters

- Rank is limited to 2.
- Stack Height is limited to 1.

- CA Mirror is limited to 0.
- Data mask is limited 1.
- Address width is limited to 17.
- Row width is limited to 14, 15, 16, 17, or 18.
- Column width is limited to 10.
- Bank width is limited to 2. (Listed as *Bank address in a bank group* for Micron parts).
- Bank Group width is limited to 1 for x16 devices and 2 for x4/x8 devices. (Listed as *Bank group address* in Micron parts).
- CS width is limited to 1 or 2 for components and DIMMs, and 2 or 4 for LRDIMMs.
- CKE width is limited to 1 and 2.
- ODT width is limited to 1 and 2.
- CK width is limited to 1 and 2.
- Memory Speed grade is limited to the value defined in the memory vendor data sheet.

  ***Note*:** This value is for information purposes only and is not used by the IP
- Memory density is limited to the value defined in the memory vendor data sheet.

  ***Note*:** This value is for information purposes only and is not used by the IP.
- Component density is limited to the value defined in the memory vendor data sheet.

  ***Note*:** This value is for information purposes only and is not used by the IP.
- Memory device width is limited to 4, 8, or 16 for components, and 64 or 72 for DIMMs.
- Memory component width is limited to 4, 8, and 16.
- Data bits per strobe is limited to 4 and 8.
- I/O Voltage is limited to 1.2V.
- Data width is limited to 8, 16, 24, 32, 40, 48, 56, 64, 72, 80.

  ***Note*:** Data width is limited to a maximum of 9 components.
- Min period has a range between 750ps and 1600ps.
- Max period is limited to 1600ps.
- **tCKE** is limited to 5000ps.
- **tFAW** is limited to 10000-35000ps.

  ***Note*:** This value often has a range that varies with frequency. It is critical that this is entered correctly based on the target memory interface rate.
- **tFAW_dllr** is limited to 16 tck.

Note: This parameter is only used for 3DS parts and should be set to **0** when using non-3DS parts.

- **tMRD** is limited to 8-10tck.
- **tRAS** is limited to 32000-35000ps.
- **tRCD** is limited to12500-15000ps.
- **tREFI** is limited to 3900000-7800000ps.
- **tRFC** is limited to 90000-550000ps.
- **tRFC_dlr** is limited to 90000-120000 ps.

  Note: This parameter is only used for 3DS parts and should be set to **0** when using non-3DS parts.
- **tRP** is limited to 12500-15000ps.
- **tRRD_S** is limited to 2500-6000ps.

  Note: This value often has a range that varies with frequency. It is critical that this is entered correctly based on the target memory interface rate.
- **tRRD_L** is limited to 4900-7500ps.

  Note: This value often has a range that varies with frequency. It is critical that this is entered correctly based on the target memory interface rate.
- **tRRD_dlr** is limited to 4 tck.

  Note: This parameter is only used for 3DS parts and should be set to **0** when using non-3DS parts.
- **tRTP** is limited to 7500ps.
- **tWR** is limited to 15000ps.
- **tWTR_S** is limited to 2500ps.
- **tWTR_L** is limited to 7500ps.
- **tXPR** is limited to 100-560ns.

  Note: This value often has a range that varies with frequency. It is critical that this is entered correctly based on the target memory interface rate.
- **tZQCS** is limited to 128tck.
- **tZQINIT** is limited to 1024tck.
- **tCCD_3ds** is limited to 4-5 tck.

  Note: This parameter is only used for 3DS parts and should be set to '0' when using non-3DS parts.
- CAS Latency is limited to 9 to 24 but specific values can be obtained from the memory vendor data sheet.

  Note: This value often has a range that varies with frequency. It is critical that this is entered correctly based on the target memory interface rate.

- CAS Write Latency has a range between 9 to 20 but specific values can be obtained from the memory vendor data sheet.

  ***Note*:** This value often has a range that varies with frequency. It is critical that this is entered correctly based on the target memory interface rate.

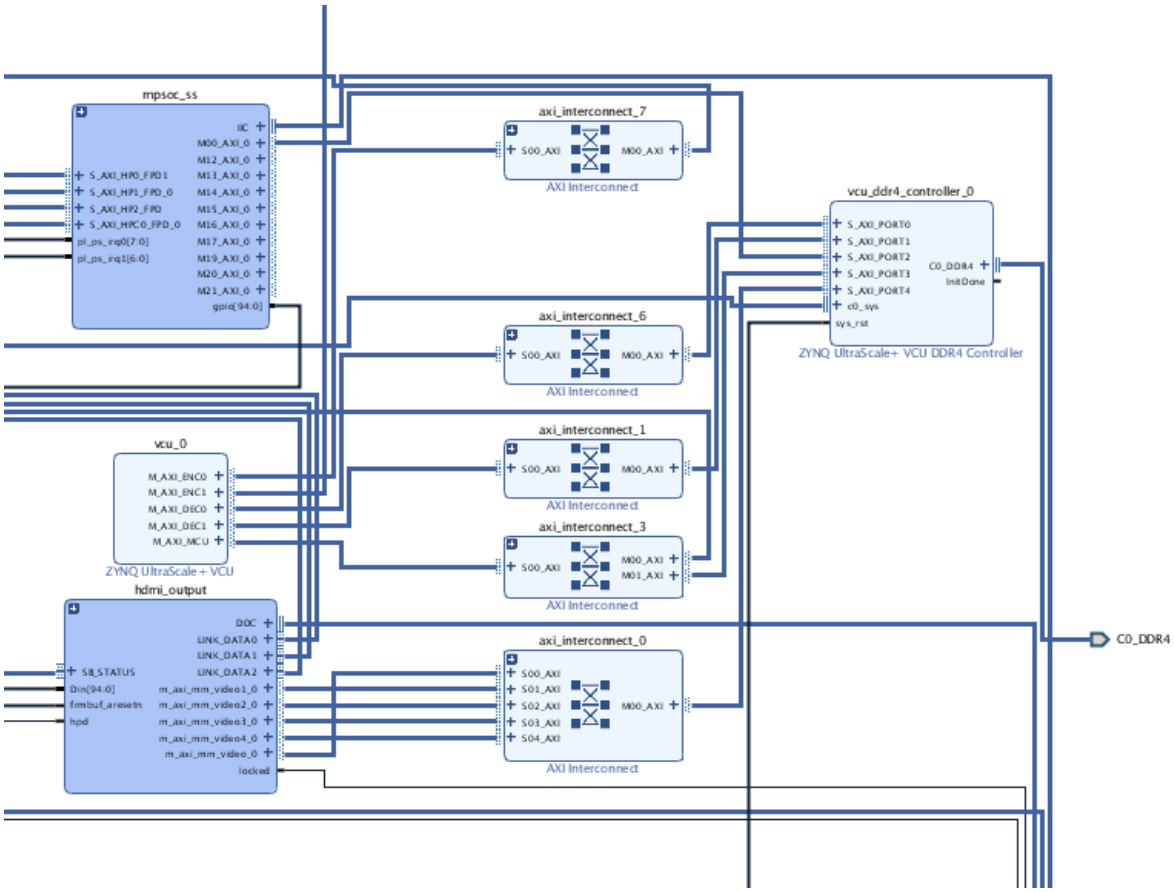
- Burst Length is limited to 8.
- RTT (nominal) - ODT is limited to RZQ/6.

  ***Note*:** This parameter has been removed starting in the 2016.4 example CSV as the IP automatically defines this value based on slot and rank selection.

Connecting with VCU LogiCORE IP

The following figure shows how the IP is connected to VCU LogiCORE IP.

Figure 24: **IP Connection**



The following table shows the recommended port connection.

Table 51: **Port Connection**

VCU DDR4 Controller Port	Master Port	Priority
S_AXI_PORT0	M_AXI_DEC0 (through AXI interconnect)	High
S_AXI_PORT1	M_AXI_DEC1 (through AXI interconnect)	High
S_AXI_PORT2	M_AXI_MCU (through AXI interconnect)	Low
S_AXI_PORT3	M_AXI_HPM0/1 (through AXI interconnect)	Low
S_AXI_PORT4	M00_AXI (Video mixer or frame buffer read master)	High

I/O Planning

DDR4 SDRAM I/O pin planning is completed with the full design pin planning using the Vivado I/O pin planner. DDR4 SDRAM I/O pins can be selected through several Vivado I/O pin planner features including assignments using I/O Ports view, Package view, or Memory Bank/Byte Planner. Pin assignments can additionally be made through importing an XDC or modifying the existing XDC file.

These options are available for all DDR4 SDRAM designs and multiple DDR4 SDRAM IP instances can be completed in one setting. To learn more about the available Memory IP pin planning options, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899).

Constraining the Core

Required Constraints

For DDR3/DDR4 SDRAM Vivado IDE, you specify the pin location constraints. For more information on I/O standard and other constraints, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899). The location is chosen by the Vivado IDE according to the banks and byte lanes chosen for the design.

The I/O standard is chosen by the memory type selection and options in the Vivado IDE and by the pin type. A sample for `dq[0]` is shown here.

```
set_property PACKAGE_PIN AF20 [get_ports "c0_ddr4_dq[0]"]
set_property IOSTANDARD POD12_DCI [get_ports "c0_ddr4_dq[0]"]
```

Internal VREF is always used for DDR4. Internal VREF is optional for DDR3. A sample for DDR4 is shown here.

```
set_property INTERNAL_VREF 0.600 [get_iobanks 45]
```

***Note*:** Internal VREF is automatically generated by the tool and you do not need to specify it. The VREF value listed in this constraint is not used with PODL12 I/Os. The initial value is set to 0.84V. The calibration logic adjusts this voltage as needed for maximum interface performance.

The system clock must have the period set properly:

```
create_clock -name c0_sys_clk -period.938 [get_ports c0_sys_clk_p]
```

For HR banks, update the `output_impedance` of all the ports assigned to HR banks pins using the `reset_property` command. For more information, see AR: 63852.

Maximum Delay Constraints

The following code example shows the maximum delay constraints for the ZCU106 (125 MHz)

```
set_max_delay -datapath_only -from [get_clocks mmcm_clkout0] -to
[get_clocks mmcm_clkout2] 2.900
set_max_delay -datapath_only -from [get_clocks mmcm_clkout2] -to
[get_clocks mmcm_clkout0] 3.900
```

The following code example shows the maximum delay constraints for the ZCU104 (300 MHz)

```
set_max_delay -datapath_only -from [get_clocks mmcm_clkout0] -to
[get_clocks mmcm_clkout2] 3.231
set_max_delay -datapath_only -from [get_clocks mmcm_clkout2] -to
[get_clocks mmcm_clkout0] 3.897
```

***Note*:** Clock constraints for IPs are managed in Vivado and you need not enter constraint values.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

For more information on clocking, see Clocking.

Clock Frequencies

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

The DDR3/DDR4 SDRAM tool generates the appropriate I/O standards and placement based on the selections made in the Vivado IDE for the interface type and options.

Simulation

Simulation of the Zynq UltraScale™ EV Architecture Video Codec Unit DDR4 LogiCORE IP is not supported.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

Data Rates Supported by VCU DDR4 Controller

Table 52: **VCU DDR4 Controller Data Rates**

UltraScale+ Part	Data Rate		
	2133	2400	2666
xczu7ev-fbvb900-1-e	YES	YES	NO
xczu7ev-fbvb900-1-i	YES	YES	NO
xczu7ev-fbvb900-1L-i	NO	NO	NO
xczu7ev-fbvb900-1LV-i	NO	NO	NO
xczu7ev-fbvb900-2-e	YES	YES	YES
xczu7ev-fbvb900-2-i	YES	YES	YES
xczu7ev-fbvb900-2L-e	YES	YES	YES
xczu7ev-fbvb900-2LV-e	YES	YES	NO
xczu7ev-fbvb900-3-e	YES	YES	YES
xczu7ev-ffvc1156-1-e	YES	YES	NO
xczu7ev-ffvc1156-1-i	YES	YES	NO
xczu7ev-ffvc1156-1L-i	NO	NO	NO
xczu7ev-ffvc1156-1LV-i	NO	NO	NO
xczu7ev-ffvc1156-2-e	YES	YES	YES
xczu7ev-ffvc1156-2-i	YES	YES	YES
xczu7ev-ffvc1156-2L-e	YES	YES	YES
xczu7ev-ffvc1156-2LV-e	YES	YES	NO
xczu7ev-ffvc1156-3-e	YES	YES	YES
xczu7ev-ffvf1517-1-e	YES	YES	NO

Table 52: **VCU DDR4 Controller Data Rates** *(cont'd)*

UltraScale+ Part	Data Rate		
xczu7ev-ffvf1517-1-i	YES	YES	NO
xczu7ev-ffvf1517-1L-i	NO	NO	NO
xczu7ev-ffvf1517-1LV-i	NO	NO	NO
xczu7ev-ffvf1517-2-e	YES	YES	YES
xczu7ev-ffvf1517-2-i	YES	YES	YES
xczu7ev-ffvf1517-2L-e	YES	YES	YES
xczu7ev-ffvf1517-2LV-e	YES	YES	NO
xczu7ev-ffvf1517-3-e	YES	YES	YES
xczu4ev-fbvb900-1-e	YES	YES	NO
xczu4ev-fbvb900-1-i	YES	YES	NO
xczu4ev-fbvb900-1L-i	NO	NO	NO
xczu4ev-fbvb900-1LV-i	NO	NO	NO
xczu4ev-fbvb900-2-e	YES	YES	YES
xczu4ev-fbvb900-2-i	YES	YES	YES
xczu4ev-fbvb900-2L-e	YES	YES	YES
xczu4ev-fbvb900-2LV-e	YES	YES	NO
xczu4ev-fbvb900-3-e	YES	YES	YES
xczu4ev-sfvc784-1-e	NO	NO	NO
xczu4ev-sfvc784-1-i	NO	NO	NO
xczu4ev-sfvc784-1L-i	NO	NO	NO
xczu4ev-sfvc784-1LV-i	NO	NO	NO
xczu4ev-sfvc784-2-e	YES	YES	NO
xczu4ev-sfvc784-2-i	YES	YES	NO
xczu4ev-sfvc784-2L-e	YES	YES	NO
xczu4ev-sfvc784-2LV-e	YES	YES	NO
xczu4ev-sfvc784-3-e	YES	YES	NO
xazu4ev-sfvc784-1-i	NO	NO	NO
xazu4ev-sfvc784-1LV-i	NO	NO	NO
xazu4ev-sfvc784-1Q-q	NO	NO	NO
xazu5ev-sfvc784-1-i	NO	NO	NO
xazu5ev-sfvc784-1LV-i	NO	NO	NO
xazu5ev-sfvc784-1Q-q	NO	NO	NO
xazu7ev-fbvb900-1-i	NO	NO	NO
xazu7ev-fbvb900-1Q-q	NO	NO	NO
xczu5ev-fbvb900-1-e	YES	YES	NO
xczu5ev-fbvb900-1-i	YES	YES	NO
xczu5ev-fbvb900-1L-i	NO	NO	NO
xczu5ev-fbvb900-1LV-i	NO	NO	NO
xczu5ev-fbvb900-2-e	YES	YES	YES

Table 52: **VCU DDR4 Controller Data Rates** *(cont'd)*

UltraScale+ Part	Data Rate		
xczu5ev-fbvb900-2-i	YES	YES	YES
xczu5ev-fbvb900-2L-e	YES	YES	YES
xczu5ev-fbvb900-2LV-e	YES	YES	NO
xczu5ev-fbvb900-3-e	YES	YES	YES
xczu5ev-sfvc784-1-e	NO	NO	NO
xczu5ev-sfvc784-1-i	NO	NO	NO
xczu5ev-sfvc784-1L-i	NO	NO	NO
xczu5ev-sfvc784-1LV-i	NO	NO	NO
xczu5ev-sfvc784-2-e	YES	YES	NO
xczu5ev-sfvc784-2-i	YES	YES	NO
xczu5ev-sfvc784-2L-e	YES	YES	NO
xczu5ev-sfvc784-2LV-e	YES	YES	NO
xczu5ev-sfvc784-3-e	YES	YES	NO

XILINX

Section III

VCU Sync IP v1.0

XILINX

Chapter 12

Introduction

The VCU Sync IP core is a video buffer fence IP designed to be used with the Zynq® UltraScale+™ MPSoCs EV series designs. This section provides information about using, customizing, and simulating the VCU Sync IP core for Zynq UltraScale+ MPSoCs. It also describes the architecture and provides details on customizing and interfacing to the VCU.

IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq® UltraScale+™ MPSoC EV Devices
Supported User Interfaces	AXI4
Resources	See Resource Utilization
Provided with Core	
Design Files	RTL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Xilinx Constraints File (XDC)
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite
Simulation[3]	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado® Design Suite
Support	
Release Notes and Known Issues	Master Answer Record: 76600
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado® IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
3. Behavioral simulations using only Verilog simulation models are supported. Netlist (post-synthesis and post-implementation) simulations are not supported.

Overview

The VCU Sync IP core is designed to act as a fence IP between the Video DMA and VCU IPs. It is used in video applications that require ultra-low latencies. The Sync IP performs AXI-transaction-level tracking so that the producer and consumer can be synchronized at the granularity of AXI transactions instead of the granularity of the video buffer level. The Sync IP element is responsible for synchronizing buffers between the capture DMA and the VCU encoder. The Sync IP can only synchronize between capture DMA and VCU encoder.

While the capture element is writing into the DRAM, the capture hardware writes video buffers in raster scan order and the Sync IP monitors the buffer level. It allows the encoder to read input buffer data, if the requested data is already written by the DMA; otherwise, it blocks the encoder until the DMA completes its writes. On the decoder side, the VCU decoder writes the decoded video buffer into the DRAM in block raster scan order and the display reads data in raster scan order.

Figure 25: **Use of Sync IP in UltraScale+ EV Series Devices Accessing PSDDR**

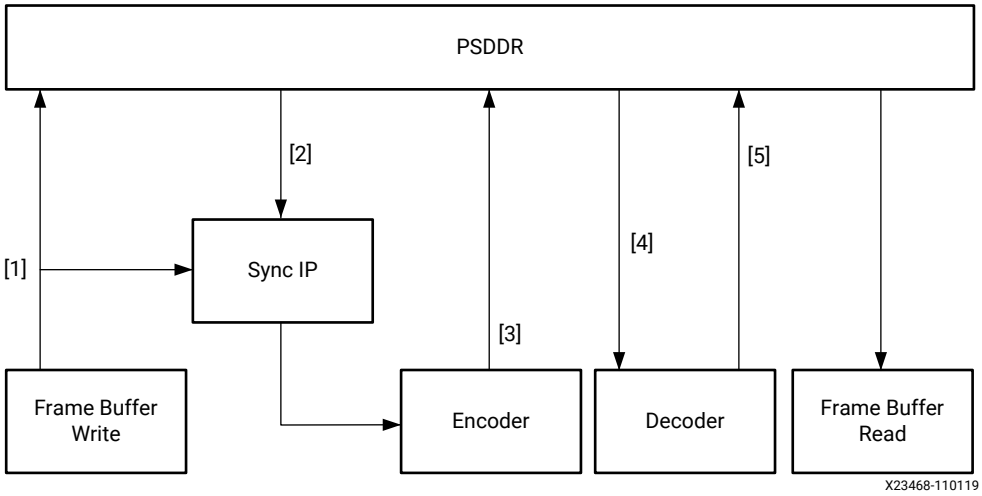
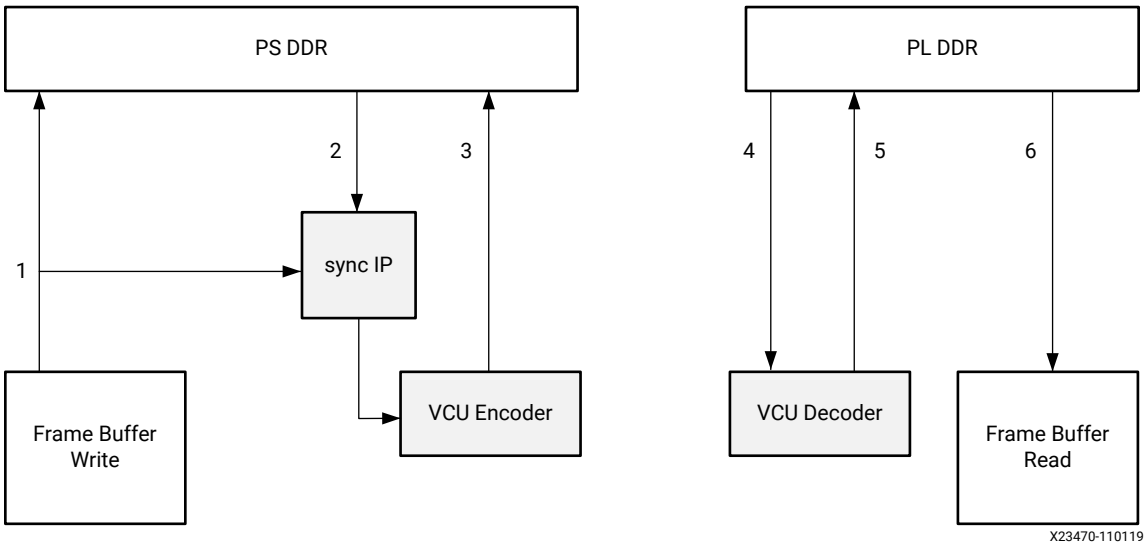


Figure 26: **Use of Sync IP in UltraScale+ EV Series Devices Accessing PLDDR**



1: The frame buffer writes into the memory. In encoder mode, the Sync IP snoops the transactions.

2: The VCU encoder reads the transactions. The Sync IP only allows reads when the frame buffer has completed the writes to those sections of the memory.

3: The VCU encoder writes the compressed video stream back to DRAM.

4: The VCU decoder reads the compressed video stream from DRAM.

5: The VCU decoder writes back decoded frames into DRAM.

6: The display reads decoded frames after half the frame is written by the VCU decoder.

Features

- The 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.
- The Sync IP core supports the tracking and control of four simultaneous channels.

Licensing and Ordering

This Xilinx LogiCORE IP module is provided at no additional cost for Zynq UltraScale+ MPSoC EV Devices with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License. Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

N/A

Performance

N/A

Resource Utilization

For details about performance and resource use, see table Table 53: Sync IP Multistream Utilization.

Table 53: **Sync IP Multistream Utilization**

Name	PLDDR	PSDDR
CLB LUTS (230400)	21587	21466
CLB Registers (460800)	30554	30356
CARRY8 (28800)	1620	1620
F7 MUXES (11520)	215	165
F8 MUXES (57600)	4	0
BLOCK RAM TILE (312)	8	8
DSPs (1728)	0	0
HPIOBDIFFINBUF (192)	0	0
BITSLICE_RX_TX (416)	0	0
GLOBAL CLOCK BUFFERS (544)	0	0
PLL (16)	0	-
MMCM (8)	0	-

***Note*:** The Sync IP logic core GUI has an option of selecting the DDR4 memory size depending on the requirement. Table Table 53: Sync IP Multistream Utilization gives the full address width utilization.

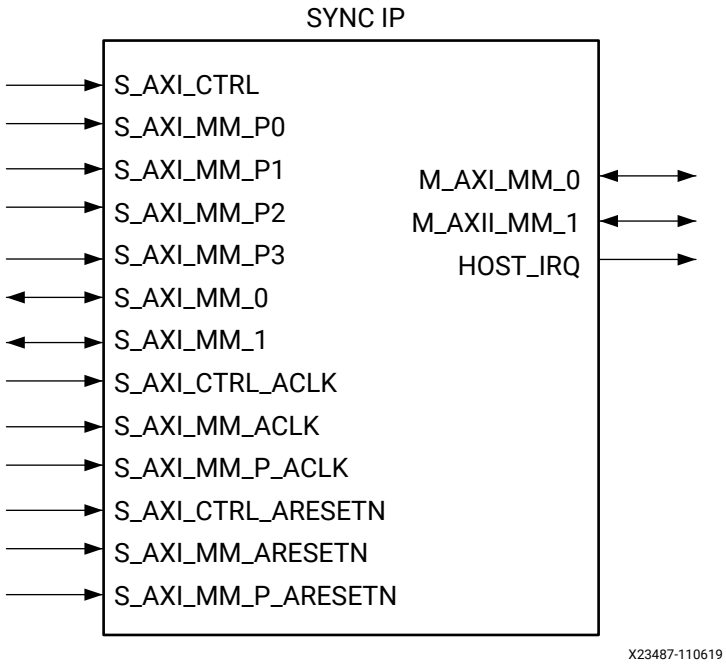
Port Descriptions

AXI Clock Port/Reset Port Connection

The following figure shows the block diagram of the VCU Sync IP core that has seven input AXI ports and two output AXI ports.

- **S_AXI_CTRL_ACLK/S_AXI_MM_ALCLK/S_AXI_MM_P_ACLK:** The AXI ports work with respect to this clock and reset. The clock frequency used for the Sync IP is 300 MHz.
- **S_AXI_CTRL_ARESETN/S_AXI_CTRL_MM_ARESETN/S_AXI_CTRL_MM_P_ARESETN:** An active-Low reset is used in the Sync IP.

Figure 27: **Sync IP**



Controller AXI Ports

The AXI specification is available at http://www.arm.com/products/system-ip/amba/amba-open-specifications.php.

Table 54: **Controller AXI Write PortX Interface**

Signal	Direction	Description
Write Address Channel		
s_axi_ctrl_awaddrX [11:0]	Input	Byte address
s_axi_ctrl_awvalidX	Input	
s_axi_ctrl_awreadyX	Output	
s_axi_ctrl_awprotX[2:0]	Input	
Write Data Channel		
s_axi_ctrl_wdataX [31:0]	Input	
s_axi_ctrl_wstrbX [3:0]	Input	Write data
s_axi_ctrl_wvalidX	Input	Byte enable
s_axi_ctrl_wreadyX	Output	
Write response channel		
s_axi_ctrl_brespX[1:0]	Output	Tied to zero (OKAY)
s_axi_ctrl_bvalidX	Output	
s_axi_ctrl_breadyX	Input	

Table 55: **Controller AXI Read PortX Interface**

Signal	Direction	Description
Read Address Channel		
s_axi_ctrl_arprotX[2:0]	Input	
s_axi_ctrl_araddrX [11:0]	Input	Byte address
s_axi_ctrl_arvalidX	Input	
s_axi_ctrl_arreadyX	Output	
Read Data Channel		
s_axi__ctrl_rdataX[31:0]	Output	Read data
s_axi__ctrl_rrespX[1:0]	Output	Tied to zero (OKAY)
s_axi__ctrl_rvalidX	Output	
s_axi__ctrl_rreadyX	Input	

Consumer AXI Ports

Table 56: **Consumer AXI Write PortX Interface**

Signal	Direction	Description
Write Address Channel		
s_axi_mm_x_awid[3:0]	Input	
s_axi_mm_x_awaddr [63:0]	Input	Byte address
s_axi_mm_x_awlen [7:0]	Input	Length of burst (number of transfer minus 1)

Table 56: **Consumer AXI Write PortX Interface** *(cont'd)*

Signal	Direction	Description
s_axi_mm_x_awsize [2:0]	Input	Transfer width: 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit
s_axi_mm_x_awburst [1:0]	Input	Burst type 00: Fixed 01: Incremental 10: Wrapping 11: Reserved :
s_axi_mm_x_awvalid	Input	
s_axi_mm_x_awready	Output	
s_axi_mm_x_awcache[3:0]	Input	
s_axi_mm_x_awlock	Input	
s_axi_mm_x_awprot[2:0]	Input	
s_axi_mm_x_awqos[3:0]	Input	
s_axi_mm_x_awregion[3:0]	Input	
s_axi_mm_x_awuser	Input	
Write Data Channel		
s_axi_mm_x_wdata [127:0]	Input	Write data
s_axi_mm_x_wstrb [15:0]	Input	Byte enable
s_axi_mm_x_wlast	Input	Not used by the port
s_axi_mm_x_wvalid	Input	
s_axi_mm_x_wready	Output	
s_axi_mm_x_wuser	Input	
Write response channel		
s_axi_mm_x_bid[3:0]	Output	
s_axi_mm_x_bresp[1:0]	Output	Tied to zero (OKAY)
s_axi_mm_x_bvalid	Output	
s_axi_mm_x_bready	Input	
s_axi_mm_x_buser	Output	

Table 57: **Consumer AXI Read PortX Interface**

Signal	Direction	Description
Read Address Channel		
s_axi_mm_x_arid[3:0]	Input	
s_axi_mm_x_araddr [63:0]	Input	Byte address

Table 57: **Consumer AXI Read PortX Interface** *(cont'd)*

Signal	Direction	Description
s_axi_mm_x_arlen[7:0]	Input	Length of burst (number of transfers minus 1)
s_axi_mm_x_arsize[2:0]	Input	Transfer width: 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit
s_axi_mm_x_arburst[1:0]	Input	Burst type: 00: Fixed 01: Incremental 10: Wrapping 11: Reserved
s_axi_mm_x_arvalid	Input	
s_axi_mm_x_arready	Output	
s_axi_mm_x_arcache	Input	
s_axi_mm_x_arlock	Input	
s_axi_mm_x_arprot[2:0]	Input	
s_axi_mm_x_arqos[3:0]	Input	
s_axi_mm_x_arregion[3:0]	Input	
s_axi_mm_x_aruser	Input	
Read Data Channel		
s_axi_mm_x_rid[3:0]	Output	
s_axi_mm_x_rdata[127:0]	Output	Read Data
s_axi_mm_x_rresp[1:0]	Output	Tied to zero (OKAY)
s_axi_mm_x_rlast	Output	
s_axi_mm_x_rvalid	Output	
s_axi_mm_x_rready	Input	
s_axi_mm_x_ruser	Output	

Producer AXI Ports

Table 58: **Producer AXI Write PortX Interface**

Signal	Direction	Description
Write Address Channel		
s_axi_mm_p_x_awid[3:0]	Input	
s_axi_mm_p_x_awaddr [63:0]	Input	Byte address
s_axi_mm_p_x_awlen [7:0]	Input	Length of burst (number of transfer minus 1)

Table 58: **Producer AXI Write PortX Interface** *(cont'd)*

Signal	Direction	Description
s_axi_mm_p_x_awsize [2:0]	Input	Transfer width: 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit
s_axi_mm_p_x_awburst [1:0]	Input	Burst type: 00: Fixed 01: Incremental 10: Wrapping 11: Reserved
s_axi_mm_p_x_awvalid	Input	
s_axi_mm_p_x_awready	Input	
s_axi_mm_p_x_awcache[3:0]	Input	
s_axi_mm_p_x_awlock	Input	
s_axi_mm_p_x_awprot[2:0]	Input	
s_axi_mm_p_x_awqos[3:0]	Input	
s_axi_mm_p_x_awregion[3:0]	Input	
S_axi_mm_p_x_user	Input	
Write Data Channel		
s_axi_mm_p_x_wdata [127:0]	Input	Read data
s_axi_mm_p_x_wstrb [15:0]	Input	Byte enable
s_axi_mm_p_x_wlast	Input	Not used by the port
s_axi_mm_p_x_wvalid	Input	
s_axi_mm_p_x_wready	Input	
s_axi_mm_p_x_wuser	Input	
Write Response Channel		
s_axi_mm_p_x_bid[3:0]	Input	
s_axi_mm_p_x_bresp[1:0]	Input	Tied to zero (OKAY)
s_axi_mm_p_x_bvalid	Input	
s_axi_mm_p_x_bready	Input	
s_axi_mm_p_x_buser	Input	

Table 59: **Producer AXI Read PortX Interface**

Signal	Direction	Description
Read Address Channel		
s_axi_mm_p_x_arid[3:0]	Input	
s_axi_mm_p_x_araddr [63:0]	Input	Byte address
s_axi_mm_p_x_arlen[7:0]	Input	Length of burst (number of transfer minus 1)

Table 59: **Producer AXI Read PortX Interface** *(cont'd)*

Signal	Direction	Description
s_axi_mm_p_x_arsize[2:0]	Input	Transfer width: 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit
s_axi_mm_p_x_arburst[1:0]	Input	Burst type: 00: Fixed 01: Incremental 10: Wrapping 11: Reserved
s_axi_mm_p_x_arvalid	Input	
s_axi_mm_p_x_arready	Input	
s_axi_mm_p_x_arcache	Input	
s_axi_mm_p_x_arlock	Input	
s_axi_mm_p_x_arprot[2:0]	Input	
s_axi_mm_p_x_arqos[3:0]	Input	
s_axi_mm_p_x_arregion[3:0]	Input	
s_axi_mm_p_x_aruser	Input	
Read Data Channel		
s_axi_mm_p_x_rid[3:0]	Input	
s_axi_mm_p_x_rdata[127:0]	Input	Read data
s_axi_mm_p_x_rresp[1:0]	Input	Tied to zero (OKAY)
s_axi_mm_p_x_rlast	Input	
s_axi_mm_p_x_rvalid	Input	
s_axi_mm_p_x_rready	Input	
s_axi_mm_p_x_ruser	Input	

Consumer Master AXI Ports

Table 60: **Consumer Master AXI Write PortX Interface**

Signal	Direction	Description
Write Address Channel		
m_axi_mm_x_awid[3:0]	Output	
m_axi_mm_x_awaddr [63:0]	Output	Byte address
m_axi_mm_x_awlen [7:0]	Output	Length of burst (number of transfer minus 1)

Table 60: **Consumer Master AXI Write PortX Interface** *(cont'd)*

Signal	Direction	Description
m_axi_mm_x_awsize [2:0]	Output	Transfer width: 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit
m_axi_mm_x_awburst [1:0]	Output	Burst type: 00: Fixed 01: Incremental 10: Wrapping 11: Reserved
m_axi_mm_x_awvalid	Output	
m_axi_mm_x_awready	Input	
m_axi_mm_x_awcache[3:0]	Output	
m_axi_mm_x_awlock	Output	
m_axi_mm_x_awprot[2:0]	Output	
m_axi_mm_x_awqos[3:0]	Output	
m_axi_mm_x_awregion[3:0]	Output	
m_axi_mm_x_awuser	Output	
Write Data Channel		
m_axi_mm_x_wdata [127:0]	Output	Read data
m_axi_mm_x_wstrb [15:0]	Output	Byte enable
m_axi_mm_x_wlast	Output	Not used by the port
m_axi_mm_x_wvalid	Output	
m_axi_mm_x_wready	Output	
m_axi_mm_x_wuser	Output	
Write Response Channel		
s_axi_mm_p_x_bid[3:0]	Input	
s_axi_mm_p_x_bresp[1:0]	Input	Tied to zero (OKAY)
s_axi_mm_p_x_bvalid	Input	
s_axi_mm_p_x_bready	Output	
s_axi_mm_p_x_buser	Input	

Table 61: **Consumer Master AXI Read PortX Interface**

Signal	Direction	Description
Read Address Channel		
m_axi_mm_x_arid[3:0]	Output	
m_axi_mm_x_araddr [63:0]	Output	Byte address
m_axi_mm_x_arlen[7:0]	Output	Length of burst (number of transfers minus 1)

Table 61: **Consumer Master AXI Read PortX Interface** *(cont'd)*

Signal	Direction	Description
m_axi_mm_x_arsize[2:0]	Output	Transfer width: 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit
m_axi_mm_x_arburst[1:0]	Output	Burst type: 00: Fixed 01: Incremental 10: Wrapping 11: Reserved
m_axi_mm_x_arvalid	Output	
m_axi_mm_x_arready	Input	
m_axi_mm_x_arcache	Output	
m_axi_mm_x_arlock	Output	
m_axi_mm_x_arprot[2:0]	Output	
m_axi_mm_x_arqos[3:0]	Output	
m_axi_mm_x_arregion[3:0]	Output	
m_axi_mm_x_aruser	Output	
Read Data Channel		
m_axi_mm_x_rid[3:0]	Input	
m_axi_mm_x_rdata[127:0]	Input	Read data
m_axi_mm_x_rresp[1:0]	Input	Tied to zero (OKAY)
m_axi_mm_x_rlast	Input	
m_axi_mm_x_rvalid	Input	
m_axi_mm_x_rready	Output	
m_axi_mm_x_ruser	Input	

Mapping for AXI Ports

Table 62: **Encoder Mapping of AXI Ports**

Sync IP Ports	Connected to Ports
S_AXI_CTRL	Register Configuration – Connected to MPSoC/Arm
S_AXI_MM_P0	Producer 0 (Video Capture VDMA) Snoop Bus
S_AXI_MM_P1	Producer 1 (Video Capture VDMA) Snoop Bus
S_AXI_MM_P2	Producer 2 (Video Capture VDMA) Snoop Bus
S_AXI_MM_P3	Producer 3 (Video Capture VDMA) Snoop Bus
S_AXI_MM_0	Consumer Input 0 (VCU Encoder0)

Table 62: **Encoder Mapping of AXI Ports** *(cont'd)*

Sync IP Ports	Connected to Ports
S_AXI_MM_1	Consumer Input 1 (VCU Encoder1)
M_AXI_MM_0	Consumer Output 0 to Memory
M_AXI_MM_1	Consumer Output 1 to Memory

Table 63: **Decoder Mapping of AXI Ports**

Sync IP Ports	Connected to Ports
S_AXI_CTRL	Register Configuration; Connected to MPSoC/Arm
S_AXI_MM_P0	Producer 0 (VCU Decoder 0) Snoop Bus
S_AXI_MM_P1	Producer 1 (VCU Decoder 1) Snoop Bus
S_AXI_MM_0	Consumer 0 (Video Sink VDMA)
S_AXI_MM_1	Consumer 1 (Video Sink VDMA)
M_AXI_MM_0	To Memory
M_AXI_MM_1	To Memory

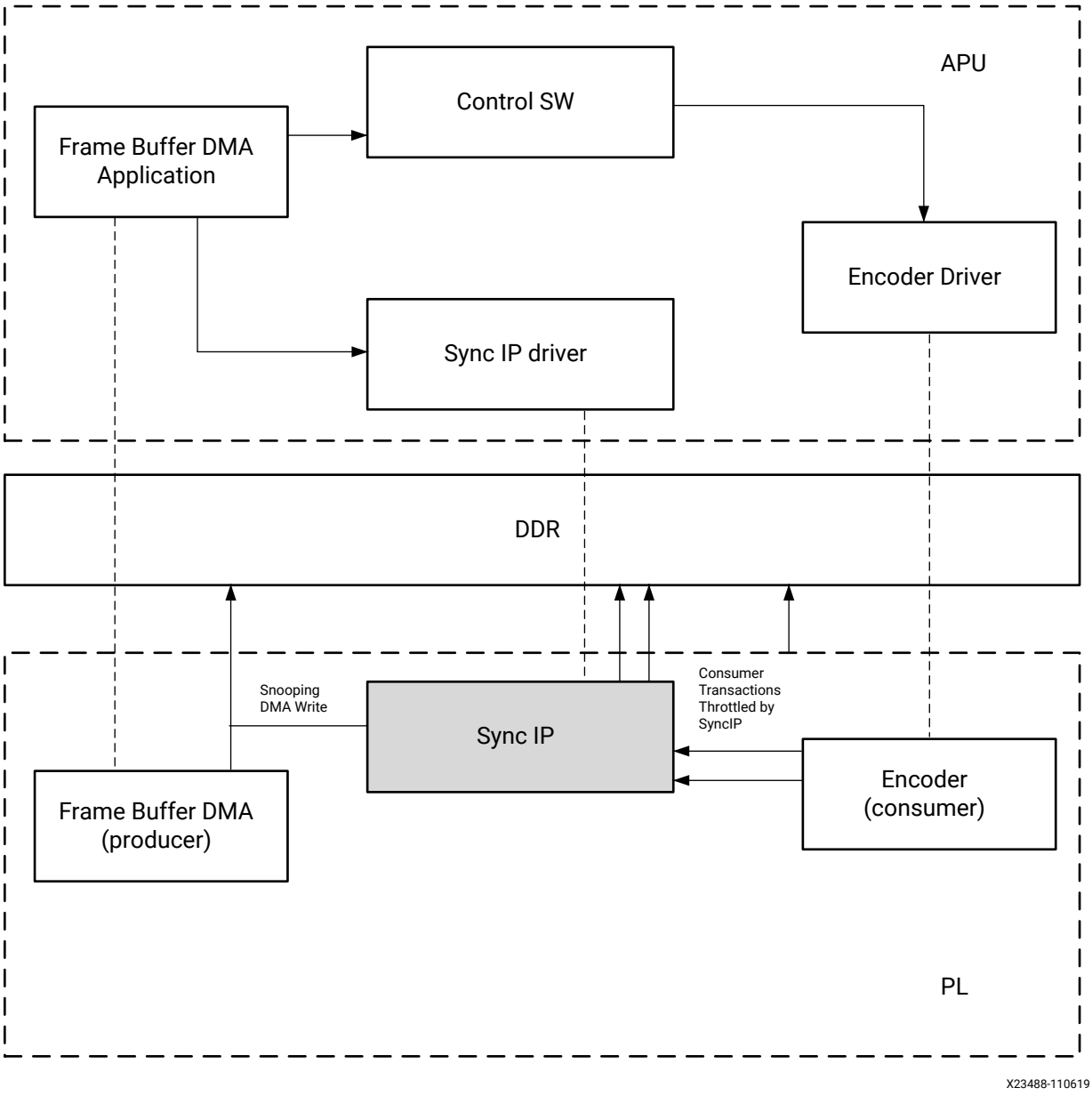
XILINX

Chapter 14

Core Architecture

This section describes the UltraScale architecture-based FPGAs Memory Interface Solutions core with an overview of the modules and interfaces.

Figure 28: **Sync IP Core Architecture Overview**



Sync IP Implementation Overview

The major modules of the VCU Sync IP core are the control, consumer, and producer modules.

Control Module

The control module implements the programming register interface and the ring buffer mechanism of buffer control.

Consumer Module

The consumer module receives AXI read transactions from the consumer and schedules the reads to memory based on the tracking information from the producer. The consumer module implements fencing: It holds the read transactions in the internal FIFO and waits for the producer write transactions to complete. When the producer write transactions for a specific section of memory are completed, the consumer allows the read transactions to go to the DRAM.

Producer Module

The producer module tracks AXI write transactions from the producer and sends write tracking information to the consumer for gating or un-gating the read transactions.

Designing with the Core

This section includes guidelines and additional information to facilitate designing with the core.

Clocking

The Sync IP datapath and control path run in the same clock domain as VCU Xilinx low-latency mode. It is the same as the VCU encoder AXI bus.

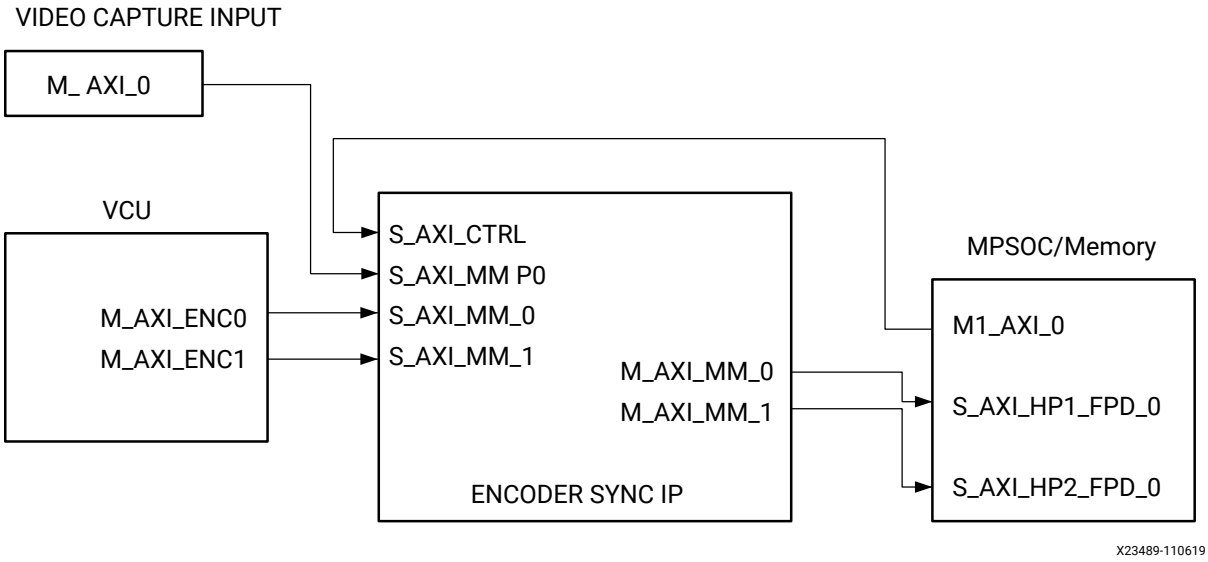
Resets

The Sync IP uses the same reset, the synchronous reset, as the VCU Xilinx low-latency mode. There is no specific reset sequence for the Sync IP. Additionally, the Sync IP has per channel soft reset which is used to clear the internal states of each channel. This soft-reset is software programmable and it is auto cleared internally.

Connectivity with VCU Sync IP

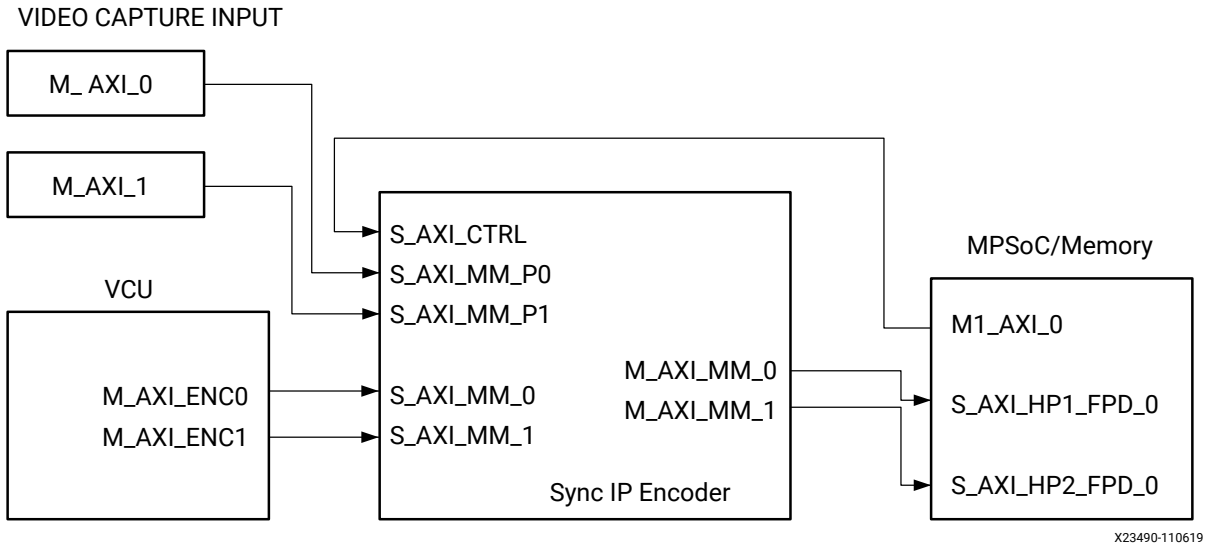
Connectivity with VCU Sync IP (Encoder: Single Stream)

Figure 29: **Video Capture Input**



Connectivity with VCU Sync IP (Encoder: Two Stream)

Figure 30: **Video Capture Input**



Software Flow Overview

The following diagrams show block designs for encoder/decoder separation, multi-stream (2-stream) Xilinx low-latency mode, and single-stream Xilinx low-latency mode.

Figure 31: **Multi-Stream (2-Stream) Xilinx Low Latency**

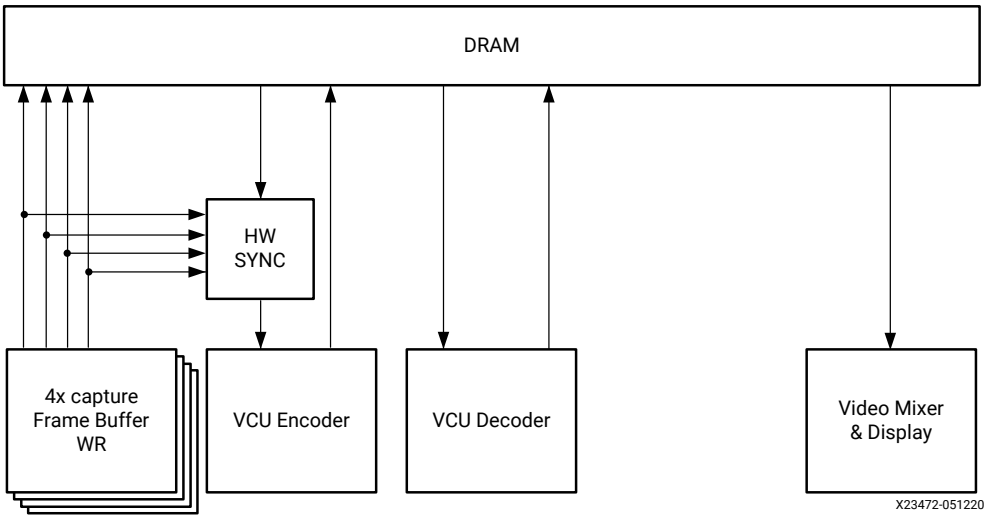
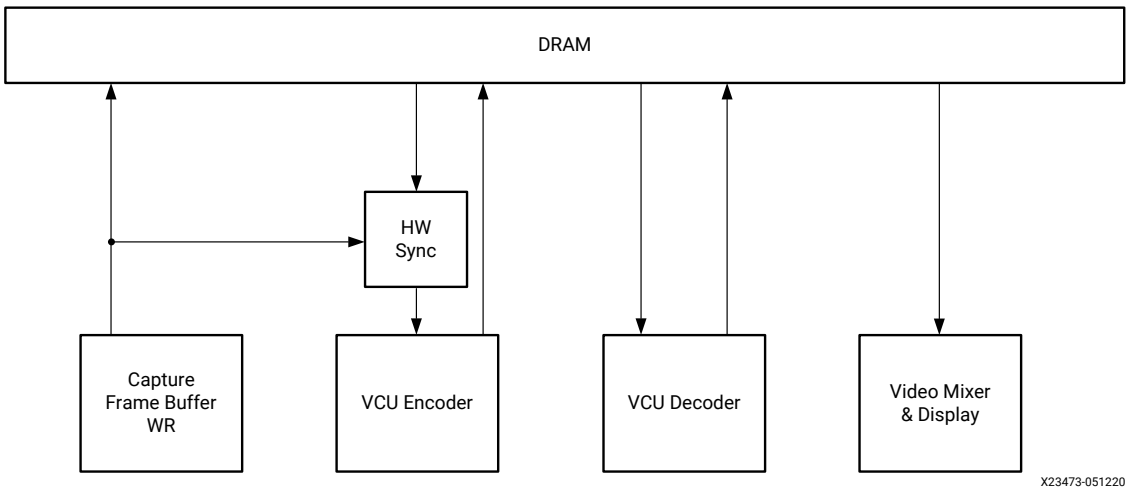


Figure 32: **Single-Stream Xilinx Low Latency**



Software Architecture and Buffer Synchronization Mechanism

The VCU encoder and decoder operate in slice mode for low-latency use cases. An input frame is divided into multiple slices (8 or 16) horizontally, and the encoder generates a slice_done interrupt at the end of every slice. Generated NAL unit data can be passed to a downstream element immediately without waiting for the whole frame to be encoded. The VCU decoder also starts processing data as soon as one slice of data is ready in the decoder circular buffer instead of waiting for complete frame data. The hardware Sync IP shown in the block diagram is responsible for synchronizing the AXI read/writes between capture DMA and the VCU encoder.

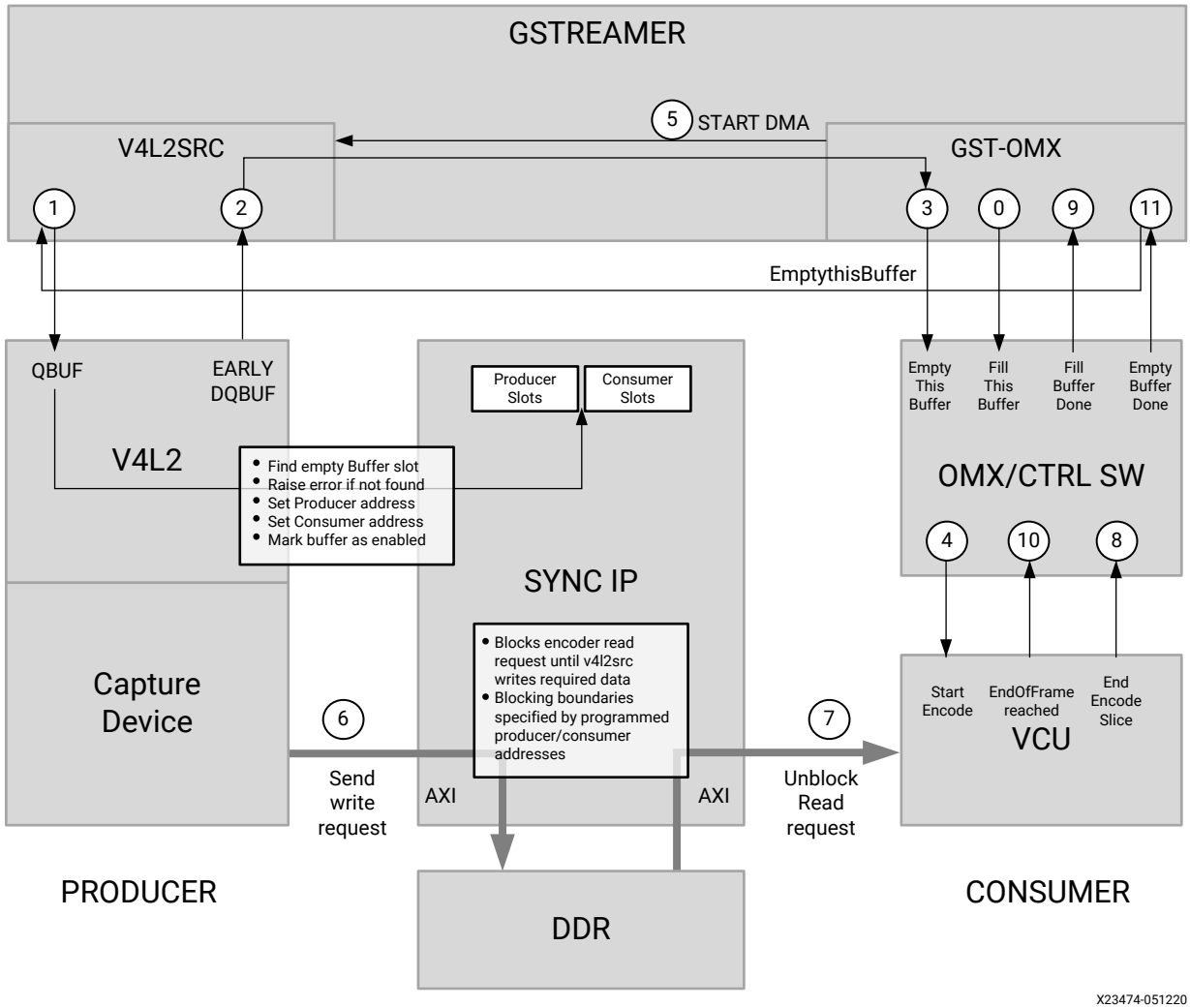
Capture DMA writes video buffers in raster scan order. The Sync IP core monitors the buffer level while capture DMA is writing into DRAM, and allows the encoder to read input buffer data, if the requested data is already written by DMA; otherwise, it blocks encoder AXI transactions until the Capture DMA completes its writes to that section.

On the decoder side, the VCU decoder writes decoded video buffers into DRAM in block raster scan order, and the display reads the data in raster scan order. The software ensures a phase difference of ~frame_period/2 between the VCU decoder start and display read so that the decoder is ahead of the display. This is achieved by releasing decoded buffers early to display stack and waiting 1/2 frame duration at base-sink/kmssink before setting the plane for display.

Sync IP Software Programming Model

Capture to Encode

Figure 33: **Capture to Encode**



The v4l2src component (producer) programs the Sync IP for the capture to encode use case. The capture driver releases the buffer to the encoder module immediately without filling the data (early call-back) when the pipeline goes into a playing state. The Sync IP between the capture device and VCU encoder is responsible for buffer synchronization.

The API flow is as follows:

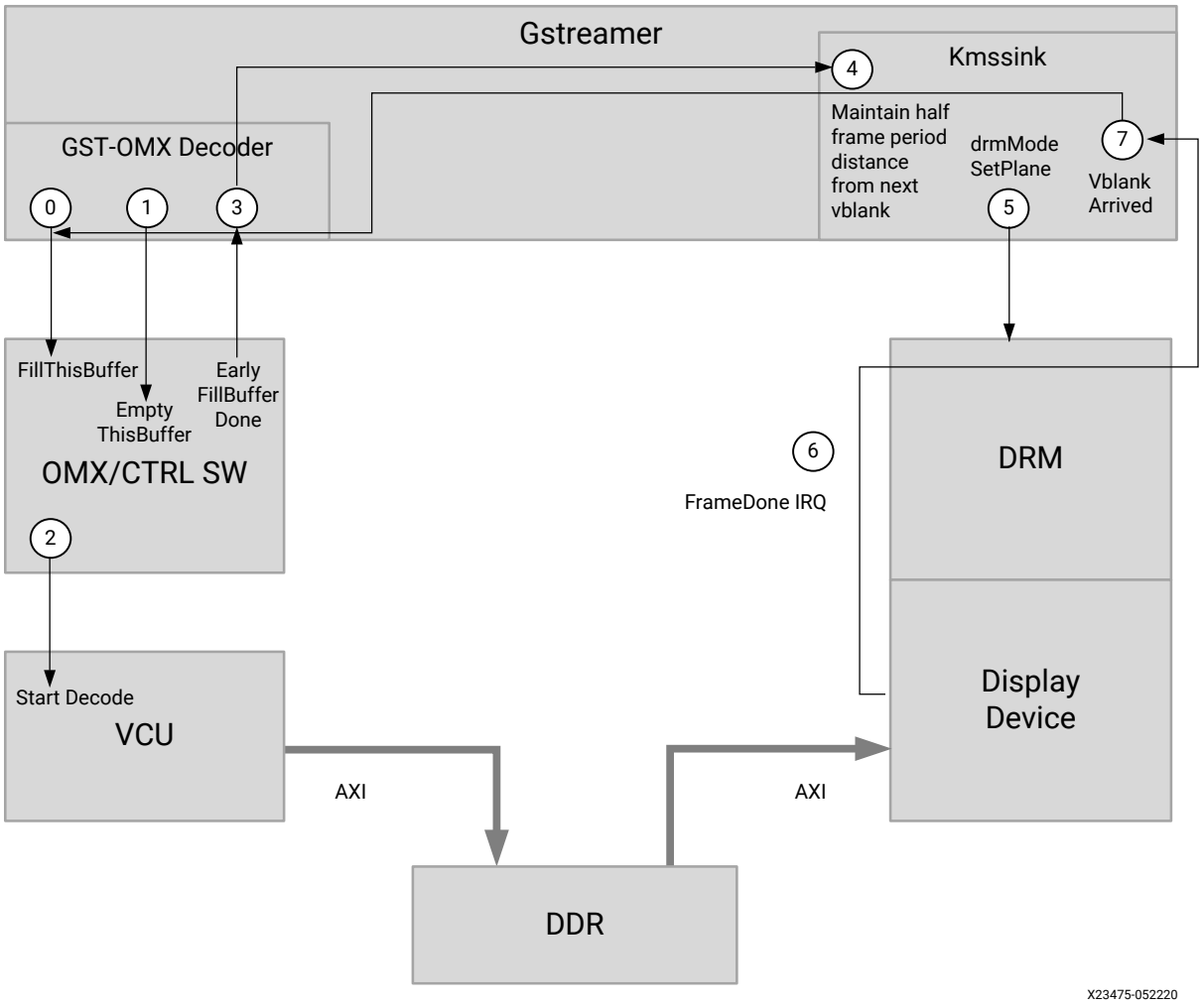
1. v4l2src programs syncip as per address ranges for the first input buffer and enables the sync IP.
2. v4l2src calls VIDIOC_DQBUF and sends empty input buffer to omx encoder using early dequeue mechanism.
3. omx encoder receives this empty buffer and starts generating read request after calling OMX_EmptyThisBuffer and sends an event to v4l2src to start the DMA and start filling input buffer.
4. SyncIP blocks the omx encoder until v4l2src is done writing data corresponding to read request made by omx encoder.
5. Similarly, for consecutive buffers v4l2src programs SyncIP, submits buffer to omx encoder using VIDIOC_DQBUF and syncip block the omx encoder until v4l2src has written sufficient data. This maintains the synchronization between producer (v4l2src) and consumer (omxh265enc/omxh264enc).

   You can find sample Sync IP programming code and API descriptions in the following directories:

```
gst-plugins-good/sys/v4l2/ext/xlnx-ll/
xlnxsync.h (SyncIp driver ioctl header file
xvfbsync.c (SyncIp Application API definition file)
xvfbsync.h (SyncIp Application API header file)
```

Decode to Display

Figure 34: **Decode to Display**



The VCU OMX decoder component has a custom early FillBufferDone callback which releases the output buffer to the next component, the display at the start of decoding, thus allowing concurrent access to the buffer for the decoder and display components and thus reducing the latency. To maintain synchronization between the decoder and display components so that display DMA does not start reading before the decoder has written, the decoder is at least half a frame period ahead of display as per the following API flow.

1. Gst-Omx calls FillThisBuffer with the decoder output buffer.
2. Gst-Omx call EmptyThisBuffer with the decoder input buffer.
3. The OMX decoder generates early FillBufferDone callback and sends out empty buffer to the succeeding video sink kmssink element.
4. The kmssink predicts the time when the vertical blanking signal is going to come based upon the previous vertical blanking time stamp.

5. If the time to reach vertical blanking signal as per prediction is more than half frame period, then there is no extra wait required and buffer can be submitted directly to DRM driver using drmModeSetPlane.
6. If the time to reach vertical blanking signal as per prediction is less than half frame period than kmssink wait for extra time period such that time to reach vertical blanking signal plus the extra time period should match the half frame period.
7. This half frame period window give enough time for decoder to write enough data such that it is always ahead of display by at least half frame period and thus making sure that display DMA does not under-run by reading data before the decoder has written.

Design Flow Steps

This section describes customizing and generating the core, constraining the core, and the simulation, synthesis, and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
- *Vivado Design Suite User Guide: Designing with IP* (UG896)
- *Vivado Design Suite User Guide: Getting Started* (UG910)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900)

Customizing and Generating the Core

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado® Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

Customizing the VCU Sync IP

You can customize the VCU Sync IP core in the Vivado IDE. The following configuration options are available:

- **Configuration page:**
  - Number of Capture channel

    In the Capture channel, there is an option to select from ONE to FOUR. (Default: FOUR)
  - Number of Consumers

    In the consumer, there is an option to select from ONE to FOUR. (Default: TWO)
- **Implementation optimization page:**
  - Memory Size

    This option is to enable the user to select the external memory size of the board. The range of the memory size are fixed. Values are 2GB, 4GB, 16GB and 16K GB. (Default: 16kGB)
  - Enable Multi Clock

    This option is to enable when the design uses more than one clocks for capture, consumer and control side. (Default: Disable)
  - Producer address alignment

    This option is to enable the customer to configure the producer write size. This will aid the resource optimization. The options are fixed. Options are Unaligned, 32-byte aligned, 64-byte aligned, 128-byte aligned, 256-byte aligned, and 512-byte aligned (Default: Unaligned)

The following figure shows the GUI configuration details:

Figure 35: **Configuration Tab**

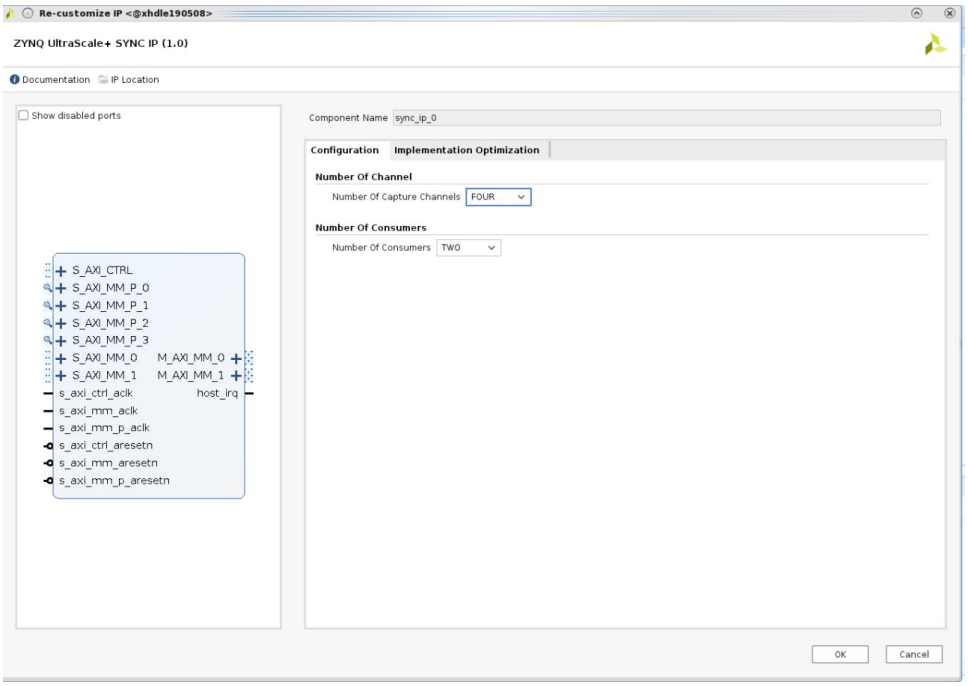
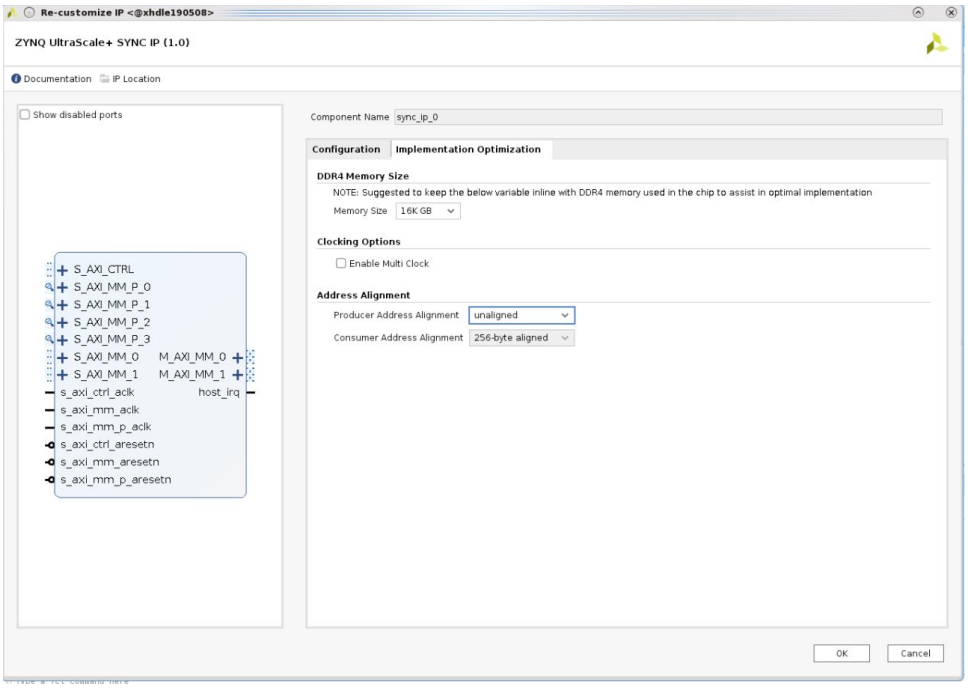


Figure 36: **Implementation Optimization Tab**



Supported Configuration

Table 64: **Encoder**

Number of Producers	Number of Consumers	Clock Frequency for Producer/ Consumer	Clock Frequency for AXI_CNTRL
Producers - 1	Consumers – 1 or 2	300 MHz	100 MHz
Producers - 2	Consumers – 1 or 2	300 MHz	100 MHz
Producers - 3	Consumers – 1 or 2	300 MHz	100 MHz
Producers - 4	Consumers – 1 or 2	300 MHz	100 MHz

Connecting the Sync IP with the VCU LogiCORE IP

The following diagrams show how the Sync IP is connected to the VCU LogiCORE IP.

Figure 37: **Sync IP Single-Stream VCU Encoder Connections**

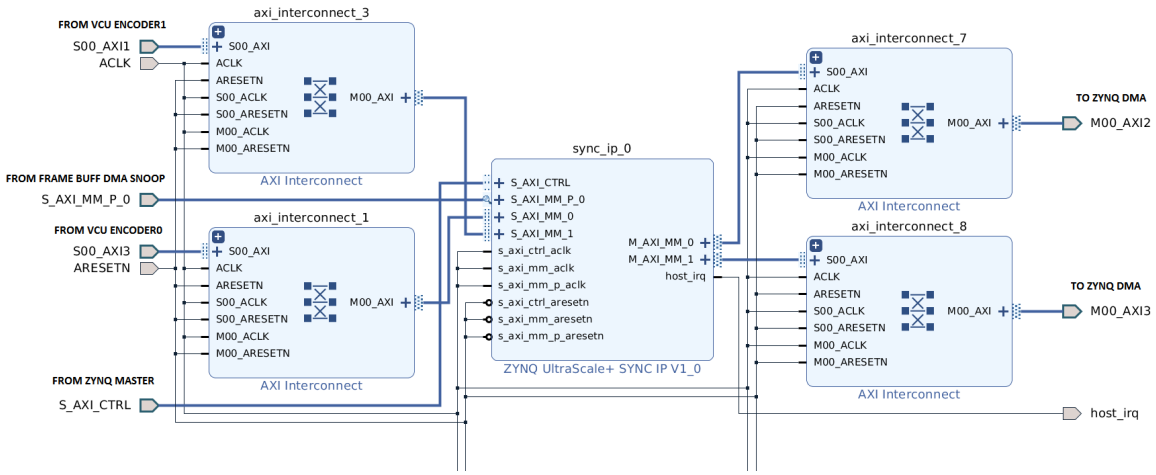


Figure 38: **Sync IP Four-Stream VCU Encoder**

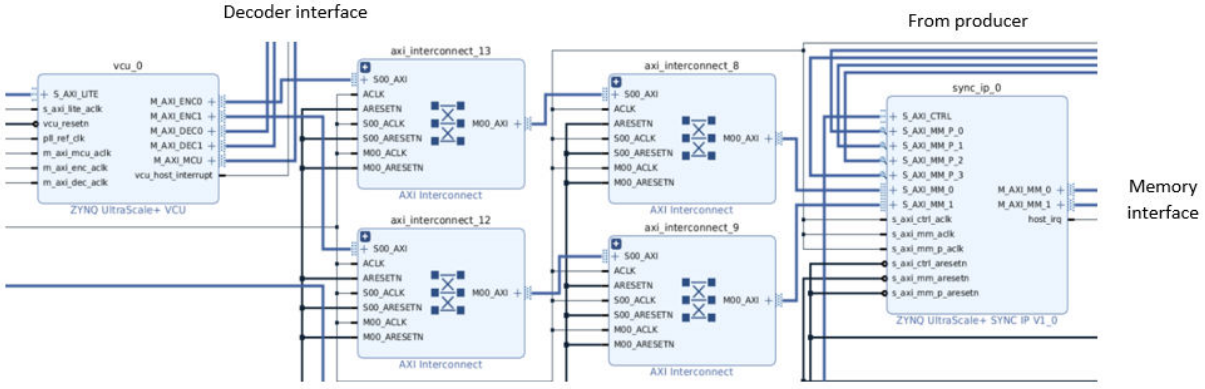
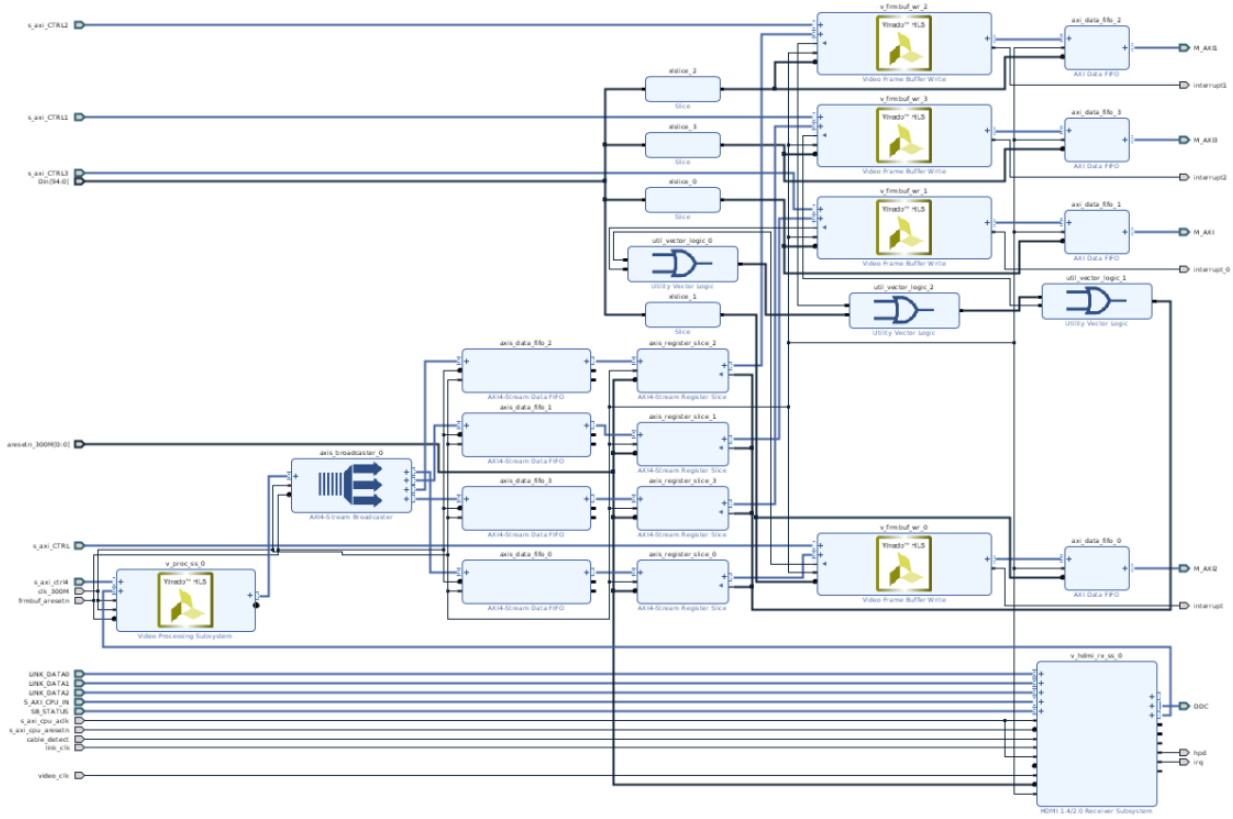


Figure 39: **HDMI Input Four-Stream Connections**



Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

None

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

For more information on clocking, see Clocking.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

N/A

Simulation

Simulation of the VCU Sync IP is not supported.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

Using the Release Package

The VCU TRD 2020.1 the version consists of four design-modules with the Sync IP as part of the design.

- **Xilinx low-latency mode PS DDR NV12 HDMI Audio Video Capture and Display:** This is a VCU based HDMI design to showcase ultra low latency support using Sync IP, encoding and decoding with PS DDR for NV12 format. This module also supports single-stream audio. See the wiki page and build and run the flow of this design module.
- **Xilinx low-latency mode PL DDR NV16 HDMI Video Capture and Display:** This is a VCU based HDMI design to showcase ultra low latency support using Sync IP, encoding with PS DDR and decoding with PL DDR for NV16 format. See the wiki page and build and run the flow of this design module.
- **Xilinx low-latency mode PL DDR XV20 HDMI Video Capture and Display:** This is a VCU based HDMI design to showcase ultra low latency support using Sync IP, encoding with PS DDR and decoding with PL DDR for XV20 format. See the wiki page and build and run the flow of this design module.
- **Xilinx low-latency mode PL DDR XV20 SDI Video Capture and Display:** This is a VCU based SDI design to showcase ultra low latency support using Sync IP, encoding with PS DDR and decoding with PL DDR for XV20 format. See the wiki page and build and run the flow of this design module.

XILINX

Section IV

Performance and Debugging

XILINX

Chapter 20

Latency in the VCU Pipeline

The Video Codec Unit (VCU) is designed to support video streams at resolutions up to 3840×2160 pixels at 60 frames per second (4K UHD at 60 Hz) with group of pictures (GOP) B-frames (hardest case). Latency is defined between frame boundaries and all GOP types are allowed. Some GOP types require display reordering buffers.

Glass-to-Glass Latency

As illustrated in the following figure, glass-to-glass latency (L) is the sum of the following:

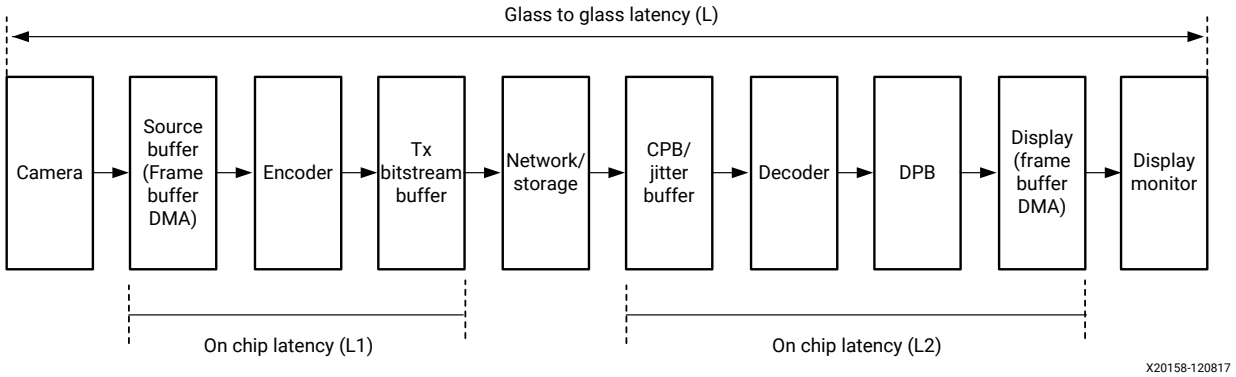
- Camera latency
- On-chip latency (L1)
  - Source frame buffer DMA latency
  - Encoder latency
  - Transmission bitstream buffer latency
- Network or storage latency
- On-chip latency (L2)
  - Coding Picture Buffer (CPB)/jitter buffer latency
  - Decoder latency
  - Decoded picture buffer (DPB) latency
  - Display frame buffer DMA latency
- Display monitor latency

When B-frames are enabled, one frame of latency is incurred for each B-frame due to the usage of the reordering buffer. To optimize the CPB latency, a handshaking mechanism in PL is required between decoder and the display DMA. It is assumed that both capture side and display side works on a common VSYNC timing.

VSYNC timing can be asynchronous and a clock recovery mechanism is needed to synchronize source timing with sync.

With independent VSYNC timing and without clock recovery mechanism, it requires one additional frame latency to synchronize with the display devices.

Figure 40: **Glass to Glass Latency**



***Note*:** These numbers do not include the latency information from the interconnect in the fabric. For more information on memory parts, see the *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150).

VCU Latency Modes

The VCU supports four latency modes: normal latency, reduced latency (also called no-reordering mode), low latency, and Xilinx low latency modes. The pipeline instantaneous latency may vary depending upon the frame structure, encoding standard, levels, profiles, and target bitrate.

- **Normal-Latency:** The VCU encoder and decoder works at frame level. All possible frame types (I, P, and B) are supported and there is no restriction on GOP structure. The end-to-end latency depends on the profile/level, GOP structure, and the number of internal buffers used for processing. This is standard latency and can be used with any control rate mode.
- **No Reordering (Reduced-Latency):** The VCU encoder works at frame level. Hardware rate control is used to reduce the bitrate variations. I-only, IPPP, and low-delay-P are supported. There is no output reordering, thus reducing latency on the decoder side. The VCU continues to operate at frame level.
- **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 it will be available immediately for next element processing. So, 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.

- **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 full frame. In the Xilinx low-latency mode, the capture and display also work at subframe level thus reducing 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 once the producer is done writing the data required for that read request. This functionality to maintain synchronization is managed by a separate IP block called the Xilinx synchronization IP.

  Similarly, the decoder and the display are also allowed to have concurrent access to the same buffer, but here there is no separate hardware synchronization IP block between them. The software handles the synchronization by making sure that buffer starts getting displayed only when the decoder has written at least half a frame period of data.

  Similar to the low-latency mode, the Xilinx low-latency also supports a maximum of four streams for the encoder and two streams for the decoder. See Section III: VCU Sync IP v1.0 for more information.

The maximum number of streams should be equivalent to 4kp60 bandwidth. Following are the possible combinations of latency modes:

- Possible combination for normal and reduced latency:
  - For live and file sources:
    - One instance of 3840x2160p60
    - Two instances of 3840x2160p30
    - Four instances of 1920x1080p60
    - Eight instances of 1920x1080p30
  - For file source only:
    - 32 instances of 640x480@30
    - 32 instances of 720x480@30
- Possible combinations for low latency and Xilinx low latency:
  - Four instances of 1920x1080p60 streams at encoder and two instances of 1920x1080p60 streams at decoder
  - Two instances of 3840x2160p30 streams at encoder and two instances of 3840x2160p30 streams at decoder

Xilinx Low Latency Limitations

The Xilinx low-latency mode has the following limitation:

- The VCU encoder and decoder use mono-threaded micro-controller based scheduler for sending commands to underlying hardware IP blocks. For the multi-stream environment it is possible that command requests for one stream can get blocked if the scheduler is busy serving commands for another stream in parallel. This may cause momentary spike in latency. Also the spike in latency can occur in an already running pipeline whenever a new pipeline is launched or closed as the command requests for running stream can get blocked while a new channel is being created for encoding a new stream, or an existing channel for an already running stream gets destroyed during closure. Latencies may go higher as the number of streams increase.

Encoder and Decoder Latencies with Xilinx Low Latency Mode

Encoder

The following table shows the latency numbers for 1x, 2x and 4x encoder latencies. This data is for NV12 1080p60 HEVC It captures the instantaneous latency of the pipeline by capturing number of samples occurring at different ranges.

Table 65: **Encoder Latencies**

Range (ms)	1x[1]	2x[2]	4x[3]
[0-1]	0	0	0
[1-2]	0	0	0
[2-3]	1818	813	110
[3-4]	19892	21127	2468
[4-5]	0	7	10443
[5-6]	0	1	9148
[6-7]	0	0	9
[7-8]	0	1	7
[8-9]	0	2	1
[9-10]	0	0	0
[10-11]	0	0	1

Notes:

1. 1x: Single stream use-case
2. 2x: Use case with two streams (e.g. video0 and video1) running in parallel
3. 4x: Use case with four streams (e.g. video0, video1, video2 and video3) running in parallel

Decoder

The following table shows the latency numbers for 1x and 2x decoder latencies. This data is for NV12 4kp30 HEVC:

Table 66: **Decoder Latencies**

Range (ms)	1x[1]	2x[2]
[0-1]	0	0
[1-2]	0	0
[2-3]	0	0
[3-4]	0	0
[4-5]	0	0
[5-6]	0	0
[6-7]	0	2
[7-8]	4	6
[8-9]	10820	8737
[9-10]	4	2130
[10-11]	4	3

Notes:

1. 1x: Single stream use-case
2. 2x: Use case with two streams (e.g. video0 and video1) running in parallel

Recommended Parameters for Xilinx Low-latency Mode

This section describes some recommended settings to be used for encoder and decoder to run Xilinx low-latency mode pipelines with optimum latency.

- Use encoder and decoder parameters as shown in the following table.

Table 67: **Parameters for Xilinx Low-Latency Mode**

Codec and Streams	Encoder Parameters				kmssink Parameters	
	target-bitrate	periodicity-idr	cpb-size	initial-delay	max-lateness	processing-deadline
All 1x AVC/ HEVC[2]	25000	240	500	250	5 ms (default)	0 ms (default)
2x 1080p AVC/ HEVC[2]	12500	240	500	250	5 ms (default)	0 ms (default)
4x 1080p AVC[3]	6250	240	500	250	5 ms (default)	0 ms (default)
4x 1080p HEVC[3]	6250	240	500	250	5 ms (default)	5 ms (NV12) 7 ms (XV20)
2x 4kp30 AVC[2]	12500	240	500	250	5 ms (default)	0 ms (default)
2x 4kp30 HEVC[2]	12500	240	500	250	10 ms	0 ms (default)

Notes:

1. 1x: Single stream use-case
2. 2x: Use case with two streams (for example, video0 and video1) running in parallel
3. 4x: Use case with four streams (for example, video0, video1, video2 and video3) running in parallel

The following are the common encoder parameters for Xilinx low latency pipelines:

Table 68: **Encoder Parameters**

num-slices	control-rate	gop-mode	filler-data	prefetch-buffer	gdr-mode
8	low-latency	low-delay-p	0	True	horizontal

Note: Use the decoder parameter `internal-entropy-buffers=3` for XV20 2x 4kp30 use case only. Use decoder parameter `internal-entropy-buffers=3` for XV20 2x 4kp30 due to memory constraints. Use `processing-deadline=5 ms` for 4x HEVC use-cases as shown in above table to mitigate high latencies in the multi-stream environment.

- For 4x hevc serial pipelines, it is recommended to set `processing-deadline = 5 ms` for NV12 and `processing-deadline = 7 ms` for XV20 to mitigate higher latencies due to multistream as mentioned in the table.
- It is recommended to set the decoder parameter `internal-entropy-buffers=3` for XV20 2x 4kp30 use case only due to memory constraints.
- It is recommended to use more output buffers for HEVC use-case in multistream scenarios to optimize the latencies further. This is possible by prefixing pipeline with ENC_EXTRA_OP_BUFFERS as shown in the following pipes:

```
ENC_EXTRA_OP_BUFFERS=10 gst-launch-1.0 -v v4l2src io-mode=dmabuf
device=/dev/video0 ! video/x-raw/(memory:XLNXLL/), width=3840,
height=2160, format=NV12, framerate=30/1 ! omxh265enc control-rate=low-
latency target-bitrate=6250 filler-data=0 prefetch-buffer=true num-
slices=8 periodicity-idr=240 cpb-size=500 gdr-mode=horizontal initial-
delay=250 gop-mode=low-delay-p ! video/x-h265, alignment=nal ! queue max-
size-buffers=0 ! omxh265dec low-latency=1 ! video/x-raw/
(memory:XLNXLL/) ! queue max-size-bytes=0 ! fpsdisplaysink name=fpssink
text-overlay=false 'video-sink=kmssink bus-id=a0070000.v_mix plane-id=33
max-lateness=5000000 hold-extra-sample=1 show-preroll=false render-
rectangle=<0,0,3840,2160> sync=true processing-deadline=0' sync=true -v
> /run/pipeline0_$filename.txt 2>&1 &
```

```
ENC_EXTRA_OP_BUFFERS=10  gst-launch-1.0 -v v4l2src io-mode=dmabuf
device=/dev/video1 ! video/x-raw/(memory:XLNXLL/), width=3840,
height=2160, format=NV12, framerate=30/1 ! omxh265enc control-rate=low-
latency target-bitrate=6250 filler-data=0 prefetch-buffer=true num-
slices=8 periodicity-idr=240 cpb-size=500 gdr-mode=horizontal initial-
delay=250 gop-mode=low-delay-p ! video/x-h265, alignment=nal ! queue max-
size-buffers=0 ! omxh265dec low-latency=1 ! video/x-raw/
(memory:XLNXLL/) ! queue max-size-bytes=0 ! fpsdisplaysink name=fpssink
text-overlay=false 'video-sink=kmssink bus-id=a0070000.v_mix plane-id=34
max-lateness=5000000 hold-extra-sample=1 show-preroll=false render-
rectangle=<1920,0,3840,2160> sync=true processing-deadline=0' sync=true -
v > /run/pipeline1_$filename.txt 2>&1 &
```

VCU End-to-End Latency

The following table shows the latency for the VCU pipeline stages.

Table 69: **VCU End-to-End Latency**

Use Case[5]	Capture	Encode (HEVC / AVC)	Decode	Display
Normal Latency	16.6 ms	18 ms / 35 ms	200 ms	16.6 ms
Reduced Latency	16.6 ms	18 ms / 35 ms	50 ms	16.6 ms
Low Latency	16.6 ms	4 ms / 10 ms	17 ms	16.6 ms
Xilinx Low-Latency[3, 4]	1 ms	4 ms / 10 ms	9 ms	16.6 ms

Notes:

1. The values in the above table are estimated numbers for 4kp60.
2. In case of normal latency, the decode reported latency is calculated according to the below formula. The DPB size here is depends on profile and level reported latency = DPB size + internal entropy buffers + reconstruction buffers + concealment buffers.
3. Xilinx low-latency is achieved using Sync IP. For more information, see Section III: VCU Sync IP v1.0.
4. Xilinx low-latency supports 25 Mb/s for single stream of 2160p60 and 6 Mb/s for four streams of 1080p60.
5. The exact worst case latencies depend upon the input content and system load. Latency numbers mentioned in this table correspond to the theoretical latencies reported by the respective elements. The latencies of elements should stay under the reported ones.

Latency Numbers

Latency numbers are derived from the following equations:

Table 70: **Latency Numbers**

Use Case	Equation[1]	Latency
v4l2src	1 frame period for the capture buffer	16.66 ms
omxh265enc (HEVC encoder)	1 frame period for input capture buffer rounded to ms + 1 ms margin = 17 ms + 1 ms	18 ms
omxh264enc (AVC encoder)	1 frame period for input capture buffer rounded to ms + 1 intermediate buffer + 1 ms margin = 17 ms + 17 ms + 1 ms	35 ms
Normal Latency [omxh265dec & omxh264dec (AVC/HEVC Decoder)]	DPB size (5 for default level i.e. 5.2) + internal entropy buffers (default=5) + reconstruction buffers (default=1)+ concealment buffers (default=1) = 12 *16.66 ms	200 ms
Reduced Latency	1 frame period to insert frame into decoder + 1 frame period to decode + 1 frame period margin = 16.6 ms + 16.6 ms + 16.6 ms	50 ms

Notes:

1. All the equations in this table have been calculated for 60fps video.

Usage

Normal Latency

This is standard latency and can be used with any of the rate control mode.

- **Gstreamer Encoder Pipeline:**

```
gst-launch-1.0 v4l2src io-mode=4 device=/dev/video0 ! video/x-raw,
width=1920, height=1080, framerate=60/1, format=NV12 ! omxh264enc !
video/x-h264, alignment=au! Fakesink
```

- **Gstreamer Decoder Pipeline:**

```
gst-launch-1.0 filesrc location="input-file.mp4" ! qtdemux name=demux
demux.video_0 ! h264parse ! video/x-h264, alignment=au ! omxh264dec low-
latency=0 ! queue max-size-bytes=0 ! fakevideosink
```

- **Ctrl-SW Command:** The v4l2 capture control software encoder application demonstrates Xilinx's Low-Latency feature using the VCU **ctrlsw** APIs. It is an enhanced version of normal VCU **ctrlsw** app (**ctrlsw_encoder**).

  Refer to this wiki page section for commands and additional details.

No Reordering (Reduced Latency)

For encoder, set rate control mode to low-latency with I only, IPPP, or low-delay-P mode. For decoder, set low-latency parameter with alignment to AU through caps for sink pad.

- **Gstreamer Encoder Pipeline:**

```
gst-launch-1.0 v4l2src io-mode=4 device=/dev/video0 ! video/x-raw,
width=1920, height=1080, framerate=60/1, format=NV12 ! omxh264enc gop-
mode= (basic or low-delay-p), b-frames=0 control-rate=low-latency !
fakesink
```

- **Gstreamer Decoder Pipeline:**

```
gst-launch-1.0 filesrc location="input-file.mp4" ! qtdemux name=demux
demux.video_0 ! h264parse ! video/x-h264, alignment=au ! omxh264dec low-
latency=1 ! queue max-size-bytes=0 ! fakevideosink
```

- **Ctrl-SW command:** At the control software level, it can be specified using command line arguments.

```
ctrlsw_decoder -avc -in input-avc-file.h264 -out ouput.yuv --no-reordering
```

***Note*:** There is no argument at encoder side for no-reordering.

Low Latency

This mode supports sub-frame latency. For encoder, recommended to set rate control mode to low-latency rate for best performance and pass alignment = NAL through caps at encoder source pad. For decoder, set low-latency parameter with alignment set to NAL through caps for sink pad. Low latency can be enabled at the encoder and decoder sides as follows:

```
omxh264enc ! video/x-h264, alignment=nal ! ... ! omxh264dec low-
latency=1 ! ...
```

- **Gstreamer Encoder Pipeline:**

```
gst-launch-1.0 v4l2src io-mode=4 device=/dev/video0 ! video/x-raw,
width=1920, height=1080, framerate=60/1, format=NV12 ! omxh264enc gop-
mode= (default or low-delay-p), b-frames=0 control-rate=low-latency !
video/x-h264, alignment=nal ! fakesink
```

- **Gstreamer Decoder Pipeline:**

```
gst-launch-1.0 filesrc location="input-file.mp4" ! qtdemux name=demux
demux.video_0 ! h264parse! video/x-h264, alignment=nal ! omxh264dec low-
latency=1 ! queue max-size-bytes=0 ! fakevideosink
```

- **Ctrl-SW command:** At the control software level, it can be specified through command line arguments.

```
ctrlsw_encoder -cfg encode_simple.cfg --slicelat
ctrlsw_decoder -avc -in input-avc-file.h264 -out ouput.yuv -slicelat
```

Xilinx Low Latency

Xilinx low latency can be enabled by adding an extra caps video/x-raw(memory:XLNXLL) at v4l2src and decoder side as follows

```
v4l2src ! video/x-raw(memory:XLNXLL), width=3840, height=2160, ... !
omxh264enc ! video/x-h264, alignment=nal ! ... ! omxh264dec low-latency=1 !
video/x-raw(memory:XLNXLL) ...
```

- **4x 1080p60 AVC NV12 Xilinx Low-Latency Pipeline:**

```
gst-launch-1.0 -v v4l2src io-mode=dmabuf device=/dev/video0 ! video/x-raw/
(memory:XLNXLL/), width=1920, height=1080, format=NV12, framerate=60/1 !
queue ! omxh264enc control-rate=low-latency target-bitrate=6000 prefetch-
buffer=TRUE num-slices=8 periodicity-idr=240 cpb-size=500 initial-
delay=250 gdr-mode=horizontal gop-mode=low-delay-p ! video/x-h264,
alignment=nal ! queue max-size-buffers=0 ! omxh264dec low-latency=1 !
video/x-raw/(memory:XLNXLL/) ! queue max-size-bytes=0 ! fpsdisplaysink
name=fpssink text-overlay=false 'video-sink=kmssink bus-id=a00c0000.v_mix
plane-id=30 hold-extra-sample=1 show-preroll=false sync=true' sync=true -v
```

```
gst-launch-1.0 -v v4l2src io-mode=dmabuf device=/dev/video1 ! video/x-raw/
(memory:XLNXLL/), width=1920, height=1080, format=NV12, framerate=60/1 !
queue ! omxh264enc control-rate=low-latency target-bitrate=6000 prefetch-
buffer=TRUE num-slices=8 periodicity-idr=240 cpb-size=500 initial-
delay=250 gdr-mode=horizontal gop-mode=low-delay-p ! video/x-h264,
alignment=nal ! queue max-size-buffers=0 ! omxh264dec low-latency=1 !
video/x-raw/(memory:XLNXLL/) ! queue max-size-bytes=0 ! fpsdisplaysink
name=fpssink text-overlay=false 'video-sink=kmssink bus-id=a00c0000.v_mix
plane-id=31 hold-extra-sample=1 show-preroll=false sync=true' sync=true -v
```

```
gst-launch-1.0 -v v4l2src io-mode=dmabuf device=/dev/video2 ! video/x-raw/
(memory:XLNXLL/), width=1920, height=1080, format=NV12, framerate=60/1 !
queue ! omxh264enc control-rate=low-latency target-bitrate=6000 prefetch-
buffer=TRUE num-slices=8 periodicity-idr=240 cpb-size=500 initial-
delay=250 gdr-mode=horizontal gop-mode=low-delay-p ! video/x-h264,
alignment=nal ! fpsdisplaysink name=fpssink text-overlay=false 'video-
sink=fakesink sync=true' sync=true -v
```

```
gst-launch-1.0 -v v4l2src io-mode=dmabuf device=/dev/video3 ! video/x-raw/
(memory:XLNXLL/), width=1920, height=1080, format=NV12, framerate=60/1 !
queue ! omxh264enc control-rate=low-latency target-bitrate=6000 prefetch-
buffer=TRUE num-slices=8 periodicity-idr=240 cpb-size=500 initial-
delay=250 gdr-mode=horizontal gop-mode=low-delay-p ! video/x-h264,
alignment=nal ! fpsdisplaysink name=fpssink text-overlay=false 'video-
sink=fakesink sync=true' sync=true -v
```

- **4kp60 AVC NV12 Xilinx Low-Latency Serial Pipeline:**

```
gst-launch-1.0 -v v4l2src io-mode=dmabuf device=/dev/video0 ! video/x-raw\
(memory:XLNXLL\), width=3840, height=2160, format=NV12, framerate=60/1 !
omxh264enc num-slices=8 control-rate=low-latency gop-mode=low-delay-p
target-bitrate=25000 cpb-size=500 gdr-mode=horizontal initial-delay=250
periodicity-idr=240 filler-data=0 prefetch-buffer=true ! video/x-h264,
alignment=nal ! queue max-size-buffers=0 ! omxh264dec low-latency=1 !
video/x-raw\(memory:XLNXLL\) ! queue max-size-bytes=0 ! fpsdisplaysink
name=fpssink text-overlay=false 'video-sink=kmssink bus-id=a0070000.v_mix
plane-id=33  hold-extra-sample=1 show-preroll=false sync=true' sync=true -
v
```

- **2x 4kp30 HEVC NV12 Xilinx Low-Latency Serial Pipeline:**

```
ENC_EXTRA_OP_BUFFERS=10 gst-launch-1.0 -v v4l2src io-mode=dmabuf
device=/dev/video0 ! video/x-raw\(memory:XLNXLL\), width=3840,
height=2160, format=NV12, framerate=30/1 ! omxh265enc control-rate=low-
latency target-bitrate=12500 filler-data=0 prefetch-buffer=true num-
slices=8 periodicity-idr=240 cpb-size=500 gdr-mode=horizontal initial-
delay=250 gop-mode=low-delay-p ! video/x-h265, alignment=nal ! queue max-
```

```
size-buffers=0 ! omxh265dec low-latency=1 ! video/x-raw\(memory:XLNXLL
\) ! queue max-size-bytes=0 ! fpsdisplaysink name=fpssink text-
overlay=false 'video-sink=kmssink bus-id=a0070000.v_mix plane-id=33 hold-
extra-sample=1 show-preroll=false render-rectangle=<0,0,1920,2160> max-
lateness=10000000 sync=true' sync=true -v > /run/pipeline0.txt 2>&1 &
```

```
ENC_EXTRA_OP_BUFFERS=10 gst-launch-1.0 -v v4l2src io-mode=dmabuf
device=/dev/video1 ! video/x-raw\(memory:XLNXLL\), width=3840,
height=2160, format=NV12, framerate=30/1 ! omxh265enc control-rate=low-
latency target-bitrate=12500 filler-data=0 prefetch-buffer=true num-
slices=8 periodicity-idr=240 cpb-size=500 gdr-mode=horizontal initial-
delay=250 gop-mode=low-delay-p ! video/x-h265, alignment=nal ! queue max-
size-buffers=0 ! omxh265dec low-latency=1 ! video/x-raw\(memory:XLNXLL
\) ! queue max-size-bytes=0 ! fpsdisplaysink name=fpssink text-
overlay=false 'video-sink=kmssink bus-id=a0070000.v_mix plane-id=34  hold-
extra-sample=1 show-preroll=false render-rectangle=<1920,0,3840,2160> max-
lateness=10000000 sync=true' sync=true -v > /run/pipeline1_$filename.txt
2>&1 &
```

- **4kp60 AVC NV12 Xilinx Low-Latency Streaming Pipeline:**
  - **Client:**

```
gst-launch-1.0 udpsrc port=5004 buffer-size=60000000
caps="application/x-rtp, media=video, clock-rate=90000, payload=96,
encoding-name=H264" ! rtpjitterbuffer latency=7 ! rtph264depay !
h264parse ! video/x-h264, alignment=nal ! omxh264dec low-latency=1 !
video/x-raw\(memory:XLNXLL\) ! queue max-size-bytes=0 ! fpsdisplaysink
name=fpssink text-overlay=false video-sink="kmssink bus-
id=a0070000.v_mix plane-id=33" sync=true -v > /run/client0_pipeline.log
2>&1 &
```

  - **Server:**

```
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 !
omxh264enc 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-h264,
alignment=nal ! rtph264pay ! udpsink buffer-size=60000000
host=192.168.0.2 port=5004 max-lateness=-1 qos-dscp=60 async=false max-
bitrate=120000000 -v > /run/server0_pipeline.log 2>&1 &
```

- **Ctrl-SW command:** The Xilinx Low Latency mode is not supported in the example control software encoder application ctrlsw_encoder, that comes with petalinux as this latency mode is used mainly for live use-cases that the ctrlsw_encoder does not support. However, there is a separate v4l2 capture control software encoder application implemented which also supports Xilinx's Low-Latency feature using the ctrlsw APIs. It is an extension to the normal VCU ctrlsw app (ctrlsw_encoder) to support live record and streamout use-cases. Please refer to this wiki page for commands and additional details.

Latency

Measuring Total Latency of a Pipeline

The GStreamer framework includes a tracing module that helps determine source to sink latencies by injecting custom events at source and processing them at sinks. It effectively measures the time between when the buffer is produced by the source pad of the first element and when it reaches the sink pad of the last element. Consider the following pseudo pipeline:

```
source! element! element! sink
```

The GStreamer tracing module measures the latency introduced by element ! element, which is the inner processing of the pipeline. The rest (introduced by the capture source and display device) cannot be measured accurately by the user space.

Each element reports to GStreamer the maximum latency time it takes to output the buffer after it is received. GStreamer uses this information for synchronization.

The latency tracer module gives instantaneous latencies which might not be the same as the reported latencies. The latencies might 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.

You can measure the average userland latency using the following formulas:

$$\text{Average userland latency} = \text{MAX}\ (\text{Pipeline reported latency}, \text{AVG}\ (\text{Instantaneous latency}))$$

The following section provides an example of measuring the latency-related data for the AVC 4kp60 capture→encode→decode→display use case. Follow the setup steps described in the wiki.

Checking Reported Latency of the Pipeline

To check the reported latencies, run the following command:

Pipeline

```
$GST_DEBUG="basesink:5" GST_DEBUG_FILE="/run/reported_serial_4k_avc.txt"
gst-launch-1.0 -v v4l2src io-mode=dmabuf-import device=/dev/video0 num-
buffers=1000 '!' video/x-raw, width=3840, height=2160, format=NV12,
framerate=60/1 '!' omxh264enc control-rate=low-latency target-bitrate=20000
filler-data=0 prefetch-buffer=TRUE num-slices=16 '!' video/x-h264,
alignment=nal '!' queue max-size-buffers=0 '!' omxh264dec low-latency=1 '!'
queue max-size-bytes=0 '!' fpsdisplaysink name=fpssink text-overlay=false
'video-sink=kmssink bus-id=a0070000.v_mix plane-id=32 hold-extra-sample=1
show-preroll=false sync=true' sync=true -v
```

Check Reported Latency

```
grep -inr "latency set" serial_4k_avc.txt
335:0:00:00.393117490 19139  0x558a470470 DEBUG        basesink
gstbasesink.c:4487:gst_base_sink_send_event:<kmssink0> latency set to
0:00:00.000000000
603:0:00:00.404372324 19139  0x558a470470 DEBUG        basesink
gstbasesink.c:4487:gst_base_sink_send_event:<kmssink0> latency set to
0:00:00.000000000
892:0:00:00.633146383 19139  0x558a470470 DEBUG        basesink
gstbasesink.c:4487:gst_base_sink_send_event:<kmssink0> latency set to
0:00:00.000000000
6239:0:00:00.979049764 19139  0x7f800041e0 DEBUG         basesink
gstbasesink.c:4487:gst_base_sink_send_event:<kmssink0> latency set to
0:00:00.016000000
```

Pipeline reported latency = Non-zero value assigned to the latency set field in above string=16 ms

Checking Instantaneous Latencies

Measuring Instantaneous Latency for Serial Pipeline

To check the instantaneous latencies, run the following command:

```
GST_DEBUG="GST_TRACER:7" GST_TRACERS=latency GST_DEBUG_FILE="/run/
instantaneous_latency_serial_4k_avc.txt" gst-launch-1.0 -v v4l2srcio-
mode=dmabuf-import device=/dev/video0 num-buffers=1000 '!' video/x-raw,
width=3840, height=2160, format=NV12, framerate=60/1 '!' omxh264enc control-
rate=low-latency target-bitrate=20000 filler-data=0 prefetch-buffer=TRUE
num-slices=16 '!' video/x-h264, alignment=nal '!' queue max-size-buffers=0
'!' omxh264dec low-latency=1 '!' queue max-size-bytes=0 '!' fpsdisplaysink
name=fpssink text-overlay=false 'video-sink=kmssink bus-id=a0070000.v_mix
plane-id=32 hold-extra-sample=1 show-preroll=false sync=true' sync=true -v
vi /run/instantaneous_latency_serial_4k_avc.txt
ement-reported-latency, element-id=(string)%s, element=(string)%s,
live=(boolean)%i, min=(guint64)%lu, max=(guint64)%lu, ts=(guint64)%lu;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)621055684,
ts=(guint64)1162727046

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)477090690,
ts=(guint64)1194432314

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)460875449,
ts=(guint64)1194776858

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)30808839,
ts=(guint64)1678839645;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)25715240,
ts=(guint64)1693021362;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)11618481,
ts=(guint64)1709828871;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)9285958,
ts=(guint64)1726560813;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)9199919,
ts=(guint64)1743164022;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)9160456,
ts=(guint64)1759786273;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
```

```
element=(string)fpssink, sink=(string)sink, time=(guint64)9220741,
ts=(guint64)1776525395;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)9283527,
ts=(guint64)1793272278;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)9489308,
ts=(guint64)1810173597;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)9632312,
ts=(guint64)1826959224;

src0, src=(string)src, sink-element-id=(string)0x559373a200, sink-
element=(string)fpssink, sink=(string)sink, time=(guint64)9166446,
ts=(guint64)1843155083;
```

The initial few readings might go high due to the initialization time, and after that the latency becomes stable, as shown in the previous snapshot where it stabilizes to ~9.4 ms.

The latencies should only be measured accurately when pipeline synchronization is set as enabled (sync=true). The instantaneous latencies might go high (up to one frame period) because GStreamer has an extra margin of buffer duration (except Xilinx low-latency mode). So, if the instantaneous latency is under the reported latency + buffer duration, it does not drop frames.

Total latency = MAX (pipeline reported latency (16 ms), AVG (instantaneous latency) (9.4 ms))

For the preceding use case, the total latency is 16 ms.

Measuring Latencies with Streaming Pipeline

- **Gstreamer Stream-out Pipeline:**

```
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
```

Check for the time (in nanosecond for latency) marked in **bold** in the following logs. The initial few readings may go high due to initialization time and after that it will become stable. 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;
```

```
0:00:21.650023739 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)12221661, ts=(guint64)21649971634;
12017981, ts=(guint64)21666434929;
```

- **Gstreamer Stream-in Pipeline:**

```
GST_DEBUG="GST_TRACER:7" GST_TRACERS="latency" GST_DEBUG_FILE=/run/
client.txt gst-launch-1.0 udpsrc port=5004 buffer-size=60000000
caps="application/x-rtp, media=video, clock-rate=90000, payload=96,
encoding-name=H265" ! queue ! rtph265depay ! h265parse ! video/x-h265,
alignment=nal ! omxh265dec low-latency=1 ! video/x-raw ! queue max-size-
bytes=0 ! fpsdisplaysink name=fpssink text-overlay=false video-
sink="kmssink bus-id=a0070000.v_mix plane-id=30" sync=true
```

Check for the time (in nanosecond for latency) marked in **bold** in the following logs. The initial few readings may go high due to initialization time and after that it will become stable. For example, the following logs shows ~17 ms of latency for stream-in pipeline.

```
0:00:11.979430881 6989  0x557ba91000 TRACE      GST_TRACER :0:: latency,
src-element-id=(string)0x557bb83480, src-element=(string)udpsrc0,
src=(string)src,
sink-element-id=(string)0x557bbce130, sink-element=(string)fpssink,
sink=(string)sink, time=(guint64)16888957, ts=(guint64)11979242832;
15518371, ts=(guint64)12011235439;
15872096, ts=(guint64)12028272281;
```

Checking Reported Latencies of Individual Elements of Pipeline

Modify as below:

To check the reported latencies of each element, run the following command:

```
GST_DEBUG=*v4l2*:6,*omx*:6,*base*:6 GST_DEBUG_FILE="/run/latency.txt”
gst-launch-1.0 <pipeline>
grep -inr "latency” /run/latency.txt | grep v4l2
grep -inr “latency” /run/latency.txt | grep omx
```

This should show the latency reported by v4l2src, encoder and decoder. For example, for the following use-case:

```
GST_DEBUG="*v4l2*:6,*omx*:6,*base*:6" GST_DEBUG_FILE="/run/
serial_4k_avc.txt" gst-launch-1.0 -v v4l2src io-mode=dmabuf num-buffers=100
device=/dev/video0 ! video/x-raw, width=3840, height=2160, format=NV12,
framerate=60/1 ! omxh264enc control-rate=low-latency target-bitrate=25000
filler-data=0 prefetch-buffer=TRUE num-slices=8 periodicity-idr=240 cpb-
size=500 gdr-mode=horizontal initial-delay=250 gop-mode=low-delay-p !
video/x-h264, alignment=nal ! queue max-size-buffers=0 ! omxh264dec low-
latency=1 ! queue max-size-bytes=0 ! fpsdisplaysink name=fpssink text-
overlay=false 'video-sink=kmssink bus-id=a0070000.v_mix plane-id=33 hold-
extra-sample=1 show-preroll=false sync=true fullscreen-overlay=1' sync=true
-v
```

Checking v4l2 reported latency:

```
grep -inr "report latency" serial_avc.txt | grep "v4l2"
```

```
0:00:00.895191805 15284   0x7f98004850 DEBUG                 v4l2src
gstv4l2src.c:779:gst_v4l2src_query: report latency min 0:00:00.016666666
max 0:00:00.533333312
```

According to the above log, the v4l2src reported latency is 16.66 ms.

Checking OMX encoder reported latency:

```
grep -inr "latency" serial_avc.txt | grep "omxh264enc"
```

```
0:00:00.349687650 15284   0x556fe762d0 DEBUG  omxvideoenc
gstomxvideoenc.c:2461:gst_omx_video_enc_set_latency:h264enc0> retrieved
latency of 10 ms
```

According to the above log, the encoder reported latency is 10 ms.

Checking OMX decoder reported latency:

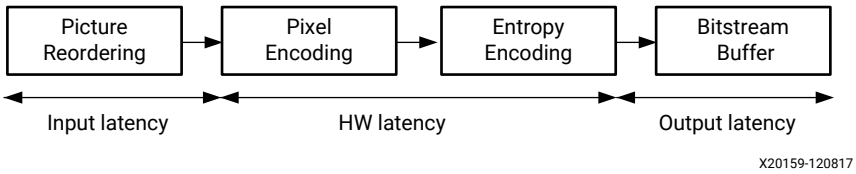
```
grep -inr "latency" serial_avc.txt | grep "omxh264dec"
```

```
0:00:00.459566429 15284   0x556ff60000 DEBUG  omxvideodec
gstomxvideodec.c:2483:gst_omx_video_dec_set_latency:h264dec0> retrieved
latency of 17 ms
```

VCU Encoder Latency

The following figure shows the encoder latency.

Figure 41: **Encoder Latency**

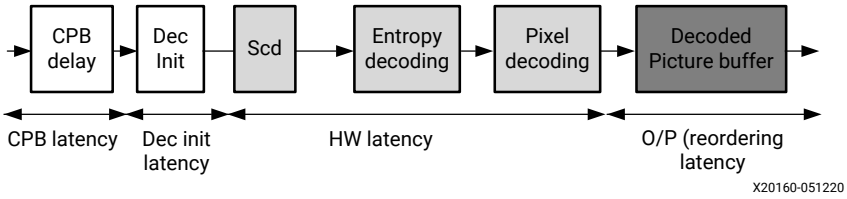


The overall latency of the encoder is the steady state latency which is equal to the sum of the input latency, the hardware latency, and the output latency. The bitstream buffer latency is application dependent. The picture reordering latency equals one frame duration per B-frame.

VCU Decoder Latency

The following figure shows the decoder latency.

Figure 42: **Decoder Latency**



The overall latency of the decoder is the steady state latency, equal to the sum of the hardware latency and the output latency. Hardware latency is the sum of the successive cancellation decoding (SCD) latency, the entropy decoding latency, and the pixel decoding latency. Initialization latency is the sum of the CPB latency and the Decoder Initialization (Dec Init) latency.

Chapter 21

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

TIP: *If the IP generation halts with an error, there might be a license issue.*

Finding Help on Xilinx.com

To help in the design and debug process when using the core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. The Xilinx Community Forums are also available where members can learn, participate, share, and ask questions about Xilinx solutions.

Documentation

This product guide is the main document associated with the core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx® Documentation Navigator. Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use keywords such as:

- Product name
- Tool message(s)

- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the Core

AR 76600.

Technical Support

Xilinx provides technical support on the Xilinx Community Forums for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To ask questions, navigate to the Xilinx Community Forums.

Debug Tools

There are many tools available to address H.264/H.265 Video Codec Unit (VCU) design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx® devices.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) for more information.

Reference Boards

Various Xilinx development boards support the VCU. These boards can be used to prototype designs and establish that the core can communicate with the system.

- ZCU106

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the debug feature for debugging the specific problems.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design (if applicable) and that all constraints were met during implementation.

- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `locked` port.
- If your outputs go to 0, check your licensing.

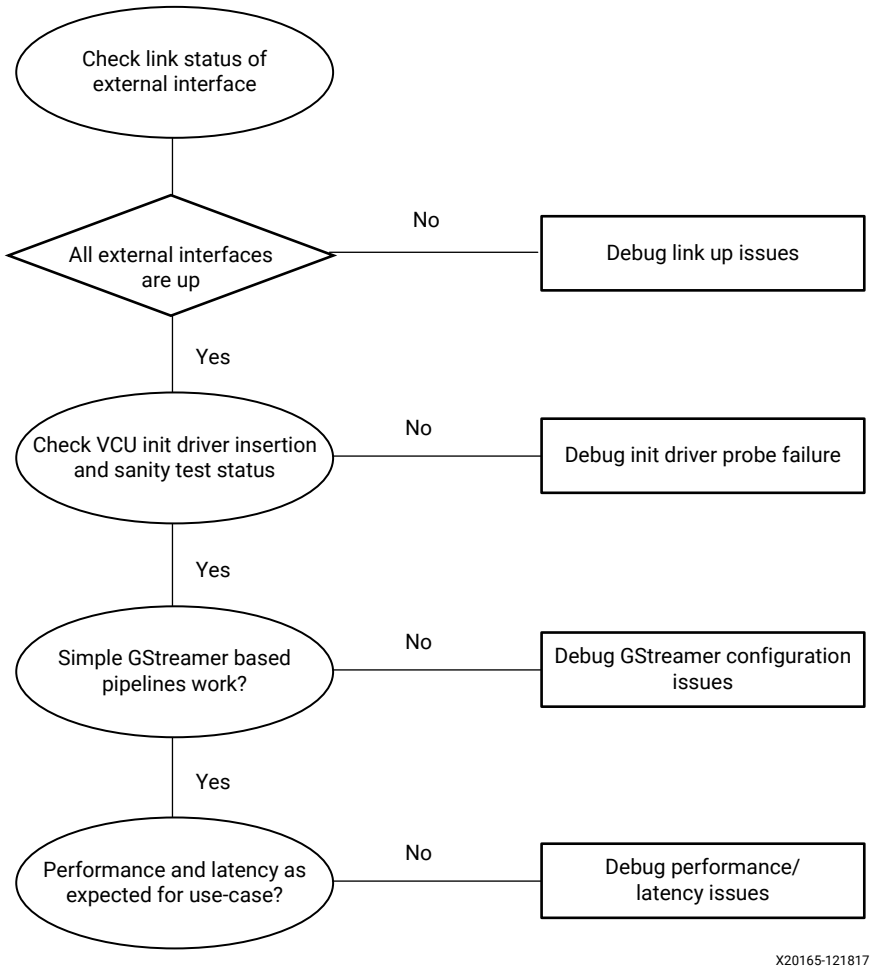
Debugging a VCU-based System

Troubleshooting a VCU based system can be complicated. This appendix offers a decision tree to guide you efficiently to the most productive areas to investigate. The troubleshoot steps apply to a VCU based system capable of performing live capture, encode, decode, transport, and display.

Debug Flow

The following figure shows the system level debug flow.

Figure 43: **Debug Flow**



Troubleshooting

Debugging External Interfaces

If you are using capture and display interfaces based on high-speed serial I/O (eg. HDMI, MIPI, SDI etc), ensure physical links are up before debugging the upper layers. Please follow these steps:

1. Monitor for LEDs that indicate heart beat clocks that are derived based on reference clock input to transceiver. If the heart beat clocks are absent, check for GT PLL lock status.
2. If PLL is not locked, check for Video PHY IP configuration and reference clock constraint. Most of the time, the reference clock is sourced from a programmable clock chip. Ensure the frequency is programmed properly in the device tree source file for the programmable clock chip and the frequency is not modified by other Linux devices (happens when the `phandle` of the clock source node is shared between multiple components).
3. If the capture interface is compatible with the Video for Linux (v4l2) framework, use the media-ctl API to verify the link topology and link status.

   ```
   media-ctl -p -d /dev/mediaX
   ```

   The mediaX represents the pipeline device in the v4l2 pipeline. If you have multiple TPG/HDMI pipelines, they appear as `/dev/media0`, `/dev/media1`, etc. The link status is indicated in the corresponding sub-device node. For example, a HDMI RXSS node indicates link status for the HDMI link in its sub-device properties while using the above command. If link up fails, a "no-link" message appears. Otherwise, valid resolution with proper color format is detected.
4. If the display interface is compatible with DRM/KMS framework, please ensure DRM is linking up by running the one of following commands:
   - If the PL mixer block is used, use `modetest -M xilinx_drm_mixer`.
   - If the PL mixer block is not used, use `modetest -M xilinx_drm`.

   If you want to display a pattern to ensure the display path is working, use one of the following commands:

   - To display using the BG24 format, use `modetest -M xilinx_drm_mixer -s <connector_id>@<crtc_id>:3840x2160-60@BG24`.
   - To display using the NV12 format, use `modetest -M xilinx_drm_mixer -P <crtc_id>:3840x2160-60@NV12`.

   Run the `modetest` command multiple times to ensure a stable link at different resolution before proceeding with different VCU use-cases.

5. If the capture pipeline (v4l2 based) or display pipeline (DRM/KMS based) are broken, then `media-ctl` or `modetest` applications show a broken pipeline and sub-devices are not being linked properly. If so, browse through the boot log to see if there is a probe failure for any of the v4l2 sub-device drivers. If the probe fails, check your dts file for any property name mismatch or missing/incorrect property.

Debugging the VCU Interface

To debug the VCU interface, perform the following steps:

1. Ensure VCU init driver probe is successful during boot process. In the boot log, you can see the following messages

```
xilinx-vcu <axilite address>.vcu: xvcu_probe: Probed successfully
```

2. If PLL fails to lock, VCU initialization driver reports a lock status failure message. Ensure that the PLL input reference clock frequency of VCU LogiCORE™ IP is set properly in the IPI block design. Ensure that the PLL clock source (parent clock) is modelled correctly in device tree node as a fixed clock source. Ensure VCU init driver node properties are proper and `phandle` for PLL reference clock is passed correctly.
3. After successful probe of VCU init driver, check VCU drivers with the following command:

```
lsmod
```

   which should show al5d, al5e and Allegro modules as being inserted.

4. Check if sufficient CMA is allocated. VCU operation requires at least 1000 MB of CMA. You can check for available CMA by using

```
cat /proc/meminfo
```

5. If CMA size is not sufficient, increase the size either in U-Boot using the following steps:
   a. Stop the boot process at U-Boot level and run the following command to increase the CMA size or while building the Linux kernel using kernel configuration property.

```
setenv bootargs ${bootargs} cma=xxxxM
```

   b. In the kernel config (for PetaLinux, run `petalinux-config -c kernel` and for standalone compilation, run make `menuconfig`), change the size of the CMA by selecting **Library Routines → DMA Contiguous Memory Allocator → Size in MegaBytes**.
6. Run a VCU sanity test using on of the VCU Control Software sample applications. You can use `ctrlsw_encoder` and `ctrlsw_decoder` applications to run a sanity test.

Debugging Control Software Application

To debug control software based application (`ctrlsw_encoder`, `ctrlsw_decoder`), perform the following steps.

1. Run file to file-based encoder and decoder sample applications to ensure they operate as expected.
   - H.264 decoding file-to-file

     ```
     ctrlsw_decoder -avc -in input-avc-file.h264 -out ouput.yuv
     ```

   - H.265 decoding file-to-file

     ```
     ctrlsw_decoder -hevc -in input-hevc-file.h265 -out ouput.yuv
     ```

   - Encoding file-to-file

     ```
     ctrlsw_encoder -cfg encode_simple.cfg
     ```

   The application exits with appropriate error message. Refer error description table for more details. For example, if the VCU power is insufficient to encode 4kp resolution file at 120fps, the following message appears:

   ```
   ctrlsw_encoder -cfg AVC_test.cfg -i test_4Kp120_nv12.yuv -o /dev/null
   Allegro DVT2 - AVC/HEVC Encoder Reference Software v1.0.41 - Copyright
   (C) 2018 Confidential material
   Channel creation failed, processing power of the available cores
   insufficient
   Codec error: Channel creation failed, processing power of the available
   cores insufficient
   To debug above error, change either Frame rate(4kp60) or
   resolution(1080p120) in cfg file which can be handled by VCU and re-run.
   ```

   Application exit with memory error if CMA is insufficient. For example:

   ```
   ctrlsw_encoder -cfg AVC_test.cfg -i test_4Kp60_nv12.yuv -o /dev/null
   Allegro DVT2 - AVC/HEVC Encoder Reference Software v1.0.41 - Copyright
   (C) 2018
   Confidential material
   [53.119829] cma: cma_alloc: alloc failed, req-size: 3078 pages, ret: -12
   [53.126542] al5e a0100000.al5e: Can't alloc DMA buffer of size 12607488
   GET_DMA_FD: Cannot allocate memory
   [53.137916] cma: cma_alloc: alloc failed, req-size: 10522 pages, ret: -12
   [53.144712] al5e a0100000.al5e: Failed internal buffers allocation,
   channel wasn't
   created
   Memory shortage detected (DMA, embedded memory or virtual memory)
   Codec error: Memory shortage detected (DMA, embedded memory or virtual
   memory)
   ```

   To mitigate the memory error, check that CMATotal and CMAFree are sufficient using the following command:

   ```
   cat /proc/meminfo
   ```

   Increase CMA size either in U-Boot using a command, `setenv bootargs ${bootargs} cma=xxxxM` or, while building the Linux kernel using kernel configuration property.

2. If the output file is not generated and no failure/error message occurs on the terminal, check for kernel level message using a command:

   ```
   dmesg
   ```

Check "al5e", “al5d” keyword in the dmesg log for the error, if any.

3. If the application freezes, debug using “gdb”. For that, remove optimization from control software Makefile as below.

```
replace: CFLAGS+=-O3
By CFLAGS+=-O0
```

Then run the application using the following command:

```
gdb -args ctrlsw_encoder -cfg AVC_test.cfg -i test_4Kp60_nv12.yuv -
o /dev/null
```

which provides the gdb shell. Type “run” to execute the application

```
(gdb)run
```

When it hangs, use the backtrace function to view the last function flow from where it hangs.

```
(gdb)bt
```

Debugging GStreamer Based Application

To debug a GStreamer based application, perform the following steps.

1. Run capture to display pipeline to ensure live capture to display is working as expected.

```
gst-launch-1.0 -v v4l2src io-mode=4 device=/dev/video1 ! video/
x-raw,format=NV12,width=3840,height=2160, framerate=30/1 ! kmssink
driver-name=xilinx_drm_mixer
```

2. If you see a streamon error with the capture → display pipeline, ensure that the format is set to NV12 using v4l2-ctl and media-ctl API inside hdmi configuration script. Here is an example

```
v4l2-ctl -d /dev/video1 --set-fmt-
video=width=3840,height=2160,pixelformat='NV12'
media-ctl -d /dev/media1 -V "\"a0080000.scaler\":0
[fmt:RBG888_1X24/3840x2160 field:none]"
media-ctl -d /dev/media1 -V "\"a0080000.scaler\":1
[fmt:VYYUYY8_1X24/3840x2160 field:none]"
```

3. Run a pipeline with VCU components in the pipeline. Use omxh264/omxh265enc/dec components in the pipeline.
4. For a list of supported properties of omxh264/omxh265enc/dec, use the following command:

```
gst-inspect-1.0 <omxh264/omxh265enc/dec>
```

which lists all the supported properties of VCU encoder and decoder blocks. Use the properties as appropriate in the pipeline.

5. Start with a known pipeline example like the one below:

```
gst-launch-1.0 -v \
v4l2src num-buffers=2100 device=/dev/video8 io-mode=4 \
! video/x-raw,format=NV12,width=3840,height=2160, framerate=30/1 \
! omxh265enc target-bitrate=70000 prefetch-buffer=TRUE \control-rate=2
gop-length=30 b-frames=0 \
! video/x-h265, profile=main,level=\
(string\
)5.1,tier=main \
! omxh265dec low-latency=0 internal-entropy-buffers=2 \
! queue \
! fpsdisplaysink name=fpssink text-overlay=false \
video-sink="kmssink max-lateness=100000000 async=false \
sync=true driver-name=xilinx_drm_mixer" -v
```

Debugging Performance Issues

For additional debugging information, see the Debugging Tools page of GStreamer website at https://gstreamer.freedesktop.org/documentation/tutorials/basic/debugging-tools.html.

If the problem is low frame rate or frame dropping, follow these steps to debug the system.

1. Use the `fpsdisplaysink` element to report frame rate and dropped frame count (refer to the example above)
2. Check the QoS settings of HP ports that interface VCU with PS DDR. Check for outstanding transaction count configuration. Note that for VCU traffic, the QoS should be set as Best Effort (BE) and outstanding transaction count should be set to maximum (0xF for AFI ports).
3. Check if SMMU is enabled in the device tree. If SMMU is enabled, disable it because it imparts longer latency in the transaction completion.
4. Check if encoder buffer (prefetch-buffer) is used in the user design. If not, check if encoder buffer impacts the performance. If performance improves with encoder buffer, it might indicate a system bandwidth issue. Use the following sample pipeline to enable the prefetch buffer in design:

```
gst-launch-1.0 videotestsrc ! omxh265enc prefetch-buffer=true ! fakesink
```

5. Try with different encoder/decoder properties to see if the performance drop is related to any of the properties. Avoid B-frames in the pipeline to see if there is any performance improvement. If there is improvement, it might indicate a system bandwidth issue. Reduce the bit rate to see if there is improvement in framer ate. If reducing target-bit rate gives better throughput, it might indicate a system bandwidth issue.
6. Check for CPU usage while the pipeline is running. A higher CPU usage indicates that there could be an impact in interrupt processing time which explains the lower framerate.
7. Try using a `queue` element between two GStreamer plugins that are in the datapath to check for any performance improvement.
8. Check for DDR bandwidth utilization using DDR APM and VCU APM.

9. Use gst-shark (a GStreamer-based tool) to verify performance and create scheduletime and interlatency plots to understand which element is causing performance drops.
10. Using environment variables, you can increase encoder input and output buffers count for debug or performance tuning purpose. Use the ENC_EXTRA_IP_BUFFERS and ENC_EXTRA_OP_BUFFERS environment variable to provide extra buffers needed on encoder ports. For example, suppose X number of buffers are allocated for encoder input/output by default. To add five more buffers to it, assign `ENC_EXTRA_IP_BUFFERS=5`. So, new allocated buffers for encoder input is X+5. Similarly, for encoder output buffers, use ENC_EXTRA_OP_BUFFERS. The pipelines are as follows:

```
ENC_EXTRA_IP_BUFFERS=5 gst-launch-1.0 -v v4l2src io-mode=4 device=/dev/
video0 num-buffers=1000 ! video/x-raw, format=NV12, width=3840,
height=2160, framerate=60/1 ! omxh264enc control-rate=constant target-
bitrate=50000 prefetch-buffer=TRUE ! video/x-h264, profile=high !
filesink location=test.avc
```

```
ENC_EXTRA_OP_BUFFERS=5 gst-launch-1.0 -v v4l2src io-mode=4 device=/dev/
video0 num-buffers=1000 ! video/x-raw, format=NV12, width=3840,
height=2160, framerate=60/1 ! omxh264enc control-rate=constant target-
bitrate=50000 prefetch-buffer=TRUE ! video/x-h264, profile=high !
filesink location=test.avc
```

Debugging Latency Issues

If the problem is high end-to-end latency, follow these steps to debug the system:

1. Understand how much the jitter buffer is used in the client side pipeline (udpsrc). Please note that the latency for the `rtpjitterbuffer` should correspond to the CPB size in the server pipeline. Often it is useful to use LowLatency rate control (or hardware rate control) algorithm to maintain a lower CPB buffer that reduces the `rtpjitterbuffer` latency.
2. Many times, latency is related to frame drop. Ensure there is no frame drop in the pipeline before measuring latency.
3. Check if use of the filler-data setting in the encoder pipeline is causing longer latency. If so, set filler-data=false in the pipeline and check for latency
4. Use a fine-tuned internal-entropy-buffers count. Note that internal-entropy-buffers setting impacts the latency and an optimal value needs to be used. You can update this decoder property to ensure no frame drop first and later to optimize the pipeline latency.

Interface Debug

AXI4-Lite Interfaces

To verify that the interface is functional, try reading from a register that does not have all 0s as its default value. Output `s_axi_arready` asserts when the read address is valid, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `s_axi_aclk` and `aclk` inputs are connected and toggling.
- The interface is not being held in reset, and `s_axi_areset` is an active-Low reset.
- The interface is enabled, and `s_axi_aclken` is active-High (if used).
- The main core clocks are toggling and that the enables are also asserted.

AXI4-Stream Interfaces

If data is not being transmitted or received, check the following conditions:

- If transmit `<interface_name>_tready` is stuck Low following the `<interface_name>_tvalid` input being asserted, the core cannot send data.
- If the receive `<interface_name>_tvalid` is stuck Low, the core is not receiving data.
- Check that the `aclk` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed.
- Check core configuration.

XILINX

Section V

Application Software Development

Overview

The Video Codec Unit (VCU) software stack has a layered architecture programmable at several levels of abstraction available to software developers, as shown in the following figure. The application interfaces from high level to low level are:

- GStreamer
- OpenMAX Integration Layer
- VCU Control Software

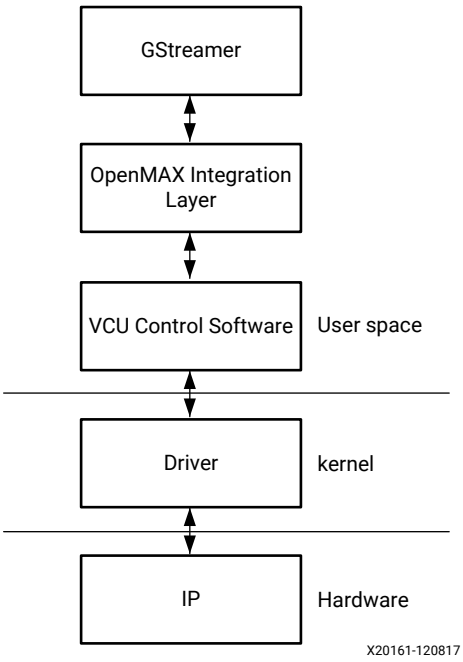
The GStreamer is a cross-platform open source multimedia framework. GStreamer provides the infrastructure to integrate multiple multimedia components and create pipelines. The GStreamer framework is implemented on the OpenMAX Integration Layer API-supported GStreamer version is 1.14.4.

The OpenMAX Integration Layer API defines a royalty-free standardized media component interface to enable developers and platform providers to integrate and communicate with multimedia codecs implemented in hardware or software.

The VCU Control Software is the lowest level software visible to VCU application developers. All VCU applications must use a Xilinx® provided VCU Control Software, directly or indirectly. The VCU Control Software includes custom kernel modules, custom user space library, and the ctrlsw_encoder and ctrlsw_decoder applications. The OpenMAX IL (OMX) layer is integrated on top of the VCU Control Software.

User applications can use the layer or layers of the VCU software stack that are most appropriate to their requirements.

Figure 44: **VCU Software Stack**



Software Prerequisites

All of the software prerequisites for using the VCU are included in Xilinx PetaLinux included in Vitis™ software development platform release. Refer to the Release Notes Links at the bottom of the Embedded Design Hub - PetaLinux Tools or to the Release Notes link on the download page.

For the Xilinx Linux kernel, refer to xilinx_zynqmp_defconfig at `linux-xlnx/arch/arm64/configs/xilinx_zynqmp_defconfig`, where all the Xilinx driver configurations options are enabled.

For the vanilla Linux kernel, refer to xilinx_zynqmp_defconfig to enable and disable a Xilinx driver in the Linux kernel. If the design enables or disables the Xilinx IP, the corresponding device-tree node should be set to enable the driver to probe at run time kernel.

The application software using the VCU is written on top of the following libraries and modules, shown in the following table.

Table 71: **Application Software**

Software	Version	Source
Gstreamer core library and plugins	1.16.3	https://github.com/Xilinx/gst-plugins-good/tree/xlnx-rebase-v1.16.3 https://github.com/Xilinx/gst-plugins-base/tree/xlnx-rebase-v1.16.3 https://github.com/Xilinx/gst-plugins-bad/tree/xlnx-rebase-v1.16.3 https://github.com/Xilinx/gstreamer/tree/xlnx-rebase-v1.16.3 https://github.com/Xilinx/gst-omx/tree/xlnx-rebase-v1.16.3
OpenMAX Integration Layer API	1.1.2	https://github.com/Xilinx/vcu-omx-il/tree/xlnx_rel_v2021.2
VCU Control Software	Library version: 0.25.0 Application version: 1.0.61	https://github.com/Xilinx/vcu-ctrl-sw/tree/xlnx_rel_v2021.2
VCU firmware	1.0.0	https://github.com/Xilinx/vcu-firmware/tree/xlnx_rel_v2021.2
VCU kernel modules		https://github.com/Xilinx/vcu-modules/tree/xlnx_rel_v2021.2
VCU recipe files		https://github.com/Xilinx/meta-xilinx/tree/rel-v2021.2/meta-xilinx-bsp/recipes-multimedia/vcu
Gstreamer recipe files		Core recipes: https://github.com/Xilinx/poky/tree/rel-v2021.2/meta/recipes-multimedia/gstreamer Recipe bbappend files: https://github.com/Xilinx/meta-petalinux/tree/rel-v2021.2/recipes-multimedia/gstreamer

XILINX

Chapter 23

Encoder and Decoder Software Features

Encoder Software Features

The VCU supports multi-standard video encoding, shown in the following table.

Table 72: **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 and Frame Rate[1, 2]	4096x2160p60 with specific device (-2, -3) Up to 3840x2160p60	4096x2160p60 with specific device (-2,-3) Up to 3840x2160p60
Bit Depth		
GStreamer	8-bit, 10-bit	8-bit, 10-bit
OMX	8-bit, 10-bit	8-bit, 10-bit
VCU Control Software	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
VCU Control Software	4:2:0, 4:2:2	4:2:0, 4:2:2
Slices Types	I, P, and B	I, P, and B

Table 72: **Encoder Features** *(cont'd)*

Video Coding Parameter	H.265 (HEVC)	H.264 (AVC)
Bit Rate	Limited by level and profile	Limited by level and profile

Notes:

1. The H.265 (HEVC) minimum picture resolution is 128×128.
2. The H.264 (AVC) minimum picture resolution is 80×96.

GStreamer Encoding Parameters

The GStreamer encoding parameters are shown in the following table.

Table 73: **GStreamer Encoding Parameters**

VCU Parameter	GStreamer Property	Description
Rate Control Mode	control rate	Available bit rate control modes, Constant Bit rate (CBR) and Variable Bit rate (VBR) Encoding 0 = Disable (CONST_QP, Constant QP) 1 = Variable (VBR) 2 = Constant (CBR) 3 = Variable skip frames 4 = Constant skip frames 2130706433=low-latency (Hardware rate control, used in slice level encoding) 2130706434=capped-variable (CVBR) Default value: 2 ***Note*:** Variable Skip Frames and Constant Skip Frames are coming from gstreamer default plugin, not supported in VCU.[1, 2, 3, 4, 5]
Target Bit Rate[6]	target bitrate	Target bitrate in Kbps Default value: 64
Maximum Bit Rate[6]	max bitrate	Maximum bitrate, in Kbps, used in VBR rate control mode. Default value: target-bitrate. The max-bitrate value should always be the target-bitrate Default value: 64
Profile	Through caps	Supported profiles for corresponding codec are mentioned in Table 72: Encoder Features Default value: HEVC_MAIN
Level	Through caps	Supported Levels for corresponding codec are mentioned in Table 72: Encoder Features Default value: 51
Tier	Through caps	Supported Tier for H.265 (HEVC) codec is mentioned in Table 72: Encoder Features Default value: MAIN_TIER
Slice QP / I-frame QP	quant-i-frames	Quantization parameter for I-frames in CONST_QP mode, also used as initial QP in other rate control modes. Range 0–51. Default value: 30

Table 73: **GStreamer Encoding Parameters** *(cont'd)*

VCU Parameter	GStreamer Property	Description
P-frame QP	quant-p-frames	Quantization parameter for P-frames in CONST_QP mode. Range 0–51. Default value: 30
B-frame QP	quant-b-frames	Quantization parameter for B-frames in CONST_QP mode. Range 0–51. Default value: 30
GOP Length	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	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 is set. Default value: 0
Number of Slices	num-slices	Specifies the number of slices used for each frame. Each slice contains one or more full LCU row or rows and are spread over the frame as regularly as possible. The minimum value is 1. Max supported value as below: In low-latency mode H.264(AVC): 32 H.265 (HEVC): 22 More than 16 slices will not bring much benefit in low latency mode because of overhead In normal latency-mode H.264(AVC): picture_height/16 H.265(HEVC): minimum of picture_height/32 Above max supported values in normal mode will work but HEVC has some more limitations when encoder uses multiple cores When HEVC encoder uses multiple cores (i.e. anything beyond 1080p60 resolution) then the max num-slices is limited by "Max # of tile rows MaxTileRows". Refer to Table 74: Tier and Level Requirements. In 4k encoding, num-slices should be set to 8 for all latency-modes to achieve best performance. Default value: 1
Minimum QP	min-qp	Minimum QP value allowed in encoding session. Range 0–51. Default value: 10
Maximum QP	max-qp	Maximum QP value allowed in encoding session. Range 0–51. Default value: 51

Table 73: **GStreamer Encoding Parameters** *(cont'd)*

VCU Parameter	GStreamer Property	Description
GOP Configuration	gop-mode	Specifies group of pictures configuration 0 = basic (IPPP..IPPP) 1 = basic-b (basic GOP settings, includes only B-frames) 2 = pyramidal (advanced GOP pattern with hierarchical B frame, works with B=3, 5, 7 and 15) 3 = pyramidal-b (advanced GOP pattern with hierarchical B-frames, includes only B-frames) 4 = adaptive (advanced GOP pattern with adaptive B-frames) 5 = low-delay-p (IPPPPP) 6 = low-delay-b (IBBBBB) Both open and closed GOP structures are supported. When Gop.FreqIDR ≠ Gop.Length and B-frames ≠ 0, open GOP structures are possible. For pyramidal gop-mode, encoder uses closed GOPs only. Without B-frames, it only uses closed GOPs. With B-frames, gop-mode=pyramidal, it uses closed GOPs. With B-frames, gop-mode=default and periodicity-idr = 0, it uses open GOPs. With B-frames, gop-mode=default and IDR_freq= GOP. Length, it uses closed GOPs. (For different IDR_freq, you can get mix of open and closed GOPs) Default value: 0
Gradual Decoder Refresh	gdr-mode	Specifies which Gradual Decoder Refresh scheme should be used when gop-mode = low_delay_p 0 = disable 1 = vertical (Gradual refresh using a vertical bar moving from left to right) 2 = horizontal (Gradual refresh using a horizontal bar moving from top to bottom) Default value: 0
QP Control Mode	qp-mode	QP control mode used by the VCU encoder 0 = uniform (Use the one QP for all coding units of the frame) 1 = roi (Adjust QP according to the regions of interest defined on each frame. ROI metadata must be supplied) 2 = auto (Let the VCU encoder change the QP for each coding unit according to its content) Default value: 2
Filler data	filler-data	Enable/Disable filler data adding functionality in CBR rate control mode. Boolean: TRUE or FALSE Default value: True

Table 73: **GStreamer Encoding Parameters** *(cont'd)*

VCU Parameter	GStreamer Property	Description
Entropy Mode	entropy-mode	Specifies the entropy mode for H.264 (AVC) encoding process 0 = CAVLC 1 = CABAC Default value: 1
Deblocking Filter	loop-filter-mode	Enables/disables the deblocking filter option 0 = enable 1 = disable 2 = disable-slice-boundary (Excludes slice boundaries from filtering) Default value: 0
IDR picture frequency	periodicity-idr	Specifies the number of frames between consecutive instantaneous decoder refresh (IDR) pictures. The periodicity-idr property was formerly called gop-freq-idr. Allowed values: <Positive value> or -1 to disable IDR insertion Default value: 0 (first frame is IDR)
Initial Removal Delay	initial-delay	Specifies the initial removal delay as specified in the HRD model in milliseconds. Not used when control-rate = disable ***Note***: If this value is set too low (less than 1 frame period), you may see reduced visual quality. Default value: 1500
Coded Picture buffer size	cpb-size	Specifies the coded picture buffer (CPB) as specified in the HRD model in milliseconds. Not used when control-rate = disable Default value: 3000
Dependent slice	dependent-slice	Specifies whether the additional slices are dependent on other slice segments or regular slices in multiple slices encoding sessions. Used in H.265 (HEVC) encoding only. Boolean: TRUE or FALSE Default value: False
Target slice size	slice-size	If set to 0, slices are defined by the num-slices parameter, else it specifies the target slice size in bytes. Used to automatically split the bitstream into approximately equally-sized slices. Range 0–65,535. Default value: 0
Scaling Matrix	scaling-list	Specifies the scaling list mode 0 = flat 1 = default Default value: 1
PrefetchLevel2	prefetch-buffer	Enable/Disable L2Cache buffer in encoding process Default value: false
Vertical Search Range	low-bandwidth	Specifies low bandwidth mode. Decreases the vertical search range used for P-frame motion estimation. TRUE or FALSE. Default values: FALSE

Table 73: **GStreamer Encoding Parameters** *(cont'd)*

VCU Parameter	GStreamer Property	Description
Slice Height	slice-height	This parameter is used for 2017.4 and earlier releases only. It is deprecated in 2018.3 and later releases. Specifies input buffer height alignment of upstream element, if any. Default value: 1
Aspect-Ratio	aspect-ratio	Selects the display aspect ratio of the video sequence to be written in SPS/VUI. 0 = auto - 4:3 for SD video,16:9 for HD video, unspecified for unknown format 1 = aspect_ratio_4_3 (4:3 aspect ratio) 2 = aspect_ratio_16_9 (16:9 aspect ratio) 3 = none (Aspect ratio information is not present in the stream) Default value: 0
Constrained Intra Prediction	constrained-intra-prediction	If enabled, prediction only uses residual data and decoded samples from neighboring coding blocks that are coded using intra prediction modes. Boolean: TRUE or FALSE. Default value: False
Long term Ref picture	long-term-ref	If enabled, encoder accepts dynamically inserting and using long-term reference picture events from upstream elements Boolean: TRUE or FALSE Default value: False
Long term picture frequency	long-term-freq	Periodicity of Long-term reference picture marking in encoding process Units in frames, distance between two consecutive long-term reference pictures Default value: 0
Dual pass encoding	look-ahead	The number of frames processed ahead of second pass encoding. If smaller than 2, dual pass encoding is disabled. Default value: 0
Alignment	Through caps	Encoder alignment =au (normal mode) =nal (low-latency mode) Default value: au Note: When encoder alignment is set to low-latency mode, it is recommended that you set the rate control mode (control-rate) to low latency. See VCU Latency Modes on how to set alignment. When using Low Latency mode, the encoder and decoder are limited by the number of internal cores. The encoder has a maximum of four streams and the decoder has a maximum of two streams.

Table 73: **GStreamer Encoding Parameters** *(cont'd)*

VCU Parameter	GStreamer Property	Description
Max Quality Target	max-quality-target	Caps quality at certain limit keeping the bit rate variable. Only used with capped variable rate-control. It xcps the quality at a certain high limit, keeping bitrate variable Range: 0 to 20 (20 = lossless quality) Default value: 14 The allowed values are between 0 and 20 (where 0 is really poor quality and 20 is close to visual lossless quality). If the encoder cannot reach this quality level due to the video complexity and/or a restricted bitrate, or If the quality target is too high, then the Capped-VBR has no effect compared to VBR. ***Note***: The controlSW parameter name in cfg is MaxQuality.
Max Picture Size	max-picture-size	You can curtail instantaneous peak in the bit-stream using this parameter. It works in CBR/VBR rate-control only. When Max Picture Size is enabled, the VCU encoder uses CBR/VBR but it also enables the hardware rate control module to keep track of encoding frame size. The hardware rate control module adjusts the QPs within the frame to ensure that the encoded picture sizes honor the provided max-picture-size. MaxPictureSize = TargetBitrate / FrameRate * AllowedPeakMargin For 100 Mbps TargetBitrate, 60 fps FrameRate, and 10% AllowedPeakMargin, MaxPictureSize = (100 Mb/s / 60 fps) * 1.1 = 1834 Kb per frame.
Slice Type Value Selection	uniform-slice-type	Enable/Disable uniform slice type in slice header. When this is enabled, the following slice-type values are used in slice header. • I slice: slice-type = 7 is used • P slice: slice-type=5 is used • B slice: slice-type=6 is used Supported from 2021.1 release onwards.
Input yuv Crop Feature	input-crop	The `input-crop` parameter sets `<pos-x, pos-y, cropWidth, cropHeight>` values for the VCU encoder. The property type of `max-picture-sizes` is `GstValueArray`. This parameter is supported only from release 2021.1 onwards.
Latency	latency-mode	For 2018.3 and prior releases (Not used in 2019.1 or later releases; for 2019.1 and future releases, use the Alignment parameter), specifies encoder latency modes: 0 = normal (provides frame level output to the next element) 1 = low-latency (provides slice level output to the next element) Default value: 0 ***Note***: The low-latency latency mode is not recommended when Rate Control is CBR/VBR.

Table 73: **GStreamer Encoding Parameters** *(cont'd)*

VCU Parameter	GStreamer Property	Description
Default ROI Quality	default-roi-quality	Default quality level to apply to each Region of Interest 0=high – Delta QP of -5 1=medium – Delta QP of 0 2=low – Delta QP of +5 3=don't-care – Maximum delta QP value Default: 0
Frame Skip	skip-frame	If enabled and encoded, picture exceeds the CPB buffer size; the specific picture is discarded and replaced by a picture with all MB/CTB encoded as skip. Only use of control-rate=constant/variable and b-frames are less than 2. Default: FALSE
Maximum number of consecutive frame skips	max-consecutive-skip	Specifies the maximum number of consecutive skipped frames if skip-frame is enabled. Default Value: 4294967295
Max picture size for frame types	max-picture-sizes	Max picture sizes based on frame types ('<I, P, B>') Maximum picture size of I, P, and B frames in Kb, encoded picture size is limited to max-picture-size-x value. If set it to 0, the max-picture-size-x does not have any effect. GstValueArray of GValue of type "gint" Write only. The property type of max-picture-sizes is GstValueArray.
Intraframes interval	interval-intraframes	Interval of coding Intra frames Default: 0
Beta offset for deblocking filters	loop-filter-beta-offset	Beta offset for the deblocking filter is used only when loop-filter-mode is enabled. Range: -6 to 6 Default: -1

Table 73: **GStreamer Encoding Parameters** *(cont'd)*

VCU Parameter	GStreamer Property	Description
Alpha offset for deblocking filters	loop-filter-alpha-c0-offset	Alpha C0 offset for the deblocking filter, used only when loop-filter-mode is enabled. Range: -6 to 6 Default: -1

Notes:

1. Rate control is handled in MCU FW only and no signals (either in Software API or FPGA signals) that are triggered during rate control process.
2. 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 "HRD" model, i.e. 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 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.
3. 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 be exceeded up to the maximum bit rate. So, the QP has to be increased by a smaller factor. A buffer overflow results in an unchanged QP and a lower bit rate.
4. Both CBR and VBR use frame-level statistics from the hardware to update the initial QP for the next frame (rate control can be combined with QP control that can adjust the QP at block level for improving subjective quality).
5. LOW_LATENCY rate control (hardware rate control) computes the QP at block level to reach a target bitstream size for each frame in an accurate way, and is especially useful for the support of low-latency pipelines.
6. Xilinx low latency supports 25 Mb/s for single stream of 4K and 6 Mb/s for 4x stream of 1080.
7. The following is an example for the v4l2src encode and file sink using custom max-pictures-sizes:

```
gst-launch-1.0 v4l2src io-mode=4 ! video/x-raw, width=1920, height=1080 format=NV12,
framerate=60/1 ! omxh265enc control-rate=2 target-bitrate=20000 gop-mode=5 qp-mode=auto gop-
length=60 cpb-size=1000 initial-delay=500 max-picture-sizes="<666,333,0>" ! filesink
location="/media/card/output_maxpicture1.hevc"
```

8. The sample pipeline for the `input-crop` feature is as follows:

```
gst-launch-1.0 filesrc location= "test_720x243.nv16" ! videoparse width=720 height=243
format=nv16 framerate=60/1 ! omxh264enc input-crop="<0,1,720,240>" ! filesink
location=test.avc -v
```

Tier and Level Limits

The tier and level limits are shown in the following table.

Table 74: **Tier and Level Requirements**

Level	Max Luma Picture Size	Max CPB Size		Max Slice Segment per Picture	Max # of Tile Rows	Max # of Tile Columns
		Main Tier	High Tier			
1	36,864	350	-	16	1	1
2	122,880	1,500	-	16	1	1
2.1	245,760	3,000	-	20	1	1
3	552,960	6,000	-	30	2	2

Table 74: **Tier and Level Requirements** *(cont'd)*

Level	Max Luma Picture Size	Max CPB Size		Max Slice Segment per Picture	Max # of Tile Rows	Max # of Tile Columns
		Main Tier	High Tier			
3.1	983,040	10,000	-	40	3	3
4	2,228,224	12,000	30,000	75	5	5
4.1	2,228,224	20,000	50,000	75	5	5
5	8,912,896	25,000	100,000	200	11	10
5.1	8,912,896	40,000	160,000	200	11	10
5.2	8,912,896	60,000	240,000	200	11	10
6	35,651,584	60,000	240,000	600	22	20
6.1	35,651,584	120,000	480,000	600	22	20
6.2	35,651,584	240,000	800,000	600	22	20

Max Supported Num Slices for 1080p and 4k Resolution

The following table shows the max supported num-slices for 1080p and 4k resolution in the subframe/normal latency mode.

Table 75: **Max supported num-slices in 1080p and 4k resolution.**

Mode	Encoding	FullHD	4K
Normal	AVC	68	135
	HEVC	34	22
Low latency	AVC	32	32
	HEVC	32	22

Decoder Software Features

Table 76: **Decoder 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 (Except FMO/ASO) Main High High10 High 4:2:2 High10 Intra High 4:2:2 Intra

Table 76: **Decoder Features** *(cont'd)*

Video Coding Parameter	H.265 (HEVC)	H.264 (AVC)
Levels	Up to 5.1 High Tier	Up to 5.2
Resolutions	4096x2160p60 with specific device(-2,-3) Up to 3840x2160p60	4096x2160p60 with specific device(-2,-3) Up to 3840x2160p60
Chroma format	4:2:0, 4:2:2	4:2:0, 4:2:2
Bit Depth	8-bit, 10-bit	8-bit, 10-bit

GStreamer Decoding Parameters

Table 77: **GStreamer Decoding Parameters**

Parameter	GStreamer property	Description
Entropy Buffers	internal-entropy-buffers	Specifies decoder internal entropy buffers, used to smooth out entropy decoding performance. Specify values in integer between 2 and 16. Increasing buffering-count increases decoder memory footprint. Default value: 5. Set this value to 10 for higher bit-rate use cases. For example, uses cases where the bitrate is more than 100 Mb/s.
Latency	latency-mode	For 2019.1 and later releases: Specifies decoder latency mode. (0): false - Normal mode (1): true - If alignment is AU, reduced-latency mode and if alignment is NAL, low-latency mode Default value: 0. For 2018.3 and prior releases: 0 = default 1 = reduced-latency (Low reference DPB mode) 2 = low-latency ***Note*:** When using low latency mode the encoder and decoder are limited by the number of internal cores. The encoder has a maximum of four streams and the decoder has a maximum of two stream. For more details, see VCU Latency Modes.
Split-input mode	split-input	When enabled, the decoder has 1 to 1 mapping for input and output buffers. When disabled, the decoder copies all the input buffers to internal circular buffer and processes them. Default: FALSE
QoS	Qos	Drop late frames based on QOS events. Default: FALSE

Decoder Maximum Bit Rate Tests

Pipeline used for measuring maximum bit rate for which decoder produces 30 fps.

Table 78: **Decoder Maximum Bit Rate Tests**

Codec	IPPP	IPBB	Intra-only
H.264 (AVC)	350 Mb/s	335 Mb/s	400 Mb/s
H.265 (HEVC)	275 Mb/s	275 Mb/s	270 Mb/s

Example pipeline used for measurement:

```
gst-launch-1.0 filesrc location="/run/test_file.mp4 " ! qtdemux !
h264parse ! omxh264dec internal-entropy-buffers=9 ! queue max-size-
bytes=0 ! fpsdisplaysinkname=fpssink text-overlay=false video-sink=kmssink
sync=true fps-update-interval=3000 -v
```

Max-Bitrate Benchmarking

The following tables summarize the maximum bit rate achievable for 3840x2610p60 resolution, XV20 pixel format at GStreamer level. The maximum supported target bit rate values vary based on what elements and type of input used in the pipeline.

Maximum Bit Rate support for Record Use Case with 4kp60 Resolution

Table 79: **Encoder Maximum Bit Rate Tests with XV20 format**

Video Recording (Live video capture → VCU encoder → Parser → Muxer → filesink)						
Format	**Codec**	**Entropy Mode**	**Rate Control Mode**	**B-Frames = 4**	**DDR Mode**	**Max Target Bitrate**
4:2:2, 10-bit	H.264 (AVC)	CABAC	VBR	IBBBBP	PS-DDR	160 Mb/s
		CAVLC	VBR	IBBBBP	PS-DDR	160 Mb/s
	H.265 (HEVC)	-	VBR	IBBBBP	PS-DDR	267 Mb/s

Example pipeline used for measurement:

```
ENC_EXTRA_IP_BUFFERS=5 gst-launch-1.0 -e v4l2src device=/dev/video0 io-
mode=4 num-buffers=7320 ! video/x-raw, width=3840, height=2160,
format=NV16_10LE32, framerate=60/1 ! omxh265enc control-rate=variable
target-bitrate=267000 max-bitrate=320400  gop-mode=basic gop-length=60 b-
frames=4 num-slices=8 prefetch-buffer=TRUE cpb-size=1000 initial-
delay=500 ! video/x-h265 , profile=main-422-10   ! queue max-size-bytes=0 !
h265parse ! mp4mux ! fpsdisplaysink name=fps1sink text-overlay=false fps-
update-interval=1000 video-sink="filesink location=/run/
test_4kp60_xv20_hevc_267000.mp4"
```

***Note*:**

1. It is recommended to store output file in RAM or SATA.

2. Achieved target bit rate depends on complexity of video content.
3. Above data is captured by recording a file in mp4 container format.

Maximum Bit Rate Support for Playback Use Case with 4kp60 Resolution

Playback (Filesrc → Decoder → Display)						
Format	**Codec**	**Entropy Mode**	**Rate Control Mode**	**B-Frames = 4**	**DDR Mode**	**Max Target Bitrate**
4:2:2, 10-bit	H.264 (AVC)	CABAC	VBR	IBBBBP	PL_DDR	120 Mb/s
		CAVLC	VBR	IBBBBP	PL_DDR	160 Mb/s
	H.265 (HEVC)	-	VBR	IBBBBP	PL_DDR	267 Mb/s

Example pipeline used for measurement:

```
gst-launch-1.0 filesrc location=/run/test_4kp60_xv20_hevc_267000.mp4 !
qtdemux ! h265parse ! omxh265dec internal-entropy-buffers=5 ! queue max-
size-bytes=0 ! fpsdisplaysink name=fps1sink text-overlay=false video-
sink="kmssink bus-id="a00c0000.v_mix" plane-id=33 render-
rectangle=<0,0,3840,2160> fullscreen-overlay=1" -v
```

***Note*:** It is recommended to keep input file in RAM or SATA.

Maximum Bit Rate Support for Streaming Use Case with 4kp60 Resolution

Table 80: **Maximum Bit Rate Support for Streaming Use Case with 4kp60 Resolution**

Video Streaming (Live video capture → VCU encoder → Parser → rtppay → Stream-out Stream-in → rtpdepay → Decoder → Display)						
Format	**Codec**	**Rate Control Mode**	**Latency Mode**	**B-Frames = 0**	**DDR Mode**	**Max Target Bitrate**
4:2:2, 10-bit	H.264 (AVC)	LOW_LATENCY	Normal	IPPP	Encoder (PS_DDR), Decoder (PL_DDR)	90 Mb/s
			Reduced			90 Mb/s
			Low latency			25 Mb/s
			Xilinx low latency			25 Mb/s
		CBR + max-picture-size	Normal			90 Mb/s
			Reduced			90 Mb/s
	H.265 (HEVC)	LOW_LATENCY	Normal			130 Mb/s
			Reduced			130 Mb/s
			Low latency			25 Mb/s
			Xilinx low latency			25 Mb/s
		CBR + max-picture-size	Normal			110 Mb/s
			Reduced			110 Mb/s

Example pipeline used for measurement:

- **H.265 (HEVC) with Low-latency Rate Control Mode and Xilinx Low Latency:**
  - **Server:**

```
gst-launch-1.0 -v v4l2src device=/dev/video0 io-mode=4 ! video/x-raw\
(memory:XLNXLL\),
format=NV16_10LE32,width=3840,height=2160,framerate=60/1 ! queue !
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 ! queue max-size-buffers=0 ! rtph265pay ! udpsink buffer-
size=60000000 host=192.168.0.2 port=5004 async=false max-lateness=-1
qos-dscp=60 max-bitrate=120000000 -v
```

  - **Client:**

```
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 ! queue !
rtph265depay ! h265parse ! video/x-h265, alignment=nal ! queue !
omxh265dec low-latency=1 ! video/x-raw\(memory:XLNXLL\) ! queue max-
size-bytes=0 ! fpsdisplaysink name=fpssink text-overlay=false video-
sink="kmssink bus-id=a00c0000.v_mix" sync=true -v
```

- **H.265 (HEVC) with CBR Rate Control Mode and Normal Latency Mode:**
  - **Server:**

```
gst-launch-1.0 -v v4l2src device=/dev/video0 io-mode=4 ! video/x-
raw,format=NV16_10LE32,width=3840,height=2160,framerate=60/1 ! queue !
omxh265enc num-slices=8 gop-length=120 periodicity-idr=120 control-
rate=constant max-picture-size=2017 prefetch-buffer=true target-
bitrate=110000 gop-mode=low-delay-p ! video/x-h265, alignment=au !
rtph265pay ! udpsink buffer-size=60000000 host=192.168.0.2 port=5004
max-lateness=-1 qos-dscp=60 async=false max-bitrate=120000000 -v
```

  - **Client:**

```
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=1000 ! rtph265depay !
h265parse ! video/x-h265, alignment=au ! omxh265dec low-latency=0 !
queue max-size-bytes=0 ! fpsdisplaysink name=fpssink text-overlay=false
video-sink="kmssink bus-id=a00c0000.v_mix fullscreen-overlay=1"
sync=true -v
```

***Note*:** The above data is captured by streaming elementary stream over RTP.

Maximum Bit Rate Support for Serial Use Case with 4kp60 Resolution

Table 81: **Maximum Bit Rate Support for Serial Use Case with 4kp60 Resolution**

Serial (Live video capture → VCU encoder → VCU decoder → Display)						
Format	**Codec**	**Rate Control Mode**	**Latency Mode**	**B-Frames = 0 or 4**	**DDR Mode**	**Max Target Bitrate**
4:2:2, 10-bit	H.264 (AVC)	LOW_LATENCY	Low Latency	IPPP	Encoder (PS_DDR), Decoder (PL_DDR)	25 Mb/s
			Xilinx Low Latency	IPPP		25 Mb/s
		CBR + max-picture-size	Normal	IBBBBP		90 Mb/s
			Reduced	IPPP		200 Mb/s
	H.265 (HEVC)	LOW_LATENCY	Low Latency	IPPP		25 Mb/s
			Xilinx Low Latency	IPPP		25 Mb/s
		CBR + max-picture-size	Normal	IBBBBP		120 Mb/s
			Reduced	IPPP		200 Mb/s

Example pipeline used for measurement:

- **H.265 (HEVC) with Low-latency Rate Control Mode and Xilinx Low Latency:**

```
gst-launch-1.0 -v v4l2src io-mode=dmabuf device=/dev/video0 ! video/x-raw\
(memory:XLNXLL\), width=3840, height=2160, format=NV16_10LE32,
framerate=60/1 ! queue ! omxh265enc num-slices=8 control-rate=low-latency
gop-mode=low-delay-p target-bitrate=25000 cpb-size=500 gdr-
mode=horizontal initial-delay=250 periodicity-idr=240 filler-data=0
prefetch-buffer=true ! video/x-h265, alignment=nal ! queue max-size-
buffers=0 ! omxh265dec low-latency=1 ! video/x-raw\(memory:XLNXLL\) !
queue max-size-bytes=0 ! fpsdisplaysink name=fpssink text-overlay=false
"video-sink=kmssink bus-id=a00c0000.v_mix hold-extra-sample=1 show-
preroll=false sync=true" sync=true -v
```

- **H.265 (HEVC) with CBR Rate Control Mode and Normal Latency Mode:**

```
gst-launch-1.0 -v v4l2src device=/dev/video0 io-mode=4 ! video/x-
raw,format=NV16_10LE32,width=3840,height=2160,framerate=60/1 ! omxh265enc
num-slices=8 gop-length=60 b-frames=4 control-rate=2 prefetch-buffer=true
target-bitrate=120000 max-picture-size=2200 ! video/x-h265,
alignment=au ! queue ! video/x-h265, profile=main-422-10, alignment=au !
omxh265dec low-latency=0 ! queue max-size-bytes=0 ! fpsdisplaysink
name=fpssink text-overlay=false "video-sink=kmssink bus-id=a00c0000.v_mix
fullscreen-overlay=1" sync=true -v
```

Gstreamer and V4L2 Formats

The Gstreamer and V4L2 Formats are described in this section.

- These formats signify the memory layout of pixels. It applies at the encoder input and the decoder output side.
- The encoder needs to know the memory layout of pixel at input side for reading the raw data, so the corresponding video format needs to be specified at encoder sink pad using caps.
- For the decoder, you can specify the format to be used to write the pixel in memory by specifying the corresponding Gstreamer video format using caps at decoder source pad.
- When the format is not supported between two elements, the cap negotiation fails and Gstreamer returns an error. In that case, you can use the video convert element to perform software conversion from one format to another.

The following table shows the GStreamer and V4L2 related formats that are supported.

Table 82: **Gstreamer and V4L2 Formats**

Pixel Format	V4L2	GStreamer
YUV 420 8-bit	V4L2_PIX_FMT_NV12	GST_VIDEO_FORMAT_NV12
YUV 420 10-bit	V4L2_PIX_FMT_XV15	GST_VIDEO_FORMAT_NV12_10LE32
YUV 422 8-bit	V4L2_PIX_FMT_NV16	GST_VIDEO_FORMAT_NV16
YUV 422 10-bit	V4L2_PIX_FMT_XV20	GST_VIDEO_FORMAT_NV16_10LE32

***Note*:**

1. The fourcc codes are the last four characters of v4L2 pixel formats (for example, GREY/XY10).
2. Y_Only 8-bit and Y_Only 10-bit are supported only at the control software layer for the encoder and decoder.
3. For more information on YUV 444 format support, refer this article.

Preparing PetaLinux to Run VCU Applications

IMPORTANT! *QoS settings must be tuned based on application. Review the recommendations for adjusting the QoS settings and make the recommended changes according to system to avoid performance issues when using the VCU.*

Before VCU applications can be run successfully in PetaLinux, changing the Quality of Service (QoS) settings may be required to change the priority of any AFI port to read/write data and may help with bandwidth related issues. Changing the QoS settings and the command issuing capabilities of the AXI ports to the VCU Decoder (M_AXI_DEC0 and M_AXI_DEC1) must be modified to avoid traffic congestion with respect to Display port in case of decode and display use-case. If the QoS settings are not modified, the DisplayPort/HDMI transmission may under-run, producing green or black frames intermittently during video playback. Not increasing the command issuing capability may limit the framerate for challenging settings such as 4kp60.

See Designing with the Core regarding the ports connected to the decoder. Understanding where the following addresses came from will enable you to adjust the following commands for designs using alternate AXI connection. See the *Zynq UltraScale+ Device Register Reference* (UG1087) for AXI control register addresses and settings.

1. Boot the board using a PetaLinux pre-built image.
2. Login with username:root and password:root
3. Set read and write QoS of the port connected to M_AXI_DEC0
   - Set the S_AXI_HP0_FPD RDQoS (AFIFM) register to LOW_PRIORITY

     ```
     devmem 0xFD380008 w 0x3
     ```

   - Set the S_AXI_HP0_FPD WRQoS (AFIFM) register to LOW_PRIORITY

     ```
     devmem 0xFD38001C w 0x3
     ```

4. Set read and write QoS of the port connected to M_AXI_DEC1
   - Set the S_AXI_HP3_FPD RDQoS (AFIFM) register to LOW_PRIORITY

     ```
     devmem 0xFD3B0008 w 0x3
     ```

   - Set the S_AXI_HP3_FPD WRQoS (AFIFM) register to LOW_PRIORITY

     ```
     devmem 0xFD3B001C w 0x3
     ```

5. Increase the read and write issuing capability of the port connected to M_AXI_DEC0. By default, it can take a maximum of four requests at a time, and increasing the issuing capability may keep the ports busy with always some requests in the queue.
   - Set the S_AXI_HP0_FPD RDISSUE (AFIFM) register to allow 16 commands

     ```
     devmem 0xFD380004 w 0xF
     ```

   - Set the S_AXI_HP0_FPD WRISSUE (AFIFM) register to allow 16 commands

     ```
     devmem 0xFD380018 w 0xF
     ```

6. Increase the read and write issuing capability of the port connected to M_AXI_DEC1
   - `devmem 0xFD3B0004 w 0xF`
   - Set the S_AXI_HP3_FPD WRISSUE (AFIFM) register to allow 16 commandsSet the S_AXI_HP3_FPD RDISSUE (AFIFM) register to allow 16 commands

     ```
     devmem 0xFD3B0018 w 0xF
     ```

Now, the GStreamer, OMX, and Xilinx Control Software pipelines can be run on the board.

Integrating the VCU and GStreamer Patches

1. Extract PetaLinux BSP.
2. Create "recipe-multimedia" folder in project-spec/meta-user folder.

   ```
   cd project-spec/meta-user
   mkdir recipe-multimedia
   ```

3. For gstreamer patches, follow below steps
   a. Create “gstreamer” directory in recipe-multimedia folder.

      ```
      cd recipe-multimedia
      mkdir gstreamer
      ```

   b. There are 5 different recipes files for gstreamer that downloads the code and compile
      - `gstreamer1.0_%.bbappend`
      - `gstreamer1.0-omx_%.bbappend`
      - `gstreamer1.0-plugins-bad_%.bbappend`
      - `gstreamer1.0-plugins-base_%.bbappend`
      - `gstreamer1.0-plugins-good_%.bbappend`

c. Depending upon patches to which gstreamer package it belongs to, bbappend file for that package needs to be created to get those patches applied and compiled on latest source code. For example, if patch fix is for gst-omx, follow these steps

i. Create a `gstreamer1.0-omx` directory in the `recipe-multimedia/gstreamer` folder

```
cd gstreamer
mkdir gstreamer1.0-omx
```

ii. Copy gst-omx patches in `gstreamer1.0-omx` directory.

```
cp test1.patch recipe-multimedia/gstreamer/gstreamer1.0-omx
cp test2.patch recipe-multimedia/gstreamer/gstreamer1.0-omx
```

iii. Create a `gstreamer1.0-omx_%.bbappend` file in `recipe-multimedia/gstreamer` folder.

```
vi gstreamer1.0-omx_%.bbappend
```

iv. Append the following lines in the `gstreamer1.0-omx_%.bbappend` file.

```
FILESEXTRAPATHS_prepend: = "${THISDIR}/gstreamer1.0-omx:"
SRC_URI_append = " \
file://test1.patch \
file://test2.patch \“
```

Create similar bbappend files and folder for other gstreamer package to integrate any custom patches in PetaLinux build.

4. For VCU patches, follow these steps:

a. Create a `vcu` directory in the `recipe-multimedia` folder.

```
cd project-spec/meta-user/recipe-multimedia
mkdir vcu
```

b. There are four different recipes files for VCU that downloads the code and compile

- `kernel-module-vcu_%.bbappend`
- `vcu-firmware_%.bbappend`
- `libvcu-xlnx_%.bbappend`
- `libomxil-xlnx_%.bbappend`

c. Depending upon patches to which VCU source code it belongs to, bbappend file for that code base needs to be created to get those patches applied and compiled on latest source code. For example, if the patch fix is for VCU drivers, follow these steps:

i. Create a `kernel-module-vcu` directory in the `recipe-multimedia/vcu` folder

```
cd vcu
mkdir kernel-module-vcu
```

ii. Copy VCU driver patches to the `kernel-module-vcu` directory.

```
cp test1.patch recipe-multimedia/vcu/kernel-module-vcu
cp test2.patch recipe-multimedia/vcu/kernel-module-vcu
```

iii. Create a `kernel-module-vcu_%.bbappend` file in the `recipe-multimedia/vcu` folder:

```
vi kernel-module-vcu_%.bbappend
```

iv. Append the following lines to the `kernel-module-vcu_%.bbappend` file.

```
FILESEXTRAPATHS_prepend: = "${THISDIR}/ kernel-module-vcu:"
SRC_URI_append = " \
file://test1.patch \
file://test2.patch \“
```

Create similar bbappend files and folder for other VCU component to integrate any custom patches in PetaLinux build.

5. Follow PetaLinux build steps to generate updated binaries.

Note: If you are not compiling with PetaLinux, review the recipes for additional files necessary for setting up GStreamer. For example, you must include the `/etc/xdg/gstomx.conf` in the root file system. This file tells gst-omx where to find the OMX integration layer library - libOMX.allegro.core.so.1.

VCU Encoder Features

The VCU encoder supports the following features.

Both the Xilinx VCU Control Software application and the GStreamer support scheduled bitrate changes at given frame numbers. The pre-built PetaLinux image includes the zynqmp_vcu_encode application in `/usr/bin/`. Source code of the GStreamer application is part of the gst-omx repository (https://github.com/Xilinx/gst-omx/tree/xlnx-rebase-v1.16.3/examples/zynqultrascaleplus).

See the usage message for options:

```
zynqmp_vcu_encode --help
```

Constraints

The initial encoding session should start with the worst case maximum value for dynamic bitrate and dynamic B-frames parameters, for example, encoding session should start with num-bframes = 4 if you are planning to modify B-frames dynamically during the encode session. Similarly, for target-bitrate the encoding session should start with max-bitrate planned. You can alter the bitrate later, during encode.

Dynamic-Bitrate

Dynamic-bitrate is the ability to change encoding bitrate (target-bitrate) while the encoder is active.

To change the bitrate of the video at frame number 100 to 1 Mb/s:

```
zynqmp_vcu_encode -w 3840 -h 2160 -e avc -r 5000 -o /run/op.h264 -i /run/
input.yuv -d BR:100:1000
```

The syntax of the -d parameter is:

<keyword>:<framenumber>:<value>

Dynamic GOP

Dynamic GOP is the ability to change gop-length, number of B-Frames, and forcing IDR picture while the encoder is active.

To change the gop-length of the video at frame number 100 to 45:

```
zynqmp_vcu_encode -w 3840 -h 2160 -e avc -g 30 -o /run/op.h264 -i /run/
input.yuv -d GL:100:45
```

To change the number of B-frames of the video at frame number 20 to 2:

```
zynqmp_vcu_encode -w 3840 -h 2160 -e avc -b 4 -o /run/op.h264 -i /run/
input.yuv -d BFrm:20:2
```

To insert a key frame at frame number 35:

```
zynqmp_vcu_encode -w 3840 -h 2160 -e avc -b 4 -o /run/op.h264 -i /run/
input.yuv -d KF:35
```

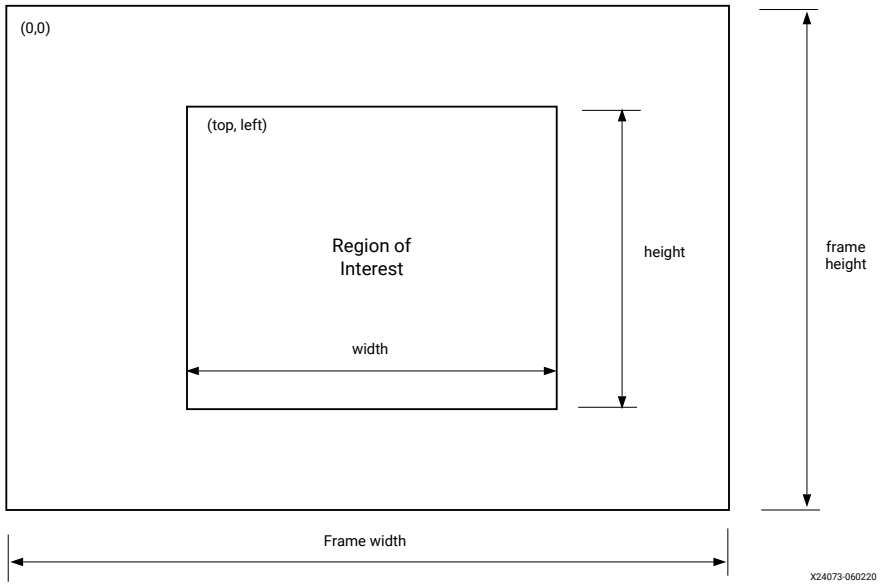
Guidelines for Dynamic Controls

- Both the bitrate and the GOP length should be set to the largest possible value allowed by the application, to increase the video quality.
- Dynamic parameters should not be changed frequently, as that may cause stability issues since the algorithm does not have time to settle. It is recommended that you wait for a time period of at least twice or thrice the GOP length when changing the dynamic parameters.
- An Intra/IDR frame is inserted on the first frame corresponding to a scene change.
- Dynamic GOP changes comes in effect immediately, that is, either the current GOP is closed earlier, or it is extended so that its actual size corresponds to the new parameters.
- Dynamic bitrate changes are considered immediately (may impact the QP of the next frames). However, the new average target bitrate may take some time depending on the content, bitrate delta, and so on. It may take up to ~1 GOP duration to converge.

Region of Interest Encoding

The VCU encoder currently supports Region of Interest (ROI) encoding which allows you to define several independent or overlapped regions within a frame. The ROI 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). An example ROI defined in a frame is shown in the following figure .

Figure 45: **Region of Interest Encoding**



You can provide the region of interest (ROI) location (top, left) in pixels and width and height in pixels along with quality index. Multiple and overlapped ROI regions within a frame are supported. The sample GStreamer application only adds one ROI region but you can attach multiple ROI meta data properties to the buffer.

The input format is:

```
ROI:<frame number>:<top>x<left>:<width>x<height>:<ROI type>
```

A sample GStreamer application command line option to encode video with ROI region attached to specific frame number at 100 is:

```
zynqmp_vcu_encode -w 3840 -h 2160 -e avc -o /run/op.h264 -i /run/input.yuv -
d ROI:100:1280x360:1220x1500:high
```

Custom ROI Delta-QP

The new custom ROI delta-qp feature allows you to provide your own delta-qp value for any region of interest to modify the quality level.

The following dynamic command has been added to the zynqmp_vcu_encode application:

```
"ROI_BY_VALUE:<frame number>:<top>x<left>:<width>x<height>:<delta_qp>""
```

This creates a new gstreamer meta structure: `roi-by-value/omx-alg`. Using this meta structure, the gstreamer fills out the new OMX config, `OMX_ALG_VIDEO_CONFIG_REGION_OF_INTEREST_BY_VALUE`, to send the parameters to the VCU.

The following is an example pipeline:

```
zynqmp_vcu_encode -w 1920 -h 1080 -e avc -f 60 -c 2 -g 30 -o test.h264 -i
test.yuv -d ROI_BY_VALUE:0:1200x300:200x200:10 -q 1
```

Note: This feature must be enabled at the start of the stream (frame 0). This is a limitation of the VCU. Values for delta-qp must be within –32 to 31 because they are relative QP values.

ROI-QP

You can set ROI live streaming dynamically using the GStreamer and control software.

GStreamer

As showcased in example application code, use the following two APIs to set ROI region and quality for each frame: https://github.com/Xilinx/gst-omx/blob/xlnx-rebase-v1.16.3/examples/zynqultrascaleplus/test-vcu-encode.c (line number #501).

- `gst_buffer_add_video_region_of_interest_meta()`
- `gst_video_region_of_interest_meta_add_param()`

Control Software

The control software exports a few APIs to provide ROI information to the encoder per frame. Use these APIs to add ROIs in live streaming.

- `AL_RoiMngr_Create()` called during encoder creation.
- `AL_RoiMngr_Clear()`, `AL_Roi_Mngr_AddROI()`, and `AL_RoiMngr_FillBuff()` are called for each frame to provide ROI information to encoder.

Refer to the Doxygen output for more details on APIs.

LongTerm Reference Picture

Inserts long-term picture dynamically in a GOP and uses LT reference for particular picture based on user input.

Sample Gstreamer application cmd line option to insert a longterm picture at frame 10:

```
>> zynqmp_vcu_encode -w 3840 -h 2160 -e avc -o /run/op.h264 -i /run/
input.yuv -d IL:10
```

Command line option to use LTR for frame 15:

```
>> zynqmp_vcu_encode -w 3840 -h 2160 -e avc -o /run/op.h264 -i /run/
input.yuv -d UL:15
```

Adaptive GOP

Specify the maximum number of B-frames that can be used in a GOP and set the GOP mode to adaptive using `GopCtrlMode=ADAPTIVE_GOP` in the encoder configuration file at control software and `gop-mode=adaptive` at 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 specified.

Insertion of SEI Data at Encoder

`AL_Encoder_AddSei()` 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. The SEI cannot exceed 2 KB. This should be largely sufficient for all the SEI NALs.

The stream buffer needs to have enough space to add the SEI section. If not, `AL_Encoder_AddSei()` reports a failure. Prefix and suffix SEI are supported. The encoder puts the SEI section at the correct place in the stream buffer to create a conformant stream. The control software application show cases adding a SEI message using `AL_Encoder_AddSei()`. To insert multiple SEI messages to stream, `AL_Encoder_AddSei()` can be called multiple times or multiple SEI payloads can be added to `AL_Encoder_AddSei()` API. 2 KB limit is per stream buffer, i.e. all SEI messages per AU should be with-in the 2 KB limit. SEI messages are already synchronized with corresponding video frames; there should not be a need for timestamping mechanism for synchronization. VCU-SW supports adding SEI meta-data through omx/gstreamer application. The following OMX_API /Indexes and Struct are used to insert SEI meta-data.

- OMX_ALG_IndexConfigVideoInsertPrefixSEI
- OMX_ALG_IndexConfigVideoInsertSuffixSEI
- Struct: OMX_ALG_VIDEO_CONFIG_DATA

You can insert your own data using gstreamer application at any frame. The reference implementation function is as follows:

```
static gboolean send_downstream_event (GstPad * pad, GstStructure * s)
{
GstEvent *event;
GstPad *peer;
event = gst_event_new_custom (GST_EVENT_CUSTOM_DOWNSTREAM, s);
peer = gst_pad_get_peer (pad);
if (!gst_pad_send_event (peer, event))
    _g_printerr ("Failed to send custom event\n");_
    gst_object_unref (peer);
    return TRUE;
}
```

```
static GstPadProbeReturn buffer_probe (GstPad * pad, GstPadProbeInfo *
info,
gpointer user_data)
{
const gchar *data = "some data";
GstBuffer *sei;
GstStructure *s;
sei = gst_buffer_new_wrapped (g_strdup (data), strlen (data));
s = gst_structure_new ("omx-alg/insert-prefix-sei", "payload-type",
G_TYPE_UINT, 77,
"payload", GST_TYPE_BUFFER, sei, NULL);
send_downstream_event (pad, s);
gst_buffer_unref (sei);
}
```

Refer to Xilinx VCU Control Software API for more details.

SEI Decoder API

The SEI message can be retrieved from the decoder. The `AL_CB_ParsedSEI` callback is invoked each time the decoder parses an SEI. The sei_payload data is provided as described in Annex D.3 of ITU-T. Anti-emulation has been removed by the decoder. The control software application shows an example of API usage which can be triggered using the `-sei-file` option on the command-line. The VCU-SW supports parsing of SEI meta-data using the omx/gstreamer application too.

When an SEI is parsed, the OMX component should call `EventHandle` with eEvent based on the SEI type: OMX_ALG_EventSEIPrefixParsed and OMX_ALG_EventSEISuffixParsed. The following is the reference implementation to handle SEI event using Gstreamer application:

```
static gboolean bus_call (GstBus * bus, GstMessage * msg, gpointer data)
{
    GMainLoop *loop = (GMainLoop *) data;
    switch (GST_MESSAGE_TYPE (msg)) {
    case GST_MESSAGE_APPLICATION: {
            const GstStructure *s;
guint sei_type;
GstBuffer *buf;
gboolean is_prefix;
s = gst_message_get_structure (msg);

if (gst_structure_has_name (s, "omx-alg/sei-parsed")) {
    gst_structure_get (s, "payload-type", G_TYPE_UINT, &sei_type,
"payload", GST_TYPE_BUFFER, &buf, "prefix", G_TYPE_BOOLEAN, &is_prefix,
NULL);
    gst_util_dump_buffer (buf);
    gst_buffer_unref (buf);
}
break;
}
}
return TRUE;
}
```

The limitations are:

- These are non-blocking callbacks.
- These extraction call-backs are asynchronous to display order; as soon as SEI NAL unit is parsed in the decoder, the callback is triggered from the control software. You must maintain a FIFO, if there is any re-ordering due to B-frames.

Refer to the Xilinx VCU Control Software API for more details.

Dual Pass Encoding

Encode the video twice; the first pass collects information about the sequence, and stats are used to improve the encoding of the second pass. The two levels are:

- Real-time GOP/frame-level dual-pass < lookahead mode >
- Offline sequence level dual-pass < two pass modes >

Until the 2019.1 release, only real-time GOP/Frame-level dual-pass is supported. IDR frames are automatically inserted based on first-pass scene change detection. The QP of each frame is adjusted based on internal stream size/complexity statistics/Scene change. Gop/Fame-level dualpass encoding is enabled by using "LookAhead" parameter at control-software and "look-ahead" at Gstramer.

Dual pass is not supported in the low latency mode.

Constraints: A maximum of 4kp30 is allowed in dual pass mode because the maximum VCU performance is 4kp60 and dual pass reduces the performance to half.

LookAhead = <value>

- Default value is 0: Dual pass is disabled.
- Above 2 (>=2): Scene change detection and correction.
- Above 10: Constant quality, using frame complexity

Scene Change Detection

The Xilinx video scene change detection IP provides a video processing block that implements scene change detection algorithm. The IP core calculates the histogram on a vertically subsampled luma frame for consecutive frames. The histogram of these frames are then compared using the sum of absolute differences (SAD). This IP core programmable through a comprehensive register interface to control the frame size, video format, and subsampling value. For more information, see the *Video Scene Change Detection LogiCORE IP Product Guide* (PG322) .

The scene change detection IP is a configurable IP core that can read up to eight video streams in memory mode. In memory mode. All inputs are read from memory mapped AXI4 interface. The IP supports resolutions ranging from 64x64 to 8192x4320 with various 8-bit and 10-bit color formats. The IP sends the interrupt after generating the SAD values for all the input streams for every frame. In case of memory mode, the SAD values are calculated for every input stream one after the other sequentially and the interrupt is after the SAD calculation of the final stream. On the interrupt generation, the SAD values are read from SAD registers for configured number of streams to compare them with a user decided threshold value to determine if a scene change has occurred.

Interlaced scene change detection is supported in the memory mode. For interlaced content, SAD values are computed for each field instead of each frame, so a scene change can be detected in between fields of the same frame.

The most important application of scene change detection are video surveillance, machine learning and video conference. In these use cases, the SCD IP would be in the capture pipeline and the generate event is attached with each buffer captured. The same event is passed along with the buffer to the encoder, where the encoder makes decisions to insert an I-frame instead of a P-frame or a B-frame. Inserting an I-frame at the place where a scene is changed would retain the quality of the video.

Gstreamer pipelines:

- File → decode → scd → encode

```
gst-launch-1.0 filesrc location=file.mp4 ! qtdemux ! h264parse !
omxh264dec ! queue ! xilinxscd io-mode=5 ! queue ! omxh264enc target-
bitrate=20000 control-rate=2 cpb-size=5000 ! filesink location=file.264
```

- YUV → scd → encode

```
gst-launch-1.0 filesrc location=file.yuv ! rawvideoparse width=1920
height=1080 format=nv12 framerate=30/1 ! xilinxscd ! omxh264enc target-
bitrate=8000 control-rate=2 cpb-size=4000 ! filesink location=file.264
```

Interlaced Video

The Interlaced video is a technique to double the perceived frame rate of a video display with the same bandwidth. The interlaced frame contains two fields captured at two different times. This improves motion perception to the viewer, and reduces flicker by taking advantage of viewing in rapid succession as continuous motion.

Interlaced signals require a display that is capable of showing the individual fields in a sequential order. CRT displays are made for displaying interlaced signals. Interlaced scan refers to drawing a video image on display screen by scanning each line or row of pixels. This technique uses two fields to create a frame. One field contains all odd-numbered lines in the image and the other contains all even-numbered lines.

The Xilinx VCU supports encoding and decoding of H265 interlaced video. VCU can decode and encode of various video resolutions like 1920x1080i, 720x576i, and 720x480i.

Gstreamer pipelines:

- filesrc → omxh265dec → omxh265enc → filesink

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

- v4l2src → omxh265enc → omxh265dec → kmssink

```
 gst-launch-1.0 v4l2src io-mode=4 ! video/x-raw\(format:Interlaced
\),interlace-mode=alternate,
format=NV16_10LE32,width=1920,height=1080,framerate=30/1 ! omxh265enc
target-bitrate=10000 control-rate=2 ! omxh265dec ! queue max-size-
bytes=-1 ! kmssink bus-id=drm-pl-disp-drv connector-
properties="props,sdi_mode=0,sdi_data_stream=2,is_frac=0"
```

- Audio serial pipeline: v4l2src → omxh265enc → omxh265dec → kmssink alsasrc → audioconvert → audioresample → alsasink

```
gst-launch-1.0 v4l2src device=/dev/video0 io-mode=4 ! video/x-raw\
(format:Interlaced\),interlace-mode=alternate,field-order=bottom-field-
first,format=NV16_10LE32,width=1920,height=1080,framerate=30/1 !
omxh265enc target-bitrate=20000 ! queue ! omxh265dec ! queue max-size-
bytes=0 ! kmssink bus-id=amba_pl@0:drm-pl-disp-drv connector-
properties="props,sdi_mode=0,sdi_data_stream=2,is_frac=0" show-preroll-
frame=false sync=true alsasrc device=hw:1,1 ! queue ! audioconvert !
audioresample ! audio/x-raw, rate=48000, channels=2, format=S24_32LE !
alsasink device="hw:1,0"
```

- Audio file playback pipeline: filesrc → tsdemux → omxh265dec → kmssink tsdemux → faad → audioconvert → audioresample → alsasink

```
gst-launch-1.0 filesrc location=/media/usb/video/
sdi_cap_1080i_aac_coffea_long.ts ! video/mpegts ! tsdemux name=demux
demux. ! queue ! h265parse ! omxh265dec ! queue ! kmssink bus-
id=amba_pl@0:drm-pl-disp-drv connector-
properties="props,sdi_mode=0,sdi_data_stream=2,is_frac=0" show-preroll-
frame=false sync=true demux. ! queue ! aacparse ! faad ! volume
volume=9 ! audioconvert ! audioresample ! audio/x-raw, rate=48000,
channnels=2, format=S24_32LE ! alsasink device="hw:1,0" provide-
clock=false
```

***Note*:** It is suggested not to use alsasink clock as the Gstreamer clock provider. There is a limitation of interlaced audio that alsasink clock and video rendering depending on each other, so setting alsasink provide-clock to FALSE for file playback case.

GDR Intra Refresh

Gradual Decoder Refresh (GDR) when `GOPCtrlMode` is set to LOW_DELAY_P, the GDRMode parameter 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 specifies the frequency at which the refresh pattern should happen. To allow full picture refreshing, Gop.FreqIDR parameter should be greater than the number of CTB/MB rows (GDR_HORIZONTAL) or columns (GDR_VERTICAL).

When the GDR is enabled, the encoder inserts SPS/PPS along with an SEI recovery message at the start of a refresh pattern, and the decoder can synchronize to the SEI recovery points in the bit-stream. The advantage of having the non-IDR decoder synchronization is that there is no longer any need to send IDR pictures in case of a packet loss scenario. Inserting periodic IDR pictures in video encoding generates a spike in bit consumption which is not recommended for video streaming. Vertical GDR mode has an MV (motion vector) limitation that can cause full picture reconstruction to take more than one refresh cycle. Refer below table for more details:

Table 83: **GDR Limitation**

	AVC/H.264	HEVC/H.265
GDR_VERTICAL: A vertical intra-column moving from left to right	Exact match is supported. Deblocking filter is automatically disabled	Exact match is NOT supported; it can take up to two refresh intervals to fully reconstruct the frame deblocking filter is automatically disabled
GDR_HORIZONTAL: A horizontal intra-row moving from top-to-bottom	Exact match is supported deblocking filter is automatically disabled	Exact match is supported deblocking filter is automatically disabled

A sample gstreamer pipeline is as follows. Use "gdr-mode=horizontal" & "periodicity-idr=<higher than Picture Height in Mbs>" in the encoder element.

```
gst-launch-1.0 filesrc location="/mnt/test_4Kp60_NV12.yuv"! rawvideoparse
width=3840 height=2160 format=nv12 framerate=30/1! omxh264enc control-
rate=constant target-bitrate=60000 gop-mode=low-delay-p gdr-mode=horizontal
periodicity-idr=270! video/x-h264, profile=high! Filesink location="/run/
test_output.avc"
```

At the control software level, add the following section and parameter (in the GOP section of the cfg file)

```
[GOP] #-----------------------------------------------------
GopCtrlMode         = LOW_DELAY_P
Gop.GdrMode         = GDR_HORIZONTAL
Gop.FreqIDR         = 270
```

***Note*:** SPS/PPS is inserted at the start of each GDR frame start. VCU decoder can synchronize onto any GDR start frame, no need to start with the IDR frame always. The GDR frequency can be set using `GOP.FreqIDR` at control software and periodicity-idr at gstreamer when GDR mode is enabled.

Dynamic Resolution Change — VCU Encoder/Decoder

Dynamic Resolution Change is supported at the VCU control software and Gstreamer level for both the encoder/decoder. This feature is not supported at the OMX level yet.

VCU Decoder

An input compressed stream can contain multiple resolutions. The VCU Decoder can decode pictures without re-creating a channel.

Constraints

- The maximum resolution must be known.
- All the streams should belong to same codec either AVC or HEVC.
- Same chroma-format and bit-depth.
- Maximum resolution can be set either with the pre-allocation mode or with the first valid SPS. In case the SPS defines a resolution higher than the one provided in pre-allocation, the decoder skips the frames until the next valid SPS.
- While going from low to high without pre-allocation, resolutions lower than or equal to the first valid resolution that are met are decoded. Pictures and references should always fit in the DPB.

CtrlSWAPI

```
-prealloc-args max-widthxmax-height: video-mode:chroma-
mode:bitdepth:profile-idc:level
```

Callbacks

resolutionFoundCB is raised each time a new resolution is found.

- Example command (with pre-allocated arguments)

```
ctrlsw_decoder -i input.avc -avc --prealloc-args 3840x2160:
progr:420:8:66:51 -o output.yuv
```

- Example command (without pre-allocated arguments)

```
ctrlsw_decoder -i input.avc -avc -o output.yuv
```

DRC Decode: To decode a DRC file which is encoded at the control software level using the Gstreamer, use:

```
gst-launch-1.0 filesrc location=test.avc ! h264parse ! omxh264dec low-
latency=0 internal-entropy-buffers=5 ! queue max-size-bytes=0 ! filesink
location=test.yuv
```

VCU Encoder

Input sources may contain several resolutions:

Constraints

Maximum resolution must be known. You can only change the resolution; changing the chroma mode or bit-depth is not allowed.

CtrlSWAPI

`AL_Encoder_SetInputResolution`:Maximum resolution is specified in channel parameters.

Refer to the API section for more details on new APIs. Provide multiple resolution files in the .cfg file in the following format:

```
[INPUT] #-------------------------------------------------
YUVFile          = input_1.yuv
Format           = NV12
Width           = 1920
Height           = 1080
CmdFile          = input_cmd.txt
[DYNAMIC_INPUT] #-------------------------------------------------
YUVFile          = input_2.yuv
Width           = 720
Height           = 480
```

Provide the number of frame index for input and variable fps parameters (for DRC with variable FPS) in the `input_cmd.txt` file.

```
50: Fps=30, Input=1
90: Fps=60, Input=2
```

Use the following command to generate an output file that contains an encoded file containing various resolutions.

```
ctrlsw_encoder -cfg test.cfg -o output
```

DRC Encode: Checking encoder use case for DRC using transcoding of DRC file which is encoded at the control software level, use:

```
gst-launch-1.0 filesrc location=test.avc ! h264parse ! omxh264dec low-
latency=0 internal-entropy-buffers=9 ! queue max-size-bytes=0 ! omxh265enc
target-bitrate=50000 ! video/x-h265, profile=main, alignment=au ! filesink
location=test_out.hevc
```

Note: In case of Gstreamer level, the encoded file should have the first resolution as the maximum resolution of the stream.

Frameskip Support for the VCU Encoder

The VCU encoder currently supports the frame skip feature to strictly achieve the required target bitrate. 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.

You can also control the maximum number of consecutive skipped frames. However, if the maximum number of consecutive skipped frames is set too low, the output bitstream may not achieve the target bitrate. So, you may need to experiment to find a balance between the achieved bitrate and number of actual encoded frames.

Constraints

- This only applies if the reference frame has already been encoded. The first frame is encoded irrespective of the buffer overflow. Only subsequent pictures are discarded if actual frame size is exceeded.

  If the CPB is very small and the video is complex, all frames might be discarded.
- HRD compliance for dynamic commands is not guaranteed when the bitrate or fps change. Skip frames can be added or missed when those parameters change.
- 。 The back-and-forth visual effect can appear when the value of NumB is greater than 1 and `enable-skip` is set to true.

- In the dual pass mode, frame skip can only be used on the second pass.
- Inserting the frame skip in GOP pattern with more than one B-frame can have back and forth visual effect. NumB >1 is not recommended.
- Avoid frame skip in adaptive GOP feature.

Not Supported

- Frameskip is not supported in the low latency (--slicelat) mode.
- Frameskip is not supported in Pyramidal GOP.

Control Software API

- The AL_RC_OPT_ENABLE_SKIP bit is used in "eOptions" member variable of AL_TRCParam structure, which is encapsulated in AL_TEncChanParam structure to enable the frameskip feature.
- The `nal_ref_idc` flag can be used to detect skip frame as all the skip frames are not used for reference, which means that if the slice type is P and nal_ref_idc = 0, then that frame is a skip frame in the IPPPPPP GOP mode.
- The frame skip flag feature notifies user applications about which frame has been skipped to allow users to drop/do some extra processing on skipped frames.
- A new `bSkipped` flag has been added into the buffer metadata. When an encoded output buffer is ready, user applications can extract the buffer metadata from the stream buffer using the control software API, `AL_Buffer_GetMetaData()`. The boolean flag `bSkipped` indicates whether or not that particular buffer was skipped.
- A new uMaxConsecSkip member variable of the AL_TRCParam structure is used to set the maximum number of consecutive skipped frames.

OMX API

- New SetParam Index: OMX_ALG_IndexParamVideoSkipFrame
- Data structure: OMX_ALG_VIDEO_PARAM_SKIP_FRAME
- bEnableSkipFrame flag of OMX_ALG_VIDEO_PARAM_SKIP_FRAME structure is used to indicate if skip frame should be enabled.
- nMaxConsecutiveSkipFrame of OMX_ALG_VIDEO_PARAM_SKIP_FRAME structure is used to indicate the maximum number of consecutive skipped frames.

For enabling frameskip, enable the "EnableSkip" parameter in cfg file:

```
EnableSkip = TRUE
```

For setting the maximum number of consecutive skipped frames, set the “MaxConsecutiveSkip” parameter in cfg file:

```
MaxConsecutiveSkip = 5
```

Use the following commands:

- `ctrlsw_encoder -cfg test.cfg`
- `ctrlsw_encoder -cfg test.cfg -i test.yuv --input-width 1280 --input-height 720 -o test.avc --print-picture-type`

Gstreamer Pipeline

```
gst-launch-1.0 filesrc location=/mnt/gst_src/FIFA_4Kp60_nv12.yuv !
videoparse width=3840 height=2160 format=nv12 framerate=60/1 ! omxh264enc
target-bitrate=1000 control-rate=constant num-slices=8 gop-mode=basic gop-
length=60 prefetch-buffer=TRUE skip-frame=true ! video/x-h264,
profile=high, alignment=au ! filesink location=4k60_nv12.avc
```

```
gst-launch-1.0 filesrc location=/mnt/gst_src/FIFA_4Kp60_nv12.yuv !
videoparse width=3840 height=2160 format=nv12 framerate=60/1 ! omxh265enc
target-bitrate=1000 control-rate=constant num-slices=8 gop-mode=basic gop-
length=60 prefetch-buffer=TRUE skip-frame=true ! video/x-h265,
profile=main, alignment=au ! filesink location=4k60_nv12.hevc
```

```
gst-launch-1.0 filesrc location=/mnt/gst_src/FIFA_4Kp60_nv12.yuv !
videoparse width=3840 height=2160 format=nv12 framerate=60/1 ! omxh264enc
target-bitrate=1000 control-rate=constant num-slices=8 gop-mode=basic
goplength=
60 prefetch-buffer=TRUE skip-frame=true max-consecutive-skip=5 ! video/x-
h264,
profile=high, alignment=au ! filesink location=4k60_nv12.avc
```

```
gst-launch-1.0 filesrc location=/mnt/gst_src/FIFA_4Kp60_nv12.yuv !
videoparse width=3840 height=2160 format=nv12 framerate=60/1 ! omxh265enc
target-bitrate=1000 control-rate=constant num-slices=8 gop-mode=basic
goplength=
60 prefetch-buffer=TRUE skip-frame=true max-consecutive-skip=5 ! video/x-
h265,
profile=main, alignment=au ! filesink location=4k60_nv12.hevc
```

32-Streams Support

This feature checks functionality of encoding, decoding, and transcoding capacity at a time for 32 streams parallelly.

Table 84: **32-Streams Support**

Use-case	B_frames	Resolution	Format	Working Instances				
				Encode Only		Decode Only		Transcode
				CtrlSW[1]	Gstreamer[1]	CtrlSW	Gstreamer	Gstreamer[1]
Encode Only: AVC Decode Only: AVC Transcode: AVC → HEVC	Encode Only: 4 Decode Only: 4 Transcode: decode: 4 and encode: 3	640x480p30	NV12	32 inst → 14 fps (CPU: 100%) 23 inst → 32 fps (CPU: 10%)	32 inst → 11 fps (CPU: 98.6%) 24 inst → 30 fps (CPU: 28.7%)	32 inst → 39 fps (CPU: 23%)[2]	32 inst → 30 fps (CPU: 24.8%)[2]	32 inst → 14 fps (CPU: 98%) 23 inst → 30 fps (CPU: 26%)
			NV16	32 inst → 13 fps (CPU: 100%) 23 inst → 32 fps (CPU: 12%)	32 inst → 11 fps (CPU: 99.0%) 24 inst → 30 fps (CPU: 36.6%)	32 inst → 28 fps (CPU: 26%) 29 inst → 30 fps (CPU: 23%) [1]	32 inst → 30 fps (CPU: 25.3%)[2]	32 inst → 14 fps (CPU: 98%) 23 inst → 30 fps (CPU: 26%)
			XV15	32 inst → 13 fps (CPU: 100%) 23 inst → 32 fps (CPU: 14%)	32 inst → 11 fps (CPU: 98.7%) 24 inst → 30 fps (CPU: 36.2%)	32 inst → 31 fps (CPU: 25%)[2]	32 inst → 30 fps (CPU: 24.5%)[2]	32 inst → 14 fps (CPU: 98%) 23 inst → 30 fps (CPU: 27%)
			XV20	32 inst → 14 fps (CPU: 100%) 23 inst → 32 fps (CPU: 32%)	32 inst → 11 fps (CPU: 98.6%) 24 inst → 30 fps (CPU: 49.8%)	32 inst → 22 fps (CPU: 23%) 23 inst → 30 fps (CPU: 35%) [1]	32 inst → 17 fps (CPU: 25.7%) 30 inst → 30 fps (CPU: 22.5%) [1]	32 inst → 14 fps (CPU: 98%) 22 inst → 30 fps (CPU: 24%)
		720x480p30	NV12	32 inst → 14 fps (CPU: 100%) 22 inst → 32 fps (CPU: 20%)	32 inst → 12 fps (CPU: 98.5%) 23 inst → 30 fps (CPU: 29.4%)	32 inst → 36 fps (CPU: 27%)[2]	32 inst → 30 fps (CPU: 24.5%)[2]	32 inst → 14 fps (CPU: 98%) 22 inst → 30 fps (CPU: 26%)
			NV16	32 inst → 14 fps (CPU: 100%) 22 inst → 32 fps (CPU: 14%)	32 inst → 11 fps (CPU: 98.8%) 21 inst → 30 fps (CPU: 39.5%)	32 inst → 26 fps (CPU: 28%) 29 inst → 30 fps (CPU: 32%) [1]	32 inst → 17 fps (CPU: 25.0%) 30 inst → 30 fps (CPU: 23.0%) [1]	32 inst → 14 fps (CPU: 98%) 22 inst → 30 fps (CPU: 28%)
			XV15	32 inst → 14 fps (CPU: 100%) 23 inst → 32 fps (CPU: 13%)	32 inst → 11 fps (CPU: 98.6%) 21 inst → 30 fps (CPU: 35.4%)	32 inst → 30 fps (CPU: 30%)[2]	32 inst → 30 fps (CPU: 23.9%)[2]	32 inst → 14 fps (CPU: 98%) 23 inst → 30 fps (CPU: 27%)
			XV20	32 inst → 14 fps (CPU: 100%) 22 inst → 32 fps (CPU: 16%)	32 inst → 11 fps (CPU: 98.3%) 21 inst → 30 fps (CPU: 46.1%)	32 inst → 20 fps (CPU: 34%) 20 inst → 31 fps (CPU: 35%) [1]	32 inst → 17 fps (CPU: 25.8%) 28 inst → 30 fps (CPU: 23.3%) [1]	32 inst → 14 fps (CPU: 98%) 23 inst → 30 fps (CPU: 26%)

Table 84: **32-Streams Support** *(cont'd)*

Use-case	B_frames	Resolution	Format	Working Instances				
				Encode Only		Decode Only		Transcode
				CtrlSW[1]	Gstreamer[1]	CtrlSW	Gstreamer	Gstreamer[1]
Encode Only: HEVC Decode Only: HEVC Transcode: HEVC → AVC	Encode Only: 4 Decode Only: 4 Transcode: decode: 4 & encode: 3	640x480p30	NV12	32 inst → 14 fps (CPU: 100%) 22 inst → 32 fps (CPU: 10%)	32 inst → 12 fps (CPU: 98.9%) 23 inst → 30 fps (CPU: 27.2%)	32 inst → 47 fps (CPU: 28%)[2]	32 inst → 30 fps (CPU: 24.9%)[2]	32 inst → 13 fps (CPU: 98%) 22 inst → 30 fps (CPU: 36%)
			NV16	32 inst → 14 fps (CPU: 100%) 22 inst → 32 fps (CPU: 11%)	32 inst → 12 fps (CPU: 98.5%) 23 inst → 30 fps (CPU: 35.0%)	32 inst → 34 fps (CPU: 29%)[2]	32 inst → 30 fps (CPU: 24.2%)[2]	32 inst → 13 fps (CPU: 98%) 22 inst → 30 fps (CPU: 36%)
			XV15	32 inst → 14 fps (CPU: 100%) 22 inst → 32 fps (CPU: 25%)	32 inst → 12 fps (CPU: 98.5%) 23 inst → 30 fps (CPU: 35.1%)	32 inst → 40 fps (CPU: 36%)[2]	32 inst → 30 fps (CPU: 25.8%)[2]	32 inst → 13 fps (CPU: 98%) 23 inst → 30 fps (CPU: 42%)
			XV20	32 inst → 14 fps (CPU: 100%) 22 inst → 32 fps (CPU: 32%)	32 inst → 12 fps (CPU: 98.5%) 23 inst →30 fps (CPU: 48.4%)	32 inst → 27 fps (CPU: 29%) 29 inst → 30 fps (CPU: 33%) [1]	32 inst → 30 fps (CPU: 23.9%)[2]	32 inst → 13 fps (CPU: 98%) 23 inst → 30 fps (CPU: 49%)
		720x480p30	NV12	32 inst → 13 fps (CPU: 100%) 22 inst → 32 fps (CPU: 20%)	32 inst → 12 fps (CPU: 98.7%) 23 inst → 30 fps (CPU: 31.6%)	32 inst → 43 fps (CPU: 27%)[2]	32 inst → 30 fps (CPU: 24.9%)[2]	32 inst → 13 fps (CPU: 98%) 22 inst → 30 fps (CPU: 39%)
			NV16	32 inst → 13 fps (CPU: 100%) 22 inst → 32 fps (CPU: 26%)	32 inst → 12 fps (CPU: 98.5%) 23 inst → 30 fps (CPU: 39.7%)	32 inst → 31 fps (CPU: 27%)[2]	32 inst → 30 fps (CPU: 23.6%)[2]	32 inst → 13 fps (CPU: 98%) 22 inst → 30 fps (CPU: 41%)
			XV15	32 inst → 13 fps (CPU: %) 22 inst → 31 fps (CPU: 14%)	32 inst → 12 fps (CPU: 99.0%) 23 inst → 30 fps (CPU: 38.4%)	32 inst → 37 fps (CPU: 34%)[2]	32 inst → 30 fps (CPU: 25.4%)[2]	32 inst → 13 fps (CPU: 98%) 23 inst → 30 fps (CPU: 40%)
			XV20	32 inst → 13 fps (CPU: 100%) 22 inst → 31 fps (CPU: 19%)	32 inst → 12 fps (CPU: 98.8%) 22 inst → 30 fps (CPU: 48.5%)	32 inst → 25 fps (CPU: 36%) 27 inst → 30 fps (CPU: 35%) [1]	32 inst → 30 fps (CPU: 25.2%)[2]	32 inst → 13 fps (CPU: 98%) 23 inst → 30 fps (CPU: 42%)

Notes:

1. Achieved the correct fps.
2. Close to the correct fps.

Resolution: 720x480, 640x480, 720x576

This is the benchmarking for 32 streams using ctrlsw and gstreamer which involves validating the fps and CPU usage for different resolutions with different input formats - NV12, NV16, XV15, XV20.

***Note*:** Encode-only tests are done with file source input and expected to have < 30 fps due to raw buffer copy.

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 hardware section.

Table 85: **Format : XV20 , Resolution : 4096x2160p60 , Num-Slice : 8**

Standard	Rate-Control	GOP	GOP-length	Num-bframes	Max Bitrate (Mb/s)	Total CPU (4 cores) (%)
HVEC	CBR	I Only	0	N/A	341	35%
	CBR	IBBBBP	60	4	145.6	20%
	Low Latency	I Only	0	N/A	361	37%
AVC	CBR	I Only	0	NA	235	30%
	CBR	IBBBBP	60	4	77	16%
	Low Latency	I Only	0	N/A	251	43%

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.

Frame Delta-QP Option Based on temporal-ID

Encoder rate control assigns the QP based on rate-control algorithm type and target-rate. You can choose an extra delta-QP based on its temporal-ID on top of the rate-control generated QPs. This enables you to increase or decrease the picture quality based on the temporal layers IDs. As you go higher up the layers, it is better to have a higher average QP to maintain better subjective quality. Example: If Layer0 - QPs = "X", and Layer1 is "X + 2", and Layer2 is "X+4," then deltaQPs should be set as 2 and 4 accordingly for Layer1 and Layer2.

Not Supported: RateCtrlMode=CONST_QP is not supported for delta_QP

To enable delta-QP option, enable the following parameter in the cfg file.

```
Gop.TempDQP         = 1 1 1 1
```

LOAD_LDA Based on Temporal ID

Similarly, having different base frame QPs depending on the temporal layer, you can choose different a Lambda table for different layers. This improves the subjective quality depending on video content. To enable a temporal ID, enable the following parameter in the cfg file and copy `Lambdas.hex` file in the working directory. Refer to the 5.1.3 Lambda File Format

```
LambdaFactors = 0.20 0.35 0.59 0.60 0.61 0.62
```

Intra MB forcing using External QP-Table/ LOAD_QP Mode

This forces specific blocks to be encoded in intra prediction mode. The `bool AL_Encoder_Process (AL_HEncoder hEnc, AL_TBuffer* pFrame, AL_TBuffer* pQpTable)` API forces Intra at block level and can be controlled with the QP table in the API. 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
- The 6 LSBs of each byte are used for QP: absolute QP in [0;51] or relative QP in [-32;31]
- The 2 MSBs are reserved for the prediction mode, as shown in the following table.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Prediction mode		QP					

Table 86: **Prediction Mode and Description**

Prediction mode	Description
0	Encoder free choice
1	Force Intra Prediction mode
2	Reserved
3	Reserved

Constraints

- To use force intra at block level, external QP tables must be enabled. Then, a QP table must be provided for each frame.
- This feature is supported with RELATIVE_QP as well; Prediction mode and QP bit distribution remains the same as above.

- This feature is also support with ROI_QP mode. The entire ROI region can be encoded with Intra MBs using the following quality option for the region:

```
<quality> = INTRA_QUALITY
```

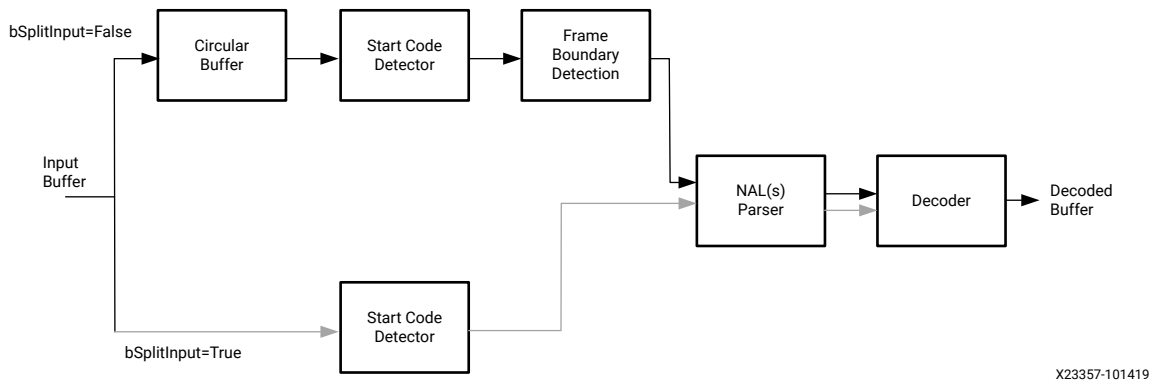
Decoder Meta-data Transfer Using 1-to-1 Relation Between Input and Output Buffer

Until now, 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.

Decoder config setting bSplitInput is added to enable and disable this feature at control-sw. gst-omx decoder supports also this new parameter. When split-input is enabled, the decoder has a 1-to-1 mapping for input and output buffers. When disabled, the decoder copies all input buffers to internal circular buffer and processes them.

Constraints: When split-input mode is enabled for decoder, input to decoder should be aligned with the frame (AU) or slice (NAL unit) boundary. Part of AU or NAL is not allowed.

Figure 46: **Decoder Input**



DEFAULT_GOP_B and PYRAMIDAL_GOP_B GOP Control Modes

Two new GOP control modes, Control software and Gstreamer framework are supported

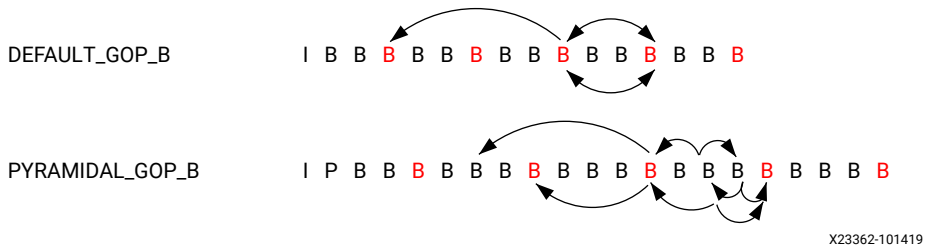
Control_SW

DEFAULT_GOP_B & PYRAMIDAL_GOP_B: Patterns are identical to DEFAULT_GOP and PYRAMIDAL_GOP except that P frames are replaced with B Pictures.

Gstreamer

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.

Figure 47: **DEFAULT_GOP_B and PYRAMIDAL_GOP_B patterns**



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.

Control-SW API:

In command file, define the settings "LF.BetaOffset" and “LF.TcOffset”. Encode Library APIs are (more details can be found in API description section)

```
bool AL_Encoder_SetLoopFilterBetaOffset(AL_HEncoder hEnc, int8_t
iBetaOffset);
bool AL_Encoder_SetLoopFilterTcOffset(AL_HEncoder hEnc, int8_t iTcOffset);
```

Gstreamer

Omxh264enc/omxh265enc elements support below two mutable parameters, values can be modified during run time.

1. loop-filter-beta-offset: Beta offset for the deblocking filter; used only when loop-filter-mode is enabled.
2. 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.

LOAD_QP Support at Gstreamer

Along with each frame, it is possible to associate a QP Table specifying, for each encoding block of the frame, the QP you want to use. The QP value can be relative or absolute. The QP table buffer must contain a byte per MB/CTB, in raster scan format:

1. In AVC, one byte per 16x16 MB
2. In HEVC, one byte per 32x32 CTB

Pipeline:

```
zynqmp_vcu_encode -w 3840 -h 2160 -e hevc -f 30 -c 2 -g 30 -b 1 -o op.h265 -
q 3 -i test_UHD.yuv -d LOADQP:0:QPs.hex
```

DMA_PROXY Module Usage

Enable/Disable dmaproxy Module

dmaproxy is a loaded module and there is no gstreamer parameter to control this. In the PetaLinux BSP, it is auto-loaded on bootup and therefore, omxh264enc/omxh265enc uses dmaproxy. If you want to remove the dmaproxy functionality, use rmmod dmaproxy.

The encoder OMX component uses dmaproxy APIs for data transfer while reconstructing the complete encoded output frame. The dmaproxy internally uses Xilinx ZDMA APIs via the standard Linux kernel DMA framework for data transfer.

If dmaproxy is not loaded, then the encoder OMX component uses a CPU copy during the reconstruction of complete encoded o/p frame.

Multi-stream Support in dmaproxy Module

dmaproxy uses zynqmp-dma channels. So, the number of multi-stream supported by dmaproxy depends on the available free zynqmp-dma channel in that design/build. Usually 8/16 zynqmp-dma channels are available in ZCU106.

Further Optimization with dmaproxy Usage

By default, all interrupts are served by cpu0 so that dmaproxy is used in the most optimized way (mostly in multi-stream use cases). It is recommended that you distribute zynqmp-dma interrupts across all CPUs. To do so, follow these steps:

1. Find the zynqmp-dma interrupt used by dmaproxy in cat `/proc/interrupts`.
2. Change its affinity value to move it to a different CPU code.

```
echo X > /proc/irq/<irq number>/smp_affinity
```

XAVC

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

Constraints: You need to provide the information required to create XAVC bitstream following the specifications. For instance, you must provide at least the resolution, framerate, clock ratio, etc

You have to increase `AL_ENC_MAX_SEI_SIZE` to 10 kb (`include/lib_common/StreamBuffer.h`) in order to support XAVC INTRA CBG.

IMPORTANT! *The default size of AL_ENC_MAX_SEI_SIZE is 10 kb from 2020.1 onwards.*

***Note*:** Currently the VCU encoder does not support all XAVC features. A list of unsupported XAVC features are as follows:

- The VCU encoder does not support Class 480 4k Intra Profile.
- The VCU encoder does not support XAVC interlaced.
- The VCU encoder will only produce h264 bitstreams. It is up to user applications to wrap this within mp4 or mxf. containers

Other than the above restrictions, all VCU encoder XAVC bitstreams have been compliance verified using the Sony provided 'AvcChecker_XAVC_v12.exe'.

CtrlSW Profile

Following are the new profiles added:

- AL_PROFILE_XAVC_HIGH10_INTRA_CBG
- AL_PROFILE_XAVC_HIGH10_INTRA_VBR
- AL_PROFILE_XAVC_HIGH_422_INTRA_CBG
- AL_PROFILE_XAVC_HIGH_422_INTRA_VBR
- AL_PROFILE_XAVC_LONG_GOP_MAIN_MP4
- AL_PROFILE_XAVC_LONG_GOP_HIGH_MP4
- AL_PROFILE_XAVC_LONG_GOP_HIGH_MXF
- AL_PROFILE_XAVC_LONG_GOP_HIGH_422_MXF

OMX Profile

New OMX_ALG_VIDEO_AVCPROFILETYPE:

- OMX_ALG_VIDEO_XAVCProfileHigh10_Intra_CBG
- OMX_ALG_VIDEO_XAVCProfileHigh10_Intra_VBR
- OMX_ALG_VIDEO_XAVCProfileHigh422_Intra_CBG
- OMX_ALG_VIDEO_XAVCProfileHigh422_Intra_VBR
- OMX_ALG_VIDEO_XAVCProfileLongGopMain_MP4
- OMX_ALG_VIDEO_XAVCProfileLongGopHigh_MP4
- OMX_ALG_VIDEO_XAVCProfileLongGopHigh_MXF
- OMX_ALG_VIDEO_XAVCProfileLongGopHigh422_MXF

Supported Profiles for XAVC Encoder

Table 87: **Supported Profiles for XAVC Encoder**

Profile	Format	Resolution	Gop Length
xavc-high-10-intra-cbg	XV15	1080p	1
xavc-high-4:2:2-intra-cbg	XV20	4K,1080p	1
xavc-high-4:2:2-intra-vbr	XV20	4K,1080p	1
xavc-long-gop-main-mp4	NV12	1080p	1 sec (30/60)
xavc-long-gop-high-mp4	NV12	1080p	1 sec (30/60)

Table 87: **Supported Profiles for XAVC Encoder** *(cont'd)*

Profile	Format	Resolution	Gop Length
xavc-long-gop-high-mxf	NV12	1080p	1 sec (30/60)
xavc-long-gop-high-4:2:2-mxf	XV20	4K,1080p	1 sec (30/60)

Supported Profiles for XAVC Decoder

Table 88: **Supported Profiles for XAVC Decoder**

Serial Number	Profile	Codec	Resolution	Bit-depth	Result
1	High 4:2:2 Intra	AVC	1920x1080	10	Supported
			1280x720		
			3840x2160		
			4096x2160		
			2048x1080		
			interlaced stream: mb_adaptive_frame_field_flag == 1 : NOT SUPPORTED		
2	4:2:2@High	AVC	1920x1080	10	Supported
			1280x720		
			interlaced stream: mb_adaptive_frame_field_flag == 1 : NOT SUPPORTED		
3	High	AVC	3840x2160	8	Supported
4	High 10 Intra	AVC	interlaced stream: mb_adaptive_frame_field_flag == 1 : NOT SUPPORTED		

XAVC Examples

Gstreamer Pipelines

```
gst-launch-1.0 v4l2src io-mode=4 device=/dev/video0 ! video/x-raw,
width=3840, height=2160, format=NV16_10LE32, framerate=60/1 ! omxh264enc
control-rate=variable target-bitrate=200000 gop-mode=basic gop-length=1 b-
frames=0 prefetch-buffer=TRUE entropy-mode=CABAC num-slices=8 max-picture-
sizes='<3333,0,0>' ! video/x-h264 , profile=xavc-high-4:2:2-intra-
cbg ,alignment=nal ! filesink location=/run/costarica_crvar_xv20.xavc
```

```
gst-launch-1.0 v4l2src io-mode=4 device=/dev/video0 ! video/x-raw,
width=3840, height=2160, format=NV16_10LE32, framerate=60/1 ! omxh264enc
control-rate=constant target-bitrate=90000 gop-mode=basic gop-length=60 b-
frames=0 prefetch-buffer=TRUE entropy-mode=CABAC num-slices=8 max-picture-
sizes='<4500,1500,0>' ! video/x-h264, profile=xavc-long-gop-high-4:2:2-mxf,
alignment=nal ! filesink location=/run/costarica_crconst_xv20.xavc
```

***Note*:** Use the following calculations for max-picture-sizes value:

- For IPPPP mode, use " max-picture-sizes <3*target_bitrate_Kbps/fps, target_bitrate_Kbps/fps, 0>

  For example: 90 Mb/s, 60 fps, use max-picture-sizes=<4500, 1500, 0>
- For I only mode, use " max-picture-sizes <target_bitrate_Kbps/fps, 0, 0>

  For example: 90 Mb/s, 60 fps, use max-picture-sizes=<1500, 0, 0>

Control Software

```
ctrlsw_encoder -cfg Common_cavlc.cfg -i FoodMarket_4Kp60_XV20.yuv --input-
format
XV20 --input-width 3840 --input-height 2160 --profile
XAVC_HIGH_422_INTRA_VBR
--chroma-mode CHROMA_4_2_2 --ip-bitdepth 10 --num-slices 8 --gop-mode
DEFAULT_GOP
--gop-length 1 --gop-numB 0 --framerate 60 --bitrate 200000 --max-bitrate
200000 -o
/run/TC_065_cvbr_ctrlsw_latest.xavc --ratectrl-mode CBR
```

```
ctrlsw_encoder -cfg Common.cfg -i Football_4Kp30_XV20.yuv --input-format
XV20
--input-width 3840 --input-height 2160 --profile XAVC_HIGH_422_INTRA_VBR
--chroma-mode CHROMA_4_2_2 --ip-bitdepth 10 --num-slices 8 --gop-mode
DEFAULT_GOP
--gop-length 1 --gop-numB 0 --framerate 30 --bitrate 60000 --max-bitrate
75000 -o /
run/TC_198_cvbr_ctrlsw.xavc --ratectrl-mode VBR
```

HDR10

The VCU supports HDR10 encoding and decoding. To comply with HDR standards, the VCU supports HDR10 colour primaries, transfer characteristics, and matrix coefficients in the sequence parameter set of intra-encoded frames. Additionally, you can insert and extract Mastering Display Color Volume and Content Light Level supplemental enhancement information packets.

Gstreamer Pipelines

HDR10 video can be captured using the v4l2src element (assuming the necessary EDID and design changes have been made). v4l2src automatically detects an HDR10 source and transmits the metadata and colorimetry. Currently, HDR10 capture and display is only supported with the Xilinx HDMI IPs. An example serial pipeline:

```
gst-launch-1.0 v4l2src io-mode=4 device=/dev/video0 ! video/x-raw,
width=3840, height=2160, format=NV12_10LE32, framerate=60/1 ! omxh265enc
target-bitrate=20000 filler-data=0 prefetch-buffer=TRUE num-slices=8 !
queue max-size-buffers=0 ! omxh265dec ! queue max-size-bytes=0 !
fpsdisplaysink name=fpssink text-overlay=false 'video-sink=kmssink bus-
id=a00c0000.v_mix plane-id=34 hold-extra-sample=1 show-preroll=false
sync=true fullscreen-overlay=1' sync=true -v
```

Gstreamer parsers also support HDR10 parsing. An example transcode pipeline:

```
gst-launch-1.0 filesrc location="HDR.hevc" ! h265parse ! omxh265dec ! queue
max-size-bytes=0 ! omxh265enc control-rate=constant target-bitrate=25000
prefetch-buffer=TRUE num-slices=8 ! fpsdisplaysink name=fpssink text-
overlay=false 'video-sink=filesink location=transcoded_HDR.hevc' -v
```

Control Software

See HDR10 File Format.

HLG Support

Hybrid Log-Gamma (HLG) is a HDR standard aimed at providing backwards compatibility with non-HDR monitors, while also achieving a wider color gamut for HDR monitors. As HLG EOTF is similar to standard gamma functions, HLG streams look proper when played on SDR monitors. Meanwhile, because the gamma and PQ curves are completely different, the PQ EOTF streams are displayed properly and usually look washed out when played on a SDR monitor. Unlike HDR10, HLG does not require any extra metadata that goes along with the video data. HLG streams consist of the HLG transfer characteristics/EOTF and BT2020 color primaries and BT2020 color matrix.

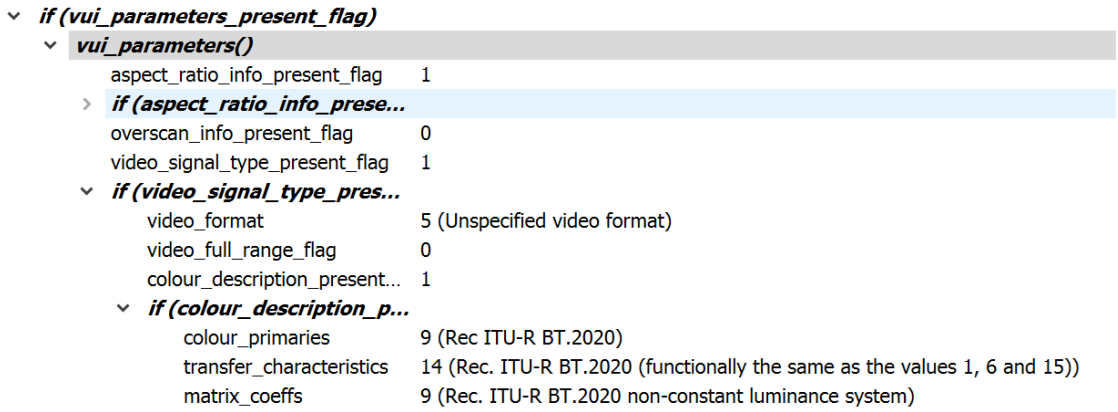
From VCU point of view, there are two types of HLG that can be enabled.

- There is a backwards compatible mode, that uses the BT2020 value in the SPS VUI parameters, instead of the HLG transfer characteristics. Then the VCU encoder inserts a *Alternative Transfer Characteristics* (ATC) SEI with the HLG value.
- There is a HLG only mode. This directly uses the HLG value in the SPS VUI parameters.

Table 89: **Format and Parameters**

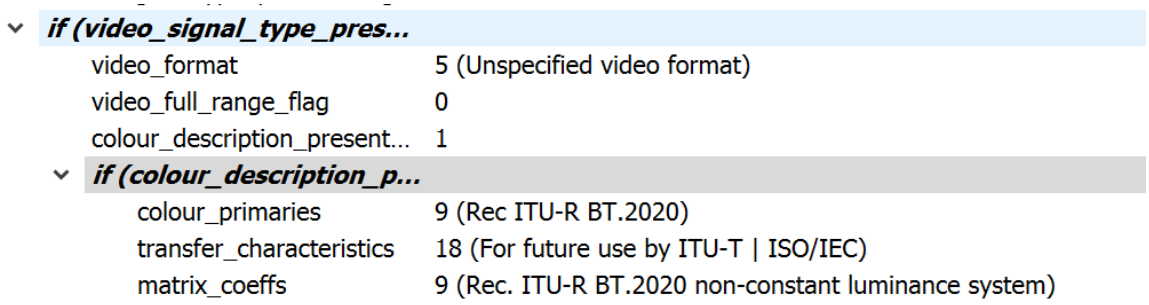
Format	VUI			SEI
	Color Primaries	Transfer Characteristics	Matrix Coefficients	
UHD-TV HDR HLG10	9 (BT.2020)	14 (Rec.2020)	9 (BT.2020)	preferred_transfer_characteristics=18
UHD-TV HDR HLG10	9 (BT.2020)	18 (HLG)	9 (BT.2020)	None

Sample VUI parameters in encoded bitstream for Backward Compatible mode are as follows:



Along with the preceding VUI `colour_description_parameter`, the encoder embeds SEI message (ATC), which indicates that `preferred_transfer_characteristics` is set to 18.

HLG Only mode VUI is as follows:



Sample Gstreamer Pipelines

Gstreamer v4l2src capture element sets corresponding caps information (color-matrix, transfer-function ...etc) if corresponding SDI_Rx is capable for HLG capture, and then gst-omx encoder and parser plug-ins are updated to handle these caps to support HLG.

HLG Only (HLG EOTF) Mode, 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_enco
ding=1"
show-preroll-frame=false sync=true'
```

Backwards Compatible (SDR EOTF + HLG SEI) 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 hlg-sdr-compatible=TRUE 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_enco
ding=1"
 show-preroll-frame=false sync=true'
```

Control-SW Application

Set the following encoder configuration parameters for HLG support.

Section [SETTINGS]:

- TransferCharac = TRANSFER_BT_2100_HLG
- ColourMatrix = COLOUR_MAT_BT_2100_YCBCR
- ColourDescription=COLOUR_DESC_BT_2020
- EnableSEI = SEI_ATC

GRAY8/GRAY10 Support

VCU encoder and decoder supports encoding and decoding of GRAY8 and GRAY10 format support at gstreamer level from release 2021.1 onwards. Sample serial (encode + decode) test case pipelines are as follows:

```
gst-launch-1.0 -v videotestsrc ! video/x-raw, width=1920, height=1080,
format=GRAY8, framerate=30/1 ! queue ! omxh265enc target-bitrate=6250
num-slices=8 control-rate=low-latency ! video/x-h265, alignment=au !
queue max-size-buffers=0 ! omxh265dec ! queue max-size-bytes=0 !
fpsdisplaysink name=fpssink text-overlay=false video-sink="kmssink
bus-id="fd4a0000.display" fullscreen-overlay=1" -v
```

```
gst-launch-1.0 -v videotestsrc ! video/x-raw, width=1920, height=1080,
format=GRAY10_LE32, framerate=30/1 ! queue ! omxh265enc target-
bitrate=6250 num-slices=8 control-rate=low-latency ! video/x-h265,
alignment=au ! queue max-size-buffers=0 ! omxh265dec ! queue max-size-
bytes=0 ! fpsdisplaysink name=fpssink text-overlay=false video-
sink="kmssink bus-id="fd4a0000.display" fullscreen-overlay=1" -v
```

Decoder Output Buffer Vertical Alignment

This feature is only from 2021.1 onwards.

VCU decoder output buffer vertical alignment of active video can be adjusted provided that actual buffer height is sufficient enough for adjustment.

Example, If Decoder gets 720x240 resolution video as input, then actual output video buffer resolution will be 736x256 (considering 32x alignment). A new configuration parameter called "output-position=<x,y>" is added to the decoder from 2021.1 onwards,.The x and y parameters decide the position of active video data in the buffer.

If output-position is set to <0,1> , that means the top-line in the video buffer will be skipped and active video data will be written from 2nd line onwards.

Sample control-sw and gstreamer pipelines: are as follows:

```
gst-launch-1.0 filesrc location=test_input.avc ! h264parse !
omxh264dec output-position="<0,2>" ! filesink
location=decoded_2lines.yuv
```

```
ctrlsw-decoder -in test_input.avc -avc -o test_output.yuv --output-
position 0,2
```

GStreamer Pipelines

Examples of running GStreamer from the PetaLinux command line are as follows. To see the description of gstreamer elements and properties used in each of them, use the `gst-inspect-1.0` command.

For example, to get description of each parameters for "omxh264dec" element, enter the following at the command prompt:

```
gst-inspect-1.0 omxh264dec
```

H.264 Decoding

Decode H.264 based input file and display it over the monitor connected to the display port

```
gst-launch-1.0 filesrc location="input-file.mp4" ! qtdemux name=demux
demux.video_0 ! h264parse ! omxh264dec ! queue max-size-bytes=0 ! kmssink
bus-id=fd4a0000.display fullscreen-overlay=1
```

H.265 Decoding

Decode H.265 based input file and display it over the monitor connected to the display port

```
gst-launch-1.0 filesrc location="input-file.mp4" ! qtdemux name=demux
demux.video_0 ! h265parse ! omxh265dec ! queue max-size-bytes=0 ! kmssink
bus-id=fd4a0000.display fullscreen-overlay=1
```

***Note*:** Input-file.mp4 can be of any of the following formats:

- 4:2:0 8-bit
- 4:2:2 8-bit
- 4:2:0 10-bit
- 4:2:2 10-bit

High Bitrate Bitstream Decoding

To reduce frame decoding time for bitstreams greater than 100 Mb/s at 4kP30, use the following options:

- Increase internal decoder buffers (internal-entropy-buffers parameter) to 9 or 10.
- Add a queue at the decoder input side

The following command decodes an H.264 MP4 file using an increased number of internal entropy buffers and displays it via DisplayPort.

```
gst-launch-1.0 filesrc location="input-file.mp4" ! qtdemux name=demux
demux.video_0 ! h264parse ! queue max-size-bytes=0 ! omxh264dec internal-
entropy-buffers=10 ! queue max-size-bytes=0 ! kmssink bus-
id=fd4a0000.display fullscreen-overlay=1
```

H.264 Encoding

Encode input 3840×2160 4:2:0 8-bit (NV12) YUV file to H.264.

```
gst-launch-1.0 filesrc location="input-file.yuv" ! rawvideoparse
format=nv12 width=3840 height=2160 framerate=30/1 ! omxh264enc ! filesink
location="output.h264"
```

Encoding 3840×2160 4:2:2 8-bit (NV16) YUV file to H.264.

```
gst-launch-1.0 filesrc location="input-file.yuv" ! rawvideoparse
format=nv16 width=3840 height=2160 framerate=30/1 ! omxh264enc ! filesink
location="output.h264"
```

Encoding 3840×2160 4:2:0 10-bit (NV12_10LE32) YUV file to H.264.

```
gst-launch-1.0 filesrc location="input-file.yuv" ! rawvideoparse
format=nv12-10le32 width=3840 height=2160 framerate=30/1 ! omxh264enc !
filesink location="output.h264"
```

Encoding 3840×2160 4:2:2 10-bit (NV16_10LE32) YUV file to H.264.

```
gst-launch-1.0 filesrc location="input-file.yuv" ! rawvideoparse
format=nv16-10le32 width=3840 height=2160 framerate=30/1 ! omxh264enc !
filesink location="output.h264"
```

H.265 Encoding

Encoding 3840×2160 4:2:0 8-bit (NV12) YUV file to H.265.

```
gst-launch-1.0 filesrc location="input-file.yuv" ! rawvideoparse
format=nv12 width=3840 height=2160 framerate=30/1 ! omxh265enc ! filesink
location="output.h265"
```

Encoding 3840×2160 4:2:2 8-bit (NV16) YUV file to H.265.

```
gst-launch-1.0 filesrc location="input-file.yuv" ! rawvideoparse
format=nv16 width=3840 height=2160 framerate=30/1 ! omxh265enc ! filesink
location="output.h265"
```

Encoding 3840×2160 4:2:0 10-bit (NV12_10LE32) YUV file to H.265.

```
gst-launch-1.0 filesrc location="input-file.yuv" ! rawvideoparse
format=nv12-10le32 width=3840 height=2160 framerate=30/1 ! omxh265enc !
filesink location="output.h265"
```

Encoding 3840×2160 4:2:2-10-bit (NV16_10LE32) YUV file to H.265.

```
gst-launch-1.0 filesrc location="input-file.yuv" ! rawvideoparse
format=nv16-10le32 width=3840 height=2160 framerate=30/1 ! omxh265enc !
filesink location="output.h265"
```

***Note*:** The command lines above assume the file `input-file.yuv` is in the format specified.

Transcode from H.264 to H.265

Convert 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"
```

Transcode from H.265 to H.264

Convert H.265 based input container format file into H.264 format

```
gst-launch-1.0 filesrc location="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="output.h264"
```

Multistream Encoding Decoding Simultaneously

Encode and decode the input YUV into two streams by using two different encoder and two different decoder elements.

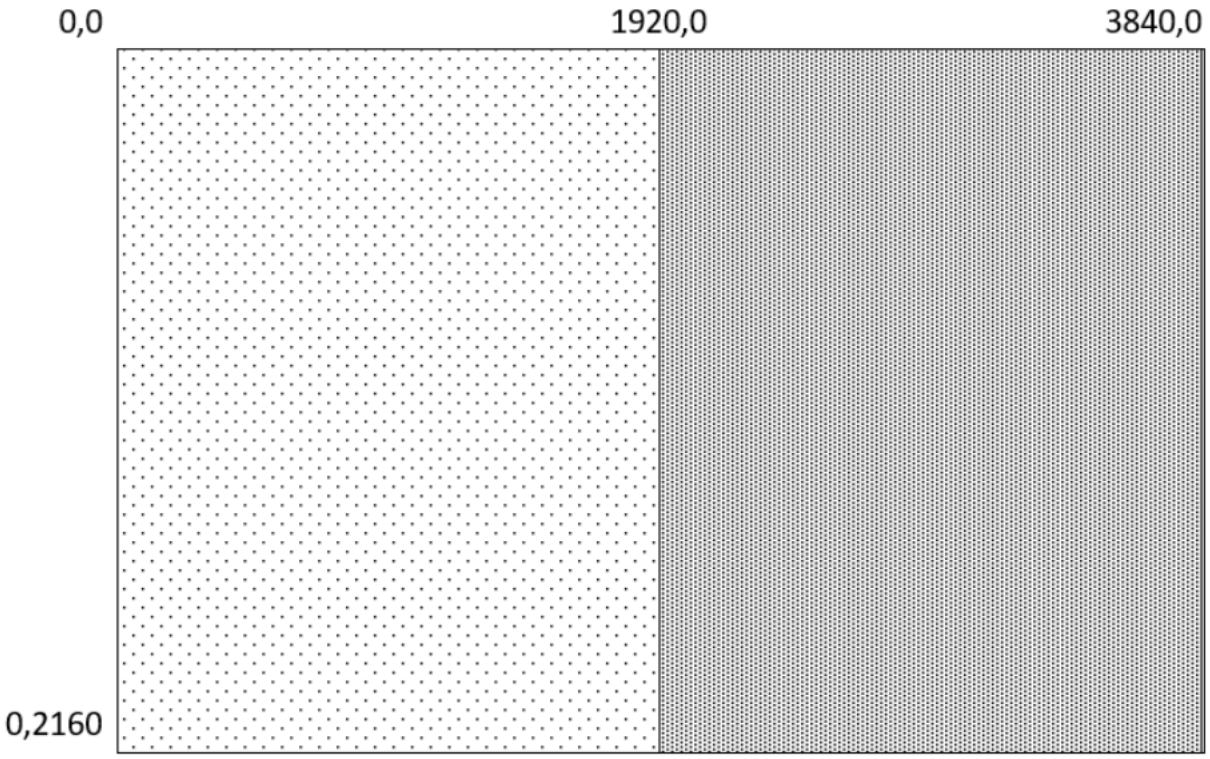
- V4l2 → AVC encode → AVC Decode → HEVC encode → fakesink (AVC and HEVC encoding in single pipeline)

```
gst-launch-1.0 -v v4l2src device=/dev/video0 io-mode=4 num-buffers=300 !
video/x-raw, width=3840, height=2160, format=NV12, framerate=30/1 ! tee
name=t ! queue max-size-bytes=0 ! omxh264enc target-bitrate=25000 control-
rate=constant num-slices=8 gop-mode=basic gop-length=30 prefetch-
buffer=TRUE ! video/x-h264, profile=high, alignment=au ! queue !
omxh264dec ! queue max-size-bytes=0 ! omxh265enc target-bitrate=25000
control-rate=constant num-slices=8 gop-mode=basic gop-length=30 prefetch-
buffer=TRUE ! video/x-h265, profile=main, alignment=au ! queue max-size-
bytes=0 ! fpsdisplaysink name=fpssink text-overlay=false video-
sink="fakesink sync=true" sync=true -v &
```

- V4L2 → AVC encode → AVC decode → kmssink V4l2 → HEVC encode → HEVC decode → kmssink (AVC & HEVC encoding and decoding in single pipeline)

```
gst-launch-1.0 -v v4l2src device=/dev/video0 io-mode=4 num-buffers=800 !
video/x-raw, width=3840, height=2160, format=NV12, framerate=30/1 !
omxh264enc target-bitrate=25000 control-rate=constant num-slices=8 gop-
mode=basic gop-length=30 prefetch-buffer=TRUE ! video/x-h264,
profile=high, alignment=au ! queue ! omxh264dec ! queue max-size-
bytes=0 ! fpsdisplaysink name=fpssink1 text-overlay=false video-
sink="kmssink bus-id="a0070000.v_mix" hold-extra-sample=TRUE plane-id=33
render-rectangle=<0,0,1920,2160>" sync=true v4l2src device=/dev/video4 io-
mode=4 num-buffers=800 ! video/x-raw, width=3840, height=2160,
format=NV12, framerate=30/1 ! omxh265enc target-bitrate=25000 control-
rate=constant num-slices=8 gop-mode=basic gop-length=30 prefetch-
buffer=TRUE ! video/x-h265, profile=main, alignment=au ! queue !
omxh265dec ! queue max-size-bytes=0 ! fpsdisplaysink name=fpssink2 text-
overlay=false video-sink="kmssink bus-id="a0070000.v_mix" hold-extra-
sample=TRUE plane-id=34 render-rectangle=<1920,0,1920,2160>" sync=true -v
&
```

In the above gstreamer pipeline, the Xilinx mixer IP is used to overlay plane 33 and 34 onto the primary display plane. Plane 33 is displayed at 0,0 with a width of 1920 and height of 2160. Plane 34 is displayed at 1920,0 with a width of 1920 and height of 2160. The output resembles the following figure:



***Note*:** H.264 and H.265 encoding and decoding are possible simultaneously if the maximum bandwidth of all the streams does not over subscribe the supported bandwidth of the Zynq UltraScale+ MPSoC VCU and system configuration.

Multistream Decoding

Decode the H.265 input file using four decoder elements simultaneously and saving them to separate files

```
gst-launch-1.0 filesrc location=input_1920x1080.mp4 ! qtdemux ! h265parse !
tee name=t
t. ! queue ! omxh265dec ! queue max-size-bytes=0 ! filesink
location="output_0_1920x1080.yuv"
t. ! queue ! omxh265dec ! queue max-size-bytes=0 ! filesink
location="output_1_1920x1080.yuv"
t. ! queue ! omxh265dec ! queue max-size-bytes=0 ! filesink
location="output_2_1920x1080.yuv"
t. ! queue ! omxh265dec ! queue max-size-bytes=0 ! filesink
location="output_3_1920x1080.yuv"
```

***Note*:** The tee element is used to feed same input file into four decoder instances; you can use separate gst-launch-1.0 application to fed different inputs.

Multistream Encoding

Encode the input YUV file into eight streams by using eight encoder elements simultaneously.

```
gst-launch-1.0 filesrc location=input_nv12_1920x1080.yuv ! rawvideoparse
width=1920 height=1080 format=nv12 framerate=30/1 ! tee name=t
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_0.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_1.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_2.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_3.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_4.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_5.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_6.h264"
t. ! queue ! omxh264enc control-rate=2 target-bitrate=8000 ! video/x-h264,
profile=high ! filesink location="ouput_7.h264"
```

***Note*:** The tee element is used to feed the same input file into eight encoder instances. You can separate the gst-launch-1.0 application to be fed with different inputs.

For alternate input YUV formats, the following changes are required in the above pipeline:

Table 90: **Alternate Input YUV Formats**

Format/Profile	Arguments
4:2:2 8-bit (NV16)	format=nv16 profile=high-4:2:2
4:2:0 10-bit (NV12_10LE32)	format=nv12-10le32 profile=high-10
4:2:2 10-bit (NV16_10LE32)	format=nv16-10le32 profile=high-4:2:2
4:0:0 8-bit (GRAY8)	Not supported
4:0:0 10-bit (GRAY10_LE32)	Not supported

Transcoding and Streaming via Ethernet

```
gst-launch-1.0 filesrc location="test_0003.mp4" ! qtdemux ! h264parse !
omxh264dec !omxh265enc control-rate=2 target-bitrate=20000 ! h265parse !
queue ! rtph265pay !udpsink host=192.168.1.1 port=50000 buffer-
size=20000000 async=false max-lateness=-1 qos-dscp=60
```

***Note*:** 192.168.1.1 is an example client IP address. You might need to modify this with actual client IP address.

Streaming via Ethernet and Decoding to the Display Pipeline

```
gst-launch-1.0 udpsrc port=50000 caps="application/x-rtp, media=video,
clock-rate=90000, payload=96, encoding-name=H265" buffer-size=20000000 !
rtpjitterbuffer latency=1000 ! rtph265depay ! h265parse ! omxh265dec !
queue ! kmssink bus-id=fd4a0000.display fullscreen-overlay=1 sync=false
```

Recommended Settings for Streaming

The following are the recommended settings for streaming:

- Low-latency RC with low-delay-p gop-mode, gdr-mode=horizontal, periodicity-idr=Picture height in MBs
- Low-latency RC with low-delay-p gop-mode and periodicity-idr=twice the framerate.
- CBR RC with low-delay-p gop-mode, gdr-mode=horizontal, periodicity-idr=Picture height in MBs
- CBR RC with low-delay-p gop-mode and periodicity-idr=twice the framerate, cpb-size="1000”

For AVC, 1 MB=16x16 pixels and for HEVC, 1MB=32x32 pixels. You have to calculate picture height in Mbs= roundup(Height,64)/#Mb rows.

- If you are not using buffer-size property of `udpsrc`, then you must set it manually using `sysctl` command as per their network bandwidth utilization and requirements.

  ```
  -> sysctl -w net.core.rmem_default=60000000
  ```

- VBR is not a preferred mode of streaming.

Verified GStreamer Elements

Table 91: **Verified GStreamer Elements**

Element	Description
filesink	Writes incoming data to a file in the local file system
filesrc	Reads data from a file in the local file system
h264parse	Parses a H.264 encoded stream
h265parse	Parses a H.265 encoded stream
kmssink	Renders video frames directly in a plane of a DRM device

Table 91: **Verified GStreamer Elements** *(cont'd)*

Element	Description
omxh264dec	Decodes OpenMAX H.264 video
omxh264enc	Encodes OpenMAX H.264 video
omxh265dec	Decodes OpenMAX H.265 video
omxh265enc	Encodes OpenMAX H.265 video
qtdemux	Demuxes a .mov file into raw or compressed audio and/or video streams.
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
rtph264depay	Extracts an H.264 video payload from an RTP packet stream
rtph264pay	Encapsulates an H.264 video in an RTP packet stream
rtph265depay	Extracts an H.265 video payload from an RTP packet stream
rtph265pay	Encapsulates an H.265 video in an RTP packet stream
rtpjitterbuffer	Reorders and removes duplicate RTP packets as they are received from a network source
tee	Splits data to multiple pads
udpsink	Sinks UDP packets to the network
udpsrc	Reads UDP packets from the network
v4l2src	Captures video from v4l2 devices, like webcams and television tuner cards
rawvideoparse	Converts a byte stream into video frames

Verified Containers Using GStreamer

Table 92: **Verified Containers Using GStreamer**

Format	AVC (Supported: Yes/No)	HEVC (Supported: Yes/No)
MP4	Yes	Yes
MPEG2-TS	Yes	Yes
MKV	Yes	Yes
AVI	Yes	No
MPEG2-PS	Yes	No
FLV	Yes	No
3GP	Yes	No

Verified Streaming Protocols Using GStreamer

Table 93: **Verified Streaming Protocols Using GStreamer**

Protocol	AVC (Supported: Yes/No)	HEVC (Supported: Yes/No)	Format
RTP	Yes	Yes	TS
UDP	Yes	Yes	TS
TCP	Yes	Yes	TS
HTTP	Yes	Yes	M3U8, TS, MP4

XILINX

Chapter 27

OpenMax Integration Layer

OpenMax Integration Layer Sample Applications

Two sample applications built using OpenMax Integration Layer are available. The source code for the OpenMax sample applications omx_encoder and omx_decoder are at https://github.com/Xilinx/vcu-omx-il/tree/master/exe_omx.

- **H.265 Decoding File to File:** The `omx_decoder -help` command shows all the options.

```
omx_decoder input-file.h265 -hevc -o out.yuv
```

- **H.265 Encoding File to File:** The `omx_encoder -help` command shows all the options.

```
omx_encoder inputfile.yuv -w 352 -h 288 -r 30 -avc -o out.h264
```

***Note*:** Input YUV file should be in NV12 or NV16 format for 8-bit input sources.

Sync IP API

Structure and Function Definitions

All the API and related structures are located at https://github.com/Xilinx/gst-plugins-good/blob/xlnx-rebase-v1.16.3/sys/v4l2/ext/xlnx-ll/xvfbsync.h.

SyncIp

Description

Holds a handle to save the hardware configuration like maximum number of channels, maximum number of users, maximum number of buffers, buffer descriptor, mutex for locking and channel status.

See

`xvfbsync.h`

SyncChannel

Description

Holds the channel id assigned by the driver and the status of the channel. Each independent stream gets unique channel context with unique SyncChannel structure and channel id.

EncSyncChannel, DecSyncChannel

Description

Encoder and decoder specific wrapper sync IP channel structures.

int xvfbsync_syncip_chan_populate (SyncIp *syncip, SyncChannel * syncip_chan, uint32_t fd)

Description

Get the hardware configuration and capabilities and initialize Sync IP structure. Update the reserved channel id assigned by driver and creates a thread to poll for Sync IP errors.

int xvfbsync_enc_sync_chan_populate (EncSyncChannel * enc_sync_chan, SyncChannel *sync_chan, uint32_t hardware_horizontal_stride_alignment, uint32_t hardware_vertical_stride_alignment)

Description

Initialize encoder Sync IP channel.

int xvfbsync_enc_sync_chan_enable (EncSyncChannel * enc_sync_chan)

Description

Enable Sync IP channel.

int xvfbsync_enc_sync_chan_set_intr_mask (EncSyncChannel * enc_sync_chan, ChannelIntr * intr_mask)

Description

Set Sync IP interrupt mask. Application can mask unnecessary Sync IP interrupts thus reducing overall system load by setting appropriate intr_mask.

int xvfbsync_enc_sync_chan_add_buffer (EncSyncChannel * enc_sync_chan, XLNXLLBuf * buf)

Description

Push buffer to Sync IP channel.

int xvfbsync_syncip_reset_err_status (EncSyncChannel * enc_sync_chan, ChannelErrIntr *err_intr)

Description

Unmask Sync IP interrupt.

int xvfbsync_enc_sync_chan_depopulate (EncSyncChannel * enc_sync_chan)

Description

Disable Sync IP channel and clean Sync IP configurations and resources.

Programming Sequence of Synchronization IP

The producer is V4L2 based application programs the buffer parameters to synchronization IP and give the early buffer done signal to consumer. Based on the signal, consumer perform a read transaction on the buffer and blocks on. The synchronization IP, block the consumer transactions until producer completes writing a slice data to the memory.

- Open the synchronization device using device name `/dev/xlnxsync0`.
- Get the IP capabilities and status using `xvfbsync_syncip_chan_populate()`. The capabilities are maximum number of channels, maximum number of users, and reserved channel.
- Initialize encoder sync IP channel using `xvfbsync_enc_sync_chan_populate()`.
- Enable the channel using `xvfbsync_enc_sync_chan_enable()`.
- Set the Sync IP interrupt mask using `xvfbsync_enc_sync_chan_set_intr_mask()`. This disables the masked interrupts.

- The buffer is programmed using `xvfbsync_enc_sync_chan_add_buffer()`. The parameter should contain channel ID, dma buffer descriptor, luma start and end addresses, and chroma start and end addresses. The buffer addresses should be programmed for both producer and consumer Luma end offset is calculated based on Luma size, pitch, and actual width of the buffer. The calculation, as shown below, is similar for Chroma end offset too.

  Luma end offset = Luma start pointer + Luma size + pitch - actual luma width - 1

  Chroma end offset = Chroma start pointer + Chroma size + pitch - actual chroma width - 1

  Table 94: **Calculating Offsets**

Color Format	Luma start Address	Luma End Offset	Chroma Start Offset	Chroma End Offset
YUV 420 8-bit 1080p	Luma start pointer	1920x1080 + 1920 - 1920 - 1	Chroma start pointer	1920x540 + 1920 - 1920 - 1
YUV 422 10-bit 4k	Luma start pointer	5120x2176 + 2176 - 2176 - 1	Chroma start pointer	5120x2176 + 5120 - 5120 - 1

- The application register for POLLPRI errors. Using all the above steps, the Sync IP gets configured with buffer address details and does the synchronization by making sure that consumer requests for the particular buffer get unblocked only when producer has written sufficient data. If any error occurs on running pipeline then Sync IP driver will mask corresponding interrupt bit and unblocks the poll thread with error event so that application will get the channel error status, once the particular error gets resolve or obsolete then application should unmask the Sync IP interrupt by using `xvfbsync_syncip_reset_err_status()`.
- After completion of the use case, the application should reset the syncip context and de-initialize the syncip channel by using `xvfbsync_enc_sync_chan_depopulate()`. This will disable the particular Sync IP channel and do necessary cleanup of resources.

Programming Sequence of Synchronization IP

The producer is V4L2 based application programs the buffer parameters to synchronization IP and give the early buffer done signal to consumer. Based on the signal, consumer perform a read transaction on the buffer and blocks on. The synchronization IP, block the consumer transactions until producer completes writing a slice data to the memory.

- Open the synchronization device using device name `/dev/xlnxsync0`.
- Get the IP capabilities and status using `xvfbsync_syncip_chan_populate()`. The capabilities are maximum number of channels, maximum number of users, and reserved channel.
- Initialize encoder sync IP channel using `xvfbsync_enc_sync_chan_populate()`.

- Enable the channel using `xvfbsync_enc_sync_chan_enable()`.
- Set the Sync IP interrupt mask using `xvfbsync_enc_sync_chan_set_intr_mask()`. This disables the masked interrupts.
- The buffer is programmed using `xvfbsync_enc_sync_chan_add_buffer()`. The parameter should contain channel ID, dma buffer descriptor, luma start and end addresses, and chroma start and end addresses. The buffer addresses should be programmed for both producer and consumer Luma end offset is calculated based on Luma size, pitch, and actual width of the buffer. The calculation, as shown below, is similar for Chroma end offset too.

  Luma end offset = Luma start pointer + Luma size + pitch - actual luma width - 1

  Chroma end offset = Chroma start pointer + Chroma size + pitch - actual chroma width - 1

  Table 95: **Calculating Offsets**

Color Format	Luma start Address	Luma End Offset	Chroma Start Offset	Chroma End Offset
YUV 420 8-bit 1080p	Luma start pointer	1920x1080 + 1920 - 1920 - 1	Chroma start pointer	1920x540 + 1920 - 1920 - 1
YUV 422 10-bit 4k	Luma start pointer	5120x2176 + 2176 - 2176 - 1	Chroma start pointer	5120x2176 + 5120 - 5120 - 1

- The application register for POLLPRI errors. Using all the above steps, the Sync IP gets configured with buffer address details and does the synchronization by making sure that consumer requests for the particular buffer get unblocked only when producer has written sufficient data. If any error occurs on running pipeline then Sync IP driver will mask corresponding interrupt bit and unblocks the poll thread with error event so that application will get the channel error status, once the particular error gets resolve or obsolete then application should unmask the Sync IP interrupt by using `xvfbsync_syncip_reset_err_status()`.
- After completion of the use case, the application should reset the syncip context and de-initialize the syncip channel by using `xvfbsync_enc_sync_chan_depopulate()`. This will disable the particular Sync IP channel and do necessary cleanup of resources.

Chapter 29

VCU Control Software

The VCU Control Software operates on the frame or slice levels. Its responsibilities are:

- Generating NAL (Network Abstraction Layer) units for encoder.
- Parsing NAL units for decoder.
- Composing and queuing commands for each frame to the MCU Firmware.
- Retrieves status of each frame.
- Concatenates video bit stream generated by hardware and software.

Xilinx VCU Control Software API

The VCU Control Software API is comprised of an encoder library and a decoder library, both in user space, which user space applications link with to control VCU hardware.

Error Checking and Reporting

There several error handling mechanisms in the VCU Control Software API. The most common mechanism is for functions to return a status value, such as a boolean or a pointer that is NULL in the failing case.

The encoder and decoder objects each store an error code to be accessed with `AL_Encoder_GetFrameError` and `AL_Decoder_GetFrameError`, respectively.

User-defined callbacks are sometimes notified of unusual conditions by passing NULL for a pointer that is not normally NULL or do not provide any notification but assume the callback itself uses one of the accessor functions to retrieve the error status from an encoder object or a decoder object.

In unusual or unexpected circumstances, some functions may report errors directly to the console. These are system errors and the messages contain the messages in the following table.

Table 96: **Encoder and Decoder Error Messages**

Error Types	Description
AL_SUCCESS	The operation succeeded without encountering any error
AL_ERROR	Unknown error
AL_ERR_NO_MEMORY	No memory left so couldn't allocate resource. This can be dma memory, mcu specific memory if available or simply virtual memory shortage
AL_ERR_STREAM_OVERFLOW	The generated stream couldn't fit inside the allocated stream buffer
AL_ERR_TOO_MANY_SLICES	If SliceSize mode is supported, the constraint couldn't be respected as too many slices were required to respect it
AL_ERR_CHAN_CREATION_NO_CHANNEL_AVAILABLE	The scheduler can't handle more channel (fixed limit of AL_SCHEDULER_MAX_CHANNEL)
AL_ERR_CHAN_CREATION_RESOURCE_UNAVAILABLE	The processing power of the available cores is insufficient to handle this channel
AL_ERR_CHAN_CREATION_NOT_ENOUGH_CORES	Couldn't spread the load on enough cores (a special case of ERROR_RESOURCE_UNAVAILABLE) or the load can't be spread so much (each core has a requirement on the minimal number of resources it can handle)
AL_ERR_REQUEST_MALFORMED	Some parameters in the request have an invalid value
AL_ERR_CMD_NOT_ALLOWED	The dynamic command is not allowed in some configuration
AL_ERR_INVALID_CMD_VALUE	The value associated with the command is invalid (in the current configuration)

There are various ways encoded bitstream can be corrupted and detecting those errors in a compressed bitstream is complex because of the syntax element coding and parsing dependencies. The errors are usually not detected on corrupted bit but more likely on the following syntax elements.

For example, an encoded bitstream has scaling matrices and "scaling matrices present bit" is corrupted in the stream. When a decoder reads this bitstream, it first assumes that there is no scaling matrices present in the stream and goes on parsing actual scaling matrix data as next syntax element which may cause an error. Ideally, the error was corruption of scaling matrix bit, but the decoder is not able to detect that, and such kind of scenarios are common in video codecs.

The following errors are detected by the hardware IP which conceals the remaining part of the slice; there is no error code, only single flag indicating if the slice has been concealed or not.

```
End of CtB/MB flag
End of slice flag
Invalid Prediction Mode (AVC only)
Invalid RefIdx (AVC only)
MB Skip error (AVC only)
Out of range residual
```

Refer VPS/SPS/PPS parsing function for more details on error handling and reporting:

https://github.com/Xilinx/vcu-ctrl-sw/tree/master/lib_parsing

```
lib_parsing/AvcParser.c and lib_parsing/HevcParser.c and check the calls to
the macro COMPLY.
```

In addition, monitor the `AL_AVC_ParseSliceHeader` and `AL_HEVC_ParseSliceHeader` functions in `lib_parsing/SliceHdrParsing.c` and check the return false paths.

Memory Management

Memory operations are indirected through function pointers. The AL_Allocator default implementation simply wraps malloc and free, etc.

Two higher level techniques are used for memory management: reference counted buffers, and buffer pools. A reference counted buffer is created with a zero reference count. The AL_Buffer_Ref and AL_Buffer_Unref functions increment and decrement the reference count, respectively. The AL_Buffer interface separates the management of buffer metadata from the management of the data memory associated with the buffer. Usage of the reference count is optional.

The AL_TBufPool implementation manages a buffer pool with a ring buffer. Some ring buffers have sizes fixed at compile time. Exceeding the buffer pool size results in undefined behavior. See AL_Decoder_PutDisplayPicture.

AL_Buffer

Holds a handle to memory managed by a AL_TAllocator object. Provides reference count, mutex, and callback to be invoked when the reference count is decremented to zero. Use of the reference count is optional. This type is opaque. The implementation uses al_t_BufferImpl.

See

BufferAPI.h, al_t_BufferImpl

void AL_Buffer_Ref(AL_TBuffer* hBuf)

Increases the reference count of hBuf by one.

See

AL_Buffer_Create_And_Allocate, AL_Buffer_WrapData

void AL_Buffer_Unref(AL_TBuffer* hBuf)

Decreases the reference count of hBuf by one. If the reference count is zero, calls the pCallback function associated with the buffer.

See

AL_Buffer_Create_And_Allocate, AL_Buffer_WrapData

AL_TAllocator

The AL_TAllocator type enables the developer to either wrap or replace memory management functions for memory tracking, alternative DMA buffer handling, etc.

Type	Field	Description
const AL_AllocatorVtable*	vtable	Pointer to function pointers for memory management functions.

See

AL_AllocatorVtable

AL_AllocatorVtable

Supplies memory management functions for the AL_TAllocator type.

Table 97: **Function Descriptions**

Type	Field	Description
bool (*pfnDestroy)(AL_TAllocator*)	pfnDestroy	Releases resources associated with the allocator. This function relates to the allocator, not an allocated handle.
AL_HANDLE (*pfnAlloc)(AL_TAllocator*, size_t)	pfnAlloc	Allocates a handle of a given size. Returns NULL if allocation fails.
bool (*pfnFree)(AL_TAllocator*, AL_HANDLE)	pfnFree	Releases resources associated with a handle returned by pfnAlloc. Returns true, if successful and false otherwise.
AL_VADDR (*pfnGetVirtualAddr)(AL_TAllocator*, AL_HANDLE)	pfnGetVirtualAddr	Translates the given handle into a virtual address.
AL_PADDR (*pfnGetPhysicalAddr)(AL_TAllocator*, AL_HANDLE)	pfnGetPhysicalAddr	Translates the given handle into a physical address.

Table 97: **Function Descriptions** *(cont'd)*

Type	Field	Description
AL_HANDLE (*pfnAllocNamed) (AL_TAllocator*, size_t, char const* name)	pfnAllocNamed	Allocates a buffer with a given name. The name is intended for developer code and is not used internally to the VCU Encoder API or VCU Decoder API.

void AL_Buffer_SetData(const AL_TBuffer* hBuf, uint8_t* pData)

Assigns pData to the data pointer of the buffer hBuf.

AL_HANDLE AL_Allocator_Alloc(AL_TAllocator* pAllocator, size_t zSize)

Return

Returns a pointer to newly allocated memory or NULL if allocation fails.

AL_VADDR AL_Allocator_GetVirtualAddr(AL_TAllocator* pAllocator, AL_HANDLE hBuf)

Return

Returns hBuf.

See

LinuxDma_GetVirtualAddr, LinuxDma_Map, AL_Allocator_GetPhysicalAddr

AL_PADDR AL_Allocator_GetPhysicalAddr(AL_TAllocator* pAllocator, AL_HANDLE hBuf)

Return

Returns NULL.

See

AL_Allocator_GetVirtualAddr

AL_EMetaType

Table 98: **Enumeration Constant Descriptions**

Enumeration Constant	Value	Description
AL_META_TYPE_SOURCE	0	Source buffer
AL_META_TYPE_STREAM	1	Stream buffer
AL_META_TYPE_CIRCULAR	2	Circular buffer
AL_META_TYPE_PICTURE	3	Picture buffer
AL_META_TYPE_EXTENDED	0x7F000000	Extended buffer

See

BufferMeta.h

AL_TStreamMetaData* AL_StreamMetaData_Create(uint16_t uMaxNumSection)

Allocate a stream metadata object. Metadata objects are associated with a buffer and provide context regarding how the buffer should be processed. The uMaxNumSection argument determines how many section structures to allocate. The section structure associates a flag with a region of a buffer as set by AL_StreamMetaData_AddSection. The metadata object must be associated with exactly one AL_TBuffer object. Functions that deallocate buffers such as AL_Buffer_Destroy invoke the destroy function of each metadata object.

Return

Returns a pointer to an AL_TStreamMetaData structure capable of specifying uMaxNumSection sections, unless memory allocation fails, in which case it returns NULL.

See

AL_StreamMetaData_ClearAllSections, AL_StreamMetaData_AddSection, AL_Buffer_AddMetaData, AL_Buffer_Destroy, AL_MAX_SECTION, BufferStreamMeta.h

void AL_StreamMetaData_ClearAllSections(AL_TStreamMetaData* pMetaData)

Sets the section count to zero.

See

AL_StreamMetaData_AddSection

uint16_t AL_StreamMetaData_AddSection(AL_TStreamMetaData* pMetaData, uint32_t uOffset, uint32_t uLength, uint32_t uFlags)

Assigns the parameters defining a new section. The uOffset and uLength are expressed in bytes. If the region extends beyond the end of the buffer, memory corruption may result. The uFlags argument is a bit field. See the AL_Section_Flags enumeration.

Return

Returns the section ID (index) of the added section.

See

AL_ StreamMetaData_ClearAllSectionData, AL_StreamMetaData_ChangeSection, AL_StreamMetaData_SetSectionFlags, AL_StreamMetaData_Create, AL_Section_Flags

AL_Section_Flags

Section flags are used to specify attributes that apply to a region of a buffer.

Table 99: **Constant Values**

Constant	Value	Section Attribute
SECTION_SYNC_FLAG	0x40000000	Data from an IDR
SECTION_END_FRAME_FLAG	0x20000000	End of Frame
SECTION_CONFIG_FLAG	0x10000000	SPS, PPS, VPS, or an AUD

See

AL_StreamMetaData_AddSection

bool AL_Buffer_AddMetaData(AL_TBuffer* pBuf, AL_TMetaData* pMeta)

Adds the pMeta pointer to the metadata of pBuf. Metadata objects are associated with a buffer and provide context regarding how the buffer should be processed. It is inadvisable to add more than one metadata of a given type.

Return

Returns true on success. Returns false if memory allocation fails. Thread-safe.

See

AL_Buffer_GetMetaData, AL_Buffer_RemoveMetaData

AL_TMetaData* AL_Buffer_GetMetaData(AL_TBuffer* pBuf, AL_EMetaType eType)

Return

Gets the first metadata of pBuf that has type eType. Returns NULL if no matching metadata is found. Thread-safe.

See

AL_Buffer_AddMetaData, AL_Buffer_RemoveMetaData

bool AL_Buffer_RemoveMetaData(AL_TBuffer* pBuf, AL_TMetaData* pMeta)

This function removes the pMeta pointer from the metadata of pBuf.

Return

Always returns true. Thread-safe.

See

AL_Buffer_GetMetaData, AL_Buffer_AddMetaData

AL_TBuffer* AL_Buffer_Create_And_Allocate(AL_TAllocator* pAllocator, size_t zSize, PFN_RefCount_CallBack pCallBack)

Allocates and initializes a buffer able to hold zSize bytes. Free the buffer with AL_Buffer_Destroy. Thread safe. The pCallBack is called after the buffer reference count reaches zero and the buffer can safely be reused. The AL_Buffer does not take ownership of the memory associated with its AL_HANDLE. Use AL_Allocator_Free to remove the AL_HANDLE memory, and then use AL_Buffer_Destroy to remove the buffer itself.

***Note*:** Use of reference counting is optional.

Return

Returns a pointer to the buffer.

See

AL_Buffer_Ref, AL_Buffer_Unref, AL_Buffer_SetUserData, AL_Buffer_Destroy, AL_Buffer_Create, AL_Buffer_Create_And_Allocate_Named

void AL_Buffer_Destroy(AL_TBuffer* pBuf)

Frees memory associated with buffer pBuf including metadata. The reference count of pBuf must be zero before this function is called. Does not deallocate the AL_HANDLE used for data memory. This memory is managed separately by the caller. For example, it might be freed using the reference count callback.

See

AL_Allocator_Free

AL_TBuffer* AL_Buffer_WrapData(uint8_t* pData, size_t zSize, PFN_RefCount_CallBackpCallBack)

Allocates a buffer pointing to pData with a reference count of zero. The caller is expected to call AL_Buffer_Ref to increment the reference count. When the reference count is decremented to zero, pCallBack is invoked.

Return

Returns a buffer, if successful. Returns NULL, otherwise.

See

AL_Buffer_Ref, AL_Buffer_Unref

bool AL_Allocator_Free(AL_TAllocator* pAllocator, AL_HANDLE hBuf)

Frees memory associated with buffer hBuf.

See

AL_TAllocator

AL_TAllocator* DmaAlloc_Create(const char* deviceFile)

Opens deviceFile and allocates a small structure to record the device name, file descriptor, and function pointers for manipulating the allocator. The deviceFile must be `/dev/allegroIP`. DmaAlloc_Create does not take ownership of the device file string. When the allocator is no longer needed, AL_Allocator_Destroy should be called to free associated system resources.

Return

Returns a pointer to an allocator if successful, or NULL otherwise.

See

AL_Allocator_Destroy

bool AL_Allocator_Destroy(AL_TAllocator* pAllocator)

Closes the underlying file descriptor and frees associated resources. This function is called when the given memory allocator is no longer needed, invoked when destroying an encoder or decoder instance. After this function is invoked, all handles previously returned by the allocator's alloc function should be considered invalid.

Return

Returns true if successful and false, otherwise.

See

DmaAlloc_Create

int AL_GetAllocSize_DecReference(AL_TDimension tDim, AL_EChromaMode eChromaMode, uint8_t uBitDepth, AL_EFbStorageMode eFrameBufferStorageMode)

Retrieves the size of a reference frame buffer.

[in] tDim	Frame dimensions (width, height) in pixels
[in] eChromaMode	Chroma sampling mode
[in] uBitDepth	Size in bits of picture samples
[in] eStorageMode	Frame buffer storage mode

Return

Returns the size in bytes needed for the reference frame buffer.

int AL_CalculatePitchValue(int iWidth, uint8_t uBitDepth, AL_EFbStorageMode eStorageMode)

[in] iWidth	frame width in pixels
[in] uBitDepth	YUV bit-depth
[in] eStorageMode	source storage mode

Return

Returns the pitch value depending on the source format. Assumes 32-bit burst alignment.

See

AL_GetBitDepth, GetStorageMode

Data Format Conversion

Many picture data conversion functions are provided. All convert an AL_TBuffer to an AL_TBuffer. Because the behavior is evident from the function names and the parameter types are unvarying, after one example, these functions are listed below in tabular form with each (row, column) entry corresponding to a (source, destination) pair.

void I0AL_To_I420(AL_TBuffer const* pSrc, AL_TBuffer* pDst)

Table 100: **Data Conversion Forms**

From	I0AL	I2AL	I420	I422	IYUV	NV12	NV16	P010	P210	RX0A	RX2A	RXmA	T608	T60A	T628	T62A	T6m8	Y010	Y800	YV12
I0AL	•				•	•		•		•								•	•	•
I2AL							•		•		•									
I420	•				•	•		•		•								•	•	•
I422							•		•		•									
IYUV	•		•			•		•		•									•	•
NV12	•		•		•			•		•									•	•
NV16		•		•					•		•									
P010	•		•		•	•		•										•	•	•
P210		•		•																
RX0A	•		•		•	•		•										•	•	•
RX2A		•		•			•		•											
T608	•		•		•	•		•										•	•	•
T60A	•		•		•	•		•										•	•	•
T628		•		•			•		•									•	•	
T62A		•		•			•		•									•	•	
T6m8			•																	
Y010										•		•								
Y800	•		•		•	•		•		•		•						•	•	•
YV12	•		•		•	•		•		•									•	

Constants

config.h

Table 101: **Function Descriptions**

Name	Value	Description
AVC_MAX_HORIZONTAL_RANGE_P	16	AVC maximum horizontal range for P slices
AVC_MAX_VERTICAL_RANGE_P	16	AVC maximum vertical range for P slices
AVC_MAX_HORIZONTAL_RANGE_B	8	AVC maximum horizontal range for B slices
AVC_MAX_VERTICAL_RANGE_B	8	AVC maximum vertical range for B slices
HEVC_MAX_HORIZONTAL_RANGE_P	32	HEVC maximum horizontal range for P slices
HEVC_MAX_VERTICAL_RANGE_P	32	HEVC maximum vertical range for P slices
HEVC_MAX_HORIZONTAL_RANGE_B	16	HEVC maximum horizontal range for B slices
HEVC_MAX_VERTICAL_RANGE_B	16	HEVC maximum vertical range for B slices

Table 101: **Function Descriptions** *(cont'd)*

Name	Value	Description
ENCODER_CORE_FREQUENCY	666,666,666	666.67 Mhz
ENCODER_CORE_FREQUENCY_MARGIN	10	10 Hz
ENCODER_CYCLES_FOR_BLK_32X32	4,900	Used internally
AL_ENC_NUM_CORES	4	Number of Encoder cores
AL_DEC_NUM_CORES	2	Number of Decoder cores
HW_IP_BIT_DEPTH	10	10 bpc
AL_CORE_MAX_WIDTH	4096	Used internally
AL_L2P_MAX_SIZE	4,259,840	Physical memory available for the level-2 prefetch cache in bytes
AL_MAX_ENC_SLICE	200	Maximum number of slices used by the Encoder
AL_BLK16X16_QP_TABLE	0	Used internally

AL_EChromaMode AL_GetChromaMode(TFourCC tFourCC)

Implements the following mapping.

Table 102: **Mapping of FourCC with Chroma Mode**

FourCC	Chroma Mode
I420, IYUV, YV12, NV12, I0AL, P010, T608, T60A, T508, T50A, RX0A	4:2:0
YV16, NV16, I422, P210, I2AL, T628, T62A, T528, T52A, RX2A	4:2:2
Y800, Y010, T6m8, T6mA, T5m8, T5mA, RXmA	Monochrome

Return

Returns the Chroma mode for the given tFourCC argument. If the FourCC mode is not defined, either an assertion violation occurs or -1 is returned.

See

AL_EChromaMode, FOURCC

FOURCC(A)

Converts A into a FourCC value by translating the literal characters of A into their ASCII codes and packing them into a 32-bit value, e.g. FOURCC(A321) becomes 0x33 32 31 41.

AL_EPicFormat

Table 103: **AL_EPicFormat Constants**

Constant	Value	Luma	Chroma	Bit Depth	Chroma Mode	Description
AL_400_8BITS	0x0088	8	8	8	0	4:0:0, 8-bpc
AL_420_8BITS	0x0188	8	8	8	1	4:2:0, 8-bpc
AL_422_8BITS	0x0288	8	8	8	2	4:2:2, 8-bpc
AL_444_8BITS	0x0388	8	8	8	3	4:4:4, 8-bpc
AL_400_10BITS	0x00AA	10	10	10	0	4:0:0, 10-bpc
AL_420_10BITS	0x01AA	10	10	10	1	4:2:0, 10-bpc
AL_422_10BITS	0x02AA	10	10	10	2	4:2:2, 10-bpc
AL_444_10BITS	0x03AA	10	10	10	3	4:4:4, 10-bpc

See

AL_GET_BITDEPTH_LUMA, AL_GET_BITDEPTH_CHROMA, AL_GET_BITDEPTH, AL_GET_CHROMA_MODE, AL_SET_BITDEPTH_LUMA, AL_SET_BITDEPTH_CHROMA, AL_SET_BITDEPTH, AL_SET_CHROMA_MODE

AL_GET_BITDEPTH_LUMA(PicFmt)

Return

Returns the Luma depth from PicFmt. PicFmt must be an AL_EPicFormat value.

See

AL_EPicFormat

AL_GET_BITDEPTH_CHROMA(PicFmt)

Return

Returns the Chroma depth from PicFmt. PicFmt must be an AL_EPicFormat value.

See

AL_EPicFormat

AL_GET_BITDEPTH(PicFmt)

Return

Returns the maximum of the Luma depth and the Chroma depth from PicFmt. PicFmt must be an AL_EPicFormat value.

See

AL_EPicFormat

AL_GET_CHROMA_MODE(PicFmt)

Return

Returns the Chroma mode from PicFmt. PicFmt must be an AL_EPicFormat value.

See

AL_EPicFormat

AL_SET_BITDEPTH_LUMA(PicFmt, BitDepth)

Return

Assigns BitDepth to the low-order byte (byte 0) of PicFmt. PicFmt must be an AL_EPicFormat value.

See

AL_EPicFormat

AL_SET_BITDEPTH_CHROMA(PicFmt, BitDepth)

Return

Assigns BitDepth to byte 1 of PicFmt. PicFmt must be an AL_EPicFormat value.

See

AL_EPicFormat

AL_SET_BITDEPTH(PicFmt, BitDepth)

Return

Assigns BitDepth to byte 0 and byte 1 of PicFmt. PicFmt must be an AL_EPicFormat l-value.

See

AL_EPicFormat

AL_SET_CHROMA_MODE(PicFmt, BitDepth)

Return

Assigns BitDepth to byte 2 of PicFmt. PicFmt must be an AL_EPicFormat l-value.

See

AL_EPicFormat

uint8_t AL_GetBitDepth(TFourCC tFourCC)

Implements the following mapping.

Table 104: **FourCC Bit Depth**

FourCC	Bit Depth
I420, IYUV, YV12, NV12, I422, YV16, NV16, Y800, T6m8, T608, T628, T5m8, T508, T528	8-bpc
I0AL, P010, I2AL, P210, Y010, T6mA, T60A, T62A, T5mA, T50A, T52A, RX0A, RX2A, RXmA	10-bpc

Return

Returns 8 or 10 depending on the given FourCC value. If the FourCC mode is not defined, either an assertion violation occurs or -1 is returned.

See

FOURCC

int GetPixelSize(uint8_t uBitDepth)

Return

Returns 1 if uBitDepth is 8 or less. Returns 2 if uBitDepth is 9 or greater.

AL_EFbStorageMode GetStorageMode(TFourCC tFourCC)

Return

Returns AL_FB_TILE_32x4 or AL_FB_TILE_64x4 according to the tFourCC argument.

See

AL_Is32x4Tiled, AL_Is64x4Tiled

AL_EFbStorageMode AL_GetSrcStorageMode(AL_ESrcMode eSrcMode)

Implements the following mapping.

Table 105: **Source Compression Modes**

Source Compression Mode	Storage Mode
AL_SRC_TILE_64x4, AL_SRC_COMP_64x4	AL_FB_TILE_64x4
AL_SRC_TILE_32x4, AL_SRC_COMP_32x4	AL_FB_TILE_32x4
Other	AL_FB_RASTER

See

AL_EFbStorageMode, AL_ESrcMode

int GetNumLinesInPitch(AL_EFbStorageMode eFrameBufferStorageMode)

Description

Implements the following mapping.

Table 106: **Mapping Storage Mode with Pitch Lines**

Storage Mode	Pitch Lines
AL_FB_TILE_64x4	4
AL_FB_TILE_32x4	4
AL_FB_RASTER	1

See

AL_EFbStorageMode, AL_ESrcMode

AL_IS_AVC(Prof)

Description

Returns true if Prof is AVC. Prof must be an AL_EProfile value.

See

AL_EProfile

AL_IS_HEVC(Prof)

Description

Returns true if Prof is HEVC. Prof must be an AL_EProfile value.

See

AL_EProfile

AL_IS_STILL_PROFILE(Prof)

Description

Returns true if Prof is HEVC Main Still Profile. Prof must be an AL_EProfile value.

See

AL_EProfile

bool AL_IsSemiPlanar(TFourCC tFourCC)

Description

Returns true if tFourCC is a tiled format or one of: NV12, P010, NV16, P210, RX0A, or RX2A.

See

AL_Is64x4Tiled, AL_Is32x4Tiled

bool AL_IsTiled(TFourCC tFourCC)

Description

Returns true if tFourCC is a tiled format.

See

AL_Is64x4Tiled, AL_Is32x4Tiled

bool AL_Is32x4Tiled(TFourCC tFourCC)

Description

Returns true if tFourCC is one of: T508, T528, T5m8, T50A, T52A, or T5mA.

See

AL_IsTiled, AL_Is64x4Tiled

bool AL_Is64x4Tiled(TFourCC tFourCC)

Description

Returns true if tFourCC is one of: T608, T628, T6m8, T60A, T62A, or T6mA.

See

AL_IsTiled, AL_Is32x4Tiled

bool AL_Is10bitPacked(TFourCC tFourCC)

Description

Returns true if tFourCC is a tiled format or one of: RX0A, RX2A, or RXmA.

void AL_GetSubsampling(TFourCC fourcc, int* sx, int* sy)

Description

Writes the subsampling of the FourCC mode into sx and sy as shown in the following mapping.

Table 107: **Chroma Modes**

Chroma Mode	sx	sy
4:2:2	2	2
4:2:0	2	1
Other	1	1

TFourCC AL_EncGetSrcFourCC(AL_TPicFormat const picFmt)

Description

Implements the following mapping. If the picture format picFmt not listed below, either an assertion violation occurs or -1 is returned.

Table 108: **Storage Tile Mode**

Storage Tile Mode	Chroma Mode	BPC	FourCC
64×4	4:2:2	8	T628
		10	T62A
	4:2:0	8	T608
		10	T60A
	Mono	8	T6m8
		10	T6mA
32×4	4:2:2	8	T528
		10	T52A
	4:2:0	8	T508
		10	T50A
	Mono	8	T5m8
		10	T5mA

See

AL_GetSrcFourCC, Get64x64FourCC, Get32x4FourCC

Driver* AL_GetHardwareDriver()

Description

The hardware driver is static structure holding function pointers. The device is opened while creating the encoder.

Return

Returns a pointer to the driver function pointer structure.

See

AL_SchedulerMcu_Create, AL_Encoder_Create

TScheduler* AL_SchedulerMcu_Create(Driver* driver, AL_TAllocator* pDmaAllocator)

Description

Allocates and initializes memory for the scheduler.

Return

Returns true if successful and false, otherwise.

See

AL_GetHardwareDriver, DmaAlloc_Create, AL_SchedulerMcu_Destroy

TScheduler* AL_SchedulerMcu_Destroy(AL_TSchedulerMcu* schedulerMcu)

Description

Frees memory associated with the schedulerMcu.

Return

Returns true if successful and false, otherwise.

See

AL_Encoder_Create, AL_SchedulerMcu_Create

AL_ECodec AL_GetCodec(AL_EProfile eProf)

Return

Returns AL_CODEC_AVC or AL_CODEC_HEVC according to eProf.

VCU Control Software Sample Applications

The `ctrlsw_encoder` and `ctrlsw_decoder` are complete sample applications that encode and decode video respectively. These applications are intended as a learning aid for the VCU Control Software API and for troubleshooting. The source code for the `ctrlsw_encoder` and `ctrlsw_decoder` applications are at https://github.com/Xilinx/vcu-ctrl-sw.

Sample configuration files and input .yuv file mentioned in examples below can be found in the VCU Control Software source tree `test/cfg` folder. The parameters are described after the examples below.

- H.264 Decoding File to File

```
ctrlsw_decoder -avc -in input-avc-file.h264 -out ouput.yuv
```

- H.265 Decoding File to File

```
ctrlsw_decoder -hevc -in input-hevc-file.h265 -out ouput.yuv
```

- Encoding File to File

```
ctrlsw_encoder -cfg encode_simple.cfg
```

***Note*:** For a complete list parameters, type the following in the command line:

```
ctrlsw_decoder --help
```

```
ctrlsw_encoder --help
```

VCU Control Software Encoder Parameters

Input Parameters

Table 109: **Encoder Input Parameters**

Parameter	Description and Possible Values
YUVFile	YUV file name
Width	Frame width in pixels
Height	Frame height in pixels

Table 109: **Encoder Input Parameters** *(cont'd)*

Parameter	Description and Possible Values
Format	I420 (Planar Format): YUV file contains 4:2:0 8-bit video samples stored in planar format with all picture Luma (Y) samples followed by Chroma samples (all U samples then all V samples) IYUV (Planar Format): Same as I420. YV12: Same as I420 with inverted U and V order NV12 (Semi-planar Format): YUV file contains 4:2:0 8-bit video samples stored in planar format with all picture Luma (Y) samples followed by interleaved U and V Chroma samples. NV16 (Semi-planar Format): YUV file contains 4:2:2 8-bit video samples stored in planar format with all picture Luma (Y) samples followed by interleaved U and V Chroma samples. I0AL (Planar Format): YUV file contains 4:2:0 10-bit video samples each stored in a 16-bit word in planar format with all picture Luma (Y) samples followed by Chroma samples (all U samples then all V samples). P010 (Semi-planar Format): YUV file contains 4:2:0 10-bit video samples each stored in a 16-bit word in planar format with all picture Luma (Y) samples followed by interleaved U and V Chroma samples. I422 (Planar Format): YUV file contains 4:2:2 8-bit video samples stored in planar format with all picture Luma (Y) samples followed by Chroma samples (all U samples then all V samples). YV16 (Planar Format): Same as I422 I2AL (Planar Format): YUV file contains 4:2:2 10-bit video samples each stored in a 16-bit word in planar format with all picture Luma (Y) samples followed by Chroma samples (all U samples then all V samples). P210 (Semi-planar Format): YUV file contains 4:2:2 10-bit video samples each stored in a 16-bit word in planar format with all picture Luma (Y) samples followed by interleaved U and V Chroma samples. XV20 (Semi-planar Format): YUV file contains 4:2:2 10-bit video samples where 3 samples are stored per 32-bit word using a semi-planar format with all picture Luma(Y) samples followed by interleaved U and V Chroma samples. Y800 (Planar Format): Input file contains monochrome 8-bit video samples. XV15 (Semi-planar Format): YUV file contains 4:2:0 10-bit video samples where 3 samples are stored per 32-bit word using a semi-planar format with all picture Luma(Y) samples followed by interleaved U and V Chroma samples. Y010 (Planar Format): Input files contains monochrome 10-bit video samples each stored in a 16-bit word. ***Note*:** I420, IYUV, YV12, I0AL, I422, YV16, and I2AL formats are planar and undergo software conversion to become semi-planar.
Framerate	Number of frames per second of the YUV input file. When this parameter is not present, its value is set equal to the framerate specified in Rate Control Parameters. When this parameter is greater than the frame rate specified in the rate control section the rate specified in the rate control section, the encoder repeats some frames encoder drops some frames; when this parameter is lower than the frame.
CmdFile	Text file specifying commands to perform at given frame numbers. The commands include scene change notification, key frame insertion, etc.
RoiFile	Text file specifying a sequence of Region of Interest changes at given frame numbers.
HDRFile	Name of the file specifying HDR SEI contents.
QpTablesFolder	Specifies the location of the files containing the QP tables to use for each frame.
TwoPassFile	File containing the first pass statistics.
Input Buffer Cropping Parameters	
CropWidth	Cropped input yuv Width
CropHeight	Cropped input yuv Height
CropPosX	Cropped input yuv Pixel position X
CropPosY	Cropped input yuv Pixel position Y

Dynamic Input Section Parameters

Table 110: **Dynamic Input Section Parameters**

Parameter	Description and Possible Values
Height	The height of the current source.
Width	The width of the current source.
YUVFile	The YUV source in a different resolution than main input.

Output Parameters

Table 111: **Encoder Output Parameters**

Parameter	Description and Possible Values
BitstreamFile	Elementary stream output file name
RecFile	Optional output file name for reconstructed picture (in YUV format). When this parameter is not present, the reconstructed YUV file is not saved.
Format	FOURCC code of the reconstructed YUV file format, see Input section for possible values. If not specified, the output uses the format of the input.

Rate Control Parameters

Table 112: **Encoder Rate Control Section Parameters**

Parameter	Description and Possible Values
RateCtrlMode	Selects the way the bit rate is controlled CONST_QP: No rate control, all pictures use the same QP defined by the SliceQP parameter. CBR: Use constant bit rate control. VBR: Use variable bit rate control. LOW_LATENCY: Use variable bit rate for low latency application. CAPPED_VBR, PLUGIN Default value: CONST_QP
BitRate	Target bit rate in Kb/s. Not used when RateCtrlMode = CONST_QP Default value: 4000
MaxBitRate	Target bit rate in Kb/s. Not used when RateCtrlMode = CONST_QP Default value: 4000 ***Note*:** When RateCtrlMode is CBR, MaxBitRate shall be set to the same value as BitRate.
FrameRate	Number of frames per second Default value: 30
SliceQP	Quantization parameter. When RateCtrlMode = CONST_QP the specified QP is applied to all slices. When RateCtrlMode = CBR the specified QP is used as initial QP. Allowed values: from 0 to 51 Default value: 30

Table 112: **Encoder Rate Control Section Parameters** *(cont'd)*

Parameter	Description and Possible Values
MinQP[1]	Minimum QP value allowed. This parameter is especially useful when using VBR rate control. In VBR rate control the value AUTO can be used to let the encoder select the MinQP according to SliceQP. Allowed values: from 0 to SliceQP Default value: 10
MaxQP	Maximum QP value allowed. Allowed values: from SliceQP to 51 Default value: 51
InitialDelay	Specifies the initial removal delay as specified in the HRD model, in seconds. Not used when RateCtrlMode = CONST_QP. ***Note*:** If this value is set too low (less than 1 frame period), you may see reduced visual quality. Default value: 1.5
CPBSize	Specifies the size of the Coded Picture Buffer as specified in the HRD model, in seconds. Not used when RateCtrlMode = CONST_QP. Default value: 3.0
IPDelta	Specifies the QP difference between I frames and P frames. Allowed values: AUTO or positive value (≥ 0) Default value: AUTO
PBDelta	Specifies the QP difference between P frames and B frames. Allowed values: AUTO or positive value (≥ 0) Default value: AUTO
ScnChgResilience	Specifies the scene change resilience handling during encode process. Improves quality during scene changes. ENABLE, DISABLE, TRUE, FALSE Default value: TRUE
MaxPictureSize[2]	Maximum frame size allowed in Kb/s. If only MaxPictureSize is provided, it sets MaxPictureSize for all I, P, and B frames. To set MaxPictureSize individually, refer MaxPictureSize.I, MaxPictureSize.P, and MaxPictureSize.B MaxPictureSize = TargetBitrate(in Kbits/s) / FrameRate * AllowedPeakMargin Recommend a 10%AllowedPeakMargin. That is, MaxPictureSize = (100Mb/s / 60 fps) * 1.1 = 1834 Kbits per frame
EnableSkip	Enabling frameskip feature. available values: DISABLE, ENABLE, FALSE, TRUE Default value: FALSE
MaxConsecutiveSkip	Specifies the maximum number of consecutive skipped frames if EnableSkip is enabled. Default value: 4294967295
MaxPictureSize.B	Specifies a coarse size (in Kbits) for B-frame that should not be exceeded. Default value: DISABLE
MaxPictureSize.I	Specifies a coarse size (in Kbits) for I-frame that should not be exceeded. Default value: DISABLE
MaxPictureSize.P	Specifies a coarse size (in Kbits) for P-frame that should not be exceeded. Default value: DISABLE

Table 112: **Encoder Rate Control Section Parameters** *(cont'd)*

Parameter	Description and Possible Values
MaxQuality	Max quality target. Default value: 14.000000

Notes:

1. In VBR the MinQP is computed based on the InitialQP. For example, MinQP = InitialQP - 8. When set to AUTO, the InitialQP is computed based on the video resolution, frame rate, and the target bit-rate. MinQP value less than 10 can generate a very huge picture in case of scene change, and the rate controller may take a longer time with low quality, to recover and achieve the target bit-rate. When InitialQP is 8 (less than 10), the automatic MinQP is always set to 10. When the value of MinQP is AUTO, it should have the same behavior in AVC as in HEVC. When the value of RateCtrlMode is LOW_LATENCY, it enables the HW rate-control of the VCU. This means the QP is adapted within the frame, thus preventing a huge frame. So MinQP does need to be constrained as in frame level ratecontrol.
2. When user set MaxPictureSize parameter, MCU firmware enables the hardware rate control module to keep track of encoding frame size, It adjusts the QPs with in the frame to make sure encoded picture sizes honor the provided max-picture-size. Initial statistics is based on the start of the frame (first few macro block (MB) rows), and the rate control algorithm further modulates the QP as the picture progresses to restrict the maximum picture size with the limit specified by the MaxPictureSize, as much as possible. This is different from than the normal rate control (CBR/VBR), without the MaxPIctureSize parameters enabled, since the rate control only receives feedback after the picture is encoded. This means that the QP withing a frame is manipulated to curtail the frame size to the specified limit (MaxPictureSize). When the MaxPictureSize is enabled, it uses the hardware (VCU Encoder IP, has hardware rate control module in it, which is generally used for low-latency application (low-latency rcmode)) statistics to restrict maximum picture size with in the limit as much as possible.

The CPBSize default value is set to 3 sec in the VCU software (if allowed by the defined encoding level and bit rate parameters), with an option to change this value based on the application requirements. The encoder CBR rate control tries to reach the target bit rate over the period of the GOP length but the main constraint is that it must avoid buffer underflows/overflows as defined by the standard Hypothetical Reference Decoder (HRD) model. A larger CPBSize allows the Encoder rate control to distribute the encoded bits over a larger number of frames so that it can handle larger bit rate variations among consecutive frames and increase video quality. Setting the `CPBSize` so a smaller value can reduce the bit rate peaks but can also impact the video quality.

IMPORTANT! *CPB size tuning must be done based on the application.*

- Video recording and storage applications, where instantaneous bit rate fluctuations are unimportant, and can support larger buffering, and should set the CPBSize to larger values (~1s-3s).
- It is recommended to set the CPBSize to a value that is greater than the GOP length duration (for example, for a 60 fps setting, the GOP length could be set to 60 frames and the CPBSize to more than 1s.)
- Applications that require smaller bit rate variations can reduce the `CPBSize` but it is recommended to set a value larger than ~6 frame periods. It is also recommended in such cases to enable the low-latency rate control mode.
- Low-latency applications should use a "low-delay" GOP type (or intra-only GOP type) and then can reduce the CPBSize down to ~1-2 frame periods.

- For applications that require lower bit-rates like one Mb/s and minimal variation expected in the instantaneous bit-rate, it is recommended to have smaller CPBSize, which makes it easier for the VCU to keep a constant bit-rate. The only reason to scale the CPBSize between very high rates and very low rates is to trade-off quality versus instantaneous bit-rate fluctuations.
- If your use case does bother about instantaneous bit-rate fluctuations (which can cause high CPU usage resulting in dropped frames), then its recommended to increase the CPBSize, which gives the rate algorithm a bit more flexibility to allocate more bits to specific frames, which can improve the quality. For very high bitrates like 100 Mb/s, the quality difference should be negligible, but it might make a difference as the bit-rate decreases.
- Ensure to update InitialDelay whenever you change the CPBSize. InitialDelay can be any value less than or equal to the CPBSize.
- The InitialDelay should have a very minimal effect on the frame-bits consumption. Xilinx recommends that you use half of the CPBSize, but for a small CPBSize (approximately 6 frames), set `CPBSize=InitialDelay`. The default InitialDelay parameter is 1.5s (half of the default CPBSize of 3s).
- The CPBSize and InitialDelay parameters are generally used to verify the hypothetical reference decoder (HDF) buffer model conformance. The InitialDelay parameter specifies the time at which the first picture data needs to be removed from the buffer. It does not refer to the physical buffers, but it helps in verifying the HRD conformance. For more details, see the H264 standard Annex C.
- It is not recommended to have the CPBSize smaller than 6 frames (approximately). This can result in cases where the bits, allocated to each frame, are not as expected because the algorithm does not have enough frames for adjusting the quantization.

Group of Pictures Parameters

Table 113: **Encoder GOP Section Parameters**

Parameter	Description and Possible Values
GopCtrlMode	Specifies the Group Of Pictures configuration mode. Allowed values: 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 with B-pictures only. Each B-picture use the picture just before 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.

Table 113: **Encoder GOP Section Parameters** *(cont'd)*

Parameter	Description and Possible Values
Gop.Length	GOP length in frames including the I-picture. Used only when GopCtrlMode is set to DEFAULT_GOP. Should be set to 0 for Intra-only. Range: 0–1,000. Default value: 30
Gop.FreqIDR	Specifies the minimum number of frames between two IDR pictures (AVC and HEVC). IDR insertion depends on the position of the GOP boundary. Allowed values: positive integer or -1 to disable IDR region. Default value: -1
Gop.FreqLT	Specifies the long term reference picture refresh frequency in number of frames. Allowed values: positive integer or 0 to disable use of Long term reference picture Default value: 0
Gop.NumB	Maximum number of consecutive B-frames in a GOP. Used only when GopCtrlMode is set to DEFAULT_GOP or PYRAMIDAL_GOP. Allowed values: When GopCtrlMode is set to DEFAULT_GOP, Gop.NumB shall be in range 0 to 4. When GopCtrlMode is set to PYRAMIDAL_GOP, Gop.NumB shall take 3, 5, 7, and 15.
Gop.GdrMode[1]	When GopCtrlMode is set to LOW_DELAY_P or LOW_DELAY_B this parameter specifies whether a Gradual Decoder Refresh (GDR) scheme should be used or not. When GDR is enabled, the Gop.Length specifies the frequency at which the refresh pattern should happen. To allow full picture refreshing, this parameter should be greater than the number of CTB/MB rows (GDR_HORIZONTAL) or columns (GDR_VERTICAL). DISABLE: no GDR GDR_VERTICAL: Gradual refresh using a vertical bar moving from left to right. GDR_HORIZONTAL: Gradual refresh using a horizontal bar moving from top to bottom. Default value: DISABLE ***Note*:** When GDR is enabled, the Gop.FreqIDR specifies the frequency at which the refresh pattern should happen. To allow full picture refreshing, this parameter should be greater than the number of CTB/MB rows (GDR_HORIZONTAL) or columns (GDR_VERTICAL).
Gop.EnableLT	Enables/Disables long term reference picture support Allowed Values: TRUE/FALSE Default value: FALSE
Gop.FreqLT	Specifies the frequency of LTR pictures in GOP. Default value: 0
Gop.TempDQP	Specifies a Delta QP for pictures with temporal-ID 1 to 4 Default value: 0 0 0 0

Notes:

1. When GDR is enabled,, the GOP.FreqIDR specifies the frequency at which the refresh pattern should occur. To allow the full picture to refresh, this parameter should be set to a value greater than the number of CTB/MB rows (GDR_HORIZONTAL) or columns (GDR_VERTICAL).

Settings Parameters

Table 114: **Encoder Settings Parameters**

Parameter	Description and Possible Values
Profile	Specifies the profile to which the bitstream conforms. Allowed values: AVC_BASELINE, AVC_C_BASELINE, AVC_C_HIGH, AVC_HIGH, AVC_HIGH10, AVC_HIGH10_INTRA, AVC_HIGH_422, AVC_HIGH_422_INTRA, AVC_MAIN, AVC_PROG_HIGH, HEVC_MAIN, HEVC_MAIN10, HEVC_MAIN10_INTRA, HEVC_MAIN_422, HEVC_MAIN_422_10, HEVC_MAIN_422_10_INTRA, HEVC_MAIN_INTRA, HEVC_MAIN_STILL, HEVC_MONO, HEVC_MONO10, XAVC_HIGH10_INTRA_CBG, XAVC_HIGH10_INTRA_VBR, XAVC_HIGH_422_INTRA_CBG, XAVC_HIGH_422_INTRA_VBR, XAVC_LONG_GOP_HIGH_422_MXF, XAVC_LONG_GOP_HIGH_MP4, XAVC_LONG_GOP_HIGH_MXF, XAVC_LONG_GOP_MAIN_MP4. Default value: HEVC_MAIN
Level	Specifies the Level to which the bitstream conforms Allowed values: from 1.0 to 5.1 for H.265 (HEVC), from 1.0 to 5.2 for H.264 (AVC) Default value: 5.1
Tier	Specifies the tier to which the bitstream conforms (H.265 (HEVC) only) Allowed values: MAIN_TIER, HIGH_TIER Default value: MAIN_TIER
ChromaMode	Selects the Chroma subsampling mode used to encode the stream Allowed values: CHROMA_MONO: The stream is encoded in Monochrome (4:0:0) CHROMA_4_2_0: The stream is encoded with 4:2:0 Chroma subsampling CHROMA_4_2_2: The stream is encoded with 4:2:2 Chroma subsampling Default value: CHROMA_4_2_0
BitDepth	Specifies the bit depth of the Luma and Chroma samples in the encoded stream. Allowed values: 8 or 10 Default value: 8
NumSlices	Specifies the number of slices used for each frame. Each slice contains one or more full LCU row(s) and are spread over the frame as regularly as possible. Allowed values: from 1 up to number of Coding unit rows in the frame. Default value: 1
SliceSize	Target Slice Size specifies the target slice size, in bytes, that the encoder uses to automatically split the bitstream into approximately equally-sized slices, with a granularity of one LCU. This impacts performance, adding an overhead of one LCU per slice to the processing time of a command. This parameter is not supported in H.264 (AVC) encoding when using multiple cores. When SliceSize is zero, slices are defined by the NumSlices parameter. This parameter is directly sent to the Encoder IP and specifies only the size of the Slice Data. It does not include any margin for the slice header. So it is recommended to set the SliceSize parameter with the target value lowered by 5%. For example if your target value is 1500 bytes per slice, you should set "SliceSize = 1425" in the configuration file. Allowed values: 1000-65,535 or 0 to disable the automatic slice splitting. Default value: 0
DependentSlice	When there are several slices per frame (e.g. NumSlices is greater than 1 or SliceSize is greater than 0), this parameter specifies whether the additional slice are Dependent slice segments or regular slices (H.265 (HEVC) only). Allowed values: FALSE, TRUE Default value: FALSE

Table 114: **Encoder Settings Parameters** *(cont'd)*

Parameter	Description and Possible Values
EntropyMode	Selects the entropy coding mode if Profile is set to AVC_MAIN, AVC_HIGH, AVC_HIGH10 or AVC_HIGH_422 (AVC only) Allowed values: MODE_CABAC: the stream is encoded with CABAC MODE_CAVLC: the stream is encoded with CAVLC Default value: MODE_CABAC
CabacInit	Specifies the CABAC initialization table index (H.264 (AVC)) / initialization flag (H.265 (HEVC)). Allowed values: from 0 to 2 (H.264 (AVC)), from 0 to 1 (H.265 (HEVC)) Default value: 0
PicCbQpOffset	Specifies the QP offset for the first Chroma channel (Cb) at picture level. (H.265 (HEVC) only) Allowed values: from –12 to +12 Default value: 0
PicCrQpOffset	Specifies the QP offset for the second Chroma channel (Cr) at picture level (H.265 (HEVC) only) Allowed values: from –12 to +12 Default value: 0
SliceCbQpOffset	Specifies the QP offset for the first Chroma channel (Cb) at slice level. Allowed values: from –12 to +12 Default value: 0
SliceCrQpOffset	Specifies the QP offset for the second Chroma channel (Cr) at slice level Allowed values: from –12 to +12 Default value: 0 ***Note***: (PicCbQPOffset + SliceCbQPOffset) shall be in range –12 to +12 (PicCrQPOffset + SliceCrQPOffset) shall be in range –12 to +12
ScalingList	Specifies the scaling list mode (H.264 (AVC) and H.265 (HEVC) only). Allowed values: FLAT, DEFAULT
QpCtrlMode	Specifies how to generate the QP per CU. UNIFORM_QP: All CUs of the slice use the same QP. AUTO_QP: The QP is chosen according to the CU content using a pre-programmed lookup table. LOAD_QP: the QPs of each LCU come from an external buffer loaded from a file. The file shall be named QPs.hex and located in the working directory. The file format is described in Quantization Parameter (QP) File Format. ADAPTIVE_AUTO_QP: Dynamically compute QP by MB on the fly. RELATIVE_QP, ROI_QP Default value: UNIFORM_QP
CuQpDeltaDepth	Specifies the Qp per CU granularity (H.265 (HEVC) only). Used only when QpCtrlMode is set to AUTO_QP or ADAPTIVE_AUTO_QP 0: down to 32×32 1: down to 16×16 2: down to 8×8 Default value: 0

Table 114: **Encoder Settings Parameters** *(cont'd)*

Parameter	Description and Possible Values
ConstrainedIntraPred	Specifies the value of constrained_intra_pred_flag syntax element. Allowed values: ENABLE, DISABLE Default value: DISABLE
VrtRange_P	Specifies the vertical search range used for P-frame motion estimation: Allowed values for H.265 (HEVC): 16 or 32: using 16 allows to reduce the memory bandwidth (Low Bandwidth mode) Allowed values for H.264 (AVC): 8 or 16: using 8 allows to reduce the memory bandwidth (Low Bandwidth mode) Default value: 32 (HEVC) 16 (AVC)
LoopFilter	Enables/disables the deblocking filter. Allowed values: ENABLE, DISABLE Default value: ENABLE
LoopFilter.CrossSlice	Enables/disables in-loop filtering across the left and upper boundaries of each slice of the frame. Used only when LoopFilter is set to ENABLE. Allowed values: ENABLE, DISABLE Default value: ENABLE
LoopFilter.CrossTile	Enables/disables in-loop filtering across the left and upper boundaries of each tile of the frame. (H.265 (HEVC) only) Used only when LoopFilter is set to ENABLE. Allowed values: ENABLE, DISABLE Default value: ENABLE
LoopFilter.BetaOffset	Specifies the beta offset for the deblocking filter. Used only when LoopFilter is set to ENABLE. Allowed values: from –6 to +6 Default value: –1
LoopFilter.TcOffset	Specifies the Alpha_c0 offset (H.264 (AVC)) or Tc offset (H.265 (HEVC)) for the deblocking filter. Used only when Loop Filter is set to ENABLE. Allowed values: from –6 to +6 Default value: –1
CacheLevel2	Enables/disables the optional Encoder buffer. This can be used to reduce the memory bandwidth and it can slightly reduce the video quality. If enabling this parameter displays an error message from the encoder, it means that the encoder buffer size provided by the VCU driver is too small to handle the minimum motion estimation range. Allowed values: ENABLE, DISABLE Default value: DISABLE
AspectRatio	Selects the display aspect ratio of the video sequence to be written in SPS/VUI. Allowed values: ASPECT_RATIO_AUTO 4:3 for common SD video, 16:9 for common HD video, unspecified for unknown format. ASPECT_RATIO_4_3 4:3 aspect ratio ASPECT_RATIO_16_9 16:9 aspect ratio ASPECT_RATIO_NONE Aspect ratio information is not present in the stream. ASPECT_RATIO_1_1 Default value: ASPECT_RATIO_AUTO

Table 114: **Encoder Settings Parameters** *(cont'd)*

Parameter	Description and Possible Values
LookAhead	LookAhead is the size of the LookAhead (number of frames between the two passes): 0: LookAhead Disabled Above 2: SceneChange Detection and Correction Above 10: Constant quality using frame complexity + Scenechange Detection Default value: 0
VideoMode	Specifies Video mode. Available values: INTERLACED_BOTTOM, INTERLACED_TOP, PROGRESSIVE Default value: PROGRESSIVE
EnableSEI[3]	Determines which supplemental enhancement information are sent with the stream. Available values: SEI_ALL, SEI_BP, SEI_CLL, SEI_MDCV, SEI_NONE, SEI_PT, SEI_RP. Default value: SEI_NONE
LambdaCtrlMode	Specifies the lambda values used for rate-distortion optimization. Available values: AUTO_LDA, DEFAULT_LDA, DYNAMIC_LDA, LOAD_LDA. Default value: AUTO_LDA
LambdaFactors	Specifies the lambda factor for each picture: I, P and B by increasing temporal ID Default value: 0 0 0 0 0 0
EnableFillerData	Specifies if filler data can be added to stream or not Available values: DISABLE, ENABLE, ENC (same as ENABLE but for backward compatibility only) and APP (Encoded stream contains uninitialized filler data that should be filled by the application layer) Default value: ENABLE
AvcLowLat	Enables a special synchronization mode for AVC low latency encoding (validation only). Available values: DISABLE, ENABLE. Default value: DISABLE
ColourDescription	Available values: COLOUR_DESC_BT_2020, COLOUR_DESC_BT_470_NTSC, COLOUR_DESC_BT_601_NTSC, COLOUR_DESC_BT_601_PAL, COLOUR_DESC_BT_709, COLOUR_DESC_EBU_3213, COLOUR_DESC_GENERIC_FILM, COLOUR_DESC_RESERVED, COLOUR_DESC_SMPTE_240M, COLOUR_DESC_SMPTE_EG_432, COLOUR_DESC_SMPTE_RP_431, COLOUR_DESC_SMPTE_ST_428, COLOUR_DESC_UNSPECIFIED. Default value: "COLOUR_DESC_BT_709"
ColourMatrix	Specifies the matrix coefficients used in deriving luma and chroma signals from RGB (HDR setting). Available values: COLOUR_MAT_BT_2100_YCBCR, COLOUR_MAT_UNSPECIFIED. Default value: COLOUR_MAT_UNSPECIFIED
CostMode	Available values: DISABLE, ENABLE. Default value: ENABLE
EnableAUD	Determines if Access Unit Delimiter are added to the stream or not. Available values: DISABLE, ENABLE. Default value: ENABLE
FileScalingList	If ScalingList is CUSTOM, specifies the file containing the quantization matrices
NumCore	Number of cores to use for this encoding. Default value: AUTO
SCDFirstPass	During first pass, to encode faster, enable only the scene change detection. Available values: DISABLE, ENABLE. Default value: DISABLE

Table 114: **Encoder Settings Parameters** *(cont'd)*

Parameter	Description and Possible Values
SliceLat	Enables slice latency mode. Available values: DISABLE, ENABLE. Default value: DISABLE
SrcFormat	Available values: COMP_32x4, COMP_64x4, NVX, TILE_32x4, TILE_64x4. Default value: NVX
SubframeLatency	Enables the subframe latency mode. Available values: DISABLE, ENABLE. Default value: DISABLE
TransferCharac	Specifies the reference opto-electronic transfer characteristic function (HDR setting). Available values: TRANSFER_BT_2100_PQ, TRANSFER_UNSPECIFIED. Default value: TRANSFER_UNSPECIFIED
TwoPass	Index of the pass currently encoded in the Twopass mode. Default value: DISABLE
`UniformSliceType`	Enables using higher slice_types values Available values: TRUE, FALSE. Default value: FALSE When this is TRUE, the following slice-type values are used in the slice header. • I slice: slice-type = 7 is used. • P slice: slice-type=5 is used • B slice: slice-type=6 is used. Supported from 2021.1 release onwards.

Notes:

1. When using GStreamer, following mentioned gop-length and b-frames combinations are not supported but that are mostly unused in real time scenarios. ({2, 1}, {3, 2}, {4, 2}, {4, 3}, {5, 3}, {5, 4}, {6, 3}, {6, 4}, {7, 4}, {8, 4}, {9, 3}, {11, 4}, {12, 4}, {16, 4}).
2. When using GStreamer, you cannot use the "Closed GOP" structures due to Xilinx specification.
3. SEI_UDU stands for user_data_unregistered SEI message. EnableSEI = SEI_ALL, then it enables all supported SEI which includes the SEI_UDU. - SEI_UDU generates a specific SEI_UDU (prefix) along with empty filler data NAL units. EnableSEI = SEI_BP | SEI_PT, then you will not observe any dummy filler data NAL units in the bitstream.

Run Parameters

Table 115: **Encoder Run Section Parameters**

Parameter	Description and Possible Values
Loop	Specifies whether the encoder should loop back to the beginning of the YUV input stream when it reaches the end of the file. Allowed values: TRUE, FALSE Default value: FALSE
MaxPicture	Number of frames to encode Allowed values: integer value greater than or equal to 1, or ALL to reach the end of the YUV input stream. (ALL should not be used with Loop = TRUE, otherwise the encoder never ends). Default value: ALL

Table 115: **Encoder Run Section Parameters** *(cont'd)*

Parameter	Description and Possible Values
FirstPicture	Specifies the first frame to encode. Allowed values: integer value between 0 and the number of pictures in the input YUV file. Default value: 0
BitrateFile	The generated stream size for each picture and bitrate information is written to this file.
InputSleep	Time period in milliseconds. The encoder is given frames each time period (at a minimum). Default value: 0
ScnChgLookAhead	Specify the number of frames for the scene change look ahead. Default value: 3
UseBoard	Specifies if you are using the reference model (DISABLE) or the actual hardware (ENABLE). Available values: DISABLE, ENABLE. Default value: ENABLE

Quantization Parameter (QP) File Format

The QP table modes are enabled by using QpCtrlMode = LOAD_QP or QpCtrlMode = LOAD_QP | RELATIVE_QP.

In this case, the reference model uses the file `QPs.hex` in working directory to specify the QP values at LCU level. Each line of QPs.hex contains one 8-bit hexadecimal value. For H.264 (AVC), there is one byte per 16×16 MB (in raster scan format): absolute QP in [0;51] or relative QP in [-32;31]. For H.265 (HEVC), there is one byte per 32×32 LCU (in raster scan format): absolute QP in [0;51] or relative QP in [-32;31].

Note: Only the 6 LSBs of each byte are used for QP or Segment ID, the 2 MSBs are reserved.

For example, to specify the following relative QP table:

-1	-2	0	1	1	4
-4	-1	1	2	2	1
-1	0	-1	1	1	4
0	0	-2	2	2	6
1	-2	0	1	4	2

The `QPs.hex` file for H.265 (HEVC) is:

```
3F
3E
00
01
01
04
3C
3F
...
```

If a file with name equal to `QP_<frame number>.hex` is present in the working folder, the encoder uses it for frame number <frame number> instead of `QPs.hex`.

For example if you have the following files in the working folder:

- `QP_0.hex`
- `QP_4.hex`
- `QPs.hex`

The encoder uses QP_0.hex for frame #0, QP_4.hex for frame #4 and QPs.hex for all other frames (i.e. frame #1, #2, #3, #5, #6 ...).

Load QP

Updating or loading QPs using LoadQP option is not supported at Gstreamer, but it is possible to use loadQP in livestreaming at Control software level. The QP table is passed an input along with input frame buffer using `AL_Encoder_Process` API. You must update pQPTable values for each frame in live-streaming.

```
bool AL_Encoder_Process(AL_HEncoder hEnc, AL_TBuffer * pFrame, AL_TBuffer *
pQpTable)
```

Pushes a frame buffer to the encoder. According to the GOP pattern, this frame buffer could or could not be encoded immediately.

Parameters

[in]	hEnc	Handle to an encoder object
[in]	pFrame	Pointer to the frame buffer to encode The pFrame buffer needs to have an associated AL_TSrcMetaData describing how the yuv is stored in memory. The memory of the buffer should not be altered while the encoder is using it. There are some restrictions associated to the source metadata. The chroma pitch has to be equal to the luma pitch and must be 32bits aligned and should be superior to the minimum supported pitch for the resolution (See `AL_EncGetMinPitch()`). The chroma offset shouldn't fall inside the luma block. The FourCC should be the same as the one from the channel (See `AL_EncGetSrcFourCC()`).
[in]	pQpTable	Pointer to an optional qp table used if the external qp table mode is enabled

Returns

If the function succeeds the return value is nonzero (true) If the function fails the return value is zero (false).

Lambda File Format

The encoder uses lambda factors per QP value as bitrate as opposed to quality trade-off. By default, the encoder IP use internal lambda factors but it can also use user specified lambda factors. This feature is enabled when `LambdaCtrlMode` is set to LOAD_LDA and when a `Lambdas.hex` file is available in the working directory. Each line of `Lambdas.hex` contains one 32-bit hexadecimal value per QP values (that is 52 lines for QP in range [0...51]).

Each 32 bits word is split as follows:

- **Bit 7 to 0:** Lambda factor for slice B
- **Bit 15 to 8:** Lambda factor for slice P
- **Bit 23 to 16:** Lambda factor for slice I
- **Bit 31 to 24:** Lambda factor for the motion estimation block (this value shall be a power of two)

```
1 2 3 4
01010101 <-------QP 0
01010101
01010101
01010101
01010101
01010101
01010101
01010101
01010101
01010101
01010101
01010101
01010101
01010101
01010101
01010101
01020101
02020202
02020202

100b0a0e
100c0b10
200e0c12
200f0e14
20110f17
20131119
2015131d
20181520
401b1824
401e1b28
40221e2d
40262133
402a2539
40302a40
80352f48
803c3551
80433b5b <-------QP 51
```

- **1:** Motion Estimation
- **2:** Slice I
- **3:** Slice P
- **4:** Slice B

***Note*:** Usually, the lambdas factors follow the equation: $\lambda(QP) = N - 2(QP6 - 2)$, where N is different for I, P and B factors.

Command File Format

The encoder supports input commands simulating dynamic events, like dynamic bitrate and key frame insertion. This feature is enabled when a command file is referenced in the configuration file. Refer to the CmdFile parameter in Input Parameters. The command file format is described below. Each line of the file defines one frame identifier followed by one or more commands applying to this frame.

Syntax

```
<frame number>: <command> [, <command>]
```

Where <command> is one of the following:

Table 116: **Command Descriptions**

Command	Description
SC	Scene change
KF	KeyFrame (restarting new GOP with IDR)
LT	Set next encoded picture as long term reference picture
UseLT	Use long term the reference picture
GopLen=<length>	Change the Gop length
NumB=<numB>	Change the number of consecutive B-Frame
BR=<Kbits/s>	Change the target bit rate
Fps=<frames/s>	Change the frame rate

***Note*:** Keywords such as KF are case sensitive.

Example

```
0: LT
20: KF
30: BR =100, GopLen = 10
45: SC
50: UseLT
101: GopLen= 20, NumB =1
```

ROI File Format

The region of interest definition starts with a line specifying the frame identifier, the background quality and an ROI order for overlapping regions, followed by one or more lines each defining a ROI for this frame and the following frames until the next ROI definition.

Syntax

```
frame <index>: [BkgQuality=<quality>, Order=<order>]
```

<posX> :<posY>, <width>x<height>, <quality>

Quality	Description	Delta QP used
HIGH_QUALITY	Higher quality than the rate-control given value	-5
MEDIUM_QUALITY	Same quality than the rate-control given value	0
LOW_QUALITY	Lower quality than the rate-control given value	+5
NO_QUALITY	Worse possible quality for region without interest	+31[1]

Notes:

1. Maximum delta QP value.

Where <posX>, <posY>, <width>, and <height> are in pixel unit. They are then automatically rounded to the bounding LCU units (16×16 in AVC and 32×32 in HEVC). The <quality> is one of the following:

The <order> is one of the following value:

Table 117: **Order Descriptions**

Order	Description
QUALITY_ORDER	Use quality of the region having the best quality
INCOMING_ORDER	Use quality of the earliest defined region

***Note*:** Keywords such as KF are case sensitive.

Here is an example Region of Interest file.

```
frame 0: BkgQuality= MEDIUM _QUALITY, Order=INCOMING_ORDER
6:4, 8x4, LOW_QUALITY
11:9, 5x5, HIGH_QUALITY
20:15, 5x7, MEDIUM_QUALITY
frame 13:
12:8, 7x5, HIGH_QUALITY
14:8, 5x5, NO_QUALITY
```

HDR10 File Format

The VCU encoder supports HDR10 encoding at the control software level. You need to set the following configuration:

```
#------------------------------------------------------------------------------
[INPUT]
#------------------------------------------------------------------------------
HDRFile = HDRSEIs.txt


#------------------------------------------------------------------------------
[SETTINGS]
#------------------------------------------------------------------------------
TransferCharac = TRANSFER_BT_2100_PQ
ColourMatrix = COLOUR_MAT_BT_2100_YCBCR
EnableSEI = SEI_MDCV | SEI_CLL
ColourDescription = COLOUR_DESC_BT_2020
```

You can define the HDR10 metadata to be inserted through the HDRFile option. The HDR file needs to be in the following format:

```
mastering_display_colour_volume: <display_primaries_GX>
<display_primaries_GY> <display_primaries_BX> <display_primaries_BY>
<display_primaries_RX> <display_primaries_RY> <white_point_x>
<white_point_y> <max_display_mastering_luminance>
<min_display_mastering_luminance>
content_light_level_info: <max_content_light_level>
<max_pic_average_light_level>
```

Control Software

- Encoder application command to insert the HDR10 metadata:

  ```
  ctrlsw_encoder -cfg HDR-XilinxExample.cfg
  ```

- Decoder application command to extract the HDR10 metadata:

  ```
  ctrlsw_decoder -hevc -in bitstream.hevc -out decoded.yuv --hdr-file
  HDRSEIs_output.txt
  ```

  ```
  ctrlsw_decoder -avc -in bitstream.avc -out decoded.yuv --hdr-file
  HDRSEIs_output.txt
  ```

Driver

There are multiple VCU modules. The VCU Init(xlnx_vcu) which is part of Linux Kernel and which handles PL Registers such as VCU Gasket and the clocking. The other three kernel drivers (al5e, al5d, allegro) together are the core VCU driver. The decoder driver is called al5d and encoder driver is called al5e. The common driver is called allegro.

The allegro driver has the following responsibilities:

- Loading the MCU firmware
- Initiating the MCU boot sequence.
- Writing mailbox messages into memory shared between APU and MCU.
- Providing notification of new mailbox messages.

The VCU Init driver source code is at https://github.com/Xilinx/linux-xlnx/blob/xlnx_rebase_v4.19/drivers/soc/xilinx/

The VCU modules (allegro, al5e, al5d) source code is at: https://github.com/Xilinx/vcu-modules

All VCU driver modules (xlnx_vcu, allegro, al5e, al5d) are compiled as runtime kernel modules and are loaded once kernel boot-up. Modules load in the following sequence.

1. The VCU Init driver is loaded (`xlnx_vcu`).
2. The VCU Init driver loads the Allegro modules.
3. Allegro modules are loaded in the following order:
   a. allegro
   b. al5e
   c. al5d

   You can use the `lsmod` command to verify whether the VCU modules were loaded properly. To load the modules, use the `modprobe<driver name>` command and load the drivers in the above mentioned sequence.

MCU Firmware

The MCU firmware running on the MCU has the following responsibilities:

- Transforming frame-level commands from VCU Control Software to slice level commands for the hardware IP core.
- Configuring hardware registers for each command.
- Performing rate control between each frame.

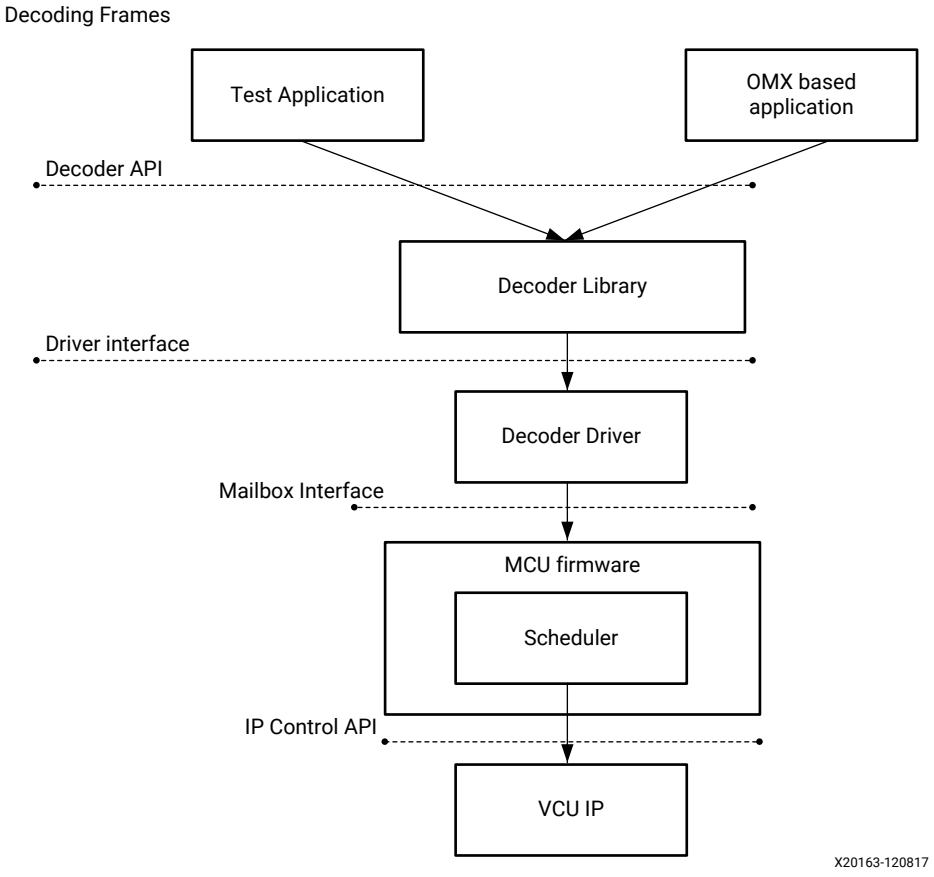
Chapter 32

Encoder and Decoder Stacks

Decoder Stack

The following figure shows the decoder software stack.

Figure 48: **Decoder Overview**



Application

The application can either be test pattern generator or an OpenMAX-based application that uses the VCU decoder.

Decoder Library

The decoder library enables applications to communicate with the MCU firmware through the decoder driver.

Decoder Driver

The decoder driver passes control information as well as buffer pointers of the video to the MCU firmware. The decoder driver uses a mailbox communication technique to pass this information to the MCU firmware.

MCU Firmware

The firmware receives control and buffer information through mailbox. Appropriate action is taken and status is communicated back to decoder driver.

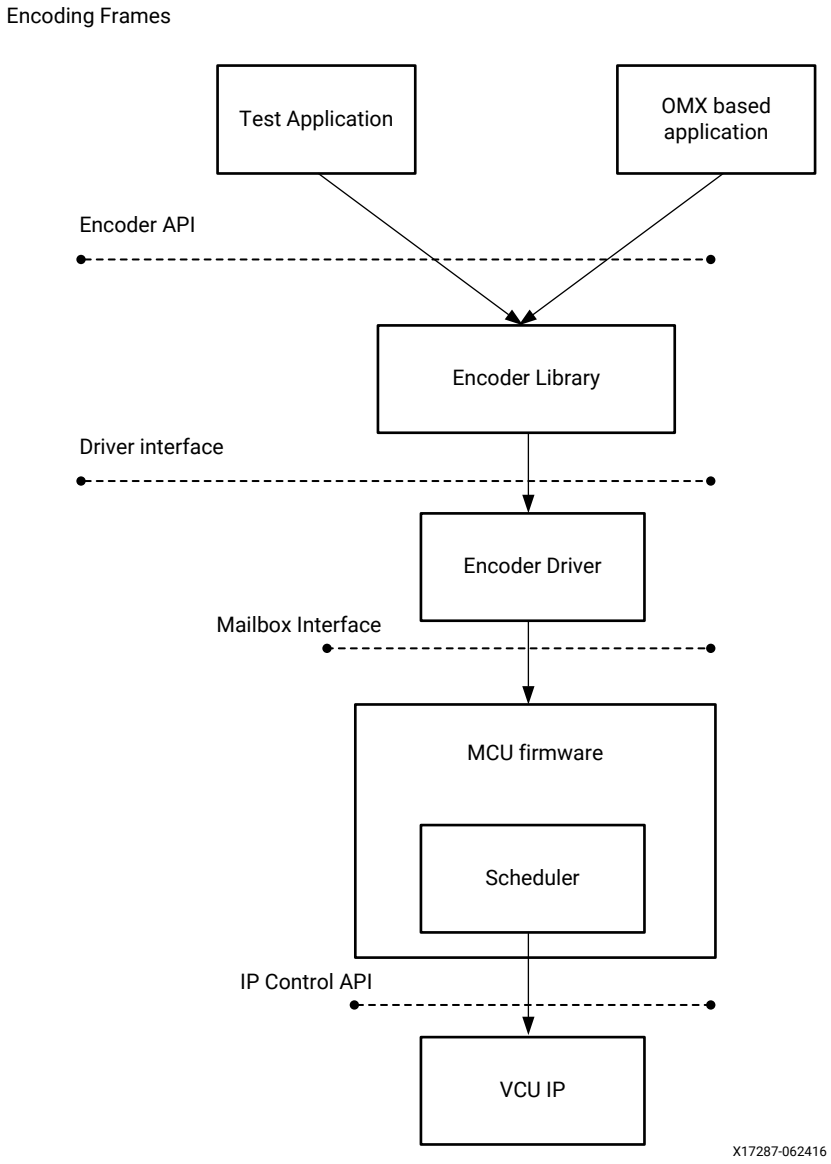
Scheduler

The scheduler, which is part of MCU firmware, programs the hardware IP, handles interrupts and manages the multi-channel and multi-slice aspects of the decoding.

Encoder Stack

The following figure shows the encoding software stack.

Figure 49: **Encoder Overview**



Application

Application refers to any OpenMAX based or standalone application that uses the underlying encoder capabilities of the VCU.

Encoder Library

The encoder library provides the entry points for configuring the encoder and sending frames to the encoder.

Encoder Driver

The encoder driver passes control information and buffer pointers of the video bit stream on which VCU encoder has to operate to the MCU firmware. The encoder driver uses mailbox communication technique to pass this information to MCU firmware.

MCU Firmware

The firmware receives control and buffer information through mailbox. Appropriate action is taken and status is communicated back to encoder driver.

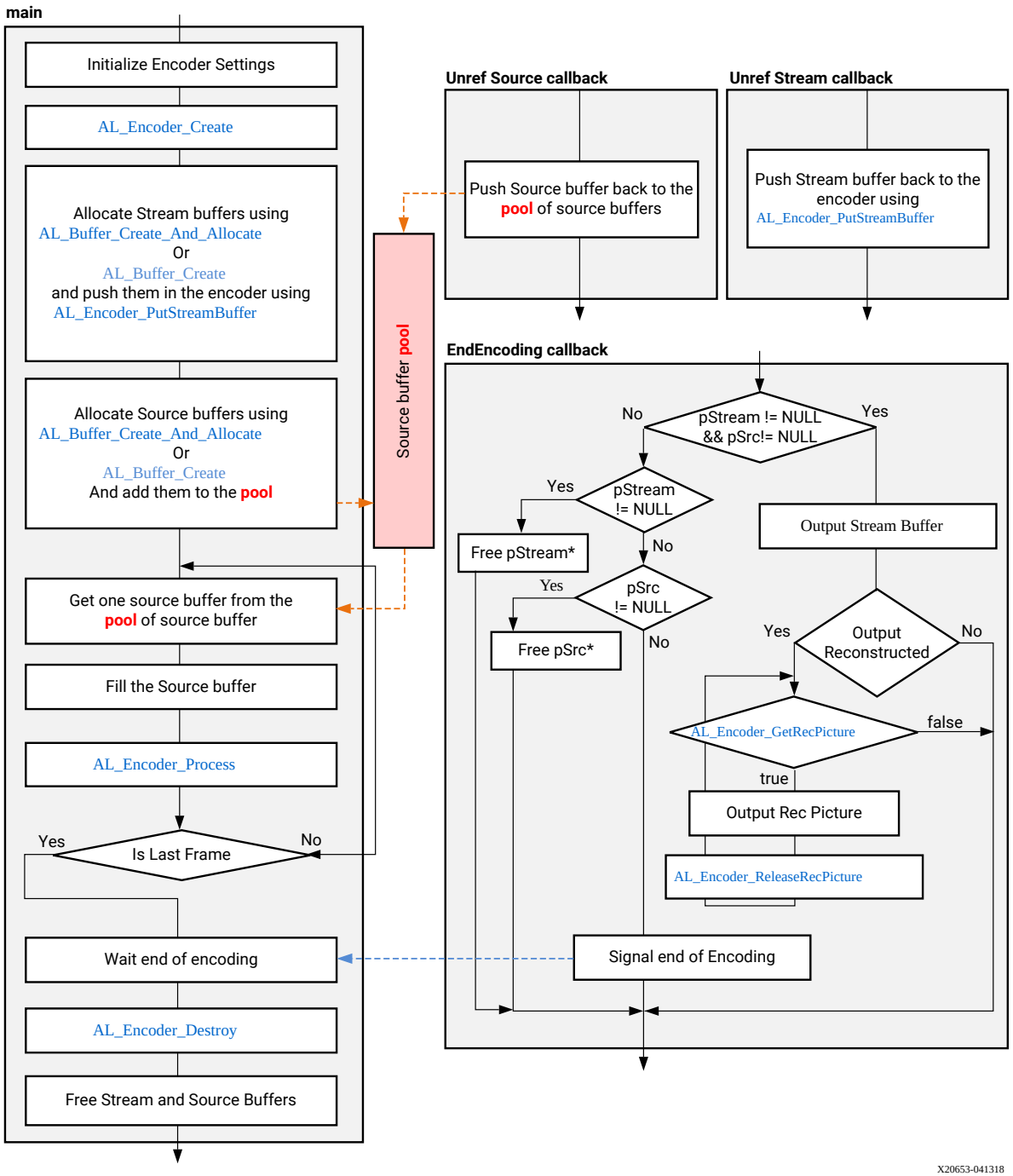
Scheduler

The scheduler directs the activity of the hardware, handles interrupts, and manages the multi-channel and multi-slice aspects of the encoding.

Encoder Flow

The following figure shows the typical flow of control using the VCU Control Software API.

Figure 50: **Encoder API Workflow Example**



Encoder API

The Encoder API is defined in the following header files:

- https://github.com/Xilinx/vcu-ctrl-sw/blob/master/include/lib_encode/lib_encoder.h
- https://github.com/Xilinx/vcu-ctrl-sw/blob/master/include/lib_common_enc/Settings.h
- https://github.com/Xilinx/vcu-ctrl-sw/blob/master/include/lib_common/StreamBuffer.h

The example application `ctrlsw_encoder` demonstrates how to use the API.

AL_ERR AL_Encoder_GetLastError(AL_HEncoder hEnc)

[in] hEnc	handle to the encoder

Description

Returns the error code from the context structure in the encoder. Thread-safe.

Table 118: **Error Code Descriptions**

Error Code	Description
AL_ERR_INIT_FAILED	Failed to initialize the decoder due to parameters inconsistency
AL_ERR_CHAN_CREATION_NO_CHANNEL_AVAILABLE	Channel not created. No channel available
AL_ERR_CHAN_CREATION_RESOURCE_UNAVAILABLE	Channel not created. Processing power of the available cores insufficient
AL_ERR_CHAN_CREATION_NOT_ENOUGH_CORES	Channel not created. Couldn't spread the load on enough cores
AL_ERR_REQUEST_MALFORMED	Channel not created. Request was malformed
AL_ERR_NO_FRAME_DECODED	No frame decoded
AL_ERR_RESOLUTION_CHANGE	Resolution Change is not supported
AL_ERR_NO_MEMORY	Memory shortage detected (DMA, embedded memory, or virtual memory shortage)
AL_SUCCESS	Success

See

Error.h, AL_Decoder_GetLastError

AL_TEncSettings

Description

The Encoder settings are described in the following structure.

Table 119: **Encoder Settings**

Type	Field	Description
AL_TEncChanParam	tChParam	

Table 119: **Encoder Settings** *(cont'd)*

Type	Field	Description
bool	bEnableAUD	Enable Access Unit Delimiter. Default: true
bool	bEnableFillerData	Enable filler data. Default: true
uint32_t	uEnableSEI	Enable supplemental enhancement information. See AL_SeiFlag. Default: SEI_NONE
AL_EAspectRatio	eAspectRatio	Specifies the display aspect ratio
AL_EColourDescription	eColourDescription	COLOUR_DESC_BT_709 = 1 COLOUR_DESC_BT_470_PAL = 5 (default)
AL_EScalingList	eScalingList	AL_SCL_FLAT = 0 AL_SCL_DEFAULT = 1 (default) AL_SCL_CUSTOM = 2 AL_SCL_RANDOM = 3
bool	bDependentSlice	
bool	bDisIntra	
bool	bForceLoad	Default: true
int32_t	iPrefetchLevel2	This parameter specifies the maximum size of the memory used for the Encoder buffer in Bytes. Value must be ≥ 64 and ≤ picture width in pixels. Encoder buffer is only used when CacheLevel2 is enabled.
uint32_t	uL2PSize	
uint16_t	uClipHrzRange	Level 2 cache horizontal range. Must be ? 64 and ? picture width in pixels
uint16_t	uClipVrtRange	Level 2 cache vertical range. Must be ≥ 64 and ≤ picture width in pixels.
AL_EQpCtrlMode	eQpCtrlMode	Quality Parameter control mode. See AL_EQpCtrlMode. Default: UNIFORM_QP.
int	NumView	Default: 1
uint8_t	ScalingList[4][6][64]	
uint8_t	SclFlag[4][6]	
uint32_t	bScalingListPresentFlags	
uint8_t	DcCoeff[8]	
uint8_t	DcCoeffFlag[8]	
bool	bEnableWatchdog	Unused

See

AL_Settings_SetDefaults, AL_Settings_SetDefaultParam, AL_Settings_CheckValidity, AL_Settings_CheckCoherency, AL_Encoder_Create, AL_Common_Encoder_CreateChannel, AL_ExtractNalsData, AL_AVC_SelectScalingList, AL_HEVC_SelectScalingList, AL_HEVC_GenerateVPS, AL_AVC_UpdateHrdParameters, AL_HEVC_UpdateHrdParameters, AL_AVC_GenerateSPS, AL_HEVC_GenerateSPS, AL_AVC_GeneratePPS, AL_HEVC_GeneratePPS, GetHevcMaxTileRow, ConfigureChannel, ParseRateControl, ParseGop, ParseSettings, ParseHardware, ParseMatrice (sic), RandomMatrice (sic), GenerateMatrice (sic), ParseScalingListFile, PostParsingChecks, GetScalingListWrapped, GetScalingList

AL_EChEncOptions

Description

Table 120: **AL_EChEncOptions Options**

Name	Value
AL_OPT_WPP	0x00000001
AL_OPT_TILE	0x00000002
AL_OPT_LF	0x00000004
AL_OPT_LF_X_SLICE	0x00000008
AL_OPT_LF_X_TILE	0x00000010
AL_OPT_SCL_LST	0x00000020
AL_OPT_CONST_INTRA_PRED	0x00000040
AL_OPT_QP_TAB_RELATIVE	0x00000080
AL_OPT_FIX_PREDICTOR	0x00000100
AL_OPT_CUSTOM_LDA	0x00000200
AL_OPT_ENABLE_AUTO_QP	0x00000400
AL_OPT_ADAPT_AUTO_QP	0x00000800
AL_OPT_TRANSFO_SKIP	0x00002000
AL_OPT_FORCE_REC	0x00008000
AL_OPT_FORCE_MV_OUT	0x00010000
AL_OPT_FORCE_MV_CLIP	0x00020000
AL_OPT_LOWLAT_SYNC	0x00040000
AL_OPT_LOWLAT_INT	0x00080000
AL_OPT_RDO_COST_MODE	0x00100000

AL_ERR AL_Encoder_Create(AL_HEncoder* hEnc, TScheduler* pScheduler, AL_TAllocator* pAlloc, AL_TEncSettings const* pSettings, AL_CB_EndEncoding callback)

[out] hEnc	Handle to the new created encoder
[in] pScheduler	Pointer to the scheduler object
[in] pAlloc	Pointer to a AL_TAllocator interface
[in] pSettings	Pointer to a AL_TEncSettings structure specifying the encoder parameters
[in] callback	Callback to be called when the encoding of a frame is finished

Description

Creates a new encoder and returns a handle to it. The encoding format is fixed when the encoder object is created. For applications that encode both H.264 and H.265 streams, to switch from one encoding to the other, the encoder object must be destroyed and recreated with different settings. The parameters of the VCU LogiCore are fixed in the Programmable Logic bitstream and should be selected for the most demanding case.

Return

On success, returns AL_SUCCESS. On error, returns AL_ERROR, AL_ERR_NO_MEMORY, or return value of ioctl. AL_ERROR may indicate a memory allocation failure, failure to open `/dev/allegroIP`, or failure to post a message to the encoder. Error return codes from ioctl indicate failure interacting with the device driver.

See

DmaAlloc_Create, AL_Settings_SetDefaultParam, AL_SchedulerMcu_Create, AL_GetHardwareDriver, AL_Encoder_Destroy, AL_Allocator_Destroy, AL_CB_EndEncoding

AL_CB_EndEncoding

Description

This callback is invoked when any of the following conditions occur:

Table 121: **Function Type Descriptions**

Type	Field	Description
void (*func)(void* pUserParam, AL_TBuffer* pStream, AL_TBuffer const* pSrc,int iLayerID)	func	Called when a frame is encoded, the end of the stream (EOS) is reached, or the stream buffer is released
void*	userParam	User-defined

void AL_Settings_SetDefaults(AL_TEncSettings* pSettings)

Description

Assigns values to pSettings corresponding to HEVC Main profile, Level 51, Main tier, 4:2:0, 8-bits per channel, GOP length = 32, Target Bit Rate = 4,000,000, frame rate = 30, etc. For the complete list of settings see AL_Settings_SetDefaults in Settings.c.

See

Settings.c.

void AL_Settings_SetDefaultParam(AL_TEncSettings* pSettings)

Description

If pSettings->tChParam.eProfile is AVC, set pSettings->tChParam.uMaxCuSize to 4. This corresponds to 16×16. If HEVC and pSettings->tChParam.uCabacInitIdc > 1, set it to 1.

See

AL_Settings_CheckValidity

int AL_Settings_CheckValidity(AL_TEncSettings* pSettings, AL_TEncChanParam * pChParam, FILE* pOut)

Description

If pOut is non-NULL, messages are written to pOut listing the invalid settings in pSettings.

Return

Returns the number of errors found in pSettings.

See

AL_Settings_CheckCoherency

int AL_Settings_CheckCoherency(AL_TEncSettings * pSettings, AL_TEncChanParam * pChParam, TFourCC tFourCC, FILE * pOut)

Description

Checks and corrects some encoder parameters in pSettings. If pOut is non-NULL, messages are written to pOut listing some of the invalid settings in pSettings.

The following settings may be corrected:

```
eQpCtrlMode
eScalingList
iPrefetchLevel2
tChParam.eEntropyMode
tChParam.eOptions
tChParam.ePicFormat
tChParam.eProfile
tChParam.iCbPicQpOffset
tChParam.iCbSliceQpOffset
tChParam.iCrPicQpOffset
tChParam.tGopParam.uFreqIDR
tChParam.tGopParam.uFreqLT
tChParam.tGopParam.uGopLength
tChParam.tGopParam.uNumB
tChParam.tRCParam.uCPBSize
tChParam.tRCParam.uInitialRemDelay
tChParam.tRCParam.uMaxBitRate
tChParam.tRCParam.uTargetBitRate
tChParam.uCuQPDeltaDepth
tChParam.uLevel
tChParam.uMaxCuSize
tChParam.uMaxTuSize
tChParam.uMinCuSize
tChParam.uNumSlices
tChParam.uTier
uL2PSize
```

Note: Not all settings corrections are accompanied by a warning message.

Return

0 if no incoherency.

Number of incoherency, if incoherency were found.

-1 if a fatal incoherency was found.

See

Settings.c, AL_Settings_CheckValidity

int AL_GetMitigatedMaxNalSize(AL_TDimension tDim, AL_EChromaMode eMode, int iBitDepth)

Description

The mitigated worst case NAL size is PCM plus one slice per row.

Return

Returns the maximum size of one NAL unit rounded up to the nearest multiple of 32.

See

AL_TDimension, AL_EChromaMode, AL_GetMaxNalSize

AL_TBufPoolConfig

Description

Structure used to configure the AL_TBufPool.

Table 122: **Function Type Descriptions**

Type	Field	Description
uint32_t	uNumBuf	Number of buffers in the pool
size_t	zBufSize	Size in bytes of the buffers that will fill the pool
char const*	debugName	String to distinguish buffers in the debugger
AL_TMetaData*	pMetaData	Metadata added to each buffer in the pool

See

AL_SrcMetaData_Create

bool AL_Encoder_PutStreamBuffer(AL_HEncoder hEnc, AL_TBuffer* pStream)

[in] hEnc	Handle to an encoder object
[in] pStream	Pointer to the stream buffer given to the encoder

Description

Tells the Encoder where to write the encoded bitstream. The firmware can only handle 320 stream buffers at a given time. However, as the encoder releases one stream buffer after each encoded frame (or slice), the function AL_Encoder_PutStreamBuffer can be called more than 320 times. The pStream argument must have an associated AL_TStreamMetaData pointer added to it. The metadata can be created by AL_StreamMetaData_Create(sectionNumber, uMaxSize) and added with AL_Buffer_AddMetaData(pStream, pMeta), subject to the following constraints:

- sectionNumber = AL_MAX_SECTION, or greater if needed (such as for added SEI within the stream)
- uMaxSize shall be 32-bit aligned

  Note: Does not perform error checking for too many buffers or duplicate buffers.

Return

Returns true

See

AL_StreamMetaData_Create, AL_Buffer_AddMetaData, AL_Encoder_Create, AL_Encoder_Destroy

bool AL_Encoder_Process(AL_HEncoder hEnc, AL_TBuffer* pFrame, AL_TBuffer* pQpTable)

[in] hEnc	Handle to the Encoder object
[in] pFrame	Pointer to the frame buffer to encode
[in] pQpTable	Pointer to an optional quality parameter table used if the external quality parameter mode is enabled. See AL_TEncSettings.eQpCtrlMode

Description

Pushes a frame buffer to the Encoder. The GOP pattern determines whether or not the frame can be encoded immediately. The data associated with pFrame must not be altered by the application during encoding. The pFrame buffer must have an associated AL_TSrcMetaData describing how the YUV data is stored in memory. There are some restrictions associated to the source metadata:

- The Chroma pitch and Luma pitch must be equal
- The Chroma pitch must be 32-bit aligned
- The Chroma pitch should be no less than the minimum supported pitch for the resolution
- The Chroma offset should not fall inside the Luma block
- The FourCC should match the channel (See AL_EncGetSrcFourCC()).

Return

Returns true on success and false otherwise. Call AL_Encoder_GetLastError for the error status code.

See

AL_Encoder_ReleaseFrameBuffer, AL_TEncSettings, AL_CB_EndEncoding.

void AL_Encoder_Destroy(AL_HEncoder hEnc)

Description

Traverses the encoder context hEnc->pCtx deleting and clearing various structures and fields. Frees memory associated with the encoder hEnc.

AL_EBufMode

Description

Indicates blocking or non-blocking mode for certain buffer operations. If a function expecting an AL_EBufMode is called with a value other than AL_BUF_MODE_BLOCK or AL_BUF_MODE_NON_BLOCK, the behavior is undefined.

Table 123: **AL_EBufMode Description**

AL_EBufMode Description	Value
AL_BUF_MODE_BLOCK	0
AL_BUF_MODE_NONBLOCK	1

See

AL_Decoder_PushBuffer, AL_GetWaitMode

int AL_Encoder_AddSei(AL_HEncoder hEnc, AL_TBuffer* pStream, bool isPrefix, int iPayloadType, uint8_t* pPayload, int iPayloadSize, int iTempId);

[in] hEnc	Handle to the encoder
[in] pStream	stream buffer, required to possess a stream metadata
[in] isPrefix	Discriminate between prefix and suffix SEI
[in] iPayloadType	SEI payload type. See Annex D.3 of ITU-T [in] pPayload. Raw data of the SEI payload
[in] iPayloadSize	Size of the raw data payload
[in] iTempId	Temporal id of the raw data payload

Description

Add an SEI to the stream This function should be called after the encoder has encoded the bitstream. The maximum final size of the SEI in the stream cannot exceed 2Ko. The SEI payload does not need to be anti emulated, this is done by the Encoder.

Return

Returns section id.

int AL_Encoder_NotifySceneChange(AL_HEncoder hEnc, int iAhead)

[in] hEnc	Handle to an encoder object
[in]iAhead	Number of frames until the scene change happens. The allowed range is [0...31]

void AL_Encoder_NotifyUseLongTerm(AL_HEncoder hEnc)

Description

Notifies the encoder that the long-term reference frame will be used. This can improve background quality for use cases with unchanging backgrounds such as fixed-camera surveillance.

void AL_Encoder_NotifyIsLongTerm(AL_HEncoder hEnc)

Description

Notify the encoder that the next reference picture is a long term reference picture.

bool AL_Encoder_RestartGop(AL_HEncoder hEnc);

[in] hEnc	Handle to an encoder object

Description

Requests the encoder to insert a Keyframe and restart a new Gop.

Return

If successful, returns true. If unsuccessful, returns false. Call `AL_Encoder_GetLastError` to get the error code.

bool AL_Encoder_SetGopLength(AL_HEncoder hEnc, int iGopLength)

Description

Informs the encoder that the GOP length has changed. If the on-going GOP is longer than the new iGopLength, the encoder restarts the GOP immediately. Otherwise, the encoder restarts the GOP when it reaches the new length. The iGopLength argument must be between 1 and 1,000, inclusive.

Return

If successful, returns true. If unsuccessful, returns false. Call AL_Encoder_GetLastError to get the error code.

bool AL_Encoder_SetGopNumB(AL_HEncoder hEnc, int iNumB)

Description

Informs the encoder of the number of consecutive B-frames between I- and P-frames.

Return

If successful, returns true. If unsuccessful, returns false. Call AL_Encoder_GetLastError to get the error code.

bool AL_Encoder_SetBitRate(AL_HEncoder hEnc, int iBitRate)

Description

Informs the encoder of the target bit rate in bits per second.

Return

If successful, returns true. If unsuccessful, returns false. Call AL_Encoder_GetLastError to get the error code.

bool AL_Encoder_SetInputResolution (AL_HEncoder hEnc, AL_TDimension tDim);

[in] hEnc	Handle to an encoder object
[in] tDim	The new dimension of pushed frames

Description

Changes the resolution of the input frames to encode from the next pushed frame.

Return

If successful, returns true. If unsuccessful, returns false. Call AL_Encoder_GetLastError to get the error code.

bool AL_Encoder_SetQP(AL_HEncoder hEnc, int16_t iQP)

Description

Changes the quantization parameter for the next pushed frame.

Return

If successful, returns true. If unsuccessful, returns false. Call AL_Encoder_GetLastError to get the error code.

bool AL_Encoder_GetRecPicture(AL_HEncoder hEnc, TRecPic* pRecPic);

[in] hEnc	Handle to an encoder object
[in] pRecPic	Pointer to structure that receives the buffer information.

Description

When the encoder has been created with `bEnableRecOutput` set to true, the `AL_Encoder_GetRecPicture` function allows to retrieve the reconstructed frame picture in display order.

Return

Returns true if a reconstructed buffer is available, otherwise false.

void AL_Encoder_ReleaseRecPicture(AL_HEncoder hEnc, TRecPic* pRecPic)

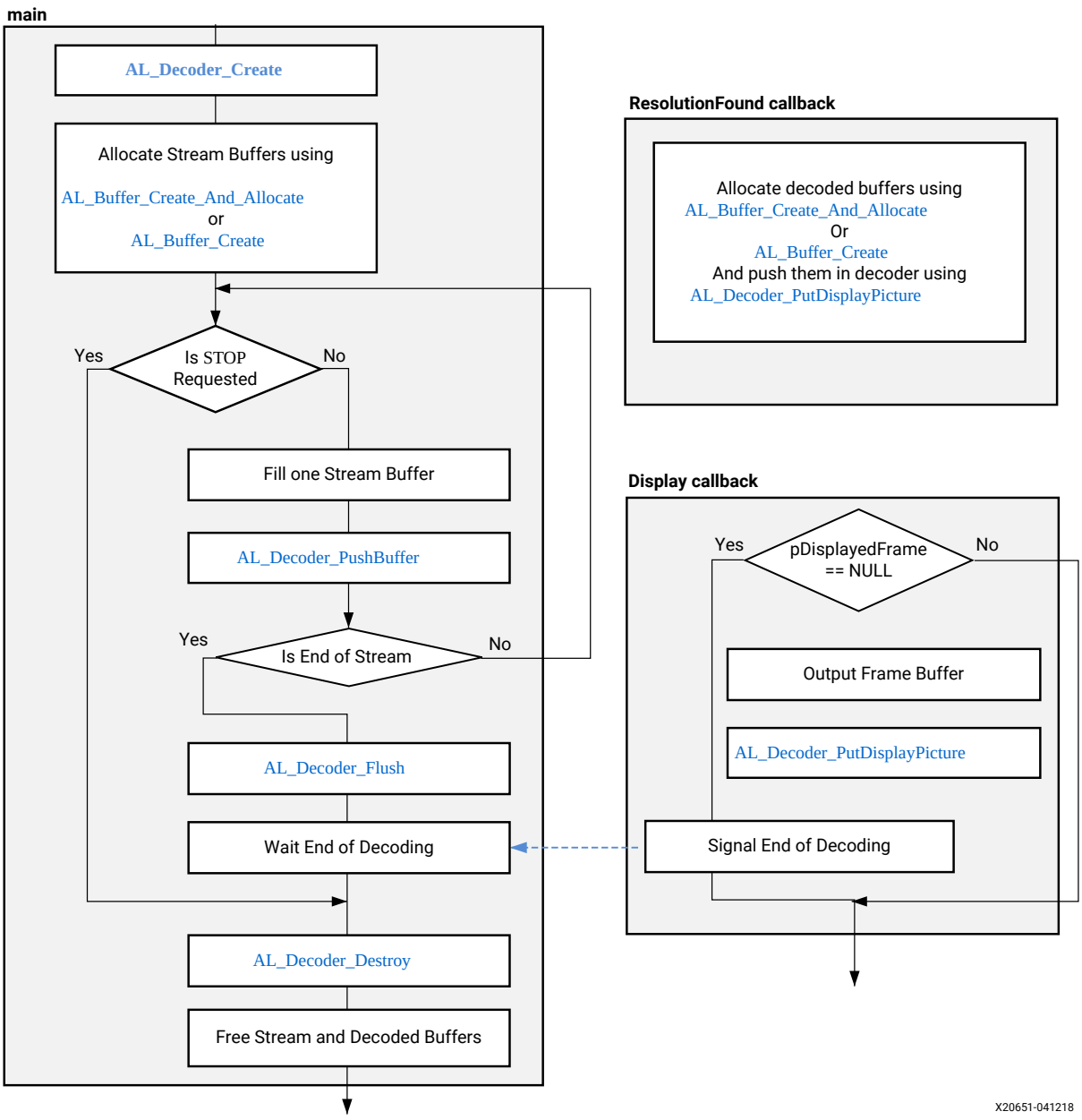
Description

Release reconstructed buffer previously obtains through AL_Encoder_GetRecPicture.

Decoder Flow

The following figure shows an example of using the VCU Control Software API.

Figure 51: **Decoder API Workflow Example**



Decoder API

The Decoder API is defined in https://github.com/Xilinx/vcu-ctrl-sw/blob/master/include/lib_decode. The API is documented below.

The example application ctrlsw_decoder demonstrates how to use the API.

AL_ERR AL_Decoder_Create(AL_HDecoder* hDec, AL_TIDecChannel* pDecChannel, AL_TAllocator* pAllocator, AL_TDecSettings* pSettings, AL_TDecCallBacks* pCB)

[out] hDec	Handle to the newly created decoder
[in] pDecChannel	Pointer to the Scheduler structure
[in] pAllocator	Pointer to an allocator
[in] pSettings	Pointer to the decoder settings
[in] pCB	Pointer to the decoder callbacks

Description

Creates a decoder object. The AL_TIDecChannel object is created by AL_DecChannelMcu_Create. The AL_TAllocator object is created by DmaAlloc_Create. The AL_TDecSettings object and the AL_TDecCallBacks object are initialized by settings their fields directly.

Return

On success, returns AL_SUCCESS. Otherwise, returns one of the following errors:

- AL_ERROR – Unidentified error
- AL_ERR_NO_MEMORY – Memory allocation failure
- AL_ERR_INIT_FAILED – See CheckSettings

See

AL_DecChannelMcu_Create, `/dev/allegroDecodeIP`, DmaAlloc_Create, AssignSettings, CheckSettings, AL_TDecCallBacks

AL_TDecCallBacks

Description

Table 124: **Function Type Descriptions**

Type	Field	Description
AL_CB_EndDecoding	endDecodingCB	Called when a frame is decoded
AL_CB_Display	displayCB	Called when a buffer is ready to be displayed
AL_CB_ResolutionFound	resolutionFoundCB	Called when a resolution change occurs
AL_CB_ParsedSei	parsedSeiCB	Called when a SEI is parsed

See

AL_CB_EndDecoding, AL_CB_Display, AL_CB_ResolutionFound

AL_CB_EndDecoding

Description

Table 125: **Function Type Descriptions**

Type	Field	Description
void (*func)(AL_TBuffer* pDecodedFrame, void* pUserParam)	func	Called when a frame is decoded. On error, pDecodedFrame is NULL. Use AL_Decoder_GetLastError for more information.
void*	userParam	User-defined

See

AL_TDecCallBacks, AL_Decoder_GetLastError

AL_CB_Display

Description

Table 126: **Function Type Descriptions**

Type	Field	Description
void (*func)(AL_TBuffer* pDisplayedFrame, AL_TInfoDecode* pInfo, void* pUserParam)	func	Called when a frame is displayed. On end of stream, pDisplayedFrame is NULL. Use AL_Decoder_GetLastError to check for error information.
void*	userParam	User-defined

See

AL_TDecCallBacks, AL_Decoder_GetLastError, AL_Decoder_PutDisplayPicture

AL_CB_ParsedSei

Description

Resolution SEI callback definition is parsed.

Table 127: **Function Type Descriptions**

Type	Field	Description
void (* func)(int iPayloadType, uint8_t* pPayload, int iPayloadSize, void* pUserParam);	func	Called when a SEI is parsed. Anti-emulation has already been removed by the decoder from the payload. See Annex D.3 of ITU-T for the sei_payload syntax.
void*	userParam	User-defined

AL_CB_ResolutionFound

Description

Resolution change callback definition.

Type	Field	Description
void (*func)(int BufferNumber, int BufferSize, AL_TStreamSettings const* pSettings, AL_TCropInfo const* pCropInfo, void* pUserParam)	func	Called only once when the first decoding process occurs. The decoder doesn't support a change of resolution inside a stream. Use AL_Decoder_GetLastError to check for error information.
void*	userParam	User-defined

See

AL_TDecCallBacks, AL_Decoder_GetLastError, AL_Decoder_PutDisplayPicture

AL_TIDecChannel* AL_DecChannelMcu_Create()

Description

Returns a pointer to the newly allocated AL_TIDecChannel object or NULL if allocation fails.

AL_TIDecChannel

AL_CB_EndFrameDecoding

Description

When the decoder has finished decoding a frame, this callback is invoked. This callback structure contains a function pointer to a function that takes a user parameter and a picture status. The picture status is a pointer to an AL_TDecPicStatus. The default callback is AL_Default_Decoder_EndDecoding.

Table 128: **Function Type Descriptions**

Type	Field	Description
void (*func)(void* pUserParam, AL_TBuffer* pStream, AL_TDecPicStatus* pPicStatus)	func	Called when a frame is decoded
void*	userParam	User-defined

See

AL_Default_Decoder_EndDecoding

void AL_Decoder_Destroy(AL_HDecoder hDec)

Description

Releases resources associated with the Decoder hDec.

See

AL_Decoder_Create

void AL_Decoder_PutDisplayPicture(AL_HDecoder hDec, AL_TBuffer* pDisplay)

Description

Adds the display buffer pDisplay to the decoder's internal buffer pool used for outputting decoded pictures. At most 50 display buffers are allowed. Depending on the compiler settings, exceeding this limit either results in an assertion violation or the buffer is cleared but not added to the pool.

See

AL_TFrmBufPool, FRM_BUF_POOL_SIZE

uint32_t AL_Decoder_GetMinPitch(uint32_t uWidth, uint8_t uBitDepth, AL_EFbStorageMode eFrameBufferStorageMode);

[in] uWidth	Width of the reconstructed buffers in pixels
[in] uBitDepth	Stream depth (8 or 10)
[in] eFrameBufferStorageMode	Frame buffer storage mode

Description

Give the minimum stride supported by the decoder for its reconstructed buffers.

uint32_t AL_Decoder_GetMinStrideHeight(uint32_t uHeight);

[in] uHeight	Height of the picture in pixels

Description

Give the minimum stride height supported by the decoder.

Restriction

The decoder still only supports a stride height set to AL_Decoder_GetMinStrideHeight. All other strideHeight is ignored.

bool AL_Decoder_PreallocateBuffers(AL_HDecoder hDec)

Description

Allocates memory according to the stream settings and the selected codec. Also performs some buffer initialization.

Return

Returns true if allocation succeeded and false otherwise.

See

AL_Default_Decoder_PreallocateBuffers, AL_Default_Decoder_AllocPool, AL_Default_Decoder_AllocMv, AL_PictMngr_Init

oid AL_Default_Decoder_EndDecoding(void* pUserParam, AL_TDecPicStatus* pStatus)

Description

The default function pointer that is invoked when the decoder has finished decoding a frame performs various display buffer operations:

- Updates the display buffer CRC

- Updates buffer pointers, counters, and reference counts
- Releases unused frame decoding memory
- Signals that a frame is ready to be displayed
- Prints a message to `stdout` if the decoder needs to be restarted

See

AL_PictMngr_EndDecoding, AL_t_PictureManagerCallbacks, AL_PictMngr_UnlockRefMvID

AL_TDecPicStatus

Description

Associates counters, CRC, and flags with a frame.

Table 129: **Function Type Descriptions**

Type	Field	Description
uint8_t	uFrmID	Frame identifier in the display FIFO
uint8_t	uMvID	Motion vector identifier
uint32_t	uNumLCU	Number of entries in the Largest Coding Unit (LCU)
uint32_t	uNumBytes	Unused
uint32_t	uNumBins	Unused
uint32_t	uCRC	Frame CRC
bool	bConceal	Is the frame concealed?
bool	bHanged	Did the decoder timeout?

AL_TDecSettings

Description

Table 130: **Function Type Descriptions**

Type	Field	Description
int	iStackSize	Size of the command stack handled by the decoder
int	iBitDepth	Output bit depth
uint8_t	uNumCore	Number of decoder cores used
uint32_t	uFrameRate	User-supplied frame rate used if a syntax element specifying the frame rate is not present
uint32_t	uClkRatio	User-supplied clock ratio used if a syntax element does not supply it

Table 130: **Function Type Descriptions** *(cont'd)*

Type	Field	Description
bool	bIsAvc	If true, AVC decoding is selected; if false HEVC decoding is selected
bool	bParallelWPP	If true, wavefront parallelization processing is enabled. Not supported. Must be false.
uint8_t	uDDRWidth	Width of the DDR uses by the decoder. Either 16 or 32, depending on the board design.
bool	bDisableCache	If true, the decoder cache is disabled
bool	bLowLat	If true, low latency decoding is in use
bool	bForceFrameRate	If true, the user-defined frame rate overrides the stream frame rate
bool	bFrameBufferCompression	If true, internal frame buffer compression should be used
AL_EFbStorageMode	eFBStorageMode	AL_FB_RASTER = 0 AL_FB_TILE_32x4 = 2 AL_FB_TILE_64x4 = 3
AL_EDecUnit	eDecUnit	Sub-frame latency control AL_AU_UNIT = 0 AL_VCL_NAL_UNIT = 1
AL_EDpbMode	eDpbMode	Low reference frame mode control AL_DPB_NORMAL = 0 AL_DPB_NO_REORDERING
AL_TStreamSettings	tStream	Decoder output stream

AL_TSrcMetaData* AL_SrcMetaData_Create(AL_TDimension tDim, AL_TPlane tYPlane, AL_TPlane tUVPlane, TFourCC tFourCC);

[in] tDim	Dimension of the the picture (width and height in pixels)
[in] tYPlane	Array of luma plane parameters (offset and pitch in bytes)
[in] tUVPlane	Array of chroma plane parameters (offset and pitch in bytes)
[in] tFourCC	FourCC of the framebuffer

Description

Create a source metadata.

Return

On success, returns a pointer to the metadata. In case of an allocation failure, the value returned is NULL.

int AL_SrcMetaData_GetLumaSize(AL_TSrcMetaData* pMeta);

Description

Get the size of the luma inside the picture.

Return

Returns size of the luma region.

int AL_SrcMetaData_GetChromaSize(AL_TSrcMetaData* pMeta);

[in] pMeta	A pointer to the source metadata.

Description

Get the size of the chroma inside the picture.

Return

Returns size of the chroma region.

bool AL_Decoder_PushBuffer(AL_HDecoder hDec, AL_TBuffer* pBuf, size_t uSize,)

Description

Pushes a buffer into the decoder queue. Generates an incoming work event. It is decoded as soon as possible. The uSize argument indicates how many bytes of the buffer to decode. The eMode argument specifies whether or not to block. The function pointer supplied in the AL_CB_EndDecoding is invoked with a pointer to the decoded frame buffer and a user parameter. This and other function pointers are provided when the decoder object is created. Note the provided buffer pBuf must not have AL_TCircMetaData associated with it.

See

AL_EBufMode, AL_CB_EndDecoding, AL_Decoder_Create

void AL_Decoder_Flush(AL_HDecoder hDec)

Description

Requests the Decoder flush the decoding request stack when stream parsing is finished.

void AL_Decoder_GetMaxBD(AL_HDecoder hDec)

Description

Returns the maximum bit depth supported for the current stream profile.

int32_t RndPitch(int32_t iWidth, uint8_t uBitDepth, AL_EFbStorageMode eFrameBufferStorageMode)

Return

Returns the pitch rounded up to the burst DMA alignment.

int32_t RndHeight(int32_t iHeight)

Return

Returns the height aligned up to the nearest 64-bit boundary.

int AL_GetNumLCU(AL_TDimension tDim, uint8_t uLCUSize)

Description

Expresses the area pixels defined by tDim in units of 2uLCUSize. The uLCUSize argument must be 4, 5, or 6 corresponding to blocks of 16×16, 32×32, or 64×64, respectively.

Return

Returns the number of LCU in a frame.

int AL_GetAllocSize_HevcCompData(AL_TDimension tDim, AL_EChromaMode eChromaMode)

[in] tDim	Frame dimension (width, height) in pixels
[in] eChromaMode	Chroma sub-sampling mode

Return

Returns the size of an HEVC compressed buffer (LCU header + MVDs + Residuals).

int AL_GetAllocSize_AvcCompData(AL_TDimension tDim, AL_EChromaMode eChromaMode)

[in] tDim	Frame dimension (width, height) in pixels
[in] eChromaMode	Chroma sub-sampling mode

Return

Returns the size of an AVC compressed buffer (LCU header + MVDs + Residuals).

Function: int AL_GetAllocSize_DecCompMap(AL_TDimension tDim);

[in] tDim	Frame dimension (width, height) in pixels
[in] tDim	Frame dimension (width, height) in pixels

Figure 52: **Return**

Returns maximum size in bytes of the compressed map buffer (LCU Offset and size).

int AL_GetAllocSize_HevcMV(AL_TDimension tDim)

[in] tDim	Frame dimension (width, height) in pixels
[in] tDim	Frame dimension (width, height) in pixels

Return

Returns the size in bytes needed for the co-located HEVC motion vector buffer.

int AL_GetAllocSize_AvcMV(AL_TDimension tDim)

[in] tDim	Frame dimension (width, height) in pixels
in] tDim	Frame dimension (width, height) in pixels

Return

Returns the size (in bytes) needed for the co-located AVC motion vector buffer.

int AL_GetAllocSize_Frame(AL_TDimension tDim, AL_EChromaMode eChromaMode, uint8_t uBitDepth, bool bFrameBufferCompression, AL_EFbStorageMode eFrameBufferStorageMode)

[in] tDim	Frame dimension (width, height) in pixels
[in] eChromaMode	Chroma mode
[in] uBitDepth	Bit Depth
[in] bFrameBufferCompression	Specifies if compression is needed
[in] eFrameBufferStorageMode	Storage Mode of the frame buffer

Return

Returns the size in bytes of the output frame buffer.

int AL_GetAllocSize_DecReference(AL_TDimension tDim, int iPitch AL_EChromaMode eChromaMode, AL_EFbStorageMode eFrameBufferStorageMode)

[in] tDim	Frame dimension (width, height) in pixel
[in] eChromaMode	Chroma subsampling
[in] uBitDepth	Size in bits of picture samples
[in] eFrameBufferStorageMode	Storage Mode of the frame buffer

Return

Returns the size in bytes of a reference frame buffer.

uint32_t GetAllocSizeEP2(AL_TDimension tDim, uint8_t uMaxCuSize)

[in] tDim	Frame size in pixels
[in] uMaxCuSize	Maximum Size of a Coding Unit

Return

Returns maximum size in bytes needed to store a QP Ctrl Encoder parameter buffer (EP2).

uint32_t AL_GetAllocSize(AL_TDimension tDim, uint8_t uBitDepth, AL_EChromaMode eChromaMode, AL_EFbStorageMode eStorageMode)

[in] tDim	Frame size in pixels
[in] uBitDepth	YUV bit-depth
[in] eChromaMode	Chroma Mode
[in] eStorageMode	Source Storage Mode

Return

Returns maximum size in bytes needed for the YUV frame buffer.

uint32_t GetAllocSize_Src(AL_TDimension tDim, uint8_t uBitDepth, AL_EChromaMode eChromaMode, AL_ESrcMode eSrcFmt)

[in] tDim	Frame size in pixels
[in] uBitDepth	YUV bit-depth
[in] eChromaMode	Chroma mode
[in] eSrcFmt	Source format used by the hardware IP

Return

Returns size in bytes of the YUV Source frame buffer.

int AL_CalculatePitchValue(int iWidth, uint8_t uBitDepth, AL_EFbStorageMode eStorageMode)

[in] iWidth	Frame width in pixel unit
[in] uBitDepth	YUV bit-depth
[in] eStorageMode	Source storage mode

Return

Returns pitch value in bytes.

AL_EFbStorageMode AL_GetSrcStorageMode(AL_ESrcMode eSrcMode)

[in] iWidth	Frame width in pixel unit
[in] eSrcMode	Source Mode

Return

Returns the Source frame buffer storage mode.

AL_ERR AL_Decoder_GetFrameError(AL_HDecoder hDec, AL_TBuffer* pBuf);

[in] hDec	Handle to a decoder object.
[in] pBuf	Pointer to the decoded picture buffer for which to get the error status.

Description

Retrieves the error status related to a specific frame.

Return

Returns the frame error status.

AL_ERR AL_Decoder_GetLastError(AL_HDecoder hDec)

[in] hDec	handle to the decoder

Return

Returns the error code from the context structure in the decoder. Thread-safe.

Table 131: **Error Code Descriptions**

Error Code	Description
AL_ERR_INIT_FAILED	Failed to initialize the decoder due to parameters inconsistency
AL_ERR_CHAN_CREATION_NO_CHANNEL_AVAILABLE	The channel was not created because no channel available
AL_ERR_CHAN_CREATION_RESOURCE_UNAVAILABLE	The channel was not created because the processing power of the available cores is insufficient
AL_ERR_CHAN_CREATION_NOT_ENOUGH_CORES	The channel was not created because too few processing cores were available to spread the load
AL_ERR_REQUEST_MALFORMED	The channel was not created because the request was malformed
AL_ERR_NO_FRAME_DECODED	No frame was decoded
AL_ERR_RESOLUTION_CHANGE	Resolution Change is not supported
AL_ERR_NO_MEMORY	A memory shortage was detected (DMA, embedded memory, or virtual memory shortage)

Tuning Visual Quality

Quality of encoded video is primarily a function of target-bit rate and the type of the video content. There few encoder parameters which can be used to adjust the encoder quality. Perform the following steps to debug HEVC/AVC encoder quality issues.

1. Adjust the number of B-frames (Gop.NumB) according to the amount of motion, e.g. increased to 2 for static scenes or video conference-like of content, or reduced to 0 for sequences with a lot of motion and/or high frame rates
2. The VBR rate control mode can improve the average quality when some parts of the sequence have lower complexity and/or motion
3. For video conference or when random access is not needed, replace the IPP... GOP by the LOW_DELAY_P GOP and optionally enable the GDR intra refresh
4. If there are many scene changes, enable the ScnChgResilience setting to reduce artifacts following scene change transitions
5. If scene changes can be detected by the system, the encoder's scene change signaling API should be called instead (i.e. with ScnChgResilience disabled) for the encoder to dynamically adapt the encoding parameters and GOP pattern. The scene change information can be provided in a separate input file (CmdFile) when using the control software test application.

TIP: *To improve the video quality, consider using Scene Change Detect hardware.*

6. If the highest PSNR figures are targeted instead of subjective quality, it is recommended to set QPCtrlMode = UNIFORM_QP and ScalingList = FLAT.
7. Evaluate the valid bit rate range for the given video stream by using CONST_QP rate control algorithm.
8. Use a QP value of 22, 27, 32, 37 with the CONST_QP rate control algorithm
9. Using the bit rate range from step 2, evaluate PSNR for CBR/VBR/Low latency rate control algorithms
10. Run similar experiments for libx264 or libx265 encoders.

    Here is an example pipeline for CBR:

```
ffmpeg -s 1280x720 -pix_fmt yuv420p -framerate 50 -I /srv/data1/kvikrama/
vcu_video_quality_runs/source/old_town_cross_420_720p50.yuv -c:v libx265
-preset medium -x265-params bframes=0:ref=1:keyint=30:rc
lookahead=0:ipratio=1:pbratio=1:trellis=0 -b:v 1M -minrate 1M -maxrate
1M -bufsize 2M  -frames 500  old_town_cross_420_720p50_x265.mp4
```

11. Calculate the BD-rate using JCT-VC common test conditions evaluation metric.
12. If there is difference in PSNR between VCU and libx264/libx265, tune the following parameters to see impact:

MinQP, MaxQP, ScnChgResilience (should be Enabled), NumSlices (set to 1).

XILINX

Chapter 36

Optimum VCU Encoder Parameters for Use Cases

Video Streaming

- Video streaming use-case requires very stable bitrate graph for all pictures.
- Avoid periodic large Intra pictures during encoding session
- Low-latency rate control (hardware RC) is the preferred control-rate for video streaming, it tries to maintain equal amount frame sizes for all pictures.
- Avoid periodic Intra frames instead use low-delay-p (IPPPPP) with Intra refresh enable (gdr-mode=horizontal or vertical)
- VBR is not a preferred mode of streaming.

AVC Encoder Performance Settings

- It is preferred to use eight or higher slices for better AVC encoder performance.
- AVC standard does not support Tile mode processing which results in processing of MB rows sequentially for entropy coding.

Low Bitrate AVC Encoding

- Enable profile=high and use qp-mode=auto for low-bitrate encoding use-cases.
- High profile enables 8x8 transform which results in better video quality at low-bitrates.
- Enable GopMode=low-delay-p to improve quality for low-bitrate use-cases.

Example Design

VCU Out of the Box Examples

The supported VCU out of box examples are listed below. Default Desktop application shows two VCU examples (4K AVC Decode and 4K HEVC Decode) icons corresponding to VCU-Decode → Display use case, more details are mentioned in Example-1.

- VCU-Decode → Display: Decodes AVC/HEVC encoded container stream and Displays on DP monitor. It supports aac/vorbis audio decode along with video.
- USB-Camera → Encode → Decode → Display: Captures raw video frames from Camera and Encode/Decode using VCU and display using DP monitor.
- USB-Camera → Decode → Display: Captures encoded video content from Camera, Decode using VCU and display using DP monitor.
- Transcode → to File: Transcodes input AVC/HEVC encoded bitstream into HEVC/AVC format and stores on disk.
- Transcode → Stream out using Ethernet ... Streaming In → Decode → Display: Transcodes input AVC/HEVC encoded bitstream into HEVC/AVC format and streams out to another board using RTP/UDP, and second board receives the data decode using VCU, Display using DP monitor.
- USB-Camera → Audio/Video Encode → File: Video recording use-case.
- USB-Camera → Encode → stream out ... stream-in → Decode → Display: Video conference use-case.

Verifying the Examples

The following sections describe how to verify the example.

Boot the Board (ZCU106/ZCU104) Using the PetaLinux BSP

Xilinx PetaLinux board support package (BSP) is a Linux operating system running a sample user design.

1. Ensure the board is connected to Ethernet to download sample video content from Xilinx web server.

2. If the board is connected to private network, then export proxy settings in `/home/root/.bashrc` file as follows:

```
create/open a bashrc file using "vi ~/. bashrc"
```

3. Insert the following lines into the .bashrc file:

```
export http_proxy="<private network proxy address>"
export https_proxy="<private network proxy address>"
```

   If the board is not connected to Internet, then the compressed video files can be downloaded using host machine. Copy the input files into the `/home/root/` folder and use the following commands to download the content on the host Linux-machine.

4. Download AVC sample file:

```
wget petalinux.xilinx.com/sswreleases/video-files/
bbb_sunflower_2160p_30fps_normal_avc.mp4
```

5. Download HEVC sample file:

```
wget petalinux.xilinx.com/sswreleases/video-files/
bbb_sunflower_2160p_30fps_normal_hevc.mkv
```

Example-1: (Decode → Display)

The Matchbox Desktop has the following applications:

- **4K AVC Decode:** For 4K AVC Decode, click on the application to download sample AVC/AAC encoded bitstream and run VCU Example-1 (Decode → Display). If the sample AVC content is already present in `/home/root`, then it decode→display the content.
- **4K HEVC Decode:** For 4K HEVC Decode, click on the application to download sample HEVC/Vorbis encoded bitstream and run VCU Example-1 (Decode → Display).
  - Run the script with `-h` option shows all possible options.

```
vcu-demo-decode-display.sh -i /home/root/
bbb_sunflower_2160p_30fps_normal_avc.mp4 -c avc -a aac
vcu-demo-decode-display.sh -i /home/root/
bbb_sunflower_2160p_30fps_normal_hevc.mkv -c hevc -a vorbis
```

  - Run the following command to play a YouTube video to ensure that the Ethernet is connected to the board

```
vcu-demo-decode-display.sh -u "youtube-URL"
```

  - Run the following command for help

```
vcu-demo-decode-display.sh -h
```

Example-2: (USB-Camera → Encode → Decode → Display)

1. Connect USB camera to the board (Verified Cameras: Logitech HD camera, C920).

2. Run the following command for USB video capture serial pipeline

```
vcu-demo-camera-encode-decode-display.sh -s 640x480
```

3. Run the following commands for USB video capture serial pipeline with audio for the applicable setting:

   - **ALSA Library API:**

```
vcu-demo-camera-encode-decode-display.sh -s 640x480 -a aac --use-
alsasrc -i "hw:1,0" --use-alsasink --audio-output "hw:0,0"
```

   - **Pulse Library API:**

```
vcu-demo-camera-encode-decode-display.sh -s 640x480 -a aac -i
"alsa_input.usb-
046d_HD_Pro_Webcam_C920_C06272EF-02. analog-stereo" \
--use-pulsesrc --use-pulsesink --audio-output "alsa_output.
platform-fd4a0000.zynqmp- display_zynqmp_dp_snd_card. analog-stereo"
```

   - **Gstreamer Autoaudiosrc and Autoaudiosink plugins:**

```
vcu-demo-camera-encode-decode-display.sh -s 640x480 -a aac
```

   ***Note*:** In above example using gstreamer auto plugins, gstreamer selects the API and devices to be used for capture and playback automatically. Normally, it tries to use default devices enumerated at bootup by pulseaudio server or as mentioned in the alsa configuration file. So, it might not choose the device you want to use sometimes, in which case you can use the above-mentioned script arguments.

***Note*:** For selecting the values of arguments to be passed to `-i` and `---audio-output` option, refer section 10.2.

Example-3: (USB-Camera → Decode → Display)

Connect USB camera to the board (Verified Cameras: Logitech HD camera, C920).

1. Connect USB camera to the board (Verified Cameras: Logitech HD camera, C920).
2. Run the following command for USB video decode display pipeline:

```
vcu-demo-camera-decode-display.sh -s 1920x1080
```

3. Run the command for USB video decode pipeline with audio:

   - **ALSA Library API:**

```
vcu-demo-camera-decode-display.sh -a aac --use-alsasrc -i "hw:1,0" --
use-alsasink --audio-output "hw:0,0"
```

   - **Pulse Library API:**

```
vcu-demo-camera-decode-display.sh -s 640x480 -a aac -i
"alsa_input.usb-046d_HD_Pro_Webcam_C920_C06272EF-02. analog-stereo" \
--use-pulsesrc --use-pulsesink --audio-output "alsa_output.
platform-fd4a0000.zynqmp-display_zynqmp_dp_snd_card. analog-stereo"
```

Note: For "How to Guide" on selecting the values of arguments to be passed to -i and ---audio-output option, refer section 10.2.

- **Gstreamer Autoaudiosrc and Autoaudiosink plugins:**

```
vcu-demo-camera-decode-display.sh -a aac
```

IMPORTANT! *Resolutions for Example 2 and 3 must be set based on USB Camera Capabilities.*

To find capabilities, use `v4l2-ctl --list-formats-ext`. If V4lutils is not installed in the pre-built image, install it using dnf or rebuild the PetaLinux image including v4lutils.

Example-4: (Transcode → to File)

This example requires input sample files. If Example-1 is executed already, then sample videos are stored in the `/home/root` folder.

1. Use the following command to transcode sample avc file into hevc file.

```
vcu-demo-transcode-to-file.sh -i /home/root/
bbb_sunflower_2160p_30fps_normal_avc.mp4 -c avc -a aac -o /home/root/
transcode.mkv
```

2. Use Example-1 to view the transcoded file on the Display. The sample command is shown below:

```
vcu-demo-decode-display.sh -i /home/root/transcode.mkv -c hevc -a vorbis
--use-alsasink --audio-output "hw:0"
```

Example-5: (Transcode → Stream out using Ethernet ... Streaming In → Decode → Display)

This example requires two boards. Board-1 is used for transcoding and streaming-out (as a server) and board-2 is used for streaming-in and decoding purposes (as a client). The VLC player on the host machine can be used as a client instead of board-2.

1. Ensure input video file is copied to board-1 for streaming.
2. Connect two boards back to back with an Ethernet cable or make sure both the boards are connected to the same Ethernet hub.
   a. If the VLC player is used as client, ensure that the host machine and server (board-1) are connected to the same Ethernet hub or connect them back to back using an Ethernet cable.

      Set client IP address and execute stream-in → Decode example on board-2, if board-2 is used as a client and connect the boards back to back with an Ethernet cable.

```
ifconfig eth0 192.168.0.2
```

Note: Setting up an IP is not required if the boards are already connected to the same LAN.

```
vcu-demo-streamin-decode-display.sh -c avc
```

Note: This means client is receiving AVC bitstream from server, Use -c hevc if server is sending HEVC bitstream.

3. If the VLC player is used as client, set the host machine IP address to 192.168.0.2 if it is connected to the board directly with an Ethernet cable.

   Note: Setting up an IP address is not required if the boards are already connected to the same LAN.

4. Create a `test.sdp` file on the host with the following content (add a separate line in `test.sdp` for each of the following items) and play `test.sdp` on host machine.

   ```
   v=0 c=IN IP4 <Client machine IP address>
   m=video 50000 RTP/AVP 96
   a=rtpmap:96 H264/90000
   a=framerate=30
   ```

   Trouble-shoot for VLC player setup: IP4 is client-IP address. H264/H265 is used based on received codec type on the client. Turn-off the firewall in the host machine if packets are not received to VLC.

5. Set server IP and execute Transcode → stream-out example on board-1

   ```
   ifconfig eth0 192.168.0.1
   ```

   Note: Setting of IP is not required if the boards are already connected to the same LAN.

   ```
   vcu-demo-transcode-to-streamout.sh -i /home/root/
   bbb_sunflower_2160p_30fps_normal_hevc.mkv -c hevc -b 5000 -a <Client
   Machine/Board IP address>
   ```

Example-6: (Camera Audio/Video → Encode → File A/V Record)

1. Connect a USB camera to the board (Verified cameras: Logitech HD camera C920).
2. Execute the following command for USB video recording.

   ```
   vcu-demo-camera-encode-file.sh -a aac -n 1000 --use-alsasrc -i "hw:1"
   ```

An output file `camera_output.ts` is generated in the current directory which has recorded an audio/video file in the mpegts container.

The file can be played on media players like VLC.

Example-7: (Camera Audio/Video → Stream out using Ethernet ... Streaming In → Decode → Display with Audio)

This example requires two boards: Board-1 is used for encoding live A/V feed from camera and streaming-out (as a server) and board-2 is used for streaming-in and decoding purposes to play, as a client, the received audio/video stream.

1. Connect two boards back to back with an Ethernet cable or make sure both the boards are connected to the same Ethernet hub.
2. Set the client IP address and execute stream-in → Decode example on board-2.

```
ifconfig eth0 192.168.0.2
```

Note: Setting of IP is not required if the boards are already connected to the same LAN.

```
vcu-demo-streamin-decode-display.sh -c avc --audio-type aac
```

Note: This means client is receiving AVC bitstream from server, use `-c hevc`. If server is sending HEVC bitstream).

3. Set server IP and run Camera → Encode → stream-out on board-1.

```
ifconfig eth0 192.168.0.1 (Note: setting of ip is not required if the
boards already connected to same LAN)
vcu-demo-camera-encode-streamout.sh --audio-type aac --use-alsasrc -i
"hw:1" --address 192.168.0.2
```

Running Examples with Jupyter Notebook

These examples can also be run through Jupyter notebook GUI utility. To build and run these examples, follow these steps:

1. To build the example, follow these steps:
   a. Create PetaLinux project using released 2020.1 zcu106/zcu104 BSPs.
   b. Configure and build image:

```
petalinux-config -c rootfs
```

   c. Enable the **gstreamer-vcu-notebooks** option under **User Packages** and then build the BSP

```
>> petalinux-build
```

2. To run the example, follow these steps:
   a. Make sure that the ZCU106/104 board is connected to the Ethernet.
   b. Boot the board using generated images.
   c. Login to kernel using username as `root` and password as `root`.
   d. The Jupyter notebook server is executed on the boot. The URL to connect to jupyter-notebook is in the `/var/log/jupyter.log` file.

```
>> cat /var/log/jupyter.log
netip is net_ip= echo
CONNECTING... 0 seconds
netip is
CONNECTING... 2 seconds
netip is
CONNECTING... 4 seconds
netip is
```

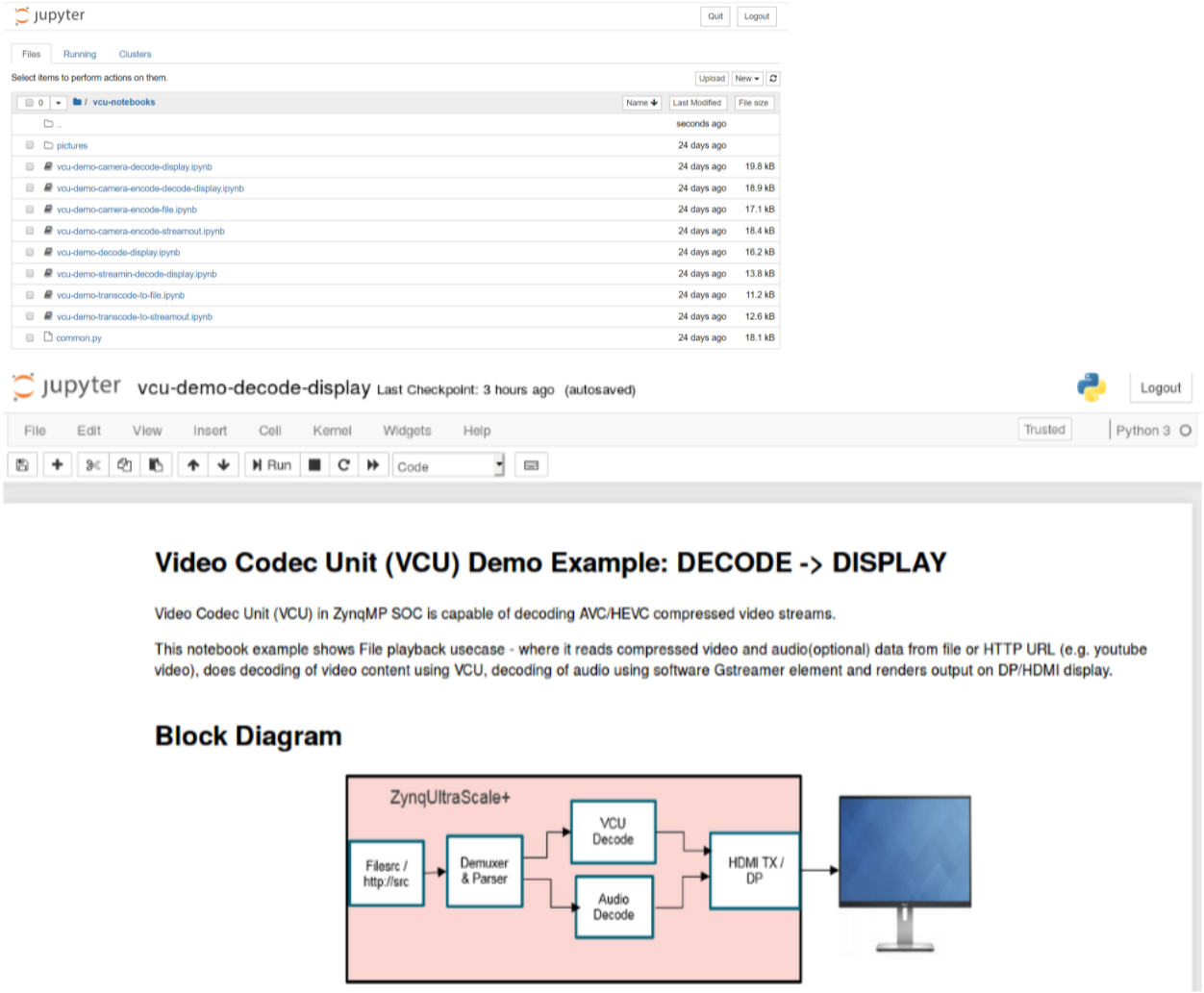
```
172.19.0.23
[I 21:11:06.985 NotebookApp] Writing notebook server cookie secret
to /.local/share/jupyter/runtime/notebook_cookie_secret
[I 21:11:08.292 NotebookApp] Serving notebooks from local
directory: /usr/share/example-notebooks
[I 21:11:08.293 NotebookApp] The Jupyter Notebook is running at:
[I 21:11:08.293 NotebookApp] http://172.19.0.23:8888/?
token=f1897000565b0255786124d56318b4852f9062f768a877a7
[I 21:11:08.293 NotebookApp] Use Control-C to stop this server and
shut down all kernels (twice to skip confirmation).
[C 21:11:08.318 NotebookApp]

To access the notebook, open this file in a browser:

file:///.local/share/jupyter/runtime/nbserver-2812-open.html
Or copy and paste one of these URLs:
http://172.19.0.23:8888/?
token=f1897000565b0255786124d56318b4852f9062f768a877a7
```

e. Copy the generated URL to the remote host's web browser. For example: http://172.19.0.23:8888/ ?token=f1897000565b0255786124d56318b4852f9062f768a877a7

The host PC's browser tab should show the following VCU notebook examples:

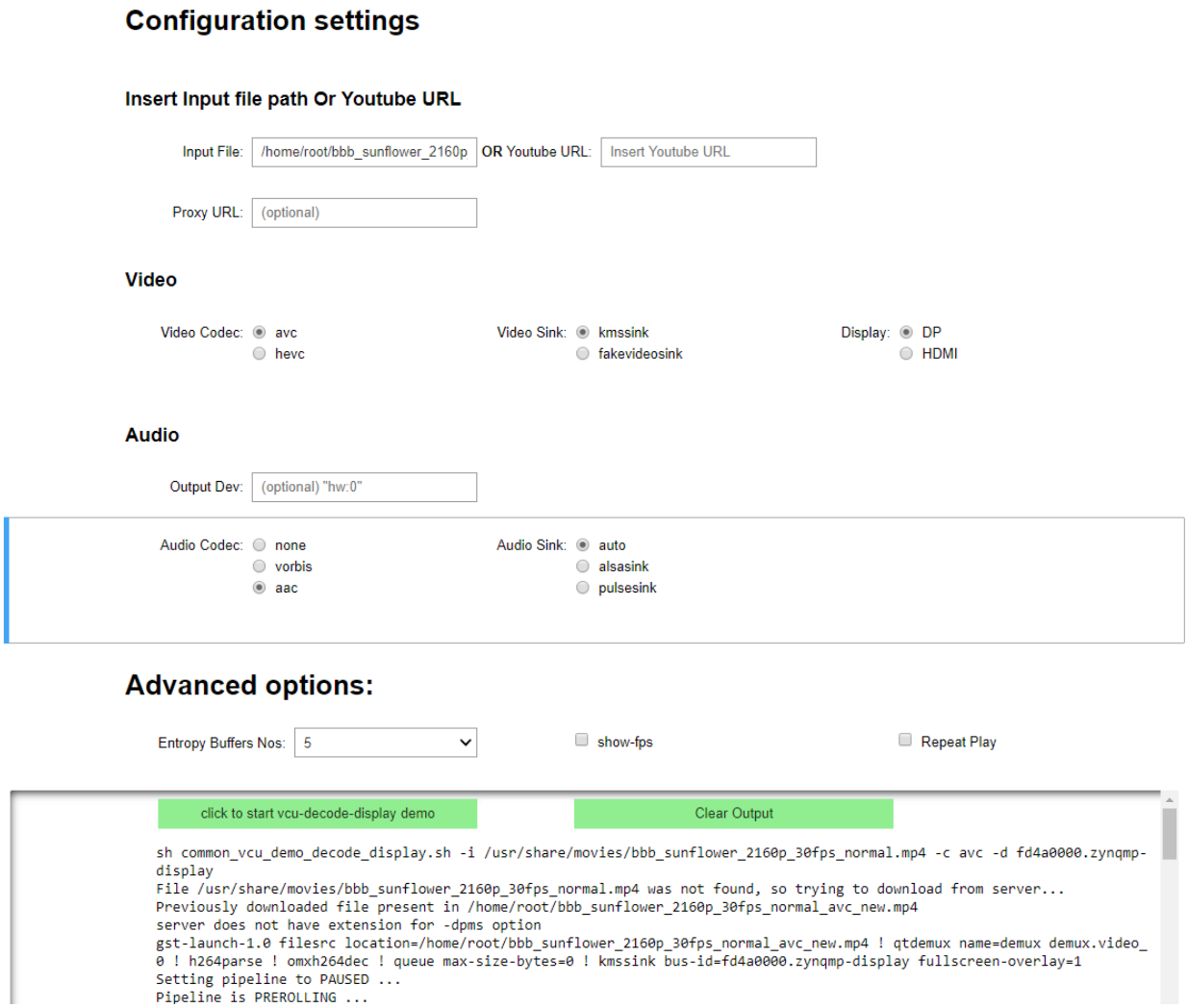


f. Click restart the kernel and run all.

g. Fill the data as per requirement. This is optional. If you do not fill or select data, the example runs with the default values.

h. Once the selection of parameters is done, click **Start Demo**.

i. The output of script running on the board will be displayed on the Jupyter console. For example, Decode→Display Jupyter notebook output is as shown in the following figure:



3. To clear the output log follow these steps:

   a. Press the stop button, in case Jupyter notebook is still running.

   b. Click **Clear Output**.

   ***Note*:** The output is cleared only when the example is stopped in-between or completed.

Determining Pulse Audio and ALSA Device Names

The audio device name (to provide with -i option) of audio source and playback device (to provide with `--audio-output`) can be found using `arecord` and `aplay` utilities respectively.

ALSA sound device names for capture devices

```
arecord -l
```

```
**** List of CAPTURE Hardware Devices ****
card 1: C920 [HD Pro Webcam C920], device 0: USB Audio [USB Audio]
Subdevices: 1/1
Subdevice #0: subdevice #0
In this example, card number of capture device is 1 and device id is 0
hence "hw:1,0" can be passed to alsasrc to refer to this capture device
using -i option.
```

In this example, card number 0 is being used for playback device for display port channel 0 and device id is 0, so `hw:0,0` can be used for selecting display port channel 0 as a playback device using the `--audio-output` option.

ALSA sound device names for playback devices

```
aplay -1
```

```
**** List of PLAYBACK Hardware Devices ****
card 0: monitor [DisplayPort monitor], device 0: (null) xilinx-dp-snd-codec-
dai-0 []
Subdevices: 1/1
Subdevice #0: subdevice #0
card 0: monitor [DisplayPort monitor], device 1: (null) xilinx-dp-snd-codec-
dai-1 []
Subdevices: 1/1
Subdevice #0: subdevice #0
```

In this example, card number 0 is being used for playback device for display port channel 0 and device id is 0, so `hw:0,0` can be used for selecting display port channel 0 as a playback device using `--audio-output` option.

PulseAudio sound device names for capture devices

```
pactl list short sources
```

```
alsa_input.usb-046d_HD_Pro_Webcam_C920_758B5BFF-02.analog-stereo ...
```

In this example, `alsa_input.usb-046d_HD_Pro_Webcam_C920_758B5BFF-02.analog-stereo` is the name of audio capture device which can be selected using -i option.

PulseAudio sound device names playback devices

```
pactl list short sinks
```

```
alsa_output.platform-fd4a0000.zynqmp-display_zynqmp_dp_snd_card. analog-
stereo ...
```

In this example, `alsa_output.platform-fd4a0000.zynqmp-display_zynqmp_dp_snd_card.analog-stereo` is the name of audio playback device which can be selected using --audio-output option.

XILINX

Section VI

Appendices

Codec Parameters for Different Use Cases

The codec parameters for different use cases are given in the following table:

Table 132: **Codec Parameters**

Use Case	Quality	Latency Mode	Encoder Settings	Decoder Settings
Recording, Transcoding	High	Normal	control-rate=variable qp-mode=auto min-qp=10 max-qp=51 initial-delay=500 cpb-size=1000 prefetch-buffer=TRUE periodicity-idr=240 gop-length=2*[fps]	internal-entropy-buffers=5 (9-10 if target-bitrate>=100Mbps) low-latency=0
Streaming, Video Conferencing	Low	Reduced Latency, Low Latency, Xilinx Low Latency	control-rate=low-latency target-bitrate=<25000 qp-mode=auto min-qp=10 max-qp=51 gop-mode=low-delay-p gdr-mode=horizontal/vertical initial-delay=500 cpb-size=1000 filler-data=0 num-slices=8 prefetch-buffer=TRUE max-picture-sizes='<2*[avg frame size],[avg frame size],0>' where [avg frame size] = target-bitrate/fps periodicity-idr=240 b-frames=0 loop-filter-mode=0 loop-filter-beta-offset=-6 loop-filter-alpha-c0-offset=-6	low-latency=1

QoS Configurations

Check the Read QoS, Write QoS, Read Commands Issuing Capability, and Write Commands Issuing Capability configuration of HP ports that interface the VCU with the PS DDR.

***Note*:** For VCU traffic, the QoS should be set as Best Effort (BE) and outstanding transaction count for read/write commands should be set to maximum, which is `0xF` for all AXI HP ports.

The AXI-QoS{3:0] lines behavior define three types of following Traffic in Decimal format on the AXI Bus.

Table 133: **Default QoS Values**

Traffic Class	Read QoS Value	Write QoS Value
Best Effort (BE)	0-3	0-7
Video (V)	4-11	8-15
Low Latency (LL)	12-15	N/A

Following is the reference QoS configuration script as per VCU recommendation.

```
#!/bin/bash

/sbin/devmem 0xfd380008 w 0x3 -> RDQoS for S_AXI_HP0_FPD
/sbin/devmem 0xfd390008 w 0x3 -> RDQoS for S_AXI_HP1_FPD
/sbin/devmem 0xfd3a0008 w 0x3 -> RDQoS for S_AXI_HP2_FPD
/sbin/devmem 0xfd3b0008 w 0x3 -> RDQoS for S_AXI_HP3_FPD

/sbin/devmem 0xfd38001c w 0x3 -> WRQoS for S_AXI_HP0_FPD
/sbin/devmem 0xfd39001c w 0x3 -> WRQoS for S_AXI_HP1_FPD
/sbin/devmem 0xfd3a001c w 0x3 -> WRQoS for S_AXI_HP2_FPD
/sbin/devmem 0xfd3b001c w 0x3 -> WRQoS for S_AXI_HP3_FPD

/sbin/devmem 0xfd380004 w 0xF -> RDISSUE for S_AXI_HP0_FPD
/sbin/devmem 0xfd390004 w 0xF -> RDISSUE for S_AXI_HP1_FPD
/sbin/devmem 0xfd3A0004 w 0xF -> RDISSUE for S_AXI_HP2_FPD
/sbin/devmem 0xfd3B0004 w 0xF -> RDISSUE for S_AXI_HP3_FPD

/sbin/devmem 0xfd380018 w 0xF -> WRISSUE for S_AXI_HP0_FPD
/sbin/devmem 0xfd390018 w 0xF -> WRISSUE for S_AXI_HP1_FPD
/sbin/devmem 0xfd3A0018 w 0xF -> WRISSUE for S_AXI_HP2_FPD
/sbin/devmem 0xfd3B0018 w 0xF -> WRISSUE for S_AXI_HP3_FPD
```

IRQ Balancing

Various multimedia use-cases involving video codecs such as audio/video conferencing, video-on-demand, playback, and record use-cases also involve multiple other peripherals such as ethernet, video capture pipeline related IPs icluding image sensor and image signal processing engines, DMA engines, and display pipeline related IP like video mixers and HDMI transmitters, which in turn use unique interrupt lines for communicating with the CPU.

In these scenarios, it becomes important to distribute the interrupt processing load across multiple CPU cores instead of utilizing the same core for all the peripherals/IP. Distributing the IRQ across CPU cores optimizes the latency and performance of the running use-case as the IRQ context switching and ISR handling load gets distributed across multiple CPU cores.

Each peripheral/IP is assigned a unique interrupt number by the Linux kernel. Whenever a peripheral or IP needs to signal something to the CPU (like it has completed a task or detected something), it sends a hardware signal to the CPU and the kernel retrieves the associated IRQ number and then calls the associated interrupt service routine. The IRQ numbers can be retrieved using the following command. This command also lists the number of interrupts processed by each core, the interrupt type, and comma-delimited list of drivers registered to receive that interrupt.

```
$cat /proc/interrupts
```

TheZynq UltraScale+ MPSoC has four CPU cores available. If running a plain PetaLinux image without any irqbalance daemon, then by default all IRQ requests are processed by CPU 0 by the Linux scheduler. To assign a different CPU core to process a particular IRQ number, the IRQ affinity for that particular interrupt needs to be changed. The IRQ affinity value defines which CPU cores are allowed to process that particular IRQ. For more information, see https://www.kernel.org/doc/Documentation/IRQ-affinity.txt.

By default, the IRQ affinity value for each peripheral is set to `0xf`, which means that all four CPU cores are allowed to process interrupt as shown in the following example using the IRQ number 42.

```
$cat /proc/irq/42/smp_affinity
output: f
```

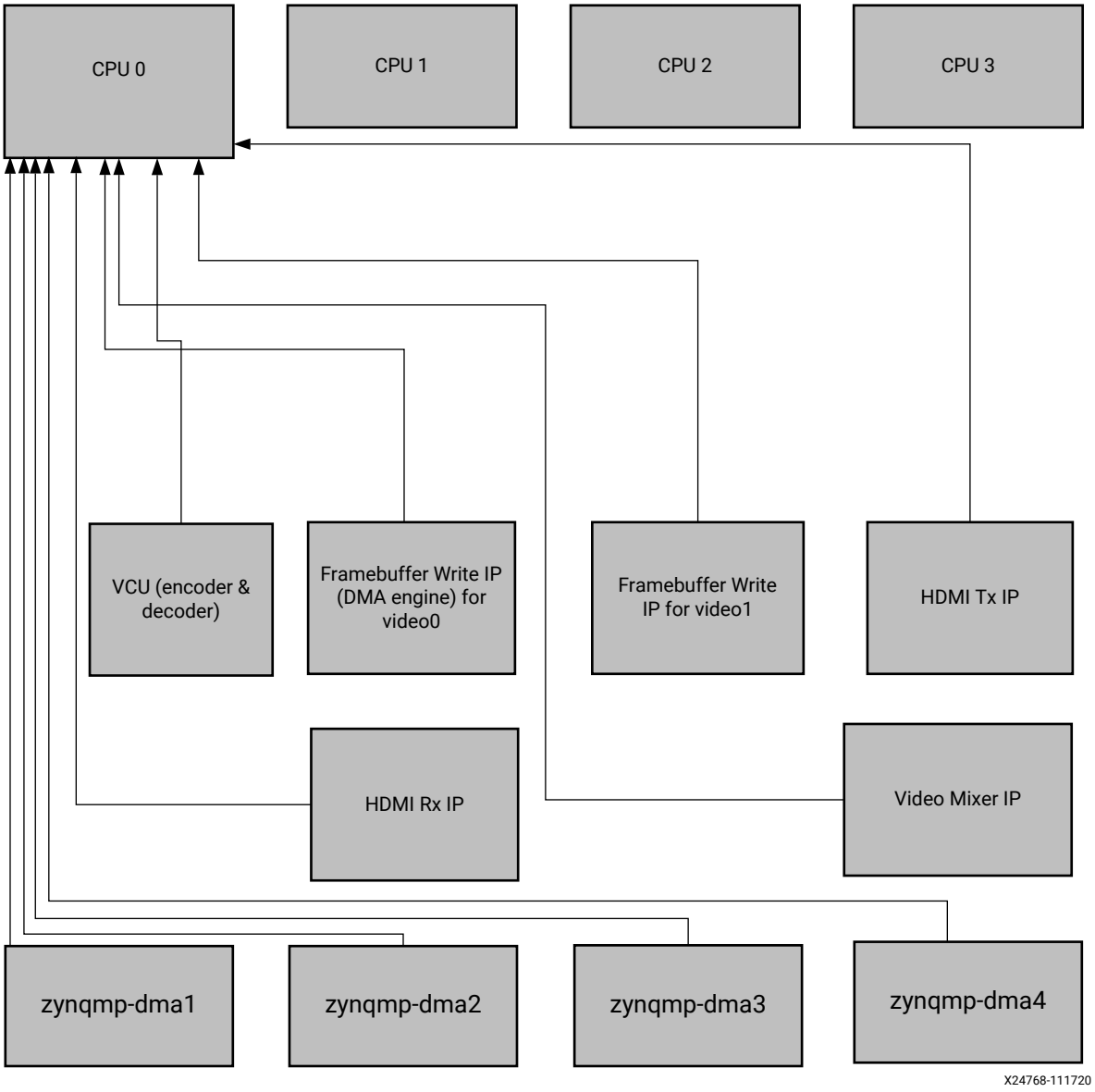
To restrict this IRQ to a CPU core n, you have to set a mask for only the n^{th} bit. For example, if you want to route to only CPU core 2, then set the mask for the second bit using the value `0x4`.

```
echo 4 > /proc/irq/42/smp_affinity
```

The following section shows how IRQ balancing can be performed before running a multistream video conferencing use-case that involves multiple peripherals and video IP.

As explained in **Encoder Features → DMA_PROXY module usage**, various DMA channels are used for constructing encoded output, which in turn also utilize different interrupt lines as depicted by the zynqmp-dma blocks in the following figure.

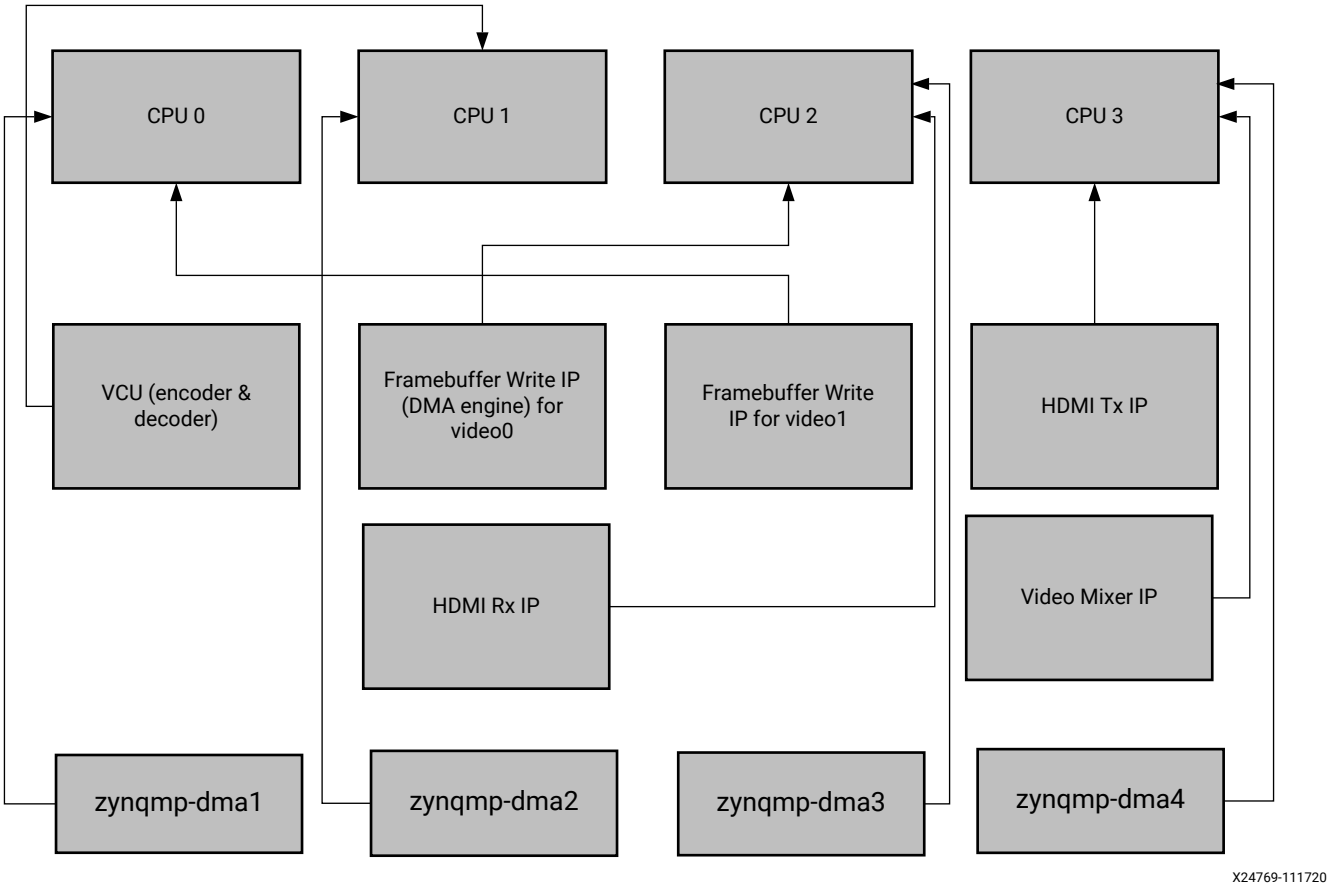
Figure 53: **Default IRQ Assignment to CPU Core 0**



As seen in the previous figure, all interrupt requests from different peripherals goes to CPU 0 by default.

To distribute the interrupt requests across different CPU cores as show in the following figure, follow these steps:

Figure 54: **Distributed Interrupt Layout**



1. Find the IRQ numbers for each of the above peripherals.

```
root@zcu106-zynqmp:~/llp2_2020.2_scripts/Serial/1080p/nv12# cat /proc/interrupts | grep
al5

 49:    1250127      47679          0          0     GICv2 127 Level     a0120000.al5d,
a0100000.al5e

root@zcu106-zynqmp:~/llp2_2020.2_scripts/Serial/1080p/nv12# cat /proc/interrupts | grep
xilinx_frame
 52:      18662          0        822          0     GICv2 122 Level     xilinx_framebuffer
 53:      19170          0          0          0  interrupt-controller@a0055000   3 Level
-level     xilinx_framebuffer
 54:      18825          0          0          0  interrupt-controller@a0055000   0 Level
-level     xilinx_framebuffer
 55:      18463          0          0          0  interrupt-controller@a0055000   1 Level
-level     xilinx_framebuffer
 57:          0          0          0          0     GICv2 121 Level     xilinx_framebuffer

root@zcu106-zynqmp:~/llp2_2020.2_scripts/Serial/1080p/nv12# cat /proc/interrupts | grep
```

```
xilinx-hdmi
 56:     544834          0    4911768          0     GICv2 123 Level     xilinx-hdmi-rx
 58:      86730          0          0     813482     GICv2 125 Level     xilinx-hdmitxss

root@zcu106-zynqmp:~/llp2_2020.2_scripts/Serial/1080p/nv12# cat /proc/interrupts | grep
mixer
 59:      86752          0          0     814292     GICv2 128 Level     xlnx-mixer

root@zcu106-zynqmp:~/llp2_2020.2_scripts/Serial/1080p/nv12# cat /proct/interrupts | grep
zynqmp-dma
12:   42151036          0          0          0     GICv2 156 Level         zynqmp-dma
13:   31494805   10644207          0          0     GICv2 157 Level     zynqmp-dma
14:   31483922          0   10643127          0     GICv2 158 Level     zynqmp-dma
15:   31518024          0          0   10595920     GICv2 159 Level     zynqmp-dma
NOTE: Here there are multiple zynqmp-dma interrupt lines so to check which ones are getting
utilized,
you first need to run the usecase and then check which interrupt lines are getting utilized.
```

The numbers on the left are the IRQ numbers for the respective peripherals.

2. Assign CPU 1 to VCU IRQ with number 49.

```
echo 2 > /proc/irq/49/smp_affinity #VCU
```

3. Assign CPU 2 to HDMI RX and the framebuffer write IP to CPU 2

```
echo 4 > /proc/irq/52/smp_affinity #Primary HDMI Rx

echo 4 > /proc/irq/56/smp_affinity #Primary HDMI Tx
```

4. Assign CPU 3 to HDMI TX and Video mixer IP

```
echo 8 > /proc/irq/58/smp_affinity #Tx

echo 8 > /proc/irq/59/smp_affinity #Mixer
```

By default, the interrupts for video1 xilinx_framebuffer DMA engine and various other peripherals are already being processed by CPU 0 so there is no need to modify the smp_affinity for the same. Using the previous commands, the IRQ is distributed as per the scheme mentioned in the previous figure, which can also be seen by running the following command when the use-case is running and observing whether interrupts for the peripherals are going to respective CPU cores as intended or not. Likewise, similar scheme of distributing interrupts can be followed for other use-cases too depending upon the peripherals being used, system load, and intended performance.

```
$ cat /proc/interrupts
```

5. Assign a unique CPU to each zynqmp-dma channel if possible

```
echo 0 > /proc/irq/12/smp_affinity #zynqmp-dma1
echo 2 > /proc/irq/13/smp_affinity #zynqmp-dma2
echo 4 > /proc/irq/14/smp_affinity #zynqmp-dma3
echo 8 > /proc/irq/15/smp_affinity #zynqmp-dma4
```

By default the interrupts for other peripherals will be processed by cpu 0 so there is no need to modify the smp_affinity for the same. Using the preceding commands, the IRQ will get distributed as per the scheme mentioned in , which can also be seen by running the following command when the use-case is running:

```
 cat /proc/interrupts
 12:       42151036          0                  0               0     GICv2 156 Level
zynqmp-dma
 13:       31494805      10644207          0               0     GICv2 157 Level     zynqmp-
dma
 14:       31483922          0             10643127    0     GICv2 158 Level     zynqmp-dma
 15:       31518024          0                  0          10595920  GICv2 159 Level
zynqmp-dma

 49:     1250127      47679          0          0     GICv2 127 Level     a0120000.al5d,
a0100000.al5e
 52:       18662          0        822          0     GICv2 122 Level     xilinx_framebuffer
 53:       19170          0          0          0  interrupt-controller@a0055000   3 Level
-level     xilinx_framebuffer
 54:       18825          0          0          0  interrupt-controller@a0055000   0 Level
-level     xilinx_framebuffer
 55:       18463          0          0          0  interrupt-controller@a0055000   1 Level
-level     xilinx_framebuffer
 56:      544834          0    5528886          0     GICv2 123 Level     xilinx-hdmi-rx
 57:           0          0          0          0     GICv2 121 Level     xilinx_framebuffer
 58:       86730          0          0     915677     GICv2 125 Level     xilinx-hdmitxss
 59:       86752          0          0     915711     GICv2 128 Level     xlnx-mixer
 60:       42021          0          0          0     GICv2 138 Level     a00c0000.sync_ip
```

Upgrading

2020.2 VCU Ctrl-SW API Migration

This section provides migration guide from 2020.1 to 2020.2 release. It covers list of modified and newly added VCU control software API for 2020.2 release. For more details, see:

- Readme at Doxygen folder in VCU control software repository https://github.com/Xilinx/vcu-ctrl-sw/tree/master/Doxygen.
- See Xilinx VCU Control Software API.

Modified APIs

The following table contains list of modified encoder and decoder control software APIs in 2020.2 release.

Table 134: **Modified Encoder and Decoder Control Software APIs and Variables**

2020.1 Release	2020.2 Release	Motives
bSplitInput is of bool type in AL_TDecSettings. It is true for decoder in split input mode and false for unsplit input mode.	bSplitInput is renamed to eInputMode of enum type and it can be assigned as enum values like AL_DEC_UNSPLIT_INPUT, AL_DEC_SPLIT_INPUT	Simpler and clearer way to handle such settings.
void AL_Decoder_SetParam(AL_HDecoder hDec, bool bConceal, bool bUseBoard, int iFrmID, int iNumFrm, bool bForceCleanBuffers, bool shouldPrintFrameDelimiter);	“bool bConceal” is removed. bool bUseBoard is now replaced with const char* sPrefix. Its value can be "Fpga" or "Ref". void AL_Decoder_SetParam(AL_HDecoder hDec, const char* sPrefix, int iFrmID, int iNumFrm, bool bForceCleanBuffers, bool bShouldPrintFrameDelimiter);	Removed not used parameters and using informative text instead of bool datatype to increase readably
In decoder,AL_DecGetAllocSize_Frame_Y/ AL_DecGetAllocSize_Frame_UV	AL_DecGetAllocSize_Frame_PixPlane	As per New chroma planes in CtrlSW, planar buffers must be allocated with Plane-U/Plane-V (Plane-UV usage restricted to semiplanar
In encoder, AL_GetAllocSizeSrc_Y/ AL_GetAllocSizeSrc_UV	AL_GetAllocSizeSrc_PixPlane	As per New chroma planes in CtrlSW, planar buffers must me allocated with Plane-U/Plane-V (Plane-UV usage restricted to semiplanar

Table 134: **Modified Encoder and Decoder Control Software APIs and Variables** *(cont'd)*

2020.1 Release	2020.2 Release	Motives
Configuration file parser API, MaxPictureSize() used to take values in Kb.	CFG file API changes as MaxPictureSizeInBits (I,P,B) and takes values in bits instead of Kb	To have better control.

New APIs

The following table contain list of newly added encoder and decoder control software APIs in 2020.2 release.

Table 135: **New APIs**

2020.2 Release	API Description
Added zRcPluginDmaSize and pRcPluginDmaContext in AL_TEncChanParam	Scheduler use at encoder side
iParsingID is added in AL_CB_EndParsing callback.	Retrieves the iParsingID. The iParsingID corresponds to the id in the AL_HandleMetaData associated with the buffer*
int AL_Buffer_AllocateChunkNamed(AL_TBuffer* pBuf, size_t zSize, char const* name);	Allocates and binds a new memory chunk to the buffer
Add (char const* name) the name of the buffer as extra argument in AL_PixMapBuffer_Allocate_And_AddPlanes()	Debugs and tracks allocation
Add AL_TRANSFER_CHARAC_BT_2100_HLG as new enum value in AL_ETransferCharacteristics	Added support for new metadata
Add AL_TDynamicMeta_ST2094_10 dynamic SEI in AL_THDRSEIs structure	Added support for new HDR Related SEIs
Add AL_TDynamicMeta_ST2094_40 dynamic SEI in AL_THDRSEIs structure	Added support for new HDR Related SEIs
Add support for new added HDR related SEIs in AL_Encoder_SetHDRSEIs()	Function AL_Encoder_SetHDRSEIs can be called dynamically to change the value of ST2094_10 and ST2094_40 SEIs
Added AL_Plane_GetBufferPixelPlanes / AL_Plane_GetBufferPlanes	Required planes for a FourCC can be obtained with these APIS
Add new file `lib_common/PicFormat.h`	Lists the formats of the pictures extracted from `lib_common/SliceConsts.h`
Add new file `lib_common/CodecHook.h`	Hooks the code to help user applications

2020.1 VCU Ctrl-SW API Migration

This section provides a list of modified, newly added VCU control software APIs in 2020.1 release in comparison to 2019.2 release. For more details, see:

- Readme in the Doxygen folder in VCU control software repository: https://github.com/Xilinx/vcu-ctrl-sw/tree/master/Doxygen.
-

- See Xilinx VCU Control Software API.

Global API Changes

- Moving functions into clearer and more specifics files, which includes profiles and nuts.
- Macros are becoming static inline function when it is possible and/or useful.
- Inline become INLINE when it is possible and/or useful.

Problems in Stabilizing the API

Continued modification of AL_TEncSettings and the AL_TEncChanParam is required because these structures are used internally and many of their members can be obsolete when some internal algorithm changes. Moreover, new members cannot be added without deleting the now irrelevant ones for backward compatibility as these structures are sent to the MCU and thus, the space taken by each of these structures is relevant. This means that for the API to become stable and for semantic versioning to become 1.x (see https://semver.org/ for more information), the processes of getting the encoder settings from user to hide the affected structures from the user has to be changed. This can be done with an opaque settings object that hides the actual structures and is accessed using methods.

The luma and chroma planes interface needs to be further refined before freezing it as it does not provide the possibility to specify different buffers for the luma and chroma as is.

Modified APIs

The following table contains list of modified encoder and decoder control software APIs in 2020.1 release.

Table 136: **Modified Encoder and Decoder Control Software APIs and Variables**

2019.2 Release	2020.1 Release	Motives
AL_SrcMetaDatauses offsets to specify the chroma and the luma part of the buffer.	AL_TSrcMetaDatabecomes AL_TPixMapMetaData. It allows storing pixel planes on different chunks	This simplifies the handling of the luma and chroma planes for the user. This enables, in a further release to support separate buffers forthe luma and forthe chroma, relaxing the constraint on memory allocation and adding flexibility.
The AL_TBufferapi does not allocate the memory itself and only provide an easy access to it via its API. It wraps a memory buffer allocated using AL_TAllocatorapi. The AL_TBuffer takes ownership of the memory buffer and frees it at destruction.	TheAL_TBuffer API does not allocate the memory itself and only provide an easy access to it via its API. It wraps memory buffers allocated using the AL_TAllocator API. The AL_TBuffer takes ownership of the memory buffers and frees it at destruction.	The AL_TBuffer API provides an easy access to one or multiple memory chunks. Chunks of an AL_TBufferare memory buffers that share the same lifetime, reference counting mechanism, and custom user information, through metadata.
AL_META_TYPE_SOURCE	AL_META_TYPE_PIXMAP	Simpler and clearer way to handle src

Table 136: **Modified Encoder and Decoder Control Software APIs and Variables** *(cont'd)*

2019.2 Release	2020.1 Release	Motives
AL_TSrcMetaData* AL_SrcMetaData_Create(AL_TDimension tDim, AL_TPlane tYPlane, AL_TPlane tUVPlane, TFourCC tFourCC)	AL_TPixMapMetaData* AL_PixMapMetaData_Create(AL_TDimension tDim, AL_TPlane tYPlane, AL_TPlane tUVPlane, TFourCC tFourCC)	Adapt to new API
AL_TSrcMetaData* AL_SrcMetaData_CreateEmpty(TFourCC tFourCC)	AL_TPixMapMetaData* AL_PixMapMetaData_CreateEmpty(TFourCC tFourCC)	Adapt to new API
void AL_SrcMetaData_AddPlane(AL_TSrcMetaData* pMeta, AL_TPlane tPlane, AL_EPlaneId ePlaneId)	void AL_PixMapMetaData_AddPlane(AL_TPixMapMetaData* pMeta, AL_TPlane tPlane, AL_EPlaneId ePlaneId)	Adapt to new API
AL_TSrcMetaData* AL_SrcMetaData_Clone(AL_TSrcMetaData* pMeta)	AL_TPixMapMetaData* AL_PixMapMetaData_Clone(AL_TPixMapMetaData* pMeta)	Adapt to new API
int AL_SrcMetaData_GetOffsetY(AL_TSrcMetaData* pMeta)	int AL_PixMapMetaData_GetOffsetY(AL_TPixMapMetaData* pMeta)	Adapt to new API
int AL_SrcMetaData_GetOffsetUV(AL_TSrcMetaData* pMeta)	int AL_PixMapMetaData_GetOffsetUV(AL_TPixMapMetaData* pMeta)	Adapt to new API
int AL_SrcMetaData_GetChromaSize(AL_TSrcMetaData* pMeta)	int AL_PixMapMetaData_GetChromaSize(AL_TPixMapMetaData* pMeta)	Adapt to new API
TScheduler	AL_IEncScheduler	Clearer definition
TRecPic	AL_TRecPic	Add missing prefix

New APIs

The following table contain list of newly added encoder and decoder control software APIs in 2020.1 release.

Table 137: **New APIs**

2020.1 Release	API Description
AL_TBuffer* AL_Buffer_CreateEmpty(AL_TAllocator* pAllocator, PFN_RefCount_CallBack pCallBack)	Creates a buffer structure without bind memory
int AL_Buffer_AllocateChunk(AL_TBuffer* pBuf, size_t zSize)	Allocate and bind a new memory chunk to the buffer
int AL_Buffer_AddChunk(AL_TBuffer* pBuf,AL_HANDLEhChunk, size_t zSize)	Add an already allocated chunk to a buffer
static AL_INLINEint8_t AL_Buffer_GetChunkCount(const AL_TBuffer* pBuf)	Get the number of chunks belonging to the buffer
static AL_INLINEbool AL_Buffer_HasChunk(const AL_TBuffer* pBuf, int8_t iChunkIdx)	Check if a buffer has a chunk with specific index
uint8_t* AL_Buffer_GetDataChunk	Gets a pointer to a chunk data
size_t AL_Buffer_GetSize(const AL_TBuffer* pBuf)	Gets the buffer size. If the buffer contains multiple chunks, return the size of the first chunk

Table 137: **New APIs** *(cont'd)*

2020.1 Release	API Description
size_t AL_Buffer_GetSizeChunk(const AL_TBuffer* pBuf, int iChunkIdx)	Gets the size of a chunk
AL_META_TYPE_RATECTRL	Rate control statistics meta
uint32_t AL_GetAllocSizeSrc_Y(AL_ESrcMode eSrcFmt, int iPitch, int iStrideHeight);	Retrieves the size of the luma component of a YUV frame buffer
uint32_t AL_GetAllocSizeSrc_UV(AL_ESrcMode eSrcFmt,int iPitch, int iStrideHeight, AL_EChromaModeeChromaMode)	Retrieves the size of the choma component of a YUV frame buffer
AL_CB_EndParsing	It is called every time an input buffer is parsed by the hardware. If bSplitInput is disable, pParsedFrame should not have AL_THandleMetaData
bool AL_Encoder_SetFreqIDR(AL_HEncoderhEnc, int iFreqIDR)	Changes the IDR frequency. If the new frequency is shorter than the number of frames already encoded since the last IDR, insert and IDR as soon as possible. Otherwise, the next IDR is inserted when the new IDR frequency is reached.
bool AL_Encoder_SetQPBounds(AL_HEncoderhEnc, int16_t iMinQP, int16_t iMaxQP)	Changes the bounds of the QP set by the rate control
bool AL_Encoder_SetQPIPDelta(AL_HEncoderhEnc, int16_t uIPDelta);	Changes the QP delta between I frames and P frames
bool AL_Encoder_SetQPPBDelta(AL_HEncoder hEnc, int16_t uPBDelta)	Changes the QP delta between P frames and B frames

Removed APIs

The following table contains a list of removed encoder and decoder control software APIs for the current release.

Table 138: **Removed APIs**

Removed API	Motives
void AL_Buffer_SetData(const AL_TBuffer* pBuf, uint8_t* pData)	Not used anymore
void AL_CopyYuv(AL_TBuffer const* pSrc, AL_TBuffer* pDst)	Not used anymore

2019.2 VCU Ctrl-SW API Migration

This section provides migration guide from 2019.1 to 2019.2 release. It covers list of modified and newly added VCU control software API for 2019.2 release. For more details, see:

- Readme at Doxygen folder in VCU control software repository
- See Xilinx VCU Control Software API in this chapter

Modified APIs

The following table contains a list of modified encoder and decoder control software APIs in 2019.2 release.

Table 139: **Modified Encoder and Decoder Control Software APIs and Variables**

2019.1 Release	2019.2 Release	Motives
XXX	XXX	Namespacing: add missing AL_ in front of enum/struct (i.e DEFAULT_LDA → AL_DEFAULT_LDA, ...)
inline	AL_INLINE	Cross-platform compilation
#define XXX	static AL_INLINE XXX	Force type ? reduce user errors
uint32_t AL_GetAllocSizeSrc(AL_TDimension tDim, uint8_t uBitDepth, AL_EChromaMode eChromaMode, AL_ESrcMode eSrcFmt, int iPitch, int iStrideHeight)	uint32_t AL_GetAllocSizeSrc(AL_TDimension tDim, AL_EchromaMode eChromaMode, AL_ESrcMode eSrcFmt, int iPitch, int iStrideHeight)	Bitdepth is not needed anymore
uMaxPictureSize	pMaxPictureSize[3]	Support Max Picture for I, P, B
AL_QPCtrlMode	AL_EQpCtrlMode which decide the QP Ctrl mode and the AL_EqpTableMode which decide table type	Allow lot of combinations and reduce user errors. Remove enum which should not appear in the API (i.e. BORDER_QP, ...)
Old registers definition (lib_ip_ctrl)	New registers definition (lib_ip_ctrl)	Genericness code. Huge increase in flexibility and help to add potential new features.
int Rtos_DriverPoll(void* drv, int timeout)	int Rtos_DriverPoll(void* drv, int timeout, unsigned long flags)	Add flag into polling.
bool bEnableFillerData	AL_EfillerCtrlMode eEnableFillerData	Select if filler should be moved in the CtrlSW or by the application

New APIs

This table contain list of newly added encoder and decoder control software APIs in 2019.2 release.

Table 140: **New APIs**

2019.2 Release	API Description
void Rtos_SetCurrentThreadName(const char* pThreadName);	Retrieve thread during debugging
bool AL_Encoder_SetLoopFilterBetaOffset(AL_HEncoder hEnc, int8_t iBetaOffset)	Changes dynamically the loop filter beta offset
bool AL_Encoder_SetLoopFilterTcOffset(AL_HEncoder hEnc, int8_t iTcOffset)	Changes dynamically the loop filter TC offset
bool AL_Encoder_SetHDRSEIs(AL_HEncoder hEnc, AL_THDRSEIs* pHDRSEIs)	Specify HDR SEIs to insert in the bitstream
bool bSplitInput	Send stream data by decoding unit

Table 140: **New APIs** *(cont'd)*

2019.2 Release	API Description
static AL_INLINE bool AL_IS_QP_TABLE_REQUIRED(AL_EQpTableMode eMode)	
static AL_INLINE bool AL_IS_AUTO_OR_ADAPTIVE_QP_CTRL(AL_EQpCtrlMode eMode)	
AL_THDRMetaData	Retrieve HDR from buffer
EndParsingCallback (decoder side)	You can add a callback on EndParsing (usefull with bSplitInput = true)

2019.1 VCU Ctrl-SW API Migration

This section provides migration guide from 2018.3 to 2019.1 release. It covers list of modified and newly added VCU control software API for 2019.1 release. For more details, see:

- Readme at Doxygen folder in VCU control software repository
- See Xilinx VCU Control Software API in this chapter

Global API Changes

- Added `AL_VERSION_MAJOR`, `AL_VERSION_MINOR`, and `AL_VERSION_STEP` defines.
- Since the semantic versioning continues to be in 0.x, the API is not stable.
- Macros are becoming static inline function when it is possible / useful.
- All the prototypes of the functions are now complete prototype; functions with no parameters are declared using (void) to ensure signature safety.

Problems to Stabilize the API

You cannot stop modifying the AL_TEncSettings and the AL_TEncChanParam because these structures are used internally and many of their members can be obsolete when some internal algorithm changes. New members cannot be added without deleting obsolete/ irrelevant members for backward compatibility as these structures are sent to the MCU and thus, the space taken by each of these structures is relevant.

This means that for the API to become stable and for semantic versioning to become 1.x (See https://semver.org/) we need to change the way we get the encoder settings from the user to hide the affected structures from the user. This can be done with an opaque settings object that will hide the actual structures and that will be accessed using methods. The luma and chroma planes interface needs to be further refined before freezing it as they do not let you specify different buffers for the luma and chroma.

Modified APIs

This table contains list of modified encoder and decoder control software APIs in 2019.1 release.

Table 141: **Modified Encoder and Decoder Control Software APIs and Variables**

2018.3 Release	2019.1 Release	Motives
AL_SrcMetaData uses offsets to specify the chroma and the luma part of the buffer.	AL_SrcMetaData uses planes to specify the chroma and the luma part of the buffer	This simplifies the handling of the luma and the chroma planes. In future releases, this will support separate buffers for the luma and the chroma, relaxing the constraint on memory allocation and adding flexibility. This removes the AL_ToffsetYC and AL_Tpitches structures as the data is now held in AL_Tplane. This makes the library easier to use beginners as this api looks like what is used in V4L2 or in gstreamer.
AL_DPB_LOW_REF	AL_DPB_NO_REORDERING	**Defect**: The DPB_LOW_REF does not lower the number of references, it tells the DPB that no reordering will occur and that the specification of the dpb can be changed to ensure lower latency.
int AL_Encoder_AddSei(AL_HEncoder hEnc, AL_TBuffer* pStream, bool isPrefix, int iPayloadType, uint8_t* pPayload, int iPayloadSize)	int AL_Encoder_AddSei(AL_HEncoder hEnc, AL_TBuffer* pStream, bool isPrefix, int iPayloadType, uint8_t* pPayload, int iPayloadSize, int iTempId)	

New APIs

This table contain list of newly added encoder and decoder control software APIs in 2019.1 release.

Table 142: **New APIs**

2019.1 Release	API Description
bool AL_Encoder_SetInputResolution(AL_HEncoder hEnc, AL_TDimension tDim)	Changes the input resolution dynamically. This is related to the dynamic resolution change feature.
AL_MetaData_Clone(AL_TMetaData* pMeta)	This function becomes part of the vtable of the AL_TMetadata, effectively becoming a copy constructor.

Removed APIs

This table contains a list of removed encoder and decoder control software APIs for the current release.

Table 143: **Removed APIs**

Removed API	Motive
AL_ERR_RESOLUTION_CHANGE	The resolution change is now supported.

Verification, Compliance, and Interoperability

Verification

Tests included:

- ctrlsw and GStreamer file based encoder and decoder usecases
- Serial Live usecases (functional and performance)
- Multistream live usecases (functional and performance)
- Feature based testing (ctrlsw and gstreamer)
- Video streaming testing
- OOB testing
- Customer error streams
- Latency measurement
- Maxbitrate measurement
- Audio testing

Compliance Testing

All possible supported VCU encoder profile/level streams are decoded successfully using the JM/HM reference decoder and md5sum of the decoded output is matched with the VCU decoder.

Interoperability

VCU encoded streams have been successfully decoded using the JM decoder. Third party encoded streams (for example, the Argon HEVC test suite) have been successfully decoded using the VCU decoder.

XAVC

- **Verification:** Tests included:

- Encoder tests
- Decoder tests

- **Compliance:**
  - Encoding streams using the VCU encoder with XAVC profile and providing it as an input to the Sony format verifier for checking the compliance with their standards.
  - Decoding XAVC standard .mxf files for validating decoder functionalities.

Chapter 43

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 provide supplemental material useful with this product guide:

1. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
2. *Vivado Design Suite User Guide: Designing with IP* (UG896)
3. *Vivado Design Suite User Guide: Getting Started* (UG910)
4. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
5. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
6. *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899)
7. *Vivado Design Suite User Guide: Implementation* (UG904)
8. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906)
9. *AXI Interconnect LogiCORE IP Product Guide* (PG059)
10. *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150)
11. *UltraScale Architecture SelectIO Resources User Guide* (UG571)
12. *Zynq UltraScale+ MPSoC Production Errata* (EN285)
13. *Zynq UltraScale+ Device Technical Reference Manual* (UG1085)
14. *Zynq UltraScale+ Device Register Reference* (UG1087)
15. *Zynq UltraScale+ MPSoC Data Sheet: Overview* (DS891)
16. *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925)
17. *Zynq UltraScale+ MPSoC Processing System LogiCORE IP Product Guide* (PG201)
18. *Video Scene Change Detection LogiCORE IP Product Guide* (PG322)
19. *PetaLinux Tools Documentation: Reference Guide* (UG1144)
20. OpenMax Integration Layer
21. GStreamer
22. *Zynq UltraScale+ MPSoC ZCU106 Video Codec Unit Targeted Reference Design User Guide* (UG1250)
23. *Multimedia User Guide* (UG1449)

Training Resources

1. Designing FPGAs using the Vivado Design Suite 1

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
10/27/2021 Version 1.2	
Clocking	Updated section to mention PLDDR memory support.
Customizing the VCU DDR4 Controller	Updated the section.
Port connection recommendations for different use cases with VCU DDR Memory Controller	Added a new section.
Navigating Content by Design Process	Added a new section.
Gstreamer and V4L2 Formats	Updated the section.
Constraining the Core	Added a new table.
06/16/2021 Version 1.2	
Section V: Application Software Development	Added the HLG Support topic in the VCU Encoder Features section.
11/24/2020 Version 1.2	
Section I: H.264/H.265 Video Codec Unit v1.2	Updated the Customizing and Generating the Core, Encoder Buffer , and Enabling PL-DDR for VCU sections.
Section II: Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP v1.1	Updated the Resource Utilization, Designing with the Core, and Customizing the VCU DDR4 Controller sections.
Section III: VCU Sync IP v1.0	Updated the Resource Utilization, Port Descriptions, Sync IP Software Programming Model sections.
Section IV: Performance and Debugging	Updated Xilinx Low Latency Limitations.
Section V: Application Software Development	Updated Encoder and Decoder Software Features, GStreamer Pipelines, VCU Encoder Features, and VCU Control Software sections. Added the HDR10 section
Section VI: Appendices	Added three new sections: IRQ Balancing, QoS Configurations, and 2020.2 VCU Ctrl-SW API Migration.
08/17/2020 Version 1.2	
General updates	The content in this document was reorganized into the following sections: • Section I: H.264/H.265 Video Codec Unit v1.2 • Section II: Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP v1.1 • Section III: VCU Sync IP v1.0 • Section IV: Performance and Debugging • Section V: Application Software Development • Section VI: Appendices
VCU Encoder Features	Updated the XAVC section.
06/03/2020 Version 1.2	
Section III: VCU Sync IP v1.0	Added audio support for Xilinx® low-latency. Updated sections: Software Flow Overview, Software Architecture and Buffer Synchronization Mechanism, and Connecting the Sync IP with the VCU LogiCORE IP.

Section	Revision Summary
Latency in the VCU Pipeline	Added Xilinx Low Latency Limitations, Encoder and Decoder Latencies with Xilinx Low Latency Mode, and Recommended Parameters for Xilinx Low-latency Mode. Updated pipelines in Usage and Latency.
2020.1 VCU Ctrl-SW API Migration	Added new section.
Region of Interest Encoding	Updated section.
Max-Bitrate Benchmarking	Added new section.
XAVC	Updated section.
GStreamer Pipelines	Updated pipelines in this section.
General Updates	N/A
12/10/2019 Version 1.2	
VCU Encoder Features	Updated section
11/22/2019 Version 1.2	
Resource Utilization	Updated section
Section III: VCU Sync IP v1.0	Added new chapter
VCU Latency Modes	Added Xilinx low-latency mode section
VCU Encoder Features	Added new features
Running Examples with Jupyter Notebook	Added new section
2019.2 VCU Ctrl-SW API Migration	Added new section
Others	Updated design flow steps for 32-stream support, updated snapshots and added supported resolutions for 32-stream. Updated simulation section for H.264/H.265 Video Codec Unit and DDR4 controller. Updated Gstreamer, OMX and VCU software links for 2019.2. Updated Encoder and Decoder Gstreamer parameters. Updated Vertical GDR mode MV (motion vector) limitation in GDR Intra Refresh. Updated Ctrl-SW and OMX API details for Frameskip support. Updated 32-Streams Support section. Updated Control Software Encoder parameters. Updated encoder and decoder register address from absolute to relative address. Updated supported RAM parts and configuration for DDR4controller. Updated DDR4 controller IP version to 1.1. Added Chapter 7, VCU Sync IP v1.0 for ultra low latency support. Updated Clocking and Resets section for sample FBDIV calculations and signal vcu_resetn recommendations.
05/22/2019 Version 1.2	
2019.1 VCU Ctrl-SW API Migration	Added section
VCU Latency Modes	Added section
VCU Encoder Features	Added new features: Dynamic Resolution Change at VCU Encoder/Decoder, Frame skip support for VCU Encoder, 32-streams support, DCI-4k Encode/Decode, Temporal-ID

Section	Revision Summary
Others	Updated Enable PL DDR section. Updated CPB Size Guidelines. Updated Gstreamer, VCU Version. Updated Memory Footprint Calculation. Updated all VCU kernel module details. Updated Encoder and Decoder Gstreamer parameters. Added more information on behavior when MinQP=AUTO. Updated Default values and additional parameters at Control software level. Replaced videoparse with rawvideoparse. Updated Constraints, Limitation for VCU Features. Added information for additional memory requirement for AVC 4k. Updated Cache memory requirement.
12/05/2018 Version 1.2	
Encoder Block , Decoder Block	Added encoder and decoder block diagrams
Encoder Buffer Requirements	Added section
Decoder Buffer Requirements	Added section
Section II: Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP v1.1	Added chapter
VCU Encoder Features	Added new features: GDR Intra Refresh, LongTerm Ref. Picture, Adaptive GOP, Insertion of SEI data at Encoder, SEI Decoder API, Dual pass Encoding, Scene change detection, Interlaced video.
Input Parameters	Updated parameters
Settings Parameters	Updated parameters
Register Space	Updated Soft IP Registers table.
VCU Out of the Box Examples	Updated examples
Others	Added details of various latency mode. Added Optimum VCU Encoder parameters for use-cases. Updated software prerequisites. Added information on utilizing ROI and LoadQP function from live streaming sources. Updated control software debugging information and error codes. Updated Registers and GStreamer elements. Replaced AL_Encoder.exe and AL_Decoder.exe applications with ctrlsw_encode and ctrlsw_decoder, respectively. Replaced omx_encoder.exe and omx_decoder.exe applications with omx_encode and omx_decoder, respectively. Increased stream support from 8 to 32.
06/06/2018 Version 1.1	
General Updates	N/A
04/04/2018 Version 1.1	
Xilinx VCU Control Software API	Updateded section

Section	Revision Summary
Others	Added instructions for configuring the VCU core. Updated supported profiles and options. Changed gop-freq-idr to periodicity-idr. Documented usage of the sample applications.
2/20/2017 Version 1.0	
Reorganized content and updated API	N/A
10/04/2017 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright