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

LogiCORE IP Product Guide

Vivado Design Suite

PG252 May 22, 2019

XILINX

Table of Contents

Introduction

The Xilinx® LogiCORE™ IP H.264/H.265 Video Codec Unit (VCU) core for Zynq® UltraScale+™ MPSoC devices is capable of performing video compression and decompression of simultaneous video streams at resolutions up to 3840×2160 pixels at 60 frames per second (4K UHD at 60Hz). 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, biomedical instrumentation, etc.

Features

- 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 60Hz

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

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq® UltraScale+™ MPSoC Family EV Devices
Supported User Interfaces	AXI4-Lite, AXI4-Memory Mapped
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	Verilog
Constraints File	Xilinx Design Constraints (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	
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.

Features (Continued)

- Progressive support for H.264 and H.265; Interlace support for H.265
- Video input:
  - YCbCr 4:2:2, YCbCr 4:2:0, and Y-only (monochrome)
  - 8- and 10-bit per color channel

Overview

Introduction

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. Figure 1-1 shows the top-level block diagram with the VCU core.

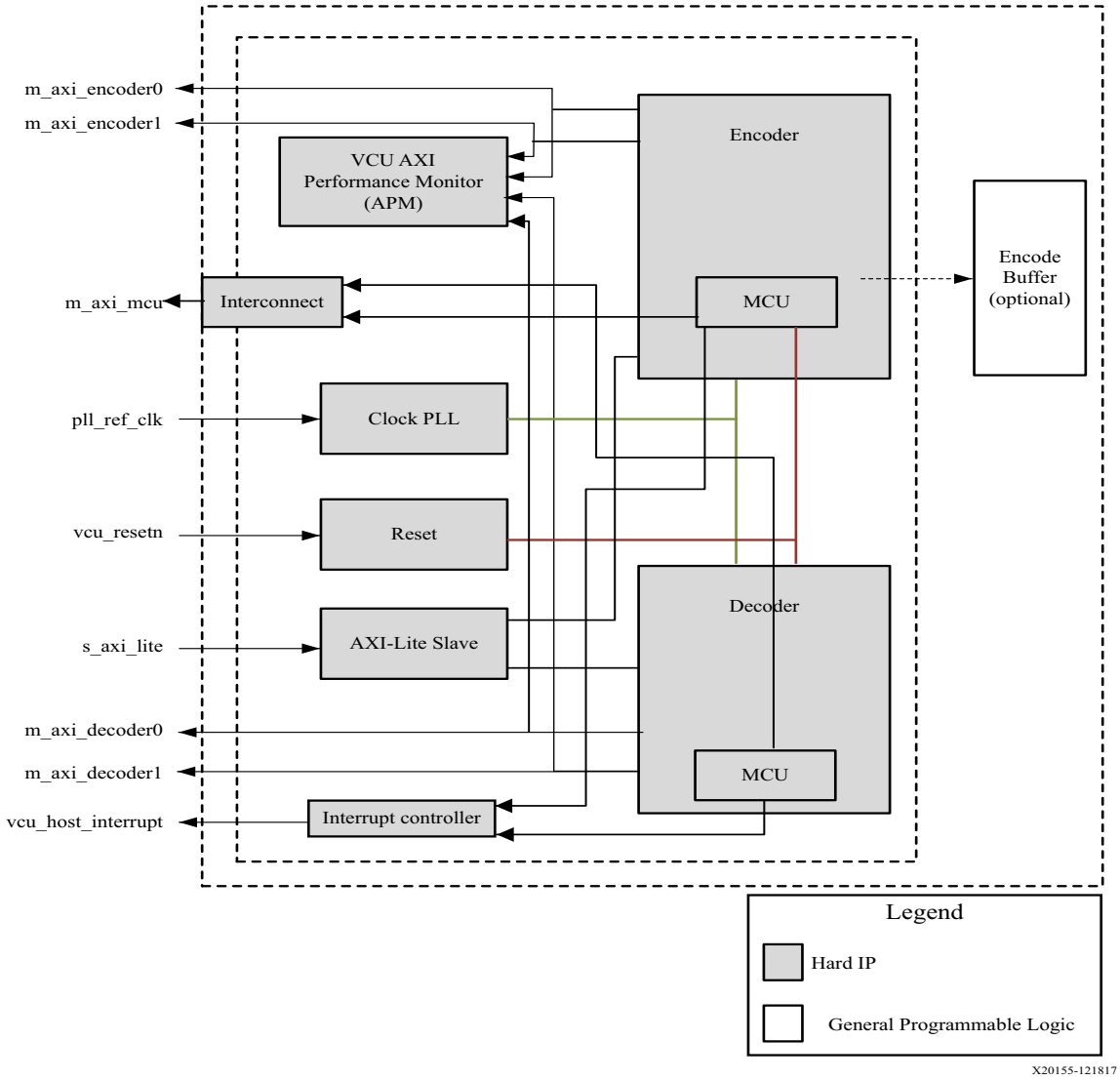

Figure 1-1: **Top-level Block Diagram**

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 60Hz performance. Figure 3-1 shows the top-level interfaces and detailed architecture of the Encoder block. The encoder also 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. Refer to VCU Control Software in Chapter 12 for more information. 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 60Hz 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. See VCU Control Software in Chapter 12 for more information. 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.

MCU Overview

The Encoder and Decoder blocks each implement a 32-bit 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 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. Figure 1-2 shows the use case example where the VCU core works with the PS and the PL DDR external memory.

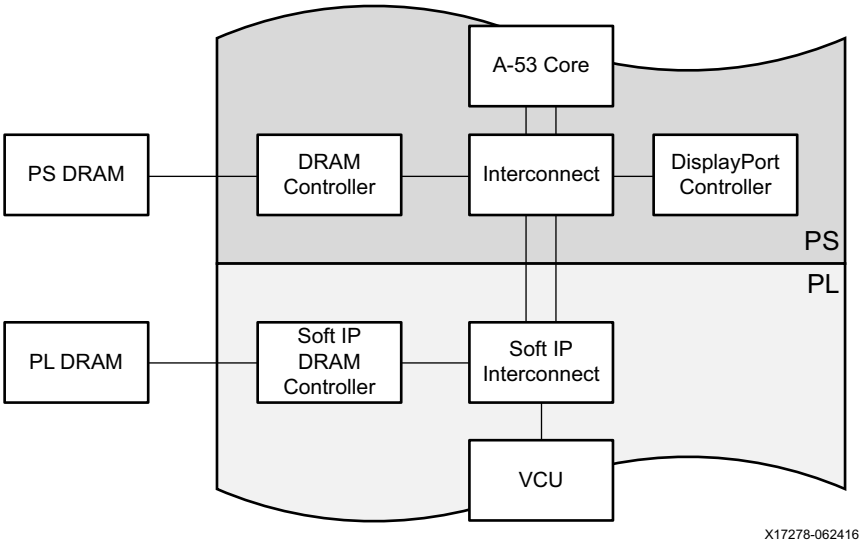

Figure 1-2: **VCU Application**

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 *Zynq UltraScale+ MPSoC Production Errata* [Ref 12] for more information.

Licensing and Ordering Information

License Type

This Xilinx LogiCORE IP module is only available on Zynq® UltraScale+™ MPSoC Family EV Devices and is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License. Information about this and 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.

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.

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 following are typical clock frequencies for the target devices are described in the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics*. 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 60Hz. 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 60Hz 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. See DDR Memory Footprint Requirements in Chapter 3 for more information.

For simultaneous encoder and decoder operation (including transcode use cases), consider utilizing both a Xilinx PS Memory Controller and dedicated a Xilinx VCU Memory Controller. See Chapter 6, Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP for more information.

For details about resource utilization, visit Performance and Resource Utilization.

Core Interfaces and Register Space

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 Figure 2-1.

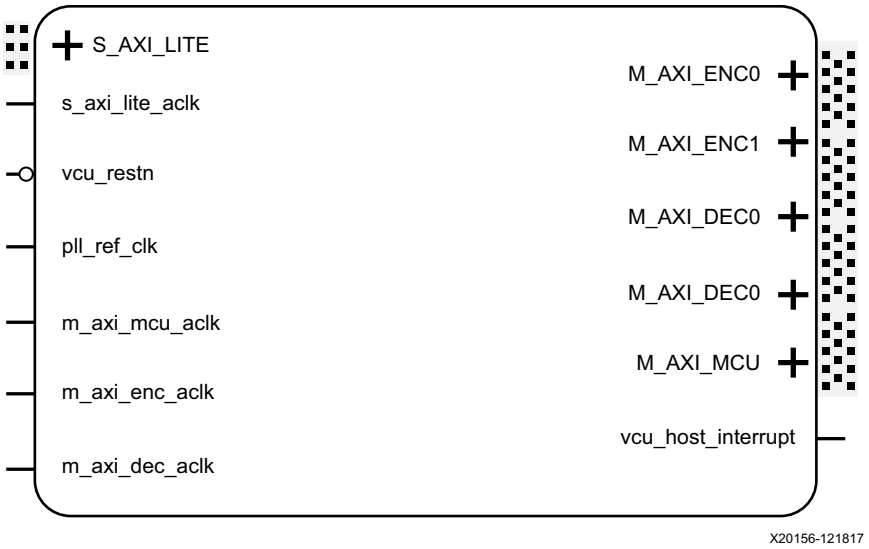

Figure 2-1: **VCU Core Top-Level Signaling Interface**

Table 2-1 summarizes the core interfaces.

Table 2-1: **VCU Interfaces**

Interface Name	Interface Type	Description
M_AXI_ENC0	AXI-4 memory mapped master interface	128-bit memory mapped interface for Encoder block. Refer to Encoder Block Overview in Chapter 1 for more details.
M_AXI_ENC1	AXI-4 memory mapped master interface	128-bit memory mapped interface for Encoder block. Refer to Encoder Block Overview in Chapter 1 for more details.
M_AXI_DEC0	AXI-4 memory mapped master interface	128-bit memory mapped interface for Decoder block. Refer to Decoder Block Overview in Chapter 1 for more details.
M_AXI_DEC1	AXI-4 memory mapped master interface	128-bit memory mapped interface for Decoder block. Refer to Decoder Block Overview in Chapter 1 for more details.
M_AXI_MCU	AXI-4 memory mapped master interface	32-bit memory mapped interface for MCU. Refer to MCU Overview in Chapter 1 for more details.
S_AXI__LITE	AXI4-Lite memory mapped slave interface	AXI4-Lite memory mapped interface for external master access. Refer to MCU Overview in Chapter 1 for more details.

Common Interface Signals

Table 2-2 summarizes the signals which are either shared by, or not part of the dedicated AXI4 interfaces.

Table 2-2: **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

Core Design Signals

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

Table 2-3: **Core Design Signals**

`Usrclk`	These are the clock rates that are set for each of the memory speed bins: 1. For 2133 speed grade, the usrclk is 267 2. For 2400 speed grade, usrclk is 250 3. For 2667 speed grade, usrclk is 250
`sRst_out -`	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.
`InitDone`	This signal specifies DDR4 calibration completion.

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

Register Space

The Zynq UltraScale+ VCU soft IP implements registers in the programmable logic. Table 2-4 summarizes the soft IP registers. These registers are accessible from the PS via the AXI4-Lite bus.

Table 2-4: **Soft IP Registers**

Register	Address Offset	Width	Type	Definition
Video Configuration				
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
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

Table 2-4: **Soft IP Registers *(Cont'd)***

Register	Address Offset	Width	Type	Definition
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
VCU_GASKET_INIT	0x41074	32	Rw	Ensure there is no pending AXI transaction in VCU AXI bus/AXI 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 2-4 represent blended values that are input in GUI.

The VCU Encoder and Decoder blocks are shown in Figure 2-2 and Figure 2-3, respectively.

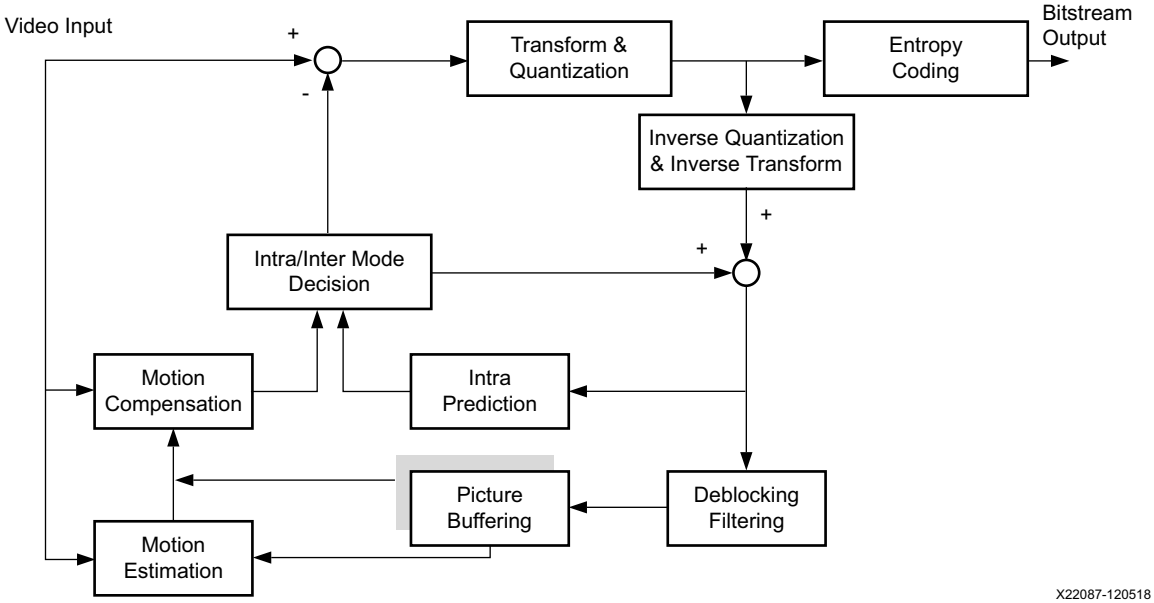

Figure 2-2: **VCU Encoder Block**

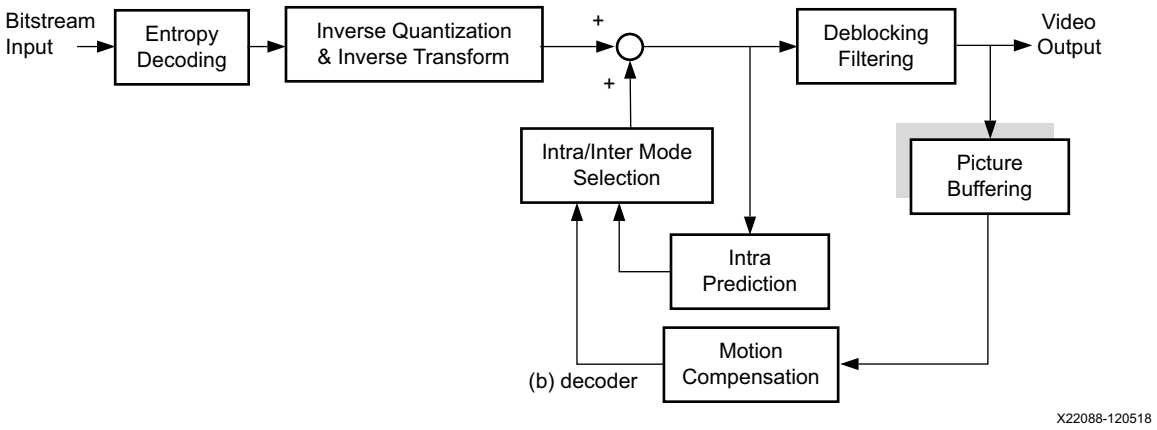

Figure 2-3: **VCU Decoder Block**

Encoder Block

Introduction

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.

Features

Table 3-1 describes the VCU Encoder block features.

Table 3-1: **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) Thirty two 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

Table 3-1: **Encoder Block Features *(Cont'd)***

Video Coding Feature	H.264	H.265
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
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

Table 3-1: **Encoder Block Features *(Cont'd)***

Video Coding Feature	H.264	H.265
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).

Table 3-2 summarizes the maximum bit rate achievable for different profile/level combinations.

Table 3-2: **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)

Table 3-2: **Maximum Bit Rate**

Standard	Level	Profile	Maximum Bit Rate (Mbits/s)
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

Figure 3-1 shows the top-level interfaces and detailed architecture of the Encoder block.

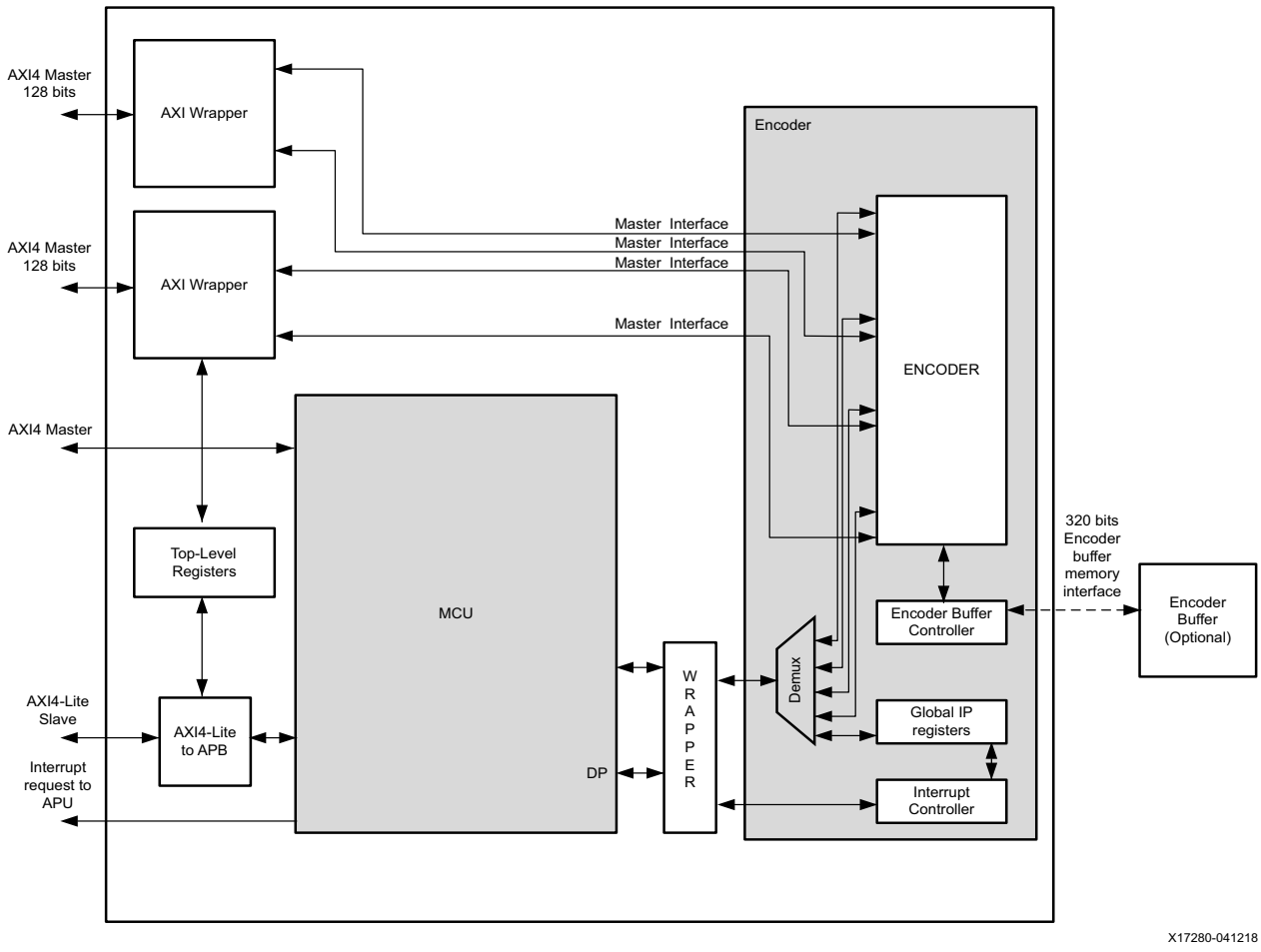

Figure 3-1: **Detailed Architecture of the Encoder Block**

Note: The AXI-4 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 via 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 AXI-4 master interfaces are used to fetch video input data, load and store intermediate data, store compressed data back to memory.
- A 32-bit AXI-4 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`**).

Table 3-3 shows the list of ports/interfaces of the top-level Encoder block.

Table 3-3: **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	AXI-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

Table 3-3: **Encoder Ports *(Cont'd)***

Name	Size (bits)	Dir	Description
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
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

Table 3-3: **Encoder Ports *(Cont'd)***

Name	Size (bits)	Dir	Description
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
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

Table 3-3: **Encoder Ports *(Cont'd)***

Name	Size (bits)	Dir	Description
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
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

Table 3-3: **Encoder Ports *(Cont'd)***

Name	Size (bits)	Dir	Description
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
AXI-Lite Slave Interface (s_axi_lite_aclk domain)			
pl_vcu_awaddr_axi_lite	20	Input	AXI Lite write address bus
pl_vcu_awprot_axi_lite	3	Input	AXI Lite write protection signal
pl_vcu_awvalid_axi_lite	1	Input	AXI Lite write address valid signal
vcu_pl_awready_axi_lite	1	Output	AXI Lite write address ready signal
pl_vcu_wdata_axi_lite	32	Input	AXI Lite write data channel
pl_vcu_wstrb_axi_lite	4	Input	AXI Lite write strobe signal
pl_vcu_wvalid_axi_lite	1	Input	AXI Lite write data valid signal
vcu_pl_wready_axi_lite	1	Output	AXI Lite write ready signal
vcu_pl_bresp_axi_lite	2	Output	AXI Lite write response channel
vcu_pl_bvalid_axi_lite	1	Output	AXI Lite write response valid signal
pl_vcu_bready_axi_lite	1	Input	AXI Lite write response ready signal
pl_vcu_araddr_axi_lite	20	Input	AXI Lite read address channel
pl_vcu_arprot_axi_lite	3	Input	AXI Lite read channel protection signal
pl_vcu_arvalid_axi_lite	1	Input	AXI Lite read address valid signal
vcu_pl_arready_axi_lite	1	Output	AXI Lite read address ready signal
vcu_pl_rdata_axi_lite	32	Output	AXI Lite read data bus
vcu_pl_rresp_axi_lite	2	Output	AXI Lite read response signal
vcu_pl_rvalid_axi_lite	1	Output	AXI Lite read data valid signal
pl_vcu_rready_axi_lite	1	Input	AXI 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

Table 3-3: **Encoder Ports *(Cont'd)***

Name	Size (bits)	Dir	Description
Vcu_pl_enc_al_I2c_addr	17	Output	Address
Pl_vcu_enc_al_I2c_rdata	320	Input	Read data
Vcu_pl_enc_al_I2c_wvalid	1	Output	Write data valid
Vcu_pl_enc_al_I2c_wdata	320	Output	Write data

Clocking

Refer to Chapter 7, Clocking and Resets for more information on clocking.

Reset

Refer to Chapter 7, 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 Chapter 5, 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 Chapter 5, Microcontroller Unit Overview.

The 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* [Ref 15] 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 12-18 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 3-4: **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 Vivado Integrated Design Environment in Chapter 11 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 Table 3-5 and Table 3-6, respectively.

Table 3-5: **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 3-6: **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 Table 3-7.

Table 3-7: **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
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.

Table 3-7: **Encoder Buffer Requirements *(Cont'd)***

Requirement	Description
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 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 logic CORE 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 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.

4kp30 requires lesser space that 4kp60 because they use 2 cores and 4 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 Vivado GUI which computes and provides to the driver, the maximum number of cores used in that use case (4 for 4K60). 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 2 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 utilized 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 4 (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 3-8: **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 can not set the value of num-cores to max in software 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

Table 3-9 shows the worst-case memory footprint for various encoding schemes.

Table 3-9: **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

Table 3-9 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 3-10: **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
1280×720 (MB)	27	26	21	20	20	19	17	16

Notes:

1. AVC requires extra intermediate buffers when it is using multiple-cores, AVC standard does not support Tile processing unlike HEVC, 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 ~100MB delta between HEVC and AVC CMA requirements for 3840x2160.
3. Includes memory for 2 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 3-11: **Buffer-Formula Mapping**

Buffer	Formula
Source Frame	height x width x bit depth (10bit = 4/3, 8bit = 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 (10bit = 10/8, 8bit = 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 (10bit = 10/8, 8bit = 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 multi-stream use case: 4 1080p30 AVC, 4:2:2 chroma format, 10-bit depth.

Table 3-12: Multi-stream Use Case

Buffers	No. of Buffers	Single Stream(1080p30)	Multi-Stream (4 1080p30)
Source Frame	5	27 MB	105MB
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

Table 3-12: Multi-stream Use Case ***(Cont'd)***

Buffers	No. of Buffers	Single Stream(1080p30)	Multi-Stream (4 1080p30)
Bitstream Buffer	2	5 MB	20 MB
Other Buffers	1	1 MB	4 MB
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 4 bank groups, whereas the x16 mode has only 2; and DDR4 allows for simultaneous bank group access. For more information, see Xilinx Answer: 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 Figure 3-2. Chroma pixels are stored in 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.

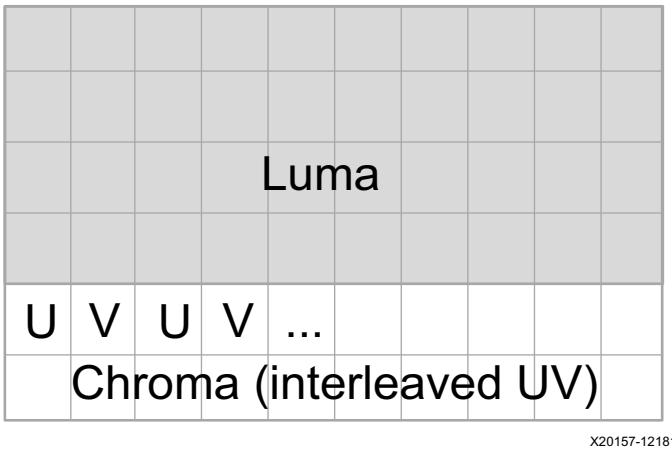

Figure 3-2: **Frame Layout**

Two packing formats are supported in external memory: 8 bits per component or 10 bits per component, shown in Table 3-13 and Table 3-14, 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.

Table 3-13: **Source Frame Buffer Format with 8-bit Component Format**

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)...						

Table 3-14: **Source Frame Buffer Format with 10-bit Component Format**

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 Figure 3-3.

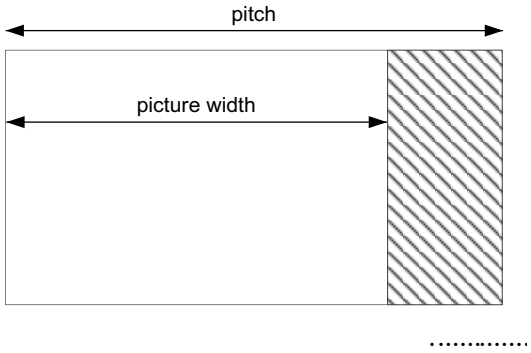

Figure 3-3: **Frame Buffer Pitch**

Notes:

1. Encoder input buffers width and height should be in multiple of 32.
2. 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

Table 3-15 lists the Encoder block registers. For additional information, see *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 14].

Table 3-15: **Encoder Registers**

Register	Address	Width	Type	Reset Value	Description
MCU_RESET	0×A0009000	32	mixed	0x00000000	MCU Subsystem Reset
MCU_RESET_MODE	0xA0009004	32	mixed	0x00000001	MCU Reset Mode
MCU_STA	0xA0009008	32	mixed	0x00000000	MCU Status
MCU_WAKEUP	0xA000900C	32	mixed	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
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
ITC_MCU_IRQ	0xA0009100	32	mixed	0x00000000	MCU Interrupt Trigger
ITC_CPU_IRQ_MSK	0xA0009104	32	rw	0x00000000	CPU Interrupt Mask
ITC_CPU_IRQ_CLR	0xA0009108	32	mixed	0x00000000	CPU Interrupt Clear
ITC_CPU_IRQ_STA	0xA000910C	32	mixed	0x00000000	CPU Interrupt Status
AXI_BW	0xA0009204	32	rw	0x00000000	AXI Bandwidth Measurement Window
AXI_ADDR_OFFSET_IP	0xA0009208	32	rw	0x00000000	Video Data Address Offset
AXI_RBW0	0xA0009210	32	ro	0x00000000	AXI Read Bandwidth Status 0
AXI_RBW1	0xA0009214	32	ro	0x00000000	AXI Read Bandwidth Status 1
AXI_WBW0	0xA0009218	32	ro	0x00000000	AXI Write Bandwidth Status 0
AXI_WBW1	0xA000921C	32	ro	0x00000000	AXI Write Bandwidth Status 1
AXI_RBL0	0xA0009220	32	rw	0x00000000	AXI Read Bandwidth Limiter 0
AXI_RBL1	0xA0009224	32	rw	0x00000000	AXI Read Bandwidth Limiter 1

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 2 for static scenes or video conferencing or reduce to 0 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 may want to 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**.

Decoder Block

Introduction

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.

Features

Table 4-1 describes the Decoder block features.

Table 4-1: **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)

Table 4-1: **VCU Features *(Cont'd)***

Video Coding Feature	H.264	H.265
Performance at 667 MHz • Thirty two 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 8 • 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
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.

Table 4-2 describes the VCU Decoder block maximum supported bit rates.

Table 4-2: **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)

Table 4-2: **VCU Decoder Maximum Supported Bit Rates *(Cont'd)***

Standard	Level	Profile	Maximum Bit Rate (Mbits/s)
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

Figure 4-1 shows the block diagram of the Decoder block.

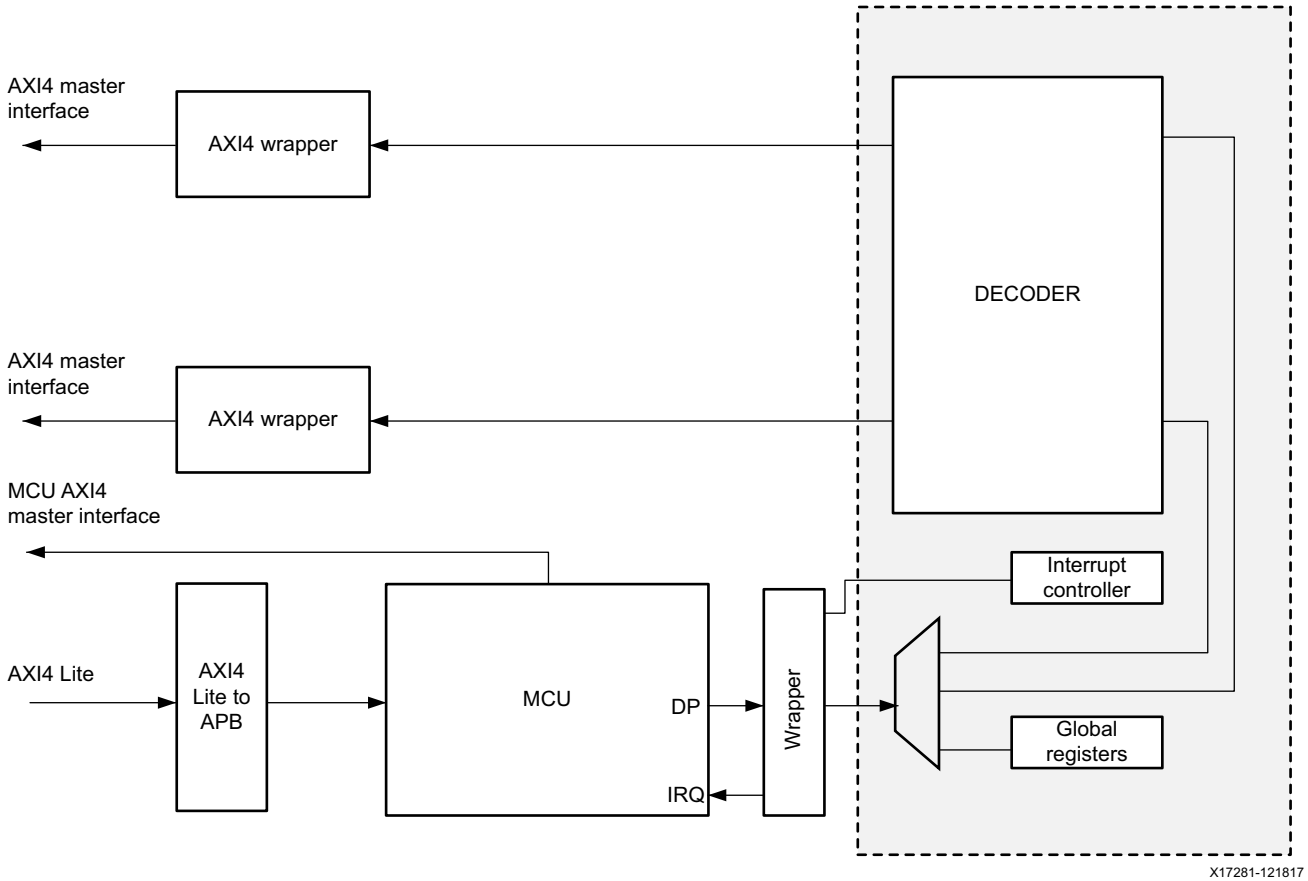

Figure 4-1: **Detailed Architecture of the Decoder Block**

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 AXI-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 AXI-4 master interfaces are used to fetch video input data and store video output data from/to the system memory.

An AXI-4 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 Decoder ports (ports beginning with **`m_axi_dec`**). Table 4-3 shows the Decoder block AXI-4 master interface ports.

Table 4-3: **Decoder Ports**

Name	Width	Dir	Description
vcu_pl_dec_araddr0/1	44	Output	AXI-4 ARADDR signal
vcu_pl_dec_arburst0/1	2	Output	AXI-4 ARBURST signal
vcu_pl_dec_arid0/1	4	Output	AXI-4 ARID signal
vcu_pl_dec_arlen0/1	8	Output	AXI-4 ARLEN signal
pl_vcu_dec_arready0/1	1	Input	AXI-4 ARREADY signal
vcu_pl_dec_arsize0/1	3	Output	AXI-4 ARSIZE signal
vcu_pl_dec_arvalid0/1	1	Output	AXI-4 ARVALID signal
vcu_pl_dec_awaddr0/1	44	Output	AXI-4 AWADDR signal
vcu_pl_dec_awburst0/1	2	Output	AXI-4 AWBURST signal
vcu_pl_dec_awid0/1	4	Output	AXI-4 AWID signal
vcu_pl_dec_awlen0/1	8	Output	AXI-4 AWLEN signal
pl_vcu_dec_awready0/1	1	Input	AXI-4 AWREADY signal
vcu_pl_dec_awsize0/1	3	Output	AXI-4 AWSIZE signal
vcu_pl_dec_awvalid0/1	1	Output	AXI-4 AWVALID signal
pl_vcu_dec_bresp0/1	2	Input	AXI-4 BRESP signal
vcu_pl_dec_bready0/1	1	Output	AXI-4 BREADY signal
pl_vcu_dec_bvalid0/1	1	Input	AXI-4 BVALID signal
pl_vcu_dec_bid0/1	4	Input	AXI-4 BID signal
pl_vcu_dec_rdata0/1	128	Input	AXI-4 RDATA signal
pl_vcu_dec_rid0/1	4	Input	AXI-4 RID signal
pl_vcu_dec_rlast0/1	1	Input	AXI-4 RLAST signal
vcu_pl_dec_rready0/1	1	Output	AXI-4 RREADY signal

Table 4-3: **Decoder Ports *(Cont'd)***

Name	Width	Dir	Description
pl_vcu_dec_rresp0/1	2	Input	AXI-4 RRESP signal
pl_vcu_dec_rvalid0/1	1	Input	AXI-4 RVALID signal
vcu_pl_dec_wdata0/1	128	Output	AXI-4 WDATA signal
vcu_pl_dec_wlast0/1	1	Output	AXI-4 WLAST signal
pl_vcu_dec_wready0/1	1	Input	AXI-4 WREADY signal
vcu_pl_dec_wvalid0/1	1	Output	AXI-4 WVALID signal
vcu_pl_dec_awprot0/1	1	Output	AXI-4 AWPROT signal, controlled from System Level Control Register (SLCR)
vcu_pl_dec_arprot0/1	1	Output	AXI-4 ARPROT signal, controlled from SLCR
vcu_pl_dec_awqos0/1	4	Output	AXI-4 AWQOS signal, controlled from SLCR
vcu_pl_dec_arqos0/1	4	Output	AXI-4 ARQOS signal, controlled from SLCR
vcu_pl_dec_awcache0/1	4	Output	AXI-4 AWCACHE signal, controlled from SLCR
vcu_pl_dec_arcache0/1	4	Output	AXI-4 ARCACHE signal, controlled from SLCR

Clocking

Refer to Chapter 7, Clocking and Resets for more information on clocking.

Reset

Refer to Chapter 7, 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 Table 4-4.

Table 4-4: **Decoder Buffer Requirements**

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

Table 4-5 shows the worst-case memory footprint required for different buffer sizes.

Table 4-5: **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
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

Table 4-6 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 4-6: **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 4 bank groups, whereas the x16 mode has only 2; 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: 8 bits per component or 10 bits per component, shown in Table 4-7 and Table 4-8, 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. Table 4-7 and Table 4-8 show the raster scan format supported by the decoder block for 8-bit and 10-bit color depth.

Table 4-7: **Source Frame Buffer Format with 8-bit Component Format**

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)...						

Table 4-8: **Source Frame Buffer Format with 10-bit Component Format**

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.

Table 4-9: **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

Table 4-10 lists the Decoder block registers. For additional information, see *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 14].

Table 4-10: **Decoder Registers**

Register Name	Address	Width	Type	Reset Value	Description
MCU_RESET	0xA0029000	32	mixed	0x00000000	MCU Subsystem Reset
MCU_RESET_MODE	0xA0029004	32	mixed	0x00000001	MCU Reset Mode
MCU_STA	0xA0029008	32	mixed	0x00000000	MCU Status
MCU_WAKEUP	0xA002900C	32	mixed	0x00000000	MCU Wake-up
MCU_ADDR_OFFSET_IC0	0xA0029010	32	rw	0x00000000	MCU Instruction Cache Address Offset 0
MCU_ADDR_OFFSET_IC1	0xA0029014	32	rw	0x00000000	MCU Instruction Cache Address Offset 1
MCU_ADDR_OFFSET_DC0	0xA0029018	32	rw	0x00000000	MCU Data Cache Address Offset 0
MCU_ADDR_OFFSET_DC1	0xA002901C	32	rw	0x00000000	MCU Data Cache Address Offset 1
ITC_MCU_IRQ	0xA0029100	32	mixed	0x00000000	MCU Interrupt Trigger
ITC_CPU_IRQ_MSK	0xA0029104	32	rw	0x00000000	CPU Interrupt Mask
ITC_CPU_IRQ_CLR	0xA0029108	32	mixed	0x00000000	CPU Interrupt Clear
ITC_CPU_IRQ_STA	0xA002910C	32	mixed	0x00000000	CPU Interrupt Status
AXI_BW	0xA0029204	32	rw	0x00000000	AXI Bandwidth Measurement Window
AXI_ADDR_OFFSET_IP	0xA0029208	32	rw	0x00000000	Video Data Address Offset

Table 4-10: **Decoder Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
AXI_RBW0	0xA0029210	32	ro	0x00000000	AXI Read Bandwidth Status 0
AXI_RBW1	0xA0029214	32	ro	0x00000000	AXI Read Bandwidth Status 1
AXI_WBW0	0xA0029218	32	ro	0x00000000	AXI Write Bandwidth Status 0
AXI_WBW1	0xA002921C	32	ro	0x00000000	AXI Write Bandwidth Status 1
AXI_RBL0	0xA0029220	32	rw	0x00000000	AXI Read Bandwidth Limiter 0
AXI_RBL1	0xA0029224	32	rw	0x00000000	AXI Read Bandwidth Limiter 1

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

Introduction

The Video Codec Unit (VCU) core includes two microcontroller unit (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

Figure 5-1 shows the top-level interfaces and detailed architecture of the MCU.

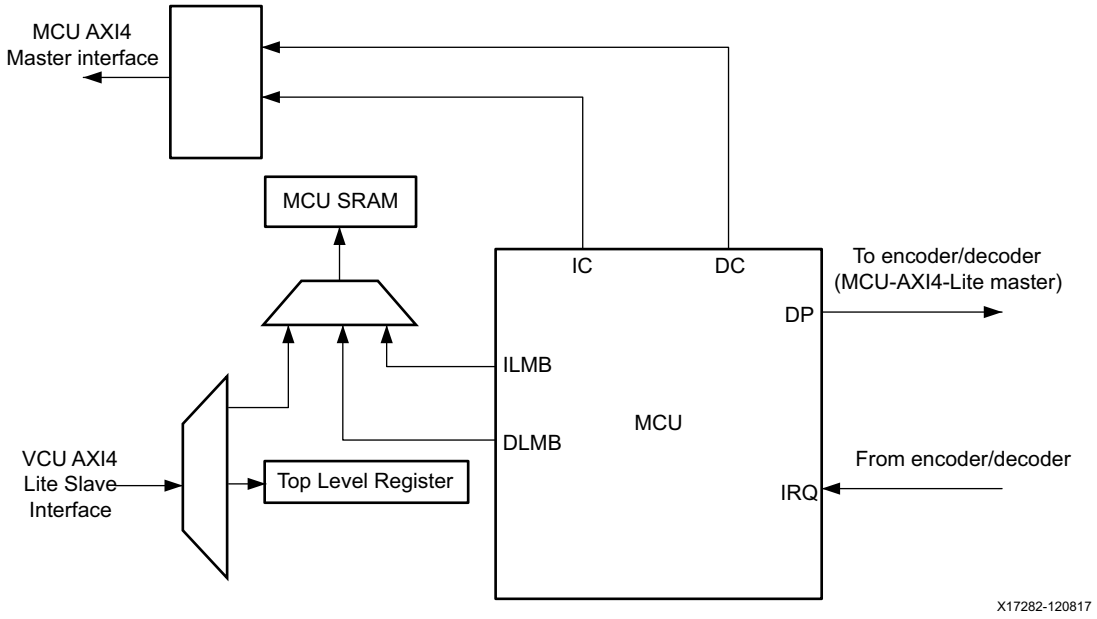

Figure 5-1: **MCU (Top-level)**

The MCU interfaces to peripherals using a 32-bit AXI4-Lite master interface. It has a local memory bus and AXI-4 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

Table 5-1 shows the AXI4 instruction and data cache interface ports of MCU.

Table 5-1: **MCU Ports**

Port	Size (bits)	Dir	Description
vcu_pl_mcu_m_axi_ic_dc_araddr	44	Output	AXI-4 read address
vcu_pl_mcu_m_axi_ic_dc_arburst	2	Output	AXI-4 read burst type
vcu_pl_mcu_m_axi_ic_dc_arcache	4	Output	AXI-4 ARCACHE value
vcu_pl_mcu_m_axi_ic_dc_arid	3	Output	AXI-4 read master ID
vcu_pl_mcu_m_axi_ic_dc_arlen	8	Output	AXI-4 read burst size
vcu_pl_mcu_m_axi_ic_dc_arlock	1	Output	AXI-4 ARLOCK signal
vcu_pl_mcu_m_axi_ic_dc_arprot	3	Output	AXI-4 ARPROT signal
vcu_pl_mcu_m_axi_ic_dc_arqos	4	Output	AXI-4 ARQOS signal
pl_vcu_mcu_m_axi_ic_dc_arready	1	Input	AXI-4 ARREADY signal
vcu_pl_mcu_m_axi_ic_dc_arsize	3	Output	AXI-4 ARSIZE signal
vcu_pl_mcu_m_axi_ic_dc_arvalid	1	Output	AXI-4 ARVALID signal
vcu_pl_mcu_m_axi_ic_dc_awaddr	44	Output	AXI-4 AWADDR signal
vcu_pl_mcu_m_axi_ic_dc_awburst	2	Output	AXI-4 AWBURST signal
vcu_pl_mcu_m_axi_ic_dc_awcache	4	Output	AXI-4 AWCACHE signal
vcu_pl_mcu_m_axi_ic_dc_awid	3	Output	AXI-4 AWID signal
vcu_pl_mcu_m_axi_ic_dc_awlen	8	Output	AXI-4 AWLEN signal
vcu_pl_mcu_m_axi_ic_dc_awlock	1	Output	AXI-4 AWLOCK signal
vcu_pl_mcu_m_axi_ic_dc_awprot	3	Output	AXI-4 AWPROT signal
vcu_pl_mcu_m_axi_ic_dc_awqos	4	Output	AXI-4 AWQOS signal
pl_vcu_mcu_m_axi_ic_dc_awready	1	Input	AXI-4 AWREADY signal
vcu_pl_mcu_m_axi_ic_dc_awsize	3	Output	AXI-4 AWSIZE signal
vcu_pl_mcu_m_axi_ic_dc_awvalid	1	Output	AXI-4 AWVALID signal
pl_vcu_mcu_m_axi_ic_dc_bid	3	Input	AXI-4 BID signal

Table 5-1: **MCU Ports *(Cont'd)***

Port	Size (bits)	Dir	Description
vcu_pl_mcu_m_axi_ic_dc_bready	1	Output	AXI-4 BREADY signal
pl_vcu_mcu_m_axi_ic_dc_bresp	2	Input	AXI-4 BRESP signal
pl_vcu_mcu_m_axi_ic_dc_bvalid	1	Input	AXI-4 BVALID signal
pl_vcu_mcu_m_axi_ic_dc_rdata	32	Input	AXI-4 RDATA signal
pl_vcu_mcu_m_axi_ic_dc_rid	3	Input	AXI-4 RID signal
pl_vcu_mcu_m_axi_ic_dc_rlast	1	Input	AXI-4 RLAST signal
vcu_pl_mcu_m_axi_ic_dc_rready	1	Output	AXI-4 RREADY signal
pl_vcu_mcu_m_axi_ic_dc_rresp	2	Input	AXI-4 RRESP signal
pl_vcu_mcu_m_axi_ic_dc_rvalid	1	Input	AXI-4 RVALID signal
vcu_pl_mcu_m_axi_ic_dc_wdata	32	Output	AXI-4 WDATA signal
vcu_pl_mcu_m_axi_ic_dc_wlast	1	Output	AXI-4 WLAST signal
pl_vcu_mcu_m_axi_ic_dc_wready	1	Input	AXI-4 WREADY signal
vcu_pl_mcu_m_axi_ic_dc_wstrb	4	Output	AXI-4 WSTRB signal
vcu_pl_mcu_m_axi_ic_dc_wvalid	1	Output	AXI-4 WVALID signal

Table 5-2 summarizes the AXI4-Lite slave interface ports of the MCU subsystem.

Table 5-2: **AXI4-Lite Slave Ports**

Port	Width	Direction	Description
pl_vcu_awaddr_axi_lite_apb	20	Input	AXI-4 AWADDR signal
pl_vcu_awprot_axi_lite_apb	3	Input	AXI-4 AWPROT signal
pl_vcu_awvalid_axi_lite_apb	1	Input	AXI-4 AWVALID signal
vcu_pl_awready_axi_lite_apb	1	Output	AXI-4 AWREADY signal
pl_vcu_wdata_axi_lite_apb	32	Input	AXI-4 WDATA signal
pl_vcu_wstrb_axi_lite_apb	4	Input	AXI-4 WSTRB signal
pl_vcu_wvalid_axi_lite_apb	1	Input	AXI-4 WVALID signal
vcu_pl_wready_axi_lite_apb	1	Output	AXI-4 WREADY signal
vcu_pl_bresp_axi_lite_apb	2	Output	AXI-4 BRESP signal
vcu_pl_bvalid_axi_lite_apb	1	Output	AXI-4 BVALID signal
pl_vcu_bready_axi_lite_apb	1	Input	AXI-4 BREADY signal
pl_vcu_araddr_axi_lite_apb	20	Input	AXI-4 ARADDR signal
pl_vcu_arprot_axi_lite_apb	3	Input	AXI-4 ARPROT signal
pl_vcu_arvalid_axi_lite_apb	1	Input	AXI-4 ARVALID signal
vcu_pl_arready_axi_lite_apb	1	Output	AXI-4 ARREADY signal
vcu_pl_rdata_axi_lite_apb	32	Output	AXI-4 RDATA signal

Table 5-2: **AXI4-Lite Slave Ports *(Cont'd)***

Port	Width	Direction	Description
vcu_pl_rresp_axi_lite_apb	2	Output	AXI-4 RRESP signal
vcu_pl_rvalid_axi_lite_apb	1	Output	AXI-4 RVALID signal
pl_vcu_rready_axi_lite_apb	1	Input	AXI-4 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 memory 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 Chapter 12, Application Software Development.

MCU Register Overview

Table 5-3 lists the MCU registers. For additional information, see *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 14].

Table 5-3: **Encoder MCU Registers**

Register	Address	Width	Type	Reset Value	Description
MCU_RESET	0xA0009000	32	mixed	0x00000000	MCU Subsystem Reset
MCU_RESET_MODE	0xA0009004	32	mixed	0x00000001	MCU Reset Mode
MCU_STA	0xA0009008	32	mixed	0x00000000	MCU Status
MCU_WAKEUP	0xA000900C	32	mixed	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
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

XILINX

Chapter 6

Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP

Introduction

The Xilinx® Zynq UltraScale+™ architecture-based FPGAs VCU DDR4 IP 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+ architecture-based MPSoC. It also describes the core architecture and provides details on customizing and interfacing to the core.

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family	Zynq® UltraScale+™ EV Devices
Supported User Interfaces	AXI4-Memory Mapped
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	RTL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows	
Design Entry	Vivado® Design Suite
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

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 (PHY) 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 utilization 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.

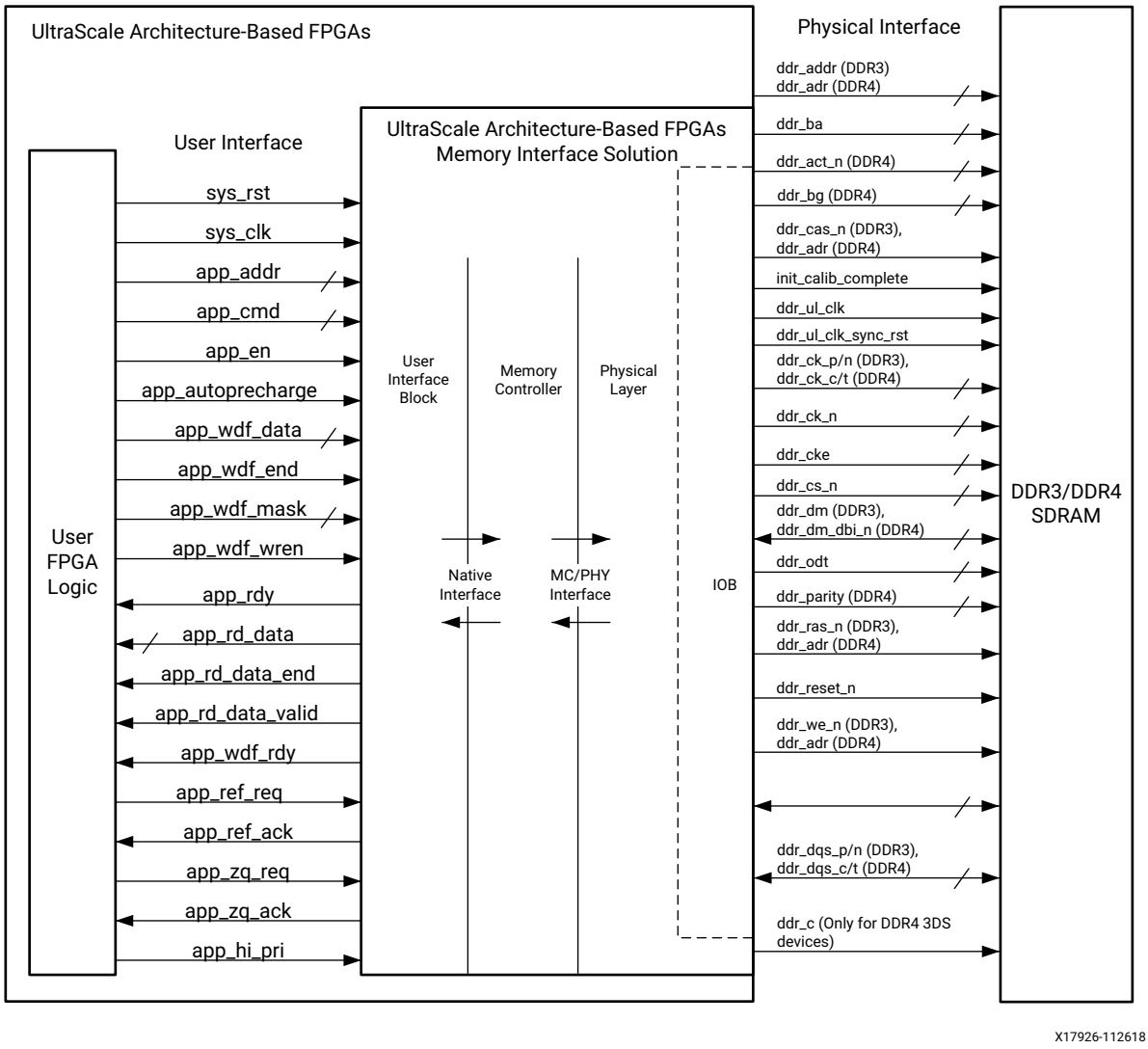

Figure 6-1: **UltraScale Architecture-Based FPGAs DDR4 Memory Interface Solution**

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+™ MPSoC devices.*

- 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 Component Memory for x8 (KVR21SE15S8/4), x16 (MT40A256M16GE-07), and SODIMM X8

    - Support for Zynq-UltraScale+ EV series -1e , -2 , -3e parts and supported frequencies 2133 MT/s, 2400 MT/s, 2667 MT/s.
    - Not supported for -1i * parts
    - Supports AXI
    - Reorders FIFO for better efficiency
- x8, 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
        - UltraScale+ (Zynq)
        - 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 XCZU*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.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado synthesis
- Vivado implementation
- write_bitstream (Tcl command)

Product Specification

Standards

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

For more information on UltraScale™ architecture documents, see References in Appendix B,.

Performance

System Throughput

Table 6-1 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 6-1: **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	NA	NA

Table 6-2 summarizes performance in decoded images/sec for decoding a 4kp60 video stream, 4:2:2, 10-bit, using 4 B-frames.

Table 6-2: **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

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

Note: 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* [Ref 10].

Resource Utilization

For full details about performance and resource utilization, visit Performance and Resource Utilization (DDR4).

Table 6-3: **IP Performance**

Name	CLU LUTS (23040)	CLB Registers (460800)	CARRY8 (28800)	F7 MUXES (11520)	F8 MUXES (57600)	BLOCK RAM TILE (312)	DSPS (1728)	HPIOBDIFFINBUF (192)	BITSLICE_RX_TX (416)	GLOBAL CLOCK BUFFERS (544)	PLL (16)	MMCM (8)
vcu_ddr4_controller	38586	34771	1834	1408	502	123	3	9	106	9	3	1

Port Descriptions

Figure 6-2 shows the block diagram of VCU DDR4 Controller which has five axi ports, **`s_axi_clk`**, **`s_axi_rst`**, **`c0_sys_clk`**, **`sys_rst`**.

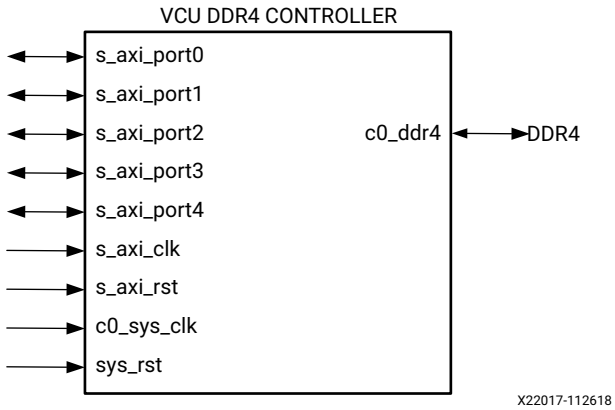

Figure 6-2: **AXI Clock Port / Reset Port Connection**

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 VCIU 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 [1]. However, they do not perform reordering. They support all burst types (narrow transfer, incremental/wrapping/fixed bursts).

Table 6-4 and Table 6-5 represents the AXI port numbers, i.e, 0,1,2,3,4.

Table 6-4: **AXI Write Ports**

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

Table 6-4: **AXI Write Ports *(Cont'd)***

Signal	Direction	Description
pl_barco_slot_awburstX [1:0]	In	Burst type: 00: Fixed 01: Incremental 10: Wrapping 11: Reserved
pl_barco_slot_awvalidX	In	
barco_pl_slot_awreadyX	Out	
Write Data Channel		
pl_barco_slot_widX[15:0]	In	
pl_barco_slot_wdataX [127:0]	In	Write data
pl_barco_slot_wstrbX [15:0]	In	Byte enable
pl_barco_slot_wlastX	In	Not used by the port
pl_barco_slot_wvalidX	In	
barco_pl_slot_wreadyX	Out	
Write response channel		
barco_pl_slot_bidX[15:0]	Out	
barco_pl_slot_brespX[1:0]	Out	Tied to zero(OKAY).
barco_pl_slot_bvalidX	Out	
pl_barco_slot_breadyX	In	

Table 6-5: **AXI Read Ports**

Signal	Direction	Description
Read Address Channel		
pl_barco_slot_aridX[15:0]	In	
pl_barco_slot_araddrX [31:0]	In	Byte address
pl_barco_slot_arlenX[7:0]	In	Length of burst (number of transfer minus 1)
pl_barco_slot_arsizeX[2:0]	In	Transfer width: 000: 8-bit 001: 16-bit 010: 32-bit 011: 64-bit ...
pl_barco_slot_arburstX [1:0]	In	Burst type: 00: Fixed 01: Incremental 10: Wrapping 11: Reserved
pl_barco_slot_arvalidX	In	

Table 6-5: **AXI Read Ports *(Cont'd)***

Signal	Direction	Description
barco_pl_slot_arreadyX	Out	
Read Data Channel		
barco_pl_slot_ridX[15:0]	Out	
barco_pl_slot_rdataX[127:0]	Out	Read data
barco_pl_slot_rrespX[1:0]	Out	Tied ti zero (OKAY)
barco_pl_slot_rlastX	Out	
barco_pl_slot_rvalidX	Out	
pl_barco_slot_rreadyX	In	

Endianness

BA317 ports use normally little-endian convention. The Table 6-6 shows accesses to same data in memory from different kind of ports.

Table 6-6: **BA317 Ports**

Port	Endianness	Address	Data
64-bit buffered port	Little-endian	0x0	0x0807060504030201
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	0x100F0E0D0C0B0A09080706050403 0201
32-bit AXI port	Little-endian	0x0	0x04030201
		0x1	0x08070605

Physical Interface

Table 6-7 lists all available physical interfaces with the supported SDRAM and FPGA devices. Some additional FPGA's might be supported, because they are compatible with the listed FPGA's.

Table 6-7: **Physical Interface**

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

Vendor guidelines apply.

UltraScale/UltraScale+ Xilinx Phy (phyXilinxUltrascale)

Refer to *LogiCORE IP UltraScale Architecture-Based FPGAs Memory IP Product Guide* (PG150) [Ref 10] for details.

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 Figure 6-3.

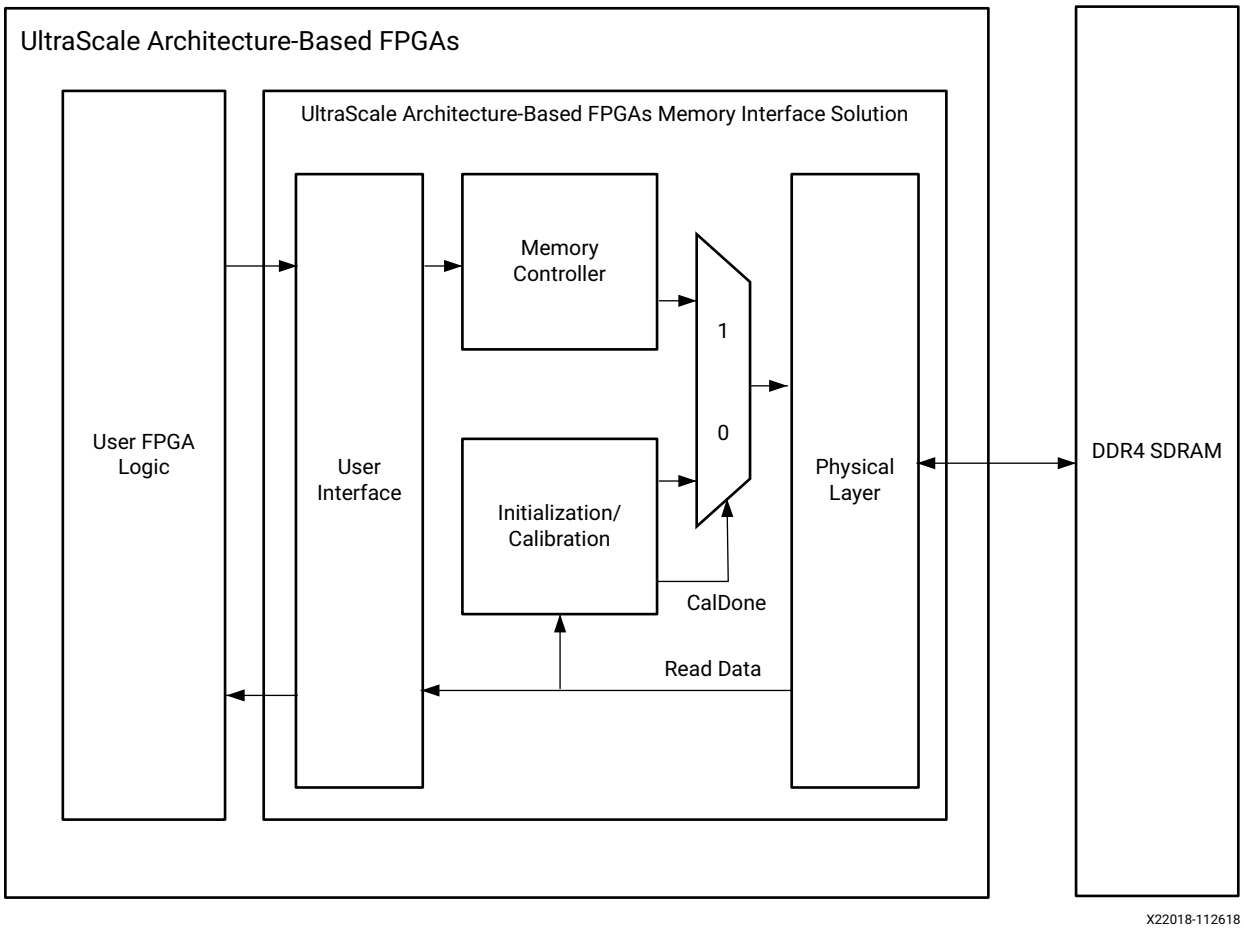

Figure 6-3: **UltraScale Architecture-Based FPGAs Memory Interface Solution Core Architecture**

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

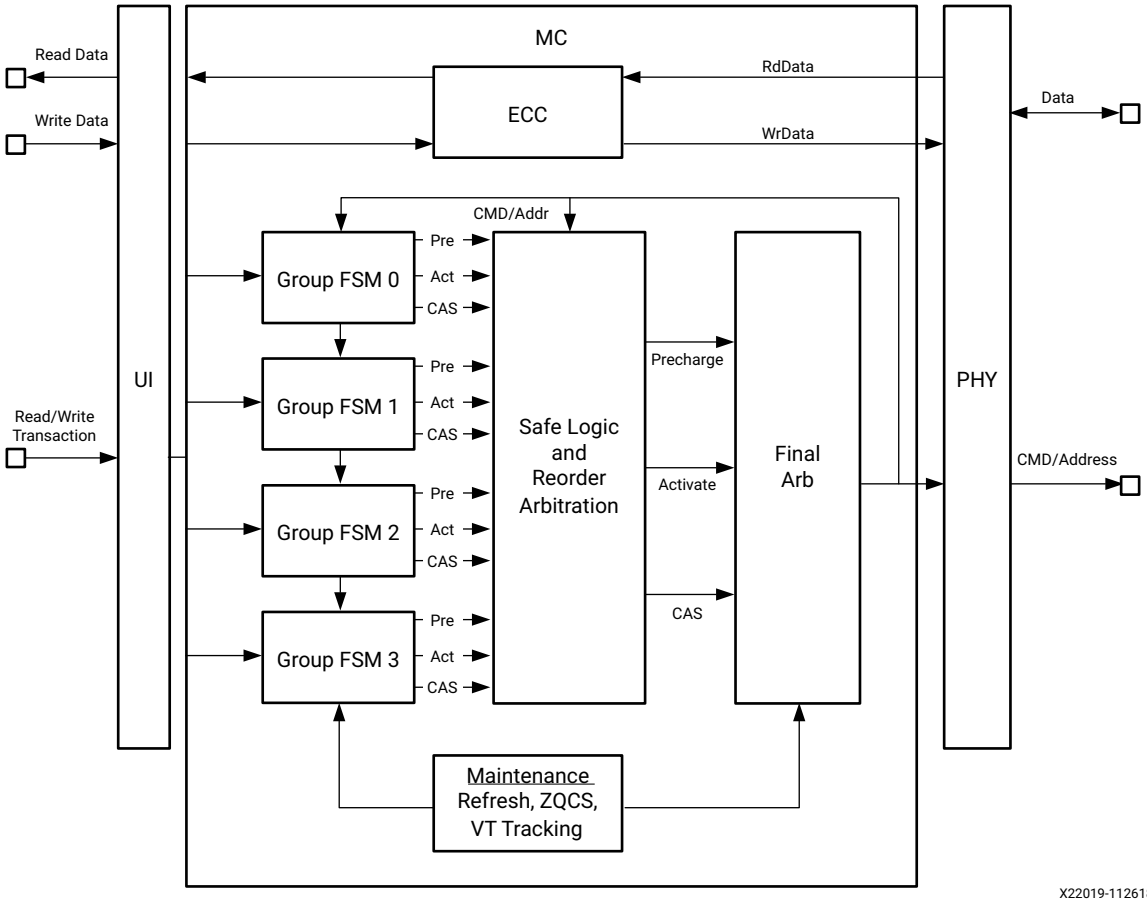

Figure 6-4: **Memory Controller Block Diagram**

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 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 *LogiCORE IP UltraScale Architecture-Based FPGAs Memory IP Product Guide* (PG150) [Ref 10] for more information.

Designing with the Core

Clocking

In the 2018.3 there is a hard requirement for reference clock:

- 125 MHz (x16)
- 300 MHz (x8)

In addition, for the speedgrade support in 2018.3:

- -1e speedgrade, the supported frequency is 2400 MT/s
- -2e speedgrade, the supported frequency can be up to 2667 MT/s

The memory interface requires one MMCM, one TXPLL per I/O bank used by the memory interface, and two **BUFG**s. 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) [Ref 11].

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)
- Must use integer multiply and output divide values. Equations for the cases are as follows:
  - For x8 – 2400: 300000000(input clk)*5/6 = 250 MHz
  - For x8 – 2133: 300000000(input clk)*16/3*6 = 266 MHz

- For x16 – 2400, 2667: 125000000(input clk)*8/4 = 250 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-High 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 in Chapter 7.

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

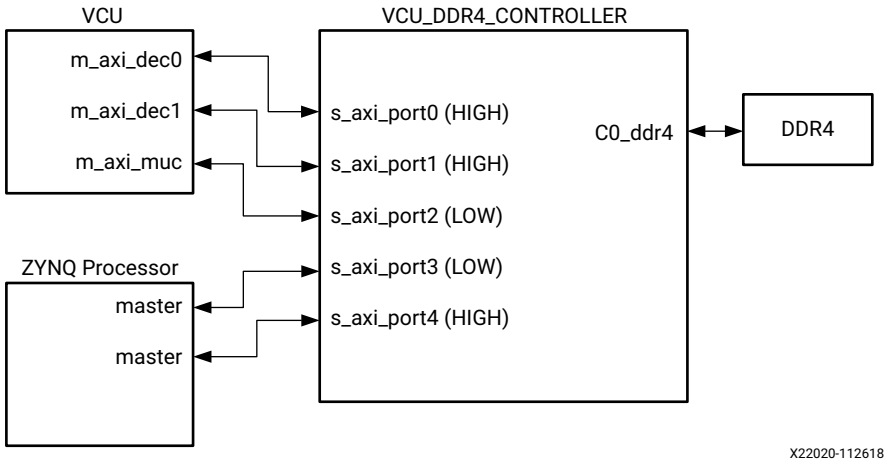

Figure 6-5: **Connecting to the VCU IP**

Table 6-8: **AXI Ports Map and Priority**

AXI PORTS	MAPPING	PORT PRIORITY (Fixed in v1.0)
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 Rational

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

The ports consuming the highest bandwidth via Decoder ports and Zynq MP 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 *LogiCORE IP UltraScale Architecture-Based FPGAs Memory IP Product Guide* (PG150) [Ref 10] 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.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core.

Customizing the VCU DDR4 Controller

You can customize the GUI based on the following options:

- DRAM bus width selection- can be x8 (zcu104 evaluation board) or x16 (zcu106 evaluation board)
- DRAM speed bin selection- 2133 and 2400 for x8 selection. 2400 and 2667 for x16 selection

Other parameters are fixed in 2018.3 wizard. In 2019.1 release, Xilinx will add more customization options for the logiCORE.

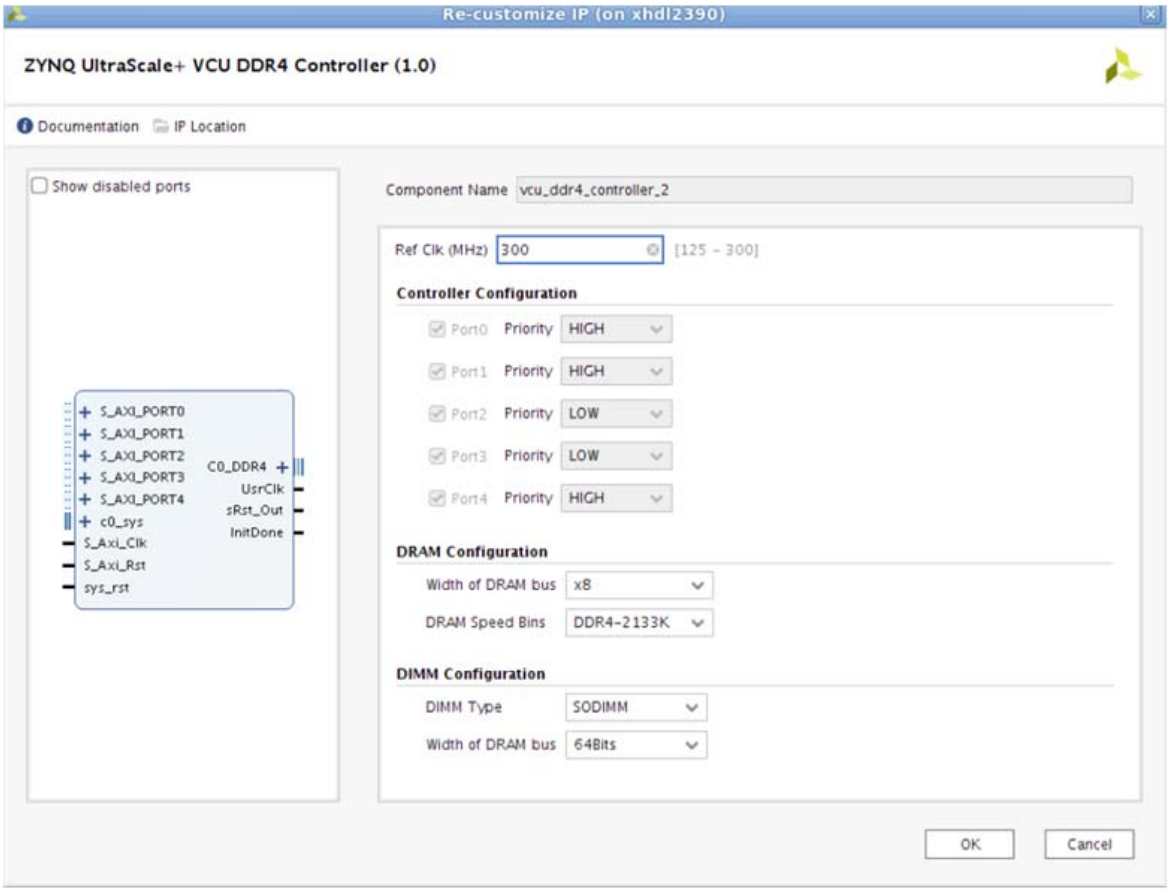

Figure 6-6: **X8 GUI Configuration**

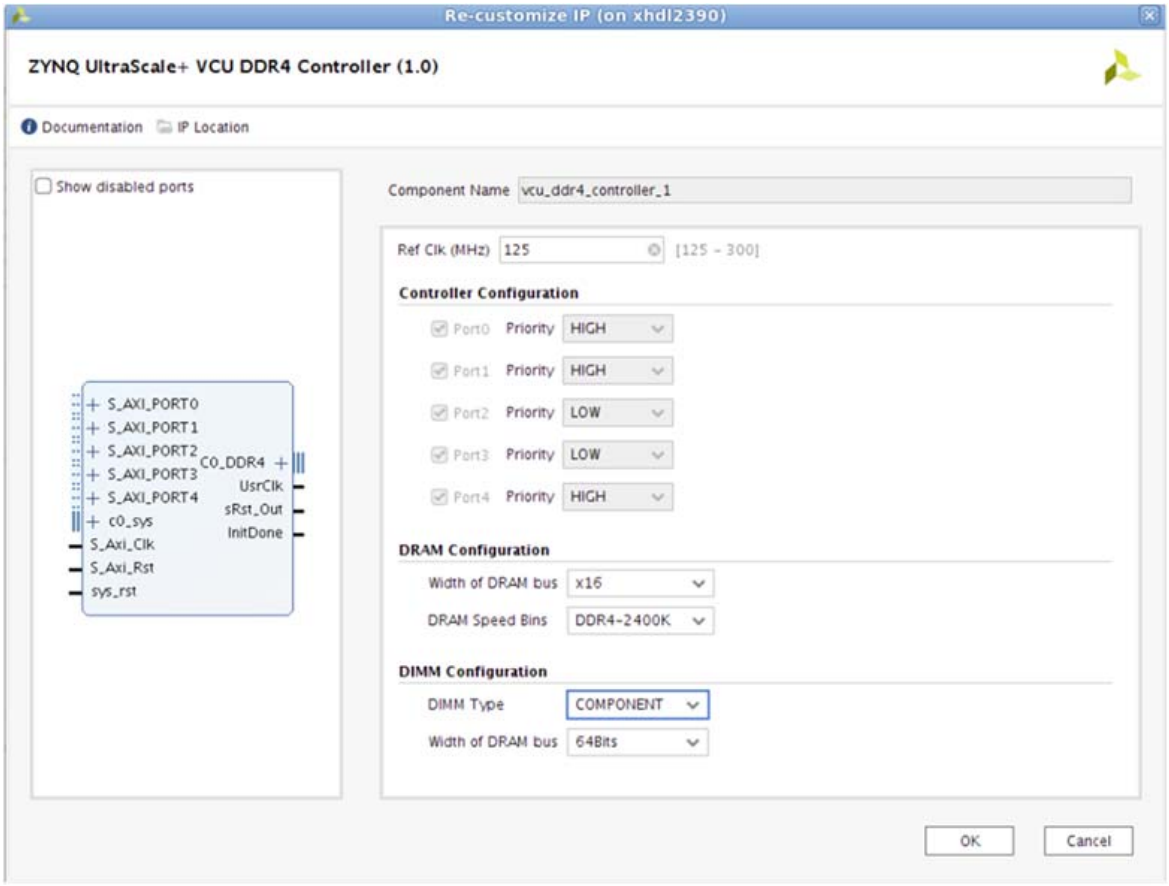

Figure 6-7: **X16 GUI Configuration**

The port priority is fixed in 2018.3 release. Five ports are enabled in the IP. The high priority ports need to be connected to decoder/display interface. The low priority ports need to be connected to PS, Micro Controller Unit (MCU) interface.

Table 6-9: **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 Bits	1	2400 MT/s –- 17-17-17 2667 MT/s – 19-19-19
X8 (KVR21SE15S8/4)	DDR4 - 2400, 2133	COMPONENT, SODIMM	300 Mhz	64 Bits	1	2400 MT/s – 17-17-17 2133 MT/s – 15-15-15

Connecting with VCU LogiCORE IP

Figure 6-4 shows how the IP is connected to VCU LogiCORE IP.

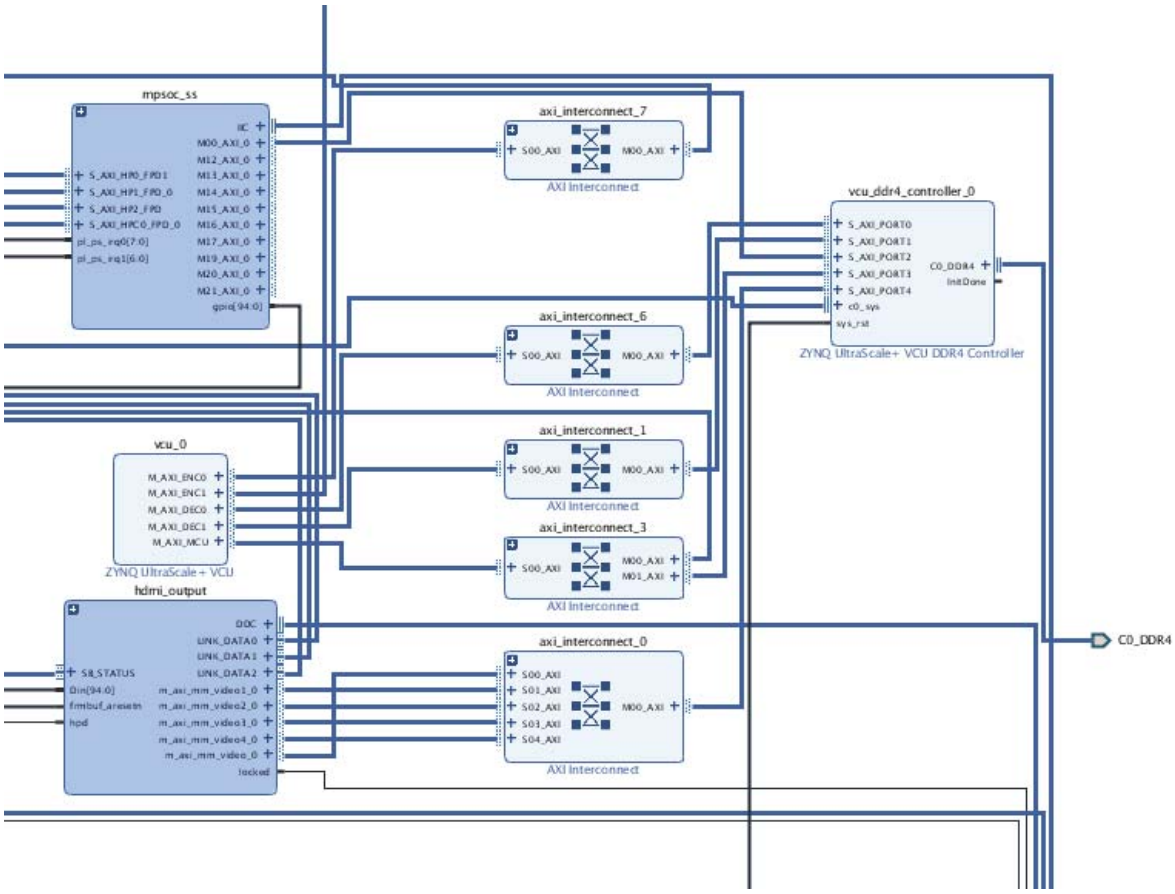

Figure 6-8: **IP Connection**

Table 6-10 shows the recommended port connection.

Table 6-10: **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) [Ref 3].

Constraining the Core

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

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) [Ref 3]. 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

For 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
```

For 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 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

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5].

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 2].

Clocking and Resets

Introduction

The Video Codec Unit (VCU) core supports one clocking topology:

- Internal PLL: An internal VCU phase locked loop (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 PL is in powerup/configuration mode, the VCU core is held in reset.
- After 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 VCU's reset state.
- 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. Table 7-1 describes the clock domains in VCU core.

Table 7-1: **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.
AXI-Lite slave port clock	167	s_axi_lite_aclk, AXI-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.

See The Encoder Buffer in Chapter 3 for more information.

Figure 7-1 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 AXI-Lite clock from the PL.
- **m_axi_mcu_aclk** is the MCU AXI master clock from the PL.

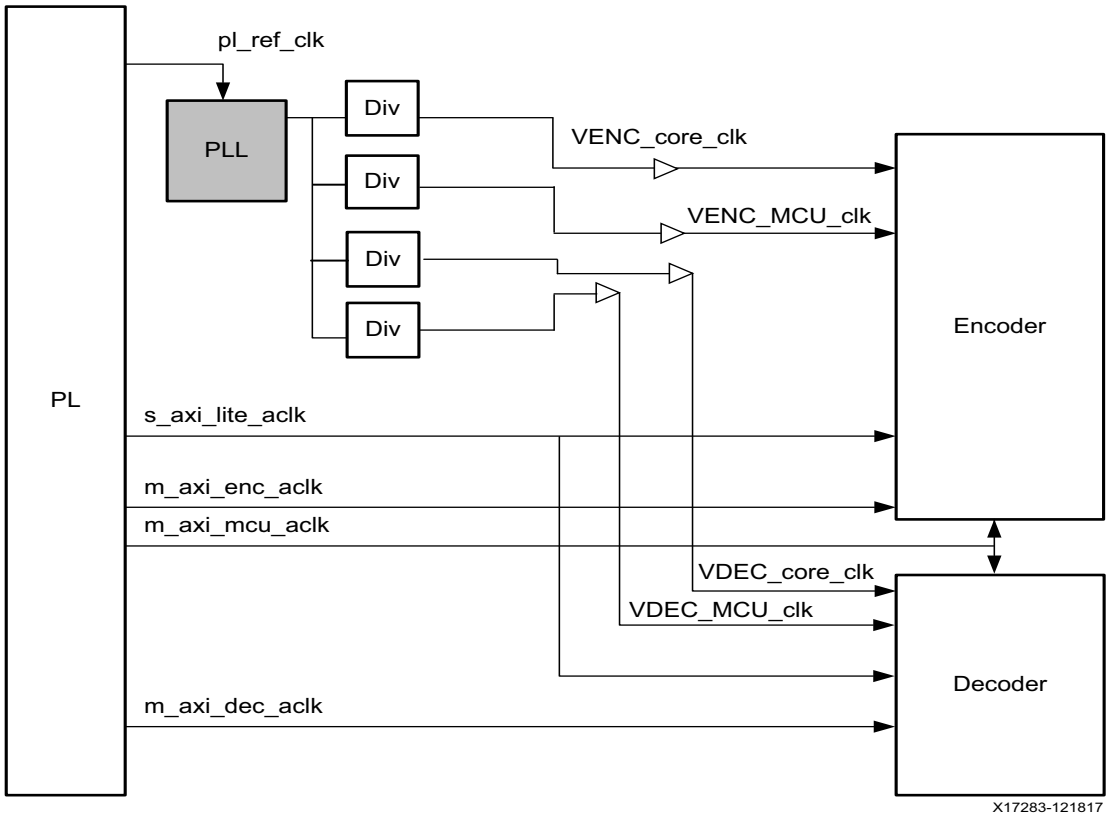

Figure 7-1: **Clock Generation Options**

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:
  - $\texttt{s\_axi\_lite\_aclk} \leq 2 \times \texttt{m\_axi\_enc\_aclk}$
  - $\texttt{s\_axi\_lite\_aclk} \leq 2 \times \texttt{m\_axi\_dec\_aclk}$

Refer to Chapter 5, 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/Decoder block clocks. Typically, the PLL has an external source such as an Si570 XO programmable clock generator connected directly via an IBUFDS to the PLL reference clock. Alternatively, the IBUFDS may drive an MMCM to enable other modules to share 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 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 VCO's output clock 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 that 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 *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* [Ref 16] for more information on the operating range of f_{vco}.

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

IMPORTANT: *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 VCU during PL power up and the initialization sequence for VCU are as follows:

- The PL is not yet configured. In this condition 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 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 *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 14].

Note: The VCU core clocks are available while 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 Table 7-2. 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 Table 7-2.

Table 7-2: **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

Table 7-2: **PLL Programming for Integer Feedback Divider Values *(Cont'd)***

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
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
59	3	2	3	63	625
60	3	2	3	63	625
61	3	2	3	63	600
62	3	2	3	63	600

Table 7-2: **PLL Programming for Integer Feedback Divider Values *(Cont'd)***

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
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 7-2: **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

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 at least for 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.

Note: The reset does not need to be asserted between changes to the VCU configuration during run-time via the VCU Control Software.

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/AXI 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.

See Table 2-2 for more information.

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.

Clocking and Reset Registers

Table 7-3 lists the clocking and reset registers.

Table 7-3: **Clocking and Reset Registers**

Register	Address	Width	Type	Reset Value	Description
CRL_WPROT	0xA0040020	1	rw	0x00000000	CRL SLCR Write protection register
VCU_PLL_CTRL	0xA0040024	32	mixed	0x0000510F	PLL Basic Control
VCU_PLL_CFG	0xA0040028	32	mixed	0x00000000	Helper data
PLL_STATUS	0xA0040060	32	mixed	0x00000008	Status of the PLLs

Latency in the VCU Pipeline

The Video Codec Unit (VCU) is designed to support video streams at resolutions up to 3,840×2,160 pixels at 60 frames per second (4K UHD at 60Hz) 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 Figure 8-1, 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.

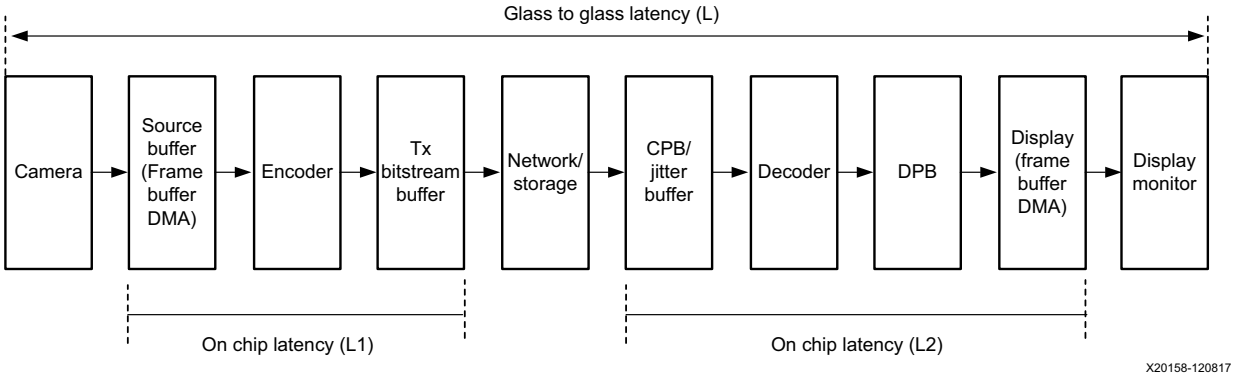

Figure 8-1: **Glass to Glass Latency**

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* [Ref 18].

VCU Latency Modes

The VCU supports three rate control mode: Normal, No reordering, and Low. Depending on the frame structure, encoding standard, levels and profiles and target-bitrate, the latency across the pipeline varies.

- **Normal-Latency**: VCU encoder and decoder works at frame level. All possible frame types (I, P and B) are supported, no restriction on GOP structure. The end-to-end latency depends on the profile/level, GOP-Structure and number of internal buffers used for processing. This is standard latency and can be used with any of control rate mode.
- **No Reordering (Reduced-Latency)**: VCU encoder still works at frame level. Hardware rate control is used to reduce the bitrate variations. I only, IPPP and low-delay-P are supported, no output reordering, thus reducing latency at Decoder side. VCU continues to operate at frame level.
- **Low-Latency**: The frame is divided into multiple slices, VCU Encoder output and decoder input are processed in slice mode. VCU Encoder input and Decoder output still works in frame mode. VCU Encoder generates 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.

Table 8-1 shows the latency for VCU pipeline stages.

Table 8-1: **VCU Latency**

Use Case	Capture	Encode	Decode	Display
Normal Latency	16.6 ms	16.6 ms	83.33 ms	16.6 ms
Reduced Latency	16.6 ms	16.6 ms	50 ms	16.6 ms
Low Latency	16.6 ms	3 ms	22.6 ms	16.6 ms

Note: The values in the above table are estimated numbers.

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: At Control Software level, it can be specified via command line arguments. By default, ctrlsw_encoder and ctrlsw_decoder application works in Normal latency mode, so no arguments required to be passed.

  ```
  ctrlsw_encoder –cfg encode_simple.cfg

  ctrlsw_decoder -avc -in input-avc-file.h264 -out ouput.yuv
  ```

  There is another parameter (--framelat) which can also be used for normal latency mode.

  ```
  ctrlsw_decoder -avc -in input-avc-file.h264 -out ouput.yuv --framelat
  ```

- **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 Control Software level, it can be specified via 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.

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 Control Software level, it can be specified via command line arguments.

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

VCU Encoder Latency

Figure 8-2 show the Encoder latency.

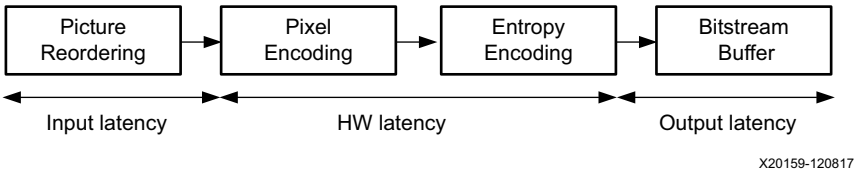

Figure 8-2: **Encoder Latency**

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

VCU Decoder Latency

Figure 8-3 show the Decoder latency.

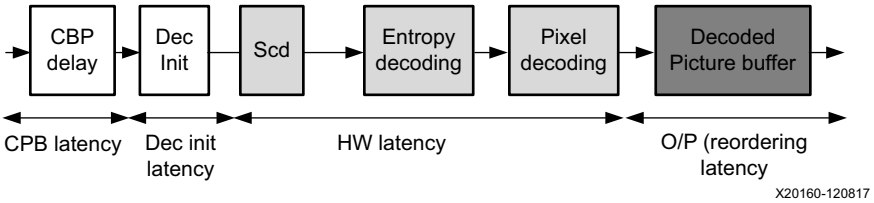

Figure 8-3: **Decoder Latency**

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.

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 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

Figure 9-1 shows the VAPM.

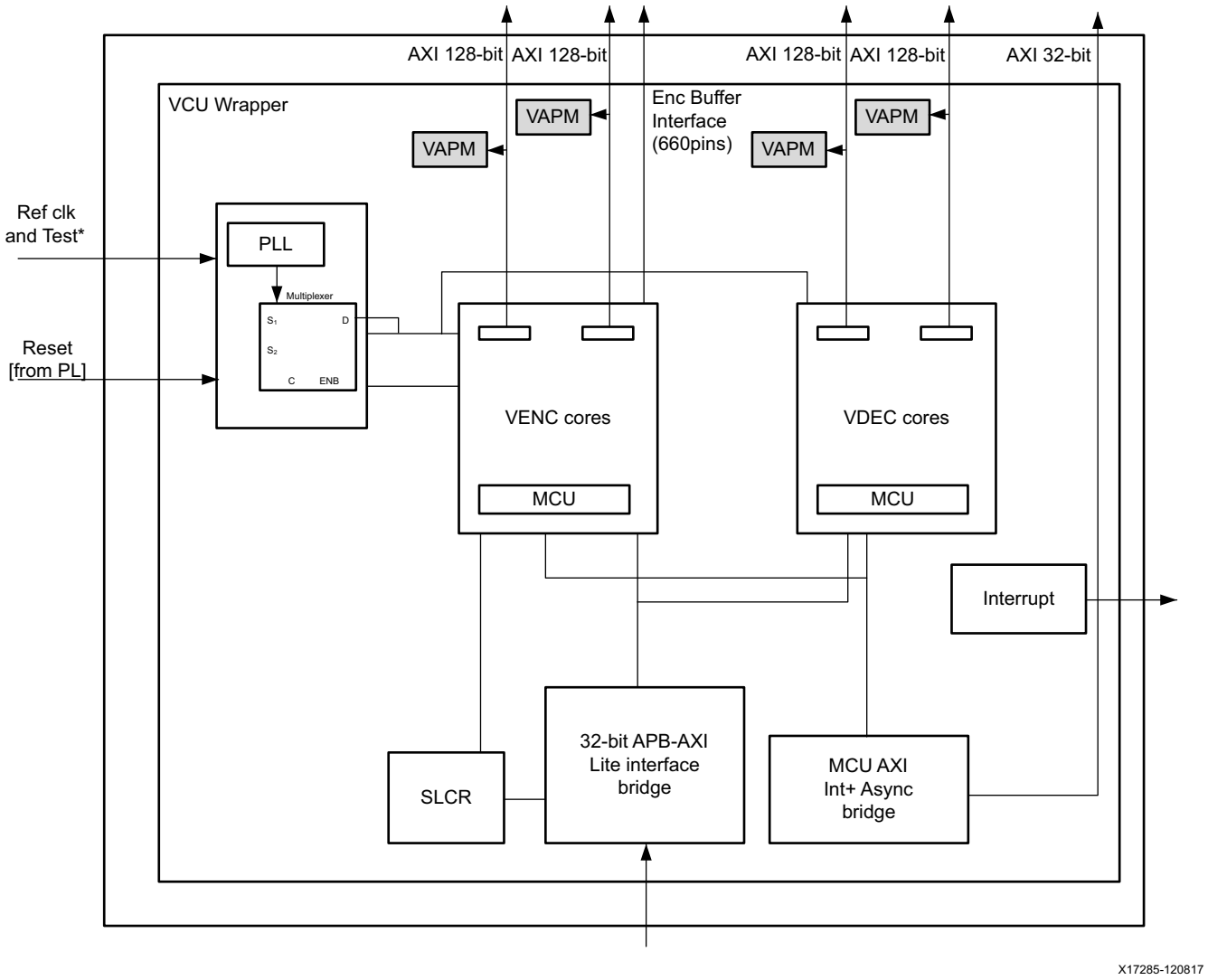

Figure 9-1: **VCU AXI Performance Monitor (VAPM)**

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.APM*n*_TRG[start_stop] (*n*=0,1,2,3)** bit. A measurement is triggered when start bit is set from 0 to 1 in this register and measurement 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 fixed length measurement window. When the counter reaches 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 **APM*n*_TIMER** register and a capture pulse is generated to store the measured values in **VCU_SLCR** result registers.

The VAPM is capable of doing the following measurements:

- AXI Read and Write Transaction Measurement
- AXI Read and Write Byte Count Measurement
- AXI Transaction Latency Measurement

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 start/stop mode or 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 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 single ID or all IDs for latency calculation. For additional information, see *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 14].

Table 9-1: **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
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

Table 9-1: **APM Registers *(Cont'd)***

Register	Address	Width	Type	Reset Value	Description
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
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

Table 9-1: **APM Registers *(Cont'd)***

Register	Address	Width	Type	Reset Value	Description
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
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

Table 9-1: **APM Registers *(Cont'd)***

Register	Address	Width	Type	Reset Value	Description
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

Designing with the Core

This chapter 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 AXI-4 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).

For high bandwidth VCU applications requiring simultaneous encoder and decoder operation, the encoder should be connected to the PS DDR and decoder should be connected to the PL DDR. Refer to Chapter 6, Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP 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.

Design Flow Steps

This chapter 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) [Ref 1]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 2]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5]

Vivado Integrated Design Environment

The VCU can only be added to a Vivado IP integrator block design in the Vivado Design Suite. For more detailed information on customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 1]. 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. In the Flow Navigator, click on **Create Block Diagram** or **Open Block Design** under the IP Integrator heading.
2. Right click in the diagram and select **Add IP**.

   A searchable IP catalog opens. You can also add IP by clicking on the Add IP button on the left side of the IP Integrator Block Design canvas.

3. Click on the IP name and press **Enter** or double-click on the IP name.
4. Double-click the selected IP block or select the **Customize Block** command from the right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 2] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4].

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

Note: The LogiCORE™ IP is available through the Vivado IP Integrator (IPI) flow only. It is not available as a standalone IP.

The Basic configuration tab, shown in Figure 11-1, allows for the selection of video parameters used to calculate the Encoder buffer size and total dynamic power used by Encoder or Decoder blocks.

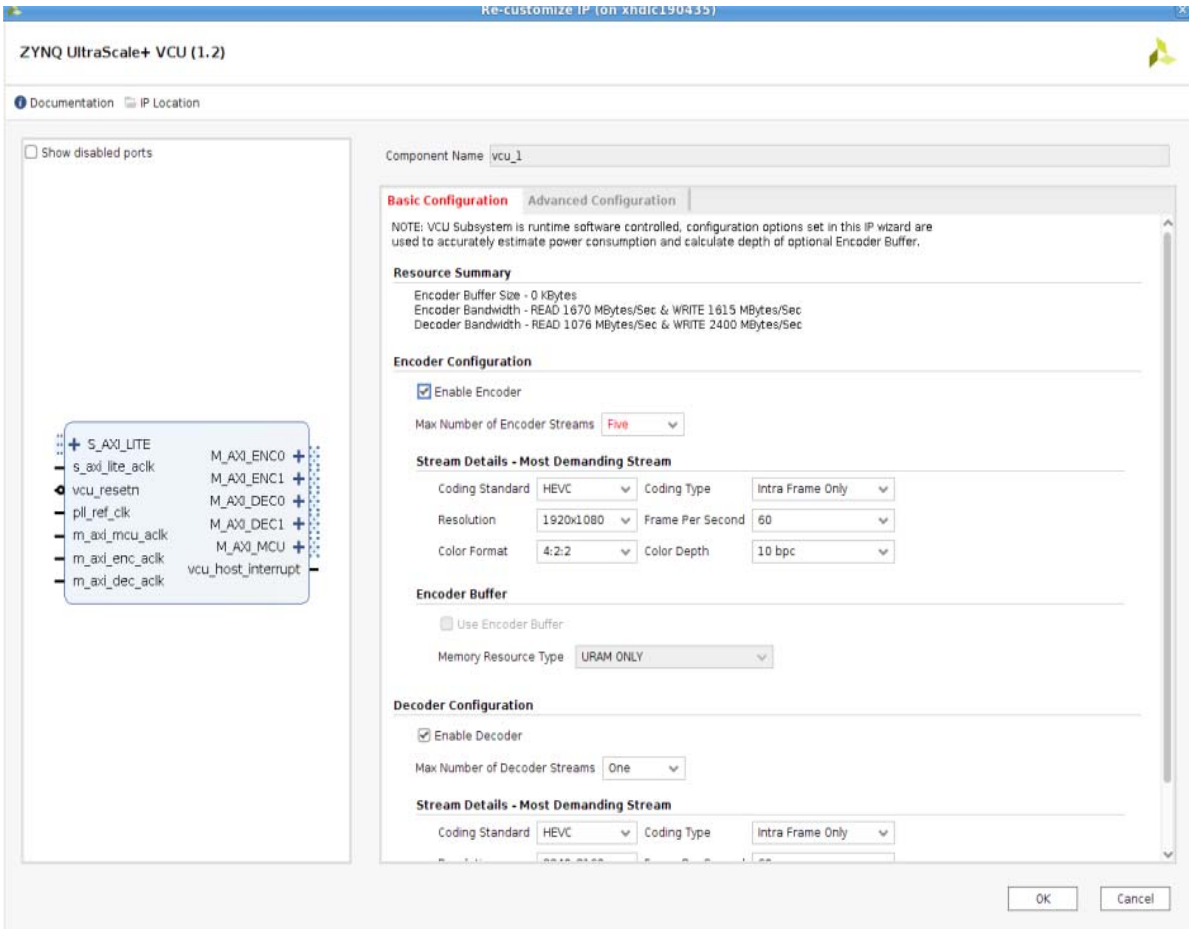

Figure 11-1: **Basic Configuration Tab**

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 Encoder Buffer size. See VCU Control Software Sample Applications, page 164 and Xilinx VCU Control Software API, page 179 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
  - **Max Number of Encoder Streams**: Select one to eight 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:
    - 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 15fps 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 video quality.
  - **Memory Resource Type**: Select of the following memory type options:

- UltraRAM ONLY
- BlockRAM ONLY
- COMBINATION - UltraRAM and blockRAM

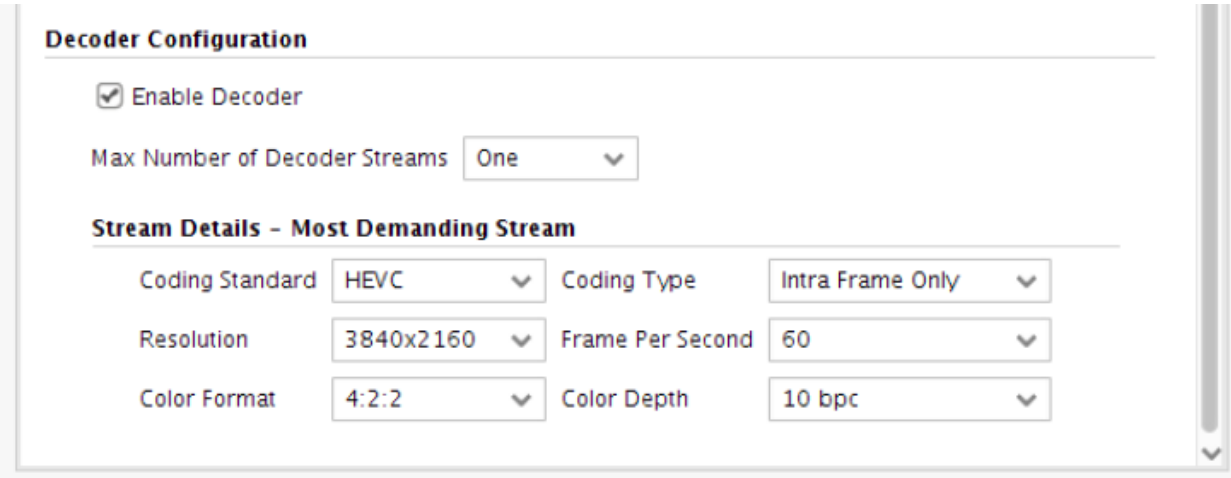

Figure 11-2: **Decoder Basic Configuration Tab**

- **Decoder Configuration**: The decoder has the following parameters:
  - **Enable Decoder**: Enables the decoder and associated parameters.
  - **Max Number of Decoder Streams**: Select one to eight 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:
    - 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 15fps 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.

The Advanced Configuration tab, shown in Figure 11-3, allows you to override the Encoder Buffer memory depth, compression features, and the Encoder core clock.

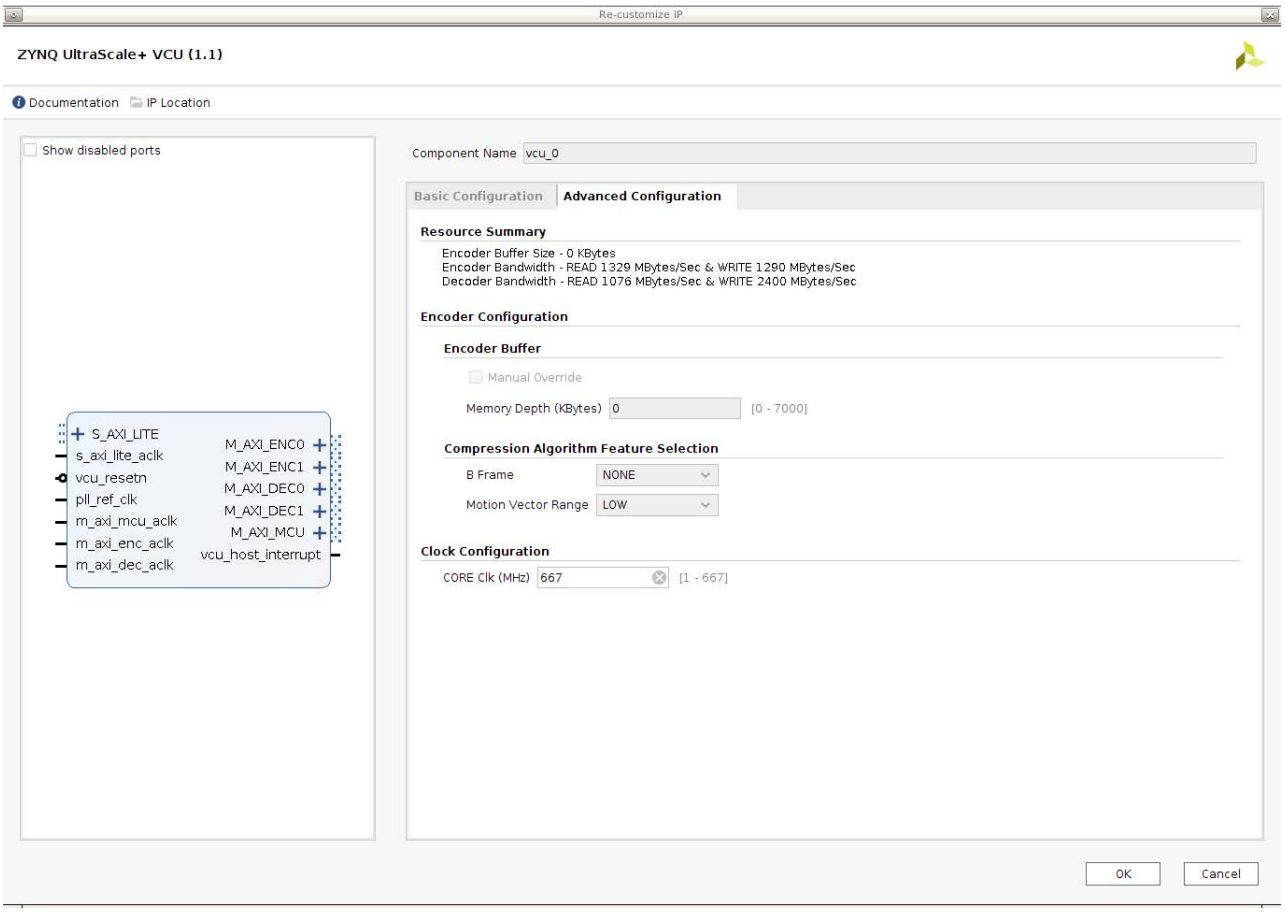

Figure 11-3: **Advanced Configuration Tab**

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 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:
  - NONE - lowest latency

    - STANDARD - GOP configuration IPPP with Intra period of 30ms or GOP configuration IPBBBBPBBBBP with num-b-frames=4.

      Note: 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.

    - HIERARCHICAL - Also known as pyramidal. Works with 3, 5, or 7 B-frames.
- **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 max resolution times the max no. of streams re-presents max bandwidth plan to use on your system.

   For case listed above:

    - Set the **Max Number of Decoded Streams** to **3**.
    - Set the **Resolution** to **1920x1080**.
    - Set **Frames Per Second** to **60**.

Note:

- 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.
- Encoder buffer option is enabled for Intra and Inter frame coding only. The Encoder buffer is used for motion estimation.
- GUI configuration is primarily used to help 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, perform the following steps:

1. Launch the Vivado IDE and create a new project. (Figure 11-4)

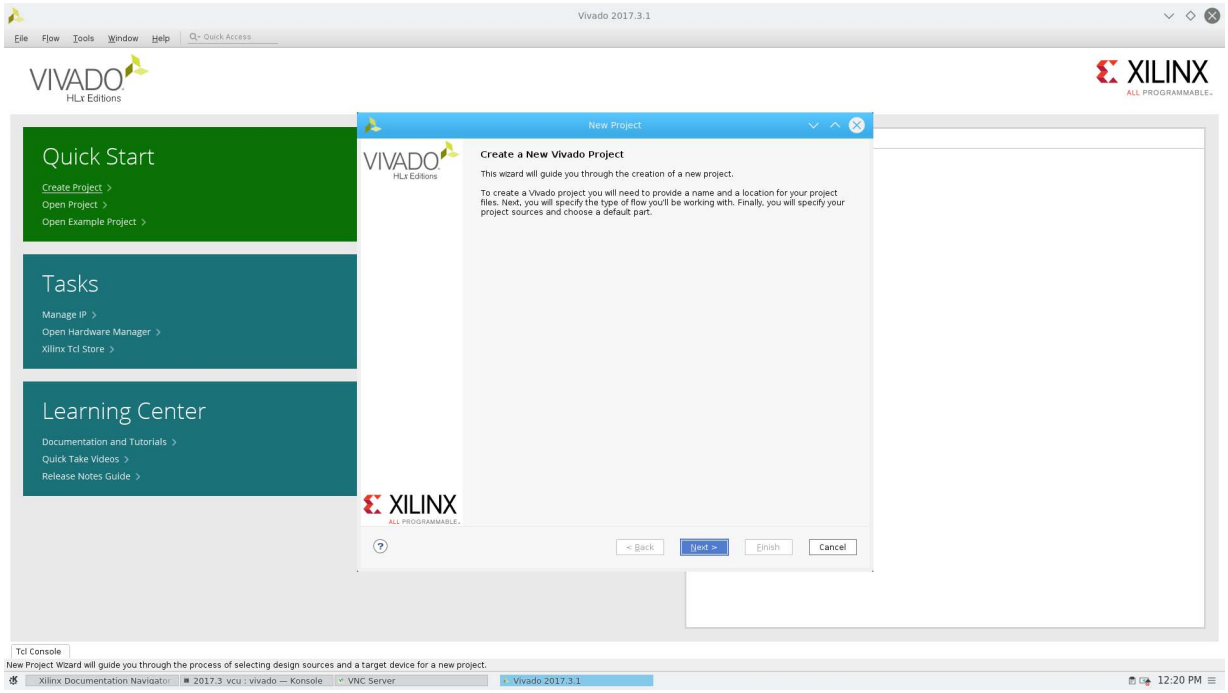

Figure 11-4: **New Vivado Project**

2. Click **Next** on **New Project** wizard until you reach the **Family Selection** window.
3. Select a target device for the VCU core. (Figure 11-5).

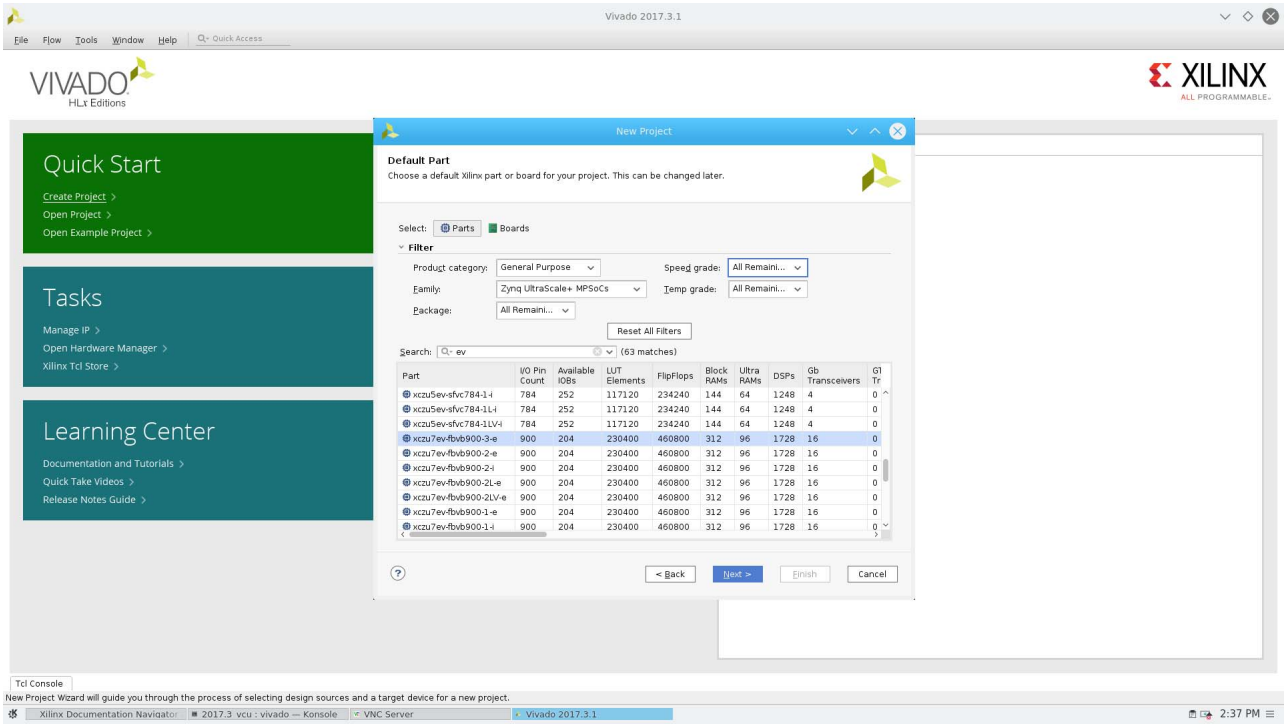

Figure 11-5: **Family Selection Tab**

4. Click on the **Project Settings** window. Click on the Implementation, as shown in Figure 11-6.

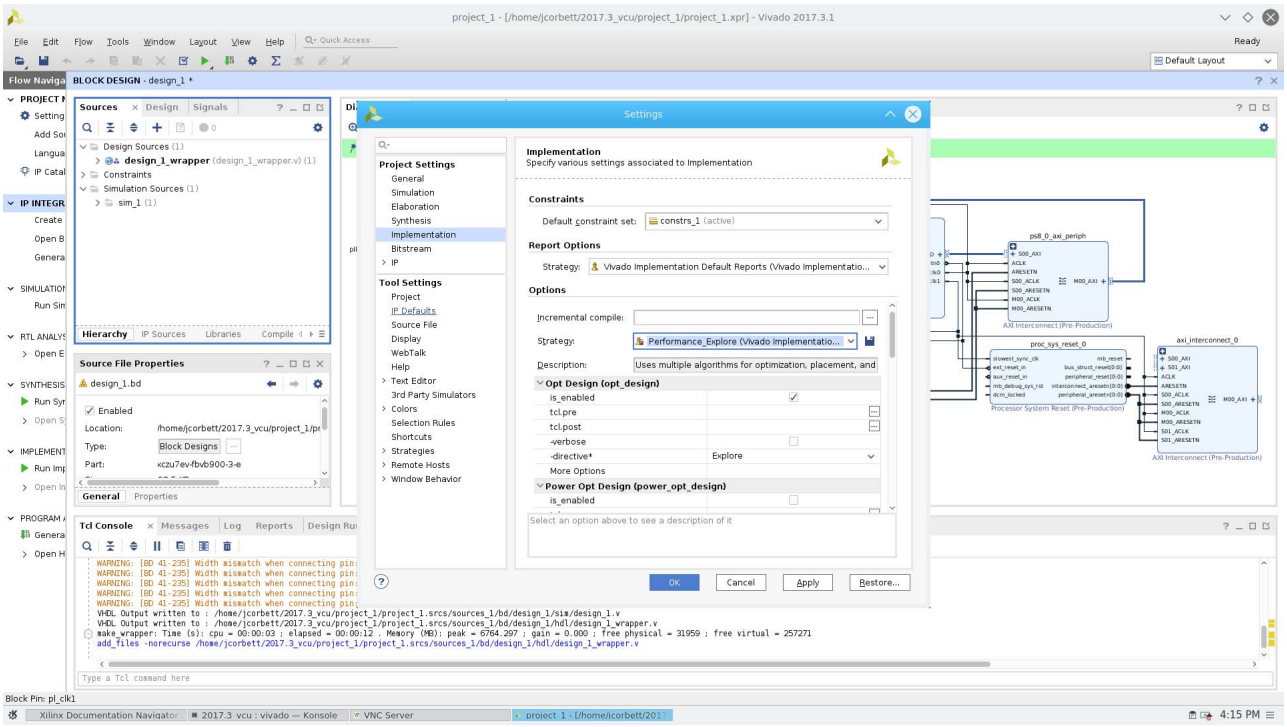

Figure 11-6: **Project Settings Tab**

5. In the **Settings** window, enable the **Performance_Explore** option.
   Settings > Implementation > Options > Strategy: Performance_Explore
   See *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* [Ref 8] for more information.
6. Click on **Create Block Design** option.
7. Click on **Add IP** option and type VCU. The following IP appears.

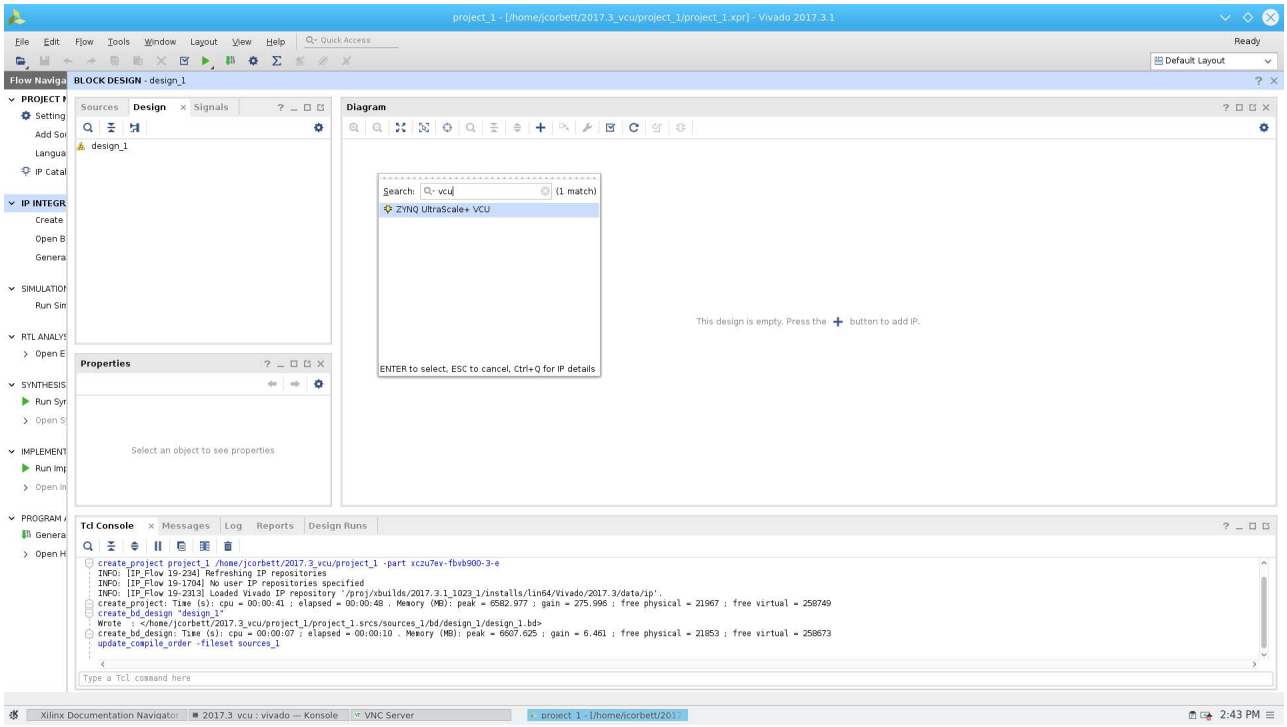

Figure 11-7: **Zynq UltraScale+ VCU**

8. Add Zynq UltraScale+ VCU to the block design.
9. Add Zynq UltraScale+ MPSoC IP to the block design as shown in Figure 11-8.

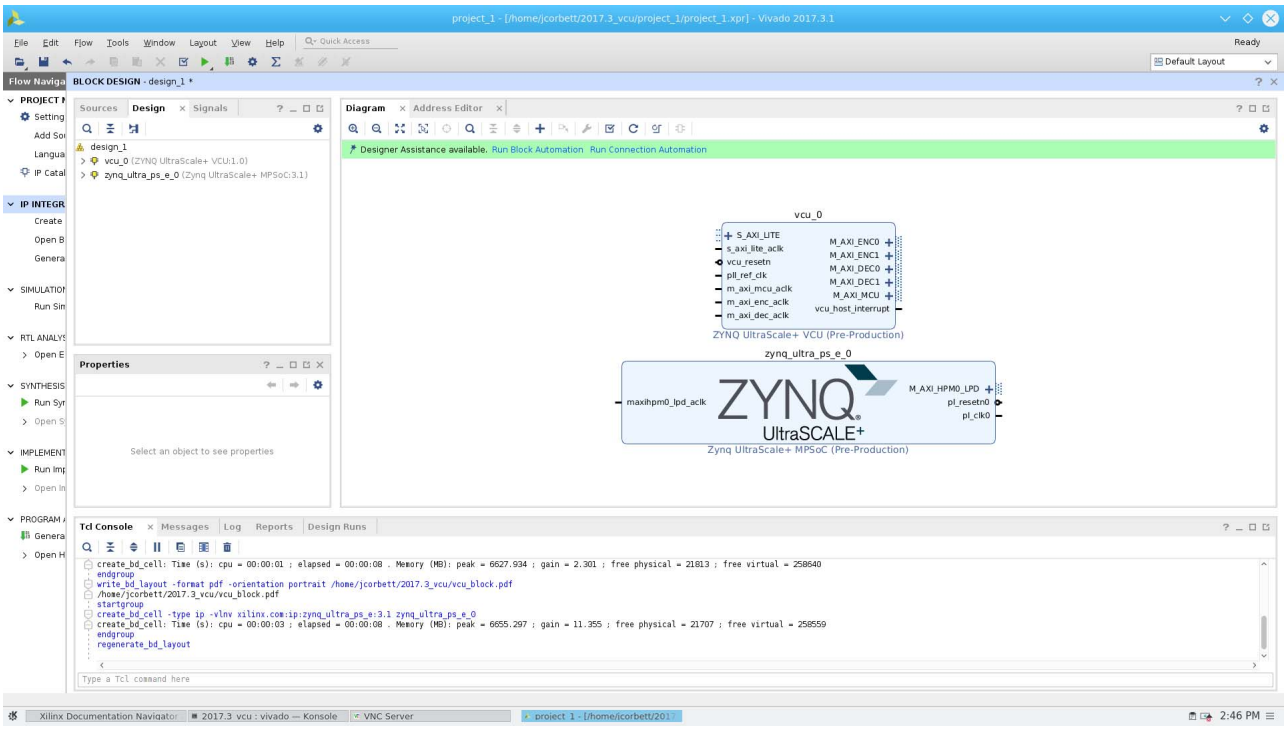

Figure 11-8: **Zynq UltraScale+ MPSoC**

10. Configure Zynq UltraScale+ MPSoC to enable AXI slave interfaces, clocking, and PL-PS interrupt signal per your design requirements. Refer to *Zynq UltraScale+ MPSoC Processing System Product Guide* [Ref 17] for configuration options of the Zynq UltraScale+ MPSoC IP.

    Figure 11-9 shows an example of configuring the PS-PL interface signals.

11. Select PL1 clock frequency as 333 MHz as shown in Figure 11-9.

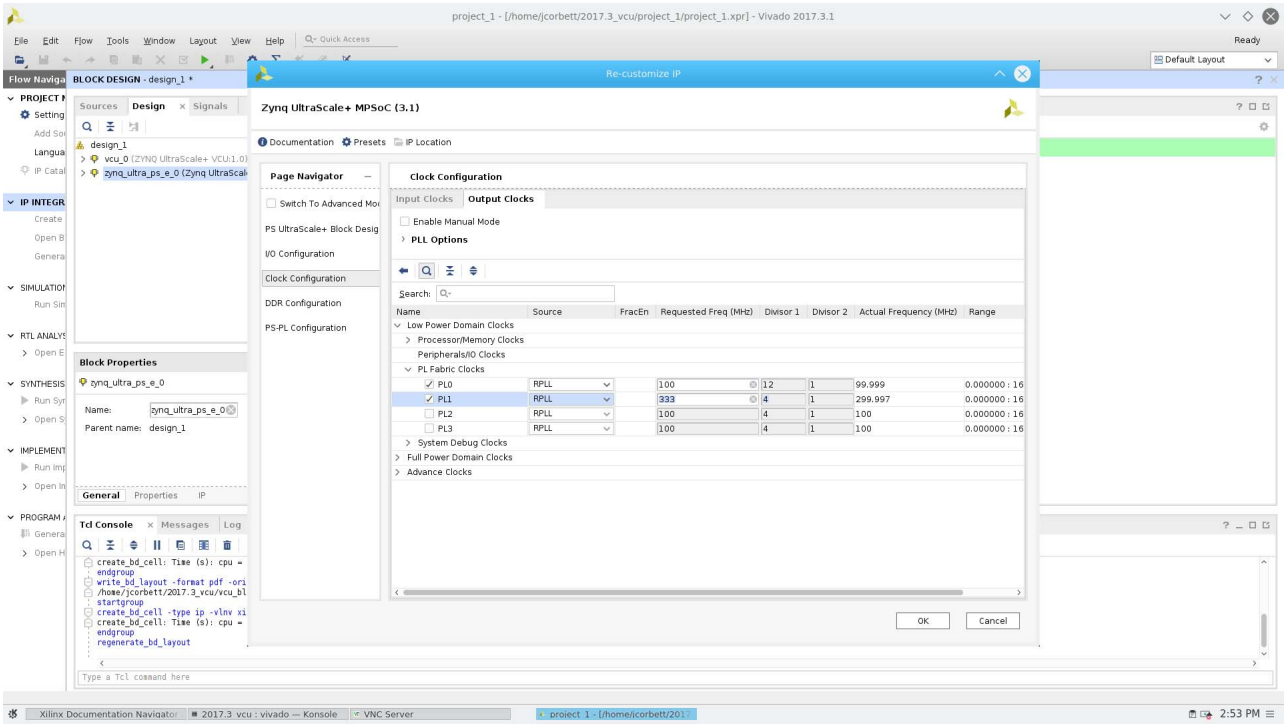

Figure 11-9: **Re-customize IP**

12. Enable IRQ0 [0-7] and HP0-3 ports as shown in Figure 11-10.

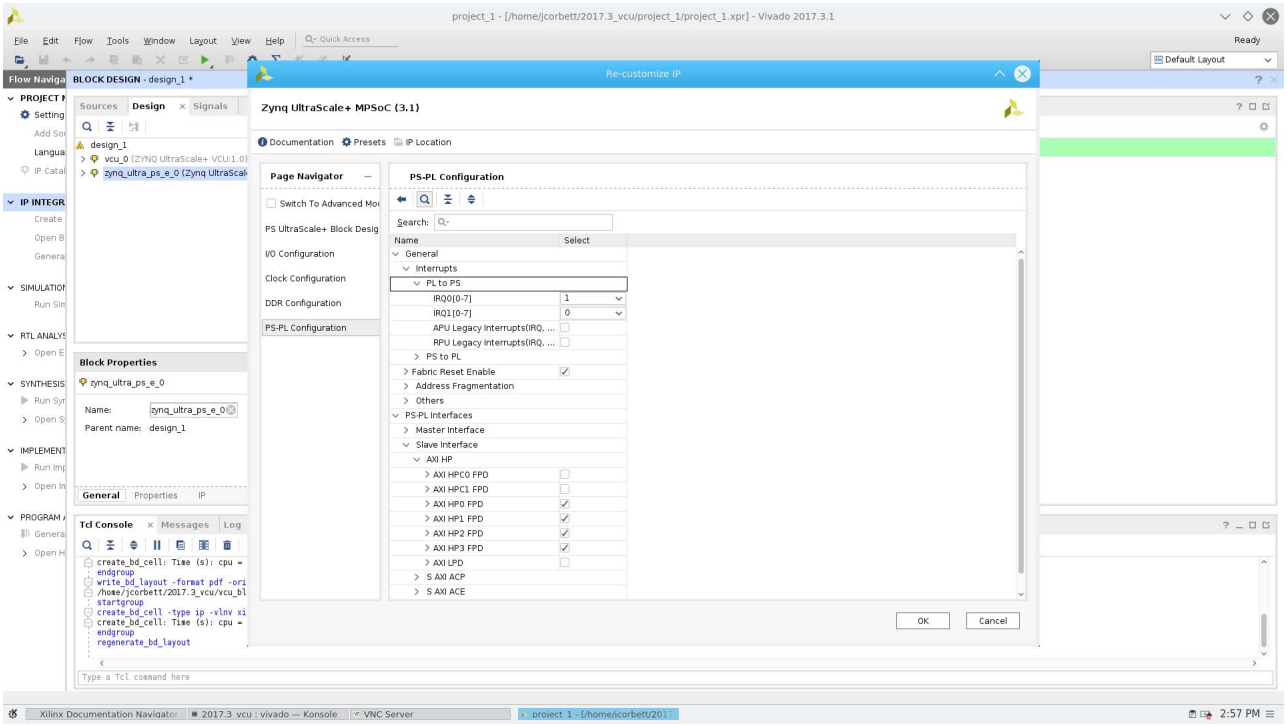

Figure 11-10: **Output PL-PS Configuration**

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

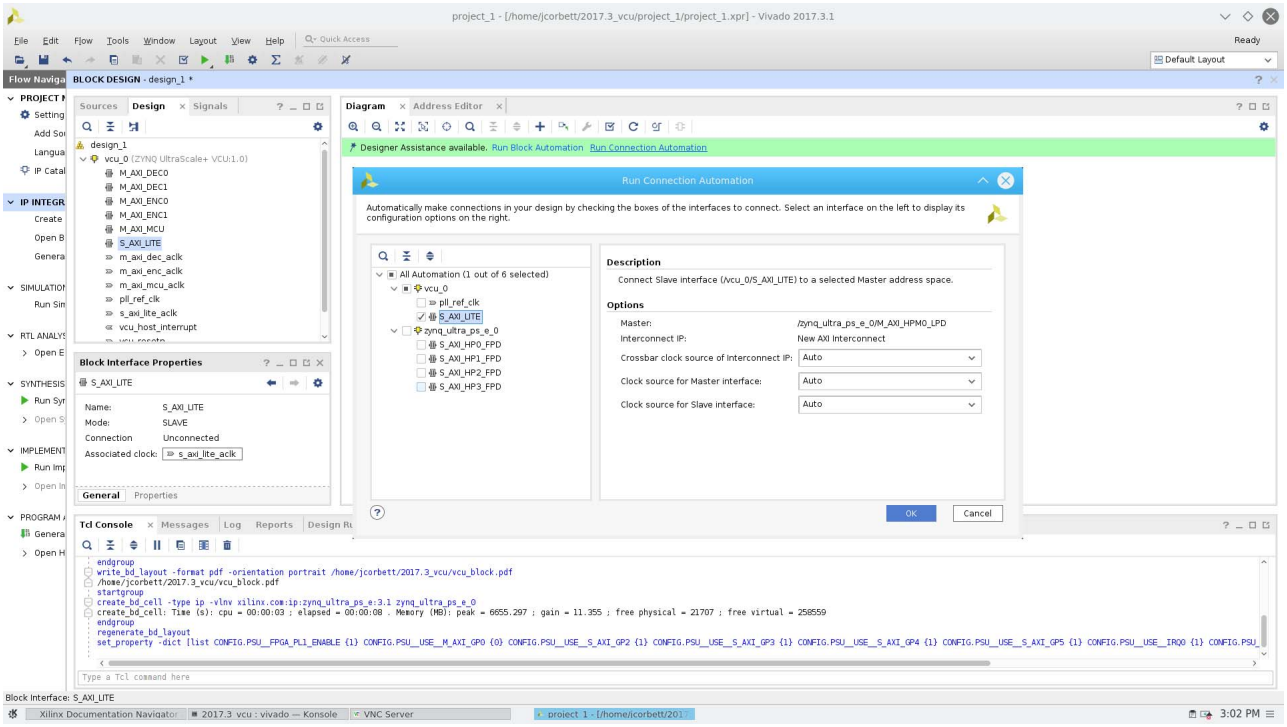

Figure 11-11: **Connection Wizard**

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.

    From *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 14], the address and description of S_AXI_HP1_FPD can be found. 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.

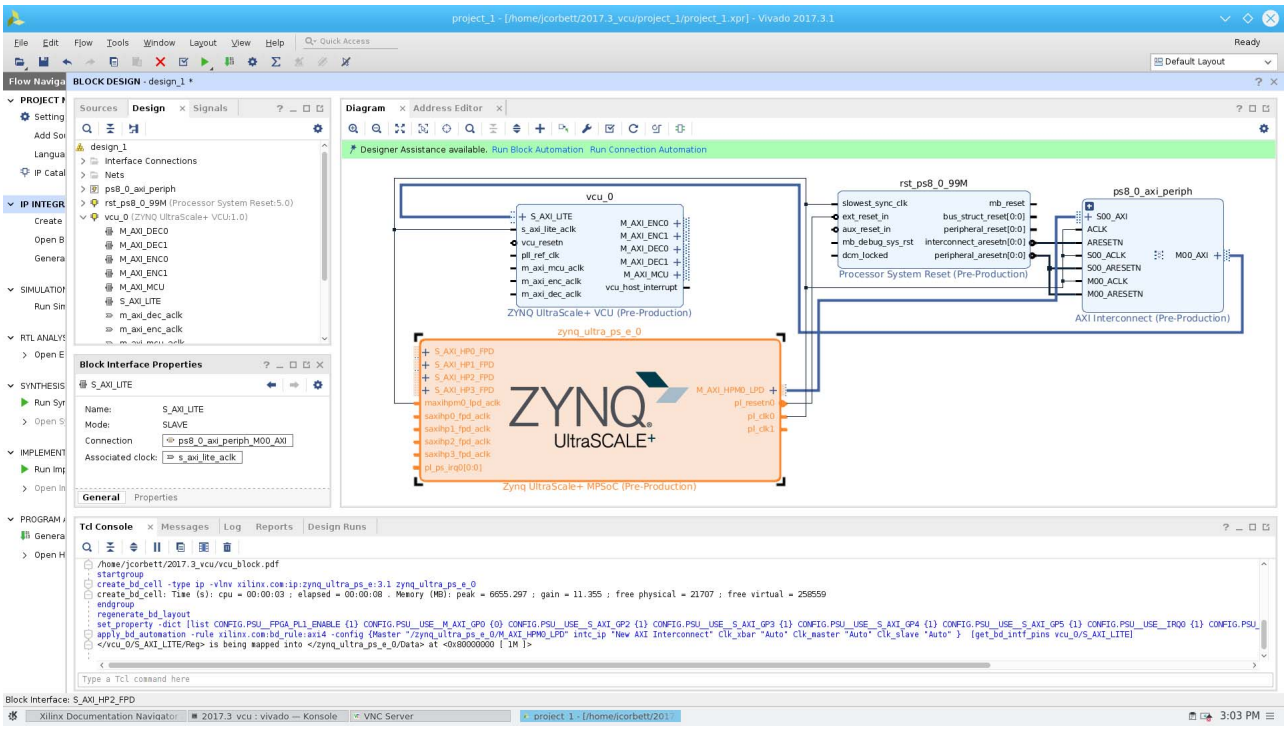

Figure 11-12: **Connection Wizard**

15. Add the AXI Interconnect IP and set number of slave interfaces to **2** and master interface to **1** as shown in Figure 11-13.

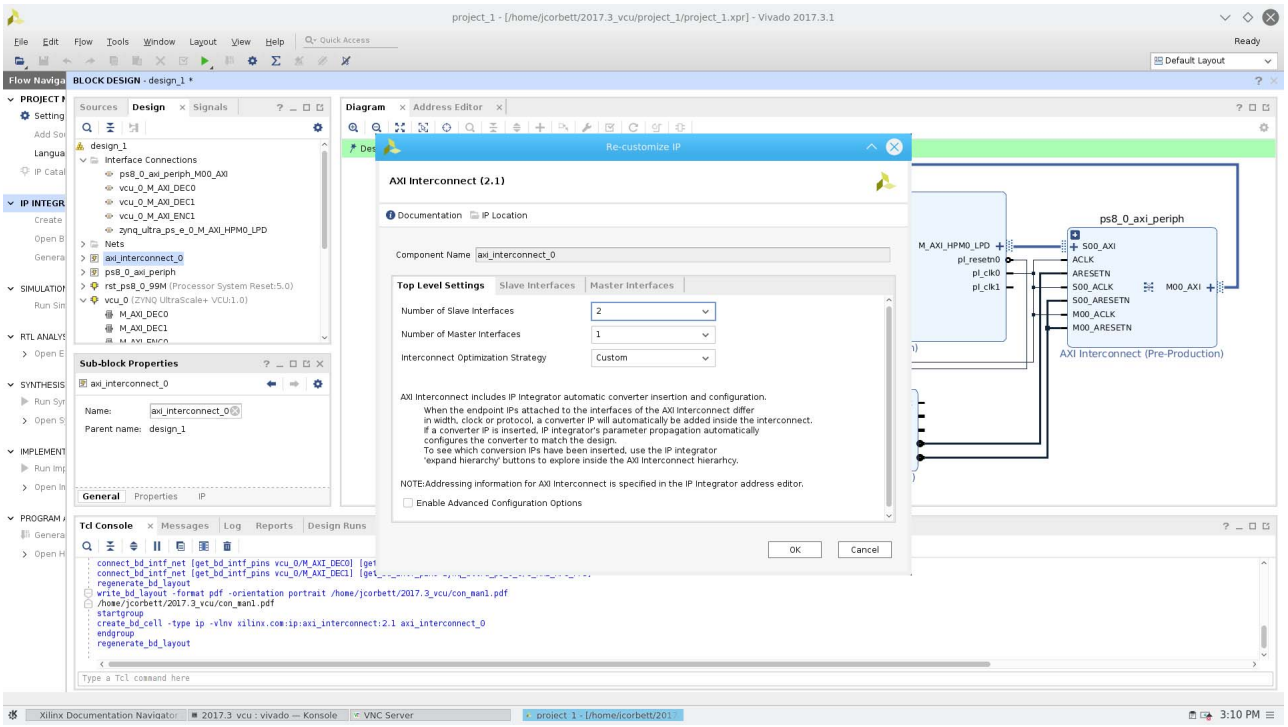

Figure 11-13: **AXI Interconnect**

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 Figure 11-14.
    - 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.

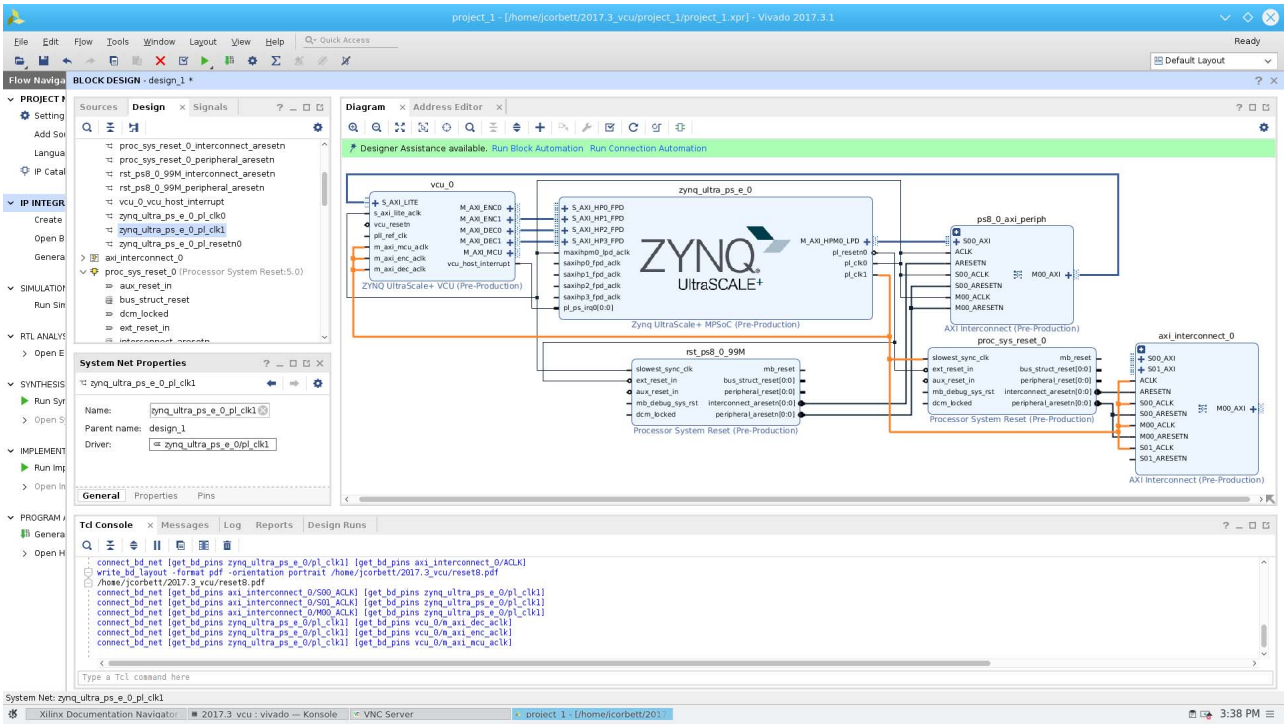

Figure 11-14: **Diagram**

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 **`vcu_resetn`** signal of Zynq UltraScale+ VCU constant 1 using LogiCORE IP.
20. Make **`pll_ref_clk`** signal as external.
21. In the **Address Editor** tab, expand EncData address segment and auto assign the addresses. Figure 11-15 shows an example address map.

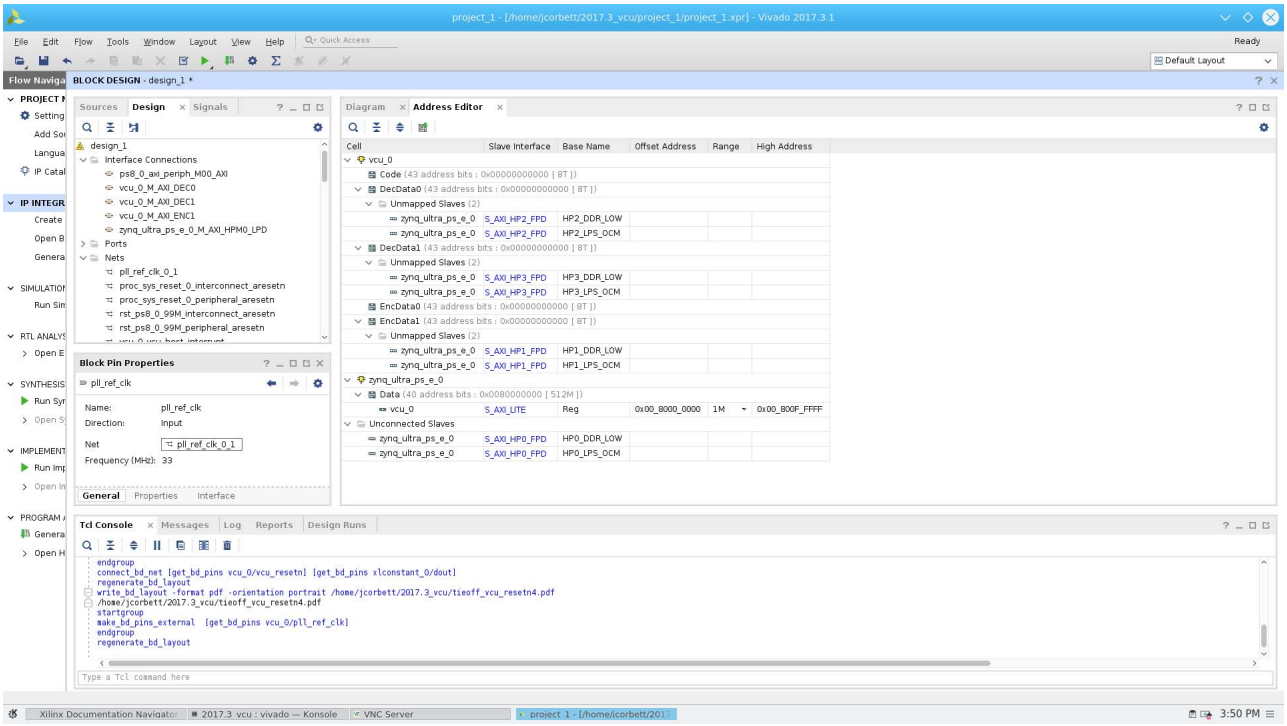

Figure 11-15: **Address Editor**

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

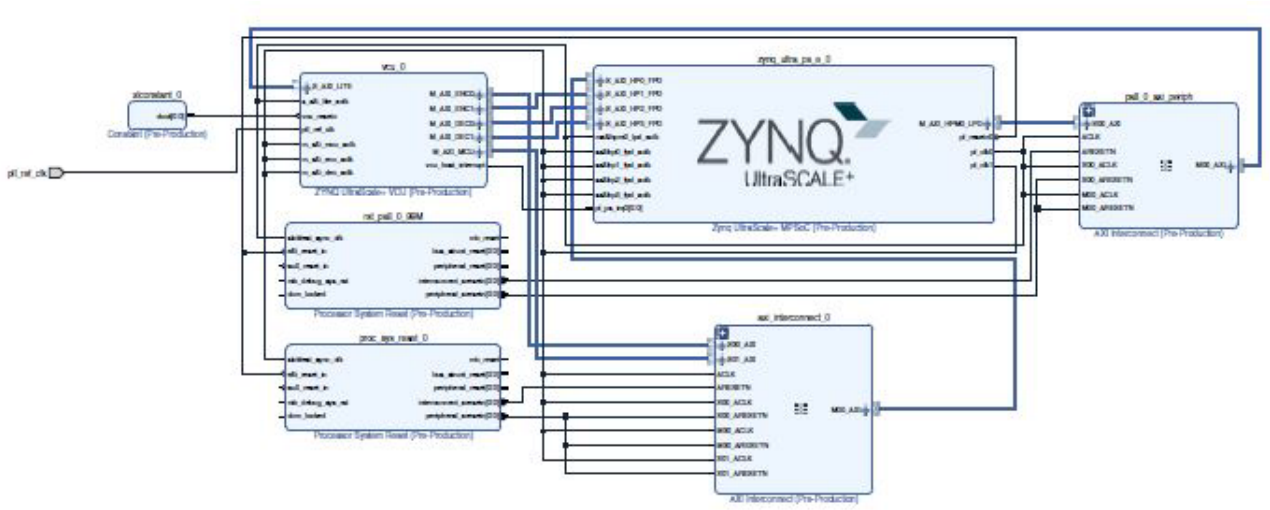

Figure 11-16: **Validate Block Design**

23. Create a top-level Vivado wrapper by right-clicking on Block Design and selecting **Create HDL Wrapper** option as shown in Figure 11-17.

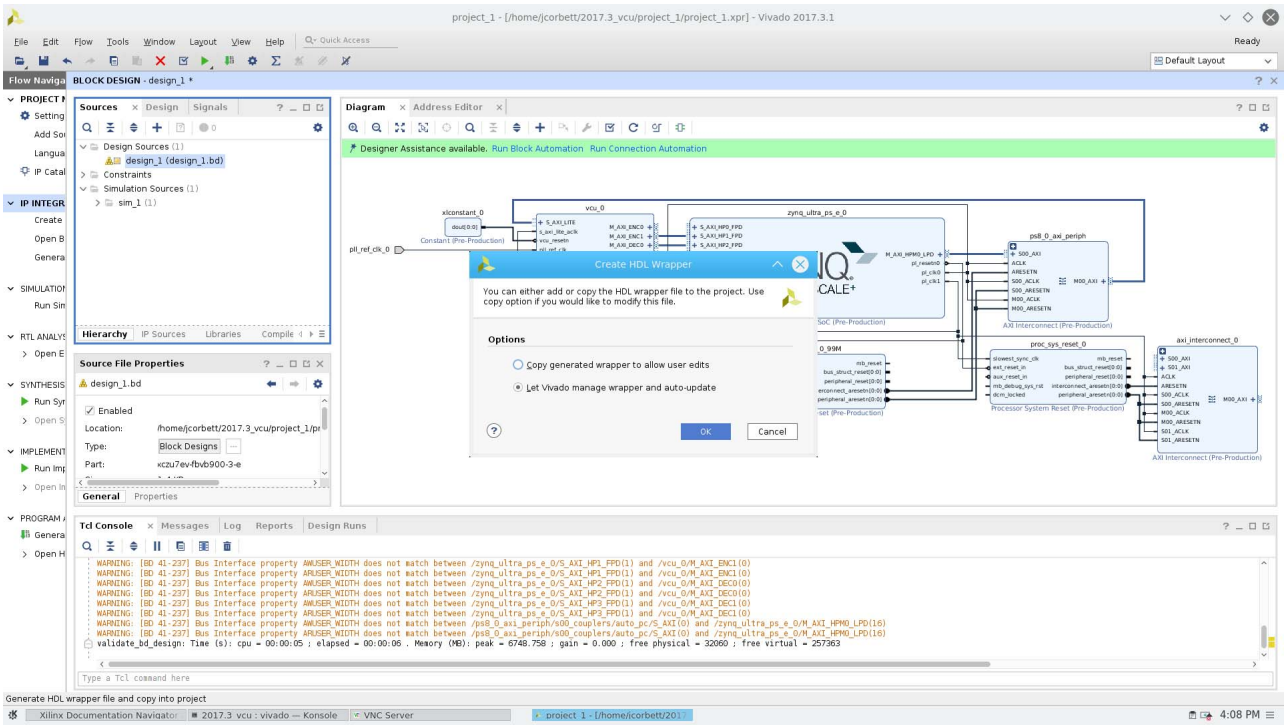

Figure 11-17: **Create HDL Wrapper**

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 Decoder

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's 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 DDR bandwidth would be sufficient to achieve transcoding at 4k@60fps. Figure 11-18 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 PS_DDR to PL_DDR is achieved by DMA transfers.

2 GB Access Limit

The following access limits apply:

- VCU IP can access 4GB range from aligned 4GB base address and MCU can access buffers within 2GB 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.

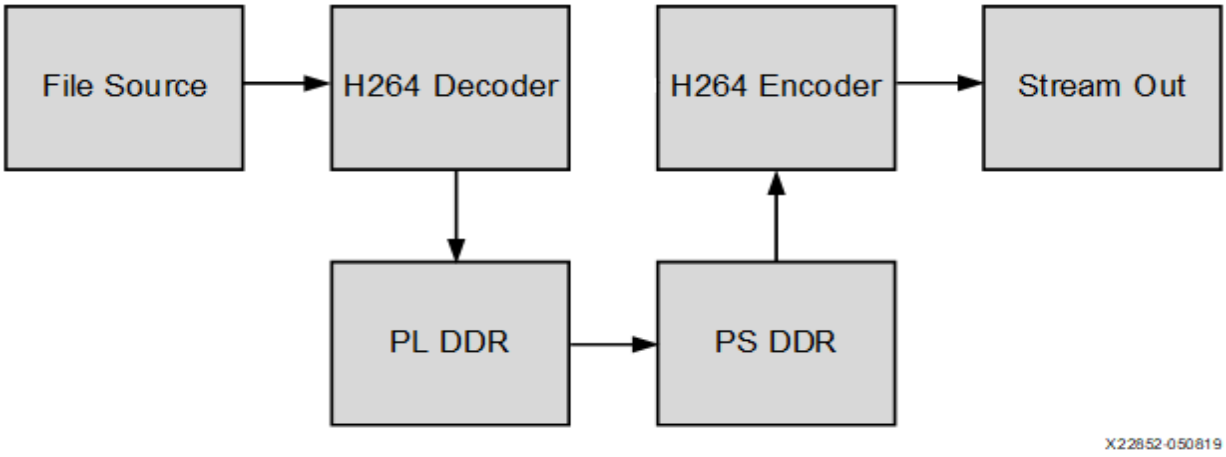

Figure 11-18: **Use Case 2**

Enabling PL-DDR

To enable PL-DDR:

1. Extract the PetaLinux BSP.
2. Update DDR, Decoder, and mem2mem nodes required 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.
   b. Update the device tree node to have a dedicated memory for the VCU decoder.
   c. The VCU decoder device tree node changes to allocate memory from PL_DDR.

   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 are:

```
/include/ "system-conf.dtsi"

/ {

};

&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>;

        plmem_vcu_dec: pool@0 {
                reg = <0x48 0x00000000 0x0 0x70000000>;
        };
};

&decoder {
        xlnx,dedicated-mem = <&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
```

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  ! v4l2video0convert
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! v4l2video0convert
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 max 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) [Ref 2].

XILINX

Chapter 12

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 Figure 12-1. 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. [Ref 22].

The OpenMAX Integration Layer API [Ref 21] 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 AL_Encode and AL_Decode 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.

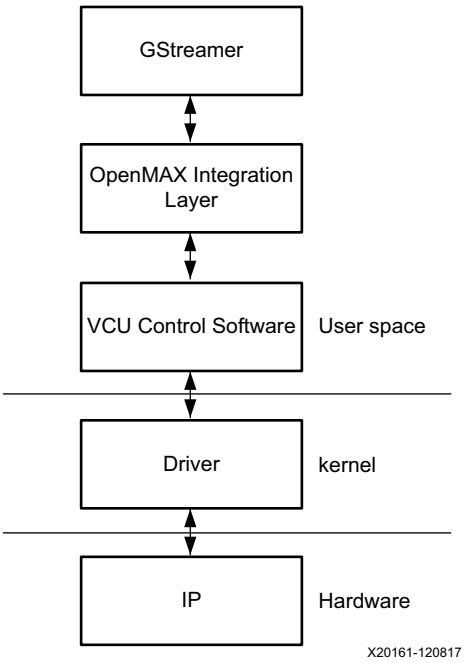

Figure 12-1: **VCU Software Stack**

Software Prerequisites

All of the software prerequisites for using the VCU are included in Xilinx® PetaLinux included in Xilinx Software Development Kit (SDK) 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.

The application software using the VCU is written on top of the following libraries and modules, shown in Table 12-1.

Table 12-1: **Application Software**

Software	Version	Source
GStreamer plugins		https://github.com/Xilinx/gst-plugins-good/tree/rel-v2019.1 https://github.com/Xilinx/gst-plugins-base/tree/rel-v2019.1 https://github.com/Xilinx/gst-plugins-bad/tree/rel-v2019.1 https://github.com/Xilinx/gstreamer/tree/rel-v2019.1 https://github.com/Xilinx/gst-omx/tree/rel-v2019.1
Software: Gstreamer library	1.14.4	https://gstreamer.freedesktop.org/
OpenMAX™ Integration Layer API	1.1.2	https://github.com/Xilinx/vcu-omx-il/tree/master-rel-2019.1

Table 12-1: **Application Software *(Cont'd)***

Software	Version	Source
VCU Control Software	1.0.44	https://github.com/Xilinx/vcu-ctrl-sw/tree/master-rel-2019.1
VCU firmware	1.0.0	https://github.com/Xilinx/vcu-firmware/tree/master-rel-2019.1
VCU kernel modules		https://github.com/Xilinx/vcu-modules/tree/master-rel-2019.1
VCU recipe files		https://github.com/Xilinx/meta-xilinx/tree/rel-v2019.1/meta-xilinx-bsp/recipes-multimedia/vcu
Gstreamer recipe files		https://github.com/Xilinx/meta-petalinux/tree/rel-v2019.1/recipes-multimedia/gstreamer

Encoder Features

The VCU supports multi standard video encoding, shown in Table 12-2.

Table 12-2: **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) 3840×2160p60 3840×2160p30 1920×1080p60 1920×1080p30 1280×720p60 1280×720p30	4096x2160p60 with specific device (-2,-3) 3840×2160p60 3840×2160p30 1920×1080p60 1920×1080p30 1280×720p60 1280×720p30
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

Table 12-2: **Encoder Features *(Cont'd)***

Video Coding Parameter	H.265 (HEVC)	H.264 (AVC)
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
Bit Rate	Limited by level and profile	Limited by level and profile

Note:

1. The H.265 (HEVC) minimum picture resolution is 128×128.
2. The H.264 (AVC) minimum picture resolution is 80×96.

The tier and level limits are shown in Table 12-3.

Table 12-3: **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
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/4K resolution in Subframe/Normal latency mode.

Mode	Encoding	FullHD	4K
Normal	AVC	68	135
	HEVC	34	22
Low latency	AVC	32	32
	HEVC	32	22

GStreamer Encoding Parameters

The GStreamer encoding parameters are shown in Table 12-4.

Table 12-4: **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 (Not Supported) 4 = Constant Skip Frames (Not Supported) 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	target-bitrate	Target bitrate in Kbps **Default value**: 64
Maximum Bit Rate	max-bitrate	Max bitrate in Kbps, used in VBR rate control mode. **Default value**: target-bitrate. The max-bitrate value should always be ≥ target-bitrate **Default value**: 64
Profile	Through caps	Supported profiles for corresponding codec are mentioned in Table 12-2 **Default value**: HEVC_MAIN
Level	Through caps	Supported Levels for corresponding codec are mentioned in Table 12-2 **Default value**: 51
Tier	Through caps	Supported Tier for H.265 (HEVC) codec is mentioned in Table 12-2 **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
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

Table 12-4: **GStreamer Encoding Parameters *(Cont'd)***

VCU Parameter	GStreamer Property	Description
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 will be limited by "Max # of tile rows MaxTileRows". Refer to Table 12-3. **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 12-4: **GStreamer Encoding Parameters *(Cont'd)***

VCU Parameter	GStreamer Property	Description
GOP Configuration	gop-mode	Specifies Group of Pictures configuration 0 = basic (IPPP mode with Intra period of 30) 1 = pyramidal (hierarchical) (Advanced GOP pattern with hierarchical B-frames. Works with B=3, 5 and 7) 2 = adaptive (Adaptive B-frames) 3 = low_delay_p (IPPPPP...) 4 = 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**: 0
Filler data	filler-data	Enable/Disable filler data adding functionality in CBR rate control mode. Boolean: true or false **Default value**: True

Table 12-4: **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**: 1
Initial Removal Delay	initial-delay	Specifies the initial removal delay as specified in the HRD model in milliseconds. Not used when control-rate = disable **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

Table 12-4: **GStreamer Encoding Parameters *(Cont'd)***

VCU Parameter	GStreamer Property	Description
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**: H.264(AVC): 16 H.265(HEVC): 32
Slice Height	slice-height	Specify 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/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

Table 12-4: **GStreamer Encoding Parameters *(Cont'd)***

VCU Parameter	GStreamer Property	Description
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.
Max Quality Target	max-quality-target	Caps quality at certain limit keeping the bit rate variable. Only used with capped variable rate-control. Range: 0 to 20 (20=lossless quality) Default value: 14

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 HW 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 (a.k.a. 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.

Decoder Features

Table 12-5: **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
Levels	Up to 5.1 High Tier	Up to 5.2
Resolutions	4096x2160p60 with specific device(-2,-3) 3840×2160p60 3840×2160p30 1920×1080p60 1920×1080p30 1280×720p60 1280×720p30	4096x2160p60 with specific device(-2,-3) 3840×2160p60 3840×2160p30 1920×1080p60 1920×1080p30 1280×720p60 1280×720p30
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 12-6: **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 usecases. For example, uses cases where the bitrate is more than 100 Mb/s.
Latency	low-latency	Specifies decoder latency mode. (0): false - Normal mode (1): true - if alignment is AU, reduced-latency - if alignment is NAL, low-latency **Default value**: 0. **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.

Decoder Maximum Bit Rate Tests

Pipeline used for measuring maximum bit rate for which decoder produces 30 fps.

Table 12-7: **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 ! fpsdisplaysink
name=fpssink text-overlay=false video-sink=kmssink sync=true
fps-update-interval=3000 -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 encoder input and decoder output side.
- 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 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 12-8: **Gstreamer and V4L2 Formats**

Pixel Format	V4L2	GStreamer
Y_Only 8-bit	V4L2_PIX_FMT_GREY	GST_VIDEO_FORMAT_GRAY8
Y_Only 10-bit	V4L2_PIX_FMT_XY10	GST_VIDEO_FORMAT_GRAY10_LE32
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: The fourcc codes are the last 4 characters of v4L2 pixel formats (GREY/XY10... etc.).

Preparing PetaLinux to Run VCU Applications

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 Chapter 10, Designing with the Core regarding the ports connected to the decoder. Understanding where the addresses mentioned below came from will enable you to adjust the commands below for designs using alternate AXI connection. See the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 14] 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
   - Set the S_AXI_HP3_FPD RDISSUE (AFIFM) register to allow 16 commands

```
devmem 0xFD3B0004 w 0xF
```

   - Set the S_AXI_HP3_FPD WRISSUE (AFIFM) register to allow 16 commands

```
devmem 0xFD3B0018 w 0xF
```

Now above mentioned 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 project-spec/meta-user/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 e.g. If patch fix is for gst-omx, follow below steps

      - Create "gstreamer1.0-omx" directory in "recipe-multimedia/gstreamer" folder

```
mkdir gstreamer1.0-omx
```

- 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
```

- Create “gstreamer1.0-omx_%.bbappend” file in “recipe-multimedia/gstreamer” folder
- Add below lines in “gstreamer1.0-omx_%.bbappend” file

```
FILESEXTRAPATHS_prepend: = "${THISDIR}/gstreamer1.0-omx:"
SRC_URI_append = " \
file://test1.patch \
 file://test2.patch \
"
```

Note: Create similar bbappend files and folder for other gstreamer package to integrate any custom patches in PetaLinux Build.

4. For VCU patches, follow below steps

a. Create “vcu” directory in recipe-multimedia folder.

```
cd project-spec/meta-user/recipe-multimedia
mkdir vcu
```

b. There are 4 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 e.g. If patch fix is for vcu drivers, follow below steps

- Create “kernel-module-vcu” directory in “recipe-multimedia/vcu” folder

```
mkdir kernel-module-vcu
```

- Copy vcu driver patches in “kernel-module-vcu” directory.

```
cp test1.patch recipe-multimedia/vcu/kernel-module-vcu
cp test2.patch recipe-multimedia/vcu/kernel-module-vcu
```

- Create “kernel-module-vcu_%.bbappend” file in “recipe-multimedia/vcu” folder
- Add below lines in “kernel-module-vcu_%.bbappend” file

```
FILESEXTRAPATHS_prepend: = "${THISDIR}/ kernel-module-vcu:"
SRC_URI_append = " \
file://test1.patch \
 file://test2.patch \
```

“

Note: 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: For users not compiling with PetaLinux, ensure to review the recipes for additional files necessary for setting up GStreamer. For example, you must include the following file in the root file system: **/etc/xdg/gstomx.conf**. This file tells gst-omx where to find the OMX integration layer library - libOMX.allegro.core.so.1.

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 to **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 compressed video files can be downloaded using host machine and copy input files into "`/home/root/`" folder. Use the following commands to download the content on 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 two 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)**.

  - Running 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 below command to play youtube video, make sure ethernet is connected to board

  ```
  vcu-demo-decode-display.sh -u "youtube-URL"
  ```

- Run below 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 below command for USB video capture serial pipeline

```
vcu-demo-camera-encode-decode-display.sh -s 640x480
```

3. Run below command 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"
```

   Note: For 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-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 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.

Example-3: (USB-Camera → Decode → Display)

1. Connect USB camera to the board (Verified Cameras: Logitech HD camera, C920).
   - Run below command for USB video decode display pipeline

```
vcu-demo-camera-decode-display.sh -s 1920x1080
```

   - Run below 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.*

- Capabilities can be found using "v4l2-ctl --list-formats-ext".
- V4lutils if not installed in the pre-built image, need to install using dnf or rebuild 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 "/home/root" folder.

1. Use below 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, sample command is mentioned 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 stream-out (as a server) and board 2 is used for streaming-in and decode purpose (as a client) or VLC player on the host machine can be used as 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 Ethernet cable or make sure both the boards are connected to same Ethernet hub.
   a. If VLC player is used as client, ensure that the host machine and server (board-1) are connected to same Ethernet hub or Connect them back to back using Ethernet cable.
   - Client settings

     Set client IP address and execute stream-in → Decode example on board 2, if board-2 is used as client and connected the boards back to back with ethernet cable.

```
ifconfig eth0 192.168.0.2
```

Note: Setting of ip is not required if the boards already connected to 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 VLC player is used as client then, set the host machine IP address to 192.168.0.2, if it is connected to board directly with ethernet cable.

Note: Setting of IP address is not required if the boards already connected to same LAN

4. Create test.sdp file on host with below content (Add separate line in test.sdp for each item below) 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 firewall in 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 already connected to 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 USB camera to the board (Verified Cameras: Logitech HD camera C920).
2. Execute below command for USB video recording.

```
vcu-demo-camera-encode-file.sh -a aac -n 1000 --use-alsasrc -i "hw:1"
```

Output file **camera_output.ts** is generated on current directory which will have recorded Audio/Video file in 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 stream-out (as a server) and board 2 is used for streaming-in and decode purpose to play received Audio/Video stream (as a client).

1. Connect two boards back to back with Ethernet cable or make sure both the boards are connected to same Ethernet hub.
2. Set 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 already connected to 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
   ```

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.
  ```

- ALSA sound device names for playback devices
  - aplay -l

  ```
  **** 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 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.

VCU - Encoder Feature Description

The VCU encoder supports the following features.

- Dynamic Parameters
  - Target-bitrate
  - Gop-Length
  - Number of B-frames
- Inserting Key Frame (IDR)
- Region of Interest Encoding
- Long-term Reference Picture support
- Adaptive GOP
- Dual pass encoding
- SEI Insertion and Extraction
- Scene Change Detection
- Interlaced Video
- GDR Intra Refresh
- Dynamic Resolution Change – VCU Encoder/Decoder.

- Frameskip support for VCU Encoder
- 32-streams support
- DCI-4k Encode/Decode
- External QP Table
- Temporal-ID support for VCU Encoder

Both the Xilinx VCU Control Software application and 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 the GStreamer application is part of gst-omx repo (https://github.com/Xilinx/gst-omx/tree/xilinx-master/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 ex: 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, later user can alter the bitrate during encode.

Dynamic-Bitrate

Dynamic-bitrate is the ability to change encoding bitrate (target-bitrate) while encoder is active.

To change the bitrate of 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>:<frame number>:<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 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

Region of Interest Encoding tags regions in a video frame to be encoded with user supplied quality (high, medium, low, and dont-care) relative to the picture background (untagged region). An example ROI defined in a frame is shown in Figure 12-2.

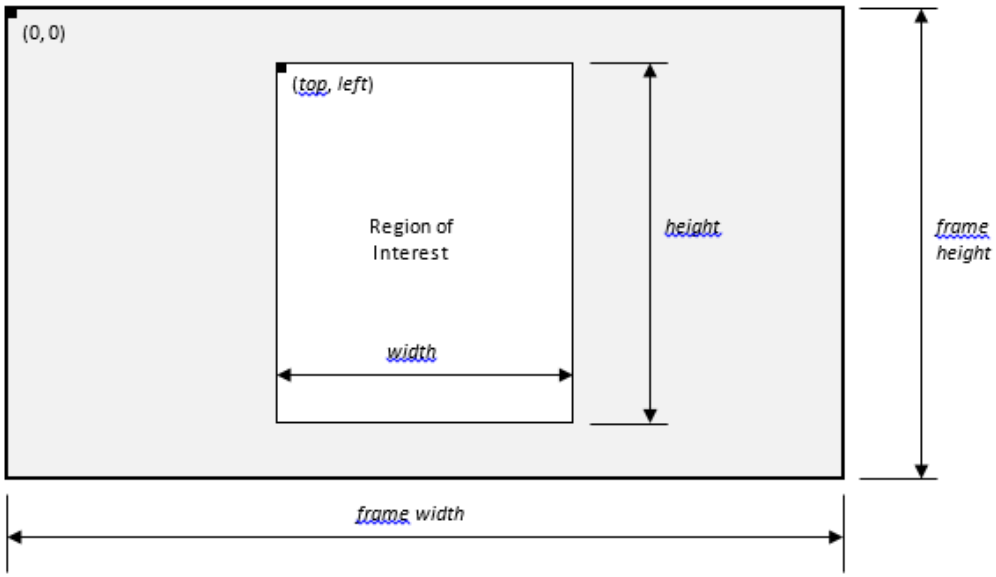

Figure 12-2: **Region of Interest Encoding**

The user provides 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 users 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
```

ROI-QP

You can set ROI live streaming dynamically using the GStreamer and Control SW.

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-xlnx/blob/xilinx-master/examples/zynqultrascaleplus/test-vcu-encode.c line number #301.

- gst_buffer_add_video_region_of_interest_meta()
- gst_video_region_of_interest_meta_add_param()

CtrlSW:

The Control-SW exports a few APIs to provide ROI information to encoder per each 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 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 longterm picture at frame 10 is:

```
>> zynqmp_vcu_encode -w 3840 -h 2160 -e avc  -o /run/op.h264 -i /run/input.yuv -d
IL:10
```

cmd line option to use LTR for frame 15 is:

```
>> zynqmp_vcu_encode -w 3840 -h 2160 -e avc  -o /run/op.h264 -i /run/input.yuv -d
UL:15
```

Adaptive GOP

- User specifies 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 encoder configuration file at control software and gop-mode=adaptive at GStreamer
- The encoder adapts the number of B 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 specified by the user

Insertion of SEI Data at Encoder

- **`AL_Encoder_AddSei()`**: Adds SEI NAL to the stream. User is responsible for the sei payload and has to write it as specified in Annex D.3 of ITU-T. The encoder does the rest including anti-emulation of the payload
- Limitation: The SEI can't exceed 2KB. 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.
- 2KB limit is per stream buffer, i.e. all SEI messages per AU should be with-in the 2KB limit.
- SEI messages are already synchronized with corresponding video frames, there should not be a need for any 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
  - Indexes
    - 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 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 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 ctrlSW. You need to maintain a FIFO, if there is any re-ordering due to B-frames.

Refer to 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.

Two levels:

- Real-time GOP/frame-level dual-pass < lookahead mode >
- Offline sequence level dual-pass < two pass modes >

The 2019.1 release supports only Real-time GOP/Frame-level dual-pass.

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 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 histogram on a vertically subsampled luma frame for consecutive frames. The histogram of these frames are then are 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 *Video Scene Change Detection: LogiCORE IP Product Guide* [Ref 19].

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 8bit and 10bit 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 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.

The most important application of scene change detection are video surveillance, machine learning and video conference. In these use cases 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 encoder, where the encoder makes decision to insert I-frame instead of P-frame or B-frame. Inserting an I-frame at the place where scene is changed would retain the quality of the video.

Gstreamer pipelines:

1. 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
```

2. 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 continues 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:

1. 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
```

2. v4l2src -> omxh265enc -> omxh265dec -> kmssink

```
gst-launch-1.0 v4l2src io-mode=4 ! video/
x-raw,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"
```

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) encoder inserts Intra MBs row/column in the picture to refresh 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 GDR is enabled, Encoder inserts SPS/PPS along with SEI recovery message at start of refresh pattern, and Decoder can synchronize to SEI Recovery points in the bit-stream. Advantage of having non-IDR Decoder synchronization is, no need to send IDR pictures in case of packet loss scenario. Inserting periodic IDR pictures in video encoding generates spike in bit consumption which is not recommended for video streaming

A sample gstreamer pipeline is as follows:

Use "gdr-mode=horizontal" & "periodicity-idr=<higher than Picture Height in Mbs>" in the encoder element.

Example:

```
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 Control software level add below section and parameter (in GOP section of cfg file)

```
[GOP] #-------------------------------------------------
GopCtrlMode                 = LOW_DELAY_P
Gop.GdrMode                 = GDR_HORIZONTAL
Gop.FreqIDR                 = 270
```

Dynamic Resolution Change – VCU Encoder/Decoder

- Dynamic Resolution Change is supported at the VCU control software level for both the Encoder/Decoder. This feature is not supported at the OMX/Gstreamer 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 (what?) 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
```

- **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 .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 input_cmd.txt file

```
50: Fps=30, Input=1
90: Fps=60, Input=2
```

- **Command**

```
ctrlsw_encoder -cfg test.cfg -o output
```

- The output file that is generated contains an encoded file containing various resolutions.

Frameskip support for VCU Encoder

- **Feature description**:
  - 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.
- **Constraints**:
  - This only applies if the reference frame has already been encoded. The first frame will be encoded irrespective of the buffer overflow. Only subsequent pictures will be discarded if actual frame size is exceeded.

    Note: 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 bitrate or fps change. Skip frames can be added or missed when those parameters change.
  - Back-and-forth effect visual effect can appear when the value of NumB is greater than 1 and **`enable-skip`** is set to true.
  - In Dual Pass mode, frame skip can only be used on the second pass.
  - Inserting frame skip in GOP pattern with more than 1 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 Low Latency (--slicelat) mode.
  - Frameskip is not supported in Pyramidal GOP.
- **Ctrl-SW API**:
  - There will be an additional bEnableSkipFrame flag in the AL_TEncSettings structure like the existing bEnableFillerData
- **OMX API**:
  - New SetParam Index: OMX_ALG_IndexParamVideoSkipFrame
  - Data structure: OMX_ALG_VIDEO_PARAM_SKIP_FRAME
- For enabling frameskip, enable “EnableSkip” parameter in cfg file

```
EnableSkip                    = TRUE
```

- **Command**

```
ctrlsw_encoder -cfg test.cfg
```

32-Streams Support

The following figure shows the 32-streams support:

Use-case	B_frames	Resolution	Format	Working Instances					Comment
				Encode Only		Decode Only		Transcode	
				Ctrlsw	gstreamer	Ctrlsw	gstreamer	gstreamer	
Encode Only: AVC **Decode Only:** AVC **Transcode:** AVC -> HEVC	**Encode Only:** 4 **Decode Only:** 4 **Transcode:** decode: 4 & encode: 3	640x480p30	nv12	32 inst -> 17Fps 24 inst -> 32Fps	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 42Fps	32 inst -> 30Fps	32 inst -> 16Fps 25 inst -> 30Fps	
			nv16	32 inst -> 18Fps 25 inst -> 32fps	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 32Fps	32 inst -> 30Fps	32 inst -> 16Fps 24 inst -> 30Fps	
			xv15	32 inst -> 17Fps 25 inst -> 32Fps	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 35Fps	32 inst -> 30Fps	32 inst -> 16Fps 24 inst -> 30Fps	
			xv20	32 inst -> 16Fps 25 inst -> 32Fps	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 28Fps 29 inst -> 30Fps	32 inst -> 30Fps	32 inst -> 16Fps 24 inst -> 30Fps	
		720x480p30	nv12	32 inst -> 17Fps 24 inst -> 31Fps	32 inst -> 16Fps 23 inst -> 30Fps	32 inst -> 39Fps	32 inst -> 30Fps	32 inst -> 16Fps 24 inst -> 30Fps	
			nv16	32 inst -> 17Fps 24 inst -> 30Fps	32 inst -> 15Fps 23 inst -> 30Fps	32 inst -> 32Fps	32 inst -> 30Fps	32 inst -> 16Fps 24 inst -> 30Fps	
			xv15	32 inst -> 17Fps 24 inst -> 32Fps	32 inst -> 16Fps 23 inst -> 30Fps	32 inst -> 32Fps	32 inst -> 30Fps	32 inst -> 16Fps 24 inst -> 30Fps	
			xv20	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 15Fps 23 inst -> 30Fps	32 inst -> 27Fps 28 inst -> 30Fps	32 inst -> 30Fps	32 inst -> 16Fps 24 inst -> 30Fps	
		720x576p25	nv12	32 inst -> 17Fps 26 inst -> 25Fps	32 inst -> 16Fps 25 inst -> 25Fps	32 inst -> 33Fps	32 inst -> 25Fps	32 inst -> 16Fps 25 inst -> 25Fps	
			nv16	32 inst -> 16Fps 25 inst -> 32Fps	32 inst -> 15Fps 25 inst -> 25Fps	32 inst -> 27Fps	32 inst -> 25Fps	32 inst -> 16Fps 25 inst -> 25Fps	
			xv15	32 inst -> 16Fps 26 inst -> 31Fps	32 inst -> 16Fps 25 inst -> 25Fps	32 inst -> 28Fps	32 inst -> 25Fps	32 inst -> 16Fps 25 inst -> 25Fps	
			xv20	32 inst -> 17Fps 25 inst -> 32Fps	32 inst -> 15Fps 25 inst -> 25Fps	32 inst -> 25Fps	32 inst -> 25Fps	29 inst -> 19Fps 25 inst -> 25Fps	**Transcode:** memory error observed after 29 instances for 3 b-frames. Working with 0 b-frames.
Encode Only: HEVC **Decode Only:** HEVC **Transcode:** HEVC -> AVC	**Encode Only:** 4 **Decode Only:** 4 **Transcode:** decode: 4 & encode: 3	640x480p30	nv12	32 inst -> 16Fps 24 inst -> 32Fps	32 inst -> 17Fps 26 inst -> 30Fps	32 inst -> 48Fps	32 inst -> 30Fps	32 inst -> 15Fps 22 inst -> 30Fps	
			nv16	32 inst -> 17Fps 24 inst -> 32Fps	32 inst -> 17Fps 25 inst -> 30Fps	32 inst -> 36Fps	32 inst -> 30Fps	32 inst -> 15Fps 22 inst -> 30Fps	
			xv15	32 inst -> 17Fps 24 inst -> 32Fps	32 inst -> 17Fps 25 inst -> 30Fps	32 inst -> 44Fps	32 inst -> 30Fps	32 inst -> 15Fps 22 inst -> 30Fps	
			xv20	32 inst -> 16Fps 24 inst -> 32Fps	32 inst -> 16Fps 25 inst -> 30Fps	32 inst -> 31Fps	32 inst -> 30Fps	32 inst -> 15Fps 22 inst -> 30Fps	
		720x480p30	nv12	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 45Fps	32 inst -> 30Fps	32 inst -> 15Fps 22 inst -> 30Fps	
			nv16	32 inst -> 17Fps 24 inst -> 31Fps	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 34Fps	32 inst -> 30Fps	32 inst -> 15Fps 22 inst -> 30Fps	
			xv15	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 41Fps	32 inst -> 30Fps	32 inst -> 15Fps 22 inst -> 30Fps	
			xv20	32 inst -> 16Fps 23 inst -> 32Fps	32 inst -> 16Fps 24 inst -> 30Fps	32 inst -> 33Fps	32 inst -> 30Fps	32 inst -> 15Fps 22 inst -> 30Fps	
		720x576p25	nv12	32 inst -> 17Fps 25 inst -> 31Fps	32 inst -> 16Fps 27 inst -> 25Fps	32 inst -> 39Fps	32 inst -> 25Fps	32 inst -> 15Fps 24 inst -> 25Fps	
			nv16	32 inst -> 16Fps 26 inst -> 25Fps	32 inst -> 16Fps 27 inst -> 25Fps	32 inst -> 31Fps	32 inst -> 25Fps	32 inst -> 15Fps 24 inst -> 25Fps	
			xv15	32 inst -> 16Fps 25 inst -> 31Fps	32 inst -> 16Fps 26 inst -> 25Fps	32 inst -> 34Fps	32 inst -> 25Fps	32 inst -> 15Fps 24 inst -> 25Fps	
			xv20	32 inst -> 16Fps 26 inst -> 25Fps	32 inst -> 16Fps 26 inst -> 25Fps	32 inst -> 28Fps	32 inst -> 25Fps	32 inst -> 15Fps 24 inst -> 25Fps	

Figure 12-3: **32-Stream Support**

Note: Encode only tests are done with file source input and expected to have < 30fps due to raw buffer copy.

DCI-4k Encode/Decode:

VCU is capable of encoding or decoding at 4096x2160p60, 422, provided VCU Core-clk frequency is set to 712 Mhz, keeping rest of AXI and MCU frequency as recommend in HW section.

Table 12-9: **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	NA	341	35%
	CBR	IBBBBP	60	4	145.6	20%
	Low Latency	I Only	0	NA	361	37%
AVC	CBR	I Only	0	NA	235	30%
	CBR	IBBBBP	60	4	77	16%
	Low Latency	I Only	0	NA	251	43%

Temporal-ID support for VCU Encoder:

- VCU Encoder will assign temporal layer ID to each frame's based on its Hierarchical layer as per AVC/HEVC standard. This enables having temporal-ID based QP and Lambda table control for encode session. This is for Pyramidal GOP only.
- **Frame delta-QP option based on temporal-ID**
  - Encoder rate control assigns QP based on rate-control algorithm type and target-rate, user can choose extra delta-QP based on its temporal-ID on top of rate-control generated QPs. This enables users to increase or decrease picture quality based on temporal Layers IDs. As we go higher in the layers it is better to have higher average QP to maintain better subjective quality.
  - Example: If Layer0 - QPs = "X", if Layer1 to be "X + 2", and Layer2 to be "X+4" then deltaQPs should be set as 2, and 4 accordingly for layer 1 and 2.
  - Not Supported:
    - RateCtrlMode=CONST_QP is not supported for delta_QP
  - To enable delta-QP option, enable below parameter in cfg file.
    - Gop.TempDQP = 1 1 1 1
- **LOAD_LDA based on temporal ID**
  - Similarly having different base frame QP depending on temporal Layer, user can choose different Lambda table for different layers. This will also improve subjective quality depending on video content.
  - To enable temporal ID, enable below parameter in cfg file and copy “Lambdas.hex” file in working directory. Refer 5.1.3 Lambda File Format

    LambdaFactors = 0.20 0.35 0.59 0.60 0.61 0.62

GStreamer

Examples of running GStreamer from the PetaLinux command line are provided below. 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 monitor connected to 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.zynqmp-display fullscreen-overlay=1
```

H.265 Decoding

Decode H.265 based input file and display it over monitor connected to 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.zynqmp-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 options below:

- Increase internal decoder buffers (internal-entropy-buffers parameter) to 9 or 10.
- Add a queue at decoder input side

The command below 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.zynqmp-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 Decoding

Decode H.265 input file using four decoder elements simultaneously, 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: tee element is used to feed same input file into 4 decoder instances, user can use separate gst-launch-1.0 application to fed different inputs.

Multistream Encoding

Encode 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 following changes are required in above pipeline:

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 may 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.zynqmp-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 for HEVC, 1MB=32x32 pixels, so user need to calculate picture height in Mbs= roundup(Height,64)/#Mb rows

Note:

- 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 12-10: **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
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 12-11: **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 12-12: **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

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

```
omx_decoder input-file.h265 -hevc -o out.yuv
```

The **omx_decoder -help** command shows all the options.

H.265 Encoding File to File

```
omx_encoder inputfile.yuv -w 352 -h 288 -r 30 -avc -o out.h264
```

The **omx_encoder -help** command shows all the options.

Note: Input YUV file should be in NV12 or NV16 format for 8-bit input sources.

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.

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 3 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 **`Ismod`** 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.

Encoding Stack

Figure 12-4 shows the Encoding software stack.

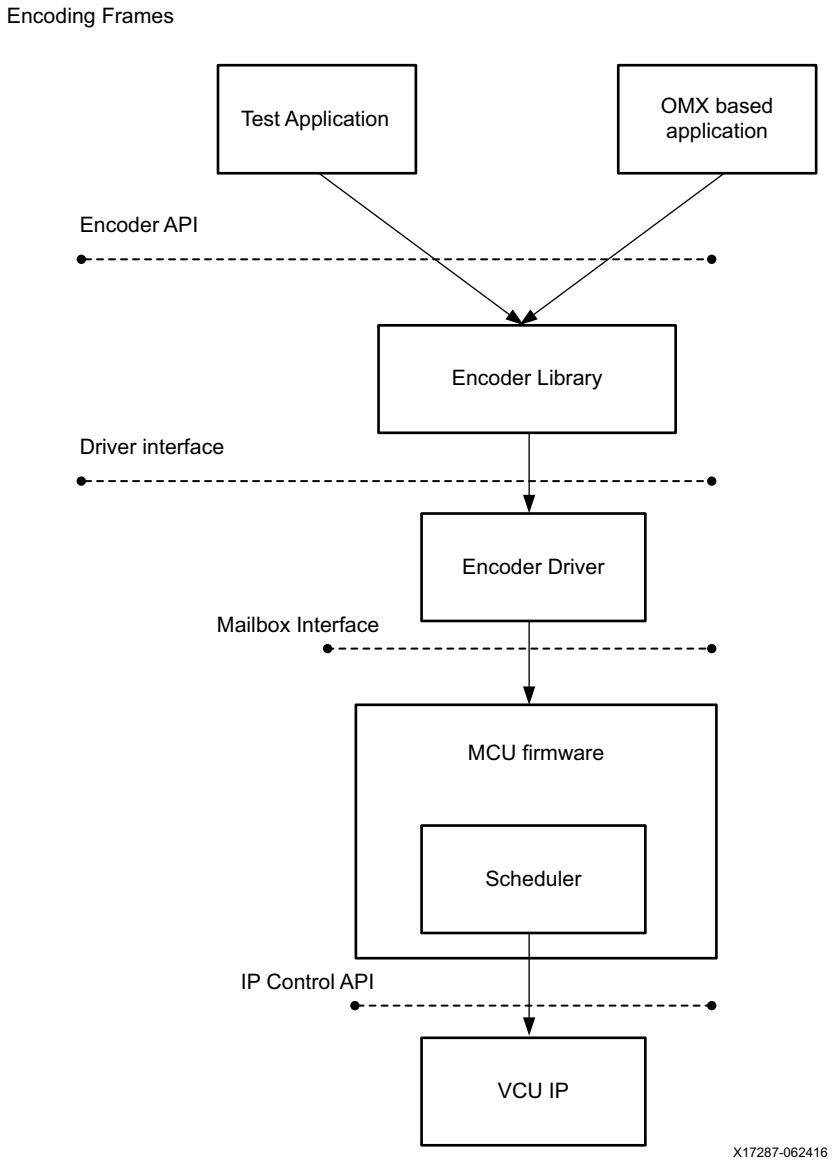

Figure 12-4: **Encoder Overview**

Application

Application refers to any OpenMAX based or standalone application that uses the VCU's underlying encoder capabilities.

Encoder Library

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

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.

Decoder Stack

Figure 12-5 shows the Decoder software stack.

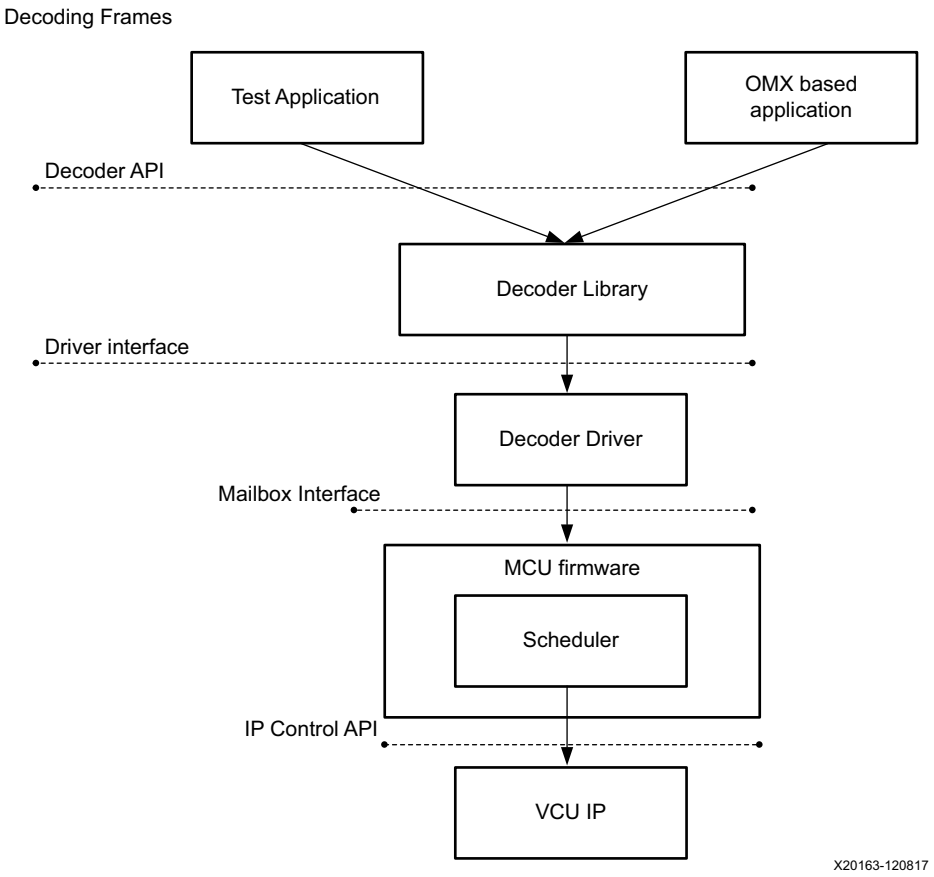

Figure 12-5: **Decoder Overview**

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 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 HW IP, handles interrupts and manages the multi-channel and multi-slice aspects of the decoding.

VCU Control Software Sample Applications

`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 12-13: **Encoder Input Parameters**

Parameter	Description and Possible Values
YUVFile	YUV file name
Width	Frame width in pixels
Height	Frame height in pixels
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.

Table 12-13: **Encoder Input Parameters *(Cont'd)***

Parameter	Description and Possible Values
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.

Output Parameters

Table 12-14: **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 12-15: **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.
BitRate	Target bit rate in Kbits/s. Not used when RateCtrlMode = CONST_QP **Default value**: 4000
MaxBitRate	Target bit rate in Kbits/s. Not used when RateCtrlMode = CONST_QP **Default value**: 4000 Notes: When RateCtrlMode is CBR, MaxBitRate shall be set to the same value as BitRate
FrameRate	Number of frames per second **Default value**: 30

Table 12-15: **Encoder Rate Control Section Parameters *(Cont'd)***

Parameter	Description and Possible Values
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
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. **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)
PBDelta	Specifies the QP difference between P frames and B frames. Allowed values: AUTO or positive value (≥ 0)
ScnChgResilience	Specifies the scene change resilience handling during encode process. Improves quality during scene changes. ENABLE/DISABLE.
MaxPictureSize	Maximum frame size allowed in Kb/s. 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

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 will be adapted within the frame, preventing a huge frame. So MinQP does need to be constrained as in frame level ratecontrol.

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 (HDR) 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 (~1sec-3sec).
- 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 1 sec.)
- 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 1.5 seconds (half of the default CPBSize of 3 seconds).

- 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 12-16: **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.
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 Allowed values: When GopCtrlMode is set to DEFAULT_GOP, Gop.NumB shall be in range 0 to 4.

Table 12-16: **Encoder GOP Section Parameters *(Cont'd)***

Parameter	Description and Possible Values
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
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 12-17: **Encoder Settings Parameters**

Parameter	Description and Possible Values
Profile	Specifies the Profile to which the bitstream conforms Allowed values: HEVC_MAIN, HEVC_MAIN10, HEVC_MAIN_STILL, HEVC_MONO, HEVC_MAIN_422_10, HEVC_MAIN_INTRA, HEVC_MAIN10_INTRA, HEVC_MAIN_422_10_INTRA AVC_BASELINE, AVC_MAIN, AVC_HIGH, AVC_HIGH10, AVC_HIGH_422, AVC_HIGH10_INTRA, AVC_HIGH_422_INTRA **Default value**: HVEC_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

Table 12-17: **Encoder Settings Parameters *(Cont'd)***

Parameter	Description and Possible Values
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
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

Table 12-17: **Encoder Settings Parameters *(Cont'd)***

Parameter	Description and Possible Values
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 Notes (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. **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**: 2
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)

Table 12-17: **Encoder Settings Parameters *(Cont'd)***

Parameter	Description and Possible Values
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	The optional Encoder buffer can be used to reduce the memory bandwidth. This option can slightly reduce the video quality. This parameter specifies the maximum size of the memory used for the Encoder buffer in KB. When the value is 0 or DISABLE, the Encoder buffer is not used. When the value is too low to handle the minimum motion estimation range, the encoder fails and displays an error message. Allowed values: DISABLE or a positive integer **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 **Default value**: ASPECT_RATIO_AUTO
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

Table 12-17: **Encoder Settings Parameters *(Cont'd)***

Parameter	Description and Possible Values
VideoMode	Specifies Video mode. Available values: INTERLACED_BOTTOM, INTERLACED_TOP, PROGRESSIVE **Default value**: PROGRESSIVE
EnableSEI	Determines the Supplemental Enhancement information with the stream. Available values: SEI_ALL, SEI_BP, SEI_NONE, SEI_PT, SEI_RP **Default value**: None
LambdaCtrlMode	Specifies the lambda values used for rate-distortion optimization. Available values: AUTO_LDA, CUSTOM_LDA, DEFAULT_LDA, DYNAMIC_LDA, LOAD_LDA **Default value**: DYNAMIC_LDA
LambdaFactors	Specifies the lambda factor for each picture: I, P and B by increasing temporal ID **Default value**: 0.20 0.352 0.59 0.59 0.59 0.59
EnableFillerData	Specifies if filler data can be added to stream or not Available values: DISABLE, ENABLE, TRUE, FALSE **Default value**: TRUE

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.

Run Parameters

Table 12-18: **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 **Defalut 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). **Defalut value**: ALL
FirstPicture	Specifies the first frame to encode. Allowed values: integer value between 0 and the number of pictures in the input YUV file. **Defalut value**: 0

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
...
```

Relative QP per LCU

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/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 couldn't 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 2)

```
1 2 3 4

01010101 <-------QP 0
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: λ(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:

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.

For 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

Quality	Description	Delta QP used
LOW_QUALITY	Lower quality than the rate-control given value	+5
NO_QUALITY	Worse possible quality for region without interest	+31 (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:

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
```

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.

Common

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 Table 12-19.

Table 12-19: **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 bit-stream can be corrupted and detecting those errors in compressed bit-stream is complex especially 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 bit-stream has scaling matrices and "scaling matrices present bit" is corrupted in the stream. When decoder reads this bit-stream, 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 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 additon, 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.

Structure

AL_Buffer

Description

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

Function

void AL_Buffer_Ref(AL_TBuffer* hBuf)

Description

Increases the reference count of hBuf by one.

See

AL_Buffer_Create_And_Allocate, AL_Buffer_WrapData

Function

void AL_Buffer_Unref(AL_TBuffer* hBuf)

Description

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

Structure

AL_TAllocator

Description

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

Structure

AL_AllocatorVtable

Description

Supplies memory management functions for the AL_TAllocator type.

Type	Field	Description
Function pointer(1)	pfnDestroy	Releases resources associated with the allocator. Note, this function relates to the allocator, not an allocated handle.
Function pointer(2)	pfnAlloc	Allocates a handle of a given size. Returns NULL if allocation fails.
Function pointer(3)	pfnFree	Releases resources associated with a handle returned by pfnAlloc. Returns true, if successful and false otherwise.
Function pointer(4)	pfnGetVirtualAddr	Translates the given handle into a virtual address.
Function pointer(5)	pfnGetPhysicalAddr	Translates the given handle into a physical address.
Function pointer(6)	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.

Notes:

1. bool (*pfnDestroy)(AL_TAllocator*)
2. AL_HANDLE (*pfnAlloc)(AL_TAllocator*, size_t)
3. bool (*pfnFree)(AL_TAllocator*, AL_HANDLE)
4. AL_VADDR (*pfnGetVirtualAddr)(AL_TAllocator*, AL_HANDLE)
5. AL_PADDR (*pfnGetPhysicalAddr)(AL_TAllocator*, AL_HANDLE)
6. AL_HANDLE (*pfnAllocNamed)(AL_TAllocator*, size_t, char const* name)

Function

void AL_Buffer_SetData(const AL_TBuffer* hBuf, uint8_t* pData)

Description

Assigns pData to the data pointer of the buffer hBuf.

Function

AL_HANDLE AL_Allocator_Alloc(AL_TAllocator* pAllocator, size_t zSize)

Return

Returns a pointer to newly allocated memory or NULL if allocation fails.

Function

AL_VADDR AL_Allocator_GetVirtualAddr(AL_TAllocator* pAllocator, AL_HANDLE hBuf)

Return

Returns hBuf.

See

LinuxDma_GetVirtualAddr, LinuxDma_Map, AL_Allocator_GetPhysicalAddr

Function

AL_PADDR AL_Allocator_GetPhysicalAddr(AL_TAllocator* pAllocator, AL_HANDLE hBuf)

Return

Returns NULL.

See

AL_Allocator_GetVirtualAddr

Enumeration

AL_EMetaType

Description

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

Function

AL_TStreamMetaData* AL_StreamMetaData_Create(uint16_t uMaxNumSection)

Description

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

Function

void AL_StreamMetaData_ClearAllSections(AL_TStreamMetaData* pMetaData)

Description

Sets the section count to zero.

See

AL_StreamMetaData_AddSection

Function

uint16_t AL_StreamMetaData_AddSection(AL_TStreamMetaData* pMetaData,
uint32_t uOffset, uint32_t uLength, uint32_t uFlags)

Description

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

Enumeration

AL_Section_Flags

Description

Section flags are used to specify attributes that apply to a region of a buffer.

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

Function

bool AL_Buffer_AddMetaData(AL_TBuffer* pBuf, AL_TMetaData* pMeta)

Description

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

Function

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

Function

bool AL_Buffer_RemoveMetaData(AL_TBuffer* pBuf, AL_TMetaData* pMeta)

Description

Removes the pMeta pointer from the metadata of pBuf.

Return

Always returns true. Thread-safe.

See

AL_Buffer_GetMetaData, AL_Buffer_AddMetaData

Function

AL_TBuffer* AL_Buffer_Create_And_Allocate(AL_TAllocator* pAllocator, size_t zSize, PFN_RefCount_CallBack pCallBack)

Description

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. Note, use of reference counting is optional. 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.

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

Function

void AL_Buffer_Destroy(AL_TBuffer* pBuf)

Description

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

Function

AL_TBuffer* AL_Buffer_WrapData(uint8_t* pData, size_t zSize,
PFN_RefCount_CallBack pCallBack)

Description

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

Function

bool AL_Allocator_Free(AL_TAllocator* pAllocator, AL_HANDLE hBuf)

Description

Frees memory associated with buffer hBuf.

See

AL_TAllocator

Function

AL_TAllocator* DmaAlloc_Create(const char* deviceFile)

Description

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

Function

bool AL_Allocator_Destroy(AL_TAllocator* pAllocator)

Description

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

Function

int AL_GetAllocSize_DecReference(AL_TDimension tDim, AL_EChromaMode eChromaMode, uint8_t uBitDepth, AL_EFbStorageMode eFrameBufferStorageMode)

[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

Description

Retrieves the size of a reference frame buffer.

Return

Returns the size in bytes needed for the reference frame buffer.

Function

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.

Function

void I0AL_To_I420(AL_TBuffer const* pSrc, AL_TBuffer* pDst)

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	●		●		●	●		●										●	●	●

From	I0AL	I2AL	I420	I422	IYUV	NV12	NV16	P010	P210	RX0A	RX2A	RXmA	T608	T60A	T628	T62A	T6m8	Y010	Y800	YV12
T60A	●		●		●	●		●										●	●	●
T628		●		●			●		●									●	●	
T62A		●		●			●		●									●	●	
T6m8			●																	
Y010										●		●								
Y800	●		●		●	●		●		●		●						●	●	●
YV12	●		●		●	●		●		●									●	

Constants

config.h

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
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

Function

AL_EChromaMode AL_GetChromaMode(TFourCC tFourCC)

Description

Implements the following mapping.

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

Macro

FOURCC(*A*)

Description

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.

Enumeration

AL_EPicFormat

Description

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

Macro

AL_GET_BITDEPTH_LUMA(PicFmt)

Return

Returns the Luma depth from PicFmt. PicFmt must be an AL_EPicFormat value.

See

AL_EPicFormat

Macro

AL_GET_BITDEPTH_CHROMA(PicFmt)

Return

Returns the Chroma depth from PicFmt. PicFmt must be an AL_EPicFormat value.

See

AL_EPicFormat

Macro

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

Macro

AL_GET_CHROMA_MODE(PicFmt)

Return

Returns the Chroma mode from PicFmt. PicFmt must be an AL_EPicFormat value.

See

AL_EPicFormat

Macro

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

Macro

AL_SET_BITDEPTH_CHROMA(PicFmt, BitDepth)

Return

Assigns BitDepth to byte 1 of PicFmt. PicFmt must be an AL_EPicFormat value.

See

AL_EPicFormat

Macro

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

Macro

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

Function

uint8_t AL_GetBitDepth(TFourCC tFourCC)

Description

Implements the following mapping.

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

Function

int GetPixelSize(uint8_t uBitDepth)

Return

Returns 1 if uBitDepth is 8 or less. Returns 2 if uBitDepth is 9 or greater.

Function

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

Function

AL_EFbStorageMode AL_GetSrcStorageMode(AL_ESrcMode eSrcMode)

Description

Implements the following mapping.

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

Function

int GetNumLinesInPitch(AL_EFbStorageMode eFrameBufferStorageMode)

Description

Implements the following mapping.

Storage Mode	Pitch Lines
AL_FB_TILE_64x4	4
AL_FB_TILE_32x4	4
AL_FB_RASTER	1

See

AL_EFbStorageMode, AL_ESrcMode

Macro

AL_IS_AVC(Prof)

Description

Returns true if Prof is AVC. Prof must be an AL_EProfile value.

See

AL_EProfile

Macro

AL_IS_HEVC(Prof)

Description

Returns true if Prof is HEVC. Prof must be an AL_EProfile value.

See

AL_EProfile

Macro

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

Function

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

Function

bool AL_IsTiled(TFourCC tFourCC)

Description

Returns true if tFourCC is a tiled format.

See

AL_Is64x4Tiled, AL_Is32x4Tiled

Function

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

Function

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

Function

bool AL_Is10bitPacked(TFourCC tFourCC)

Description

Returns true if tFourCC is a tiled format or one of: RX0A, RX2A, or RXmA.

Function

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.

Chroma Mode	sx	sy
4:2:2	2	2
4:2:0	2	1
Other	1	1

Function

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.

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

Function

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

Function

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

Function

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

Function

AL_ECodec AL_GetCodec(AL_EProfile eProf)

Return

Returns AL_CODEC_AVC or AL_CODEC_HEVC according to eProf.

Encoder Flow

Figure 12-6 shows he typical flow of control using the VCU Control Software API.

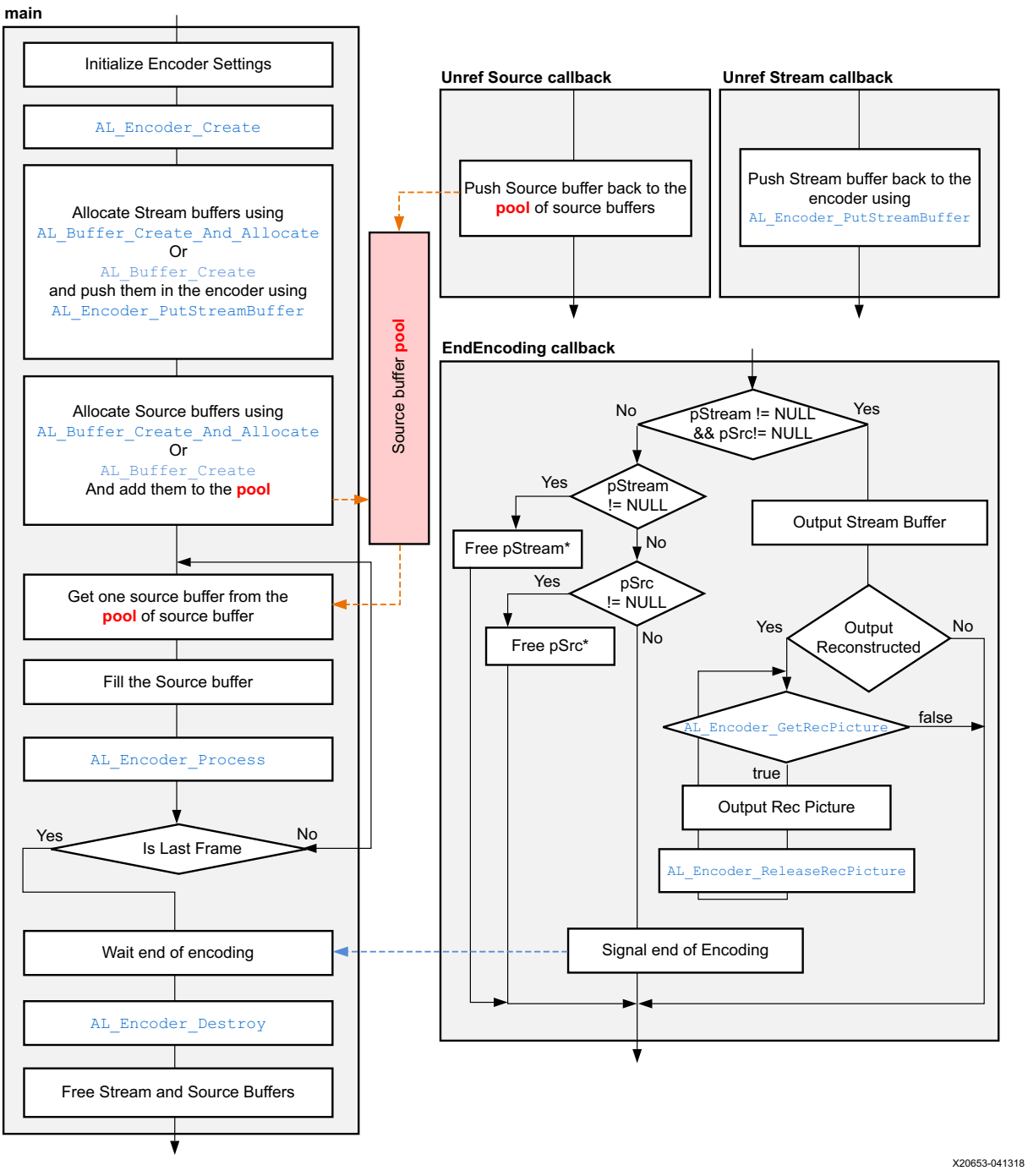

Figure 12-6: **Encoder API Workflow Example**

Encoder API

The Encoder API is defined in 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 API is documented below.

The example application **`ctrlsw_encoder`** demonstrates how to use the API.

Function

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.

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

Structure

AL_TEncSettings

Description

The Encoder settings are described in the following structure.

Type	Field	Description
AL_TEncChanParam	tChParam	
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	CacheLevel2 in bytes. Must be ≥ 64 and ≤ picture width in pixels
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 ≥ 8 and ≤ picture height 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

Enumeration

AL_EChEncOptions

Description

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

Function

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

Structure

AL_CB_EndEncoding

Description

This callback is invoked when any of the following conditions occur:

Type	Field	Description
Function pointer(1)	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

Notes:

1. void (*func)(void* pUserParam, AL_TBuffer* pStream, AL_TBuffer const* pSrc,int iLayerID)

Function

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.

Function

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

Function

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

Function

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

Function

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

Structure

AL_TBufPoolConfig

Description

Structure used to configure the AL_TBufPool.

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

Function

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

Function

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

Function

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.

Enumeration

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.

AL_EBufMode Description	Value
AL_BUF_MODE_BLOCK	0
AL_BUF_MODE_NONBLOCK	1

See

AL_Decoder_PushBuffer, AL_GetWaitMode

Function

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.

Function

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 will happen. The allowed range is [0...31]

Function

void AL_Encoder_NotifyUseLongTerm(AL_HEncoder hEnc)

Description

Notify the encoder that long-term reference frame will be used. This can improve background quality for use cases with unchanging backgrounds such as fixed-camera surveillance.

Function

void AL_Encoder_NotifyIsLongTerm(AL_HEncoder hEnc)

Description

Notify the encoder that the next reference picture is a long term reference picture.

Function

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.

Function

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 will restart 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.

Function

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.

Function

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.

Function

bool AL_Encoder_SetFrameRate(AL_HEncoder hEnc, uint16_t uFrameRate, uint16_t uClkRatio)

Description

Tells the encoder to change the encoding frame rate, calculated as follows.

fps = iFrameRate × 1,000 ÷ iClkRatio

For example, when uFrameRate = 60 and uClkRatio = 1,001, then the frame rate will be 59.94 fps.

Return

If successful, returns true. If unsuccessful, returns false. Call AL_Encoder_GetLastError to get the error code.

Function

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.

Function

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.

Function

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.

Function

void AL_Encoder_ReleaseRecPicture(AL_HEncoder hEnc, TRecPic* pRecPic)

Description

Release reconstructed buffer previously obtains through AL_Encoder_GetRecPicture.

Decoder Flow

Figure 12-7 shows an example of using the VCU Control Software API.

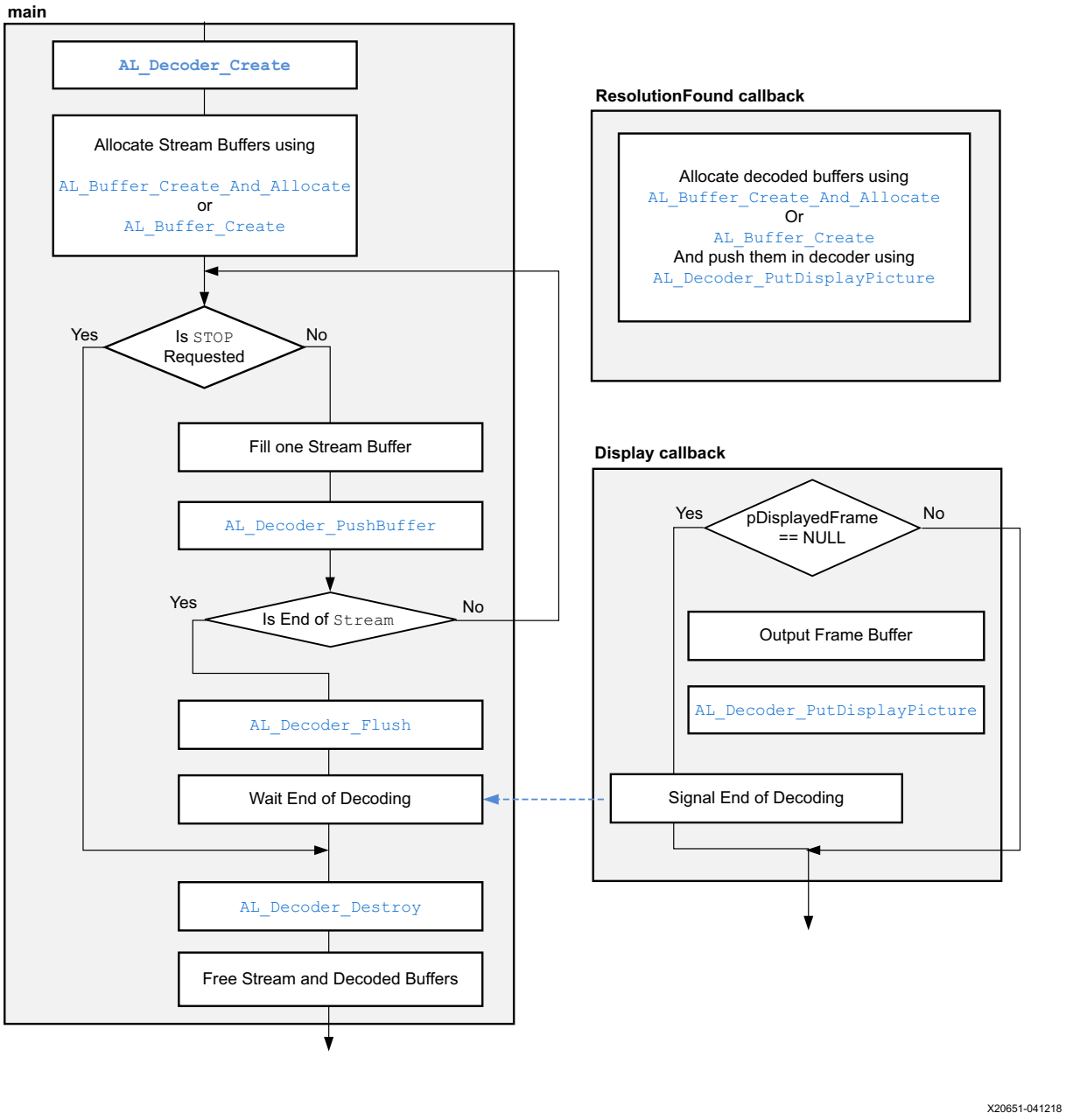

Figure 12-7: **Decoder API Workflow Example**

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.

Function

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

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

Structure

AL_TDecCallBacks

Description

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

Structure

AL_CB_EndDecoding

Description

Table 12-20:

Type	Field	Description
Function pointer(1)	func	Called when a frame is decoded. On error, *pDecodedFrame* will be NULL. Use **AL_Decoder_GetLastError** for more information.
void*	userParam	User-defined

Notes:
1. void (*func)(AL_TBuffer* pDecodedFrame, void* pUserParam)

See

AL_TDecCallBacks, AL_Decoder_GetLastError

Structure

AL_CB_Display

Description

Type	Field	Description
Function pointer(1)	func	Called when a frame is displayed. On end of stream, *pDisplayedFrame* will be NULL. Use **AL_Decoder_GetLastError** to check for error information.
void*	userParam	User-defined

Notes:
1. void (*func)(AL_TBuffer* pDisplayedFrame, AL_TInfoDecode* pInfo, void* pUserParam)

See

AL_TDecCallBacks, AL_Decoder_GetLastError, AL_Decoder_PutDisplayPicture

Structure

AL_CB_ParsedSei

Description

Resolution SEI callback definition is parsed.

Type	Field	Description
Function pointer(1)	func	Called when a SEI is parsed. Antiemulation 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

Notes:
1. void (* func)(int iPayloadType, uint8_t* pPayload, int iPayloadSize, void* pUserParam);

Structure

AL_CB_ResolutionFound

Description

Resolution change callback definition.

Type	Field	Description
Function pointer[1]	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

Notes:

1. void (*func)(int BufferNumber, int BufferSize, AL_TStreamSettings const* pSettings, AL_TCropInfo const* pCropInfo, void* pUserParam)

See

AL_TDecCallBacks, AL_Decoder_GetLastError, AL_Decoder_PutDisplayPicture

Function

AL_TIDecChannel* AL_DecChannelMcu_Create()

Description

Returns a pointer to the newly allocated AL_TIDecChannel object or NULL if allocation fails.

Structure

AL_TIDecChannel

Structure

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.

Type	Field	Description
Function pointer[1]	func	Called when a frame is decoded
void*	userParam	User-defined

Notes:
1. void (*func)(void* pUserParam, AL_TBuffer* pStream, AL_TDecPicStatus* pPicStatus)

See

AL_Default_Decoder_EndDecoding

Function

void AL_Decoder_Destroy(AL_HDecoder hDec)

Description

Releases resources associated with the Decoder hDec.

See

AL_Decoder_Create

Function

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 will result in an assertion violation; or the buffer will be cleared, but not added to the pool.

See

AL_TFrmBufPool, FRM_BUF_POOL_SIZE

Function

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.

Function

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.

Function

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

Function

void 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

Structure

AL_TDecPicStatus

Description

Associates counters, CRC, and flags with a frame.

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?

Structure

AL_TDecSettings

Description

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
bool	bIsAvc	If true, AVC decoding is selected; if false HEVC decoding is selected

Type	Field	Description
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

Function

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.

Function

int AL_SrcMetaData_GetLumaSize(AL_TSrcMetaData* pMeta);

Description

Get the size of the luma inside the picture.

Return

Returns size of the luma region.

Function

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.

Function

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 will be 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 will be 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

Function

void AL_Decoder_Flush(AL_HDecoder hDec)

Description

Requests the Decoder flush the decoding request stack when stream parsing is finished.

Function

void AL_Decoder_GetMaxBD(AL_HDecoder hDec)

Description

Returns the maximum bit depth supported for the current stream profile.

Function

int32_t RndPitch(int32_t iWidth, uint8_t uBitDepth,
AL_EFbStorageMode eFrameBufferStorageMode)

Return

Returns the pitch rounded up to the burst DMA alignment.

Function

int32_t RndHeight(int32_t iHeight)

Return

Returns the height aligned up to the nearest 64-bit boundary.

Function

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.

Function

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).

Function

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

Return

Returns maximum size in bytes of the compressed map buffer (LCU Offset and size).

Function

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.

Function

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.

Function

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.

Function

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.

Function

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).

Function

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.

Function

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.

Function

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.

Function

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.

Function

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.

Function

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.

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)

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

We can't 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 12-21: **Modified Encoder and Decoder Control Software APIs**

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.

Table 12-21: ***(Cont'd)*Modified Encoder and Decoder Control Software APIs**

2018.3 Release	2019.1 Release	Motives
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 12-22: **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 12-23: **Removed APIs**

Removed API	Motives
AL_ERR_RESOLUTION_CHANGE	The resolution change is now supported.

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).

Optimum VCU Encoder Parameters for Use-Cases

Video Streaming

- Video streaming use-case requires very stable bitrate graph for all pictures.
- It is good to 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.
- Good to 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 8 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.

Appendix A

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 Video Codec Unit, 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.

Documentation

This product guide is the main document associated with the Video Codec Unit. 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.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

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 proper 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 Video Codec Unit

AR: 66763

Technical Support

Xilinx provides technical support at the Xilinx Support web page 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 contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

There are many tools available to address Video Codec Unit 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 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) [Ref 6].

Reference Boards

Various Xilinx development boards support the Video Codec Unit. 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.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- 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

Figure A-1 shows the system level debug flow.

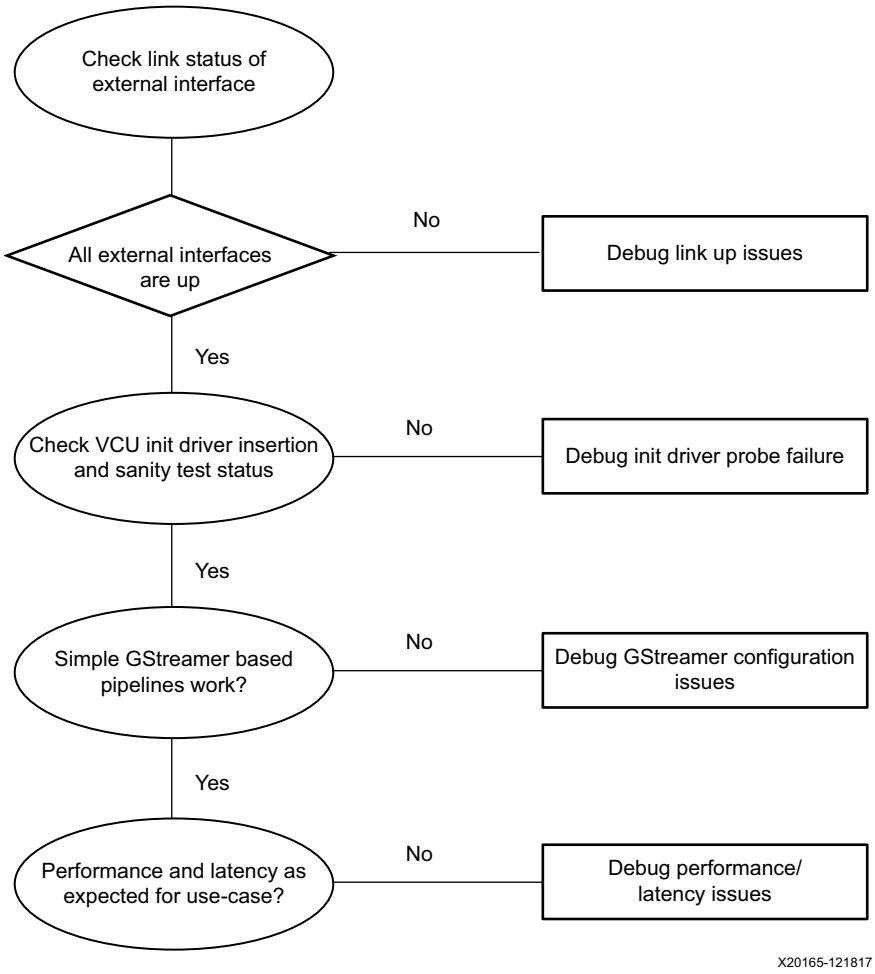

Figure A-1: **Debug Flow**

Troubleshooting

Debugging External Interfaces

If you are using capture and display interfaces based on high-speed serial IO (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.

   ```
   xmedia-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,

   ```
   modetest -M xilinx_drm_mixer
   ```

   If the PL mixer block is not used,

   ```
   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 BG24 format,

   ```
   modetest -M xilinx_drm_mixer -s <connector_id>@<crtc_id>:3840x2160-60@BG24
   ```

To display using NV12 format

```
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 a command:

   ```
   cma=xxxxM
   ```

   or while building the Linux kernel using kernel configuration property.

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 not sufficient.

   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:

```
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 hangs, 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'
xmedia-ctl -d /dev/media1 -V "\"a0080000.scaler\":0  [fmt:RBG888_1X24/3840x2160
field:none]"
```

```
xmedia-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 first. Here is an example:

   ```
   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-size=630 \
    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 since 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 utilization while the pipeline is running. A higher CPU utilization indicates there could be impact in interrupt processing time which explains 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.

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.
- If the simulation has been run, verify in simulation and/or a debug feature capture that the waveform is correct for accessing the AXI4-Lite interface.

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

Appendix B

Additional Resources and Legal Notices

Xilinx Resources

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

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 I/O and Clock Planning* (UG899)
4. *Vivado Design Suite User Guide: Getting Started* (UG910)
5. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
6. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
7. *Vivado Design Suite User Guide: Implementation* (UG904)
8. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906)
9. *LogiCORE IP AXI Interconnect Product Guide* (PG059)
10. *LogiCORE IP UltraScale Architecture-Based FPGAs Memory 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+ MPSoC Register Reference* (UG1087)
15. *Zynq UltraScale+ MPSoC Data Sheet: Overview* (DS891)
16. *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925)
17. *LogiCORE IP Zynq UltraScale+ MPSoC Processing System Product Guide* (PG201)

18. *UltraScale Architecture-Based FPGAs Memory IP* (PG150)
19. *Video Scene Change Detection: LogiCORE IP Product Guide* (PG322)
20. *PetaLinux Tools Documentation* (UG1144)
21. OpenMax Integration Layer (https://www.khronos.org/registry/OpenMAX-IL/specs/OpenMAX_IL_1_1_2_Specification.pdf)
22. GStreamer (https://gstreamer.freedesktop.org/features/)

Training Resources

1. Vivado Design Suite Hands-on Introductory Workshop
2. Vivado Design Suite Tool Flow

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/22/2019	1.2	Added API migration guide Added 2019.1 New API changes Updated Enable PL DDR section Added latency mode details with pipeline 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 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 support for VCU Encoder, SEI message insertion and extraction at Gstreamer level. 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	1.2	Updated Soft IP Registers table. Added Encoder and Decoder block diagrams. Added Encoder and Decoder buffer requirements. Added Chapter 6, Zynq UltraScale+ EV Architecture Video Codec Unit DDR4 LogiCORE IP. Added details of various latency mode. Added Optimum VCU Encoder parameters for use-cases. Updated software prerequisites. Updated VCU Out of the Box Examples. 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. Updated Encoder Input parameters. Updated Encoder Settings parameters. 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, respectfully. Replaced omx_encoder.exe and omx_decoder.exe applications with omx_encode and omx_decoder, respectfully. Increased stream support from 8 to 32.

Date	Version	Revision
06/06/2018	1.1	General updates.
04/04/2018	1.1	Updated Control Software API. 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.
12/20/2017	1.0	Reorganized content and updated API.
10/04/2017	1.0	Initial Release.

Please Read: Important Legal Notices