HDMI 1.4/2.0 Transmitter Subsystem v3.1

Product Guide

Vivado Design Suite

PG235 (v3.1) November 21, 2019

XILINX®

Table of Contents

Introduction

The HDMI 1.4/2.0 Transmitter Subsystem is a hierarchical IP that bundles a collection of HDMI™ IP sub-cores and outputs them as a single IP. It is an out-of-the-box ready-to-use HDMI 1.4/2.0 Transmitter Subsystem and avoids the need to manually assemble sub-cores to create a working HDMI system.

Features

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

Send Feedback

IP Facts

Subsystem Facts Table	
Subsystem Specifics	
Supported Device Family[1]	UltraScale+™ Families (GTHE4, GTYE4) UltraScale™ Architecture (GTHE3) Zynq-7000 SoC 7 series (GTXE2)[2] Artix®-7 (GTPE2)[2]
Supported User Interfaces	AXI4-Lite, AXI4-Stream
Resources	Performance and Resource Utilization web page
Provided with Subsystem	
Design Files	RTL
Example Design	Vivado® IP integrator and associated software application example
Test Bench	Not Provided
Constraints File	XDC
Simulation Model	Not Provided
Supported S/W Driver[3]	Standalone, Linux
Tested Design Flows[4]	
Design Entry	Vivado® Design Suite
Simulation	Not Provided
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 65911
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:
1. For a complete list of supported devices, see the Vivado IP catalog
2. Only HDMI 1.4 is supported in Artix and Kintex -1 devices.
3. Standalone driver details can be found in <Install Directory>/Vitis/<Release>/data/embeddedsw/doc/ xilinx_drivers_api_toc.htm.
   Linux OS and driver support information is available from the Linux HDMI TX Driver Page.
4. For the supported versions of third-party tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The HDMI 1.4/2.0 Transmitter Subsystem is a feature-rich soft IP incorporating all the necessary logic to properly interface with PHY layers and provide HDMI™ encoding functionality. The subsystem is a hierarchical IP that bundles a collection of HDMI TX-related IP sub-cores and outputs them as a single IP. The subsystem takes incoming video and audio streams and transfers them to an HDMI stream. The stream is then forwarded to the video PHY layer.

The subsystem can be configured at design time through a single interface in the Vivado® Integrated Design Environment (IDE) for performance and quality.

Applications

High-Definition Multimedia Interface (HDMI) is a common interface used to transport video and audio and is seen in almost all consumer video equipment such as DVD and media players, digital televisions, camcorders, mobile tablets and phones. The omnipresence of the interface has also spread to most professional equipment such as professional cameras, video switchers, converters, monitors and large displays used in video walls and public display signs.

For tested video resolutions for the subsystem see Appendix A, Verification, Compliance, and Interoperability.

Related Information
Verification, Compliance, and Interoperability

Unsupported Features

The following features are not supported in this subsystem:

- Lip sync
- CEC
- HEAC
- HDMI 2.0 dual view

- HDMI 2.0 multi stream audio
- 3D Audio

Licensing and Ordering Information

License Type

This Xilinx® module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado® Design Suite. For full access to all subsystem functionalities in simulation and in hardware, you must purchase a license for the subsystem. To generate a full license, visit the product licensing web page. Evaluation licenses and hardware timeout licenses might be available for this subsystem. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the Xilinx HDMI web page.

Information about other Xilinx® IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx® 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 an error. License checkpoints are enforced by the following tools:

- Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

Note: IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.

If a Hardware Evaluation License is being used, the core stops transmitting HDMI Stream after timeout. This timeout is based on system CPU clock. For example, if system is running at 100 MHz, the IP times out after approximately 4 hours of normal operation when Hardware Evaluation License is being used.

Product Specification

This chapter includes a description of the subsystem and details about the performance and resource utilization.

Introduction

Because the HDMI 1.4/2.0 TX Subsystem is hierarchically packaged, you can configure it by setting the parameters in the Vivado® Integrated Design Environment (IDE) interface and the subsystem creates the required hardware accordingly.

A high-level block diagram of the HDMI 1.4/2.0 TX Subsystem is shown in the following figure.

Figure 1: **Subsystem Block Diagram**
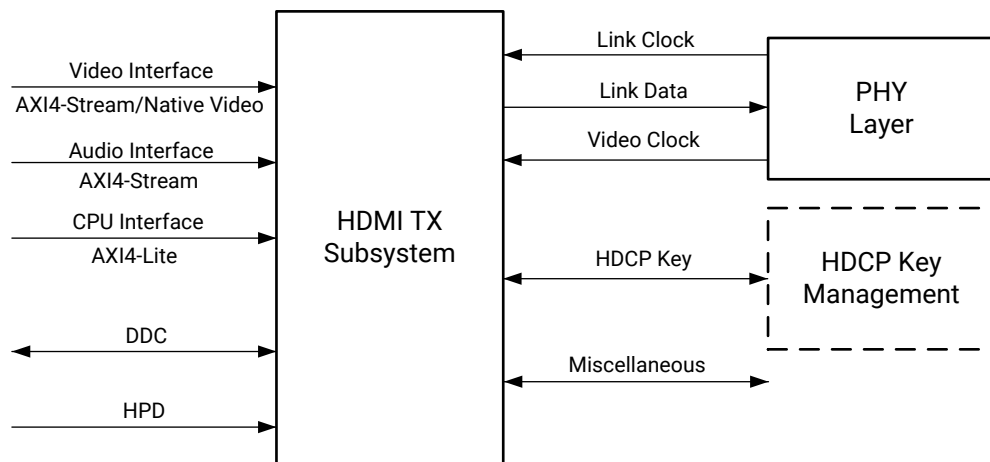
The HDMI 1.4/2.0 TX Subsystem is constructed on top of an HDMI TX core. Various supporting modules are added around the HDMI TX core with respect to your configuration. The HDMI TX core is designed to support native video interface, however many of the existing video processing IP cores are AXI4-Stream-based. It is a natural choice to add supporting modules (Video Timing Controller and AXI4-Stream to Video Out Bridge) to allow the HDMI 1.4/2.0 TX Subsystem to support AXI4-Stream-based video input. Doing this allows the HDMI 1.4/2.0 TX Subsystem to work seamlessly with other Xilinx video processing IP cores. The HDMI 1.4/2.0 TX Subsystem has a built-in capability to optionally support both HDCP 1.4 and HDCP 2.2 encryption.

The HDMI 1.4/2.0 TX Subsystem supports the following types of video interface:

- AXI4-Stream Video Interface
- Native Video Interface
- Native Video (Vectored Data Enable (DE)) Interface

The following figure shows the internal structure of the HDMI 1.4/2.0 TX Subsystem when AXI4-Stream is selected as the video interface. In this illustration, both HDCP 1.4 and HDCP 2.2 are selected and both Video over AXIS compliant NTSC/PAL Support and Video over AXIS compliant YUV420 Support are selected.

Figure 2: **HDMI TX Subsystem Internal Structure in AXI4-Stream Video Interface Mode**
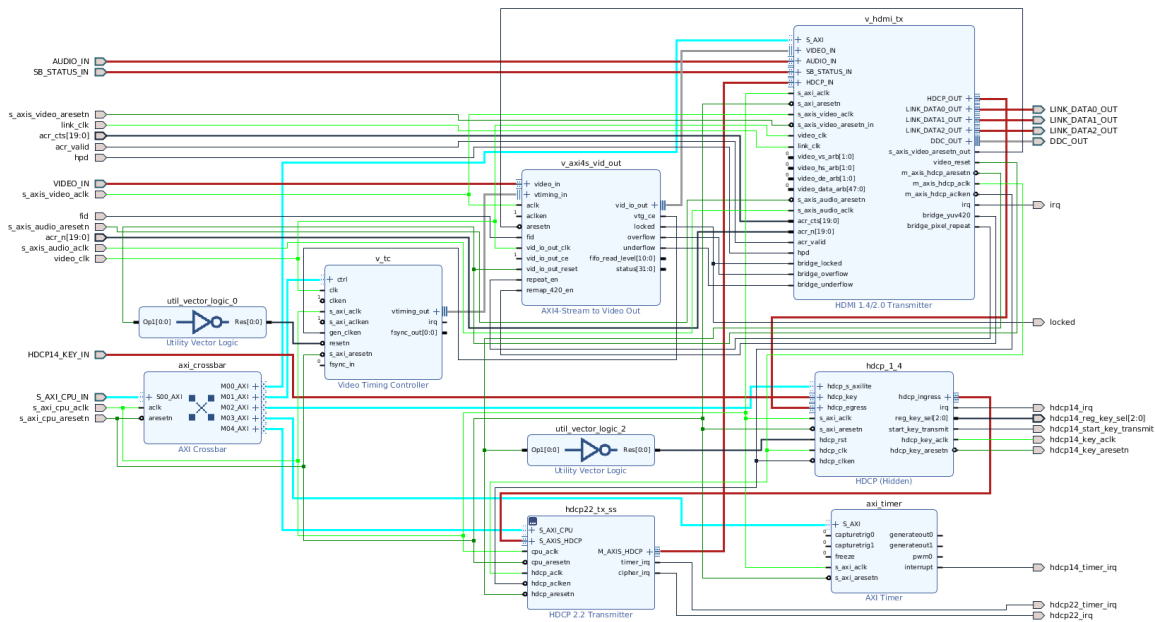
The HDMI 1.4/2.0 TX Subsystem also provides an option to support a native video interface. Some applications require support of customized resolutions, which are not divisible by the PPC setting (4 or 2). Therefore, the HDMI 1.4/2.0 TX Subsystem also provides a native video (Vectored DE) interface option to enable this application. When native video interface (with or without Vectored DE) is selected, the HDMI 1.4/2.0 TX Subsystem is constructed without the Video Timing Controller and AXI4-Stream to Video Out bridge. Therefore, the HDMI 1.4/2.0 TX Subsystem is allowed to take native video from its own video devices and convert into HDMI signals. In native video mode, the HDMI 1.4/2.0 TX Subsystem still has a built-in capability to optionally support both HDCP 1.4 and HDCP 2.2 encryption.

The following figure shows the internal structure of the HDMI 1.4/2.0 TX Subsystem when native video is selected as the video interface. In this illustration, both HDCP 1.4 and HDCP 2.2 are selected.

Figure 3: **HDMI TX Subsystem Internal Structure in Native Video Interface Mode**
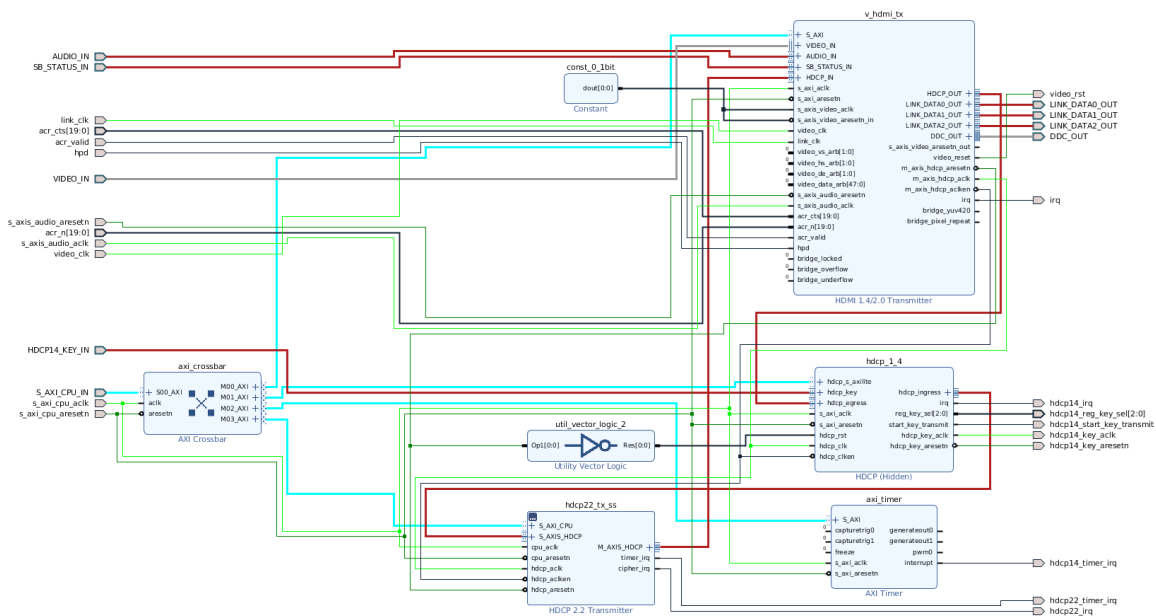
The following figure shows the internal structure of the HDMI 1.4/2.0 TX Subsystem when Native Video (Vectored DE) interface is selected as the video interface. In this illustration, both HDCP 1.4 and HDCP 2.2 are selected.

Figure 4: **HDMI TX Subsystem Internal Structure in Native Video (Vectored DE) Interface Mode**
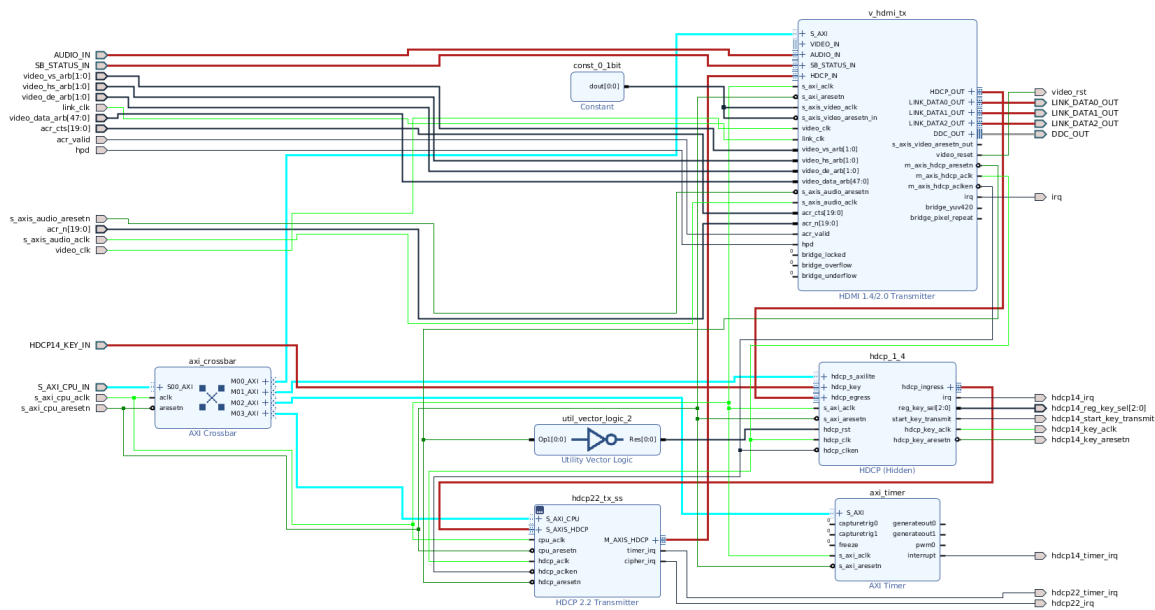
The data width of the video interface is configured in the Vivado IDE by setting the **Number of Pixels Per Clock on Video Interface** and the **Max Bits Per Component** parameters.

The audio interface is a 32-bit AXI4-Stream slave bus, which transports multiple channels of uncompressed audio data to the subsystem.

The CPU interface is an AXI4-Lite bus interface, which is connected to a MicroBlaze, Zynq-7000 SoC, or Zynq® UltraScale+™ MPSoC processor. Multiple sub-modules are used to construct the HDMI 1.4/2.0 TX Subsystem and all the sub-modules which require software access are connected through an AXI crossbar. Therefore, the MicroBlaze, Zynq®-7000 SoC, or Zynq® UltraScale+™ MPSoC processor is able to access and control each individual sub-modules inside the HDMI 1.4/2.0 TX Subsystem.

IMPORTANT! *The direct register level access to any of the sub-modules is not supported.*

The HDMI 1.4/2.0 TX Subsystem device driver has an abstract layer of API to allow you to implement certain functions. This AXI4-Lite slave interface supports single beat read and write data transfers (no burst transfers).

The subsystem converts the video stream and audio stream into an HDMI stream, based on the selected video format set by the processor core through the CPU interface. The subsystem then transmits the HDMI stream to the PHY Layer (Video PHY Controller) which converts the data into electronic signals which are then sent to an HDMI sink through an HDMI cable.

The subsystem also supports the features described in the following sections.

Audio Clock Regeneration Signals

The transmitter audio peripherals provide a dedicated Audio Clock Regeneration (ACR) input interface.

The audio clock regeneration architecture is not part of the HDMI 1.4/2.0 TX Subsystem. You must provide an audio clock to the application. This can be achieved by using an internal PLL or external clock source, depending on the audio clock requirements, audio sample frequency and jitter. When the HDMI 1.4/2.0 TX Subsystem is used in DVI mode, the ACR inputs are ignored. You can decide to leave them open or connect them to some fixed values (for example, connecting `acr_cts`, `acr_n`, and `acr_valid` to 0).

Related Information
Example Design

Display Data Channel (DDC)

The subsystem allows the end-user to build an HDMI source device, which negotiates with the targeted HDMI sink device for supported features and capabilities.. The communication between the source device(s) and the sink device is implemented through the DDC lines, which is an I2C bus included on the HDMI cable.

Status and Control Data Channel (SCDC)

The subsystem supports the following two bits in the SCDC register address offset 0x20 for TMDS configurations (Table 10-19 of the HDMI 2.0 specification, *HDMI Specifications* (http://www.hdmi.org/manufacturer/specification.aspx)).

- Bit 1: TMDS_Bit_Clock_Ratio

- Bit 0: Scrambling_Enable

  Automatically handled by the HDMI TX subsystem driver at Stream Start through the API, `XV_HdmiTxSs_StreamStart`.

Two underlining subcore API drivers are called to set the two SCDC bits, depending on the output video stream.

- `XV_HdmiTx_Scrambler(InstancePtr->HdmiTxPtr);`

  - Enables HDMI TX scrambler for HDMI 2.0 video and disable scrambler for HDMI 1.4 video stream

  - Updates the scrambler bit in the sink TMDS Configuration register

- `V_HdmiTx_ClockRatio(InstancePtr->HdmiTxPtr);`

  - sets the TMDS clock ratio bit for HDMI 2.0 video.

An API is also available for the HDMI TX subcore to show the SCDC register values of the sink (for debugging or advanced use cases).

```
void XV_HdmiTx_ShowSCDC(XV_HdmiTx *InstancePtr);
```

Hot Plug Detect

The subsystem supports the Hot Plug Detect (HPD) feature, which is a communication mechanism between the HDMI source and HDMI sink devices. For example, when an HDMI cable is inserted between the HDMI source and sink devices, the HPD signal is asserted, which triggers the subsystem to start communicating with the sink device.

> **IMPORTANT!** *The HPD from HDMI Sink is at 5V. Therefore, a level shifter is required before connecting it to the corresponding FPGA input pin in the HDMI 1.4/2.0 TX Subsystem design. For example, if the HPD pin of the HDMI 1.4/2.0 TX Subsystem is at 3.3V bank, a level shifter is required to translate HPD signal from 5V to 3.3V.*

AUX Packets

For HDMI, all data island packets consist of a 4-byte packet header and a 32 bytes of packet contents. The packet header, represented in the following figure, contains 24 data bits (3 bytes) and 8 bits (1 byte) of BCH ECC parity.

Figure 5: **Packet Header**

Byte\Bit #	7	6	5	4	3	2	1	0
HB0	Packet Type							
HB1	packet-specific data							
HB2	packet-specific data							
ECC	ECC							

The packet body, represented in the following figure, is made from four subpackets; each subpacket includes 56 bits (7 bytes) of data and 8 bits (1 byte) of BCH ECC parity.

Figure 6: **Packet Body**

Subpacket#	Byte\Bit #	7	6	5	4	3	2	1	0
Subpacket0 (Checksum + 6 data bytes + ECC)	PB0	Checksum							
	PB1	Data Byte 1							
	PB2...PB5	Data Byte 2 - Data Byte 5							
	PB6	Data Byte 6							
	ECC1	ECC							
Subpacket1 (7 Data Bytes + ECC)	PB7	Data Byte 7							
	PB8...PB12	Data Byte 8 - Data Byte 12							
	PB13	Data Byte 13							
	ECC2	ECC							
Subpacket2 (7 Data Bytes + ECC)	PB14	Data Byte 14							
	PB15...PB19	Data Byte 15 - Data Byte 19							
	PB20	Data Byte 20							
	ECC3	ECC							
Subpacket3 (7 Data Bytes + ECC)	PB21	Data Byte 21							
	PB22...PB26	Data Byte 22 - Data Byte 26							
	PB27	Data Byte 27							
	ECC3	ECC							

Note:

- ECC is calculated in the HDMI 1.4/2.0 TX Subsystem. Therefore, you must construct HB0...HB2, and PB0, PB1...PB26, PB27 according to HDMI specs in the software.

- When calculating the checksum value (PB0), the ECC values are ignored.

- Refer to section 5.2.3.4 and 5.2.3.5 of the HDMI 1.4 specification *HDMI Specifications* (http://www.hdmi.org/manufacturer/specification.aspx) for more information on the Aux packet structure.

In the following table, the packet types highlighted in blue are handled by hardware and the packet types highlighted in yellow are handled by software.

Table 1: **Hardware and Software Packet Types**

Packet Type Value	Packet Type
0x00	Null
0x01	Audio Clock Regeneration (N/CTS)
0x02	Audio Sample (L-PCM and IEC 61937 compressed formats)
0x03	General Control
0x04	ACP Packet
0x05	ISRC1 Packet
0x06	ISRC2 Packet
0x07	One Bit Audio Sample Packet

Send Feedback

Table 1: **Hardware and Software Packet Types** *(cont'd)*

Packet Type Value	Packet Type
0x08	DST Audio Packet
0x09	High Bitrate (HBR) Audio Stream Packet (IEC 61937)
0x0A	Gamut Metadata Packet
0x80+InfoFrame Type	InfoFrame Packet
0x00	Reserved
0x01	Vendor-Specific
0x02	Auxiliary Video Information (AVI)
0x03	Source Product Descriptor
0x04	Audio
0x05	MPEG Source
0x06	NTSC VBI
0x07	Dynamic Range and Mastering (HDR)

In the HDMI 1.4/2.0 TX Subsystem driver, a common API is defined for sending auxiliary packets.

```
void XV_HdmiTxSs_SendGenericAuxInfoframe(XV_HdmiTxSs *InstancePtr, void
*Aux)
```

where,

- InstancePtr is a pointer to the HDMI 1.4/2.0 TX Subsystem instance.

- Aux is a 36 byte array contains the complete AUX packet.

The aux packet data structure defined in the driver is:

```
typedef union {
    u32 Data;
    u8 Byte[4];
} XHdmiC_AuxHeader;
typedef union {
    u32 Data[8];
    u8 Byte[32];
} XHdmiC_AuxData;
typedef struct {
  XHdmiC_AuxHeader Header;
  XHdmiC_AuxData Data;
} XHdmiC_Aux;
```

For packet types not highlighted in the previous table (HDR, HLG), you can construct packets with respect to the specification, and send them using the same API call. In this case, you must calculate the CRC whenever needed and place the CRC at the right location so that the HDMI Sink can decode the packet.

InfoFrames

As shown in the previous table, the software handles the Vendor Specific InfoFrame (VSIF), the Auxiliary Video Information (AVI) InfoFrame, and the Audio InfoFrame. The three InfoFrame types are added as part of the HDMI 1.4/2.0 TX Subsystem data structure, each having its own well-defined structure.

AVI InfoFrame

Use the following code example for AVI InfoFrames:

```
typedef struct XHDMIC_AVI_InfoFrame {
  unsigned char Version;
  XHdmiC_Colorspace ColorSpace;
  u8 ActiveFormatDataPresent;
  XHdmiC_BarInfo BarInfo;
  XHdmiC_ScanInfo ScanInfo;
  XHdmiC_Colorimetry Colorimetry;
  XHdmiC_PicAspectRatio PicAspectRatio;
  XHdmiC_ActiveAspectRatio ActiveAspectRatio;
  unsigned char Itc;
  XHdmiC_ExtendedColorimetry ExtendedColorimetry;
  XHdmiC_RGBQuantizationRange QuantizationRange;
  XHdmiC_NonUniformPictureScaling NonUniformPictureScaling;
  unsigned char VIC;
  XHdmiC_YccQuantizationRange YccQuantizationRange;
  XHdmiC_ContentType ContentType;
  XHdmiC_PixelRepetitionFactor PixelRepetition;
  u16 TopBar;
  u16 BottomBar;
  u16 LeftBar;
  u16 RightBar;
} XHdmiC_AVI_InfoFrame;
```

Audio InfoFrame

Use the following code example for Audio InfoFrames:

```
typedef struct XHdmiC_Audio_InfoFrame {
  unsigned char Version;
  XHdmiC_AudioChannelCount ChannelCount;
  XHdmiC_AudioCodingType CodingType;
  XHdmiC_SampleSize SampleSize;
  XHdmiC_SamplingFrequency SampleFrequency;
  u8 CodingTypeExt;
  u8 ChannelAllocation;
  XHdmiC_LFEPlaybackLevel LFE_Playback_Level;
  XHdmiC_LevelShiftValue LevelShiftVal;
  unsigned char Downmix_Inhibit;
} XHdmiC_AudioInfoFrame;
```

Vendor Specific InfoFrame (VSIF)

Use the following code example for VSIF:

```
typedef struct {
    u8 Version;
    u32 IEEE_ID;
    XHdmiC_VSIF_Video_Format Format;
    union {
        u8 HDMI_VIC;
        XHdmiC_3D_Info Info_3D;
    };
} XHdmiC_VSIF;
```

The following API functions are defined in the HDMI Common driver to facilitate the construction of the InfoFrame packets. You must set the corresponding fields in the data structure and use the call functions below based on InfoFrame type. The API function constructs the packet and simultaneously calculates a CRC for both packet header and packet body.

```
XHdmiC_Aux XV_HdmiC_AVIIF_GeneratePacket(XHdmiC_AVI_InfoFrame
*infoFramePtr);
XHdmiC_Aux XV_HdmiC_AudioIF_GeneratePacket(XHdmiC_AudioInfoFrame
*AudioInfoFrame);
XHdmiC_Aux XV_HdmiC_VSIF_GeneratePacket(XHdmiC_VSIF *VSIFPtr);
```

General Control Packet

The General Control Packet (GCP) is primarily handled by the hardware but is also passed to the software so that the information can be used at the system application level.

The General Control Packet is automatically generated by the HDMI 1.4/2.0 TX Subsystem according to the video stream and sent out within 384 clocks after vsync. Four API functions are defined to set or clear the AVMUTE field as per their system application requirement.

```
void XV_HdmiTxSs_SetGcpAvmuteBit(XV_HdmiTxSs *InstancePtr);
void XV_HdmiTxSs_ClearGcpAvmuteBit(XV_HdmiTxSs *InstancePtr);
void XV_HdmiTxSs_SetGcpClearAvmuteBit(XV_HdmiTxSs *InstancePtr);
void XV_HdmiTxSs_ClearGcpClearAvmuteBit(XV_HdmiTxSs *InstancePtr);
```

HDCP

As part of the HDMI 1.4/2.0 TX Subsystem, the Xilinx LogiCORE IP High-bandwidth Digital Content Protection (HDCP) transmitters are designed to transmit audiovisual content securely between two devices that are HDCP capable. In this HDMI 1.4/2.0 TX Subsystem, both HDCP 1.4 and HDCP 2.2 transmitter IP cores are included, and can be enabled by the HDCP option in the Vivado IDE. However because HDCP 2.2 supersedes the HDCP 1.4 protocol and does not provide backwards compatibility, you need to decide and choose targeted content protection schemes from the Vivado IDE. Four different options are available to choose from:

• No HDCP

- HDCP 1.4 only

- HDCP 2.2 only

- HDCP 1.4 and HDCP 2.2

As a guideline, HDCP 2.2 is used to encrypt content at Ultra-High Definition (UHD) while HDCP 1.4 is the legacy content protection scheme used at lower resolutions.

The following figure shows a configuration of the HDMI TX where both HDCP 1.4 and 2.2 are enabled. With both HDCP protocols enabled, the HDMI Subsystem configures itself in the cascade topology where the HDCP 1.4 and HDCP 2.2 are connected back-to-back. The HDCP Egress interface of the HDMI transmitter sends unencrypted audiovisual data, which is encrypted by the active HDCP block and sent back into the HDMI transmitter over the HDCP Ingress interface for transmission over the link. The HDMI TX Subsystem ensures that only one of the HDCP protocols is active at any given time and the other is passive by calling the relevant HDMI 1.4/2.0 TX Subsystem API functions.

Figure 7: **HDCP 1.4 and HDCP 2.2 over HDMI Transmitter**
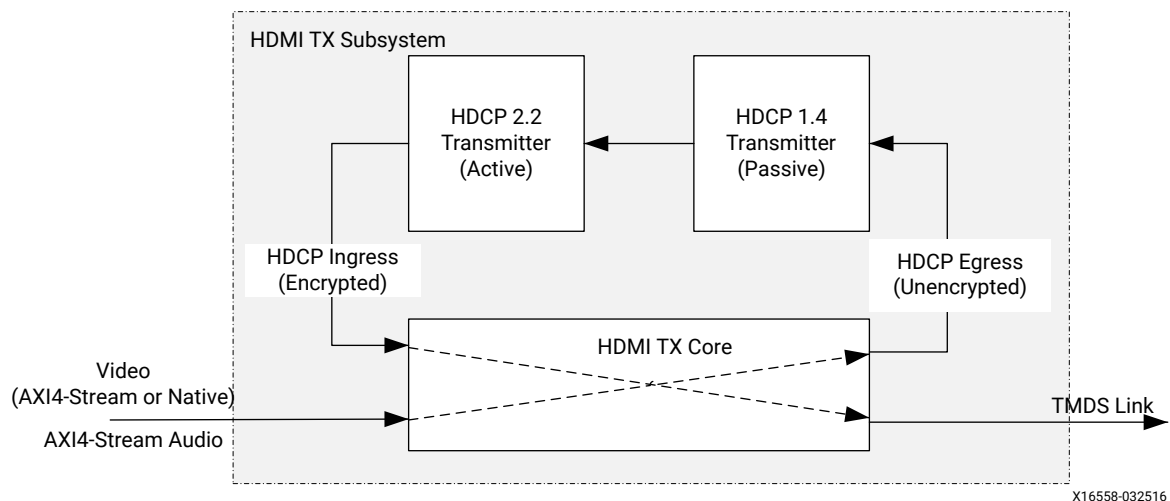
For more details on HDCP, see the *HDCP 1.x Product Guide* (PG224) and the *HDCP 2.2 LogiCORE IP Product Guide* (PG249).

Standards

The HDMI 1.4/2.0 TX Subsystem is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. See the *Vivado Design Suite: AXI Reference Guide* (UG1037) and the *AXI4-Stream Video IP and System Design Guide* (UG934) for additional information. Also, see the HDMI specifications (*HDMI Specifications* (http://www.hdmi.org/manufacturer/specification.aspx)).

The HDMI 1.4/2.0 TX Subsystem is compliant with the HDMI 1.4b and HDMI 2.0 specification.

The Xilinx HDCP 1.4 is designed to be compatible with High-bandwidth Digital Content Protection system Revision 1.4.

The Xilinx HDCP 2.2 is compliant with the HDCP 2.2 specification entitled High-bandwidth Digital Content Protection, Mapping HDCP to HDMI, Revision 2.2, issued by Digital Content Protection (DCP) LLC.

Performance and Resource Utilization

For full details about performance and resource utilization, visit the Performance and Resource Utilization web page.

Maximum Frequencies

Refer to the following documents for information on DC and AC switching characteristics. The frequency ranges specified in these documents must be adhered to for proper transceiver and core operation.

- *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892)

- *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893)

- *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS182)

- *Virtex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS183)

- *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS181)

- *Zynq-7000 SoC (Z-7007S, Z-7012S, Z-7014S, Z-7010, Z-7015, and Z-7020) Data Sheet: DC and AC Switching Characteristics* (DS187)

- *Zynq-7000 SoC (Z-7030, Z-7035, Z-7045, and Z-7100) Data Sheet: DC and AC Switching Characteristics* (DS191)

- *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922)

- *Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics* (DS923)

- *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925)

- *Zynq UltraScale+ RFSoC Data Sheet: DC and AC Switching Characteristics* (DS926)

Port Descriptions

The HDMI 1.4/2.0 TX Subsystem supports two types of video input stream interfaces, which eventually is mapped to the HDMI 1.4/2.0 TX Subsystem VIDEO_IN interface.

- AXI4-Stream Video interface

- Native Video Interface

- Native Video Vectored DE

AXI4-Stream Pinouts

The following four figures show the HDMI 1.4/2.0 TX Subsystem ports when AXI4-Stream is selected as the video interface. The VIDEO_IN port is expanded in the figure to show the detail of the AXI4-Stream video bus signals. The subsystem has the following default interfaces:

- AXI4-Lite CPU control interface (S_AXI_CPU_IN)

- AXI4-Stream Video Interface (VIDEO_IN)

- AXI4-Stream Audio Interface (AUDIO_IN)

Note: In the following diagrams, for all AXI4-Stream interfaces `video_data_width = (int((3*BPC*PPC+7)/8))*8` and for all native interfaces `video_data_width = 3*BPC*PPC`.

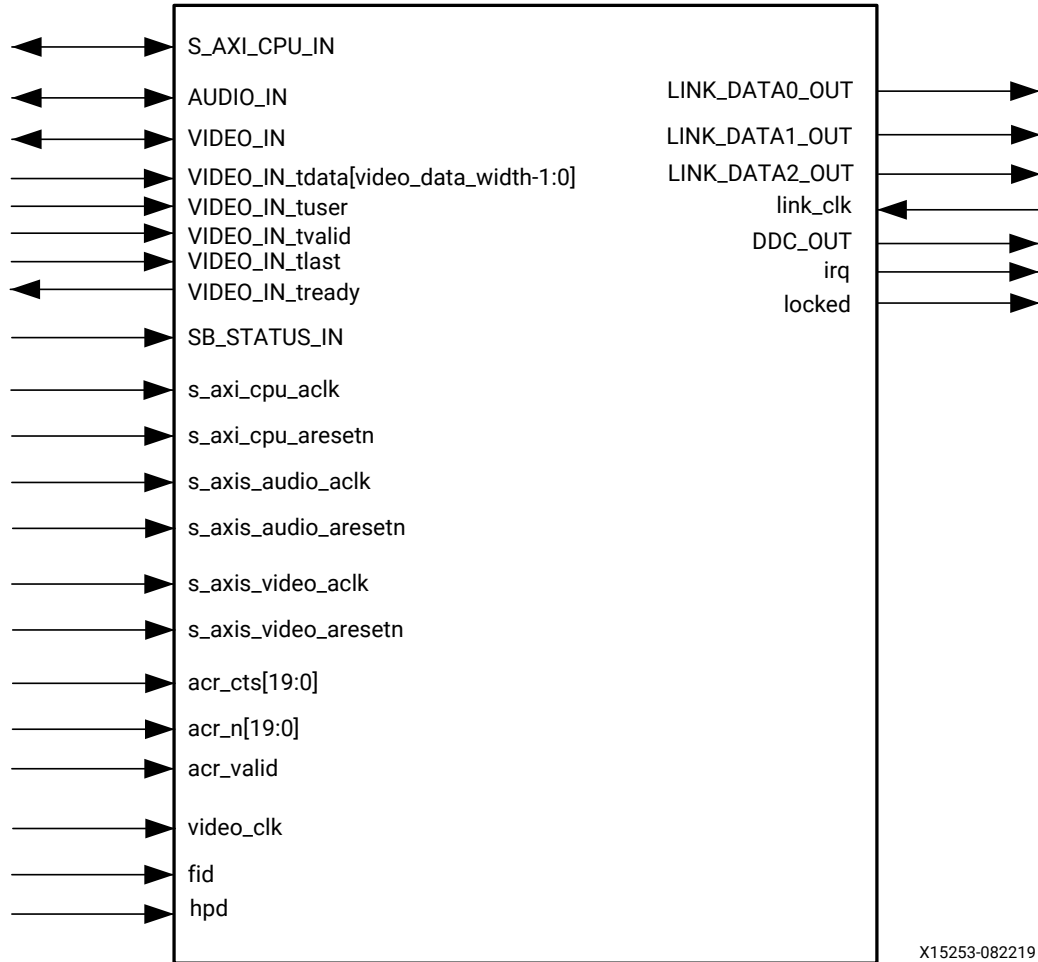*Figure 8:* **HDMI TX Subsystem Pinout – AXI4-Stream Video Interface (No HDCP)**

```
        ┌──────────────────────────────────────────────────────┐
   ◄────┤ S_AXI_CPU_IN                                          │
        │                                                       │
   ◄────┤ AUDIO_IN                           LINK_DATA0_OUT ├────►
        │                                                       │
   ◄────┤ VIDEO_IN                           LINK_DATA1_OUT ├────►
        │                                                       │
   ─────►│ VIDEO_IN_tdata[video_data_width-1:0] LINK_DATA2_OUT ├────►
   ─────►│ VIDEO_IN_tuser                            link_clk │◄────
   ─────►│ VIDEO_IN_tvalid                                       │
   ─────►│ VIDEO_IN_tlast                          DDC_OUT ├────►
   ◄────┤ VIDEO_IN_tready                              irq ├────►
        │                                            locked ├────►
   ─────►│ SB_STATUS_IN                                         │
        │                                                       │
   ─────►│ s_axi_cpu_aclk                                       │
        │                                                       │
   ─────►│ s_axi_cpu_aresetn                                    │
        │                                                       │
   ─────►│ s_axis_audio_aclk                                    │
        │                                                       │
   ─────►│ s_axis_audio_aresetn                                 │
        │                                                       │
   ─────►│ s_axis_video_aclk                                    │
        │                                                       │
   ─────►│ s_axis_video_aresetn                                 │
        │                                                       │
   ─────►│ acr_cts[19:0]                                        │
        │                                                       │
   ─────►│ acr_n[19:0]                                          │
        │                                                       │
   ─────►│ acr_valid                                            │
        │                                                       │
   ─────►│ video_clk                                            │
        │                                                       │
   ─────►│ fid                                                  │
   ─────►│ hpd                                   X15253-082219  │
        └──────────────────────────────────────────────────────┘
```

Send Feedback

Figure 9: **HDMI TX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 1.4 Only)**

```
                    ┌──────────────────────────────────┐
   S_AXI_CPU_IN ◄──►│                                  │
                    │                                  │
   AUDIO_IN ◄──────►│                   LINK_DATA0_OUT │──►
                    │                                  │
   VIDEO_IN ◄──────►│                   LINK_DATA1_OUT │──►
   VIDEO_IN_tdata[video_data_width-1:0] LINK_DATA2_OUT │──►
   VIDEO_IN_tuser ─►│                          link_clk│◄──
   VIDEO_IN_tvalid ►│                                  │
   VIDEO_IN_tlast ─►│                           DDC_OUT│──►
   VIDEO_IN_tready ◄│                               irq│──►
                    │                            locked│──►
   SB_STATUS_IN ───►│                                  │
                    │                   hdcp14_key_aclk │──►
   s_axi_cpu_aclk ─►│                hdcp14_key_aresetn │──►
                    │           hdcp14_start_key_transmit│──►
   s_axi_cpu_aresetn│             hdcp14_reg_key_sel[2:0]│──►
                    │                                  │
   s_axis_audio_aclk│                       hdcp14_irq │──►
                    │                 hdcp14_timer_irq │──►
   s_axis_audio_aresetn                               │
                    │                    HDCP14_KEY_IN │◄──
   s_axis_video_aclk│                                  │
                    │                                  │
   s_axis_video_aresetn                               │
                    │                                  │
   acr_cts[19:0] ──►│                                  │
                    │                                  │
   acr_n[19:0] ────►│                                  │
   acr_valid ──────►│                                  │
                    │                                  │
   video_clk ──────►│                                  │
                    │                                  │
   fid ────────────►│                                  │
   hpd ────────────►│                                  │
                    └──────────────────────────────────┘
                                              X16559-103019
```

Send Feedback

Figure 10: **HDMI TX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 2.2 Only)**


```
                    S_AXI_CPU_IN                              LINK_DATA0_OUT
                    AUDIO_IN                                  LINK_DATA1_OUT
                    VIDEO_IN                                  LINK_DATA2_OUT
                    VIDEO_IN_tdata[video_data_width-1:0]          link_clk
                    VIDEO_IN_tuser                               DDC_OUT
                    VIDEO_IN_tvalid                                  irq
                    VIDEO_IN_tlast                                 locked
                    VIDEO_IN_tready                             hdcp22_irq
                    SB_STATUS_IN                          hdcp22_timer_irq
                    s_axi_cpu_aclk
                    s_axi_cpu_aresetn
                    s_axis_audio_aclk
                    s_axis_audio_aresetn
                    s_axis_video_aclk
                    s_axis_video_aresetn
                    acr_cts[19:0]
                    acr_n[19:0]
                    acr_valid
                    video_clk
                    fid
                    hpd
                                                         X16560-102819
```
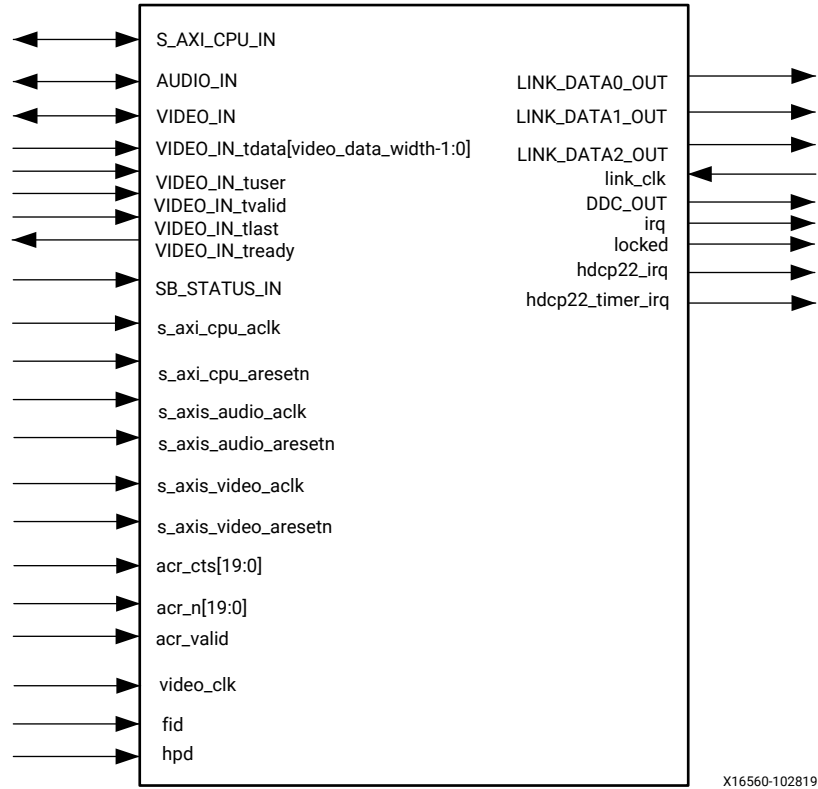
Figure 11: **HDMI TX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 1.4 and HDCP 2.2)**


```
                        ┌─────────────────────────────────────────┐
  ◄──────────────────   │ S_AXI_CPU_IN                            │
  ◄──────────────────   │ AUDIO_IN                 LINK_DATA0_OUT  │ ──────────►
  ◄──────────────────   │ VIDEO_IN                 LINK_DATA1_OUT  │ ──────────►
  ──────────────────►   │ VIDEO_IN_tdata[video_data_width-1:0]     │
                        │                          LINK_DATA2_OUT  │ ──────────►
  ──────────────────►   │ VIDEO_IN_tuser                   link_clk│ ◄──────────
  ──────────────────►   │ VIDEO_IN_tvalid                 DDC_OUT  │ ──────────►
  ──────────────────►   │ VIDEO_IN_tlast                      irq  │ ──────────►
  ◄──────────────────   │ VIDEO_IN_tready                  locked  │ ──────────►
  ──────────────────►   │ SB_STATUS_IN               hdcp14_key_aclk│ ──────────►
  ──────────────────►   │ s_axi_cpu_aclk          hdcp14_key_aresetn│ ──────────►
  ──────────────────►   │ s_axi_cpu_aresetn   hdcp14_start_key_transmit│ ─────►
  ──────────────────►   │ s_axis_audio_aclk    hdcp14_reg_key_sel[2:0]│ ──────►
  ──────────────────►   │ s_axis_audio_aresetn         hdcp14_irq  │ ──────────►
  ──────────────────►   │ s_axis_video_aclk       hdcp14_timer_irq │ ──────────►
  ──────────────────►   │ s_axis_video_aresetn       HDCP14_KEY_IN │ ◄──────────
  ──────────────────►   │ acr_cts[19:0]                hdcp22_irq  │ ──────────►
  ──────────────────►   │ acr_n[19:0]             hdcp22_timer_irq │ ──────────►
  ──────────────────►   │ acr_valid                               │
  ──────────────────►   │ video_clk                               │
  ──────────────────►   │ fid                                     │
  ──────────────────►   │ hpd                                     │
                        └─────────────────────────────────────────┘
                                                     X16561-102819
```

Native Video Pinouts

The following four figures show the HDMI 1.4/2.0 TX Subsystemports when native video is selected as the video interface. The VIDEO_IN port is expanded in the figure to show the detailed native video bus signals.

Note: In the following diagrams, for all native interfaces `video_data_width = 3*BPC*PPC`.

Send Feedback

Figure 12: **HDMI TX Subsystem Pinout – Native Video Interface (No HDCP)**

X16562-103119

Send Feedback
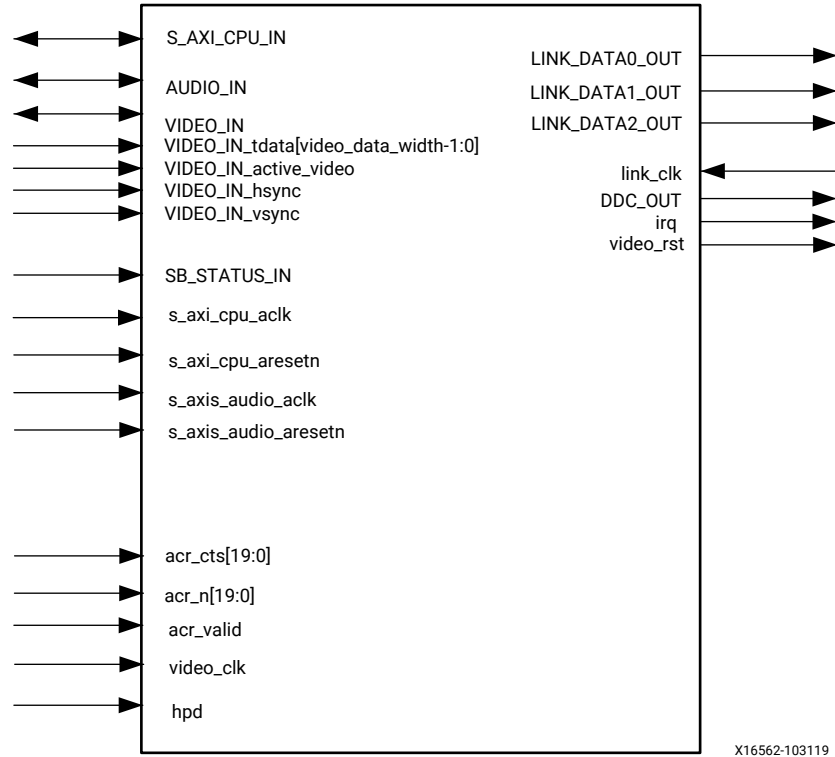
Figure 13: **HDMI TX Subsystem Pinout – Native Video Interface (HDCP 1.4 Only)**

S_AXI_CPU_IN

AUDIO_IN

VIDEO_IN

VIDEO_IN_tdata[video_data_width-1:0]
VIDEO_IN_active_video
VIDEO_IN_hsync
VIDEO_IN_vsync

SB_STATUS_IN

s_axi_cpu_aclk

s_axi_cpu_aresetn

s_axis_audio_aclk

s_axis_audio_aresetn

acr_cts[19:0]

acr_n[19:0]

acr_valid

video_clk

hpd

LINK_DATA0_OUT
LINK_DATA1_OUT
LINK_DATA2_OUT

link_clk

DDC_OUT
irq
video_rst

hdcp14_key_aclk
hdcp14_key_aresetn
hdcp14_start_key_transmit
hdcp14_reg_key_sel[2:0]

hdcp14_irq
hdcp14_timer_irq

HDCP14_KEY_IN

X16563-103119

Figure 14: **HDMI TX Subsystem Pinout – Native Video Interface (HDCP 2.2 Only)**

X16564-103119
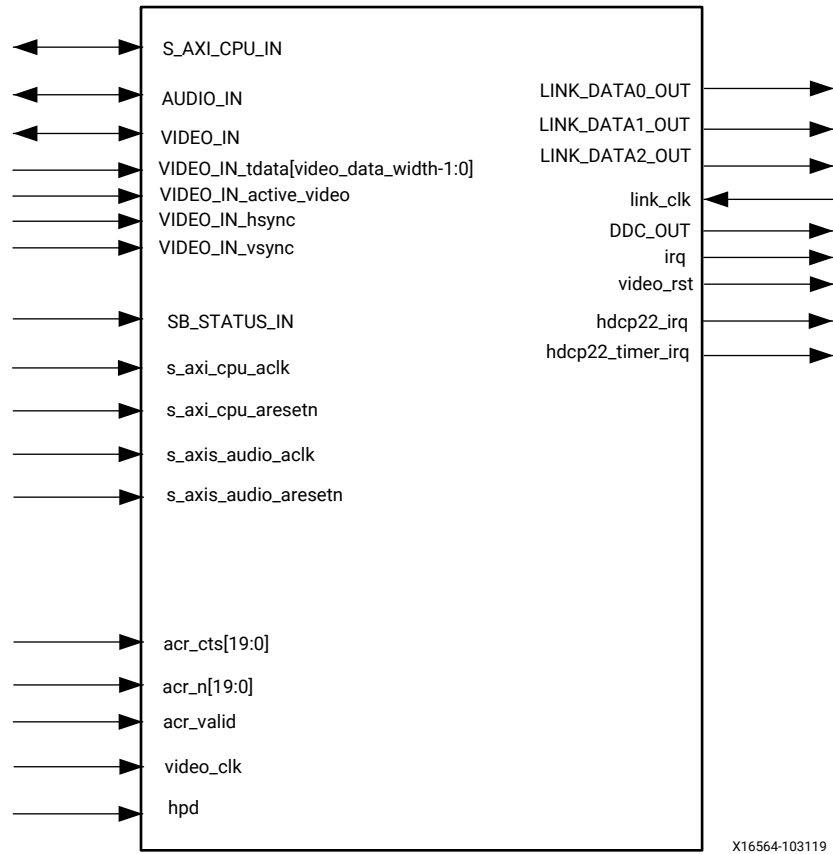
Figure 15: **HDMI TX Subsystem Pinout – Native Video Interface (HDCP 1.4 and HDCP 2.2)**


```
S_AXI_CPU_IN                                LINK_DATA0_OUT

AUDIO_IN                                    LINK_DATA1_OUT

VIDEO_IN                                    LINK_DATA2_OUT
VIDEO_IN_tdata[video_data_width-1:0]            link_clk
VIDEO_IN_active_video
VIDEO_IN_hsync                                 DDC_OUT
VIDEO_IN_vsync                                    irq
                                              video_rst

SB_STATUS_IN                               hdcp14_key_aclk
                                        hdcp14_key_aresetn
s_axi_cpu_aclk
                                     hdcp14_start_key_transmit
s_axi_cpu_aresetn                     hdcp14_reg_key_sel[2:0]

s_axis_audio_aclk                             hdcp14_irq
                                         hdcp14_timer_irq
s_axis_audio_aresetn                      HDCP14_KEY_IN
                                             hdcp22_irq
                                        hdcp22_timer_irq

acr_cts[19:0]

acr_n[19:0]

acr_valid

video_clk

hpd
```

X23115-103019

Native Video (Vectored DE)

Note: In the following diagrams, for all native interfaces `video_data_width = 3*BPC*PPC`.

Send Feedback

Figure 16: **HDMI TX Subsystem Pinout – Native Video (Vectored DE) (No HDCP)**


```
                    ┌──────────────────────────────────────┐
  ──────────────►   │  S_AXI_CPU_IN          LINK_DATA0_OUT │  ──────────►
                    │                        LINK_DATA1_OUT │  ──────────►
  ──────────────►   │  AUDIO_IN              LINK_DATA2_OUT │  ──────────►
                    │                                       │
  ──────────────►   │  video_data_arb[video_data_width-1:0] │
  ──────────────►   │  video_hs_arb[PPC-1:0]       link_clk │  ◄──────────
  ──────────────►   │  video_vs_arb[PPC-1:0        DDC_OUT  │  ──────────►
  ──────────────►   │  video_de_arb[PPC-1:0]            irq │  ──────────►
                    │                            video_rst  │  ──────────►
  ──────────────►   │  SB_STATUS_IN                         │
                    │                                       │
  ──────────────►   │  s_axi_cpu_aclk                       │
                    │                                       │
  ──────────────►   │  s_axi_cpu_aresetn                    │
                    │                                       │
  ──────────────►   │  s_axis_audio_aclk                    │
                    │                                       │
  ──────────────►   │  s_axis_audio_aresetn                 │
                    │                                       │
  ──────────────►   │  acr_cts[19:0]                        │
                    │                                       │
  ──────────────►   │  acr_n[19:0]                          │
                    │                                       │
  ──────────────►   │  acr_valid                            │
                    │                                       │
  ──────────────►   │  video_clk                            │
                    │                                       │
  ──────────────►   │  hpd                                  │
                    └──────────────────────────────────────┘
```
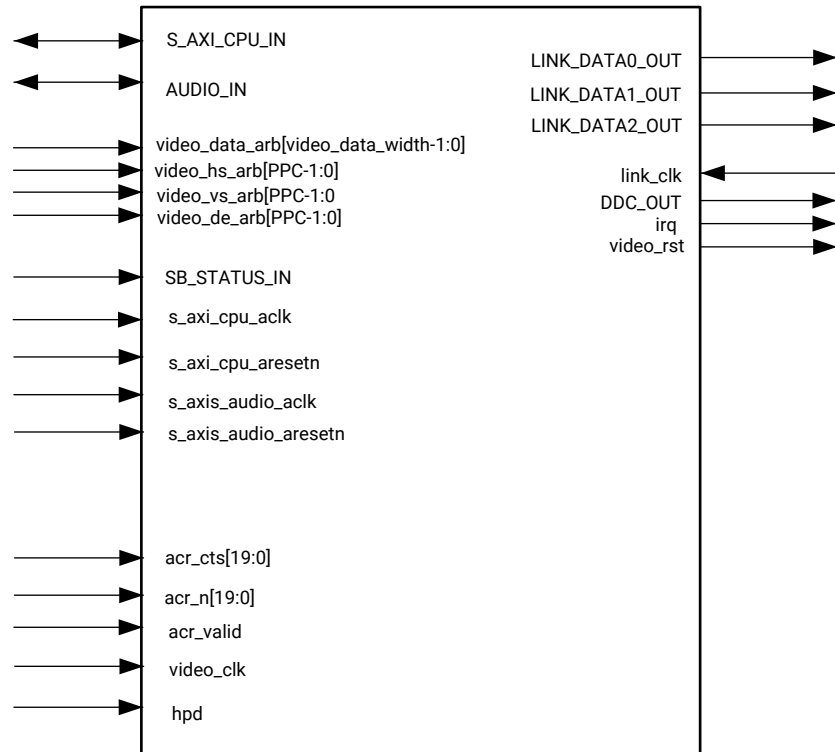
X23457-103019

Figure 17: **HDMI TX Subsystem Pinout – Native Video (Vectored DE) (HDCP 1.4 Only)**

S_AXI_CPU_IN

AUDIO_IN

video_data_arb[video_data_width-1:0]
video_hs_arb[PPC-1:0]
video_vs_arb[PPC-1:0
video_de_arb[PPC-1:0]

SB_STATUS_IN

s_axi_cpu_aclk

s_axi_cpu_aresetn

s_axis_audio_aclk

s_axis_audio_aresetn

acr_cts[19:0]

acr_n[19:0]

acr_valid

video_clk

hpd

LINK_DATA0_OUT
LINK_DATA1_OUT
LINK_DATA2_OUT
link_clk
DDC_OUT
irq
video_rst

hdcp14_key_aclk
hdcp14_key_aresetn
hdcp14_start_key_transmit
hdcp14_reg_key_sel[2:0]

hdcp14_irq
hdcp14_timer_irq

HDCP14_KEY_IN

X23456-103019

Send Feedback

Figure 18: **HDMI TX Subsystem Pinout – Native Video (Vectored DE) (HDCP 2.2 Only)**

```
         ┌─────────────────────────────────────────────┐
◄──────►  S_AXI_CPU_IN                 LINK_DATA0_OUT  ──────►
                                       LINK_DATA1_OUT  ──────►
◄──────►  AUDIO_IN                     LINK_DATA2_OUT  ──────►

──────►   video_data_arb[video_data_width-1:0]  link_clk  ◄──────
──────►   video_hs_arb[PPC-1:0]             DDC_OUT  ──────►
──────►   video_vs_arb[PPC-1:0                  irq  ──────►
──────►   video_de_arb[PPC-1:0]            video_rst  ──────►

──────►   SB_STATUS_IN                   hdcp22_irq  ──────►
                                     hdcp22_timer_irq  ──────►
──────►   s_axi_cpu_aclk

──────►   s_axi_cpu_aresetn

──────►   s_axis_audio_aclk

──────►   s_axis_audio_aresetn




──────►   acr_cts[19:0]

──────►   acr_n[19:0]

──────►   acr_valid

──────►   video_clk

──────►   hpd
         └─────────────────────────────────────────────┘
                                              X23455-103019
```

Send Feedback
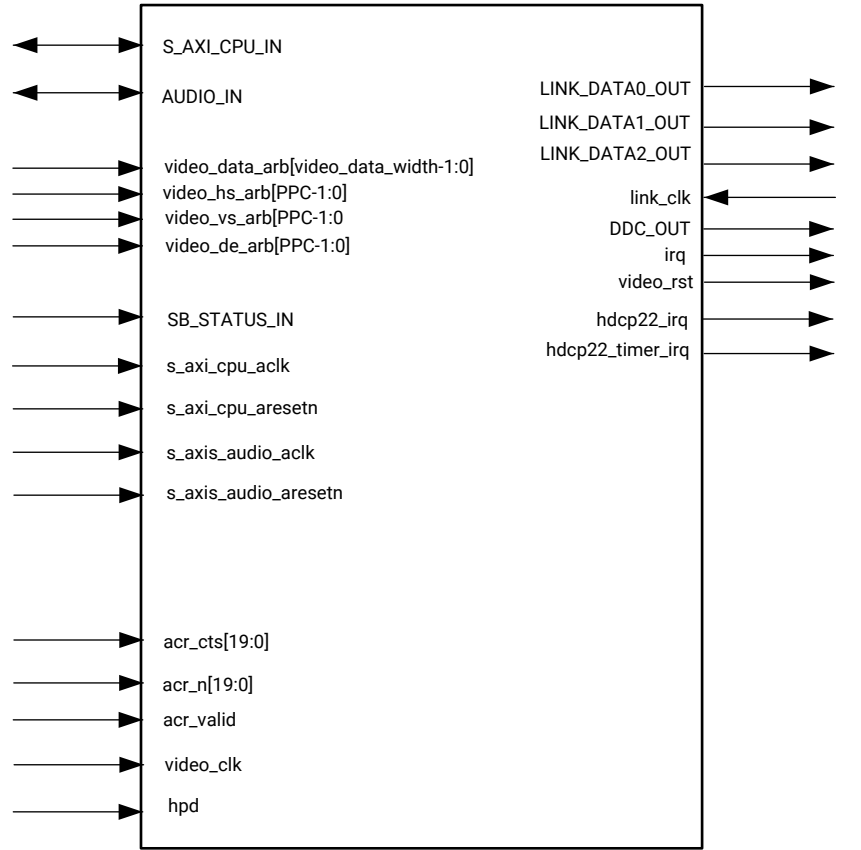
Figure 19: **HDMI TX Subsystem Pinout – Native Video (Vectored DE) (HDCP 1.4 and HDCP 2.2)**
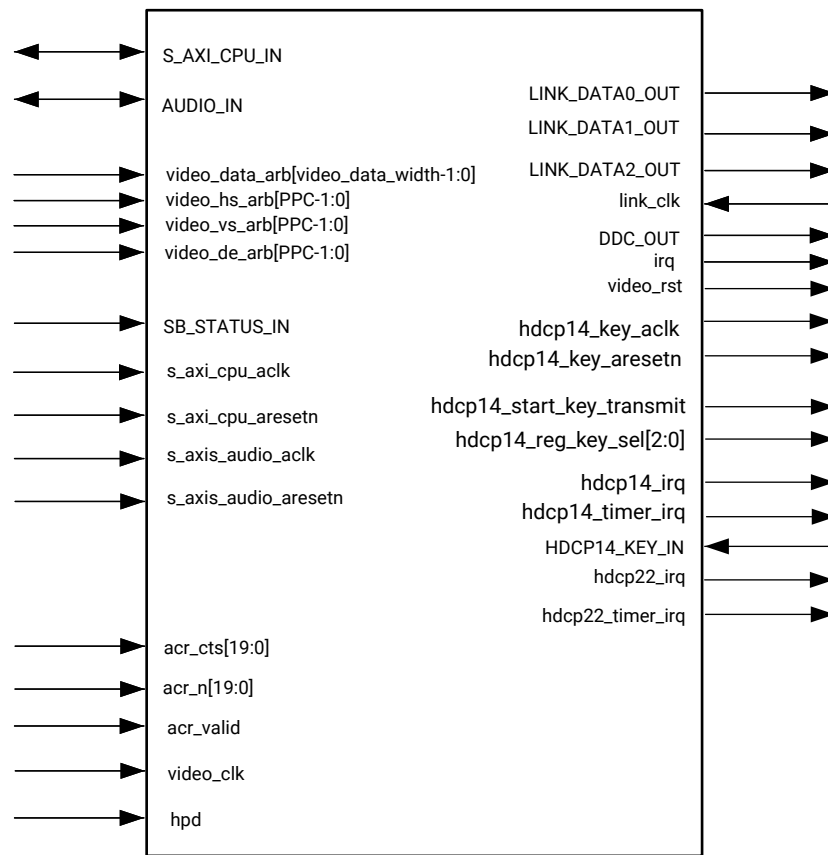
X23454-103119

CPU Interface

The following table shows the AXI4-Lite control interface signals. This interface is an AXI4-Lite interface and runs at the `s_axi_cpu_aclk` clock rate. Control of the subsystem is only supported through the subsystem driver.

IMPORTANT! *Direct register level access to any of the sub-modules is not supported. Instead, all accesses are done through the HDMI 1.4/2.0 TX Subsystem driver APIs.*

Table 2: **CPU Interface Ports**

Name	Direction	Width	Description
s_axi_cpu_aresetn	Input	1	Reset (Active-Low)
s_axi_cpu_aclk	Input	1	Clock for AXI4-Lite control interface
S_AXI_CPU_IN_awaddr	Input	17	Write address
S_AXI_CPU_IN_awprot	Input	3	Write address protection
S_AXI_CPU_IN_awvalid	Input	1	Write address valid

Send Feedback

Table 2: **CPU Interface Ports** *(cont'd)*

Name	Direction	Width	Description
S_AXI_CPU_IN_awready	Output	1	Write address ready
S_AXI_CPU_IN_wdata	Input	32	Write data
S_AXI_CPU_IN_wstrb	Input	4	Write data strobe
S_AXI_CPU_IN_wvalid	Input	1	Write data valid
S_AXI_CPU_IN_wready	Output	1	Write data ready
S_AXI_CPU_IN_bresp	Output	2	Write response
S_AXI_CPU_IN_bvalid	Output	1	Write response valid
S_AXI_CPU_IN_bready	Input	1	Write response ready
S_AXI_CPU_IN_araddr	Input	17	Read address
S_AXI_CPU_IN_arprot	Input	3	Read address protection
S_AXI_CPU_IN_arvalid	Input	1	Read address valid
S_AXI_CPU_IN_aready	Output	1	Read address ready
S_AXI_CPU_IN_rdata	Output	32	Read data
S_AXI_CPU_IN_rresp	Output	2	Read data response
S_AXI_CPU_IN_rvalid	Output	1	Read data valid
S_AXI_CPU_IN_rready	Input	1	Read data ready

AXI4-Stream Video Interface

The following table shows the signals for AXI4-Stream video input interface. This interface is an AXI4-Stream slave interface and runs at the `s_axis_video_aclk` clock rate. The data width is user-configurable in the Vivado IDE by setting **Max Bits Per Component** (BPC) and **Number of Pixels Per Clock on Video Interface** (PPC).

Table 3: **AXI4-Stream Video Interface**

Name	Direction	Width	Description
s_axis_video_aclk	Input	1	AXI4-Stream clock
s_axis_video_aresetn	Input	1	Reset (Active-Low)
VIDEO_IN_tdata	Input	(int((3*BPC*PPC+7)/8))*8	Data
VIDEO_IN_tlast	Input	1	End of line
VIDEO_IN_tready	Output	1	Ready
VIDEO_IN_tuser	Input	1	Start of frame
VIDEO_IN_tvalid	Input	1	Valid

Notes:

1. The Video Data width for AXI4-Stream interface is byte aligned. For example, for 10 bpc, 2 ppc, the data width is 64 bits.

Native Video Interface

The following table shows the signals for the native video interface. This interface is a standard video interface and runs at the `video_clk` clock rate. The data width is user-configurable in the Vivado IDE by setting **Max Bits Per Component** (BPC) and **Number of Pixels Per Clock on Video Interface** (PPC).

Table 4: **Native Video Interface**

Name	Direction	Width	Description
video_clk[1]	Input	1	Video clock
VIDEO_IN_active_video[5]	Input	1	Active video
VIDEO_IN_data	Input	video_data_width[3]	Data
VIDEO_IN_hsync[5]	Input	1	Horizontal sync
VIDEO_IN_vsync[5]	Input	1	Vertical sync

Notes:

1. `video_clk` is generated by the Video PHY Controller.
2. When native video interface is selected, `s_axis_video_aclk` and `s_axis_video_aresetn` are removed from the HDMI 1.4/2.0 TX Subsystem interface ports.
3. video_data_width = 3*BPC*PPC.
4. When native video interface is selected, there is no hardware reset.
5. You must provide the correct video timing information. You can choose to use the Xilinx® Video Timing Controller (vtc) or design your own vtc module to generate the timing control signals for the native video interface.

Native Video (Vectored DE) Interface

Both the AXI4-Stream and native video interface can only support resolutions with video timing divisible by PPC (2 or 4). To extend support to resolutions with video timing not divisible by PPC (2 or 4), the Native Video (Vectored Date Enable (DE)) interface can be used. The following table shows the signals. The data width is user-configurable in the Vivado IDE by setting Max Bits Per Component (BPC) and Number of Pixels Per Clock on Video Interface (PPC).

Table 5: **Native Video (Vectored DE) Interface**

Name	Direction	Width	Description
video_clk[1]	Input	1	Video clock
video_data_arb	Input	video_data_width[2]	Data
video_hs_arb	Input	PPC	Horizontal sync
video_vs_arb	Input	PPC	Vertical sync

Table 5: **Native Video (Vectored DE) Interface** *(cont'd)*

Name	Direction	Width	Description
video_de_arb	Input	PPC	Active video data enable

Notes:

1. `video_clk` is generated by the Video PHY Controller.
2. video_data_width = 3*BPC*PPC.
3. When native video interface is selected, `s_axis_video_aclk` and `s_axis_video_aresetn` are removed from the HDMI 1.4/2.0 TX Subsystem interface ports.
4. When native video interface is selected, there is no hardware reset.

The following table summarizes the supported video timing for each interface, depending on the PPC settings.

Table 6: **Supported Video Timing**

Video Interface	PPC	Video Timing		
		Divisible by 4	**Divisible by 2, But Not Divisible by 4**	**Not Divisible by 2**
AXI4-Stream	2	Supported	Supported	Not supported
	4	Supported	Not supported	Not supported
Native	2	Supported	Supported	Not supported
	4	Supported	Not supported	Not supported
Native (Vectored DE)	2	Supported	Supported	Supported
	4	Supported	Supported	Supported

Audio Input Stream Interface

The following table shows the signals for AXI4-Stream audio streaming interfaces. The audio interface transports 24-bits L-PCM or 16-bits HBR audio samples in the IEC 60958 format. A maximum of eight channels are supported. The audio interface is a 32-bit AXI4-Stream slave interface and runs at the `s_axis_audio_aclk` clock rate.

Table 7: **Audio Input Stream Interface**

Name	Direction	Width	Description
s_axis_audio_aclk	Input	1	Clock (The audio streaming clock must be greater than or equal or greater than 128 times the audio sample frequency)
s_axis_audio_aresetn	Input	1	Reset (Active-Low)

Table 7: **Audio Input Stream Interface** *(cont'd)*

Name	Direction	Width	Description
AUDIO_IN_tdata	Input	32	Data [31] P (Parity) [30] C (Channel status) [29] U (User bit) [28] V (Validity bit) [27:4] Audio sample word [3:0] Preamble code 4'b0001 Subframe 1/start of audio block 4'b0010 Subframe 1 4'b0011 Subframe 2
AUDIO_IN_tid	Input	3	Channel ID
AUDIO_IN_tready	Output	1	Ready
AUDIO_IN_tvalid	Input	1	Valid

Audio Clock Regeneration Interface

The audio clock regeneration (ACR) interface has a Cycle Time Stamp (CTS) parameter vector and an Audio Clock Regeneration Value (N) parameter vector. Both vectors are 20 bits wide. The valid signal is driven High when the CTS and N parameters are stable. For more information, see Chapter 7 of the HDMI 1.4 specification (*HDMI Specifications* (http://www.hdmi.org/manufacturer/specification.aspx)).

On the rising edge of the valid signal, the TX reads the CTS and N parameters from the ACR input interface and transmits an audio clock regeneration packet.

The following table shows the Audio Clock Regeneration (ACR) interface signals. This interface runs at the `s_axis_audio_aclk` clock rate.

Table 8: **Audio Clock Regeneration (ACR) Interface**

Name	Direction	Width	Description
acr_cts	Input	20	CTS
acr_n	Input	20	N
acr_valid	Input	1	Valid

If an HDMI system does not require audio, tie the following input ports Low:

- AUDIO_IN (except tready)

- s_axis_audio_aresetn

- s_axis_audio_aclk

- acr_cts

Send Feedback

- acr_n

- acr_valid

⭐ **IMPORTANT!** *When multiple channel audio is enabled in the system, ensure that the audio data is properly sent to its respective channel allocation. Unused channels must be packed with zero (that is, Mute) to avoid audio channel swapping, which means audio data might appear in unexpected channel locations.*

Note: The L-PCM (Packet Type 0x02) allows you to pack up to 24 bits of audio from the Audio Data Stream. The HBR (Packet Type 0x09) allows you to pack up to 16 bits of audio from the Audio Data Stream. The data is taken from MSB (bit 27:12). Compressed Audio (IEC 61937) can also be sent the same as L-PCM data (IEC60958). However, it is your responsibility to compress the audio data with your custom logic.

HDMI Link Output Interface

The following table shows the HDMI Link interface signals. This interface runs at the `link_clk` clock rate.

Table 9: **HDMI Link Interface**

Name	Direction	Width	Description
link_clk	Input	1	Link clock
LINK_DATA0_OUT_tdata	Output	ppc*10	Link data 0
LINK_DATA0_OUT_tvalid	Output	1	Link Data 0 Valid
LINK_DATA1_OUT_tdata	Output	ppc*10	Link data 1
LINK_DATA1_OUT_tvalid	Output	1	Link Data 1 Valid
LINK_DATA2_OUT_tdata	Output	ppc*10	Link data 2
LINK_DATA2_OUT_tvalid	Output	1	Link Data 2 Valid

Data Display Channel Interface

The following table shows the Data Display Channel interface signals.

Table 10: **Data Display Channel (DDC) Interface**

Name	Direction	Width	Description
ddc_scl_i	Input	1	DDC serial clock in
ddc_scl_o	Output	1	DDC serial clock out
ddc_scl_t	Output	1	DDC serial clock tri-state
ddc_sda_i	Input	1	DDC serial data in
ddc_sda_o	Output	1	DDC serial data out
ddc_sda_t	Output	1	DDC serial data tri-state

HDCP 1.4 Key Input Interface (AXI4-Stream Slave Interface)

The following table shows the signals for HDCP 1.4 key interface. This interface runs at the `hdcp14_key_aclk` (which is running at the AXI4-Lite clock).

Table 11: **HDCP 1.4 Key Input Interface**

Name	Direction	Width	Description
HDCP_KEY_IN_tdata	Input	64	HDCP 1.4 key data
HDCP_KEY_IN_tlast	Input	1	End of key data
HDCP_KEY_IN_tready	Output	1	Ready
HDCP_KEY_IN_tuser	Input	8	Start of key data
HDCP_KEY_IN_tvalid	Input	1	Valid
hdcp14_key_aclk	Output	1	AXI4-Stream clock
hdcp14_key_aresetn	Output	1	Reset (Active-Low)
hdcp14_start_key_transmit	Output	1	Start key transmit
hdcp14_reg_key_sel	Output	3	Key select

For the HDCP 1.4 transmitter, an HDCP Key Management module is needed, which is able to send keys over the AXI4-Stream interface to the HDCP 1.4 controller. The following figure shows an example of how the HDMI 1.4/2.0 TX Subsystem is connected to the HDCP Key Management module through a Key Management Bus (AXI4-Stream). The HDCP Key Management module is not part of the HDMI 1.4/2.0 TX Subsystem. For HDCP 1.4 design details, see the *HDCP 1.x Product Guide* (PG224).

Figure 20: **HDCP 1.4 Key Management Bus (AXI4-Stream)**

X16566-091819
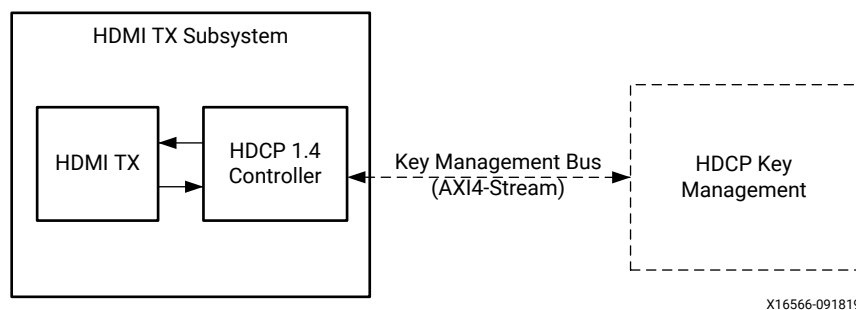
However, the HDCP 2.2 key is handled slightly differently as it is solely controlled by the software application. The user application is responsible for providing the infrastructure to securely store and retrieve the keys to be loaded into the HDCP 2.2 drivers. For the detailed list of keys that are required to be loaded by the user application, see the *HDCP 2.2 LogiCORE IP Product Guide* (PG249).

HDCP 2.2 Interrupt Outputs

The following table shows the signals for HDCP 2.2 interrupt output ports.

Table 12: **HDCP 2.2 Interrupt Output Interface**

Name	Direction	Width	Description
hdcp22_irq	Output	1	HDCP 2.2 interrupt
hdcp22_timer_irq	Output	1	HDCP 2.2 timer interrupt

Miscellaneous Signals with AXI4-Stream Video Interface

The following table shows the miscellaneous signals with AXI4-Stream video interface selected.

Table 13: **Miscellaneous Signals**

Name	Direction	Width	Description
hpd	Input	1	If XGUI option: Hot Plug Detect active-High (Default) 0 - Hot Plug Detect is released 1 - Hot Plug Detect is asserted If XGUI option: Hot Plug Detect active-Low[1] 0 - Hot Plug Detect is asserted 1 - Hot Plug Detect is released
locked	Output	1	Flag indicating the AXI4-Stream to Video Out Bridge in the subsystem is locked to the incoming video stream. 0 - no lock 1 - locked
irq	Output	1	Interrupt request for CPU. Active-High.
video_clk	Input	1	Reference Native Video Clock When AXI4-Stream is selected as Video Interface, an AXI4-Stream to Video Out Bridge module is added to the HDMI 1.4/2.0 TX Subsystem to convert AXI4-Stream Video into Native Video. The HDMI TX core uses this `video_clk` to clock in the Video Data together with the `video_cke_in`.
SB_STATUS_IN_tdata	Input	2	Side Band Status input signals Bit 0: link_rdy Bit 1: video_rdy
SB_STATUS_IN_tvalid	Input	1	Side Band Status input valid

Send Feedback

Table 13: **Miscellaneous Signals** *(cont'd)*

Name	Direction	Width	Description
fid	Input	1	Field ID for AXI4-Stream bus. Used only for interlaced video. 0 - even field 1 - odd field This bit is sampled coincident with the SOF on the AXI4-Stream bus. If the signal is not used, set the input to Low..

Notes:

1. The Hot Plug Detect (HPD) signal is driven by an HDMI sink and asserted when the HDMI cable is connected to notify the HDMI source of the presence of an HDMI sink. In some cases, the HDMI sink is simply connected to 5V power signal. Therefore, in the PCB, if you choose to use a voltage divider or level shifter, the HPD polarity remains as active-High. However, if you add an inverter to the HPD signal, then the HPD polarity must be set to active-Low in the HDMI 1.4/2.0 TX Subsystem GUI.

Miscellaneous Signals with Native Video Interface

The following table shows the miscellaneous signals with native video interface selected.

Table 14: **Miscellaneous Signals with Native Video Interface**

Name	Direction	Width	Description
hpd	Input	1	If XGUI option: Hot Plug Detect Active-High (Default) 0 - Hot Plug Detect is released 1 - Hot Plug Detect is asserted If XGUI option: Hot Plug Detect Active-Low [1] 0 - Hot Plug Detect is asserted 1 - Hot Plug Detect is released
irq	Output	1	Interrupt request for CPU. Active-High.
SB_STATUS_IN_tdata	Input	2	Side Band Status input signals Bit 0: link_rdy Bit 1: video_rdy
SB_STATUS_IN_tvalid	Input	1	Side Band Status input valid
video_rst	Output	1	Video reset signal in video_clk domain. Active-High.

Notes:

1. The Hot Plug Detect (HPD) signal is driven by an HDMI sink and asserted when the HDMI cable is connected to notify the HDMI source of the presence of an HDMI sink. In most cases, the HDMI sink is simply connected to 5V power signal. Therefore, in the PCB, if you choose to use a voltage divider or level shifter, the HPD polarity remains as Active-High. However, if you add an inverter to the HPD signal, then the HPD polarity must be set to Active-Low in the HDMI 1.4/2.0 TX Subsystem GUI.

Clocks and Resets

The following table provides an overview of the clocks and resets. See Clocking and Resets for more information.

Send Feedback

Table 15: **Clocks and Resets**

Name	Direction	Width	Description
s_axi_cpu_aclk	Input	1	AXI4-Lite CPU control interface clock.
s_axi_cpu_aresetn	Input	1	Reset, associated with s_axi_cpu_aclk (active-Low). The s_axi_cpu_aresetn signal resets the entire subsystem including the data path and AXI4-Lite registers.
s_axis_video_aclk	Input	1	AXI4-Stream video input clock.
s_axis_video_aresetn	Input	1	Reset, associated with s_axis_video_aclk (active-Low). Resets the AXI4-Stream data path for the video input.
s_axis_audio_aclk	Input	1	AXI4-Stream Audio input clock. (The audio streaming clock must be greater than or equal to 128 times the audio sample frequency)
s_axis_audio_aresetn	Input	1	Reset, associated with s_axis_audio_aclk (active-Low). Resets the AXI4-Stream data path for the audio input.
link_clk	Input	1	HDMI Link data output clock. This connects to the Video PHY Controller Link clock output.
video_clk	Input	1	Clock for the native video interface.

Notes:
1. The reset should be asserted until the associated clock becomes stable.

Related Information
Clocking
Resets

Designing with the Subsystem

This chapter includes guidelines and additional information to facilitate designing with the subsystem.

General Design Guidelines

The subsystem connects to other hardware components to construct a complete HDMI TX system. These hardware components usually are different from device to device. For example, Kintex®-7 devices have a different PLL architecture from UltraScale™ devices. You must fully understand the system and adjust the subsystem parameters accordingly.

Control Interface

The HDMI 1.4/2.0 TX Subsystem is controlled over an AXI4-Lite interface using the Xilinx API described in the appendix, Application Software Development; the appendix also describes how to integrate the subsystem API into a software application. The API is the only provided control for the HDMI TX subsystems and the register interface descriptions are not provided.

Audio Data Stream

An AXI4-Stream audio cycle is illustrated in the following figure. The data is captured when both the valid (TVLD) and ready (TRDY) signals are asserted. The HDMI 1.4/2.0 TX Subsystem expects the channels in sequential order. If the channel data is not in order, the channel data might be mapped into other channel sample slots. Therefore, ensure that the audio stream source sends out adjacent channels in sequential order (CH0, CH1, etc).

Figure 21: **Audio Cycle**
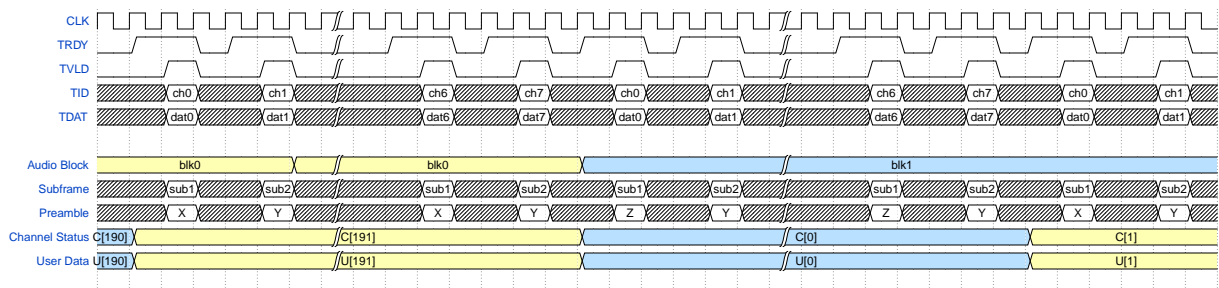
In the HDMI 1.4/2.0 TX Subsystem, L-PCM (IEC 60958, Packet Type 0x02) and HBR (Packet Type 0x09) are handled by the hardware. The audio information, such as the format and number of channels, are set through API calls. Then the hardware packs the audio into the blanking period of the video stream accordingly.

```
void XV_HdmiTxSs_SetAudioFormat(XV_HdmiTxSs *InstancePtr, u8 format);
void XV_HdmiTxSs_SetAudioChannels(XV_HdmiTxSs *InstancePtr, u8
AudioChannels);
```

where:

1. `InstancePtr` is a pointer to the `XV_HdmiTxSs` instance.

2. `format` is a selector of Audio Format

   - 1: HBR
   - 0: L-PCM

IMPORTANT! *When multiple channel audio is enabled in the system, ensure that the audio data is properly sent to their perspective channel allocation. Unused channels must be packed with zero (i.e. Mute) to avoid audio channel swapping, which means audio data might appear in unexpected channel locations.*

Video Input Stream Interface

The AXI4-Stream video interface supports dual or quad pixels per clock with 8 bits, 10 bits, 12 bits and 16 bits per component for RGB, YUV444, and YUV420 color spaces. The color depth in YUV422 color space is always 12-bits per pixel.

When the parameter, **Max Bits Per Component**, is set to 16, the following figure shows the data format for quad pixels per clock to be fully compliant with the AXI4-Stream video protocol.

Figure 22: **Quad Pixels Data Format (Max Bits Per Component = 16)**
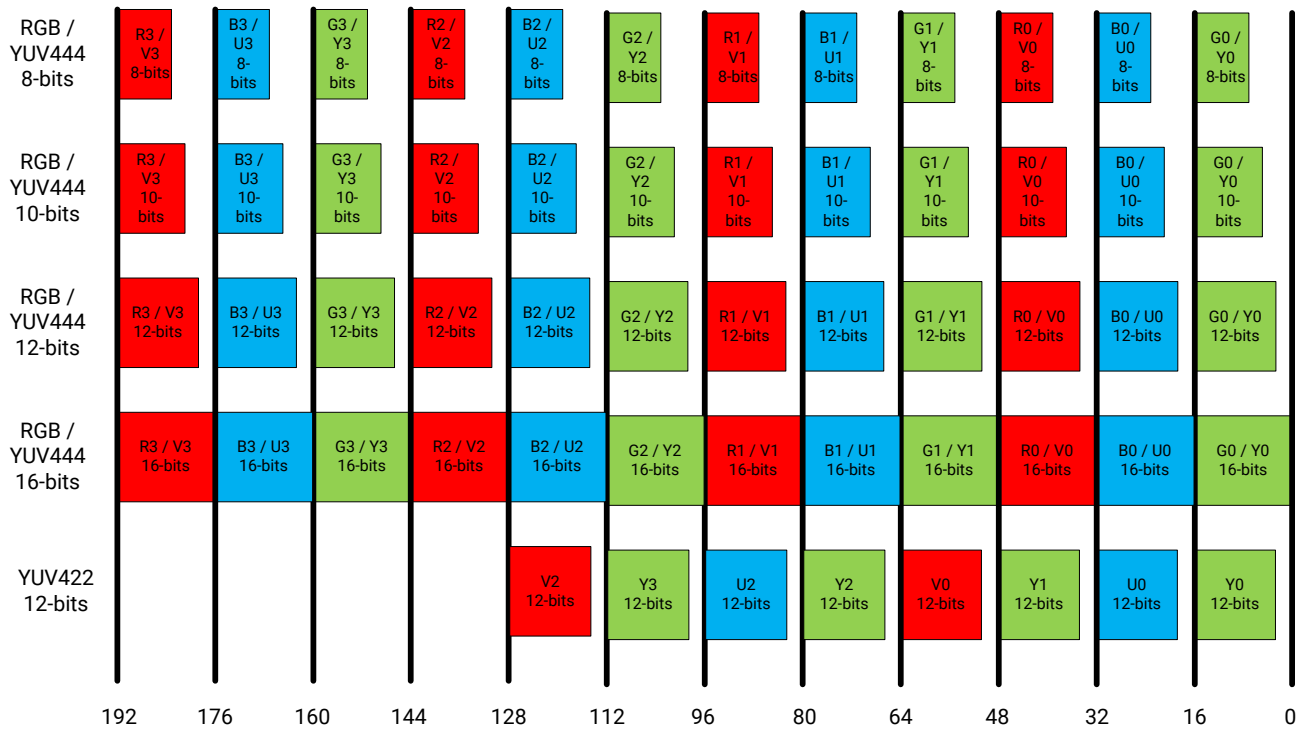
X15244-111915

A data format for a fully compliant AXI4-Stream video protocol dual pixels per clock is illustrated in the following figure.

Figure 23: **Dual Pixels Data Format (Max Bits Per Component = 16)**
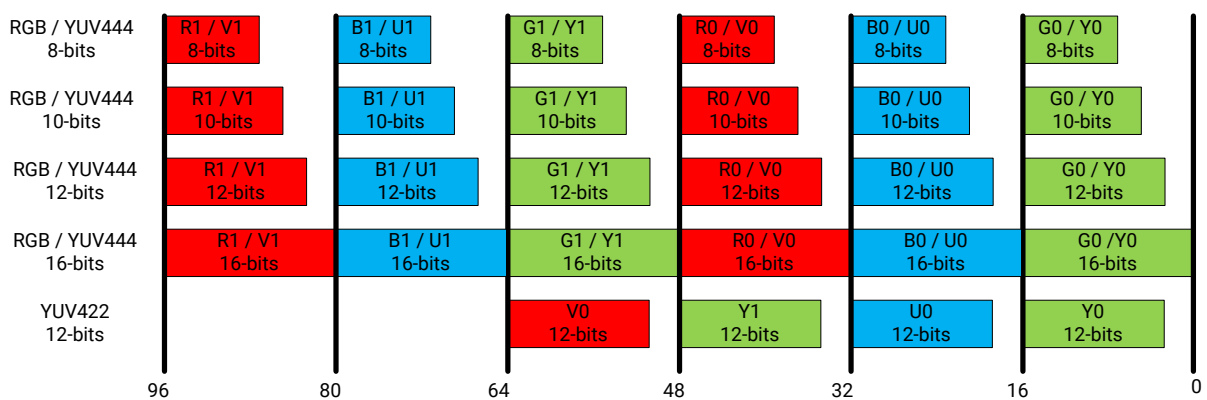
X15245-111615

When the parameter, Max Bits Per Component, is set to 12, video formats with actual bits per component larger than 12 is truncated to the Max Bits Per Component. The remaining least significant bits are discarded. If the actual bits per component is smaller than Max Bits Per Component set in the Vivado IDE, all bits are transported with the MSB aligned and the remaining LSB bits are padded with 0. This applies to all Max Bits Per Component settings.

Table 16: **Max Bits Per Component Support**

Max Bits Per Component	Actual Bits Per Component	Bits Transported by Hardware
16	8	[7:0]
	10	[9:0]
	12	[11:0]
	16	[15:0]
12	8	[7:0]
	10	[9:0]
	12	[11:0]
	16	[15:4]
10	8	[7:0]
	10	[9:0]
	12	[11:2]
	16	[15:6]
8	8	[7:0]
	10	[9:2]
	12	[11:4]
	16	[15:8]

As an illustration, when **Max Bits Per Component** is set to 12, the following figure shows the data format for quad pixels per clock to be fully compliant with the AXI4-Stream video protocol.

Send Feedback

Figure 24: **Quad Pixels Data Format (Max Bits Per Component = 12)**
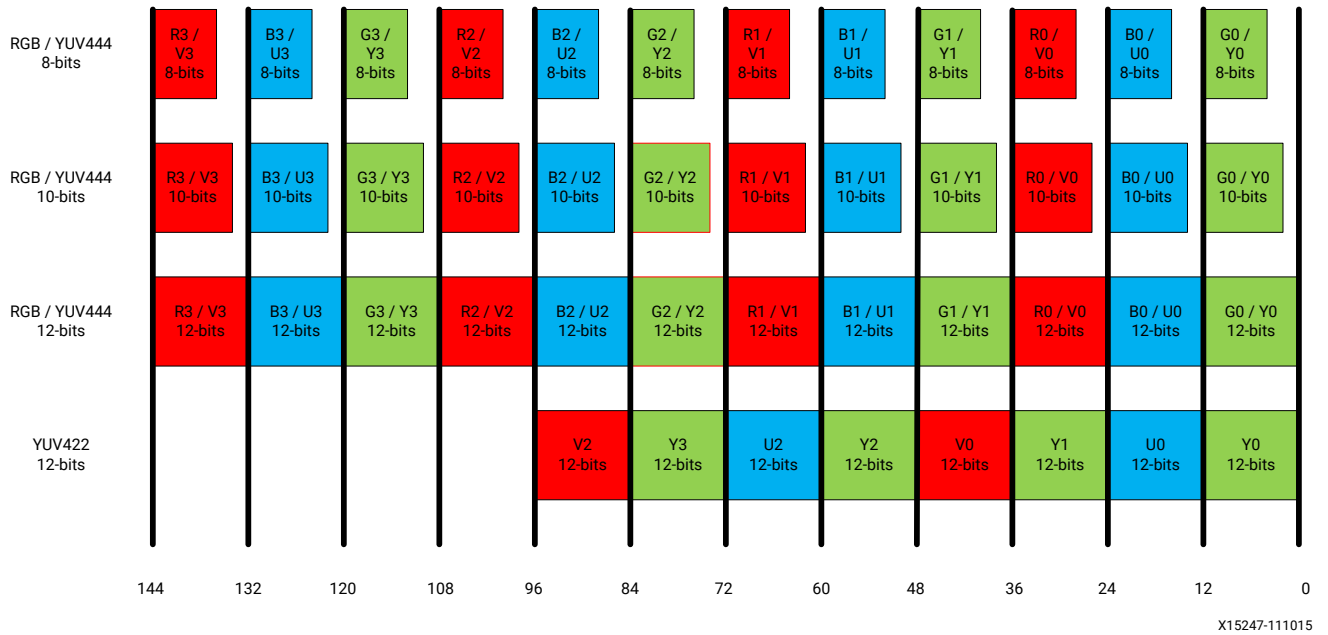
A data format for a fully compliant AXI4-Stream video protocol with dual pixels per clock is illustrated in the following figure.

Figure 25: **Dual Pixels Data Format (Max Bits Per Component = 12)**
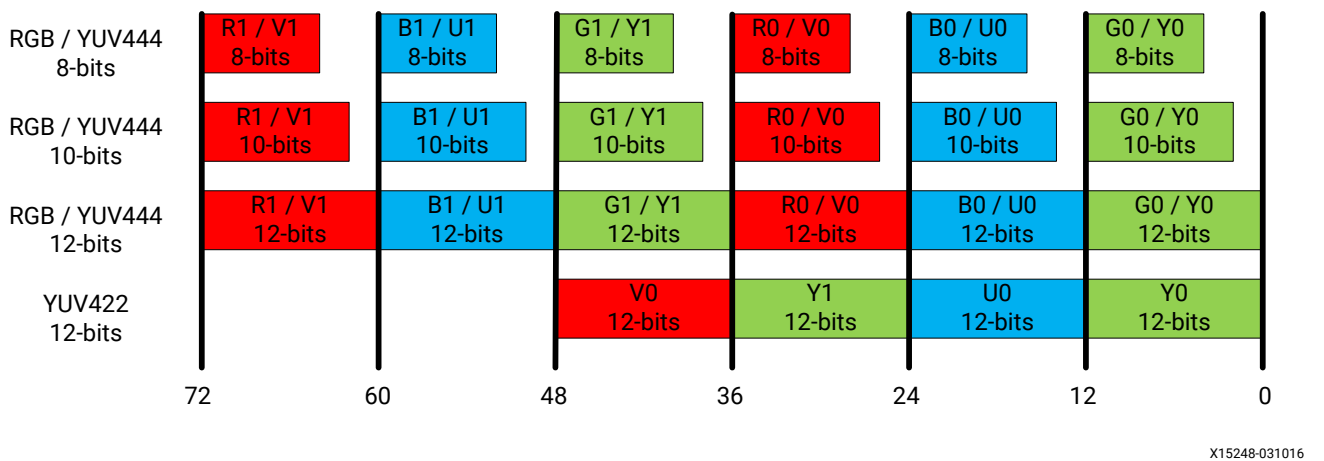
The video interface can also transport quad and dual pixels in the YUV420 color space. The following figures show the data format for quad and dual pixels formats, respectively.

Figure 26: **YUV420 Color Space Quad Pixels Data Format (Native)**

X15250-082219
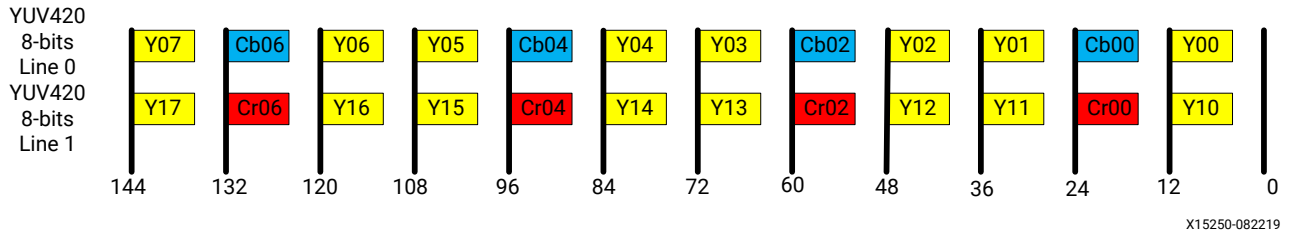
Figure 27: **YUV420 Color Space Dual Pixels Data Format (Native)**

X15251-111015
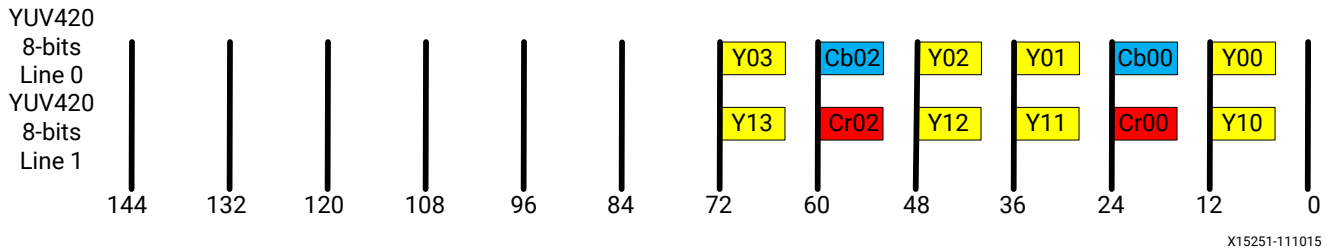
Similarly, for YUV 4:2:0 deep color (10, 12, or 16 bits), the data representation is the same as shown in the previous two figures. The only difference is that each component carries more bits (10, 12, and 16). However the current data format is not compliant with the AXI4-Stream video protocol. Therefore, a remapping feature is added to the HDMI Receiver Subsystem to convert HDMI native video into AXI4-Stream video. This feature can be enabled from the HDMI Receiver Subsystem GUI. To illustrate how the data remapping feature works for YUV 4:2:0 video from Native Video into AXI4-Stream, the previous figure is extended and represented in the following figure to show native video data associated with the clock and control signals.

Send Feedback

Figure 28: **Native HDMI Video Interface**
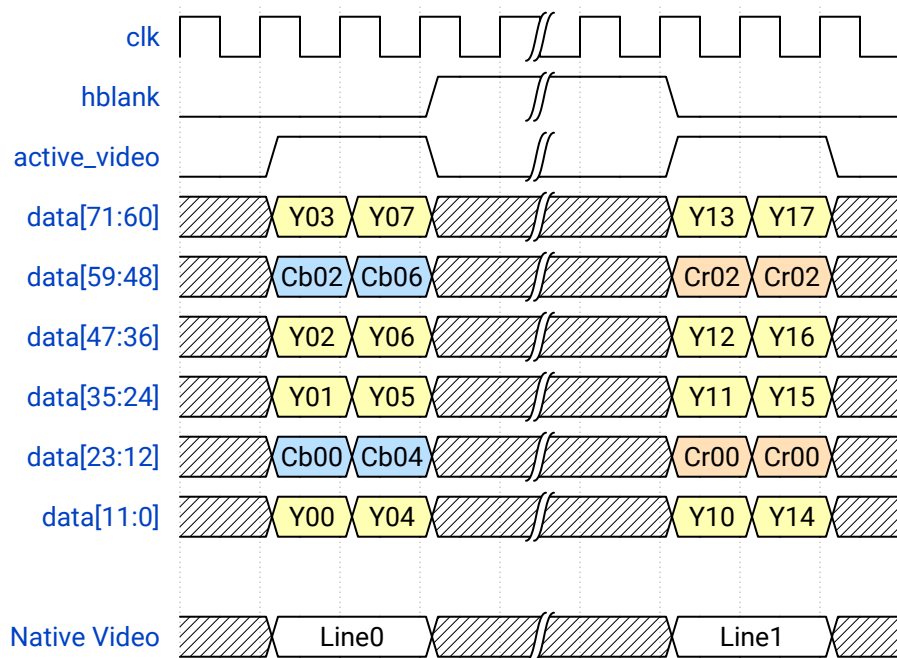
When transporting using AXI4-Stream, the data representation must be compliant with the protocol defined in the *AXI4-Stream Video IP and System Design Guide* (UG934). With the remapping feature, the same native video data is converted into AXI4-Stream formats, shown in the following figure. As stated in *AXI4-Stream Video IP and System Design Guide* (UG934), the 4:2:0 format adds vertical sub-sampling to the 4:2:2 format, which is implemented in Video over AXI4-Stream by omitting the chroma data on every other line.

Figure 29: **YUV 4:2:0 AXI4-Stream Video Data (Dual Pixel per Clock)**
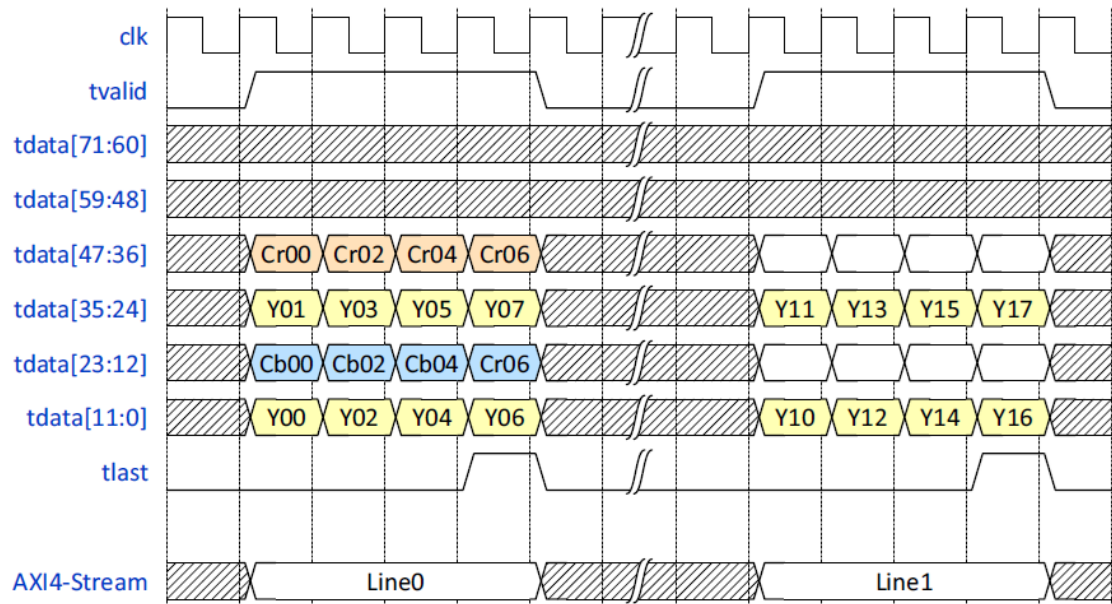
Note: For RGB/YUV444/YUV422 formats, video data is directly mapped from AXI4-Stream to the native video interface without any line buffer. Therefore,the Quad and Dual Pixels Data Format graphics shown previously, represent the data interface for both AXI4-Stream and native video. The ,control signals are omitted in the figures.

The subsystem provides full flexibility to construct a system using the configuration parameters, maximum bits per component and number of pixels per clock. Set these parameters so that the video clock and link clock are supported by the targeted device. For example, when dual pixels per clock is selected, the AXI4-Stream video need to run at higher clock rate comparing with quad pixels per clock design. In this case, it is more difficult for the system to meet timing requirements. Therefore the quad pixels per clock data mapping is recommended for designs intended to send higher video resolutions, for example, 4kp60 video.

Some video resolutions (for example, 720p60) have horizontal timing parameters (1650) which are not a multiple of 4. In this case the dual pixels per clock data mapping must be chosen.

For more information on the video AXI4-Stream interface and video data format, see the *AXI4-Stream Video IP and System Design Guide* (UG934).

Interlaced Video

The HDMI 1.4/2.0 TX Subsystem supports both AXI4-Stream video and native video interface.

- When AXI4-Stream is selected, an AXI4-Stream to Video Out core is used to support the HDMI 1.4/2.0 TX Subsystem. Because the AXI4-Stream carries only active video data, the AXI4-Stream to Video Out core takes input from an AXI4-Stream slave interface and converts it into a Native Video stream, which is then fed to the HDMI TX core.

- When Native Interface is selected, the native video stream must be prepared and fed to the `Video_In` port of the HDMI 1.4/2.0 TX Subsystem, which is directly connected to the HDMI TX core inside the HDMI 1.4/2.0 TX Subsystem.

- When Native Interface (Vectored DE) is selected, the native video stream must be prepared and fed to the video_*_arb ports of the HDMI 1.4/2.0 TX Subsystem, which is directly connected to the HDMI TX core inside the HDMI 1.4/2.0 TX Subsystem.

The HDMI 1.4/2.0 TX Subsystem is designed to support both progressive and interlaced video. In this section, the focus is to show how to handle interlaced video as it is more straightforward for progressive video.

Taking 1920x1080@50 Hz (I) as an example, the detailed timing information is shown in the following table.

Table 17: **Timing Data**

Name	Timing Field Subset	Value
HActive		1920
HBlank		720
	HFrontPorch	528
	HSyncWidth	44
	HBackPorch	148
HTotal		2640
VActive		540
F0VBlank		22
	F0PVFrontPorch	2
	F0PVSyncWidth	5
	F0PVBackPorch	15
F0PVTotal		562
F1VBlank		23
	F1VFrontPorch	3
	F1VSyncWidth	5
	F1VBackPorch	15
F1VTotal		563

For interlaced video, each frame consists of two fields. One field carries the odd lines and the other field carries the even lines. After putting both fields together, you get the complete frame. Therefore,

Vertical Active per Field = Vertical Active Lines / 2

Send Feedback

Frame Rate = Field Rate / 2.

In this example,

VActive = 1080/2 = 540
Field Rate = 50 Hz
Frame Rate = 50/2 = 25 Hz

To design using the AXI4-Stream Interface, generate two fields of video with timing using the values from the previous table. For complete timing information, refer to CEA-861-F. Only active video data compliant with AXI4-Stream protocol is needed. The AXI4-Stream to Video Out core inside the HDMI 1.4/2.0 TX Subsystem converts the AXI4-Stream video into native video. Ensure that `fid` is driven to align with the field video data. For details, refer to the *AXI4-Stream to Video Out LogiCORE IP Product Guide* (PG044).

To design using the Native interface, generate two fields of native video with timing using the values from the previous table. Ensure that the HSYNC and VSYNC are driven using the values from the previous table. Because a frame can have odd number of lines (for example, 1125 for 1080i50), the two fields might result in a different total number of lines (for example, Field 0 has 522 lines, and Field 1 has 523 lines).

Interlaced Video with Pixel Repetition

For some video formats with TMDS rates below 25 MHz (for example, 13.5 for 480i/NTSC) can be transmitted using a pixel-repetition scheme.

In the HDMI 1.4/2.0 TX Subsystem,

* Enabled through GUI parameter upon IP generation (select Video over AXIS compliant NTSC/PAL Support)

* Pixel repetition is only available for the AXI4-Stream interface

Note: Using AXI4-Stream, pixel repetition only supports RGB and YUV444 color space, but not 12-bit YUV422. Using the native interface, you can create customized pixel replication logic that supports all color spaces including 12-bit YUV422.

After it is enabled in the HDMI 1.4/2.0 TX Subsystem, you must prepare interlaced video as normal, then the HDMI 1.4/2.0 TX Subsystem replicates each pixel twice before sending the data out.

Because only NTSC/480i60 and PAL/576i50 are supported for pixel repetition, ensure that the correct XVDIC from the Video Common library is selected:

* XVIDC_VM_1440x576_50_I => PAL/576i50

* XVIDC_VM_1440x480_60_I => NTSC/480i60

For example,

480i60 video is 720x480 @ 30 Hz, which is made from two fields of 720x240 @ 30 Hz video.

You must select XVIDC_VM_1440x480_60_I in the software. Then in the hardware system, prepare two fields of 720x240 @ 30 Hz video (AXI4-Stream Video) and send them to the HDMI 1.4/2.0 TX Subsystem. Then HDMI 1.4/2.0 TX Subsystem repeats each pixel twice. When the video is sent out by the HDMI 1.4/2.0 TX Subsystem, it is sent as two fields of 1440x240 @ 30 Hz video.

```
HDMI TX timing
------------
HDMI TX Mode - HDMI
HDMI Video Mask is Disabled

        Color Format:     RGB
        Color Depth:      8
        Pixels Per Clock: 2
        Mode:             Interlaced
        Frame Rate:       60Hz
        Resolution:       1440x480@60Hz (I)
        Pixel Clock:      27027000

        HSYNC Timing: hav=1440, hfp=38, hsw=124(hsp=0), hbp=114, htot=1716
        VSYNC Timing (Field 0): vav=0240, vfp=04, vsw=03(vsp=0), vbp=015, vtot=0262
        VSYNC Timing (Field 1): vav=0240, vfp=05, vsw=03(vsp=0), vbp=015, vtot=0263
```

Clocking

The `S_AXI_CPU_IN`, `VIDEO_IN`, and `AUDIO_IN` can be run at their own clock rate. The HDMI link interfaces and native video interface also run at their own clock rate. Therefore, five separate clock interfaces are provided called `s_axi_cpu_aclk`, `s_axis_video_aclk`, `s_axis_audio_aclk`, `link_clk`, and `video_clk` respectively.

The audio streaming clock must be greater than or equal to 128 times the audio sample frequency. Because audio clock regeneration is not part of the HDMI 1.4/2.0 TX Subsystem you must provide an audio clock to the application. This can be achieved by using an internal PLL or external clock source.

IMPORTANT! *As stated in HDMI 1.4b Specification section 7.2.4: For any IEC 61937 compressed audio with an IEC 60958 frame rate at or below 192 kHz, the ACR fs value shall be equal to the frame rate. For any such stream with an IEC 60958 frame rate above 192 kHz, the ACR fs value shall be a quarter of the frame rate. Therefore, for HBR audio, while calculating N & CTS, the fs to be used is a quarter of the frame rate. (for example, 768 kHz => 768/4=192 kHz).*

The HDMI clock structure is illustrated in the following figure and table.

Send Feedback

Figure 30: **HDMI Clocking Structure**
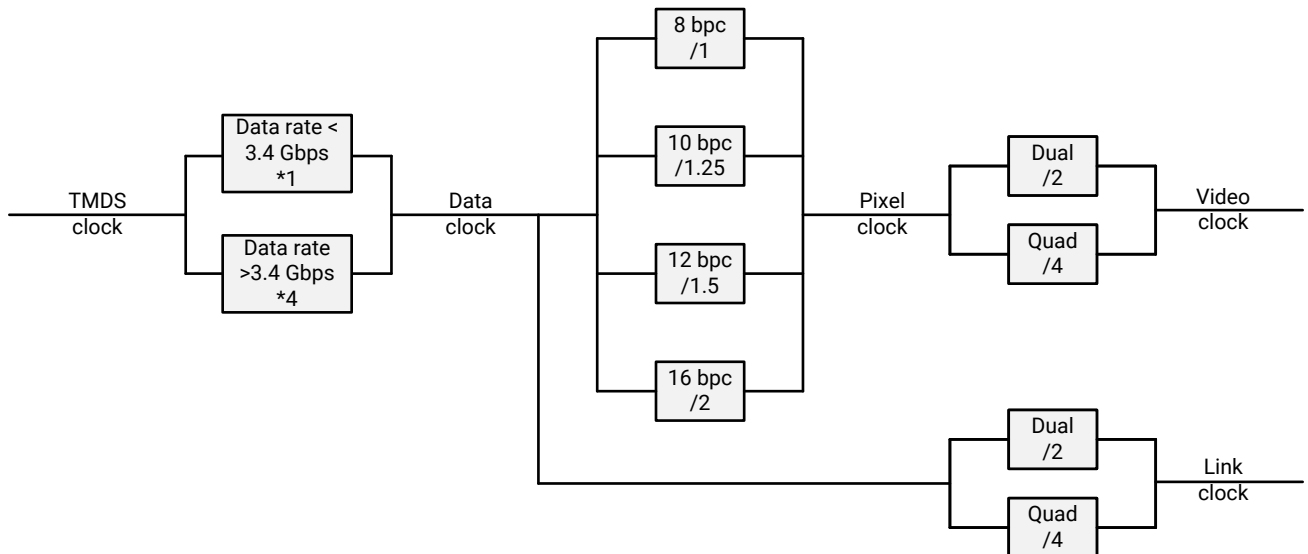
X23117-081419

Table 18: **HDMI Clocking**

Clock	Function	Freq/Rate	Example[1]
TMDS	Source synchronous clock to HDMI interface (This is the actual clock on the HDMI cable).	= 1/10 data rate (for data rates < 3.4 Gb/s)	Data rate = 2.97 Gb/s TMDS clock = 2.97/10 = 297 MHz
		= 1/40 data rate (for data rates > 3.4 Gb/s)	Data rate = 5.94 Gb/s TMDS clock = 5.94/40 = 148.5 MHz
Data	This is the actual data rate clock. This clock is not used in the system. It is only listed to illustrate the clock relations.	= TMDS clock (for data rates < 3.4 Gb/s)	Data rate = 2.97 Gb/s Data clock = TMDS clock * 1 = 297 MHz
		= TMDS clock * 4 (for data rates > 3.4 Gb/s)	Data rate = 5.94 Gb/s Data clock = TMDS clock * 4 = 594 MHz TMDS clock = 148.5 MHz
Link	Clock used for data interface between the Video PHY layer module and subsystem	for dual pixel video: clock=data clock/2 for quad pixel video: clock=data clock/4	TMDS clock = 297 MHz Data clock = 297 MHz Link clock = 297 MHz/2=148.5 MHz for dual pixel wide interface Link clock = 297 MHz/4 = 74.25 MHz for quad pixel wide interface Data clock = 594 MHz Link clock = 594 MHz/2=297 MHz for dual pixel wide interface Link clock = 594 MHz/4=148.5 MHz for quad pixel wide interface
Pixel	This is the internal pixel clock. This clock is not used in the system. It is only listed to illustrate the clock relations.	for 8 bpc pixel clock = data clock for 10bpc pixel clock = data clock/1.25 for 12bpc pixel clock = data clock/1.5 for 16 bpc pixel clock = data clock/2	

Send Feedback

Chapter 4: Designing with the Subsystem

Table 18: **HDMI Clocking** *(cont'd)*

Clock	Function	Freq/Rate	Example[1]
Video	Clock used for video interface	for dual pixel video clock = pixel clock/2 for quad pixel video clock = pixel clock/4	297 MHz/2 = 148.5 MHz for dual pixel wide interface 297 MHz/4 = 74.25 MHz for quad pixel wide interface For more information on how to choose the correct PLL in the targeted devices, see the *Video PHY Controller LogiCORE IP Product Guide* (PG230).

Notes:

1. The examples in the Example column are only for reference and do not cover all the possible resolutions. Each GT has its own hardware requirements and limitations. Therefore, to use the HDMI 1.4/2.0 TX Subsystem with different GT devices, calculate the clock frequencies and make sure the targeted device can support it. When using the HDMI 1.4/2.0 TX Subsystem with the Xilinx Video PHY Controller IP core, more information can be found in the *Video PHY Controller LogiCORE IP Product Guide* (PG230).

For example, 1080p60, 12 BPC, and 2 PPC are used to show how all the clocks are derived.

Table 19: **Example Settings**

Video Resolution	Horizontal Total	Horizontal Active	Vertical Total	Vertical Active	Frame Rate (Hz)
1080p60	2200	1920	1125	1080	60

The pixel clock represents the total number of pixels that need to be sent every second.

Therefore,

Pixel clock = Htotal × Vtotal × Frame Rate =2200 x 1125 x 60 =148,500,000 = 148.5 MHz
Video clock = (Pixel clock)/PPC=148.5/2=74.25 MHz
Data clock = Pixel clock × BPC/8=148.5× 12/8=222.75 MHz
Link clock = (Data clock)/PPC=222.75/2=111.375 MHz

Using the associative property in this example,

Data clock = 222.75 MHz < 340 MHz

then

TMDS clock = Data clock = 222.75 MHz

The following figure shows how the clock is distributed in the HDMI 1.4/2.0 TX Subsystem and the relationship to the Video PHY Controller.

Send Feedback

Figure 31: **HDMI Transmitter Subsystem and Video PHY Controller**
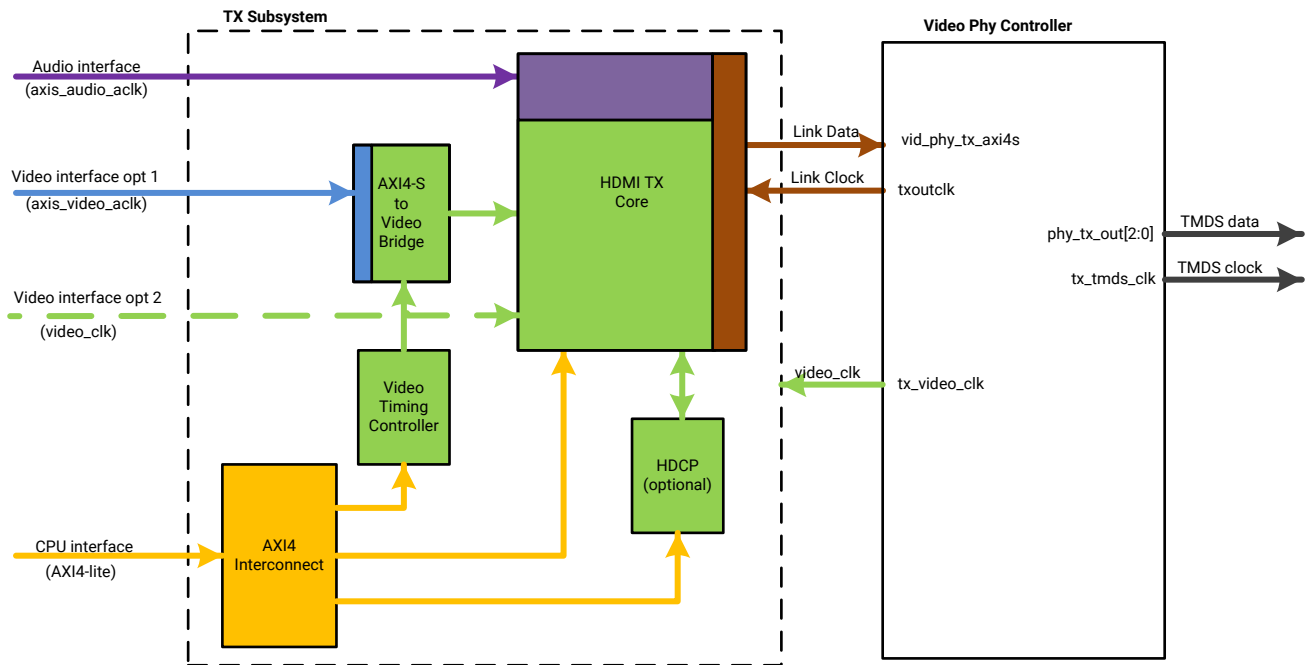
The HDMI 1.4/2.0 TX Subsystem can support either AXI4-Stream video or native video.

- When AXI4-Stream is selected, the video stream is sent to the HDMI 1.4/2.0 TX Subsystem through the video interface in AXI4-Stream format running at `axis_video_aclk`. The AXI4-Stream is then processed and converted into native video stream by the AXI4-Stream to Video Out bridge core with the help of the Video Timing Controller module. Because the AXI4-Stream carries only active video data, the AXI4-Stream to Video Out core takes input from an AXI4-Stream slave interface and converts it into a native video stream, which is then fed to the HDMI TX core. The HDMI TX core then packs the native video data with audio data and other auxiliary data into Link Data and sent to the Video PHY Controller at `Link Clock`.

- When the native (or native vectored DE) interface is selected, a video stream is sent to the HDMI 1.4/2.0 TX Subsystem as native video and directly passed to the HDMI TX core. The data is then packed with audio data and other auxiliary data into `Link Data` and sent to Video PHY Controller at `Link Clock`.

Based on the system requirement, the Video PHY Controller generates `Link Clock` and `Video Clock` for the HDMI 1.4/2.0 TX Subsystem for each targeted video resolution. Meanwhile, the `axis_audio_aclk`, `axis_video_aclk`, and AXI4-Lite clocks are free running clocks in the system usually generated by the clock wizard.

Resets

Each AXI input interface has its own reset signal. The reset signals, `s_axi_cpu_aresetn`, `s_axis_video_aresetn` and `s_axis_audio_aresetn` are for `S_AXI_CPU_IN`, `VIDEO_IN` (AXI4-Stream Video Interface), and `AUDIO_IN` respectively. These three reset signals are active-Low. Because the reset signal is used across multiple sub-blocks in the subsystem, keep the system in the reset state until all the clocks are stabilized. You can use the `locked` signal from the clock generation block as a reset signal.

Note: There is no dedicated hardware reset for the VIDEO_IN interface when Native Video interface is selected. However, the HDMI 1.4/2.0 TX Subsystem outputs a `video_rst` signal, which you can use to reset its Native Video Source generation modules.

Send Feedback

Design Flow Steps

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

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)

- *Vivado Design Suite User Guide: Designing with IP* (UG896)

- *Vivado Design Suite User Guide: Getting Started* (UG910)

Customizing and Generating the Subsystem

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

The HDMI 1.4/2.0 TX Subsystem can be added to a Vivado IP integrator block design in the Vivado Design Suite and can be customized using IP catalog.

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

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP subsystem 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 the Enter key on your keyboard 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.
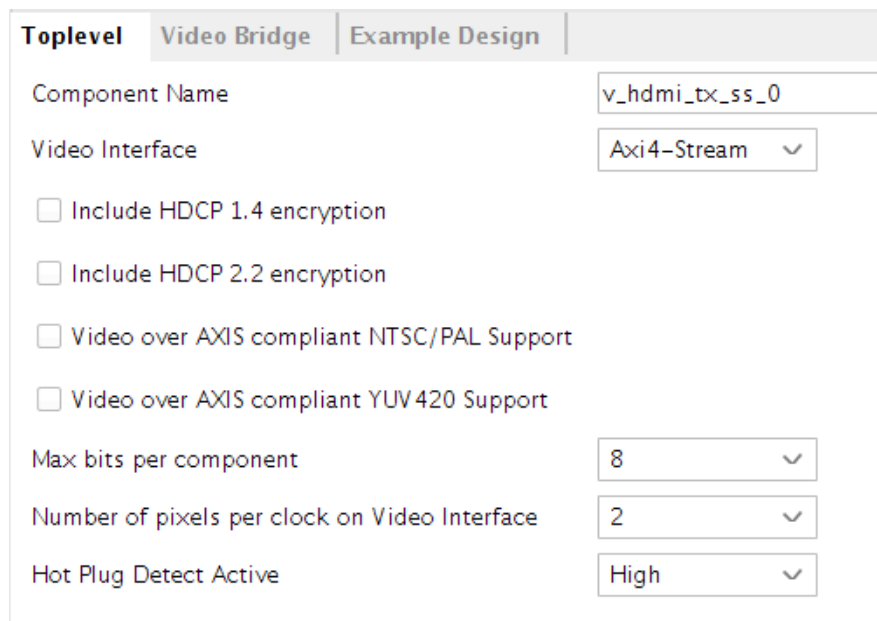
For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

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

Top Level Tab

The top level tab is shown in the following figure.

Figure 32: **Top Level Tab**

The parameters on the top level tab are as follows.

- **Component Name:** The component name is set automatically by IP integrator.

- **Video Interface:** This option selects the video interface for the HDMI 1.4/2.0 TX Subsystem. The allowable options are AXI4-Stream, Native Video, or Native Video (Vectored DE).

- **Include HDCP 1.4 Encryption:** This option enables HDCP 1.4 encryption.

- **Include HDCP 2.2 Encryption:** This option enables HDCP 2.2 encryption.

Note: HDCP Encryption options are only configurable if you have a HDCP license; otherwise it is disabled and grayed out from the GUI.

Send Feedback

- **Max bits per component:** This option selects the maximum bits per component. The allowable options are, 8, 10, 12 or 16 bits. This parameter is to set the maximum "allowed" bits per component, and the actual bits per component can be set from the software API to a different value. However, the actual bits per component is bounded by the Max bits per component. For example, if the Max bits per component is set to 16, the user can set the actual bits per component from the software API to any of the values, 8, 10, 12 or 16. But if the Max bits per component is set to 8, you can only set the actual bits per component to 8 through the software API.

- **Number of pixels per clock on Video Interface:** This option selects the number of pixels per clock. The allowable options are 2 or 4 pixels.

> **IMPORTANT!** *Pixels per clock (PPC) can only be selected at IP generation time, and must remain static in the design. Some video format with a total horizontal timing that is NOT divisible by 4 (for example, 720p60 has a total horizontal pixel of 1650, which is not divisible by 4) are not supported. If the design is intended to support resolutions not divisible by 4, ensure that PPC=2 is selected in the Vivado IDE. Custom resolutions not divisible by 2 should not be used with the subsystem.*

- **Video over AXIS compliant NTSC/PAL Support:** This option enables the HDMI 1.4/2.0 TX Subsystem to support Video over AXIS compliant NTSC/PAL.

  - A pixel repetition of 2 is supported by current hardware.

  - 480i60 and 576i50 resolutions are supported in current software.

  - Set the corresponding fields in the AVI InfoFrame so that the Pixel repetition information is sent to the HDMI sink device.

  Note: Using AXI4-Stream, Pixel Repetition only supports RGB and YUV444 color space, but not 12-bit YUV422. You can use the native interface to create customized pixel replication logic to support all color spaces including 12-bit YUV422.

- **Video over AXIS compliant YUV420 Support:** This option enables the HDMI 1.4/2.0 TX Subsystem to support Video over AXIS compliant YUV420.

> **IMPORTANT!** *For YUV420 video, the line width is doubled, therefore, all the horizontal resolution fields must be divided by 2 to be supported in the HDMI 1.4/2.0 TX Subsystem. For example, when PPC=2 is selected in the Vivado IDE, the total horizontal resolution must be divisible by 4 (instead of 2). In this case, because 720p60 has a total horizontal pixel of 1650, which is not divisible by 4, it is not supported for YUV420 format. Similarly, when PPC=4 is selected in Vivado, the total horizontal resolution need to be divisible by 8 (instead of 4).*
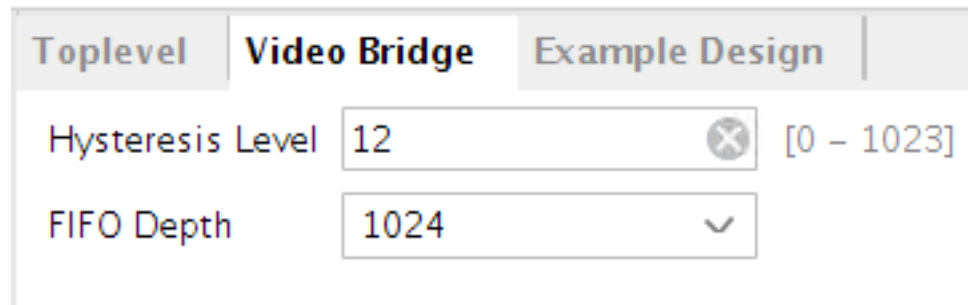
- **Hot Plug Detect Active:** This option selects the HPD active polarity. The allowable options are High or Low.

Video Bridge Tab

The Video Bridge tab is shown in the following figure.

Figure 33: **Video Bridge Tab**

The parameters on the Video Bridge tab are as follows:

- **Hysteresis Level:** Allowable range: 0—1023. Defines the "cushion" level of the frame buffer, that is, the number of locations that are considered the minimum fill level for FIFO operation to start. Generally, this value should be between 12 and 20. It must be at least 16 less than the depth of the FIFO, and at least 16 less than the number of active video lines.

- **FIFO Depth:** Specifies the number of locations in the input FIFO. The allowable values are 32, 1024, 2048, 4096, and 8192.

Example Design Tab

The Example Design tab is shown in the following figure.

Figure 34: **Example Design Tab**

- **Design Topology:** Allows you to choose the topology of example design to be generated. The allowable options are Pass-Through, TX Only, and Pass-Through+ I2S Audio (ZCU102 Only).

- **Axilite Frequency:** Allows you to choose the AXI4-Lite CPU clock. In this release, the following options have been verified.

  - 7 series: 50 MHz, 100 MHz, 150 MHz

  - UltraScale/UltraScale+ Devices: 50 MHz, 100 MHz, 150 MHz, 200 MHz

- **Video Phy Controller Setting:** Allows the configuration of the Transmitter PLL type and Receiver PLL Type to the Video PHY Controller prior generating the example design. It also allows user to selectively opt-out the NI-DRU to optimize resource use if the video resolution they plan to support doesn't require NI-DRU. See the *Video PHY Controller LogiCORE IP Product Guide* (PG230) for details about NI-DRU requirements.

- **Example Design Overview:** A system block diagram to show the overview of the example design to be generated.
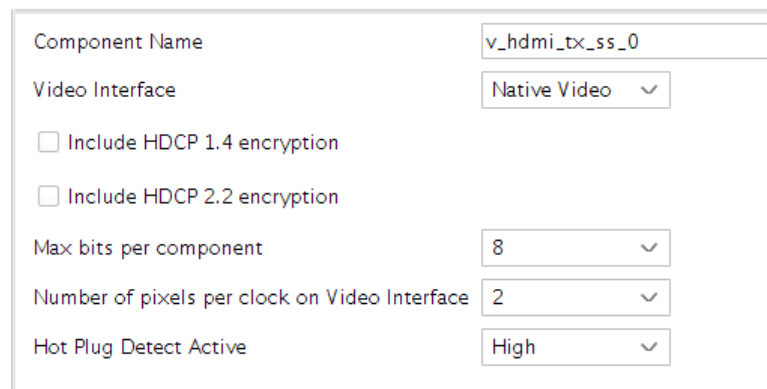
> **IMPORTANT!** *When the example design targets the VCU118 board and Design Topology is set to Pass-Through, the Include NIDRU option under the Video PHY Controller setting is grayed out and unchecked by default.*

Note: When the AXI4-Lite clock is set at a higher frequency, it is more likely to have timing violations. You must adjust the clock rate to achieve timing closure without impacting system performance.

Native Video/Native Video (Vectored DE) Interface Option

The native video interface option window is shown in the following figure.

Figure 35: **Native Video Interface Option**

- **Include HDCP 1.4 Encryption:** This option enables HDCP 1.4 encryption

- **Include HDCP 2.2 Encryption:** This option enables HDCP 2.2 encryption.

Note: HDCP 1.4 and 2.2 Encryption options are only configurable if you have a HDCP license, else it is disabled and cannot be selected from the GUI.

> **IMPORTANT!** *The Open Example Design is not supported for Native Video Interface. Therefore, the Example Design Tab is not available when Native Video is selected.*

User Parameters

The following table shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 20: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Toplevel		
Video Interface AXI4-Stream Native Video Native Video (Vectored DE)	C_VID_INTERFACE 0 1 2	AXI4-Stream
Include HDCP 1.4 Encryption Exclude (Untick) Include (Tick)	C_INCLUDE_HDCP_1_4 FALSE TRUE	Exclude
Include HDCP 2.2 Encryption Exclude (Untick) Include (Tick)	C_INCLUDE_HDCP_2_2 FALSE TRUE	Exclude
Video over AXIS compliant NTSC/PAL Support Exclude (Untick) Include (Tick)	C_INCLUDE_LOW_RESO_VID FALSE TRUE	Exclude
Video over AXIS compliant YUV420 Support Exclude (Untick) Include (Tick)	C_INCLUDE_YUV420_SUP FALSE TRUE	Exclude
Max bits per component 8 10 12 16	C_MAX_BITS_PER_COMPONENT 8 10 12 16	8
Number of pixels per clock on Video Interface 2 4	C_INPUT_PIXELS_PER_CLOCK 2 4	2
Hot Plug Detect Active High Low	C_HPD_INVERT High Low	High
Video Bridge		
Hysteresis Level	C_HYSTERESIS_LEVEL	12
FIFO Depth 32 1024 2048 4096 8192	C_ADDR_WIDTH 32 1024 2048 4096 8192	1024
Example Design		
Design Topology Pass-Through Tx Only Pass-Through + I2S Audio	C_EXDES_TOPOLOGY 0 1 4	0

Send Feedback

Table 20: **Vivado IDE Parameter to User Parameter Relationship** *(cont'd)*

Vivado IDE Parameter/Value	User Parameter/Value	Default Value

AXI4-Lite Frequency
    50
    100
    150
    200
	C_EXDES_AXILITE_FREQ
	    50
	    100
	    150
	    200
		100

TX PLL Type
    CPLL
    QPLL(GTXE2)
    QPLL01(GTHE3/4)
	C_EXDES_TX_PLL_SELECTION
	    0
	    3
	    6
		0 (GTXE2)
		6 (GTHE3/4)

RX PLL Type
    CPLL
    QPLL(GTXE2L)
    QPLL01(GTHE3/4)
	C_EXDES_RX_PLL_SELECTION
	    0
	    3
	    6
		3 (GTXE2)
		0 (GTHE3/4)

Include NIDRU
    Exclude (Untick)
    Include (Tick)
	C_EXDES_NIDRU
	    false
	    true
		true

Output Generation

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

Constraining the Subsystem

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

Required Constraints

There are clock frequency constraints for the `s_axi_cpu_aclk`, `s_axis_video_aclk`, `s_axis_audio_aclk`, `link_clk`, and `video_clk`. For example,

```
create_clock -name s_axi_cpu_aclk -period 10.0 [get_ports s_axi_cpu_aclk]
create_clock -name s_axis_audio_aclk -period 10.0 [get_ports
s_axis_audio_aclk]
create_clock -name link_clk -period 13.468 [get_ports link_clk]
create_clock -name video_clk -period 6.734 [get_ports video_clk]
create_clock -name s_axis_video_aclk -period 5.0 [get_ports
s_axis_video_aclk]
```

When using this subsystem in the Vivado Design Suite flow with Video PHY Controller modules, `link_clk` and `video_clk` are generated from the Video PHY Controller. Therefore, the clock constraints are set to the Video Controller constraints instead of these generated clocks. See Clocking in the *Video PHY Controller LogiCORE IP Product Guide* (PG230) for more information.

`s_axi_cpu_aclk`, `s_axis_video_aclk`, and `s_axis_audio_aclk` constraints are generated at system-level, for example by using a clock wizard.

Device, Package, and Speed Grade Selections

For more information on the device constraint/dependency, see the *Video PHY Controller LogiCORE IP Product Guide* (PG230).

The following table shows the device and speed grade selections for HDMI 1.4/2.0 TX Subsystem.

Table 21: **Device and Speed Grade Selections**

Device Family	PPC	2				4			
	BPC	8	10	12	16	8	10	12	16
	Speed Grade								
Zynq-7000 SoC	−1	HDMI 1.4[1]				HDMI 1.4[1]			
	−2	HDMI 2.0[2]				HDMI 2.0[2]			
Zynq UltraScale + MPSoC	−1	HDMI 2.0[2]				HDMI 2.0[2]			
	−2								
Artix-7	−1	HDMI 1.4[1]				HDMI 1.4[1]			
	−2	HDMI 1.4[1]				HDMI 1.4[1]			
Kintex-7	−1	HDMI 1.4[1]				HDMI 1.4[1]			
	−2	HDMI 2.0[2]				HDMI 2.0[2]			
Kintex UltraScale	−1	HDMI 2.0[2]				HDMI 2.0[2]			
	−2								
Virtex-7	−1	HDMI 1.4[1]				HDMI 2.0[2]	HDMI 1.4[1]		
	−2	HDMI 2.0[2]				HDMI 2.0[2]			
Virtex UltraScale	−1	HDMI 2.0[2]				HDMI 2.0[2]			
	−2								
Virtex UltraScale+	−1	HDMI 2.0[2]				HDMI 2.0[2]			
	−2								

Notes:

1. All HDMI 1.4 resolutions can be supported.
2. Full HDMI 2.0 resolutions support up to 4096 x 2160 @ 60fps.

Clock Frequencies

See the Clocking section for more information.

Related Information
Clocking

Clock Management

This section is not applicable for this IP subsystem.

Clock Placement

This section is not applicable for this IP subsystem.

Banking

This section is not applicable for this IP subsystem.

Transceiver Placement

This section is not applicable for this IP subsystem.

I/O Standard and Placement

This section is not applicable for this IP subsystem.

Simulation

Simulation of the subsystem is not supported.

Synthesis and Implementation

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

Example Design

This chapter contains step-by-step instructions for generating an HDMI Example Design from the HDMI 1.4/2.0 TX Subsystem by using the Vivado® Design Suite flow.

Summary

The HDMI 1.4/2.0 TX Subsystem allows users to customize the example design based on their system requirements. The following table shows a summary of the hardware required for each targeted board, supported processors, topologies, and the corresponding Vitis™ software platform import example options.

Table 22: **Example Design Support Summary**

Development Boards	Additional Hardware	Processor	Topology	Vitis Import Example
KC705/KCU105/VCU118[1]	inrevium TB-FMCH-HDMI4K FMC daughter card	MicroBlaze™	Pass-through	Passthrough_Microblaze
			TX Only	TxOnly_Microblaze
ZC706		A9	Pass-through	Passthrough_A9
			TX Only	TxOnly_A9

Send Feedback

Table 22: **Example Design Support Summary** *(cont'd)*

Development Boards	Additional Hardware	Processor	Topology	Vitis Import Example
ZCU102[2]/ZCU104/ZCU106	-	A53	Pass-through	Passthrough_A53
			TX Only	TxOnly_A53
			Pass-Through	Repeater_A53[3]
			Pass-Through + I2S Audio (ZCU102 Only)	Passthrough_Audio_I2S_A53
		R5	Pass-through	Passthrough_R5
			TX Only	TxOnly_R5

Notes:

1. For VCU118 board, no dedicated on-board GT reference clocks are available to support the HDMI Transmitter and NI-DRU on the HDMI Receiver simultaneously due to board design limitations. Therefore, if Pass-through topology is selected for the VCU118 board, NI-DRU is disabled.

2. The *HDMI + I2S-PMOD Audio* can be generated *only* using the configurations in the following table.

3. A dedicated repeater application is added to demonstrate repeater functionality. Note the following:

    The Repeater feature has been removed from standard pass-through application for all supported processors.

    This application passes repeater compliance tests (CTS) on SL8800 running on a ZCU102 A53 processor.

    The same application can also be used for other applications.

    You must increase the BRAM size from 512K to 1M in hardware IP integrator to run repeater function on the MicroBlaze™ processor. This is because that the repeater functionality is more complex and requires more resources.

Table 23: **HDMI + I2S-PMOD Audio Configuration**

Description	Value
Board	ZCU102 Revision 1.x
Video Interface	AXI4-Stream
Include HDCP 1.4 Encryption	Disabled
Include HDCP 2.2 Encryption	Disabled
Video over AXIS compliant NTSC/PAL Support	Disabled
Video over AXIS compliant YUV420 Support	Disabled
Max bits per component	8
Number of pixels per clock on Video Interface	2
Design Topology	Pass-Through + I2S Audio
Axilite Frequency	100
Tx PLL Type	QPLL0/1
Rx PLL Type	CPLL
Include NIDRU	Enabled

This chapter covers the design considerations of a High-Definition Multimedia Interface (HDMI) 2.0 implementation using the performance features of these Xilinx® subsystems and IP:

• HDMI 1.4/2.0 with HDCP 1.4/2.2 Transmitter Subsystem

- HDMI 1.4/2.0 with HDCP 1.4/2.2 Receiver Subsystem (For Pass-through topology only)
- Video PHY Controller

The design features the transmit-only and the pass-through operation modes for the HDMI solution. In the transmit-only mode, the design displays a color bar pattern from the LogiCORE IP Test Pattern Generator (TPG) core. In the pass-through mode, an external HDMI source is used to send video data over the HDMI design. The reference design demonstrates the use of the High-bandwidth Digital Content Protection System (HDCP) Revision 1.4/2.2 capability of the HDMI solution. HDCP is used to securely send audiovisual data from an HDCP protected transmitter to HDCP protected downstream receivers. Typically, HDCP 2.2 is used to encrypt content at Ultra High Definition (UHD) while HDCP 1.4 is used as a legacy encryption scheme for lower resolutions.

Hardware

The example design is built around the HDMI 1.4/2.0 Transmitter Subsystem (HDMI_TX_SS), the HDMI 1.4/2.0 Receiver Subsystem (HDMI_RX_SS) (Optional), and the Video PHY Controller (VPHY) core and leverages existing Xilinx IP cores to form the complete system. The following two figures are illustrations of the overall HDMI example design block diagram targeting various Xilinx evaluation kits.

Figure 36: **KC705/KCU105/ZC706/VCU118 HDMI Example Design Block Diagram**

Send Feedback
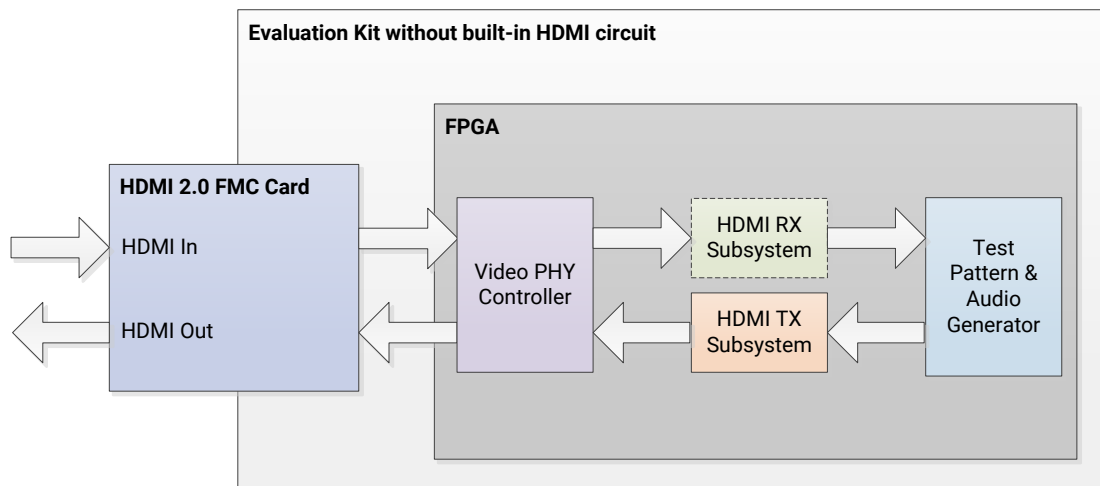
Figure 37: **ZCU102/ZCU104/ZCU106 HDMI Example Design Block Diagram**
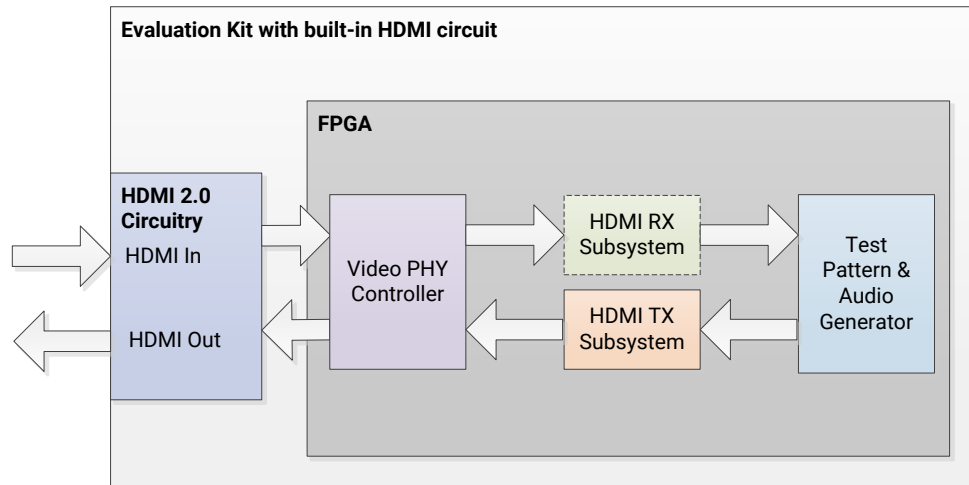
IMPORTANT! *When an unpowered HDMI source is connected to the HDMI receiver in a pass-through system, the HDMI example design UART can get flooded with a Starting Colorbar message because of a limitation of the ZCU102 board design.*

The Video PHY Controller core has been configured for the HDMI application that allows transmission and reception (optional) of HDMI video/audio to and from the HDMI 2.0 daughter card or on-board HDMI 2.0 circuitry.

Figure 38: **KC705/KCU105/ZC706/VCU118 HDMI Reference Design Clock and Datapath Diagram**
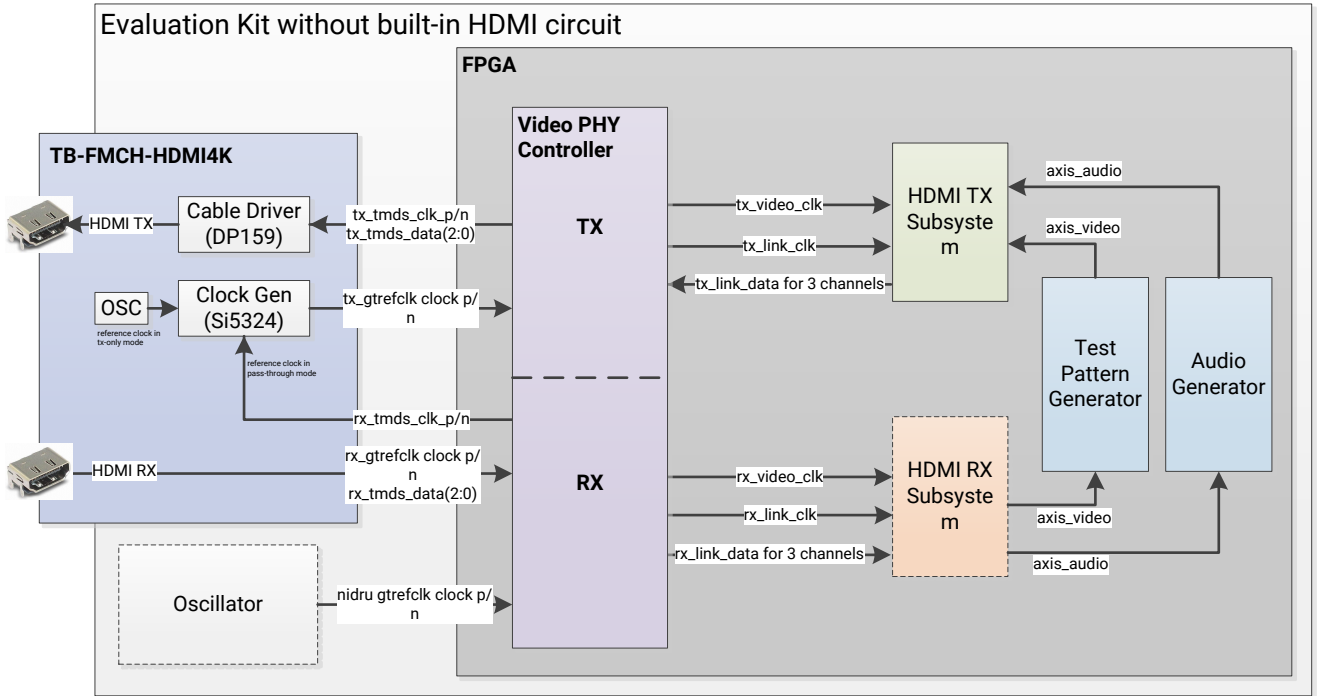
Figure 39: **ZCU102/ZCU104/ZCU106 HDMI Reference Design Clock and Datapath Diagram**
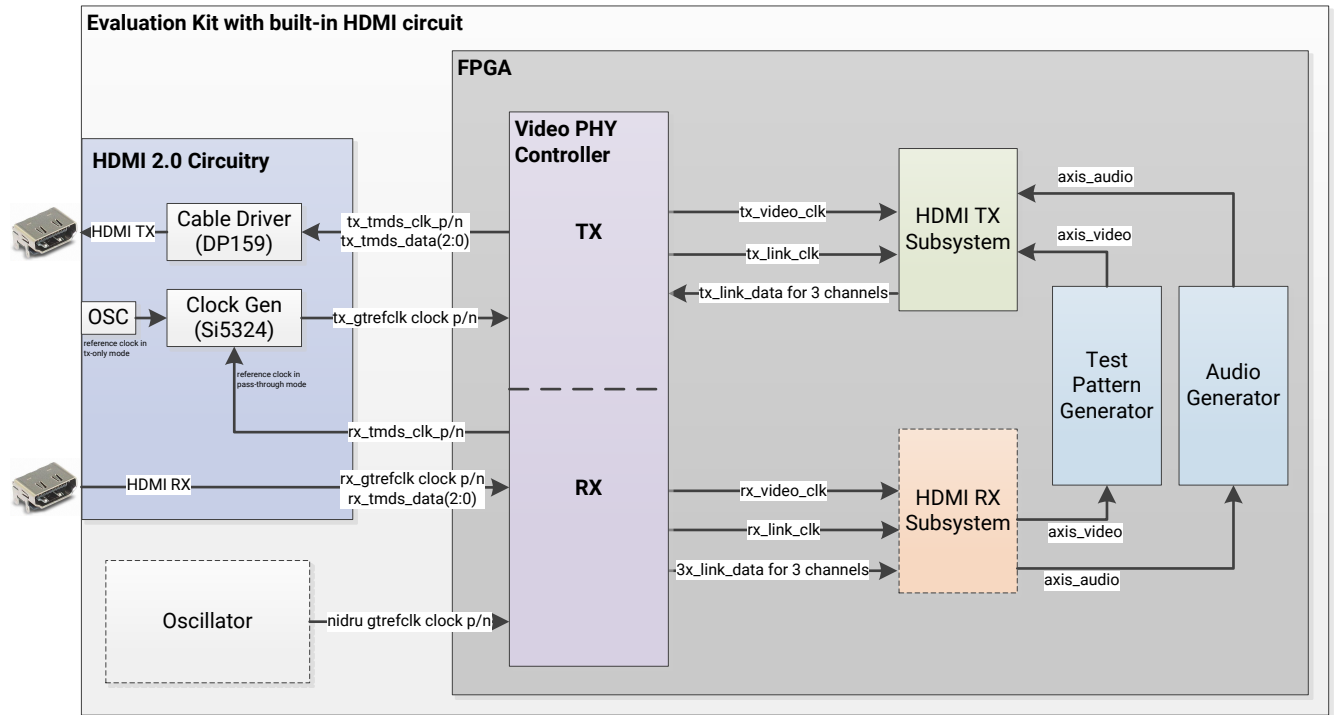
> ⭐ **IMPORTANT!** *he TI HDMI cable driver chip DP159 is used in all the example design, either on the evaluation board itself or on the inrevium TB-FMCH-HDMI4K FMC daughter card. In the HDMI example design software, a sample DP159 driver is provided as reference. The settings have not been calibrated for various use cases. If you are using the DP159 in your product, you need to adjust the settings based on the circuit design. Some further fine tuning might still be needed depending on the compliance results.*

TMDS DATA PCB trace rules required to meet HDMI compliance requirements for the TMDS181 (only when RX is used) and SN68DP159 devices are as follows.

Inter-pair skew for DATA[0:2] lanes must be:

- Max 10 ps inter-pair skew FPGA→ retimer
- Max 10 ps inter-pair skew retimer→ connector

Intra-pair skew for DATA[0:2] lanes must be:

- Max 1 ps intra-pair skew FPGA→ retimer
- Max 1 ps intra-pair skew retimer→ connector

Send Feedback

- Target impedance to be 100Ω ±7% (Max ±10%)

  ○ A single excursion is permitted out to a max/min of 100Ω ±25% and of a duration less than 250 ps

In pass-through mode, the Video PHY Controller core receives the high-speed serial video stream, converts it to parallel data streams, forwards it to the HDMI_RX_SS core, which extracts the video and audio streams from the HDMI stream and converts it to separate AXI video and audio streams. The AXI video goes through the TPG core and the AXI audio goes through a customized audio generation block. The two AXI streams eventually reach the HDMI_TX_SS core, which converts the AXI video and audio streams back to an HDMI stream before being transmitted by the Video PHY Controller core as a high-speed serial data stream. The transition minimized differential signaling (TMDS) clock from the HDMI In interface is forwarded to the HDMI TX transceiver through the SI53xx clock generator in the HDMI 2.0 FMC card or on-board HDMI 2.0 circuitry.

Note: The ZCU104 uses a different clock generator, the IDT 8T49N24x. A sample driver is provided as part of example application software. It is not calibrated for the best performance to pass compliance. You might need to fine tune its settings in your design if you are using the same chip.

In TX-only mode, the colorbar pattern is generated by the TPG as an AXI video stream and the low frequency audio is generated by the customized audio processing block as an AXI audio stream. The two streams are forwarded to the HDMI_TX_SS for HDMI stream conversion and then to the Video PHY Controller for transmission.

High-level control of the system is provided by a simplified embedded processor subsystem containing I/O peripherals and processor support IP. A clock generator block and a processor system reset block supply clock and reset signals for the system, respectively. See the following two figures for block diagrams of the three types of processor subsystems supported by the HDMI example design flow.

Figure 40: **HDMI Reference Design Block Diagram (MicroBlaze)**
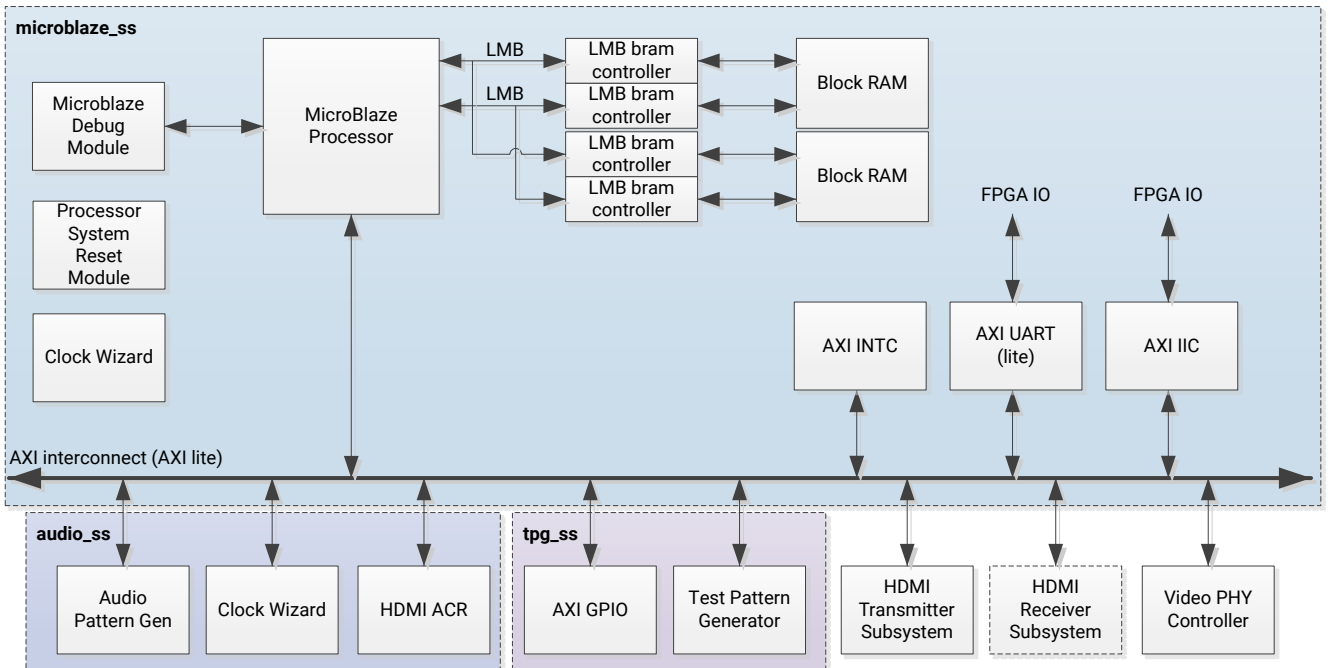
Figure 41: **HDMI Reference Design Block Diagram (Zynq or Zynq UltraScale+ MPSoC)**
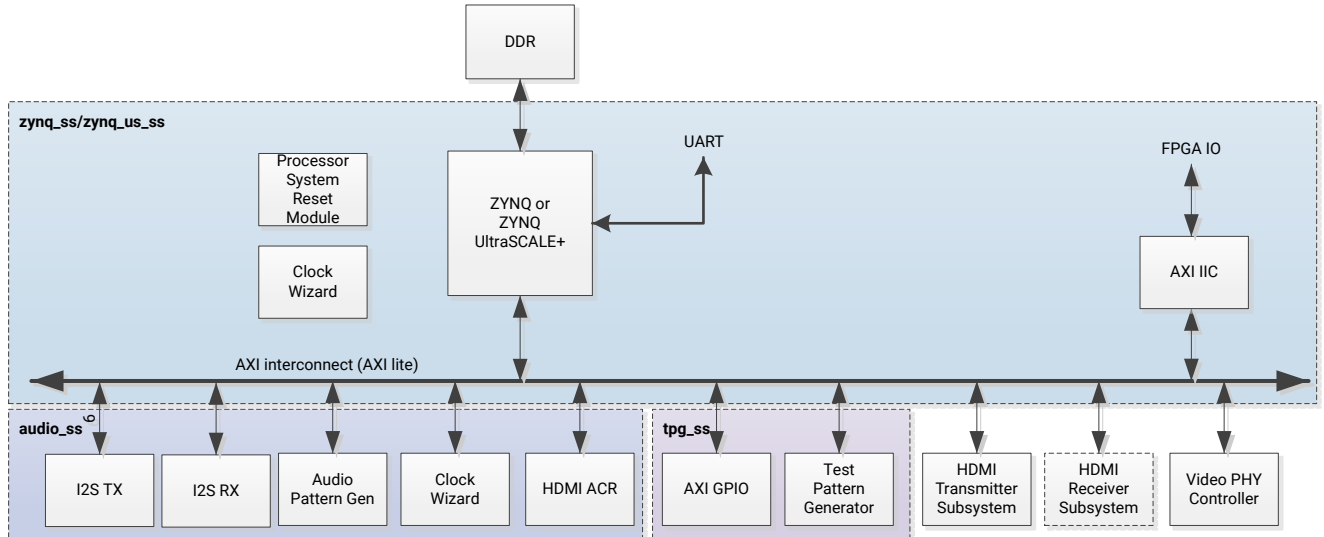
Example Design Specifics

In addition to the Video PHY Controller, HDMI Transmitter Subsystem and HDMI Receiver Subsystem core, the complete example design includes the following cores:

- MicroBlaze or Zynqor Zynq UltraScale+ MPSoC

Send Feedback

- MicroBlaze Debug Module (Only for MicroBlaze based processor subsystem)

- AXI Interconnect

- Local Memory Bus (LMB) (Only for MicroBlaze based processor subsystem)

- LMB BRAM Interface Controller (Only for MicroBlaze based processor subsystem)

- Block Memory Generator (Only for MicroBlaze based processor subsystem)

- Clocking Wizard

- Processor System Reset Module

- AXI UART Lite (Only for MicroBlaze based processor subsystem)

- AXI Interrupt Controller (INTC) (Only for MicroBlaze based processor subsystem)

- AXI IIC Bus Interface

- AXI GPIO

- Video Test Pattern Generator

- AXI4-Stream Register Slice

- Utility Buffer

- Utility Vector Logic

- AUD_PAT_GEN (Custom IP)

- HDMI_ACR_CTRL (Custom IP)

- HDCP_KEYMNGMT_BLK (Custom IP)

  Note: When a Custom IP is added to an IP integrator design as RTL reference module, the auto assigned address can be maximized to any of the available space. Therefore, if you unmap AUD_PAT_GEN, HDMI_ACR_CTRL or HDCP_KEYMNGMT_BLK for the Example design, after reassigning the address, you must set the address to a smaller range (for example, 64k).

Running the Example Design

1. Open the Vivado Design Suite and create a new project.

2. In the pop-up window, press Next until you get to the page to select the Xilinx part or board for the project.
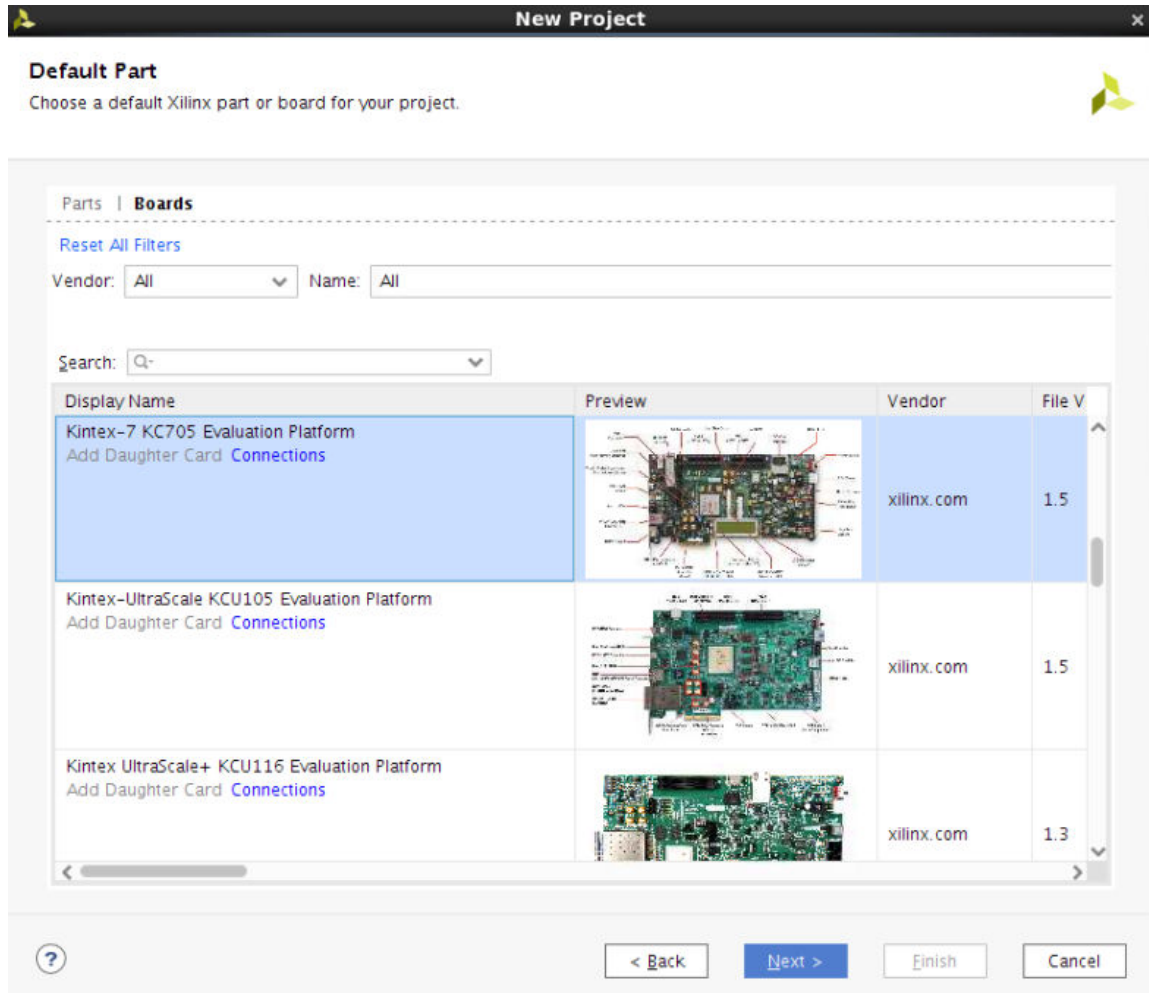
3. Select the target board, then click **Next→Finish**. (KC705, KCU105, ZC706, ZCU102, ZCU104, ZCU106, and VCU118 are supported.)

4. A Vivado Project opens. In Flow Navigator, PROJECT MANAGER, click **IP Catalog** to open it. Then double-click HDMI 1.4/2.0 Transmitter Subsystem in Video Connectivity.

Send Feedback
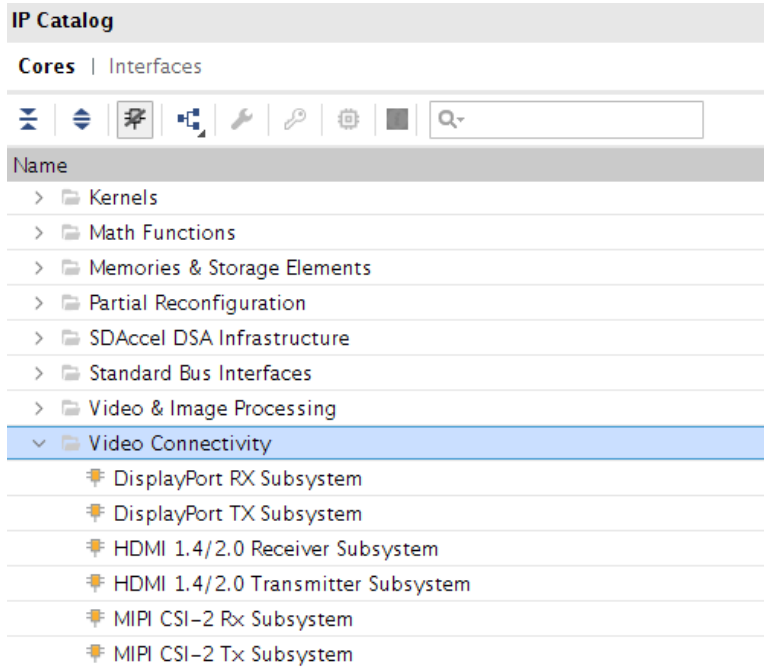
5. A Customize IP window opens. Configure the HDMI 1.4/2.0 TX Subsystem, then select OK.

   a. Refer to the Design Flow Steps chapter for a detailed description on Customizing and Generating the Subsystem.

   b. You can rename the IP component name, which is used as example design project name.

   c. The native video interface, including the native video (Vectored DE) interface, is *not* supported in the example design flow.

Send Feedback
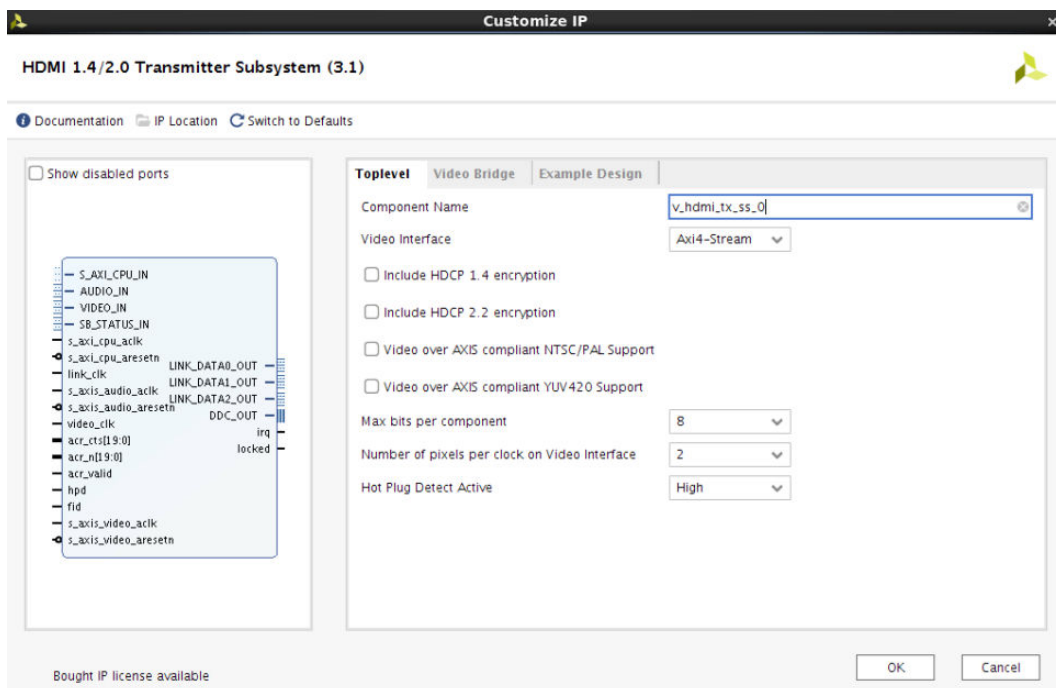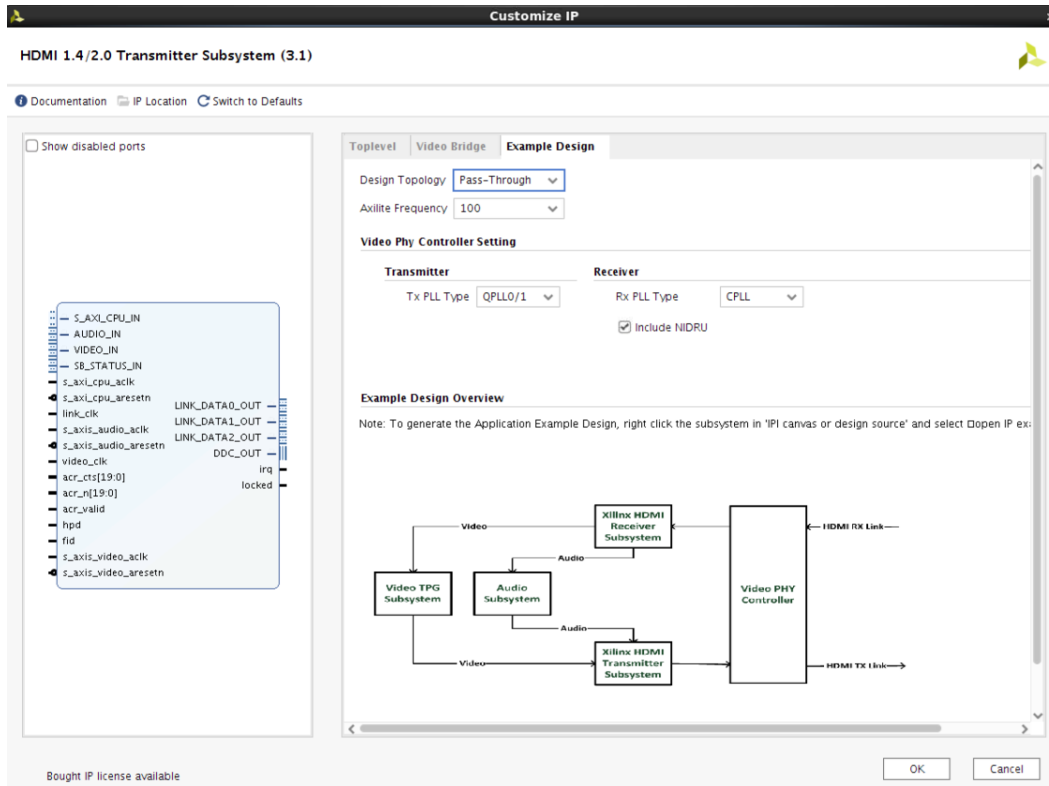
6. The Generate Output Products dialog box opens. Select Generate.

   Note: You can optionally select Skip if you only want to create an example design and leave the IP generation to a later stage.
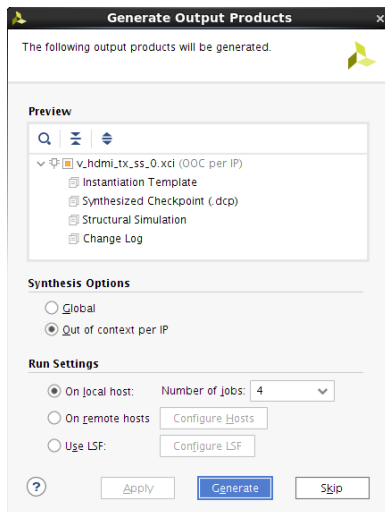
7. The IP component with provided name is added to Design Sources. Right click on it and select **Open IP Example Design**.
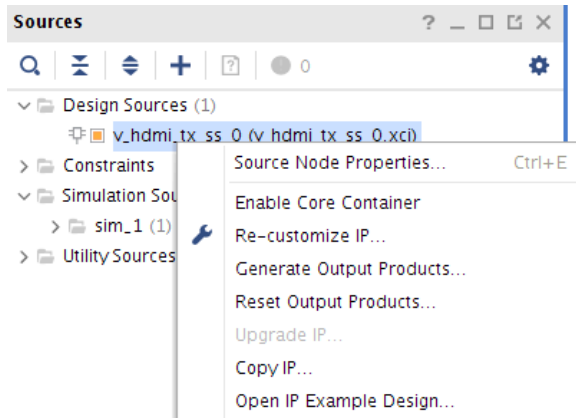
8. Choose the target Example project directory, then select OK.

9. A new Vivado project launches, in which an HDMI Example Design is generated with Block Design to show the system structure. Select **Run Synthesis, Implementation, and Generate Bitstream** to build the design. An overall system IP integrator block diagram of the KC705-based example design is shown:
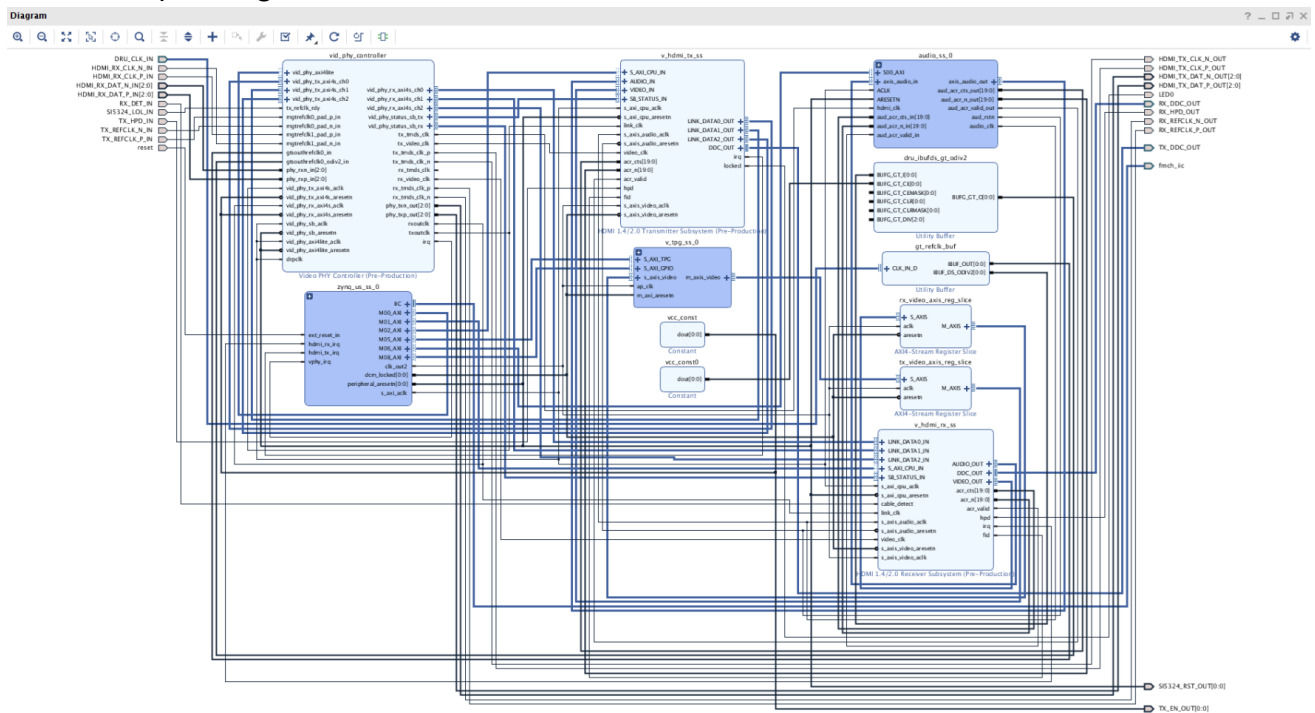
Note: If you close the block design and re-open it again, then re-run Validate Design, the following warning message, which does not have any functionality impact, is shown in the console.

```
WARNING: [BD 41-1731] Type mismatch between connected pins: /v_hdmi_tx/
s_axis_video_aresetn_out(undef) and /v_axi4s_vid_out/aresetn(rst)
```

Related Information

[Customizing and Generating the Subsystem](#)

Export the Generated Example Design

After the hardware is created successfully, the next step is to export the generated the HDMI example design hardware to build the HDMI application software using the Xilinx Vitis Integrated Design Environment (IDE).

1. To export the hardware design, in the generated HDMI example design Vivado project, select **File → Export → Export Hardware**.
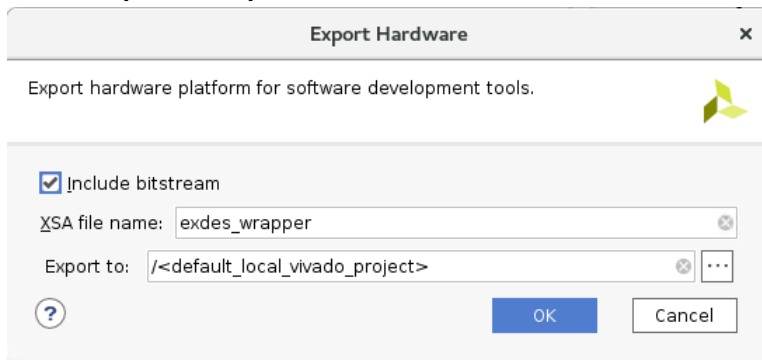
2. In the Export Hardware window, select OK. The hardware definition file is exported to a folder (usually the Vivado project root directory).

3. Create a directory for the Vitis workspace on the Vivado Tcl console.

```
mkdir <vitis_workspace>
```

4. Go to the `<vitis_workspace>` directory and execute the Vitis IDE on the Vivado Tcl console.

```
cd <vitis_workspace>
vitis &
```

Build Software Application Using Vitis IDE

1. Open the Vitis IDE.

2. Select **File → New → Platform Project** to create a new platform project.
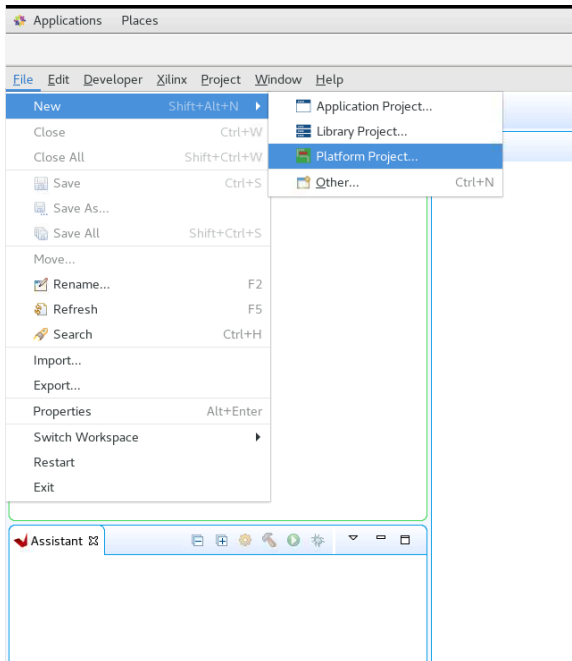
3. Set the desired Project Name and select **Next**.

www.xilinx.com
81
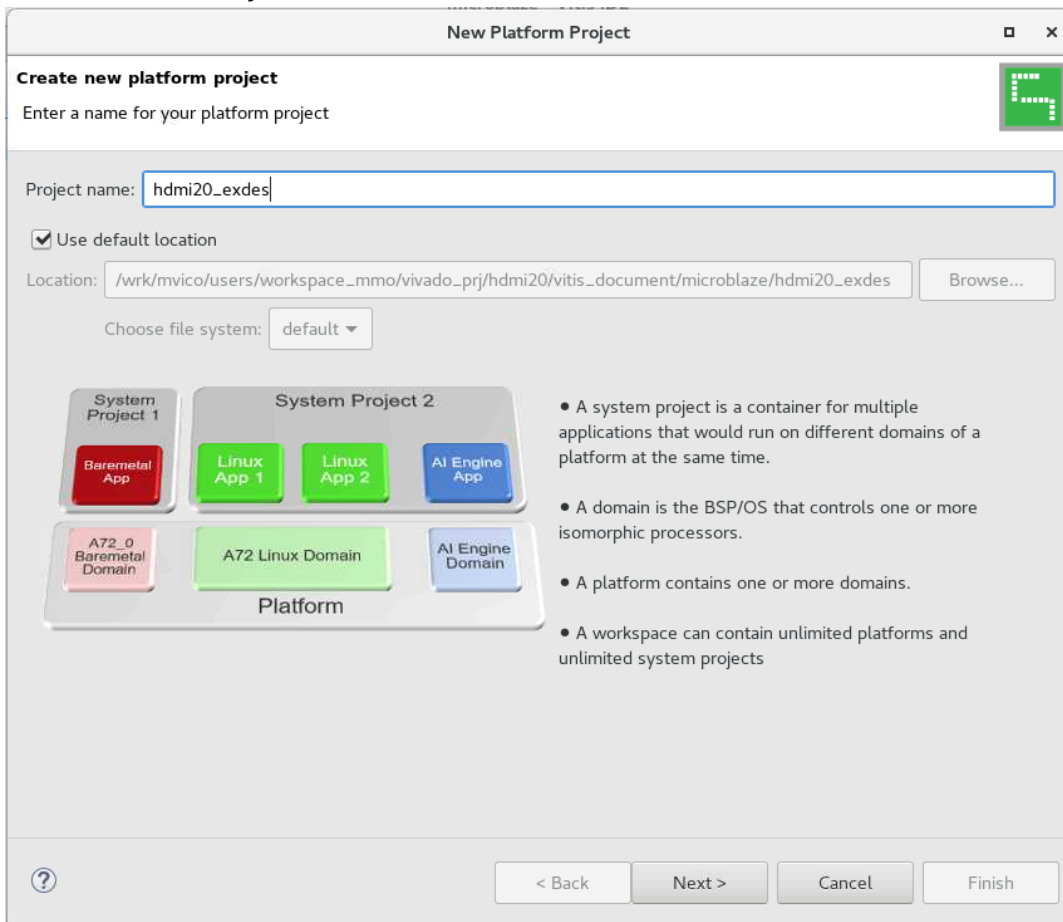
4. Select Create from hardware specification (XSA) and select **Next**.
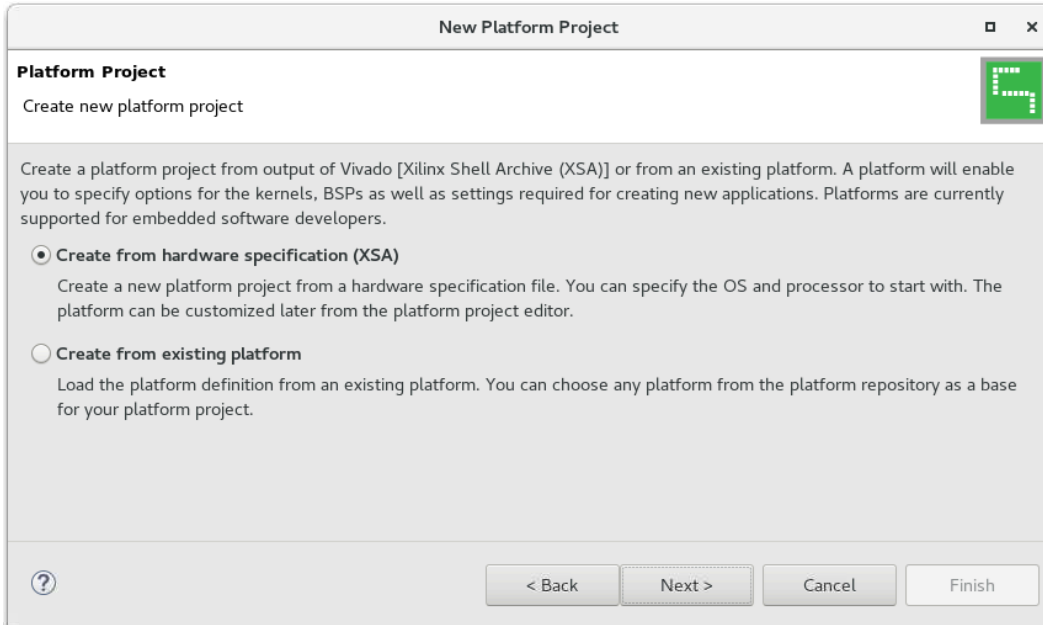
5. Select the generated XSA from the HDMI 2.0 Example Design, and select **Finish**.
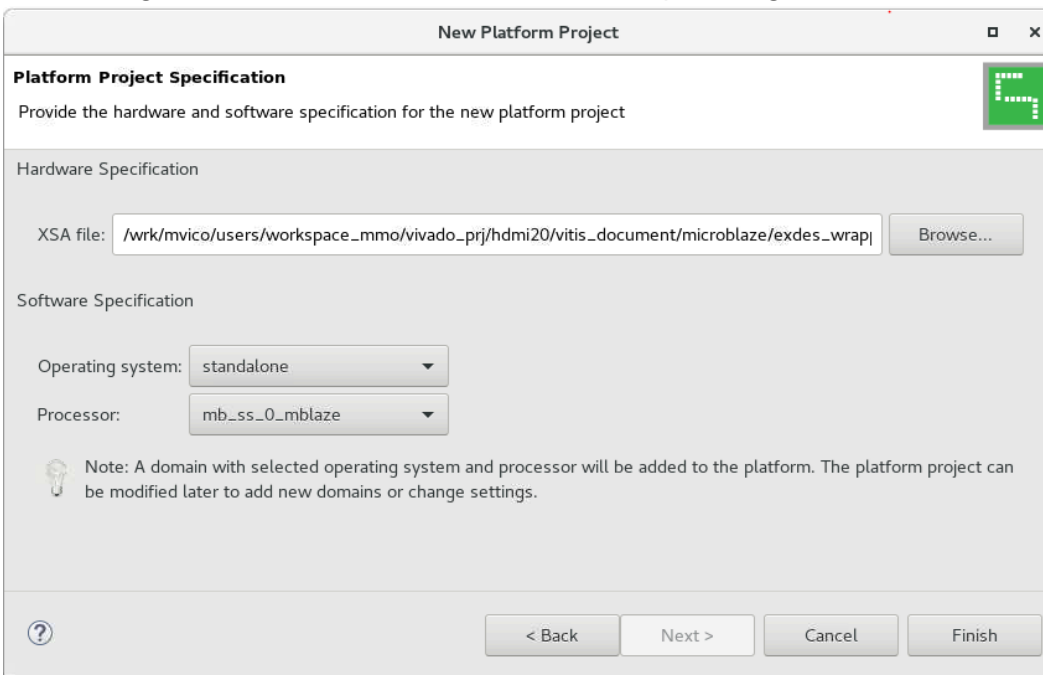
6. Build/Generate the platform - Right-click Platform and select **Build Project**.
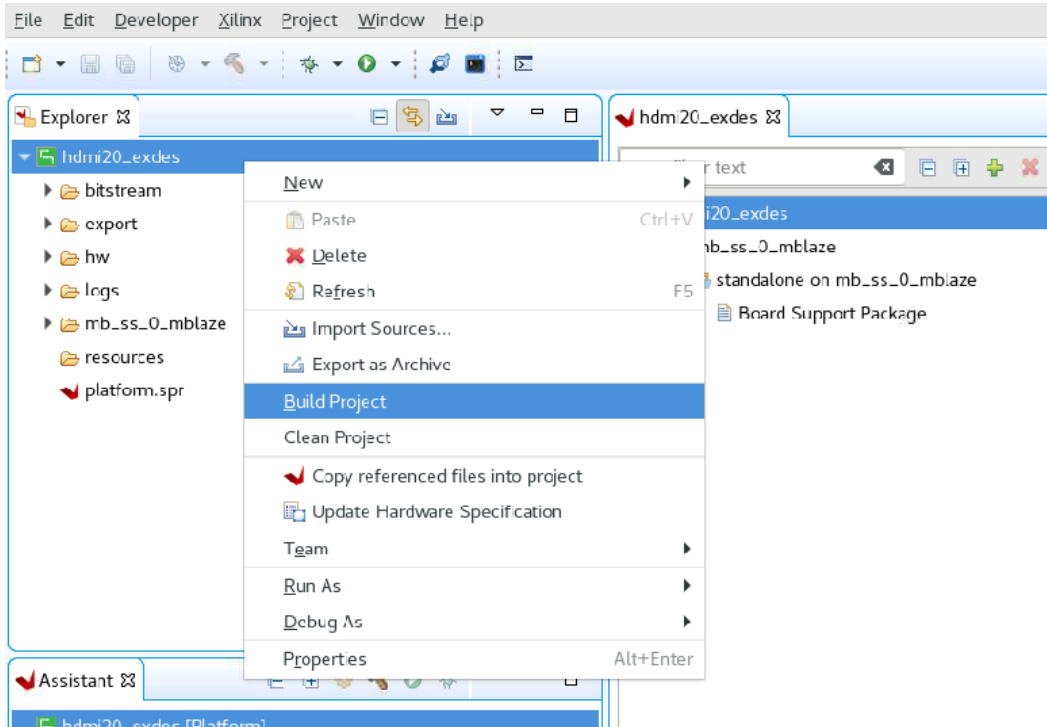
7. At the platform.spr page, Select **Board Support Package** under the standalone, and expand the Peripheral Drivers.
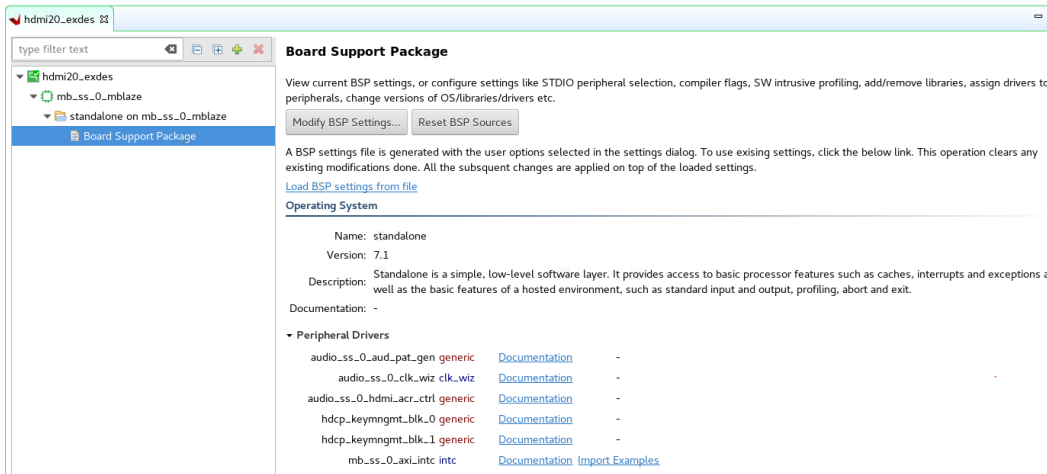
8. Find the v_hdmi_rx_ss or v_hdmi_tx_ss, and select **Import Examples**.

9. An Import Examples window opens with a list of applications. See the screenshot and select the respective application and click **OK**. See the Example Design Support Summary table in the Summary topic to select the respective application.
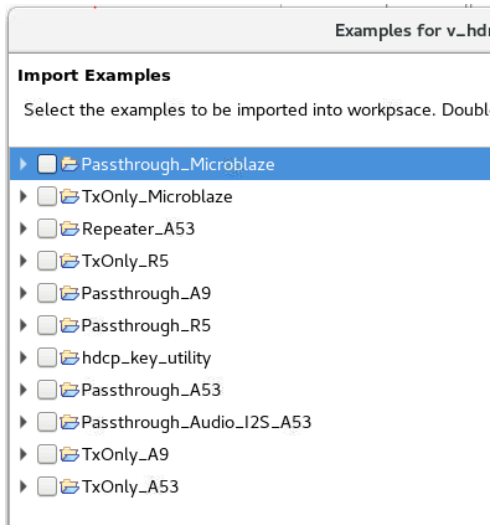
10. Build the application.
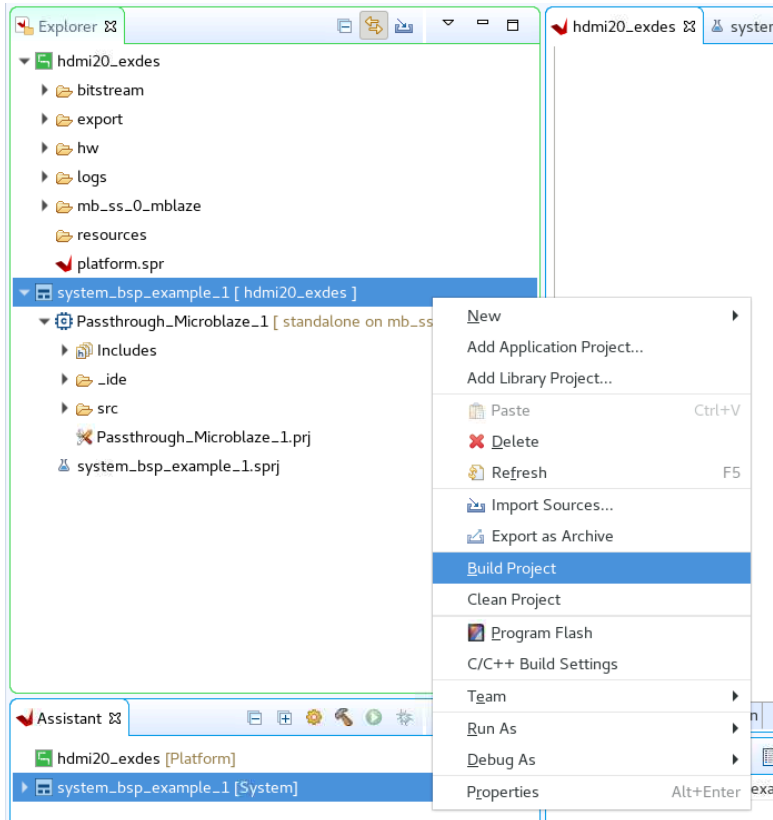
The Example Application will be built and a `.elf` file is ready to use.

Related Information
Summary

HDCP Key Utility

An optional hdcp_key_utility application software is available for using the same hardware to program your own HDCP encryption keys into the EEPROM (FMC or on-board).

To hdcp_key_utility application:

1. Import Example from the Vitis software platform and choose hdcp_key_utility.

2. Open `hdcp_key_utility.c` from the Vitis project.

3. The arrays Hdcp22Lc128, Hdcp22Key, Hdcp14Key1, and Hdcp14Key2 hold the HDCP keys and are empty. Fill these arrays with the acquired HDCP keys. The arrays are defined in big endian byte order.

4. Save the file and compile the design.

5. Run the design.

The terminal displays the following output:

```
HDCP Key EEPROM v1.0
This tool encrypts and stores the HDCP 2.2 certificate and HDCP 1.4 keys
into the EEPROM on the HDMI FMC board
Enter Password ->
```

The HDCP keys are encrypted using this password. The same password is used in the reference design to decrypt the HDCP keys.

The application is terminated after completing the programming of HDCP keys.

Note: The keys only need to be programmed into the EEPROM once.

Formatting HDCP Keys for HDCP 1.x

The `hdcp_key_utility.c` has two (empty) HDCP 1.x key arrays.

- The Hdcp14Key1 array

  This array holds the HDCP 1.x RX KSV and Keys.

- The Hdcp14Key2 array.

  This array holds the HDCP 1.x RX KSV and Keys.

The arrays have a size of 328 bytes and contain the Key Selection Vector (KSV) (5 bytes padded with zeros to 8 bytes) and key set (320 bytes), where each key is 7 bytes padded with zeros to 8 bytes.

To format the HDCP 1.x keys for the key_utility, use the followiing steps:

1. Discard the 20 byte SHA-1.

2. Pad each key on the right with one byte of 0s (KSV is already padded).

   You should now have 1 x 8 byte KSV + 40 x 8 byte Keys.

3. Byte swap each 8 byte set to reverse their order (convert from Little-endian to Big-endian).

   Note: The facsimile keys given in HDCP 1.4 spec are already in little-endian format, so byte swap is not needed when using them for test purpose.

   The final result should be a 328 byte HDCP 1.4 keyset.

Formatting HDCP Keys for HDCP 2.2

The `hdcp_key_utllity.c` has two (empty) HDCP 2.2 key arrays.

- The `hdcp22_lc128` array

  The global secret constant for the HDCP 2.2 TX. The size is 16 bytes and the license constant from the HDCP 2.2 TX certificate is placed into the array (position 4-19).

- The `Hdcp22RxPrivateKey` array

  This array holds the HDCP 2.2 RX certificate. The array has a size of 902 bytes. The contents are the header (4), license constant (36 bytes) and key set (862), so the total is 902 bytes.

  Note: If a pass-through example design is built which contains an HDMI RX component, you can use the same `hdcp_key_utility` application to program the HDCP 2.2 RX Private Key.

Running the Reference Design (MicroBlaze)

Use the following steps to execute the system using generated bitstream and software elf from the example design.

1. Launch the Xilinx System Debugger by selecting **Start → All Programs → Xilinx Design Tools → Vivado 2019.2 → Vivado 2019.2 Tcl Shell**.

2. In the Xilinx command shell window, change to the Example Design Project directory:

   ```
   Vivado% cd ./<IP instance name>_ex
   ```

3. Invoke Xilinx System Debugger (xsdb).

   ```
   Vivado% xsdb
   ```

4. Establish connections to debug targets.

   ```
   xsdb% connect
   ```

5. Download the bitstream to the FPGA.

   ```
   xsdb% fpga -file ./<IP instance name>_ex.runs/impl_1/exdes_wrapper.bit
   ```

6. Set the target processor.

   ```
   xsdb% target
     1* xcvu9p
     2  MicroBlaze Debug Module at USER2
     3  MicroBlaze #0 (Running)
   xsdb% target -set 3
   ```

7. Download the software `.elf` to the FPGA.

   ```
   xsdb% dow ./<vitis_workspace>/<application_name>_1/Debug/
   <application_name>_1.elf
   ```

8. Run the software.

   ```
   xsdb% stop
   xsdb% rst
   xsdb% con
   ```

9. Exit the XSDB command prompt.

   ```
   xsdb% exit
   ```

Running the Reference Design (A53 on Zynq UltraScale+ Devices)

1. Launch the Xilinx System Debugger by selecting **Start → All Programs → Xilinx Design Tools → Vivado 2019.2 → Vivado 2019.2 Tcl Shell**.

2. In the Xilinx command shell window, change to the Example Design Project directory:

```
Vivado% cd ./<IP instance name>_ex
```

3. Invoke Xilinx System Debugger (xsdb).

```
Vivado% xsdb
```

4. Establish connections to debug targets.

```
xsdb% connect
```

5. List all available JTAG targets.

```
xsdb% targets
  1 PS TAP
  2 PMU
  3 PL
  4 PSU
  5 RPU (Reset)
  6 Cortex-R5 #0 (RPU Reset)
  7 Cortex-R5 #1 (RPU Reset)
  8 APU
  9* Cortex-A53 #0 (Running)
  10 Cortex-A53 #1 (Power On Reset)
  11 Cortex-A53 #2 (Power On Reset)
  12 Cortex-A53 #3 (Power On Reset)
```

Note: The target number for PSU, PL, APU and Cortex™-A53 might be different. Run the targets and ensure they are using the correct target number.

6. Download the bitstream to the FPGA.

```
xsdb% source <Vitis Install folder>/scripts/vitis/util/zynqmp_utils.tcl
xsdb% targets -set 4 (PSU)
xsdb% rst -system
xsdb% after 3000
xsdb% target -set 3 (PL)
xsdb% fpga -file ./<IP instance name>_ex.runs/impl_1/exdes_wrapper.bit
```

7. Set the target processor.

```
xsdb% target -set 8 (APU)
xsdb% source ./<vitis_workspace>/<platform_name>/hw/psu_init.tcl
xsdb% psu_init
xsdb% after 1000
xsdb% psu_ps_pl_isolation_removal
xsdb% after 1000
xsdb% psu_ps_pl_reset_config
xsdb% catch {psu_protection}
xsdb% target -set 9 (Cortex-A53 #0)
xsdb% rst -processor
```

Send Feedback

8. Download the software `.elf` to the FPGA.

```
xsdb% dow ./<vitis_workspace>/<application_name>_1/Debug/
<application_name>_1.elf
```

9. Run the software.

```
xsdb% con
```

10. Exit the XSDB command prompt.

```
xsdb% exit
```

Running the Reference Design (R5 on Zynq UltraScale+ Devices)

1. Launch the Xilinx System Debugger by selecting **Start → All Programs → Xilinx Design Tools → Vivado 2019.2 → Vivado 2019.2 Tcl Shell**.

2. In the Xilinx command shell window, change to the Example Design Project directory.

```
Vivado% cd ./<IP instance name>_ex
```

3. Invoke Xilinx System Debugger (xsdb).

```
Vivado% xsdb
```

4. Establish connections to debug targets.

```
xsdb% connect
```

5. List all available JTAG targets.

```
xsdb% targets
  1 PS TAP
  2 PMU
  3 PL
  4 PSU
  5 RPU (Reset)
  6 *Cortex-R5 #0 (RPU Reset)
  7 Cortex-R5 #1 (RPU Reset)
  8 APU
  9 Cortex-A53 #0 (Running)
  10 Cortex-A53 #1 (Power On Reset)
  11 Cortex-A53 #2 (Power On Reset)
  12 Cortex-A53 #3 (Power On Reset)
```

Note: The target number for the PSU, PL, RPU, and Cortex-R5F might be different. Run the targets and ensure they are using the correct target number.

6. Download the bitstream to the FPGA.

```
xsdb% source <Vitis Install folder>/scripts/vitis/util/zynqmp_utils.tcl
xsdb% targets -set 4 (PSU)
xsdb% rst -system
xsdb% after 3000
xsdb% target -set 3 (PL)
xsdb% fpga -file ./<IP instance name>_ex.runs/impl_1/exdes_wrapper.bit
```

7. Set the target processor.

```
xsdb% target -set 5 (RPU)
xsdb% source ./<vitis_workspace>/<platform_name>/hw/psu_init.tcl
xsdb% psu_init
xsdb% after 1000
xsdb% psu_ps_pl_isolation_removal
xsdb% after 1000
xsdb% psu_ps_pl_reset_config
xsdb% catch {psu_protection}
xsdb% target -set 6 (Cortex-R5 #0)
xsdb% rst -processor
```

8. Download the software `.elf` to the FPGA.

```
xsdb% dow ./<vitis_workspace>/<application_name>_1/Debug/
<application_name>_1.elf
```

9. Run the software.

```
xsdb% con
```

10. Exit the XSDB command prompt.

```
xsdb% exit
```

Running the Reference Design (A9 on Zynq)

1. Launch the Xilinx System Debugger by selecting **Start → All Programs → Xilinx Design Tools → Vivado 2019.2 → Vivado 2019.2 Tcl Shell**.

2. In the Xilinx command shell window, change to the Example Design Project directory.

```
Vivado% cd ./<IP instance name>_ex
```

3. Invoke Xilinx System Debugger (xsdb).

```
Vivado% xsdb
```

4. Establish connections to debug targets.

```
xsdb% connect
```

Send Feedback

5. List all available JTAG targets.

```
xsdb% targets
   1 APU
   2* ARM Cortex-A9 MPCore #0 (Suspended)
   3 ARM Cortex-A9 MPCore #1 (Suspended)
   4 xc7z045
```

6. Download the bitstream to the FPGA.

```
xsdb% source <Vitis Install folder>/scripts/vitis/util/zynqutils.tcl
xsdb% targets -set 1 (APU)
xsdb% rst -system
xsdb% after 3000
xsdb% target -set 4 (xc7z045)
xsdb% fpga -file ./<IP instance name>_ex.runs/impl_1/exdes_wrapper.bit
```

7. Set the target processor.

```
xsdb% targets -set 1 (APU)
xsdb% loadhw ./<vitis_workspace>/<platform_name>/hw/exdes_wrapper.xsa
xsdb% targets -set 1 (APU)
xsdb% ps7_init
xsdb% ps7_post_config
xsdb% targets -set 2 (ARM Cortex-A9 MPCore #0)
```

8. Download the software `.elf` to the FPGA.

```
xsdb% dow ./<vitis_workspace>/<application_name>_1/Debug/
<application_name>_1.elf
```

9. Run the software.

```
xsdb% con
```

10. Exit the XSDB command prompt.

```
xsdb% exit
```

⭐ **IMPORTANT!** *When using the TB-FMCH-HDMI4K example design with the KCU105 board, you must set the FMC VADJ_1V8 Power Rail before programing the FPGA with the bitstream generated in the Example Design flow. The following topic shows how to set the VADJ power rail when using the KCU105 board. For more details about the KCU105 board, see the KCU105 Board User Guide (UG917).*

KCU105 Board FMCH VADJ Adjustment

The KCU105 board system controller must apply power to the VADJ power rail for the HDMI 2.0 FMC card (TB-FMCH-HDMI4K). Most new boards are per-programmed and should be detected. The VADJ is powered when the DS19 LED (located near the power switch on the KCU105 board) is ON.

If an older version KCU105 board is used, or the board is not properly programmed upon receiving, you must manually set the VADJ power rail to 1.8V for the HDMI 2.0 FMC card prior to bitstream configuration.

Perform these steps to set the VADJ power rail through the UART terminal are:

1. Connect a USB cable between the USB UART connector of the KCU105 board and a PC running Windows.

2. Use the Windows Device Manager to determine which virtual COM port is assigned to the UART for the Zynq®-7000 SoC system controller and which is assigned to the UART for the UltraScale™ device. In the list of COM ports in the Device Manager window, the enhanced COM port associated with the CP210x, is the one connected to the KCU105 board system controller and the standard COM port is the one connected to the device UART.

3. Open a terminal window (115200, 8, N, 1) and set the COM port to the one communicating with the KCU105 board system controller.

4. After the UART terminal is connected, power cycle the KCU105 board to refresh the system controller menu in the UART terminal. Select this option in the system controller menu:

   a. Adjust FPGA Mezzanine Card (FMC) Settings.

5. In the next menu, select:

   a. Set FMC VADJ to 1.8V.
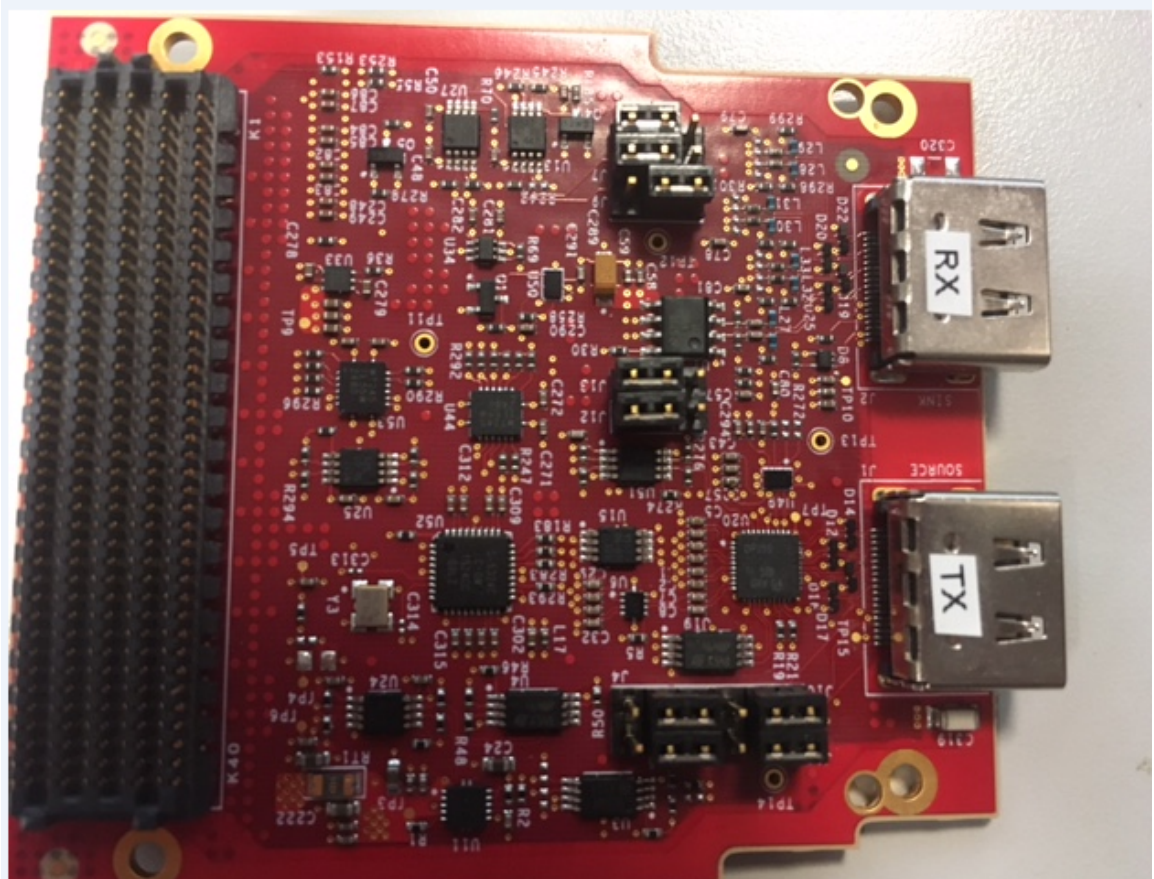
IMPORTANT! *Ensure that the jumpers are set correctly on their HDMI 2.0 FMC daughter card.*

Verification, Compliance, and Interoperability

Interoperability

Interoperability tests for the HDMI 1.4/2.0 TX Subsystem have been conducted with the following hardware setup.

Hardware Testing

The HDMI 1.4/2.0 TX Subsystem has been validated using

- Kintex®-7 FPGA Evaluation Kit (KC705)
- Kintex UltraScale FPGA Evaluation Kit (KCU105)
- Inrevium Artix®-7 FPGA ACDC A7 Evaluation Board
- Zynq-7000 SoC evaluation board (ZC706)
- Zynq® UltraScale+™ MPSoC ZCU102 Evaluation Kit
- Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit
- Zynq UltraScale+ MPSoC ZCU106 Evaluation Kit
- Virtex® UltraScale+™ FPGA VCU118 Evaluation Kit

The HDMI 1.4/2.0 TX Subsystem is tested with the following sink devices:

- Quantum Data 980B
- Quantum Data 780B
- Dell U2414Q
- Dell U2412M
- Dell U2713HM
- Acer S277HK

- Asus PQ321

- Sharp TV (LC-60LE740E)

- Philips TV (7800 series)

- Samsung UHDTV (UE40HU6900S)

- Murideo video analyzer / SIX-A

- DELL P2415Q

- Philips 288P6LJEB

- LG 27mu67

Video Resolutions

The following figure shows the hardware setup for the AXI4-Stream video Interface. An HDMI source connects to the Video PHY Controller, which converts the HDMI Video into LINK DATA and sends to the HDMI RX Subsystem. Then, the HDMI RX Subsystem translates the LINK DATA into AXI4-Stream Video and sends to the Test Pattern Generator. By setting the Test Pattern Generator to pass-through mode, the AXI4-Stream Video from the HDMI RX Subsystem is passed to HDMI TX Subsystem where it gets translated to LINK DATA again and sends back to the Video PHY Controller.The Video PHY Controller then converts it back to HDMI Video and sends to the HDMI Sink.

The Test Pattern Generator can also be configured to generate certain video pattern in the AXI4-Stream video format, which can be used to test the HDMI 1.4/2.0 TX Subsystem alone instead of relying on the video received from the RX Subsystem.

Figure 42: **Test Setup for AXI4-Stream Video Interface**
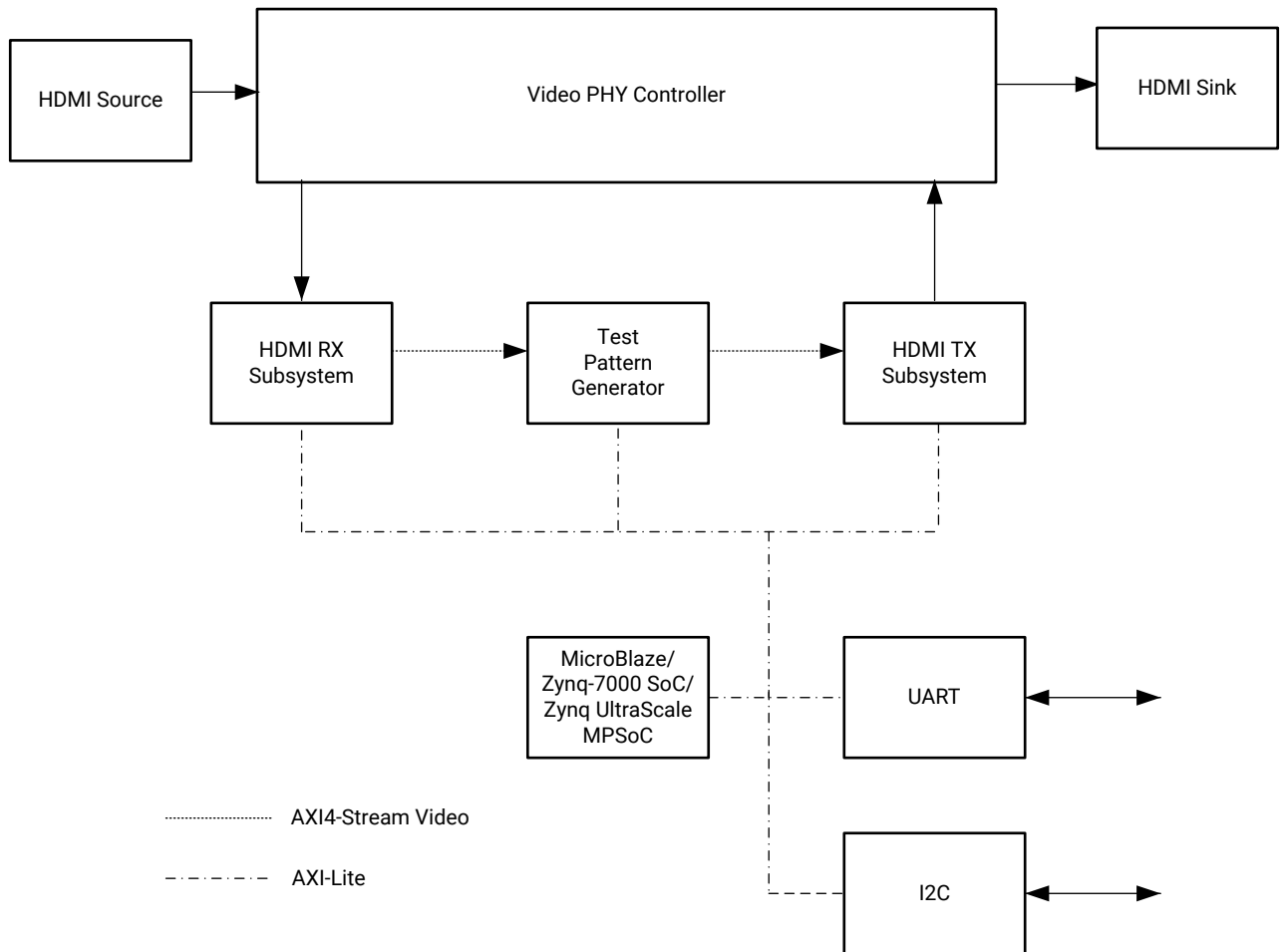
For Video PHY Controller settings and PLL selections, see the *Video PHY Controller LogiCORE IP Product Guide* (PG230).

Similarly, the following figure shows the hardware setup for the native Video Interface. The only difference is that two Video Bridge modules are added in between the HDMI RX Subsystem and the Test Pattern Generator, and between the Test Pattern Generator to the HDMI TX Subsystem.

This is because the Test Pattern Generator can be configured to generate certain video pattern in AXI4-Stream video format, which can be used to test the HDMI TX Subsystem alone instead of relying on the video received from the HDMI RX Subsystem.

Send Feedback

Figure 43: **Test Setup for Native Video Interface**
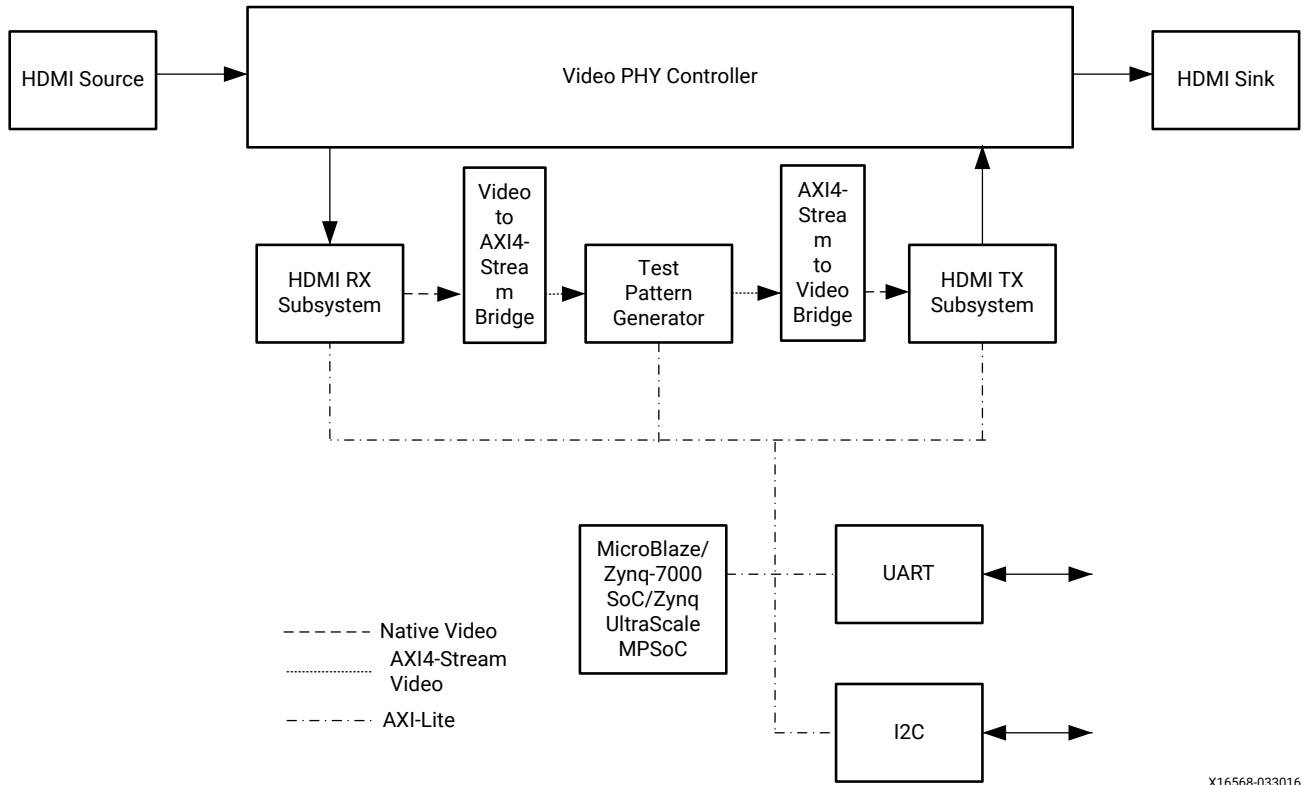
The following tables show the video resolutions that were tested as part of the release for different video formats.

Tested Video Resolutions for RGB 4:4:4 and YCbCr 4:4:4

Table 24: **Tested Video Resolutions for RGB 4:4:4 and YCbCr 4:4:4**

Resolution	Horizontal		Vertical		Frame Rate (Hz)
	Total	**Active**	**Total**	**Active**	
480i60[1]	858 (1716)	720 (1440)	525	480	60
576i50[1]	864 (1728)	720 (1440)	625	576	50
1080i50	2640	1920	1125	1080	50
1080i60	2200	1920	1125	1080	60
480p60	858	720	525	480	60
576p50	864	720	625	576	50
720p50	1980	1280	750	720	50
720p60	1650	1280	750	720	60
1080p24	2750	1920	1125	1080	24

Table 24: **Tested Video Resolutions for RGB 4:4:4 and YCbCr 4:4:4** *(cont'd)*

Resolution	Horizontal		Vertical		Frame Rate (Hz)
	Total	Active	Total	Active	
1080p25	2640	1920	1125	1080	25
1080p30	2200	1920	1125	1080	30
1080p50	2640	1920	1125	1080	50
1080p60	2200	1920	1125	1080	60
1080p120	2200	1920	1125	1080	120
2160p24	5500	3840	2250	2160	24
2160p25	5280	3840	2250	2160	25
2160p30	4400	3840	2250	2160	30
2160p60[3]	4400	3840	2250	2160	60
4096x2160p60[3]	4400	4096	2250	2160	60
vgap60	800	640	525	480	60
svgap60	1056	800	628	600	60
xgap60	1344	1024	806	768	60
sxgap60	1688	1280	1066	1024	60
wxgap60	1440	1280	790	768	60
wxga+p60	1792	1366	798	768	60
uxgap60	2160	1600	1250	1200	60
wuxgap60	2592	1920	1245	1200	60
wsxgap60	2240	1680	1089	1050	60

Notes:

1. Pixel repetition is enabled for 480i60 and 576i50. Therefore, the actual horizontal pixels are double the actual resolution.
2. Not all resolutions can be supported due to VPHY limitation. For details, refer to the *Video PHY Controller LogiCORE IP Product Guide* (PG230).
3. For 4kp60 YUV444/RGB video, only 8-bits is supported as defined by HDMI 2.0 spec due to bandwidth limitation.

Tested Video Resolutions for YCbCr 4:2:2

Table 25: **Tested Video Resolutions for YCbCr 4:2:2 at 12 Bits/Component**

Resolution	Horizontal		Vertical		Frame Rate (Hz)
	Total	Active	Total	Active	
1080i50	2640	1920	1125	1080	50
1080i60	2200	1920	1125	1080	60
480p60	858	720	525	480	60
576p50	864	720	625	576	50
720p50	1980	1280	750	720	50
720p60	1650	1280	750	720	60
1080p24	2750	1920	1125	1080	24

Table 25: **Tested Video Resolutions for YCbCr 4:2:2 at 12 Bits/Component** *(cont'd)*

Resolution	Horizontal		Vertical		Frame Rate (Hz)
	Total	Active	Total	Active	
1080p25	2640	1920	1125	1080	25
1080p30	2200	1920	1125	1080	30
1080p50	2640	1920	1125	1080	50
1080p60	2200	1920	1125	1080	60
2160p24	5500	3840	2250	2160	24
2160p25	5280	3840	2250	2160	25
2160p30	4400	3840	2250	2160	30
vgap60	800	640	525	480	60
svgap60	1056	800	628	600	60
wxgap60	1440	1280	790	768	60
wxga+p60	1792	1366	798	768	60
uxgap60	2160	1600	1250	1200	60
wuxgap60	2592	1920	1245	1200	60
wsxgap60	2240	1680	1089	1050	60

Tested Video Resolutions for YCbCr 4:2:0

Table 26: **Tested Video Resolutions for YCbCr 4:2:0 at 8, 10, 12, 16 Bits/Component in TMDS Mode**

Resolution	Horizontal		Vertical		Frame Rate (Hz)
	Total	Active	Total	Active	
2160p60	4400	3840	2250	2160	60

Supported NTSC/PAL Resolutions

Table 27: **Supported NTSC/PAL Resolutions**

	PPC	Color Space	Color Depth
480p60	2	RGB, YUV444	8,10,12,16
576p50	2, 4	RGB, YUV444	8,10,12,16
480i60	2	RGB, YUV444	8,10,12,16
576i50	2	RGB, YUV444	8,10,12,16
480p60	2	YUV422	12
576p50	2, 4	YUV422	12

Send Feedback

Upgrading

This appendix contains information about upgrading to a more recent version of the IP core.

Port Changes for 2019.2

The Native Video (Vectored DE) interface has been added. If you select this interface, the VIDEO_IN interface is replaced with individual ports. See Port Descriptions, Native Video (Vectored DE) for details.

Related Information
Native Video (Vectored DE)

Parameter Changes

The following table lists the parameter changes across the core versions.

Table 28: **Vivado IDE Parameters Changes across Versions**

Vivado IDE Parameter	v2.0	v3.0	v3.1	Note
Design Topology	Not Applicable[1]	New[3]	No Change[2]	New options in v3.0 for example design configuration. See the Customizing and Generating the Subsystem section for details.
Tx PLL Type	Not Applicable[1]	New[3]	No Change[2]	
Rx PLL Type	Not Applicable[1]	New[3]	No Change[2]	
Include NIDRU	Not Applicable[1]	New[3]	No Change[2]	
Axilite Frequency	Not Applicable[1]	Not Applicable[1]	New[3]	New option in v3.1 for example design configuration. See the Customizing and Generating the Subsystem section for details.

Notes:
1. Vivado IDE option is not available in this version.
2. Vivado IDE option remains as in previous version.
3. New Vivado IDE option added in this version.

Related Information

Customizing and Generating the Subsystem

Port Changes

The Link Data port width has changed to be configurable based on pixel per clock selection. The port width is reduced to 20 bits from the original 40 bits when user selects 2 pixels per clock.

The Link Data port width remains the default 40 bits for 4 pixels per clock.

Other Changes

N/A.

Migration Notes

When migrating from version 2016.3 or earlier, note the following:

- Hot Plug Detect Active has been added to HDMI 1.4/2.0 Transmitter Subsystem GUI.

  Choose High in the Example Design (according to board design).

- Hot Plug Detect Active has been added to HDMI 1.4/2.0 Receiver Subsystem GUI.

  Choose Low in Example Design (according to board design).

- Cable Detect Active has been added to HDMI 1.4/2.0 Receiver Subsystem GUI.

  Choose Low in Example Design (according to board design).

- HDCP 1.4/2.2 is enabled by default in Example Design application software.

  Removed UART option to Enable HDCP 1.4 or HDCP 2.2.

- Auto switching has been added to the Example Design Application software.

  You do not need to choose HDCP 1.4 or HDCP 2.2 from UART. A corresponding HDCP is selected according to the capability of connected source/sink. If the device support both HDCP 1.4 and HDCP 2.2, the priority is given to HDCP 2.2.

- HDCP repeater feature has been added.

  You can enabled/disable it by selecting **h** from UART menu.

- System log is moved from direct UART printout to event log.

Send Feedback

You can display the event log by selecting **z** from UART menu.

Send Feedback

Debugging

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

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 an error. License checkpoints are enforced by the following tools:

- Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT! *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Finding Help on Xilinx.com

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

Documentation

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

Answer Records

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

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

- Product name
- Tool message(s)
- Summary of the issue encountered

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

Master AR for HDMI 1.4/2.0 TX Subsystem

AR: 65911

Technical Support

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

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

To ask questions, navigate to the Xilinx Community Forums.

Debug Tools

There are many tools available to address HDMI 1.4/2.0 TX Subsystem design issues. It is important to know which tools are useful for debugging various situations.

Send Feedback

Vivado Design Suite Debug Feature

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

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

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

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

Reference Boards

Various Xilinx development boards support the HDMI 1.4/2.0 TX Subsystem. These boards can be used to prototype designs and establish that the subsystem can communicate with the system.

- 7 series FPGA evaluation board
  - KC705
- UltraScale FPGA evaluation board
  - KCU105
- Zynq-7000 SoC evaluation board
  - ZC706
- UltraScale+ FPGA evaluation board
  - ZCU102
  - ZCU104
  - ZCU106
  - VCU118

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 and all other constraints were met during implementation.

- 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.

    ○ User LEDs (KC705/KCU105/ZC706/ZCU102/ZCU104/ZCU106/VCU118)

    ○ LED0 - HDMI 1.4/2.0 TX Subsystem lock (when the HDMI Example Design is used)

    ○ Use the debug port to check if there are link data driven to the Video PHY Controller core.

    ○ See the Debugging Appendix in the *Video PHY Controller LogiCORE IP Product Guide* (PG230), and ensure there is no problem with clocking.

> ⭐ **IMPORTANT!** *If you observe any issues, test the same design on more than one HDMI sink to ensure that the issue observed is not specific to one particular sink model.*

In-system Debug

For in-system debugging, relevant information can be verified using UART before checking the hardware signals.

1. Press **i** to check the system information.

2. Press **z** to check the event log.

    The following table shows two examples of system flow event logs.

Table 29: **System Flow Event Log**

At System Start-up	While Changing Video Stream from UART Menu
| ```
VPHY log
------
GT init start
GT init done
TX frequency event
TX timer event
TX MMCM reconfig done
QPLL reconfig done
GT TX reconfig start
GT TX reconfig done
TX MMCM lock
QPLL lock
TX reset done
TX alignment done

HDMI TX log
------
Initializing HDMI TX core....
Initializing VTC core....
Reset HDMI TX Subsystem....
TX cable is connected....
TX Stream is Down
TX Set Stream, with TMDS (32)
TX Audio Unmuted
TX Stream is Up
``` | ```
VPHY log
------
TX frequency event
QPLL lost lock
TX frequency event
TX timer event
TX MMCM reconfig done
QPLL reconfig done
GT TX reconfig start
GT TX reconfig done
TX MMCM lock
QPLL lock
TX reset done
TX alignment done

HDMI TX log
------
TX Set Stream, with TMDS (128)
TX Stream is Down
TX Audio Unmuted
TX Stream is Up
``` |

3. Press **e** to check the EDID.

   In this release, the application software reads and parses the sink EDID to determine the sink capability. However, the current application does not block you from setting certain video formats although it is not supported by the sink. Instead, a warning message is prompted to indicate the sink limitation when you type **e**.

   In this release, for the EDID parsing feature, three VERBOSITY levels are defined. You can set it according to your requirements. It is accessible from the `video_common` library, in `xvidc_cea861.h`.

```
#define XVIDC_EDID_VERBOSITY          0
```

   where `XVIDC_EDID_VERBOSITY` is defined in the following table.

XVIDC_EDID_VERBOSITY	Description
0 (Default)	Read and parse the EDID No display of capability
1	Read and parse the EDID Display of basic capability of the sink
2	Read and parse the EDID Display of full capability of the sink

> ⭐ **IMPORTANT!** *In the Example design running on MicroBlaze™, limited BRAM resources are allocated to store the software binary. If you enable the `XVIDC_EDID_VERBOSITY` to a higher level, which consumes more software resources, you might need to increase the BRAM allocation. Otherwise, you might experience system instability such as UART hangs.*

4. Press **h** to check the HDCP status.

   Note: The HDCP debug menu is not enabled by default. You can enable it by setting it in `xhdmi_example.h`.

   ```
   /* Enabling this will enable HDCP Debug menu */
   #define HDCP_DEBUG_MENU_EN 1
   ```

   Note: You must check the software size so that it does not exceed the allocated BRAM.

5. It is recommended that you use the following steps while changing the video mode through the HDMI application during TX-Only mode (as a color bar or any supported video pattern available) in a pass-through topology or TX-only topology example design:

   a. Select the intended resolution from the menu (by default, the video mode color space is set to RGB, frame rate is set to 60 Hz (except, 576p and 576i which are 50 Hz, and the 3840x2160p (SB) which is 30 Hz, and the color depth is set to 8 BPC)

   b. Select the intended frame rate from the menu (this will only change the frame rate on top of the video mode set in 5.a)

   c. Select the intended color space from the menu (this will only change the color space on top of the video mode set in 5.b)

   d. Select the intended color depth from the menu (this will only change the color depth on top of the video mode set in 5.c)

   Note: Color depth option should be set at the last step.

Interface Debug

AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. 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 subsystem clocks are toggling and that the enables are also asserted.

- Add AXI4-Lite interface to ILA, and analysis data captured when triggering at `s_axi_rvalid`.

AXI4-Stream Interfaces

If data is not being transmitted or received, check the following conditions:

- If the receive <interface_name>_tready is stuck Low, the subsystem cannot send data. Check if there is an issue at the AXI4-Stream Slave.
- Check that the `aclk` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed.
- Check subsystem configuration.

AXI4-Stream Audio Interface

To ensure that the audio is working in the HDMI 1.4/2.0 TX Subsystem, the AXI4-Stream must be constructed as described below.

The HDMI 1.4/2.0 TX Subsystem supports up to eight audio channels. The audio data is transmitted through an AXI4-Stream audio interface, which is a customized AXI4-Stream protocol that is used to send audio samples with sideband signals defined in the AES3 specification.

The sub-frame format for audio sample is shown as follows.

Figure 44: **Sub-Frame Format**

A frame is uniquely composed of two sub-frames. The first sub-frame normally starts with preamble X, and the second sub-frame always starts with preamble Y. However, every 192 frames forms one Audio Block. And the first sub-frame in each Audio Block starts with a preamble Z. An illustration is shown as follows.

Figure 45: **Audio Sub-Frames**

In the case of more than 2 channels, every 2 channels are considered as a single AES3 audio block. For example, using 8 audio channels, one audio block consists of 192*8 audio samples. For the first 8 samples of an audio block, the preamble for audio ch0, ch2, ch4, ch6 are Z. In remaining part of audio block, the preamble for audio ch0, ch2, ch4, ch6 are X. The preambles for audio ch1, ch3, ch5, ch7 are always Y through out of the whole audio block. An illustration of 8 channel audio is shown below.

Figure 46: **8 Channel Audio Timing**

If 8 channel audio is enabled in your design, and only N Channels (where N is less than 8, for example, 6) out of 8 channels carry valid audio data, for the unused channels, you must pack the audio data with zeros and the sub-frame data allocation follows as shown above.

Send Feedback

Application Software Development

Device Drivers

The HDMI 1.4/2.0 TX Subsystem driver abstracts the included supporting elements and provides you with an API for control. The API can be easily integrated into your application thereby providing an out-of-the-box solution.

The subsystem driver is a bare-metal driver, which provides an abstracted view of the feature set provided by each sub-core. It dynamically manages the data and control flow through the processing elements, based on the input/output stream configuration set at run time. Internally, it relies on sub-core drivers to configure the sub-core IP blocks.

Architecture

The subsystem driver provides an easy-to-use, well-defined API to help integrate the subsystem in an application without having to understand the underlying complexity of configuring each and every sub-core.

The subsystem driver consists of the following:

- **Subsystem layer:** Queries exported hardware to determine the subsystem hardware configuration and pull-in sub-core drivers, at build time. It abstracts sub-core drivers, which interface with hardware at register level, into a set of functional APIs. The subsystem driver uses these APIs to dynamically manage the data flow through processing elements.

- **Sub-core drivers:** Every included sub-core has a driver associated with it that provides APIs to interface with the core hardware.

The following figure shows the HDMI 1.4/2.0 TX Subsystem architecture.

Figure 47: **Subsystem Driver Architecture**

The HDMI 1.4/2.0 TX Subsystem is a MAC subsystem which works with a Video PHY Controller (PHY) to create a video connectivity system. The HDMI 1.4/2.0 TX Subsystem is tightly coupled with the Xilinx Video PHY Controller, which itself is independent and offers flexible architecture with multiple-protocol support. Both MAC and PHY are dynamically programmable through the AXI4-Lite interface.

Figure 48: **MAC Interfaces with PHY**

Usage

The HDMI 1.4/2.0 TX Subsystem provides a set of API functions for application code to use. Additionally, when the HDMI 1.4/2.0 TX Subsystem hardware interrupts are generated, the subsystem driver is invoked to configure the system accordingly. The HDMI 1.4/2.0 TX Subsystem provides a callback structure to hook up your own callback functions.

First, the video stream must be started in the application. Then, valid AUX data and audio data can be inserted after the video is locked. However, because the application knows what video format is being sent and what audio format is embedded, the ACR number can be calculated and set before the audio stream is ready to be sent.

In the following sections, only HDMI related modules are covered. The user application needs to take care of system peripheral, such as timer, UART, external system clock generator, etc.

Note: Because the HDMI Transmit and Receive Subsystems have many common features, the HDMI Common Library is introduced for the purpose of defining common data structures which can be shared by both subsystems.

HDMI TX Subsystem Flow

The HDMI 1.4/2.0 TX Subsystem in general responds to the following two events:

- Hot-plug signal (HPD) from the sink device

- Software user application to indicate a change of video stream (for example, change of video resolution)

The main program flow is shown in the following figure. At execution, the software application initializes the HDMI 1.4/2.0 TX Subsystem IP and registers the callback functions in the provided hooks.

Figure 49: **TX Flow**

X23121-081519

Application Integration

The following figure shows example code to show how an HDMI 1.4/2.0 TX Subsystem can be used in your application.

Figure 50: **Application Example Code**

```
1819    XV_HdmiTxSs HdmiTxSs;          /* HDMI TX SS structure */
1820    XV_HdmiTxSs_Config *XV_HdmiTxSs_ConfigPtr;
1821
1822    // Initialize HDMI TX Subsystem
1823    XV_HdmiTxSs_ConfigPtr =
1824            XV_HdmiTxSs_LookupConfig(XPAR_V_HDMI_TX_SS_0_V_HDMI_TX_DEVICE_ID);
1825
1826    if(XV_HdmiTxSs_ConfigPtr == NULL)
1827    {
1828        HdmiTxSs.IsReady = 0;
1829    }
1830
1831    //Initialize top level and all included sub-cores
1832    Status = XV_HdmiTxSs_CfgInitialize(&HdmiTxSs, XV_HdmiTxSs_ConfigPtr,
1833                                    XV_HdmiTxSs_ConfigPtr->BaseAddress);
1834    if(Status != XST_SUCCESS)
1835    {
1836      xil_printf("ERR:: HDMI TX Subsystem Initialization failed %d\r\n", Status);
1837    }
1838
1839    //Register HDMI TX SS Interrupt Handler with Interrupt Controller
1840    Status |= XIntc_Connect(&Intc,
1841                XPAR_MICROBLAZE_SS_AXI_INTC_0_V_HDMI_TX_SS_0_IRQ_INTR,
1842                (XInterruptHandler)XV_HdmiTxSS_HdmiTxIntrHandler,
1843                (void *)&HdmiTxSs);
1844
1845    if (Status == XST_SUCCESS){
1846        XIntc_Enable(&Intc,
1847                XPAR_MICROBLAZE_SS_AXI_INTC_0_V_HDMI_TX_SS_0_IRQ_INTR);
1848    }
```

To integrate and use the HDMI 1.4/2.0 TX Subsystem driver in your application, the following steps must be followed:

1. Include the subsystem header file `xv_hdmitxss.h` that defines the subsystem object.

2. Declare and allocate space for the subsystem instance in your application code. For example:

   ```
   XV_HdmiTxSs HdmiTxSs;
   ```

3. In the subsystem driver instance, there is a metadata structure to store the subsystem hardware configuration. Declare a pointer variable in the application code to point to the instance:

   ```
   XV_HdmiTxSs_Config *XV_HdmiTxSs_ConfigPtr;
   ```

4. For each subsystem instance, the data structures declared in the previous two steps need to be initialized based on its hardware configuration, which is passed through metadata structure from `xparameters.h` uniquely identified by device ID.

   To initialize the subsystem, call the following two API functions:

   ```
   XV_HdmiTxSs_Config* XV_HdmiTxSs_LookupConfig(u32 DeviceId);
   int XV_HdmiTxSs_CfgInitialize(XV_HdmiTxSs *InstancePtr,
                                 XV_HdmiTxSs_Config *CfgPtr,
                                 u32 EffectiveAddr);
   ```

   The Device ID can be found in `xparameters.h`:

   ```
   XPAR_[HDMI TX Subsystem Instance Name in IPI]_DEVICE_ID
   ```

5. Each interrupt source has an associated ISR defined in the subsystem. Register the ISR with the system interrupt controller and enable the interrupt.

   ```
   int XIntc_Connect(XIntc *InstancePtr,
                     u8 Id,
                     XInterruptHandler Handler,
                     void *CallBackRef);
   void XIntc_Enable(XIntc *InstancePtr,
                     u8 Id);
   ```

   where ID can be found in `xparameters.h`.

HDCP TX Overview

The HDMI 1.4/2.0 TX Subsystem driver is responsible for combining the HDCP 1.4 and HDCP 2.2 driver APIs into a single common API for use by the user level application. The common HDCP driver API is able to handle the following HDCP configurations: HDCP 1.4 only, HDCP 2.2 only, and both. When both protocols are enabled, the common HDCP driver ensures that only one is active at any given time.

HDCP TX Driver Integration

This section describes the steps required to initialize and run the HDCP TX. The application should call the functions roughly in the order specified to ensure that the driver operates properly. When only a single HDCP protocol is enabled, either 1.4 or 2.2, a subset of the function calls might be needed.

1. Load the HDCP production keys into the HDMI subsystem. This function needs to be called for each key that is loaded. If HDCP 1.4 and 2.2 are enabled all the keys must be loaded, otherwise a subset of the keys are loaded. Note that the byte arrays used to store the key octet strings for HDCP are defined in big endian byte order.

   - `XV_HdmiTxSs_HdcpSetKey`

     ○ `XV_HDMITXSS_KEY_HDCP14`

     ○ `XV_HDMITXSS_KEY_HDCP22_LC128` (128-bit DCP Licensed Constant)

> ⭐ **IMPORTANT!** *In an HDCP-enabled system, after the key management block is successfully initialized, a bit is set to block reading out the key again. The bit remains set until the device is reprogrammed. Therefore, the keys can only be read once during initialization.*

2. Initialize the HDMI 1.4/2.0 TX Subsystem driver after the HDCP keys have been loaded. Initializing the subsystem starts the HDCP 1.4/2.2 drivers internally.

3. Connect the HDCP interrupt handlers to the interrupt controller interrupt ID:

   - `XV_HdmiTxSS_HdcpIntrHandler`

   - `XV_HdmiTxSS_HdcpTimerIntrHandler`

   - `XV_HdmiTxSS_Hdcp22TimerIntrHandler`

4. Set the HDCP user callback functions. These callback functions are optional and used to hook into the HDCP state machine and allow the user to take action at various stages of the HDCP protocol. If there is no use for the callback at the application level, then the callback can be left undefined.

   - `XV_HdmiTxSs_SetCallback`

     ○ `XV_HDMITXSS_HANDLER_HDCP_AUTHENTICATED`

     ○ `XV_HDMITXSS_HANDLER_HDCP_DOWNSTREAM_TOPOLOGY_AVAILABLE`

     ○ `XV_HDMITXSS_HANDLER_HDCP_UNAUTHENTICATED`

5. Execute the poll function to run the HDCP state machine. This function checks to see which HDCP protocol is enabled, and then execute only the active protocol. The call to this function can be inserted in the main loop of the user application and should execute continuously. Because the HDCP TX state machine is run using this poll function, it is important to ensure that this function is given adequate CPU runtime, especially during authentication attempts.

   - `XV_HdmiTxSs_HdcpPoll`

6. Optionally, set the HDCP protocol capability. The default option is both, which means that if both HDCP 1.4 and HDCP 2.2 are included as part of the HDMI subsystem, the transmitter tries to authenticate with either protocol based on the capability of the downstream device. Note that HDCP 2.2 is given priority over HDCP 1.4. If the capability is set to none, then authentication attempts are ignored.

   - `XV_HdmiTxSs_HdcpSetCapability`

     ○ `XV_HDMITXSS_HDCP_NONE`

     ○ `XV_HDMITXSS_HDCP_14`

     ○ `XV_HDMITXSS_HDCP_22`

     ○ `XV_HDMITXSS_HDCP_BOTH`

7. Authentication should be initiated only after the transmission of video to the downstream device. It is the responsibility of the user application to determine when to issue authentication requests. Authentication requests are commonly initiated for the following events: stream-up, and HPD toggle. In the event that the first authentication request is not successful, the user application can issue another authentication request.

    - `XV_HdmiTxSs_HdcpPushEvent`

        ◦ `XV_HDMITXSS_HDCP_AUTHENTICATE_EVT`

8. Check the status of authentication. These checks could be performed before issuing authentication requests.

    - `XV_HdmiTxSs_HdcpIsAuthenticated`

    - `XV_HdmiTxSs_HdcpIsInProgress`

9. When authentication is successful, the application is allowed to enable encryption. The enablement of encryption can happen any time after successful authentication and is the responsibility of the application to manage. For example, an application might decide to enable encryption only for restricted content, but disable encryption for standard content.

    - `XV_HdmiTxSs_HdcpEnableEncryption`

    - `XV_HdmiTxSs_HdcpDisableEncryption`

10. Check the status of the cipher encryption. This is the instantaneous encryption status of the cipher and can change between subsequent frames. For repeater or pass-through applications, special care must be taken to block downstream content if the upstream interface is encrypted while the downstream interface is not encrypted.

    - `XV_HdmiTxSs_HdcpIsEncrypted`

11. Check the overall HDCP protocol status and log data. You can also set the level of detail for log information reported.

    - `XV_HdmiTxSs_HdcpInfo`

    - `XV_HdmiTxSs_SetInfoDetail`

Integrate Video PHY Controller Driver for HDMI 1.4/2.0 TX Subsystem Use

Because the HDMI 1.4/2.0 TX Subsystem is closely coupled with the Video PHY Controller, the following example code demonstrates how a Video PHY Controller can be used in your application.

Figure 51: **Application Example Code**

```
2039        XVphy Vphy;                  /* VPHY structure */
2040        XVphy_Config *XVphyCfgPtr;
2041        // Initialize Video PHY
2042        XVphyCfgPtr = XVphy_LookupConfig(XPAR_VID_PHY_CONTROLLER_0_DEVICE_ID);
2043        if (XVphyCfgPtr == NULL) {
2044           print("Video PHY device not found\n\r\r");
2045           return XST_FAILURE;
2046        }
2047
2048        /* Initialize HDMI VPHY */
2049        Status = XVphy_HdmiInitialize(&Vphy, 0,
2050                    XVphyCfgPtr, XPAR_CPU_CORE_CLOCK_FREQ_HZ);
2051        if (Status != XST_SUCCESS) {
2052           print("HDMI VPHY initialization error\n\r");
2053           return XST_FAILURE;
2054        }
2055
2056        /* Register VPHY Interrupt Handler */
2057        Status = XIntc_Connect(&Intc,
2058                    XPAR_MICROBLAZE_SS_AXI_INTC_0_VID_PHY_CONTROLLER_0_IRQ_INTR,
2059                    (XInterruptHandler)XVphy_InterruptHandler,
2060                    (void *)&Vphy);
2061
2062        if (Status != XST_SUCCESS) {
2063           print("HDMI VPHY Interrupt Vec ID not found!\n\r");
2064           return XST_FAILURE;
2065        }
2066
2067        /* Enable VPHY Interrupt */
2068        XIntc_Enable(&Intc,
2069                XPAR_MICROBLAZE_SS_AXI_INTC_0_VID_PHY_CONTROLLER_0_IRQ_INTR);
```

To integrate and use the Video PHY Controller for the HDMI 1.4/2.0 TX Subsystem in the application code, the following steps must be followed:

1. Include the subsystem header file `xvphy.h` that defines the subsystem object.

2. Declare and allocate space for a Video PHY Controller instance in your application code.

   Example:

   ```
   XVphy Vphy;
   ```

3. In the Video PHY Controller instance, there is a metadata structure to store its hardware configuration. Declare a pointer variable in the application code to point to the instance:

   ```
   XVphy_Config *XVphyCfgPtr;
   ```

4. For each Video PHY Controller instance, the above data structure needs to be initialized based on its hardware configuration, which is passed through meta-structure from `xparameters.h` uniquely identified by the device ID.

Send Feedback

To initialize the Video PHY Controller, call the following two API functions:

```
XVphy_Config *XVphy_LookupConfig(u16 DeviceId);
u32 XVphy_HdmiInitialize(XVphy *InstancePtr,
                         u8 QuadId,
                         XVphy_Config *CfgPtr,
                         u32 SystemFrequency);
```

The Device ID can be found in `xparameters.h`:

```
XPAR_[Video PHY Controller Instance Name in IPI]_DEVICE_ID
```

Similarly, `SystemFrequency` is the system frequency, which can also be found in `xparameters.h`

Note:

- Xilinx recommends initializing the PHY controller after the HDMI 1.4/2.0 TX Subsystem initialization is completed.

- Registering the PHY Controller interrupts are part of system application integration. Steps are shown in the previous section and not repeated here.

Interrupts

All interrupts generated by the HDMI 1.4/2.0 TX Subsystem are listed here:

1. HPD – Peripheral I/O to detect HDMI cable 5.0V signal

   a. Rising edge – If HPD disconnect interrupt has happened and HPD line is asserted for more than 10 ms.

   b. Falling edge – If HPD line is de-asserted for more than 100 ms.

   c. Toggle - A pulse occurred on the HPD line with a pulse width between 50 and 99 ms.

2. Link Ready – Every time the PHY Controller is reconfigured, the `link_clk` is regenerated. An HDMI TX sub-core register bit (link status bit) reflects the change of `link_clk` status. . When stable `link_clk` is detected, it is set to 1. When `link_clk` becomes unstable, it is set to 0. The Link Ready is an interrupt to detect the change of the link status bit.

   a. Rising edge – Link is up

   b. Falling edge – Link is down

3. Vertical Sync – This is to reflect the change of HDMI TX sub-core `vsync` input signal in its video interface bus.

   a. Rising edge – Vertical Sync is detected

4. Video Bridge Unlocked - The AXI4-Stream to Video Out Bridge lost lock with the incoming AXI4 Video Stream.

5. HDCP1.4 Interrupt (only available when HDCP 1.4 is enabled in hardware)

6. HDCP 1.4 Timer Interrupt (only available when HDCP 1.4 is enabled in hardware)

7. HDCP 2.2 Timer Interrupt (only available when HDCP 2.2 is enabled in hardware)

Table 30: **Mapping Between Interrupt Sources and Application Callback Functions**

Interrupts	Callback
HPD	XV_HDMITXSS_HANDLER_CONNECT
	XV_HDMITXSS_HANDLER_TOGGLE
Link Ready (note this is edge triggered)	XV_HDMITXSS_HANDLER_STREAM_UP
	XV_HDMITXSS_HANDLER_STREAM_DOWN
	Note: Two callbacks are mapped to the same interrupt source.
	Link Ready rising edge: Stream Up
	Link Ready falling edge: Stream Down
Vertical Sync	XV_HDMITXSS_HANDLER_VS
Video Bridge Unlocked	XV_HDMITXSS_HANDLER_BRDGUNLOCK
HDCP 1.4 Interrupt	
HDCP 1.4 Timer Interrupt	
HDCP 2.2 Timer Interrupt	
	XV_HDMITXSS_HANDLER_HDCP_AUTHENTICATE XV_HDMITXSS_HANDLER_HDCP_DOWNSTREAM_TOPOLOGY_AVAILABLE XV_HDMITXSS_HANDLER_HDCP_UNAUTHENTICATED Note: This callback function is not directly mapped to any interrupt source. Instead it is executed when the HDCP authentication state machine has reached the authenticated state.

Application Callback Functions

The subsystem driver provides a mechanism for the application to register a user-defined function that gets called within an interrupt context.

Callback functions defined in the application code must be registered with provided handlers, using the following defined API:

```
int XV_HdmiTxSs_SetCallback(XV_HdmiTxSs *InstancePtr,
                            u32 HandlerType,
                            void *CallbackFuncPtr,
                            void *CallbackRef);
```

Available handlers are defined in `xv_hdmitxss.h`:

- XV_HDMITXSS_HANDLER_CONNECT

- XV_HDMITXSS_HANDLER_STREAM_UP

- XV_HDMITXSS_HANDLER_STREAM_DOWN

- XV_HDMITXSS_HANDLER_VS

- XV_HDMITXSS_HANDLER_HDCP_AUTHENTICATE

XV_HDMITXSS_HANDLER_CONNECT

This interrupt is triggered every time an HDMI TX cable connection or disconnection (HPD level transition) occurs. The callback function must perform the following:

1. Check if the event is cable connected or cable disconnected:

```
XV_HdmiTxSs *HdmiTxSsPtr = (XV_HdmiTxSs *)CallbackRef;

HdmiTxSsPtr->IsStreamConnected
1 - Connected
0 - Disconnected
```

2. Enable or disable the differential input clock buffer depending on if cable connection or disconnection occurs, respectively.

```
void XVphy_IBufDsEnable(XVphy *InstancePtr,
                        u8 QuadId,
                        XVphy_DirectionType Dir,
                        u8 Enable);
```

3. Detect if the HDMI sink connected is HDMI 2.0 capable and if cable is connected.

```
int XV_HdmiTxSs_DetectHdmi20(XV_HdmiTxSs *InstancePtr);
```

4. Now, the HDMI sink has been detected, retrieve the sink EDID information, and store it in a local buffer (256 bytes) using the following API:

```
int XV_HdmiTxSs_ReadEdid(XV_HdmiTxSs *InstancePtr, u8 *Buffer);
```

XV_HDMITXSS_HANDLER_STREAM_UP

This interrupt is triggered every time the Video PHY Controller is reconfigured and the output clock is stabilized and ready for the HDMI 1.4/2.0 TX Subsystem to transmit a video stream.

The callback function must perform the following:

1. If an HDMI Retimer or equalizer is used in the system, configure the Retimer with the correct setting based on the required line rate.

2. Enable the TX TMDS clock by calling the Video PHY Controller API:

```
void XVphy_Clkout1OBufTdsEnable(XVphy *InstancePtr,
                                XVphy_DirectionType Dir,
                                u8 Enable);
```

3. Set HDMI 1.4/2.0 TX Subsystem sampling rate with the Video PHY Controller TX sampling rate.

```
void XV_HdmiTxSs_SetSamplingRate(XV_HdmiTxSs *InstancePtr,
                                 u8 SamplingRate);
```

Send Feedback

XV_HDMITXSS_HANDLER_STREAM_DOWN

This interrupt is triggered every time the Video PHY Controller is reconfigured and the output clock is not stable for the HDMI 1.4/2.0 TX Subsystem to stream video.

The callback function might disable the TX TMDS clock by calling the Video PHY Controller API:

```
void XVphy_Clkout1OBufTdsEnable(XVphy *InstancePtr,
                                XVphy_DirectionType Dir,
                                u8 Enable);
```

XV_HDMITXSS_HANDLER_VS

This interrupt is triggered every time when an input video stream vertical sync is detected by the HDMI TX sub-core.

The callback function can be used to construct and send InfoFrames to the sink.

```
void XV_HdmiTxSs_SendAuxInfoframe(XV_HdmiTxSs *InstancePtr,
                                  void *Aux);
```

XV_HDMITXSS_HANDLER_HDCP_AUTHENTICATE

This interrupt is triggered when a cable is connected, a HDCP 1.4 or HDCP 2.2 is enabled, and HDCP is entering an authentication state.

The callback function must perform the following:

1. Enable HDCP encryption.
2. Signal to the system that authentication has successfully completed.

Video PHY Controller Interrupt Handlers for HDMI 1.4/2.0 TX Subsystem

There are several interrupt handlers available in the Video PHY Controller driver to hook up with user-defined callback functions to support HDMI 1.4/2.0 TX Subsystem functionality. These interrupt handlers are defined in `xvphy.h`:

- XVPHY_HDMI_HANDLER_TXINIT
- XVPHY_HDMI_HANDLER_TXREADY

Callback functions need to be defined in the application code and hooked up with these interrupt handlers.

```
void XVphy_SetHdmiCallback(XVphy *InstancePtr,
                           XVphy_HdmiHandlerType HandlerType,
                           void *CallbackFunc,
                           void *CallbackRef);
```

XVPHY_HDMI_HANDLER_TXINIT

This interrupt is triggered every time the Video PHY Controller detects an HDMI TX reference clock change.

The callback function must initialize a reference clock change process for the HDMI 1.4/2.0 TX Subsystem.

```
void XV_HdmiTxSs_RefClockChangeInit(XV_HdmiTxSs *InstancePtr);
```

XVPHY_HDMI_HANDLER_TXREADY

This interrupt is triggered every time the Video PHY Controller TX reset lock is done or when the Video PHY Controller TX alignment is done.

The callback function can update the Video PHY Controller ready for TX information to the application software.

Follow the steps in the Example Design chapter to create an example design, which contains all the procedures implemented and can serve as a reference for integrating the HDMI 1.4/2.0 TX Subsystem into your system.

Example Use Cases

In this section, some typical use cases are illustrated with how system reacts at run time to certain events and what is expected for you to perform. For actions expected in the callback functions, see Application Callback Functions for more information.

Related Information
Application Callback Functions

Use Case 1: Cable Plug In

HPD interrupt is received indicating Cable Connection.

* Callback function registered to XV_HDMITXSS_HANDLER_CONNECT Interrupt type is called.

Use Case 2: Cable Plug Out

HPD interrupt is received indicating Cable Disconnection.

* Callback function registered to XV_HDMITXSS_HANDLER_CONNECT Interrupt type is called.

Use Case 3: Send Infoframe

Vertical Sync (VS) interrupt is received.

• Callback function registered to XV_HDMITXSS_HANDLER_VS Interrupt type is called.

Use Case 4: Send Video Stream

1. Disable the Video PHY Controller TMDS clock for the HDMI 1.4/2.0 TX Subsystem through the API:

```
XVphy_Clkout1OBufTdsEnable(XVphy *InstancePtr,
                           XVphy_DirectionType Dir,
                           u8 Enable);
```

Example:

```
XVphy_Clkout1OBufTdsEnable(VphyPtr,
                           XVPHY_DIR_TX,
                           (FALSE));
```

2. Set the HDMI 1.4/2.0 TX Subsystem stream parameters through the API:

```
u32 XV_HdmiTxSs_SetStream(XV_HdmiTxSs *InstancePtr,
                          XVidC_VideoMode VideoMode,
                          XVidC_ColorFormat ColorFormat,
                          XVidC_ColorDepth Bpc,
                          XVidC_3DInfo *Info3D);
```

Example:

```
TmdsClock = XV_HdmiTxSs_SetStream(HdmiTxSsPtr,
                                  VideoMode,
                                  ColorFormat,
                                  Bpc,
                                  NULL);
```

3. Set the Video PHY Controller TX reference clock:

```
VphyPtr->HdmiTxRefClkHz = TmdsClock;
```

4. Set the HDMI TX parameter for the Video PHY Controller:

```
u32 XVphy_SetHdmiTxParam(XVphy *InstancePtr,
                         u8 QuadId,
                         XVphy_ChannelId ChId,
                         XVidC_PixelsPerClock Ppc,
                         XVidC_ColorDepth Bpc,
                         XVidC_ColorFormat ColorFormat);
```

Example:

```
Result = XVphy_SetHdmiTxParam(VphyPtr,
                              0,
                              XVPHY_CHANNEL_ID_CHA,
                              HdmiTxSsVidStreamPtr->PixPerClk,
                              HdmiTxSsVidStreamPtr->ColorDepth,
                              HdmiTxSsVidStreamPtr->ColorFormatId);
```

5. Program the external clock generator to provide the Reference TMDS clocks for the Video PHY Controller.

6. Video PHY Controller HDMI TX Init interrupt is received.

   • Callback function registered to XVPHY_HDMI_HANDLER_TXINIT Interrupt type is called.

7. Video PHY Controllerr HDMI TX Ready interrupt is received.

   • Callback function registered to XVPHY_HDMI_HANDLER_TXREADY Interrupt type is called.

8. HDMI TX Stream UP interrupt is received.

   • Callback function registered to XV_HDMITXSS_HANDLER_STREAM_UP Interrupt type is called.

Use Case 5: Support Multiple Channels Audio

Define: N = Number of Audio Channel

1. Change the Audio Infoframe by setting the channel count in the API

```
void XV_HdmiTxSs_SendAuxInfoframe(XV_HdmiTxSs *InstancePtr,
                                  void *AuxPtr);
/* 2 Channel count. Audio coding type refer to stream */
InstancePtr->HdmiTxPtr->Aux.Data.Byte[1] = N - 1;
```

2. Set HDMI TX SS audio channels using this API:

```
void XV_HdmiTxSs_SetAudioChannels(XV_HdmiTxSs *InstancePtr,
                                  u8 AudioChannels);
```

Example:

```
XV_HdmiTxSs_SetAudioChannels(&HdmiTxSs, N);
```

3. To demo using example design application software, update the following section of codes in `xhdmi_example.c`:

```
/* Enable 2-channel audio */
XhdmiAudGen_SetEnabChannels(&AudioGen, 2);
XhdmiAudGen_SetPattern(&AudioGen, 1, XAUD_PAT_PING);
XhdmiAudGen_SetPattern(&AudioGen, 2, XAUD_PAT_PING);
```

Example: To support eight channel audio:

```
/* Enable 8-channel audio */
XhdmiAudGen_SetEnabChannels(&AudioGen, 8);
XhdmiAudGen_SetPattern(&AudioGen, 1, XAUD_PAT_PING);
XhdmiAudGen_SetPattern(&AudioGen, 2, XAUD_PAT_PING);
XhdmiAudGen_SetPattern(&AudioGen, 3, XAUD_PAT_PING);
XhdmiAudGen_SetPattern(&AudioGen, 4, XAUD_PAT_PING);
XhdmiAudGen_SetPattern(&AudioGen, 5, XAUD_PAT_PING);
XhdmiAudGen_SetPattern(&AudioGen, 6, XAUD_PAT_PING);
XhdmiAudGen_SetPattern(&AudioGen, 7, XAUD_PAT_PING);
XhdmiAudGen_SetPattern(&AudioGen, 8, XAUD_PAT_PING);
```

Note: If you enable eight channel audio in your design, only six out of eight channels are used to carry valid audio data. For the unused channels, you must pack the audio data with zeros by muting them.

```
  XhdmiAudGen_SetPattern(&AudioGen, 7, XAUD_PAT_MUTE);
  XhdmiAudGen_SetPattern(&AudioGen, 8, XAUD_PAT_MUTE);
```

To update the audio channel allocation.

Information can be found in Table 20 in CED-861-D, under Audio InfoFrame Data Byte 4.

In the API,

```
void XV_HdmiTxSs_SendAuxInfoframe(XV_HdmiTxSs *InstancePtr, void
*AuxPtr);
```

You must set the data byte value before calculating the CRC.

Example

```
/* Channel Allocation */
InstancePtr->HdmiTxPtr->Aux.Data.Byte[4] = 0x13;
```

You can choose to construct your own infoframe in the application software, and use `API` `XV_HdmiTxSs_SendGenericAuxInfoframe` to send out.

Use Case 6: Enable HDMI Mode

Use the following API:

```
XV_HdmiTxSS_SetHdmiMode(&HdmiTxSs);
XV_HdmiTxSs_AudioMute(&HdmiTxSs, FALSE);
```

Use Case 7: Enable DVI Mode

Use the following API:

```
XV_HdmiTxSS_SetDviMode(&HdmiTxSs);
XV_HdmiTxSs_AudioMute(&HdmiTxSs, TRUE);
```

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

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

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

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

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

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

References

These documents provide supplemental material useful with this guide:

1. *Vivado Design Suite: AXI Reference Guide* (UG1037)

2. *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892)

3. *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893)

4. *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS182)

5. *Virtex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS183)

6. *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS181)

7. *Zynq-7000 SoC (Z-7007S, Z-7012S, Z-7014S, Z-7010, Z-7015, and Z-7020) Data Sheet: DC and AC Switching Characteristics* (DS187)

8. *Zynq-7000 SoC (Z-7030, Z-7035, Z-7045, and Z-7100) Data Sheet: DC and AC Switching Characteristics* (DS191)

9. *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922)

10. *Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics* (DS923)

11. *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925)

12. *Zynq UltraScale+ RFSoC Data Sheet: DC and AC Switching Characteristics* (DS926)

13. *HDMI Specifications* (http://www.hdmi.org/manufacturer/specification.aspx)

14. *HDCP Specifications* (http://www.digital-cp.com/hdcp-specifications)

15. *AXI4-Stream Video IP and System Design Guide* (UG934)

16. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)

17. *Vivado Design Suite User Guide: Designing with IP* (UG896)

18. *Vivado Design Suite User Guide: Getting Started* (UG910)

19. *ISE to Vivado Design Suite Migration Guide* (UG911)

20. *KCU105 Board User Guide* (UG917)

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

22. *Vivado Design Suite User Guide: Implementation* (UG904)

23. *AXI Interconnect LogiCORE IP Product Guide* (PG059)

24. *Video PHY Controller LogiCORE IP Product Guide* (PG230)

25. *HDCP 2.2 LogiCORE IP Product Guide* (PG249)

26. *HDCP 1.x Product Guide* (PG224)

27. *AXI4-Stream to Video Out LogiCORE IP Product Guide* (PG044)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
11/21/2019 Version 3.1	
Chapter 6: Example Design	Updated Example Design from SDK to the Vitis software platform.
Native Video (Vectored DE) Native Video/Native Video (Vectored DE) Interface Option	Added support for Vectored DE to the native video interface.
02/12/2019 Version 3.1	
N/A	• Added HDCP Repeater functionality • Added Pass-through + I2S support • Added PCB design guidelines for TMDS181 and DP159 • Updated Example Design steps
04/04/2018 Version 3.1	
N/A	• Added new board supports (ZCU104, ZCU106, VCU118). • Added new Example design features. • Updated Example Design steps. • Added Migrating and upgrading section. • Added debug information. • Updated AXI-Lite CPU clock supports. • Updated YUV420 remapping feature illustration.
12/20/2017 Version 3.0	
N/A	• Updated Example Design steps.. • Added notes for DP159 Settings. • Updated notes for CPU clock requirements.
10/04/2017 Version 3.0	
N/A	• Added Example design topology supports (TX-Only, Pass-through). • Added Example design VPHY Configuration Support (NI-DRU Enable/Disable, TXPLL selection, RXPLL selection). • Added Example design new board supports (ZCU102). • Added SDK application supports (RX, Pass-through and HDCP key utility) • Added Information for Example design description. • Added Software Flow diagram. • Added Information if not all audio channels are used. • Added Information about Native Video. • Added Information about Interlaced Video.
04/05/2017 Version 2.0	
N/A	Removed single pixel per clock support

Section	Revision Summary
11/30/2016 Version 2.0	
N/A	Added example design migration notes.
10/05/2016 Version 2.0	
N/A	• Added example design flow. • Added HPD XGUI option. • Added software use cases. • Updated Xilinx AUTOMOTIVE APPLICATIONS DISCLAIMER.
06/08/2016 Version 2.0	
N/A	Updated optional video over AXI4-Stream support.
04/06/2016 Version 2.0	
N/A	• Added Features section in IP Facts. • Updated Unsupported Features in Overview chapter • Updated Product Specification chapter. • Updated Designing with the Subsystem chapter. • Updated Design Flow Steps chapter. • Updated Hardware Testing and Video Resolutions sections. • Updated Application Software Development appendix.
11/18/2015 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

Send Feedback

www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

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

Copyright