HDMI 1.4/2.0 Transmitter Subsystem v3.0

LogiCORE IP Product Guide

Vivado Design Suite

PG235 October 4, 2017

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The HDMI 1.4/2.0 Transmitter Subsystem is a hierarchical IP that bundles a collection of HDMI® TX 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 TX 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
- Support color space for RGB, YUV 4:4:4, YUV 4:2:2, YUV 4:2:0
- Support AXI4-Stream Video input stream and Native Video input stream
- Audio support for up to 8 channels
- High bit rate (HBR)
- Info frames
- Data Display Channel (DDC)
- Hot-Plug Detection
- 3D video support
- Optional High Bandwidth Digital Copy Protection (HDCP) 1.4 support
- Optional HDCP 2.2 support
- Optional Video over AXIS compliant NTSC/PAL Support
- Optional Video over AXIS compliant YUV420 Support
- Optional HPD Active polarity

LogiCORE™ IP Facts Table	
Subsystem Specifics	
Supported Device Family[1]	UltraScale+™ Families (GTHE4) UltraScale™ Architecture (GTHE3) Zynq®-7000 All Programmable SoC 7 Series (GTXE2, GTHE2) Artix®-7 (GTPE2)
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
Test Bench	Not Provided
Constraints File	XDC
Simulation Model	Not Provided
Supported S/W Driver[2]	Standalone
Tested Design Flows[3]	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. Standalone driver details can be found in the SDK directory (<install_directory>/SDK/<release>/data/embeddedsw/doc/xilinx_drivers.htm). Linux OS and driver support information is available from the Xilinx Wiki page.
3. For the supported versions of the 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.

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

Licensing and Ordering Information

License Checkers

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

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

IMPORTANT: *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.

License Type

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

Product Specification

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

Introduction

Because the HDMI 1.4/2.0 Transmitter 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 Transmitter Subsystem is shown in Figure 2-1.

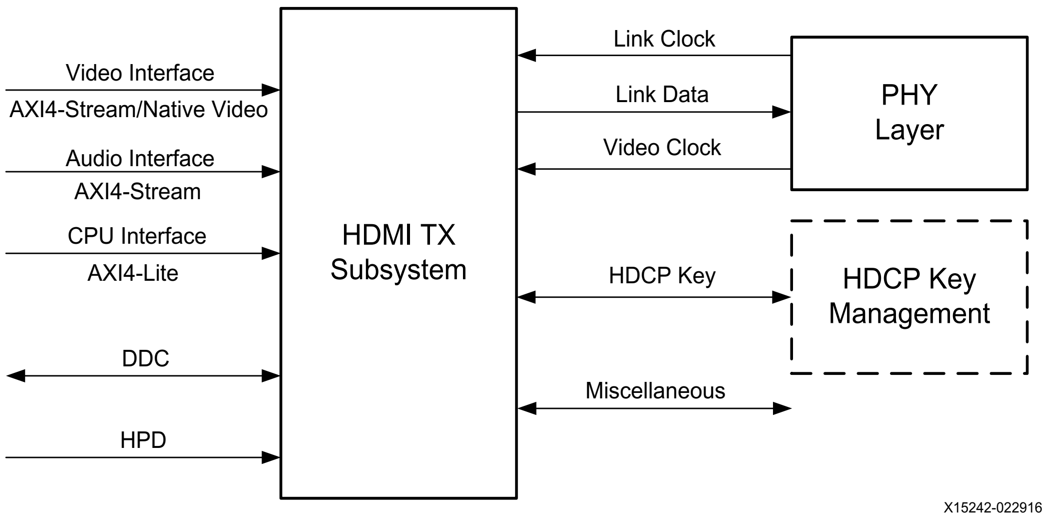


Figure 2-1: **Subsystem Block Diagram**

The HDMI 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 some supporting modules (Video Timing Controller and AXI4-Stream to Video Out Bridge) to construct HDMI TX Subsystem to be able to supportAXI4-Stream based video. By performing this, HDMI TX Subsystem is able to work seamlessly with other Xilinx video processing IP cores.

The HDMI TX Subsystem has a built-in capability to optionally support both HDCP 1.4 and HDCP 2.2 encryption.

Figure 2-2 shows the internal structure of the HDMI TX Subsystem when **AXI4-Stream Video Interface** is selected as 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.

The HDMI 1.4/2.0 Transmitter Subsystem supports two types of video interface:

- AXI4-Stream Video Interface
- Native Video Interface

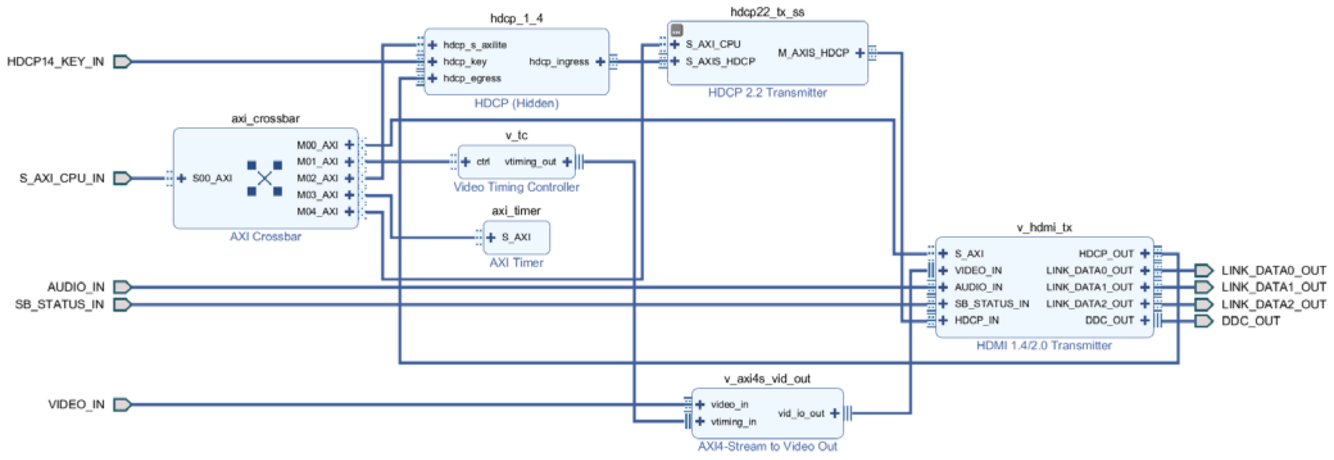


Figure 2-2: **HDMI TX Subsystem Internal Structure in AXI4-Stream Video Interface Mode**

The HDMI TX Subsystem also provides an option to support a native video interface. When **Native Video Interface** is selected, the HDMI TX Subsystem is constructed without the Video Timing Controller and AXI4-Stream to Video Out Bridge. Therefore, the HDMI TX Subsystem is allowed to take native video from its own video devices and convert into HDMI signals.In native video mode, the HDMI TX Subsystem still has a built-in capability to optionally support both HDCP 1.4 and HDCP 2.2 encryption.

Figure 2-3 shows the internal structure of the HDMI TX Subsystem when **Native Video Interface** is selected as video interface. In this illustration, both HDCP 1.4 and HDCP 2.2 are selected.

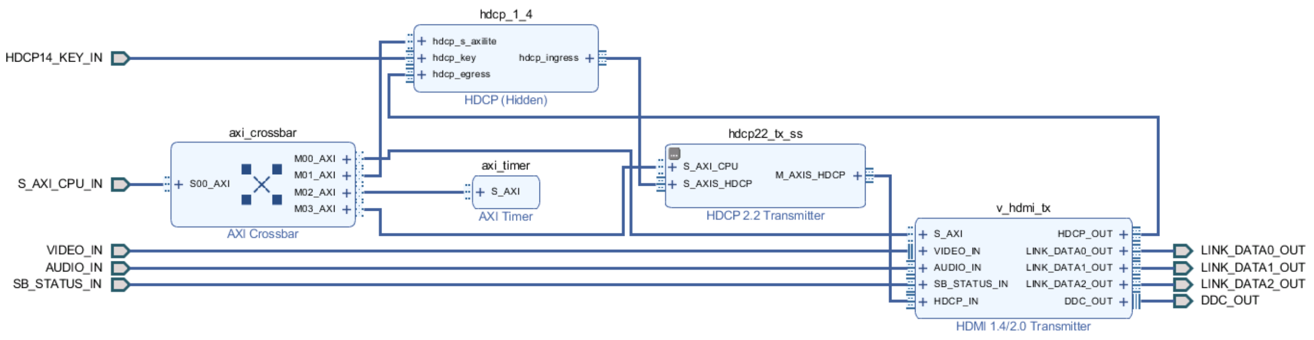


Figure 2-3: **HDMI TX Subsystem Internal Structure in Native Video Interface Mode**

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 submodules are used to construct the HDMI TX Subsystem and all the submodules 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 submodules inside the HDMI TX Subsystem.

IMPORTANT: *The direct register level access to any of the submodules is not supported.*

The HDMI 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 a HDMI sink through a 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 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 HDMI TX subsystem is used in DVI mode, the ACR inputs are ignored. You can decide to leave them open or connect them to some fix values (for example, connecting `acr_cts`, `acr_n`, and `acr_valid` to 0). See Chapter 5, Example Design for an example ACR module that is part of the audio pattern generation system.

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 following two bits in SCDC register address offset 0x20 for TMDS configurations (Table 10-19 of HDMI 2.0 spec).

- Bit 1 TMDS_BIt_Clock_Ratio
- Bit 0 Scrambling_Enable

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

- XV_HdmiTxSs_StreamStart

Two underlining subcore API drivers are called to set the above two SCDC bits with respect to the video stream to sent.

- XV_HdmiTx_Scrambler(InstancePtr->HdmiTxPtr); is to used to:
  - Enable HDMI TX scrambler for HDMI 2.0 video and disable scrambler for HDMI 1.4 video stream.
  - Update scrambler bit in Sink's TMDS Configuration register
- V_HdmiTx_ClockRatio(InstancePtr->HdmiTxPtr); is to set TMDS Clock Ratio bit for HDMI 2.0 video.

An API is also available at HDMI TX Subcore driver to show the sink's SCDC register values. (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 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.

InfoFrames

There are two basic InfoFrames expected in any HDMI system, which are Auxiliary Video Information (AVI) Infoframe and Audio Infoframe. Both are handled by the HDMI TX Subsystem drivers. The HDMI TX Subsystem driver is able to construct and send Vendor Specific InfoFrames to support some specific features, such as 3D video support. All InfoFrames are described in detail in CEA-861-F.

In the HDMI TX Subsystem driver, an extra API is prepared if you want to define your own InfoFrames. As a guideline, an InfoFrame is structured with a 4-byte header and 32-byte data (payload). Both header and payload must be constructed prior to sending the information frame API function call. You also need to calculate your own CRC and place the CRC at the right location so that the HDMI Sink is able to decode the InfoFrame.

This is an example of a function call:

```
XV_HdmiTxSs_SendGenericAuxInfoframe(HdmiTxSsPtr, AuxPtr);
```

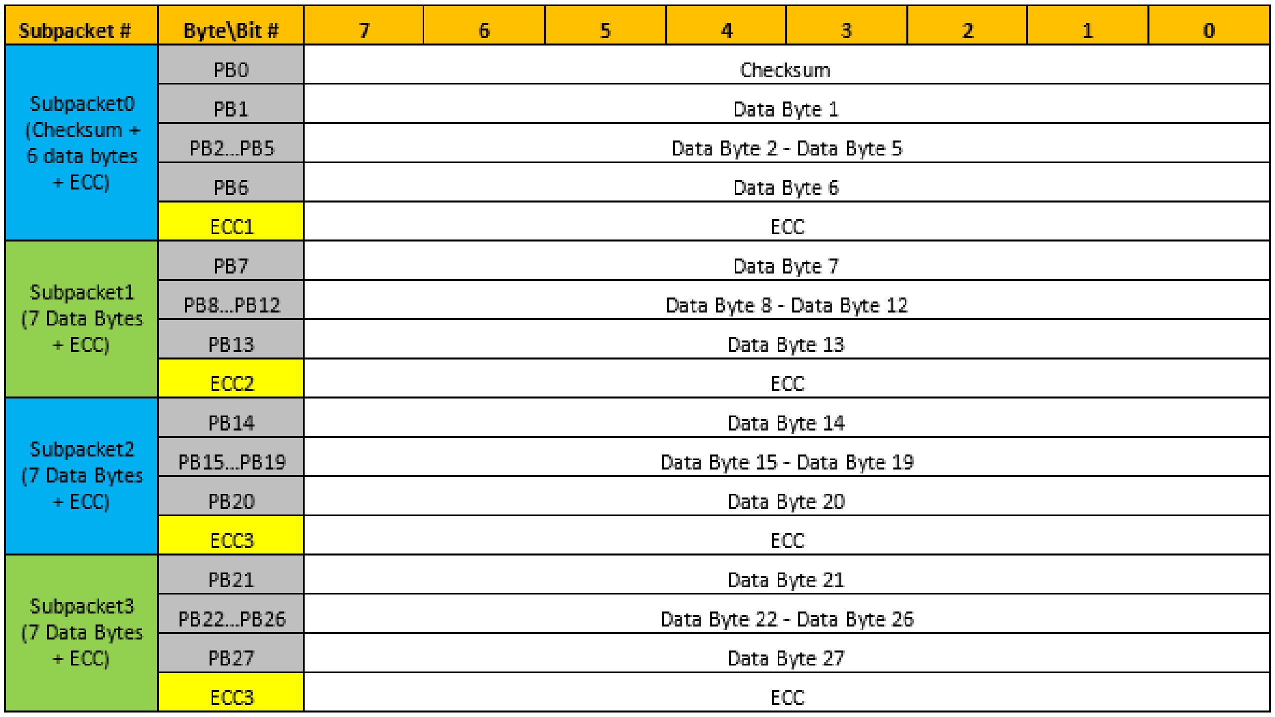
`HdmiTxSsPtr` is a pointer to the HDMI TX Subsystem, and `AuxPtr` is the pointer to the array where the InfoFrame header and data are stored.

Figure 2-4 graphically represents an HDMI Infoframe structure, which is one type of HDMI data island packet. For HDMI, all data island packets consist of a 4-byte packet header and a 32 bytes of packet contents. The packet header contains 24 data bits (3 bytes) and 8 bits (1 byte) of BCH ECC parity.

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

Figure 2-4: **Packet Header**

The packet body, graphically represented in Figure 2-5, is made from four subpackets; each subpacket includes 56 bits (7 bytes) of data and 8 bits (1 byte) of BCH ECC parity.



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							

Figure 2-5: **Packet Body**

Notes:

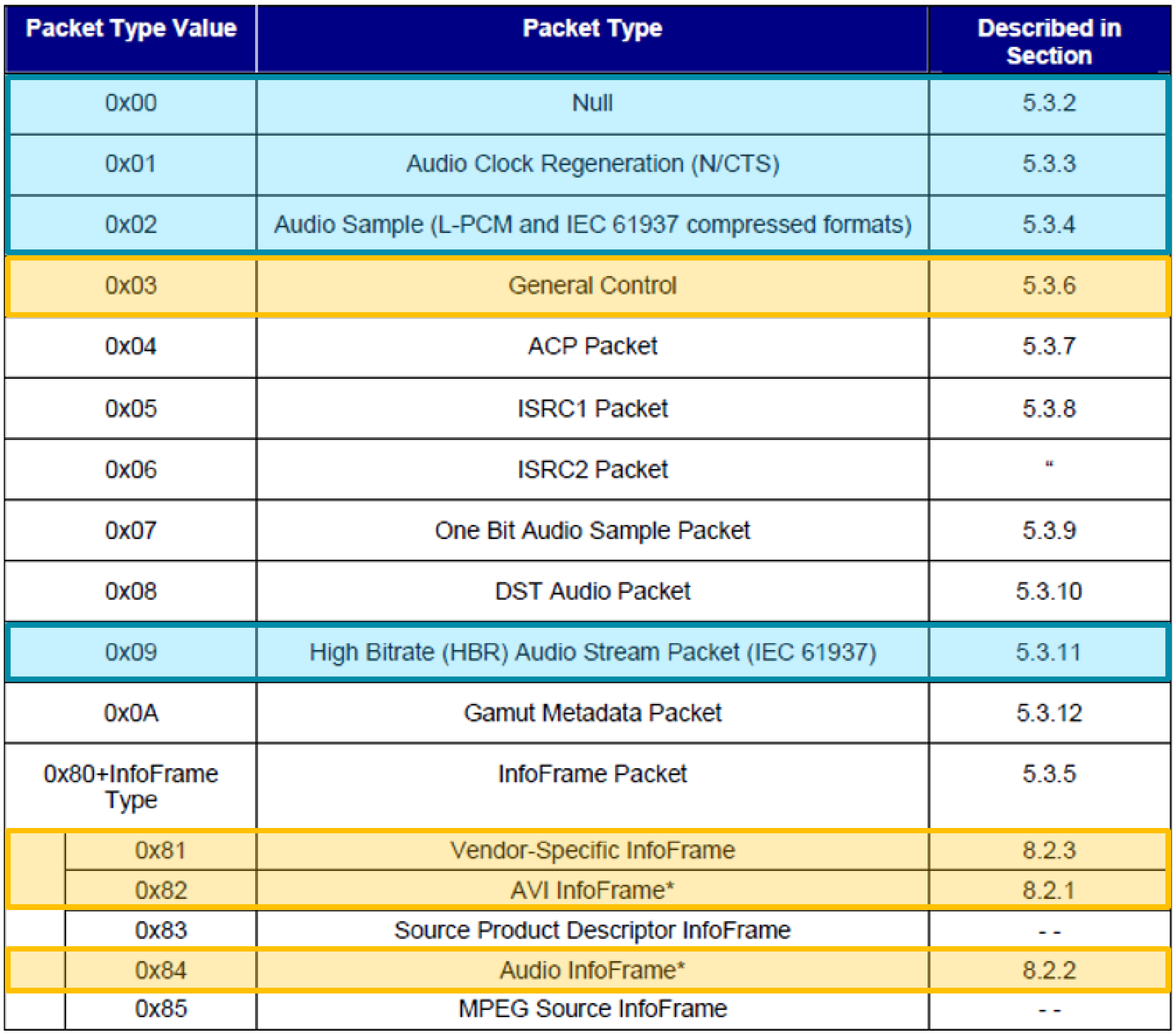
1. ECC is calculated in HDMI 1.4/2.0 Transmitter Subsystem core. Therefore, must construct HB0...HB2, and PB0, PB1...PB26, PB27 according to HDMI specs in the software.
2. When calculating the checksum value (PB0), the ECC values are ignored.

Refer to section 5.2.3.4 and 5.2.3.5 of the HDMI 1.4 Specification [Ref 12] for more information on the InfoFrame structure.

AUX Packets Handling

This section describes how AUX packets are handled in HDMI TX Subsystem, and the limitations of the subsystem.

In Figure 2-6, the packet types highlighted in blue are handled by hardware and the packet types highlighted in yellow are handled by software.



Packet Type Value	Packet Type	Described in Section
0x00	Null	5.3.2
0x01	Audio Clock Regeneration (N/CTS)	5.3.3
0x02	Audio Sample (L-PCM and IEC 61937 compressed formats)	5.3.4
0x03	General Control	5.3.6
0x04	ACP Packet	5.3.7
0x05	ISRC1 Packet	5.3.8
0x06	ISRC2 Packet	"
0x07	One Bit Audio Sample Packet	5.3.9
0x08	DST Audio Packet	5.3.10
0x09	High Bitrate (HBR) Audio Stream Packet (IEC 61937)	5.3.11
0x0A	Gamut Metadata Packet	5.3.12
0x80+InfoFrame Type	InfoFrame Packet	5.3.5
0x81	Vendor-Specific InfoFrame	8.2.3
0x82	AVI InfoFrame*	8.2.1
0x83	Source Product Descriptor InfoFrame	- -
0x84	Audio InfoFrame*	8.2.2
0x85	MPEG Source InfoFrame	- -

Figure 2-6: **Hardware and Software Packet Types**

For those packet types not highlighted, a generic API function is available in HDMI TX Subsystem driver to support it from user application. You must prepare the complete packet (including calculating CRC) in its application software and call `XV_HdmiTxSs_SendGenericAuxInfoframe` API. The recommended place to push the packet is in HDMI TX Vsync interrupt callback.

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

where,

- InstancePtr is a pointer to HDMI TX Subsystem instance.
- Aux is a 36 byte array contains the complete AUX packet.

Limitations

Only limited fields are supported in current implementation. The supported fields are highlighted in yellow with respective to the packet type.

General Control Packet:

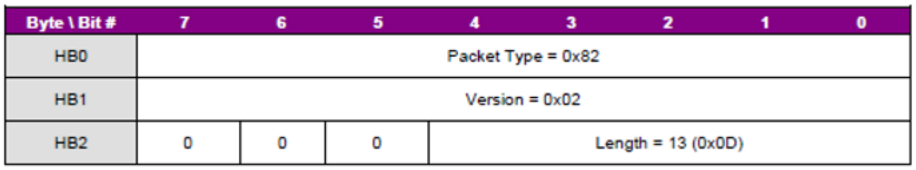
Pixel Packing Phase and Color depth information are retrieved from video stream and sent through general control packet by the driver. However, AVMUTE and Default_Phase fields are not updated and remain as zero always.

Byte \ Bit #	7	6	5	4	3	2	1	0
HB0	0	0	0	0	0	0	1	1
HB1	0	0	0	0	0	0	0	0
HB2	0	0	0	0	0	0	0	0

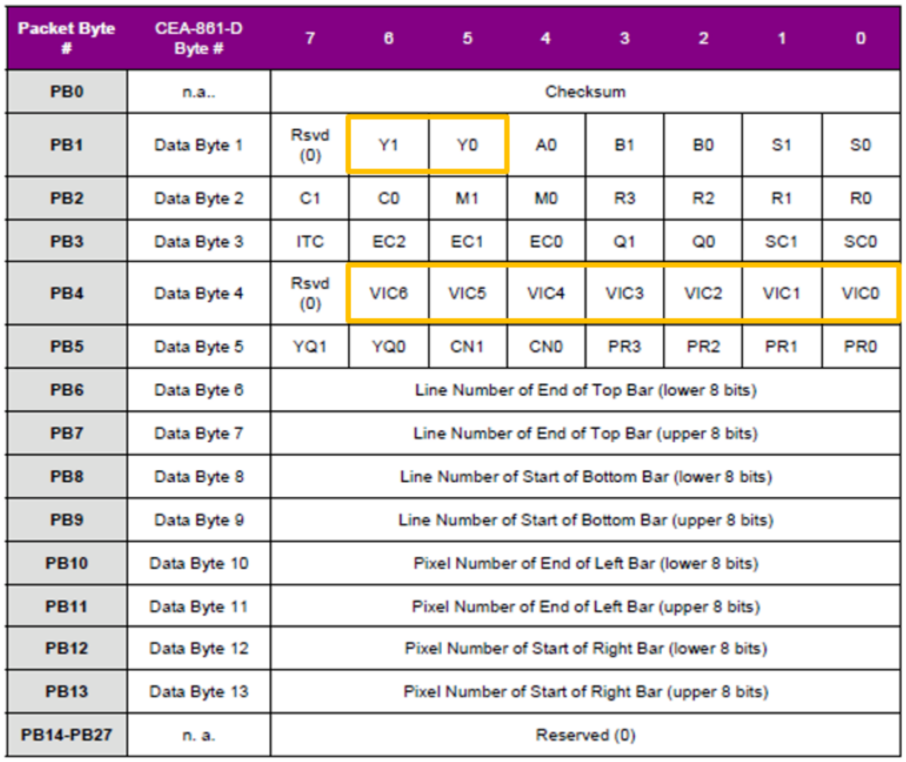
Byte \ Bit #	7	6	5	4	3	2	1	0
SB0	0	0	0	Clear_AVMUTE	0	0	0	Set_AVMUTE
SB1	PP3	PP2	PP1	PP0	CD3	CD2	CD1	CD0
SB2	0	0	0	0	0	0	0	Default_Phase
SB3	0	0	0	0	0	0	0	0
SB4	0	0	0	0	0	0	0	0
SB5	0	0	0	0	0	0	0	0
SB6	0	0	0	0	0	0	0	0

AVI Infoframe

Color space and VIC information are retrieved from video stream and sent through AVI Infoframe. Other fields are not updated and remain as zero always.



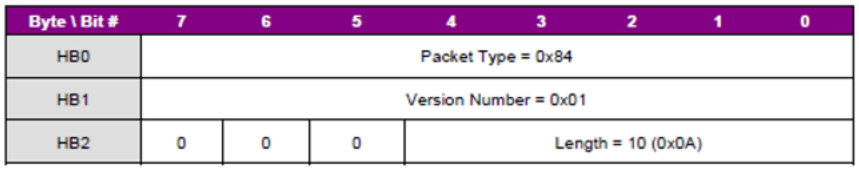
Byte \ Bit #	7	6	5	4	3	2	1	0
HB0	Packet Type = 0x82							
HB1	Version = 0x02							
HB2	0	0	0	Length = 13 (0x0D)				



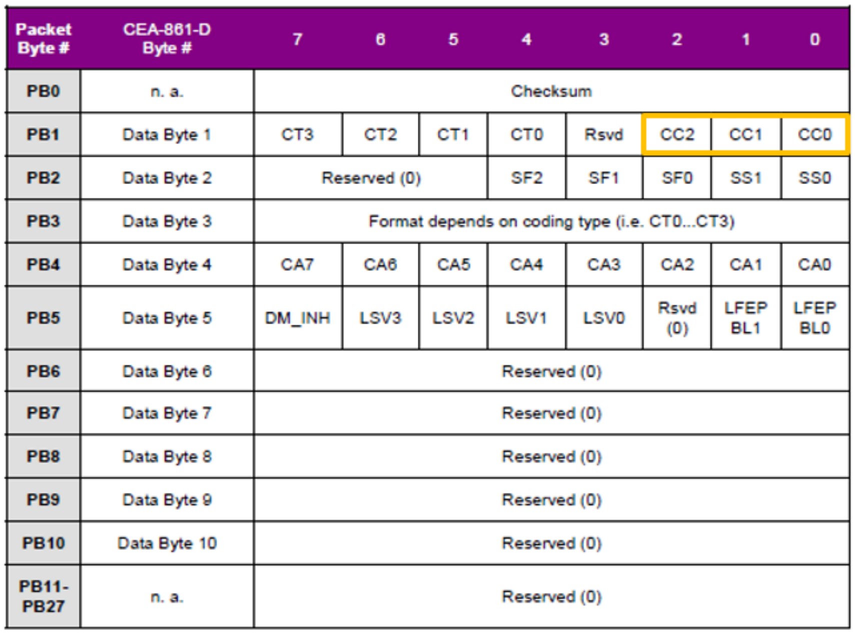
Packet Byte #	CEA-861-D Byte #	7	6	5	4	3	2	1	0
PB0	n.a..	Checksum							
PB1	Data Byte 1	Rsvd (0)	Y1	Y0	A0	B1	B0	S1	S0
PB2	Data Byte 2	C1	C0	M1	M0	R3	R2	R1	R0
PB3	Data Byte 3	ITC	EC2	EC1	EC0	Q1	Q0	SC1	SC0
PB4	Data Byte 4	Rsvd (0)	VIC6	VIC5	VIC4	VIC3	VIC2	VIC1	VIC0
PB5	Data Byte 5	YQ1	YQ0	CN1	CN0	PR3	PR2	PR1	PR0
PB6	Data Byte 6	Line Number of End of Top Bar (lower 8 bits)							
PB7	Data Byte 7	Line Number of End of Top Bar (upper 8 bits)							
PB8	Data Byte 8	Line Number of Start of Bottom Bar (lower 8 bits)							
PB9	Data Byte 9	Line Number of Start of Bottom Bar (upper 8 bits)							
PB10	Data Byte 10	Pixel Number of End of Left Bar (lower 8 bits)							
PB11	Data Byte 11	Pixel Number of End of Left Bar (upper 8 bits)							
PB12	Data Byte 12	Pixel Number of Start of Right Bar (lower 8 bits)							
PB13	Data Byte 13	Pixel Number of Start of Right Bar (upper 8 bits)							
PB14-PB27	n. a.	Reserved (0)							

Audio Infoframe

Only number of channels are set for Audio Infoframe and it is hard-coded to 2 in the HDMI TX Subsystem driver. Other fields are not updated and remain as zero always.



Byte \ Bit #	7	6	5	4	3	2	1	0
HB0	Packet Type = 0x84							
HB1	Version Number = 0x01							
HB2	0	0	0	Length = 10 (0x0A)				



Packet Byte #	CEA-861-D Byte #	7	6	5	4	3	2	1	0
PB0	n. a.	Checksum							
PB1	Data Byte 1	CT3	CT2	CT1	CT0	Rsvd	CC2	CC1	CC0
PB2	Data Byte 2	Reserved (0)			SF2	SF1	SF0	SS1	SS0
PB3	Data Byte 3	Format depends on coding type (i.e. CT0...CT3)							
PB4	Data Byte 4	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0
PB5	Data Byte 5	DM_INH	LSV3	LSV2	LSV1	LSV0	Rsvd (0)	LFEP BL1	LFEP BL0
PB6	Data Byte 6	Reserved (0)							
PB7	Data Byte 7	Reserved (0)							
PB8	Data Byte 8	Reserved (0)							
PB9	Data Byte 9	Reserved (0)							
PB10	Data Byte 10	Reserved (0)							
PB11-PB27	n. a.	Reserved (0)							

HDCP

As part of the HDMI TX Subsystem, the Xilinx® LogiCORE™ IP High-bandwidth Digital Content Protection (HDCP™) transmitters are designed for transmission of audiovisual content securely between two devices that are HDCP capable. In this HDMI TX Subsystem, both HDCP 1.4 and HDCP 2.2 Transmitter IP cores are included. 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.

Figure 2-10 shows a configuration of the HDMI transmitter 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 are active at any given time and the other is passive by calling the relevant HDMI TX Subsystem API functions.

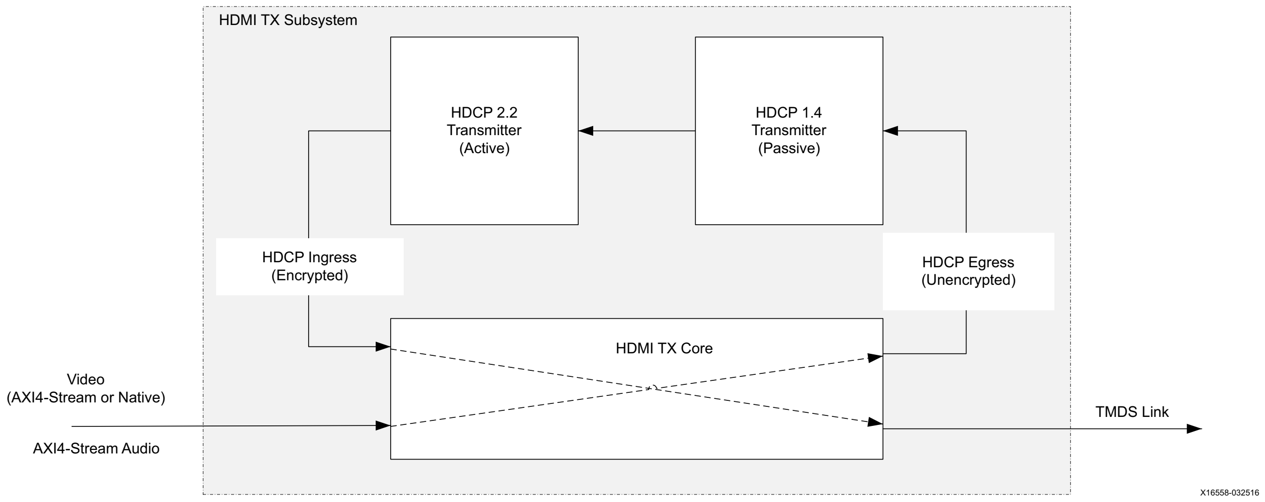

HDMI TX Subsystem
HDCP 2.2 Transmitter (Active)
HDCP 1.4 Transmitter (Passive)
HDCP Ingress (Encrypted)
HDCP Egress (Unencrypted)
HDMI TX Core
Video (AXI4-Stream or Native)
AXI4-Stream Audio
TMDS Link
X16558-032516

Figure 2-10: **HDCP 1.4 and HDCP 2.2 over HDMI Transmitter**

For more details on HDCP, see the *HDCP v1.4 Product Guide* (PG224) [Ref 26] and *HDCP v2.2 Product Guide* (PG249) [Ref 25].

Standards

The HDMI 1.4/2.0 Transmitter Subsystem is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. See the *Vivado AXI Reference Guide* (UG1037) [Ref 1] for additional information. Also, see HDMI specifications [Ref 12].

The HDMI TX Subsystem is compliant with the HDMI 1.4b and HDMI 2.0 specification [Ref 12].

The Xilinx HDCP 1.4 is designed to be compatible with High-bandwidth Digital Content Protection system Revision 1.4 [Ref 13].

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 [Ref 13].

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) [Ref 2]
- *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3]
- *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS182) [Ref 4]
- *Virtex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS183) [Ref 5]
- *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS181) [Ref 6]
- *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS181) [Ref 6]
- *Zynq-7000 All Programmable SoC: DC and AC Switching Characteristics* (DS187) [Ref 7]
- *Zynq-7000 All Programmable SoC: DC and AC Switching Characteristics* (DS191) [Ref 8]
- *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 9]
- *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 10]
- *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 11]

Port Descriptions

Figure 2-11 to Figure 2-14 show the HDMI 1.4/2.0 Transmitter Subsystem ports when AXI4-Stream is selected as video interface. The VIDEO_IN port is expanded in the figure to show the detail AXI4-Stream Video bus signals.

The following subsystem has three default interfaces:

- AXI4-Lite control interface (S_AXI_CPU_IN)
- Video Interface (VIDEO_OUT)
- Audio Interface (AUDIO_OUT)

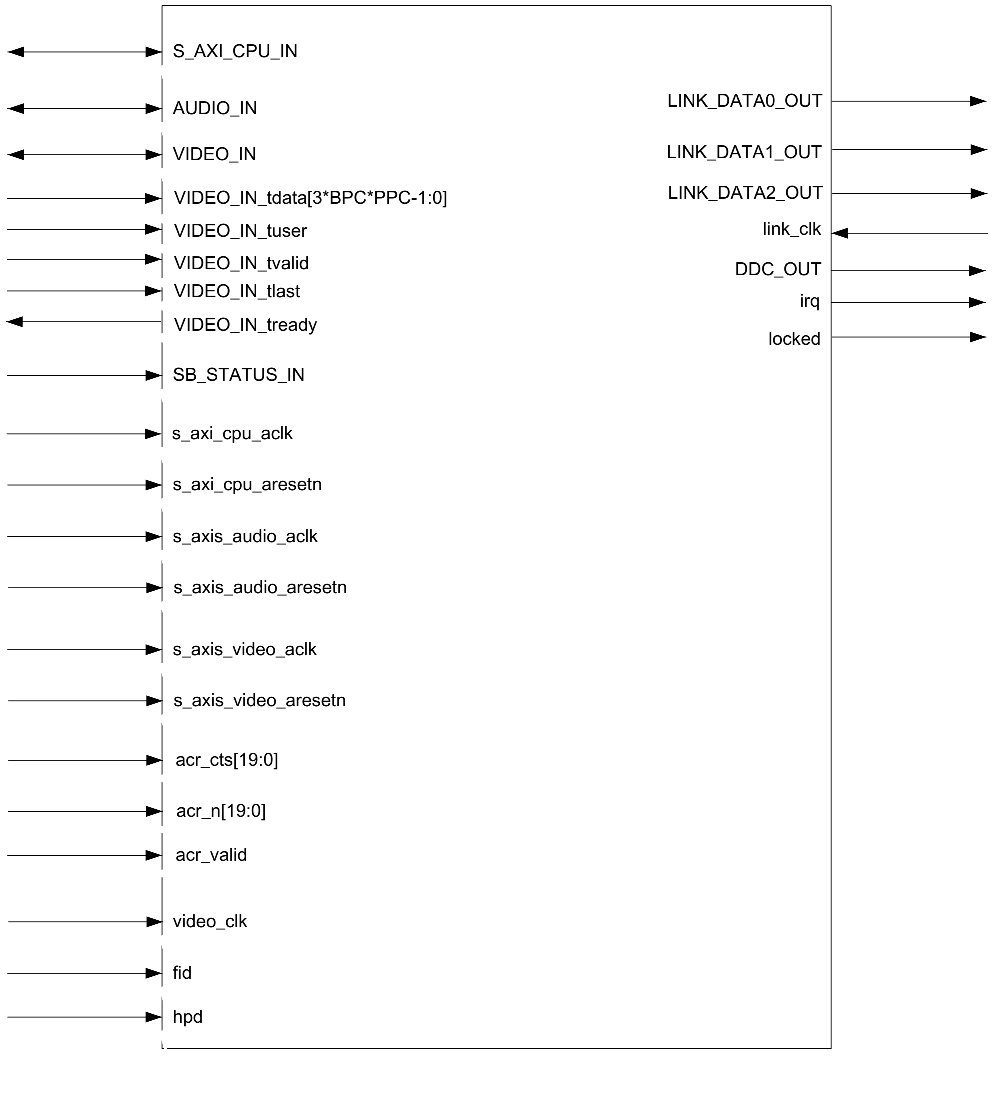


Figure 2-11: **HDMI TX Subsystem Pinout – AXI4-Stream Video Interface (No HDCP)**

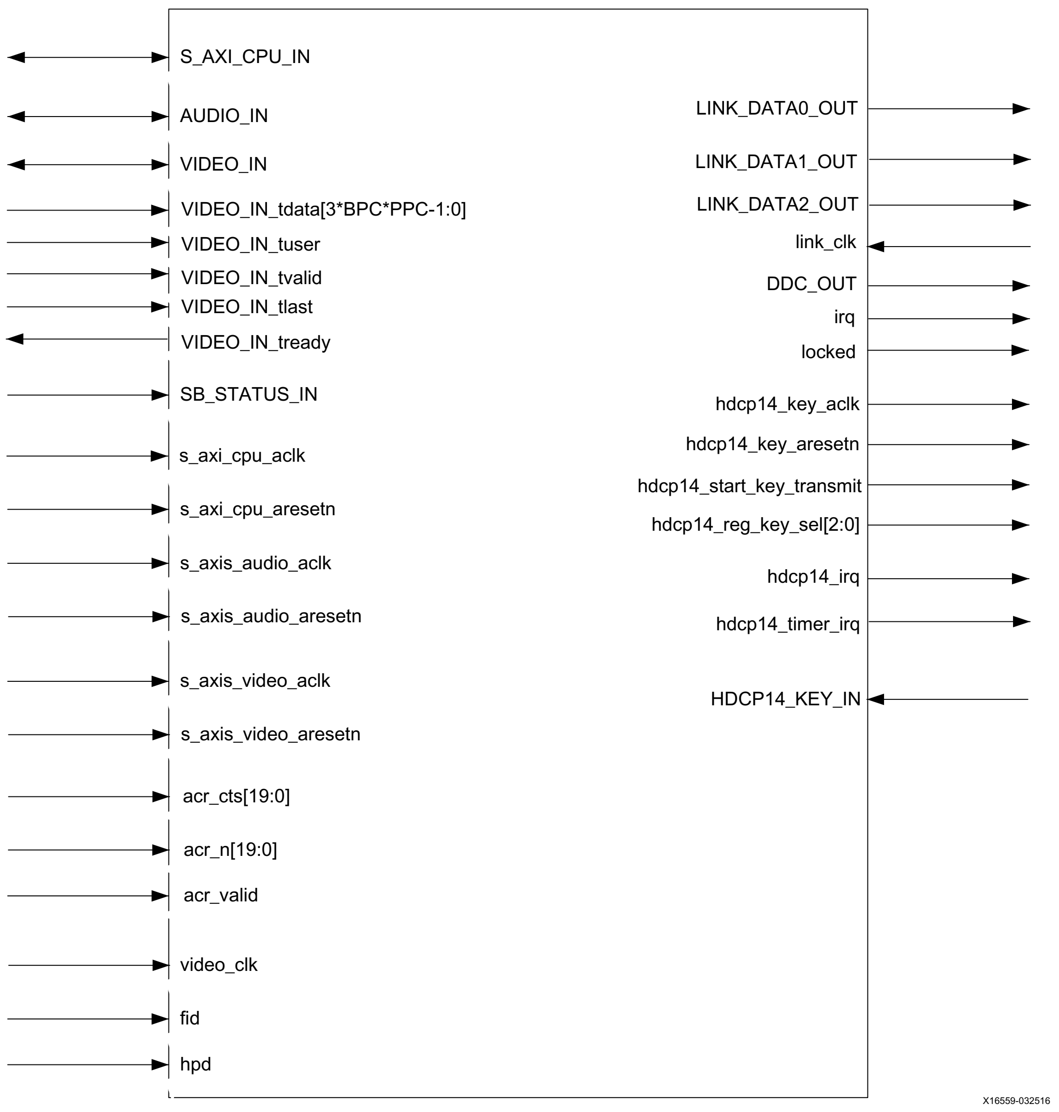


Figure 2-12: **HDMI TX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 1.4 Only)**

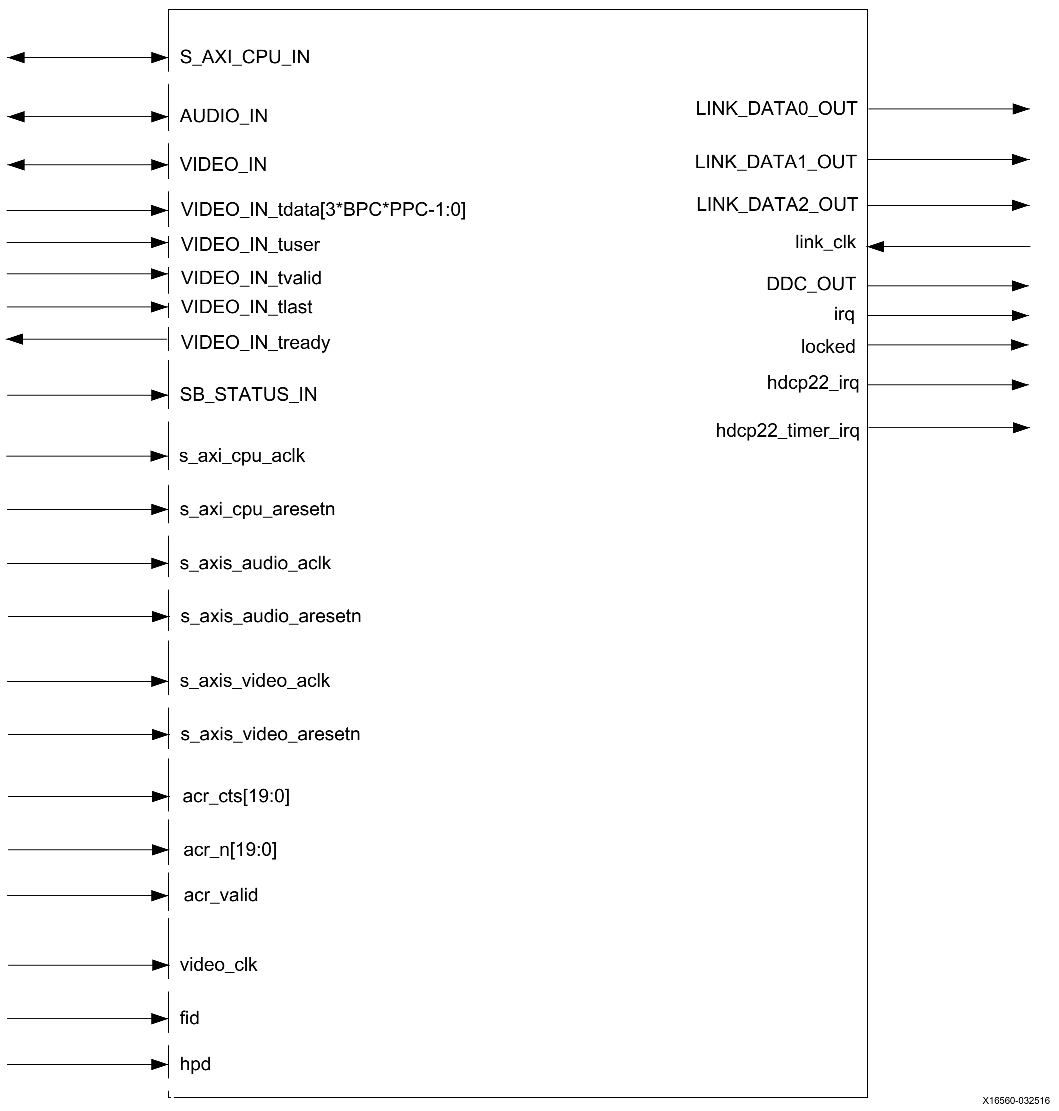


Figure 2-13: **HDMI TX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 2.2 Only)**

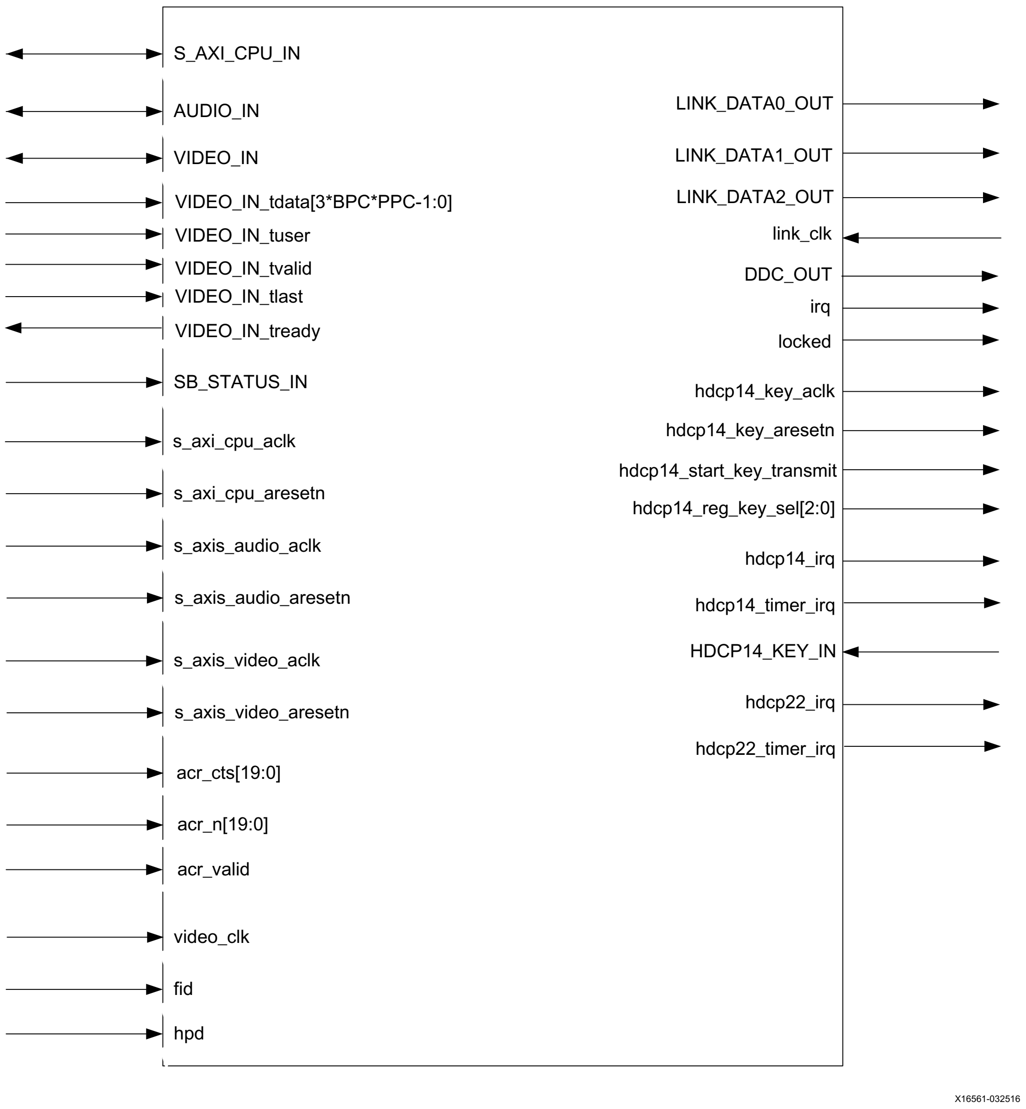


Figure 2-14: **HDMI TX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 1.4 and HDCP 2.2)**

Figure 2-15 to Figure 2-18 show the HDMI 1.4/2.0 Transmitter Subsystem ports when Native Video is selected as video interface. The VIDEO_IN port is expanded in the figure to show the detail Native Video bus signals.

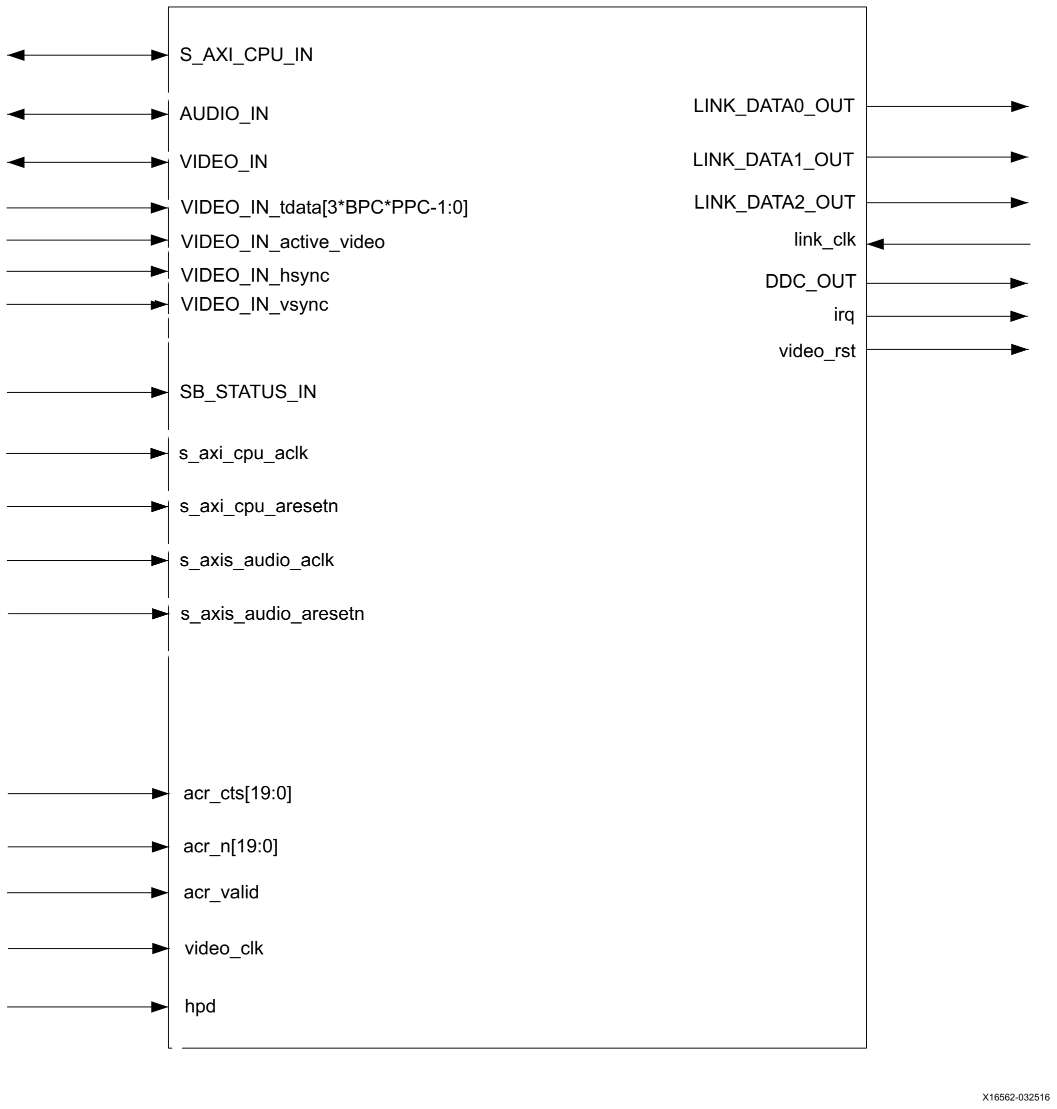


Figure 2-15: **HDMI TX Subsystem Pinout – Native Video Interface (No HDCP)**

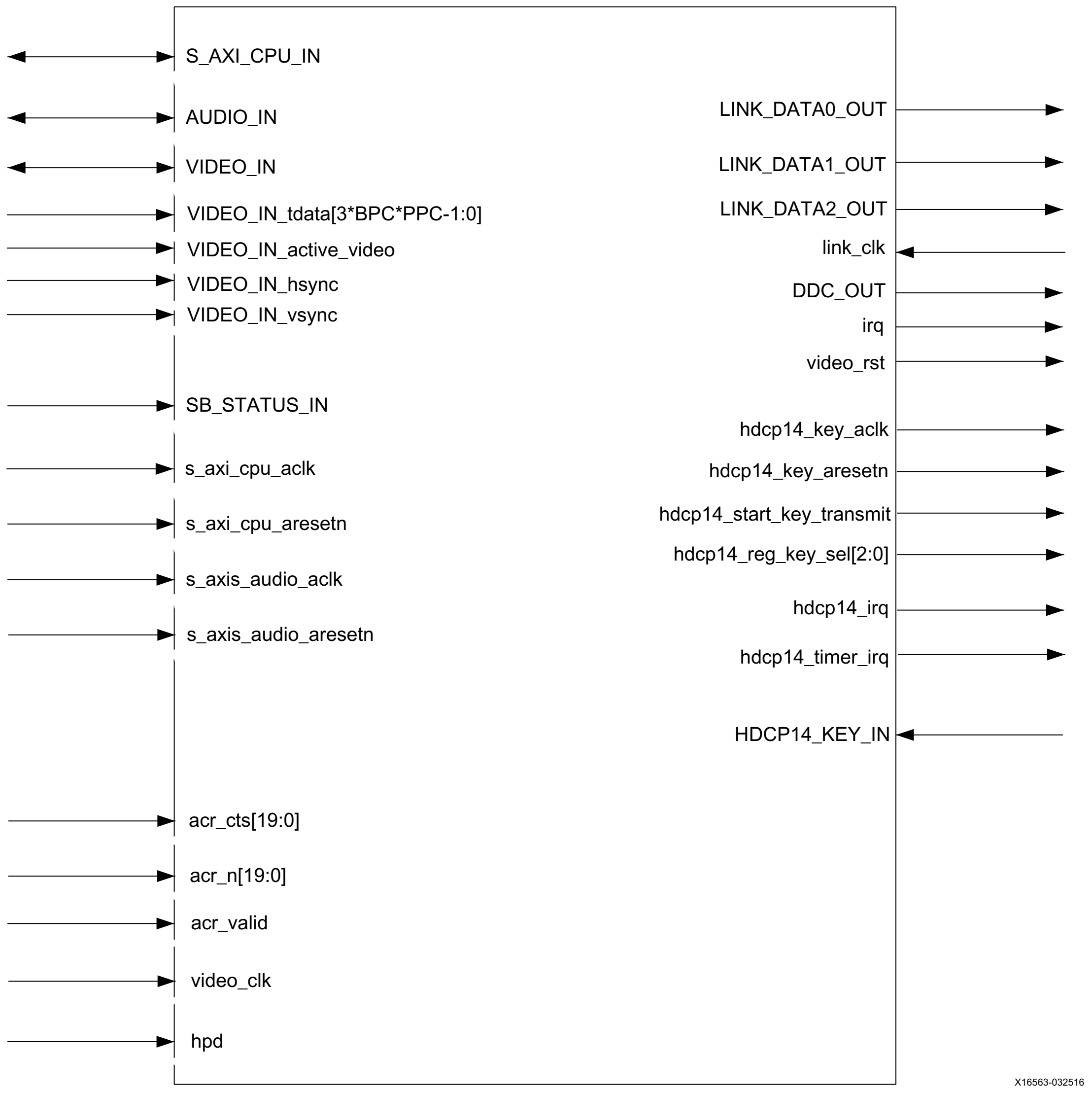


Figure 2-16: **HDMI TX Subsystem Pinout – Native Video Interface (HDCP 1.4 Only)**

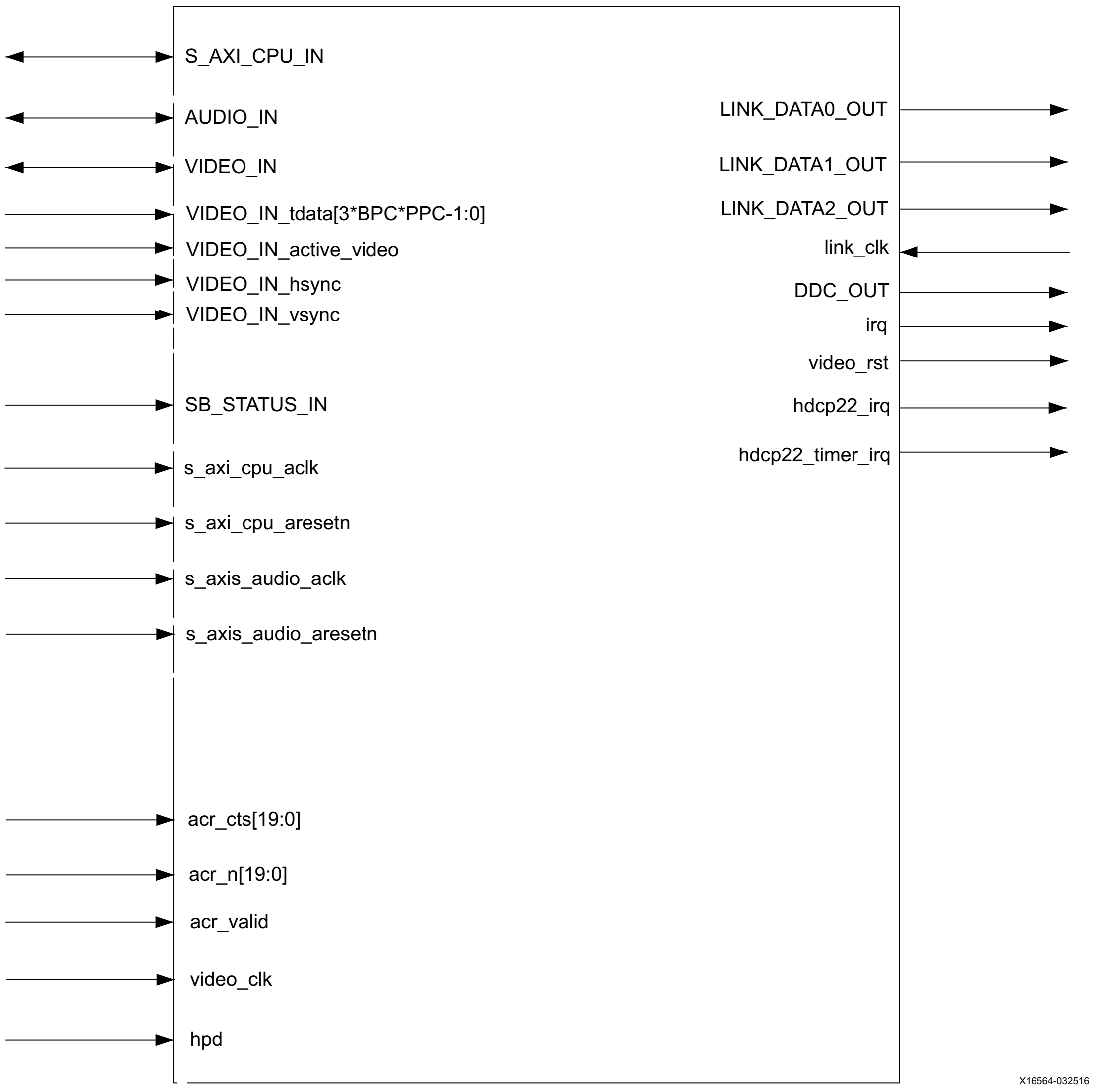


Figure 2-17: **HDMI TX Subsystem Pinout – Native Video Interface (HDCP 2.2 Only)**

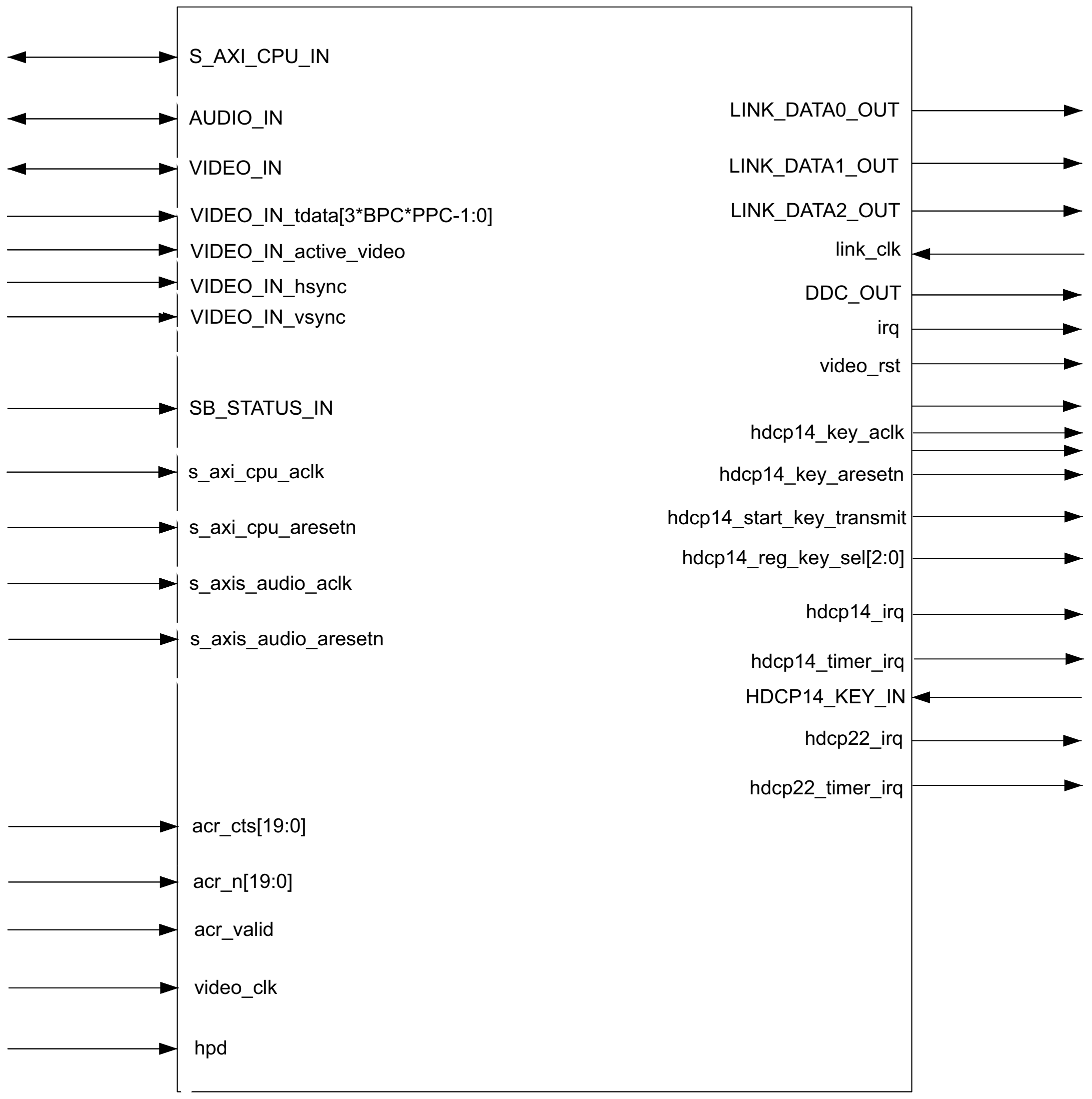


Figure 2-18: **HDMI TX Subsystem Pinout – Native Video Interface (HDCP 1.4 and HDCP 2.2)**

CPU Interface

Table 2-1 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: *The direct register level access to any of the submodules is not supported. Instead, all the accesses are done through driver APIs.*

Table 2-1: **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
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

Video Input Stream Interface

This HDMI 1.4/2.0 Transmitter Subsystem is supporting two types of video input stream interfaces, which eventually is mapped to HDMI 1.4/2.0 Transmitter Subsystem VIDEO_IN interface.

- AXI4-Stream Video interface
- Native Video Interface

Table 2-2 shows the signals for AXI4-Stream video input streaming 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 2-2: **Video Input Stream 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	3*BPC*PPC	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

Native Video Input Interface

Table 2-3 shows the signals for Native video input interface. This interface is a standard video interface and runs at `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 2-3: **Native Video InputInterface**

Name	Direction	Width	Description
video_clk	Input	1	Video clock
VIDEO_IN_active_video(4)	Input	1	Active video
VIDEO_IN_data	Input	3*BPC*PPC	Data
VIDEO_IN_hsync(4)	Input	1	Horizontal sync

Table 2-3: **Native Video InputInterface *(Cont'd)***

Name	Direction	Width	Description
VIDEO_IN_vsync(4)	Input	1	Vertical sync

Notes:

1. When native video interface is selected, s_axis_video_aclk and s_axis_video_aresetn are removed from the HDMI 1.4/2.0 Transmitter Subsystem interface ports.
2. video_clk is generated by *Video PHY Controller LogiCORE IP Product Guide* (PG230) [Ref 24].
3. When native video interface is selected, there is no hardware reset.
4. You must provide the correct video timing information. You can choose to use Xilinx Video Timing Controller (vtc) or design your own vtc module to generate the timing control signals for native video interface.

Audio Input Stream Interface

Table 2-4 shows the signals for AXI4-Stream audio input streaming interfaces. The audio interface transports 24-bits 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 2-4: **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)
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 [Ref 12].

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.

Table 2-5 shows the Audio Clock Regeneration (ACR) interface signals. This interface runs at the `s_axis_audio_aclk` clock rate.

Table 2-5: **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 a HDMI system does not require audio, tie the following input ports to LOW:

- AUDIO_IN (except tready)
- s_axis_audio_aresetn
- s_axis_audio_aclk
- acr_cts
- acr_n
- acr_valid

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 may 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 and uncompress the data audio with your custom logic.

IMPORTANT: *L-PCM (IEC60958) Audio is the only Audio format tested on board by Xilinx only in Example Design.*

HDMI Link Output Interface

Table 2-6 shows the HDMI Link Output interface signals. This interface runs at the `link_clk` clock rate.

Table 2-6: **HDMI Link Output Interface**

Name	Direction	Width	Description
link_clk	Input	1	Link clock
LINK_DATA0_OUT_tdata	Output	40	Link data 0
LINK_DATA0_OUT_tvalid	Output	1	Link Data 0 Valid
LINK_DATA1_OUT_tdata	Output	40	Link data 1
LINK_DATA1_OUT_tvalid	Output	1	Link Data 1 Valid
LINK_DATA2_OUT_tdata	Output	40	Link data 2
LINK_DATA2_OUT_tvalid	Output	1	Link Data 2 Valid

Data Display Channel Interface

Table 2-7 shows the Data Display Channel interface signals.

Table 2-7: **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)

Table 2-8 shows the signals for HDCP 1.4 key interface. This interface runs at the `hdcp14_key_aclk` (which is running at AXI4 Lite Clock).

Table 2-8: **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

Table 2-8: **HDCP 1.4 Key Input Interface *(Cont'd)***

Name	Direction	Width	Description
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. Figure 2-19 shows an example of how the HDMI 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 TX Subsystem. For HDCP 1.4 design details, see the *HDCP v1.4 Product Guide* (PG224) [Ref 26].

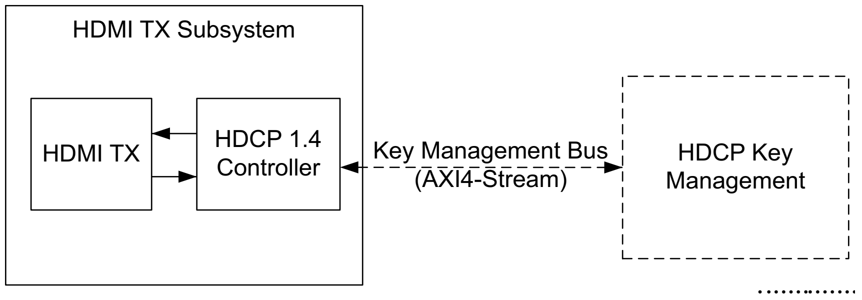


Figure 2-19: **HDCP 1.4 Key Management Bus (AXI4-Stream)**

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 v2.2 Product Guide* (PG249) [Ref 25].

HDCP 2.2 Interrupt Outputs

Table 2-9 shows the signals for HDCP 2.2 interrupt output ports.

Table 2-9: **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

Table 2-10 shows the miscellaneous signals with AXI4-Stream video interface selected.

Table 2-10: **Miscellaneous Signals with AXI4-Stream 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
locked	Output	1	Flag indicating the subsystem is locked to the incoming video steam. 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 TX Subsystem to convert AXI4-Stream Video into Native Video. HDMI TX core uses this video_clk to clock in the Video Data.
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
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.

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 HDMI Transmitter Subsystem GUI.

Miscellaneous Signals with Native Video Interface

Table 2-11 shows the miscellaneous signals with native video interface selected.

Table 2-11: **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.

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 HDMI Transmitter Subsystem GUI.

Clocks and Resets

Table 2-12 provides an overview of the clocks and resets. See Clocking and Resets in Chapter 3 for more information.

Table 2-12: **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.

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. Therefore, you need to fully understand the system and adjust the subsystem parameters accordingly. Appendix C, Application Software Development describes how to integrate the subsystem API into a software application.

Audio Data Stream

An AXI4-Stream audio cycle is illustrated in Figure 3-1. The data is captured when both the valid (TVLD) and ready (TRDY) signals are asserted. The HDMI 1.4/2.0 Transmitter 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).

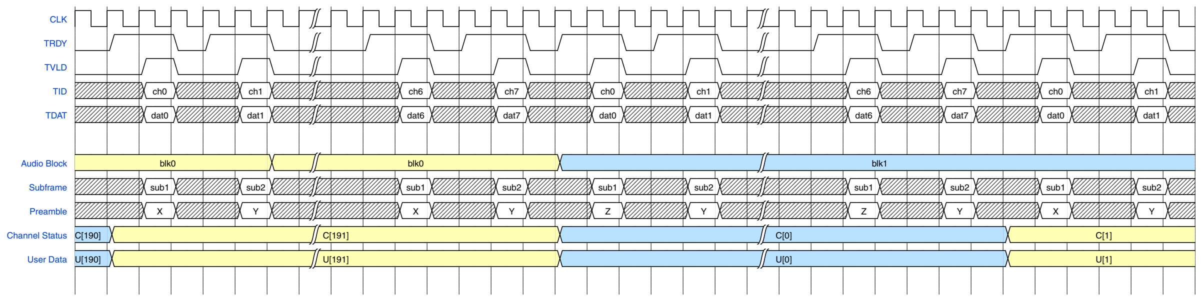


Figure 3-1: **Audio Cycle**

In HDMI 1.4/2.0 Transmitter Subsystem, the number of Audio Channels is set through the software driver. You must enable the correct number of audio channel according to your use case and send the corresponding audio channel data mapping to the channel ID (TID). For example, if you intend to send out 8 channel audio, then you must set Audio Channel number to 8 in HDMI 1.4/2.0 Transmitter Subsystem driver. Then, the corresponding audio data must be prepared and sent to HDMI 1.4/2.0 Transmitter Subsystem in the hardware, as described in Figure 3-1.

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 may appear in unexpected channel locations.*

In HDMI TX Subsystem, L-PCM (IEC 60958, Packet Type 0x02) and HBR (Packet Type 0x09) are handled by the hardware. An API function is available that allows for the setting of the audio format, and the hardware packs the audio based on the Audio Format selected.

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

where:

InstancePtr is a pointer to XV_HdmiTxSs instance

format is a selector of Audio Format

- 1:HBR
- 0:L-PCM

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 and uncompress the data audio with your custom logic.

IMPORTANT: *L-PCM (IEC60958) Audio is the only Audio format tested on board by Xilinx only in Example Design.*

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, Figure 3-2 shows the data format for quad pixels per clock to be fully compliant with the AXI4-Stream video protocol. A data format for a fully compliant AXI4-Stream video protocol dual pixels per clock is illustrated in Figure 3-3.

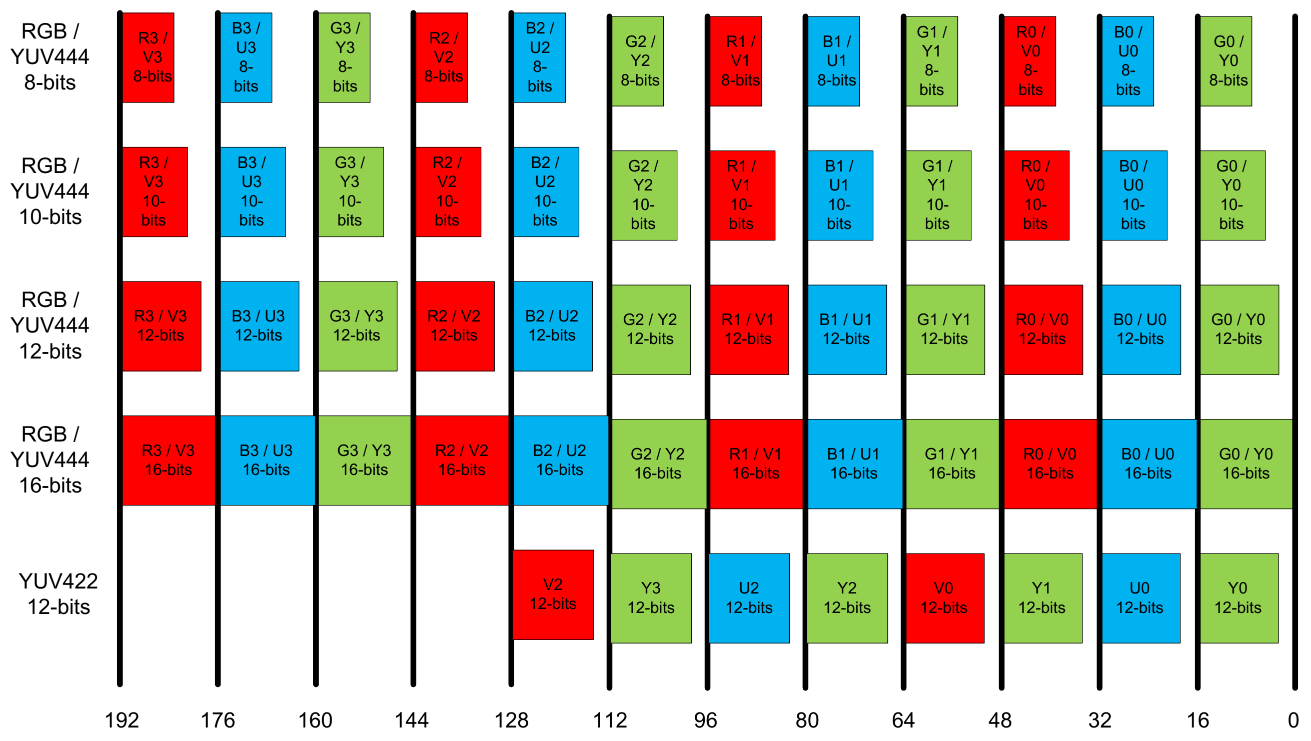


Figure 3-2: **Quad Pixels Data Format (Max Bits Per Component = 16)**

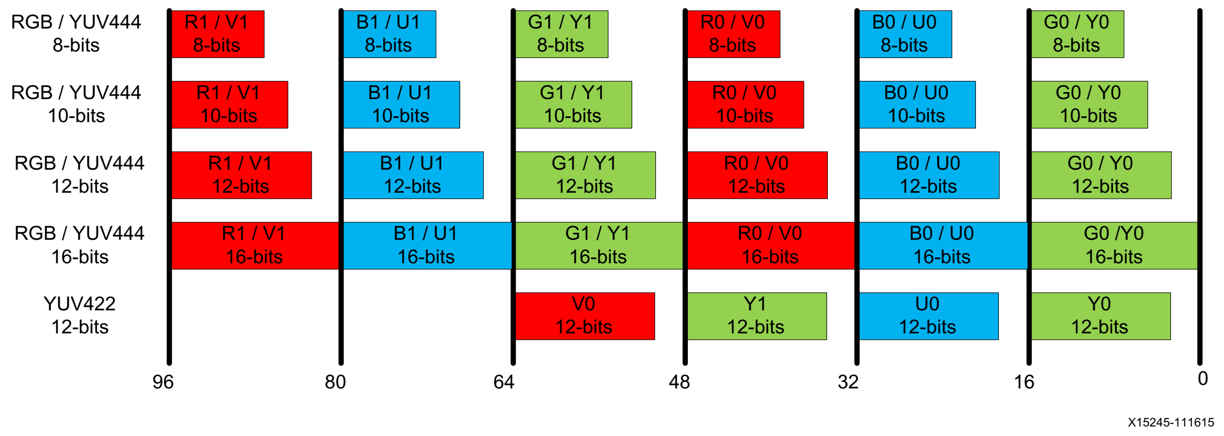


Figure 3-3: **Dual Pixels Data Format (Max Bits Per Component = 16)**

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 3-1: **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, Figure 3-4 shows the data format for quad pixels per clock to be fully compliant with the AXI4-Stream video protocol. A data format for a fully compliant AXI4-Stream video protocol with dual pixels per clock is illustrated in Figure 3-5.

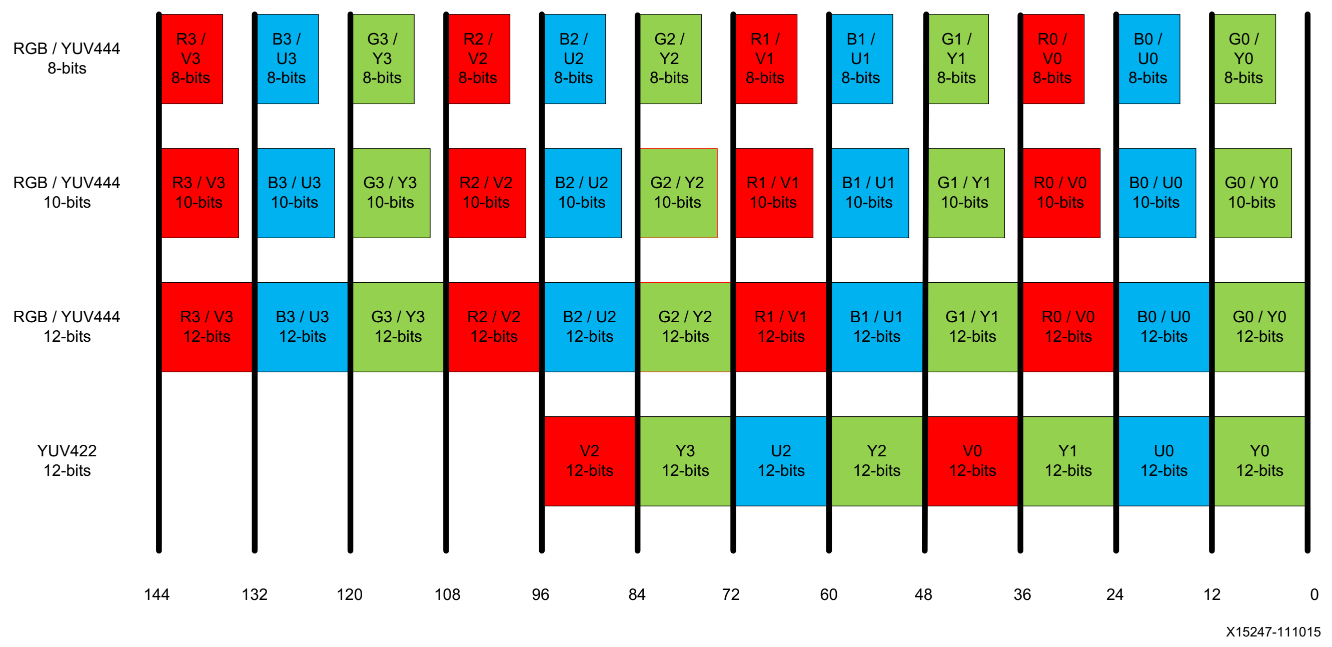


Figure 3-4: **Quad Pixels Data Format (Max Bits Per Component = 12)**

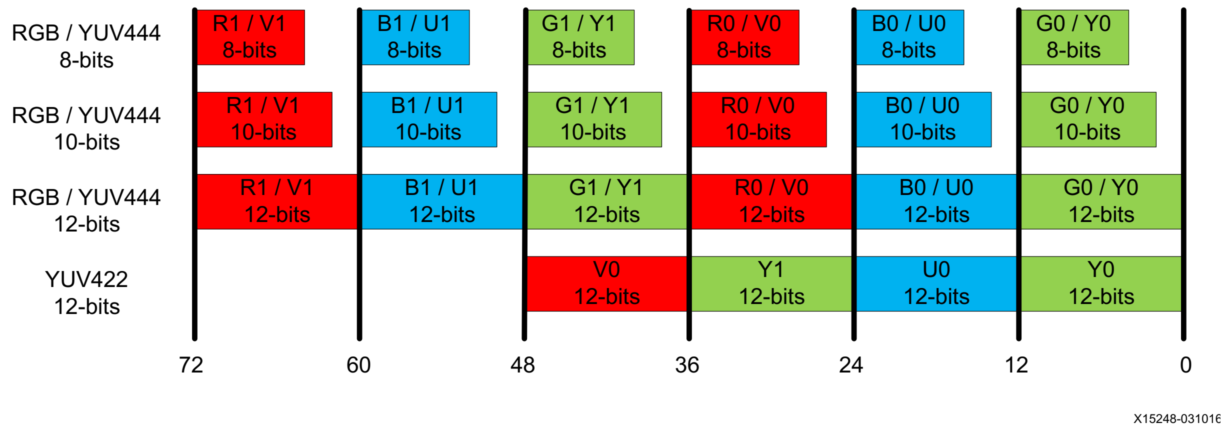


Figure 3-5: **Dual Pixels Data Format (Max Bits Per Component = 12)**

The video interface can also transport quad and dual pixels in the YUV420 color space. However the current data format is not complaint with the AXI4-Stream video protocol. Figure 3-6 and Figure 3-7 show the data format for quad and dual pixels formats.

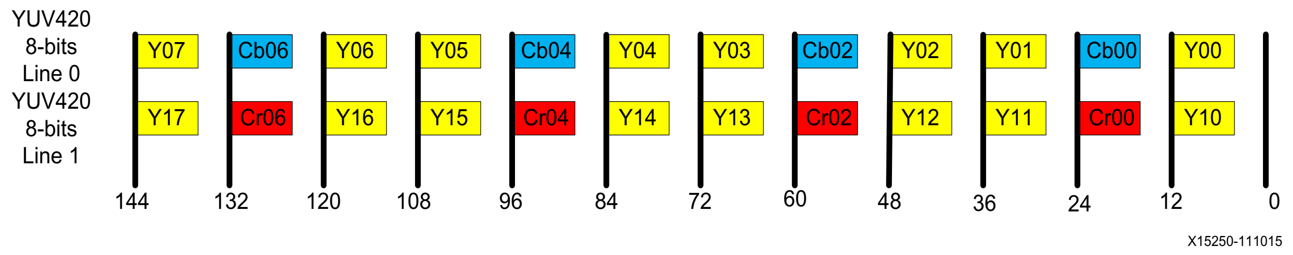


Figure 3-6: **YUV420 Color Space Quad Pixels Data Format**

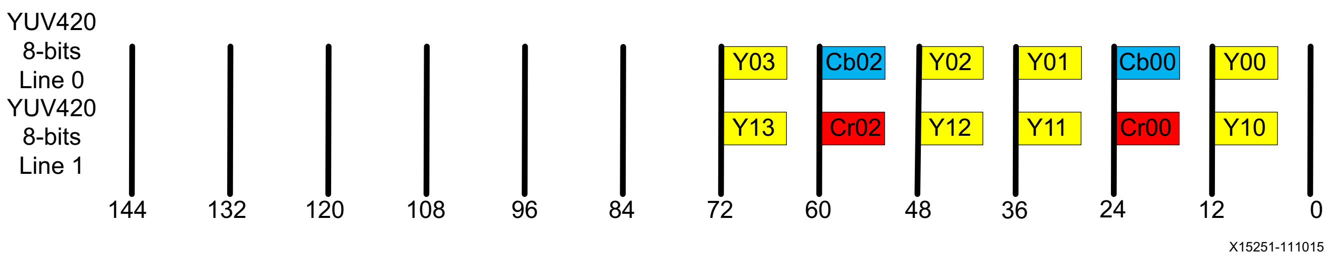


Figure 3-7: **YUV420 Color Space Dual Pixels Data Format**

Similarly, for YUV 4:2:0 deep color (10, 12, or 16 bits), the data representation is the same as shown in Figure 3-6 and Figure 3-7. The only difference is that each component carries more bits (10, 12, and 16). To make the YUV 4:2:0 compatible with *AXI4-Stream Video IP and System Design Guide* [Ref 14], enable it from the HDMI Transmitter Subsystem GUI.

Using an 8-bit video as an example, Figure 3-8 illustrates the YUV 4:2:0 AXI4-Stream video data representation in *AXI4-Stream Video IP and System Design Guide* [Ref 14].

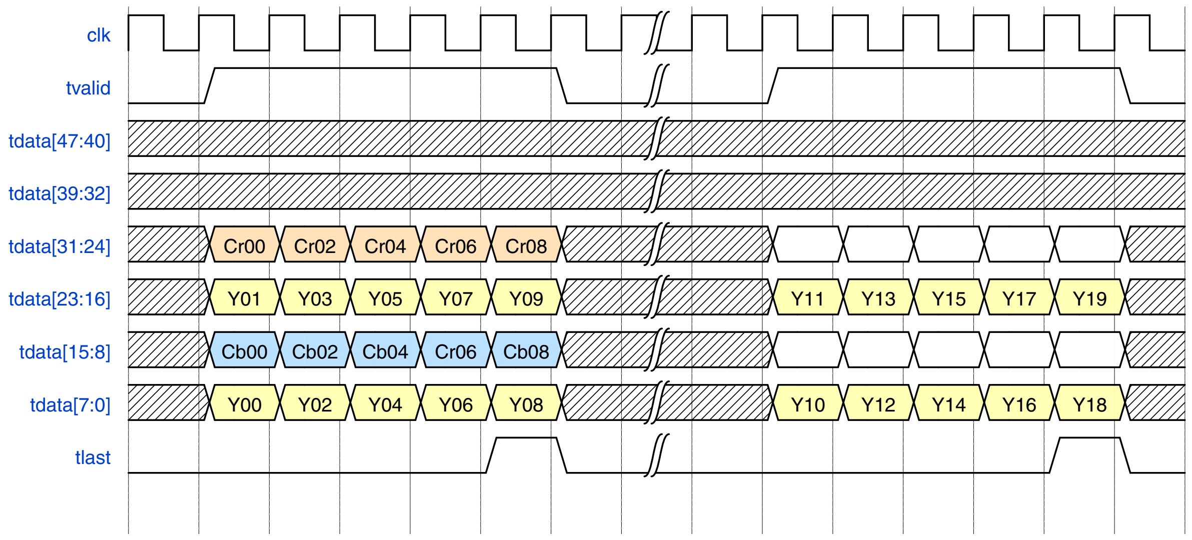


Figure 3-8: **YUV 4:2:0 AXI4-Stream Video Data (Dual Pixel per Clock)**

However, in the native HDMI video interface, the video data representation must be as shown in Figure 3-9.

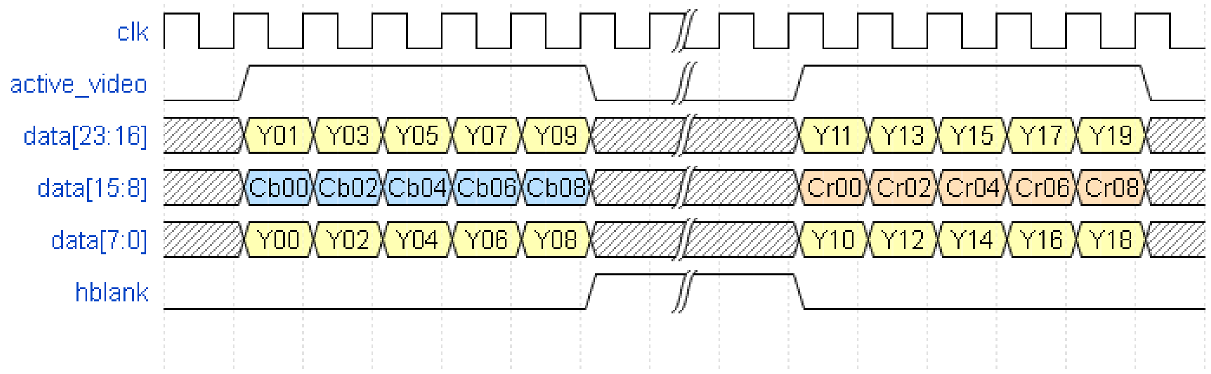


Figure 3-9: **Native HDMI Video Interface**

Therefore, a remapping feature is added to HDMI 1.4/2.0 Transmitter Subsystem to convert HDMI native video into AXI4-Stream video.

Note: For RGB/YUV444/YUV422 formats, video data is directly mapped from AXI4-Stream to Native Video interface without any line buffer. Therefore, Figure 3-2 to Figure 3-5 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 meeting timing requirements. Therefore the quad pixels per clock data mapping is recommended for design 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) [Ref 14].

Interlaced Video

The HDMI 1.4/2.0 Transmitter 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 Transmitter Subsystem, which is directly connected to the HDMI TX core inside the HDMI 1.4/2.0 Transmitter Subsystem.

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

Taking 1920x1080@50Hz (I) as an example, the detail timing information is shown in Table 3-2.

Table 3-2: **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 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
- Frame Rate = Field Rate / 2.

In this example,

VActive = 1080/2 = 540

Field Rate = 50Hz

Frame Rate = 50/2 = 25Hz

To design using the AXI4-Stream Interface, generate two fields of video with timing using the values from Table 3-2. For complete timing information, refer to CEA-861-F [Ref 28]. Only active video data compliant with AXI4-Stream protocol is needed. The AXI4-Stream to Video Out core inside HDMI 1.4/2.0 Transmitter 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 *AXI4-Stream to Video Out LogiCORE IP Product Guide* (PG044) [Ref 27].

To design using the Native Interface, generate two fields native video with timing using the values from Table 3-2. Ensure that the HSYNC and VSYNC are driven using the values from Table 3-2. Since a frame may have odd number of lines (e.g. 1125 for 1080i50), the two fields may result in a different total number of lines (e.g. 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 (e.g. 13.5 for 480i/NTSC) can be transmitted using pixel-repetition scheme.

In the HDMI 1.4/2.0 Transmitter Subsystem,

- Pixel repetition of two is supported for NTSC/480i60 or PAL/576i50
- Enabled through GUI parameter upon IP generation
- Pixel repetition is only available for AXI4-Stream Interface

After it is enabled in the HDMI 1.4/2.0 Transmitter Subsystem, you must prepare interlaced video as normal, then the HDMI 1.4/2.0 Transmitter 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 @ 30Hz, which is made from two fields of 720x240 @ 30Hz video.

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

Clocking

The `S_AXI_CPU_IN`, `VIDEO_OUT`, and `AUDIO_OUT` 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 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: *The AXI4-Lite CPU clock must run at 100 Mhz.*

The HDMI clock structure is illustrated in Figure 3-10 and Table 3-3.

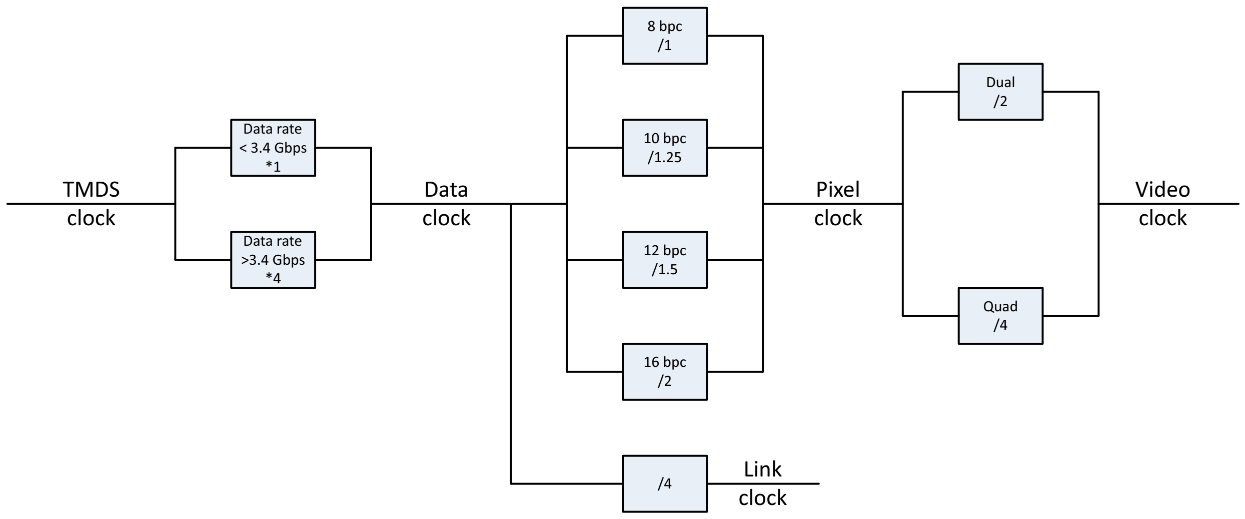


Figure 3-10: **HDMI Clocking Structure**

Table 3-3: **Clocking**

HDMI Clocking			
Clock	**Function**	**Freq/Rate**	**Example**(1)
TMDS clock	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) = 1/40 data rate (for data rates > 3.4 Gb/s)	Data rate = 2.97 Gb/s TMDS clock = 2.97/10 = 297 MHz Data rate = 5.94 Gb/s TMDS clock = 5.94/40 = 148.5 MHz
Data clock	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) = TMDS clock * 4 (for data rates > 3.4 Gb/s)	Data rate = 2.97 Gb/s Data clock = TMDS clock * 1 = 297 MHz Data rate = 5.94 Gb/s Data clock = TMDS clock * 4 = 594 MHz TMDS clock = 148.5 MHz
Link clock	Clock used for data interface between HDMI PHY Layer Module and subsystem	= 1/4 of data clock	TMDS clock = 297 MHz Data clock = 297 MHz Link clock = 297 MHz/4 = 74.25 MHz Data clock = 594 MHz Link clock = 594 MHz/4 = 148.5 MHz

Pixel clock	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 10 bpc pixel clock = data clock/1.25
		for 12 bpc pixel clock = data clock/1.5
		for 16 bpc pixel clock = data clock/2	

Video clock	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) [Ref 24].

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 Transmitter Subsystem with different GT devices, calculate the clock frequencies and make sure the targeted device is able to support it. When using the HDMI 1.4/2.0 Transmitter Subsystem with Xilinx Video PHY Controller IP core, more information can be found in *Video PHY Controller LogiCORE IP Product Guide* (PG230) [Ref 24].

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

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

Pixel clock represents the total number of pixels need to be sent every second. Therefore,

Pixel clock = Htotal × Vtotal × Frame Rate
=2200 x 1125 x 60
=148,500,000
= 148.5Mhz

Link clock = (Data clock)/4=222.75/4=55.6875Mhz

Video clock = (Pixel clock)/PPC=148.5/2=74.25Mhz

Data clock = Pixel clock × BPC/8=148.5× 12/8=222.75Mhz

Using the associative property in this example,

Data clock = 222.75Mhz < 340Mhz

then

TMDS clock = Data clock = 222.75Mhz

Figure shows how the clock is distributed in HDMI TX Subsystem and the relationship to the Video PHY Controller.

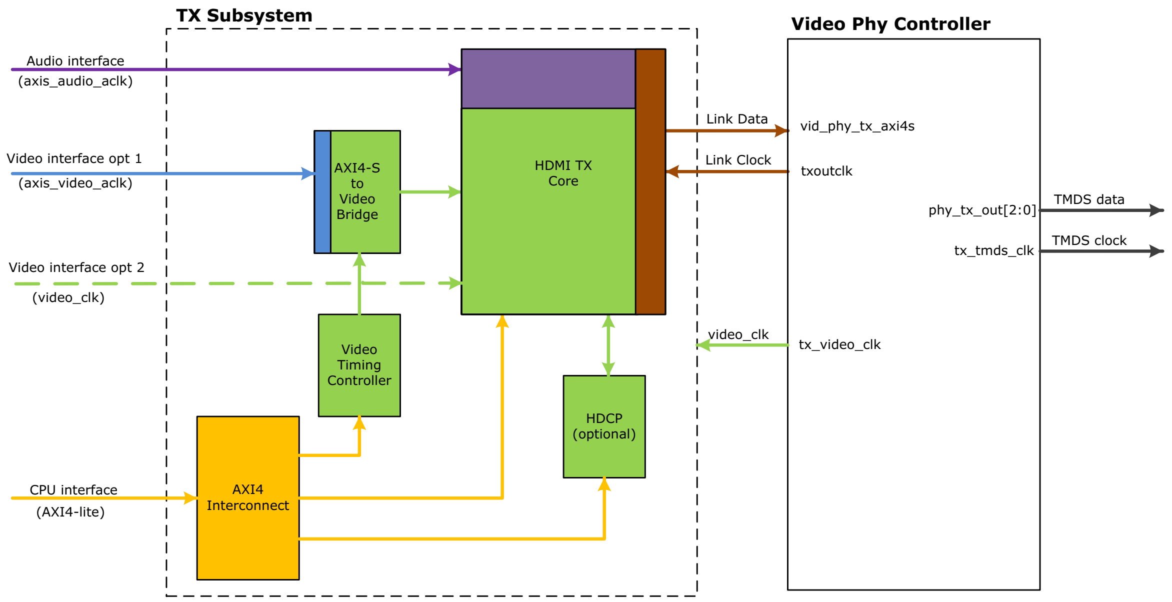


Figure 3-11: **HDMI Transmitter Subsystem and Video PHY Controller**

The HDMI TX Subsystem is able to support either AXI4-Stream Video or Native Video.

- When **AXI4-Stream** is selected, the video stream is sent to HDMI TX Subsystem through Video Interface in AXI4-Stream format running at axis_video_aclk. The AXI4-Stream is then processed and converted into Native Video stream by AXI4-Stream to Video Out Bridge core with the help of 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 pack the native video data with Audio data and other auxiliary data into Link Data and sent to Video PHY Controller at Link Clock.
- When **Native Interface** is selected, Video stream is sent to HDMI TX Subsystem as native video and directly passed to HDMI TX core. The data are then packed and binded with Audio data and other auxiliary data into Link Data and sent to Video PHY Controller at Link Clock.

Based on the system requirement, Video PHY Controller is generating Link Clock and Video Clock for HDMI 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 clock wizard with reference to some on-board oscillators.

Resets

Each AXI 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 VIDEO_IN interface when Native Video interface is selected. However, HDMI TX Subsystem outputs a `video_rst` signal, which you can use to reset its Native Video Source generation modules.

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) [Ref 15]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 16]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 17]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 18]

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 Transmitter Subsystem can be added to a Vivado IP integrator block design in the Vivado Design Suite and can be customized using IP catalog. For more detailed information on customizing and generating the subsystem in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 15]. 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 subsystem 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.

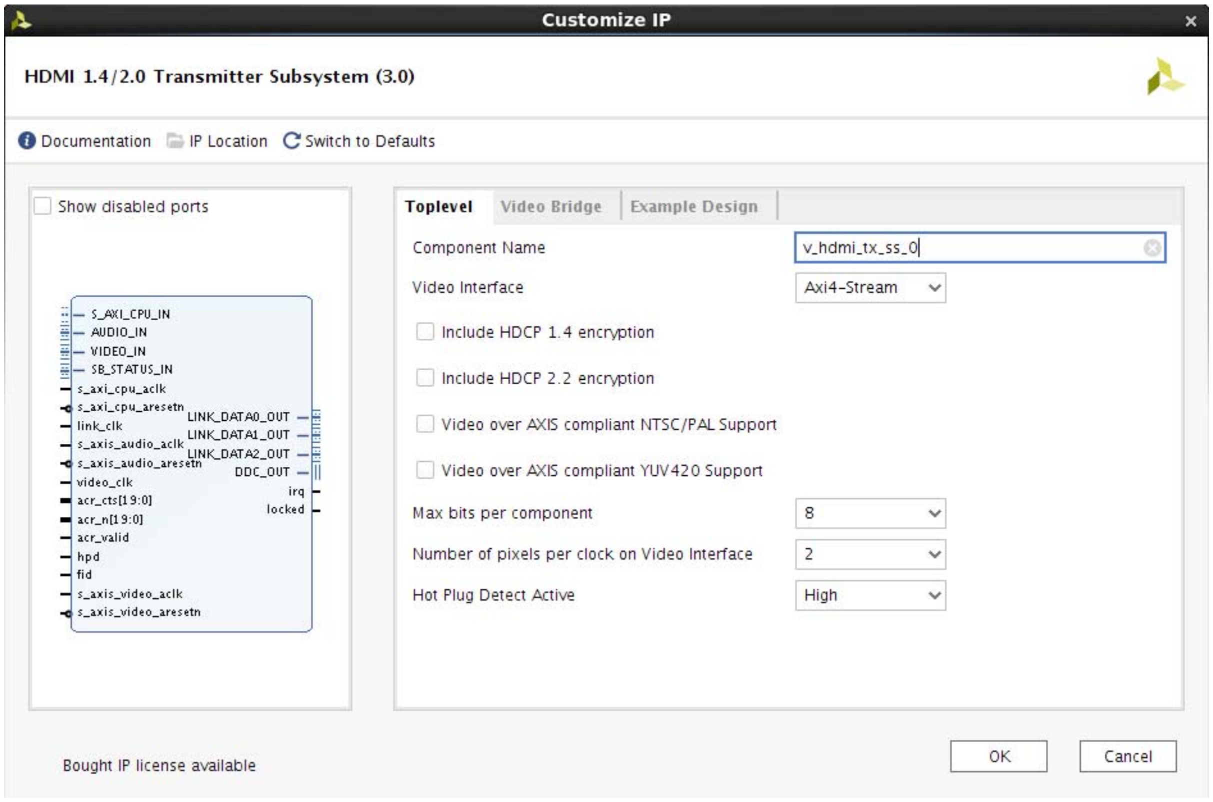
3. Click on the IP name and press the Enter key on your keyboard or double click on the IP name.
4. Double-click the selected IP block or select the **Customize Block** command from the right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 16] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 17].

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

Top Level Tab

The Top level tab is shown in Figure 4-1.



HDMI 1.4/2.0 Transmitter Subsystem (3.0)

Component Name: v_hdmi_tx_ss_0
Video Interface: Axi4-Stream
[ ] Include HDCP 1.4 encryption
[ ] Include HDCP 2.2 encryption
[ ] Video over AXIS compliant NTSC/PAL Support
[ ] Video over AXIS compliant YUV420 Support
Max bits per component: 8
Number of pixels per clock on Video Interface: 2
Hot Plug Detect Active: High

Bought IP license available

Figure 4-1: **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 TX subsystem. The allowable options are **AXIS-Stream** or **Native Video**.

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

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

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. See Table 3-1 for more details on bit mapping.

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 resolution 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 this kind of video formats, ensure that **PPC=2** is selected in Vivado.*

Video over AXIS compliant NTSC/PAL Support: This option enables the HDMI 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.

Video over AXIS compliant YUV420 Support: This option enables the HDMI TX subsystem to support Video over AXIS compliant YUV420.

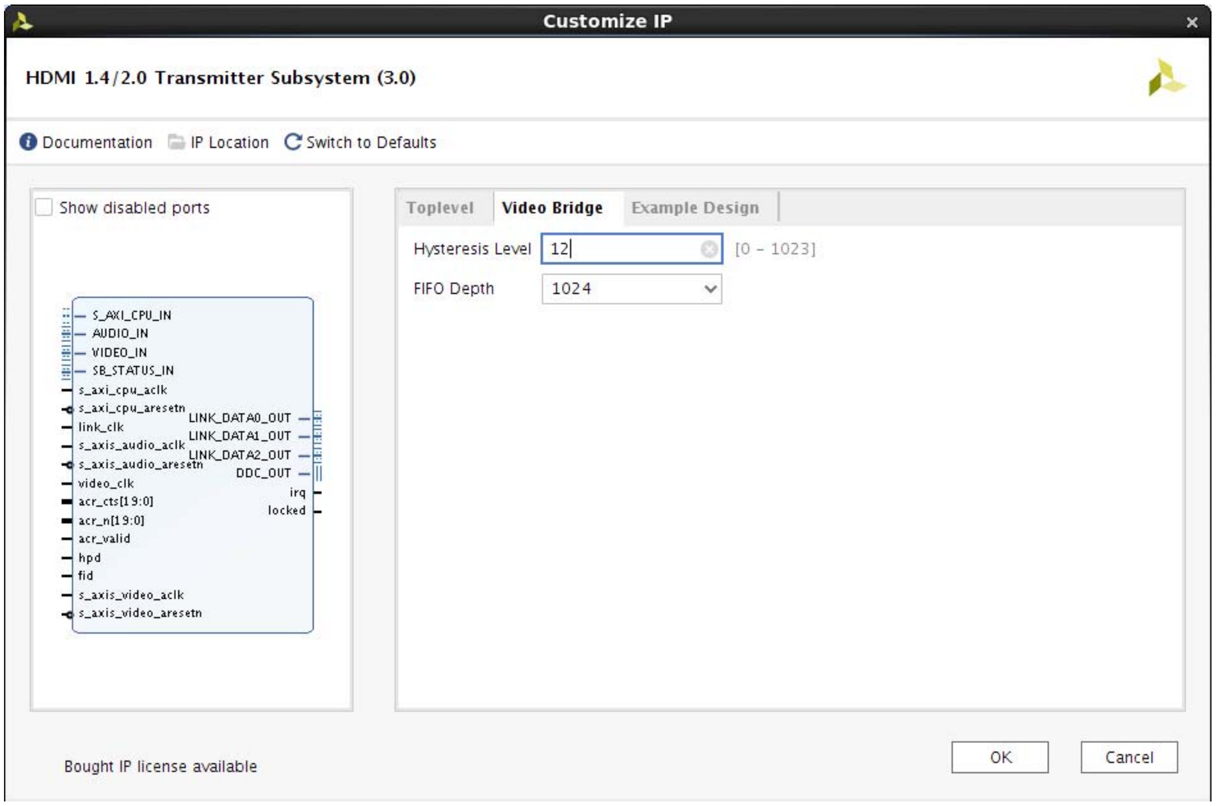
IMPORTANT: *For YUV420 Video, the line width is doubled, therefore, all the horizontal resolution fields must be divided by 2 to be supported in the HDMI TX Subsystem. For example, when PPC=2 is selected in the Vivado IDE, the total horizontal resolution must be divisible by 4 (instead of 2). In this case, 720p60 has a total horizontal pixel of 1650, which is not divisible by 4, so 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.

Note: HDCP 1.4 and 2.2 Encryption options are only configurable if you have a HDCP license, else it is disabled and greyed out from the GUI.

Video Bridge Tab

The Video Bridge tab is shown in Figure 4-2.



Hysteresis Level: 12 [0 – 1023]
FIFO Depth: 1024

Figure 4-2: **Video Bridge Tab**

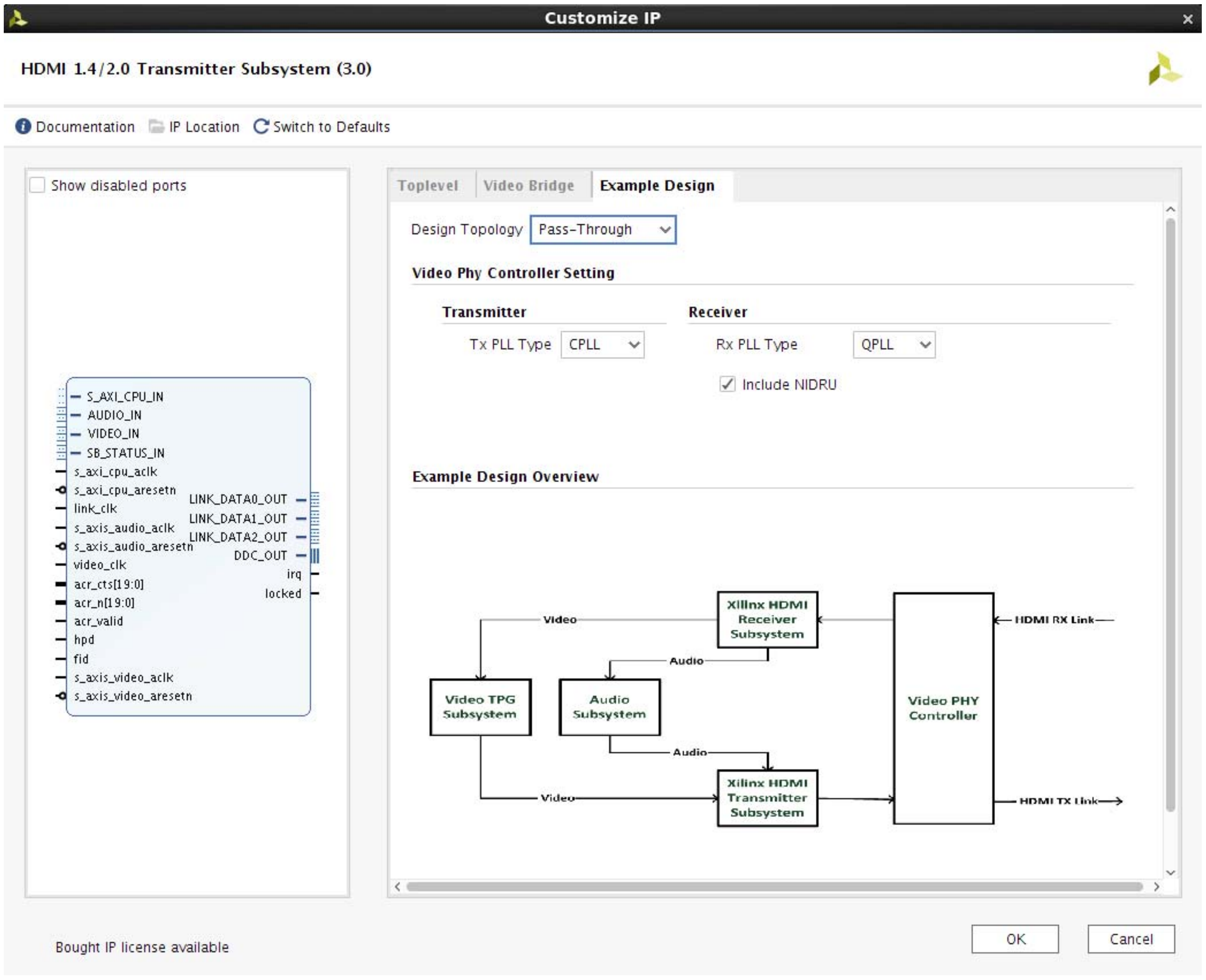
The parameters on the Video Bridge tab are as follows:

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 Figure 4-3.



HDMI 1.4/2.0 Transmitter Subsystem (3.0)

Design Topology: Pass-Through

Video Phy Controller Setting

Transmitter

Tx PLL Type: CPLL

Receiver

Rx PLL Type: QPLL

✓ Include NIDRU

Example Design Overview

Bought IP license available

Figure 4-3: **Example Design Tab**

Design Topology: Allows you to choose the topology of example design to be generated. The allowable options are Pass-Through and Tx Only.

where,

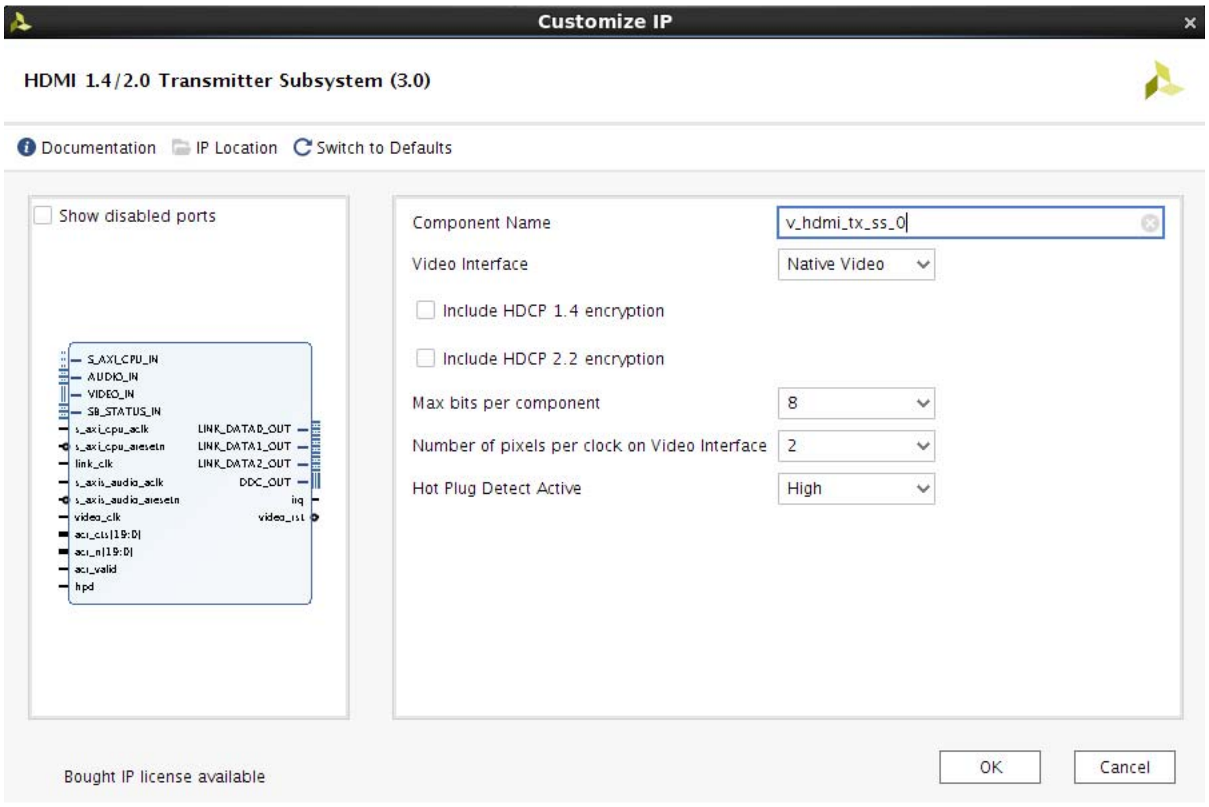
- Pass-Through showcases the HDMI system built with one HDMI TX Subsystem and one HDMI RX Subsystem, sharing the same Video PHY Controller.
- Tx Only showcases the HDMI system built with only one HDMI TX Subystem and Video PHY Controller. A Frame CRC helper core is added to the Tx Only topology to facilitate system monitor and debugging.

Video Phy Controller Setting Section: 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 utilization if the video resolution they plan to support doesn't require NI-DRU. Refer to *Video PHY Controller Product Guide* [Ref 24] for details about NI-DRU requirements.

Example Design Overview Section: A system block digram to show the overview of the example design to be generated.

Native Video Interface Option

The native video interface option window is shown in Figure 4-4.


Customize IP

HDMI 1.4/2.0 Transmitter Subsystem (3.0)

Documentation | IP Location | Switch to Defaults

☐ Show disabled ports

Component Name: v_hdmi_tx_ss_0

Video Interface: Native Video

☐ Include HDCP 1.4 encryption

☐ Include HDCP 2.2 encryption

Max bits per component: 8

Number of pixels per clock on Video Interface: 2

Hot Plug Detect Active: High

Bought IP license available

OK | Cancel

Figure 4-4: **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

Table 4-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Toplevel		
Video Interface	C_VID_INTERFACE	AXI4-Stream
AXI4-Stream	0	
Native Video	1	
Include HDCP 1.4 Encryption	C_INCLUDE_HDCP_1_4	Exclude
Exclude (Untick)	FALSE	
Include (Tick)	TRUE	
Include HDCP 2.2 Encryption	C_INCLUDE_HDCP_2_2	Exclude
Exclude (Untick)	FALSE	
Include (Tick)	TRUE	
Video over AXIS compliant NTSC/PAL Support	C_INCLUDE_LOW_RESO_VID	Exclude
Exclude (Untick)	FALSE	
Include (Tick)	TRUE	
Video over AXIS compliant YUV420 Support	C_INCLUDE_YUV420_SUP	Exclude
Exclude (Untick)	FALSE	
Include (Tick)	TRUE	
Max bits per component	C_MAX_BITS_PER_COMPONENT	8
8	8	
10	10	
12	12	
16	16	
Number of pixels per clock on Video Interface	C_INPUT_PIXELS_PER_CLOCK	2
2	2	
4	4	
Hot Plug Detect Active	C_HPD_INVERT	High
High	High	
Low	Low	
Video Bridge		
Hysteresis Level	C_HYSTERESIS_LEVEL	12

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
FIFO Depth	C_ADDR_WIDTH	1024
32	32	
1024	1024	
2048	2048	
4096	4096	
8192	8192	
Example Design		
Design Topology	C_EXDES_TOPOLOGY	0
Pass-Through	0	
Tx Only	1	
TX PLL Type	C_EXDES_TX_PLL_SELECTION	0 (GTXE2) 6 (GTHE3/4)
CPLL	0	

QPLL(GTXE2)
QPLL01(GTHE3/4)
	3
	6	

RX PLL Type	C_EXDES_RX_PLL_SELECTION	3 (GTXE2) 0 (GTHE3/4)
CPLL	0	

QPLL(GTXE2)
QPLL01(GTHE3/4)
	3
	6	

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

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 16].

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 PHY Controller constraints instead of these generated clocks. See *Clocking* in the *Video PHY Controller LogiCORE™ IP Product Guide* (PG230) [Ref 24] 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) [Ref 24].

Table 4-2 shows the device and speed grade selections for HDMI 1.4/2.0 Transmitter Subsystem.

Table 4-2: **Device and Speed Grade Selections**

Device Family	PPC	2				4			
	BPC	8	10	12	16	8	10	12	16
	Speed Grade								
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[2]				HDMI 1.4[1]			
	–2	HDMI 2.0[1]				HDMI 2.0[2]			
Kintex UltraScale	–1	HDMI 2.0[2]				HDMI 2.0[2]			
	–2								

Table 4-2: **Device and Speed Grade Selections *(Cont'd)***

Device Family	PPC	2				4			
	BPC	8	10	12	16	8	10	12	16
	Speed Grade								
Virtex-7	–1	HDMI 1.4[2]				HDMI 2.0[2]	HDMI 1.4[1]		
	–2	HDMI 2.0[1]				HDMI 2.0[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

The AXI4-Lite CPU clock must run at 100 Mhz. See Clocking in Chapter 3 for more information.

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) [Ref 16].

Example Design

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

Summary

HDMI 1.4/2.0 Transmitter Subsystem allows users to customize the example design based on their system requirements. It offers the full flexibility with the following system parameters:

Topology:

- TX-Only
- Pass-through

VPHY Configuration:

- NI-DRU Enable/Disable
- TXPLL Selection
- RXPLL Selection

Board:

- KC705
- KCU105
- ZC706
- ZCU102

Processor:

- MicroBlaze
- A9
- A53
- R5

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

- 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.

The inrevium TB-FMCH-HDMI4K FMC daughter card is required for example designs targeting the following boards:

- Kintex®-7 FPGA Evaluation Kit (KC705)
- Kintex® UltraScale™ FPGA Evaluation Kit (KCU105)
- Zynq®-7000 All Programmable SoC evaluation board (ZC706)

And it is not required while targeting the following boards, instead the on-board HDMI 2.0 circuitry is used.

- ZCU102 Evaluation Board

Hardware

The example design is built around the HDMI 1.4/2.0 Transmitter Subsystem (HDMI_TX_SS), HDMI 1.4/2.0 Receiver Subsystem (HDMI_RX_SS) (Optional), Video PHY (VPHY) Controller core and leverages existing Xilinx IP cores to form the complete system. Figure 5-1 and Figure 5-2 are illustrations of Overall HDMI Example Design block diagram while targeting various Xilinx Evaluation Kits.

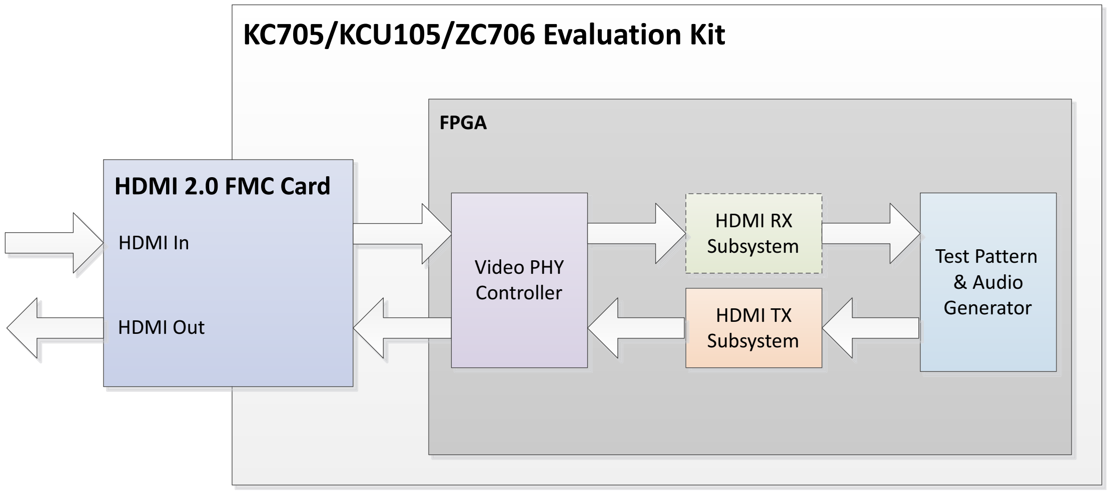


Figure 5-1: **KC705/KCU105/ZC706 HDMI Example Design Block Diagram**

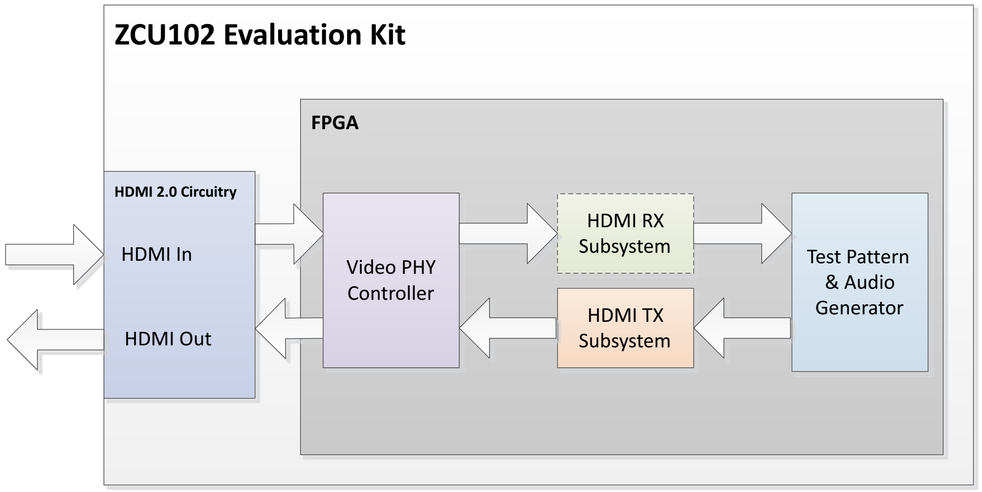


Figure 5-2: **ZCU102 HDMI Example Design Block Diagram**

The VPHY 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.

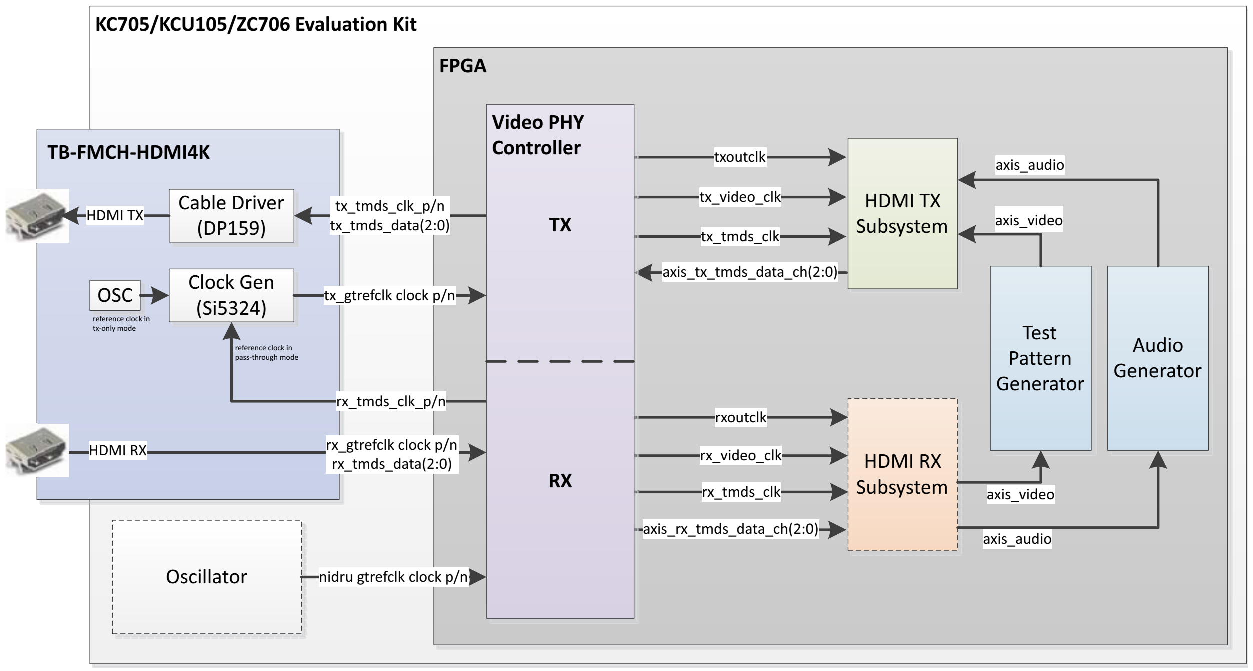


Figure 5-3: **C705/KCU105/ZC706 HDMI Reference Design Clock and Datapath Diagram**

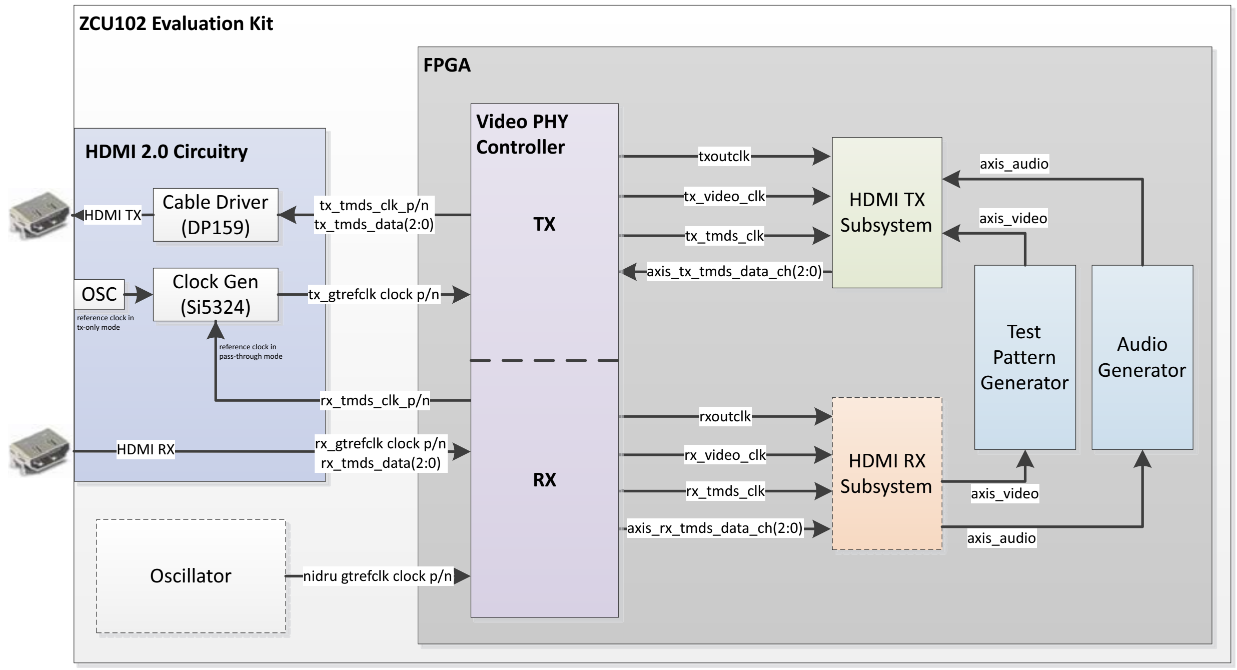


Figure 5-4: **ZCU102 HDMI Reference Design Clock and Datapath Diagram**

In pass-through mode, the VPHY Controller core recovers 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 VPHY 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 via the SI53xx clock generator in the HDMI 2.0 FMC card or on-board HDMI 2.0 circuitry.

In TX only mode, the colorbar pattern is generated by the TPG in form of AXI video stream and the low frequency audio is generated by the customized audio processing block in form of AXI audio stream. The two streams are forwarded to the HDMI_TX_SS for HDMI stream conversion then to the VPHY 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 Figure 5-5 and Figure 5-6 for a block diagrams of the three types of processor subsystems supported by HDMI Example Design flow.

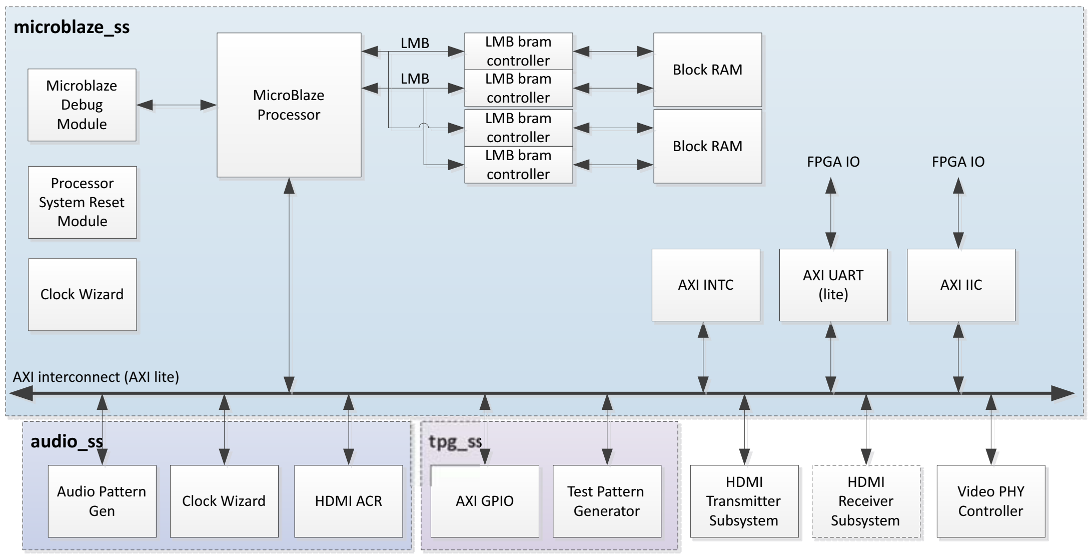


Figure 5-5: **HDMI Reference Design Block Diagram (MicroBlaze)**

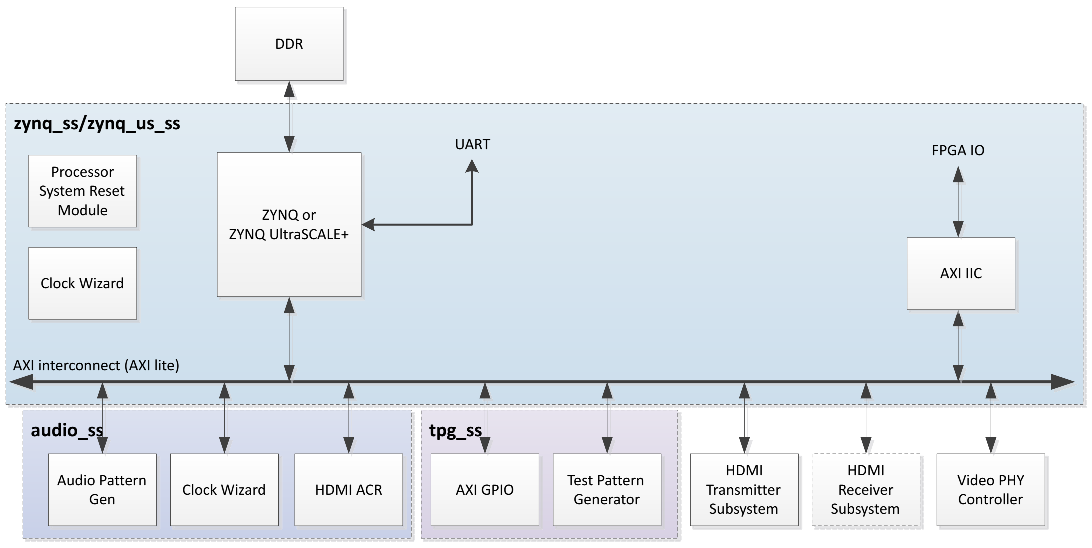


Figure 5-6: **HDMI Reference Design Block Diagram (Zynq or Zynq UltraScale+)**

Example Design Specifics

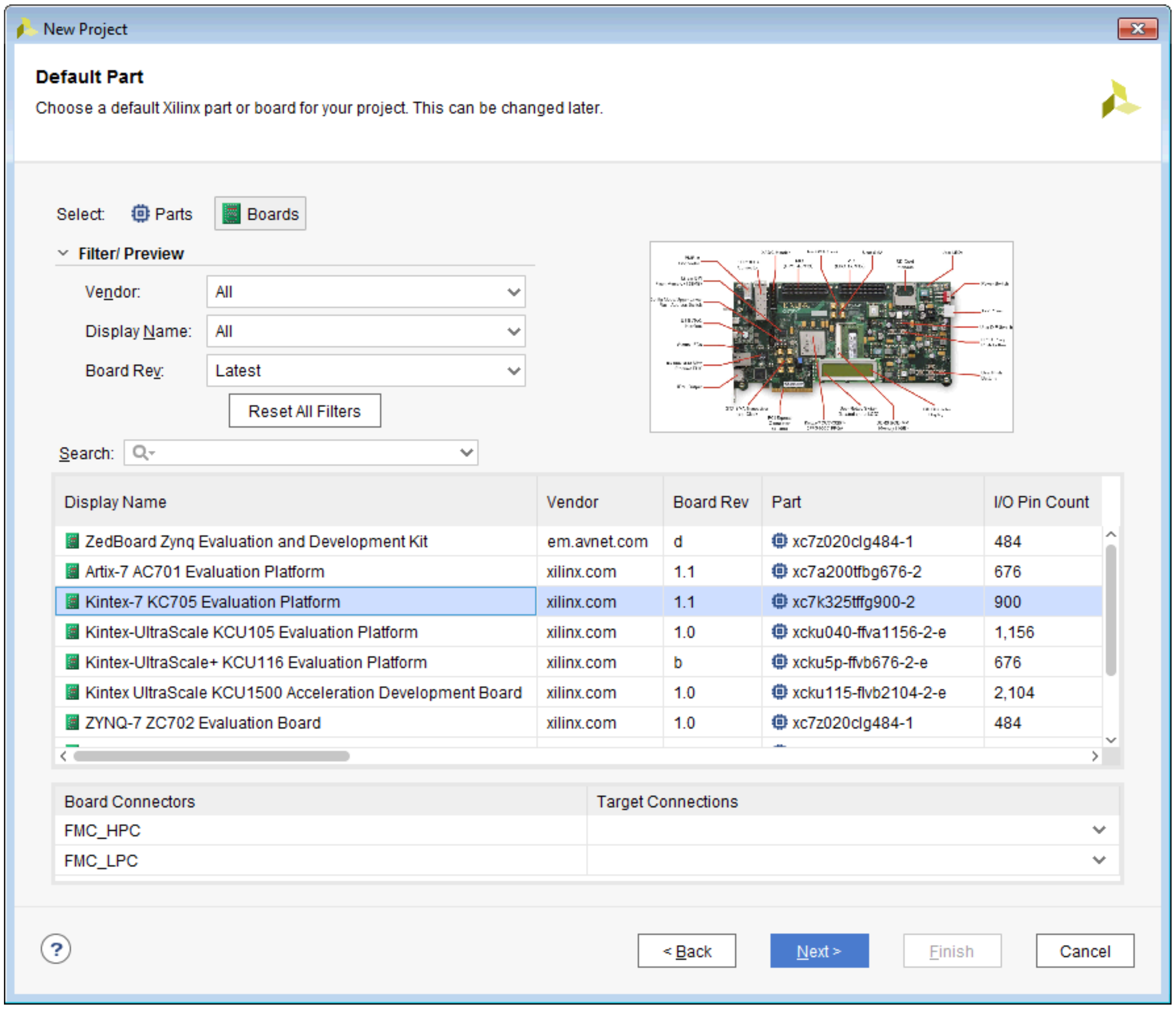
In addition to the Video PHY Controller, HDMI Transmitter Subsystem and HDMI Receiver Subsystem core, the complete example design includes the following cores:

- MicroBlaze or Zynq or Zynq UltraScale+
- MicroBlaze Debug Module (Only for MicroBlaze based processor subsystem)
- AXI Interconnect
- Local Memory Bus (Only for MicroBlaze based processor subsystem)
- LMB BRAM Controller (Only for MicroBlaze based processor subsystem)
- Block Memory Generator (Only for MicroBlaze based processor subsystem)
- Clocking Wizard
- Processor System Reset
- AXI UARTLite (Only for MicroBlaze based processor subsystem)
- AXI Interrupt Controller (Only for MicroBlaze based processor subsystem)
- AXI IIC
- 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)

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 Xilinx® part or board for the project.



New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts | Boards

Filter/ Preview

Vendor:	All
Display Name:	All
Board Rev:	Latest

Reset All Filters

Search:

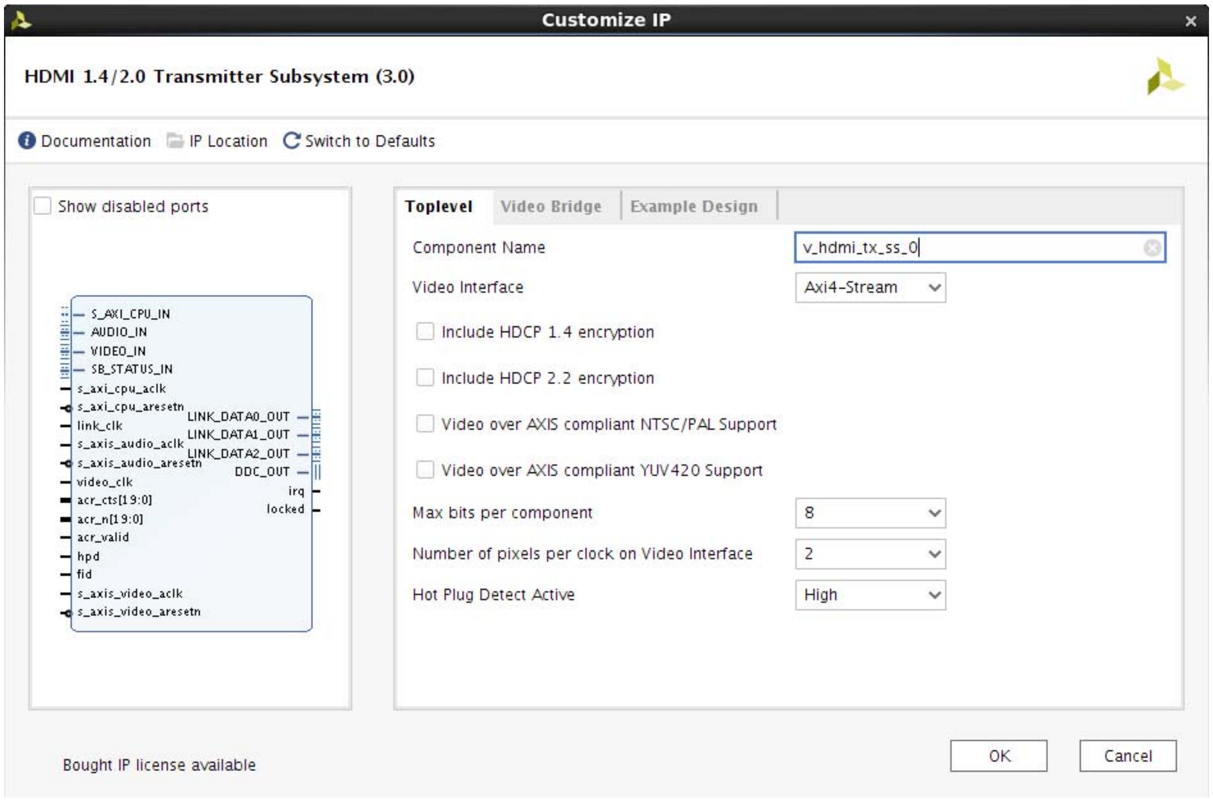
Display Name	Vendor	Board Rev	Part	I/O Pin Count
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484
Artix-7 AC701 Evaluation Platform	xilinx.com	1.1	xc7a200tfbg676-2	676
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900
Kintex-UltraScale KCU105 Evaluation Platform	xilinx.com	1.0	xcku040-ffva1156-2-e	1,156
Kintex-UltraScale+ KCU116 Evaluation Platform	xilinx.com	b	xcku5p-ffvb676-2-e	676
Kintex UltraScale KCU1500 Acceleration Development Board	xilinx.com	1.0	xcku115-flvb2104-2-e	2,104
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	1.0	xc7z020clg484-1	484

Board Connectors	Target Connections
FMC_HPC	
FMC_LPC	

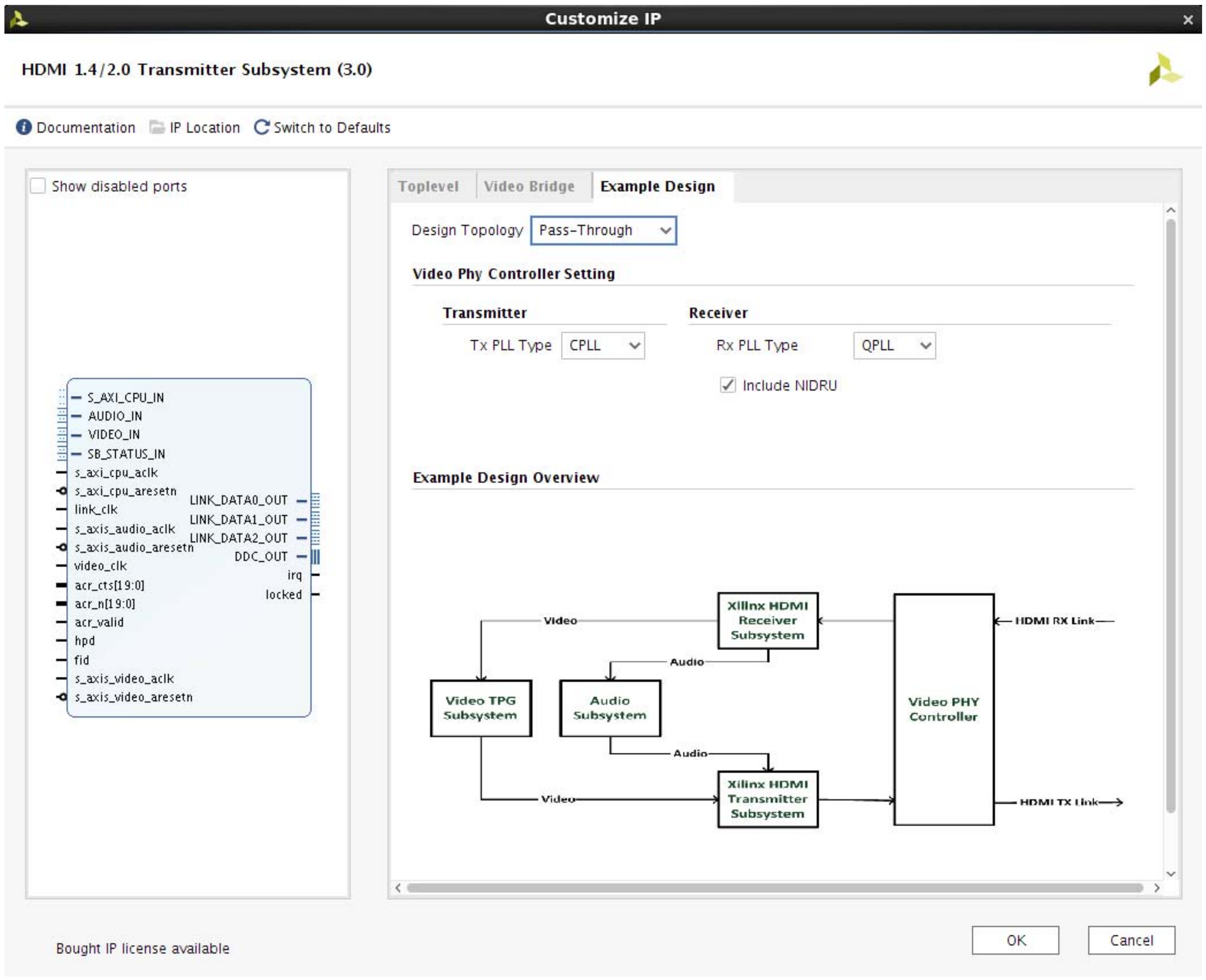
3. Select the Board. (KC705, KCU105, ZC706, and ZCU102 are supported.)
4. Click **Finish**.
5. Click **IP Catalog** and select HDMI 1.4/2.0 Transmitter Subsystem under Video Connectivity, then double click on it.
    - For the Example Design flow, Native Video Interface is not supported.

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

6. Configure HDMI 1.4/2.0 Transmitter Subsystem, then click **OK**.



Component Name: v_hdmi_tx_ss_0
Video Interface: Axi4-Stream
Include HDCP 1.4 encryption
Include HDCP 2.2 encryption
Video over AXIS compliant NTSC/PAL Support
Video over AXIS compliant YUV420 Support
Max bits per component: 8
Number of pixels per clock on Video Interface: 2
Hot Plug Detect Active: High
Bought IP license available



Customize IP

HDMI 1.4/2.0 Transmitter Subsystem (3.0)

Documentation | IP Location | Switch to Defaults

☐ Show disabled ports

Toplevel | Video Bridge | Example Design

Design Topology: Pass-Through

Video Phy Controller Setting

Transmitter

Tx PLL Type: CPLL

Receiver

Rx PLL Type: QPLL

☑ Include NIDRU

Example Design Overview

Bought IP license available

OK | Cancel

The **Generate Output Products** dialog box appears.

7. Click on **Generate**.

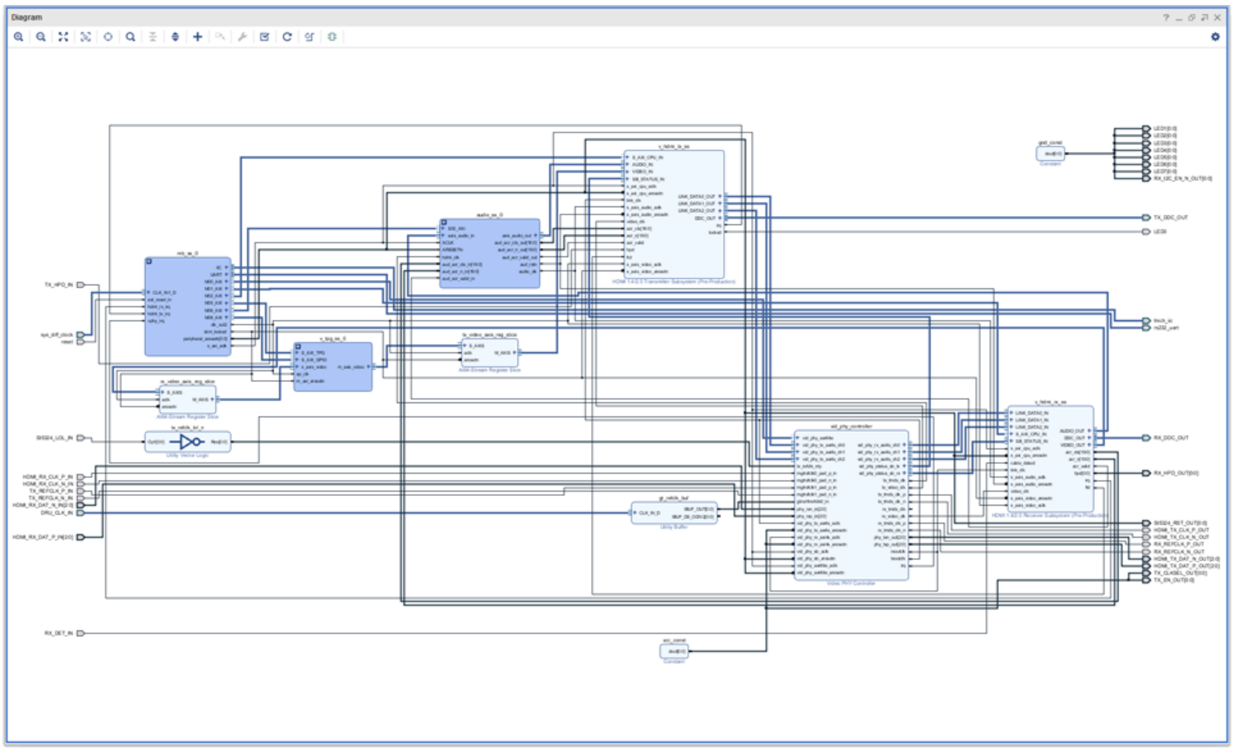
   a. You may optionally click **Skip** if you just want to generate the example design.

8. Right click on the HDMI 1.4/2.0 Transmitter Subsystem component under Design source, and click **Open IP Example Design**.

9. Choose the target project location, then click **OK**.

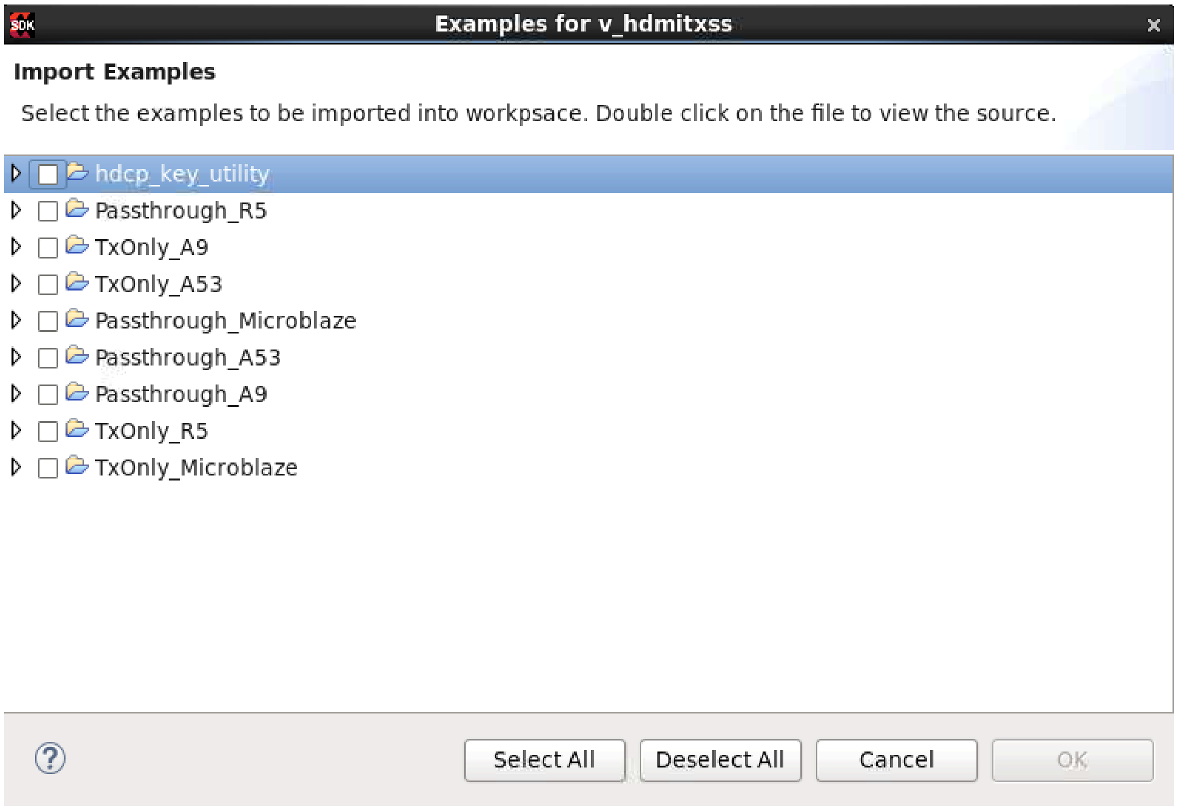
   The IPI Design is then generated. You may choose to Run Synthesis, Implementation, or Generate Bitstream.

   An overall system IPI block diagram of the KC705 based example design is shown below.



10. Export Hardware to prepare for SDK Example Design Flow (**File->Export->Export Hardware**).
11. Click **OK**. (Use the default Export Location <Local to Project> for the example design.)
12. Launch SDK (**File->Launch SDK**).
13. Choose SDK workspace location. By default, it is "Local to Project."

    Vivado SDK is launched.
14. Create Board Support Package in Vivado SDK.
15. Enter BSP project name and click **Finish**.
16. Click **OK**.
17. From system.mss, find the HDMI 1.4/2.0 Transmitter Subsystem and click on **Import Examples**.
18. Select xhdmi_example.



Import Examples

Select the examples to be imported into workpsace. Double click on the file to view the source.

- hdcp_key_utility
- Passthrough_R5
- TxOnly_A9
- TxOnly_A53
- Passthrough_Microblaze
- Passthrough_A53
- Passthrough_A9
- TxOnly_R5
- TxOnly_Microblaze

Refer to Table 5-1 to select the respective application.

Table 5-1: **Applications for Example Design**

Board	Processor	Topology	Import Example Selection
KC705/KCU105	MicroBlaze	Pass-through	Passthrough_MicroBlaze
KC705/KCU105	MicroBlaze	Tx Only	TxOnly_MicroBlaze
ZC706	A9	Pass-through	Passthrough_A9
ZC706	A9	Tx Only	TxOnly_A9
ZCU102	A53	Pass-through	Passthrough_A53
ZCU102	A53	Tx Only	TxOnly_A53
ZCU102	R5	Pass-through	Passthrough_R5
ZCU102	R5	Tx Only	TxOnly_R5

The example application is built successfully. The .elf is ready to use.

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 SDK and choose hdcp_key_utility.
2. Open hdcp_key_utility.c from the SDK 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, follow the steps below:

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 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 (KC705)

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 2017.3 > Vivado 2017.3 Tcl Shell**.
2. In the Xilinx command shell window, change to the Example Design Project directory:

   ```
   Vivado% cd ./v_hdmi_tx_ss_0_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 ./v_hdmi_tx_ss_0_ex.runs/impl_1/
   exdes_wrapper.bit
   ```

6. Set the target processor.

```
xsdb% target -set 3
```

7. Download the software .elf to the FPGA.

```
xsdb% dow ./v_hdmi_tx_ss_0_ex.sdk/<name of bsp>_xhdmi_example_1/
Debug/<name of bsp>_xhdmi_example_1.elf
```

8. Run the software.

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

9. Exit the XSDB command prompt.

```
xsdb% exit
```

Note: TB-FMCH-HDMI4K is not needed for ZCU102 example designs.

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 bitstream generated from Example Design Flow. KCU105 Board FMCH VADJ Adjustment shows the steps on how to set the VADJ power rail when using KCU105 board. For more details about KCU105 Board, to KCU105 Board User Guide [Ref 20].*

KCU105 Board FMCH VADJ Adjustment

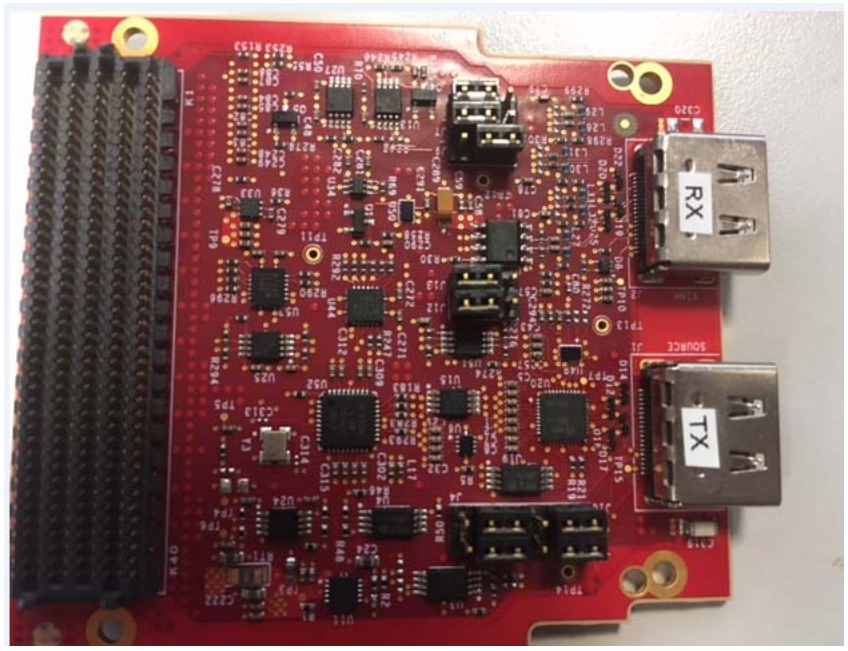
The KCU105 board system controller must apply power to the VADJ power rail for the HDMI 2.0 FMC card (TB-FMCH-HDMI4K). Most new boards are per-programmed and should be detected. The VADJ is powered when the DS19 LED (located near the power switch on the KCU105 board) is ON.

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 AP SoC system controller and which is assigned to the UART for the UltraScale FPGA. 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 FPGA 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.
   b.



IMPORTANT: *Ensure that the jumpers are set correctly on their HDMI 2.0 FMC daughter card.*

Migration Notes

When migrating from version 2016.3 or earlier, make note of 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.

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

Verification, Compliance, and Interoperability

Interoperability

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

Hardware Testing

The HDMI 1.4/2.0 Transmitter 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 All Programmable SoC evaluation board (ZC706)
- ZCU102 Evaluation Board

The HDMI 1.4/2.0 Transmitter 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 analyser / SIX-A
- DELL P2415Q
- Philips 288P6LJEB
- LG 27mu67

Video Resolutions

Figure A-1 shows the hardware setup for AXI4-Stream Video Interface. An HDMI source connects to 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 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 TX Subsystem alone instead of relying on the video received from the HDMI RX Subsystem.

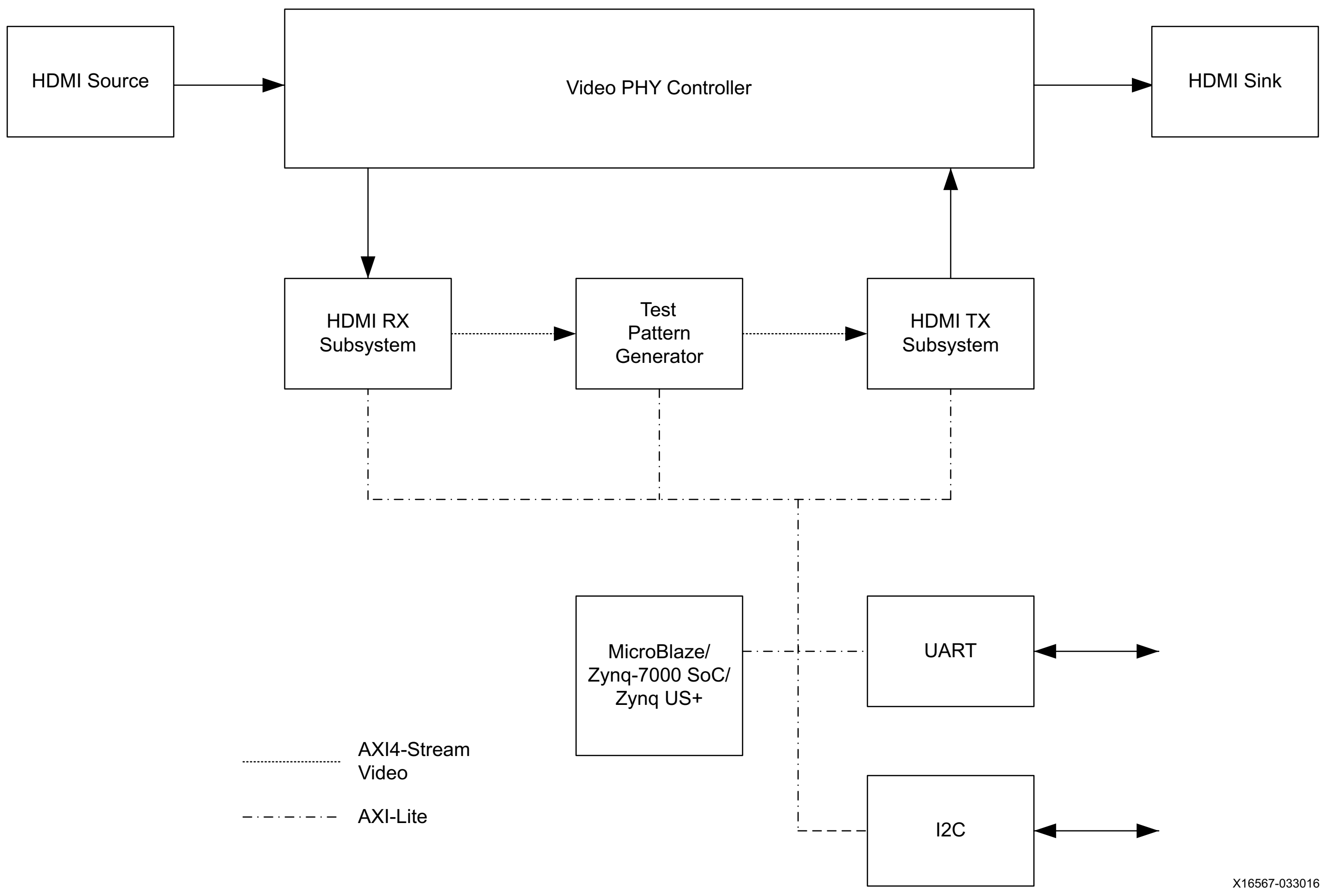


Figure A-1: **Test Setup for AXI4-Stream Video Interface**

For Video PHY Controller settings and PLL selections, see the *Video PHY Controller LogiCORE IP Product Guide* (PG230) [Ref 24].

Similarly, Figure A-2 shows the hardware setup for 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.

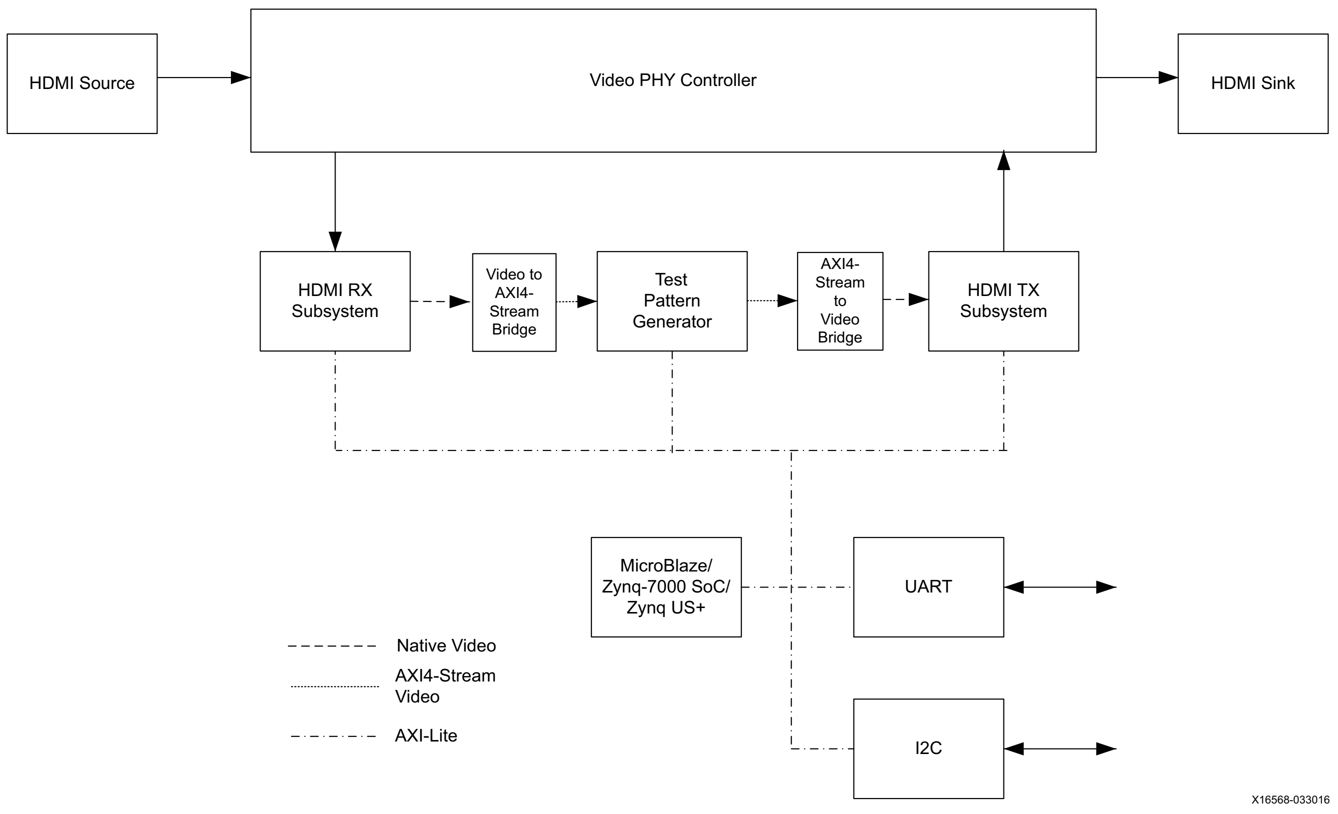


Figure A-2: **Test Setup for Native Video Interface**

Table A-1, Table A-2, and Table A-3 show the video resolutions that were tested as part of the release for different video formats. The following limitations apply:

- When **Number of pixels per clock on AXI4-S output** (PPC) set to 4, only video formats with total horizontal resolution divisible by 4 are supported. For example, a video resolution 720p60 has horizontal total pixel 1650, which is not divisible by 4. Therefore, it can't be supported in the design with PPC set to 4.

Table A-1: **Tested Video Resolutions for RGB 4:4:4 and YCbCr 4:4:4**

Resolution	Horizontal		Vertical		Frame Rate (Hz)
	Total	Active	Total	Active	
480i60	858	720	525	480	60
576i50	864	720	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

Table A-1: **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	
1080p24	2750	1920	1125	1080	24
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(2)	4400	3840	2250	2160	60
4096x2160p60(2)	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. Not all resolutions can be supported due to VPHY limitation. For details, refer to *Video PHY Controller LogiCORE IP Product Guide* (PG230) [Ref 24].
2. For 4kp60 YUV444/RGB video, only 8-bits is supported as defined by HDMI 2.0 spec due to bandwidth limitation.

Table A-2: **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 A-2: **Tested Video Resolutions for YCbCr 4:2:2 at 12 Bits/component**

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

Table A-3: **Tested Video Resolutions for YCbCr 4:2:0 at 8, 10, 12, 16 Bits/Component**

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

Debugging

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

TIP: *If the IP generation halts with an error, there might be a license issue. See License Checkers in Chapter 1 for more details.*

Finding Help on Xilinx.com

To help in the design and debug process when using the HDMI 1.4/2.0 Transmitter 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.

Documentation

This product guide is the main document associated with the HDMI 1.4/2.0 Transmitter 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 proper keywords such as

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

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

Master Answer Record for the HDMI 1.4/2.0 Transmitter Subsystem

AR: 65911

Technical Support

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

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

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

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

Vivado Design Suite Debug Feature

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

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

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

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 21].

Reference Boards

Various Xilinx development boards support the HDMI 1.4/2.0 Transmitter 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 All Programmable SoC evaluation board
  - ZC706
- UltraScale+ FPGA evaluation board
  - ZCU102

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)
  - LED0 - HDMI TX subsystem lock (when HDMI Example Design is used)
  - Use debug port to check if there are link data coming from Video PHY Controller core.
  - Refer to the *Debugging* Appendix in *Video PHY Controller LogiCORE IP Product Guide* (PG230) [Ref 24], and ensure there is no problem with clocking issues.
- The system flow/state information from the event logs by UART option **z**.

Table B-1: **System Flow and State Information**

At system start-up	While changing video stream from UART Menu
VPHY log	VPHY log
------	------
GT init start	TX frequency event
GT init done	QPLL lost lock
TX frequency event	TX frequency event
TX timer event	TX timer event
TX MMCM reconfig done	TX MMCM reconfig done
QPLL reconfig done	QPLL reconfig done
GT TX reconfig start	GT TX reconfig start
GT TX reconfig done	GT TX reconfig done
TX MMCM lock	TX MMCM lock
QPLL lock	QPLL lock
TX reset done	TX reset done
TX alignment done	TX alignment done
HDMI TX log	HDMI TX log
------	------
Initializing HDMI TX core....	TX Set Stream, with TMDS (128)
Initializing VTC core....	TX Stream is Down
Reset HDMI TX Subsystem....	TX Audio Unmuted
TX cable is connected....	TX Stream is Up
TX Stream is Down	
TX Set Stream, with TMDS (32)	
TX Audio Unmuted	
TX Stream is Up	

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:

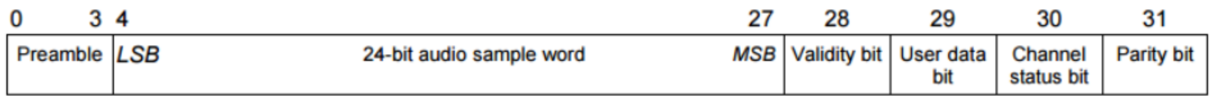
- If received <interface_name>_tready is stuck low, the subsystem cannot send data. Check if there is an issue at the AXI4 Stream Slave.
- Check that the `aclk` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed.
- Check subsystem configuration.

AXI4-Stream Audio Interface

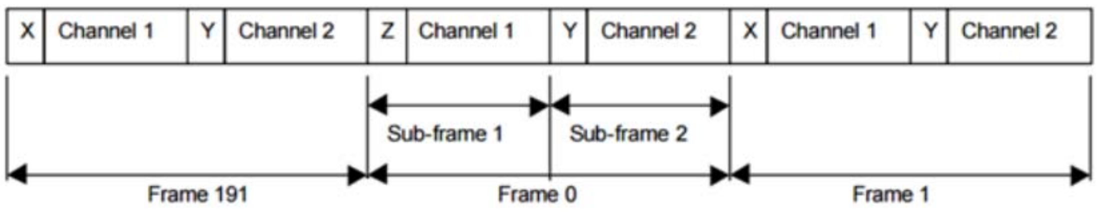
To ensure that the audio is working in HDMI 1.4/2.0 Transmitter Subsystem, the AXI4-Stream must be constructed as described below.

The HDMI 1.4/2.0 Transmitter Subsystem supports up to 8 audio channels. The audio data is transmitted through AXI4-Stream audio interface, which is a customized AXI4-Stream protocol that is used to send audio samples with sideband signals defined in AES3 specification.

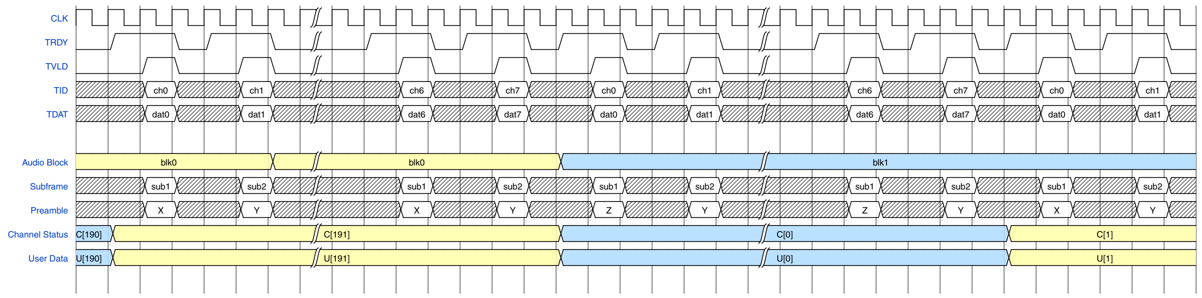
The sub-frame format for audio sample is shown as below.



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 form one “Audio Block”. And the first sub-frame in each “Audio Block” starts with a preamble “Z”. An illustration is shown below.



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.



If 8 channel audio is enabled in your design, only 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 per illustrated above.

Application Software Development

Device Drivers

The HDMI 1.4/2.0 Transmitter 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.

Figure C-1 shows the HDMI 1.4/2.0 Transmitter Subsystem architecture.

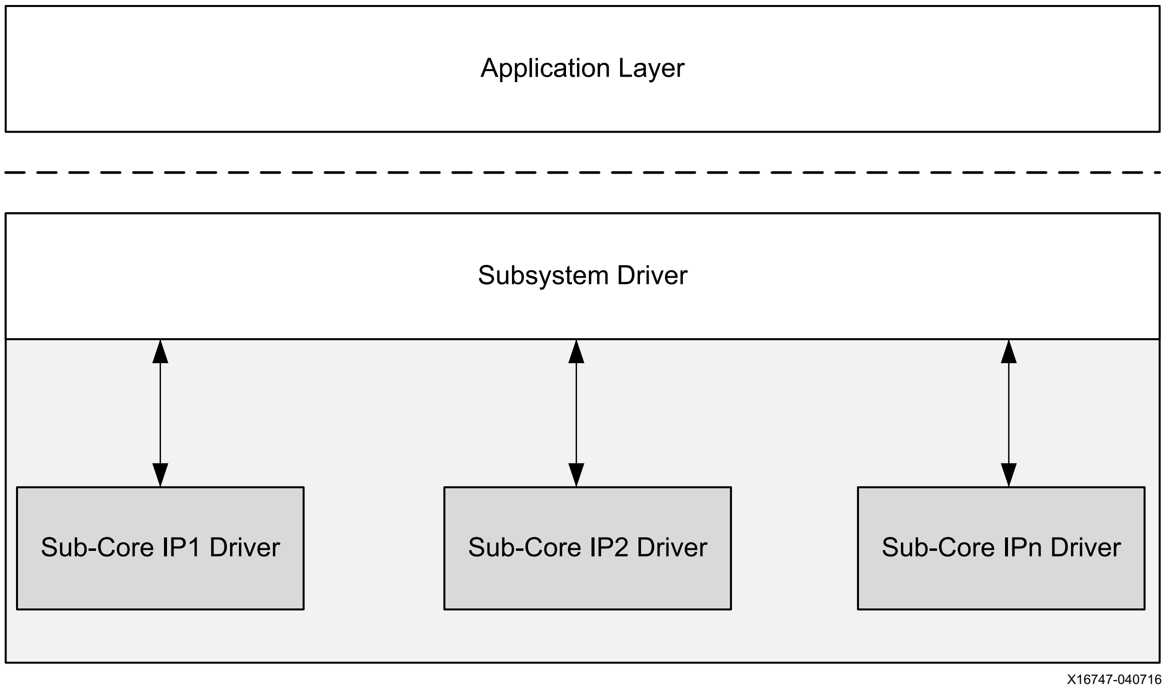


Figure C-1: **Subsystem Driver Architecture**

The HDMI 1.4/2.0 Transmitter 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 Transmitter Subsystem is tightly coupled with the Xilinx Video PHY Controller, which itself is independent and offer flexible architecture with multiple-protocol support. Both MAC and PHY are dynamically programmable through the AXI4-Lite interface.

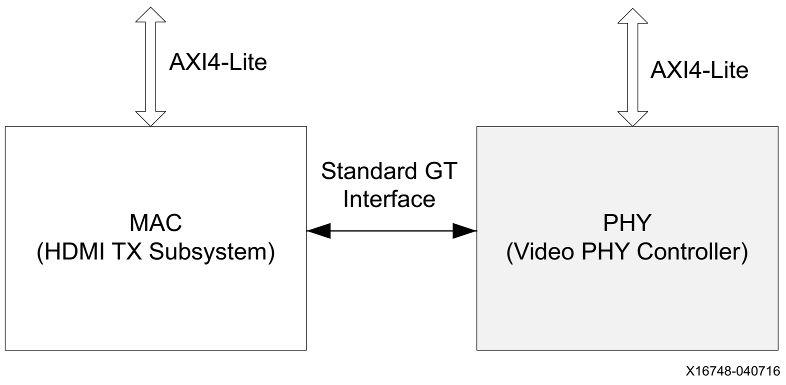


Figure C-2: **MAC Interfaces with PHY**

Usage

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

Ensure that the video stream has started. Then, valid AUX data and audio data can be inserted after the video is locked. However, because the application knows what video format will be sent and what audio format will be embedded. With this information, the ACR number can be calculated and set before 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.

HDMI TX Subsystem Flow

HDMI TX Subsystem in general responses to the following two events:

- Hot-plug signal (HPD) from the sink device
- Software user application to indicate a change of video stream (e.g. change of video resolution)

The main program flow is shown in Figure C-3. At execution, the software application initializes the HDMI TX Subsystem IP and registers the callback functions in the provided hooks.

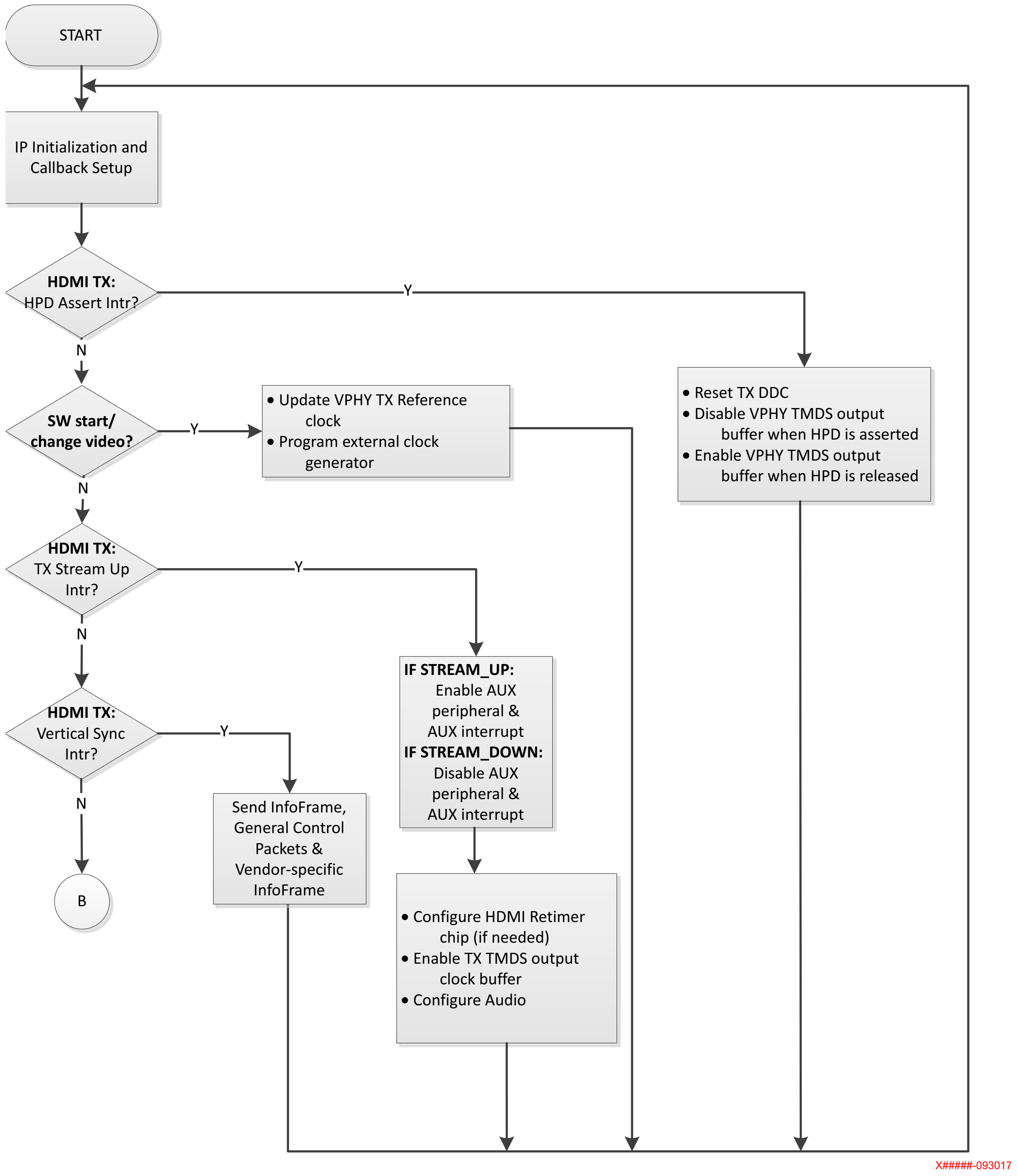
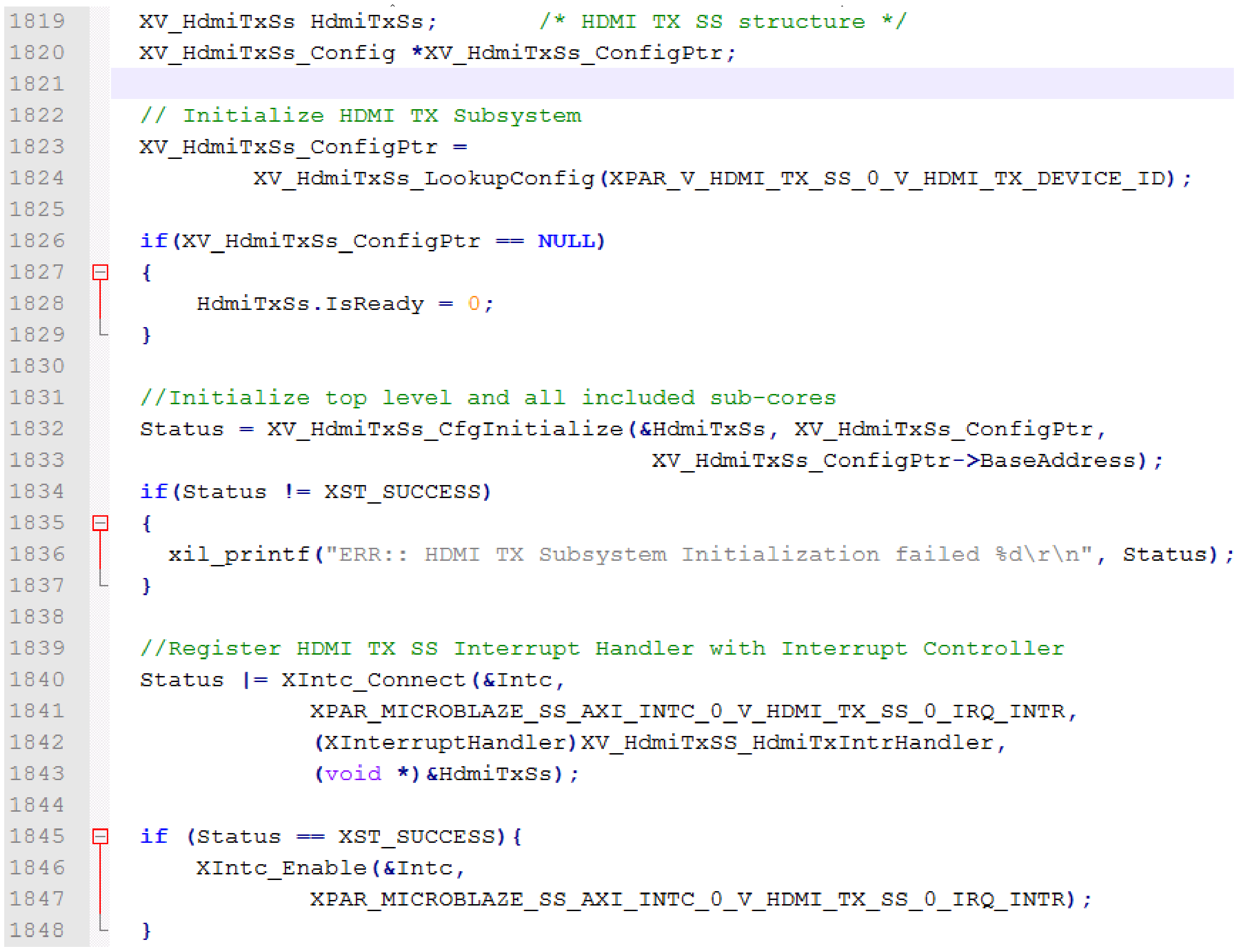

START
IP Initialization and Callback Setup
HDMI TX: HPD Assert Intr?
Y
N
SW start/ change video?
Y
• Update VPHY TX Reference clock
• Program external clock generator
N
HDMI TX: TX Stream Up Intr?
Y
N
HDMI TX: Vertical Sync Intr?
Y
N
B
• Reset TX DDC
• Disable VPHY TMDS output buffer when HPD is asserted
• Enable VPHY TMDS output buffer when HPD is released
IF STREAM_UP: Enable AUX peripheral & AUX interrupt
IF STREAM_DOWN: Disable AUX peripheral & AUX interrupt
Send InfoFrame, General Control Packets & Vendor-specific InfoFrame
• Configure HDMI Retimer chip (if needed)
• Enable TX TMDS output clock buffer
• Configure Audio
X#####-093017

Figure C-3: **TX Flow**

Application Integration

Figure C-4 shows an example code on how an HDMI 1.4/2.0 Transmitter Subsystem can be used in your application.



```
XV_HdmiTxSs HdmiTxSs;        /* HDMI TX SS structure */
XV_HdmiTxSs_Config *XV_HdmiTxSs_ConfigPtr;

// Initialize HDMI TX Subsystem
XV_HdmiTxSs_ConfigPtr =
        XV_HdmiTxSs_LookupConfig(XPAR_V_HDMI_TX_SS_0_V_HDMI_TX_DEVICE_ID);

if(XV_HdmiTxSs_ConfigPtr == NULL)
{
    HdmiTxSs.IsReady = 0;
}

//Initialize top level and all included sub-cores
Status = XV_HdmiTxSs_CfgInitialize(&HdmiTxSs, XV_HdmiTxSs_ConfigPtr,
                                   XV_HdmiTxSs_ConfigPtr->BaseAddress);
if(Status != XST_SUCCESS)
{
  xil_printf("ERR:: HDMI TX Subsystem Initialization failed %d\r\n", Status);
}

//Register HDMI TX SS Interrupt Handler with Interrupt Controller
Status |= XIntc_Connect(&Intc,
            XPAR_MICROBLAZE_SS_AXI_INTC_0_V_HDMI_TX_SS_0_IRQ_INTR,
            (XInterruptHandler)XV_HdmiTxSS_HdmiTxIntrHandler,
            (void *)&HdmiTxSs);

if (Status == XST_SUCCESS){
    XIntc_Enable(&Intc,
            XPAR_MICROBLAZE_SS_AXI_INTC_0_V_HDMI_TX_SS_0_IRQ_INTR);
}
```

Figure C-4: **Application Example Code**

To integrate and use the HDMI 1.4/2.0 Transmitter 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 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 steps 2 and 3 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 Transmitter Subsystem driver is responsible for combining HDCP 1.4 and HDCP 2.2 drivers 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)
2. Initialize the HDMI 1.4/2.0 Transmitter Subsystem driver after the HDCP keys have been loaded. Initializing the subsystem begins 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`

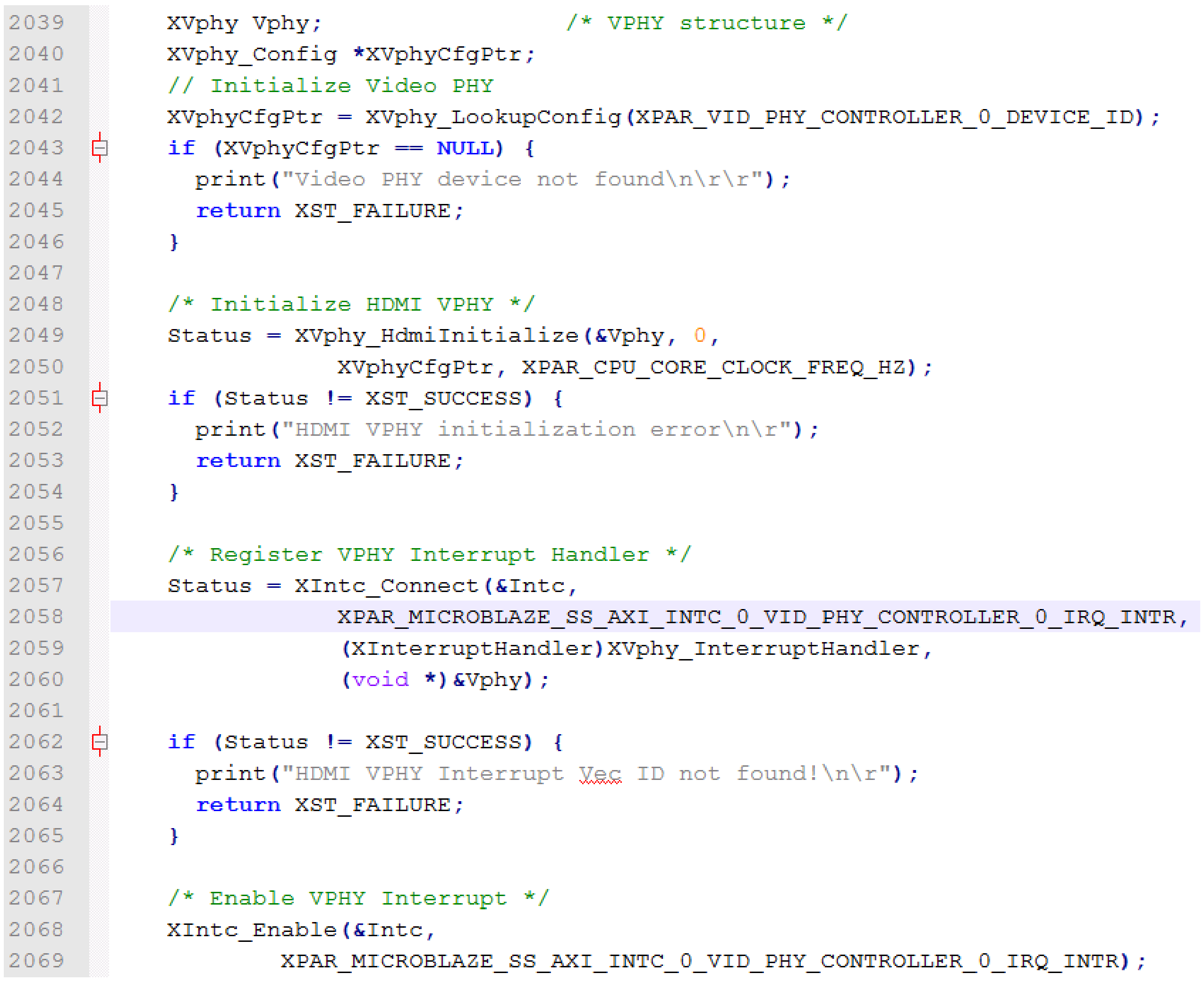
4. Set the HDCP user callback functions. These callback functions are used to hook into HDCP state machine and allow the user to take action at various stages of the protocol. If there is no use for the callback at the application level, then the callback can be left undefined.
   - `XV_HdmiTxSs_SetCallback`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

Because the HDMI 1.4/2.0 Transmitter 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.



```
XVphy Vphy;                 /* VPHY structure */
XVphy_Config *XVphyCfgPtr;
// Initialize Video PHY
XVphyCfgPtr = XVphy_LookupConfig(XPAR_VID_PHY_CONTROLLER_0_DEVICE_ID);
if (XVphyCfgPtr == NULL) {
  print("Video PHY device not found\n\r\r");
  return XST_FAILURE;
}

/* Initialize HDMI VPHY */
Status = XVphy_HdmiInitialize(&Vphy, 0,
            XVphyCfgPtr, XPAR_CPU_CORE_CLOCK_FREQ_HZ);
if (Status != XST_SUCCESS) {
  print("HDMI VPHY initialization error\n\r");
  return XST_FAILURE;
}

/* Register VPHY Interrupt Handler */
Status = XIntc_Connect(&Intc,
            XPAR_MICROBLAZE_SS_AXI_INTC_0_VID_PHY_CONTROLLER_0_IRQ_INTR,
            (XInterruptHandler)XVphy_InterruptHandler,
            (void *)&Vphy);

if (Status != XST_SUCCESS) {
  print("HDMI VPHY Interrupt Vec ID not found!\n\r");
  return XST_FAILURE;
}

/* Enable VPHY Interrupt */
XIntc_Enable(&Intc,
        XPAR_MICROBLAZE_SS_AXI_INTC_0_VID_PHY_CONTROLLER_0_IRQ_INTR);
```

Figure C-5: **Application Example Code**

To integrate and use the Video PHY Controller for HDMI 1.4/2.0 Transmitter 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 device ID.

   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 Video PHY controller after the HDMI 1.4/2.0 Transmitter Subsystem initialization is completed.
- Registering the Video 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 Transmitter Subsystem are listed here:

1. **HPD** – Peripheral I/O to detect HDMI cable 5.0V signal
   a. **Rising edge** – Cable connected
   b. **Falling edge** – Cable disconnected
2. **Link Ready** – Every time when Video 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. HDCP 1.4 Interrupt (only available when HDCP 1.4 is enabled in hardware)

5. HDCP 1.4 Timer Interrupt (only available when HDCP 1.4 is enabled in hardware)
6. HDCP 2.2 Timer Interrupt (only available when HDCP 2.2 is enabled in hardware)

Table C-1: **Mapping between Interrupt Sources and Application Callback Functions**

Interrupts	Callback
HPD	XV_HDMITXSS_HANDLER_CONNECT
Link Ready ***Note:*** It 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
HDCP 1.4 Interrupt	
HDCP 1.4 Timer Interrupt	
HDCP 2.2 Timer Interrupt	
	XV_HDMITXSS_HANDLER_HDCP_AUTHENTICATE ***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

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_VS
- XV_HDMITXSS_HANDLER_STREAM_UP
- XV_HDMITXSS_HANDLER_STREAM_DOWN
- XV_HDMITXSS_HANDLER_HDCP_AUTHENTICATE

XV_HDMITXSS_HANDLER_CONNECT

This interrupt is triggered every time when an HDMI TX cable connection or disconnection (HPD level transition) occurs.

The callback function needs to 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_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_STREAM_UP

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

The callback function needs to perform the following:

1. If a HDMI Retimer or equalizer is used in the system, configure the Retimer with the correct setting based on the required line rate.
2. Enable TX TMDS Clock by calling Video PHY Controller API:

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

3. Set HDMI 1.4/2.0 Transmitter Subsystem Sampling Rate with the Video PHY Controller TX Sampling Rate.

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

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 HDMI 1.4/2.0 Transmitter Subsystem to stream video.

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

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

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 needs to 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 Transmitter 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 Transmitter 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 changes.

The callback function needs to initialize a reference clock change process for HDMI 1.4/2.0 Transmitter 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 Video PHY Controller TX alignment is done.

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

Follow the steps in Chapter 5, Example Design 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 Transmitter 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.

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 TDMS clock for HDMI 1.4/2.0 Transmitter Subsystem through 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 Transmitter Subsystem stream parameters through 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 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 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 Controller 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 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 8 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 8 channel audio in your design, only 6 out of 8 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 may 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 provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (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 the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

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

References

These documents provide supplemental material useful with this product guide:

1. *Xilinx Vivado 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 All Programmable SoC: DC and AC Switching Characteristics* (DS187)
8. *Zynq-7000 All Programmable SoC: DC and AC Switching Characteristics* (DS191)
9. *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922)
10. *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923)
11. *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925)
12. HDMI specifications (www.hdmi.org/manufacturer/specification.aspx)
13. HDCP specifications (www.digital-cp.com/hdcp-specifications)
14. *AXI4-Stream Video IP and System Design Guide* (UG934)
15. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
16. *Vivado Design Suite User Guide: Designing with IP* (UG896)
17. *Vivado Design Suite User Guide: Getting Started* (UG910)
18. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
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 Product Guide* (PG059)
24. *Video PHY Controller Product Guide* (PG230)
25. *HDCP v2.2 Product Guide* (PG249)
26. *HDCP v1.4 Product Guide* (PG224)
27. *AXI4-Stream to Video Out LogiCORE IP Product Guide* (PG044)
28. ANSI/CTA Standard (https://standards.cta.tech/kwspub/published_docs/ANSI-CTA-861-F-Preview.pdf)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/04/2017	3.0	• 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 (TX, 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	2.0	• Removed single pixel per clock support
11/30/2016	2.0	• Added example design migration notes.
10/05/2016	2.0	• Added example design flow. • Added HPD XGUI option. • Added software use cases. • Updated Xilinx **AUTOMOTIVE APPLICATIONS DISCLAIMER**.
06/08/2016	2.0	• Updated optional video over AXI-Stream support.
04/06/2016	2.0	• 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	1.0	Initial Xilinx release.

Please Read: Important Legal Notices