HDMI 1.4/2.0 Receiver Subsystem v3.1

Product Guide

Vivado Design Suite

PG236 (v3.1) December 11, 2020

Table of Contents

Introduction

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

Features

- HDMI 2.0 and 1.4b compatible
- 2 or 4 symbol/pixel per clock input
- Supports resolutions up to 4,096 x 2,160 @ 60 fps
- 8, 10, 12, and 16-bit deep color support
- Supports both integer and non-integer frame rates
- Dynamic support for RGB, YUV 4:4:4, YUV 4:2:2, YUV 4:2:0 color formats
- Supports AXI4-Stream video output stream and native video output stream
- Audio support for up to 32 channels
- High bit rate (HBR) Audio
- 3D Audio support
- Optional HDCP 2.3/1.4 decryption support
- Info frames
- Data Display Channel (DDC)
- Hot-Plug Detection
- 3D video support
- Optional video over AXIS compliant NTSC/PAL support
- Optional video over AXIS compliant YUV420 support
- Optional HPD active polarity
- Optional cable detect active polarity

- Supports HDR video transport (Dynamic Range and Mastering info frames)
  - Traditional Gamma - SDR
  - Traditional Gamma - HDR
  - HDR 10 - SMPTE ST 2084
  - Hybrid Log Gamma (HLG)

IP Facts

Subsystem Facts Table	
Subsystem Specifics	
Supported Device Family[1]	UltraScale+™ Families (GTHE4, GTYE4) UltraScale™ Architecture (GTHE3) Zynq-7000 SoC 7 series (GTXE2)[2] Artix®-7 (GTPE2)[2] Versal™ ACAPs (GTYE5)
Supported User Interfaces	AXI4-Lite, AXI4-Stream
Resources	Performance and Resource Utilization web page
Provided with Subsystem	
Design Files	RTL
Example Design	Vivado® IP integrator and associated software application example
Test Bench	Not Provided
Constraints File	XDC
Simulation Model	Not Provided
Supported S/W Driver[3]	Standalone, Linux
Tested Design Flows[4]	
Design Entry	Vivado® Design Suite
Simulation	Not Provided
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 54546
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog
2. Only HDMI 1.4 is supported in Artix-7 and Kintex-7 -1 devices.
3. Standalone driver details can be found in <Install Directory>/Vitis/<Release>/data/embeddedsw/doc/xilinx_drivers_api_toc.htm.
   Linux OS and driver support information is available from the Linux HDMI RX Driver Page.
4. For the supported versions of third-party tools, see the Xilinx Design Tools: Release Notes Guide.

Chapter 2

Overview

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
    - Port Descriptions
    - Clocks and Resets
    - Customizing and Generating the Subsystem
    - Chapter 6: Example Design

Subsystem Overview

The HDMI 1.4/2.0 Receiver Subsystem is a feature-rich soft IP incorporating all the necessary logic to properly interface with PHY layers and provide HDMI decoding functionality. The subsystem is a hierarchical IP that bundles a collection of HDMI RX-related IP sub-cores and outputs them as a single IP. The subsystem receives the captured TMDS data from the PHY layer. It then extracts the video and audio streams from the HDMI stream and converts it to video and audio streams.

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 Verification, Compliance, and Interoperability.

Related Information

Verification, Compliance, and Interoperability

Unsupported Features

The following features are not supported in this subsystem:

- Lip sync
- CEC
- HEAC
- Dual view
- Multi-stream audio

Licensing and Ordering Information

License Type

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

For more information, visit the Xilinx HDMI web page.

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

IMPORTANT! *Xilinx provides the HDCP IP License (including evaluation license) for HDCP adopters only. See the DPC Licensee list for details.*

License Checkers

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

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

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

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

Chapter 3

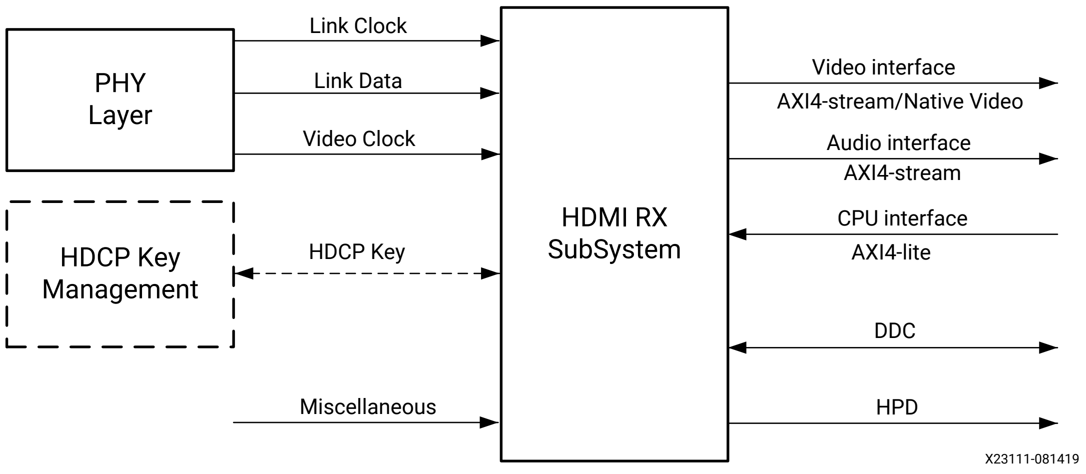
Product Specification

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

Introduction

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

Figure 1: **Subsystem Block Diagram**



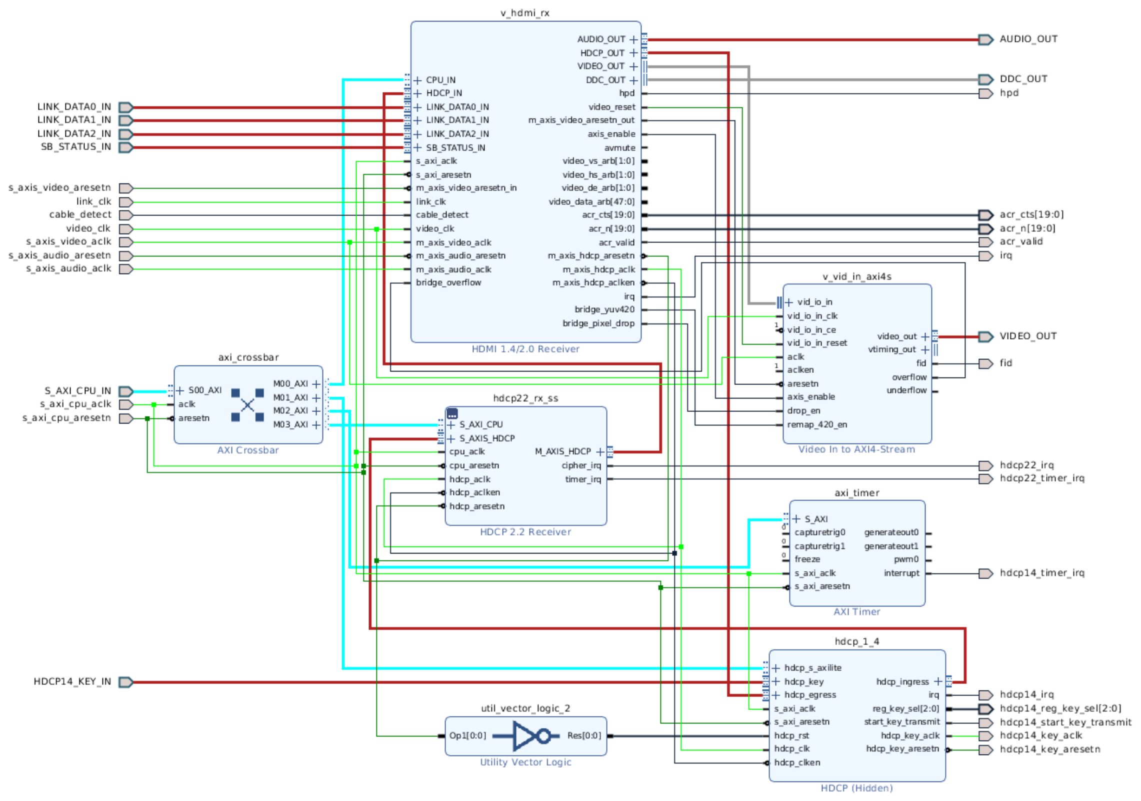
The HDMI 1.4/2.0 RX Subsystem is constructed on top of an HDMI RX core. Various supporting modules are added around the HDMI RX core with respect to your configuration. The HDMI RX core is designed to support native video interface, however many of the existing video processing IP cores are AXI4-Stream-based. It is a natural choice to add a converter module (Video In to AXI4-Stream) to enable the HDMI 1.4/2.0 RX Subsystem to output AXI4-Stream-based video. Doing this allows the HDMI 1.4/2.0 RX Subsystem to work seamlessly with other Xilinx video processing IP cores. The HDMI 1.4/2.0 RX Subsystem has a built-in capability to optionally support both HDCP 1.4 and HDCP 2.3 encryption. The HDMI 1.4/2.0 RX Subsystem has a built-in capability to optionally support both HDCP 1.4 and HDCP 2.3 decryption.

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

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

The following figure shows the internal structure of the HDMI 1.4/2.0 RX Subsystem when AXI4-Stream is selected as the video interface. In this illustration, both HDCP 1.4 and HDCP 2.3 are selected and both Video over AXIS compliant NTSC/PAL Support and Video over AXIS compliant YUV420 Support are selected.

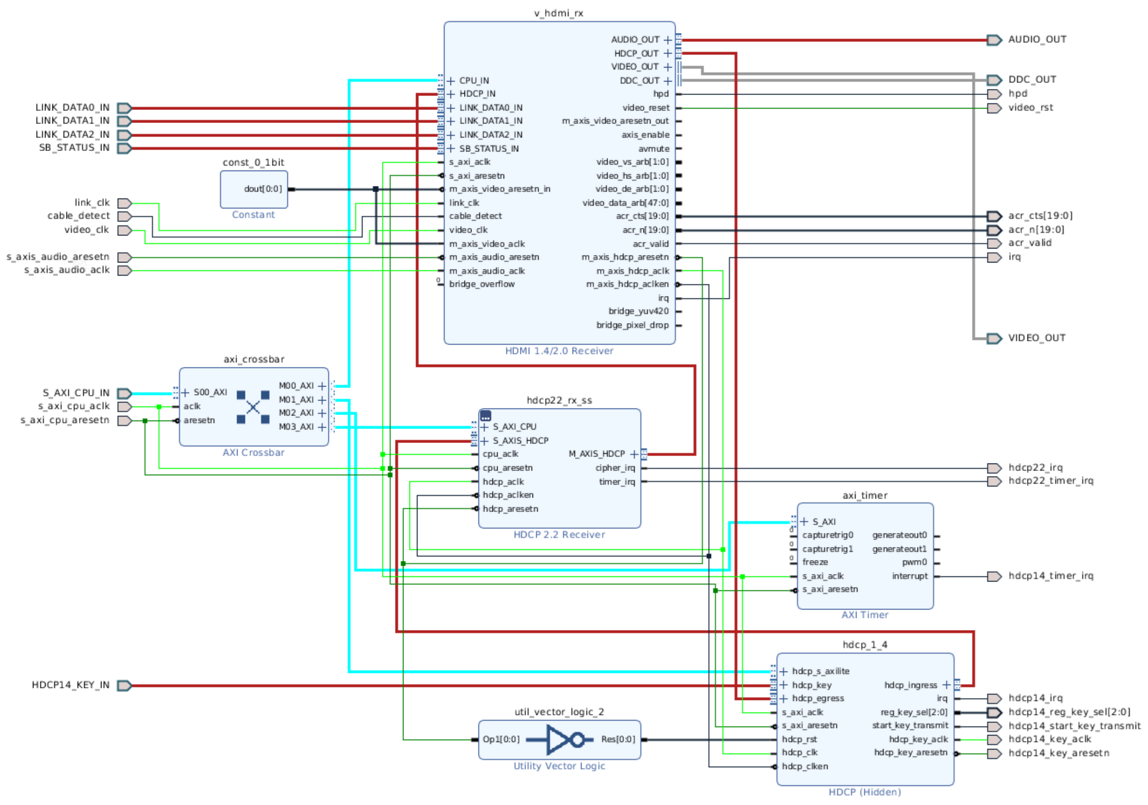
Figure 2: **HDMI RX Subsystem Internal Structure in AXI4-Stream Video Interface Mode**



The HDMI 1.4/2.0 RX Subsystem also provides an option to support a native video interface by constructing the subsystem without the Video In to AXI4-Stream bridging module. Some applications require support of customized resolutions, which are not divisible by the PPC setting (4 or 2). Therefore, the HDMI 1.4/2.0 RX Subsystem also provides a native video (Vectored DE) interface option to enable this application. When native video interface (with or without Vectored DE) is selected, the HDMI 1.4/2.0 RX Subsystem outputs native video to its own video devices. In native video mode, the HDMI 1.4/2.0 RX Subsystem still has a built-in capability to optionally support both HDCP 1.4 and HDCP 2.3 decryption.

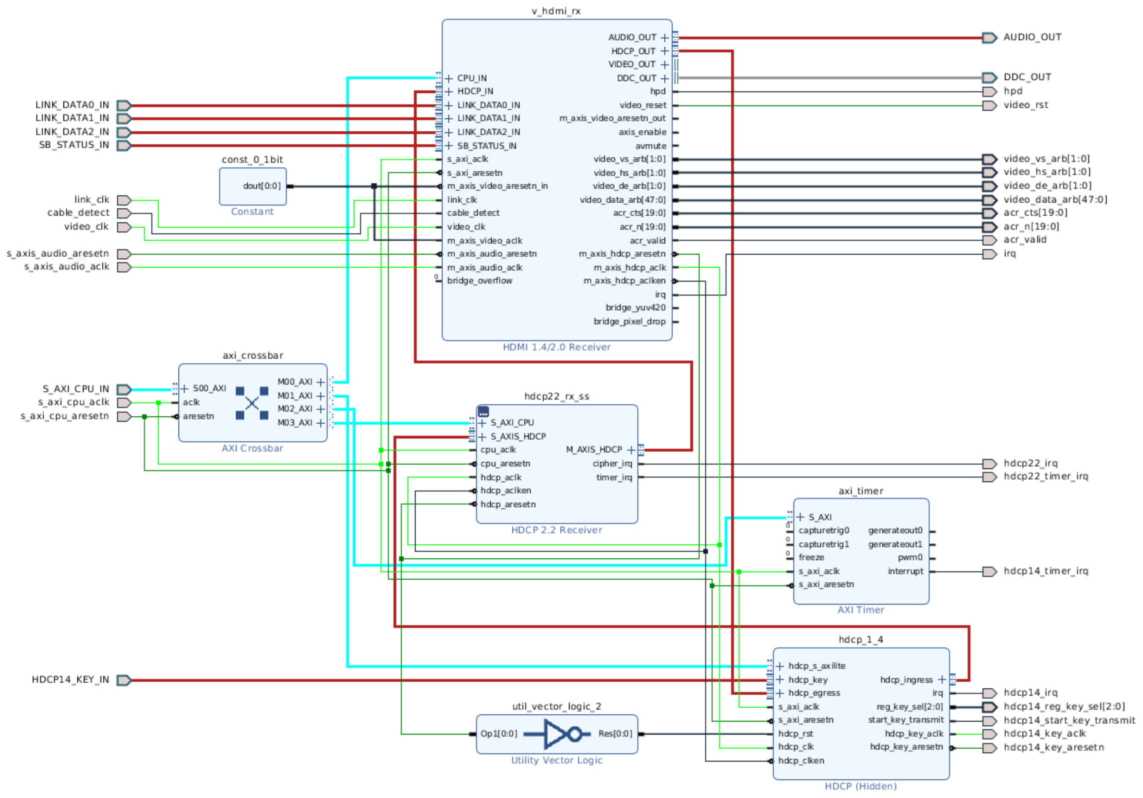
The following figure shows the internal structure of the HDMI 1.4/2.0 RX Subsystem when native video is selected as the video interface. In this illustration, both HDCP 1.4 and HDCP 2.3 are selected.

Figure 3: **HDMI RX Subsystem Internal Structure in Native Video Interface Mode**



The following figure shows the internal structure of the HDMI 1.4/2.0 RX Subsystem when Native Video (Vectored DE) interface is selected as the video interface. In this illustration, both HDCP 1.4 and HDCP 2.3 are selected.

Figure 4: **HDMI 1.4/2.0 RX Subsystem Internal Structure in Native Video (Vectored DE) Interface Mode**



The data width of the video interface is configured in the Vivado IDE by setting the **Number of Pixels Per Clock on Video Interface** and the **Max Bits Per Component** parameters.

The audio interface is a 32-bit AXI4-Stream master bus. The subsystem converts the captured audio to a multiple channel AXI audio stream and outputs the audio data on this interface.

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

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

The HDMI 1.4/2.0 RX 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 HDMI 1.4/2.0 RX Subsystem is connected to a Xilinx Video PHY Controller/HDMI GT Subsystem, which takes electronic signals from an HDMI cable and translates it into HDMI stream. Then, the HDMI 1.4/2.0 RX Subsystem converts the HDMI stream into video stream and audio stream. Based on the configuration selected, the HDMI 1.4/2.0 RX Subsystem sends the video stream in either Native Video format or AXI4-Stream format together with the AXI4-Stream Audio to other processing modules.

***Note*:** The HDMI GT Subsystem comprises of the HDMI GT Controller IP and the Versal ACAPs Transceivers Wizard IP. For more information *HDMI GT Controller LogiCORE IP Product Guide* (PG334).

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

Audio Clock Regeneration Signals

The subsystem can output Audio Clock Regeneration (ACR) signals that allow receiver audio peripherals to regenerate the audio clock.

The audio clock regeneration architecture is not part of the HDMI 1.4/2.0 RX Subsystem. You must provide an audio clock to the application. This can be achieved by using an internal PLL or external clock source, depending on the audio clock requirements, audio sample frequency and jitter. When the HDMI 1.4/2.0 RX Subsystem is used in DVI mode, the ACR outputs can be left unconnected.

Related Information

Example Design

Display Data Channel (DDC)

The subsystem allows the end-user to build an HDMI sink device, which negotiates with the targeted HDMI source 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.

***Note*:** The DDC signals can be left disconnected when using the HDMI core in DVI only mode without HDCP.

Status and Control Data Channel (SCDC)

The subsystem supports the following two bits in the SCDC register address offset 0x20 for TMDS configurations (Table 10-19 of the HDMI 2.0 specification).

- Bit 1: TMDS_Bit_Clock_Ratio
- Bit 0: Scrambling_Enable

Two APIs are available in the driver to use this feature.

- `XV_HdmiRx_DdcScdcEnable` is used to enable the SCDC register
- `XV_HdmiRx_DdcScdcClear` is used to clear the SCDC register

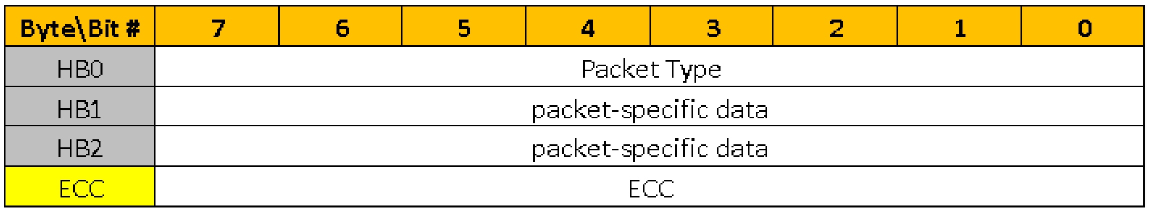
Hot Plug Detect

The subsystem supports the Hot Plug Detect (HPD) feature, which is a communication mechanism between the HDMI source and HDMI sink devices. For example, when an HDMI cable is inserted between the HDMI source and HDMI sink devices, the cable-detect signal is asserted. The subsystem then outputs a `hpd` signal, which triggers the start of a communication between the source device and sink device.

AUX Packets

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

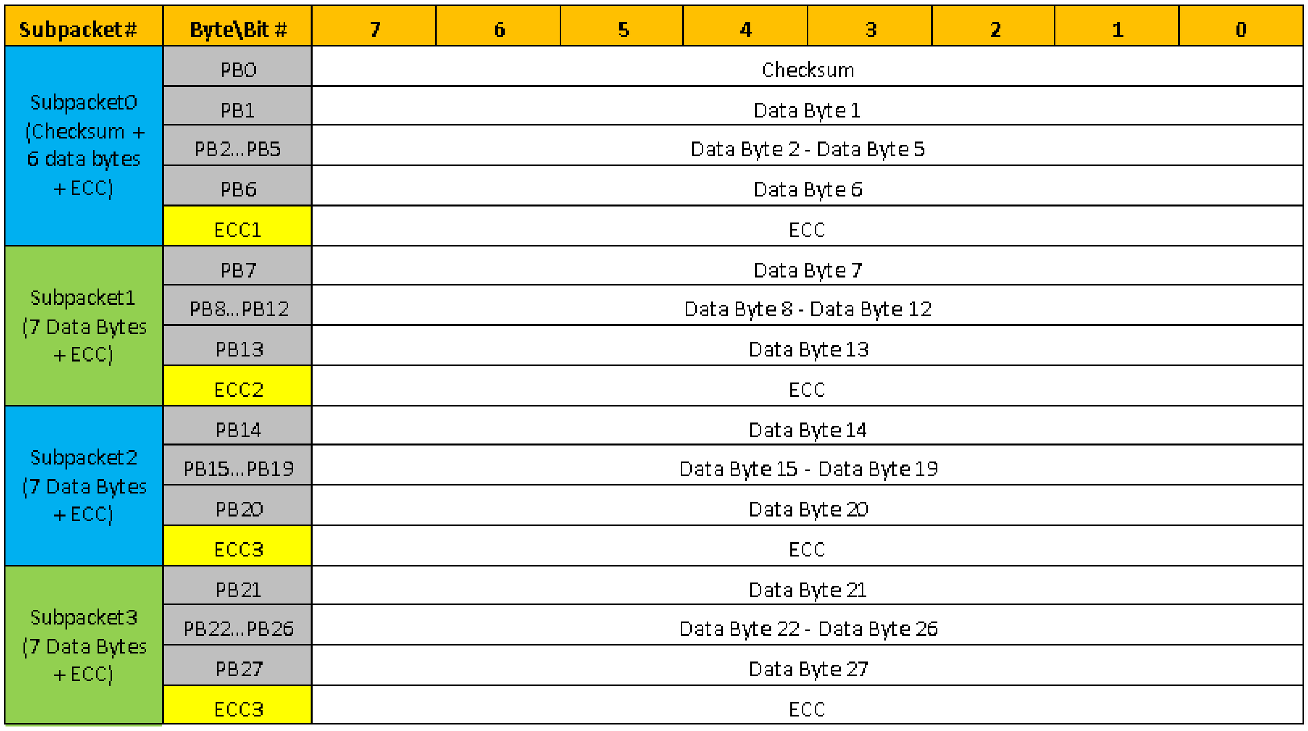
Figure 5: **Packet Header**



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

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

Figure 6: **Packet Body**



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

***Note*:**

- ECC is calculated in the HDMI 1.4/2.0 RX Subsystem. Therefore, you must construct HB0...HB2, and PB0, PB1...PB26, PB27 according to HDMI specs in the software.
- When calculating the checksum value (PB0), the ECC values are ignored.
- Refer to section 5.2.3.4 and 5.2.3.5 of the HDMI 1.4 specification for more information on the Aux packet structure.

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

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

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

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

Packet Type Value	Packet Type
0x08	DST Audio Packet
0x09	High Bitrate (HBR) Audio Stream Packet (IEC 61937)
0x0A	Gamut Metadata Packet
0x0B	3D Audio Sample Packet (LPCM Format Only)
0x0C	One Bit 3D Audio Sample Packet
0x0D	Audio MetaData Packet
0x80+InfoFrame Type	InfoFrame Packet
0x00	Reserved
0x01	Vendor-Specific
0x02	Auxiliary Video Information (AVI)
0x03	Source Product Descriptor
0x04	Audio
0x05	MPEG Source
0x06	NTSC VBI
0x07	Dynamic Range and Mastering (HDR)

Null packets are dropped by the HDMI 1.4/2.0 RX Subsystem automatically. ACR, Audio Sample, One-bit Audio, DST Audio, and HBR Audio are routed to the audio interface. The remaining packets (packet types not highlighted in the previous table) are routed to the software for further processing by generating AUX packet interrupts.

In the HDMI 1.4/2.0 RX Subsystem driver, there is a generic API function to allow you to retrieve aux packets.

```
XHdmiC_Aux *XV_HdmiRxSs_GetAuxiliary(XV_HdmiRxSs *InstancePtr);
```

where,

- InstancePtr is a pointer to the HDMI 1.4/2.0 RX Subsystem instance.
- It returns a pointer to a data structure contains the complete AUX packet.

The aux packet data structure defined in the driver is:

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

The HDMI Common driver also defines API functions to parse important InfoFrame packets and general control packets so that this information is available to be used in the user application software.

The list of available parser APIs are:

```
void XV_HdmiC_ParseAVIInfoFrame(XHdmiC_Aux *AuxPtr, XHdmiC_AVI_InfoFrame
*infoFramePtr);
void XV_HdmiC_ParseAudioInfoFrame(XHdmiC_Aux *AuxPtr, XHdmiC_AudioInfoFrame
*AudIFPtr);
int XV_HdmiC_VSIF_ParsePacket(XHdmiC_Aux *AuxPtr, XHdmiC_VSIF *VSIFPtr);
void XV_HdmiC_ParseGCP(XHdmiC_Aux *AuxPtr, XHdmiC_GeneralControlPacket
*GcpPtr);
void XV_HdmiC_ParseDRMIF(XHdmiC_Aux *AuxPtr, XHdmiC_DRMInfoFrame
*DRMInfoFrame);
```

The parsed information is stored in data structures with respect to the packet type.

Other packet types are not parsed by the driver. Instead, upon the AUX interrupt, you must retrieve those packets and process them in the application software. For example, if an HDMI source sends NTSC VBI Infoframe, upon receiving them by HDMI Subsystem, the NTSC VBI packet (both header and payload) is stored in the hardware FIFO, an AUX interrupt is generated. Then the application software reads out the AUX from the FIFO. By checking the packet header, the application knows that the packet is NTSC VBI. Then the application software must parse the payload data and handle it according to their system requirements.

InfoFrames

As shown in the previous table, the software handles the Vendor Specific InfoFrame (VSIF), the Auxiliary Video Information (AVI) InfoFrame, the Audio InfoFrame, and the Dynamic Range and Mastering InfoFrame. The four InfoFrame types are added as part of the HDMI 1.4/2.0 RX Subsystem data structure, each having its own well-defined structure. For more details on InfoFrames, refer to Section 6 of CTA-861-G.

AVI InfoFrame

Use the following code example for AVI InfoFrames:

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

Audio InfoFrame

Use the following code example for Audio InfoFrames:

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

Vendor Specific InfoFrame (VSIF)

Use the following code example for VSIF:

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

Dynamic Range and Mastering InfoFrame

Use the following code example for Dynamic Range and Mastering (HDR) InfoFrames:

```
typedef struct XHdmiC_DRM_InfoFrame {
    XHdmiC_DRM_EOTF EOTF;
    XHdmiC_DRM_Static_Metadata_Descp_Id Static_Metadata_Descriptor_ID;
    struct {
        u16 x,y;
    } disp_primaries[3];
    struct {
        u16 x,y;
    } white_point;
    u16 Max_Disp_Mastering_Luminance;
    u16 Min_Disp_Mastering_Luminance;
    u16 Max_Content_Light_Level;
    u16 Max_Frame_Average_Light_Level;
} XHdmiC_DRMInfoFrame;
```

General Control Packet

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

```
typedef struct XHdmiC_GCP_Packet {
  unsigned char Set_AVMUTE;
  unsigned char Clear_AVMUTE;
  XHdmiC_ColorDepth ColorDepth;
  XHdmiC_PixelPackingPhase PixelPackingPhase;
  unsigned char Default_Phase;
} XHdmiC_GeneralControlPacket;
```

HDCP

As part of the HDMI 1.4/2.0 RX Subsystem, the Xilinx LogiCORE IP High-bandwidth Digital Content Protection (HDCP) receivers are designed to receive audiovisual content securely between two devices that are HDCP capable. In this HDMI 1.4/2.0 RX Subsystem, both HDCP 1.4 and HDCP 2.3 receiver IP cores are included and can be enabled by the HDCP option in the Vivado IDE. However because HDCP 2.3 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 2.3 only
- HDCP 1.4 and HDCP 2.3

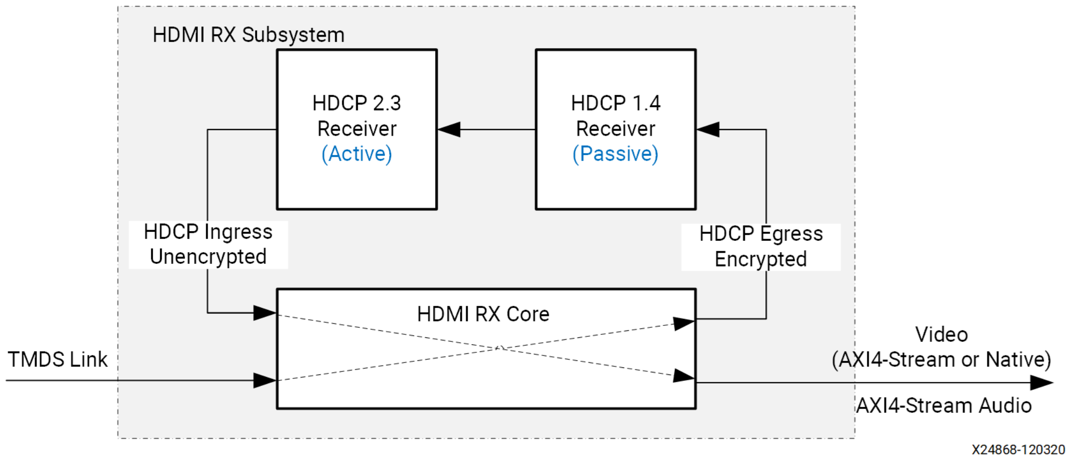
***Note*:** HDCP 2.3 is backwards compatible with HDCP 2.2.

As a guideline, HDCP 2.3 is used to decrypt content at Ultra-High Definition (UHD) while HDCP 1.4 is the legacy content protection scheme used at lower resolutions.

The following figure shows a configuration of the HDMI RX where both HDCP 1.4 and 2.3 are enabled. With both HDCP protocols enabled, the HDMI Subsystem configures itself in the cascade topology where the HDCP 1.4 and HDCP 2.3 are connected back-to-back. The HDCP Egress interface of the HDMI receiver sends encrypted audiovisual data, which is decrypted by the active HDCP block and sent back into the HDMI receiver over the HDCP Ingress interface to send to other video processing modules in the system through AXI4-Stream Video interface or Native Video interface.

The HDMI 1.4/2.0 RX Subsystem automatically handles HDCP protocol switching when both HDCP 1.4 and HDCP 2.3 protocols are included in the subsystem. Automatic protocol switching is comprised of protocol detection and protocol enablement. The protocol is detected based on the incoming DDC read/write transaction to the HDCP DDC slave device with address 0x74. The HDCP DDC slave is partitioned into a HDCP 1.4 section (0x00 thru 0x44) and a HDCP 2.3 section (0x50 thru 0xC0). The HDMI receiver subsystem detects the protocol based on which section of the HDCP DDC slave is accessed. Upon a protocol detection event, an interrupt is generated and the interrupt service routine disables and resets both protocols, then enable the detected protocol. The entire detection and enablement procedure is transparent to the user application. When the HDMI receiver subsystem includes only a single HDCP protocol, this procedure is not required.

Figure 7: **HDCP 1.4 and HDCP 2.3 over HDMI Receiver**



For more details on HDCP, see the *HDCP 1.x Product Guide* (PG224) and the *HDCP 2.2 LogiCORE IP Product Guide* (PG249).

IMPORTANT! *HDMI IP suports HDCP 2.3; however, all logs are shown as 2.2 which will be fixed in a future release.*

Standards

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

The HDMI 1.4/2.0 RX Subsystem is designed to be compliant with the HDMI 1.4b and HDMI 2.0 specification.

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

The Xilinx HDCP 2.3 is designed to be compliant with the HDCP 2.3 specification entitled High-bandwidth Digital Content Protection, Mapping HDCP to HDMI, Revision 2.3, issued by Digital Content Protection (DCP) LLC.

Performance and Resource Utilization

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

Maximum Frequencies

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

- *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892)
- *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893)
- *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS182)
- *Virtex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS183)
- *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS181)
- *Zynq-7000 SoC (Z-7007S, Z-7012S, Z-7014S, Z-7010, Z-7015, and Z-7020) Data Sheet: DC and AC Switching Characteristics* (DS187)
- *Zynq-7000 SoC (Z-7030, Z-7035, Z-7045, and Z-7100) Data Sheet: DC and AC Switching Characteristics* (DS191)
- *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922)

- *Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics* (DS923)
- *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925)
- *Zynq UltraScale+ RFSoC Data Sheet: DC and AC Switching Characteristics* (DS926)
- *Versal Prime Series Data Sheet: DC and AC Switching Characteristics* (DS956)
- *Versal AI Core Series Data Sheet: DC and AC Switching Characteristics* (DS957)
- *Versal AI Edge Series Data Sheet: DC and AC Switching Characteristics* (DS958)

Port Descriptions

The HDMI 1.4/2.0 RX Subsystem supports two types of video output stream interfaces, which eventually is mapped to the HDMI 1.4/2.0 RX Subsystem VIDEO_OUT interface.

- AXI4-Stream Video interface
- Native Video Interface
- Native Video Vectored DE

AXI4-Stream Pinouts

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

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

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

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

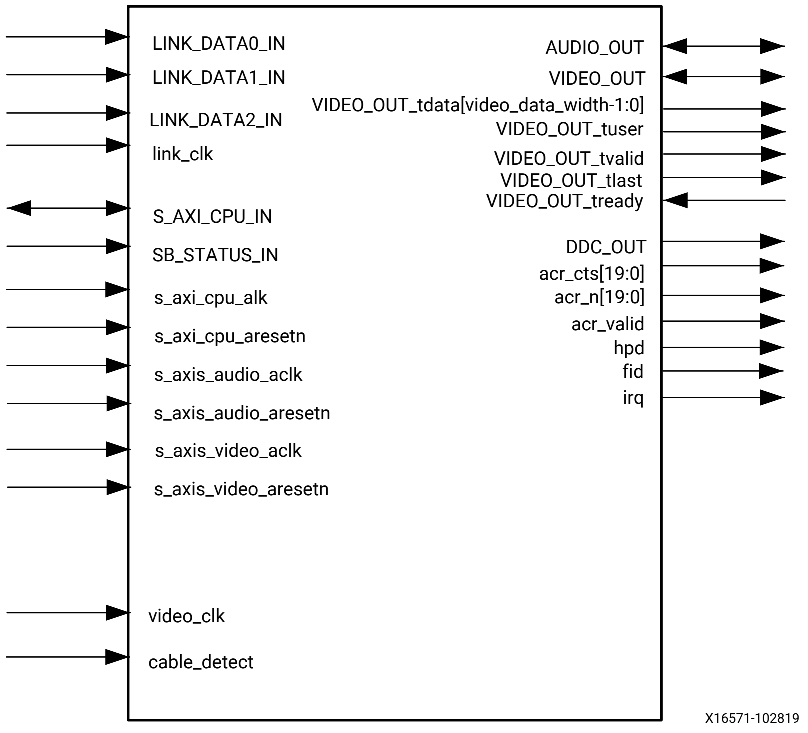


Figure 9: **HDMI RX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 1.4 Only)**

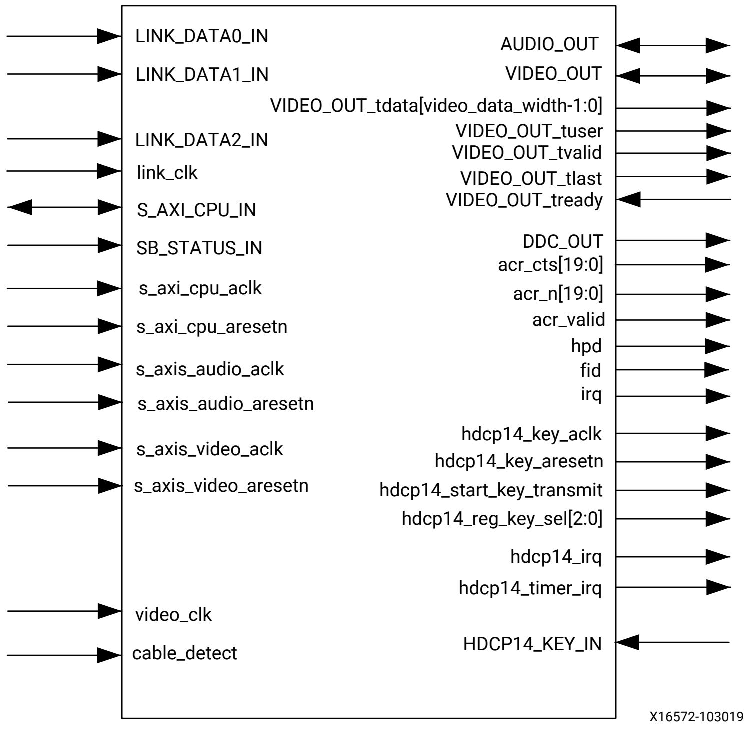


Figure 10: **HDMI RX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 2.3 Only)**

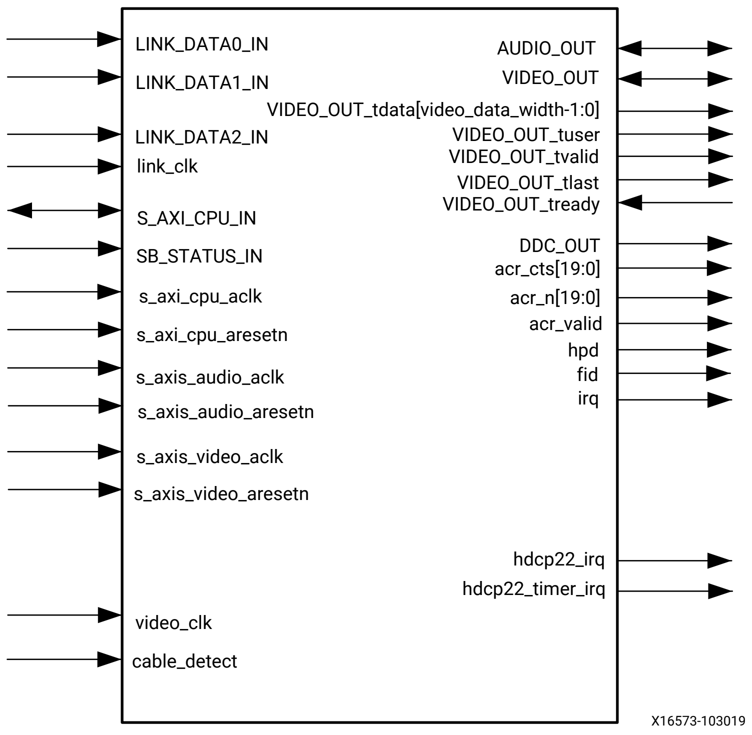
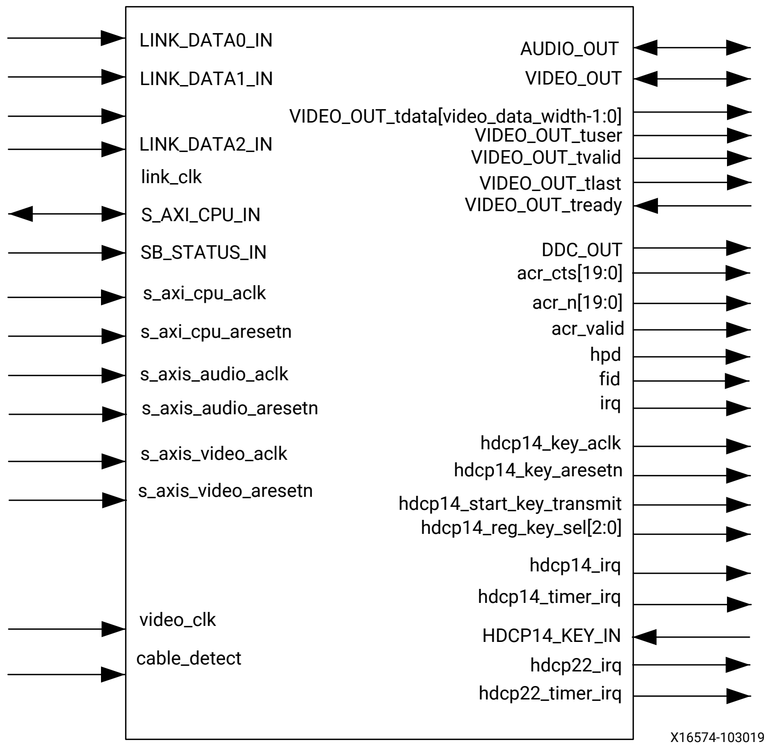


Figure 11: **HDMI RX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 1.4 and HDCP 2.3)**



Native Video Pinouts

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

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

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

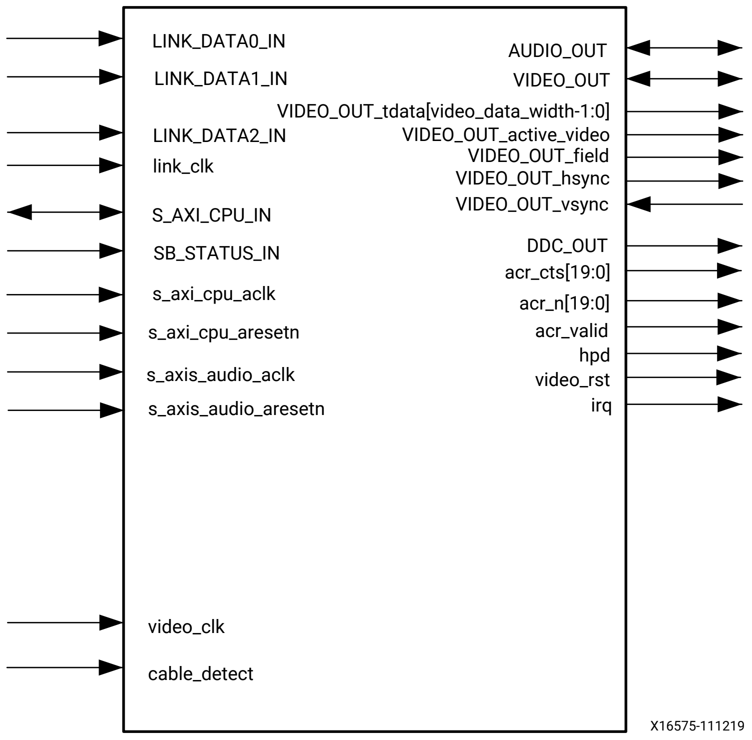


Figure 13: **HDMI RX Subsystem Pinout – Native Video Interface (HDCP 1.4 Only)**

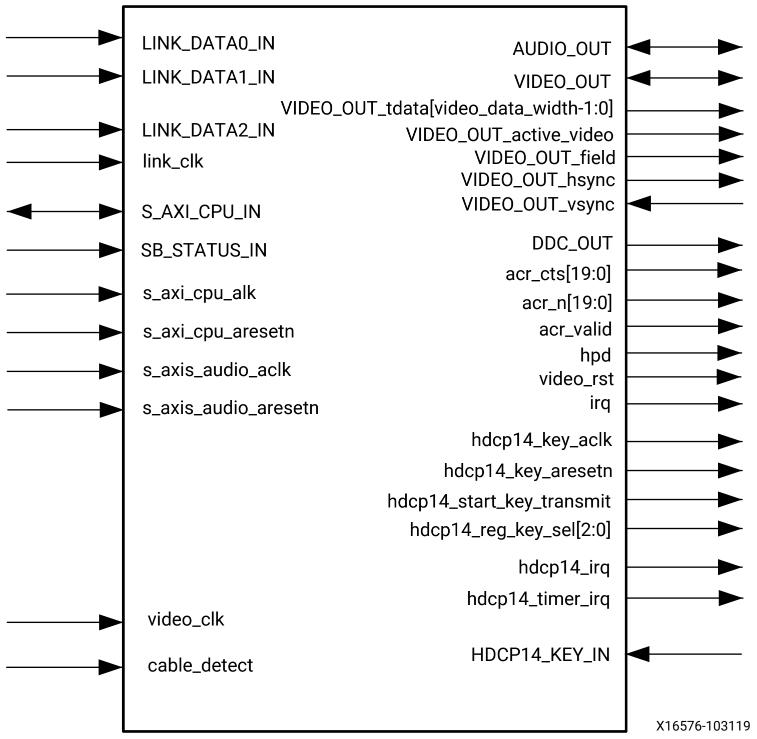


Figure 14: **HDMI RX Subsystem Pinout – Native Video Interface (HDCP 2.3 Only)**

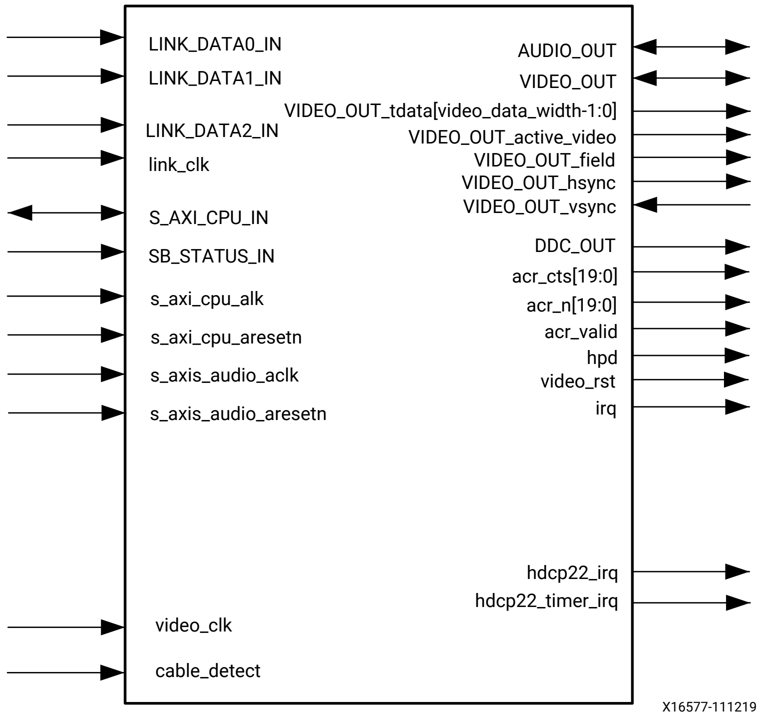
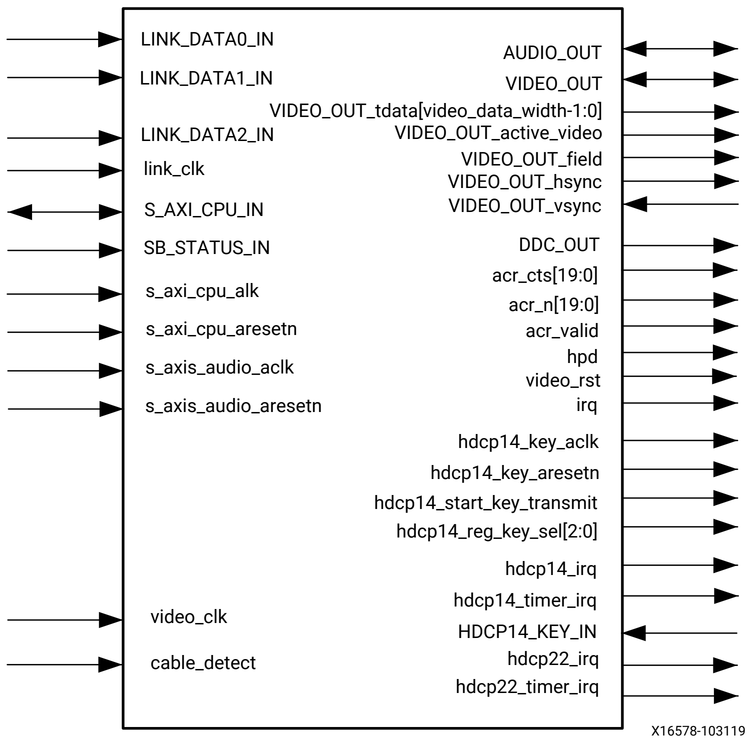


Figure 15: **HDMI RX Subsystem Pinout – Native Video Interface**



Native Video (Vectored DE)

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

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

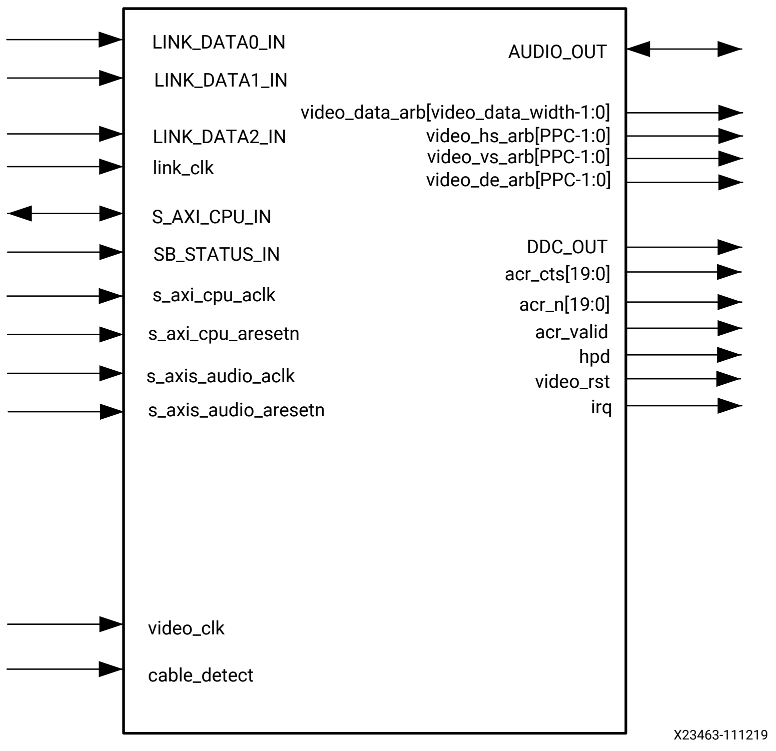


Figure 17: **HDMI RX Subsystem Pinout – Native Video (Vectored DE) (HDCP 1.4 Only)**

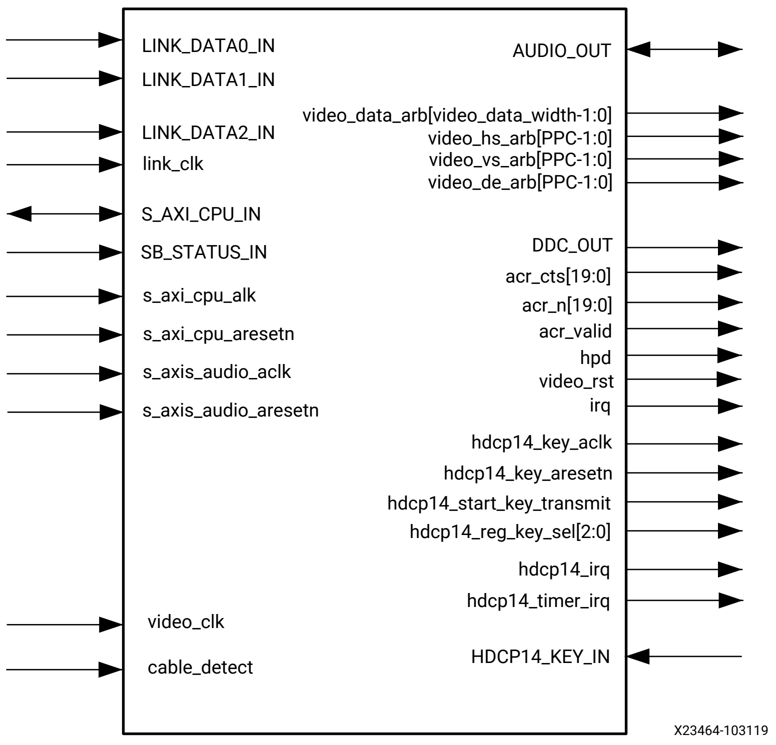


Figure 18: **HDMI RX Subsystem Pinout – Native Video (Vectored DE) (HDCP 2.3 Only)**

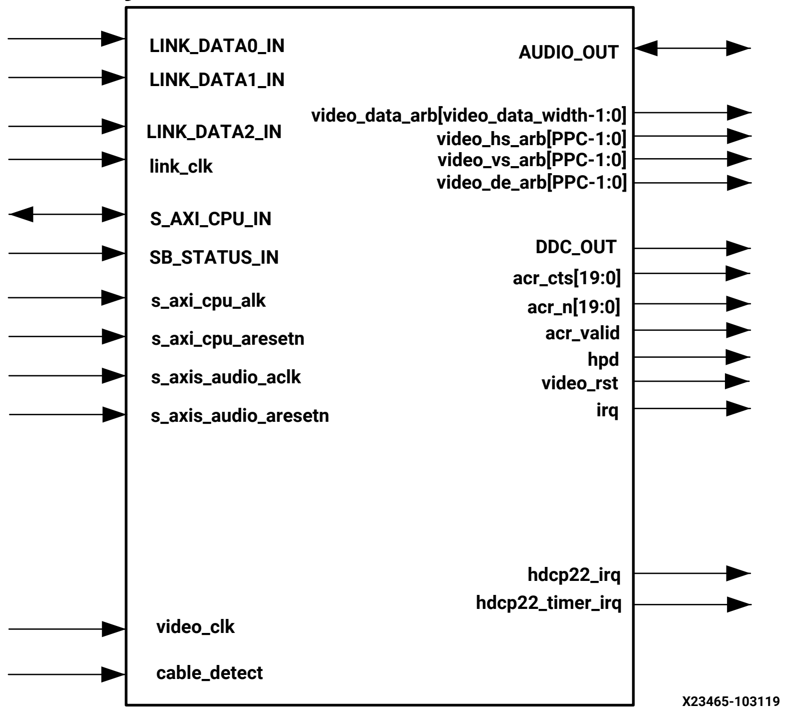
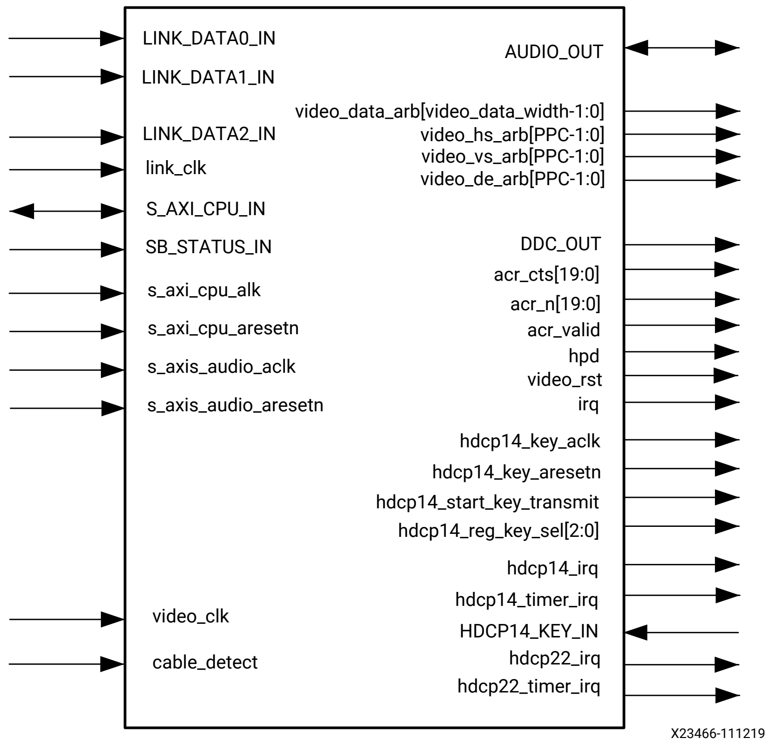


Figure 19: **HDMI RX Subsystem Pinout – Native Video (Vectored DE) (HDCP 1.4 and HDCP 2.3)**



CPU Interface

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

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

Table 2: **CPU Interface Ports**

Name	I/O	Width	Description
s_axi_cpu_aresetn	I	1	Reset (Active-Low)
s_axi_cpu_aclk	I	1	Clock for AXI4-Lite control interface
S_AXI_CPU_IN_awaddr	I	19	Write address
S_AXI_CPU_IN_awprot	I	3	Write address protection
S_AXI_CPU_IN_awvalid	I	1	Write address valid
S_AXI_CPU_IN_awready	O	1	Write address ready
S_AXI_CPU_IN_wdata	I	32	Write data
S_AXI_CPU_IN_wstrb	I	4	Write data strobe
S_AXI_CPU_IN_wvalid	I	1	Write data valid
S_AXI_CPU_IN_wready	O	1	Write data ready
S_AXI_CPU_IN_bresp	O	2	Write response
S_AXI_CPU_IN_bvalid	O	1	Write response valid
S_AXI_CPU_IN_bready	I	1	Write response ready
S_AXI_CPU_IN_araddr	I	19	Read address
S_AXI_CPU_IN_arprot	I	3	Read address protection
S_AXI_CPU_IN_arvalid	I	1	Read address valid
S_AXI_CPU_IN_aready	O	1	Read address ready
S_AXI_CPU_IN_rdata	O	32	Read data
S_AXI_CPU_IN_rresp	O	2	Read data response
S_AXI_CPU_IN_rvalid	O	1	Read data valid
S_AXI_CPU_IN_rready	I	1	Read data ready

AXI4-Stream Video Interface

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

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

Name	I/O	Width	Description
s_axis_video_aclk	I	1	AXI4-Stream clock
s_axis_video_aresetn	I	1	Reset (Active-Low)
VIDEO_OUT_tdata	O	(int((3*BPC*PPC+7)/8))*8	Data
VIDEO_OUT_tlast	O	1	End of line
VIDEO_OUT_tready	I	1	Ready
VIDEO_OUT_tuser	O	1	Start of frame
VIDEO_OUT_tvalid	O	1	Valid

Notes:

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

Native Video Interface

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

Table 4: **Native Video Interface**

Name	I/O	Width	Description
video_clk[1]	I	1	Video clock
VIDEO_OUT_field	O	1	Field ID (only for interlaced video)
VIDEO_OUT_active_video	O	1	Active video
VIDEO_OUT_data	O	video_data_width[3]	Data
VIDEO_OUT_hsync	O	1	Horizontal sync
VIDEO_OUT_vsync	O	1	Vertical sync

Notes:

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

Native Video (Vectored DE) Interface

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

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

Name	I/O	Width	Description
video_clk[1]	I	1	Video clock
video_data_arb	O	video_data_width[2]	Data
video_hs_arb	O	PPC	Horizontal sync
video_vs_arb	O	PPC	Vertical sync
video_de_arb	O	PPC	Active video data enable

Notes:

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

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

Table 6: **Supported Video Timing**

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

Audio Output Stream Interface

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

Table 7: **Audio Output Stream Interface**

Name	I/O	Width	Description
s_axis_audio_aclk	I	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	I	1	Reset (Active-Low)
AUDIO_OUT_tdata	O	32	Data [31] P (Parity) [30] C (Channel status) [29] U (User bit) [28] V (Validity bit) [27:4] Audio sample word [27:4] Audio sample word for L-PCM/3D-Audio [27:12] Audio sample word for HBR [3:0] Preamble code 4'b0001 Subframe 1/Left Channel and start of audio block 4'b0010 Subframe 1/Left Channel 4'b0011 Subframe 2/Right Channel
AUDIO_OUT_tid	O	8	Channel ID
AUDIO_OUT_tready	I	1	Ready
AUDIO_OUT_tvalid	O	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.

The subsystem should set up the CTS and N parameters before asserting the valid signal.

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

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

Name	I/O	Width	Description
acr_cts	O	20	CTS
acr_n	O	20	N
acr_valid	O	1	Valid

HDMI Link Input Interface

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

Table 9: **HDMI Link Interface**

Name	I/O	Width	Description
link_clk	I	1	Link clock
LINK_DATA0_IN_tdata	Input	ppc*10	Link data 0
LINK_DATA0_IN_tvalid	Input	1	Link Data 0 Valid
LINK_DATA1_IN_tdata	Input	ppc*10	Link data 1
LINK_DATA1_IN_tvalid	Input	1	Link Data 1 Valid
LINK_DATA2_IN_tdata	Input	ppc*10	Link data 2
LINK_DATA2_IN_tvalid	Input	1	Link Data 2 Valid

Data Display Channel Interface

The following table shows the Data Display Channel interface signals.

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

Name	I/O	Width	Description
ddc_scl_i	I	1	DDC serial clock in
ddc_scl_o	O	1	DDC serial clock out
ddc_scl_t	O	1	DDC serial clock tri-state
ddc_sda_i	I	1	DDC serial data in
ddc_sda_o	O	1	DDC serial data out
ddc_sda_t	O	1	DDC serial data tri-state

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

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

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

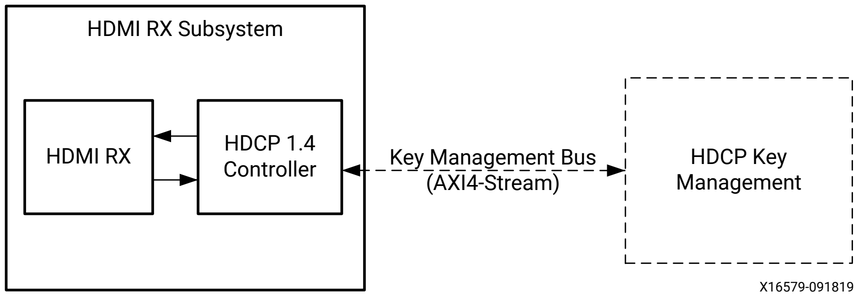
Name	I/O	Width	Description
HDCP_KEY_IN_tdata	I	64	HDCP 1.4 key data
HDCP_KEY_IN_tlast	I	1	End of key data
HDCP_KEY_IN_tready	O	1	Ready
HDCP_KEY_IN_tuser	I	8	Start of key data
HDCP_KEY_IN_tvalid	I	1	Valid
hdcp14_key_aclk	O	1	AXI4-Stream clock
hdcp14_key_aresetn	O	1	Reset (Active-Low)
hdcp14_start_key_transmit	O	1	Start key transmit

Table 11: **HDCP 1.4 Key Input Interface** *(cont'd)*

Name	I/O	Width	Description
hdcp14_reg_key_sel	O	3	Key select

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

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



However, the HDCP 2.3 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 2.3 drivers. For the detailed list of keys that are required to be loaded by the user application, see the *HDCP 2.2 LogiCORE IP Product Guide* (PG249).

HDCP 2.3 Interrupt Outputs

The following table shows the signals for HDCP 2.3 interrupt output ports.

Table 12: **HDCP 2.3 Interrupt Output Interface**

Name	I/O	Width	Description
hdcp22_irq	O	1	HDCP 2.3 interrupt
hdcp22_timer_irq	O	1	HDCP 2.3 timer interrupt

HDCP 1.4 Interrupt Outputs

The following table shows the signals for HDCP 1.4 interrupt output ports.

Table 13: **HDCP 1.4 Interrupt Output Interface**

Name	Direction	Width	Description
Hdcp14_irq	Output	1	HDCP 1.4 interrupt
Hdcp14_timer_irq	Output	1	HDCP 1.4 timer interrupt

Miscellaneous Signals with AXI4-Stream Video Interface

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

Table 14: **Miscellaneous Signals**

Name	I/O	Width	Description
hpd	O	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
cable_detect	I	1	If XGUI option: Cable Detect active-High (Default) 0 - Cable Detect is released 1 - Cable Detect is asserted If XGUI option: Cable Detect active-Low [2] 0 - Cable Detect is asserted 1 - Cable Detect is released
irq	O	1	Interrupt request for CPU. Active-High.
video_clk	I	1	Reference Native Video Clock When AXI4-Stream is selected as Video Interface, a Video In to AXI4-Stream Bridge module is added to the HDMI RX Subsystem to convert Native Video into AXI4-Stream Video. HDMI RX core uses this `video_clk` to clock out the video data.
SB_STATUS_IN_tdata	I	2	Side Band Status input signals Bit 0: link_rdy Bit 1: video_rdy
SB_STATUS_IN_tvalid	I	1	Side Band Status input valid

Table 14: **Miscellaneous Signals** *(cont'd)*

Name	I/O	Width	Description
fid	I	1	Field ID for AXI4-Stream bus. Used only for interlaced video. 0 - even field 1 - odd field For progress video the output is always Low.

Notes:

1. The Hot Plug Detect (HPD) signal is driven by an HDMI sink and asserted when the HDMI cable is connected to notify the HDMI source of the presence of an HDMI sink. When designing an HDMI sink system using the HDMI 1.4/2.0 RX Subsystem, in the PCB, if you choose to use a voltage level shifter, the HPD polarity remains as active-High. However, if you choose to add an inverter to the HPD signal, then the HPD polarity must be set to active-Low in the HDMI 1.4/2.0 RX Subsystem GUI. There are two common ways of using HPD: Toggle HPD to trigger HDCP authentication process (usually 100 ~ 500 ms). Or a longer HPD toggle (>1s), the HDMI sink is notifying the source its present without cable unplug and plug. The software API used to assert and release HPD is `XV_HdmiRxSs_SetHpd`.
2. The Cable Detect signal is connected to a 5V power signal from the HDMI cable connector via some level shifter to notify the HDMI 1.4/2.0 RX Subsystem that an HDMI source is connected.

Miscellaneous Signals with Native Video Interface

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

Table 15: **Miscellaneous Signals with Native Video Interface**

Name	I/O	Width	Description
hpd	I	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
cable_detect	I	1	If XGUI option: Cable Detect Active-High (Default) 0 - Cable Detect is released 1 - Cable Detect is asserted If XGUI option: Cable Detect Active-Low [2] 0 - Cable Detect is asserted 1 - Cable Detect is released
irq	O	1	Interrupt request for CPU. Active-High.
SB_STATUS_IN_tdata	I	2	Side Band Status input signals Bit 0: link_rdy Bit 1: video_rdy
SB_STATUS_IN_tvalid	I	1	Side Band Status input valid

Table 15: **Miscellaneous Signals with Native Video Interface** *(cont'd)*

Name	I/O	Width	Description
video_rst	O	1	Video reset signal in video_clk domain. Active-High.

Notes:

1. The Hot Plug Detect (HPD) signal is driven by an HDMI sink and asserted when the HDMI cable is connected to notify the HDMI source of the presence of an HDMI sink. In most cases, the HDMI sink is simply connected to 5V power signal. Therefore, in the PCB, if you choose to use a voltage divider or level shifter, the HPD polarity remains as Active-High. However, if you add an inverter to the HPD signal, then the HPD polarity must be set to Active-Low in the HDMI 1.4/2.0 RX Subsystem GUI. When designing an HDMI sink system using HDMI Receiver Subsystem, in the PCB, if you choose to use a voltage level shifter, the HPD polarity remains as Active-High. However, if you choose to add an inverter to the HPD signal, then the HPD polarity must be set to Active-Low in HDMI Receiver Subsystem GUI. There are two common ways of using HPD: Toggle HPD to trigger HDCP authentication process (usually 100 ~ 500 ms). Or a longer HPD toggle (>1s), the HDMI sink is notifying the source its present without cable unplug and plug. The software API used to assert and release HPD is XV_HdmiRxSs_SetHpd.
2. The Cable Detect signal is connected to a 5V power signal from the HDMI cable connector through some level shifter to notify the HDMI 1.4/2.0 RX Subsystem that an HDMI source is connected.

Clocks and Resets

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

Table 16: **Clocks and Resets**

Name	I/O	Width	Description
s_axi_cpu_aclk	I	1	AXI4-Lite CPU control interface clock.
s_axi_cpu_aresetn	I	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	I	1	AXI4-Stream video output clock.
s_axis_video_aresetn	I	1	Reset, associated with `s_axis_video_aclk` (active-Low).Resets the AXI4-Stream data path for the video output.
s_axis_audio_aclk	I	1	AXI4-Stream Audio output clock. (The audio streaming clock must be greater than or equal to 128 times the audio sample frequency)
s_axis_audio_aresetn	I	1	Reset, associated with `s_axis_audio_aclk` (active-Low). Resets the AXI4-Stream data path for the audio output.
link_clk	I	1	HDMI Link data output clock. This connects to the Video PHY Controller/HDMI GT Subsystem link clock output.
video_clk	I	1	Clock for the native video interface.

Notes:

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

Related Information

Clocking
Resets

Chapter 4

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

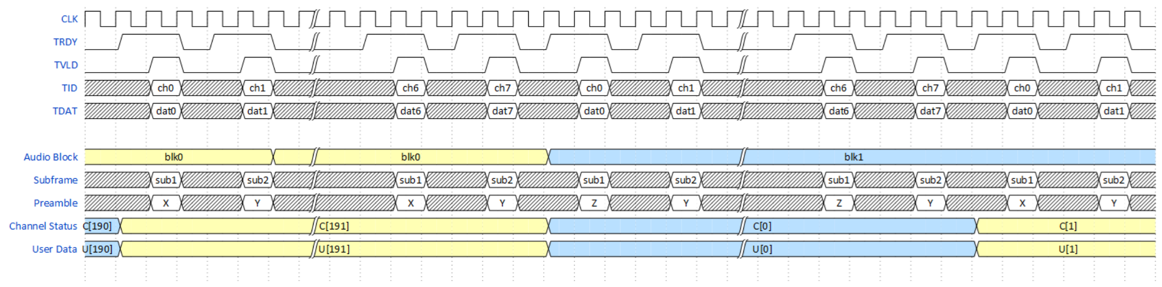
Control Interface

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

Audio Data Stream

An AXI4-Stream audio cycle is illustrated in the following figure. The data is valid when both the valid (TVLD) and ready (TRDY) signals are asserted. The HDMI 1.4/2.0 RX Subsystem sends out adjacent channels in sequential order (CH0, CH1, etc). Usually, the audio stream receiver also 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. Audio signals are defined in Table 7: Audio Output Stream Interface.

Figure 21: **Audio Cycle**



In the HDMI 1.4/2.0 RX Subsystem, the number of Audio Channels is set through the software driver. You must enable the correct number of audio channels 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 the Audio Channel number to 8 in the HDMI 1.4/2.0 RX Subsystem driver. Then, the corresponding audio data must be prepared and sent to the HDMI 1.4/2.0 RX Subsystem in the hardware, as described in the previous figure.

If an HDMI system does not require audio, tie all audio related input ports Low. The ports are:

- s_axis_audio_aresetn
- s_axis_audio_aclk
- AUDIO_OUT_tready

Audio related output can be left unconnected. The ports are:

- AUDIO_OUT (except tready)
- acr_cts
- acr_n
- acr_valid

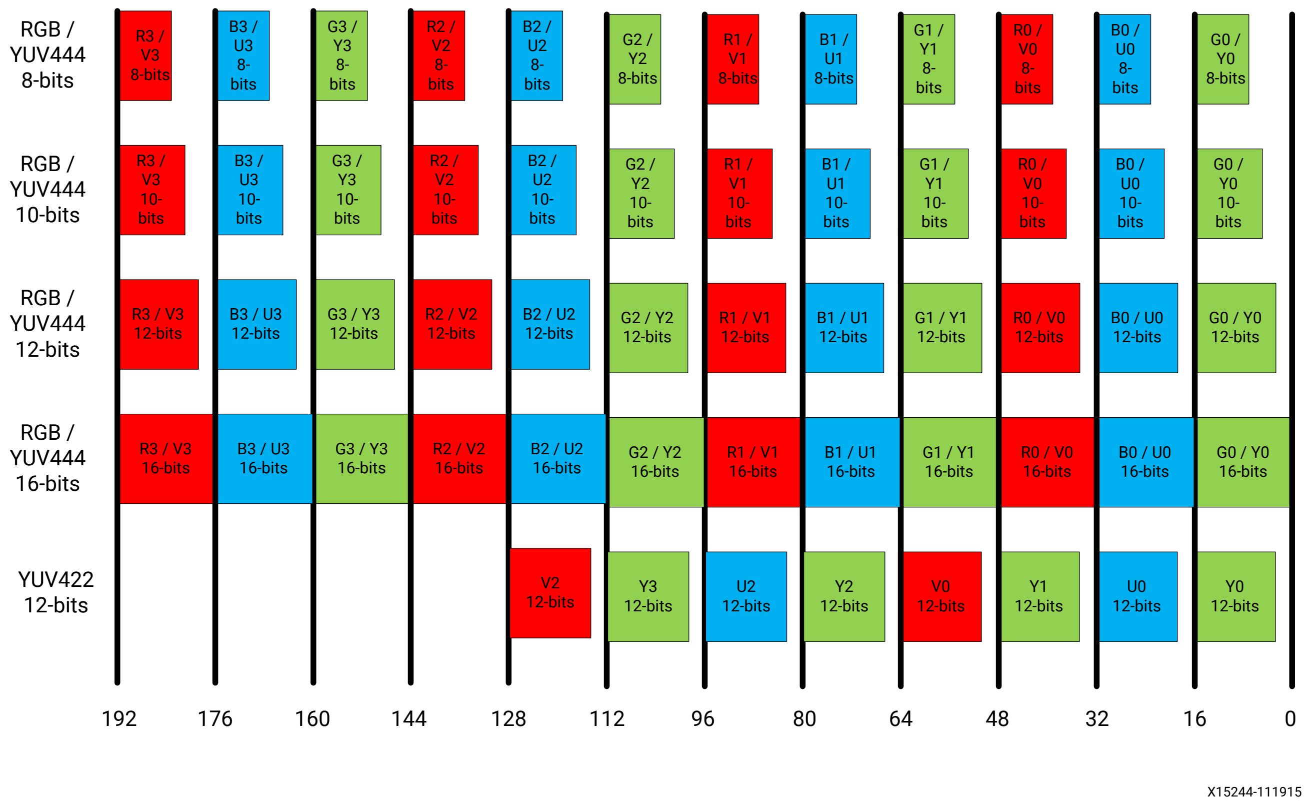
In the HDMI 1.4/2.0 RX Subsystem, L-PCM (IEC 60958, Packet Type 0x02), HBR (Packet Type 0x09), and 3D Audio (Packet Type 0x0B) are handled by the hardware. For Aux packet with packet type = 0x02, the audio data are routed to the audio interface. Therefore, even IEC 61937 compressed audio is available at the audio interface. In this case, you must develop your module to retrieve and uncompress the audio.

Video Output Stream Interface

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

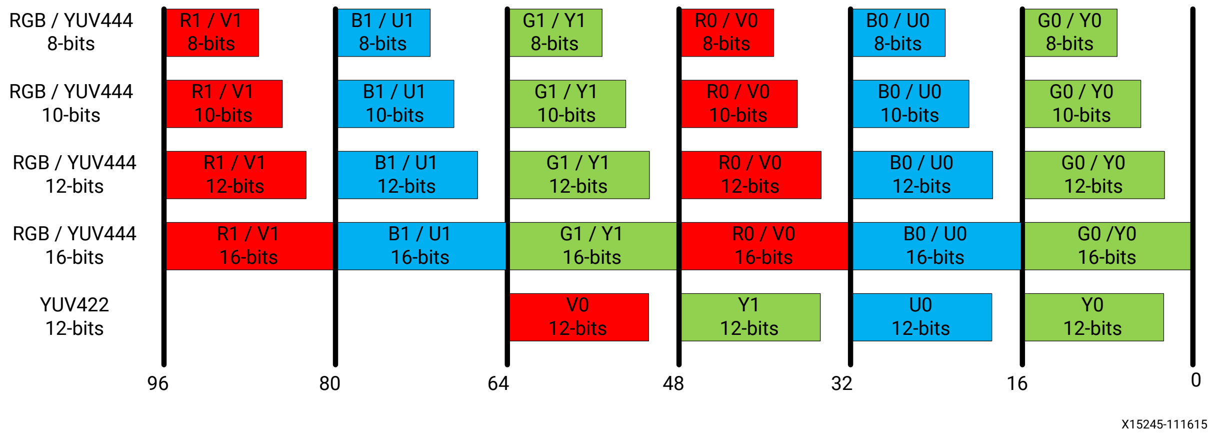
When the parameter, **Max Bits Per Component**, is set to 16, the following figure shows the data format for quad pixels per clock to be fully compliant with the AXI4-Stream video protocol.

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



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

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



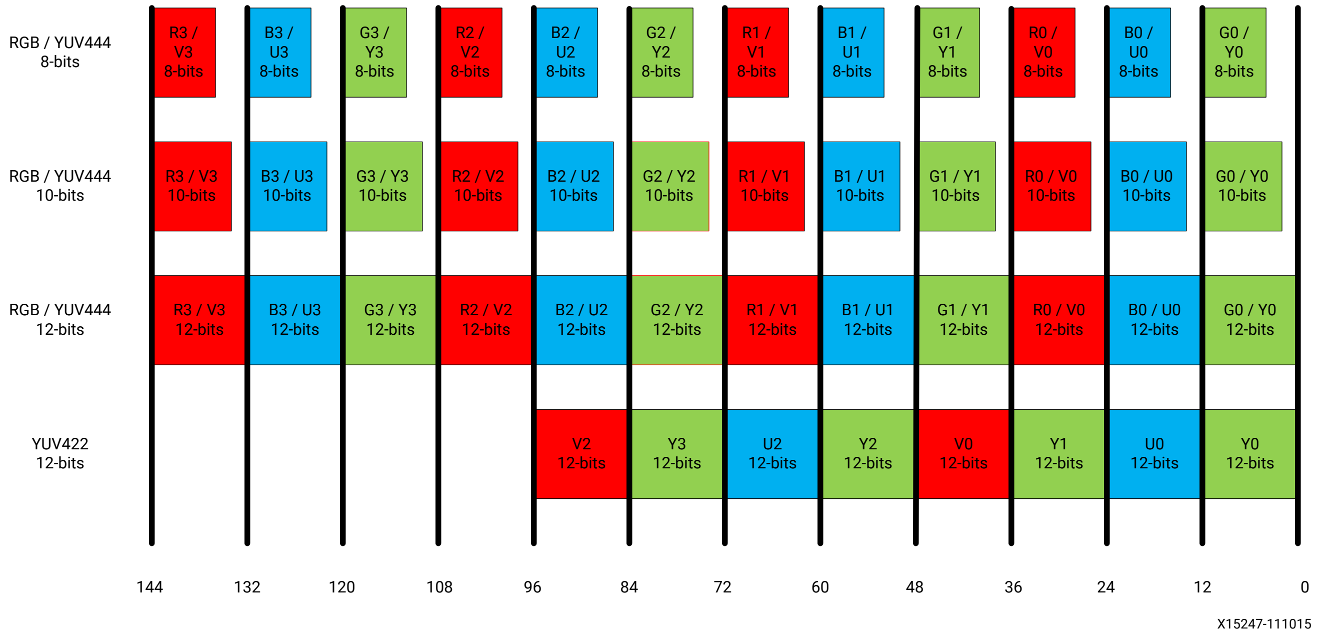
When the parameter, Max Bits Per Component, is set to 12, video formats with actual bits per component larger than 12 is truncated to the Max Bits Per Component. The remaining least significant bits are discarded. If the actual bits per component is smaller than Max Bits Per Component set in the Vivado 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 17: **Max Bits Per Component Support**

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

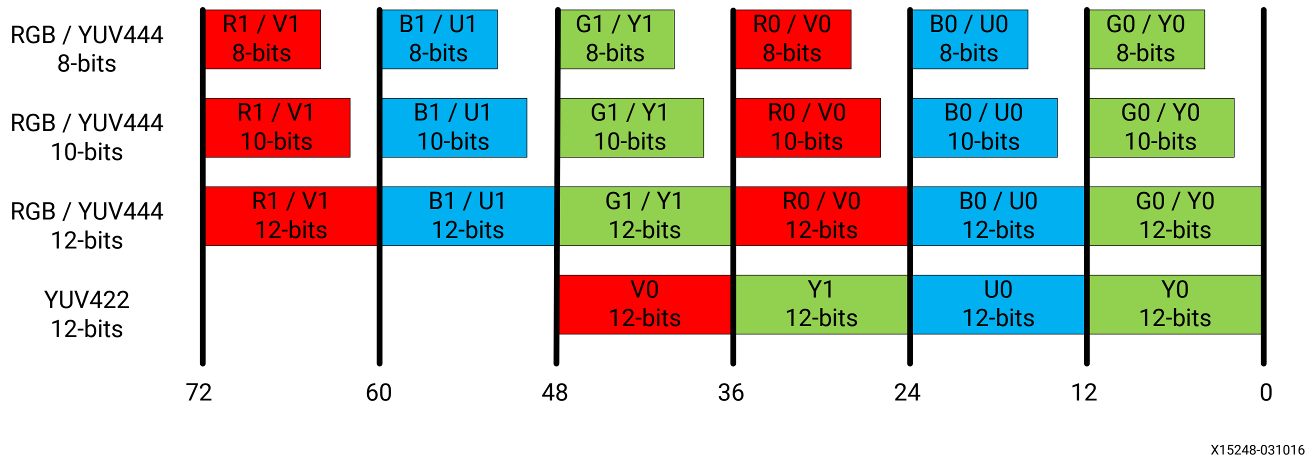
As an illustration, when **Max Bits Per Component** is set to 12, the following figure shows the data format for quad pixels per clock to be fully compliant with the AXI4-Stream video protocol.

Figure 24: **Quad Pixels Data Format (Max Bits Per Component = 12)**



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

Figure 25: **Dual Pixels Data Format (Max Bits Per Component = 12)**



The video interface can also transport quad and dual pixels in the YUV420 color space. The following figures show the data format for quad and dual pixels formats, respectively.

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

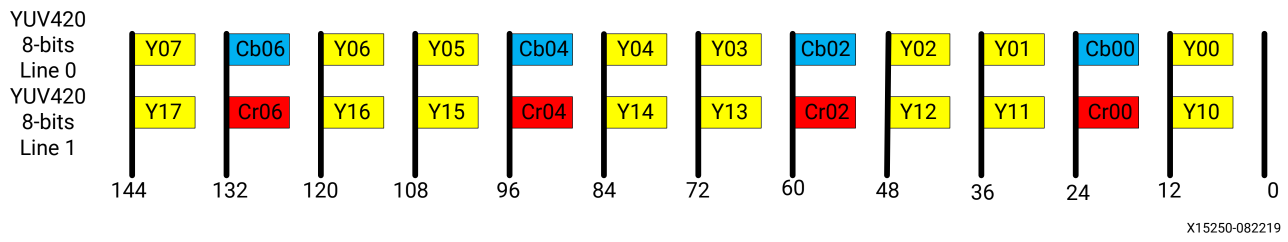
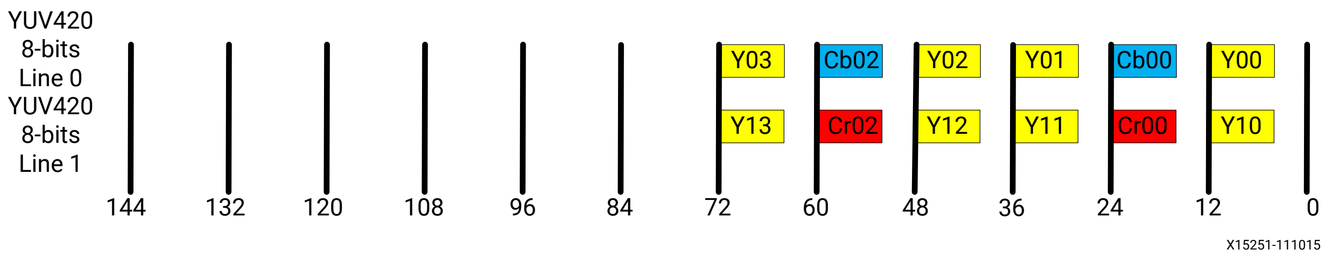
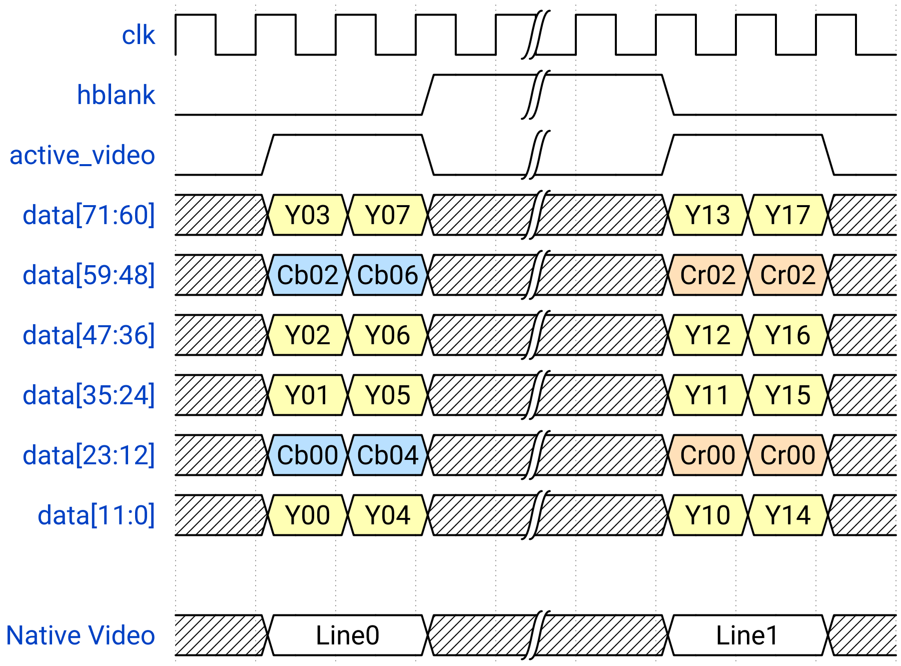


Figure 27: **YUV420 Color Space Dual Pixels Data Format (Native)**



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

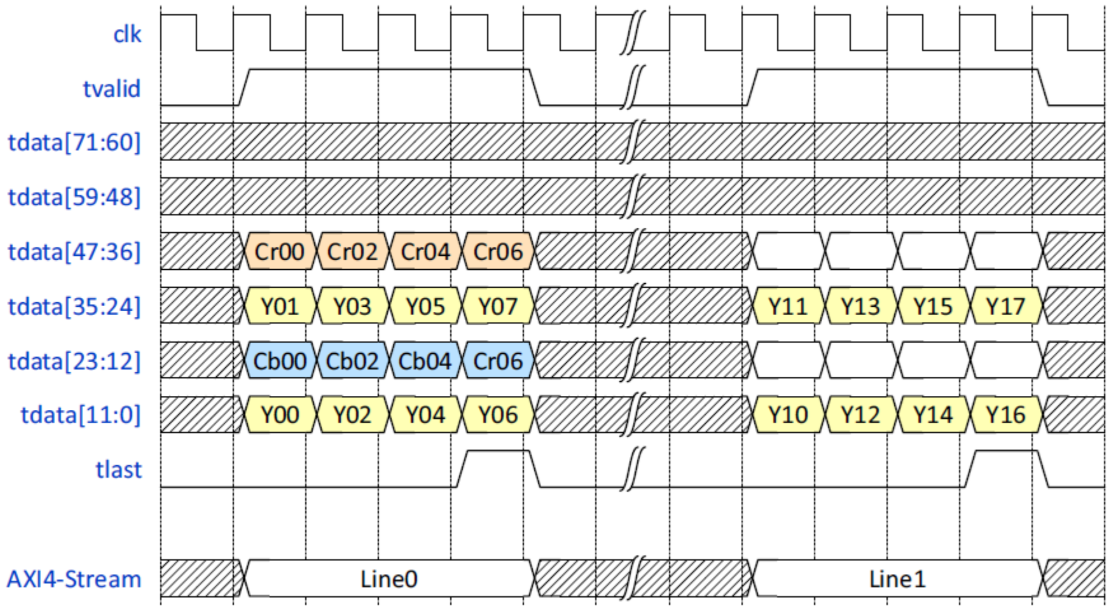
Figure 28: **Native HDMI Video Interface**



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

However, in the native HDMI video interface, the video data representation must be as shown in the following figure.

Figure 29: **YUV 4:2:0 AXI4-Stream Video Data (Dual Pixel per Clock)**



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

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

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 supported video timing for different PPC, see Table 6: Supported Video Timing.

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

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 1.4/2.0 RX Subsystem you must provide an audio clock to the application. This can be achieved by using an internal PLL or external clock source.

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

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

Figure 30: **HDMI Clocking Structure**

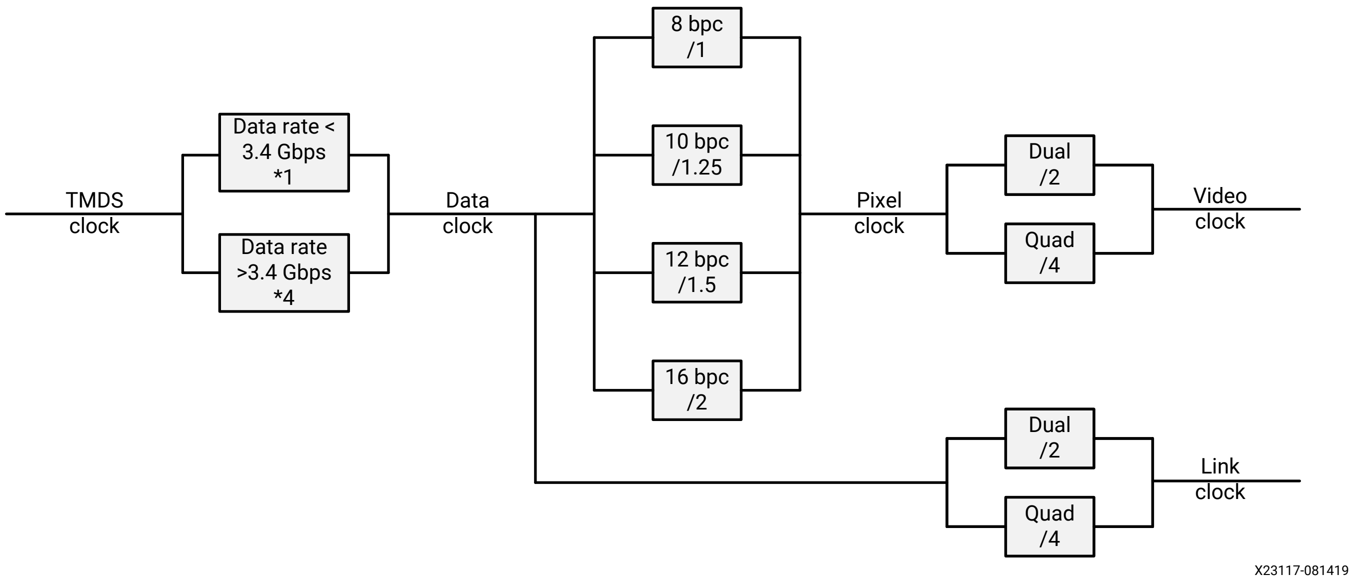


Table 18: **HDMI Clocking**

Clock	Function	Freq/Rate	Example[1]
TMDS	Source synchronous clock to HDMI interface (This is the actual clock on the HDMI cable).	= 1/10 data rate (for data rates < 3.4 Gb/s)	Data rate = 2.97 Gb/s TMDS clock = 2.97/10 = 297 MHz
		= 1/40 data rate (for data rates > 3.4 Gb/s)	Data rate = 5.94 Gb/s TMDS clock = 5.94/40 = 148.5 MHz
Data	This is the actual data rate clock. This clock is not used in the system. It is only listed to illustrate the clock relations.	= TMDS clock (for data rates < 3.4 Gb/s)	Data rate = 2.97 Gb/s Data clock = TMDS clock * 1 = 297 MHz
		= TMDS clock * 4 (for data rates > 3.4 Gb/s)	Data rate = 5.94 Gb/s Data clock = TMDS clock * 4 = 594 MHz TMDS clock = 148.5 MHz

Link	Clock used for data interface between the Video PHY layer module and subsystem	For dual pixel video:
		Clock=data clock/2
		For quad pixel video:
		clock=data clock/4
			TMDS clock = 297 MHz
			Data clock = 297 MHz
			Link clock = 297 MHz/2=148.5 MHz for dual pixel wide interface
			Link clock = 297 MHz/4 = 74.25 MHz for quad pixel wide interface
			Data clock = 594 MHz
			Link clock = 594 MHz/2=297 MHz for dual pixel wide interface
			Link clock = 594 MHz/4=148.5 MHz for quad pixel wide interface

Pixel	This is the internal pixel clock. This clock is not used in the system. It is only listed to illustrate the clock relations.	For 8 bpc pixel clock = data clock For 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	Data clock = 297 MHz For 8 bpc pixel clock = 297 MHz For 10 bpc pixel clock = 297/1.25 = 237.6 MHz For 12 bpc pixel clock = 297/1.5 = 198 MHz For 16 bpc pixel clock = 297/1.5 = 148.5 MHz
Video	Clock used for video interface	For dual pixel video clock = pixel clock/2 For quad pixel video clock = pixel clock/4	297 MHz/2 = 148.5 MHz for dual pixel wide interface 297 MHz/4 = 74.25 MHz for quad pixel wide interface For more information on how to choose the correct PLL in the targeted devices, see the *Video PHY Controller LogiCORE IP Product Guide* (PG230) for non-Versal devices and the *HDMI GT Controller LogiCORE IP Product Guide* (PG334) for Versal devices.

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 RX Subsystem with different GT devices, calculate the clock frequencies and make sure the targeted device can support it. When using the HDMI 1.4/2.0 RX Subsystem with the Xilinx Video PHY Controller IP core, more information can be found in the *Video PHY Controller LogiCORE IP Product Guide* (PG230). When using the HDMI 1.4/2.0 RX Subsystem with the Xilinx HDMI GT Subsystem IP core (Versal devices), more information can be found in the *HDMI GT Controller LogiCORE IP Product Guide* (PG334).

IMPORTANT! *The HDMI 1.4/2.0 RX Subsystem supports both integer and non-integer frame rates . However the HDMI 1.4/2.0 RX Subsystem reports frame rate information based on the Video Identification Code (VIC) ID. The VIC ID is common for integer and non-integer frame rates and the HDMI 1.4/2.0 RX Subsystem always reports the integer frame rate. To determine the non-integer frame rate, Xilinx recommends you read the receiver Frequency Register (0x210). For more details, refer to the register definitions in the Video PHY Controller LogiCORE IP Product Guide (PG230) or HDMI GT Controller LogiCORE IP Product Guide (PG334).*

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

Table 19: **Example Settings**

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

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

Therefore,

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

Using the associative property in this example,

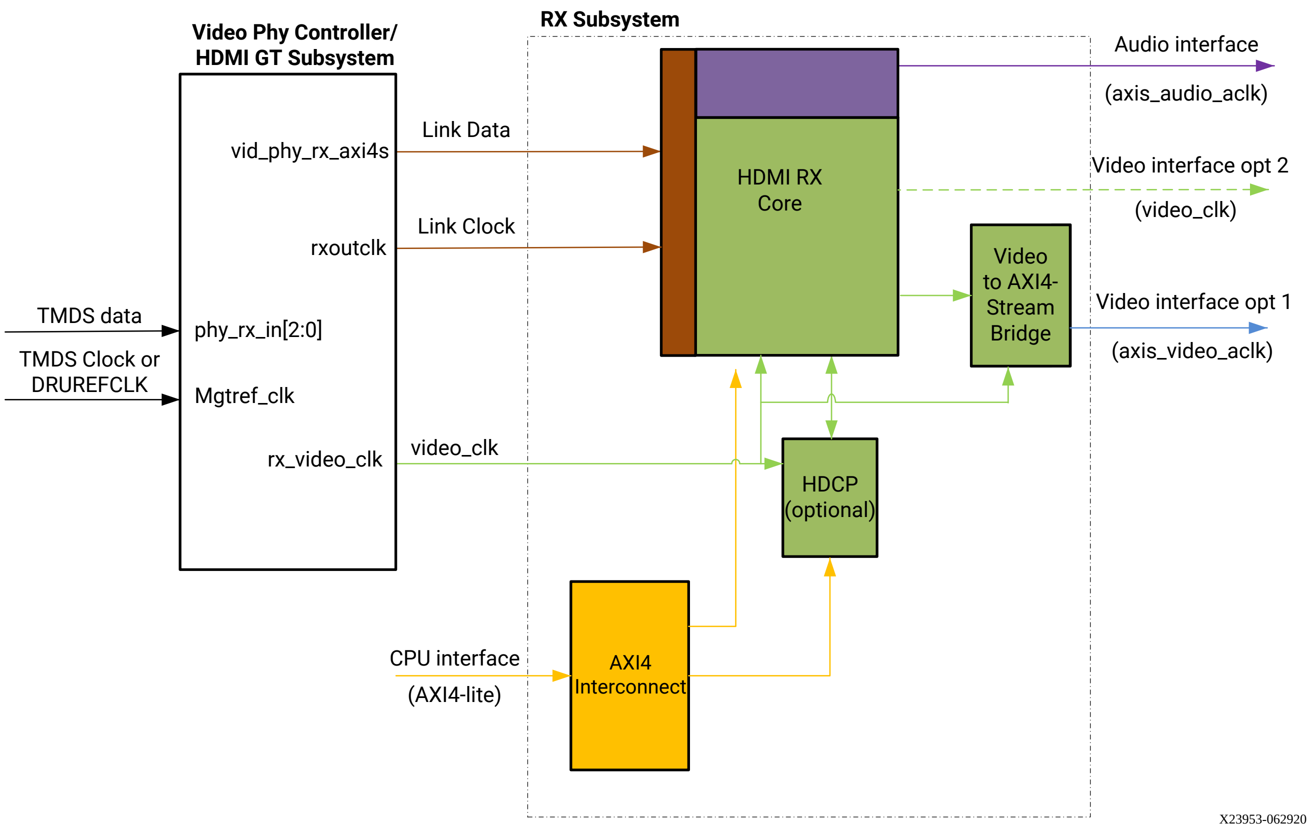
Data clock = 222.75 MHz < 340 MHz

then

TMDS clock = Data clock = 222.75 MHz

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

Figure 31: **HDMI 1.4/2.0 RX Subsystem and Video PHY Controller/HDMI GT Subsystem**



The HDMI 1.4/2.0 RX Subsystem can support either AXI4-Stream video or native video.

The Video PHY Controller/HDMI GT Subsystem converts serial TMDS data into parallel Link Data and sends to the HDMI 1.4/2.0 RX Subsystem at `Link Clock`. The HDMI RX core then unpacks the Link Data into native video stream, audio data, and other auxiliary data.

- When AXI4-Stream is selected, a Video In to AXI4-Stream bridge core in the HDMI 1.4/2.0 RX Subsystem helps to covert the native video stream into AXI4-Stream, and then outputs from the HDMI 1.4/2.0 RX Subsystem at `axis_video_aclk`.
- When the native (or native vectored DE) interface is selected, the native video stream is sent out from the HDMI 1.4/2.0 RX Subsystem directly from HDMI RX core at the `video_clk` rate.

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

Resets

Each AXI input interface has its own reset signal. The reset signals, `s_axi_cpu_aresetn`, `s_axis_video_aresetn` and `s_axis_audio_aresetn` are for `S_AXI_CPU_IN`, `VIDEO_OUT` (AXI4-Stream Video Interface), and `AUDIO_OUT` 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_OUT interface when Native Video interface is selected. However, the HDMI 2.1 RX Subsystem outputs a `video_rst` signal, which you can use to reset its supporting Native Video Source processing modules.

Chapter 5

Design Flow Steps

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

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

Customizing and Generating the Subsystem

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

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

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

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

1. In the **Flow Navigator**, click on **Create Block Diagram** or **Open Block Design** under the IP Integrator heading.
2. Right click in the diagram and select **Add IP**.

   A searchable IP catalog opens. You can also add IP by clicking on the Add IP button on the left side of the IP Integrator Block Design canvas.
3. Click on the IP name and press the Enter key on your keyboard or double click on the IP name.

4. Double-click the selected IP block or select the **Customize Block** command from the right-click menu.

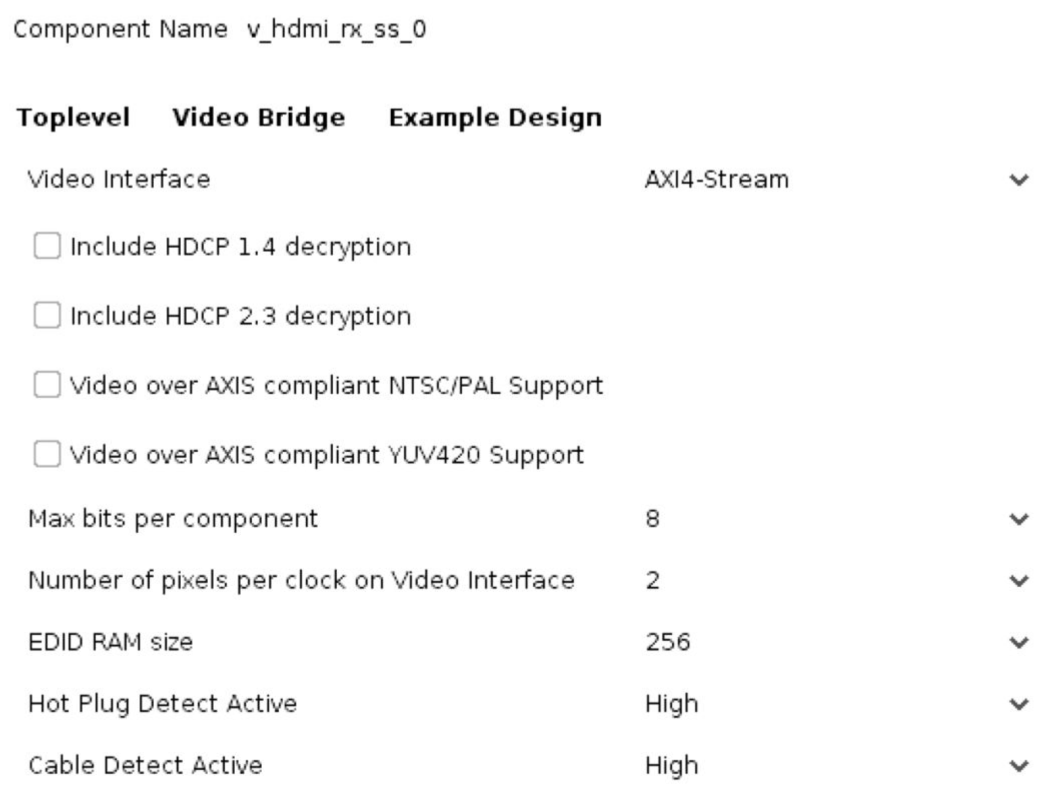
For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

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

Top Level Tab

The top level tab is shown in the following figure.

Figure 32: **Top Level Tab**



Component Name v_hdmi_rx_ss_0

Video Interface	AXI4-Stream
☐ Include HDCP 1.4 decryption	
☐ Include HDCP 2.3 decryption	
☐ 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
EDID RAM size	256
Hot Plug Detect Active	High
Cable Detect Active	High

The parameters on the top level tab are as follows.

- **Component Name:** The component name is set automatically by IP integrator.
- **Video Interface:** This option selects the video interface for the HDMI 1.4/2.0 RX Subsystem. The allowable options are AXI4-Stream, Native Video, or Native Video (Vectored DE).

- **Include HDCP 1.4 Decryption:** This option enables HDCP 1.4 decryption.
- **Include HDCP 2.3 Decryption:** This option enables HDCP 2.3 decryption.

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

- **Video over AXIS compliant NTSC/PAL Support:** This option enables the HDMI 1.4/2.0 RX Subsystem to support Video over AXIS compliant NTSC/PAL.
  - A pixel repetition of 2 is supported by current hardware.
  - The pixel repetition factor must be decoded from the received AVI InfoFrame.

  ***Note*:** The Video In to AXI4-Stream IP supports a pixel repetition of 2. It is used as a subcore in the HDMI 1.4/2.0 RX Subsystem. However, it has the limitation of supporting only RGB and YUV444 color space, but not YUV422. Therefore, when AXI4-Stream is used, pixel repetition of 2 is supported. The repeated pixel is dropped by the HDMI 1.4/2.0 RX Subsystem. When the native interface is used, none of the pixels are dropped. All pixels are sent through native video output interface, which you can use to create customized pixel replication logic to support all color spaces including 12-bit YUV422.

- **Video over AXIS Compliant YUV420 Support:**

  This option enables the HDMI 1.4/2.0 RX Subsystem to support video over AXIS compliant YUV420.

- **Max bits per component:** This option selects the maximum bits per component. The allowable options are, 8, 10, 12 or 16 bits. This parameter is to set the maximum "allowed" bits per component, and the actual bits per component can be set from the software API to a different value. However, the actual bits per component is bounded by the Max bits per component. For example, if the Max bits per component is set to 16, the user can set the actual bits per component from the software API to any of the values, 8, 10, 12 or 16. But if the Max bits per component is set to 8, you can only set the actual bits per component to 8 through the software API.
- **Number of pixels per clock on Video Interface:** This option selects the number of pixels per clock. The allowable options are 2 or 4 pixels.

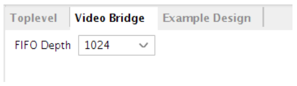
  IMPORTANT! *Pixels per clock (PPC) can only be selected at IP generation time, and must remain static in the design. Some video format with a total horizontal timing that is NOT divisible by 4 (for example, 720p60 has a total horizontal pixel of 1650, which is not divisible by 4) are not supported. If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

- **Hot Plug Detect Active:** This option selects the HPD active polarity. The allowable options are High or Low.
- **EDID RAM size:** The allowable options are, 256, 512, 1024 or 4096.
- **Cable Detect Active:** This option selects the Cable Detect active polarity. The allowable options are High or Low.

Video Bridge Tab (Video AXI4-Stream Interface Only)

The Video Bridge tab is shown in the following figure.

Figure 33: **Video Bridge Tab**



FIFO Depth 1024

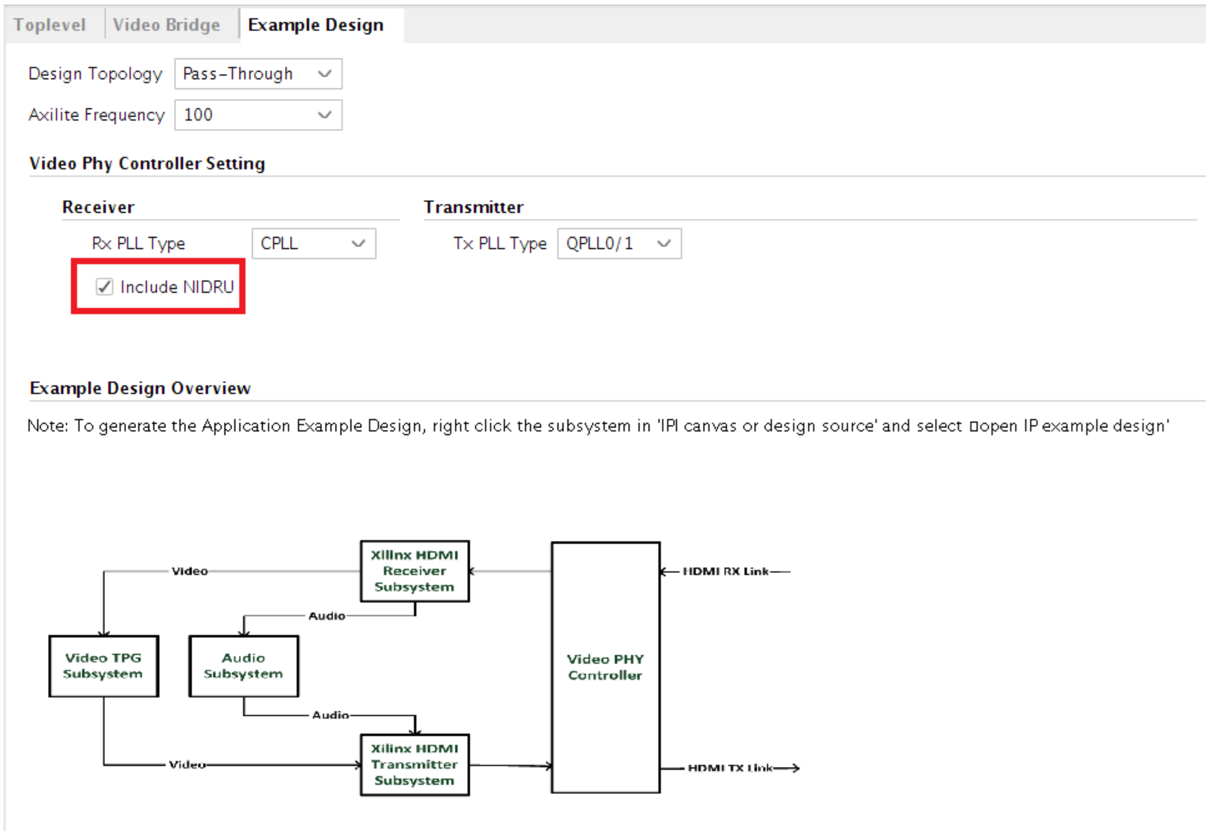
The parameter on the Video Bridge tab is as follows:

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

Example Design Tab

The Example Design tab is shown in the following figure.

Figure 34: **Example Design Tab (HDCP Not Included)**



Design Topology Pass-Through

Axilite Frequency 100

Rx PLL Type CPLL

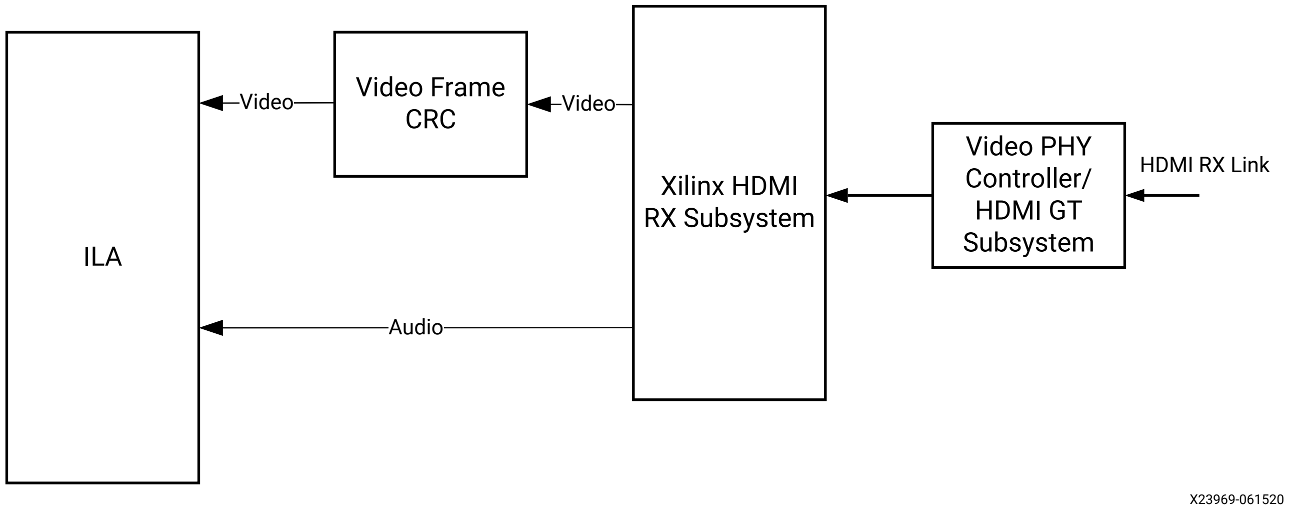
☑ Include NIDRU

Tx PLL Type QPLL0/1

Note: To generate the Application Example Design, right click the subsystem in 'IPI canvas or design source' and select open IP example design'

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

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



- **Axilite Frequency:** Allows you to choose the AXI4-Lite CPU clock. In this release, the following options have been verified.
  - 7 series: 50 MHz, 100 MHz, 150 MHz
  - UltraScale/UltraScale+ Devices: 50 MHz, 100 MHz, 150 MHz, 200 MHz
  - Versal ACAPs: 100 MHz
- **Video Phy Controller Setting:** Allows the configuration of the Transmitter PLL type and Receiver PLL Type to the Video PHY Controller/HDMI GT Subsystem prior generating the example design. It also allows user to selectively opt-out the NI-DRU to optimize resource use if the video resolution they plan to support does not require NI-DRU. See the *Video PHY Controller LogiCORE IP Product Guide* (PG230) or the *HDMI GT Controller LogiCORE IP Product Guide* (PG334) for details about NI-DRU requirements for Versal ACAPs.
- **Example Design Overview:** A system block diagram to show the overview of the example design to be generated.

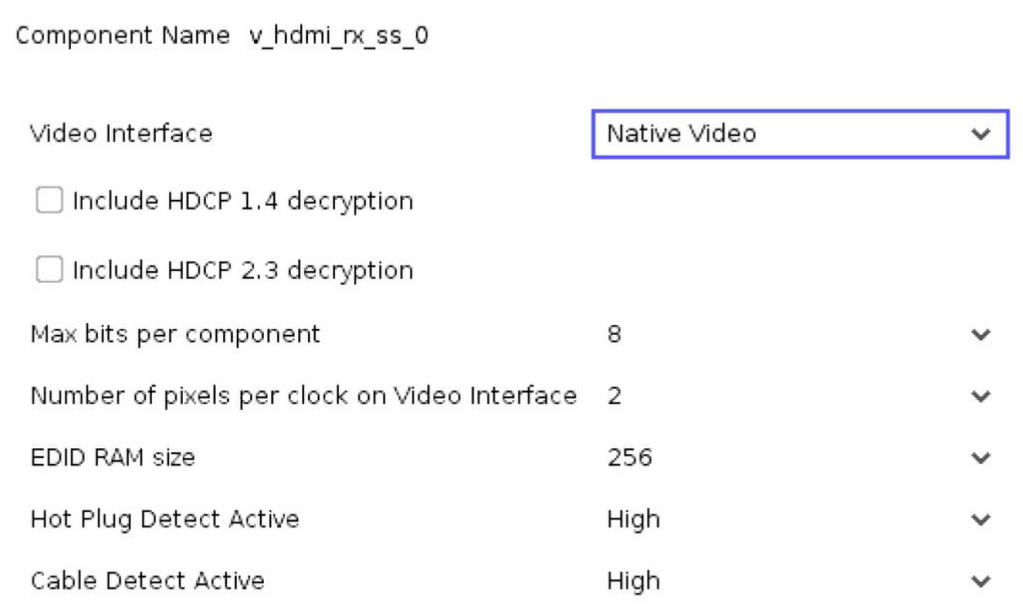
IMPORTANT! *When the example design targets the VCU118 board and Design Topology is set to Pass-Through, the Include NIDRU option under the Video PHY Controller setting is grayed out and unchecked by default.*

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

Native Video/Native Video (Vectored DE) Interface Option

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

Figure 35: **Native Video Interface Option**



Component Name v_hdmi_rx_ss_0

Video Interface: Native Video

☐ Include HDCP 1.4 decryption

☐ Include HDCP 2.3 decryption

Max bits per component	8
Number of pixels per clock on Video Interface	2
EDID RAM size	256
Hot Plug Detect Active	High
Cable Detect Active	High

- **Include HDCP 1.4 Decryption:** This option enables HDCP 1.4 decryption.
- **Include HDCP 2.3 Decryption:** This option enables HDCP 2.3 decryption.

***Note*:** HDCP 1.4 and 2.3 Decryption options are only configurable if you have a HDCP license, else it is disabled and cannot be selected from the GUI.

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

User Parameters

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

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

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Toplevel		
Video Interface	C_VID_INTERFACE	AXI4-Stream
AXI4-Stream	0	
Native Video	1	
Native Video (Vectored DE)	2	
Include HDCP 1.4 Decryption	C_INCLUDE_HDCP_1_4	Exclude
Exclude (Untick)	FALSE	
Include (Tick)	TRUE	
Include HDCP 2.3 Decryption	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	
Cable Detect Active	C_CD_INVERT	High
High	High	
Low	Low	
EDID RAM Size	C_EDID_RAM_SIZE	256
256	256	
512	512	
1024	1024	
4096	4096	
Video Bridge		
FIFO Depth	C_ADDR_WIDTH	1024
32	32	
1024	1024	
2048	2048	
4096	4096	
8192	8192	

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

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Example Design		
Design Topology Pass-Through Rx Only	C_EXDES_TOPOLOGY 0 2	0
AXI4-Lite Frequency[1] 50 100 150 200	C_EXDES_AXILITE_FREQ 50 100 150 200	100
TX PLL Type CPLL QPLL(GTXE2) QPLL01(GTHE3/4) LCPLL (GTYE5) RPLL (GTYE5)	C_EXDES_TX_PLL_SELECTION 0 3 6 7 8	0 (GTXE2) 6 (GTHE3/4) 7 (GTYE5)
RX PLL Type CPLL QPLL(GTXE2L) QPLL01(GTHE3/4) LCPLL (GTYE5) RPLL (GTYE5)	C_EXDES_RX_PLL_SELECTION 0 3 6 7 8	3 (GTXE2) 0 (GTHE3/4) 8 (GTYE5)
Include NIDRU Exclude (Untick) Include (Tick)	C_EXDES_NIDRU false true	true

Notes:

1. Versal ACAPs support only 100 MHz.

Output Generation

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

Constraining the Subsystem

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

Required Constraints

Clock frequency constraints are required for the `s_axi_cpu_aclk`, `s_axis_video_aclk`, `s_axis_audio_aclk`, `link_clk`, and `video_clk`.

```
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/HDMI GT Subsystem modules, `link_clk` and `video_clk` are generated from the Video PHY Controller/HDMI GT Subsystem. Therefore, the clock constraints are set to the Video PHY Controller/HDMI GT Subsystem constraints instead of these generated clocks. See Clocking in the *Video PHY Controller LogiCORE IP Product Guide* (PG230)/*HDMI GT Controller LogiCORE IP Product Guide* (PG334) for more information.

The `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)/*HDMI GT Controller LogiCORE IP Product Guide* (PG334).

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

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

Device Family	**PPC**	**2**				**4**			
	BPC	**8**	**10**	**12**	**16**	**8**	**10**	**12**	**16**
	Speed Grade								
Zynq-7000 SoC	–1	HDMI 1.4[1]				HDMI 1.4[1]			
	–2, -3	HDMI 2.0[2]				HDMI 2.0[2]			
Zynq UltraScale + MPSoC	–1	HDMI 2.0[2]				HDMI 2.0[2]			
	–2, -3								
Artix-7	–1	HDMI 1.4[1]				HDMI 1.4[1]			
	–2, -3	HDMI 1.4[1]				HDMI 1.4[1]			
Kintex-7	–1	HDMI 1.4[1]				HDMI 1.4[1]			
	–2, -3	HDMI 2.0[2]				HDMI 2.0[2]			

Table 21: **Device and Speed Grade Selections** *(cont'd)*

Device Family	PPC	2				4			
	BPC	8	10	12	16	8	10	12	16
	Speed Grade								
Kintex UltraScale	–1	HDMI 2.0[2]				HDMI 2.0[2]			
	–2, -3								
Virtex-7	–1	HDMI 1.4[1]				HDMI 2.0[2]	HDMI 1.4[1]		
	–2, -3	HDMI 2.0[2]				HDMI 2.0[2]			
Virtex UltraScale	–1	HDMI 2.0[2]				HDMI 2.0[2]			
	–2, -3								
Virtex UltraScale+	–1	HDMI 2.0[2]				HDMI 2.0[2]			
	–2, -3								
Versal ACAPs	–1	HDMI 2.0[2]				HDMI 2.0[2]			
	–2, -3								

Notes:

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

Clock Frequencies

See the Clocking section for more information.

Related Information

Clocking

Clock Management

This section is not applicable for this IP subsystem.

Clock Placement

This section is not applicable for this IP subsystem.

Banking

This section is not applicable for this IP subsystem.

Transceiver Placement

This section is not applicable for this IP subsystem.

I/O Standard and Placement

This section is not applicable for this IP subsystem.

Simulation

Simulation of the subsystem is not supported.

Synthesis and Implementation

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

Example Design

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

Summary

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

Table 22: **Example Design Support Summary**

Development Boards	Additional Hardware	Processor	Topology	Vitis Import Example
KC705/KCU105/ VCU118[1]	inrevium TB-FMCH-HDMI4K FMC daughter card	MicroBlaze™	Pass-through	Passthrough_Microblaze
			RX Only	RxOnly_Microblaze
ZC706		A9	Pass-through	Passthrough_A9
			RX Only	RxOnly_A9
ZCU102[2]/ZCU104/ ZCU106	-	A53	Pass-through	Passthrough_A53
			RX Only	RxOnly_A53
			Pass-Through	Repeater_A53[3]
			Pass-Through + I2S Audio (ZCU102 Only)	Passthrough_Audio_I2S_A53
		R5	Pass-through	Passthrough_R5
			RX Only	RxOnly_R5

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

Development Boards	Additional Hardware	Processor	Topology	Vitis Import Example
VCK190/VMK180		A72	Pass-through	Passthrough_A72

Notes:

1. For VCU118 board, no dedicated on-board GT reference clocks are available to support the HDMI Transmitter and NI-DRU on the HDMI Receiver simultaneously due to board design limitations. Therefore, if Pass-through topology is selected for the VCU118 board, NI-DRU is disabled.
2. The HDMI + I2S-PMOD Audio can be generated *only* using the configurations in the following table.
3. A dedicated repeater application is added to demonstrate repeater functionality. Note the following:

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

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

   The same application can also be used for other applications.

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

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

- HDMI 1.4/2.0 with HDCP 1.4/2.3 Receiver Subsystem
- HDMI 1.4/2.0 with HDCP 1.4/2.3 Transmitter Subsystem (For Pass-through topology only)
- Video PHY Controller/HDMI GT Subsystem

The design features the receive-only and the pass-through operation modes for the HDMI solution. In the pass-through mode, an external HDMI source is used to send video data over the HDMI design. In the receive-only mode, an external HDMI source is used to send video data, and the HDMI Example design receives and detect the video. You can check the video information from the UART menu. However, in receive-only mode, there won't be video available for visual check. The example design demonstrates the use of the High-bandwidth Digital Content Protection System (HDCP) Revision 1.4/2.3 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 2.3 is used to encrypt content at Ultra High Definition (UHD) while HDCP 1.4 is used as a legacy encryption scheme for lower resolutions.

Hardware

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

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

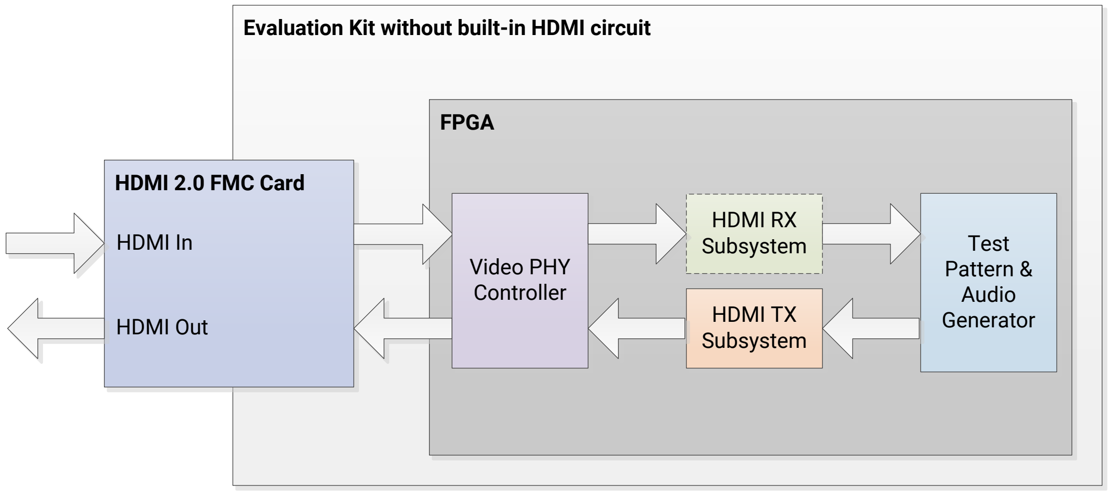
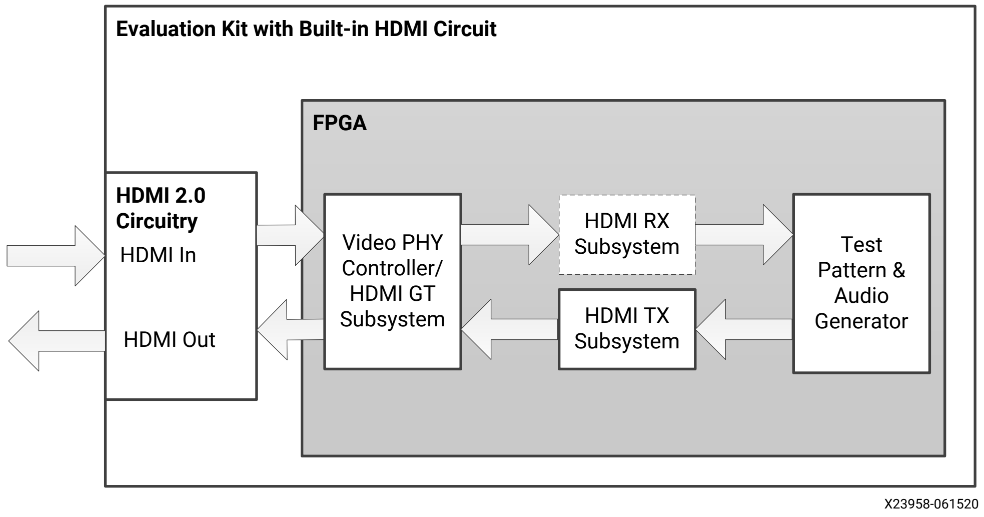


Figure 37: **ZCU102/ZCU104/ZCU106/VCK190/VMK180 HDMI Example Design Block Diagram**


Evaluation Kit with Built-in HDMI Circuit
FPGA
HDMI 2.0 Circuitry
HDMI In
HDMI Out
Video PHY Controller/ HDMI GT Subsystem
HDMI RX Subsystem
HDMI TX Subsystem
Test Pattern & Audio Generator
X23958-061520

IMPORTANT! *When an unpowered HDMI source is connected to the HDMI receiver in a pass-through system, the HDMI example design UART can get flooded with a Starting Colorbar message because of a limitation of the ZCU102 board design.*

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

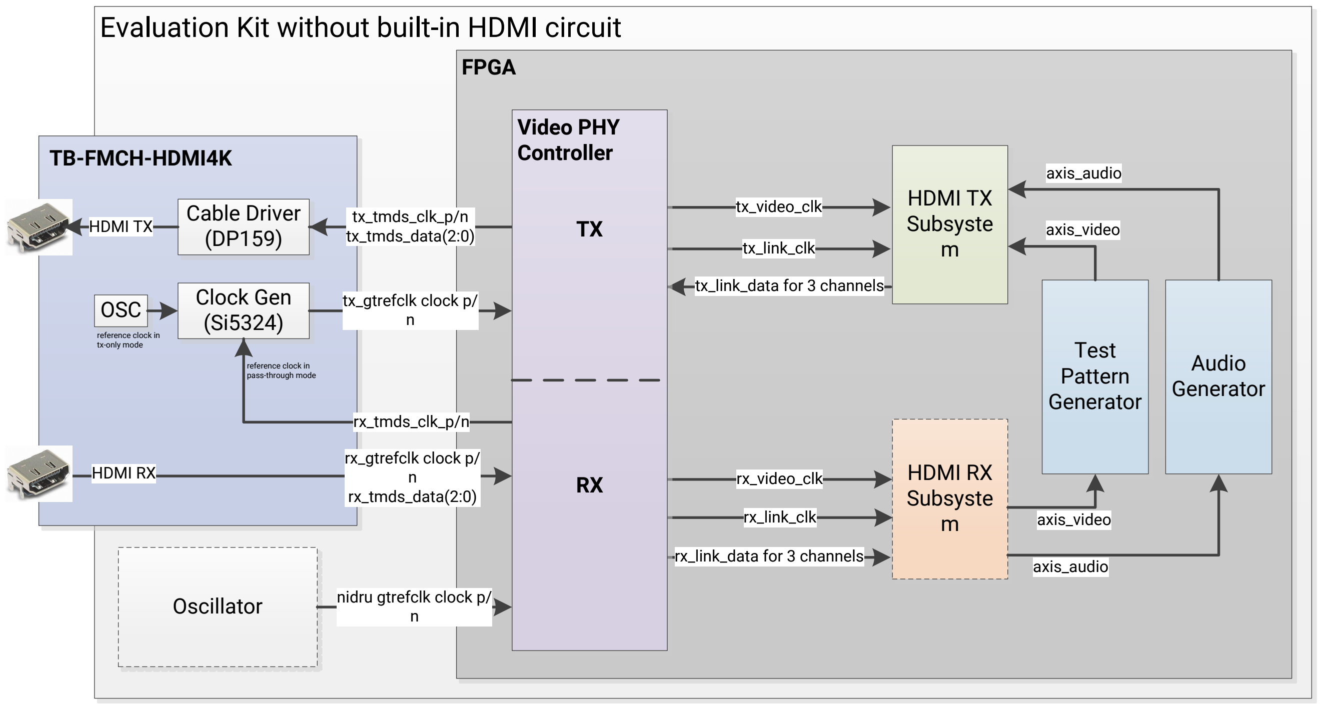
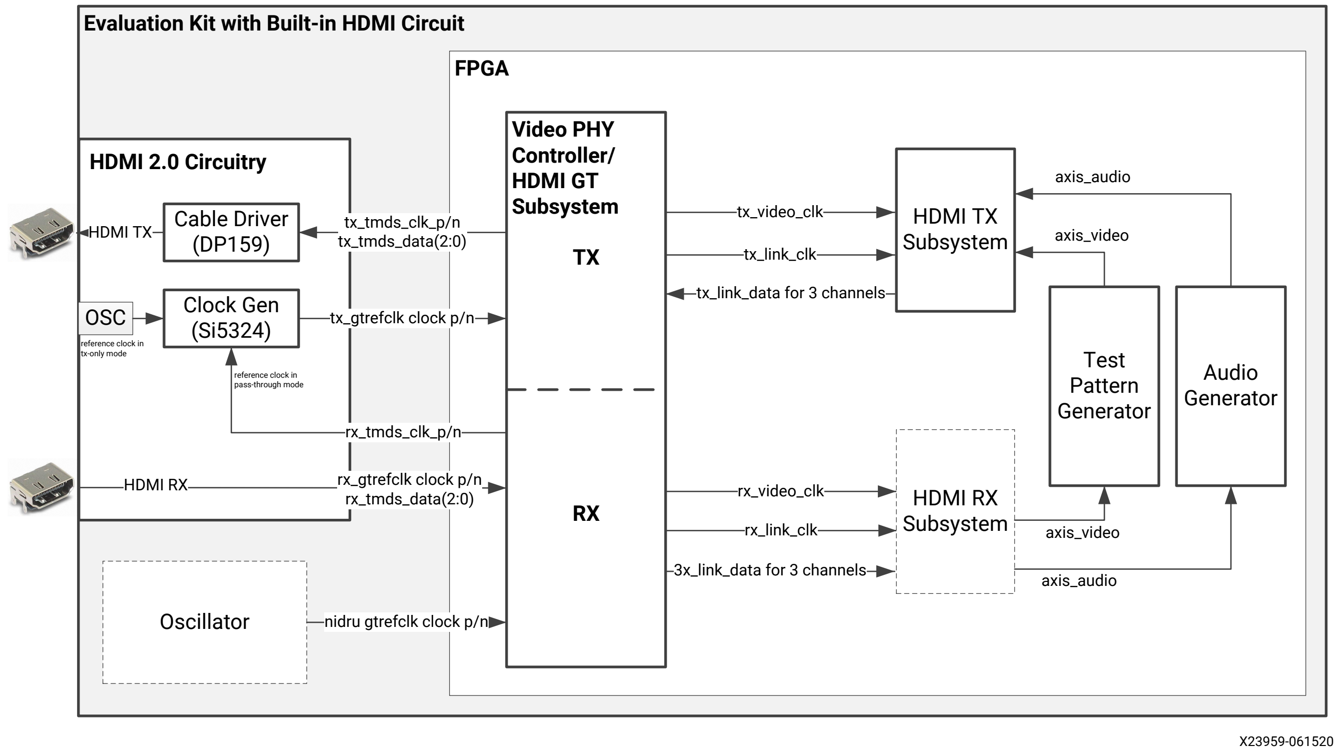


Figure 38: **KC705/KCU105/ZC706/VCU118 HDMI Reference Design Clock and Datapath Diagram**

Figure 39: **ZCU102/ZCU104/ZCU106/VCK190/VMK180 HDMI Reference Design Clock and Datapath Diagram**



***Note*:** The ZCU104 Evaluation Kit has IDT8T49N241 as the clock generator.

IMPORTANT! *The TI HDMI cable driver chip DP159 is used in all the example design, either on the evaluation board itself or on the inrevium TB-FMCH-HDMI4K FMC daughter card. In the HDMI example design software, a sample DP159 driver is provided as reference. In HDMI example design, DP159 mode depends on HDMI line rate. Automatic re-driver to re-timer crossover at 1.0 Gb/s (default) for HDIM1.4, Automatic re-timer for HDMI2.0 The settings have not been calibrated for various use cases. If you are using the DP159 in your product, you need to adjust the settings based on the circuit design. Some further fine tuning might still be needed depending on the compliance results.*

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

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

In the RX-only mode, an external HDMI source is used to send video data, and the HDMI Example design receives and detects the video. You can check the video information from the UART menu.

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

Figure 40: **HDMI Reference Design Block Diagram (MicroBlaze)**

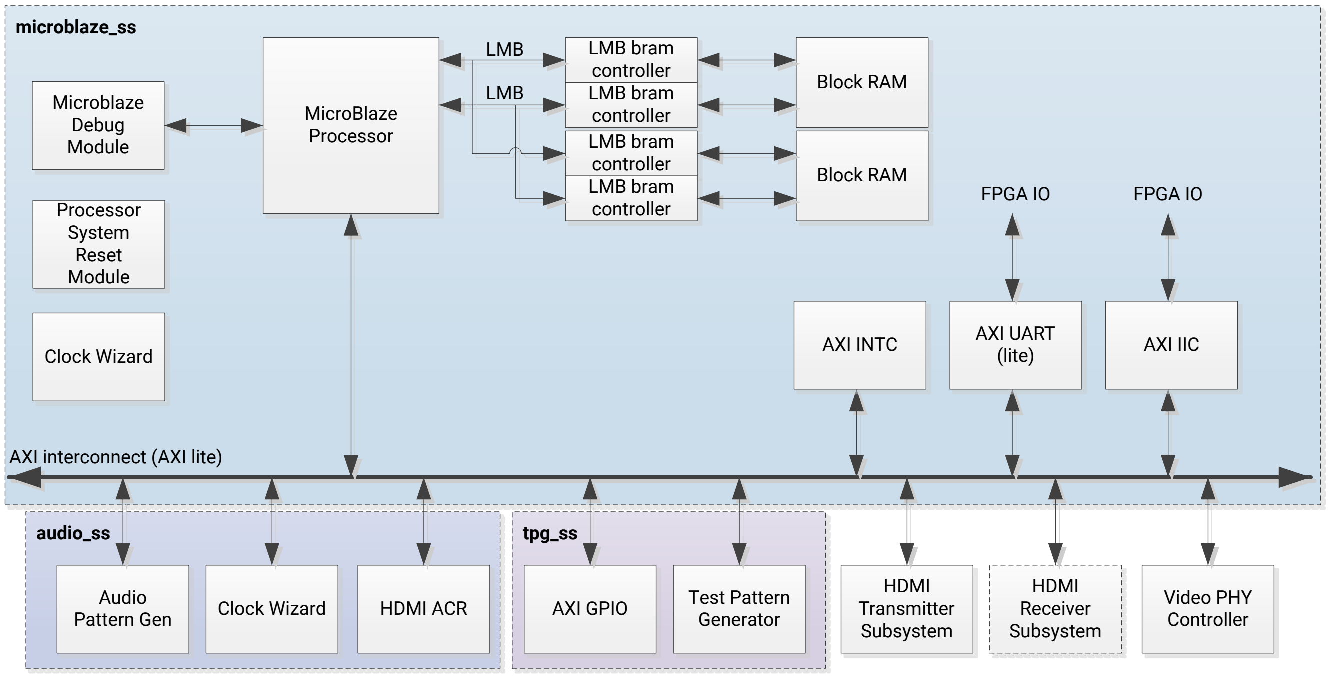
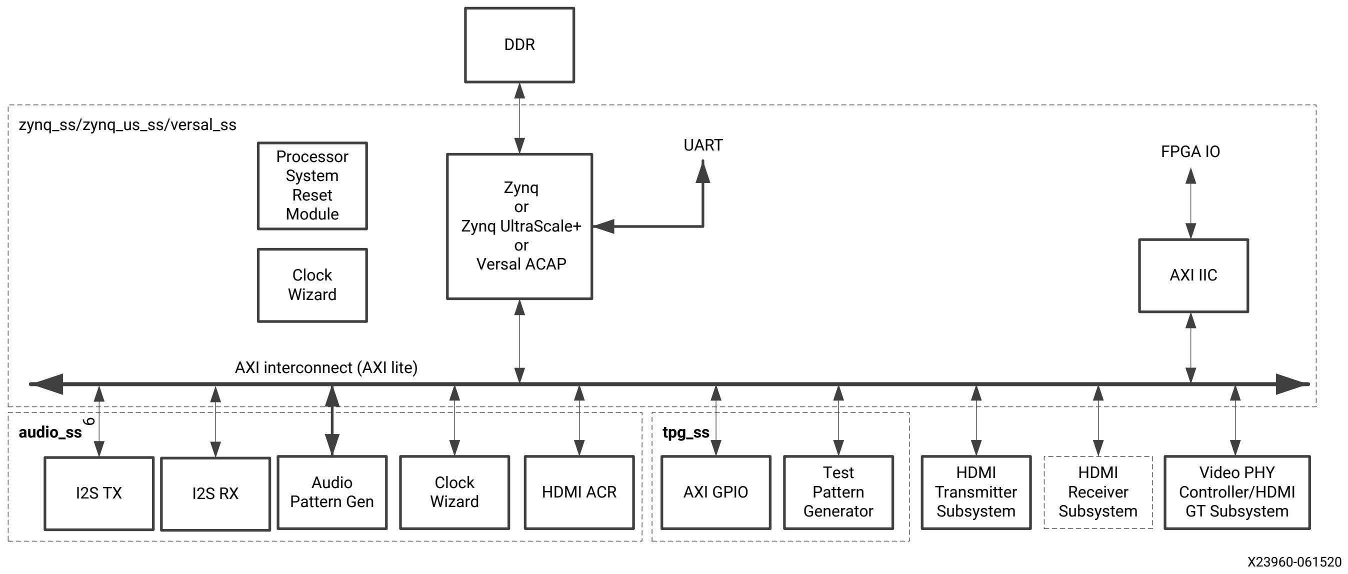


Figure 41: **HDMI Reference Design Block Diagram (Zynq or Zynq UltraScale+ MPSoC or Versal ACAP)**


DDR
zynq_ss/zynq_us_ss/versal_ss
Processor System Reset Module
Clock Wizard
Zynq or Zynq UltraScale+ or Versal ACAP
UART
FPGA IO
AXI IIC
AXI interconnect (AXI lite)
audio_ss
6
I2S TX
I2S RX
Audio Pattern Gen
Clock Wizard
HDMI ACR
tpg_ss
AXI GPIO
Test Pattern Generator
HDMI Transmitter Subsystem
HDMI Receiver Subsystem
Video PHY Controller/HDMI GT Subsystem
X23960-061520

Example Design Specifics

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

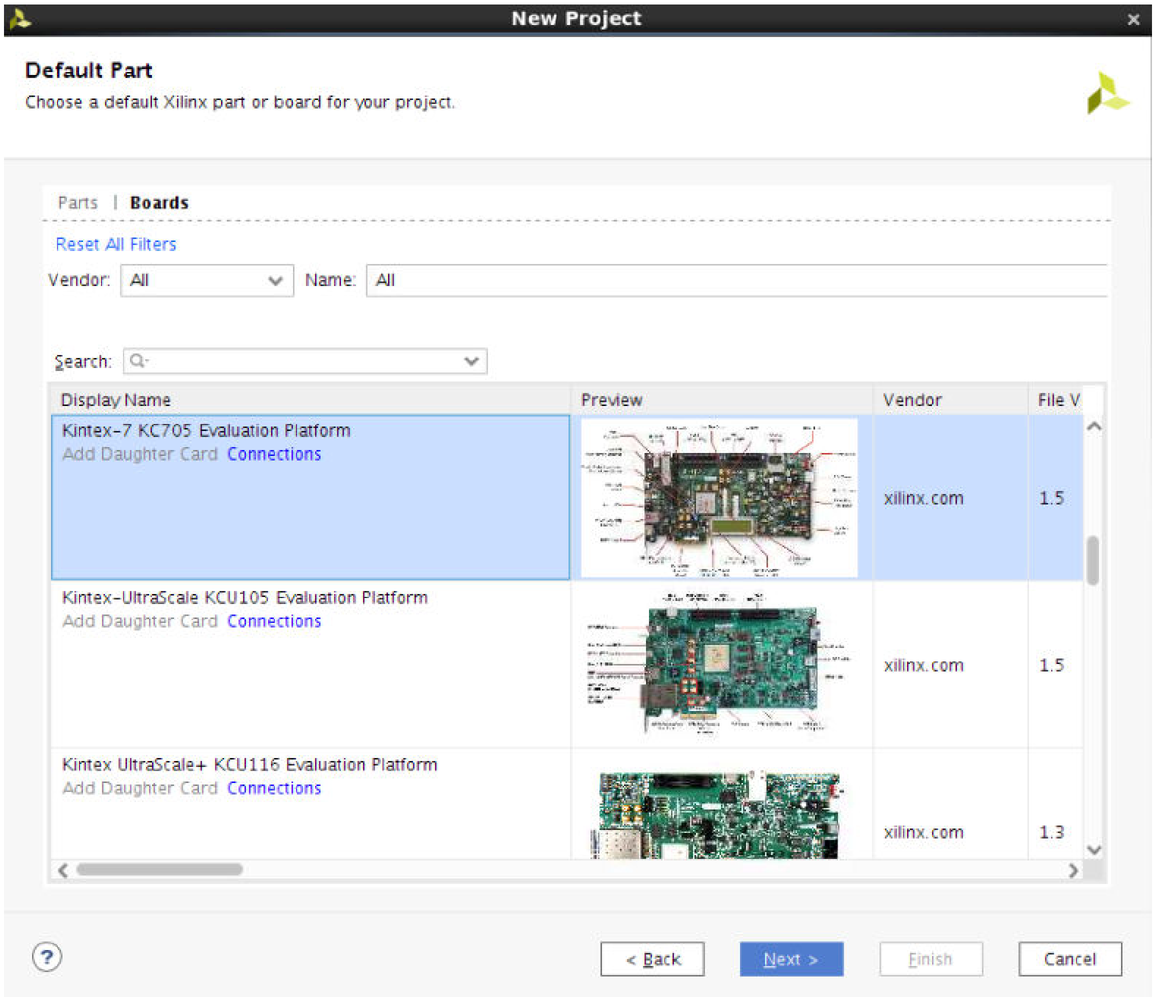
- MicroBlaze, Zynq, Zynq UltraScale+ MPSoC, or Versal ACAP
- MicroBlaze Debug Module (Only for MicroBlaze based processor subsystem)
- AXI Interconnect
- Local Memory Bus (LMB) (Only for MicroBlaze based processor subsystem)
- LMB BRAM Interface Controller (Only for MicroBlaze based processor subsystem)
- Block Memory Generator (Only for MicroBlaze based processor subsystem)
- Clocking Wizard
- Processor System Reset Module
- AXI UART Lite (Only for MicroBlaze based processor subsystem)
- AXI Interrupt Controller (INTC) (Only for MicroBlaze based processor subsystem)
- AXI IIC Bus Interface
- AXI GPIO
- Video Test Pattern Generator
- AXI4-Stream Register Slice
- Utility Buffer

- Utility Vector Logic
- AUD_PAT_GEN (Custom IP)
- HDMI_ACR_CTRL (Custom IP)
- HDCP_KEYMNGMT_BLK (Custom IP)

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

Running the Example Design

1. Open the Vivado Design Suite and create a new project.
2. In the pop-up window, press Next until you get to the page to select the Xilinx part or board for the project.

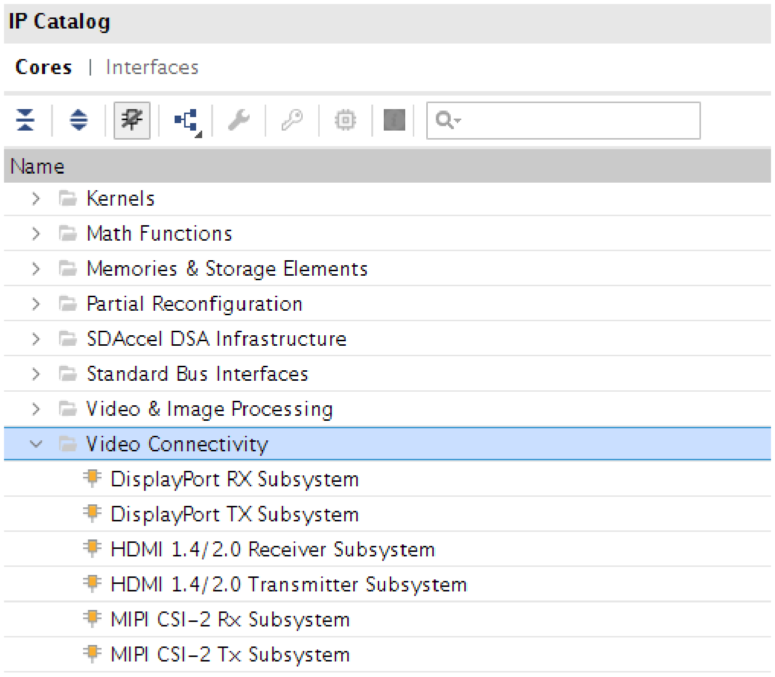


Display Name	Preview	Vendor	File V
Kintex-7 KC705 Evaluation Platform Add Daughter Card Connections		xilinx.com	1.5
Kintex-UltraScale KCU105 Evaluation Platform Add Daughter Card Connections		xilinx.com	1.5
Kintex UltraScale+ KCU116 Evaluation Platform Add Daughter Card Connections		xilinx.com	1.3

Note: Ensure that you have selected the Versal board in the project setup in the Vivado Design Suite. To do this, when you get to the Default Part screen, select the **Boards** tab, and select **Install/Update Boards**. This takes you to the Xhub Stores, where you can right-click on the VCK190/VMK180 board and select **Install** to install the VCK190/VMK180 board support files.

3. Select the target board, then click **Next→Finish**. (KC705, KCU105, ZC706, ZCU102, ZCU104, ZCU106, VCU118, VCK190, and VMK180 are supported.)

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

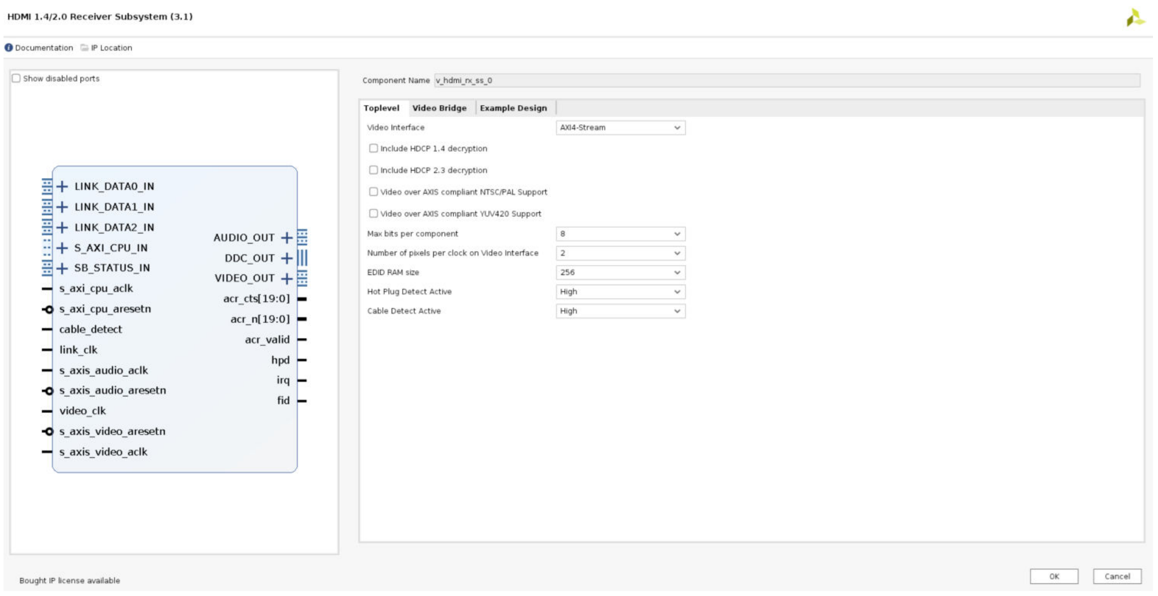


5. A Customize IP window opens. Configure the HDMI 1.4/2.0 RX Subsystem, then select OK.

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

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

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



HDMI 1.4/2.0 Receiver Subsystem (3.1)

Show disabled ports

Component Name v_hdmi_rx_ss_0

Video Interface AXI4-Stream

Include HDCP 1.4 decryption

Include HDCP 2.3 decryption

Video over AXIS compliant NTSC/PAL Support

Video over AXIS compliant YUV420 Support

Max bits per component 8

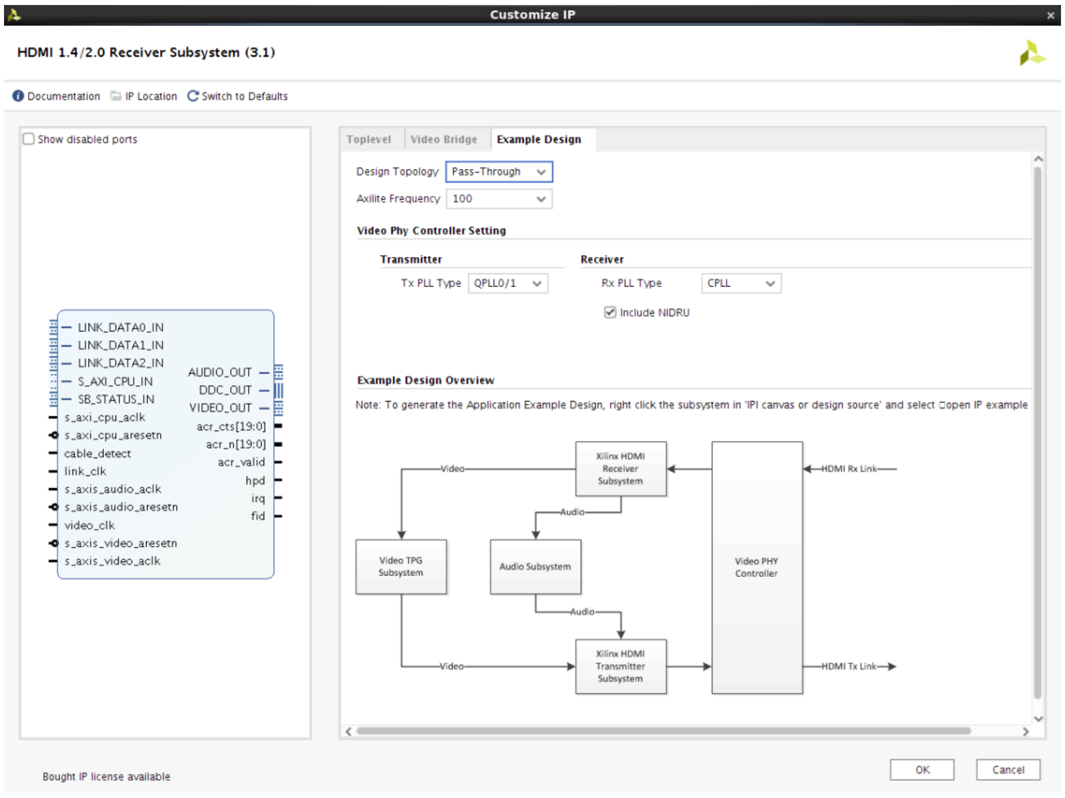
Number of pixels per clock on Video Interface 2

EDID RAM size 256

Hot Plug Detect Active High

Cable Detect Active High

Bought IP license available



HDMI 1.4/2.0 Receiver Subsystem (3.1)

Show disabled ports

Design Topology Pass-Through

Axilite Frequency 100

Video Phy Controller Setting

Transmitter

Tx PLL Type QPLL0/1

Receiver

Rx PLL Type CPLL

Include NIDRU ✓

Example Design Overview

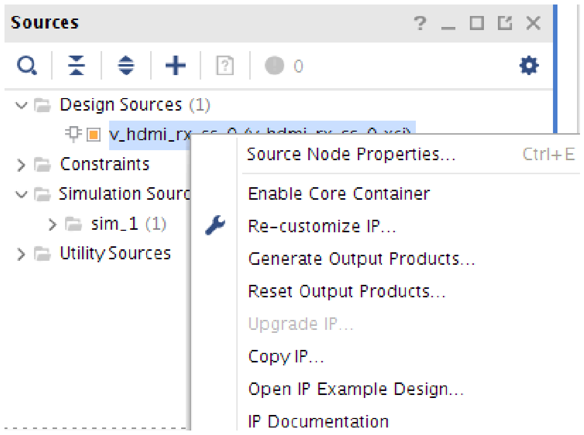
Note: To generate the Application Example Design, right click the subsystem in 'IPI canvas or design source' and select Open IP example

Bought IP license available

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

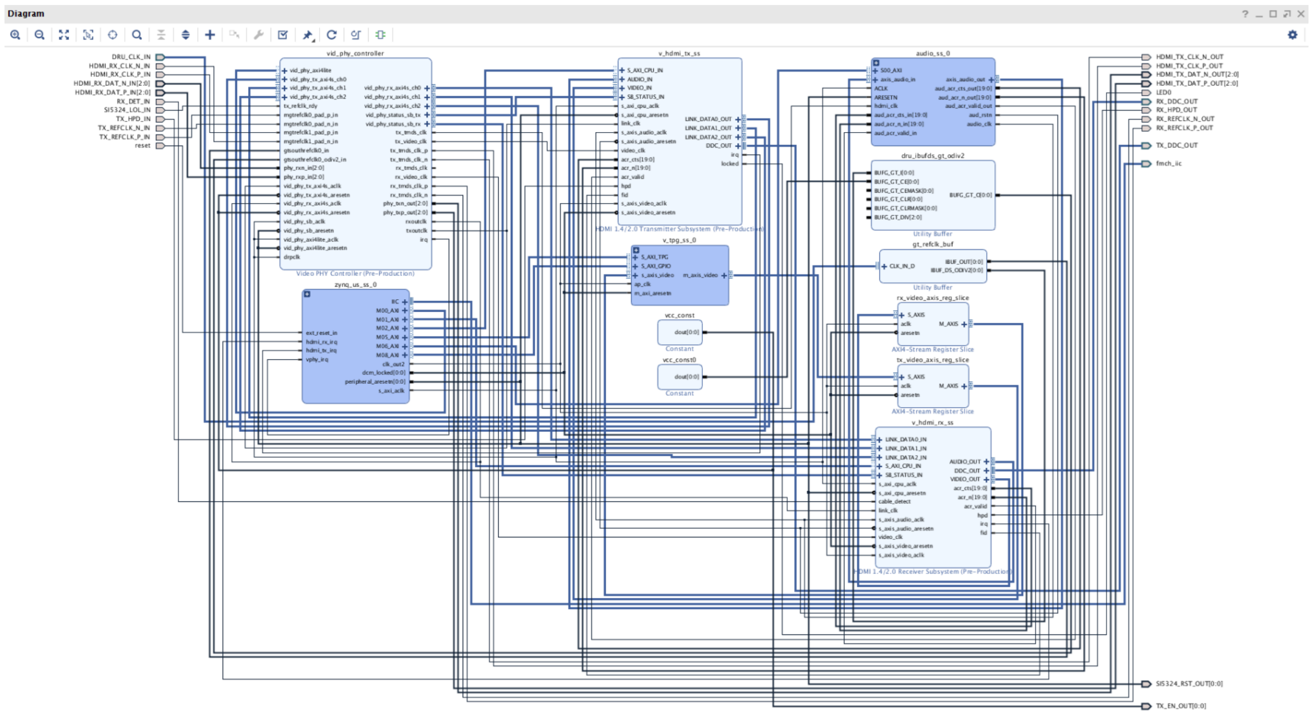
   ***Note*:** You can optionally select Skip if you only want to create an example design and leave the IP generation to a later stage.

7. The IP component with provided name is added to Design Sources. Right click on it and select **Open IP Example Design**.



8. Choose the target Example project directory, then select OK.

9. A new Vivado project launches, in which an HDMI Example Design is generated with Block Design to show the system structure. Select **Run Synthesis, Implementation, and Generate Bitstream** to build the design. An overall system IP integrator block diagram of the KC705-based example design is shown:

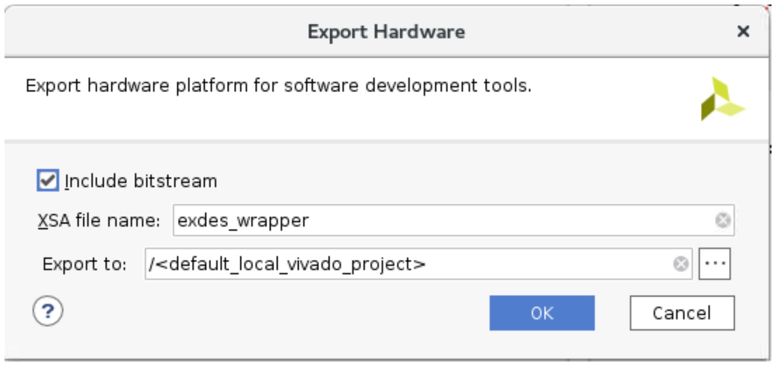


> **IMPORTANT!** *The following error message might appear after closing and re-opening the clock block design:* `WARNING: [BD 41-1731] Type mismatch between connected pins: /v_hdmi_rx/ m_axis_video_aresetn_out(undef) and /v_vid_in_axi4s/ aresetn(rst)` *but it does not affect any functionality.*

Export the Generated Example Design

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

1. To export the hardware design, in the generated HDMI example design Vivado project, select **File → Export → Export Hardware.**



Export Hardware

Export hardware platform for software development tools.

☑ Include bitstream

XSA file name: exdes_wrapper

Export to: /<default_local_vivado_project>

2. In the Export Hardware window, select OK. The hardware definition file is exported to a folder (usually the Vivado project root directory).
3. Create a directory for the Vitis workspace on the Vivado Tcl console.

```
mkdir <vitis_workspace>
```

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

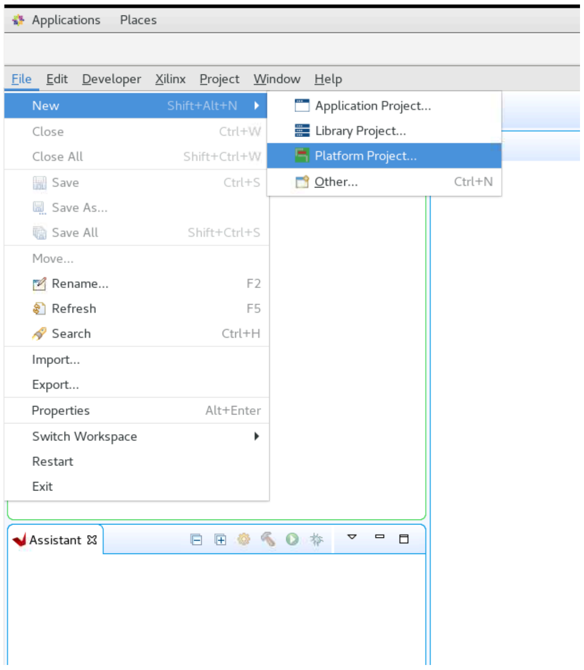
```
cd <vitis_workspace>
vitis &
```

Related Information

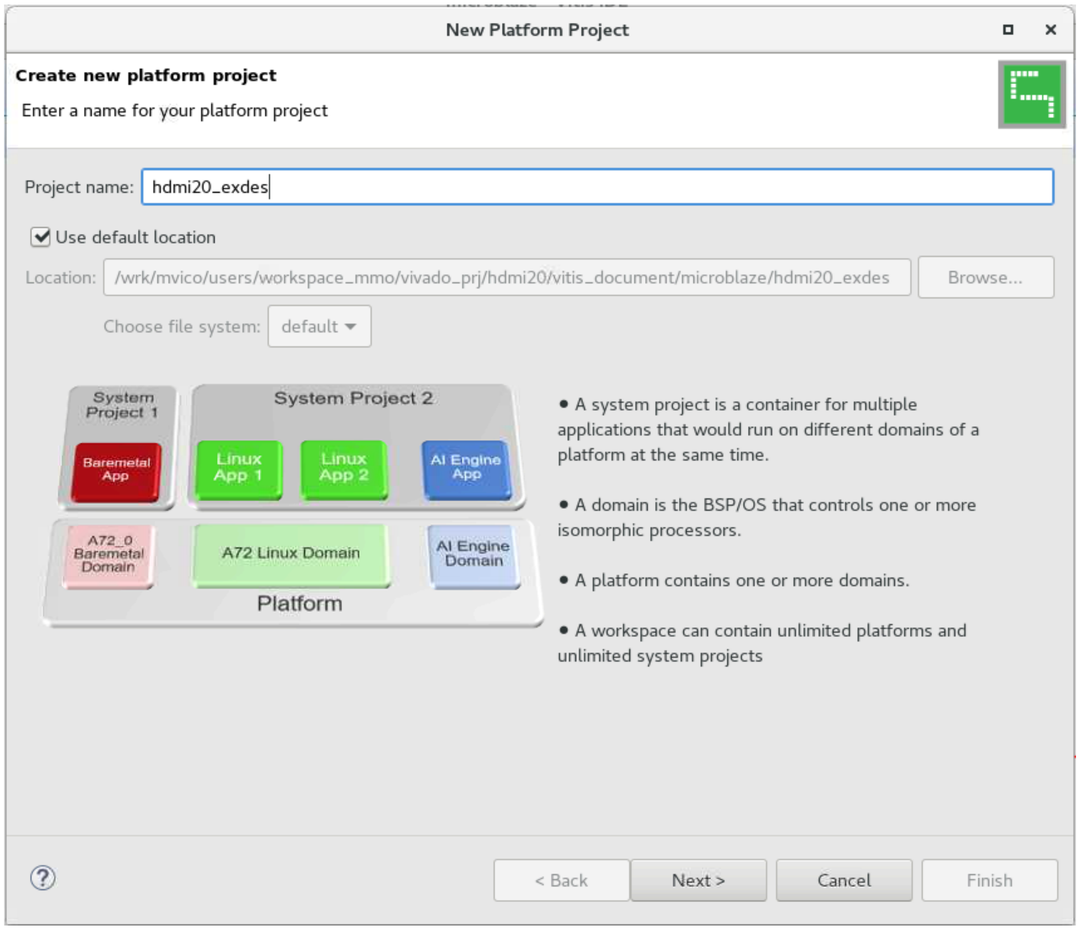
Summary

Build Software Application Using Vitis IDE

1. Open the Vitis IDE.
2. Select **File → New → Platform Project** to create a new platform project.



3. Set the desired Project Name and select **Next**.


New Platform Project

Create new platform project

Enter a name for your platform project

Project name: hdmi20_exdes

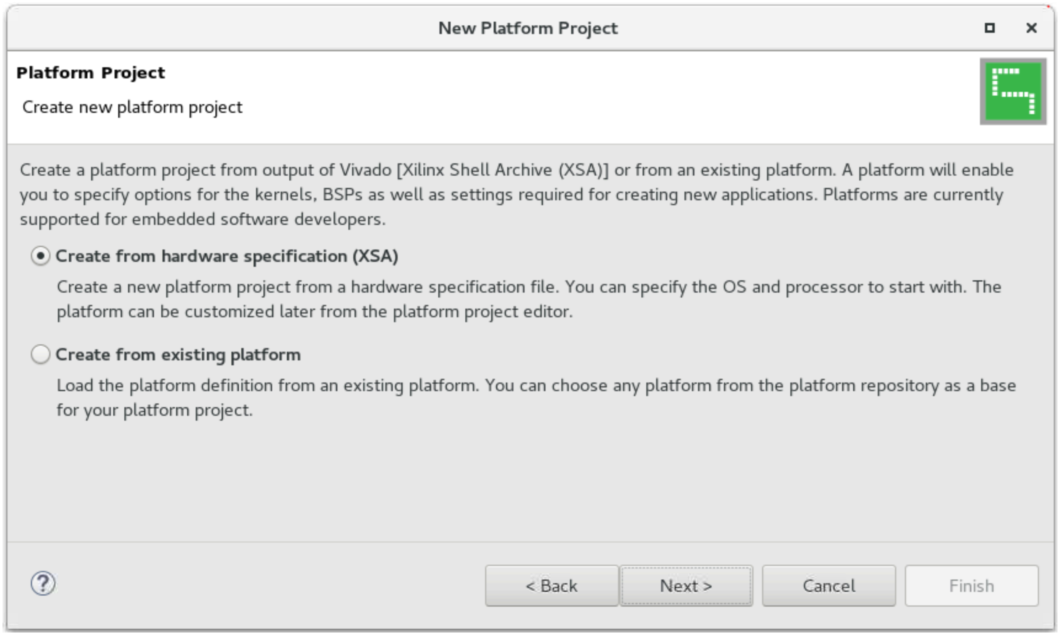
☑ Use default location

Location: /wrk/mvico/users/workspace_mmo/vivado_prj/hdmi20/vitis_document/microblaze/hdmi20_exdes

Choose file system: default

- A system project is a container for multiple applications that would run on different domains of a platform at the same time.
- A domain is the BSP/OS that controls one or more isomorphic processors.
- A platform contains one or more domains.
- A workspace can contain unlimited platforms and unlimited system projects

4. Select Create from hardware specification (XSA) and select **Next**.



New Platform Project

Platform Project

Create new platform project

Create a platform project from output of Vivado [Xilinx Shell Archive (XSA)] or from an existing platform. A platform will enable you to specify options for the kernels, BSPs as well as settings required for creating new applications. Platforms are currently supported for embedded software developers.

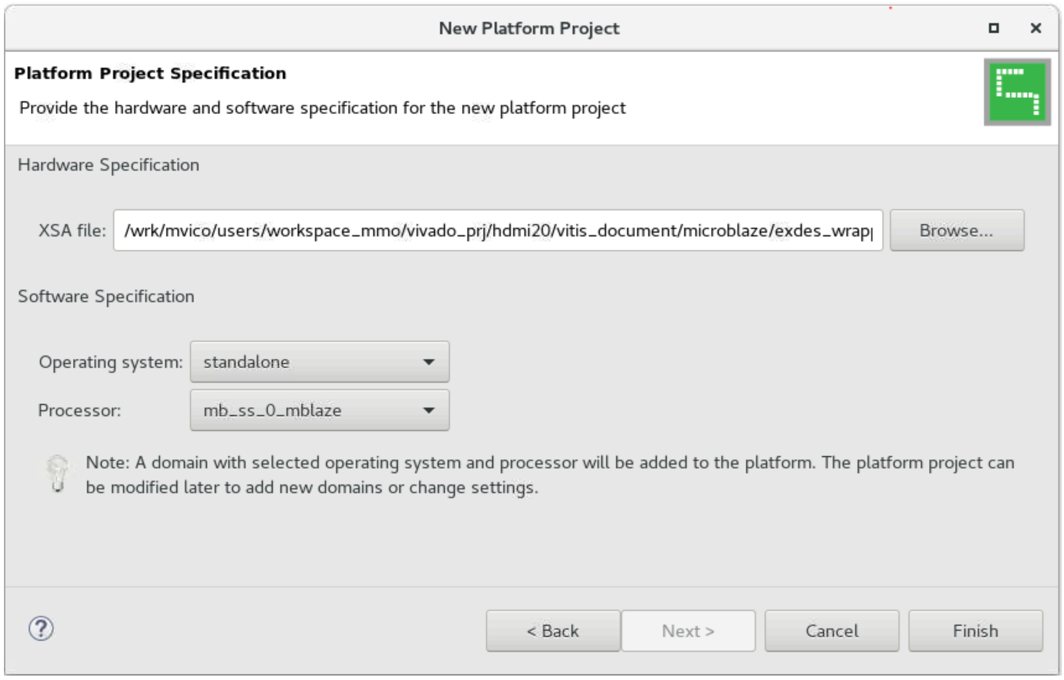
(●) **Create from hardware specification (XSA)**

Create a new platform project from a hardware specification file. You can specify the OS and processor to start with. The platform can be customized later from the platform project editor.

( ) **Create from existing platform**

Load the platform definition from an existing platform. You can choose any platform from the platform repository as a base for your platform project.

5. Select the generated XSA from the HDMI 2.0 Example Design, and select **Finish**.



New Platform Project

Platform Project Specification

Provide the hardware and software specification for the new platform project

Hardware Specification

XSA file: /wrk/mvico/users/workspace_mmo/vivado_prj/hdmi20/vitis_document/microblaze/exdes_wrap

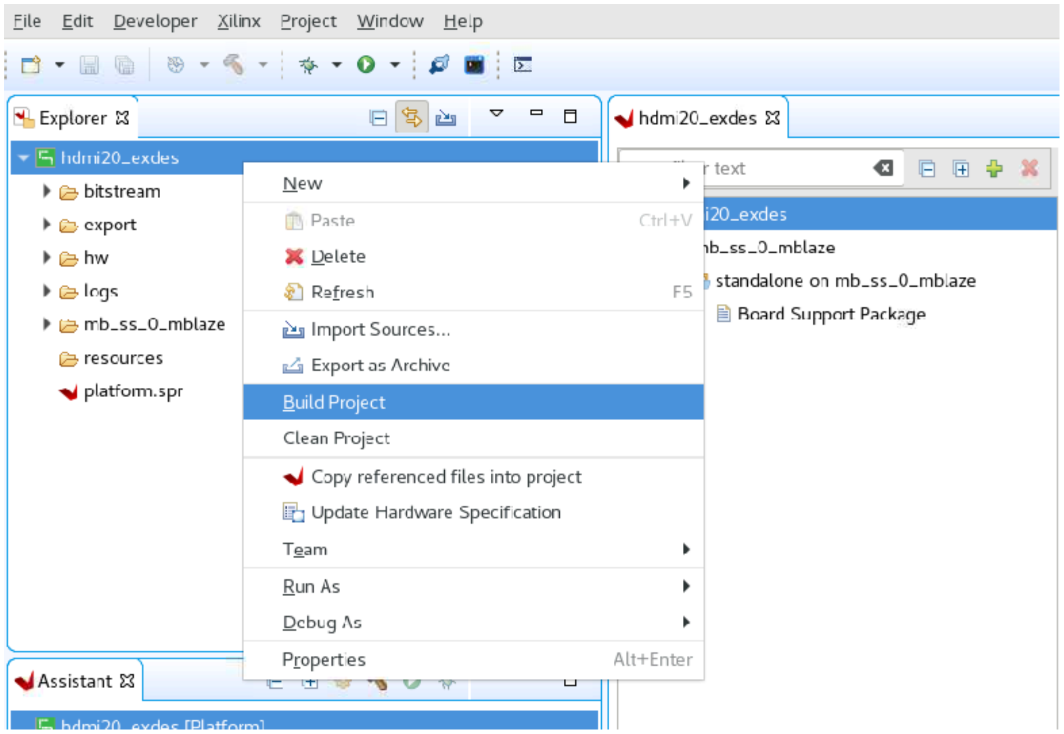
Software Specification

Operating system: standalone

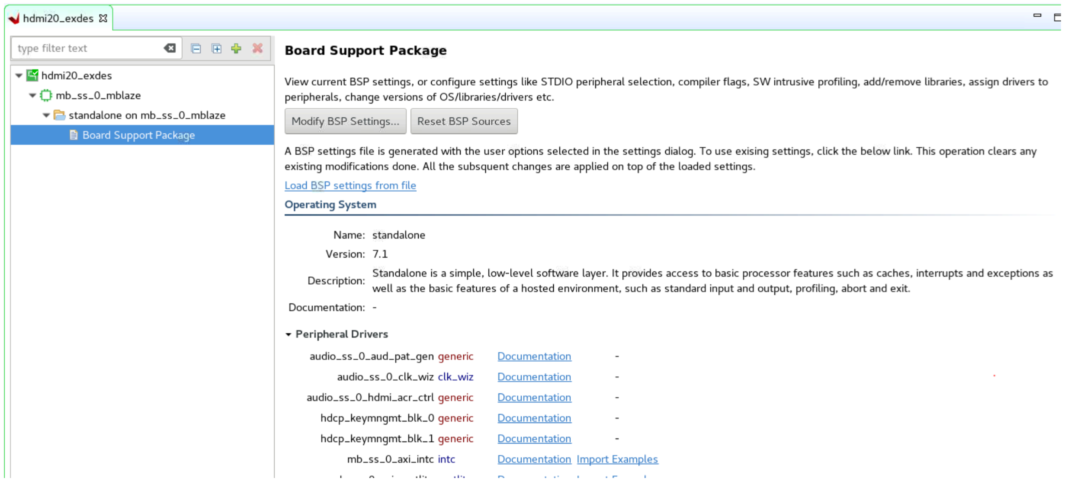
Processor: mb_ss_0_mblaze

Note: A domain with selected operating system and processor will be added to the platform. The platform project can be modified later to add new domains or change settings.

6. Build/Generate the platform - Right-click Platform and select **Build Project**.



7. At the platform.spr page, Select **Board Support Package** under the standalone, and expand the Peripheral Drivers.



Board Support Package

View current BSP settings, or configure settings like STDIO peripheral selection, compiler flags, SW intrusive profiling, add/remove libraries, assign drivers to peripherals, change versions of OS/libraries/drivers etc.

Modify BSP Settings... | Reset BSP Sources

A BSP settings file is generated with the user options selected in the settings dialog. To use exising settings, click the below link. This operation clears any existing modifications done. All the subsquent changes are applied on top of the loaded settings.

Load BSP settings from file

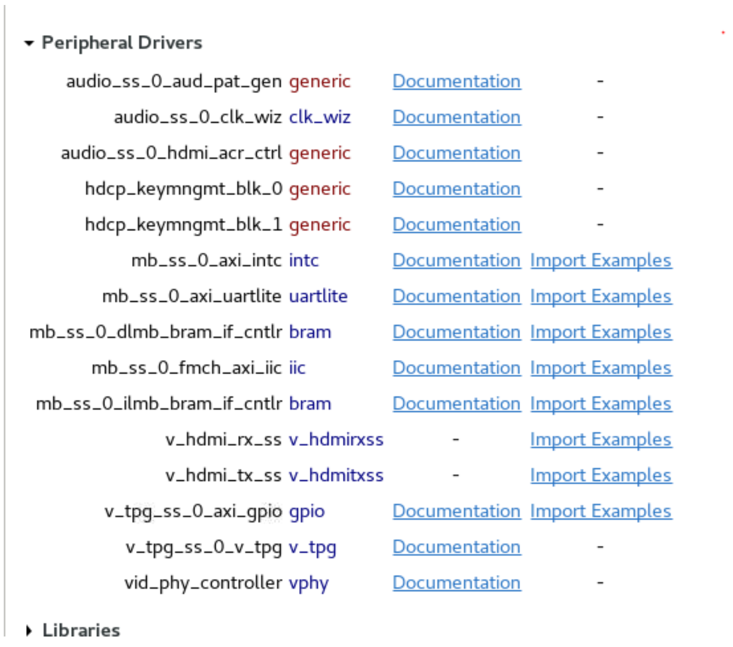
Operating System

Name:	standalone
Version:	7.1
Description:	Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.
Documentation:	-

Peripheral Drivers

audio_ss_0_aud_pat_gen	generic	Documentation	-
audio_ss_0_clk_wiz	clk_wiz	Documentation	-
audio_ss_0_hdmi_acr_ctrl	generic	Documentation	-
hdcp_keymngmt_blk_0	generic	Documentation	-
hdcp_keymngmt_blk_1	generic	Documentation	-
mb_ss_0_axi_intc	intc	Documentation	Import Examples

8. Find the v_hdmi_rx_ss or v_hdmi_tx_ss, and select **Import Examples**.

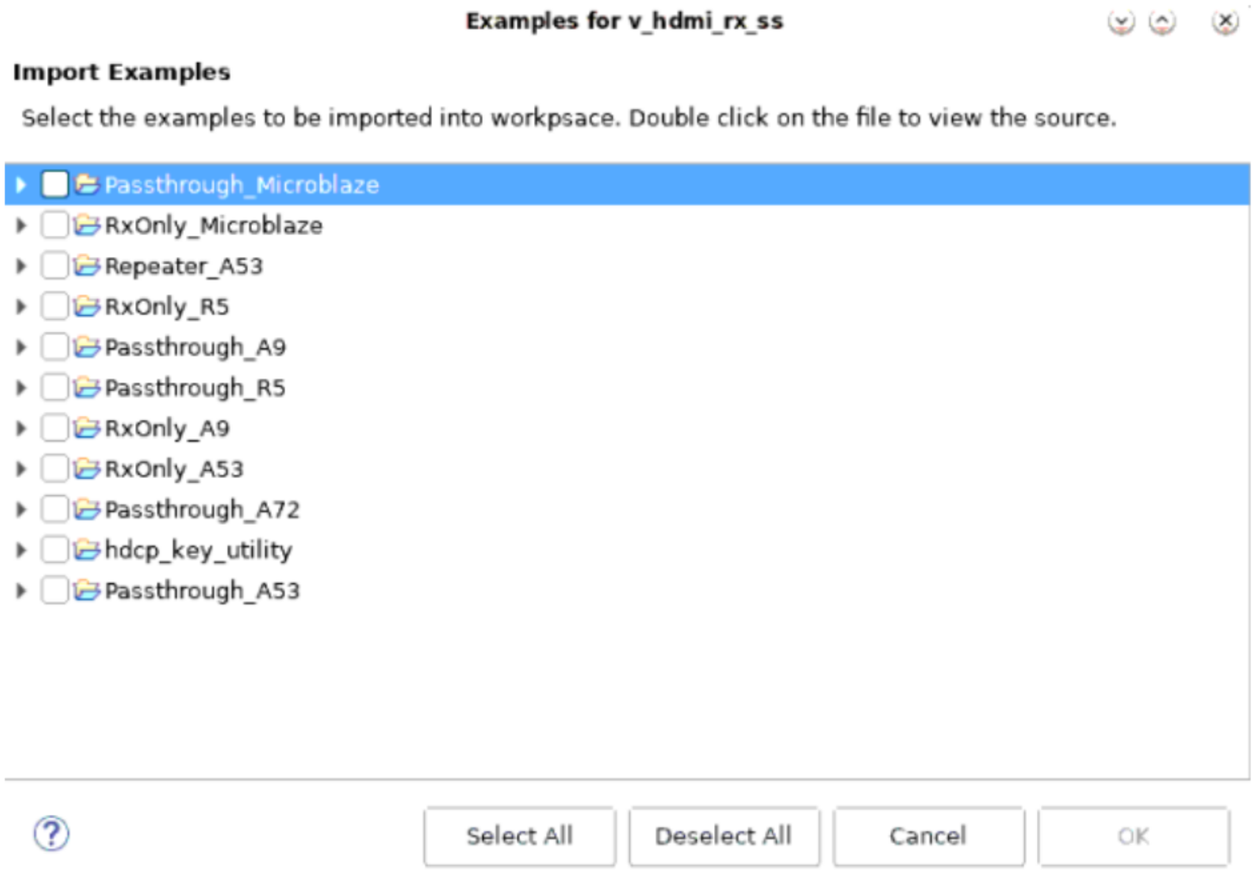


Peripheral Drivers

audio_ss_0_aud_pat_gen	generic	Documentation	-
audio_ss_0_clk_wiz	clk_wiz	Documentation	-
audio_ss_0_hdmi_acr_ctrl	generic	Documentation	-
hdcp_keymngmt_blk_0	generic	Documentation	-
hdcp_keymngmt_blk_1	generic	Documentation	-
mb_ss_0_axi_intc	intc	Documentation	Import Examples
mb_ss_0_axi_uartlite	uartlite	Documentation	Import Examples
mb_ss_0_dlmb_bram_if_cntlr	bram	Documentation	Import Examples
mb_ss_0_fmch_axi_iic	iic	Documentation	Import Examples
mb_ss_0_ilmb_bram_if_cntlr	bram	Documentation	Import Examples
v_hdmi_rx_ss	v_hdmirxss	-	Import Examples
v_hdmi_tx_ss	v_hdmitxss	-	Import Examples
v_tpg_ss_0_axi_gpio	gpio	Documentation	Import Examples
v_tpg_ss_0_v_tpg	v_tpg	Documentation	-
vid_phy_controller	vphy	Documentation	-

Libraries

9. An Import Examples window opens with a list of applications. See the screenshot and select the respective application and click **OK**. See the Example Design Support Summary table in the Summary topic to select the respective application.



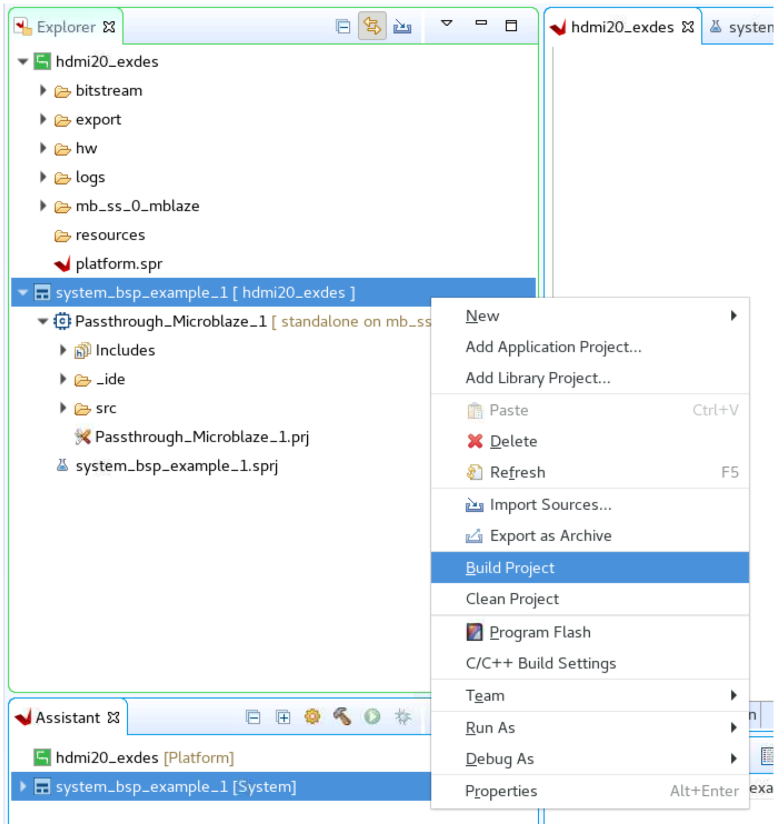
Examples for v_hdmi_rx_ss

Import Examples

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

- Passthrough_Microblaze
- RxOnly_Microblaze
- Repeater_A53
- RxOnly_R5
- Passthrough_A9
- Passthrough_R5
- RxOnly_A9
- RxOnly_A53
- Passthrough_A72
- hdcp_key_utility
- Passthrough_A53

10. Build the application.



The Example Application will be built and a `.elf` file is ready to use.

Related Information

Summary

HDCP Key Utility

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

To hdcp_key_utility application:

1. Import Example from the Vitis software platform and choose hdcp_key_utility.
2. Open `hdcp_key_utility.c` from the Vitis project.
3. The arrays Hdcp22Lc128, Hdcp22Key, Hdcp14Key1, and Hdcp14Key2 hold the HDCP keys and are empty. Fill these arrays with the acquired HDCP keys. The arrays are defined in big endian byte order.
4. Save the file and compile the design.
5. Run the design.

The terminal displays the following output:

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

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

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

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

Formatting HDCP Keys for HDCP 1.x

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

- The Hdcp14Key1 array

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

- The Hdcp14Key2 array.

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

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

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

1. Discard the 20 byte SHA-1.
2. Pad each key on the right with one byte of 0s (KSV is already padded).

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

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

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

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

Formatting HDCP Keys for HDCP 2.3

The `hdcp_key_utllity.c` has two (empty) HDCP 2.3 key arrays.

- The `hdcp22_lc128` array

  The global secret constant for the HDCP 2.3 TX. The size is 16 bytes and the license constant from the HDCP 2.3 TX certificate is placed into the array (position 4-19).

- The Hdcp22RxPrivateKey array

  This array holds the HDCP 2.3 RX certificate. The array has a size of 902 bytes. The contents are the header (4), license constant (36 bytes) and key set (862), so the total is 902 bytes.

  Note: If a pass-through example design is built which contains an HDMI RX component, you can use the same hdcp_key_utility application to program the HDCP 2.3 RX Private Key.

Running the Reference Design (MicroBlaze)

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

1. Launch the Xilinx System Debugger by selecting **Start → All Programs → Xilinx Design Tools → Vivado 2020.2 → Vivado 2020.2 Tcl Shell**.
2. In the Xilinx command shell window, change to the Example Design Project directory:

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

3. Invoke Xilinx System Debugger (xsdb).

   ```
   Vivado% xsdb
   ```

4. Establish connections to debug targets.

   ```
   xsdb% connect
   ```

5. Download the bitstream to the FPGA.

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

6. Set the target processor.

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

7. Download the software .elf to the FPGA.

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

8. Run the software.

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

9. Exit the XSDB command prompt.

   ```
   xsdb% exit
   ```

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

1. Launch the Xilinx System Debugger by selecting **Start→All Programs→Xilinx Design Tools →Vivado 2020.2→Vivado 2020.2 Tcl Shell**.
2. In the Xilinx command shell window, change to the Example Design Project directory:

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

3. Invoke Xilinx System Debugger (xsdb).

   ```
   Vivado% xsdb
   ```

4. Establish connections to debug targets.

   ```
   xsdb% connect
   ```

5. List all available JTAG targets.

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

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

6. Download the bitstream to the FPGA.

   ```
   xsdb% target -set 3 (PL)
   xsdb% fpga -file ./<IP instance name>_ex.runs/impl_1/exdes_wrapper.bit
   xsdd% after 2000
   ```

7. Set the target processor.

   ```
   xsdb% target -set 9 (Cortex-A53 #0)
   xsdb% rst -processor
   xsdb% dow ./<vitis_workspace>/<platform_name>/export/<platform_name>/sw/
   <platform_name>/boot/fsbl.elf
   xsdb% after 1000
   xsdb% con
   xsdb% after 1000
   xsdb% stop
   xsdb% after 1000
   ```

***Note*:** The target number for PSU, PL, and APU running Cortex-A53 might be different, to ensure that the associated target numbers are correct, re-run step 5: List all available JTAG targets.

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

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

9. Run the software.

```
xsdb% con
```

10. Exit the XSDB command prompt.

```
xsdb% exit
```

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

1. Launch the Xilinx System Debugger by selecting **Start → All Programs → Xilinx Design Tools → Vivado 2020.2 → Vivado 2020.2 Tcl Shell**.
2. In the Xilinx command shell window, change to the Example Design Project directory.

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

3. Invoke Xilinx System Debugger (xsdb).

```
Vivado% xsdb
```

4. Establish connections to debug targets.

```
xsdb% connect
```

5. List all available JTAG targets.

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

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

6. Download the bitstream to the FPGA.

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

7. Set the target processor.

```
xsdb% target -set 9 (Cortex-A53 #0)
xsdb% rst -processor
xsdb% dow ./<vitis_workspace>/<platform_name>/export/<platform_name>/sw/
<platform_name>/boot/fsbl.elf
xsdb% after 1000
xsdb% con
xsdb% after 1000
xsdb% stop
xsdb% after 1000
```

Note: The target number for PSU, PL, and RPU might be different, to ensure that the associated target numbers are correct, re-run step5: List all available JTAG targets.

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

```
xsdb% target -set 6 (Cortex-R5 #0)
xsdb% rst -processor
xsdb% dow ./<vitis_workspace>/<application_name>_1/Debug/
<application_name>_1.elf
```

9. Run the software.

```
xsdb% con
```

10. Exit the XSDB command prompt.

```
xsdb% exit
```

Running the Reference Design (A9 on Zynq)

1. Launch the Xilinx System Debugger by selecting **Start → All Programs → Xilinx Design Tools → Vivado 2020.2 → Vivado 2020.2 Tcl Shell**.

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

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

3. Invoke Xilinx System Debugger (xsdb).

```
Vivado% xsdb
```

4. Establish connections to debug targets.

```
xsdb% connect
```

5. List all available JTAG targets.

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

6. Download the bitstream to the FPGA.

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

7. Set the target processor.

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

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

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

9. Run the software.

```
xsdb% con
```

10. Exit the XSDB command prompt.

```
xsdb% exit
```

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

Running the Reference Design (A72 on Versal)

1. Launch the Xilinx System Debugger by selecting **Start → All Programs → Xilinx Design Tools → Vivado 2020.2 → Vivado 2020.2 Tcl Shell**.
2. In the Xilinx command shell window, change to the Example Design Project directory.

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

3. Invoke Xilinx System Debugger (xsdb).

   ```
   Vivado% xsdb
   ```

4. Establish connections to debug targets.

   ```
   xsdb% connect
   ```

5. List all available JTAG targets.

   ```
   xsdb% targets
   1 Versal vjtag40
       2 RPU (PS POR is active)
           3 Cortex-R5 #0 (PS POR is active)
           4 Cortex-R5 #1 (PS POR is active)
       5 APU (FPD domain isolation)
          6 Cortex-A72 #0 (FPD domain isolation)
          7 Cortex-A72 #1 (FPD domain isolation)
       8 PPU
          9 MicroBlaze PPU (Sleeping after reset)
       10 PSM
       11 PMC
       12 PL
   ```

6. Download the bitstream to the FPGA.

   ```
   xsdb% device program ./<IP instance name>_ex.runs/impl_1/
   exdes_wrapper.pdi
   xsdb% after 1000
   ```

7. Set the target processor.

   ```
   xsdb% targets -set -filter {name =~ "Cortex-A72 #0"}
   xsdb% rst -proc
   xsdb% after 1000
   ```

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

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

9. Run the software.

   ```
   xsdb% con
   ```

10. Exit the XSDB command prompt.

    ```
    xsdb% exit
    ```

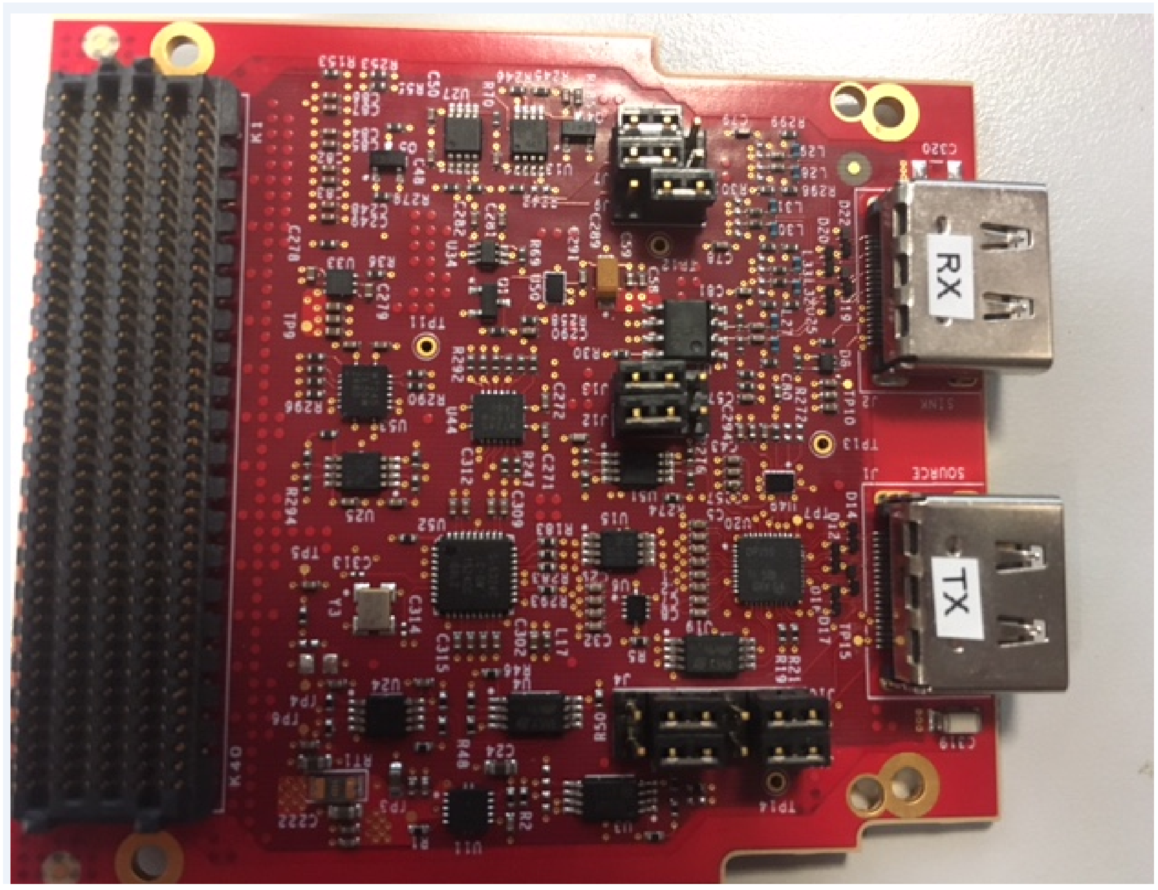
KCU105 Board FMCH VADJ Adjustment

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

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

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

1. Connect a USB cable between the USB UART connector of the KCU105 board and a PC running Windows.
2. Use the Windows Device Manager to determine which virtual COM port is assigned to the UART for the Zynq®-7000 SoC system controller and which is assigned to the UART for the UltraScale™ device. In the list of COM ports in the Device Manager window, the enhanced COM port associated with the CP210x, is the one connected to the KCU105 board system controller and the standard COM port is the one connected to the device UART.
3. Open a terminal window (115200, 8, N, 1) and set the COM port to the one communicating with the KCU105 board system controller.
4. After the UART terminal is connected, power cycle the KCU105 board to refresh the system controller menu in the UART terminal. Select this option in the system controller menu:
   a. Adjust FPGA Mezzanine Card (FMC) Settings.
5. In the next menu, select:
   a. Set FMC VADJ to 1.8V.



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

Verification, Compliance, and Interoperability

Compliance

HDCP compliance has been performed with the following DCP certified test equipment:

- SimplayLabs SL8800
  - HDCP 1.4: 1A, 1B, 2C
  - HDCP 2.3: 1A, 1B, 2C
- Quantum Data QD980
  - HDCP 2.2: 1A, 1B, 2C

PCB Recommendation

TMDS DATA PCB trace rules necessary to meet HDMI compliance requirements for the TMDS181 and SN68DP159 (only when TX is used) device are as follows:

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

- Max 10 ps inter-pair skew FPGA->retimer.
- Max 10 ps inter-pair skew retimer->connector.

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

- Max 1 ps intra-pair skew FPGA->retimer.
- Max 1 ps intra-pair skew retimer->connector.
- Target impedance to be 100Ω ±7% (Max ±10%).
  - A single excursion is permitted out to a max/min of 100Ω ±25% and of a duration less than 250 ps.

For HDMI subsystems in high density designs the default PCB recommendations for the decoupling capacitors might not be sufficient. This is especially true for pipelines that are run on the same clock. Make sure to validate your design requirements prior to creating a PCB. For more information see the *UltraScale Architecture PCB Design User Guide* (UG583) for your architecture.

Interoperability

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

Hardware Testing

The HDMI 1.4/2.0 RX Subsystem has been validated using

- Kintex®-7 FPGA Evaluation Kit (KC705)
- Kintex UltraScale FPGA Evaluation Kit (KCU105)
- Inrevium Artix®-7 FPGA ACDC A7 Evaluation Board
- Zynq-7000 SoC evaluation board (ZC706)
- Zynq® UltraScale+™ MPSoC ZCU102 Evaluation Kit
- Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit
- Zynq UltraScale+ MPSoC ZCU106 Evaluation Kit
- Virtex® UltraScale+™ FPGA VCU118 Evaluation Kit
- Versal™ AI Core Series VCK190 Evaluation Kit
- Versal Prime Series VMK180 Evaluation Kit

This release is tested for FRL with the following source devices:

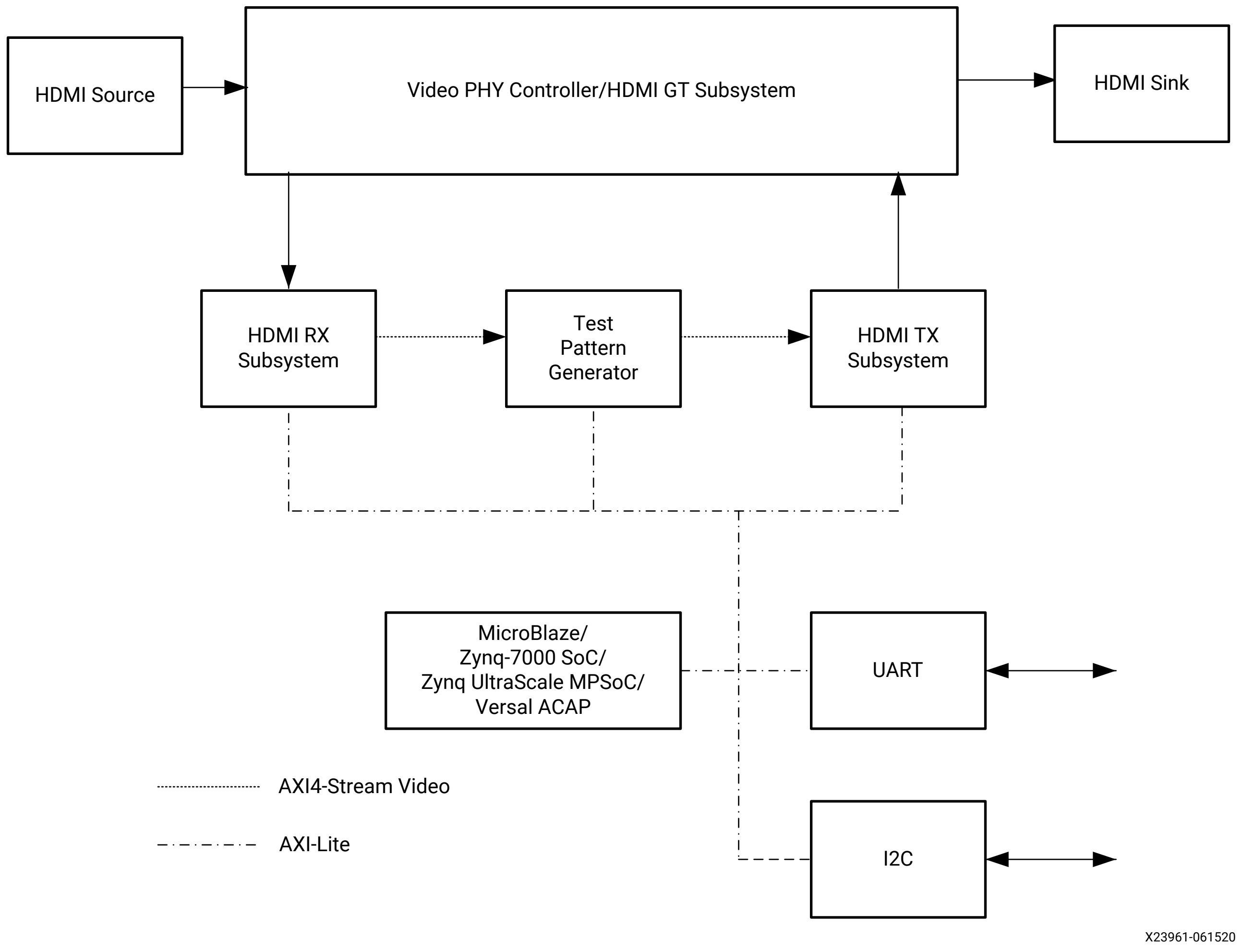
- Quantum Data 980B
- Quantum Data 780B
- Dell U2414Q
- Dell U2412M
- Dell U2713HM
- Acer S277HK
- Asus PQ321

- Sharp TV (LC-60LE740E)
- Philips TV (7800 series)
- Samsung UHDTV (UE40HU6900S)
- Murideo video analyzer / SIX-A
- DELL P2415Q
- Philips 288P6LJEB
- Apple TV (Gen 2/3/4)
- Android M8 media player
- Apple MacBook Pro
- Google Chromecast
- Open Hour media box
- Dell Latitude laptop (E7240)
- Intel HD Graphics 4000
- Nvidia GTX970 graphics card
- UGOOS media box
- LG 27mu67
- LG BP736
- Philips BDP2180K
- Sony BDP-S3500
- Sony BDP-S6500
- Samsung BD-J5900
- Murideo video generator / SIX-G
- Nvidia shield
- Roku 4
- Nvidia GTX980

Video Resolutions

The following figure shows the hardware setup for the AXI4-Stream video Interface. An HDMI source connects to the Video PHY Controller/HDMI GT Subsystem, 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/HDMI GT Subsystem. The Video PHY Controller/HDMI GT Subsystem then converts it back to HDMI Video and sends to the HDMI Sink.

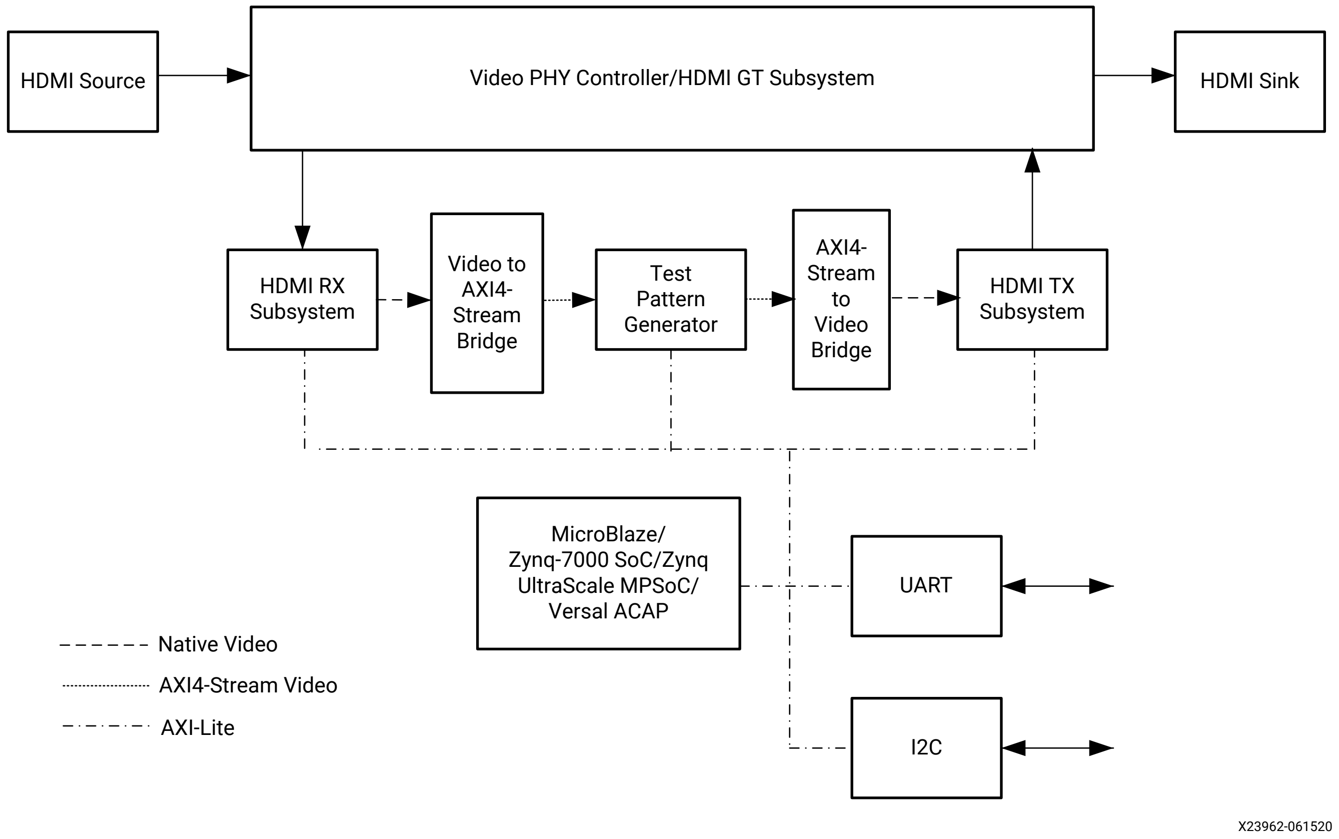
Figure 42: **Test Setup for AXI4-Stream Video Interface**



For Video PHY Controller/HDMI GT Subsystem settings and PLL selections, see the *Video PHY Controller LogiCORE IP Product Guide* (PG230)/*HDMI GT Controller LogiCORE IP Product Guide* (PG334).

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

Figure 43: **Test Setup for Native Video Interface**


HDMI Source
Video PHY Controller/HDMI GT Subsystem
HDMI Sink
HDMI RX Subsystem
Video to AXI4-Stream Bridge
Test Pattern Generator
AXI4-Stream to Video Bridge
HDMI TX Subsystem
MicroBlaze/ Zynq-7000 SoC/Zynq UltraScale MPSoC/ Versal ACAP
UART
I2C
Native Video
AXI4-Stream Video
AXI-Lite
X23962-061520

The following tables show the video resolutions that were tested as part of the release for different video formats.

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

Table 23: **Tested Video Resolutions for RGB 4:4:4 and YCbCr 4:4:4**

Resolution	Horizontal		Vertical		Frame Rate (Hz)
	Total	Active	Total	Active	
480i60[1]	858 (1716)	720 (1440)	525	480	60
576i50[1]	864 (1728)	720 (1440)	625	576	50
1080i50	2640	1920	1125	1080	50
1080i60	2200	1920	1125	1080	60
480p60	858	720	525	480	60

Table 23: **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	
576p50	864	720	625	576	50
720p50	1980	1280	750	720	50
720p60	1650	1280	750	720	60
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[4]	4400	3840	2250	2160	60
4096x2160p60[4]	4400	4096	2250	2160	60
vgap60	800	640	525	480	60
svgap60	1056	800	628	600	60
xgap60	1344	1024	806	768	60
sxgap60	1688	1280	1066	1024	60
wxgap60	1440	1280	790	768	60
wxga+p60	1792	1366	798	768	60
uxgap60	2160	1600	1250	1200	60
wuxgap60	2592	1920	1245	1200	60
wsxgap60	2240	1680	1089	1050	60

Notes:

1. Pixel repetition is enabled for 480i60 and 576i50. Therefore, the actual horizontal pixels are double the actual resolution.
2. Not all resolutions can be supported due to VPHY limitation. For details, see the *Video PHY Controller LogiCORE IP Product Guide* (PG230)/*HDMI GT Controller LogiCORE IP Product Guide* (PG334).
3. In this release, UXGA 60 Hz is supported in the HDMI 1.4/2.0 RX Subsystem for 8, 10, and 12 bits per component only.
4. For 4kp60 YUV444/RGB video, only 8-bits is supported as defined by HDMI 2.0 spec due to bandwidth limitation.

Tested Video Resolutions for YCbCr 4:2:2

Table 24: **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

Table 24: **Tested Video Resolutions for YCbCr 4:2:2 at 12 Bits/Component** *(cont'd)*

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

Tested Video Resolutions for YCbCr 4:2:0

Table 25: **Tested Video Resolutions for YCbCr 4:2:0 at 8, 10 Bits/Component in TMDS Mode**

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

Supported NTSC/PAL Resolutions

Table 26: **Supported NTSC/PAL Resolutions**

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

Table 26: **Supported NTSC/PAL Resolutions** *(cont'd)*

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

Appendix B

Upgrading

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

Port Changes for 2020.1

`AUDIO_IN_tid` port increased to 8 bits to Support 3D Audio (Up to 32 Channels)

Related Information

Audio Output Stream Interface

Port Changes for 2019.2

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

Related Information

Native Video (Vectored DE)

Port Changes

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

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

Parameter Changes

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

Table 27: **Vivado IDE Parameters Changes across Versions**

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

Notes:

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

Related Information

Customizing and Generating the Subsystem

Other Changes

N/A.

Migration Notes

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

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

  Choose High in the Example Design (according to board design).
- Hot Plug Detect Active has been added to HDMI 1.4/2.0 Receiver Subsystem GUI.

  Choose Low in Example Design (according to board design).
- Cable Detect Active has been added to HDMI 1.4/2.0 Receiver Subsystem GUI.

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

- HDCP 1.4/2.3 is enabled by default in Example Design application software.

  Removed UART option to Enable HDCP 1.4 or HDCP 2.3.

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

  You do not need to choose HDCP 1.4 or HDCP 2.3 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 2.3, the priority is given to HDCP 2.3.

- HDCP repeater feature has been added.

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

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

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

Appendix C

Debugging

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

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

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

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

Finding Help on Xilinx.com

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

Documentation

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

Answer Records

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

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

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

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

Master AR for HDMI 1.4/2.0 RX Subsystem

AR: 54546

Technical Support

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

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

To ask questions, navigate to the Xilinx Community Forums.

Debug Tools

There are many tools available to address HDMI 1.4/2.0 RX 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 to interact with the logic debug LogiCORE IP cores, including:

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

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

Reference Boards

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

- 7 series FPGA evaluation board
  - KC705
- UltraScale FPGA evaluation board
  - KCU105
- Zynq-7000 SoC evaluation board
  - ZC706
- UltraScale+ FPGA evaluation board
  - ZCU102
  - ZCU104
  - ZCU106
  - VCU118
- Versal™ AI Core / Prime Series evaluation board
  - VCK190
  - VMK180

Hardware Debug

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

General Checks

- Ensure that all the timing constraints and all other constraints were met during implementation.
- Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `locked` port.
- If your outputs go to 0, check your licensing.
  - User LEDs (KC705/KCU105/ZC706/ZCU102/ZCU104/ZCU106/VCU118/VCK190/VMK180)
  - LED0 - HDMI 1.4/2.0 RX Subsystem lock (when the HDMI Example Design is used)
  - See the Debugging Appendix in the *Video PHY Controller LogiCORE IP Product Guide* (PG230)/*HDMI GT Controller LogiCORE IP Product Guide* (PG334), and ensure there is no problem with clocking.

IMPORTANT! *Not all HDMI sources are fully compliant. Often, a source can send unexpected video or auxiliary packets. Therefore, if you observe issues with a particular source, test the same design on other sources to ensure there are no problems with the video information sent from the HDMI source.*

In-system Debug

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

1. Press **i** to check the system information.
2. Press **z** to check the event log.

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

Table 28: **System Flow Event Log**

At System Start-up	While Changing Video Stream from UART Menu
VPHY log	VPHY log
------	------
GT init start	CPLL lost lock
GT init done	RX frequency event
RX frequency event	RX frequency event
RX timer event	RX timer event
RX DRU disable	RX DRU disable
CPLL reconfig done	CPLL reconfig done
GT RX reconfig start	GT RX reconfig start
GT RX reconfig done	GT RX reconfig done
CPLL lock	CPLL lock
RX reset done	RX reset done
RX MMCM reconfig done	RX MMCM reconfig done
HDMI RX log	HDMI RX log
------	------
Initializing HDMI RX core....	RX Stream is Down
Initializing HDCP 1.4 core....	RX Stream is Down
Initializing AXI Timer core....	RX TMDS reference clock change
Initializing HDCP 2.2 core....	RX Stream Init
RX HDCP 1.4 Enabled	RX Stream Start
RX HDCP 2.2 Disabled	RX Stream is Up
Reset HDMI RX Subsystem....	
RX cable is connected....	
RX HDCP 1.4 Enabled	
RX HDCP 2.2 Disabled	
RX HDCP 1.4 Enabled	
RX HDCP 2.2 Disabled	
RX TMDS reference clock change	
RX Stream Init	
RX Stream Start	
RX Stream is Up	

Table 29: **System Flow Event Log (For Versal Pass-through Designs)**

At System Start-up	While Changing Video Stream from UART Menu
HDMIPHY log ------ GT init start GT init done TX frequency event TX timer event TX GPO Rising Edge Detected TX MMCM reconfig done GT TX reconfig done TX MMCM lock LCPLL lock TX reset done RX frequency event RX timer event RX DRU disable RX GPO Rising Edge Detected GT RX reconfig done RPLL lock RX reset done RPLL lost lock RPLL lock RX reset done RX MMCM reconfig done RX MMCM lock TX frequency event TX frequency event TX timer event TX GPO Rising Edge Detected TX MMCM reconfig done GT TX reconfig done LCPLL lost lock TX MMCM lock LCPLL lock TX reset done HDMI RX log ------ Initializing HDMI RX core.... Initializing HDCP 1.4 core.... Initializing AXI Timer core.... Initializing HDCP 2.2 core.... RX HDCP 1.4 Enabled RX HDCP 2.2 Disabled Reset HDMI RX Subsystem.... RX cable is connected.... RX HDCP 1.4 Enabled RX HDCP 2.2 Disabled RX HDCP 1.4 Disabled RX HDCP 2.2 Enabled RX TMDS reference clock change RX Stream Init RX Stream Init RX mode changed to HDMI RX Stream Start RX Stream is Up	HDMIPHY log ------ TX frequency event RX frequency event RX frequency event RX timer event RX DRU disable RX GPO Rising Edge Detected GT RX reconfig done RPLL lock RX reset done RX MMCM reconfig done RX MMCM lock RX frequency event RX frequency event RX timer event RX DRU disable RX GPO Rising Edge Detected GT RX reconfig done RPLL lock RX reset done RX MMCM reconfig done RX MMCM lock HDMI RX log ------ RX Stream is Down RX Stream is Down RX mode changed to DVI RX TMDS reference clock change RX Stream Init RX mode changed to HDMI RX Stream Start RX Stream is Up RX Stream is Down RX Stream is Down RX mode changed to DVI RX TMDS reference clock change RX Stream Init RX mode changed to HDMI RX Stream Start RX Stream is Up

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

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

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

```
#define XVIDC_EDID_VERBOSITY          0
```

where `XVIDC_EDID_VERBOSITY` is defined in the following table.

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

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

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

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

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

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

5. The default EDID in the HDMI example application does not support HBR. If you intend to bring up HBR feature in your design or Xilinx example design, ensure the HDMI RX EDID is updated to support HBR audio. Otherwise, some commercial HDMI sources cannot send out HBR audio. Some sources might even disable the HBR option from the menu.

   The EDID clone feature can be used in the example design to enable HBR in HDMI RX example design (pass-through topology). Perform the following steps to clone an EDID from an HBR audio receiver:

   a. Connect the HDMI TX to the HBR audio receiver

   b. Key in the HDCP Password (optional)

   c. Clone the EDID (In UART, press **e**, then press **2**)

   d. Connect the HBR Audio Source to the HDMI RX

TMDS Polarity Check

Sometimes when designing the board, it is possible that the board designer has swapped the TMDS polarity to make the routing easier. In this case, the Video PHY Controller/HDMI GT Subsystem must be configured accordingly. When the polarity is wrong, the system will have difficulty detecting an incoming video stream, even on those common resolutions (for example, 1080p60) and the detected timing can be far from the expected value. You should start looking at the data pins for the GT being routed to the FPGA to check whether they have been swapped.

An API is available in the Video PHY Controller/HDMI GT Subsystem driver to allow you to set the polarity. If you experience system behavior as described above, you can use this API to change the polarity to confirm whether it is polarity swapping that is causing the system issue.

```
u32 XVphy_SetPolarity(XVphy *InstancePtr,
                      u8 QuadId,
                      XVphy_ChannelId ChId,
                      XVphy_DirectionType Dir,
                      u8 Polarity);
```

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.

AXI4-Stream Interfaces

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

- If the transmit `<interface_name>_tready` is stuck Low following the `<interface_name>_tvalid` input being asserted, the subsystem cannot send data.
- If the receive `<interface_name>_tvalid` is stuck Low, the subsystem is not receiving data.

- Check that the `aclk` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed.
- Check subsystem configuration.

Application Software Development

Device Drivers

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

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

Architecture

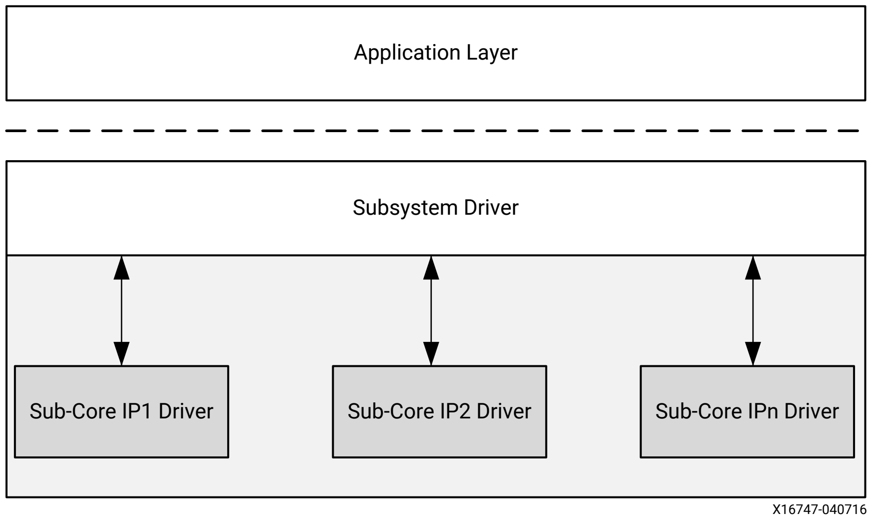
The subsystem driver provides an easy-to-use, well-defined API to help integrate the subsystem in an application without having to understand the underlying complexity of configuring each and every sub-core.

The subsystem driver consists of the following:

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

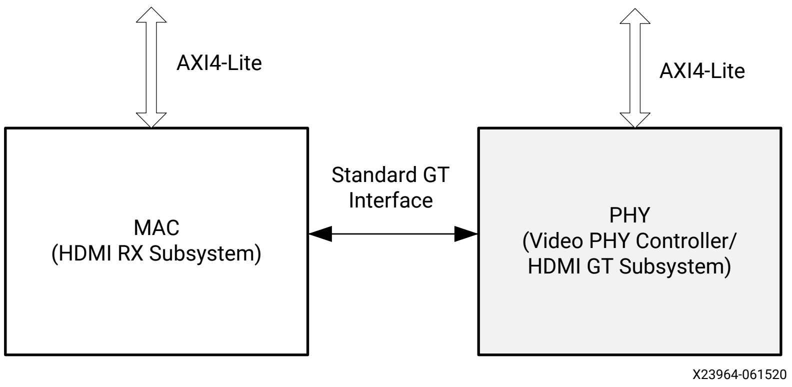
The following figure shows the HDMI 1.4/2.0 RX Subsystem architecture.

Figure 44: **Subsystem Driver Architecture**



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

Figure 45: **MAC Interfaces with PHY**



Usage

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

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

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

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

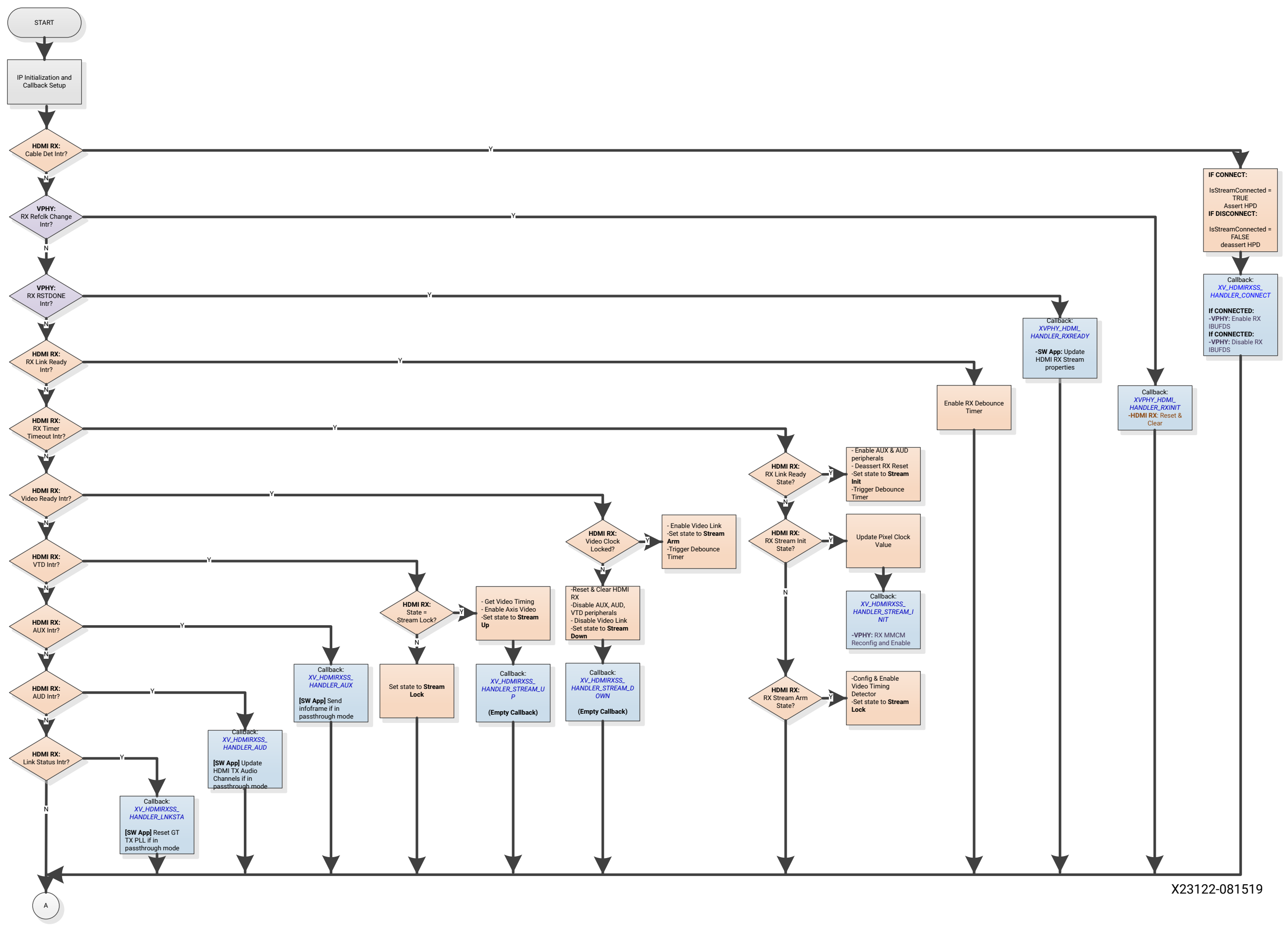
HDMI RX Subsystem Flow

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

- HDMI Cable Detect (5v) from the source device
- HDMI Video Stream changes in the term of TMDS frequency change from the source device

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

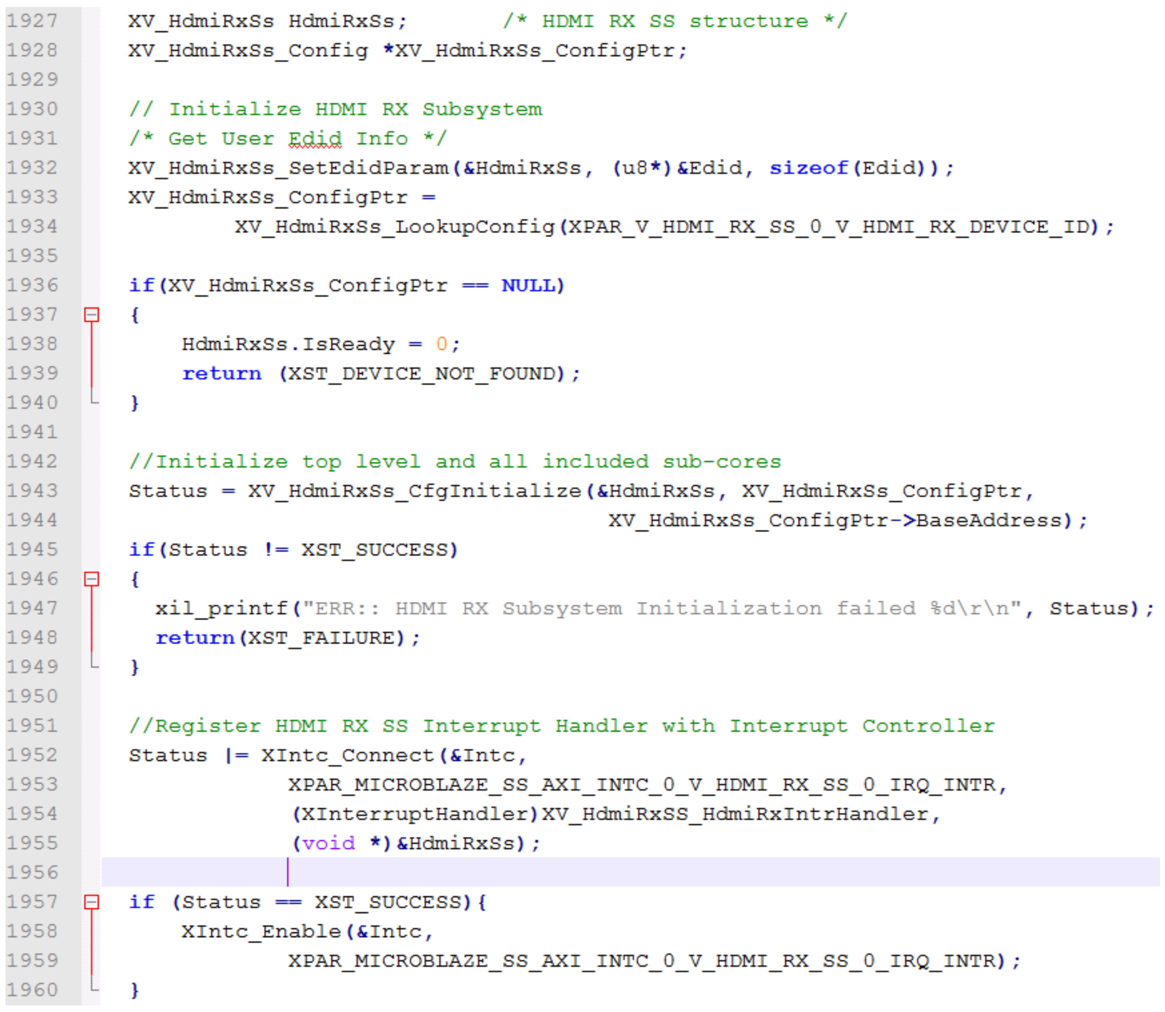
Figure 46: **RX Flow**


START
IP Initialization and Callback Setup
HDMI RX: Cable Det Intr?
VPHY: RX Refclk Change Intr?
VPHY: RX RSTDONE Intr?
HDMI RX: RX Link Ready Intr?
HDMI RX: RX Timer Timeout Intr?
HDMI RX: Video Ready Intr?
HDMI RX: VTD Intr?
HDMI RX: AUX Intr?
HDMI RX: AUD Intr?
HDMI RX: Link Status Intr?
Y
N
IF CONNECT: IsStreamConnected = TRUE Assert HPD IF DISCONNECT: IsStreamConnected = FALSE deassert HPD
Callback: XV_HDMIRXSS_ HANDLER_CONNECT If CONNECTED: -VPHY: Enable RX IBUFDS If CONNECTED: -VPHY: Disable RX IBUFDS
Callback: XVPHY_HDMI_ HANDLER_RXINIT -HDMI RX: Reset & Clear
Callback: XVPHY_HDMI_ HANDLER_RXREADY -SW App: Update HDMI RX Stream properties
Enable RX Debounce Timer
HDMI RX: RX Link Ready State?
- Enable AUX & AUD peripherals - Deassert RX Reset -Set state to Stream Init -Trigger Debounce Timer
HDMI RX: RX Stream Init State?
Update Pixel Clock Value
Callback: XV_HDMIRXSS_ HANDLER_STREAM_I NIT -VPHY: RX MMCM Reconfig and Enable
HDMI RX: RX Stream Arm State?
-Config & Enable Video Timing Detector -Set state to Stream Lock
HDMI RX: Video Clock Locked?
- Enable Video Link -Set state to Stream Arm -Trigger Debounce Timer
-Reset & Clear HDMI RX -Disable AUX, AUD, VTD peripherals - Disable Video Link -Set state to Stream Down
Callback: XV_HDMIRXSS_ HANDLER_STREAM_D OWN (Empty Callback)
HDMI RX: State = Stream Lock?
- Get Video Timing - Enable Axis Video -Set state to Stream Up
Callback: XV_HDMIRXSS_ HANDLER_STREAM_U P (Empty Callback)
Set state to Stream Lock
Callback: XV_HDMIRXSS_ HANDLER_AUX [SW App] Send infoframe if in passthrough mode
Callback: XV_HDMIRXSS_ HANDLER_AUD [SW App] Update HDMI TX Audio Channels if in passthrough mode
Callback: XV_HDMIRXSS_ HANDLER_LNKSTA [SW App] Reset GT TX PLL if in passthrough mode
A
X23122-081519

Application Integration

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

Figure 47: **Application Example Code**



```
XV_HdmiRxSs HdmiRxSs;       /* HDMI RX SS structure */
XV_HdmiRxSs_Config *XV_HdmiRxSs_ConfigPtr;

// Initialize HDMI RX Subsystem
/* Get User Edid Info */
XV_HdmiRxSs_SetEdidParam(&HdmiRxSs, (u8*)&Edid, sizeof(Edid));
XV_HdmiRxSs_ConfigPtr =
        XV_HdmiRxSs_LookupConfig(XPAR_V_HDMI_RX_SS_0_V_HDMI_RX_DEVICE_ID);

if(XV_HdmiRxSs_ConfigPtr == NULL)
{
    HdmiRxSs.IsReady = 0;
    return (XST_DEVICE_NOT_FOUND);
}

//Initialize top level and all included sub-cores
Status = XV_HdmiRxSs_CfgInitialize(&HdmiRxSs, XV_HdmiRxSs_ConfigPtr,
                                    XV_HdmiRxSs_ConfigPtr->BaseAddress);
if(Status != XST_SUCCESS)
{
  xil_printf("ERR:: HDMI RX Subsystem Initialization failed %d\r\n", Status);
  return(XST_FAILURE);
}

//Register HDMI RX SS Interrupt Handler with Interrupt Controller
Status |= XIntc_Connect(&Intc,
            XPAR_MICROBLAZE_SS_AXI_INTC_0_V_HDMI_RX_SS_0_IRQ_INTR,
            (XInterruptHandler)XV_HdmiRxSS_HdmiRxIntrHandler,
            (void *)&HdmiRxSs);

if (Status == XST_SUCCESS){
    XIntc_Enable(&Intc,
            XPAR_MICROBLAZE_SS_AXI_INTC_0_V_HDMI_RX_SS_0_IRQ_INTR);
}
```

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

1. Include the subsystem header file `xv_hdmirxss.h` that defines the subsystem object.
2. Provide the storage for a subsystem driver instance in your application code. For example:

```
XV_HdmiRxSs HdmiRxSs;
```

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_HdmiRxSs_Config *XV_HdmiRxSs_ConfigPtr;
```

4. Set the EDID parameter for the HDMI 1.4/2.0 RX Subsystem Subsystem.

```
void XV_HdmiRxSs_SetEdidParam(XV_HdmiRxSs *InstancePtr,
                              u8 *EdidDataPtr,
                              u16 Length);
```

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

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

```
XV_HdmiRxSs_Config* XV_HdmiRxSs_LookupConfig(u32 DeviceId);
int XV_HdmiRxSs_CfgInitialize(XV_HdmiRxSs *InstancePtr,
                              XV_HdmiRxSs_Config *CfgPtr,
                              u32 EffectiveAddr);
```

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

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

6. 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`.

***Note*:**

1. Prepare the 256 bytes of EDID data and store in an array before calling the specified API function.
2. The EDID data is loaded into the HDMI 1.4/2.0 RX Subsystem during initialization. This is handled by the subsystem driver, no user intervention is needed.

HDCP RX Overview

The HDMI 1.4/2.0 RX Subsystem driver is responsible for combining the HDCP 1.4 and HDCP 2.3 driver APIs into a single common API for use by the user level application. The common HDCP driver API is able to handle the following HDCP configurations: HDCP 1.4 only, HDCP 2.3 only, and both. When both protocols are enabled, the common HDCP driver ensures that only one is active at any given time.

HDCP RX Driver Integration

This section describes the steps required to initialize and run the HDCP RX. 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 2.3, 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 2.3 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_HdmiRxSs_HdcpSetKey`
     - `XV_HDMIRXSS_KEY_HDCP14`
     - `XV_HDMIRXSS_KEY_HDCP22_LC128` (128-bit DCP Licensed Constant)
     - `XV_HDMIRXSS_KEY_HDCP22_PRIVATE` (902-byte DCP Receiver Device Key Set)

Position in Bytes	Size in Bytes	Description
0–39	40	Reserved
40–561	522	Device Public Certificate
562–901	340	Device Private Key[1]

Notes:

1. The actual Private Key size is 320 bytes. The additional 20 bytes is for a hash code that can be used by the software application to verify the keys before loading them. These additional bytes are not used by the HDCP driver.

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

2. Initialize the HDMI 1.4/2.0 RX Subsystem driver after the HDCP keys have been loaded. Initializing the subsystem starts the HDCP 1.4/2.3 drivers internally.
3. Connect the HDCP interrupt handlers to the interrupt controller interrupt ID:
   - `XV_HdmiRxSS_HdcpIntrHandler`
   - `XV_HdmiRxSS_HdcpTimerIntrHandler`
4. Set the HDCP user callback functions. These callback functions are optional and used to hook into the HDCP state machine and allow the user to take action at various stages of the HDCP protocol. If there is no use for the callback at the application level, then the callback can be left undefined.
   - `XV_HdmiRxSs_SetCallback`
     - `XV_HDMIRXSS_HANDLER_HDCP_AUTHENTICATED`
     - `XV_HDMIRXSS_HANDLER_HDCP_AUTHENTICATION_REQUEST`

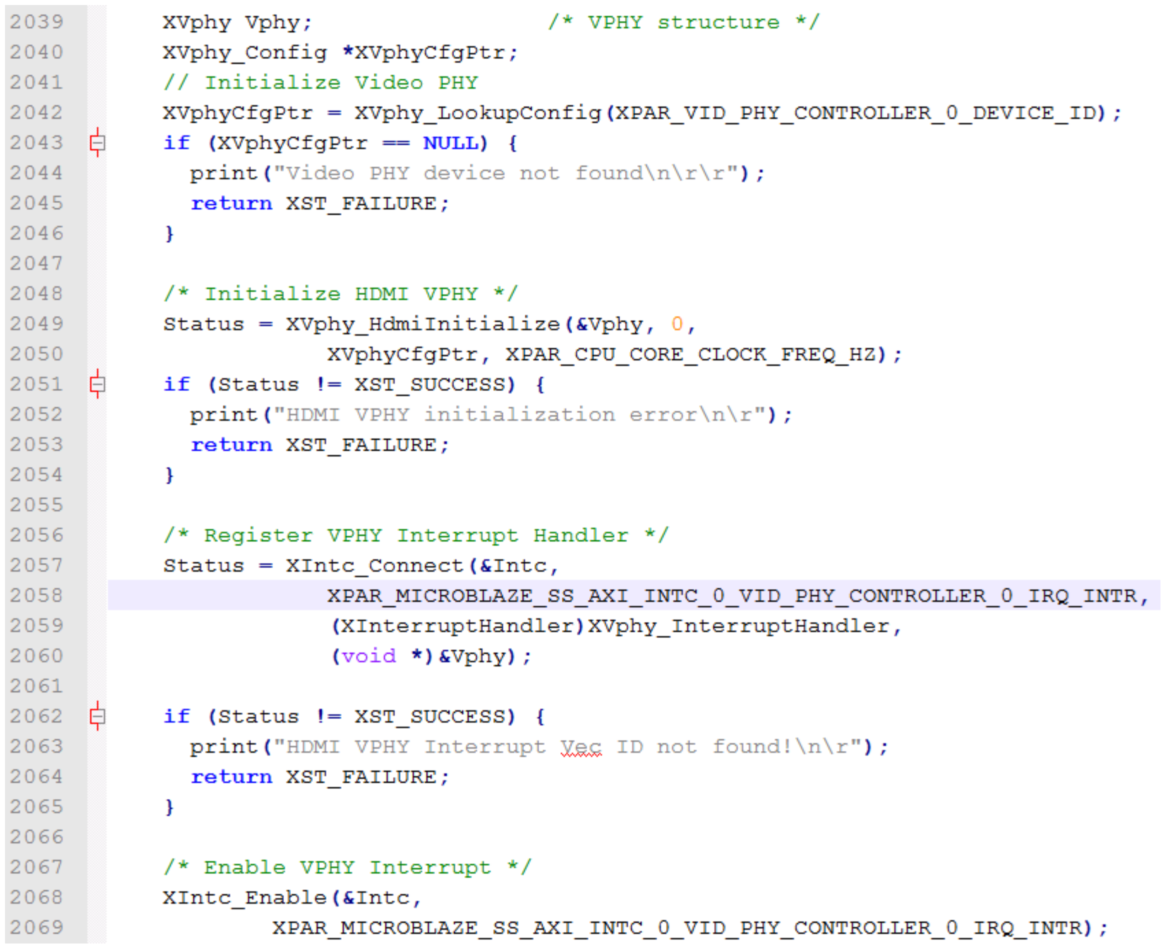
  - `XV_HDMIRXSS_HANDLER_HDCP_STREAM_MANAGE_REQUEST`
  - `XV_HDMIRXSS_HANDLER_HDCP_UNAUTHENTICATED`
  - `XV_HDMIRXSS_HANDLER_HDCP_TOPOLOGY_UPDATE`
  - `XV_HDMIRXSS_HANDLER_HDCP_ENCRYPTION_UPDATE`

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 RX 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_HdmiRxSs_HdcpPoll`
6. Set the HDCP protocol capability to notify the upstream transmitter which protocols are supported. Setting the HDCP protocol capability is optional and is used to override the default capability of the receiver. The default capability is set to both protocols. There are only two valid capability options: both and none. Setting the capability to none disables the HDCP DDC slave device. Note that setting the protocol capability does not enable the protocol. The enabling of the protocol is automatically handled by the subsystem, thus the legacy XV_HdmiRxSs_HdcpSetProtocol API is deprecated. When the capability is changed, HPD should be immediately toggled to get the attention of the upstream transmitter.
   - `XV_HdmiRxSs_HdcpSetCapability`
     - `XV_HDMIRXSS_HDCP_NONE`
     - `XV_HDMIRXSS_HDCP_BOTH`
7. Check the status of authentication.
   - `XV_HdmiRxSs_HdcpIsAuthenticated`
8. Check the status of the cipher encryption. This is the instantaneous encryption status of the cipher and can change between subsequent frames.
   - `XV_HdmiRxSs_HdcpIsEncrypted`
9. Check the overall HDCP protocol status and log data. You can also set the level of detail for log information reported.
   - `XV_HdmiRxSs_HdcpInfo`
   - `XV_HdmiRxSs_SetInfoDetail`

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

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

Figure 48: **Application Example Code**



```
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);
```

To integrate and use the Video PHY Controller/HDMI GT Subsystem for the HDMI 1.4/2.0 RX 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/HDMI GT Subsystem instance in your application code.

   Example:

   ```
   XVphy Vphy;
   ```

3. In the Video PHY Controller/HDMI GT Subsystem 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 instance, the above data structure needs to be initialized based on its hardware configuration, which is passed through meta-structure from `xparameters.h` uniquely identified by the device ID.

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

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

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

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

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

***Note*:**

- Xilinx recommends initializing the PHY controller after the HDMI 1.4/2.0 RX Subsystem initialization is completed.
- Registering the PHY Controller interrupts are part of system application integration. Steps are shown in the previous section and not repeated here.

Interrupts

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

1. HPD – Peripheral I/O to assert HDMI cable 5.0V signal
    a. Rising edge – Cable connected
    b. Falling edge – Cable disconnected
2. Link Ready – Every time the PHY Controller is reconfigured, the `link_clk` is regenerated. An HDMI RX 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. Video Ready – This interrupt is generated by the HDMI RX sub-core to reflect the status of the received video stream.
    a. Rising edge – Video Stream is stable (StreamUp)
    b. Falling edge – Video Stream is not stable (StreamDown)
4. HDMI Receiver Auxiliary Infoframe Interrupt – This interrupt is generated when an Auxiliary Infoframe is received.
5. HDMI Receiver Audio Infoframe Interrupt – This interrupt is generated when an Audio Infoframe is received.
6. HDCP1.4 Interrupt (only available when HDCP 1.4 is enabled in hardware)

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

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

Interrupts	Callback
HPD	XV_HDMIRXSS_HANDLER_CONNECT
Link Ready	
Video Ready (edge triggered)	XV_HDMIRXSS_HANDLER_STREAM_UP XV_HDMIRXSS_HANDLER_STREAM_DOWN Video Ready rising edge: Stream Up Video Ready falling edge: Stream Down
	XV_HDMIRXSS_HANDLER_STREAM_INIT This callback function is not directly mapped to any interrupt source. Instead it is executed when a stream is detected and the Video PHY Controller/HDMI GT Subsystem is stabilized for the HDMI 1.4/2.0 RX Subsystem to start stream locking.
HDMI Receiver Auxiliary Infoframe Interrupt	XV_HDMIRXSS_HANDLER_AUX
HDMI Receiver Audio Interrupt	XV_HDMIRXSS_HANDLER_AUD
HDCP 1.4 Interrupt	
HDCP 1.4 Timer Interrupt	
HDCP 2.3 Timer Interrupt (only available when HDCP 2.3 is enabled in hardware)	
	XV_HDMIRXSS_HANDLER_HDCP_AUTHENTICATE This callback function is not directly mapped to any interrupt source. Instead it is executed when the HDCP authentication state machine has reached the authenticated state.

Application Callback Functions

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

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

```
int XV_HdmiRxSs_SetCallback(XV_HdmiRxSs *InstancePtr,
                            u32 HandlerType,
                            void *CallbackFuncPtr,
                            void *CallbackRef);
```

Available handlers are defined in `xv_hdmirxss.h`:

- XV_HDMIRXSS_HANDLER_CONNECT
- XV_HDMIRXSS_HANDLER_AUX
- XV_HDMIRXSS_HANDLER_AUD
- XV_HDMIRXSS_HANDLER_STREAM_UP
- XV_HDMIRXSS_HANDLER_STREAM_DOWN

- XV_HDMIRXSS_HANDLER_STREAM_INIT

XV_HDMIRXSS_HANDLER_CONNECT

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

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

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

2. Clear Video PHY RX TMDS Clock ratio when cable is disconnected:

```
Vphy.HdmiRxTmdsClockRatio = 0;
```

XV_HDMIRXSS_HANDLER_AUX

This interrupt is triggered every time when an Auxiliary InfoFrame packet is received.

The callback function must retrieve the InfoFrame packet data to be used by the system application.

XV_HDMIRXSS_HANDLER_AUD

This interrupt is triggered every time an active audio stream is detected or the number of active audio channels changes.

The callback function can update the audio information to the application software.

***Note*:** You can call function `XV_HdmiRxSs_GetAudioFormat` in Audio interrupt callback.

- 1 - LPCM
- 2 - HBR

XV_HDMIRXSS_HANDLER_STREAM_UP

This interrupt is triggered every time an HDMI video stream becomes locked.

The callback function can update stream up information to the application software.

XV_HDMIRXSS_HANDLER_STREAM_DOWN

This interrupt is triggered when an HDMI video stream becomes unlocked.

The callback function can update stream down information to the application software. The application software might start a timer and put the system into standby mode if the HDMI 1.4/2.0 RX Subsystem remains unlocked for a certain time.

XV_HDMIRXSS_HANDLER_STREAM_INIT

This interrupt is triggered every time a stream is detected and the Video PHY Controller/HDMI GT Subsystem is stabilized for the HDMI 1.4/2.0 RX Subsystem to start stream locking.

The callback function must perform the following steps:

1. Check the event is for cable connected or cable disconnected.

   ```
   XV_HdmiRxSs *HdmiRxSsPtr = (XV_HdmiRxSs *)CallbackRef;
   HdmiRxSsPtr->IsStreamConnected
   ```

   - 1 - Connected
   - 0 - Disconnected

2. Retrieve the video stream information.

   ```
   XVidC_VideoStream *XV_HdmiRxSs_GetVideoStream(XV_HdmiRxSs *InstancePtr);
   ```

3. Calculate the HDMI MMCM parameter based on the video stream information received.

   ```
   u32 XVphy_HdmiCfgCalcMmcmParam(XVphy *InstancePtr,
                                  u8 QuadId,
                                  XVphy_ChannelId ChId,
                                  XVphy_DirectionType Dir,
                                  XVidC_PixelsPerClock Ppc,
                                  XVidC_ColorDepth Bpc);
   ```

4. Enable the Video PHY Controller/HDMI GT Subsystem to start the MMCM.

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

   where,

   `InstancePtr` is a pointer to the XVphy core instance.

   `QuadId` is the GT quad ID to operate on.

   `Dir` is an indicator for TX or RX.

   ***Note*:** QuadId is not used and should be set to 0.

   For example:

   ```
   XVphy_MmcmStart(&Vphy, 0, XVPHY_DIR_RX);
   ```

XV_HDMIRXSS_HANDLER_HDCP_AUTHENTICATE

If HDCP 1.4 or HDCP 2.3 is enabled in the HDMI 1.4/2.0 RX Subsystem hardware, this interrupt is triggered when HDCP passes its authentication state.

The callback function can update HDCP authentication status to the application software.

Video PHY Controller/HDMI GT Subsystem Interrupt Handlers for HDMI 1.4/2.0 RX Subsystem

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

- XVPHY_HDMI_HANDLER_RXINIT
- XVPHY_HDMI_HANDLER_RXREADY

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_RXINIT

This interrupt is triggered every time the Video PHY Controller/HDMI GT Subsystem detects an HDMI RX reference clock change.

The callback function must perform the following:

1. Initialize a reference clock change process for the HDMI 1.4/2.0 RX Subsystem.

   ```
   void XV_HdmiRxSs_RefClockChangeInit(XV_HdmiRxSs *InstancePtr);
   ```

2. Set the Video PHY Controller/HDMI GT Subsystem HDMI RX reference TMDS clock ratio.

   ```
   VphyPtr->HdmiRxTmdsClockRatio = HdmiRxSs.TMDSClockRatio;
   ```

XVPHY_HDMI_HANDLER_RXREADY

This interrupt is triggered every time the Video PHY Controller/HDMI GT Subsystem RX reset lock is done.

The callback function must perform the following:

1. Check the Video PHY Controller/HDMI GT Subsystem PLL type.

   ```
   XVphy_PllType XVphy_GetPllType(XVphy *InstancePtr,
                                  u8 QuadId,
                                  XVphy_DirectionType Dir,
                                  XVphy_ChannelId ChId);
   ```

2. Set the HDMI 1.4/2.0 RX Subsystem video stream according to the PLL type.

```
XV_HdmiRxSs_SetStream(XV_HdmiRxSs *InstancePtr,
                      u32 Clock,
                      u32 LineRate);
```

   where both Clock and LineRate are from the Video PHY Controller/HDMI GT Subsystem data structure.

Follow the steps in Chapter 6: Example Design chapter to create an example design, which contains all the procedures implemented and can serve as a reference for integrating the HDMI 1.4/2.0 RX Subsystem into your system.

Example Use Cases

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

Related Information

Application Callback Functions

Use Case 1: Cable Plug In

HPD interrupt is received indicating Cable Connection.

- Callback function registered to XV_HDMIRXSS_HANDLER_CONNECT Interrupt type is called.

Use Case 2: Cable Plug Out

1. RX Stream Down interrupt is received.
   - Callback function registered to XV_HDMIRXSS_HANDLER_STREAM_DOWN interrupt type is called.
2. HPD interrupt is received indicating Cable Disconnection.
   - Callback function registered to XV_HDMIRXSS_HANDLER_CONNECT Interrupt type is called.

Use Case 3: Received Video Stream

1. Video PHY Controller/HDMI GT Subsystem HDMI RX Init interrupt is received.
   - Callback function registered to XVPHY_HDMI_HANDLER_RXINIT Interrupt type is called.
2. Video PHY Controller/HDMI GT Subsystem HDMI RX Ready interrupt is received.

- Callback function registered to XVPHY_HDMI_HANDLER_RXREADY interrupt type is called.

3. RX Audio Interrupt is received.
   - Callback function registered to XV_HDMIRXSS_HANDLER_AUD interrupt type is called.
4. RX Stream Initialization Interrupt is received.
   - Callback function registered to XV_HDMIRXSS_HANDLER_HDCP_AUTHENTICATEiInterrupt type is called.
5. When video stream is locked, the RX stream up interrupt is received.
   - Callback function registered to XV_HDMIRXSS_HANDLER_STREAM_UP interrupt type is called.

Now, the video stream has been detected and you can retrieve the stream information using the provided API:

```
XVidC_VideoStream *XV_HdmiRxSs_GetVideoStream(XV_HdmiRxSs *InstancePtr);
```

Use Case 4: Video Stream Change

1. RX Stream Down interrupt is received.
   - Callback function registered to XV_HDMIRXSS_HANDLER_STREAM_DOWN interrupt type is called.
2. Video PHY Controller/HDMI GT Subsystem HDMI RX Init interrupt is received.
   - Callback function registered to XVPHY_HDMI_HANDLER_RXINIT interrupt type is called.
3. Video PHY Controller/HDMI GT Subsystem HDMI RX Ready interrupt is received.
   - Callback function registered to XVPHY_HDMI_HANDLER_RXREADY interrupt type is called.
4. RX Audio Interrupt is received.
   - Callback function registered to XV_HDMIRXSS_HANDLER_AUD interrupt type is called.
5. RX Stream Initialization Interrupt is received.
   - Callback function registered to XV_HDMIRXSS_HANDLER_STREAM_INIT interrupt type is called.
6. When Video stream is locked, the RX stream up interrupt is received.
   - Callback function registered to XV_HDMIRXSS_HANDLER_STREAM_UP interrupt type is called.

Now, the video stream has been detected and you can retrieve the stream information using the provided API:

```
XVidC_VideoStream *XV_HdmiRxSs_GetVideoStream(XV_HdmiRxSs *InstancePtr);
```

Use Case 5: Receive Infoframe

The Auxiliary InfoFrame received interrupt is received.

Callback function registered to XV_HDMIRXSS_HANDLER_AUX Interrupt type is called.

Use Case 6: How to Disable Internal EDID

If you do not want to enable the internal EDID support, you can comment out the function `XV_HdmiRx_DdcLoadEdid` in `xv_hdmirxss_coreinit.c`, using a function call.

```
int XV_HdmiRxSs_SubcoreInitHdmiRx(XV_HdmiRxSs *HdmiRxSsPtr)
```

Example:

```
// Load EDID
// XV_HdmiRx_DdcLoadEdid(HdmiRxSsPtr->HdmiRxPtr, HdmiRxSsPtr->EdidPtr,
// HdmiRxSsPtr->EdidLength);
```

Appendix E

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

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

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

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

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

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

References

These documents provide supplemental material useful with this guide:

1. *Vivado Design Suite: AXI Reference Guide* (UG1037)
2. *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892)
3. *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893)
4. *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS182)
5. *Virtex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS183)
6. *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS181)
7. *Zynq-7000 SoC (Z-7007S, Z-7012S, Z-7014S, Z-7010, Z-7015, and Z-7020) Data Sheet: DC and AC Switching Characteristics* (DS187)
8. *Zynq-7000 SoC (Z-7030, Z-7035, Z-7045, and Z-7100) Data Sheet: DC and AC Switching Characteristics* (DS191)
9. *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922)
10. *Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics* (DS923)
11. *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925)
12. *Zynq UltraScale+ RFSoC Data Sheet: DC and AC Switching Characteristics* (DS926)
13. *Versal Prime Series Data Sheet: DC and AC Switching Characteristics* (DS956)
14. *Versal AI Core Series Data Sheet: DC and AC Switching Characteristics* (DS957)
15. *Versal AI Edge Series Data Sheet: DC and AC Switching Characteristics* (DS958)
16. *HDMI Specifications* (http://www.hdmi.org/manufacturer/specification.aspx)
17. *HDCP Specifications* (http://www.digital-cp.com/hdcp-specifications)
18. *AXI4-Stream Video IP and System Design Guide* (UG934)
19. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
20. *Vivado Design Suite User Guide: Designing with IP* (UG896)
21. *Vivado Design Suite User Guide: Getting Started* (UG910)
22. *ISE to Vivado Design Suite Migration Guide* (UG911)
23. *KCU105 Board User Guide* (UG917)
24. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
25. *Vivado Design Suite User Guide: Implementation* (UG904)
26. *AXI Interconnect LogiCORE IP Product Guide* (PG059)
27. *Video PHY Controller LogiCORE IP Product Guide* (PG230)
28. *HDMI GT Controller LogiCORE IP Product Guide* (PG334)
29. *HDCP 2.2 LogiCORE IP Product Guide* (PG249)

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

31. *Video In to AXI4-Stream LogiCORE IP Product Guide* (PG043)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/11/2020 Version 3.1	
N/A	Added HDCP 2.3 support
07/14/2020 Version 3.1	
N/A	Added support for Versal ACAP devices.
07/08/2020 Version 3.1	
Features AUX Packets Audio Output Stream Interface Audio Data Stream Clocking Device, Package, and Speed Grade Selections Running the Reference Design (A53 on Zynq UltraScale+ Devices)	Added 3D Audio Support
11/21/2019 Version 3.1	
Chapter 6: Example Design	Updated Example Design from SDK to the Vitis software platform.
Native Video (Vectored DE) Native Video/Native Video (Vectored DE) Interface Option	Added support for Vectored DE to the native video interface.
02/12/2019 Version 3.1	
N/A	• Added HDCP Repeater functionality • Added PCB design guidelines for TMDS181 and DP159 • Updated Example Design steps
04/04/2018 Version 3.1	
N/A	• Added VCU118 Example design info and steps • Added new board supports (ZCU104, ZCU106, VCU118). • Added new Example design features. • Updated Example Design steps. • Added Migrating and upgrading section. • Added debug information. • Updated AXI-Lite CPU clock supports. • Updated YUV420 remapping feature illustration.

Section	Revision Summary
12/20/2017 Version 3.0	
N/A	• Updated Example Design steps. • Added notes for DP159 Settings. • Updated notes for CPU clock requirements.
10/04/2017 Version 3.0	
N/A	• Added Example design topology supports (RX-Only, Pass-through). • Added Example design VPHY Configuration Support (NI-DRU Enable/Disable, TXPLL selection, RXPLL selection). • Added Example design new board supports (ZCU102). • Added SDK application supports (RX, Pass-through and HDCP key utility) • Added Information for Example design description. • Added Software Flow diagram. • Added Information if not all audio channels are used. • Added Information about Native Video. • Added Information about Interlaced Video.
04/05/2017 Version 2.0	
N/A	Removed single pixel per clock support
11/30/2016 Version 2.0	
N/A	Added example design migration notes.
10/05/2016 Version 2.0	
N/A	• Added example design flow. • Added HPD XGUI option. • Added software use cases. • Updated Xilinx AUTOMOTIVE APPLICATIONS DISCLAIMER.
06/08/2016 Version 2.0	
N/A	Updated optional video over AXI-Stream support.
04/06/2016 Version 2.0	
N/A	• Added Features section in IP Facts. • Updated Unsupported Features in Overview chapter • Updated Product Specification chapter. • Updated Designing with the Subsystem chapter. • Updated Design Flow Steps chapter. • Updated Hardware Testing and Video Resolutions sections. • Updated Application Software Development appendix.
11/18/2015 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright