Table of Contents

IP Facts

Chapter 1: Overview
  Applications ................................................................. 5
  Unsupported Features .................................................. 5
  Licensing and Ordering Information ............................ 6

Chapter 2: Product Specification
  Introduction ................................................................. 7
  Standards ................................................................. 17
  Performance and Resource Utilization ......................... 17
  Port Descriptions ...................................................... 18
  Clocks and Resets .................................................... 34

Chapter 3: Designing with the Subsystem
  General Design Guidelines ......................................... 35
  Clocking ................................................................. 41
  Resets ................................................................. 45

Chapter 4: Design Flow Steps
  Customizing and Generating the Subsystem .................... 47
  Constraining the Subsystem ...................................... 56
  Simulation .............................................................. 58
  Synthesis and Implementation .................................. 58

Chapter 5: Example Design
  Summary ................................................................. 59
  Running the Example Design ..................................... 65

Appendix A: Verification, Compliance, and Interoperability
  Compliance ............................................................ 81
  Interoperability ....................................................... 82
  Hardware Testing ..................................................... 82
  Video Resolutions ................................................... 83
Appendix B: Migrating and Upgrading
Upgrading in the Vivado Design Suite .................................................. 88

Appendix C: Debugging
Finding Help on Xilinx.com .............................................................. 90
Debug Tools ......................................................................................... 91
Hardware Debug .............................................................................. 92
In-system Debug ............................................................................. 93
Interface Debug .............................................................................. 96

Appendix D: Application Software Development
Device Drivers ................................................................................ 98

Appendix E: Additional Resources and Legal Notices
Xilinx Resources ............................................................................. 115
Documentation Navigator and Design Hubs .................................. 115
References ......................................................................................... 116
Revision History .............................................................................. 117
Please Read: Important Legal Notices ........................................... 118
Introduction

The HDMI 1.4/2.0 Receiver Subsystem is a hierarchical IP that bundles a collection of HDMI® RX 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 RX system.

Features

- HDMI 2.0 and 1.4b compatible
- 2 or 4 symbol/pixel per clock input
- Supports resolutions up to 4,096 x 2,160 @ 60 fps
- 8, 10, 12, and 16-bit Deep-color support
- Support color space for RGB, YUV 4:4:4, YUV 4:2:2, YUV 4:2:0
- Support AXI4-Stream Video output stream and Native Video output stream
- Audio support for up to 8 channels
- High bit rate (HBR)
- Info frames
- Data Display Channel (DDC)
- Hot-Plug Detection
- 3D video support
- Optional High Bandwidth Digital Content Protection (HDCP) 1.4 support
- Optional HDCP 2.2 support
- Optional Video over AXIS compliant NTSC/PAL Support
- Optional Video over AXIS compliant YUV420 Support
- Optional HPD Active polarity
- Optional Cable Detect Active polarity

<table>
<thead>
<tr>
<th>Subsystem Specifics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Device Family [1]</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Supported User Interfaces</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance and Resource Utilization web page</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Provided with Subsystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Files</td>
</tr>
<tr>
<td>Example Design</td>
</tr>
<tr>
<td>Test Bench</td>
</tr>
<tr>
<td>Constraints File</td>
</tr>
<tr>
<td>Simulation Model</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tested Design Flows [4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Entry</td>
</tr>
<tr>
<td>Simulation</td>
</tr>
<tr>
<td>Synthesis</td>
</tr>
</tbody>
</table>

Notes:
1. For a complete list of supported devices, see the Vivado IP catalog.
2. Only HDMI 1.4 is supported in Artix and Kintex -1 devices.
3. Standalone driver details can be found in the SDK directory (<install_directory>/SDK/<release>/data/embeddedsw/doc/xilinx_drivers.htm). Linux OS and driver support information is available from the Xilinx Wiki page.
4. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
Chapter 1

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 video 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 Appendix A, Verification, Compliance, and Interoperability.

Unsupported Features

The following features are not supported in this subsystem:

- Lip sync
- CEC
- HEAC
- HDMI 2.0 dual view
- HDMI 2.0 multi stream audio
Licensing and Ordering Information

License Checkers

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

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

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

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

License Type

This Xilinx® LogiCORE™ IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado® Design Suite. For full access to all subsystem functionalities in simulation and in hardware, you must purchase a license for the subsystem. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the Xilinx HDMI web page.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.
Chapter 2

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 Receiver Subsystem is shown in Figure 2-1.

The HDMI 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 RX Subsystem to output AXI4-Stream based video. By performing this, HDMI RX Subsystem is able to work seamlessly with other Xilinx video processing IP cores. The HDMI RX Subsystem has a built-in capability to optionally support both HDCP 1.4 and HDCP 2.2 decryption.

Figure 2-2 shows the internal structure of the HDMI RX Subsystem when **AXI4-Stream Video Interface** is selected as video interface. In this illustration, both HDCP 1.4 and HDCP 2.2 are selected and both Video over AXIS compliant NTSC/PAL Support and Video over AXIS compliant YUV420 Support are selected.
The HDMI 1.4/2.0 Receiver Subsystem supports two types of video interface:

- AXI4-Stream Video Interface
- Native Video Interface

The HDMI RX Subsystem also provides an option to support a native video interface by constructing the HDMI RX Subsystem without the Video In to AXI4-Stream Bridging module. Therefore, the HDMI RX Subsystem is allowed to output native video to its own video devices. In native video mode, the HDMI RX Subsystem still has a built-in capability to optionally support both HDCP 1.4 and HDCP 2.2 decryption.
Figure 2-3 shows the internal structure of the HDMI RX Subsystem when **Native Video Interface** is selected as video interface.

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, or Zynq UltraScale™+ MPSoC processor. Multiple submodules are used to construct the HDMI RX Subsystem and all the submodules which require software access are connected through an AXI crossbar. Therefore, the MicroBlaze, Zynq-7000 SoC, or Zynq UltraScale+ MPSoC processor is able to access and control each individual submodules inside the HDMI RX Subsystem.

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

The HDMI 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 RX subsystem is connected to a Xilinx Video PHY Controller, which takes electronic signals from a HDMI cable and translates it into HDMI stream. Then, the HDMI RX subsystem converts the HDMI stream into video stream and audio stream. Based on the configuration selected, the HDMI 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.

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

**Display Data Channel (DDC)**

The subsystem allows the end-user to build an HDMI 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 following two bits in SCDC register address offset 0x20 for TMDS configurations (Table 10-19 of HDMI 2.0 spec).

- Bit 1 TMDS_BIt_Clock_Ratio
- Bit 0 Scrambling_Enable

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

- XV_HdmiRx_DdcScdcEnable is used to enable SCDC register.
- XV_HdmiRx_DdcScdcClear is used to clear SCDC register.
Hot Plug Detect

The subsystem supports the Hot Plug Detect (HPD) feature, which is a communication mechanism between HDMI source and HDMI sink devices. For example, when an HDMI cable is inserted between the HDMI source and 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 Figure 2-4, contains 24 data bits (3 bytes) and 8 bits (1 byte) of BCH ECC parity.

<table>
<thead>
<tr>
<th>Byte\Bit #</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>HB0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Packet Type</td>
</tr>
<tr>
<td>HB1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>packet-specific data</td>
</tr>
<tr>
<td>HB2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>packet-specific data</td>
</tr>
<tr>
<td>ECC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ECC</td>
</tr>
</tbody>
</table>

*Figure 2-4: Packet Header*

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

Notes:
1. ECC is calculated in HDMI 1.4/2.0 Receiver Subsystem core. Therefore, you must construct HB0…HB2, and PB0, PB1…PB26, PB27 according to HDMI specs in the software.
2. When calculating the checksum value (PB0), the ECC values are ignored.
3. Refer to section 5.2.3.4 and 5.2.3.5 of the HDMI 1.4 Specification [Ref 13] for more information on the Aux packet structure.

In Table 2-1, the packet types highlighted in blue are handled by hardware and the rest are handled by software.

Table 2-1: Hardware and Software Packet Types

<table>
<thead>
<tr>
<th>Packet Type Value</th>
<th>Packet Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Null</td>
</tr>
<tr>
<td>0x01</td>
<td>Audio Clock Regeneration (N/CTS)</td>
</tr>
<tr>
<td>0x02</td>
<td>Audio Sample (L-PCM and IEC 61937 compressed formats)</td>
</tr>
<tr>
<td>0x03</td>
<td>General Control</td>
</tr>
<tr>
<td>0x04</td>
<td>ACP Packet</td>
</tr>
<tr>
<td>0x05</td>
<td>ISRC1 Packet</td>
</tr>
<tr>
<td>0x06</td>
<td>ISRC2 Packet</td>
</tr>
<tr>
<td>0x07</td>
<td>One Bit Audio Sample Packet</td>
</tr>
<tr>
<td>0x08</td>
<td>DST Audio Packet</td>
</tr>
</tbody>
</table>
Null packets are dropped by HDMI RX Subsystem automatically.

ACR, Audio Sample, One-bit Audio, DST Audio and HBR Audio are routed to audio interface.

The remaining packets (packet types not highlighted in Table 2-1) are routed to the software for further processing by generating AUX packet interrupts.

In the HDMI RX Subsystem driver, there is a generic API function for you to retrieve aux packets.

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

where,
- `InstancePtr` is a pointer to HDMI 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:

```c
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;
```

---

**Table 2-1: Hardware and Software Packet Types (Cont’d)**

<table>
<thead>
<tr>
<th>Packet Type Value</th>
<th>Packet Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x09</td>
<td>High Bitrate (HBR) Audio Stream Packet (IEC 61937)</td>
</tr>
<tr>
<td>0x0A</td>
<td>Gamut Metadata Packet</td>
</tr>
<tr>
<td>0x80+InfoFrame Type</td>
<td>InfoFrame Packet</td>
</tr>
<tr>
<td>0x00</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x01</td>
<td>Vendor-Specific</td>
</tr>
<tr>
<td>0x02</td>
<td>Auxiliary Video Information (AVI)</td>
</tr>
<tr>
<td>0x03</td>
<td>Source Product Descriptor</td>
</tr>
<tr>
<td>0x04</td>
<td>Audio</td>
</tr>
<tr>
<td>0x05</td>
<td>MPEG Source</td>
</tr>
<tr>
<td>0x06</td>
<td>NTSC VBI</td>
</tr>
<tr>
<td>0x07</td>
<td>Dynamic Range and Mastering (HDR)</td>
</tr>
</tbody>
</table>
HDMI Common driver also defines API functions to parse important InfoFrame packets and general control packets so that those information are available to be used in user application software.

The list of available parser APIs are:

```c
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);
```

The parsed information are 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 a HDMI source sends HDR (or HLG) Infoframe, upon receiving them by HDMI Subsystem, the HDR 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 HDR. Then the application software must parse the payload data and handle it according to their system requirements.

**InfoFrames**

As shown in Table 2-1, the software handles General Control Packet, AVI InfoFrame, Audio InfoFrame and Vendor Specific InfoFrame (VSIF). The four packet types are added as part of the HDMI Receiver Subsystem data structure, each having its own well-defined structure.

**AVI InfoFrame**

Use the following code example for AVI InfoFrames:

```c
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;
```
Chapter 2: Product Specification

Audio InfoFrame

Use the following code example for Audio InfoFrames:

```c
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:

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

General Control Packet

```c
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 RX Subsystem, the Xilinx® LogiCORE™ IP High-bandwidth Digital Content Protection (HDCP™) receivers are designed for receiving of audiovisual content securely between two devices that are HDCP capable. In this HDMI RX Subsystem, both HDCP 1.4 and HDCP 2.2 Receiver IP cores are included. However because HDCP 2.2 supersedes the HDCP 1.4 protocol and does not provide backwards compatibility, you need to decide and choose targeted content protection schemes from the Vivado IDE. Four different options are available to choose from:
• No HDCP
• HDCP 1.4 only
• HDCP 2.2 only
• HDCP 1.4 and HDCP 2.2

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

Figure 2-6 shows a configuration of the HDMI receiver where both HDCP 1.4 and 2.2 are enabled. With both HDCP protocols enabled, the HDMI Subsystem configures itself in the cascade topology where the HDCP 1.4 and HDCP 2.2 are connected back-to-back. The HDCP Egress interface of the HDMI 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 receiver subsystem automatically handles HDCP protocol switching when both HDCP 1.4 and HDCP 2.2 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 2.2 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 2-6: HDCP 1.4 and HDCP 2.2 over HDMI Receiver](image-url)
See *HDCP v2.2 Product Guide* (PG249) [Ref 25] and *HDCP v1.4 Product Guide* (PG224) [Ref 26] for more information.

---

**Standards**

The HDMI 1.4/2.0 Receiver Subsystem is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. See the *Vivado AXI Reference Guide* (UG1037) [Ref 1] and *AXI4-Stream Video IP and System Design Guide* (UG934) [Ref 15] for additional information. Also, see HDMI specifications [Ref 13].

The HDMI RX Subsystem is compliant with the HDMI 1.4b and HDMI 2.0 specification [Ref 13].

The Xilinx HDCP 1.4 is designed to be compatible with High-bandwidth Digital Content Protection system Revision 1.4 [Ref 14].

The Xilinx HDCP 2.2 is compliant with the HDCP 2.2 specification entitled High-bandwidth Digital Content Protection, Mapping HDCP to HDMI, Revision 2.2, issued by Digital Content Protection (DCP) LLC [Ref 14].

---

**Performance and Resource Utilization**

For full details about performance and resource utilization, visit the [Performance and Resource Utilization web page](#).

**Maximum Frequencies**

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

- *Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 2]
- *Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 3]
- *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS182) [Ref 4]
- *Virtex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS183) [Ref 5]
- *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS181) [Ref 6]
- *Zynq-7000 SoC: DC and AC Switching Characteristics* (DS187) [Ref 7]
- *Zynq-7000 SoC: DC and AC Switching Characteristics* (DS191) [Ref 8]
Chapter 2: Product Specification

- *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 9]
- *Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 10]
- *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 11]
- *Zynq UltraScale+ RFSoC Data Sheet: DC and AC Switching Characteristics* (DS926) [Ref 12]

---

### Port Descriptions

**Figure 2-7 to Figure 2-10** show the HDMI 1.4/2.0 Receiver Subsystem ports when AXI4-Stream is selected as video interface. The VIDEO_OUT port is expanded in the figure to show the detail AXI4-Stream Video bus signals.

The following subsystem has three default interfaces:

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

\[
\text{video\_data\_width} = \text{int}(3\times\text{BPC}\times\text{PPC}+7)/8\times8
\]

\textbf{Figure 2-7:} HDMI RX Subsystem Pinout – AXI4-Stream Video Interface (No HDCP)
Figure 2-8: HDMI RX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 1.4 Only)
Figure 2-9: HDMI RX Subsystem Pinout – AXI4-Stream Video Interface (HDCP 2.2 Only)
Figure 2-11 to Figure 2-14 show the HDMI 1.4/2.0 Receiver Subsystem ports when Native Video is selected as video interface. The VIDEO_OUT port is expanded in the figure to show the detail Native Video bus signals.
Figure 2-11: HDMI RX Subsystem Pinout – Native Video Interface (No HDCP)
Figure 2-12: HDMI RX Subsystem Pinout – Native Video Interface (HDCP 1.4 Only)
Figure 2-13: HDMI RX Subsystem Pinout – Native Video Interface (HDCP 2.2 Only)
Figure 2-14: HDMI RX Subsystem Pinout – Native Video Interface (HDCP 1.4 and HDCP 2.2)
CPU Interface

Table 2-2 shows the AXI4-Lite control interface signals. This interface is an AXI4-Lite interface and runs at the s_axi_cpu_aclk clock rate. Control of the subsystem is only supported through the subsystem driver.

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

**Table 2-2: CPU Interface Ports**

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_axi_cpu_arsetn</td>
<td>Input</td>
<td>1</td>
<td>Reset (Active-Low)</td>
</tr>
<tr>
<td>s_axi_cpu_aclk</td>
<td>Input</td>
<td>1</td>
<td>Clock for AXI4-Lite control interface</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_awaddr</td>
<td>Input</td>
<td>19</td>
<td>Write address</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_awprot</td>
<td>Input</td>
<td>3</td>
<td>Write address protection</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_awvalid</td>
<td>Input</td>
<td>1</td>
<td>Write address valid</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_awready</td>
<td>Output</td>
<td>1</td>
<td>Write address ready</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_wdata</td>
<td>Input</td>
<td>32</td>
<td>Write data</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_wstrb</td>
<td>Input</td>
<td>4</td>
<td>Write data strobe</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_wvalid</td>
<td>Input</td>
<td>1</td>
<td>Write data valid</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_wready</td>
<td>Output</td>
<td>1</td>
<td>Write data ready</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_bresp</td>
<td>Output</td>
<td>2</td>
<td>Write response</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_bvalid</td>
<td>Output</td>
<td>1</td>
<td>Write response valid</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_bready</td>
<td>Input</td>
<td>1</td>
<td>Write response ready</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_araddr</td>
<td>Input</td>
<td>19</td>
<td>Read address</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_arprot</td>
<td>Input</td>
<td>3</td>
<td>Read address protection</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_arvalid</td>
<td>Input</td>
<td>1</td>
<td>Read address valid</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_aready</td>
<td>Output</td>
<td>1</td>
<td>Read address ready</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_rdata</td>
<td>Output</td>
<td>32</td>
<td>Read data</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_rresp</td>
<td>Output</td>
<td>2</td>
<td>Read data response</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_rvalid</td>
<td>Output</td>
<td>1</td>
<td>Read data valid</td>
</tr>
<tr>
<td>S_AXI_CPU_IN_rready</td>
<td>Input</td>
<td>1</td>
<td>Read data ready</td>
</tr>
</tbody>
</table>
Video Output Stream Interface

This HDMI 1.4/2.0 Receiver Subsystem is supporting two types of video output stream interfaces, which eventually is mapped to HDMI 1.4/2.0 Receiver Subsystem VIDEO_OUT interface.

- AXI4-Stream Video interface
- Native Video Interface

Table 2-3 shows the signals for AXI4-Stream video output streaming 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>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_axis_video_aclk</td>
<td>Input</td>
<td>1</td>
<td>AXI4-Stream clock</td>
</tr>
<tr>
<td>s_axis_video_aresen</td>
<td>Input</td>
<td>1</td>
<td>Reset (Active-Low)</td>
</tr>
<tr>
<td>VIDEO_OUT_tdata</td>
<td>Output</td>
<td>(int((3<em>BPC</em>PPC+7)/8))*8</td>
<td>Data</td>
</tr>
<tr>
<td>VIDEO_OUT_tlast</td>
<td>Output</td>
<td>1</td>
<td>End of line</td>
</tr>
<tr>
<td>VIDEO_OUT_tready</td>
<td>Input</td>
<td>1</td>
<td>Ready</td>
</tr>
<tr>
<td>VIDEO_OUT_tuser</td>
<td>Output</td>
<td>1</td>
<td>Start of frame</td>
</tr>
<tr>
<td>VIDEO_OUT_tvalid</td>
<td>Output</td>
<td>1</td>
<td>Valid</td>
</tr>
</tbody>
</table>

Notes:
1. For AXI4 Stream interface, Video Data width is byte aligned. For example, 10bpc,2ppc data width is 64bits.

Native Video Output Interface

Table 2-4 shows the signals for Native video output interface. This interface is a standard video interface and runs at video_clk clock rate. The data width is user-configurable in the Vivado IDE by setting Max Bits Per Component (BPC) and Number of Pixels Per Clock on Video Interface (PPC).

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>video_clk</td>
<td>Input</td>
<td>1</td>
<td>Video clock</td>
</tr>
<tr>
<td>VIDEO_OUT_field</td>
<td>Output</td>
<td>1</td>
<td>Field ID (only for interlaced video)</td>
</tr>
<tr>
<td>VIDEO_OUT_active_video</td>
<td>Output</td>
<td>1</td>
<td>Active video</td>
</tr>
<tr>
<td>VIDEO_OUT_data</td>
<td>Output</td>
<td>3<em>BPC</em>PPC</td>
<td>Data</td>
</tr>
<tr>
<td>VIDEO_OUT_hsync</td>
<td>Output</td>
<td>1</td>
<td>Horizontal sync</td>
</tr>
</tbody>
</table>
Chapter 2: Product Specification

Audio Output Stream Interface

Table 2-5 shows the signals for AXI4-Stream audio output streaming interfaces. The audio interface transports 24-bits L-PCM or 16-bits HBR audio samples in the IEC 60958 format. A maximum of eight channels are supported. The audio interface is a 32-bit AXI4-Stream master interface and runs at the \( s_{axis\_audio\_aclk} \) clock rate.

### Table 2-5: Audio Output Stream Interface

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_{axis_audio_aclk} )</td>
<td>Input</td>
<td>1</td>
<td>Clock (The audio streaming clock must be greater than or equal to or greater than 128 times the audio sample frequency)</td>
</tr>
<tr>
<td>( s_{axis_audio_aresetn} )</td>
<td>Input</td>
<td>1</td>
<td>Reset (Active-Low)</td>
</tr>
</tbody>
</table>

#### Notes:

1. When native video interface is selected, \( s_{axis\_video\_aclk} \) and \( s_{axis\_video\_aresetn} \) are removed from the HDMI 1.4/2.0 Receiver Subsystem interface ports.
2. \( video\_clk \) is generated by Video PHY Controller LogiCORE IP Product Guide (PG230) [Ref 24].
3. When native video interface is selected, there is no hardware reset.

Audio Clock Regeneration Interface

The audio clock regeneration (ACR) interface has a Cycle Time Stamp (CTS) parameter vector and an Audio Clock Regeneration Value (N) parameter vector. Both vectors are 20 bits wide. The valid signal is driven High when the CTS and N parameters are stable. For more information, see Chapter 7 of the HDMI 1.4 specification [Ref 13].
The subsystem should set up the CTS and N parameters before asserting the valid signal.

Table 2-6 shows the Audio Clock Regeneration (ACR) interface signals. This interface runs at the `s_axis_audio_aclk` clock rate.

### Table 2-6: Audio Clock Regeneration (ACR) Interface

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acr_cts</td>
<td>Output</td>
<td>20</td>
<td>CTS</td>
</tr>
<tr>
<td>acr_n</td>
<td>Output</td>
<td>20</td>
<td>N</td>
</tr>
<tr>
<td>acr_valid</td>
<td>Output</td>
<td>1</td>
<td>Valid</td>
</tr>
</tbody>
</table>

#### HDMI Link Input Interface

Table 2-7 shows the HDMI Link Input interface signals. This interface runs at the `link_clk` clock rate.

### Table 2-7: HDMI Link Input Interface

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link_clk</td>
<td>Input</td>
<td>1</td>
<td>Link clock</td>
</tr>
<tr>
<td>LINK_DATA0_IN_tdata</td>
<td>Input</td>
<td>PPC*10</td>
<td>Link data 0</td>
</tr>
<tr>
<td>LINK_DATA0_IN_tvalid</td>
<td>Input</td>
<td>1</td>
<td>Link Data 0 Valid</td>
</tr>
<tr>
<td>LINK_DATA1_IN_tdata</td>
<td>Input</td>
<td>PPC*10</td>
<td>Link data 1</td>
</tr>
<tr>
<td>LINK_DATA1_IN_tvalid</td>
<td>Input</td>
<td>1</td>
<td>Link Data 1 Valid</td>
</tr>
<tr>
<td>LINK_DATA2_IN_tdata</td>
<td>Input</td>
<td>PPC*10</td>
<td>Link data 2</td>
</tr>
<tr>
<td>LINK_DATA2_IN_tvalid</td>
<td>Input</td>
<td>1</td>
<td>Link Data 2 Valid</td>
</tr>
</tbody>
</table>

#### Data Display Channel Interface

Table 2-8 shows the Data Display Channel interface signals.

### Table 2-8: Data Display Channel (DDC) Interface

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddc_scl_i</td>
<td>Input</td>
<td>1</td>
<td>DDC serial clock in</td>
</tr>
<tr>
<td>ddc_scl_o</td>
<td>Output</td>
<td>1</td>
<td>DDC serial clock out</td>
</tr>
<tr>
<td>ddc_scl_t</td>
<td>Output</td>
<td>1</td>
<td>DDC serial clock tri-state</td>
</tr>
<tr>
<td>ddc_sda_i</td>
<td>Input</td>
<td>1</td>
<td>DDC serial data in</td>
</tr>
<tr>
<td>ddc_sda_o</td>
<td>Output</td>
<td>1</td>
<td>DDC serial data out</td>
</tr>
<tr>
<td>ddc_sda_t</td>
<td>Output</td>
<td>1</td>
<td>DDC serial data tri-state</td>
</tr>
</tbody>
</table>
HDCP 1.4 Key Input Interface (AXI4-Stream Slave Interface)

Table 2-9 shows the signals for HDCP 1.4 key interface. This interface runs at the hdcp14_key_aclk (which is running at AXI4 Lite Clock).

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDCP_KEY_IN_tdata</td>
<td>Input</td>
<td>64</td>
<td>HDCP 1.4 key data</td>
</tr>
<tr>
<td>HDCP_KEY_IN_tlast</td>
<td>Input</td>
<td>1</td>
<td>End of key data</td>
</tr>
<tr>
<td>HDCP_KEY_IN_tready</td>
<td>Output</td>
<td>1</td>
<td>Ready</td>
</tr>
<tr>
<td>HDCP_KEY_IN_tvalid</td>
<td>Input</td>
<td>1</td>
<td>Valid</td>
</tr>
<tr>
<td>hdcp14_key_aclk</td>
<td>Output</td>
<td>1</td>
<td>AXI4-Stream clock</td>
</tr>
<tr>
<td>hdcp14_key_aresetn</td>
<td>Output</td>
<td>1</td>
<td>Reset (Active-Low)</td>
</tr>
<tr>
<td>hdcp14_start_key_transmit</td>
<td>Output</td>
<td>1</td>
<td>Start key transmit</td>
</tr>
<tr>
<td>hdcp14_reg_key_sel</td>
<td>Output</td>
<td>3</td>
<td>Key select</td>
</tr>
</tbody>
</table>

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. Figure 2-15 shows an example of how the HDMI 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 RX Subsystem. For HDCP 1.4 design details, see the HDCP v1.4 Product Guide (PG224) [Ref 26].

![Figure 2-15: HDCP 1.4 Key Management Bus (AXI4-Stream)](image)

However, the HDCP 2.2 key is handled slightly differently as it is solely controlled by the software application. The user application is responsible for providing the infrastructure to securely store and retrieve the keys to be loaded into the HDCP 2.2 drivers. For the detailed list of keys that are required to be loaded by the user application, see the HDCP v2.2 Product Guide (PG249) [Ref 25].
HDCP 2.2 Interrupt Outputs

Table 2-10 shows the signals for HDCP 2.2 interrupt output ports.

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdcp22_irq</td>
<td>Output</td>
<td>1</td>
<td>HDCP 2.2 interrupt</td>
</tr>
<tr>
<td>hdcp22_timer_irq</td>
<td>Output</td>
<td>1</td>
<td>HDCP 2.2 timer interrupt</td>
</tr>
</tbody>
</table>

Miscellaneous Signals with AXI4-Stream Video Interface

Table 2-11 shows the miscellaneous signals with AXI4-Stream video interface selected.

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpd</td>
<td>Output</td>
<td>1</td>
<td>If XGUI option: Hot Plug Detect Active High (Default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Hot Plug Detect is released</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Hot Plug Detect is asserted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If XGUI option: Hot Plug Detect Active Low (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Hot Plug Detect is asserted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Hot Plug Detect is released</td>
</tr>
<tr>
<td>cable_detect</td>
<td>Input</td>
<td>1</td>
<td>If XGUI option: Cable Detect Active High (Default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Cable Detect is released</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Cable Detect is asserted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If XGUI option: Cable Detect Active Low (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Cable Detect is asserted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Cable Detect is released</td>
</tr>
<tr>
<td>video_clk</td>
<td>Input</td>
<td>1</td>
<td>Reference Native Video Clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When AXI4-Stream is selected as Video Interface, a Video In to AXI4-Stream</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bridge module is added to the HDMI RX Subsystem to convert Native Video</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>into AXI4-Stream Video. HDMI RX core uses this video_clk to clock out the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Video Data.</td>
</tr>
<tr>
<td>SB_STATUS_IN_tdata</td>
<td>Input</td>
<td>2</td>
<td>Side Band Status input signals</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0: link_rdy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 1: video_rdy</td>
</tr>
<tr>
<td>SB_STATUS_IN_tvalid</td>
<td>Input</td>
<td>1</td>
<td>Side Band Status input valid</td>
</tr>
<tr>
<td>fid</td>
<td>Output</td>
<td>1</td>
<td>Field ID for AXI4-Stream bus. Used only for interlaced video.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - even field</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - odd field</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For progress video the output is always Low.</td>
</tr>
</tbody>
</table>
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 a 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 ~ 500ms). 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 RX Subsystem that a HDMI source is connected.

### Miscellaneous Signals with Native Video Interface

Table 2-12 shows the miscellaneous signals with native video interface selected.

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpd</td>
<td>Output</td>
<td>1</td>
<td>If XGUI option: Hot Plug Detect Active High (Default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Hot Plug Detect is released</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Hot Plug Detect is asserted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If XGUI option: Hot Plug Detect Active Low (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Hot Plug Detect is asserted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Hot Plug Detect is released</td>
</tr>
<tr>
<td>cable_detect</td>
<td>Input</td>
<td>1</td>
<td>If XGUI option: Cable Detect Active High (Default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Cable Detect is released</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Cable Detect is asserted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If XGUI option: Cable Detect Active Low (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Cable Detect is asserted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Cable Detect is released</td>
</tr>
<tr>
<td>irq</td>
<td>Output</td>
<td>1</td>
<td>Interrupt request for CPU. Active-High.</td>
</tr>
<tr>
<td>SB_STATUS_IN_tdata</td>
<td>Input</td>
<td>2</td>
<td>Side Band Status input signals</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 0: link_rdy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bit 1: video_rdy</td>
</tr>
<tr>
<td>SB_STATUS_IN_tvalid</td>
<td>Input</td>
<td>1</td>
<td>Side Band Status input valid</td>
</tr>
<tr>
<td>video_rst</td>
<td>Output</td>
<td>1</td>
<td>Video reset signal in video_clk domain. Active-High.</td>
</tr>
</tbody>
</table>

1. The Hot Plug Detect (HPD) signal is driven by an HDMI sink and asserted when the HDMI cable is connected to notify the HDMI source of the presence of an HDMI sink. In most cases, the HDMI sink is simply connected to 5V power signal. Therefore, in the PCB, if you choose to use a voltage divider or level shifter, the HPD polarity remains as Active High. However, if you add an inverter to the HPD signal, then the HPD polarity must be set to Active Low in HDMI Transmitter Subsystem GUI. When designing a 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 ~ 500ms). 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 RX Subsystem that a HDMI source is connected.
Chapter 2: Product Specification

Clocks and Resets

Table 2-13 provides an overview of the clocks and resets. See Clocking and Resets in Chapter 3 for more information.

Table 2-13: Clocks and Resets

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_axis_cpu_aclk</td>
<td>Input</td>
<td>1</td>
<td>AXI4-Lite CPU control interface clock.</td>
</tr>
<tr>
<td>s_axis_cpu_aretn</td>
<td>Input</td>
<td>1</td>
<td>Reset, associated with s_axis_cpu_aclk (active-Low). The s_axis_cpu_aretn signal resets the entire subsystem including the data path and AXI4-Lite registers.</td>
</tr>
<tr>
<td>s_axis_video_aclk</td>
<td>Input</td>
<td>1</td>
<td>AXI4-Stream video output clock.</td>
</tr>
<tr>
<td>s_axis_video_aretn</td>
<td>Input</td>
<td>1</td>
<td>Reset, associated with s_axis_video_aclk (active-Low). Resets the AXI4-Stream data path for the video output.</td>
</tr>
<tr>
<td>s_axis_audio_aclk</td>
<td>Input</td>
<td>1</td>
<td>AXI4-Stream Audio output clock. (The audio streaming clock must be greater than or equal to 128 times the audio sample frequency)</td>
</tr>
<tr>
<td>s_axis_audio_aretn</td>
<td>Input</td>
<td>1</td>
<td>Reset, associated with s_axis_audio_aclk (active-Low). Resets the AXI4-Stream data path for the audio output.</td>
</tr>
<tr>
<td>link_clk</td>
<td>Input</td>
<td>1</td>
<td>HDMI Link data input clock. This connects to the Video PHY Controller Link clock output.</td>
</tr>
<tr>
<td>video_clk</td>
<td>Input</td>
<td>1</td>
<td>Clock for the native video interface.</td>
</tr>
</tbody>
</table>

Notes:
1. The reset should be asserted until the associated clock becomes stable.
Chapter 3

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

Audio Data Stream

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

![Figure 3-1: Audio Cycle](image)

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

If an HDMI system does not require audio, tie all audio related input ports to 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 HDMI RX Subsystem, L-PCM (IEC 60958, Packet Type 0x02) and HBR (Packet Type 0x09) 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, Figure 3-2 shows the data format for quad pixels per clock to be fully compliant with the AXI4-Stream video protocol. A data format for a fully compliant AXI4-Stream video protocol dual pixels per clock is illustrated in Figure 3-3.
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.

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

Figure 3-3: Dual Pixels Data Format (Max Bits Per Component = 16)
### Table 3-1: Max Bits Per Component Support

<table>
<thead>
<tr>
<th>Max Bits Per Component</th>
<th>Actual Bits Per Component</th>
<th>Bits Transported by Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>8</td>
<td>[7:0]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[9:0]</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>[11:0]</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>[15:0]</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>[7:0]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[9:0]</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>[11:0]</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>[15:4]</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>[7:0]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[9:0]</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>[11:2]</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>[15:6]</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>[7:0]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[9:2]</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>[15:8]</td>
</tr>
</tbody>
</table>

As an illustration, when **Max Bits Per Component** is set to 12, Figure 3-4 shows the data format for quad pixels per clock to be fully compliant with the AXI4-Stream video protocol. A data format for a fully compliant AXI4-Stream video protocol with dual pixels per clock is illustrated in Figure 3-5.
The video interface can also transport quad and dual pixels in the YUV420 color space. 

**Figure 3-6:** YUV420 Color Space Quad Pixels Data Format
Similarly, for YUV 4:2:0 deep color (10, 12, or 16 bits), the data representation is the same as shown in Figure 3-6 and Figure 3-7. The only difference is that each component carries more bits (10, 12, and 16). However the current data format is not compliant with the AXI4-Stream video protocol. Therefore, a remapping feature is added to HDMI 1.4/2.0 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, Figure 3-7 is extended to Figure 3-8 to represent native video data associated with the clock and control signals. When transporting using AXI4-Stream, the data representation need to be compliant to the protocol defined in UG934. With the remapping feature, the same native video data will be converted into AXI4-Stream formats, which is shown in Figure 3-9. As stated in AXI4-Stream Video IP and System Design Guide [Ref 15], the 4:2:0 format adds vertical subsampling to the 4:2:2 format, which is implemented in Video over AXI4-Stream by omitting the chroma data on every other line.

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

Figure 3-8: Native HDMI Video Interface

However, in the native HDMI video interface, the video data representation must be as shown in Figure 3-9.
Chapter 3: Designing with the Subsystem

**Note:** For RGB/YUV444/YUV422, Video data are directly mapped from AXI4 Stream to Native Video interface without any line buffer. Therefore, Figure 3-2 to Figure 3-5 are common to represent data interface for both AXI4 Stream and Native Video. The control signals are omitted in the figures.

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

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

For more information on the video AXI4-Stream interface and video data format, see the AXI4-Stream Video IP and System Design Guide (UG934) [Ref 15].

### 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 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 192kHz, the ACR fs value shall be equal to the frame rate. For any such stream with an IEC 60958 frame rate above 192kHz, the ACR fs value shall be 1/4th of the frame rate. Therefore, for HBR audio, while calculating N & CTS, the fs to be used is 1/4th of the frame rate. (e.g. 768kHz => 768/4=192kHz).

The HDMI clock structure is illustrated in Figure 3-10 and Table 3-2.

![Figure 3-10: HDMI Clocking Structure](image-url)
### Table 3-2: Clocking

<table>
<thead>
<tr>
<th>Clock</th>
<th>Function</th>
<th>Freq/Rate</th>
<th>Example(^{(1)})</th>
</tr>
</thead>
</table>
| TMDS clock| Source synchronous clock to HDMI interface (This is the actual clock on the HDMI cable). | = 1/10 data rate (for data rates < 3.4 Gb/s) | Data rate = 2.97 Gb/s  
               |                                           | = 1/40 data rate (for data rates > 3.4 Gb/s)  | TMDS clock = 2.97/10 = 297 MHz  
|           |                                                                           |                                | Data rate = 5.94 Gb/s  
               |                                                                           |                                | TMDS clock = 5.94/40 = 148.5 MHz |
| Data clock| This is the actual data rate clock. This clock is not used in the system. It is only listed to illustrate the clock relations. | = TMDS clock (for data rates < 3.4 Gb/s) | Data rate = 2.97 Gb/s  
|           |                                                                           | = TMDS clock * 4 (for data rates > 3.4 Gb/s)  | Data clock = TMDS clock * 1 = 297 MHz  
|           |                                                                           |                                | Data rate = 5.94 Gb/s  
|           |                                                                           |                                | Data clock = TMDS clock * 4 = 594 MHz  
|           |                                                                           |                                | TMDS clock = 148.5 MHz |
| Link clock| Clock used for data interface between HDMI PHY Layer Module and subsystem | for dual pixel video clock=data clock/2 | TMDS clock = 297 MHz  
|           |                                                                           | for quad pixel video clock=data clock/4 | 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 rate = 594 Gb/s  
|           |                                                                           |                                | 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 clock| This is the internal pixel clock. This clock is not used in the system. It is only listed to illustrate the clock relations. | for 8 bpc pixel clock = data clock  
|           |                                                                           | for 10 bpc pixel clock = data clock/1.25  
|           |                                                                           | for 12 bpc pixel clock = data clock/1.5  
|           |                                                                           | for 16 bpc pixel clock = data clock/2  
|           |                                                                           |                                |
Chapter 3: Designing with the Subsystem

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

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

\[
\text{Pixel clock} = \text{Htotal} \times \text{Vtotal} \times \text{Frame Rate}
\]

\[
= 2200 \times 1125 \times 60
\]

\[
= 148,500,000
\]

\[
= 148.5 \text{Mhz}
\]

\[
\text{Video clock} = (\text{Pixel clock})/\text{PPC} = 148.5/2 = 74.25 \text{Mhz}
\]

\[
\text{Data clock} = \text{Pixel clock} \times \text{BPC}/8 = 148.5 \times 12/8 = 222.75 \text{Mhz}
\]

\[
\text{Link clock} = (\text{Data clock})/\text{PPC} = 222.75/2 = 111.375 \text{Mhz}
\]

Using the associative property in this example,

\[
\text{Data clock} = 222.75 \text{Mhz} < 340 \text{Mhz}
\]

then

\[
\text{TMDS clock} = \text{Data clock} = 222.75 \text{Mhz}
\]

Figure 3-11 shows how the clock is distributed in HDMI RX Subsystem and the relationship to the Video PHY Controller.

<table>
<thead>
<tr>
<th>Video Resolution</th>
<th>Horizontal Total</th>
<th>Horizontal Active</th>
<th>Vertical Total</th>
<th>Vertical Active</th>
<th>Frame Rate (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1080p60</td>
<td>2200</td>
<td>1920</td>
<td>1125</td>
<td>1080</td>
<td>60</td>
</tr>
</tbody>
</table>
Chapter 3: Designing with the Subsystem

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

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

- When **AXI4-Stream** is selected, a Video to AXI4_Stream Bridge core in HDMI RX Subsystem helps to covert the native video stream into AXI4-Stream, and then sent out from HDMI RX Subsystem at `axis_video_aclk`.

- When **Native Interface** is selected, native video stream is sent out from HDMI RX Subsystem directly from HDMI RX core at `video_clk`.

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

**Resets**

Each AXI input interface has its own reset signal. The reset signals, `s_axi_cpu_arresetn`, `s_axis_video_arresetn` and `s_axis_audio_arresetn` 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, HDMI RX Subsystem outputs a video_rst signal, which you can use to reset its supporting Native Video processing modules.
Chapter 4

Design Flow Steps

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

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 16]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 17]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 18]

Customizing and Generating the Subsystem

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

The HDMI 1.4/2.0 Receiver Subsystem can be added to a Vivado IP integrator block design in the Vivado Design Suite and can be customized using IP catalog. For more detailed information on customizing and generating the subsystem in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 16]. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl Console.

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

1. In the Flow Navigator, click on Create Block Diagram or Open Block Design under the IP Integrator heading.

2. Right click in the diagram and select Add IP.

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

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

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 17] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 18].

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

### Top Level Tab

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

![Top level tab](image-url)
Chapter 4: Design Flow Steps

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 RX subsystem. The allowable options are AXI4-Stream or Native Video.

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

**Include HDCP 2.2 Decryption:** This option enables HDCP 2.2 decryption.

*Note:* HDCP 1.4 and 2.2 decryption options are only configurable if you have a HDCP license.

**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. For more information on bits mapping, see Table 3-1.

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

---

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

**Video over AXIS compliant NTSC/PAL Support:** This option enables the HDMI RX subsystem to support Video over AXIS compliant NTSC/PAL.

- A pixel repetition of 2 is supported by current hardware
- The pixel repetition factor need to be decoded from the received AVI InfoFrame.

*Note:* Video In to AXI4-Stream IP supports a pixel repetition of 2. It is used as a subcore in HDMI 1.4/2.0 Receiver Subsystem. However, it has 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 HDMI 1.4/2.0 Receiver Subsystem. When 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 RX subsystem to support Video over AXIS compliant YUV420.

**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 Figure 4-2.

![Video Bridge Tab](image)

*Figure 4-2: Video Bridge Tab*

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 Figure 4-3.

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

where,

• Pass-Through showcases the HDMI system built with one HDMI TX Subsystem and one HDMI RX Subsystem, sharing the same Video PHY Controller.

• Rx Only showcases the HDMI system built with only one HDMI RX Subystem and Video PHY Controller. A Frame CRC helper core is added to the Rx Only topology to facilitate system monitor and debugging.
Axilite Frequency: Allows you to choose AXI4-Lite CPU clock. In this release, below options are offered and tested.

- **GTX/7 Series**: 50MHz, 100MHz, 150MHz
- **UltraScale/ UltraScale+ Devices**: 50MHz, 100MHz, 150MHz, 200Mhz

Video PHY Controller Setting Section: Allows the configuration of the Transmitter PLL type and Receiver PLL Type to the Video PHY Controller prior generating the example design. It also allows user to selectively opt-out the NI-DRU to optimize resource utilization if the video resolution they plan to support doesn’t require NI-DRU. Refer to Video PHY Controller Product Guide [Ref 24] for details about NI-DRU requirements.

**Note:** When the Example Design targets VCU118 board and Design Topology is set to Pass-Through, the Include NIDRU option under Video PHY Controller Setting is grayed out and unchecked by default.

Example Design Overview Section: A system block diagram to show the overview of the example design to be generated.
Native Video Interface Option

The native video interface option window is shown in Figure 4-5.

**Figure 4-5: Native Video Interface Option**

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

**Include HDCP 2.2 Decryption:** This option enables HDCP 2.2 decryption.

*Note:* HDCP 1.4 and 2.2 Decryption options are only configurable if you have a HDCP license, else it is disabled.

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

User Parameters

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

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

<table>
<thead>
<tr>
<th>Vivado IDE Parameter/Value</th>
<th>User Parameter/Value</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toplevel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video Interface</td>
<td>C_VID_INTERFACE</td>
<td>AXI4-Stream</td>
</tr>
<tr>
<td>AXI4-Stream</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
### Table 4-1: Vivado IDE Parameter to User Parameter Relationship (Cont’d)

<table>
<thead>
<tr>
<th>Vivado IDE Parameter/Value</th>
<th>User Parameter/Value</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Video</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Include HDCP 1.4 Decryption</td>
<td>C_INCLUDE_HDCP_1_4</td>
<td>Exclude</td>
</tr>
<tr>
<td>Exclude (Untick)</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>Include (Tick)</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>Include HDCP 2.2 Decryption</td>
<td>C_INCLUDE_HDCP_2_2</td>
<td>Exclude</td>
</tr>
<tr>
<td>Exclude (Untick)</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>Include (Tick)</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>Video over AXIS compliant NTSC/PAL Support</td>
<td>C_INCLUDE_LOW_RESO_VID</td>
<td>Exclude</td>
</tr>
<tr>
<td>Exclude (Untick)</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>Include (Tick)</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>Video over AXIS compliant YUV420 Support</td>
<td>C_INCLUDE_YUV420_SUP</td>
<td>Exclude</td>
</tr>
<tr>
<td>Exclude (Untick)</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>Include (Tick)</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>Max bits per component</td>
<td>C_MAX_BITS_PER_COMPONENT</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Number of pixels per clock on Video Interface</td>
<td>C_INPUT_PIXELS_PER_CLOCK</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Hot Plug Detect Active</td>
<td>C_HPD_INVERT</td>
<td>High</td>
</tr>
<tr>
<td>High</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>Cable Detect Active</td>
<td>C_CD_INVERT</td>
<td>High</td>
</tr>
<tr>
<td>High</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>EDID RAM Size</td>
<td>C_EDID_RAM_SIZE</td>
<td>256</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>512</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>4096</td>
<td></td>
</tr>
</tbody>
</table>

Video Bridge
## Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 17].
Constraining the Subsystem

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

Required Constraints

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

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

When using this subsystem in the Vivado® Design Suite flow with Video PHY Controller modules, link_clk and video_clk are generated from the Video PHY Controller. Therefore, the clock constraints are set to the Video PHY Controller constraints instead of these generated clocks. See Clocking in the Video PHY Controller LogiCORE™ IP Product Guide (PG230) [Ref 24] for more information.

s_axi_cpu_aclk, s_axis_video_aclk, and s_axis_audio_aclk constraints are generated at system-level, for example by using a clock wizard.
Device, Package, and Speed Grade Selections

For more information on the device constraint/dependency, see the Video PHY Controller LogiCORE IP Product Guide (PG230) [Ref 24].

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

Table 4-2: Device and Speed Grade Selections

<table>
<thead>
<tr>
<th>Device Family</th>
<th>PPC</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BPC</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Speed Grade</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zynq-7000 SoC</td>
<td>–1</td>
<td>HDMI 1.4(1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>–2</td>
<td>HDMI 2.0(2)</td>
<td></td>
</tr>
<tr>
<td>Zynq UltraScale+ MPSoC</td>
<td>–1</td>
<td>HDMI 2.0(2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>–2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Artix-7</td>
<td>–1</td>
<td>HDMI 1.4(1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>–2</td>
<td>HDMI 1.4(1)</td>
<td></td>
</tr>
<tr>
<td>Kintex-7</td>
<td>–1</td>
<td>HDMI 1.4(1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>–2</td>
<td>HDMI 2.0(2)</td>
<td></td>
</tr>
<tr>
<td>Kintex UltraScale</td>
<td>–1</td>
<td>HDMI 2.0(2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>–2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtex-7</td>
<td>–1</td>
<td>HDMI 1.4(1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>–2</td>
<td>HDMI 2.0(2)</td>
<td></td>
</tr>
<tr>
<td>Virtex UltraScale</td>
<td>–1</td>
<td>HDMI 2.0(2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>–2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtex UltraScale+</td>
<td>–1</td>
<td>HDMI 2.0(2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>–2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 Clocking in Chapter 3 for more information.

Clock Management

This section is not applicable for this IP subsystem.
Clock Placement
This section is not applicable for this IP subsystem.

Banking
This section is not applicable for this IP subsystem.

Transceiver Placement
This section is not applicable for this IP subsystem.

I/O Standard and Placement
This section is not applicable for this IP subsystem.

Simulation
Simulation of the subsystem is not supported.

Synthesis and Implementation
For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 17].
Chapter 5

Example Design

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

Summary

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

Table 5-1: Example Design Support Summary

<table>
<thead>
<tr>
<th>Development Boards</th>
<th>Additional Hardware</th>
<th>Processor</th>
<th>Topology</th>
<th>SDK Import Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>KC705/KCU105/VCU118(1)</td>
<td>inrevium TB-FMCH-HDMI4K FMC daughter card</td>
<td>MicroBlaze™</td>
<td>Pass-through</td>
<td>Passthrough_Microblaze</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rx Only</td>
<td>RxOnly_Microblaze</td>
<td></td>
</tr>
<tr>
<td>ZC706</td>
<td></td>
<td>A9</td>
<td>Pass-through</td>
<td>Passthrough_A9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rx Only</td>
<td>RxOnly_A9</td>
<td></td>
</tr>
<tr>
<td>ZCU102/ZCU104/ZCU106</td>
<td>-</td>
<td>A53</td>
<td>Pass-through</td>
<td>Passthrough_A53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rx Only</td>
<td>RxOnly_A53</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pass-Through</td>
<td>Repeater_A53(2)</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td></td>
<td>A53</td>
<td>Pass-through</td>
<td>Passthrough_R5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rx Only</td>
<td>RxOnly_R5</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. For VCU118 board, no dedicated on-board GT reference clocks are available to support HDMI Transmitter and NI-DRU on HDMI Receiver simultaneously due to board design limitation. Therefore, if Pass-through topology is selected for VCU118 board, NI-DRU is disabled.
2. A dedicated repeater application is added to demonstrate repeater functionality. Please take notes of below points:
   - Repeater feature has been removed from standard passthrough application for all the supported processors.
   - This application passes repeater compliance tests (CTS) on SL8800 running on ZCU102 A53 processor.
   - The same application can be used for other applications too.
   - User will need to increase the BRAM size from 512K to 1M in hardware IPI in order to run repeater function on 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® LogiCORE™ IPs:

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

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 may 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 1.4/2.2 capability of the HDMI solution. HDCP is used to securely send audiovisual data from an HDCP protected transmitter to HDCP protected downstream receivers. Typically, HDCP 2.2 is used to encrypt content at Ultra High Definition (UHD) while HDCP 1.4 is used as a legacy encryption scheme for lower resolutions.

**Hardware**

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

![Figure 5-1](image_url)
**IMPORTANT:** When an unpowered HDMI source is connected to the HDMI Receiver in a pass-through system, the HDMI Example Design UART may get flooded with Starting Colorbar message because of a limitation to the ZCU102 board design.

The VPHY Controller core has been configured for the HDMI application that allows transmission and reception (optional) of HDMI video/audio to and from the HDMI 2.0 daughter card or on-board HDMI 2.0 circuitry.
IMPORTANT: If you use both HDMI Transmitter Subsystems and HDMI Receiver Subsystems in their products, the TI HDMI Cable driver chip DP159 is used in all the Example design, either on Evaluation board itself or on The inrevium TB-FMCH-HDMI4K FMC daughter card. In the HDMI Example Design software, a sample DP159 driver is provided as reference. The settings have not been calibrated for various use cases. If users are using DP159 in their product, they need to adjust the settings based on the circuit design. Some further fine tuning may still be needed according to the compliance results.

In pass-through mode, the VPHY Controller core recovers 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 VPHY Controller core as a high-speed serial data stream. The transition minimized differential signaling (TMDS) clock from the HDMI In interface is forwarded to the HDMI TX transceiver via the SI53xx clock generator in the HDMI 2.0 FMC card or on-board HDMI 2.0 circuitry.

Note: ZCU104 uses a different clock generator IDT 8T49N24x. A sample driver is provided as part of example application software. It is not calibrated for the best performance or to pass compliance. You may need to fine tune its settings in their own design if the same chip is used in their product.

In the RX-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.
High-level control of the system is provided by a simplified embedded processor subsystem containing I/O peripherals and processor support IP. A clock generator block and a processor system reset block supply clock and reset signals for the system, respectively. See Figure 5-5 and Figure 5-6 for a block diagrams of the three types of processor subsystems supported by HDMI Example Design flow.

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

**Figure 5-6:** HDMI Reference Design Block Diagram (Zynq-7000 SoC or Zynq UltraScale+ MPSoC)
Example Design Specifics

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

- MicroBlaze or Zynq-7000 SoC or Zynq UltraScale+ MPSoC
- MicroBlaze Debug Module (Only for MicroBlaze based processor subsystem)
- AXI Interconnect
- Local Memory Bus (Only for MicroBlaze based processor subsystem)
- LMB BRAM Controller (Only for MicroBlaze based processor subsystem)
- Block Memory Generator (Only for MicroBlaze based processor subsystem)
- Clocking Wizard
- Processor System Reset
- AXI UARTLite (Only for MicroBlaze based processor subsystem)
- AXI Interrupt Controller (Only for MicroBlaze based processor subsystem)
- AXI IIC
- AXI GPIO
- Video Test Pattern Generator
- AXI4-Stream Register Slice
- Utility Buffer
- Utility Vector Logic
- AUD_PAT_GEN (Custom IP)
- HDMI_ACR_CTRL (Custom IP)
- HDCP_KEYMNGMT_BLK (Custom IP)

**Note:** When a Custom IP is added to a PI design as RTL reference module, the auto assigned address may 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 (e.g. 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 Xilinx® part or board for the project.

3. Select the target Board, then click **Next > Finish**. (In this release, KC705, KCU105, ZC706, ZCU102, ZCU104, ZCU106 and VCU118 are supported.)

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

5. A Customize IP window opens. Configure HDMI 1.4/2.0 Receiver Subsystem, then select **OK**.
Chapter 5: Example Design

a. Refer to Chapter 4, Design Flow Steps for detail description on Customizing and Generating the Subsystem.

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

c. Native Video Interface is NOT supported in Example Design Flow.
6. The **Generate Output Products** dialog box opens. Select **Generate**.
a. You may optionally select **Skip** if you only want to create an example design and leave the IP generation to later stage.

7. The IP component with given 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 a HDMI Example Design is generated with Block Design to show the system structure. You may choose to build the design by **Run Synthesis, Implementation, and Generate Bitstream**. An overall system IPI block diagram of the KC705 based example design is shown below.
10. After the Hardware is created successfully, the next step is to export the generated HDMI Example Design hardware to build HDMI application software using Xilinx SDK. To export hardware design, in the generated HDMI Example Design Vivado project, select File > Export-> Export Hardware.

11. In Export Hardware window opens, select OK. The hardware definition file is exported to a folder (usually .sdk folder) local to the Vivado project.

12. The Hardware is exported successfully. Then Launch SDK in the same generated HDMI Example Design Vivado project by selecting File > Launch SDK.

13. A Launch SDK window will open, Click OK to use the default location. Xilinx SDK is launched and Hardware design information is shown by default in system.hdf.

15. A Board Support Package Settings window will open. Select OK to close it.

16. A tab to the system.mss will open. In the system.mss, there will be a section named Peripheral Drivers. From the list of Peripheral Drivers, find the HDMI 1.4/2.0 Receiver Subsystem driver (v_hdmiri_ss) and click on Import Examples.
17. An Import Examples window opens with a list of application. Refer to Table 5-3 to select the respective application and click OK.

18. The Example Application will be built and .elf will be ready to use.
Refer to Table 5-1 to select the respective application.

**HDCP Key Utility**

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

To hdcp_key_utility application:

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

The terminal displays the following output:

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

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

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

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

**Formatting HDCP Keys for HDCP 1.x**

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

- The Hdcp14Key1 array
  
  This array holds the HDCP 1.x RX KSV and Keys.

- The Hdcp14Key2 array
  
  This array holds the HDCP 1.x RX KSV and Keys.

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

To format the HDCP 1.x keys for the key_utility, follow the steps below:
1. Discard the 20 byte SHA-1.
2. Pad each key on the right with one byte of 0s (KSV is already padded).

You should now have $1 \times 8$ byte KSV + $40 \times 8$ byte Keys.

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

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

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

**Formatting HDCP Keys for HDCP 2.2**

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

- The `hdcp22_lc128` array

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

- The `Hdcp22RxPrivateKey` array

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

**Running the Reference Design (KC705)**

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

1. Launch the Xilinx System Debugger by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2018.3 > Vivado 2018.3 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:
6. Set the target processor.

xsdb% target -set 3

7. Download the software .elf to the FPGA.

xsdb% dow ./<IP instance name>_ex.sdk/<name of bsp>_xhdmi_example_1/Debug/<name of bsp>_xhdmi_example_1.elf

8. Run the software.

xsdb% stop
xsdb% rst
xsdb% con

9. Exit the XSDB command prompt.

xsdb% exit

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

1. Launch the Xilinx System Debugger by selecting Start > All Programs > Xilinx Design Tools > Vivado 2018.3 > Vivado 2018.3 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)
Chapter 5: Example Design

6. Download the bitstream to the FPGA:

```plaintext
xsdb% targets -set 4 (PSU)
xbdb% rst -system
xsdb% after 3000
xsdb% target -set 3 (PL)
xbdb% fpga -file ./<IP instance name>_ex.runs/impl_1/exdes_wrapper.bit
```

7. Set the target processor.

```plaintext
xsdb% target -set 8 (APU)
xbdb% source ./<IP instance name>_ex.sdk/exdes_wrapper_hw_platform_0/
psu_init.tcl
xsdb% psu_init
xsdb% after 1000
xsdb% psu_ps_pl_isolation_removal
xsdb% after 1000
xsdb% psu_ps_pl_reset_config
xsdb% catch (psu_protection)
xbdb% target -set 9 (Cortex-A53 #0)
xbdb% rst -processor
```

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

8. Download the software .elf to the FPGA

```plaintext
xsdb% dow ./<IP instance name>_ex.sdk/<name of bsp>_xhdmi_example_zynq_us_1/
Debug/<name of bsp>_xhdmi_example_zynq_us_1.elf
```

9. Run the software.

```plaintext
xsdb% con
```

10. Exit the XSDB command prompt.

```plaintext
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 2018.3 > Vivado 2018.3 Tcl Shell**.

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

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

3. Invoke Xilinx System Debugger (**xsdb**).

```plaintext
Vivado% xsdb
```

4. Establish connections to debug targets.

```plaintext
xsdb% connect
```

5. List all available JTAG targets

```plaintext
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)

6. Download the bitstream to the FPGA:
   xsdb% targets -set 4 (PSU)
   xsdb% rst -system
   xsdb% after 3000
   xsdb% target -set 3 (PL)
   xsdb% fpga -file ./<IP instance name>_ex.runs/impl_1/exdes_wrapper.bit

7. Set the target processor.
   xsdb% target -set 5 (RPU)
   xsdb% source ./<IP instance name>_ex.sdk/exdes_wrapper_hw_platform_0/
   psu_init.tcl
   xsdb% psu_init
   xsdb% after 1000
   xsdb% psu_ps_pl_isolation_removal
   xsdb% after 1000
   xsdb% psu_ps_pl_reset_config
   xsdb% catch {psu_protection}
   xsdb% target -set 6 (Cortex-R5 #0)
   xsdb% rst -processor

   **Note:** The target number for PSU, PL, and RPU may be different, to ensure that the associated target numbers are correct, re-run step 5.

8. Download the software .elf to the FPGA
   xsdb% dow ./<IP instance name>_ex.sdk/<name of bsp>_xhdmi_example_zynq_us_1/
   Debug/
   <name of bsp>_0_xhdmi_example_zynq_us_1.elf

9. Run the software.
   xsdb% con

10. Exit the XSDB command prompt.
    xsdb% exit

**Running the Reference Design (A9 on Zynq-7000 AP SoC)**

1. Launch the Xilinx System Debugger by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2018.3 > Vivado 2018.3 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 <SDK Install folder>/scripts/sdk/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 ./<IP instance name>_ex.sdk/exdes_wrapper_hw_platform_0/
   system.hdf
   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 ./<IP instance name>_ex.sdk/<name of bsp>_Passthrough_A9_1/Debug/
   <name of bsp>_Passthrough_A9_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 bitstream generated from Example Design Flow. **KCU105 Board FMCH VADJ Adjustment** shows the steps on how to set the VADJ power rail when using KCU105 board. For more details about KCU105 Board, to KCU105 Board User Guide [Ref 20].
KCU105 Board FMCH VADJ Adjustment

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

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

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

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

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

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

4. After the UART terminal is connected, power cycle the KCU105 board to refresh the system controller menu in the UART terminal. Select this option in the system controller menu:
   a. Adjust FPGA Mezzanine Card (FMC) Settings.

5. In the next menu, select:
   a. Set FMC VADJ to 1.8V.
IMPORTANT: Ensure that the jumpers are set correctly on their HDMI 2.0 FMC daughter card.
Appendix A

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 2.2: 1A, 1B, 2C
- Quantum Data QD980
  - HDCP 2.2: 1A, 1B, 2C

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

Intra-pair skew for DATA[0:2] lanes must be:
- Max 1ps intra-pair skew FPGA->retimer.
- Max 1ps intra-pair skew retimer->connector.
- Target impedance to be 100ohm +/-7% (Max +/-10%).
  - A single excursion is permitted out to a max/min of 100ohms +/-25% and of a duration less than 250ps.
Appendix A: Verification, Compliance, and Interoperability

Interoperability

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

Hardware Testing

The HDMI 1.4/2.0 Receiver 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

This release is tested with the following source devices:

- Quantum Data 980B
- Quantum Data 780B
- 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

Figure A-1 shows the hardware setup for AXI4-Stream Video Interface. An HDMI source connects to Video PHY Controller, which converts the HDMI Video into LINK DATA and sends to the HDMI RX Subsystem. Then, the HDMI RX Subsystem translates the LINK DATA into AXI4-Stream Video and sends to the Test Pattern Generator. By setting the Test Pattern Generator to pass-through mode, the AXI4-Stream Video from the HDMI RX Subsystem is passed to HDMI TX Subsystem where it gets translated to LINK DATA again and sends back to the Video PHY Controller. The Video PHY Controller then converts it back to HDMI Video and sends to HDMI Sink.
For Video PHY Controller settings and PLL selections, see the Video PHY Controller LogiCORE IP Product Guide (PG230) [Ref 24].

Similarly, Figure A-2 shows the hardware setup for Native Video Interface. The only difference is that two Video Bridge modules are added in between the HDMI RX Subsystem and the Test Pattern Generator, and between the Test Pattern Generator to the HDMI TX Subsystem.
Table A-1, Table A-2, and Table A-3 show the video resolutions that were tested as part of the release for different video formats.

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

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Horizontal</th>
<th>Vertical</th>
<th>Frame Rate (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Active</td>
<td>Total</td>
</tr>
<tr>
<td>480i60</td>
<td>858</td>
<td>720</td>
<td>525</td>
</tr>
<tr>
<td>576i50</td>
<td>864</td>
<td>720</td>
<td>625</td>
</tr>
<tr>
<td>1080i50</td>
<td>2640</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>1080i60</td>
<td>2200</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>480p60</td>
<td>858</td>
<td>720</td>
<td>525</td>
</tr>
<tr>
<td>576p50</td>
<td>864</td>
<td>720</td>
<td>625</td>
</tr>
<tr>
<td>720p50</td>
<td>1980</td>
<td>1280</td>
<td>750</td>
</tr>
<tr>
<td>720p60</td>
<td>1650</td>
<td>1280</td>
<td>750</td>
</tr>
<tr>
<td>1080p24</td>
<td>2750</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>1080p25</td>
<td>2640</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>1080p30</td>
<td>2200</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>1080p50</td>
<td>2640</td>
<td>1920</td>
<td>1125</td>
</tr>
</tbody>
</table>
### Appendix A: Verification, Compliance, and Interoperability

#### Table A-1: Tested Video Resolutions for RGB 4:4:4 and YCbCr 4:4:4 (Cont'd)

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Horizontal</th>
<th>Vertical</th>
<th>Frame Rate (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Active</td>
<td>Total</td>
</tr>
<tr>
<td>1080p60</td>
<td>2200</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>1080p120</td>
<td>2200</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>2160p24</td>
<td>5500</td>
<td>3840</td>
<td>2250</td>
</tr>
<tr>
<td>2160p25</td>
<td>5280</td>
<td>3840</td>
<td>2250</td>
</tr>
<tr>
<td>2160p30</td>
<td>4400</td>
<td>3840</td>
<td>2250</td>
</tr>
<tr>
<td>2160p60(3)</td>
<td>4400</td>
<td>3840</td>
<td>2250</td>
</tr>
<tr>
<td>4096x2160p60(3)</td>
<td>4400</td>
<td>4096</td>
<td>2250</td>
</tr>
<tr>
<td>vgap60</td>
<td>800</td>
<td>640</td>
<td>525</td>
</tr>
<tr>
<td>svgap60</td>
<td>1056</td>
<td>800</td>
<td>628</td>
</tr>
<tr>
<td>xgap60</td>
<td>1344</td>
<td>1024</td>
<td>806</td>
</tr>
<tr>
<td>sxgap60</td>
<td>1688</td>
<td>1280</td>
<td>1066</td>
</tr>
<tr>
<td>wgap60</td>
<td>1440</td>
<td>1280</td>
<td>790</td>
</tr>
<tr>
<td>wxga+p60</td>
<td>1792</td>
<td>1366</td>
<td>798</td>
</tr>
<tr>
<td>xgap60</td>
<td>2160</td>
<td>1600</td>
<td>1250</td>
</tr>
<tr>
<td>wxgap60</td>
<td>2592</td>
<td>1920</td>
<td>1245</td>
</tr>
<tr>
<td>wsgap60</td>
<td>2240</td>
<td>1680</td>
<td>1089</td>
</tr>
</tbody>
</table>

**Notes:**
1. Not all resolutions can be supported due to VPHY limitation. For details, refer to [Video PHY Controller LogiCORE IP Product Guide (PG230)](Ref 24).
2. In this release, UXGA 60 Hz is supported in the HDMI 1.4/2.0 Receiver Subsystem for 8, 10, and 12 bits per component only.
3. For 4kp60 YUV444/RGB video, only 8 bits is supported as defined by HDMI 2.0 spec due to bandwidth limitation.

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

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Horizontal</th>
<th>Vertical</th>
<th>Frame Rate (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Active</td>
<td>Total</td>
</tr>
<tr>
<td>1080i50</td>
<td>2640</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>1080i60</td>
<td>2200</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>480p60</td>
<td>858</td>
<td>720</td>
<td>525</td>
</tr>
<tr>
<td>576p50</td>
<td>864</td>
<td>720</td>
<td>625</td>
</tr>
<tr>
<td>720p50</td>
<td>1980</td>
<td>1280</td>
<td>750</td>
</tr>
<tr>
<td>720p60</td>
<td>1650</td>
<td>1280</td>
<td>750</td>
</tr>
<tr>
<td>1080p24</td>
<td>2750</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>1080p25</td>
<td>2640</td>
<td>1920</td>
<td>1125</td>
</tr>
<tr>
<td>1080p30</td>
<td>2200</td>
<td>1920</td>
<td>1125</td>
</tr>
</tbody>
</table>
### Table A-2: Tested Video Resolutions for YCbCr 4:2:0 at 8, 10, 12, 16 Bits/Component

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Horizontal</th>
<th></th>
<th></th>
<th>Frame Rate (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Active</td>
<td>Total</td>
<td>Active</td>
</tr>
<tr>
<td>1080p50</td>
<td>2640</td>
<td>1920</td>
<td>1125</td>
<td>1080</td>
</tr>
<tr>
<td>1080p60</td>
<td>2200</td>
<td>1920</td>
<td>1125</td>
<td>1080</td>
</tr>
<tr>
<td>2160p24</td>
<td>5500</td>
<td>3840</td>
<td>2250</td>
<td>2160</td>
</tr>
<tr>
<td>2160p25</td>
<td>5280</td>
<td>3840</td>
<td>2250</td>
<td>2160</td>
</tr>
<tr>
<td>2160p30</td>
<td>4400</td>
<td>3840</td>
<td>2250</td>
<td>2160</td>
</tr>
<tr>
<td>vgap60</td>
<td>800</td>
<td>640</td>
<td>525</td>
<td>480</td>
</tr>
<tr>
<td>svgap60</td>
<td>1056</td>
<td>800</td>
<td>628</td>
<td>600</td>
</tr>
<tr>
<td>wxgap60</td>
<td>1440</td>
<td>1280</td>
<td>790</td>
<td>768</td>
</tr>
<tr>
<td>wxga+p60</td>
<td>1792</td>
<td>1366</td>
<td>798</td>
<td>768</td>
</tr>
<tr>
<td>uxgap60</td>
<td>2160</td>
<td>1600</td>
<td>1250</td>
<td>1200</td>
</tr>
<tr>
<td>wuxgap60</td>
<td>2592</td>
<td>1920</td>
<td>1245</td>
<td>1200</td>
</tr>
<tr>
<td>wsxgap60</td>
<td>2240</td>
<td>1680</td>
<td>1089</td>
<td>1050</td>
</tr>
</tbody>
</table>

### Table A-3: Tested Video Resolutions for YCbCr 4:2:2 at 12 Bits/component

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Horizontal</th>
<th></th>
<th></th>
<th>Frame Rate (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Active</td>
<td>Total</td>
<td>Active</td>
</tr>
<tr>
<td>2160p60</td>
<td>4400</td>
<td>3840</td>
<td>2250</td>
<td>2160</td>
</tr>
</tbody>
</table>
Appendix B

Migrating and Upgrading

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

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations between core versions.

Parameter Changes

Table lists the parameter changes across the core versions.

<table>
<thead>
<tr>
<th>GUI Parameter</th>
<th>v2.0</th>
<th>v3.0</th>
<th>v3.1</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Topology</td>
<td>Not Applicable</td>
<td>New</td>
<td>No Change</td>
<td>New options in v3.0 for example design configuration. Refer to “Customizing and Generating the Subsystem” section for details.</td>
</tr>
<tr>
<td>Tx PLL Type</td>
<td>Not Applicable</td>
<td>New</td>
<td>No Change</td>
<td></td>
</tr>
<tr>
<td>Rx PLL Type</td>
<td>Not Applicable</td>
<td>New</td>
<td>No Change</td>
<td></td>
</tr>
<tr>
<td>Include NIDRU</td>
<td>Not Applicable</td>
<td>New</td>
<td>No Change</td>
<td></td>
</tr>
<tr>
<td>Axilite Frequency</td>
<td>Not Applicable</td>
<td>Not Applicable</td>
<td>New</td>
<td>New option in v3.1 for example design configuration. Refer to “Customizing and Generating the Subsystem” section for details.</td>
</tr>
</tbody>
</table>

Notes:
1. Not Applicable: GUI option is not available in this version
2. No Change: GUI option remain as in previous version
3. New: New GUI option added in this version

Port Changes

Link Data port width has changed to be configurable based on pixel per clock selection. The port width is reduced to 20 bits from the original 40 bits when user selects 2 pixels per clock.
The Link Data port width remains the default 40 bits for 4 pixels per clock.

**Other Changes**

There are no other changes.

**Migration Notes**

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

- Hot Plug Detect Active has been added to HDMI 1.4/2.0 Transmitter Subsystem GUI. Choose High in the Example Design (according to board design).
- Hot Plug Detect Active has been added to HDMI 1.4/2.0 Receiver Subsystem GUI. Choose Low in Example Design (according to board design).
- Cable Detect Active has been added to HDMI 1.4/2.0 Receiver Subsystem GUI. Choose Low in Example Design (according to board design).
- HDCP 1.4/2.2 is enabled by default in Example Design application software. Removed UART option to Enable HDCP 1.4 or HDCP 2.2.
- Auto switching has been added to the Example Design Application software. You do not need to choose HDCP 1.4 or HDCP 2.2 from UART. A corresponding HDCP is selected according to the capability of connected source/sink. If the device support both HDCP 1.4 and HDCP 2.2, the priority is given to HDCP 2.2.
- HDCP repeater feature has been added. You can enabled/disable it by selecting "h" from UART menu.
- System log is moved from direct UART printout to event log. You can display the event log by selecting "z" from UART menu.
Appendix C

Debugging

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

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

---

Finding Help on Xilinx.com

To help in the design and debug process when using the HDMI 1.4/2.0 Receiver Subsystem, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

**Documentation**

This product guide is the main document associated with the HDMI 1.4/2.0 Receiver Subsystem. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

**Answer Records**

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

Answer Records for this subsystem can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as...
• Product name
• Tool message(s)
• Summary of the issue encountered

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

Master Answer Record for the HDMI 1.4/2.0 Receiver Subsystem
AR: 54546

Technical Support

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

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

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

Additional support can be found at the Xilinx Video - Community Forums.

---

Debug Tools

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

Vivado Design Suite Debug Feature

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

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

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

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

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

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

Hardware Debug

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

General Checks

- Ensure that all the timing constraints and all other constraints were met during implementation.
- Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the locked port.
- If your outputs go to 0, check your licensing.
  - User LEDs (KC705/KCU105/ZC706/ZCU102/ZCU104/ZCU106/VCU118)
  - LED0 - HDMI TX subsystem lock (when HDMI Pass-Through Example Design is used)
**IMPORTANT:** Not all the HDMI sources are fully compliant. Many times, a source may send unexpected video or auxiliary packets. Therefore, if you observe issues with a particular source, test the same design on others sources to ensure there are no problems with the video information sent from HDMI source.

---

**In-system Debug**

For in-system debugging assistance, relevant information is brought to UART for you to verify before checking the hardware signals.

1. Press 'I' to check the system information.
2. Press 'z' to check the event log.

   Table C-1 shows two examples of system flow event logs.
Appendix C: Debugging

Press 'e' to check the EDID.

In this release, application software will read and parse sink's EDID to know sink's capability. However, current application doesn't block the user to set certain video formats although it is not supported by the sink. Instead, a "warning" message will be prompted to indicating the sink's limitation when user presses 'e'.

<table>
<thead>
<tr>
<th>Table C-1: System Flow Event Log</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>At system start-up</strong></td>
</tr>
<tr>
<td>VPHY log</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>GT init start</td>
</tr>
<tr>
<td>GT init done</td>
</tr>
<tr>
<td>RX frequency event</td>
</tr>
<tr>
<td>RX timer event</td>
</tr>
<tr>
<td>RX DRU disable</td>
</tr>
<tr>
<td>CPLL reconfig done</td>
</tr>
<tr>
<td>GT RX reconfig start</td>
</tr>
<tr>
<td>GT RX reconfig done</td>
</tr>
<tr>
<td>CPLL lock</td>
</tr>
<tr>
<td>RX reset done</td>
</tr>
<tr>
<td>RX MMCM reconfig done</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HDMI RX log</th>
<th>HDMI RX log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initializing HDMI RX core....</td>
<td>Initializing HDMI RX core....</td>
</tr>
<tr>
<td>Initializing HDCP 1.4 core....</td>
<td>Initializing HDCP 1.4 core....</td>
</tr>
<tr>
<td>Initializing AXI Timer core....</td>
<td>Initializing AXI Timer core....</td>
</tr>
<tr>
<td>Initializing HDCP 2.2 core....</td>
<td>Initializing HDCP 2.2 core....</td>
</tr>
<tr>
<td>RX HDCP 1.4 Enabled</td>
<td>RX HDCP 1.4 Enabled</td>
</tr>
<tr>
<td>RX HDCP 2.2 Disabled</td>
<td>RX HDCP 2.2 Disabled</td>
</tr>
<tr>
<td>Reset HDMI RX Subsystem....</td>
<td>Reset HDMI RX Subsystem....</td>
</tr>
<tr>
<td>RX cable is connected....</td>
<td>RX cable is connected....</td>
</tr>
<tr>
<td>RX HDCP 1.4 Enabled</td>
<td>RX HDCP 1.4 Enabled</td>
</tr>
<tr>
<td>RX HDCP 2.2 Disabled</td>
<td>RX HDCP 2.2 Disabled</td>
</tr>
<tr>
<td>RX HDCP 1.4 Enabled</td>
<td>RX HDCP 1.4 Enabled</td>
</tr>
<tr>
<td>RX HDCP 2.2 Disabled</td>
<td>RX HDCP 2.2 Disabled</td>
</tr>
<tr>
<td>RX TMDS reference clock change</td>
<td>RX TMDS reference clock change</td>
</tr>
<tr>
<td>RX Stream Init</td>
<td>RX Stream Init</td>
</tr>
<tr>
<td>RX Stream Start</td>
<td>RX Stream Start</td>
</tr>
<tr>
<td>RX Stream is Up</td>
<td>RX Stream is Up</td>
</tr>
</tbody>
</table>
In this release, for the EDID parsing feature, 3 VERBOSITY level are defined. User may set it according to their requirements. It is accessible from video_common library, in xvidc_cea861.h

```c
#define XVIDC_EDID_VERBOSITY 0
```

where

<table>
<thead>
<tr>
<th>XVIDC_EDID_VERBOSITY</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Default)</td>
<td>Read and parse the EDID</td>
</tr>
<tr>
<td></td>
<td>No display of capability</td>
</tr>
<tr>
<td>1</td>
<td>Read and parse the EDID</td>
</tr>
<tr>
<td></td>
<td>Display of Sink’s basic capability</td>
</tr>
<tr>
<td>2</td>
<td>Read and parse the EDID</td>
</tr>
<tr>
<td></td>
<td>Display of sink’s full capability</td>
</tr>
</tbody>
</table>

**IMPORTANT:** In Example design running on MicroBlaze, limited BRAM resources are allocated to store software binary. If user enables the XVIDC_EDID_VERBOSITY to higher level, which will consume more software resources, user may need to increase the BRAM allocation. Otherwise, you may experience system instability such as UART hangs etc.

3. Press ‘h’ to check HDCP status.

   **Note:** HDCP Debug Menu is not enabled by default. You can enable it by setting it in xhdmi_example.h

   ```c
   /* Enabling this will enable HDCP Debug menu */
   #define HDCP_DEBUG_MENU_EN 1
   
   **Note:** You must check the software size so that it doesn’t exceed the BRAM allocated.

4. 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 can not send out HBR audio. Some sources may even disable the HBR option from the menu.

   The EDID clone feature may 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 a 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 while designing the board, it is possible that the board designer to swap the TMDS polarity in order to make the routing easier. In this case, the Vid_Phy_Controller need to be configured accordingly. When the polarity is wrong, usually the system will have problem detecting incoming video stream, even on those common resolutions (e.g. 1080p60). And the detected timing looks way off the expected values. User should start looking at the data pins for the GT being routed to the FPGA to check whether they have been swapped.

API is available in Vid_Phy_Controller driver to allow user to set the polarity. If user experiences system behavior as described above, they can use this API to change the polarity to confirm whether it is the polarity swapping causing the system issue.

```c
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_arreset` 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 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.
Appendix D

Application Software Development

Device Drivers

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

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

Architecture

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

The subsystem driver consists of the following:

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

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

Figure D-1 shows the HDMI 1.4/2.0 Receiver Subsystem architecture.
The HDMI 1.4/2.0 Receiver Subsystem is a MAC subsystem which works with a Video PHY Controller (PHY) to create a video connectivity system. The HDMI 1.4/2.0 Receiver Subsystem is tightly coupled with the Xilinx Video PHY Controller, which itself is independent and offers flexible architecture with multiple-protocol support. Both MAC and PHY are dynamically programmable through the AXI4-Lite interface.

**Usage**

The HDMI 1.4/2.0 Receiver Subsystem provides a set of API functions for application code to use. On top of that, when HDMI 1.4/2.0 Receiver Subsystem hardware interrupts are generated, the subsystem driver is invoked to configure the system accordingly. HDMI 1.4/2.0 Receiver Subsystem provides callback structure to hook up your own callback functions.
Ensure that the video stream has started. Then, valid AUX data and audio data can be inserted after the video is locked. However, because the application knows what video format will be sent and what audio format will be embedded. With this information, the ACR number can be calculated and set before audio stream is ready to be sent.

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

Note: Because the HDMI 1.4/2.0 Transmitter Subsystem and HDMI 1.4/2.0 Receiver Subsystem have many common features, HDMI Common Library is introduced for this purpose to define common data structures which can be shared by both subsystems.

**HDMI RX Subsystem Flow**

HDMI RX Subsystem in general responses 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 Figure D-3. At execution, the software application initializes the HDMI RX Subsystem IP and registers the callback functions in the provided hooks.
Application Integration

Figure D-4 shows an example code on how an HDMI 1.4/2.0 Receiver Subsystem can be used in your application.
To integrate and use the HDMI 1.4/2.0 Receiver 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:

   ```c
   XV_HdmiRxSs HdmiRxSs;
   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 \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);
   }
```

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:

   ```c
   XV_HdmiRxSs_Config *XV_HdmiRxSs_ConfigPtr;
   }
   
   1. Set EDID parameter for HDMI 1.4/2.0 Receiver Subsystem Subsystem.

   ```c
   void XV_HdmiRxSs_SetEdidParam(XV_HdmiRxSs *InstancePtr,
                                  u8 *EdidDataPtr,
                                  
   Figure D-4: Application Example Code

   To integrate and use the HDMI 1.4/2.0 Receiver 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:

   ```c
   XV_HdmiRxSs HdmiRxSs;
   
   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 \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);
   ```
5. For each subsystem instance, the data structures declared in steps 2 and 3 need to be initialized based on its hardware configuration, which is passed through metadata structure from xparameters.h uniquely identified by device ID.

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

```c
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.

```c
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 mentioned API function.

2. The EDID data is loaded into the HDMI 1.4/2.0 Receiver Subsystem during initialization. This is handled by the subsystem driver, no user intervention is needed.

### HDCP RX Overview

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

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

   o 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)

   IMPORTANT: In a 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 FPGA is reprogrammed. Therefore, the keys can only be read ONCE during initialization.

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

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

   o XV_HdmiRxSS_HdcpIntrHandler
   o XV_HdmiRxSS_HdcpTimerIntrHandler

4. Set the HDCP user callback functions. These callbacks are optional and can be used as a hook to take action during various parts of the HDCP protocol. If there is no use for a callback at the application level, they can be left undefined.

   o XV_HdmiRxSs_SetCallback
     - XV_HDMIRXSS_HANDLER_HDCP_AUTHENTICATED
     - XV_HDMIRXSS_HANDLER_HDCP_UNAUTHENTICATED
     - XV_HDMIRXSS_HANDLER_HDCP_AUTHENTICATION_REQUEST
     - XV_HDMIRXSS_HANDLER_HDCP_STREAM_MANAGE_REQUEST
     - XV_HDMIRXSS_HANDLER_HDCP_TOPOLOGY_UPDATE
     - XV_HDMIRXSS_HANDLER_HDCP_ENCRYPTION_UPDATE

<table>
<thead>
<tr>
<th>Position in Bytes</th>
<th>Size in Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–39</td>
<td>40</td>
<td>Reserved</td>
</tr>
<tr>
<td>40–561</td>
<td>522</td>
<td>Device Public Certificate</td>
</tr>
<tr>
<td>562–901</td>
<td>340</td>
<td>Device Private Key(1)</td>
</tr>
</tbody>
</table>

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.

Position in Bytes Size in Bytes Description
0–39 40 Reserved
40–561 522 Device Public Certificate
562–901 340 Device Private Key(1)
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. Protocol enablement is automatically handled by the subsystem, thus the legacy XV_HdmiRxSs_HdcpSetProtocol API is deprecated. Anytime the capability has 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
     - XV_HDMIRXSS_HDCP_22

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 Driver for HDMI RX Subsystem Usage**

Because the HDMI 1.4/2.0 Receiver Subsystem is closely coupled with the Video PHY Controller, the following example code demonstrates how a Video PHY Controller can be used in your application.
To integrate and use the Video PHY Controller for HDMI 1.4/2.0 Receiver Subsystem in the application code, the following steps must be followed:

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

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

   Example:
   ```c
   XVphy Vphy;
   ```

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

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

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

```c
XVphy_Config *XVphy_LookupConfig(u16 DeviceId);
uint32_t XVphy_HdmiInitialize(XVphy *InstancePtr,
                               u8 QuadId,
                               XVphy_Config *CfgPtr,
                               uint32_t SystemFrequency);
```

The Device ID can be found in xparameters.h:

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

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

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

### Interrupts

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

1. **HPD** – Peripheral I/O to detect HDMI cable 5.0V signal
   - **Rising edge** – Cable connected
   - **Falling edge** – Cable disconnected

2. **Link Ready** – Every time when Video PHY Controller is reconfigured, the \texttt{link\_clk} is regenerated. An HDMI RX sub-core register bit (link status bit) reflects the change of \texttt{link\_clk} status. When stable \texttt{link\_clk} is detected, it is set to 1. When \texttt{link\_clk} becomes unstable, it is set to 0. The Link Ready is an interrupt to detect the change of the link status bit.
   - **Rising edge** – Link is up
   - **Falling edge** – Link is down

3. **Video Ready** – This interrupt is generated by HDMI RX sub-core to reflect the status of received video stream.
   - **Rising edge** – Video Stream is stable (StreamUp)
   - **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. HDCP 1.4 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 D-1: Mapping between Interrupt Sources and Application Callback Functions |
|---|---|
| Interrupts | Callback |
| HPD | XV_HDMIRXSS_HANDLER_CONNECT |
| Video Ready | <br> XV_HDMIRXSS_HANDLER_STREAM_UP <br> XV_HDMIRXSS_HANDLER_STREAM_DOWN |
| Note: It is edge triggered. | Video Ready rising edge: Stream Up <br> Video Ready falling edge: Stream Down |
| 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 2.2 Timer Interrupt (only available when HDCP 2.2 is enabled in hardware) | XV_HDMIRXSS_HANDLER_HDCP_AUTHENTICATE |
| Note: This callback function is not directly mapped to any interrupt source. Instead it is executed when the HDCP authentication state machine has reached the authenticated state. |

**Application Callback Functions**

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

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

```c
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_HDCP_AUTHENTICATE

**XV_HDMIRXSS_HANDLER_CONNECT**

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

The callback function needs to perform the following:

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

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

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

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

**XV_HDMIRXSS_HANDLER_AUX**

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

The callback function needs to retrieve the InfoFrame packet data for system application to use.

**XV_HDMIRXSS_HANDLER_AUD**

This interrupt is triggered every time when 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 when 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 turns 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 Receiver 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 is stabilized for the HDMI 1.4/2.0 Receiver Subsystem to start stream locking.

The callback function needs to 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 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 to start 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:

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

**XV_HDMIRXSS_HANDLER_HDCP_AUTHENTICATE**

If HDCP 1.4 or HDCP 2.2 is enabled in the HDMI 1.4/2.0 Receiver 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 Interrupt Handlers for HDMI 1.4/2.0 Receiver Subsystem**

There are several interrupt handlers available in the Video PHY Controller driver to hook up with user-defined callback functions to support HDMI 1.4/2.0 Receiver 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.

```c
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 detects an HDMI RX reference clock changes.

The callback function needs to perform the following:

1. Initialize a reference clock change process for an HDMI 1.4/2.0 Receiver Subsystem.
   ```c
   void XV_HdmiRxSs_RefClockChangeInit(XV_HdmiRxSs *InstancePtr);
   ```
2. Set Video PHY Controller HDMI RX reference TMDS clock ratio.
   ```c
   VphyPtr->HdmiRxTmdsClockRatio = HdmiRxSs.TMDSClockRatio;
   ```

**XVPHY_HDMI_HANDLER_RXREADY**

This interrupt is triggered every time the Video PHY Controller RX reset lock is done.
The callback function needs to perform the following:

1. Check the Video PHY Controller PLL Type.

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

2. Set the HDMI 1.4/2.0 Receiver Subsystem Video Stream according to the PLL type.

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

   Where both Clock and LineRate are from Video PHY Controller data structure.

Follow the steps in Chapter 5, Example Design to create an example design, which contains all the procedures implemented and can serve as a reference for integrating the HDMI 1.4/2.0 Receiver Subsystem into your system.

**Example Use Cases**

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

**Use Case 1: Cable Plug In**

HPD interrupt is received indicating Cable Connection.

- Callback function registered to `XV_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 RX Init interrupt is received.

   - Callback function registered to `XVPHY_HDMI_HANDLER_RXINIT` Interrupt type is called.

2. Video PHY Controller 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_STREAM_INIT` Interrupt 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:

```c
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 RX Init interrupt is received.
   - Callback function registered to `XVPHY_HDMI_HANDLER_RXINIT` Interrupt type is called.

3. Video PHY Controller 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:

```c
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`, under function call.

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

Example:

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

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

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

- In the Xilinx Documentation Navigator, click the Design Hubs View tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.
Appendix E: Additional Resources and Legal Notices

References

These documents provide supplemental material useful with this product guide:

1. Xilinx Vivado AXI Reference Guide (UG1037)
2. Kintex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics (DS892)
3. Virtex UltraScale FPGAs Data Sheet: DC and AC Switching Characteristics (DS893)
4. Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics (DS182)
5. Virtex-7 FPGAs Data Sheet: DC and AC Switching Characteristics (DS183)
6. Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics (DS181)
7. Zynq-7000 SoC: DC and AC Switching Characteristics (DS187)
8. Zynq-7000 SoC: DC and AC Switching Characteristics (DS191)
9. Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics (DS922)
10. Virtex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics (DS923)
11. Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics (DS925)
12. Zynq UltraScale+ RFSoC Data Sheet: DC and AC Switching Characteristics (DS926)
13. HDMI specifications (www.hdmi.org/manufacturer/specification.aspx)
14. HDCP specifications (www.digital-cp.com/hdcp-specifications)
15. AXI4-Stream Video IP and System Design Guide (UG934)
19. ISE to Vivado Design Suite Migration Guide (UG911)
20. KCU105 Board User Guide (UG917)
23. AXI Interconnect LogiCORE IP Product Guide (PG059)
25. HDCP v2.2 Product Guide (PG249)
26. HDCP v1.4 Product Guide (PG224)
27. Video In to AXI4-Stream LogiCORE IP Product Guide (PG043)
## Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>02/12/2019</td>
<td>3.1</td>
<td>• Added HDCP Repeater functionality</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added PCB design guidelines for TMDS181 and DP159</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Example Design steps</td>
</tr>
<tr>
<td>04/04/2018</td>
<td>3.1</td>
<td>• Added VCU118 Example design info and steps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added new board supports (ZCU104, ZCU106, VCU118).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added new Example design features.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Example Design steps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Migrating and upgrading section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added debug information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated AXI-Lite CPU clock supports.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated YUV420 remapping feature illustration.</td>
</tr>
<tr>
<td>12/20/2017</td>
<td>3.0</td>
<td>• Updated Example Design steps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added notes for DP159 Settings.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated notes for CPU clock requirements.</td>
</tr>
<tr>
<td>10/04/2017</td>
<td>3.0</td>
<td>• Added Example design topology supports (RX-Only, Pass-through).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Example design VPHY Configuration Support (NI-DRU Enable/Disable, TXPLL selection, RXPLL selection).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Example design new board supports (ZCU102).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added SDK application supports (RX, Pass-through and HDCP key utility)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Information for Example design description.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Software Flow diagram.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Information if not all audio channels are used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Information about Native Video.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Information about Interlaced Video.</td>
</tr>
<tr>
<td>04/05/2017</td>
<td>2.0</td>
<td>• Removed single pixel per clock support</td>
</tr>
<tr>
<td>11/30/2016</td>
<td>2.0</td>
<td>• Added example design migration notes.</td>
</tr>
<tr>
<td>10/05/2016</td>
<td>2.0</td>
<td>• Added example design flow.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added HPD XGUI option.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added software use cases.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Xilinx <strong>AUTOMOTIVE APPLICATIONS DISCLAIMER.</strong></td>
</tr>
<tr>
<td>06/08/2016</td>
<td>2.0</td>
<td>• Updated optional video over AXI-Stream support.</td>
</tr>
</tbody>
</table>
Please Read: Important Legal Notices

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available “AS IS” and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx’s limited warranty, please refer to Xilinx’s Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx’s Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

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

© Copyright 2015–2019 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.