DisplayPort 1.4 TX Subsystem v1.0

Product Guide

Vivado Design Suite

PG299 April 4, 2018

Table of Contents

Introduction

DisplayPort 1.4 TX Subsystem implements functionality of a video source as defined by the Video Electronics Standards Association (VESA)'s DisplayPort standard v1.4 and supports driving resolutions of up to Full Ultra HD (FUHD) at 30 fps. The Xilinx® DisplayPort subsystems provide highly integrated but straightforward IP blocks requiring very little customization by the user.

Features

- Support for DisplayPort Source (TX) transmissions.
- Supports single stream transport (SST) at FUHD at 30 fps
- Dynamic lane supports (1, 2, or 4 lanes)
- Dynamic link rate support (1.62/2.7/5.4/8.1 Gb/s)
- Dynamic support of 6, 8, 10, 12, or 16 bits per component (BPC).
- Dynamic support of RGB/YCbCr444/ YCbCr422 color formats.
- Supports 16-bit Video PHY (GT) Interface
- Supports 2 to 8 channel Audio.
- Supports native or AXI4-Stream video input interface.
- Supports SDP packet for static HDR mode.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	UltraScale+™ Families (GTHE4) UltraScale™ Families (GTHE3)
Supported User Interfaces	AXI4-Stream, AXI4-Lite, Native video
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Hierarchical subsystem packaged with DisplayPort TX core and other IP cores
Example Design	Vivado IP Integrator
Test Bench	N/A
Constraints File	IP cores delivered with XDC files
Simulation Model	N/A
Supported S/W Driver	Standalone
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

This chapter contains an overview of the core as well as details about features, licensing, and standards. The DisplayPort 1.4 TX Subsystem is a full feature, hierarchically packaged subsystem with a DisplayPort Transmit (TX) core ready to use in applications in large video systems.

RECOMMENDED: *Xilinx recommends a redriver for the TX subsystem solution.*

Table 1-1 shows the UltraScale™ and UltraScale+™ families core support.

Table 1-1: **Core Support**

Features	UltraScale (GTHE3)	UltraScale+ (GTHE4)
DisplayPort 1.4 – 8.1 Gb/s (without HDCP)	Yes(1)	Yes
DisplayPort 1.4 – 8.1 Gb/s (with HDCP)	Roadmap(2)	Roadmap(3)

Notes:

1. You need to try different Implementation Strategies/pblocks to meet timing at 8.1 Gb/s. To learn more about Timing Closure Techniques, see the *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949) [Ref 2].
2. This feature is limited to High Bit Rate 2 (HBR2).
3. This feature is supported in a future release. For more information, contact Xilinx Support.

Feature Summary

- Single stream transport (SST) mode.
- Dynamic support of different bits per color (6, 8, 10, 12 or 16) and line rates.
- Dynamic support of RGB/YCbCr444/ YCbCr422 color formats.
- Support for native and AXI4-Stream video input interface.

Unsupported Features

- In-band stereo is not supported.
- Video AXI4-Stream interface is not scalable with dynamic pixel mode selection.
- Dual-pixel splitter is not supported in native video mode.
- MST is not supported.
- HDCP 1.3/2.2 are not supported.
- eDP and iDP are not supported.
- GTC is not supported.
- Non-LPCM audio is not supported.
- DSC + FEC
- 16/32 channel Audio

Licensing and Ordering

License Type

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

If you have a current license for EF-DI-DISPLAYPORT, you do not need a new license for the DisplayPort 1.4 TX Subsystem.

For more information about licensing for the core, see the DisplayPort product 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.

TIP: To verify that you need a license, check the "License" column of the IP Catalog. "Included" means that a license is included with the Vivado Design Suite; "Purchase" means that you have to purchase a license to use the core or subsystem.

Note: This uses the same license for DisplayPort 1.2 and DisplayPort 1.4 IPs, but the license file (.lic) needs to be regenerated.

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.*

Product Specification

This chapter contains a high-level overview of the core as well as performance and port details.

Overview

The DisplayPort 1.4 TX Subsystem, in both AXI4-Stream and native interface, operates in the single stream transport (SST) video mode.

AXI4-Stream Video Interface

When configured with the AXI4-Stream interface, the subsystem is packaged with three subcores: DisplayPort Transmitter core, Video Timing Controller (VTC) and DP AXI4-Stream to Video Bridge. Because the DisplayPort 1.4 TX Subsystem is hierarchically packaged, you select the parameters and the subsystem creates the required hardware. Figure 2-1 shows the architecture of the subsystem assuming SST with a single stream.

The subsystem includes a multi-pixel AXI4-Stream Video Protocol interface. The DisplayPort 1.4 TX Subsystem outputs the video using the DisplayPort v1.4 protocol. The DisplayPort 1.4 TX Subsystem works in conjunction with the *Video PHY Controller Product Guide* (PG230) [Ref 1] configured for the DP protocol.

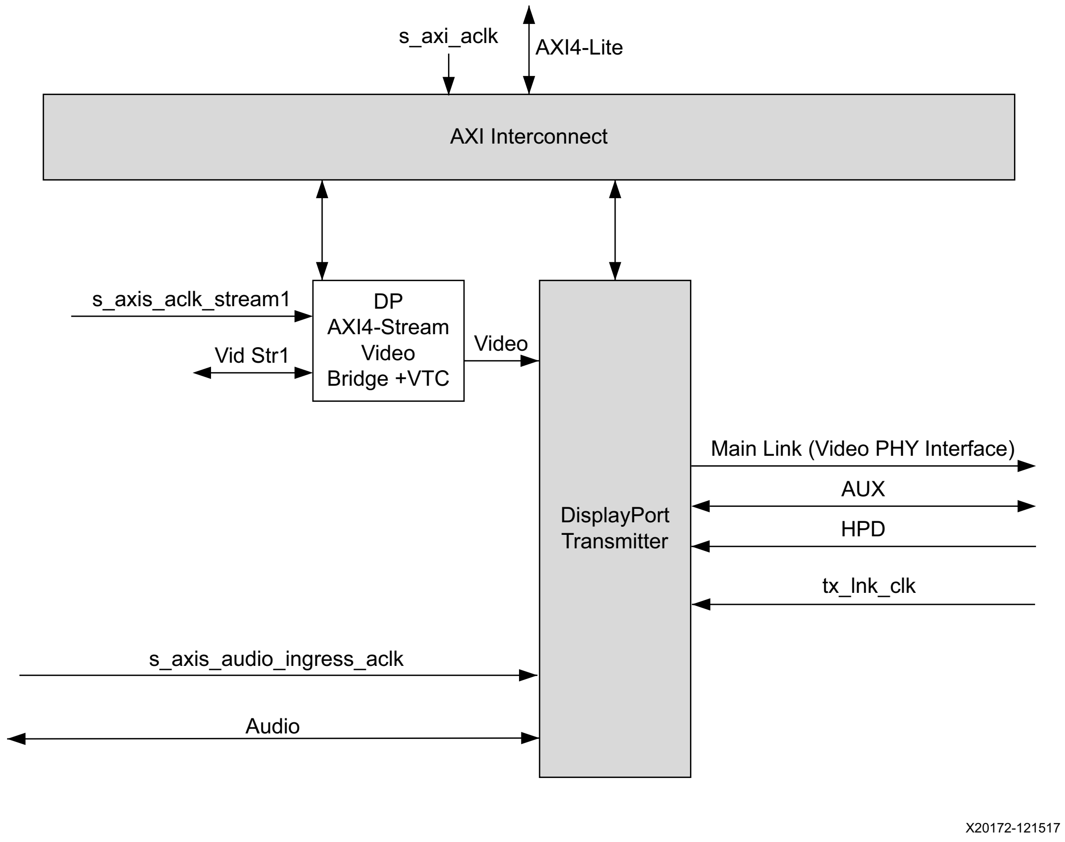


Figure 2-1: **DisplayPort 1.4 TX Subsystem AXI4-Stream Video Interface Block Diagram**

Pixel Mapping on AXI4-Stream Interface

By default, the pixel mode is selected based on Pixel Frequency in the subsystem driver. The following shows the different Pixel per Clock (PPC) for each Pixel Frequency:

- For 1 PPC, lane count is 1.
- For 2 PPC, lane count is 2.
- For 4 PPC, lane count is 4.

Also, you can override pixel width dynamically. For example, if the driver selects a 2 pixel mode as default, you can change the pixel mode to 1.

- For pixel mode of 1, valid pixels are available only in pixel 0 position.
- For pixel mode of 2, valid pixels are available only in pixel 0 and pixel 1 position.
- For pixel mode of 4, valid pixels are available only in pixel 0, pixel 1, pixel 2, and pixel 3 position.

The data width of the AXI4-Stream interface depends on different parameters of the core.

```
Pixel_Width = MAX_BPC × 3
```

```
Interface Width = Pixel Width × LANE_COUNT
```

For example, if the system is generated using 4 lanes with `MAX_BPC` of 16, the data width of the AXI4-Stream interface is 16 × 4 × 3 which equals to 192.

Table 2-1 shows the pixel mapping examples for an AXI4-Stream interface.

Table 2-1: **Pixel Mapping Examples on AXI4-Stream Interface**

MAX _BPC	LANES	Pixel Width	Interface Width	Video BPC	Pixel 3			Pixel 2			Pixel 1			Pixel 0		
16	4	48	192	16	191:176	175:160	159:144	143:128	127:112	111:96	95:80	79:64	63:48	47:32	31:16	15:0
16	2	48	96	16	–	–	–	–	–	–	95:80	79:64	63:48	47:32	31:16	15:0
16	1	48	48	16	–	–	–	–	–	–	–	–	–	47:32	31:16	15:0
12	4	36	144	12	143:132	131:120	119:108	107:96	95:84	83:72	71:60	59:48	47:36	35:24	23:12	11:0
12	2	36	72	12	–	–	–	–	–	–	71:60	59:48	47:36	35:24	23:12	11:0
12	1	36	40	12	–	–	–	–	–	–	–	–	–	35:24	23:12	11:0
10	4	30	120	10	119:110	109:100	99:90	89:80	79:70	69:60	59:50	49:40	39:30	29:20	19:10	9:0
10	2	30	64	10	–	–	–	–	–	–	59:50	49:40	39:30	29:20	19:10	9:0
10	1	30	32	10	–	–	–	–	–	–	–	–	–	29:20	19:10	9:0
8	4	24	96	8	95:88	87:80	79:72	71:64	63:56	55:48	47:40	39:32	31:24	23:16	15:8	7:0
8	2	24	48	8	–	–	–	–	–	–	47:40	39:32	31:24	23:16	15:8	7:0
8	1	24	24	8	–	–	–	–	–	–	–	–	–	23:16	15:8	7:0
16	4	48	192	12	191:180	175:164	159:148	143:132	127:116	111:100	95:84	79:68	63:52	47:36	31:20	15:4
16	2	48	96	12	–	–	–	–	–	–	95:84	79:68	63:52	47:36	31:20	15:4
16	1	48	48	12	–	–	–	–	–	–	–	–	–	47:36	31:20	15:4
12	4	36	144	10	143:134	131:122	119:110	107:98	95:86	83:74	71:62	59:50	47:38	35:26	23:14	11:2
12	2	36	72	10	–	–	–	–	–	–	71:62	59:50	47:38	35:26	23:14	11:2
12	1	36	40	10	–	–	–	–	–	–	–	–	–	35:26	23:14	11:2
10	4	30	120	8	119:112	109:102	99:92	89:82	79:72	69:62	59:52	49:42	39:32	29:22	19:12	9:2
10	2	30	64	8	–	–	–	–	–	–	59:52	49:42	39:32	29:22	19:12	9:2
10	1	30	32	8	–	–	–	–	–	–	–	–	–	29:22	19:12	9:2
8	4	24	96	6	95:90	87:82	79:74	71:66	63:58	55:50	47:42	39:34	31:26	23:18	15:10	7:2
8	2	24	48	6	–	–	–	–	–	–	47:42	39:34	31:26	23:18	15:10	7:2
8	1	24	24	6	–	–	–	–	–	–	–	–	–	23:18	15:10	7:2

Table 2-1: **Pixel Mapping Examples on AXI4-Stream Interface *(Cont'd)***

MAX _BPC	LANES	Pixel Width	Interface Width	Video BPC	Pixel 3			Pixel 2			Pixel 1			Pixel 0		
16	4	48	192	10	191:182	175:166	159:150	143:134	127:118	111:102	95:86	79:70	63:54	47:38	31:22	15:6
16	2	48	96	10	–	–	–	–	–	–	95:86	79:70	63:54	47:38	31:22	15:6
16	1	48	48	10	–	–	–	–	–	–	–	–	–	47:38	31:22	15:6
12	4	36	144	8	143:136	131:124	119:112	107:100	95:88	83:76	71:64	59:52	47:40	35:28	23:16	11:4
12	2	36	72	8	–	–	–	–	–	–	71:64	59:52	47:40	35:28	23:16	11:4
12	1	36	40	8	–	–	–	–	–	–	–	–	–	35:28	23:16	11:4
10	4	30	120	6	119:114	109:104	99:94	89:84	79:74	69:64	59:54	49:44	39:34	29:24	19:14	9:4
10	2	30	64	6	–	–	–	–	–	–	59:54	49:44	39:34	29:24	19:14	9:4
10	1	30	32	6	–	–	–	–	–	–	–	–	–	29:24	19:14	9:4
16	4	48	192	8	191:184	175:168	159:152	143:136	127:120	111:104	95:88	79:72	63:56	47:40	31:24	15:8
16	2	48	96	8	–	–	–	–	–	–	95:88	79:72	63:56	47:40	31:24	15:8
16	1	48	48	8	–	–	–	–	–	–	–	–	–	47:40	31:24	15:8
12	4	36	144	6	143:138	131:126	119:114	107:102	95:90	83:78	71:66	59:54	47:42	35:30	23:18	11:6
12	2	36	72	6	–	–	–	–	–	–	71:66	59:54	47:42	35:30	23:18	11:6
12	1	36	36	6	–	–	–	–	–	–	–	–	–	35:30	23:18	11:6
16	4	48	192	6	191:186	175:170	159:154	143:138	127:122	111:106	95:90	79:74	63:58	47:42	31:26	15:10
16	2	48	96	6	–	–	–	–	–	–	95:90	79:74	63:58	47:42	31:26	15:10
16	1	48	48	6	–	–	–	–	–	–	–	–	–	47:42	31:26	15:10

Notes:

1. The padding bits are zeros.

Native Video Interface

With the Native interface enabled, the subsystem is by default packaged with only one subcore: DisplayPort Transmit core. Figure 2-2 shows the architecture of the subsystem assuming SST with a single native video stream. The subsystem includes a multi-pixel Native Video Protocol interface. The DisplayPort 1.4 TX subsystem outputs the video using the DisplayPort v1.4 protocol, works in conjunction with Video PHY Controller configured for the DP protocol.

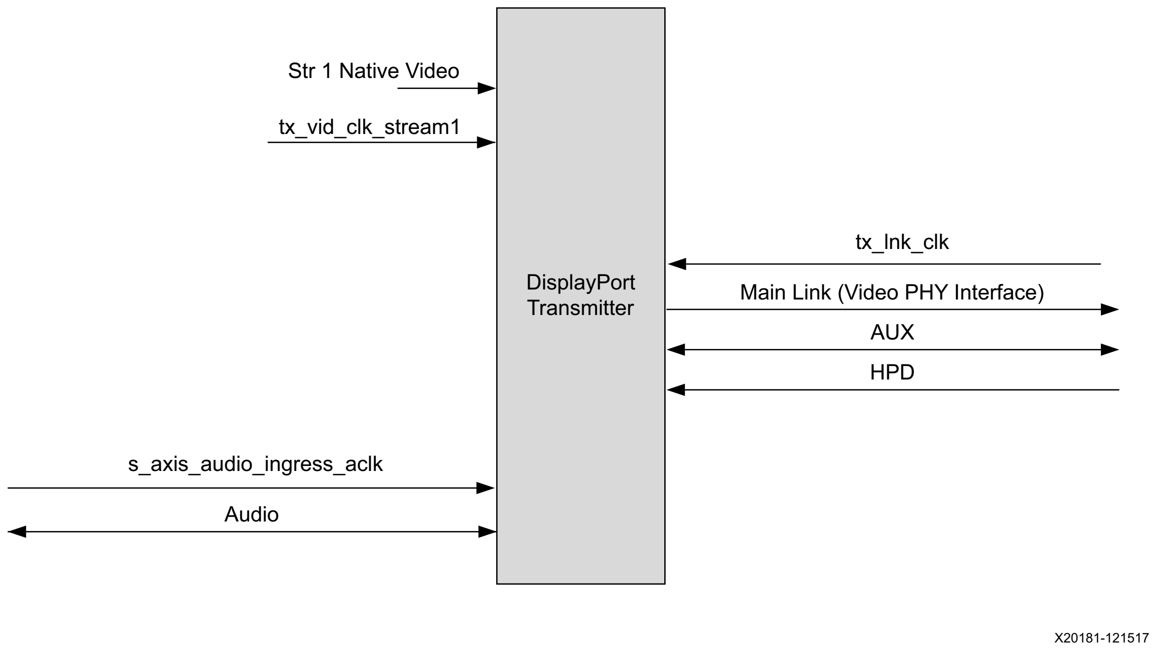


Figure 2-2: **DisplayPort 1.4 TX Subsystem Native Video Block Diagram**

Pixel Mapping on Native Interface

The primary interface for user image data has been modeled on the industry standard for display timing controller signals. The port list consists of video timing information encoded in a vertical and horizontal sync pulse and data valid indicator. These single bit control lines frame the active data and provide flow control for the AXI4-Stream video.

Vertical timing is framed using the vertical sync pulse which indicates the end of frame N-1 and the beginning of frame N. The vertical back porch is defined as the number of horizontal sync pulses between the end of the vertical sync pulse and the first line containing active pixel data. The vertical front porch is defined as the number of horizontal sync pulses between the last line of active pixel data and the start of the vertical sync pulse. When combined with the vertical back porch and the vertical sync pulse width, these parameters form what is commonly known as the vertical blanking interval.

At the trailing edge of each vertical sync pulse, the user data interface resets key elements of the image datapath. This provides for a robust user interface that recovers from any kind of interface error in one vertical interval or less.

Figure 2-3 shows the typical signaling of a full frame of data.

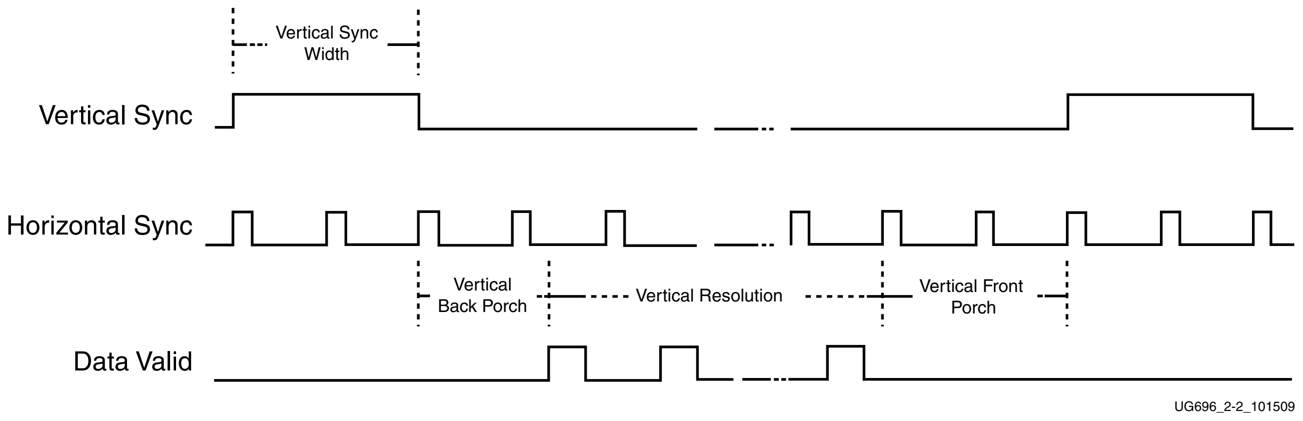


Figure 2-3: **User Interface Vertical Timing**

Similarly, the horizontal timing information is defined by a front porch, back porch, and pulse width. The porch values are defined as the number of clocks between the horizontal sync pulse and the start or end of active data. Pixel data is only accepted into the image data interface when the data valid flag is active-High, as shown in Figure 2-4.

Note that the data valid signal must remain asserted for the duration of a scan line. Dropping the valid signal might result in improper operation.

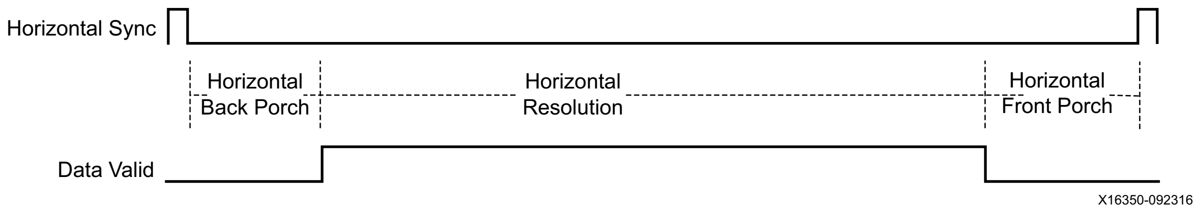


Figure 2-4: **User Interface Horizontal Timing**

In the two-dimensional image plane, these control signals frame a rectangular region of active pixel data within the total frame size. This relationship of the total frame size to the active frame size is shown in Figure 2-5.

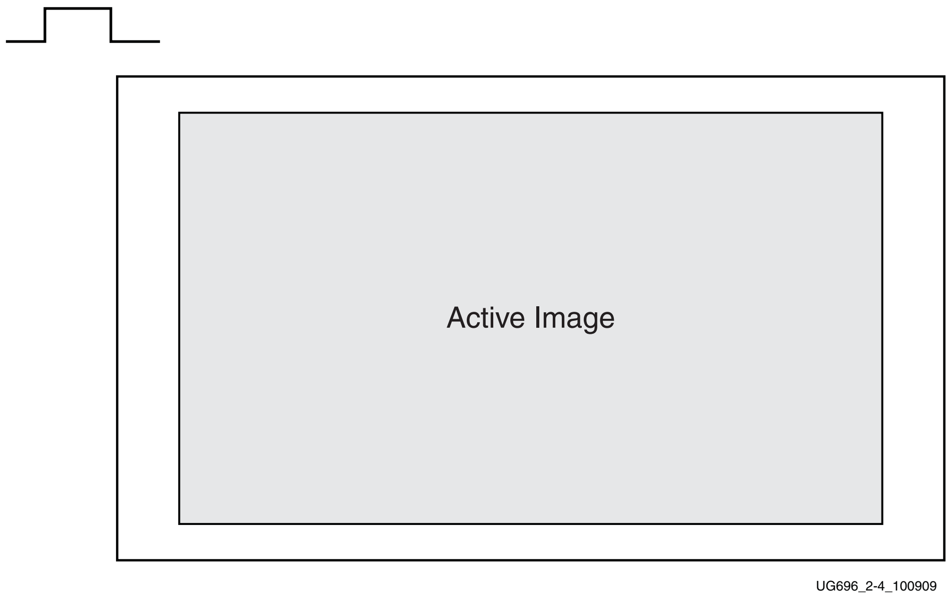


Figure 2-5: **Active Image Data**

The User Data Interface can accept one, two, or four pixels per clock cycle. The vid_pixel width is always 48 bits, regardless of if all bits are used. For pixel mappings that do not require all 48 bits, the convention used for this core is to occupy the MSB bits first and leave the lower bits either untied or driven to zero. Table 2-2 provides the proper mapping for all supported data formats.

Table 2-2: **Pixel Mapping for the User Data Interface**

Format	BPC/BPP	R	G	B	Cr	Y	Cb	Cr/Cb	Y
RGB	6/18	[47:42]	[31:26]	[15:10]	–	–	–	–	–
RGB	8/24	[47:40]	[31:24]	[15:8]	–	–	–	–	–
RGB	10/30	[47:38]	[31:22]	[15:6]	–	–	–	–	–
RGB	12/36	[47:36]	[31:20]	[15:4]	–	–	–	–	–
RGB	16/48	[47:32]	[31:16]	[15:0]	–	–	–	–	–
YCrCb444	6/18	–	–	–	[47:42]	[31:26]	[15:10]	–	–
YCrCb444	8/24	–	–	–	[47:40]	[31:24]	[15:8]	–	–
YCrCb444	10/30	–	–	–	[47:38]	[31:22]	[15:6]	–	–
YCrCb444	12/36	–	–	–	[47:36]	[31:20]	[15:4]	–	–
YCrCb444	16/48	–	–	–	[47:32]	[31:16]	[15:0]	–	–
YCrCb422	8/16	–	–	–	–	–	–	[47:40]	[31:24]
YCrCb422	10/20	–	–	–	–	–	–	[47:38]	[31:22]

Table 2-2: **Pixel Mapping for the User Data Interface *(Cont'd)***

Format	BPC/BPP	R	G	B	Cr	Y	Cb	Cr/Cb	Y
YCrCb422	12/24	–	–	–	–	–	–	[47:36]	[31:20]
YCrCb422	16/32	–	–	–	–	–	–	[47:32]	[31:16]
YONLY	8/8	–	–	–	–	–	–	–	[47:40]
YONLY	10/10	–	–	–	–	–	–	–	[47:38]
YONLY	12/12	–	–	–	–	–	–	–	[47:36]
YONLY	16/16	–	–	–	–	–	–	–	[47:32]

Notes:

1. For a YCrCb 4:2:2, the input follows YCr, YCb, YCr, YCb and so on. This means Cr and Cb are mapped to the same bits on the video input ports of the source core. The source core expects YCb first, followed by YCr.

Selecting the Pixel Interface

To determine the necessary pixel interface to support a specific resolution, it is important to know the active resolution and blanking information.

Note: In a quad pixel interface, if the resolution is not divisible by 4, you should add zeros at the end of frame, over the video interface pixel data.

For example:

To support an active resolution of 2560 x 1600 @ 60, there are two possible blanking formats: Normal Blanking and Reduced Blanking, as defied by the VESA standard.

2560 x 1600 @ 60 + Blanking = 3504 x 1658 @ 60

Requires a Pixel clock of 348.58 MHz

2560 x 1600 @ 60 + Reduced Blanking = 2720 x 1646 @ 60

Requires a Pixel clock of 268.63 MHz

Assuming a pixel clock of 150 MHz and a dual Pixel interface:

2560 x 1600 @ 60 + Blanking = 3504 x 1658 @ 60 = 348.58 MHz

348.58 MHz / 2 = 172.28 MHz

2560 x 1600 @ 60 + Reduced Blanking = 2720 x 1646 @ 60 = 268.63 MHz

268.63 MHz / 2 = 134.31 MHz

With a dual Pixel interface, the DisplayPort IP can support 2560 x 1600 only if there is a Reduced Blanking input. If full Blanking support is needed, then a 4 Pixel interface should be used.

Figure 2-6, to Figure 2-8 show timing diagrams for the three Pixel interface options.

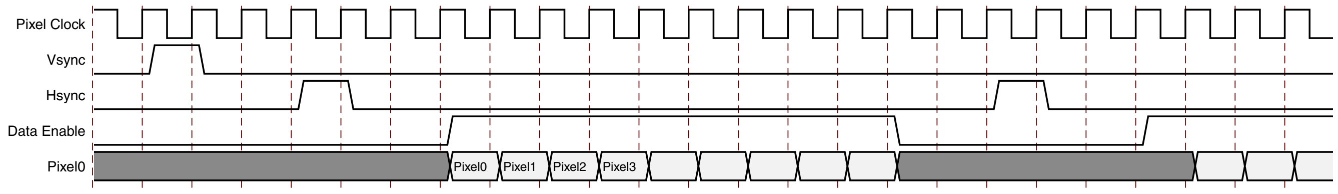


Figure 2-6: **Single Pixel Timing**

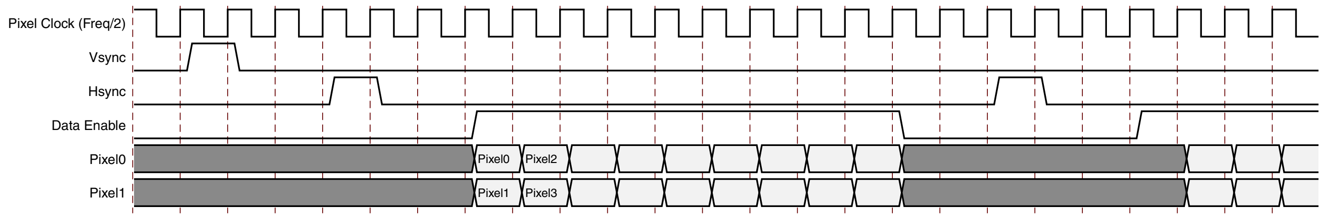


Figure 2-7: **Dual Pixel Timing**

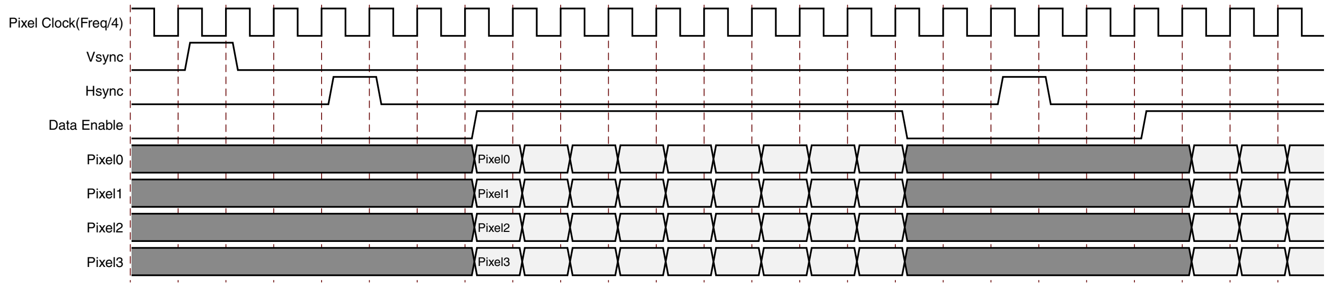


Figure 2-8: **Quad Pixel Timing**

DisplayPort AXI4-Stream to Video Bridge

The DisplayPort AXI4-Stream to Video Bridge maps the video over the AXI4-Stream interface to native video format as required by the DisplayPort Transmitter IP core. The bridge uses the Xilinx AXI4 to Video Out core to convert the format between AXI4-Stream to DisplayPort native video.

Table 2-3 shows the color mapping for the AXI4-Stream interface.

Table 2-3: **AXI4-Stream Interface Data Mapping**

	AXI4-Stream Interface											
	Pixel 3			Pixel 2			Pixel 1			Pixel 0		
	Comp3	Comp2	Comp1	Comp3	Comp2	Comp1	Comp3	Comp2	Comp1	Comp3	Comp2	Comp1
RGB	R	G	B	R	B	G	R	B	G	R	B	G
YCbCr444	Cr	Y	Cb	Cr	Cb	Y	Cr	Cb	Y	Cr	Cb	Y
YCbCr422	Cr/Cb	Y	–	Cr/Cb	Y	–	Cr/Cb	Y	–	Cr/Cb	Y	–
Y-Only	Y	–	–	Y	–	–	Y	–	–	Y	–	–

Notes:
1. For component widths, see Table 2-1.

For details about the Video Out Bridge, see the *AXI4-Stream to Video Out Product Guide* (PG044) [Ref 12]. For details about the video over AXI4-Stream, see the *AXI Reference Guide* (UG1037) [Ref 11]. The receive side of the bridge is Video over AXI4-Stream. For more details, see Port Descriptions.

Video Timing Controller

The Xilinx Video Timing Controller is used for generation of video timing. Video Timing Controller is required when the subsystem is configured in AXI4-Stream interface mode. For details on this core, see the *Video Timing Controller Product Guide* (PG016) [Ref 13].

IMPORTANT: *You must program proper front porch and back porch blanking period generation.*

DisplayPort Transmit

The DisplayPort Transmit core contains the following components as shown in Figure 2-9:

- **Main Link**: Provides delivery of the primary video stream.
- **Secondary Link**: Integrates the delivery of audio information into the Main Link blanking period.
- **AUX Channel**: Establishes the dedicated source to sink communication channel.

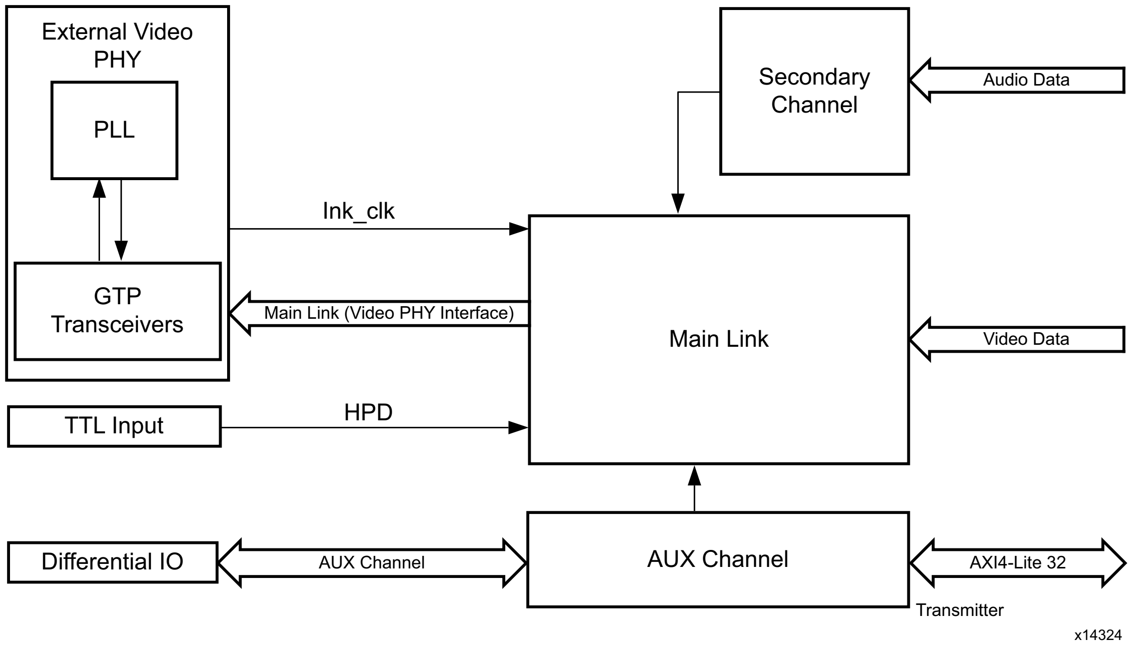


Figure 2-9: **DisplayPort Transmit Core Block Diagram**

AXI Interconnect

The subsystem uses Xilinx AXI Interconnect IP core, as a crossbar which contains an AXI4-Lite interface. Figure 2-10 shows the AXI slave structure within the DisplayPort 1.4 TX Subsystem.

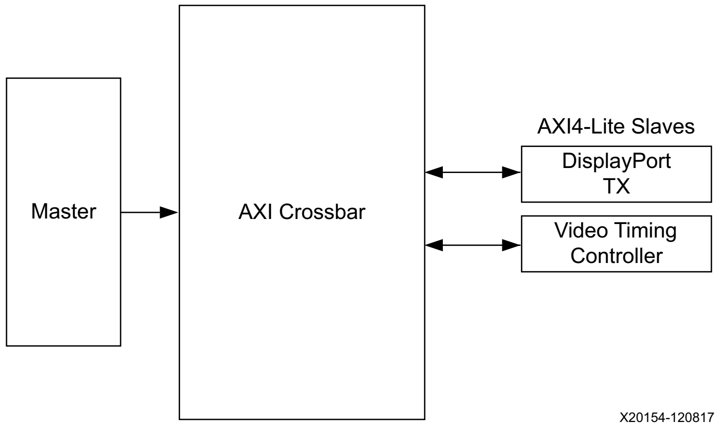


Figure 2-10: **AXI4-Lite Interconnect within DisplayPort 1.4 TX Subsystem**

Note: Video Timing Controller and Dual splitter are present only when subsystem is generated in AXI4-Stream interface mode.

Standards

The DisplayPort 1.4 TX Subsystem is compatible with the DisplayPort v1.4 standard as well as the AXI4-Lite, and AXI4-Stream interfaces.

IMPORTANT: *Xilinx DisplayPort subsystems have passed compliance certification. If you are interested in accessing the compliance report or seeking guidance for the compliance certification of your products, contact your local Xilinx sales representative.*

Resource Utilization

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

Port Descriptions

The DisplayPort 1.4 TX Subsystem ports are described in Table 2-4.

Table 2-4: **DisplayPort 1.4 TX Subsystem Ports**

Signal Name	Direction	Width	Description
AXI4-Lite Interface			
s_axi_aclk	Input	1	AXI Bus Clock.
s_axi_aresetn	Input	1	AXI Reset. Active-Low.
s_axi_awadd	Input	19	Write Address
s_axi_awpro	Input	3	Protection type.
s_axi_awvalid	Input	1	Write address valid.
s_axi_awready	Output	1	Write address ready.
s_axi_wdata	Input	32	Write data bus.
s_axi_wstrb	Input	4	Write strobes.
s_axi_wvalid	Input	1	Write valid.
s_axi_wready	Output	1	Write ready.
s_axi_bresp	Output	2	Write response.
s_axi_bvalid	Output	1	Write response valid.
s_axi_bready	Input	1	Response ready.
s_axi_araddr	Input	19	Read address.
s_axi_arprot	Input	3	Protection type.

Table 2-4: **DisplayPort 1.4 TX Subsystem Ports *(Cont'd)***

Signal Name	Direction	Width	Description
s_axi_arvalid	Input	1	Read address valid.
s_axi_arready	Output	1	Read address ready.
s_axi_rdata	Output	32	Read data.
s_axi_rresp	Output	2	Read response.
s_axi_rvalid	Output	1	Read valid.
s_axi_rready	Input	1	Read ready.
AXI4-Stream Interface (Enabled when the AXI4-Stream interface is selected)			
s_axis_aclk_stream1	Input	1	AXI4-Stream clock.
s_axis_aresetn_stream1	Input	1	AXI4-Stream reset. Active-Low.
s_axis_video_stream1_tdata	Input	192	Video data input. Maximum width is 192.
s_axis_video_stream1_tlast	Input	1	Video end of line.
s_axis_video_stream1_tready	Output	1	AXI4-Stream tready output.
s_axis_video_stream1_tuser	Input	1	Video start of frame.
s_axis_video_stream1_tvalid	Input	1	Video valid.
Native Video Interface (Enabled when native video is selected)			
tx_video_stream1_tx_vid_vsync	Input	1	Vertical sync pulse. Active on the rising edge.
tx_video_stream1_tx_vid_hsync	Input	1	Horizontal sync pulse. Active on the rising edge
tx_video_stream1_tx_vid_enable	Input	1	User data video enable.
tx_video_stream1_tx_vid_pixel0	Input	48	Video data
tx_video_stream1_tx_vid_pixel1	Input	48	Video data
tx_video_stream1_tx_vid_pixel2	Input	48	Video data
tx_video_stream1_tx_vid_pixel3	Input	48	Video data
tx_video_stream1_tx_vid_oddeven	Input	1	Odd/even field select. Indicates an odd (1) or even (0) field polarity.
User Ports			
tx_vid_clk_stream1	Input	1	User video clock.
tx_vid_rst_stream1	Input	1	User video reset. Active-High.
tx_hpd	Input	1	Hot-plug detect signal to TX from RX.
Audio AXI4-Stream Interface			
s_axis_audio_ingress_aclk	Input	1	AXI4-Stream clock.
s_axis_audio_ingress_aresetn	Input	1	Active-Low reset.

Table 2-4: **DisplayPort 1.4 TX Subsystem Ports *(Cont'd)***

Signal Name	Direction	Width	Description
s_axis_audio_ingress_tdata	Input	32	AXI4-Stream data input. • [3:0] - Preamble Code ◦ 4'b0001: Subframe1/ Start of audio block ◦ 4'b0010: Subframe 1 ◦ 4'b0011: Subframe 2 • [27:4] - Audio Sample Word • [28] - Validity Bit (V) • [29] - User Bit (U) • [30] - Channel Status (C) • [31] - Parity (P)
s_axis_audio_ingress_tid	Input	8	• [3:0] - Audio Channel ID • [7:4] - Audio Packet Stream ID
s_axis_audio_ingress_tvalid	Input	1	Valid indicator for audio data from master.
s_axis_audio_ingress_tready	Output	1	Ready indicator from DisplayPort source.
External Video PHY Sideband Status Interface			
s_axis_phy_tx_sb_status_tdata	Output	8	Sideband status to Video PHY
s_axis_phy_tx_sb_status_tready	Input	1	Sideband status ready input from Video PHY
s_axis_phy_tx_sb_status_tvalid	Output	1	Sideband status data valid to Video PHY
External Video PHY Clock Interface			
tx_lnk_clk	Input	1	Link clock input from external Video PHY
External Video PHY Lane n [n = 0 to Lane_Count-1] Interface			
m_axis_lnk_tx_lanen_tdata	Output	32	Lanen Data to External Video PHY
m_axis_lnk_tx_lanen_tvalid	Output	1	Lanen Data Valid to External Video PHY
m_axis_lnk_tx_lanen_tready	Input	1	Lanen Data Ready from External Video PHY
m_axis_lnk_tx_lanen_tuser	Output	12	Lanen User data out to External Video PHY
AUX Signals			
aux_tx_io_n	Output	1	Negative polarity AUX Manchester-II data.
aux_tx_io_p	Output	1	Positive polarity AUX Manchester-II data.

Table 2-4: **DisplayPort 1.4 TX Subsystem Ports *(Cont'd)***

Signal Name	Direction	Width	Description
aux_tx_channel_in_p	Input	1	Positive polarity AUX channel input. Valid when AUX IO Type is unidirectional
aux_tx_channel_in_n	Input	1	Negative polarity AUX channel input. Valid when AUX IO Type is unidirectional
aux_tx_channel_out_p	Output	1	Positive polarity AUX channel Output. Valid when AUX IO Type is unidirectional
aux_tx_channel_out_n	Output	1	Negative Polarity AUX channel output. Valid when AUX IO Type is unidirectional
aux_tx_data_out	Output	1	AUX data out. Valid when AUX IO buffer location is external
aux_tx_data_in	Input	1	AUX data input. Valid when AUX IO buffer location is external
aux_tx_data_en_out_n	Output	1	AUX data output enable. Active low. Valid only when AUX IO buffer location is external
Interrupt Interface			
dptxss_dp_irq	Output	1	DisplayPort 1.4 TX IP interrupt out
dptxss_hdcp_irq	Output	1	HDCP IP interrupt out
dptxss_timer_irq	Output	1	AXI Timer IP interrupt output valid only when HDCP is enabled

Register Space

This section details registers available in the DisplayPort 1.4 TX Subsystem. The address map is split into following regions:

- VTC 0
- DisplayPort Transmit

Video Timing Controller Registers

For details about the Video Timing Controller (VTC) registers, see the *Video Timing Controller Product Guide* (PG016) [Ref 13].

DisplayPort Registers

The DisplayPort Configuration Data is implemented as a set of distributed registers which can be read or written from the AXI4-Lite interface. These registers are considered to be synchronous to the AXI4-Lite domain and asynchronous to all others.

For parameters that might change while being read from the configuration space, two scenarios might exist. In the case of single bits, either the new value or the old value is read as valid data. In the case of multiple bit fields, a lock bit might be used to prevent the status values from being updated while the read is occurring. For multi-bit configuration data, a toggle bit is used indicating that the local values in the functional core should be updated.

Any bits not specified in Table 2-5 are considered reserved and returns 0 upon read. The power on reset values of all the registers are 0 unless it is specified in the definition. Only address offsets are listed in Table 2-5. Base addresses are configured by the AXI Interconnect.

Table 2-5: **DisplayPort Source Core Configuration Space**

Offset	R/W	Definition
		Link Configuration Field
0x000	RW	LINK_BW_SET. Main link bandwidth setting. The register uses the same values as those supported by the DPCD register of the same name in the sink device. • [7:0] – LINK_BW_SET: Sets the value of the main link bandwidth for the sink device. ◦ 0x06 = 1.62 Gb/s ◦ 0x0A = 2.7 Gb/s ◦ 0x14 = 5.4 Gb/s ◦ 0x1E = 8.1 Gb/s
0x004	RW	LANE_COUNT_SET. Sets the number of lanes used by the source in transmitting data. • [4:0] – Set to 1, 2, or 4
0x008	RW	ENHANCED_FRAME_EN • [0] -Set to 1 by the source to enable the enhanced framing symbol sequence.
0x00C	RW	TRAINING_PATTERN_SET. Sets the link training mode. • [2:0] – Set the link training pattern according to the two bit code. ◦ 000 = Training off ◦ 001 = Training pattern 1, used for clock recovery ◦ 010 = Training pattern 2, used for channel equalization ◦ 011 = Training pattern 3, used for channel equalization ◦ 111 = Training pattern 4, used for channel equalization

Table 2-5: **DisplayPort Source Core Configuration Space *(Cont'd)***

Offset	R/W	Definition
0x010	RW	LINK_QUAL_PATTERN_SET. Transmit the link quality pattern. • [2:0] – Enable transmission of the link quality test patterns. ∘ 000 = Link quality test pattern not transmitted ∘ 001 = D10.2 test pattern (unscrambled) transmitted ∘ 010 = Symbol Error Rate measurement pattern ∘ 011 = PRBS7 transmitted ∘ 100 = Custom 80-Bit pattern ∘ 101 = HBR2 compliance pattern
0x014	RW	SCRAMBLING_DISABLE. Set to 1 when the transmitter has disabled the scrambler and transmits all symbols. • [0] – Disable scrambling.
0x01C	WO	SOFTWARE_RESET. Reads will return zeros. • [0] – Soft Video Reset: When set, video logic is reset (stream 1). • [7] – AUX Soft Reset. When set, AUX logic is reset.
0x020	RW	Custom 80-Bit quality pattern Bits[31:0]
0x024	RW	Custom 80-Bit quality pattern Bits[63:32]
0x028	RW	[31:16] - Reserved [15:0] - Customer 80-bit quality pattern Bits[80:64]
Core Enables		
0x080	RW	TRANSMITTER_ENABLE. Enable the basic operations of the transmitter. • [0] – When set to 1, stream transmission is enabled. When set to 0, all lanes of the main link output stuffing symbols.
0x084	RW	MAIN_STREAM_ENABLE. Enable the transmission of main link video information. • [0] – When set to 0, the active lanes of the DisplayPort transmitter will output only VB-ID information with the NoVideo flag set to 1. ***Note:*** Main stream enable/disable functionality is gated by the VSYNC input. The values written in the register are applied at the video frame boundary only.
0x090	RW	VIDEO_PACKING_CLOCK_CONTROL: This register is used when GT data width is 32-bit.To meet the bandwidth requirement for the resolutions where vid_clk/vid_pixel_mode < lnk_clk frequency and with BPC 12/16 the video packing has to work at lnk_clk, setting the bit to 1 enables the packing from lnk_clk domain. By default video data packing is done in Vid_clk.All the resolutions with less than or equal to 10-BPC works with packing at vid_clk. [0] – set to 1 to enable the video data packing to work in lnk_clk for SST video
0x0C0	WO	FORCE_SCRAMBLER_RESET. Reads from this register always return 0x0. • [0] – 1 forces a scrambler reset.

Table 2-5: **DisplayPort Source Core Configuration Space *(Cont'd)***

Offset	R/W	Definition
0x0F0	RW	TX_LINE_RESET_DISABLE. TX line reset disable. This register bits have to be used to disable the end of line reset to the internal video pipe in case of reduced blanking video support. [0] - End of line reset disable to the SST video stream
		Core ID
0x0FC	RO	CORE_ID. Returns the unique identification code of the core and the current revision level. • [31:24] – DisplayPort protocol major version • [23:16] – DisplayPort protocol minor version • [15:8] – DisplayPort protocol revision • [7:0] ◦ 0x00: Transmit ◦ 0x01: Receive The CORE_ID value for the protocol and core is DisplayPort Standard v1.4 protocol with a Transmit core: 32'h01_04_00_00.
		AUX Channel Interface

0x100	RW	AUX_COMMAND_REGISTER. Initiates AUX channel commands of the specified length.
		• [12] – Address only transfer enable. When this bit is set to 1, the source initiates Address only transfers (STOP is sent after the command).
		• [11:8] – AUX Channel Command.
		  ◦ 0x8 = AUX Write
		  ◦ 0x9 = AUX Read
		  ◦ 0x0 = IC Write
		  ◦ 0x4 = IC Write MOT
		  ◦ 0x1 = IC Read
		  ◦ 0x5 = IC Read MOT
		  ◦ 0x2 = IC Write Status
		• [3:0] – Specifies the number of bytes to transfer with the current command. The range of the register is 0 to 15 indicating between 1 and 16 bytes of data.

0x104	WO	AUX_WRITE_FIFO. FIFO containing up to 16 bytes of write data for the current AUX channel command. • [7:0] – AUX Channel byte data.
0x108	RW	AUX_ADDRESS. Specifies the address for the current AUX channel command. • [19:0] – Twenty bit address for the start of the AUX Channel burst.

Table 2-5: **DisplayPort Source Core Configuration Space *(Cont'd)***

Offset	R/W	Definition
0x10C	RW	AUX_CLOCK_DIVIDER. Contains the clock divider value for generating the internal 1 MHz clock from the AXI4-Lite host interface clock. The clock divider register provides integer division only and does not support fractional AXI4-Lite clock rates (for example, set to 75 for a 75 MHz AXI4-Lite clock). • [15:8] – The number of AXI4-Lite clocks (defined by the AXI4-Lite clock name: s_axi_aclk) equivalent to the recommended width of AUX pulse. Allowable values include: 8,16,24,32,40 and 48. • [7:0] – Clock divider value. From DisplayPort Protocol spec, AUX Pulse Width range = 0.4 to 0.6 µs. For example, for AXI4-Lite clock of 50 MHz (= 20 ns), the filter width, when set to 24, falls in the allowable range as defined by the protocol spec. ((20 × 24 = 480)) Program a value of 24 in this register.
0x110	RC	TX_USER_FIFO_OVERFLOW. Indicates an overflow in the user FIFO. The event can occur if the video rate does not match the TU size programming. • [0] – FIFO_OVERFLOW_FLAG: 1 indicates that the internal FIFO has detected an overflow condition. This bit clears upon read.
0x130	RO	INTERRUPT_SIGNAL_STATE. Contains the raw signal values for those conditions which might cause an interrupt. • [3] – REPLY_TIMEOUT: 1 indicates that a reply timeout has occurred. • [2] – REPLY_STATE: 1 indicates that a reply is currently being received. • [1] – REQUEST_STATE: 1 indicates that a request is currently being sent. • [0] – HPD_STATE: Contains the raw state of the HPD pin on the DisplayPort connector.
0x134	RO	AUX_REPLY_DATA. Maps to the internal FIFO which contains up to 16 bytes of information received during the AUX channel reply. Reply data is read from the FIFO starting with byte 0. The number of bytes in the FIFO corresponds to the number of bytes requested. • [7:0] – AUX reply data

0x138	RO	AUX_REPLY_CODE. Reply code received from the most recent AUX Channel request. The AUX Reply Code corresponds to the code from the DisplayPort Standard.
		Note: The core does not retry any commands that were Deferred or Not Acknowledged.
		• [3:2]
		  ◦ 00 = I2C ACK
		  ◦ 01 = I2C NACK
		  ◦ 10 = I2C DEFER
		• [1:0]
		  ◦ 00 = AUX ACK
		  ◦ 01 = AUX NACK
		  ◦ 10 = AUX DEFER

0x13C	RW	AUX_REPLY_COUNT. Provides an internal counter of the number of AUX reply transactions received on the AUX Channel. Writing to this register clears the count. • [7:0] – Current reply count.

Table 2-5: **DisplayPort Source Core Configuration Space *(Cont'd)***

Offset	R/W	Definition
0x140	RC	INTERRUPT_STATUS. Source core interrupt status register. A read from this register clears all values. Write operation is illegal and clears the values. • [9] – Audio packet ID mismatch interrupt, sets when incoming audio packet ID over AXI4-Stream interface does not match with the info frame packet stream ID. • [5] – EXT_PKT_TXD: Extended packet is transmitted and controller is ready to accept new packet. Extended packet address space can also be used to send the audio copy management packet/ISRC packet/VSC packets. • [4] – HPD_PULSE_DETECTED: A pulse on the HPD line was detected. The duration of the pulse can be determined by reading 0x150. • [3] – REPLY_TIMEOUT: A reply timeout has occurred. • [2] – REPLY_RECEIVED: An AUX reply transaction has been detected. • [1] – HPD_EVENT: The core has detected the presence of the HPD signal. This interrupt asserts immediately after the detection of HPD and after the loss of HPD for 2 msec. • [0] – HPD_IRQ: An IRQ framed with the proper timing on the HPD signal has been detected.
0x144	RW	INTERRUPT_MASK. Masks the specified interrupt sources from asserting the axi_init signal. When set to a 1, the specified interrupt source is masked. This register resets to all 1s at power up. The respective MASK bit controls the assertion of axi_int only and does not affect events updated in the INTERRUPT_STATUS register. • [9] – Mask Audio packet ID mismatch interrupt. • [5] – EXT_PKT_TXD: Mask Extended Packet Transmitted interrupt. • [4] – HPD_PULSE_DETECTED: Mask HPD Pulse interrupt. • [3] – REPLY_TIMEOUT: Mask reply timeout interrupt. • [2] – REPLY_RECEIVED: Mask reply received interrupt. • [1] – HPD_EVENT: Mask HPD event interrupt. • [0] – HPD_IRQ: Mask HPD IRQ interrupt.
0x148	RO	REPLY_DATA_COUNT. Returns the total number of data bytes actually received during a transaction. This register does not use the length byte of the transaction header. • [4:0] – Total number of data bytes received during the reply phase of the AUX transaction.
0x14C	RO	REPLY_STATUS • [15:12] – RESERVED • [11:4] – REPLY_STATUS_STATE: Internal AUX reply state machine status bits. • [3] – REPLY_ERROR: When set to a 1, the AUX reply logic has detected an error in the reply to the most recent AUX transaction. • [2] – REQUEST_IN_PROGRESS: The AUX transaction request controller sets this bit to a '1' while actively transmitting a request on the AUX serial bus. The bit is set to 0 when the AUX transaction request controller is idle. • [1] – REPLY_IN_PROGRESS: The AUX reply detection logic sets this bit to a 1 while receiving a reply on the AUX serial bus. The bit is 0 otherwise. • [0] – REPLY_RECEIVED: This bit is set to '0' when the AUX request controller begins sending bits on the AUX serial bus. The AUX reply controller sets this bit to 1 when a complete and valid reply transaction has been received.

Table 2-5: **DisplayPort Source Core Configuration Space *(Cont'd)***

Offset	R/W	Definition
0x150	RO	HPD_DURATION • [15:0] – Duration of the HPD pulse in microseconds.
0x154	RO	Free running counter incrementing for every 1 MHz.
Main Stream Attributes (Refer to the DisplayPort Standard for more details [Ref 3].)		
0x180	RW	MAIN_STREAM_HTOTAL. Specifies the total number of clocks in the horizontal framing period for the main stream video signal. • [15:0] – Horizontal line length total in clocks.
0x184	RW	MAIN_STREAM_VTOTAL. Provides the total number of lines in the main stream video frame. • [15:0] – Total number of lines per video frame.
0x188	RW	MAIN_STREAM_POLARITY. Provides the polarity values for the video sync signals. Polarity information is packed and sent in the MSA packet. See the Main Stream Attribute Data Transport section of the *DisplayPort Standard v1.4 Specification* [Ref 4]. • 0 = Active-High • 1 = Active-Low • [1] – VSYNC_POLARITY: Polarity of the vertical sync pulse. • [0] – HSYNC_POLARITY: Polarity of the horizontal sync pulse.
0x18C	RW	MAIN_STREAM_HSWIDTH. Sets the width of the horizontal sync pulse. • [14:0] – Horizontal sync width in clock cycles.
0x190	RW	MAIN_STREAM_VSWIDTH. Sets the width of the vertical sync pulse. • [14:0] – Width of the vertical sync in lines.
0x194	RW	MAIN_STREAM_HRES. Horizontal resolution of the main stream video source. • [15:0] – Number of active pixels per line of the main stream video.
0x198	RW	MAIN_STREAM_VRES. Vertical resolution of the main stream video source. • [15:0] – Number of active lines of video in the main stream video source.
0x19C	RW	MAIN_STREAM_HSTART. Number of clocks between the leading edge of the horizontal sync and the start of active data. • [15:0] – Horizontal start clock count.
0x1A0	RW	MAIN_STREAM_VSTART. Number of lines between the leading edge of the vertical sync and the first line of active data. • [15:0] – Vertical start line count.

Table 2-5: **DisplayPort Source Core Configuration Space *(Cont'd)***

Offset	R/W	Definition
0x1A4	RW	MAIN_STREAM_MISC0. Miscellaneous stream attributes. • [7:0] – Implements the attribute information contained in the DisplayPort MISC0 register described in section 2.2.4 of the standard. • [12] – 0: Default Behavior. 1: Enables mode to sync Ext packet transmission with Vsync event. • [11] – Maud control (Advanced Users) • [10] – Audio Only Mode. When enabled, controller inserts information/timestamp packets every 512 BS symbols. By default the value is 0. • [9] – Sync/Async Mode for Audio • [8] – Override Audio Clocking Mode • [7:5] – Bit depth per color/component. • [4] – YCbCr Colorimetry. • [3] – Dynamic Range. • [2:1] – Component Format. • [0] – Synchronous Clock.
0x1A8	RW	MAIN_STREAM_MISC1. Miscellaneous stream attributes. • [7:0] – Implements the attribute information contained in the DisplayPort MISC1 register described in section 2.2.4 of the standard. • [6:3] – Reserved. • [2:1] – Stereo video attribute. • [0] – Interlaced vertical total even.
0x1AC	RW	M-VID. If synchronous clocking mode is used, this register must be written with the M value as described in section 2.2.5.2 of the standard. When in asynchronous clocking mode, the M value for the video stream is automatically computed by the source core and written to the main stream. These values are not written into the M-VID register for readback. • [23:0] – Unsigned M value.
0x1B0	RW	TRANSFER_UNIT_SIZE. Sets the size of a transfer unit in the framing logic On reset, transfer size is set to 64. This register must be written as described in section 2.2.1.4.1 of the standard. • [6:0] – This number should be 32 or 64 and is set to a fixed value that depends on the inbound video mode. Note that bit 0 cannot be written (the transfer unit size is always even).
0x1B4	RW	N-VID. If synchronous clocking mode is used, this register must be written with the N value as described in section 2.2.5.2 of the standard. When in asynchronous clocking mode, the M value for the video stream is automatically computed by the source core and written to the main stream. These values are not written into the N-VID register for readback. • [23:0] – Unsigned N value.

Table 2-5: **DisplayPort Source Core Configuration Space *(Cont'd)***

Offset	R/W	Definition
0x1B8	RW	USER_PIXEL_WIDTH. Selects the width of the user data input port. In SST, the user pixel width should always be equal to the active line count generated in hardware. • [2:0]: ◦ 1 - Single pixel wide interface ◦ 2 - Dual pixel wide interface. Valid for designs with 2 or 4 lanes. ◦ 4 - Quad pixel wide interface.Valid for designs with 4 lanes only.
0x1BC	RW	USER_DATA_COUNT_PER_LANE. This register is used to translate the number of pixels per line to the native internal 16-bit datapath. If (HRES * bits per pixel) is divisible by 16, then word_per_line = ((HRES × bits per pixel)/16) Else word_per_line = (INT((HRES × bits per pixel)/16)) + 1 For single-lane design: Set USER_DATA_COUNT_PER_LANE = words_per_line - 1 For 2-lane design: If words_per_line is divisible by 2, then Set USER_DATA_COUNT_PER_LANE = words_per_line - 2 Else Set USER_DATA_COUNT_PER_LANE = words_per_line + MOD(words_per_line,2) - 2 For 4-lane design: If words_per_line is divisible by 4, then Set USER_DATA_COUNT_PER_LANE = words_per_line - 4 Else Set USER_DATA_COUNT_PER_LANE = words_per_line + MOD(words_per_line,4) - 4
0x1C0	RW	MAIN_STREAM_INTERLACED. Informs the DisplayPort transmitter main link that the source video is interlaced. By setting this bit to a 1, the core sets the appropriate fields in the VBID value and Main Stream Attributes. This bit must be set to a 1 for the proper transmission of interlaced sources. • [0] – Set to a 1 when transmitting interlaced images.
0x1C4	RW	MIN_BYTES_PER_TU. Programs source to use MIN number of bytes per transfer unit. The calculation should be done based on the DisplayPort Standard. • [6:0] – Set the value to INT((VIDEO_BW/LINK_BW)*TRANSFER_UNIT_SIZE)
0x1C8	RW	FRAC_BYTES_PER_TU. Calculating MIN bytes per TU is often not a whole number. This register is used to hold the fractional component. • [9:0] – The fraction part of ((VIDEO_BW/LINK_BW)*TRANSFER_UNIT_SIZE) scaled by 1024 is programmed in this register.

Table 2-5: **DisplayPort Source Core Configuration Space *(Cont'd)***

Offset	R/W	Definition
0x1CC	RW	INIT_WAIT. This register defines the number of initial wait cycles at the start of a new line by the Framing logic. This allows enough data to be buffered in the input FIFO. The default value of INIT_WAIT is 0x20. If (MIN_BYTES_PER_TU ≤ 4) • [7:0] – Set INIT_WAIT to 64 else if color format is RGB/YCbCr_444 • [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU) else if color format is YCbCr_422 • [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)/2 else if color format is Y_Only • [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)/3
PHY Configuration Status		
0x280	RO	PHY_STATUS. Provides the current status from the PHY. • [31:30] – Unused, read as 0. • [29:28] – Transmitter buffer status, lane 3. • [27:26] – Unused, read as 0. • [25:24] – Transmitter buffer status, lane 2. • [23:22] – Unused, read as 0. • [21:20]- Transmitter buffer status, lane 1. • [19:18] – Unused, read as 0. • [17:16] – Transmitter buffer status, lane 0. • [15:7] – Unused, read as 0. • [6] – FPGA fabric clock PLL locked. • [5] – PLL for lanes 2 and 3 locked. • [4] – PLL for lanes 0 and 1 locked. • [3:2] – Reset done for lanes 2 and 3. • [1:0] – Reset done for lanes 0 and 1.

DisplayPort Audio

The DisplayPort Audio registers are listed in Table 2-6.

Table 2-6: **DisplayPort Audio Registers**

Offset	R/W	Definition
0x300	R/W	TX_AUDIO_CONTROL. Enables audio stream packets in main link and provides buffer control. • [16]: Set to 1 to mute the audio over link • [0]: Audio Enable
0x304	R/W	TX_AUDIO_CHANNELS. Used to input active channel count. Transmitter collects audio samples based on this information. • [2:0] Channel Count
0x308	WO	TX_AUDIO_INFO_DATA. [31:0] Word formatted as per CEA 861-C Info Frame. Total of eight words should be written in following order: • 1st word – ◦ [31:24] = HB3 ◦ [23:16] = HB2 ◦ [15:8] = HB1 ◦ [7:0] = HB0 • 2nd word – DB3,DB2,DB1,DB0 .... • 8th word –DB27,DB26,DB25,DB24 The data bytes DB1...DBN of CEA Info frame are mapped as DB0-DBN-1. No protection is provided for wrong operations by software.
0x328	R/W	TX_AUDIO_MAUD. M value of audio stream as computed by transmitter. • [23:0] = Unsigned value computed when audio clock and link clock are synchronous.

Table 2-6: **DisplayPort Audio Registers *(Cont'd)***

Offset	R/W	Definition
0x32C	R/W	TX_AUDIO_NAUD. N value of audio stream as computed by transmitter. • [23:0] = Unsigned value computed when audio clock and link clock are synchronous.
0x330 to 0x350	WO	TX_AUDIO_EXT_DATA. [31:0] = Word formatted as per Extension packet described in protocol standard. Extended packet is fixed to 32 Bytes length. The controller has buffer space for only one extended packet. Extension packet address space can be used to send the audio Copy management packet/ISRC packet/VSC packets. TX is capable of sending any of these packets. VSC/EXT packets should use the same address space. A total of nine words should be written in following order: • 1st word - ◦ [31:24] = HB3 ◦ [23:16] = HB2 ◦ [15:8] = HB1 ◦ [7:0] = HB0 • 2nd word - DB3,DB2,DB1,DB0 ... • 9th word -DB31,DB30,DB29,DB28 See the DisplayPort Standard for HB* definition. No protection is provided for wrong operations by software. This is a key-hole memory. So, nine writes to this address space is required.

Designing with the Core

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

DisplayPort Overview

The Source core moves a video stream from a standardized main link through a complete DisplayPort Link Layer and onto High-Speed Serial I/O for transport to a Sink device.

Main Link Setup and Management

This section is intended to elaborate on and act as a companion to the link training procedure in the *VESA DisplayPort Standard v1.4* [Ref 3].

Xilinx® advises all users of the source core to use a MicroBlaze™ processor or similar embedded processor to properly initialize and maintain the link. The tasks encompassed in the Link and Stream Policy Makers are likely too complicated to be efficiently managed by a hardware-based state machine. Xilinx does not recommend using the RTL based controllers.

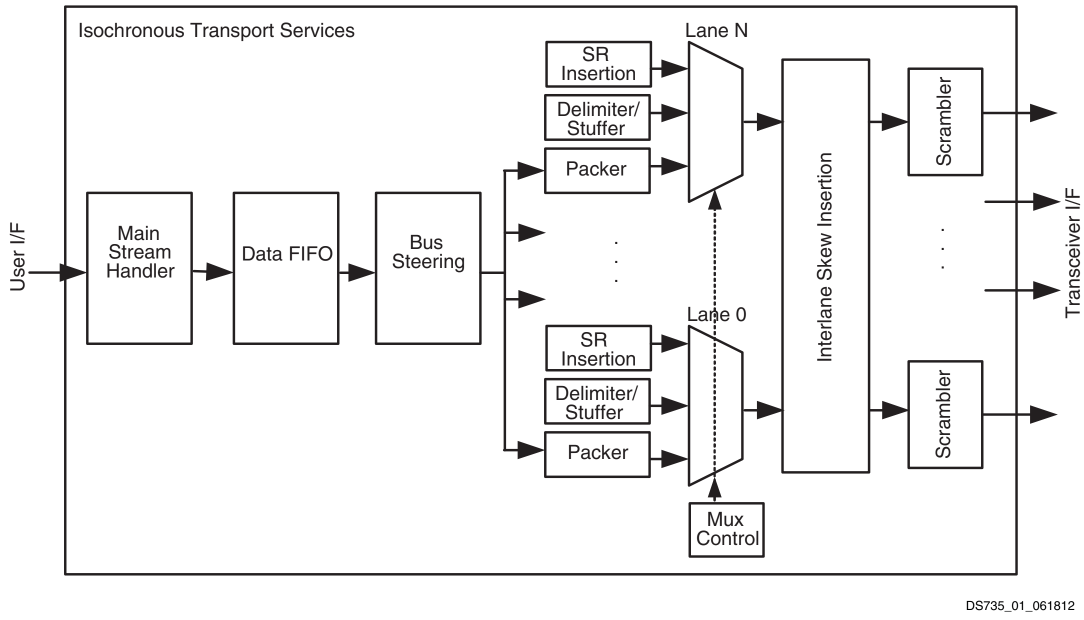


Figure 3-1: **Source Main Link Datapath**

Link Training

The link training commands are passed from the DPCD register block to the link training function. When set into the link training mode, the functional datapath is blocked and the link training controller issues the specified pattern. Care must be taken to place the Sink device in the proper link training mode before the source state machine enters a training state. Otherwise, unpredictable results might occur.

Figure 3-2 shows the flow diagram for link training. For details, refer to the *VESA DisplayPort Standard v1.4* [Ref 3].

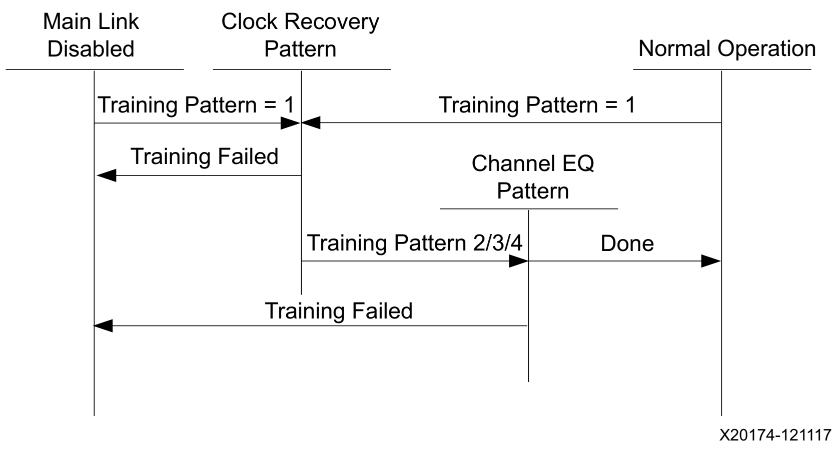


Figure 3-2: **Link Training States**

Source Core Setup and Initialization

The following text contains the procedural tasks required to achieve link communication. For description of the DPCD, see *VESA DisplayPort Standard v1.4* [Ref 3].

IMPORTANT: *During initialization ensure that TX8B10BEN is not cleared in offset 0x0070 of the corresponding Video PHY Controller Product Guide (PG230) [Ref 1]. For this release, information on the DisplayPort 1.4 is not available in the PG230.*

Source Core Setup

1. Place the PHY into reset.
2. Disable the transmitter.
    - TRANSMITTER_ENABLE = `0x00`
3. Set the clock divider.
    - AUX_CLOCK_DIVIDER = (see register description for proper value)
4. Select and set up the reference clock for the desired link rate in the Video PHY Controller.
5. Bring the PHY out of reset.
6. Wait for the PHY to be ready.
7. Enable the transmitter.
    - TRANSMITTER_ENABLE = `0x01`
8. (Optional) Turn on the interrupt mask for HPD.
    - INTERRUPT_MASK = `0x00`

Note: At this point, the source core is initialized and ready to use. The link policy maker should be monitoring the status of HPD and taking appropriate action for connect/disconnect events or HPD interrupt pulses.

Upon HPD Assertion

1. Read the DPCD capabilities fields out of the sink device (`0x00000` to `0x0000B`) though the AUX channel.
2. Determine values for lane count, link speed, enhanced framing mode, downspread control and main link channel code based on each link partners' capability and needs.
3. Write the configuration parameters to the link configuration field (`0x00100` to `0x00101`) of the DPCD through the AUX channel.

Note: Some sink devices' DPCD capability fields are unreliable. Many source devices start with the maximum transmitter capabilities and scale back as necessary to find a configuration the sink device can handle. This could be an advisable strategy instead of relying on DPCD values.

4. Equivalently, write the appropriate values to the Source core's local configuration space.
    a. LANE_COUNT_SET
    b. LINK_BW_SET
    c. ENHANCED_FRAME_EN
    d. PHY_CLOCK_SELECT

Training Pattern 1 Procedure (Clock Recovery)

1. Turn off scrambling and set training pattern 1 in the source through direct register writes.
    - SCRAMBLING_DISABLE = `0x01`
    - TRAINING_PATTERN_SET = `0x01`
2. Turn off scrambling and set training pattern 1 in the sink DPCD (`0x00102` to `0x00106`) through the AUX channel.
3. Wait for the aux read interval configured in TRAINING_AUX_RD_INTERVAL DPCD Register (`0x0000E`) before reading status registers for all active lanes (`0x00202` to `0x00203`) through the AUX channel.
4. If clock recovery failed, check for voltage swing or pre-emphasis level increase requests (`0x00206` to `0x00207`) and react accordingly.
    - Run this loop up to five times. If after five iterations this has not succeeded, reduce link speed if at high speed and try again. If already at low speed, training fails.

Training Pattern 2/3/4 Procedure (Symbol Recovery, Interlane Alignment)

1. Turn off scrambling and set training pattern 2 in the source through direct register writes.
    - SCRAMBLING_DISABLE = `0x01`
    - Set training pattern to pattern 2, pattern 3, or pattern 4 based on the Sink DPCD capability
2. Set proper state for scrambling and set training pattern in the sink DPCD (`0x00102` to `0x00106`) through the AUX channel.
3. Wait for aux read interval configured in TRAINING_AUX_RD_INTERVAL DPCD Register (`0x0000E`) then read status registers for all active lanes (`0x00202` to `0x00203`) through the AUX channel.
4. Check the channel equalization, symbol lock, and interlane alignment status bits for all active lanes (`0x00204`) through the AUX channel.
5. If any of these bits are not set, check for voltage swing or pre-emphasis level increase requests (`0x00206` to `0x00207`) and react accordingly.

6. Run this loop up to five times. If after five iterations this has not succeeded, reduce link speed if at high speed and Return to the instructions for Training Pattern 1. If already at low speed, training fails.
7. Signal the end of training by enabling scrambling and setting training pattern to `0x00` in the sink device (`0x00102`) through the AUX channel.
8. On the source side, re-enable scrambling and turn off training.
   - TRAINING_PATTERN_SET = `0x00`
   - SCRAMBLING_DISABLE = `0x00`

At this point, training has completed.

Enabling Main Link Video

Main link video should not be enabled until a proper video source has been provided to the source core. Typically the source device wants to read the EDID from the attached sink device to determine its capabilities, most importantly its preferred resolution and other resolutions that it supports should the preferred mode not be available. Once a resolution has been determined, set the Main Stream Attributes in the source core (`0x180` to `0x1B0`). Enable the main stream (`0x084`) only when a reliable video source is available.

IMPORTANT: *When the main link video is enabled, the scrambler/de-scrambler must be reset every 512th BS Symbol as described in section 2.2.1.1 of the DisplayPort standard. For simulation purposes, you should force a scrambler reset by writing a "1" to 0x0c0 before the main link is enabled to reduce the amount of time after startup needed to align the scramber/de-scrambler.*

Accessing the Link Partner

The DisplayPort core is configured through the AXI4-Lite host interface. The host processor interface uses the DisplayPort AUX Channel to read the register space of the attached sink device and determines the capabilities of the link. Accessing DPCD and EDID information from the Sink is done by writing and reading from register space `0x100` through `0x144`. (For information on the DPCD register space, refer to the *VESA DisplayPort Standard v1.4*.)

Before any AUX channel operation can be completed, you must first set the proper clock divider value in `0x10C`. This must be done only one time after a reset. The value held in this register should be equal to the frequency of `s_axi_aclk`. So, if `s_axi_aclk` runs at 135 MHz, the value of this register should be 135 (`'h87`). This register is required to apply a proper divide function for the AUX channel sample clock, which must operate at 1 MHz.

The act of writing to the AUX_COMMAND initiates the AUX event. Once an AUX request transaction is started, the host should not write to any of the control registers until the REPLY_RECEIVED bit is set to 1, indicating that the sink has returned a response.

AUX Write Transaction

An AUX write transaction is initiated by setting up the AUX_ADDRESS, and writing the data to the AUX_WRITE_FIFO followed by a write to the AUX_COMMAND register with the code `0x08`. Writing the command register begins the AUX channel transaction. The host should wait until either a reply received event or reply timeout event is detected. These events are detected by reading INTERRUPT_STATUS registers (either in ISR or polling mode).

When the reply is detected, the host should read the AUX_REPLY_CODE register and look for the code 0x00 indicating that the AUX channel has successfully acknowledged the transaction.

Figure 3-3 shows a flow of an AUX write transaction.

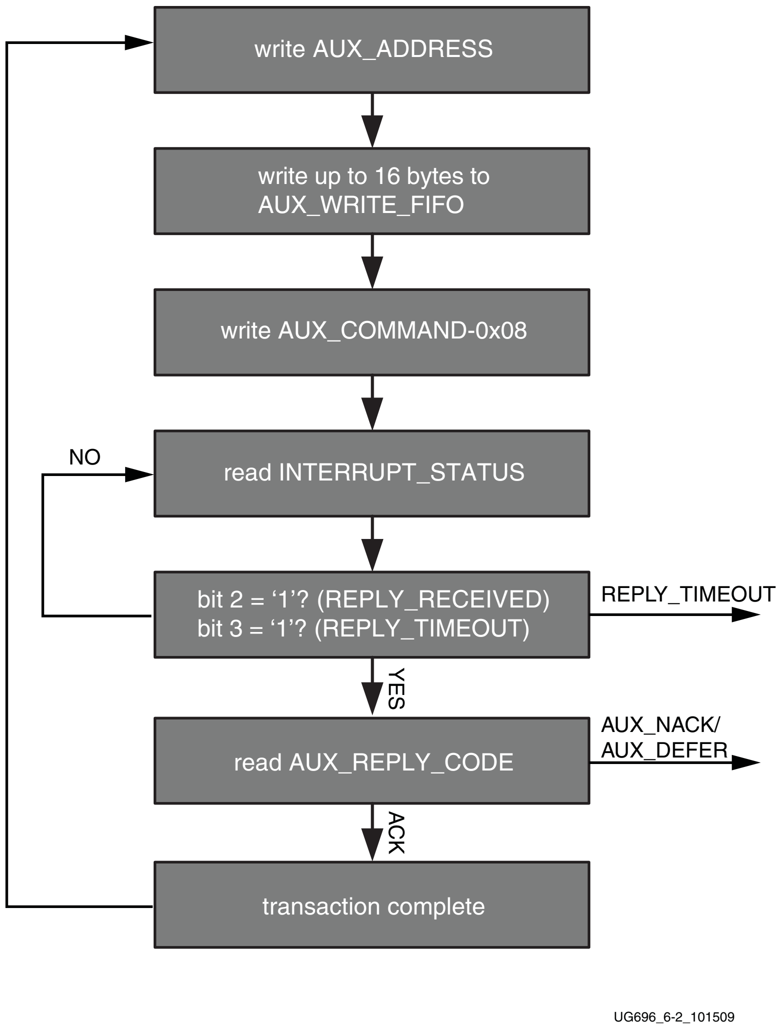


Figure 3-3: **AUX Write Transaction**

AUX Read Transaction

The AUX read transaction is prepared by writing the transaction address to the AUX_ADDRESS register. Once set, the command and the number of bytes to read are written to the AUX_COMMAND register. After initiating the transfer, the host should wait for an interrupt or poll the INTERRUPT_STATUS register to determine when a reply is received.

When the REPLY_RECEIVED signal is detected, the host might then read the requested data bytes from the AUX_REPLY_DATA register. This register provides a single address interface to a byte FIFO which is 16 elements deep. Reading from this register automatically advances the internal read pointers for the next access.

Figure 3-4 shows a flow of an AUX read transaction.

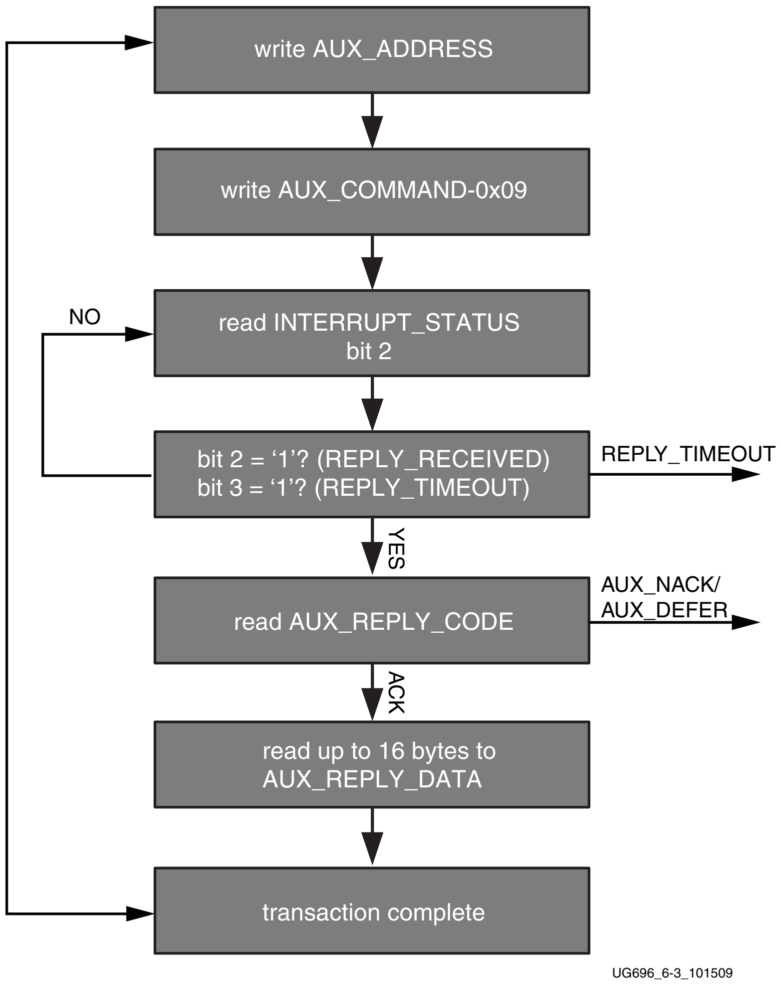


Figure 3-4: **AUX Read Transaction**

Commanded I2C Transactions

The core supports a special AUX channel command intended to make I2C over AUX transactions faster and easier to perform. In this case, the host will bypass the external I2C master/slave interface and initiate the command by directly writing to the register set.

The sequence for performing these transactions is exactly the same as a native AUX channel transaction with a change to the command written to the AUX_COMMAND register. The supported I2C commands are summarized in Table 3-1.

Table 3-1: **I2C over AUX Commands**

AUX_COMMAND[11:8]	Command
0x0	IIC Write
0x4	IIC Write MOT
0x1	IIC Read
0x5	IIC Read MOT
0x6	IIC Write Status with MOT
0x2	IIC Write Status

By using a combination of these commands, the host might emulate an I2C transaction.

Figure 3-5 shows the flow of commanded I2C transactions.

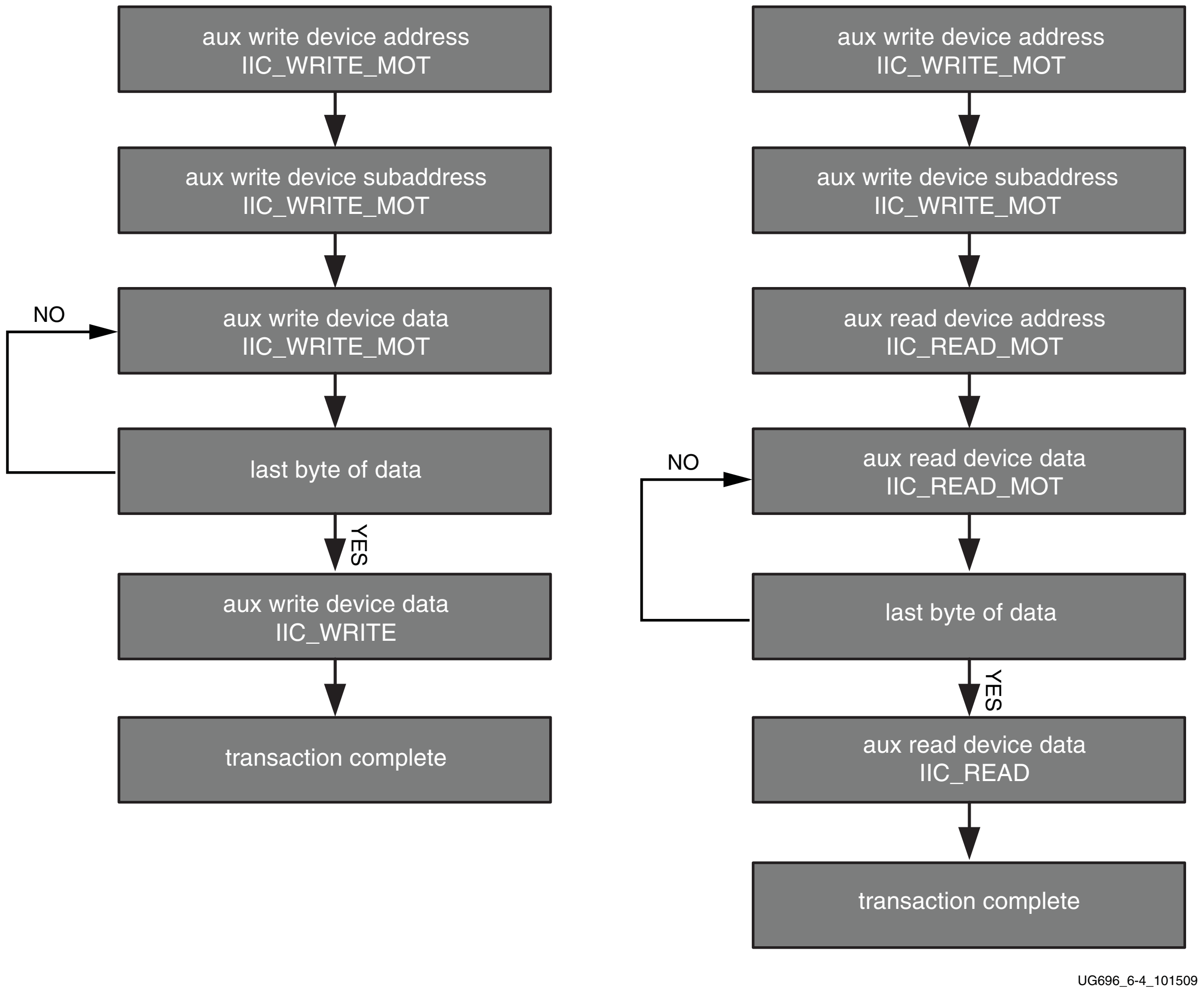


Figure 3-5: **Commanded I2C Device Transactions, Write (Left) and Read (Right)**

Since I2C transactions might be significantly slower than AUX channel transactions, the host should be prepared to receive multiple AUX_DEFER reply codes during the execution of the above state machines.

The AUX-I2C commands are as follows:

- MOT Definition:
  - Middle Of Transaction bit in the command field.
  - This controls the stop condition on the I2C slave.
  - For a transaction with MOT set to 1, the I2C bus is not STOPPED, but left to remain the previous state.
  - For a transaction with MOT set to 0, the I2C bus is forced to IDLE at the end of the current command or in special Abort cases.

- Partial ACK:
  - For I2C write transactions, the Sink core can respond with a partial ACK (ACK response followed by the number of bytes written to I2C slave).

Special AUX commands include:

- Write Address Only and Read Address Only: These commands do not have any length field transmitted over the AUX channel. The intent of these commands are to:
  - Send address and RD/WR information to I2C slave. No Data is transferred.
  - End previously active transaction, either normally or through an abort.

  The Address Only Write and Read commands are generated from the source by using bit [12] of the command register with command as I2C WRITE/READ.
- Write Status: This command does not have any length information. The intent of the command is to identify the number of bytes of data that have been written to an I2C slave when a Partial ACK or Defer response is received by the source on a AUX-I2C write.

  The Write status command is generated from the source by using Bit[12] of the command register with command as I2C WRITE STATUS.
- IIC Timeout: The sink controller monitors the IIC bus after a transaction starts and looks for an IIC stop occurrence within 1 second. If an IIC stop is not received, it is considered as an IIC timeout and the sink controller issues a stop condition to release the bus. This timeout avoids a lock-up scenario.

Generation of AUX transactions are described in Table 3-2.

Table 3-2: **Generation of AUX Transactions**

Transaction	AUX Transaction	I2C Transaction	Usage	Sequence
Write Address only with MOT = 1	START -> CMD -> ADDRESS -> STOP	START -> DEVICE_ADDR -> WR -> ACK/NACK	Setup I2C slave for Write to address defined	1. Write AUX Address register (0x108) with device address. 2. Issue command to transmit transaction by writing into AUX command register (0x100). Bit[12] must be set to 1.
Read Address only with MOT = 1	START -> CMD -> ADDRESS -> STOP	START -> DEVICE_ADDR -> RD -> ACK/NACK	Setup I2C slave for Read to address defined.	1. Write AUX Address register with device address. 2. Issue command to transmit transaction by writing into AUX command register. Bit [12] must be set to 1.

Table 3-2: **Generation of AUX Transactions *(Cont'd)***

Transaction	AUX Transaction	I2C Transaction	Usage	Sequence
Write / Read Address only with MOT = 0	START -> ADDRESS -> STOP	STOP	To stop the I2C slave, used as Abort or normal stop.	1. Write AUX Address register (0x108) with device address. 2. Issue command to transmit transaction by writing into AUX command register (0x100). Bit[12] must be set to 1.
Write with MOT = 1	START -> CMD -> ADDRESS -> LENGTH -> D0 to DN -> STOP	I2C bus is IDLE or New device address START -> START/RS -> DEVICE_ADDR -> WR -> ACK/NACK -> DATA0 -> ACK/NACK to DATAN -> ACK/NACK I2C bus is in Write state and the same device address DATA0 -> ACK/NACK to DATAN -> ACK/NACK	Setup I2C slave write data.	1. Write AUX Address register (0x108) with device address. 2. Write the data to be transmitted into AUX write FIFO register (0x104). 3. Issue write command and data length to transmit transaction by writing into AUX command register (0x100). Bits[3:0] represent length field.

Table 3-2: **Generation of AUX Transactions *(Cont'd)***

Transaction	AUX Transaction	I2C Transaction	Usage	Sequence
Write with MOT = 0	START -> CMD -> ADDRESS -> LENGTH -> D0 to DN -> STOP	I2C bus is IDLE or Different I2C device address START -> START/RS -> DEVICE_ADDR -> WR -> ACK/NACK -> DATA0 -> ACK/NACK to DATAN -> ACK/NACK -> STOP I2C bus is in Write state and the same I2C device address DATA0 -> ACK/NACK to DATAN -> ACK/NACK -> STOP	Setup I2C slave write data and stop the I2C bus after the current transaction.	1. Write AUX Address register (0x108) with device address. 2. Write the data to be transmitted into AUX write FIFO register (0x104). 3. Issue write command and data length to transmit transaction by writing into AUX command register (0x100). Bits[3:0] represent length field.
Read with MOT = 1	START -> CMD -> ADDRESS -> LENGTH -> STOP	I2C bus is IDLE or Different I2C device address START -> START/RS -> DEVICE_ADDR -> RD -> ACK/NACK -> DATA0 -> ACK/NACK to DATAN -> ACK/NACK I2C bus is in Write state and the same I2C device address DATA0 -> ACK/NACK to DATAN -> ACK/NACK	Setup I2C slave read data.	1. Write AUX Address register (0x108) with device address. 2. Issue read command and data length to transmit transaction by writing into AUX command register (0x100). Bits[3:0] represent the length field.

Table 3-2: **Generation of AUX Transactions *(Cont'd)***

Transaction	AUX Transaction	I2C Transaction	Usage	Sequence
Read with MOT = 0	START -> CMD -> ADDRESS -> LENGTH -> D0 to DN -> STOP	I2C bus is IDLE or Different I2C device address START -> START/RS -> DEVICE_ADDR -> RD -> ACK/NACK -> DATA0 -> ACK/NACK to DATAN -> ACK/NACK -> STOP I2C bus is in Write state and the same I2C device address DATA0 -> ACK/NACK to DATAN -> ACK/NACK -> STOP	Setup I2C slave read data and stop the I2C bus after the current transaction.	1. Write AUX Address register (0x108) with device address. 2. Issue read command and data length to transmit transaction by writing into AUX command register (0x100). Bits[3:0] represent the length field.
Write Status with MOT = 1	START -> CMD -> ADDRESS -> STOP	No transaction	Status of previous write command that was deferred or partially ACKED.	1. Write AUX Address register (0x108) with device address. 2. Issue status update command to transmit transaction by writing into AUX command register (0x100). Bit[12] must be set to 1.
Write Status with MOT = 0	START -> CMD -> ADDRESS -> STOP	Force a STOP and the end of write burst	Status of previous write command that was deferred or partially ACKED. MOT = 0 will ensure the bus returns to IDLE at the end of the burst.	1. Write AUX Address register (0x108) with device address. 2. Issue status update command to transmit transaction by writing into AUX command register (0x100). Bit[12] must be set to 1.

Handling I2C Read Defers/Timeout:

- The Sink core could issue a DEFER response for a burst read to I2C. The following are the actions that can be taken by the Source core.
    - Issue the same command (previously issued read, with same device address and length) and wait for response. The Sink core on completion of the read from I2C (after multiple defers) should respond with read data.

- Abort the current read using:
    - Read to a different I2C slave
    - Write command
    - Address-only Read or write with MOT = 0.

Handling I2C Write Partial ACK:

- The sink could issue a partial ACK response for a burst Write to I2C. The following are the actions that can be taken by the Source core:
    - Use the Write status command to poll the transfers happening to the I2C. On successful completion, the sink should issue a NACK response to these requests while intermediate ones will get a partial ACK.
    - Issue the same command for a response (previously issued with the same device address, length and data) and wait for a response. On completion of the write to I2C (after multiple partial ACKs), the Sink core should respond with an ACK.
    - Abort the current Write using:
        - Write to a different I2C slave
        - Read command
        - Address-only Read or Write with MOT = 0.

Handling I2C Write Defer/Timeout:

- The Sink core could issue a Defer response for a burst write to I2C. The following are the actions that can be taken by the Source core:
    - Use the Write status command to poll the transfers happening to the I2C. On successful completion, the Sink core should issue an ACK response to these requests while intermediate ones will get partial ACKs.
    - Issue the same command (previously issued with the same device address, length and data) and wait for response. The Sink core on completion of the write to I2C (after multiple Defers) should respond with an ACK.
    - Abort the current Write using:
        - Write to a different I2C slave
        - Read command
        - Address only Read or Write with MOT = 0.

AUX IO Location

DisplayPort source can have AUX IO located inside the IP or external to the IP based on the AUX IO location selection through GUI. The AUX IO type can be uni-directional/ bidirectional when the AUX IO is located inside the IP.

Transmitter Audio/Video Clock Generation

The transmitter clocking architecture supports both the asynchronous and synchronous clocking modes included in the *DisplayPort Standard v1.4*. The clocking mode is selected by way of the Stream Clock Mode register (MAIN_STREAM_MISC0 Bit[0]). When set to 1, the link and stream clock are synchronous, in which case the MVid and NVid values are a constant. In synchronous clock mode, the source core uses the MVid and NVid register values programmed by the host processor via the AXI4-Lite interface.

When the Stream Clock Mode register is set to 0, asynchronous clock mode is enabled and the relationship between MVid and NVid is not fixed. In this mode, the source core will transmit a fixed value for NVid and the MVid value provided as a part of the clocking interface.

Figure 3-6 shows a block diagram of the transmitter clock generation process.

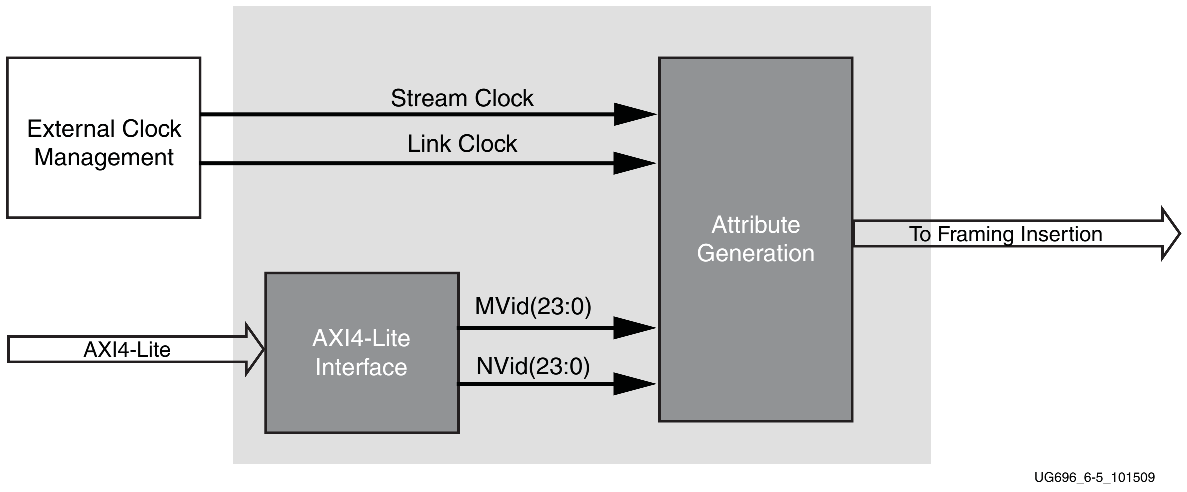


Figure 3-6: **Transmitter Audio/Video Clock Generation**

Hot Plug Detection

The Source device must debounce the incoming HPD signal by sampling the value at an interval > 250 μs. For a pulse width between 500 μs and 1 ms, the Sink device has requested an interrupt. The interrupt is passed to the host processor through the AXI4-Lite interface.

If HPD signal remains Low for > 2 ms, then the sink device has been disconnected and the link should be shut down. This condition is also passed through the AXI4-Lite interface as an interrupt. The host processor must properly determine the cause of the interrupt by reading the appropriate DPCD registers and take the appropriate action. For details, refer to the *VESA DisplayPort Standard v1.4* [Ref 3].

HPD Event Handling

HPD signaling has three use cases:

- Connection event defined as HPD_EVENT is detected, and the state of the HPD is 1.
- Disconnection event defined as HPD_EVENT is detected, and the state of the HPD is 0.
- HPD IRQ event as captured in the INTERRUPT_STATUS register bit 0.

Figure 3-7 shows the source core state and basic actions to be taken based on HPD events.

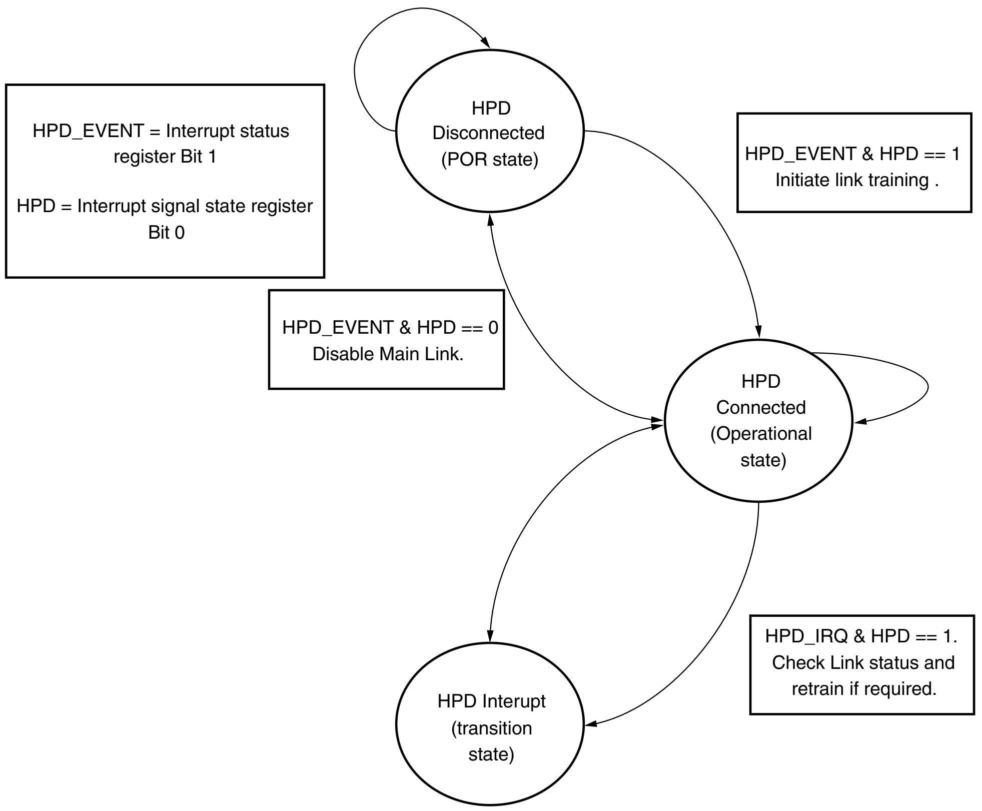

HPD_EVENT = Interrupt status register Bit 1
HPD = Interrupt signal state register Bit 0
HPD Disconnected (POR state)
HPD_EVENT & HPD == 1 Initiate link training .
HPD_EVENT & HPD == 0 Disable Main Link.
HPD Connected (Operational state)
HPD_IRQ & HPD == 1. Check Link status and retrain if required.
HPD Interupt (transition state)

Figure 3-7: **HPD Event Handling in Source Core**

Secondary Channel Operation

The current version of the DisplayPort IP supports 8-channel Audio. Secondary Channel features from the DisplayPort Standard v1.4 are supported.

The DisplayPort Audio IP core is offered as modules to provide flexibility and freedom to modify the system as needed. As shown in Figure 3-8, the Audio interface to the DisplayPort core is defined using an AXI4-Stream interface to improve system design and IP integration.

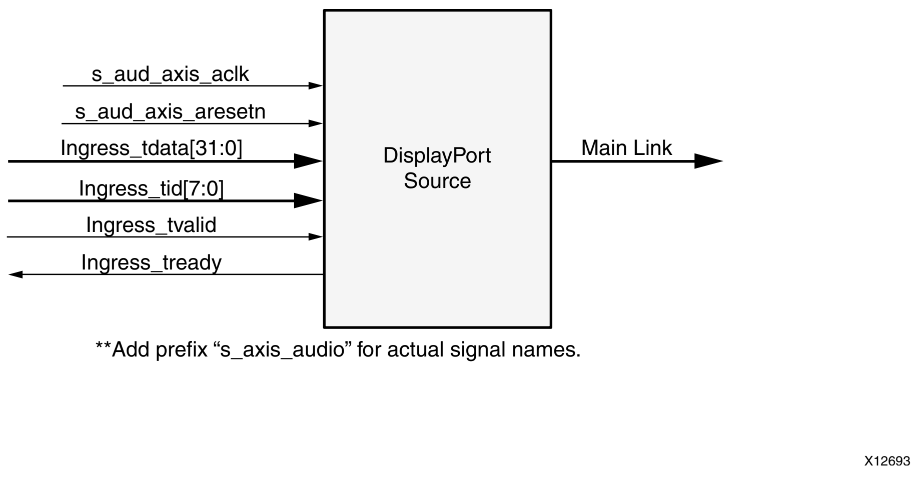


Figure 3-8: **Audio Data Interface of DisplayPort Source System**

32-bit AXI TDATA is formatted according as follows:

Control Bits + 24-bit Audio Sample + Preamble

The ingress channel buffer in the DisplayPort core accepts data from the AXI4-Stream interface based on buffer availability and audio control programming. A valid transfer takes place when `tready` and `tvalid` are asserted as described in the AXI4-Stream protocol. The ingress channel buffer acts as a holding buffer.

The DisplayPort Source has a fixed secondary packet length [Header = 4 Bytes + 4 Parity Bytes, Payload = 32 Sample Bytes + 8 Parity Bytes]. In a 1-2 channel transmission, the Source accumulates eight audio samples in the internal channel buffer, and then sends the packet to main link.

Multi Channel Audio

DisplayPort transmitter requires Info frame configuration to transmit multi-channel audio. The Info frame contains the number of channels and its speaker mapping. Streaming TID should contain the Audio channel ID along with audio data, based on the number of channels configured.

For multi-stream audio, secondary data packet ID in the Info frame packet should match with the stream ID over the audio AXI4-Stream interface (`TID[7:4]`).

Programming DisplayPort Source

1. Disable Audio by writing `0x00` to TX_AUDIO_CONTROL register. The disable bit also flushes the buffers in DisplayPort Source and sets the MUTE bit in VB-ID. Xilinx recommends following this step when there is a change in video/audio parameters.
2. Write Audio Info Frame (Based on your requirement. This might be optional for some systems.). Audio Info Frame consists of 8 writes. The order of write transactions are important and follow the steps mentioned in the DisplayPort Audio Registers, offset `0x308` (Table 2-6).
3. Write Channel Count to TX_AUDIO_CHANNELS register (the value is actual count -1).
4. If the system is using synchronous clocking then write MAUD and NAUD values to TX_AUDIO_MAUD and TX_AUDIO_NAUD registers, respectively.
5. Enable Audio by writing `0x01` to TX_AUDIO_CONTROL register.

Re-Programming Source Audio

1. Disable Audio in DisplayPort 1.4 TX core.
2. Wait until Video/Audio clock is recovered and stable.
3. Enable Audio in DisplayPort 1.4 TX core.
4. Wait for some time (in µs).

Info Packet Management

The core provides an option to program a single Info packet. The packet is transmitted to Sink once per every video frame or 8192 cycles.

To change an Info packet during transmission, follow these steps:

1. Disable Audio (Since new info packet means new audio configuration). The disable audio also flushes internal audio buffers.
2. Follow steps mentioned in Programming DisplayPort Source.

Extension Packet Management

A single packet buffer is provided for the extension packet. If the extension packet is available in the buffer, the packet is transmitted as soon as there is availability in the secondary channel. The packet length is FIXED to eight words (32 bytes). For VSC Ext packet to be aligned with the Vertical Blanking region, set Bit[12] of 0x1A4 register and then program the packet data.

Use the following steps to write an extended packet in the DisplayPort Source controller:

1. Write nine words (as required) into TX_AUDIO_EXT_DATA buffer.

2. Wait for EXT_PKT_TXD interrupt.
3. Write new packet (follow step 1).

Audio Clocking (Recommendation)

The system should have a clock generator (preferably programmable) to generate 512 × fs (Audio Sample Rate) clock frequency. The same clock (`aud_clk`) is used by DisplayPort Source device to calculate MAUD and NAUD when running in asynchronous clocking mode.

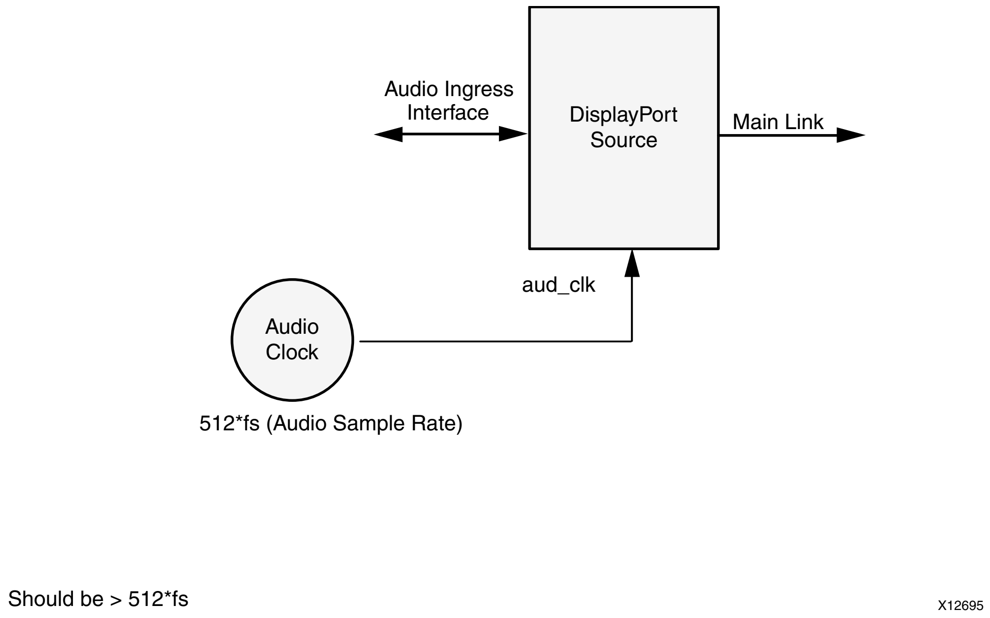


Figure 3-9: **Source: Audio Clocking**

Reduced Blanking

DisplayPort IP supports CVT standard RB and RB2 reduced blanking resolutions. As per the CVT specifications RB/RB2 resolution has HBLANK ≤ 20% HTOTAL, HBLANK = 80/160 and HRES%8 = 0.

For the CVT standard, RB/RB2 resolutions end of the line reset need to be disabled by setting the corresponding bit in the Line reset disable register (offset address 0x0F0 for transmitter). For the Non-CVT reduced blanking resolutions, where HRES is non multiple of 8, end of line reset is required to clear extra pixels in the video path for each line.

DisplayPort transmitter knows the resolution ahead of time hence reset disable can be done during initialization. In DisplayPort receiver when video mode change interrupt occurs the MSA registers can be read to know whether the resolution is reduced blanking or standard resolution and the corresponding bit can be set.

Clocking

This section describes the link clock (`tx_lnk_clk`) and the video clock (`tx_vid_clk_stream1`). The AXI4-Stream to Video bridge can handle asynchronous clocking. The value is based on the Consumer Electronics Association (CEA)/VESA Display Monitor Timing (DMT) standard for given video resolutions.

The `tx_lnk_clk` is a link clock input to the DisplayPort 1.4 TX Subsystem generated by the Video PHY (GT). The frequency of `tx_lnk_clk` is `<line_rate>`/20 MHz for 16-bit interface.

In native mode, the TX video clock has to be as per the value based on the Consumer Electronics Association (CEA)/VESA Display Monitor Timing (DMT) standard for given video resolutions.

The core uses six clock domains:

- **lnk_clk**: The `txoutclk` from the Video PHY is connected to the TX subsystem link clock. Most of the core operates in link clock domain. This domain is based on the `lnk_clk_p/n` reference clock for the transceivers. The link rate switching is handled by a DRP state machine in the core PHY later. When the lanes are running at 2.7 Gb/s, `lnk_clk` operates at 135 MHz. When the lanes are running at 1.62 Gb/s, `lnk_clk` operates at 81 MHz. When the lanes are running at 5.4 Gb/s, `lnk_clk` operates at 270 MHz.

  Note: `lnk_clk` = `link_rate`/20, when GT-Data width is 16-bit. `lnk_clk` = `link_rate`/40, when GT-Data width is 32-bit.
- **vid_clk**: This is the primary user interface clock. It has been tested to run as fast as 150 MHz, which accommodates to a screen resolution of 2560x1600 when using two-wide pixels and larger when using the four-wide pixels. Based on the *DisplayPort Standard*, the video clock can be derived from the link clock using `mvid` and `nvid`.
- **s_axi_aclk**: This is the processor domain. It has been tested to run as fast as 135 MHz. The AUX clock domain is derived from this domain, but requires no additional constraints. In UltraScale FPGA `s_axi_aclk` clock is connected to a free-running clock input. `gtwiz_reset_clk_freerun_in` is required by the reset controller helper block to reset the transceiver primitives. A new GUI parameter is added for AXI_Frequency, when the DisplayPort IP is targeted to UltraScale FPGA. The requirement is `s_axi_aclk` ≤ `lnk_clk`.
- **aud_clk**: This is the audio interface clock. The frequency will be equal to 512 × audio sample rate.
- **s_aud_axis_aclk**: This clock is used by the source audio streaming interface. This clock should be = 512 × audio sample rate.
- **m_aud_axis_aclk**: This clock is used by the sink audio streaming interface. This clock should be = 512 × audio sample rate.

Resets

The subsystem has one reset input for each of the AXI4-Lite, AXI4-Stream and Video interfaces:

- `s_axi_aresetn` – Active-Low AXI4-Lite reset. This resets all the programming registers.
- `tx_vid_reset_stream1` – Active-High video pipe reset.
- `s_axis_aresetn_stream1` – Active-Low AXI4-Stream interface reset.
- `m_aresetn_stream1` – Active-Low reset for streams one and two.

Address Map Example

Table 3-3 shows an example based on a subsystem base address of `0x44C0_0000` (19 bits). The DisplayPort 1.4 TX Subsystem requires a 19-bit address mapping, starting at an offset address of `0x00000`.

This address map example is applicable when TX subsystem is configured in the AXI4-Stream Interface mode.

Table 3-3: **Address Map Example**

	SST
DisplayPort 1.4 TX Core	0x44C0_0000
VTC 0	0x44C0_1000

Design Flow Steps

This chapter describes customizing and generating the 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 4]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7]

Customizing and Generating the Subsystem

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

If you are customizing and generating the subsystem in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 4] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the subsystem by specifying values for the various parameters associated with the subsystem IP cores using the following steps:

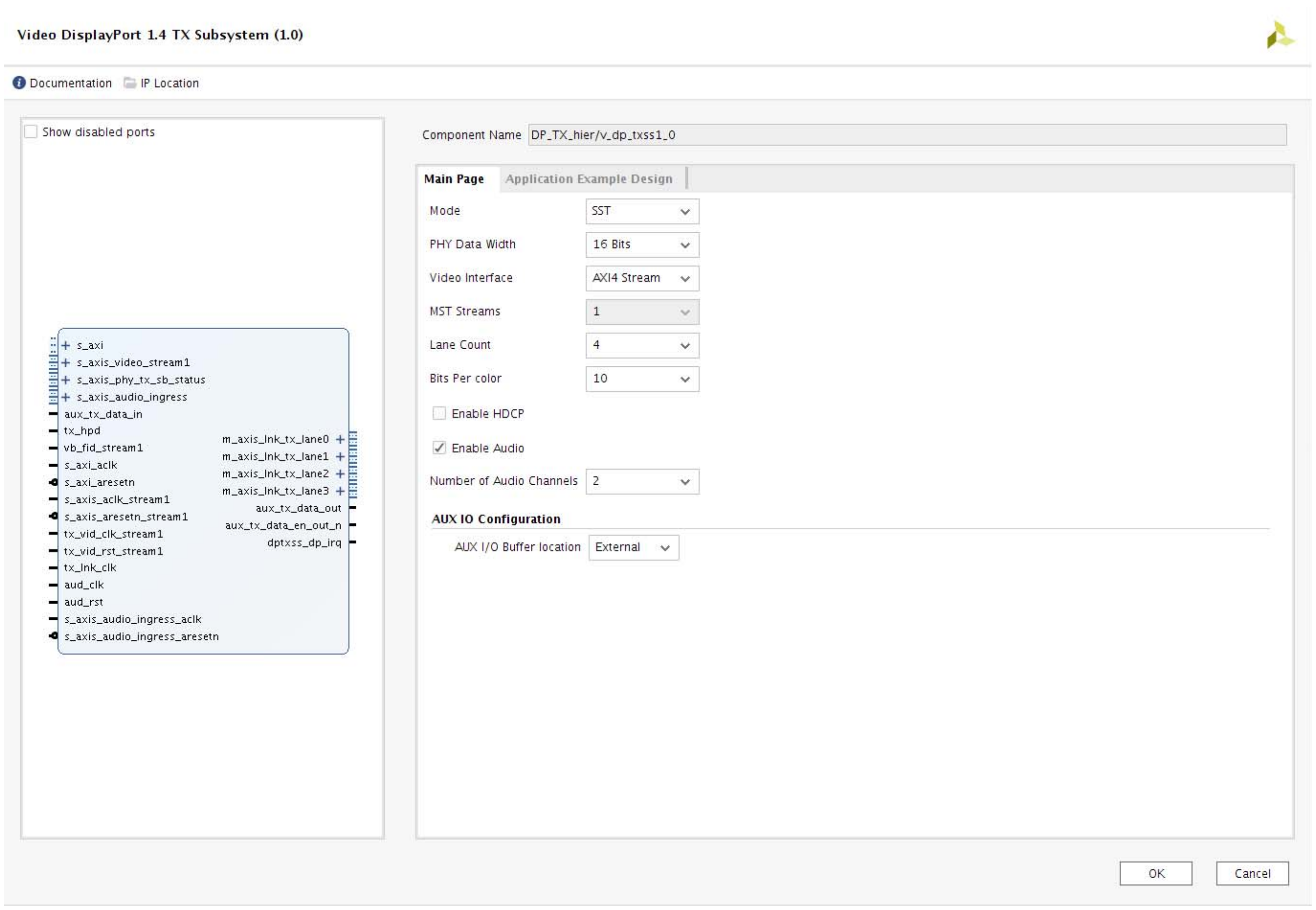
1. Select the subsystem from the IP catalog.
2. Double-click the selected subsystem or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6].

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

Customizing the IP

The configuration screen is shown in Figure 4-1.



Component Name: DP_TX_hier/v_dp_txss1_0

Mode: SST
PHY Data Width: 16 Bits
Video Interface: AXI4 Stream
MST Streams: 1
Lane Count: 4
Bits Per color: 10
Enable HDCP: [ ]
Enable Audio: [✓]
Number of Audio Channels: 2

AUX IO Configuration
AUX I/O Buffer location: External

Figure 4-1: **Configuration Screen**

- **Component Name –** The Component Name is used as the name of the top-level wrapper file for the core. The underlying netlist still retains its original name. Names must begin with a letter and must be composed from the following characters: a through z, 0 through 9, and "_". The name displayport_0 is used as internal module name and should not be used for the component name. The default is v_dp_txss1_0.
- **Mode –** Selects the desired resolution for the video stream out. Options are SST or MST.
- **PHY Data Width –** Selects the 16-bit GT data width.
- **Video Interface –** Selects the AXI4-Stream or native for the input video interface.
- **MST Streams –** Selects the maximum number of streams in MST mode (grayed out for this release).
- **Lane Count –** Selects the maximum number of lanes.

- **Bits Per Color –** Selects the desired maximum bit per component (BPC).
- **Enable HDCP –** Enables the HDCP encryption (grayed out for this release).
- **Enable Audio –** Enables the audio support.
- **Audio Channels –** Selects the number of audio channels.
- **AUX I/O Buffer location –** Selects the buffer location for AUX channel

User Parameters

Table 4-1 shows the relationship between the GUI fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl console). The line rate and pixel mode support in the DisplayPort 1.4 TX Subsystem is through software. Maximum pixel mode support is aligned to the lane count.

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

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Mode	MODE	SST
PHY Data Width	PHY_DATA_WIDTH	16
Video Interface	VIDEO_INTERFACE	AXI4 Stream
MST Streams(1)	NUM_STREAMS	1
Lane Count	LANE_COUNT	4
Bits Per Color	BITS_PER_COLOR	8
Enable HDCP(1)	HDCP_ENABLE	0
Enable Audio	AUDIO_ENABLE	0
Number Of Audio Channels	AUDIO_CHANNELS	2
Pixel Mode	PIXEL_MODE	1/2/4 (Valid only in native mode)
AUX I/O Buffer Location	AUX_IO_LOC	Internal

Notes:
1. MST and HDCP features are not supported and grayed out in the GUI.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5].

Constraining the Core

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

Required Constraints

There are no required constraints for this core.

Device, Package, and Speed Grade Selections

See IP Facts for details about supported devices.

Clock Frequencies

See Clocking in Chapter 3 for more details about clock frequencies. For more information on GT clocking, see the *Video PHY Controller Product Guide* (PG230) [Ref 1].

Clock Management

There are no specific clock management constraints.

Clock Placement

There are no specific clock placement constraints.

Banking

For more information on the specific banking constraints, see the *Video PHY Controller Product Guide* (PG230) [Ref 1].

Transceiver Placement

For more information on the specific transceiver placement constraints, see the *Video PHY Controller Product Guide* (PG230) [Ref 1].

I/O Standard and Placement

This section contains details about I/O constraints.

AUX Channel

The *VESA DisplayPort Standard* [Ref 3] describes the AUX channel as a bidirectional LVDS signal. For 7 series designs, the core uses IOBUFDS (bidirectional buffer) as the default with the LVDS standard. You should design the board as recommended by the VESA DP Protocol Standard. For reference, see the example design XDC file.

For UltraScale+™ and UltraScale™ families supporting HR IO banks, use the following constraints:

For Source:

```
set_property IOSTANDARD LVDS_25   [get_ports aux_tx_io_p]
set_property IOSTANDARD LVDS_25   [get_ports aux_tx_io_n]
```

For Sink:

```
set_property IOSTANDARD LVDS_25   [get_ports aux_rx_io_p]
set_property IOSTANDARD LVDS_25   [get_ports aux_rx_io_n]
```

For UltraScale+ and UltraScale families supporting HP IO banks, use the following constraints:

For Source:

```
set_property IOSTANDARD LVDS [get_ports aux_tx_io_p]
set_property IOSTANDARD LVDS [get_ports aux_tx_io_n]
```

For Sink:

```
set_property IOSTANDARD LVDS   [get_ports aux_rx_io_p]
set_property IOSTANDARD LVDS   [get_ports aux_rx_io_n]
```

HPD

The HPD signal can operate in either a 3.3V or 2.5V I/O bank. By definition in the standard, it is a 3.3V signal.

For UltraScale+ and UltraScale families supporting HR IO banks, use the following constraints:

```
set_property IOSTANDARD LVCMOS25 [get_ports hpd];
```

For UltraScale+ and UltraScale families supporting HP IO banks, use the following constraints:

```
set_property IOSTANDARD LVCMOS18 [get_ports hpd];
```

Board design and connectivity should follow *DisplayPort Standard* recommendations with proper level shifting.

Simulation

There is no example design simulation support for DisplayPort 1.4 TX Subsystem.

Synthesis and Implementation

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

Example Design

Note: All example designs use the Inrevium DP1.4 FMC card.

This chapter contains step-by-step instructions for generating an Application Example Design from the DisplayPort Subsystem by using the Vivado® Design Suite flow.

RECOMMENDED: *For ZCU102 (Revision 1.0 or later), you should set up a 1.8V setting after connecting the DisplayPort FMC. See the Setting the FMC Voltage to 1.8V section. For more information, see the ZCU102 System Controller – GUI Tutorial (XTP433) [Ref 16].*

Table 5-1 shows the available example designs for DisplayPort TX.

Table 5-1: **Available Example Designs**

GT Type	Topology	Video PHY Config		Hardware	GT Data Width	BPC	Processor
		(TXPLL)	(RXPLL)				
GTHE3	Pass-through without HDCP1.3	QPLL	CPLL	KCU105 + Inrevium DP1.4 FMC[1]	2-byte	8	MicroBlaze
GTHE4	RX only	–	CPLL	ZCU102 + Inrevium DP1.4 FMC[1]	2-byte	10	R5
	TX only	QPLL	–	ZCU102 + Inrevium DP1.4 FMC[1]	2-byte	10	R5

Notes:

1. Contact Xilinx Marketing for more information on DP1.4 FMC.

Building the Example Design

1. Open the Vivado Design Suite and click **Create Project** (Figure 5-1).

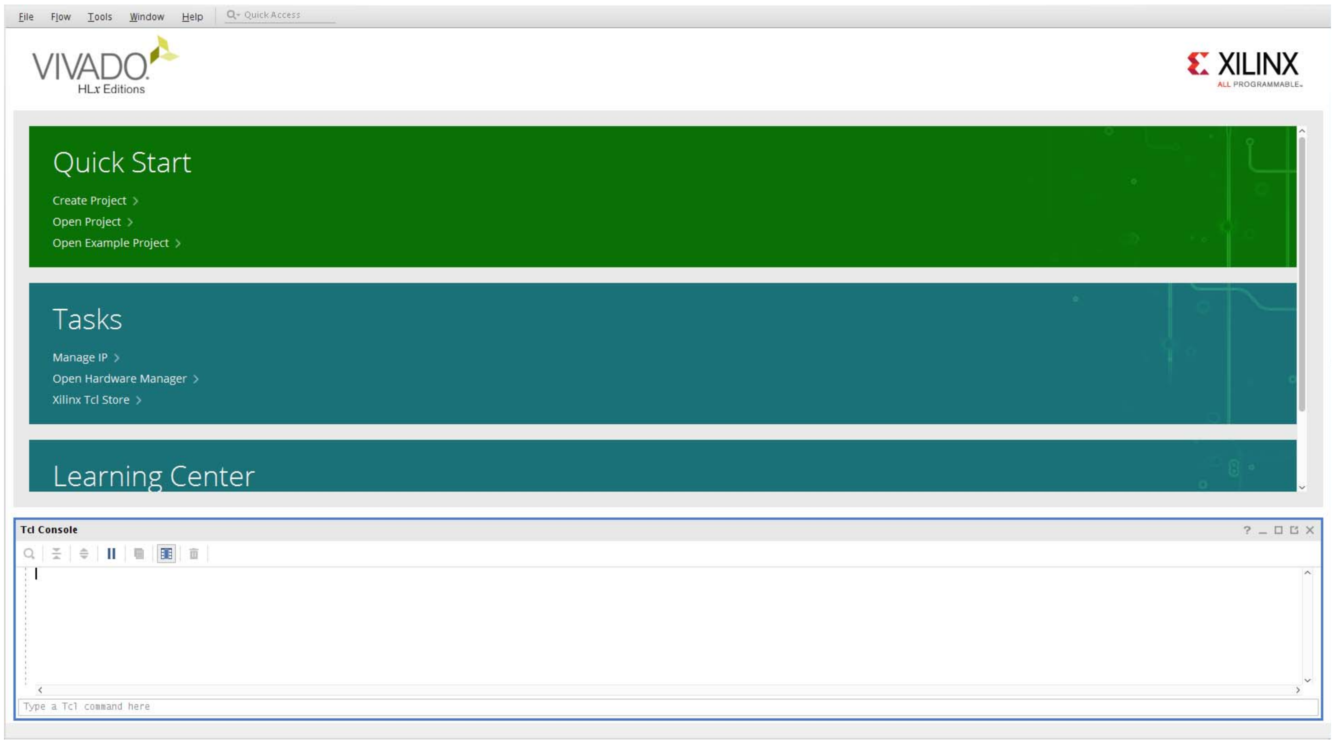
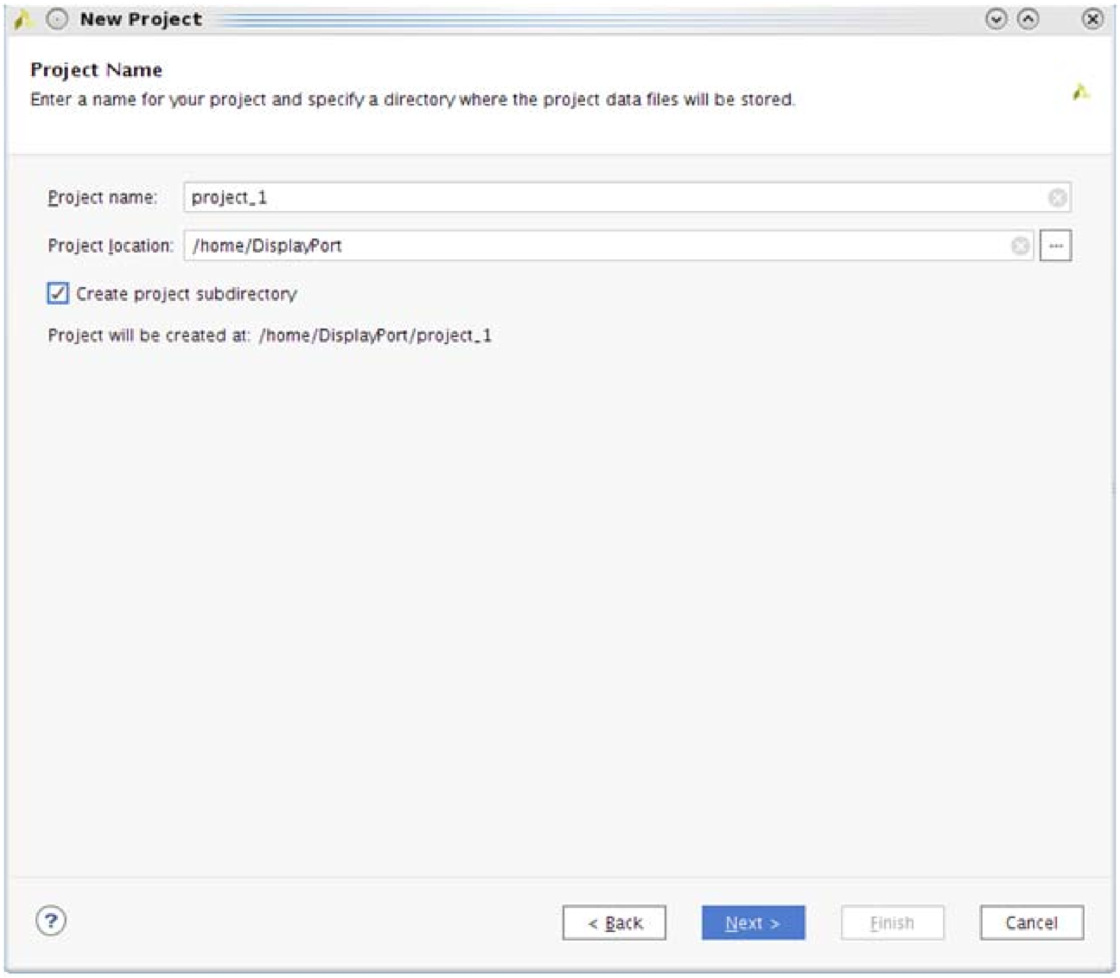


Figure 5-1: **Vivado Design Suite Quick Start**

2. In the **New Project** window (Figure 5-2), enter a **Project name**, **Project location**, and click **Next** up to the Board/Part selection window.


New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: project_1

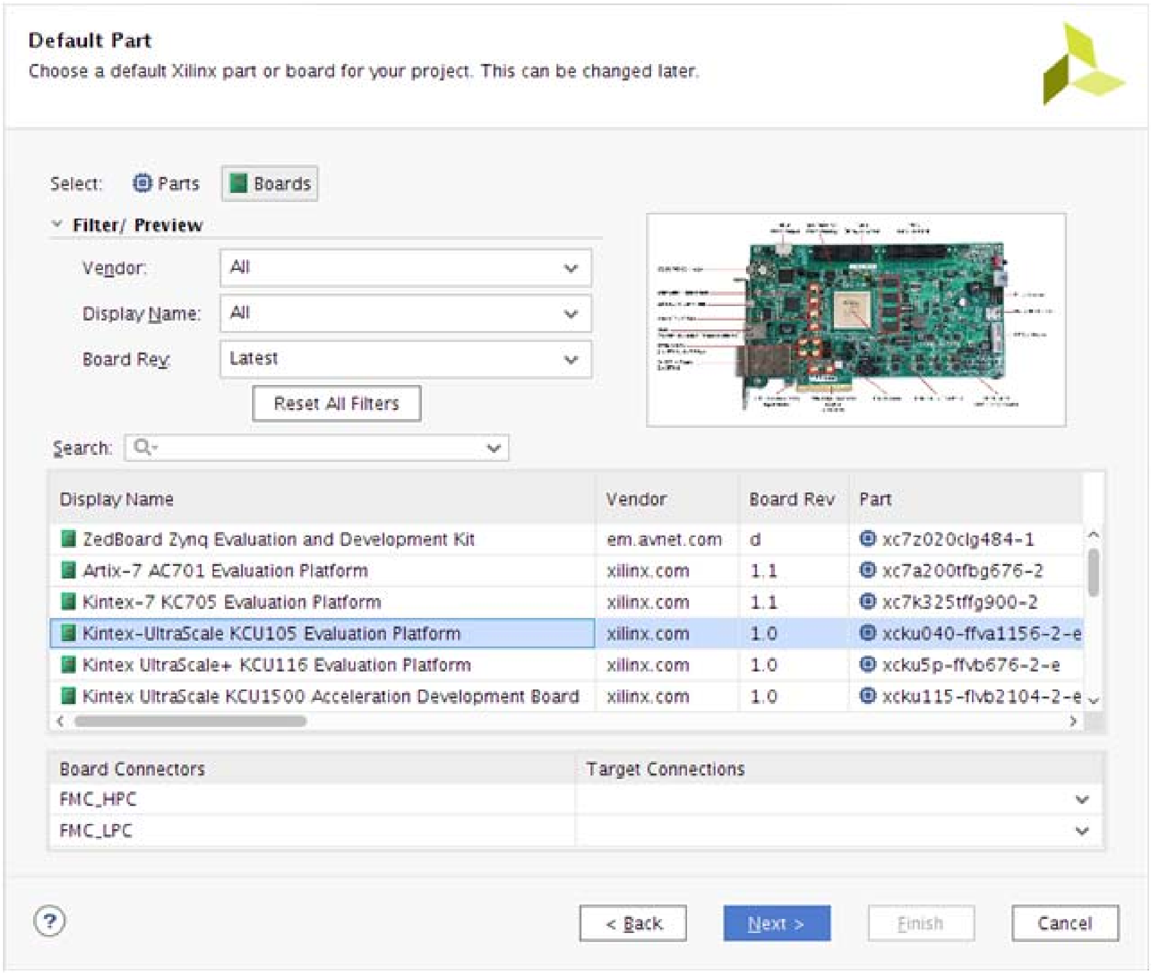
Project location: /home/DisplayPort

☑ Create project subdirectory

Project will be created at: /home/DisplayPort/project_1

Figure 5-2: **New Project**

3. In the **Default Part** window (Figure 5-3), select the Board as per your requirement. Application Example Designs are available for KCU105 and ZCU102.


Default Part

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

Select: Parts | Boards

Filter/ Preview

Vendor: All

Display Name: All

Board Rev: Latest

Reset All Filters

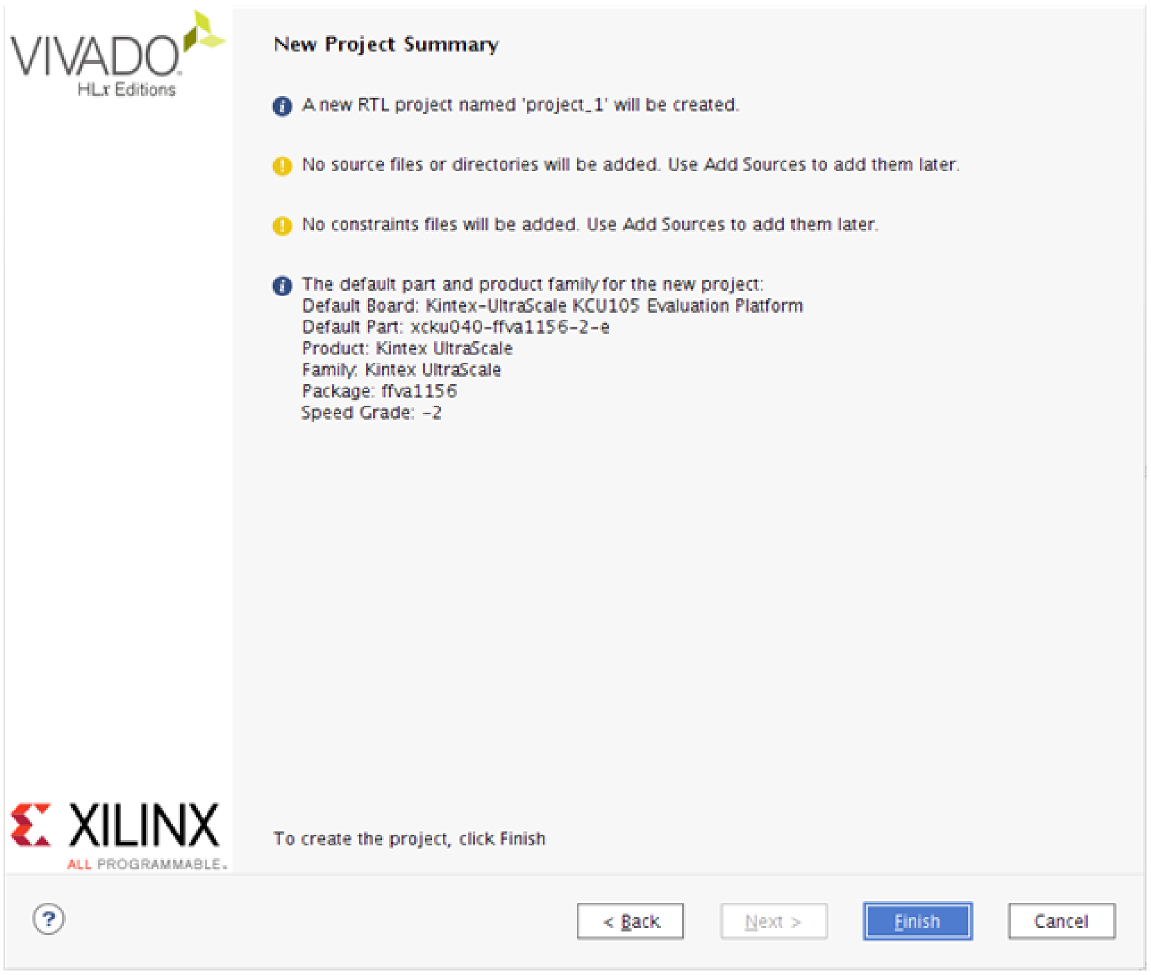
Search:

Display Name	Vendor	Board Rev	Part
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1
Artix-7 AC701 Evaluation Platform	xilinx.com	1.1	xc7a200tfbg676-2
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2
Kintex-UltraScale KCU105 Evaluation Platform	xilinx.com	1.0	xcku040-ffva1156-2-e
Kintex UltraScale+ KCU116 Evaluation Platform	xilinx.com	1.0	xcku5p-ffvb676-2-e
Kintex UltraScale KCU1500 Acceleration Development Board	xilinx.com	1.0	xcku115-flvb2104-2-e

Board Connectors	Target Connections
FMC_HPC	
FMC_LPC	

Figure 5-3: **Board Selection**

4. Click **Finish** (Figure 5-4).



New Project Summary

A new RTL project named 'project_1' will be created.

No source files or directories will be added. Use Add Sources to add them later.

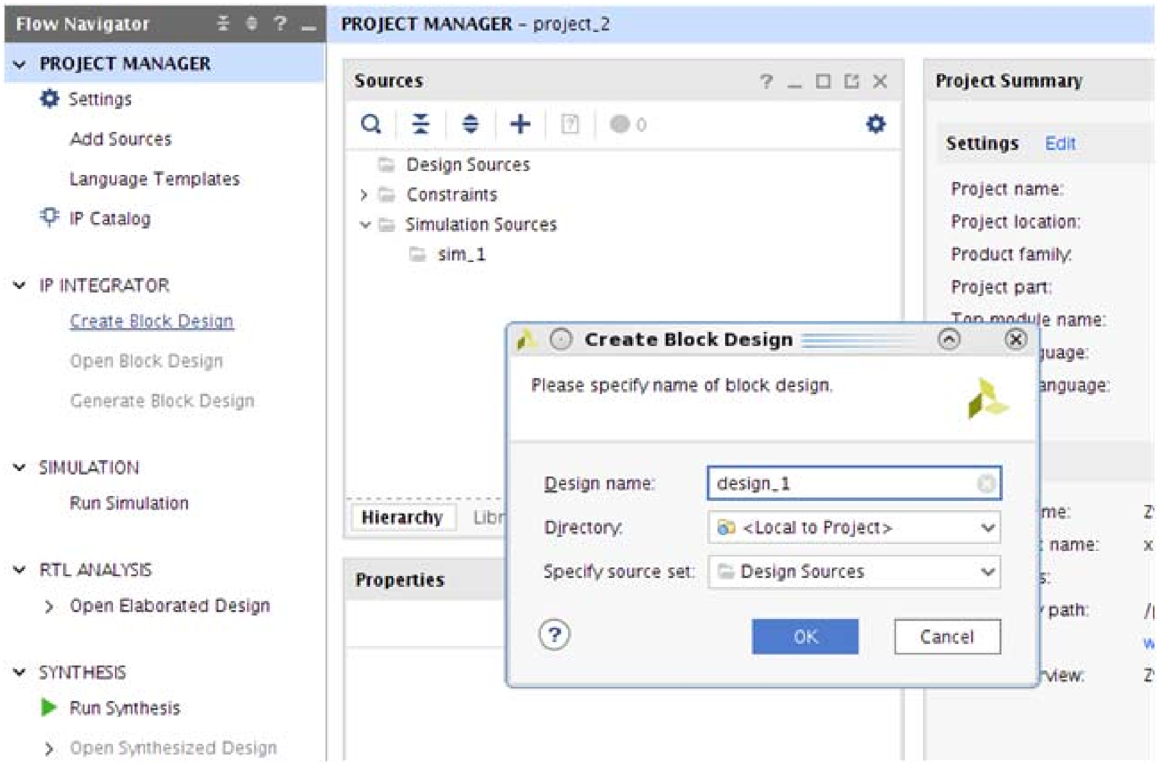
No constraints files will be added. Use Add Sources to add them later.

The default part and product family for the new project:
Default Board: Kintex-UltraScale KCU105 Evaluation Platform
Default Part: xcku040-ffva1156-2-e
Product: Kintex UltraScale
Family: Kintex UltraScale
Package: ffva1156
Speed Grade: -2

To create the project, click Finish

Figure 5-4: **New Project Summary**

5. In the Flow Navigator (Figure 5-5), click **Create Block Design** (BD). Select a name for BD and click **OK**.



Please specify name of block design.

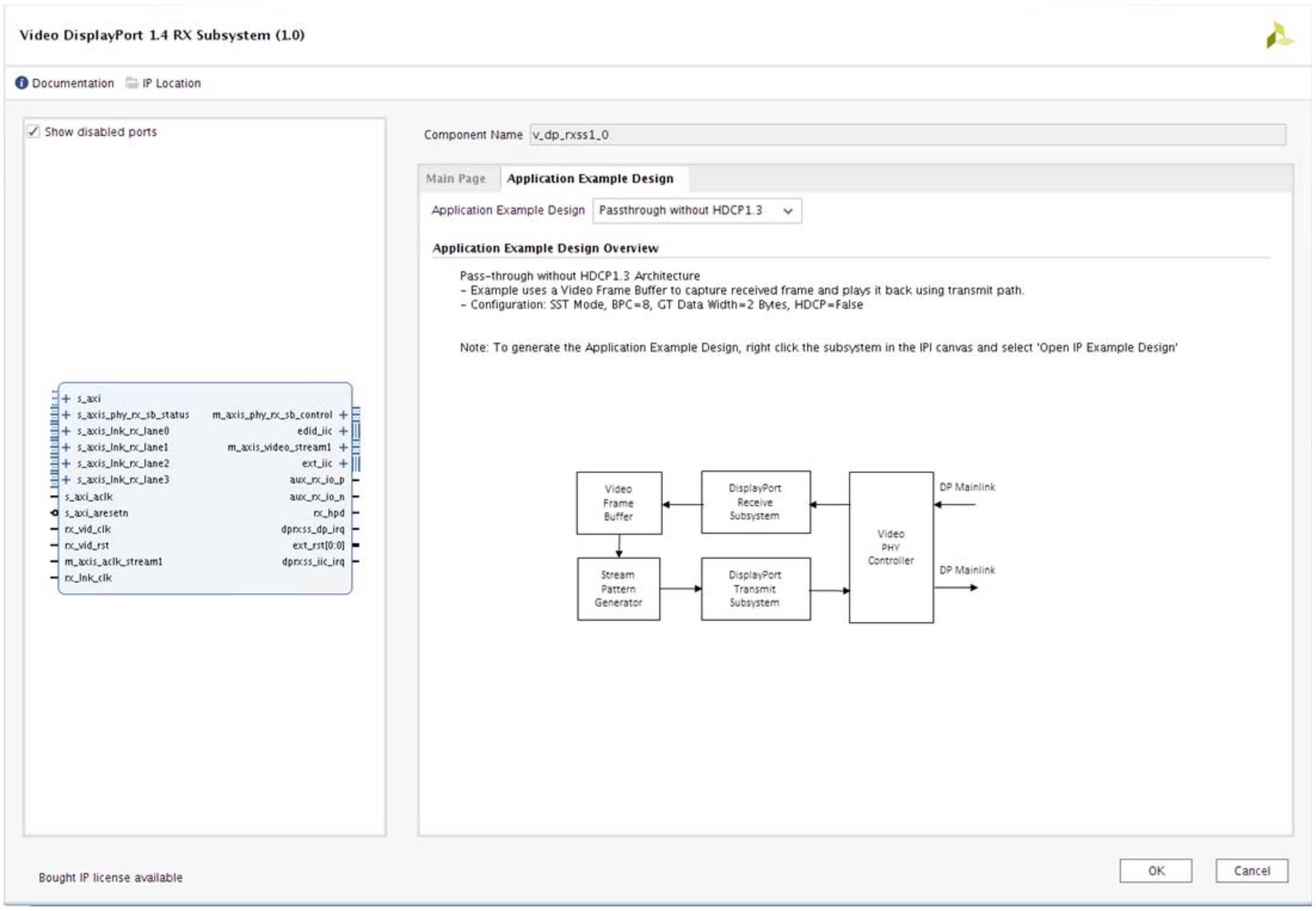
Design name: design_1

Directory: <Local to Project>

Specify source set: Design Sources

Figure 5-5: **Create Block Design**

6. Right-click BD and click **Add IP**. Search for DisplayPort 1.4 and select either the DisplayPort RX Subsystem IP (for RX only (ZCU102) or Pass-through (KCU105) designs) or the DisplayPort 1.4 TX Subsystem IP (for TX only (ZCU102) or Pass-through (KCU105) designs).
7. Double-click the IP and go to the **Application Example Design** tab in the **Customize IP** window (Figure 5-6). Select the supported topology in the **Application Example Design** drop-down box. Click **OK** and **Save** the block design.


Video DisplayPort 1.4 RX Subsystem (1.0)

Documentation | IP Location

✓ Show disabled ports

Component Name: v_dp_rxss1_0

Main Page | Application Example Design

Application Example Design: Passthrough without HDCP1.3

Application Example Design Overview

Pass-through without HDCP1.3 Architecture
- Example uses a Video Frame Buffer to capture received frame and plays it back using transmit path.
- Configuration: SST Mode, BPC=8, GT Data Width=2 Bytes, HDCP=False

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

Bought IP license available

Figure 5-6: **Application Example Design Topology**

8. Right-click the **DisplayPort Subsystem** IP under Design source in the **Design** tab and click **Open IP Example Design** (Figure 5-7).

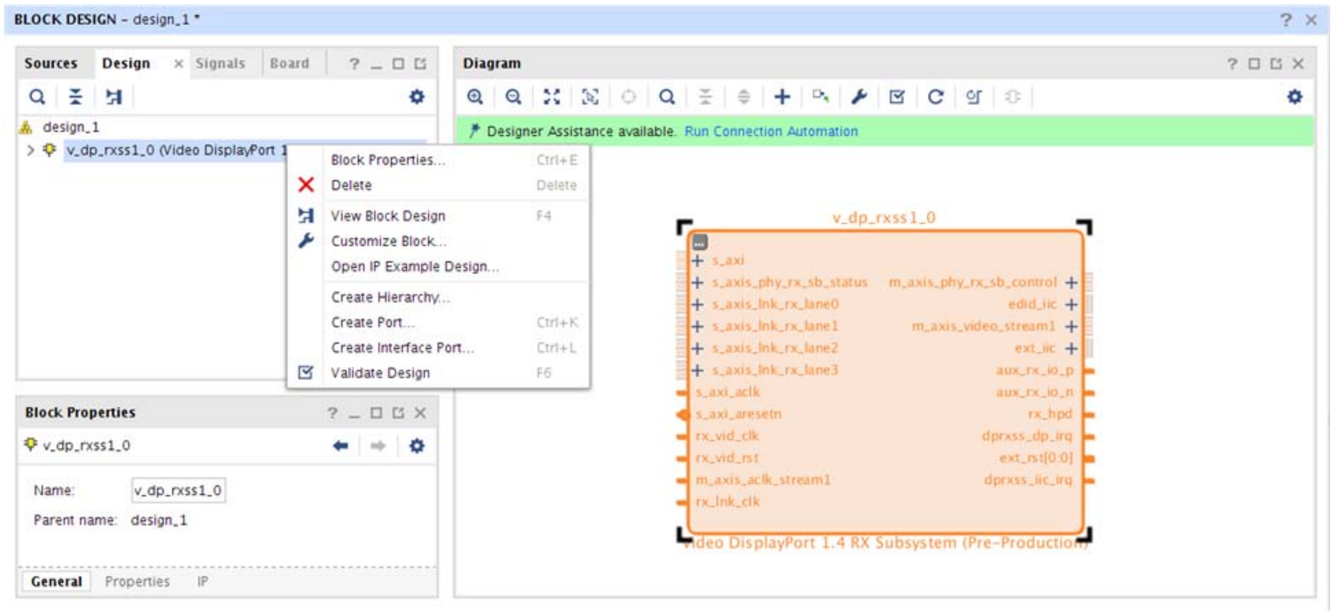
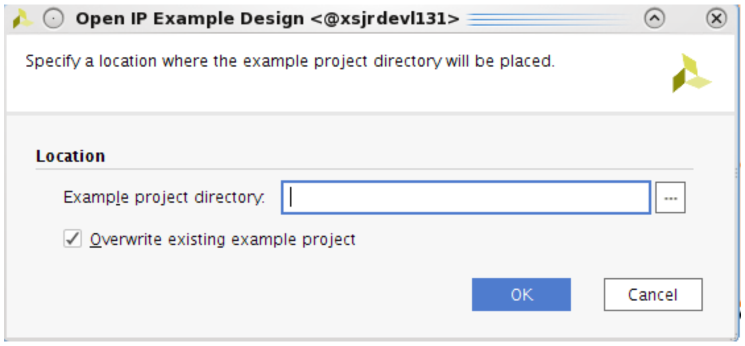


Figure 5-7: **Open IP Example Design**

9. Choose **Example project directory** (Figure 5-8) and click **OK**.



Specify a location where the example project directory will be placed.

Location

Example project directory:

☑ Overwrite existing example project

Figure 5-8: **Example Project Directory**

10. Figure 5-9 shows the Vivado IP integrator design. Choose the **Generate Bitstream**.

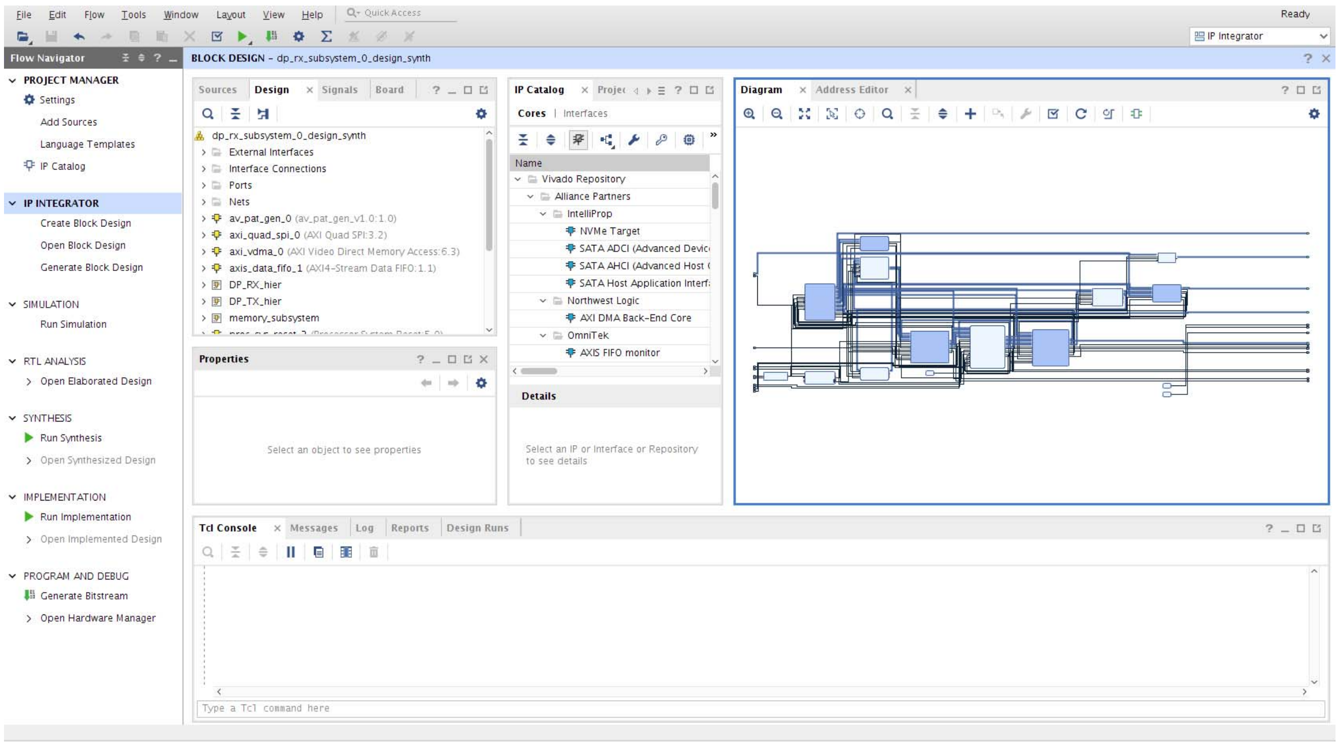


Figure 5-9: **IP Integrator Design**

11. Export the hardware to SDK. Click **File > Export > Export Hardware** (Figure 5-10).

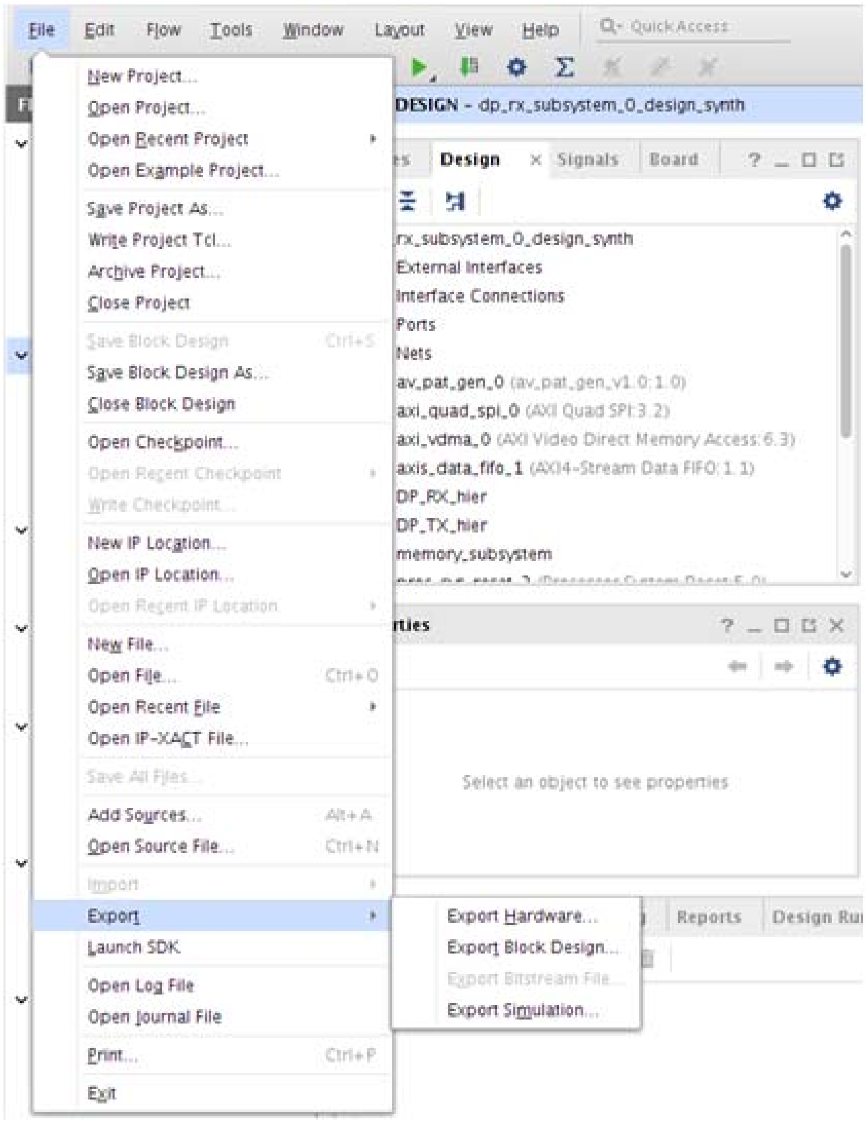
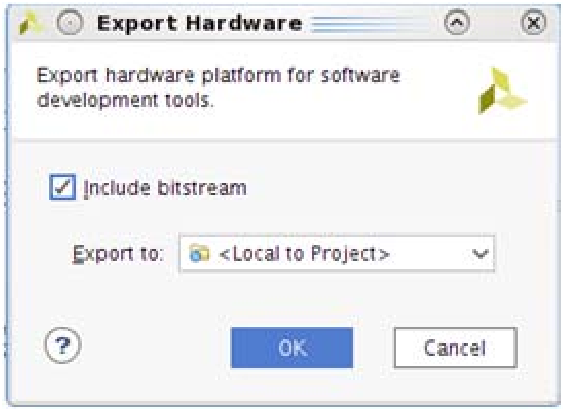


Figure 5-10: **Export Hardware for SDK Example Design Flow**

12. Ensure the **Include bitstream** is enabled and click **OK** (use the default **Export Location <Local to Project>**) (Figure 5-11).



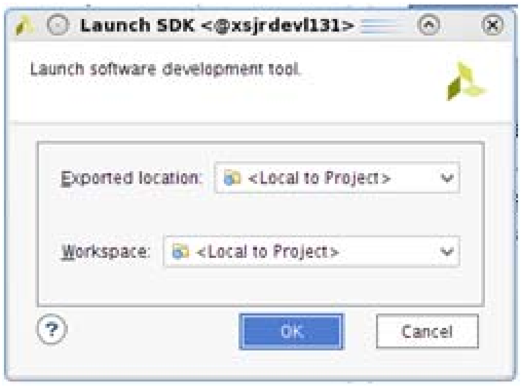
Export hardware platform for software development tools.

☑ Include bitstream

Export to: <Local to Project>

Figure 5-11: **Export Hardware**

13. Click **File > Launch SDK**. Choose the SDK Workspace location. Keep the exported location default configuration (**<Local to Project>**) (Figure 5-12).



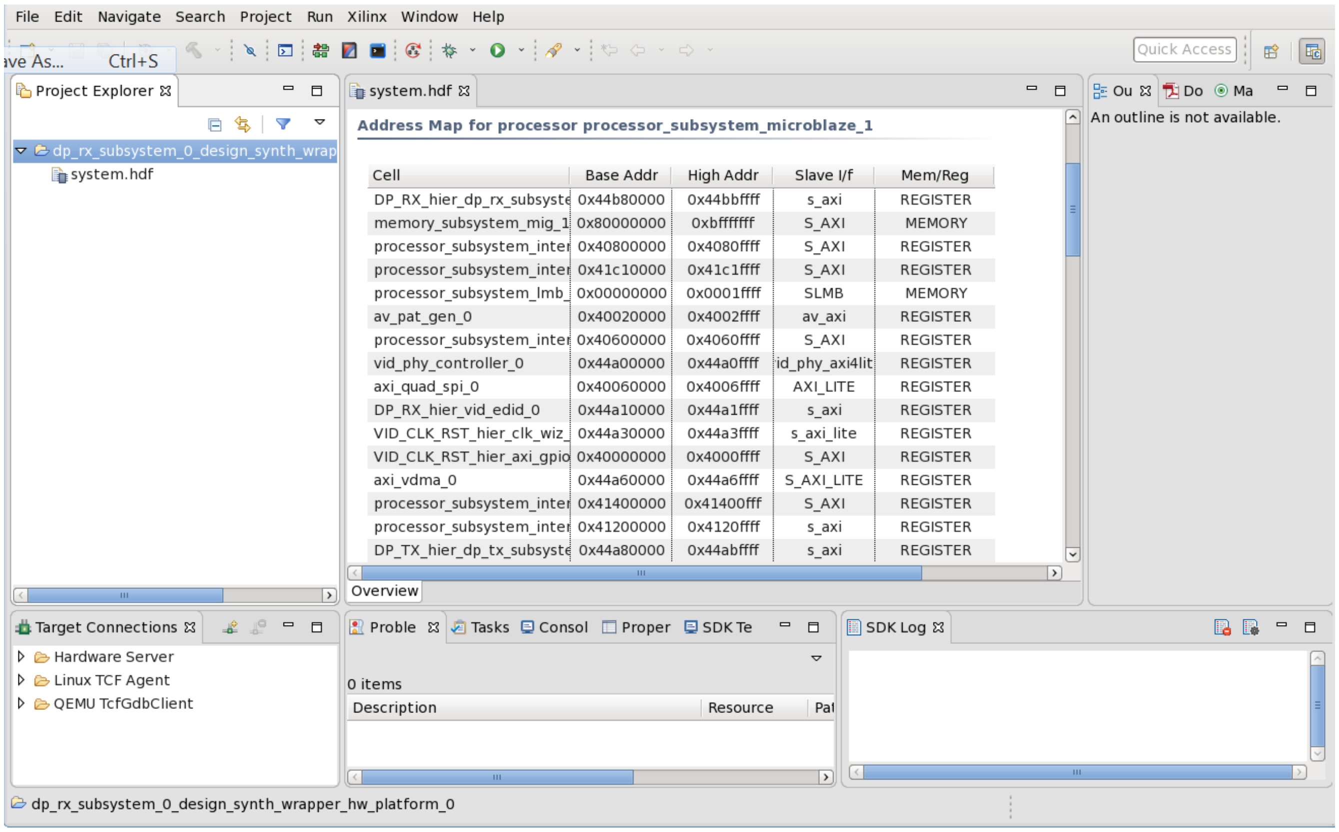
Launch software development tool.

Exported location: <Local to Project>

Workspace: <Local to Project>

Figure 5-12: **Launch SDK**

14. Figure 5-13 shows an example of the launched Vivado SDK.



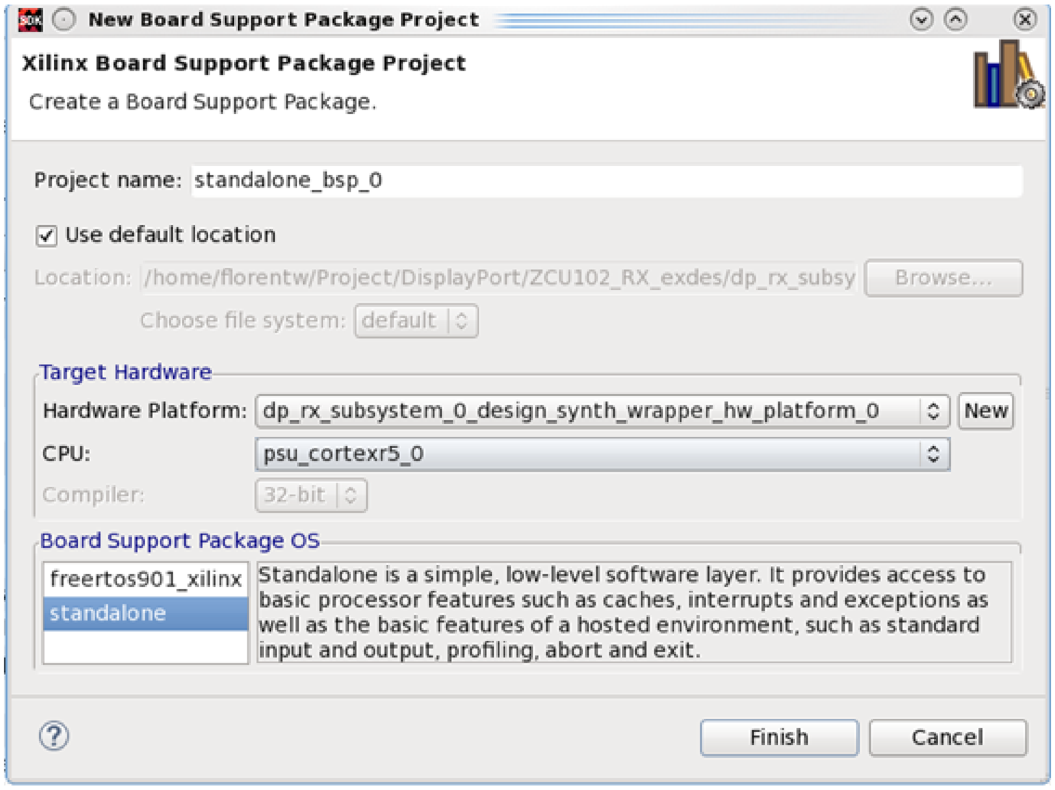
An outline is not available.

Address Map for processor processor_subsystem_microblaze_1

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
DP_RX_hier_dp_rx_subsyste	0x44b80000	0x44bbffff	s_axi	REGISTER
memory_subsystem_mig_1	0x80000000	0xbfffffff	S_AXI	MEMORY
processor_subsystem_inter	0x40800000	0x4080ffff	S_AXI	REGISTER
processor_subsystem_inter	0x41c10000	0x41c1ffff	S_AXI	REGISTER
processor_subsystem_lmb_	0x00000000	0x0001ffff	SLMB	MEMORY
av_pat_gen_0	0x40020000	0x4002ffff	av_axi	REGISTER
processor_subsystem_inter	0x40600000	0x4060ffff	S_AXI	REGISTER
vid_phy_controller_0	0x44a00000	0x44a0ffff	id_phy_axi4lit	REGISTER
axi_quad_spi_0	0x40060000	0x4006ffff	AXI_LITE	REGISTER
DP_RX_hier_vid_edid_0	0x44a10000	0x44a1ffff	s_axi	REGISTER
VID_CLK_RST_hier_clk_wiz_	0x44a30000	0x44a3ffff	s_axi_lite	REGISTER
VID_CLK_RST_hier_axi_gpio	0x40000000	0x4000ffff	S_AXI	REGISTER
axi_vdma_0	0x44a60000	0x44a6ffff	S_AXI_LITE	REGISTER
processor_subsystem_inter	0x41400000	0x41400fff	S_AXI	REGISTER
processor_subsystem_inter	0x41200000	0x4120ffff	s_axi	REGISTER
DP_TX_hier_dp_tx_subsyste	0x44a80000	0x44abffff	s_axi	REGISTER

Figure 5-13: **Vivado SDK**

15. To create a Board Support Package (BSP), click **File > New > Board Support Package**. Enter the BSP **Project name**, click **Finish** (Figure 5-14), and then **OK**. For ZCU102 board, ensure the target CPU is the Cortex R5_0 (`psu_cortexr5_0`)


New Board Support Package Project

Xilinx Board Support Package Project

Create a Board Support Package.

Project name: standalone_bsp_0

☑ Use default location

Location: /home/florentw/Project/DisplayPort/ZCU102_RX_exdes/dp_rx_subsy

Choose file system: default

Target Hardware

Hardware Platform: dp_rx_subsystem_0_design_synth_wrapper_hw_platform_0

CPU: psu_cortexr5_0

Compiler: 32-bit

Board Support Package OS

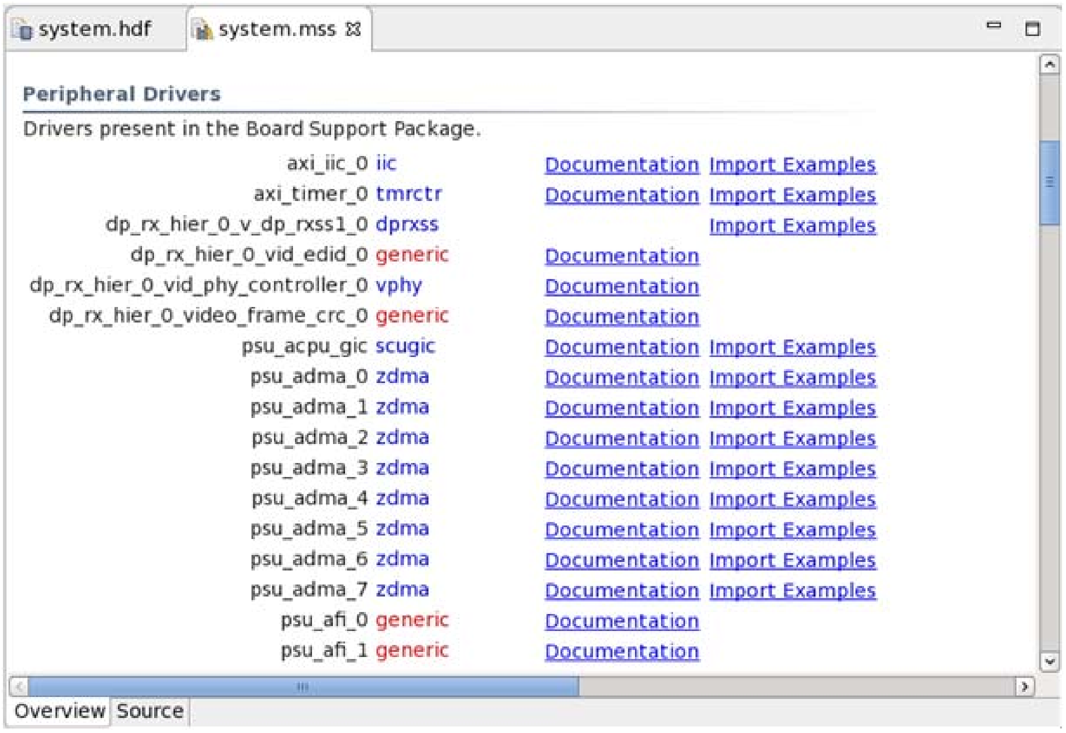
freertos901_xilinx

standalone

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

Figure 5-14: **New Board Support Package Project**

16. Find DisplayPort RX/TX Subsystem Driver in the `system.mss` file (Figure 5-15). If it is not, open the file from the BSP in the **Project Explorer**. Click **Import Examples** (Figure 5-16).



system.hdf | system.mss

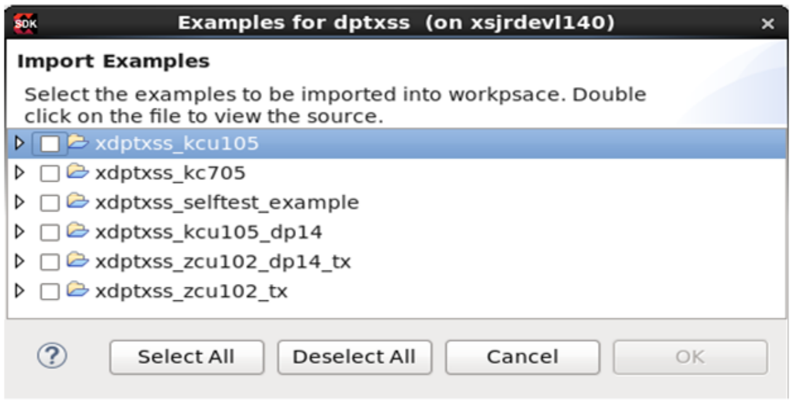
Peripheral Drivers

Drivers present in the Board Support Package.

axi_iic_0 iic	Documentation Import Examples
axi_timer_0 tmrctr	Documentation Import Examples
dp_rx_hier_0_v_dp_rxss1_0 dprxss	Import Examples
dp_rx_hier_0_vid_edid_0 generic	Documentation
dp_rx_hier_0_vid_phy_controller_0 vphy	Documentation
dp_rx_hier_0_video_frame_crc_0 generic	Documentation
psu_acpu_gic scugic	Documentation Import Examples
psu_adma_0 zdma	Documentation Import Examples
psu_adma_1 zdma	Documentation Import Examples
psu_adma_2 zdma	Documentation Import Examples
psu_adma_3 zdma	Documentation Import Examples
psu_adma_4 zdma	Documentation Import Examples
psu_adma_5 zdma	Documentation Import Examples
psu_adma_6 zdma	Documentation Import Examples
psu_adma_7 zdma	Documentation Import Examples
psu_afi_0 generic	Documentation
psu_afi_1 generic	Documentation

Overview Source

Figure 5-15: **system.mss**



Examples for dptxss (on xsjrdevl140)

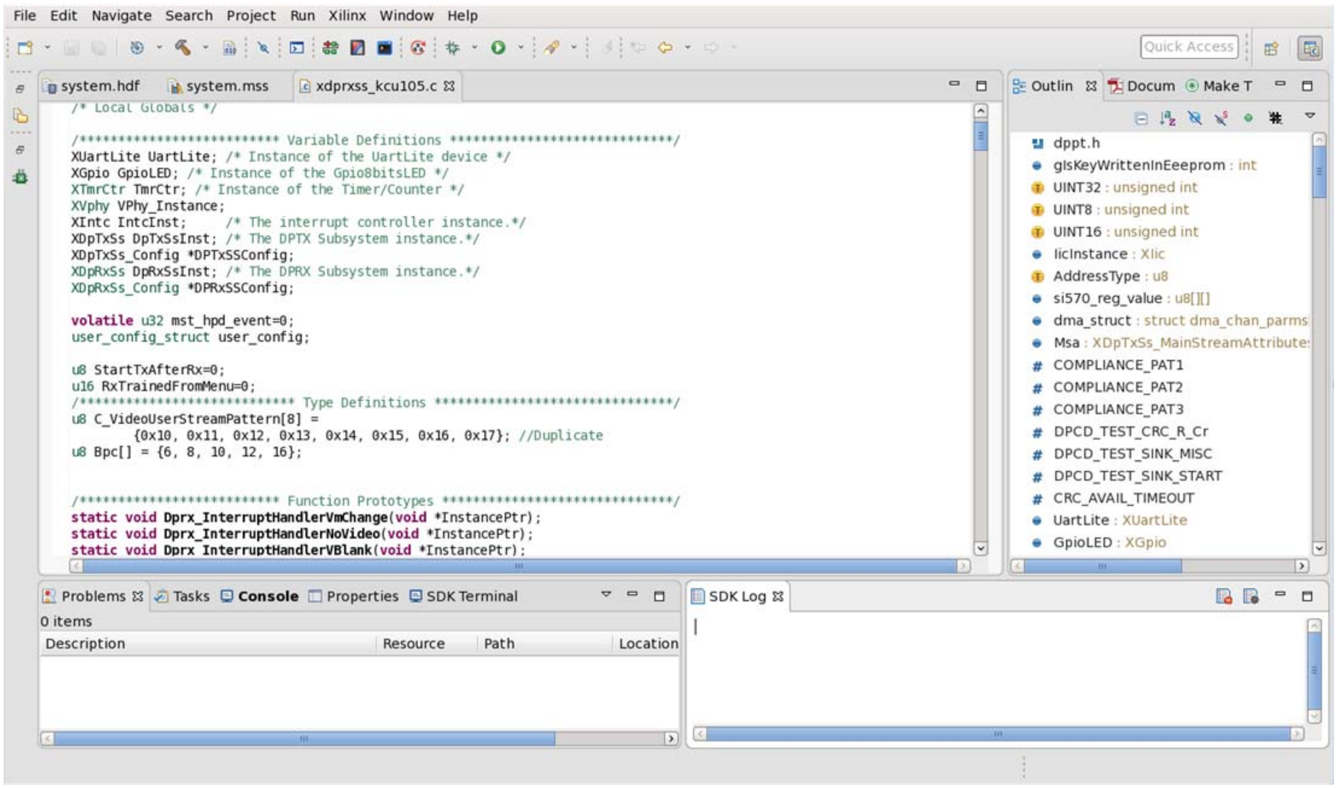
Import Examples

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

- ☐ xdptxss_kcu105
- ☐ xdptxss_kc705
- ☐ xdptxss_selftest_example
- ☐ xdptxss_kcu105_dp14
- ☐ xdptxss_zcu102_dp14_tx
- ☐ xdptxss_zcu102_tx

Figure 5-16: **Import Examples**

17. Select the Example Application corresponding to your hardware:
    - For Pass-through KCU105 project, select `*_kcu105_dp14` option.
    - For RX only ZCU102 project, select `*_zcu102_dp14_rxonly` option in RX Subsystem Driver.
    - For TX only ZCU102 project, select `*_zcu102_dp14_txonly` option in TX Subsystem Driver.
18. Figure 5-17 shows the example application successfully built and ready to use.



```
/* Local Globals */

/************************** Variable Definitions *****************************/
XUartLite UartLite; /* Instance of the UartLite device */
XGpio GpioLED; /* Instance of the Gpio8bitsLED */
XTmrCtr TmrCtr; /* Instance of the Timer/Counter */
XVphy VPhy_Instance;
XIntc IntcInst;     /* The interrupt controller instance.*/
XDpTxSs DpTxSsInst; /* The DPTX Subsystem instance.*/
XDpTxSs_Config *DPTxSSConfig;
XDpRxSs DpRxSsInst; /* The DPRX Subsystem instance.*/
XDpRxSs_Config *DPRxSSConfig;

volatile u32 mst_hpd_event=0;
user_config_struct user_config;

u8 StartTxAfterRx=0;
u16 RxTrainedFromMenu=0;
/***************************** Type Definitions *****************************/
u8 C_VideoUserStreamPattern[8] =
        {0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17}; //Duplicate
u8 Bpc[] = {6, 8, 10, 12, 16};


/************************** Function Prototypes *****************************/
static void Dprx_InterruptHandlerVmChange(void *InstancePtr);
static void Dprx_InterruptHandlerNoVideo(void *InstancePtr);
static void Dprx_InterruptHandlerVBlank(void *InstancePtr);
```

```
dppt.h
glsKeyWrittenInEeeprom : int
UINT32 : unsigned int
UINT8 : unsigned int
UINT16 : unsigned int
IicInstance : XIic
AddressType : u8
si570_reg_value : u8[][]
dma_struct : struct dma_chan_parms
Msa : XDpTxSs_MainStreamAttribute:
COMPLIANCE_PAT1
COMPLIANCE_PAT2
COMPLIANCE_PAT3
DPCD_TEST_CRC_R_Cr
DPCD_TEST_SINK_MISC
DPCD_TEST_SINK_START
CRC_AVAIL_TIMEOUT
UartLite : XUartLite
GpioLED : XGpio
```

Figure 5-17: **Successful Application Example Design**

Hardware Setup and Run

1. Connect the Tokyo Electron Device Limited (TED) TB-FMCH-DP3 module to the HPC FMC connector on the KCU105 board or to the HPC0 connector on the ZCU102 depending on your design.
2. Connect a USB cable (Type A to mini B) from the host PC to the USB UART port on the KCU105 for serial communication. In the case of KCU105 or ZCU102, use Type A to micro B type of USB cable.
3. Connect a JTAG USB Platform cable or a USB Type A to Micro B cable from the host PC to the board for programming bit and `elf` files.
4. For the pass-through or TX only applications, connect a DP cable from the TX port of the TED TB-FMCH-DP-3 module to a monitor, as shown in Figure 5-18.
5. For the pass-through or RX only applications, connect a DP cable from the RX port of the TED TB-FMCH-DP-3 module to a DP source (GPU), as shown in Figure 5-18.

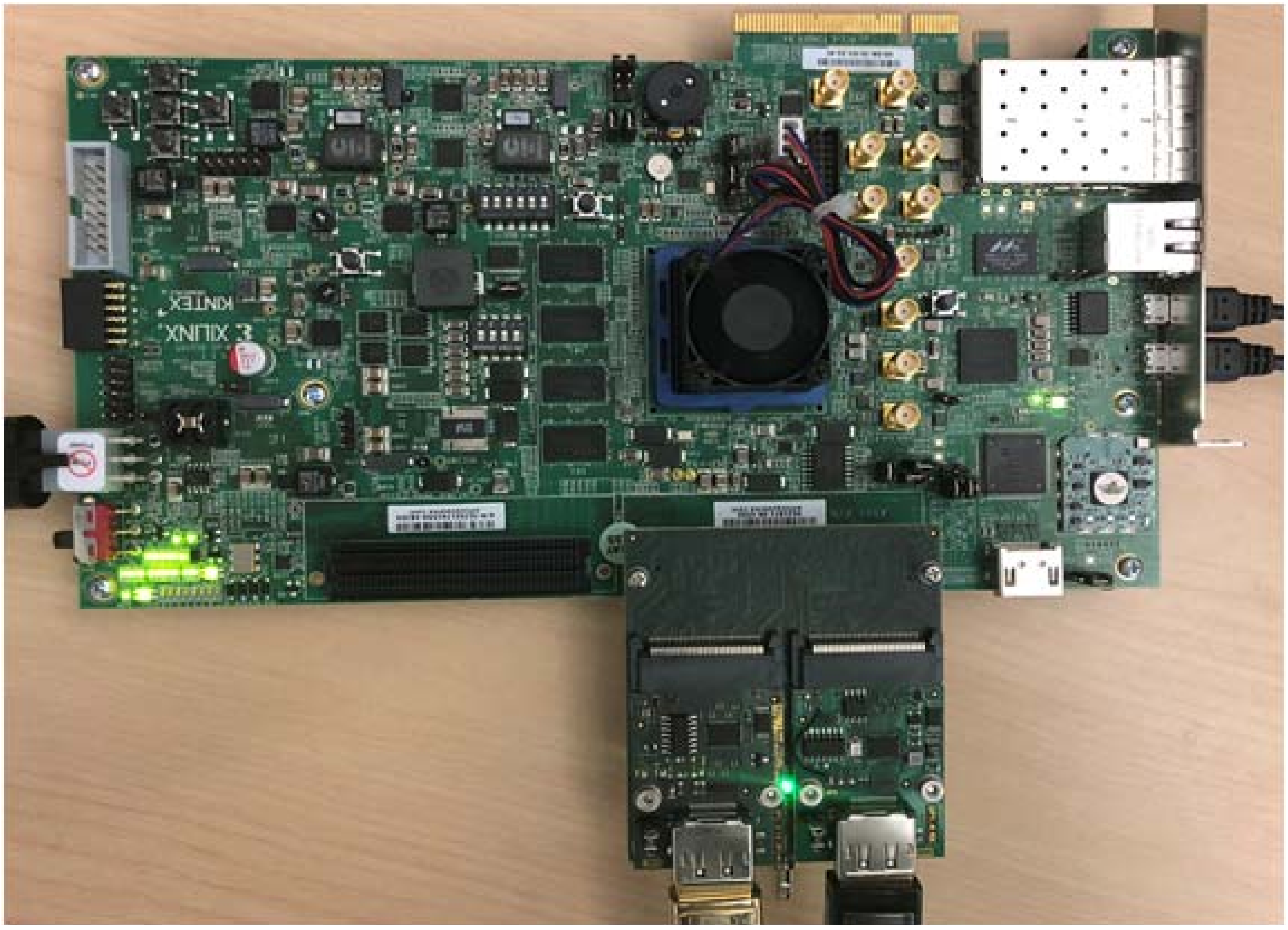


Figure 5-18: **KCU105 Board Setup**

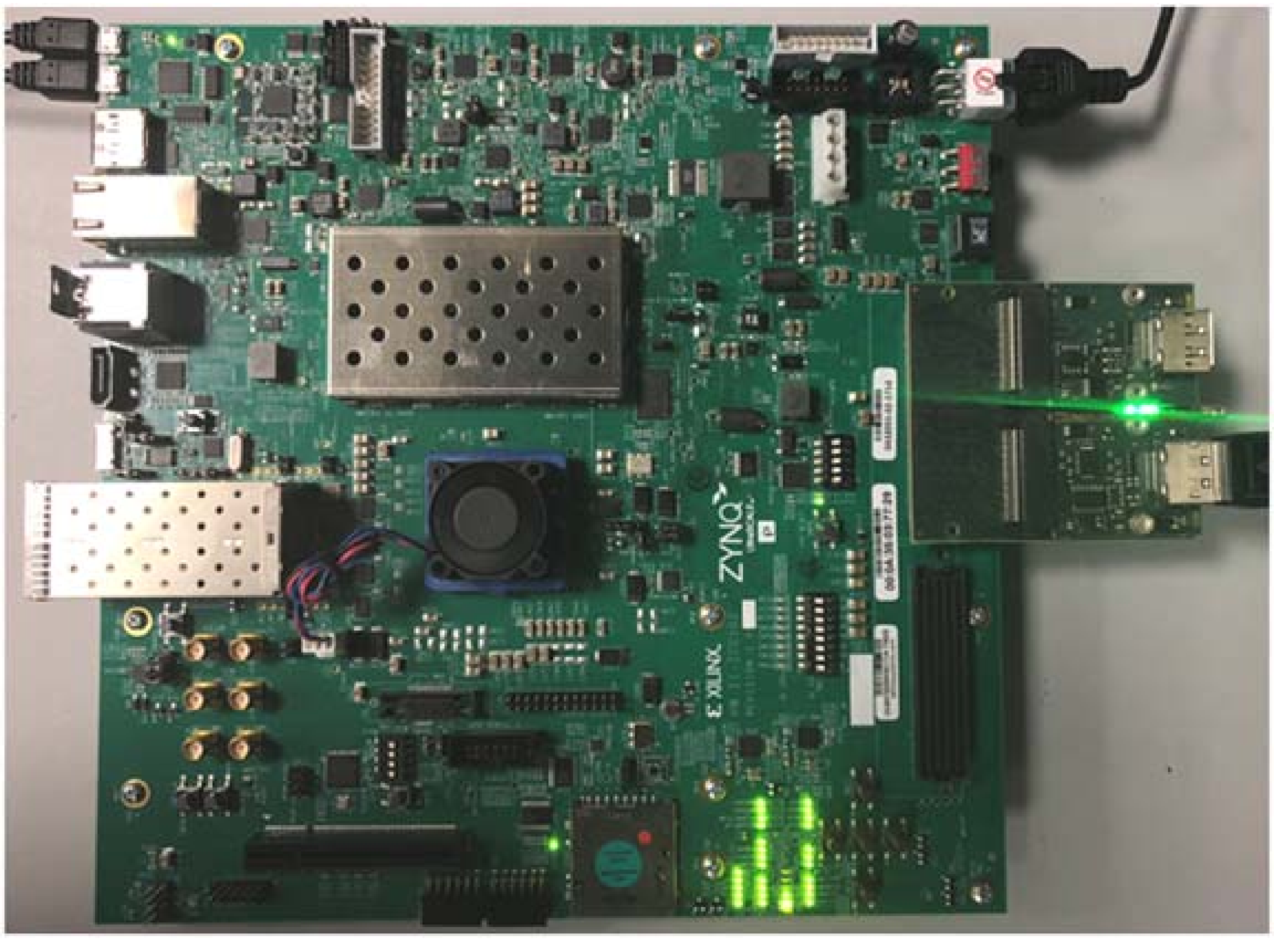


Figure 5-19: **ZCU102 Board Setup**

6. Set the mode pin to SW15:

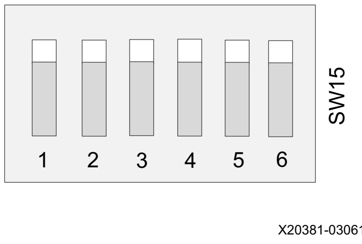


Figure 5-20: **SW15 in 111111 Position on KCU105**

Set the mode pin to SW6:

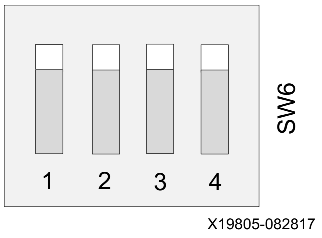
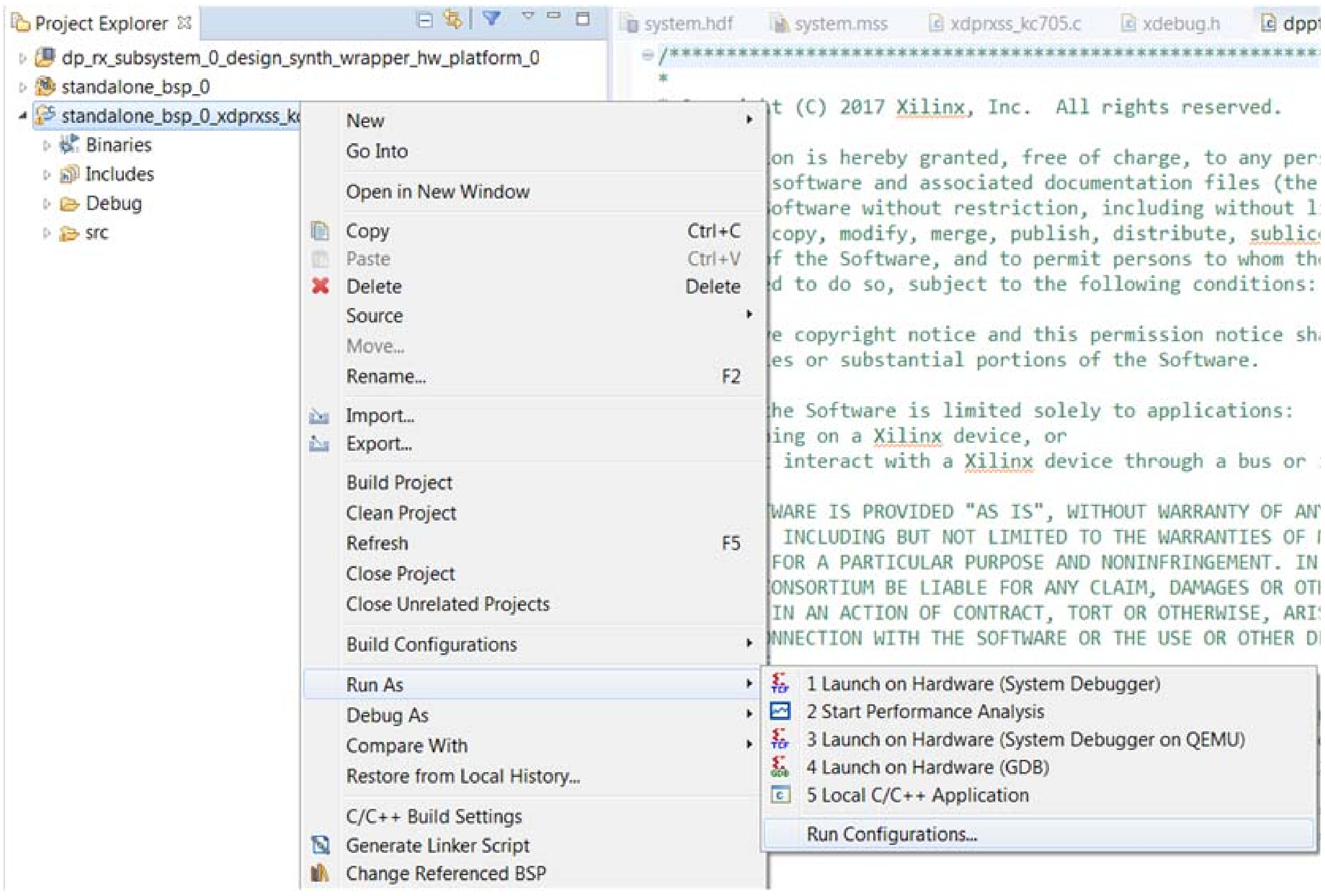


Figure 5-21: **SW6 in 1111 Position on ZCU102**

7. Connect the power supply and power on the board.
8. Start an UART terminal program such as Tera Term or Putty with the following settings:
    a. Baud rate = 115200
    b. Data bits = 8
    c. Parity = none
    d. Stop bits = 1
    e. Flow Control = none

    Note: With the ZCU102 board, there are four COM ports available.

9. In the Vivado SDK, under the **Project Explorer**, right-click the application and click **Run As > Run Configurations** (Figure 5-22).



```
/*************************************************************
 *
        t (C) 2017 Xilinx, Inc.  All rights reserved.

        on is hereby granted, free of charge, to any per
        software and associated documentation files (the
        oftware without restriction, including without l
        copy, modify, merge, publish, distribute, sublic
        f the Software, and to permit persons to whom th
        d to do so, subject to the following conditions:

        e copyright notice and this permission notice sh
        es or substantial portions of the Software.

        he Software is limited solely to applications:
        ing on a Xilinx device, or
         interact with a Xilinx device through a bus or

        WARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF AN
         INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
        FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN
        ONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OT
        IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARI
        NNECTION WITH THE SOFTWARE OR THE USE OR OTHER D
```

Figure 5-22: **Project Explorer**

10. In the **Run Configurations** popup menu, right-click **Xilinx C/C++ application (System Debugger)** and click **New** (Figure 5-23).

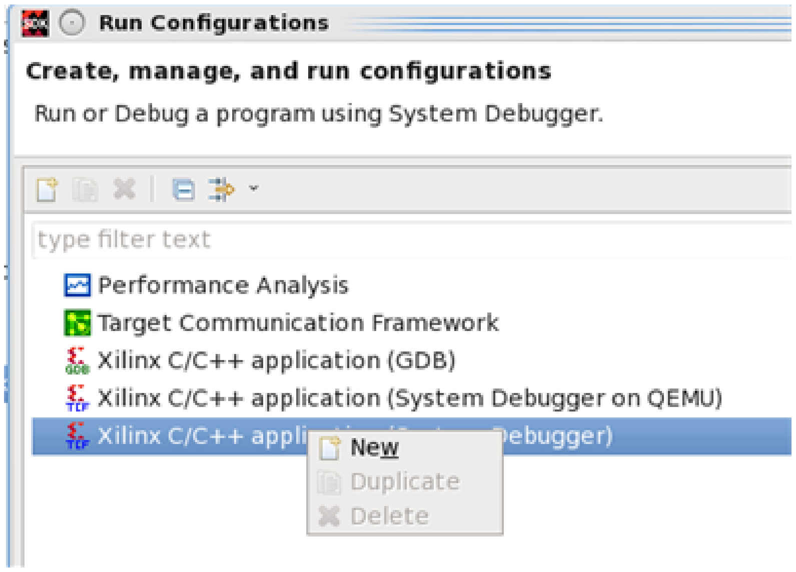
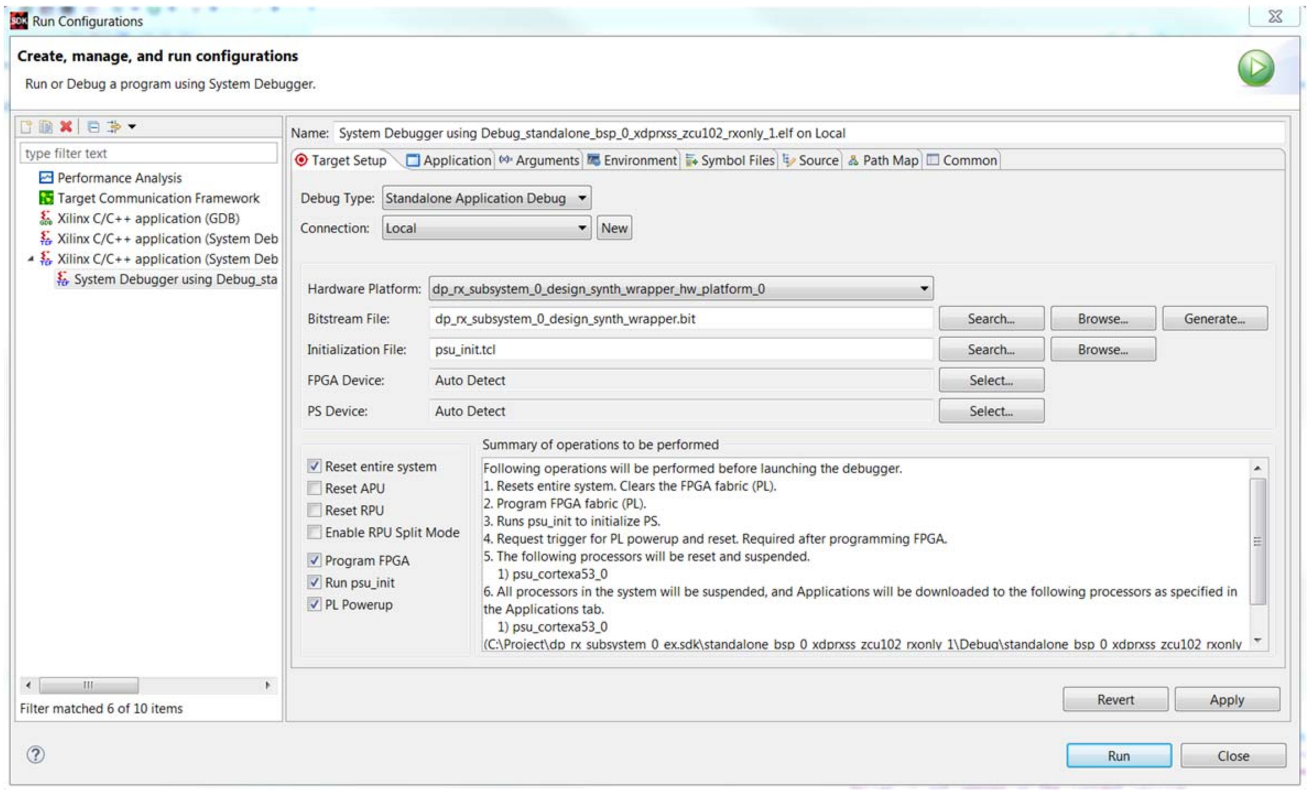

Run Configurations
Create, manage, and run configurations
Run or Debug a program using System Debugger.
type filter text
Performance Analysis
Target Communication Framework
Xilinx C/C++ application (GDB)
Xilinx C/C++ application (System Debugger on QEMU)
Xilinx C/C++ application (System Debugger)
New
Duplicate
Delete

Figure 5-23: **Run Configurations**

11. In the **Target Setup** tab (Figure 5-24), ensure the Connection is set to **Local** and the **Reset entire system** and **Program FPGA** are enabled. If running the ZCU102, also ensure that **Run psu_init** and **PL Powerup** are enabled.


Run Configurations
Create, manage, and run configurations
Run or Debug a program using System Debugger.
type filter text
Performance Analysis
Target Communication Framework
Xilinx C/C++ application (GDB)
Xilinx C/C++ application (System Deb
Xilinx C/C++ application (System Deb
System Debugger using Debug_sta
Filter matched 6 of 10 items
Target Setup
Application
Arguments
Environment
Symbol Files
Source
Path Map
Common
Search...
Browse...
Generate...
Select...
Revert
Apply
Run
Close

Name: System Debugger using Debug_standalone_bsp_0_xdprxss_zcu102_rxonly_1.elf on Local

Debug Type: Standalone Application Debug

Connection: Local

Hardware Platform: dp_rx_subsystem_0_design_synth_wrapper_hw_platform_0

Bitstream File: dp_rx_subsystem_0_design_synth_wrapper.bit

Initialization File: psu_init.tcl

FPGA Device: Auto Detect

PS Device: Auto Detect

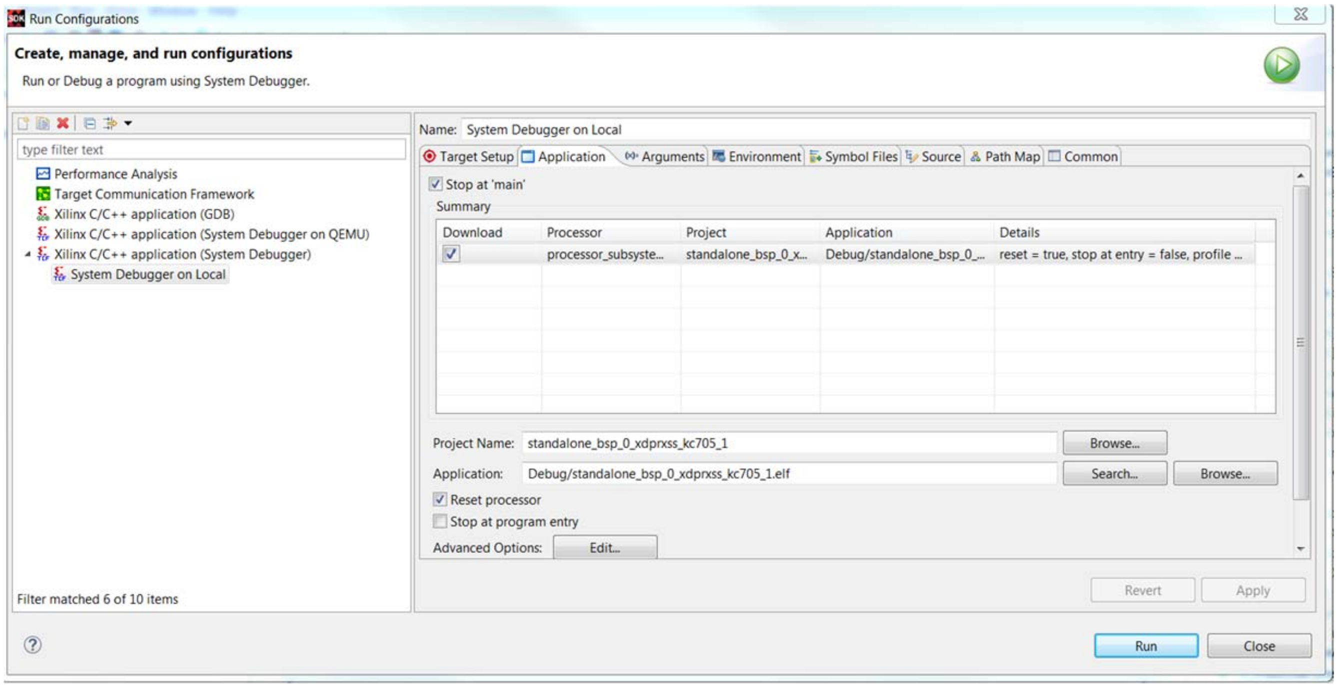
- [✓] Reset entire system
- [ ] Reset APU
- [ ] Reset RPU
- [ ] Enable RPU Split Mode
- [✓] Program FPGA
- [✓] Run psu_init
- [✓] PL Powerup

Summary of operations to be performed

```
Following operations will be performed before launching the debugger.
1. Resets entire system. Clears the FPGA fabric (PL).
2. Program FPGA fabric (PL).
3. Runs psu_init to initialize PS.
4. Request trigger for PL powerup and reset. Required after programming FPGA.
5. The following processors will be reset and suspended.
   1) psu_cortexa53_0
6. All processors in the system will be suspended, and Applications will be downloaded to the following processors as specified in
the Applications tab.
   1) psu_cortexa53_0
(C:\Project\dp_rx_subsystem_0_ex.sdk\standalone_bsp_0_xdprxss_zcu102_rxonly_1\Debug\standalone_bsp_0_xdprxss_zcu102_rxonly
```

Figure 5-24: **Target Setup**

12. In the **Application** tab (Figure 5-25), ensure the application download is enabled and click **Run**.


Run Configurations
Create, manage, and run configurations
Run or Debug a program using System Debugger.
type filter text
Performance Analysis
Target Communication Framework
Xilinx C/C++ application (GDB)
Xilinx C/C++ application (System Debugger on QEMU)
Xilinx C/C++ application (System Debugger)
System Debugger on Local
Filter matched 6 of 10 items
Target Setup
Application
Arguments
Environment
Symbol Files
Source
Path Map
Common
Browse...
Search...
Browse...
Edit...
Revert
Apply
Run
Close

Name: System Debugger on Local

☑ Stop at 'main'

Summary

Download	Processor	Project	Application	Details
☑	processor_subsyste...	standalone_bsp_0_x...	Debug/standalone_bsp_0_...	reset = true, stop at entry = false, profile ...

Project Name: standalone_bsp_0_xdprxss_kc705_1

Application: Debug/standalone_bsp_0_xdprxss_kc705_1.elf

☑ Reset processor

☐ Stop at program entry

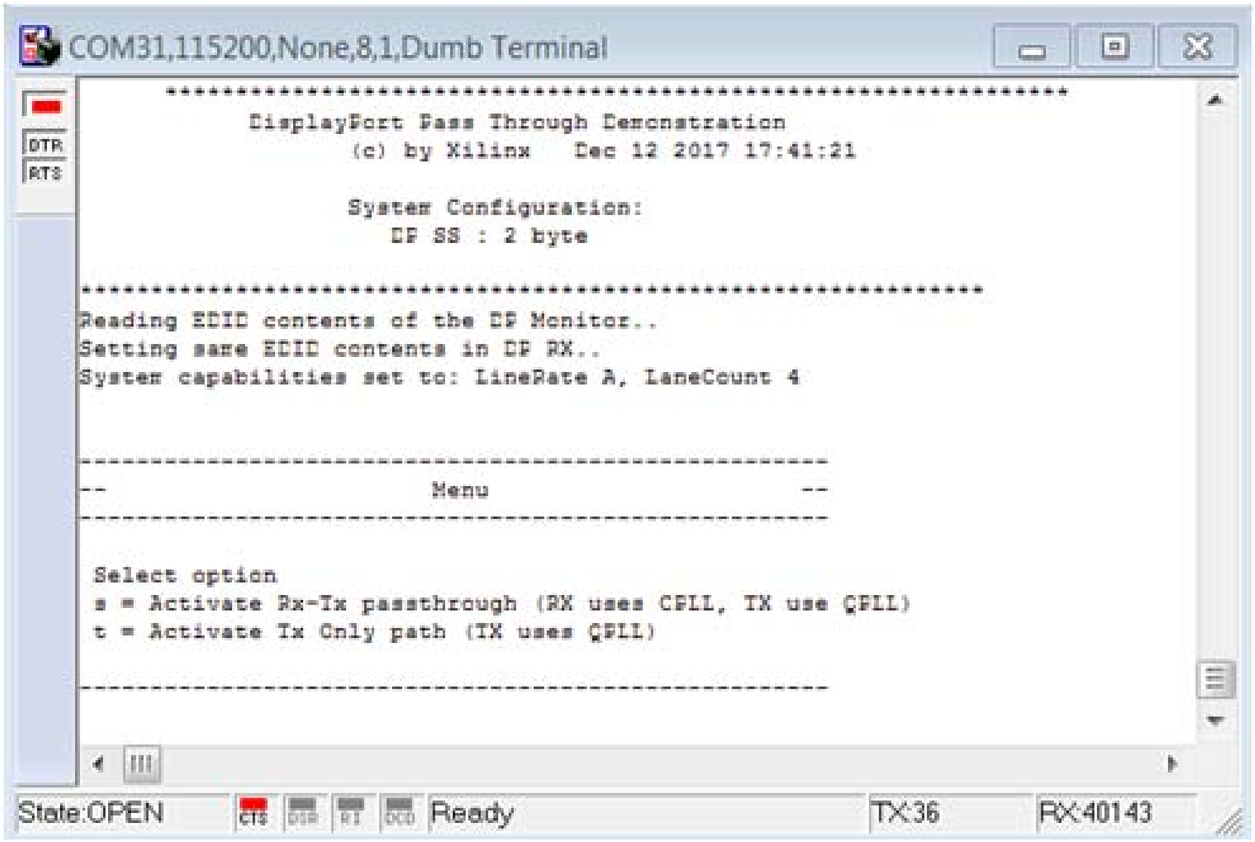
Advanced Options:

Figure 5-25: **Application**

Display User Console

Pass-Through Application (KCU105)

As soon as the application is executed, it checks if a Monitor is connected or not. If a monitor is already connected, then it starts up the following options as shown in Figure 5-26 to choose from (KCU105).



```
          ****************************************************************
                DisplayPort Pass Through Demonstration
                       (c) by Xilinx    Dec 12 2017 17:41:21

                       System Configuration:
                          DP SS : 2 byte

****************************************************************
Reading EDID contents of the DP Monitor..
Setting same EDID contents in DP RX..
System capabilities set to: LineRate A, LaneCount 4


-----------------------------------------------------
--                      Menu                       --
-----------------------------------------------------

 Select option
 s = Activate Rx-Tx passthrough (RX uses CPLL, TX use QPLL)
 t = Activate Tx Only path (TX uses QPLL)

-----------------------------------------------------
```

Figure 5-26: **DisplayPort User Console**

Selecting either `r` or `s` puts the system in Pass-Through mode, where the Video received by RX is forwarded to TX. This configures the `vid_phy_controller` and sets up the DisplayPort for RX. If a DisplayPort Source (for example, GPU) is already connected to DP RX, then it starts the training. Else, the training happens when the cable is plugged in. As soon as the training is completed, the application starts the DP TX Subsystem. The video should be seen on the monitor once the TX is up. Figure 5-26 shows the UART transcript. The transcript might differ based on the training done by GPU.

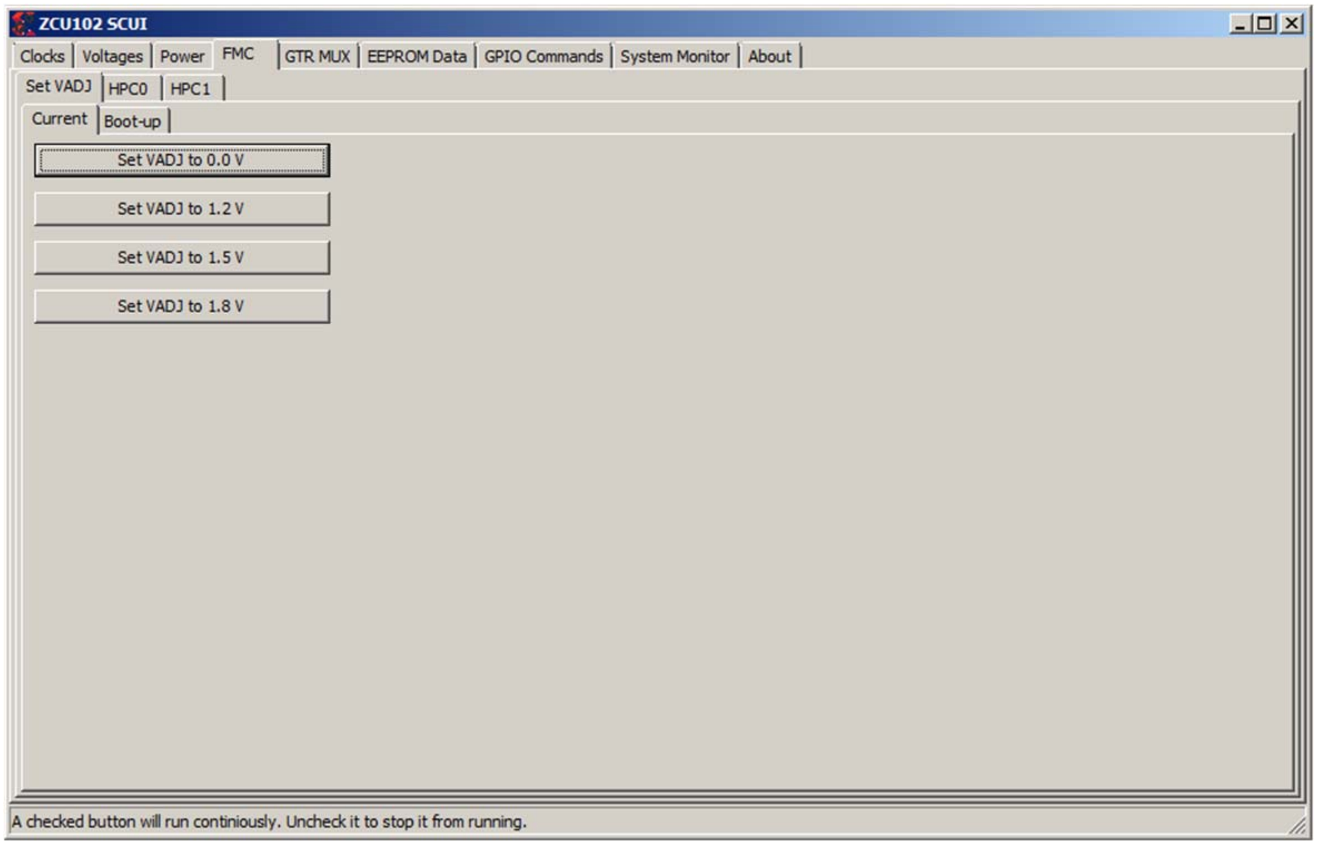
Setting the FMC Voltage to 1.8V

To run the example design on the ZCU102 board, ensure that only one ZCU102 board is connected to the host PC. This tool does not work with multiple ZCU102 connected to the host PC. There is no UART selection in this tool. Also, the FMC voltage is set to 1.8V. If you forget to set the FMC voltage, the following symptoms might occur:

- Random AUX failures
- Training failures

To set the FMC VADJ voltage:

1. Connect the ZCU102 board from the host PC to the USB UART port and power up the board.
2. Open the ZCU102 SCUI tool and select the **FMC** tab. On the **Set VADJ** tab, select the **Set VADJ to 1.8V**.



A checked button will run continiously. Uncheck it to stop it from running.

Figure 5-27: **ZCU102 SCUI**

Note: The SCUI tool can only be used with one board per one PC. If there are more than two ZCU102 boards connected to a PC, then it does not work.

Tested Equipment

Table 5-2 lists the tested equipment used with the example design.

Table 5-2: **Sink Equipment**

Sink Type	Brand Name	Model Name
Monitor	Acer	S277HKWMIDPP
Monitor	Dell	S2817
Monitor	Dell	UP3218K
Monitor	LG	27UD68P
Tester	Unigraf	UCD-323
Tester	Unigraf	UCD-400
Tester	Unigraf	DPR-100

Upgrading

There is no direct upgrade path due to the new retimer. Xilinx® recommends starting with a new design.

Frequently Asked Questions

Q. Can both RX and TX be used on the same GT quad for DisplayPort?

A. Yes. The Video PHY Controller supports the capability of performing both RX and TX on the GT quads. However, they cannot be different protocols.

Q. Does the Video PHY Controller support different protocols for RX and TX?

A. No. The Video PHY Controller must use the same protocol if both RX and TX is being used.

Q. I am having link training issues. What are some things that can be done to improve link training?

A. Perform the following:

1. Verify that all relevant ARs are taken into account.
2. Increase the AUX_DEFER value in register offset `0x004`.

Q. Does the Xilinx subsystem support my resolution and frame rate?

A. DisplayPort should operate at any resolution and frame rate as long as the DisplayPort link is not oversubscribed. Use the following equation to determine if the custom resolution can be supported:

$$(H_{Total} \times V_{Total} \times \text{bits\_per\_component} \times \text{frame rate}) < (0.8 \times \text{link\_lane} \times \text{num\_lanes})$$

Driver Documentation

The driver documentation can be found at the Xilinx GitHub page.

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 DisplayPort 1.4 TX 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 DisplayPort 1.4 TX 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 core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

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

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

Master Answer Record for the DisplayPort 1.4 TX Subsystem

AR: 70295

Technical Support

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

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

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

Debug Tools

There are many tools available to address DisplayPort 1.4 TX 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 9].

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. Xilinx recommends having an external auxiliary channel analyzer to understand the transactions between the Source and Sink cores.

General Checks

- Check the DisplayPort Source is DisplayPort 1.4 compliant.
- Make sure you are using proper DisplayPort 1.4 certified cable which is tested to run at 8.1 Gb/s.
- Ensure that the Signal Integrity of the lines is as per the DisplayPort standards for the AUX, TX, and Clock Input lines.

Transmit – Training Issue

This section contains debugging steps for issues with the clock recovery or channel equalization at sink and if the Training Done is Low.

- Try with a working sink such as the DisplayPort Analyzer sink device.
- Use a DisplayPort 1.4 certified cable. Change the cable and check again.
- Put a DisplayPort AUX Analyzer in the Transmit path and check if the various training stages match with the one's mentioned in Main Link Setup and Management in Chapter 3.
- Probe the `lnk_clk` output and check if the SI of the clock is within the Phase Noise mask of the respective GT.
- Check status registers in the Video PHY Controller for Reset done (`0x0020`) and PLL lock Status (`0x0018`)

Transmit – Main Link Problem After Training

This section contains debugging steps if the monitor is not displaying video even after a successful training, or if the monitor display is noisy and has many errors.

- Perform a software reset on the register `0x01C` and check if the video is proper now.
- Check if the MAIN_STREAM_ENABLE register is set to 1.
- Ensure that the MSA parameters match the Video being sent by the TX.

- Check the video pixel clock generation. Ensure that the Video Clock is based on the resolution being sent.
- Dump the DisplayPort source registers and compare against a working log.
- Check the symbol and disparity errors in the Sink through DPCD registers. This could be due to cable issue or PHY (GT) alignment issue.

Transmit – Audio

This section contains debugging steps for issues with audio communication.

- Check if MAUD and NAUD registers are correctly programmed and `aud_clk` is calculated as expected to be 512 × fs.
- Follow steps mentioned in Programming DisplayPort Source in Chapter 3.
- Check if the TX_AUDIO_CHANNELS register value matches with the input audio samples sent
- Check if the TX_AUDIO_INFO_DATA is correctly formatted as per CEA 861-C info frame specification.
- Ensure all the inputs data bits of `s_axis_audio_ingress_tdata` and `s_axis_audio_ingress_tid` are correctly sent as per the format specified.

Transmit – Misaligned Data

This section contains debugging steps for issues with data appearing to be misaligned or shifted on the monitor.

- Check the EDID timings to verify they are within the CVT standard RB and RB2 reduced blanking resolutions.
- Using EDID timings outside of the CVT standard can cause timing issues.

To fix this, define VTC_ADJUST_FOR_BS_TIMING in the `xdptxss_vtc.c`. This moves the BS symbol into the front porch to fix a swing in the BS timing caused by a non-standard CVT timing.

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

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

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

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

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

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

References

These documents provide supplemental material useful with this product guide:

1. *Video PHY Controller Product Guide* (PG230)
2. *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949)
3. *VESA DisplayPort Standard v1.4*, February 23, 2016
4. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
5. *Vivado Design Suite User Guide: Designing with IP* (UG896)
6. *Vivado Design Suite User Guide: Getting Started* (UG910)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *ISE to Vivado Design Suite Migration Guide* (UG911)
9. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
10. *Vivado Design Suite User Guide: Implementation* (UG904)
11. *AXI Reference Guide* (UG1037)
12. *AXI4-Stream to Video Out LogiCORE IP Product Guide* (PG044)
13. *Video Timing Controller LogiCORE IP Product Guide* (PG016)
14. *HDCP Controller Product Guide* (PG224)
15. *AXI Timer Product Guide* (PG079)
16. *ZCU102 System Controller – GUI Tutorial* (XTP433)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/04/2018	1.0	Initial Xilinx release.

Please Read: Important Legal Notices