# DisplayPort TX Subsystem v2.1

## *Product Guide*

**Vivado Design Suite**

**PG199 June 17, 2019**

# Table of Contents

## Introduction

The DisplayPort TX Subsystem implements the functionality of a video source as defined by the Video Electronics Standards Association (VESA®) DisplayPort standard v1.2a and supports driving resolutions of up to Ultra HD (UHD) at 60 fps. The Xilinx® DisplayPort subsystems provide highly integrated but straightforward IP blocks requiring very little customization.

## Features

- Support for DisplayPort Source (TX) transmissions
- Supports multi-stream transport (MST) and single stream transport (SST) at UHD at 60 fps
- Dynamic lane supports (1, 2, or 4 lanes)
- Dynamic link rate support (1.62/2.7/5.4 Gb/s)
- Dynamic support of 8, 10, 12, or 16 bits per component (BPC)
- Dynamic support of RGB/YCbCr444/ YCbCr422/Y-Only color formats
- Wide screen support with internal split of up to two streams of the same resolution in AXI4-Stream video interface mode
- Support 32 or 16-bit video PHY (GT) interface
- Supports 2 to 8 channel audio
- Supports HDCP 1.3 encryption
- Supports native or AXI4-Stream video input interface

| LogiCORE IP Facts Table | |
|---|---|
| **Core Specifics** | |
| Supported Device Family[(1)][(2)] | UltraScale+™ Families (GTHE4)<br>UltraScale™ Families (GTHE3)<br>Zynq®-7000 SoC (GTXE2)<br>Virtex®-7 (GTXE2) and Kintex®-7 (GTXE2) |
| 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[(3)]** | |
| Design Entry | Vivado® Design Suite |
| Simulation | For supported simulators, see the Xilinx Design Tools: Release Notes Guide. |
| Synthesis | Vivado Synthesis |
| **Support** | |
| Provided by Xilinx at the Xilinx Support web page | |

**Notes:**

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For HDCP: UltraScale/UltraScale+ supports up to 5.4 Gb/s, Kintex-7/Virtex-7 (-1 speed grade supports up to 2.7 Gb/s, -2/ -3 supports up to 5.4 Gb/s), and Artix-7 is not supported.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

# Overview

The DisplayPort TX Subsystem is a full feature, hierarchically packaged subsystem with a DisplayPort Transmit (TX) core ready to use in applications in large video systems.

## Feature Summary

- UHD up to 60 fps supports up to four streams for multi-stream transport (MST) and single stream transport (SST) modes
- Dynamic lane supports (1, 2, or 4 lanes)
- Dynamic support of different bits per color (6, 8, 10, 12 or 16) and line rates
- Dynamic support of RGB/YCbCr444/ YCbCr422/Y-Only color formats
- Support optional HDCP 1.3 Controller
- Support for native and AXI4-Stream video input interface

## Unsupported Features

- Audio in MST mode
- In-band stereo
- HDCP in MST mode.
- HDCP 2.x
- Video AXI4-Stream interface is not scalable with dynamic pixel mode selection
- Dual-pixel splitter in native video mode
- eDP and iDP
- GTC
- Non-LPCM audio
- No support for interlaced video in AXI4-Stream Interface

## 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 might be available for this core or subsystem. Contact your local Xilinx sales representative for information about pricing and availability.

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

### 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

The DisplayPort TX Subsystem, in both AXI4-Stream and native interface, operates in the following video modes:

- Single stream transport (SST)
- Multi-stream transport (MST)

The DisplayPort TX subsystem outputs video using the DisplayPort v1.2a protocol and works in conjunction with the Video PHY Controller, configured for the DisplayPort protocol. For more information on the Video PHY Controller, see the *Video PHY Controller Product Guide* (PG230) [Ref 2].

## 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)
- DP AXI4-Stream to Video Bridge

In MST mode, the subsystem also includes a dual splitter. In SST mode, the subsystem also includes an optional HDCP controller for encryption and an AXI Timer core as a helper core for HDCP functionality.

Because the DisplayPort 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 MST with four streams. The subsystem includes a multi-pixel AXI4-Stream Video Protocol interface.

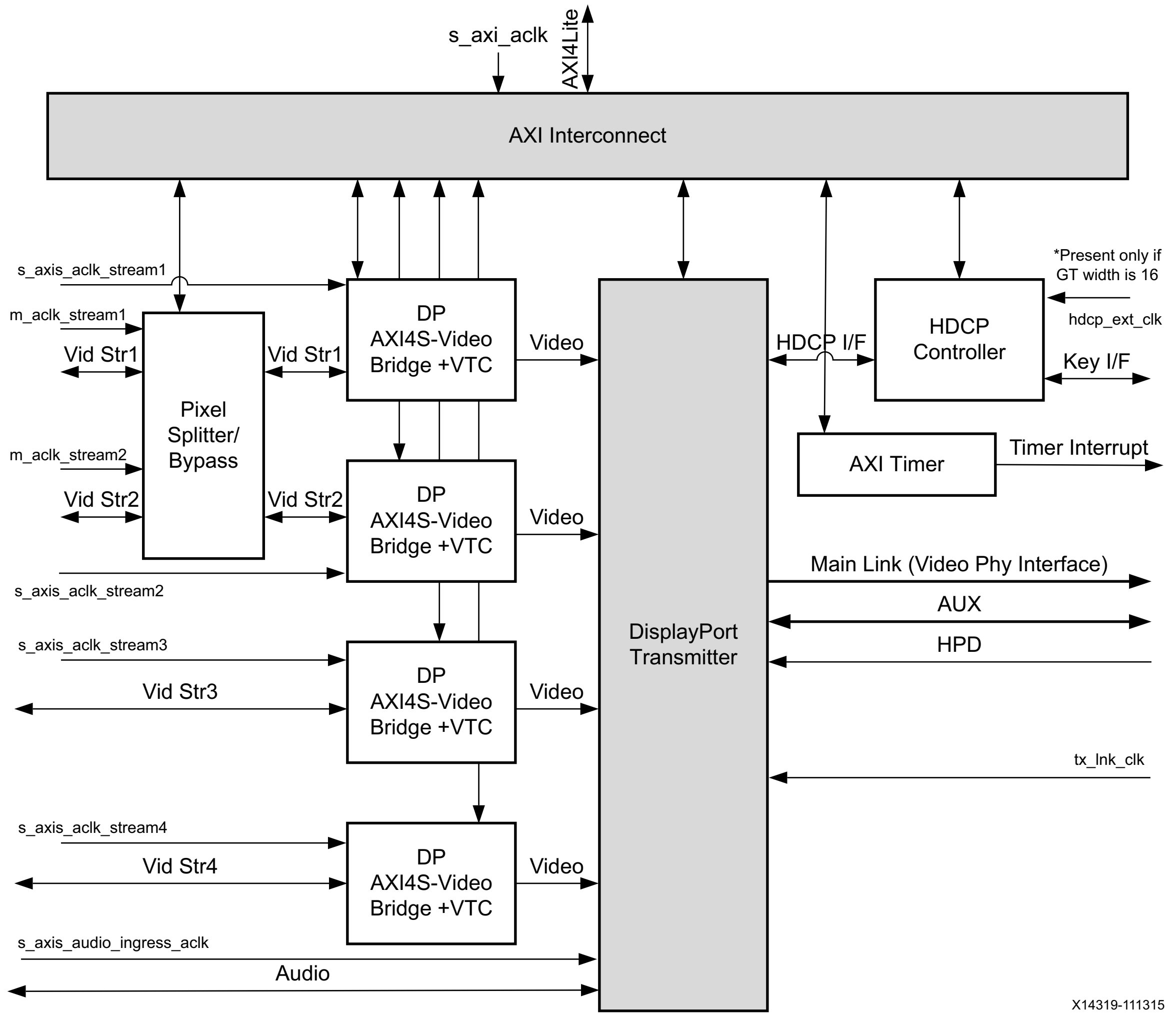


*Figure 2-1:* **DisplayPort TX Subsystem AXI4-Stream Video Interface Block Diagram**

***Note:*** MST HDCP is not supported.

## *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, Pixel Frequency < 75 MHz.
- For 2 PPC, Pixel Frequency ≥ 75 and < 300 MHz.
- For 4 PPC, Pixel Frequency ≥ 300 MHz.

Also, you can override the 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 four lanes with `MAX_BPC` of 16, the data width of the AXI4-Stream interface is 16 × 4 × 3 = 192.

Table 2-1 shows the pixel mapping examples for an AXI4-Stream interface implemented in the DisplayPort TX Subsystem.

*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 |
| | 2 | 48 | 96 | | – | – | – | – | – | – | 95:80 | 79:64 | 63:48 | 47:32 | 31:16 | 15:0 |
| | 1 | 48 | 48 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 36 | 72 | | – | – | – | – | – | – | 71:60 | 59:48 | 47:36 | 35:24 | 23:12 | 11:0 |
| | 1 | 36 | 36 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 30 | 60 | | – | – | – | – | – | – | 59:50 | 49:40 | 39:30 | 29:20 | 19:10 | 9:0 |
| | 1 | 30 | 30 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 24 | 48 | | – | – | – | – | – | – | 47:40 | 39:32 | 31:24 | 23:16 | 15:8 | 7:0 |
| | 1 | 24 | 24 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 48 | 96 | | – | – | – | – | – | – | 95:84 | 79:68 | 63:52 | 47:36 | 31:20 | 15:4 |
| | 1 | 48 | 48 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 36 | 72 | | – | – | – | – | – | – | 71:62 | 59:50 | 47:38 | 35:26 | 23:14 | 11:2 |
| | 1 | 36 | 36 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 30 | 60 | | – | – | – | – | – | – | 59:52 | 49:42 | 39:32 | 29:22 | 19:12 | 9:2 |
| | 1 | 30 | 30 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 24 | 48 | | – | – | – | – | – | – | 47:42 | 39:34 | 31:26 | 23:18 | 15:10 | 7:2 |
| | 1 | 24 | 24 | | – | – | – | – | – | – | – | – | – | 23:18 | 15:10 | 7:2 |
| 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 |
| | 2 | 48 | 96 | | – | – | – | – | – | – | 95:86 | 79:70 | 63:54 | 47:38 | 31:22 | 15:6 |
| | 1 | 48 | 48 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 36 | 72 | | – | – | – | – | – | – | 71:64 | 59:52 | 47:40 | 35:28 | 23:16 | 11:4 |
| | 1 | 36 | 36 | | – | – | – | – | – | – | – | – | – | 35:28 | 23:16 | 11:4 |

*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 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| | 2 | 30 | 60 | | – | – | – | – | – | – | 59:54 | 49:44 | 39:34 | 29:24 | 19:14 | 9:4 |
| | 1 | 30 | 30 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 48 | 96 | | – | – | – | – | – | – | 95:88 | 79:72 | 63:56 | 47:40 | 31:24 | 15:8 |
| | 1 | 48 | 48 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 36 | 72 | | – | – | – | – | – | – | 71:66 | 59:54 | 47:42 | 35:30 | 23:18 | 11:6 |
| | 1 | 36 | 36 | | – | – | – | – | – | – | – | – | – | 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 |
| | 2 | 48 | 96 | | – | – | – | – | – | – | 95:90 | 79:74 | 63:58 | 47:42 | 31:26 | 15:10 |
| | 1 | 48 | 48 | | – | – | – | – | – | – | – | – | – | 47:42 | 31:26 | 15:10 |

**Notes:**

1. The padding bits are zeros.

Table 2-1 shows the pixel mapping on an AXI4-Stream interface for all video sampling modes (4:4:4, 4:2:2, and Y-only). Implementation of 4:2:2 and Y-only sampling mode pixel mapping is different from the UG934 guidelines (see the *AXI4-Stream Video IP and System Design Guide* (UG934) [Ref 15]).

Table 2-2 shows the color component mapping for different video sampling modes for the AXI4-Stream pixel mapping shown in Table 2-1. The 422 and Y-only sampling modes are shown to highlight the difference between video sampling mode mappings. In Table 2-2, MAX_BPC is 16, LANES=4 and Video BPC=16.

*Table 2-2:* **Color Component Mapping for Different Video Sampling Modes**

| Pixel Width | Interface Width | Video Sampling Mode | Pixel 3 | | | Pixel 2 | | | Pixel 1 | | | Pixel 0 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 48 | 192 | 444 | 191:176 | 175:160 | 159:144 | 143:128 | 127:112 | 111:96 | 95:80 | 79:64 | 63:48 | 47:32 | 31:16 | 15:0 |
| 32 | 128 | 422 | V2 [191:176] | Y3 [175:160] | – | U2 [143:128] | Y2 [127:112] | – | V0 [95:80] | Y1 [79:64] | – | U0 [47:32] | Y0 [31:16] | – |
| 16 | 64 | Y-Only | Y2 [191:176] | – | – | Y2 [143:128] | – | – | Y1 [95:80] | – | – | Y0 [47:32] | – | – |

Table 2-3 shows UG934-compliant pixel mapping examples over an AXI4-Stream interface.

*Table 2-3:* **Pixel Mapping Examples on AXI4-Stream Interface (UG934-Compliant Mode)**

| MAX_BPC | LANES | Pixel Width | Interface Width | Video BPC | Video Sampling Mode | Pixel 3 | | | Pixel 2 | | | Pixel 1 | | | Pixel 0 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4 | 24 | 96 | | 444 | 95:88 | 87:80 | 79:72 | 71:64 | 63:56 | 55:48 | 47:40 | 39:32 | 31:24 | 23:16 | 15:8 | 7:0 |
| | 4 | 16 | 64 | | 422 | 63:56 | | 55:48 | 47:40 | | 39:32 | 31:24 | | 23:16 | 15:8 | | 7:0 |
| | 4 | 8 | 32 | | Y-Only | | | 31:24 | | | 23:16 | | | 15:8 | | | 7:0 |
| | 2 | 24 | 48 | | 444 | – | – | – | – | – | – | 47:40 | 39:32 | 31:24 | 23:16 | 15:8 | 7:0 |
| 8 | 2 | 16 | 32 | 8 | 422 | – | | – | – | | – | 31:24 | | 23:16 | 15:8 | | 7:0 |
| | 2 | 8 | 16 | | Y-Only | | | – | | | – | | | 15:8 | | | 7:0 |
| | 1 | 24 | 24 | | 444 | – | – | – | – | – | – | – | – | – | 23:16 | 15:8 | 7:0 |
| | 1 | 16 | 16 | | 422 | – | | – | – | | – | – | | – | 15:8 | | 7:0 |
| | 1 | 8 | 8 | | Y-Only | | | – | | | – | | | – | | | 7:0 |
| | 4 | 30 | 120 | | 444 | 119:112 | 109:102 | 99:92 | 89:82 | 79:72 | 69:62 | 59:52 | 49:42 | 39:32 | 29:22 | 19:12 | 9:2 |
| | 4 | 20 | 80 | | 422 | 79:72 | | 69:62 | 59:52 | | 49:42 | 39:32 | | 29:22 | 19:12 | | 9:2 |
| | 4 | 10 | 40 | | Y-Only | | | 39:32 | | | 29:22 | | | 19:12 | | | 9:2 |
| | 2 | 30 | 60 | | 444 | – | – | – | – | – | – | 59:52 | 49:42 | 39:32 | 29:22 | 19:12 | 9:2 |
| 10 | 2 | 20 | 40 | 8 | 422 | – | | – | – | | – | 39:32 | | 29:22 | 19:12 | | 9:2 |
| | 2 | 10 | 20 | | Y-Only | | | – | | | – | | | 19:12 | | | 9:2 |
| | 1 | 30 | 30 | | 444 | – | – | – | – | – | – | – | – | – | 29:22 | 19:12 | 9:2 |
| | 1 | 20 | 20 | | 422 | – | | – | – | | – | – | | – | 19:12 | | 9.2 |
| | 1 | 10 | 10 | | Y-Only | | | – | | | – | | | – | | | 9:2 |
| | 4 | 36 | 144 | | 444 | 143:136 | 131:124 | 119:112 | 107:100 | 95:88 | 83:76 | 71:64 | 59:52 | 47:40 | 35:28 | 23:16 | 11:4 |
| | 4 | 24 | 96 | | 422 | 95:88 | | 83:76 | 71:64 | | 59:52 | 47:40 | | 35:28 | 23:16 | | 11:4 |
| | 4 | 12 | 48 | | Y-Only | | | 47:40 | | | 35:28 | | | 23:16 | | | 11:4 |
| | 2 | 36 | 72 | | 444 | – | – | – | – | – | – | 71:64 | 59:52 | 47:40 | 35:28 | 23:16 | 11:4 |
| 12 | 2 | 24 | 48 | 8 | 422 | – | | – | – | | – | 47:40 | | 35:28 | 23:16 | | 11:4 |
| | 2 | 12 | 24 | | Y-Only | | | – | | | – | | | 23:16 | | | 11:4 |
| | 1 | 36 | 36 | | 444 | – | – | – | – | – | – | – | – | – | 35:28 | 23:16 | 11:4 |
| | 1 | 24 | 24 | | 422 | – | | – | – | | – | – | | – | 23:16 | | 11:4 |
| | 1 | 12 | 12 | | Y-Only | | | – | | | – | | | – | | | 11:4 |
| | 4 | 48 | 192 | | 444 | 191:184 | 175:168 | 159:152 | 143:136 | 127:120 | 111:104 | 95:88 | 79:72 | 63:56 | 47:40 | 31:24 | 15:8 |
| | 4 | 32 | 128 | | 422 | 127:120 | | 111:104 | 95:88 | | 79:72 | 63:56 | | 47:40 | 31:24 | | 15:8 |
| | 4 | 16 | 64 | | Y-Only | | | 63:56 | | | 47:40 | | | 31:24 | | | 15:8 |
| | 2 | 48 | 96 | | 444 | – | – | – | – | – | – | 95:88 | 79:72 | 63:56 | 47:40 | 31:24 | 15:8 |
| 16 | 2 | 32 | 64 | 8 | 422 | – | | – | – | | – | 63:56 | | 47:40 | 31:24 | | 15:8 |
| | 2 | 16 | 32 | | Y-Only | | | – | | | – | | | 31:24 | | | 15:8 |
| | 1 | 48 | 48 | | 444 | – | – | – | – | – | – | – | – | – | 47:40 | 31:24 | 15:8 |
| | 1 | 32 | 32 | | 422 | – | | – | – | | – | – | | – | 31:24 | | 15:8 |
| | 1 | 16 | 16 | | Y-Only | | | – | | | – | | | – | | | 15:8 |

## Native Video Interface

With the native interface enabled, the subsystem is, by default, packaged with only one subcore, the DisplayPort Transmit core. In SST mode, the TX subsystem also includes the option to enable the HDCP controller for encryption and the AXI Timer core as a helper core for HDCP functionality. Figure 2-2 shows the architecture of the subsystem assuming MST with four native video streams. The subsystem includes a multi-pixel native video protocol interface.

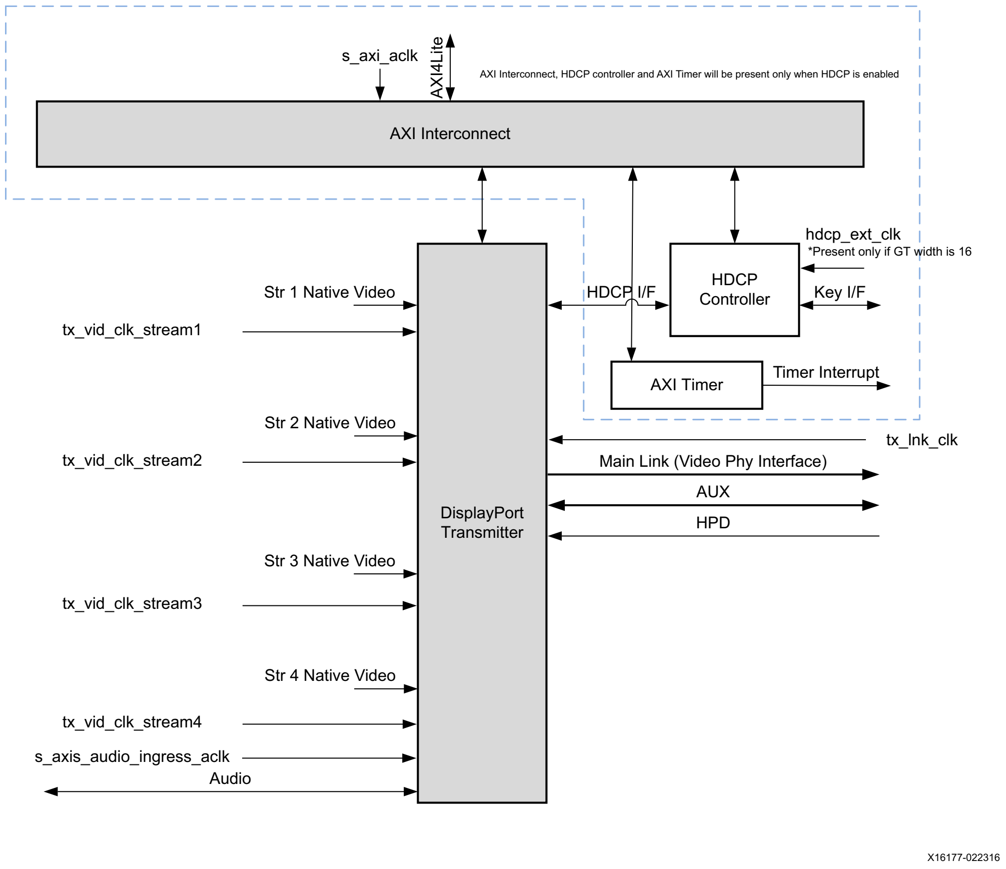


*Figure 2-2:* **DisplayPort TX Subsystem Native Video Block Diagram**

***Note:*** MST HDCP is not supported.

### *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.

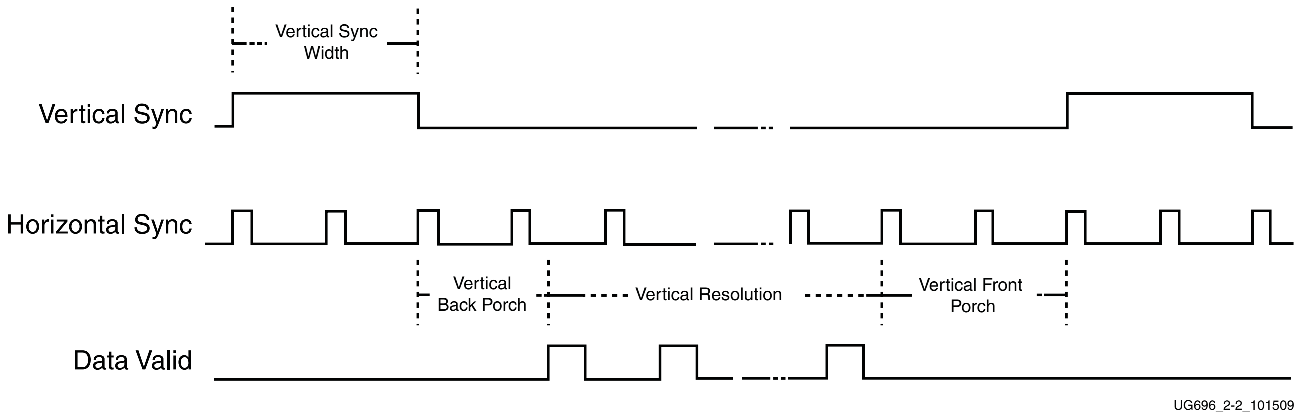


*Figure 2-3:* **User Interface Vertical Timing**

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.

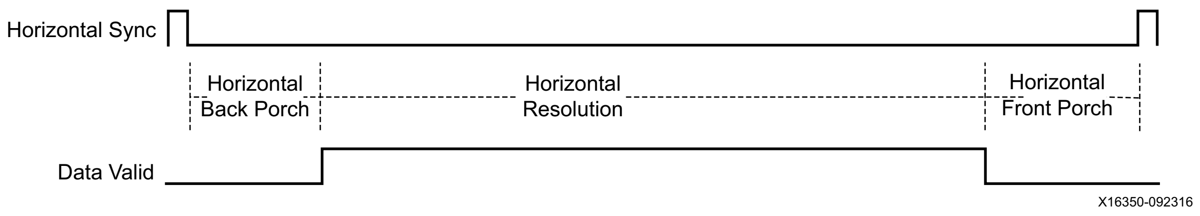


*Figure 2-4:* **User Interface Horizontal Timing**

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.

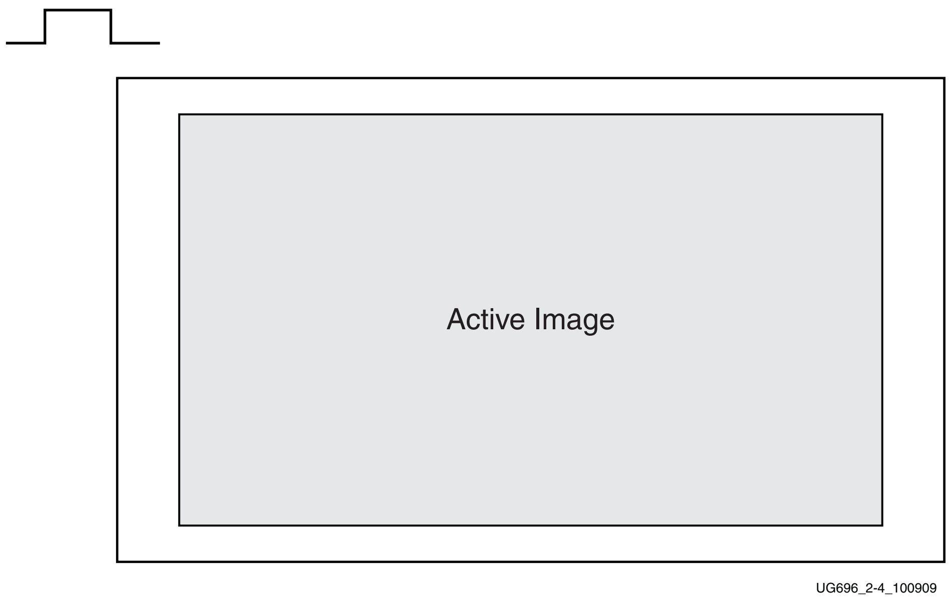


*Figure 2-5:* **Active Image Data**

The User Data Interface can accept one, two, or four pixels per clock cycle. The `vid_pixel` width is always 48 bits, regardless of whether 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 2-4 provides the mapping for all supported data formats.

*Table 2-4:* **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] | – | – | – | – | – |

*Table 2-4:* **Pixel Mapping for the User Data Interface *(Cont'd)***

| Format | BPC/BPP | R | G | B | Cr | Y | Cb | Cr/Cb | Y |
|---|---|---|---|---|---|---|---|---|---|
| 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] |
| 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 clock for the 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 the 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.

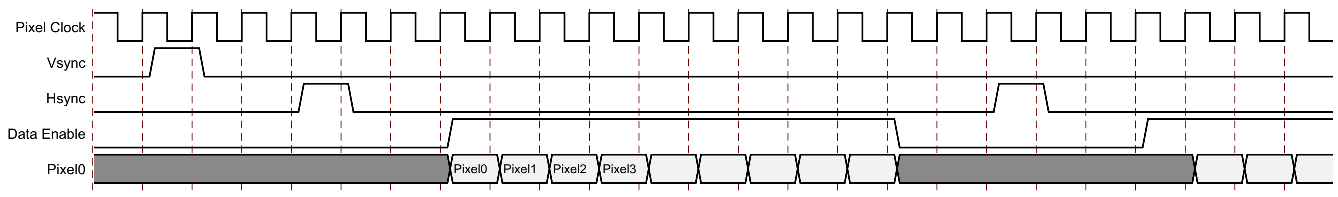


*Figure 2-6:* **Single Pixel Timing**

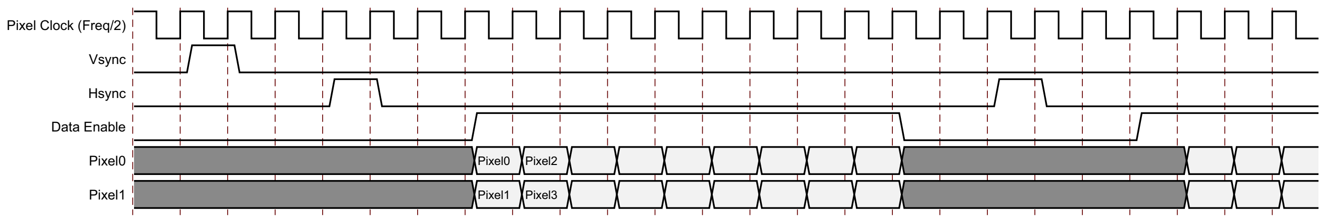


*Figure 2-7:* **Dual Pixel Timing**

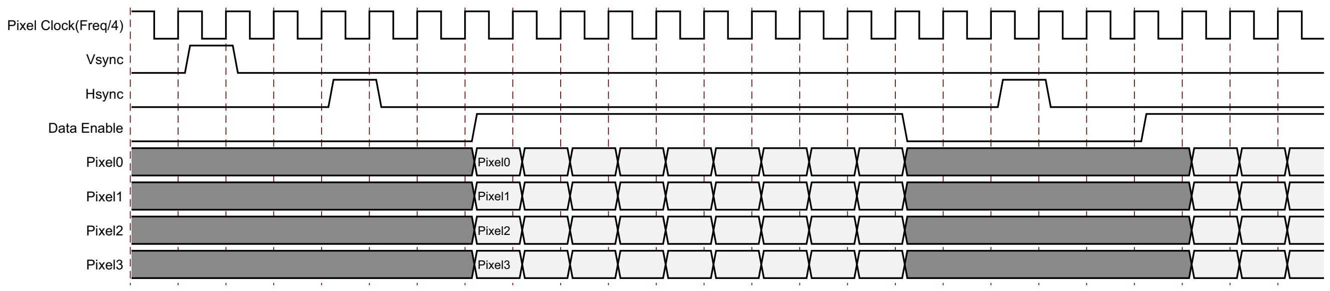


*Figure 2-8:* **Quad Pixel Timing**

## DisplayPort Dual Splitter

The Dual Splitter is used to vertically split the frame to support MST with two streams, as shown in Figure 2-9. Despite the frames being split, you will see this as one frame. The dual splitter has a buffer to hold the data for up to one and a half scan lines.

***Note:*** The dual splitter is present only when MST is enabled in AXI4-Stream interface mode. While using the dual splitter, ensure that the unused input video streams are grounded.

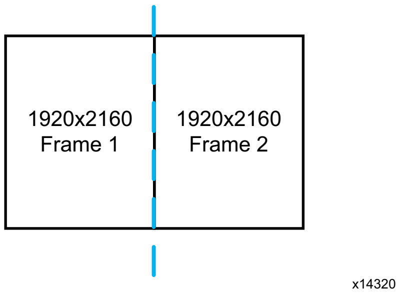


*Figure 2-9:* **Vertically Split Frame**

### *Splitter Interface*

The splitter input and output are video over AXI4-Stream interface. Figure 2-10 shows the timing of this interface.

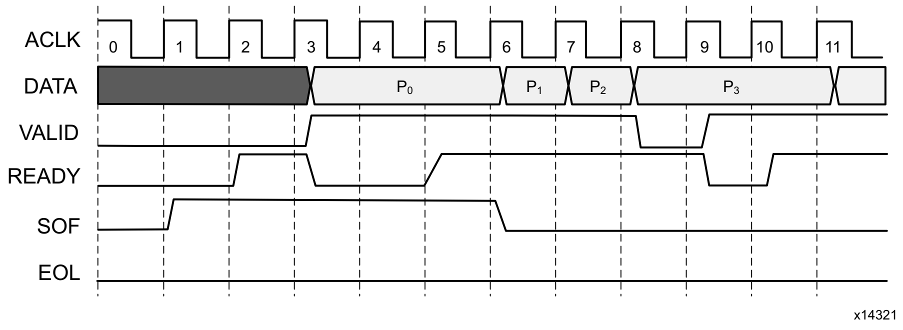


*Figure 2-10:* **Video over AXI4-Stream Interface Timing**

Based on the mode, the Core Control register (CORE_CONTROL_REG) of the dual splitter must be configured for input and output samples per clock. See Dual Splitter Registers for a description of CORE_CONTROL_REG.

## 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 and DisplayPort native video.

Table 2-5 shows the color mapping for the AXI4-Stream interface.

*Table 2-5:* **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 11]. For details about video over AXI4-Stream, see the *AXI Reference Guide* (UG1037) [Ref 10]. The input of the bridge is Video over AXI4-Stream. For more details, see Port Descriptions.

***Note:*** In MST mode, there are N number of bridges in the subsystem, where N = the number of AXI4-Stream inputs to the subsystem.

## Video Timing Controller

The Xilinx Video Timing Controller is used for generation of video timing. Video The 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 12].

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

## DisplayPort Transmit

The DisplayPort Transmit core contains the following components as shown in Figure 2-11:

- **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.

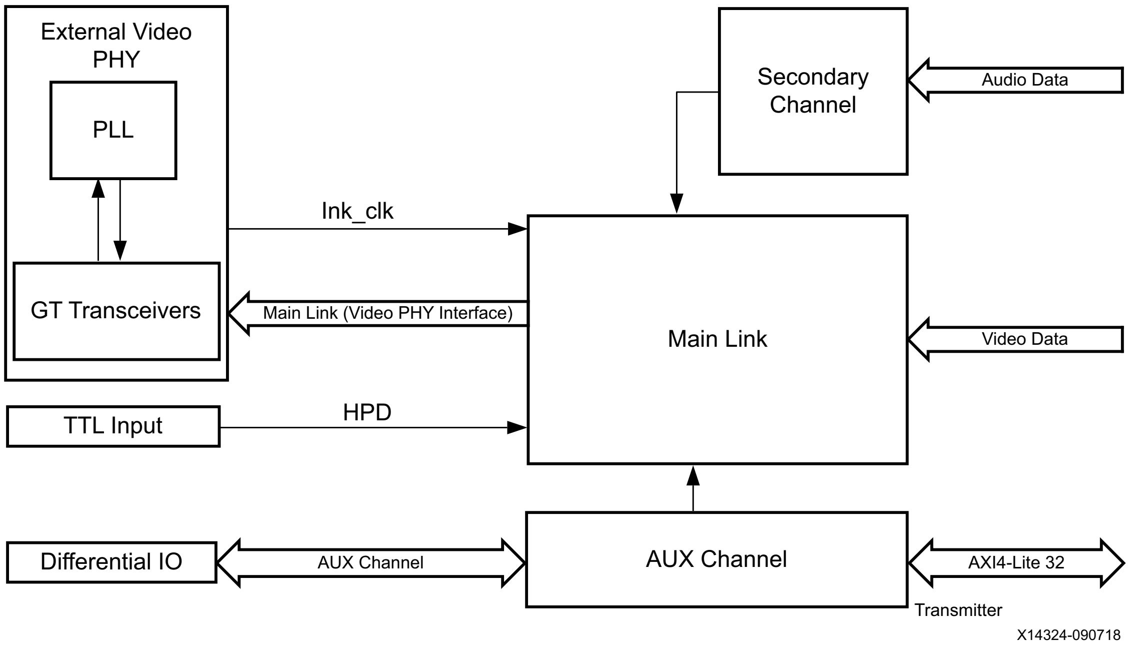


*Figure 2-11:* **DisplayPort Transmit Core Block Diagram**

## AXI Interconnect

The subsystem uses Xilinx AXI Interconnect IP core, as a crossbar which contains an AXI4-Lite interface. Figure 2-12 shows the AXI slave structure within the DisplayPort TX Subsystem.

***Note:*** For MST with *N* streams, there are *N* Video Timing Controllers. See Address Map Example in Chapter 3.

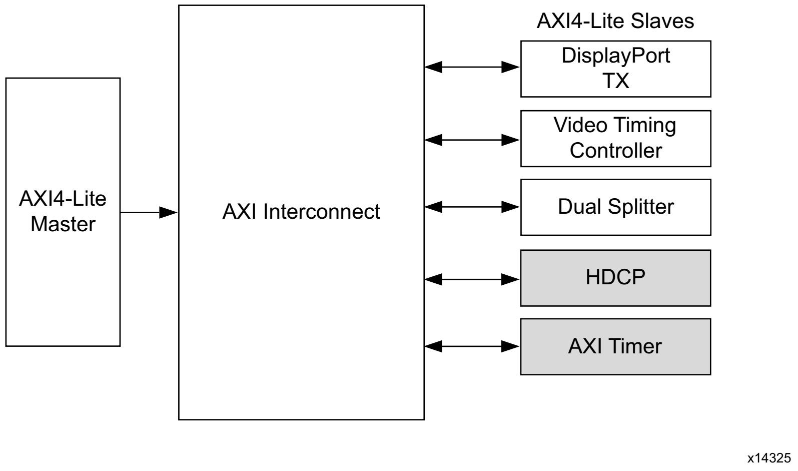


*Figure 2-12:* **AXI4-Lite Interconnect within DisplayPort TX Subsystem**

## HDCP Controller

The HDCP v1.3 protocol specifies a secure method of transmitting audiovisual content. Further, the audiovisual content can be transmitted over a DisplayPort interface. The HDCP Controller is used for data encryption with the DisplayPort Transmitter IP in the DisplayPort TX Subsystem.

Figure 2-13 shows the DisplayPort TX Subsystem with HDCP controller. For more details on HDCP, see the *HDCP Controller Product Guide* (PG224) [Ref 13].

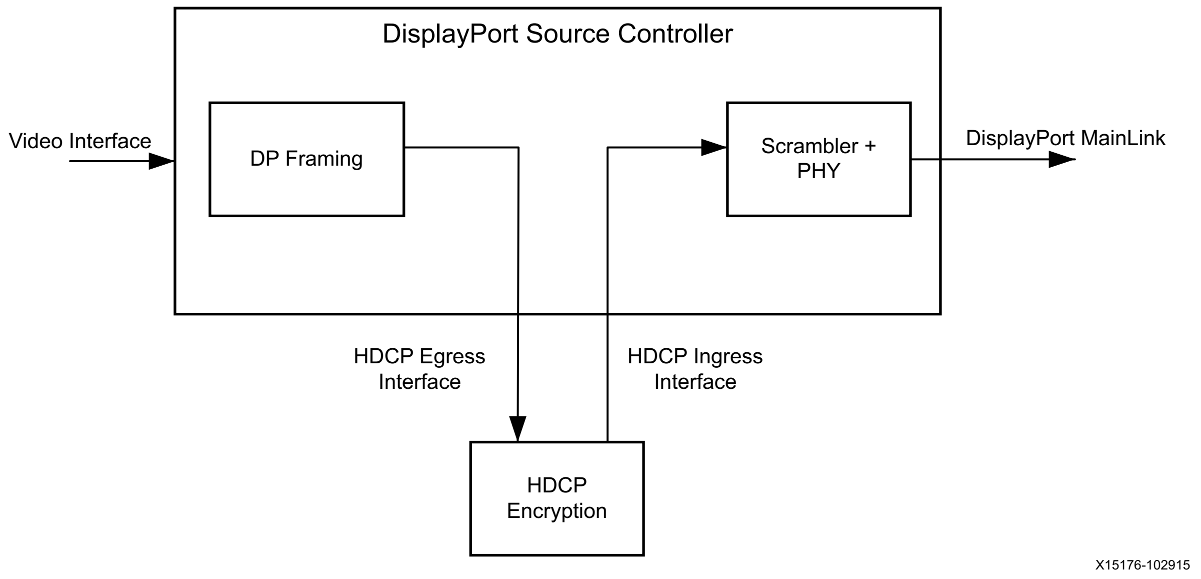


*Figure 2-13:* **DisplayPort TX with HDCP Controller**

***Note:*** MST HDCP is not supported.

## AXI Timer

A 32-bit AXI Timer is used in the DisplayPort TX subsystem when the HDCP controller is enabled for encryption. The AXI Timer can be accessed through an AXI4 master interface for basic timer functionality in the system.

## Standards

The DisplayPort TX Subsystem is compatible with the DisplayPort v1.2a Standard, HDCP v1.3 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 TX Subsystem ports are described in the following tables.

### AXI4-Lite Interface Ports

*Table 2-6:* **AXI4-Lite Interface Ports**

| Signal Name | I/O | Description |
|---|---|---|
| s_axi_aclk | I | AXI Bus Clock. |
| s_axi_aresetn | I | AXI Reset. Active-Low. |
| s_axi_awaddr[18:0] | I | Write Address |
| s_axi_awprot | I | Protection type. |
| s_axi_awvalid | I | Write address valid. |
| s_axi_awready | O | Write address ready. |
| s_axi_wdata[31:0] | I | Write data bus. |
| s_axi_wstrb[3:0] | I | Write strobes. |
| s_axi_wvalid | I | Write valid. |
| s_axi_wready | O | Write ready. |
| s_axi_bresp[1:0] | O | Write response. |
| s_axi_bvalid | O | Write response valid. |
| s_axi_bready | I | Response ready. |
| s_axi_araddr[18:0] | I | Read address. |
| s_axi_arprot[1:0] | I | Protection type. |
| s_axi_arvalid | I | Read address valid. |
| s_axi_arready | O | Read address ready. |
| s_axi_rdata[31:0] | O | Read data. |
| s_axi_rresp[1:0] | O | Read response. |
| s_axi_rvalid | O | Read valid. |
| s_axi_rready | I | Read ready. |

## SST AXI4-Stream Interface

This interface is enabled when the AXI4-Stream interface is selected (<n>= 1).

*Table 2-7:* **SST AXI4-Stream Interface Ports**

| Signal Name | I/O | Description |
|---|---|---|
| s_axis_aclk_stream1 | I | AXI4-Stream clock. |
| s_axis_aresetn_stream1 | I | AXI4-Stream reset. Active-Low. |
| s_axis_video_stream1_tdata[191:0] | I | Video data input. Maximum width is 192.<br>For more information on the interface width, see Table 2-1. |
| s_axis_video_stream1_tlast | I | Video end of line. |
| s_axis_video_stream1_tready | O | AXI4-Stream tready output. |
| s_axis_video_stream1_tuser | I | Video start of frame. |
| s_axis_video_stream1_tvalid | I | Video valid. |

## SST Native Video Interface

This interface is enabled when native video is selected (<n> = 1).

*Table 2-8:* **SST Native Video Interface Ports**

| Signal Name | I/O | Description |
|---|---|---|
| tx_vid_clk_stream1 | I | User video clock. |
| tx_vid_rst_stream1 | I | User video reset. Active-High. |
| tx_video_stream1_tx_vid_vsync | I | Vertical sync pulse. Active on the rising edge. |
| tx_video_stream1_tx_vid_hsync | I | Horizontal sync pulse. Active on the rising edge |
| tx_video_stream1_tx_vid_enable | I | User data video enable. |
| tx_video_stream1_tx_vid_pixel0[47:0] | I | Video data |
| tx_video_stream1_tx_vid_pixel1[47:0] | I | Video data |
| tx_video_stream1_tx_vid_pixel2[47:0] | I | Video data |
| tx_video_stream1_tx_vid_pixel3[47:0] | I | Video data |
| tx_video_stream1_tx_vid_oddeven | I | Odd/even field select. Indicates an odd (1) or even (0) field polarity. Default is 0. If not used, this pin should be connected to ground. |

## MST AXI4-Stream Interface

See Clocking in Chapter 3 for the clock values.

*Table 2-9:* **MST AXI4-Stream Interface Ports**

| Signal Name | I/O | Description |
|---|---|---|
| s_axis_aclk_stream<n> | I | MST stream clock. |
| s_axis_aresetn_stream<n> | I | MST stream reset. Active-Low. |
| s_axis_video_stream<n>_tdata[191:0] | I | MST stream video data input. Maximum width is 192. |
| s_axis_video_stream<n>_tlast | I | MST stream video end of line. |
| s_axis_video_stream<n>_tready | O | MST stream input ready. |
| s_axis_video_stream<n>_tuser | I | MST stream video start of frame. |
| s_axis_video_stream<n>_tvalid | I | MST stream video valid. |
| m_aclk_stream1 | I | Video pipe clock for stream1. Used in MST configuration. |
| m_aresetn_stream1 | I | Active-Low video pipe reset for stream 1. Used in MST configuration. |
| m_aclk_stream2 | I | Video pipe clock for stream 2. Used in MST configuration. |
| m_aresetn_stream2 | I | Active-Low video pipe reset for stream 2. Used in MST configuration. |
| tx_vid_clk_stream<n> | I | User data clock for MST stream *n*. |
| tx_vid_rst_stream<n> | I | Active-High user video reset. |
| tx_video_stream<n>_tx_vid_vsync | I | Vertical sync pulse. Active on the rising edge. |
| tx_video_stream<n>_tx_vid_hsync | I | Horizontal sync pulse. Active on the rising edge |
| tx_video_stream<n>_tx_vid_enable | I | User data video enable. |
| tx_video_stream<n>_tx_vid_pixel0[47:0] | I | Video data |
| tx_video_stream<n>_tx_vid_pixel1[47:0] | I | Video data |
| tx_video_stream<n>_tx_vid_pixel2[47:0] | I | Video data |
| tx_video_stream<n>_tx_vid_pixel3[47:0] | I | Video data |
| tx_video_stream<n>_tx_vid_oddeven | I | Odd/even field select. Indicates an odd (1) or even (0) field polarity. Default is 0. If not used, this pin should be connected to ground. |

**Notes:**

1. <n>= 2 to 4.

## User Port

*Table 2-10:* **User Port**

| Signal Name | I/O | Description |
|---|---|---|
| tx_hpd | I | Hot-plug detect signal to TX from RX. |

## Audio AXI4-Stream Interface

*Table 2-11:* **Audio AXI4-Stream Interface Ports**

| Signal Name | I/O | Description |
|---|---|---|
| s_axis_audio_ingress_aclk | I | AXI4-Stream clock. |
| s_axis_audio_ingress_aresetn | I | Active-Low reset. |
| s_axis_audio_ingress_tdata[31:0] | I | AXI4-Stream data input.<br>• [3:0] – Preamble Code<br>◦ 4'b0001: Subframe1/ Start of audio block<br>4'b0010: Subframe 1<br>4'b0011: Subframe 2<br>• [27:4] – Audio Sample Word<br>• [28] – Validity Bit (V)<br>• [29] – User Bit (U)<br>• [30] – Channel Status (C)<br>• [31] – Parity (P) |
| s_axis_audio_ingress_tid[7:0] | I | • [3:0] – Audio Channel ID<br>• [7:4] – Audio Packet Stream ID |
| s_axis_audio_ingress_tvalid | I | Valid indicator for audio data from master. |
| s_axis_audio_ingress_tready | O | Ready indicator from DisplayPort source. |

## External Video PHY Sideband Status Interface

*Table 2-12:* **External Video PHY Sideband Status Interface Ports**

| Signal Name | I/O | Description |
|---|---|---|
| s_axis_phy_tx_sb_status_tdata[7:0] | O | Sideband status to Video PHY |
| s_axis_phy_tx_sb_status_tready | I | Sideband status ready input from Video PHY |
| s_axis_phy_tx_sb_status_tvalid | O | Sideband status data valid to Video PHY |

## External Video PHY Clock Interface

*Table 2-13:* **External Video PHY Clock Interface Ports**

| Signal Name | I/O | Description |
|---|---|---|
| tx_lnk_clk | I | Link clock input from external Video PHY |

## External Video PHY Lane n Interface

*Table 2-14:* **External Video PHY Lane n Interface Ports**

| Signal Name | I/O | Description |
|---|---|---|
| m_axis_lnk_tx_lane<n>_tdata[31:0] | O | Lane<n> Data to External Video PHY |
| m_axis_lnk_tx_lane<n>_tvalid | O | Lane<n> Data Valid to External Video PHY |
| m_axis_lnk_tx_lane<n>_tready | I | Lane<n> Data Ready from External Video PHY |
| m_axis_lnk_tx_lane<n>_tuser[11:0] | O | Lane<n> User data out to External Video PHY |

**Notes:**
1. <n>=0 to Lane_Count-1.

## HDCP Key Interface

*Table 2-15:* **HDCP Key Interface Ports**

| Signal Name | I/O | Description |
|---|---|---|
| hdcp_ext_clk | I | HDCP external clock (enabled when HDCP is selected with 16-bit GT interface) |
| hdcp_key_aclk | I | Key clock |
| hdcp_key_aresetn | I | Key Interface reset. Active-Low |
| hdcp_key_tdata[63:0] | I | AXI4-Stream Key Tdata |
| hdcp_key_last | I | AXI4-Stream Key Tlast |
| hdcp_key_tready | O | AXI4-Stream Key Tready |
| hdcp_key_tuser[7:0] | I | AXI4-Stream Key TUSER. KMB should send the Key number from 0 to 41. 0 corresponds to KSV and 1 to 40 are the HDCP Keys count. |
| hdcp_key_tvalid | I | AXI4-Stream Key Tvalid |
| reg_key_sel[2:0] | O | To select the one of the eight sets of 40 keys. |
| start_key_transmit | O | An Active-High pulse that is used to start key transmit. |

## AUX Signals

*Table 2-16:* **AUX Signal Ports**

| Signal Name | I/O | Description |
|---|---|---|
| aux_tx_io_n | O | Negative polarity AUX Manchester-II data. |
| aux_tx_io_p | O | Positive polarity AUX Manchester-II data. |
| aux_tx_channel_in_p | I | Positive polarity AUX channel input.<br>Valid when AUX IO Type is unidirectional. |
| aux_tx_channel_in_n | I | Negative polarity AUX channel input.<br>Valid when AUX IO Type is unidirectional. |
| aux_tx_channel_out_p | O | Positive polarity AUX channel Output.<br>Valid when AUX IO Type is unidirectional |

Table 2-16: **AUX Signal Ports *(Cont'd)***

| Signal Name | I/O | Description |
|---|---|---|
| aux_tx_channel_out_n | O | Negative Polarity AUX channel output.<br>Valid when AUX IO Type is unidirectional |
| aux_tx_data_out | O | AUX data out. Valid when AUX IO buffer location is external |
| aux_tx_data_in | I | AUX data input. Valid when AUX IO buffer location is external |
| aux_tx_data_en_out_n | O | AUX data output enable. Active-Low.<br>Valid only when AUX IO buffer location is external |

### Interrupt Interface

Table 2-17: **Interrupt Interface Ports**

| Signal Name | I/O | Description |
|---|---|---|
| dptxss_dp_irq | O | DisplayPort TX IP interrupt out |
| dptxss_hdcp_irq | O | HDCP IP interrupt out |
| dptxss_timer_irq | O | AXI Timer IP interrupt output valid only when HDCP is enabled |

## Register Space

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

- Dual Splitter
- VTC 0 (Up to 3 for 4 streams)
- DisplayPort Transmit
- HDCP Controller
- AXI Timer

**TIP:** *For details about accessing these registers, see Programming Sequence in Chapter 3.*

### Dual Splitter Registers

Table 2-18 defines the Dual Splitter registers.

*Table 2-18:* **Dual Splitter Register Definitions**

| Offset | Register | Access | Default Value | Definition |
|---|---|---|---|---|
| 0x0000 | GENR_CONTROL_REG | R/W | 0x2 | • [0] – Enables the splitter.<br>• [1] – Register update.<br>• [31] – Soft reset bit.<br>Other registers can be programed by writing a value 2 to this register. At the end of programing set the register to 3. |
| 0x0008 | GENR_ERROR_REG | R/W | 0x0 | • [0] – Slave EOL early.<br>• [1] – Slave EOL late.<br>• [2] – Slave SOF early.<br>• [3] – Slave SOF late. |
| 0x000C | IRQ_ENABLE | R/W | 0 | [0] – Interrupt based on the error conditions. |
| 0x0020 | TIME_CONTROL REG | R/W | 0x0870_0F00 | Contains the input image size:<br>• [15:0] – Height[1]<br>• [31:16] – Width<br>***Note:*** For UHD @60 frame split mode, HRES must be programmed to actual HRES/4. |
| 0x0100 | CORE_CONTROL_REG | R/W | 0x00_01_01_01 | For UHD @ 60 frame split mode, this register can be programmed to 0x020404. For all other modes, it can be 0x10404.<br>• [7:0] – Input number of samples per clock.<br>• [15:8] – Output number of samples per clock.<br>• [23:16] – Number of image segments.<br>• [31:24] – Number of samples overlapping the segments. Should be programmed to 0 because the subsystem supports two frames without overlap. |

**Notes:**

1. Height refers to VRES and the WIDTH refers to HRES.

## Video Timing Controller Registers

For details about the Video Timing Controller (VTC) registers, see the *Video Timing Controller Product Guide* (PG016) [Ref 12].

## DisplayPort Transmit IP Core 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 2-19 to Table 2-25 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 the following tables. Base addresses are configured by the AXI Interconnect.

## *Link Configuration Field*

*Table 2-19:* **Link Configuration Field**

| Offset | R/W | Definition |
|---|---|---|
| 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.<br>• [7:0] – LINK_BW_SET: Sets the value of the main link bandwidth for the sink device.<br>◦ 0x06 = 1.62 Gb/s<br>◦ 0x0A = 2.7 Gb/s<br>◦ 0x14 = 5.4 Gb/s |
| 0x004 | RW | LANE_COUNT_SET. Sets the number of lanes used by the source in transmitting data.<br>• [4:0] – Set to 1, 2, or 4 |
| 0x008 | RW | ENHANCED_FRAME_EN<br>• [0] –Set to 1 by the source to enable the enhanced framing symbol sequence. |
| 0x00C | RW | TRAINING_PATTERN_SET. Sets the link training mode.<br>• [1:0] – Set the link training pattern according to the two bit code.<br>◦ 00 = Training off<br>◦ 01 = Training pattern 1, used for clock recovery<br>◦ 10 = Training pattern 2, used for channel equalization<br>◦ 11 = Training pattern 3, used for channel equalization for cores with DisplayPort Standard v1.2a. |
| 0x010 | RW | LINK_QUAL_PATTERN_SET. Transmit the link quality pattern.<br>• [2:0] – Enable transmission of the link quality test patterns.<br>◦ 000 = Link quality test pattern not transmitted<br>◦ 001 = D10.2 test pattern (unscrambled) transmitted<br>◦ 010 = Symbol Error Rate measurement pattern<br>◦ 011 = PRBS7 transmitted<br>◦ 100 = Custom 80-bit pattern<br>◦ 101 = HBR2 compliance pattern |

*Table 2-19:* **Link Configuration Field *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x014 | RW | SCRAMBLING_DISABLE. Set to 1 when the transmitter has disabled the scrambler and transmits all symbols.<br>• [0] – Disable scrambling. |
| 0x01C | WO | SOFTWARE_RESET. Reads will return zeros.<br>• [0] – Soft Video Reset: When set, video logic is reset (stream 1).<br>• [1] – Soft Video Reset: When set, video logic is reset (stream 2).<br>• [2] – Soft Video Reset: When set, video logic is reset (stream 3).<br>• [3] – Soft Video Reset: When set, video logic is reset (stream 4).<br>• [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 | [15:0] – Custom 80-bit quality pattern Bits[79:64]<br>[31:16] – Reserved |

### *Core Enables*

*Table 2-20:* **Core Enables**

| Offset | R/W | Definition |
|---|---|---|
| 0x080 | RW | TRANSMITTER_ENABLE. Enable the basic operations of the transmitter.<br>• [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.<br>• [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.<br>***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.<br>• [0] – set to 1 to enable the video data packing to work in lnk_clk for SST video or for Stream 1 in MST.<br>• [1] – set to 1 to enable the video data packing to work in lnk_clk for Stream 2 in MST.<br>• [2] – set to 1 to enable the video data packing to work in lnk_clk for Stream 3 in MST.<br>• [3] – set to 1 to enable the video data packing to work in lnk_clk for Stream 4 in MST. |
| 0x0C0 | WO | FORCE_SCRAMBLER_RESET. Reads from this register always return 0x0.<br>• [0] – 1 forces a scrambler reset. |

*Table 2-20:* **Core Enables *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x0D0 | RW | TX_MST_CONFIG: MST Configuration.<br>• [0] – MST Enable: Set to 1 to enable MST functionality.<br>• [1] – VC Payload Updated in sink: This is an WO bit. Set to 1 after reading DPCD register 0x2C0 (bit 0) is set. |
| 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.<br>• [0] – End of line reset disable to the SST video stream/ MST video stream1<br>• [1] – End of line reset disable to the MST video stream2<br>• [2] – End of line reset disable to the MST video stream3<br>• [3] – End of line reset disable to the MST video stream4 |

## Core ID

*Table 2-21:* **Core ID**

| Offset | R/W | Definition |
|---|---|---|
| 0x0F8 | RO | VERSION_REGISTER. For example, for displayport_v7.0, the VERSION REGISTER is 32'h07_00_0_0_00.<br>• [31:24] – Core major version.<br>• [23:16] – Core minor version.<br>• [15:12] – Core version revision.<br>• [11:8] – Core Patch details.<br>• [7:0] – Internal revision. |
| 0x0FC | RO | CORE_ID. Returns the unique identification code of the core and the current revision level.<br>• [31:24] – DisplayPort protocol major version<br>• [23:16] – DisplayPort protocol minor version<br>• [15:8] – DisplayPort protocol revision<br>• [7:0]<br>  ◦ 0x00: Transmit<br>  ◦ 0x01: Receive<br>The CORE_ID values for the various protocols and cores are:<br>• DisplayPort Standard v1.1a protocol with a Transmit core: 32'h01_01_0a_00<br>• DisplayPort Standard v1.2a protocol with a Transmit core: 32'h01_02_0a_00 |

### *AUX Channel Interface*

*Table 2-22:* **AUX Channel Interface**

| Offset | R/W | Definition |
|---|---|---|
| 0x100 | RW | AUX_COMMAND_REGISTER. Initiates AUX channel commands of the specified length.<br>• [12] – Address only transfer enable. When this bit is set to 1, the source initiates Address only transfers (STOP is sent after the command).<br>• [11:8] – AUX Channel Command.<br>  ◦ 0x8 = AUX Write<br>  ◦ 0x9 = AUX Read<br>  ◦ 0x0 = I2C Write<br>  ◦ 0x4 = I2C Write MOT<br>  ◦ 0x1 = I2C Read<br>  ◦ 0x5 = I2C Read MOT<br>  ◦ 0x2 = I2C Write Status<br>• [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.<br>• [7:0] – AUX Channel byte data. |
| 0x108 | RW | AUX_ADDRESS. Specifies the address for the current AUX channel command.<br>• [19:0] – Twenty bit address for the start of the AUX Channel burst. |
| 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).<br>• [7:0] – Clock divider value.<br>• [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.<br>From DisplayPort Protocol spec, AUX Pulse Width range = 0.4 to 0.6 μs.<br>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.<br>((20 × 24 = 480))<br>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.<br>• [0] – FIFO_OVERFLOW_FLAG: 1 indicates that the internal FIFO has detected an overflow condition. This bit clears upon read. |

*Table 2-22:* **AUX Channel Interface *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x130 | RO | INTERRUPT_SIGNAL_STATE. Contains the raw signal values for those conditions which might cause an interrupt.<br>• [3] – REPLY_TIMEOUT: 1 indicates that a reply timeout has occurred.<br>• [2] – REPLY_STATE: 1 indicates that a reply is currently being received.<br>• [1] – REQUEST_STATE: 1 indicates that a request is currently being sent.<br>• [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.<br>• [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.<br>***Note:*** The core does not retry any commands that were Deferred or Not Acknowledged.<br>• [1:0]<br>  ◦ 00 = AUX ACK<br>  ◦ 01 = AUX NACK<br>  ◦ 10 = AUX DEFER<br>• [3:2]<br>  ◦ 00 = I2C ACK<br>  ◦ 01 = I2C NACK<br>  ◦ 10 = I2C 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.<br>• [7:0] – Current reply count. |
| 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.<br>• [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.<br>• [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.<br>• [4] – HPD_PULSE_DETECTED: A pulse on the HPD line was detected. The duration of the pulse can be determined by reading 0x150.<br>• [3] – REPLY_TIMEOUT: A reply timeout has occurred.<br>• [2] – REPLY_RECEIVED: An AUX reply transaction has been detected.<br>• [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 ms.<br>• [0] – HPD_IRQ: An IRQ framed with the proper timing on the HPD signal has been detected. |

*Table 2-22:* **AUX Channel Interface *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x144 | RW | INTERRUPT_MASK. Masks the specified interrupt sources from asserting the axi_init signal. When set to 1, the specified interrupt source is masked.<br>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.<br>• [9] – Mask Audio packet ID mismatch interrupt.<br>• [5] – EXT_PKT_TXD: Mask Extended Packet Transmitted interrupt.<br>• [4] – HPD_PULSE_DETECTED: Mask HPD Pulse interrupt.<br>• [3] – REPLY_TIMEOUT: Mask reply timeout interrupt.<br>• [2] – REPLY_RECEIVED: Mask reply received interrupt.<br>• [1] – HPD_EVENT: Mask HPD event interrupt.<br>• [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.<br>• [4:0] – Total number of data bytes received during the reply phase of the AUX transaction. |
| 0x14C | RO | REPLY_STATUS<br>• [15:12] – RESERVED<br>• [11:4] – REPLY_STATUS_STATE: Internal AUX reply state machine status bits.<br>• [3] – REPLY_ERROR: When set to 1, the AUX reply logic has detected an error in the reply to the most recent AUX transaction.<br>• [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.<br>• [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.<br>• [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. |
| 0x150 | RO | HPD_DURATION<br>• [15:0] – Duration of the HPD pulse in microseconds. |
| 0x154 | RO | Free running counter incrementing for every 1 MHz. |

### *Main Stream Attributes*

*Table 2-23:* **Main Stream Attributes**

| Offset | R/W | Definition[1] |
|---|---|---|
| 0x180 | RW | MAIN_STREAM_HTOTAL. Specifies the total number of clocks in the horizontal framing period for the main stream video signal.<br>• [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.<br>• [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 v1.2a Specification* [Ref 1].<br>• 0 = Active-High<br>• 1 = Active-Low<br>• [1] – VSYNC_POLARITY: Polarity of the vertical sync pulse.<br>• [0] – HSYNC_POLARITY: Polarity of the horizontal sync pulse. |
| 0x18C | RW | MAIN_STREAM_HSWIDTH. Sets the width of the horizontal sync pulse.<br>• [14:0] – Horizontal sync width in clock cycles. |
| 0x190 | RW | MAIN_STREAM_VSWIDTH. Sets the width of the vertical sync pulse.<br>• [14:0] – Width of the vertical sync in lines. |
| 0x194 | RW | MAIN_STREAM_HRES. Horizontal resolution of the main stream video source.<br>• [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.<br>• [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.<br>• [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.<br>• [15:0] – Vertical start line count. |

*Table 2-23:* **Main Stream Attributes *(Cont'd)***

| Offset | R/W | Definition[(1)] |
|---|---|---|
| 0x1A4 | RW | MAIN_STREAM_MISC0. Miscellaneous stream attributes.<br>• [7:0] – Implements the attribute information contained in the DisplayPort MISC0 register described in section 2.2.4 of the standard.<br>• [0] – Synchronous Clock.<br>• [2:1] – Component Format.<br>• [3] – Dynamic Range.<br>• [4] – YCbCr Colorimetry.<br>• [7:5] – Bit depth per color/component.<br>• [8] – Override Audio Clocking Mode<br>• [9] – Sync/Async Mode for audio<br>• [10] – Audio Only Mode. When enabled, controller inserts information/timestamp packets every 512 BS symbols. By default the value is 0.<br>• [11] – Maud control (Advanced Users) |
| 0x1A8 | RW | MAIN_STREAM_MISC1. Miscellaneous stream attributes.<br>• [7:0] – Implements the attribute information contained in the DisplayPort MISC1 register described in section 2.2.4 of the standard.<br>• [0] – Interlaced vertical total even.<br>• [2:1] – Stereo video attribute.<br>• [6:3] – Reserved. |
| 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.<br>• [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.<br>• [6:0] – This number should be in the range of 32 to 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.<br>• [23:0] – Unsigned N value. |

*Table 2-23:* **Main Stream Attributes *(Cont'd)***

| Offset | R/W | Definition[1] |
|---|---|---|
| 0x1B8 | RW | USER_PIXEL_WIDTH. Selects the width of the user data input port. Use quad pixel mode in MST. In SST, the user pixel width should always be less than or equal to the active lane count generated in hardware.<br>• [2:0]:<br>◦ 1 – Single pixel wide interface<br>◦ 2 – Dual pixel wide interface. Valid for designs with 2 or 4 lanes.<br>◦ 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.<br>If (HRES x bits per pixel) is divisible by 16, then<br>words_per_line = ((HRES × bits per pixel)/16)<br>Else<br>words_per_line = (INT((HRES × bits per pixel)/16)) + 1<br>For single-lane design:<br>Set USER_DATA_COUNT_PER_LANE = words_per_line - 1<br>For 2-lane design:<br>If words_per_line is divisible by 2, then<br>Set USER_DATA_COUNT_PER_LANE = words_per_line - 2<br>Else<br>Set USER_DATA_COUNT_PER_LANE = words_per_line + MOD(words_per_line,2) - 2<br>For 4-lane design:<br>If words_per_line is divisible by 4, then<br>Set USER_DATA_COUNT_PER_LANE = words_per_line - 4<br>Else<br>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 1 for the proper transmission of interlaced sources.<br>• [0] – Set to 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. MIN_BYTES_PER_TU should be ≥ 4 when GT Data width is selected as 32-bit.<br>• [6:0] – Set the value to INT((VIDEO_BW/LINK_BW) x 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.<br>• [9:0] – The fraction part of ((VIDEO_BW/LINK_BW) x TRANSFER_UNIT_SIZE) scaled by 1024 is programmed in this register. |

*Table 2-23:* **Main Stream Attributes *(Cont'd)***

| Offset | R/W | Definition[1] |
|---|---|---|
| 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.<br>If (MIN_BYTES_PER_TU ≤ 4)<br>• [7:0] – Set INIT_WAIT to 64<br>else if color format is RGB/YCbCr_444<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)<br>else if color format is YCbCr_422<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)/2<br>else if color format is Y_Only<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)/3 |
| 0x1D0 | RW | STREAM1. Average Stream Symbol Timeslots per MTP Config:<br>• [9:0] – TS_FRAC: Program fraction × 1000 in this field. See the *DisplayPort Standard* section 2.6.3.3 VC Payload Size Determination by a Source Payload Bandwidth Manager.<br>• [23:16] – TS_INT: Program integer value based on the calculations. |
| 0x1D4 | RW | STREAM2. Average Stream Symbol Timeslots per MTP Config:<br>• [9:0] – TS_FRAC: Program fraction × 1000 in this field. See the *DisplayPort Standard* section 2.6.3.3 VC Payload Size Determination by a Source Payload Bandwidth Manager.<br>• [23:16] – TS_INT: Program integer value based on the calculations. |
| 0x1D8 | RW | STREAM3. Average Stream Symbol Timeslots per MTP Config:<br>• [9:0] – TS_FRAC: Program fraction × 1000 in this field. See the *DisplayPort Standard* section 2.6.3.3 VC Payload Size Determination by a Source Payload Bandwidth Manager.<br>• [23:16] – TS_INT: Program integer value based on the calculations. |
| 0x1DC | RW | STREAM4. Average Stream Symbol Timeslots per MTP Config:<br>• [9:0] – TS_FRAC: Program fraction × 1000 in this field. See the *DisplayPort Standard* section 2.6.3.3 VC Payload Size Determination by a Source Payload Bandwidth Manager.<br>• [23:16] – TS_INT: Program integer value based on the calculations. |

**Notes:**
1. See the DisplayPort Standard for more details [Ref 1].

### PHY Configuration Status

*Table 2-24:* **PHY Configuration Status**

| Offset | R/W | Definition |
|---|---|---|
| 0x280 | RO | PHY_STATUS. Provides the current status from the PHY.<br>• [1:0] – Reset done for lanes 0 and 1.<br>• [3:2] – Reset done for lanes 2 and 3.<br>• [4] – PLL for lanes 0 and 1 locked.<br>• [5] – PLL for lanes 2 and 3 locked.<br>• [6] – FPGA fabric clock PLL locked.<br>• [15:7] – Unused, read as 0.<br>• [17:16] – Transmitter buffer status, lane 0.<br>• [19:18] – Unused, read as 0.<br>• [21:20] – Transmitter buffer status, lane 1.<br>• [23:22] – Unused, read as 0.<br>• [25:24] – Transmitter buffer status, lane 2.<br>• [27:26] – Unused, read as 0.<br>• [29:28] – Transmitter buffer status, lane 3.<br>• [31:30] – Unused, read as 0. |
| 0x4FC | RO | SINK_VID_FRAMING_ERROR_STATUS: Sink Video Framing error status. This is a debug register that is valid when GT data width is 32-bit.<br>• [1:0] – Stream1 error status in framing.<br>• [9:8] – Stream2 error status in framing.<br>• [17:16] – Stream3 error status in framing.<br>• [25:24] – Stream4 error status in framing. |

### MST Interface

*Table 2-25:* **MST Interface**

| Offset | R/W | Definition |
|---|---|---|
| 0x500 | RW | MAIN_STREAM_HTOTAL_STREAM2. Specifies the total number of clocks in the horizontal framing period for the main stream video signal.<br>• [15:0] – Horizontal line length total in clocks. |
| 0x504 | RW | MAIN_STREAM_VTOTAL_STREAM2. Provides the total number of lines in the main stream video frame.<br>• [15:0] – Total number of lines per video frame. |
| 0x508 | RW | MAIN_STREAM_POLARITY_STREAM2. Provides the polarity values for the video sync signals.<br>• [1] – VSYNC_POLARITY: Polarity of the vertical sync pulse.<br>• [0] – HSYNC_POLARITY: Polarity of the horizontal sync pulse. |
| 0x50C | RW | MAIN_STREAM_HSWIDTH_STREAM2. Sets the width of the horizontal sync pulse.<br>• [14:0] – Horizontal sync width in clock cycles. |

*Table 2-25:* **MST Interface *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x510 | RW | MAIN_STREAM_VSWIDTH_STREAM2. Sets the width of the vertical sync pulse.<br>• [14:0] – Width of the vertical sync in lines. |
| 0x514 | RW | MAIN_STREAM_HRES_STREAM2. Horizontal resolution of the main stream video source.<br>• [15:0] – Number of active pixels per line of the main stream video. |
| 0x518 | RW | MAIN_STREAM_VRES_STREAM2. Vertical resolution of the main stream video source.<br>• [15:0] – Number of active lines of video in the main stream video source. |
| 0x51C | RW | MAIN_STREAM_HSTART_STREAM2. Number of clocks between the leading edge of the horizontal sync and the start of active data.<br>• [15:0] – Horizontal start clock count. |
| 0x520 | RW | MAIN_STREAM_VSTART_STREAM2. Number of lines between the leading edge of the vertical sync and the first line of active data.<br>• [15:0] – Vertical start line count. |
| 0x524 | RW | MAIN_STREAM_MISC0_STREAM2. Miscellaneous stream attributes.<br>• [7:0] – Implements the attribute information contained in the DisplayPort MISC0 register described in section 2.2.4 of the standard.<br>• [0] – Synchronous Clock.<br>• [2:1] – Component Format.<br>• [3] – Dynamic Range.<br>• [4] – YCbCr Colorimetry.<br>• [7:5] – Bit depth per color/component. |
| 0x528 | RW | MAIN_STREAM_MISC1_STREAM2. Miscellaneous stream attributes.<br>• [7:0] – Implements the attribute information contained in the DisplayPort MISC1 register described in section 2.2.4 of the standard.<br>• [0] – Interlaced vertical total even.<br>• [2:1] – Stereo video attribute.<br>• [6:3] – Reserved. |
| 0x52C | RW | M-VID_STREAM2. 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 as automatically computed by the source core and written to the main stream. These values are not written into the M-VID register for readback.<br>• [23:0] – Unsigned M value. |
| 0x530 | RW | TRANSFER_UNIT_SIZE_STREAM2. Sets the size of a transfer unit in the framing logic On reset, transfer size is set to 64.<br>• [6:0] – This number should be in the range of 32 to 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). |

*Table 2-25:* **MST Interface *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x534 | RW | N-VID_STREAM2. 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 as automatically computed by the source core and written to the main stream. These values are not written into the N-VID register for readback.<br>• [23:0] – Unsigned N value. |
| 0x538 | RW | USER_PIXEL_WIDTH_STREAM2. Selects the width of the user data input port. Use quad pixel mode in MST.<br>• [2:0]:<br>  ◦ 1 = Single pixel wide interface<br>  ◦ 2 = Dual pixel wide interface<br>  ◦ 4 = Quad pixel wide interface |
| 0x53C | RW | USER_DATA_COUNT_PER_LANE_STREAM2. This register is used to translate the number of pixels per line to the native internal datapath.<br>If (HRES × bits per pixel) is divisible by 16, then<br>  words_per_line = ((HRES x bits per pixel)/16)<br>Else<br>  words_per_line = (INT((HRES × bits per pixel)/16)) + 1<br>For single-lane design:<br>  Set USER_DATA_COUNT_PER_LANE = words_per_line - 1<br>For 2-lane design:<br>If words_per_line is divisible by 2, then<br>  Set USER_DATA_COUNT_PER_LANE = words_per_line - 2<br>Else<br>  Set USER_DATA_COUNT_PER_LANE = words_per_line + MOD(words_per_line,2) - 2<br>For 4-lane design:<br>If words_per_line is divisible by 4, then<br>  Set USER_DATA_COUNT_PER_LANE = words_per_line - 4<br>Else<br>  Set USER_DATA_COUNT_PER_LANE = words_per_line + MOD(words_per_line,4) - 4 |
| 0x540 | RW | MAIN_STREAM_INTERLACED_STREAM2. Informs the DisplayPort transmitter main link that the source video is interlaced. By setting this bit to a '1', the core will set the appropriate fields in the VBID value and Main Stream Attributes. This bit must be set to 1 for the proper transmission of interlaced sources.<br>• [0] – Set to 1 when transmitting interlaced images. |
| 0x544 | RW | MIN_BYTES_PER_TU_STREAM2: Programs source to use MIN number of bytes per transfer unit. The calculation should be done based on the DisplayPort Standard.<br>• [7:0] – Set the value to INT((LINK_BW/VIDEO_BW) x TRANSFER_UNIT_SIZE) |
| 0x548 | RW | FRAC_BYTES_PER_TU_STREAM2: Calculating MIN bytes per TU will often not be a whole number. This register is used to hold the fractional component.<br>• [9:0] – The fraction part of ((LINK_BW/VIDEO_BW) x TRANSFER_UNIT_SIZE) scaled by 1000 is programmed in this register. |

*Table 2-25:* **MST Interface *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x54C | RW | INIT_WAIT_STREAM2: 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.<br>If (MIN_BYTES_PER_TU ≤ 4)<br>• [7:0] – Set INIT_WAIT to 64<br>else if color format is RGB/YCbCr_444<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)<br>else if color format is YCbCr_422<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)/2<br>else if color format is Y_Only<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)/3 |
| 0x550 | RW | MAIN_STREAM_HTOTAL_STREAM3. Specifies the total number of clocks in the horizontal framing period for the main stream video signal.<br>• [15:0] – Horizontal line length total in clocks. |
| 0x554 | RW | MAIN_STREAM_VTOTAL_STREAM3. Provides the total number of lines in the main stream video frame.<br>• [15:0] – Total number of lines per video frame. |
| 0x558 | RW | MAIN_STREAM_POLARITY_STREAM3. Provides the polarity values for the video sync signals.<br>• [1] – VSYNC_POLARITY: Polarity of the vertical sync pulse.<br>• [0] – HSYNC_POLARITY: Polarity of the horizontal sync pulse. |
| 0x55C | RW | MAIN_STREAM_HSWIDTH_STREAM3. Sets the width of the horizontal sync pulse.<br>• [14:0] – Horizontal sync width in clock cycles. |
| 0x560 | RW | MAIN_STREAM_VSWIDTH_STREAM3. Sets the width of the vertical sync pulse.<br>• [14:0] – Width of the vertical sync in lines. |
| 0x564 | RW | MAIN_STREAM_HRES_STREAM3. Horizontal resolution of the main stream video source.<br>• [15:0] – Number of active pixels per line of the main stream video. |
| 0x568 | RW | MAIN_STREAM_VRES_STREAM3. Vertical resolution of the main stream video source.<br>• [15:0] – Number of active lines of video in the main stream video source. |
| 0x56C | RW | MAIN_STREAM_HSTART_STREAM3. Number of clocks between the leading edge of the horizontal sync and the start of active data.<br>• [15:0] – Horizontal start clock count. |
| 0x570 | RW | MAIN_STREAM_VSTART_STREAM3. Number of lines between the leading edge of the vertical sync and the first line of active data.<br>• [15:0] – Vertical start line count. |

*Table 2-25:* **MST Interface *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x574 | RW | MAIN_STREAM_MISC0_STREAM3. Miscellaneous stream attributes.<br>• [7:0] – Implements the attribute information contained in the DisplayPort MISC0 register described in section 2.2.4 of the standard.<br>• [0] – Synchronous Clock.<br>• [2:1] – Component Format.<br>• [3] – Dynamic Range.<br>• [4] – YCbCr Colorimetry.<br>• [7:5] – Bit depth per color/component. |
| 0x578 | RW | MAIN_STREAM_MISC1_STREAM3. Miscellaneous stream attributes.<br>• [7:0] – Implements the attribute information contained in the DisplayPort MISC1 register described in section 2.2.4 of the standard.<br>• [0] – Interlaced vertical total even.<br>• [2:1] – Stereo video attribute.<br>• [6:3] – Reserved. |
| 0x57C | RW | M-VID_STREAM3. 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 as automatically computed by the source core and written to the main stream. These values are not written into the M-VID register for readback.<br>• [23:0] – Unsigned M value |
| 0x580 | RW | TRANSFER_UNIT_SIZE_STREAM3. Sets the size of a transfer unit in the framing logic On reset, transfer size is set to 64.<br>• [6:0] – This number should be in the range of 32 to 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). |
| 0x584 | RW | N-VID_STREAM3. 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 as automatically computed by the source core and written to the main stream. These values are not written into the N-VID register for readback.<br>• [23:0] – Unsigned N value |
| 0x588 | RW | USER_PIXEL_WIDTH_STREAM3. Selects the width of the user data input port. Use quad pixel mode in MST.<br>• [2:0]:<br>◦ 1 = Single pixel wide interface<br>◦ 2 = Dual pixel wide interface<br>◦ 4 = Quad pixel wide interface |

Table 2-25: **MST Interface *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x58C | RW | USER_DATA_COUNT_PER_LANE_STREAM3. This register is used to translate the number of pixels per line to the native internal 16-bit datapath.<br>If (HRES x bits per pixel) is divisible by 16, then<br>words_per_line = ((HRES × bits per pixel)/16)<br>Else<br>words_per_line = (INT((HRES × bits per pixel)/16)) + 1<br>For single-lane design:<br>Set USER_DATA_COUNT_PER_LANE = words_per_line - 1<br>For 2-lane design:<br>If words_per_line is divisible by 2, then<br>Set USER_DATA_COUNT_PER_LANE = words_per_line - 2<br>Else<br>Set USER_DATA_COUNT_PER_LANE = words_per_line + MOD(words_per_line,2) - 2<br>For 4-lane design:<br>If words_per_line is divisible by 4, then<br>Set USER_DATA_COUNT_PER_LANE = words_per_line - 4<br>Else<br>Set USER_DATA_COUNT_PER_LANE = words_per_line + MOD(words_per_line,4) - 4 |
| 0x590 | RW | MAIN_STREAM_INTERLACED_STREAM3. Informs the DisplayPort transmitter main link that the source video is interlaced. By setting this bit to a 1, the core will set the appropriate fields in the VBID value and Main Stream Attributes. This bit must be set to 1 for the proper transmission of interlaced sources.<br>• [0] – Set to 1 when transmitting interlaced images. |
| 0x594 | RW | MIN_BYTES_PER_TU_STREAM3: Programs source to use MIN number of bytes per transfer unit. The calculation should be done based on the DisplayPort Standard.<br>• [7:0] – Set the value to INT((LINK_BW/VIDEO_BW) x TRANSFER_UNIT_SIZE) |
| 0x598 | RW | FRAC_BYTES_PER_TU_STREAM3: Calculating MIN bytes per TU is often not a whole number. This register is used to hold the fractional component.<br>• [9:0] – The fraction part of ((LINK_BW/VIDEO_BW) × TRANSFER_UNIT_SIZE) scaled by 1000 is programmed in this register. |
| 0x59C | RW | INIT_WAIT_STREAM3: 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.<br>If (MIN_BYTES_PER_TU ≤ 4)<br>• [7:0] – Set INIT_WAIT to 64<br>else if color format is RGB/YCbCr_444<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)<br>else if color format is YCbCr_422<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)/2<br>else if color format is Y_Only<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)/3 |

*Table 2-25:* **MST Interface *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x5A0 | RW | MAIN_STREAM_HTOTAL_STREAM4. Specifies the total number of clocks in the horizontal framing period for the main stream video signal.<br>• [15:0] – Horizontal line length total in clocks. |
| 0x5A4 | RW | MAIN_STREAM_VTOTAL_STREAM4. Provides the total number of lines in the main stream video frame.<br>• [15:0] – Total number of lines per video frame. |
| 0x5A8 | RW | MAIN_STREAM_POLARITY_STREAM4. Provides the polarity values for the video sync signals.<br>• [1] – VSYNC_POLARITY: Polarity of the vertical sync pulse.<br>• [0] – HSYNC_POLARITY: Polarity of the horizontal sync pulse. |
| 0x5AC | RW | MAIN_STREAM_HSWIDTH_STREAM4. Sets the width of the horizontal sync pulse.<br>• [14:0] – Horizontal sync width in clock cycles. |
| 0x5B0 | RW | MAIN_STREAM_VSWIDTH_STREAM4. Sets the width of the vertical sync pulse.<br>• [14:0] – Width of the vertical sync in lines. |
| 0x5B4 | RW | MAIN_STREAM_HRES_STREAM4. Horizontal resolution of the main stream video source.<br>• [15:0] – Number of active pixels per line of the main stream video. |
| 0x5B8 | RW | MAIN_STREAM_VRES_STREAM4. Vertical resolution of the main stream video source.<br>• [15:0] – Number of active lines of video in the main stream video source. |
| 0x5BC | RW | MAIN_STREAM_HSTART_STREAM4. Number of clocks between the leading edge of the horizontal sync and the start of active data.<br>• [15:0] – Horizontal start clock count. |
| 0x5C0 | RW | MAIN_STREAM_VSTART_STREAM4. Number of lines between the leading edge of the vertical sync and the first line of active data.<br>• [15:0] – Vertical start line count. |
| 0x5C4 | RW | MAIN_STREAM_MISC0_STREAM4. Miscellaneous stream attributes.<br>• [7:0] – Implements the attribute information contained in the DisplayPort MISC0 register described in section 2.2.4 of the standard.<br>• [0] – Synchronous Clock.<br>• [2:1] – Component Format.<br>• [3] – Dynamic Range.<br>• [4] – YCbCr Colorimetry.<br>• [7:5] – Bit depth per color/component. |
| 0x5C8 | RW | MAIN_STREAM_MISC1_STREAM4. Miscellaneous stream attributes.<br>• [7:0] – Implements the attribute information contained in the DisplayPort MISC1 register described in section 2.2.4 of the standard.<br>• [0] – Interlaced vertical total even.<br>• [2:1] – Stereo video attribute.<br>• [6:3] – Reserved. |

*Table 2-25:* **MST Interface *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x5CC | RW | M-VID_STREAM4. 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 as automatically computed by the source core and written to the main stream. These values are not written into the M-VID register for readback.<br>• [23:0] – Unsigned M value. |
| 0x5D0 | RW | TRANSFER_UNIT_SIZE_STREAM4. Sets the size of a transfer unit in the framing logic On reset, transfer size is set to 64.<br>• [6:0] – This number should be in the range of 32 to 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). |
| 0x5D4 | RW | N-VID_STREAM4. 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 as automatically computed by the source core and written to the main stream. These values are not written into the N-VID register for readback.<br>• [23:0] – Unsigned N value. |
| 0x5D8 | RW | USER_PIXEL_WIDTH_STREAM4. Selects the width of the user data input port. Use quad pixel mode in MST.<br>• [2:0]:<br>◦ 1 = Single pixel wide interface<br>◦ 2 = Dual pixel wide interface<br>◦ 4 = Quad pixel wide interface |
| 0x5DC | RW | USER_DATA_COUNT_PER_LANE_STREAM4. This register is used to translate the number of pixels per line to the native internal 16-bit datapath.<br>If (HRES × bits per pixel) is divisible by 16, then<br>words_per_line = ((HRES × bits per pixel)/16)<br>Else<br>words_per_line = (INT((HRES × bits per pixel)/16)) + 1<br>For single-lane design:<br>Set USER_DATA_COUNT_PER_LANE = words_per_line - 1<br>For 2-lane design:<br>If words_per_line is divisible by 2, then<br>Set USER_DATA_COUNT_PER_LANE = words_per_line - 2<br>Else<br>Set USER_DATA_COUNT_PER_LANE = words_per_line + MOD(words_per_line,2) - 2<br>For 4-lane design:<br>If words_per_line is divisible by 4, then<br>Set USER_DATA_COUNT_PER_LANE = words_per_line - 4<br>Else<br>Set USER_DATA_COUNT_PER_LANE = words_per_line + MOD(words_per_line,4) - 4 |

*Table 2-25:* **MST Interface *(Cont'd)***

| Offset | R/W | Definition |
|---|---|---|
| 0x5E0 | RW | MAIN_STREAM_INTERLACED_STREAM4. 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 1 for the proper transmission of interlaced sources.<br>• [0] – Set to 1 when transmitting interlaced images. |
| 0x5E4 | RW | MIN_BYTES_PER_TU_STREAM4. Programs source to use MIN number of bytes per transfer unit. The calculation should be done based on the DisplayPort Standard.<br>• [7:0] – Set the value to INT((LINK_BW/VIDEO_BW) x TRANSFER_UNIT_SIZE) |
| 0x5E8 | RW | FRAC_BYTES_PER_TU_STREAM4. Calculating MIN bytes per TU is often not a whole number. This register is used to hold the fractional component.<br>• [9:0] – The fraction part of ((LINK_BW/VIDEO_BW) × TRANSFER_UNIT_SIZE) scaled by 1000 is programmed in this register. |
| 0x5EC | RW | INIT_WAIT_STREAM4. 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.<br>If (MIN_BYTES_PER_TU ≤ 4):<br>• [7:0] – Set INIT_WAIT to 64<br>else if color format is RGB/YCbCr_444<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)<br>else if color format is YCbCr_422<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)/2<br>else if color format is Y_Only<br>• [7:0] – Set INIT_WAIT to (TRANSFER_UNIT_SIZE - MIN_BYTES_PER_TU)/3 |
| 0x800 to 0x8FF | WO | PAYLOAD_TABLE. This address space maps to the VC payload table that is maintained in the core.<br>• [7:0] – Payload data |

### DisplayPort Audio

The DisplayPort Audio registers are listed in Table 2-26.

*Table 2-26:* **DisplayPort Audio Registers**

| Offset | R/W | Definition |
|---|---|---|
| 0x300 | R/W | TX_AUDIO_CONTROL. Enables audio stream packets in main link and provides buffer control.<br>• [0] Audio Enable for SST. In MST, Audio Enable for STREAM1.<br>• [1] Audio Enable for STREAM2 in MST. N/A for SST.<br>• [2] Audio Enable for STREAM3 in MST. N/A for SST.<br>• [3] Audio Enable for STREAM4 in MST. N/A for SST.<br>• 16] Set to 1 to mute the audio over link for SST. In MST, set to 1 to mute the Audio on STREAM1.<br>• [17] Set to 1 to mute the audio over link for MST STREAM2. N/A for SST.<br>• [18] Set to 1 to mute the audio over link for MST STREAM3. N/A for SST.<br>• [19] Set to 1 to mute the audio over link for MST STREAM4. N/A for SST. |
| 0x304 | R/W | TX_AUDIO_CHANNELS. Used to input active channel count. Transmitter collects audio samples based on this information.<br>• [2:0] Channel Count |
| 0x308 | WO | TX_AUDIO_INFO_DATA.<br>[31:0] Word formatted as per CEA 861-C Info Frame. Total of eight words should be written in following order:<br>• 1st word –<br>◦ [7:0] = HB0<br>◦ [15:8] = HB1<br>◦ [23:16] = HB2<br>◦ [31:24] = HB3<br>• 2nd word – DB3,DB2,DB1,DB0<br>....<br>• 8th word – DB27,DB26,DB25,DB24<br>The data bytes DB1...DBN of CEA Info frame are mapped as DB0-DBN-1.<br>No protection is provided for wrong operations by software. |
| 0x328 | R/W | TX_AUDIO_MAUD. M value of audio stream as computed by transmitter.<br>• [23:0] = Unsigned value computed when audio clock and link clock are synchronous. |

*Table 2-26:* **DisplayPort Audio Registers *(Cont'd)***

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

## HDCP Registers

For details about the HDCP registers, see the *HDCP Controller Product Guide* (PG224) [Ref 13]

## AXI Timer Registers

For details about the AXI Timer registers, see the *AXI Timer Product Guide* (PG079) [Ref 14]

# 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 as described in section 3.5.1.3 of the *VESA® DisplayPort Standard v1.2a* [Ref 1].

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 unlikely to be efficiently managed by a hardware-based state machine. Xilinx does not recommend using the RTL-based controllers.

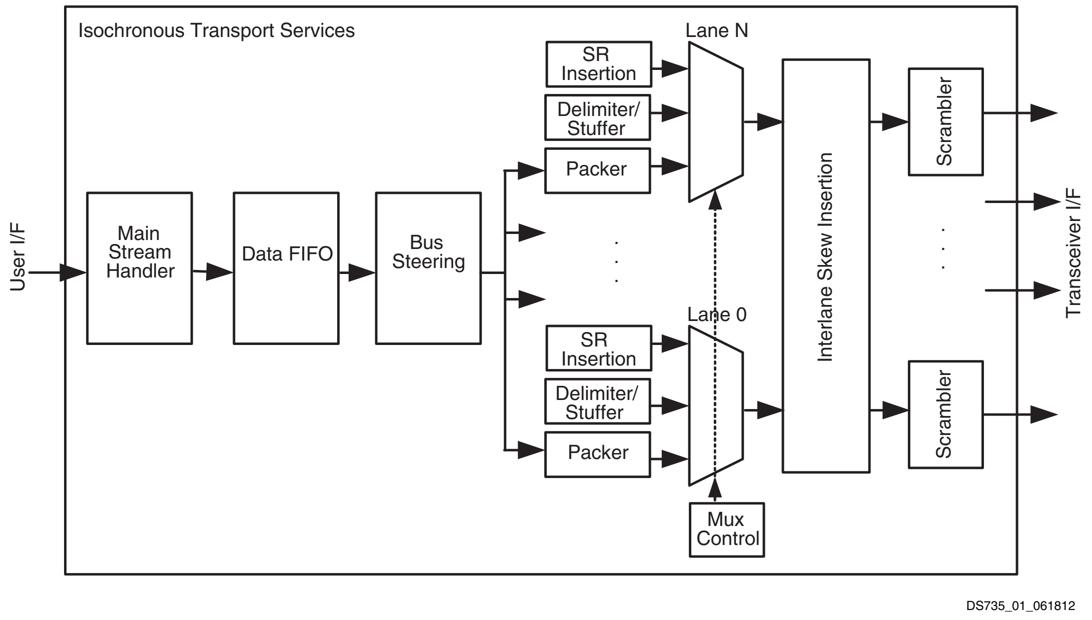


*Figure 3-1:* **Source Main Link Datapath**

### *Link Training*

The link training commands are passed from the DPCD register block to the link training function. When set to 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 v1.2a* [Ref 1].

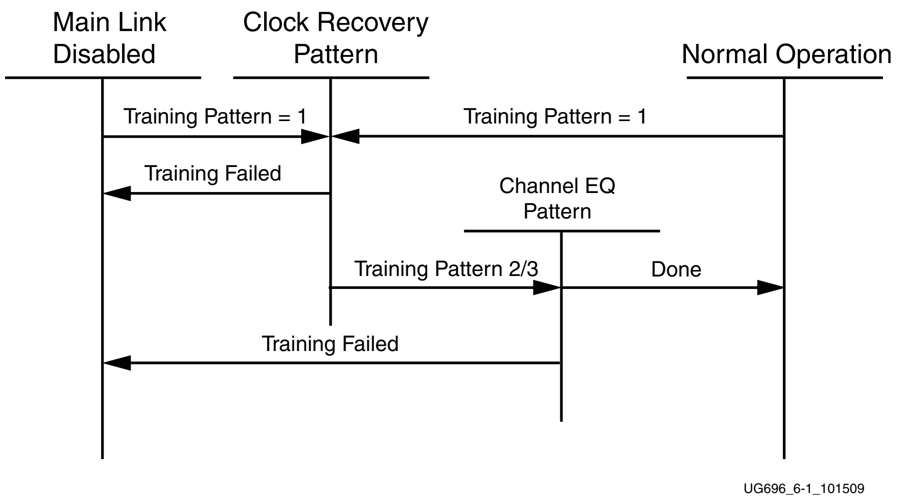


*Figure 3-2:* **Link Training States**

### 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 v1.2a* [Ref 1].

**IMPORTANT:** *During initialization ensure that TX8B10BEN is not cleared in offset 0x0070 of the corresponding Video PHY Controller Product Guide (PG230) [Ref 2].*

#### 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 the capability and needs of each link partner.
3. Write the configuration parameters to the link configuration field (0x00100 to 0x00101) of the DPCD through the AUX channel.

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

4. Equivalently, write the appropriate values to the local configuration space of the Source core.
    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 Procedure (Symbol Recovery, Interlane Alignment)**

1. Turn off scrambling and set training pattern 2 in the source through direct register writes.
    - SCRAMBLING_DISABLE = 0x01
    - TRAINING_PATTERN_SET = 0x02
2. Turn off scrambling and set training pattern 2 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.

***Note:*** Training pattern 3 replaces training pattern 2 for 5.4 Gb/s link rate devices. See the DisplayPort Standard v1.2a for details.

#### Enabling Main Link Video

The 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. When 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/descrambler 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/descrambler.*

## 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 v1.2a*.)

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.

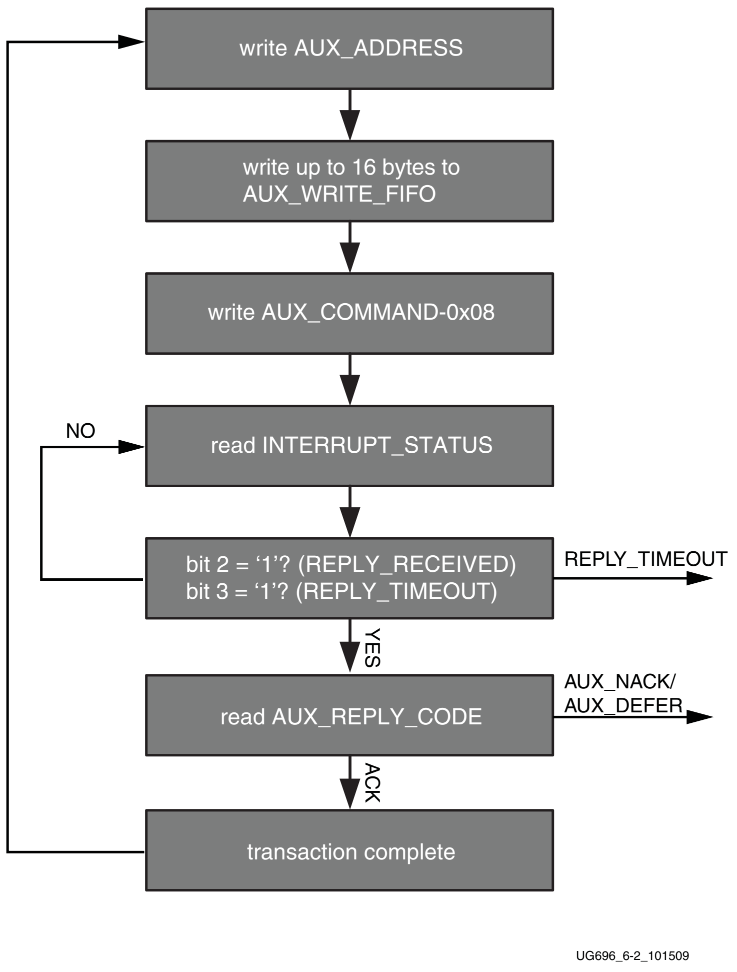


*Figure 3-3:* **AUX Write Transaction**

### *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.

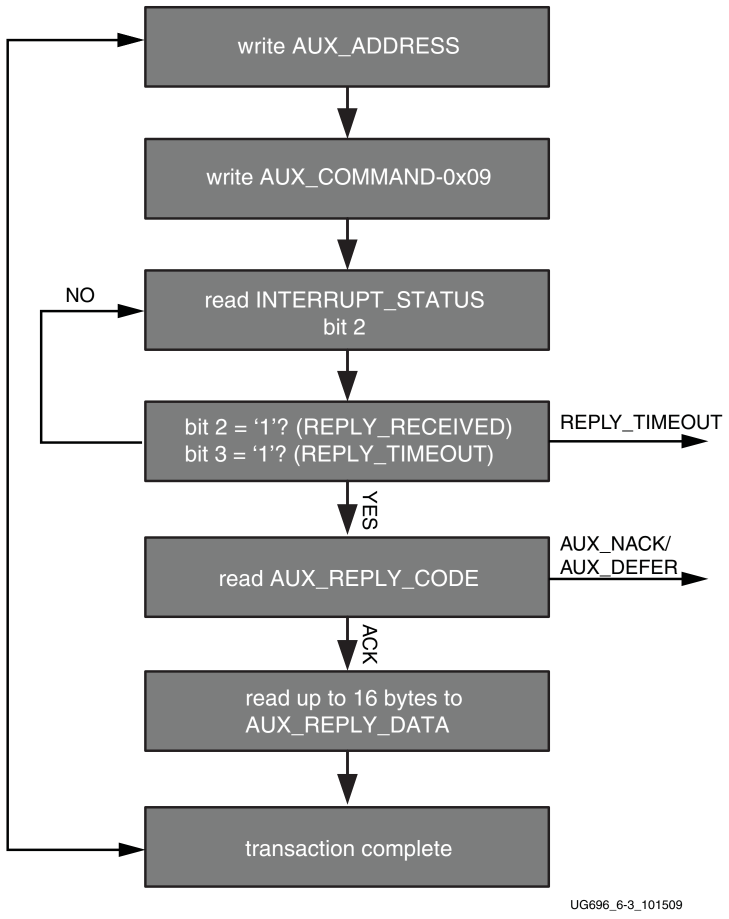


*Figure 3-4:* **AUX Read Transaction**

## I2C Over AUX 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 can emulate an I2C transaction.

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

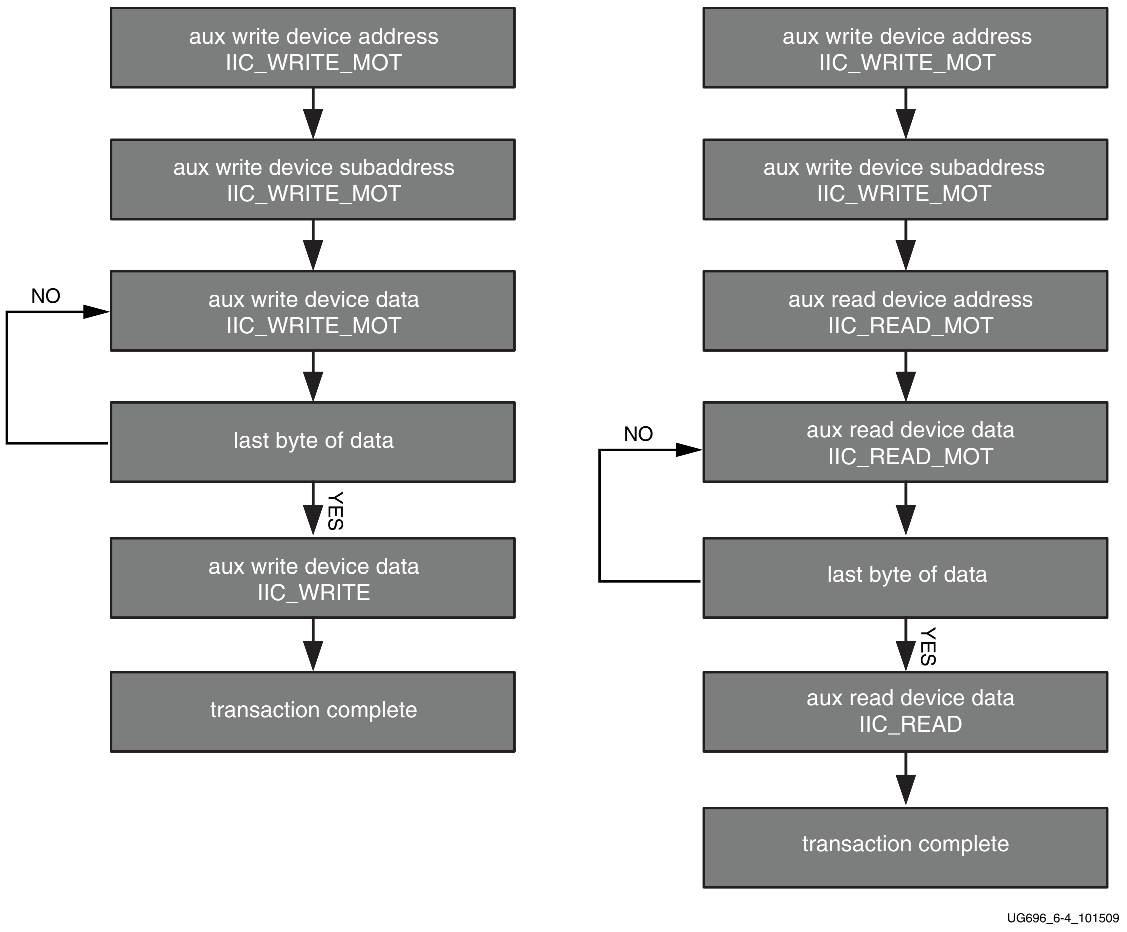


*Figure 3-5:* **Commanded I2C Device Transactions, Write (Left) and Read (Right)**

Because 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**

| AUX Transaction | Sequence |
|---|---|
| **Write Address only with MOT = 1** | |

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

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

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

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

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

| AUX Transaction | | Sequence |
|---|---|---|
| **Write Status with MOT = 1** | | |
| **AUX Transaction** | START ->CMD ->ADDRESS ->STOP | 1. Write AUX Address register (0x108) with device address.<br>2. Issue status update command to transmit transaction by writing into AUX command register (0x100). Bit[12] must be set to 1. |
| **I2C Transaction** | No transaction | |
| **Usage** | Status of previous write command that was deferred or partially ACKED. | |
| **Write Status with MOT = 0** | | |
| **AUX Transaction** | START ->CMD ->ADDRESS ->STOP | 1. Write AUX Address register (0x108) with device address.<br>2. Issue status update command to transmit transaction by writing into AUX command register (0x100). Bit[12] must be set to 1. |
| **I2C Transaction** | Force a STOP and the end of write burst | |
| **Usage** | Status of previous write command that was deferred or partially ACKED. MOT = 0 ensures that the bus returns to IDLE at the end of the burst. | |

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 unidirectional/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 v1.2a*. 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, the 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.

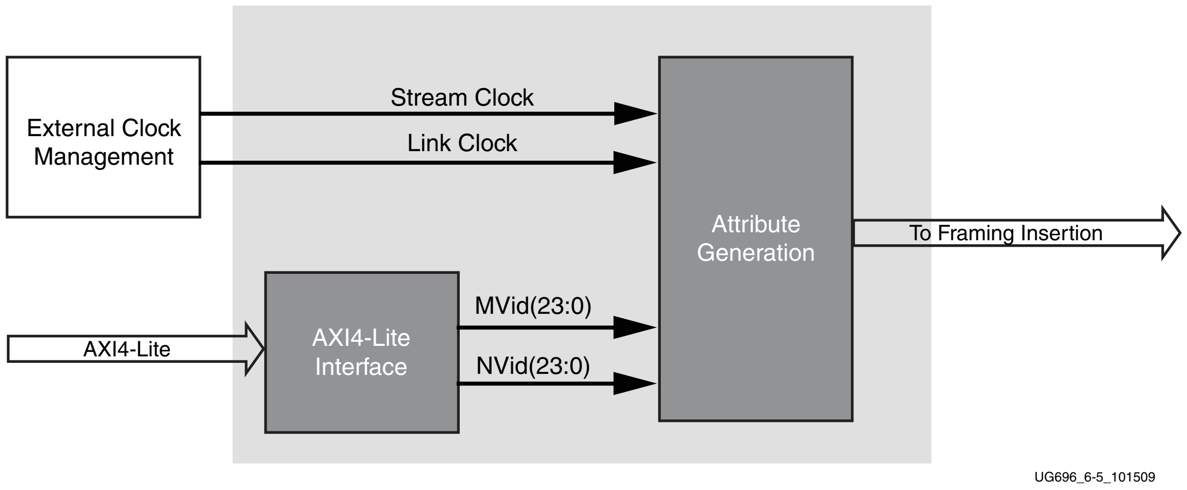


*Figure 3-6:* **Transmitter Audio/Video Clock Generation**

## Hot Plug Detection

The Source device must de-bounce 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 v1.2a* [Ref 1].

## 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.

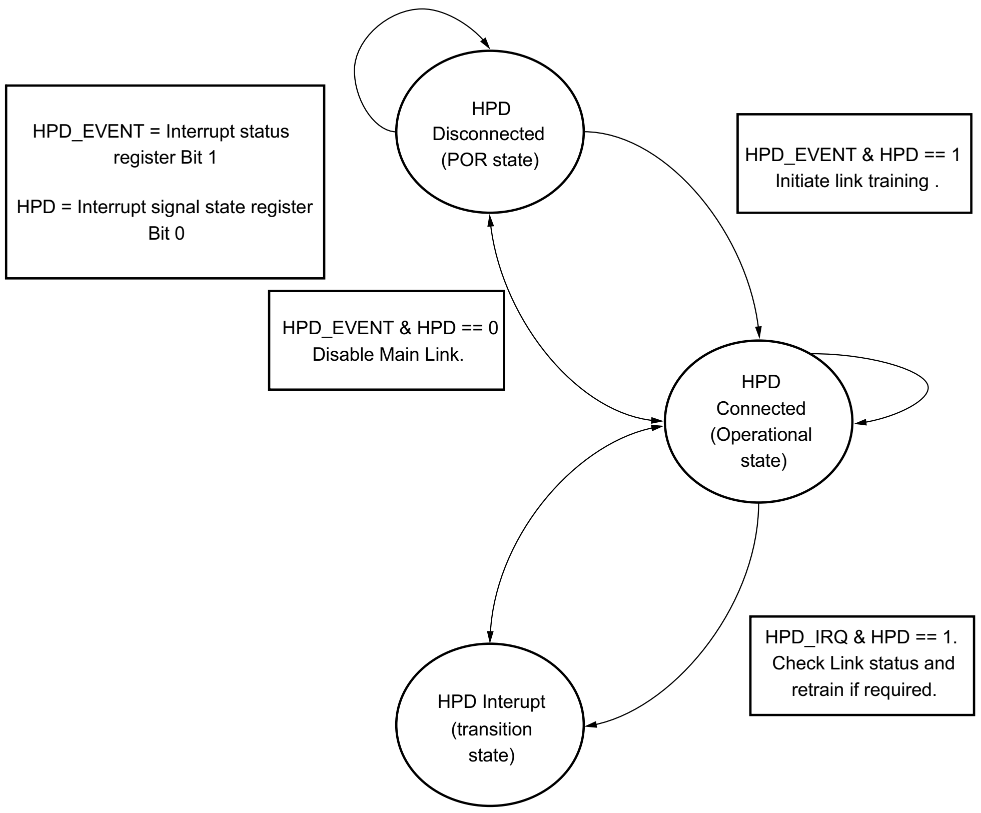

HPD_EVENT = Interrupt status register Bit 1
HPD = Interrupt signal state register Bit 0
HPD Disconnected (POR state)
HPD_EVENT & HPD == 1 Initiate link training .
HPD_EVENT & HPD == 0 Disable Main Link.
HPD Connected (Operational state)
HPD_IRQ & HPD == 1. Check Link status and retrain if required.
HPD 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 v1.2a 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.

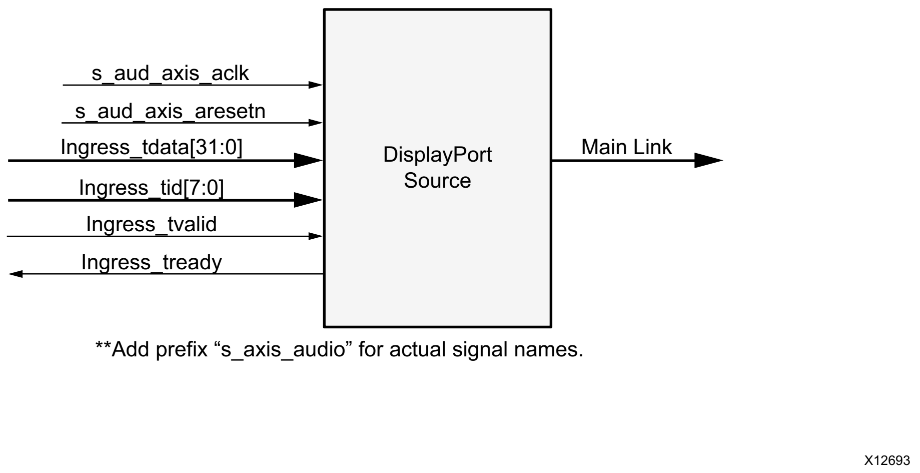


*Figure 3-8:* **Audio Data Interface of DisplayPort Source System**

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 streaming 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 2-26).

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 TX core.
2. Wait until video/audio clock is recovered and stable.
3. Enable audio in DisplayPort 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 the sink once per every video frame or 8192 cycles.

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

1. Disable audio (because 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).

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.

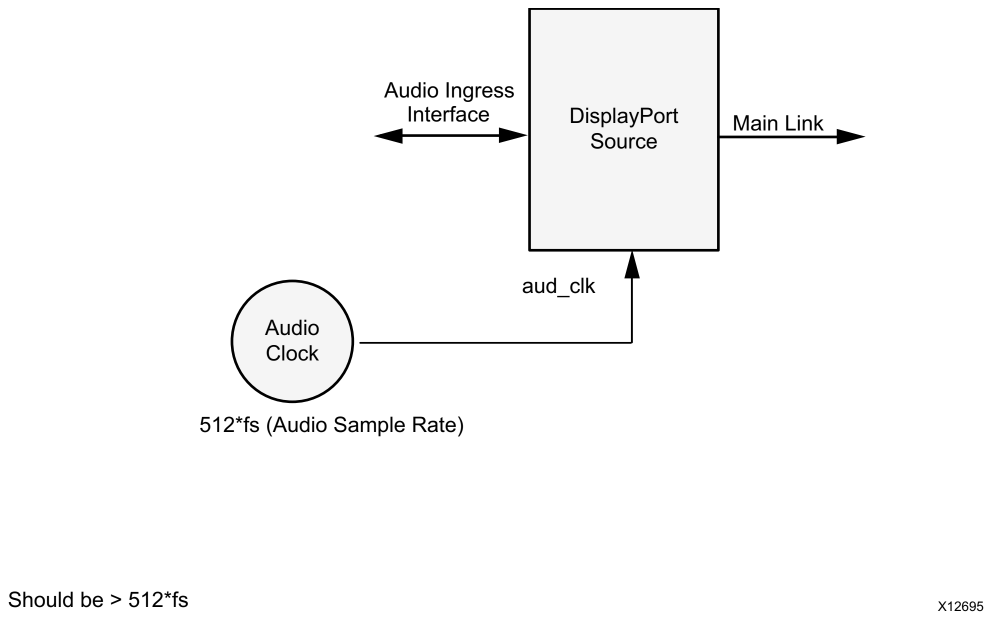


*Figure 3-9:* **Source: Audio Clocking**

## Programming the Core in MST Mode

The section details the steps to program the core in MST mode.

### *Enabling MST*

The following steps are recommended to enable MST functionality:

1. Bring up the main link by following training procedure.
2. Send sideband messages using the AUX channel to discover the link (how many downstream nodes are connected and their capabilities).
3. Enable MST by writing 1 to bit 0 of the MST Config register.
4. Discover MST downstream devices as recommended in section 1.2.1 in the *DisplayPort Standard*.
5. Allocate timeslots based on configuration and the Sink Payload Bandwidth Number (PBN). Typical sideband messages used before VC Payload allocation are Link Address Request, Clear Payload Table, and Enumerate Path Resources.
    a. Program VC Payload Buffer 12'h0x800 onwards as per allocation requirement.
    b. Program the Sink core with the same allocation timeslots using AUX channel as described in section 2.6.4 in the *DisplayPort Standard*.
    c. Wait until sink accepts allocation programming (check DPCD reads to monitor status).

   d. After the sink sets VC Payload Allocated (DPCD Address = 0x02C0), set VC Payload Allocated bit in MST Config register (12'h0x0D0). This enables the source controller to send an ACT trigger.

6. Wait until ACT Handled bit is set in DPCD Address (0x02C0).
7. Program Video attributes for required streams. Program user pixel width to 4 for all the streams.
8. Program Rate Governing registers 0x1D0, 0x1D4, 0x1D8, and 0x1DC based on the stream requirement.
   - Program TRANSFER UNIT Size = # of timeslots allocated for that stream. (VC payload size source)
   - Program FRAC_BYTES_PER_TU = TS_FRAC
   - Program MIN_BYTES_PER_TU = TS_INT
   - Program INIT_WAIT = 0

   ***Note:*** Repeat step 7 for each steam.
9. Enable MST by writing 1 to bit 0 of MST Config register.

After these steps are done, the source controller starts sending MST traffic as per VC Payload programming in the main link.

### *Payload Bandwidth Management*

The following steps manage payload bandwidth in the source controller.

1. Calculate Target_Average_StreamSymbolTimeSlotsPerMTP based on the *DisplayPort Standard v1.2* or later. Program VC payload size with calculated Target_Average_StreamSymbolTimeSlotsPerMTP and align it with nearest even boundary.

   For example if the value is 13, program VC payload size for this particular stream to 14.
2. In MST mode when GT data width is 4 bytes the VC Payload should be multiple of 4.
3. The VC payload calculation for (1920x2200) stream, RGB color sampling, 8 Bits Per Color at 5.4 Gb/s, 4 lanes is given here.

   VC Payload Band width = LINK_RATE × Lane_count × 100 (see Table 2-61 in the *VESA DisplayPort Standard v1.2a* [Ref 1])

   = 5.4 × 4 × 100 = 2160

Average Stream symbol Time slot per MTP = (Pixel_rate × Bits_per_pixel/8/VC Payload_Band width) × 64 (see Section 2.6.3.3 in the *VESA DisplayPort Standard v1.2a* [Ref 1])

= (297 MHz × 24 /8/2160) × 64 = 26.4

VC Payload Size = 2/4 symbol aligned of (Average Stream symbol Time slot per MTP)

= 28

4. Program VC Payload table as defined in DPCD standard.
5. Program VC Payload table in source controller as defined in registers 12'h0x800 – 12'h0x8FC.

## 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. Similarly for MST mode, `tx_vid_clk_stream<n>` and `s_axis_aclk_stream<n>` can be same or the `s_axis_aclk_stream<n>` can be at a higher frequency than `tx_vid_clk_stream<n>`.

The `tx_lnk_clk` is a link clock input to the DisplayPort TX Subsystem generated by the Video PHY (GT). The frequency of `tx_lnk_clk` is `<line_rate>`/40 MHz for the 32-bit video PHY(GT) data interface and `<line_rate>`/20 MHz for the 16-bit interface. See Table 3-3 for the recommended values.

In the 16-bit GT interface, the `hdcp_ext_clk` input must be driven from an external MMCM where it has a frequency requirement of `hdcp_ext_clk` = `tx_lnk_clk`/2 MHz.

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

*Table 3-3:* **Clocking**

| Resolution | AXI4-Stream (s_axis_aclk_stream1) | Video Pipe (m_aclk_stream1) | User Video Clock (tx_vid_clk_stream1) |
|---|---|---|---|
| UHD at the 60 fps (frame split mode) | 148.5(1) | 74.25(1) | 74.25(1) |
| Other Modes | Video Clock(2) | Video Clock(2) | Video Clock(2) |

**Notes:**
1. For MST stream 1 and stream 2 only.
2. For all four streams when MST mode is enabled. See DMT/CEA spec for video clock range for each DMT resolution.

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 up to 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 up to 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.

For more information on clocking, see the *Video PHY Controller Product Guide* (PG230) [Ref 2].

## 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. For MST with four streams, there are four video resets.
- `s_axis_aresetn_stream1`: Active-Low AXI4-Stream interface reset. For MST with four streams, there are four resets corresponding to each stream.
- `m_aresetn_stream1`: Active-Low reset for streams one and two.

## Address Map Example

Table 3-4 shows an example based on a subsystem base address of `0x44C0_0000` (19 bits). The DisplayPort 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 AXI4-Stream Interface mode. In the mode, the Dual Splitter and Video Timing Controller are not present.

*Table 3-4:* **Address Map Example**

| | SST | MST |
|---|---|---|
| DisplayPort TX Core | 0x44C0_0000 | 0x44C0_0000 |
| Dual Splitter | N/A | 0x44C0_1000 |
| VTC 0 | 0x44C0_1000 | 0x44C0_2000 |
| VTC 1 (N = 2) | N/A | 0x44C0_3000 |
| VTC 2 (N = 3) | N/A | 0x44C0_4000 |
| VTC 3 (N = 4) | N/A | 0x44C0_5000 |
| HDCP Controller | 0x44C0_2000 | N/A |
| AXI Timer | 0x44C0_3000 | N/A |

## Programming Sequence

This section contains the programming sequence for the subsystem using UHD@60 in MST mode with two streams. Program and enable the components of the DisplayPort TX Subsystem in the following order. HDCP Controller and AXI Timer address map exist when HDCP is enabled in SST mode.

1. DisplayPort TX Core
2. Dual Splitter
3. Video Timing Controller

### Dual Splitter Programming

Use the following steps to program the Dual Splitter.

1. Write 0x02 in GENR_CONTROL_REG. This begins the programming sequence and the Dual Splitter register update bit is set.
2. Write vertical resolution and horizontal resolution in TIME_CONTROL_REG.

3. The Dual Splitter is used in a configuration where the input frame must be split into two vertical halves. Write the overlap, number of segments, output samples per clock and input samples per clock in CORE_CONTROL_REG.

   For 4k frame split mode, write 0x02_04_04 to register 0x100 (number of segments = 2; number of samples per clock at output = 4; number of samples per clock at input = 4).

   For other modes, write 0x010404 (number of segments =1 (bypass); number of output samples = 4; number of input samples = 4).

4. Write 0x03 in GENR_CONTROL_REG to enable the Dual Splitter for programmed resolutions and splitting functionality.

When programming the Dual Splitter, note the following:

- There should be no overlap of the two segments in a frame.
- Segment 0 of the Dual Splitter is the left frame and Segment 1 is the right frame.
- The timing of two segments of the splitter is independent, but by the start of a new line, both the segments complete the previous line.
- For UHD @ 60 in frame split mode, the width of the frame (HRES) must be equal to actual HRES/4.

# 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 3]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 5]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 6]

## 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 3] 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:

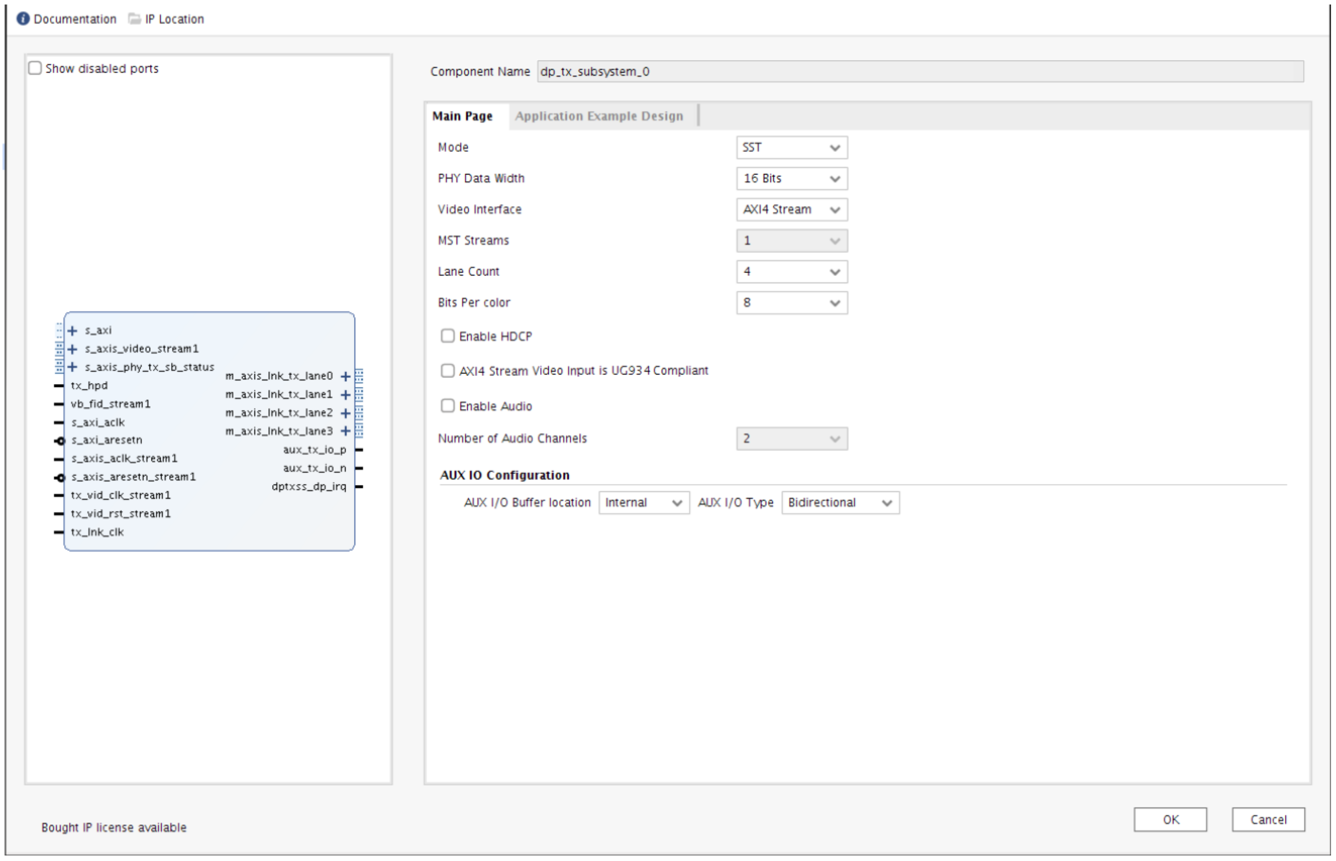
1. Select the subsystem from the IP catalog.
2. Double-click the selected subsystem or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 5].

***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_subsystem_0

Mode SST

PHY Data Width 16 Bits

Video Interface AXI4 Stream

MST Streams 1

Lane Count 4

Bits Per color 8

☐ Enable HDCP

☐ AXI4 Stream Video Input is UG934 Compliant

☐ Enable Audio

Number of Audio Channels 2

AUX IO Configuration

AUX I/O Buffer location Internal AUX I/O Type Bidirectional

Bought IP license available

*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 dp_tx_subsystem_n (n depends on the number of DisplayPort instances in the design. (n = 0 for the first instance added to the design.).
- **Mode**: Selects the desired mode for the video stream out. Options are SST or MST.
- **PHY Data Width**: Selects 16-bit or 32-bit GT data width.
- **Video Interface**: Selects AXI4-Stream or native for the video interface.
- **Pixel Mode**: Enables when native interface is selected. Select single, dual or quad pixel mode. Maximum pixel mode supported is aligned with lane count. Pixel mode can be changed dynamically through software but this does not affect the video streaming width.

- **MST Streams**: Enables when MST mode is selected. Selects the maximum number of streams in MST mode. This is configured through software up to the maximum value. Valid options are 2, 3, or 4.
- **Lane Count**: Selects the maximum number of lanes. Lane count can be changed dynamically through software. Valid options are 1, 2, or 4.
- **Bits Per Color**: Selects the desired maximum bit per component (BPC). Valid options are 8, 10, 12, or 16. This can be changed in software up to the maximum value.
- **Enable HDCP**: Enables HDCP encryption. Available only when SST mode is selected.

  ***Note:*** HDCP requires a separate license.
- **AXI4-Stream Video Input is UG934-Compliant**: Select this option to make the AXI4-Stream input-compliant with UG934 (see the *AXI4-Stream Video IP and System Design Guide* (UG934) [Ref 15]).
- **Enable Audio:** Enables audio support. Available only when SST mode is selected.
- **Number of Audio Channels**: Enables only when SST mode is selected and audio support enabled. Selects the number of audio channels. Valid options are 2 to 8.
- **AUX I/O Buffer location**: Selects buffer location for AUX channel. Valid options are Internal or External.
- **AUX I/O Type**: Selection of bidirectional or unidirectional buffer type.

## 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).

*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 |
| Pixel Mode | PIXEL_MODE | Quad |
| MST Streams | NUM_STREAMS | 1 |
| Lane Count | LANE_COUNT | 4 |
| Bits Per Color | BITS_PER_COLOR | 8 |
| Enable HDCP | HDCP_ENABLE | 0 |
| AXI4-Stream Video Input is UG934-Compliant | UG934_COMPLIANCE | 0 |
| Enable Audio | AUDIO_ENABLE | 0 |
| Number Of Audio Channels | AUDIO_CHANNELS | 2 |

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

| Vivado IDE Parameter/Value | User Parameter/Value | Default Value |
|---|---|---|
| AUX IO Buffer Location | AUX_IO_LOC | Internal |
| AUX IO Type | AUX_IO_TYPE | Bidirectional |

### Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4].

## 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 2].

### 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 2].

## Transceiver Placement

For more information on the specific transceiver placement constraints, see the *Video PHY Controller Product Guide* (PG230) [Ref 2].

## I/O Standard and Placement

This section contains details about I/O constraints.

### *AUX Channel*

The *VESA DisplayPort Standard* [Ref 1] 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 Kintex®-7 devices 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 Kintex-7 and Virtex®-7 devices 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 Kintex-7 devices supporting HR IO banks, use the following constraints:

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

For Virtex-7 devices 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 TX Subsystem.

## Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 4].

# Example Design

***Note:*** All example designs use the TED DP1.2 FMC Card from inrevium (TB-FMCH-DP3).

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 1.8 V section. For more information, see the ZCU102 System Controller – GUI Tutorial (XTP433) [Ref 16].*

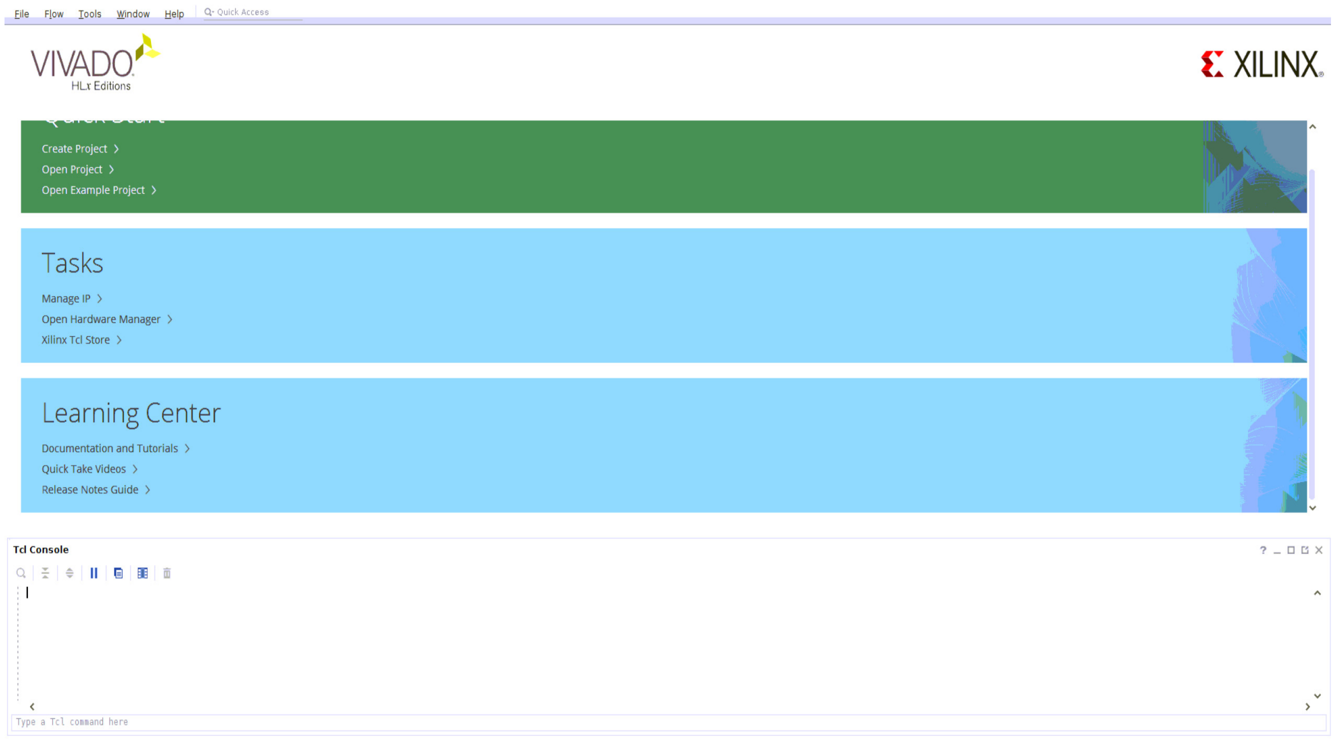
Table 5-1 shows the available example designs for DisplayPort TX.

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

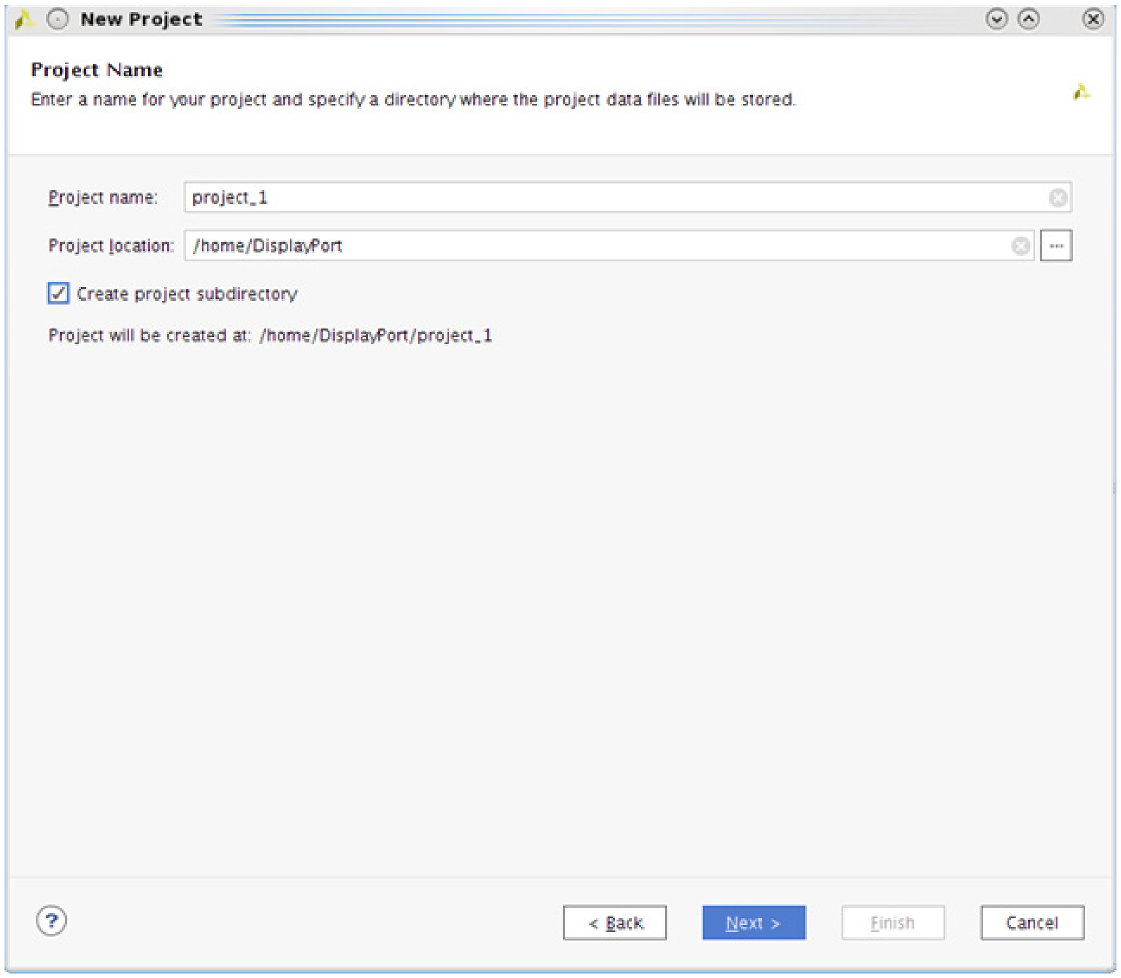
| GT Type | Topology | Video PHY Config (TXPLL) | Video PHY Config (RXPLL) | Hardware | GT Data Width | BPC | Processor |
|---|---|---|---|---|---|---|---|
| GTXE2 | Pass-through with HDCP1.3 | CPLL | CPLL | KC705 + TED DP1.2 FMC | 2-byte | 10 | MicroBlaze |
| | Pass-through without HDCP1.3 | CPLL | CPLL | KC705 + TED DP1.2 FMC | 4-byte | 10 | MicroBlaze |
| GTHE3 | Pass-through without HDCP1.3 | QPLL | CPLL | KCU105 + TED DP1.2 FMC | 2-byte | 10 | MicroBlaze |
| GTHE4 | RX only | – | CPLL | ZCU102 + TED DP1.2 FMC | 2-byte | 10 | A53 |
| | TX only | QPLL | – | ZCU102 + TED DP1.2 FMC | 2-byte | 10 | A53 |

## Building the Example Design

1. Open the Vivado Design Suite and click **Create Project**.



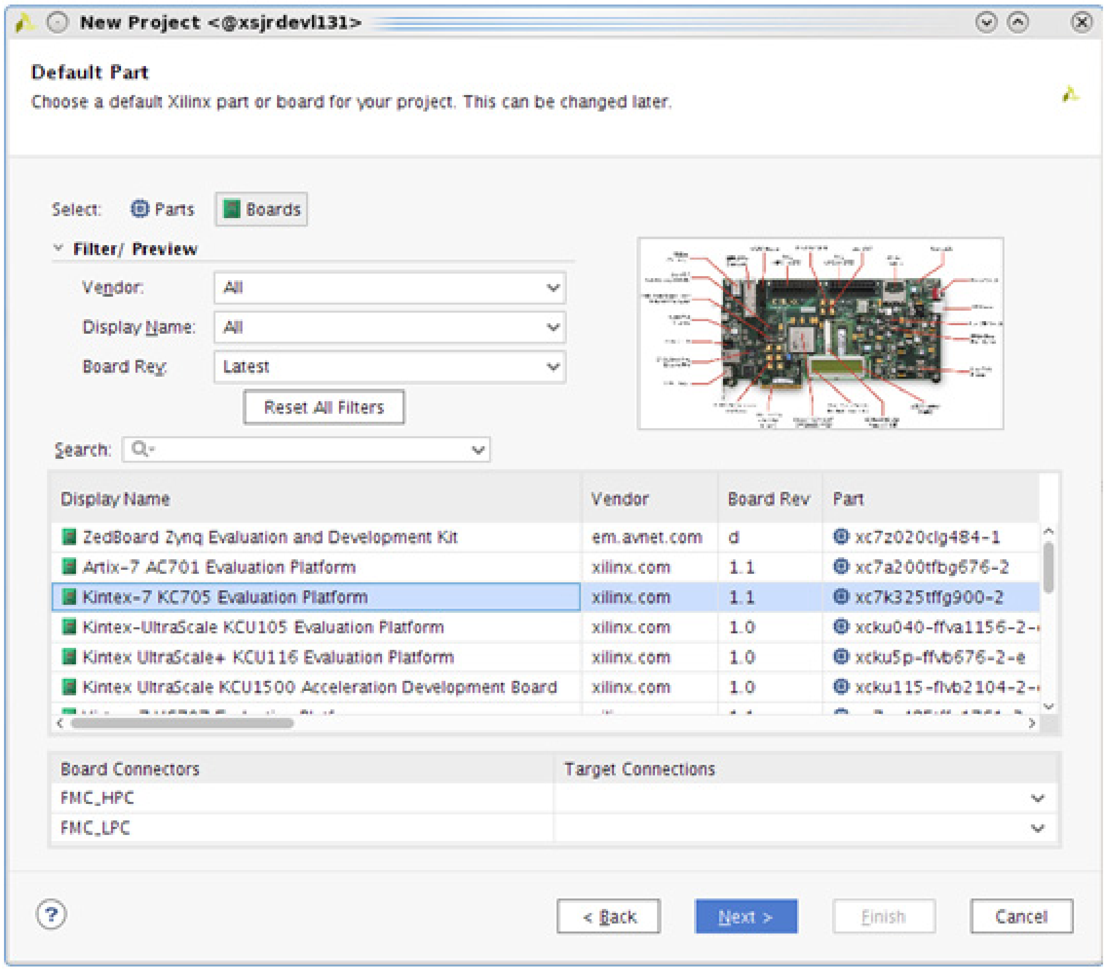
2. In the **New Project** window, enter a **Project name**, **Project location**, and click **Next** up to the Board/Part selection window.



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

3. In the **Default Part** window, select the Board as per your requirement. Application Example Designs are available for KC705, KCU105, and ZCU102. As an example, the Kintex®-7 KC705 board is selected.



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

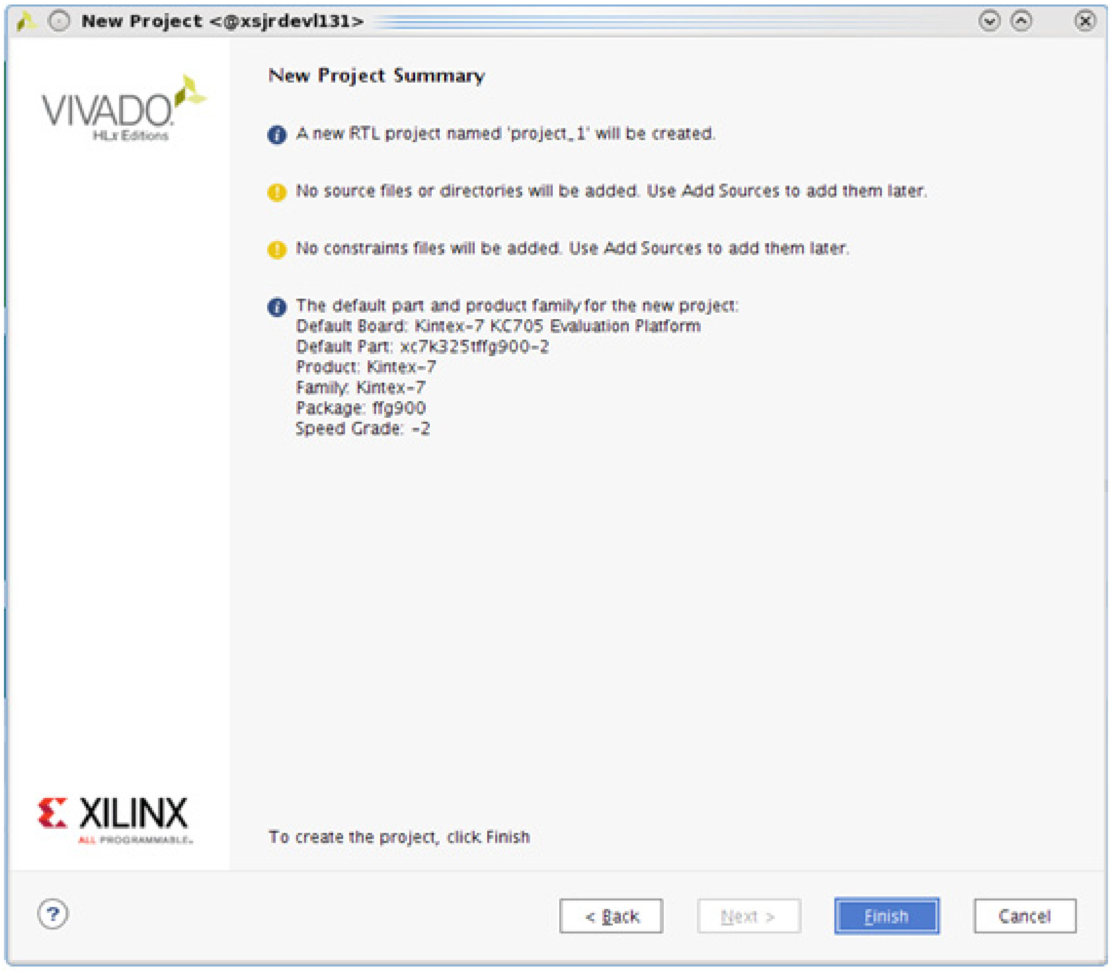
Board Rev: Latest

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- |
| 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- |

| Board Connectors | Target Connections |
|---|---|
| FMC_HPC | |
| FMC_LPC | |

4. Click **Finish**.



New Project Summary

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

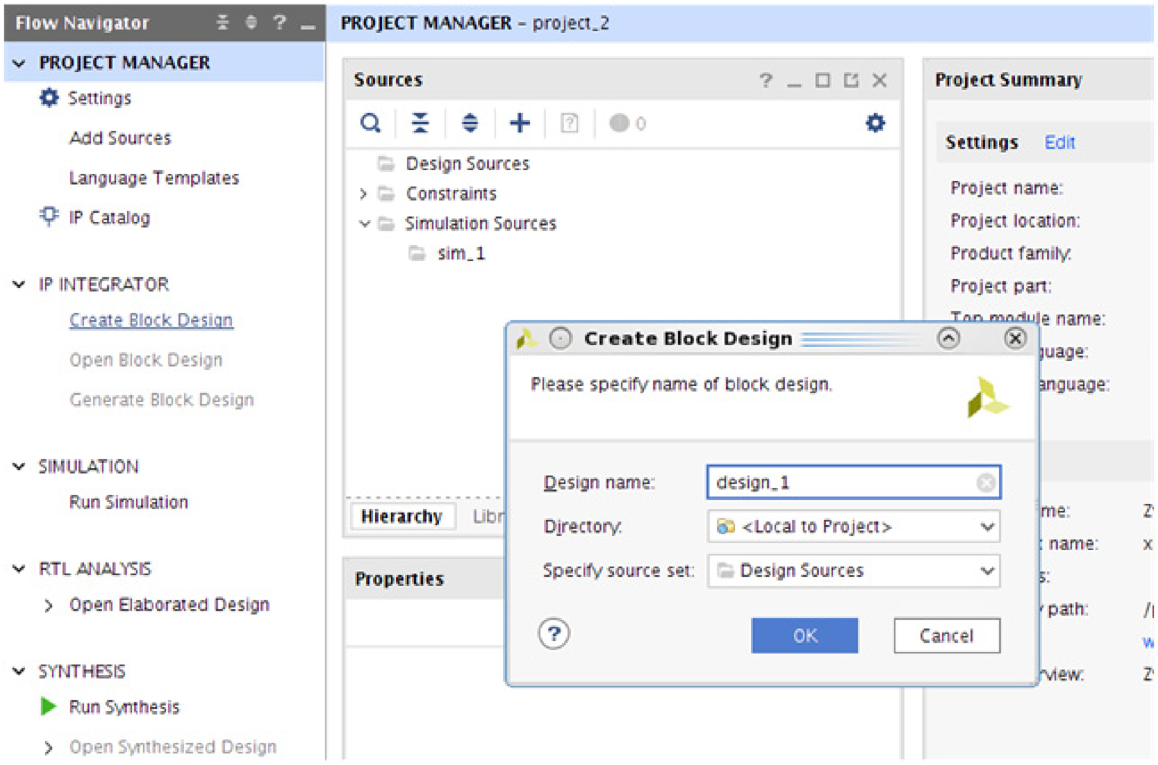
No source files or directories will be added. Use Add Sources to add them later.

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-7 KC705 Evaluation Platform
Default Part: xc7k325tffg900-2
Product: Kintex-7
Family: Kintex-7
Package: ffg900
Speed Grade: -2

To create the project, click Finish

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



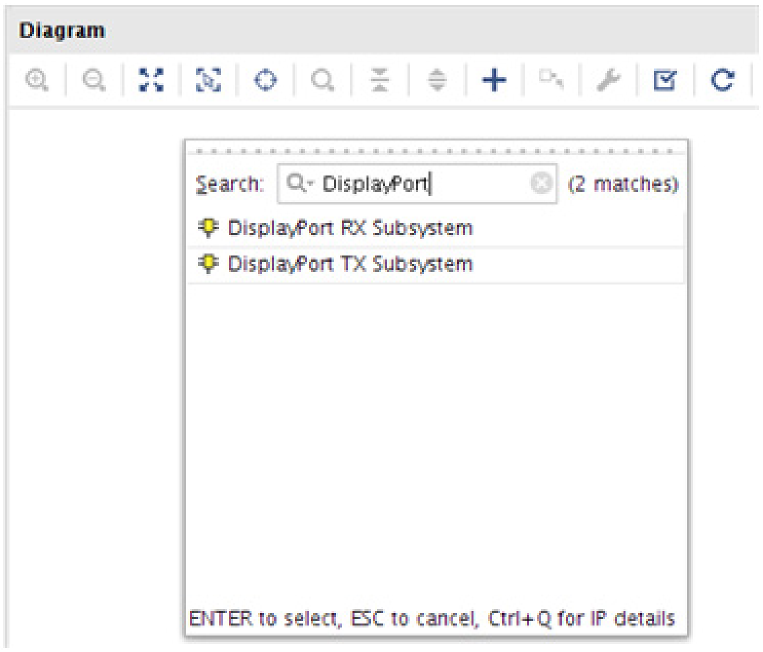
Please specify name of block design.

Design name: design_1

Directory: <Local to Project>

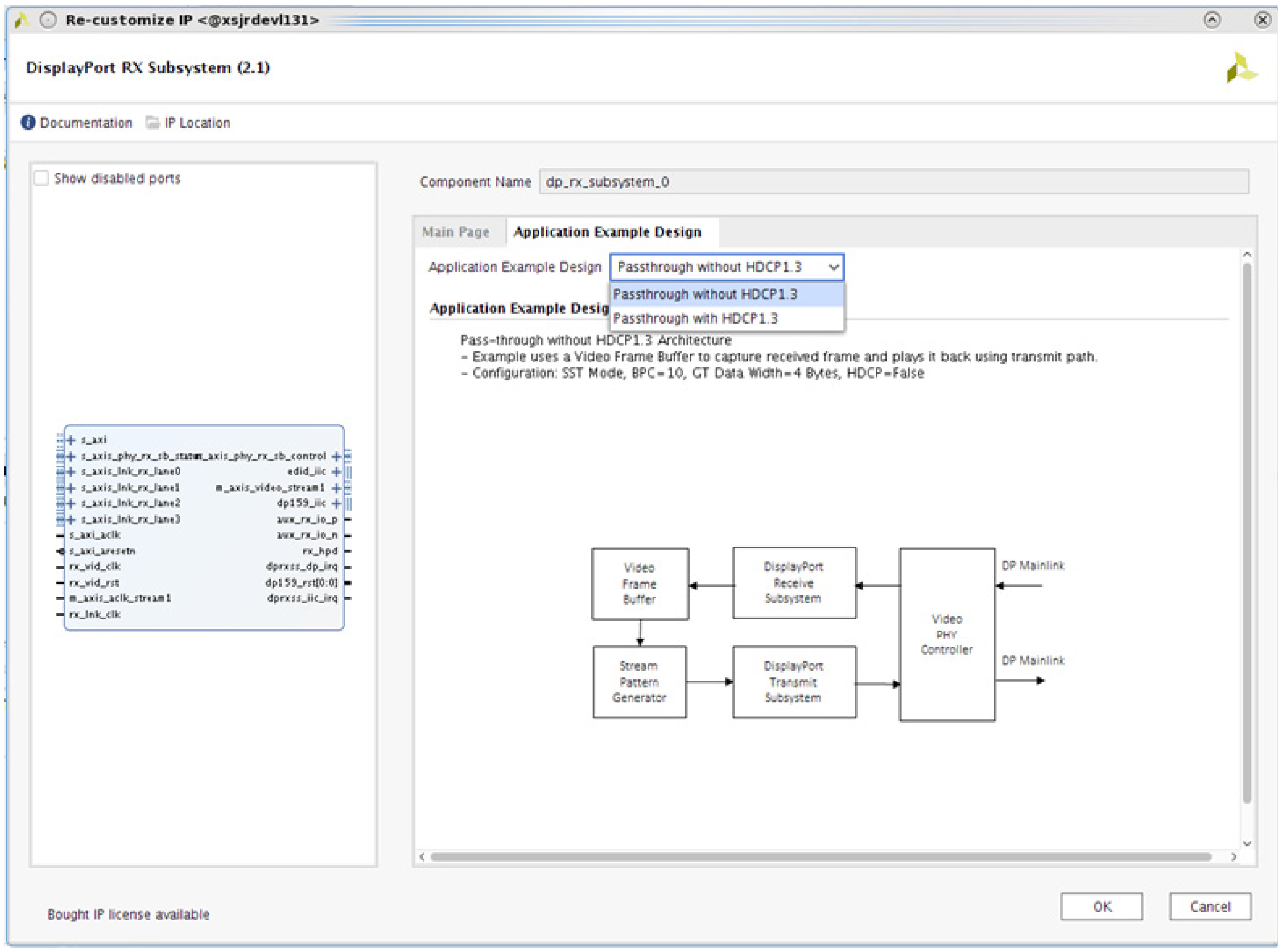
Specify source set: Design Sources

6. Right-click BD and click **Add IP**. Search for DisplayPort and select either the DisplayPort RX Subsystem IP (for RX only (ZCU102) or Pass-through (KC705, KCU105) designs) or the DisplayPort TX Subsystem IP (for TX only (ZCU102) or Pass-through (KC705, KCU105) designs).



ENTER to select, ESC to cancel, Ctrl+Q for IP details

7. Double-click the IP and go to the **Application Example Design** tab in the **Customize IP** window. Select the supported topology in the **Application Example Design** drop-down box. Click **OK** and **Save** the block design.



Re-customize IP <@xsjrdevl131>

DisplayPort RX Subsystem (2.1)

Component Name: dp_rx_subsystem_0

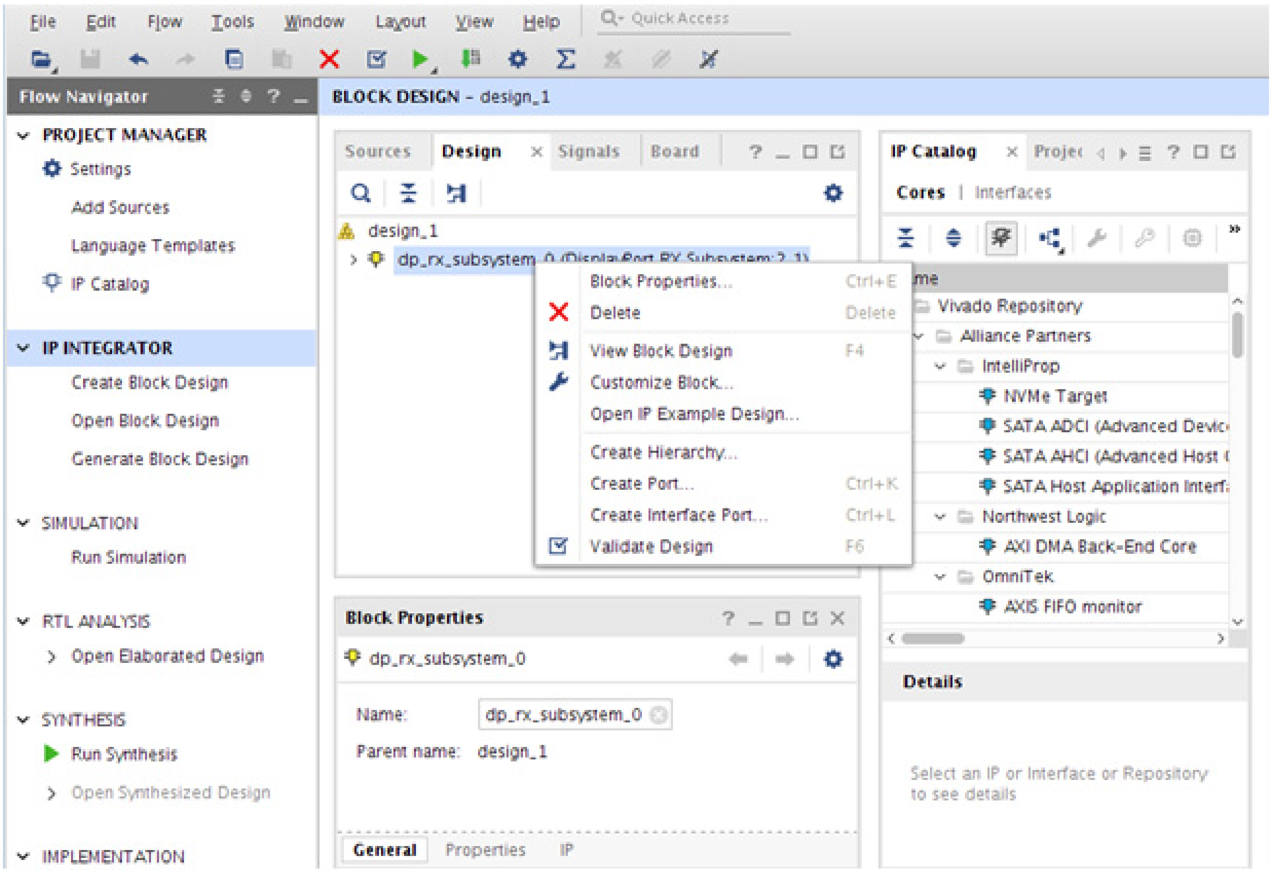
Application Example Design: Passthrough without HDCP1.3

Application Example Design

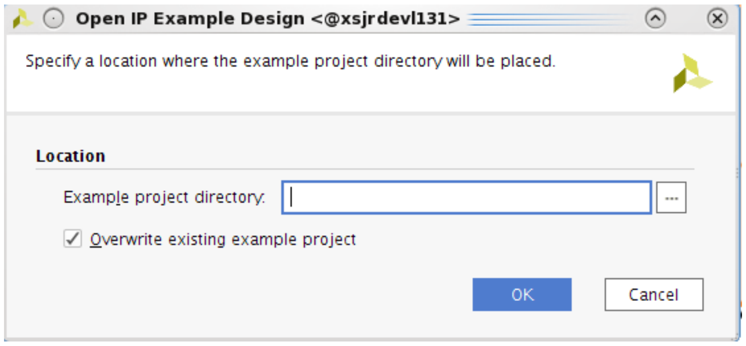
Pass-through without HDCP1.3 Architecture
- Example uses a Video Frame Buffer to capture received frame and plays it back using transmit path.
- Configuration: SST Mode, BPC=10, GT Data Width=4 Bytes, HDCP=False

Bought IP license available

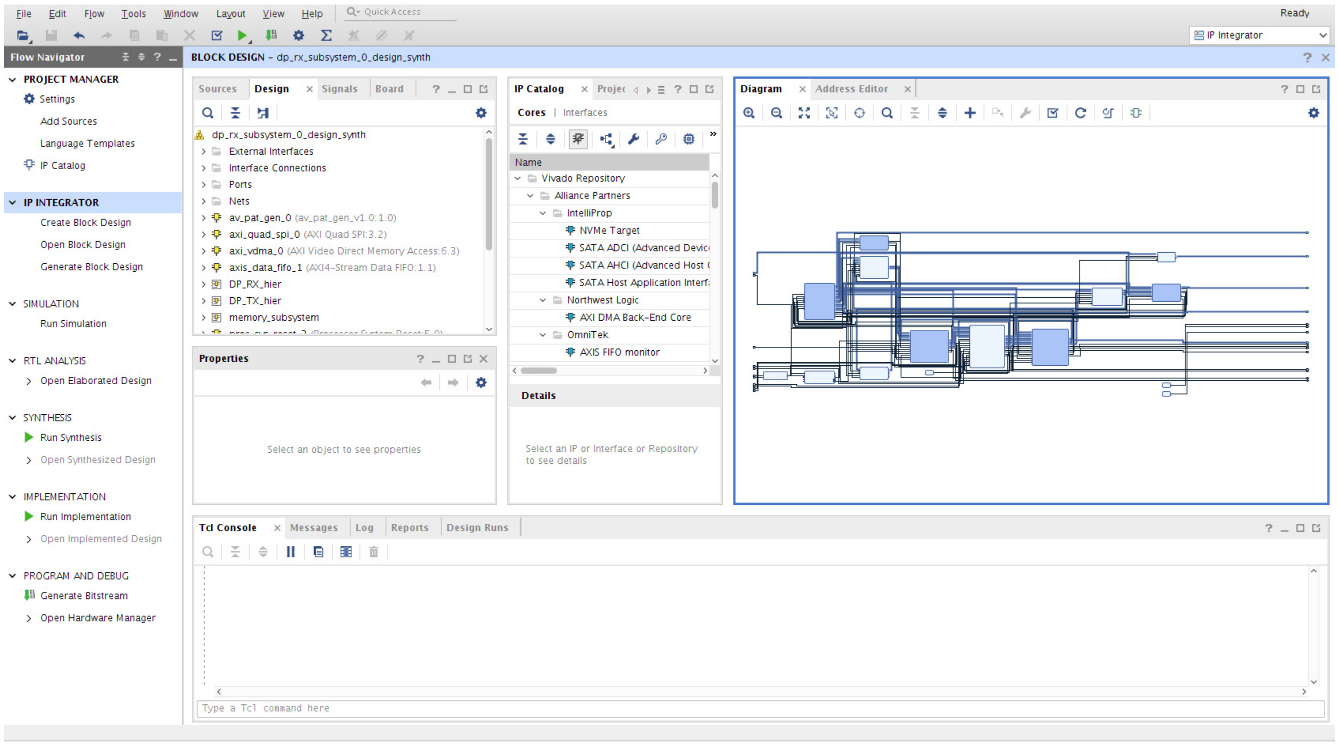
8. Right-click the **DisplayPort Subsystem** IP under Design source in the **Design** tab and click **Open IP Example Design**.



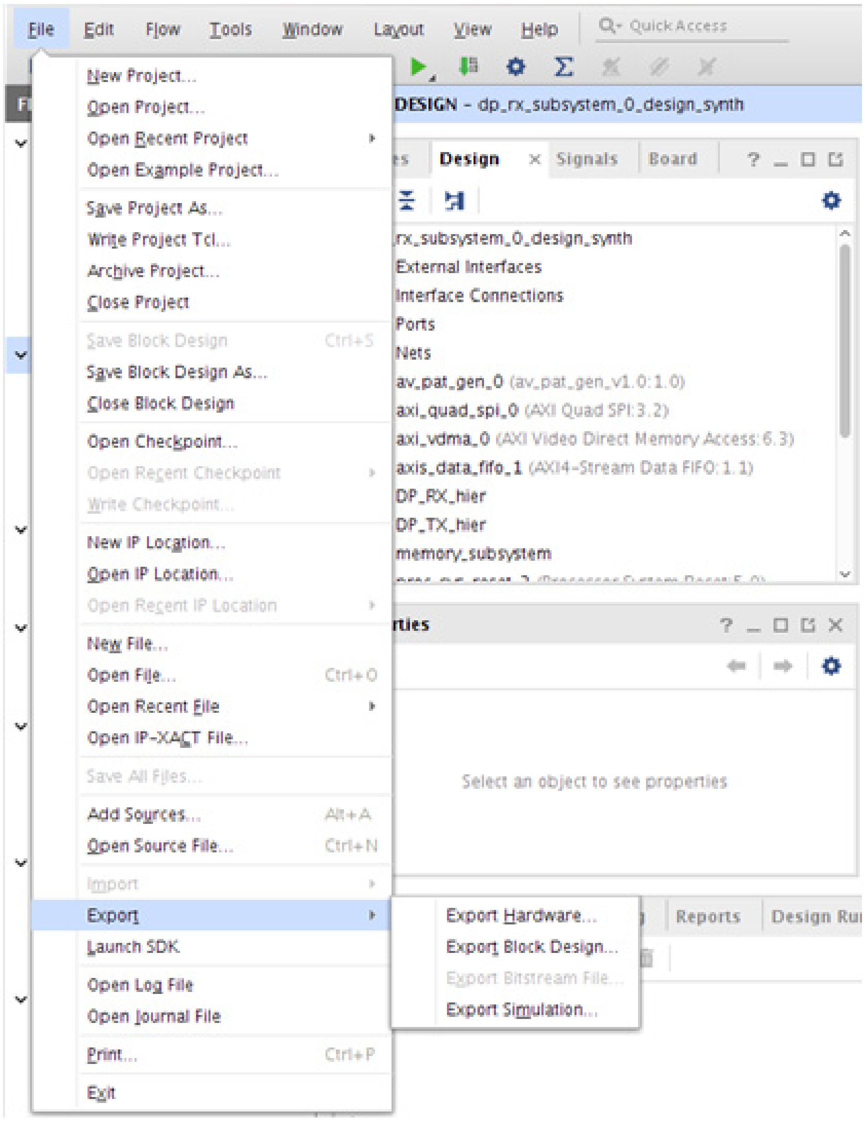
9. Choose **Example project directory** and click **OK**.


Open IP Example Design <@xsjrdevl131>
Specify a location where the example project directory will be placed.
Location
Example project directory:
Overwrite existing example project
OK
Cancel

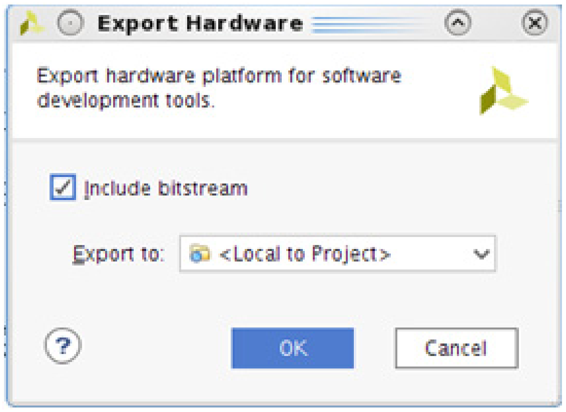
10. The following figure shows the Vivado IP integrator design. Choose the **Generate Bitstream**.



11. Export the hardware to SDK. Click **File > Export > Export Hardware**.



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



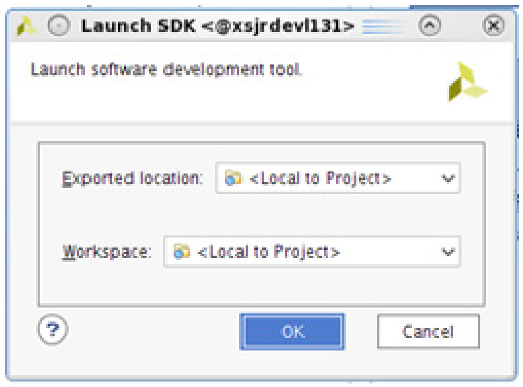
Export Hardware

Export hardware platform for software development tools.

☑ Include bitstream

Export to: <Local to Project>

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



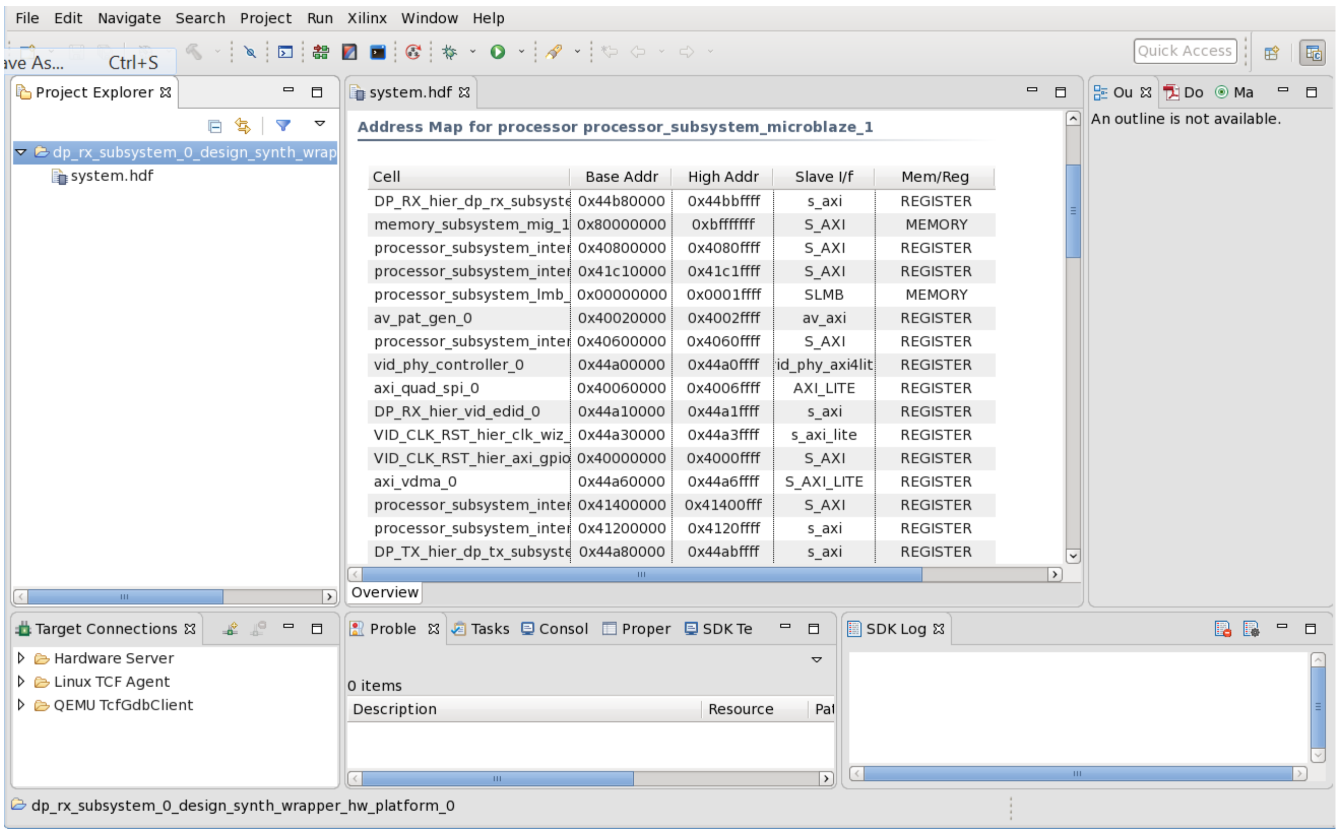
Launch SDK <@xsjrdevl131>

Launch software development tool.

Exported location: <Local to Project>

Workspace: <Local to Project>

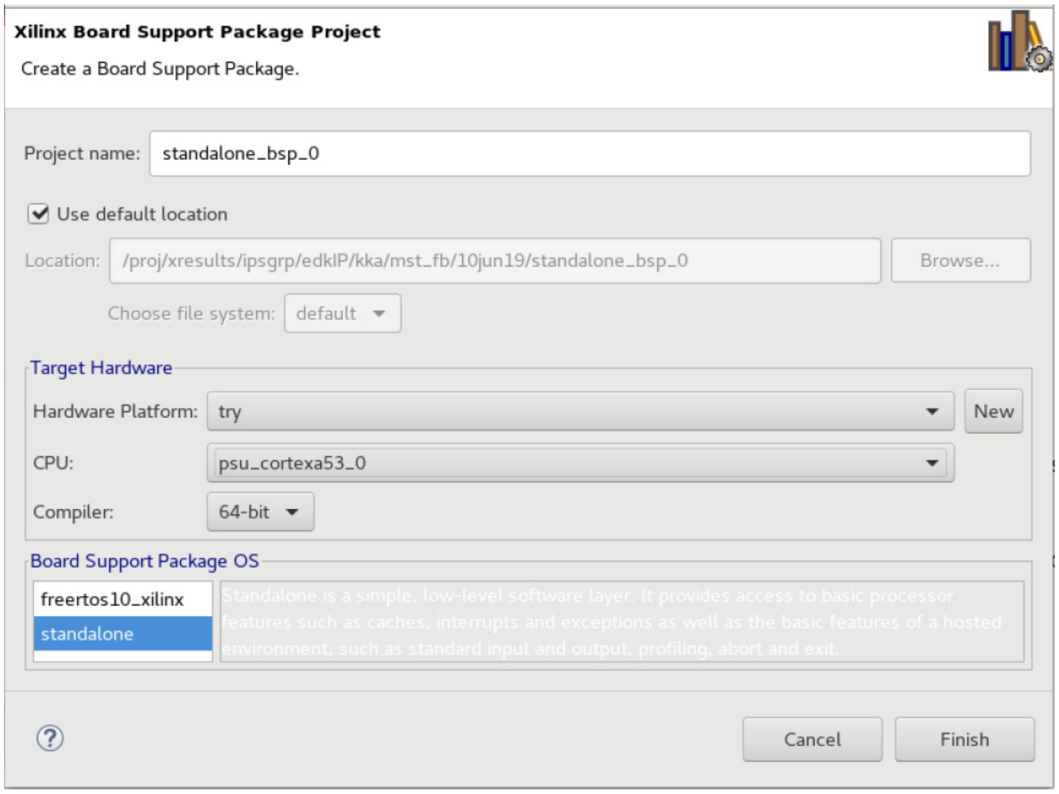
14. The following figure shows an example of the launched Vivado SDK.


File Edit Navigate Search Project Run Xilinx Window Help
Quick Access
Project Explorer
dp_rx_subsystem_0_design_synth_wrap
system.hdf
Ou Do Ma
An outline is not available.
Overview
Target Connections
Hardware Server
Linux TCF Agent
QEMU TcfGdbClient
Proble Tasks Consol Proper SDK Te
0 items
Description Resource Pat
SDK Log
dp_rx_subsystem_0_design_synth_wrapper_hw_platform_0

**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 |

15. To create a Board Support Package (BSP), click **File > New > Board Support Package**. Enter the BSP **Project name**, click **Finish**, and then **OK**. For ZCU102 board, ensure the target CPU is the Cortex A53 (`psu_cortexa53_0`).



**Xilinx Board Support Package Project**

Create a Board Support Package.

Project name: standalone_bsp_0

☑ Use default location

Location: /proj/xresults/ipsgrp/edkIP/kka/mst_fb/10jun19/standalone_bsp_0

Choose file system: default

Target Hardware

Hardware Platform: try

CPU: psu_cortexa53_0

Compiler: 64-bit

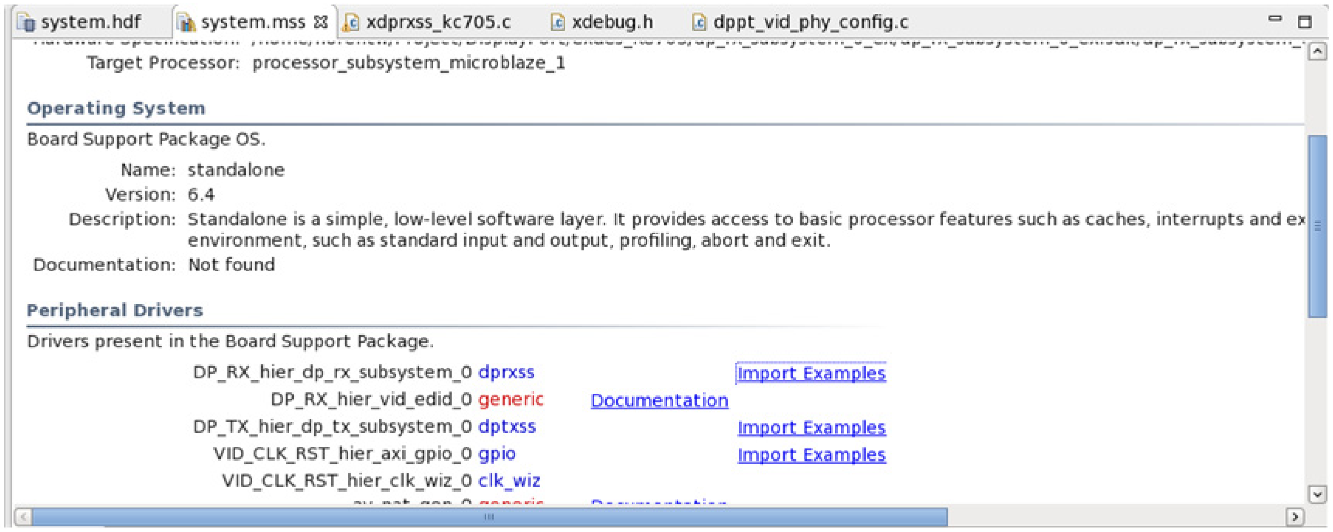
Board Support Package OS

freertos10_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.

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



Target Processor: processor_subsystem_microblaze_1

Operating System

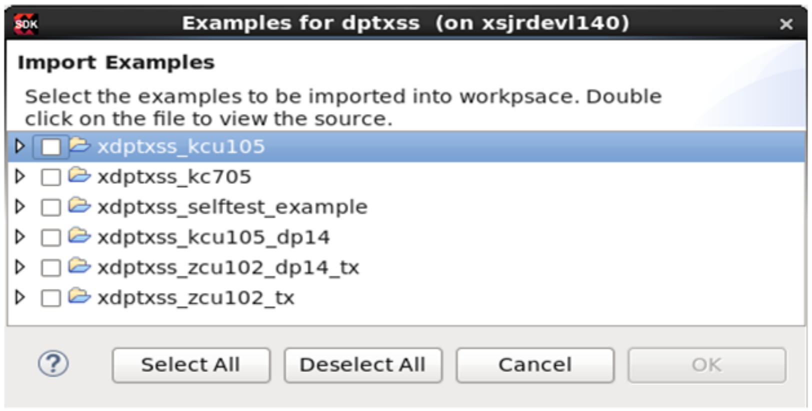
Board Support Package OS.

Name: standalone
Version: 6.4
Description: Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and ex
environment, such as standard input and output, profiling, abort and exit.
Documentation: Not found

Peripheral Drivers

Drivers present in the Board Support Package.

| | | |
|---|---|---|
| DP_RX_hier_dp_rx_subsystem_0 dprxss | | Import Examples |
| DP_RX_hier_vid_edid_0 generic | Documentation | |
| DP_TX_hier_dp_tx_subsystem_0 dptxss | | Import Examples |
| VID_CLK_RST_hier_axi_gpio_0 gpio | | Import Examples |
| VID_CLK_RST_hier_clk_wiz_0 clk_wiz | | |



Import Examples

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

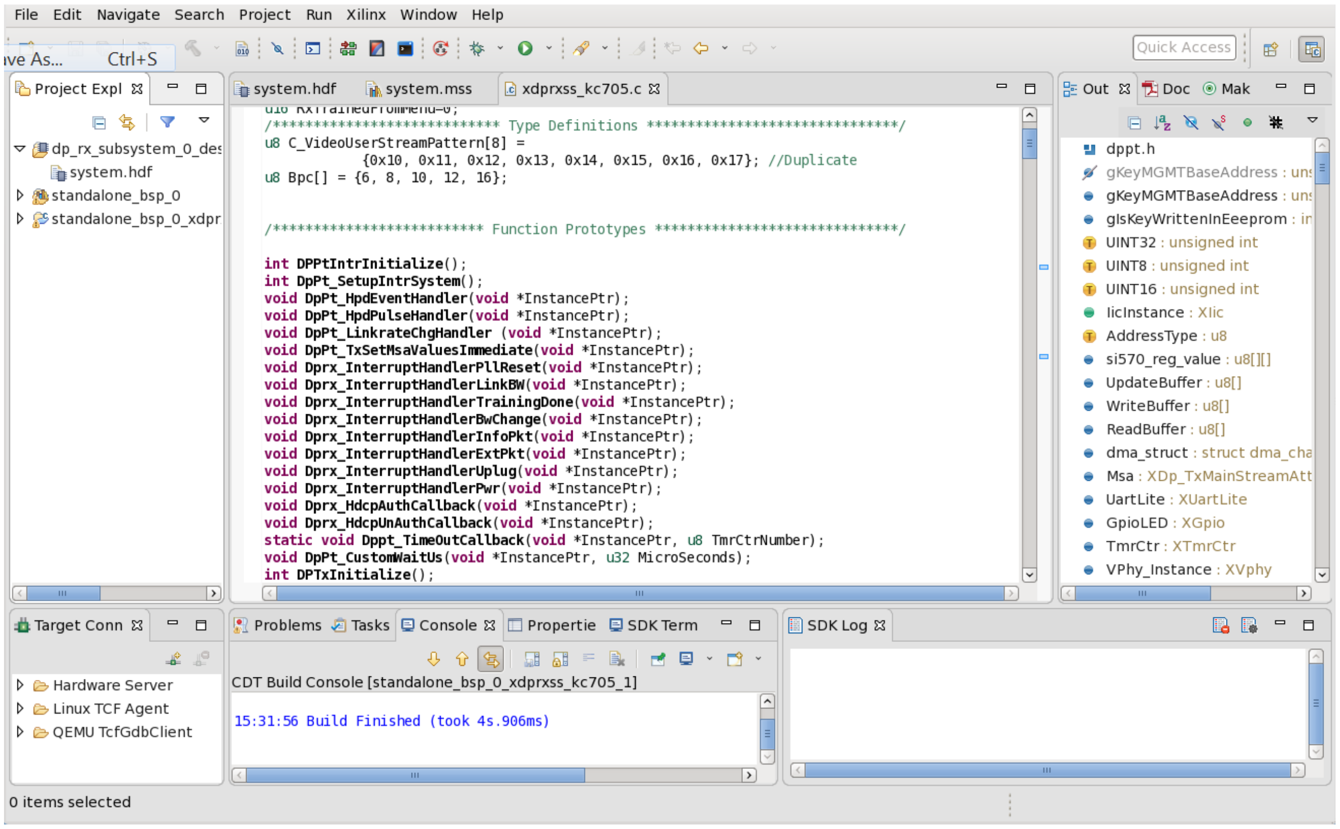
- xdptxss_kcu105
- xdptxss_kc705
- xdptxss_selftest_example
- xdptxss_kcu105_dp14
- xdptxss_zcu102_dp14_tx
- xdptxss_zcu102_tx

17. Select the Example Application corresponding to your hardware:
    - For Pass-through KC705 project, select `*_kc705` option.

***Note:*** In the KC705, for HDCP Pass-through application, select the HDCP option in the Subsystem IP configuration screen before opening the Example Design.

- For Pass-through KCU105 project, select `*_kcu105` option.
- For RX only ZCU102 project, select `*_zcu102_rxonly` option in RX Subsystem Driver.
- For TX only ZCU102 project, select `*_zcu102_txonly` option in TX Subsystem Driver.

18. shows the example application successfully built and ready to use.



```
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 ******************************/

int DPPtIntrInitialize();
int DpPt_SetupIntrSystem();
void DpPt_HpdEventHandler(void *InstancePtr);
void DpPt_HpdPulseHandler(void *InstancePtr);
void DpPt_LinkrateChgHandler (void *InstancePtr);
void DpPt_TxSetMsaValuesImmediate(void *InstancePtr);
void Dprx_InterruptHandlerPllReset(void *InstancePtr);
void Dprx_InterruptHandlerLinkBW(void *InstancePtr);
void Dprx_InterruptHandlerTrainingDone(void *InstancePtr);
void Dprx_InterruptHandlerBwChange(void *InstancePtr);
void Dprx_InterruptHandlerInfoPkt(void *InstancePtr);
void Dprx_InterruptHandlerExtPkt(void *InstancePtr);
void Dprx_InterruptHandlerUplug(void *InstancePtr);
void Dprx_InterruptHandlerPwr(void *InstancePtr);
void Dprx_HdcpAuthCallback(void *InstancePtr);
void Dprx_HdcpUnAuthCallback(void *InstancePtr);
static void Dppt_TimeOutCallback(void *InstancePtr, u8 TmrCtrNumber);
void DpPt_CustomWaitUs(void *InstancePtr, u32 MicroSeconds);
int DPTxInitialize();
```

CDT Build Console [standalone_bsp_0_xdprxss_kc705_1]

```
15:31:56 Build Finished (took 4s.906ms)
```

## Hardware Setup and Run

1. Connect the Tokyo Electron Device Limited (TED) TB-FMCH-DP3 module to the HPC FMC connector on the KC705 (or 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 KC705 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 5-1.
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 5-1.

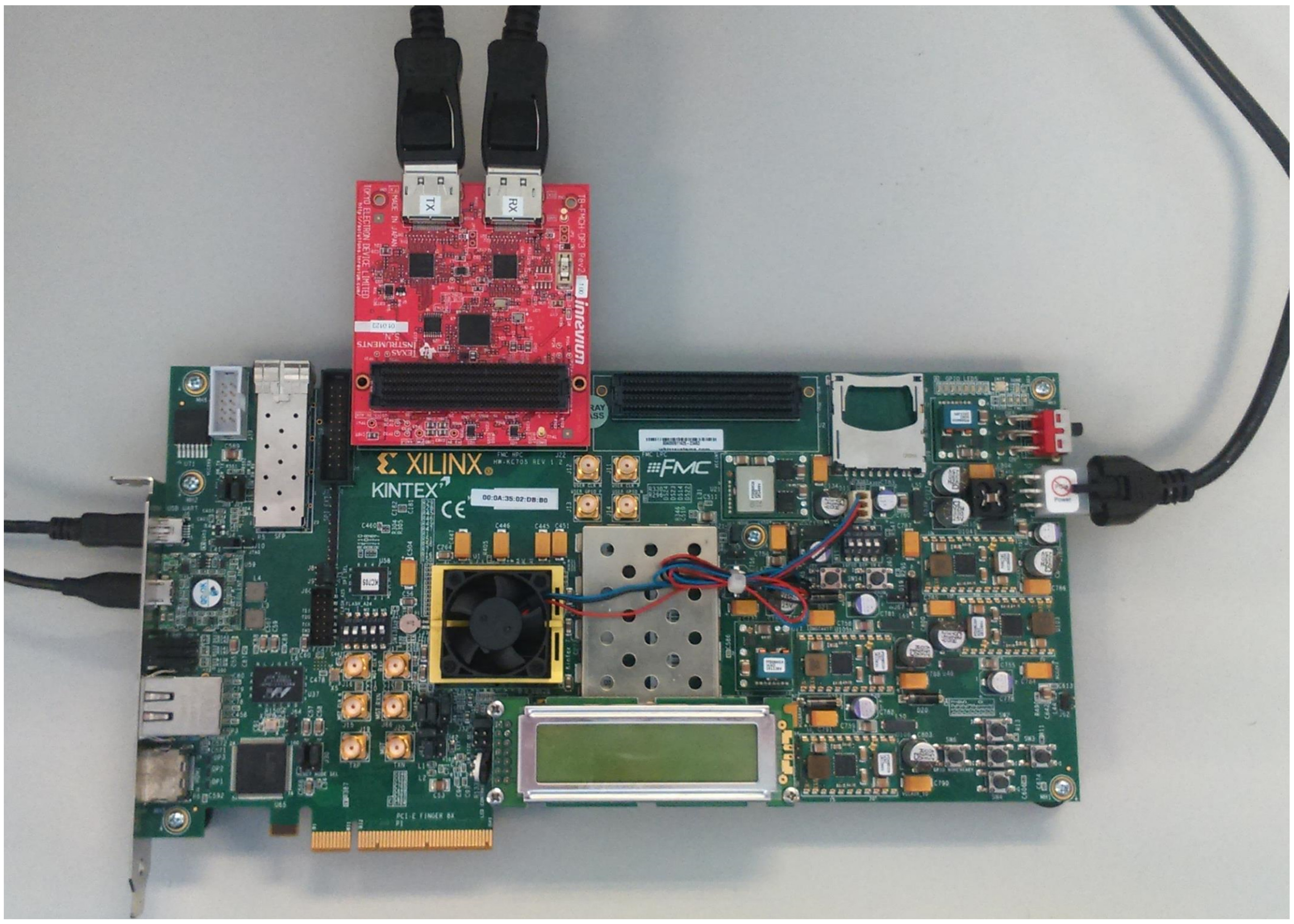


*Figure 5-1:* **KC705 Board Setup**

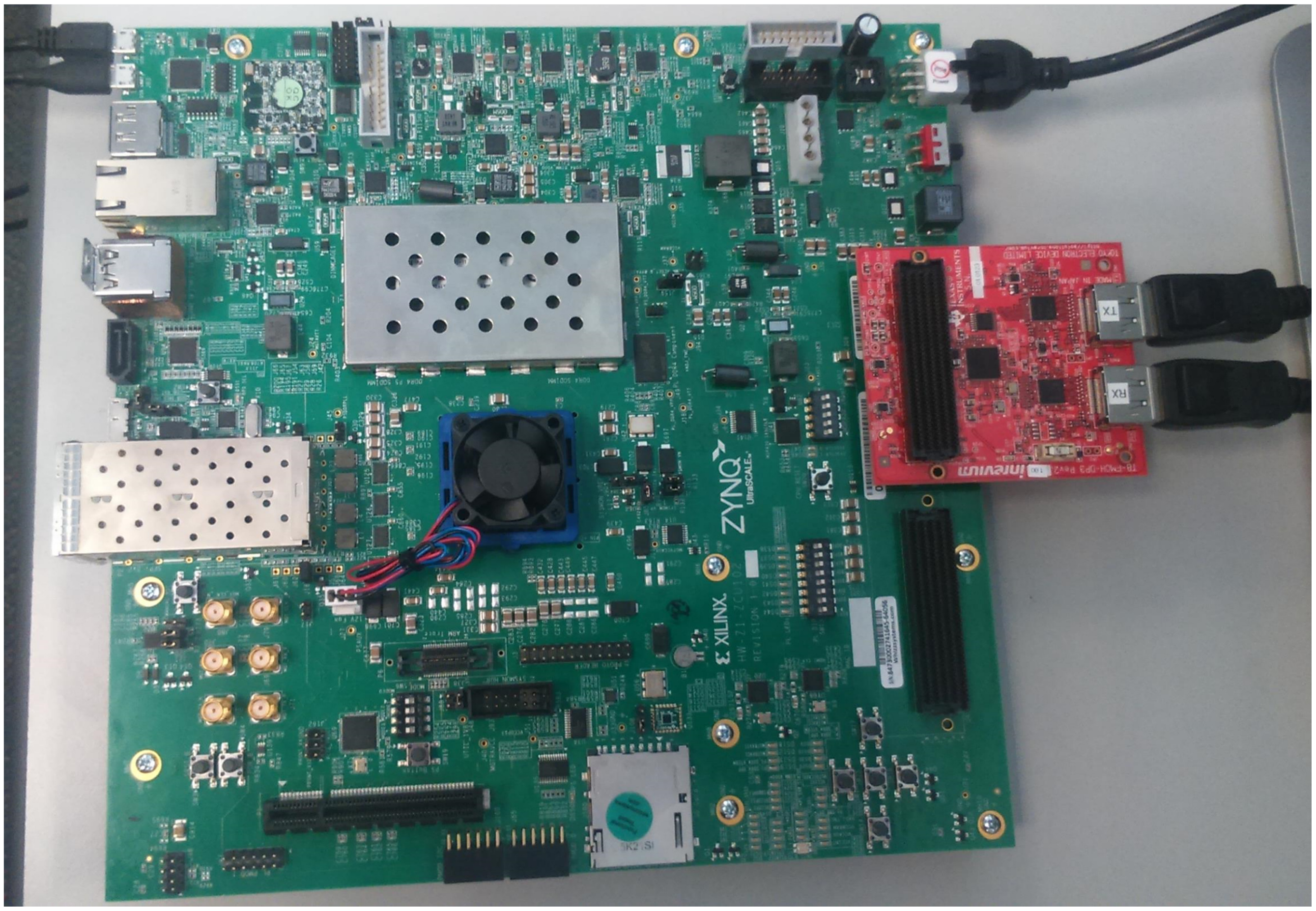
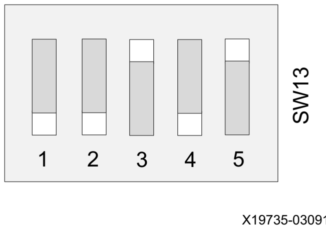
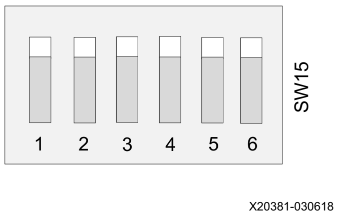


*Figure 5-2:* **ZCU102 Board Setup**

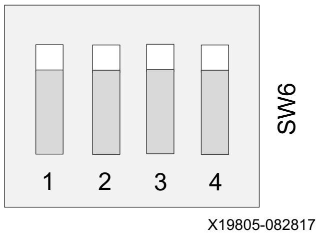
6. Set the mode pin to JTAG:



Set the mode pin to SW15:



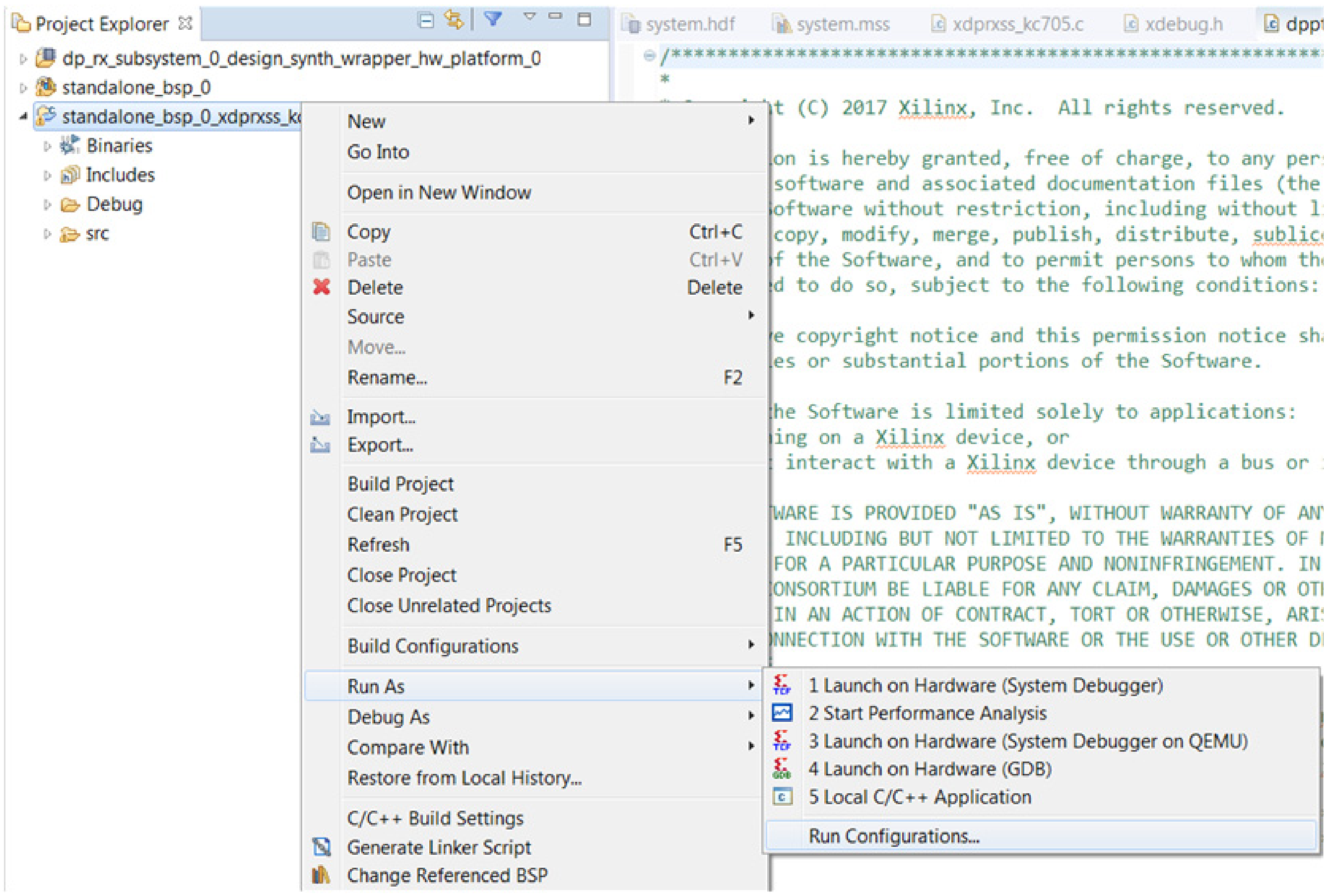
Set the mode pin to SW6:



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

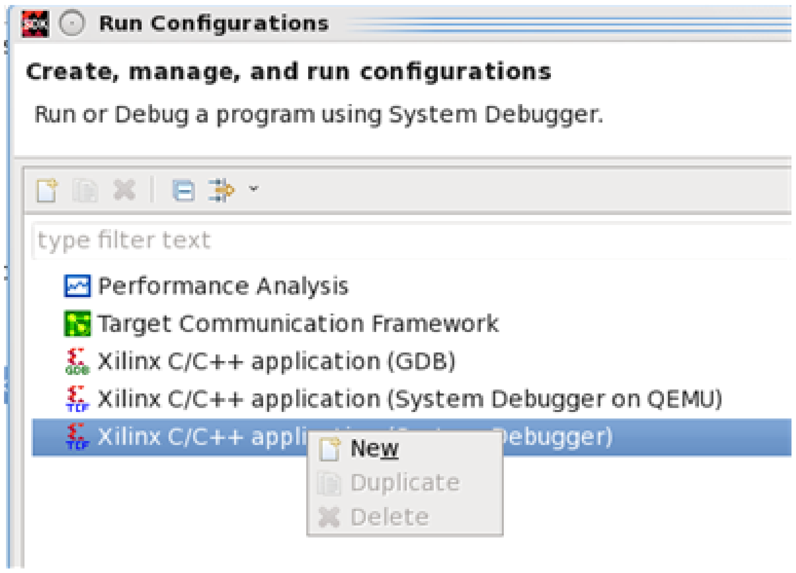
   ***Note:*** With the ZCU102 board, there are four COM ports available.

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



```
/******************************************************************
 *
ht (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
```

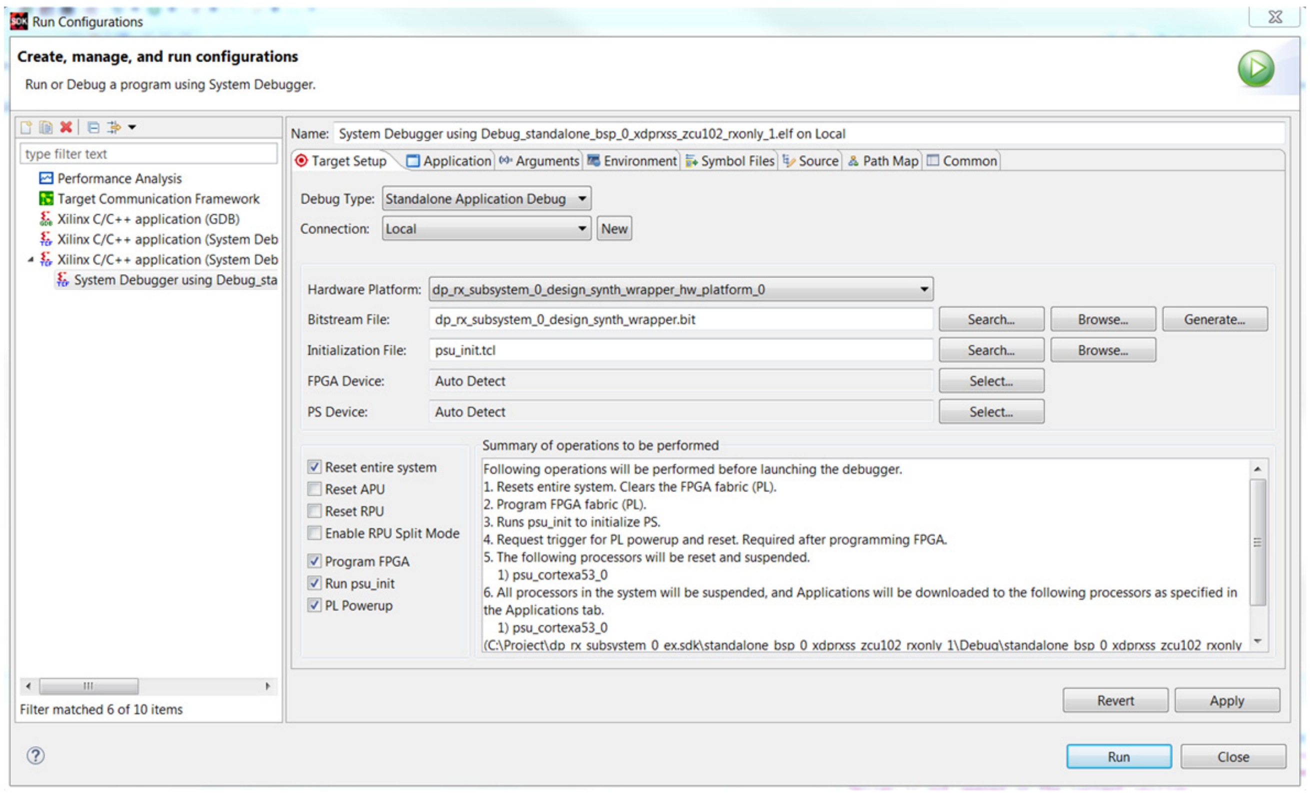
10. In the **Run Configurations** popup menu, right-click **Xilinx C/C++ application (System Debugger)** and click **New**.


Run Configurations
type filter text
Performance Analysis
Target Communication Framework
Xilinx C/C++ application (GDB)
Xilinx C/C++ application (System Debugger on QEMU)
Xilinx C/C++ appli ebugger)
New
Duplicate
Delete

Create, manage, and run configurations

Run or Debug a program using System Debugger.

11. In the **Target Setup** tab, 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

Name: System Debugger using Debug_standalone_bsp_0_xdprxss_zcu102_rxonly_1.elf on Local

Target Setup | Application | Arguments | Environment | Symbol Files | Source | Path Map | Common

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

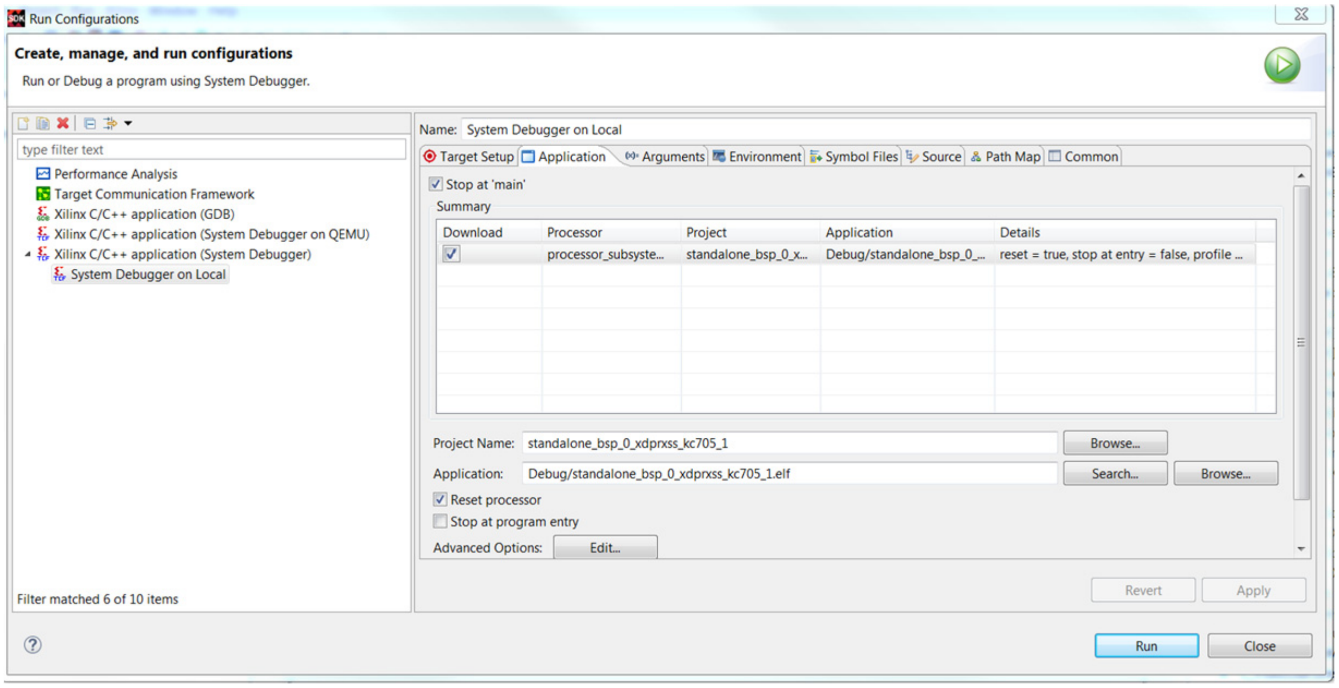
PS Device: Auto Detect

- ☑ 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
```

12. In the **Application** tab, ensure the application download is enabled and click **Run**.



Run Configurations

Create, manage, and run configurations

Run or Debug a program using System Debugger.

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

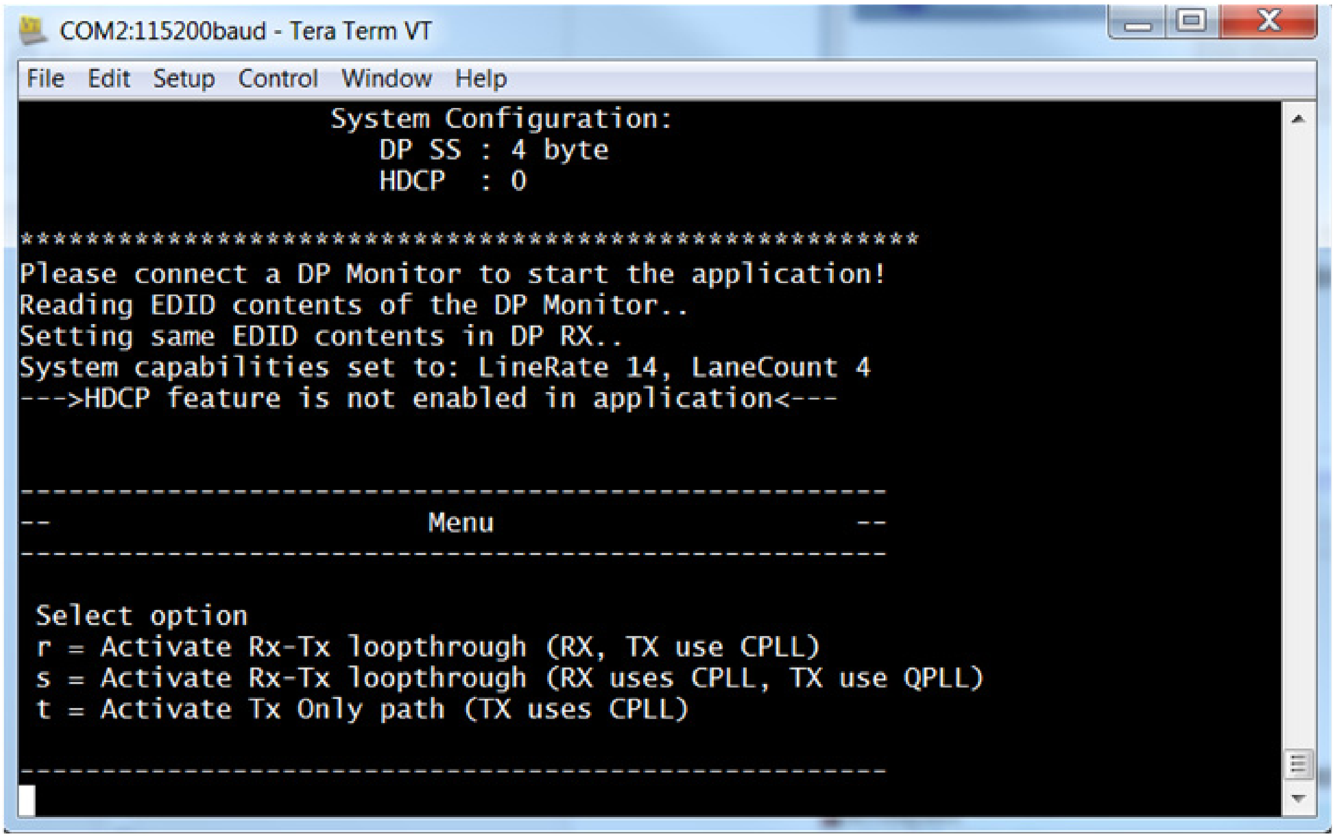
☐ Stop at program entry

Advanced Options:

## Display User Console

### Pass-Through Application (KC705 and 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 5-3 to choose from (KC705).


COM2:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
                   System Configuration:
                      DP SS : 4 byte
                      HDCP  : 0

*******************************************************
Please connect a DP Monitor to start the application!
Reading EDID contents of the DP Monitor..
Setting same EDID contents in DP RX..
System capabilities set to: LineRate 14, LaneCount 4
--->HDCP feature is not enabled in application<---


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

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

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

*Figure 5-3:* **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 5-3 shows the UART transcript. The transcript might differ based on the training done by GPU.

## HDCP Support and Operation

On KC705, an application example design with HDCP support is available. HDCP is supported by the DisplayPort TX Subsystem (SST mode only).

***Note:*** To have a working HDCP solution it is necessary to have valid HDCP keys and a monitor that supports HDCP. Xilinx does not provide any HDCP keys. See Configuring HDCP Keys and Key Management to set up and manage HDCP Keys.

The application detects the HDCP capabilities of the monitor that is connected to DisplayPort TX Subsystem. HDCP feature is enabled only if the monitor is capable of displaying HDCP content.

In Pass-Through mode (options `r` or `s`), the DisplayPort source (for example, GPU) detects the HDCP capability of the DisplayPort Sink and starts the authenticate process. When the Authentication is successful, the GPU can start playing the encrypted HDCP content. The DP Sink, automatically detects this and starts decrypting the encrypted content. The decrypted content has to be encrypted again before being displayed on the monitor connected to the DisplayPort TX Subsystem. This has to be done to ensure that the unencrypted content is not displayed.

The HDCP authentication process with the DisplayPort Monitor starts when application detects encrypted content on the RX. After the authentication is completed, the DisplayPort TX encrypts the video content before sending it to the monitor. The application tries 100 attempts to authenticate. If it is not able to authenticate within 100 attempts, then it switches the TX video to color bar pattern that is, the unencrypted HDCP content is not displayed.

In TX only mode (selection `t`), you have to manually initiate the HDCP authentication and encryption.

If the monitor does not support HDCP, the HDCP functionality is disabled in the application.

## Configuring HDCP Keys and Key Management

The application software does not use the raw HDCP keys directly. To use the HDCP keys, they have to be first encrypted and then added into the application. You have to manually perform this process. This section provides the scripts and software that helps you encrypt the HDCP keys.

## Using the Encryption Software

For more information on using the encryption software, see AR: 70605. Before you start, download and extract the zip files. To generate the AES encrypted HDCP keys, you must have the following keys:

1. 32-byte AES key
2. Valid HDCP keys

***Note:*** Xilinx does not provide any of the above keys. The application delivered with this software is created with invalid keys and hence does not play HDCP content.

Follow the steps mentioned here to generate the AES encrypted HDCP key block:

1. Unzip the project and go to the `hdcp_util/keys` directory. You can find the encryption block in this directory.
2. Modify the `gDefaultKey` array in the following file to a user specified 32 bytes unique key. This is the 32-byte AES key mentioned in point (1) above

   ```
   hdcp_util/keys/key-encryptor/common/src/keyfile.c
   ```
3. Navigate to `hdcp_util/keys/key-encryptor/build/linux` folder and execute the following command from linux terminal.

   ```
   ./build.sh
   ```

   This creates `hdcp-enc.bin` file in the same folder.
4. From the same directory, execute the following command to create the AES encrypted file `keymgmt_data.c`.

   ```
   ./hdcp-enc.bin -c devb1_keys.dat devb2_keys.dat …. devb<n>_keys.dat
   ```

   ***Note:*** The user-provided `devb<n>_keys.dat` file is expected to have one and only one original HDCP key. An original HDCP key has one 5 byte Key Selection Vector and 40 private keys. If you have multiple HDCP keys, each key should be housed exclusively in one `.dat` file.
5. The AES encrypted HDCP key block is now created as an array in the `keymgmt_data.c` file.
6. Ensure that the following AES keys in the reference design matches with the keys in step 2.

   ```
   \**kc705_system_directly**\**sw_directly**\src\keys.c
   ```

   The HDCP keys are now encrypted and ready to be used. You can any of the following two ways to use the HDCP keys.

## Using the AES Encryption Key Block from the EEPROM

You can use the `hdcp_util/iic_wr_util/kc705` project provided with the IP to write the keys to the EEPROM. You need to copy the AES encrypted block generated in the `keymgmt_data.c` file as described in the previous section to the HDCP_KEYS array in the `hdcp_util/iic_wr_util/kc705/sw/src/iic_keys.c` file.

***Note:*** The size of the keys is calculated and stored in the HDCP_KEYS_SZ variable. Ensure that the size of the keys fits in the EEPROM.

Navigate to the `hdcp_util/iic_wr_util/kc705/sw/` folder. Use the Vivado build in the command prompt and run the following command:

```
xsct ./all.tcl
```

This creates the `iic_eeprom.elf` file in `hdcp_util/iic_wr_util/kc705/sw/iic_eeprom/Debug` folder.

You can then use this `elf` file and the `design_1_wrapper.bit` file provided in `hdcp_util/iic_wr_util/kc705/ready_to_download` folder to program the EEPROM on the board with the AES encrypted keys.

Set `gIsKeyWrittenInEeeprom = TRUE` in `\**kc705_system_directly**\**software_directly**\src\xdp*xss_kc705.c` file. Ensure that the IIC device ID is correctly set in `keygen_config.h` file.

## Using the AES Encryption Key Block from the Block RAM

To use keys from block RAM, you have to copy the AES Encrypted HDCP keys generated in the previous section to the `KEYMGMT_ENCDATA` array defined in the following file in the reference design:

```
\**kc705_system_directly**\software_directly\src\keys.c
```

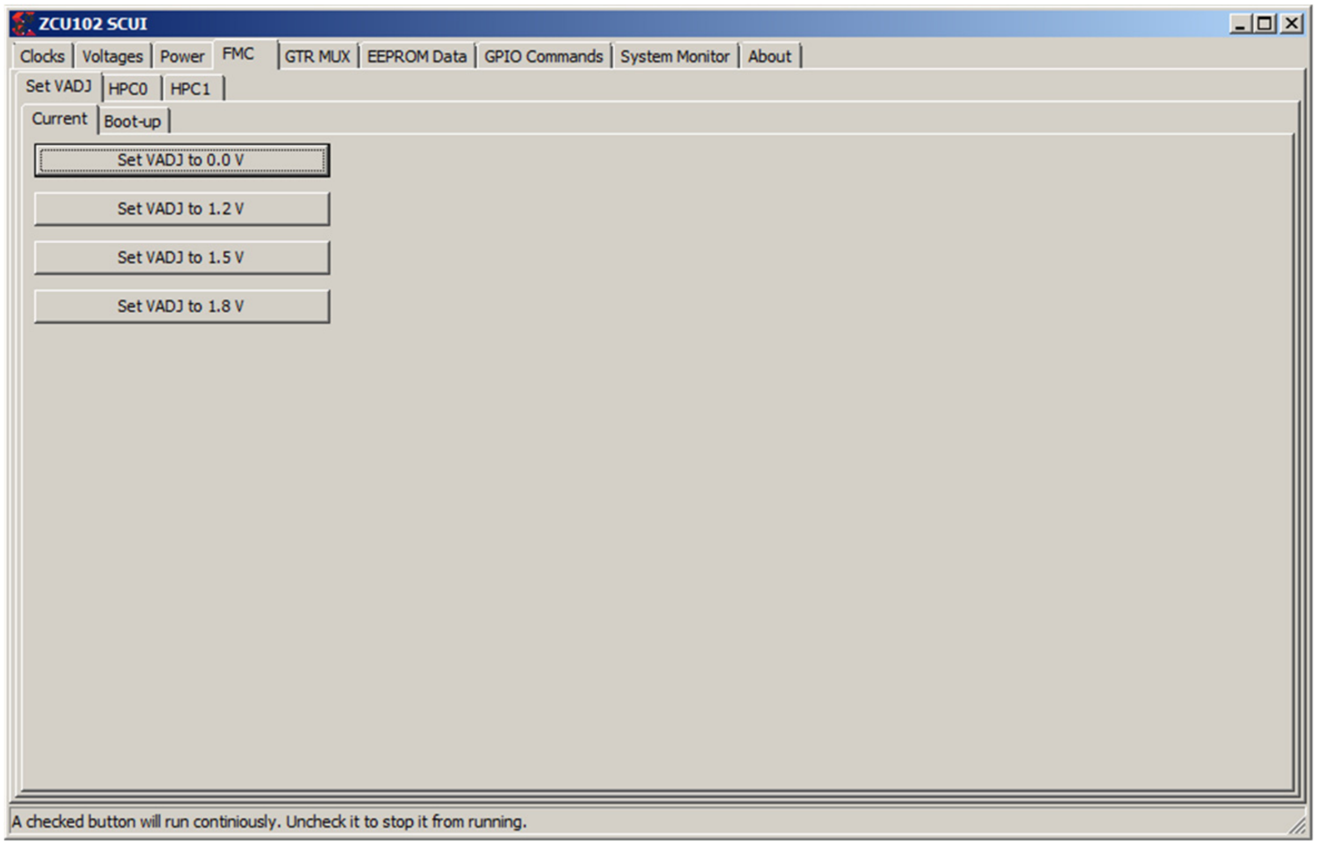
This process enables the reference design to read keys from block RAM.

Set `gIsKeyWrittenInEeeprom = FALSE` in `\**kc705_system_directly**\**software_directly**\src\xdp*xss_kc705.c` file.

## Setting the FMC Voltage to 1.8 V

To run the example design on the ZCU102 board, ensure that the FMC voltage is set to 1.8 V. 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 1.8 V**.



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

## Test Equipment

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

*Table 5-2:* **Sink Equipment**

| Sink Type | Brand Name | Model Name |
|---|---|---|
| Monitor | Acer | S277HKWMIDPP |
| Monitor | Dell | S2817Q |
| Monitor | Dell | P2414H |
| Monitor | Dell MST | P2415Q |
| Monitor | LG | 27UD68P |
| Monitor | ASUS | PB279 |
| Monitor | Philips | 288P6LJ |
| Monitor | Dell MST | U2413 |
| Tester | Unigraf | UCD-323 |
| Tester | Unigraf | DPR-120 |
| Tester | Unigraf | DPR-100 |

# Upgrading

When migrating from the LogiCORE DisplayPort IP to the DisplayPort TX Subsystem with the Video PHY Controller (*Video PHY Controller Product Guide* (PG230) [Ref 2]), Xilinx recommends removing the LogiCORE DisplayPort IP in its entirety and then implementing the DisplayPort TX Subsystem. The following notes will help with the migration:

- Use the example design as a reference to ensure that all connections are correct.
- The LogiCORE DisplayPort IP integrates the transceivers whereas the transceivers reside in the Video PHY Controller of the subsystem implementation.
- The DisplayPort Subsystem has the option to have a native pixel or an AXI4-Stream interface.
- All associated signals that were part of the transceivers, reference clocks and transceiver lanes, are now part of the Video PHY Controller.
- The parameters for the number of DisplayPort lanes and the PHY data width need to match between the DisplayPort TX Subsystem and the Video PHY Controller.
- The link clock for the DisplayPort TX Subsystem is generated by the Video PHY Controller.
- The `lnk_clk_[p/n]` of the LogiCORE DisplayPort IP should be connected to the `mgtrefclk0_pad_[p/n]_in` of the Video PHY Controller.
- MMCM programming has been moved to software (if the DisplayPort TX Subsystem is configured with Buffer Bypass enabled). The recommendation is to use the example application as a reference for the MMCM programming.

# 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.

*Appendix D*

# 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 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 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 Subsystem**

AR: 59384

### 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 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 8].

## 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 DisplayPort1.2a compliant.
- Make sure you are using proper DisplayPort1.2a certified cable which is tested to run at 5.4 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 v1.2a 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.
- Make sure the transceiver lane LOC pin assignment is correct and no lane swapping is present.

## 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, use void `XDpTxSs_VtcAdjustBSTimingEnable(XDpTxSs *InstancePtr)`. This moves the BS symbol into the front porch to fix a swing in the BS timing caused by a non-standard CVT timing.

## Software Debug

This section shows how to navigate to the DisplayPort debug driver information.

1. Click **bsp** in the SDK and right-click to select **Board Support Package Settings** (Figure D-1).

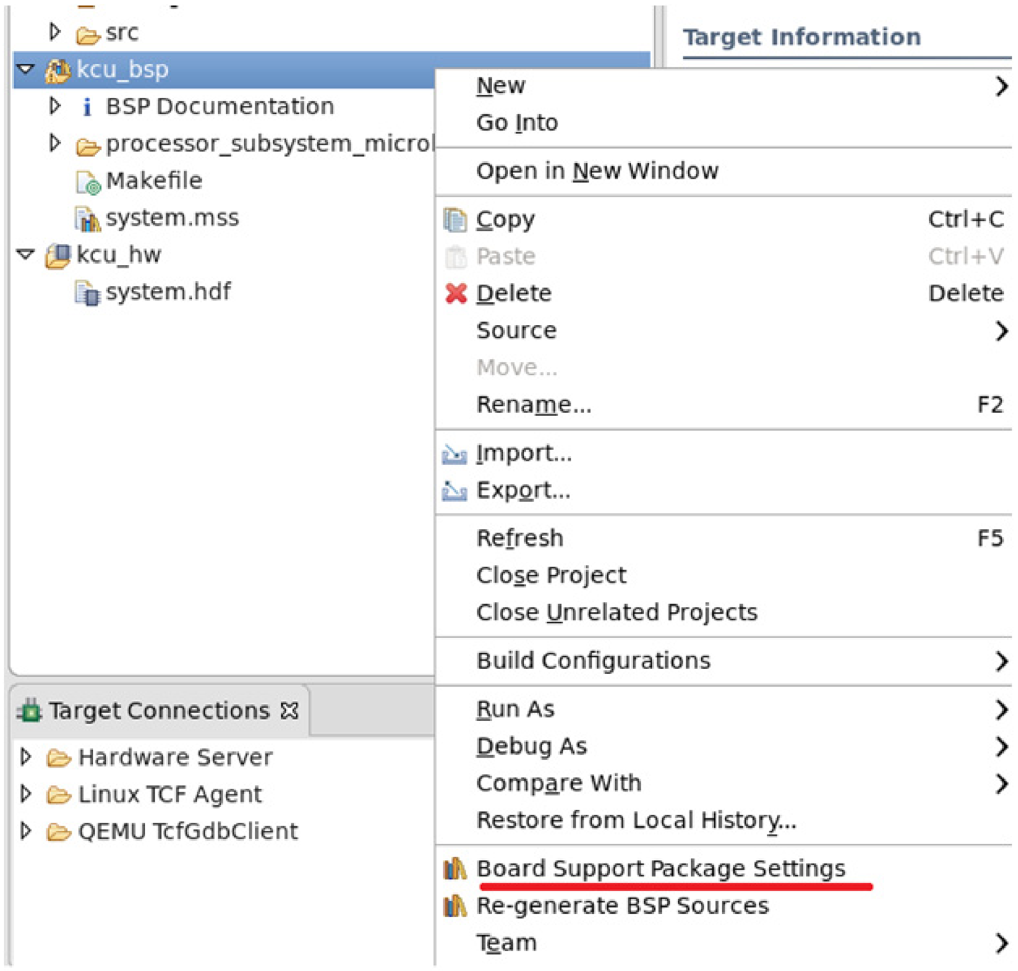
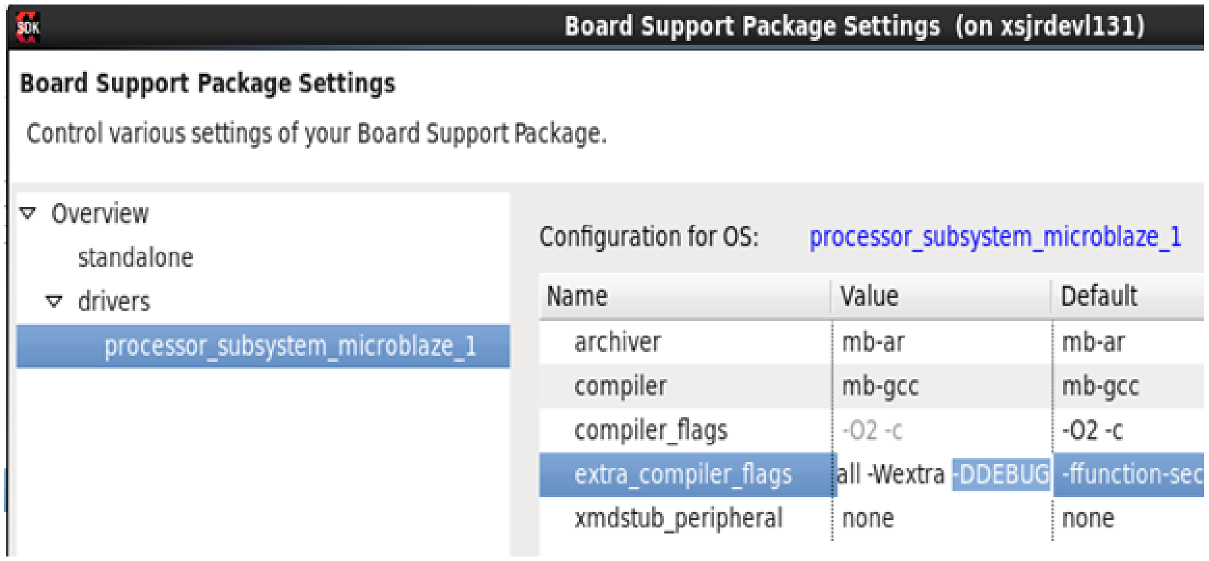


*Figure D-1:* **Path to Board Support Package Settings**

2. The **Board Support Package Settings** window pops up. Navigate to the **processor_subsystem_microblaze_*** (Figure D-2).

3. Select the **extra_compiler_flags** in the Name column. Add the **-DDEBUG** flag in the Value column. You can view all of the debug information from the DisplayPort driver as well as DisplayPort subsystem driver debug information.



Board Support Package Settings (on xsjrdevl131)

**Board Support Package Settings**

Control various settings of your Board Support Package.

Configuration for OS: processor_subsystem_microblaze_1

| Name | Value | Default |
|---|---|---|
| archiver | mb-ar | mb-ar |
| compiler | mb-gcc | mb-gcc |
| compiler_flags | -O2 -c | -O2 -c |
| extra_compiler_flags | all -Wextra -DDEBUG | -ffunction-sec |
| xmdstub_peripheral | none | none |

*Figure D-2:* **Board Support Package Settings Window**

# Application Software Development

The software is capable of detecting an MST/SST RX connected to the subsystem based on if a MST or SST software flow is executed. Figure E-1 shows the DisplayPort TX Subsystem application software flow.

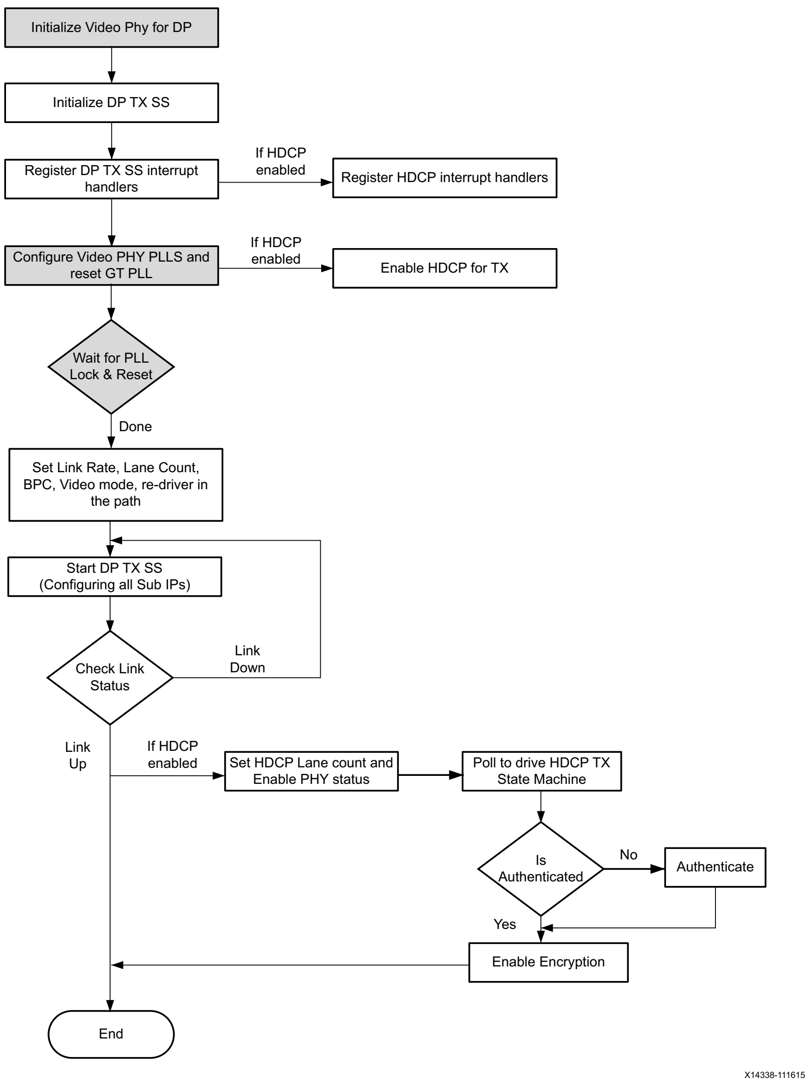

Initialize Video Phy for DP
Initialize DP TX SS
Register DP TX SS interrupt handlers
If HDCP enabled
Register HDCP interrupt handlers
Configure Video PHY PLLS and reset GT PLL
If HDCP enabled
Enable HDCP for TX
Wait for PLL Lock & Reset
Done
Set Link Rate, Lane Count, BPC, Video mode, re-driver in the path
Start DP TX SS (Configuring all Sub IPs)
Check Link Status
Link Down
Link Up
If HDCP enabled
Set HDCP Lane count and Enable PHY status
Poll to drive HDCP TX State Machine
Is Authenticated
No
Authenticate
Yes
Enable Encryption
End
X14338-111615

*Figure E-1:* **Software Flow**

***Note:*** The Video PHY is external to the DisplayPort TX Subsystem and must be configured for the subsystem to work as expected. For more details on the Video PHY configuration, see the *Video PHY Product Guide* (PG230) [Ref 2].

# 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. VESA *DisplayPort Standard v1.2a*, December 22, 2009
2. *Video PHY Controller Product Guide* (PG230)
3. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
4. *Vivado Design Suite User Guide: Designing with IP* (UG896)
5. *Vivado Design Suite User Guide: Getting Started* (UG910)
6. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
7. *ISE to Vivado Design Suite Migration Guide* (UG911)
8. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
9. *Vivado Design Suite User Guide: Implementation* (UG904)
10. *AXI Reference Guide* (UG1037)
11. *AXI4-Stream to Video Out LogiCORE IP Product Guide* (PG044)
12. *Video Timing Controller LogiCORE IP Product Guide* (PG016)
13. *HDCP Controller Product Guide* (PG224)
14. *AXI Timer Product Guide* (PG079)
15. *AXI4-Stream Video IP and System Design Guide* (UG934)
16. *ZCU102 System Controller GUI Tutorial* (XTP433) (registration required)

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 06/17/2019 | 2.1 | Minor updates |
| 12/05/2018 | 2.1 | • Color Component Mapping table, Table 2-3, added.<br>• Odd/Even description updated, Table 2-8 and Table 2-9.<br>• DisplayPort Audio TX_AUDIO_CONTROL updated, Table 2-26.<br>• Updated Transmit – Misaligned Data section. |
| 04/04/2018 | 2.1 | • Added Native Video support and dynamic lane supports in IP facts table.<br>• Updated Unsupported Features section in Overview chapter.<br>• Updated Overview section in Product Specification chapter.<br>• Added AXI4-Stream Interface Data Mapping table in Product Specification chapter.<br>• Updated Pixel Mapping Examples on AXI4-Stream Interface table.<br>• Updated Pixel Mapping on AXI4-Stream Interface section.<br>• Added Audio Streaming Signals section.<br>• Updated DisplayPort TX Subsystem Ports table.<br>• Updated Clock section and Address Map Example table in Designing with the Core chapter.<br>• Updated Customizing the IP section in Design Flow Steps chapter.<br>• Updated Constraining the Core section in Design Flow Steps chapter.<br>• Added available example designs in Example Design chapter.<br>• Added JTAG mode pin position KCU105 figure in Example Design chapter.<br>• Added HDCP Support and Operation and Configuring HDCP Keys and Key Management sections in the Example Design chapter.<br>• Added MMCM programming in Upgrading appendix.<br>• Added Transmit – Misaligned Data and Software Debug sections in Debugging appendix. |
| 12/20/2017 | 2.1 | • Added Vivado IP Integrator to IP Facts table.<br>• Updated Setting the FMC Voltage to 1.8V section. |
| 10/04/2017 | 2.1 | • Updated Supported Device Family in IP Facts.<br>• Added Pixel Mapping for Stream Interface table and Pixel Mapping to Native Interface section in Product Specification chapter.<br>• Added Example Design chapter.<br>• Added Upgrading appendix.<br>• Added FAQ appendix.<br>• Added new Driver Documentation appendix. |
| 07/14/2017 | 2.0 | Updated 7 series (GTHE2) and Artix-7 support in IP Facts. |
| 06/07/2017 | 2.0 | Vivado Design Suite release for DisplayPort TX v2.0. |

| Date | Version | Revision |
|---|---|---|
| 04/05/2017 | 2.0 | • Updated Supported Device Family in IP Facts table.<br>• Added In-band bullet in Unsupported Features section.<br>• Added DisplayPort Registers Source Core in Product Specification chapter.<br>• Added Source Overview in Designing with the Core chapter.<br>• Added Reduced Blanking in Designing with the Core chapter.<br>• Updated Hardware Debug section in Debug Appendix. |
| 12/20/2016 | 2.0 | Added HDCP note for Supported Device Family in IP Facts table. |
| 11/30/2016 | 2.0 | Added Important note in Standards section. |
| 10/05/2016 | 2.0 | Updated HDCP features. |
| 04/06/2016 | 2.0 | Added support for 16 bit GT interface and native with pixel mode. |
| 11/18/2015 | 1.0 | Initial Xilinx release. |

## Please Read: Important Legal Notices