DisplayPort IP Core User Guide

Last updated for Altera Complete Design Suite: 14.0
UG-01131
2014.06.30

Subscribe
Send Feedback

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Contents

DisplayPort IP Core Quick Reference

1

2014.06.30

UG-01131 Subscribe Send Feedback

This document describes the Altera® DisplayPort MegaCore® function, which provides support for next-generation video display interface technology.

The DisplayPort IP core is part of the MegaCore IP Library, which is distributed with the Quartus® II software and is downloadable from the Altera website at **www.altera.com**.

Note: For system requirements and installation instructions, refer to the *Altera Software Installation and Licensing Manual.*

Table 1-1: DisplayPort IP core

Item		Description
Release Information	Version	14.0
	Release Date	June 2014
	Ordering Code	IP-DP-v1.1a
	Product ID	0109
	Vendor ID	6AF7

Item		Description

IP Core Information	Core Features	• Conforms to the Video Electronics Standards Association (VESA) specification version 1.2a
		• Scalable main data link
		  • 1, 2, or 4 lane operation
		  • 1.62, 2.7, and 5.4 Gbps per lane with an embedded clock
		• Color support
		  • 16, 18, 20, 24, 30, 32, 36, or 48 bits per pixel (bpp) color depths
		  • RGB and YCrCb color modes
		• 40-bit (quad symbol) and 20-bit (dual symbol) transceiver data interface
		• Support for 1, 2, or 4 parallel pixels per clock
		• Source
		  • Embedded controller AUX channel operation
		  • Accepts standard H-sync/V-sync/data enable RGB and YCrCb input video formats
		  • Supports audio and video streams
		• Sink
		  • Finite state machine (FSM) and embedded controller AUX channel operation
		  • Produces a proprietary video output
		• Produces a standard Avalon® Streaming (Avalon-ST) interface for use in the Altera VIP Suite
		• Auxiliary channel for 2-way communication (link and device management)
		• Hot plug detect (HPD)
		  • Sink announces its presence
		  • Sink requests the source's attention
		• AC coupling and low EMI

	Typical Application	• Interfaces within a PC or monitor • External display connections, including interfaces between a PC and monitor or projector, between a PC and TV, or between a device such as a DVD player and TV display
	Device Family Support	Arria® V GX, Arria V GZ, Cyclone® V, and Stratix® V FPGA devices. Refer to the *What's New in Altera IP* page of the Altera website for detailed information.
	Design Tools	

Item		Description
		• IP Catalog in the Quartus II software for IP design instantiation and compilation • TimeQuest timing analyzer in the Quartus II software for timing analysis • ModelSim-Altera software for design simulation

Related Information

- **Altera Software Installation and Licensing**
- **What's New in Altera IP**

About This IP Core 2

2014.06.30

UG-01131 Subscribe Send Feedback

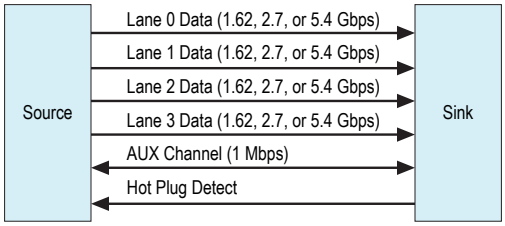
This document describes the Altera® DisplayPort MegaCore® function, which provides support for next-generation video display interface technology. The Video Electronics Standards Association (VESA) defines the DisplayPort standard as an open digital communications interface for use in internal connections such as:

- Interfaces within a PC or monitor
- External display connections, including interfaces between a PC and monitor or projector, between a PC and TV, or between a device such as a DVD player and TV display

The Altera DisplayPort source has a scalable main link with 1, 2, or 4 lanes for a total up to 21.6 Gbps bandwidth. A bidirectional AUX channel with 1 Mbps Manchester encoding provides side-band communication. The sink uses a hot plug detect (HPD) signal to announce its presence, and the source uses the same signal to initiate link configuration.

Figure 2-1: DisplayPort Source and Sink Communication

The main link has three selectable data rates: 1.62, 2.7, and 5.4 Gbps.



Device Family Support

The following table lists the level of support offered by the DisplayPort IP core for each Altera device family.

- *Preliminary support*—Indicates that the IP core is verified with preliminary timing models for this device family. the IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.
- *Final support*—The IP core is verified with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

Table 2-1: Device Family Support

Device Family	Support
Arria V GX	Preliminary support
Arria V GZ	Preliminary support
Cyclone V	Preliminary support
Stratix V	Preliminary support
Other device families	No support

Table 2-2: Link Rate Support by Device Family

Device Family	20-bit mode	40-bit mode
Arria V GX	RBR, HBR	RBR, HBR, HBR2
Arria V GZ	RBR, HBR, HBR2	RBR, HBR, HBR2
Cyclone V	Reduced Bit Rate (RBR) , High Bit Rate (HBR)	RBR, HBR
Stratix V	RBR, HBR, HBR2	RBR, HBR, HBR2

IP Core Verification

Before releasing a publicly available version of the DisplayPort IP core, Altera runs a comprehensive verification suite in the current version of the Quartus® II software. These tests use standalone methods and the Qsys system integration tool to create the instance files. These files are tested in simulation and hardware to confirm functionality. Altera tests and verifies the DisplayPort IP core in hardware for different platforms and environments.

The DisplayPort IP core has been tested at VESA Plugtest events and passes the Unigraf DisplayPort Link Layer CTS tests.

Performance and Resource Utilization

This section contains tables showing IP core variation size and performance examples.

The following table lists the resources and expected performance for selected variations. The results were obtained using the Quartus II software v14.0 for the following devices:

- Arria V (5AGXFB3H4F40C5)
- Cyclone V (5CGTFD9E5F35C7)
- Stratix V (5SGXEA7K2F40C2)

Table 2-3: DisplayPort IP Core FPGA Resource Utilization

M10K for Arria V devices and M20K for Stratix V devices.

Device	Parameters		ALMs	Logic Registers		Memory	
	Transceiver Interface (bits)	Mode, Lanes, and Color Depth		Primary	Secondary	Bits	M10K or M20K
Arria V GX	20	Duplex, 4 lanes, 24 bpp	7,144	9,837	0	29k	18
	40		13,693	13,286	0	42k	32
Cyclone V	20	Duplex, 4 lanes, 24 bpp	7,200	9,995	0	29k	18
	40		13,735	13,480	0	42k	32
Stratix V	20	Duplex, 4 lanes, 24 bpp	7,171	10,175	0	29k	18
	40		13,679	13,762	0	42k	32

Related Information

Fitter Resources Reports

More information about Quartus II resource utilization reporting.

Getting Started 3

2014.06.30

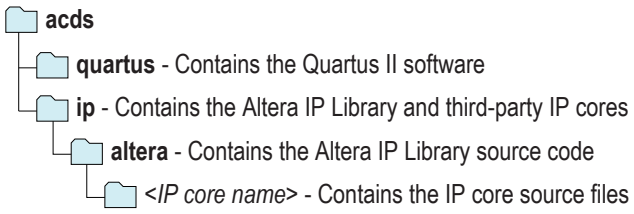
UG-01131 Subscribe Send Feedback

This chapter provides an overview of the DisplayPort design flow. The IP core is installed as part of the Quartus II installation process.

Installing and Licensing IP Cores

The Quartus II software includes the Altera IP Library. The library provides many useful IP core functions for production use without additional license. You can fully evaluate any licensed Altera IP core in simulation and in hardware until you are satisfied with its functionality and performance. Some Altera IP cores, such as MegaCore® functions, require that you purchase a separate license for production use. After you purchase a license, visit the Self Service Licensing Center to obtain a license number for any Altera product.

Figure 3-1: IP Core Installation Path



Note: The default IP installation directory on Windows is ***<drive>*:\altera**<*version number*>; on Linux it is <*home directory*>**/altera/** <*version number*>.

Related Information

- **Altera Licensing Site**
- **Altera Software Installation and Licensing Manual**

OpenCore Plus IP Evaluation

Altera's free OpenCore Plus feature allows you to evaluate licensed MegaCore IP cores in simulation and hardware before purchase. You need only purchase a license for MegaCore IP cores if you decide to take your design to production. OpenCore Plus supports the following evaluations:

- Simulate the behavior of a licensed IP core in your system.

- Verify the functionality, size, and speed of the IP core quickly and easily.
- Generate time-limited device programming files for designs that include IP cores.
- Program a device with your IP core and verify your design in hardware

OpenCore Plus evaluation supports the following two operation modes:

- Untethered—run the design containing the licensed IP for a limited time.
- Tethered—run the design containing the licensed IP for a longer time or indefinitely. This requires a connection between your board and the host computer.

Note: All IP cores using OpenCore Plus in a design time out simultaneously when any IP core times out.

IP Catalog and Parameter Editor

The Quartus II IP Catalog (**Tools** > **IP Catalog**) and parameter editor help you easily customize and integrate IP cores into your project. You can use the IP Catalog and parameter editor to select, customize, and generate files representing your custom IP variation.

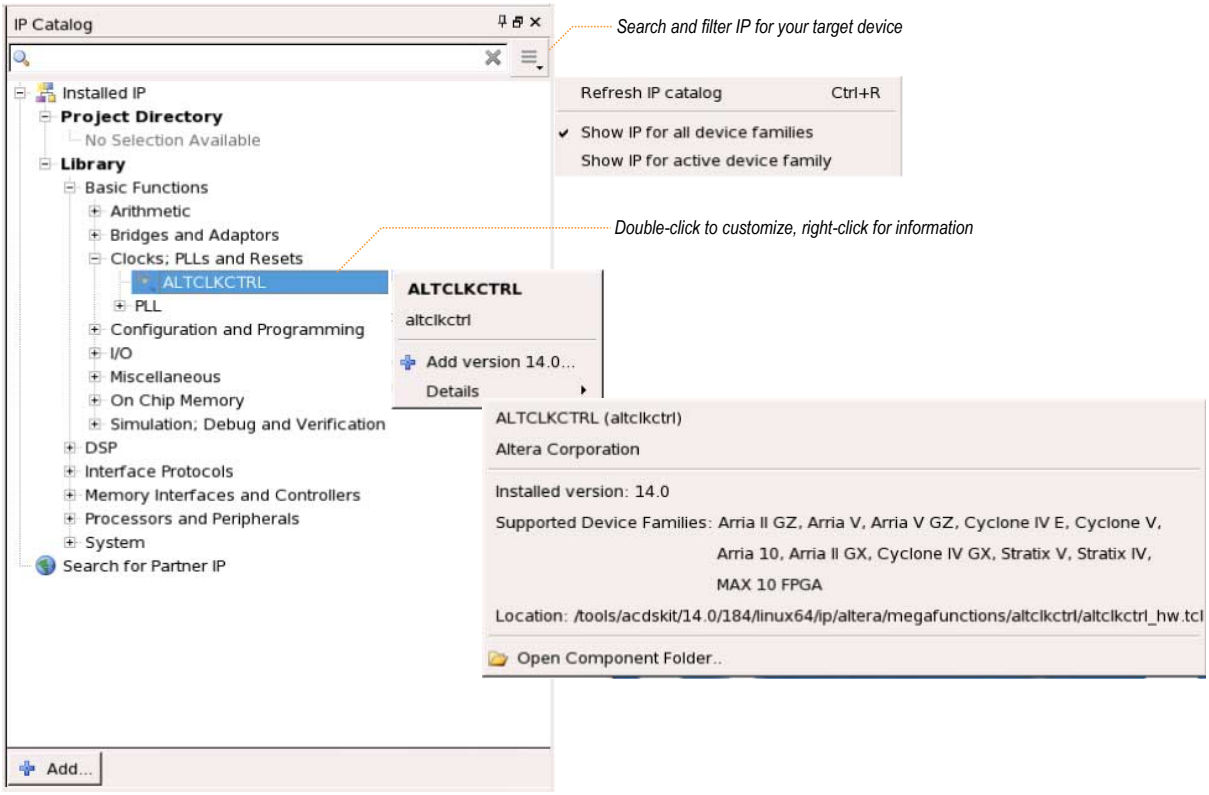
The IP Catalog automatically displays the IP cores available for your target device. Double-click any IP core name to launch the parameter editor and generate files representing your IP variation. The parameter editor prompts you to specify your IP variation name, optional ports, architecture features, and output file generation options. The parameter editor generates a top-level **.qsys** or **.qip** file representing the IP core in your project. Alternatively, you can define an IP variation without an open Quartus II project. When no project is open, select the **Device Family** directly in IP Catalog to filter IP cores by device.

Note: The IP Catalog is also available in Qsys (**View** > **IP Catalog**). The Qsys IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the Quartus II IP Catalog.

Use the following features to help you quickly locate and select an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**.
- Search to locate any full or partial IP core name in IP Catalog. Click **Search for Partner IP**, to access partner IP information on the Altera website.
- Right-click an IP core name in IP Catalog to display details about supported devices, installation location, and links to documentation.

Figure 3-2: Quartus II IP Catalog



Note: The IP Catalog and parameter editor replace the MegaWizard™ Plug-In Manager in the Quartus II software. The Quartus II software may generate messages that refer to the MegaWizard Plug-In Manager. Substitute "IP Catalog and parameter editor" for "MegaWizard Plug-In Manager" in these messages.

Specifying IP Core Parameters and Options

Follow these steps to specify IP core parameters and options.

1. In the IP Catalog (**Tools** > **IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name for your custom IP variation. This name identifies the IP core variation files in your project. If prompted, also specify the target Altera device family and output file HDL preference. Click **OK**.
3. Specify parameters and options for your IP variation:
   - Optionally select preset parameter values. Presets specify all initial parameter values for specific applications (where provided).
   - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
   - Specify options for generation of a timing netlist, simulation model, testbench, or example design (where applicable).

- Specify options for processing the IP core files in other EDA tools.

4. Click **Finish** or **Generate** to generate synthesis and other optional files matching your IP variation specifications. The parameter editor generates the top-level **.qip** or **.qsys** IP variation file and HDL files for synthesis and simulation. Some IP cores also simultaneously generate a testbench or example design for hardware testing.
5. To generate a simulation testbench, click **Generate** > **Generate Testbench System**. **Generate Testbench System** is not available for some IP cores that do not provide a simulation testbench.
6. To generate a top-level HDL example for hardware verification, click **Generate** > **HDL Example**. **Generate** > **HDL Example** is not available for some IP cores.

The top-level IP variation is added to the current Quartus II project. Click **Project** > **Add/Remove Files in Project** to manually add a **.qip** or **.qsys** file to a project. Make appropriate pin assignments to connect ports.

Simulating the Design

You can simulate your DisplayPort IP core variation using the simulation model that the MegaWizard Plug-In Manager generates. The simulation model files are generated in vendor-specific subdirectories of your project directory. The DisplayPort IP core includes a simulation example.

The following sections teach you how to simulate your MegaWizard Plug-In Manager flow generated DisplayPort IP core variation with the generated simulation model.

Related Information

- **DisplayPort IP Core Simulation Example** on page 7-1

Simulating with the ModelSim Simulator

To simulate using the Mentor Graphics ModelSim simulator, perform the following steps:

1. Start the ModelSim simulator.
2. In ModelSim, change directory to the project simulation directory <*variation*>**_sim/mentor**.
3. Type the following commands to set up the required libraries and compile the generated simulation model:

```
do msim_setup.tcl

ld

run -all
```

Compiling the Full Design and Programming the FPGA

You can use the **Start Compilation** command on the Processing menu in the Quartus II software to compile your design. After successfully compiling your design, program the targeted Altera device with the Programmer and verify the design in hardware.

Related Information

- **Quartus II Incremental Compilation for Hierarchical and Team-Based Design**

- **Device Programming**

DisplayPort Source 4

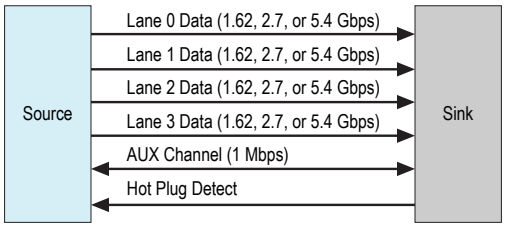
2014.06.30

UG-01131 Subscribe Send Feedback

Source Overview

The DisplayPort source has a scalable main link with 1, 2, or 4 lanes for a total up to 21.6 Gbps bandwidth. A bidirectional AUX channel with 1 Mbps Manchester encoding provides side-band communication.

Figure 4-1: DisplayPort Source



The main link has three selectable data rates: 1.62, 2.7, and 5.4 Gbps. The source device sets the lane count and link rate combination (referred to as the policy) according to the sink's capabilities and required video bandwidth. The IP core transmits the video and audio streams on the main link with embedded clocking. The DisplayPort protocol embeds the clocks such that the pixel and audio clocks are decoupled from the transmission clock.

The IP core transmits data in a scrambled ANSI 8B/10B format. The data transmission includes redundancy for error detection. The secondary data stream, such as an audio stream, uses a Reed-Solomon encoder for error correction.

The AUX channel is an AC-coupled differential pair for bidirectional communication. The signaling is a self-clocked Manchester encoding at 1 Mbps. As in the 100-T Ethernet protocol, the encoder uses a preceding synchronization pattern in each 16-byte maximum packet.

The AUX channel uses a master-slave hierarchy in which the source (master) initiates all communication.

Source Functional Description

The DisplayPort source has a complete set of parameters for optimizing device resources.

The DisplayPort source consists of a DisplayPort encoder block, a transceiver management block, and a controller interface block with an Avalon-MM interface for connecting with an embedded controller such as a Nios II processor. You configure the ports using an RTL wrapper instantiation or by implementing the IP core as a Qsys component.

Figure 4-2: DisplayPort Source Top-Level Block Diagram

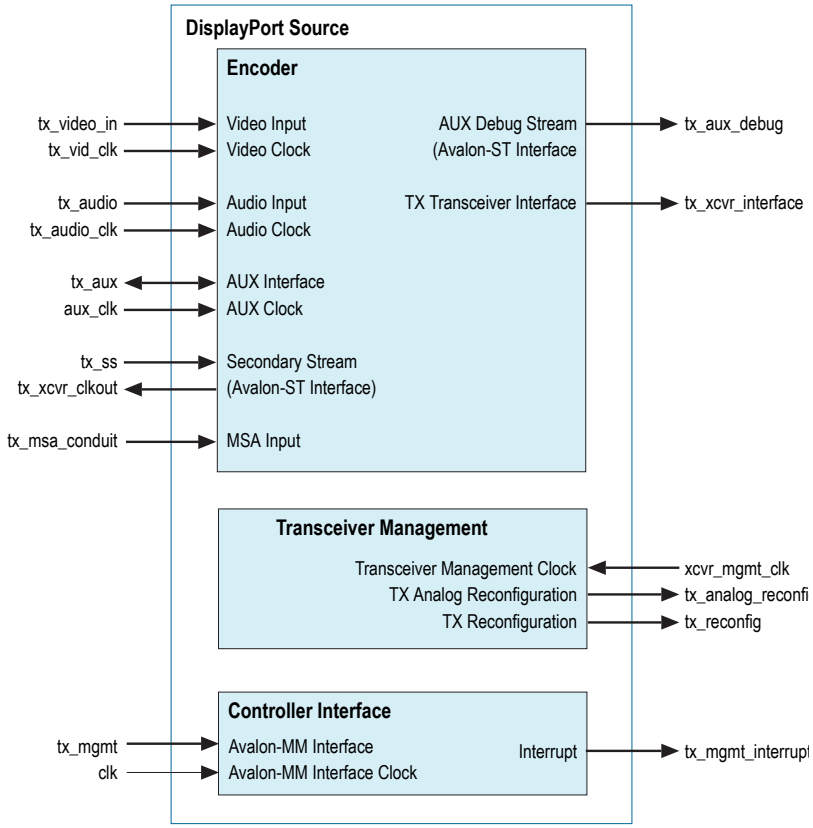
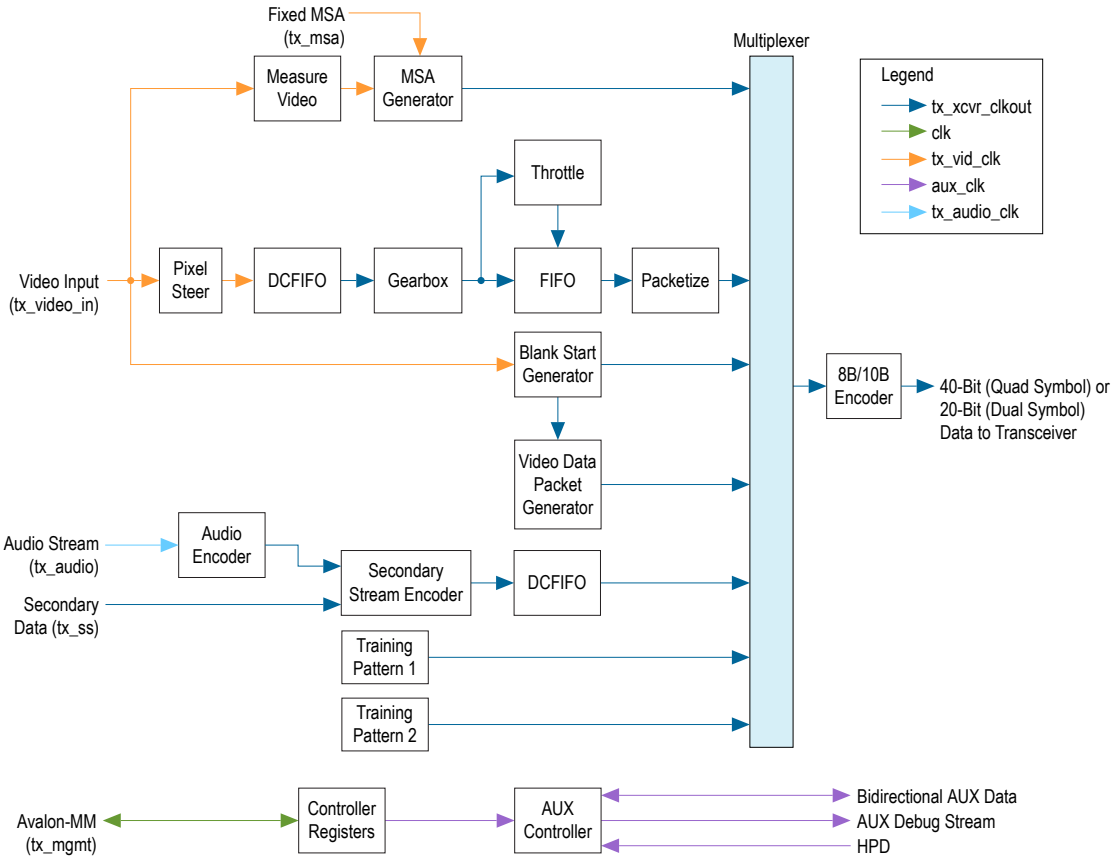


Figure 4-3: DisplayPort Source Functional Block Diagram



The source accepts a standard H-sync, V-sync, and data enable video stream for encoding. The IP core latches and processes the video data before processing it using the `tx_vid_clk` input. The video data width supports 6 to 16 bits-per color (bpc) and is user selectable. If you set the **Pixel input mode** option to **Dual** or **Quad**, the video input can accept two or four pixels per clock, thereby extending the pixel clock rate capability. The IP core forwards video input to three parallel paths.

Main Data Path

The main data path consists of the packetizer, measurement, and blank generator paths. The IP core multiplexes data from these three paths and outputs it through an 8B/10B encoder.

Packetizer Path

The packetizer path provides video data resampling and packetization, and consists of the following steps:

1. The pixel steer block decimates the data to the requested lane count (1, 2, or 4).
2. The DCFIFO crosses the data into the main link clock domain (`tx_xcvr_clkout`, generated by the transceiver), which can be 270, 135, 81, 67.5, or 40.5 MHz depending on the actual main link rate requested and the symbols per clock.
3. The gearbox resamples the video data according to the specified color depth. You can optimize the gearbox by implementing fewer color depths. For example, you can reduce the resources required to implement

the system by supporting only the color depths you need instead of the complete set of color depths specified in the DisplayPort specification.

4. The IP core packetizes the re-sampled data. The DisplayPort specification requires data to be sent in a transfer unit (TU), which can be 32 to 64 link symbols long. To reduce complexity, the DisplayPort source uses a fixed 64-symbol TU. The specification also requires that the video data be evenly distributed within the TUs composing a full active video line. A throttle function distributes the data and regulates it to ensure that the TUs leaving the IP core are evenly packed.

Note: A minimal DisplayPort system should support both 6 and 8 bpc. The VESA DisplayPort specification requires support for a mandatory VGA fail safe mode (640 x 480 at 6 bpc).

The packetizer punctuates the outgoing 16-bit data stream with the correct packet comma codes. Internally, the packetizer uses a symbol and a TU counter to ensure that it respects the TU boundaries.

Measurement Path

The measurement path determines the video geometry required for the DisplayPort main stream attributes (MSA), which are sent once every vertical blanking interval. Optionally, the IP core can import a fixed MSA data parameter from an external port, removing the measurement logic. This feature is useful for embedded systems that only use known resolutions and synchronous pixel clocks.

Blank Generator Path

The blank generator path determines when to send the blank start comma codes with their corresponding video data packets. This path can operate in enhanced or standard framing mode.

Multiplexer

The IP core multiplexes the packetized data, MSA data, and blank generator data into a single stream. The combined data goes through 8B/10B encoding and is available as a 20-bit double-rate or a 40-bit quad-rate DisplayPort encoded video port. The 20- or 40-bit port connects directly to the Altera high-speed output transceiver.

During training periods, the source can send the DisplayPort clock recovery and symbol lock test patterns (training pattern 1 and training pattern 2, respectively).

The source also implements an AUX channel controller, which you access using an embedded controller. The embedded controller acts as an Avalon-MM master and sends read/write commands to the Avalon-MM slave interface. The IP core clocks the AUX channel using a 16 MHz clock input (`aux_clk`).

Related Information

Controller Interface on page 4-10

Embedded DisplayPort (eDP) Support

The DisplayPort IP core is compliant with eDP version 1.3. eDP is based on the VESA DisplayPort standard. It has the same electrical interface and can share the same video port on the controller. The DisplayPort IP core supports both full (normal) link training (default) and fast link training, which is a mandatory eDP feature.

Source Parameters

You set parameters for the source using the DisplayPort parameter editor.

Table 4-1: Source Parameters

Parameter	Description
Device family	Targeted device family (Arria V GX, Arria V GZ, Cyclone V, or Stratix V) ; matches the project device family.
Support DisplayPort source	Enable DisplayPort source.
Maximum video input color depth	Video input interface port bits per color (bpc). Determines top-level video input port width (for example, 6 bpc = 18 bits, 16 bpc = 48 bits).
TX maximum link rate	Maximum link rate. 5.4 Gbps, 2.7 Gbps, 1.62 Gbps. **Note:** Cyclone V devices do not support 5.4 Gbps.
Maximum lane count	Maximum lanes used (1, 2, or 4).
Symbol output mode	Specify how many symbols are transferred during each clock cycle: dual or quad symbol, or TX transceiver data width: dual (20 bits) or quad (40 bits) .
Pixel input mode	Specify the number of pixels per clock (single, dual, or quad symbol). • If you select dual pixels per clock, the pixel clock is ½ of the full rate clock and the video port becomes two times wider. • If you select four pixels per clock, the pixel clock is ¼ of the full rate clock and the video port becomes four times wider.
Scrambler seed value	Initial seed for scrambler block. Use 16'hFFFF for normal DP and 16'hFFFE for eDP.
Invert transceiver polarity	Invert transceiver polarity.
Support analog reconfiguration	Enable the analog reconfiguration interface if you are not using an external re-driver solution.
Enable AUX debug stream	Send source AUX traffic output to an Avalon-ST port.
Import fixed MSA	Used fixed MSA.
Interlaced input video	Interlace the input video. Turn on for interlaced, turn off for progressive.
Support CTS test automation	Support CTS test automation.
Support secondary data channel	Enables secondary data.

Parameter	Description
Support audio data channel	Enables audio packet encoding. **Note:** To use this parameter, you must turn on the **Support secondary data channel** parameter.
Number of audio data channels	Number of audio channels supported.
6-bpc RGB or YCbCr 4:4:4 (18 bpp)	Support 18 bpp encoding. **Note:** The IP core does not support YCbCr for 14.0 release.
8-bpc RGB or YCbCr 4:4:4 (24 bpp)	Support 24 bpp encoding. **Note:** The IP core does not support YCbCr for 14.0 release.
10-bpc RGB or YCbCr 4:4:4 (30 bpp)	Support 30 bpp encoding. **Note:** The IP core does not support YCbCr for 14.0 release.
12-bpc RGB or YCbCr 4:4:4 (36 bpp)	Support 36 bpp encoding. **Note:** The IP core does not support YCbCr for 14.0 release.
16-bpc RGB or YCbCr 4:4:4 (48 bpp)	Support 48 bpp decoding. **Note:** The IP core does not support YCbCr for 14.0 release.
8-bpc YCbCr 4:2:2 (16 bpp)	Support 16 bpp encoding. Reserved for future use. **Note:** The IP core does not support YCbCr for 14.0 release.
10-bpc YCbCr 4:2:2 (20 bpp)	Support 20 bpp encoding. Reserved for future use. **Note:** The IP core does not support YCbCr for 14.0 release.
12-bpc YCbCr 4:2:2 (24 bpp)	Support 24 bpp encoding. Reserved for future use. **Note:** The IP core does not support YCbCr for 14.0 release.

Parameter	Description
16-bpc YCbCr 4:2:2 (32 bpp)	Support 32 bpp encoding. Reserved for future use. **Note:** The IP core does not support YCbCr for 14.0 release.

Source Interfaces

The following tables list the source's port interfaces. Your instantiation contains only the interfaces that you have enabled.

Table 4-2: Controller Interface

Interface	Port Type	Clock Domain	Port	Direction	Description
clk	Clock	N/A	`clk`	Input	Clock for embedded controller.
reset	Reset	`clk`	`reset`	Input	Reset for embedded controller.
tx_mgmt	AV-MM	`clk`	`tx_mgmt_address[8:0]`	Input	Avalon-MM interface for embedded controller.
			`tx_mgmt_chipselect`	Input	
			`tx_mgmt_read`	Input	
			`tx_mgmt_write`	Input	
			`tx_mgmt_writedata[31:0]`	Input	
			`tx_mgmt_readdata[31:0]`	Output	
			`tx_mgmt_waitrequest`	Output	
tx_mgmt_irq	IRQ	`clk`	`tx_mgmt_irq`	Output	Interrupt for embedded controller.

Table 4-3: Transceiver Management Interface

n is the number of TX lanes

Interface	Port Type	Clock Domain	Port	Direction	Description
xcvr_mgmt_clk	Clock	N/A	`xcvr_mgmt_clk`	Input	Transceiver management clock.

Interface	Port Type	Clock Domain	Port	Direction	Description
tx_analog_reconfig	Conduit	xcvr_mgmt_clk	tx_vod[2n - 1:0]	Output	Transceiver analog reconfiguration handshaking.
			tx_emp[2n - 1:0]	Output	
			tx_analog_reconfig_req	Output	
			tx_analog_reconfig_ack	Input	
			tx_analog_reconfig_busy	Input	
tx_reconfig	Conduit	xcvr_mgmt_clk	tx_link_rate[1:0]	Output	Transceiver link rate reconfiguration handshaking.
			tx_reconfig_req	Output	
			tx_reconfig_ack	Input	
			tx_reconfig_busy	Input	

Table 4-4: Video Interface

v is the number of bits per color and *p* is the pixels per clock (1 = single, 2 = dual, and 4 = quad)

Interface	Port Type	Clock Domain	Port	Direction	Description
tx_vid_clk	Clock	N/A	tx_vid_clk	Input	Video clock.
tx_video_in	Conduit	tx_vid_clk	tx_vid_data[3v*p-1:0]	Input	Video data and standard H/V synchronization video port input.
			tx_vid_v_sync[p-1:0]	Input	
			tx_vid_h_sync[p-1:0]	Input	
			tx_vid_f[p-1:0]	Input	
			tx_vid_de[p-1:0]	Input	

Table 4-5: AUX Interface

Interface	Port Type	Clock Domain	Port	Direction	Description
aux_clk	Clock	N/A	aux_clk	Input	AUX channel clock.
aux_reset	Reset	aux_clk	aux_reset	Input	AUX channel reset.
tx_aux	Conduit	aux_clk	tx_aux_in	Input	AUX channel interface.
			tx_aux_out	Output	
			tx_aux_oe	Output	
			tx_hpd	Input	

Interface	Port Type	Clock Domain	Port	Direction	Description
tx_aux_debug	AV-ST	aux_clk	tx_aux_debug_data[31:0]	Output	Avalon-ST stream of AUX data for debugging.
			tx_aux_debug_valid	Output	
			tx_aux_debug_sop	Output	
			tx_aux_debug_eop	Output	
			tx_aux_debug_err	Output	
			tx_aux_debug_cha	Output	

Table 4-6: Secondary Interface

Interface	Signal Type	Clock Domain	Port	Direction	Description
tx_xcvr_clkout	Clock	N/A	tx_xcvr_clkout	Output	TX transceiver clock out and clock for secondary stream
MSA (tx_msa_conduit)	Conduit	tx_xcvr_clkout	tx_msa[191:0]	Input	Input port for fixed MSA parameters.
Secondary Stream (tx_ss)	AV-ST	tx_xcvr_clkout	tx_ss_data[127:0]	Input	Secondary stream interface.
			tx_ss_valid	Input	
			tx_ss_ready	Output	
			tx_ss_sop	Input	
			tx_ss_eop	Input	

Table 4-7: Audio Interface

m is the number of TX audio channels

Interface	Signal Type	Clock Domain	Port	Direction	Description
Audio (tx_audio)	Clock	N/A	tx_audio_clk	Input	Audio clock
	Conduit	tx_audio_clk	tx_audio_lpcm_data[m*32-1:0]	Input	Audio sample data interface
			tx_audio_valid	Input	
			tx_audio_mute	Input	

Table 4-8: TX Transceiver Interface

n is the number of TX lanes, s is the number of symbols per clock

Interface	Port Type	Clock Domain	Port	Direction	Description
TX transceiver interface	Clock	N/A	tx_std_clkout[n-1:0]	Input	TX transceiver clock out
	Conduit	tx_std_clkout	tx_parallel_data[n*s*10-1:0]	Output	Parallel data for TX transceiver
	Conduit	N/A	tx_pll_powerdown[n-1:0]	Output	PLL power down for TX transceiver
	Conduit	xcvr_mgmt_clk	tx_digitalreset[n-1:0]	Output	Resets the digital TX portion of TX transceiver
	Conduit	N/A	tx_analogreset[n-1:0]	Output	Resets the analog TX portion of TX transceiver
	Conduit	N/A	tx_cal_busy[n-1:0]	Input	Calibration in progress signal from TX transceiver
	Conduit	N/A	tx_pll_locked[n-1:0]	Input	PLL locked signal from TX transceiver

Controller Interface

The controller interface allows you to control the source from an external or on-chip controller, such as the Nios II processor. The controller can control the DisplayPort link parameters and the AUX channel controller.

The AUX channel controller interface works with a simple serial-port-type peripheral that operates in a polled mode. Because the DisplayPort AUX protocol is a master-slave interface, the DisplayPort source (the master) starts a transaction by sending a request and then waits for a reply from the attached sink.

The controller interface includes a single interrupt source. The interrupt notifies the controller of an HPD signal state change. Your system can interrogate the DP_TX_STATUS register to determine the cause of the interrupt. Writing to the DP_TX_STATUS register clears the pending interrupt event.

Related Information

- **DisplayPort Source Register Map** on page 10-1

AUX Interface

The IP core has three ports that control the serial data across the AUX channel:

- Data input (tx_aux_in)
- Data output (tx_aux_out)
- Output enable (tx_aux_oe). The output enable port controls the direction of data across the bidirectional link.

These ports are clocked by the source's 16 MHz clock (`aux_clk`). The AUX channel's physical layer is a bidirectional 2.5 V SSTL Class II interface.

The source's AUX controller allows you to capture all bytes sent from and received by the AUX channel, which is useful for debugging. The IP core provides a standard stream interface that you can use to drive an Avalon-ST FIFO component directly.

Table 4-9: Source AUX Debug Interface Ports

Port	Comments
`tx_aux_debug_data[31:0]`	The source AUX debug interface inserts a 1 µs timestamp counter in bits [31:8]; bits [7:0] represent the byte received or transmitted.
`tx_aux_debug_valid`	Qualifies valid stream data.
`tx_aux_debug_sop`	Indicates the message packet's first byte.
`tx_aux_debug_eop`	Indicates the message packet's last byte. The last byte should be ignored and is not part of the message.
`tx_aux_debug_err`	Indicates if the IP core detects an error in the current byte.
`tx_aux_debug_cha`	Indicates the direction of the current byte. 1 = byte transmitted by the source, 0 = byte received from the sink.

Related Information

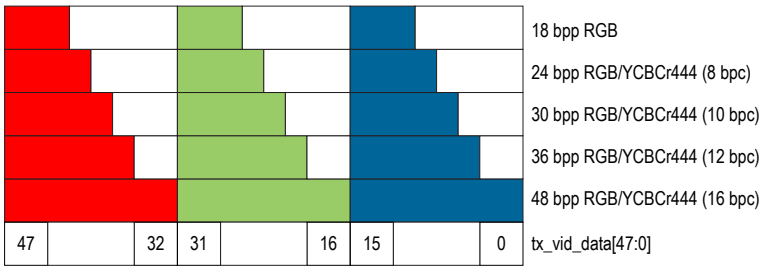
AN 522: Implementing Bus LVDS Interface in Supported Altera Device Families

Video Interface

The core inputs video to be encoded via the tx_video_in interface, which provides a standard H-sync and V-sync input with support for interlaced or progressive video. You specify the data input width via a parameter. The same input port transfers RGB and YCrCb data in either 4:4:4 or 4:2:2 color format. Data is most-significant bit aligned and formatted for 4:4:4.

Figure 4-4: Video Input Data Format

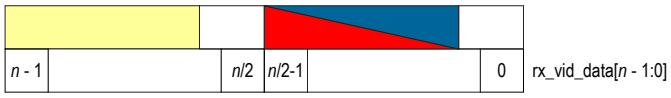
18 bpp to 48 bpp Port Width when tx_video_in Port Width is 48 (16 bpc, 1 Pixel per Clock)



The following figure shows the sub-sampled 4:2:2 color format for a video port width of *n*. The most-significant half of the video port always transfers the Y component while the least-significant half of the

video port transfers the alternate Cr or Cb component. If the Y/Cb/Cr component widths are less than *n*/2, they must be most-significant bit aligned with respect to the *n* and *n*/2-1 boundaries.

Figure 4-5: Sub-Sampled 4:2:2 Color Format Video Port



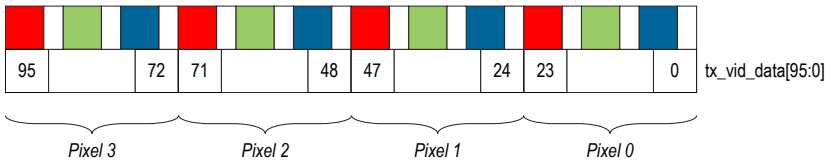
If you set the **Pixel input mode** option to **Dual** or **Quad**, the IP core inputs two or four pixels in parallel, respectively. To support video resolutions with horizontal active, front porch or back porch of a length not divisible by 2 or 4, the following signals are widened:

- Horizontal and vertical syncs
- Data enable

The following figure shows the pixel data order from least significant bits to most significant bits.

Figure 4-6: Video Input Data Alignment

For RGB 18 bpp when tx_video_in port Width is 96 (8 bpc, 4 Pixels per Clock)



TX Transceiver Interface

The transceiver or Native PHY IP core instance is no longer instantiated within the DisplayPort IP core.

The DisplayPort IP uses a soft 8B/10B encoder. This interface provides TX encoded video data (`tx_parallel_data`) in either dual symbol (20-bit) or quad symbol (40-bit) mode and drives the digital reset (`tx_digitalreset`), analog reset (`tx_analogreset`), and PLL powerdown signals (`tx_pll_powerdown`) of the transceiver.

Transceiver Reconfiguration Interface

You can reconfigure the transceiver to accept double or single reference clock:

- Double reference clocks—162 MHz clock for RBR or 270 MHz clock for HBR or HBR2.

  During run-time, you can reconfigure the transceiver to use either one of the two clocks. You use the Transceiver Reconfiguration Controller to switch between the two reference clocks. To switch them, reconfigure the logical reference clock source for the TX CMU PLL. The IP core sets `tx_link_rate` to:

  - 00 (RBR)
  - 01 (HBR)
  - 10 (HBR2)

- Single reference clock—135 MHz clock for all bit rates: RBR, HBR, and HBR2.

  During run-time, you can reconfigure the transceiver to operate in either one of the bit rate by changing TX CMU PLL divide ratio.

When the IP core makes a request, the `tx_reconfig_req` port goes high. The user logic asserts `tx_reconfig_ack` and then reconfigures the transceiver. During reconfiguration, the user logic holds `tx_reconfig_busy` high. The user logic drives it low when reconfiguration completes.

Note: The transceiver requires a reconfiguration controller. Reset the transceiver to a default state upon power-up.

Related Information

- **Altera Transceiver PHY IP Core User Guide**
  For more information about how to reconfigure the transceiver.
- **AN 645: Dynamic Reconfiguration of PMA Controls in Stratix V Devices**
  For more information about using the Transceiver Reconfiguration Controller to reconfigure the Stratix V Physical Media Attachment (PMA) controls dynamically.
- **AN 676: Using the Transceiver Reconfiguration Controller for Dynamic Reconfiguration in Arria V and Cyclone V Devices**
  For more information about using the Transceiver Reconfiguration Controller to reconfigure the Arria V Physical Media Attachment (PMA) controls dynamically.
- **AN 678: High-Speed Link Tuning Using Signal Conditioning Circuitry**
  For information about link tuning.

Transceiver Analog Reconfiguration Interface

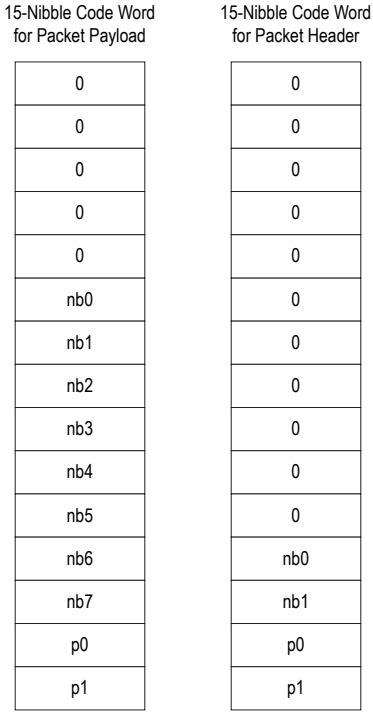
The tx_analog_reconfig interface uses the `tx_vod` and `tx_emp` transceiver management control ports. You must map these ports for the device you are using. To change these values, the core drives `tx_analog_reconfig_req` high. Then, the user logic sets `tx_analog_reconfig_ack` high to acknowledge and drives `tx_analog_reconfig_busy` high during reconfiguration. When reconfiguration completes, the user logic drives `tx_analog_reconfig_busy` low.

Secondary Stream Interface

You can transmit the secondary stream data over the DisplayPort main link through the secondary stream (tx_ss) interface. This interface uses handshaking and back pressure to control packet delivery. Internally, the core uses a FIFO to store packets until a slot becomes available on the main link. If the FIFO fills up, the secondary stream interface stops accepting packets and applies back pressure. The packet must be available at the time of sending because the ss_tx port does not support forward pressure.

The tx_ss interface input data format corresponds to four, 15-nibble code words as specified by the DisplayPort version 1.1a specification section 2.2.6.3. These 15-nibble code words are supplied by the upstream Reed-Solomon encoder. The format differs for header and payload as shown in the following figure.

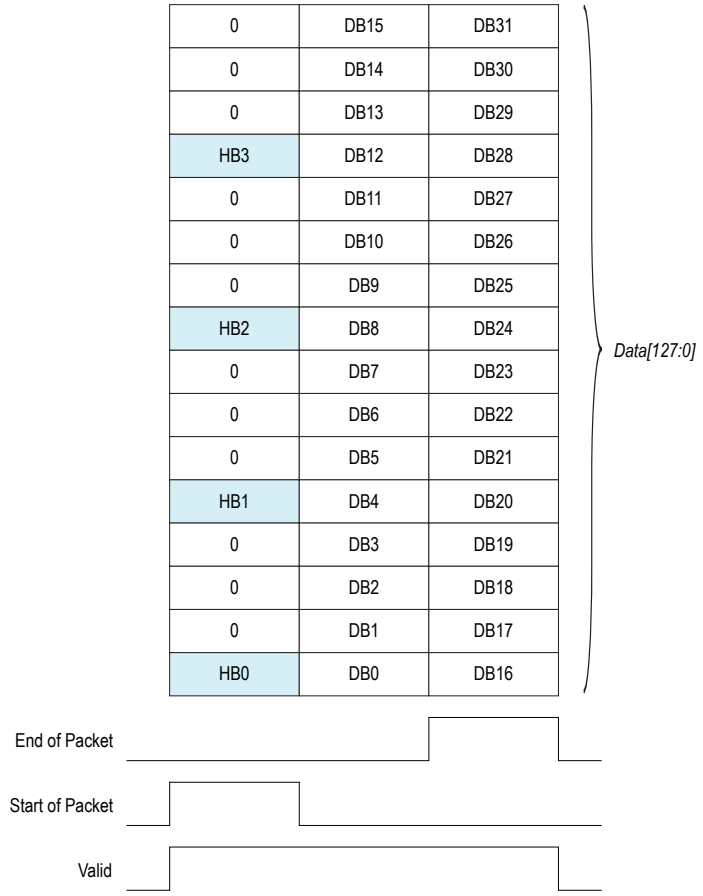
Figure 4-7: Secondary Stream Input Data Format



The following figure shows a typical secondary stream packet with a four byte header (HB0, HB1, HB2 and HB3) and a 32-byte payload (DB0 … DB31). The core calculates the associated parity bytes. The secondary stream interface uses the start-of-packet (SOP) and end-of-packet (EOP) to determine if the current input is a header or payload.

Payloads that only contain the first 16 bytes can assert the EOP on the second cycle to terminate the packet sequence. Data is clocked in to the secondary stream interface via the `tx_xcvr_clk`. This clock is the same phase and frequency as the main-link lane 0 clock.

Figure 4-8: Typical Secondary Stream Packet



Audio Interface

The audio encoder is upstream of the secondary stream encoder. It generates the audio infoframe, timestamp, and audio sample packets from the incoming audio sample data stream. Then, it sends the three packet types to the secondary stream encoder before they are transmitted to the downstream sink device.

The audio port is parameterized for the number of audio channels required in the design. You can use 2 to 8 channels. Each channel's audio data is sent to the `tx_audio_lpcm_data` port.

The IP core requires a `tx_audio_valid` signal for designs in which the `tx_audio_clk` signal is higher than the actual sample clock. The `tx_audio_valid` signal qualifies the audio data on the `tx_audio_lpcm_data` input.

Table 4-10: Audio Signals

Signal	Comments
`tx_audio_clk`	Audio interface input clock.
`tx_audio_valid`	Audio input data valid.

Signal	Comments
tx_audio_mute	When asserted, indicates that audio muting is enabled.
tx_audio_lpcm_data[m*32-1:0]	*m*-channel, 32-bit audio sample data.

Figure 4-9: Audio Sample Data Bits

The packing format uses an IEC-60958-type encoding.

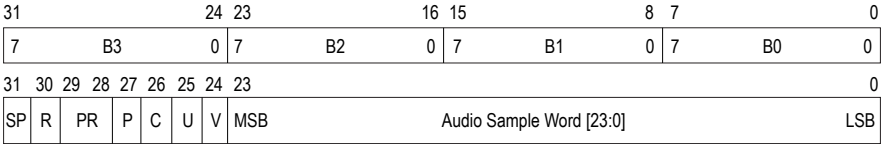


Table 4-11: Audio Sample Bit Field Definitions

Bit Name	Bit Position	Description
Audio sample word	Byte 2, bits 7:0 Byte 1, bits 7:0 Byte 0, bits 7:0	Audio data. The data content depends on the audio coding type. For LPCM audio, the audio most significant bit (MSB) is placed in byte 2, bit 7. If the audio data size is less than 24 bits, unused least significant bits (LSB) must be zero padded.
V	Byte 3, bit 0	Validity flag.
U	Byte 3, bit 1	User bit.
C	Byte 3, bit 2	Channel status.
P	Byte 3, bit 3	Parity bit.
PR	Byte 3, bits 4 - 5	Preamble code and its correspondence with IEC-60958 preamble: 00: Subframe 1 and start of the audio block (11101000 preamble) 01: Subframe1 (1110010 preamble) 10: Subframe 2 (1110100 preamble)
R	Byte3, bit 6	Reserved bit; must be 0.

Bit Name	Bit Position	Description
SP	Byte 3, bit 7	Sample present bit: 1: Sample information is present and can be processed. 0: Sample information is not present. All one-sample channels, used or unused, must have the same sample present bit value. This bit is useful for situations in which 2-channel audio is transported over a 4-lane main link. In this operation, main link lanes 2 and 3 may or may not have the audio sample data. This bit indicates whether the audio sample is present or not.

The source automatically generates the audio infoframe and fills it with only information about the number of channels used. Use the audio channel status to provide any information about the audio stream needed by downstream devices.

MSA Interface

For applications that use a known video source signal, the added resource of video measurement can be removed. In this scenario, the DP Source uses the MSA values presented on the tx_msa_conduit signal bundle. The bundle contents is shown below,

```
wire [191:0] tx_msa_conduit = {Mvid[23:0], Nvid[23:0], Htotal[15:0], Vtotal[15:0], HSP,
HSW[14:0], Hstart[15:0], Vstart[15:0], VSP, VSW[14:0], Hwidth[15:0], Vheight[15:0],
MISC0[7:0], MISC1[7:0]};
```

Table 4-12: tx_msa_conduit Port Signals

Bit	Signal	Comments
191:168	`Mvid[23:0]`	Mvid for the main video stream. Used for stream clock recovery from link symbol clock.
167:144	`Nvid[23:0]`	Nvid for the main video stream. Used for stream clock recovery from link symbol clock.
143:128	`Htotal[15:0]`	Horizontal total of received video stream in pixels
127:112	`Vtotal[15:0]`	Vertical total of received video stream in lines
111	`HSP`	H-sync polarity 0 = Active high, 1 = Active low
110:96	`HSW[14:0]`	H-sync width in pixels
95:80	`Hstart[15:0]`	Horizontal active start from H-sync start in pixels (H-sync width + Horizontal back porch)
79:64	`Vstart[15:0]`	Vertical active start from V-sync start in lines (V-sync width + Vertical back porch)

Bit	Signal	Comments
63	VSP	V-sync polarity 0 = Active high, 1 = Active low
62:48	VSW[14:0]	V-sync width in lines
47:32	Hwidth[15:0]	Active video width in pixels
31:16	Vheight[15:0]	Active video height in lines
15:8	MISC0[7:0]	The MISC0[7:1] and MISC1[7] fields indicate the color encoding format. The color depth is indicated in MISC0[7:5]: • 000 - 6 bpc • 001 - 8 bpc • 010 - 10 bpc • 011 - 12 bpc • 100 - 16 bpc For details about the encoding format, refer to the DisplayPort v1.2 specification.
7:0	MISC1[7:0]	

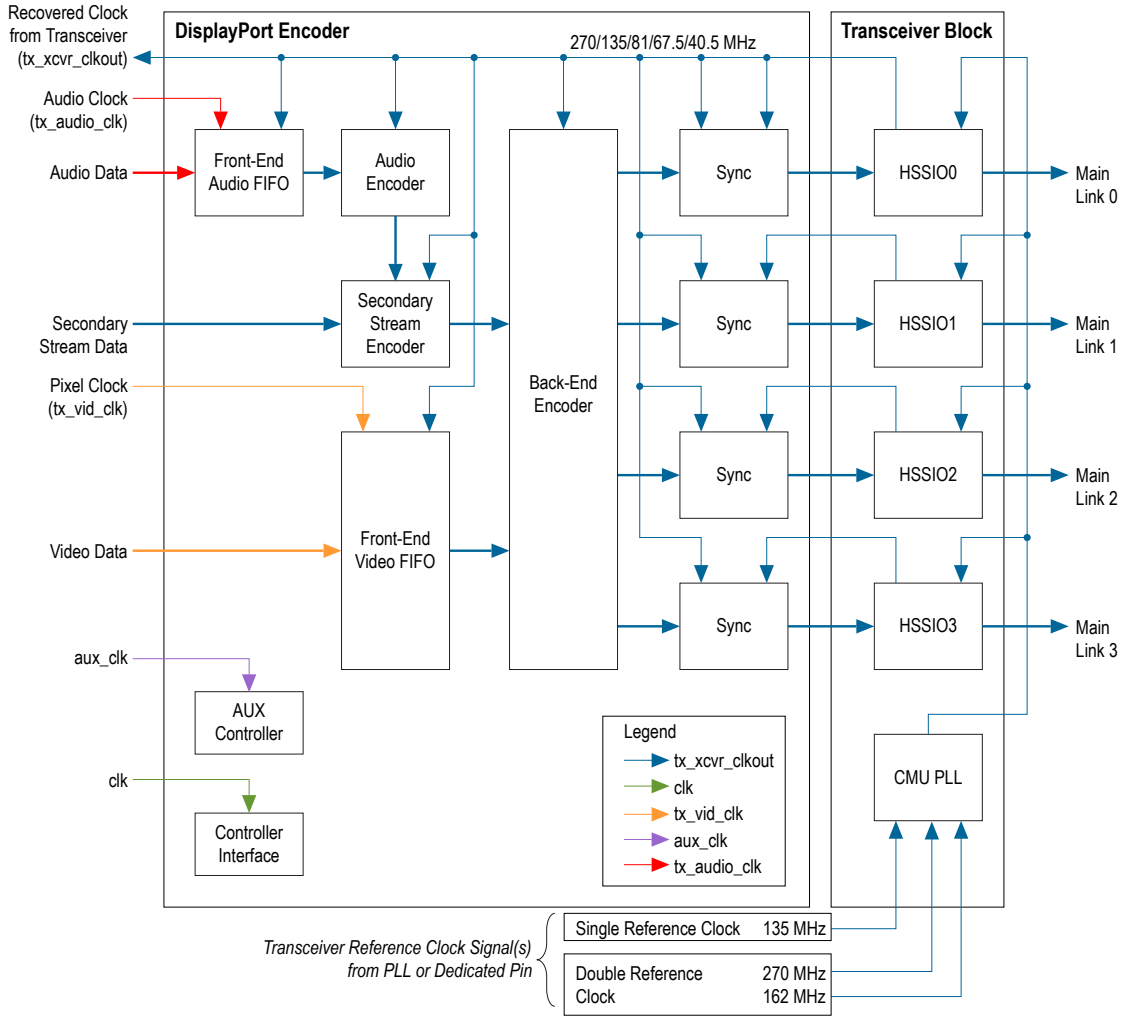
Source Clock Tree

The source uses the following clocks:

- Local pixel clock (tx_vid_clk), which clocks video data into the IP core.
- Main link clock (tx_xcvr_clkout), which clocks data out of the IP core and into the high-speed serial output (HSSI) components. The main link clock is the output of the CMU PLL clock. You can supply the CMU PLL with one of the following options:
  - Double reference clocks—either one of two clocks (270 MHz or 162 MHz) based on the actual data rate requested on the tx_link_rate port
  - Single reference clock (135 MHz)

    You can use other frequencies by changing the CMU PLL divider ratios and/or reconfiguring the transceiver. The 20- or 40- bit data fed to the HSSI is synchronized to a single HSSI[0] clock. If you select the dual symbol mode option, this clock is equal to the link rate divided by 20 (270, 135, or 81 MHz). If you turn on quad symbol mode, this clock is equal to the link rate divided by 40 (135, 67.5, or 40.5 MHz).
- 16 MHz clock (aux_clk), which the IP core requires to encode or decode the AUX channel. A separate clock (clk) clocks the Avalon-MM interface.
- tx_audio_clk for the audio interface.

Figure 4-10: Source Clock Tree



DisplayPort Sink 5

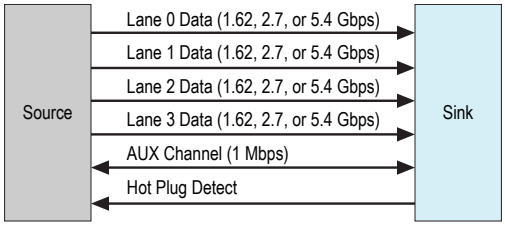
2014.06.30

UG-01131 Subscribe Send Feedback

Sink Overview

The DisplayPort sink has a scalable main link with 1, 2, or 4 lanes for a total up to 21.6 Gbps bandwidth. A bidirectional AUX channel with 1 Mbps Manchester encoding provides side-band communication. The sink drives a hot plug detect (HPD) signal to notify the source that a sink is present. Additionally, it provides an interrupt mechanism so that the sink can get the source's attention.

Figure 5-1: DisplayPort Sink Block Diagram



The main link has three selectable data rates: 1.62, 2.7, and 5.4 Gbps. The source device sets the lane count and link rate combination (referred to as the policy) according to the sink's capabilities and required video bandwidth.

The AUX channel is an AC-coupled differential pair for bidirectional communication. The signaling is a self-clocked Manchester encoding at 1 Mbps. Like 100-T Ethernet, the encoder uses a preceding synchronization pattern in each 16-byte maximum packet. The AUX channel uses a master/slave hierarchy in which the source (master) initiates all communication.

Sink Functional Description

The DisplayPort sink has a complete set of parameters for optimizing device resources.

The DisplayPort sink consists of a DisplayPort decoder block, a transceiver management block, and a controller interface block with an Avalon-MM interface for connecting with an embedded controller such as the Nios II processor. You can configure the ports using an RTL wrapper instantiation or implementing the IP core as a Qsys component.

Figure 5-2: DisplayPort Sink Top-Level Block Diagram

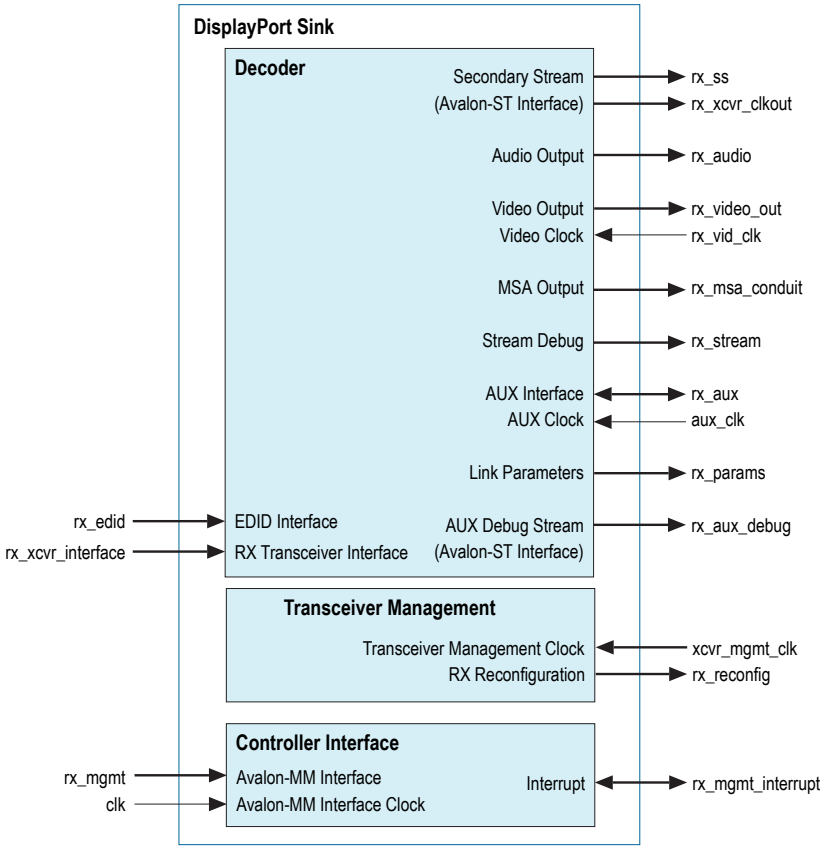
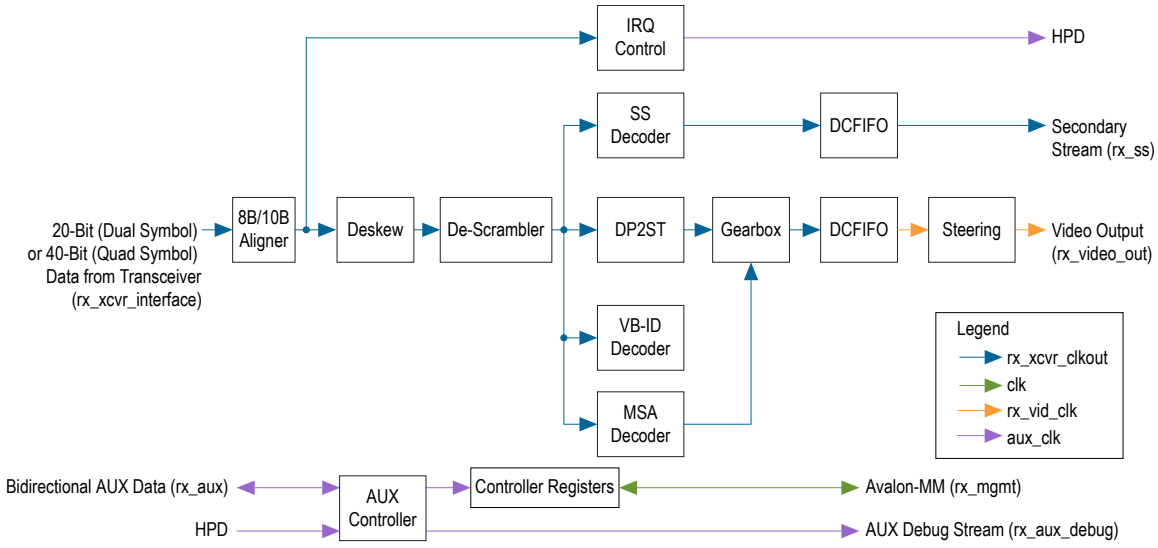


Figure 5-3: DisplayPort Sink Functional Block Diagram



The device transceiver sends 20-bit (dual symbol) or 40-bit (quad symbol) parallel DisplayPort data to the sink. Each data lane is clocked in to the IP core by its own respective clock output from the transceiver. Inside the sink, the four independent clock domains are synchronized to the lane 0 clock. Then, the IP core performs the following actions:

1. The IP core aligns the data stream and performs 8B/10B decoding.
2. The IP core deskews the data and then descrambles it.
3. The IP core splits the unscrambled data stream into parallel paths.
   a. The SS decoder block performs secondary stream decoding, which the core transfers into the `rx_xcvr_clkout` domain through a DCFIFO.
   b. The main data path extracts all pixel data from the incoming stream. Then, the gearbox block re-samples the pixel data into the current bit-per-pixel data width. Next, the IP core crosses the pixel data into the `rx_vid_clk` domain via a DCFIFO. Finally, the IP core steers the data into a single, dual, or quad pixel data stream.
   c. MSA decode path.
   d. Video decode path.

You configure the sink to output the video data as a proprietary data stream. You specify the output pixel data width at 6, 8, 10, 12, or 16 bpc. This format can interface with downstream Altera Video and Image Processing (VIP) Suite components.

The AUX controller can operate in an autonomous mode in which the sink controls all AUX channel activity without an external embedded controller. The IP core outputs an AUX debugging stream so that you can inspect the activity on the AUX channel in realtime.

Embedded DisplayPort (eDP) Support

The DisplayPort IP core is compliant with eDP version 1.3. eDP is based on the VESA DisplayPort standard. It has the same electrical interface and can share the same video port on the controller. The DisplayPort IP core supports both full (normal) link training and fast link training, which is a mandatory eDP feature.

Sink Parameters

You set parameters for the sink using the DisplayPort parameter editor.

Table 5-1: Sink Parameters

Parameter	Description
Device family	Targeted device family (Arria V, Arria V GZ, Cyclone V, or Stratix V); matches the project device family.
Support DisplayPort sink	Enable DisplayPort sink.
Maximum video output color depth	Video output interface port bits per color (bpc). Determines top level video output port width (e.g., 6 bpc = 18 bits, 16 bpc = 48 bits).

Parameter	Description
RX maximum link rate	Maximum link rate. 5.4 Gbps, 2.7 Gbps, 1.62 Gbps **Note:** Cyclone V devices do not support 5.4 Gbps.
Maximum lane count	Maximum lanes used (1, 2, or 4).
Symbol input mode	Specify how many symbols are transferred during each clock cycle (dual or quad symbol), or RX transceiver data width; dual (20 bits) or quad (40 bits).
Pixel output mode	Specify the number of pixels per clock (single, dual, or quad symbol). • If you select dual pixels per clock, the pixel clock is ½ of the full rate clock and the video port becomes two times wider. • If you select four pixels per clock, the pixel clock is ¼ of the full rate clock and the video port becomes four times wider.
Sink scrambler seed value	Scrambler block initial seed value. Use 16'hFFFF for DP and 16'hFFFFE for eDP.
Invert transceiver polarity	Invert the transceiver polarity.
Export MSA	Outputs MSA on top level port interface.
IEEE OUI	Specify an IEEE organizationally unique identifier (OUI) as part of the DPCD registers.
Enable GPU control	Use an embedded controller to control the sink.
Enable AUX debug stream	Enable AUX traffic output to an Avalon-STport.
Support CTS test automation	Support automated test features.
Support secondary data channel	Enable secondary data.
Support audio data channel	Enable audio packet decoding.
Number of audio data channels	Number of audio channels supported. **Note:** To use this parameter, you must turn on the **Support secondary data channel** parameter.
6-bpc RGB or YCbCr 4:4:4 (18 bpp)	Support 18 bpp decoding. **Note:** The IP core does not support YCbCr for 14.0 release.

Parameter	Description
8-bpc RGB or YCbCr 4:4:4 (24 bpp)	Support 24 bpp decoding. **Note:** The IP core does not support YCbCr for 14.0 release.
10-bpc RGB or YCbCr 4:4:4 (30 bpp)	Support 30 bpp decoding. **Note:** The IP core does not support YCbCr for 14.0 release.
12-bpc RGB or YCbCr 4:4:4 (36 bpp)	Support 36 bpp decoding. **Note:** The IP core does not support YCbCr for 14.0 release.
16-bpc RGB or YCbCr 4:4:4 (48 bpp)	Support 48 bpp decoding. **Note:** The IP core does not support YCbCr for 14.0 release.
8-bpc YCbCr 4:2:2 (16 bpp)	Support 16 bpp decoding. Reserved for future use.
10-bpc YCbCr 4:2:2 (20 bpp)	Support 20 bpp decoding. Reserved for future use.
12-bpc YCbCr 4:2:2 (24 bpp)	Support 24 bpp decoding. Reserved for future use.
16-bpc YCbCr 4:2:2 (32 bpp)	Support 32 bpp decoding. Reserved for future use.

Sink Interfaces

The following tables summarize the sink's interfaces. Your instantiation contains only the interfaces that you have enabled.

Table 5-2: Controller Interface

Interface	Port Type	Clock Domain	Port	Direction	Description
clk	Clock	N/A	`clk`	Input	Clock for embedded controller.
reset	Reset	`clk`	`reset`	Input	Reset for embedded controller.

Interface	Port Type	Clock Domain	Port	Direction	Description
rx_mgmt	AV-MM	clk	rx_mgmt_address[8:0]	Input	Avalon-MM interface for embedded controller.
			rx_mgmt_chipselect	Input	
			rx_mgmt_read	Input	
			rx_mgmt_write	Input	
			rx_mgmt_writedata[31:0]	Input	
			rx_mgmt_readdata[31:0]	Output	
			rx_mgmt_waitrequest	Output	
rx_mgmt_irq	IRQ	clk	rx_mgmt_irq	Output	Interrupt for embedded controller.

Table 5-3: Transceiver Management Interface

Interface	Port Type	Clock Domain	Port	Direction	Description
xcvr_mgmt_clk	Clock	N/A	xcvr_mgmt_clk	Input	Transceiver management clock.
rx_reconfig	Conduit	xcvr_mgmt_clk	rx_link_rate[1:0]	Output	Transceiver link rate reconfiguration handshaking.
			rx_reconfig_req	Output	
			rx_reconfig_ack	Input	
			rx_reconfig_busy	Input	

Table 5-4: Video Interface

v is the number bits per color and *p* is the pixels per clock (1 = single, 2 = dual, and 4 = quad)

Interface	Port Type	Clock Domain	Port	Direction	Description
rx_vid_clk	Clock	N/A	rx_vid_clk	Input	Video clock.

Interface	Port Type	Clock Domain	Port	Direction	Description
rx_video_out	Conduit	rx_vid_clk	rx_vid_valid[p-1:0]	Output	Video output.
			rx_vid_sol	Output	
			rx_vid_eol	Output	
			rx_vid_sof	Output	
			rx_vid_eof	Output	
			rx_vid_locked	Output	
			rx_vid_overflow	Output	
			rx_vid_data[3v*p-1:0]	Output	

Table 5-5: AUX Interface

Interface	Port Type	Clock Domain	Port	Direction	Description
aux_clk	Clock	N/A	aux_clk	Input	AUX channel clock.
aux_reset	Reset	aux_clk	aux_reset	Input	AUX channel reset.
rx_aux	Conduit	aux_clk	rx_aux_in	Input	AUX channel interface.
			rx_aux_out	Output	
			rx_aux_oe	Output	
			rx_hpd	Output	
rx_aux_debug	AV-ST	aux_clk	rx_aux_debug_data[31:0]	Output	Avalon-ST stream of AUX data for debugging.
			rx_aux_debug_valid	Output	
			rx_aux_debug_sop	Output	
			rx_aux_debug_eop	Output	
			rx_aux_debug_err	Output	
			rx_aux_debug_cha	Output	

Interface	Port Type	Clock Domain	Port	Direction	Description
EDID (rx_edid)	AV-MM	aux_clk	rx_edid_address[7:0]	Output	Avalon-MM master interface to external on-chip memory for EDID.
			rx_edid_read	Output	
			rx_edid_write	Output	
			rx_edid_writedata[7:0]	Output	
			rx_edid_readdata[7:0]	Input	
			rx_edid_waitrequest	Input	

Table 5-6: Debugging Interface

s is the number of symbols per clock.

Interface	Signal Type	Clock Domain	Port	Direction	Description
Link Parameters (rx_params)	Conduit	aux_clk	rx_lane_count[4:0]	Output	Sink current link lane count value.
Debugging (rx_stream)	Conduit	rx_xcvr_clkout	rx_stream_data[4*8*s-1:0]	Output	Raw symbol output stream.
			rx_stream_ctrl[4*s-1:0]	Output	
			rx_stream_valid	Output	
			rx_stream_clk	Output	

Table 5-7: Secondary Interface

Interface	Signal Type	Clock Domain	Port	Direction	Description
rx_xcvr_clkout	Clock	N/A	rx_xcvr_clkout	Output	Clock
MSA (rx_msa_conduit)	Conduit	rx_xcvr_clkout	rx_msa[216:0]	Output	Output for current MSA parameters received from the source.
Secondary Stream (rx_ss)	AV-ST	rx_xcvr_clkout	rx_ss_data[159:0]	Output	Secondary stream interface.
			rx_ss_valid	Output	
			rx_ss_sop	Output	
			rx_ss_eop	Output	

Table 5-8: Audio Interface

`m` is the number of RX audio channels

Interface	Signal Type	Clock Domain	Port	Direction	Description
Audio (rx_audio)	Conduit	`rx_xcvr_clkout`	`rx_audio_lpcm_data[m*32-1:0]`	Output	Decoded audio data
			`rx_audio_valid`	Output	
			`rx_audio_mute`	Output	
			`rx_audio_infoframe[39:0]`	Output	

Table 5-9: RX Transceiver Interface

n is the number of RX lanes, s is the number of symbols per clock

Interface	Port Type	Clock Domain	Port	Direction	Description
RX transceiver interface	Clock	N/A	rx_std_clkout[n-1:0]	Input	RX transceiver recovered clock.
	Conduit	rx_std_clkout	rx_parallel_data[n*s*10-1:0]	Input	Parallel data from RX transceiver .
	Conduit	N/A	rx_is_lockedtoref[n-1:0]	Input	When asserted, indicates that the RX CDR PLL is locked to the reference clock.
	Conduit	N/A	rx_is_lockedtodata[n-1:0]	Input	When asserted, indicates that the RX CDR PLL is locked to the incoming data.
	Conduit	rx_xcvr_clkout	rx_bitslip[n-1:0]	Output	Use to control bit slipping manually.
	Conduit	N/A	rx_cal_busy[n-1:0]	Input	Calibration in progress signal from RX transceiver.
	Conduit	xcvr_mgmt_clk	rx_analogreset[n-1:0]	Output	When asserted, resets the RX CDR.
	Conduit	xcvr_mgmt_clk	rx_digitalreset[n-1:0]	Output	When asserted, resets the RX PCS.
	Conduit	xcvr_mgmt_clk	rx_set_locktoref[n-1:0]	Output	Forces the RX CDR circuitry to lock to the phase and frequency of the input reference clock.
	Conduit	xcvr_mgmt_clk	rx_set_locktodata[n-1:0]	Output	Forces the RX CDR circuitry to lock to the received data.

Controller Interface

The controller interface allows you to control the sink from an external or on-chip controller, such as the Nios II processor for debugging. The controller interface is an Avalon-MM slave that also allows access to the sink's internal status registers.

The sink asserts the rx_mgmt_irq port when issuing an interrupt to the controller.

Related Information

-
- **DisplayPort Sink Register Map and DPCD Locations** on page 11-1

AUX Interface

The IP core has three ports to control the serial data across the AUX channel:

- Data input (`rx_aux_in`)
- Data output (`rx_aux_out`)
- Output enable (`rx_aux_oe`). The output enable port controls the direction of data across the bidirectional link.

The AUX channel's physical layer is a bidirectional 2.5 V SSTL Class II interface.

A state machine decodes the incoming AUX channel's Manchester encoded data using the 16 MHz clock. The message parsing drives the state machine input directly. The state machine performs all lane training and EDID link-layer services.

The sink's AUX interface also generates appropriate HPD IRQ events. These events occur if the sink's main link decoder detects a signal loss.

AUX Debug Interface

The AUX controller lets you capture all bytes sent from and received by the AUX channel, which is useful for debugging. The IP core supports a standard stream interface that can drive an Avalon-ST FIFO component directly.

Table 5-10: Sink AUX Debug Interface Ports

The table below describes the stream ports.

Port	Comments
`rx_aux_debug_data[31:0]`	The sink AUX debug interface inserts a 1 µs timestamp counter in bits [31:8]. Bits [7:0] represent the bytes received or transmitted.
`rx_aux_debug_valid`	Qualifies valid stream data.
`rx_aux_debug_sop`	Indicates the message packet's first byte.
`rx_aux_debug_eop`	Indicates the message packet's last byte. The last byte should be ignored and is not part of the message.
`rx_aux_debug_err`	Indicates if the core detects an error in the current byte.
`rx_aux_debug_cha`	Indicates the direction of the current byte. 1 = byte transmitted by the source, 0 = byte received from the sink.

EDID Interface

You can use the Avalon-MM EDID interface to access an on-chip memory region containing the sink's EDID data. The AUX sink controller reads and writes to this memory region according to traffic on the AUX channel.

The Avalon-MM interface uses an 8-bit address with an 8-bit data bus. The interface assumes a read latency of 1.

Note: The IP core does not instantiate this interface if your design uses a controller to control the sink; for instance when you turn on the **Enable GPU control** parameter.

Refer to the *VESA Enhanced Extended Display Identification Data Implementation Guide* for more information.

Debugging Interface

Link Parameters Interface

The sink provides link level data for debugging and configuring external components using the `rx_lane_count` port.

Video Stream Out Interface

This interface provides access to the post-scrambler DisplayPort data, which is useful for low-level debugging source equipment. The 8-bit symbols received are organized as shown in the following tables, where *n* increases with time (at each main link clock cycle, by 2 for dual-symbol mode or by 4 for quad-symbol mode).

Table 5-11: rx_stream_data Dual-Symbol Mode

Bit	Comments
63:56	Lane 3 symbol *n* + 1
55:48	Lane 3 symbol *n*
47:40	Lane 2 symbol *n* + 1
39:32	Lane 2 symbol *n*
31:24	Lane 1 symbol *n* + 1
23:16	Lane 1 symbol *n*
15:8	Lane 0 symbol *n* + 1
7:0	Lane 0 symbol *n*

Table 5-12: rx_stream_data Quad-Symbol Mode

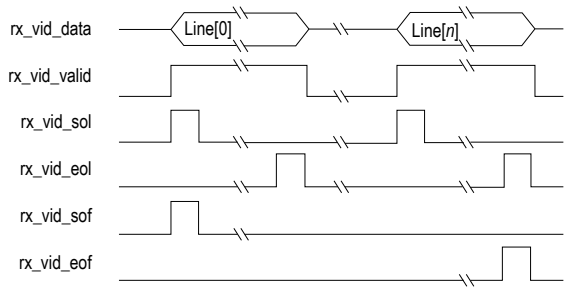
Bit	Comments
127:120	Lane 3 symbol *n* + 3
119:112	Lane 3 symbol *n* + 2
111:104	Lane 3 symbol *n* + 1
103:96	Lane 3 symbol *n*
95:88	Lane 2 symbol *n* + 3
87:80	Lane 2 symbol *n* + 2
79:72	Lane 2 symbol *n* + 1
71:64	Lane 2 symbol *n*
63:56	Lane 1 symbol *n* + 3
55:48	Lane 1 symbol *n* + 2
47:40	Lane 1 symbol *n* + 1
39:32	Lane 1 symbol *n*
31:24	Lane 0 symbol *n* + 3
23:16	Lane 0 symbol *n* + 2
15:8	Lane 0 symbol *n* + 1
7:0	Lane 0 symbol *n*

When data is received, data is output on lane 0, lanes 0 and 1, or on all four lanes according to how many lanes are currently used and link trained on the main link. The IP core provides the data output immediately after the data passes through the descrambler and features all control symbols, data, and original timing. As data is always valid at each and every clock cycle, the `rx_stream_valid` signal remains asserted.

Video Interface

This interface (rx_video_out) allows access to the video data as a non-Avalon-ST stream. You can use this stream to interface with an external pixel clock recovery function. The stream provides synchronization pulses at the start and end of active lines, and at the start and end of active frames.

Figure 5-4: Video Out Image Port Timing Diagram

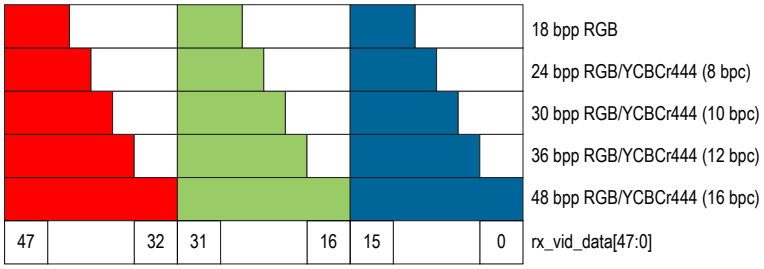


The `rx_vid_overflow` signal is always valid, regardless of the logical state of `rx_vid_valid`. `rx_vid_overflow` is asserted for at least one clock cycle when the sink core internal video data FIFO runs into an overflow condition. This condition can occur when the `rx_vid_clk` frequency is too low to transport the received video data successfully.

You specify the maximum data color depth in the DisplayPort parameter editor. The same output port transfers both RGB and YCrCb data in either 4:4:4 or 4:2:2 color format. Data is most-significant bit aligned and formatted for 4:4:4.

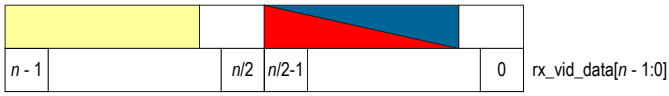
Figure 5-5: Video Output Data Format

18 bpp to 48 bpp Port Width when rx_video_out Port Width is 48 (16 bpc, 1 Pixel per Clock)



The following figure shows the sub-sampled 4:2:2 color format for a video port width of *n*. The most-significant half of the video port always transfers the Y component while the least-significant half of the video port transfers the alternate Cr or Cb component. If the Y/Cb/Cr component widths are less than *n*/2, they are most-significant bit aligned with respect to the *n* and *n*/2-1 boundaries.

Figure 5-6: Sub-Sampled 4:2:2 Color Format Video Port

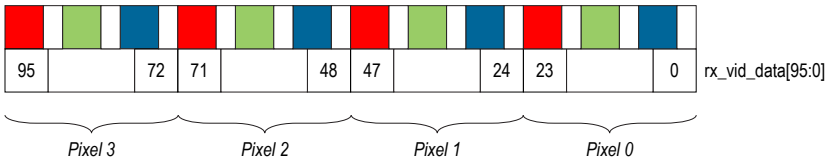


If you set the **Pixel ouput mode** option to **Dual** or **Quad**, the IP core outputs two or four pixels in parallel, respectively. To support video resolutions with horizontal active, front and pack porches with lengths that are not divisible by two or four, `rx_vid_valid` is widened. For example, for two pixels per clock, `rx_vid_valid[0]` is asserted when pixel *N* belongs to active video and `rx_vid_valid[1]` is asserted when pixel *N* + 1 belongs to active video.

The following figure shows the pixel data order from least significant bits to most significant bits.

Figure 5-7: Video Output Alignment

For RGB 18 bpp when rx_video_out Port Width is 96 (8 bpc, 4 Pixels per Clock)



Related Information

Video and Image Processing Suite User Guide

Clocked Video Input Interface

rx_video_out can interface with a clocked video input (CVI). CVI accepts the following video signals with a separate synchronization mode: `datavalid`, `de`, `h_sync`, `v_sync`, `f`, `locked`, and `data`. The DisplayPort rx_video_out interface has the following signals: `rx_vid_valid`, `rx_vid_sol`, `rx_vid_eol`, `rx_vid_sof`, `rx_vid_eof`, `rx_vid_locked`, and `rx_vid_data`. The following table describes how to connect the CVI and DisplayPort sink signals.

Table 5-13: Connecting CVI Signals to DisplayPort Sink Signals

CVI Signal	DisplayPort Sink Signal	Comment
`vid_data`	`rx_vid_data`	Video data
`vid_datavalid`	–	Drive high because the video data is not oversampled.
`vid_f`	–	Drive low because the video data is progressive.
`vid_locked`	`rx_vid_locked`	The core asserts this signal when a stable stream is present.
`vid_de`	`rx_vid_valid`	Indicates the active picture region of a line.
`vid_h_sync`	`rx_vid_eol`	The `rx_vid_eol` signal generates the `vid_h_sync` pulse by delaying it (by 1 clock cycle) to appear in the horizontal blanking period after the active video ends (`rx_vid_valid` is deasserted).
`vid_v_sync`	`rx_vid_eof`	The `rx_vid_eof` signal generates the `vid_v_sync` pulse by delaying it (by 1 clock cycle) to appear in the vertical blanking period after the active video ends (`rx_vid_valid` is deasserted).

Example 5-1: Verilog HDL CVI — DisplayPort Sink Example

// CVI V-sync and H-sync are derived from delayed versions of the eol and eof signals

```
always @ (posedge clk_video)
begin
 rx_vid_h_sync <= rx_vid_eol;
 rx_vid_v_sync <= rx_vid_eof;
end
assign vid_data = rx_vid_data;
assign vid_datavalid = 1'b1;
assign vid_f = 1'b0;
assign vid_locked = rx_vid_locked;
assign vid_de = rx_vid_valid;
assign vid_h_sync = rx_vid_h_sync;
assign vid_v_sync = rx_vid_v_sync;
```

RX Transceiver Interface

The transceiver or Native PHY IP core instance is no longer instantiated within the DisplayPort IP core. The DisplayPort IP uses a soft 8B/10B decoder. This interface receives RX transceiver recovered data (`rx_parallel_data`) in either dual symbol (20-bit) or quad symbol (40-bit) mode, and drives the digital reset (`rx_digitalreset`), analog reset (`rx_analogreset`), and controls the CDR circuitry locking mode.

Transceiver Reconfiguration Interface

You can reconfigure the transceiver to accept double or single reference clock :

- Double reference clocks—162 MHz clock for reduced bit rate (RBR) or 270 MHz clock for high bit rate (HBR or HBR2).

  During run-time, you can reconfigure the transceiver to use either one of the two clocks. You use the Transceiver Reconfiguration Controller to switch between the two reference clocks. To switch them, reconfigure the logical reference clock source for the RX CDR PLLs. The IP core sets `rx_link_rate` to:

  - 00 (RBR)
  - 01 (HBR)
  - 10 (HBR2)

- Single reference clock—135 MHz clock for all bit rates: RBR, HBR, and HBR2.

  During run-time, you can reconfigure the transceiver to operate in either one of the bit rate by changing RX CDR PLLs divider ratio.

When the IP core makes a request, the `rx_reconfig_req` port goes high. The user logic asserts `rx_reconfig_ack`, and then reconfigures the transceiver. During reconfiguration, the user logic holds `rx_reconfig_busy` high. The user logic drives it low when reconfiguration completes.

Note: The transceiver requires a reconfiguration controller. Reset the transceiver to a default state upon power-up.

Related Information

- **Altera Transceiver PHY IP Core User Guide**
  For more information about how to reconfigure the transceiver.
- **AN 645: Dynamic Reconfiguration of PMA Controls in Stratix V Devices**
  For more information about using the Transceiver Reconfiguration Controller to reconfigure the Stratix V Physical Media Attachment (PMA) controls dynamically.
- **AN 676: Using the Transceiver Reconfiguration Controller for Dynamic Reconfiguration in Arria V and Cyclone V Devices**
  For more information about using the Transceiver Reconfiguration Controller to reconfigure the Arria V Physical Media Attachment (PMA) controls dynamically.
- **AN 678: High-Speed Link Tuning Using Signal Conditioning Circuitry**
  For information about link tuning.

Secondary Stream Interface

The secondary streams data can be received via the rx_ss interfaces. The interfaces do not allow for back-pressure and assume the downstream logic can handle complete packets. The rx_ss interface does not distinguish between the types of packets it receives.

The format rx_ss interface output corresponds to four 15-nibble code words as specified by the DisplayPort 1.1a specification section 2.2.6.3. These 15-nibble code words would typically be supplied to the downstream Reed-Solomon decoder. The format differs for both header and payload, as shown in the following figure.

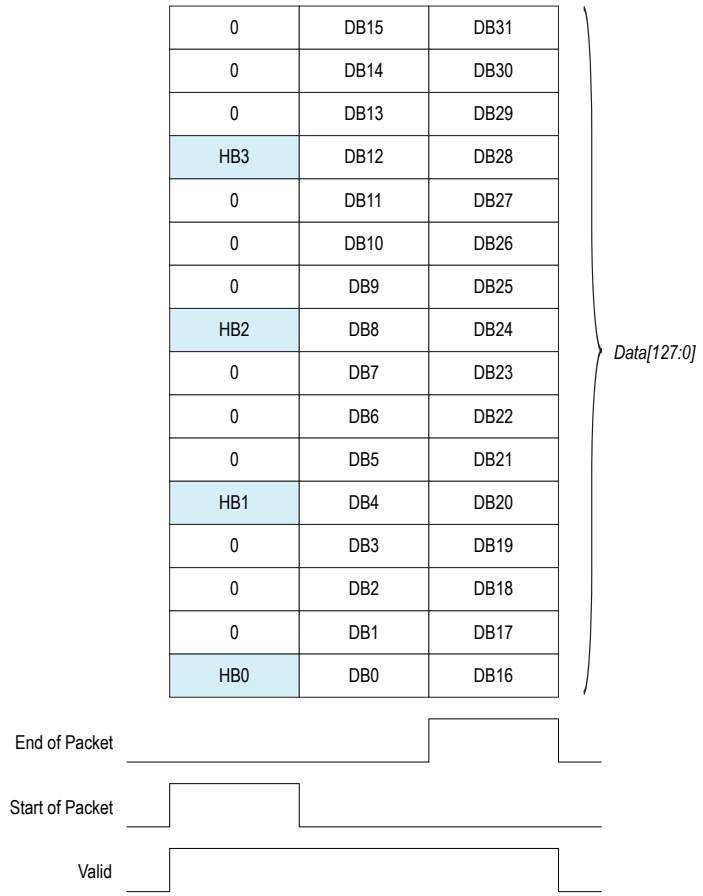
Figure 5-8: rx_ss Input Data Format

15-Nibble Code Word for Packet Payload	15-Nibble Code Word for Packet Header
0	0
0	0
0	0
0	0
0	0
nb0	0
nb1	0
nb2	0
nb3	0
nb4	0
nb5	0
nb6	nb0
nb7	nb1
p0	p0
p1	p1

The following figure shows a typical secondary stream packet with the four byte header (HB0, HB1, HB2, and HB3) and 32-byte payload (DB0, ..., DB31). Each symbol has an associated parity nibble (PB0, ..., PB11). Downstream logic can use the start-of-packet and end-of-packet to determine if the current input is a header or payload symbol.

Data is clocked out of the rx_ss port using the `rx_ss_clk` signal. This signal is the same phase and frequency as the main link lane 0 clock.

Figure 5-9: Typical Secondary Stream Packet



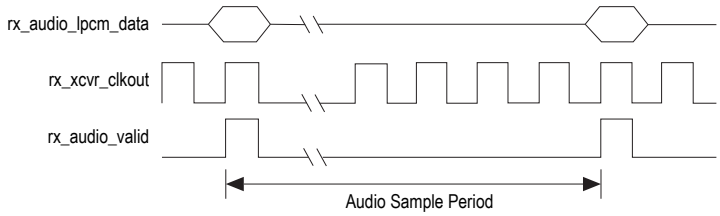
Audio Interface

The audio interfaces are downstream from the secondary stream decoder. They extract and decode the audio infoframe packets, audio timestamp packets, and audio sample data.

The audio timestamp packet payload contains *M* and *N* values, which the sink uses to recover the source's audio sample clock. The `rx_audio` port uses the values to generate the `rx_audio_valid` signal according to sample audio data. Data is clocked out using the `rx_xcvr_clkout` signal. The `rx_xcvr_clkout` signal comes from the rx parallel clock from the RX transceiver. This clock runs at link data rate/20 for dual symbol mode and link data rate/40 for quad symbol mode.

The sink generates the `rx_audio_valid` signal using the *M* and *N* values, and asserts it at the current audio sample clock rate. The `rx_audio_mute` signal indicates whether audio data is present on the DisplayPort interface.

Figure 5-10: rx_audio Data Output



The captured audio infoframe is available on the audio port. The 5-byte port corresponds to the 5 bytes used in the audio infoframe (refer to CEA-861-D). The audio infoframe describes the type of audio content.

MSA Interface

The rx_msa_conduit ports allow designs access to the MSA and VB-ID parameters on a top-level port. The following table shows the 217-bit port bundle assignments. The prefixes `msa` and `vbid` denote parameters from the MSA and VB-ID packets, respectively.

The sink asserts bit `msa_valid` when all `msa_` signals are valid and de-asserted during MSA update. The MSA parameters are assigned to zero when the sink is not receiving valid video data.

The `msa_lock` bit is asserted when the MSA fields have been correctly formatted for the last 15 video frames. Because `msa_lock` changes state only when `msa_valid` = 1, you can use its rising edge to strobe new MSA values following an idle video period (e.g., when the source changes video resolution); you can use its de-asserted state to invalidate received video data.

The sink asserts bit `vbid_strobe` for one clock cycle when the VB-ID is detected and all `vbid_` signals are valid to be read.

Table 5-14: rx_msa_conduit Port Signals

Bit	Signal	Comments
216	`msa_lock`	0 = MSA fields format error. 1 = MSA fields correctly formatted.
215	`vbid_strobe`	0 = VB-ID fields invalid, 1 = VB-ID fields valid.
214:209	`vbid_vbid[5:0]`	VB-ID bit field: • vbid[0] - VerticalBlanking_Flag • vbid[1] - FieldID_Flag (for progressive video, this remains 0) • vbid[2] - Interlace_Flag • vbid[3] - NoVideoStream_Flag • vbid[4] - AudioMute_Flag • vbid[5] - HDCP SYNC DETECT
208:201	`vbid_Mvid[7:0]`	Least significant 8 bits of Mvid for the video stream
200:193	`vbid_Maud[7:0]`	Least significant 8 bits of Maud for the audio stream

Bit	Signal	Comments
192	msa_valid	0 = MSA fields are invalid or being updated, 1 = MSA fields are valid
191:168	msa_Mvid[23:0]	Mvid value for the main video stream. Used for stream clock recovery from link symbol clock.
167:144	msa_Nvid[23:0]	Nvid value for the main video stream. Used for stream clock recovery from link symbol clock.
143:128	msa_Htotal[15:0]	Horizontal total of received video stream in pixels
127:112	msa_Vtotal[15:0]	Vertical total of received video stream in lines
111	msa_HSP	H-sync polarity 0 = Active high, 1 = Active low
110:96	msa_HSW[14:0]	H-sync width in pixel count
95:80	msa_Hstart[15:0]	Horizontal active start from H-sync start in pixels (H-sync width + Horizontal back porch)
79:64	msa_Vstart[15:0]	Vertical active start from V-sync start in lines (V-sync width + Vertical back porch)
63	msa_VSP	V-sync polarity 0 = Active high, 1 = Active low
62:48	msa_VSW[14:0]	V-sync width in lines
47:32	msa_Hwidth[15:0]	Active video width in pixels
31:16	msa_Vheight[15:0]	Active video height in lines
15:8	msa_MISC0[7:0]	The MISC0[7:1] and MISC1[7] fields indicate the color encoding format. The color depth is indicated in MISC0[7:5]: • 000 - 6 bpc • 001 - 8 bpc • 010 - 10 bpc • 011 - 12 bpc • 100 - 16 bpc For details about the encoding format, refer to the DisplayPort v1.2 specification.
7:0	msa_MISC1[7:0]	

Sink Clock Tree

The IP core receives DisplayPort serial data across the high-speed serial interface (HSSI). The HSSI requires a 162 or 270 MHz clock for correct data locking. You can supply these two frequencies to the HSSI using a

reference clock provided by an Altera PLL or pins. Alternatively, you can supply a single reference clock frequency of 135 MHz.

The IP core synchronizes HSSI 20- or 40-bit data to a single HSSI[0] clock that clocks the data into the DisplayPort front-end decoder.

- If you select dual symbol mode, this clock is equal to the link rate divided by 20 (270, 135, or 81 MHz).
- If you turn on quad symbol mode, this clock is equal to the link rate divided by 40 (135, 67.5, or 40.5 MHz).

The IP core crosses the reconstructed pixel data into a local pixel clock (`rx_vid_clk`) through an output DCFIFO, which drives the pixel stream output. The `rx_vid_clk` must be higher than or equal to the pixel clock in the up-stream source. If `rx_vid_clk` is slower than the up-stream pixel clock, the DCFIFO overflows. If the `rx_vid_clk` is faster than the up-stream source pixel clock, the output port experiences a de-assertion of the valid port on cycles in which pixel data is not available. The optimum frequency is the exact clock rate in the up-stream source. However, determining this clock frequency requires pixel clock recovery techniques that are beyond the scope of this document.

Secondary stream data is clocked by `rx_xcvr_clkout`. The sink IP core also requires a 16 MHz clock (`aux_clk`) to drive the internal AUX controller and an Avalon clock for the Avalon-MM interface (`clk`).

Figure 5-11: Sink Clock Tree (Single Reference Clock)

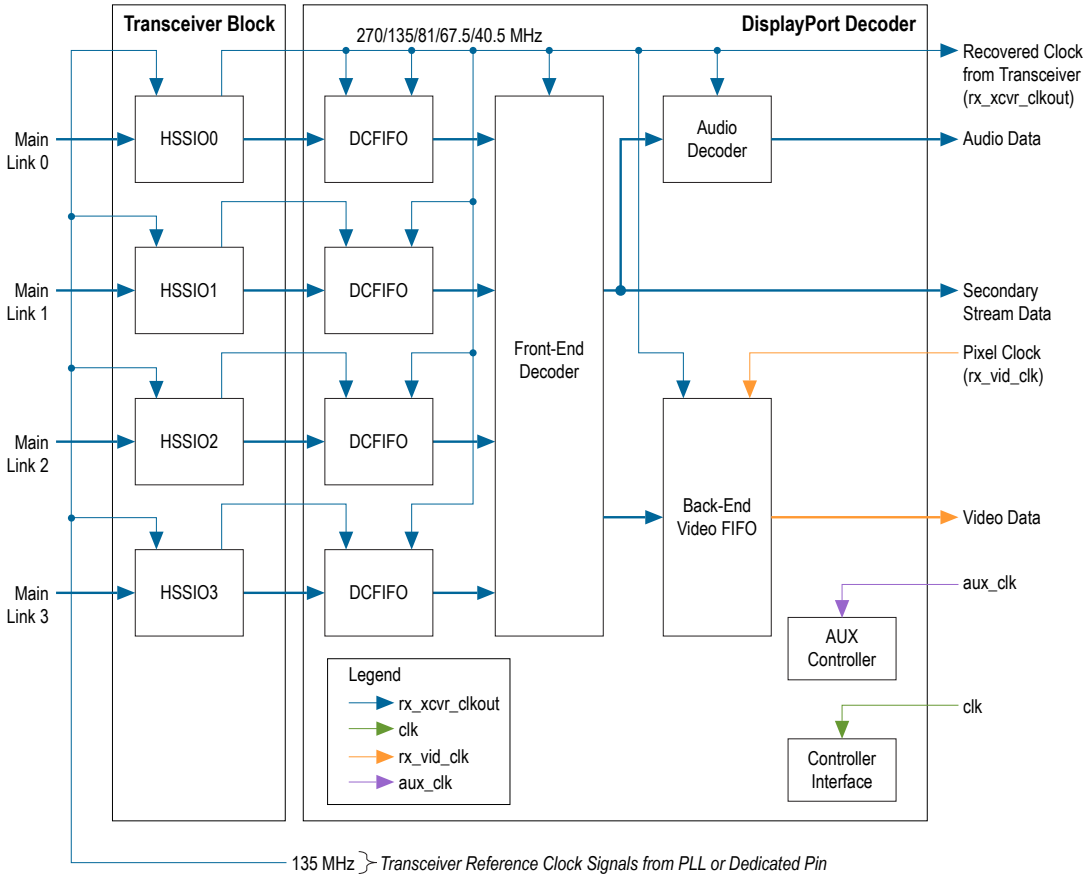
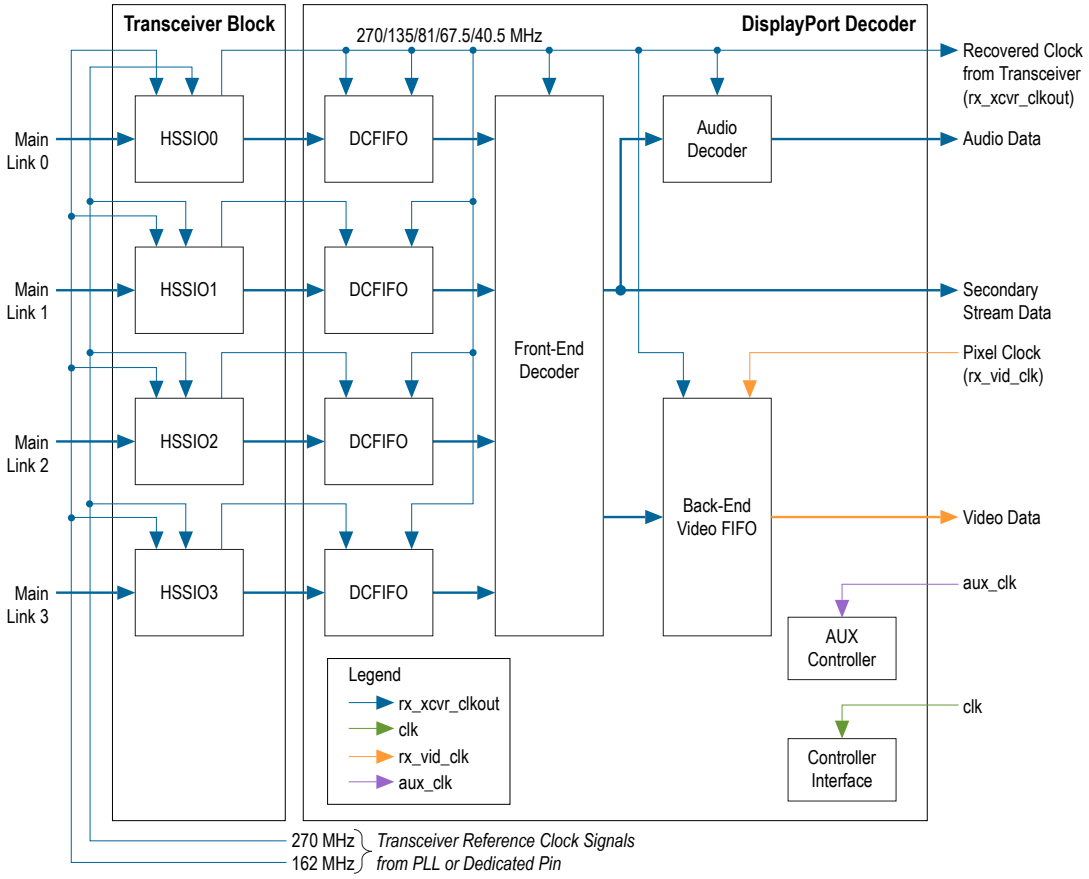


Figure 5-12: Sink Clock Tree (Double Reference Clocks)



DisplayPort IP Core Hardware Demonstration

6

2014.06.30

UG-01131

Subscribe Send Feedback

Introduction

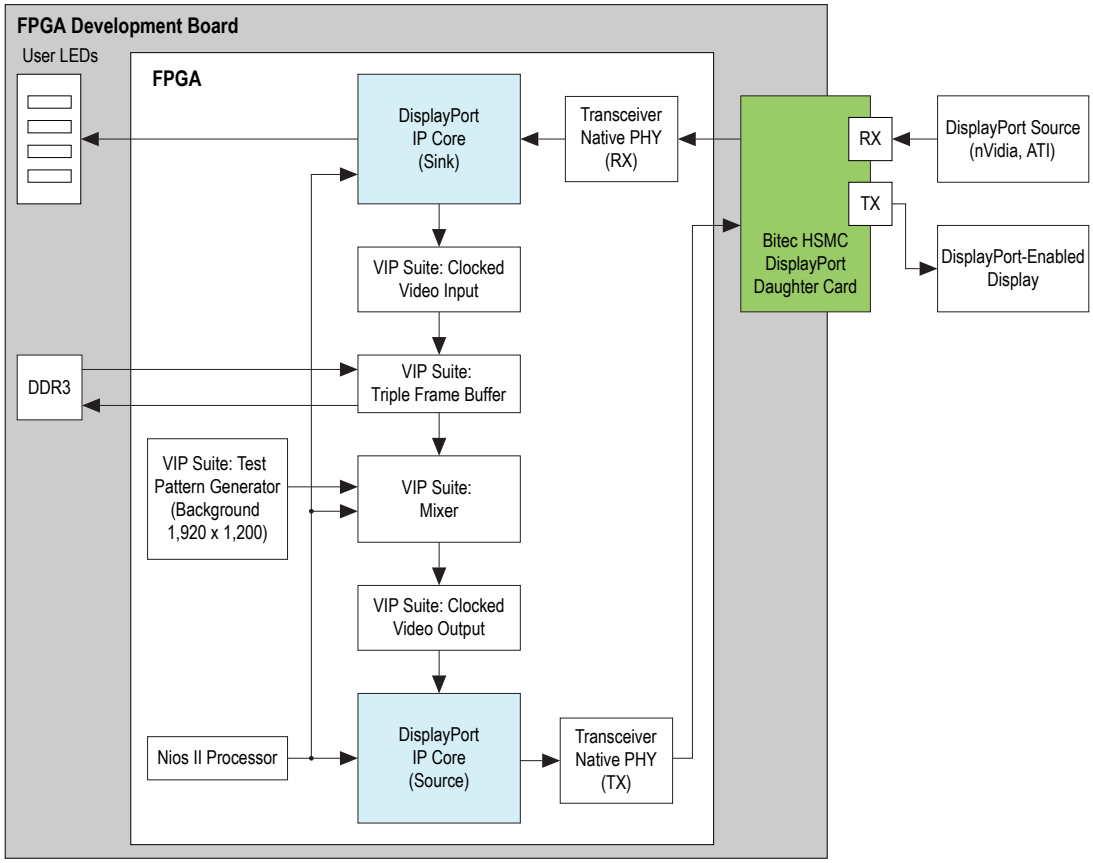
The Altera DisplayPort hardware demonstration evaluates the functionality of the DisplayPort IP core and provides a starting point for you to create your own design. The example design uses a fully functional OpenCore Plus evaluation version, giving you the freedom to explore the core and understand its performance in hardware.

The design performs an SDRAM loop-through for a standard DisplayPort video stream. You connect a DisplayPort-enabled device—such as an nVidia or ATI graphics card transmitter—to the Transceiver Native PHY RX, and the DisplayPort sink input. The DisplayPort sink decodes the port into a standard video stream and triple frame buffers it into external SDRAM. The hardware demonstration mixes a buffered image with a 1,920 x 1,200 color bar image and sends it to the DisplayPort source, and the Transceiver Native PHY TX. The DisplayPort source port of the HSMC daughter card transmits the image to a monitor.

Note: If you use another Altera development board, you must change the device assignments and the pin assignments. You make these changes in the **assignments.tcl** file. This file is described in a later section.

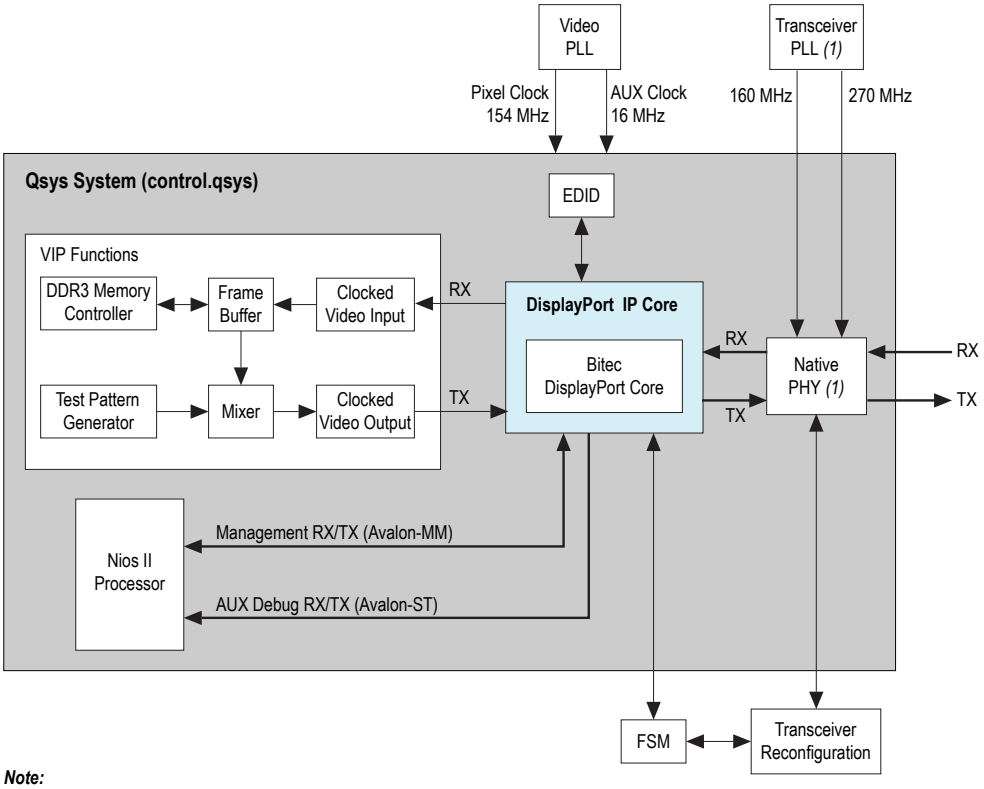
Note: If you use another DisplayPort daughter card, you must change the pin assignments, Qsys system, and software.

Figure 6-1: Hardware Demonstration Overview



The DisplayPort sink uses its internal state machine to negotiate link training upon power up. A Nios II embedded processor performs the source link management; software performs the link training management.

Figure 6-2: Hardware Demonstration Block Diagram



Note:
(1) Alternatively, you can configure the Native PHY to use a single reference clock (135 MHz).

During operation, you can adjust the DisplayPort source resolution (graphics card) from the PC and observe the effect on the IP core. The Nios II software prints the source and sink AUX channel activity. Pressing a push-button prints the current TX and RX main stream attributes (MSA). The development board user LEDs illuminate to indicate the function shown in the following table.

Table 6-1: LED Function

Stratix V LED	Arria V LED	Cyclone V	Function
USER_LED_G0	USER1_LED_G0	USER_LED0	This LED indicates that source has successfully lane trained and is sending video (`rx_vid_locked` drives this LED). The LED turns off if the source is not driving good video.
USER_LED_G6	USER1_LED_G6	USER_LED6	This LED indicates that PLL is locked and stable; if it flickers, the board itself has an issue.

Stratix V LED	Arria V LED	Cyclone V	Function
USER_LED_G7	USER1_LED_G7	USER_LED7	This LED is for the system reset (tracks the reset that waits for PLL lock before releasing). It goes high while the chip is resetting.
USER_LED_G1	USER1_LED_G1	USER_LED1	This LED illuminates for 1-lane designs.
USER_LED_G2	USER1_LED_G2	USER_LED2	This LED illuminates for 2-lane designs.
USER_LED_G3	USER1_LED_G3	USER_LED3	This LED illuminates for 4-lane designs.

Tip: When creating your own design, note the following design tips:

- The Bitec daughter card has inverted transceiver polarity. When creating your own sink (RX) design, use the **Invert transceiver polarity** option to enable or disable inverted polarity.
- The DisplayPort standard reverses the RX and TX transceiver channels to minimize noise for one- or two-lane applications. If you create your own design targeting the Bitec daughter card, ensure that the following signals share the same transceiver channel:
  - `TX0` and `RX3`
  - `TX1` and `RX2`
  - `TX2` and `RX1`
  - `TX3` and `RX0`

Refer to the **assignments.tcl** file for an example of how the channels are assigned in the hardware demonstration.

Transceiver and Clocking

The device's Gigabit transceivers operate at 5.4, 2.7, and 1.62 Gbps and require one of the following reference clocks:

- double reference clocks—270 and 162 MHz. When the link rate changes, a small state machine reconfigures the transceiver to select the appropriate reference clock, and changes the transceiver PLL settings.
- single reference clock—135 MHz. When the link rate changes, the state machine only reconfigures the transceiver PLL settings.

Table 6-2: Arria V Transceiver Native PHY TX and RX Settings

The table shows the Arria V Transceiver Native PHY settings for TX and RX using double and single reference clocks.

Parameters	Settings	
	Double Reference Clocks	Single Reference Clock
Datapath Options		
Enable TX datapath	On	On
Enable RX datapath	On	On
Enable standard PCS	On	On
Number of data channels	1, 2 or 4	1, 2 or 4
	Note: If you select 1 or 2, you must instantiate the PHY instance multiple times for all data channels as per maximum lane count parameter. These values are for non-bonded mode.	
Bonding mode	×1 or ×N	×1* or ×N
	Note: If you select ×1, you must instantiate the PHY instance multiple times for all data channels as per maximum lane count parameter. This value is for non-bonded mode.	
Enable simplified data interface	On	
PMA		
Data rate	2700 Mbps (when TX maximum link rate = 2.7 Gbps)	2700 Mbps (when TX maximum link rate = 2.7 Gbps)
TX local clock division factor	2 (when TX/RX maximum link rate = 2.7Gbps)	2 (when TX/RX maximum link rate = 2.7Gbps)
TX PMA		
Enable TX PLL dynamic reconfiguration	On	On
Number of TX PLLs	1	1
Main TX PLL logical index	0	0
Number of TX PLL reference clock	2	1

TX PLL0		
PLL type	CMU	CMU
Reference clock frequency	270 MHz	135 MHz
Selected reference clock source	1	0
Selected clock network	×1 or ×N	×1 or ×N
	Note: If you select ×1, you must instantiate the PHY instance multiple times for all data channels as per maximum lane count parameter. This value is for non-bonded mode.	
RX PMA		
Enable CDR dynamic reconfiguration	On	On
Number of CDR reference clocks	2	1
Selected CDR reference clock	0	0
Selected CDR reference clock frequency	270 MHz	135 MHz
PPM detector threshold	1000 ppm	1000 ppm
Enable rx_is_lockedtodata port	On	On
Enable rx_is_lockedtoref port	On	On
Enable rx_set_locktodata and rx_set_locktoref ports	On	On
Standard PCS		
Standard PCS protocol mode	Basic	Basic
Standard PCS/PMA interface width	20 (when symbol output mode is dual)	20 (when symbol output mode is dual)
Byte Serializer and Deserializer		
Enable TX byte serializer	Off (when symbol output mode is dual)	Off (when symbol output mode is dual)
Enable RX byte deserializer	Off (when symbol output mode is dual)	Off (when symbol output mode is dual)

Note: Currently, only Arria V GX, Arria V GZ, and Stratix V devices support 5.4 Gbps operation.

Required Hardware

The hardware demonstration requires the following hardware:

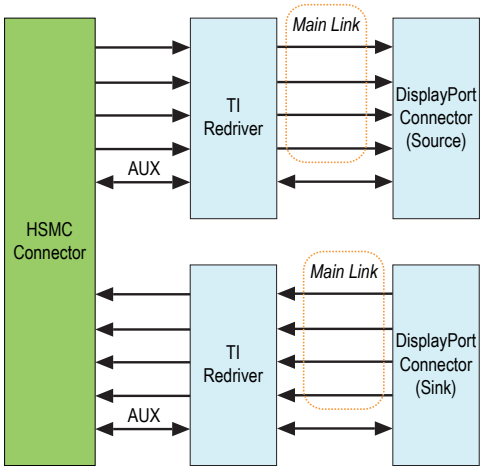
- Altera FPGA kit (includes USB cable to connect the board to your PC); the demonstration supports the following kits:
  - Stratix V GX FPGA Development Kit
  - Arria V GX FPGA Development Kit
  - Arria V GX FPGA Starter Kit
  - Cyclone V GT FPGA Development Kit
- Bitec DisplayPort HSMC daughter card
- PC with a DisplayPort output
- Monitor with a DisplayPort input
- Two DisplayPort cables
  - One cable connects from the graphics card to the FPGA development board
  - The other cable connects from the FPGA development board to the monitor

Note: Altera recommends that you first test the PC and monitor by connecting the PC directly to the monitor to ensure that you have all drivers installed correctly.

The Bitec HSMC DisplayPort daughter card interfaces Altera GX FPGA devices to DisplayPort source and sink devices. High speed 5.4Gbps DisplayPort re-drivers are used on both the source and sink signal paths to improve signal integrity. The re-drivers ensure close PHY layer compatibility at the DP connectors.

Figure 6-3: Bitec HSMC Daughter Card

The figure shows a high level diagram of the Bitec HSMC daughter card.



The following figures illustrate the schematic diagrams of the Bitec HSMC daughter card.

Figure 6-4: HSMC Connector Schematic Diagram

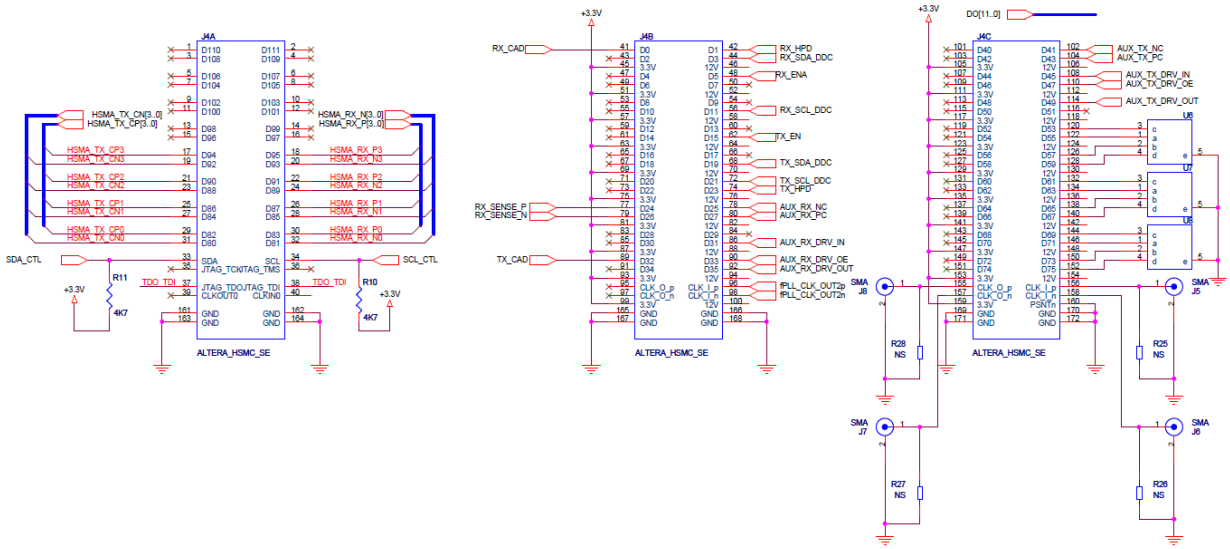


Figure 6-5: TI Re-driver to DP Source Connector Schematic Diagram

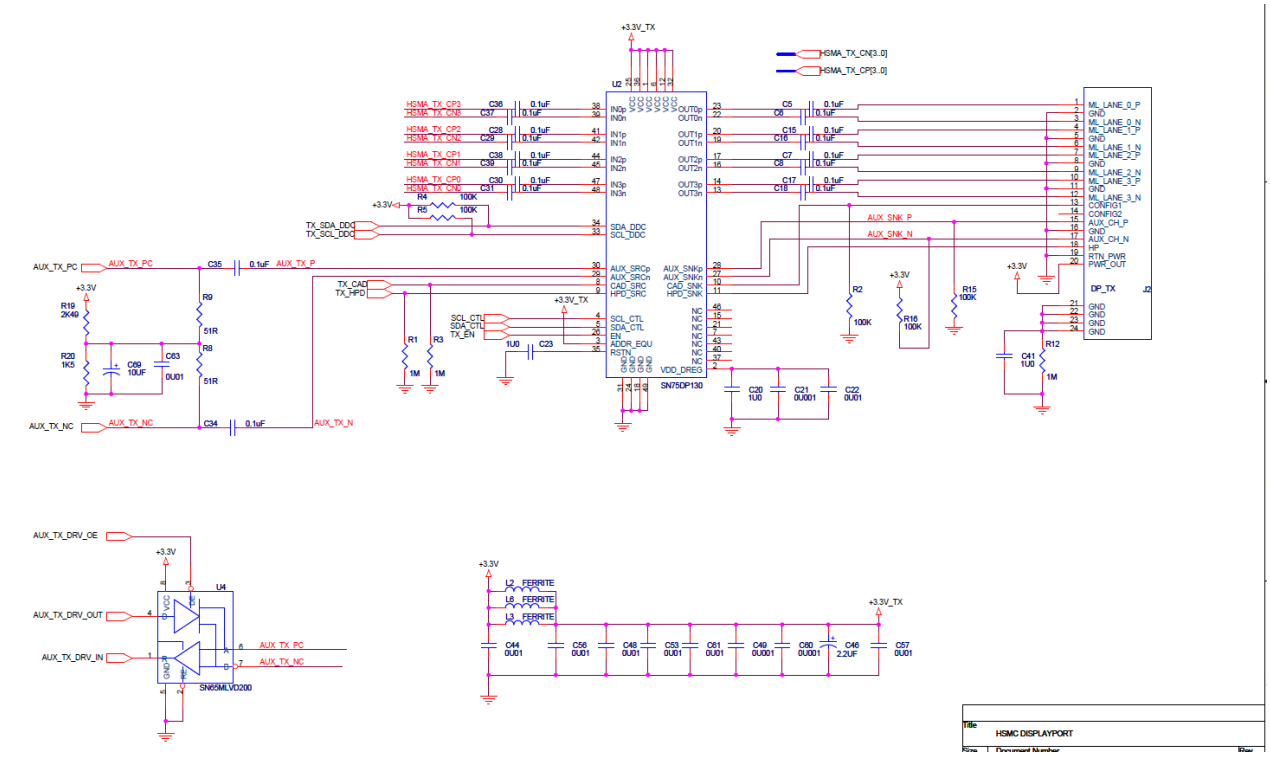


Figure 6-6: DP Sink Connector to TI Re-driver Schematic Diagram

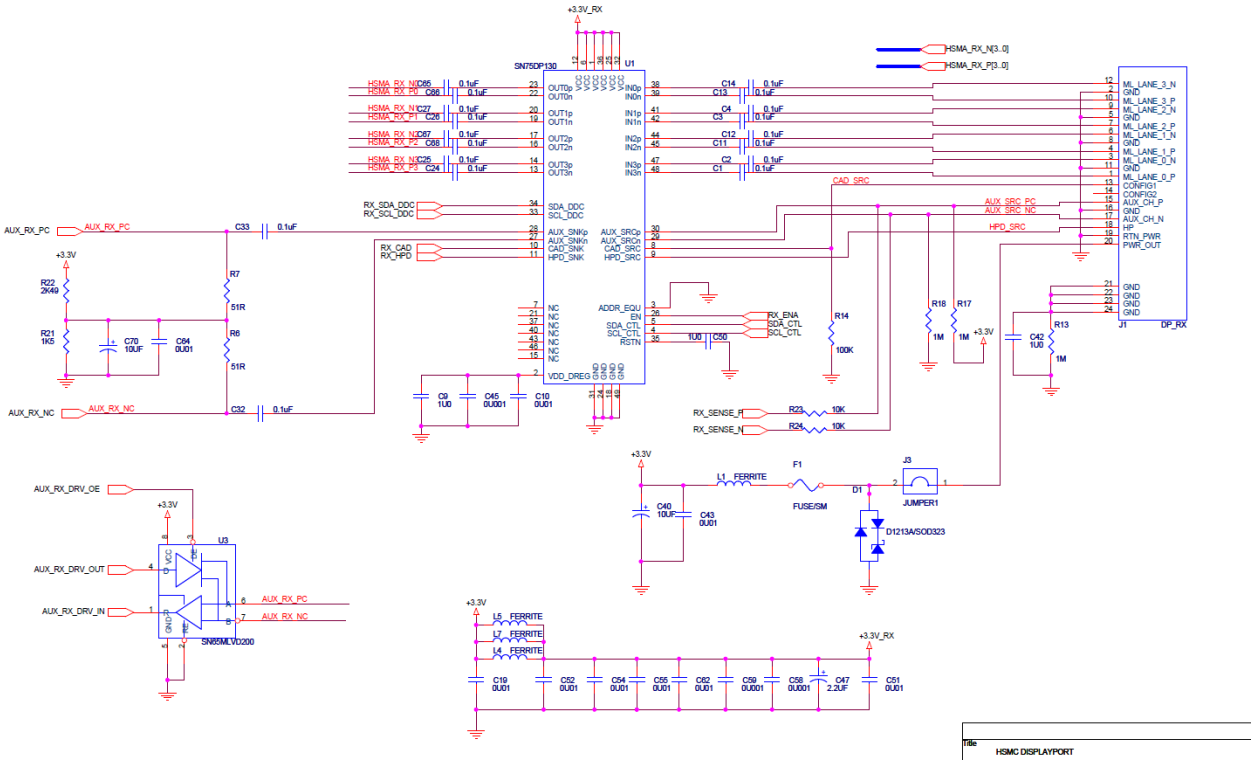
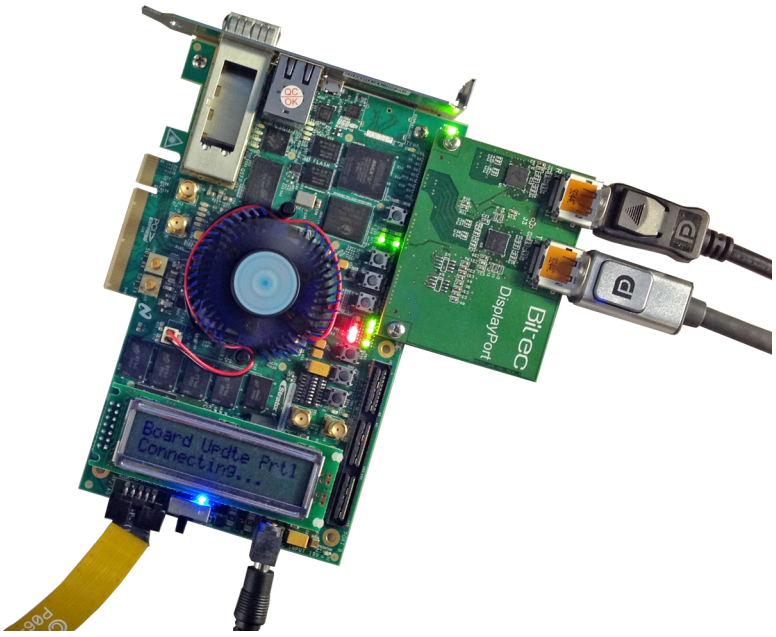


Figure 6-7: Example Hardware Setup

Example hardware setup using the FPGA development board, Bitec daughter card, and cables.



Related Information

- **More information about the Bitec daughter card on the Bitec web site.**
- **Stratix V GX FPGA Development Kit**
- **Arria V GX FPGA Development Kit**
- **Arria V GX FPGA Starter Kit**
- **Cyclone V GT FPGA Development Kit**

Design Walkthrough

Setting up and running the DisplayPort hardware demonstration consists of the following steps. A variety of scripts automate these steps.

1. Set up the hardware.
2. Copy the design files to your working directory.
3. Build the FPGA design.
4. Build the software, download it into the FPGA, and run the software.
5. Power-up the DisplayPort monitor and view the results.

Set Up the Hardware

Set up the hardware using the following steps:

1. Connect the Bitec daughter card to the FPGA development board.
2. Connect the development board to your PC using a USB cable.

   Note: The FPGA development board has an On-Board USB-Blaster™ II connection. If your version of the board does not have this connection, you can use an external USB-Blaster cable. Refer to the documentation for your board for more information.

3. Connect a DisplayPort cable from the DisplayPort TX on the Bitec HSMC daughter card to a DisplayPort monitor (do not power up the monitor).
4. Power-up the development board.
5. Connect one end of a DisplayPort cable to your PC (do not connect the other end to anything).

Copy the Design Files to Your Working Directory

In this step, you copy the hardware demonstration files to your working directory. Copy the files using the command:

`cp -r` *<IP root directory>*`/altera/altera_dp/hw_demo/`*<device_board>* *<working directory>*

where *<device_board>* is **av** for Arria V GX development kit, **av_sk** for Arria V GX starter kit, **cv** for Cyclone V GT development kit, and **sv** for Stratix V development kit.

Your working directory should contain the files shown in the following table.

Table 6-3: Hardware Demonstration Files

Files are named with <*prefix*>_<*name*>.<*extension*> where <*prefix*> represents the device (**av** for Arria V devices, **cv** for Cyclone V devices, and **sv** for Stratix V devices).

File Type	File	Description
Verilog HDL design files	<*prefix*>_ **dp_demo.v**	Top-level design file.
	dp_mif_mappings.v	Table translating MIF mappings for transceiver reconfiguration. This module is instantiated in **reconfig_mgmt_hw_ctrl.v**. It maps the 2-bit requested data rates to the Memory Initialization File (**.mif**) settings that need to be written during a direct reconfiguration mode of the transceiver reconfiguration controller IP core.
	dp_analog_mappings.v	Table translating VOD and pre-emphasis settings. This module is instantiated in **reconfig_mgmt_hw_ctrl.v**. It maps per-channel 2-bit VOD and 2-bit pre-emphasis settings from the DisplayPort source to the transceiver analog settings. Note: To enable analog reconfiguration, turn on the **Support analog reconfiguration** option in the DisplayPort source parameter editor. By default, the analog reconfiguration is turned off. If you are not using an external re-driver solution, turn on this option.
	reconfig_mgmt_hw_ctrl.v	Reconfiguration manager top-level. This module is a high-level FSM that generates the control signals to reconfigure the VOD and pre-emphasis, selects the PLL reference clock, and reconfigures clock divider setting. It loops through all the channels and transceiver settings.
	reconfig_mgmt_write.v	Reconfiguration manager FSM for a single write command. This module is instantiated in the **reconfig_mgmt_hw_ctrl.v**. It generates a reconfiguration write cycle on the Avalon-MM interface to the transceiver reconfiguration controller IP core. This action is performed with a simple state machine that steps through the low-level commands to write to the transceiver reconfiguration controller IP core.

File Type	File	Description
IP Catalog files	<*prefix*>_ **video_pll.v** <*prefix*>_ **xcvr_pll.v** <*prefix*>_ **aux_buffer.v** <*prefix*>_ **xcvr_reconfig.v** <*prefix*>_ **native_phy_rx.v** <*prefix*>_ **native_phy_tx.v**	IP Catalog variants for the various helper IP cores.
Qsys system	<*prefix*>**control.qsys**	Qsys system file.
Scripts	**runall.tcl**	Script to set up the project, generate the IP and Qsys system, and compile.
	assignments.tcl	Top-level TCL file to create the project assignments.
Miscellaneous	<*prefix*>_ **dp_demo.sdc**	Top-level SDC file.
	edid_memory.hex	Initial content for the EDID ROM.
Software files (in the software directory)	**batch_script.sh**	Master script to program the device and build/run the software.
	rerun.sh	Script to rerun the software without rebuilding.
	dp_demo_src	Directory containing the example application source code.
	btc_dprx_syslib	System library for the RX API.
	btc_dptx_syslib	System library for the TX API.

Build the FPGA Design

In this step you use a script to build and compile the FPGA design. Type the command:

```
quartus_sh -t runall.tcl
```

This script executes the following commands where <*prefix*> is **av** for Arria V devices, **cv** for Cyclone V devices, and **sv** for Stratix V devices:

- Load required packages :
  - load_package flow
  - load_package misc

- Regenerate the IP :
  - qexec "qmegawiz -silent <*prefix*>_video_pll.v"
  - qexec "qmegawiz -silent <*prefix*>_xcvr_pll.v"
  - qexec "qmegawiz -silent <*prefix*>_aux_buffer.v"
  - qexec "qmegawiz -silent <*prefix*>_xcvr_reconfig.v"
  - qexec "qmegawiz -silent <*prefix*>_native_phy_rx.v"
  - qexec "qmegawiz -silent <*prefix*>_native_phy_tx.v"
- Regenerate the Qsys system :
  - qexec "ip-generate --project-directory=./ \
  - --output-directory=./<*prefix*>_control/synthesis/ \
  - --file-set=QUARTUS_SYNTH \
  - --report-file=sopcinfo:./<*prefix*>_control.sopcinfo \
  - --report-file=html:./<*prefix*>_control.html \
  - --report-file=qip:./<*prefix*>_control/synthesis/<*prefix*>_control.qip \
  - --component-file=./<*prefix*>_control.qsys"
- Create the project, overwriting any previous settings files :
  - project_new <*prefix*>_dp_demo -overwrite
- Add the assignments to the project:
  - source assignments.tcl
- Run quartus_map to generate a netlist for the DDR pin assignments script:
  - execute_module -tool map
- Close the project before running the pin assignment script:
  - projectlose
- Run the DDR pin assignments script generated by Qsys:
  - qexec "quartus_sta -t ./<*prefix*>ontrol/synthesis/submodules/<*prefix*>_control_ddr_p0_pin_assignments.tcl <*prefix*>_dp_demo"
- Re-open the project and do a full compile:
  - project_open <*prefix*>_dp_demo
- Compile the project:
  - execute_flow -compile
- Clean up by closing the project:
  - projectlose

Build, Load, and Run the Software

In this step you build the software, load it into the device, and run the software.

1. In a Windows Command Prompt, navigate to the hardware demonstration **software** directory.

2. Launch a Nios II command shell. You can launch it using several methods, for example, from the Windows task bar or within the Qsys system. To launch a Nios II command shell from the Windows Command Prompt, type the command:

```
start "" %SOPC_KIT_NIOS2%\"Nios II Command Shell.bat"
```

3. From within the Nios II command shell execute the following command to build the software, program the device, download the Nios II program, and launch a debug terminal:

```
./batch_script.sh <USB cable number>
```

Note: To find <*USB cable number*>, use the `jtagconfig` command.

The script also creates the **dp_demo**, and **dp_demo_bsp** subdirectories inside the software directory.

If you have already built the software, use the **rerun.sh** script to program the device, download the Nios II program, and launch the terminal:

```
./rerun.sh
```

Refer to *Chapter 15: Nios II Software Build Tools Reference* in the Nios II Software Developer's Handbook for a description of the commands in these scripts.

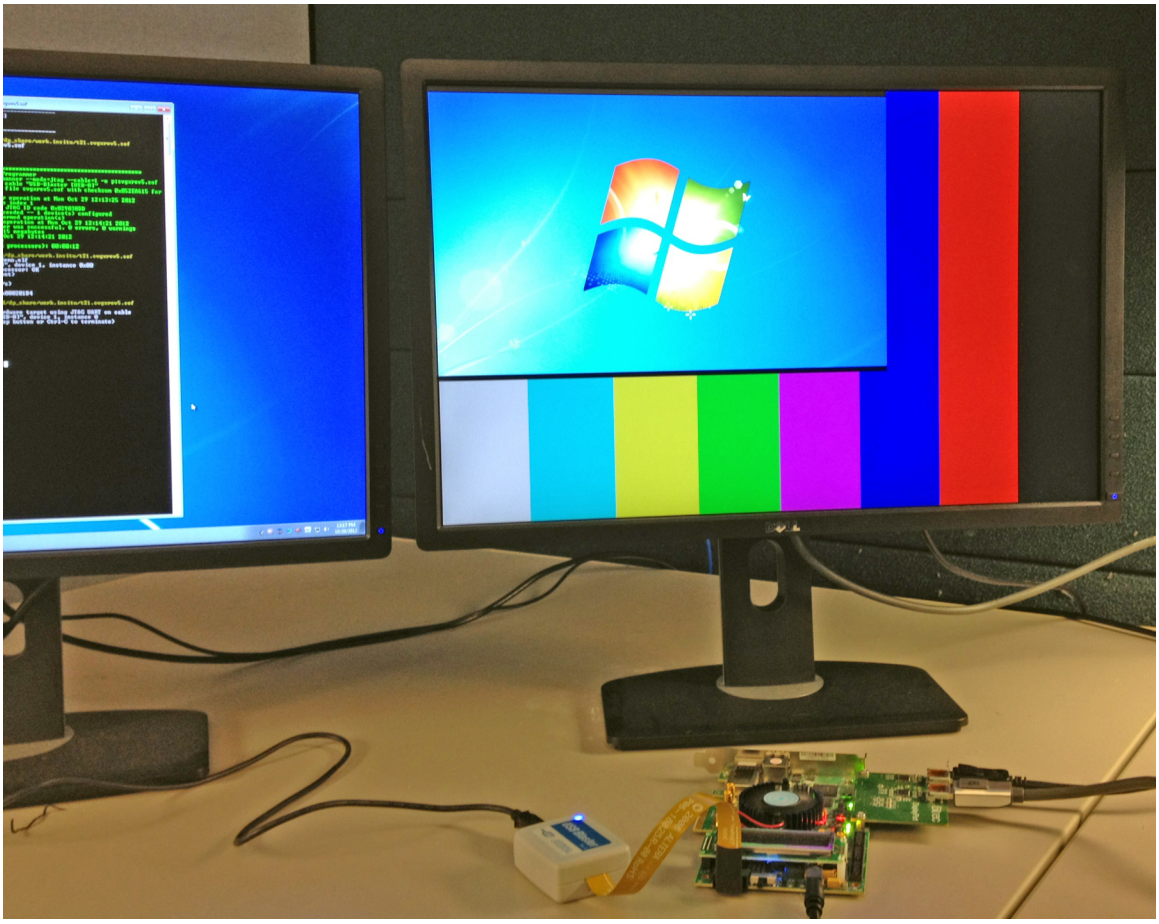
View the Results

In this step you view the results of the hardware demonstration in the Nios II command shell and on the DisplayPort monitor.

1. Power-up the connected DisplayPort monitor. The hardware demonstration displays a VIP test pattern (color bars).
2. Connect the free end of the Display Port cable that you connected to your PC to the DisplayPort RX on the Bitec HSMC daughter card. The PC now has the DisplayPort monitor available as a second monitor. Depending on the resolution set in the PC and the resolution supported by your monitor, the hardware demonstration displays a VIP test pattern (color bars) mixed with your graphic card output as shown in the following figure. The PC resolution was verified up to 1920×1080.

Note: Some PC drivers and graphic card adapters do not enable the DisplayPort hardware automatically upon hot plug detection. You may need to start the adapter's control utility (e.g., Catalist Control Center, nVidia Control Panel, etc.) and manually enable the DisplayPort display.

Figure 6-8: Color Bars Mixed with Graphic Card Output

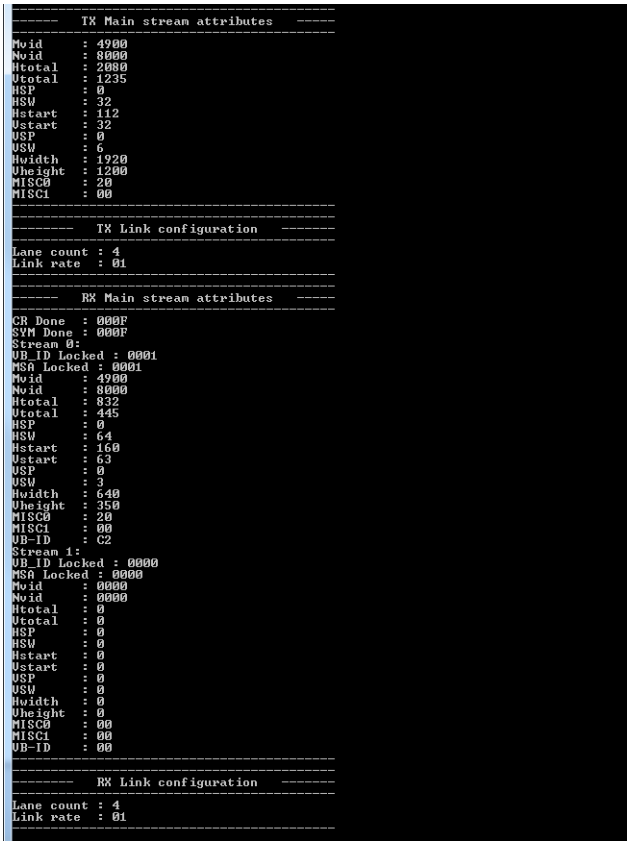


3. Open your graphic card adapter's control utility; it shows a monitor named BITECDP01. Using the control panel, you can adjust the resolution of the BITECDP01 monitor, which typically results in link training, related AUX channel traffic, and a corresponding new image size on the monitor.

   Note: If you do not see visible output on the monitor, press pushbutton 2 (USER1_PB2 for Arria V, USER_PB1 for Cyclone V, or USER_PB2 for Stratix V) to generate a reset, causing the DisplayPort TX core to re-train the link.

   Pressing pushbutton 0 (USER1_PB0 for Arria V, USER_PB2 for Cyclone V, or USER_PB0 for Stratix V) retrieves MSA statistics from the source and sink connections. The Nios II Command Shell displays the AUX channel traffic during link training with the monitor.

Figure 6-9: MSA Output



The Nios II AUX printout shows each message packet on a separate line.

- The first field is the incremental timestamp in microseconds.
- The second field indicates whether the message packet is from or to the DisplayPort sink IP core (SNK) or DisplayPort source IP core (SRC).
- The following two fields show the request and response headers and payloads. The DPCD address field on request messages are decoded into their respective DPCD location names.

When connected and enabled, USER1_LED_G0 (Arria V), USER_LED0 (Cyclone V), or USER_LED_G0 (Stratix V) on the development board illuminates to indicate that the DisplayPort receiver has locked correctly. The Nios II terminal also displays the AUX channel traffic related to link training between the graphics adapter and the DisplayPort sink.

Figure 6-10: Typical Sink Link Training Output

```
00000192 [SRC] Req sent   AUX_WR @ 0102 (TRAINING_PATTERN_SET) 80 01 02 04 21 00
 00 00 00
00000551 [SRC] Reply got  AUX_ACK 00
00000162 [SRC] Req sent   AUX_RD @ 0202 (LANE0_1_STATUS) 90 02 02 01
00000153 [SRC] Reply got  AUX_ACK 00 00 00
00000202 [SRC] Req sent   AUX_RD @ 0206 (ADJUST_REQUEST_LANE0_1) 90 02 06 01
00000153 [SRC] Reply got  AUX_ACK 00 CC CC
00000270 [SRC] Req sent   AUX_WR @ 0103 (TRAINING_LANE0_SET) 80 01 03 03 38 38 3
8 38
00000495 [SRC] Reply got  AUX_ACK 00
00000364 [SRC] Req sent   AUX_RD @ 0202 (LANE0_1_STATUS) 90 02 02 01
00000154 [SRC] Reply got  AUX_ACK 00 11 11
00000202 [SRC] Req sent   AUX_RD @ 0206 (ADJUST_REQUEST_LANE0_1) 90 02 06 01
00000153 [SRC] Reply got  AUX_ACK 00 CC CC
00000223 [SRC] Req sent   AUX_WR @ 0102 (TRAINING_PATTERN_SET) 80 01 02 00 22
00000169 [SRC] Reply got  AUX_ACK 00
00000158 [SRC] Req sent   AUX_RD @ 0206 (ADJUST_REQUEST_LANE0_1) 90 02 06 01
00000154 [SRC] Reply got  AUX_ACK 00 CC CC
00000255 [SRC] Req sent   AUX_WR @ 0103 (TRAINING_LANE0_SET) 80 01 03 03 38 38 3
8 38
00000440 [SRC] Reply got  AUX_ACK 00
00000161 [SRC] Req sent   AUX_RD @ 0202 (LANE0_1_STATUS) 90 02 02 01
00000154 [SRC] Reply got  AUX_ACK 00 11 11
00000202 [SRC] Req sent   AUX_RD @ 0206 (ADJUST_REQUEST_LANE0_1) 90 02 06 01
00000154 [SRC] Reply got  AUX_ACK 00 88 88
00000262 [SRC] Req sent   AUX_WR @ 0103 (TRAINING_LANE0_SET) 80 01 03 03 10 10 1
0 10
00000439 [SRC] Reply got  AUX_ACK 00
00000969 [SRC] Req sent   AUX_RD @ 0202 (LANE0_1_STATUS) 90 02 02 01
00000153 [SRC] Reply got  AUX_ACK 00 11 11
00000203 [SRC] Req sent   AUX_RD @ 0206 (ADJUST_REQUEST_LANE0_1) 90 02 06 01
00000154 [SRC] Reply got  AUX_ACK 00 44 44
00000264 [SRC] Req sent   AUX_WR @ 0103 (TRAINING_LANE0_SET) 80 01 03 03 08 08 0
8 08
00000440 [SRC] Reply got  AUX_ACK 00
00000970 [SRC] Req sent   AUX_RD @ 0202 (LANE0_1_STATUS) 90 02 02 01
00000154 [SRC] Reply got  AUX_ACK 00 11 11
00000202 [SRC] Req sent   AUX_RD @ 0206 (ADJUST_REQUEST_LANE0_1) 90 02 06 01
00000153 [SRC] Reply got  AUX_ACK 00 00 00
00000264 [SRC] Req sent   AUX_WR @ 0103 (TRAINING_LANE0_SET) 80 01 03 03 00 00 0
0 00
00000470 [SRC] Reply got  AUX_ACK 00
00000967 [SRC] Req sent   AUX_RD @ 0202 (LANE0_1_STATUS) 90 02 02 01
00000154 [SRC] Reply got  AUX_ACK 00 11 11
00000201 [SRC] Req sent   AUX_RD @ 0206 (ADJUST_REQUEST_LANE0_1) 90 02 06 01
00000153 [SRC] Reply got  AUX_ACK 00 44 44
00000265 [SRC] Req sent   AUX_WR @ 0103 (TRAINING_LANE0_SET) 80 01 03 03 08 08 0
8 08
00000472 [SRC] Reply got  AUX_ACK 00
00000969 [SRC] Req sent   AUX_RD @ 0202 (LANE0_1_STATUS) 90 02 02 01
00000154 [SRC] Reply got  AUX_ACK 00 77 77
00000201 [SRC] Req sent   AUX_RD @ 0206 (ADJUST_REQUEST_LANE0_1) 90 02 06 01
00000154 [SRC] Reply got  AUX_ACK 00 44 44
00000224 [SRC] Req sent   AUX_WR @ 0102 (TRAINING_PATTERN_SET) 80 01 02 00 00
00000170 [SRC] Reply got  AUX_ACK 00
00000000 [SNK] Req got    AUX_RD @ 010A ( ) 90 01 0A 00
00000170 [SNK] Reply sent AUX_ACK 00 00
00001500 [SNK] Req got    AUX_RD @ 000D ( ) 90 00 0D 00
00000170 [SNK] Reply sent AUX_ACK 00 00
00001645 [SNK] Req got    AUX_RD @ 0002 (MAX_LANE_COUNT) 90 00 02 00
00000170 [SNK] Reply sent AUX_ACK 00 84
00001598 [SNK] Req got    AUX_WR @ 010A ( ) 80 01 0A 00 00
00000186 [SNK] Reply sent AUX_ACK 00
00001725 [SNK] Req got    AUX_WR @ 0100 (LINK_BW_SET) 80 01 00 00 0A
00000186 [SNK] Reply sent AUX_ACK 00
00001459 [SNK] Req got    AUX_WR @ 0101 (LANE_COUNT_SET) 80 01 01 00 84
00000186 [SNK] Reply sent AUX_ACK 00
00001945 [SNK] Req got    AUX_WR @ 0102 (TRAINING_PATTERN_SET) 80 01 02 04 21 00
 00 00 00
00000250 [SNK] Reply sent AUX_ACK 00
00001848 [SNK] Req got    AUX_RD @ 0202 (LANE0_1_STATUS) 90 02 02 05
00000170 [SNK] Reply sent AUX_ACK 00 11 11 80 00 00 00
00002263 [SNK] Req got    AUX_WR @ 0102 (TRAINING_PATTERN_SET) 80 01 02 04 22 00
 00 00 00
```

DisplayPort IP Core Simulation Example 7

2014.06.30

UG-01131 Subscribe Send Feedback

Introduction

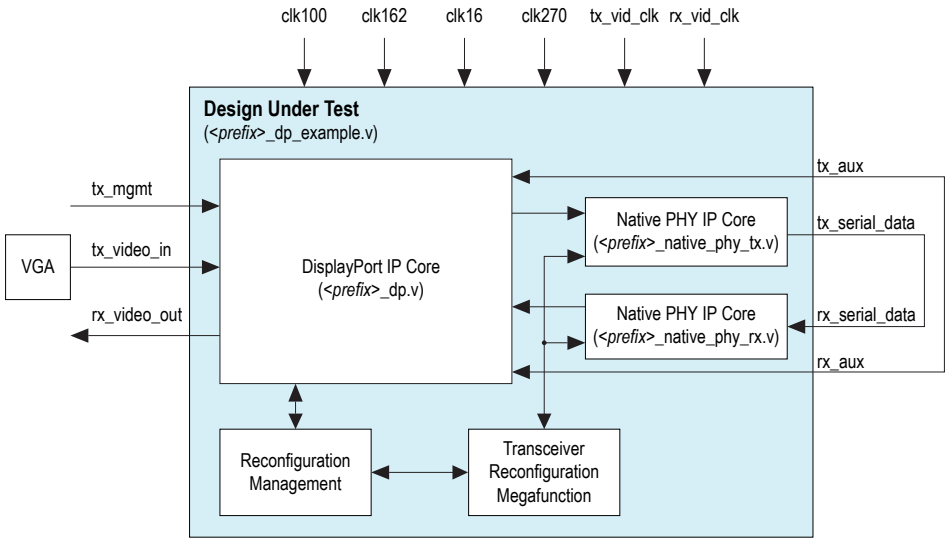
The Altera DisplayPort simulation example allows you to evaluate the functionality of the DisplayPort IP Core and provides a starting point for you to create your own simulation. This example targets the ModelSim SE simulator.

The simulation example instantiates the DisplayPort IP core with default settings, TX and RX enabled, and 8 bpc. The core has the **Support CTS test automation** option turned on, which is required for the simulation to pass.

The test harness instantiates the design under test (DUT) and a VGA driver. It also generates the clocks and top-level stimulus. The design manipulates the tx_mgmt interface in the main loop to establish a link and send several frames of video data. The test harness checks that the sent data's CRC matches the received data's CRC for three frames.

Figure 7-1: Simulation Example Block Diagram

Files are named <*prefix*>_<*name*>.<*extension*> where <*prefix*> represents the device (**sv** for Stratix V devices and **av** for Arria V devices).



Design Walkthrough

Setting up and running the DisplayPort simulation example consists of the following steps:

1. Copy the simulation files to your target directory.
2. Generate the IP simulation files and scripts, and compile and simulate.
3. View the results.

You use a script to automate these steps.

Copy the Simulation Files to Your Working Directory

Copy the simulation example files to your working directory using the command:

`cp -r` *<IP root directory>*`/altera/altera_dp/sim_example/<`*device*`>` *<working directory>*

where *<device>* is **av** for Arria V devices, **cv** for Cyclone V devices, and **sv** for Stratix V devices.

Your working directory should contain the files shown below.

Table 7-1: Simulation Example Files

Files are named *<prefix>*_*<name>*.*<extension>* where *<prefix>* represents the device (**av** for Arria V devices, **cv** for Cyclone V devices, and **sv** for Stratix V devices).

File Type	File	Description
System Verilog HDL design files	*<prefix>*_**dp_harness.sv**	Top-level test harness.
Verilog HDL design files	*<prefix>*_**dp_example.v**	Design under test (DUT).
	dp_mif_mappings.v	Table translating MIF mappings for transceiver reconfiguration.
	dp_analog_mappings.v	Table translating VOD and pre-emphasis settings.
	reconfig_mgmt_hw_ctrl.v	Reconfiguration manager top-level.
	reconfig_mgmt_write.v	Reconfiguration manager FSM for a single write command.
	clk_gen.v	Clock generation file.
	freq_check.sv	Top-level file for the frequency checker.
	rx_freq_check.sv	RX frequency checker.
	tx_freq_check.sv	TX frequency checker.
	vga_driver.v	VGA driver (generates a test image).

File Type	File	Description
IP Catalog files	<*prefix*>_ **dp.v**	IP Catalog variant for the DisplayPort IP Core.
	<*prefix*>_ **xcvr_reconfig.v**	IP Catalog variant for the transceiver reconfiguration core.
	<*prefix*>_ **native_phy_rx.v**	IP Catalog variant for the RX transceiver.
	<*prefix*>_ **native_phy_tx.v**	IP Catalog variant for the TX transceiver.
Scripts	**runall.sh**	This script generates the IP simulation files and scripts, and compiles and simulates them.
	msim_dp.tcl	Compiles and simulates the design in the ModelSim software.
Waveform **.do** files	**all.do**	Waveform that shows a combination of all waveforms.
	reconfig.do	Waveform that shows the signals involved in reconfiguring the transceiver.
	rx_video_out.do	Waveform that shows the rx_video_out signals from the DisplayPort core mapped to CVI input.
	tx_video_in.do	Waveform that shows the `tx_vid_v_sync`, `tx_vid_h_sync`, `de`, `tx_vid_de`, `tx_vid_f`, and `tx_vid_data[23:0]` signals at 256 pixels per line and 8 bpp, i.
Miscellaneous files	**readme.txt**	Documentation for the simulation example.
	edid_memory.hex	Initial content for the EDID ROM.

Generate the IP Simulation Files and Scripts, and Compile and Simulate

In this step you use a script to generate the IP simulation files and scripts, and compile and simulate them. Type the command:

```
sh runall.sh
```

This script executes the following commands (where <*prefix*> is **av** for Arria V devices, **cv** for Cyclone V devices, and **sv** for Stratix V devices):

- Generate the simulation files for the DisplayPort, transceivers, and transceiver reconfiguration IP cores:
  - `qmegawiz -silent <prefix>_xcvr_reconfig.v`
  - `qmegawiz -silent <prefix>_dp.v`
  - `qmegawiz -silent <prefix>_native_phy_rx.v`
  - `qmegawiz -silent <prefix>_native_phy_tx.v`
- Merge the four resulting **msim_setup.tcl** scripts to create a single **mentor/msim_setup.tcl**:
  - `ip-make-simscript --spd=./<prefix>_xcvr_reconfig.spd --spd=./<prefix>_dp.spd --spd=./<prefix>_native_phy_rx.spd --spd=./<prefix>_native_phy_tx.spd`
- Compile and simulate the design in the ModelSim software:
  - `vsim -c -do msim_dp.tcl`

The simulation sends several frames of video after reconfiguring the DisplayPort source (TX) and sink (RX) to use the HBR (2.7 G) rate. A successful result is seen by the CTS test automation logic's CRC checks. These checks compare the CRC of the transmitted image with the result measured at the sink. The result is successful if the sink detects three matching frames.

Example 7-1: Example Successful Result

```
# Testing Link HBR Rate Training Pattern 1
# Testing Video Input Frame Number = 00
# Testing Link HBR Rate Training Pattern 2
# TX Frequency Change Detected, Measured Frequency = 135 MHz
# RX Frequency Change Detected, Measured Frequency = 135 MHz
# ...
# SINK CRC_R = 9b40, CRC_G = 9b40, CRC_B = 9b40,
# SOURCE CRC_R = 9b40, CRC_G = 9b40, CRC_B = 9b40,
# Pass: Test Completed
```

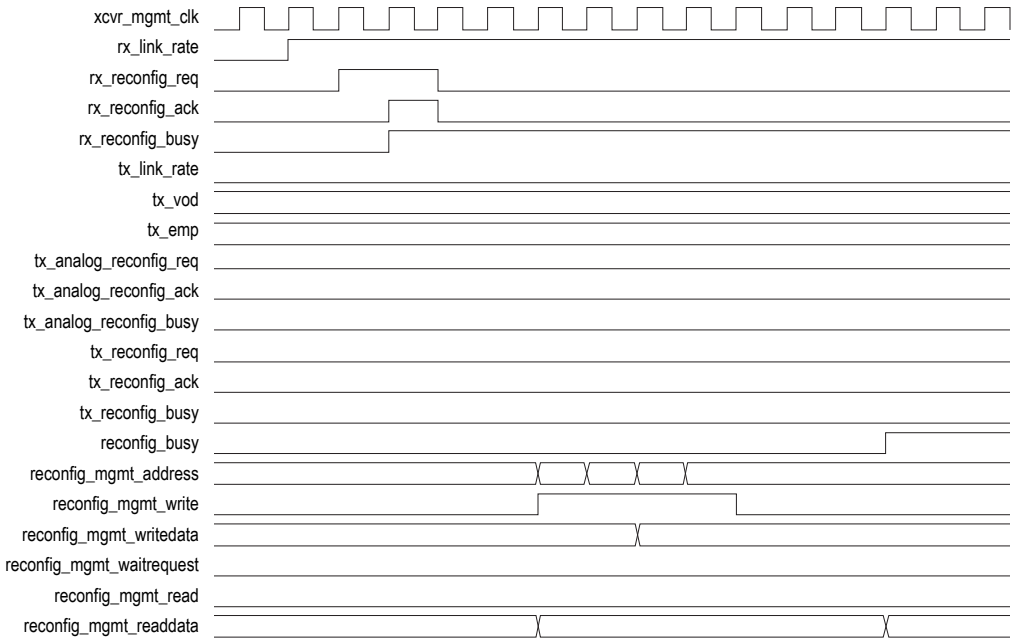
View the Results

You can view the results in the ModelSim GUI by loading various **.do** files in the Wave viewer.

1. Launch the ModelSim GUI with the **vsim** command.
2. In the ModelSim Tcl window, execute the **dataset open** command: `dataset open vsim.wlf`
3. Select **View > Open Wave files**.
4. Load the **.do** files to view the waveforms (refer back to Table 6–1 for a listing of the files).

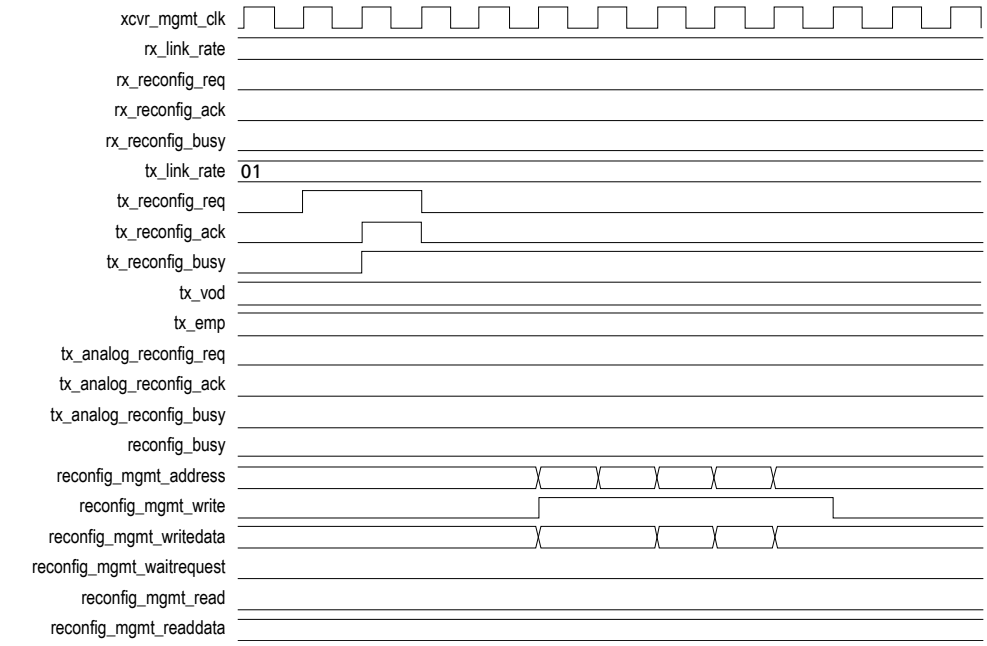
   In the following example, `rx_link_rate` is set to 1 (HBR). When the core makes a request, the `rx_reconfig_req` port goes high. The user logic asserts `rx_reconfig_ack` and then reconfigures the transceiver. During reconfiguration, the user logic holds `rx_reconfig_busy` high; the user logic drives it low when reconfiguration completes.

Figure 7-2: RX Reconfiguration Waveform



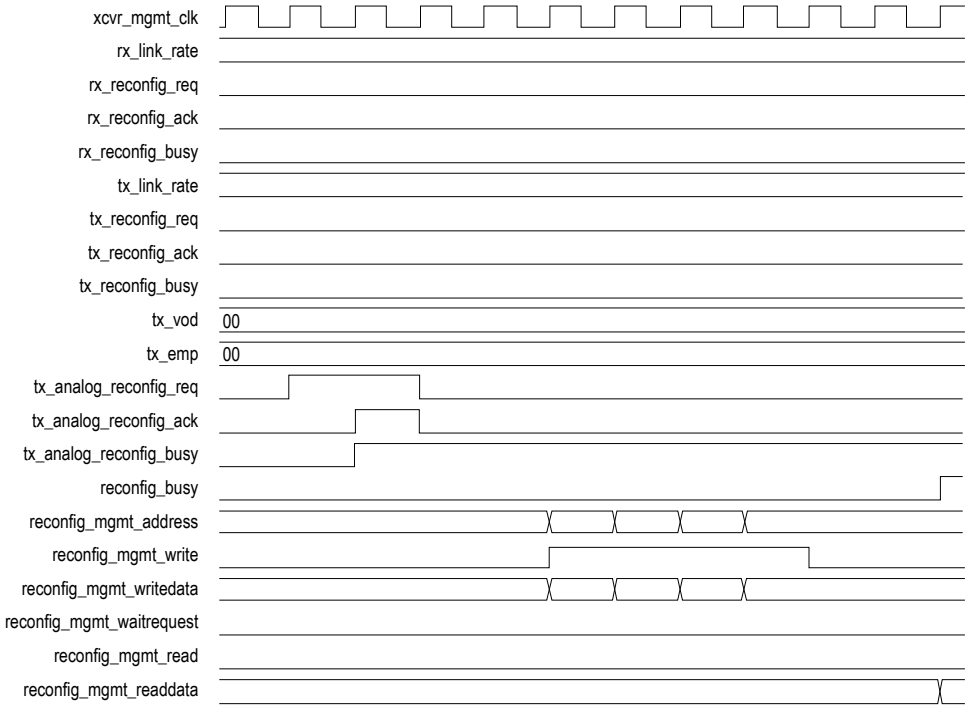
In the following example, `tx_link_rate` is set to 1 (HBR). When the core makes a request, the `tx_reconfig_req` port goes high. The user logic asserts `tx_reconfig_ack` and then reconfigures the transceiver. During reconfiguration, the user logic holds `tx_reconfig_busy` high; the user logic drives it low when reconfiguration completes.

Figure 7-3: TX Reconfiguration Waveform



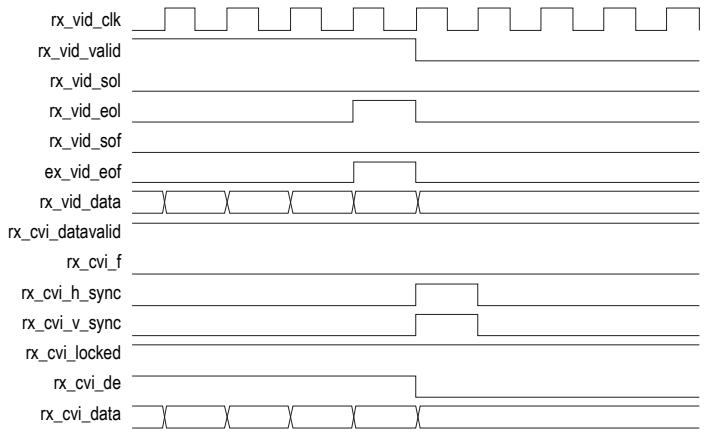
In the following example, tx_vod and tx_emp are both set to 00. When the core makes a request, the tx_analog_reconfig_req port goes high. The user logic asserts tx_analog_reconfig_ack and then reconfigures the transceiver. During reconfiguration, the user logic holds tx_analog_reconfig_busy high; the user logic drives it low when reconfiguration completes.

Figure 7-4: TX Analog Reconfiguration Waveform



The following figure shows an example RX video waveform when interfacing to CVI. The rx_vid_eol signal generates the h_sync pulse by delaying it (by 1 clock cycle) to appear in the horizontal blanking period after the active video ends (VALID is deasserted). The rx_vid_eof signal generates the v_sync pulse by delaying it (by 1 clock cycle) to appear in the vertical blanking period after the active video ends (VALID is deasserted).

Figure 7-5: RX Video Waveform



DisplayPort IP Core Compilation Example

8

2014.06.30

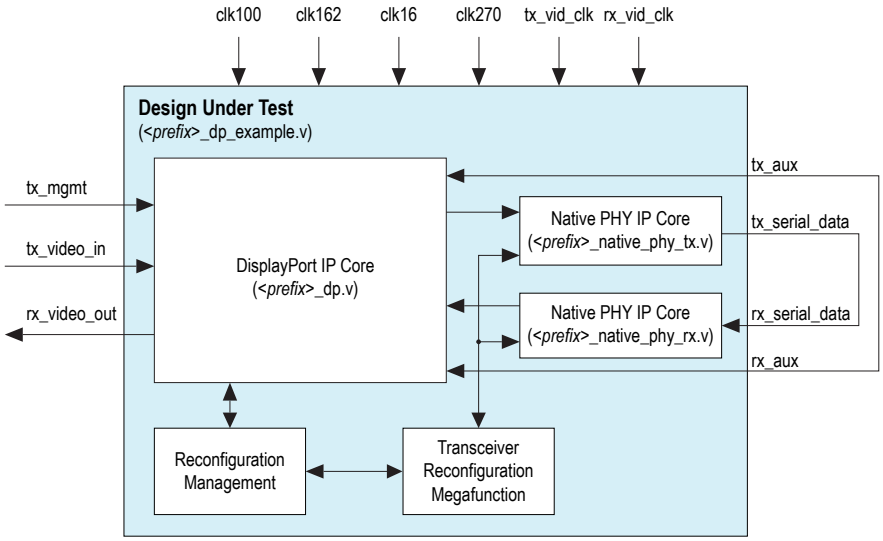
UG-01131

Subscribe Send Feedback

Introduction

The Altera DisplayPort compilation example allows you to evaluate the timing and resource requirements of the DisplayPort IP Core. The compilation example instantiates the DisplayPort IP core with default settings, TX and RX enabled, and 8 bpc. The core has the **Support CTS test automation** option turned on.

Figure 8-1: Compilation Example Block Diagram



Design Walkthrough

Setting up and running the DisplayPort compilation example consists of the following steps:

1. Copy the compilation files to your target directory.
2. Generate the IP compilation files, compile, and view the results.

   You use a script to automate these steps.

Copy the Compilation Files to Your Working Directory

Copy the compilation example files to your working directory using the command:

```
cp -r <IP root directory>/altera/altera_dp/example/<device> <working directory>
```

where <*device*> is **av** for Arria V devices, **cv** for Cyclone V devices, and **sv** for Stratix V devices. Your working directory should contain the files shown below.

Table 8-1: Compilation Example Files

Files are named <*prefix*>_<*name*>.<*extension*> where <*prefix*> represents the device (**av** for Arria V devices, **cv** for Cyclone V devices, and **sv** for Stratix V devices).

File Type	File	Description
Verilog HDL design files	<*prefix*>**_dp_example.v**	Design under test (DUT).
	dp_mif_mappings.v	Table translating MIF mappings for transceiver reconfiguration.
	dp_analog_mappings.v	Table translating VOD and pre-emphasis settings.
	reconfig_mgmt_hw_ctrl.v	Reconfiguration manager top-level.
	reconfig_mgmt_write.v	Reconfiguration manager FSM for a single write command.
IP Catalog files	<*prefix*>**_ dp.v**	IP Catalog variant for the DisplayPort IP core.
	<*prefix*>**_ xcvr_reconfig.v**	IP Catalog variant for the transceiver reconfiguration core.
	<*prefix*>**_ native_phy_rx.v**	IP Catalog variant for the RX transceiver.
	<*prefix*>**_ native_phy_tx.v**	IP Catalog variant for the TX transceiver.
Scripts	**runall.tcl**	This script generates the IP compilation files and scripts, and compiles and simulates them.
	assignments.tcl	This script defines the project settings.
Miscellaneous files	**readme.txt**	Documentation for the compilation example.

Generate the IP Compilation Files, Compile, and View the Results

Use the following command to generate the IP compilation files and compile them:

```
quartus_sh -t runall.tcl
```

This script executes the following commands:

- Load required packages:
  - load_package flow
  - load_package misc

- Regenerate the IP:
  - qexec "qmegawiz -silent <*prefix*>_xcvr_reconfig.v"
  - qexec "qmegawiz -silent <*prefix*>_dp.v"
  - qexec "qmegawiz -silent <*prefix*>_native_phy_rx.v"
  - qexec "qmegawiz -silent <*prefix*>_native_phy_tx.v"
- Create the project, overwriting any previous settings files:
  - project_new <*prefix*>_dp_example -overwrite
- Add the assignments to the project:
  - source assignments.tcl
- Compile the project:
  - execute_flow -compile
- Clean up by closing the project:
  - project_close

View the compilation results in the Quartus II software.

DisplayPort API Reference 9

2014.06.30

UG-01131 Subscribe Send Feedback

You can use the DisplayPort IP core to instantiate sources and sinks. Source instantiations require an embedded controller (Nios II processor or another controller) to act as the policy maker. Sink instantiations greatly benefit from and may optionally use a controller.

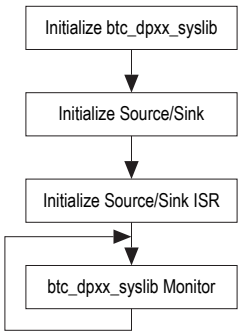
Altera provides software for source and sink instantiations as two system libraries for the Nios II processor (`btc_dptx_syslib` and `btc _dprx_syslib`, respectively). The IP core includes an example main program (`dp_demo_src/main.c`), which demonstrates basic system library use.

Using the Library

The following figure describes a typical user application flow. The user application must initialize the library as its first operation. Next, the application should initialize the instantiated devices (sink and/or source), partly in the `btc_dptx_syslib` and `btc_dprx_syslib` data structures and partly in the user application. You must also implement interrupt service routines (ISRs) to handle interrupts generated by the DisplayPort core.

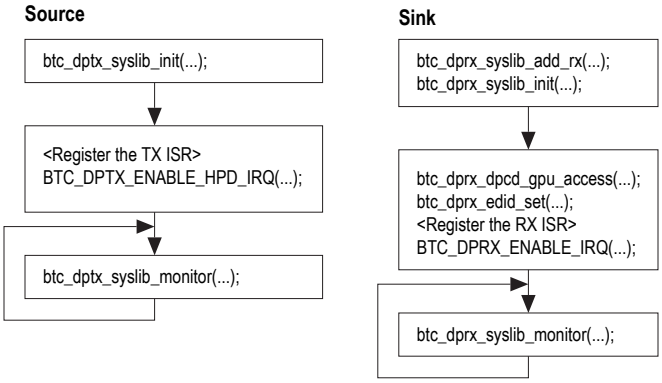
When initialization completes, the user application should periodically invoke the library monitoring function.

Figure 9-1: Typical User Application Flow



The following figure shows a more detailed view of these operations. For a sink application, the user application must initialize the DPCD content and the EDID. Additionally, for both source and sink applications, an interrupt ISR must be registered.

Figure 9-2: Typical Source and Sink User Application Library Calls



Sink instantiations issue an interrupt to the GPU when an AUX channel Request is received from the connected source. Source instantiations issue an interrupt to the GPU when a logic state change is detected on the HPD signal generated by the connected DisplayPort sink.

Because sources always act as AUX channel masters, they can manage AUX communication by initiating a transaction (by sending a request) and then polling the IP core registers waiting to receive a reply. Optionally, source instantiations can also issue an interrupt to the GPU when an AUX channel reply is received from the connected DisplayPort sink, allowing the GPU to execute other tasks while waiting for AUX channel replies.

Enable or disable source and sink interrupts with the following library macros:

- `BTC_DPTX_ENABLE_HPD_ IRQ()`
- `BTC_DPTX_DISABLE_HPD_ IRQ()`
- `BTC_DPTX_ENABLE_AUX_ IRQ()`
- `BTC_DPTX_DISABLE_AUX_ IRQ()`
- `BTC_DPRX_ENABLE_ IRQ()`
- `BTC_DPRX_DISABLE_ IRQ()`

`btc_dprx_syslib` manages one to four sink instances by disabling all GPU interrupts when invoked and restoring them to their previous state on exiting. Therefore, most of the library public functions implement critical sections.

The GPU main program should minimize overhead when serving interrupts generated by sink instances (i.e., interrupts related to a connected source's AUX channel requests).

Interrupts generated by source instances (i.e., interrupts related to a connected sink's HPD activity) can be served with a lower priority. In designs where the same GPU handles both source and sink instances, the GPU must allow for nested interrupts originated by sinks. That is, a sink must be allowed to interrupt a source interrupt service routine (but not another sink interrupt service routine).

Example 9-1: Typical Sink ISR Implementation

```
btc_dprx_aux_get_request (0,&cmd,&address,&length,data);
btc_dprx_aux_handler(0,cmd,address,length,data);
```

Example 9-2: Typical Source ISR Implementation

```
BTC_DPTX_DISABLE_HPD_IRQ(...);
<Enable nested interrupt>
if (HPD asserted)
{
   <read Sink EDID>
   <set video output resolution>
   btc_dptx_link_training(...);
}
else if (HPD deasserted)
   btc_dptx_video_enable(..., 0);
else if (IRQ_HPD)
{
   <check link status>
   if (Test Automation request)
   btc_dptx_test_autom(…);
}
<Disable nested interrupt>
BTC_DPTX_DISABLE_HPD_IRQ(...);
```

btc_dprx_syslib API Reference

This section provides information about the DisplayPort sink system library functions (`btc_dprx_syslib`), including:

- C prototype
- Function description
- Whether the function is thread-safe when running in a multi-threaded environment
- Whether the function can be invoked from an ISR
- Example

btc_dprx_aux_get_request

Prototype:	`int btc_dprx_aux_get_request(`
	`  BYTE  rx_idx`
	`  BYTE  *cmd,`
	`  unsigned int *address,`
	`  BYTE  *length,`
	`  BYTE  *data)`

Thread-safe:	Yes
Available from ISR:	Yes
Include:	`< btc_dprx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	• `rx_idx`—Sink instance index (0 - 3) • `cmd`—Pointer to command • `address`—Pointer to address • `length`—Pointer to length (0 - 16) • `data`–Pointer to data received
Description:	This function retrieves an AUX channel request issued by the connected DisplayPort source. `cmd` and `address` are the command byte and the address in the original request received, respectively (refer to the DisplayPort specification for more details). When the request is a write, `*data` fills with the data bytes sent by the source. To support address-only requests, `length` is the original `len` byte sent by the source incremented by one.
Example:	`btc_dprx_aux_get_request(0, pcmd, padd, plen, pwrdata);`

Related Information

btc_dprx_aux_handler on page 9-5

btc_dprx_aux_handler

Prototype:	`int btc_dprx_aux_handler(`
	`  BYTE rx_idx`
	`  BYTE cmd,`
	`  unsigned int address,`
	`  BYTE length,`
	`  BYTE *data)`

Thread-safe:	Yes
Available from ISR:	Yes
Include:	`< btc_dprx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	• `rx_idx`—Sink instance index (0 - 3) • `cmd`—Command • `address`—Address • `length`–Length (0 - 16) • `data`–Pointer to data being written
Description:	This function processes an AUX channel request issued by the connected DisplayPort source. `cmd` and `address` are the command byte and the address in the original request received, respectively (refer to the DisplayPort specification for more details). When the request is a write, `data` must point to the data bytes sent by the source. To support address-only requests, `length` is the original `len` byte sent by the source incremented by one. When the request is a read, `data` is not used and can be NULL. This function provides all the functionality of the DPCD registers implemented inside the system library, including: • DPCD locations read/write support • EDID read support • Link training execution • Forwarding of AUX channel replies back to the source
Example:	`btc_dprx_aux_handler(0, pcmd, padd, plen, pwrdata);`

Related Information

btc_dprx_aux_get_request on page 9-4

btc_dprx_aux_post_reply

Prototype:	`int btc_dprx_aux_post_reply(`
	`  BYTE rx_idx`
	`  BYTE cmd,`
	`  BYTE size,`
	`  BYTE *data)`

Thread-safe:	Yes
Available from ISR:	Yes
Include:	< `btc_dprx_syslib.h` >
Return:	0 = success, 1 = fail
Parameters:	• `rx_idx`—Sink instance index (0 - 3) • `cmd`–Command • `size`—Number of data bytes transmitted (0 - 16) • `data`–Pointer to data transmitted
Description:	This function transmits an AUX channel reply to the connected DisplayPort source. `cmd` is the reply command byte (refer to the DisplayPort specification for more details) . When the reply includes read data, `*data` fills with the data bytes sent to the source. To support replies with no data returned, `size` is the actual `len` byte sent to the source incremented by one.
Example:	`btc_dprx_aux_post_reply (0, 0x10, 0, NULL); //Reply AUX_NACK`

Related Information

btc_dprx_aux_get_request on page 9-4

btc_dprx_baseaddr

Prototype:	`unsigned int btc_dprx_baseaddr(BYTE rx_idx)`
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< `btc_dprx_syslib.h` >
Return:	0 = success, 1 = fail
Parameters	`rx_idx`—Sink instance index (0 - 3)

Description:	This function returns the RX instance's base address connected to the given port number.
Example:	`addr = btc_dprx_baseaddr(0);`

btc_dprx_dpcd_gpu_access

Prototype:	`int btc_dprx_dpcd_gpu_access(`
	`  BYTE rx_idx`
	`  BYTE wrcmd,`
	`  unsigned int address,`
	`  BYTE length,`
	`  BYTE *data)`

Thread-safe:	Yes
Available from ISR:	Yes
Include:	`< btc_dprx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	• `rx_idx`—Sink instance index (0 - 3) • `wrcmd`–0 = read, 1 = write • `address`—Address • `length`—Length (1 - 255) • `data`—Pointer to data
Description:	This function allows the controller to access the sink's DPCD locations (implemented in the system library) for reading and writing data. `data` must point to a location containing `length` bytes (writes) or be able to accommodate `length` bytes (reads).
Example:	`btc_dprx_dpcd_gpu_access(0, 1, 0x00000, 1, pwrdata);`

btc_dprx_edid_set

Prototype:	`int btc_dprx_edid_set(`
	`  BYTE rx_idx`
	`  BYTE port,`
	`  BYTE *edid_data,`
	`  BYTE num_blocks)`

Thread-safe:	Yes
Available from ISR:	Yes
Include:	`< btc_dprx_syslib.h >`
Return:	0 = success, 1 = fail

Parameters:	• rx_idx—Sink instance index (0 - 3) • port–RX port (stream) number (0 – 3) • edid_data–Pointer to EDID data memory • num_blocks–EDID size in blocks
Description:	This function allows the controller to set the content of the sink's EDID (implemented in the system library). The library references the EDID data and does not copy it. One block is 128-bytes long. The system library accepts a maximum of 4 blocks (512-byte long EDIDs). Each streaming sink port has its own EDID.
Example:	`btc_dprx_edid_set(0, 0, pmy_edid, 2);`

btc_dprx_hpd_get

Prototype:	`int btc_dprx_hpd_get(BYTE rx_idx)`
Thread-safe:	Yes
Available from ISR:	Yes
Include:	`<btc_dprx_syslib.h>`
Return:	0 = success, 1 = fail
Parameters:	rx_idx–Sink instance index (0 - 3)
Description:	Returns the current logic level of the RX HPD.
Example:	`btc_dprx_hpd_get(0);`

Related Information

- **btc_dprx_hpd_pulse** on page 9-9
- **btc_dprx_hpd_set** on page 9-9

btc_dprx_hpd_pulse

Prototype:	`void btc_dprx_hpd_pulse(BYTE rx_idx)`
Thread-safe:	Yes
Available from ISR:	Yes
Include:	`< btc_dprx_syslib.h >`
Return:	–
Parameters:	`rx_idx`—Sink instance index (0 - 3)
Description:	This function deasserts (i.e., sets to 0) the RX HPD for 750 s. You can use this function to send an `IRQ_HPD` pulse to the connected DisplayPort source. Before invoking this function, you must have invoked `btc_dprx_hpd_set` with level = 1 (i.e., HPD must be set to 1).
Example:	`btc_dprx_pulse(0);`

Related Information

- **btc_dprx_hpd_get** on page 9-8
- **btc_dprx_hpd_set** on page 9-9

btc_dprx_hpd_set

Prototype:	`void btc_dprx_hpd_set(`
	`  BYTE rx_idx,`
	`  int  level)`

Thread-safe:	Yes
Available from ISR:	Yes
Include:	`< btc_dprx_syslib.h >`
Return:	–
Parameters:	• `rx_idx`—Sink instance index (0 - 3) • `level`–0 or 1
Description:	This function allows the controller to set the logic level of the RX HPD.
Example:	`btc_dprx_hpd_set(0,1);`

Related Information

- **btc_dprx_hpd_get** on page 9-8

- **btc_dprx_hpd_pulse** on page 9-9

btc_dprx_syslib_add_rx

Prototype:	`int btc_dprx_syslib_add_rx(`
	`  BYTE  rx_idx,`
	`  unsigned int rx_base_addr,`
	`  unsigned int rx_irq_id,`
	`  unsigned int rx_irq_num,`
	`  unsigned int rx_num_of_sinks)`

Thread-safe:	No
Available from ISR:	No
Include:	`< btc_dprx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	• `rx_idx`—Sink instance index (0 - 3) • `rx_base_addr`—RX base address • `rx_irq_id`—RX IRQ ID • `rx_irq_num`–RX IRQ number • `rx_num_of_sinks`—Number of streaming sinks used (1 - 4)
Description:	This function declares a sink (RX) instance to the system library. It should be invoked once for each existing sink instance, starting from `rx_idx = 0`. After all sink have been declared, invoke `btc_syslib_ init ( )`.
Example:	`btc_dprx_syslib_add_rx (0, BITEC_DP_0_AV_RX_CONTROL_BASE, BITEC_DP_0_AV_RX_CONTROL_IRQ_INTERRUPT_CONTROLLER_ID, BITEC_DP_0_AV_RX_CONTROL_IRQ, 2);`

Related Information

btc_dprx_syslib_init on page 9-11

btc_dprx_syslib_info

Prototype:	`void btc_dprx_syslib_info(`
	`  BYTE *max_sink_num,`
	`  BYTE *mst_support)`

Thread-safe:	Yes
Available from ISR:	Yes
Include:	`< btc_dprx_syslib.h >`
Return:	None
Parameters:	• `max_sink_num`—Pointer for maximum number of sinks supported • `mst_support`—Pointer for MST support
Description:	This function returns information about the system library capabilities. On return, `max_sink_num` is set with the maximum number of supported sink instances (1 - 4) and `mst_support` is set to zero if MST is not supported and 1 if it is supported.
Example:	`btc_dprx_syslib_info(pmaxsink,pmst);`

btc_dprx_syslib_init

Prototype:	`int btc_dprx_syslib_init(void)`
Thread-safe:	No
Available from ISR:	No
Include:	`< btc_dprx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	No
Description:	This function initializes the system library. It should be invoked once after `btc_dprx_syslib_add_ rx ( )`.
Example:	`btc_dprx_syslib_init();`

Related Information

btc_dprx_syslib_add_rx on page 9-10

btc_dprx_syslib_monitor

Prototype:	`int btc_dprx_syslib_monitor(void)`
Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dprx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	No
Description:	This function calls the system library sink housekeeping monitor, which is responsible for: • Handling RX-side received sideband message requests. • Forwarding RX-side sideband message replies. The software should invoke this function periodically or at least every 50 ms.
Example:	`btc_dprx_syslib_monitor();`

btc_dptx_syslib API Reference

This section provides information about the DisplayPort source system library functions (`btc_dptx_syslib`), including:

- C prototype
- Function description
- Whether the function is thread-safe when running in a multi- threaded environment
- Whether the function can be invoked from an ISR
- Example

btc_dptx_aux_i2c_read

Prototype:	`int btc_dptx_aux_i2c_read(`
	`  BYTE address,`
	`  BYTE size,`
	`  BYTE *data,`
	`  BYTE mot)`

Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	• `address`–I^2C address • `size`–Number of bytes (1 - 16) • `data`—Pointer to data to be read • `mot`—Middle of transaction (0 or 1)
Description:	This function reads 1 to 16 data bytes from the connected DisplayPort sink's I^2C interface mapped over the AUX channel.
Example:	`btc_dptx_aux_i2c_read(0x50, 16, data, 1);`

Related Information

btc_dptx_aux_i2c_write on page 9-14

btc_dptx_aux_i2c_write

Prototype:	`int btc_dptx_aux_i2c_write(`
	`  BYTE address,`
	`  BYTE size,`
	`  BYTE *data,`
	`  BYTE mot)`

Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	• `address`–I^2C address • `size`–Number of bytes (1 - 16) • `data`–Pointer to data to be written • `mot`—Middle of transaction (0 or 1)
Description:	This function writes 1 to 16 data bytes to the connected DisplayPort sink's I^2C interface mapped over the AUX channel.
Example:	`btc_dptx_aux_i2c_write(0x50, 1, data, 1);`

Related Information

btc_dptx_aux_i2c_read on page 9-13

btc_dptx_aux_read

Prototype:	`int btc_dptx_aux_read(`
	`  unsigned int address,`
	`  BYTE size,`
	`  BYTE *data)`

Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	• 0 = AUX_ACK replied • 1 = Source internal error • 2 = Reply timeout • 3 = AUX_NACK replied • 4 = AUX_DEFER replied • 5 = Invalid reply
Parameters	• `address`—DPCD start address • `size`–Number of bytes (1 - 16) • `data`—Pointer for data to be read
Description:	This function reads 1 to 16 data bytes from the connected DisplayPort sink's DPCD.
Example:	`btc_dptx_aux_read(0x202, 2, &status);`

Related Information

btc_dptx_aux_write on page 9-16

btc_dptx_aux_write

Prototype:	`int btc_dptx_aux_write(`
	`  unsigned int address,`
	`  BYTE size,`
	`  BYTE *data)`

Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	• 0 = AUX_ACK replied • 1 = Source internal error • 2 = Reply timeout • 3 = AUX_NACK replied • 4 = AUX_DEFER replied • 5 = Invalid reply
Parameters	• `address`—DPCD start address • `size`–Number of bytes (1 - 16) • `data`–Pointer to data to be written
Description:	This function writes 1 to 16 data bytes to the connected DisplayPort sink's DPCD.
Example:	`btc_dptx_aux_write(0x600, 1, data_ptr);`

Related Information

btc_dptx_aux_read on page 9-15

btc_dptx_baseaddr

Prototype:	`unsigned int btc_dptx_baseaddr(void)`
Thread-safe:	Yes
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	No
Description:	This function returns the base address of the TX instance connected to the given port number.
Example:	`addr = btc_dptx_baseaddr();`

btc_dptx_edid_block_read

Prototype:	`int btc_dptx_edid_block_read(`
	`  BYTE block,`
	`  BYTE *data)`

Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	• `block`—Block number (0 - 3) • `data`–Pointer for data to be read
Description:	Reads one block (128 bytes) from the EDID of the connected DisplayPort sink.
Example:	`btc_dptx_edid_block_read(2, pdata);`

Related Information

btc_dptx_edid_read on page 9-18

btc_dptx_edid_read

Prototype:	`int btc_dptx_edid_read(BYTE *data)`
Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	`data`–Pointer for data to be read
Description:	This function reads the complete EDID of the connected DisplayPort sink. `data` must be able to contain the whole EDID (allow for 512 bytes).
Example:	`btc_dptx_edid_read(pdata);`

Related Information

btc_dptx_edid_block_read on page 9-17

btc_dptx_fast_link_training

Prototype:	`int btc_dptx_fast_link_training(`
	`  unsigned int link_rate,`
	`  unsigned int lane_count,`
	`  unsigned int volt_swing,`
	`  unsigned int pre_emph,`
	`  unsigned int new_cfg)`

Thread-safe:	No
Available from ISR:	Yes
Include:	< `btc_dptx_syslib.h` >
Return:	0 = success, 1 = fail
Parameters:	• `link_rate`–Link rate (Gbps): 0 = 1.62; 1 = 2.70; 2 = 5.40 • `lane_count`—1, 2, or 4 • `volt_swing`–0, 1, 2, or 3 • `pre_emph`–0, 1, 2, or 3 • `new_cfg`–0 = ignore the other parameters; 1 = use provided parameters
Description:	This function performs fast link training with the connected DisplayPort sink. When performing fast link training, the IP core outputs training pattern 1 for 1 ms followed by training pattern 2 for 1 ms. The function returns a 1 if link training fails or if the DPCD flag `NO_AUX_HANDSHAKE_LINK_TRAINING` = 0 (at location 00103h). • If `new_cfg` = 1, the IP core updates the sink's DPCD with the provided `link_rate` and `lane_count`, sets its own transceiver with the provided `volt_swing` and `pre_emph`, and then performs fast link training. • If `new_cfg` = 0, the IP core uses the current transceiver setting, link rate, and lane count, and performs fast link training.
Example:	`btc_dptx_fast_link_training(1, 4, 1, 0, 1);`

Related Information

btc_dptx_link_training on page 9-20

btc_dptx_link_training

Prototype:	`int btc_dptx_link_training(`
	`  unsigned int link_rate,`
	`  unsigned int lane_count)`

Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	• `link_rate`–Link rate (Gbps): 0 = 1.62; 1 = 2.70; 2 = 5.40 • `lane_count`—1, 2, or 4
Description:	This function performs link training with the connected DisplayPort sink.
Example:	`btc_dptx_link_training(1, 4);`

btc_dptx_set_color_space

Prototype:	`int btc_dptx_set_color_space(`
	`  BYTE format,`
	`  BYTE bpc,`
	`  BYTE range,`
	`  BYTE colorimetry)`

Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	• `format`—0 = RGB; 1 = YCbCr 4:2:2; 2 = YCbCr 4:4:4 • `bpc`—Color depth (bpc): 0 = 6; 1 = 8; 2 = 10; 3 = 12; 4 = 16 • `range`—0 = VESA; 1 = CEA • `colorimetry`—0 = BT601-5; 1 = BT709-5
Description:	This function sets the color space for TX transmitted video.
Example:	`btc_dptx_set_color_space(0, 1, 0, 0);`

btc_dptx_syslib_init

Prototype:	`int btc_dptx_syslib_init(`
	`  unsigned int tx_base_addr,`
	`  unsigned int tx_irq_id,`
	`  unsigned int tx_irq_num)`

Thread-safe:	No
Available from ISR:	No
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	• `tx_base_addr`—TX base address • `tx_irq_id`—TX IRQ ID • `tx_irq_num`–TX IRQ number
Description:	Initializes the system library. Should be invoked as the first function in the library by `main( )`. Set the base address of TX or RX to `BTC_NOT_PRESENT` if TX or RX not instantiated.
Example:	`btc_dptx_syslib_init( BITEC_DP_0_AV_TX_CONTROL_BASE, BITEC_DP_0_AV_TX_CONTROL_IRQ_INTERRUPT_CONTROLLER_ID, BITEC_DP_0_AV_TX_CONTROL_IRQ) ;`

btc_dptx_syslib_monitor

Prototype:	`int btc_dptx_syslib_monitor(void)`
Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	No
Description:	This function calls the system library source housekeeping monitor. The software should invoke this function periodically or at least every 50 ms.
Example:	`btc_dptx_syslib_monitor();`

btc_dptx_test_autom

Prototype:	`int btc_dptx_test_autom(void)`
Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	No
Description:	This function handles test automation requests from the connected DisplayPort sink. You should invoke this function after the IP core senses an HPD IRQ and identifies it as a test automation request. The function implements `TEST_LINK_TRAINING` and `TEST_EDID_READ`.
Example:	`btc_dptx_test_autom();`

btc_dptx_video_enable

Prototype:	`int btc_dptx_video_enable(BYTE enabled)`
Thread-safe:	No
Available from ISR:	Yes
Include:	`< btc_dptx_syslib.h >`
Return:	0 = success, 1 = fail
Parameters:	`enabled`–0 = output idle pattern; 1 = output active video
Description:	This function enables the TX to output either active video or an idle pattern. After successful link training, the TX outputs active video by default.
Example:	`btc_dptx_video_enable(1);`

btc_dpxx_syslib Additional Types

In addition to the standard ANSI C defined types, `btc_dpxx_syslib` uses the following types:

- `#define BYTE unsigned char`

btc_dprx_syslib Supported DPCD Locations

Sink-Supported DPCD Locations on page 11-25 provides a list of DPCD locations currently supported in `btc_dprx_syslib` sink instantiations. Read accesses to unsupported locations receive a response of `NATIVE_ACK` with data content set to zero. Write accesses to unsupported locations receive a response of `NATIVE_NACK`.

DisplayPort Source Register Map 10

2014.06.30

UG-01131 Subscribe Send Feedback

DisplayPort source instantiations require an embedded controller (Nios II processor or another controller) to act as the policy maker. This section describes the register map.

Table 9–1 describes the notation used in this section.

Table 10-1: Notation

Shorthand	Definition
RW	Read/write
RO	Read only
WO	Write only
CRO	Clear on read or write, read only
CWO	Clear on read or write, write only

General Registers

This section describes the general registers.

DPTX_TX_CONTROL

The IRQ is asserted when AUX_IRQ_EN = 1 and in register DPTX_AUX_CONTROL flag MSG_READY = 1. IRQ is de-asserted by setting AUX_IRQ_EN to 0 or reading from DPTX_AUX_COMMAND. IRQ is also asserted if HPD_IRQ_EN = 1 and a new HPD event is detected (HPD_EVENT in register DPTX_TX_STATUS different from 00). IRQ is de-asserted by setting HPD_IRQ_EN to 0 or reading from DPTX_TX_STATUS.

The TX_LINK_RATE drives the respective tx_reconfig port. Setting LANE_COUNT to 00000 causes the transmitter GXB to always send a logical zero (i.e., a constant voltage level). You can use this function to cause a "power down" for link layer compliance testing.

Address: 0x0000

Direction: RW

Reset: 0x00000000

Table 10-2: DPTX_TX_CONTROL Bits

Bit	Bit Name	Function
31	HPD_IRQ_EN	Enables an IRQ issued to the Nios II processor on an HPD event: • 0 = disable • 1 = enable
30	AUX_IRQ_EN	Enables an IRQ issued to the Nios II processor when an AUX channel transaction reply is received from the sink: • 0 = disable • 1 = enable
29:21	Unused	
20	RESERVED	Reserved
19	ENHANCED_FRAME	0 = Standard framing 1 = Enhanced framing
18:17	TX_LINK_RATE	Main link rate: • 0 = 1.62 Gbps • 1 = 2.7 Gbps • 2 = 5.4 Gbps
16:15	Unused	
14	ASYNC_CLOCK	0 = Synchronous 1 = Asynchronous
13:10	Unused	
9:5	LANE_COUNT	Lane count: • 00000 = Reserved • 00001 = 1 • 00010 = 2 • 00100 = 4
4	Unused	
3:0	TP	Current training pattern: • 0000 = Normal video • 0001 = Training pattern 1 • 0010 = Training pattern 2 • 0011 = Video idle pattern • 1000 = Symbol error rate measurement pattern

DPTX_TX_STATUS

The IP core issues an IRQ to the Nios II processor if the DPTX_TX_CONTROL registers HPD_IRQ_EN is 1 and the IP core detects a new HPD event. HPD_EVENT provides information about the event that caused the interrupt. The interrupt and HPD_EVENT bit fields are both cleared by reading the DPTX_TX_STATUS register.

Address: 0x0001

Direction: CRO

Reset: 0x00000000

Table 10-3: DPTX_TX_STATUS Bits

Bit	Bit Name	Function
31:4	Unused	
3	RESERVED	Reserved
2	HPD_LEVEL	Current HPD logic level
1:0	HPD_EVENT	HPD event causing IRQ (read to clear): • 00 = No event • 01 = HPD plug event (long HPD) • 10 = HPD IRQ (short HPD) • 11 = Reserved

MSA Registers

The MSA registers are allocated at addresses:

- 0x0020 through 0x002e for Stream0
- 0x0040 through 0x004e for Stream1
- 0x0060 through 0x006e for Stream2
- 0x0080 through 0x008e for Stream3

Note: Only registers for Stream0 are listed in the following sections.

DPTX0_MSA_MVID

Address: 0x0020

Direction: RO

Reset: 0x00000000

Table 10-4: DPTX0_MSA_MVID Bits

Bit	Bit Name	Function
31:24	Unused	

Bit	Bit Name	Function
23:0	MVID	Main stream attribute MVID

DPTX0_MSA_NVID

Address: 0x0021

Direction: RO

Reset: 0x00000000

Table 10-5: DPTX0_MSA_NVID Bits

Bit	Bit Name	Function
31:24	Unused	
23:0	NVID	Main stream attribute NVID

DPTX0_MSA_HTOTAL

Address: 0x0022

Direction: RO

Reset: 0x00000000

Table 10-6: DPTX0_MSA_HTOTAL Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	HTOTAL	Main stream attribute HTOTAL

DPTX0_MSA_VTOTAL

Address: 0x0023

Direction: RO

Reset: 0x00000000

Table 10-7: DPTX0_MSA_VTOTAL Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	VTOTAL	Main stream attribute VTOTAL

DPTX0_MSA_HSP

Address: 0x0024

Direction: RO

Reset: 0x00000000

Table 10-8: DPTX0_MSA_HSP Bits

Bit	Bit Name	Function
31:1	Unused	
0	HSP	Main stream attribute horizontal sync polarity: • 0 = Positive • 1 = Negative

DPTX0_MSA_HSW

Address: 0x0025

Direction: RO

Reset: 0x00000000

Table 10-9: DPTX0_MSA_HSW Bits

Bit	Bit Name	Function
31:15	Unused	
14:0	HSW	Main stream attribute horizontal sync width

DPTX0_MSA_HSTART

Address: 0x0026

Direction: RO

Reset: 0x00000000

Table 10-10: DPTX0_MSA_HSTART Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	HSTART	Main stream attribute HSTART

DPTX0_MSA_VSTART

Address: 0x0027

Direction: RO

Reset: 0x00000000

Table 10-11: DPTX0_MSA_VSTART Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	VSTART	Main stream attribute VSTART

DPTX0_MSA_VSP

Address: 0x0028

Direction: RO

Reset: 0x00000000

Table 10-12: DPTX0_MSA_VSP Bits

Bit	Bit Name	Function
31:1	Unused	
0	VSP	Main stream attribute vertical sync polarity • 0 = Positive • 1 = Negative

DPTX0_MSA_VSW

Address: 0x0029

Direction: RO

Reset: 0x00000000

Table 10-13: DPTX0_MSA_VSW Bits

Bit	Bit Name	Function
31:15	Unused	
14:0	VSW	Main stream attribute vertical sync width

DPTX0_MSA_HWIDTH

Address: 0x002a

Direction: RO

Reset: 0x00000000

Table 10-14: DPTX0_MSA_HWIDTH Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	HWIDTH	Main stream attribute HWIDTH

DPTX0_MSA_VHEIGHT

Address: 0x002b

Direction: RO

Reset: 0x00000000

Table 10-15: DPTX0_MSA_VHEIGHT Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	VHEIGHT	Main stream attribute VHEIGHT

DPTX0_MSA_MISC0

Address: 0x002c

Direction: RO

Reset: 0x00000000

Table 10-16: DPTX0_MSA_MISC0 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	MISC0	Main stream attribute MISC0

DPTX0_MSA_MISC1

Address: 0x002d

Direction: RO

Reset: 0x00000000

Table 10-17: DPTX0_MSA_MISC1 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	MISC1	Main stream attribute MISC1

DPTX0_MSA_COLOUR

Address: 0x002e

Direction: RW

Reset: 0x00000001

Table 10-18: DPTX0_MSA_MISC1 Bits

Bit	Bit Name	Function
31:7	Unused	
6	COLORIMETRY	0 = ITU-R BT601-5 1 = ITU-R BT709-5
5	DYNAMIC_RANGE	0 = VESA (from 0 to maximum) 1 = CEA range
4:3	COMPONENT_FORMAT	00 = RGB 01 = YCbCr 4:2:2 10 = YCbCr 4:4:4 11 = Reserved
2:0	BPP	Bits per pixel format • 000 = 6 bpc • 001 = 8 bpc • 010 = 10 bpc • 011 = 12 bpc • 100 = 16 bpc

Link Voltage and Pre-Emphasis Controls

This section describes the registers for the link voltage and pre-emphasis controls.

DPTX_PRE_VOLT0

These ports drive the respective tx_analog_reconfig ports.

Address: 0x0010

Direction: RW

Reset: 0x00000000

Table 10-19: DPTX_PRE_VOLT0 Bits

Bit	Bit Name	Function
31:4	Unused	

Bit	Bit Name	Function
3:2	PRE0	Pre-emphasis output on lane 0
1:0	VOLT0	Voltage swing output on lane 0

DPTX_PRE_VOLT1

These ports drive the respective tx_analog_reconfig ports.

Address: 0x0011

Direction: RW

Reset: 0x00000000

Table 10-20: DPTX_PRE_VOLT1 Bits

Bit	Bit Name	Function
31:4	Unused	
3:2	PRE1	Pre-emphasis output on lane 1
1:0	VOLT1	Voltage swing output on lane 1

DPTX_PRE_VOLT2

These ports drive the respective tx_analog_reconfig ports.

Address: 0x0012

Direction: RW

Reset: 0x00000000

Table 10-21: DPTX_PRE_VOLT2 Bits

Bit	Bit Name	Function
31:4	Unused	
3:2	PRE2	Pre-emphasis output on lane 2
1:0	VOLT2	Voltage swing output on lane 2

DPTX_PRE_VOLT3

These ports drive the respective tx_analog_reconfig ports.

Address: 0x0013

Direction: RW

Reset: 0x00000000

Table 10-22: DPTX_PRE_VOLT3 Bits

Bit	Bit Name	Function
31:4	Unused	
3:2	PRE3	Pre-emphasis output on lane 3
1:0	VOLT3	Voltage swing output on lane 3

DPTX_RECONFIG

RECONFIG_ANALOG drives the tx_analog_reconfig port (tx_analog_reconfig_req) , while RECONFIG_LINKRATE drives the tx_reconfig port, (tx_reconfig_req) . GXB_BUSY connects to the tx_analog_reconfig input ports (tx_analog_reconfig_busy, and tx_reconfig tx_reconfig_busy).

Address: 0x0014

Direction: RW

Reset: 0x00000000

Table 10-23: DPTX_RECONFIG Bits

Bit	Bit Name	Function
31	GXB_BUSY	Read-only flag where: • 0 = Transceiver is not busy • 1 = Transceiver is busy
30:2	Unused	
1	RECONFIG_LINKRATE	0 = Keep current transceiver link rate 1 = Reconfigure the transceiver with the link rate in DPTX_TX_CONTROL (TX_LINK_RATE)
0	RECONFIG_ANALOG	0 = Keep current transceiver analog configuration 1 = Reconfigure transceiver with analog values in DPTX_PRE_VOLT0-3

Link Quality Pattern Generation Register

If you use this register, you can force the source to output a standard PRBS7 test pattern on each of the selected lanes, which can be useful for performing BER measurements.

Address: 0x0015

Direction: RW

Reset: 0x00000000

Table 10-24: DPTX_LINK_QUALITY_PATTERN Bits

Bit	Bit Name	Function
31:12	Unused	
11:9	LQ_PATT_LANE3	Pattern selection for lane 3: • 000 = No training pattern (normal mode) • 011 = PRBS7 transmitted
8:6	LQ_PATT_LANE2	Pattern selection for lane 2: • 000 = No training pattern (normal mode) • 011 = PRBS7 transmitted
5:3	LQ_PATT_LANE1	Pattern selection for lane 1: • 000 = No training pattern (normal mode) • 011 = PRBS7 transmitted
2:0	LQ_PATT_LANE0	Pattern selection for lane 0: • 000 = No training pattern (normal mode) • 011 = PRBS7 transmitted

Timestamp

The Nios II processor can use this global, free-running counter to generate timestamps and delays. The same counter is used in both sink and source instantiations (DPRX_TIMESTAMP is always equal to DPTX_TIMESTAMP).

Address: 0x001F

Direction: RO

Reset: 0x00000000

Table 10-25: DPTX_TIMESTAMP Bits

Bit	Bit Name	Function
31:24	Unused	8'b00000000
23:0	TIMESTAMP	Free-running counter value (1 tick equals 100 µs)

Audio Register

This register controls the values related to the audio data stream.

Address: 0x002f

Direction: RW

Reset: The maximum number of channels supported minus 1 0x00000000 to 0x00000007

Table 10-26: DPTX_AUD_CONTROL Bits

Bit	Bit Name	Function
31	SOFT_MUTE	1 = Audio is muted 0 = Audio is muted if tx_audio_mute is asserted
30:24	Unused	
17:16	LFEBPL	Audio InfoFrame LFE playback level (LFEPBL, see CEA-861-E specification)
15:8	CA	Audio InfoFrame channel allocation (CA, see CEA-861-E specification)
7:4	LSV	Audio InfoFrame level shift value (LSV, see CEA-861-E specification)
3	DM_INH	Audio InfoFrame down mix inhibit flag (DM_INH, see CEA-861-E specification)
2:0	CH_COUNT	Channel count • 000 = 1 channel • 001 = 2 channels ... • 111 = 8 channels

CRC Registers

Computed video CRC red component, DPTX0_CRC_R, bits.

Address: 0x0030

Direction: RO

Reset: 0x00000000

Table 10-27: DPTX_CRC_R Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	CRC_R	Input video CRC for the red component

Computed video CRC green component, DPTX0_CRC_G, bits.

Address: 0x0031

Direction: RO

Reset: 0x00000000

Table 10-28: DPTX_CRC_G Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	CRC_G	Input video CRC for the green component

Computed video CRC blue component, DPTX0_CRC_B, bits.

Address: 0x0032

Direction: RO

Reset: 0x00000000

Table 10-29: DPTX_CRC_B Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	CRC_B	Input video CRC for the blue component

AUX Controller Interface

This section describes the registers that connect with the AUX controller interface.

DPTX_AUX_CONTROL

For transaction requests:

1. Wait for READY_TO_TX to be 1.
2. Write registers DPTX_AUX_COMMAND to DPTX_AUX_BYTE18 with the transaction command, address, length (0 – 15) fields, and data payload.
3. Write LENGTH with the transaction's total message length (3 for header + 1 for length byte + 0 to 16 for data bytes).
4. The request transmission begins.

For transaction replies:

1. Issue a transaction request.
2. Wait for MSG_READY to be 1. Implement a timeout.
3. Read the transaction reply's total length from LENGTH.
4. Read the transaction reply's command from the DPTX_AUX_COMMAND register. This transaction clears MSG_READY and LENGTH.
5. Read the transaction reply's data payload from registers DPTX_AUX_BYTE0 to DPTX_AUX_BYTE15 (read LENGTH - 1 bytes).

Address: 0x0100

Direction: RW

Reset: 0x00000000

Table 10-30: DPTX_AUX_CONTROL Bits

Bit	Bit Name	Function
31	MSG_READY	0 = Waiting for a reply 1 = A reply has been completely received
30	READY_TO_TX	0 = Busy sending a request or waiting for a reply 1 = Ready to send a request
29:5	Unused	
4:0	LENGTH	For the next transaction request, total length of message to be transmitted (3 – 20), for the last received transaction reply, total length of message received (1 – 17).

DPTX_AUX_CMD

Address: 0x0101

Direction: RW

Reset: 0x00000000

Table 10-31: DPTX_AUX_CMD Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	COMMAND	AUX transaction command for the next request or received in the most recent reply (refer to the DisplayPort specification for details). Reading of this register clears MSG_READY and LENGTH in DPTX_AUX_CONTROL register .

DPTX_AUX_BYTE0

AUX Transaction Byte 0 Register.

Address: 0x0102

Direction: RW

Reset: 0x00000000

Table 10-32: DPTX_AUX_BYTE0 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction address [15:8] for the next request, or data(0) received in the last reply

DPTX_AUX_BYTE1

AUX Transaction Byte 1 Register.

Address: 0x0103

Direction: RW

Reset: 0x00000000

Table 10-33: DPTX_AUX_BYTE1 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction address [7:1] for the next request, or data(1) received in the last reply

DPTX_AUX_BYTE2

AUX Transaction Byte 2 Register.

Address: 0x0104

Direction: RW

Reset: 0x00000000

Table 10-34: DPTX_AUX_BYTE2 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction length[3:0] for the next request, or data(2) received in the last reply (refer to the DisplayPort specification for details)

DPTX_AUX_BYTE3

AUX Transaction Byte 3 Register.

Address: 0x0105

Direction: RW

Reset: 0x00000000

Table 10-35: DPTX_AUX_BYTE3 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(0) for the next request, or data(3) received in the last reply

DPTX_AUX_BYTE4

AUX Transaction Byte 4 Register.

Address: 0x0106

Direction: RW

Reset: 0x00000000

Table 10-36: DPTX_AUX_BYTE4 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(1) for the next request, or data(4) received in the last reply

DPTX_AUX_BYTE5

AUX Transaction Byte 5 Register.

Address: 0x0107

Direction: RW

Reset: 0x00000000

Table 10-37: DPTX_AUX_BYTE5 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(2) for the next request, or data(5) received in the last reply

DPTX_AUX_BYTE6

AUX Transaction Byte 6 Register.

Address: 0x0108

Direction: RW

Reset: 0x00000000

Table 10-38: DPTX_AUX_BYTE6 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(3) for the next request, or data(6) received in the last reply

DPTX_AUX_BYTE7

AUX Transaction Byte 7 Register.

Address: 0x0109

Direction: RW

Reset: 0x00000000

Table 10-39: DPTX_AUX_BYTE7 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(4) for the next request, or data(7) received in the last reply

DPTX_AUX_BYTE8

AUX Transaction Byte 8 Register.

Address: 0x010a

Direction: RW

Reset: 0x00000000

Table 10-40: DPTX_AUX_BYTE8 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(5) for the next request, or data(8) received in the last reply

DPTX_AUX_BYTE9

AUX Transaction Byte 9 Register.

Address: 0x010b

Direction: RW

Reset: 0x00000000

Table 10-41: DPTX_AUX_BYTE9 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(6) for the next request, or data(9) received in the last reply

DPTX_AUX_BYTE10

AUX Transaction Byte 10 Register.

Address: 0x010c

Direction: RW

Reset: 0x00000000

Table 10-42: DPTX_AUX_BYTE10 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(7) for the next request, or data(10) received in the last reply

DPTX_AUX_BYTE11

AUX Transaction Byte 11 Register.

Address: 0x010d

Direction: RW

Reset: 0x00000000

Table 10-43: DPTX_AUX_BYTE11 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(8) for the next request, or data(11) received in the last reply

DPTX_AUX_BYTE12

AUX Transaction Byte 12 Register.

Address: 0x010e

Direction: RW

Reset: 0x00000000

Table 10-44: DPTX_AUX_BYTE12 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(9) for the next request, or data(12) received in the last reply

DPTX_AUX_BYTE13

AUX Transaction Byte 13 Register.

Address: 0x010f

Direction: RW

Reset: 0x00000000

Table 10-45: DPTX_AUX_BYTE13 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(10) for the next request, or data(13) received in the last reply

DPTX_AUX_BYTE14

AUX Transaction Byte 14 Register.

Address: 0x0110

Direction: RW

Reset: 0x00000000

Table 10-46: DPTX_AUX_BYTE14 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(11) for the next request, or data(14) received in the last reply

DPTX_AUX_BYTE15

AUX Transaction Byte 15 Register.

Address: 0x0111

Direction: RW

Reset: 0x00000000

Table 10-47: DPTX_AUX_BYTE15 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(12) for the next request, or data(15) received in the last reply

DPTX_AUX_BYTE16

AUX Transaction Byte 16 Register.

Address: 0x0112

Direction: RW

Reset: 0x00000000

Table 10-48: DPTX_AUX_BYTE16 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(13) for the next request

DPTX_AUX_BYTE17

AUX Transaction Byte 17 Register.

Address: 0x0113

Direction: RW

Reset: 0x00000000

Table 10-49: DPTX_AUX_BYTE17 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(14) for the next request

DPTX_AUX_BYTE18

AUX Transaction Byte 18 Register.

Address: 0x0114

Direction: RW

Reset: 0x00000000

Table 10-50: DPTX_AUX_BYTE18 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(15) for the next request

DPTX_AUX_RESET

Address: 0x0117

Direction: WO

Reset: 0x00000000

Table 10-51: DPTX_AUX_RESET Bits

Bit	Bit Name	Function
31:1	Unused	
0	CLEAR	Asserting CLEAR resets the AUX Controller state machine: • 0 = No action • 1 = AUX Controller reset

DisplayPort Sink Register Map and DPCD Locations 11

2014.06.30

UG-01131 Subscribe Send Feedback

DisplayPort sink instantiations greatly benefit from and may optionally use an embedded controller (Nios II processor or another controller). This section describes the register map.

Table 11-1: Notation

Shorthand	Definition
RW	Read/write
RO	Read only
WO	Write only
CRO	Clear on read or write, read only
CWO	Clear on read or write, write only

General Registers

This section describes the general registers.

DPRX_RX_CONTROL

The IRQ is asserted when `AUX_IRQ_EN` = 1 and in register `DPRX_AUX_CONTROL` the flag `MSG_READY` = 1. IRQ is de-asserted by setting `AUX_IRQ_EN` to 0 or reading from `DPRX_AUX_COMMAND`. `RECONFIG_LINKRATE` drives the `rx_reconfig_req` . `RX_LINK_RATE` drives `rx_link_rate`.

Address: 0x0000

Direction: RW

Reset: 0x00000000

Table 11-2: DPRX_RX_CONTROL Bits

Bit	Bit Name	Function
31:30	Unused	

Bit	Bit Name	Function
29	LQA_ACTIVE	• 0 = Link Quality Analysis not used • 1 = Link Quality Analysis in progress
28:14	Unused	
13	RECONFIG_LINKRATE	0 = Keep current transceiver link rate 1 = Reconfigure the transceiver with link rate RX_LINK_RATE
12:11	Unused	
10	GXB_RESET	0 = Sink transceiver enabled 1 = Sink transceiver reset
9:8	TP	Current training pattern: • 00 = Normal video • 01 = Training pattern 1 • 10 = Training pattern 2
7	SCRAMBLER_DISABLE	0 = Scrambler enabled 1 = Scrambler disabled
6:5	RX_LINK_RATE	Main link rate: • 0 = 1.62 Gbps • 1 = 2.7 Gbps • 2 = 5.4 Gbps
4:0	LANE_COUNT	Lane count: • 00001 = 1 • 00010 = 2 • 00100 = 4

This register is also available in read-only mode when not using a controller.

Table 11-3: DPRX_RX_CONTROL Bits (Non-Controller Mode)

Bit	Bit Name	Function
31:7	Unused	
6:5	RX_LINK_RATE	Main link rate: • 0 = 1.62 Gbps • 1 = 2.7 Gbps • 2 = 5.4 Gbps

Bit	Bit Name	Function
4:0	LANE_COUNT	Lane count: • 00001 = 1 • 00010 = 2 • 00100 = 4

DPRX_RX_STATUS

GXB_BUSY connects to the rx_reconfig_busy input port.

Address: 0x0001

Direction: CRO

Reset: 0x00000000

Table 11-4: DPRX_RX_STATUS Bits

Bit	Bit Name	Function
31:18	Unused	
17	GXB_BUSY	0 = Transceiver not busy 1 = Transceiver busy
16	SYNC_LOSS	This flag can be reset by writing it to 1: 0 = Symbol lock on all lanes in use 1 = Symbol lock lost on one or more of the used lanes
15:8	Unused	
7	SYM_LOCK3	0 = Symbol unlocked (lane 3) 1 = Symbol locked (lane 3)
6	SYM_LOCK2	0 = Symbol unlocked (lane 2) 1 = Symbol locked (lane 2)
5	SYM_LOCK1	0 = Symbol unlocked (lane 1) 1 = Symbol locked (lane 1)
4	SYM_LOCK0	0 = Symbol unlocked (lane 0) 1 = Symbol locked (lane 0)
3	CR_LOCK3	0 = Clock unlocked (lane 3) 1 = Clock locked (lane 3)
2	CR_LOCK2	0 = Clock unlocked (lane 2) 1 = Clock locked (lane 2)

Bit	Bit Name	Function
1	CR_LOCK1	0 = Clock unlocked (lane 1) 1 = Clock locked (lane 1)
0	CR_LOCK0	0 = Clock unlocked (lane 0) 1 = Clock locked (lane 0)

This register is also available in read-only mode when not using a controller.

Table 11-5: DPRX_RX_STATUS Bits(Non-Controller Mode)

Bit	Bit Name	Function
31:17	Unused	
16	SYNC_LOSS	This flag can be reset by writing it to 1: 0 = Symbol lock on all lanes in use 1 = Symbol lock lost on one or more of the used lanes
15:8	Unused	
7	SYM_LOCK3	0 = Symbol unlocked (lane 3) 1 = Symbol locked (lane 3)
6	SYM_LOCK2	0 = Symbol unlocked (lane 2) 1 = Symbol locked (lane 2)
5	SYM_LOCK1	0 = Symbol unlocked (lane 1) 1 = Symbol locked (lane 1)
4	SYM_LOCK0	0 = Symbol unlocked (lane 0) 1 = Symbol locked (lane 0)
3	CR_LOCK3	0 = Clock unlocked (lane 3) 1 = Clock locked (lane 3)
2	CR_LOCK2	0 = Clock unlocked (lane 2) 1 = Clock locked (lane 2)
1	CR_LOCK1	0 = Clock unlocked (lane 1) 1 = Clock locked (lane 1)
0	CR_LOCK0	0 = Clock unlocked (lane 0) 1 = Clock locked (lane 0)

DPRX_BER_CONTROL

Address: 0x0010

Direction: CRW

Reset: 0x00000000

Note: When PHY_SINK_TEST_LANE_EN equals 1, CR_LOCK and SYM_LOCK bits (register DPRX_RX_STATUS) are forced to 1 for lanes that are not being tested.

Table 11-6: DPRX_BER_CONTROL Bits

Bit	Bit Name	Function
31:23	Unused	
22:21	PHY_SINK_TEST_LANE_SEL	Specifies the lane that is being tested, when PHY_SINK_TEST_LANE_EN is 1, • 00 = Lane 0 • 01 = Lane 1 • 10 = Lane 2 • 11 = Lane 3
20	PHY_SINK_TEST_LANE_EN	Writing this bit at 1 enables single lane PHY test, Write 0 to disable single lane PHY test.
19	RST3	Writing this bit at 1 resets the lane 3 bit-error counter in register DPRX_BER_CNT1. Always reads as 0.
18	RST2	Writing this bit at 1 resets the lane 2 bit-error counter in register DPRX_BER_CNT1. Always reads as 0.
17	RST1	Writing this bit at 1 resets lane 1 bit-error counter in register DPRX_BER_CNT0. Always reads as 0.
16	RST0	Writing this bit at 1 resets lane 0 bit-error counter in register DPRX_BER_CNT0. Always reads as 0.
15:14	Unused	
13:11	PATT3	Pattern selection for lane 3: • 000 = No training pattern (normal mode) • 011 = PRBS7 • 101 = HBR2CPAT
10:8	PATT2	Pattern selection for lane 2: • 000 = No training pattern (normal mode) • 011 = PRBS7 • 101 = HBR2CPAT

Bit	Bit Name	Function
7:5	PATT1	Pattern selection for lane 1: • 000 = No training pattern (normal mode) • 011 = PRBS7 • 101 = HBR2CPAT
4:2	PATT0	Pattern selection for lane 0: • 000 = No training pattern (normal mode) • 011 = PRBS7 • 101 = HBR2CPAT
1:0	CNTSEL	Count selection: • 00 = Disparity and illegal comma codes • 01 = Disparity • 10 = Illegal comma codes • 11 = Reserved

DPRX_BER_CNT0

Bit-error counters for lane 0 and lane 1.

Address: 0x0003

Direction: RO

Reset: 0x00000000

Table 11-7: DPRX_BER_CNT0 Bits

Bit	Bit Name	Function
31	Unused	
30:16	CNT1	Symbol error counter for lane 1
15	Unused	
14:0	CNT0	Symbol error counter for lane 0

DPRX_BER_CNT1

Bit-error counter register for lane 2 and lane 3.

Address: 0x0004

Direction: RO

Reset: 0x00000000

Table 11-8: DPRX_BER_CNT1 Bits

Bit	Bit Name	Function
31	Unused	
30:16	CNT3	Symbol error counter for lane 3
15	Unused	
14:0	CNT2	Symbol error counter for lane 2

Timestamp

The Nios II processor can use this global, free-running counter to generate timestamps and delays. The same counter is used in both sink and source instantiations (DPRX_TIMESTAMP is always equal to DPTX_TIMESTAMP).

Address: 0x0005

Direction: RO

Reset: 0x00000000

Table 11-9: DPRX_TIMESTAMP Bits

Bit	Bit Name	Function
31:24	Unused	8'b00000000
23:0	TIMESTAMP	Free-running counter value (1 tick equals 100 µs)

MSA Registers

The MSA registers are allocated at addresses:

- 0x0020 through 0x002f for Stream0
- 0x0040 through 0x004f for Stream1
- 0x0060 through 0x006f for Stream2
- 0x0080 through 0x008f for Stream3

Note: Only registers for Stream0 are listed in the following sections. Registers for Stream0 are also available in non-controller mode.

DPRX0_MSA_MVID

Address: 0x0020

Direction: RO

Reset: 0x00000000

Table 11-10: DPRX0_MSA_MVID Bits

Bit	Bit Name	Function
31:24	Unused	
23:0	MVID	Main stream attribute MVID

DPRX0_MSA_NVID

Address: 0x0021

Direction: RO

Reset: 0x00000000

Table 11-11: DPRX0_MSA_NVID Bits

Bit	Bit Name	Function
31:24	Unused	
23:0	NVID	Main stream attribute NVID

DPRX0_MSA_HTOTAL

Address: 0x0022

Direction: RO

Reset: 0x00000000

Table 11-12: DPRX0_MSA_HTOTAL Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	HTOTAL	Main stream attribute HTOTAL

DPRX0_MSA_VTOTAL

Address: 0x0023

Direction: RO

Reset: 0x00000000

Table 11-13: DPRX0_MSA_VTOTAL Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	MVID	Main stream attribute VTOTAL

DPRX0_MSA_HSP

MSA horizontal synchronization polarity register, DPRX0_MSA_HSP.

Address: 0x0024

Direction: RO

Reset: 0x00000000

Table 11-14: DPRX0_MSA_HSP Bits

Bit	Bit Name	Function
31:1	Unused	
0	HSP	Main stream attribute horizontal synchronization polarity • 0 = Positive • 1 = Negative

DPRX0_MSA_HSW

MSA horizontal synchronization width register, DPRX0_MSA_HSW.

Address: 0x0025

Direction: RO

Reset: 0x00000000

Table 11-15: DPRX0_MSA_HSW Bits

Bit	Bit Name	Function
31:15	Unused	
14:0	HSW	Main stream attribute horizontal synchronization width

DPRX0_MSA_HSTART

Address: 0x0026

Direction: RO

Reset: 0x00000000

Table 11-16: DPRX0_MSA_HSTART Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	HSTART	Main stream attribute HSTART

DPRX0_MSA_VSTART

Address: 0x0027

Direction: RO

Reset: 0x00000000

Table 11-17: DPRX0_MSA_VSTART Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	VSTART	Main stream attribute VSTART

DPRX0_MSA_VSP

MSA vertical synchronization polarity register, DPRX0_MSA_VSP.

Address: 0x0028

Direction: RO

Reset: 0x00000000

Table 11-18: DPRX0_MSA_VSP Bits

Bit	Bit Name	Function
31:1	Unused	
0	VSP	Main stream attribute vertical synchronization polarity • 0 = Positive • 1 = Negative

DPRX0_MSA_VSW

MSA vertical synchronization width register, DPRX0_MSA_VSW.

Address: 0x0029

Direction: RO

Reset: 0x00000000

Table 11-19: DPRX0_MSA_VSW Bits

Bit	Bit Name	Function
31:15	Unused	
14:0	VSW	Main stream attribute vertical synchronization width

DPRX0_MSA_HWIDTH

TX control register, DPRX0_MSA_HWIDTH.

Address: 0x002a

Direction: RO

Reset: 0x00000000

Table 11-20: DPRX0_MSA_HWIDTH Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	HWIDTH	Main stream attribute HWIDTH

DPRX0_MSA_VHEIGHT

Address: 0x002b

Direction: RO

Reset: 0x00000000

Table 11-21: DPRX0_MSA_WHEIGHT Bits

Bit	Bit Name	Function
31:16	Unused	
15:0	VHEIGHT	Main stream attribute VHEIGHT

DPRX0_MSA_MISC0

Address: 0x002c

Direction: RO

Reset: 0x00000000

Table 11-22: DPRX0_MSA_MISC0 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	MISC0	Main stream attribute MISC0

DPRX0_MSA_MISC1

Address: 0x002d

Direction: RO

Reset: 0x00000000

Table 11-23: DPRX0_MSA_MISC1 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	MISC1	Main stream attribute MISC1

DPRX0_VBID

VB-ID register, DPRX0_VBID.

Address: 0x002e

Direction: RO

Reset: 0x00000000

Table 11-24: DPRX0_VBID Bits

Bit	Bit Name	Function
31:8	Unused	
7	MSA_LOCK	0 = MSA unlocked 1 = MSA locked (on all lanes)
6	VBID_LOCK	0 = VB-ID unlocked 1 = VB-ID locked (on all lanes)
5:0	VBID	VB-ID flags (refer to the DisplayPort specification)

Audio Registers

The audio registers are allocated at addresses:

- 0x0020 through 0x003f for Stream0
- 0x0040 through 0x005f for Stream1
- 0x0060 through 0x007f for Stream2
- 0x0080 through 0x009f for Stream3

Note: Only registers for Stream0 are listed in the following sections.

DPRX0_AUD_MAUD

Received audio Maud register, DPRX0_AUD_MAUD.

Address: 0x0030

Direction: RO

Reset: 0x00000000

Table 11-25: DPRX0_AUD_MAUD Bits

Bit	Bit Name	Function
31:24	Unused	
23:0	MAUD	Received audio Maud

DPRX0_AUD_NAUD

Received audio Naud register, DPRX0_AUD_NAUD.

Address: 0x0031

Direction: RO

Reset: 0x00000000

Table 11-26: DPRX0_AUD_NAUD Bits

Bit	Bit Name	Function
31:24	Unused	
23:0	NAUD	Received audio Naud

DPRX0_AUD_AIF0

Received audio InfoFrame register, DPRX0_AUD_AIF0.

Address: 0x0032

Direction: RO

Reset: 0x00000000

Table 11-27: DPRX0_AUD_AIF0 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	AIF	Received audio InfoFrame byte 0 (refer to CEA-861-E specification)

DPRX0_AUD_AIF1

Received audio InfoFrame register, DPRX0_AUD_AIF1.

Address: 0x0033

Direction: RO

Reset: 0x00000000

Table 11-28: DPRX0_AUD_AIF1 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	AIF	Received audio InfoFrame byte 1 (refer to CEA-861-E specification)

DPRX0_AUD_AIF2

Received audio InfoFrame register, DPRX0_AUD_AIF2.

Address: 0x0034

Direction: RO

Reset: 0x00000000

Table 11-29: DPRX0_AUD_AIF2 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	AIF	Received audio InfoFrame byte 2 (refer to CEA-861-E specification)

DPRX0_AUD_AIF3

Received audio InfoFrame register, DPRX0_AUD_AIF3.

Address: 0x0035

Direction: RO

Reset: 0x00000000

Table 11-30: DPRX0_AUD_AIF3 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	AIF	Received audio InfoFrame byte 3 (refer to CEA-861-E specification)

DPRX0_AUD_AIF4

Received audio InfoFrame register, DPRX0_AUD_AIF4.

Address: 0x0036

Direction: R0

Reset: 0x00000000

Table 11-31: DPRX0_AUD_AIF4 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	AIF	Received audio InfoFrame byte 4 (refer to CEA-861-E specification)

AUX Controller Interface

The following sections describe the registers for the AUX Controller interface.

DPRX_AUX_CONTROL

For transaction requests:

1. Wait for MSG_READY (in register DPRX_AUX_STATUS) to be 1, or enable the interrupt with AUX_IRQ_EN and wait for the interrupt request.
2. Read the transaction request total length from LENGTH.
3. Read the transaction request command from DPRX_AUX_COMMAND., which clears MSG_READY and LENGTH.
4. Read the transaction request data payload from registers DPRX_AUX_BYTE0 to DPRX_AUX_BYTE15 (read LENGTH - 1 bytes).

For transaction replies:

1. Wait for READY_TO_TX (in register DPRX_AUX_STATUS) to be 1. Implement a timeout.
2. Write registers DPRX_AUX_COMMAND to DPRX_AUX_BYTE18 with transaction command and data payload.
3. Write LENGTH with the transaction total message length (1 to 17, 1 for the command plus 1 to 16 for the data payload) and set TX_STROBE to 1. This sequence starts the reply transmission.

The sink asserts the IRQ when AUX_IRQ_EN = 1 and MSG_READY = 1. To deassert IRQ, set AUX_IRQ_EN to 0 or read from DPRX_AUX_COMMAND.

Address: 0x0100

Direction: RW

Reset: 0x00000000

Table 11-32: DPRX_AUX_CONTROL Bits

Bit	Bit Name	Function
31	MSG_READY	0 = Waiting for a request 1 = A request has been completely received
30	READY_TO_TX	0 = Busy sending a reply or request waiting 1 = Ready to send a reply
29:9	Unused	

Bit	Bit Name	Function
8	AUX_IRQ_EN	Issues an IRQ to Nios II processor when the sink receives an AUX channel transaction from the source. 0 = Disable 1 = Enable
7	TX_STROBE	Writing this bit at 1 starts a reply transmission. Always read this bit as 0.
6:5	Unused	
4:0	LENGTH	For the next transaction reply, total length of message to be transmitted (1 – 17), for the last received transaction request, total length of message received (1 – 17).

DPRX_AUX_STATUS

AUX transaction status register, DPRX_AUX_STATUS.

Address: 0x0101

Direction: RO

Reset: 0x00000000

Table 11-33: DPRX_AUX_STATUS Bits

Bit	Bit Name	Function
31	MSG_READY	0 = Waiting for a request 1 = Receives a request
30	READY_TO_TX	0 = Busy sending a reply or waiting for a request 1 = Ready to send a reply
29:2	Unused	
1	SRC_PWR_DETECT	0 = Upstream power not detected 1 = Upstream power detected
0	SRC_CABLE_DETECT	0 = Upstream cable not detected 1 = Upstream cable detected

DPRX_AUX_COMMAND

AUX transaction command register, DPRX_AUX_COMMAND.

Address: 0x0102

Direction: RW

Reset: 0x00000000

Table 11-34: DPRX_AUX_COMMAND Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	COMMAND	AUX transaction command for the next reply or received in the last request (refer to the DisplayPort specification) . Reading of this register clears MSG_READY and LENGTH in DPRX_AUX_CONTROL register.

DPRX_AUX_BYTE0

AUX Transaction Byte 0 Register.

Address: 0x0103

Direction: RW

Reset: 0x00000000

Table 11-35: DPRX_AUX_BYTE0 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction address[15:8] received in the last request, or data(0) for the next reply

DPRX_AUX_BYTE1

AUX Transaction Byte 1 Register.

Address: 0x0104

Direction: RW

Reset: 0x00000000

Table 11-36: DPRX_AUX_BYTE1 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction address[7:1] received in the last request, or data(1) for the next reply

DPRX_AUX_BYTE2

AUX Transaction Byte 2 Register.

Address: 0x0105

Direction: RW

Reset: 0x00000000

Table 11-37: DPRX_AUX_BYTE2 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction length[3:0] received in the last request, or data(2) for the next reply (refer to DisplayPort specification)

DPRX_AUX_BYTE3

AUX Transaction Byte 3 Register.

Address: 0x0106

Direction: RW

Reset: 0x00000000

Table 11-38: DPRX_AUX_BYTE3 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(0) received in the last request, or data(3) for the next reply

DPRX_AUX_BYTE4

AUX Transaction Byte 4 Register.

Address: 0x0107

Direction: RW

Reset: 0x00000000

Table 11-39: DPRX_AUX_BYTE4 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(1) received in the last request, or data(4) for the next reply

DPRX_AUX_BYTE5

AUX Transaction Byte 5 Register.

Address: 0x0108

Direction: RW

Reset: 0x00000000

Table 11-40: DPRX_AUX_BYTE5 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BY T E	Transaction data(2) received in the last request, or data(5) for the next reply

DPRX_AUX_BYTE6

AUX Transaction Byte 6 Register.

Address: 0x0109

Direction: RW

Reset: 0x00000000

Table 11-41: DPRX_AUX_BYTE6 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(3) received in the last request, or data(6) for the next reply

DPRX_AUX_BYTE7

AUX Transaction Byte 7 Register.

Address: 0x010a

Direction: RW

Reset: 0x00000000

Table 11-42: DPRX_AUX_BYTE7 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(4) received in the last request, or data(7) for the next reply

DPRX_AUX_BYTE8

AUX Transaction Byte 8 Register.

Address: 0x010b

Direction: RW

Reset: 0x00000000

Table 11-43: DPRX_AUX_BYTE8 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(5) received in the last request, or data(8) for the next reply

DPRX_AUX_BYTE9

AUX Transaction Byte 9 Register.

Address: 0x010c

Direction: RW

Reset: 0x00000000

Table 11-44: DPRX_AUX_BYTE9 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(6) received in the last request, or data(9) for the next reply

DPRX_AUX_BYTE10

AUX Transaction Byte 10 Register.

Address: 0x010d

Direction: RW

Reset: 0x00000000

Table 11-45: DPRX_AUX_BYTE10 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(7) received in the last request, or data(10) for the next reply

DPRX_AUX_BYTE11

AUX Transaction Byte 11 Register.

Address: 0x010e

Direction: RW

Reset: 0x00000000

Table 11-46: DPRX_AUX_BYTE11 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(8) received in the last request, or data(11) for the next reply

DPRX_AUX_BYTE12

AUX Transaction Byte 12 Register.

Address: 0x010f

Direction: RW

Reset: 0x00000000

Table 11-47: DPRX_AUX_BYTE12 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(9) received in the last request, or data(12) for the next reply

DPRX_AUX_BYTE13

AUX Transaction Byte 13 Register.

Address: 0x0110

Direction: RW

Reset: 0x00000000

Table 11-48: DPRX_AUX_BYTE13 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(10) received in the last request, or data(13) for the next reply

DPRX_AUX_BYTE14

AUX Transaction Byte 14 Register.

Address: 0x0111

Direction: RW

Reset: 0x00000000

Table 11-49: DPRX_AUX_BYTE14 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(11) received in the last request, or data(14) for the next reply

DPRX_AUX_BYTE15

AUX Transaction Byte 15 Register.

Address: 0x0112

Direction: RW

Reset: 0x00000000

Table 11-50: DPRX_AUX_BYTE15 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(12) received in the last request, or data(15) for the next reply

DPRX_AUX_BYTE16

AUX Transaction Byte 16 Register.

Address: 0x0113

Direction: RW

Reset: 0x00000000

Table 11-51: DPRX_AUX_BYTE16 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(13) received in the last request

DPRX_AUX_BYTE17

AUX Transaction Byte 17 Register.

Address: 0x0114

Direction: RW

Reset: 0x00000000

Table 11-52: DPRX_AUX_BYTE17 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(14) received in the last request

DPRX_AUX_BYTE18

AUX Transaction Byte 18 Register.

Address: 0x0115

Direction: RW

Reset: 0x00000000

Table 11-53: DPRX_AUX_BYTE18 Bits

Bit	Bit Name	Function
31:8	Unused	
7:0	BYTE	Transaction data(15) received in the last request

DPRX_AUX_I2C0

AUX to I2C0 management. The sink routes all AUX channel accesses to I2C slave addresses of values between START_ADDR and END_ADDR to I2C0.

Address: 0x0116

WO

0x00000000

Table 11-54: DPRX_AUX_I2C0 Bits

Bit	Bit Name	Function
31:15	Unused	
14:8	END_ADDR	I2C slave end address
7	Unused	
6:0	START_ADDR	I2C slave start address

DPRX_AUX_I2C1

AUX to I2C1 management. The sink routes all AUX channel accesses to I2C slave addresses of values between START_ADDR and END_ADDR to I2C1.

Address: 0x0117

WO

0x00000000

Table 11-55: DPRX_AUX_I2C1 Bits

Bit	Bit Name	Function
31:15	Unused	
14:8	END_ADDR	I2C slave end address
7	Unused	
6:0	START_ADDR	I2C slave start address

DPRX_AUX_RESET

Address: 0x0118

Direction: WO

Reset: 0x00000000

Table 11-56: DPRX_AUX_RESET Bits

Bit	Bit Name	Function
31:1	Unused	
0	CLEAR	Asserting CLEAR resets the AUX controller state machine: • 0 = No action • 1 = AUX Controller reset

DPRX_AUX_HPD

HPD control.

Address: 0x0119

Direction: RW

Reset: 0x00000000

Table 11-57: DPRX_AUX_HPD Bits

Bit	Bit Name	Function
31:13	Unused	
12	HPD_IRQ	Writing this bit at 1 generates a 0.75-ms long HPD IRQ (low pulse). This bit is WO. To use this bit, HPD_EN must be 1.

Bit	Bit Name	Function
11	HPD_EN	HPD logic level 0 = Deasserted (low) 1 = Asserted (high)
10:0	Unused	

Sink-Supported DPCD Locations

The following table describes the DPCD locations (or location groups) that are supported in DisplayPort sink instantiations.

Table 11-58: DPCD Locations

Location Name	Address	Without Controller	With Controller
DPCD_REV	0x0000	Yes	Yes
MAX_LINK_RATE	0x0001	Yes	Yes
MAX_LANE_COUNT	0x0002	Yes	Yes
MAX_DOWNSPREAD	0x0003	Yes	Yes
NORP	0x0004	Yes	Yes
DOWNSTREAMPORT_PRESENT	0x0005	Yes	Yes
MAIN_LINK_CHANNEL_CODING	0x0006	Yes	Yes
DOWN_STREAM_PORT_COUNT	0x0007	Yes	Yes
RECEIVE_PORT0_CAP_0	0x0008	Yes	Yes
RECEIVE_PORT0_CAP_1	0x0009	Yes	Yes
RECEIVE_PORT1_CAP_0	0x000A	Yes	Yes
RECEIVE_PORT1_CAP_1	0x000B	Yes	Yes
I2C_SPEED_CONTROL	0x000C	—	Yes
EDP_CONFIGURATION_CAP	0x000D	—	Yes
TRAINING_AUX_RD_INTERVAL	0x000E	—	Yes
ADAPTER_CAP	0x000F	—	Yes
FAUX_CAP	0x0020	—	Yes
MST_CAP	0x0021	—	Yes

Location Name	Address	Without Controller	With Controller
NUMBER_OF_AUDIO_ENDPOINTS	0x0022	—	Yes
GUID	0x0030	—	Yes
DWN_STRM_PORTX_CAP	0x0080	Yes	Yes
LINK_BW_SET	0x0100	Yes	Yes
LANE_COUNT_SET	0x0101	Yes	Yes
TRAINING_PATTERN_SET	0x0102	Yes	Yes
TRAINING_LANE0_SET	0x0103	Yes	Yes
TRAINING_LANE1_SET	0x0104	Yes	Yes
TRAINING_LANE2_SET	0x0105	Yes	Yes
TRAINING_LANE3_SET	0x0106	Yes	Yes
DOWNSPREAD_CTRL	0x0107	Yes	Yes
MAIN_LINK_CHANNEL_CODING_SET	0x0108	Yes	Yes
I2C_SPEED_CONTROL	0x0109	—	Yes
EDP_CONFIGURATION_SET	0x010A	—	Yes
LINK_QUAL_LANE0_SET	0x010B	—	Yes
LINK_QUAL_LANE1_SET	0x010C	—	Yes
LINK_QUAL_LANE2_SET	0x010D	—	Yes
LINK_QUAL_LANE3_SET	0x010E	—	Yes
TRAINING_LANE0_1_SET2	0x010F	—	Yes
TRAINING_LANE2_3_SET2	0x0110	—	Yes
MSTM_CTRL	0x0111	—	Yes
AUDIO_DELAY[7:0]	0x0112	—	Yes
AUDIO_DELAY[15:8]	0x0113	—	Yes
AUDIO_DELAY[23:6]	0x0114	—	Yes
ADAPTER_CTRL	0x01A0	—	Yes
BRANCH_DEVICE_CTRL	0x01A1	—	Yes
PAYLOAD_ALLOCATE_SET	0x01C0	—	Yes
PAYLOAD_ALLOCATE_START_TIME_SLOT	0x01C1	—	Yes

Location Name	Address	Without Controller	With Controller
PAYLOAD_ALLOCATE_TIME_SLOT_COUNT	0x01C2	—	Yes
SINK_COUNT	0x0200	Yes	Yes
DEVICE_SERVICE_IRQ_VECTOR	0x0201	Yes	Yes
LANE0_1_STATUS	0x0202	Yes	Yes
LANE2_3_STATUS	0x0203	Yes	Yes
LANE_ALIGN_STATUS_UPDATED	0x0204	Yes	Yes
SINK_STATUS	0x0205	Yes	Yes
ADJUST_REQUEST_LANE0_1	0x0206	Yes	Yes
ADJUST_REQUEST_LANE2_3	0x0207	Yes	Yes
SYMBOL_ERROR_COUNT_LANE0	0x0210	Yes	Yes
SYMBOL_ERROR_COUNT_LANE1	0x0212	Yes	Yes
SYMBOL_ERROR_COUNT_LANE2	0x0214	Yes	Yes
SYMBOL_ERROR_COUNT_LANE3	0x0216	Yes	Yes
TEST_LANE_COUNT	0x0220	Yes	—
TEST_PATTERN	0x0221	Yes	—
TEST_H_TOTAL_LSB	0x0222	Yes	—
TEST_H_TOTAL_MSB	0x0223	Yes	—
TEST_V_TOTAL_LSB	0x0224	Yes	—
TEST_V_TOTAL_MSB	0x0225	Yes	—
TEST_H_START_LSB	0x0226	Yes	—
TEST_H_START_MSB	0x0227	Yes	—
TEST_V_START_LSB	0x0228	Yes	—
TEST_V_START_MSB	0x0229	Yes	—
TEST_HSYNC_LSB	0x022A	Yes	—
TEST_HSYNC_MSB	0x022B	Yes	—
TEST_VSYNC_LSB	0x022C	Yes	—
TEST_VSYNC_MSB	0x022D	Yes	—
TEST_H_WIDTH_LSB	0x022E	Yes	—

Location Name	Address	Without Controller	With Controller
TEST_H_WIDTH_MSB	0x022F	Yes	—
TEST_V_HEIGHT_LSB	0x0230	Yes	—
TEST_V_HEIGHT_MSB	0x0231	Yes	—
TEST_MISC_LSB	0x0232	Yes	—
TEST_MISC_MSB	0x0233	Yes	—
TEST_REFRESH_RATE_NUMERATOR	0x0234	Yes	—
TEST_CRC_R_Cr	0x0240	Yes	—
TEST_CRC_G_Y	0x0242	Yes	—
TEST_CRC_B_Cb	0x0244	Yes	—
TEST_SINK_MISC	0x0246	Yes	—
TEST_RESPONSE	0x0260	Yes	—
TEST_EDID_CHECKSUM	0x0261	Yes	—
TEST_SINK	0x0270	Yes	Yes
PAYLOAD_TABLE_UPDATE_STATUS	0x02C0	—	Yes
VC_PAYLOAD_ID_SLOT_1 to _63	0x02C1	—	Yes
IEEE_OUI	0x0300	—	Yes
IEEE_OUI	0x0301	—	Yes
IEEE_OUI	0x0302	—	Yes
DEVICE_IDENTIFICATION_STRING	0x0303	—	Yes
HARDWARE_REVISION	0x0309	—	Yes
FWSW_MAJOR	0x030A	—	Yes
FWSW_MINOR	0x030B	—	Yes
RESERVED	0x030C	—	Yes
RESERVED	0x030D	—	Yes
RESERVED	0x030E	—	Yes
RESERVED	0x030F	—	Yes
IEEE_OUI	0x0400	—	Yes
IEEE_OUI	0x0401	—	Yes

Location Name	Address	Without Controller	With Controller
IEEE_OUI	0x0402	—	Yes
DEVICE_IDENTIFICATION_STRING	0x0403	—	Yes
HARDWARE_REVISION	0x0409	—	Yes
FWSW_MAJOR	0x040A	—	Yes
FWSW_MINOR	0x040B	—	Yes
RESERVED (0x040C to 0x04FF)	0x040C	—	Yes
IEEE_OUI	0x0500	Yes	Yes
IEEE_OUI	0x0501	Yes	Yes
IEEE_OUI	0x0502	Yes	Yes
DEVICE_IDENTIFICATION_STRING	0x0503	—	Yes
HARDWARE_REVISION	0x0509	—	Yes
FWSW_MAJOR	0x050A	—	Yes
FWSW_MINOR	0x050B	—	Yes
RESERVED (0x050C to 0x05FF)	0x050C	—	Yes
SET_POWER_STATE	0x0600	Yes	Yes
DOWN_REQ (0x1000 to 0x102F)	0x1000	—	Yes
DOWN_REP (0x1400 to 0x142F)	0x1400	—	Yes
SINK_COUNT_ESI	0x2002	—	Yes
DEVICE_SERVICE_IRQ_VECTOR_ESI0	0x2003	—	Yes
DEVICE_SERVICE_IRQ_VECTOR_ESI1	0x2004	—	Yes
LINK_SERVICE_IRQ_VECTOR_ESI0	0x2005	—	Yes
LANE0_1_STATUS	0x200C	—	Yes
LANE2_3_STATUS_ESI	0x200D	—	Yes
LANE_ALIGN STATUS_UPDATED_ESI	0x200E	—	Yes
SINK_STATUS_ESI	0x200F	—	Yes

Additional Information 12

2014.06.30

UG-01131 Subscribe Send Feedback

Document Revision History

The following table lists the revision history for this document.

Table 12-1: Document Revision History

Date	Version	Changes

June 2014	14.0	• Native PHY is removed from the IP core; included information about how to instantiate the PHY outside the DisplayPort IP core.
		• Updated the source and sink block diagrams.
		• Updated the source and sink register map information.
		• Added new sink register bits:
		  • `LQA ACTIVE`
		  • `PHY_SINK_TEST_LANE_SEL`
		  • `PHY_SINK_TEST_LANE_EN`
		  • `AUX_IRQ_EN`
		  • `TX_STROBE`
		  • `DPRX_AUX_STATUS` bits
		  • `DPRX_AUX_I2C0` bits
		  • `DPRX_AUX_I2C0` bits
		  • `DPRX_AUX_HPD` bits
		• Removed these sink register bits:
		  • `HPD_IRQ`
		  • `HPD_EN`
		  • `DPRX_AUX_IRQ_EN` bits
		• Added a new source register bit:
		  • `VTOTAL`
		• Added source TX transceiver interface signals
		• Removed these source signals:
		  • `xcvr_refclk`
		  • `tx_serial_data`
		  • `xcvr_reconfig`
		• Added sink audio and RX transceiver interface signals.
		• Removed these sink signals:
		  • `xcvr_refclk`
		  • `rx_serial_data`
		  • `xcvr_reconfig`
		• Added information about Transceiver Reconfiguration Interface for source and sink.
		• Added information about single clock reference (135MHz) for source and sink.
		• Added information about Bitec HSMC DisplayPort daughter card in the *Hardware Demonstration* chapter.
		• Updated the API reference.

Date	Version	Changes
November 2013	13.1	• Updated the source and sink register map information. • Added dual and quad pixel mode support. • Added support for quad symbol (40-bit) transceiver data interface. • Added support for Cyclone V devices. • Added HBR2 support for Arria V and Arria V GZ devices. • Added information about eDP support. • Updated the API reference.
May 2013	13.0	• Added information on audio support. • Added HBR2 support for Stratix V devices. • Added information on secondary data support.
February 2013	12.1 SP1 (Beta)	Second beta release: • Updated the filenames for the hardware demonstration and simulation example. • Added chapter describing the IP core's compilation example. • Miscellaneous updates.
December 2012	12.1 (Beta)	Initial beta release.

How to Contact Altera

Table 12-2: How to Contact Altera

To locate the most up-to-date information about Altera products, refer to this table. You can also contact your local Altera sales office or sales representative.

Contact	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support: general	Email	nacomp@altera.com
Nontechnical support: software licensing	Email	authorization@altera.com

Related Information

- www.altera.com/support
- www.altera.com/training
- custrain@altera.com

- www.altera.com/literature
- nacomp@altera.com
- authorization@altera.com