



# DisplayPort Intel® FPGA IP User Guide

Updated for Intel® Quartus® Prime Design Suite: **19.1**



**Subscribe**

**Send Feedback**

**UG-01131 | 2019.04.01**

Latest document on the web: [PDF](#) | [HTML](#)



## Contents

---

<b>1. DisplayPort Intel® FPGA IP Quick Reference.....</b>	<b>9</b>
1.1. DisplayPort Terms and Acronyms.....	10
<b>2. About This IP Core.....</b>	<b>13</b>
2.1. Device Family Support.....	14
2.2. IP Core Verification.....	15
2.3. Performance and Resource Utilization.....	15
<b>3. Getting Started.....</b>	<b>18</b>
3.1. Installing and Licensing Intel FPGA IP Cores.....	18
3.1.1. Intel FPGA IP Evaluation Mode.....	19
3.2. Specifying IP Core Parameters and Options.....	21
3.3. Simulating the Design.....	21
3.3.1. Simulating with the ModelSim Simulator.....	22
3.4. Compiling the Full Design and Programming the FPGA.....	22
<b>4. DisplayPort Intel FPGA IP Core Hardware Design Examples.....</b>	<b>23</b>
4.1. DisplayPort Intel FPGA IP Hardware Design Examples for Intel Arria 10, Intel Cyclone 10 GX, and Intel Stratix 10 Devices.....	23
4.2. DisplayPort Intel FPGA IP Hardware Design Examples for Arria V, Cyclone V, and Stratix V Devices.....	23
4.2.1. Clock Recovery Core.....	27
4.2.2. Transceiver and Clocking.....	32
4.2.3. Required Hardware.....	34
4.2.4. Design Walkthrough.....	34
4.2.5. DisplayPort Link Training Flow.....	40
4.2.6. DisplayPort Post Link Training Adjust Request Flow (LQA).....	41
4.2.7. DisplayPort MST Source User Application.....	42
<b>5. DisplayPort Source.....</b>	<b>44</b>
5.1. Main Data Path.....	45
5.1.1. Video Packetizer Path.....	45
5.1.2. Video Geometry Measurement Path.....	46
5.1.3. Audio and Secondary Stream Encoder Path.....	46
5.1.4. Training and Link Quality Patterns Generator.....	47
5.2. Controller Interface.....	47
5.3. Sideband Channel.....	47
5.4. Source Embedded DisplayPort (eDP) Support.....	48
5.5. Source Interfaces.....	48
5.5.1. Controller Interface.....	52
5.5.2. AUX Interface.....	53
5.5.3. Video Interface.....	53
5.5.4. TX Transceiver Interface.....	57
5.5.5. Transceiver Reconfiguration Interface.....	57
5.5.6. Transceiver Analog Reconfiguration Interface.....	58
5.5.7. Secondary Stream Interface.....	58
5.5.8. Audio Interface.....	62
5.6. Source Clock Tree.....	65



<b>6. DisplayPort Sink.....</b>	<b>67</b>
6.1. Sink Embedded DisplayPort (eDP) Support.....	69
6.2. Sink Interfaces.....	69
6.2.1. Controller Interface.....	74
6.2.2. AUX Interface.....	74
6.2.3. Debugging Interface.....	75
6.2.4. Video Interface.....	76
6.2.5. Clocked Video Input Interface.....	78
6.2.6. RX Transceiver Interface.....	79
6.2.7. Transceiver Reconfiguration Interface.....	80
6.2.8. Secondary Stream Interface.....	81
6.2.9. Audio Interface.....	83
6.2.10. MSA Interface.....	84
6.3. Sink Clock Tree.....	86
<b>7. DisplayPort Intel FPGA IP Parameters.....</b>	<b>88</b>
7.1. DisplayPort Intel FPGA IP Source Parameters.....	88
7.2. DisplayPort Intel FPGA IP Sink Parameters .....	89
<b>8. DisplayPort Intel FPGA IP Core Simulation Example.....</b>	<b>91</b>
8.1. Design Walkthrough.....	91
8.1.1. Copy the Simulation Files to Your Working Directory.....	92
8.1.2. Generate the IP Simulation Files and Scripts, and Compile and Simulate.....	93
8.1.3. View the Results.....	94
<b>9. DisplayPort API Reference.....</b>	<b>97</b>
9.1. Using the Library.....	97
9.2. btc_dprx_syslib API Reference.....	99
9.3. btc_dprx_aux_get_request.....	99
9.4. btc_dprx_aux_handler.....	100
9.5. btc_dprx_aux_post_reply.....	101
9.6. btc_dprx_baseaddr.....	101
9.7. btc_dprx_dpcd_gpu_access.....	101
9.8. btc_dprx_edid_set.....	102
9.9. btc_dprx_hpd_get.....	102
9.10. btc_dprx_hpd_pulse.....	103
9.11. btc_dprx_hpd_set.....	103
9.12. btc_dprx_lt_eyeq_init.....	104
9.13. btc_dprx_lt_force.....	104
9.14. btc_dprx_rtl_ver.....	105
9.15. btc_dprx_sw_ver.....	105
9.16. btc_dprx_syslib_add_rx.....	105
9.17. btc_dprx_syslib_info.....	106
9.18. btc_dprx_syslib_init.....	106
9.19. btc_dprx_syslib_monitor.....	107
9.20. btc_dptx_syslib API Reference.....	107
9.21. btc_dptx_aux_i2c_read.....	107
9.22. btc_dptx_aux_i2c_write.....	108
9.23. btc_dptx_aux_read.....	109
9.24. btc_dptx_aux_write.....	109
9.25. btc_dptx_baseaddr.....	110



9.26. btc_dptx_edid_block_read.....	110
9.27. btc_dptx_edid_read.....	111
9.28. btc_dptx_fast_link_training.....	111
9.29. btc_dptx_hpd_change.....	112
9.30. btc_dptx_is_link_up.....	112
9.31. btc_dptx_link_bw.....	112
9.32. btc_dptx_link_training.....	113
9.33. btc_dptx_rtl_ver.....	113
9.34. btc_dptx_set_color_space.....	113
9.35. btc_dptx_sw_ver.....	114
9.36. btc_dptx_syslib_add_tx.....	114
9.37. btc_dptx_syslib_init.....	115
9.38. btc_dptx_syslib_monitor.....	115
9.39. btc_dptx_test_autom.....	115
9.40. btc_dptx_video_enable.....	116
9.41. btc_dptx_mst_allocate_payload_rep.....	116
9.42. btc_dptx_mst_allocate_payload_req.....	117
9.43. btc_dptx_mst_clear_payload_table_rep.....	117
9.44. btc_dptx_mst_clear_payload_table_req.....	118
9.45. btc_dptx_mst_conn_stat_notify_req.....	118
9.46. btc_dptx_mst_down_rep_irq.....	118
9.47. btc_dptx_mst_enable.....	119
9.48. btc_dptx_mst_enum_path_rep.....	119
9.49. btc_dptx_mst_enum_path_req.....	120
9.50. btc_dptx_mst_get_msg_transact_ver_rep.....	120
9.51. btc_dptx_mst_get_msg_transact_ver_req.....	121
9.52. btc_dptx_mst_link_address_rep.....	121
9.53. btc_dptx_mst_link_address_req.....	122
9.54. btc_dptx_mst_remote_dpccd_wr_rep.....	122
9.55. btc_dptx_mst_remote_dpccd_wr_req.....	123
9.56. btc_dptx_mst_remote_i2c_rd_rep.....	123
9.57. btc_dptx_mst_remote_i2c_rd_req.....	124
9.58. btc_dptx_mst_set_color_space.....	124
9.59. btc_dptx_mst_tavgts_set.....	125
9.60. btc_dptx_mst_up_req_irq.....	125
9.61. btc_dptx_mst_vcpid_set.....	126
9.62. btc_dptx_mst_vcptab_addvc.....	126
9.63. btc_dptx_mst_vcptab_clear.....	126
9.64. btc_dptx_mst_vcptab_delvc.....	127
9.65. btc_dptx_mst_vcptab_update.....	127
9.66. btc_dptxll_syslib API Reference.....	128
9.67. btc_dptxll_hpd_change.....	128
9.68. btc_dptxll_hpd_irq.....	128
9.69. btc_dptxll_mst_cmp_ports.....	129
9.70. btc_dptxll_mst_edid_read_rep.....	129
9.71. btc_dptxll_mst_edid_read_req.....	130
9.72. btc_dptxll_mst_get_device_ports.....	130
9.73. btc_dptxll_mst_set_csn_callback.....	130
9.74. btc_dptxll_mst_topology_discover.....	131
9.75. btc_dptxll_stream_allocate_rep.....	131
9.76. btc_dptxll_stream_allocate_req.....	132



9.77. btc_dptxll_stream_calc_VCP_size.....	132
9.78. btc_dptxll_stream_delete_rep.....	133
9.79. btc_dptxll_stream_delete_req.....	133
9.80. btc_dptxll_stream_get.....	134
9.81. btc_dptxll_stream_set_color_space.....	134
9.82. btc_dptxll_stream_set_pixel_rate.....	135
9.83. btc_dptxll_sw_ver.....	135
9.84. btc_dptxll_syslib_add_tx.....	136
9.85. btc_dptxll_syslib_init.....	136
9.86. btc_dptxll_syslib_monitor.....	137
9.87. btc_dpxx_syslib Additional Types.....	137
9.88. btc_dprx_syslib Supported DPCD Locations.....	137
<b>10. DisplayPort Source Register Map and DPCD Locations.....</b>	<b>138</b>
10.1. Source General Registers.....	138
10.1.1. DPTX_TX_CONTROL.....	138
10.1.2. DPTX_TX_STATUS.....	139
10.1.3. DPTX_TX_VERSION.....	140
10.2. Source MSA Registers.....	140
10.2.1. DPTX0_MSA_MVID.....	140
10.2.2. DPTX0_MSA_NVID.....	141
10.2.3. DPTX0_MSA_HTOTAL.....	141
10.2.4. DPTX0_MSA_VTOTAL.....	141
10.2.5. DPTX0_MSA_HSP.....	142
10.2.6. DPTX0_MSA_HSW.....	142
10.2.7. DPTX0_MSA_HSTART.....	142
10.2.8. DPTX0_MSA_VSTART.....	143
10.2.9. DPTX0_MSA_VSP.....	143
10.2.10. DPTX0_MSA_VSW.....	143
10.2.11. DPTX0_MSA_HWIDTH.....	144
10.2.12. DPTX0_MSA_VHEIGHT.....	144
10.2.13. DPTX0_MSA_MISC0.....	144
10.2.14. DPTX0_MSA_MISC1.....	145
10.2.15. DPTX0_MSA_COLOR.....	145
10.2.16. DPTX0_VBID.....	146
10.3. Source Link PHY Control and Status.....	146
10.3.1. DPTX_PRE_VOLT0.....	146
10.3.2. DPTX_PRE_VOLT1.....	147
10.3.3. DPTX_PRE_VOLT2.....	147
10.3.4. DPTX_PRE_VOLT3.....	147
10.3.5. DPTX_RECONFIG.....	148
10.3.6. DPTX_TEST_80BIT_PATTERN1.....	148
10.3.7. DPTX_TEST_80BIT_PATTERN2.....	148
10.3.8. DPTX_TEST_80BIT_PATTERN3.....	149
10.4. Source Timestamp.....	149
10.5. Source CRC Registers.....	149
10.6. Source Audio Registers.....	150
10.7. Source MST Registers.....	151
10.7.1. DPTX_MST_VCPTAB0.....	152
10.7.2. DPTX_MST_VCPTAB1.....	152
10.7.3. DPTX_MST_VCPTAB2.....	153



- 10.7.4. DPTX\_MST\_VCPTAB3..... 153
- 10.7.5. DPTX\_MST\_VCPTAB4..... 153
- 10.7.6. DPTX\_MST\_VCPTAB5..... 154
- 10.7.7. DPTX\_MST\_VCPTAB6..... 154
- 10.7.8. DPTX\_MST\_VCPTAB7..... 155
- 10.7.9. DPTX\_MST\_TAVG\_TS..... 155
- 10.8. Source AUX Controller Interface..... 156
  - 10.8.1. DPTX\_AUX\_CONTROL..... 156
  - 10.8.2. DPTX\_AUX\_COMMAND..... 157
  - 10.8.3. DPTX\_AUX\_BYTE0..... 157
  - 10.8.4. DPTX\_AUX\_BYTE1..... 158
  - 10.8.5. DPTX\_AUX\_BYTE2..... 158
  - 10.8.6. DPTX\_AUX\_BYTE3..... 158
  - 10.8.7. DPTX\_AUX\_BYTE4..... 159
  - 10.8.8. DPTX\_AUX\_BYTE5..... 159
  - 10.8.9. DPTX\_AUX\_BYTE6..... 159
  - 10.8.10. DPTX\_AUX\_BYTE7..... 160
  - 10.8.11. DPTX\_AUX\_BYTE8..... 160
  - 10.8.12. DPTX\_AUX\_BYTE9..... 160
  - 10.8.13. DPTX\_AUX\_BYTE10..... 161
  - 10.8.14. DPTX\_AUX\_BYTE11..... 161
  - 10.8.15. DPTX\_AUX\_BYTE12..... 161
  - 10.8.16. DPTX\_AUX\_BYTE13..... 162
  - 10.8.17. DPTX\_AUX\_BYTE14..... 162
  - 10.8.18. DPTX\_AUX\_BYTE15..... 162
  - 10.8.19. DPTX\_AUX\_BYTE16..... 163
  - 10.8.20. DPTX\_AUX\_BYTE17..... 163
  - 10.8.21. DPTX\_AUX\_BYTE18..... 163
  - 10.8.22. DPTX\_AUX\_RESET..... 163
- 10.9. Source-Supported DPCD Locations..... 164
- 11. DisplayPort Sink Register Map and DPCD Locations..... 166**
  - 11.1. Sink General Registers..... 166
    - 11.1.1. DPRX\_RX\_CONTROL..... 166
    - 11.1.2. DPRX\_RX\_STATUS..... 167
    - 11.1.3. DPRX\_BER\_CONTROL..... 169
    - 11.1.4. DPRX\_BER\_CNT0..... 170
    - 11.1.5. DPRX\_BER\_CNT1..... 171
  - 11.2. Sink Timestamp..... 171
  - 11.3. Sink Bit-Error Counters..... 171
    - 11.3.1. DPRX\_BER\_CNTI0..... 172
    - 11.3.2. DPRX\_BER\_CNTI1..... 172
  - 11.4. Sink MSA Registers..... 172
    - 11.4.1. DPRX0\_MSA\_MVID..... 173
    - 11.4.2. DPRX0\_MSA\_NVID..... 173
    - 11.4.3. DPRX0\_MSA\_HTOTAL..... 173
    - 11.4.4. DPRX0\_MSA\_VTOTAL..... 173
    - 11.4.5. DPRX0\_MSA\_HSP..... 174
    - 11.4.6. DPRX0\_MSA\_HSW..... 174
    - 11.4.7. DPRX0\_MSA\_HSTART..... 174
    - 11.4.8. DPRX0\_MSA\_VSTART..... 175



11.4.9. DPRX0_MSA_VSP.....	175
11.4.10. DPRX0_MSA_VSW.....	175
11.4.11. DPRX0_MSA_HWIDTH.....	175
11.4.12. DPRX0_MSA_VHEIGHT.....	176
11.4.13. DPRX0_MSA_MISC0.....	176
11.4.14. DPRX0_MSA_MISC1.....	176
11.4.15. DPRX0_MSA_COLOR.....	177
11.4.16. DPRX0_VBID.....	177
11.5. Sink Audio Registers.....	178
11.5.1. DPRX0_AUD_MAUD.....	178
11.5.2. DPRX0_AUD_NAUD.....	178
11.5.3. DPRX0_AUD_AIF0.....	178
11.5.4. DPRX0_AUD_AIF1.....	179
11.5.5. DPRX0_AUD_AIF2.....	179
11.5.6. DPRX0_AUD_AIF3.....	179
11.5.7. DPRX0_AUD_AIF4.....	180
11.6. Sink MST Registers.....	180
11.6.1. DPRX_MST_VCPTAB0.....	181
11.6.2. DPRX_MST_VCPTAB1.....	182
11.6.3. DPRX_MST_VCPTAB2.....	182
11.6.4. DPRX_MST_VCPTAB3.....	183
11.6.5. DPRX_MST_VCPTAB4.....	183
11.6.6. DPRX_MST_VCPTAB5.....	184
11.6.7. DPRX_MST_VCPTAB6.....	184
11.6.8. DPRX_MST_VCPTAB7.....	185
11.7. Sink AUX Controller Interface.....	185
11.7.1. DPRX_AUX_CONTROL.....	185
11.7.2. DPRX_AUX_STATUS.....	186
11.7.3. DPRX_AUX_COMMAND.....	187
11.7.4. DPRX_AUX_BYTE0.....	187
11.7.5. DPRX_AUX_BYTE1.....	187
11.7.6. DPRX_AUX_BYTE2.....	188
11.7.7. DPRX_AUX_BYTE3.....	188
11.7.8. DPRX_AUX_BYTE4.....	188
11.7.9. DPRX_AUX_BYTE5.....	189
11.7.10. DPRX_AUX_BYTE6.....	189
11.7.11. DPRX_AUX_BYTE7.....	189
11.7.12. DPRX_AUX_BYTE8.....	190
11.7.13. DPRX_AUX_BYTE9.....	190
11.7.14. DPRX_AUX_BYTE10.....	190
11.7.15. DPRX_AUX_BYTE11.....	191
11.7.16. DPRX_AUX_BYTE12.....	191
11.7.17. DPRX_AUX_BYTE13.....	191
11.7.18. DPRX_AUX_BYTE14.....	192
11.7.19. DPRX_AUX_BYTE15.....	192
11.7.20. DPRX_AUX_BYTE16.....	192
11.7.21. DPRX_AUX_BYTE17.....	193
11.7.22. DPRX_AUX_BYTE18.....	193
11.7.23. DPRX_AUX_I2C0.....	193
11.7.24. DPRX_AUX_I2C1.....	194
11.7.25. DPRX_AUX_RESET.....	194



11.7.26. DPRX_AUX_HPDP.....	194
11.8. Sink CRC Registers.....	195
11.9. Sink-Supported DPCD Locations.....	196
<b>12. DisplayPort Intel FPGA IP User Guide Archives.....</b>	<b>200</b>
<b>13. Document Revision History for the DisplayPort Intel FPGA IP User Guide.....</b>	<b>201</b>



## 1. DisplayPort Intel® FPGA IP Quick Reference

The DisplayPort Intel® FPGA IP provides support for next-generation video display interface technology.

The DisplayPort Intel FPGA IP core is part of the Intel FPGA IP Library, which is distributed with the Intel Quartus® Prime software..

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

Information		Description
<b>Release Information</b>	Version	19.1
	Release Date	April 2019
	Ordering Code	IP-DP
<b>IP Core Information</b>	Core Features	<ul style="list-style-type: none"> <li>• Conforms to the <i>Video Electronics Standards Association (VESA) DisplayPort Standard version 1.4</i></li> <li>• Scalable main data link <ul style="list-style-type: none"> <li>– 1, 2, or 4 lane operation</li> <li>– 1.62, 2.7, 5.4, and 8.1 gigabits per second (Gbps) per lane with an embedded clock <sup>(1)</sup></li> </ul> </li> <li>• Color support <ul style="list-style-type: none"> <li>– RGB 18, 24, 30, 36, or 48 bpp</li> <li>– YCbCr 4:4:4 24, 30, 36, or 48 bpp</li> <li>– YCbCr 4:2:2 16, 20, 24, or 32 bpp</li> <li>– YCbCr 4:2:0 12, 15, 18, or 24 bpp</li> </ul> </li> <li>• 40-bit (quad symbol) and 20-bit (dual symbol) transceiver data interface</li> <li>• Support for 1, 2, or 4 parallel pixels per clock</li> <li>• Support for 2 or 8 audio channels</li> <li>• Multi-stream transport (MST) support <ul style="list-style-type: none"> <li>– Intel Arria® 10 devices support up to 4 streams</li> <li>– Arria V devices support up to 2 streams</li> <li>– Stratix V devices support up to 4 streams</li> </ul> </li> <li>• Support for progressive and interlaced video</li> <li>• Source support for proprietary video image format (optional)</li> <li>• Support for adaptive sync feature</li> </ul>

*continued...*

<sup>(1)</sup> 8.1 Gbps is available only in the Intel Quartus Prime Pro Edition software.



Information		Description
		<ul style="list-style-type: none"> <li>Support for High Dynamic Range (HDR) metadata transport using secondary stream data packet</li> <li>Auxiliary channel for 2-way communication (link and device management)</li> <li>Hot plug detect (HPD)               <ul style="list-style-type: none"> <li>Sink announces its presence</li> <li>Sink requests the source's attention</li> </ul> </li> </ul>
	Typical Application	<ul style="list-style-type: none"> <li>Interfaces within a PC or monitor</li> <li>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</li> </ul>
	Device Family Support	Intel Stratix® 10 (H-tile and L-tile, Intel Arria 10, Intel Cyclone® 10 GX, Arria V, Cyclone V, and Stratix V FPGA devices.
	Design Tools	<ul style="list-style-type: none"> <li>IP Catalog in the Intel Quartus Prime software for IP design instantiation and compilation</li> <li>Timing Analyzer in the Intel Quartus Prime software for timing analysis</li> <li>ModelSim* - Intel FPGA Edition, NCSim, Riviera-PRO*, VCS*/VCS MX, and Xcelium* Parallel software for design simulation</li> </ul>

**Note:** The DisplayPort Intel FPGA IP core provides support for Global Time Code (GTC). For further inquiries, contact your nearest Intel sales representative or file an Intel Premier Support (IPS) case at <https://www.intel.com/content/www/us/en/programmable/my-intel/mal-home.html>.

**Related Information**

- [DisplayPort Intel Arria 10 FPGA IP Design Example User Guide](#)  
For more information about the Intel Arria 10 design example.
- [DisplayPort Intel Cyclone 10 GX FPGA IP Design Example User Guide](#)  
For more information about the Intel Cyclone 10 GX design example.
- [DisplayPort Intel Stratix 10 FPGA IP Design Example User Guide](#)  
For more information about the Intel Stratix 10 design examples.
- [DisplayPort Intel FPGA IP User Guide Archives](#) on page 200  
Provides a list of user guides for previous versions of the Intel FPGA DisplayPort IP core.

**1.1. DisplayPort Terms and Acronyms**

The tables list the commonly used DisplayPort terms and acronyms.

**Table 1. DisplayPort Acronyms**

Acronym	Description
API	Application Programming Interface
AUX	Auxiliary
bpc	Bits per Component
bpp	Bits per Pixel
<i>continued...</i>	



Acronym	Description
BE	Blanking End
BS	Blanking Start
DP	DisplayPort
DPCD	DisplayPort Configuration Data
eDP	Embedded DisplayPort
EDID	Enhanced Display Identification Data
GPU	Graphics Processor Unit
HBR	High Bit Rate (2.7 Gbps per lane)
HBR2	High Bit Rate 2 (5.4 Gbps per lane)
HBR3	High Bit Rate 3 (8.1 Gbps per lane)
HPD	Hot Plug Detect
MST	Multi-Stream Transport
Maud	M value for audio
Mvid	M value for video
Naud	N value for audio
Nvid	N value for video
RBR	Reduced Bit Rate (1.62 Gbps per lane)
RGB	Red Green Blue
RX	Receiver
SDP	Secondary-Data Packet
SE	SDP End
SR	Scrambler Reset
SS	SDP Start
SST	Single-Stream Transport
TX	Transmitter

**Table 2. DisplayPort Terms**

Term	Definition
Link Symbol Clock (LSym_Clk)	Link Symbol clock frequency ( $f_{LSym\_Clk}$ ) across link rate: - <ul style="list-style-type: none"> <li>HBR3 (8.1Gbps) = 810 MHz</li> <li>HBR2 (5.4Gbps) = 540 MHz</li> <li>HBR (2.7Gbps) = 270 MHz</li> <li>RBR (1.62Gbps) = 162 MHz</li> </ul> <i>Note: LSym_Clk is equivalent to LS_Clk in VESA DisplayPort Standard version 1.4.</i>
Link Speed Clock (Ls_clk)	Transceiver recovered clock out.

**continued...**



Term	Definition
	Link Speed clock frequency equals: $f\_LSym\_clk / SYMBOLES\_PER\_CLOCK$ .
Stream Clock or Pixel Clock (Strm_Clk)	Used for transferring stream data into a DisplayPort transmitter within a DisplayPort Source device or from a DisplayPort receiver within a DP Sink device. Video and audio (optional) are likely to have separate stream clocks. Stream clock frequency ( $f\_Strm\_clk$ ) represent the pixel rate. For example, $f\_Strm\_clk$ for 1080p60 (CEA-861-F VIC16) is 148.5 Mhz.
Video Clock (vid_clk)	Video clock frequency equals: $f\_Strm\_clk / PIXELS\_PER\_CLOCK$

## 2. About This IP Core

This document describes the DisplayPort Intel FPGA IP core, 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:

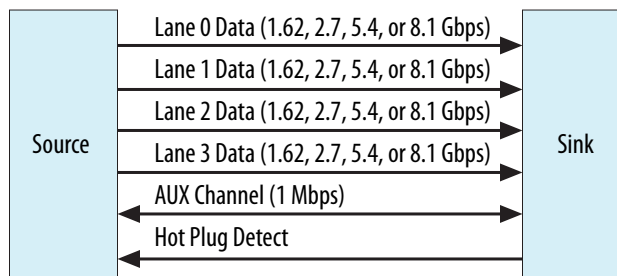
- Interfaces within a PC or monitor
- External display connections, including interfaces between a PC and monitor or projector, between a PC and TV, or between a device such as a DVD player and TV display

The DisplayPort Intel FPGA IP core supports scalable Main Link with 1, 2, or 4 lanes, with 4 selectable data rates on each lane: 1.62 Gbps, 2.7 Gbps, 5.4 Gbps, and 8.1 Gbps.

Main Link transports video and audio streams with embedded clocking to decoupled pixel and audio clocks from the transmission clock. The IP core transmits Main Link's data in scrambled ANSI 8B/10B format and includes redundancy in the data transmission for error detection. For secondary data, such as audio, the IP core uses Solomon Reed coding for error detection.

The DisplayPort's AUX channel consists of an AC-coupled terminated differential pair. AUX channel uses Manchester II coding for its channel coding and provides a data rate of 1 Mbps. Each transaction takes less than 500  $\mu$ s with a maximum burst data size of 16 bytes.

**Figure 1. DisplayPort Source and Sink Communication**





## 2.1. Device Family Support

**Table 3. Intel Device Family Support**

Device Family	Support Level
Intel Stratix 10 (H-tile and L-tile)	Final
Intel Arria 10	Final
Intel Cyclone 10 GX	Final
Arria V GX/GT/GS	Final
Arria V GZ	Final
Cyclone V	Final
Stratix V	Final

The following terms define device support levels for Intel FPGA IP cores:

- **Advance support**—the IP core is available for simulation and compilation for this device family. Timing models include initial engineering estimates of delays based on early post-layout information. The timing models are subject to change as silicon testing improves the correlation between the actual silicon and the timing models. You can use this IP core for system architecture and resource utilization studies, simulation, pinout, system latency assessments, basic timing assessments (pipeline budgeting), and I/O transfer strategy (data-path width, burst depth, I/O standards tradeoffs).
- **Preliminary support**—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.

The following table lists the link rate support offered by the DisplayPort Intel FPGA IP core for each Intel FPGA family.

**Table 4. Link Rate Support by Device Family**

Device Family	Dual Symbol (20-Bit Mode)	Quad Symbol (40-Bit Mode)	FPGA Fabric Speed Grade
Intel Stratix 10 (H-tile and L-tile)	RBR, HBR, HBR2	RBR, HBR, HBR2, HBR3	1, 2 <i>Note:</i> HBR3 support is preliminary.
Intel Arria 10	RBR, HBR, HBR2	RBR, HBR, HBR2, HBR3	1, 2
Intel Cyclone 10 GX	RBR, HBR, HBR2	RBR, HBR, HBR2, HBR3	5, 6
Stratix V	RBR, HBR, HBR2	RBR, HBR, HBR2	1, 2, 3
Arria V GX/GT/GS	RBR, HBR	RBR, HBR, HBR2	3, 4, 5
Arria V GZ	RBR, HBR, HBR2	RBR, HBR, HBR2	Any supported speed grade
Cyclone V	RBR, HBR	RBR, HBR	Any supported speed grade



**Table 5. Adaptive Sync Support by Device Family**

The Adaptive Sync feature is available only in the Intel Quartus Prime Pro Edition software.

Device Family	Adaptive Sync Support
Intel Stratix 10 (H-tile and L-tile)	Yes
Intel Arria 10	Yes
Intel Cyclone 10 GX	Yes

To enable the Adaptive Sync feature, refer to [Table 22](#) on page 53 and [Video Interface \(TX Video IM Enable = 1\)](#) on page 55. For detailed implementation of the feature, refer to the *DisplayPort SST Parallel Loopback with Adaptive Sync Support* section in the respective DisplayPort Intel FPGA IP design example user guides.

## 2.2. IP Core Verification

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

## 2.3. 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 Intel Quartus Prime software version 19.1 for the following devices:

- Arria V (5AGXFB3H4F40C5)
- Cyclone V (5CGTFD9E5F35C7)
- Stratix V (5SGXEA7K2F40C2)
- Intel Arria 10 (10AX115S2F45I1SG)
- Intel Cyclone 10 GX (10CX220YF780E5G)
- Intel Stratix 10 (1SG280HU1F50E2VGS1)



**Table 6. DisplayPort Intel FPGA IP Core FPGA Resource Utilization**

The table below shows the resource information for Arria V and Cyclone V devices using M10K; Intel Arria 10, Intel Stratix 10, and Stratix V devices using M20K. The resources were obtained using the following parameter settings:

- Mode = simplex
- Maximum lane count = 4 lanes
- Maximum video input color depth = 8 bits per color (bpc)
- Pixel input mode = 1 pixel per clock

Device	Streams	Direction	Symbol per Clock	ALMs	Logic Registers		Memory	
					Primary	Secondary	Bits	M10K or M20K
Intel Stratix 10	SST (Single Stream)	RX	Dual	4967	6748	884	16256	11
			Quad	6976	8344	1112	18816	14
		TX	Dual	4800	6353	533	12176	15
			Quad	7716	8853	641	22688	29
Intel Arria 10	SST (Single Stream)	RX	Dual	4,322	6,851	1,283	28,288	13
			Quad	9,297	10,955	1319	34,496	36
		TX	Dual	4,978	6,330	955	12,664	15
			Quad	8,264	8,545	1,156	17,096	13
	MST (4 Streams)	RX	Quad	36,403	38,337	2,700	105,728	88
		TX	Quad	41,999	55,483	6,000	99,808	86
Intel Cyclone 10 GX	SST (Single Stream)	RX	Dual	4,322	6,851	1,283	28,288	13
			Quad	9,297	10,955	1319	34,496	36
		TX	Dual	4,978	6,330	955	12,664	15
			Quad	8,264	8,545	1,156	17,096	13
Arria V GX	SST	RX	Dual	7,677	9,786	661	19,648	36
			Quad	9,247	11,114	900	34,496	36
		TX	Dual	8,263	10,304	320	22,816	20
			Quad	12,660	13,040	1243	33,632	31
	MST (2 Streams)	RX	Quad	17,996	19,619	1,884	51,328	54
		TX	Quad	22,601	26,302	2,488	57,792	62
Cyclone V GX	SST	RX	Dual	6,236	7,619	2,864	19,648	36
			Quad	7,769	8,925	3,190	34,496	36
		TX	Dual	8,222	10,267	494	22,816	20
			Quad	12,628	13,003	1,359	33,632	31
Stratix V GX/Arria V GZ	SST	RX	Dual	7,743	9,972	563	19,648	36
			Quad	9,344	11,420	732	34,496	36
		TX	Dual	6,725	10,067	645	22,816	20

*continued...*





Device	Streams	Direction	Symbol per Clock	ALMs	Logic Registers		Memory	
					Primary	Secondary	Bits	M10K or M20K
			Quad	12,168	13,060	1,223	33,632	31
	MST (4 Streams)	RX	Quad	31,079	27,789	3,108	56,320	48
		TX	Quad	33,218	30,363	2,613	45,696	68

### Related Information

[Fitter Resources Reports](#)

More information about Intel Quartus Prime resource utilization reporting.

## 3. Getting Started

---

This chapter provides a general overview of the Intel FPGA IP core design flow to help you quickly get started with the DisplayPort Intel FPGA IP core. The IP core is installed as part of the Intel Quartus Prime installation process. You can select and parameterize any Intel FPGA IP core from the library. Intel provides an integrated parameter editor that allows you to customize the DisplayPort IP core to support a wide variety of applications. The parameter editor guides you through the setting of parameter values and selection of optional ports.

### Related Information

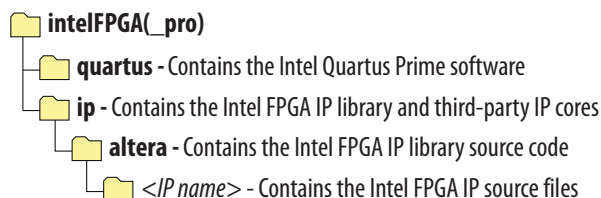
- [Introduction to Intel FPGA IP Cores](#)  
Provides general information about all Intel FPGA IP cores, including parameterizing, generating, upgrading, and simulating IP cores.
- [Creating Version-Independent IP and Platform Designer Simulation Scripts](#)  
Create simulation scripts that do not require manual updates for software or IP version upgrades.
- [Project Management Best Practices](#)  
Guidelines for efficient management and portability of your project and IP files.

### 3.1. Installing and Licensing Intel FPGA IP Cores

The Intel Quartus Prime software installation includes the Intel FPGA IP library. This library provides many useful IP cores for your production use without the need for an additional license. Some Intel FPGA IP cores require purchase of a separate license for production use. The Intel FPGA IP Evaluation Mode allows you to evaluate these licensed Intel FPGA IP cores in simulation and hardware, before deciding to purchase a full production IP core license. You only need to purchase a full production license for licensed Intel IP cores after you complete hardware testing and are ready to use the IP in production.

The Intel Quartus Prime software installs IP cores in the following locations by default:

**Figure 2. IP Core Installation Path**





**Table 7. IP Core Installation Locations**

Location	Software	Platform
<drive>:\intelFPGA_pro\quartus\ip\altera	Intel Quartus Prime Pro Edition	Windows*
<drive>:\intelFPGA\quartus\ip\altera	Intel Quartus Prime Standard Edition	Windows
<home directory>:/intelFPGA_pro/quartus/ip/altera	Intel Quartus Prime Pro Edition	Linux*
<home directory>:/intelFPGA/quartus/ip/altera	Intel Quartus Prime Standard Edition	Linux

**Note:** The Intel Quartus Prime software does not support spaces in the installation path.

### 3.1.1. Intel FPGA IP Evaluation Mode

The free Intel FPGA IP Evaluation Mode allows you to evaluate licensed Intel FPGA IP cores in simulation and hardware before purchase. Intel FPGA IP Evaluation Mode supports the following evaluations without additional license:

- Simulate the behavior of a licensed Intel FPGA 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.

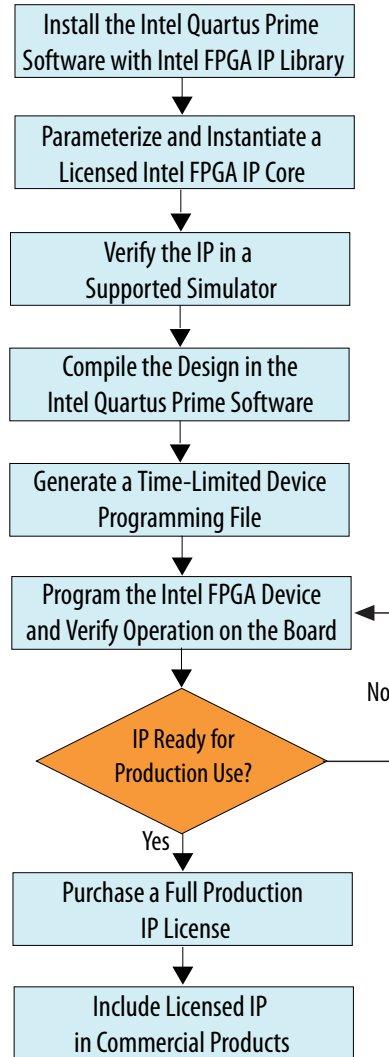
Intel FPGA IP Evaluation Mode supports the following operation modes:

- **Tethered**—Allows running the design containing the licensed Intel FPGA IP indefinitely with a connection between your board and the host computer. Tethered mode requires a serial joint test action group (JTAG) cable connected between the JTAG port on your board and the host computer, which is running the Intel Quartus Prime Programmer for the duration of the hardware evaluation period. The Programmer only requires a minimum installation of the Intel Quartus Prime software, and requires no Intel Quartus Prime license. The host computer controls the evaluation time by sending a periodic signal to the device via the JTAG port. If all licensed IP cores in the design support tethered mode, the evaluation time runs until any IP core evaluation expires. If all of the IP cores support unlimited evaluation time, the device does not time-out.
- **Untethered**—Allows running the design containing the licensed IP for a limited time. The IP core reverts to untethered mode if the device disconnects from the host computer running the Intel Quartus Prime software. The IP core also reverts to untethered mode if any other licensed IP core in the design does not support tethered mode.

When the evaluation time expires for any licensed Intel FPGA IP in the design, the design stops functioning. All IP cores that use the Intel FPGA IP Evaluation Mode time out simultaneously when any IP core in the design times out. When the evaluation time expires, you must reprogram the FPGA device before continuing hardware verification. To extend use of the IP core for production, purchase a full production license for the IP core.

You must purchase the license and generate a full production license key before you can generate an unrestricted device programming file. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (<project name>\_time\_limited.sof) that expires at the time limit.

Figure 3. Intel FPGA IP Evaluation Mode Flow



**Note:** Refer to each IP core's user guide for parameterization steps and implementation details.

Intel licenses IP cores on a per-seat, perpetual basis. The license fee includes first-year maintenance and support. You must renew the maintenance contract to receive updates, bug fixes, and technical support beyond the first year. You must purchase a full production license for Intel FPGA IP cores that require a production license, before generating programming files that you may use for an unlimited time. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (*<project name>\_time\_limited.sof*) that expires at the time limit. To obtain your production license keys, visit the [Self-Service Licensing Center](#).

The [Intel FPGA Software License Agreements](#) govern the installation and use of licensed IP cores, the Intel Quartus Prime design software, and all unlicensed IP cores.



#### Related Information

- [Intel Quartus Prime Licensing Site](#)
- [Introduction to Intel FPGA Software Installation and Licensing](#)

## 3.2. Specifying IP Core Parameters and Options

Follow these steps to specify the DisplayPort IP core parameters and options.

1. Create a Intel Quartus Prime project using the **New Project Wizard** available from the File menu.
2. On the **Tools** menu, click **IP Catalog**.
3. Under **Installed IP**, double-click **Library > Interface Protocols > Audio&Video > DisplayPort Intel FPGA IP**.  
The parameter editor appears.
4. In the parameter editor, 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 targeted Intel FPGA family and output file HDL preference. Click **OK**.
5. Specify parameters and options in the DisplayPort parameter editor:
  - 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 processing the IP core files in other EDA tools.
6. Click **Generate** to generate the IP core and supporting files, including simulation models.
7. Click **Close** when file generation completes.
8. Click **Finish**.
9. If you generate the DisplayPort Intel FPGA IP core instance in a Intel Quartus Prime project, you are prompted to add Intel Quartus Prime IP File (.qip) and Intel Quartus Prime Simulation IP File (.sip) to the current Intel Quartus Prime project.

## 3.3. Simulating the Design

You can simulate your DisplayPort Intel FPGA IP core variation using the simulation model that the Intel Quartus Prime software generates. The simulation model files are generated in vendor-specific subdirectories of your project directory. The DisplayPort Intel FPGA IP core includes a simulation example.

The following sections teach you how to simulate the generated DisplayPort Intel FPGA IP core variation with the generated simulation model.

#### Related Information

[DisplayPort Intel FPGA IP Core Simulation Example](#) on page 91

### 3.3.1. 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
```

### 3.4. Compiling the Full Design and Programming the FPGA

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

#### Related Information

- [Intel Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design](#)  
Provides more information about compiling the design.
- [Programming Intel FPGA Devices](#)  
Provides more information about programming the device.



## 4. DisplayPort Intel FPGA IP Core Hardware Design Examples

---

Intel offers design examples that you can simulate, compile, and test in hardware.

The implementation of the DisplayPort Intel FPGA IP on hardware requires additional components specific to the targeted device.

### 4.1. DisplayPort Intel FPGA IP Hardware Design Examples for Intel Arria 10, Intel Cyclone 10 GX, and Intel Stratix 10 Devices

The DisplayPort Intel FPGA IP core offers design examples that you can generate through the IP catalog in the Intel Quartus Prime Pro Edition software.

For detailed information about the DisplayPort Intel FPGA IP design examples, refer to the following user guides:

#### Related Information

- [DisplayPort Intel Arria 10 FPGA IP Design Example User Guide](#)  
For more information about the Intel Arria 10 design example.
- [DisplayPort Intel Cyclone 10 GX FPGA IP Design Example User Guide](#)  
For more information about the Intel Cyclone 10 GX design example.
- [DisplayPort Intel Stratix 10 FPGA IP Design Example User Guide](#)  
For more information about the Intel Stratix 10 design examples.

### 4.2. DisplayPort Intel FPGA IP Hardware Design Examples for Arria V, Cyclone V, and Stratix V Devices

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

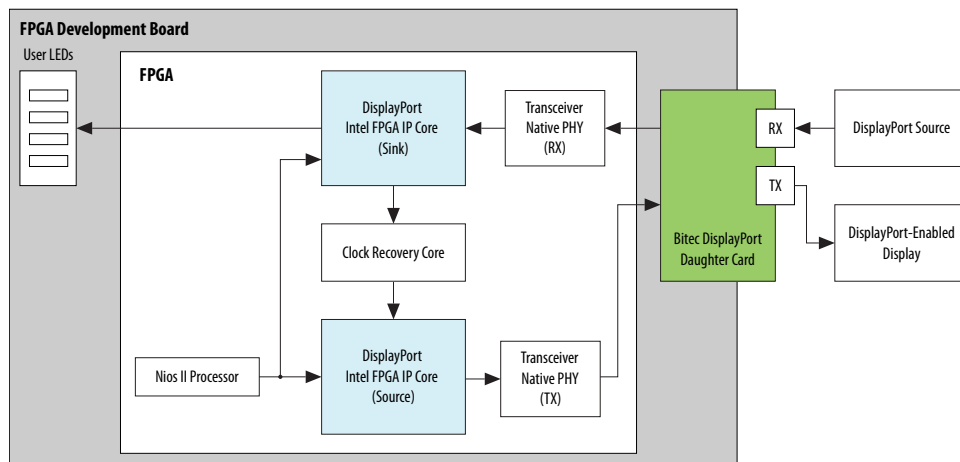
This design performs a loop-through for a standard DisplayPort video stream. You connect a DisplayPort-enabled device—such as a graphics card with DisplayPort interface—to the Transceiver Native PHY RX, and the DisplayPort sink input. The DisplayPort sink decodes the port into a standard video stream and sends it to the clock recovery core. The clock recovery core synthesizes the original video pixel clock to be transmitted together with the received video data. You require the clock recovery feature to produce video without using a frame buffer. The clock recovery core then sends the video data to the DisplayPort source, and the Transceiver Native PHY TX. The DisplayPort source port of the daughter card transmits the image to a monitor.

The design uses the development board from the following kits:

- Arria V GX FPGA Starter Kit
- Cyclone V GT FPGA Development Kit
- Stratix V GX FPGA Development Kit

**Note:** If you use another Intel FPGA development board, you must change the device assignments and the pin assignments. You make these changes in the `assignments.tcl` file. If you use another DisplayPort daughter card, you must change the pin assignments, Platform Designer system, and software.

**Figure 4. Hardware Design 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 5. Hardware Design Block Diagram

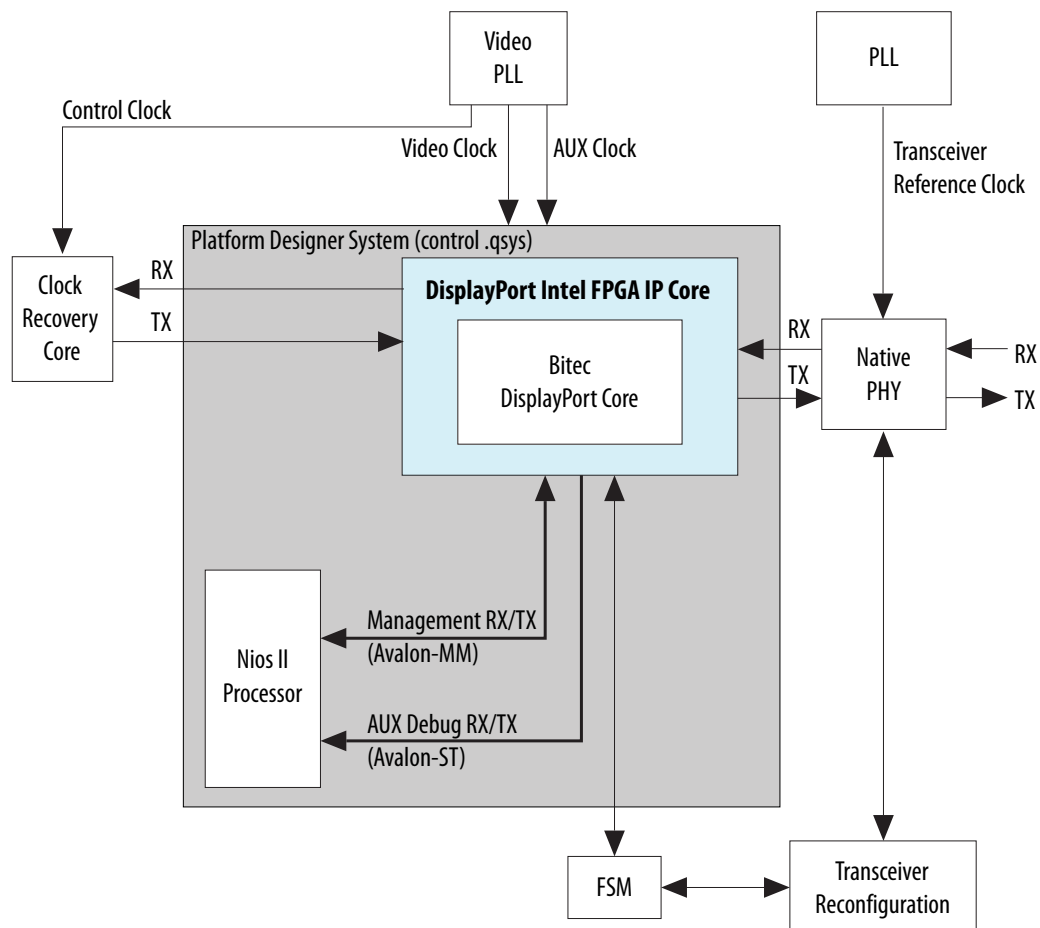


Table 8. Clock Source for the Hardware Design

Clock	Frequency	Description
AUX Clock	16 MHz	Used as primary clock source for Auxiliary encoder and decoder. Refer to Source <a href="#">AUX Interface</a> on page 53 and Sink <a href="#">AUX Interface</a> on page 74 for more information.
Control Clock	60 MHz	Used for Pixel Clock Recovery (PCR) module loop controller and fPLL reconfiguration blocks.
Native PHY Reference Clock	135 MHz	Used as Native PHY reference clock for Transceiver CMU PLL.
Video Clock	160 MHz or 300 MHz	Video Clock has two functions in this demonstration. <ul style="list-style-type: none"> <li>• <code>rxN_vid_clock</code> for transferring video data from the sink decoder.</li> <li>• Input to PCR module as <code>vid_data</code> clock source.</li> </ul>



**Note:** When rxN\_vid\_clock is used for transferring the sink device's video data and control, the clock frequency must be equal or faster than the upstream device Stream Clock (Strm\_Clk) / PIXELS\_PER\_CLOCK. For example:

- If the upstream device transmits video data at 1080@60 (Strm\_Clk = 148.5 MHz) and the sink device is configured at PIXELS\_PER\_CLOCK = 1, the device must drive rxN\_vid\_clk at a minimal frequency of 148.5 MHz.
- If the sink device is configured at PIXELS\_PER\_CLOCK = 4, the device must drive rxN\_vid\_clk at a minimal frequency of 37.125 MHz (148.5 MHz/4).

The DisplayPort hardware demonstration uses the IOPLL to drive rxN\_vid\_clock with a fixed clock frequency.

- For designs with HBR2 at PIXELS\_PER\_CLOCK = 4, the recommended rxN\_vid\_clock frequency is 160 MHz to support 4K@60 resolution
- For designs with HBR2 at PIXELS\_PER\_CLOCK = 2, the recommended rxN\_vid\_clock frequency is 300 MHz to support 4K@60 resolution

**Table 9. LED Function**

The development board user LEDs illuminate to indicate the functions described in the table below.

Supported Intel FPGAs	Function
USER_LED[0]	This LED indicates that source is successfully lane-trained and is sending video. rxN_vid_locked drives this LED. This LED turns off if the source is not driving good video.
USER_LED[1]	This LED illuminates for 1-lane designs.
USER_LED[2]	This LED illuminates for 2-lane designs.
USER_LED[3]	This LED illuminates for 4-lane designs.
USER_LED[7:6]	These LEDs indicate the RX link rate. <ul style="list-style-type: none"> <li>• 00 = RBR</li> <li>• 01 = HBR</li> <li>• 10 = HBR2</li> </ul>

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

- The Bitec HSMC 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



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. Press a push-button to print the current TX and RX MSAs.

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

#### Related Information

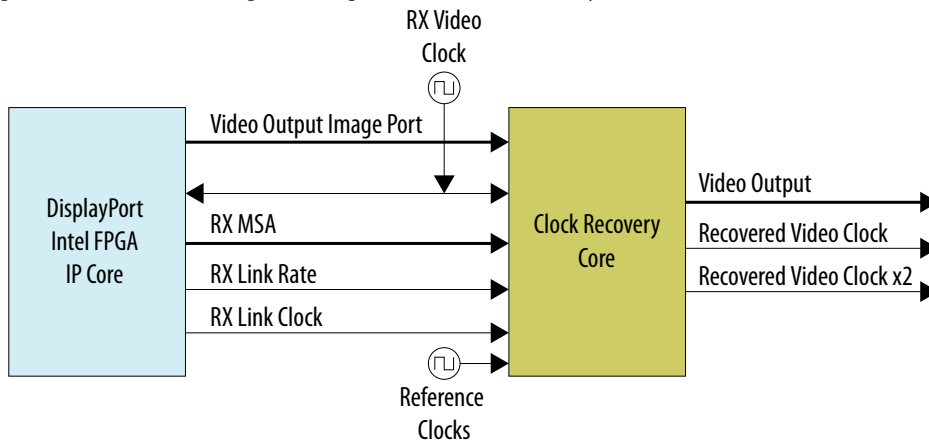
- [Stratix V GX FPGA Development Kit](#)
- [Arria V GX FPGA Starter Kit](#)
- [Cyclone V GT FPGA Development Kit](#)
- [AN 745: Design Guidelines for Intel FPGA DisplayPort Interface](#)

### 4.2.1. Clock Recovery Core

The clock recovery core is a single encrypted module called `bitec_clkrec`.

**Figure 6. Clock Recovery Core Integration Diagram**

The figure below shows the integration diagram of the clock recovery core.



To synthesize the video pixel clock from the link clock, the clock recovery core gathers information about the current MSA and the currently used link rate from the DisplayPort sink.

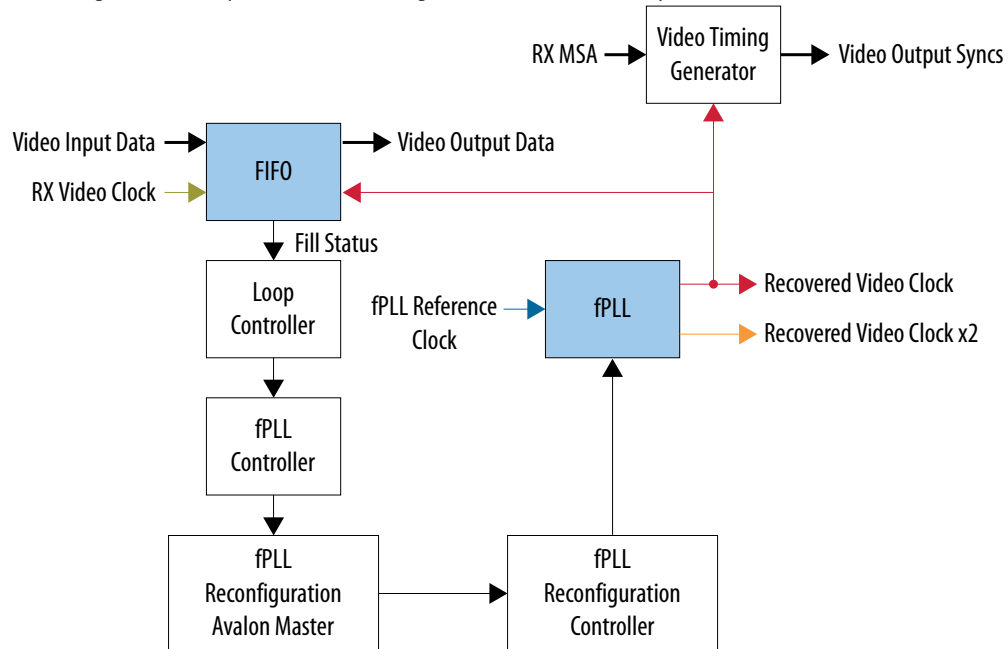
The clock recovery core produces resynchronized video data together with the following clocks:

- Recovered video pixel clock
- Second clock with twice the recovered pixel clock frequency

The video output data is synchronous to the recovered video clock. You can use the second clock as a reference clock for the TX transceiver, which is optionally used to serialize the video output data.

**Figure 7. Clock Recovery Core Functional Diagram**

The following shows a simplified functional diagram of the clock recovery core.



The clock recovery core clocks the video data input gathered from the DisplayPort sink into a dual-clock FIFO at the received video clock speed. The core reads from the video data input using the recovered video clock.

- **Video Timing Generator:** This block uses the received MSA to create `h-sync`, `v-sync`, and `data enable` signals that are synchronized to the recovered video clock.
- **Loop Controller:** This block monitors the FIFO fill level and regulates its throughput by altering the original `Mvid` value read from the MSA. The block feeds the modified `Mvid` to the `fPLL Controller`, which calculates a set of parameters suitable for the `fPLL Controller`. This set of parameters provides the value to create a recovered video clock frequency corresponding to the new `Mvid` value. The calculated `fPLL` parameters are written by the `fPLL Reconfiguration Avalon Master` to the `fPLL Reconfiguration Controller` internal registers.
- **Reconfiguration Controller:** This block serializes the parameter values and writes them to the `fPLL IP core`.
- **fPLL:** Generates the recovered video clock and a second clock with twice the frequency.

#### 4.2.1.1. Clock Recovery Core Parameters

You can use these parameters to configure the clock recovery core.

**Table 10. Clock Recovery Core Parameters**

Parameter	Default Value	Description
SYMBOLS_PER_CLOCK	4	Specifies the configuration of the DisplayPort RX transceiver used. Set to <b>2</b> for 20-bit mode (Dual symbol) or to <b>4</b> for 40-bit mode (Quad symbol).
CLK_PERIOD_NS	10	Specifies the period (in nanoseconds) of the control clock input signal connected to the port. <i>Note:</i> The recommended control clock frequency is 60 MHz. Set this parameter to 17.
DEVICE_FAMILY	Arria V	Identifies the device used. The values are <b>Arria V</b> , <b>Cyclone V</b> , and <b>Stratix V</b> .
FIXED_NVID	0	Specifies the configuration of the DisplayPort RX received video clocking used. Set to <b>1</b> for asynchronous clocking, where the Nvid value is fixed to 32'h8000. Set to <b>0</b> if the value of Nvid is a variable of 32'h8000 or any other value. <i>Note:</i> Most DisplayPort source devices transmit video using asynchronous clocking. For optimized resource usage, Intel recommends you to set the <b>FIXED_NVID</b> parameter to <b>1</b> .
PIXELS_PER_CLOCK	4	Specifies how many pixels in parallel (for each clock cycle) are gathered from the DisplayPort RX. Set to <b>1</b> for single pixel, <b>2</b> for dual, or <b>4</b> for quad pixels per clock cycle.
BPP	48	Specifies the width (in bits) of a single pixel. Set to <b>18</b> for 6-bit color, <b>24</b> for 8-bit color, and so on up to <b>48</b> for 16-bit color.

#### 4.2.1.2. Clock Recovery Interface

The following table lists the signals for the clock recovery core.

**Table 11. Clock Recovery Interface Signals**

Interface	Port Type	Clock Domain	Port	Direction	Description
control clock	Clock	N/A	clk	Input	Control logic clock. This clock runs the loop controller and fPLL reconfiguration related blocks. Intel recommends you use a 60 MHz clock.
RX link clock	Clock	N/A	rx_link_clk	Input	DisplayPort transceiver link clock. This clock is a divided version of the RX main link clock or divided by 4. <ul style="list-style-type: none"> <li>Divided by 2 when the sink core is instantiated in 20-bit mode (2 symbols per clock)</li> <li>Divided by 4 when the sink core is instantiated in 40-bit mode (4 symbols per clock)</li> </ul>
reset	Reset	clk	areset	Input	Asynchronous reset. This is an active-high signal.

*continued...*



Interface	Port Type	Clock Domain	Port	Direction	Description
RX link rate	Conduit	asynchronous	rx_link_rate[1:0]	Input	DisplayPort RX link rate. <ul style="list-style-type: none"> <li>• 00 = RBR (1.67 Gbps)</li> <li>• 01 = HBR (2.70 Gbps)</li> <li>• 10 = HBR2 (5.40 Gbps)</li> </ul> You need this information for the clock recovery clock to correctly calculate the fPLL parameters.
RX MSA	Conduit	rx_link_clk	rx_msa[216:0]	Input	A set of different signals containing the following information: <ul style="list-style-type: none"> <li>• MSA attributes and status</li> <li>• VB-ID attributes and status</li> <li>• Received video blanking timing</li> </ul> You must connect this set of signals <i>as is</i> from the DisplayPort Intel FPGA IP core to the clock recovery core.
Video Input	Conduit	vidin_clk	vidin_clk	Input	Pixel clock.
			vidin_data (BPP*PIXELS_PER_CLOCK-1:0)	Input	Pixel data.
			vidin_valid	Input	You must assert this signal when all signals on this port are valid.
			vidin_sol	Input	Start of video line.
			vidin_eol	Input	End of video line.
			vidin_sof	Input	Start of video frame.
			vidin_eof	Input	End of video frame.
Video Output	Conduit	rec_clk	rec_clk	Output	Reconstructed video clock.
			rec_clk_x2	Output	Reconstructed video clock double frequency.
			vidout (BPP*PIXELS_PER_CLOCK-1:0)	Output	Pixel data.
			hsync	Output	Horizontal sync. This signal can be active-high or active-low depending on the sync polarity from MSA.
			vsync	Output	Vertical sync. This signal can be active-high or active-low depending on the sync polarity from MSA.

**continued...**

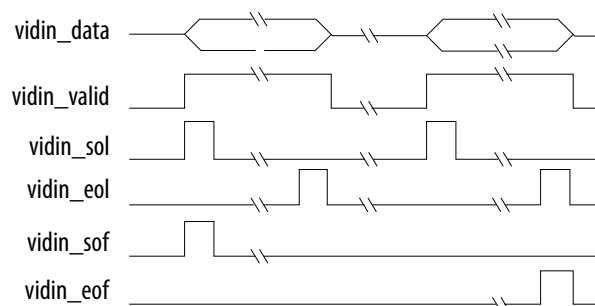


Interface	Port Type	Clock Domain	Port	Direction	Description
			de	Output	Data enable. This signal is always active high.
			field2	Output	The clock recovery core asserts this signal during the second video field for interlaced timings.
			reset_out	Output	The clock recovery core asserts this signal when the other video output signals are not valid. This signal is asynchronous.

#### 4.2.1.2.1. Video Input Port

You must connect the clock recovery core video input port to the DisplayPort sink core video output image port.

**Figure 8. Video Input Port Timing Diagram**



When the `PIXELS_PER_CLOCK` parameter is greater than 1, all input pixels are supposed to be valid when you assert `vidin_valid`. The parameter only supports timings with horizontal active width divisible by 2 (`PIXELS_PER_CLOCK = 2`) or 4 (`PIXELS_PER_CLOCK = 4`).

The clock recovery core video output port produces pixel data with standard `hsync`, `vsync`, or `de` timing. All signals are synchronous to the reconstructed video clock `rec_clk`, unless mentioned otherwise. For designs using a TX transceiver, you can use `rec_clk` as its reference clock.

You can use `rec_clk_x2` as a reference clock for transceivers that have reference clocks with frequencies lower than the minimum pixel clock frequency received. For example, the Video Graphics Array (VGA) 25-MHz resolution when the transceiver's minimum reference clock is 40 MHz.

The clock recovery core asserts `reset_out` when the remaining port signals are not valid. For example, during a recovered video resolution change when the `rec_clk` and `rec_clk_x2` signals are not yet locked and stable. Intel recommends that you use `reset_out` to reset the downstream logic connected to the video output port.

During the hardware demonstration 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. Press one of the push buttons to print the current TX and RX MSA.



### 4.2.2. Transceiver and Clocking

The device's Gigabit transceivers operate at 5.4, 2.7, and 1.62 Gbps, and require a 135-MHz single reference clock. When the link rate changes, the state machine only reconfigures the transceiver PLL settings.

**Table 12. Arria V Transceiver Native PHY TX and RX Settings**

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

Parameters	Single Reference Clock Settings
<b>Datapath Options</b>	
Enable TX datapath	On
Enable RX datapath	On
Enable standard PCS	On
Number of data channels	1, 2 or 4 <i>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.</i>
Bonding mode	×1* or ×N <i>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.</i>
Enable simplified data interface	
<b>PMA</b>	
Data rate	1620 Mbps (when TX maximum link rate = 1.62 Gbps) 2700 Mbps (when TX maximum link rate = 2.7 Gbps) 5400 Mbps (when TX maximum link rate = 5.4 Gbps)
TX local clock division factor	1
<b>TX PMA</b>	
Enable TX PLL dynamic reconfiguration	On
Number of TX PLLs	1
Main TX PLL logical index	0
Number of TX PLL reference clock	1
<b>TX PLL0</b>	
PLL type	CMU
Reference clock frequency	135 MHz
Selected reference clock source	0
Selected clock network	×1 or ×N <i>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.</i>





<b>RX PMA</b>	
Enable CDR dynamic reconfiguration	On
Number of CDR reference clocks	1
Selected CDR reference clock	0
Selected CDR reference clock frequency	135 MHz
PPM detector threshold	1000 ppm
Enable rx_is_lockedtoata port	On
Enable rx_is_lockedtoeref port	On
Enable rx_set_locktoata and rx_set_locktoeref ports	On

<b>Standard PCS</b>	
Standard PCS protocol mode	Basic
Standard PCS/PMA interface width	20

<b>Byte Serializer and Deserializer</b>	
Enable TX byte serializer	Off (when symbol output mode is Dual) On (when symbol output mode is Quad)
Enable RX byte deserializer	Off (when symbol output mode is Dual) On (when symbol output mode is Quad)

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

**Related Information**

- [Arria V GX, GT, SX, and ST Device Datasheet](#)
- [Arria V GZ Device Datasheet](#)
- [Cyclone V Device Datasheet](#)
- [Stratix V Device Datasheet](#)



### 4.2.3. Required Hardware

The hardware demonstration requires the following hardware:

- Intel FPGA kit (includes USB cable to connect the board to your PC); the demonstration supports the following kits:
  - Stratix V GX FPGA Development Kit (5SGXEA7K2F40C2)
  - Arria V GX FPGA Starter Kit (5AGXFB3H4F40C5)
  - Cyclone V GT FPGA Development Kit (5CGTFD9E5F35C7)
- Bitec DisplayPort daughter card (HSMC revision 11 and later)
- 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:* Intel 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.

#### Related Information

- [Stratix V GX FPGA Development Kit](#)
- [Arria V GX FPGA Starter Kit](#)
- [Cyclone V GT FPGA Development Kit](#)

### 4.2.4. 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.

#### 4.2.4.1. 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 Intel FPGA Download Cable II connection. If your version of the board does not have this connection, you can use an external Intel FPGA Download 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).

#### 4.2.4.2. 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\_sk\_4k** for Arria V GX starter kit, **cv** for Cyclone V GT development kit, **sv** for Stratix V development kit, **mst\_av** for Arria V MST design, and **mst\_sv** for Stratix V MST design.

You can also copy the design example through the DisplayPort Intel FPGA parameter editor. Turn on **Generate Example Design** on the DisplayPort Intel FPGA parameter editor before you generate your design. The software copies the SST design example files from `altera/altera_dp/hw_demo/<device_board>` to your working directory.

*Note:* The generated design example may not be aligned to your configured parameter settings.

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

**Table 13. Hardware Demonstration Files for Arria V, Cyclone V, and Stratix V Devices**

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	top.v	Top-level design file.
	bitec_reconfig_alt_<prefix>.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 the clock divider setting. The FSM loops through all the channels and transceiver settings.
	altera_pll_reconfig_core.v altera_pll_reconfig_mif_reader.v altera_pll_reconfig_top.v bitec_cc_fifo.v bitec_cc_pulse.v	Clock recovery core encrypted design files.
<i>continued...</i>		



File Type	File	Description
	bitec_clkrec.v bitec_fpll_cntrl.v bitec_fpll_reconf.v bitec_loop_cntrl.v bitec_vsyncgen.v clkrec_pll_<prefix>.v	
IP Catalog files	video_pll<prefix>.v pll_135.v gxb_reconfig.v gxb_reset.v gxb_rx.v gxb_tx.v	IP Catalog variants for the various helper IP cores.
Platform Designer system	control.qsys	Platform Designer system file.
Intel Quartus Prime IP files	bitec_reconfig_alt_<prefix>.qip bitec_clkrec_dist.qip bitec_clkrec.qip	Intel Quartus Prime IP files that list the required submodule files.
Scripts	runall.tcl	Script to set up the project, generate the IP and Platform Designer system, and compile.
	assignments.tcl	Top-level TCL file to create the project assignments.
	build_ip.tcl	TCL file to build the DisplayPort example design IP blocks.
	build_sw.sh	Script to compile the software.
Miscellaneous	example.sdc	Top-level SDC file.
	bitec_clkrec.sdc	Clock recovery core SDC file.
Software files (in the software directory)	dp_demo_src\ btc_dprx_syslib\ btc_dptx_syslib\ 	Directory containing the example application source code. System library for the RX API. System library for the TX API.

#### 4.2.4.3. Build the FPGA Design

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

```
./runall.tcl (Intel Quartus Prime Standard Edition)
```

This script basically builds the IPs and software, as well as performs Intel Quartus Prime full compilation.

#### 4.2.4.4. Load and Run the Software

In this step, you load the software 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 Platform Designer system.
3. From within the Nios II command shell, execute the following command to program the device, download the Nios II program, and launch a debug terminal:

```
bash nios2-configure-sof <project_name>.sof <USB cable number>; nios2-terminal<USB cable number>
```

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

4. To download the Software .elf file separately, execute the following command in the Nios II command shell:

```
bash nios2-download <project_name>.elf
```

#### Related Information

##### [Nios II Classic Software Developer's Handbook](#)

The Nios II Software Build Tools Reference provides more information about the Nios II commands.

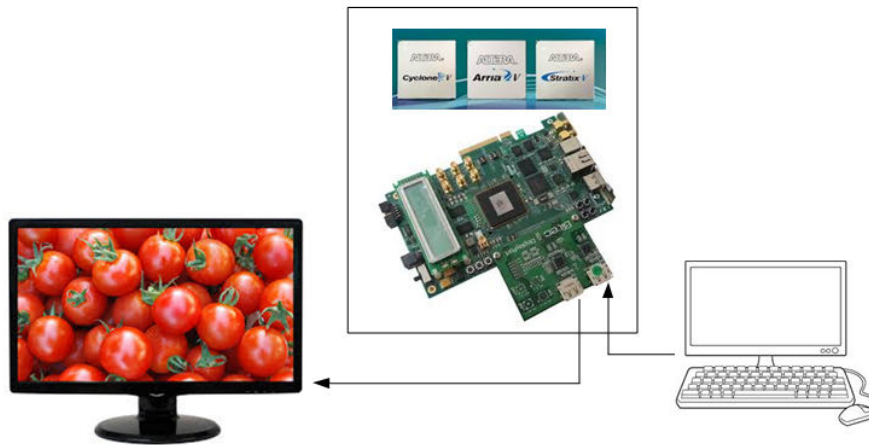
#### 4.2.4.5. View the Results

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.
2. Connect the free end of the Display Port cable that you connected to your PC to the DisplayPort RX on the Bitec daughter card. The PC now has the DisplayPort monitor available as a second monitor. The hardware demonstration loops through and displays the graphic card output as received by the sink core.

*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. Catalyst Control Center, NVIDIA Control Panel) and manually enable the DisplayPort display.

Figure 9. Loop-through Hardware Demonstration



3. You can use your graphic card control panel to adjust the resolution of the DisplayPort 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 push button (CPU\_RESETN) to generate a reset, causing the DisplayPort TX core to re-train the link.

Press push button 0 (USER\_PB[0]) to retrieve 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 10. MSA Output

```

----- TX Main stream attributes -----
--- Stream 0 ---
MISC0 : 20      MISC1 : 00
Mvid   : D42C   Nvid   : 8000
Htotal : 2720   Uttotal : 1646
HSP    : 0000   HSW    : 0032
Hstart : 0112   Ustart : 0043
USP    : 0000   USW    : 0006
Hwidth : 2560   Uheight : 1600
CRC R  : 9065   CRC G  : 9292   CRC B  : edb6
--- Stream 1 ---
MISC0 : 00      MISC1 : 00
Mvid   : 0000   Nvid   : 0000
Htotal : 0000   Uttotal : 0000
HSP    : 0000   HSW    : 0000
Hstart : 0000   Ustart : 0000
USP    : 0000   USW    : 0000
Hwidth : 0000   Uheight : 0000
CRC R  : 0000   CRC G  : 0000   CRC B  : 0000
----- TX Link configuration -----
Lane count : 4
Link rate  : 1620 Mbps
----- RX Main stream attributes -----
--- Stream 0 ---
UB-ID lock : 1   MSA lock : 1
UB-ID : 00 MISC0 : 40 MISC1 : 00
Mvid   : 3FC4   Nvid   : 8000
Htotal : 2720   Uttotal : 1646
HSP    : 0000   HSW    : 0032
Hstart : 0112   Ustart : 0043
USP    : 0000   USW    : 0006
Hwidth : 2560   Uheight : 1600
CRC R  : 36a8   CRC G  : 120e   CRC B  : f1bb
--- Stream 1 ---
UB-ID lock : 1   MSA lock : 1
UB-ID : 00 MISC0 : 40 MISC1 : 00
Mvid   : 2DE6   Nvid   : 8000
Htotal : 2592   Uttotal : 1245
HSP    : 0001   HSW    : 0200
Hstart : 0536   Ustart : 0042
USP    : 0000   USW    : 0006
Hwidth : 1920   Uheight : 1200
----- RX Link configuration -----
CR Done: F      SYM Done: F
Lane count : 4
Link rate  : 5400 Mbps
BER0 : 0000   BER1 : 0000
BER2 : 0000   BER3 : 0000
    
```

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 (SNK) or source (SRC).
- The next two fields show the request and response headers and payloads. The DPCD address field on request messages are decoded into the respective DPCD location names.



When connected and enabled, `USER_PB[0]` on the development board illuminates to indicate that the DisplayPort receiver has locked correctly.

#### 4.2.5. DisplayPort Link Training Flow

Upon Hot Plug detection, the DisplayPort source configures the link through link training.

The DisplayPort source device accesses the sink's DPCD register block through the AUX channel to determine the sink's capability and status and initiate the Link Training command.

The sequence below describes the Link Training flow after HPD assertion:

1. The DisplayPort source reads the DPCD Capabilities fields offset 0x00000 – 0x0000D to determine the sink device's capability.
2. The source writes to the Link Configuration field offset 0x00100 – 0x00101 to configure the Link Bandwidth and Lane Count according to the sink device's requirements.

After Link Configuration, the source initiates Link Training Pattern Sequence 1.

1. The source writes to offset 0x00102 to select Training Pattern 1 and Disable Scrambling. The source sends Training Pattern 1 through the Main Link at the same time.
2. The source writes to offset 0x00103 – 0x00106 to configure the Link Training Control for every lane.
3. The source reads from offset 0x0000E for `TRAINING_AUX_RD_INTERVAL` value.
4. The source waits for a period of time specified in `TRAINING_AUX_RD_INTERVAL` before it reads the Link Status (0x00202 – 0x00207) from the sink device.
5. If the clock recovery core (`CR_DONE`) fails in one or more lanes:
  - The source checks for the Link Driver setting adjust request (0x00206 – 0x00207) and responds accordingly.
  - In the same Link Driver setting, if the source has already repeated Training Pattern Sequence 1 for 5 times, the source will lower the Link Bandwidth (from HBR2 to HBR to RBR) in offset 0x00100 and starts back at Step 1.
  - If the Link Bandwidth is already in the lowest rate (RBR), then Link Training fails.

For Link Training Pattern Sequence 2:

1. The source writes to offset 0x00102 to select Training Pattern 2 and Disable Scrambling. The source sends Training Pattern 2 through the Main Link at the same time.
2. The source writes to offset 0x00103 – 0x00106 to configure the Link Training Control for every lane.
3. The source reads from offset 0x0000E for `TRAINING_AUX_RD_INTERVAL` value.
4. The source waits for a period of time specified in `TRAINING_AUX_RD_INTERVAL` before it reads the Link Status (0x00202 – 0x00207) from the sink device.
5. If `CR_DONE` (0x00202) fails in one or more lanes, abort Training Pattern Sequence 2, and restart Training Pattern Sequence 1.





6. If CR\_DONE passes all lanes, check if the following operations fail or pass:
  - CHANNEL\_EQ\_DONE
  - SYMBOL\_LOCKED
  - INTERLANE\_ALIGN\_DONE
7. If CHANNEL\_EQ\_DONE, SYMBOL\_LOCKED or INTERLANE\_ALIGN\_DONE fails in one or more lanes:
  - The source checks for the Link Driver setting adjust request (0x00206 – 0x00207) and responds accordingly.
  - In the same Link Driver setting, if the source has already repeated Training Pattern Sequence 2 for 5 times, the source will lower the Link Bandwidth (from HBR2 to HBR to RBR) in offset 0x00100, aborts Training Pattern Sequence 2, and restarts Link Training Pattern Sequence 1.
  - If the Link Bandwidth is already in the lowest rate (RBR), then Link Training fails.
8. If Training Pattern Sequence 2 passes, then Link Training completes.
9. The source writes to offset 0x00102 to disable Link Training.

*Note:* If both DisplayPort source and sink support HBR2, replace Training Pattern Sequence 2 with Training Pattern Sequence 3.

#### 4.2.6. DisplayPort Post Link Training Adjust Request Flow (LQA)

After Link Training completes, you can use the Post Link Training Adjust Request Sequence to fine-tune the transmitter driver setting and receiver equalization setting.

The DisplayPort sink supports Post Link Training Adjust Request Sequence feature (as defined in the *VESA DisplayPort Standard 1.3*).

The DisplayPort Intel FPGA IP core controls this feature.

1. During Link Training Sequence, when the source reads DPCD offset 0x00002, and the sink have 0x00002 bit [5] (POST\_LT\_ADJ\_REQ\_SUPPORT) set to 1.
2. If the source supports this feature, it writes to offset 0x00101 bit [5] (POST\_LT\_ADJ\_REQ\_GRANTED) to grant Post Link Training Adjust Request.
3. After Link Training Sequence completes, the source writes to offset 0x00102 to disable Link Training.
4. The sink sets DPCD 0x00204 bit [1] (POST\_LT\_ADJ\_REQ\_IN\_PROGRESS) to 1 and fine-tunes the Link driver setting (Voltage swing and Pre-emphasis).
5. The source reads offset 0x00204 bit [1] to check if Sink Post Link Training Adjust Sequence is in progress.
6. After 5 – 10 ms, the source reads DPCD ADJUST\_REQUEST\_LANE x (0x00206 – 0x00207).
  - If the value changes, the source writes to offset 0x00206 – 0x00207 to configure the Link driver setting accordingly to the requested value.
  - If value not changed, repeat steps 5 – 6. If these steps are repeated 6 times, the source clears offset 0x00101 bit [5] to not grant and proceed to Normal Active Video Transmission.
7. If the sink device's Link Status (0x00202 – 0x00204) clears after step 6,



- Abort Post Link Training Adjust Request Sequence.
- The source clears offset 0x00101 bit [5] (not grant).
- Restart with Link Training Sequence 1.

**Note:** All the `POST_LT_ADJ_REQ` registers and flow definition are available only in the *VESA DisplayPort Standard 1.3*.

### 4.2.7. DisplayPort MST Source User Application

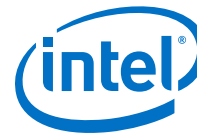
For MST source instantiations, you need to create a user application at the top software layer to invoke the Link Layer level API functions of the `btc_dptxll_syslib` library.

The `btc_dptxll_syslib` library handles most of the Link Layer functionality. The library performs marginal SST operation, which in turn, becomes evident for MST operations. The `btc_dptxll_syslib` library uses the services provided by the `btc_dptx_syslib` library.

You can use the user application to perform MST discovery topology by invoking a single API function (`btc_dptxll_mst_get_device_port()`). In turn, the `btc_dptxll_syslib` library implements this functionality by invoking a number of `btc_dptx_syslib` MST messaging functions such as `btc_dptx_mst_link_address_req()`, `btc_dptx_mst_enum_path_req()`, and `btc_dptx_mst_remote_i2c_rd_req()`.

A typical MST source user application must perform the following steps to display an image on a connected DisplayPort sink device:

1. Wait for HPD signal to become 1.
2. Read the connected sink DPCD version and MST capabilities.
  - If the sink is not MST capable, only a single-stream (SST) connection is possible. In this case, no further action is required as SST connections are mostly handled automatically.
  - If the sink supports MST, skip this step.
3. Perform MST topology discovery by collecting all device ports reachable through the connected sink. Invoke `btc_dptxll_mst_get_device_ports()` until either its outcome is valid or an error is returned. For a successful return value, move to the following step.
4. Browse through the list of the device ports and search for a suitable device output port. This step highly depends on the definition of *suitable device port*. Some applications may require reading of the device port EDID to check the desired resolution supported by the port (use `btc_dptxll_mst_edid_read_req()` and `btc_dptxll_mst_edid_read_rep()` API functions). If a suitable device output port is found, move to the next step.
5. Verify if the main link connection between the DisplayPort source and connected sink is still up.
  - If the link is down, perform a new Link Training.
  - If the link is up, move to the next step.



*Note:* While you can perform the earlier steps even when the main link connection is down, the following steps require the connection to be up. The source needs the connection to calculate the available data bandwidth and make allocation.

6. Set the video pixel rate of the desired stream by invoking `btc_dptx11_stream_set_pixel_rate()`.
7. Calculate the required VCP size for the stream by invoking `btc_dptx11_stream_calc_VCP_size()`.
8. Verify if the required VCP size (number of time slots needed to transport the stream) is available to transport to the desired device output port. Then, move to the next step.
9. Allocate the stream data to be transported to the desired device output port by invoking `btc_dptx11_stream_allocate_req()`
10. Wait for the source to make allocation. Invoke `btc_dptx11_stream_allocate_rep()` until either the allocation is complete or an error is returned. For a successful allocation, move to the following step.
11. The allocation of the stream to the device output port completes. MST data transport is now active.
12. Handle received `CONNECTION_STATUS_NOTIFY` messages according to the changed topology.

## 5. DisplayPort Source

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.

**Figure 11. DisplayPort Source Top-Level Block Diagram**

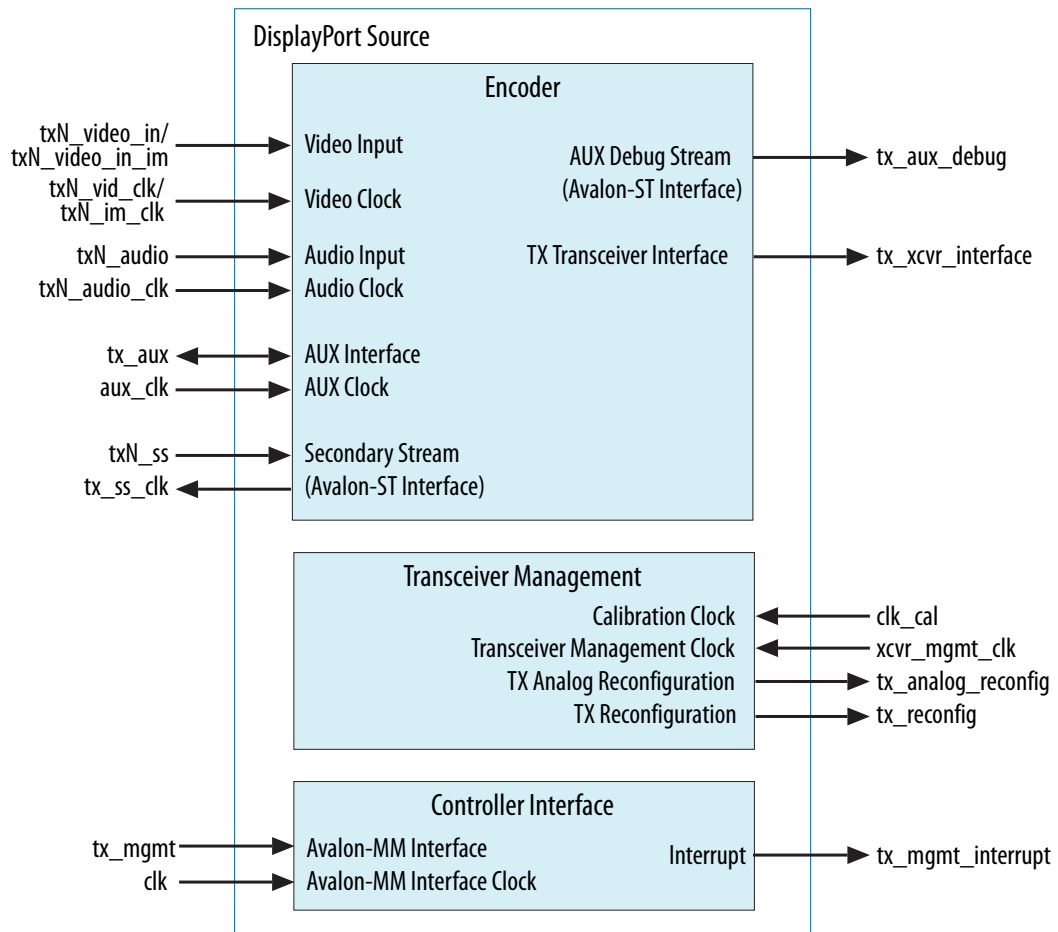
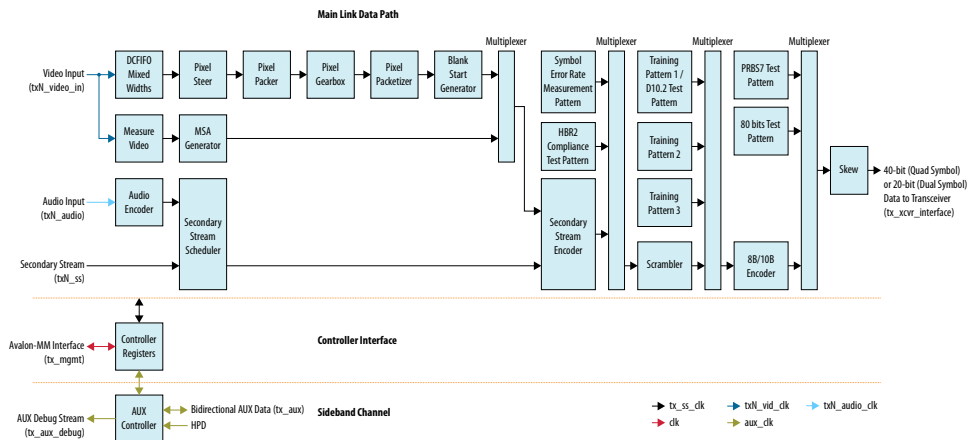




Figure 12. DisplayPort Source Functional Block Diagram



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, such as color reordering, before processing it using the txN\_video\_in input. N represents the stream number: tx\_video\_in (Stream 0), tx1\_video\_in (Stream 1), tx2\_video\_in (Stream 2), and tx3\_video\_in (Stream 3). Streams 1, 2, and 3 are only available when you turn on the **Support MST** parameter and specify the **Max stream count** parameter to 2, 3, or 4 streams respectively.

The video data width supports 6 to 16 bits per color (bpc) and is user selectable. If you set **Pixel input mode** to Dual or Quad, the video input can accept two or four pixels per clock, thereby extending the pixel clock rate capability.

## 5.1. Main Data Path

The main link data path consists of the video packetizer, video geometry measurement, audio and secondary stream encoder, and training and link quality patterns generator.

The IP core multiplexes data from these four paths and transmits it through a scrambler and an 8B/10B encoder. All the symbols, both those transmitted during video display period and those transmitted during video blanking period, are skewed by two Link symbol period between adjacent lanes.

### 5.1.1. Video Packetizer Path

The video packetizer path provides video data resampling and packetization.

The video packetizer path consists of the following steps:

1. The mixed-width DCFIFO crosses the video data from the video clock domain (txN\_vid\_clk) into the main link clock domain (tx\_ss\_clk) generated by the transceiver. This main clock can be 270, 202.5, 135, 81, 67.5, or 40.5 MHz, depending on the actual main link rate requested and the symbols per clock.
2. The pixel steer block aligns the video data so that the first active pixel of each video line occupies the least significant position.
3. The pixel packer block decimates the video data to the requested lane count (1, 2, or 4).

4. The pixel gearbox block 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 maximum color depths you need instead of the complete set of color depths specified in the *VESA DisplayPort Standard*.
5. The DisplayPort Intel FPGA IP core packetizes the resampled data. The *VESA DisplayPort Standard* 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. The pixel packetizer punctuates the outgoing video stream with the correct packet comma codes, such as blank end (BE), fill start (FS), and fill end (FE). Internally, the pixel packetizer uses a symbol and a TU counter to ensure that it respects the TU boundaries.
6. The blank start generator determines when to send the blank start (BS) comma codes with their corresponding video data packets. This block operates in enhanced or standard framing mode.

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

### 5.1.2. Video Geometry Measurement Path

The video geometry measurement path determines the video geometry (such as HTOTAL, VTOTAL, and VHEIGHT) required for the DisplayPort main stream attributes (MSA), which are sent once every vertical blanking interval.

The MSA generator provides the MSA packet framed with secondary start (SS) and secondary end (SE) comma codes based on the requested lane count. The multiplexer then combines the packetized data from the video packetizer path and the MSA data into a single stream.

### 5.1.3. Audio and Secondary Stream Encoder Path

The audio encoder generates the Audio InfoFrame, Audio Timestamp, and Audio Sample packets from the incoming audio sample data stream. The secondary stream scheduler arbitrates the data flow among the Audio InfoFrame, Audio Timestamp, and Audio Sample packets and the incoming secondary stream packet into a single secondary stream in a round robin method.

Based on the requested lane count, the secondary stream encoder packetizes and inserts the secondary stream packets into the combined packetized video and MSA data.

The secondary stream encoder path consists of the following steps:

1. The secondary stream encoder determines the valid windows of opportunity during vertical and horizontal blanking regions for secondary stream packets.
2. The secondary stream encoder derives the parity byte and performs nibble interleaving for enhancing error-correcting capability.
3. The encoder packetizes the secondary stream packets with SS and SE.
4. The encoder inserts the secondary stream packets into the merged video and MSA data.



#### 5.1.4. Training and Link Quality Patterns Generator

The IP core multiplexes the packetized data, MSA data, and blank generator data into a single stream.

The combined data goes through a scrambler and an 8B/10B encoder, 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 Intel FPGA high-speed output transceiver.

During training periods, the source can send the DisplayPort clock recovery and symbol lock test patterns (training pattern 1, training pattern 2, and training pattern 3, respectively), upon receiving the request from downstream DisplayPort sink.

The DisplayPort source also supports a test procedure for measuring the link quality, including these features:

- Transmission of a Nyquist pattern (repetition of D10.2 symbols without scrambling)
- Symbol Error measurement pattern
- PRBS7 bit pattern
- Custom 80-bit repeating pattern
- HBR2 Compliance EYE pattern

Only the Symbol Error measurement pattern and HBR2 Compliance EYE pattern require both scrambling and 8B/10B encoding. The PRBS7 pattern and Custom 80-bit pattern do not require scrambling or 8B/10B encoding. Training patterns 1, 2, and 3, and D10.2 test pattern require only 8B/10B encoding.

### 5.2. 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 controls the main link data path and the sideband channel.

### 5.3. Sideband Channel

The DisplayPort Intel FPGA IP core uses the sideband communication over sideband channel (AUX channel and HPD) to manage topology and virtual channel connection/main link, and performs main link symbol mapping.

The AUX controller interface works with a simple serial-port-type peripheral that operates in a polled mode. It captures all bytes sent from and received by the AUX channel, which is useful for debugging. The IP core clocks the AUX controller using a 16 MHz clock input (`aux_clk`).

## 5.4. Source Embedded DisplayPort (eDP) Support

The DisplayPort Intel FPGA 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 source IP core supports:

- Full (normal) link training—default
- Fast link training—mandatory eDP feature

## 5.5. Source Interfaces

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

**Table 14. 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	32-bit word addressing address
			tx_mgmt_chipselect	Input	Assert for valid read or write access
			tx_mgmt_read	Input	Assert to indicate a read transfer
			tx_mgmt_write	Input	Assert to indicate a write transfer
			tx_mgmt_writedata[31:0]	Input	Data for write transfers
			tx_mgmt_readdata[31:0]	Output	Data for read transfers
			tx_mgmt_waitrequest	Output	Asserted when the DisplayPort Intel FPGA IP core is unable to respond to a read or write request. Forces the GPU to wait until the IP core is ready to proceed with the transfer.
tx_mgmt_irq	IRQ	clk	tx_mgmt_irq	Output	Interrupt for embedded controller

**Table 15. 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
clk_cal	Clock	N/A	clk_cal	Input	A 50-MHz calibration clock input. This clock must be synchronous to the clock used for the Transceiver

**continued...**





Interface	Port Type	Clock Domain	Port	Direction	Description
					Reconfiguration block (xcvr_mgmt_clk), external to the DisplayPort source.
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_link_rate_8bits[7:0]	Output	
			tx_reconfig_req	Input	
			tx_reconfig_ack	Input	
			tx_reconfig_busy	Input	

**Note:** Value of tx\_link\_rate[1:0]: 0 = 1.62 Gbps, 1 = 2.70 Gbps, 2 = 5.40 Gbps, 3 = 8.10 Gbps; value of tx\_link\_rate\_8bits[7:0]: 0x06 = 1.62 Gbps, 0x0a = 2.70 Gbps, 0x14 = 5.40 Gbps, 0x1e = 8.10 Gbps.

**Note:** For devices using a 50-MHz xcvr\_mgmt\_clk clock, connect the same clock directly also to the clk\_cal signal. For devices using a 100-MHz xcvr\_mgmt\_clk clock, connect the same clock to clk\_cal signal through a by-2 divider.

[Transceiver Analog Reconfiguration Interface](#) on page 58

[Transceiver Reconfiguration Interface](#) on page 57

### Video Interface

When you turn off **Enable Video input Image port**, the source uses the standard HSYNC/VSYNC/DE ports in txN\_vid\_clk and txN\_video\_in interfaces.

**Table 16. Video Interface (HSYNC/VSYNC/DE Interface)**

v is the number of bits per color, p is the pixels per clock (1 = single, 2 = dual, and 4 = quad). N is the stream number; for example, tx\_vid\_clk represents Stream 0, tx1\_vid\_clk represents Stream 1, and so on.

Interface	Port Type	Clock Domain	Port	Direction	Description
txN_vid_clk	Clock	N/A	txN_vid_clk	Input	Video clock
txN_video_in	Conduit	txN_vid_clk	txN_vid_data[3v*p-1:0]	Input	Video data and standard H/V synchronization video port input
			txN_vid_v_sync[p-1:0]	Input	
			txN_vid_h_sync[p-1:0]	Input	
			txN_vid_de[p-1:0]	Input	

When you turn on **Enable Video input Image port**, the source uses the txN\_im\_clk and txN\_video\_in\_im interfaces.

**Table 17. Video Interface (TX Video IM Interface)**

v is the number of bits per color, p is the pixels per clock (1 = single, 2 = dual, and 4 = quad). N is the stream number; for example, tx\_im\_clk represents Stream 0, tx1\_im\_clk represents Stream 1, and so on.

Interface	Port Type	Clock Domain	Port	Direction	Description
txN_im_clk	Clock	N/A	txN_im_clk	Input	Video Image clock
txN_video_in	Conduit	txN_im_clk	txN_im_sol	Input	Start of video line
			txN_im_eol	Input	End of video line
			txN_im_sof	Input	Start of video frame
			txN_im_eof	Input	End of video frame
			txN_im_data[3v*p-1:0]		Video input data
			txN_im_valid[p-1:0]		Video data valid. Each bit must assert when all other signals on this port are valid and the corresponding pixel belongs to active video.
			txN_im_locked		Video locked <ul style="list-style-type: none"> <li>0 = Unlocked</li> <li>1 = Locked</li> </ul>
			txN_im_interlace		Video interlaced <ul style="list-style-type: none"> <li>0 = Progressive video</li> <li>1 = Interlaced video</li> </ul>
txN_im_field		Video field <ul style="list-style-type: none"> <li>0 = Bottom field (or progressive)</li> <li>1 = Top field</li> </ul>			

**Table 18. 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	Active-high AUX channel reset
tx_aux	Conduit	aux_clk	tx_aux_in	Input	AUX channel data input
			tx_aux_out	Output	AUX channel data output
			tx_aux_oe	Output	Output buffer enable
			tx_hpd	Input	Hot plug detect
tx_aux_debug	AV-ST	aux_clk	tx_aux_debug_data[31:0]	Output	Formatted AUX channel debug data
			tx_aux_debug_valid	Output	Asserted when all the other signals on this port are valid
			tx_aux_debug_sop	Output	Start of packet (start of AUX request or reply)

*continued...*



Interface	Port Type	Clock Domain	Port	Direction	Description
			tx_aux_debug_eop	Output	End of packet (end of AUX request or reply)
			tx_aux_debug_err	Output	Asserted when an AUX channel bit error is detected
			tx_aux_debug_cha	Output	The channel number for data being transferred on the current cycle. Used as AUX channel data direction. 0 = Reply (from DisplayPort sink) 1 = Request (to DisplayPort sink)

[AUX Interface](#) on page 53

**Table 19. Secondary Interface**

N is the stream number; for example, tx<sub>ss</sub> represents Stream 0, tx<sub>1ss</sub> represents Stream 1, and so on.

Interface	Signal Type	Clock Domain	Port	Direction	Description
tx <sub>ss</sub> _clk	Clock	N/A	tx <sub>ss</sub> _clk	Output	TX transceiver clock out and clock for secondary stream
Secondary Stream (tx <sub>N</sub> <sub>ss</sub> )	AV-ST	tx <sub>ss</sub> _clk	tx <sub>N</sub> <sub>ss</sub> _data[127:0]	Input	Secondary stream interface
			tx <sub>N</sub> <sub>ss</sub> _valid	Input	
			tx <sub>N</sub> <sub>ss</sub> _ready	Output	
			tx <sub>N</sub> <sub>ss</sub> _sop	Input	
			tx <sub>N</sub> <sub>ss</sub> _eop	Input	

[Secondary Stream Interface](#) on page 58

**Table 20. Audio Interface**

m is the number of TX audio channels. N is the stream number; for example, tx<sub>audio</sub> represents Stream 0, tx<sub>1audio</sub> represents Stream 1, and so on.

Interface	Signal Type	Clock Domain	Port	Direction	Description
Audio (tx <sub>N</sub> <sub>audio</sub> )	Clock	N/A	tx <sub>N</sub> <sub>audio</sub> _clk	Input	Audio clock
	Conduit	tx <sub>N</sub> <sub>audio</sub> _clk	tx <sub>N</sub> <sub>audio</sub> _lpcm_data[m*32-1:0]	Input	m channels of 32-bit audio sample data.
			tx <sub>N</sub> <sub>audio</sub> _valid	Input	Must be asserted when valid data is available on tx <sub>N</sub> <sub>audio</sub> _lpcm_data
			tx <sub>N</sub> <sub>audio</sub> _mute	Input	Must be asserted when audio is muted

[Audio Interface](#) on page 62

**Table 21. TX Transceiver Interface**

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

*Note:* Connect the DisplayPort signals to the Native PHY signals of the same name.

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. Equivalent to Link Speed Clock (ls_clk).
	Conduit	tx_std_clkout	tx_parallel_data[n*s*10-1:0]	Output	Parallel data for TX transceiver
	Conduit	N/A	tx_pll_powerdown	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 <i>Note:</i> Required only for Arria V, Cyclone V, and Stratix V devices.
	Conduit	N/A	tx_analogreset[n-1:0]	Output	Resets the analog TX portion of TX transceiver <i>Note:</i> Required only for Arria V, Cyclone V, and Stratix V devices.
	Conduit	N/A	tx_cal_busy[n-1:0]	Input	Calibration in progress signal from TX transceiver
	Conduit	N/A	tx_pll_locked	Input	PLL locked signal from TX transceiver

[Transceiver Reconfiguration Interface](#) on page 57

### 5.5.1. 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 and DPCD Locations](#) on page 138

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



## 5.5.2. 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 source's AUX controller captures 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.

### Related Information

- [AN 522: Implementing Bus LVDS Interface in Supported Altera Device Families](#)
- [AN 745: Design Guidelines for Intel FPGA DisplayPort Interface](#)  
Provides more information about the AUX channel circuitry implementation.

## 5.5.3. Video Interface

The core sends video to be encoded through the txN\_video\_in or txN\_video\_in\_im interface, depending on whether or not you turn on the **TX Video IM Enable** parameter.

**Table 22. Video Input Feature Comparisons**

The table below shows the simplified comparison between the 2 different ways to feed video data to the source core.

Interface	Video Data Constraints	Calculated MSA Parameters	User-provided Required MSA Parameters	User-provided Optional MSA Parameters	Adaptive Sync Support
txN_video_in	<ul style="list-style-type: none"> <li>• HS/VS/DE and real pixel clock available</li> <li>• Video data temporally correct</li> </ul>	All	None	None	No
txN_video_in_im	Video data temporally correct	<ul style="list-style-type: none"> <li>• MVID</li> <li>• HWIDTH</li> <li>• VHEIGHT</li> </ul>	HTOTAL	<ul style="list-style-type: none"> <li>• VTOTAL</li> <li>• HSP</li> <li>• HSW</li> <li>• HSTART</li> <li>• VSTART</li> <li>• VSP</li> <li>• VSW</li> </ul>	Yes

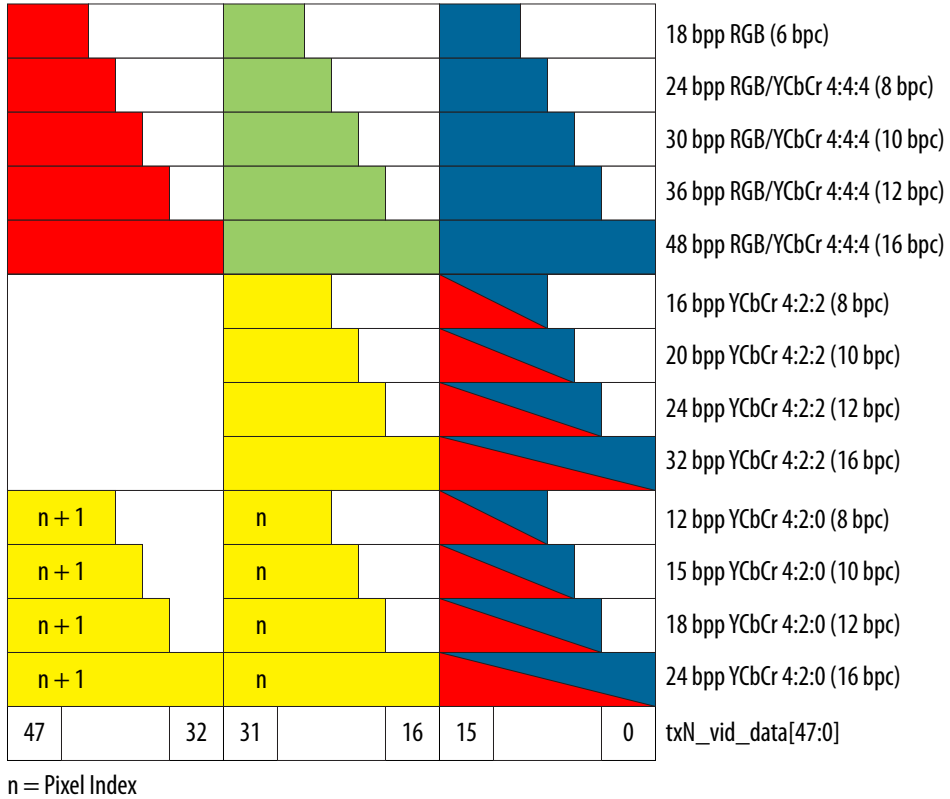
### 5.5.3.1. Video Interface (TX Video IM Enable = 0)

If you do not enable the video image interface feature, the core uses the traditional HSYNC/VSYNC/DE video input interface (txN\_video\_in).

You specify the data input width through the **Maximum video input color depth** parameter. The same input port transfers RGB and YCbCr data in 4:4:4, 4:2:2, or 4:2:0 color format. Data is most-significant bit aligned.

**Figure 13. Video Input Data Format**

18 bpp to 48 bpp for RGB/YCbCr 4:4:4, 16 bpp to 32 bpp for YCbCr 4:2:2, and 12 bpp to 24 bpp for YCbCr 4:2:0 port width when txN\_video\_in port width is 48 (**Maximum video input color depth = 16 bpc, Pixel input mode = Single**)



**Table 23. Video Ports for 4:2:2 and 4:2:0 Color Formats**

Color Format	Description
Sub-sampled 4:2:2 color format	<ul style="list-style-type: none"> <li>Video port bits 47:32 are unused</li> <li>Video port bits 31:16 always transfer the Y component</li> <li>Video port bits 15:0 always transfer the alternate Cb or Cr component</li> </ul>
Sub-sampled 4:2:0 color format	<ul style="list-style-type: none"> <li>For even lines (starting with line 0)               <ul style="list-style-type: none"> <li>Video port bits 47:32 always transfer the <math>Y_{n+1}</math> component.</li> <li>Video port bits 31:16 always transfer the <math>Y_n</math> component.</li> <li>Video port bits 15:0 always transfer the <math>Cb_n</math> component.</li> </ul> </li> <li>For odd lines               <ul style="list-style-type: none"> <li>Video port bits 47:32 always transfer the <math>Y_{n+1}</math> component.</li> <li>Video port bits 31:16 always transfer the <math>Y_n</math> component.</li> <li>Video port bits 15:0 always transfer the <math>Cr_n</math> component.</li> </ul> </li> </ul>

**Note:** The frequency of txN\_vid\_clk must be halved when YCbCr 4:2:0 is used because two pixels are fed into a single clock cycle.



**Table 24. YCbCr 4:2:0 Input Data Ordering Compared to RGB 4:4:4**

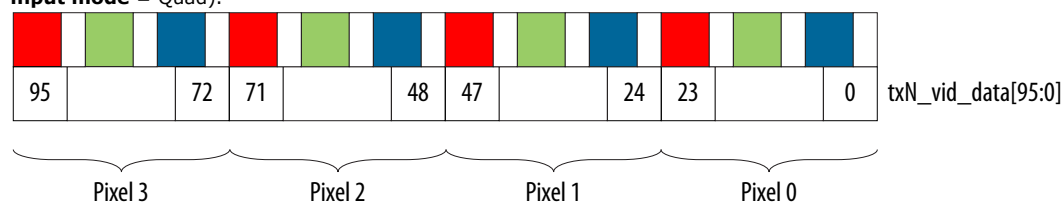
Pixel Indexes	R Position	G Position	B Position
0 and 1	Y1	Y0	<ul style="list-style-type: none"> <li>• Cb0 (Even lines)</li> <li>• Cr0 (Odd lines)</li> </ul>
2 and 3	Y3	Y2	<ul style="list-style-type: none"> <li>• Cb2 (Even lines)</li> <li>• Cr2 (Odd lines)</li> </ul>
4 and 5	Y5	Y4	<ul style="list-style-type: none"> <li>• Cb4 (Even lines)</li> <li>• Cr4 (Odd lines)</li> </ul>
...	...	...	...

If you set **Pixel input mode** to Dual or Quad, the IP core sends 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 data enable, horizontal sync, and vertical sync signals are widened.

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

**Figure 14. Video Input Data Alignment**

For RGB 18 bpp when txN\_video\_in port width is 96 (**Maximum video input color depth** = 8 bpc, **Pixel input mode** = Quad).



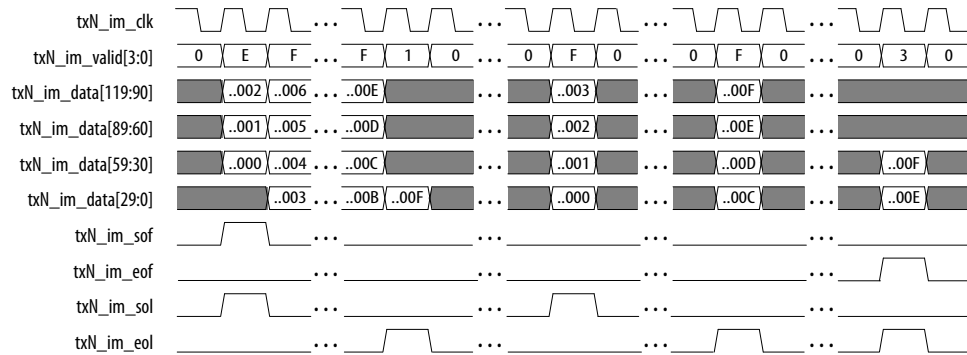
### 5.5.3.2. Video Interface (TX Video IM Enable = 1)

If you enable the video image interface feature, the core uses the video image interface (txN\_video\_in\_im).

The txN\_video\_in\_im ports replace the txN\_video\_in ports when you turn on the **TX Video IM Enable** parameter. The txN\_video\_in\_im ports (N = 0 to 3) transmit video data when either the horizontal/vertical syncs or the exact pixel clock is not available. The streams need synchronization pulses at the start and end of active lines and active frames.

The timing diagram below illustrates the behavior of the ports when TX\_PIXELS\_PER\_CLOCK = 4, TX\_VIDEO\_BPC = 10, and line length = 16 pixels.

**Figure 15. Video Image Interface Ports Timing Diagram**



- You specify the data input width through the **Maximum video input color depth** parameter. The core uses the same output port to transfer both RGB and YCbCr data in either 4:4:4, 4:2:2, or 4:2:0 color format.
- The data organization and pixel ordering of the txN\_im\_data ports are identical to the ones of the txN\_vid\_data signals.
- When you configure the **Pixel input mode** parameter to Dual or Quad, the IP core sends two or four pixels in parallel respectively.
- The txN\_im\_valid signal is widened to support video horizontal resolutions not divisible by two or four. For example, if TX\_PIXELS\_PER\_CLOCK = 2, txN\_im\_valid[0] must assert when pixel N belongs to active video and txN\_im\_valid[1] must assert when pixel N+1 belongs to active video.
- For interlaced video, the core samples txN\_im\_field when txN\_im\_sof asserts. When txN\_im\_field asserts, it marks txN\_im\_data as belonging to the top field.
- The frequency of the txN\_im\_clk signal must be equal to or higher than the frequency of the maximum video pixel clock to be transmitted divided by the pixel input mode.
- Not all clock cycles need to contain valid (active) pixel data; only those indicated by the assertion of txN\_im\_valid.
- The txN\_video\_in\_im ports support the Adaptive Sync feature.

The source core measures only some of the MSA parameters from the incoming video signal:

- MVID
- HWIDTH
- VHEIGHT VSP and VSW

The GPU MSA registers for the remaining MSA parameters are Read/Write and you can set the value for these parameters:

- HTOTAL and VTOTAL
- HSP and HSW
- HSTART and VSTART





**Note:** The source core needs only `HTOTAL` because the core calculates the value of `MVID` from the interval time between `txN_im_s01` pulses and the amount of pixels accounted for. The source core ignores the rest of the MSA parameters and forwards to the connected sink.

### 5.5.4. TX Transceiver Interface

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

The DisplayPort Intel FPGA IP core 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.

### 5.5.5. Transceiver Reconfiguration Interface

You can reconfigure the transceiver to accept single reference clock. The single reference clock is a 135 MHz clock for all bit rates: RBR, HBR, HBR2, and HBR3.

During run-time, you can reconfigure the transceiver to operate in either one of the bit rates by changing TX 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

- [AN 645: Dynamic Reconfiguration of PMA Controls in Stratix V Devices](#)  
Provides more information about using the Transceiver Reconfiguration Controller to reconfigure the Stratix V Physical Media Attachment (PMA) controls dynamically.
- [V-Series Transceiver PHY IP Core User Guide](#)  
Provides more information about how to reconfigure the transceiver for 28-nm devices.
- [AN 676: Using the Transceiver Reconfiguration Controller for Dynamic Reconfiguration in Arria V and Cyclone V Devices](#)  
Provides 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](#)  
Provides more information about link tuning.
- [Intel Arria 10 Transceiver PHY User Guide](#)  
Provides more information about how to reconfigure the transceiver for Intel Arria 10 devices.



- [Intel Cyclone 10 GX Transceiver PHY User Guide](#)  
Provides more information about how to reconfigure the transceiver for Intel Cyclone 10 GX devices.
- [Intel Stratix 10 L- and H-tile Transceiver PHY User Guide](#)  
Provides more information about how to reconfigure the transceiver for Intel Stratix 10 L-tile and H-tile devices.

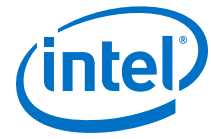
### 5.5.6. Transceiver Analog Reconfiguration Interface

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.

### 5.5.7. Secondary Stream Interface

You can transmit the secondary stream data over the DisplayPort main link through the secondary stream (`txN_ss`) interface. This interface uses handshaking and back pressure to control packet delivery.

*Note:* The DisplayPort Intel FPGA IP core supports InfoFrame SDP versions 1.2 and 1.3 over the Main-Link. InfoFrame SDP version 1.2 shall be used to convey Audio InfoFrame control information, as specified in *CEA-861-F* and *CEA-861.2*. Other InfoFrame coding types, as specified in *CEA-861-F, Table 5*, and *CEA-861.3*, shall use InfoFrame SDP version 1.3. Refer to the *DisplayPort Specification Section 2.2.5.1* for detailed definition.

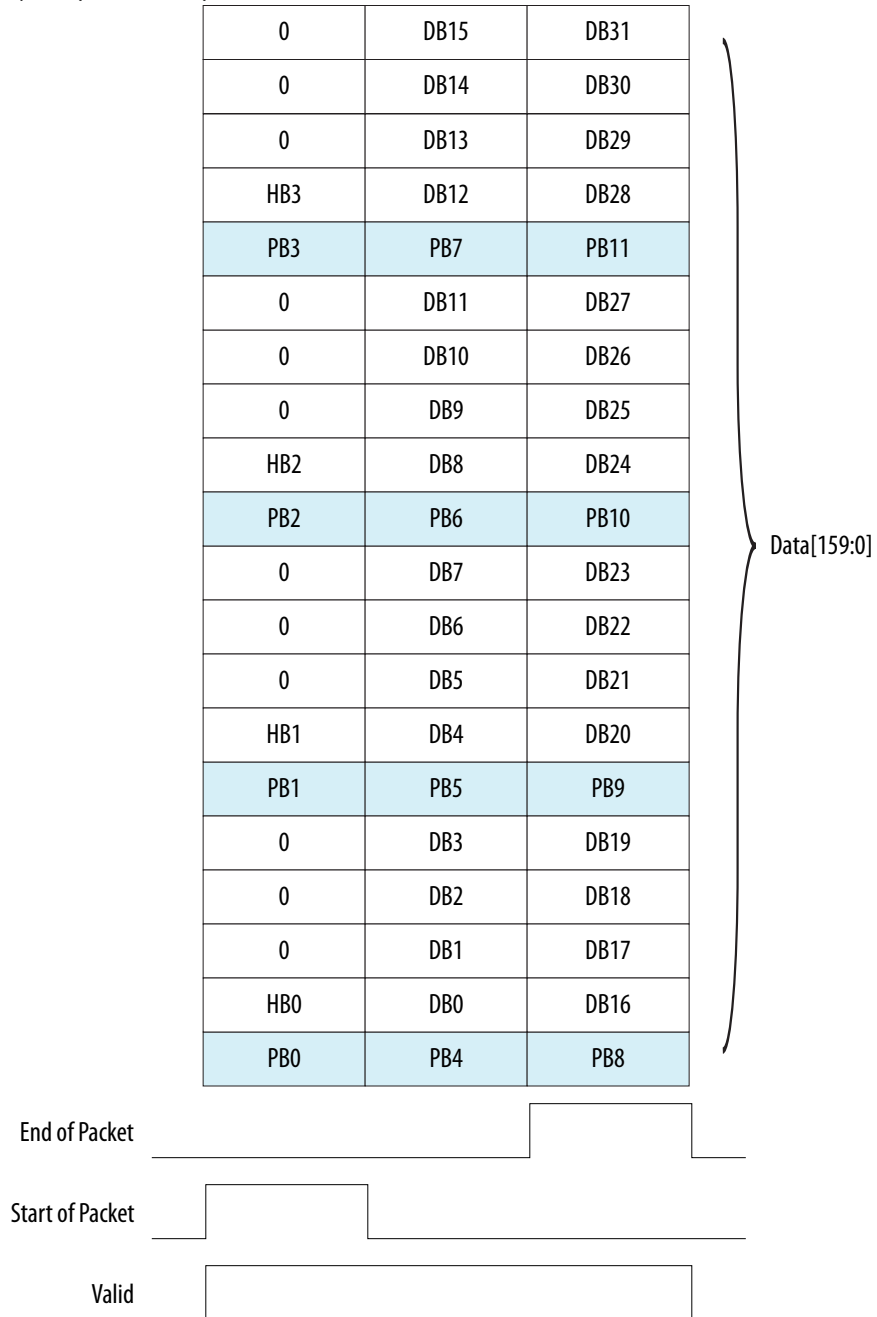


**Figure 16. Secondary Stream Input Data Format**

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

**Figure 17. Typical Secondary Stream Packet**

This figure shows a typical secondary stream packet with a four-byte header (HB0, HB1, HB2 and HB3) and a 32-byte payload (DB0 ... DB31).

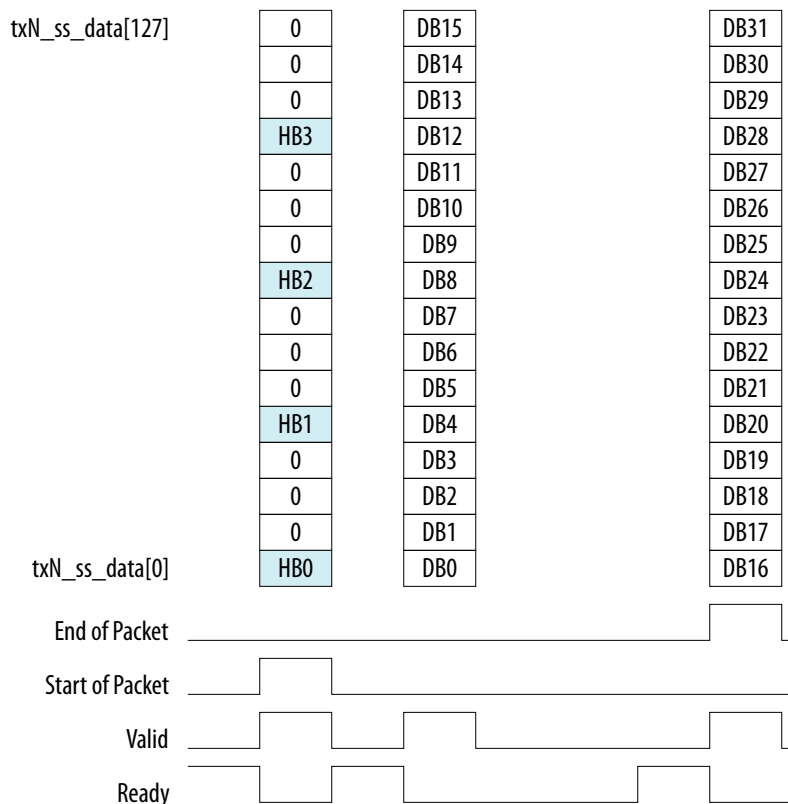


The core calculates the associated parity bytes. The secondary stream interface uses the start-of-packet (SOP) and end-of-packet (EOP) to determine if the current input is a header or payload.



The ready latency is 1 clock cycle for the payload sub-packets. When core is ready, it sends the header forward. When the header is forwarded, the 16-byte payload (DB0 ... DB15 and DB16 ... DB31) must be available and the core must assert its associated valid signal on the next clock cycle when the output ready signal is high. The valid signal must remain low until the ready signal is high.

**Figure 18. Typical Secondary Stream Packet Flow**



The core supports only 16-byte and 32-byte payloads. Payloads that contain only the first 16 data bytes can assert the EOP on the second valid pulse to terminate the packet sequence. The core clocks in the data to the secondary stream interface through `tx_ss_clk`. `tx_ss_clk` is at the same phase and frequency as the main link lane 0 clock.

You can also use the secondary stream data packet to transport HDR metadata. CTA-861-G specification defines the HDR InfoFrame packet information as Packet Type, Version, data packets, and so on. The HDR metadata must follow InfoFrame SDP version 1.3 format defined in the *VESA DisplayPort Standard version 1.4a*.

For example, if the CTA-861-G specification-defined HDR InfoFrame type is 0x07, the *VESA DisplayPort Standard version 1.4a*-defined SDP InfoFrame Header Byte 1 as secondary-data packet type is 80h + Non-audio InfoFrame type value. The Header Byte 1 (HB1 in [Figure 18](#) on page 61) must be written to 87h.

### 5.5.8. Audio Interface

The audio encoder is upstream of the secondary stream encoder. It generates the Audio InfoFrame, Audio 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.

You can configure the audio port for the number of audio channels required in the design. You can use 2 or 8 channels. Each channel's audio data is sent to the txN\_audio\_lpcm\_data port.

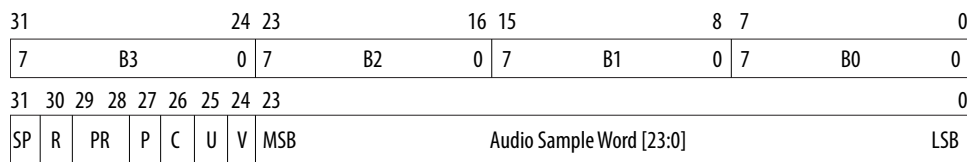
- Channel 1 audio data should be present at txN\_audio\_lpcm\_data[31:0]
- Channel 2 audio data should be present at txN\_audio\_lpcm\_data[63:32] and so on.

The IP core requires a txN\_audio\_valid signal for designs in which the txN\_audio\_clk signal is higher than the actual sample clock. The txN\_audio\_valid signal qualifies the audio data on the txN\_audio\_lpcm\_data input. If txN\_audio\_clk is the actual sample clock, you can tie the txN\_audio\_valid signal to 1.

The figure and table below illustrate the audio sample data bits and bit field definitions, respectively.

**Figure 19. Audio Sample Data Bits**

The packing format complies to both IEC-60958-1 and IEC-60958-3 standards.



**Table 25. 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.
SP	Byte 3, bit 7	Sample present bit: 1: Sample information is present and can be processed.

*continued...*



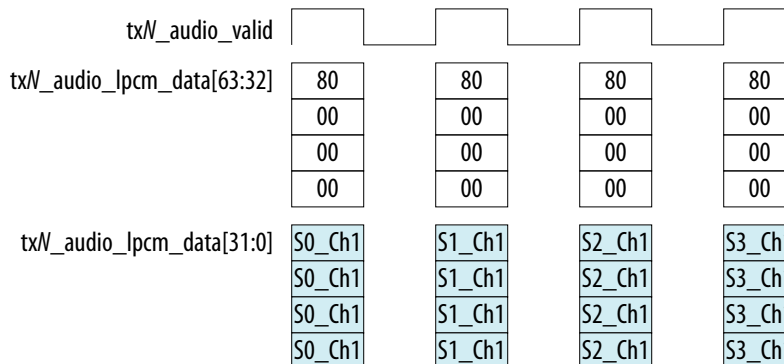
Bit Name	Bit Position	Description
		<p>0: Sample information is not present.</p> <p>All one-sample channels, used or unused, must have the same sample present bit value.</p> <p>This bit is useful for situations in which 2-channel audio is transported over a 4-lane main link. In this operation, main link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

When you configure the DisplayPort Intel FPGA IP core for 2 or 8 channels, you can transmit any number of audio channels fewer than or equal to the number of channels you selected.

To transmit 1 channel of audio over the IP core configured at 2 audio channels:

- You must configure the source audio register's CH\_COUNT bits to 000b using the embedded controller.
- You also need to set the SP bit to 1 and the other bits to 0 on the txN\_audio\_lpcm\_data[63:32] signal. The IP core performs 2-channel layout mapping for 1 and 2 audio channels, which requires the SP bit to be the same for all one-sample channels.

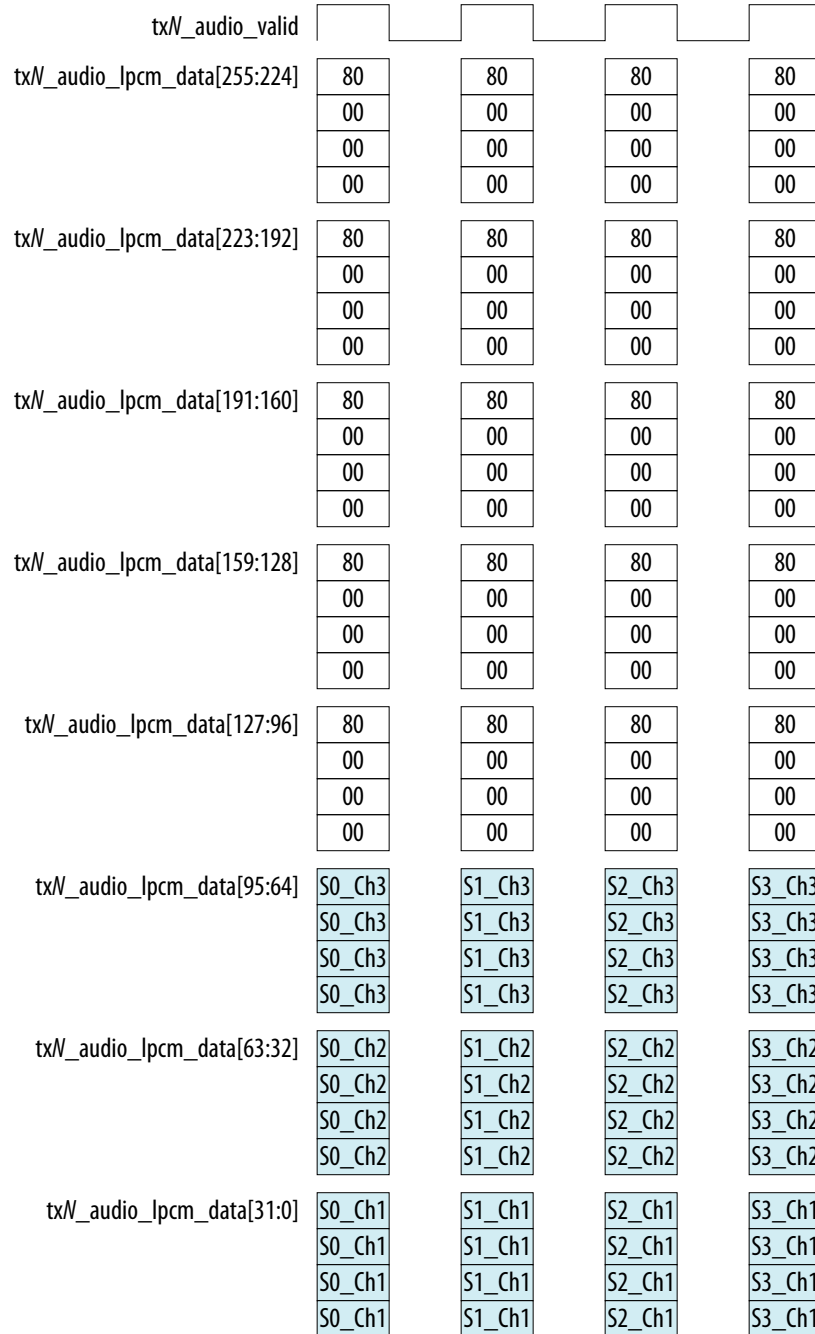
**Figure 20. Typical 1-Channel Audio Flow Over 2-Channel Audio TX Core**



To transmit 3-8 channels of audio, the IP core performs 8-channel layout mapping. For example, to transmit 3 audio channels over the IP core configured at 8 audio channels:

- You must configure the source audio register's CH\_COUNT bits to 010b using the embedded controller.
- You also need to provide the data as shown in the figure below.

Figure 21. Typical 3-Channel Audio Flow Over 8-Channel Audio TX Core



The DisplayPort Intel FPGA IP core internally calculates the Maud based on a fixed (8000h) to generate the Audio Timestamp packet. The IP core generates the Audio InfoFrame packet based on the information from the DisplayPort source audio registers: LFEBPL, CA, LSV, and DM\_INH. The IP core continues transmitting the Audio Timestamp, Audio InfoFrame, and Audio Sample packets even when the main





video stream is no longer transmitting. When there is no video stream, the IP core transmits an Audio Sample packet after each BS symbol, and transmits an Audio Timestamp and Audio InfoFrame once after every 512<sup>th</sup> BS symbol set.

The source automatically generates the Audio InfoFrame and fills it with only information about the number of channels used.

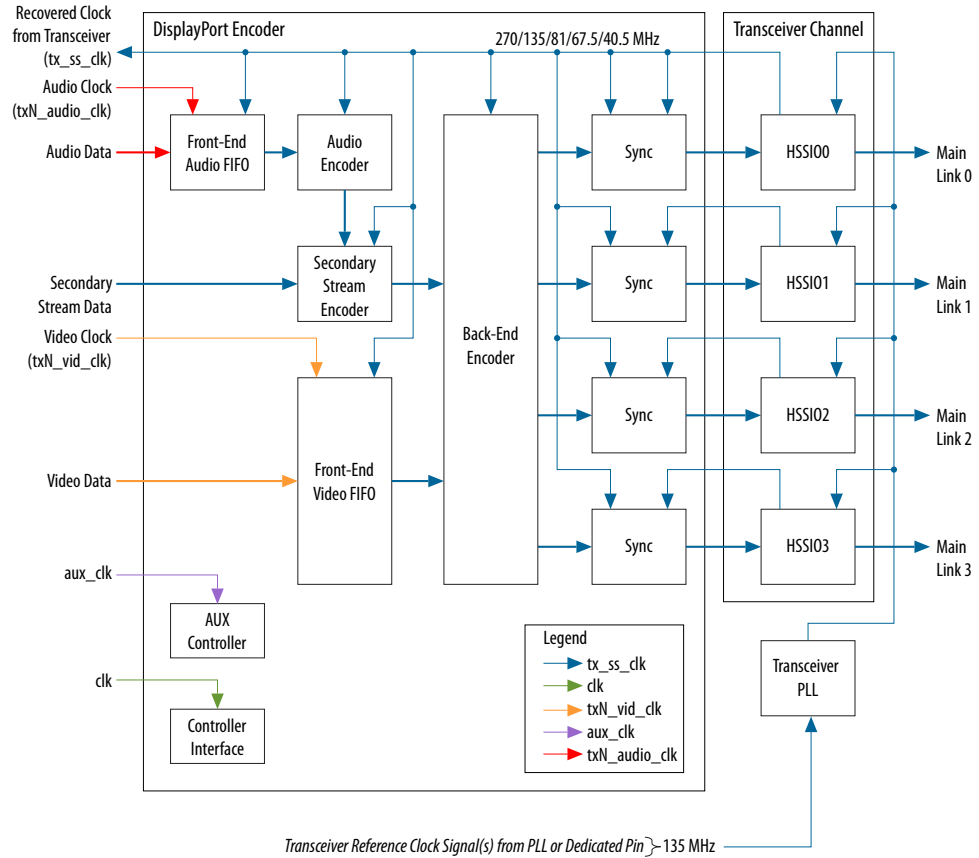
Use the audio channel status to provide any information about the audio stream needed by downstream devices.

## 5.6. Source Clock Tree

The source uses the following clocks:

- Local pixel clock (`txN_vid_clk`), which clocks video data into the IP core.
- Main link clock (`tx_ss_clk`), 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 the 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, this clock is equal to the link rate divided by 20 (270, 135, or 81 MHz). If you select the quad symbol mode, this clock is equal to the link rate divided by 40 (202.5, 135, 67.5, or 40.5 MHz). The core supports only asynchronous local pixel clock and main link clock.
- 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.
- `txN_audio_clk` for the audio interface.

Figure 22. Source Clock Tree



## 6. DisplayPort Sink

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.

**Figure 23. DisplayPort Sink Top-Level Block Diagram**

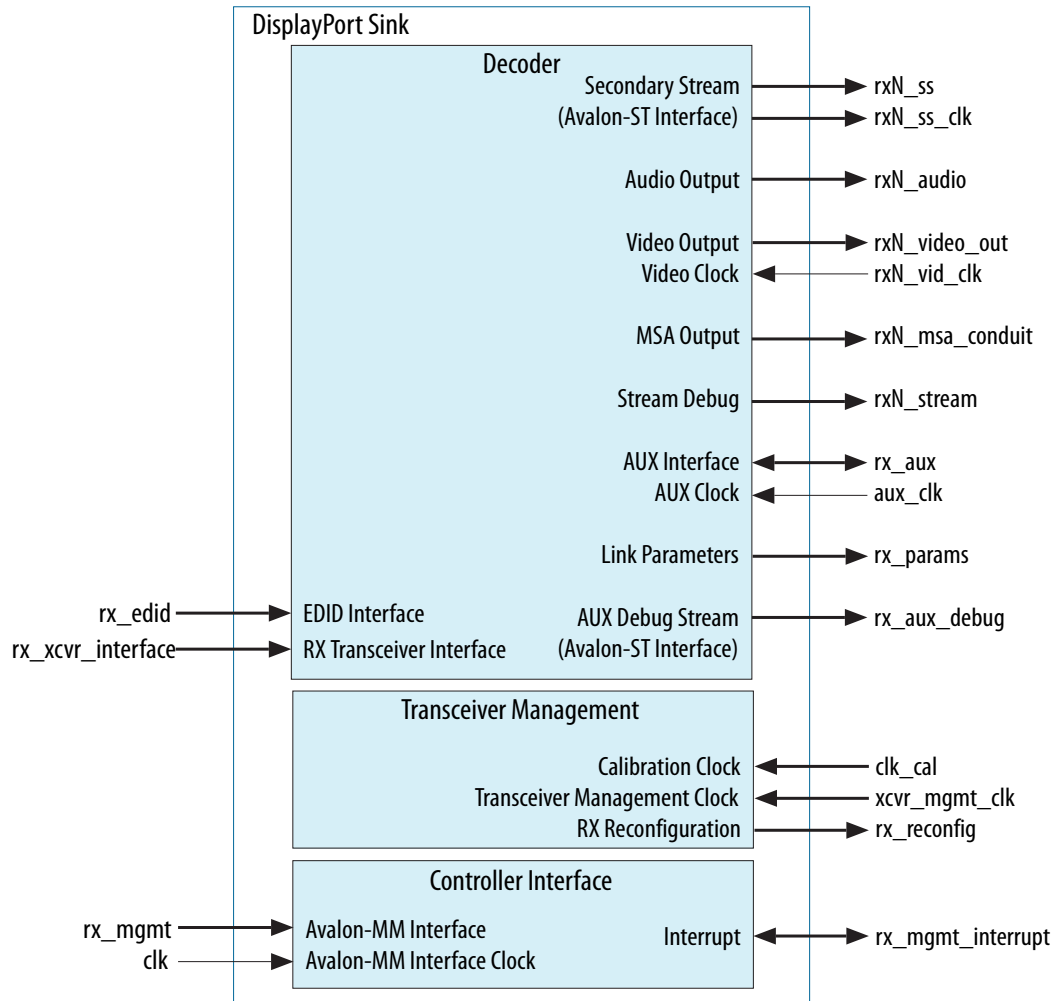
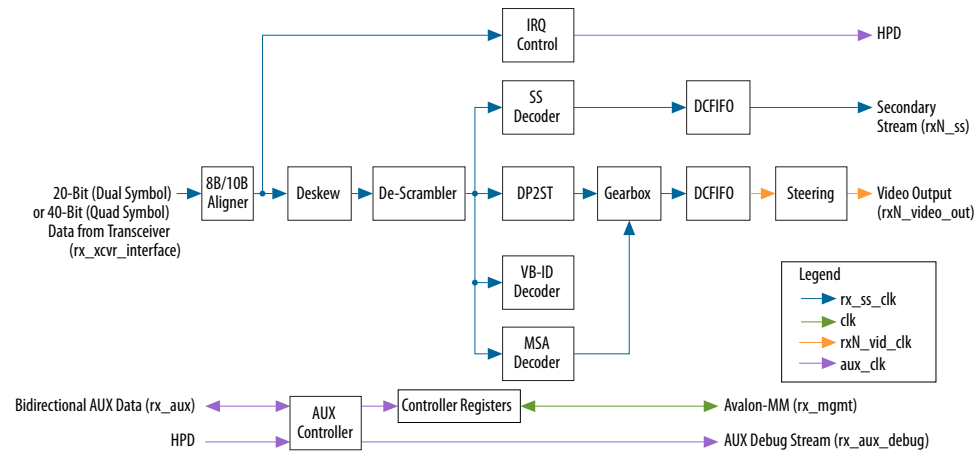


Figure 24. DisplayPort Sink Functional Block Diagram



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_ss_clk` domain through a DCFIFO.
  - b. The main data path extracts all pixel data from the incoming stream. Then, the gearbox block resamples the pixel data into the current bit-per-pixel data width. Next, the IP core crosses the pixel data into the `rxN_vid_clk` domain through 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 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 real time.



## 6.1. Sink Embedded DisplayPort (eDP) Support

The DisplayPort Intel FPGA 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 sink supports:

- Full (normal) link training—default
- Fast link training—mandatory eDP feature
- Black video—mandatory eDP feature

## 6.2. Sink Interfaces

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

**Table 26. 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	Active-high reset signal for embedded controller
rx_mgmt	AV-MM	clk	rx_mgmt_address[8:0]	Input	32-bit word addressing address
			rx_mgmt_chipselect	Input	Must be asserted for valid read or write access
			rx_mgmt_read	Input	Must be asserted to indicate a read transfer
			rx_mgmt_write	Input	Must be asserted to indicate a write transfer
			rx_mgmt_writedata[31:0]	Input	Data for write transfers
			rx_mgmt_readdata[31:0]	Output	Data for read transfers
			rx_mgmt_waitrequest	Output	Asserted when the DisplayPort Intel FPGA IP core is unable to respond to a read or write request. Forces the GPU to wait until the IP core is ready to proceed with the transfer.
rx_mgmt_irq	IRQ	clk	rx_mgmt_irq	Output	The IP core asserts this signal to issue an interrupt to the GPU

[Controller Interface](#) on page 74

**Table 27. Transceiver Management Interface**

n is the number of RX lanes.

Interface	Port Type	Clock Domain	Port	Direction	Description
xcvr_mgmt_clk	Clock	N/A	xcvr_mgmt_clk	Input	Transceiver management clock
clk_cal	Clock	N/A	clk_cal	Input	Calibration clock
rx_reconfig	Conduit	xcvr_mgmt_clk	rx_link_rate[1:0]	Output	Transceiver link rate reconfiguration handshaking
			rx_link_rate_8bits[7:0]	Output	
			rx_reconfig_req	Output	
			rx_reconfig_ack	Input	
			rx_reconfig_busy	Input	
rx_analog_reconfig	Conduit	xcvr_mgmt_clk	rx_vod [2n-1:0]	Output	Transceiver analog reconfiguration handshaking
			rx_emp [2n-1:0]	Output	
			rx_analog_reconfig_req	Output	

**Note:** Value of rx\_link\_rate[1:0]: 0 = 1.62 Gbps, 1 = 2.70 Gbps, 2 = 5.40 Gbps, 3 = 8.10 Gbps; value of rx\_link\_rate\_8bits[7:0]: 0x06 = 1.62 Gbps, 0x0a = 2.70 Gbps, 0x14 = 5.40Gbps, 0x1e = 8.10 Gbps

[Transceiver Reconfiguration Interface on page 80](#)

**Table 28. Video Interface**

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

Interface	Port Type	Clock Domain	Port	Direction	Description
rxN_vid_clk	Clock	N/A	rxN_vid_clk	Input	Video clock
rxN_video_out	Conduit	rx_vid_clk	rxN_vid_valid[p-1:0]	Output	Each bit is asserted when all other signals (except rxN_vid_overflow) on this port are valid and the corresponding pixel belongs to active video. Width configurable
			rxN_vid_sol	Output	Start of video line
			rxN_vid_eol	Output	End of video line
			rxN_vid_sof	Output	Start of video frame
			rxN_vid_eof	Output	End of video frame
			rxN_vid_locked	Output	1 = Video locked 0 = Video unlocked
			rxN_vid_overflow	Output	1 = Video data overflow detected 0 = No overflow detected This signal is always valid.

*continued...*



Interface	Port Type	Clock Domain	Port	Direction	Description
			rxN_vid_interlace	Output	1 = Interlaced 0 = Progressive
			rxN_vid_field	Output	1 = Top field 0 = Bottom field (or progressive)
			rxN_vid_data[3v*p-1:0]	Output	Width configurable

Video Interface on page 76

**Table 29. 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	Active-high AUX channel reset
rx_aux	Conduit	aux_clk	rx_aux_in	Input	AUX channel data input
			rx_aux_out	Output	AUX channel data output
			rx_aux_oe	Output	Output buffer enable
			rx_hpd	Output	Hot plug detect
			rx_cable_detect	Input	Upstream cable detect
			rx_pwr_detect	Input	Upstream power detect
rx_aux_debug	AV-ST	aux_clk	rx_aux_debug_data[31:0]	Output	Formatted AUX channel debug data
			rx_aux_debug_valid	Output	Asserted when all the other signals on this port are valid
			rx_aux_debug_sop	Output	Start of packet (start of AUX request or reply)
			rx_aux_debug_eop	Output	End of packet (end of AUX request or reply)
			rx_aux_debug_err	Output	Indicates if the IP core detects an error in the current byte
			rx_aux_debug_cha	Output	The channel number for data being transferred on the current cycle. Used as AUX channel data direction. 1 = Reply (to DisplayPort source) 0 = Request (from DisplayPort source)
EDID (rx_edid)	AV-MM	aux_clk	rx_edid_address[7:0]	Output	8-bit byte addressing address
			rx_edid_read	Output	Asserted to indicate a read transfer
			rx_edid_write	Output	Asserted to indicate a write transfer

*continued...*



Interface	Port Type	Clock Domain	Port	Direction	Description
			rx_edid_writedata[7:0]	Output	Data for write transfers
			rx_edid_readdata[7:0]	Input	Data for read transfers
			rx_edid_waitrequest	Input	Must be asserted when the Slave is unable to respond to a read or write request. Forces the DisplayPort Intel FPGA IP core to wait until the Slave is ready to proceed with the transfer.

[AUX Interface](#) on page 74

**Table 30. Debugging Interface**

s is the number of symbols per clock and N is the stream number.

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 (rxN_stream)	Conduit	rx_ss_clk	rxN_stream_data[4*8*s-1:0]	Output	Post scrambler data
			rxN_stream_ctrl[4*s-1:0]	Output	Post scrambler comma marker. The IP core asserts each bit when the relative 8-bit byte is a comma code, and deasserts each bit when the byte is a data value. Bit 0 qualifies rxN_stream_data[7:0] byte, bit 1 qualifies the rxN_stream_data[15:8] byte, and so on.
			rxN_stream_valid	Output	Asserted for one clock cycle when rxN_stream_data[63:0] and rxN_stream_ctrl[7:0] are valid
			rxN_stream_clk	Output	Port clock

[Debugging Interface](#) on page 75

**Table 31. Secondary Interface**

N is the stream number; for example, rx\_msa\_conduit represents Stream 0, rx1\_msa\_conduit represents Stream 1, and so on .

Interface	Signal Type	Clock Domain	Port	Direction	Description
rx_ss_clk	Clock	N/A	rx_ss_clk	Output	Clock
MSA (rxN_msa_conduit)	Conduit	rx_ss_clk	rxN_msa[216:0]	Output	Output for current MSA parameters received from the source
Secondary Stream (rxN_ss)	AV-ST	rx_ss_clk	rxN_ss_data[159:0]	Output	Secondary stream interface

*continued...*





Interface	Signal Type	Clock Domain	Port	Direction	Description
			rxN_ss_valid	Output	
			rxN_ss_sop	Output	
			rxN_ss_eop	Output	

Secondary Stream Interface on page 81

**Table 32. Audio Interface**

m is the number of RX audio channels. N is the stream number; for example, rx\_audio represents Stream 0, rx1\_audio represents Stream 1, and so on .

Interface	Signal Type	Clock Domain	Port	Direction	Description
Audio (rxN_audio)	Conduit	rx_ss_clk	rxN_audio_lpcm_data[m*32-1:0]	Output	N channels of 32-bit audio sample data.
			rxN_audio_valid	Output	Asserted when valid data is available on rxN_audio_lpcm_data
			rxN_audio_mute	Output	Asserted when audio is muted
			rxN_audio_infoframe[39:0]	Output	40-bit bundle containing the Audio InfoFrame packet

Audio Interface on page 83

**Table 33. RX Transceiver Interface**

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

*Note:* Connect the DisplayPort signals to the Native PHY signals of the same name.

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 Equivalent to Link Speed Clock (ls_clk).
	Conduit	rx_xcvr_clkout	rx_parallel_data[n*s*10-1:0]	Input	Parallel data from RX transceiver
	Conduit	rx_xcvr_clkout	rx_restart	Output	Use to reset the RX PHY Reset Controller when the RX data loses alignment <i>Note:</i> Required for Intel Arria 10, Intel Cyclone 10 GX, and Intel Stratix 10 devices. Not used in Arria V, Cyclone V, and Stratix V devices.
	Conduit	N/A	rx_is_lockedtoref[n-1:0]	Input	When asserted, indicates that the RX CDR PLL is locked to the reference clock

**continued...**

Interface	Port Type	Clock Domain	Port	Direction	Description
	Conduit	N/A	rx_is_lockedto[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 <i>Note:</i> Required only for Arria V, Cyclone V, and Stratix V devices.
	Conduit	xcvr_mgmt_clk	rx_digitalreset[n-1:0]	Output	When asserted, resets the RX PCS <i>Note:</i> Required only for Arria V, Cyclone V, and Stratix V devices.
	Conduit	xcvr_mgmt_clk	rx_set_lockto[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_lockto[n-1:0]	Output	Forces the RX CDR circuitry to lock to the received data

[RX Transceiver Interface on page 79](#)

### 6.2.1. 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 166](#)

### 6.2.2. 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.

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.

The sink core uses the `rx_cable_detect` signal to detect when a source (upstream) device is physically connected and the `rx_pwr_detect` signal to detect when a source device is powered. The sink core keeps the `rx_hpd` signal deasserted if both the `rx_cable_detect` and `rx_pwr_detect` signals are not asserted.

### 6.2.2.1. 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.

### 6.2.2.2. 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.

### 6.2.3. Debugging Interface

#### 6.2.3.1. Link Parameters Interface

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

#### 6.2.3.2. 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 table, 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 34. rxN\_stream\_data Bit Allocation**

Bit	Dual-Symbol Mode	Quad-Symbol Mode
127:120	Not applicable	Lane 3 symbol $n + 3$
119:112	Not applicable	Lane 3 symbol $n + 2$
111:104	Not applicable	Lane 3 symbol $n + 1$
103:96	Not applicable	Lane 3 symbol $n$
95:88	Not applicable	Lane 2 symbol $n + 3$
<i>continued...</i>		

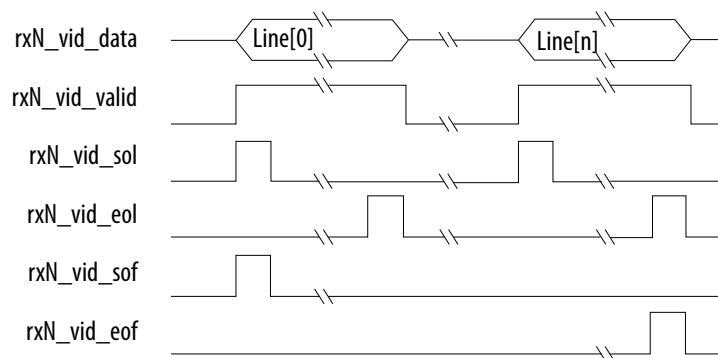
Bit	Dual-Symbol Mode	Quad-Symbol Mode
87:80	Not applicable	Lane 2 symbol $n + 2$
79:72	Not applicable	Lane 2 symbol $n + 1$
71:64	Not applicable	Lane 2 symbol $n$
63:56	Lane 3 symbol $n + 1$	Lane 1 symbol $n + 3$
55:48	Lane 3 symbol $n$	Lane 1 symbol $n + 2$
47:40	Lane 2 symbol $n + 1$	Lane 1 symbol $n + 1$
39:32	Lane 2 symbol $n$	Lane 1 symbol $n$
31:24	Lane 1 symbol $n + 1$	Lane 0 symbol $n + 3$
23:16	Lane 1 symbol $n$	Lane 0 symbol $n + 2$
15:8	Lane 0 symbol $n + 1$	Lane 0 symbol $n + 1$
7:0	Lane 0 symbol $n$	Lane 0 symbol $n$

When data is received, data is produced 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 `rxN_stream_valid` signal remains asserted.

### 6.2.4. Video Interface

This interface (`rxN_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 25. Video Out Image Port Timing Diagram**



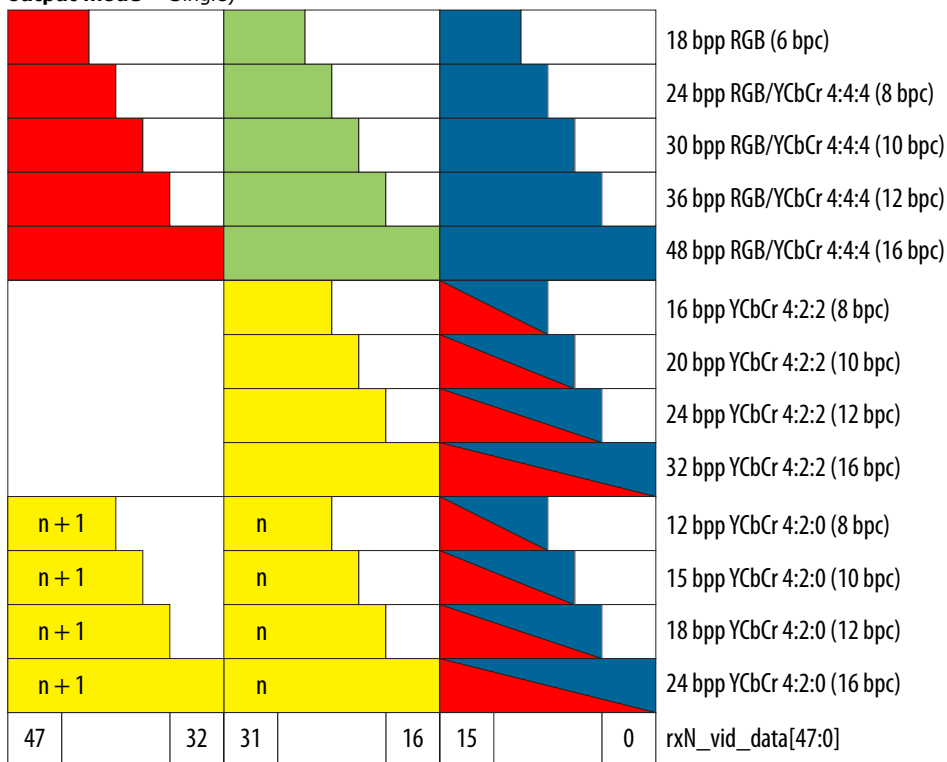
The `rxN_vid_overflow` signal is always valid, regardless of the logical state of `rxN_vid_valid`. `rxN_vid_overflow` is asserted for at least one clock cycle when the sink core internal video data FIFO runs into an overflow condition. This condition can occur when the `rxN_vid_clk` frequency is too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



Specify the maximum data color depth in the DisplayPort parameter editor. The same output port transfers both RGB and YCbCr data in 4:4:4, 4:2:2, or 4:2:0 color format. Data is most-significant bit aligned and formatted for 4:4:4.

**Figure 26. Video Output Data Format**

18 bpp to 48 bpp for RGB/YCbCr 4:4:4, 16 bpp to 32 bpp for YCbCr 4:2:2, and 12 bpp to 24 bpp for YCbCr 4:2:0 port width when `rxN_video_out` port width is 48 (**Maximum video output color depth** = 16 bpc, **Pixel output mode** = Single)



n = Pixel Index

**Table 35. Video Ports for 4:2:2 and 4:2:0 Color Formats**

Color Format	Description
Sub-sampled 4:2:2 color format	<ul style="list-style-type: none"> <li>Video port bits 47:32 are unused</li> <li>Video port bits 31:16 always transfer the Y component</li> <li>Video port bits 15:0 always transfer the alternate Cb or Cr component</li> </ul>
Sub-sampled 4:2:0 color format	<ul style="list-style-type: none"> <li>For even lines (starting with line 0)                             <ul style="list-style-type: none"> <li>Video port bits 47:32 always transfer the <math>Y_{n+1}</math> component.</li> <li>Video port bits 31:16 always transfer the <math>Y_n</math> component.</li> <li>Video port bits 15:0 always transfer the <math>Cb_n</math> component.</li> </ul> </li> <li>For odd lines                             <ul style="list-style-type: none"> <li>Video port bits 47:32 always transfer the <math>Y_{n+1}</math> component.</li> <li>Video port bits 31:16 always transfer the <math>Y_n</math> component.</li> <li>Video port bits 15:0 always transfer the <math>Cr_n</math> component.</li> </ul> </li> </ul>

**Table 36. YCbCr 4:2:0 Input Data Ordering Compared to RGB 4:4:4**

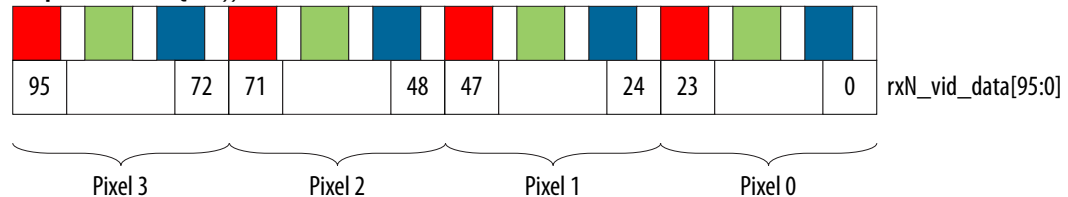
Pixel Indexes	R Position	G Position	B Position
0 and 1	Y1	Y0	<ul style="list-style-type: none"> <li>• Cb0 (Even lines)</li> <li>• Cr0 (Odd lines)</li> </ul>
2 and 3	Y3	Y2	<ul style="list-style-type: none"> <li>• Cb2 (Even lines)</li> <li>• Cr2 (Odd lines)</li> </ul>
4 and 5	Y5	Y4	<ul style="list-style-type: none"> <li>• Cb4 (Even lines)</li> <li>• Cr4 (Odd lines)</li> </ul>
...	...	...	...

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

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

**Figure 27. Video Output Alignment**

For RGB 18 bpp when `rxN_video_out` port width is 96 (**Maximum video output color depth = 8 bpc, Pixel output mode = Quad**)



**Related Information**

[Video and Image Processing Suite User Guide](#)

Provides more information about Clocked Video Input.

**6.2.5. Clocked Video Input Interface**

The `rxN_video_out` interface may 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 `rxN_video_out` interface has the following signals: `rxN_vid_valid`, `rxN_vid_sol`, `rxN_vid_eol`, `rxN_vid_sof`, `rxN_vid_eof`, `rxN_vid_locked`, and `rxN_vid_data`.

The following table describes how to connect the CVI and DisplayPort sink signals.

*Note:* This example uses the Intel Clocked Video Input IP core.



**Table 37. Connecting CVI Signals to DisplayPort Sink Stream 0 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	rx_vid_field	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 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_h_sync = rx_vid_h_sync;

assign vid_de = rx_vid_valid;

assign vid_v_sync = rx_vid_v_sync;
```

**6.2.6. RX Transceiver Interface**

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

## 6.2.7. Transceiver Reconfiguration Interface

You can reconfigure the transceiver to accept a single reference clock of 135 MHz for all bit rates: RBR, HBR, HBR2, and HBR3.

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

- [AN 645: Dynamic Reconfiguration of PMA Controls in Stratix V Devices](#)  
Provides more information about using the Transceiver Reconfiguration Controller to reconfigure the Stratix V Physical Media Attachment (PMA) controls dynamically.
- [V-Series Transceiver PHY IP Core User Guide](#)  
Provides more information about how to reconfigure the transceiver for 28-nm devices.
- [AN 676: Using the Transceiver Reconfiguration Controller for Dynamic Reconfiguration in Arria V and Cyclone V Devices](#)  
Provides 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](#)  
Provides more information about link tuning.
- [Intel Arria 10 Transceiver PHY User Guide](#)  
Provides more information about how to reconfigure the transceiver for Intel Arria 10 devices.
- [Intel Cyclone 10 GX Transceiver PHY User Guide](#)  
Provides more information about how to reconfigure the transceiver for Intel Cyclone 10 GX devices.
- [Intel Stratix 10 L- and H-tile Transceiver PHY User Guide](#)  
Provides more information about how to reconfigure the transceiver for Intel Stratix 10 L-tile and H-tile devices.





## 6.2.8. Secondary Stream Interface

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

*Note:* The DisplayPort Intel FPGA IP core supports InfoFrame SDP versions 1.2 and 1.3 over the Main-Link. INFOFRAME SDP version 1.2 shall be used to convey Audio INFOFRAME control information, as specified in *CEA-861-F* and *CEA-861.2*. Other INFOFRAME coding types, as specified in *CEA-861-F, Table 5*, and *CEA-861.3*, shall use INFOFRAME SDP version 1.3. Refer to the *VESA DisplayPort Standard version 1.2a, Section 2.2.5.1* for detailed definition.

The format of the `rxN_ss` interface output corresponds to four 15-nibble code words as specified by the *VESA DisplayPort Standard version 1.2a, Section 2.2.6.3*. These 15-nibble code words are typically supplied to the downstream Reed-Solomon decoder. The format differs for both header and payload, as shown in the following figure.

**Figure 28. rxN\_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

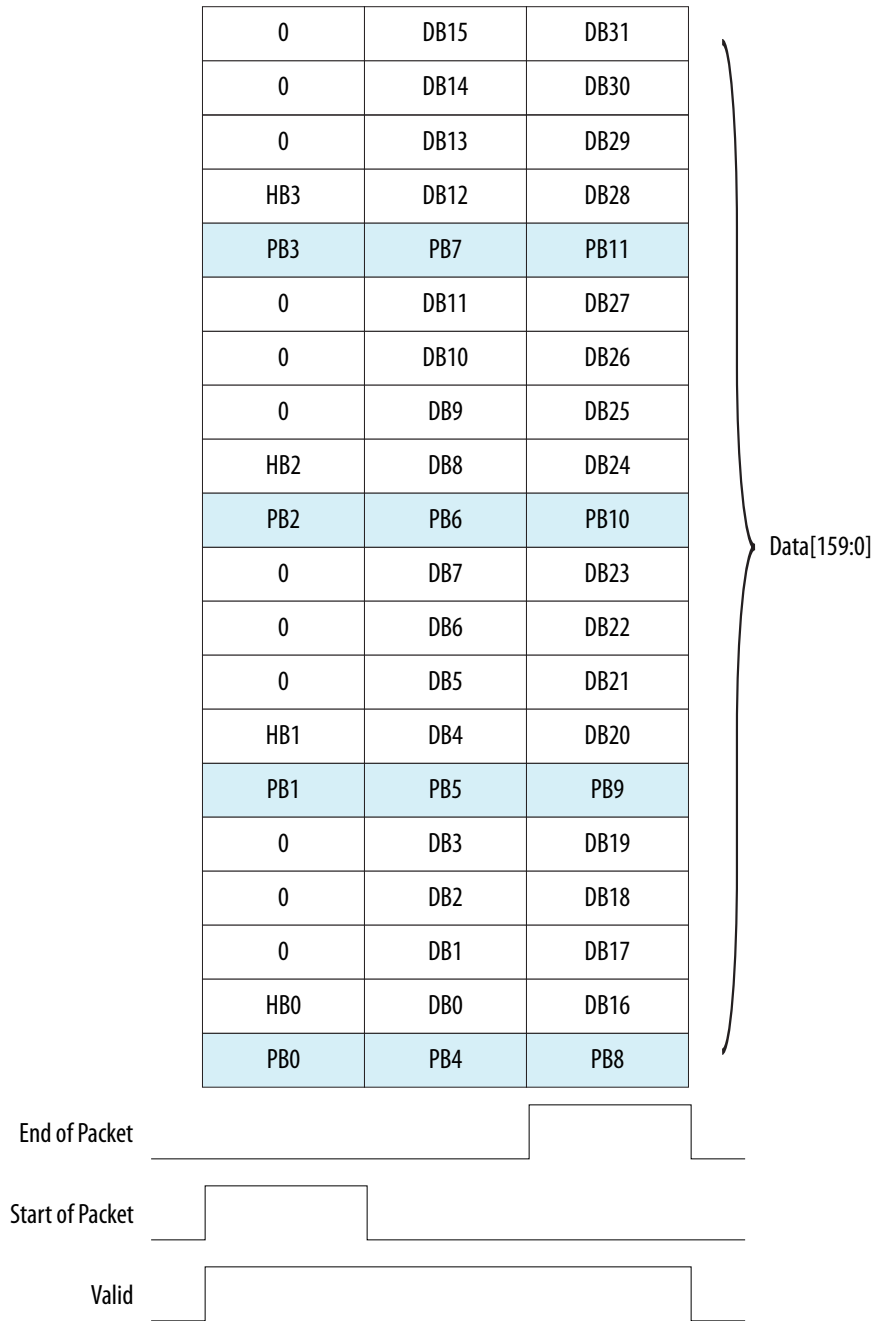


The following figure shows a typical secondary stream packet with the four byte header (HB0, HB1, HB2, and HB3) and 32-byte payload (DB0, ..., DB31). Each symbol has an associated parity nibble (PB0, ..., PB11). Downstream logic uses start-of-packet and end-of-packet to determine if the current input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

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



Figure 29. Typical Secondary Stream Packet



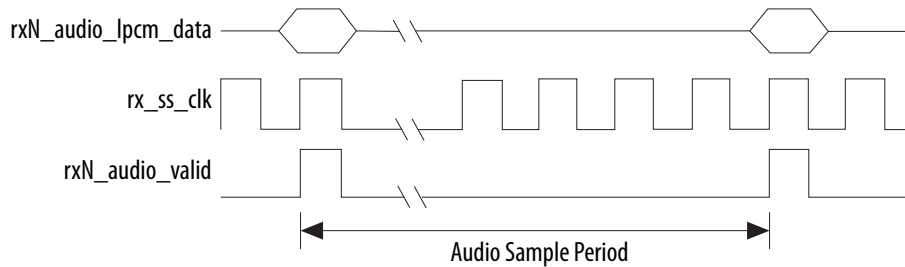
### 6.2.9. Audio Interface

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 `rxN_audio` port uses the values to generate the `rxN_audio_valid` signal according to sample audio data. Data is clocked out using the `rx_ss_clk` signal. The `rx_ss_clk` 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 `rxN_audio_valid` signal using the  $M$  and  $N$  values, and asserts it at the current audio sample clock rate. The `rxN_audio_mute` signal indicates whether audio data is present on the DisplayPort interface.

**Figure 30. rxN\_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.

### 6.2.10. MSA Interface

The `rxN_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 Vertical Blank ID (VB-ID) packets, respectively.

The sink asserts bit `msa_valid` when all `msa_` signals are valid and deasserts during MSA update. The sink assigns the MSA parameters to zero when it is not receiving valid video data.

The sink asserts the `msa_lock` bit 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; for example, when the source changes video resolution. You can use its deasserted state to invalidate received video data.

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

**Table 38. rxN\_msa\_conduit Port Signals**

Bit	Signal	Description
216	<code>msa_lock</code>	0 = MSA fields format error. 1 = MSA fields correctly formatted.
215	<code>vbid_strobe</code>	0 = VB-ID fields invalid, 1 = VB-ID fields valid.
<i>continued...</i>		



Bit	Signal	Description
214:209	vbid_vbid[5:0]	VB-ID bit field: <ul style="list-style-type: none"> <li>vbid[0] - VerticalBlanking_Flag</li> <li>vbid[1] - FieldID_Flag (for progressive video, this remains 0)</li> <li>vbid[2] - Interlace_Flag</li> <li>vbid[3] - NoVideoStream_Flag</li> <li>vbid[4] - AudioMute_Flag</li> <li>vbid[5] - HDCP SYNC DETECT</li> </ul>
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
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]	<p>The MISC0[7:1] and MISC1[7] fields indicate the color encoding format. The color depth is indicated in MISC0[7:5]:</p> <ul style="list-style-type: none"> <li>000 - 6 bpc</li> <li>001 - 8 bpc</li> <li>010 - 10 bpc</li> <li>011 - 12 bpc</li> <li>100 - 16 bpc</li> </ul> <p>For details about the encoding format, refer to the <i>VESA DisplayPort Standard version 1.4</i>.</p>
7:0	msa_MISC1[7:0]	

### 6.3. Sink Clock Tree

The IP core receives DisplayPort serial data across the high-speed serial interface (HSSI). The HSSI requires a 135 MHz clock for correct data locking. You can supply this frequency to the HSSI using a reference clock provided by an Intel FPGA PLL or pins.

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 (202.5, 135, 67.5, or 40.5 MHz).

The IP core crosses the reconstructed pixel data into a local video clock (`rxN_vid_clk`) through an output DCFIFO, which drives the pixel stream output. The `rxN_vid_clk` frequency must be higher than or equal to the video clock in the up-stream source.

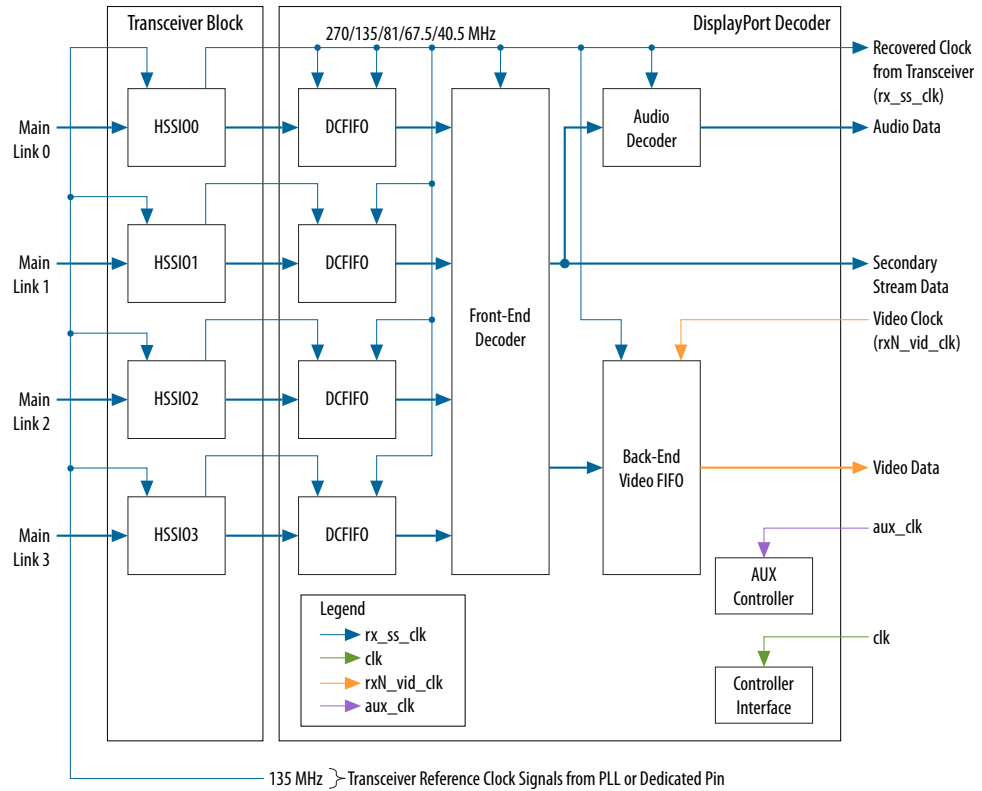
- If `rxN_vid_clk` is slower than the up-stream video clock, the DCFIFO overflows.
- If the `rxN_vid_clk` is faster than the up-stream source video clock, the output port experiences a deassertion of the valid port on cycles in which pixel data is not available.

The optimum frequency is the exact clock rate in the up-stream source. You require pixel clock recovery techniques to determine this clock frequency.

Secondary stream data is clocked by `rx_ss_clk`. 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 31. Sink Clock Tree



### Related Information

[Clock Recovery Core](#) on page 27

Provides more information about determining the optimum frequency.

## 7. DisplayPort Intel FPGA IP Parameters

Use the settings in the DisplayPort Intel FPGA IP parameter editor to configure your design.

**Note:** For DisplayPort Intel FPGA IP design example parameters, refer to the DisplayPort Intel FPGA IP design example user guide for the respective devices.

### Related Information

- [DisplayPort Intel Arria 10 FPGA IP Design Example User Guide](#)  
For more information about the Intel Arria 10 design example.
- [DisplayPort Intel Cyclone 10 GX FPGA IP Design Example User Guide](#)  
For more information about the Intel Cyclone 10 GX design example.
- [DisplayPort Intel Stratix 10 FPGA IP Design Example User Guide](#)  
For more information about the Intel Stratix 10 design examples.
- [DisplayPort Intel FPGA IP Hardware Design Examples for Arria V, Cyclone V, and Stratix V Devices](#) on page 23

### 7.1. DisplayPort Intel FPGA IP Source Parameters

You set parameters for the source using the DisplayPort Intel FPGA IP parameter editor.

**Table 39. Source Parameters**

Parameter	Description
<b>Device family</b>	Select the targeted device family: Intel Stratix 10, Intel Arria 10, Intel Cyclone 10 GX, Arria V GX, Arria V GZ, Cyclone V, or Stratix V.
<b>Support DisplayPort source</b>	Turn on to enable DisplayPort source.
<b>Maximum video input color depth</b>	Determines the maximum video input color depth (bits per color) supported by the DisplayPort source. Select 6, 8, 10, 12 or 16 bpc. <i>Note:</i> DisplayPort source supports RGB, YCbCr 4:4:4, YCbCr 4:2:2, and YCbCr 4:2:0 video formats by default.
<b>TX maximum link rate</b>	Select the maximum link rate supported: 8.1 Gbps, 5.4 Gbps, 2.7 Gbps, or 1.62 Gbps. <i>Note:</i> Cyclone V devices only support up to 2.7 Gbps. 8.10 Gbps is only available in quad symbols per clock for Intel Arria 10, Intel Cyclone 10 GX, and Intel Stratix 10 devices.
<b>Maximum lane count</b>	Select the maximum lanes supported: 1, 2, or 4. <i>Note:</i> If you turn on the <b>Support MST</b> parameter, the maximum lane count is fixed to 4 lanes.
<b>Symbol output mode</b>	Determines the TX transceiver data width in symbols per clock. Select dual (20 bits) or quad (40 bits).
<i>continued...</i>	

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

\*Other names and brands may be claimed as the property of others.





Parameter	Description
	Dual symbol mode saves logic resource but requires the core to run at twice the clock frequency of quad symbol mode. If timing closure is a problem in the device, you should consider using quad symbol mode.
<b>Pixel input mode</b>	Select the number of pixels per clock (single, dual, or quad symbol). <ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul>
<b>Scrambler seed value</b>	Select the initial seed value for the scrambler block. <ul style="list-style-type: none"> <li>DP: 16'hFFFF</li> <li>eDP: 16'hFFFE</li> </ul>
<b>Enable Video input Image port</b>	Turn on to enable the video image interface. Turn off to use the traditional HSYNC/VSYNC/DE video input interface.
<b>Support analog reconfiguration</b>	Turn on to reconfigure VOD and pre-emphasis values.
<b>Enable AUX debug stream</b>	Turn on to send source AUX traffic output to an Avalon-ST port.
<b>Support CTS test automation</b>	Turn on to support CTS test automation.
<b>Support GTC</b>	The Global Time Code (GTC) feature is not available. However, if you want to use this feature, contact your nearest Intel FPGA sales representative or file a Service Request.
<b>Support secondary data channel</b>	Turn on to enable secondary data.
<b>Support audio data channel</b>	Turn on to enable audio packet encoding. <i>Note:</i> To use this parameter, you must turn on the <b>Support secondary data channel</b> parameter.
<b>Number of audio data channels</b>	Select the number of audio channels (2 or 8).
<b>Support MST</b>	Turn on to enable multi-stream support. <i>Note:</i> For multi-stream support, the maximum lane count is fixed to four lanes.
<b>Max stream count</b>	Specify the maximum amount of streams supported: 2, 3, or 4. <i>Note:</i> To use this parameter, you must turn on the <b>Support MST</b> parameter.

## 7.2. DisplayPort Intel FPGA IP Sink Parameters

You set parameters for the sink using the DisplayPort Intel FPGA IP parameter editor.

**Table 40. Sink Parameters**

Parameter	Description
<b>Device family</b>	Select the targeted device family: Intel Stratix 10, Intel Arria 10, Intel Cyclone 10 GX, Arria V GX, Arria V GZ, Cyclone V, or Stratix V.
<b>Support DisplayPort sink</b>	Turn on to enable DisplayPort sink.
<b>Maximum video output color depth</b>	Determines the maximum video input color depth (bits per color) supported by the DisplayPort source. Select 6, 8, 10, 12 or 16 bpc. DisplayPort sink supports RGB, YCbCr 4:4:4, YCbCr 4:2:2, and YCbCr 4:2:0 video formats by default. <i>Note:</i> If you turn on the <b>Support MST</b> parameter, the maximum video input color depth is limited to 8 bpc.

*continued...*



Parameter	Description
<b>RX maximum link rate</b>	Select the maximum link rate supported: 8.1 Gbps, 5.4 Gbps, 2.7 Gbps, or 1.62 Gbps. <i>Note:</i> Cyclone V devices only support up to 2.7 Gbps. 8.10 Gbps is only available in quad symbols per clock for Intel Arria 10, Intel Cyclone 10 GX, and Intel Stratix 10 devices.
<b>Maximum lane count</b>	Select the maximum lanes supported: 1, 2, or 4. <i>Note:</i> If you turn on the <b>Support MST</b> parameter, the maximum lane count is fixed to 4 lanes.
<b>Symbol input mode</b>	Determines the RX transceiver data width in symbols per clock. Select dual (20 bits) or quad (40 bits). Dual symbol mode saves logic resource but requires the core to run at twice the clock frequency of quad symbol mode. If timing closure is a problem in the device, you should consider using quad symbol mode.
<b>Pixel output mode</b>	Select the number of pixels per clock (single, dual, or quad symbol). <ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul>
<b>Sink scrambler seed value</b>	Select the initial seed value for the scrambler block. <ul style="list-style-type: none"> <li>DP: 16'hFFFF</li> <li>eDP: 16'hFFFE</li> </ul>
<b>Export MSA</b>	Turn on to enable the sink to export the MSA interface to the top-level port interface.
<b>IEEE OUI</b>	Specify an IEEE organizationally unique identifier (OUI) as part of the DPCD registers.
<b>Enable GPU control</b>	Turn on to use an embedded controller to control the sink.
<b>Enable AUX debug stream</b>	Turn on to enable AUX traffic output to an Avalon-ST port.
<b>Support CTS test automation</b>	Turn on to support automated test features.
<b>Support GTC</b>	The Global Time Code (GTC) feature is not available. However, if you want to use this feature, contact your nearest Intel FPGA sales representative or file a Service Request.
<b>Support secondary data channel</b>	Turn on to enable secondary data.
<b>Support audio data channel</b>	Turn on to enable audio packet decoding. <i>Note:</i> To use this parameter, you must also turn on <b>Support secondary data channel</b> . <i>Note:</i> The IP core does not support audio data channel if you turn on the <b>Support MST</b> parameter.
<b>Number of audio data channels</b>	Select the number of audio channels (2 or 8).
<b>Support MST</b>	Turn on to enable multi-stream support. You must turn on <b>Enable GPU control</b> to support MST mode. <i>Note:</i> For multi-stream support, the maximum lane count is fixed to four lanes.
<b>Max stream count</b>	Specify the maximum amount of streams supported: 2, 3, or 4. <i>Note:</i> To use this parameter, you must turn on the <b>Support MST</b> parameter.

## 8. DisplayPort Intel FPGA IP Core Simulation Example

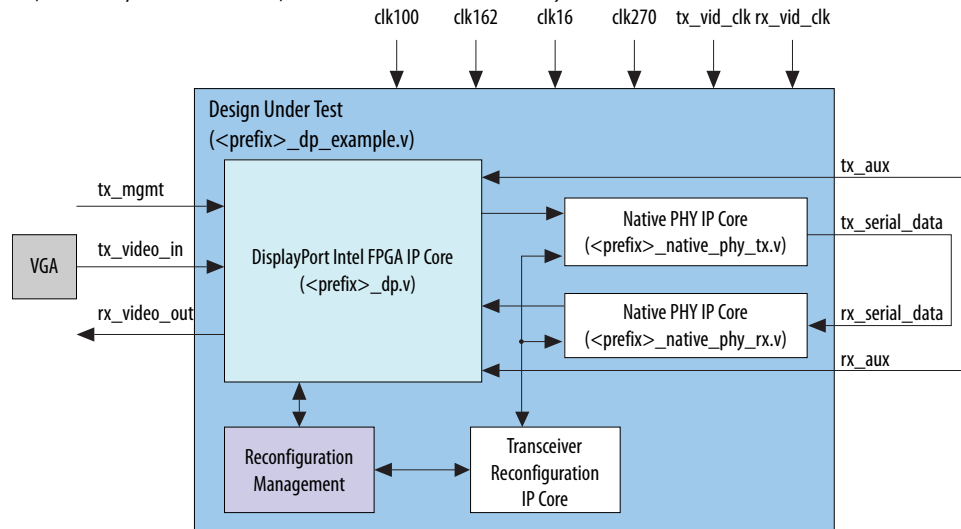
The DisplayPort simulation example allows you to evaluate the functionality of the DisplayPort Intel FPGA IP core and provides a starting point for you to create your own simulation. This example targets the ModelSim SE simulator.

The simulation example instantiates the DisplayPort Intel FPGA IP core with default settings, TX and RX enabled, and 8 bits per color. The core has the **Support CTS test automation** parameter 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 32. Simulation Example Block Diagram for Arria V, Cyclone V, and Stratix V Devices**

The 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).



### 8.1. 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.

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

\*Other names and brands may be claimed as the property of others.



You use a script to automate these steps.

### 8.1.1. 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 41. Simulation Example Files for Arria V, Cyclone V, and Stratix V Devices**

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).
IP Catalog files	<prefix>_dp.v	IP Catalog variant for the DisplayPort Intel FPGA 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 Intel FPGA IP core mapped to the CVI input.
<i>continued...</i>		



File Type	File	Description
	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.

### 8.1.2. 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:

- Generate the simulation files for the DisplayPort, transceivers, and transceiver reconfiguration IP cores:

Arria V, Cyclone V, and Stratix V devices; (where *<prefix>* is *av* for Arria V devices, *cv* for Cyclone V devices, and *sv* for Stratix V devices)

```

- 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**:

Arria V, Cyclone V, and Stratix V devices; (where *<prefix>* is *av* for Arria V devices, *cv* for Cyclone V devices, and *sv* for Stratix V devices)

```

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 2. 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
```

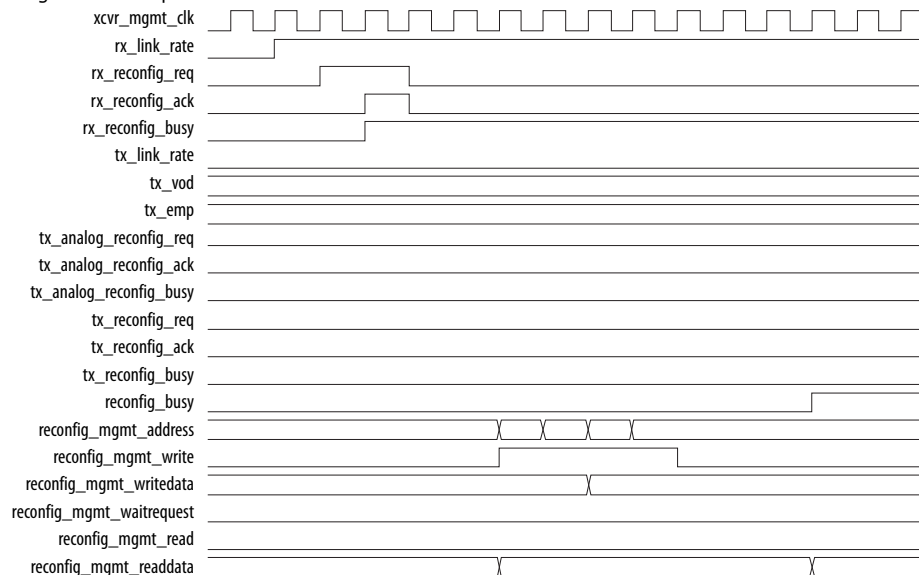
### 8.1.3. View the Results

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 7-1 for a listing of the files).

#### Figure 33. RX Reconfiguration Waveform

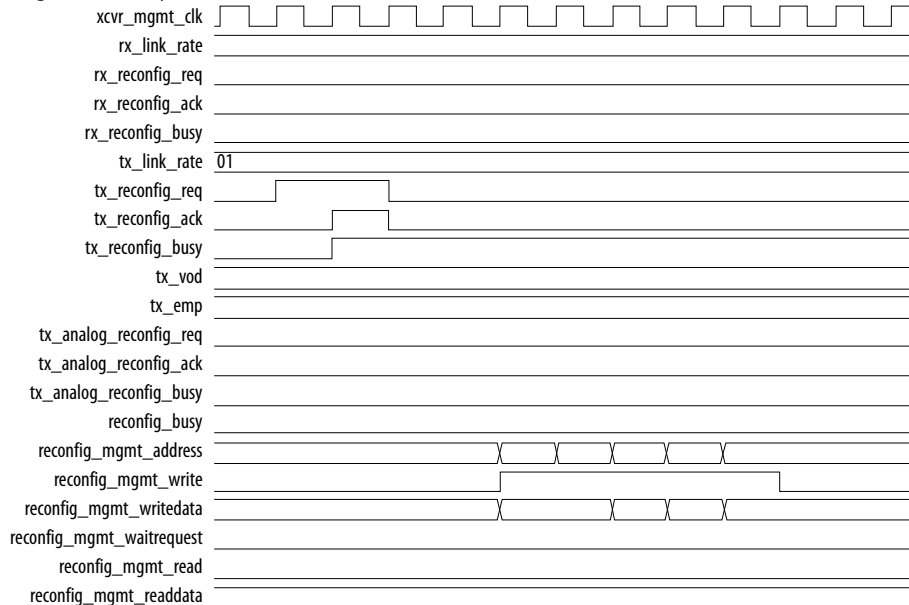
In the timing diagram below, `rx_link_rate` is set to 1 (HBR). When the core makes a request, the `rx_reconfig_req` port goes high. The user logic asserts `rx_reconfig_ack` and then reconfigures the transceiver. During reconfiguration, the user logic holds `rx_reconfig_busy` high; the user logic drives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





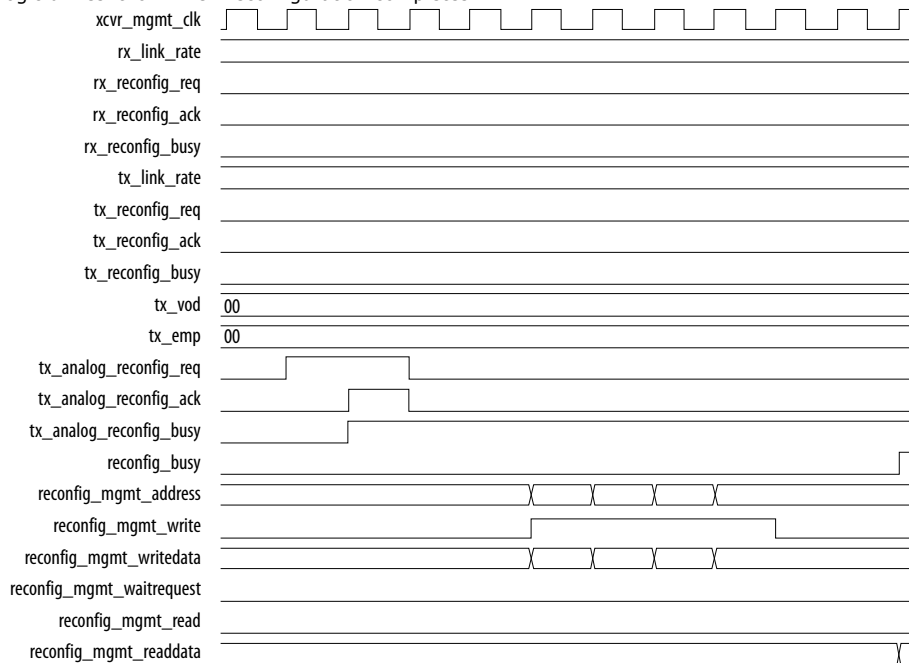
**Figure 34. TX Reconfiguration Waveform**

In the timing diagram below, `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.



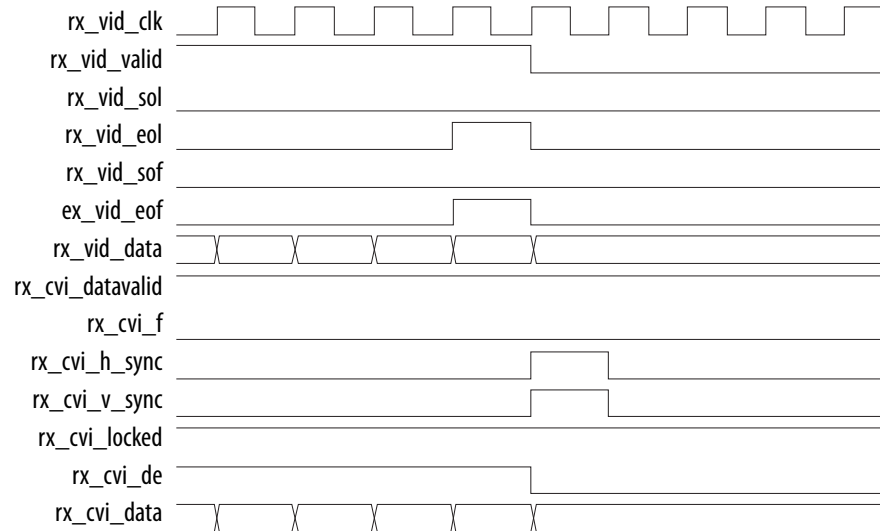
**Figure 35. TX Analog Reconfiguration Waveform**

In the timing diagram below, `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 36. RX Video Waveform**

This timing diagram 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).





## 9. DisplayPort API Reference

You can use the DisplayPort Intel FPGA 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.

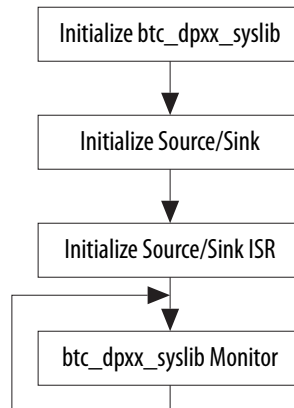
Intel provides software for source and sink instantiations as two system libraries for the Nios II processor (`btc_dptx_syslib` and `btc_dprx_syslib`, respectively). The IP core includes an example main program (`dp_demo_src/main.c`), which demonstrates basic system library use.

### 9.1. 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.

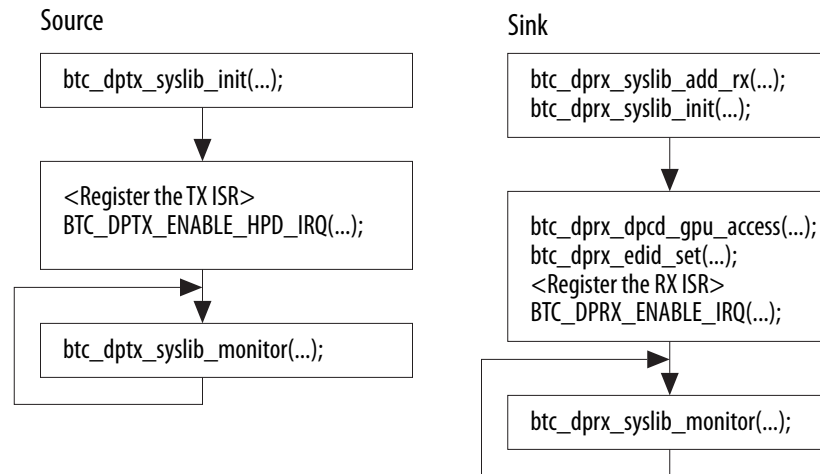
When initialization completes, the user application should periodically invoke the library monitoring function.

**Figure 37. 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 38. 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 3. Typical Sink ISR Implementation

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

### Example 4. 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(...);
```

## 9.2. 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

### 9.3. btc\_dprx\_aux\_get\_request

Prototype:	<pre>int btc_dprx_aux_get_request(     BYTE          rx_idx,     BYTE          *cmd,     unsigned int  *address,     BYTE          *length,     BYTE          *data)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dprx_syslib.h >
Return:	0 = success, 1 = fail
<i>continued...</i>	

Parameters:	<ul style="list-style-type: none"> <li>• rx_idx—Sink instance index (0 - 3)</li> <li>• cmd—Pointer to command</li> <li>• address—Pointer to address</li> <li>• length—Pointer to length (0 - 16)</li> <li>• data—Pointer to data received</li> </ul>
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 <i>VESA DisplayPort Standard</i> 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:	<code>btc_dprx_aux_get_request(0, pcmd, padd, plen, pwrdata);</code>

**Related Information**

[btc\\_dprx\\_aux\\_handler](#) on page 100

## 9.4. btc\_dprx\_aux\_handler

Prototype:	<pre>int btc_dprx_aux_handler(     BYTE      rx_idx     BYTE      cmd,     unsigned int address,     BYTE      length,     BYTE      *data)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dprx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>• rx_idx—Sink instance index (0 - 3)</li> <li>• cmd—Command</li> <li>• address—Address</li> <li>• length—Length (0 - 16)</li> <li>• data—Pointer to data being written</li> </ul>
Description:	<p>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 <i>VESA DisplayPort Standard</i> 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.</p> <p>This function provides all the functionality of the DPCD registers implemented inside the system library, including:</p> <ul style="list-style-type: none"> <li>• DPCD locations read/write support</li> <li>• EDID read support</li> <li>• Link training execution</li> <li>• Forwarding of AUX channel replies back to the source</li> </ul>
Example:	<code>btc_dprx_aux_handler(0, pcmd, padd, plen, pwrdata);</code>

**Related Information**

[btc\\_dprx\\_aux\\_get\\_request](#) on page 99



## 9.5. btc\_dprx\_aux\_post\_reply

Prototype:	<pre>int btc_dprx_aux_post_reply(     BYTE rx_idx     BYTE cmd,     BYTE size,     BYTE *data)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dprx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>rx_idx—Sink instance index (0 - 3)</li> <li>cmd—Command</li> <li>size—Number of data bytes transmitted (0 - 16)</li> <li>data—Pointer to data transmitted</li> </ul>
Description:	This function transmits an AUX channel reply to the connected DisplayPort source. cmd is the reply command byte (refer to the <i>VESA DisplayPort Standard</i> 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:	<pre>btc_dprx_aux_post_reply (0, 0x10, 0, NULL); //Reply AUX_NACK</pre>

### Related Information

[btc\\_dprx\\_aux\\_get\\_request](#) on page 99

## 9.6. 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:	<pre>addr = btc_dprx_baseaddr(0);</pre>

## 9.7. btc\_dprx\_dpccd\_gpu\_access

Prototype:	<pre>int btc_dprx_dpccd_gpu_access(     BYTE rx_idx     BYTE wrcmd,     unsigned int address,     BYTE length,     BYTE *data)</pre>
Thread-safe:	Yes

*continued...*



Available from ISR:	Yes
Include:	< btc_dprx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>rx_idx—Sink instance index (0 - 3)</li> <li>wrcmd—0 = read, 1 = write</li> <li>address—Address</li> <li>length—Length (1 - 255)</li> <li>data—Pointer to data</li> </ul>
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);

## 9.8. btc\_dprx\_edid\_set

Prototype:	<pre>int btc_dprx_edid_set(     BYTE rx_idx     BYTE port,     BYTE *edid_data,     BYTE num_blocks)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dprx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>rx_idx—Sink instance index (0 - 3)</li> <li>port—RX port (stream) number (0 - 3)</li> <li>edid_data—Pointer to EDID data memory</li> <li>num_blocks—EDID size in blocks</li> </ul>
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 bytes long EDIDs). Each streaming sink port has its own EDID.
Example:	btc_dprx_edid_set(0, 0, pmy_edid, 2);

## 9.9. 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
<i>continued...</i>	



Parameters:	rx_idx—Sink instance index (0 - 3)
Description:	Returns the current logic level of the RX HPD.
Example:	<code>btc_dprx_hpd_get(0);</code>

#### Related Information

- [btc\\_dprx\\_hpd\\_pulse](#) on page 103
- [btc\\_dprx\\_hpd\\_set](#) on page 103

### 9.10. btc\_dprx\_hpd\_pulse

Prototype:	<pre>void btc_dprx_hpd_pulse(     BYTE rx_idx,     BYTE dev_irq_vect0,     BYTE dev_irq_vect1,     BYTE link_irq_vect0)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dprx_syslib.h >
Return:	-
Parameters:	<ul style="list-style-type: none"> <li>• rx_idx—Sink instance index (0 - 3)</li> <li>• dev_irq_vect0—Device Service IRQ vector 0. This value is OR-ed to DPCD locations 0x0201 and 0x2003</li> <li>• dev_irq_vect1—Device Service IRQ vector 0. This value is OR-ed to DPCD locations 0x2004</li> <li>• link_irq_vect0—Device Service IRQ vector 0. This value is OR-ed to DPCD locations 0x2005</li> </ul>
Description:	<p>This function deasserts (sets to 0) the RX HPD for 750 s. You can use this function to send an IRQ_HPDPulse to the connected DisplayPort source.</p> <p>DPCD locations 0x0201 and 0x2003-0x2005 are set accordingly to given parameters before the pulse is generated and IRQ vector information is provided to the source.</p> <p>Before invoking this function, you must have invoked <code>btc_dprx_hpd_set</code> with level = 1 (HPD must be set to 1).</p>
Example:	<code>btc_dprx_hpd_pulse(0, 0, 0, 0);</code>

#### Related Information

- [btc\\_dprx\\_hpd\\_get](#) on page 102
- [btc\\_dprx\\_hpd\\_set](#) on page 103

### 9.11. btc\_dprx\_hpd\_set

Prototype:	<pre>void btc_dprx_hpd_set(     BYTE rx_idx,     int level)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dprx_syslib.h >
<b>continued...</b>	

Return:	-
Parameters:	<ul style="list-style-type: none"> <li>rx_idx—Sink instance index (0 - 3)</li> <li>level—0 or 1</li> </ul>
Description:	This function allows the controller to set the logic level of the RX HPD.
Example:	<code>btc_dprx_hpd_set(0,1);</code>

#### Related Information

- [btc\\_dprx\\_hpd\\_get](#) on page 102
- [btc\\_dprx\\_hpd\\_pulse](#) on page 103

### 9.12. btc\_dprx\_lt\_eyeq\_init

Prototype:	<pre>void btc_dprx_lt_eyeq_init(     BYTE rx_idx,     BYTE enabled,     BYTE log_chan_from,     BYTE log_chan_to,     unsigned int rcnf_base_addr)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dprx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>rx_idx—Sink instance index (0 - 3)</li> <li>enabled—0 to disable Eye Viewer (default), 1 to enable Eye Viewer</li> <li>log_chan_from—Reconfiguration controller first logical channel related to this sink (lane0)</li> <li>log_chan_to—Reconfiguration controller last logical channel related to this sink (higher lane supported)</li> <li>rcnf_base_addr—Reconfiguration controller base address</li> </ul>
Description:	This function to enable or disable equalizer (AC Gain) automatic management using the Eye Viewer feature of supporting devices. When enabled, a number of RX transceiver features must be supported and their reconfiguration must be enabled too.
Example:	<code>btc_dprx_lt_eyeq_init (0,1,0,3,RECONFIG_MGMT_BASE);</code>

### 9.13. btc\_dprx\_lt\_force

Prototype:	<pre>void btc_dprx_lt_force(BYTE rx_idx)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dprx_syslib.h >
Return:	-
<i>continued...</i>	





Parameters:	<code>rx_idx</code> —Sink instance index (0 - 3)
Description:	This function brings the main link down and generates an IRQ_HPD forcing the connected source to perform a new Link Training.
Example:	<code>btc_dprx_lt_force (0);</code>

## 9.14. btc\_dprx\_rtl\_ver

Prototype:	<pre>void btc_dprx_rtl_ver(   BYTE *major,   BYTE *minor,   BYTE *rev)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	<code>&lt; btc_dprx_syslib.h &gt;</code>
Return:	–
Parameters:	<ul style="list-style-type: none"> <li><code>major</code>—Pointer to major version</li> <li><code>minor</code>—Pointer to minor version</li> <li><code>rev</code>—Pointer to revision)</li> </ul>
Description:	This function returns the version of the RX core (RTL).
Example:	<code>btc_dprx_rtl_ver(&amp;maj, &amp;min, &amp;rev);</code>

## 9.15. btc\_dprx\_sw\_ver

Prototype:	<pre>void btc_dprx_sw_ver(   BYTE *major,   BYTE *minor,   BYTE *rev)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	<code>&lt; btc_dprx_syslib.h &gt;</code>
Return:	–
Parameters:	<ul style="list-style-type: none"> <li><code>major</code>—Pointer to major version</li> <li><code>minor</code>—Pointer to minor version</li> <li><code>rev</code>—Pointer to revision)</li> </ul>
Description:	This function returns the version of the RX system library.
Example:	<code>btc_dprx_sw_ver(&amp;maj, &amp;min, &amp;rev);</code>

## 9.16. btc\_dprx\_syslib\_add\_rx

Prototype:	<pre>int btc_dprx_syslib_add_rx(   BYTE rx_idx,   unsigned int rx_base_addr,</pre>
<i>continued...</i>	

	<pre>unsigned int rx_irq_id, unsigned int rx_irq_num, unsigned int rx_num_of_sinks, unsigned int options)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dprx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>• rx_idx—Sink instance index (0 - 3)</li> <li>• rx_base_addr—RX base address</li> <li>• rx_irq_id—RX IRQ ID</li> <li>• rx_irq_num—RX IRQ number</li> <li>• rx_num_of_sinks—Number of streaming sinks used (1 - 4)</li> <li>• options—OR-ed options for this instance or 0 if unused</li> </ul>
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 sinks have been declared, invoke btc_dprx_syslib_init ( ).
Example:	<pre>btc_dprx_syslib_add_rx (0, DP_RX_SINK_BASE, DP_RX_SINK_IRQ_INTERRUPT_CONTROLLER_ID, DP_RX_SINK_IRQ, 2, BTC_DPRX_OPT_DISABLE_ERRMON);</pre>

### Related Information

[btc\\_dprx\\_syslib\\_init](#) on page 106

## 9.17. btc\_dprx\_syslib\_info

Prototype:	<pre>void btc_dprx_syslib_info( BYTE *max_sink_num, BYTE *mst_support)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dprx_syslib.h >
Return:	None
Parameters:	<ul style="list-style-type: none"> <li>• max_sink_num—Pointer for maximum number of sinks supported</li> <li>• mst_support—Pointer for MST support</li> </ul>
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:	<pre>btc_dprx_syslib_info(pmaxsink, pmst);</pre>

## 9.18. btc\_dprx\_syslib\_init

Prototype:	int btc_dprx_syslib_init(void)
Thread-safe:	No
<i>continued...</i>	



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 <code>btc_dprx_syslib_add_rx ( )</code> .
Example:	<code>btc_dprx_syslib_init();</code>

### Related Information

[btc\\_dprx\\_syslib\\_add\\_rx](#) on page 105

## 9.19. btc\_dprx\_syslib\_monitor

Prototype:	<code>int btc_dprx_syslib_monitor(void)</code>
Thread-safe:	No
Available from ISR:	No
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: <ul style="list-style-type: none"> <li>• Handling RX-side received sideband message requests.</li> <li>• Forwarding RX-side sideband message replies.</li> </ul> The software should invoke this function periodically or at least every 50 ms.
Example:	<code>btc_dprx_syslib_monitor();</code>

## 9.20. 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

## 9.21. btc\_dptx\_aux\_i2c\_read

Prototype:	<pre>int btc_dptx_aux_i2c_read(     BYTE tx_idx,     BYTE address,</pre>
<i>continued...</i>	

	<pre>BYTE size, BYTE *data, BYTE mot)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>address—I<sup>2</sup>C address</li> <li>size—Number of bytes (1 - 16)</li> <li>data—Pointer to data to be read</li> <li>mot—Middle of transaction (0 or 1)</li> </ul>
Description:	This function reads 1 to 16 data bytes from the connected DisplayPort sink's I <sup>2</sup> C interface mapped over the AUX channel.
Example:	<code>btc_dptx_aux_i2c_read(0, 0x50, 16, data, 1);</code>

### Related Information

[btc\\_dptx\\_aux\\_i2c\\_write](#) on page 108

## 9.22. btc\_dptx\_aux\_i2c\_write

Prototype:	<pre>int btc_dptx_aux_i2c_write(     BYTE tx_idx,     BYTE address,     BYTE size,     BYTE *data,     BYTE mot)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>address—I<sup>2</sup>C address</li> <li>size—Number of bytes (1 - 16)</li> <li>data—Pointer to data to be written</li> <li>mot—Middle of transaction (0 or 1)</li> </ul>
Description:	This function writes 1 to 16 data bytes to the connected DisplayPort sink's I <sup>2</sup> C interface mapped over the AUX channel.
Example:	<code>btc_dptx_aux_i2c_write(0, 0x50, 1, data, 1);</code>

### Related Information

[btc\\_dptx\\_aux\\_i2c\\_read](#) on page 107



## 9.23. btc\_dptx\_aux\_read

Prototype:	<pre>int btc_dptx_aux_read(   BYTE tx_idx,   unsigned int address,   BYTE size,   BYTE *data)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	<ul style="list-style-type: none"> <li>• 0 = AUX_ACK replied</li> <li>• 1 = Source internal error</li> <li>• 2 = Reply timeout</li> <li>• 3 = AUX_NACK replied</li> <li>• 4 = AUX_DEFER replied</li> <li>• 5 = Invalid reply</li> </ul>
Parameters	<ul style="list-style-type: none"> <li>• tx_idx—Source instance index (0 - 3)</li> <li>• address—DPCD start address</li> <li>• size—Number of bytes (1 - 16)</li> <li>• data—Pointer for data to be read</li> </ul>
Description:	This function reads 1 to 16 data bytes from the connected DisplayPort sink's DPCD.
Example:	<code>btc_dptx_aux_read(0, 0x202, 2, &amp;status);</code>

### Related Information

[btc\\_dptx\\_aux\\_write](#) on page 109

## 9.24. btc\_dptx\_aux\_write

Prototype:	<pre>int btc_dptx_aux_write(   BYTE tx_idx,   unsigned int address,   BYTE size,   BYTE *data)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	<ul style="list-style-type: none"> <li>• 0 = AUX_ACK replied</li> <li>• 1 = Source internal error</li> <li>• 2 = Reply timeout</li> <li>• 3 = AUX_NACK replied</li> <li>• 4 = AUX_DEFER replied</li> <li>• 5 = Invalid reply</li> </ul>
<i>continued...</i>	

Parameters	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>address—DPCD start address</li> <li>size—Number of bytes (1 - 16)</li> <li>data—Pointer to data to be written</li> </ul>
Description:	This function writes 1 to 16 data bytes to the connected DisplayPort sink's DPCD.
Example:	<code>btc_dptx_aux_write(0, 0x600, 1, data_ptr);</code>

### Related Information

[btc\\_dptx\\_aux\\_read](#) on page 109

## 9.25. btc\_dptx\_baseaddr

Prototype:	<code>unsigned int btc_dptx_baseaddr(BYTE tx_idx)</code>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	<code>&lt; btc_dptx_syslib.h &gt;</code>
Return:	Base address
Parameters:	tx_idx—Source instance index (0 - 3)
Description:	This function returns the base address of the TX instance connected to the given port number.
Example:	<code>addr = btc_dptx_baseaddr(0);</code>

## 9.26. btc\_dptx\_edid\_block\_read

Prototype:	<pre>int btc_dptx_edid_block_read(     BYTE tx_idx,     BYTE block,     BYTE *data)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	<code>&lt; btc_dptx_syslib.h &gt;</code>
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>block—Block number (0 - 3)</li> <li>data—Pointer for data to be read</li> </ul>
Description:	Reads one block (128 bytes) from the EDID of the connected DisplayPort sink.
Example:	<code>btc_dptx_edid_block_read(0, 2, pdata);</code>

### Related Information

[btc\\_dptx\\_edid\\_read](#) on page 111



## 9.27. btc\_dptx\_edid\_read

Prototype:	<pre>int btc_dptx_edid_read(     BYTE tx_idx,     BYTE *data)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	Number of bytes read from EDID (0=fail)
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>data—Pointer for data to be read</li> </ul>
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:	<code>btc_dptx_edid_read(0, pdata);</code>

### Related Information

[btc\\_dptx\\_edid\\_block\\_read](#) on page 110

## 9.28. btc\_dptx\_fast\_link\_training

Prototype:	<pre>int btc_dptx_fast_link_training(     BYTE tx_idx,     unsigned int link_rate,     unsigned int lane_count,     unsigned int volt_swing,     unsigned int pre_emph,     unsigned int new_cfg)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>link_rate—Link rate: 0x06 = 1.62 Gbps; 0x0A = 2.70 Gbps; 0x14 = 5.40 Gbps</li> <li>lane_count—1, 2, or 4</li> <li>volt_swing—0, 1, 2, or 3</li> <li>pre_emph—0, 1, 2, or 3</li> <li>new_cfg—0 = ignore the other parameters; 1 = use provided parameters</li> </ul>
Description:	<p>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).</p> <ul style="list-style-type: none"> <li>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.</li> <li>If new_cfg = 0, the IP core uses the current transceiver setting, link rate, and lane count, and performs fast link training.</li> </ul>
Example:	<code>btc_dptx_fast_link_training(0, 0x0A, 4, 1, 0, 1);</code>

### Related Information

[btc\\_dptx\\_link\\_training](#) on page 113

## 9.29. btc\_dptx\_hpd\_change

Prototype:	<pre>int btc_dptx_hpd_change( BYTE tx_idx, unsigned int asserted)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	None
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>asserted—0=asserted, 1=deasserted</li> </ul>
Description:	This function informs the system library that the RX HPD signal state has changed. Invoke after an HPD stable state change (not after an HPD_IRQ)
Example:	<code>btc_dptx_hpd_change(0, 1);</code>

## 9.30. btc\_dptx\_is\_link\_up

Prototype:	<pre>int btc_dptx_is_link_up(BYTE tx_idx)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = link is down, 1 = link is up
Parameters:	tx_idx—Source instance index (0 - 3)
Description:	This function returns "1" if the main link is currently up and correctly link trained.
Example:	<code>btc_dptx_is_link_up(0);</code>

## 9.31. btc\_dptx\_link\_bw

Prototype:	<pre>int btc_dptx_link_bw(BYTE tx_idx)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = link is down, >0 = link bandwidth (162-2160 Mbps)
Parameters:	tx_idx—Source instance index (0 - 3)
Description:	This function returns the main link current bandwidth in Mbytes/s
Example:	<code>btc_dptx_link_bw(0);</code>





## 9.32. btc\_dptx\_link\_training

Prototype:	<pre>int btc_dptx_link_training(     BYTE tx_idx,     unsigned int link_rate,     unsigned int lane_count)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>link_rate—Link rate: 0x06 = 1.62 Gbps; 0x0A = 2.70 Gbps; 0x14 = 5.40 Gbps</li> <li>lane_count—1, 2, or 4</li> </ul>
Description:	This function performs link training with the connected DisplayPort sink.
Example:	<pre>btc_dptx_link_training(0, 0x06, 4);</pre>

## 9.33. btc\_dptx\_rtl\_ver

Prototype:	<pre>void btc_dptx_rtl_ver(     BYTE *major     BYTE *minor,     BYTE *rev)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	-
Parameters:	<ul style="list-style-type: none"> <li>major—Pointer to major version</li> <li>minor—Pointer to minor version</li> <li>rev—Pointer to revision)</li> </ul>
Description:	This function returns the version of the TX core (RTL).
Example:	<pre>btc_dptx_rtl_ver(&amp;maj, &amp;min, &amp;rev);</pre>

## 9.34. btc\_dptx\_set\_color\_space

Prototype:	<pre>int btc_dptx_set_color_space (     BYTE tx_idx,     BYTE format,     BYTE bpc,     BYTE range,     BYTE colorimetry,     BYTE use_vsc_sdp )</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
<b>continued...</b>	

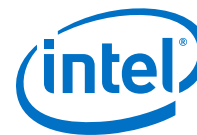
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>format—0 = RGB; 1 = YCbCr 4:4:4; 2 = YCbCr 4:2:2, 3 = YCbCr 4:2:0</li> <li>bpc—Color depth: 0 = 6bpc; 1 = 8 bpc; 2 = 10 bpc; 3 = 12 bpc; 4 = 16 bpc</li> <li>range—0 = VESA; 1 = CEA</li> <li>colorimetry—0 = BT601-5; 1 = BT709-5; refer to Table 2–120 bit[3:0] in the <i>VESA DisplayPort Standard version 1.4</i> for all colorimetry support including BT.2020.</li> <li>use_vsc_sdp—0 = use MISC0; 1 = use VSC_SDP</li> </ul> <p><i>Note:</i> If you configure use_vsc_sdp to use VSC SDP, refer to the <i>VESA DisplayPort Standard version 1.4</i> for the VSC SDP Payload Pixel Encoding/Colorimetry Format.</p>
Description:	This function sets the color space for TX (stream 0) transmitted video.
Example:	<code>btc_dptx_set_color_space(0, 0, 1, 0, 0, 0);</code>

### 9.35. btc\_dptx\_sw\_ver

Prototype:	<pre>void btc_dptx_sw_ver(     BYTE *major,     BYTE *minor,     BYTE *rev)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	-
Parameters:	<ul style="list-style-type: none"> <li>major—Pointer to major version</li> <li>minor—Pointer to minor version</li> <li>rev—Pointer to revision)</li> </ul>
Description:	This function returns the version of the TX system library.
Example:	<code>btc_dptx_sw_ver(&amp;maj, &amp;min, &amp;rev);</code>

### 9.36. btc\_dptx\_syslib\_add\_tx

Prototype:	<pre>int btc_dptx_syslib_add_tx(     BYTE tx_idx,     unsigned int tx_base_addr,     unsigned int tx_irq_id,     unsigned int tx_irq_num)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
<i>continued...</i>	



Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>tx_base_addr—TX base address</li> <li>tx_irq_id—TX IRQ ID</li> <li>tx_irq_num—TX IRQ number</li> </ul>
Description:	This function declares a source (TX) instance to the system library. It should be invoked once for each existing source instance, starting from tx_idx = 0. After all sources have been declared, invoke btc_dptx_syslib_init ( ).
Example:	<pre> btc_dptx_syslib_init (0, DP_TX_SOURCE_BASE, DP_TX_SOURCE_IRQ_INTERRUPT_CONTROLLER_ID, DP_TX_SOURCE_IRQ); </pre>

### 9.37. btc\_dptx\_syslib\_init

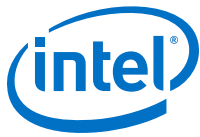
Prototype:	<code>int btc_dptx_syslib_init(void)</code>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	None
Description:	Initializes the system library. Should be invoked just once after btc_dptx_syslib_add_tx().
Example:	<code>btc_dptx_syslib_init ( );</code>

### 9.38. btc\_dptx\_syslib\_monitor

Prototype:	<code>int btc_dptx_syslib_monitor(void)</code>
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:	<code>btc_dptx_syslib_monitor();</code>

### 9.39. btc\_dptx\_test\_autom

Prototype:	<code>int btc_dptx_test_autom(BYTE tx_idx)</code>
Thread-safe:	No
<b>continued...</b>	



Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	tx_idx—Source instance index (0 - 3)
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:	<code>btc_dptx_test_autom(0);</code>

## 9.40. btc\_dptx\_video\_enable

Prototype:	<pre>int btc_dptx_video_enable(   BYTE tx_idx,   BYTE enabled)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"><li>tx_idx—Source instance index (0 - 3)</li><li>enabled—0 = output idle pattern; 1 = output active video</li></ul>
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:	<code>btc_dptx_video_enable(0, 1);</code>

## 9.41. btc\_dptx\_mst\_allocate\_payload\_rep

Prototype:	<pre>int btc_dptx_mst_allocate_payload_rep(   BYTE tx_idx,   BYTE *GUID,   BYTE *reas_for_nak,   BYTE *nak_data)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = ACK, 1 = NACK, 2 = Not ready
<i>continued...</i>	



Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>GUID—For NAK replies, GUID originating the NAK</li> <li>reas_for_nak—For NAK replies, reason_for_nak (pointer to 1 byte)</li> <li>nak_data—For NAK replies, nak_data (pointer to 2 bytes)</li> </ul>
Description:	This function returns the connected DisplayPort sink reply to the last issued ALLOCATE_PAYLOAD_DOWN_REQ MST sideband message. Call this function until either ACK or NACK is returned. '2' is returned when the reply has not yet been received.
Example:	<code>btc_dptx_mst_allocate_payload_rep(0, p_GUID, p_rfn, p_nd);</code>

## 9.42. btc\_dptx\_mst\_allocate\_payload\_req

Prototype:	<pre>int btc_dptx_mst_allocate_payload_req(   BYTE tx_idx,   BTC_RAD *RAD,   BYTE port_number,   BYTE num_sdp_streams,   BYTE *sdp_stream_sinks,   BYTE vcp_id,   unsigned int pbn)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>RAD—MST Relative Address of the destination</li> <li>port_number—Downstream device output port number</li> <li>num_sdp_streams—Number of SDP streams routed</li> <li>sdp_stream_sinks—SDP stream sink identifiers, one for each of the SDP streams routed</li> <li>vcp_id—VC Payload ID (1-7, 15)</li> <li>pbn—PBN allocated</li> </ul>
Description:	This function issues ALLOCATE_PAYLOAD_DOWN_REQ MST sideband message. Recommended VCP ID values are 1 - 7 for data streams in use, 15 for unused streams.
Example:	<code>btc_dptx_mst_allocate_payload_req(0, aRAD, 9, 2, psss, id, 32);</code>

## 9.43. btc\_dptx\_mst\_clear\_payload\_table\_rep

Prototype:	<pre>int btc_dptx_mst_clear_payload_table_rep(   BYTE tx_idx,   BYTE *GUID,   BYTE *reas_for_nak,   BYTE *nak_data)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = ACK, 1 = NACK, 2 = Not ready
<i>continued...</i>	

Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>GUID—For NAK replies, GUID originating the NAK</li> <li>reas_for_nak—For NAK replies, reason_for_nak (pointer to 1 byte)</li> <li>nak_data—For NAK replies, nak_data (pointer to 2 bytes)</li> </ul>
Description:	This function returns the connected DisplayPort sink reply to the last issued CLEAR_PAYLOAD_ID_TABLE_DOWN_REQ_MST sideband message. Call this function until either ACK or NACK is returned. '2' is returned when the reply has not yet been received.
Example:	<code>btc_dptx_mst_clear_payload_table_rep(0, p_GUID, p_rfn, p_nd);</code>

## 9.44. btc\_dptx\_mst\_clear\_payload\_table\_req

Prototype:	<pre>int btc_dptx_mst_clear_payload_table_req( BYTE tx_idx, BTC_RAD *RAD)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>RAD—MST Relatives Address of the Destination</li> </ul>
Description:	This function issues a CLEAR_PAYLOAD_ID_TABLE_DOWN_REQ_MST sideband message.
Example:	<code>btc_dptx_mst_clear_payload_table_req(0, aRAD);</code>

## 9.45. btc\_dptx\_mst\_conn\_stat\_notify\_req

Prototype:	<pre>btc_dptx_mst_conn_stat_notify_req( BYTE tx_idx)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	Pointer to UP_REQ data, NULL = none pending
Parameters:	tx_idx—Source instance index (0 - 3)
Description:	This function returns the last pending CONNECTION_STATE_NOTIFY_UP_REQ received, if any.
Example:	<code>btc_dptx_mst_conn_stat_notify_req(0);</code>

## 9.46. btc\_dptx\_mst\_down\_rep\_irq

Prototype:	<pre>int btc_dptx_mst_down_rep_irq(BYTE tx_idx)</pre>
<b>continued...</b>	



Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	tx_idx —Source instance index (0 - 3)
Description:	This function must be invoked every time the connected DisplayPort sink issues an HPD_IRQ with DOWN_REP_MSG_RDY = 1
Example:	btc_dptx_mst_down_rep_irq(0);

### 9.47. btc\_dptx\_mst\_enable

Prototype:	<pre>int btc_dptx_mst_enable(   BYTE tx_idx,   BYTE enabled)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	None
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>enabled—0 = SST framing; 1 = MST framing</li> </ul>
Description:	This function enables or disables MST framing. After HW reset framing is set by default to SST.
Example:	btc_dptx_mst_enable(0,1);

### 9.48. btc\_dptx\_mst\_enum\_path\_rep

Prototype:	<pre>int btc_dptx_mst_enum_path_rep(   BYTE tx_idx,   BTC_MST_PATH_PBN *path_pbn,   BYTE *GUID,   BYTE *reas_for_nak,   BYTE *nak_data)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = ACK, 1 = NACK, 2 = Not ready
<b>continued...</b>	

Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>path_pbn—Replied data</li> <li>GUID—For NAK replies, GUID originating the NAK</li> <li>reas_for_nak—For NAK replies, reason_for_nak (pointer to 1 byte)</li> <li>nak_data—For NAK replies, nak_data (pointer to 2 bytes)</li> </ul>
Description:	This function returns the connected DisplayPort sink reply to the last issued ENUM_PATH_RESOURCES_DOWN_REQ_MST sideband message. Call this function until either ACK or NACK is returned. '2' is returned when the reply has not yet been received.
Example:	<code>btc_dptx_mst_enum_path_rep(0, p_ppbn, p_GUID, p_rfn, p_nd);</code>

### 9.49. btc\_dptx\_mst\_enum\_path\_req

Prototype:	<pre>int btc_dptx_mst_enum_path_req(     BYTE tx_idx,     BTC_RAD *RAD,     BYTE port_number)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = ACK, 1 = NACK, 2 = Not ready
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>RAD—MST Relatives Address of the Destination</li> <li>port_number—Downstream device output port number</li> </ul>
Description:	This function issues a ENUM_PATH_RESOURCES_DOWN_REQ_MST sideband message.
Example:	<code>btc_dptx_mst_enum_path_req(0, aRAD, 9);</code>

### 9.50. btc\_dptx\_mst\_get\_msg\_transact\_ver\_rep

Prototype:	<pre>int btc_dptx_mst_get_msg_transact_ver_rep(     BYTE tx_idx,     BYTE *version,     BYTE *GUID,     BYTE *reas_for_nak,     BYTE *nak_data)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = ACK, 1 = NACK, 2 = Not ready
<i>continued...</i>	





Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>version—Replied version number</li> <li>GUID—For NAK replies, GUID originating the NAK</li> <li>reas_for_nak—For NAK replies, reason_for_nak (pointer to 1 byte)</li> <li>nak_data—For NAK replies, nak_data (pointer to 2 bytes)</li> </ul>
Description:	This function returns the connected DisplayPort sink reply to the last issued GET_MESSAGE_TRANSACTION_VERSION_DOWN_REQ_MST sideband message. Call this function until either ACK or NACK is returned. '2' is returned when the reply has not yet been received.
Example:	<code>btc_dptx_mst_get_msg_transact_ver_rep (0, &amp;ver, p_GUID, p_rfn, p_nd);</code>

### 9.51. btc\_dptx\_mst\_get\_msg\_transact\_ver\_req

Prototype:	<pre>int btc_dptx_mst_get_msg_transact_ver_req(     BYTE tx_idx,     BTC_RAD *RAD,     BYTE port_number)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>RAD—MST Relative Address of the destination</li> <li>port_number—Downstream device output port number</li> </ul>
Description:	This function issues a GET_MESSAGE_TRANSACTION_VERSION_DOWN_REQ_MST sideband message.
Example:	<code>btc_dptx_mst_get_msg_transact_ver_req(0, aRAD, 9);</code>

### 9.52. btc\_dptx\_mst\_link\_address\_rep

Prototype:	<pre>int btc_dptx_mst_allocate_payload_rep(     BYTE tx_idx,     BTC_MST_DEVICE *device,     BYTE *GUID,     BYTE *reas_for_nak,     BYTE *nak_data)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = ACK, 1 = NACK, 2 = Not ready
<i>continued...</i>	

Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>device—Replied data</li> <li>GUID—For NAK replies, GUID originating the NAK</li> <li>reas_for_nak—For NAK replies, reason_for_nak (pointer to 1 byte)</li> <li>nak_data—For NAK replies, nak_data (pointer to 2 bytes)</li> </ul>
Description:	This function returns the connected DisplayPort sink reply to the last issued LINK_ADDRESS_DOWN_REQ MST sideband message. Call this function until either ACK or NACK is returned. '2' is returned when the reply has not yet been received.
Example:	<code>btc_dptx_mst_link_address_rep(0, p_dev, p_GUID, p_rfn, p_nd);</code>

### 9.53. btc\_dptx\_mst\_link\_address\_req

Prototype:	<pre>int btc_dptx_mst_link_address_req( BYTE tx_idx, BTC_RAD *RAD)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>RAD—MST Relatives Address of the destination</li> </ul>
Description:	This function issues a LINK_ADDRESS_DOWN_REQ MST sideband message.
Example:	<code>btc_dptx_mst_link_address_req(0, aRAD);</code>

### 9.54. btc\_dptx\_mst\_remote\_dpcd\_wr\_rep

Prototype:	<pre>int btc_dptx_mst_remote_dpcd_wr_rep( BYTE tx_idx, BYTE *GUID, BYTE *reas_for_nak, BYTE *nak_data)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = ACK, 1 = NACK, 2 = Not ready
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>GUID—For NAK replies, GUID originating the NAK</li> <li>reas_for_nak—For NAK replies, reason_for_nak (pointer to 1 byte)</li> <li>nak_data—For NAK replies, nak_data (pointer to 2 bytes)</li> </ul>
Description:	This function returns the connected DisplayPort sink reply to the last issued REMOTE_DPCD_WRITE_DOWN_REQ MST sideband message. Call this function until either ACK or NACK is returned. '2' is returned when the reply has not yet been received.
Example:	<code>btc_dptx_mst_remote_dpcd_wr_rep(0, p_GUID, p_rfn, p_nd);</code>



## 9.55. btc\_dptx\_mst\_remote\_dpcd\_wr\_req

Prototype:	<pre>int btc_dptx_mst_remote_dpcd_wr_req(     BYTE tx_idx,     BTC_RAD *RAD,     BYTE port_number,     unsigned int addr,     BYTE length,     BYTE *data)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>RAD—MST Relatives Address of the Destination</li> <li>port_number—Downstream device output port number</li> <li>addr—DPCD address</li> <li>length—Number of bytes to write</li> <li>data—Data to be written</li> </ul>
Description:	This function issues a REMOTE_DPCD_WRITE_DOWN_REQ MST sideband message.
Example:	<pre>btc_dptx_mst_remote_dpcd_wr_req(0, aRAD, 9, 0x68000, 1, p_data);</pre>

## 9.56. btc\_dptx\_mst\_remote\_i2c\_rd\_rep

Prototype:	<pre>int btc_dptx_mst_remote_i2c_rd_rep(     BYTE tx_idx,     BTC_MST_I2C_RD_DATA *data,     BYTE *GUID,     BYTE *reas_for_nak,     BYTE *nak_data)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = ACK, 1 = NACK, 2 = Not ready
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>data—Replied data</li> <li>GUID—For NAK replies, GUID originating the NAK</li> <li>reas_for_nak—For NAK replies, reason_for_nak (pointer to 1 byte)</li> <li>nak_data—For NAK replies, nak_data (pointer to 2 bytes)</li> </ul>
Description:	This function returns the connected DisplayPort sink reply to the last issued REMOTE_I2C_READ_DOWN_REQ MST sideband message. Call this function until either ACK or NACK is returned. '2' is returned when the reply has not yet been received.
Example:	<pre>btc_dptx_mst_remote_i2c_rd_rep(0, p_buf, p_GUID, p_rfn, p_nd);</pre>

## 9.57. btc\_dptx\_mst\_remote\_i2c\_rd\_req

Prototype:	<pre>int btc_dptx_mst_remote_i2c_rd_req(     BYTE tx_idx,     BTC_RAD *RAD,     BYTE port_number,     BYTE num_of_wr_trans,     BTC_MST_I2C_WR_TRANS *wr_trans,     BYTE rd_i2c_dev_id,     BYTE num_of_rd_bytes)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>RAD—MST Relatives Address of the destination</li> <li>port_number—Downstream device output port number</li> <li>num_of_wr_trans—Number of write transactions</li> <li>wr_trans—Array of write transactions</li> <li>rd_i2c_dev_id—Read I<sup>2</sup>C device identifier</li> <li>num_of_rd_bytes—Number of bytes to read</li> </ul>
Description:	This function issues a REMOTE_I2C_READ_DOWN_REQ_MST sideband message.
Example:	<pre>btc_dptx_mst_remote_i2c_rd_req(0, aRAD, 9, 1, wr_trans, 0x50, 128);</pre>

## 9.58. btc\_dptx\_mst\_set\_color\_space

Prototype:	<pre>int btc_dptx_mst_set_color_space(     BYTE tx_idx,     BYTE strm_idx,     BYTE format,     BYTE bpc,     BYTE range,     BYTE colorimetry)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>strm_idx—Stream index (0 - 3)</li> <li>format—0 = RGB; 1 = YCbCr 4:4:4; 2 = YCbCr 4:2:2, 3 = YCbCr 4:2:0</li> <li>bpc—Color depth: 0 = 6bpc; 1 = 8 bpc; 2 = 10 bpc; 3 = 12 bpc; 4 = 16 bpc</li> <li>range—0 = VESA; 1 = CEA</li> <li>colorimetry—0 = BT601-5; 1 = BT709-5; refer to Table 2–120 bit[3:0] in the <i>VESA DisplayPort Standard version 1.4</i> for all colorimetry support including BT.2020.</li> </ul>
Description:	This function sets the color space for video transmitted by a video stream of the TX.
Example:	<pre>btc_dptx_mst_set_color_space(0, 0, 0, 1, 0, 0);</pre>



## 9.59. btc\_dptx\_mst\_tavgts\_set

Prototype:	<pre>int btc_dptx_mst_tavgts_set(   BYTE tx_idx,   BYTE strm_idx,   BYTE value)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>strm_idx—Stream index (0 - 3)</li> <li>value—Target Average Time Slots value (0-64).</li> </ul>
Description:	This function sets Target Average Time Slots value. A value of 64 causes the VCP Fill sequence to occupy every time slot allocated for strm_idx stream.
Example:	<code>btc_dptx_mst_tavgts_set(0, p_ppbn, p_GUID, p_rfn, p_nd);</code>

The Target Average Time Slots value (TAVG\_TS) is expressed as the fractional part of the number of time slots per MTU occupied by a stream times 64, assuming that the allocated time slots are the ceiling of this number.

For instance, if 4.7 time slots/MTU are occupied (5 time slots/MTU are allocated in the VCP ID Table):

$$\text{TAVG\_TS} = \text{CEIL}(\text{FRAC}(4.7) * 64) = \text{CEIL}(0.7 * 64) = 45$$

If TAVG\_TS is set to 64, VCP Fill is produced to each time slot allocated to the stream.

## 9.60. btc\_dptx\_mst\_up\_req\_irq

Prototype:	<pre>int btc_dptx_mst_up_req_irq(BYTE tx_idx)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	tx_idx—Source instance index (0 - 3)
Description:	This function must be invoked every time the connected DisplayPort sink issues an HPD_IRQ with UP_REQ_MSG_RDY = 1
Example:	<code>btc_dptx_mst_up_req_irq(0);</code>

The system library uses this function to handle MST sideband messages.

## 9.61. btc\_dptx\_mst\_vcpid\_set

Prototype:	<pre>int btc_dptx_mst_vcpid_set( BYTE tx_idx, BYTE strm_idx, BYTE vcpid)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>strm_idx—0 = Stream index (0 - 3)</li> <li>vcpid—VC Payload ID (1-7, 15)</li> </ul>
Description:	This function sets the VC Payload ID for one stream. Recommended VCP ID values are 1 - 7 for data streams in use, 15 for unused streams.
Example:	<code>btc_dptx_mst_vcpid_set(0,0,1);</code>

## 9.62. btc\_dptx\_mst\_vcptab\_addvc

Prototype:	<pre>int btc_dptx_mst_vcptab_addvc( BYTE tx_idx, BYTE vc_size, BYTE vc_id)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = success, >0 = index of first time slot allocated
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>vc_size—VC size in timeslot (1-63)</li> <li>vc_payload—VC Payload ID (0-7)</li> </ul>
Description:	This function allocates a Virtual Channel (VC) in the local VC Payload ID Table. Recommended VCP ID values are 1 - 7 for data streams in use, 0 for unused streams.
Example:	<code>btc_dptx_mst_vcptab_addvc(0,10,2);</code>

## 9.63. btc\_dptx\_mst\_vcptab\_clear

Prototype:	<pre>int btc_dptx_mst_vcptab_clear(BYTE tx_idx)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	None
<i>continued...</i>	



Parameters:	tx_idx—Source instance index (0 - 3)
Description:	This function clears the local VC Payload ID Table and all the VC Payload IDs. All table entries are set to '0' and all the VCP IDs are set to 15.
Example:	<code>btc_dptx_mst_vcptab_clear(0);</code>

## 9.64. btc\_dptx\_mst\_vcptab\_delvc

Prototype:	<pre>int btc_dptx_mst_vcptab_delvc(   BYTE tx_idx,   BYTE vc_id)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	0 = fail, > 0 = index of first time slot deleted
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>vc_id—VC Payload ID (1-7).</li> </ul>
Description:	This function deletes a Virtual Channel (VC) from the local VC Payload ID Table. All the VC table entries are set to '0'.
Example:	<code>btc_dptx_mst_vcptab_delvc(0,2);</code>

## 9.65. btc\_dptx\_mst\_vcptab\_update

Prototype:	<pre>int btc_dptx_mst_vcptab_update(BYTE tx_idx)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx_syslib.h >
Return:	None
Parameters:	tx_idx—Source instance index (0 - 3)
Description:	This function generates an ACT sequence and then takes into use the current local VC Payload ID Table.
Example:	<code>btc_dptx_mst_vcptab_update(0);</code>

## 9.66. btc\_dptxll\_syslib API Reference

This section provides an alphabetically ordered list of all functions in the DisplayPort source link layer system library (`btc_dptxll_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

## 9.67. btc\_dptxll\_hpd\_change

Prototype:	<pre>int btc_dptxll_hpd_change( BYTE tx_idx, unsigned int asserted)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< <code>btc_dptxll_syslib.h</code> >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>• <code>tx_idx</code>—Source instance index (0 - 3)</li> <li>• <code>asserted</code>—0 = HPD is now deasserted, 1 = HPD is now asserted</li> </ul>
Description:	This function handles an HPD stable status change (not <code>HPD_IRQ</code> ) and must be invoked after every HPD stable logical status change.
Example:	<code>btc_dptxll_hpd_change(0,1);</code>

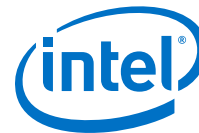
## 9.68. btc\_dptxll\_hpd\_irq

Prototype:	<pre>int btc_dptxll_hpd_irq(BYTE tx_idx)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< <code>btc_dptxll_syslib.h</code> >
Return:	0 = success, 1 = fail
Parameters:	<code>tx_idx</code> —Source instance index (0 - 3)
Description:	This function handles an <code>HPD_IRQ</code> . Must be invoked every time an <code>HPD_IRQ</code> is detected.
Example:	<code>btc_dptxll_hpd_irq(0);</code>

When `btc_dptxll_hpd_change(1)` is invoked and an MST capable DisplayPort sink device is detected, the topology discovery process is started automatically.

The process performs the following steps:





- Assigns a new GUID to sinks without one.
- Traverses all the topology connected through the DisplayPort sink.
- Adds each port found to the list of discovered ports.
- For each output port with plug status asserted and messaging capabilities, collects full and available PBN.
- Clears the sink VCP ID Table.
- Enables MST framing.

### 9.69. btc\_dptxll\_mst\_cmp\_ports

Prototype:	<pre>int btc_dptxll_mst_cmp_ports( BTC_RAD *A_RAD, BYTE A_port_number, , BTC_RAD *B_RAD, BYTE B_port_number)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptxll_syslib.h >
Return:	0 = A same as B, 1 = B is a child of A, -1 = B is not related to A
Parameters:	<ul style="list-style-type: none"> <li>• A_RAD—MST Relatives Address of port A device</li> <li>• A_port_number—Output port A number</li> <li>• B_RAD—MST Relatives Address of port B device</li> <li>• B_port_number—Output port B number</li> </ul>
Description:	This function compares port A and port B and checks if A is B's parent.
Example:	<code>btc_dptxll_mst_cmp_ports(&amp;RAD_A, 1, &amp;RAD_B, 8);</code>

### 9.70. btc\_dptxll\_mst\_edid\_read\_rep

Prototype:	<pre>int btc_dptxll_mst_edid_read_rep( BYTE tx_idx, BYTE **edid_data)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptxll_syslib.h >
Return:	0 = success, 1 = fail, 2 = not ready
Parameters:	<ul style="list-style-type: none"> <li>• tx_idx—Source instance index (0 - 3)</li> <li>• edid_data—Pointer to a 512 byte memory block internal to the system library</li> </ul>
Description:	This function returns the last EDID read data started by invoking <code>btc_dptxll_mst_edid_read_req()</code> . Call this function until either '0' or '1' is returned. '2' is returned when the operation has not yet completed. When '0' is returned, <code>edid_data</code> is set to the (unique) EDID data buffer internal to the system library. The invoking application is supposed to make a copy of the data, if needed.
Example:	<code>btc_dptxll_mst_edid_read_rep(0, p_data);</code>

### 9.71. btc\_dptxll\_mst\_edid\_read\_req

Prototype:	<pre>int btc_dptxll_mst_edid_read_req( BYTE tx_idx, BTC_RAD *device_RAD, BYTE port_number)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptxll_syslib.h >
Return:	0 = success, 1 = busy
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>device_RAD—MST Relative Address of the device</li> <li>port_number—Output port number</li> </ul>
Description:	Starts reading of a port's EDID. '1' is returned when a previous EDID read is still ongoing.
Example:	<code>btc_dptxll_mst_edid_read_req(0, &amp;aRAD, 9);</code>

### 9.72. btc\_dptxll\_mst\_get\_device\_ports

Prototype:	<pre>int btc_dptxll_mst_get_device_ports( BYTE tx_idx, BTC_MST_DEVPOR *port_list, BYTE *num_of_ports)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptxll_syslib.h >
Return:	0 = success, 1 = fail, 2 = not ready
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>port_list—List of discovered ports</li> <li>num_of_ports—Number of ports discovered</li> </ul>
Description:	This function returns the list of device ports found by the last topology discovery process. Call this function until either '0' or '1' is returned. '2' is returned when the operation has not yet completed.
Example:	<code>btc_dptxll_mst_get_device_ports(0, &amp;dev_ports, &amp;num_of_ports);</code>

The topology discovery process starts automatically after an MST capable DisplayPort sink device is connected and function `btc_dptxll_hpd_change(x, 1)` is invoked or when `btc_dptxll_mst_topology_discover()` is invoked.

### 9.73. btc\_dptxll\_mst\_set\_csn\_callback

Prototype:	<pre>int btc_dptxll_mst_set_csn_callback( BYTE tx_idx, BTC_MST_CSN_CALLBACK *cback)</pre>
Thread-safe:	No
<i>continued...</i>	



Available from ISR:	No
Include:	< btc_dptx11_syslib.h >
Return:	0 = success, 1 = fail, 2 = not ready
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>cback—Pointer to user callback function</li> </ul>
Description:	This function sets the CONNECTION_STATUS_NOTIFY user callback.
Example:	<code>btc_dptx11_mst_set_csn_callback(0,csn_handler);</code>

This function can be invoked right after `btc_dptx11_syslib_init()` by the user application to define a user-provided callback function handling received CONNECTION\_STATUS\_NOTIFY UP\_REQ MST messages.

When the user application invokes `btc_dptx11_syslib_monitor()`, if a CONNECTION\_STATUS\_NOTIFY has been received, the system library will invoke the user defined callback.

## 9.74. btc\_dptx11\_mst\_topology\_discover

Prototype:	<pre>int btc_dptx11_mst_topology_discover(     BYTE tx_idx,     BTC_RAD *device_RAD)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx11_syslib.h >
Return:	0 = success, 1 = busy
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>device_RAD—MST Relative Address of the device to start topology discovery from</li> </ul>
Description:	This function starts topology discovery.
Example:	<code>btc_dptx11_mst_topology_discover(0,&amp;RAD);</code>

The process performs the following steps:

- Assigns a new GUID to sinks without one.
- Traverses all the topology connected through `device_RAD`.
- Adds each port found to the list of discovered ports.
- For each output port with plug status asserted and messaging capabilities, collects full and available PBN.

## 9.75. btc\_dptx11\_stream\_allocate\_rep

Prototype:	<pre>int btc_dptx11_stream_allocate_rep(BYTE tx_idx)</pre>
Thread-safe:	No
<i>continued...</i>	

Available from ISR:	No
Include:	< btc_dptxll_syslib.h >
Return:	0 = success, 1 = fail, 2 = not ready
Parameters:	tx_idx—Source instance index (0 - 3)
Description:	This function checks if the last stream payload allocation was completed successfully. Call this function until either '0' or '1' is returned. '2' is returned when the operation has not yet completed.
Example:	btc_dptxll_stream_allocate_rep(0);

## 9.76. btc\_dptxll\_stream\_allocate\_req

Prototype:	<pre>int btc_dptxll_stream_allocate_req(     BYTE tx_idx,     BYTE strm_idx,     BTC_MST_DEVPOR *dev_port)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptxll_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>strm_idx—Stream index (0 - 3)</li> <li>dev_port—Device output port</li> </ul>
Description:	This function starts allocating a stream payload to a device port.
Example:	btc_dptxll_stream_allocate_req(0,0,aPort);

This function performs the following steps, for the given stream and device port:

- Sets the stream VCP ID.
- Adds the stream time slots to the local VCP ID Table.
- Adds the stream time slots to the Sink VCP ID Table.
- Generates activation (ACT) sequences until detected by the sink.

## 9.77. btc\_dptxll\_stream\_calc\_VCP\_size

Prototype:	<pre>int btc_dptxll_stream_calc_VCP_size(     BYTE tx_idx,     BYTE strm_idx)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptxll_syslib.h >
Return:	VCP size (0 = error)
<i>continued...</i>	



Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>strm_idx—Stream index (0 - 3)</li> </ul>
Description:	This function calculates the VCP size (number of time slots) needed to transmit stream strm_idx. btc_dptx11_stream_set_pixel_rate() must have been invoked before in order to define the data bandwidth required. The main link must be up when invoking btc_dptx11_stream_calc_VCP_size().
Example:	btc_dptx11_stream_calc_VCP_size(0,0);

This function calculates the following for the given stream:

- Stream VCP size
- Stream Average Time Slots per MTP
- Stream Max Target Average Time Slots per MTP

A returned VCP size exceeding 63 means that the main link current status (link bitrate and lane count) and the resulting available bandwidth are not enough to transport the stream.

### 9.78. btc\_dptx11\_stream\_delete\_rep

Prototype:	<pre>int btc_dptx11_stream_delete_rep(BYTE tx_idx)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx11_syslib.h >
Return:	0 = success, 1 = fail, 2 = not ready
Parameters:	tx_idx—Source instance index (0 - 3)
Description:	This function checks if the last stream payload deletion was completed successfully. Call this function until either '0' or '1' is returned. '2' is returned when the operation has not yet completed.
Example:	btc_dptx11_stream_delete_rep(0);

### 9.79. btc\_dptx11\_stream\_delete\_req

Prototype:	<pre>int btc_dptx11_stream_delete_req(   BYTE tx_idx,   BYTE strm_idx,   BTC_RAD *RAD,   BYTE port_number)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptx11_syslib.h >
Return:	0 = success, 1 = fail
<i>continued...</i>	

Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>strm_idx—Stream index (0 - 3)</li> <li>RAD—MST Relatives Address of the device</li> <li>port_number—Output port number</li> </ul>
Description:	This function starts deleting a stream payload from a device port.
Example:	<code>btc_dptxll_stream_delete_req(0,0,&amp;aRAD,8);</code>

This function performs the following steps, for the given stream and device port:

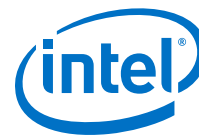
- Clears the stream VCP ID
- Deletes the stream time slots from the local VCP ID Table
- Deletes the stream time slots from the Sink VCP ID Table
- Generates activation (ACT) sequences until detected by the sink

## 9.80. btc\_dptxll\_stream\_get

Prototype:	<pre>BTC_STREAM *btc_dptxll_stream_get( BYTE tx_idx, BYTE strm_idx)</pre>
Thread-safe:	No
Available from ISR:	Yes
Include:	< btc_dptxll_syslib.h >
Return:	Pointer to the stream data
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>strm_idx—Stream index (0 - 3)</li> </ul>
Description:	This function returns a pointer to the stream info structure.
Example:	<code>btc_dptxll_stream_get(0,0);</code>

## 9.81. btc\_dptxll\_stream\_set\_color\_space

Prototype:	<pre>int btc_dptxll_stream_set_color_space( BYTE tx_idx, BYTE strm_idx, BYTE format, BYTE bpc, BYTE range, BYTE colorimetry)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptxLL_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>strm_idx—Stream index (0 - 3)</li> <li>format—0 = RGB; 1 = YCbCr 4:4:4; 2 = YCbCr 4:2:2, 3 = YCbCr 4:2:0</li> </ul>
<i>continued...</i>	



	<ul style="list-style-type: none"> <li>• <code>bpc</code>—Color depth: 0 = 6bpc; 1 = 8 bpc; 2 = 10 bpc; 3 = 12 bpc; 4 = 16 bpc</li> <li>• <code>range</code>—0 = VESA; 1 = CEA</li> <li>• <code>colorimetry</code>—0 = BT601-5; 1 = BT709-5; refer to Table 2-120 bit[3:0] in the <i>VESA DisplayPort Standard version 1.4</i> for all colorimetry support including BT.2020.</li> </ul>
Description:	This function sets the color space of a video stream.
Example:	<code>btc_dptx11_stream_set_color_space(0,0,0,1,0,0);</code>

## 9.82. `btc_dptx11_stream_set_pixel_rate`

Prototype:	<pre>int btc_dptx11_stream_set_pixel_rate(   BYTE tx_idx,   BYTE strm_idx,   unsigned int pixel_rate_kpps)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< <code>btc_dptx11_syslib.h</code> >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>• <code>tx_idx</code>—Source instance index (0 - 3)</li> <li>• <code>strm_idx</code>—Stream index (0 - 3)</li> <li>• <code>pixel_rate_kpps</code>—Pixel rate (kilopixels/sec)</li> </ul>
Description:	This function sets the pixel rate of a video stream. <code>btc_dptx11_stream_set_color_space()</code> must have been invoked before in order to define the number of bits/pixel required.
Example:	<code>btc_dptx11_stream_set_pixel_rate(0,0,154000);</code>

The function calculates the following for the given stream:

- Peak stream bandwidth
- Stream PBN value

## 9.83. `btc_dptx11_sw_ver`

Prototype:	<pre>void btc_dptx11_sw_ver(   BYTE *major,   BYTE *minor,   BYTE *rev)</pre>
Thread-safe:	Yes
Available from ISR:	Yes
Include:	< <code>btc_dptx11_syslib.h</code> >
Return:	-
<i>continued...</i>	

Parameters:	<ul style="list-style-type: none"> <li>major—Pointer to major version</li> <li>minor—Pointer to minor version</li> <li>rev—Pointer to revision)</li> </ul>
Description:	This function returns the version of the TX link layer system library.
Example:	<code>btc_dptxll_sw_ver(&amp;maj, &amp;min, &amp;rev);</code>

## 9.84. btc\_dptxll\_syslib\_add\_tx

Prototype:	<pre>int btc_dptxll_syslib_add_tx(     BYTE    tx_idx,     unsigned int max_link_rate,     unsigned int max_lane_count,     unsigned int tx_num_of_sources,     BYTE *edid_buf)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptxll_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	<ul style="list-style-type: none"> <li>tx_idx—Source instance index (0 - 3)</li> <li>max_link_rate—Maximum supported link rate. 0x06 = 1.62 Gbps; 0x0A = 2.70 Gbps; 0x14 = 5.40 Gbps</li> <li>max_lane_count—Maximum supported lane count. 1, 2 or 4</li> <li>tx_num_of_sources—Maximum number of supported MST stream source (1-4)</li> <li>edid_buf—Pointer to a 512 byte user-allocated EDID data buffer. Each source instance requires its own user-allocated EDID buffer to store the EDID of the connected sink</li> </ul>
Description:	This function declares a source (TX) instance to the system library. It should be invoked once for each existing source instance, starting from tx_idx = 0. After all sources have been declared, invoke <code>btc_dptxll_syslib_init ( )</code> .
Example:	<code>btc_dptxll_syslib_add_tx(0, 0x14, 4, DP_TX_SOURCE_MAX_NUM_OF_STREAMS, p_data);</code>

## 9.85. btc\_dptxll\_syslib\_init

Prototype:	<pre>int btc_dptxll_syslib_init(void)</pre>
Thread-safe:	No
Available from ISR:	No
Include:	< btc_dptxll_syslib.h >
Return:	0 = success, 1 = fail
Parameters:	None
Description:	Initializes the system library. Should be invoked just once after <code>btc_dptxll_syslib_add_tx ( )</code> .
Example:	<code>btc_dptxll_syslib_init ( );</code>





## 9.86. btc\_dptxll\_syslib\_monitor

Prototype:	<code>int btc_dptxll_syslib_monitor(void)</code>
Thread-safe:	No
Available from ISR:	No
Include:	<code>&lt; btc_dptxll_syslib.h &gt;</code>
Return:	0 = success, 1 = fail
Parameters:	None
Description:	This is a system library monitoring function. Must be invoked periodically at least every 50 ms.
Example:	<code>btc_dptxll_syslib_monitor();</code>

## 9.87. 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`
- `#define NIL 0xffffffff`

## 9.88. btc\_dprx\_syslib Supported DPCD Locations

[Sink-Supported DPCD Locations](#) on page 196 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`.

## 10. DisplayPort Source Register Map and DPCD Locations

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

Table 9–1 describes the notation used to describe the registers.

**Table 42. 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

### 10.1. Source General Registers

This section describes the general registers.

#### 10.1.1. 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`.

Setting `LANE_COUNT` to 00000 causes the transmitter to always send a logical zero (i.e., a constant voltage level). This function can be used as a surrogate for “power down” for link layer compliance testing.

Field `TX_LINK_RATE` drives the respective `tx_reconfig` port.

Address: 0x0000

Direction: RW

Reset: 0x00000000



Table 43. 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: <ul style="list-style-type: none"> <li>• 0 = disable</li> <li>• 1 = enable</li> </ul>
30	AUX_IRQ_EN	Enables an IRQ issued to the Nios II processor when an AUX channel transaction reply is received from the sink: <ul style="list-style-type: none"> <li>• 0 = disable</li> <li>• 1 = enable</li> </ul>
29	Unused	
28:21	TX_LINK_RATE	Main link rate expressed as multiples of 270 Mbps: <ul style="list-style-type: none"> <li>• 0x06 = 1.62 Gbps</li> <li>• 0x0a = 2.7 Gbps</li> <li>• 0x14 = 5.4 Gbps</li> <li>• 0x1e = 8.1 Gbps</li> </ul>
20	Reserved	Reserved
19	ENHANCED_FRAME	0 = Standard framing 1 = Enhanced framing
18:10	Unused	
9:5	LANE_COUNT	Lane count: <ul style="list-style-type: none"> <li>• 00000 = Reserved</li> <li>• 00001 = 1</li> <li>• 00010 = 2</li> <li>• 00100 = 4</li> </ul>
4	Unused	
3:0	TP	Current training pattern: <ul style="list-style-type: none"> <li>• 0000 = Normal video</li> <li>• 0001 = Training pattern 1 (D10.2)</li> <li>• 0010 = Training pattern 2</li> <li>• 0011 = Training pattern 3</li> <li>• 0111 = Training pattern 4</li> <li>• 0100 = Video idle pattern</li> <li>• 1001 = D10.2 test pattern (same as training pattern 1)</li> <li>• 1010 = Symbol error rate measurement pattern</li> <li>• 1011 = PRBS7</li> <li>• 1100 = 80-bit custom pattern</li> <li>• 1101 = HBR2 compliance test pattern (CP2520 pattern 1)</li> </ul>

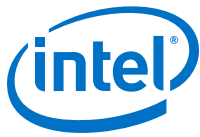
### 10.1.2. DPTX\_TX\_STATUS

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 44. 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): <ul style="list-style-type: none"><li>• 00 = No event</li><li>• 01 = HPD plug event (long HPD)</li><li>• 10 = HPD IRQ (short HPD)</li><li>• 11 = Reserved</li></ul>

### 10.1.3. DPTX\_TX\_VERSION

Address: 0x0002

Direction: RO

Reset: 0x00000000

**Table 45. DPTX\_TX\_VERSION Bits**

Bit	Bit Name	Function
31:24	Major	TX core major version number
23:16	Minor	TX core minor version number
15:0	Revision	TX core revision number

## 10.2. Source MSA Registers

The MSA registers are allocated at addresses:

- 0x0020 through 0x002f for Stream 0
- 0x0040 through 0x004f for Stream 1
- 0x0060 through 0x006f for Stream 2
- 0x0080 through 0x008f for Stream 3

*Note:* Only registers for Stream 0 are listed in the following sections.

### 10.2.1. DPTX0\_MSA\_MVID

Address: 0x0020

Direction: RO

Reset: 0x00000000

**Table 46. DPTX0\_MSA\_MVID Bits**

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

**10.2.2. DPTX0\_MSA\_NVID**

Address: 0x0021

Direction: RO

Reset: 0x00000000

**Table 47. DPTX0\_MSA\_NVID Bits**

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

**10.2.3. DPTX0\_MSA\_HTOTAL**

Address: 0x0022

Direction: RO

Reset: 0x00000000

*Note:* This register is RO if TX\_VIDEO\_IM\_ENABLE = 0 and RW if TX\_VIDEO\_IM\_ENABLE = 1.

**Table 48. DPTX0\_MSA\_HTOTAL Bits**

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

**10.2.4. DPTX0\_MSA\_VTOTAL**

Address: 0x0023

Direction: RO

Reset: 0x00000000

*Note:* This register is RO if TX\_VIDEO\_IM\_ENABLE = 0 and RW if TX\_VIDEO\_IM\_ENABLE = 1.

**Table 49. DPTX0\_MSA\_VTOTAL Bits**

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

### 10.2.5. DPTX0\_MSA\_HSP

Address: 0x0024

Direction: RO

Reset: 0x00000000

*Note:* This register is RO if TX\_VIDEO\_IM\_ENABLE = 0 and RW if TX\_VIDEO\_IM\_ENABLE = 1.

**Table 50. DPTX0\_MSA\_HSP Bits**

Bit	Bit Name	Function
31:1	Unused	
0	HSP	Main stream attribute horizontal sync polarity: <ul style="list-style-type: none"><li>• 0 = Positive</li><li>• 1 = Negative</li></ul>

### 10.2.6. DPTX0\_MSA\_HSW

Address: 0x0025

Direction: RO

Reset: 0x00000000

*Note:* This register is RO if TX\_VIDEO\_IM\_ENABLE = 0 and RW if TX\_VIDEO\_IM\_ENABLE = 1.

**Table 51. DPTX0\_MSA\_HSW Bits**

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

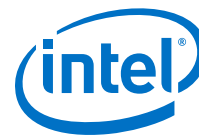
### 10.2.7. DPTX0\_MSA\_HSTART

Address: 0x0026

Direction: RO

Reset: 0x00000000

*Note:* This register is RO if TX\_VIDEO\_IM\_ENABLE = 0 and RW if TX\_VIDEO\_IM\_ENABLE = 1.

**Table 52. DPTX0\_MSA\_HSTART Bits**

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

**10.2.8. DPTX0\_MSA\_VSTART**

Address: 0x0027

Direction: RO

Reset: 0x00000000

*Note:* This register is RO if TX\_VIDEO\_IM\_ENABLE = 0 and RW if TX\_VIDEO\_IM\_ENABLE = 1.

**Table 53. DPTX0\_MSA\_VSTART Bits**

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

**10.2.9. DPTX0\_MSA\_VSP**

Address: 0x0028

Direction: RO

Reset: 0x00000000

*Note:* This register is RO if TX\_VIDEO\_IM\_ENABLE = 0 and RW if TX\_VIDEO\_IM\_ENABLE = 1.

**Table 54. DPTX0\_MSA\_VSP Bits**

Bit	Bit Name	Function
31:1	Unused	
0	VSP	Main stream attribute vertical sync polarity <ul style="list-style-type: none"> <li>0 = Positive</li> <li>1 = Negative</li> </ul>

**10.2.10. DPTX0\_MSA\_VSW**

Address: 0x0029

Direction: RO

Reset: 0x00000000

*Note:* This register is RO if TX\_VIDEO\_IM\_ENABLE = 0 and RW if TX\_VIDEO\_IM\_ENABLE = 1.

**Table 55. DPTX0\_MSA\_VSW Bits**

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

### 10.2.11. DPTX0\_MSA\_HWIDTH

Address: 0x002a

Direction: RO

Reset: 0x00000000

*Note:* This register is RO if TX\_VIDEO\_IM\_ENABLE = 0 and RW if TX\_VIDEO\_IM\_ENABLE = 1.

**Table 56. DPTX0\_MSA\_HWIDTH Bits**

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

### 10.2.12. DPTX0\_MSA\_VHEIGHT

Address: 0x002b

Direction: RO

Reset: 0x00000000

*Note:* This register is RO if TX\_VIDEO\_IM\_ENABLE = 0 and RW if TX\_VIDEO\_IM\_ENABLE = 1.

**Table 57. DPTX0\_MSA\_VHEIGHT Bits**

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

### 10.2.13. DPTX0\_MSA\_MISC0

Address: 0x002c

Direction: RO

Reset: 0x00000000



**Table 58. DPTX0\_MSA\_MISC0 Bits**

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

**10.2.14. DPTX0\_MSA\_MISC1**

Address: 0x002d

Direction: RO

Reset: 0x00000000

**Table 59. DPTX0\_MSA\_MISC1 Bits**

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

**10.2.15. DPTX0\_MSA\_COLOR**

Address: 0x002e

Direction: RW

Reset: 0x00000001

**Table 60. DPTX0\_MSA\_COLOR Bits**

Bit	Bit Name	Function
31:14	Unused	
13	USE_VSC_SDP	0 = use MISC0 1 = use VSC SDP <i>Note: If you configure this bit to use VSC SDP, refer to the VESA DisplayPort Standard version 1.4 for the VSC SDP Payload Pixel Encoding/Colorimetry Format. Y-Only and Raw format are not supported.</i>
12	DYNAMIC_RANGE	<ul style="list-style-type: none"> <li>0 = VESA (from 0 to maximum)</li> <li>1 = CEA range</li> </ul>
11:8	COLORIMETRY	<ul style="list-style-type: none"> <li>0000 = ITU-R BT601-5</li> <li>0001 = ITU-R BT709-5</li> </ul> <i>Note: Refer to Table 2-120 bit[3:0] in the VESA DisplayPort Standard version 1.4 for all colorimetry support including BT.2020.</i>

**continued...**



Bit	Bit Name	Function
7:4	ENCODING	<ul style="list-style-type: none"> <li>• 0000 = RGB</li> <li>• 0001 = YCbCr 4:4:4</li> <li>• 0010 = YCbCr 4:2:2</li> <li>• 0011 = YCbCr 4:2:0</li> </ul>
3	Unused	
2:0	BPC	Bits per pixel format <ul style="list-style-type: none"> <li>• 000 = 6 bpc</li> <li>• 001 = 8 bpc</li> <li>• 010 = 10 bpc</li> <li>• 011 = 12 bpc</li> <li>• 100 = 16 bpc</li> </ul>

### 10.2.16. DPTX0\_VBID

Address: 0x002f

Direction: RO

Reset: 0x00000000

**Table 61. DPTX0\_VBID Bits**

Bit	Bit Name	Function
31:8	Unused	
7	MSA_LOCK	0 = Input video timing unstable 1 = Input video timing stable
6:5	Unused	
4:0	VBID[4:0]	VB-ID flags (refer to the <i>VESA DisplayPort Standard</i> ).

## 10.3. Source Link PHY Control and Status

This section describes the registers for the PHY controls.

### 10.3.1. DPTX\_PRE\_VOLT0

These ports drive the respective tx\_rcfg\_vod and tx\_rcfg\_emp ports.

Address: 0x0010

Direction: RW

Reset: 0x00000000

**Table 62. DPTX\_PRE\_VOLT0 Bits**

Bit	Bit Name	Function
31:4	Unused	
3:2	PRE0	Pre-emphasis output on lane 0
1:0	VOLT0	Voltage swing output on lane 0



### 10.3.2. DPTX\_PRE\_VOLT1

These ports drive the respective `tx_rcfg_vod` and `tx_rcfg_emp` ports.

Address: 0x0011

Direction: RW

Reset: 0x00000000

**Table 63. 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

### 10.3.3. DPTX\_PRE\_VOLT2

These ports drive the respective `tx_rcfg_vod` and `tx_rcfg_emp` ports.

Address: 0x0012

Direction: RW

Reset: 0x00000000

**Table 64. 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

### 10.3.4. DPTX\_PRE\_VOLT3

These ports drive the respective `tx_rcfg_vod` and `tx_rcfg_emp` ports.

Address: 0x0013

Direction: RW

Reset: 0x00000000

**Table 65. 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

### 10.3.5. DPTX\_RECONFIG

RECONFIG\_ANALOG drives the tx\_analog\_reconfig\_req while RECONFIG\_LINKRATE drives tx\_reconfig\_req port. GXB\_BUSY is indicator of tx\_reconfig\_busy port.

Address: 0x0014

Direction: RW

Reset: 0x00000000

**Table 66. DPTX\_RECONFIG Bits**

Bit	Bit Name	Function
31	GXB_BUSY	Read-only flag where: <ul style="list-style-type: none"> <li>0 = Transceiver is not busy</li> <li>1 = Transceiver is busy</li> </ul>
30:2	Unused	
1	RECONFIG_LINKRATE	1 = Reconfigure the transceiver with the link rate in DPTX_TX_CONTROL (TX_LINK_RATE) When you set this bit to 1, it automatically clears (0) after one clock cycle.
0	RECONFIG_ANALOG	1 = Reconfigure transceiver with analog values in DPTX_PRE_VOLT0-3 When you set this bit to 1, it automatically clears (0) after one clock cycle.

### 10.3.6. DPTX\_TEST\_80BIT\_PATTERN1

Address: 0x0015

Direction: RW

Reset: 0x00000000

**Table 67. DPTX\_TEST\_80BIT\_PATTERN1 Bits**

Bit	Bit Name	Function
31:0	80BIT_PATTERN1	Bits 31:0 of the 80 bit custom pattern for PHY compliance test.

### 10.3.7. DPTX\_TEST\_80BIT\_PATTERN2

Address: 0x0016

Direction: RW

Reset: 0x00000000

**Table 68. DPTX\_TEST\_80BIT\_PATTERN2 Bits**

Bit	Bit Name	Function
31:0	80BIT_PATTERN2	Bits 63:32 of the 80 bit custom pattern for PHY compliance test.



### 10.3.8. DPTX\_TEST\_80BIT\_PATTERN3

Address: 0x0017

Direction: RW

Reset: 0x00000000

**Table 69. DPTX\_TEST\_80BIT\_PATTERN3 Bits**

Bit	Bit Name	Function
31:16	Unused	
15:0	80BIT_PATTERN3	Bits 79:64 of the 80 bit custom pattern for PHY compliance test.

## 10.4. Source 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 70. DPTX\_TIMESTAMP Bits**

Bit	Bit Name	Function
31:24	Unused	
23:0	TIMESTAMP	Free-running counter value (1 tick equals 100 $\mu$ s)

## 10.5. Source CRC Registers

The CRC registers are allocated at addresses:

- 0x0030 through 0x0032 for Stream 0
- 0x0050 through 0x0052 for Stream 1
- 0x0070 through 0x0072 for Stream 2
- 0x0090 through 0x0092 for Stream 3

*Note:* Only registers for Stream 0 are listed in the following sections.

DPTX0\_CRC\_R

Address: 0x0030

Direction: RO

Reset: 0x00000000

**Table 71. DPTX0\_CRC\_R Bits**

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

DPTX0\_CRC\_G

Address: 0x0031

Direction: RO

Reset: 0x00000000

**Table 72. DPTX0\_CRC\_G Bits**

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

DPTX0\_CRC\_B

Address: 0x0032

Direction: RO

Reset: 0x00000000

**Table 73. DPTX0\_CRC\_B Bits**

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

## 10.6. Source Audio Registers

The Audio registers are allocated at addresses:

- 0x0033 for Stream 0
- 0x0053 for Stream 1
- 0x0073 for Stream 2
- 0x0093 for Stream 3

*Note:* Only registers for Stream 0 are listed in the following sections.

Address: 0x002f

Direction: RW

Reset: The maximum number of channels supported minus 1 (0x00000000 – 0x00000007)

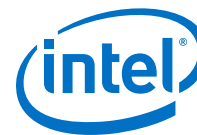


Table 74. DPTX0\_AUD\_CONTROL Bits

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 (LFEBPL, 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

## 10.7. Source MST Registers

DPTX\_MST\_CONTROL1

Address: 0x00a0

Direction: RW

Table 75. DPTX\_MST\_CONTROL1 Bits

Bit	Bit Name	Function
31	VCPTAB_UPD_FORCE	This flag always reads back at 0. 1 = Force VC payload ID table update
30	VCPTAB_UPD_REQ	This flag always reads back at 0. 1 = Request for VC payload ID table update
29:20	Unused	
19:16	VCP_ID3	VC payload ID for Stream 3
15:12	VCP_ID2	VC payload ID for Stream 2
11:8	VCP_ID1	VC payload ID for Stream 1
7:4	VCP_ID0	VC payload ID for Stream 0
3:1	Unused	
0	MST_EN	Enable or disable MST <ul style="list-style-type: none"> <li>1 = MST framing</li> <li>0 = SST framing</li> </ul>

When you assert VCPTAB\_UPD\_FORCE, the source forces the VC payload table contained in DPTX\_MST\_VCPTAB0 through DPTX\_MST\_VCPTAB7 to be taken immediately into use. No ACT sequence is generated in this case.



When you assert VCPTAB\_UPD\_REQ, the source requests to generate an ACT sequence and after that, use the VC payload table contained in DPTX\_MST\_VCPTAB0 through DPTX\_MST\_VCPTAB7.

### 10.7.1. DPTX\_MST\_VCPTAB0

VC Payload ID Table

Address: 0x00a2

Direction: RW

Reset: 0x00000000

**Table 76. DPTX\_MST\_VCPTAB0 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT7	VC payload ID for slot 7
27:24	VCPSLOT6	VC payload ID for slot 6
23:20	VCPSLOT5	VC payload ID for slot 5
19:16	VCPSLOT4	VC payload ID for slot 4
15:12	VCPSLOT3	VC payload ID for slot 3
11:8	VCPSLOT2	VC payload ID for slot 2
7:4	VCPSLOT1	VC payload ID for slot 1
3:0	Reserved	Reserved

### 10.7.2. DPTX\_MST\_VCPTAB1

VC Payload ID Table

Address: 0x00a3

Direction: RW

Reset: 0x00000000

**Table 77. DPTX\_MST\_VCPTAB1 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT15	VC payload ID for slot 15
27:24	VCPSLOT14	VC payload ID for slot 14
23:20	VCPSLOT13	VC payload ID for slot 13
19:16	VCPSLOT12	VC payload ID for slot 12
15:12	VCPSLOT11	VC payload ID for slot 11
11:8	VCPSLOT10	VC payload ID for slot 10
7:4	VCPSLOT9	VC payload ID for slot 9
3:0	VCPSLOT8	VC payload ID for slot 8





### 10.7.3. DPTX\_MST\_VCPTAB2

VC Payload ID Table

Address: 0x00a4

Direction: RW

Reset: 0x00000000

**Table 78. DPTX\_MST\_VCPTAB2 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT23	VC payload ID for slot 23
27:24	VCPSLOT22	VC payload ID for slot 22
23:20	VCPSLOT21	VC payload ID for slot 21
19:16	VCPSLOT20	VC payload ID for slot 20
15:12	VCPSLOT19	VC payload ID for slot 19
11:8	VCPSLOT18	VC payload ID for slot 18
7:4	VCPSLOT17	VC payload ID for slot 17
3:0	VCPSLOT16	VC payload ID for slot 16

### 10.7.4. DPTX\_MST\_VCPTAB3

VC Payload ID Table

Address: 0x00a5

Direction: RW

Reset: 0x00000000

**Table 79. DPTX\_MST\_VCPTAB3 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT31	VC payload ID for slot 31
27:24	VCPSLOT30	VC payload ID for slot 30
23:20	VCPSLOT29	VC payload ID for slot 29
19:16	VCPSLOT28	VC payload ID for slot 28
15:12	VCPSLOT27	VC payload ID for slot 27
11:8	VCPSLOT26	VC payload ID for slot 26
7:4	VCPSLOT25	VC payload ID for slot 25
3:0	VCPSLOT24	VC payload ID for slot 24

### 10.7.5. DPTX\_MST\_VCPTAB4

VC Payload ID Table



Address: 0x00a6

Direction: RW

Reset: 0x00000000

**Table 80. DPTX\_MST\_VCPTAB4 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT39	VC payload ID for slot 39
27:24	VCPSLOT38	VC payload ID for slot 38
23:20	VCPSLOT37	VC payload ID for slot 37
19:16	VCPSLOT36	VC payload ID for slot 36
15:12	VCPSLOT35	VC payload ID for slot 35
11:8	VCPSLOT34	VC payload ID for slot 34
7:4	VCPSLOT33	VC payload ID for slot 33
3:0	VCPSLOT32	VC payload ID for slot 32

### 10.7.6. DPTX\_MST\_VCPTAB5

VC Payload ID Table

Address: 0x00a7

Direction: RW

Reset: 0x00000000

**Table 81. DPTX\_MST\_VCPTAB5 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT47	VC payload ID for slot 47
27:24	VCPSLOT46	VC payload ID for slot 46
23:20	VCPSLOT45	VC payload ID for slot 45
19:16	VCPSLOT44	VC payload ID for slot 44
15:12	VCPSLOT43	VC payload ID for slot 43
11:8	VCPSLOT42	VC payload ID for slot 42
7:4	VCPSLOT41	VC payload ID for slot 41
3:0	VCPSLOT40	VC payload ID for slot 40

### 10.7.7. DPTX\_MST\_VCPTAB6

VC Payload ID Table

Address: 0x00a8

Direction: RW



Reset: 0x00000000

**Table 82. DPTX\_MST\_VCPTAB6 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT55	VC payload ID for slot 55
27:24	VCPSLOT54	VC payload ID for slot 54
23:20	VCPSLOT53	VC payload ID for slot 53
19:16	VCPSLOT52	VC payload ID for slot 52
15:12	VCPSLOT51	VC payload ID for slot 51
11:8	VCPSLOT50	VC payload ID for slot 50
7:4	VCPSLOT49	VC payload ID for slot 49
3:0	VCPSLOT48	VC payload ID for slot 48

### 10.7.8. DPTX\_MST\_VCPTAB7

VC Payload ID Table

Address: 0x00a9

Direction: RW

Reset: 0x00000000

**Table 83. DPTX\_MST\_VCPTAB7 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT63	VC payload ID for slot 63
27:24	VCPSLOT62	VC payload ID for slot 62
23:20	VCPSLOT61	VC payload ID for slot 61
19:16	VCPSLOT60	VC payload ID for slot 60
15:12	VCPSLOT59	VC payload ID for slot 59
11:8	VCPSLOT58	VC payload ID for slot 58
7:4	VCPSLOT57	VC payload ID for slot 57
3:0	VCPSLOT56	VC payload ID for slot 56

### 10.7.9. DPTX\_MST\_TAVG\_TS

Target Average Time Slots

Address: 0x00aa

Direction: RW

Reset: 0x40404040

**Table 84. DPTX\_MST\_TAVG\_TS Bits**

Bit	Bit Name	Function
31	Unused	
30:24	TAVG_TS3	Target Average Time Slots for Stream 3
23	Unused	
22:16	TAVG_TS2	Target Average Time Slots for Stream 2
15	Unused	
14:8	TAVG_TS1	Target Average Time Slots for Stream 1
7	Unused	
6:0	TAVG_TS0	Target Average Time Slots for Stream 0

TAVG\_TS<sub>x</sub> is expressed as the fractional part of the number of time slots per MTU occupied by Stream x times 64; assuming the allocated time slots are the ceiling of this number. For example, if 4.7 time slots/MTU are occupied (5 time slots/MTU are allocated in the VCP ID table).

$$TAVG\_TS_x = \text{CEIL}(\text{FRAC}(4.7) * 64) = \text{CEIL}(0.7 * 64) = 45$$

The achieved precision for Target Average Time Slots regulation is  $1/64 = 0.015625$ .

If TAVG\_TS<sub>x</sub> is set to a value greater than 63, VCP fill is sent to each allocated time slot.

## 10.8. Source AUX Controller Interface

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

### 10.8.1. 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 85. 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).

### 10.8.2. DPTX\_AUX\_COMMAND

Address: 0x0101

Direction: RW

Reset: 0x00000000

**Table 86. DPTX\_AUX\_COMMAND Bits**

Bit	Bit Name	Function
31:8	Unused	
7:4	COMMAND	AUX transaction command for the next request or received in the most recent reply (refer to the <i>VESA DisplayPort Standard</i> for details). Reading of this register clears MSG_READY and LENGTH in DPTX_AUX_CONTROL register.
3:0	BYTE	Transaction address[19:16] for the next request.

### 10.8.3. DPTX\_AUX\_BYTE0

AUX Transaction Byte 0 Register.

Address: 0x0102

Direction: RW

Reset: 0x00000000

**Table 87. 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



#### 10.8.4. DPTX\_AUX\_BYTE1

AUX Transaction Byte 1 Register.

Address: 0x0103

Direction: RW

Reset: 0x00000000

**Table 88. DPTX\_AUX\_BYTE1 Bits**

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

#### 10.8.5. DPTX\_AUX\_BYTE2

AUX Transaction Byte 2 Register.

Address: 0x0104

Direction: RW

Reset: 0x00000000

**Table 89. 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 <i>VESA DisplayPort Standard</i> for details)

#### 10.8.6. DPTX\_AUX\_BYTE3

AUX Transaction Byte 3 Register.

Address: 0x0105

Direction: RW

Reset: 0x00000000

**Table 90. 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



### 10.8.7. DPTX\_AUX\_BYTE4

AUX Transaction Byte 4 Register.

Address: 0x0106

Direction: RW

Reset: 0x00000000

**Table 91. 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

### 10.8.8. DPTX\_AUX\_BYTE5

AUX Transaction Byte 5 Register.

Address: 0x0107

Direction: RW

Reset: 0x00000000

**Table 92. 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

### 10.8.9. DPTX\_AUX\_BYTE6

AUX Transaction Byte 6 Register.

Address: 0x0108

Direction: RW

Reset: 0x00000000

**Table 93. 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



### 10.8.10. DPTX\_AUX\_BYTE7

AUX Transaction Byte 7 Register.

Address: 0x0109

Direction: RW

Reset: 0x00000000

**Table 94. 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

### 10.8.11. DPTX\_AUX\_BYTE8

AUX Transaction Byte 8 Register.

Address: 0x010a

Direction: RW

Reset: 0x00000000

**Table 95. 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

### 10.8.12. DPTX\_AUX\_BYTE9

AUX Transaction Byte 9 Register.

Address: 0x010b

Direction: RW

Reset: 0x00000000

**Table 96. 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





### 10.8.13. DPTX\_AUX\_BYTE10

AUX Transaction Byte 10 Register.

Address: 0x010c

Direction: RW

Reset: 0x00000000

**Table 97. 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

### 10.8.14. DPTX\_AUX\_BYTE11

AUX Transaction Byte 11 Register.

Address: 0x010d

Direction: RW

Reset: 0x00000000

**Table 98. 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

### 10.8.15. DPTX\_AUX\_BYTE12

AUX Transaction Byte 12 Register.

Address: 0x010e

Direction: RW

Reset: 0x00000000

**Table 99. 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



### 10.8.16. DPTX\_AUX\_BYTE13

AUX Transaction Byte 13 Register.

Address: 0x010f

Direction: RW

Reset: 0x00000000

**Table 100. 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

### 10.8.17. DPTX\_AUX\_BYTE14

AUX Transaction Byte 14 Register.

Address: 0x0110

Direction: RW

Reset: 0x00000000

**Table 101. 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

### 10.8.18. DPTX\_AUX\_BYTE15

AUX Transaction Byte 15 Register.

Address: 0x0111

Direction: RW

Reset: 0x00000000

**Table 102. 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



### 10.8.19. DPTX\_AUX\_BYTE16

AUX Transaction Byte 16 Register.

Address: 0x0112

Direction: RW

Reset: 0x00000000

**Table 103. DPTX\_AUX\_BYTE16 Bits**

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

### 10.8.20. DPTX\_AUX\_BYTE17

AUX Transaction Byte 17 Register.

Address: 0x0113

Direction: RW

Reset: 0x00000000

**Table 104. DPTX\_AUX\_BYTE17 Bits**

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

### 10.8.21. DPTX\_AUX\_BYTE18

AUX Transaction Byte 18 Register.

Address: 0x0114

Direction: RW

Reset: 0x00000000

**Table 105. DPTX\_AUX\_BYTE18 Bits**

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

### 10.8.22. DPTX\_AUX\_RESET

Address: 0x0117

Direction: WO



Reset: 0x00000000

**Table 106. DPTX\_AUX\_RESET Bits**

Bit	Bit Name	Function
31:1	Unused	
0	CLEAR	Asserting CLEAR resets the AUX Controller state machine: <ul style="list-style-type: none"> <li>• 0 = No action</li> <li>• 1 = AUX Controller reset</li> </ul>

## 10.9. Source-Supported DPCD Locations

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

**Table 107. DPCD Locations**

Location Name	Address
DPCD_REV	0x0000
MAX_LINK_RATE	0x0001
MAX_LANE_COUNT	0x0002
TRAINING_AUX_RD_INTERVAL	0x000E
MST_CAP	0x0021
GUID	0x0030
LINK_BW_SET	0x0100
LANE_COUNT_SET	0x0101
TRAINING_PATTERN_SET	0x0102
TRAINING_LANE0_SET	0x0103
TRAINING_LANE1_SET	0x0104
TRAINING_LANE2_SET	0x0105
TRAINING_LANE3_SET	0x0106
DOWNSPREAD_CTRL	0x0107
MSTM_CTRL	0x0111
PAYLOAD_ALLOCATE_SET	0x01C0
PAYLOAD_ALLOCATE_START_TIME_SLOT	0x01C1
PAYLOAD_ALLOCATE_TIME_SLOT_COUNT	0x01C2
SINK_COUNT	0x0200
DEVICE_SERVICE_IRQ_VECTOR	0x0201
LANE0_1_STATUS	0x0202
LANE2_3_STATUS	0x0203
LANE_ALIGN_STATUS_UPDATED	0x0204

*continued...*

## 10. DisplayPort Source Register Map and DPCD Locations

UG-01131 | 2019.04.01



Location Name	Address
SINK_STATUS	0x0205
ADJUST_REQUEST_LANE0_1	0x0206
ADJUST_REQUEST_LANE2_3	0x0207
SYMBOL_ERROR_COUNT_LANE0	0x0210
SYMBOL_ERROR_COUNT_LANE1	0x0212
SYMBOL_ERROR_COUNT_LANE2	0x0214
SYMBOL_ERROR_COUNT_LANE3	0x0216
TEST_REQUEST	0x0218
TEST_LINK_RATE	0x0219
TEST_LANE_COUNT	0x0220
PHY_TEST_PATTERN	0x0248
TEST_80BIT_CUSTOM_PATTERN (0x0250 to 0x0259)	0x0250
TEST_RESPONSE	0x0260
TEST_EDID_CHECKSUM	0x0261
PAYLOAD_TABLE_UPDATE_STATUS	0x02C0
VC_PAYLOAD_ID_SLOT_1 (0x02C1 to 0x02FF)	0x02C1
SET_POWER_STATE	0x0600
DOWN_REQ (0x1000 to 0x102F)	0x1000
UP_REQ (0x1200 to 0x122F)	0x1200
DOWN_REQ (0x1400 to 0x142F)	0x1400
UP_REQ (0x1600 to 0x162F)	0x1600

## 11. DisplayPort Sink Register Map and DPCD Locations

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 108. 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

### 11.1. Sink General Registers

This section describes the general registers.

#### 11.1.1. DPRX\_RX\_CONTROL

RECONFIG\_LINKRATE drives the rx\_reconfig\_req. RX\_LINK\_RATE drives rx\_link\_rate.

Address: 0x0000

Direction: RW

Reset: 0x00000000

**Table 109. DPRX\_RX\_CONTROL Bits**

Bit	Bit Name	Function
31:30	Unused	
29	LQA_ACTIVE	<ul style="list-style-type: none"> <li>0 = Link Quality Analysis (also known as Post-Link Training Adjust Request) not used</li> <li>1 = Link Quality Analysis (also known as Post-Link Training Adjust Request) in progress</li> </ul>
28	BLACK_VIDEO_EN	<ul style="list-style-type: none"> <li>0 = Stream 0 receives video output normally</li> <li>1 = Stream 0 receives video output with all colors set to black</li> </ul>
27:24	Unused	

*continued...*

## 11. DisplayPort Sink Register Map and DPCD Locations

UG-01131 | 2019.04.01



Bit	Bit Name	Function
23:16	RX_LINK_RATE	Main link rate expressed as multiples of 270 Mbps: <ul style="list-style-type: none"> <li>• 0x06 = 1.62 Gbps</li> <li>• 0x0a = 2.7 Gbps</li> <li>• 0x14 = 5.4 Gbps</li> <li>• 0x1e = 8.1 Gbps</li> </ul>
15:14	Unused	
13	RECONFIG_LINKRATE	This flag always reads back at 0. 1 = Reconfigure the transceiver with link rate RX_LINK_RATE
12	Unused	
11	GXB_RESET	<ul style="list-style-type: none"> <li>• 0 = Sink transceiver enabled</li> <li>• 1 = Sink transceiver reset</li> </ul>
10:8	TP	Current training pattern: <ul style="list-style-type: none"> <li>• 000 = Normal video</li> <li>• 001 = Training pattern 1</li> <li>• 010 = Training pattern 2</li> <li>• 011 = Training pattern 3</li> <li>• 111 = Training pattern 4</li> </ul>
7	SCRAMBLER_DISABLE	<ul style="list-style-type: none"> <li>• 0 = Scrambler enabled</li> <li>• 1 = Scrambler disabled</li> </ul>
6:5	Unused	
4:0	LANE_COUNT	Lane count: <ul style="list-style-type: none"> <li>• 00001 = 1</li> <li>• 00010 = 2</li> <li>• 00100 = 4</li> </ul>

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

Direction: RO

**Table 110. DPRX\_RX\_CONTROL Bits (Non-GPU Mode)**

Bit	Bit Name	Function
31:24	Unused	
23:16	RX_LINK_RATE	Main link rate expressed as multiples of 270 Mbps: <ul style="list-style-type: none"> <li>• 0x06 = 1.62 Gbps</li> <li>• 0x0a = 2.7 Gbps</li> <li>• 0x14 = 5.4 Gbps</li> </ul>
15:5	Unused	
4:0	LANE_COUNT	Lane count: <ul style="list-style-type: none"> <li>• 00001 = 1</li> <li>• 00010 = 2</li> <li>• 00100 = 4</li> </ul>

### 11.1.2. DPRX\_RX\_STATUS

GXB\_BUSY connects to the rx\_reconfig\_busy input port.

Address: 0x0001



Direction: CRO

Reset: 0x00000000

**Table 111. 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
15:9	Unused	
8	INTERLANE_ALIGN	0 = Inter-lane alignment not achieved 1 = Inter-lane alignment achieved
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)

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

Direction: RO

**Table 112. DPRX\_RX\_STATUS Bits (Non-GPU 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
15:9	Unused	
8	INTERLANE_ALIGN	0 = Inter-lane alignment not achieved 1 = Inter-lane alignment achieved
<i>continued...</i>		





Bit	Bit Name	Function
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)

### 11.1.3. DPRX\_BER\_CONTROL

Address: 0x0002

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 113. DPRX\_BER\_CONTROL Bits**

Bit	Bit Name	Function
31:28	Unused	
27	RSTI3	Writing this bit at 1 resets lane 3 bit-error counter in register DPRX_BER_CNTI1. Always reads as '0'.
26	RSTI2	Writing this bit at 1 resets lane 2 bit-error counter in register DPRX_BER_CNTI1. Always reads as '0'.
25	RSTI1	Writing this bit at 1 resets lane 1 bit-error counter in register DPRX_BER_CNTI0. Always reads as '0'.
24	RSTI0	Writing this bit at 1 resets lane 0 bit-error counter in register DPRX_BER_CNTI0. Always reads as '0'.
23	Unused	
22:21	PHY_SINK_TEST_LANE_SEL	Specifies the lane that is being tested, when PHY_SINK_TEST_LANE_EN is 1, <ul style="list-style-type: none"> <li>00 = Lane 0</li> <li>01 = Lane 1</li> <li>10 = Lane 2</li> <li>11 = Lane 3</li> </ul>

*continued...*



Bit	Bit Name	Function
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: <ul style="list-style-type: none"><li>• 000 = No test pattern (normal mode)</li><li>• 011 = PRBS7</li><li>• 101 = HBR2Compliance EYE pattern</li></ul>
10:8	PATT2	Pattern selection for lane 2: <ul style="list-style-type: none"><li>• 000 = No test pattern (normal mode)</li><li>• 011 = PRBS7</li><li>• 101 = HBR2 Compliance EYE pattern</li></ul>
7:5	PATT1	Pattern selection for lane 1: <ul style="list-style-type: none"><li>• 000 = No test pattern (normal mode)</li><li>• 011 = PRBS7</li><li>• 101 = HBR2 Compliance EYE pattern</li></ul>
4:2	PATT0	Pattern selection for lane 0: <ul style="list-style-type: none"><li>• 000 = No test pattern (normal mode)</li><li>• 011 = PRBS7</li><li>• 101 = HBR2 Compliance EYE pattern</li></ul>
1:0	CNTSEL	Count selection: <ul style="list-style-type: none"><li>• 00 = Disparity and code error counts</li><li>• 01 = Disparity error counts</li><li>• 10 = Code error counts</li><li>• 11 = Reserved</li></ul>

#### 11.1.4. DPRX\_BER\_CNT0

These registers are exposed in DPCD locations `SYMBOL_ERROR_COUNT_LANE0` and `SYMBOL_ERROR_COUNT_LANE1`.

Address: 0x0003

Direction: RO

Reset: 0x00000000

**Table 114. DPRX\_RX\_STATUS 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

### 11.1.5. DPRX\_BER\_CNT1

These registers are exposed in DPCD locations SYMBOL\_ERROR\_COUNT\_LANE2 and SYMBOL\_ERROR\_COUNT\_LANE3.

Address: 0x0004

Direction: RO

Reset: 0x00000000

**Table 115. DPRX\_RX\_STATUS 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

## 11.2. Sink 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).

DPRX\_TIMESTAMP

Address: 0x0005

Direction: RO

Reset: 0x00000000

**Table 116. DPRX\_TIMESTAMP Bits**

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

## 11.3. Sink Bit-Error Counters



### 11.3.1. DPRX\_BER\_CNTI0

Internal bit-error counters for lane 0 and lane 1.

Address: 0x0006

Direction: RO

Reset: 0x00000000

**Table 117. DPRX\_BER\_CNTI0 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

These registers are meant for internal use and are not exposed in the DPCD.

### 11.3.2. DPRX\_BER\_CNTI1

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

Address: 0x0007

Direction: RO

Reset: 0x00000000

**Table 118. DPRX\_BER\_CNTI1 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

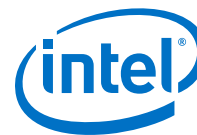
These registers are meant for internal use and are not exposed in the DPCD.

## 11.4. Sink MSA Registers

The MSA registers are allocated at addresses:

- 0x0020 through 0x002f for Stream 0
- 0x0040 through 0x004f for Stream 1
- 0x0060 through 0x006f for Stream 2
- 0x0080 through 0x008f for Stream 3

*Note:* Only registers for Stream 0 are listed in the following sections. Registers for Stream 0 are also available in non-GPU mode.



## 11. DisplayPort Sink Register Map and DPCD Locations

UG-01131 | 2019.04.01

### 11.4.1. DPRX0\_MSA\_MVID

Address: 0x0020

Direction: RO

Reset: 0x00000000

**Table 119. DPRX0\_MSA\_MVID Bits**

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

### 11.4.2. DPRX0\_MSA\_NVID

Address: 0x0021

Direction: RO

Reset: 0x00000000

**Table 120. DPRX0\_MSA\_NVID Bits**

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

### 11.4.3. DPRX0\_MSA\_HTOTAL

Address: 0x0022

Direction: RO

Reset: 0x00000000

**Table 121. DPRX0\_MSA\_HTOTAL Bits**

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

### 11.4.4. DPRX0\_MSA\_VTOTAL

Address: 0x0023

Direction: RO

Reset: 0x00000000

**Table 122. DPRX0\_MSA\_VTOTAL Bits**

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

### 11.4.5. DPRX0\_MSA\_HSP

MSA horizontal synchronization polarity register, DPRX0\_MSA\_HSP.

Address: 0x0024

Direction: RO

Reset: 0x00000000

**Table 123. DPRX0\_MSA\_HSP Bits**

Bit	Bit Name	Function
31:1	Unused	
0	HSP	Main stream attribute horizontal synchronization polarity <ul style="list-style-type: none"> <li>• 0 = Positive</li> <li>• 1 = Negative</li> </ul>

### 11.4.6. DPRX0\_MSA\_HSW

MSA horizontal synchronization width register, DPRX0\_MSA\_HSW.

Address: 0x0025

Direction: RO

Reset: 0x00000000

**Table 124. DPRX0\_MSA\_HSW Bits**

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

### 11.4.7. DPRX0\_MSA\_HSTART

Address: 0x0026

Direction: RO

Reset: 0x00000000

**Table 125. DPRX0\_MSA\_HSTART Bits**

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



### 11.4.8. DPRX0\_MSA\_VSTART

Address: 0x0027

Direction: RO

Reset: 0x00000000

**Table 126. DPRX0\_MSA\_VSTART Bits**

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

### 11.4.9. DPRX0\_MSA\_VSP

MSA vertical synchronization polarity register, DPRX0\_MSA\_VSP.

Address: 0x0028

Direction: RO

Reset: 0x00000000

**Table 127. DPRX0\_MSA\_VSP Bits**

Bit	Bit Name	Function
31:1	Unused	
0	VSP	Main stream attribute vertical synchronization polarity <ul style="list-style-type: none"> <li>0 = Positive</li> <li>1 = Negative</li> </ul>

### 11.4.10. DPRX0\_MSA\_VSW

MSA vertical synchronization width register, DPRX0\_MSA\_VSW.

Address: 0x0029

Direction: RO

Reset: 0x00000000

**Table 128. DPRX0\_MSA\_VSW Bits**

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

### 11.4.11. DPRX0\_MSA\_HWIDTH

TX control register, DPRX0\_MSA\_HWIDTH.

Address: 0x002a



Direction: RO

Reset: 0x00000000

**Table 129. DPRX0\_MSA\_HWIDTH Bits**

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

#### 11.4.12. DPRX0\_MSA\_VHEIGHT

Address: 0x002b

Direction: RO

Reset: 0x00000000

**Table 130. DPRX0\_MSA\_WHEIGHT Bits**

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

#### 11.4.13. DPRX0\_MSA\_MISC0

Address: 0x002c

Direction: RO

Reset: 0x00000000

**Table 131. DPRX0\_MSA\_MISC0 Bits**

Bit	Bit Name	Function
31:8	Unused	
7:0	MISC0	Main stream attribute MISC0 (refer to the <i>VESA DisplayPort Standard</i> )

#### 11.4.14. DPRX0\_MSA\_MISC1

Address: 0x002d

Direction: RO

Reset: 0x00000000

**Table 132. DPRX0\_MSA\_MISC1 Bits**

Bit	Bit Name	Function
31:8	Unused	
7:0	MISC1	Main stream attribute MISC1 (refer to the <i>VESA DisplayPort Standard</i> )





### 11.4.15. DPRX0\_MSA\_COLOR

Address: 0x002e

Direction: RO

Reset: 0x00000000

**Table 133. DPRX0\_MSA\_COLOR Bits**

Bit	Bit Name	Function
31:13	Unused	
12	DYNAMIC_RANGE	<ul style="list-style-type: none"> <li>0 = VESA (from 0 to maximum)</li> <li>1 = CEA range</li> </ul>
11:8	COLORIMETRY	<ul style="list-style-type: none"> <li>0000 = ITU-R BT601-5</li> <li>0001 = ITU-R BT709-5</li> </ul> <p><i>Note: Refer to Table 2-120 bit[3:0] in the VESA DisplayPort Standard version 1.4 for all colorimetry support including BT.2020.</i></p>
7:4	ENCODING	<ul style="list-style-type: none"> <li>0000 = RGB</li> <li>0001 = YCbCr 4:4:4</li> <li>0010 = YCbCr 4:2:2</li> <li>0011 = YCbCr 4:2:0</li> </ul>
3	Unused	
2:0	BPC	<p>Bits per pixel format</p> <ul style="list-style-type: none"> <li>000 = 6 bpc</li> <li>001 = 8 bpc</li> <li>010 = 10 bpc</li> <li>011 = 12 bpc</li> <li>100 = 16 bpc</li> </ul>

### 11.4.16. DPRX0\_VBID

VB-ID register, DPRX0\_VBID.

Address: 0x002f

Direction: RO

Reset: 0x00000000

**Table 134. DPRX0\_VBID Bits**

Bit	Bit Name	Function
31:8	Unused	
7	MSA_LOCK	<ul style="list-style-type: none"> <li>0 = MSA unlocked</li> <li>1 = MSA locked (on all lanes)</li> </ul>
6	VBID_LOCK	<ul style="list-style-type: none"> <li>0 = VB-ID unlocked</li> <li>1 = VB-ID locked (on all lanes)</li> </ul>
5:0	VBID	VB-ID flags (refer to the VESA DisplayPort Standard).

## 11.5. Sink Audio Registers

The audio registers are allocated at addresses:

- 0x0030 through 0x003f for Stream 0
- 0x0050 through 0x005f for Stream 1
- 0x0070 through 0x007f for Stream 2
- 0x0090 through 0x009f for Stream 3

*Note:* Only registers for Stream 0 are listed in the following sections.

### 11.5.1. DPRX0\_AUD\_MAUD

Received audio Maud register, DPRX0\_AUD\_MAUD.

Address: 0x0030

Direction: RO

Reset: 0x00000000

**Table 135. DPRX0\_AUD\_MAUD Bits**

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

### 11.5.2. DPRX0\_AUD\_NAUD

Received audio Naud register, DPRX0\_AUD\_NAUD.

Address: 0x0031

Direction: RO

Reset: 0x00000000

**Table 136. DPRX0\_AUD\_NAUD Bits**

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

### 11.5.3. DPRX0\_AUD\_AIF0

Received audio InfoFrame register, DPRX0\_AUD\_AIF0.

Address: 0x0032

Direction: RO

Reset: 0x00000000



**Table 137. 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)

#### 11.5.4. DPRX0\_AUD\_AIF1

Received audio InfoFrame register, DPRX0\_AUD\_AIF1.

Address: 0x0033

Direction: RO

Reset: 0x00000000

**Table 138. 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)

#### 11.5.5. DPRX0\_AUD\_AIF2

Received audio InfoFrame register, DPRX0\_AUD\_AIF2.

Address: 0x0034

Direction: RO

Reset: 0x00000000

**Table 139. 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)

#### 11.5.6. DPRX0\_AUD\_AIF3

Received audio InfoFrame register, DPRX0\_AUD\_AIF3.

Address: 0x0035

Direction: RO

Reset: 0x00000000

**Table 140. 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)

**11.5.7. DPRX0\_AUD\_AIF4**

Received audio InfoFrame register, DPRX0\_AUD\_AIF4.

Address: 0x0036

Direction: R0

Reset: 0x00000000

**Table 141. 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)

**11.6. Sink MST Registers**

MST controller control.

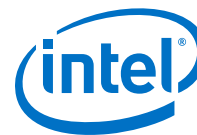
Address: 0x00a0

Direction: RW

Reset: 0x00000000

**Table 142. DPRX\_MST\_CONTROL1 Bits**

Bit	Bit Name	Function
31	VCPTAB_UPD_FORCE	This flag always reads back at 0. 1 = Force VC payload ID table update.
30	VCPTAB_UPD_REQ	<ul style="list-style-type: none"> <li>1 = Request for VC payload ID table update</li> <li>0 = No change to VC payload ID table</li> </ul>
29:20	Unused	
19:16	VCP_ID3	VC payload ID for Stream 3
15:12	VCP_ID2	VC payload ID for Stream 2
11:8	VCP_ID1	VC payload ID for Stream 1
7:4	VCP_ID0	VC payload ID for Stream 0
3:1	Unused	
0	MST_EN	Enable or disable MST <ul style="list-style-type: none"> <li>1 = MST framing</li> <li>0 = SST framing</li> </ul>



## 11. DisplayPort Sink Register Map and DPCD Locations

UG-01131 | 2019.04.01

When you assert VCPTAB\_UPD\_FORCE, the sink forces the VC payload table contained in DPRX\_MST\_VCPTAB0 through DPRX\_MST\_VCPTAB7 to be taken immediately into use.

When you assert VCPTAB\_UPD\_REQ, the sink requests the VC payload table contained in DPRX\_MST\_VCPTAB0 to DPRX\_MST\_VCPTAB7 to be taken into use after the next ACT sequence is detected.

The VC Payload ID values (1–15) used for VCP\_ID0 to VCP\_ID3 are different from those used by the DisplayPort source (1–63). The GPU must remap these values. The values used have to match those in the VC Payload ID table—DPRX\_MST\_VCPTAB0 to DPRX\_MST\_VCPTAB7 registers.

MST controller status

Address: 0x00a1

Direction: RO

Reset: 0x00000000

**Table 143. DPRX\_MST\_STATUS1 Bits**

Bit	Bit Name	Function
31	Unused	
30	VCPTAB_ACT_ACK	<ul style="list-style-type: none"> <li>1 = ACT sequence detected and VC payload updated</li> <li>0 = No change to VC payload ID table</li> </ul>
29:0	Unused	

VCPTAB\_ACT\_ACK resets to 0 when VCPTAB\_UPD\_REQ deasserted. VCPTAB\_ACT\_ACK is set to 1 if VCPTAB\_UPD\_REQ is asserted and the ACT sequence is detected, signaling that the table contained in DPRX\_MST\_VCPTAB0 to DPRX\_MST\_VCPTAB7 registers have been taken into use.

### 11.6.1. DPRX\_MST\_VCPTAB0

VC Payload ID Table

Address: 0x00a2

Direction: RW

Reset: 0x00000000

**Table 144. DPRX\_MST\_VCPTAB0 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT7	VC payload ID or slot 7
27:24	VCPSLOT6	VC payload ID or slot 6
23:20	VCPSLOT5	VC payload ID or slot 5
19:16	VCPSLOT4	VC payload ID or slot 4
<i>continued...</i>		



Bit	Bit Name	Function
15:12	VCPSLOT3	VC payload ID or slot 3
11:8	VCPSLOT2	VC payload ID or slot 2
7:4	VCPSLOT1	VC payload ID or slot 1
3:0	Reserved	Reserved

### 11.6.2. DPRX\_MST\_VCPTAB1

VC Payload ID Table

Address: 0x00a3

Direction: RW

Reset: 0x00000000

**Table 145. DPRX\_MST\_VCPTAB1 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT15	VC payload ID or slot 15
27:24	VCPSLOT14	VC payload ID or slot 14
23:20	VCPSLOT13	VC payload ID or slot 13
19:16	VCPSLOT12	VC payload ID or slot 12
15:12	VCPSLOT11	VC payload ID or slot 11
11:8	VCPSLOT10	VC payload ID or slot 10
7:4	VCPSLOT9	VC payload ID or slot 9
3:0	VCPSLOT8	VC payload ID or slot 8

### 11.6.3. DPRX\_MST\_VCPTAB2

VC Payload ID Table

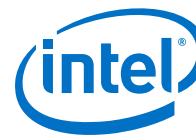
Address: 0x00a4

Direction: RW

Reset: 0x00000000

**Table 146. DPRX\_MST\_VCPTAB2 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT23	VC payload ID or slot 23
27:24	VCPSLOT22	VC payload ID or slot 22
23:20	VCPSLOT21	VC payload ID or slot 21
19:16	VCPSLOT20	VC payload ID or slot 20
15:12	VCPSLOT19	VC payload ID or slot 19
<i>continued...</i>		



## 11. DisplayPort Sink Register Map and DPCD Locations

UG-01131 | 2019.04.01

Bit	Bit Name	Function
11:8	VCPSLOT18	VC payload ID or slot 18
7:4	VCPSLOT17	VC payload ID or slot 17
3:0	VCPSLOT16	VC payload ID or slot 16

### 11.6.4. DPRX\_MST\_VCPTAB3

VC Payload ID Table

Address: 0x00a5

Direction: RW

Reset: 0x00000000

**Table 147. DPRX\_MST\_VCPTAB3 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT31	VC payload ID or slot 31
27:24	VCPSLOT30	VC payload ID or slot 30
23:20	VCPSLOT29	VC payload ID or slot 29
19:16	VCPSLOT28	VC payload ID or slot 28
15:12	VCPSLOT27	VC payload ID or slot 27
11:8	VCPSLOT26	VC payload ID or slot 26
7:4	VCPSLOT25	VC payload ID or slot 25
3:0	VCPSLOT24	VC payload ID or slot 24

### 11.6.5. DPRX\_MST\_VCPTAB4

VC Payload ID Table

Address: 0x00a6

Direction: RW

Reset: 0x00000000

**Table 148. DPRX\_MST\_VCPTAB4 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT39	VC payload ID or slot 39
27:24	VCPSLOT38	VC payload ID or slot 38
23:20	VCPSLOT37	VC payload ID or slot 37
19:16	VCPSLOT36	VC payload ID or slot 36
15:12	VCPSLOT35	VC payload ID or slot 35

*continued...*



Bit	Bit Name	Function
11:8	VCPSLOT34	VC payload ID or slot 34
7:4	VCPSLOT33	VC payload ID or slot 33
3:0	VCPSLOT32	VC payload ID or slot 32

### 11.6.6. DPRX\_MST\_VCPTAB5

VC Payload ID Table

Address: 0x00a7

Direction: RW

Reset: 0x00000000

**Table 149. DPRX\_MST\_VCPTAB5 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT47	VC payload ID or slot 47
27:24	VCPSLOT46	VC payload ID or slot 46
23:20	VCPSLOT45	VC payload ID or slot 45
19:16	VCPSLOT44	VC payload ID or slot 44
15:12	VCPSLOT43	VC payload ID or slot 43
11:8	VCPSLOT42	VC payload ID or slot 42
7:4	VCPSLOT41	VC payload ID or slot 41
3:0	VCPSLOT40	VC payload ID or slot 40

### 11.6.7. DPRX\_MST\_VCPTAB6

VC Payload ID Table

Address: 0x00a8

Direction: RW

Reset: 0x00000000

**Table 150. DPRX\_MST\_VCPTAB6 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT55	VC payload ID or slot 55
27:24	VCPSLOT54	VC payload ID or slot 54
23:20	VCPSLOT53	VC payload ID or slot 53
19:16	VCPSLOT52	VC payload ID or slot 52
15:12	VCPSLOT51	VC payload ID or slot 51

*continued...*





Bit	Bit Name	Function
11:8	VCPSLOT50	VC payload ID or slot 50
7:4	VCPSLOT49	VC payload ID or slot 49
3:0	VCPSLOT48	VC payload ID or slot 48

### 11.6.8. DPRX\_MST\_VCPTAB7

VC Payload ID Table

Address: 0x00a9

Direction: RW

Reset: 0x00000000

**Table 151. DPRX\_MST\_VCPTAB7 Bits**

Bit	Bit Name	Function
31:28	VCPSLOT63	VC payload ID or slot 63
27:24	VCPSLOT62	VC payload ID or slot 62
23:20	VCPSLOT61	VC payload ID or slot 61
19:16	VCPSLOT60	VC payload ID or slot 60
15:12	VCPSLOT59	VC payload ID or slot 59
11:8	VCPSLOT58	VC payload ID or slot 58
7:4	VCPSLOT57	VC payload ID or slot 57
3:0	VCPSLOT56	VC payload ID or slot 56

## 11.7. Sink AUX Controller Interface

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

### 11.7.1. 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. This step also 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 (approximately 10 ms) counter.
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 152. 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	
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).

### 11.7.2. DPRX\_AUX\_STATUS

AUX transaction status register, `DPRX_AUX_STATUS`.

Address: 0x0101

Direction: RO

Reset: 0x00000000

**Table 153. DPRX\_AUX\_STATUS Bits**

Bit	Bit Name	Function
31	MSG_READY	0 = Waiting for a request
<i>continued...</i>		



Bit	Bit Name	Function
		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

### 11.7.3. DPRX\_AUX\_COMMAND

AUX transaction command register, DPRX\_AUX\_COMMAND.

Address: 0x0102

Direction: RW

Reset: 0x00000000

**Table 154. 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 <i>VESA DisplayPort Standard</i> ). Reading of this register clears MSG_READY and LENGTH in DPRX_AUX_CONTROL register.

### 11.7.4. DPRX\_AUX\_BYTE0

AUX Transaction Byte 0 Register.

Address: 0x0103

Direction: RW

Reset: 0x00000000

**Table 155. 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

### 11.7.5. DPRX\_AUX\_BYTE1

AUX Transaction Byte 1 Register.

Address: 0x0104



Direction: RW

Reset: 0x00000000

**Table 156. 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

### 11.7.6. DPRX\_AUX\_BYTE2

AUX Transaction Byte 2 Register.

Address: 0x0105

Direction: RW

Reset: 0x00000000

**Table 157. 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 <i>VESA DisplayPort Standard</i> ).

### 11.7.7. DPRX\_AUX\_BYTE3

AUX Transaction Byte 3 Register.

Address: 0x0106

Direction: RW

Reset: 0x00000000

**Table 158. 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

### 11.7.8. DPRX\_AUX\_BYTE4

AUX Transaction Byte 4 Register.

Address: 0x0107

Direction: RW

Reset: 0x00000000

**Table 159. 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

**11.7.9. DPRX\_AUX\_BYTE5**

AUX Transaction Byte 5 Register.

Address: 0x0108

Direction: RW

Reset: 0x00000000

**Table 160. 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

**11.7.10. DPRX\_AUX\_BYTE6**

AUX Transaction Byte 6 Register.

Address: 0x0109

Direction: RW

Reset: 0x00000000

**Table 161. 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

**11.7.11. DPRX\_AUX\_BYTE7**

AUX Transaction Byte 7 Register.

Address: 0x010a

Direction: RW

Reset: 0x00000000

**Table 162. 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

### 11.7.12. DPRX\_AUX\_BYTES8

AUX Transaction Byte 8 Register.

Address: 0x010b

Direction: RW

Reset: 0x00000000

**Table 163. 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

### 11.7.13. DPRX\_AUX\_BYTE9

AUX Transaction Byte 9 Register.

Address: 0x010c

Direction: RW

Reset: 0x00000000

**Table 164. 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

### 11.7.14. DPRX\_AUX\_BYTE10

AUX Transaction Byte 10 Register.

Address: 0x010d

Direction: RW

Reset: 0x00000000



**Table 165. 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

### 11.7.15. DPRX\_AUX\_BYTE11

AUX Transaction Byte 11 Register.

Address: 0x010e

Direction: RW

Reset: 0x00000000

**Table 166. 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

### 11.7.16. DPRX\_AUX\_BYTE12

AUX Transaction Byte 12 Register.

Address: 0x010f

Direction: RW

Reset: 0x00000000

**Table 167. 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

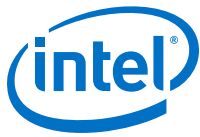
### 11.7.17. DPRX\_AUX\_BYTE13

AUX Transaction Byte 13 Register.

Address: 0x0110

Direction: RW

Reset: 0x00000000

**Table 168. 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

### 11.7.18. DPRX\_AUX\_BYTE14

AUX Transaction Byte 14 Register.

Address: 0x0111

Direction: RW

Reset: 0x00000000

**Table 169. 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

### 11.7.19. DPRX\_AUX\_BYTE15

AUX Transaction Byte 15 Register.

Address: 0x0112

Direction: RW

Reset: 0x00000000

**Table 170. 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

### 11.7.20. DPRX\_AUX\_BYTE16

AUX Transaction Byte 16 Register.

Address: 0x0113

Direction: RW

Reset: 0x00000000





**Table 171. DPRX\_AUX\_BYTE16 Bits**

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

### 11.7.21. DPRX\_AUX\_BYTE17

AUX Transaction Byte 17 Register.

Address: 0x0114

Direction: RW

Reset: 0x00000000

**Table 172. DPRX\_AUX\_BYTE17 Bits**

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

### 11.7.22. DPRX\_AUX\_BYTE18

AUX Transaction Byte 18 Register.

Address: 0x0115

Direction: RW

Reset: 0x00000000

**Table 173. DPRX\_AUX\_BYTE18 Bits**

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

### 11.7.23. DPRX\_AUX\_I2C0

AUX to I2C0 management. The sink routes all AUX channel accesses to I<sup>2</sup>C slave addresses of values between `START_ADDR` and `END_ADDR` to I2C0.

Address: 0x0116

WO

0x00000000

**Table 174. DPRX\_AUX\_I2C0 Bits**

Bit	Bit Name	Function
31:15	Unused	
14:8	END_ADDR	I <sup>2</sup> C slave end address
7	Unused	
6:0	START_ADDR	I <sup>2</sup> C slave start address

### 11.7.24. DPRX\_AUX\_I2C1

AUX to I2C1 management. The sink routes all AUX channel accesses to I<sup>2</sup>C slave addresses of values between START\_ADDR and END\_ADDR to I2C1.

Address: 0x0117

WO

0x00000000

**Table 175. DPRX\_AUX\_I2C1 Bits**

Bit	Bit Name	Function
31:15	Unused	
14:8	END_ADDR	I <sup>2</sup> C slave end address
7	Unused	
6:0	START_ADDR	I <sup>2</sup> C slave start address

### 11.7.25. DPRX\_AUX\_RESET

Address: 0x0118

Direction: WO

Reset: 0x00000000

**Table 176. DPRX\_AUX\_RESET Bits**

Bit	Bit Name	Function
31:1	Unused	
0	CLEAR	Asserting CLEAR resets the AUX controller state machine: <ul style="list-style-type: none"> <li>0 = No action</li> <li>1 = AUX Controller reset</li> </ul>

### 11.7.26. DPRX\_AUX\_HPDP

HPD control.

Address: 0x0119

Direction: RW

Reset: 0x00000000

**Table 177. DPRX\_AUX\_HPDP 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.
11	HPD_EN	HPD logic level 0 = Deasserted (low) 1 = Asserted (high)
10:0	Unused	

## 11.8. Sink CRC Registers

The CRC registers are available only Stream 0 and Stream 1 when the core is instantiated with parameter `RX_SUPPORT_AUTOMATED_TEST = 1`

DPRX0\_CRC\_R

Address: 0x0120

Direction: RO

Reset: 0x00000000

**Table 178. DPRX0\_CRC\_R Bits**

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

DPRX0\_CRC\_G

Address: 0x0121

Direction: RO

Reset: 0x00000000

**Table 179. DPRX0\_CRC\_G Bits**

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

DPRX0\_CRC\_B

Address: 0x0122

Direction: RO

Reset: 0x00000000



**Table 180. DPRX0\_CRC\_B Bits**

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

## 11.9. Sink-Supported DPCD Locations

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

**Table 181. DPCD Locations**

Location Name	Address	Without GPU	With GPU
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
NUMBER_OF_AUDIO_ENDPOINTS	0x0022	—	Yes
GUID	0x0030	Yes	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

*continued...*

## 11. DisplayPort Sink Register Map and DPCD Locations

UG-01131 | 2019.04.01



Location Name	Address	Without GPU	With GPU
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
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

*continued...*



## 11. DisplayPort Sink Register Map and DPCD Locations

UG-01131 | 2019.04.01

Location Name	Address	Without GPU	With GPU
TEST_REQUEST	0x0218	—	Yes
TEST_LINK_RATE	0x0219	—	Yes
TEST_LANE_COUNT	0x0220	—	Yes
TEST_CRC_R_Cr	0x0240	Yes	—
TEST_CRC_G_Y	0x0242	Yes	—
TEST_CRC_B_Cb	0x0244	Yes	—
TEST_SINK_MISC	0x0246	Yes	—
PHY_TEST_PATTERN	0x0248	Yes	Yes
TEST_80BIT_CUSTOM_PATTERN (0x0250 to 0x0259)	0x0250	Yes	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
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

*continued...*

## 11. DisplayPort Sink Register Map and DPCD Locations

UG-01131 | 2019.04.01



Location Name	Address	Without GPU	With GPU
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
EDP_DISPLAY_CONTROL	0x0720	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



## 12. DisplayPort Intel FPGA IP User Guide Archives

---

If an IP core version is not listed, the user guide for the previous IP core version applies.

IP Core Version	User Guide
18.1	<a href="#">DisplayPort Intel FPGA IP User Guide</a>
18.0	<a href="#">DisplayPort Intel FPGA IP User Guide</a>
17.1	<a href="#">Intel FPGA DisplayPort IP Core User Guide</a>
17.0	<a href="#">DisplayPort IP Core User Guide</a>
16.1	<a href="#">DisplayPort IP Core User Guide</a>
16.0	<a href="#">DisplayPort IP Core User Guide</a>
15.1	<a href="#">DisplayPort IP Core User Guide</a>
15.0	<a href="#">DisplayPort IP Core User Guide</a>
14.1	<a href="#">DisplayPort IP Core User Guide</a>



## 13. Document Revision History for the DisplayPort Intel FPGA IP User Guide

Document Version	Intel Quartus Prime Version	Changes
2019.04.01	19.1	<ul style="list-style-type: none"> <li>Added support for Intel Stratix 10 L-tile devices. Support for both Intel Stratix 10 L-tile and H-tile devices are final.</li> <li>Added a table that lists the support for the Adaptive Sync feature by device family in the <i>Device Family Support</i> section. This feature is available only in the Intel Quartus Prime Pro Edition software.</li> </ul>
2019.01.21	18.1	<ul style="list-style-type: none"> <li>Added preliminary support for Intel Stratix 10 devices.</li> <li>Removed the line that states that the IP supports multi-stream transport (MST) in Intel Cyclone 10 GX devices. The DisplayPort Intel FPGA IP supports MST only in Intel Arria 10 devices in the current release.</li> <li>Edited the performance resource utilization information to include data for Intel Stratix 10 devices and SST TX quad and MST data for Intel Arria 10 devices.</li> <li>Adaptive sync feature is fully supported in version 18.1 onwards.</li> <li>Updated the <i>Core Features</i> section to include support for HDR metadata transport using secondary stream data packet.</li> <li>Updated the <i>Secondary Stream Interface</i> section to add information about using the secondary stream data packet to transport HDR metadata.</li> <li>Edited the <code>btc_dptx_baseaddr</code> function information. The bit returns with the base address, and not 0 or 1.</li> <li>Added a reference link to the <i>DisplayPort Intel Stratix 10 FPGA IP Design Example User Guide</i>.</li> </ul>
2018.05.07	18.0	<ul style="list-style-type: none"> <li>Renamed DisplayPort IP core to DisplayPort Intel FPGA IP as part of standardizing and rebranding exercise.</li> <li>Added reference link to the <i>DisplayPort Intel Cyclone 10 GX FPGA IP Design Example User Guide</i>.</li> <li>Updated support for Intel Cyclone 10 GX device from advance to final.</li> <li>Edited the performance resource utilization information to include Arria V GZ.</li> <li>Add bit 0 and bit 127 to the <i>Typical Secondary Stream Packet Flow</i> diagram to indicate the direction of the streaming data.</li> <li>Edited the <code>btc_dptx_set_color_space</code> function information to include the missing code.</li> <li>Changed the typo in step 2 in the DisplayPort post link training adjust request flow (LQA). The offset 0x00101 bit [1] should be offset 0x00101 bit [5].</li> </ul>
<b>continued...</b>		

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

\*Other names and brands may be claimed as the property of others.



Document Version	Intel Quartus Prime Version	Changes
		<ul style="list-style-type: none"> <li>Added a note in the <i>btc_dptx_set_color_space</i>, <i>btc_dptx_set_color_space</i>, <i>btc_dptxll_stream_set_color_space</i>, <i>DPTX0_MSA_COLOR</i>, and <i>DPRX0_MSA_COLOR</i> topics to refer to Table 2-120 bit[3:0] in the <i>VESA DisplayPort Standard version 1.4</i> for all colorimetry support including BT.2020.</li> <li>Updated the video format information for <i>btc_dptx_set_color_space</i>, <i>btc_dptx_mst_set_color_space</i>, and <i>btc_dptxll_stream_set_color_space</i> functions. The format is 0 = RGB; 1 = YCbCr 4:4:4; 2 = YCbCr 4:2:2; 3 = YCbCr 4:2:0.</li> <li>Edited typos in the following API functions:               <ul style="list-style-type: none"> <li><i>btc_dptx_mst_conn_stat_notify_req</i></li> <li><i>btc_dptx_mst_link_address_req</i></li> <li><i>btc_dptx_mst_remote_dpcd_wr_req</i></li> <li><i>btc_dptx_mst_remote_i2c_rd_req</i></li> <li><i>btc_dptx_mst_set_color_space</i></li> <li><i>btc_dptx_mst_tavgts_set</i></li> <li><i>btc_dptxll_stream_set_pixel_rate</i></li> <li><i>btc_dptxll_syslib_add_tx</i></li> </ul> </li> </ul>

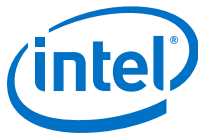
Date	Version	Changes
November 2017	2017.11.06	<ul style="list-style-type: none"> <li>Renamed DisplayPort IP core to Intel FPGA DisplayPort as per Intel rebranding.</li> <li>Changed the term Qsys to Platform Designer.</li> <li>Changed the term EyeQ to Eye Viewer as per Intel rebranding.</li> <li>Added advance support for Intel Cyclone 10 GX devices.</li> <li>Updated information that the DisplayPort Intel FPGA IP core now conforms to <i>Video Electronics Standards Association (VESA) DisplayPort Standard version 1.4</i>.</li> <li>Added data link rate support for HBR3 (8.10 Gbps). This rate is only available in quad symbols per clock for Intel Arria 10 and Intel Cyclone 10 GX devices in the Intel Quartus Prime Pro Edition software.</li> <li>Updated that the YCbCr 4:2:0 color format is fully supported starting 17.1 release.</li> <li>Updated the <i>Audio Interface</i> section to clarify that the audio packing format complies to both IEC-60958-1 and IEC-60958-3 standards.</li> <li>Moved information about the DisplayPort Intel FPGA design example parameters to the respective design example user guides.</li> <li>Added a note in the <i>Secondary Stream Interface</i> sections about InfoFrame SDP support.</li> <li>Edited the following registers:               <ul style="list-style-type: none"> <li><i>DPRX_RX_CONTROL</i> bits 10:8: Changed 111 from Reserved to Training pattern 4</li> <li><i>DPRX_BER_CONTROL</i> bits 1:0: Changed 00 to disparity error and code error counts and 10 to code error counts.</li> <li><i>DPTX0_MSA_COLOUR</i> bit 13: Added a note that if you configure this bit to use VSC SDP, refer to the <i>VESA DisplayPort Standard version 1.4</i> for the VSC SDP Payload Pixel Encoding/Colorimetry Format. Y-Only and Raw format are not supported.</li> <li><i>DPTX_RECONFIG</i> bits 1 and 0: Clarified that these bits automatically clear (0) after one clock cycle.</li> </ul> </li> </ul>
May 2017	2017.05.08	<ul style="list-style-type: none"> <li>Rebranded as Intel.</li> <li>Added preliminary support for adaptive sync feature and YCbCr 4:2:0 color format.</li> <li>Updated the <i>Device Family Support</i> section with the recommended speed grades information.</li> <li>Added input data ordering information for YCbCr 4:2:0 color format.</li> </ul>

**continued...**



Date	Version	Changes
		<ul style="list-style-type: none"> <li>• Added source support for proprietary video image format.                             <ul style="list-style-type: none"> <li>— Added information about the <b>TX Video IM Enable</b> parameter. Turn on to enable the video image interface. Turn off to use the traditional HSYNC/ VSYNC/DE video input interface.</li> <li>— Added information about the video image interface and a table showing comparison between the two interfaces.</li> </ul> </li> <li>• Added information about rx_analog_reconfig interface for the sink's <i>Transceiver Management Interface</i> table.</li> <li>• Added a note in the <i>Clocked Video Input Interface</i> section that the example given uses Intel's Clocked Video Input IP core.</li> <li>• Added information that MST parameter now supports audio data channel.</li> </ul>
October 2016	2016.10.31	<ul style="list-style-type: none"> <li>• Added information for the new <b>Design Example</b> parameters.</li> <li>• Removed all Arria 10 design example related information. For more information about Arria 10 design examples, refer to the <i>DisplayPort IP Core Design Example User Guide</i>.</li> <li>• Added information that MST parameter does not support audio data channel.</li> <li>• Added information about audio support for 2 symbols per clock.</li> <li>• Added information about DisplayPort MST source user application.</li> <li>• Updated information that the tx_analogreset[n-1:0], tx_digitalreset[n-1:0], rx_analogreset[n-1:0], and rx_digitalreset[n-1:0] signals are required only for Arria V, Cyclone V, and Stratix V devices.</li> <li>• Updated the API references.</li> <li>• Added new tx_idx parameter in TX API to support multiple TX instance.</li> <li>• Updated DisplayPort Sink and Source Register Map and DPCD locations.</li> </ul>
May 2016	2016.05.02	<ul style="list-style-type: none"> <li>• Updated performance resource utilization information for 16.0 version.</li> <li>• Added a note that the audio feature is not supported in dual symbol mode for link rates.</li> <li>• Removed all information about TX MSA. The TX MSA will be automatically inserted by the DisplayPort source core.                             <ul style="list-style-type: none"> <li>— Removed the <b>Import fixed MSA</b> parameter.</li> <li>— Removed the txN_msa_conduit signal.</li> </ul> </li> <li>• Updated the DisplayPort source functional block diagram and updated or included information for the related paths:                             <ul style="list-style-type: none"> <li>— Main link data path</li> <li>— Video packetizer path</li> <li>— Video geometry measurement path</li> <li>— Audio and secondary stream encoder path</li> <li>— Training and link quality patterns generator</li> </ul> </li> <li>• Added new information for DisplayPort source:                             <ul style="list-style-type: none"> <li>— Controller interface</li> <li>— Sideband channel</li> </ul> </li> <li>• Updated the source audio interface section to include information about 1-channel audio over 2-channel audio and 3-channel audio over 8-channel audio.</li> <li>• Updated video data format information for the DisplayPort source and sink cores.</li> <li>• Added support for black video feature for DisplayPort sink core.</li> <li>• Updated the Typical Secondary Stream Packet diagram for DisplayPort sink - changed data [127:0] to data [159:0].</li> <li>• Updated the DPTX_TX_CONTROL source register.</li> <li>• Added new information for DisplayPort hardware demonstration:                             <ul style="list-style-type: none"> <li>— DisplayPort Link Training Flow</li> <li>— DisplayPort Post Link Training Adjust Request Flow (LQA)</li> </ul> </li> <li>• Added links to archived versions of the <i>DisplayPort IP Core User Guide</i>.</li> </ul>

**continued...**



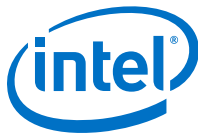
Date	Version	Changes
November 2015	2015.11.02	<ul style="list-style-type: none"> <li>• Changed instances of <i>Quartus II</i> to <i>Intel Quartus Prime</i>.</li> <li>• Updated performance resource utilization information for 15.1 version.</li> <li>• Removed information about <code>tx_vid_f</code>. The <code>tx_vid_f</code> pin is removed from the DisplayPort IP core because the signal is now handled internally by the core..</li> <li>• Added a new port, <code>rx_restart</code>, for RX transceiver interface. This port resets the RX PHY reset controller when RX data loses alignment. Only applicable for Arria 10 devices.</li> <li>• Added specific settings for Arria 10 Transceiver Native PHY, and Arria 10 hardware demonstration files for the DisplayPort hardware demonstration.</li> <li>• Added a new DisplayPort API function, <code>btc_dptx_hpd_change</code>.</li> </ul>
May 2015	2015.05.04	<ul style="list-style-type: none"> <li>• Added Arria 10 support.</li> <li>• Updated color support:               <ul style="list-style-type: none"> <li>– RGB—18, 24, 30, 36, or 48 bpp</li> <li>– YCbCr 4:4:4—24, 30, 36, or 48 bpp</li> <li>– YCbCr 4:2:2—16, 20, 24, or 32 bpp</li> </ul> </li> <li>• Removed information about Link Quality Generation register. These bits are now combined into the <code>DPTX_TX_CONTROL</code> register.</li> <li>• Added information about <code>DPTX_TEST_80BIT_PATTERN1-3</code> bits.</li> <li>• Added source-supported DPCD locations.</li> <li>• Added new sink-supported DPCD location bits: <code>TEST_REQUEST</code>, <code>TEST_LINK_RATE</code>, <code>TEST_LANE_COUNT</code>, <code>PHY_TEST_PATTERN</code>, and <code>TEST_80BIT_CUSTOM_PATTERN</code>.</li> <li>• Added Arria 10 information for the DisplayPort IP core hardware demonstration and simulation example.</li> </ul>
December 2014	2014.12.30	<p>Edited the DisplayPort RX link rate (Clock Recovery interface) for HBR2 from 4.50 Gbps to 5.40 Gbps.</p>
December 2014	2014.12.15	<ul style="list-style-type: none"> <li>• Added information about multi-stream support (MST, 1 to 4 source and sink streams). You can access this feature using these parameters:               <ul style="list-style-type: none"> <li>– Support MST</li> <li>– Max stream count</li> </ul> </li> <li>• Added support for 4Kp60 resolution.</li> <li>• Added information about clock recovery feature for the hardware demonstration.</li> <li>• Removed information for double reference clocks (162MHz and 270MHz) for transceiver clocking. The IP core no longer supports double reference clocks.</li> <li>• Added new source registers:               <ul style="list-style-type: none"> <li>– 0x00a0 (<code>DPTX_MST_CONTROL1</code>)</li> <li>– 0x00a2 (<code>DPTX_MST_VCPTAB0</code>)</li> <li>– 0x00a3 (<code>DPTX_MST_VCPTAB</code>)</li> <li>– 0x00a3 (<code>DPTX_MST_VCPTAB1</code>)</li> <li>– 0x00a4 (<code>DPTX_MST_VCPTAB2</code>)</li> <li>– 0x00a5 (<code>DPTX_MST_VCPTAB3</code>)</li> <li>– 0x00a6 (<code>DPTX_MST_VCPTAB4</code>)</li> <li>– 0x00a7 (<code>DPTX_MST_VCPTAB5</code>)</li> <li>– 0x00a8 (<code>DPTX_MST_VCPTAB6</code>)</li> <li>– 0x00a9 (<code>DPTX_MST_VCPTAB7</code>)</li> <li>– 0x00aa (<code>DPTX_MST_TAVG_TS</code>)</li> </ul> </li> </ul>

*continued...*



Date	Version	Changes								
		<ul style="list-style-type: none"> <li>• Added new sink registers:               <ul style="list-style-type: none"> <li>– 0x0006 (DPRX_BER_CNTI0)</li> <li>– 0x0007 (DPRX_BER_CNTI1)</li> <li>– 0x00a0 (DPRX_MST_CONTROL1)</li> <li>– 0x00a1 (DPRX_MST_STATUS1)</li> <li>– 0x00a2 (DPRX_MST_VCPTAB0)</li> <li>– 0x00a3 (DPRX_MST_VCPTAB1)</li> <li>– 0x00a4 (DPRX_MST_VCPTAB2)</li> <li>– 0x00a5 (DPRX_MST_VCPTAB3)</li> <li>– 0x00a6 (DPRX_MST_VCPTAB4)</li> <li>– 0x00a7 (DPRX_MST_VCPTAB5)</li> <li>– 0x00a8 (DPRX_MST_VCPTAB6)</li> <li>– 0x00a9 (DPRX_MST_VCPTAB7)</li> </ul> </li> <li>• Changed the value of the following source register bits:               <ul style="list-style-type: none"> <li>– 0x0000 - Bits RX_LINK_RATE</li> <li>– 0x0001 - Bits RX_LINK_RATE</li> <li>– 0x0002 - Bits RSTI3, RSTI2, RSTI1, RSTI0</li> </ul> </li> <li>• Added new signals:               <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%; padding: 2px;">clk_cal</td> <td style="width: 50%; padding: 2px;">Calibration clock for transceiver management interface</td> </tr> <tr> <td style="padding: 2px;">tx_link_rate_8bits rx_link_rate_8bits</td> <td style="padding: 2px;">Main link rate expressed in multiples of 270Mbps –</td> </tr> <tr> <td style="padding: 2px;">txN_video_in txN_vid_clk txN_audio txN_audio_clk txN_ss txN_msa_conduit</td> <td style="padding: 2px;">TX signals for Stream 1, 2, and 3</td> </tr> <tr> <td style="padding: 2px;">rxN_video_out rxN_vid_clk rxN_audio rxN_ss rxN_msa_conduit rxN_stream</td> <td style="padding: 2px;">RX signals for Stream 1, 2, and 3</td> </tr> </table> </li> <li>• Changed the following signal names:               <ul style="list-style-type: none"> <li>– rx_xcvr_clkout to rx_ss_clk</li> <li>– tx_xcvr_clkout to tx_ss_clk</li> </ul> </li> </ul>	clk_cal	Calibration clock for transceiver management interface	tx_link_rate_8bits rx_link_rate_8bits	Main link rate expressed in multiples of 270Mbps –	txN_video_in txN_vid_clk txN_audio txN_audio_clk txN_ss txN_msa_conduit	TX signals for Stream 1, 2, and 3	rxN_video_out rxN_vid_clk rxN_audio rxN_ss rxN_msa_conduit rxN_stream	RX signals for Stream 1, 2, and 3
clk_cal	Calibration clock for transceiver management interface									
tx_link_rate_8bits rx_link_rate_8bits	Main link rate expressed in multiples of 270Mbps –									
txN_video_in txN_vid_clk txN_audio txN_audio_clk txN_ss txN_msa_conduit	TX signals for Stream 1, 2, and 3									
rxN_video_out rxN_vid_clk rxN_audio rxN_ss rxN_msa_conduit rxN_stream	RX signals for Stream 1, 2, and 3									
June 2014	2014.06.30	<ul style="list-style-type: none"> <li>• Native PHY is removed from the IP core; included information about how to instantiate the PHY outside the DisplayPort IP core.</li> <li>• Updated the source and sink block diagrams.</li> <li>• Updated the source and sink register map information.</li> </ul>								

*continued...*



Date	Version	Changes
		<ul style="list-style-type: none"> <li>• Added new sink register bits:               <ul style="list-style-type: none"> <li>– LQA_ACTIVE</li> <li>– PHY_SINK_TEST_LANE_SEL</li> <li>– PHY_SINK_TEST_LANE_EN</li> <li>– AUX_IRQ_EN</li> <li>– TX_STROBE</li> <li>– DPRX_AUX_STATUS bits</li> <li>– DPRX_AUX_I2C0 bits</li> <li>– DPRX_AUX_I2C0 bits</li> <li>– DPRX_AUX_HPD bits</li> </ul> </li> <li>• Removed these sink register bits:               <ul style="list-style-type: none"> <li>– HPD_IRQ</li> <li>– HPD_EN</li> <li>– DPRX_AUX_IRQ_EN bits</li> </ul> </li> <li>• Added a new source register bit:               <ul style="list-style-type: none"> <li>– VTOTAL</li> </ul> </li> <li>• Added source TX transceiver interface signals</li> <li>• Removed these source signals:               <ul style="list-style-type: none"> <li>– xcvr_refclk</li> <li>– tx_serial_data</li> <li>– xcvr_reconfig</li> </ul> </li> <li>• Added sink audio and RX transceiver interface signals.</li> <li>• Removed these sink signals:               <ul style="list-style-type: none"> <li>– xcvr_refclk</li> <li>– rx_serial_data</li> <li>– xcvr_reconfig</li> </ul> </li> <li>• Added information about Transceiver Reconfiguration Interface for source and sink.</li> <li>• Added information about single clock reference (135 MHz) for source and sink.</li> <li>• Added information about Bitec HSMC DisplayPort daughter card in the <i>Hardware Demonstration</i> chapter.</li> <li>• Updated the API reference.</li> </ul>
November 2013	13.1	<ul style="list-style-type: none"> <li>• Updated the source and sink register map information.</li> <li>• Added dual and quad pixel mode support.</li> <li>• Added support for quad symbol (40-bit) transceiver data interface.</li> <li>• Added support for Cyclone V devices.</li> <li>• Added HBR2 support for Arria V and Arria V GZ devices.</li> <li>• Added information about eDP support.</li> <li>• Updated the API reference.</li> </ul>
May 2013	13.0	<ul style="list-style-type: none"> <li>• Added information on audio support.</li> <li>• Added HBR2 support for Stratix V devices.</li> <li>• Added information on secondary data support.</li> </ul>
February 2013	12.1 SP1 (Beta)	<p>Second beta release:</p> <ul style="list-style-type: none"> <li>• Updated the filenames for the hardware demonstration and simulation example.</li> <li>• Added chapter describing the IP core's compilation example.</li> <li>• Miscellaneous updates.</li> </ul>
December 2012	12.1 (Beta)	<p>Initial beta release.</p>