CPRI MegaCore Function

User Guide

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

UG-01062-5.0

Document last updated for Altera Complete Design Suite version: 12.0
Document publication date: June 2012

Feedback Subscribe

Contents

Chapter 5. Testing Features

Chapter 6. Signals

Chapter 7. Software Interface

Chapter 8. Testbenches

Appendix A. Initialization Sequence

Appendix B. Implementing CPRI Link Autorate Negotiation

Appendix C. Advanced AxC Mapping Modes

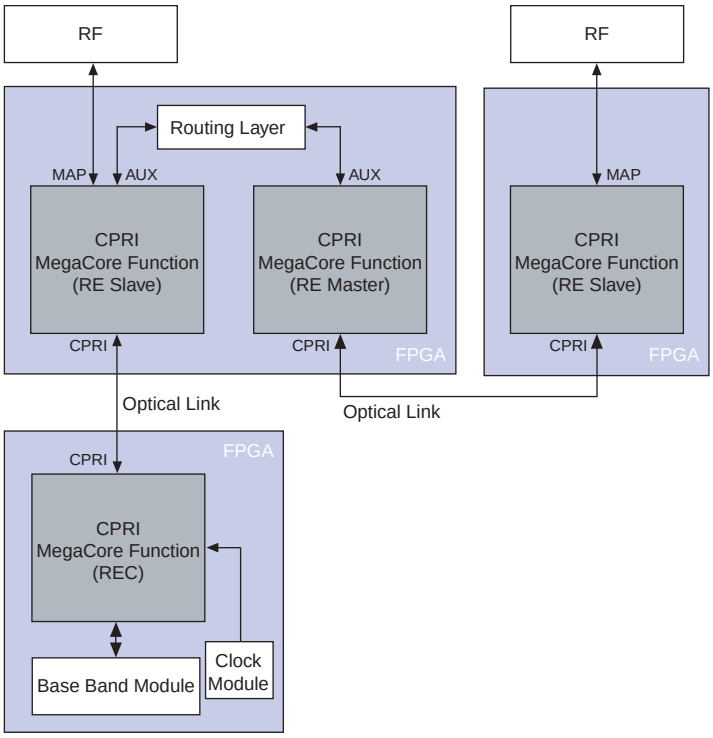
Appendix D. Delay Measurement and Calibration

1. About This MegaCore Function

The Altera® CPRI MegaCore® function implements the Common Public Radio Interface (CPRI) specification. CPRI is a high-speed serial interface designed for network radio equipment controllers (REC) to receive data from and provide data to remote radio equipment (RE).

The CPRI IP core targets high-performance, remote, radio network applications. You can configure the CPRI IP core as an RE or an REC. Figure 1–1 shows an example system implementation with a two-hop daisy chain. Optical links between devices support high performance.

Figure 1–1. Typical CPRI Application on Altera Devices



General Description

The Altera CPRI IP core implements Layer 1 and Layer 2 of the CPRI V4.2 specification. It provides access to the Layer 2 access points through various interfaces:

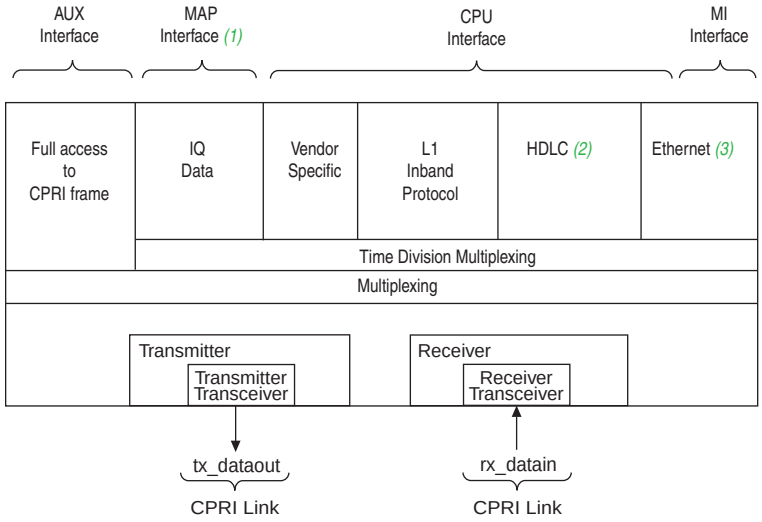
- IQ data access:
  - MAP antenna-carrier interfaces for easy IQ user data plane access based on pre-configured antenna-carrier channels.
  - Auxiliary interface for full access to the user data plane.

- Ethernet channel access:
  - Auxiliary interface for full access to the Ethernet space in the CPRI frame.
  - Register support for loading and unloading the Ethernet frame.
  - MI interface port for Ethernet Frame access.
- HDLC channel access:
  - Auxiliary interface for full access to the high level data link control (HDLC) space in the CPRI frame.
  - Register support for loading and unloading the HDLC frame.
- Vendor-specific data (VSS):
  - Auxiliary interface for full access to control bytes.
  - Register support for loading and unloading VSS space.
- Synchronization and Timing access:
  - Auxiliary interface for full access to synchronization and timing.

You configure the CPRI IP core to support either Ethernet communication with an Ethernet media access control (MAC) block included in the IP core, or communication with an external Ethernet module. The CPRI link line rate is configurable. For information about these interfaces and functionality, refer to Chapter 4, Functional Description. For information about configuration options, refer to Chapter 3, Parameter Settings.

Figure 1–2 shows the CPRI IP core interfaces. The IP core assembles the outbound CPRI frame control words and data from all of these interfaces, and unloads and routes control words and data from the inbound CPRI frame to the appropriate interfaces, based on configuration and register settings.

Figure 1–2. CPRI IP Core Interfaces



Notes to Figure 1–2:

(1) You can configure your CPRI IP core with zero, one, or multiple antenna-carrier interfaces. If you configure zero antenna-carrier interfaces, the MAP interface is not configured in your CPRI IP core. In that case you can communicate IQ data through the AUX interface to your user-defined routing layer.

(2) You can configure your CPRI IP core with or without a high-level data link control (HDLC) block.

(3) You can configure your CPRI IP core with an Ethernet MAC block or a media-independent (MI) interface (MII) block. The two options are mutually exclusive.

CPRI IP Core Features

The CPRI IP core has the following features:

- Complies with the Common Public Radio Interface (CPRI) Specification V4.2 (2010-09-29) Interface Specification for wireless base station submodule interconnections, without the full range of data sample widths.
- Supports radio equipment controller (REC) and radio equipment (RE) module configurations, including RE master, RE slave, and REC master ports.
- Supports Universal Mobile Telecommunication System (UMTS) Terrestrial Radio Access (UTRA) – frequency division duplexing (UTRA-FDD) (UMTS/Wideband Code Division Multiple Access (W-CDMA)), Evolved UTRA (E-UTRA) (3rd Generation Partnership Project (3GPP) Long Term Evolution (LTE) specification), and Worldwide interoperability for Microwave Access (WiMAX) (IEEE 802.16 standard).
- Full access to CPRI frame.

- Supports the following additional CPRI link features:
  - Programmable CPRI communication line rate (to 614.4, 1228.8, 2457.6, 3072.0, 4915.2, 6144.0, or 9830.4 Mbps) using Altera on-chip high-speed transceivers.
  - Auto-rate negotiation support.
  - Scrambling and descrambling at 4915.2 Mbps, 6144.0 Mbps, and 9830.4 Mbps.
  - Rx delay measurement.
  - Tx delay calibration.
  - Programmable hardware processing of the reset request bit in the CPRI frame.
  - Vendor-specific subchannel (VSS) communication on the CPRI link.
  - Diagnostic parallel reverse loopback paths.
- Includes the following additional interfaces:
  - Interface to external or on-chip processor, using the Altera Avalon® Memory-Mapped (Avalon-MM) interconnect specification.
  - Ethernet communication interfaces that support simultaneous Ethernet and HDLC communication to and from the CPRI link.
    - Optional configuration of Ethernet MAC.
    - Optional Media-Independent Interface for Ethernet frame access.
    - Optional configuration of HDLC block.
  - Auxiliary interface provides full access to CPRI frame.
    - Supports data transfer to and from custom mapping functions.
    - Supports data transfer from slave to master ports to implement daisy-chain topologies.
    - Supports custom IQ sample widths.
  - An IQ data interface with the following features:
    - Implements mapping methods in Sections 4.2.7.2.5 and 4.2.7.2.7 of the CPRI V4.2 Specification, and mapping Options 1 and 2 in Sections 4.2.7.2.3 and 4.2.7.2.4 of the CPRI V4.2 Specification.
    - Implements WiMAX mapping methods described in Sections 4.2.7.2.2, 4.2.7.2.5, and 4.2.7.2.7 of the CPRI V4.2 Specification.
    - Implements UMTS/LTE mapping methods described in Section 4.2.7.2 of the CPRI V4.2 Specification.
    - Implements WiMAX timing control methodology described in Section 4.2.8.2 of the CPRI V4.2 Specification.
    - Supports as many as 24 antenna-carrier interfaces.
    - Supports clocking antenna-carrier interfaces with external data channel clocks or internal IP core clock.
    - Supports synchronous buffer or simple FIFO synchronization modes for externally clocked antenna-carrier interfaces.
    - Supports independent sample rates for each antenna-carrier interface.

- Supports 15- and 16-bit data sample widths on uplink and downlink using the Altera Avalon Streaming (Avalon-ST) interconnect specification.

Device Family Support

Table 1–1 defines the device support levels for Altera IP cores.

Table 1–1. Altera IP Core Device Support Levels

FPGA Device Families	HardCopy Device Families
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.	**HardCopy Companion**—The IP core is verified with preliminary timing models for the HardCopy companion device. The IP core meets all functional requirements, but might still be undergoing timing analysis for the HardCopy 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.	**HardCopy Compilation**—The IP core is verified with final timing models for the HardCopy device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

Table 1–2 lists the level of support offered by the CPRI IP core for each Altera device family.

Table 1–2. Device Family Support

Device Family	Support
Arria® II (GX and GZ variants)	Final
Arria V (GX and GT variants)	Refer to the What's New in Altera IP page of the Altera website.
Cyclone® IV GX	Final
HardCopy® IV GX	HardCopy Compilation
Stratix® IV GX	Final
Stratix V	Refer to the What's New in Altera IP page of the Altera website.
Other device families	No support

MegaCore Verification

Before releasing a version of the CPRI IP core, Altera runs comprehensive regression tests in the current version of the Quartus® II software. These tests use the MegaWizard™ Plug-In Manager to create the instance files. Altera tests these files in simulation and hardware to confirm functionality.

Altera tests and verifies the CPRI IP core in hardware, especially the deterministic latency feature, for different platforms and environments.

Performance and Resource Utilization

This section contains tables showing IP core variation size and performance examples. For resource utilization information for additional CPRI IP core variations, refer to the reports the Quartus II software generates during compilation.

Table 1–3 lists the resources and expected performance for CPRI IP core variations configured with the following features:

- Operate in REC master mode
- Include autorate negotiation support if it is available at the relevant line rate in the device family (turned off in Arria V GT variations with CPRI line rate 9830.4 Mbps)
- Provide Ethernet access through the MI interface
- Do not provide an HDLC block
- Use Basic mapping mode
- Clock the AxC channels with independent clocks (the **Enable MAP interface synchronization with core clock** parameter is turned off)

The numbers of combinational ALUTs and logic registers are rounded up to the nearest 100.

Table 1–3 lists results obtained with the Quartus II software v12.0 for the following devices:

- Arria V GT (5AGTBD7E3F35I5)
- Arria V GX (5AGXFB3H4F35C4 for 6144, 4915.2, and 3072 Mbps variations and 5AGXFB3H6F35C6 for other variations)
- Stratix V GX (5SGXMA5N2F40I2 for 9830.4 Mbps variations and 5SGXMA5N2F40I4 for other variations)

Table 1–3. CPRI IP Core FPGA Resource Utilization (Part 1 of 2)

Device	Parameters		Memory			
	Line Rate (Mbps)	Number of Antenna-Carrier Interfaces	Combinational ALUTs	Logic Registers	M10K or M20K Blocks (1)	Memory ALUTs
Arria V GT	9830.4	0	4300	4000	11	—
		1	4800	4800	21	—
		4	5400	5500	27	—
		8	6000	6400	35	—

Table 1–3. CPRI IP Core FPGA Resource Utilization (Part 2 of 2)

Device	Parameters		Memory			
	Line Rate (Mbps)	Number of Antenna-Carrier Interfaces	Combinational ALUTs	Logic Registers	M10K or M20K Blocks (1)	Memory ALUTs
Arria V GX	614.4	0	3000	3100	5	—
		1	3800	4000	15	—
		2	3900	4200	17	—
		3	4100	4500	19	—
		4	4300	4800	21	—
	1228.8, 2457.6, 3072, 4915.2, 6144	0	3000	3100	5	—
		1	3800	4200	15	—
		4	4300	4800	21	—
		8	5000	5800	29	—
Stratix V GX	614.4	0	3000	3100	4	9
		1	3800	4000	11	9
		2	4000	4300	13	9
		3	4100	4500	15	9
		4	4300	4800	17	9
	1228.8, 2457.6, 3072, 4915.2, 6144, 9830.4	0	3100	3200	5	—
		1	3800	4300	11	—
		4	4400	5100	17	—
		8	5000	6200	25	—

Note to Table 1–3:

(1) M10K blocks in Arria V devices and M20K blocks in Stratix V devices.

Table 1–4 shows the slowest device family speed grade that supports each CPRI line rate in each device family. Lower speed grade numbers correspond to faster devices.

Table 1–4. Slowest Recommended Device Family Speed Grades (1) (Part 1 of 2)

Device Family or Variant	CPRI Line Rate (Mbps)						
	614.4	1228.8	2457.6	3072.0	4915.2	6144	9830.4
Arria II GX	–6	–6	–6	–6	I3 (2)	I3 (2)	(3)
Arria II GZ	–4	–4	–4	–4	–3	–3	(3)
Arria V GX	C6	C6	C6	I5	I5	I5	(3)
Arria V GT	C6	C6	C6	I5	I5	I5	I5
Cyclone IV GX	C8, I7	C8, I7	C8, I7	–7	(3)	(3)	(3)
Stratix IV GX	–4	–4	–4	–4	–4	–3	(3)

Table 1–4. Slowest Recommended Device Family Speed Grades (1) (Part 2 of 2)

Device Family or Variant	CPRI Line Rate (Mbps)						
	614.4	1228.8	2457.6	3072.0	4915.2	6144	9830.4
Stratix V GX	–4	–4	–4	–4	–4	–4	-2

Notes to Table 1–4:

(1) The entry –*x* indicates that both the industrial speed grade I*x* and the commercial speed grade C*x* are supported for this device family and CPRI line rate.

(2) Only the I3 speed grade is available for a CPRI IP core that runs at this line rate and targets the Arria II GX device family.

(3) This CPRI line rate is not supported for this device family.

Release Information

Table 1–5 provides information about this release of the CPRI IP core.

Table 1–5. CPRI Release Information

Item	Description
Version	12.0
Release Date	June 2012
Ordering Code	IP-CPRI
Product ID	00CB
Vendor ID	6AF7

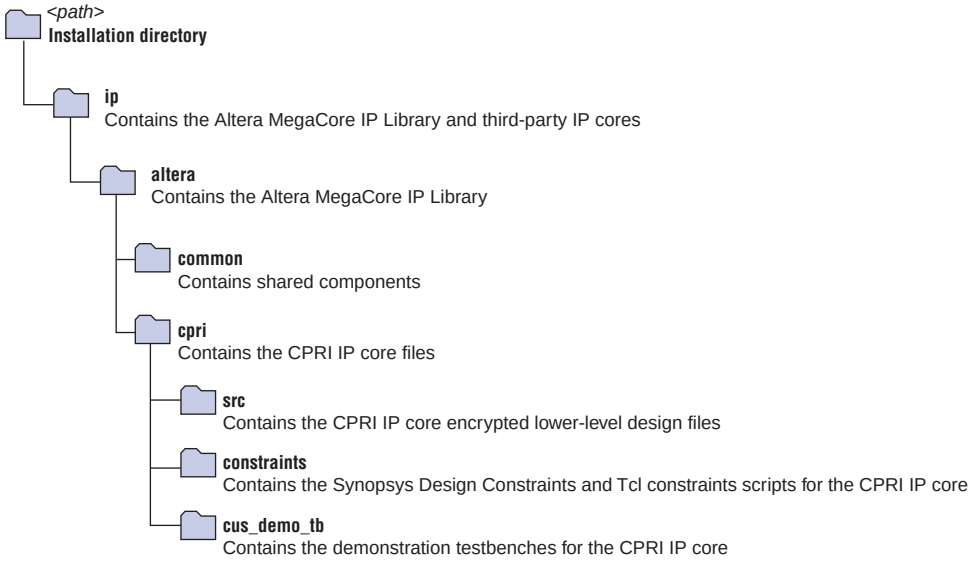
Altera verifies that the current version of the Quartus II software compiles the previous version of each Altera IP core. Any exceptions to this verification are reported in the *MegaCore IP Library Release Notes and Errata*. Altera does not verify compilation with IP core versions older than the previous release.

Installation and Licensing

The CPRI IP core is part of the MegaCore IP Library, which is distributed with the Quartus II software. The combined software is downloadable from the Altera website, www.altera.com.

Figure 1–3 shows the directory structure after you install the CPRI IP core, where <*path*> is the installation directory. The default installation directory on Windows is **C:\altera**<*version number*>; on Linux it is **/opt/altera**<*version number*>**.**

Figure 1–3. Directory Structure



You can use Altera's free OpenCore Plus evaluation feature to evaluate the CPRI IP core in simulation and in hardware before you purchase a license. You must purchase a license for the CPRI IP core only when you are satisfied with its functionality and performance, and you want to take your design to production.

After you purchase a license for the CPRI IP core, you can request a license file from the Altera website at **www.altera.com/licensing** and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have internet access, contact your local Altera representative.

OpenCore Plus Evaluation

With the Altera free OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction (Altera IP core or AMPP^{SM} megafunction) in your system using the Quartus II software and Altera-supported VHDL and Verilog HDL simulators
- Verify the functionality of your design and evaluate its size and speed quickly and easily
- Generate time-limited device programming files for designs that include Altera IP cores
- Program a device and verify your design in hardware

OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation supports the following two operation modes:

- *Untethered*—the design runs for a limited time.
- *Tethered*—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely.

All megafunctions in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior might be masked by the time-out behavior of the other megafunctions.

For Altera IP cores, the untethered time-out is 1 hour; the tethered time-out value is indefinite.

Your design stops working after the hardware evaluation time expires.

The CPRI IP core then behaves as if the `reset` and `cpu_reset` signals are asserted: the CPRI link and the CPU interface reset. The transceivers do not reset, because the transceiver quad might be shared with other designs, IP cores, and megafunctions. The CPRI IP core cannot achieve frame synchronization, and cannot participate in further CPRI communication.

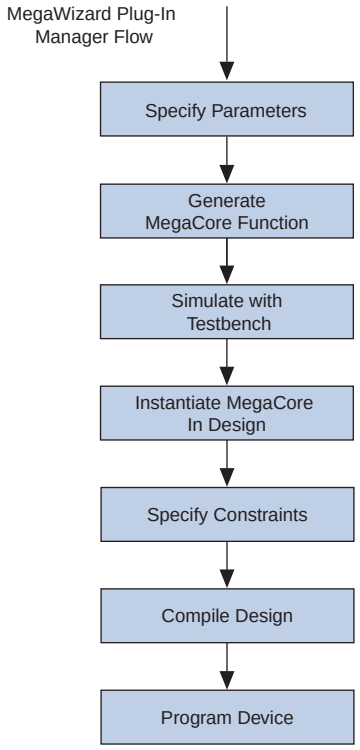
For information about installation and licensing, refer to *Altera Software Installation and Licensing*. For information about the OpenCore Plus evaluation feature, refer to *AN 320: OpenCore Plus Evaluation of Megafunctions*.

You can customize the CPRI IP core to support a wide variety of applications. You use the MegaWizard Plug-In Manager in the Quartus II software to parameterize a custom IP core variation in a CPRI parameter editor. The CPRI parameter editor lets you interactively set parameter values and select optional ports.

MegaWizard Plug-In Manager Design Flow

Figure 2–1 shows the stages for creating a system with the CPRI IP core and the Quartus II software. Each stage is described in detail in subsequent sections.

Figure 2–1. CPRI Design Flow



The MegaWizard Plug-In Manager flow allows you to customize the CPRI IP core, and manually integrate the function in your design.

Specifying Parameters

To specify CPRI IP core parameters using the MegaWizard Plug-In Manager, perform the following steps:

1. Create a Quartus II project using the **New Project Wizard** available from the File menu.

2. Launch the **MegaWizard Plug-In Manager** from the Tools menu, and follow the prompts in the MegaWizard Plug-In Manager interface to create a custom CPRI IP core variation.

   To select the CPRI IP core, click
   I**nstalled Plug-Ins** > **Interfaces > CPRI > CPRI v12.0**.

3. Specify the parameters. For details about these parameters, refer to Chapter 3, Parameter Settings.

   As you specify parameters, the CPRI parameter editor displays messages about the variation that your current settings define. If your settings define a variation for which a testbench is automatically generated when the CPRI IP core is generated, an information message tells you the name of the relevant testbench. For more information about the testbenches and the variations that provide them, refer to Chapter 8, Testbenches.

4. Click **Finish** to generate the CPRI IP core and supporting files.

   You might have to wait several minutes for file generation to complete.

5. When you are prompted to generate an example design, turn on **Generate Example Design**. You must turn on this option to generate the testbenches described in Chapter 8, Testbenches.

6. Click **Generate**. Despite the moving progress bar, generation does not progress until you click this button.

7. If you generate the CPRI IP core instance in a Quartus II project, you are prompted to add the Quartus II IP File (**.qip**) to the current Quartus II project. You can also turn on **Automatically add Quartus II IP Files to all projects**.

   The **.qip** file is generated by the parameter editor, and contains information about the generated IP core. In most cases, the **.qip** file contains all of the necessary assignments and information required to process the IP core or system in the Quartus II compiler. The parameter editor generates a single **.qip** file for each IP core.

Generating your custom CPRI IP core variation creates a set of HDL files and simulation models. You can now integrate your custom CPRI IP core variation in your design, simulate, and compile.

When you integrate your CPRI IP core variation in your design, observe the following connection and I/O assignment requirements:

- In Arria II, Cyclone IV GX, and Stratix IV GX designs:
  - Ensure that you connect the calibration clock (`gxb_cal_blk_clk`) to a clock signal with the appropriate frequency range of 10–125 MHz. The `cal_blk_clk` ports on other components that use transceivers must be connected to the same clock signal.
  - Add a dynamic reconfiguration block (`altgx_reconfig`) and connect it as specified in the *Arria II Device Handbook*, *Cyclone IV Device Handbook*, or *Stratix IV Device Handbook*. This block supports offset cancellation to compensate for analog voltages offset from required ranges due to process variations. The design compiles without the `altgx_reconfig` block, but it cannot function correctly in hardware.
  - To support the correct signal connections from the CPRI IP core to the dynamic reconfiguration block, in the ALTGX MegaWizard Plug-In Manager, on the **Reconfiguration Settings** tab, turn on **Analog controls**.
- In Arria V and Stratix V designs, add an Altera Transceiver Reconfiguration Controller and connect it as specified in the *Altera Transceiver PHY IP Core User Guide*. This block supports offset cancellation to compensate for analog voltages offset from required ranges due to process variations. The design does not compile without the Altera Transceiver Reconfiguration Controller, but it cannot function correctly in hardware.

Before you compile your system to generate a Programmable Object File (**.pof**) with which to configure your device, Altera recommends that you create assignments for the high-speed transceiver VCCH settings.

To create assignments for the high-speed transceiver VCCH settings, perform the following steps:

1. In the Quartus II window, on the Assignments menu, click **Assignment Editor**.
2. In the **<<new>>** cell in the **To** column, type the top-level signal name for your CPRI IP core instance `gxb_txdataout` signal.
3. Double-click in the **Assignment Name** column and click **I/O Standard**.
4. Double-click in the **Value** column and click your standard (for example, **1.5-V PCML)**.
5. In the new **<<new>>** row, repeat steps 2 to 4 for your CPRI IP core instance `gxb_rxdatain` signal.

Simulation Files

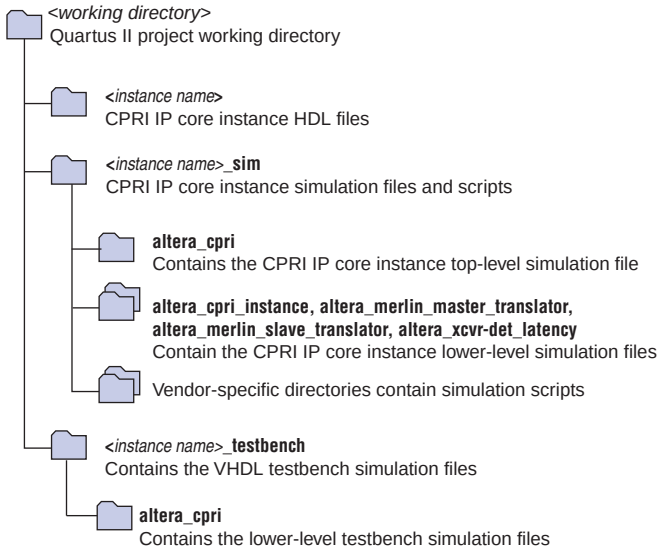
Generating a CPRI IP core creates an <*instance_name*>**_sim** directory with a subdirectory for each of several different simulators. Each of the vendor-specific directories contains files and scripts to simulate your CPRI IP core with that vendor's simulation tools.

The <*instance_name*>**_sim/altera_cpri** directory contains the top-level simulation file for your CPRI IP core.

Generating a CPRI IP core creates a more complex directory structure for Arria V and Stratix V variations than for variations that target other device families, because the Arria V and Stratix V variations instantiate an Altera Deterministic Latency PHY IP core or an Altera Native PHY IP core. In an Arria V or Stratix V variation, your <*instance_name*>**_sim** directory contains multiple subdirectories, one for each of the various components in the Arria V or Stratix V CPRI IP core, and individual directories for vendors for three different simulators. Each of the vendor-specific directories contains files and scripts to simulate your CPRI IP core with that vendor's simulation tools.

Figure 2–2 shows the directory structure of your CPRI IP core that contains a Deterministic Latency PHY IP core and generates a VHDL testbench. For information about the CPRI IP core variations that provide a VHDL testbench, refer to "Simulating the Design".

Figure 2–2. Generated CPRI IP Core Directory Structure for Most Arria V and Stratix V Variations



The **altera_xcvr_det_latency** directory contains the files to simulate the Altera Deterministic Latency PHY IP core that is generated as part of your CPRI IP core. It also contains a **mentor** subdirectory with IEEE encrypted files to simulate the PHY IP core efficiently.

Simulating the Design

During the design process, to check your design quickly, you can simulate your CPRI IP core with any of several Altera-supported EDA simulation tools.

For more information about these tools and how to simulate designs created using the Quartus II software, refer to the "Simulation" section in volume 3 of the *Quartus II Handbook*.

You can simulate your CPRI IP core variation using its IP functional simulation model and VHDL demonstration testbench. The IP functional simulation model, and testbench files for the CPRI IP core variations that support demonstration testbenches, are generated in your project directory when you generate your CPRI IP core. The testbench files include scripts to compile and run the demonstration testbench. The testbench demonstrates how to instantiate a model in a design and includes simple stimuli to control the user interfaces of the CPRI IP core.

A Verilog HDL testbench is not generated. If you specify Verilog HDL in the MegaWizard Plug-In Manager, it generates a Verilog HDL IP functional simulation model for the CPRI IP core. If your CPRI IP core variation is listed in Table 2–1, the corresponding VHDL demonstration testbench is also generated. You can use this model with the VHDL demonstration testbench for simulation using a mixed-language simulator.

For a complete list of models or libraries required to simulate the CPRI IP core, refer to the **compile[**_<*variation*>**]**_<*HDL*>**.do** scripts provided with the demonstration testbenches described in Chapter 8, Testbenches.

Not all variations provide demonstration testbenches. To view example scripts and to run a demonstration testbench, you must generate a variation that provides a testbench. Table 2–1 lists the CPRI variations that provide a testbench. Refer to Chapter 8, Testbenches for information about the specific testbench generated for each variation in Table 2–1. In addition to the variations specified in Table 8–4 on page 8–7, you generate VHDL testbenches with the corresponding Verilog HDL IP core variations, as shown in Table 2–1.

Table 2–1. CPRI IP Core Variations that Provide a Demonstration Testbench

Properties Common to all Variations with Testbench	Device Family	Enable Autorate Negotiation	Reference Clock Frequency	Include MAC Block	Number of Antenna-Carrier Interfaces
REC master clocking, 0.6144 Gbps line rate, Include HDLC Block is Off, Enable MAP interface synchronization is Off	Arria II	Off	—	On or Off	3
		Off	—	Off	0
	Arria V, Stratix V	Off	61.44 MHz	On or Off	3
		Off		Off	0
		On		On	0
	Cyclone IV GX, Stratix IV GX	Off	—	On or Off	3
		Off	—	Off	0
		On	—	On	0

For information about IP functional simulation models, refer to the *Simulating Altera Designs* chapter in volume 3 of the *Quartus II Handbook*.

Specifying Constraints

Altera provides a Synopsys Design Constraints (**.sdc**) file that you must apply to ensure that the CPRI IP core meets design timing requirements. In most cases the script requires modification for your design. For modification guidelines, refer to Appendix E, Integrating the CPRI IP Core Timing Constraints in the Full Design.

For information about timing analyzers, refer to the Quartus II Help and the "Timing Analysis" section in volume 3 of the *Quartus II Handbook*.

Compiling and Programming the Device

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

Before compiling your CPRI IP core or other incomplete CPRI design in the Quartus II software, you must assign unconnected CPRI IP core signals to virtual pins.

For information about compiling your design in the Quartus II software, refer to the *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*. For information about programming an Altera device, refer to the "Device Programming" section in volume 3 of the *Quartus II Handbook*.

Instantiating Multiple CPRI IP Cores

If you want to instantiate multiple CPRI IP cores in an Arria II, Cyclone IV GX, or Stratix IV GX device, you must observe a few additional requirements.

When your design contains multiple IP cores, you must ensure that the `gxb_cal_blk_clk` input and `gxb_powerdown` signals are connected according to the requirements for your target device family, and that the instances each have different starting channel numbers.

You must ensure that a single calibration clock source drives the `gxb_cal_blk_clk` input to each CPRI IP core (or any other megafunction or user logic that uses the ALTGX megafunction).

When you merge multiple CPRI IP cores in a single transceiver block, the same signal must drive `gxb_powerdown` to each of the CPRI IP core variations and other megafunctions, Altera IP cores, and user logic that use the ALTGX megafunction.

Multiple CPRI IP cores in a single device must use distinct transceiver channels. You enforce this restriction by specifying different starting channel numbers for the distinct CPRI IP cores. The starting channel number is a parameter whose value you specify for each CPRI IP core in the CPRI parameter editor. Refer to Chapter 3, Parameter Settings.

To configure multiple CPRI IP cores in a single transceiver block, you must specify in your Quartus Settings File (**.qsf**) that these CPRI link data lines are configured in the same `GXB_TX_PLL_RECONFIG_GROUP`, using the following syntax for each outgoing CPRI link `cN_gxb_txdataout`:

```
set_instance_assignment -name GXB_TX_PLL_RECONFIG_GROUP 1 -to cN_gxb_txdataout
```

3. Parameter Settings

You customize the CPRI IP core by specifying parameters in the CPRI parameter editor, which you access from the MegaWizard Plug-In Manager in the Quartus II software.

This chapter describes the parameters and how they affect the behavior of the CPRI IP core. You can modify parameter values to specify the following CPRI IP core properties:

- Clocking mode—whether this CPRI IP core instance is configured with slave clocking mode (RE slave) or with master clocking mode (REC or RE master).
- Line rate.
- Autorate negotiation—whether this CPRI IP core instance supports the connection of external logic to implement autorate negotiation.
- Starting channel number.
- Depth of the low-level receiver elastic buffer.
- Transceiver reference clock frequency. This option is available only in Arria V and Stratix V devices.
- Ethernet MAC—whether to include an internal Ethernet MAC block or provide an MII to connect to an external Ethernet module. These two options are mutually exclusive.
- HDLC block—whether to include an internal HDLC block or not.
- Number of antenna-carrier interfaces.
- Whether the antenna-carrier interfaces are clocked by the CPRI IP core clock `cpri_clkout` or by external clocks.

Physical Layer Parameters

This section lists the parameters that affect the configuration of the physical layer of the CPRI IP core.

Operation Mode Parameter

The **Operation mode** parameter specifies whether the CPRI IP core is configured with slave clocking mode or with master clocking mode. An REC is configured with master clocking mode.

Line Rate Parameter

The **Line rate** parameter specifies the line rate on the CPRI link in gigabits per second (Gbps). Table 3–1 lists the CPRI line rates that each device family supports. A checkmark indicates a supported variation.

Table 3–1. Device Family Support for CPRI Line Rates

Device Family or Variant	CPRI Line Rate (Mbps)						
	614.4	1228.8	2457.6	3072.0	4915.2	6144	9830.4
Arria II GX	✓	✓	✓	✓	✓ (1)	✓ (1)	—
Arria II GZ	✓	✓	✓	✓	✓	✓	—
Arria V GX	✓	✓	✓	✓	✓	✓	—
Arria V GT	✓	✓	✓	✓	✓	✓	✓
Cyclone IV GX	✓	✓	✓	✓	—	—	—
Stratix IV GX	✓	✓	✓	✓	✓	✓	—
Stratix V GX	✓	✓	✓	✓	✓	✓	✓
Stratix V GT	✓	✓	✓	✓	✓	✓	✓

Note to Table 3–1:

(1) If you specify a CPRI line rate of 4.9152 or 6.144 Gbps for a variation that targets an Arria II GX device, your Quartus II project must target an I3 speed grade device. The parameter editor does not enforce this restriction. However, if you violate this restriction, compilation fails because the design cannot meet timing in hardware.

Enable Autorate Negotiation

Autorate negotiation is the process of stepping down from a higher target CPRI line rate to a lower target CPRI line rate if you are unable to establish a link at the higher line rate. If your CPRI IP core has autorate negotiation enabled, and you program it to step down from its highest target CPRI line rate to its lower target CPRI line rates when it does not achieve frame synchronization, your CPRI IP core achieves frame synchronization at the highest possible CPRI line rate in its range of potential line rates, depending on the capability of its CPRI partner.

For information about the autorate negotiation feature, refer to Appendix B, Implementing CPRI Link Autorate Negotiation.

In the current release, CPRI IP core variations configured at the CPRI line rate of 9830.4 Mbps that target an Arria V device do not support autorate negotiation.

Turn on the **Enable auto-rate negotiation** parameter to specify that your CPRI IP core supports autorate negotiation. By default, this parameter is turned off.

Transceiver Starting Channel Number

You can specify the starting number for the CPRI IP core transceiver. For a CPRI IP core master, the **Master transceiver starting channel number** specifies the starting channel number for the transceiver.

For a CPRI IP core configured with slave clocking mode, the **Slave transmitter starting channel number** and **Slave receiver starting channel number** are two separate parameters. Both must have values that are starting channel numbers available in your design. The two numbers must be different but the Quartus II software creates an FPGA configuration with a single slave transceiver.

If you instantiate multiple CPRI IP cores on the same device, you must ensure each uses distinct transceiver channels.

These parameters are not available in Arria V and Stratix V devices.

Rx Elastic Buffer Depth

You can specify the depth of the Rx elastic buffer in the CPRI Receiver block. The **Receiver buffer depth** value is the $\log_2$ of the Rx elastic buffer depth. Allowed values are 4 to 8, inclusive.

The default depth of the Rx elastic buffer is 64, specified by the **Receiver buffer depth** parameter default value of 6. For most systems, the default Rx elastic buffer depth is adequate to handle dispersion, jitter, and wander that can occur on the link while the system is running. However, the parameter is available for cases in which additional depth is required.

Altera recommends that you set **Receiver buffer depth** to 4 in CPRI RE slave variations.

CPRI IP core variations configured at a CPRI line rate of 9830.4 Mbps that target an Arria V GT device do not include an Rx elastic buffer. However, this parameter affects the depth of the RX buffer between the soft PCS and the Altera Transceiver Native PHY IP core, instead. Refer to Figure 4–4 on page 4–7.

For information about the Altera Transceiver Native PHY IP core, refer to the *Altera Transceiver PHY IP Core User Guide*.

The value you specify for **Receiver buffer depth** is referred to as WIDTH_RX_BUF in this user guide.

For more information about the Rx elastic buffer, refer to "Rx Elastic Buffer" on page 4–49.

Transceiver Reference Clock Frequency

If your CPRI variation targets an Arria V or a Stratix V device, the **Transceiver reference clock frequency** parameter is available. Use this parameter to modify the expected frequency of the CPRI transceiver input reference clock to the frequency of an available clock for your design.

The frequency you specify is an input parameter to the Altera Deterministic Latency PHY IP core that is included in your Arria V or Stratix V CPRI variation. Values available at each CPRI line rate are the reference clock frequencies for which the Deterministic Latency PHY IP core supports the target CPRI line rate. The default value is 122.88 MHz.

For more information about the Altera Deterministic Latency PHY IP core, refer to the *Altera Transceiver PHY IP Core User Guide*.

Data Link Layer Parameters

This section lists the parameter that affects the configuration of the data link layer of the CPRI IP core.

Include MAC Block

Turn on the **Include MAC block** parameter to specify that your CPRI IP core includes an internal Ethernet MAC block. By default, this parameter is not turned on. If this parameter is not turned on, the CPRI IP core implements the media-independent (MI) interface (MII) to your own external Ethernet MAC, instead.

If this parameter is not turned on in your CPRI IP core, your application cannot access the Ethernet registers. Attempts to access these registers read zeroes and do not write successfully, as for a reserved register address.

For information about the internal Ethernet MAC block, refer to "Accessing the Ethernet Channel" on page 4–42.

For information about the MII, refer to "Media Independent Interface to an External Ethernet Block" on page 4–34.

Include HDLC Block

Turn on the **Include HDLC block** parameter to specify that your CPRI IP core includes an internal HDLC block. By default, this parameter is not turned on.

If this parameter is not turned on in your CPRI IP core, your application cannot access the HDLC registers. Attempts to access these registers read zeroes and do not write successfully, as for a reserved register address.

Application Layer Parameters

This section lists the parameters that affect the configuration of the application layer of the CPRI IP core.

Mapping Mode

The **Mapping mode(s)** parameter specifies whether your CPRI IP core MAP interface supports a programmable AxC mapping mode or is configured with a specific mapping mode. Table 3–2 lists the supported values.

Table 3–2. MAP Interface AxC Mapping Mode Support

Value	Description
All	If you select this value, you configure a CPRI IP core which you can program dynamically to be in any mapping mode. In this case, you determine the current mapping mode for your CPRI IP core by programming the `map_mode` field of the `CPRI_MAP_CONFIG` register (0x100). For backward compatibility with previous releases of the CPRI IP core, the value of **All** is the default value for this parameter. For information about the `map_mode` register field, refer to Table 7–31 on page 7–14.
Basic	Your CPRI IP core MAP interface is configured to function in basic mapping mode only. This mapping mode has the following features: ■ Conforms to the description in Sections 4.2.7.2.2 and 4.2.7.2.3 of the *CPRI Specification V4.2 Interface Specification*. ■ Supports communication that complies with the LTE/E-UTRA or UMTS/WCDMA standard. For information about the basic mapping mode in the CPRI IP core, refer to "MAP Interface Mapping Modes" on page 4–11.
Advanced 1	Your CPRI IP core MAP interface is configured in a single AxC mapping mode only, a mode that has the following features: ■ Conforms to Method 1: IQ Sample Based described in Section 4.2.7.2.5 of the *CPRI Specification V4.2 Interface Specification*. ■ Supports communication that complies with the WiMAX standard. For information about this AxC mapping mode, refer to Appendix C, Advanced AxC Mapping Modes.
Advanced 2	Your CPRI IP core MAP interface is configured in a single AxC mapping mode only, a mode that has the following features: ■ Conforms to Method 3: Backward Compatible described in Section 4.2.7.2.4 of the *CPRI Specification V4.2 Interface Specification*. ■ Supports communication that complies with the WiMAX or LTE/E-UTRA standard. For information about this AxC mapping mode, refer to Appendix C, Advanced AxC Mapping Modes.
Advanced 3	Your CPRI IP core MAP interface is configured in a single AxC mapping mode only, a legacy mode that has the following features: ■ Conforms to Method 1: IQ Sample Based described in Section 4.2.7.2.5 of the *CPRI Specification V4.2 Interface Specification.* ■ Supports communication that complies with the LTE/E-UTRA standard. This mode does not support 16-bit wide IQ data samples. Refer to Table 7–31 on page 7–14. For information about this AxC mapping mode, refer to Appendix C, Advanced AxC Mapping Modes.

Number of Antenna-Carrier Interfaces

The **Number of antenna/carrier interfaces** parameter specifies the number of antenna-carrier interfaces, or data channels, in your CPRI IP core. The supported values are 0 to 24. Set this parameter to the maximum number of data channels you expect your CPRI IP core to use at the same time.

If you set this parameter to zero, your CPRI IP core does not implement the CPRI MAP interface. For example, you might use this option if your CPRI IP core passes IQ data samples through the AUX interface to an external custom mapping function that you provide.

You can specify in software that some of the antenna-carrier interfaces that you configure in your CPRI IP core are not active. This feature allows you to change the number of active and enabled data channels dynamically.

The combination of CPRI IP core line rate, sampling width, and sampling rate restricts the number of active antenna-carrier interfaces your CPRI IP core can support. For example, if your CPRI IP core operates at line rate 3.072 Gbps, it can support as many as 20 active antenna-carrier interfaces, but if your CPRI IP core operates at line rate 1.2288 Gbps, it can support a maximum of eight active antenna-carrier interfaces. For details, refer to Table 4–4 and Table 4–5 on page 4–14.

The software configuration feature allows you to modify the number of active antenna-carrier interfaces; if you modify this number, you must keep in mind the restrictions for your current CPRI line rate. Otherwise, data is dropped in the mapping to and from the individual antenna-carrier interfaces.

If you set the `map_ac` field of the `CPRI_MAP_CNT_CONFIG` register to a number *N* that is lower than the value you specify for **Number of antenna/carrier interfaces**, then the first *N* data channels are active and the others are not. In addition, for each antenna-carrier interface you can use the relevant `map_rx_enable bit` of the `CPRI_IQ_RX_BUF_CONTROL` register and the relevant `map_tx_enable` bit of the `CPRI_IQ_TX_BUF_CONTROL` register to enable or disable the specific data channel and direction. A data channel must be configured, active, and enabled to function. If it is configured and active but not enabled, data to and from it is ignored.

The value you specify for **Number of antenna/carrier interfaces** is referred to as N_MAP in this user guide.

For more information about the antenna-carrier interfaces in a CPRI IP core, refer to "MAP Interface" on page 4–10.

Enable Internally-Clocked Synchronization Mode

If you configure one or more antenna-carrier interfaces, the option to **Enable MAP interface synchronization with core clock** is available. If you turn on this option, both the MAP receiver interface and the MAP transmitter interface are clocked with the CPRI IP core internal clock, `cpri_clkout`. If you turn off this option, these interfaces are clocked with individual Rx and Tx clocks for each antenna-carrier interface.

If you turn on this option, the CPRI IP core coordinates communication on these interfaces in the internally-clocked synchronization mode. Turning on this option simplifies synchronization of data transfers to and from the antenna-carrier interfaces.

The Boolean value you specify for **Enable MAP interface synchronization with core clock** is referred to as SYNC_MAP in this user guide. Table 3–3 shows the correspondence between the parameter, the MAP interface synchronization mode, and the clocks that clock the antenna-carrier interfaces.

Table 3–3. Meaning of Enable Map Interface synchronization with core clock Parameter

Enable MAP interface synchronization with core clock	SYNC_MAP	MAP Interface Synchronization Mode	Clocks for Antenna-Carrier Interfaces
On	1	Internally-clocked mode	`cpri_clkout`
Off	0	Synchronous buffer or FIFO mode	`mapN_rx_clk`, `mapN_tx_clk`, for antenna-carrier interfaces `N` = 1 ... (N_MAP – 1)

For more information about these clocks, refer to "Clocking Structure" on page 4–3. For more information about the synchronization modes for the Rx and Tx MAP interfaces, and how they vary depending on your selection of this option, refer to "MAP Interface" on page 4–10.

4. Functional Description

The CPRI protocol interface complies with the CPRI Specification V4.2. The specification divides the protocol into a two-layer hierarchy: a physical layer (layer 1) and a data link layer (layer 2). The specification describes the following three communication planes:

- User data
- Control and management (C&M)
- Timing synchronization information

More detailed information about the CPRI specification is available from the CPRI website at www.cpri.info.

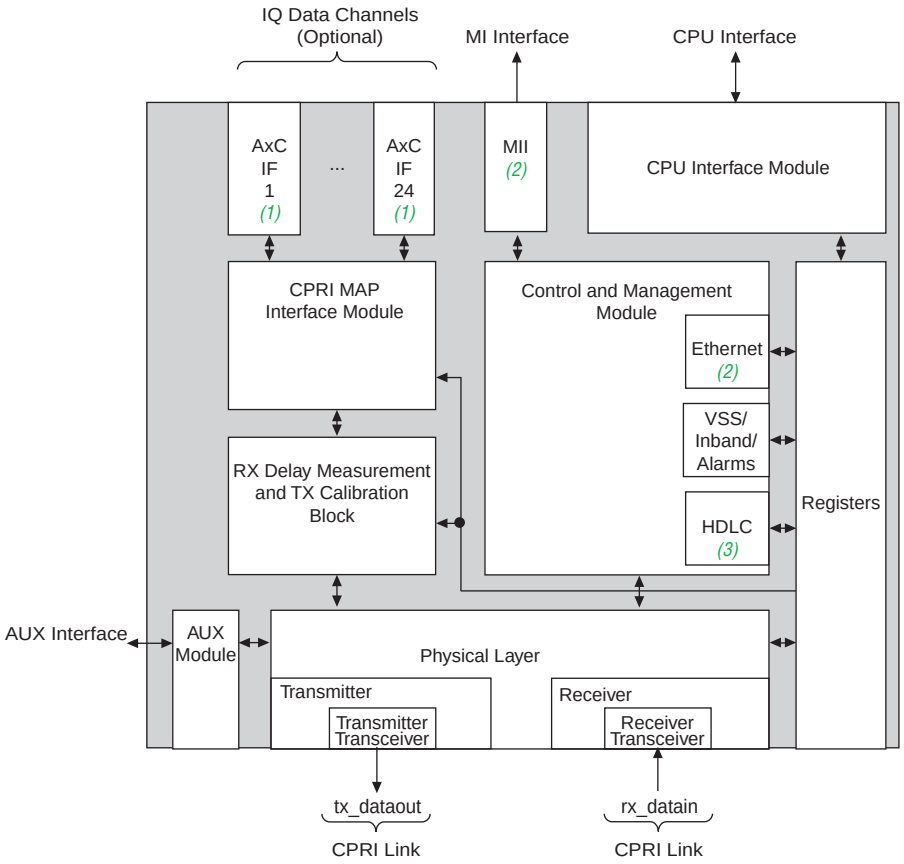
The Altera CPRI IP core implements layer 1 and layer 2 of the specification in the CPRI protocol interface module. This chapter describes the individual data and control interfaces available to you and how the data on these interfaces is loaded and unloaded from the CPRI frame.

This chapter contains the following sections:

Architecture Overview

Figure 4–1 shows the main blocks of the CPRI IP core.

Figure 4–1. CPRI IP Core Block Diagram



Notes to Figure 4–1:

(1) You can configure your CPRI IP core with zero, one, or multiple IQ data channels.
(2) You can configure your CPRI IP core with an Ethernet MAC block or an MII block. The two options are mutually exclusive.
(3) You can configure your CPRI IP core with or without a High-Level Data Link Controller (HDLC) block.

The Altera CPRI IP core supports the following interfaces:

- MAP Interface
- Auxiliary Interface
- Media Independent Interface to an External Ethernet Block
- CPU Interface
- CPRI link interface described in CPRI Protocol Interface Layer (Physical Layer)

Information about the signals on the individual interfaces is available in the following sections and in Chapter 6, Signals.

The following sections describe the individual interfaces and clocks.

Clocking Structure

The CPRI IP core has a variable number of clock domains. The clock domains in your CPRI IP core variation depend on the following factors:

- Number of antenna-carrier interfaces.
- Whether the MII is configured.
- Whether the antenna-carrier interfaces are clocked internally. Refer to "Enable Internally-Clocked Synchronization Mode" on page 3–6.
- Target device family.
- In one case, different CPRI line rates.

The input clock frequency requirements depend on the target device family and CPRI line rate. Refer to Table 4–2 on page 4–8 for these requirements.

You can configure a CPRI IP core in master or slave clocking mode, as described in "Operation Mode Parameter" on page 3–1. REC configurations and RE master configurations use master clocking mode, and RE slave configurations use slave clocking mode. Your design must handle some of the transceiver input clocks differently in the two different clocking modes.

Table 4–1 describes the individual clocks. The clocking diagrams in Figure 4–2 on page 4–5 to Figure 4–4 on page 4–7 show the clocks and clock domain boundaries. Table 4–2 on page 4–8 lists the clock frequencies for the different CPRI IP core variations.

CPRI IP Core Clocks

Table 4–1 describes the clock domains in the CPRI IP core.

For more information about these clocks, including driver requirements, refer to Chapter 6, Signals. For expected input clock frequencies refer to Chapter 6, Signals and to Table 4–2 on page 4–8.

Table 4–1. CPRI IP Core Clocks (Part 1 of 2)

Clock Name	Direction	Configuration Requirements	Description
`cpri_clkout`	Output	Present in all CPRI IP cores	Main clock for the CPRI IP core. The CPRI IP core derives this clock from the transceiver transmit PLL, and the frequency of this clock depends on the CPRI line rate. For more information refer to "CPRI Communication Link Line Rates" on page 4–7.
`mapN_tx_clk` for `N` in `0`..(N_MAP–1)	Input	Present in variations configured with N_MAP > 0 antenna-carrier interfaces and with **Enable MAP interface synchronization with core clock** turned off	Expected rate of received data on antenna-carrier interface N. The frequency of this clock is the sample rate on the incoming antenna-carrier interface. For more information about data channel sample rates, refer to Table 4–4 and Table 4–5 on page 4–14.
`mapN_rx_clk` for `N` in `0`..(N_MAP–1)	Input		Clocks the transmissions of antenna-carrier interface N. The frequency of this clock is the sample rate on the outgoing antenna-carrier interface. For more information about data channel sample rates, refer to Table 4–4 and Table 4–5 on page 4–14.

Table 4–1. CPRI IP Core Clocks (Part 2 of 2)

Clock Name	Direction	Configuration Requirements	Description
clk_ex_delay	Input	Present in all CPRI IP cores	Clock for extended delay measurement. For more information refer to "Extended Rx Delay Measurement" on page D–5.
cpri_mii_txclk	Output	Present in variations configured with an MI interface	Clocks the MII transmitter module. This clock has the same frequency as the cpri_clkout clock. The frequency depends on the CPRI line data rate. Refer to "CPRI Communication Link Line Rates" on page 4–7.
cpri_mii_rxclk	Output		Clocks the MII receiver module. This clock has the same frequency as the cpri_clkout clock. The frequency depends on the CPRI line data rate. Refer to "CPRI Communication Link Line Rates" on page 4–7.
cpu_clk	Input	Present in all CPRI IP cores	Clock that controls the input to the CPU interface of the CPRI IP core and drives the CPU interface. Assumed to be asynchronous with the cpri_clkout clock. The maximum frequency is constrained by f_{MAX} and can vary based on the device family and speed grade.
gxb_refclk	Input	Present in all CPRI IP cores	Reference clock for the transceiver PLLs. In master clocking mode, this clock drives both the receiver PLL and the transmitter PLL in the transceiver. In slave clocking mode, this clock drives the receiver PLL.
gxb_cal_blk_clk	Input	Not present in variations that target an Arria V or Stratix V device	Transceiver calibration-block clock.
reconfig_clk	Input	Present in all CPRI IP cores	Transceiver dynamic reconfiguration block clock.
gxb_pll_inclk	Input	Present in all CPRI IP cores	Input clock to the transmitter PLL in a CPRI IP core configured in slave clocking mode. If the CPRI IP core is configured in master clocking mode, it does not use this clock. In master clocking mode, you must tie this input low.
pll_clkout	Output	Present in all CPRI IP cores	Generated from transceiver clock data recovery circuit. Intended to connect to an external PLL for jitter clean-up in slave clocking mode.
usr_pma_clk	Input	Present in variations configured at 9830.4 Gbps that target an Arria V GT device	Extra clock signal required to drive the PMA in these CPRI IP core variations. Refer to Table 6–15 on page 6–17 for driver frequency and synchronization requirements.
usr_clk	Input		Extra clock signal required to drive the PCS in these CPRI IP core variations. Refer to Table 6–15 on page 6–17 for driver frequency and synchronization requirements.

Clock Diagrams for the CPRI IP Core

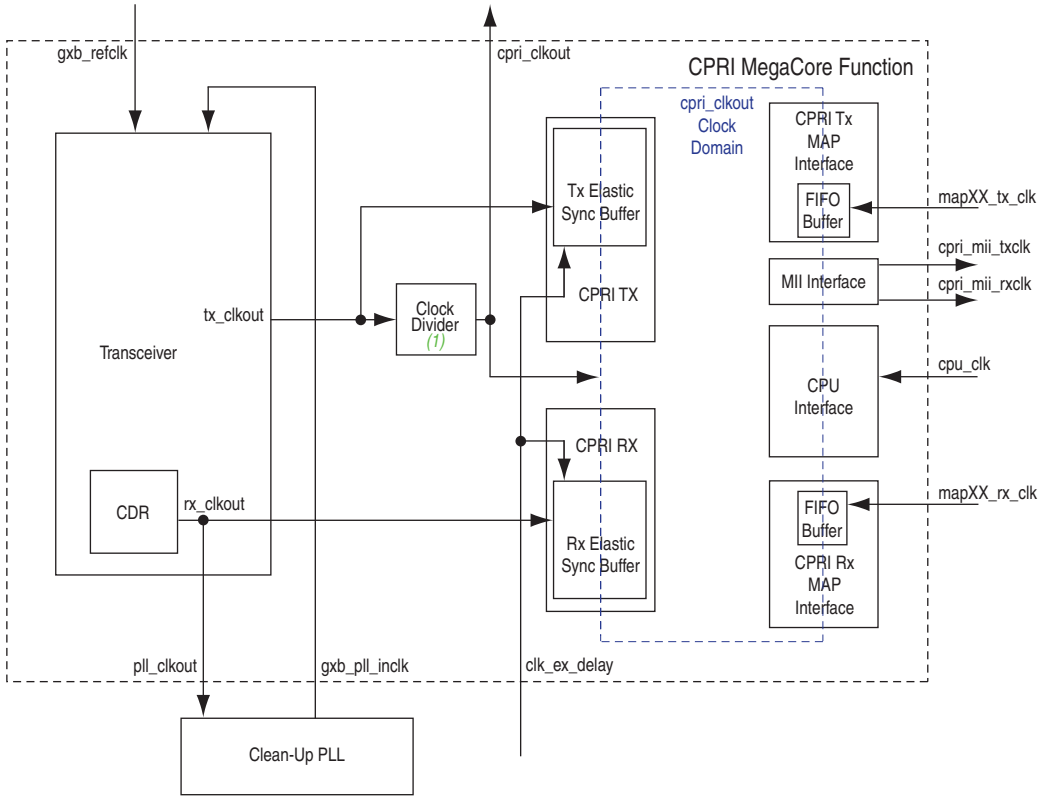
Figure 4–2 and Figure 4–3 show the clocking schemes for CPRI IP cores configured as RE slaves, RE masters, and REC masters that do not target an Arria V GT device or that are not configured with a CPRI line rate of 9830.4 Mbps.

Figure 4–4 on page 4–7 shows the clocking schemes for CPRI IP cores configured as RE slaves, RE masters, and REC masters with a CPRI line rate of 9830.4 Mbps that target an Arria V GT device. These variations have no clock divider and no Tx elastic buffer or Rx elastic buffer. However, they require two additional synchronized input clocks, `usr_pma_clk`, which you must drive at the frequency of 122.88 MHz, and `usr_clk`, which you must drive at the frequency of 245.76 MHz. Recall that these variations do not support autorate negotiation.

Clock Diagrams for Most CPRI IP Core Variations

Figure 4–2 shows the clock diagram for a CPRI IP core configured as an RE slave, unless the IP core is configured with CPRI line rate 9.830.4 Mbps and targets an Arria V GT device.

Figure 4–2. CPRI IP Core Slave Clocking Except for Arria V GT 9.8 Gbps Variations

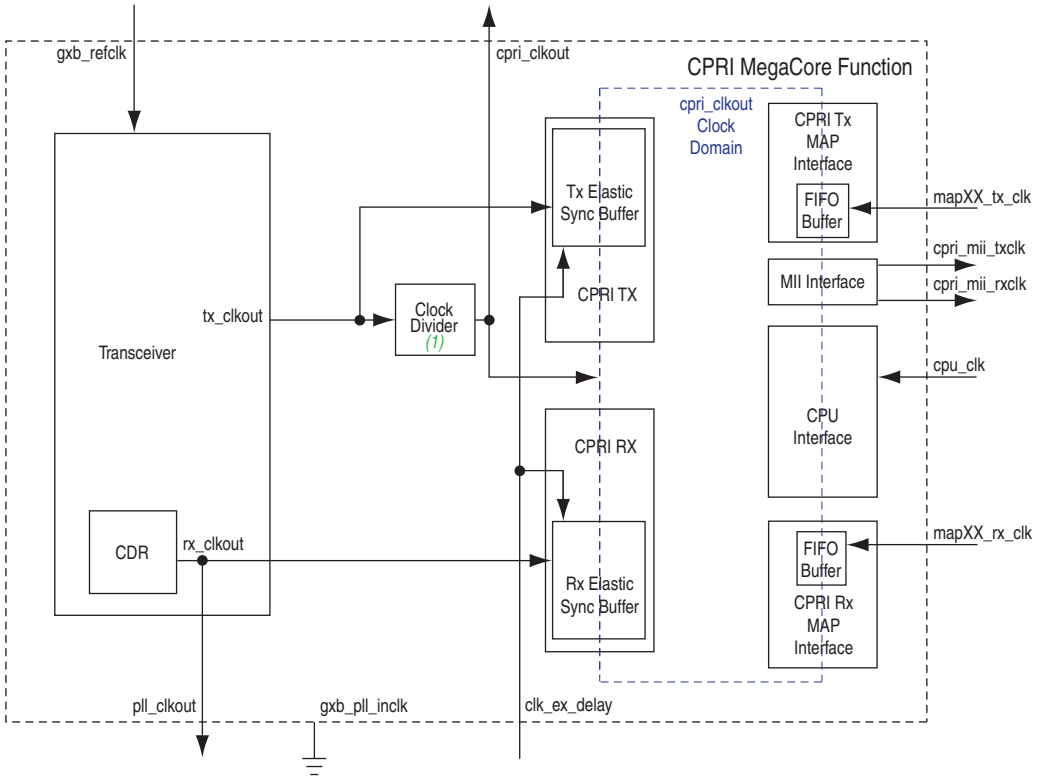


Note to Table 4–2:

(1) The clock divider factor depends on the device family. In device families with a factor of 1, the divider is not configured. Table 4–15 on page 4–53 lists the datapath width and clock divider by device family.

Figure 4–3 shows the clock diagram for a CPRI IP core configured as an REC master or as an RE master, unless the IP core is configured with CPRI line rate 9830.4 Mbps and targets an Arria V GT device.

Figure 4–3. CPRI IP Core Master Clocking Except for Arria V GT 9.8 Gbps Variations



Note to Table 4–3:

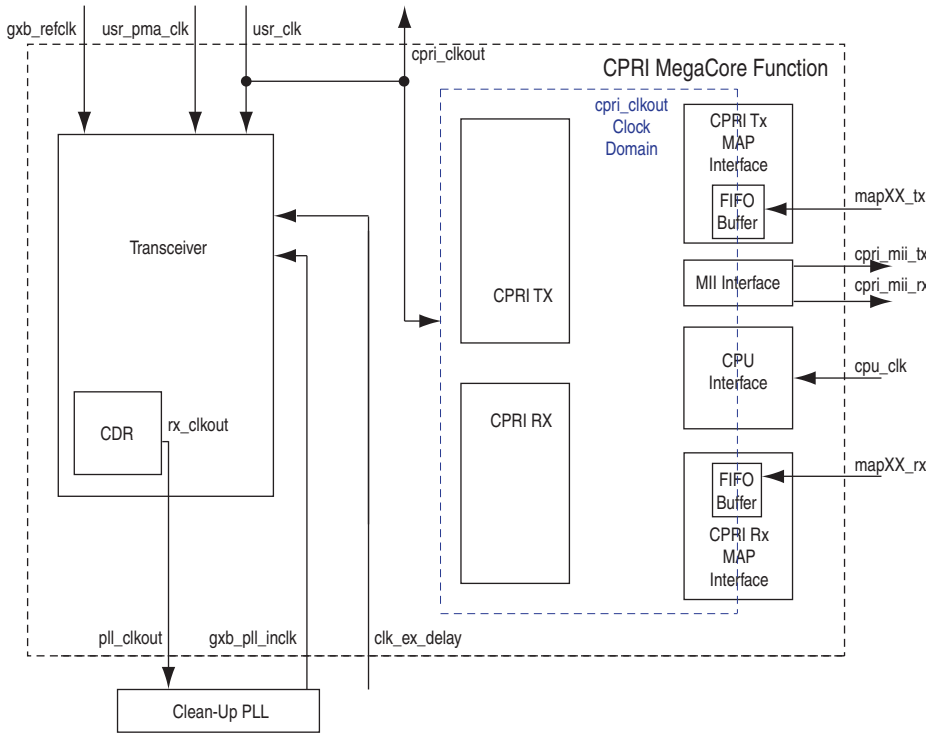
(1) The clock divider factor depends on the device family. In device families with a factor of 1, the divider is not configured. Table 4–15 on page 4–53 lists the datapath width and clock divider by device family.

Clock Diagram for CPRI IP Core Arria V GT Variations at 9830.4 Mbps

CPRI IP core variations with a CPRI line rate of 9830.4 Mbps that target an Arria V GT device have a different clocking scheme. These variations have no clock divider, and have neither an RX elastic buffer nor a TX elastic buffer. They use two additional input clock signals, `usr_clk` and `usr_pma_clk`. Table 6–15 on page 6–17 describes the requirements for these two input clock signals.

Figure 4–4 shows the clocking scheme for a CPRI IP core with a CPRI line rate of 9830.4 Mbps that targets an Arria V GT device. The figure notes describe the differences between the input clock requirements for the REC and RE master variations, which are configured in master clocking mode, and the input clock requirements for the RE slave variations, which are configured in slave clocking mode.

Figure 4–4. CPRI IP Core Clocking in Arria V GT 9.8 Gbps Variations (1), (2), (3)



Notes to Figure 4–4:

(1) The cleanup PLL is relevant only for variations configured in slave clocking mode.

(2) In variations configured in slave clocking mode, the `usr_clk` and `usr_pma_clk` input clocks must be driven by a common source from the cleanup PLL. For additional constraints these clocks require, refer to Table 6–15 on page 6–17.

(3) In variations configured in master clocking mode, you must tie the `gxb_pll_inclk` input signal low.

CPRI Communication Link Line Rates

The CPRI specification specifies line rates of n × 614.4 Mbps for various values of n. The CPRI IP core supports different ranges of line rates in different device families. Table 3–1 on page 3–2 lists the CPRI line rate support available in the different device families.

Table 4–2 shows the relationship between line rates, default transceiver reference clock (gxb_refclk) rates, parallel recovered clock (pll_clkout) rates, and internal clock (cpri_clkout) rates.

Table 4–2. CPRI Link Line Rates and Clock Rates for CPRI MegaCore Function (1)

Line Rate (Mbps)	Clock Frequency (MHz)						
	Default gxb_refclk Frequency (If line rate is supported)			cpri_clkout Frequency (If line rate is supported)	pll_clkout Frequency (If line rate is supported)		
	Arria II GX and Cyclone IV GX Devices	Arria II GZ and Stratix IV GX Devices	Arria V and Stratix V Devices		Arria II GX and Cyclone IV GX Devices	Arria II GZ, Arria V, and Stratix IV GX Devices	Stratix V Devices
614.4	61.44	61.44	122.88	15.36	61.44	61.44	61.44
1228.8	61.44	61.44	122.88	30.72	61.44	30.72	30.72
2457.6	122.88	61.44	122.88	61.44	122.88	61.44	61.44
3072	153.60	76.80	122.88	76.80	153.60	76.80	76.80
4915.2 (2)	245.76	122.88	122.88	122.88	245.76	122.88	122.88
6144 (2)	307.20	153.60	122.88	153.60	307.20	153.60	153.60
9830.4 (3)	—	—	122.88	245.76	—	122.88	245.76

Notes to Table 4–2:

(1) In this table, device families can be grouped with other device families that do not support all of the same CPRI line rates. The values apply only for supported CPRI line rates for each device family.

(2) The CPRI IP core does not support CPRI line rates 4915.2 Mbps and 6144 Mbps in variations that target Cyclone IV GX devices.

(3) The CPRI IP core supports CPRI line rate 9830.4 Mbps in variations that target Stratix V (GX or GT) and Arria V GT devices. The CPRI IP core does not support CPRI line rate 9830.4 Mbps for any other devices, including Arria V GX devices.

The cpri_clkout frequency depends only on the CPRI line rate. The pll_clkout frequency depends on the CPRI line rate and on the datapath width through the transceiver, except in Arria V and Stratix V devices. The datapath width is determined by device family, as shown in Table 4–15 on page 4–53.

The gxb_refclk clock is the incoming reference clock for the device transceiver's PLL. Altera allows you to program the transceiver to work with any of a set of gxb_refclk frequencies that the PLL in the transceiver can convert to the required internal clock speed for the CPRI IP core line rate. The parameter editor in which you configure the gxb_refclk frequency depends on the target device family for your CPRI IP core variation.

When you generate a CPRI IP core variation that targets an Arria II, Cyclone IV GX, or Stratix IV GX device, you generate an ALTGX megafunction with specific default settings. These default transceiver settings configure a transceiver that works correctly with the CPRI IP core when the input gxb_refclk clock has the frequency shown in Table 4–2. However, you can edit the ALTGX megafunction instance to specify a different gxb_refclk frequency that is more convenient for your design, for example, to enable you to use an existing clock in your system as the gxb_refclk reference clock.

When you generate a CPRI IP core variation that targets an Arria V or Stratix V device, you generate an Altera Deterministic Latency PHY IP core or Altera Native PHY IP core with specific default settings. However, you set the gxb_refclk frequency in the CPRI parameter editor. As described in Chapter 3, Parameter Settings, for these target devices the CPRI parameter editor provides a list of potential transceiver reference clock frequencies from which you select the frequency that is most convenient for your design.

Reset Requirements

The CPRI IP core has multiple independent reset signals.To reset the CPRI IP core completely, you must assert all the reset signals.

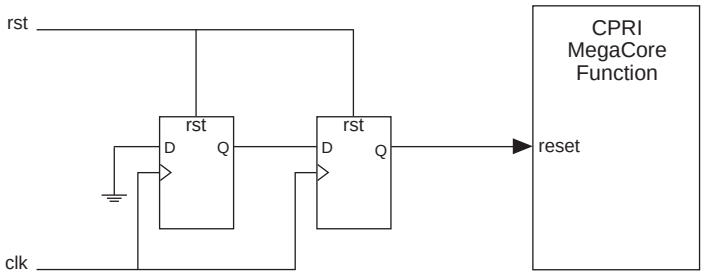
You can assert all reset signals asynchronously to any clock. However, each reset signal must be asserted for at least one full clock period of a specific clock, and be deasserted synchronously to the rising edge of that clock. For example, the CPU interface reset signal, cpu_reset, must be deasserted on the rising edge of cpu_clk. Table 4–3 lists the reset signals and their corresponding clock domains.

Table 4–3. Reset Signals and Corresponding Clock Domains

Reset Signal	Clock Domain	Description
reset	reconfig_clk	Resets the CPRI protocol interface. Drives the reset controller.
gxb_powerdown	—	Powers down and resets the high-speed transceiver block. For setup and hold times, refer to the relevant device handbook. This signal is not present in CPRI IP core variations that target an Arria V or Stratix V device.
reset_ex_delay	clk_ex_delay	Resets the extended delay measurement block.
config_reset	cpri_clkout	Resets the registers to their default values.
cpu_reset	cpu_clk	Resets the CPU interface.
mapN_rx_reset	mapN_rx_clk	Resets the MAP Channel N receiver block in FIFO or synchronous buffer MAP synchronization mode.
mapN_tx_reset	mapN_tx_clk	Resets the MAP Channel N transmitter block in FIFO or synchronous buffer MAP synchronization mode.

You must implement logic to ensure the minimal hold time and synchronous deassertion of each reset input signal to the CPRI IP core. Figure 4–5 shows a circuit that ensures these conditions for one reset signal.

Figure 4–5. Circuit to Ensure Synchronous Deassertion of Reset Signal



For more information about the requirements for reset signals, refer to Chapter 6, Signals.

The CPRI IP core has a dedicated reset control module to enforce the specific reset requirements of the high-speed transceiver module. This reset controller generates the recommended reset sequence for the transceiver. The `reset` signal controls the reset control module.

In Arria V and Stratix V devices, the Altera Deterministic Latency PHY IP core or Altera Native PHY IP core that is generated with the CPRI IP core implements the reset controller. In earlier device families, the reset control module is internal to the CPRI IP core, but external to the ALTGX megafunction instance generated with the CPRI IP core.

After reset, your software must perform link synchronization and other initialization tasks. For information about the required initialization sequence following CPRI IP core reset, refer to Appendix A, Initialization Sequence.

MAP Interface

The CPRI IP core MAP interface comprises the individual antenna-carrier interfaces, or data channels, through which the CPRI IP core transfers IQ sample data to and from the RF implementation. The MAP interface is implemented as an incoming and an outgoing Avalon-ST interface. The Avalon-ST interface provides a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface.

For information about the Avalon-ST interface, refer to *Avalon Interface Specifications*.

The CPRI IP core communicates with the RF implementations (antenna-carriers) through multiple AxC interfaces, or data channels. A CPRI IP core configured with a MAP interface module can have as many as 24 data channels, and as few as one data channel. If a CPRI IP core is configured with zero data channels, it does not have a MAP interface module. The **Number of antenna/carrier interfaces** value you set in the parameter editor determines the number of channels in your CPRI IP core configuration. Each data channel communicates with the corresponding RF implementation using two 32-bit Avalon-ST interfaces, one interface for incoming communication and one interface for outgoing communication.

The MAP interface module controls transmission and reception of data on the AxC interfaces.

This section contains the following topics:

- MAP Interface Mapping Modes
- MAP Receiver Interface
- MAP Transmitter Interface

MAP Interface Mapping Modes

The CPRI IP core supports basic and advanced MAP interface mapping modes.

In the basic mapping mode, all of the AxC interfaces use the same sample rate and sample width, and the uplink and downlink sample rates are identical.

In the advanced mapping modes, different data channels can use different sample rates, and the sample rates need not be integer multiples of 3.84 MHz. However, all data channels use the same sample width.

If you select **All** as the value for **Mapping mode(s)** in the CPRI parameter editor, the `map_mode` field of the `CPRI_MAP_CONFIG` register determines the mapping mode your CPRI IP core implements currently. Otherwise, the value you specify for this parameter determines the single mapping mode your CPRI IP core implements.

The CPRI IP core supports the following MAP interface mapping modes:

- Basic mapping mode—This mode is programmed with the value of 2'b00 in the `map_mode` register field, and is described in the following section.
- Advanced 1—This mode is programmed with the value of 2'b01 in the `map_mode` register field, and is described in Appendix C, Advanced AxC Mapping Modes.
- Advanced 2—This mode is programmed with the value of 2'b10 in the `map_mode` register field, and is described in Appendix C, Advanced AxC Mapping Modes.
- Advanced 3—This mode is programmed with the value of 2'b11 in the `map_mode` register field, and is described in Appendix C, Advanced AxC Mapping Modes.

Basic AxC Mapping Mode

The basic mapping mode supports the LTE/E-UTRA and UMTS/WCDMA standards. This mapping mode is implemented when you configure and program your CPRI IP core in either of the following ways:

- If you select **Basic** as the value for **Mapping mode(s)** in the CPRI parameter editor.

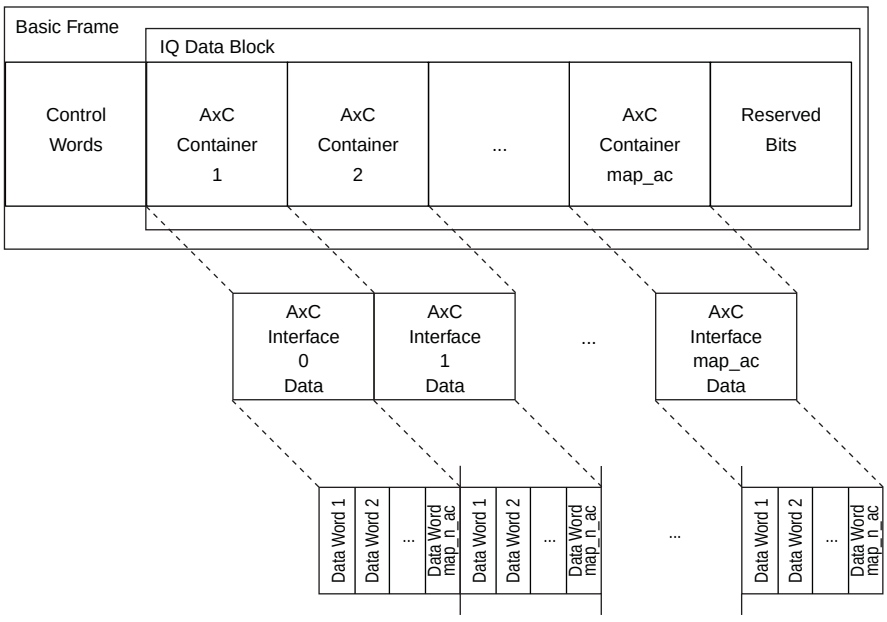
- If you select **All** as the value for **Mapping mode(s)** in the CPRI parameter editor and you program the `map_mode` field of the `CPRI_MAP_CONFIG` register with the value of 2'b00.

In this basic mapping mode, all of the AxC interfaces use the same sample rate and sample width. The CPRI IP core supports sample rates of 3.84×10^6 through 30.72×10^6 ($3.84 \times 10^6 \times 8$) samples per second, in increments of 3.84×10^6, and sample widths of 15 bits and 16 bits. The uplink and downlink sample rates are identical.

In this mode, the `map_ac` field of the `CPRI_MAP_CNT_CONFIG` register specifies the number of active data channels, that is, those that have a corresponding AxC container in the IQ data block of each basic frame. This number must be less than or equal to the N_MAP value you selected for **Number of antenna/carrier interfaces** in the parameter editor, which is the number of channels configured in the CPRI IP core instance. The `map_n_ac` field of the `CPRI_MAP_CNT_CONFIG` register holds the oversampling factor for the data channels. This value is an integer from 1 to 8. The sample rate—number of samples per second—is the product of 3.84×10^6 and the oversampling factor.

In the basic mapping mode, AxC containers are packed in the IQ data block in the packed position (Option 1) illustrated in Section 4.2.7.2.3 of the CPRI V4.2 Specification. Figure 4–6 shows how the AxC containers map to the individual active data channels. The oversampling factor is the number of 32-bit data words in each AxC container.

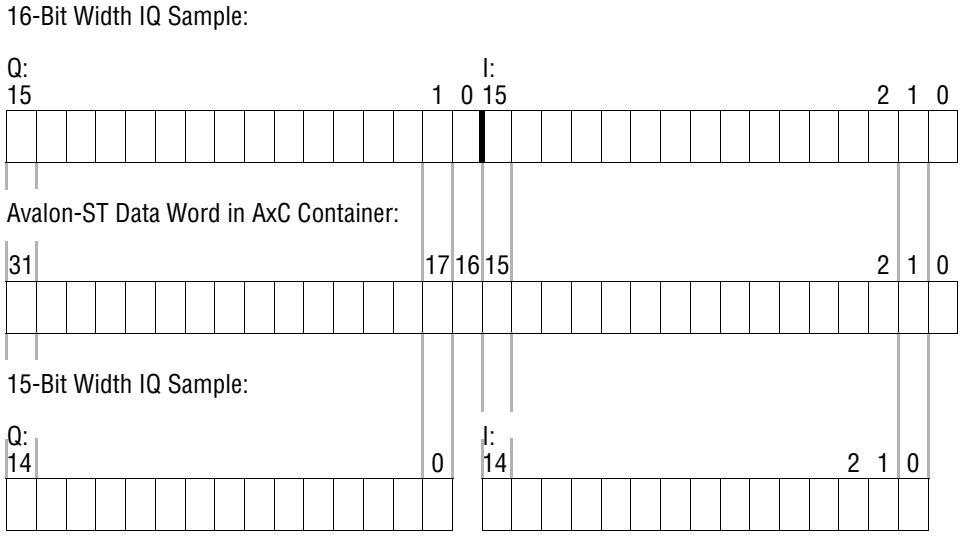
Figure 4–6. CPRI Basic Mapping Mode



The CPRI IP core does not support AxC interface reordering. When the value of `map_ac` is less than N_MAP, the first `map_ac` AxC interfaces, of the existing N_MAP interfaces, are active. Note that an active AxC interface transmits and receives data on its data channel based on the values of the relevant `map_rx_enable` bit of the `CPRI_IQ_RX_BUF_CONTROL` register and the relevant `map_tx_enable` bit of the `CPRI_IQ_TX_BUF_CONTROL` register. Any data in an AxC container for an active but disabled channel is ignored, and an incoming AxC container designated from a disabled channel is ignored.

The `map_15bit_mode` field of the `CPRI_MAP_CONFIG` register specifies the sample width. The sample width is the number of significant bits —15 or 16—in each 16-bit half (originally, I- or Q-sample) of the 32-bit data word on the Avalon-ST data channel. In 15-bit mode, the least significant bit in each half of the 32-bit word is ignored when received from the data channel on input signal `mapN_tx_data[31:0]`, and is set to 0 when transmitted on the data channel in output signal `mapN_rx_data[31:0]`. Therefore, bit 15 and bit 31 of the data word correspond to bit 14 of the I and Q samples, respectively; bit 1 and bit 17 of the data word correspond to bit 0 of the I and Q samples, respectively; and bits 0 and 16 of the data word are ignored. In 16-bit mode, bit 15 and bit 31 of the data word correspond to bit 15 of the I and Q samples, respectively, and bit 0 and bit 16 of the data word correspond to bit 0 of the I and Q samples, respectively. Figure 4–7 shows the bit correspondence for both sample widths.

Figure 4–7. Bit Correspondence Between IQ Sample and 32-Bit Avalon-ST Data



You set the oversampling factor to match the frequency of your active data channels. The CPRI line rate determines the number of bits in the IQ data block of each basic frame. If your CPRI IP core has a high line rate and a low oversampling factor, it can accommodate a larger number of active data channels than if the line rate were lower or the oversampling factor higher.

In 15-bit mode, inside the CPRI IP core, bits 0 and 16 of the Avalon-ST data are absent from the compact IQ data word representation. Therefore, despite the fact that in 15-bit mode the IQ data goes out on the data channel in 32-bit words, formatted as

shown in Figure 4–7, the maximum number of active data channels is higher in 15-bit mode. Table 4–4 shows the correspondence between these frequency factors in 16-bit mode, and Table 4–5 shows the correspondence between these factors in 15-bit mode.

Table 4–4. Maximum Number of Active Data Channels in 16-Bit Mode

CPRI Line Rate (Mbps)	Number of Bits in IQ Data Block	Maximum Number of Active Data Channels in 16-Bit Mode					
		Data Channel Bandwidth LTE (MHz)	2.5	5	10	15	20
		Sample Rate (10^6 Sample/Sec)	3.84	7.68	15.36	23.04	30.72
614.4	120		3	1	—	—	—
1228.8	240		7	3	2	1	—
2456.7	480		15	7	3	2	1
3072	600		18	9	4	3	2
4915.2	960		30 (1)	15	7	5	3
6144	1200		37 (1)	18	9	6	4
9830.4	1920		60 (1)	30 (1)	15	10	7

Note to Table 4–4:

(1) The maximum number of data channels supported by the CPRI IP core is 24. The numbers in the table that are larger than 24 are hypothetical; the CPRI IP core cannot implement them.

Table 4–5. Maximum Number of Active Data Channels in 15-Bit Mode

CPRI Line Rate (Mbps)	Number of Bits in IQ Data Block	Maximum Number of Active Data Channels in 15-Bit Mode					
		Data Channel Bandwidth LTE (MHz)	2.5	5	10	15	20
		Sample Rate (10^6 Sample/Sec)	3.84	7.68	15.36	23.04	30.72
614.4	120		4	2	1	—	—
1228.8	240		8	4	2	1	1
2456.7	480		16	8	4	2	2
3072	600		20	10	5	3	2
4915.2	960		32 (1)	16	8	5	4
6144	1200		40 (1)	20	10	6	5
9830.4	1920		64 (1)	32 (1)	16	10	8

Note to Table 4–5:

(1) The maximum number of data channels supported by the CPRI IP core is 24. The numbers in the table that are larger than 24 are hypothetical; the CPRI IP core cannot implement them.

In 16-bit mode, the total number of bits in all the AxC containers in a basic frame is

$2 \times 16 \times \texttt{map\_n\_ac} \times \texttt{map\_ac}$

In 15-bit mode, the total number of bits in all the AxC containers in a basic frame is

$2 \times 15 \times \texttt{map\_n\_ac} \times \texttt{map\_ac}$

This value must be no larger than the number of bits in the IQ data block. The number of bits in an IQ data block depends on the CPRI line rate, as shown in Table 4–4 and Table 4–5.

If the combination of CPRI line rate, `map_n_ac` value, and `map_ac` value requires more data bits than the number of data bits that fit in the IQ data block, the data for the first active data channels is transferred correctly, but the data for data channels beyond the number indicated in Table 4–4 or Table 4–5 is not transferred correctly.

The following CPRI IP core registers are ignored in basic mapping mode:

- `CPRI_MAP_TBL_CONFIG` register (Table 7–33 on page 7–15)
- `CPRI_MAP_TBL_INDEX` register (Table 7–34 on page 7–16)
- `CPRI_MAP_TBL_RX` register (Table 7–35 on page 7–16)
- `CPRI_MAP_TBL_TX` register (Table 7–36 on page 7–17)

Advanced AxC Mapping Modes

The CPRI IP core provides advanced AxC mapping modes to support the following mapping methods from the CPRI V4.2 Specification:

- Method 1: IQ Sample Based, described in Section 4.2.7.2.5 of the CPRI V4.2 Specification.
- Method 3: Backward Compatible, described in Section 4.2.7.2.7 of the CPRI V4.2 Specification.

In the advanced mapping modes, different data channels can use different sample rates, and the sample rates need not be integer multiples of 3.84 MHz. However, all data channels use the same sample width.

Your CPRI IP core implements one of the advanced AxC mapping modes when you configure and program your CPRI IP core in any of the following ways:

- If you select **Advanced 1**, **Advanced 2**, or **Advanced 3** as the value for **Mapping mode(s)** in the CPRI parameter editor.
- If you select **All** as the value for **Mapping mode(s)** in the CPRI parameter editor and you program the `map_mode` field of the `CPRI_MAP_CONFIG` register with the value of 2'b01, 2'b10, or 2'b11.

For more information about the advanced AxC mapping modes in the Altera CPRI IP core, refer to Appendix C, Advanced AxC Mapping Modes.

MAP Receiver Interface

The CPRI IP core MAP receiver interface presents the IQ data that the CPRI IP core unloads from the CPRI frame received on the CPRI link. The MAP receiver implements an Avalon-ST interface protocol. Refer to "MAP Receiver Signals" on page 6–1 for details of the interface communication signals.

The MAP receiver interface presents the IQ data on each antenna-carrier interface according to one of three different synchronization modes. The synchronization mode is determined by your selection in the CPRI parameter editor and by the value you program in the map_rx_sync_mode field of the CPRI_MAP_CONFIG register (Table 7–31 on page 7–14), as shown in Table 4–6.

Table 4–6. MAP Rx Synchronization Mode Determined by CPRI_MAP_CONFIG Register Bits

SYNC_MAP (1)	map_rx_sync_mode (register bit [2])	Rx Synchronization Mode
0	0	FIFO mode (page 4–17)
0	1	Synchronous buffer mode (page 4–18)
1	— (2)	Internally-clocked mode (page 4–20)

Notes to Table 4–6:

(1) You determine the value of SYNC_MAP when you generate your CPRI IP core. Refer to Chapter 3, Parameter Settings.

(2) When SYNC_MAP has the value of 1, the value in the map_rx_sync_mode bit of the CPRI_MAP_CONFIG register is ignored.

Table 4–7 lists the clocks for the AxC interfaces in the different Rx synchronization modes.

Table 4–7. MAP Rx Interface Clocks Determined by Rx Synchronization Mode

Rx Synchronization Mode	AxC Channel Clocks
FIFO mode	Each AxC Rx interface is clocked by its own mapN_rx_clk clock driven by the application.
Synchronous buffer mode	
Internally-clocked mode	Every AxC interface is clocked by the CPRI IP core clock, cpri_clkout.

You determine the AxC interface clocks when you turn the **Enable MAP interface synchronization with core clock parameter** on (SYNC_MAP = 1) or off (SYNC_MAP = 0) in the CPRI parameter editor before you generate your CPRI IP core.

MAP Receiver Interface Signals in Different Synchronization Modes

The different CPRI IP core MAP synchronization modes use different interface signals. Table 4–8 lists the MAP receiver interface signals used in each of these modes. Table notes indicate the correct interpretation of the different symbols.

Table 4–8. MAP Receiver Interface Signals by Synchronization Mode (1) (Part 1 of 2)

Signal Name	Direction	Available in Synchronization Mode		
		FIFO	Synchronous Buffer	Internally Clocked
map{23…0}_rx_clk	Input	✓	✓	— (2)
map{23…0}_rx_reset	Input	✓	✓	— (2)
map{23…0}_rx_ready	Input	✓	1 (3)	— (2), (4)
map{23…0}_rx_data[31:0]	Output	✓	✓	✓
map{23…0}_rx_valid	Output	✓	— (2)	✓
map{23…0}_rx_resync	Input	— (2)	✓	— (2)

Table 4–8. MAP Receiver Interface Signals by Synchronization Mode (1) (Part 2 of 2)

Signal Name	Direction	Available in Synchronization Mode		
		FIFO	Synchronous Buffer	Internally Clocked
map{23…0}_rx_start	Output	— (2)	— (2)	✓
map{23…0}_rx_status_data [2:0]	Output	✓	✓	✓

Notes to Table 4–8:

(1) A checkmark indicates the signal is used in a synchronization mode, and a dash indicates the signal is not used in that synchronization mode.

(2) An entry with a dash indicates a signal that does not participate in the MAP receiver interface communication in this synchronization mode. The signal is either not present in the configuration or is ignored. An input signal that is ignored is ignored by the CPRI IP core. An output signal that is ignored should be ignored by the application. Refer to Table 6–1 on page 6–1 for information about the case that is relevant for each signal.

(3) A zero or one indicates the application must hold this input signal low or high, respectively.

(4) Altera recommends that you tie the mapN_rx_ready signals high or low in your internally-clocked variation, rather than leave them floating.

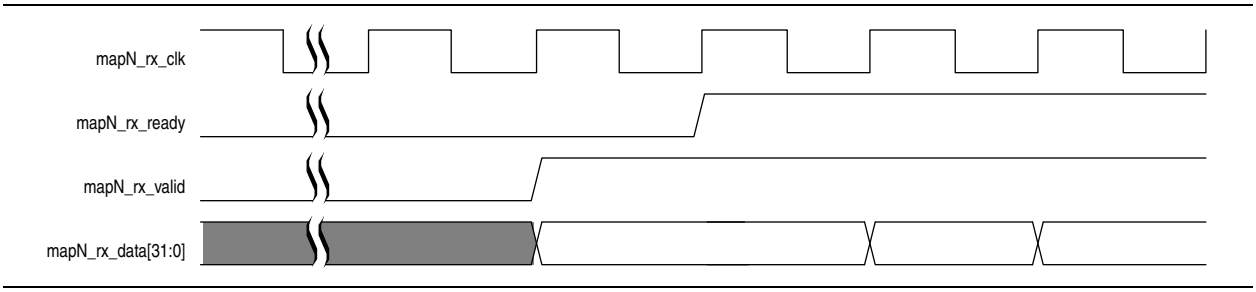
For descriptions of the signals in Table 4–8, refer to Table 6–1 on page 6–1 and to the following sections.

MAP Receiver in FIFO Mode

In FIFO mode, each data channel, or AxC interface, is clocked by an application-driven clock mapN_rx_clk, and has an output data-available signal, mapN_rx_valid. Each AxC interface N asserts its mapN_rx_valid signal when it has data available to send on this data channel—when the buffer level is above the threshold indicated in the CPRI_MAP_RX_READY_THR register.

For details about the behavior of the individual signals in FIFO mode, refer to "MAP Receiver Signals" on page 6–1. Figure 4–8 shows the typical behavior of the MAP Rx signals in this synchronization mode.

Figure 4–8. MAP Receiver Interface in FIFO Mode



When the application is ready to receive data on the data channel, it asserts the mapN_rx_ready signal. While the CPRI IP core asserts the mapN_rx_valid signal and the mapN_rx_ready signal is not asserted, the CPRI IP core holds the data value on mapN_rx_data[31:0]. The application must assert the mapN_rx_ready signal before the mapN Rx buffer overflows, to avoid data corruption. While the mapN_rx_ready signal

is not yet asserted, the mapN Rx buffer continues to fill. When it overflows, the new data overwrites current data in the mapN Rx buffer. Each mapN Rx buffer is implemented as a circular buffer, so the data is overwritten starting at the current head of the mapN Rx buffer, that is, starting from the initial data not yet sent out on the data channel.

FIFO-based communication is simple but does not allow easy control of buffer delay. The delay through each mapN Rx buffer depends on your programmed threshold value and the application. Data is not sent to a data channel before the buffer threshold is reached, so the delay through the buffer depends on the fill level. Each AxC interface has the same buffer threshold, but each Rx buffer reaches that threshold independently.

MAP Receiver in Synchronous Buffer Mode

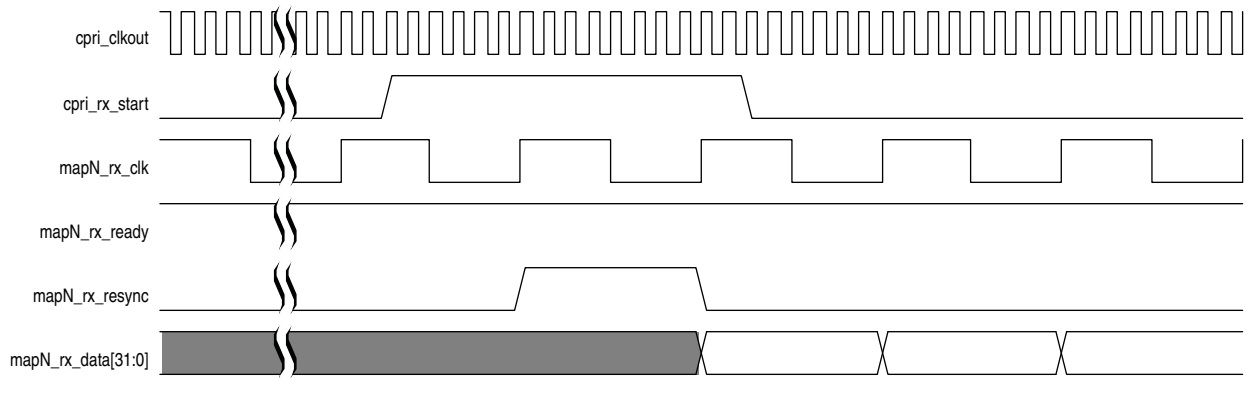
In synchronous buffer mode, each AxC interface has a resynchronization signal, `mapN_rx_resync`. The application that controls the data channel asserts its resynchronization signal synchronously with the `mapN_rx_clk` clock. After the application asserts the resynchronization signal, it begins reading data on the `mapN_rx_data[31:0]` data bus for the individual AxC interface.

In synchronous buffer mode, the application should ignore the `mapN_rx_valid` output signals and hold the `mapN_rx_ready` input signals high. The CPRI IP core does assert the `mapN_rx_valid` output signals in response to the `mapN_rx_ready` signals. If the application does not hold the `mapN_rx_ready` input signals high, the CPRI IP core MAP Rx interface does not function correctly.

For details about the behavior of the individual signals in synchronous buffer mode, refer to "MAP Receiver Signals" on page 6–1.

Figure 4–9 shows the behavior of the MAP Rx signals in synchronous buffer mode. In this example, the CPRI line rate is 2457.6 Mbps. The `cpri_rx_start` signal is asserted for the duration of a single frame, and the CPRI line rate determines the duration of a basic frame in `cpri_clkout` cycles. At 2457.6 Mbps, a basic frame is 16 `cpri_clkout` cycles. At this line rate, as shown in Table 4–2 on page 4–8, the `cpri_clkout` frequency is 61.44 MHz. The `mapN_rx_clk` frequency is 7.68 MHz (oversampling rate 2), approximately 0.125 times the `cpri_clkout` frequency.

Figure 4–9. MAP Receiver Interface in Synchronous Buffer Mode

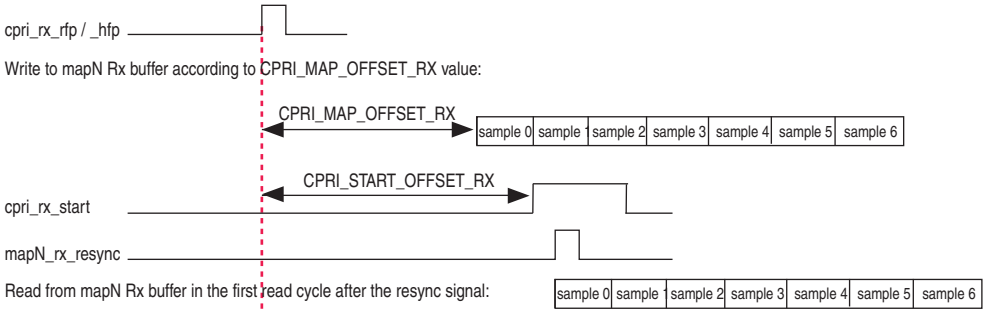


To ensure IP core control over the resynchronization signal timing, Altera recommends that your application trigger the mapN_rx_resync signal with the CPRI IP core output signal cpri_rx_start. The CPRI AUX interface asserts the cpri_rx_start signal according to the offset value specified in the user-programmable CPRI_START_OFFSET_RX register.

Asserting the resynchronization signal ensures correct alignment between the RF implementation and the CPRI basic frame at the appropriate offset from the start of the 10 ms radio frame. You control the mapN_rx_resync signals to ensure that the IP core accommodates your application-specific constraints.

Figure 4–10 shows the roles of the CPRI_START_OFFSET_RX and CPRI_MAP_OFFSET_RX registers in ensuring correct alignment.

Figure 4–10. User-Controlled Delays to the AxC Data Channels in Rx Synchronous Buffer Mode



The values programmed in the CPRI_START_OFFSET_RX register control the assertion of the cpri_rx_start signal. The values in the start_rx_offset_z, start_rx_offset_x, and start_rx_offset_seq fields specify a hyperframe number, basic frame number, and word number in the basic frame, respectively, within the 10 ms frame.

The CPRI master transmitter loads the AxC container block on the CPRI link at a specific location in the 10 ms frame; the system programs the information for this location in the CPRI_START_OFFSET_RX register. The CPRI slave receiver learns the location of the AxC container block from the CPRI_START_OFFSET_RX register.

For example, if the CPRI_START_OFFSET_RX register is programmed with the value 0x00020001, the CPRI receiver asserts the cpri_rx_start signal at word index 2 of basic frame 1 of hyperframe 0 in the 10ms frame. The data channel application samples the cpri_rx_start signal, detects it is asserted, and then synchronizes the received IQ sample to the RX MAP AxC interface by asserting the mapN_rx_resync signal. Assertion of the mapN_rx_resync signal resets the read pointer of current antenna-carrier interface (mapN) Rx buffer to zero. The mapN_rx_data can safely be sampled by the data channel one cycle after the mapN_rx_resync signal is asserted.

The offset programmed in the CPRI_MAP_OFFSET_RX register tells the MAP receiver interface when to reset the write pointer of the Rx buffer: when the internal counters match the value in the CPRI_MAP_OFFSET_RX register, the write pointer resets. If the offset in this register has the value of zero, the write pointer resets at the start of every 10 ms radio frame. After the MAP receiver block resets the write pointer, it begins transferring IQ data from the CPRI frame to the Rx buffer.

In advanced mapping modes, the K counter is reset to zero at the same time, so that it advances from zero with the transfer of the data to the MAP Rx buffer, tracking the packing of the CPRI data contents into the AxC container block.

Because the mapN Rx buffer should not be read before it is written, the offset specified in the CPRI_MAP_OFFSET_RX register must precede the offset specified in the CPRI_START_OFFSET_RX register. The CPRI IP core informs you of buffer overflow and underflow (in the CPRI_IQ_RX_BUF_STATUS register described in Table 7–48 on page 7–21, as reported in the mapN_rx_status_data output signals described in Table 6–1 on page 6–1), but it does not prevent them from occurring. Altera recommends that you implement a separate tracking protocol to ensure you do not overflow or underflow the mapN Rx buffer.

You set the values in the CPRI_START_OFFSET_RX and CPRI_MAP_OFFSET_RX registers to specify the timeslot in the 10 ms radio frame in which your application expects to sample the data on the antenna-carrier interface.

In synchronous buffer mode, because programmed offsets control the mapN Rx buffer pointers, the delay through each mapN Rx buffer can be quantified.

In synchronous buffer mode, Altera recommends that you use sample rates that are integer multiples of 3.84 MHz, or for implementing the WiMAX protocol, that you use sample rates that provide the exact frequency required.

MAP Receiver in the Internally-Clocked Mode

In the internally-clocked mode, cpri_clkout drives the antenna-carrier interfaces, in contrast to the other two synchronization modes in which the antenna-carrier interfaces are clocked by the input mapN_rx_clk clocks. Each AxC interface has only a two-stage buffer, and data passes quickly from the MAP block out to the individual data channels. Each AxC interface has a ready output signal, mapN_rx_start. Each AxC interface asserts its ready signal when it first has data ready to transmit on this data channel.

The CPRI IP core asserts the mapN_rx_start and mapN_rx_valid signals simultaneously, synchronously with the cpri_clkout clock, when it makes data available on the mapN_rx_data[31:0] data bus for the individual AxC interface. It may also assert mapN_rx_valid before valid data is available. In that case, it does not assert mapN_rx_start. In each 10 ms radio frame, for each antenna-carrier channel N, the application should ignore the mapN_rx_valid and mapN_rx_data signals until the CPRI IP core asserts the mapN_rx_start signal. Refer to Figure 4–11 for an example.

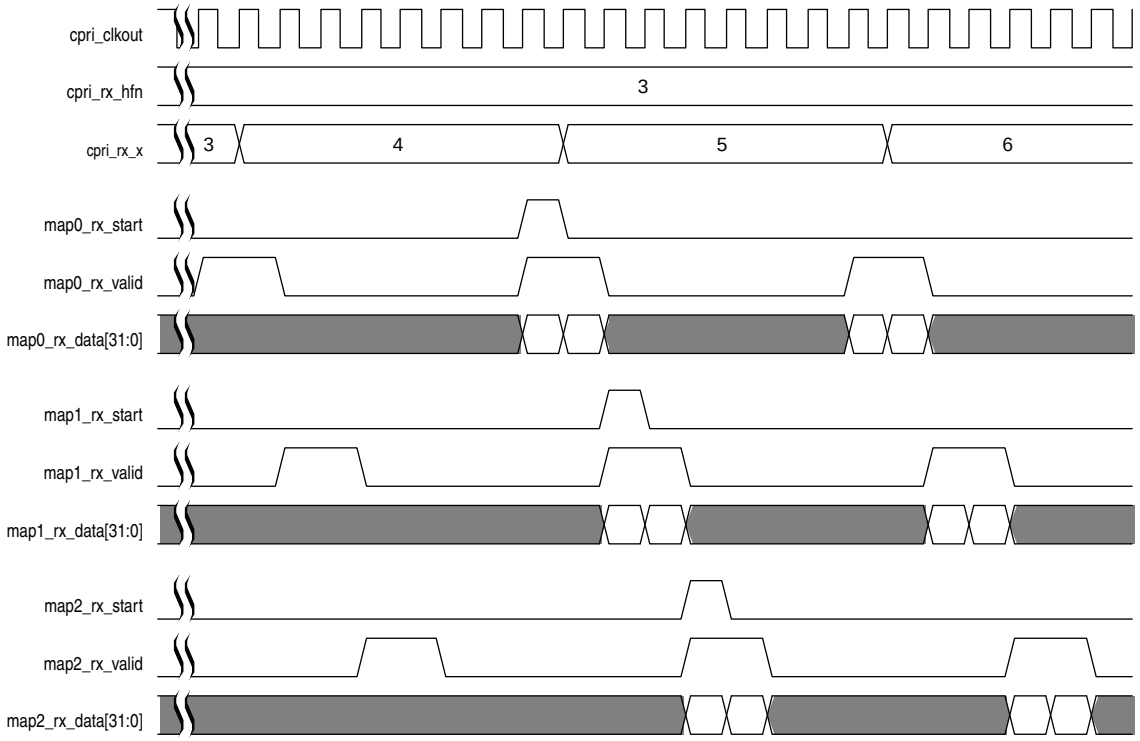
For details about the behavior of the individual signals in the internally-clocked mode, refer to "MAP Receiver Signals" on page 6–1.

Figure 4–11 shows an example of the behavior of the MAP Rx signals in this synchronization mode in the basic mapping mode (map_mode = 2'b00). The example CPRI IP core is configured and programmed with the following features:

- CPRI line rate is 1228.8 Mbps. Therefore the duration of a basic frame is 8 cpri_clkout cycles.
- Three active antenna-carrier interfaces.

- In the CPRI_MAP_OFFSET_RX register, the cpri_rx_offset_z field has the value of 3 and the cpri_rx_offset_x field has the value of 4.

Figure 4–11. MAP Receiver Interface in the internally-Clocked Mode



In Figure 4–11, the map0_rx_start signal pulses synchronously with the first rising edge of map0_rx_valid following the CPRI frame offset specified in the CPRI_MAP_OFFSET_RX register. The mapN_rx_valid signals are asserted in round-robin order, following the basic mapping mode.

The internally-clocked mode is useful only with the basic mapping mode. The advantage of the advanced mapping modes is their support for different clocks on different antenna-carrier interfaces, a feature not available with the internally-clocked synchronization mode.

MAP Transmitter Interface

The MAP transmitter interface receives data from the data channels and passes it to the CPRI protocol interface to transmit on the CPRI link. The MAP transmitter implements an Avalon-ST interface protocol. Refer to "MAP Transmitter Signals" on page 6–3 for details of the interface communication signals.

MAP transmitter communication on the individual data map interfaces coordinates the transfer of data according to one of three different synchronization modes. The synchronization mode is determined by your selection in the CPRI parameter editor and by the value you program in the map_tx_sync_mode field of the CPRI_MAP_CONFIG register (Table 7–31 on page 7–14), as shown in Table 4–9.

Table 4–9. MAP Tx Synchronization Mode Determined by CPRI_MAP_CONFIG Register Bits

SYNC_MAP (1)	map_tx_sync_mode (register bit [3])	Tx Synchronization Mode
0	0	FIFO mode (page 4–23)
0	1	Synchronous buffer mode (page 4–24)
1	— (2)	Internally-clocked mode (page 4–26)

Notes to Table 4–9:

(1) You determine the value of SYNC_MAP when you generate your CPRI IP core. Refer to Chapter 3, Parameter Settings.

(2) When SYNC_MAP has the value of 1, the value in the map_tx_sync_mode bit of the CPRI_MAP_CONFIG register is ignored.

Table 4–10 lists the clocks for the AxC interfaces in the different Tx synchronization modes.

Table 4–10. MAP Tx Interface Clocks Determined by Tx Synchronization Mode

Tx Synchronization Mode	AxC Channel Clocks
FIFO mode	Each AxC Tx interface is clocked by its own mapN_tx_clk clock driven by the application.
Synchronous buffer mode	
Internally-clocked mode	Every AxC interface is clocked by the CPRI IP core clock, cpri_clkout.

You determine the AxC interface clocks when you turn the **Enable MAP interface synchronization with core clock parameter** on (SYNC_MAP = 1) or off (SYNC_MAP = 0) in the CPRI parameter editor before you generate your CPRI IP core.

MAP Transmitter Interface Signals in Different Synchronization Modes

The different CPRI IP core MAP synchronization modes use different interface signals. Table 4–11 lists the MAP transmitter interface signals used in each of these modes. Table notes indicate the correct interpretation of the different symbols.

Table 4–11. MAP Transmitter Interface Signals by Synchronization Mode (1) (Part 1 of 2)

Signal Name	Direction	Available in Synchronization Mode		
		FIFO	Synchronous Buffer	Internally Clocked
map{23…0}_tx_clk	Input	✓	✓	— (2)
map{23…0}_tx_reset	Input	✓	✓	— (2)
map{23…0}_tx_valid	Input	✓	✓	✓
map{23…0}_tx_data[31:0]	Input	✓	✓	✓
map{23…0}_tx_ready	Output	✓	— (2)	✓
map{23…0}_tx_resync	Input	— (2)	✓	— (2)

Table 4–11. MAP Transmitter Interface Signals by Synchronization Mode (1) (Part 2 of 2)

Signal Name	Direction	Available in Synchronization Mode		
		FIFO	Synchronous Buffer	Internally Clocked
map{23…0}_tx_status_data [2:0]	Output	✓	✓	✓

Notes to Table 4–11:

(1) A checkmark indicates the signal is used in a synchronization mode, and a dash indicates the signal is not used in that synchronization mode.

(2) An entry with a dash indicates a signal that does not participate in the MAP receiver interface communication in this synchronization mode. The signal is either not present in the configuration or is ignored. An input signal that is ignored is ignored by the CPRI IP core. An output signal that is ignored should be ignored by the application. Refer to Table 6–2 on page 6–4 for information about the case that is relevant for each signal.

For descriptions of the signals in Table 4–11, refer to Table 6–2 on page 6–4 and to the following sections.

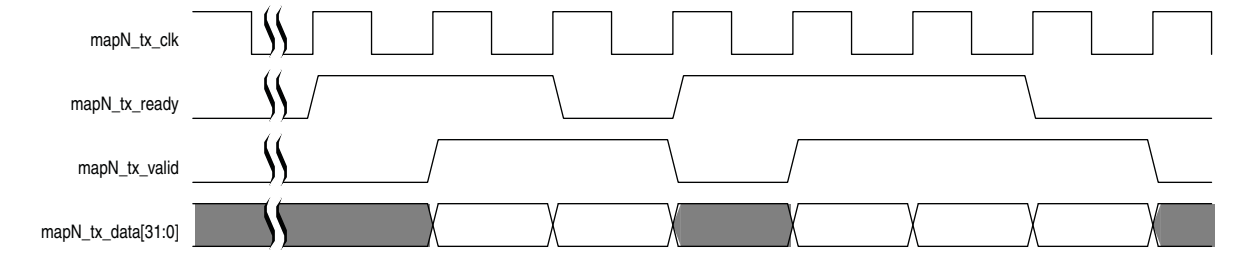
MAP Transmitter in FIFO Mode

In FIFO mode, each data channel, or AxC interface, has an output ready signal, mapN_tx_ready. Each AxC interface asserts its ready signal when it is ready to receive data on this data channel for transmission to the CPRI protocol interface—when the buffer level is at or below the threshold indicated in the CPRI_MAP_TX_READY_THR register.

After the CPRI IP core asserts the mapN_tx_ready signal, the application is expected to respond by asserting the mapN_tx_valid signal and presenting data on mapN_tx_data. In every mapN_tx_clk cycle immediately following a mapN_tx_clk cycle in which mapN_tx_ready is (becomes or remains) asserted, the application can present valid data on mapN_tx_data, as prescribed by the Avalon-ST specification with READY_LATENCY value 1.

For details about the behavior of the individual signals in FIFO mode, refer to "MAP Transmitter Signals" on page 6–3. Figure 4–12 shows the expected typical behavior of the MAP Tx signals in this synchronization mode.

Figure 4–12. MAP Transmitter Interface in FIFO Mode



FIFO-based communication is simple but does not allow easy control of buffer delay. The delay through each mapN Tx buffer depends on your programmed threshold value and the application. Data is not read from the mapN Tx buffer until the buffer threshold is reached, so the delay through the buffer depends on the fill level. Each AxC interface has the same buffer threshold, but each Tx buffer reaches that threshold independently.

MAP Transmitter in Synchronous Buffer Mode

In the synchronized communication, called synchronous buffer mode, each AxC interface has an incoming resynchronization signal, mapN_tx_resync. Application software asserts this resynchronization signal synchronously with the mapN_tx_clk clock. When the application software asserts the resynchronization signal, it also asserts the mapN_tx_valid signal and begins sending valid data on the mapN_tx_data[31:0] data bus for the individual AxC interface.

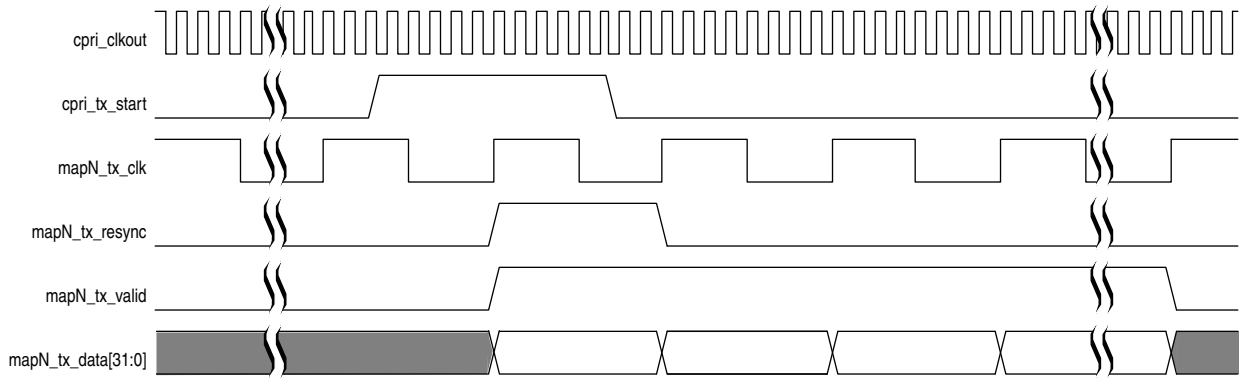
In synchronous buffer mode, the application should ignore the mapN_tx_ready output signals. However, it should assert the mapN_tx_valid input signals when sending valid data. The CPRI IP core holds the mapN_tx_ready output signals high. The application must assert the mapN_tx_valid input signals when or immediately after it asserts the mapN_tx_resync signals. However, if the application does not assert the mapN_tx_valid input signals in the same cycle as the mapN_tx_resync signals, and subsequently reasserts mapN_tx_resync while mapN_tx_valid is still high, data in transition through the MAP Tx interface buffer is lost.

Altera recommends that your application assert the mapN_tx_valid input signals when it asserts the mapN_tx_resync signals.

For details about the behavior of the individual signals in synchronous buffer mode, refer to "MAP Transmitter Signals" on page 6–3.

Figure 4–13 shows the expected typical behavior of the MAP Tx signals in this synchronization mode. In this example, the CPRI line rate is 2457.6 Mbps. The cpri_tx_start signal is asserted for the duration of a single frame, and the CPRI line rate determines the duration of a basic frame in cpri_clkout cycles. At 2457.6 Mbps, a basic frame is 16 cpri_clkout cycles. At this line rate, as shown in Table 4–2 on page 4–8, the cpri_clkout frequency is 61.44 MHz. The mapN_tx_clk frequency is 7.68 MHz (oversampling rate 2), approximately 0.125 times the cpri_clkout frequency.

Figure 4–13. MAP Transmitter Interface in Synchronous Buffer Mode

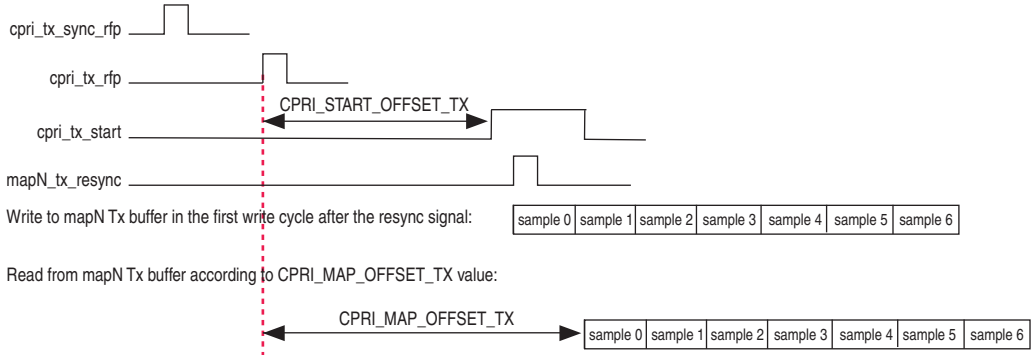


To ensure IP core control over the resynchronization signal timing, Altera recommends that your application trigger the mapN_tx_resync signal with the CPRI IP core output signal cpri_tx_start. The CPRI AUX interface asserts the cpri_tx_start signal according to the offset value specified in the user-programmable CPRI_START_OFFSET_TX register.

Asserting the resynchronization signal ensures correct alignment between the RF implementation and the CPRI basic frame at the appropriate offset from the start of the 10 ms radio frame. In addition to ensuring that application-specific constraints are accommodated, the system can set the CPRI_START_OFFSET_TX register to an offset that precedes the desired frame position in the CPRI transmission, in anticipation of the delays through the antenna-carrier interface Tx buffer and out to the CPRI Tx frame buffer. For information about these delays, refer to "Tx Path Delay" on page D–9.

Figure 4–14 shows the roles of the CPRI_START_OFFSET_TX and CPRI_MAP_OFFSET_TX registers in ensuring correct alignment.

Figure 4–14. User-Controlled Delays in Accepting Data From the AxC Data Channels in Synchronous Buffer Mode



The values programmed in the CPRI_START_OFFSET_TX register control the assertion of the cpri_tx_start signal by the CPRI transmitter. The values in the start_tx_offset_z, start_tx_offset_x, and start_tx_offset_seq fields specify a hyperframe number, basic frame number, and word (sequence) number in the basic frame, respectively, within the 10 ms frame.

The system source of the AxC payload transmits the AxC container block on the data channel to target a specific location in the 10 ms frame; the system programs the information for this location in the CPRI_START_OFFSET_TX and CPRI_MAP_OFFSET_TX registers. The CPRI transmitter learns the location of the AxC container block on the AxC interface from the CPRI_START_OFFSET_TX register. For example, if the CPRI_START_OFFSET_TX register is programmed with the value 0x000595FE, the CPRI transmitter must assert the cpri_tx_start signal at word index 5 of basic frame 254 of hyperframe 149 in the 10ms frame. Altera recommends that the data channel application sample the cpri_tx_start signal, and when it detects the cpri_tx_start signal is asserted, assert the mapN_tx_resync signal to indicate that the samples on mapN_tx_data can begin to fill the data words at the specified position in the CPRI frame. Assertion of the mapN_tx_resync signal resets the write pointer of the current antenna-carrier interface (mapN) Tx buffer to zero, so that the entire buffer is available to receive the data from the data channel. The data on mapN_tx_data[31:0] can safely be loaded in the mapN Tx buffer in the same cycle that the mapN_tx_resync signal is asserted.

On the CPRI side of the mapN Tx buffer, the MAP transmitter interface reads data from the mapN Tx buffer and sends it to the CPRI transmitter interface. The offset programmed in the CPRI_MAP_OFFSET_TX register tells the MAP transmitter interface when to reset the read pointer of the mapN Tx buffer and start transferring data from the buffer to the CPRI transmitter interface. The K counter is reset to zero at the same time, so that it advances from zero with the transfer of the data to the CPRI transmitter interface, tracking the packing of the AxC container block contents into the CPRI frame.

Because the mapN Tx buffer should not be read before it is written, the offset specified in the CPRI_START_OFFSET_TX register must precede the offset specified in the CPRI_MAP_OFFSET_TX register. The CPRI IP core informs you of buffer overflow and underflow (in the CPRI_IQ_TX_BUF_STATUS register described in Table 7–49 on page 7–21 and as reported in the mapN_tx_status_data output vector described in Table 6–2 on page 6–4), but it does not prevent them from occurring. Altera recommends that you implement a separate tracking protocol to ensure you do not overflow or underflow the mapN Tx buffer.

In synchronous buffer mode, because programmed offsets control the mapN Tx buffer pointers, the delay through each mapN Tx buffer can be quantified.

MAP Transmitter in the Internally-clocked Mode

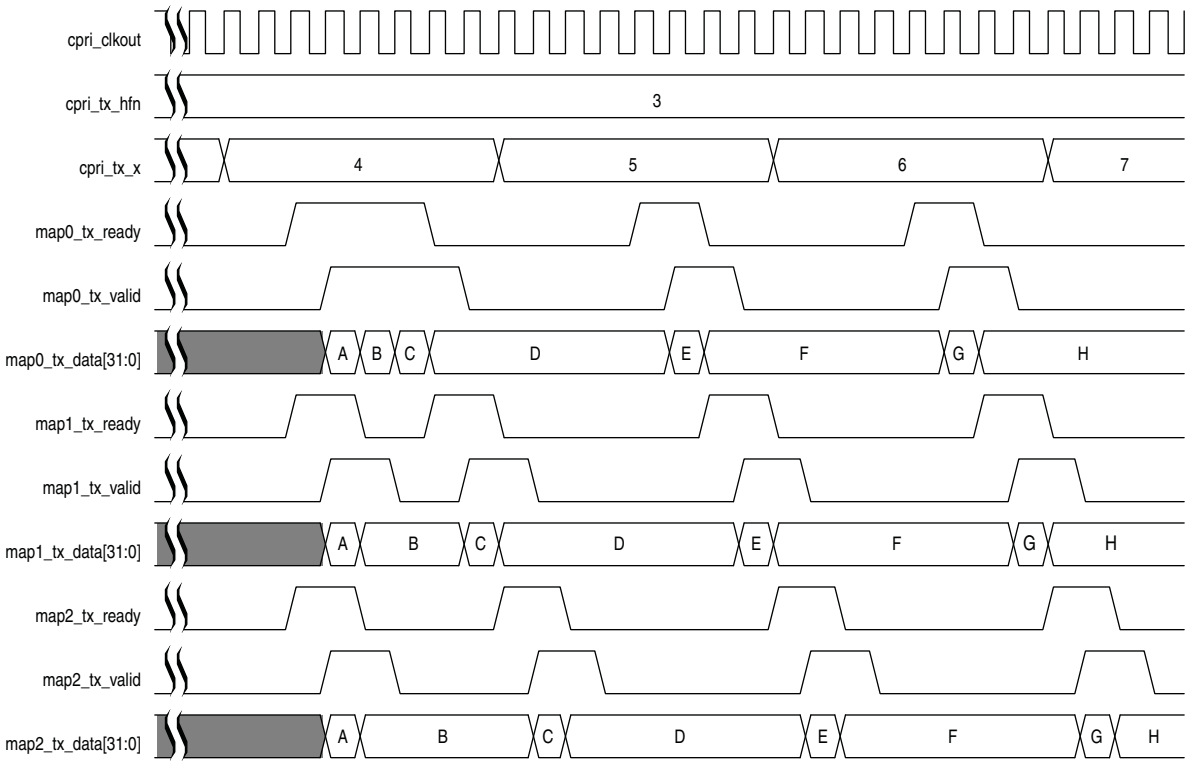
In the internally-clocked mode, each data channel, or AxC interface, has an output ready signal, mapN_tx_ready. Each AxC interface asserts its ready signal when it is ready to receive data on this data channel for transmission to the CPRI protocol interface—when the buffer level is at or below the threshold indicated in the CPRI_MAP_TX_READY_THR register.

After the CPRI IP core asserts the mapN_tx_ready signal, the application is expected to respond by asserting the mapN_tx_valid signal and presenting data on mapN_tx_data. In every cpri_clkout cycle in which mapN_tx_ready is asserted, the application can present valid data on mapN_tx_data, as prescribed by the Avalon-ST specification with READY_LATENCY value 1.

For details about the behavior of the individual signals in the internally-clocked mode, refer to "MAP Transmitter Signals" on page 6–3.

Figure 4–15 shows an example of the behavior of the MAP Tx signals in this synchronization mode in the basic mapping mode (`map_mode` = 2'b00).

Figure 4–15. MAP Transmitter Interface in the Internally-Clocked Mode



In the internally-clocked mode the delay in the AxC interface block from each data channel can be quantified, because this delay is determined solely by the value in the `CPRI_MAP_OFFSET_TX` register.

Auxiliary Interface

The CPRI auxiliary interface enables multi-hop routing applications and provides timing reference information for transmitted and received frames.

The auxiliary (AUX) interface allows you to connect CPRI IP core instances and other system components together by supporting a direct connection to a user-defined routing layer or custom mapping block. You implement this routing layer, which is not defined in the CPRI V4.2 Specification, outside the CPRI IP core. The AUX interface supports the transmission and reception of IQ data and timing information between an RE slave and an RE master, allowing you to define a custom routing layer that enables daisy-chain configurations of RE master and slave ports. Your custom routing layer determines the IQ sample data to pass to other REs to support multi-hop network configurations or to bypass the CPRI IP core MAP interface to implement custom mapping algorithms outside the IP core.

The CPRI IP core implements the AUX receiver and AUX transmitter interfaces as separate Avalon-ST interfaces. The AUX transmitter receives data to be transmitted on the outgoing CPRI link, and the AUX receiver transmits data received from the incoming CPRI link.

For information about the Avalon-ST interface, refer to *Avalon Interface Specifications*.

AUX Receiver Module

The AUX receiver module transmits data that the CPRI IP core received on the CPRI link to the outgoing AUX Avalon-ST interface. In addition, it provides detailed information about the current state in the Rx CPRI frame synchronization state machine. This information is useful for custom user logic, including frame synchronization across hops in multihop configurations.

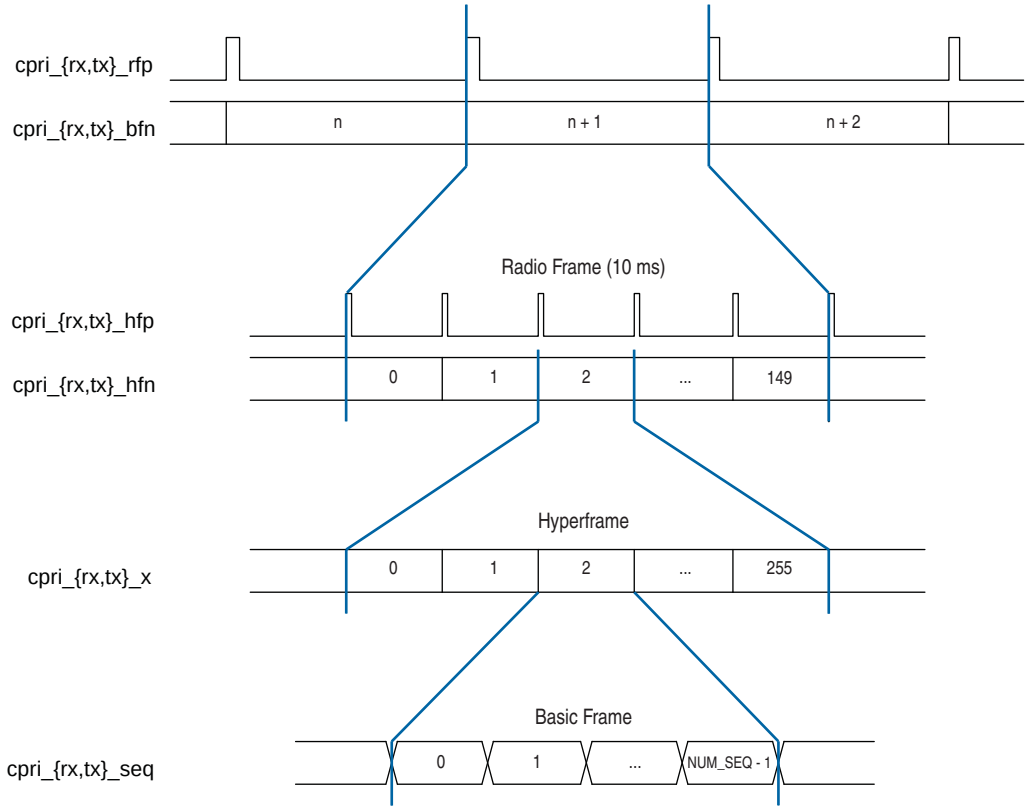
The AUX interface receiver module provides the following data and synchronization lines:

- `cpri_rx_sync_state`—when set, indicates that Rx, HFN, and BFN synchronization have been achieved in CPRI receiver frame synchronization
- `cpri_rx_start`—asserted for the duration of the first basic frame following the offset defined in the `CPRI_START_OFFSET_RX` register
- `cpri_rx_rfp` and `cpri_rx_hfp`—synchronization pulses for start of 10 ms radio frame and start of hyperframe
- `cpri_rx_bfn` and `cpri_rx_hfn`—current radio frame and hyperframe numbers
- `cpri_rx_x`—index number of the current basic frame in the current hyperframe
- `cpri_rx_seq`—index number of the current 32-bit word in the current basic frame
- `cpri_rx_aux_data`—outgoing data port for sending data and control words received on the CPRI link out on the AUX interface

The output synchronization signals are derived from the CPRI frame synchronization state machine. These signals are all fields in the `aux_rx_status_data` bus. For additional information about the AUX receiver signals, refer to Table 6–3 on page 6–6.

Figure 4–16 shows the relationship between the synchronization pulses and numbers.

Figure 4–16. Synchronization Pulses and Numbers on the AUX Interfaces



The AUX receiver presents data on the AUX interface in fixed 32-bit words. The mapping to 32-bit words depends on the CPRI IP core line rate. Figure 4–17 shows how the data received from the CPRI protocol interface module is mapped to the AUX Avalon-ST 32-bit interface.

Figure 4–17. AUX Interface Data at Different CPRI Line Rates (Part 1 of 3)

614.4 Mbps Line Rate:

	Sequence number on AUX interface			
	0	1	2	3
[31:24]	#Z.X.0.0 (1)	#Z.X.4.0	#Z.X.8.0	#Z.X.12.0
[23:16]	#Z.X.1.0	#Z.X.5.0	#Z.X.9.0	#Z.X.13.0
[15:8]	#Z.X.2.0	#Z.X.6.0	#Z.X.10.0	#Z.X.14.0
[7:0]	#Z.X.3.0	#Z.X.7.0	#Z.X.11.0	#Z.X.15.0

Figure 4–17. AUX Interface Data at Different CPRI Line Rates (Part 2 of 3)

1228.8 Mbps Line Rate:

	Sequence number on AUX interface				
	0	1	2	...	7
[31:24]	#Z.X.0.0 (1)	#Z.X.2.0	#Z.X.4.0	...	#Z.X.14.0
[23:16]	#Z.X.0.1 (1)	#Z.X.2.1	#Z.X.4.1	...	#Z.X.14.1
[15:8]	#Z.X.1.0	#Z.X.3.0	#Z.X.5.0	...	#Z.X.15.0
[7:0]	#Z.X.1.1	#Z.X.3.1	#Z.X.5.1	...	#Z.X.15.1

2457.6 Mbps Line Rate:

	Sequence number on AUX interface				
	0	1	2	...	15
[31:24]	#Z.X.0.0 (1)	#Z.X.1.0	#Z.X.2.0	...	#Z.X.15.0
[23:16]	#Z.X.0.1 (1)	#Z.X.1.1	#Z.X.2.1	...	#Z.X.15.1
[15:8]	#Z.X.0.2 (1)	#Z.X.1.2	#Z.X.2.2	...	#Z.X.15.2
[7:0]	#Z.X.0.3 (1)	#Z.X.1.3	#Z.X.2.3	...	#Z.X.15.3

3072.0 Mbps Line Rate:

	Sequence number on AUX interface					
	0	1	2	...	18	19
[31:24]	#Z.X.0.0 (1)	#Z.X.0.4 (1)	#Z.X.1.3	...	#Z.X.14.2	#Z.X.15.1
[23:16]	#Z.X.0.1 (1)	#Z.X.1.0	#Z.X.1.4	...	#Z.X.14.3	#Z.X.15.2
[15:8]	#Z.X.0.2 (1)	#Z.X.1.1	#Z.X.2.0	...	#Z.X.14.4	#Z.X.15.3
[7:0]	#Z.X.0.3 (1)	#Z.X.1.2	#Z.X.2.1	...	#Z.X.15.0	#Z.X.15.4

4915.0 Mbps Line Rate:

	Sequence number on AUX interface					
	0	1	2	...	30	31
[31:24]	#Z.X.0.0 (1)	#Z.X.0.4 (1)	#Z.X.1.0	...	#Z.X.14.0	#Z.X.15.4
[23:16]	#Z.X.0.1 (1)	#Z.X.0.5 (1)	#Z.X.1.1	...	#Z.X.14.1	#Z.X.15.5
[15:8]	#Z.X.0.2 (1)	#Z.X.0.6 (1)	#Z.X.2.2	...	#Z.X.14.2	#Z.X.15.6
[7:0]	#Z.X.0.3 (1)	#Z.X.0.7 (1)	#Z.X.2.3	...	#Z.X.15.3	#Z.X.15.7

Figure 4–17. AUX Interface Data at Different CPRI Line Rates (Part 3 of 3)

6144.0 Mbps Line Rate:

	Sequence number on AUX interface					
	0	1	2	...	38	39
[31:24]	#Z.X.0.0 (1)	#Z.X.0.4 (1)	#Z.X.0.8 (1)	...	#Z.X.15.2	#Z.X.15.6
[23:16]	#Z.X.0.1 (1)	#Z.X.0.5 (1)	#Z.X.0.9 (1)	...	#Z.X.15.3	#Z.X.15.7
[15:8]	#Z.X.0.2 (1)	#Z.X.0.6 (1)	#Z.X.1.0	...	#Z.X.15.4	#Z.X.15.8
[7:0]	#Z.X.0.3 (1)	#Z.X.0.7 (1)	#Z.X.1.1	...	#Z.X.15.5	#Z.X.15.9

9830.4 Mbps Line Rate:

	Sequence number on AUX interface						
	0	1	2	3	...	62	63
[31:24]	#Z.X.0.0 (1)	#Z.X.0.4 (1)	#Z.X.0.8 (1)	#Z.X.0.12 (1)	...	#Z.X.15.8	#Z.X.15.12
[23:16]	#Z.X.0.1 (1)	#Z.X.0.5 (1)	#Z.X.0.9 (1)	#Z.X.0.13 (1)	...	#Z.X.15.9	#Z.X.15.13
[15:8]	#Z.X.0.2 (1)	#Z.X.0.6 (1)	#Z.X.0.10 (1)	#Z.X.0.14 (1)	...	#Z.X.15.10	#Z.X.15.14
[7:0]	#Z.X.0.3 (1)	#Z.X.0.7 (1)	#Z.X.0.11 (1)	#Z.X.0.15 (1)	...	#Z.X.15.11	#Z.X.15.15

Note to Figure 4–17:

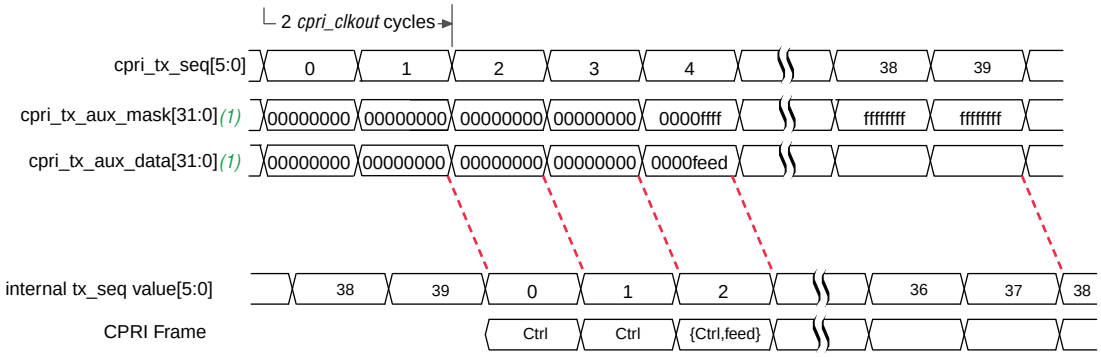
(1) Light blue table cells indicate control word bytes. White table cells indicate data word bytes.

AUX Transmitter Module

The AUX transmitter module receives data on the incoming AUX Avalon-ST interface and sends it to the CPRI IP core physical layer to transmit on the CPRI link. In addition, it outputs CPRI link frame synchronization information, to enable synchronization of the AUX data.

The incoming data on the AUX interface must match the CPRI frame with a delay of exactly two cpri_clkout clock cycles. The cpri_tx_seq[5:0] value that you read at the AUX Tx interface is two cpri_clkout cycles ahead of the internal sequence number that tracks the CPRI frame. If you want your IQ sample to land at sequence number N of the CPRI frame, then you must present your sample at the AUX Tx interface when cpri_tx_seq[5:0] has the value of N+2. Figure 4–18 shows the expected timing on the incoming AUX connection in a variation with a CPRI line rate of 6144.4 Mbps.

Figure 4–18. Incoming AUX Link Synchronization



Note to Figure 4–18:

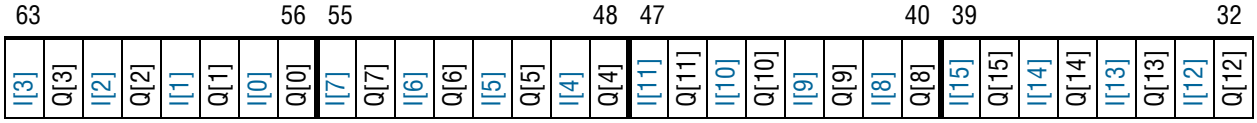
(1) The cpri_tx_aux_data and cpri_tx_aux_mask signals are fields in the aux_tx_mask_data input bus. Refer to Table 6–4 on page 6–7.

In Figure 4–18, the application presents data when cpri_tx_seq[5:0] has the value of 4, and sets the value of cpri_tx_aux_mask, to ensure the data is loaded in the CPRI frame immediately following the control word. Because the CPRI line rate in this example is 6144.4 Mbps, the length of the control word is ten bytes. Therefore, the application presents the data when cpri_tx_seq[5:0] has the value of 4 to ensure the data is loaded in the CPRI frame at position 2.

In addition, to ensure the CPRI IP core transmits the incoming AUX data correctly on the CPRI link, you must format the incoming AUX data in the correct order to match the CPRI IP core internal data representation. If you connect two Altera CPRI IP cores through a routing layer, and your routing layer does not modify the data transmission order, then the correct order is guaranteed. However, if a different application transmits data to the CPRI IP core AUX interface, it must enforce the data order that the CPRI IP core expects.

Incoming AUX data to the CPRI IP core appears on cpri_tx_aux_data[31:0], also called aux_tx_mask_data[64:32]. Byte [31:24] (64:56]) is transmitted first, and byte [7:0] (39:32]) is transmitted last: cpri_tx_aux_data[31:24] is byte 0 in the transmission order, and contains the least significant I- and Q-nibbles of the data sample. Figure 4–19 illustrates the required data order on this data bus.

Figure 4–19. Required Data Sample Order in aux_tx_mask_data[63:32] (cpri_tx_aux_data[31:0])



The CPRI IP core passes the incoming AUX data through to the CPRI link unmodified. You must ensure that the incoming AUX data bits already include any CRC values expected by the application at the other end of the CPRI link.

The CPRI transmitter frame synchronization state machine provides the following data and synchronization signals on the AUX interface to enable the required precise frame timing:

- `cpri_tx_start`—asserted for the duration of the first basic frame following the offset defined in the `CPRI_START_OFFSET_TX` register
- `cpri_tx_rfp` and `cpri_tx_hfp`—synchronization pulses for start of 10 ms radio frame and start of hyperframe
- `cpri_tx_bfn` and `cpri_tx_hfn`—current radio frame and hyperframe numbers
- `cpri_tx_x`—index number of the current basic frame in the current hyperframe
- `cpri_tx_seq`—index number of the current 32-bit word in the current basic frame
- `cpri_tx_aux_data`—incoming data port for data on the AUX link
- `cpri_tx_aux_mask`—incoming bit mask for AUX link data that indicates bits that must be transmitted without changes to the CPRI link

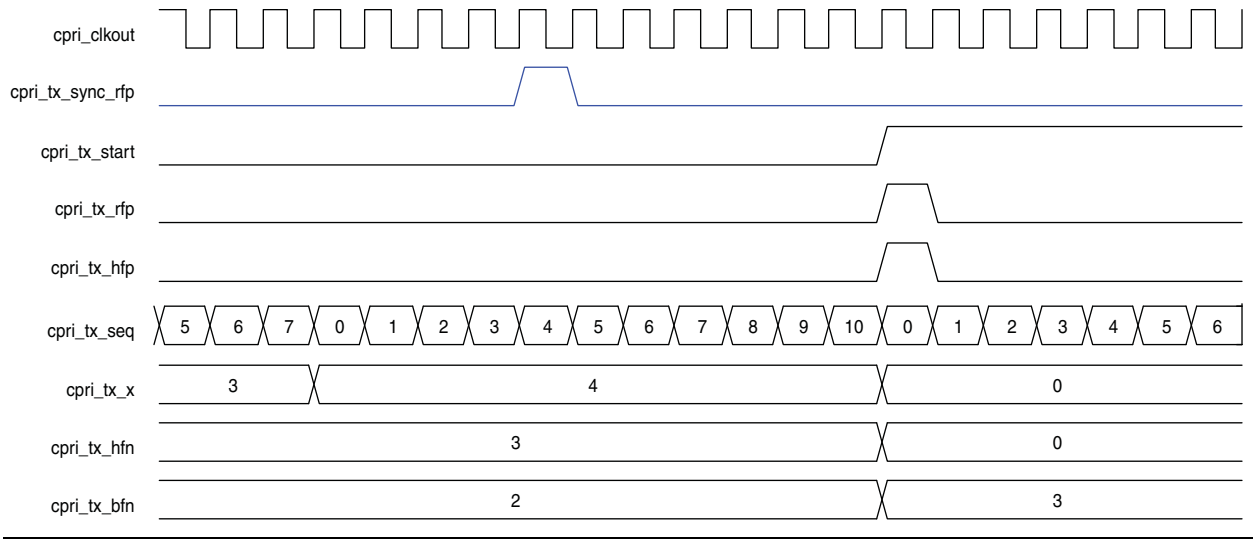
  The CPRI IP core layer 1 uses the `cpri_tx_aux_mask` to select the enabled bit values in the control transmit table. When mask bits are set, the corresponding data bits from the AUX interface fill the CPRI frame, overriding any internally-generated information. You must deassert all the mask bits during K28.5 character insertion in the outgoing CPRI frame (which occurs when Z=X=0). Otherwise, the CPRI IP core asserts an error signal `cpri_tx_error` on the following `cpri_clkout` clock cycle to indicate that the K28.5 character expected by the CPRI link protocol has been overwritten. You must also ensure you do not override synchronization counter values in the control word.

The AUX transmitter module also receives a synchronization pulse in an REC master. Application software can pulse the `cpri_tx_sync_rfp` input signal to resynchronize the 10 ms radio frame. Asserting this signal resets the frame synchronization machine in an REC master.

In response to the rising edge of its `cpri_tx_sync_rfp` input signal (`aux_tx_mask_data[64]`), a CPRI REC master IP core restarts the 10 ms radio frame. The rising edge of the `cpri_tx_sync_rfp` signal must be synchronous with the `cpri_clkout` clock. On the seventh `cpri_clkout` cycle following a `cpri_tx_sync_rfp` pulse, the `cpri_tx_hfp` and `cpri_tx_rfp` signals pulse, the `cpri_tx_x` and `cpri_tx_hfn` signals have the value 0, and the `cpri_tx_bfn` signal increments from its

previous value. Figure 4–20 illustrates the behavior of the CPRI IP core signals in response to the cpri_tx_sync_rfp pulse.

Figure 4–20. CPRI REC Master Response to cpri_tx_sync_rfp Resynchronization Pulse



For more information about the relationships between the synchronization pulses and numbers, refer to Figure 4–16 on page 4–29. For the mapping of data between the AUX interface and the CPRI link, refer to Figure 4–17 on page 4–29.

The cpri_tx_aux_data and cpri_tx_aux_mask signals are fields of the aux_tx_mask_data bus. The other signals described in the preceding list are fields of the aux_tx_status_data bus. For additional information about the AUX transmitter signals, refer to Table 6–4 on page 6–7.

Media Independent Interface to an External Ethernet Block

The media independent (MI) interface, or MII, allows the CPRI IP core to communicate directly with an external Ethernet MAC block, replacing the internal Ethernet MAC. You specify in the CPRI parameter editor whether to implement this interface or to use the Ethernet MAC block available with the CPRI IP core. The two options are mutually exclusive.

If you configure the CPRI IP core with the MII, you must implement the Ethernet MAC block outside the CPRI IP core.

The MI interface is not a true media-independent interface, because it is clocked by the cpri_clkout clock (which drives the cpri_mii_txclk and cpri_mii_rxclk clock signals directly), whose frequencies do not match the usual 2.5 MHz and 25 MHz frequencies of the media-independent protocol specification. If you use this interface, your external Ethernet block must communicate with the CPRI IP core synchronously with the cpri_mii_txclk and cpri_mii_rxclk clocks.

The MII supports the bandwidth described in the CPRI V4.2 Specification in Table 12, Achievable Ethernet bit rates.

MII Transmitter

The MII transmitter module receives data from the external Ethernet MAC block and writes it to the CPRI transmitter module, which transmits it on the CPRI link. It performs 4B/5B encoding on the incoming data nibbles before sending them to the CPRI transmitter module.

After the CPRI IP core achieves frame synchronization, the MII transmitter module can accept incoming data on the MII. The MII transmitter module asserts the `cpri_mii_txrd` signal to indicate it is ready to accept data from the external Ethernet MAC block. After the `cpri_mii_txrd` signal is asserted, the external Ethernet block asserts the `cpri_mii_txen` signal to indicate it is ready to provide data. The MII transmitter module deasserts the `cpri_mii_txrd` signal in the cycle following each cycle in which it receives data. It may remain deasserted for multiple cycles, to prevent buffer overflow. While the `cpri_mii_txrd` signal remains low, the external Ethernet block must maintain the data value on `cpri_mii_txd`.

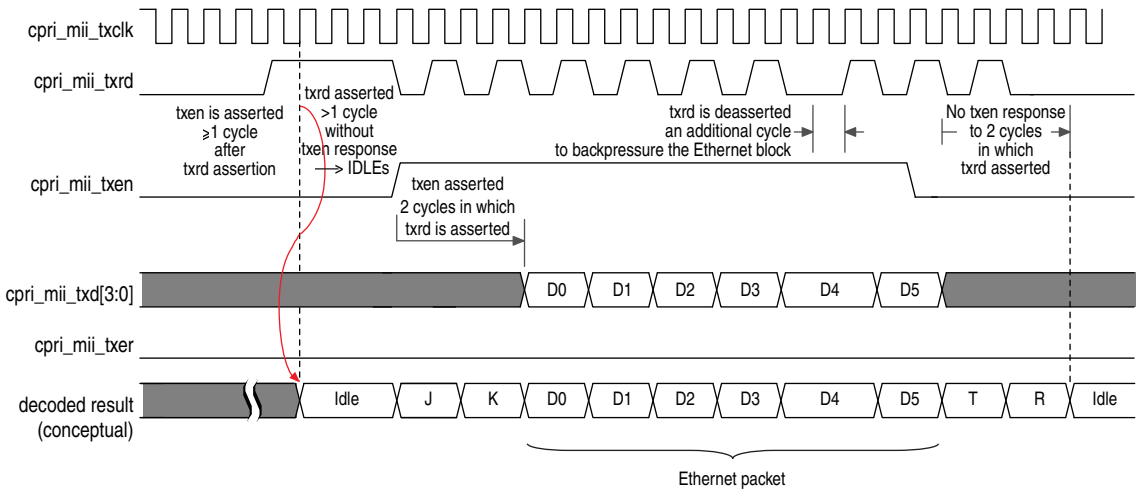
During the first `cpri_mii_txclk` cycle in which `cpri_mii_txen` is asserted, the MII module inserts an Ethernet J symbol (5'b11000) in the buffer of data to be transmitted to the CPRI link; during the second cycle in which `cpri_mii_txen` is asserted, the MII module inserts an Ethernet K symbol (5'b10001) in this buffer. These two symbols indicate Ethernet start-of-packet. While the CPRI MII transmitter is inserting the J and K symbols, it ignores incoming data on `cpri_mii_txd`. Refer to Figure 4–21.

Typically, the external Ethernet block asserts `cpri_mii_txen` one clock cycle after `cpri_mii_txrd` is asserted. While the `cpri_mii_txen` signal remains asserted, the MII transmitter module reads data on the `cpri_mii_txd` input data bus. Following this data sequence, in the first two `cpri_mii_txclk` cycles in which the `cpri_mii_txen` signal is not asserted, the MII module inserts an Ethernet end-of-packet symbol—T followed by R. While the CPRI MII transmitter is inserting the T and R symbols, it ignores incoming data on `cpri_mii_txd`. Refer to Figure 4–21.

While `cpri_mii_txen` is asserted, the `cpri_mii_txer` input signal indicates that the current nibble on `cpri_mii_txd` is suspect. Therefore, if the MII transmitter module observes that both `cpri_mii_txen` and `cpri_mii_txer` are asserted in the same `cpri_mii_txclk` cycle, the MII module inserts an Ethernet HALT symbol (5'b00100). Figure 4–23 on page 4–38 provides an example in which the `cpri_mii_txer` signal is asserted, and shows how the error indication propagates to the MII receiver module on the CPRI link slave.

Figure 4–21 illustrates the MII transmitter protocol with no input errors. The cpri_mii_txen signal remains asserted for the duration of the packet transfer. Although cpri_mii_txrd can be reasserted every other cycle during transmission of an Ethernet packet on cpri_mii_txd, this need not always occur. The CPRI MII transmitter can deassert cpri_mii_txrd for more than one cycle to backpressure the external Ethernet block. In that case, the external Ethernet block must maintain the data value on cpri_mii_txd until the cycle following reassertion of cpri_mii_txrd.

Figure 4–21. CPRI MII Transmitter Example



If cpri_mii_txen is deasserted while cpri_mii_txrd is deasserted, and is not reasserted in the cycle following the reassertion of cpri_mii_txrd, then the CPRI MII transmitter inserts a T symbol in the packet; therefore, the external Ethernet block must reassert cpri_mii_txen in the cycle following reassertion of cpri_mii_txrd, during transmission of an Ethernet packet on cpri_mii_txd.

For more information about the MII transmitter module, refer to "CPRI MII Transmitter Signals" on page 6–10.

MII Receiver

The MII receiver module receives data from the CPRI link by reading it from the CPRI receiver module. It performs 4B/5B decoding on the 5-bit data values before transmitting them as 4-bit data values on the MII.

After the CPRI IP core achieves frame synchronization, the MII receiver module can send data to the external Ethernet block. The MII receiver module transmits the K nibble to indicate start-of-frame on the MII. The J nibble of the start-of-frame is consumed by the CPRI IP core, and is not transmitted on the MII.

The MII receiver module transmits the K nibble and then the data to the cpri_mii_rxd output data bus and asserts the cpri_mii_rxdv signal to indicate that the data currently on cpri_mii_rxd is valid. It sends the K nibble and the data to the cpri_mii_rxd output data bus on the rising edge of the cpri_mii_rxclk clock. During the first cpri_mii_rxclk cycle of every new data value on cpri_mii_rxd, the MII receiver module asserts the cpri_mii_rxwr signal. After the MII receiver module completes sending data to the external Ethernet block, it deasserts the cpri_mii_rxdv signal.

While frame synchronization is not achieved, the cpri_mii_rxer signal remains asserted and cpri_mii_rxdv remains deasserted.

Figure 4–22 illustrates the MII receiver protocol.

Figure 4–22. CPRI MII Receiver Example

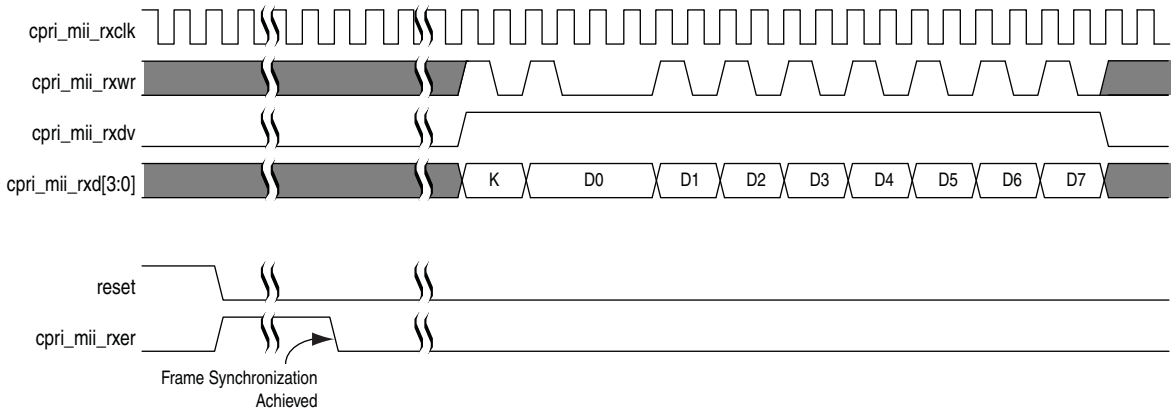
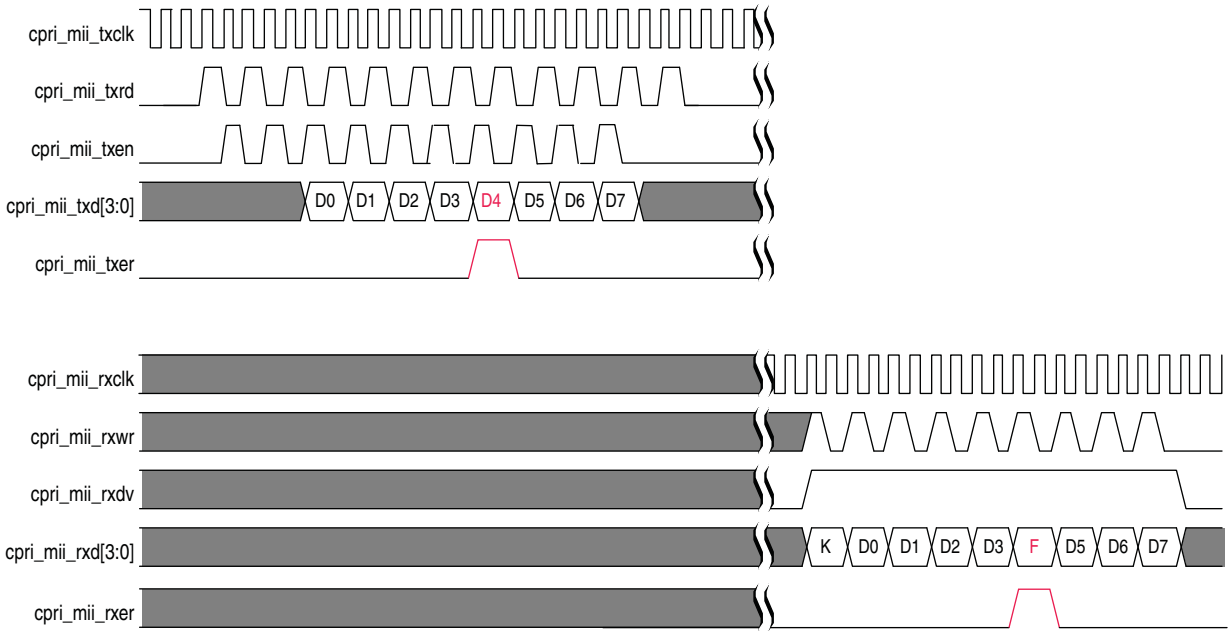


Figure 4–23 shows an example timing diagram in which an input error is noted on the MII of a transmitting RE or REC master, and the data from the MII is transmitted on the CPRI link to a receiving RE slave. The timing diagram shows the MII signals on the transmitting master and the receiving slave. The data value captured on the MII transmitter module of the RE or REC master when `cpri_mii_txer` is asserted, is passed to the CPRI link as a 5-bit Ethernet HALT symbol (5'b00100). The RE slave MII receiver module decodes this symbol as an F (4'b1111) while the `cpri_mii_rxer` signal is asserted.

Figure 4–23. CPRI MII Signals on Transmitting RE or REC Master and on Receiving RE Slave



For more information about the MII receiver module, refer to "CPRI MII Receiver Signals" on page 6–10.

CPU Interface

Use the CPU interface to communicate the contents of the control word of a CPRI hyperframe — VSS, Ethernet, High-Level Data Link Controller (HDLC), and synchronization and timing information — and to access status and configuration information in the CPRI IP core registers. An on-chip processor such as the Nios II processor, or an external processor, can access the CPRI configuration address space using this interface.

The CPU interface provides an Avalon-MM slave interface that accesses all registers in the CPRI IP core. The Avalon-MM slave executes transfers between the CPRI IP core and the user-defined logic in your design.

For information about the Avalon-MM interface, refer to *Avalon Interface Specifications*.

Each of the three sources of input to the CPU interface communicates with the CPRI IP core by reading and writing registers through a single Avalon-MM port on the CPU interface. Arbitration among the different sources must occur outside the CPRI IP core.

If the CPRI IP core is configured with an MII, the application cannot access the IP core's Ethernet registers through the CPU interface. However, if the HDLC block is configured, you can access the IP core's HDLC registers whether or not the MII is configured.

For more information about the CPRI IP core registers, refer to Chapter 7, Software Interface.

Accessing the Hyperframe Control Words

You can access the 256 control words in a hyperframe through the CPRI IP core CPU interface. The CPRI_CTRL_INDEX register (Table 7–7 on page 7–4) and the CPRI_RX_CTRL register (Table 7–8 on page 7–4) support your application in reading the incoming control words, and the CPRI_CONFIG register (Table 7–6 on page 7–3), CPRI_CTRL_INDEX register, and CPRI_TX_CTRL register (Table 7–9 on page 7–5) support the application in writing to outgoing control words.

Register support only provides you access to the initial byte of each control word. You can access the full control words through the CPRI IP core AUX interface.

Table 4–12 summarizes the relevant register fields. For complete information, refer to the register tables in Chapter 7, Software Interface.

Table 4–12. Register Support for Control Word Access

Register	Register Bits	Field Name	Description
CPRI_CTRL_INDEX (Table 7–7)	[7:0]	cpri_ctrl_index	Index for CPRI control byte monitoring and insertion. The value in this field determines the control receive and control transmit table entries that appear in the CPRI_RX_CTRL and CPRI_TX_CTRL registers.
CPRI_RX_CTRL (Table 7–8)	[7:0]	rx_control_data	Most recent received CPRI control word from CPRI hyperframe position Z.x.0, where x is the index in the cpri_ctrl_index field of the CPRI_CTRL_INDEX register.
CPRI_TX_CTRL (Table 7–9)	[8]	tx_control_insert	Control byte transmit enable.
	[7:0]	tx_control_data	CPRI control byte to be transmitted in CPRI hyperframe position Z.x.0, where x is the index in the cpri_ctrl_index field of the CPRI_CTRL_INDEX register.
CPRI_CONFIG (Table 7–6)	[0]	tx_ctrl_insert_en	Master enable for insertion of tx_control_data contents in CPRI control word. This signal enables control bytes for which the tx_control_insert bit is high to be written to the CPRI frame.

Recording and Retrieving the Incoming Control Bytes

A control receive table contains a 1-byte entry for each of the 256 control words in the current hyperframe. To read a control byte, your application must write the control word number X to the CPRI_CTRL_INDEX register and then read the last received #Z.X.0 control byte in the CPRI_RX_CTRL register. Because each table entry is a single byte, you can use this access method only to retrieve the first byte of a control word.

Writing the Outgoing Control Bytes

A control transmit table contains an entry for each of the 256 control words in the current hyperframe. Each control transmit table entry contains a control byte field and an enable bit. As the frame is created, if a control word entry is enabled, and the global tx_ctrl_insert_en bit in the CPRI_CONTROL register is set, the low-level transmitter writes the control byte to the first byte of the CPRI frame's control word.

To write a control byte in the control transmit table, write the control word number X to the CPRI_CTRL_INDEX register and then write the next intended #Z.X.0 control byte and set the tx_control_insert bit in the CPRI_TX_CTRL register. After you update the control transmit table, set the tx_ctrl_insert_en bit of the CPRI_CONFIG register to enable the CPRI IP core to write the values from the control transmit table to the control words in the outgoing CPRI frame.

The tx_control_insert bit of the CPRI_TX_CTRL register enables or disables the transmission of the corresponding byte in the control transmit table in the CPRI frame. The tx_ctrl_insert_en bit of the CPRI_CONFIG register is the master enable: when it is set, the CPRI IP core writes all table entries with the tx_control_insert bit set into the CPRI frame.

Control Word Order

The entries in the control receive and control transmit tables match the organization of control words in subchannels from the CPRI specification. Figure 4–24 shows this word order. The figure is Figure 15 of the CPRI V4.2 Specification.

Figure 4–24. Illustration of Subchannels in a Hyperframe (Part 1 of 2)

	Xs == 0	1	2	3
Ns == 0	0: K28.5	Synchronization and Timing		
1	1: HDLC link	65: HDLC	129: HDLC	193: HDLC
2	2: L1 In-band	66: L1 in-band	130: L1 in-band	194: P (20 = 0x14)
3	3: Reserved	67: Reserved		
4	4: Reserved	...		
5	...			
	...			
14	14: Reserved			
15	15: Reserved	79: Reserved	143: Reserved	207: Reserved
16	Vendor-specific			
	...			

Figure 4–24. Illustration of Subchannels in a Hyperframe (Part 2 of 2)

19				
20 Pointer P --->	20: Ethernet			
	...			
62	62	126	190	254
63	63	127	191	255

Figure 4–24 illustrates how the 256 control words in the hyperframe are organized as 64 subchannels of four control words each. The figure illustrates why the index X of a control word is Ns + 64 × Xs, where Ns is the subchannel index and Xs is the index of the control word within the subchannel.

Control Word Transmission Example

To write to the vendor-specific portion of the control word in a transmitted hyperframe, perform the following steps:

1. Identify the indices for the vendor-specific portion of the transmit control table, using the formula X = Ns + 64 × Xs.

   In the example, Ns = 16 and Xs = 0,1,2, and 3. Therefore, the indices to be written are 16, 80, 144, and 208.

2. For each value X in 16, 80, 144, and 208, perform the following steps:

   a. Write the value X to the `cpri_ctrl_index` field of the `CPRI_CTRL_INDEX` register.

   b. Write the control byte to the `tx_control_data` field of the `CPRI_TX_CTRL` register and set the `tx_control_insert` field of the `CPRI_TX_CTRL` register to the value of 1.

3. After you update the control transmit table with the control bytes, to insert the data in the next outgoing CPRI frame, set the `tx_ctrl_insert_en` field of the `CPRI_CONFIG` register to the value of 1.

Control Word Retrieval Example

To retrieve the first byte of the vendor-specific portion of a control word in the most recent received hyperframe, perform the following steps:

1. Identify the indices for the vendor-specific portion of the transmit control table, using the formula X = Ns + 64 × Xs.

   In the example, Ns = 16 and Xs = 0,1,2, and 3. Therefore, the indices to be read are 16, 80, 144, and 208.

2. For each value X in 16, 80, 144, and 208, perform the following steps:

   a. Write the value X to the `cpri_ctrl_index` field of the `CPRI_CTRL_INDEX` register.

   b. In the following `cpu_clk` cycle, read the control byte in the `rx_control_data` field of the `CPRI_RX_CTRL` register.

Accessing the Ethernet Channel

If you turn on the **Include MAC block** parameter, your CPRI IP core includes an internal Ethernet Media Access Controller (MAC). If you turn off this parameter, an MII is available for you to connect to your own external Ethernet MAC. In that case, the internal Ethernet MAC is not available and your application cannot access the Ethernet registers. If the internal Ethernet MAC is turned off, attempts to access these registers read zeroes and do not write successfully, as for a reserved register address.

The Ethernet MAC is responsible for processing the Ethernet frame. The Ethernet MAC unloads the Ethernet frame from the CPRI frame and stages it in the Ethernet registers, where it is accessible through the CPU interface. The Ethernet MAC also handles the flow of Ethernet data to the CPRI frame, by loading it from the Ethernet registers into the Ethernet space in the CPRI hyperframe.

The CPRI specification dictates that a CPRI hyperframe that contains Ethernet data also contain a pointer to the start of that data in control byte Z.194.0. The pointer value 0x0 indicates that no Ethernet channel is supported in the current hyperframe. A valid pointer holds a subchannel index value between 0x14 and 0x3F, inclusive. The length of the Ethernet data can extend beyond the end of the hyperframe; if a received Ethernet frame exceeds 1536 bytes, the Ethernet module resets, unless the `rx_long_frame_en` bit of the `ETH_CONFIG_1` register is set.

The CPRI transmitter reads the pointer value from the `tx_fast_cm_ptr` field of the `CPRI_CM_CONFIG` register and writes it in CPRI control byte Z.194.0 in the outgoing CPRI hyperframe. The `rx_fast_cm_ptr` field of the `CPRI_CM_STATUS` register holds the current pointer value, determined during the software set-up sequence or by dynamic modification, in which the same new pointer value is received in CPRI control byte Z.194.0 four hyperframes in a row.

Software can configure the Ethernet channel by writing to the `ETH_CONFIG_1` register through the CPRI IP core Avalon-MM CPU interface. For additional information about this register, refer to Chapter 7, Software Interface.

Transmitting Ethernet Traffic

To transmit an Ethernet frame, the CPRI IP core must load the frame in a Tx Ethernet buffer. Application software can direct the CPRI IP core to load the Ethernet frame in the Tx Ethernet buffer by reading and writing the following registers:

- `ETH_CONFIG_2` register at offset 0x20C (Table 7–54 on page 7–23)—Configure the CPRI IP core to automatically calculate the Frame check sequence and insert it at the end of the frame data, by setting the `crc_enable` field in bit 0 of this register.
- `ETH_TX_STATUS` register at offset 0x204 (Table 7–52 on page 7–22)—Poll the `tx_ready_block` and `tx_ready` fields of this register. If the `tx_ready` field has a value of 1, you can load a 4-byte word to the Tx Ethernet buffer. If the `tx_ready_block` field has a value of 1, you can load a block of eight 4-byte entries to the Tx Ethernet buffer without polling the `tx_block_ready` or `tx_ready` bits between CPU write operations.
- `ETH_TX_DATA` register at offset 0x220 (Table 7–59 on page 7–24)—Load data in this register. To load a block of eight 4-byte entries to the Tx Ethernet buffer, you must execute eight CPU write operations to this register.

- ETH_TX_CONTROL register at offset 0x21C (Table 7–58 on page 7–24)—Before you load the final word of an Ethernet frame in the ETH_TX_DATA register (or ETH_TX_DATA_WAIT register (Table 7–60 on page 7–24)), set the tx_eop field and write the tx_length field of this register to indicate how many bytes in the final word are padding.

The Ethernet Tx buffer holds 64 4-byte entries, for a total of 256 bytes. When transmitting Ethernet frames larger than the capacity of the Tx Ethernet buffer, you must ensure you do not overflow or underflow the buffer. If the Ethernet transmitter module writes data to the ETH_TX_DATA register when the Ethernet Tx buffer is not ready, the tx_abort bit is set in the ETH_TX_STATUS register and the current Ethernet packet is aborted. To prevent the Ethernet transmitter module from aborting a frame, you can write the data to the ETH_TX_DATA_WAIT register. The ETH_TX_DATA_WAIT register can accept data when the Ethernet Tx buffer is not ready for new data.

You must write each frame's data to the ETH_TX_DATA register continuously. The Ethernet transmitter module ensures the correct bit order for transmission on the CPRI link. If the crc_enable field of the ETH_CONFIG_2 register has the value of 0, you must insert the CRC in the frame data, because the Ethernet receiver module checks CRC. In this case, you must reverse the bit order of the CRC bytes so that the most significant byte of the CRC is transmitted first.

If you set the crc_enable field of the ETH_CONFIG_2 register to the value of 1, the Tx Ethernet automatically calculates the Frame check sequence and inserts it at the end of the Ethernet frame data in the Tx Ethernet buffer.

Software can set the tx_discard bit in the ETH_TX_CONTROL register, which in turn causes the tx_abort bit in the ETH_TX_STATUS register to be set. The Ethernet transmitter module can also set the tx_abort bit directly.

The Tx Ethernet controller reads the Tx Ethernet buffer after you set the tx_eop bit of the ETH_TX_CONTROL register and write the final word in the ETH_TX_DATA register. If you disable the store-and-forward feature by resetting the tx_st_fwd field of the ETH_FWD_CONFIG register at offset 0x244 (Table 7–64 on page 7–25), the Tx Ethernet controller also reads the Tx Ethernet buffer whenever the number of words in the Tx Ethernet buffer is above a programmable threshold.

Interrupts

Software can enable interrupts by setting bits in the ETH_CONFIG_1 register at offset 0x208 (Table 7–53 on page 7–23). The intr_en bit is the Ethernet global interrupt enable and intr_tx_en is the Ethernet Tx interrupt enable. If both of these two bits are set, software can use the status in the ETH_TX_STATUS register to generate interrupts. For example, using the tx_ready_block bit to generate an interrupt ensures that the CPU is interrupted only when a full 32-bit packet of data is ready to transfer to the Ethernet Tx buffer.

Receiving Ethernet Traffic

The Ethernet receiver module receives Ethernet data from the CPRI link by reading it from the Ethernet Rx buffer through an Ethernet register.

This section describes how the Ethernet receiver module performs MAC address filtering according to the ETH_CONFIG_1, ETH_ADDR_LSB, and ETH_ADDR_MSB registers, provides status information to the CPU interface in the ETH_RX_STATUS register, and allows the CPU interface to insert wait states in the Ethernet channel.

For additional information about the Ethernet receiver registers, refer to Chapter 7, Software Interface.

MAC Address Filtering

To enable MAC address checking, set the mac_check bit of the ETH_CONFIG_1 register. If the mac_check bit is reset to the value of zero, the Ethernet receiver accepts all received packets.

You can enable the following three MAC address filters:

- Unicast filtering: check that the destination MAC address is the address specified in the ETH_ADDR_LSB and ETH_ADDR_MSB registers. If the mac_check bit is not set, this filter is disabled.
- Multicast filtering: if the least significant bit of the first destination MAC address byte, the group address bit, is set to 1, use the ETH_HASH_TABLE register to determine whether to accept this destination MAC address. Because the hash algorithm might not filter the destination address as intended, you must implement full address validation in software if you enable multicast filtering. To enable multicast filtering, set the multicast_flt_en bit of the ETH_CONFIG_1 register.
- Broadcast filtering: accept all packets with destination MAC address 0xFFFFFFFFFFFF, the Ethernet broadcast address. To enable broadcast filtering, set the broadcast_en bit of the ETH_CONFIG_1 register.

Ethernet Rx Buffer Status

The CPRI IP core reports relevant Ethernet Rx buffer status to the CPU interface by updating the following fields of the ETH_RX_STATUS register:

- The ETH_RX_STATUS rx_ready bit indicates that at least one word of data is available in the Ethernet Rx buffer and ready to be read.
- The ETH_RX_STATUS rx_eop bit indicates that the next ready data word contains the end-of-packet byte.
- The ETH_RX_STATUS rx_length field indicates the number of valid bytes in the end-of-packet word.
- The ETH_RX_STATUS rx_abort bit indicates that the current received packet is aborted.
- The ETH_RX_STATUS rx_ready_block bit indicates that the next block of packet data is ready to be read and does not contain the end-of-packet byte.
- The ETH_RX_STATUS rx_ready_end bit indicates that the end-of-packet byte is ready in the Ethernet Rx buffer.

Software can set the ETH_RX_CONTROL rx_discard bit to abort the current received packet. The Ethernet receiver ensures that following read from the Ethernet Rx buffer is a start-of-packet word.

Ethernet Data Transfer

The next ready data word is available in the ETH_RX_DATA and ETH_RX_DATA_WAIT registers. If no Ethernet data word is ready, reading from the ETH_RX_DATA_WAIT register inserts wait states in the Ethernet channel. If no Ethernet data word is ready, reading from the ETH_RX_DATA register causes the rx_abort bit to be set. The CPU interface receiver module reads the Ethernet packet data one word at a time from one of these registers.

Accessing the HDLC Channel

If you turn on the **Include HDLC block** parameter, your CPRI IP core includes an internal High-Level Data Link Controller (HDLC) block. If you turn off this parameter, the internal HDLC block is not available and your application cannot access the HDLC registers. If the internal HDLC block is turned off, attempts to access these registers read zeroes and do not write successfully, as for a reserved register address.

In the CPRI IP core, the HDLC block, or slow data link layer, passes HDLC data between the CPU interface and the CPRI receiver and transmitter interfaces to the CPRI link. The CPRI specification dictates that the HDLC channel rate is specified in the three lowest bits of control byte Z.66.0. The value 3'b000 indicates that no HDLC channel is supported in the current hyperframe. Table 4–13 shows the possible rate configurations.

Table 4–13. HDLC Channel Bit Rates

Value in Z.66.0.0[2:0]	HDLC Bit Rate (Kbps)	Minimum CPRI Line Rate (Mbps)
000	—	614.4
001	240	614.4
010	480	614.4
011	960	1228.8
100	1920	2457.6
101	2400	3072.0
110	3840	4915.2
	4800	6144.0
	7680	9830.4
111	(1)	

Note to Table 4–13:

(1) When Z.66.0.0[2:0] holds value 3'b111, the HDLC bit rate is the highest HDLC bit rate possible for the current CPRI line rate. You can derive that bit rate from the other entries in this table.

The HDLC channel rate is determined during the software set-up sequence or by dynamic modification, in which the same new pointer value is received in CPRI control byte Z.66.0 four hyperframes in a row. The accepted receive rate is specified in the rx_slow_cm_rate field of the CPRI_CM_STATUS register, and the transmit rate is specified in the tx_slow_cm_rate field of the CPRI_CM_CONFIG register.

The CPU interface control for the HDLC channel is identical to the CPU interface control for the Ethernet channel, with the following exceptions:

- HDLC register names replace ETH with HDLC
- HDLC channel control has fewer configurations than the Ethernet channel control
- HDLC channel control does not support address filtering

The CPRI IP core implements the CRCDT CRC-16 allowed by the HDLC specification, rather than the CRC-32.

CPRI Protocol Interface Layer (Physical Layer)

The physical layer of the CPRI protocol is also called layer 1. This layer controls the electrical characteristics of the CPRI link, the time-division multiplexing of the separate information flows in the protocol, and low-level signaling. The CPRI protocol interface module of the CPRI IP core incorporates Altera's high-speed transceivers to implement layer 1. The transceivers are configured in deterministic latency mode, supporting the extended delay measurement requirements of the CPRI specification.

This section describes features and blocks of the CPRI protocol interface module. Figure 4–25 shows a high-level block diagram of this module.

Features

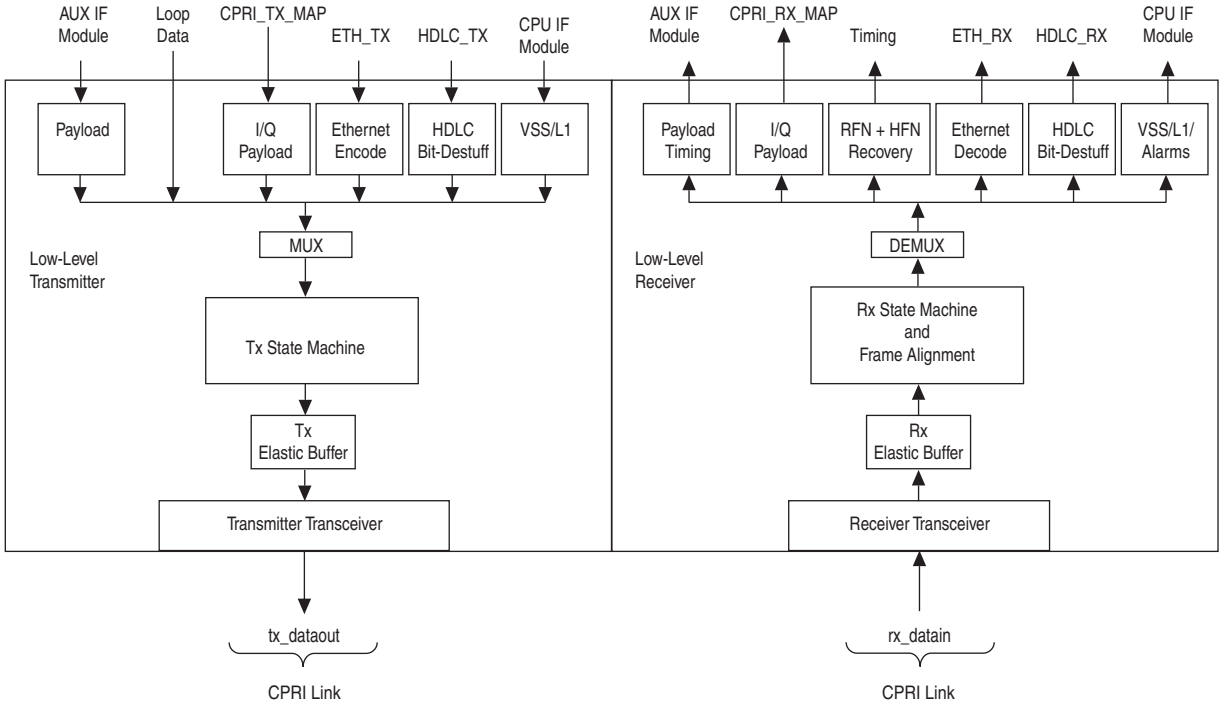
The physical layer has the following features:

- Frame synchronization
- Transmitter and receiver with the following features:
  - High-speed data serialization and deserialization
  - Clock and data recovery (receiver)
  - 8B/10B encoding and decoding
  - Frame and control word assembly and delineation
  - Error detection
  - Deterministic latency
- Software interface (status and control registers)
- Error reporting
- Clock decoupling

Physical Layer Architecture

Figure 4–25 shows the architecture of the physical layer.

Figure 4–25. Physical Layer High Level Block Diagram



Ensuring the Physical Layer Routes Your Data as Expected

Layer 1 routes data from the MAP, Auxiliary, and CPU interfaces to the outgoing CPRI frame, and routes data from the CPRI frame to the MAP, Auxiliary, and CPU interfaces. To ensure the data is routed as you intend, observe the following guidelines:

- To configure a CPRI IP core variation that supports only the AUX interface, in the CPRI parameter editor, set the number of antenna-carrier interfaces to the value of 0.
- To program a subset of the configured antenna-carrier channels as active antenna-carrier channels, set the `map_ac` field of the `CPRI_MAP_CNT_CONFIG` register to the appropriate number of channels. Refer to "Number of Antenna-Carrier Interfaces" on page 3–6. The combination of CPRI line rate, MAP interface sample width (programmed in the `map_15bit_mode` field of the `CPRI_MAP_CONFIG` register), and sampling rate (programmed in the `map_n_ac` field of the `CPRI_MAP_CNT_CONFIG` register) restricts the number of active antenna-carrier interfaces your CPRI IP core can support without data corruption. Refer to Table 4–4 and Table 4–5 on page 4–14. Programming these register fields affects how your AxC samples are packed in the data channels. You can program these register fields, and they have the same effect on the MAP interface, whether or not your CPRI IP core variation uses the AUX interface.

- If your CPRI IP core variation and application support both an AUX interface and a MAP interface, use the `cpri_tx_aux_mask` mask signal (bits [31:0] of the `aux_tx_mask_data[64:0]` bus described in Table 6–4 on page 6–7) to override the MAP interface (data) and CPU interface (control words) write access to the CPRI frame data per data bit. The mask signal is a MUX select. Setting a bit in the mask ensures the corresponding data bit inserted in the outgoing CPRI frame is data from the AUX interface. Resetting a bit in the mask ensures the corresponding bit inserted in the outgoing CPRI frame is data from the MAP interface or control words from the CPU interface.
- The AUX interface routes raw data. It passes control words unexamined as if they were data. Your application can separate the control and data words in the AUX stream if your application requires that they be separated.
- When the source of the data for the CPRI frame is not the AUX interface, you must ensure you deassert the bits in `cpri_tx_aux_mask` to prevent AUX data from being inserted in the outgoing CPRI frame.

Receiver

The receiver in the low-level interface receives the input from the CPRI link, and performs the following tasks:

- Converts the data to the main clock domain
- Performs CPRI frame detection
- Separates data and control words
- Descrambles data at 4915.2 Mbps, 6144.0, and 9830.4 Mbps CPRI line rates (optional)
- Separates data for the MAP interface block, the AUX module, the Ethernet MAC block or the MII module, and the HDLC module.
- Detects loss of signal (LOS), loss of frame (LOF), remote alarm indication (RAI), and service access point (SAP) defect indication (SDI) errors

High-Speed Transceiver

The high-speed transceiver on the CPRI IP core CPRI protocol interface is configured with the Altera ALTGX megafunction in Arria II, Cyclone IV GX, and Stratix IV GX devices, with the Altera Deterministic Latency PHY IP core in Arria V and Stratix V GX devices and in some variations in Stratix V GT devices, and with the Altera Native PHY IP core in variations with a CPRI line rate of 9830.4 Mbps in Stratix V GT devices.

The transceiver receiver implements 8B/10B decoding and the deterministic latency protocol. The deterministic latency protocol is designed to meet the 16.276 ns round-trip delay measurement accuracy requirements R21 and R21A of the CPRI specification.

For information about the high-speed transceiver blocks, refer to *volume 2* of the *Arria II Device Handbook*, to *volume 2* of the *Cyclone IV Device Handbook*, or to *volume 2* and *volume 3* of the *Stratix IV Device Handbook*.

For information about the Altera Deterministic Latency PHY IP core and the Altera Native PHY IP core, refer to the *Altera Transceiver PHY IP Core User Guide*.

Rx Elastic Buffer

The low-level interface receiver converts data from the transceiver clock domain and data width to the main CPRI IP core clock domain and data width using a synchronization FIFO called the Rx elastic buffer. The Rx elastic buffer data output is clocked with the `cpri_clkout` clock. The Rx elastic buffer data input is synchronous with the `rx_clkout` clock from the transceiver. The width of an Rx elastic buffer entry is 32 bits, and the `rx_clkout` clock clocks the transceiver data, which is 8, 16, or 32 bits wide. For details, refer to "Clock Diagrams for the CPRI IP Core" on page 4–4.

The default depth of the Rx elastic buffer is 64 32-bit entries. For most systems, the default Rx elastic buffer depth is adequate to handle dispersion, jitter, and wander that can occur on the link while the system is running. However, the **Receiver buffer depth** parameter is available for cases in which additional depth is required.

Altera recommends that you set **Receiver buffer depth** to **4** in CPRI RE slave variations, specifying a depth of 16 32-bit entries.

You must realign and resynchronize the Rx elastic buffer after a dynamic CPRI line rate change. Resynchronizing the Rx elastic buffer resets its pointers. Program the `CPRI_RX_DELAY_CTRL` register to realign and resynchronize the Rx elastic buffer.

The Rx elastic buffer adds variable delay to the Rx path through the CPRI IP core. Refer to "Extended Rx Delay Measurement" on page D–5.

Descrambling

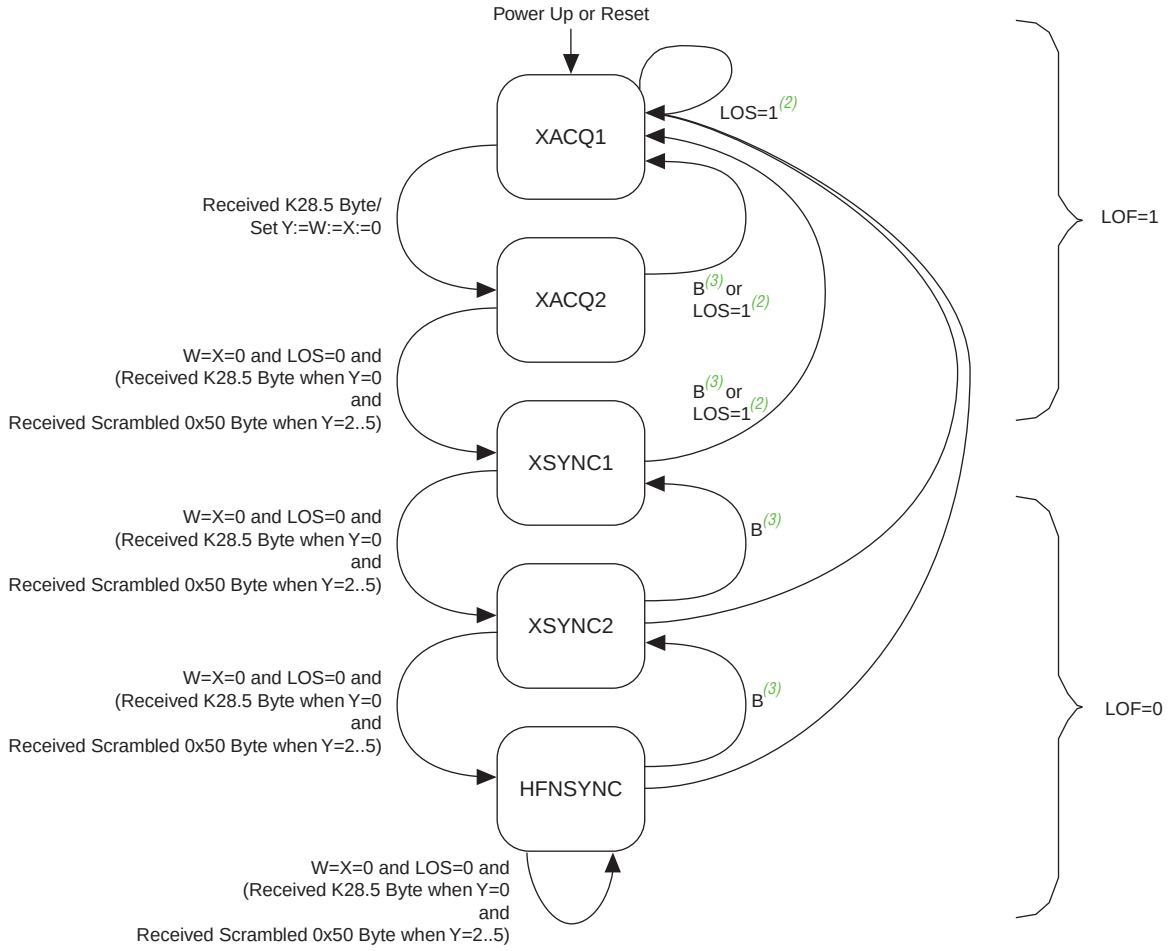
If the `tx_prot_version` field of the `CPRI_TX_PROT_VER` register (Table 7–25 on page 7–12) holds the value 2, and the CPRI data rate is 4915.2 Mbps, 6144.0 Mbps, or 9830.4 Mbps, the low-level CPRI receiver may need to descramble the incoming data, depending on the values in the `CPRI_RX_SCR_SEED` register.

When the `rx_scr_act_indication` field of the `CPRI_RX_SCR_SEED` register (Table 7–27 on page 7–12) is set, the low-level CPRI receiver descrambles the data words according to the CPRI V4.2 Specification, using the seed in the `rx_scr_seed` field of the `CPRI_RX_SCR_SEED` register. The seed value may be zero, indicating the incoming data is not scrambled.

Frame Synchronization

During frame synchronization, LOF is set to zero. LOS—the assertion of the `gxb_los` signal—resets the frame synchronization state machine. Figure 4–26 shows the frame synchronization state machine. If scrambling is configured in the CPRI link partner (based on the value at Z.2.0 in the incoming CPRI communication), additional actions and conditions apply on the state machine transitions, according to the CPRI V4.2 Specification. The CPRI IP core sets the values in the `CPRI_RX_SCR_SEED` register according to these conditions.

Figure 4–26. CPRI Frame Synchronization Machine (1)



Notes to Figure 4–26:

(1) If the tx_prot_version field of the CPRI_TX_PROT_VER register (Table 7–25 on page 7–12) holds the value 1, scrambling is not turned on. In this case, the conditions when Y is in 2..5 are ignored.

(2) LOS=1 returns the state machine to the XACQ1 state. This transition has highest priority.

(3) Condition B is: Received byte not K28.5 when Y=W=X=0 or for some k in 2..5, received byte(unscrambled) not 0x50 when W=X=0 and Y=k.

Alarm Indications

The CPRI IP core can detect and report the following alarms:

- Loss of signal (LOS)—the CPRI IP core reports this alarm in the `rx_los` field of the `CPRI_STATUS` register at offset 0x4 (Table 7–5 on page 7–3).
- Loss of frame (LOF)—the CPRI IP core reports this alarm by resetting the `rx_state` field of the `CPRI_STATUS` register at offset 0x4 (Table 7–5 on page 7–3).

Your application detects the following alarms by reading the last received #Z.130.0 control byte in the `CPRI_RX_CTRL` register:

- Remote alarm indication (RAI)
- Service access point (SAP) defect indication (SDI) errors
- Reset requests received over the CPRI link

The frame synchronization machine detects LOS and LOF directly. You can program your application to detect and respond to RAI and SDI errors as appropriate. Refer to "Accessing the Hyperframe Control Words" on page 4–39 for information about retrieving these alarms from the hyperframe control word.

The CPRI IP core handles incoming reset requests on the CPRI link by signalling the application to assert the reset signal to reset the IP core. The application reads the requests using the CPU interface. The following section describes the additional support the CPRI IP core provides to process this special command.

Reset Control Word

A CPRI IP core in master clocking mode can send a reset request through the CPRI link and a CPRI IP core in slave clocking mode can receive a reset request through the CPRI link. As required by the CPRI specification, the reset control information is sent in bit 0 of the CPRI hyperframe control word Z.130.0. This reset bit communicates both reset request and reset acknowledge.

Table 4–14 lists the signals and register fields that determine the CPRI IP core's response to a reset request received on the CPRI link and that determine whether it sends a reset request on the CPRI link.

Table 4–14. Conditions That Trigger a Reset Request or Enable a Reset Acknowledge on the CPRI Link

Register or Signal Name	Register Bits	Field Name	Trigger Conditions for Sending Reset Request (Master) or ACK (Slave)	
CPRI_HW_RESET (Table 7-12)	[0]	reset_gen_en	1	—
	[1]	reset_gen_force	1	—
	[3]	reset_hw_en	0	1
hw_reset_assert (Table 6-15)	—	—	—	1

A CPRI IP core in master mode transmits a reset request to the RE slave nodes to which it is connected under either of the trigger conditions shown in Table 4–14. The behavior of a CPRI IP core in slave mode that receives a reset request on the CPRI link depends on the same enable fields in its own CPRI_HW_RESET register. For reset acknowledgements, as for the original reset request conditions, if the reset_hw_en bit is asserted, the reset_gen_en bit is ignored.

The CPRI specification requires that the Z.130.0 reset bit must be detected by the CPRI partner in ten consecutive hyperframes before the CPRI partner confirms the reset request. The reset generation request is in effect while the condition that triggered the reset request remains in effect, until the reset acknowledge control bit is detected on the incoming CPRI link.

To abort a reset request, set or reset a register field to negate the condition. Specifically, to abort a reset request made by asserting the reset_gen_force bit in the CPRI_HW_RESET register, set the reset_gen_en bit of the CPRI_HW_RESET register to 0. To abort a reset request made by asserting the hw_reset_assert input signal, set the reset_hw_en bit of the CPRI_HW_RESET register to 0.

To acknowledge the reset request, the CPRI transmitter must send a reset acknowledge on the CPRI link, by setting the Z.130.0 reset bit in five consecutive outgoing hyperframes. If one of the acknowledgement conditions in Table 4–14 holds, the CPRI transmitter sends the reset acknowledge on the CPRI link. If the `reset_out_en` bit of the `CPRI_HW_RESET` register is set, the CPRI IP core asserts the external `hw_reset_req` signal until the reset occurs. This signal informs the application layer of the low-level reset request.

After it transmits the five consecutive reset acknowledge bits, the CPRI transmitter sets the `reset_gen_done` and `reset_gen_done_hold` bits of its own `CPRI_HW_RESET` register. If the `reset_hw_en` bit is set and the `hw_reset_req` signal is asserted, you must set the `hw_reset_assert` signal, to tell the CPRI transmitter to send a reset acknowledge on the CPRI link.

For more information about the `CPRI_HW_RESET` register, refer to Table 7–12 on page 7–5. For more information about the `hw_reset_assert` input signal, refer to Table 6–15 on page 6–17.

After reset, your software must perform link synchronization and other initialization tasks. For information about the required initialization sequence following CPRI IP core reset, refer to Appendix A, Initialization Sequence.

Transmitter

The transmitter in the low-level interface transmits output to the CPRI link. This module performs the following tasks:

- Assembles data and control words in proper output format
- Transmits standard frame sequence
- Optionally scrambles the outgoing data transmission at 4915.2 Mbps, 6144.0 Mbps, and 9830.4 Mbps CPRI line rates
- Inserts the following control words in their appropriate locations in the outgoing hyperframe:
  - Synchronization control byte (K28.5) and filler bytes (D16.2) in the synchronization control word
  - Hyperframe number (HFN)
  - Basic frame number (BFN)
  - HDLC bit rate
  - Pointer to start of Ethernet data in current frame
  - 4B/5B-encoded fast C&M Ethernet frames
  - Bit-stuffed slow C&M HDLC frames
  - Enabled control transmit table entries
- Converts the data to the transceiver clock domain.

When no data is available to transmit on the CPRI link, the transmitter transmits the standard frame sequence with zeroed control words and all-zero data.

Scrambling

When the `tx_prot_version` field of the `CPRI_TX_PROT_VER` register (Table 7–25 on page 7–12) holds the value 2, the low-level CPRI transmitter scrambles the data words according to the CPRI V4.2 Specification, using the seed in the `tx_scr_seed` field of the `CPRI_TX_SCR_SEED` register (Table 7–26 on page 7–12).

Tx Elastic Buffer

The low-level interface transmitter converts data from the main CPRI IP core clock domain and data width to the transceiver clock domain and data width using a synchronization FIFO called the Tx elastic buffer. The Tx elastic buffer data input is clocked with the `cpri_clkout` clock, and the buffer data output is clocked with the `tx_clkout` clock from the transceiver. Data in the Tx elastic buffer is 32 bits wide, and the data bus to the transceiver is 8, 16, or 32 bits wide, depending on the target device family and the CPRI line rate. The CPRI IP core derives the `cpri_clkout` clock from the Tx output clock of the transceiver, divided as necessary to support the data width conversion to and from the 32-bit wide elastic buffers. Table 4–15 shows the data bus widths and clock divisors for the different device families and CPRI line rates.

Table 4–15. Transceiver Datapath Width and tx_clkout Divider

CPRI Line Rate (Mbps)	Device Family	Transceiver Datapath Width (Bits)	tx_clkout Divider
614.4	All	8	4
Greater than 614.4	Arria II GX, Cyclone IV GX	16	2
	Arria II GZ, Arria V, Stratix IV GX, and Stratix V	32	1

High-Speed Transceiver

The high-speed transceiver on the CPRI IP core CPRI protocol interface is configured with the Altera ALTGX megafunction in Arria II, Cyclone IV GX, and Stratix IV GX devices, with the Altera Deterministic Latency PHY IP core in Arria V and Stratix V GX devices and in some variations in Stratix V GT devices, and with the Altera Native PHY IP core in variations with a CPRI line rate of 9830.4 Mbps in Stratix V GT devices.

The transceiver transmitter implements 8B/10B encoding and the deterministic latency protocol. It transforms the 16-bit parallel input data to the Arria II GX or Cyclone IV GX transmitter, or 32-bit parallel input data to the Arria II GZ, Arria V, Stratix IV GX, or Stratix V transmitter, to 8-bit data before 8B/10B encoding. The 10-bit encoded data is then serialized and sent to the CPRI link differential output pins.

The deterministic latency protocol is designed to meet the 16.276-ns round-trip delay measurement accuracy requirements R21 and R21A of the CPRI specification.

For information about the high-speed transceiver blocks, refer to *volume 2* of the *Arria II Device Handbook*, to *volume 2* of the *Cyclone IV Device Handbook*, or to *volume 2* and *volume 3* of the *Stratix IV Device Handbook*.

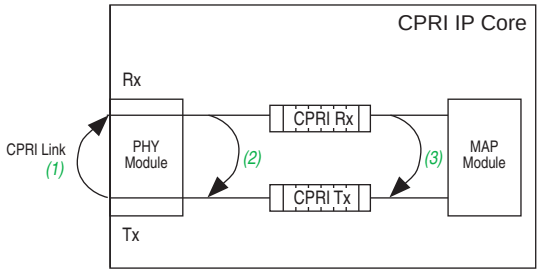
For information about the Altera Deterministic Latency PHY IP core and the Altera Native PHY IP core, refer to the *Altera Transceiver PHY IP Core User Guide*.

This chapter describes the loopback and PRBS testing features of the CPRI IP core.

Loopback Modes

The CPRI IP core supports multiple loopback modes to help you test your CPRI design. Figure 5–1 illustrates the supported loopback paths.

Figure 5–1. CPRI IP Core Supported Loopback Paths



Notes to Figure 5–1:

(1) External loopback mode to test a single CPRI REC master.
(2) Internal reverse loopback mode configured in an RE slave's CPRI_PHY_LOOP register.
(3) Internal reverse loopback mode configured in an RE slave's CPRI_CONFIG register.

The following sections describe these loopback modes.

External Loopback

The CPRI IP core supports an external loopback configuration on the CPRI link. You can use this configuration to test the full Tx and Rx paths from an application, through the CPRI link, and back to the application.

The CPRI testbenches provided in your CPRI IP installation configure the DUT in this loopback mode. Refer to Chapter 8, Testbenches.

To configure this loopback mode, you connect a CPRI REC master's CPRI Tx interface to its CPRI Rx interface by physically connecting the CPRI IP core's high-speed transceiver output pins to its high-speed transceiver input pins. As for any CPRI link, the connection medium must support the data rate requirements of the CPRI IP core. Altera recommends that you implement this type of loopback connection through an SFP cable.

Only an REC master can function correctly in a CPRI link external loopback configuration. An RE slave in external loopback configuration cannot achieve frame synchronization, because the CPRI Rx interface must lock on to the K28.5 character before the CPRI Tx interface can begin sending K28.5 characters. Therefore, no K28.5 character is ever transmitted on the RE slave loopback CPRI link.

Internal Reverse Loopback

The CPRI IP core supports two different internal reverse loopback paths that you can configure in software in a CPRI RE slave, and multiple loopback modes along those paths. The following sections describe these modes.

Physical Layer Loopback Mode

In the physical layer reverse loopback mode, a CPRI RE slave sends CPRI frames of incoming CPRI data and control words from the PHY module back through the PHY module in outgoing CPRI communication. The PHY reverse loopback path is labeled (2) in Figure 5–1.

In this mode, the PHY reverse loopback path is active whether or not frame synchronization has been achieved. The path includes 8B10B encoding and decoding, but only enough core CPRI functionality to handle the transition from the receiver clock domain to the transmitter clock domain.

You configure a CPRI RE slave in physical layer loopback mode by setting the `loop_mode` bit in the `CPRI_PHY_LOOP` register described in Table 7–13 on page 7–6. If this bit is set, the reverse loopback path through the CPRI Rx and Tx buffers is not active, irrespective of any setting that should activate that path.

Reverse Loopback Through CPRI Rx and Tx Buffers

The CPRI IP core provides support for an additional, more comprehensive testing loopback path in several different modes. The testing loopback modes activate a reverse loopback path that sends incoming CPRI communication from the CPRI Rx buffer back through the CPRI Tx buffer and the PHY module to the CPRI link in outgoing CPRI communication. This testing loopback path is labeled (3) in Figure 5–1.

Several loopback modes are available on this reverse loopback path. You can specify that full CPRI frames, including all incoming CPRI data and control words, are sent back in outgoing CPRI communication. You can also specify that only data be looped back, or that only certain categories of control words be looped back. In these modes, the CPRI RE slave generates the remainder of the outgoing CPRI frame content locally.

You configure a CPRI RE slave in testing loopback mode by setting the appropriate value in the `loop_mode` field of the `CPRI_CONFIG` register described in Table 7–6 on page 7–3. The register description includes the full encodings to specify the different loopback mode values.

PRBS Generation and Validation

The CPRI IP core supports generation and validation of several predetermined pseudo-random binary sequences (PRBS) for antenna-carrier interface and Rx and Tx path testing.

The MAP interface module generates and checks the PRBS. If you configure no antenna-carrier interfaces in your CPRI IP core, your IP core does not include a MAP block and therefore does not support PRBS testing.

The value in the prbs_mode field of the CPRI_PRBS_CONFIG register (Table 7–44 on page 7–20) specifies whether the MAP interface module is in data mode or in PRBS mode, and the generated pattern for loopback mode. The value applies to all AxC interfaces. The following prbs_mode values are available:

- 00: Indicates that data samples, and not a PRBS test pattern, are expected on the AxC interfaces. This value indicates the MAP interface module is not in PRBS mode.
- 01: Indicates an incremental counter sequence, starting at zero at the start of a 10 ms radio frame, and counting to 255 before rolling over. The counter value appears in both halves of the 32-bit data word.
- 10: Indicates an inverted $2^{23} – 1$ PRBS sequence. Each pattern appears in both halves of the 32-bit data word.

The value 11 is reserved.

The CPRI_PRBS_STATUS register (Table 7–45 on page 7–20) records the PRBS error detection status for each AxC interface.

You can perform PRBS testing with a single REC master across a CPRI link in loopback configuration, or across a CPRI link between two CPRI IP cores. To perform PRBS testing across a CPRI link between two CPRI IP cores, you must program the RE slave in reverse loopback mode and then program the REC master in PRBS mode.

To perform PRBS testing across a CPRI link, perform the following steps:

1. In the CPRI slave, program one of these registers to set up an internal reverse loopback path:
   - Set the loop_mode field of the CPRI_PHY_LOOP register to the value of 1. This loopback mode and the register are described in "Loopback Modes" on page 5–1 and in Table 7–13 on page 7–6.
   - Set the loop_mode field of the CPRI_CONFIG register to the value of 2'b001 or 2'b010. The value of 2'b001 specifies that all data and control words are looped back. The value of 2'b010 specifies that all data is looped back, and that the CPRI RE slave generates the outgoing control words locally. The PRBS pattern is restricted to the data words in the incoming CPRI frame, so either of these two loopback modes is adequate to send the full PRBS pattern back to the generating CPRI REC master.

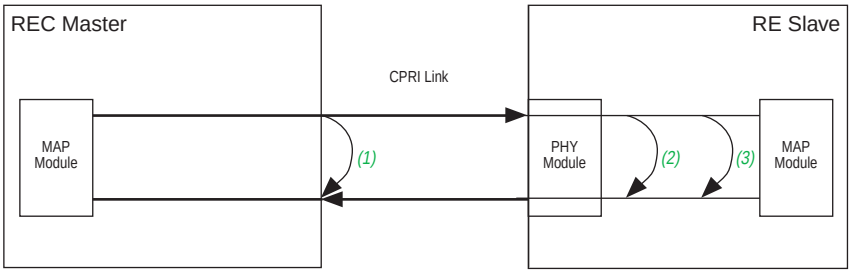
     These loopback modes and the register are described in "Loopback Modes" on page 5–1 and in Table 7–6 on page 7–3.
2. In the CPRI master, program the prbs_mode field of the CPRI_PRBS_CONFIG register for your preferred PRBS pattern according to the information in this section and in Table 7–44 on page 7–20.

The internal loopback mode you select determines the extent of the Rx and Tx path testing in the RE slave IP core. For information about the two internal reverse loopback modes and the differences between them, refer to "Loopback Modes" on page 5–1.

To perform PRBS testing across a CPRI link in external loopback configuration, connect the CPRI IP core's high-speed transceiver output to its high-speed transceiver input, and after the CPU interface is available for programming, perform step 2.

Figure 5–2 shows the three different loopback modes that support PRBS testing.

Figure 5–2. CPRI IP Core Loopback Modes That Support PRBS Testing



Notes to Figure 5–2:

(1) External loopback mode to test a single CPRI REC master.
(2) Internal reverse loopback mode (physical layer loopback mode) configured in the RE slave's CPRI_PHY_LOOP register.
(3) Internal reverse loopback mode (testing loopback mode) configured in the RE slave's CPRI_CONFIG register.

6. Signals

This chapter describes all the top-level signals of the Altera CPRI IP core.

MAP Interface Signals

Table 6–1 and Table 6–2 list the signals used by the MAP interface modules of the CPRI IP core. The MAP interfaces are implemented as Avalon-ST interfaces.

Refer to the *Avalon Interface Specifications* for details about the Avalon-ST interface.

MAP Receiver Signals

The behavior of many of the MAP receiver interface signals depends on the CPRI IP core's current MAP Rx synchronization mode. The mode is determined by your selection in the CPRI parameter editor and by the `CPRI_MAP_CONFIG` register (Table 7–31 on page 7–14), as shown in Table 4–6 on page 4–16.

"MAP Receiver Interface" on page 4–15 includes a description of signal handshaking in all three synchronization modes, and timing diagrams that illustrate the expected behavior of these signals.For a summary of signal availability in the different synchronization modes, refer to Table 4–8 on page 4–16.

Table 6–1 lists the MAP receiver interface signals.

Table 6–1. MAP Receiver Interface Signals (Part 1 of 3)

Signal	Direction	Description
`map{23…0}_rx_clk`	Input	Clock signal for each antenna-carrier interface. These clocks are not supported in the internally-clocked mode. In the interally-clocked mode, `cpri_clkout` clocks the antenna-carrier interfaces.
`map{23…0}_rx_reset`	Input	Reset signal for each antenna-carrier interface in synchronous buffer mode and in FIFO mode. This reset is associated with the `mapN_rx_clk` clock. These signals are not supported in the internally-clocked mode. `mapN_rx_reset` can be asserted asynchronously, but must stay asserted at least one cycle of the associated clock and must be deasserted synchronously with that clock. Refer to Figure 4–5 on page 4–10 for a circuit that shows how to enforce synchronous deassertion of a reset signal.

Table 6–1. MAP Receiver Interface Signals (Part 2 of 3)

Signal	Direction	Description
map{23…0}_rx_ready	Input	Read-ready signal for each antenna-carrier interface, in FIFO mode. Indicates to the CPRI IP core that the application is ready to receive data on the corresponding data channel in the next clock cycle. Asserted by the sink to mark ready cycles, which are cycles in which transfers can occur. If ready is asserted on cycle N, the cycle (N+READY_LATENCY) is a ready cycle. The MAP receiver interface in FIFO mode is designed for READY_LATENCY equal to 1. In synchronous buffer mode, the application must hold the mapN_rx_ready signals high continuously. In the internally-clocked mode, the CPRI IP core ignores this signal.
map{23…0}_rx_data[31:0]	Output	32-bit read data being transmitted on each antenna-carrier interface. Bits [15:0] are the I component of the IQ sample. Bits [31:16] are the Q component of the IQ sample. In FIFO mode, data is valid as early as one mapN_rx_clk clock cycle after the application asserts the read-ready input signal mapN_rx_ready, but is only valid while the CPRI IP core asserts the mapN_rx_valid signal. In synchronous buffer mode, data is valid one mapN_rx_clk clock cycle after the application asserts the mapN_rx_resync signal. To ensure valid data in synchronous buffer mode, the application should only assert the mapN_rx_resync signal after the CPRI IP core asserts the cpri_rx_start signal. However, the CPRI IP core does not enforce this requirement. In the internally-clocked mode, data is valid one cpri_clkout clock cycle after the CPRI IP core asserts the mapN_rx_start output signal, but is only valid while the CPRI IP core asserts the mapN_rx_valid signal.
map{23…0}_rx_valid	Output	Valid signal for FIFO mode and for the internally-clocked synchronization mode. In FIFO mode, this signal is asserted when the mapN Rx buffer exceeds the threshold level in the map_rx_ready_thr field of the CPRI_MAP_RX_READY_THR register. Although each data channel has its own mapN_rx_valid signal, all data channels use the same map_rx_ready_thr threshold value. This signal qualifies all the other output signals of the MAP receiver interface. On every rising edge of the clock at which mapN_rx_valid is high, mapN_rx_data can be sampled. In the internally-clocked mode, the CPRI IP core asserts each mapN_rx_valid signal one cpri_clkout clock cycle after it asserts the corresponding mapN_rx_start signal. In synchronous buffer mode,the map{23...0}_rx_valid signals do not participate in data transfer synchronization, and the application should ignore these signals.

Table 6–1. MAP Receiver Interface Signals (Part 3 of 3)

Signal	Direction	Description
map{23…0}_rx_resync	Input	Resynchronization signal for use in synchronous buffer mode. When this signal is asserted, the read pointer of the mapN Rx buffer is reset to zero. This signal is synchronous to the mapN_rx_clk clock. To ensure valid data in synchronous buffer mode, the application should only assert the mapN_rx_resync signal after the CPRI IP core asserts the cpri_rx_start signal. However, the CPRI IP core does not enforce this requirement. In FIFO mode the map{23...0}_rx_resync signals do not participate in data transfer synchronization, and the CPRI IP core ignores these signals. In the internally-clocked mode, these signals are not present.
map{23…0}_rx_start	Output	In the internally-clocked mode, the CPRI IP core asserts each mapN_rx_start signal to indicate the start of valid data on the corresponding antenna-carrier interface (mapN_rx_data) in the current 10 ms radio frame. This signal is synchronous with the cpri_clkout clock. When it asserts mapN_rx_start, the CPRI IP core also asserts the mapN_rx_valid signal and transmits valid data on the corresponding antenna-carrier interface. In FIFO mode and in synchronous buffer mode, the map{23...0}_rx_start signals do not participate in data transfer synchronization, and the application should ignore these signals.
map{23…0}_rx_status_data[2:0]	Output	This vector contains the following status bits: [2] cpri_map_rx_overflow: Rx FIFO overflow indicator for this antenna-carrier interface. This signal is synchronous to the cpri_clkout clock, and is asserted following a write to a full buffer. This signal reflects the value in the appropriate bit of the buffer_rx_overflow field of the CPRI_IQ_RX_BUF_STATUS register (Table 7–48 on page 7–21). [1] cpri_map_rx_underflow: Rx FIFO underflow indicator for this antenna-carrier interface. This signal is synchronous to the cpri_clkout clock, and is asserted following a read from an empty buffer. This signal reflects the value in the appropriate bit of the buffer_rx_underflow field of the CPRI_IQ_RX_BUF_STATUS register (Table 7–48 on page 7–21). [0] cpri_map_rx_en: Indicates that this antenna-carrier interface is enabled. The value is determined in the CPRI_IQ_RX_BUF_CONTROL register. Use this signal to disable external logic for inactive AxC interfaces and to map interface clock gating to save power.

MAP Transmitter Signals

The behavior of many of the MAP transmitter interface signals depends on the CPRI IP core's current TX synchronization mode. The mode is determined by your selection in the CPRI parameter editor and by the CPRI_MAP_CONFIG register (Table 7–31 on page 7–14), as shown in Table 4–9 on page 4–22.

"MAP Transmitter Interface" on page 4–21 includes a description of signal handshaking in all three synchronization modes, and timing diagrams that illustrate the expected behavior of these signals. For a summary of signal availability in the different synchronization modes, refer to Table 4–11 on page 4–22.

Table 6–2 lists the MAP transmitter interface signals.

Table 6–2. MAP Transmitter Interface Signals (Part 1 of 2)

Signal	Direction	Description
map{23…0}_tx_clk	Input	Clock signal for each antenna-carrier interface. These clocks are not supported in the internally-clocked mode. In the interally-clocked mode, cpri_clkout clocks the antenna-carrier interfaces.
map{23…0}_tx_reset	Input	Reset signal for each antenna-carrier interface in synchronous buffer mode and in FIFO mode. This reset is associated with the mapN_tx_clk clock. These signals are not supported in the internally-clocked mode. mapN_tx_reset can be asserted asynchronously, but must stay asserted at least one cycle of the associated clock, and must be deasserted synchronously with that clock. Refer to Figure 4–5 on page 4–10 for a circuit that shows how to enforce synchronous deassertion of a reset signal.
map{23…0}_tx_valid	Input	Write-valid signal for each antenna-carrier interface. This signal qualifies all the other Avalon-ST input signals of the MAP transmitter interface. On every rising edge of the clock at which mapN_tx_valid is high, data is sampled by the CPRI IP core. In FIFO mode, the application can assert mapN_tx_valid in any mapN_tx_clk cycle immediately following a mapN_tx_clk cycle in which the CPRI IP core asserts the mapN_tx_ready signal for the corresponding antenna-carrier interface. In synchronous buffer mode, the application must assert the mapN_tx_valid signal at the same time as or immediately after it asserts the mapN_tx_resync resynchronization signal. However, Altera recommends that the application assert these two signals simultaneously. Refer to "MAP Transmitter in Synchronous Buffer Mode" on page 4–24. In the internally-clocked mode, the application must wait at least one cpri_clkout cycle after the IP core asserts mapN_tx_ready before asserting the mapN_tx_valid signal; READY_LATENCY is 1.
map{23…0}_tx_data[31:0]	Input	32-bit write data from each antenna-carrier interface. Data is valid starting one mapN_tx_clk clock cycle (cpri_clkout clock cycle in the internally-clocked mode) after the write-valid bit is asserted. Bits [15:0] are the I component of the IQ sample. Bits [31:16] are the Q component of the IQ sample.

Table 6–2. MAP Transmitter Interface Signals (Part 2 of 2)

Signal	Direction	Description
map{23…0}_tx_ready	Output	Ready signal for each antenna-carrier interface. In FIFO mode, the ready signal is asserted when the mapN Tx buffer falls below the threshold level in the map_tx_ready_thr field of the CPRI_MAP_TX_READY_THR register. Although each data channel has its own mapN_tx_ready signal, all data channels use the same map_tx_ready_thr threshold value. Indicates that the CPRI IP core is ready to receive data on the data channel in the current clock cycle. Asserted by the Avalon-ST sink to mark ready cycles, which are the cycles in which transfers can take place. If ready is asserted on cycle N, the cycle (N+READY_LATENCY) is a ready cycle. In the MAP transmitter interface in FIFO mode, READY_LATENCY is equal to 0, so the cycle on which mapN_tx_ready is asserted is the ready cycle. In the internally-clocked mode, the CPRI IP core asserts the ready signal one cycle before the antenna-carrier interface is ready to receive data on the data channel. In this mode, READY_LATENCY is equal to 1. In synchronous buffer mode, the map{23...0}_tx_ready signals do not participate in data transfer synchronization, and the application should ignore these signals.
map{23…0}_tx_resync	Input	Resynchronization signal for use in synchronous buffer mode. This signal is synchronous to the mapN_tx_clk clock. In FIFO mode the map{23...0}_tx_resync signals do not participate in data transfer synchronization, and the CPRI IP core ignores these signals. In the internally-clocked mode, these signals are not present.
map{23…0}_tx_status_data	Output	This vector contains the following status bits: [2] cpri_map_tx_overflow: Tx FIFO overflow indicator for this antenna-carrier interface. This signal is synchronous to the cpri_clkout clock, and is asserted following a write to a full buffer. This signal reflects the value in the appropriate bit of the buffer_tx_overflow field of the CPRI_IQ_TX_BUF_STATUS register (Table 7–49 on page 7–21). [1] cpri_map_tx_underflow: Tx FIFO underflow indicator for this antenna-carrier interface. This signal is synchronous to the cpri_clkout clock, and is asserted following a read from an empty buffer. This signal reflects the value in the appropriate bit of the buffer_tx_underflow field of the CPRI_IQ_TX_BUF_STATUS register (Table 7–49 on page 7–21). [0] cpri_map_tx_en: Indicates that this antenna-carrier interface is enabled. The value is determined in the CPRI_IQ_TX_BUF_CONTROL register. Use this signal to disable external logic for inactive AxC interfaces and to map interface clock gating to save power.

Auxiliary Interface Signals

Table 6–3 through Table 6–4 list the signals on the CPRI IP core auxiliary interface. All the signals in Table 6–3 through Table 6–4 are clocked by the internal clock visible on the cpri_clkout port.

AUX Receiver Signals

Table 6–3 lists the signals on the AUX receiver interface. For additional information about these signals, refer to "AUX Receiver Module" on page 4–28.

Table 6–3. AUX Receiver Interface Signals

Signal	Direction	Bit	Description
aux_rx_status_data [75:0]	Output	[75]	cpri_rx_rfp: Synchronization pulse for start of 10 ms radio frame. The pulse occurs at the start of the radio frame on the CPRI receiver interface.
		[74]	cpri_rx_start: Indicates the start of the first basic frame on the AUX interface, and can be used by an AxC software application to trigger the AxC-specific resynchronization signal used in the MAP interface synchronous buffer mode. The cpri_rx_start signal is asserted at the offset defined in the CPRI_START_OFFSET_RX register. The count to the offset starts at the cpri_rx_rfp or cpri_rx_hfp pulse, depending on values set in the register. Refer to Table 7–39 on page 7–18. The signal is asserted for the duration of the basic frame.
		[73]	cpri_rx_hfp: Synchronization pulse for start of hyperframe. The pulse occurs at the start of the hyperframe on the CPRI receiver interface.
		[72:61]	cpri_rx_bfn: Current radio frame number.
		[60:53]	cpri_rx_hfn: Current hyperframe number. Value is in the range 0–149.
		[52:45]	cpri_rx_x: Index number of the current basic frame in the current hyperframe. Value is in the range 0–255.
		[44:39]	cpri_rx_k: Sample counting K counter. Counts the basic frame position of the AxC Container Block for mapping IQ samples when map_mode field in the CPRI_MAP_CONFIG register has value 01 or 10. This signal is not used when map_mode value is 00.
		[38:33]	cpri_rx_seq: Index number of the current 32-bit word in the current basic frame being transmitted on the AUX link. Depending on the CPRI line rate, this signal has the following range: ■ 614.4 Mbps line rate: range is 0 –3 ■ 1228.8 Mbps line rate: range is 0–7 ■ 2457.6 Mbps line rate: range is 0–15 ■ 3072.2 Mbps line rate: range is 0–19 ■ 4915.2 Mbps line rate: 0–31 ■ 6144.0 Mbps line rate: 0–39 ■ 9830.4 Mbps line rate: 0–63
		[32]	cpri_rx_sync_state: When set, indicates that Rx, HFN, and BFN synchronization have been achieved in CPRI receiver frame synchronization.
		[31:0]	cpri_rx_aux_data: Data transmitted on the AUX link. Data is transmitted in 32-bit words. Byte [31:24] is transmitted first, and byte [7:0] is transmitted last.

AUX Transmitter Signals

Table 6–4 lists the signals on the AUX transmitter interface. For additional information about these signals, refer to "AUX Transmitter Module" on page 4–31.

Table 6–4. AUX Transmitter Interface Signals (Part 1 of 2)

Signal	Direction	Bits	Description
aux_tx_status_data [43:0]	Output	[43]	cpri_tx_error: Indicates that in the previous cpri_clkout cycle, the cpri_tx_aux_mask[31:0] mask bits were not deasserted during K28.5 character insertion in the outgoing CPRI frame (which occurs when Z=X=0).
		[42:37]	cpri_tx_seq: Index number of the current 32-bit word in the two-cycle-offset basic frame to be received on the AUX link. Depending on the CPRI line rate, this signal has the following range: ■ 614.4 Mbps line rate: range is 0 –3 ■ 1228.8 Mbps line rate: range is 0–7 ■ 2457.6 Mbps line rate: range is 0–15 ■ 3072.2 Mbps line rate: range is 0–19 ■ 4915.2 Mbps line rate: 0–31 ■ 6144.0 Mbps line rate: 0–39 ■ 9830.4 Mbps line rate: 0–63
		[36:31]	cpri_tx_k: Sample counting K counter. Counts the basic frame position of the AxC Container Block for mapping IQ samples when map_mode field in the CPRI_MAP_CONFIG register has value 01 or 10. This signal is not used when map_mode value is 00.
		[30:23]	cpri_tx_x: Index number of the current basic frame in the current hyperframe. Value is in the range 0–255.
		[22:15]	cpri_tx_hfn: Current hyperframe number. Value is in the range 0–149.
		[14:3]	cpri_tx_bfn: Current radio frame number.
		[2]	cpri_tx_hfp: Synchronization pulse for start of hyperframe. The pulse occurs at the start of the hyperframe on the CPRI transmitter interface.
		[1]	cpri_tx_start: Indicates the start of the first basic frame on the AUX interface, and can be used by an AxC software application to trigger the AxC-specific resynchronization signal used in MAP synchronous buffer mode. The cpri_tx_start signal is asserted at the offset defined in the CPRI_START_OFFSET_TX register. The count to the offset starts at the cpri_tx_rfp or cpri_tx_hfp pulse, depending on values set in the register. Refer to Table 7–40 on page 7–18. The signal is asserted for the duration of the basic frame.
		[0]	cpri_tx_rfp: Synchronization pulse for start of 10 ms radio frame. The pulse occurs at the start of the radio frame on the CPRI transmitter interface.

Table 6–4. AUX Transmitter Interface Signals (Part 2 of 2)

Signal	Direction	Bits	Description
aux_tx_mask_data [64:0]	Input	[64]	cpri_tx_sync_rfp: Synchronization input used in REC master to control the start of a new 10 ms radio frame. Asserting this signal resets the frame synchronization machine. The CPRI IP core uses the rising edge of the pulse for synchronization. For information about the CPRI IP core response to a pulse on this signal, refer to Figure 4–20 on page 4–34 and surrounding text.
		[63:32]	cpri_tx_aux_data: Data received on the AUX link, aligned with cpri_tx_seq with a delay of two cpri_clkout cycles. Data is transmitted in 32-bit words. Byte [31:24] is transmitted first, and byte [7:0] is transmitted last.
		[31:0]	cpri_tx_aux_mask: Bit mask for insertion of data from cpri_tx_aux_data in the outgoing CPRI frame. Assertion of a bit in this mask overrides insertion of data to the corresponding bit in the outgoing CPRI frame from any other source. Therefore, the mask bits must be deasserted during K28.5 character insertion in the outgoing CPRI frame, which occurs when Z=X=0. If you do not deassert the mask bits during K28.5 character insertion in the outgoing CPRI frame, the cpri_tx_error output signal is asserted in the following cpri_clkout cycle.

Extended Rx Status Signals

Table 6–5 lists the signals on the extended Rx status interface. All of these signals report on the status of the CPRI receiver frame synchronization machine.

Table 6–5. Extended Rx Status Signals

Signal	Direction	Bits	Description
extended_rx_status_data [11:0]	Output	[11]	cpri_rx_los: CPRI receiver LOS indication (active high). This bit reflects the value in the rx_los field of the CPRI_INTR register (Table 7–4 on page 7–2).
		[10:8]	cpri_rx_lcv: Current CPRI receiver 8B/10B line code violation count in current clock cycle. This information enables CPRI link debug when the control word does not appear or is malformed.
		[7]	cpri_rx_hfn_state: When set, indicates that hyperframe synchronization (HFN) has been achieved in CPRI receiver frame synchronization.
		[6]	cpri_rx_bfn_state: When set, indicates that basic frame synchronization (BFN) has been achieved in CPRI receiver frame synchronization.
		[5]	cpri_rx_freq_alarm: Frequency alarm. When set, indicates a frequency difference greater than four clock cycles between cpri_clkout and the recovered received clock from the CPRI receiver interface.
		[4:2]	cpri_rx_cnt_sync: CPRI receiver frame synchronization state machine state number. Tracks the number of the current state in its state type. When the state machine is in state XACQ1, the value of cpri_rx_cnt_sync is 0; when the state is XACQ2, cpri_rx_cnt_sync has value 1; when the state is XSYNC1, cpri_rx_cnt_sync has value 0; and so on. Refer to Figure 4–26 on page 4–50.
		[1:0]	cpri_rx_state: Indicates the type of state of the CPRI receiver frame synchronization state machine. The following values are defined: 00 - LOS state 01 - XACQ state 10 - XSYNC state 11 - HFNSYNC state In the HFNSYNC state (cpri_rx_state has value 0x3 and cpri_rx_cnt_sync has value 0x1), Rx synchronization has been achieved, except for initialization of the hyperframe and basic frame numbers. You must wait for cpri_rx_hfn_state and cpri_rx_bfn_state to have value 1, indicating that the hyperframe number and basic frame number are initialized.

CPRI MII Signals

Table 6–6 and Table 6–7 list the signals used by the CPRI MII module of the CPRI IP core. The CPRI MII is enabled if you turn off **Include MAC block** in the CPRI parameter editor. The CPRI MII signals are available only if you enable the CPRI MII. For information about the MII handshaking protocol implementation, refer to "Media Independent Interface to an External Ethernet Block" on page 4–34.

CPRI MII Receiver Signals

Table 6–6 lists the CPRI MII receiver signals.

Table 6–6. CPRI MII Receiver Interface Signals

Signal	Direction	Description
cpri_mii_rxclk	Output	Clocks the MII receiver interface. The cpri_clkout clock drives this signal.
cpri_mii_rxwr	Output	Ethernet write signal. Indicates the presence of a new K nibble or data value on cpri_mii_rxd[3:0]. This signal is asserted during the first cpri_mii_rxclk cycle in which the K nibble or a new data value appears on cpri_mii_rxd[3:0].
cpri_mii_rxdv	Output	Ethernet receive data valid. Indicates the presence of valid data or initial K nibble on cpri_mii_rxd[3:0].
cpri_mii_rxer	Output	Ethernet receive error. Indicates an error in the current nibble of cpri_mii_rxd or indicates that the CPRI link is not initialized, and therefore an error might be present in the frame being transferred to the external Ethernet block. This signal is deasserted at reset, and asserted after reset until the CPRI IP core achieves frame synchronization.
cpri_mii_rxd[3:0]	Output	Ethernet receive nibble data. Data bus for data from the CPRI IP core to the external Ethernet block. All bits are deasserted during reset, and all bits are asserted after reset until the CPRI IP core achieves frame synchronization.

CPRI MII Transmitter Signals

Table 6–7 lists the CPRI MII transmitter signals. These signals are available if you exclude the MAC block from the CPRI IP core.

Table 6–7. CPRI MII Transmitter Interface Signals (Part 1 of 2)

Signal	Direction	Description
cpri_mii_txclk	Output	Clocks the MII transmitter interface. The cpri_clkout clock drives this signal.
cpri_mii_txen	Input	Valid signal from the external Ethernet block, indicating the presence of valid data on cpri_mii_txd[3:0]. This signal is also asserted while the CPRI MII transmitter block inserts J and K nibbles in the data stream to form the start-of-packet symbol. This signal is typically asserted one cycle after cpri_mii_txrd is asserted. After that first cycle following the assertion of cpri_mii_txrd, if cpri_mii_txen is not yet asserted, the CPRI MII transmitter module inserts Idle cycles until the first cycle in which cpri_mii_txen is asserted. If cpri_mii_txen is asserted and subsequently deasserted while cpri_mii_txrd remains asserted, the CPRI MII transmitter module inserts the end-of-packet sequence.
cpri_mii_txer	Input	Ethernet transmit coding error. When this signal is asserted, the CPRI IP core inserts an Ethernet HALT symbol in the data it passes to the CPRI link.

Table 6–7. CPRI MII Transmitter Interface Signals (Part 2 of 2)

Signal	Direction	Description
cpri_mii_txd[3:0]	Input	Ethernet transmit nibble data. The data transmitted from the external Ethernet block to the CPRI IP core, for transmission on the CPRI link. This input bus is synchronous to the rising edge of the cpri_clkout clock.
cpri_mii_txrd	Output	Ethernet read request. Indicates that the MII block is ready to read data on cpri_mii_txd[3:0]. Valid data is recognized 2 cpri_mii_txclk cycles after cpri_mii_txen is asserted in response to cpri_mii_txrd. The cpri_mii_txrd signal remains asserted for 2 cpri_mii_txclk cycles following deassertion of cpri_mii_txen. Deasserting cpri_mii_txrd while cpri_mii_txen is still asserted backpressures the external Ethernet block.

CPU Interface Signals

Table 6–8 lists the CPU interface signals. The CPU interface is implemented as an Avalon-MM interface.

Refer to the *Avalon Interface Specifications* for details about the Avalon-MM interface.

Table 6–8. CPU Interface Signals (Part 1 of 2)

Signal	Direction	Description
cpu_clk	Input	CPU clock signal.
cpu_reset	Input	CPU peripheral reset. This reset is associated with the cpu_clk clock. cpu_reset can be asserted asynchronously, but must stay asserted at least one cpu_clk cycle and must be de-asserted synchronously with cpu_clk. Refer to Figure 4–5 on page 4–10 for a circuit that shows how to enforce synchronous deassertion of a reset signal.
cpu_irq	Output	Merged CPU interrupt indicator. This signal is the OR of all the bits in the vector cpu_irq_vector.
cpu_irq_vector[4:0]	Output	This vector contains the following interrupt bits: [4] cpu_irq_cpri: Interrupt bit from CPRI_INTR register. This signal is the OR of all three interrupt bits in the CPRI_INTR register. [3] cpu_irq_eth_rx: Interrupt from the Ethernet receiver module. [2] cpu_irq_eth_tx: Interrupt from the Ethernet transmitter module. [1] cpu_irq_hdlc_rx: Interrupt from the HDLC receiver module. [0] cpu_irq_hdlc_tx: Interrupt from the HDLC transmitter module.
cpu_address[13:0]	Input	CPU word address. Corresponds to bits [15:2] of a byte address with LSBs 2'b00. If you connect an Avalon-MM interface to the CPU interface, connect bits [15:2] of the incoming Avalon-MM address to cpu_address.
cpu_write	Input	CPU write request.
cpu_read	Input	CPU read request.

Table 6–8. CPU Interface Signals (Part 2 of 2)

Signal	Direction	Description
cpu_byteenable[3:0]	Input	CPU data byteenable signal. Enables specific byte lanes during transfers on ports of width less than 32 bits. Each bit in the cpu_byteenable signal corresponds to a byte lane in cpu_writedata and cpu_readdata. The least significant bit of cpu_byteenable corresponds to the lowest byte of each data bus. The bit value 1 indicates an enabled byte lane, and the bit value 0 indicates a disabled byte lane. Enabled byte lanes must be adjacent: valid values of cpu_byteenable include only a single sequence of 1's. For more information, refer to the definition of the byteenable signal in the Avalon-MM specification in the *Avalon Interface Specifications*.
cpu_writedata[31:0]	Input	CPU write data.
cpu_readdata[31:0]	Output	CPU read data.
cpu_waitrequest	Output	Indicates that the CPU interface is busy executing an operation. When this signal is deasserted, the operation is complete and the data is valid.

Physical Layer Signals

Table 6–9 through Table 6–14 list the input and output signals of the physical layer of the CPRI IP core. Refer to Figure 4–25 on page 4–47 for details of the I/O signals.

CPRI Data Signals

Table 6–9 lists the CPRI data link signals.

Table 6–9. CPRI Protocol Interface

Signal	Direction	Description
gxb_rxdatain	Input	Receive unidirectional serial data. This signal is connected over the CPRI link to the txdataout line of the transmitting device.
gxb_txdataout	Output	Transmit unidirectional serial data. This signal is connected over the CPRI link to the rxdatain line of the receiving device.

Layer 1 Clock and Reset Signals

Table 6–10 lists the layer 1 clock and reset signals.

Table 6–10. CPRI Reference Clock and Main Reset Signals

Signal	Direction	Description
gxb_refclk	Input	Transceiver reference clock. In master clocking mode, this clock generates the internal clock cpri_clkout for the CPRI IP core and custom logic.
reset	Input	Transceiver reset. This reset is associated with the reconfig_clk clock. A reset controller module propagates this reset to the CPRI IP core cpri_clkout clock domain as well. reset can be asserted asynchronously, but must stay asserted at least one clock cycle and must be de-asserted synchronously with the clock with which it is associated. Refer to Figure 4–5 on page 4–10 for a circuit that shows how to enforce synchronous deassertion of reset.
reset_done	Output	Indicates that the reset controller has completed the transceiver reset sequence.

Layer 1 Error Signal

Table 6–11 lists the layer 1 error signal for the CPRI IP core.

Table 6–11. Layer 1 Error Signal

Signal	Direction	Description
gxb_los	Input	Loss of Signal (LOS) signal from small form-factor pluggable (SFP) module.

Autorate Negotiation Signals

Table 6–12 lists the autorate negotiation signals for the CPRI IP core. These output signals enable the autorate negotiation hardware and software outside the CPRI IP core to quickly monitor autorate negotiation status, and are implemented in all device families.

In Cyclone IV GX devices, channel reconfiguration is enabled to support autorate negotiation. Table 6–13 lists the signals implemented in CPRI IP cores targeted to Cyclone IV GX devices to support scan-chain based reconfiguration.

Table 6–12. Autorate Negotiation Signals

Signal	Direction	Description
datarate_en	Output	Indicates whether autorate negotiation is enabled. This signal reflects the value in the i_datarate_en field of the AUTO_RATE_CONFIG register described in Table 7–21 on page 7–10.
datarate_set[4:0]	Output	CPRI line rate to be used in next attempt to achieve frame synchronization. This signal reflects the value currently in the i_datarate_set field of the AUTO_RATE_CONFIG register described in Table 7–21 on page 7–10. The CPRI line rate is encoded in this field with the following values: 00001: 614.4 Mbps 00010: 1228.8 Mbps 00100: 2457.6 Mbps 00101: 3072.0 Mbps 01000: 4915.0 Mbps (not supported for Cyclone IV GX devices) 01010: 6144.0 Mbps (not supported for Cyclone IV GX devices) 10000: 9830.4 Mbps (supported only for Stratix V devices)

Table 6–13. Scan-Chain Based Reconfiguration Interface Signals For CPRI Autorate Negotiation in Cyclone IV GX Devices

Signal	Direction	Description
pll_areset	Input	Resets the PLL. Signal must be asserted after PLL reconfiguration. Connect to the areset signal for the PLL.
pll_configupdate	Input	When this signal is asserted, the PLL counters are updated with the contents of the scan chain. Signal is asserted for a single pll_scanclk cycle. Connect to the PLL reconfiguration scan chain configupdate signal.
pll_scanclk	Input	Clocks the shift registers in the PLL reconfiguration scan chain.The maximum frequency of this clock is 100 MHz.
pll_scanclkena	Input	Indicates scan data can be shifted in on the following pll_scanclk cycle. Connect to the PLL reconfiguration scan chain scanclkena signal.

Table 6–13. Scan-Chain Based Reconfiguration Interface Signals For CPRI Autorate Negotiation in Cyclone IV GX Devices

Signal	Direction	Description
pll_scandata	Input	Serial data scanned into the scan chain. Connect to the PLL reconfiguration scan chain scandata signal.
pll_reconfig_done	Output	Indicates PLL reconfiguration is complete.
pll_scandataout	Output	Output stream shifted out of the scan chain.

Transceiver Signals

Table 6–14 lists the transceiver signals that are connected directly to the transceiver block. In many cases these signals must be shared by multiple transceiver blocks that are implemented in the same device

Table 6–14. Transceiver Signals (Part 1 of 3)

Signal	Direction	Description
gxb_cal_blk_clk	Input	The Arria II GX, Arria II GZ, Cyclone IV GX, and Stratix IV GX transceivers' on-chip termination resistors are calibrated by a single calibration block. This circuitry requires a calibration clock. The frequency range of the gxb_cal_blk_clk is 10–125 MHz. For more information, refer to the *Transceiver Architecture for Arria II Devices* chapter in volume 2 of the *Arria II Device Handbook,* the *Cyclone IV Transceivers Architecture* chapter in volume 2 of the *Cyclone IV Device Handbook*, or the *Stratix IV Transceiver Architecture* chapter in volume 2 of the *Stratix IV Device Handbook*. This signal is not present in Arria V and Stratix V variations.
gxb_pll_inclk	Input	Input clock to the transceiver PLL. If the CPRI IP core is configured in master clocking mode, it does not use this clock. In master clocking mode, you must tie this input to 0. In slave clocking mode, the gxb_pll_inclk signal connects directly to the rx_cruclk input signal of the transceiver's PLL.
reconfig_clk (1)	Input	Reference clock for the dynamic reconfiguration controller. The frequency range for this clock is 37.5–50 MHz.
reconfig_togxb_s_tx [3:0]([139:0] for Arria V and Stratix V devices) (1)	Input	Driven from an external dynamic reconfiguration block to the slave transmitter transceiver block. Supports the selection of multiple transceiver channels for dynamic reconfiguration.
reconfig_togxb_s_rx [3:0]([69:0] for Arria V and Stratix V devices) (1)	Input	Driven from an external dynamic reconfiguration block to the slave receiver transceiver block. Supports the selection of multiple transceiver channels for dynamic reconfiguration.
reconfig_togxb_m[3:0] ([139:0] for Arria V and Stratix V devices) (1)	Input	Driven from an external dynamic reconfiguration block to the master transceiver block. Supports the selection of multiple transceiver channels for dynamic reconfiguration.
reconfig_fromgxb_s_tx [16:0] ([4:0] for Cyclone IV GX devices; [91:0] for Arria V and Stratix V devices)	Output	Driven to an external dynamic reconfiguration block from the slave transmitter transceiver block. The bus identifies the transceiver channel whose settings are being transmitted to the dynamic reconfiguration block.

Table 6–14. Transceiver Signals (Part 2 of 3)

Signal	Direction	Description
reconfig_fromgxb_s_rx [16:0] ([4:0] for Cyclone IV GX devices; [45:0] for Arria V and Stratix V devices)	Output	Driven to an external dynamic reconfiguration block from the slave receiver transceiver block. The bus identifies the transceiver channel whose settings are being transmitted to the dynamic reconfiguration block.
reconfig_fromgxb_m [16:0] ([4:0] for Cyclone IV GX devices; [91:0] for Arria V and Stratix V devices)	Output	Driven to an external dynamic reconfiguration block from the master transceiver block. The bus identifies the transceiver channel whose settings are being transmitted to the dynamic reconfiguration block.
reconfig_busy	Input	Indicates the busy status of the dynamic reconfiguration controller. After the device powers up, this signal remains low for the first reconfig_clk clock cycle. It is then asserted and remains high while the dynamic reconfiguration controller performs offset cancellation on all the receiver channels connected to the ALTGX_RECONFIG instance. This signal is deasserted when offset cancellation completes successfully. This signal is not present in Arria V and Stratix V variations.
reconfig_write	Input	Indicates the user is writing to the dynamic reconfiguration controller to implement the autorate negotiation feature. Asserting this signal instructs the CPRI reset controller to perform the reset sequence for dynamic reconfiguration of the transceiver. For details about dynamic reconfiguration, refer to the relevant device handbook.If you are not using the autorate configuration feature, you must tie this input to 0. This signal is not present in Arria V variations configured at the CPRI line rate of 9830.4 Mbps, which do not support the autorate negotiation feature.
reconfig_done	Input	Indicates the dynamic reconfiguration controller has completed the reconfiguration operation. Asserting this signal instructs the CPRI reset controller to complete the reset sequence for dynamic reconfiguration of the transceiver. For details about dynamic reconfiguration, refer to the relevant device handbook. If you are not using the autorate negotiation feature, you must tie this input to 0. This signal is not present in Arria V variations configured at the CPRI line rate of 9830.4 Mbps, which do not support the autorate negotiation feature.
gxb_pll_locked	Output	Indicates the transceiver transmitter PLL is locked to the input reference clock. This signal is asynchronous.
gxb_rx_pll_locked	Output	Indicates the transceiver CDR is locked to the input reference clock. This signal is asynchronous.
gxb_rx_freqlocked	Output	Transceiver clock data recovery (CDR) lock mode indicator. If this signal is high, the transceiver CDR is in lock-to-data (LTD) mode. If this signal is low, the transceiver CDR is in lock-to-reference clock (LTR) mode.

Table 6–14. Transceiver Signals (Part 3 of 3)

Signal	Direction	Description
gxb_powerdown	Input	Transceiver block power down. This signal resets and powers down all analog and digital circuitry in the transceiver block, including physical coding sublayer (PCS), physical media attachment (PMA), clock multiplier unit (CMU) channels, and central control unit (CCU). This signal does not affect the gxb_refclk buffers and reference clock lines. All the gxb_powerdown input signals of IP cores intended to be placed in the same quad must be tied together. The gxb_powerdown signal must be tied low or must remain asserted for at least 2 ms whenever it is asserted. This signal is not present in Arria V and Stratix V variations.
gxb_rx_disperr[1:0]	Output	Transceiver 8B/10B disparity error indicator. If either bit is high, a disparity error was detected on the associated received code group.
gxb_rx_errdetect[1:0]	Output	Transceiver 8B/10B code group violation or disparity error indicator. If either bit is high, a code group violation or disparity error was detected on the associated received code group. Use the gxb_rx_disperr signal to determine whether this signal indicates a code group violation or a disparity error. For details, refer to the relevant device handbook.

Note to Table 6–14:

(1) Refer to "Instantiating Multiple CPRI IP Cores" on page 2–6 for information about how to successfully combine multiple high-speed transceiver channels—whether in two CPRI IP core instances or in a CPRI IP core and in another component—in the same quad.

In addition to customization of the transceiver through the transceiver parameter editor, you can use the transceiver reconfiguration block to dynamically modify the parameter interface. The dynamic reconfiguration block lets you reconfigure the following PMA settings:

- Pre-emphasis
- Equalization
- Offset cancellation
- V_{OD} on a per channel basis

You must configure the dynamic reconfiguration block in any CPRI design that targets an Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX device.

For more information about the transceiver reconfiguration block and about offset cancellation, refer to the appropriate device handbook.

Clock and Reset Interface Signals

Table 6–15 describes the CPRI IP core clock and reset signals not described in other sections with their associated modules.

Table 6–15. CPRI IP Core Clock and Reset Signals

Signal	Direction	Description
clk_ex_delay	Input	Extended delay measurement clock. This clock must be driven from a common source with the transceiver reference clock.
reset_ex_delay	Input	Reset for extended delay measurement block.This reset is associated with the clk_ex_delay clock. reset_ex_delay can be asserted asynchronously, but must stay asserted at least one clock cycle and must be de-asserted synchronously with the clock with which it is associated. Refer to Figure 4–5 on page 4–10 for a circuit that shows how to enforce synchronous deassertion of a reset signal.
config_reset	Input	Register reset. This reset is associated with the cpri_clkout clock. config_reset can be asserted asynchronously, but must stay asserted at least one clock cycle and must be de-asserted synchronously with the clock with which it is associated. Refer to Figure 4–5 on page 4–10 for a circuit that shows how to enforce synchronous deassertion of a reset signal.
pll_clkout	Output	Generated from transceiver clock data recovery circuit. Intended to connect to an external PLL for jitter clean-up.
cpri_clkout	Output	CPRI core clock. Provided for observation and debugging.
hw_reset_req	Output	Hardware reset request detected from received reset control word. This signal is set after the received reset control word is set in ten consecutive basic frames, if the reset_out_en bit of the CPRI_HW_RESET register is set. This signal is cleared in reset. It can be used to inform the application layer of the low-level reset request.
hw_reset_assert	Input	Indicates a reset request should be sent to the CPRI link partner on the CPRI link, using bit 0 of the CPRI hyperframe control word Z.130.0. If the reset_hw_en bit of the CPRI_HW_RESET register is set, the CPRI IP core sends the reset request on the CPRI link. The hw_reset_assert signal is detected on the rising edge of cpri_clkout.
usr_pma_clk	Input	One of two extra clock signals required for CPRI IP core variations configured at 9830.4 Mbps that target an Arria V GT device. When configured at this CPRI line rate, a CPRI IP core that targets an Arria V GT device does not support autorate negotiation. The CPRI IP core requires that usr_pma_clk be driven at 122.88 MHz from a common source with, and synchronized with, the driver of usr_clk. In master clocking mode, it must have a common source with the gxb_refclk signal, and in slave clocking mode, it must be driven from the cleanup PLL.
usr_clk	Input	One of two extra clock signals required for CPRI IP core variations configured at 9830.4 Mbps that target an Arria V GT device. When configured at this CPRI line rate, a CPRI IP core that targets an Arria V GT device does not support autorate negotiation. The CPRI IP core requires that usr_clk be driven at 245.76 MHz from a common source with, and synchronized with, the driver of usr_pma_clk. it must have a common source with the gxb_refclk signal, and in slave clocking mode, it must be driven from the cleanup PLL.

7. Software Interface

The Altera CPRI IP core supports the following sets of registers that control the CPRI IP core or query its status:

- CPRI Protocol Interface Registers
- MAP Interface and AUX Interface Configuration Registers
- Ethernet Registers
- HDLC Registers

All of the registers are 32 bits wide and their addresses are shown as hexadecimal values. The registers can be accessed only on a 32-bit (4-byte) basis. The addressing for the registers therefore increments by units of 4.

Reserved fields are labelled in the register tables. These fields are reserved for future use and your design should not write to or rely on a specific value being found in any reserved field or bit.

A remote device can access these registers only by issuing read and write operations through the CPU interface.

Table 7–1 lists the access codes that describe the type of register bits.

Table 7–1. Register Access Codes

Code	Description
RC	Read to clear
RO	Read-only
RW	Read/write
UR0	Unused bits/read as 0
WO	Write-only; read as 0

Table 7–2 lists the CPRI IP core register address ranges.

Table 7–2. CPRI IP Core Register Address Ranges

Address Range	Interface
0x00–0x68	CPRI Protocol Interface Registers
0x100–0x1A4	MAP Interface and AUX Interface Configuration Registers
0xF4–0x1FC	Reserved
0x200–0x24C	Ethernet Registers
0x250–0x2FC	Reserved
0x300–0x334	HDLC Registers

CPRI Protocol Interface Registers

This section lists the CPRI protocol interface registers. Table 7–3 provides a memory map for the CPRI protocol interface registers. Table 7–4 through Table 7–29 describe the CPRI protocol interface registers in the CPRI IP core.

Table 7–3. CPRI Protocol Interface Registers Memory Map

Address	Name	Expanded Name
0x0	CPRI_INTR	Interrupt Control and Status
0x4	CPRI_STATUS	CPRI Status
0x8	CPRI_CONFIG	CPRI Configuration
0xC	CPRI_CTRL_INDEX	CPRI Control Word Index
0x10	CPRI_RX_CTRL	CPRI Received Control Word
0x14	CPRI_TX_CTRL	CPRI Transmit Control Word
0x18	CPRI_LCV	CPRI Line Code Violation Counter
0x1C	CPRI_RX_BFN	CPRI Recovered Radio Frame Counter
0x20	CPRI_HW_RESET	Hardware Reset From Control Word
0x24	CPRI_PHY_LOOP	Physical Layer Loopback Control
0x28	CPRI_CM_CONFIG	CPRI Control and Management Configuration
0x2C	CPRI_CM_STATUS	CPRI Control and Management Status
0x30	CPRI_RX_DELAY_CONTROL	Receiver Delay Control
0x34	CPRI_RX_DELAY	Receiver Delay
0x38	CPRI_ROUND_DELAY	Round Trip Delay
0x3C	CPRI_EX_DELAY_CONFIG	Extended Delay Measurement Configuration
0x40	CPRI_EX_DELAY_STATUS	Extended Delay Measurement Status
0x44	Reserved	
0x48	AUTO_RATE_CONFIG	Autorate Negotiation
0x4C	CPRI_INTR_PEND	Pending Interrupt Status
0x50	CPRI_N_LCV	LCV Threshold
0x54	CPRI_T_LCV	LCV Test Period
0x58	CPRI_TX_PROT_VER	Tx Protocol Version
0x5C	CPRI_TX_SCR_SEED	Tx Scrambler Seed
0x60	CPRI_RX_SCR_SEED	Rx Scrambler Support
0x64	CPRI_TX_BITSLIP	Tx Bitslip
0x68	CPRI_AUTO_CAL	Autocalibration

Table 7–4. CPRI_INTR—Interrupt Control and Status—Offset: 0x0 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:6]	UR0	Reserved.	31'h0
intr_los_lcv_en	[5]	RW	los_lcv interrupt enable.	1'h0
RSRV	[4:2]	UR0	Reserved.	3'h0

Table 7–4. CPRI_INTR—Interrupt Control and Status—Offset: 0x0 (Part 2 of 2)

Field	Bits	Access	Function	Default
intr_hw_reset_en	[1]	RW	hw_reset interrupt enable. Controls whether a reset request received over the CPRI link raises an interrupt on the CPU IRQ line.	1'h0
intr_en	[0]	RW	CPRI protocol interface module interrupt enable. The Ethernet and HDLC modules have separate interrupt enable control bits.	1'h0

Table 7–5. CPRI_STATUS—CPRI Status—Offset: 0x4

Field	Bits	Access	Function	Default
RSRV	[31:12]	UR0	Reserved.	20'h0
rx_rfp_hold	[11]	RC	Radio frame pulse received. This bit is asserted every 10 ms. (1)	1'h0
rx_freq_alarm_hold	[10]	RC	CPRI receive clock is not synchronous with system clock (cpri_clkout). This alarm is asserted each time mismatches are found between the recovered CPRI receive clock and the system clock cpri_clkout. (1)	1'h0
rx_state_hold	[9]	RC	Hold rx_state. (1)	1'h0
rx_los_hold	[8]	RC	Hold rx_los. (1)	1'h0
RSRV	[7:6]	UR0	Reserved.	2'h0
los_lcv	[5]	RO	Loss of signal (LOS) detected. This alarm is asserted if excessive line code violations (LCVs) are detected, based on two counters and two programmable threshold values. The first counter counts up to the expected amount of time to CPRI link synchronization, during which the second counter does not count LCVs. The second counter counts LCVs up to the threshold—the number of LCVs after which this alarm is asserted. The CPRI_T_LCV register at offset 0x54 specifies the expected amount of time to CPRI link synchronization, and the CPRI_N_LCV register at offset 0x50 holds the threshold number of LCVs after which this alarm is asserted.	1'h0
RSRV	[4]	UR0	Reserved.	1'h0
rx_bfn_state	[3]	RO	Indicates BFN (Node B radio frame) synchronization has been achieved.	1'h0
rx_hfn_state	[2]	RO	Indicates HFN synchronization has been achieved.	1'h0
rx_state	[1]	RO	When set, indicates that Rx HFN and BFN synchronization have been achieved in CPRI receiver frame synchronization. You can read this field to determine whether the Rx link is established.	1'h0
rx_los	[0]	RO	Indicates either excessive 8B/10B violations (> 15) or incoming LOS signal on dedicated line from SFP optical module (gxb_los signal).	1'h0

Note to Table 7–5:

(1) This register field is a read-to-clear field. You must read the register twice to read the true value of the field after frame synchronization is achieved. If you observe this bit asserted during link initialization, read the register again after link initialization to confirm any errors.

Table 7–6. CPRI_CONFIG—CPRI Configuration—Offset: 0x8 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:6]	UR0	Reserved.	26'h0
tx_enable	[5]	RW	Enable transmission on CPRI link.	1'h0

Table 7–6. CPRI_CONFIG—CPRI Configuration—Offset: 0x8 (Part 2 of 2)

Field	Bits	Access	Function	Default
loop_mode	[4:2]	RW	Testing loopback mode. The reverse loopback paths specified in this register field include the transmission framing block, in contrast to the lower-level loopback path specified in the CPRI_PHY_LOOP register at offset 0x24. The loopback paths specified in this register field are only enabled after frame synchronization, and can only be activated in a CPRI RE slave. The following field values are defined: 000: No loopback. 001: Full CPRI frame loop. Incoming CPRI data and control words are sent back in outgoing CPRI communication. 010: IQ sample loop. Incoming CPRI data are sent back in outgoing CPRI communication; control words are generated locally. 011: Fast C&M loop. Incoming CPRI C&M control and data words are sent back in outgoing CPRI communication; remaining data and control words are generated locally. 100: Fast C&M and VSS loop. Incoming CPRI C&M and vendor-specific control words are sent back in outgoing CPRI communication; data and remaining control words are generated locally. Note that this loopback mode is superseded by the 1-bit physical layer loop mode specified in the CPRI_PHY_LOOP register at offset 0x24. If both register fields hold non-zero values, the value in the CPRI_PHY_LOOP register takes precedence.	3'h0
RSRV	[1]	RO	Reserved.	1'h0
tx_ctrl_insert_en	[0]	RW	Master enable for insertion of tx_control_data contents in CPRI control word. This signal enables control bytes for which the tx_control_insert bit is high to be written to the CPRI frame.	1'h0

Table 7–7. CPRI_CTRL_INDEX—CPRI Control Word Index—Offset: 0xC

Field	Bits	Access	Function	Default
RSRV	[31:8]	UR0	Reserved.	24'h0
cpri_ctrl_index	[7:0]	RW	Index for CPRI control byte monitoring and insertion. The value in this field determines the control receive and control transmit table entries that appear in the CPRI_RX_CTRL and CPRI_TX_CTRL registers.	8'h0

Table 7–8. CPRI_RX_CTRL—CPRI Received Control Word—Offset: 0x10

Field	Bits	Access	Function	Default
RSRV	[31:8]	UR0	Reserved.	24'h0
rx_control_data	[7:0]	RW	Most recent received CPRI control word from CPRI hyperframe position Z.x.0, where x is the index in the cpri_ctrl_index field of the CPRI_CTRL_INDEX register.	8'h0

Table 7–9. CPRI_TX_CTRL—CPRI Transmit Control Word—Offset: 0x14

Field	Bits	Access	Function	Default
RSRV	[31:9]	UR0	Reserved.	23'h0
tx_control_insert	[8]	RW	Control byte transmit enable.	1'h0
tx_control_data	[7:0]	RW	CPRI control byte to be transmitted in CPRI hyperframe position Z.x.0, where x is the index in the cpri_ctrl_index field of the CPRI_CTRL_INDEX register.	8'h0

Table 7–10. CPRI_LCV—CPRI Line Code Violation Counter—Offset: 0x18

Field	Bits	Access	Function	Default
RSRV	[31:8]	UR0	Reserved.	24'h0
cpri_lcv	[7:0]	RO	Number of line code violations (LCVs) detected in the 8B/10B decoding block in the transceiver. Enables CPRI link debugging. This register saturates at the value 255; after it reaches 255, it maintains this value until reset. This counter is not used to determine whether the N_LCV threshold (Table 7–23 on page 7–11) is reached, because it includes LCVs that occur during initialization—before T_LCV (Table 7–24 on page 7–11) is reached—and because it saturates.	8'h0

Table 7–11. CPRI_BFN—CPRI Recovered Radio Frame Counter—Offset: 0x1C

Field	Bits	Access	Function	Default
RSRV	[31:12]	UR0	Reserved.	20'h0
bfn	[11:0]	RO	Current BFN (node B radio frame number) number. Value obtained from BFN alignment state machine.	12'h0

Table 7–12. CPRI_HW_RESET—Hardware Reset From Control Word—Offset: 0x20 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:8]	UR0	Reserved.	24'h0
reset_gen_done_hold	[7]	RC	Hold reset_done.	1'h0
reset_gen_done	[6]	RO	Indicates that a reset request or acknowledgement has been successfully sent on the CPRI link by the CPRI transmitter.	1'h0
reset_detect_hold	[5]	RC (1)	Hold reset_detect.	1'h0
reset_detect	[4]	RO	Indicates that reset request has been detected in the incoming stream on the CPRI link by the CPRI receiver.	1'h0

Table 7–12. CPRI_HW_RESET—Hardware Reset From Control Word—Offset: 0x20 (Part 2 of 2)

Field	Bits	Access	Function	Default
reset_hw_en	[3]	RW	Enable generation of reset request or acknowledge by CPRI transmitter, as indicated by the hw_reset_assert input signal. This enable bit has higher priority than the reset_gen_en bit; if this enable bit is set, the reset_gen_force bit is ignored. Note that when a CPRI RE slave detects a reset request in incoming CPRI communication, and the reset_hw_en bit is set, the user must assert the hw_reset_assert input signal to the CPRI RE slave, to force it to send a reset acknowledge by setting the reset bit in outgoing CPRI communication at Z.130.0.	1'h0
reset_out_en	[2]	RW	Enable reset output.	1'h0
reset_gen_force	[1]	RW	Force generation of reset request or acknowledge by CPRI transmitter.	1'h0
reset_gen_en	[0]	RW	Enable generation of reset request or acknowledge by CPRI transmitter, as indicated by the reset_gen_force bit. This enable bit has lower priority than the reset_hw_en bit; if the reset_hw_en bit is set, this bit and the reset_gen_force bit are ignored.	1'h0

Note to Table 7–12:

(1) This register field is a read-to-clear field. You must read the register twice to read the true value of the field after frame synchronization is achieved. If you observe this bit asserted during link initialization, read the register again after link initialization to confirm any errors.

For additional information about the CPRI_HW_RESET register, refer to "Reset Requirements" on page 4–9.

Table 7–13. CPRI_PHY_LOOP—Physical Layer Loopback Control—Offset: 0x24 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:5]	UR0	Reserved.	27'h0
loop_resync	[4]	RC (1)	Indicates that reset resynchronization is detected. This bit is typically set when the CPRI receiver clock and cpri_clkout have different frequencies, as measured in the physical layer internal loopback path.	1'h0
RSRV	[3:1]	UR0	Reserved.	2'h0

Table 7–13. CPRI_PHY_LOOP—Physical Layer Loopback Control—Offset: 0x24 (Part 2 of 2)

Field	Bits	Access	Function	Default
loop_mode	[0]	RW	Physical layer loopback mode. The following values are defined: 0: No loopback. 1: Full CPRI frame loop. Incoming CPRI data and control words are sent back as-is in outgoing CPRI communication. This low-level reverse loopback path is active whether or not frame synchronization has been achieved; the path includes 8B/10B encoding and decoding, but only enough core CPRI functionality to handle the transition from the receiver clock domain to the transmitter clock domain. This loopback mode takes precedence over the 3-bit loop_mode specified in the CPRI_CONFIG register at offset 0x8: if this field has value 1, the 3-bit loop_mode value is ignored.	2'h0

Note to Table 7–13:

(1) This register field is a read-to-clear field. You must read the register twice to read the true value of the field after frame synchronization is achieved. If you observe this bit asserted during link initialization, read the register again after link initialization to confirm any errors.

Table 7–14. CPRI_CM_CONFIG—CPRI Control and Management Configuration—Offset: 0x28

Field	Bits	Access	Function	Default
RSRV	[31:11]	UR0	Reserved.	20'h0
tx_slow_cm_rate	[10:8]	RW	Rate configuration for slow C&M (HDLC). To be inserted in CPRI control byte Z.66.0.	3'h0
RSRV	[7:6]	UR0	Reserved.	2'h0
tx_fast_cm_ptr	[5:0]	RW	Pointer to first CPRI control word used for fast C&M (Ethernet). To be inserted in CPRI control byte Z.194.0.	8'h14

Table 7–15. CPRI_CM_STATUS—CPRI Control and Management Status—Offset: 0x2C (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:12]	UR0	Reserved.	20'h0
rx_slow_cm_rate_valid	[11]	RO	Indicates that a valid slow C&M rate has been accepted.	1'h0

Table 7–15. CPRI_CM_STATUS—CPRI Control and Management Status—Offset: 0x2C (Part 2 of 2)

Field	Bits	Access	Function	Default
rx_slow_cm_rate	[10:8]	RO	Accepted receive slow C&M rate, as determined during the software set-up sequence, or by dynamic modification, in which the same new pointer value is received in incoming CPRI control byte Z.66.0 four hyperframes in a row. The following values are defined: 000: No HDLC channel. 001: 240 Kbps 010: 480 Kbps 011: 960 Kbps 100: 1920 Kbps 101: 2400 Kbps 110: 3840, 4800, or 7680 Kbps, depending on the current CPRI line rate, as specified in Table 4–13 on page 4–45. For information about compatible slow C&M rates and CPRI line rates, refer to Table 4–13 on page 4–45.	3'h0
RSRV	[7]	UR0	Reserved.	1'h0
rx_fast_cm_ptr_valid	[6]	RO	Indicates that a valid fast C&M pointer has been accepted.	1'h0
rx_fast_cm_ptr	[5:0]	RO	Accepted receive fast C&M pointer, as determined during the software set-up sequence or by dynamic modification, in which the same new pointer value is received in incoming CPRI control byte Z.194.0 four hyperframes in a row. The value is between 0x24 and 0x3F, inclusive.	6'h0

Table 7–16. CPRI_RX_DELAY_CTRL—Receiver Delay Control—Offset: 0x30

Field	Bits	Access	Function	Default
RSRV	[31:17]	UR0	Reserved.	15'h0
rx_buf_resync	[16]	RW	Force CPRI receiver buffer (Rx elastic buffer) realignment. Altera recommends that you resynchronize the Rx elastic buffer after a dynamic CPRI line rate change. Resynchronizing might lead to data loss or corruption.	1'h0
RSRV	[15:WIDTH_RX_BUF] (1)	UR0	Reserved.	0
rx_buf_int_delay	[(WIDTH_RX_BUF-1):0] (1)	RW	Initial buffer delay with which to align the Rx elastic buffer. After you modify the value of this field, you must set the rx_buf_resync bit to resynchronize the buffer.	$2^{WIDTH\_RX\_BUF-1}$

Note to Table 7–16:

(1) WIDTH_RX_BUF is the value specified for the **Receiver buffer depth** parameter. This value is $\log_2$ of the depth of the Rx elastic buffer. By default, it is set to six, specifying a 64-entry buffer. Altera recommends that you set it to four, specifying a 16-entry buffer, in slave configurations.

Table 7–17. CPRI_RX_DELAY—Receiver Delay—Offset: 0x34

Field	Bits	Access	Function	Default
RSRV	[31:(WIDTH_RX_BUF+2)] (1)	UR0	Reserved.	0
rx_buf_delay	[(WIDTH_RX_BUF+1):2] (1)	RO	Current receive buffer fill level. Unit is 32-bit words. Maximum value is $2^{WIDTH\_RX\_BUF}$-1.	0
rx_byte_delay	[1:0]	RO	Current byte-alignment delay. This field was relevant for the Rx path delay calculation in previous releases of the CPRI IP core, but is not relevant for the Rx path delay calculation in the current release of the CPRI IP core. Refer to "Rx Path Delay Components" on page D–3.	2'h0

Note to Table 7–17:

(1) WIDTH_RX_BUF is the value specified for the **Receiver buffer depth** parameter. This value is $\log_2$ of the depth of the Rx elastic buffer. By default, it is set to six, specifying a 64-entry buffer. Altera recommends that you set it to four, specifying a 16-entry buffer, in slave configurations.

Table 7–18. CPRI_ROUND_DELAY—Round Trip Delay—Offset: 0x38

Field	Bits	Access	Function	Default
RSRV	[31:20]	UR0	Reserved.	12'h0
rx_round_trip_delay	[19:0]	RO	Measured round trip delay from cpri_tx_rfp to cpri_rx_rfp. Unit is cpri_clkout clock periods.	20'h0

Table 7–19. CPRI_EX_DELAY_CONFIG—Extended Delay Measurement Configuration—Offset: 0x3C

Field	Bits	Access	Function	Default
RSRV	[31:25]	UR0	Reserved.	7'h0
tx_ex_delay	[24:16]	RW	Integration period for Tx buffer extended delay measurement. Program this field with the user-defined value N, where M/N = clk_ex_delay period / cpri_clkout period. Refer to "CPRI Receive Buffer Delay Calculation Example" on page D–6.	9'h0
RSRV	[15:9]	UR0	Reserved.	7'h0
rx_ex_delay	[8:0]	RW	Integration period for Rx buffer extended delay measurement. Program this field with the user-defined value N, where M/N = clk_ex_delay period / cpri_clkout period. Refer to "CPRI Receive Buffer Delay Calculation Example" on page D–6.	9'h0

Table 7–20. CPRI_EX_DELAY_STATUS—Extended Delay Measurement Status—Offset: 0x40 (Part 1 of 2)

Field	Bits	Access	Function	Default
tx_ex_buf_delay_valid	[31]	RC	Indicates that the tx_ex_buf_delay field has been updated.	1'h0
tx_ex_buf_delay	[30:18]	RO	Tx buffer extended delay measurement result. Unit is cpri_clkout clock periods. Refer to "Extended Tx Delay Measurement" on page D–11.	0
RSRV	[17]	UR0	Reserved.	1'h0
rx_ex_buf_delay_valid	[16]	RC	Indicates that the rx_ex_buf_delay field has been updated.	1'h0

Table 7–20. CPRI_EX_DELAY_STATUS—Extended Delay Measurement Status—Offset: 0x40 (Part 2 of 2)

Field	Bits	Access	Function	Default
RSRV	[15:(WIDTH_RX_BUF+9)] (1)	UR0	Reserved.	0
rx_ex_buf_delay	[(WIDTH_RX_BUF+8):0] (1)	RO	Rx buffer extended delay measurement result. Unit is cpri_clkout clock periods. Refer to "Extended Rx Delay Measurement" on page D–5.	0

Note to Table 7–20:

(1) WIDTH_RX_BUF is the value specified for the **Receiver buffer depth** parameter. This value is $\log_2$ of the depth of the Rx elastic buffer. By default, it is set to six, specifying a 64-entry buffer. Altera recommends that you set it to four, specifying a 16-entry buffer, in slave configurations.

Table 7–21. AUTO_RATE_CONFIG—Autorate Negotiation Register—Offset: 0x48

Field	Bits	Access	Function	Default
RSRV	[31:6]	UR0	Reserved.	28'h0
i_datarate_en	[5]	RO	Indicates that autorate negotiation is enabled. (Value is 1'b0 if autorate negotiation is not enabled; 1'b1 if autorate negotiation is enabled, in the CPRI parameter editor). Refer to Figure B–1 and Figure B–2 for an illustration of the autorate negotiation logic in the CPRI IP core and the autorate negotiation logic you must add to your design outside the CPRI IP core.	As specified in CPRI parameter editor
i_datarate_set	[4:0]	RW	CPRI line rate to be used in next attempt to achieve frame synchronization. You set the line rate in your implementation of the autorate negotiation hardware and software outside the CPRI IP core. Refer to Appendix B, Implementing CPRI Link Autorate Negotiation, for information about how to use the autorate negotiation logic implemented in the CPRI IP core. Encode the CPRI line rate in this field with the following values: 00001: 614.4 Mbps 00010: 1228.8 Mbps 00100: 2457.6 Mbps 00101: 3072.0 Mbps 01000: 4915.0 Mbps (1) 01010: 6144.0 Mbps (1) 10000: 9830.4 Mbps (2)	4'h0

Notes to Table 7–21:

(1) This value is not valid for CPRI IP core variations that target a Cyclone IV GX device. This value is valid for CPRI MegaCore variations that target an Arria II GX device only if that device is an I3 speed grade device.

(2) This value is valid only for CPRI IP core variations that target a Stratix V device.

Table 7–22. CPRI_INTR_PEND—Interrupt Pending Status—Offset: 0x4C

Field	Bits	Access	Function	Default
RSRV	[31:6]	UR0	Reserved.	26'h0
los_lcv_pending	[5]	RW	Indicates an los_lcv interrupt is pending (the interrupt occurred but is not yet serviced).	1'h0
RSRV	[4:2]	UR0	Reserved.	4'h0
hw_reset_pending	[1]	RW	Indicates a hw_reset interrupt is pending (the interrupt occurred but is not yet serviced). In an RE slave, this bit is set when a reset request is detected in incoming CPRI communication at Z.130.0, but neither the reset_gen_en bit nor the reset_hw_en bit in the CPRI_HW_RESET register is set (so that a reset acknowledge cannot be sent to the RE master), or when the CPRI RE slave sends a reset acknowledge on the outgoing CPRI link at Z.130.0. In a master, this bit is set when a reset acknowledge is received on the incoming CPRI link at Z.130.0. Software can count assertions of this bit to confirm the reset bit in Z.130.0 was asserted in ten consecutive hyperframes to complete a CPRI-compliant reset acknowledge. Note that when a reset request is detected in incoming CPRI communication, and the reset_hw_en bit in the CPRI_HW_RESET register is set, the user must assert the hw_reset_assert input signal to the CPRI RE slave, to force it to send a reset acknowledge by setting the reset bit in outgoing CPRI communication at Z.130.0. After the reset bit is sent on the CPRI link, hw_reset_pending is asserted.	1'h1
RSRV	[0]	UR0	Reserved.	1'h0

Table 7–23. CPRI_N_LCV—LCV Threshold—Offset: 0x50

Field	Bits	Access	Function	Default
N_LCV	[31:0]	RW	The number of LCVs that triggers the assertion of the cpri_rx_los signal.	32'h0

Table 7–24. CPRI_T_LCV—LCV Test Period—Offset: 0x54

Field	Bits	Access	Function	Default
T_LCV	[31:0]	RW	The number of bytes in the initialization period during which we do not yet count LCVs toward assertion of the cpri_rx_los signal.	32d'614400

Table 7–25. CPRI_TX_PROT_VER— Tx Protocol Version —Offset: 0x58

Field	Bits	Access	Function	Default
RSRV	[31:8]	UR0	Reserved.	24'h0
tx_prot_version	[7:0]	RW	Transmit protocol version to be mapped to Z.2.0 to indicate whether or not the current hyperframe transmission is scrambled. The value 1 indicates it is not scrambled and the value 2 indicates it is scrambled.	8'h01

Table 7–26. CPRI_TX_SCR_SEED— Tx Scrambler Seed —Offset: 0x5C

Field	Bits	Access	Function	Default
RSRV	[31]	UR0	Reserved.	1'h0
tx_scr_seed	[30:0]	RW	Transmitter scrambler seed. If the seed has value 0, the transmission is not scrambled.	31'h0

Table 7–27. CPRI_RX_SCR_SEED— Rx Scrambler Support —Offset: 0x60

Field	Bits	Access	Function	Default
rx_scr_act_indication	[31]	RO	Indicates that the incoming hyperframe is scrambled. The value 1 indicates that the incoming communication is scrambled, and the value 0 indicates that it is not scrambled.	1'h0
rx_scr_seed	[30:0]	RO	Received scrambler seed. The receiver descrambles the incoming CPRI communication based on this seed.	31'h0

Table 7–28. CPRI_TX_BITSLIP— Tx Bitslip —Offset: 0x64 (1), (2), (3) (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:21]	UR0	Reserved.	11'h0
rx_bitslipboundaryselectout	[20:16]	RO	Number of bits of delay (bitslip) detected at the receiver word-aligner. Value can change at frame synchronization, when the transceiver is resetting. Any K28.5 symbol position change that occurs when word alignment is activated changes the bitslip value.	5'h0
RSRV	[15:9]	UR0	Reserved.	7'h0
tx_bitslip_en	[8]	RW	Enable manual tx_bitslipboundaryselect updates. When this bit has the value of 0 in a CPRI RE slave, the CPRI RE slave determines the value in the tx_bitslipboundaryselect field, and adds tx_bitslipboundaryselect bits of delay in the transceiver transmitter to compensate for the variability in the Rx word aligner bitslip. The CPRI IP core ignores the value in the tx_bitslipboundaryselect field in a CPRI REC or RE master. When the tx_bitslip_en bit has the value of 1, the application can write a value to the tx_bitslipboundaryselect field to manually override the value the CPRI IP core would calculate.	1'h0
RSRV	[7:5]	UR0	Reserved.	3'h0

Table 7–28. CPRI_TX_BITSLIP— Tx Bitslip —Offset: 0x64 (1), (2), (3) (Part 2 of 2)

Field	Bits	Access	Function	Default
tx_bitslipboundaryselect	[4:0]	RW	Number of bits of delay (bitslip) the CPRI IP core adds at the CPRI Tx link to compensate for the variability in the Rx word aligner bitslip. The purpose of this added delay is to ensure the variability in the round-trip delay through this CPRI RE slave remains compliant with the R-20 and R-21 deterministic latency requirements of the CPRI specification V4.2. The device family and CPRI line rate determine the following maximum values for this field: ■ Maximum value for all CPRI variations with line rate 614.4 Mbps and for all variations that target an Arria II GX or Cyclone IV GX device: 9 bits. ■ Maximum value for all other variations: 19 bits. The latency differences from different Tx bitslip delay values are observable only with an oscilloscope.	5'h0

Notes to Table 7–28:

(1) In variations that target an Arria V or Stratix V device, the Tx bitslip functionality is included in the Altera Transceiver PHY IP core that is generated as part of the CPRI variation.

(2) CPRI variations with master clocking mode (CPRI REC and RE masters) do not support the automatic bitslip calibration functionality controlled by this register.

(3) For information about the CPRI IP core Tx bitslip feature, refer to "Tx Bitslip Delay" on page D–12.

Table 7–29. CPRI_AUTO_CAL— Autocalibration (1), (2) —Offset: 0x68

Field	Bits	Access	Function	Default
RSRV	[31:30]	UR0	Reserved.	2'h0
cal_pointer	[29:26]	RO	Number of autocalibration pipeline stages currently in use. Each such stage adds one cpri_clkout cycle of delay in the Rx path.	4'h3
cal_status	[25:24]	RO	Calibration status. Valid values are: 00: Calibration is turned off 01: Calibration is running or falied with cal_rtd value too low 10: Calibration is running or failed with cal_rtd value too high 11: Calibration is successful	2'h0
RSRV	[23:21]	UR0	Reserved.	3'h0
cal_en	[20]	RW	Indicates that calibration mode is enabled. When the value in this field is 1, autocalibration is turned on. When the value in this field is 0, autocalibration is turned off.	1'h0
cal_rtd	[19:0]	RW	Desired round-trip delay value. Unit is cpri_clkout cycles.	20'h0

Notes to Table 7–29:

(1) CPRI variations with slave clocking mode (CPRI RE slaves) do not support the functionality controlled by this register.

(2) For information about the CPRI IP core autocalibration feature, refer to "Dynamic Pipelining for Automatic Round-Trip Delay Calibration" on page D–21.

MAP Interface and AUX Interface Configuration Registers

This section lists the MAP interface configuration registers. Table 7–30 provides a memory map for the MAP interface configuration registers. Table 7–31 through Table 7–49 describe the MAP interface configuration registers in the CPRI IP core.

Table 7–30. MAP Interface Configuration Registers Memory Map

Address	Name	Expanded Name
0x100	CPRI_MAP_CONFIG	CPRI Mapping Features Configuration
0x104	CPRI_MAP_CNT_CONFIG	Basic UMTS/LTE Mapping Configuration
0x108	CPRI_MAP_TBL_CONFIG	K Parameter Config for Advanced Table-Based Mapping
0x10C	CPRI_MAP_TBL_INDEX	Advanced Mapping Configuration Table Index
0x110	CPRI_MAP_TBL_RX	Advanced Mapping Rx Configuration Table
0x114	CPRI_MAP_TBL_TX	Advanced Mapping Tx Configuration Table
0x118	CPRI_MAP_OFFSET_RX	MAP Rx Frame Offset
0x11C	CPRI_MAP_OFFSET_TX	MAP Tx Frame Offset
0x120	CPRI_START_OFFSET_RX	Rx Start Frame Offset
0x124	CPRI_START_OFFSET_TX	Tx Start Frame Offset
0x128	CPRI_MAP_RX_READY_THR	CPRI Mapping Rx Ready Threshold
0x12C	CPRI_MAP_TX_READY_THR	CPRI Mapping Tx Ready Threshold
0x130	CPRI_MAP_TX_START_THR	CPRI Mapping Tx Start Threshold
0x13C	CPRI_PRBS_CONFIG	PRBS Generation Pattern Configuration
0x140–0x144	CPRI_PRBS_STATUS	PRBS Data Validation Status
0x150	CPRI_IQ_RX_BUF_CONTROL	MAP Receiver FIFO Buffer Control
0x160	CPRI_IQ_TX_BUF_CONTROL	MAP Transmitter FIFO Buffer Control
0x180–0x184	CPRI_IQ_RX_BUF_STATUS	MAP Receiver FIFO Buffer Status
0x1A0–0x1A4	CPRI_IQ_TX_BUF_STATUS	MAP Transmitter FIFO Buffer Status

Table 7–31. CPRI_MAP_CONFIG—CPRI Mapping Features Configuration—Offset: 0x100 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:5]	UR0	Reserved.	27'h0
map_15bit_mode	[4]	RW	15-bit sample width. Values are: 0: 2 × 16-bit sample width 1: 2× 15-bit sample width The Altera CPRI IP core does not support the map_15bit_mode value of 0 in the Advanced 3 mapping mode. For more information, refer to Appendix C, Advanced AxC Mapping Modes.	1'h0
map_tx_sync_mode	[3]	RW	Tx MAP synchronization mode if **Enable MAP interface synchronization with core clock** is turned off. Values are: 0: FIFO mode 1: Synchronous buffer mode	1'h0

Table 7–31. CPRI_MAP_CONFIG—CPRI Mapping Features Configuration—Offset: 0x100 (Part 2 of 2)

Field	Bits	Access	Function	Default
map_rx_sync_mode	[2]	RW	Rx MAP synchronization mode if **Enable MAP interface synchronization with core clock** is turned off. Values are: 0: FIFO mode 1: Synchronous buffer mode	1'h0
map_mode	[1:0]	RW/RO	AxC mapping mode. If you select **All** as the value for the **Mapping mode(s)** parameter in the CPRI IP core, this register field determines the current AxC mapping mode. If you select any other value for the **Mapping mode(s)** parameter, this register field is ignored (Read-only). Register field values are: 00: Basic mapping scheme (UMTS/LTE standard in which all MAP interfaces use the same sample rate, as described in the CPRI V4.2 Specification sections 4.2.7.2.2 and 4.2.7.2.3). 01: CPRI V4.2 Specification section 4.2.7.2.5: Method 1: IQ sample based. New Method 1 implementation in the Quartus II software v11.1 release. 10: CPRI V4.2 Specification section 4.2.7.2.7: Method 3: Backward compatible. 11: CPRI V4.2 Specification section 4.2.7.2.5: Method 1: IQ sample based. This implementation is available in all pre-11.1 releases of the Altera CPRI IP core as advanced mapping mode 2'b01. Values 01, 10, and 11 indicate advanced AxC mapping modes in which each MAP interface can implement a different channel rate and radio standard.	2'h0

Table 7–32. CPRI_MAP_CNT_CONFIG—Basic UMTS/LTE Mapping Configuration—Offset: 0x104 (1)

Field	Bits	Access	Function	Default
RSRV	[31:13]	UR0	Reserved.	19'h0
map_ac	[12:8]	RW	Number of active data channels (antenna-carrier interfaces).	5'h0
RSRV	[7:5]	UR0	Reserved.	3'h0
map_n_ac	[4:0]	RW	Oversampling factor on each active data channel.	5'h0

Note to Table 7–32:

(1) This register applies only to map_mode 00, in which each antenna-carrier interface has the same sample rate.

Table 7–33. CPRI_MAP_TBL_CONFIG—K Parameter Config for Advanced Table-Based Mapping—Offset: 0x0108 (1) (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:WIDTH_K]	UR0	Reserved.	0

Table 7–33. CPRI_MAP_TBL_CONFIG—K Parameter Config for Advanced Table-Based Mapping—Offset: 0x0108 (1) (Part 2 of 2)

Field	Bits	Access	Function	Default
K	[WIDTH_K-1:0]	RW	Number of basic frames in AxC container block.	0

Note to Table 7–33:

(1) This register applies only to map_mode 01, 10, or 11, the advanced mapping modes.

Table 7–34. CPRI_MAP_TBL_INDEX—Advanced Mapping Configuration Table Index—Offset: 0x10C (1)

Field	Bits	Access	Function	Default
RSRV	[31:11]	UR0	Reserved.	21'h0
map_conf_index	[10:0]	RW	Index for configuring antenna-carrier interface information in the advanced mapping Rx and Tx tables. The value in this field determines the table entries that appear in the CPRI_MAP_TBL_RX and CPRI_MAP_TBL_TX registers.	11'h0

Note to Table 7–34:

(1) This register applies only to map_mode 01, 10, or 11, the advanced mapping modes.

Table 7–35. CPRI_MAP_TBL_RX—Advanced Mapping Rx Configuration Table—Offset: 0x110 (1)

Field	Bits	Access	Function	Default
RSRV	[31:29]	UR0	Reserved.	3'h0
width	[28:24]	RW	Width of IQ sample in timeslot. Specified as 1/2 the number of bits in the IQ sample. This field is used in 15-bit mode with advanced mapping mode 01 and in 16-bit mode with all advanced mapping modes. In 15-bit mode with advanced mapping modes 10 and 11, you must set this field to the value of 15 to indicate the full 30 bits of the 32-bit timeslot.	5'h0
RSRV	23:21]	UR0	Reserved.	3'h0
position	[20:16]	RW	Starting bit position of IQ sample in timeslot. Specified as 1/2 the bit position number. This field is used in 15-bit mode with advanced mapping mode 01 and in 16-bit mode with all advanced mapping modes. In 15-bit mode with advanced mapping modes 10 and 11, you must set this field to the offset of the next available bit for your 30-bit sample in the current 32-bit timeslot.	5'h0
RSRV	[15:WIDTH_N_MAP+8]	UR0	Reserved.	0
ac	[WIDTH_N_MAP +7:8]	RW	AxC interface number.	0
RSRV	[7:1]	UR0	Reserved.	7'h0
enable	[0]	RW	Enable mapping of IQ sample into current timeslot.	1'h0

Note to Table 7–35:

(1) Currently configurable entry in the advanced mapping Rx table. This register applies only to map_mode 01, 10, or 11, the advanced mapping modes.

Table 7–36. CPRI_MAP_TBL_TX—Advanced Mapping Tx Configuration Table—Offset: 0x114 (1)

Field	Bits	Access	Function	Default
RSRV	[31:29]	UR0	Reserved.	3'h0
width	[28:24]	RW	Width of IQ sample in timeslot. Specified as 1/2 the number of bits in the IQ sample. This field is used in 15-bit mode with advanced mapping mode 01 and in 16-bit mode with all advanced mapping modes. In 15-bit mode with advanced mapping modes 10 and 11, you must set this field to the value of 15 to indicate the full 30 bits of the 32-bit timeslot.	5'h0
RSRV	23:21]	UR0	Reserved.	3'h0
position	[20:16]	RW	Starting bit position of IQ sample in timeslot. Specified as 1/2 the bit position number. This field is used in 15-bit mode with advanced mapping mode 01 and in 16-bit mode with all advanced mapping modes. In 15-bit mode with advanced mapping modes 10 and 11, you must set this field to the offset of the next available bit for your 30-bit sample in the current 32-bit timeslot.	5'h0
RSRV	[15:WIDTH_N_MAP+8]	UR0	Reserved.	0
ac	[WIDTH_N_MAP +7:8]	RW	AxC interface number.	0
RSRV	[7:1]	UR0	Reserved.	7'h0
enable	[0]	RW	Enable mapping of IQ sample into current timeslot.	1'h0

Note to Table 7–36:

(1) Currently configurable entry in the advanced mapping Tx table. This register applies only to map_mode 01, 10, or 11, the advanced mapping modes.

Table 7–37. CPRI_MAP_OFFSET_RX—MAP Rx Frame Offset (1), (2)—Offset: 0x118

Field	Bits	Access	Function	Default
RSRV	[31:17]	UR0	Reserved.	15'h0
map_rx_hf_resync	[16]	RW	Enables synchronization every hyperframe instead of every radio frame. When asserted, the map_rx_offset_z field is ignored.	1'h0
map_rx_offset_z	[15:8]	RW	Hyperframe number for start of MAP receiver AxC container block write to each enabled mapN Rx buffer.	8'h0
map_rx_offset_x	[7:0]	RW	Basic frame number for start of MAP receiver AxC container block write to each enabled mapN Rx buffer.	8'h0

Notes to Table 7–37:

(1) In synchronous buffer mode, the offset specified in this register must precede (be less than) the offset specified in the CPRI_START_OFFSET_RX register described in Table 7–39. For an explanation of this requirement and an overview of the considerations in determining the value in this register, refer to "MAP Receiver in Synchronous Buffer Mode" on page 4–18 and to "Rx Path Delay" on page D–3. If your register values do not comply with this requirement, your CPRI IP core will experience data corruption on the active data channels in the synchronous buffer synchronization mode.

(2) This register does not participate in data transfer synchronization on the antenna-carrier interfaces in FIFO mode.

Table 7–38. CPRI_MAP_OFFSET_TX—MAP Tx Frame Offset (1), (2)—Offset: 0x11C

Field	Bits	Access	Function	Default
RSRV	[31:17]	UR0	Reserved.	15'h0
map_tx_hf_resync	[16]	RW	Enables synchronization every hyperframe instead of every radio frame. When asserted, the map_tx_offset_z field is ignored.	1'h0
map_tx_offset_z	[15:8]	RW	Hyperframe number for start of read of MAP transmitter AxC container block from each enabled mapN Tx buffer. The CPRI IP core reads the data from the mapN Tx buffer and routes it to the CPRI frame buffer to be prepared for transmission on the CPRI link.	8'h0
map_tx_offset_x	[7:0]	RW	Basic frame number for start of read of MAP transmitter AxC container block from each enabled mapN Tx buffer. The CPRI IP core reads the data from the mapN Tx buffer and routes it to the CPRI frame buffer to be prepared for transmission on the CPRI link.	8'h0

Notes to Table 7–38:

(1) In synchronous buffer mode, the offset specified in this register must follow (be greater than) the offset specified in the CPRI_START_OFFSET_TX register described in Table 7–40. For an explanation of this requirement and an overview of the considerations in determining the value in this register, refer to "MAP Transmitter in Synchronous Buffer Mode" on page 4–24 and to "Tx Path Delay" on page D–9. If your register values do not comply with this requirement, your CPRI IP core will experience data corruption on the active data channels in the synchronous buffer synchronization mode.

(2) This register does not participate in data transfer synchronization on the antenna-carrier interfaces in FIFO mode.

Table 7–39. CPRI_START_OFFSET_RX—Rx Start Frame Offset (1), (2)—Offset: 0x120

Field	Bits	Access	Function	Default
RSRV	[31:25]	UR0	Reserved.	7'h0
start_rx_hf_resync	[24]	RW	Enables synchronization every hyperframe instead of every radio frame. When asserted, the start_rx_offset_z field is ignored.	1'h0
RSRV	[23:22]	UR0	Reserved.	2'h0
start_rx_offset_seq	[21:16]	RW	Sequence number for start of cpri_rx_start synchronization output.	6'h0
start_rx_offset_z	[15:8]	RW	Hyperframe number for start of cpri_rx_start synchronization output.	8'h0
start_rx_offset_x	[7:0]	RW	Basic frame number for start of cpri_rx_start synchronization output.	8'h0

Notes to Table 7–39:

(1) In synchronous buffer mode, the offset specified in this register must follow (be greater than) the offset specified in the CPRI_MAP_OFFSET_RX register described in Table 7–37. For an explanation of this requirement and an overview of the considerations in determining the value in this register, refer to "MAP Receiver in Synchronous Buffer Mode" on page 4–18 and to "Rx Path Delay" on page D–3. If your register values do not comply with this requirement, your CPRI IP core will experience data corruption on the active data channels in the synchronous buffer synchronization mode.

(2) This register does not participate in data transfer synchronization on the antenna-carrier interfaces in FIFO mode or in the internally-clocked mode.

Table 7–40. CPRI_START_OFFSET_TX—Tx Start Frame Offset (1), (2)—Offset: 0x124 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:25]	UR0	Reserved.	7'h0
start_tx_hf_resync	[24]	RW	Enables synchronization every hyperframe instead of every radio frame. When asserted, the start_tx_offset_z field is ignored.	1'h0

Table 7–40. CPRI_START_OFFSET_TX—Tx Start Frame Offset (1), (2)—Offset: 0x124 (Part 2 of 2)

Field	Bits	Access	Function	Default
RSRV	[23:22]	UR0	Reserved.	2'h0
start_tx_offset_seq	[21:16]	RW	Sequence number for start of cpri_tx_start synchronization output.	6'h0
start_tx_offset_z	[15:8]	RW	Hyperframe number for start of cpri_tx_start synchronization output.	8'h0
start_tx_offset_x	[7:0]	RW	Basic frame number for start of cpri_tx_start synchronization output.	8'h0

Notes to Table 7–40:

(1) In synchronous buffer mode, the offset specified in this register must precede (be less than) the offset specified in the CPRI_MAP_OFFSET_TX register described in Table 7–38. For an explanation of this requirement and an overview of the considerations in determining the value in this register, refer to "MAP Transmitter in Synchronous Buffer Mode" on page 4–24 and to "Tx Path Delay" on page D–9. If your register values do not comply with this requirement, your CPRI IP core will experience data corruption on the active data channels in the synchronous buffer synchronization mode.

(2) This register does not participate in data transfer synchronization on the antenna-carrier interfaces in FIFO mode or in the internally-clocked mode.

Table 7–41. CPRI_MAP_RX_READY_THR—CPRI Mapping Rx Ready Threshold—Offset: 0x128

Field	Bits	Access	Function	Default
RSRV	[31:4]	UR0	Reserved.	28'h0
map_rx_ready_thr	[3:0]	RW	Threshold for assertion of the mapN_rx_valid signal in FIFO mode, for all data channels N. The mapN_rx_valid signal is asserted only when the MAP Rx buffer for data channel N fills beyond this threshold value. All the MAP Rx buffers have the same depth, 16.	4'h8

Table 7–42. CPRI_MAP_TX_READY_THR—CPRI Mapping Tx Ready Threshold—Offset: 0x12C

Field	Bits	Access	Function	Default
RSRV	[31:4]	UR0	Reserved.	28'h0
map_tx_ready_thr	[3:0]	RW	Threshold for assertion of the mapN_tx_ready signal in FIFO mode, for all data channels N. The mapN_tx_ready signal is asserted only after the Map Tx buffer for data channel N empties to a level below this threshold value. All the MAP Tx buffers have the same depth, 16.	4'h8

Table 7–43. CPRI_MAP_TX_START_THR—CPRI Mapping Tx Start Threshold—Offset: 0x130

Field	Bits	Access	Function	Default
RSRV	[31:4]	UR0	Reserved.	28'h0
map_tx_start_thr	[3:0]	RW	In FIFO mode, threshold for starting transmission from the MAP Tx buffers for all data channels N to the CPRI transmitter interface. Data transmission from each MAP Tx buffer starts only after that MAP Tx buffer fills beyond this threshold value. All the MAP Tx buffers have the same depth, 16. This register does not participate in data transfer coordination in synchronous buffer mode or in the internally-clocked synchronization mode.	4'h7

Table 7–44. CPRI_PRBS_CONFIG—PRBS Generation Pattern Configuration—Offset: 0x13C

Field	Bits	Access	Function	Default
RSRV	[31:2]	UR0	Reserved.	30'h0
prbs_mode	[1:0]	RW	PRBS loopback and pattern mode. Values are: 00: Normal mode (IQ samples, no loopback) 01: Counter sequence (internal loopback path) 10: PRBS 2^{23}-1 inverted (internal loopback path) 11: Reserved The PRBS mode is common to all antenna-carrier interfaces.	2'h0

Table 7–45. CPRI_PRBS_STATUS—PRBS Data Validation Status—Offset: 0x140–0x144 (1)

Field	Bits	Access	Function	Default
PRBS_error	[(N_MAP+15):16]	RC	Indicates PRBS error detected on the corresponding antenna-carrier interfaces.	16'h0
PRBS_valid	[(N_MAP-1):0]]	RC	Indicates a valid PRBS pattern on the corresponding antenna-carrier receiver interfaces.	16'h0

Note to Table 7–45:

(1) If this CPRI IP core has more than 16 antenna-carrier interfaces (N_MAP > 16), the status for antenna-carrier interfaces 0 through 15 is in the register at offset 0x140, and the status for antenna-carrier interfaces 16 and up is in the register at offset 0x144. The maximum number of antenna-carrier interfaces in the CPRI IP core is 24.

Table 7–46. CPRI_IQ_RX_BUF_CONTROL—MAP Receiver FIFO Buffer Control—Offset: 0x150

Field	Bits	Access	Function	Default
RSRV	[31:N_MAP]	UR0	Reserved.	0
map_rx_enable	[(N_MAP-1):0]]	RW	Enables or disables the corresponding antenna-carrier receiver interfaces. The bits of this field propagate to the corresponding cpri_map_rx_en output signals.	(N_MAP)'h7F (all 1s)

Table 7–47. CPRI_IQ_TX_BUF_CONTROL—MAP Transmitter FIFO Buffer Control—Offset: 0x160

Field	Bits	Access	Function	Default
RSRV	[31:N_MAP]	UR0	Reserved.	0
map_tx_enable	[(N_MAP-1):0]]	RW	Enables or disables the corresponding antenna-carrier transmitter interfaces. The bits of this field propagate to the corresponding cpri_map_tx_en output signals.	(N_MAP)'h7F (all 1s)

Table 7–48. CPRI_IQ_RX_BUF_STATUS—MAP Receiver FIFO Buffer Status—Offset: 0x180–0x184 (1), (2)

Field	Bits	Access	Function	Default
buffer_rx_underflow	[(N_MAP+15):16]	RC	Indicates MAP Rx buffer underflow in the corresponding antenna-carrier interfaces.	16'h0
buffer_rx_overflow	[(N_MAP-1):0]]	RC	Indicates MAP Rx buffer overflow in the corresponding antenna-carrier interfaces.	16'h0

Notes to Table 7–48:

(1) If this CPRI IP core has more than 16 antenna-carrier interfaces (N_MAP > 16), the status for antenna-carrier interfaces 0 through 15 is in the register at offset 0x180, and the status for antenna-carrier interfaces 16 and up is in the register at offset 0x184. The maximum number of antenna-carrier interfaces in the CPRI IP core is 24.

(2) This register does not participate in data transfer synchronization on the antenna-carrier interfaces in the internally-clocked mode.

Table 7–49. CPRI_IQ_TX_BUF_STATUS—MAP Transmitter FIFO Buffer Status—Offset: 0x1A0–0x1A4 (1), (2)

Field	Bits	Access	Function	Default
buffer_tx_underflow	[(N_MAP+15):16]	RC	Indicates MAP Tx buffer underflow in the corresponding antenna-carrier interfaces.	16'h0
buffer_tx_overflow	[(N_MAP-1):0]]	RC	Indicates MAP Tx buffer overflow in the corresponding antenna-carrier interfaces.	16'h0

Notes to Table 7–49:

(1) If this CPRI IP core has more than 16 antenna-carrier interfaces (N_MAP > 16), the status for antenna-carrier interfaces 0 through 15 is in the register at offset 0x1A0, and the status for antenna-carrier interfaces 16 and up is in the register at offset 0x1A4. The maximum number of antenna-carrier interfaces in the CPRI IP core is 24.

(2) This register does not participate in data transfer synchronization on the antenna-carrier interfaces in the internally-clocked mode.

Ethernet Registers

This section lists the Ethernet registers. Table 7–50 provides a memory map for the Ethernet registers. Table 7–51 through Table 7–66 describe the Ethernet registers in the CPRI IP core.

If you turn off the **Include MAC block** parameter, your application cannot access the Ethernet registers. In that case, attempts to access these registers read zeroes and do not write successfully, as for a Reserved register address.

For more information about these registers, refer to "Accessing the Ethernet Channel" on page 4–42.

Table 7–50. CPRI Ethernet Registers Memory Map (Part 1 of 2)

Address	Name	Expanded Name
0x200	ETH_RX_STATUS	Ethernet Receiver Module Status
0x204	ETH_TX_STATUS	Ethernet Transmitter Module Status
0x208	ETH_CONFIG_1	Ethernet Feature Configuration 1
0x20C	ETH_CONFIG_2	Ethernet Feature Configuration 2
0x210	ETH_RX_CONTROL	Ethernet Rx Control
0x214	ETH_RX_DATA	Ethernet Rx Data
0x218	ETH_RX_DATA_WAIT	Ethernet Rx Data With Wait-State Insertion
0x21C	ETH_TX_CONTROL	Ethernet Tx Control

Table 7–50. CPRI Ethernet Registers Memory Map (Part 2 of 2)

Address	Name	Expanded Name
0x220	ETH_TX_DATA	Ethernet Tx Data
0x224	ETH_TX_DATA_WAIT	Ethernet Tx Data With Wait-State Insertion
0x228	Reserved	
0x22C	ETH_MAC_ADDR_MSB	Ethernet MAC Address MSB (16 bits)
0x230	ETH_MAC_ADDR_LSB	Ethernet MAC Address LSB (32 bits)
0x234	ETH_HASH_TABLE	Ethernet Multicast Filtering Hash Table
0x238–0x240	Reserved	
0x244	ETH_FWD_CONFIG	Ethernet Forwarding Configuration
0x248	ETH_CNT_RX_FRAME	Ethernet Receiver Module Frame Counter
0x24C	ETH_CNT_TX_FRAME	Ethernet Transmitter Module Frame Counter

Table 7–51. ETH_RX_STATUS—Ethernet Receiver Module Status—Offset: 0x200

Field	Bits	Access	Function	Default
RSRV	[31:7]	UR0	Reserved.	25'h0
rx_ready_block	[6]	RO	Indicates that an 8-word block of Ethernet data is available to be transmitted on the Ethernet channel.	1'h0
rx_ready_end	[5]	RO	Indicates the end-of-packet (EOP) is available in the Ethernet Rx buffer, ready to be transmitted on the Ethernet channel.	1'h0
rx_length	[4:3]	RO	Length of the final word in the packet. Values are: 00: 1 valid byte 01: 2 valid bytes 10: 3 valid bytes 11: 4 valid bytes	2'h0
rx_abort	[2]	RO	Indicates the current Ethernet Rx packet is aborted.	1'h0
rx_eop	[1]	RO	Indicates that the next ready data word contains the end-of-packet byte.	1'h0
rx_ready	[0]	RO	Indicates that at least one 32-bit word of Ethernet data is available in the Ethernet Rx buffer and ready to be read.	1'h0

Table 7–52. ETH_TX_STATUS—Ethernet Transmitter Module Status—Offset: 0x204

Field	Bits	Access	Function	Default
RSRV	[31:3]	UR0	Reserved.	29'h0
tx_ready_block	[2]	RO	Indicates that the Ethernet Tx module is ready to receive an 8-word block of data.	1'h0
tx_abort	[1]	RO	Indicates the current Ethernet Tx packet is aborted.	1'h0
tx_ready	[0]	RO	Indicates that the Ethernet Tx module is ready to receive at least one 32-bit word of data.	1'h0

Table 7–53. ETH_CONFIG_1—Ethernet Feature Configuration 1—Offset: 0x208

Field	Bits	Access	Function	Default
RSRV	[31:20]	UR0	Reserved.	11'h0
intr_tx_ready_block_en	[19]	RW	Indicates an interrupt is generated when tx_ready_block is asserted, if intr_en and intr_tx_en are asserted.	1'h0
intr_tx_abort_en	[18]	RW	Indicates an interrupt is generated when tx_abort is asserted, if intr_en and intr_tx_en are asserted.	1'h0
intr_tx_ready_en	[17]	RW	Indicates an interrupt is generated when tx_ready is asserted, if intr_en and intr_tx_en are asserted.	1'h0
intr_rx_ready_block_en	[16]	RW	Indicates an interrupt is generated when rx_ready_block is asserted, if intr_en and intr_rx_en are asserted.	1'h0
intr_rx_ready_end_en	[15]	RW	Indicates an interrupt is generated when rx_ready_end is asserted, if intr_en and intr_rx_en are asserted.	1'h0
intr_rx_abort_en	[14]	RW	Indicates an interrupt is generated when rx_abort is asserted, if intr_en and intr_rx_en are asserted.	1'h0
intr_rx_ready_en	[13]	RW	Indicates an interrupt is generated when rx_ready is asserted, if intr_en and intr_rx_en are asserted.	1'h0
intr_tx_en	[12]	RW	Ethernet Tx interrupt enable.	1'h0
intr_rx_en	[11]	RW	Ethernet Rx interrupt enable.	1'h0
intr_en	[10]	RW	Ethernet global interrupt enable.	1'h0
rx_long_frame_en	[9]	RW	Enable reception of Rx Ethernet frames longer than 1536 bytes.	1'h0
rx_preamble_abort_en	[8]	RW	Indicates that Rx frames with an illegal preamble nibble before the SFD are discarded.	1'h0
broadcast_en	[7]	RW	Enable reception of Ethernet broadcast packets.	1'h0
multicast_flt_en	[6]	RW	Enable reception of multicast Ethernet packets allowed by the hash function.	1'h0
mac_check	[5]	RW	Enable check of Rx Ethernet MAC address.	1'h0
length_check	[4]	RW	Indicates that a length check is performed on Rx packets, and those with length less than 64 bytes are discarded.	1'h0
mac_reset	[3]	RW	Reset the Ethernet MAC.	1'h1
RSRV	[2]	RO	Reserved.	1'h0
little_endian	[1]	RW	Indicates that the Ethernet channel receive and transmit data is formatted in little endian byte order.	1'h0
RSRV	[0]	RO	Reserved.	1'h0

Table 7–54. ETH_CONFIG_2—Ethernet Feature Configuration 2—Offset: 0x20C

Field	Bits	Access	Function	Default
RSRV	[31:1]	UR0	Reserved.	31'h0
crc_enable	[0]	RW	Enables insertion of Ethernet frame check sequence (FCS) at the end of the Ethernet frame.	1'h0

Table 7–55. ETH_RX_CONTROL—Ethernet Rx Control—Offset: 0x210

Field	Bits	Access	Function	Default
RSRV	[31:1]	RO	Reserved.	31'h0
rx_discard	[0]	WO	Indicates that the Ethernet receiver module should discard the current Ethernet Rx frame.	1'h0

Table 7–56. ETH_RX_DATA—Ethernet Rx Data—Offset: 0x214

Field	Bits	Access	Function	Default
rx_data	[31:0]	RO	Ethernet Rx frame data.	1'h0

Table 7–57. ETH_RX_DATA_WAIT—Ethernet Rx Data with Wait-State Insertion—Offset: 0x218

Field	Bits	Access	Function	Default
rx_data	[31:0]	RO	Ethernet Rx frame data.	1'h0

Table 7–58. ETH_TX_CONTROL—Ethernet Tx Control—Offset: 0x21C

Field	Bits	Access	Function	Default
RSRV	[31:4]	UR0	Reserved.	28'h0
tx_length	[3:2]	WO	Length of the final word in the packet. Values are: 00: 1 valid byte 01: 2 valid bytes 10: 3 valid bytes 11: 4 valid bytes This field is valid when the tx_eop bit is asserted.	2'h0
tx_discard	[1]	WO	Indicates that the Ethernet transmitter module should discard the current Ethernet Tx frame.	1'h0
tx_eop	[0]	WO	Indicates that the next data word to be written to the ETH_TX_DATA or ETH_TX_DATA_WAIT register contains the end-of-packet byte for this Tx packet.	1'h0

Table 7–59. ETH_TX_DATA—Ethernet Tx Data—Offset: 0x220

Field	Bits	Access	Function	Default
tx_data	[31:0]	RW	Ethernet Tx frame data. If the tx_ready bit of the ETH_TX_READY register is zero when tx_data is loaded, the Ethernet transmitter module aborts the packet.	32'h0

Table 7–60. ETH_TX_DATA_WAIT—Ethernet Tx Data with Wait-State Insertion—Offset: 0x224

Field	Bits	Access	Function	Default
tx_data	[31:0]	RW	Ethernet Tx frame data. If the Ethernet transmitter module writes Ethernet data to this register, it waits until data is ready, unless the CPU times out the operation.	1'h0

Table 7–61. ETH_ADDR_MSB—Ethernet MAC Address MSB—Offset: 0x22C

Field	Bits	Access	Function	Default
RSRV	[31:16]	UR0	Reserved.	16'h0
mac[47:32]	[15:0]	RW	Most significant bits (16 bits) of local Ethernet MAC address.	16'h0

Table 7–62. ETH_ADDR_LSB—Ethernet MAC Address LSB—Offset: 0x230

Field	Bits	Access	Function	Default
mac[31:0]	[31:0]	RW	Least significant bits (32 bits) of local Ethernet MAC address.	32'h0

Table 7–63. ETH_HASH_TABLE—Ethernet Multicast Filtering Hash Table—Offset: 0x234

Field	Bits	Access	Function	Default
hash	[31:0]	RW	32-bit hash table for multicast filtering. If the group address bit of the destination MAC address is set, and multicast address filtering is enabled, this register filters the packets to be accepted and discarded, as follows: If every bit set in this register is also set in the lower 32 bits of the destination MAC address, the packet is accepted. Otherwise, the packet is discarded.	32'h0

Table 7–64. ETH_FWD_CONFIG—Ethernet Forwarding Configuration—Offset: 0x244

Field	Bits	Access	Function	Default
RSRV	[31:17]	UR0	Reserved.	15'h0
tx_start_thr	[16:1]	RW	Transmit start threshold. If store-and-forward mode is disabled, transmission to the CPRI link starts when this number of 32-bit words are stored in the Tx buffer.	16'h0004
tx_st_fwd	[0]	RW	Transmit store-and-forward mode. In store-and-forward mode, a full packet is stored in the Tx buffer before transmission starts. Packets longer than the Tx buffer are aborted.	1'h0

Table 7–65. ETH_CNT_RX_FRAME—Ethernet Receiver Module Frame Counter—Offset: 0x248

Field	Bits	Access	Function	Default
eth_cnt_rx_frame	[31:0]	RO	Number of frames received from the CPRI receiver.	32'h0

Table 7–66. ETH_CNT_TX_FRAME—Ethernet Transmitter Module Frame Counter—Offset: 0x24C

Field	Bits	Access	Function	Default
eth_cnt_tx_frame	[31:0]	RO	Number of frame transmitted to the CPRI transmitter.	32'h0

HDLC Registers

This section lists the HDLC registers. Table 7–67 provides a memory map for the HDLC registers. Table 7–68 through Table 7–81 describe the HDLC registers in the CPRI IP core.

If you turn off the **Include HDLC block** parameter, your application cannot access the HDLC registers. In that case, attempts to access these registers read zeroes and do not write successfully, as for a Reserved register address.

For more information about these registers, refer to "Accessing the HDLC Channel" on page 4–45.

Table 7–67. CPRI HDLC Registers Memory Map

Address	Name	Expanded Name
0x300	HDLC_RX_STATUS	HDLC Receiver Module Status
0x304	HDLC_TX_STATUS	HDLC Transmitter Module Status
0x308	HDLC_CONFIG_1	HDLC Feature Configuration 1
0x30C	HDLC_CONFIG_2	HDLC Feature Configuration 2
0x310	HDLC_RX_CONTROL	HDLC Rx Control
0x314	HDLC_RX_DATA	HDLC Rx Data
0x318	HDLC_RX_DATA_WAIT	HDLC Rx Data With Wait-State Insertion
0x31C	HDLC_TX_CONTROL	HDLC Tx Control
0x320	HDLC_TX_DATA	HDLC Tx Data
0x324	HDLC_TX_DATA_WAIT	HDLC Tx Data With Wait-State Insertion
0x328	HDLC_RX_EX_STATUS	HDLC Rx Additional Status
0x32C	HDLC_CONFIG_3	HDLC Feature Configuration 3
0x330	HDLC_CNT_RX_FRAME	HDLC Receiver Module Frame Counter
0x334	HDLC_CNT_TX_FRAME	HDLC Transmitter Module Frame Counter

Table 7–68. HDLC_RX_STATUS—HDLC Receiver Module Status—Offset: 0x300 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:7]	UR0	Reserved.	25'h0
rx_ready_block	[6]	RO	Indicates that an eight-word block of HDLC data is available in the HDLC Rx buffer to be transmitted on the HDLC channel.	1'h0
rx_ready_end	[5]	RO	Indicates the end-of-packet (EOP) is available in the HDLC Rx buffer, ready to be transmitted on the HDLC channel.	1'h0
rx_length	[4:3]	RO	Length of the final word in the packet. Values are: 00: 1 valid byte 01: 2 valid bytes 10: 3 valid bytes 11: 4 valid bytes	2'h0
rx_abort	[2]	RO	Indicates the current HDLC Rx packet is aborted.	1'h0

Table 7–68. HDLC_RX_STATUS—HDLC Receiver Module Status—Offset: 0x300 (Part 2 of 2)

Field	Bits	Access	Function	Default
rx_eop	[1]	RO	Indicates that the next ready data word contains the end-of-packet byte.	1'h0
rx_ready	[0]	RO	Indicates that at least one 32-bit word of HDLC data is available in the HDLC Rx buffer.	1'h0

Table 7–69. HDLC_TX_STATUS—HDLC Transmitter Module Status—Offset: 0x304

Field	Bits	Access	Function	Default
RSRV	[31:3]	UR0	Reserved.	29'h0
tx_ready_block	[2]	RO	Indicates that the HDLC Tx module is ready to receive an 8-word block of data.	1'h0
tx_abort	[1]	RO	Indicates the current HDLC Tx packet is aborted.	1'h0
tx_ready	[0]	RO	Indicates that the HDLC Tx module is ready to receive at least one 32-bit word of data.	1'h0

Table 7–70. HDLC_CONFIG—HDLC Feature Configuration 1—Offset: 0x308 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:20]	UR0	Reserved.	11'h0
intr_tx_ready_block_en	[19]	RW	Indicates an interrupt is generated when tx_ready_block is asserted, if intr_en and intr_tx_en are asserted.	1'h0
intr_tx_abort_en	[18]	RW	Indicates an interrupt is generated when tx_abort is asserted, if intr_en and intr_tx_en are asserted.	1'h0
intr_tx_ready_en	[17]	RW	Indicates an interrupt is generated when tx_ready is asserted, if intr_en and intr_tx_en are asserted.	1'h0
intr_rx_ready_block_en	[16]	RW	Indicates an interrupt is generated when rx_ready_block is asserted, if intr_en and intr_rx_en are asserted.	1'h0
intr_rx_ready_end_en	[15]	RW	Indicates an interrupt is generated when rx_ready_end is asserted, if intr_en and intr_rx_en are asserted.	1'h0
intr_rx_abort_en	[14]	RW	Indicates an interrupt is generated when rx_abort is asserted, if intr_en and intr_rx_en are asserted.	1'h0
intr_rx_ready_en	[13]	RW	Indicates an interrupt is generated when rx_ready is asserted, if intr_en and intr_rx_en are asserted.	1'h0
intr_tx_en	[12]	RW	HDLC Tx interrupt enable.	1'h0
intr_rx_en	[11]	RW	HDLC Rx interrupt enable.	1'h0
intr_en	[10]	RW	HDLC global interrupt enable.	1'h0
rx_long_frame_en	[9]	RW	Enable reception of Rx HDLC frames longer than 1536 bytes.	1'h0
RSRV	[8:5]	UR0	Reserved.	4'h0
length_check	[4]	RW	Indicates that a length check is performed on Rx packets, and those with length less than 64 bytes are discarded.	1'h0
RSRV	[3:2]	UR0	Reserved.	2'h0

Table 7–70. HDLC_CONFIG—HDLC Feature Configuration 1—Offset: 0x308 (Part 2 of 2)

Field	Bits	Access	Function	Default
little_endian	[1]	RW	Indicates that the HDLC channel receive and transmit data is formatted in little endian byte order.	1'h0
RSRV	[0]	UR0	Reserved.	1'h0

Table 7–71. HDLC_CONFIG_2—HDLC Feature Configuration 2—Offset: 0x30C

Field	Bits	Access	Function	Default
RSRV	[31:1]	UR0	Reserved.	31'h0
crc_enable	[0]	RW	Enables insertion of HDLC CRC at the end of the HDLC frame.	1'h0

Table 7–72. HDLC_RX_CONTROL—HDLC Rx Control—Offset: 0x310

Field	Bits	Access	Function	Default
RSRV	[31:1]	RO	Reserved.	31'h0
rx_discard	[0]	WO	Indicates that the HDLC receiver module should discard the current HDLC Rx frame.	1'h0

Table 7–73. HDLC_RX_DATA—HDLC Rx Data—Offset: 0x314

Field	Bits	Access	Function	Default
rx_data	[31:0]	RO	HDLC Rx frame data. If the HDLC receiver module takes HDLC data from this register, if data is not ready when the module expects it, the HDLC receiver module aborts the packet.	1'h0

Table 7–74. HDLC_RX_DATA_WAIT—HDLC Rx Data with Wait-State Insertion—Offset: 0x318

Field	Bits	Access	Function	Default
rx_data	[31:0]	RO	HDLC Rx frame data. If the HDLC receiver module takes HDLC data from this register, it inserts wait states on the HDLC channel until data is ready, unless the CPU times out the operation.	1'h0

Table 7–75. HDLC_TX_CONTROL—HDLC Tx Control—Offset: 0x31C

Field	Bits	Access	Function	Default
RSRV	[31:4]	UR0	Reserved.	28'h0
tx_length	[3:2]	RW	Length of the final word in the packet. Values are: 00: 1 valid byte 01: 2 valid bytes 10: 3 valid bytes 11: 4 valid bytes This field is valid when the tx_eop bit is asserted.	1'h0
tx_discard	[1]	WO	Indicates that the HDLC transmitter module should discard the current HDLC Tx frame.	1'h0
tx_eop	[0]	RW	Indicates that the next data word to be written to the HDLC_TX_DATA or HDLC_TX_DATA_WAIT register contains the end-of-packet byte for this Tx packet.	1'h0

Table 7–76. HDLC_TX_DATA—HDLC Tx Data—Offset: 0x320

Field	Bits	Access	Function	Default
tx_data	[31:0]	RW	HDLC Tx frame data. If the HDLC transmitter module writes HDLC data to this register, if data is not ready when the module expects it, the HDLC transmitter module aborts the packet.	1'h0

Table 7–77. HDLC_TX_DATA_WAIT—HDLC Tx Data with Wait-State Insertion—Offset: 0x324

Field	Bits	Access	Function	Default
tx_data	[31:0]	RW	HDLC Tx frame data. If the HDLC transmitter module writes HDLC data to this register, it waits until data is ready, unless the CPU times out the operation.	1'h0

Table 7–78. HDLC_RX_EX_STATUS—HDLC Rx Additional Status—Offset: 0x328

Field	Bits	Access	Function	Default
RSRV	[31:7]	UR0	Reserved.	25'h0
CRC_error	[6]	RC	Indicates that an HDLC frame with a CRC error was received.	1'h0
RSRV	[5:0]	UR0	Reserved.	6'h0

Table 7–79. HDLC_CONFIG_3—HDLC Feature Configuration 3—Offset: 0x32C

Field	Bits	Access	Function	Default
RSRV	[31:17]	UR0	Reserved.	15'h0
tx_start_thr	[16:8]	RW	Transmit start threshold. If store-and-forward mode is disabled, transmission to the CPRI link starts when this number of 32-bit words are stored in the Tx buffer.	9'h004
RSRV	[7:2]	UR0	Reserved.	5'h0
rx_crc_en	[1]	RW	Indicates that CRC checking is enabled.	1'h0
tx_st_fwd	[0]	RW	Transmit store-and-forward mode. In store-and-forward mode, a full packet is stored before transmission starts. Packets longer than the Tx buffer are aborted.	1'h0

Table 7–80. HDLC_CNT_RX_FRAME—HDLC Receiver Module Frame Counter—Offset: 0x330

Field	Bits	Access	Function	Default
hdlc_cnt_rx_frame	[31:0]	RO	Number of frames received from the CPRI receiver.	32'h0

Table 7–81. HDLC_CNT_TX_FRAME—HDLC Transmitter Module Frame Counter—Offset: 0x334

Field	Bits	Access	Function	Default
hdlc_cnt_tx_frame	[31:0]	RO	Number of frame transmitted to the CPRI transmitter.	32'h0

8. Testbenches

The Altera CPRI IP core includes nine demonstration testbenches for your use. The testbenches provide examples of how to use the Avalon-MM and Avalon-ST interfaces to generate and process CPRI transactions using the MII, MAP, and AUX interfaces and how to perform autorate negotiation.

The testbenches are available only if you turn on **Generate Example Design** when prompted during generation of the CPRI IP core. Refer to "Specifying Parameters" on page 2–1.

All nine demonstration testbenches demonstrate the following functions:

- Writing to the registers
- Frame synchronization process
- Transmission and reception of CPRI link data

The individual testbenches demonstrate the additional functions listed in Table 8–1.

Table 8–1. Additional Functions Demonstrated by Individual Testbenches

Testbench	Transmission and Reception of Data on Interface			Autorate Negotiation of CPRI Line Rate
	Antenna-Carrier	MII	AUX	
tb_altera_cpri.vhd	✓	—	✓	—
tb_altera_cpri_phy.vhd	✓	—	✓	—
tb_altera_cpri_mii.vhd	✓	✓	✓	—
tb_altera_cpri_mii_phy.vhd	✓	✓	✓	—
tb_altera_cpri_mii_noiq.vhd	—	✓	✓	—
tb_altera_cpri_mii_noiq_phy.vhd	—	✓	✓	—
tb_altera_cpri_autorate.vhd	—	—	—	✓
tb_altera_cpri_c4gx_autorate.vhd	—	—	—	✓
tb_altera_cpri_autorate_phy.vhd	—	—	—	✓

The first three testbench types are each available in two versions. One version tests an Arria II, Cyclone IV GX, or Stratix IV GX DUT, and the other version, with the **_phy** suffix, tests an Arria V or Stratix V DUT.

The **tb_altera_cpri_autorate.vhd** testbench tests a Stratix IV GX DUT and the **tb_altera_cpri_c4gx_autorate.vhd** testbench tests a Cyclone IV GX DUT. The **tb_altera_cpri_autorate_phy.vhd** testbench that is generated automatically tests a Stratix V DUT.

You can generate any **_phy** testbench for an Arria V variation, but in that case you must modify the library file names in the **.do** file to test the Arria V DUT.

Each testbench consists of a CPRI IP core and a testbench that initializes the CPRI IP core and sends the generated data to the CPRI IP core interfaces listed in Table 8–1. In the testbenches, the CPRI IP core's high-speed transceiver output is looped back to its high-speed transceiver input. The testbench module provides clocking, reset, and initialization control, and processes to write to and read from the IP core's interfaces. The initialization process requires that the testbench module write to and read from the CPRI IP core registers through its CPU interface.

Figure 8–1 shows the non-MII testbenches, **tb_altera_cpri.vhd** and **tb_altera_cpri_phy.vhd**. Figure 8–2 shows the MII testbenches, **tb_altera_cpri_mii.vhd** and **tb_altera_cpri_mii_phy.vhd**. Figure 8–3 shows the MII, no IQ interfaces testbenches, **tb_altera_cpri_mii_noiq.vhd** and **tb_altera_cpri_mii_noiq_phy.vhd**. Figure 8–4, Figure 8–5, and Figure 8–6 show the autorate negotiation testbenches, **tb_altera_cpri_autorate.vhd**, which targets a Stratix IV GX device, **tb_altera_cpri_c4gx_autorate.vhd**, which targets a Cyclone IV GX device, and **tb_altera_cpri_autorate_phy.vhd**, which targets an Arria V or Stratix V device.

Figure 8–1. CPRI IP Core Non-MII Demonstration Testbenches (tb_altera_cpri.vhd and tb_altera_cpri_phy.vhd)

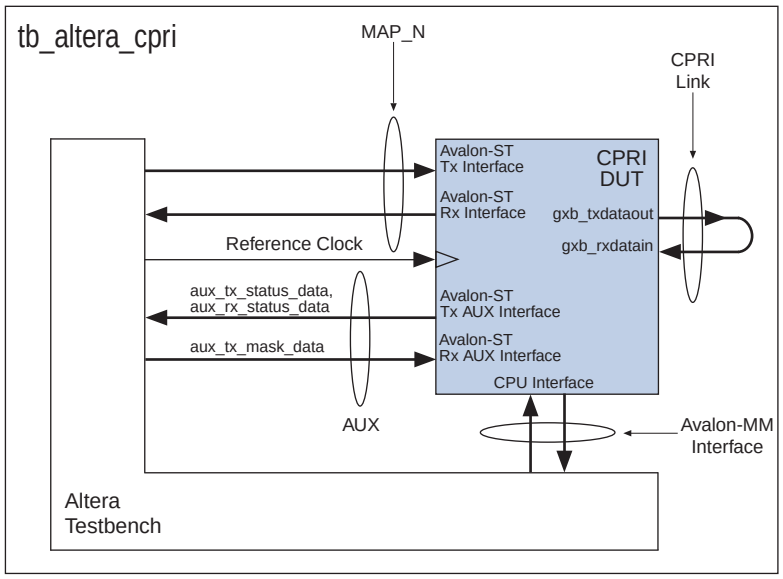


Figure 8–2. CPRI IP Core MII Demonstration Testbenches (tb_altera_cpri_mii.vhd and tb_altera_cpri_mii_phy.vhd)

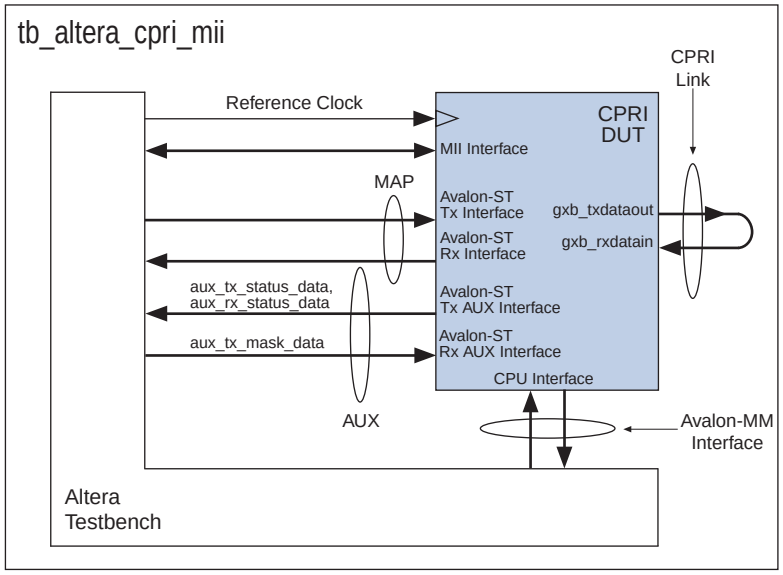


Figure 8–3. CPRI IP Core MII No IQ Demonstration Testbenches (tb_altera_cpri_mii_noiq[_phy].vhd)

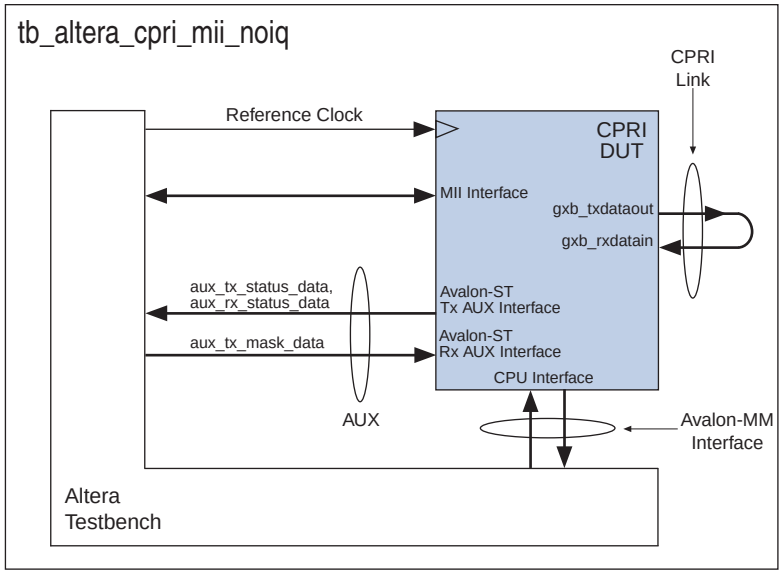


Figure 8–4. CPRI IP Core Autorate Negotiation Demonstration Testbench (tb_altera_cpri_autorate.vhd)

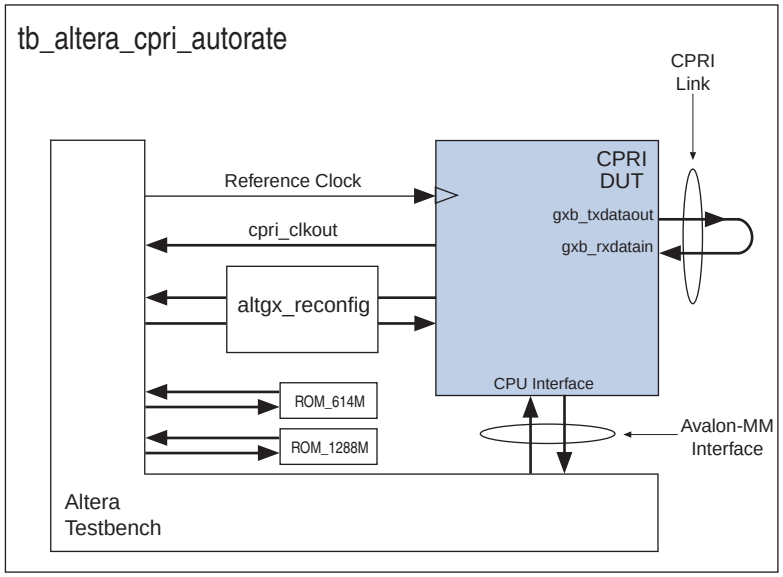


Figure 8–5. CPRI IP Core Cyclone IV GX Autorate Negotiation Testbench (tb_altera_cpri_c4gx_autorate.vhd)

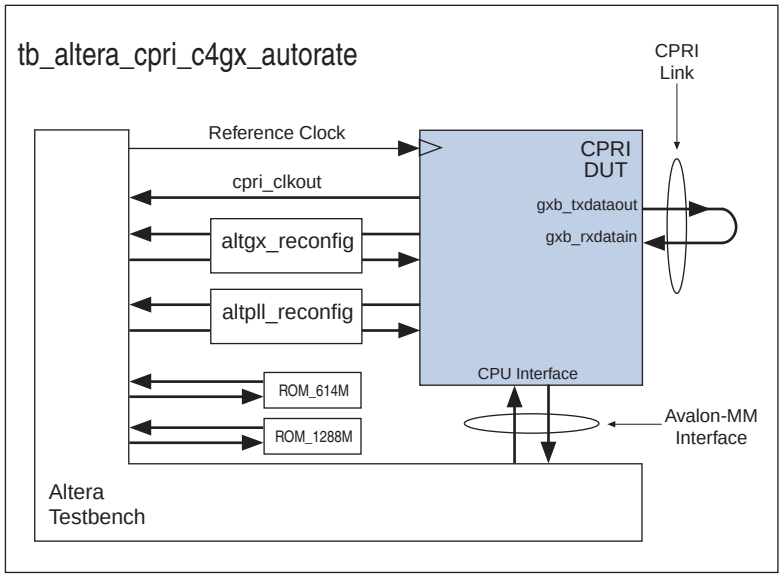
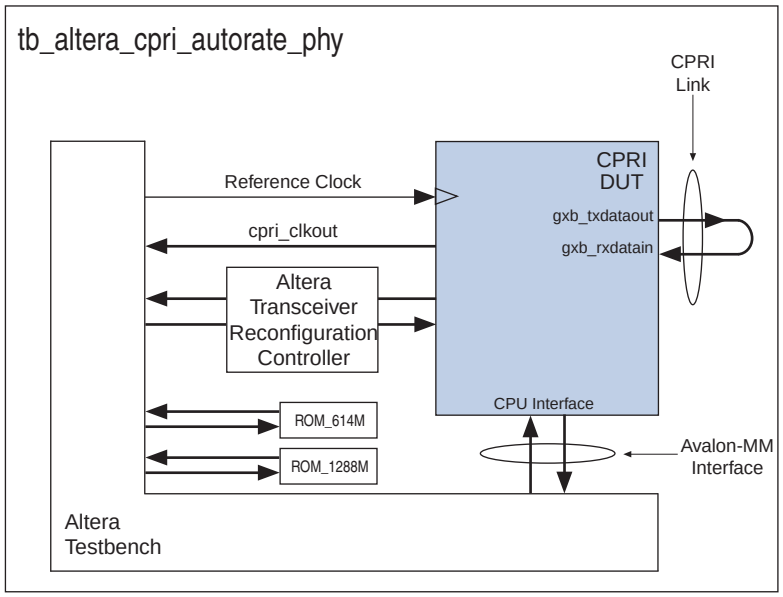


Figure 8–6. CPRI IP Core Autorate Negotiation Arria V or Stratix V Testbench (tb_altera_cpri_autorate_phy.vhd)



Test Sequence

The testbench starts by resetting the CPRI IP core. Table 8–2 lists the frequencies of the clock inputs to the CPRI IP core.

Table 8–2. Clock Frequencies for CPRI IP Core Under Test

Clock	Frequency (MHz)
gxb_refclk	61.44
cpu_clk	30.72
clk_ex_delay	30.96
mapN_tx_clk	3.84

After coming out of the reset state, the CPRI IP core starts the frame synchronization process to detect the presence of a partner and establish frame synchronization.

The **tb_altera_cpri**, **tb_altera_cpri_mii**, and **tb_altera_cpri_mii_noiq** testbenches and their **_phy** equivalents then perform the following actions:

- Sends a predetermined data sequence to the AUX interface, and checks that the data appears on the outgoing AUX interface after loopback through the CPRI link.

- Generates a sequence of 32-bit words and sends the data sequence to each antenna-carrier interface that is enabled. The **tb_altera_cpri** and **tb_altera_cpri_mii** testbenches and their **_phy** equivalents support three antenna-carrier interfaces; the **tb_altera_cpri_autorate**, **tb_altera_cpri_c4gx_autorate**, and **tb_altera_cpri_mii_noiq** testbenches and their **_phy** equivalents support no antenna-carrier interfaces.

  Each testbench with antenna-carrier interfaces enabled then checks that the data sent to the mapN interfaces appears on the outgoing antenna-carrier interface data channels, after loopback through the CPRI link.

- If relevant, sends a predetermined data sequence to the MII, and checks that the data appears as expected on the outgoing MII after loopback through the CPRI link (**tb_altera_cpri_mii** and **tb_altera_cpri_mii_noiq** testbenches and their **_phy** equivalents only).

  This test also checks the MII handling of the input error indication signal. The signal is asserted during parts of the incoming data sequence, and the expected output data reflects the correct handling of data in this case.

All testbenches perform self-checking and output the pass/fail results to your Modelsim session. In addition, each testbench includes simulator files that allow you to observe the signals in and out of the AUX interface, antenna-carrier interfaces, and MII if relevant.

Reset, Frame Synchronization, and Initialization

The reset sequence is simple—all of the reset signals for the DUT except `gxb_powerdown` and `reset_ex_delay` are asserted at the beginning of the simulation, are kept high for 500 ns, and are then deasserted. The following reset signals are asserted:

- `reset`
- `cpu_reset`
- `config_reset`
- `mapN_tx_reset` for N={1...3}
- `mapN_rx_reset` for N={1...3}

When frame synchronization completes, the value on the `cpri_rx_state` output port (bits [1:0] of the `extended_rx_status_data` bus) is 0x3 and the value on the `cpri_rx_cnt_sync` port (bits [4:2] of the `extended_rx_status_data` bus) is 0x1. Following the appearance of these values, the value of the `cpri_rx_hfn_state` output signal transitions to value 1, and then value of the `cpri_rx_bfn_state` output signal transitions to value 1. When these values appear in the waveform display, the CPRI link is up and ready to receive and send data.

Next, basic programming of the internal registers is performed in the DUT to allow CPRI communication. Table 8–3 shows the registers that are programmed in the **tb_altera_cpri** and **tb_altera_cpri_mii** (and **_phy** equivalents) DUTs. For a full description of each register, refer to Chapter 7, Software Interface.

Table 8–3. Testbench Registers

Register Address	Register Name	Description	Value
0x0008	CPRI_CONFIG	Enable CPRI control word insertion, set the CPRI MegaCore to use master clocking mode, set loop_mode to No internal loopback, and enable transmission on the CPRI link.	0x00000021
0x0104	CPRI_MAP_CNT_CONFIG	Set number of active data channels to 3 and the oversampling factor to 1.	0x00000301
0x0100	CPRI_MAP_CONFIG	Set map_mode to basic mapping scheme, set MAP transmitter and receiver synchronization mode to non-FIFO mode, and use 16-bit sample width.	0x0000000C

The autorate negotiation testbench performs autorate negotiation. Refer to Appendix B, Implementing CPRI Link Autorate Negotiation for details.

Running the Testbenches

To run the CPRI IP core testbenches, perform the following steps:

1. In the Quartus II software, create a project using the **New Project Wizard** on the File menu. Name the project cpri_top_level. If you change this name you must edit the testbench simulation **.do** file. The project targets the same device as your intended DUT. Refer to Table 8–4.
2. Generate the CPRI IP core DUT instance with the properties shown in Table 8–4. When you are prompted to generate an example design, you must turn on **Generate Example Design** and click **Generate**.

Table 8–4. MegaWizard Plug-In Manager Options for CPRI IP Core DUT (Part 1 of 2)

Parameter	Value
Device family	**tb_altera_cpri_autorate**: Stratix IV GX **tb_altera_cpri_c4gx_autorate**: Cyclone IV GX All non-PHY testbenches: Arria II, Cyclone IV GX, or Stratix IV GX All PHY testbenches: Stratix V (or Arria V (1))
Language	VHDL
File name (2)	<*working directory*>**\cpri_top_level**
Operation mode	Master (3)
Line rate	0.6144 Gbps
Enable auto-rate negotiation	**tb_altera_cpri_autorate[_phy]** and **tb_altera_cpri_c4gx_autorate**: On All other testbenches: Off

Table 8–4. MegaWizard Plug-In Manager Options for CPRI IP Core DUT (Part 2 of 2)

Parameter	Value
Transceiver reference clock frequency (Arria V and Stratix V variations only)	All PHY testbenches: 61.44 MHz
Include MAC block	**tb_altera_cpri[_phy]**, **tb_altera_cpri_autorate[_phy]**, and **tb_altera_cpri_c4gx_autorate**: On All other testbenches: Off
Include HDLC block	All testbenches: Off
Number of antenna-carrier interfaces	**tb_altera_cpri[_phy]** and **tb_altera_cpri_mii[_phy]**: 3 **tb_altera_cpri_mii_noiq[_phy]**, **tb_altera_cpri_autorate[_phy]**, and **tb_altera_cpri_c4gx_autorate**: 0
Enable MAP interface synchronization with core clock	All testbenches: Off

Notes to Table 8–4:

(1) If you generate a **_phy** testbench for an Arria V DUT, you must edit the **compile_<*variation*>.do** file to refer to the corresponding files for the Arria V device family. The **compile_<*variation*>.do** file that is generated automatically is correct for a Stratix V DUT.

(2) If you use a different path or file name, you must edit the **compile_<*variation*>.do** file to refer to the correct file for the DUT.

(3) Altera does not support an example testbench for an RE slave DUT. An RE slave in loopback configuration cannot achieve frame synchronization, because the receive CPRI protocol interface must lock on to the K28.5 character before the transmit CPRI protocol interface can begin sending K28.5 characters. Therefore, no K28.5 character is ever transmitted on the RE slave loopback CPRI link. To simulate an RE slave, you must connect the RE slave DUT to an RE master or REC CPRI IP core.

3. If you are running the **tb_altera_cpri_autorate** or **tb_altera_cpri_c4gx_autorate** testbench, you must generate the appropriate Memory Initialization Files (**.mif**) to configure the altgx_reconfig block. If you are running the **tb_altera_cpri_c4gx_autorate** testbench, the following steps also generate the appropriate **.mif** files to configure the altpll_reconfig block. To generate the files, perform the following steps:
   a. On the Assignments menu, click **Settings**.
   b. In the **Settings** dialog box, under **Category**, click **Fitter Settings**.
   c. Click **More Settings**.
   d. Turn on **Generate GXB Reconfig MIF**.
   e. Click **OK**.
   f. Click **Apply**.
   g. Click **OK**.
   h. On the Processing menu, click **Start Compilation**.

      After compilation completes, the following newly generated **.mif** files are available, depending on your target device: **reconfig_mif/cyclone4gx_**<*rate*>**_m.mif**, **cyclone4gx_**<*rate*>**_m_rx_pll1.mif**, **cyclone4gx_**<*rate*>**_m_tx_pll0.mif**, **reconfig_mif/stratix4gx_**<*rate*>**_m.mif**.
   i. In the MegaWizard Plug-In Manager, edit the existing CPRI DUT, change its data rate to 1.2288 Gbps, and regenerate. When you are prompted to generate an example design, turn off **Generate Example Design** and click **Generate**. You generate this variation only for its **.mif** files.
   j. Repeat step h. A new set of **.mif** files is generated for the new data rate.
   k. Move all of the **.mif** files from the **reconfig_mif** subdirectory to your testbench directory, <*working directory*>**/cpri_top_level_testbench/altera_cpri**.
   l. In the MegaWizard Plug-In Manager, edit the existing CPRI DUT to return it to its original data rate of 0.6144 Gbps, and regenerate. When you are prompted to generate an example design, turn off **Generate Example Design** and click **Generate**. You already generated the testbench when you generated the original DUT.

   Alternatively, for efficiency when generating the **.mif** files, you can first generate the DUT variation with CPRI data rate 1.2288 Gbps, without generating the testbench, then perform step a through step h and edit the DUT to change its data rate to 0.6144 Gbps. When you regenerate the DUT after editing, generate the testbench. Perform step j, and you have generated all the **.mif** files while minimizing the number of regeneration steps.

4. If you are running the **tb_altera_cpri_autorate_phy** testbench, full compilation automatically generates the appropriate Memory Initialization Files (**.mif**) to configure the Altera Transceiver Reconfiguration Controller. However, you must perform the full compilation at the 614.4 Mbps CPRI line rate, to generate the **.mif** for the lower line rate, before you run the testbench at the 1228.8 Mbps line rate.

   This testbench was tested on a 5SGXEA7K2F40C2 device using the 64-bit Quartus II software. Altera recommends that you compile Stratix V designs with the 64-bit Quartus II software.

   To generate the **.mif** and prepare for simulation, perform the following steps:

   a. On the Processing menu, click **Start Compilation**.

      After compilation completes, the newly generated **.mif** files **inst_xcvr_channel.mif** and **inst_xcvr_txpll0.mif** are available in the **reconfig_mif** subdirectory of the project.

   b. In the MegaWizard Plug-In Manager, edit the existing CPRI DUT, change its CPRI line rate to 1.2288 Gbps, and regenerate. When you are prompted to generate an example design, turn off **Generate Example Design** and click **Generate**.

   c. In the MegaWizard Plug-In Manager, generate an Altera Transceiver Reconfiguration Controller (**Interfaces > Transceiver PHY > Transceiver Reconfiguration Controller v12.0**) in the file **xcvr_reconfig_cpri.vhd**, with **Enable channel/PLL reconfiguration** turned on.

   d. Copy the new <*working directory*>**/xcvr_reconfig_cpri_sim** directory into <*working directory*>**/cpri_top_level_testbench/altera_cpri/**.

5. If you are using the ModelSim SE or ModelSim AE simulator, turn off simulation optimization by performing the following steps:

   a. In the ModelSim simulator, on the **Compile** menu, click **Compile Options**. The **Compiler Options** dialog box appears.

   b. Perform one of the following actions:

      i. If you are using the ModelSim SE simulator, on the **VHDL** tab and on the **Verilog & System Verilog** tab, turn off **Use vopt flow** and turn on **Disable optimizations by using -O0**.

      ii. If you are using the ModelSim AE simulator, on the **VHDL** tab and on the **Verilog & System Verilog** tab, turn on **Disable optimizations by using -O0**.

   c. Click **Apply**.

   d. Click **OK**.

6. In the ModelSim simulator, change directories to your testbench directory, <*working directory*>**/cpri_top_level_testbench/altera_cpri**.

7. If your DUT is an Arria V device, edit the file **compile[**_<*variation*>**]**_<*HDL*>**.do** to refer to the correct library file directories for an Arria V device. Refer to <*working directory*>**/cpri_top_level_sim/mentor/msim_setup.tcl** for the correct library file directory names.

8. To compile and run the appropriate testbench for the DUT you generated in step 2, using the ModelSim simulator, type the following command:

   `do compile[_<variation>]_<HDL>.do ↵`

The input to and subsequent output data from each of the AUX, map0, and MI interfaces is visible in the waveform for testbenches that have the relevant interface.

A. Initialization Sequence

This appendix describes the most basic initialization sequence for an Altera CPRI IP core.

To initialize the CPRI IP core, perform the following steps:

1. To configure the Altera FPGA with your design, download your **.sof** file to the FPGA.
2. Perform the following two actions simultaneously:
   - Perform a global CPRI IP core reset by asserting the following `reset` signals simultaneously, holding them asserted for at least three cycles of the slowest associated clock, and deasserting each as soon as possible thereafter:
     - `config_reset`
     - `cpu_reset`
     - `reset`
     - `reset_ex_delay`
     - `mapN_rx_reset`, for the appropriate values of `N`
     - `mapN_tx_reset`, for the appropriate values of `N`
   - To reset, power down, and power back up the high-speed transceiver in variations that include an ALTGX megafunction, assert the `gxb_powerdown` signal. This signal is not available in variations that target an Arria V or Stratix V device.
3. Write the value 0x21 to the `CPRI_CONFIG` register (`0x8`). This `CPRI_CONFIG` register setting enables the CPRI IP core to start sending K28.5 symbols on the CPRI link.
4. Observe the `cpri_rx_state` output signal as it transitions from value 0x0 to value 0x2 to value 0x3. When it has value 0x3, and the `cpri_rx_cnt_sync` output signal has value 0x1, the CPRI IP core CPRI receiver interface is in the HFNSYNC state. The `cpri_rx_state` output signal appears on `extended_rx_status_data[1:0]` and the `cpri_rx_cnt_sync` output signal appears on `extended_rx_status_data[4:2]`.
5. Observe the `cpri_rx_hfn_state` output signal as it transitions to value 1. When it has value 1, the hyperframe number is initialized. The `cpri_rx_hfn_state` output signal appears on `extended_rx_status_data[7]`.
6. Observe the `cpri_rx_bfn_state` output signal as it transitions to value 1. When it has value 1, the basic frame number is initialized. The `cpri_rx_bfn_state` output signal appears on `extended_rx_status_data[6]`.

The CPRI IP core can now receive and transmit data on the CPRI link, on the antenna-carrier interfaces, and on the auxiliary AUX interface.

To access the registers, the system requires an Avalon-MM master, for example a Nios II processor. The Avalon-MM master can program these registers.

B. Implementing CPRI Link Autorate Negotiation

The CPRI IP core supports autorate negotiation. This feature allows you to specify that the CPRI IP core should determine the CPRI line rate at startup dynamically, by stepping down to successively slower line rates if the low-level receiver cannot achieve frame synchronization with the current line rate. You can provide input to the low-level CPRI protocol interface receiver to implement this capability in your design, with the help of logic connected outside the CPRI IP core.

Variations that target an Arria V device support autorate negotiation to and from CPRI line rates up to 6.144 Gbps only. In these variations, you cannot modify the CPRI line rate dynamically to or from 9.8 Gbps.

If you configure your CPRI IP core for autorate negotiation, the IP core includes two output status signals and a register to collect the status information, in addition to the internal support to change CPRI line rate according to your design's input to the transceiver dynamic reconfiguration block. In Cyclone IV GX designs, the external logic must also provide line rate information to the ALTPLL_RECONFIG megafunction connected to the transceiver.

This appendix describes the steps you must follow and the external logic you must include in your design to implement CPRI line rate auto-negotiation.

Design Implementation

To use the autorate negotiation feature, you must perform the following actions:

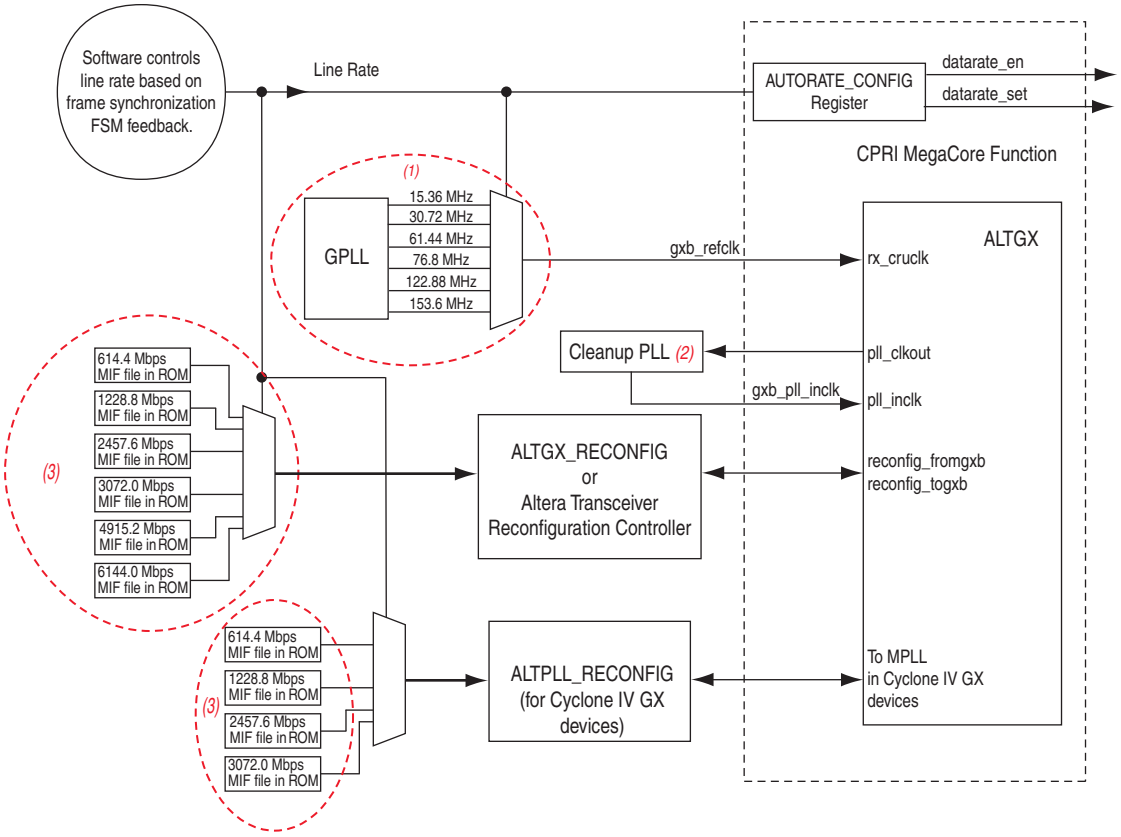
- In the CPRI parameter editor, enable autorate negotiation.
- In the CPRI parameter editor, set the transceiver to run at the highest CPRI line rate that participates in autorate negotiation in this device family.
- Include additional external data and logic in your design, such as input data to the ALTGX_RECONFIG megafunction, or Altera Transceiver Reconfiguration Controller for Arria V and Stratix V devices, for each CPRI line rate to be checked. Refer to Figure B–1 and Figure B–2.
- For Cyclone IV GX devices, you must implement logic to perform autorate negotiation by reconfiguring the transceiver directly, using the compulsory ALTGX_RECONFIG megafunction. Refer to Figure B–1 and Figure B–2.

In Cyclone IV GX devices, autorate negotiation is implemented by performing scan-chain based PLL reconfiguration of the MPLL associated with the relevant transceiver channel. Designs that target a Cyclone IV GX device therefore require an ALTPLL_RECONFIG megafunction to perform PLL reconfiguration of the MPLL.

For information about the Cyclone IV GX transceiver blocks and MPLLs, refer to volume 2 of the *Cyclone IV Device Handbook*. For information about the ALTPLL_RECONFIG megafunction, refer to the *Phase-Locked Loops Reconfiguration (ALTPLL_RECONFIG) Megafunction User Guide*.

Figure B–1 and Figure B–2 show example autorate negotiation logic block diagrams for CPRI IP cores in slave clocking mode and master clocking mode, respectively. The diagrams show all the potential CPRI line rates for an Arria II GX, Arria II GZ, Arria V, or Stratix IV GX device. However, if you remove the options for the two highest CPRI line rates, the examples are functional for Cyclone IV GX devices. If you add an option for the 9.8 Gbps CPRI line rate, the example is functional for a Stratix V device. The examples clarify the functionality provided by the CPRI IP core, and the logic and data you must configure in your design outside the CPRI IP core.

Figure B–1. Autorate Negotiation in Slave Clocking Mode



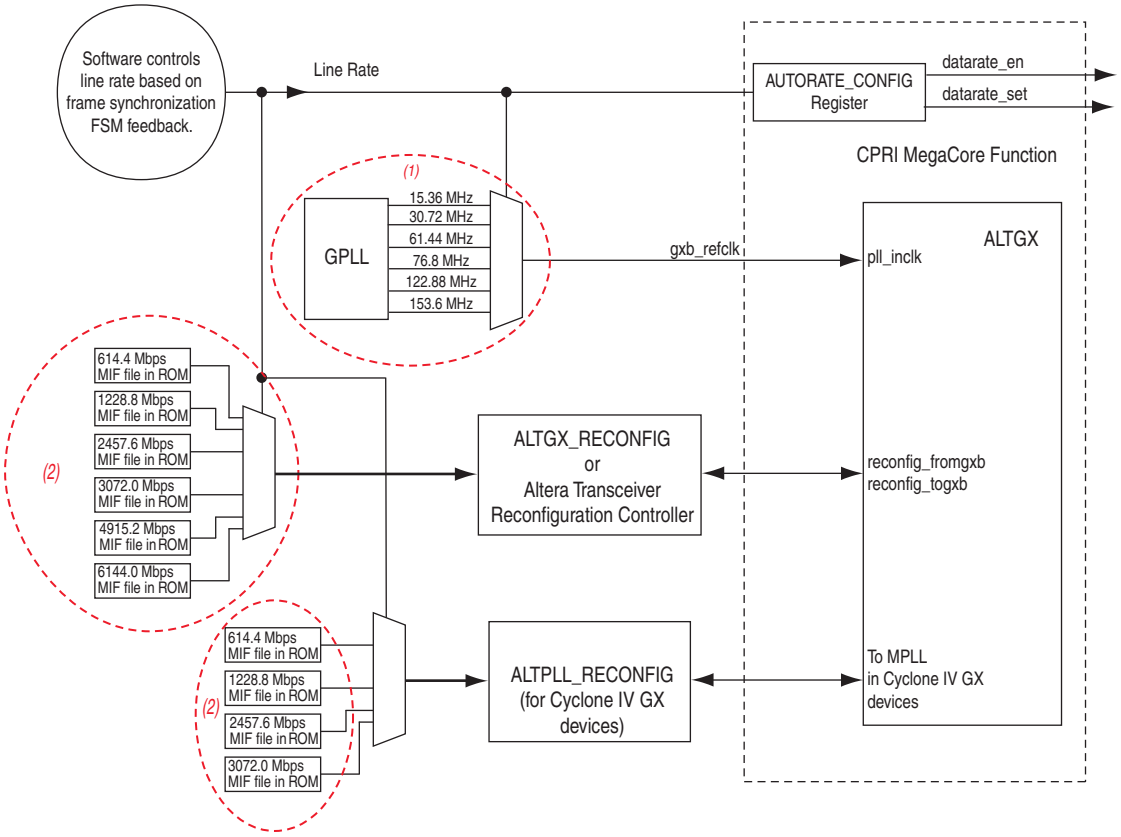
Notes for Figure B–1:

(1) Optional clock switching logic determines the value of `gxb_refclk`, depending on the desired transceiver frequency setting.

(2) You must reset the cleanup PLL configuration for different incoming and outgoing clock frequencies when the CPRI line rate changes.

(3) The number of ROMs and the rate requirements are design dependent.

Figure B–2. Autorate Negotiation in Master Clocking Mode



Notes for Figure B–2:

(1) Optional clock switching logic determines the value of `gxb_refclk`, depending on the desired transceiver frequency setting.

(2) The number of ROMs and the rate requirements are design dependent.

Configuring the CPRI IP Core for Autorate Negotiation

To ensure that the CPRI IP core implements autorate negotiation correctly, while configuring your CPRI IP core, enable autorate negotiation and set the CPRI line rate to the maximum line rate supported for autorate negotiation by the device family.

Running Autorate Negotiation

After your CPRI IP core is configured on the device, the autorate negotiation logic you configured in your design outside the CPRI IP core must perform certain steps to activate the autorate negotiation support logic in the CPRI IP core. This section describes these steps.

To start autorate negotiation in your CPRI IP core, in addition to its own initialization outside the CPRI IP core, your hardware and software must perform the following steps:

1. Confirm that the `i_datarate_en` bit of the `AUTO_RATE_CONFIG` register is set to 1. The `AUTO_RATE_CONFIG` register is described in Table 7–21 on page 7–10. You can read this value on the `datarate_en` output signal.

2. Set the logic that feeds the gxb_refclk input to the CPRI IP core to the correct value for the next CPRI line rate at which you want to try to achieve frame synchronization.
3. Configure the ALTGX_RECONFIG megafunction, or the Altera Transceiver Reconfiguration Controller for Arria V and Stratix V variations, with the **.mif** file for the desired CPRI line rate. In Arria V and Stratix V variations, alternatively, you can perform direct writes in streamer-based reconfiguration mode.
4. For a Cyclone IV GX device, configure the ALTPLL_RECONFIG megafunction with the **.mif** file for the desired CPRI line rate, by performing the following steps:
   a. Assert the write_from_rom input signal to the ALTPLL_RECONFIG megafunction. The megafunction busy output signal is asserted and remains asserted while the megafunction writes to the scan cache.
   b. After the megafunction busy output signal is deasserted, assert the megafunction reconfig signal. While PLL reconfiguration is in progress, the busy signal is again asserted.
   c. After the CPRI IP core pll_reconfig_done signal is deasserted, assert the megafunction reset_rom_address signal.
5. Set the i_datarate_set field of the AUTO_RATE_CONFIG register to the correct value for the next CPRI line rate at which you want to try to achieve frame synchronization.
6. Confirm the field is set by monitoring the datarate_set output signal.
7. Optionally, to enable confirmation of frame synchronization at the new CPRI line rate, reset the tx_enable bit of the CPRI_CONFIG register to 0.

   The frame synchronization machine shown in Figure 4–26 on page 4–50 attempts to achieve frame synchronization at the specified CPRI line rate.
8. If you reset the tx_enable bit of the CPRI_CONFIG register in step 7, after extended_rx_status_data[1:0] changes value to 0x1, set the tx_enable bit of the CPRI_CONFIG register.

   The value 0x3 on the extended_rx_status_data[1:0] signal confirms that the CPRI receiver has achieved frame synchronization.

C. Advanced AxC Mapping Modes

The advanced AxC mapping modes are implemented when map_mode has value 2'b01, 2'b10, or 2'b11 (and you specify **All** as the value for **Mapping mode(s)** in the CPRI parameter editor), or if you specify **Advanced 1**, **Advanced 2**, or **Advanced 3** as the value for **Mapping mode(s)** in the CPRI parameter editor. In these modes, different data channels can use different sample rates, and the sample rates need not be integer multiples of 3.84 MHz. However, all data channels use the same sample width.

Altera recommends that you use sample rates that are integer multiples of 3.84 MHz. However, for implementing the WiMAX protocol, Altera recommends that you use the exact WiMAX input sample rates. WiMAX applications require that your CPRI IP core implement an advanced AxC mapping mode.

The CPRI IP core supports the following advanced AxC mapping modes:

- When map_mode has the value of 2'b01 or 2'b11 (**Advanced 1** or **Advanced 3**), AxC mapping conforms to Method 1: IQ Sample Based, described in Section 4.2.7.2.5 of the CPRI V4.2 Specification.
- When map_mode has the value of 2'b10 (**Advanced 2**), AxC mapping conforms to Method 3: Backward Compatible, described in Section 4.2.7.2.7 of the CPRI V4.2 Specification.

For a list of the standards supported by the various advanced mapping modes, refer to Table 3–2 on page 3–5.

Backward Compability

The CPRI IP core supports one new advanced mapping mode in the Quartus II software 11.1 and later releases. To support the new advanced mapping mode, advanced mapping mode encodings changed in the Quartus II software 11.1 release. Table C–1 shows the correspondence between the advanced mapping mode map_mode encodings in the software 11.1 and later releases and the encodings in previous software releases. The 2'b01 encoding has a different meaning in the software 11.1 and later releases than in previous releases.

Table C–1. Advanced Mapping Mode map_mode Encodings in Software Releases

Mode	CPRI Parameter Editor Mapping mode(s) Value	map_mode Encoding	
		In Quartus II Software Releases 11.1 and 12.0	In Quartus II Software Release 11.0 and Earlier
New implementation of Method 1: IQ Sample Based	**Advanced 1**	2'b01	—
Conforms to Method 3: Backward Compatible	**Advanced 2**	2'b10	2'b10
Conforms to Method 1: IQ Sample Based	**Advanced 3**	2'b11	2'b01

All of the advanced AxC mapping modes comply with the description in Section 4.2.7.2.4 of the CPRI V4.2 Specification. Advanced mapping modes 01 and 11 comply with two different interpretations of Section 4.2.7.2.5. Advanced mapping mode 11 is available in Quartus II software releases prior to release 11.1 as advanced mapping mode 01, and the current advanced mapping mode 01 is new in the Quartus II software release 11.1.

In the Advanced 1 and Advanced 2 mapping modes, each IQ data sample is considered a different AxC container, for backward compatibility with earlier versions of the CPRI specification. However, multiple consecutive 32-bit words in the same frame may contain data samples from or for the same AxC interface. In other words, data to or from the same AxC interface may appear in consecutive timeslots, even though these IQ data samples are considered individual AxC containers. IQ data samples do not span frames. Spare bytes not assigned to an AxC container become reserved bits. These reserved bits are located at the end of the basic frame.

Advanced Mapping Mode Similarities and Differences

This section describes the similarities and differences between the different advanced mapping modes. In each advanced mapping mode, the behavior is different in the 15-bit and 16-bit modes. Figure C–1 on page C–4 illustrates an example in this section that describes the differences between the advanced mapping modes in 15-bit mode, and Figure C–2 on page C–5 illustrates an example of the supported advanced mapping modes in 16-bit mode.

In the advanced mapping modes, AxC containers are packed in the IQ data block in a flexible position (Option 2), as illustrated in Section 4.2.7.2.3 of the CPRI V4.2 Specification. Configuration tables define the mapping of AxC containers to offsets in the AxC interface timeslots.

You specify the flexible position of the start of an AxC container in its timeslot using the Rx and Tx mapping tables. You configure the Rx and Tx mapping tables through the CPU interface. You can configure one mapping table entry at a time. The table index specified in the `map_conf_index` field of the `CPRI_MAP_TBL_INDEX` register determines the Rx and Tx mapping table entries that appear in the `CPRI_MAP_TBL_RX` and `CPRI_MAP_TBL_TX` registers, respectively.The `CPRI_MAP_TBL_RX` register holds the currently configurable entry in the Rx mapping table, and the `CPRI_MAP_TBL_TX` register holds the currently configurable entry in the Tx mapping table. You must configure these tables prior to data transmission on the MAP interface, otherwise data loss may occur.

Each table entry corresponds to an IQ data sample in one AxC container block. Each table entry has an enable bit and a field in which to specify the AxC interface number for the current IQ data sample, in addition to a `position` field which specifies the starting bit position of the IQ sample in the timeslot — the current 32-bit word on the AxC interface — and a `width` field to specify the number of bits in the current data sample.

The application can specify an offset for the start of an AxC container in a timeslot; the `position` field of the table entry that corresponds to the timeslot in which that AxC container begins transmission (in the CPRI Rx direction) or appears on the data channel (in the CPRI Tx direction), holds this offset. The offset is specified in bits.

Some table entries are not available, depending on the CPRI line rate and on K. In the example illustrated in Figure C–2, the table entries 7 and 15 are not available.

In 16-bit mode in all advanced mapping modes, and in 15-bit mode in advanced mapping mode 2'b01, you can use the `width` field to specify the size of the sample that starts in the bit position indicated in the `position` field, allowing you to pack a second sample immediately following the first sample in the timeslot, or to specify a sample width larger than the timeslot. In the case of a sample that spills into the following timeslot, you must enable the following timeslot in the Rx or Tx mapping table.

In 15-bit width mode in advanced mapping modes 2'b10 and 2'b11, you must set `width` to the value of 15 (indicating a 30-bit IQ sample), and you must set `position` to specify the offset of the next available bit in the current 32-bit timeslot, because the IQ samples are packed in the timeslots with no intervening spare bits.

You can calculate the number of timeslots that correspond to a CPRI frame. Only the data bytes pass through the AxC interface; the control bytes in a CPRI frame do not pass through the AxC interface. Refer to the **Number of Bits** column in Table 4–4 on page 4–14 or Table 4–5 on page 4–14 for the number of data bits in a CPRI frame at each CPRI line data rate. The calculation depends on the presence and values of any `position` offsets, on whether the CPRI IP core is in 15-bit width mode or in 16-bit width mode, and on how remainder bytes are handled. The following discussion focuses on the cases with `position` fields all set to zero. You can increment the timeslot counts as needed to accommodate unused leading timeslot bits specified with `position` offsets.

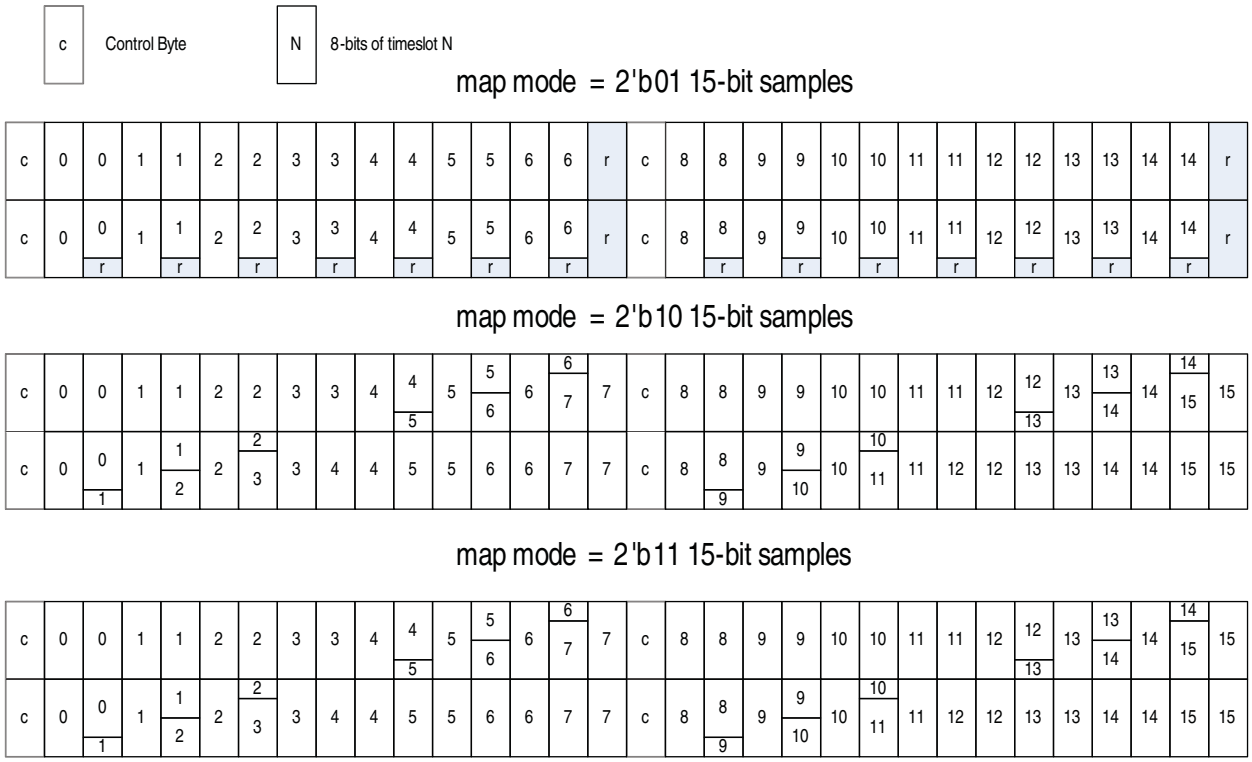
Fifteen-Bit Width Mode

In 15-bit width mode, you either pack the 30-bit data samples in the 32-bit words (in advanced mapping modes Advanced 2 (2'b10) and Advanced 3 (2'b11)), or you selectively allow gaps, specifying them with the `position` and `width` fields of the table entry (in the new Advanced 1 mapping mode (2'b01)). In 15-bit width mode, advanced AxC mapping modes 2'b10 and 2'b11 act identically, packing the data into consecutive bits. Because the number of bits in the IQ data block of every CPRI frame is a multiple of 30, packed 15-bit I- and Q-samples fill an AxC container—and one or more CPRI frames—with no spare bytes remaining. However, in the Advanced 1 mapping mode, you can specify an offset in the `position` field, potentially leaving spare bytes in the IQ data block.

Figure C–1 shows the contrast between these advanced mapping modes. In this 15-bit mode example, the CPRI data rate is 1228.8 Gbps and the value of K is two. For a CPRI IP core running at CPRI data rate 1228.8 Gbps, the number of data bits in a CPRI basic frame is 240. (Refer to Table 4–5 on page 4–14). If K (specified in the `K` field of the

CPRI_MAP_TBL_CONFIG register) has the value of two, 480 bits, or 60 bytes, of data are sent or received on the data channel.

Figure C–1. Example of Differences Between the AxC Advanced Mapping Modes in 15-Bit Mode



Note to Figure C–1:

(1) This figure uses the following conventions:
* Each column illustrates two bytes in the CPRI frame.
* The label “c” indicates a control byte.
* A numerical label indicates the index of the corresponding table entry in the Rx or Tx advanced mapping table.
* The label “r” indicates a reserved bit or set of bits. Specifically in this example, this label indicates either two bits or a full byte of reserved bits.

The example shows the mapping to timeslots, assuming a single AxC interface is active, or more concretely, the contents of the Tx or Rx advanced mapping table. In Advanced 1 mode, the Tx or Rx mapping table entries 7 and 15 are not available. In contrast, in the other two advanced mapping modes, the Tx or Rx mapping table entries 0 through 15 are valid.

Sixteen-Bit Width Mode

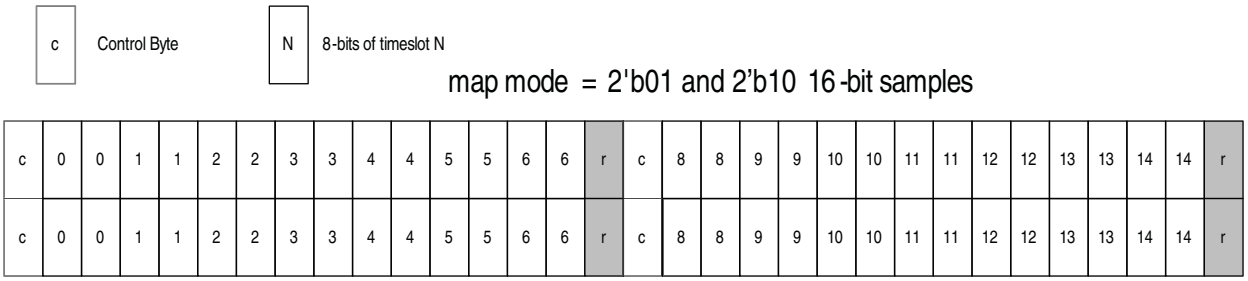
In 16-bit width mode, when map_mode has the value of 2’b01 or 2’b10, the initial 32-bit sets of data in the CPRI frame pass through the AxC interface. However, the spare bytes—bytes at the end of an IQ data block that do not fill another complete 32-bit word in the CPRI frame, or bytes at the end of a CPRI frame that do not fill another complete timeslot—are dropped in the outgoing data channel, and become reserved bits in the CPRI frame after the data arrives on the incoming data channel; these bits are expected to not contain valid AxC data in the CPRI frame.

The Altera CPRI IP core does not support the Advanced 3 mapping mode in 16-bit width mode. Advanced 3 mapping mode does not support spare bytes. Therefore, all of the data bits in a CPRI frame should theoretically pass through the AxC interface to or from the CPRI IP core. However, in the 16-bit mode, this requirement would force a single timeslot to contain information from two CPRI frames, an arrangement the Altera CPRI IP core does not support.

Figure C–2 shows the mapping between CPRI frames and the advanced mapping tables for a 16-bit mode example. In this example, the CPRI data rate is 1228.8 Gbps and the value of K is two. For a CPRI IP core running at CPRI data rate 1228.8 Gbps, the number of data bits in a CPRI basic frame is 240. (Refer to Table 4–4 on page 4–14). If K (specified in the K field of the CPRI_MAP_TBL_CONFIG register) has the value of two, 480 bits, or 60 bytes, of data are sent or received on the data channel. The figure shows how the Advanced 1 and Advanced 2 mapping modes map these 60 bytes in 16-bit mode.

In the example, the final two bytes of the data from or for each of the first and second CPRI frames are dropped or assumed reserved. The Rx or Tx mapping table entries 7 and 15 are not valid table entries, as the corresponding IQ data sample is invalid. If the CPRI IP core has a single active AxC interface, the eighth and sixteenth timeslots are empty.

Figure C–2. Example of Mapping in 16-Bit Mode



Note to Figure C–2:

(1) This figure uses the following conventions:
* Each column illustrates two bytes in the CPRI frame.
* The label “c” indicates a control byte.
* A numerical label indicates the index of the corresponding table entry in the Rx or Tx advanced mapping table.
* The label “r” indicates a byte of reserved bits

D. Delay Measurement and Calibration

This appendix describes the RX delay measurement and TX calibration features of the CPRI IP core.

Altera Delay Measurement and Calibration Features

For system configuration and correct synchronization, the CPRI IP core must meet the CPRI V4.2 Specification measurement and delay requirements. The CPRI IP core provides the following support for accurate delay measurement:

- Provides current Rx delay measurement values in the `CPRI_RX_DELAY` and `CPRI_EX_DELAY_STATUS` delay registers.
- Provides current Tx delay calibration values in the `CPRI_TX_BITSLIP` register.
- Provides current round-trip delay value in the `CPRI_ROUND_DELAY` register.
- Supports user control over delay measurement accuracy by the following methods:
  - Allows you to control the degree of delay accuracy in the status registers by programming the `CPRI_RX_DELAY_CTRL` and `CPRI_EX_DELAY_CONFIG` registers.
  - Provides an optional automatic calibration process that takes your input for the desired round-trip delay and adjusts internal delays in an attempt to match the desired value. The automatic calibration process reports its current success status in the `CPRI_AUTO_CAL` register.

The following sections describe the delay requirements and how you can use these registers to ensure that your application conforms to the CPRI V4.2 Specification delay requirements.

Delay Requirements

CPRI V4.2 Specification requirements R-17, R-18, and R-18A address jitter and frequency accuracy in the RE core clock for radio transmission. The relevant clock synchronization is performed using an external clean-up PLL that is not included in the CPRI IP core.

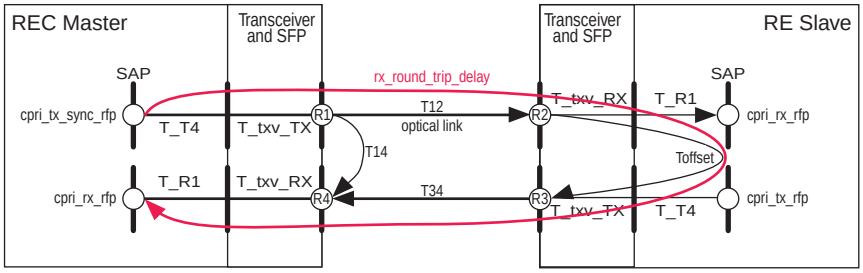
The CPRI IP core complies with CPRI V4.2 Specification requirements R-19, R-20, R-20A, R-21, and R-21A.

CPRI V4.2 Specification requirement R-20A addresses the maximum allowed delay in switching between receiving and transmitting on the AxC interface. Because the CPRI IP core provides duplex communication on the AxC interfaces, this switch requires only the programming of the relevant AxC interface Tx or Rx enable bit in the `CPRI_IQ_TX_BUF_CONTROL` or `CPRI_IQ_RX_BUF_CONTROL` register, and no delay calculation is required.

Requirement R-19 specifies that the link delay accuracy for the downlink between the synchronization master SAP and the synchronization slave SAP, excluding the cable length, be within ±8.138 ns. Requirements R-20 and R-21 extrapolate this requirement to single-hop round-trip delay accuracy. R-20 requires that the accuracy of the round-trip delay, excluding cables, be within ±16.276 ns, and R-21 requires that the round-trip cable delay measurement accuracy be within the same range. Requirement R-21A extrapolates this requirement further, to multihop round-trip delay accuracy. In calculating these delays, Altera assumes that the downlink and uplink cable delays have the same duration.

Figure D–1 shows the reference points you can use to determine the CPRI IP core delay measurements for single-hop CPRI configurations.

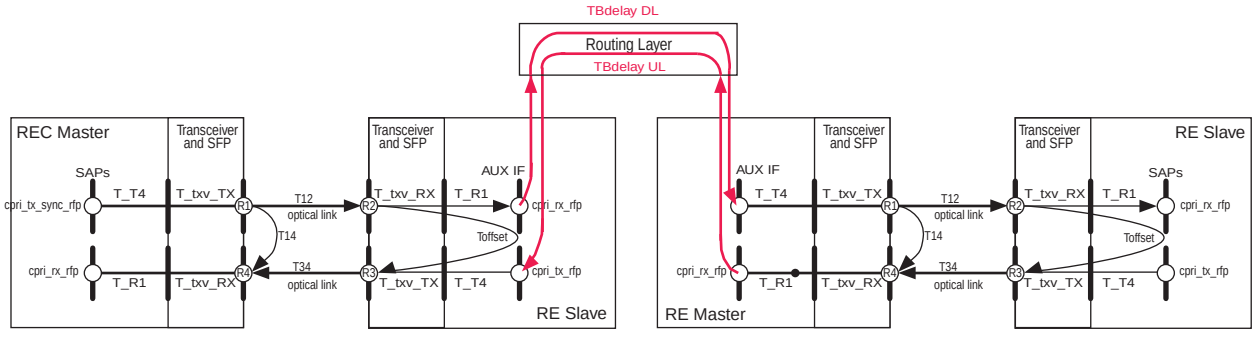
Figure D–1. Single-Hop CPRI Configuration Delay Measurement Reference Points



CPRI requirement R-21 addresses the accuracy of the round-trip cable delay, which is the sum of the T12 and T34 delays. The T12 and T34 delays are assumed to have the same duration.

Figure D–2 shows the reference points you can use to determine the CPRI IP core delay measurements for multihop CPRI configurations. The duration of TBdelay depends on your routing layer implementation.

Figure D–2. Multihop CPRI Configuration Delay Measurement Reference Points



The following sections describe the delay through the CPRI IP core on the Rx path and on the Tx path to the SAP—the AUX interface—and the deterministic values for transceiver latency and delay through the IP core. They describe the calculation of the round-trip cable delay T14, the Toffset delay, and the round-trip (SAP to SAP) delay in the single-hop and multihop cases, and describe the CPRI IP core optional round-trip delay calibration feature and how to activate it.

The "Rx Path Delay" and "Tx Path Delay" sections do not discuss the delays through the AxC blocks, because the round-trip delay calculations and the multihop configuration delay calculations do not take the AxC blocks into account. For purposes of these calculations, the relevant SAP is the AUX interface. For information about the delays through the AxC blocks, refer to "MAP Receiver Interface" on page 4–15 and "MAP Transmitter Interface" on page 4–21.

Rx Path Delay

The Rx path delay is the cumulative delay from the arrival of the first bit of a 10 ms radio frame on the CPRI Rx interface to the start of transmission of the radio frame on the AUX interface.

Rx Path Delay Components

The CPRI specification defines requirements on the path to an SAP. The CPRI IP core has one relevant SAP, the AUX interface. This section provides the information to calculate the Rx path delay to output on the AUX interface.

The delay to—but not through—the AxC blocks, that is, the delay through the MAP interface module, is the same as the delay to the AUX interface. Figure D–3 shows the Rx path delay components in all CPRI IP core variations except the variation shown in Figure D–4. Figure D–4 shows the Rx path delay components in a CPRI IP core variation with the CPRI line rate of 9.8 Gbps that targets an Arria V GT device. Both figures show the relation between the two Rx paths.

Figure D–3. Rx Path Delay to AUX Output and Through MAP Interface Block in Most CPRI IP Core Variations

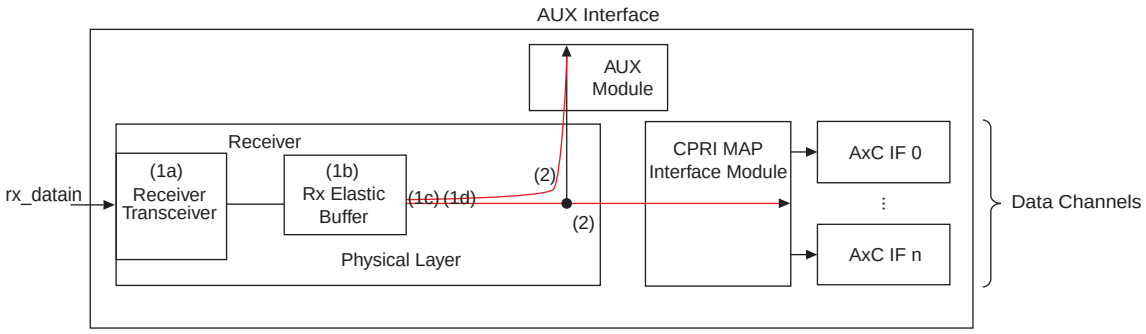
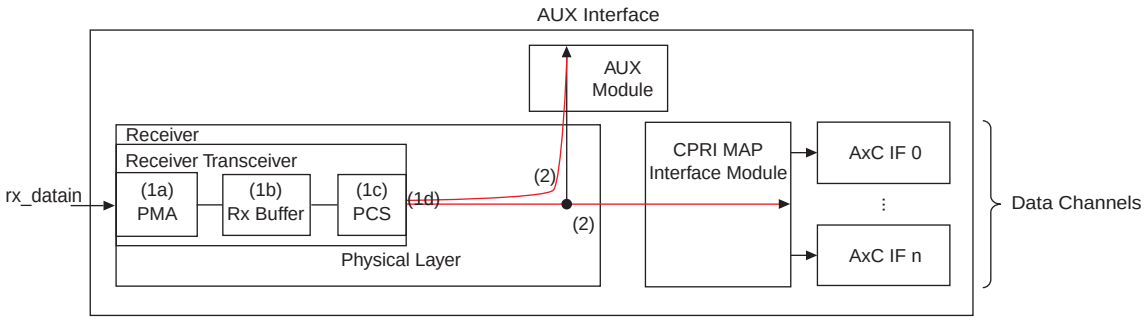


Figure D–4. Rx Path Delay to AUX Output and Through MAP Interface Block in Arria V GT 9.8 Gbps Variations



The Rx path delay to the AUX interface or through the MAP interface module in most CPRI IP core variations is the sum of the following delays:

1. The link delay is the delay between the arrival of the first bit of a 10 ms radio frame on the CPRI Rx interface and the CPRI IP core internal transmission of the radio frame pulse from the CPRI protocol interface Rx module. The link delay includes the following delays:
   a. Transceiver latency is a fixed delay through the deterministic latency path of the transceiver. Its duration depends on the device family and on the path direction (Rx or Tx). This delay includes comma alignment. Refer to "Rx Transceiver Latency" on the following pages.
   b. Delay through the clock synchronization FIFO, as well as the phase difference between the recovered receive clock and the core RE clock `cpri_clkout`. The "Extended Rx Delay Measurement" section shows how to calculate the delay in the CPRI Rx elastic buffer, which includes the phase alignment delay.
   c. Fixed two cycle byte alignment delay (two `cpri_clkout` clock cycles). The CPRI IP core now compensates for the byte alignment delay that can occur as data is shifted out of the Rx elastic buffer with a compensatory pipeline stage that ensures this delay is constant. This delay is included in the fixed component of the Rx delay listed in Table D–3 on page D–8, as described in "Fixed Rx Core Delay Component" on page D–8.
   d. Variable delay introduced by round-trip delay calibration feature. Refer to "Round-Trip Calibration Delay in Rx Path" on page D–7 and "Dynamic Pipelining for Automatic Round-Trip Delay Calibration" on page D–21.
2. Delay from the CPRI low-level receiver block to the AUX interface (or through the MAP interface block). This delay depends on the device family and CPRI data rate. This delay is `T_R1` in Figure D–1 on page D–2. Refer to "Fixed Rx Core Delay Component" on page D–8.

In the CPRI IP core variations with a CPRI line rate of 9.8 Gbps that target an Arria V GT device, the link delay (1.) includes the following delays:

   a. Fixed delay through the PMA configured with the Altera Native PHY IP core.
   b. Delay through an Rx buffer between the PMA and the PCS. The "Extended Rx Delay Measurement" section shows how to calculate this delay.
   c. Fixed delay through the PCS.
   d. Variable delay introduced by round-trip delay calibration feature. Refer to "Round-Trip Calibration Delay in Rx Path" on page D–7 and "Dynamic Pipelining for Automatic Round-Trip Delay Calibration" on page D–21. This delay component is common to all CPRI IP core variations.

The following sections describe the individual delays and how to calculate them.

Rx Transceiver Latency

In most CPRI IP core variations, the delay through the Rx transceiver is a fixed delay. In Arria V GT variations with a CPRI line rate of 9.8 Gbps, the Rx transceiver latency includes fixed delays through the PMA and PCS, and a variable delay through a buffer.

The following sections describe the Rx transceiver delay in both types of CPRI IP core variations.

Rx Transceiver Latency in Most CPRI IP Core Variations

The Altera high-speed transceiver is implemented using the deterministic latency protocol, which ensures that delays in comma alignment and in byte alignment within the transceiver are consistent.

Table D–1 shows the fixed latency through the transceiver in the receive side of the CPRI IP core. These values correspond to `T_txv_RX` in Figure D–1.

Table D–1. Fixed Latency T_txv_RX Through Receiver Transceiver

Latency Through Transceiver in cpri_clkout Clock Cycles				
Arria II GX or Cyclone IV GX Device	Arria II GZ or Stratix IV GX Device		Arria V or Stratix V Device	
	Data Rate 614.4 Mbps	Data Rate > 614.4 Mbps	Data Rate 614.4 Mbps	Data Rate > 614.4 Mbps
4.65	10.4	7.2	13.3	10.65

The clean-up PLL shown in Figure 4–2 on page 4–5 uses the recovered clock as input to the PLL that generates the `gxb_pll_inclk` signal, to ensure frequency match. To preserve the `T_txv_RX` latency in Table D–1, you must ensure that the reference clock to the clean-up PLL contains no asynchronous dividers.

Rx Transceiver Latency in Arria V GT Variations at CPRI Line Rate 9.8 Gbps

In CPRI IP core variations with a CPRI line rate of 9.8 Gbps that target an Arria V GT device, the Altera high-speed transceiver is configured with a hard PMA and a soft PCS and a buffer between them.

The fixed delay through the PMA is 3.075 `cpri_clkout` cycles.

The "Extended Rx Delay Measurement" section shows how to calculate the variable delay through the Rx buffer between the PMA and the PCS.

The fixed delay through the PCS is 19 `cpri_clkout` cycles.

The clean-up PLL shown in Figure 4–4 on page 4–7 uses the recovered clock as input to the PLL that generates the `gxb_pll_inclk`, `usr_clk`, and `usr_pma_clk` signals, to ensure frequency match. To preserve the latencies listed in this section, you must ensure that the reference clock to the clean-up PLL contains no asynchronous dividers.

Extended Rx Delay Measurement

The second component of the link delay is the delay through the CPRI Receive buffer. The latency of the CPRI Receive buffer depends on the number of 32-bit words currently stored in the buffer, and the phase difference between the recovered receive clock, which is used to write data to the buffer, and the system clock `cpri_clkout`, which is used to read data from the buffer. The CPRI IP core uses a dedicated clock, `clk_ex_delay`, to measure the Rx buffer delay to your desired precision. The `rx_ex_delay` field of the `CPRI_EX_DELAY_CONFIG` register contains the value N, such

that N clock periods of the clk_ex_delay clock are equal to some whole number M of cpri_clkout periods. For example, N may be a multiple of M, or the M/N frequency ratio may be slightly greater than 1, such as 64/63 or 128/127. The application layer specifies N to ensure the accuracy your application requires. The accuracy of the Rx buffer delay measurement is N/*least_common_multiple*(N,M) cpri_clkout periods.

The rx_buf_delay field of the CPRI_RX_DELAY register indicates the number of 32-bit words currently in the Rx buffer. After you program the rx_ex_delay field of the CPRI_EX_DELAY_CONFIG register with the value of N, the rx_ex_buf_delay field of the CPRI_EX_DELAY_STATUS register holds the current measured delay through the Rx buffer. The unit of measurement is cpri_clkout periods. The rx_ex_buf_delay_valid field indicates that a new measurement has been written to the rx_ex_buf_delay field since the previous register read. The following sections explain how you set and use these register values to derive the extended Rx delay measurement information.

M/N Ratio Selection

As your selected M/N ratio approaches 1, the accuracy provided by the use of the clk_ex_delay clock increases. Table D–2 shows some example M/N ratios and the resolutions they provide, for a CPRI IP core that runs at data rate 3072 Mbps and targets a Stratix IV GX device.

Table D–2. Resolution as a Function of M/N Ratio at 3072 Mbps on a Stratix IV GX Device

M	N	cpri_clkout Period (1)	clk_ex_delay Period (2)	Resolution
128	127	13.02 ns (1/76.80 MHz)	13.12 ns	±100 ps
64	63		13.22 ns	±200 ps
1	4		3.25 ns	±3.25 ns

Notes to Table D–2:

(1) Table 4–2 on page 4–8 lists the cpri_clkout frequency for each CPRI data rate and device family.

(2) "CPRI Receive Buffer Delay Calculation Example" shows you how to calculate the clk_ex_delay clock period for a given M, N, and cpri_clkout period.

Arria V GT Variations with CPRI Line Rate 9.8 Gbps

CPRI IP core variations with a CPRI line rate of 9.8 Gbps that target an Arria V GT device do not have an Rx elastic buffer outside the transceiver. In these variations, the same calculation applies to the Rx buffer inside the transceiver, instead.

CPRI Receive Buffer Delay Calculation Example

This section walks you through an example that shows you how to calculate the frequency at which to run clk_ex_delay, and how to program and use the registers to determine the delay through the CPRI Receive buffer.

For example, assume your CPRI IP core runs at data rate 3072 Mbps. In this case, Table 4–2 on page 4–8 shows that the cpri_clkout frequency is 76.80 MHz, so a cpri_clkout cycle is 1/(76.80 MHz).

Refer to Table D–2 for the accuracy resolution provided by some sample M/N ratios. If your accuracy resolution requirements are satisfied by an M/N ratio of 128/127, perform the following steps:

1. Program the value N=127 in the rx_ex_delay field of the CPRI_EX_DELAY_CONFIG register at offset 0x3C (Table 7–19 on page 7–9).

2. Perform the following calculation to determine the clk_ex_delay frequency that supports your desired accuracy resolution:

   clk_ex_delay period = (M/N) cpri_clkout period
   = (128/127) (1/(76.80 MHz))
   = (128/127)(13.02083 ns)
   = 13.123356 ns

   Based on this calculation, the frequency of clk_ex_delay is

   1/(13.123356 ns) = 76.20 MHz

   The following steps assume that you run clk_ex_delay at this frequency.

3. Read the value of the CPRI_EX_DELAY_STATUS register at offset 0x40 (Table 7–20 on page 7–9).

   If the rx_ex_buf_delay_valid field of the register is set to 1, the value in the rx_ex_buf_delay field has been updated, and you can use it in the following calculations. For this example, assume the value read from the rx_ex_buf_delay field is 0x107D, which is decimal 4221.

4. Perform the following calculation to determine the delay through the Rx elastic buffer:

   Delay through Rx elastic buffer = (rx_ex_buf_delay × cpri_clkout period) / N
   = (4221 × 13.02083 ns) / 127
   = 432.7632 ns

   This delay comprises (432.7632 ns / 13.02083 ns) = 33.236 cpri_clkout clock cycles.

   These numbers provide you the result for this particular example. For illustration, the preceding calculation shows the result in nanoseconds. You can derive the result in cpri_clkout clock cycles by dividing the preceding result by the cpri_clkout clock period. Alternatively, you can calculate the number of cpri_clkout clock cycles of delay through the Rx elastic buffer directly, as rx_ex_buf_delay / N.

Round-Trip Calibration Delay in Rx Path

The new dynamic pipelining feature for round-trip delay calibration introduces a delay in the Rx path in an RE slave. In CPRI IP core variations other than the Arria V GT 9.8 Gbps variations, this delay is introduced to the Rx path immediately following the Rx elastic buffer. In the Arria V GT 9.8 Gbps variations, this delay is introduced in the CPRI Rx block. The feature introduces the new delay to maintain a round-trip delay measurement as close as possible to the anticipated round-trip delay you provide to the CPRI IP core. The CPRI_AUTO_CAL register holds the anticipated delay that you program, an enable bit you turn on to activate the feature, and a status field in which the CPRI IP core reports its relative success in maintaining the round-trip delay you requested.

The register also contains a field, cal_pointer, that the CPRI IP core updates dynamically with the current number of cpri_clkout cycles of delay that this feature adds. You must include this register field value in your Rx path delay calculation. If the enable bit of the CPRI_AUTO_CAL register has the value of 0, the delay is 3 cpri_clkout cycles.

For more information about this feature, refer to "Dynamic Pipelining for Automatic Round-Trip Delay Calibration" on page D–21 and to Table 7–29 on page 7–13.

Fixed Rx Core Delay Component

In the Rx path, the delay from the CPRI low-level receiver block to the AUX interface or through the MAP interface block is fixed. This delay depends on the device family and CPRI data rate. This delay is labeled T_R1 in Figure D–1 on page D–2

This delay includes a fixed two `cpri_clkout` cycle delay that compensates for a potential byte alignment delay that can occur as data is shifted out of the Rx elastic buffer. The fixed two cycle delay is labeled (1c) in Figure D–3 on page D–3. The variable delay appears in the `rx_byte_delay` field of the `CPRI_RX_DELAY` register — when the value in `rx_byte_delay` is non-zero, a byte alignment delay of one `cpri_clkout` cycle occurs in the Rx path. However, a compensatory pipeline stage ensures this delay is constant, at one `cpri_clkout` cycle. The value in this register field is no longer relevant to the Rx path delay, and the constant delay is included in the fixed component of the Rx delay.

Table D–3 shows the fixed delays between the low-level receiver block and the AUX interface.

Table D–3. Fixed Latency T_R1 From Low-Level Receiver to AUX Interface in cpri_clkout Cycles

Data Rate 614.4 Mbps	Data Rate > 614.4 Mbps		Arria V GT Device at Data Rate 9.8304 Gbps
All Device Families	Arria II GX or Cyclone IV GX Device	Arria II GZ, Arria V, Stratix IV GX, or Stratix V Device	
4.25	4.5	5	4

Rx Path Delay to AUX Output: Calculation Example

This section shows you how to calculate the Rx path delay to the AUX output, based on the example shown in "CPRI Receive Buffer Delay Calculation Example" on page D–6. This example walks through the calculation for the case of a CPRI IP core that runs at CPRI data rate 3072 Mbps and targets an Arria II GX device. The `cal_en` field of the `CPRI_AUTO_CAL` register has the value of 0.

To calculate the Rx path delay, perform the following steps:

1. Consult Table D–1 on page D–5 for the correct value of T_txv_RX for your device family. For the example, the table yields T_txv_RX = 4.65 `cpri_clkout` clock cycles.
2. Calculate the latency through the Rx Receive buffer, including phase alignment, by following the steps in "CPRI Receive Buffer Delay Calculation Example" on page D–6 for your CPRI IP core instance. For the example, the calculations shown in "CPRI Receive Buffer Delay Calculation Example" yield a delay through the Rx Receive buffer of 33.236 `cpri_clkout` clock cycles.
3. Read the value of the `cal_pointer` field of the `CPRI_AUTO_CAL` register. In this case, the value in this field is 3. This value is consistent with the fact that the `cal_en` field of the `CPRI_AUTO_CAL` register has the value of 0.

4. Consult Table D–3 on page D–8 to determine the delay through the CPRI IP core to the AUX interface. For the example, the duration of this delay is 4.5 `cpri_clkout` clock cycles.
5. Calculate the full Rx path delay to the AUX interface by adding the values you derived in step 1 through step 4. For the example, calculate the Rx path delay as follows:

   Rx path delay = T_txv_RX + <*delay through Rx Receive buffer*>
   + <`cal_pointer` *value*> + <*delay to AUX IF*>
   = 4.65 + 33.236 + 3 + 4.5 `cpri_clkout` clock cycles
   = 45.386 `cpri_clkout` clock cycles

Tx Path Delay

The Tx path delay is the cumulative delay from the arrival of the first bit of a 10 ms radio frame on the CPRI AUX interface to the start of transmission of this data on the CPRI link. This section provides the information to calculate the Tx path delay.

The delay through the MAP interface module to the CPRI link is the same as the delay from the AUX interface. Figure D–5 shows the Tx path delay components in all CPRI IP core variations except the variation shown in Figure D–6. Figure D–6 shows the Tx path delay components in a CPRI IP core variation with the CPRI line rate of 9.8 Gbps that targets an Arria V GT device. Both figures show the relation between the two Tx paths.

Figure D–5. Tx Path Delay from AUX Interface or Through MAP Interface Block to CPRI Link in Most Variations

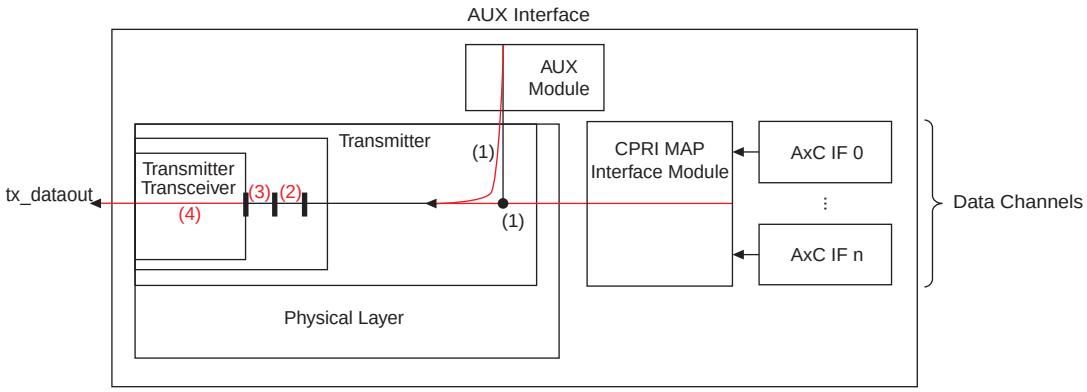
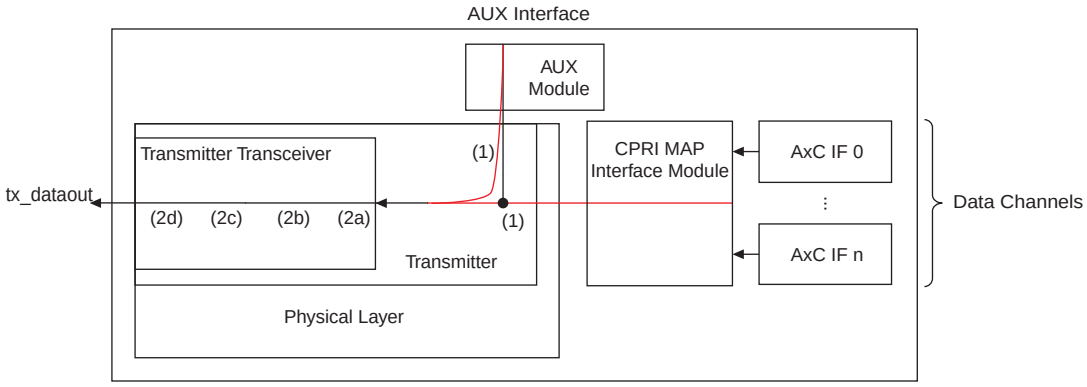


Figure D–6. Tx Path Delay to CPRI Link in Arria V GT 9.8 Gbps Variations



In the CPRI IP core the delay from the AUX interface has a fixed component and a variable component. The variable component results from the Tx elastic buffer and the Tx bitslip delay compensation feature.

The Tx path delay from the AUX interface in most CPRI IP core variations comprises the following delays:

1. Fixed delay from the AUX interface through the CPRI low-level transmitter to the Tx elastic buffer. This delay depends on the device family and CPRI data rate. This delay is `T_T4` in Figure D–1 on page D–2 and in Table D–4 on page D–11. Refer to "Fixed Tx Core Delay Component" on page D–11.
2. Variable delay through the Tx elastic buffer, as well as the phase difference between the core clock and the transceiver tx_clkout clock. The "Extended Tx Delay Measurement" section shows how to calculate the delay in the CPRI Tx elastic buffer, which includes the phase alignment delay.
3. Variable Tx bitslip delay in CPRI RE slaves. Refer to "Tx Bitslip Delay" on page D–12.
4. Link delay through the transceiver. This delay is `T_txv_TX` in Table D–5 on page D–13.

In the CPRI IP core variations with a CPRI line rate of 9.8 Gbps that target an Arria V GT device, the Tx path delay from the AUX interface comprises the following delays:

1. Fixed delay from the AUX interface through the CPRI low-level transmitter to the transceiver PCS. Refer to "Fixed Tx Core Delay Component" on page D–11.

2. Delay through the transceiver. This delay has the following components:
   a. Variable Tx bitslip delay in CPRI RE slaves. Refer to "Tx Bitslip Delay" on page D–12.
   b. Fixed delay through the soft PCS. Refer to "Tx Transceiver Latency in Arria V GT Variations at CPRI Line Rate 9.8 Gbps" on page D–13.
   c. Variable delay through the Tx buffer between the soft PCS and the PMA. The "Extended Tx Delay Measurement" section shows how to calculate this delay.
   d. Fixed delay through the PMA configured with the Altera Native PHY IP core. Refer to "Tx Transceiver Latency in Arria V GT Variations at CPRI Line Rate 9.8 Gbps" on page D–13.

Fixed Tx Core Delay Component

In the Tx path in CPRI IP core variations other than the Arria V GT 9.8 Gbps variations, the delay from the AUX interface to the Tx elastic buffer is fixed. This delay depends on the device family and CPRI data rate. This delay is labeled T_T4 in Figure D–1 on page D–2.

Table D–4 shows the fixed delay between the AUX interface and the Tx elastic buffer in these variations.

Table D–4. Fixed Latency T_T4 From AUX Interface to Tx Elastic Buffer in cpri_clkout Cycles

Data Rate 614.4 Mbps		Data Rate > 614.4 Mbps			
Arria II GX Device (ARN En/Dis) (1)	All Other Device Families	Arria II GX Device (ARN En/Dis) (1)	Arria II GZ or Stratix IV GX Device (1)	Arria V or Stratix V Device	Cyclone IV GX Device
3.75/3.5	3.75	5.5/5.0	6	6	4.5

Note to Table D–4:

(1) In Arria II GX devices, the latency depends on whether the CPRI IP core is configured with autorate negotiation enabled. The latency is presented as A/B, where A is the latency when autorate negotiation is enabled, and B is the latency when autorate negotiation is disabled.

In CPRI IP core variations with a CPRI line rate of 9.8 Gbps that target an Arria V GT device, the fixed Tx core delay component extends to the transceiver. The duration of this delay in this variation is 6 `cpri_clkout` cycles.

Extended Tx Delay Measurement

The latency of the Tx elastic buffer depends on the number of 32-bit words currently stored in the buffer, and the phase difference between the system clock `cpri_clkout`, which is used to write data to the buffer, and the transceiver clock `tx_clkout`, which is used to read data from the buffer.

The calculation of the extended Tx delay is identical to the description and example of extended Rx delay measurement in "Extended Rx Delay Measurement" on page D–5, with the substitution of `tx` for `rx` in all the register field names.

As for the extended Rx delay measurement, this same calculation applies to the Tx buffer inside the transceiver in CPRI IP core variations with a CPRI line rate of 9.8 Gbps that target an Arria V GT device.

Tx Bitslip Delay

To increase the consistency of the round-trip delay, the CPRI RE slave introduces a variable bitslip on the Tx path to complement the variability in the word aligner on the Rx path. The word aligner is encapsulated in the transceiver block. The compensation is a small number of bits — below the threshold to affect the number of `cpri_clkout` cycles—and introduces delay variability well within the R-19 requirement. The Rx bitslip value being compensated remains constant until frame resynchronization.

The CPRI IP core reports the Rx bitslip through the word aligner in the `rx_bitslipboundaryselectout` field of the `CPRI_TX_BITSLIP` register, and compensates for this variable delay by adding a bitslip in the Tx path. The current size of this bitslip in bits is available in the `tx_bitslipboundaryselect` field of the `CPRI_TX_BITSLIP` register. When you leave the `tx_bitslip_en` field at its default value of 0, this feature is active. The `tx_bitslipboundaryselect` value complements the `rx_bitslipboundaryselectout` value to ensure that the round-trip delay through the CPRI RE slave maintains acceptable proximity to a certain target value. The CPRI IP core calibrates this target value internally and adjusts `tx_bitslipboundaryselect` in response to changes in the `rx_bitslipboundaryselectout` value.

The Tx bitslip feature ensures stability in the round-trip delay through a CPRI RE core, but introduces a variable component in each of the Tx and Rx paths when considered independently. In CPRI IP cores in master clocking mode, the `tx_bitslipboundaryselect` field has the constant value of 0.

If you set the value of the `tx_bitslip_en` field to 1, you can override the current `tx_bitslipboundaryselect` value to control the Tx bitslip delay manually. Altera does not recommend implementing the manual override.

In CPRI IP core variations that target an Arria V or Stratix V device, the Tx bitslip functionality is included in the Altera Deterministic Latency PHY IP core that is generated with the CPRI IP core. These variations include the `CPRI_TX_BITSLIP` register to support manual override of the Tx bitslip delay.

Altera does not recommend implementing the manual override for the Tx bitslip.

Tx Transceiver Latency

In most CPRI IP core variations, the delay through the Tx transceiver is a fixed delay. In Arria V GT variations with a CPRI line rate of 9.8 Gbps, the Tx transceiver latency includes fixed delays through the PMA and PCS, and a variable delay through a buffer.

The following sections describe the Tx transceiver delay in both types of CPRI IP core variations.

Tx Transceiver Latency in Most CPRI IP Core Variations

The Altera high-speed transceiver is implemented using the deterministic latency protocol, which ensures that delays in comma alignment (word alignment) and in byte alignment within the transceiver are consistent.

Table D–5 shows the fixed latency through the transceiver in the transmit side of the CPRI IP core. These values correspond to T_txv_TX in Figure D–1 on page D–2.

Table D–5. Fixed Latency T_txv_TX Through Transmitter Transceiver

Latency Through Transceiver in cpri_clkout Clock Cycles				
Arria II GX or Cyclone IV GX Device	Arria II GZ or Stratix IV GX Device		Arria V or Stratix V Device	
	Data Rate 614.4 Mbps	Data Rate > 614.4 Mbps	Data Rate 614.4 Mbps	Data Rate > 614.4 Mbps
3.35	7.4	3.6	5.3	4.15

Tx Transceiver Latency in Arria V GT Variations at CPRI Line Rate 9.8 Gbps

In CPRI IP core variations with a CPRI line rate of 9.8 Gbps that target an Arria V GT device, the Altera high-speed transceiver is configured with a soft PCS and a hard PMA and a buffer between them.

In these variations, the delay through the Tx transceiver has the following components:

- The fixed delay through the PCS is 8 cpri_clkout cycles.
- The "Extended Rx Delay Measurement" section shows how to calculate the variable delay through the Tx buffer between the PCS and the PMA.
- The fixed delay through the PMA is 4.075 cpri_clkout cycles.

Therefore, the Tx transceiver latency in these variations is 12.075 cpri_clkout cycles plus the variable extended Rx delay.

T14, Toffset, Round-Trip Delay, and Round-Trip Cable Delay Calculations

The round-trip cable delay is the delay from the REC end of the CPRI downlink to the REC end of the CPRI uplink. This round-trip cable delay is shown as T14 in Figure D–1 on page D–2. The CPRI V4.2 Specification requirement R-21 requires that we ensure an accuracy of ±16.276 ns in the measurement of the round-trip cable delay in a single-hop configuration.

In contrast, the rx_round_trip_delay field of the CPRI_ROUND_DELAY register records the total round-trip delay from the start of the internal transmit radio frame in the REC to the start of the internal receive radio frame in the REC, that is, from SAP to SAP. The register value is only available in CPRI REC and RE masters.

You must subtract the internal delays through the RE or REC master from this register value to determine the value of T14, the round-trip cable delay, for the current hop.

CPRI V4.2 Specification requirements R-20 and R-21 address the round-trip delay. Requirement R-20 addresses the measurement without including the cable delay, and requirement R-21 includes the cable delay. Both requirements state that the variation must be no more than ±16.276 ns.

The CPRI IP core supports two approaches to these requirements. In the first approach, you perform calculations based on register values to determine the current delay, and check periodically to confirm that the variation in measurements over time is small enough that the requirements are met. Although extended Rx and Tx delay measurements and the Tx bitslip feature compensate for voltage and temperature variations, the fixed delays do not.

In the second approach, you activate the new dynamic pipelining feature to perform round-trip delay calibration. This feature enables the CPRI IP core to compensate dynamically for variations from a predetermined round-trip delay value that you select.

The following sections describe these two approaches.

Because the CPRI REC master and the CPRI RE slave might be on different devices, the following formulas specify the source CPRI IP core (REC or RE) for the delays in each calculation.

Round-Trip and Cable Delay Calculations for a Single-Hop Configuration

The `rx_round_trip_delay` field of the `CPRI_ROUND_DELAY` register records the delay between the outgoing `cpri_tx_rfp` signal and the outgoing `cpri_rx_rfp` signal. The `cpri_tx_rfp` signal is bit [0] of the `aux_tx_status_data` output signal bus, asserted in response to the assertion of the incoming signal `cpri_tx_sync_rfp`, which is bit [64] of the `aux_tx_mask_data` input signal, or in response to the 10 ms radio frame start based on the internal frame count in the CPRI transmitter interface. The `cpri_rx_rfp` signal is bit [75] of the `aux_rx_status_data` output signal bus, asserted in response to the start of the 10 ms radio frame on the CPRI receiver interface. In a single-hop system, shown in Figure D–1 on page D–2, the round-trip cable delay `T14` has the following components:

- `T12`—the delay from CPRI REC to CPRI RE
- The sum of the Rx and Tx path delays in the CPRI RE
- One cycle of delay for the internal loopback on the SAP in the RE slave (loopback path (3) in Figure 5–1 on page 5–1)
- `T34`—the delay from CPRI RE to CPRI REC

However, the CPRI IP core does not provide the values of `T12` and `T34`. Instead, use the following formula to calculate the round-trip cable delay `T14` in `cpri_clkout` cycles:

`T14 = rx_round_trip_delay` – <*REC Rx path delay*> – <*REC Tx path delay*>

where

- `rx_round_trip_delay` is the value in the `CPRI_ROUND_DELAY` register at offset 0x38 (Table 7–18 on page 7–9)
- <*REC Rx path delay*> is the Rx path delay, described in "Rx Path Delay" on page D–3, for the values in the CPRI REC master
- <*REC Tx path delay*> is the Tx path delay, described in "Tx Path Delay" on page D–9, for the values in the CPRI REC master

Use the following formula to calculate the Toffset delay:

Toffset = <*RE Rx path delay*> + <*RE Tx path delay*> + <*loopback delay*>,
for the path delay values in the RE slave

The formula to calculate the round-trip cable delay in a single-hop system is

Round-trip cable delay = T14 – Toffset

Tx Bitslip Delay in the Round-Trip Delay Calculation

The Tx bitslip delay that a CPRI RE slave adds to the delay through the transceiver transmitter compensates for the word aligner bitslip delay in the transceiver receiver. The total of these two bit values should be added to a detailed round-trip delay calculation. However, the total of these two bit values does not reach the duration of a single cpri_clkout cycle, nor does it reach the threshold of the CPRI specification R-20 and R-21 requirements. The bitslip delay is noticeable only with an oscilloscope.

Refer to "Tx Bitslip Delay" on page D–12 for the details of this feature.

Single-Hop Round-Trip and Cable Delay Calculation Examples

This section shows you how to calculate the round-trip cable delay in your system. The CPRI_ROUND_DELAY register value and the Rx and Tx elastic buffer delays in Example 1 are derived from hardware. The CPRI_ROUND_DELAY register value and the Rx and Tx elastic buffer delays in the other three examples are not derived from hardware.

Round-Trip and Cable Delay Calculation Example 1: Two Stratix IV GX Devices

The example walks through the calculation for the case of two link partner CPRI IP cores configured on Stratix IV GX devices, in a single-hop configuration, running at CPRI data rate 6.144 Gbps. In both devices, the cal_en field of the CPRI_AUTO_CAL register has the value of 0.

To calculate the round-trip cable delay in this system, perform the following steps:

1. Read the value in the rx_round_trip_delay field of the CPRI_ROUND_DELAY register (at register offset 0x38) of the REC master. For the example, the value is 0x6E, which is decimal 110.
2. For each of the REC master and the RE slave, read the value in the rx_ex_buf_delay field of the CPRI_EX_DELAY_STATUS register (at register offset 0x40) and the value in the rx_ex_delay field of the CPRI_EX_DELAY_CONFIG register. Read the rx_ex_buf_delay field only after the rx_ex_buf_delay_valid bit in the register is high.
3. For each of the REC master and the RE slave, divide the value in the rx_ex_buf_delay register field by the value in the rx_ex_delay register field. The result is the current Rx elastic buffer delay in cpri_clkout cycles. In this example, the Rx elastic buffer delay in the REC master is 33.5 cpri_clkout cycles, and the Rx elastic buffer delay in the RE slave is 10.5 cpri_clkout cycles.

4. Calculate the Rx path delay through the RE slave, by following the steps in "Rx Path Delay to AUX Output: Calculation Example" on page D–8. According to Table D–1 on page D–5, the correct value of T_txv_RX is 7.2 cpri_clkout cycles. According to Table D–3 on page D–8, the correct value of T_R1 is 5 cpri_clkout cycles. The Rx buffer delay is 10.5 cpri_clkout cycles, and the cal_pointer register field value is 3, yielding a total delay of 25.7 cpri_clkout cycles.

   25.7 = <*fixed transceiver delay*> + <*Rx buffer delay*> + 3 + <*fixed core delay*>
   = 7.2 + 10.5 + 3 + 5

5. Calculate the Rx path delay through the REC master, by following the steps in "Rx Path Delay to AUX Output: Calculation Example" on page D–8. The Rx buffer delay is 33.5 cpri_clkout cycles, yielding a total delay of 48.25 cpri_clkout cycles.

   48.2 = <*fixed transceiver delay*> + <*Rx buffer delay*> + 3 + <*fixed core delay*>
   = 7.2 + 33.5 + 3 + 5

6. For each of the REC master and the RE slave, read the value in the tx_ex_buf_delay field of the CPRI_EX_DELAY_STATUS register (at register offset 0x40) and the value in the tx_ex_delay field of the CPRI_EX_DELAY_CONFIG register. Read the tx_ex_buf_delay field only after the tx_ex_buf_delay_valid bit in the register is high.

7. For each of the REC master and the RE slave, divide the value in the tx_ex_buf_delay register field by the value in the tx_ex_delay register field. The result is the current Tx elastic buffer delay in cpri_clkout cycles. In this example, the Tx elastic buffer delay in the REC master is 7.5 cpri_clkout cycles, and the Tx elastic buffer delay in the RE slave is 7.5 cpri_clkout cycles.

8. Calculate the Tx path delay through the REC master. According to Table D–4 on page D–11, the correct value of T_T4 is 6 cpri_clkout cycles, and according to Table D–5 on page D–13, the correct value of T_txv_TX is 3.6 cpri_clkout cycles. You calculated the Tx elastic buffer delay in steps 6 and 7.

   Tx path delay = T_T4 + <*Tx elastic buffer delay*> + T_txv_TX = 6 + 7.5 + 3.6 = 17.1

9. Calculate the Tx path delay through the RE slave. Because the device family is the same for the REC master and the RE slave in this example, they have the same T_T4 and T_txv_TX delays. You calculated the Tx elastic buffer delay in steps 6 and 7. It is the same as the Tx elastic buffer delay in the REC master, so the total Tx path delay in the RE slave is identical to the Tx path delay in the REC master in this case.

   Tx path delay = T_T4 + <*Tx elastic buffer delay*> + T_txv_TX = 6 + 7.5 + 3.6 = 17.1

10. Calculate
    T14 = rx_round_trip_delay – <*REC Rx path delay*> – <*REC Tx path delay*>
    = 110 – 48.2 – 17.1
    = 44.7 cpri_clkout cycles

11. Calculate
    Toffset = <*RE Rx path delay*> + <*RE Tx path delay*> + <*loopback delay*>,
    = 25.7 + 17.1 + 1
    = 43.8 cpri_clkout cycles

12. Perform the final calculation. Calculate
    Round-trip cable delay = T14 – Toffset
    = 44.7 – 43.8
    = 0.9 cpri_clkout cycles

Round-Trip and Cable Delay Calculation Example 2: Two Arria II GX Devices

This example shows the calculation for the case of two link partner CPRI IP cores configured with autorate negotiation enabled on Arria II GX devices, in a single-hop configuration, running at CPRI data rate 3.072 Gbps. In both devices, the cal_en field of the CPRI_AUTO_CAL register has the value of 0.

The calculation is identical to the calculation in Example 1, except that the fixed and transceiver delays are different in Arria II GX devices than in Stratix IV GX devices. In addition, Example 2 has a different value in the rx_round_trip_delay register field. In your own system, the Rx elastic buffer and Tx elastic buffer delays may also vary.

To calculate the round-trip cable delay in this system, perform the following steps:

1. Read the value in the rx_round_trip_delay field of the CPRI_ROUND_DELAY register (at register offset 0x38) of the REC master. For the example, the value is 0x7F, which is decimal 127.
2. For each of the REC master and the RE slave, read the value in the rx_ex_buf_delay field of the CPRI_EX_DELAY_STATUS register (at register offset 0x40) and the value in the rx_ex_delay field of the CPRI_EX_DELAY_CONFIG register. Read the rx_ex_buf_delay field only after the rx_ex_buf_delay_valid bit in the register is high.
3. For each of the REC master and the RE slave, divide the value in the rx_ex_buf_delay register field by the value in the rx_ex_delay register field. The result is the current Rx elastic buffer delay in cpri_clkout cycles. In this example, the Rx elastic buffer delay in the REC master is 32.25 cpri_clkout cycles, and the Rx elastic buffer delay in the RE slave is 8.9 cpri_clkout cycles.
4. Calculate the Rx path delay through the RE slave, by following the steps in "Rx Path Delay to AUX Output: Calculation Example" on page D–8. According to Table D–1 on page D–5, the correct value of T_txv_RX is 4.65 cpri_clkout cycles. According to Table D–3 on page D–8, the correct value of T_R1 is 4.5 cpri_clkout cycles. The Rx buffer delay is 8.9 cpri_clkout cycles, and the cal_pointer register field value is 3, yielding a total delay of 21.05 cpri_clkout cycles.

    21.05 = <*fixed transceiver delay*> + <*Rx buffer delay*> + 3 + <*fixed core delay*>
    = 4.65 + 8.9 + 3 + 4.5
5. Calculate the Rx path delay through the REC master, by following the steps in "Rx Path Delay to AUX Output: Calculation Example" on page D–8. The Rx buffer delay is 32.25 cpri_clkout cycles, and the cal_pointer register field value is 3, yielding a total delay of 43.75 cpri_clkout cycles.

    44.4 = <*fixed transceiver delay*> + <*Rx buffer delay*> + 3 + <*fixed core delay*>
    = 4.65 + 32.25 + 3 + 4.5

6. For each of the REC master and the RE slave, read the value in the `tx_ex_buf_delay` field of the `CPRI_EX_DELAY_STATUS` register (at register offset 0x40) and the value in the `tx_ex_delay` field of the `CPRI_EX_DELAY_CONFIG` register. Read the `tx_ex_buf_delay` field only after the `tx_ex_buf_delay_valid` bit in the register is high.
7. For each of the REC master and the RE slave, divide the value in the `tx_ex_buf_delay` register field by the value in the `tx_ex_delay` register field. The result is the current Tx elastic buffer delay in `cpri_clkout` cycles. In this example, the Tx elastic buffer delay in the REC master is 32.25 `cpri_clkout` cycles, and the Tx elastic buffer delay in the RE slave is 8.9 `cpri_clkout` cycles.
8. Calculate the Tx path delay through the REC master. According to Table D–4 on page D–11, the correct value of T_T4 is 5.5 `cpri_clkout` cycles, and according to Table D–5 on page D–13, the correct value of T_txv_TX is 3.35 `cpri_clkout` cycles.

   Tx path delay = T_T4 + <*Tx buffer delay*> + T_txv_TX = 5.5 + 32.25 + 3.35 = 41.1
9. Calculate the Tx path delay through the RE slave. Because the device family is the same for the REC master and the RE slave in this example, they have the same T_T4 and T_txv_TX delays. You calculated the Tx elastic buffer delay in steps 6 and 7.

   Tx path delay = T_T4 + <*Tx buffer delay*> + T_txv_TX = 5.5 + 8.9 + 3.35 = 17.75
10. Calculate
    `T14` = `rx_round_trip_delay` – <*REC Rx path delay*> – <*REC Tx path delay*>
    = 127 – 44.4 – 41.1
    = 41.5 `cpri_clkout` cycles
11. Calculate
    Toffset = <*RE Rx path delay*> + <*RE Tx path delay*> + <*loopback delay*>,
    = 21.5 + 17.75 + 1
    = 40.25 `cpri_clkout` cycles
12. Perform the final calculation. Calculate
    Round-trip cable delay = `T14` – Toffset
    = 41.5 – 40.25
    = 1.25 `cpri_clkout` cycles

Round-Trip and Cable Delay Calculation Example 3: Two Different Device Families

This example shows the calculation for the case of two link partner CPRI IP cores configured with autorate negotiation enabled in a single-hop configuration, running at CPRI data rate 3.072 Gbps. The REC master is configured on a Stratix IV GX device and the RE slave is configured on an Arria II GX device. In both devices, the `cal_en` field of the `CPRI_AUTO_CAL` register has the value of 0.

The calculation is identical to the calculation in Examples 1 and 2, except that the fixed and transceiver delays are different for the two different devices, so the fixed parts of the Rx path delay and Tx path delay are different on the two devices. In addition, Example 3 has a different value in the `rx_round_trip_delay` register field. In your own system, the Rx elastic buffer delay and Tx elastic buffer delay may also vary.

To calculate the round-trip cable delay in this system, perform the following steps:

1. Read the value in the rx_round_trip_delay field of the CPRI_ROUND_DELAY register (at register offset 0x38) of the REC master. For the example, the value is 0x84, which is decimal 132.
2. For each of the REC master and the RE slave, read the value in the rx_ex_buf_delay field of the CPRI_EX_DELAY_STATUS register (at register offset 0x40) and the value in the rx_ex_delay field of the CPRI_EX_DELAY_CONFIG register. Read the rx_ex_buf_delay field only after the rx_ex_buf_delay_valid bit in the register is high.
3. For each of the REC master and the RE slave, divide the value in the rx_ex_buf_delay register field by the value in the rx_ex_delay register field. The result is the current Rx elastic buffer delay in cpri_clkout cycles. In this example, the Rx elastic buffer delay in the REC master is 32.25 cpri_clkout cycles, and the Rx elastic buffer delay in the RE slave is 8.9 cpri_clkout cycles.
4. Calculate the Rx path delay through the RE slave, by following the steps in "Rx Path Delay to AUX Output: Calculation Example" on page D–8. According to Table D–1 on page D–5, the correct value of T_txv_RX is 4.65 cpri_clkout cycles. According to Table D–3 on page D–8, the correct value of T_R1 is 4.5 cpri_clkout cycles. The Rx buffer delay is 8.9 cpri_clkout cycles, yielding a total delay of 21.05 cpri_clkout cycles.

   21.05 = <*fixed transceiver delay*> + <*Rx buffer delay*> + 3 + <*fixed core delay*>
   = 4.65 + 8.9 + 3 + 4.5
5. Calculate the Rx path delay through the REC master, by following the steps in "Rx Path Delay to AUX Output: Calculation Example" on page D–8. The Rx buffer delay is 32.25 cpri_clkout cycles, yielding a total delay of 47.45 cpri_clkout cycles.

   47.45 = <*fixed transceiver delay*> + <*Rx buffer delay*> + 3 + <*fixed core delay*>
   = 7.2 + 32.25 + 3 + 5
6. For each of the REC master and the RE slave, read the value in the tx_ex_buf_delay field of the CPRI_EX_DELAY_STATUS register (at register offset 0x40) and the value in the tx_ex_delay field of the CPRI_EX_DELAY_CONFIG register. Read the tx_ex_buf_delay field only after the tx_ex_buf_delay_valid bit in the register is high.
7. For each of the REC master and the RE slave, divide the value in the tx_ex_buf_delay register field by the value in the tx_ex_delay register field. The result is the current Tx elastic buffer delay in cpri_clkout cycles. In this example, the Tx elastic buffer delay in the REC master is 32.25 cpri_clkout cycles, and the Tx elastic buffer delay in the RE slave is 8.9 cpri_clkout cycles.
8. Calculate the Tx path delay through the REC master. According to Table D–4 on page D–11, the correct value of T_T4 is 6 cpri_clkout cycles, and according to Table D–5 on page D–13, the correct value of T_txv_TX is 3.6 cpri_clkout cycles.

   Tx path delay = T_T4 + <*Tx buffer delay*> + T_txv_TX = 6 + 32.25 + 3.6 = 41.85

9. Calculate the Tx path delay through the RE slave. According to Table D–4 on page D–11, the correct value of T_T4 is 5.5 cpri_clkout cycles, and according to Table D–5 on page D–13, the correct value of T_txv_TX is 3.35 cpri_clkout cycles.

   Tx path delay = T_T4 + <*Tx buffer delay*> + T_txv_TX = 5.5 + 8.9 + 3.35 = 17.75

10. Calculate
    T14 = rx_round_trip_delay – <*REC Rx path delay*> – <*REC Tx path delay*>
    = 132 – 47.45 – 41.85
    = 42.7 cpri_clkout cycles

11. Calculate
    Toffset = <*RE Rx path delay*> + <*RE Tx path delay*> + <*loopback delay*>
    = 21.05 + 17.75 + 1
    = 39.8 cpri_clkout cycles

12. Perform the final calculation. Calculate
    Round-trip cable delay = T14 – Toffset
    = 42.7 – 39.8
    = 2.9 cpri_clkout cycles

Round-Trip and Cable Delay Calculation Example 4: Two Different Device Families

This example describes the calculation for the case of two link partner CPRI IP cores configured with autorate negotiation enabled in a single-hop configuration, running at CPRI data rate 3.072 Gbps. The REC master is configured on an Arria II GX device and the RE slave is configured on a Stratix IV GX device.

The calculation is identical to the calculation in Example 3, except that the fixed and transceiver delays on the REC master in Example 3 are the delays on the RE slave in Example 4, and the fixed and transceiver delays on the RE slave in Example 3 are the delays on the REC master in Example 4, because these delays depend on the device family. In addition, Example 4 has a different value in the rx_round_trip_delay register field. In your own system, the Rx elastic buffer delay and Tx elastic buffer delay may also vary.

For the example, the two CPRI IP cores have the following register values:

- The rx_round_trip_delay field of the CPRI_ROUND_DELAY register (at register offset 0x38) of the REC master holds the value 0x62, which is decimal 98.
- In the REC master, the rx_ex_delay field of the CPRI_EX_DELAY_CONFIG register (at register offset 0x3C) holds the value 0x4.
- In the REC master, after the rx_ex_buf_delay_valid bit in the register is high, the rx_ex_buf_delay field of the CPRI_EX_DELAY_STATUS register (at register offset 0x40) holds the value 0x81, which is decimal 129.
- In the RE slave, the rx_ex_delay field of the CPRI_EX_DELAY_CONFIG register (at register offset 0x3C) holds the value 0x7F, which is decimal 127.
- In the RE slave, after the rx_ex_buf_delay_valid bit in the register is high, the rx_ex_buf_delay field of the CPRI_EX_DELAY_STATUS register (at register offset 0x40) holds the value 0x46A, which is decimal 1130.

From these register values, a similar calculation for the Tx elastic buffer delay, and the information in Table D–1 on page D–5, Table D–3 on page D–8, Table D–4 on page D–11, and Table D–5 on page D–13, you can calculate the following values:

1. The REC master Rx elastic buffer delay is 129 / 4 = 32.25 cpri_clkout cycles.
2. The RE slave Rx elastic buffer delay is 1130 / 127 = 8.9 cpri_clkout cycles.
3. The REC master Rx path delay is 4.65 + 32.25 + 3 + 4.5= 44.4 cpri_clkout cycles.
4. The RE slave Rx path delay is 7.2 + 8.9 +3 + 5 = 24.1 cpri_clkout cycles.
5. The Tx elastic buffer delay in the REC master is 4.5 cpri_clkout cycles. Therefore, the REC master Tx path delay is 5.5 + 4.5 + 3.35 = 13.35 cpri_clkout cycles.
6. The Tx elastic buffer delay in the RE slave is 3.2 cpri_clkout cycles. Therefore, the RE slave Tx path delay is 6 + 3.2 + 3.6 = 12.8 cpri_clkout cycles.
7. T14 = rx_round_trip_delay – <*REC Rx path delay*> – <*REC Tx path delay*>
   = 98 – 44.4 – 13.35
   = 40.25 cpri_clkout cycles
8. Toffset = <*RE Rx path delay*> + <*RE Tx path delay*> + <*loopback delay*>,
   = 24.1 + 12.8 + 1
   = 37.9 cpri_clkout cycles
9. Round-trip cable delay = T14 – Toffset
   = 40.25 – 37.9
   = 2.35 cpri_clkout cycles

Dynamic Pipelining for Automatic Round-Trip Delay Calibration

The CPRI IP core provides an additional, optional mechanism to help minimize the variation in the round-trip delay through a CPRI REC or RE master. The CPRI IP core is configured with a set of *n* (currently five) pipelined registers following the Rx elastic buffer in the Rx path. If the cal_en bit in the CPRI_AUTO_CAL register has the value of 1, the autocalibration feature is active. The user programs the cal_rtd field of the CPRI_AUTO_CAL register with the expected number of cpri_clkout cycles of round-trip delay. The CPRI IP core adjusts the number of pipeline registers the data passes through (in contrast to the number of registers it bypasses) to compensate for mismatches between the desired round-trip delay programmed in the cal_rtd field, and the actual round-trip delay recorded in the CPRI_ROUND_DELAY register.

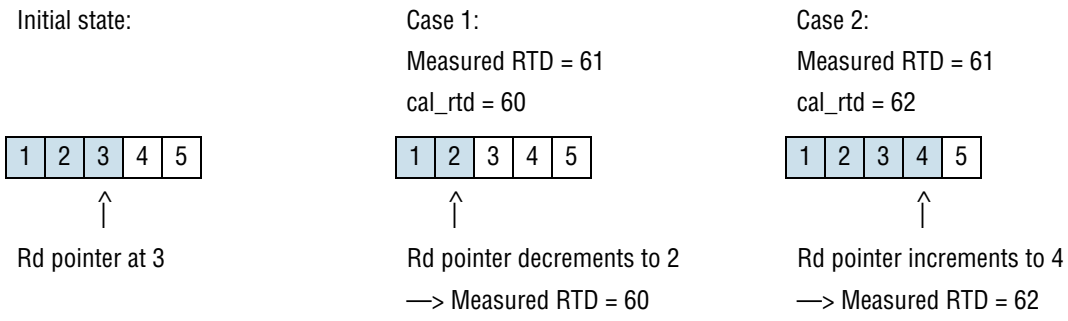
The cal_status field reports whether the CPRI IP core is successful in keeping the round-trip delay at the value you prescribed in the cal_rtd field. The value of the cal_status bits should remain at 2'b11. If the value does not remain at 2'b11, you should adjust the value in the cal_rtd field.Refer to Table 7–29 on page 7–13 for the full encoding of these status bits and how to determine whether to increase or decrease the value of cal_rtd.

Initially, the number of pipeline registers the CPRI IP core uses is one half the total number *n* of available register stages. This initial setting allows the CPRI IP core to adjust the number up or down as required, and adds *n*/2+1 latency cycles to the Rx path delay and the round-trip delay. The number of available register stages is five and the default number of register stages of delay is three.

Figure D–7 shows two example behaviors of the autocalibration feature. In the examples, the CPRI IP core changes the value of the pipeline read pointer in response to a change in the measured actual round-trip delay through the IP core. Figure D–7 shows the CPRI IP core in the following three states:

1. In the initial state, the CPRI IP core sets the read pointer for the pipeline registers to the middle register.
2. In Case 1, the application writes the value of 60 in the `cal_rtd` field. When the CPRI IP core measures the actual round-trip delay and sets the `rx_round_trip_delay` field in the `CPRI_ROUND_DELAY` register to the value of 61, the CPRI IP core responds by moving the read pointer to decrease the pipeline length, and therefore the measured round-trip delay value, by one `cpri_clkout` cycle. The adjustment achieves the desired effect: the measured round-trip delay value changes to 60.
3. In Case 2, the application writes the value of 62 in the `cal_rtd` field, instead. When the CPRI IP core measures the actual round-trip delay and sets the `rx_round_trip_delay` field in the `CPRI_ROUND_DELAY` register to the value of 61, the CPRI IP core responds by moving the read pointer to increase the pipeline length, and therefore the measured round-trip delay value, by one `cpri_clkout` cycle. The adjustment achieves the desired effect: the measured round-trip delay value changes to 62.

Figure D–7. Round-Trip Delay Autocalibration Examples



Round-Trip Calculations for a Multihop Configuration

In a multihop system, you must combine the delays between and through the different CPRI masters and CPRI RE slaves to determine the round-trip delay.

Multihop Round-Trip Delay Calculation

The value in the `rx_round_trip_delay` field of the `CPRI_ROUND_DELAY` register is meaningful only in CPRI REC and RE masters. It records the round-trip delay for the current hop only, as shown in Figure D–1 on page D–2.

To determine the round-trip delay of a full multihop system, you must add together the values in the `CPRI_ROUND_DELAY` registers of the REC and RE masters in the system, plus the delays through the external routers, and subtract the loopback delay from all the hops except the final hop. Use the following calculation, based on the labels in Figure D–2 on page D–2:

$$\text{Round-trip delay} = \sum \texttt{rx\_round\_trip\_delay}\ (\text{hop i}) + \sum_{-n} (\text{TBdelayUL} + \text{TBdelayDL})(j)$$

where the REC and RE masters in the configuration are labeled i=0,1,...,n and the routing layers in the configuration, and their uplink and downlink delays, are labeled j=0,1,...,(n-1).

As the equation shows, you must omit the loopback delay of one `cpri_clkout` cycle from the single-hop calculation for all but the final pair of CPRI link partners. The loopback delay is only relevant at the turnaround point of the full multihop path.

Multihop Round-Trip Cable Delay Calculation

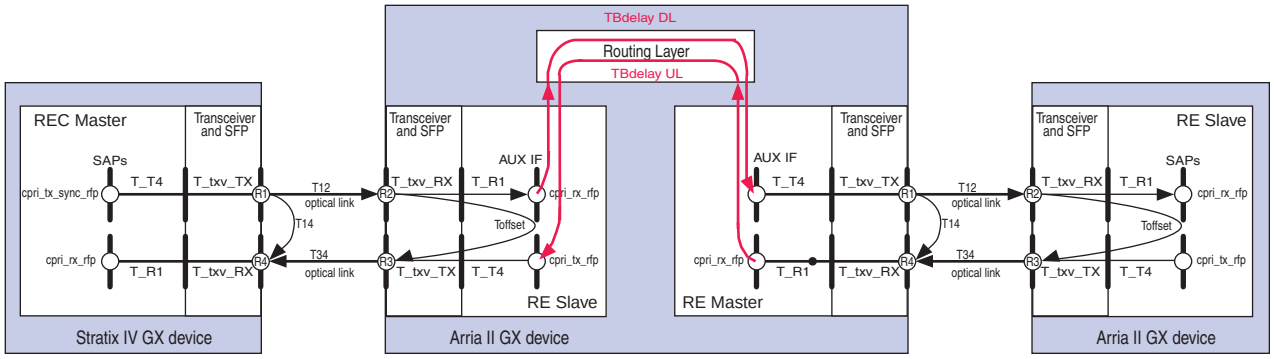
To determine the local round-trip cable delay at each hop, use the method described in "Round-Trip and Cable Delay Calculations for a Single-Hop Configuration", for the REC or RE master and the RE slave at the current hop. Half of the resulting value is assumed to be the cable delay in each direction at the current hop.

The round-trip cable delay is the sum of all the local round-trip cable delays in the multihop path.

Two-Hop Round-Trip and Cable Delay Calculation Example

This section walks through an example calculation for the system shown in Figure D–8.

Figure D–8. Two-Hop System for Multihop Delay Calculation Example



In the example, all of the four CPRI IP cores are configured with autorate negotiation enabled and are running at CPRI data rate 3.072 Gbps.

Example calculations for the first hop appear in "Round-Trip and Cable Delay Calculation Example 3: Two Different Device Families" on page D–18. Example calculations for the second hop appear in "Round-Trip and Cable Delay Calculation Example 2: Two Arria II GX Devices" on page D–17.

Assuming the multihop system has the same register values as in these two single-hop examples, you calculate the multihop round-trip delay and total cable delay as follows:

$$\text{Round-trip delay} = \sum_{i=1}^{n-1} \texttt{rx\_round\_trip\_delay}\ (\text{hop } i) + \sum_{j=1}^{n} (\text{TBdelayUL} + \text{TBdelayDL})(j)$$

$$= (132 + 127) + \text{TBdelayUL} + \text{TBdelayDL} - 1$$

$$= 258\ \texttt{cpri\_clkout}\ \text{cycles} + \text{TBdelayUL} + \text{TBdelayDL}$$

Total round-trip CPRI-link cable delay = 2.9 + 1.25 = 4.15 `cpri_clkout` cycles

The CPRI IP core does not provide a mechanism to measure the delays through the external routing layer.

E. Integrating the CPRI IP Core Timing Constraints in the Full Design

When you generate your CPRI IP core variation, the Quartus II software generates a Synopsys Design Constraints File (**.sdc**) that specifies the timing constraints for the input clocks to your IP core. At the time you generate the CPRI IP core, the design is not yet complete and the CPRI IP core is not yet connected in the design. The final clock names and paths are not yet known, and therefore the Quartus II software cannot incorporate the final signal names in the **.sdc** file it generates automatically.

Instead, you must modify the clock signal names in this file manually to integrate these constraints with the timing constraints for your full design.

This appendix describes by example how to integrate the timing constraints that the Quartus II software generates with your CPRI IP core into the timing constraints for your design.

For a list of the input clocks to the CPRI IP core, refer to Table 4–1 on page 4–3.

In the Quartus II software release v12.0, the automatically generated **altera_cpri.sdc** file contains the CPRI IP core timing constraints.

For a CPRI IP core with a single antenna-carrier interface that runs at the CPRI line rate of 3.072 Gbps and targets an Arria II GX device, the Quartus II software v12.0 generates an **altera_cpri.sdc** file with the following timing constraints:

```
#ALTGX Transceiver Reference Clock
create_clock -name gxb_refclk -period 6.510 -waveform {0.000 3.255} [get_ports
gxb_refclk]

#Clock from Clean-Up PLL (RE slave only)
create_clock -name gxb_pll_inclk -period 6.510 -waveform {0.000 3.255} [get_ports
gxb_pll_inclk]

#ALTGX Calibration Block Clock (10MHz to 125 MHz)
create_clock -name gxb_cal_blk_clk -period 8.000 -waveform {0.000 4.000}
[get_ports gxb_cal_blk_clk]

#ALTGX_RECONFIG Clock (37.5MHz to 50MHz)
create_clock -name reconfig_clk -period 20.000 -waveform {0.000 10.000}
[get_ports reconfig_clk]

#CPRI CPU Clock
create_clock -name cpu_clk -period 32.552 -waveform {0.000 16.276} [get_ports
cpu_clk]

#Extended Delay Measurement Clock
create_clock -name clk_ex_delay -period 13.123 -waveform {0.000 6.562} [get_ports
clk_ex_delay]

#Data Mapping Clock
create_clock -name map0_tx_clk -period 260.416 -waveform {0.000 130.208}
[get_ports map0_tx_clk]

create_clock -name map0_rx_clk -period 260.416 -waveform {0.000 130.208}
[get_ports map0_rx_clk]

derive_pll_clocks

derive_clock_uncertainty
```

```
set_false_path -from * -to *sync
set_false_path -from * -to *sync[*]
set_false_path -from * -to *sync1
set_false_path -from * -to *sync1[*]
set_false_path -from * -to *s0
set_false_path -from * -to *s0[*]

create_generated_clock -name txclk_div2 -source [get_pins -compatibility_mode
*transmit_pcs0|clkout] -divide_by 2 [get_registers *txclk_div2]

derive_clock_uncertainty

set_clock_groups -exclusive -group txclk_div2 -group *receive_pcs0|clkout
set_clock_groups -exclusive -group *transmit_pcs0|clkout -group
*receive_pcs0|clkout
set_clock_groups -asynchronous -group cpu_clk -group txclk_div2
set_clock_groups -asynchronous -group map*_clk -group txclk_div2
set_clock_groups -asynchronous -group clk_ex_delay -group {txclk_div2
*transmit_pcs0|clkout *receive_pcs0|clkout}
set_clock_groups -asynchronous -group reconfig_clk -group txclk_div2
```

When you embed your CPRI IP core variation in your full design, you drive the CPRI IP core clocks directly from the top-level signals of the design or indirectly through internal logic. The timing constraints for your full design must reference the clock names relative to the full design hierarchy.

Figure E–1 shows an example design that contains the example CPRI IP core variation.

Figure E–1. Clocks Driving CPRI IP Core Clocks in Example Full Design

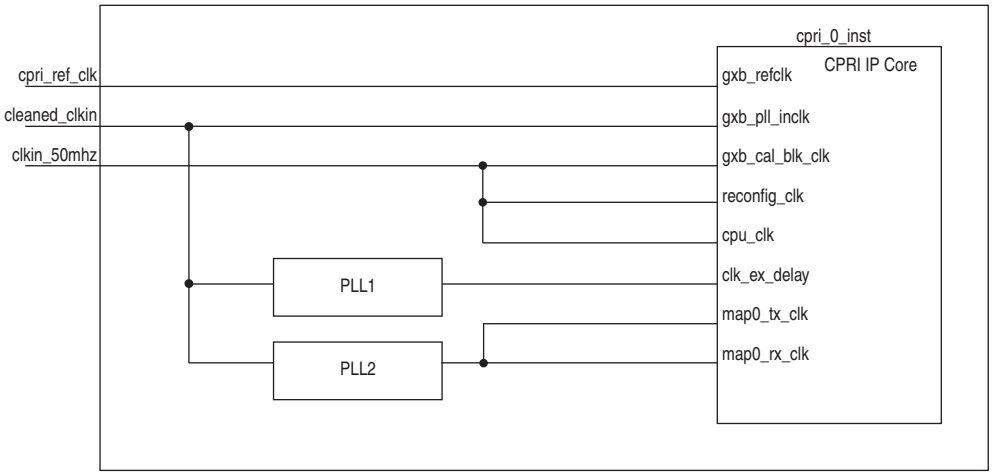


Table E–1 lists the correspondence between the clock names in the **.sdc** file and the signal names in the full design.

Table E–1. Stand-Alone IP Core Clock Names and Example Design Clock Names

Stand-Alone IP Core Clock Name	Full Design Clock Name
gxb_refclk	cpri_ref_clk
gxb_pll_inclk	cleaned_clkin
gxb_cal_blk_clk	clkin_50mhz
reconfig_clk	clkin_50mhz

Table E–1. Stand-Alone IP Core Clock Names and Example Design Clock Names

Stand-Alone IP Core Clock Name	Full Design Clock Name
cpu_clk	clkin_50mhz
clk_ex_delay	pll1|clk[0]
map0_tx_clk	pll2|clk[0]
map0_rx_clk	pll2|clk[0]

After you complete your design, you must modify the clock names in the **.sdc** file to the full-design clock names, taking into account both the CPRI IP core instance name in the full design, and the design hierarchy. After you make the required modifications, the example **.sdc** file contains the following substitute timing constraints:

```
#ALTGX Transceiver Reference Clock
create_clock -name cpri_ref_clk -period 6.510 -waveform {0.000 3.255} [get_ports
cpri_ref_clk]

#Clock from Clean-Up PLL (RE slave only)
create_clock -name cleaned_clkin -period 6.510 -waveform {0.000 3.255} [get_ports
cleaned_clkin]

#50MHz Clock to Drive Calibration Block Clock, CPU Clock, and Reconfig Clock
create_clock -name clkin_50mhz -period 20.000 -waveform {0.000 10.000} [get_ports
clkin_50mhz]

derive_pll_clocks

derive_clock_uncertainty

set_false_path -from * -to *cpri_0_inst*sync
set_false_path -from * -to *cpri_0_inst*sync[*]
set_false_path -from * -to *cpri_0_inst*sync1
set_false_path -from * -to *cpri_0_inst*sync1[*]
set_false_path -from * -to *cpri_0_inst*s0
set_false_path -from * -to *cpri_0_inst*s0[*]

create_generated_clock -name txclk_div2 -source [get_pins -compatibility_mode
*cpri_0_inst*transmit_pcs0|clkout] -divide_by 2 [get_registers
*cpri_0_inst*txclk_div2]

derive_clock_uncertainty

set_clock_groups -exclusive -group txclk_div2 -group
*cpri_0_inst*receive_pcs0|clkout
set_clock_groups -exclusive -group *cpri_0_inst*transmit_pcs0|clkout -group
*cpri_0_inst*receive_pcs0|clkout
set_clock_groups -asynchronous -group clkin_50mhz -group txclk_div2
set_clock_groups -asynchronous -group pll1|clk[0] -group txclk_div2
set_clock_groups -asynchronous -group pll2|clk[0] -group {txclk_div2
*cpri_0_inst*transmit_pcs0|clkout *cpri_0_inst*receive_pcs0|clkout}
```

The example illustrates the following guidelines you must follow when finalizing the **.sdc** file for your design:

- The CPRI IP core clock ports are not in one-to-one correspondence with the full design input clock ports. You must use the correspondence between the stand-alone IP core clocks and the full design clocks to define the integrated design timing constraints for the external clocks that drive CPRI IP core clocks directly.

- To integrate timing constraints with wild cards that identify lower level nodes in the CPRI IP core, you must modify each lower level node designator with the CPRI IP core instance name to ensure the new file constraints the correct design instance of each CPRI IP core signal name.

After you perform the manual mapping and custmize the .sdc file according to this correspondence, your file contains the correct timing constraints for the CPRI IP core in your full design.

F. Porting a CPRI IP Core from the Previous Version of the Software

This appendix describes how to port your CPRI IP core from the previous version of the Quartus II software

To upgrade your CPRI IP core that you developed and generated using the Quartus II software v11.1, to the IP core v12.0, perform the following steps:

1. Open the Quartus II software v12.0.
2. On the File menu, click **Open Project**.
3. Navigate to the location of the **.qpf** file you generated with the Quartus II software v11.1.
4. Select the **.qpf** file and click **Open**.
5. Open the existing IP core for editing in the MegaWizard Plug-In Manager.
6. For true backward compatibility, set **Mapping mode(s)** to **All**. However, if you program your IP core variation consistently to a single mapping mode, you can select the corresponding parameter value to improve resource utilization.
7. Turn on **Include HDLC block**.
8. Click **Finish**.
9. Proceed with simulation and compilation of your design.

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this user guide.

Date	Version	Changes Made
May 2012	12.0	■ Added CPRI line rate of 9.8 Gbps in Arria V GT and Stratix V devices. ■ Added support for autorate negotiation up to 6.144 Gbps in Arria V devices. ■ Added support for autorate negotiation up to 9.8 Gbps in Stratix V devices. ■ Added new parameter to specify inclusion or exclusion of an HDLC block. ■ Added new parameter to specify the MAP interface mapping mode. ■ Updated Figure 4–26 on page 4–50, CPRI Frame Synchronization Machine, to include the descrambling conditions and remove a redundant state. ■ Updated Figure 4–13 on page 4–24 and discussion of MAP interface TX synchronous buffer mode to encourage the application to assert `mapN_tx_resync` and `mapN_tx_valid` simultaneously. ■ Updated clocks presentation in "Clocking Structure" on page 4–3 and separated from reset signals presentation. ■ Updated Chapter 8, Testbenches with new testbenches for Arria V and Stratix V devices. ■ Moved information about loopback modes and PRBS generation and testing from Chapter 4, Functional Description to new Chapter 5, Testing Features. ■ Moved information about the advanced AxC mapping modes from Chapter 4, Functional Description to new appendix Appendix C, Advanced AxC Mapping Modes and updated the presentation. ■ Moved information about the RX delay measurement and TX delay calibration from Chapter 4, Functional Description to new appendix Appendix D, Delay Measurement and Calibration. ■ Added new appendix Appendix E, Integrating the CPRI IP Core Timing Constraints in the Full Design. ■ Reordered sections in Chapter 4, Functional Description to emphasize the MAP and AUX interfaces and to group together the modules accessed through the CPU interface. ■ Reordered presentation of signals in Chapter 6, Signals to reflect order in Chapter 4, Functional Description. ■ Enhanced description of control word access through CPU interface in new section "Accessing the Hyperframe Control Words" on page 4–39. ■ Updated description of Ethernet communication through the CPU interface in "Accessing the Ethernet Channel" on page 4–42. ■ Moved "Reset Control Word" on page 4–51 from Reset section of "Reset Requirements" on page 4–9 to "CPRI Protocol Interface Layer (Physical Layer)" on page 4–46.

Date	Version	Changes Made
November 2011	11.1	■ Added support for Arria V and Stratix V devices. ■ Added information about new transceiver IP (the Altera Deterministic Latency PHY IP core) in Arria V and Stratix V variations. ■ Added Tx elastic buffer and Tx extended delay measurement information. ■ Updated clocking diagrams with Tx elastic buffer and removal of divider on transceiver-side clock before clocking Rx and Tx elastic buffers. Consolidated from six figures to two. ■ Added information about new delay measurement features to enhance the consistency of the round-trip delay through a CPRI RE slave: Tx bitslip, autocalibration. ■ Added new registers CPRI_TX_BITSLIP and CPRI_AUTO_CAL to support new features. ■ Removed use of the rx_byte_delay field in the CPRI_RX_DELAY register from the RX path delay calculation. ■ Added new advanced Method 1 mapping mode and updated map_mode encodings. ■ Added new parameter to enable clocking AxC interfaces with cpri_clkout. The resulting new synchronization mode requires a new signal, mapN_rx_start, per AxC interface. ■ Added timing diagrams for three synchronization modes on MAP interface and for cpri_tx_sync_rfp response behavior. ■ Added information about data order on the AUX interface. ■ Enhanced PRBS mode description. ■ Added Loopback Modes section in Functional Description chapter. ■ Updated Appendix C, Porting a CPRI IP Core from the Previous Version of the Software. ■ Refered to new What's New in Altera IP page for information about IP core support level for some device families.

May 2011	11.0	■ Upgraded to final support for Arria II GZ and Cyclone IV GX devices.
		■ Upgraded to HardCopy Compilation support for HardCopy IV GX devices.
		■ Added byte-enable signal.
		■ Added parameter to control WIDTH_RX_BUF.
		■ Enhanced delay measurement and cpri_tx_sync_rfp signal descriptions.
		■ Modified MII and frame synchronization machine descriptions.
		■ Miscellaneous small fixes, including:
		  ■ Updated address range for MAP and AUX interface configuration registers in Table 6–2 on page 6–1 to match individual register addresses as updated for v10.1.
		  ■ Updated descriptions of frame synchronization machine and cpri_rx_cnt_sync signal.
		■ Added Appendix C, Porting a CPRI IP Core from the Previous Version of the Software.

Date	Version	Changes Made
December 2010	10.1	■ Added support for Arria II GZ devices. ■ Added support for additional CPRI data rates in Arria II GX devices. ■ Updated register addresses. ■ Added scrambler/descrambler support. ■ Enhanced descriptions of offset registers and delay calculations. ■ Added CPU interrupt for remote hardware reset. ■ Enhanced testbench suite to include one new testbench, to demonstrate autorate negotiation in Cyclone IV GX devices.
July 2010	10.0	■ Added support for Cyclone IV GX devices. ■ Added GUI parameter to enable autorate negotiation and two signals to support visibility of the feature status. ■ Enhanced descriptions of MII, MAP interface synchronous buffer mode, and use of AUX interface mask. ■ Enhanced testbench suite to include two new testbenches, to demonstrate operation with no MAP interface and to demonstrate autorate negotiation.
February 2010	9.1 SP1	Initial release.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

Contact (1)	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support (general)	Email	nacomp@altera.com
(software licensing)	Email	authorization@altera.com

Note to Table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.

Visual Cue	Meaning
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, *n* + 1. Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>*.**pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections in a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix n denotes an active-low signal. For example, `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.
	The question mark directs you to a software help system with related information.
	The feet direct you to another document or website with related information.
	The multimedia icon directs you to a related multimedia presentation.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.
	The feedback icon allows you to submit feedback to Altera about the document. Methods for collecting feedback vary as appropriate for each document.