CPRI MegaCore Function

User Guide

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

UG-01062-3.0

Document last updated for Altera Complete Design Suite version: 10.1
Document publication date: December 2010

Subscribe

Contents

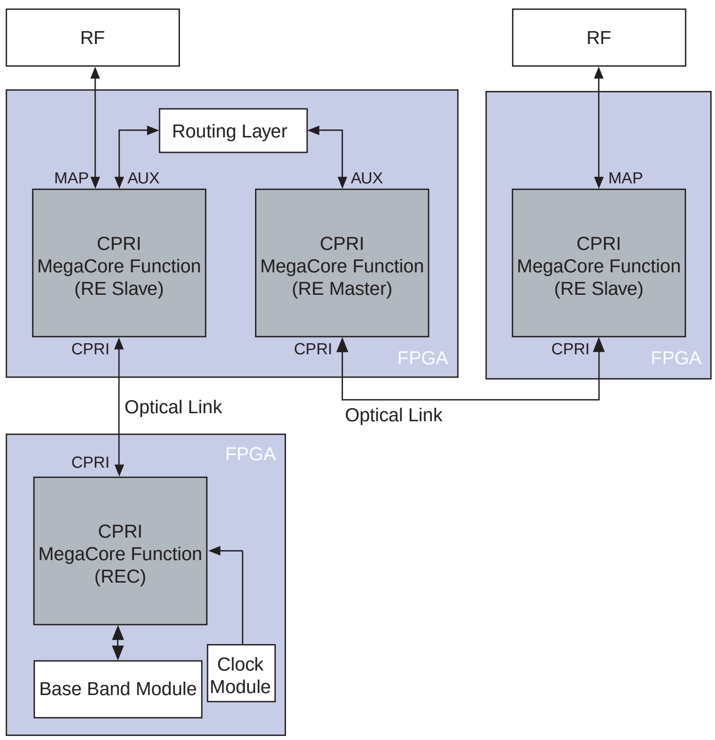
Chapter 5. Signals

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

Figure 1–1. Typical CPRI Application on Altera Devices

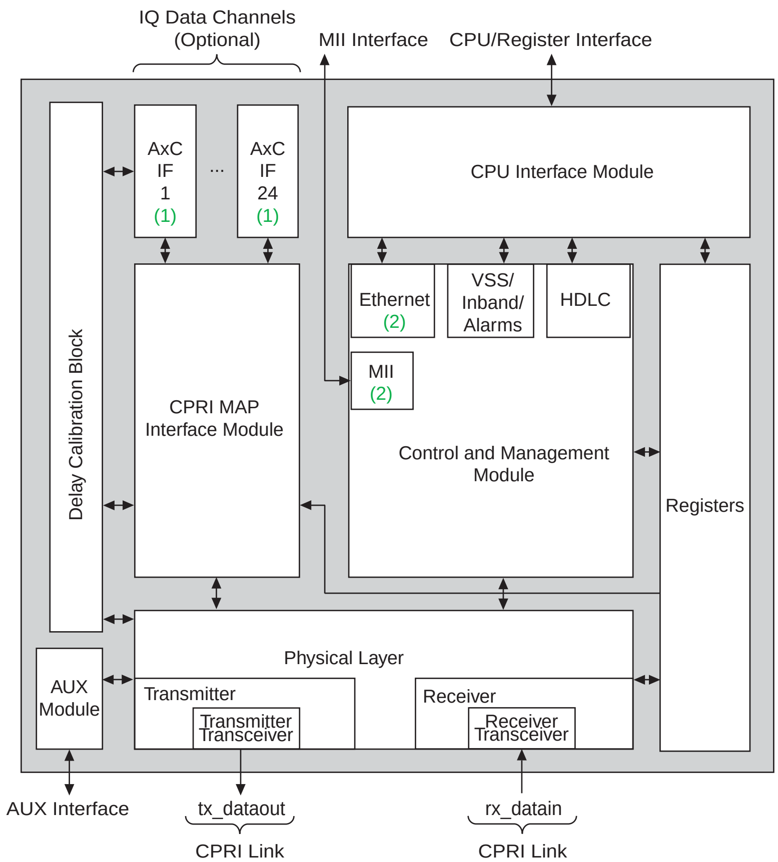


General Description

The CPRI MegaCore function includes interfaces to communicate between RE and REC (the CPRI link), to communicate with RF (IQ data channels), to communicate with a routing layer for daisy chain topologies, to communicate with a processor for register updates, and to support High-Level Data Link Control (HDLC) and Ethernet communication. You configure the CPRI MegaCore function to support either Ethernet communication with an Ethernet media access control (MAC) block included in the MegaCore function, 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 main blocks of the CPRI MegaCore function.

Figure 1–2. CPRI MegaCore Function Block Diagram



Note to Figure 1–2:

(1) You can configure your CPRI MegaCore function with zero, one, or multiple antenna-carrier interfaces.

(2) You can configure your CPRI MegaCore function with an Ethernet MAC block or an MII block.

CPRI MegaCore Function Features

The CPRI MegaCore function has the following features:

- Compliant with Common Public Radio Interface (CPRI) Specification V4.1 (2009-02-18) 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.
- Programmable CPRI communication line rate (to 614.4, 1228.8, 2457.6, 3072.0, 4915.2, or 6144.0 Mbps) using Altera on-chip high-speed transceivers.
- Includes features to support auto-negotiation of line rate.
- Supports CPRI transmission scrambling and descrambling at 4195.2 Mbps and at 6144.0 Mbps.

- Interface to external or on-chip processor, using the Altera Avalon® Memory-Mapped (Avalon-MM) interconnect specification.
- AUX interface allows you to pass data from slave to master ports or custom mappers to implement daisy-chain topologies.
- Accurate CPRI connection Rx delay measurement.
- Up to 24 antenna-carrier interfaces.
- Synchronous buffer mode or simple FIFO mode for MAP interfaces.
- Independent sample rates for each antenna-carrier interface.
- Supports 15- and 16-bit data sample widths on uplink and downlink, and 7- and 8-bit data sample widths on uplink with oversampling ratio 2, using the Altera Avalon Streaming (Avalon-ST) interconnect specification.
- IQ data interface supports mapping methods in sections 4.2.7.2.5 and 4.2.7.2.7 of the CPRI V4.1 Specification, and mapping Options 1 and 2 in sections 4.2.7.2.3 and 4.2.7.2.4 of the CPRI V4.1 Specification.
- Supports the 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 V4.1 Specification.
- Compliant with the CPRI V4.1 Specification to support UTRA-FDD (UMTS/WCDMA), E-UTRA (3GPP LTE specification), and WiMAX (IEEE 802.16 standard).
- Supports the UMTS/LTE mapping methods described in Section 4.2.7.2 of the CPRI V4.1 Specification.
- Supports the WiMAX timing control methodology described in Section 4.2.8.2 of the CPRI V4.1 Specification.
- Supports Ethernet communication to and from the CPRI link.
- Hardware reset output signal, provided as Altera-specified control word in CPRI communication frame, allows software to respond to CPRI communication partner reset request.
- Vendor-specific subchannel (VSS) communication supported by CPRI link interface.
- Separate reset signal for each clock domain.
- Diagnostic parallel reverse loopback paths.

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 shows the level of support offered by the CPRI MegaCore function for each Altera device family.

Table 1–2. Device Family Support

Device Family	Support
Arria® II GX	Final
Arria II GZ	Preliminary
Cyclone® IV GX	Preliminary
HardCopy® IV GX	HardCopy Companion
Stratix® IV GX	Final
Other device families	No support

MegaCore Verification

Before releasing a version of the CPRI MegaCore function, 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. These files are tested in simulation and hardware to confirm functionality.

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

Performance and Resource Utilization

This section contains tables showing MegaCore function variation size and performance examples.

Table 1–3 and Table 1–4 list the resources and expected performance for different CPRI MegaCore function variations.

All of these variations are in Master mode and have auto-rate negotiation turned off.

Table 1–3 shows results obtained using the Quartus II software v10.1 for the following devices:

- Arria II GX (EP2AGX260FF35C4 for line rates 614.4, 1228.8, 2457.6, and 3072 Mbps; EP2AGX125EF45I3 for line rates 4915.2 and 6144 Mbps)
- Arria II GZ (EP2AGZ225FF35C3)
- Stratix IV (EP4SGX360NF45C2)

Table 1–3. CPRI MegaCore Function FPGA Resource Utilization (Part 1 of 2)

Device	Parameters			Memory			
	Line Rate (Mbps)	Include MAC Block	Number of Antenna-Carrier Interfaces	Combinational ALUTs	Logic Registers	M9K Blocks	MLAB Cells
Arria II GX	614.4	Yes	0	4281	3444	23	4
			1	5580	4299	25	35
			2	5859	4467	27	35
			3	6092	4635	29	35
			4	6342	4803	31	35
		No	0	2163	1875	15	4
			1	3515	2730	17	35
			2	3794	2898	19	35
			3	4041	3066	21	35
			4	4297	3234	23	35
	1228.8, 2457.6, 3072, 4915.2, 6144	Yes	0	4381	3441	23	4
			1	5685	4298	25	36
			8	7461	5474	39	36
			16	9401	6818	55	36
			20	10435	7490	63	36
		No	0	2133	1904	14	4
			1	3498	2761	16	36
			8	5234	3937	30	36
			16	7199	5281	46	36
			20	8227	5953	54	36
Arria II GZ	614.4	Yes	0	4286	3444	23	4
			1	5617	4299	25	35
			2	5895	4467	27	35
			3	6132	4635	29	35
			4	6378	4803	31	35
		No	0	2163	1875	15	4
			1	3516	2730	17	35
			2	3796	2898	19	35
			3	4044	3066	21	35
			4	4301	3234	23	35

Table 1–3. CPRI MegaCore Function FPGA Resource Utilization (Part 2 of 2)

Device	Parameters			Memory			
	Line Rate (Mbps)	Include MAC Block	Number of Antenna-Carrier Interfaces	Combinational ALUTs	Logic Registers	M9K Blocks	MLAB Cells
Arria II GZ (continued)	1228.8, 2457.6, 3072, 4915.2, 6144	Yes	0	4503	3508	20	4
			1	5764	4336	22	36
			8	7513	5516	36	36
			16	9531	6874	52	36
			20	10537	7550	60	36
		No	0	2345	1939	12	4
			1	3595	2767	14	36
			8	5341	3955	28	36
			16	7303	5310	44	36
			20	8323	5976	52	36
Stratix IV GX	614.4	Yes	0	4290	3444	23	4
			1	5616	4299	25	35
			2	5898	4467	27	35
			3	6140	4635	29	35
			4	6386	4803	31	35
		No	0	2162	1875	15	4
			1	3516	2730	17	35
			2	3798	2898	19	35
			3	4041	3066	21	35
			4	4294	3234	23	35
	1228.8, 2457.6, 3072, 4915.2, 6144	Yes	0	4503	3508	20	4
			1	5764	4336	22	36
			8	7507	5514	36	36
			16	9526	6866	52	36
			24	11511	8278	68	36
		No	0	2342	1939	12	4
			1	3591	2767	14	36
			8	5324	3947	28	36
			16	7294	4289	44	36
			24	9337	6644	60	36

Table 1–4 shows results obtained using the Quartus II software v10.1 for the following device:

- Cyclone IV GX (EP4CGX150DF31C7)

Table 1–4. CPRI MegaCore Function Cyclone IV GX Resource Utilization

Device	Parameters			LEs	M9K Memory Blocks
	Line Rate (Mbps)	Include MAC Block	Number of Antenna-Carrier Interfaces		
Cyclone IV GX	614.4	Yes	0	3929	16
			1	9687	28
			2	9982	30
			3	10197	32
			4	10458	34
		No	0	3560	16
			1	5889	20
			2	6194	22
			3	6413	24
			4	6710	26
	1228.8, 2457.6, 3072	Yes	0	7785	24
			1	9738	28
			8	11632	42
			16	13766	58
			20	14878	66
		No	0	3927	16
			1	5917	20
			8	7722	34
			16	9895	50
			20	10976	58

Table 1–5 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 1–5. Slowest Recommended Device Family Speed Grades *(Note 1)* (Part 1 of 2)

Device Family	CPRI Line Rate (Mbps)					
	614.4	1228.8	2457.6	3072.0	4915.2	6144
Arria II GX	-6	-6	-6	-6	I3 *(2)*	I3 *(2)*
Arria II GZ	-4	-4	-4	-4	-3	-3
Cyclone IV GX	C8, I7	C8, I7	C8, I7	-7	*(3)*	*(3)*

Table 1–5. Slowest Recommended Device Family Speed Grades ***(Note 1)*** **(Part 2 of 2)**

Device Family	CPRI Line Rate (Mbps)					
	614.4	1228.8	2457.6	3072.0	4915.2	6144
Stratix IV GX	-4	-4	-4	-4	-4	-3

Notes to Table 1–5:

(1) In this table, 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 MegaCore function 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 1–6 provides information about this release of the CPRI MegaCore function.

Table 1–6. CPRI Release Information

Item	Description
Version	10.1
Release Date	December 2010
Ordering Code	IP-CPRI
Product ID	00CB
Vendor ID	6AF7

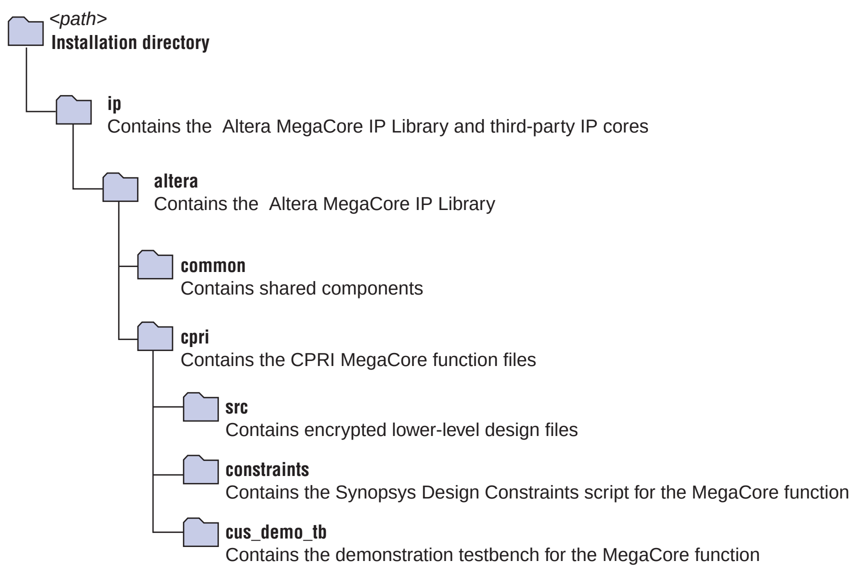
Altera verifies that the current version of the Quartus II software compiles the previous version of each MegaCore function. Any exceptions to this verification are reported in the *MegaCore IP Library Release Notes and Errata*. Altera does not verify compilation with MegaCore function versions older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

Installation and Licensing

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

Figure 1–3 shows the directory structure after you install the CPRI MegaCore function, 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 MegaCore function in simulation and in hardware before you purchase a license. You must purchase a license for the MegaCore function 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 MegaCore function, 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 MegaCore function 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 MegaCore functions
- 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 MegaCore functions, 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 MegaCore function 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, MegaCore functions, and megafunctions. The CPRI MegaCore function cannot achieve frame synchronization, and cannot participate in further CPRI communication.

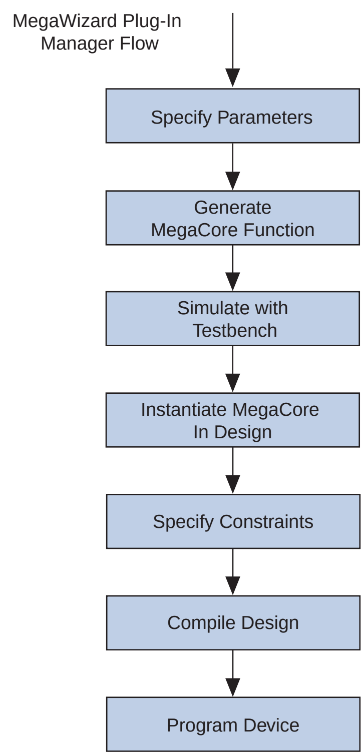
For Information About	Refer To
Installation and licensing	*Altera Software Installation and Licensing*
Open Core Plus	*AN 320: OpenCore Plus Evaluation of Megafunctions*

MegaWizard Plug-in Manager Design Flow

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

Figure 2–1 shows the stages for creating a system with the CPRI MegaCore function 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 MegaCore function, and manually integrate the function in your design.

Specify Parameters

To specify CPRI MegaCore function parameters using the MegaWizard Plug-in Manager, follow these 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 MegaCore function variation.

   To select the CPRI MegaCore function, click **Installed Plug-Ins** > **Interfaces > CPRI**.

3. Specify the parameters. For details about these parameters, refer to Chapter 3, Parameter Settings.
4. Click **Finish** to generate the MegaCore function and supporting files.

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

5. If you generate the CPRI MegaCore function 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 MegaCore function or system in the Quartus II compiler. The parameter editor generates a single **.qip** file for each MegaCore function.

You can now integrate your custom MegaCore function variation in your design, simulate, and compile.

When you integrate your CPRI MegaCore function variation in your design, note the following connection and I/O assignment requirements:

- 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.
- In Arria II GX, Arria II GZ, Cyclone IV GX, and Stratix IV GX designs, you must 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 MegaCore function to the dynamic reconfiguration block, in the ALTGX MegaWizard Plug-in Manager, on the **Reconfiguration Settings** tab, turn on **Analog controls**.

- After you generate the system, Altera recommends that you create assignments for the high-speed transceiver VCCH settings by performing the following steps:
  a. In the Quartus II window, on the Assignments menu, click **Assignment Editor**.
  b. In the **<<new>>** cell in the **To** column, type the top-level signal name for your CPRI MegaCore function instance `gxb_txdataout` signal.
  c. Double-click in the **Assignment Name** column and click **I/O Standard**.
  d. Double-click in the **Value** column and click your standard (for example, **1.5-V PCML)**.
  e. In the new **<<new>>** row, repeat steps b to d for your CPRI MegaCore function instance `gxb_rxdatain` signal.

Simulate the Design

During the design process, to check your design quickly, you can simulate your CPRI MegaCore function variation using the IP functional simulation model and the VHDL demonstration testbench. The IP functional simulation model and testbench files are generated in your project directory. The directory also includes 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 MegaCore function.

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 MegaCore function. You can use this model with the VHDL demonstration testbench for simulation using a mixed-language simulator.

For Information About	Refer To
Quartus II software	See the Quartus II Help topics: “About the Quartus II Software” “About the MegaWizard Plug-In Manager”
MegaWizard Plug-in Manager	
A complete list of models or libraries required to simulate the CPRI MegaCore function	**compile[**_<*variation*>**]**_<*HDL*>**.do** scripts provided with the demonstration testbenches described in Chapter 7, Testbenches
IP functional simulation models	*Simulating Altera Designs* chapter in volume 3 of the *Quartus II Handbook*

Specify Constraints

Altera provides a Synopsys Design Constraints (**.sdc**) file that you must apply to ensure that the CPRI MegaCore function meets design timing requirements. In most cases the script requires modification for your 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*.

Compile and Program

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.

If your design is not part of a Quartus II project, it is a standalone design. Before compiling your standalone design in the Quartus II software, you must assign CPRI I/O signals to virtual pins.

For Information About	Refer To
Compiling your design	*Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*
Programming the device	Device Programming section in volume 3 of the *Quartus II Handbook*

Instantiate Multiple CPRI MegaCore Functions

If you want to instantiate multiple CPRI MegaCore functions, you must complete a few additional steps. When your design contains multiple MegaCore functions, you must ensure that the `gxb_cal_blk_clk` input and `gxb_powerdown` signals are connected properly, and that the instances each have different starting channel numbers.

You must ensure that the `gxb_cal_blk_clk` input to each CPRI MegaCore function (or any other megafunction or user logic that uses the ALTGX megafunction) is driven by the same calibration clock source.

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

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

3. Parameter Settings

You customize the CPRI MegaCore function 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 MegaCore function. To customize your CPRI MegaCore function, you can modify parameters to specify the following MegaCore function properties:

- Line rate
- Whether this CPRI MegaCore function instance is configured with slave clocking mode (RE slave) or with master clocking mode (REC or RE master).
- Starting channel number
- Number of antenna-carrier interfaces
- Whether to enable auto-rate negotiation
- Whether to include an internal Ethernet MAC block or provide a Media Independent Interface (MII)-like interface to connect to an external Ethernet module

Line Rate Parameter

The **Line rate** parameter specifies the line rate on the CPRI link in gigabits per second (Gbps). Cyclone IV GX devices support 0.6144, 1.2288, 2.4576, and 3.072 Gbps line rates. Arria II GX, Arria II GZ, and Stratix IV GX devices support 0.6144, 1.2288, 2.4576, 3.072, 4.9152, and 6.144 Gbps line rates.

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 will probably fail, and the design cannot meet timing in hardware.

Operation Mode Parameter

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

Transceiver Starting Channel Number

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

For a CPRI MegaCore function configured with slave clocking mode, the **Slave transmitter starting channel number** and **Slave receiver starting channel number** do the same. Both numbers must be 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 MegaCore functions on the same device, you must ensure each uses distinct transceiver channels.

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 MegaCore function. The supported values are 0 to 24. Set this parameter to the maximum number of data channels you expect your CPRI MegaCore function to use at the same time.

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

Software allows you to specify that some of the antenna-carrier interfaces that you configure in your CPRI MegaCore function are not active. This feature allows you to change the number of active and enabled data channels dynamically.

The combination of CPRI MegaCore function line rate, sampling width, and sampling rate restricts the number of active antenna-carrier interfaces your CPRI MegaCore function can support. For example, if your CPRI MegaCore function operates at line rate 3.072 Gbps, it can support as many as 20 active antenna-carrier interfaces, but if your CPRI MegaCore function 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 4–26.

☞ 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, then 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 MegaCore function, refer to "CPRI MAP Interface Module" on page 4–22.

Enable Auto-Rate Negotiation

Auto-rate 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 rate. If your CPRI MegaCore function has auto-rate 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 MegaCore function 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 auto-rate negotiation feature, refer to "Auto-Rate Negotiation" on page 4–21 and Appendix B, Implementing CPRI Link Auto-Rate Negotiation.

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

Include MAC Block

Turn on the **Include MAC block** parameter to specify that your CPRI MegaCore function includes an internal Ethernet MAC block. By default, this parameter is not turned on. If this parameter is not turned on, the CPRI MegaCore function implements the MII interface to your own external Ethernet MAC, instead.

For information about the internal Ethernet MAC block, refer to "Data Link Layer for Fast Control and Management Channel (Ethernet)" on page 4–49.

For information about the MII interface, refer to "MII Interface to an External Ethernet Block" on page 4–52.

4. Functional Description

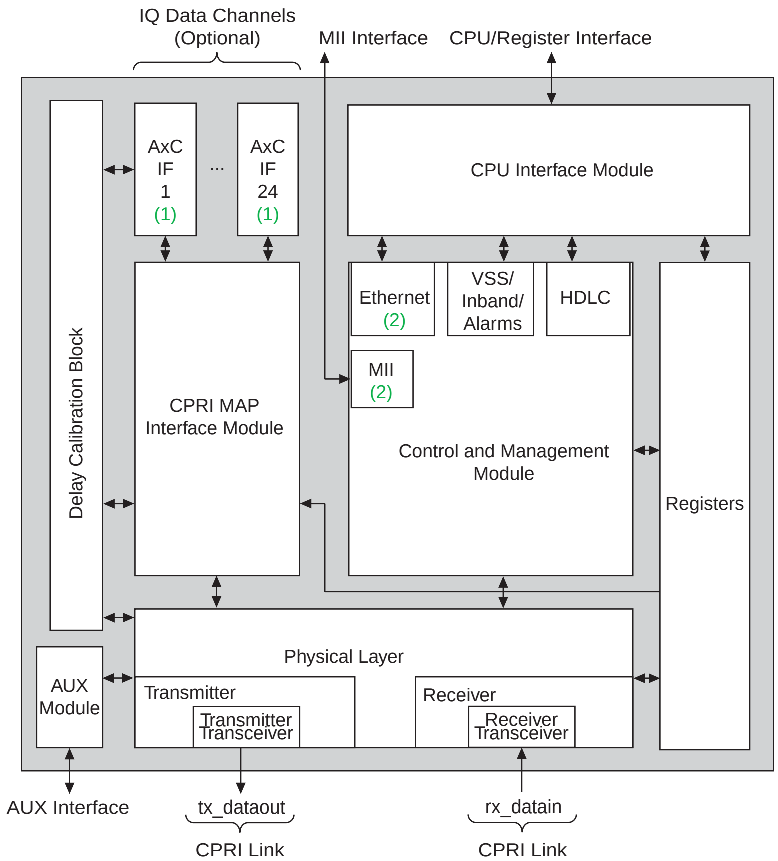
The CPRI specification divides the protocol into a physical layer (Layer 1) and a data link layer (Layer 2). Layer 1 is implemented in the CPRI interface module. This chapter describes the individual interfaces of the CPRI MegaCore function and how data passes between them.

This chapter contains the following sections:

Architecture Overview

Figure 4–1 shows the main blocks of the CPRI MegaCore function.

Figure 4–1. CPRI MegaCore Function Block Diagram



Note to Figure 4–1:

(1) You can configure your CPRI MegaCore function with zero, one, or multiple IQ data channels.

(2) You can configure your CPRI MegaCore function with an Ethernet MAC block or an MII block.

The following sections describe the individual interfaces and clocks.

Interfaces Overview

The Altera CPRI MegaCore function supports the following interfaces:

- CPRI Interface
- CPU Interface
- MAP Interface
- Auxiliary Interface
- MII Interface

CPRI Interface

The CPRI interface complies with the CPRI Specification V4.1 Interface Specification. The protocol is divided into a two-layer hierarchy: Physical layer and Data Link layer. The specification describes three communication planes: user data, control and management (C&M), and timing synchronization information.

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

CPU Interface

The CPRI MegaCore function communicates with an on-chip or external processor through its CPU interface. Use this interface to communicate Control and Management (C&M) information and for High-Level Data Link Controller (HDLC) or Ethernet communication with an internal MAC block. 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 CPRI MegaCore function does not implement arbitration among the modules that connect to it through the CPU interface.

The CPU interface is implemented as an Avalon-MM slave interface. The Avalon-MM slave executes transfers between the CPRI MegaCore function and the user-defined logic in your design. The CPU interface is the only Avalon-MM interface implemented by the CPRI MegaCore function.

For information about the CPU interface, refer to "CPU Interface Module" on page 4–22.

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

MAP Interface

The CPRI MAP interface comprises the individual antenna-carrier interfaces, or data channels, through which the CPRI MegaCore function transfers IQ sample data to and from the RF implementation. The CPRI 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 CPRI MAP interface, refer to "CPRI MAP Interface Module" on page 4–22.

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

Auxiliary Interface

The Auxiliary (AUX) interface allows you to connect 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 V4.1 Specification, outside the CPRI MegaCore function. The AUX interface supports the transmission and reception of IQ data and timing information between an RE master and an RE slave, 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 MegaCore function MAP interface to implement custom mapping algorithms outside the MegaCore function. The CPRI MegaCore function implements the AUX interface as one incoming and one outgoing Avalon-ST interface.

For more information about how this interface functions with the CPRI MegaCore function, refer to "Auxiliary Interfaces" on page 4–33.

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

MII Interface

The MII interface allows the CPRI MegaCore function to communicate directly with an external Ethernet MAC block, bypassing the internal Ethernet and HDLC implementation that communicates through the CPU Interface. You specify in the CPRI parameter editor whether to implement this interface or to use the Ethernet or HDLC MAC block available with the CPRI MegaCore function. If you configure the CPRI MegaCore function with the MII interface, you must implement the Ethernet MAC block outside the CPRI MegaCore function. For more information, refer to "MII Interface to an External Ethernet Block" on page 4–52.

Clocking and Reset Structure

The CPRI MegaCore function has a variable number of clock domains, depending on the number of antenna-carrier interfaces. In addition to the high-speed clock domains inside the Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX transceiver, the CPRI MegaCore function contains three basic clock domains, two clock domains for the MII interface if it is implemented, and two clock domains for each antenna-carrier interface.

You can configure a CPRI MegaCore function in master or slave clocking mode. REC configurations and RE master configurations use master clocking mode, and RE slave configurations use slave clocking mode.

The top-level blocks shown in "Architecture Overview" on page 4–2 define the clock domain boundaries. The clocking diagrams in Figure 4–2 on page 4–7 to Figure 4–5 on page 4–10 show the details.

MegaCore Function Basic Clock Domains

Each CPRI MegaCore function has the following three basic clock domains:

- `cpri_clkout`—Main clock for the CPRI MegaCore function. This clock is derived from the transceiver transmit PLL, and its frequency depends on the CPRI line rate. For more information about this correspondence, refer to "CPRI Communication Link Line Rates" on page 4–12.
- `clk_ex_delay`—Clock for extended delay measurement. For more information about this clock, refer to "Extended Rx Delay Measurement" on page 4–42.
- `cpu_clk`—Clock that controls the input to the CPU interface of the CPRI MegaCore function and drives the CPU interface module.

The cpri_clkout and cpu_clk clocks are assumed to be asynchronous. The cpu_clk maximum value is constrained by f_{MAX} and can vary based on the family and speed grade.

High-Speed Transceiver Clocks

The following input clocks are used by the high-speed transceiver on the CPRI MegaCore function CPRI interface:

- gxb_refclk—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—Calibration-block clock.
- reconfig_clk—Dynamic reconfiguration block clock.
- gxb_pll_inclk—Input clock to the transmitter PLL in a CPRI MegaCore function configured in slave clocking mode. If the CPRI MegaCore function 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 clock connects to the pll_inclk input signal of the Arria II GX, Arria II GZ, or Stratix IV GX transceiver's PLL, and the gxb_refclk clock connects to the rx_cruclk input signal of the transceiver. In master clocking mode, the CPRI MegaCore function connects the gxb_refclk clock to the pll_inclk input signal of the transceiver, and does not use the gxb_pll_inclk input signal. Refer to "Clock Diagrams for the CPRI MegaCore Function" on page 4–6.

In master clocking mode, the two transceiver signals pll_inclk and rx_cruclk are implemented as a single input signal to the ALTGX megafunction, named pll_inclk_rx_cruclk. However, the clocking mode implementation ensures that the CPRI MegaCore function signal is interpreted correctly.

For more 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*.

MII Interface Clock Domains

The MII interface has the following two output clocks:

- cpri_mii_txclk—Clocks the MII interface transmitter module.
- cpri_mii_rxclk—Clocks the MII interface receiver module.

Both clocks have the same frequency as the cpri_clkout clock. The frequency depends on the CPRI line data rate. Refer to Table 4–1 on page 4–12.

MAP Interface Clock Domains

Each antenna-carrier interface has the following two clocks:

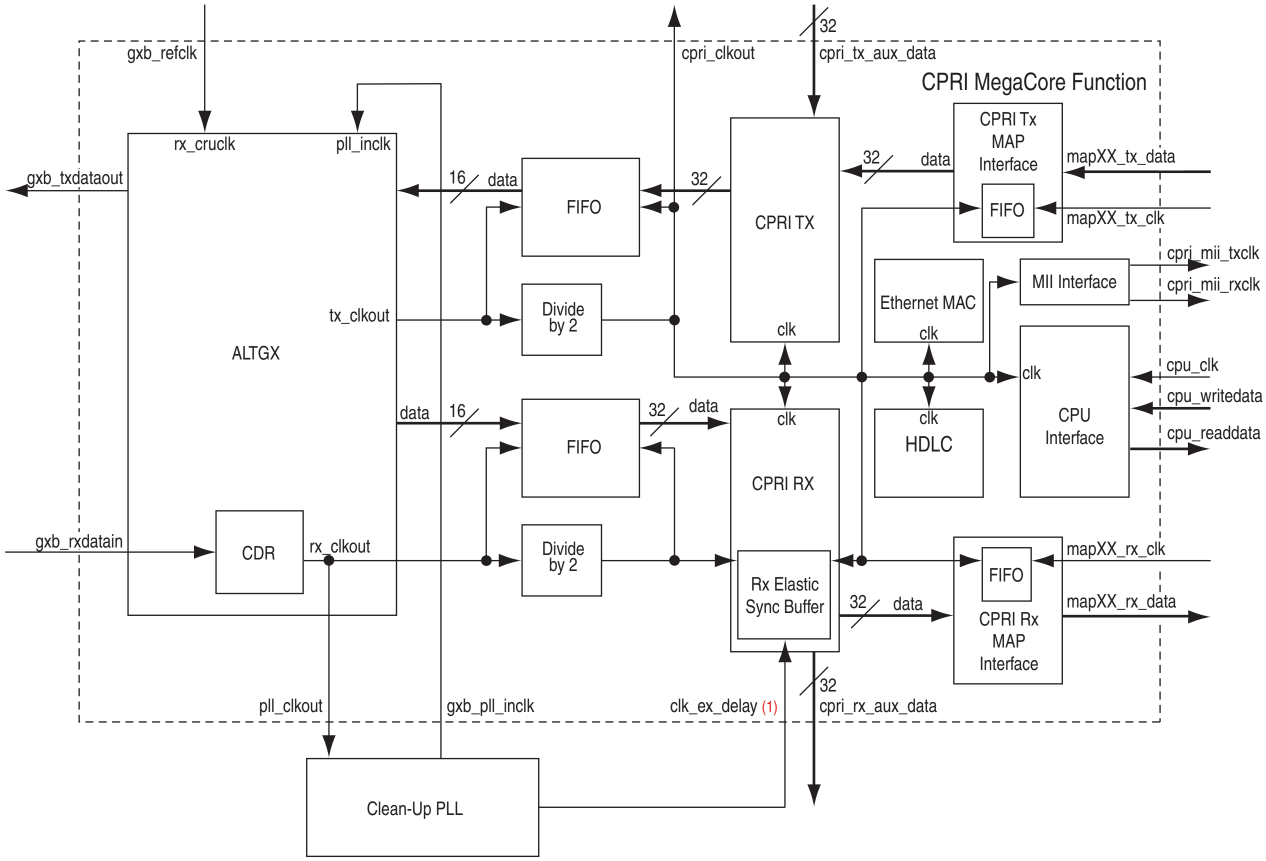
- `mapN_tx_clk`—Expected rate of received data on this antenna-carrier interface. The frequency of this clock is the sample rate on the incoming antenna-carrier interface.
- `mapN_rx_clk`— Clocks the transmissions of this antenna-carrier interface. The frequency of this clock is the sample rate on the outgoing antenna-carrier interface. For more information about data channel sample rates, refer to Table 4–4 and Table 4–5 on page 4–26.

Clock Diagrams for the CPRI MegaCore Function

Figure 4–2 to Figure 4–5 show the clocking schemes for CPRI MegaCore functions configured as RE slaves, RE masters, and REC masters in Arria II GX, Arria II GZ, Cyclone IV GX, and Stratix IV GX devices with CPRI line rate greater than 0.6144 Gbps. Figure 4–6 and Figure 4–7 show the clock diagrams for CPRI MegaCore functions configured as RE slaves, RE masters, and REC masters in all four device families with CPRI line rate 0.6144 Gbps.

Figure 4–2 shows the clock diagram for a CPRI MegaCore function configured as an RE slave with CPRI line rate greater than 0.6144 Gbps in an Arria II GX or Cyclone IV GX device.

Figure 4–2. CPRI MegaCore Function Slave Clocking in Arria II GX and Cyclone IV GX Devices



Note to Figure 4–2:

(1) The `clk_ex_delay` input clock can be driven by a cleanup PLL. However, it can also be driven by other clock logic that provides the correct M/N ratio for the accuracy required by the application. Refer to "Extended Rx Delay Measurement" on page 4–42.

Figure 4–3 shows the clock diagram for a CPRI MegaCore function configured as an REC or as an RE master with CPRI line rate greater than 0.6144 Gbps in an Arria II GX or Cyclone IV GX device.

Figure 4–3. CPRI MegaCore Function Master Clocking in Arria II GX and Cyclone IV GX Devices

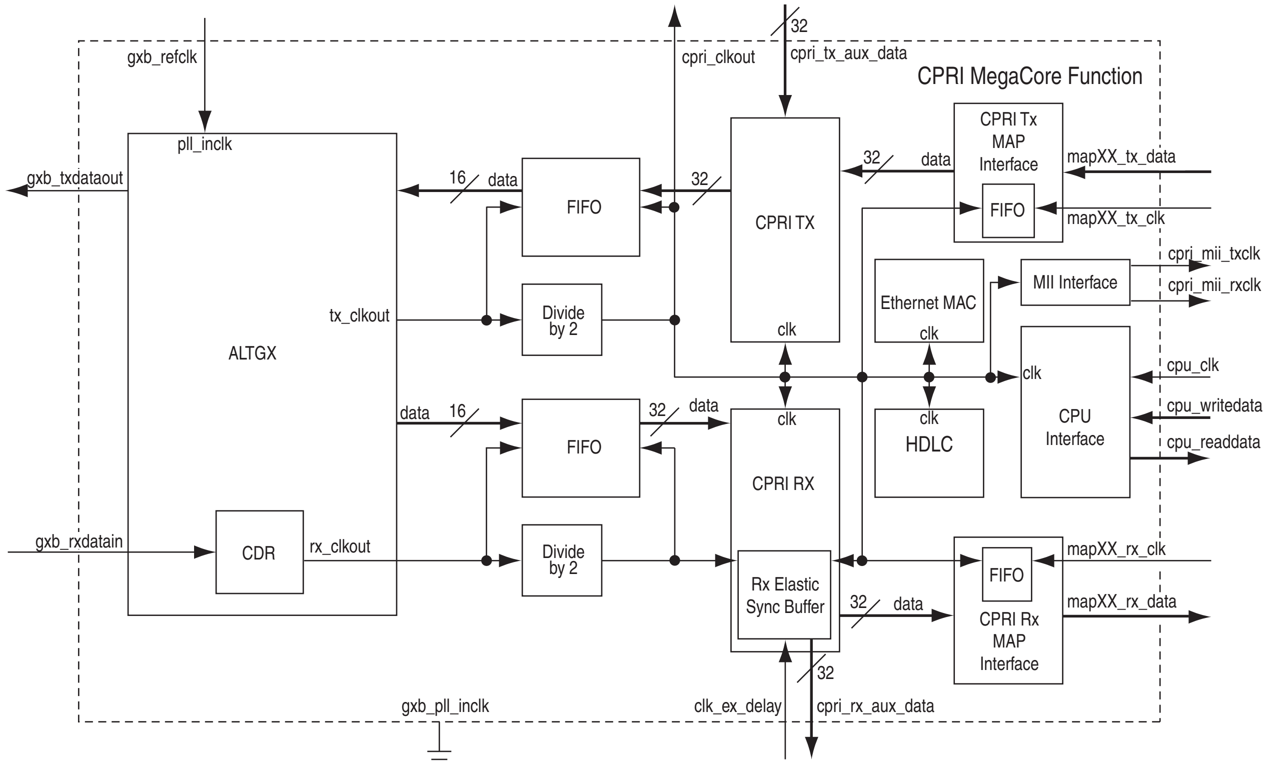
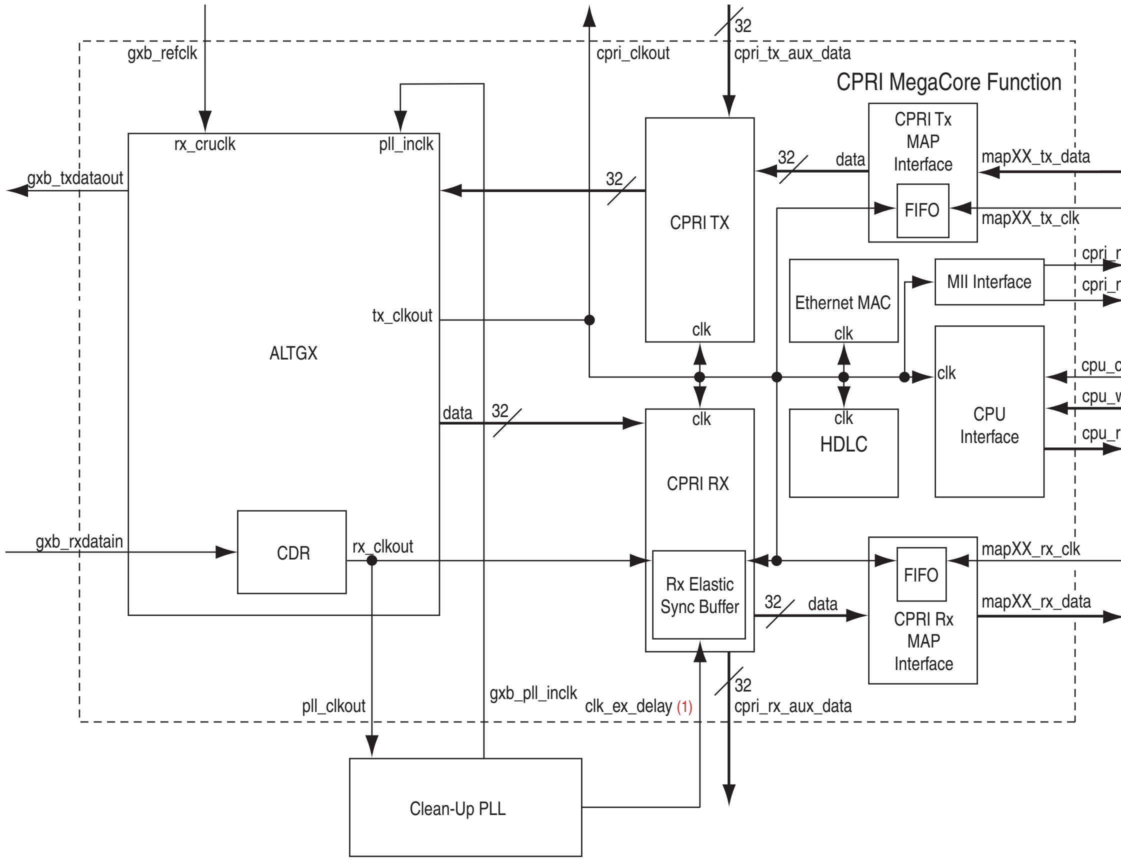


Figure 4–4 shows the clock diagram for a CPRI MegaCore function configured as an RE slave with CPRI line rate greater than 0.6144 Gbps in an Arria II GZ or Stratix IV GX device.

Figure 4–4. CPRI MegaCore Function Slave Clocking in Arria II GZ and Stratix IV GX Devices



Note to Figure 4–4:

(1) The `clk_ex_delay` input clock can be driven by a cleanup PLL. However, it can also be driven by other clock logic that provides the correct M/N ratio for the accuracy required by the application. Refer to "Extended Rx Delay Measurement" on page 4–42.

Figure 4–5 shows the clock diagram for a CPRI MegaCore function configured as an REC or as an RE master with CPRI line rate greater than 0.6144 Gbps in an Arria II GZ or Stratix IV GX device.

Figure 4–5. CPRI MegaCore Function Master Clocking in Arria II GZ and Stratix IV GX Devices

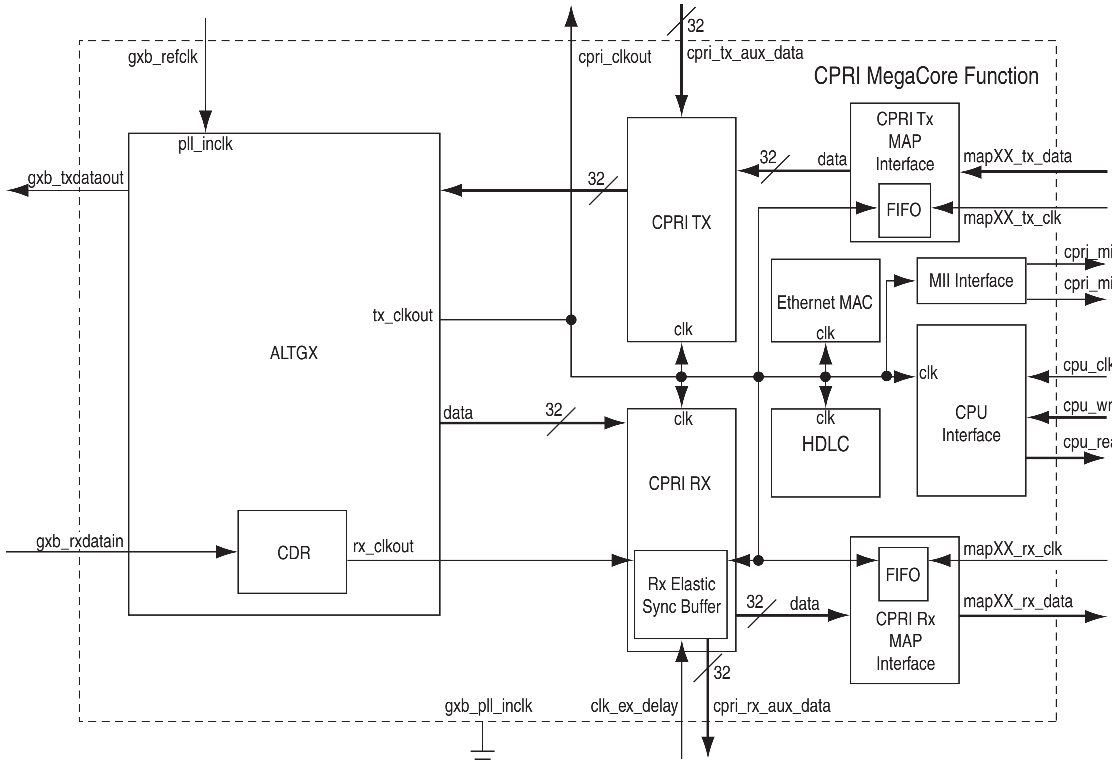
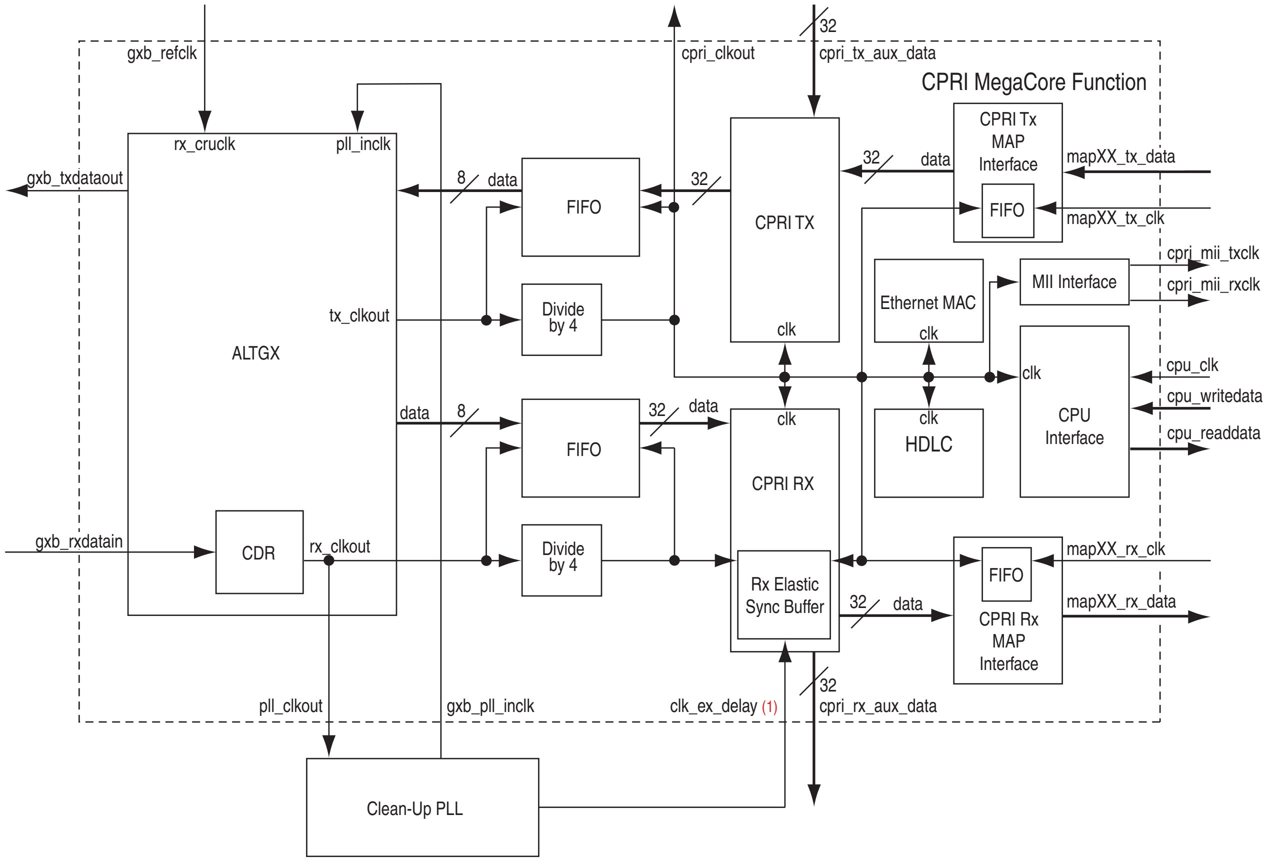


Figure 4–6 shows the clock diagram for a CPRI MegaCore function configured as an RE slave with CPRI line rate 0.6144 Gbps in an Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX device.

Figure 4–6. CPRI MegaCore Function Slave Clocking at CPRI Line Rate 0.6144 Gbps

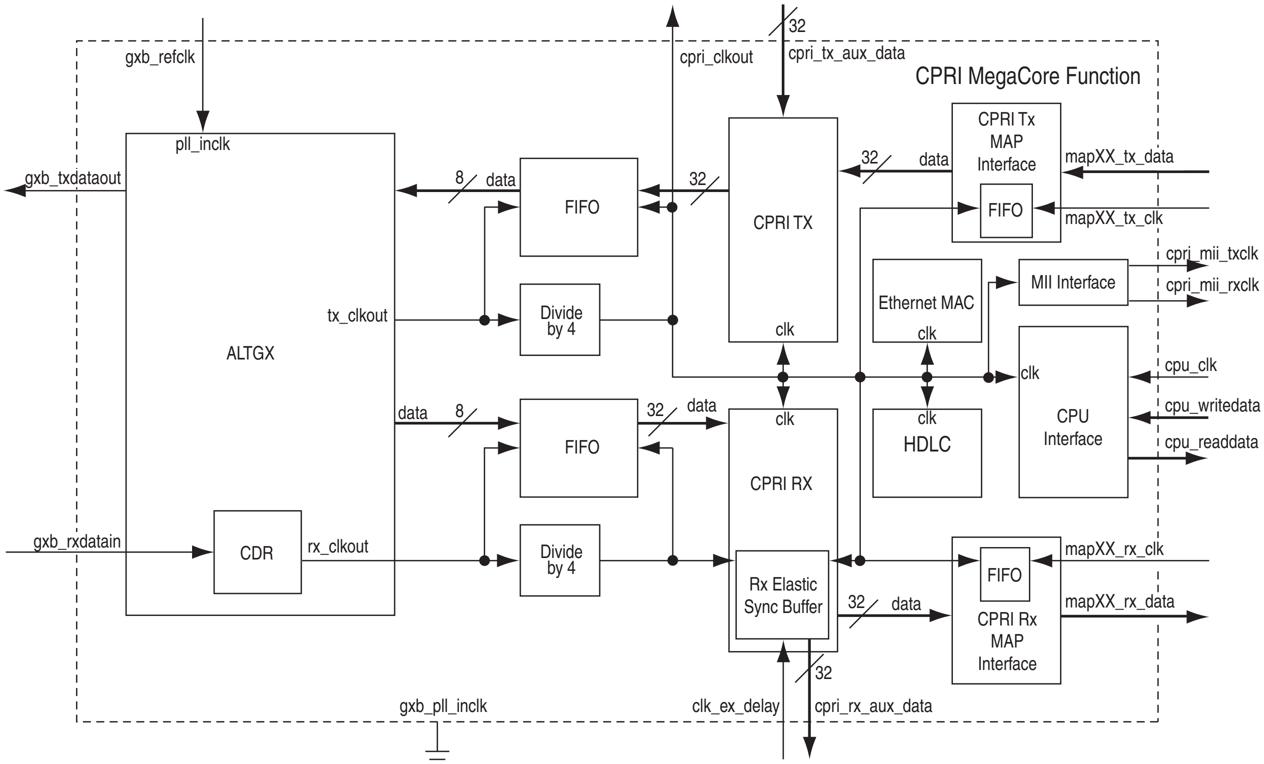


Note to Figure 4–6:

(1) The `clk_ex_delay` input clock can be driven by a cleanup PLL. However, it can also be driven by other clock logic that provides the correct M/N ratio for the accuracy required by the application. Refer to "Extended Rx Delay Measurement" on page 4–42.

Figure 4–7 shows the clock diagram for a CPRI MegaCore function configured as an REC or as an RE master with CPRI line rate 0.6144 Gbps in an Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX device.

Figure 4–7. CPRI MegaCore Function Master Clocking at CPRI Line Rate 0.6144 Gbps



CPRI Communication Link Line Rates

The CPRI specification specifies line rates of n × 614.4 Mbps for n = 1 to n = 10. The CPRI MegaCore function implements line rates of n × 614.4 Mbps for n in {1,2,4,5,8,10}. Cyclone IV GX devices support line rates of n × 614.4 Mbps only for n in {1,2,4,5}. Table 4–1 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 4–1. CPRI Link Line Rates and Clock Rates for CPRI MegaCore Function (Part 1 of 2)

Line Rate (Mbps)	Clock Frequency (MHz)					
	Default gxb_refclk Frequency			cpri_clkout Frequency (If line rate is supported)	pll_clkout Frequency (If line rate is supported)	
	In Arria II GZ and Stratix IV GX Devices	In Arria II GX Devices	In Cyclone IV GX Devices		In Arria II GX and Cyclone IV GX Devices	In Arria II GZ and Stratix IV GX Devices
614.4	61.44	61.44	61.44	15.36	61.44	61.44
1228.8	61.44	61.44	61.44	30.72	61.44	30.72

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

Line Rate (Mbps)	Clock Frequency (MHz)					
	Default gxb_refclk Frequency			cpri_clkout Frequency (If line rate is supported)	pll_clkout Frequency (If line rate is supported)	
	In Arria II GZ and Stratix IV GX Devices	In Arria II GX Devices	In Cyclone IV GX Devices		In Arria II GX and Cyclone IV GX Devices	In Arria II GZ and Stratix IV GX Devices
2457.6	61.44	122.88	122.88	61.44	122.88	61.44
3072	76.80	153.60	153.60	76.80	153.60	76.80
4915.2	122.88	245.76 *(1)*	—	122.88	245.76	122.88
6144	153.60	307.20 *(1)*	—	153.60	307.20	153.60

Note to Table 4–1:

(1) The CPRI MegaCore function supports CPRI line rates 4915.2 Mbps and 6144 Mbps in variations that target Arria II GX devices only for speed grade I3 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. The datapath width is determined by device family, and is shown in Figure 4–2 through Figure 4–7.

The clock `gxb_refclk` is the incoming reference clock for the device transceiver's PLL. When you generate a CPRI MegaCore function variation, you generate an ALTGX megafunction with specific default settings. These default transceiver settings configure a transceiver that works correctly with the CPRI MegaCore function when the input `gxb_refclk` clock has the frequency shown in Table 4–1. 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.

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 MegaCore function line rate. The ALTGX parameter editor lets you select one of the supported frequencies.

MegaCore Function Reset Process

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

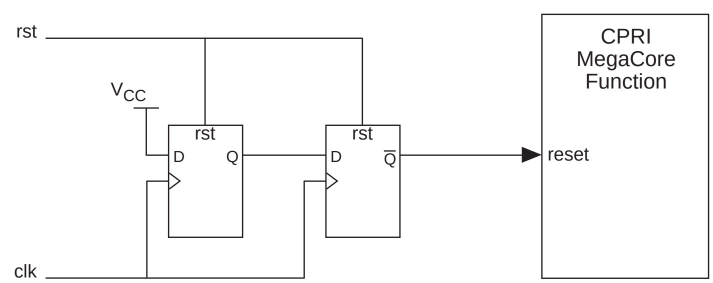
You can assert all reset signals asynchronously to any clock. However, each reset signal must be asserted for at least one full clock period of a specific clock, and be deasserted synchronously to the rising edge of that clock. For example, the CPU interface reset signal, `cpu_reset`, must be deasserted on the rising edge of `cpu_clk`. Table 4–2 shows the reset signals and their corresponding clock domains.

Table 4–2. Reset Signals and Corresponding Clock Domains

Reset Signal	Clock Domain	Description
reset	reconfig_clk	Resets the CPRI interface
gxb_powerdown	—	Powers down and resets the high-speed transceiver block. For setup and hold times, refer to the relevant device handbook.
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
mapN_tx_reset	mapN_tx_clk	Resets the MAP Channel N transmitter block

You must implement logic to ensure the minimal hold time and synchronous deassertion of each reset input signal to the CPRI MegaCore function. Figure 4–8 shows a circuit that ensures these conditions for one reset signal.

Figure 4–8. Circuit to Ensure Synchronous Deassertion of Reset Signal



For more information about the requirements for reset signals, refer to Chapter 5, Signals.

Reset Controller

The CPRI MegaCore function has a dedicated reset control module to handle the specific requirements of the high-speed transceiver module. This module generates the recommended reset sequence for the transceiver. The `reset` signal controls the reset control module.

Reset Control Word Communicated on CPRI Link

In addition, a CPRI MegaCore function can receive or send a reset request through the CPRI link. You use the CPRI MegaCore function `CPRI_HW_RESET` register, and optionally the `hw_reset_assert` input signal, to control and monitor the reset control word sent in CPRI communication. As dictated 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 is used for both reset request and reset acknowledge.

A CPRI MegaCore function in master mode transmits a reset request to the RE slave nodes to which it is connected under either of the following conditions:

- The `reset_gen_en` and `reset_gen_force` bits in the `CPRI_HW_RESET` register are set, and the `reset_hw_en` bit in the `CPRI_HW_RESET` register is not set.
- The `hw_reset_assert` input signal is asserted while the `reset_hw_en` bit in the `CPRI_HW_RESET` register is set.

The behavior of a CPRI MegaCore function 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, the `reset_hw_en` bit also takes precedence over the `reset_gen_en` bit. If the `reset_hw_en` bit is asserted, the `reset_gen_en` bit is ignored.

The following sections describe the CPRI MegaCore function behavior in sending and receiving reset requests and reset acknowledgements under the two different sets of conditions.

CPRI Link Reset Requests and Acknowledgements Based on reset_gen_force Register Field

The CPRI specification dictates that the Z.130.0 reset bit must be detected by the CPRI partner in four consecutive hyperframes before the CPRI partner confirms the reset request. The reset generation request is in effect while `reset_gen_force` remains set, until the reset acknowledge control bit is detected on the incoming CPRI link, as long as the `reset_gen_en` bit remains high.

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.

A CPRI MegaCore function in slave mode indicates that it detects a reset request sent in CPRI communication by setting the `reset_detect` and `reset_detect_hold` bits of the `CPRI_HW_RESET` register. If the `reset_gen_en` bit is set (and the `reset_hw_en` bit is not set), the CPRI transmitter sends a reset acknowledge on the CPRI link, by setting the Z.130.0 reset bit in ten consecutive outgoing hyperframes. If the `reset_out_en` bit is set, the CPRI MegaCore function 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 ten consecutive reset acknowledge bits, the CPRI transmitter sets the `reset_gen_done` and `reset_gen_done_hold` bits.

For more information about the `CPRI_HW_RESET` register, refer to Table 6–12 on page 6–5.

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

CPRI Link Reset Requests and Acknowledgements Based on hw_reset_assert Input Signal

The CPRI specification dictates that the Z.130.0 reset bit must be detected by the CPRI partner in four consecutive hyperframes before the CPRI partner confirms the reset request. The reset generation request is in effect while `hw_reset_assert` remains asserted, until the reset acknowledge control bit is detected on the incoming CPRI link, as long as the `reset_hw_en` bit remains high.

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.

A CPRI MegaCore function in slave mode indicates that it detects a reset request sent in CPRI communication by setting the `reset_detect` and `reset_detect_hold` bits of the `CPRI_HW_RESET` register.

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 ten consecutive outgoing hyperframes. If the `reset_hw_en` bit of the `CPRI_HW_RESET` register is not set, the CPRI transmitter sends a reset acknowledge only if the conditions described in "CPRI Link Reset Requests and Acknowledgements Based on reset_gen_force Register Field" hold. If the `reset_hw_en` bit of the `CPRI_HW_RESET` register is set, the CPRI transmitter can send the reset acknowledge only if the application asserts the `hw_reset_assert` signal. If the `reset_out_en` bit of the `CPRI_HW_RESET` register is set, the CPRI MegaCore function asserts the external `hw_reset_req` signal until the reset occurs. This signal informs the application layer of the low-level reset request. 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.

After it transmits the ten consecutive reset acknowledge bits, the CPRI transmitter sets the `reset_gen_done` and `reset_gen_done_hold` bits.

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

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

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 interface module of the CPRI MegaCore function 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 interface module. Figure 4–9 shows a high-level block diagram of this module.

Features

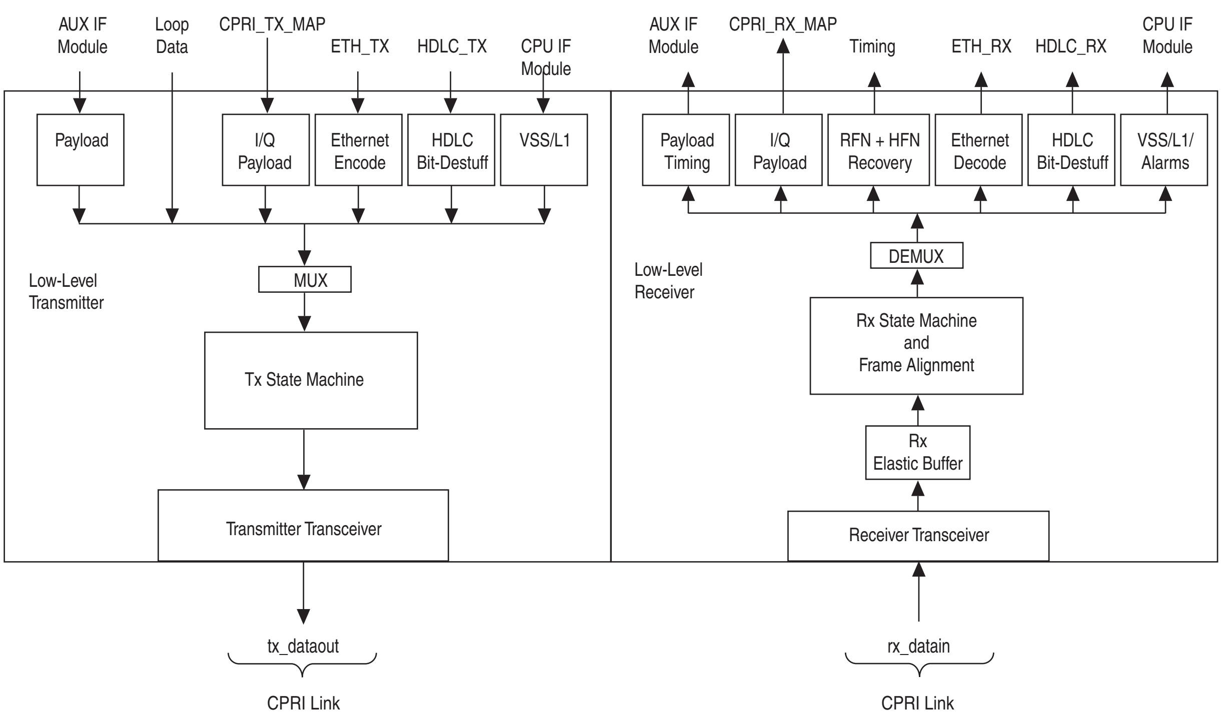
The Physical layer has the following features:

- 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/control word assembly and delineation
  - Error detection
  - Deterministic latency
- Software interface (status/control registers)
- Error reporting
- Clock decoupling

Physical Layer Architecture

Figure 4–9 shows the architecture of the Physical layer.

Figure 4–9. Physical Layer High Level Block Diagram



Low-level Interface Receiver

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

- Converts the data to the main clock domain
- Performs CPRI frame detection, supporting auto-rate negotiation
- Separates data and control words
- Optionally descrambles data at 4195.2 Mbps and 6144.0 Mbps CPRI line rates
- Separates data for the CPRI 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 transceiver is an embedded ALTGX megafunction in the Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX device. 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.

Rx Elastic Buffer

The low-level interface receiver converts data from the transceiver clock domain to the main CPRI MegaCore function clock domain 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, divided by one, two, or four, depending on the datapath width in the transceiver. The width of an Rx elastic buffer entry is 32 bits, and the `rx_clkout` clock is divided with the transceiver data conversion to 32-bit words. For details, refer to "Clock Diagrams for the CPRI MegaCore Function" on page 4–6.

Table 4–3 shows the `rx_clkout` clock divider for the different device families and CPRI data rates.

Table 4–3. Transceiver Datapath Width and rx_clkout Divider for Input to Rx Elastic Buffer

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

The depth of the Rx elastic buffer is 64 in REC configurations and 16 in RE configurations. 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.

You must realign and resynchronize the Rx elastic buffer after a dynamic CPRI line rate change. Because resynchronizing the Rx elastic buffer resets its pointers, you must ensure that the Rx elastic buffer is empty before it is resynchronized. 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 MegaCore function. Refer to "Extended Rx Delay Measurement" on page 4–42.

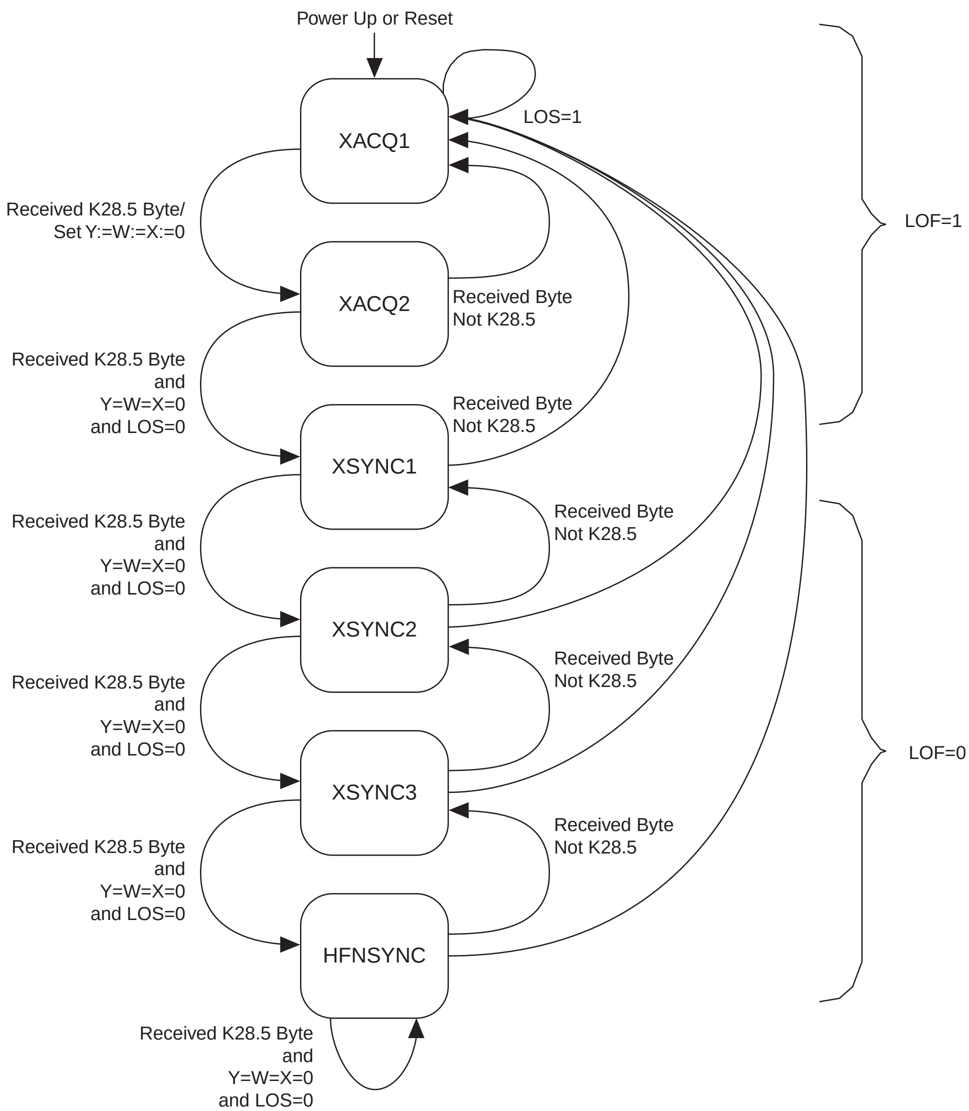
Descrambling

If the tx_prot_version field of the CPRI_TX_PROT_VER register (Table 6–25 on page 6–11) holds the value 2, and the CPRI data rate is 4195.2 Mbps or 6144.0 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 6–27 on page 6–12) is set, the low-level CPRI receiver descrambles the data words according to the CPRI V4.1 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.

Performing Frame Synchronization

During frame synchronization, LOF is set to zero. LOS—the assertion of the gxb_los signal—resets the frame synchronization state machine. Auto-rate negotiation occurs in the first stage of frame synchronization, XACQ1. Figure 4–10 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 V4.1 Specification. The CPRI MegaCore function sets the values in the CPRI_RX_SCR_SEED register according to these conditions.

Figure 4–10. CPRI Frame Synchronization Machine *(Note 1)*



Note for Figure 4–10:

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

Recording the Incoming Control Bytes

A control receive table contains a 1-byte entry for each of the 256 control words in the current hyperframe. The control receive table entries are updated only when the frame synchronization state machine is in the HFNSYNC state, in which hyperframe synchronization has been performed successfully. To read a control byte, write the frame 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.

Auto-Rate Negotiation

The auto-rate negotiation feature allows the CPRI MegaCore function to 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. If you enable the auto-rate negotiation feature, you can provide dynamic input to the low-level CPRI interface receiver to implement this capability in your design, using logic you implement outside the MegaCore function.

To use this feature, you must include additional external logic in your design. Appendix B, Implementing CPRI Link Auto-Rate Negotiation describes the external logic required to implement auto-rate negotiation in your design.

If you configure your CPRI MegaCore function for auto-rate negotiation, the MegaCore includes two output status signals and a register to collect the status information, as well as the internal support to change CPRI line rate according to your design's input to the transceiver dynamic reconfiguration block. For Cyclone IV GX designs, your design must also provide line rate information to the ALTPLL_RECONFIG megafunction connected to the transceiver.

Low-Level Interface Transmitter

The transmitter in the low-level interface transmits output to the CPRI interface. 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 4195.2 Mbps and 6144.0 Mbps CPRI line rates
- Inserts the following control words in their appropriate location 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

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 enabled 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 each of the bytes in the frame's control word. To write a control byte in the control transmit table, write the frame number X to the `CPRI_CTRL_INDEX` register and then write the next intended #Z.X.0 control byte in the `CPRI_TX_CTRL` register.

Altera recommends that you assert the config_reset signal to clear all control transmit table entries at startup before you set the tx_ctrl_insert_en bit in the CPRI_CONTROL register.

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

When the tx_prot_version field of the CPRI_TX_PROT_VER register (Table 6–25 on page 6–11) holds the value 2, the low-level CPRI transmitter scrambles the data words according to the CPRI V4.1 Specification, using the seed in the tx_scr_seed field of the CPRI_TX_SCR_SEED register (Table 6–26 on page 6–12).

The transceiver is an embedded ALTGX megafunction in the Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX device. 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 or Stratix IV GX 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.

CPU Interface Module

The CPU interface module provides an Avalon-MM slave interface that accesses all registers in the CPRI MegaCore function. This module can communicate with an on-chip or external processor, an Ethernet channel, and an HDLC channel.

The input to the CPU interface port from the processor is synchronized with the cpu_clk clock.

Each of the three sources of input to the CPU interface communicates with the CPRI MegaCore function 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 MegaCore function.

For more information about the CPRI MegaCore function registers, refer to Chapter 6, Software Interface.

CPRI MAP Interface Module

The CPRI MegaCore function communicates with the RF implementations (antenna-carriers) through multiple AxC interfaces, or data channels. A CPRI MegaCore function configured with a MAP interface module can have as many as 24 data channels, and as few as one data channel. If a CPRI MegaCore function 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 MegaCore function configuration. Each data channel communicates with the corresponding RF implementation using two 32-bit Avalon-ST interfaces, one incoming and one outgoing.

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

This section contains the following topics:

- MAP Interface Mapping Modes
- CPRI MAP Receiver Interface
- CPRI MAP Transmitter Interface
- PRBS Generation and Validation

MAP Interface Mapping Modes

The `map_mode` field of the `CPRI_MAP_CONFIG` register determines the mapping mode implemented by your CPRI MegaCore function.

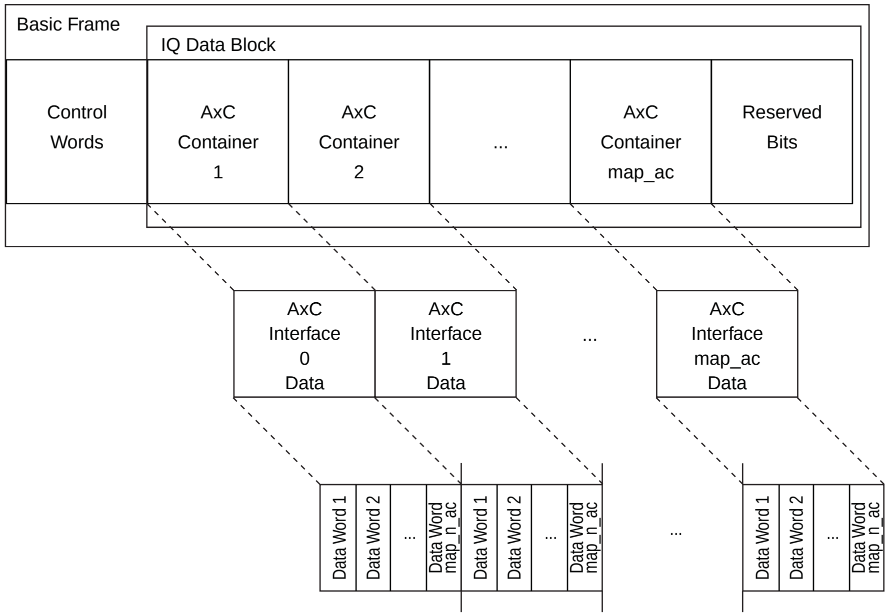
Basic AxC Mapping Mode

In the basic UMTS/LE standard mapping mode, implemented when `map_mode` has value 2'b00, all of the AxC interfaces use the same sample rate and sample width. The CPRI MegaCore function 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 MegaCore function 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 V4.1 Specification. Figure 4–11 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.

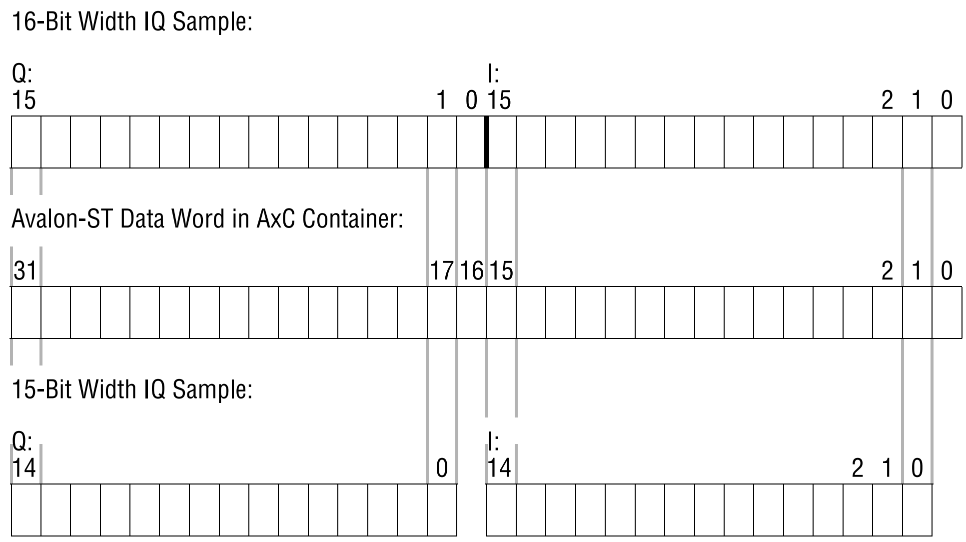
Figure 4–11. CPRI Basic Mapping Mode



The CPRI MegaCore function 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 4–12 shows the bit correspondence for both sample widths.

Figure 4–12. 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 MegaCore function 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 MegaCore function, 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 4–12, 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

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**
CPRI Line Rate (Mbps) [No. Bits per IQ Data Block]	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

Note to Table 4–4:

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

Table 4–5. 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
CPRI Line Rate (Mbps) [No. Bits per IQ Data Block]	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

Note to Table 4–5:

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

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

2 × 16 × map_n_ac × map_ac

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

2 × 15 × map_n_ac × map_ac

This value must be no larger than the number of bits in the IQ data block, shown in square brackets in the leftmost column of 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 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 MegaCore function registers are ignored in basic mapping mode:

- CPRI_MAP_TBL_CONFIG register (Table 6–31 on page 6–14)
- CPRI_MAP_TBL_INDEX register (Table 6–32 on page 6–14)
- CPRI_MAP_TBL_RX register (Table 6–33 on page 6–15)
- CPRI_MAP_TBL_TX register (Table 6–34 on page 6–15)

Advanced AxC Mapping Modes

In the advanced AxC mapping modes, implemented when map_mode has value 2’b01 or 2b’10, 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.

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 V4.1 Specification. Configuration tables define the mapping of AxC containers to offsets in the AxC interface timeslots.

The CPRI MegaCore function supports the following two advanced AxC mapping modes:

- When `map_mode` has value 2'b01, AxC mapping conforms to Method 1: IQ Sample Based, described in Section 4.2.7.2.5 of the CPRI V4.1 Specification.
- When `map_mode` has value 2'b10, AxC mapping conforms to Method 3: Backward Compatible, described in Section 4.2.7.2.7 of the CPRI V4.1 Specification.

Both of the advanced AxC mapping modes comply with the description in Section 4.2.7.2.4 of the CPRI V4.1 Specification.

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.

Each table entry corresponds to a timeslot, which is a 32-bit word on the AxC interface, in one AxC container block. In 16-bit width mode, a timeslot corresponds to as many as 32 bits of data. In 15-bit width mode, a timeslot corresponds to 30 bits of data. Each table entry has an enable bit and a field in which to specify the AxC interface number for the current timeslot, in addition to a `position` field which specifies the starting bit position of the IQ sample in the timeslot. 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. The `position` field is ignored in 15-bit width mode. You cannot specify an offset in 15-bit width mode.

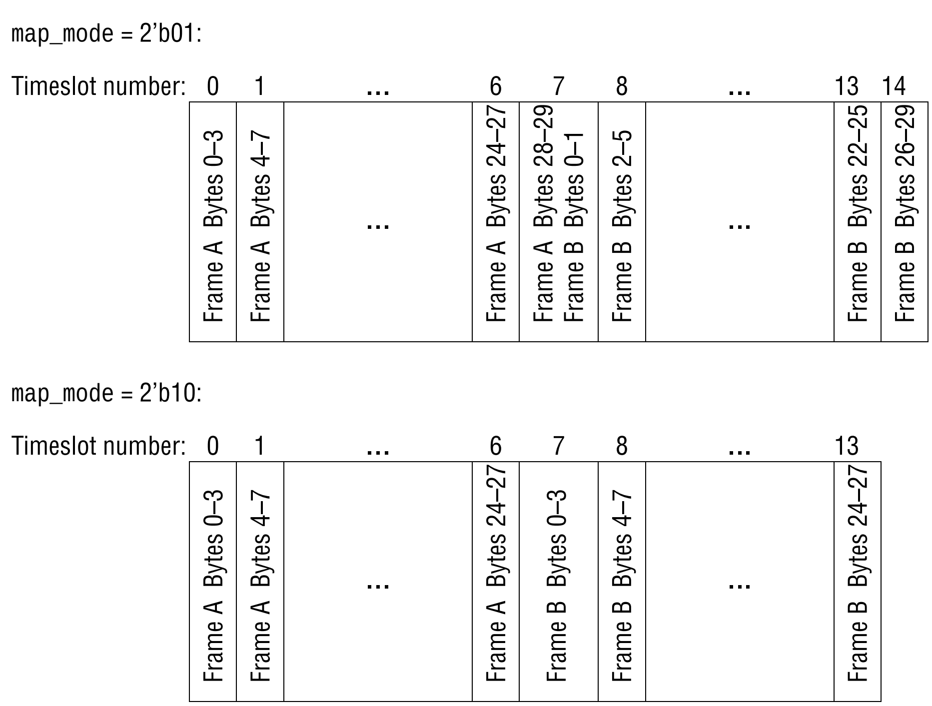
You can calculate the number of timeslots that correspond to a CPRI frame. Only the data bytes pass through the AxC interface; the control bytes in a CPRI frame do not pass through the AxC interface. Refer to the leftmost column of Table 4–4 on page 4–25 or Table 4–5 on page 4–26. The numbers in square brackets in those columns are the numbers of data bits in a CPRI frame at each CPRI line data rate. The calculation depends on the presence and values of any offsets, on whether the CPRI MegaCore function 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 offsets all set to zero. You can increment the timeslot counts as needed to accommodate the unused leading timeslot bits specified with offsets.

In 16-bit width mode, the two advanced AxC mapping modes differ in how they handle spare bytes in the CPRI frame. In 15-bit width mode, the two advanced AxC mapping modes act identically. 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.

In 16-bit width mode, when `map_mode` has value 2'b01, all of the data bits in a CPRI frame pass through the AxC interface to or from the CPRI MegaCore function. When `map_mode` has value 2'b10, the initial 32-bit sets of data in the CPRI frame pass through the AxC interface. However, when `map_mode` has value 2'b10, the spare bytes—bytes at the end of the IQ data block that do not fill another complete 32-bit word—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.

For example, for a CPRI MegaCore function 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 4–25). If K (specified in the `K` field of the `CPRI_MAP_TBL_CONFIG` register) has value two, 480 bits, or 60 bytes, of data are sent or received on the data channel. In 16-bit mode, when `map_mode` has value 2'b01, with offsets all set to zero, all of the data bits are packed in 15 timeslots. When `map_mode` has value 2'b10, 32 of these data bits are dropped, 16 from each CPRI frame, and the remaining data bits require only 14 timeslots. The first seven timeslots are identical in the two cases. However, in the eighth timeslot, when `map_mode` has value 2'b01, the final two bytes of the data from or for the first CPRI frame are followed by the first two bytes of the second CPRI frame data. The following timeslot holds the third through sixth byte of the second CPRI frame data, and so on. The fourteenth timeslot holds the 23rd through the 26th byte of the second CPRI frame data, and the fifteenth timeslot holds the 27th through the 30th bytes of the second CPRI frame data. In contrast, when `map_mode` has value 2'b10, the final two bytes of the data from or for the first CPRI frame are dropped or assumed reserved. The eighth timeslot holds the first four bytes of the second CPRI frame data, instead. Four-byte words of data from or for the second CPRI frame appear in the eighth through the fourteenth timeslots, and the final two bytes of the second CPRI frame data are dropped. Figure 4–13 illustrates this example.

Figure 4–13. Example of Difference Between Two AxC Advanced Mapping Modes



When map_mode has value 2'b01, 32-bit data samples are packed consecutively in the payload area of the AxC container block. A data sample may span two basic frames, as shown in timeslot 7 in Figure 4–13. Spare bits become reserved bits. Reserved bits are located at the end of the AxC container block.

When map_mode has value 2'b10, 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.

☞ Some table entries are not available, depending on the map_mode and sample width. For example, in Figure 4–13, when map_mode has value 2'b01, only table entries 0–14 are available, and when map_mode has value 2'b10, only table entries 0–13 are available.

CPRI MAP Receiver Interface

The CPRI MAP receiver interface transmits to the data channels data that the CPRI MegaCore function receives from the CPRI link. The CPRI MAP receiver implements an Avalon-ST interface protocol. Refer to "CPRI MAP Receiver Signals" on page 5–7 for details of the interface communication signals.

CPRI MAP receiver communication on the individual data map interfaces is FIFO-based or synchronized, as determined by the map_rx_sync_mode field of the CPRI_MAP_CONFIG register. In FIFO mode, each data channel, or AxC interface, has an output ready signal, mapN_rx_valid. Each data map interface asserts its ready signal when it is ready to transmit data on this data channel—when the buffer level is above the threshold indicated in the CPRI_MAP_RX_READY_THR register. FIFO-based communication is simple but does not allow easy control of buffer delay.

In the synchronized communication, called 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 software asserts the resynchronization signal, it begins reading data on the mapN_rx_data[31:0] data bus for the individual AxC interface.

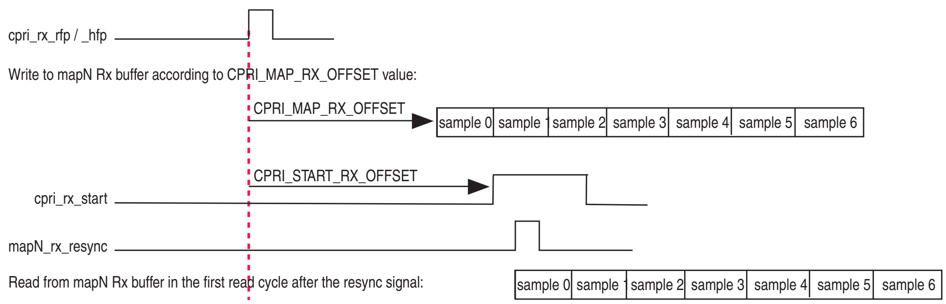
In synchronous buffer mode, the application should ignore the mapN_rx_valid output signals and hold the mapN_rx_ready input signals high. The CPRI MegaCore function does assert the mapN_rx_valid output signals in response to the mapN_rx_ready signals. The application must hold the mapN_rx_ready input signals high to allow the FIFO pointers to change values. If the application does not hold the mapN_rx_ready input signals high, the CPRI MAP Rx interface does not function correctly.

☞ To ensure IP core control over the resynchronization signal timing, Altera recommends that your application trigger the mapN_rx_resync signal with the CPRI MegaCore function 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. In addition to ensuring that application-specific constraints are accommodated, the system can set the `CPRI_START_OFFSET_RX` register to an offset that lags the desired frame position in the CPRI transmission, in anticipation of the delays from the CPRI Rx interface and through the antenna-carrier interface Rx buffer. For information about these delays, refer to "Rx Path Delay" on page 4–40.

Figure 4–15 shows the roles of the `CPRI_START_OFFSET_RX` and `CPRI_MAP_OFFSET_RX` registers in ensuring correct alignment.

Figure 4–14. User-Controlled Delays to the AxC Data Channels



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 system source of the AxC payload transmits 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 optionally asserts the `mapN_rx_resync` signal to indicate that the AxC container block can be written to the Rx MAP buffer for this data channel. Assertion of the `mapN_rx_resync` signal resets the read pointer of current antenna-carrier interface (mapN) Rx buffer to zero, so that all the data in the buffer is transmitted to the data channel. The `mapN_rx_data` can safely be sampled by the data channel one cycle after the `mapN_rx_resync` signal is asserted.

On the CPRI side of the mapN Rx buffer, the CPRI MAP receiver interface transfers data to the mapN Rx buffer. The offset programmed in the `CPRI_MAP_OFFSET_RX` register tells the CPRI MAP receiver interface when to reset the write pointer of the mapN Rx buffer and start transferring data to the buffer from the CPRI receiver interface. 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 MegaCore function informs you of buffer overflow and underflow (in the CPRI_IQ_RX_BUF_STATUS register described in Table 6–46 on page 6–18, as reported in the mapN_rx_status_data output signals described in Table 5–10 on page 5–7), 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 provide the correct timing to compensate for delays through the CPRI MegaCore function. For information about delays in the Rx path through the IP core, refer to "Rx Path Delay" on page 4–40.

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.

CPRI MAP Transmitter Interface

The CPRI MAP transmitter interface receives data from the data channels and passes it to the CPRI interface to transmit on the CPRI link. The CPRI MAP transmitter implements an Avalon-ST interface protocol. Refer to "CPRI MAP Transmitter Signals" on page 5–9 for details of the interface communication signals.

CPRI MAP transmitter communication on the individual data map interfaces is FIFO-based or synchronized, as determined by the map_tx_sync_mode field of the CPRI_MAP_CONFIG register. In FIFO mode, each data channel, or AxC interface, has an output ready signal, mapN_tx_ready. Each data map interface asserts its ready signal when it is ready to receive data on this data channel for transmission to the CPRI interface—when the buffer level is at or below the threshold indicated in the CPRI_MAP_TX_READY_THR register. FIFO-based communication is simple but does not allow easy control of buffer delay.

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. After the application software asserts the resynchronization signal, it 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.

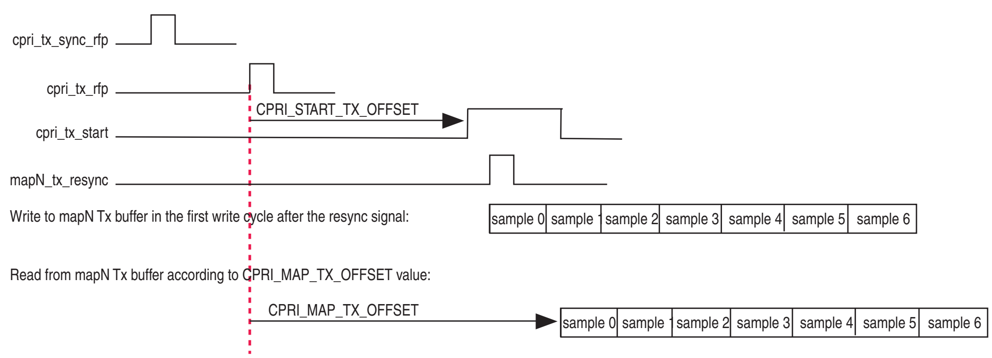
In synchronous buffer mode, the application should ignore the mapN_tx_ready output signals. However, it should assert the mapN_tx_valid input signals when sending valid data. The CPRI MegaCore function holds the mapN_tx_ready output signals high. The application must assert the mapN_tx_valid input signals immediately after asserting the mapN_tx_resync signals. If the application does not assert the mapN_tx_valid input signals high as expected, the CPRI MAP Tx interface does not function correctly.

To ensure IP core control over the resynchronization signal timing, Altera recommends that your application trigger the mapN_tx_resync signal with the CPRI MegaCore function 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 4–46.

Figure 4–15 shows the roles of the `CPRI_START_OFFSET_TX` and `CPRI_MAP_OFFSET_TX` registers in ensuring correct alignment.

Figure 4–15. User-Controlled Delays in Accepting Data From the AxC Data Channels



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 one cycle after the `mapN_tx_resync` signal is asserted.

On the CPRI side of the mapN Tx buffer, the CPRI 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 CPRI MAP transmitter interface when to reset the read pointer of the mapN Tx buffer and start transfering 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 MegaCore function informs you of buffer overflow and underflow (in the `CPRI_IQ_TX_BUF_STATUS` register described in Table 6–47 on page 6–18 and as reported in the `mapN_tx_status_data` output vector described in Table 5–11 on page 5–9), 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.

You set the values in the `CPRI_START_OFFSET_TX` and `CPRI_MAP_OFFSET_TX` registers to provide the correct timing to compensate for delays through the CPRI MegaCore function. For information about delays in the Tx path through the IP core, refer to "Tx Path Delay" on page 4–46.

PRBS Generation and Validation

The CPRI MegaCore function supports generation and validation of several predetermined pseudo-random binary sequences (PRBS) for antenna-carrier interface testing. The value in the `prbs_mode` field of the `CPRI_PRBS_CONFIG` register specifies whether the CPRI MAP interface module is in data mode or in internal loopback 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 CPRI MAP interface module is not in internal loopback testing 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 records the PRBS error detection status for each AxC interface.

Auxiliary Interfaces

The CPRI auxiliary interfaces enable multi-hop routing applications and provide timing reference information for transmitted and received frames. The AUX Receiver and AUX Transmitter interfaces are implemented 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.

AUX Receiver Module

The AUX receiver module transmits data that the CPRI MegaCore function 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 multi-hop configurations.

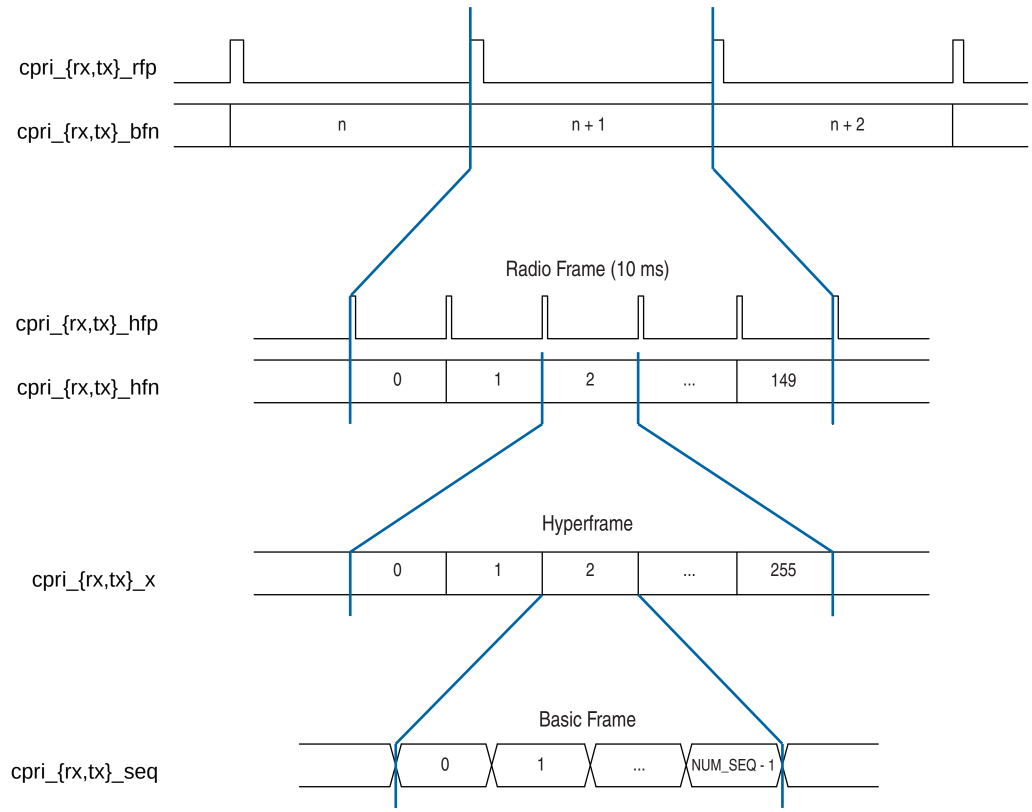
The AUX interface receiver module provides the following data and synchronization lines:

- `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 interface frame synchronization machine. Their delay following the frame on the CPRI interface reflects the quantified delay through the CPRI IP core. Refer to "Rx Path Delay" on page 4–40. These signals are all fields in the `aux_rx_status_data` bus. For additional information about the AUX receiver signals, refer to Table 5–12 on page 5–11.

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

Figure 4–17. AUX Interface Outgoing 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 Outgoing 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

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

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

Note to Figure 4–17:

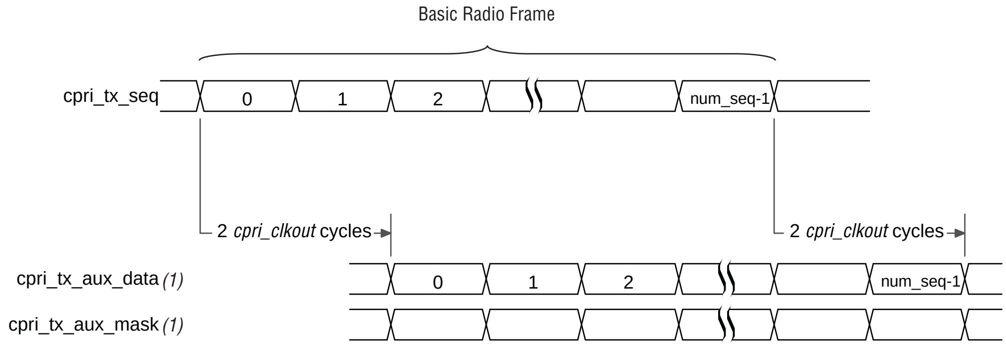
(1) Light blue table cells indicate control word bytes. White table cells indicate data word bytes.

AUX Transmitter Module

The AUX transmitter module receives data on the incoming AUX Avalon-ST interface and sends it to the CPRI MegaCore function 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 output frame synchronization information with a delay of exactly two `cpri_clkout` clock cycles. Figure 4–18 shows the expected timing on the incoming AUX connection.

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 5–13 on page 5–12.

The AUX interface transmitter module derives the frame synchronization information from the CPRI transmitter frame synchronization state machine. It provides the following data and synchronization lines 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 MegaCore function Layer 1 uses the `cpri_tx_aux_mask` to select the enabled bit values in the control transmit table. 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 MegaCore function 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.
- `cpri_tx_sync_rfp`—Synchronization input used in REC master to control the start of a new 10 ms radio frame

For information about the relationships between the synchronization pulses and numbers, refer to Figure 4–16 on page 4–35. For the mapping of data between the AUX interface and the CPRI link, refer to Figure 4–17 on page 4–35.

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 5–13 on page 5–12.

Delay Measurement

For system configuration and correct synchronization, the CPRI MegaCore function must meet the CPRI V4.1 Specification measurement and delay requirements. The CPRI MegaCore function makes the current Rx delay measurement values available in the CPRI_RX_DELAY and CPRI_EX_DELAY_STATUS delay registers, and makes the round-trip delay measurement available in the CPRI_ROUND_DELAY register. In addition, the MegaCore function allows you to specify settings that control the degree of delay accuracy in the status registers, by programming the CPRI_RX_DELAY_CTRL and CPRI_EX_DELAY_CONFIG registers.

The following sections describe the delay requirements and how you can use these registers to ensure that your application conforms to the CPRI V4.1 Specification delay requirements.

Delay Requirements

CPRI V4.1 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 MegaCore function.

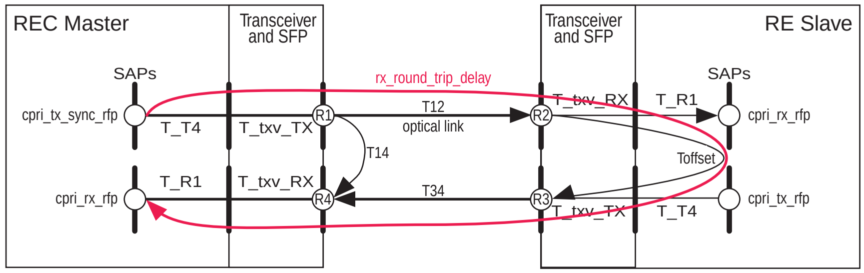
The CPRI MegaCore function complies with CPRI V4.1 Specification requirements R-19, R-20, R-20A, R-21, and R-21A.

CPRI V4.1 Specification requirement R-20A addresses the maximum allowed delay in switching between receiving and transmitting on the AxC interface. Because the CPRI MegaCore function 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 multi-hop round-trip delay accuracy. In calculating these delays, Altera assumes that the downlink and uplink cable delays have the same duration.

Figure 4–19 shows the reference points you can use to determine the CPRI MegaCore function delay measurements for single-hop CPRI configurations.

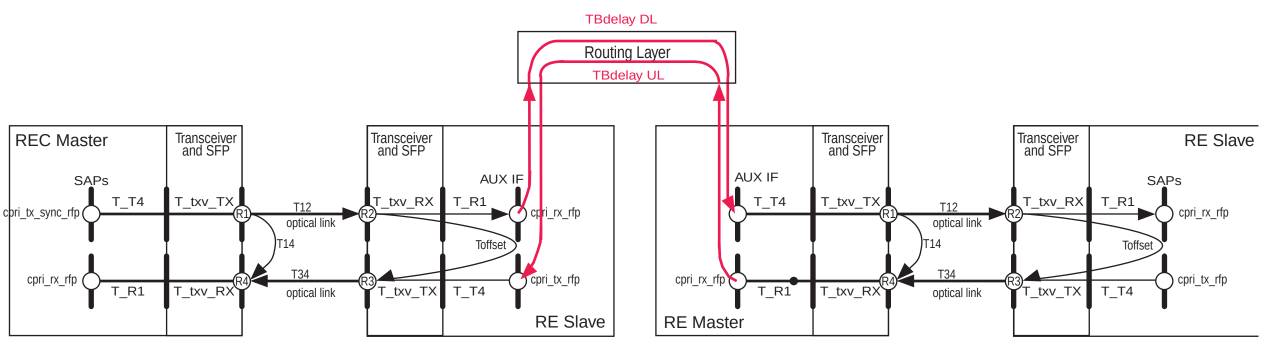
Figure 4–19. Single-Hop CPRI Configuration Delay Measurement Reference Points



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 4–20 shows the reference points you can use to determine the CPRI MegaCore function delay measurements for multi-hop CPRI configurations. The duration of TBdelay depends on your routing layer implementation.

Figure 4–20. Multi-Hop CPRI Configuration Delay Measurement Reference Points



The following sections describe the delay through the CPRI MegaCore function on the Rx path and on the Tx path to the two SAPs—the AUX interface and the MAP 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 multi-hop cases.

Rx Path Delay

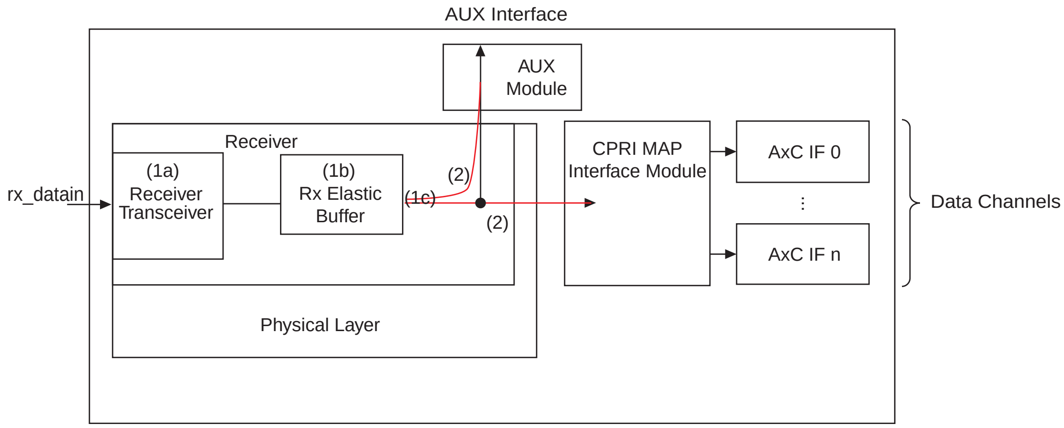
The Rx path delay is the cumulative delay from the arrival of the first bit of a 10 ms radio frame on the CPRI Rx interface to the start of transmission of the radio frame data on the AUX or AxC interface.

Rx Path Delay Components

The CPRI specification defines requirements on the path to an SAP. The CPRI MegaCore function has two relevant SAPs: the AUX interface and the MAP interface. This section provides the information to calculate the Rx path delay to output on the

AUX interface or to output on the MAP interface. Figure 4–21 shows the relation between the two Rx paths.

Figure 4–21. Rx Path Delay to AUX Output and to AxC Interfaces



The Rx path delay to the AUX interface 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 MegaCore function internal transmission of the radio frame pulse from the CPRI 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 "Transceiver Latency" on the following pages.

   b. Delay through the clock synchronization FIFO, as well as phase misalignment 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. Byte alignment delay that can occur as data is shifted out of the Rx elastic buffer. This 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.

2. Delay from the CPRI low-level receiver block to the AUX interface. This delay depends on the device family and CPRI data rate. This delay is `T_R1` in Figure 4–19 on page 4–40. Refer to "Fixed Core Delay Component" on page 4–44.

The Rx path delay to the MAP interface is the sum of the following delays:

1. Delay between the arrival of the first bit of a 10 ms radio frame on the CPRI Rx interface and the MegaCore function internal transmission of the radio frame pulse from the CPRI low-level receiver block. This delay is identical in the paths to the AUX interface and to the MAP interface. It comprises transceiver latency and the extended delay measurement described in the following sections, as well as a variable byte alignment delay.

2. Delay from the CPRI low-level receiver block to the MAP interface block. This delay is also identical to the delay in the path to the AUX interface. Refer to "Fixed Core Delay Component" on page 4–44.
3. Delay from arrival of the data at the MAP interface block, through an AxC interface and out to its data channel. This delay comprises the time during which the data waits in the mapN Rx buffer. When the AxC interfaces are in synchronous buffer mode, the timing depends on the offset values in the `CPRI_START_OFFSET_RX` and `CPRI_MAP_OFFSET_RX` registers and on the application response to the `cpri_rx_start` output signal. When the AxC interfaces are in FIFO mode, the delay depends on the programmed buffer threshold and the application. Refer to "CPRI MAP Receiver Interface" on page 4–29.

The following sections describe the individual delays and how to calculate them.

Transceiver Latency

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 4–6 shows the fixed latency through the transceiver in the receive and transmit sides of the CPRI MegaCore function. These values correspond to `T_txv_RX` and `T_txv_TX` in Figure 4–19.

Table 4–6. Fixed Latency Through Transceiver

Direction	Latency Through Transceiver in cpri_clkout Clock Cycles	
	Arria II GX or Cyclone IV GX Device (1)	Arria II GZ or Stratix IV GX Device
In CPRI Receiver (`T_txv_RX`)	4.65	6.5
In CPRI Transmitter (`T_txv_TX`)	3.35	3.5

Note to Table 4–6:

(1) The deterministic latency mode of the Cyclone IV GX device transceivers is still pending characterization.

The clean-up PLL shown in Figure 4–2 on page 4–7, Figure 4–4 on page 4–9, and Figure 4–6 on page 4–11 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 4–6, 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 MegaCore function 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 4–7 shows some example M/N ratios and the resolutions they provide, for a CPRI MegaCore function that runs at data rate 3072 Mbps and targets a Stratix IV GX device.

Table 4–7. 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

Note to Table 4–7:

(1) Table 4–1 on page 4–12 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.

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 MegaCore function runs at data rate 3072 Mbps. In this case, Table 4–1 on page 4–12 shows that the cpri_clkout frequency is 76.80 MHz, so a cpri_clkout cycle is 1/(76.80 MHz).

Refer to Table 4–7 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, follow these steps:

1. Program the value N=127 in the rx_ex_delay field of the CPRI_EX_DELAY_CONFIG register at offset 0x3C (Table 6–19 on page 6–9).

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 6–20 on page 6–9).

   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.

Fixed Core Delay Component

In the Rx path, the delay from the CPRI low-level receiver block to the AUX interface or the MAP interface is fixed. In the Tx path, the delay from the AUX or MAP interface block to the CPRI low-level transmitter block is fixed. These delays depend on the device family and CPRI data rate. Table 4–8 shows the fixed delays between the low-level transmitter or receiver block and the MAP or AUX block.

Table 4–8. Fixed Latency Between Low-Level Transmitter or Receiver and the MAP or AUX Block in cpri_clkout Cycles

Direction	Arria II GX or Cyclone IV GX Device		Arria II GZ or Stratix IV GX Device	
	Data Rate 614.4 Mbps	Data Rate > 614.4 Mbps	Data Rate 614.4 Mbps	Data Rate > 614.4 Mbps
Rx path fixed delay (T_R1)	5	5	5	3
Tx path fixed delay (T_T4)	4.5	5	4.5	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 4–43. This example walks through the calculation for the case of a CPRI MegaCore function that runs at CPRI data rate 3072 Mbps and targets an Arria II GX device.

To calculate the Rx path delay, follow these steps:

1. Consult Table 4–6 on page 4–42 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 4–43 for your CPRI MegaCore function 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 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. When the value is zero, no byte alignment delay occurs.
4. Consult Table 4–8 on page 4–44 to determine the delay through the CPRI MegaCore function to the AUX interface. For the example, the duration of this delay is five `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 worst case Rx path delay as follows:

   Rx path delay = T_txv_RX + <*delay through Rx Receive buffer*>
   + <*worst case variable byte alignment delay*>
   + <*delay to AUX IF*>
   = 4.65 + 33.236 + 1 + 5 `cpri_clkout` clock cycles
   = 43.886 `cpri_clkout` clock cycles

   The best case Rx path delay has zero byte alignment delay, for a total delay of 42.886 `cpri_clkout` clock cycles.

Rx Path Delay to AxC Output

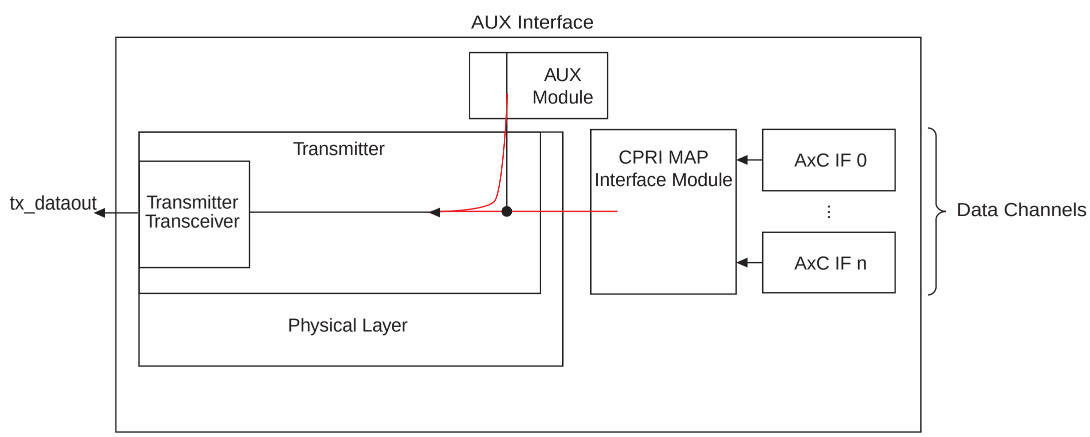
The delay through the MAP interface depends on whether the AxC data communication is programmed in FIFO mode or in synchronous buffer mode. In FIFO mode, the delay through the mapN Rx buffer depends on your programmed threshold value and the application. The data is not sent to the data channel untiI the buffer threshold is reached, so the wait time in the buffer depends on data received on the CPRI link before and after the current data. In synchronous buffer mode, because programmed offsets control the mapN Rx buffer pointers, the delay can be quantified.

The Rx path delay to output on the AxC interface has the same components 1 and 2 as the Rx path delay to the AUX interface. These components comprise the delay from input to the receiver transceiver on the CPRI link, to input to the MAP interface block. The delay from there to the individual AxC interfaces is the time the data spends in the mapN Rx buffer, before being written to the AxC interface data channel. In synchronous buffer mode, this delay is one cycle if the sample rate is a multiple of 3.84 MHz, and two cycles otherwise. Refer to "CPRI MAP Receiver Interface" on page 4–29.

Tx Path Delay

The Tx path delay is the cumulative delay from the arrival of the first bit of an IQ data sample on the CPRI AUX interface or on a CPRI AxC interface, to the start of transmission of this data on the CPRI link. Figure 4–22 shows the relation between the two Tx paths.

Figure 4–22. Tx Path Delay to AUX Output and to AxC Interfaces



In the CPRI MegaCore function the delay from the AUX interface is fixed. This path has no variable delay component, because it does not cross clock domains.

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. This delay is `T_T4` in Table 4–8 on page 4–44.
2. Link delay through the transceiver. This delay is `T_txv_TX` in Table 4–6 on page 4–42.

The Tx path delay from an AxC interface comprises the following delays:

1. Delay through the mapN Tx buffer.
2. Fixed delay from the mapN Tx buffer through the CPRI low-level transmitter to the transceiver.
3. Link delay through the transceiver.

The delay through the mapN Tx buffer depends on whether the AxC data communication is programmed in FIFO mode or in synchronous buffer mode. In FIFO mode, the delay through the mapN Tx buffer depends on your threshold value and the application. Depending on the size of the data burst, the Tx buffer may not reach the threshold until the next data burst arrives on the data channel. In synchronous buffer mode, because programmed offsets control the mapN Tx buffer pointers, the delay can be quantified.

The Tx path delay to output from the MAP interface block has the same components 1 and 2 as the Tx path delay from the AUX interface. These components comprise the delay from the MAP interface block to output on the CPRI transmitter interface. In synchronous buffer mode, the delay from the individual AxC interfaces to output from the MAP interface block is the time the data spends in the mapN Tx buffer. This delay is one cycle if the sample rate is a multiple of 3.84 MHz, and two cycles otherwise. Refer to "CPRI MAP Transmitter Interface" on page 4–31.

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 4–19 on page 4–40. The CPRI V4.1 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 or 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 V4.1 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.

Because the CPRI REC master and the CPRI RE slave might be on different devices, the following formulas specify the source MegaCore function (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 4–19 on page 4–40, 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
- `T34`—The delay from CPRI RE to CPRI REC

However, the CPRI MegaCore function 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 6–18 on page 6–9)
- *<REC Rx path delay>* is the Rx path delay, described in "Rx Path Delay" on page 4–40, for the values in the CPRI REC master
- *<Tx path delay>* is the Tx path delay, described in "Tx Path Delay" on page 4–46, 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>*, 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

Round-Trip Calculations for a Multi-Hop Configuration

In a multi-hop system, we must combine the delays between and through the different CPRI RE masters and CPRI RE slaves to determine the round-trip delay.

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 4–19 on page 4–40.

To determine the round-trip delay of a full multi-hop 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. Use the following calculation, based on the labels in Figure 4–20 on page 4–40:

Round-trip delay = ∑ `rx_round_trip_delay` (hop i) + ∑ (TBdelayUL + 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).

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.

Data Link Layer for Fast Control and Management Channel (Ethernet)

In the CPRI MegaCore function, the Ethernet Media Access Control (MAC), or fast data link layer, passes Ethernet data from the CPU interface to the CPRI transmitter interface block, and from the CPRI receiver block to the CPU interface. 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 0x24 and 0x3F, inclusive. The Altera CPRI MegaCore function reserves subchannels 0x10 to 0x23; the start of Ethernet data can be located in any following word in the hyperframe. 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 MegaCore function Avalon-MM CPU interface. For additional information about this register, refer to Chapter 6, Software Interface.

Ethernet Transmitter

The Ethernet transmitter module receives data on the Ethernet channel and writes it to an Ethernet Tx buffer, from which the CPRI transmitter module transmits it on the CPRI link.

Ethernet Data Transfer

The Ethernet transmitter module sends Ethernet data to an Ethernet Tx buffer through the `ETH_TX_DATA` or `ETH_TX_DATA_WAIT` register. Status bits in the `ETH_TX_STATUS` register indicate when the Ethernet Tx buffer is ready to receive one word, and when it is ready to receive a 32-bit packet. Before the Ethernet packet end-of-packet word is written to the `ETH_TX_DATA` or `ETH_TX_DATA_WAIT` register, the Ethernet transmitter module sets the `tx_eop` bit and configures the `tx_length` field in the `ETH_TX_CONTROL` register to indicate how many bytes in this word are padding. 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. The `ETH_TX_DATA_WAIT` register can accept data when the Ethernet Tx buffer is not ready for new data.

The Ethernet transmitter module must write frame 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 value 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.

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 set the tx_abort bit directly.

Interrupts

Software can enable interrupts by setting bits in the ETH_CONFIG_1 register. 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 can be written to the Ethernet Tx buffer.

Ethernet Receiver

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 6, Software Interface.

MAC Address Filtering

To disable MAC address checking, set the mac_check bit of the ETH_CONFIG_1 register. If the mac_check bit is set, the Ethernet receiver accepts all received packets.

All CPRI MegaCore function MAC address filters assume the MAC destination address is in the first six bytes of the fast C&M data in the CPRI hyperframe. If the MAC destination address is located elsewhere in the fast C&M data, you must set the mac_check bit.

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 enabled.
- 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 MegaCore function 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 has been 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.

Data Link Layer for Slow Control and Management Channel (HDLC)

In the CPRI MegaCore function, the High-Level Data Link Control (HDLC), 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 4–9 shows the possible rate configurations.

Table 4–9. HDLC Channel Bit Rates (Part 1 of 2)

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

Table 4–9. HDLC Channel Bit Rates (Part 2 of 2)

Value in Z.66.0.0[2:0]	HDLC Bit Rate (Kbps)	Minimum CPRI Line Rate (Mbps)
110	3840	4915.2
	4800	6144.0
111	*(1)*	

Note to Table 4–9:

(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

MII Interface to an External Ethernet Block

You can define a CPRI MegaCore function to bypass the internal Ethernet or HDLC module and communicate directly with an external Ethernet block through an MII-like interface, referred to in this document as the MII interface. This interface is not a true MII 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 an MII interface. If you use this interface, your external Ethernet block must communicate with the CPRI MegaCore function synchronously with the `cpri_mii_txclk` and `cpri_mii_rxclk` clocks.

The MII interface supports the bandwidth described in the CPRI V4.1 Specification in Table 12, Achievable Ethernet bit rates.

MII Interface Transmitter

The MII interface 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 MegaCore function achieves frame synchronization, the MII interface transmitter module can accept incoming data on the MII interface. The MII interface 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 interface 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 interface 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 interface module inserts an Ethernet K symbol (5'b10001) in this buffer. These two symbols indicate Ethernet start-of-packet. While the CPRI MII Interface transmitter is inserting the J and K symbols, it ignores incoming data on cpri_mii_txd. Refer to Figure 4–23 and to Figure 4–24.

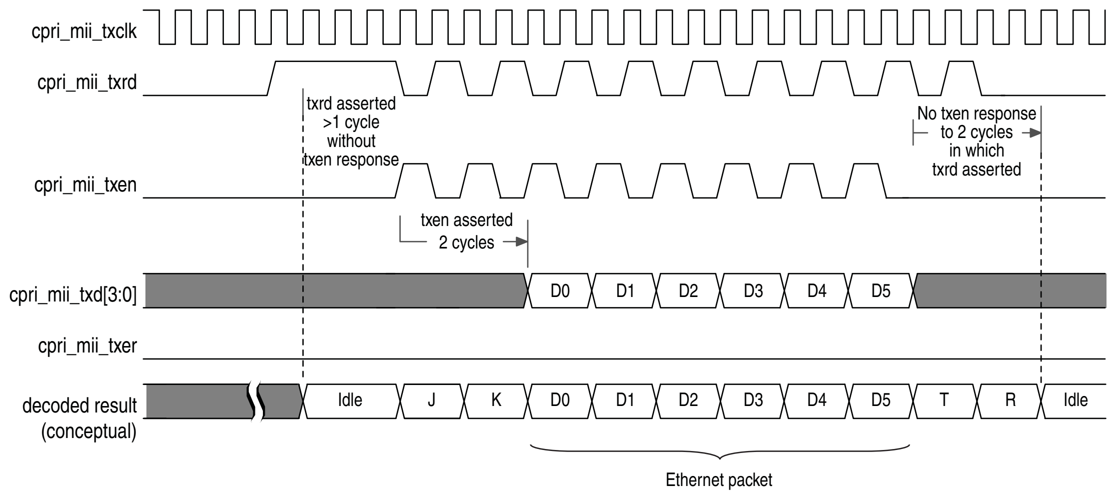
Typically, the external Ethernet block asserts cpri_mii_txen one clock cycle after cpri_mii_txrd is asserted. If not, after the initial cycle in which cpri_mii_txrd is asserted, while cpri_mii_txrd continues to be asserted but cpri_mii_txen is not yet asserted, the CPRI MII Interface transmitter inserts an Idle cycle in the buffer of data to be transmitted to the CPRI link. After cpri_mii_txen is asserted following the assertion of cpri_mii_txrd, if cpri_mii_txen is subsequently deasserted following a cycle in which cpri_mii_txrd remains asserted, the CPRI MII Interface transmitter assumes the external Ethernet block has reached end-of-frame, and begins insertion of the Ethernet end-of-packet symbol (T followed by R). While the CPRI MII Interface transmitter is inserting the T and R symbols, it ignores incoming data on cpri_mii_txd. Refer to Figure 4–23 and to Figure 4–24.

While the cpri_mii_txen signal remains asserted or is continually reasserted in response to assertion of cpri_mii_txrd, the MII interface 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 in response to the MII interface module asserting cpri_mii_txrd, the MII interface module inserts an Ethernet end-of-packet symbol in the data passed to the CPRI interface module, and then stops asserting the cpri_mii_txrd signal.

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 interface transmitter module observes that both cpri_mii_txen and cpri_mii_txer are asserted in the same cpri_mii_txclk cycle, the MII interface module inserts an Ethernet HALT symbol (5'b00100) in the data passed to the CPRI interface module. Figure 4–26 on page 4–56 provides an example in which the cpri_mii_txer signal is asserted, and shows how the error indication propagates to the MII interface receiver module on the CPRI link slave.

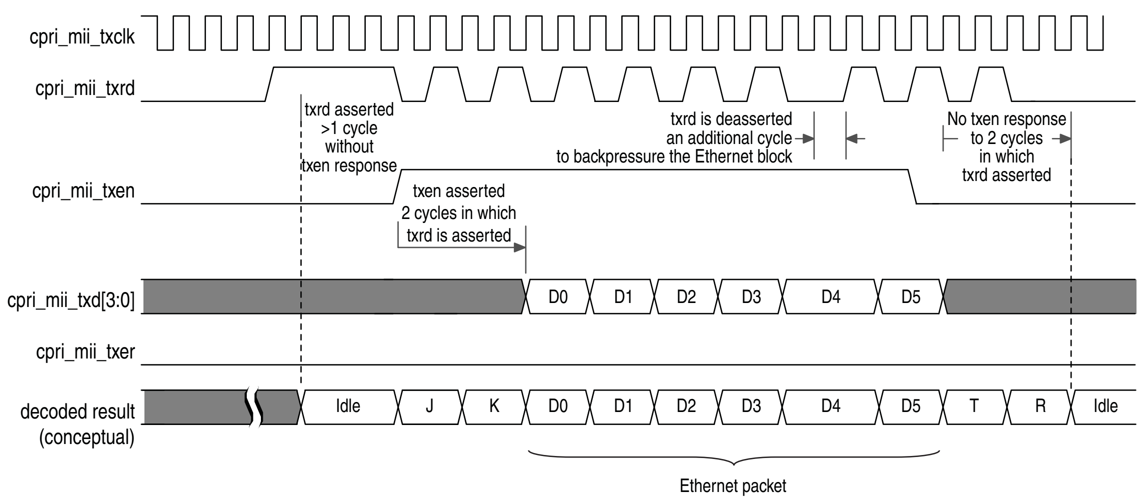
Figure 4–23 and Figure 4–24 both illustrate the MII interface transmitter protocol with no input errors.

Figure 4–23. CPRI MII Interface Transmitter Example 1



Although Figure 4–23 shows `cpri_mii_txen` continually deasserted following each cycle in which `cpri_mii_txrd` is not asserted, your external Ethernet block may optionally keep the `cpri_mii_txen` signal asserted continuously for the duration of the packet. Figure 4–24 shows a case in which `cpri_mii_txen` remains asserted for the duration of the packet transfer. In addition, although Figure 4–23 shows `cpri_mii_txrd` reasserted every other cycle during transmission of an Ethernet packet on `cpri_mii_txd`, this need not always occur. The CPRI MII Interface 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 4–24. CPRI MII Interface Transmitter Example 2



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 Interface 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 interface transmitter module, refer to “CPRI MII Interface Transmitter Signals” on page 5–6.

MII Interface Receiver

The MII interface 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 interface.

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

The MII interface 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 interface receiver module asserts the cpri_mii_rxwr signal. After the MII interface receiver module completes sending data to the external Ethernet block, it deasserts the cpri_mii_rxdv signal.

The cpri_mii_rxer signal indicates whether or not frame synchronization is achieved. While asserted, it indicates to the external Ethernet block that the data currently on cpri_mii_rxd is not valid.

Figure 4–25 illustrates the MII interface receiver protocol.

Figure 4–25. CPRI MII Interface Receiver Example

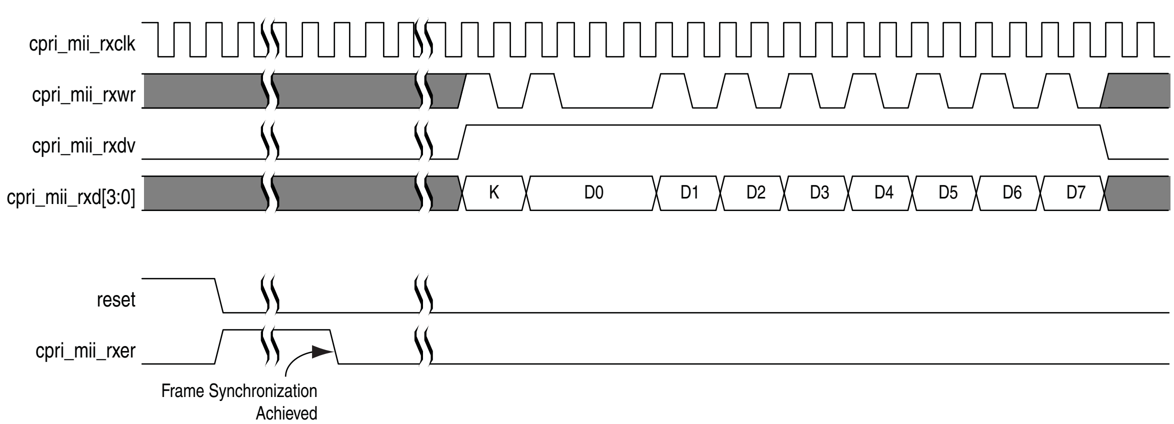
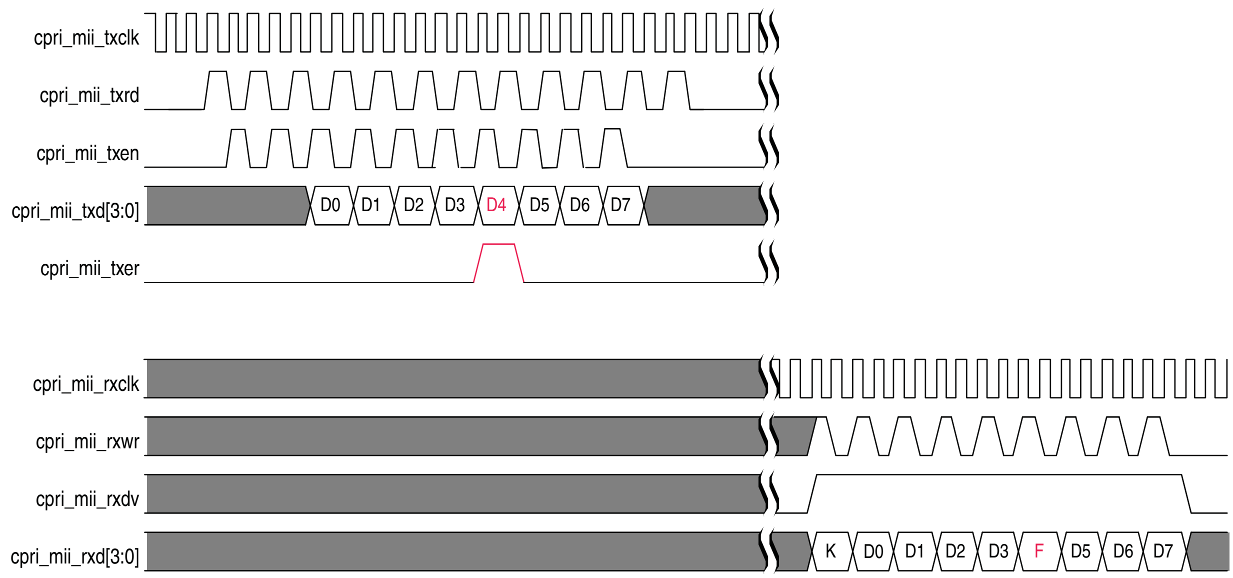


Figure 4–26 shows an example timing diagram in which an input error is noted on the MII interface of a transmitting RE or REC master, and the data from the MII interface is transmitted on the CPRI link to a receiving RE slave. The timing diagram shows the MII interface signals on the transmitting master and the receiving slave. The data value captured on the MII interface 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). This symbol is decoded by the RE slave MII interface receiver module as an F (b'41111).

Figure 4–26. CPRI MII Interface Signals on Transmitting RE or REC Master and on Receiving RE Slave



For more information about the MII interface receiver module, refer to "CPRI MII Interface Receiver Signals" on page 5–6.

This chapter describes all the top-level signals of the CPRI MegaCore function.

Physical Layer Signals

Table 5–1 through Table 5–6 list the input and output signals of the Physical layer of the CPRI MegaCore function. Refer to Figure 4–9 on page 4–17 for details of the I/O signals.

CPRI Data Signals

Table 5–1 lists the CPRI data link signals.

Table 5–1. CPRI 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 5–2 lists the Layer 1 clock and reset signals.

Table 5–2. 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 MegaCore function 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 MegaCore function 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 4–8 on page 4–14 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 5–3 lists the Layer 1 error signal for the CPRI MegaCore function.

Table 5–3. Layer 1 Error Signal

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

Auto-Rate Negotiation Signals

Table 5–4 lists the auto-rate negotiation signals for the CPRI MegaCore function. These output signals enable the auto-rate negotiation hardware and software outside the CPRI MegaCore function to quickly monitor auto-rate negotiation status, and are implemented in all device families.

In Cyclone IV GX devices, channel reconfiguration is enabled to support auto-rate negotiation. Table 5–5 lists the signals implemented in CPRI MegaCore functions targeted to Cyclone IV GX devices to support scan-chain based reconfiguration.

Table 5–4. Auto-Rate Negotiation Signals

Signal	Direction	Description
datarate_en	Output	Indicates whether auto-rate negotiation is enabled. This signal reflects the value in the i_datarate_en field of the AUTO_RATE_CONFIG register described in Table 6–21 on page 6–10.
datarate_set	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 6–21 on page 6–10. The CPRI line rate is encoded in this field using the following values: 0001: 6.144 Mbps 0010: 1228.8 Mbps 0100: 2457.6 Mbps 0101: 3072.0 Mbps 1000: 4915.0 Mbps (not supported for Cyclone IV GX devices) 1010: 6144.0 Mbps (not supported for Cyclone IV GX devices)

Table 5–5. Scan-Chain Based Reconfiguration Interface Signals For CPRI Auto-Rate 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.
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 5–6 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 5–6. Transceiver Signals (Part 1 of 2)

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*.
gxb_pll_inclk	Input	Input clock to the transceiver PLL. If the CPRI MegaCore function 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] *(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] *(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] *(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)	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.
reconfig_fromgxb_s_rx [16:0] ([4:0] for Cyclone IV GX 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)	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.

Table 5–6. Transceiver Signals (Part 2 of 2)

Signal	Direction	Description
reconfig_write	Input	Indicates the user is writing to the dynamic reconfiguration controller to implement the auto-rate 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 auto-rate configuration feature, you must tie this input to 0.
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 auto-rate configuration feature, you must tie this input to 0.
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.
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 MegaCore functions 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.
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 5–6:

(1) Refer to "Instantiate Multiple CPRI MegaCore Functions" on page 2–4 for information about how to successfully combine multiple high-speed transceiver channels—whether in two CPRI MegaCore function instances or in a CPRI MegaCore function 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.

CPU Interface Signals

Table 5–7 lists the CPU interface module 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 5–7. CPU Interface Signals

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 4–8 on page 4–14 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.
`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.

CPRI MII Interface Signals

Table 5–8 and Table 5–9 list the signals used by the CPRI MII interface module of the CPRI MegaCore function. The CPRI MII interface is enabled if you turn off **Include MAC block** in the CPRI parameter editor. The CPRI MII interface signals are available only if you enable the CPRI MII interface. For information about the MII handshaking protocol implementation, refer to "MII Interface" on page 4–4.

CPRI MII Interface Receiver Signals

Table 5–8 lists the CPRI MII interface receiver signals.

Table 5–8. 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 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 MegaCore function achieves frame synchronization.
cpri_mii_rxd[3:0]	Output	Ethernet receive nibble data. Data bus for data from the CPRI MegaCore function to the external Ethernet block. All bits are deasserted during reset, and all bits are asserted after reset until the CPRI MegaCore function achieves frame synchronization.

CPRI MII Interface Transmitter Signals

Table 5–9 lists the CPRI MII interface transmitter signals. These signals are available if you exclude the MAC block from the CPRI MegaCore function.

Table 5–9. 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 interface 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 MegaCore function inserts an Ethernet HALT symbol in the data it passes to the CPRI link.

Table 5–9. 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 MegaCore function, 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 interface 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.

CPRI MAP Interface Signals

Table 5–10 and Table 5–11 list the signals used by the CPRI MAP interface modules of the CPRI MegaCore function. The CPRI MAP interfaces are implemented as Avalon-ST interfaces.

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

CPRI MAP Receiver Signals

Table 5–10 lists the CPRI MAP receiver interface signals.

Table 5–10. CPRI MAP Receiver Interface Signals (Part 1 of 2)

Signal	Direction	Description
map{23…0}_rx_clk	Input	Clock signal for each antenna-carrier interface.
map{23…0}_rx_reset	Input	Reset signal for each antenna-carrier interface. This reset is associated with the mapN_rx_clk clock. mapN_rx_reset can be asserted asynchronously, but must stay asserted at least one mapN_rx_clk cycle and must be deasserted synchronously with mapN_rx_clk. Refer to Figure 4–8 on page 4–14 for a circuit that shows how to enforce synchronous deassertion of a reset signal.
map{23…0}_rx_ready	Input	Read-ready signal for each antenna-carrier interface. Indicates to the MegaCore function that the data channel is ready to receive data on 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 CPRI MAP receiver interface is designed for READY_LATENCY equal to 0. In synchronous buffer mode, this signal must be held high continuously.
map{23…0}_rx_data[31:0]	Output	32-bit read data being transmitted on each antenna-carrier interface. Data is valid one mapN_rx_clk clock cycle after the read-ready 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 5–10. CPRI MAP Receiver Interface Signals (Part 2 of 2)

Signal	Direction	Description
map{23…0}_rx_valid	Output	Valid signal for each antenna-carrier interface in FIFO mode. This signal is asserted when the MAP_N 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 CPRI MAP receiver interface. On every rising edge of the clock at which mapN_rx_valid is high, mapN_rx_data can be sampled.
map{23…0}_rx_resync	Input	Resynchronization signal for use in synchronous buffer mode. When this signal is asserted, the read pointer of the MAP_N Rx buffer is reset to zero. When the map_rx_sync_mode bit in the CPRI_MAP_CONFIG register is set to 1, the MAP receiver interface is in synchronous buffer mode. This signal is synchronous to the mapN_rx_clk clock.
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 6–46 on page 6–18). [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 6–46 on page 6–18). [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.

CPRI MAP Transmitter Signals

Table 5–11 lists the CPRI MAP transmitter interface signals.

Table 5–11. CPRI MAP Transmitter Interface Signals (Part 1 of 2)

Signal	Direction	Description
`map{23…0}_tx_clk`	Input	Clock signal for each antenna-carrier interface.
`map{23…0}_tx_reset`	Input	Reset signal for each antenna-carrier interface. This reset is associated with the `mapN_tx_clk` clock. `mapN_tx_reset` can be asserted asynchronously, but must stay asserted at least one `mapN_tx_clk` cycle and must be deasserted synchronously with `mapN_tx_clk`. Refer to Figure 4–8 on page 4–14 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 CPRI MAP transmitter interface. On every rising edge of the clock at which `mapN_tx_valid` is high, data is sampled by the MegaCore function. In synchronous buffer mode, the application must assert this signal immediately after it asserts the resynchronization signal.
`map{23…0}_tx_data[31:0]`	Input	32-bit write data from each antenna-carrier interface. Data is valid one `mapN_tx_clk` clock cycle 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.
`map{23…0}_tx_ready`	Output	Ready signal for each antenna-carrier interface. In FIFO mode, the ready signal is asserted when the MAP_N 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 MegaCore function is ready to receive data on the data channel in the current clock cycle. Asserted by the Avalon-ST sink to mark ready cycle*s*, 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 CPRI MAP transmitter interface, `READY_LATENCY` is equal to 0, so the cycle on which `mapN_tx_ready` is asserted is the ready cycle.
`map{23…0}_tx_resync`	Input	Resynchronization signal for use in synchronous buffer mode. When the `map_tx_sync_mode` bit in the `CPRI_MAP_CONFIG` register is set to 1, the MAP transmitter interface is in synchronous buffer mode. This signal is synchronous to the `mapN_tx_clk` clock.

Table 5–11. CPRI MAP Transmitter Interface Signals (Part 2 of 2)

Signal	Direction	Description
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 6–47 on page 6–18). [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 6–47 on page 6–18). [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 5–12 through Table 5–13 list the signals on the CPRI MegaCore function auxiliary interfaces. All the signals in Table 5–12 through Table 5–13 are clocked by the internal clock visible on the cpri_clkout port.

AUX Receiver Signals

Table 5–12 lists the signals on the AUX receiver interface.

Table 5–12. 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 MAP 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 6–37 on page 6–16. 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: ■ 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 ■ 4195.2 Mbps line rate: 0–31 ■ 6144.0 Mbps line rate: 0–39
		[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 5–13 lists the signals on the AUX transmitter interface.

Table 5–13. 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: ■ 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 ■ 4195.2 Mbps line rate: 0–31 ■ 6144.0 Mbps line rate: 0–39
		[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 6–38 on page 6–16. 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 5–13. 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. Drive this signal as a pulse asserted every 10 ms synchronous to the cpri_clkout clock. The CPRI MegaCore function uses the rising edge of the pulse for synchronization and ignores the duration and falling edge of the pulse. The number of cpri_clkout clock cycles between pulses (from rising edge to rising edge) must be exactly 256 × 4 × 150 × CPRI line rate/644 Mbps.
		[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 5–14 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 5–14. 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 6–4 on page 6–2).
		[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. Tracks the number of K28.5 symbols received so far toward the five required by the receiver frame synchronization machine to reach the XSYNC state.
		[1:0]	cpri_rx_state: Indicates the 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, 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.

Clock and Reset Interface Signals

Table 5–15 describes the CPRI MegaCore function clock and reset signals not described in other sections with their associated modules.

Table 5–15. CPRI MegaCore Function Clock and Reset Signals

Signal	Direction	Description
clk_ex_delay	Input	Extended delay measurement 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 4–8 on page 4–14 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 4–8 on page 4–14 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 four 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 MegaCore function sends the reset request on the CPRI link. The hw_reset_assert signal is detected on the rising edge of cpri_clkout.

6. Software Interface

The CPRI MegaCore function supports the following sets of registers that control the CPRI MegaCore function or query its status:

- CPRI 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 6–1 lists the access codes used to describe the type of register bits.

Table 6–1. 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 6–2 lists the CPRI MegaCore function register address ranges.

Table 6–2. CPRI MegaCore Function Register Address Ranges

Address Range	Interface
0x00–0x4C	CPRI Interface Registers
0x50–0xF0	MAP Interface and AUX Interface Configuration Registers
0xF4–0x1FC	Reserved
0x200–0x24C	Ethernet Registers
0x250–0x2FC	Reserved
0x300–0x334	HDLC Registers

CPRI Interface Registers

This section lists the CPRI interface registers. Table 6–3 provides a memory map for the CPRI interface registers. Table 6–4 through Table 6–21 describe the CPRI interface registers in the CPRI MegaCore function.

Table 6–3. CPRI 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	Auto-Rate Negotiation
0x4C	CPRI_INTR_PEND	Pending Interrupt Status

Table 6–4. CPRI_INTR—Interrupt Control and Status—Offset: 0x0

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
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 interface module interrupt enable. The Ethernet and HDLC modules have separate interrupt enable control bits.	1'h0

Table 6–5. CPRI_STATUS—CPRI Status—Offset: 0x4

Field	Bits	Access	Function	Default
RSRV	[31:12]	URO	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]	URO	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]	URO	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.	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 6–5:

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

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

Table 6–6. 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	Enable CPRI control word insertion. This enable overrides the tx_control_insert bit in the CPRI_TX_CTRL register.	1'h0

Table 6–7. 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 6–8. 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 6–9. CPRI_TX_CTRL—CPRI Transmit Control Word—Offset: 0x14

Field	Bits	Access	Function	Default
RSRV	[31:9]	URO	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 6–10. CPRI_LCV—CPRI Line Code Violation Counter—Offset: 0x18

Field	Bits	Access	Function	Default
RSRV	[31:8]	URO	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 6–23 on page 6–11) is reached, because it includes LCVs that occur during initialization—before T_LCV (Table 6–24 on page 6–11) is reached—and because it saturates.	8’h0

Table 6–11. CPRI_BFN—CPRI Recovered Radio Frame Counter—Offset: 0x1C

Field	Bits	Access	Function	Default
RSRV	[31:12]	URO	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 6–12. CPRI_HW_RESET—Hardware Reset From Control Word—Offset: 0x20 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:8]	URO	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 6–12. 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 6–12:

(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 "MegaCore Function Reset Process" on page 4–13.

Table 6–13. 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 6–13. 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 6–13:

(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 6–14. 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'h24`

Table 6–15. 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 6–15. 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 For information about compatible slow C&M rates and CPRI line rates, refer to Table 4–9 on page 4–51.	3'h0
RSRV	[7]	URO	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 6–16. CPRI_RX_DELAY_CTRL—Receiver Delay Control—Offset: 0x30

Field	Bits	Access	Function	Default
RSRV	[31:17]	URO	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)*	URO	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 6–16:

(1) WIDTH_RX_BUF is the $\log_2$ of the depth of the Rx elastic buffer. In master configurations, it is set to six, specifying a 64-entry buffer. In slave configurations, it is set to four, specifying a 16-entry buffer.

Table 6–17. 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.	2'h0

Note to Table 6–17:

(1) WIDTH_RX_BUF is the log_2 of the depth of the Rx elastic buffer. In master configurations, it is set to six, specifying a 64-entry buffer. In slave configurations, it is set to four, specifying a 16-entry buffer.

Table 6–18. 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 6–19. CPRI_EX_DELAY_CONFIG—Extended Delay Measurement Configuration—Offset: 0x3C

Field	Bits	Access	Function	Default
RSRV	[31:9]	UR0	Reserved.	23'h0
rx_ex_delay	[8:0]	RW	Integration period for 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 4–43.	9'h0

Table 6–20. CPRI_EX_DELAY_STATUS—Extended Delay Measurement Status—Offset: 0x40

Field	Bits	Access	Function	Default
RSRV	[31:17]	UR0	Reserved.	15'h0
rx_ex_buf_delay_valid	[16]	RC	Indicates that the rx_ex_buf_delay field has been updated.	1'h0
RSRV	[15:(WIDTH_RX_BUF+9)] *(1)*	UR0	Reserved.	0
rx_ex_buf_delay	[(WIDTH_RX_BUF+8):0] *(1)*	RO	Extended delay measurement result. Unit is cpri_clkout clock periods. Refer to "Extended Rx Delay Measurement" on page 4–42.	0

Note to Table 6–20:

(1) WIDTH_RX_BUF is the log_2 of the depth of the Rx elastic buffer. In master configurations, it is set to six, specifying a 64-entry buffer. In slave configurations, it is set to four, specifying a 16-entry buffer.

Table 6–21. AUTO_RATE_CONFIG—Auto-Rate Negotiation Register—Offset: 0x48

Field	Bits	Access	Function	Default
RSRV	[31:5]	UR0	Reserved.	28'h0
i_datarate_en	[4]	RO	Indicates that auto-rate negotiation is enabled. (Value is 1'b0 if auto-rate negotiation is not enabled; 1'b1 if auto-rate negotiation is enabled, in the CPRI parameter editor). Refer to Figure B–1 and Figure B–2 for an illustration of the auto-rate negotiation logic in the CPRI MegaCore function.	As specified in CPRI parameter editor
i_datarate_set	[3:0]	RW	CPRI line rate to be used in next attempt to achieve frame synchronization. You set the line rate in your implementation of the auto-rate negotiation hardware and software outside the CPRI MegaCore function. Refer to Appendix B, Implementing CPRI Link Auto-Rate Negotiation, for information about how to use the auto-rate negotiation logic implemented in the CPRI MegaCore function. Encode the CPRI line rate in this field using the following values: 0001: 6.144 Mbps 0010: 1228.8 Mbps 0100: 2457.6 Mbps 0101: 3072.0 Mbps 1000: 4915.0 Mbps *(1)* 1010: 6144.0 Mbps *(1)*	4'h0

Note to Table 6–21:

(1) This value is not valid for CPRI MegaCore 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.

Table 6–22. CPRI_INTR_PEND—Interrupt Pending Status—Offset: 0x4C (Part 1 of 2)

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

Table 6–22. CPRI_INTR_PEND—Interrupt Pending Status—Offset: 0x4C (Part 2 of 2)

Field	Bits	Access	Function	Default
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 6–23. 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 6–24. 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 6–25. 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’h0

Table 6–26. 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 6–27. 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

MAP Interface and AUX Interface Configuration Registers

This section lists the MAP interface configuration registers. Table 6–28 provides a memory map for the MAP interface configuration registers. Table 6–29 through Table 6–45 describe the MAP interface configuration registers in the CPRI MegaCore function.

Table 6–28. CPRI 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 6–29. CPRI_MAP_CONFIG—CPRI Mapping Features Configuration—Offset: 0x100 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:5]	URO	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	1’h0
map_tx_sync_mode	[3]	RW	Tx MAP synchronization mode. Values are: 0: FIFO MAP interface 1: Non-FIFO (synchronized) MAP interface (synchronous buffer mode)	1’h0
map_rx_sync_mode	[2]	RW	Rx MAP synchronization mode. Values are: 0: FIFO MAP interface 1: Non-FIFO (synchronized) MAP interface (synchronous buffer mode)	1’h0

Table 6–29. CPRI_MAP_CONFIG—CPRI Mapping Features Configuration—Offset: 0x100 (Part 2 of 2)

Field	Bits	Access	Function	Default
map_mode	[1:0]	RW	Mapping scheme. Values are: 00: Basic mapping scheme (UMTS/LTE standard in which all MAP interfaces use the same sample rate, as described in the CPRI V4.1 Specification sections 4.2.7.2.2 and 4.2.7.2.3). 01: CPRI V4.1 Specification section 4.2.7.2.5: Method 1: IQ sample based. 10: CPRI V4.1 Specification section 4.2.7.2.7: Method 3: Backward compatible. 11: Reserved. Values 01 and 10 indicate advanced AxC mapping modes in which each MAP interface can implement a different channel rate and radio standard.	2'h0

Table 6–30. CPRI_MAP_CNT_CONFIG—Basic UMTS/LTE Mapping Configuration—Offset: 0x104 ***(Note 1)***

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

Note to Table 6–30:

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

Table 6–31. CPRI_MAP_TBL_CONFIG—K Parameter Config for Advanced Table-Based Mapping—Offset: 0x0108 ***(Note 1)***

Field	Bits	Access	Function	Default
RSRV	[31:WIDTH_K]	URO	Reserved.	0
K	[WIDTH_K-1:0]	RW	Number of basic frames in AxC container block.	0

Note to Table 6–31:

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

Table 6–32. CPRI_MAP_TBL_INDEX—Advanced Mapping Configuration Table Index—Offset: 0x10C ***(Note 1)***

Field	Bits	Access	Function	Default
RSRV	[31:11]	URO	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 6–32:

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

Table 6–33. CPRI_MAP_TBL_RX—Advanced Mapping Rx Configuration Table—Offset: 0x110 *(Note 1)*

Field	Bits	Access	Function	Default
RSRV	[31:21]	UR0	Reserved.	11'h0
position	[20:16]	RW	Starting bit position of IQ sample in 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 timeslot.	1'h0

Note to Table 6–33:

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

Table 6–34. CPRI_MAP_TBL_TX—Advanced Mapping Tx Configuration Table—Offset: 0x114 *(Note 1)*

Field	Bits	Access	Function	Default
RSRV	[31:21]	UR0	Reserved.	11'h0
position	[20:16]	RW	Starting bit position of IQ sample in 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 timeslot.	1'h0

Note to Table 6–34:

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

Table 6–35. CPRI_MAP_OFFSET_RX—MAP Rx Frame Offset—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 CPRI 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 CPRI MAP receiver AxC container block write to each enabled mapN Rx buffer.	8'h0

Table 6–36. CPRI_MAP_OFFSET_TX—MAP Tx Frame Offset—Offset: 0x11C (Part 1 of 2)

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

Table 6–36. CPRI_MAP_OFFSET_TX—MAP Tx Frame Offset—Offset: 0x11C (Part 2 of 2)

Field	Bits	Access	Function	Default
map_tx_offset_z	[15:8]	RW	Hyperframe number for start of read of CPRI 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 CPRI 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

Table 6–37. CPRI_START_OFFSET_RX—Rx Start Frame Offset—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

Table 6–38. CPRI_START_OFFSET_TX—Tx Start Frame Offset—Offset: 0x124

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
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

Table 6–39. 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, 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 6–40. 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, 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 6–41. 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	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.	4'h7

Table 6–42. 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 6–43. CPRI_PRBS_STATUS—PRBS Data Validation Status—Offset: 0x140–0x144 *(Note 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 6–43:

(1) If this CPRI MegaCore function 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 MegaCore function is 24.

Table 6–44. 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 6–45. 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 6–46. CPRI_IQ_RX_BUF_STATUS—MAP Receiver FIFO Buffer Status—Offset: 0x180–0x184 *(Note 1)*

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

Note to Table 6–46:

(1) If this CPRI MegaCore function 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 MegaCore function is 24.

Table 6–47. CPRI_IQ_TX_BUF_STATUS—MAP Transmitter FIFO Buffer Status—Offset: 0x1A0–0x1A4 *(Note 1)*

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

Note to Table 6–47:

(1) If this CPRI MegaCore function 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 MegaCore function is 24.

Ethernet Registers

This section lists the Ethernet registers. Table 6–48 provides a memory map for the Ethernet registers. Table 6–49 through Table 6–64 describe the Ethernet registers in the CPRI MegaCore function.

Table 6–48. CPRI Ethernet Registers Memory Map

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
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 6–49. ETH_RX_STATUS—Ethernet Receiver Module Status—Offset: 0x200

Field	Bits	Access	Function	Default
RSRV	[31:7]	URO	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 6–50. 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 from the Ethernet channel.	1’h0
rx_abort	[1]	RO	Indicates the current Ethernet Tx packet is aborted.	1’h0
rx_ready	[0]	RO	Indicates that the Ethernet Tx module is ready to receive at least one 32-bit word of data from the Ethernet channel.	1’h0

Table 6–51. 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
RSRV	[3:2]	RO	Reserved.	2’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 6–52. 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 6–53. 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 6–54. ETH_RX_DATA—Ethernet Rx Data—Offset: 0x214

Field	Bits	Access	Function	Default
rx_data	[31:0]	RO	Ethernet Rx frame data. If the Ethernet receiver module takes Ethernet data from this register, if data is not ready when the module expects it, the Ethernet receiver module aborts the packet.	1'h0

Table 6–55. 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. If the Ethernet receiver module takes Ethernet data from this register, it inserts wait states on the Ethernet channel until data is ready, unless the CPU times out the operation.	1'h0

Table 6–56. 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.	1’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 6–57. ETH_TX_DATA—Ethernet Tx Data—Offset: 0x220

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, if data is not ready when the module expects it, the Ethernet transmitter module aborts the packet.	1'h0

Table 6–58. 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 6–59. 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 6–60. 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 6–61. 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 6–62. 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 6–63. 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 6–64. 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 6–65 provides a memory map for the HDLC registers. Table 6–66 through Table 6–79 describe the HDLC registers in the CPRI MegaCore function.

Table 6–65. 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 6–66. HDLC_RX_STATUS—HDLC Receiver Module Status—Offset: 0x300 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:7]	URO	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

Table 6–66. HDLC_RX_STATUS—HDLC Receiver Module Status—Offset: 0x300 (Part 2 of 2)

Field	Bits	Access	Function	Default
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
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 to be transmitted on the HDLC channel.	1'h0

Table 6–67. HDLC_TX_STATUS—HDLC Transmitter Module Status—Offset: 0x304

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

Table 6–68. HDLC_CONFIG—HDLC Feature Configuration 1—Offset: 0x308 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:20]	URO	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

Table 6–68. HDLC_CONFIG—HDLC Feature Configuration 1—Offset: 0x308 (Part 2 of 2)

Field	Bits	Access	Function	Default
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
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 6–69. 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 6–70. 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 6–71. 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 6–72. 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 6–73. 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 6–74. 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 6–75. 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 6–76. 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 6–77. HDLC_CONFIG_3—HDLC Feature Configuration 3—Offset: 0x32C

Field	Bits	Access	Function	Default
RSRV	[31:17]	UR0	Reserved.	15'h0
tx_start_thr	[16:1]	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 before transmission starts. Packets longer than the Tx buffer are aborted.	1'h0

Table 6–78. 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 6–79. 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

7. Testbenches

Introduction

The CPRI MegaCore function includes five 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 auto-rate negotiation.

All four demonstration testbenches demonstrate the following functions:

- Writing to the registers
- Frame synchronization process
- Transmission and reception of CPRI link data

In addition, the individual testbenches demonstrate the functions shown in Table 7–1.

Table 7–1. Additional Functions Demonstrated by Individual Testbenches

Testbench	Transmission and Reception of Data on Interface			Auto-Rate Negotiation of CPRI Line Rate
	Antenna-Carrier	MII	AUX	
tb_altera_cpri.vhd	✓	—	✓	—
tb_altera_cpri_mii.vhd	✓	✓	✓	—
tb_altera_cpri_mii_noiq.vhd	—	✓	✓	—
tb_altera_cpri_autorate.vhd	—	—	—	✓
tb_altera_cpri_c4gx_autorate.vhd	—	—	—	✓

Each testbench consists of a CPRI MegaCore function and a testbench that initializes the CPRI MegaCore function and sends the generated data to the CPRI MegaCore function interfaces listed in Table 7–1. In the testbenches, the CPRI MegaCore function'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 MegaCore function's interfaces. The initialization process requires that the testbench module write to and read from the CPRI MegaCore function registers through its CPU interface.

Figure 7–1 illustrates the non-MII interface testbench, **tb_altera_cpri.vhd**. Figure 7–2 illustrates the MII interface testbench, **tb_altera_cpri_mii.vhd**. Figure 7–3 illustrates the MII interface, no IQ interfaces testbench, **tb_altera_cpri_mii_noiq.vhd**. Figure 7–4 and Figure 7–5 illustrate the auto-rate negotiation testbenches, **tb_altera_cpri_autorate.vhd**, which targets a Stratix IV GX device, and **tb_altera_cpri_c4gx_autorate.vhd**, which targets a Cyclone IV GX device.

Figure 7–1. CPRI MegaCore Function Non-MII Interface Demonstration Testbench (tb_altera_cpri.vhd)

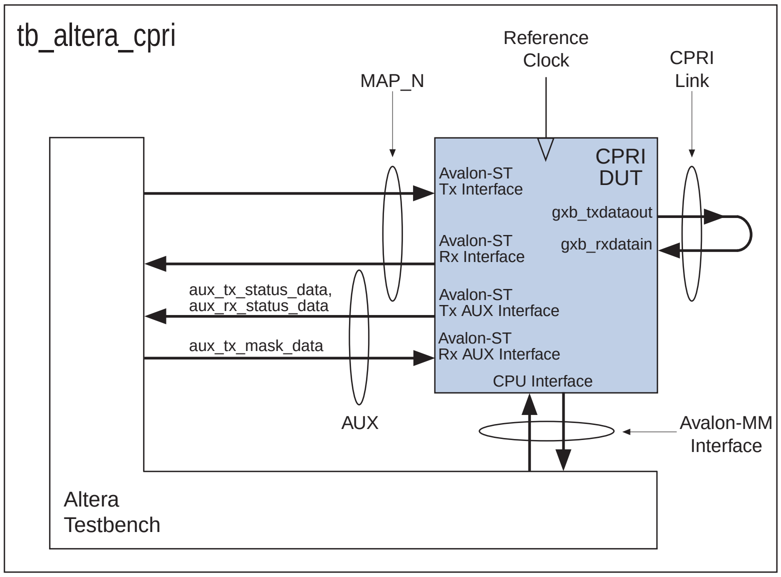


Figure 7–2. CPRI MegaCore Function MII Interface Demonstration Testbench (tb_altera_cpri_mii.vhd)

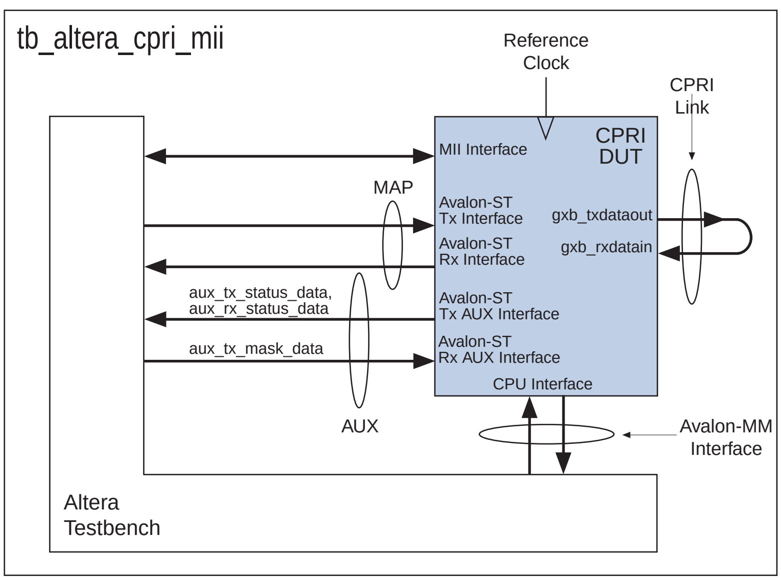


Figure 7–3. CPRI MegaCore Function MII Interface No IQ Demonstration Testbench (tb_altera_cpri_mii_noiq.vhd)

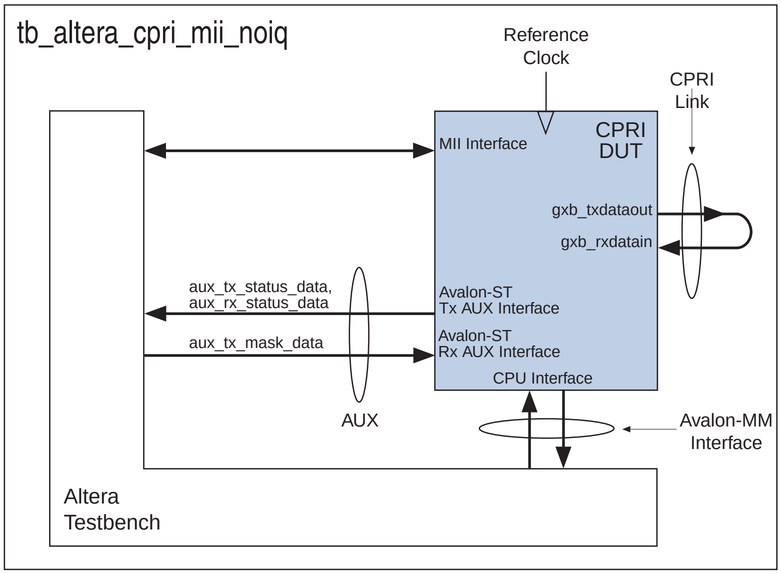


Figure 7–4. CPRI MegaCore Function Auto-rate Negotiation Demonstration Testbench (tb_altera_cpri_autorate.vhd)

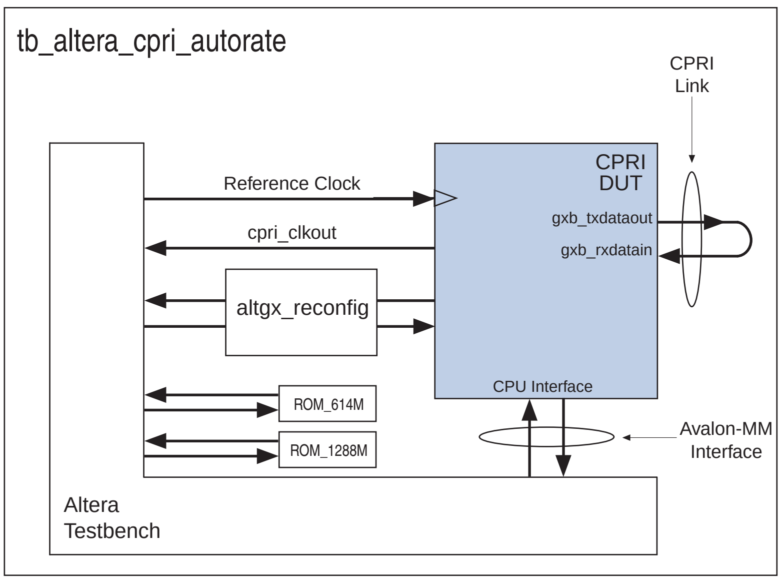
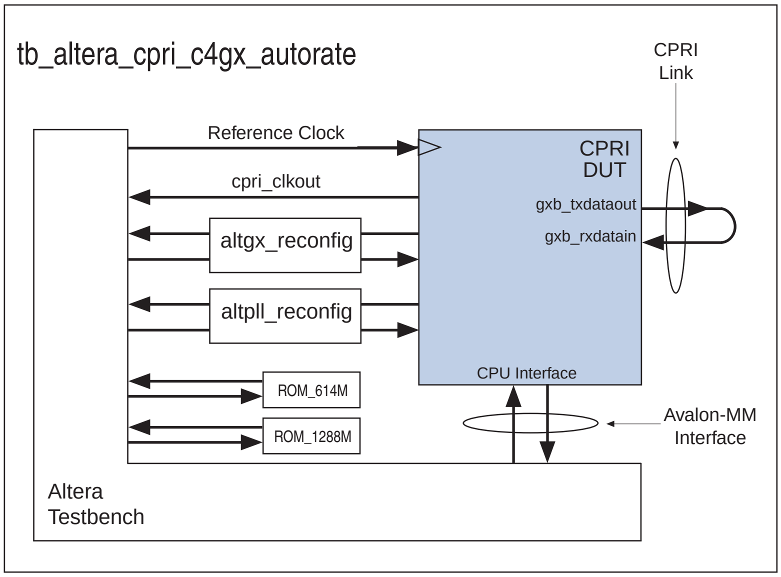


Figure 7–5. CPRI MegaCore Function Cyclone IV GX Auto-rate Negotiation Demonstration Testbench (tb_altera_cpri_c4gx_autorate.vhd)



Test Sequence

The testbench starts by resetting the CPRI MegaCore function. Table 7–2 lists the frequencies of the clock inputs to the MegaCore function.

Table 7–2. Clock Frequencies for CPRI MegaCore Function Under Test

Clock	Frequency
gxb_refclk	61.44 MHz
cpu_clk	30.72 MHz
clk_ex_delay	30.96 MHz
mapN_tx_clk	3.84 MHz

After coming out of the reset state, the MegaCore function 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 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 support three antenna-carrier interfaces; the **tb_altera_cpri_autorate**, **tb_altera_cpri_c4gx_autorate**, and **tb_altera_cpri_mii_noiq** testbenches 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 interface, and checks that the data appears as expected on the outgoing MII interface after loopback through the CPRI link (**tb_altera_cpri_mii** and **tb_altera_cpri_mii_noiq** only).

  This test also checks the MII interface 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 interface 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 0x2. 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 7–3 shows the registers that are programmed in the **tb_altera_cpri** and **tb_altera_cpri_mii** DUTs. For a full description of each register, refer to Chapter 6, Software Interface.

Table 7–3. 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 auto-rate negotiation testbench performs auto-rate negotiation. Refer to Appendix B, Implementing CPRI Link Auto-Rate Negotiation for details.

Running the Testbenches

To run the CPRI MegaCore function testbenches, perform the following steps:

1. In the Quartus II software, create a Quartus II project using the **New Project Wizard** available from the File menu. The project targets the same device as your intended DUT. Refer to Table 7–4.
2. Generate the CPRI MegaCore function DUT instance with the properties shown in Table 7–4.

Table 7–4. MegaWizard Plug-in Manager Options for CPRI MegaCore Function DUT

Parameter	Value
Device family	**tb_altera_cpri_autorate**: Stratix IV GX **tb_altera_cpri_c4gx_autorate**: Cyclone IV GX All other testbenches: Any
Language	VHDL
File name *(1)*	<*working directory*>**\cpri_top_level**
Line rate	0.6144 Gbps
Operation mode	Master *(2)*
Number of antenna-carrier interfaces	**tb_altera_cpri** and **tb_altera_cpri_mii**: 3 **tb_altera_cpri_mii_noiq**, **tb_altera_cpri_autorate**, and **tb_altera_cpri_c4gx_autorate**: 0
Enable auto-rate negotiation	**tb_altera_cpri_autorate** and **tb_altera_cpri_c4gx_autorate**: On All other testbenches: Off
Include MAC block	**tb_altera_cpri** and **tb_altera_cpri_c4gx_autorate**: On All other testbenches: Off

Notes to Table 7–4:

(1) 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.

(2) 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 interface must lock onto the K28.5 character before the transmit CPRI 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 MegaCore function.

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 Memory Initialization Files (**.mif**) to configure the altpll_reconfig block. To generate the files, follow these 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 1.228 Gbps, and regenerate.
   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. Edit the **rom_stratix4gx_**<*rate*>**_m.vhd** or **rom_cyclone4gx_**<*rate*>**_reconfig.vhd** files to remove the string `alt<chars>gxb` from the **.mif** file names.
   m. 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.
4. If you are using ModelSim SE or ModelSim AE, turn off simulation optimization by performing the following steps:
   a. In the ModelSim simulator, on the **Compile** menu, click **Compile Options**.
   b. Perform one of the following actions:
      i. If you are using the ModelSim SE simulator, in the Compiler Options dialog box, on the **VHDL** tab, turn off **Use vopt flow**.
      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**.

5. In the ModelSim simulator, change directories to your testbench directory, <*working directory*>**/cpri_top_level_testbench/altera_cpri**. This folder contains the testbench VHDL (**.vhd**) files and the **.do** files to run the testbenches.
6. To set up the library files for the Mentor Graphics ModelSim SE simulator, perform the following steps:
   a. Create a library folder, <*working directory*>**/cpri_top_level_testbench/altera_cpri/lib**.
   b. Depending on whether you are using Verilog HDL models or VHDL models, copy the appropriate versions of the following files from <*Quartus II installation directory*>**/quartus/eda/sim_lib** to the new library folder:

      altera_mf_components.vhd
      altera_mf.v or **.vhd**
      220pack.vhd
      220model.v or **.vhd**
      sgate_pack.vhd
      sgate.v or **.vhd**
      arriaii_hssi_components.vhd
      arriaii_hssi_atoms.v or **.vhd**
      stratixiv_hssi_components.vhd
      stratixiv_hssi_atoms.v or **.vhd**
      cycloneiv_hssi_components.vhd
      cycloneiv_hssi_atoms.v or **.vhd**
      cycloneiv_atoms.vhd
      cycloneiv_components.vhd
      arriaiigz_hssi_components.vhd
      arriaiigz_hssi_atoms.v or **.vhd**
      arriaiigz_atoms.vhd
      arriaiigz_components.vhd
7. Perform the following edits in the appropriate **compile[**_<*variation*>**]**_<*HDL*>**.do** file, depending on your ModelSim version and HDL:
   - To prepare to simulate VHDL files with ModelSim AE, perform the following edits:
     - Comment out all `vmap` commands.
     - Change all instances of **src/cpri_top_level.vho** to **../../cpri_top_level_sim/cpri_top_level.vho**.
     - Change all instances of **test/tb_altera_cpri[**_<*variation*>**].vhd** to **tb_altera_cpri[**_<*variation*>**].vhd**.
     - For the auto-rate negotiation testbenches, change all remaining instances of **src/**<*file*>**.vhd** to <*file*>**.vhd**.

- To prepare to simulate VHDL files with ModelSim SE, perform the following edits:
  - Change all instances of **src/cpri_top_level.vho** to **../../cpri_top_level_sim/cpri_top_level.vho**.
  - Change all instances of **test/tb_altera_cpri[_**<*variation*>**].vhd** to **tb_altera_cpri[_**<*variation*>**].vhd**.
  - For the auto-rate negotiation testbenches, change all remaining instances of **src/**<*file*>**.vhd** to <*file*>**.vhd**.
- To prepare to simulate Verilog HDL files with ModelSim SE, perform the following edits:
  - Change all instances of **src/cpri_top_level.v** to **../../cpri_top_level_sim/cpri_top_level.v**.
  - Change all instances of **test/tb_altera_cpri[_**<*variation*>**].v** to **tb_altera_cpri[_**<*variation*>**].v**.
  - For the auto-rate negotiation testbenches, change all remaining instances of **src/**<*file*>**.v** to <*file*>**.v**.

8. To compile and run the appropriate testbench for the DUT you generated in step 1, 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 MII 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 a CPRI MegaCore function.

To initialize the CPRI MegaCore function, follow these 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 MegaCore function 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, assert the `gxb_powerdown` signal.
3. Write the value 0x21 to the `CPRI_CONFIG` register (`0x8`). This `CPRI_CONFIG` register setting enables the CPRI MegaCore function to start sending K28.5 symbols on the CPRI link.
4. Observe the `cpri_rx_state` output signal as it transitions from value 0x1 to value 0x2 to value 0x3. When it has value 0x3, the CPRI MegaCore function CPRI receiver interface is in the HFNSYNC state. The `cpri_rx_state` output signal appears on `extended_rx_status_data[1:0]`.
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 MegaCore function 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 Auto-Rate Negotiation

The CPRI MegaCore function supports auto-rate negotiation. This feature allows you to specify that the CPRI MegaCore function 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 interface receiver to implement this capability in your design, with the help of logic connected outside the MegaCore function.

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 auto-rate negotiation feature, in the ALTGX parameter editor, you must perform the following actions:

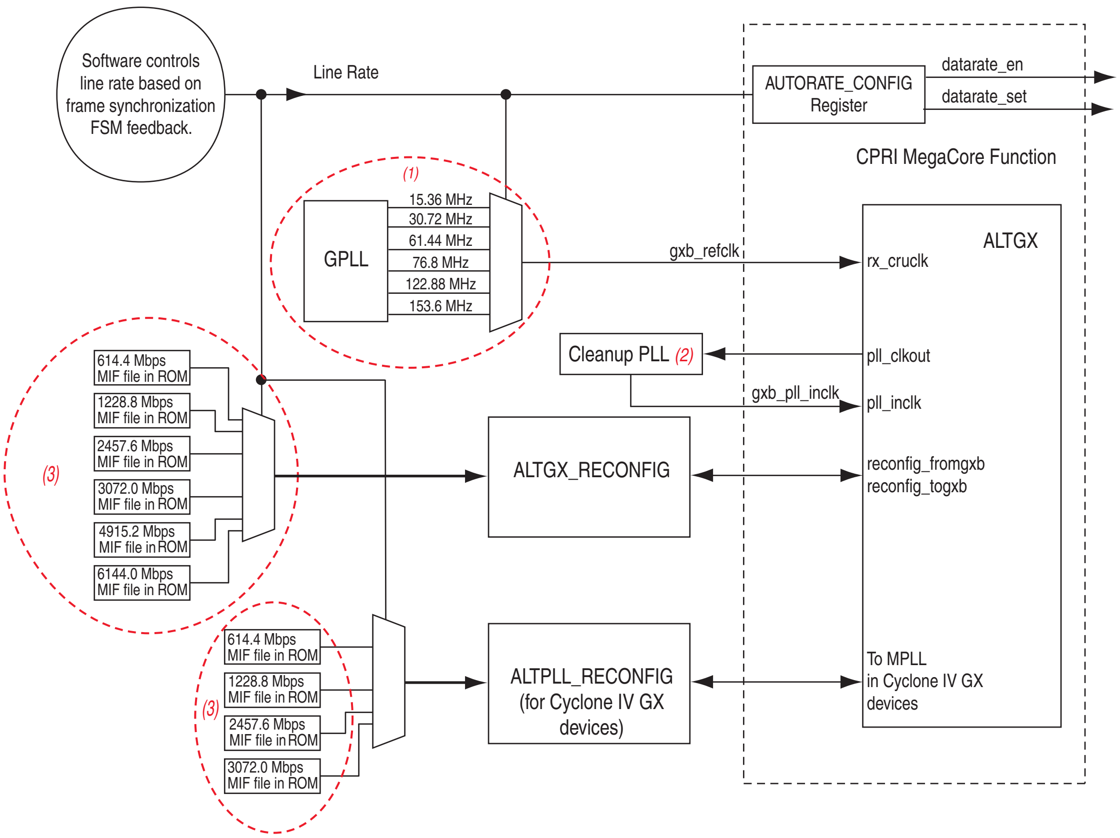
- In the CPRI parameter editor, enable auto-rate negotiation.
- In the CPRI parameter editor, set the transceiver to run at the highest CPRI line rate for this device.
- Include additional external data and logic in your design, such as input data to the ALTGX_RECONFIG megafunction for each CPRI line rate to be checked.
- For Cyclone IV GX devices, you must implement logic to perform auto-rate negotiation by reconfiguring the transceiver directly, using the compulsory ALTGX_RECONFIG megafunction.

In Cyclone IV GX devices, auto-rate 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 the *Transceivers* section 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 auto-rate negotiation logic block diagrams for CPRI MegaCore functions 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, 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. The examples clarify the functionality provided by the CPRI MegaCore function, and the logic and data you must configure in your design outside the MegaCore function.

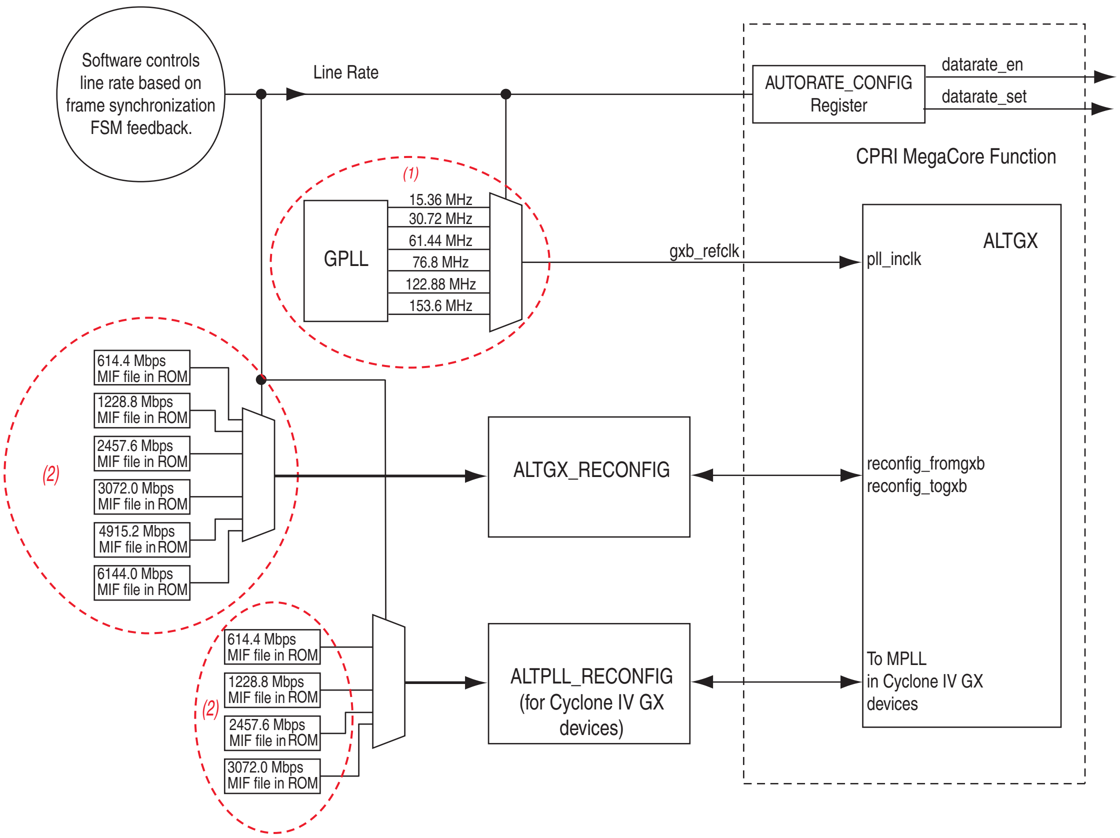
Figure B–1. Auto-Rate Negotiation in Slave Clocking Mode



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. Auto-Rate 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 MegaCore Function for Auto-Rate Negotiation

To ensure that the MegaCore function implements auto-rate negotiation correctly, while configuring your CPRI MegaCore function, enable auto-rate negotiation and set the CPRI line rate to the maximum line rate supported by the device family.

Running Auto-Rate Negotiation

After your CPRI MegaCore function is configured on the device, the auto-rate negotiation logic you configured in your design outside the MegaCore function must perform certain steps to activate the auto-rate negotiation support logic in the MegaCore function. This section describes these steps.

To start auto-rate negotiation in your CPRI MegaCore function, in addition to its own initialization outside the CPRI MegaCore function, 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 6–21 on page 6–10. You can read this value on the `datarate_en` output signal.

2. Set the logic that feeds the gxb_refclk input to the CPRI MegaCore function 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 with the **.mif** file for the desired CPRI line rate.
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 MegaCore function 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 4–10 on page 4–20 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.

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
December 2010	10.1	■ Added support for Arria II GZ devices. ■ Added support for additional CPRI data rates in Arria II GX devices ■ 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 auto-rate negotiation in Cyclone IV GX devices.
July 2010	10.0	■ Added support for Cyclone IV GX devices. ■ Added GUI parameter to enable auto-rate negotiation and two signals to support visibility of the feature status. ■ Enhanced descriptions of MII interface, 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 auto-rate 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
Non-technical 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.
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 within 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.
	A question mark directs you to a software help system with related information.
	The feet direct you to another document or website with related information.
CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
WARNING	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.