10-Gbps Ethernet Reference Design User Guide

CAUTION The IP described in this document is scheduled for product obsolescence and discontinued support as described in PDN1207. Therefore, Altera® does not recommend use of this IP in new designs. For more information about Altera's current IP offering, refer to Altera's Intellectual Property website.

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

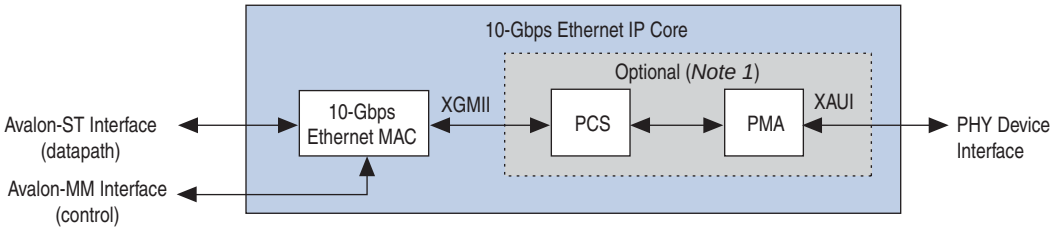
IP Core Version: 11.0
Document Date: December 2011

1. 10-Gbps Ethernet IP Datasheet

This datasheet describes the Altera® 10-Gbps Ethernet IP core which implements the IEEE 802.3 2005 and 802.1Q Ethernet standards. You can use the Quartus® II software to parameterize and implement this IP core in your design. The 10-Gbps Ethernet IP core is highly configurable. It includes an Ethernet Media Access Controller (MAC) with an Avalon® Streaming (Avalon-ST) interface on the client side, and a XAUI or a standard XGMII interface on the network side. The XAUI interface is implemented as hard IP in an Altera FPGA transceiver or as soft logic, which results in a soft 10GBASE-X XAUI PCS. Alternatively, you can choose to implement a 10-Gbps Ethernet IP core that includes only the MAC or the soft XAUI PCS. Figure 1–1 illustrates the top-level modules of this IP core.

☞ Altera categorizes this IP core as a reference design, described on the Altera website on the 10-Gbps Ethernet Reference Design web page.

Figure 1–1. 10-Gbps Ethernet IP Core Block Diagram *(Note 1)*



Note to Figure 1–1:

(1) You can implement the optional XAUI PCS in an Altera device transceiver or as soft logic, which results in a soft 10GBASE-X XAUI PCS and PMA.

1.1. Supported Features

- Flexible standard interfaces: SDR XGMII-like interface to connect to the internal 10GBASE-X (XAUI) PHY, standard XGMII interface to connect to the external PHY device, hard IP XAUI PCS and PMA to connect to an external optical module.
- Verified and tested in hardware with standard 10-Gbps Ethernet test equipment. The IP core has passed the University of New Hampshire Interoperability Lab (UNH-IOL) 10-Gbps Ethernet tests including MAC, 10GBASE-X physical coding sublayer (PCS), Reconciliation Sublayer, flow control, and XAUI physical media attachment (PMA).
- Management data I/O (MDIO) master interface for external PHY device management.
- Avalon-ST 64-bit wide client interface running at 156.25 MHz with 10-Gbps full-duplex throughput rate.

- Automatic or host-controlled flow control frame transmission can be initiated by either the host using explicit signals or by the host using configuration register. The `xon_req` and `xoff_req` signals can be controlled by either the MAC client or by the host via the configuration registers.
- Programmable pause quanta.
- Parameterizable FIFO size (64 bytes to 64 Kbytes) and programmable threshold levels.
- Programmable MAC addresses and receive packet filtering based on up to five unicast or multicast and broadcast destination MAC addresses.
- Programmable maximum receiving frame length up to 64 KBytes, including jumbo frames (1,519 to 9,618 bytes).
- Support for promiscuous (transparent) and non-promiscuous (filtered) modes of operation.
- Support for virtual LAN (VLAN) and stacked VLAN tagged frames according to the IEEE 802.1Q and 802.1ad (QinQ) standards, respectively.
- Remote (Line) and Local (Client) loopback at XGMII for system test.
- Statistics counters supporting RMON (RFC 2819), Ethernet type MIB (RFC 3635), and interface group MIB (RFC 2863).
- Programmable filtering of received frames with CRC errors, length-check error, or oversized errors.
- Easy-to-use MegaWizard™ GUI for IP parameterization and generation.
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators.
- Verilog HDL and VHDL testbench and verification environment.
- Deficit Idle Count (DIC) is supported

1.2. Performance and Resource Utilization

Table 1–1 shows the typical expected device resource utilization for different IP configurations and parameters, using the Quartus II software version 9.1 targeting a Stratix II GX (EP2SGX30DF780C3) device.

For C4 and C5 device speed grades the f_{MAX} is 156.25 MHz.

Table 1–1. 10-Gbps Ethernet Performance and Resource Utilization—Stratix II GX Device

XAUI	FIFO (eight-byte words)	MDIO	Statistics Counter	ECC	Combinational ALUTs	Logic Registers	f_{MAX} (MHz)			Memory	
							Avalon -MM	Avalon -ST	System Clock	M4K	M512
Yes	No	No	No	No	3,246	3,238	156	—	202	0	0
Yes	256	No	No	No	3,652	3,654	174	199	179	8	0
Yes	256	Yes	No	No	3,803	3,788	170	200	187	8	0
Yes	256	Yes	Yes	No	5,666	5,334	189	218	204	8	0

Table 1–2 shows the typical expected performance, using the Quartus II software v9.1 targeting a Stratix IV (EP4SGX70DF29C3) device.

Table 1–2. 10-Gbps Ethernet Performance and Resource Utilization—Stratix IV Device

XAUI	FIFO (eight-byte words)	MDIO	Statistics Counter	ECC	Combinational ALUTs	Logic Registers	f_{MAX} (MHz)			Memory
							Avalon-MM	Avalon-ST	System Clock	M9K
Yes	256	Yes	Yes	No	5,666	5,346	230	220	197	4
Yes	256	Yes	Yes	Yes	6,808	6,119	187	175	175	6
Soft XAUI only	—	—	—	No	2,545	2,198	—	—	156.25	6
Soft XAUI	256	Yes	Yes	No	8,136	7,478	177	252	188	10
Soft XAUI	256	Yes	Yes	Yes	10,080	8,409	180	175	168	13

Table 1–3 shows the typical expected performance, using the Quartus II software v9.1 targeting a Arria® II GX (EP2AGX45CU17C5) device.

Table 1–3. 10-Gbps Ethernet Performance and Resource Utilization—Arria II GX Device

XAUI	FIFO (eight-byte words)	MDIO	Statistics Counter	ECC	Combinational ALUTs	Logic Registers	f_{MAX} (MHz)			Memory
							Avalon-MM	Avalon-ST	System Clock	M9K
Yes	256	Yes	Yes	No	5,670	5,346	269	209	199	4

1.3. Revision History

Table 1–4 summarizes the new feature and device support history for this IP core.

Table 1–4. New Features and Device Support History (Part 1 of 2)

Version	Date	New Features	Device Support Added	Device Support Level
11.0	December 2011	■ Updated MegaWizard Plug-In to support the Quartus software v11.0 (the MegaWizard Plug-In is not compatible with earlier versions). Only for existing 10-Gbps Ethernet reference designs; for new 10-Gbps Ethernet reference designs, use the the 10-Gbps Ethernet MAC IP core. ■ Revised reset sequence for the transeivers for the soft PCS ■ Hardened `align_status` signal in reset sequence for the transceiver (both hard and soft PCS) ■ Added new Synopsis Design Constraints to remove unnecessary critical warnings	—	—
10.1SP1	May 2011	Corrected errors in the statistics counts (size buckets, uni/multi/broadcast, CRC error counts)	—	—
10.0	July 2010	Added support for pause frame control (PFC). Frames can be passed to the user interface (no internal support for the pause control).	—	—

Table 1–4. New Features and Device Support History (Part 2 of 2)

Version	Date	New Features	Device Support Added	Device Support Level
9.1	December 2009	■ Added support for optional error correcting code (ECC) in memories in the Soft XAUI PC and the transmit (Tx) and receive (Rx) FIFOs on Stratix IV GX devices. ■ For the hard PCS, the local fault register is now asserted after reset until the link is up. ■ On the XGMII interface, the Tx now transmits with a clock shifted 90° and the Rx interface expects a clock that is shifted by 90°. ■ Corrected address for `linkFaultDetect` register.	HardCopy® III HardCopy IV	Preliminary *(1)* Preliminary
9.0	March 2009	■ Added 2 input signals, `xon_request` and `xoff_request` that request XON and XOFF pause frames. ■ Support for 1 primary and 4 supplemental addresses. ■ 64-bit counters shadow upper 32 bits. ■ Soft XAUI and MAC-only options.	Arria II GX	Preliminary
8.1	November 2008	Initial release.	Arria GX Stratix II Stratix II GX Stratix III Stratix IV	Full *(2)* Full Full Full Preliminary

Notes to Table 1–4:

(1) *Preliminary support* means the IP core meets all functional requirements, but may still be undergoing timing analysis for the device family; it can be used in production designs with caution.

(2) *Full support* means the IP core meets all functional and timing requirements for the device family and can be used in production designs.

2. Getting Started with the 10-Gbps Ethernet IP

The chapter provides step-by-step instructions to help you get started using the Altera® 10-Gbps Ethernet IP core. It shows you how to install, generate, and simulate the 10-Gbps Ethernet IP core.

To use this document, you should already be familiar with the Ethernet protocol in general and 10-Gbps Ethernet in particular. You should also have some experience and familiarity with logic design using the Quartus® II software.

Refer to the Quartus II Development Software Literature page for information about the Quartus II design flow, including tutorials.

Altera categorizes this IP core as a reference design, described on the Altera website on the 10-Gbps Ethernet Reference Design web page.

This document describes the following development flow steps:

1. Licensing and Installation
   a. Free Time-Limited Evaluation Using OpenCore Plus Licensing
   b. Purchasing Full License for System Development
   c. Downloading and Installation
2. IP Core Parameterization
   a. Variation Options
   b. Design Flows
3. Functional Verification
   a. Testbench Files
   b. Testbench Utilities—Tasks and Procedures
   c. Running Tests
4. Implementation and Timing Analysis
   a. Synopsis Design Constraints
   b. Placement Constraints
   c. Timing Verification

2.1. Licensing and Installation

Depending on the phase of your design, the following licensing options are available.

2.1.1. Free Time-Limited Evaluation Using OpenCore Plus Licensing

You can use the OpenCore Plus feature to temporarily try out the IP core for free using one of Altera's development boards that include an Altera FPGA. There are two modes of operation:

- *Untethered*—the design runs for a limited time, typically one hour.
- *Tethered*—requires a connection between your board and the host computer. In tethered mode, the device can operate for a longer time or indefinitely.

For more information about the OpenCore Plus hardware evaluation, refer to *AN 320: OpenCore Plus Evaluation of Megafunctions*.

2.1.2. Purchasing Full License for System Development

The use of this 10-Gbps Ethernet IP core is governed by, and is subject to, the terms and conditions of the Altera License Terms and Conditions for 10-Gbps Ethernet IP core.

Refer to License Terms and Conditions for 10-Gbps Reference Design for details about purchasing a full production license.

2.1.3. Downloading and Installation

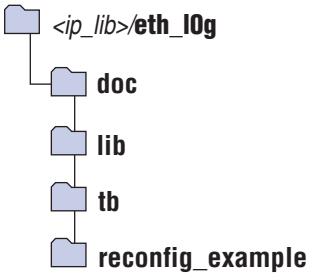
Complete the following steps to download the 10-Gbps Ethernet IP core:

1. Go to the 10-Gbps Ethernet Reference Design web page.
2. To download the library, you must be a registered user. Click the **Download Reference Design** button to complete the registration steps.
3. To download the reference design, click the **Proceed to Download Page** button.
4. On the **Download 10-Gbps Ethernet Reference Design** page, click **Download** to save the compressed file to your working directory. Uncompress and extract the **.zip** file. These instructions refer to your project library directory as `<ip_lib>`.

   Do not unzip to the standard Altera IP core directory **altera**\`<version>`**ip**.

Figure 2–1 shows the directory structure.

Figure 2–1. Structure of Downloaded (Unzipped) IP Core Directories



The **lib** directory is required for all user designs. The other directories contain documentation and other files for an example design.

You can parameterize and instantiate the 10-Gbps Ethernet IP core using either the MegaWizard Plug-In Manager or SOPC Builder design flow. The MegaWizard Plug-In Manager allows you to customize and create RTL for the IP core and then integrate this variant into your overall design.

SOPC Builder allows you can add the 10-Gbps Ethernet IP core directly to a new or existing SOPC Builder system. Use this design flow if your design includes other SOPC Builder components. When you generate an SOPC Builder system, SOPC Builder automatically creates the HDL to connect components in your system design and generates simulation models.

For more information on SOPC Builder, refer to *Volume 4 of the Quartus II Handbook*.

2.1.3.1. Install IP Core Library for MegaWizard Plug-In Manager Flow

Complete the following steps to install the design files for the MegaWizard Plug-In Manager design flow:

1. Choose **Programs > Altera > Quartus II>*<version number>*** (Windows Start menu) to run the Quartus II software.
2. To create a Quartus II project, complete the following steps:
   a. On the File menu, click **New Project Wizard**.
   b. In the **New Project Wizard**, follow the instructions to create a new Quartus II project. Navigation buttons at the bottom of the dialog wizard you through the steps. This user guide uses the project name **tge_91.qpf** in the `<project_dir>`**/tge_91** project design directory.
3. On the Assignments menu, click **Settings**.

4. In the Category list, click **Library**. Specify the following settings under **Project libraries** and **Global libraries**.
   a. In the **Project library name**, navigate to `<ip_lib>`**/eth_10g/lib** and double-click **Open**. You should see `<ip_lib>`**/eth_10g/lib** in the **Project Library Name** window.
   b. Click **Add**. The selected library appears under **Libraries**.
   c. Adapt steps a and b to include `<ip_lib>`**/eth_10g/lib** under **Global libraries**.
   d. Click **OK**.
5. On the File menu click **Save Project**. Type the project name `tgb_91`.

   You are now ready to create variations of the Altera 10-Gbps Ethernet design using the MegaWizard Plug-In Manager. You can locate the 10-Gbps Ethernet IP core in the MegaWizard Plug-In Manager by expanding **Installed Plug-Ins** > **Interfaces** > **Ethernet**.

2.1.3.2. Install IP Core Libraries for the SOPC Builder Flow

Complete the following steps to install the design files for the system-on-a-programmable chip (SOPC) Builder design flow:

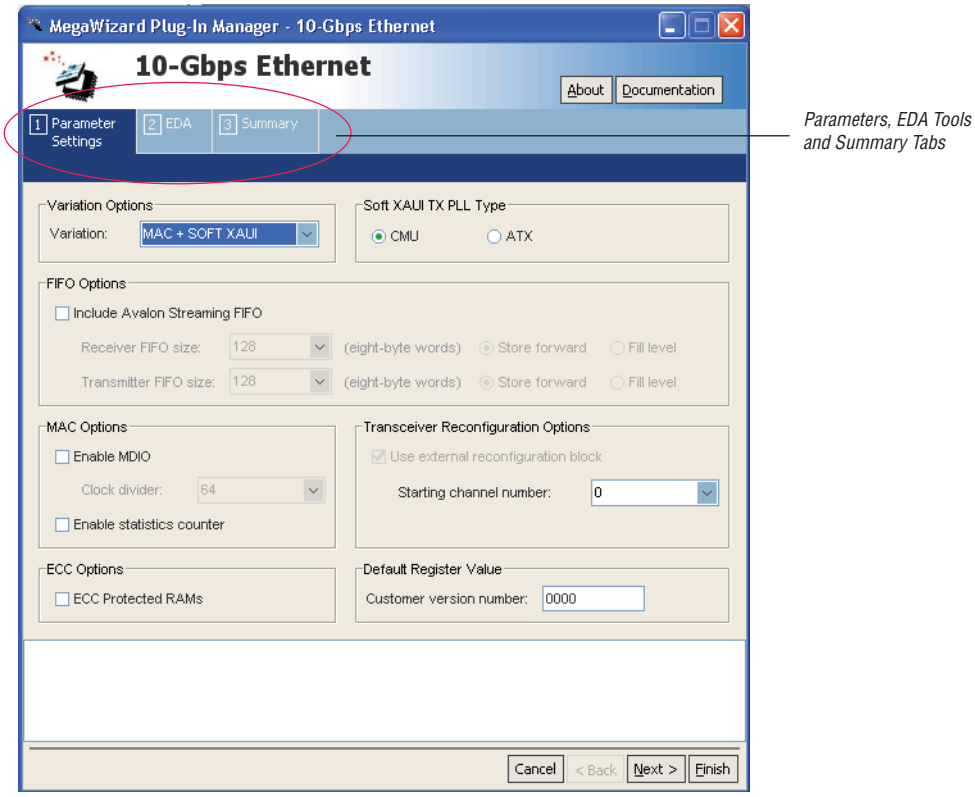
1. Install libraries for the MegaWizard Plug-In Manager as described in Section 2.1.3.1.
2. To start SOPC Builder, on the Tools menu, click **SOPC Builder**.
3. In the **Create New System** dialog box, click **Cancel**.
4. On the SOPC Builder Tools menu, click **Options**.
5. Under **Category**, click **IP Search Path**.
6. Click **Add**.
7. In the **Open** dialog box, navigate to `<ip_lib>`**/eth_10g/lib/ip_toolbench**.
8. Click **Open**.
9. Click **Finish**. You can locate the 10G-Gbps Ethernet IP core in SOPC Builder by expanding: **Interface Protocols -> Ethernet**.

For more information about SOPC Builder, refer to *SOPC Builder* which is volume 4 of the Quartus II Handbook.

2.2. IP Core Parameterization

This section provides an overview of the parameters that you can choose to customize your 10-Gbps Ethernet design. Figure 2–2 shows the MegaWizard Plug-In Manager.

Figure 2–2. 10-Gbps Ethernet Parameter Settings



Depending on the device and options that you choose, some of the parameters may not be available.

2.2.1. Variation Options

Table 2–1 describes the variation options available on the **Parameter Settings** tab.

Table 2–1. Variation Options (Part 1 of 2)

Parameter Settings	Description
MAC + XGMII	Creates a IP core with a media access control (MAC) using an Avalon® Streaming (Avalon-ST) interface on the client side and 32-bit standard DDR XGMII interface operating at 156.25 MHz on the network side.
MAC + XAUI	Creates a IP core that combines a MAC and a 10GBASE-X hard macro PHY with an Avalon-ST interface on the client side and standard XAUI interface on the network side.
MAC + Soft XAUI	Creates a IP core that has the same functionality and external interfaces as a MAC with 10GBASE-X hard macro PHY; however, the physical coding sublayer (PCS) is implemented in soft logic instead of a hard macro.This option is only available for Stratix IV devices.

Table 2–1. Variation Options (Part 2 of 2)

Parameter Settings	Description
Soft XAUI only	The **Soft XAUI only** option is only available on Stratix IV devices. It combines a 10GBASE-X XAUI soft PCS and a hard PMA. This configuration allows the design to use all the available transceivers. The soft XAUI PCS includes the transceiver megafunction and the reset sequence controller, which applies the transceiver reset signals with the required constraints.
Soft XAUI Tx PLL Type	These options are only available when you select **MAC + Soft XAUI** or **Soft XAUI Only.** These are two different PLLs: **CMU**–(clock multiplier unit) designed to achieve low Tx channel-to-channel skew. **ATX**–(auxiliary transmit) designed to improve jitter performance. For more information refer to *AN 578: Manual Placement of CMU PLLs and ATX PLLS in Stratix IV GX and GT Devices* and *Volume 3* of the *Stratix IV Device Handbook*.
MAC only	Creates a IP core that includes the MAC with an Avalon-ST interface on the client side and 64-bit XGMII interface running at 156.25MHz on network side. This interface functions like an SDR interface.

2.2.2. FIFO Options

In most applications, the client side interface of the IP core includes a FIFO between the client and the MAC. Table 2–2 describes the options that are available when you include the FIFO in the IP core.

Table 2–2. FIFO Options

Parameter	Value	Description
Receiver FIFO size	**8,16,32,64,128,256,512,1024,4096,8192**	Selects the depth of the FIFO in 8-byte words in each direction. The usable FIFO size is <*size*> – 1.
Transmitter FIFO size		
Mode	**Store forward**	The packet information passes from the write side to the read side only when the end of packet (EOP) is asserted on the write side. When you select **Store forward**, you should select the FIFO size to accommodate the longest possible frame in the system with some overhead. Altera recommends twice the maximum possible frame size as a minimum. In most cases, **Store forward** mode increases the latency and requires a deeper FIFO.
	Fill level	In **Fill level** mode**,** the FIFO begins passing data to the read side when a configurable number of bytes are available or an EOP is received.

2.2.3. MAC Options

Table 2–3 describes the MAC options.

Table 2–3. MAC Options

Parameter	Values	Description
Enable MDIO	**On/Off**	When you turn this option **On**, the IP core instantiates an management data I/O (MDIO) master. The MDIO interface provides an Avalon Memory-Mapped (Avalon-MM) to MDIO bridge to control an external PHY.
Clock divider	**4,8,16,32, 64,128,256**	The clock divider provides the division ratio for `sysclk` to generate the preferred MDIO clock (MDC). The division factor must be defined such that the MDC frequency does not exceed 2.5 MHz.
Enable statistics counter	**On/Off**	When you turn this option **On**, the IP core includes registers to report statistics on frames transmitted and received.

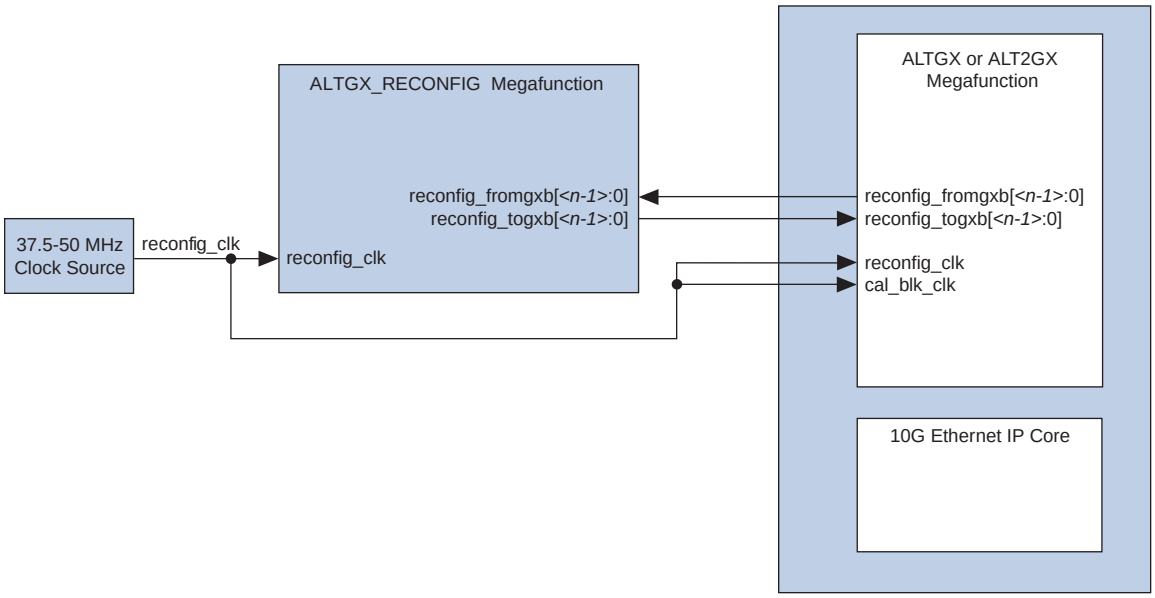
2.2.4. Transceiver Reconfiguration Options

Depending on the device and variant that you choose, reconfiguration of the transceiver may be an optional or required feature.

- If you select a Stratix II GX device and implement the **MAC + XAUI** variant, you can turn on **Use external reconfig block**, you can modify the properties of the transceiver using the ALTGX_RECONFIG megafunction.
- If you select a Stratix IV GX or Arria II GX device with any variant that includes XAUI, you must include the ALTGX_RECONFIG megafunction in your design; this option is always turned on.
- For Arria GX devices, this option is not available.

In any configuration that requires the ALTGX_RECONFIG megafunction, you must instantiate it in your design and connect it to the 10-Gbps Ethernet IP core as shown in Figure 2–3.

Figure 2–3. Top-Level Block Diagram for 10-Gbps Ethernet, ALTGX Transceiver, and ALTGX_RECONFIG Megafunction



For more information on the ALT2GXB_RECONFIG megafunction, refer to the *Stratix II GX ALT2GXB_RECONFIG Megafunction User Guide*.

2.2.5. ECC Options

For Stratix IV devices, you can turn on **ECC Protected RAMs** option to configure single-bit error correction, double-bit error detection (SECDED) for data words up to 64 bits wide on the transmit and receive FIFOs and in the Soft XAUI PCS memory, for applications that require a highly-reliable 10-Gbps Ethernet solution. When you generate the IP core by clicking **Finish** in the MegaWizard interface, an ALTECC megafunction is automatically instantiated to implement this extra memory data protection.

If you turn on the **ECC Protected RAMs**, the IP core includes a set of error insertion registers to support ECC testing, and a set of ECC statistics counters that accumulate the counts of various types of ECC errors as they are detected, corrected, or both detected and corrected.

Turning on the **ECC Protected RAMs** affects the size and maximum achievable frequency of your IP core. For example, turning on this feature increases the widths of the ECC-protected FIFOs.

For information about the ALTECC megafunction, refer to the *Error Correction Code (ALTECC_ENCODER and ALTECC_DECODER) Megafunctions User Guide*.

2.3. Design Flows

The following sections explain how to customize your 10-Gbps Ethernet IP core using the MegaWizard Plug-in Manager and SOPB Builder design flows.

2.3.1. MegaWizard Plug-in Manager Design Flow

To customize you 10-Gbps Ethernet design using the MegaWizard Plug-In Manager design flow, complete the following steps.

1. Open your Quartus II project file, **tge_91.qpf**.
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 megafunction variation. (The name you choose for your variation must not start with a number and should different from component names in the 10-Gbps Ethernet library that you installed.)
3. Specify the parameters on the **Parameter Settings** tab.

   For detailed explanation of the parameters, refer to "IP Core Parameterization" on page 2–4.
4. On the **EDA** tab, turn on **Generate simulation model** to generate an IP functional simulation model for the IP core in the selected language.

   An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software.

   ☞ Some third-party synthesis tools can use a netlist that contains only the structure of the IP core, but not detailed logic, to optimize performance of the design that contains the IP core. If your synthesis tool supports this feature, turn on **Generate netlist**.

   CAUTION Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a non-functional design.
5. On the **Summary** tab, select the files you want to generate. A grey checkmark indicates a file that is automatically generated. All other files are optional.
6. Click **Finish** to generate the IP core and supporting files.

   You can now integrate the 10-Gbps Ethernet variant generated by the MegaWizard into your system design with other custom logic.

☞ The MegaWizard generates a sample testbench, RTL files, SDC timing constraints, and placement and routing constraints. Their usage is explained in the following sections.

2.3.2. SOPC Builder Design Flow

In SOPC Builder you add the IP core directly to a new or existing SOPC Builder system. If your system includes other SOPC Builder components, such as the Nios II processor, external memory controllers, or scatter-gather DMA controllers, you can quickly create an SOPC Builder system with an Ethernet interface.

2.3.2.1. Specify Parameters

Follow the steps below to specify 10-Gbps Ethernet parameters using the SOPC Builder flow.

1. Open your Quartus II project file.
2. Launch **SOPC Builder** from the Tools menu.
3. For a new system, specify the system name and language. This example uses the system name **tge_system** and selects **Verilog** as the Target HDL.
4. Add 10-Gbps Ethernet to your system from the **System Contents** tab.

You can find **10-Gbps Ethernet** by expanding **Interface Protocols > Ethernet** or by typing any string from the component name into the search field under the Component Library list.

5. Specify options on the **Parameter Settings** tab.

For a detailed explanation of the parameters, refer to the "IP Core Parameterization" on page 2–4.

2.3.2.2. Complete the SOPC Builder System

After you define the IP core parameters, you can integrate the core into a system design and connect the 10-Gbps interfaces. This document describes a very simple SOPC Builder system that includes two additional components, the SPI Slave to Avalon-MM Master Bridge and Data Format Adapter. The design creates a local loopback from the Tx to the Rx interfaces of the 10-Gbps Ethernet IP core. This example only demonstrates the SOPC Builder design flow; it is not a functional design.

Refer to Application Note 588, *10-Gbps Ethernet Hardware Demonstration Reference Design* for a more complete design example.

Follow the steps below to complete the SOPC Builder system.

1. To add the SPI Slave to your system, expand **Bridges and Adapters > Memory Mapped** and double-click **SPI Slave to Avalon Master Bridge**. Click **Finish** to add this component to your system.

   As you add components to your system SOPC Builder reports errors in the console window. These errors disappear as you complete your system.

2. To add the Avalon-ST Data Format Adapter to your system, expand **Bridges and Adapters > Streaming** and double-click **Avalon-ST Data Format Adapter**.
3. On the **Parameter Settings** page, specify the following parameters shown in Table 2–4.

Table 2–4. Avalon-ST Data Format Adapter Parameters (Part 1 of 2)

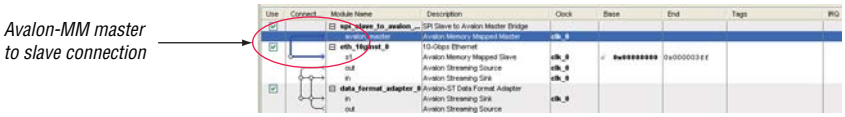
Parameter	Value
Data Symbols Per Beat	8
Include Empty Signal	**AUTO**
Channel Signal Width (bits)	0
Max Channel	0
Include Packet Support	Turn this option on

Table 2–4. Avalon-ST Data Format Adapter Parameters (Part 2 of 2)

Parameter	Value
Error Signal Width (bits)	1
Ready Latency	1
Data Bits Per Symbol	8

4. Click **Finish** to add this component to your system.
5. To complete this design, create the following connections:
6. Connect the spi_slave_to_avalon_mm_master_bridge_0 `avalon_master` Avalon master port to the eth_10ginst_0 `s1` Avalon slave port using the following procedure:
   a. Click on the `avalon_master` port then hover in the Connections column to display possible connections.

Figure 2–4. SOPC Builder Avalon-MM Master to Slave Connection



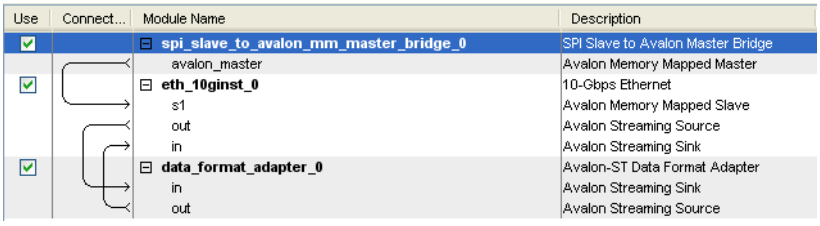
   b. Click on the open dot at the intersection of the eth_10ginst_0 `s1` port and the spi_slave_to_avalon_mm_master_bridge_0 `avalon_master` to create a connection.
7. Repeat step 6 to make the connections listed in Table 2–5.

Table 2–5. SOPC Builder Connections

Make Connection From:	To:
eth_10ginst_0 `out` port	data_format_adapter_0 `in` port
eth_10ginst_0 `in` port	data_format_adapter_0 `out` port

Figure 2–5 shows the completed system.

Figure 2–5. SOPC Builder System Connections



8. Click the **System Generation** tab.
9. If you want to simulate your SOPC builder system, select **Simulate** on the **System Generation** tab to generate a functional simulation model for the system.

10. Click **Generate** to generate the system. The 10-Gbps Ethernet variant is **eth_10ginst_0.v**. A report file, *<variation_name>***.html** describes the HDL files that make up the design and the top-level signals.

2.4. Functional Verification

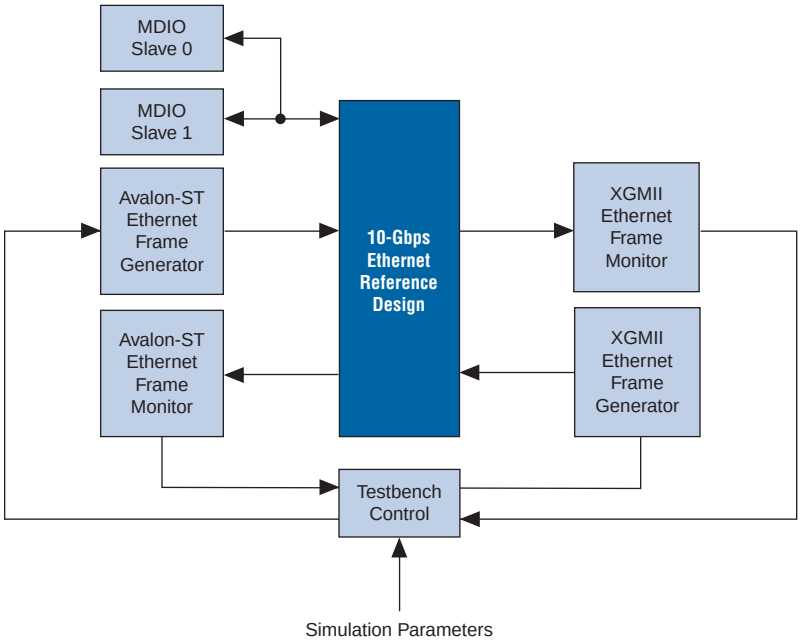
Altera provides a simple test infrastructure for basic functional verification of the customized IP core. This testbench is automatically generated when you generate your 10-Gbps Ethernet IP core. The details of the verification environment and how to use it is being described below.

The testbench consists of the following components:

- An Avalon-ST client packet generator
- Avalon-ST client packet checker
- The 10-Gbps Ethernet device under test (DUT)
- An XGMII Ethernet packet generator
- An XGMII Ethernet packet checker
- An Avalon-MM configuration module

Figure 2–6 illustrates the top-level modules in the testbench. The testbench files are stored in clear text in `<ip_lib>`**/eth_10g/tb**.

Figure 2–6. Block Diagram of the Testbench



The testbench sends and checks received Ethernet frames. The maximum supported frame size is 16 KBytes. It performs the following functions:

- Generates Ethernet frames and checks their validity

- Generates Ethernet frames with CRC errors
- Generates frames on the Avalon-ST interface
- Provides access to the IP core registers via Avalon-MM interface
- Generates frames with virtual LAN (VLAN) or stacked VLAN tags
- Performs MDIO reads and writes

2.4.1. Testbench Files

Table 2–6 describes the files that implement the testbench.

Table 2–6. Testbench Files

File Name	Description
demo_hookup.iv	The top-level testbench file. It includes the customized 10-Gbps Ethernet IP core which is the device under test (DUT), a client packet generator, and a client packet checker along with other logic blocks. This file (**demo_hookup.iv**) contains the DUT, packet generator, and packet checker, along with configuration logic.
demo_hutils.iv, tb_utils.v	These files include various test utilities to configure the DUT, the packet generator, and the packet checker. These utilities create tests such as **tb.v**.
tb.v	This file includes tests that configure the testbench and transmit and receive packets via the MAC Tx and Rx ports.
demo_run_modelsim.tcl	This is the simulation script that you can use to compile all files and run the test case.

2.4.2. Testbench Utilities—Tasks and Procedures

The **tb_utils.v** file contains many tasks and functions to configure the components of the testbench. This section provides a description of some important Verilog HDL tasks. The VHDL functions are generally similar in structure and are not specifically listed below.

2.4.2.1. Avalon-MM Tasks

These tasks read and write the configuration and MDIO registers using the Avalon-MM protocol.

For a more information about the Avalon-MM protocol refer to the *Avalon Interface Specifications*.

write_avalon

This command creates an Avalon-MM write command. You can use it to update registers.

write_avalon		
Usage	`avalon_write(data_to_write, address_to_write)`	
Arguments	`data_to_write`	The data to write at the specified `address_to_write`.
	`address_to_write`	The address of the register to write.

read_avalon

This command creates an Avalon Memory-Mapped (Avalon-MM) read command. You can use it to read register values.

read_avalon		
Usage	`avalon_read(address_to_read, data_to_read)`	
Arguments	`address_to_read`	The address of the register to read.
	`data_to_read`	The data to read from the specified address.

write_mdio

You can use this command to write the MDIO registers.

write_mdio		
Usage	`write_mdio(clause45, prtadr, devphyadr, regadr, data)`	
Arguments	`clause45`	Set to 1 if the MDIO is for clause 45; set to 0 if the MDIO is for clause 22.
	`prtadr`	Clause 45 PHY port address.
	`devphyadr`	Clause 45 address of device (clause 45) or PHY (clause 22).
	`regadr`	MDIO register address. It uses16 bits for clause 45 and the 5 low-order bits for clause 22.
	`data`	Write data for the MDIO register.

read_mdio

You can use this command to read the MDIO registers.

read_mdio		
Usage	`read_mdio(clause45, prtadr, devphyadr, regadr, data)`	
Arguments	`clause45`	Set to 1 if the MDIO is for clause 45; set to 0 if the MDIO is for clause 22.
	`prtadr`	Clause 45 PHY port address.
	`devphyadr`	Clause 45 address of device (clause 45) or PHY (clause 22).
	`regadr`	MDIO register address. It uses16 bits for clause 45 and the 5 low-order bits for clause 22.
	`data_read`	Data read from the MDIO register.

2.4.2.2. Avalon-ST Ethernet Frame Generation Tasks

You can use the following tasks to create ethernet frames to be sent on the Avalon-ST interface.

For a more information about the Avalon-ST protocol refer to the *Avalon Interface Specifications*.

gen_valid_frame

Creates a valid frame for transfer using the Avalon-ST protocol.

gen_valid_frame		
Usage	`gen_valid_frame(pkt_length, pkt_type, min_ipg)`	
Arguments	`pkt_length`	The length of the payload in bytes.
	`pkt_type`	Packet type. This the data that you entered in the length and type field of the frame.
	`min_ipg`	The minimum inter-packet gap (IPG) bytes to be generated.

gen_idles

Creates idles cycles on the Tx interface.

gen_idles		
Usage	`gen_idles (idle_length)`	
Arguments	`idle_length`	Number of idle cycles to be transmitted on the MAC Tx interface.

2.4.2.3. Ethernet Frame Generation Tasks

You can use the following tasks to generate Ethernet frames which are sent over the Avalon-ST interface.

gen_valid_frame

Creates a valid frame for transfer using the Avalon-ST protocol.

gen_valid_frame		
Usage	`gen_valid_frame(pkt_length, pkt_type, min_ipg)`	
Arguments	`pkt_length`	The physical size, which includes the payload only and must be greater than 4.
	`pkt_type`	Packet type.This is the 2 octet type of length.
	`min_ipg`	The16-bit vector packet-inter packet gap, which you must specify. The minimum value is 8 bytes

gen_crc_errored_frame

Creates a valid frame for transfer using the Avalon-ST protocol.

gen_crc_errored_frame		
Usage	`gen_crc_errored_frame(pkt_length,pkt_type, min_ipg)`	
Arguments	`pkt_length`	The physical size, which includes the payload only and must be greater than 4.
	`pkt_type`	Packet type.This is the 2 octet type of length.
	`min_ipg`	The16-bit vector packet-inter packet gap, which you must specify. The minimum value is 8 bytes

2.4.3. Running Tests

You can use the **tb.v** as sample test to perform preliminary verification of the IP core. You can extend this example to create other tests to create a complete verification suite.

2.4.3.1. Typical Test Sequence

A typical test case performs the following operations after a simulated power-on reset:

1. Initialize the 10-Gbps Ethernet IP core, which consists the following operations:
   a. Set the operation mode in the `command_config` register.
   b. Set the address via the `mac_0` and `mac_1` registers.
   c. Set the IPG for transmit frames via the `tx_ipg_length` register.
   d. Set the Avalon-ST FIFO threshold registers.
   e. Set the supplemental unicast addresses.
2. Start transmission and clear the Rx and Tx FIFOs.

2.4.3.2. Running sample test case

Before you can run the example tests, you must compile several RTL files and libraries. A script in the **/tb** directory performs these tasks. To run the script, complete the following tasks:

1. Change to the `<project_dir>`**/tge_91/tb/verilog** directory.
2. Launch ModelSim software.
3. Type the following command at the command prompt:

   `source demo_run_modelsim.tcl` ↵
4. Upon successful completion, the following text appears in the ModelSim transcript:

Example 2–1. Transcript from Successful Run in ModelSim

```
# DESC: Demonstration testbench

# ***********************************************************
Time: 45 ns  Instance: tb.DUT.nl1ii0ii.pll2
# 160005 Info: Avalon MM write at addr_hex=c, data_hex=34221400,  time=160005
# 180005 Info: Avalon MM write at addr_hex=10, data_hex=80007bab,  time=180005
# 200005 Info: Avalon MM write at addr_hex=300, data_hex=fe453681,  time=200005
# 220005 Info: Avalon MM write at addr_hex=304, data_hex=8000c43c,  time=220005
# 240005 Info: Avalon MM write at addr_hex=5c, data_hex=c,  time=240005
# 260005 Info: Avalon MM write at addr_hex=1c, data_hex=3c0,  time=260005
# 280005 Info: Avalon MM write at addr_hex=20, data_hex=40,  time=280005
# 300005 Info: Avalon MM write at addr_hex=24, data_hex=3c0,  time=300005
# 320005 Info: Avalon MM write at addr_hex=28, data_hex=40,  time=320005
# 340005 Info: Avalon MM write at addr_hex=2c, data_hex=3c0,  time=340005
# 360005 Info: Avalon MM write at addr_hex=30, data_hex=40,  time=360005
# 380005 Info: Avalon MM write at addr_hex=34, data_hex=3c0,  time=380005
# 400005 Info: Avalon MM write at addr_hex=38, data_hex=40,  time=400005
# 420005 Info: Avalon MM write at addr_hex=8, data_hex=80000203,  time=420005
# 742759 eth_gen generating packet pkt_type   100 -- pkt_size        100
# 851559 eth_gen generating packet pkt_type   101 -- pkt_size        101
# 963559 eth_gen generating packet pkt_type   102 -- pkt_size        102
# 1327247 avl_st_checker : received packet
# 1423247 avl_st_checker : received packet
# 1538447 avl_st_checker : received packet
```

```
#
#          Testbench tb.custdemo elapsed time 1000 ns

# 1075559 eth_gen generating packet pkt_type   103 -- pkt_size        103
# 1128847 avl_st_checker : received packet
# 1187559 eth_gen generating packet pkt_type   104 -- pkt_size        104
# 1231247 avl_st_checker : received packet
Testbench tb.custdemo elapsed time 2000 ns
#          Testbench tb.custdemo elapsed time 3000 ns

# 3538447 avl_st_gen generating packet pkt_type   100 -- pkt_size        100
# 3634447 avl_st_gen generating packet pkt_type   101 -- pkt_size        101
# 3730447 avl_st_gen generating packet pkt_type   102 -- pkt_size        102
# 3826447 avl_st_gen generating packet pkt_type   103 -- pkt_size        103
# 3922447 avl_st_gen generating packet pkt_type   104 -- pkt_size        104

#          Testbench tb.custdemo elapsed time 4000 ns
#          Testbench tb.custdemo elapsed time 5000 ns

# DONECHECK #1: tb.custdemo, time: 5656 ns
# ************************************************************
# $$$ End of testbench tb.custdemo at :    5666359
# chk_cnt = 1, exp_chk_cnt = 1
# err_cnt = 0, exp_err_cnt = 0
# $$$ Exit status for testbench tb.custdemo : TESTBENCH_STATUS: COMPLETED PASSED
```

2.5. Implementation and Timing Analysis

When you generate a custom 10-Gbps Ethernet IP core, the Quartus II software also generates files for timing constraints and place and route. When you instantiate your Ethernet IP core in a complete system design, you can use these scripts as a guide when creating the timing constraints for your complete system.

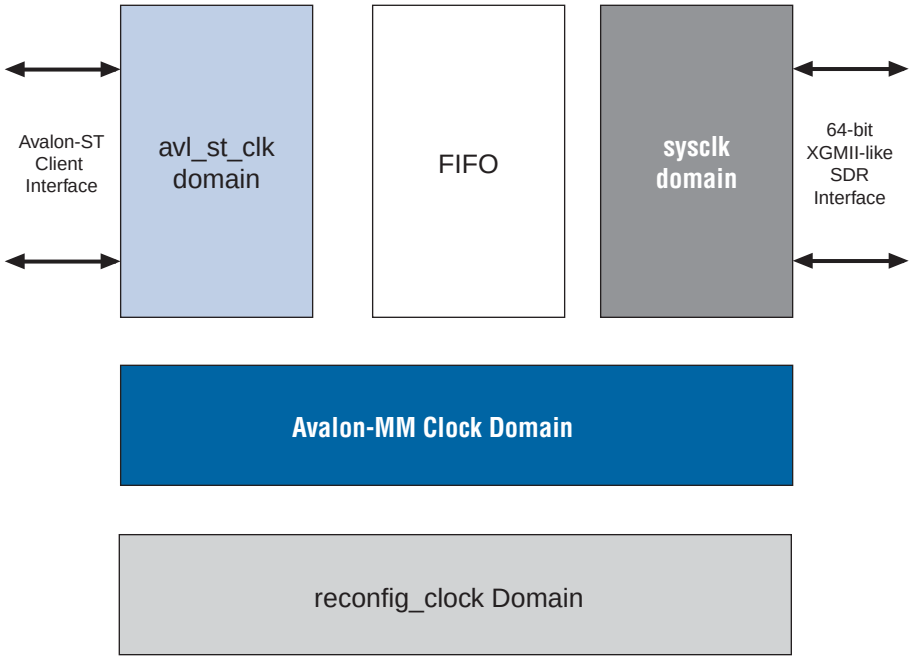
You can use the Tcl interface to add constraints to your design. In particular, the constraints in the **tge_91_contraints.tcl** file prevent shift registers from converting to memory, which is important if your design includes the optional ECC functionality. To add these constraints to your design, type the following command in the Quartus II Tcl Console window:

```
source tge_91_contraints.tcl
```

2.5.1. Clock Domains

The 10-Gbps Ethernet IP core includes at least three clock domains. There is a fourth clock domain if your design includes the ALTGX_RECONFIG megafunction discussed in "Transceiver Reconfiguration Options" on page 2–7. These clock domains are shown in Figure 2–7. The design logic synchronizes signals that cross clock boundaries. This logic is not shown in Figure 2–7.

Figure 2–7. Clock Domains for the 10-Gbps Ethernet IP Core



2.5.2. Reset Synchronization

The IP core provides a separate reset pin for each clock domain. The design uses the following reset and clock pairs:

- `reset_n`—resets the `sysclk` clock domain. `sysclk` connects to the MAC Tx and Rx and the MAC side of FIFO interfaces.
- `avl_st_reset_n`—resets the `avl_st_clk` clock domain. The clock domain is for the client side of the FIFO or MAC logic.
- `avalon_reset_n`—resets the `avalon_clk` domain.

The reset pins are active low and must be synchronized by their respective clocks before use.

2.5.3. Synopsis Design Constraints

The timing constraints are provided by the Synopsis Design Constraints File (**.sdc**). In this example, **tge_91.sdc** defines clocks and timing exceptions. The file specifies the following information:

- All clocks generated from PLLs
- Timing uncertainty from clock sources
- All clocks that are not generated from PLLs
- All asynchronous clock groups

Example 2–2 shows the constraints included in the **.sdc** for the 10-Gbps Ethernet IP core.

Example 2–2. Timing Constraints for the 10-Gbps Ethernet IP Core

```
// Create clocks other than generated clocks
create_clock -period 6.4 -name {sysclk} [get_ports *sysclk]
create_clock -period 10 -name {avalon_clk} [get_ports *avalon_clk]
create_clock -period 6.4 -name {avl_st_clk} [get_ports *avl_st_clk]
create_clock -period 20 -name {reconfig_clk} [get_ports *reconfig_clk]

// Create asynchronous clock groups
set_clock_groups -asynchronous -group {sysclk} –group {avalon_clk }
set_clock_groups -asynchronous -group {avl_st_clk} –group {avalon_clk}
set_clock_groups -asynchronous -group { avalon_clk } -group
{*eth_10g_inst|xaui_10g__genblk.xaui_10g|xaui_10g_serdes4_reconfig__genblk.xaui_10g_se
rdes4_reconfig|xaui_10g_serdes4_reconfig_*|central_clk_div0|coreclkout }
set_clock_groups -asynchronous -group { avl_st_clk } -group {
*eth_10g_inst|xaui_10g__genblk.xaui_10g|xaui_10g_serdes4_reconfig__genblk.xaui_10g_ser
des4_reconfig|xaui_10g_serdes4_reconfig_*|central_clk_div0|coreclkout }
```

Refer to "Clocks and Reset " on page 3–57 for more information about clocking.

2.5.4. Placement Constraints

In this example, some global clocks and synthesis options are specified in **tge_constraints.tcl**. Example 2–3 shows typical pin type and placement assignments.

Example 2–3. Pin Assignments in the .qsf File

```
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" –to MYPIN
set_location_assignment PIN_G14 -to MYPIN
```

Refer to the appropriate Device Handbook, Pin-Out Files for Altera Devices, and Device Pin Connection Guidelines web pages for the targeted Altera device to determine available I/O standards and pin assignments for your design.

2.5.5. Timing Verification

Upon successful compilation, the Quartus II software generates output files that specify important features of your design. Table 2–7 lists the most important output files. All of the output files are described in *<variation_name>*.**html**.

Table 2–7. Quartus II Output Files

File Name	Description
<variation_name>.**pin**	Lists the pin location assignments in the final design.
<variation_name>.**fit.rpt**	Contains information on logic utilization, resource utilization, timing models, and other useful information.
<variation_name>.**sta.rpt**	Provides the timing analysis report and the results of the timing analysis including timing path slack.

3. 10-Gbps Ethernet IP Functional Description

This chapter provides a detailed description of Altera's 10-Gbps Ethernet IP core. It begins with a high-level overview of typical 10-Gbps Ethernet systems and then provides detailed descriptions of the MAC, transmit (Tx) and receive (Rx) datapaths, ECC, software programming model, register descriptions, PHY, clock domains, and reset. It includes the following sections:

- Typical 10-Gbps Ethernet Systems
- MAC Functional Description
- ECC Options
- Software Programming Interface
- Register Descriptions
- 10-Gbps Ethernet PHY
- Clocks and Reset

Altera categorizes this IP core as a reference design, described on the Altera website on the 10-Gbps Ethernet Reference Design web page.

3.1. Typical 10-Gbps Ethernet Systems

This section provides top-level block diagrams of all of the variants that you can create when you to customize your 10-Gbps Ethernet IP core.

Figure 3–1 illustrates a system with a MAC connecting an external 10-Gbps Ethernet device over XAUI.

Figure 3–1. MAC Connecting to External Ethernet Device Over XAUI

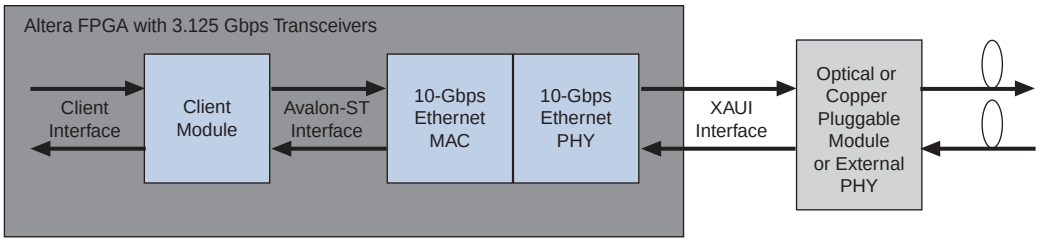
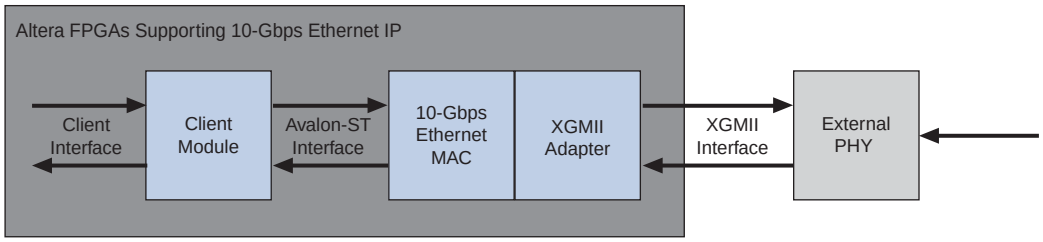


Figure 3–2 illustrates a system with a MAC connecting an external 10-Gbps Ethernet PHY over XGMII.

Figure 3–2. MAC Connection to External PHY Using XGMII



3.2. MAC Functional Description

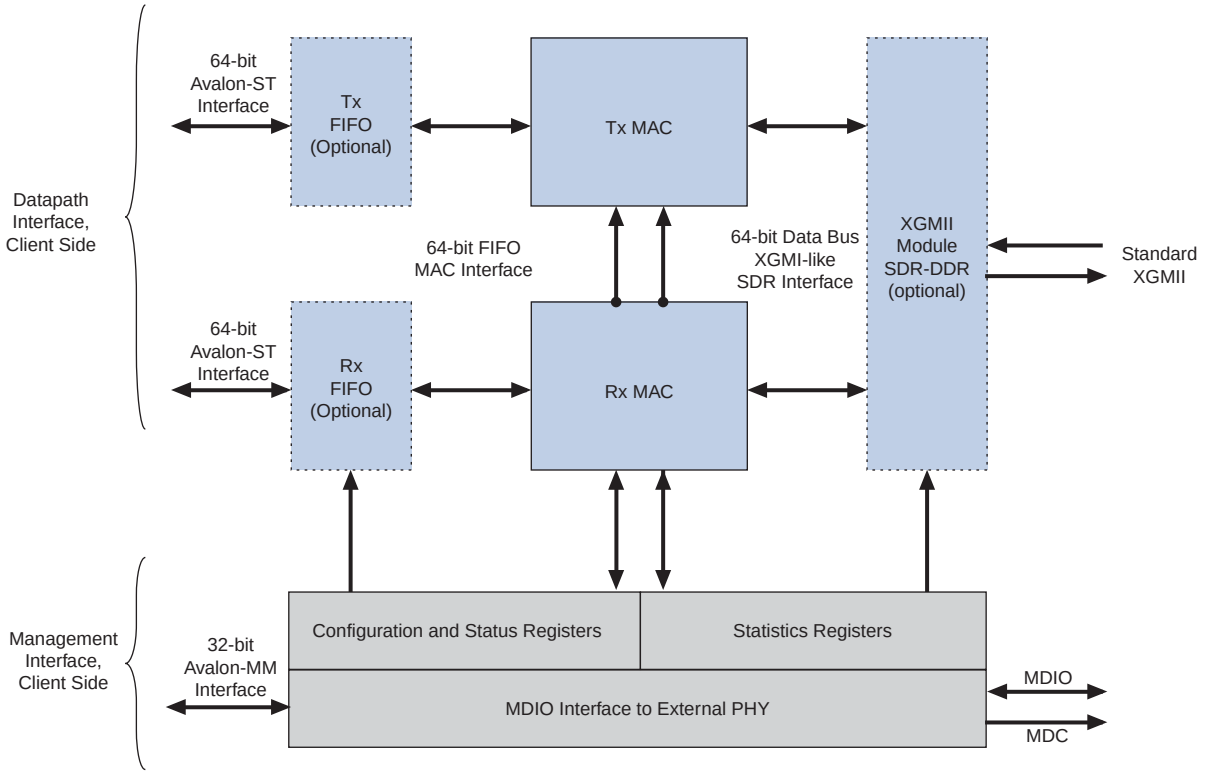
The Altera 10-Gbps Ethernet MAC implements the 10-Gbps Ethernet MAC in accordance with the IEEE 802.3 2005 specification. This module handles the flow of data between a client and Ethernet network via a 10-Gbps Ethernet PHY. In the Tx direction, the MAC accepts client frames, inserts an interpacket gap (IPG), preamble, start of frame delimiter (SFD), headers, and checksums before passing them to the PHY. The PHY encodes the MAC frame as required for reliable transmission over the media to the remote end. Similarly, in the receive direction, the MAC accepts frames via a PHY, performs checks, generates statistics, strips out the preamble and SFD, and passes the rest of the frame to the client.

The MAC includes the following interfaces:

- Data path interfaces
  - Client side–Avalon-ST 64-bits wide
  - PHY side–XAUI or XGMII
- Management interface
- Avalon-MM host slave interface

The Avalon-MM interface provides host access to the MAC and PHY configuration, status, MAC, statistics counters registers, and external PHY device registers via the MDIO interface. Figure 3–3 illustrates the top-level modules of the MAC. As Figure 3–3 illustrates, the MAC can be connected to the client directly or using an optional FIFO.

Figure 3–3. High-Level Block Diagram of the 10-Gbps Ethernet MAC

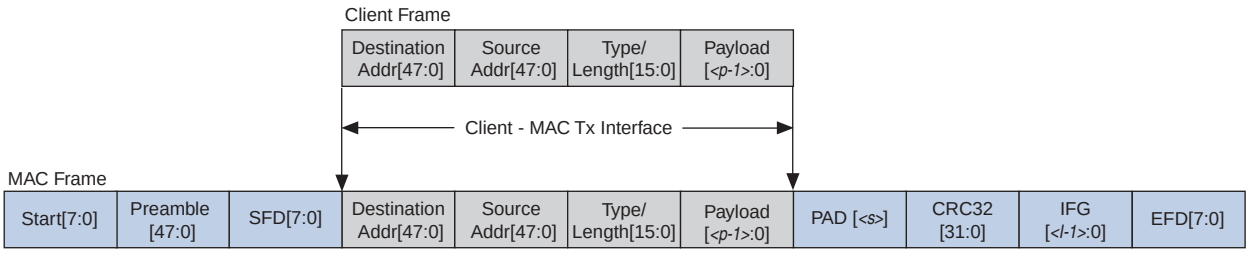


3.2.1. Transmit Datapath

The Tx MAC module receives the client payload data with the destination and source addresses and then adds, appends, or updates various header fields in accordance with the configuration specified. The MAC does not modify the destination address or the payload received from client. However, the Tx MAC module adds a preamble, pads the payload to satisfy the minimum Ethernet frame payload of 64 bytes, and calculates the CRC over the entire MAC frame. (If padding is added, it is also included in CRC calculation.) The Tx MAC module can also modify the source address, and insert interpacket gap (IPG) bytes when necessary.

Figure 3–4 illustrates the changes that the Tx MAC makes to the client frame.

Figure 3–4. Typical Client Frame at the XGMII (Tx) Interface



Notes to Figure 3–4:

(1) <*p*> = payload size = 0–1500 bytes, or 9600 bytes for jumbo frames.

(2) <*s*> = padding bytes = 0–46 bytes

(3) <*I*> = number of IPG bytes

The following sections describe the functions that the Tx module performs.

3.2.1.1. Start, Preamble, and SFD Insertion

In the Tx datapath the MAC appends a one-byte START, 6-byte preamble, and 1-byte SFD to the client frame. (This MAC module also incorporates the functions of the reconciliation sublayer.)

3.2.1.2. Address Insertion

The client provides the destination MAC address and the source address of the local MAC. However, if enabled by a Command_Config Register bit, the source MAC address can be replaced by the primary address contained in two, 32-bit MAC registers: `mac_0` and `mac_1`.

3.2.1.3. Length/Type Field Processing

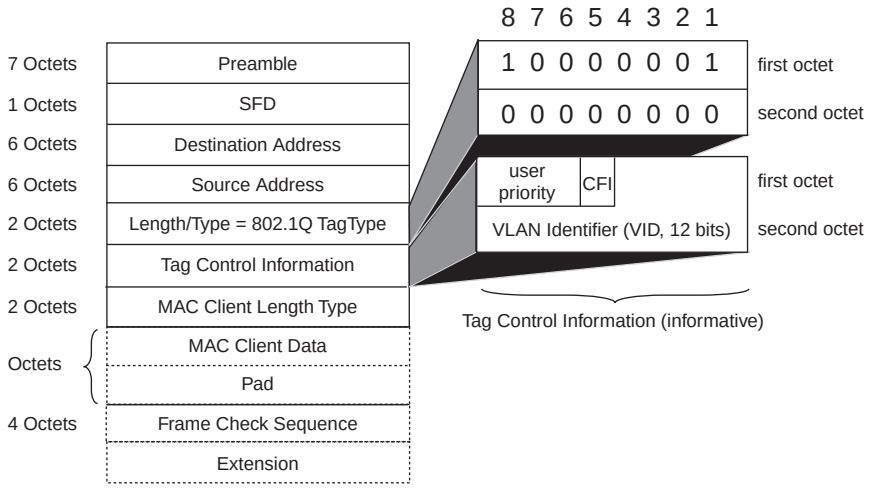
This two-byte header represents either the length of the payload or the type of MAC frame. When the value of this field is equal to or greater than 1536 (0x600) it indicates a type field. Otherwise, this field provides the length of the payload data that ranges from 0–1500 bytes. The Tx MAC does not modify this field and forwards it to the network.

3.2.1.4. VLAN and Stacked VLAN Frames Processing

The extension of a basic frame is a VLAN tagged frame, which contains an additional VLAN tag field between the source address and length/type fields as Figure 3–5 illustrates. VLAN tagging is defined by the IEEE 802.1Q standard. VLANs can identify and separate many groups' network traffic in enterprises and metro networks.VLAN tagged frames have a maximum length of 1522 bytes, excluding the preamble and the SFD bytes. In carrier Ethernet network applications based on IEEE 802.1ad provider bridge standard (QinQ) for scaling the network, frames can be tagged with two consecutive VLAN tags (stacked VLAN). Stacked VLAN frames

contain an additional 8-byte field between the source address and length/type fields. The Tx MAC forwards the VLAN or stacked VLAN tag and fields to the PHY as it would for other payload octets. Recently, the IEEE 802.1ah standard for Provider Backbone Bridge (PBB) and PBB traffic engineering (TE) or MAC-in-MAC is gaining traction which still can use this VLAN stacking for inner and outer VLAN tags.

Figure 3–5. MAC Frame Showing VLAN Fields



3.2.1.5. Frame Padding

When the length of client frame is less than 64 bytes (meaning the payload is less than 46 bytes), the Tx MAC module inserts pad bytes (0x00) after the payload to create a frame length equal to the minimum size of 64 bytes.

3.2.1.6. Frame Check Sequence (CRC-32) Insertion

The Tx MAC computes and inserts CRC32 checksum in the transmitted MAC frame. The frame check sequence (FCS) field contains a 32-bit CRC value. The MAC computes the CRC32 over the frame bytes that include the source address, destination address, length, data, and pad. The CRC checksum computation excludes the preamble, SFD, FCS, and extension. The encoding is defined by the following generating polynomial:

$$FCS(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X^{1} + 1$$

CRC bits are transmitted with MSB (X^{32}) first. (Refer to "Byte Order on the Avalon-ST Interface Lanes" on page 3–14 and "Octet Transmission on the Avalon-ST SIgnals" on page 3–14 for illustrations of byte ordering.

3.2.1.7. Interpacket Gap Generation and Insertion

The Tx MAC maintains the minimum interpacket gap (IPG) between transmitted frames required by the IEEE 802.3 Ethernet standard. The default minimum IPG minimum is maintained at 96 bit times (or 12 byte times). However, you can change the default IPG (in bytes) via the configuration register `tx_ipg_length`. You can configure the minimum IPG to any value between 8 bytes and 252 bytes times in the `tx_ipg_length` register.

The deficit idle counter maintains an average IPG. You can configure the average IPG value in the `tx_ipg_length` register (`0x05c`). The number is a multiple of 4 with a minimum of 8 and a maximum of 252. The default (IEEE required value) is 12.

To guarantee reliable PCS functionality, Altera recommends that you set the IPG to a minimum of 12 bytes. The IPG between successive frames varies and is the minimum that you specify ±3 bytes.

3.2.2. Tx Interfaces

This section describes the following Tx interfaces:

- "Tx Client Side Interfaces " on page 3–6
- "Tx FIFO Client Interface " on page 3–6
- "FIFO MAC Interface " on page 3–9
- "SDR XGMII Tx Interface " on page 3–12
- "Standard DDR XGMII Interface" on page 3–11

3.2.2.1. Tx Client Side Interfaces

The client side of the interface depends on whether you included a FIFO in your design. If a FIFO is included, the client connects to the FIFO's client side interface, otherwise the client connects directly to the MAC's client interface.

The client Tx datapath employs the Avalon Streaming (Avalon-ST) protocol. The Avalon-ST protocol is a synchronous point-to-point, unidirectional interface that connects the producer of a data stream (source) to a consumer of data (sink). The key properties of this interface include:

- Frame transfers marked by `startofpacket` and `endofpacket` signals.
- Signals from source to sink are qualified by the `valid` signal.
- Errors marking a current packet, are aligned with `endofpacket` cycle.
- Use of the `ready` signal by the sink backpressure the source. The source must respond to the `ready` signal from sink by deasserting the `valid` signal after a fixed number of cycles defined by the `ready_latency`.

The client acts as a source and the Tx FIFO/MAC acts as a sink in the transmit direction. If your design does not include the optional FIFO, the MAC does not buffer client data and it must receive data continuously between the assertions of the `startofpacket` and `endofpacket` signals.

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

3.2.2.2. Tx FIFO Client Interface

This is an Avalon-ST interface that provides backpressure from the MAC Tx sink to the client via Tx FIFO. The Tx FIFO receives data from the client and buffers it until the MAC is ready to consume it. Figure 3–6 illustrates this interface.

Figure 3–6. Client–FIFO Interface, Tx Datapath

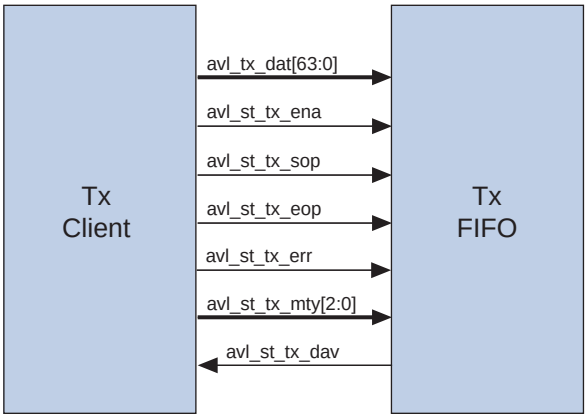


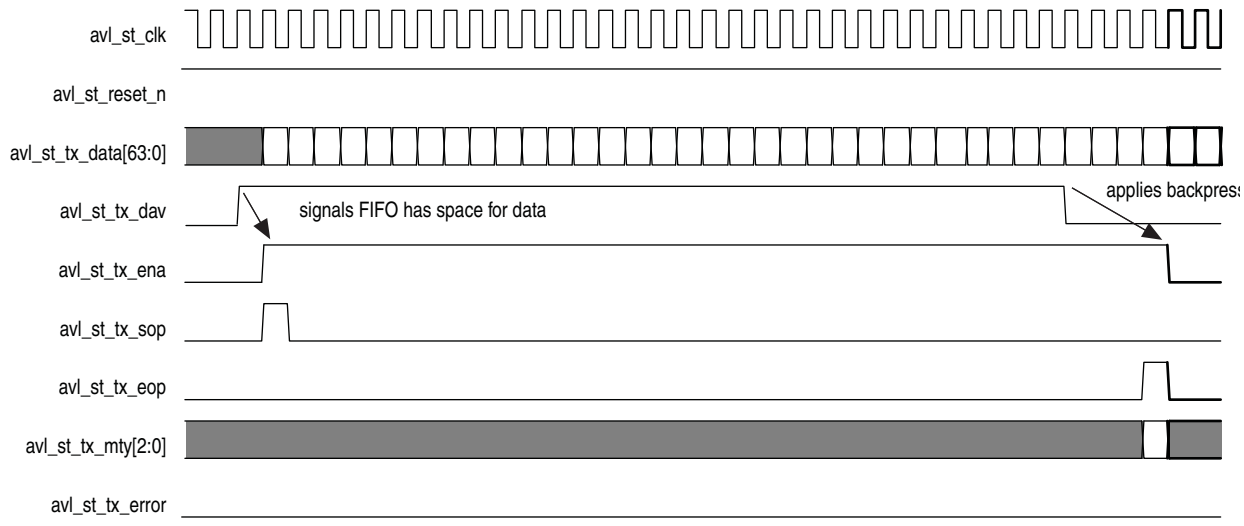
Table 3–1 describes the signals that comprise this interface.

Table 3–1. FIFO Client Interface Signals–AnAvalon-ST Interface

Signal Name	Dir	Description
`avl_st_tx_dat[63:0]`	I	64-bit client (source) data.
`avl_st_tx_ena`	I	When asserted, Indicates that `avl_st_tx_dat`, `avl_st_tx_sop`, `avl_st_tx_eop`, and `avl_st_tx_mty` are valid.
`avl_st_tx_sop`	I	Asserted for one cycle to indicate the start of client/source data.
`avl_st_tx_eop`	I	Asserted for one cycle to indicate the end of client/source data.
`avl_st_tx_mty[2:0]`	I	Specifies how many bytes of `avl_st_tx_dat[63:0]` are empty when `avl_st_tx_eop` is asserted as follows: Value — Valid Data Bits 0 — `avl_tx_dat[63:0]` 1 — `avl_tx_dat[63:8]` 2 — `avl_tx_dat[63:16]` 3 — `avl_tx_dat[63:24]` 4 — `avl_tx_dat[63:32]` 5 — `avl_tx_dat[63:40]` 6 — `avl_tx_dat[63:48]` 7 — `avl_tx_dat[63:56]`
`avl_st_tx_dav`	I	Indicates that space is available in the FIFO and it is ready to consume client data. This signal acts as the `ready` signal of the Avalon-ST interface. When this signal is asserted, the client can continue to provide new data on the `avl_st_tx_dat` with `avl_st_tx_data_ena` asserted. To backpressure to the client (source), the Tx MAC deasserts the `avl_st_tx_dav` signal.
`avl_st_tx_err`	O	Marks the current client packet as errored. This signal must be asserted by the client when it asserts the `avl_st_tx_eop`.

Figure 3–7 shows the timing for the MAC/client interface when you include the optional FIFO.

Figure 3–7. MAC/Client Interface with Tx FIFO



3.2.2.3. Tx FIFO Modes

The Tx FIFO can be configured to operate in the following two modes:

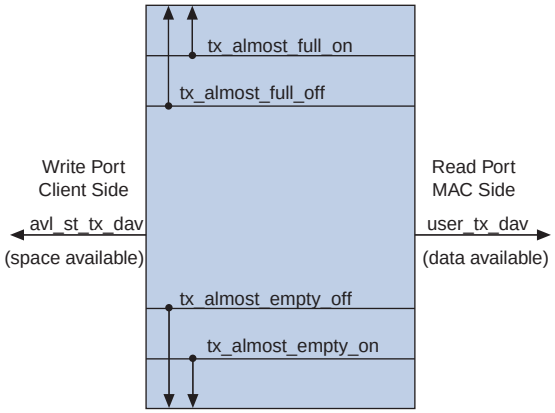
- Store and forward—In this mode, the entire client frame is stored before it is made available to the MAC Tx module. In the store and forward mode, you should specify a FIFO size that can hold the longest possible frame in the system. Altera recommends twice the maximum possible frame size.
- Fill level—In the fill level mode, the data is available for the Tx MAC when a threshold is reached or an end of packet is signal is asserted within the specified threshold. The data is must be passed continuously from the write side to the read side.

Note that when the design includes the Tx FIFO as a buffer, the client may assert `avl_st_tx_ena` between the `avl_st_tx_sop` and `avl_st_tx_eop` signals. However, the Tx FIFO must provide a continuous data stream to the Tx MAC.

3.2.2.4. Tx FIFO Back Pressure and Configurable Thresholds

The Tx FIFO provides four registers to dynamically configurable thresholds to effectively manage potential overflow and underflow conditions and increase datapath efficiency. The Tx FIFO ready signal, `avl_st_tx_dav`, controls the amount of client data written to the Tx FIFO and is used to apply backpressure. Similarly the `user_tx_dav` signal controls the amount of client data available to the Tx MAC. The FIFO is 64 bits wide. All the threshold values are specified in bytes.

Figure 3–8. FIFO thresholds in Tx FIFO



Notes to Figure 3–8:

(1) For the Rx FIFO, the write port connects to the MAC and the read port connects to the client.

Table 3–2 explains these thresholds.

Table 3–2. FIFO Thresholds

Threshold in Bytes	Description
`tx_almost_full_on`	This threshold represents the space available for write data in the FIFO. When almost full, the `avl_st_tx_dav` signal is deasserted, indicating that the client should stop sending data to FIFO.
`tx_almost_full_off`	This threshold represents the space that must be available in the FIFO before client data can be written in the FIFO. When this condition occurs, `avl_st_tx_dav` is asserted enabling the client transmission. The `tx_almost_full_off` must be >= `tx_almost_full_on`.
`tx_almost_empty_on`	This threshold represents the number of bytes available for read before the FIFO is empty. When this condition occurs, the `avl_tx_dav` signal is deasserted indicating no data is available for the Tx MAC.
`tx_almost_empty_off`	This represents the amount of data that must be available in the FIFO before reads can resume. When this condition occurs `avl_tx_dav` is asserted indicating the availability of data for read. `tx_almost_empty_off` must be >= `tx_almost_empty_on`.

3.2.2.5. FIFO MAC Interface

When the client connects to the FIFO, the Tx MAC reads client data from FIFO read port, updates the data as required, and sends the MAC frame to the PHY interface. The FIFO provides a continuous stream of bytes to the MAC between the `avl_st_tx_sop` and `avl_st_tx_eop` signals. The Tx MAC can apply backpressure using the `avl_tx_read` signal indicating when it is ready to receive data. If `avl_tx_read` is deasserted, valid data may continue for one cycle.

☞ If your design does not include the FIFO, the client should mimic the FIFO MAC interface.

Figure 3–8 illustrates the FIFO MAC Tx interface.

Figure 3–9. FIFO MAC Tx Interface

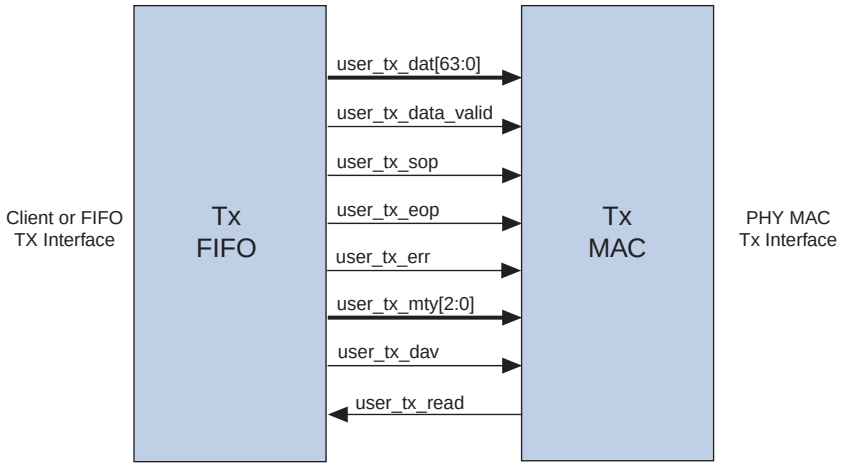


Table 3–3 describes the signals that comprise this interface.

Table 3–3. FIFO or Client MAC Interface Signals (Part 1 of 2)

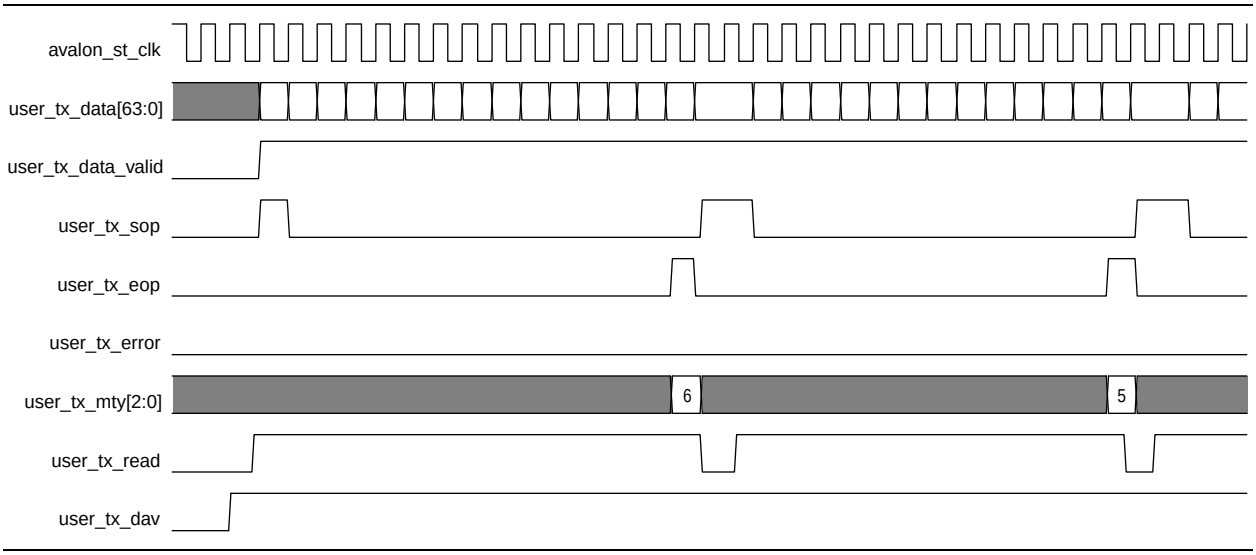
Signal Name	Dir	Description
`user_tx_dat[63:0]`	I	64-bit client or source data.
`user_tx_data_valid`	I	When asserted, Indicates that the rest of the MAC inputs are valid.
`user_tx_sop`	I	Asserted for _one cycle to indicate the start of client/source data.
`user_tx_eop`	I	Asserted for one cycle to indicate the end of client/source data.
`user_tx_mty[2:0]`	I	Specifies how many bytes of `user_tx_dat[63:0]` are empty when `user_tx_eop` is asserted as follows: Value — Valid Data Bits 0 — `user_tx_dat[63:0]` 1 — `user_tx_dat[63:8]` 2 — `user_tx_dat[63:16]` 3 — `user_tx_dat[63:24]` 4 — `user_tx_dat[63:32]` 5 — `user_tx_dat[63:40]` 6 — `user_tx_dat[63:48]` 7 — `user_tx_dat[63:56]`
`user_tx_dav`	I	Indicates that the source has data available for the MAC's consumption. This signal should only be asserted when the source has an entire frame for the MAC or is guaranteed to send the entire packet once the MAC starts requesting data. If your variant includes the optional FIFO, the FIFO drives `user_tx_dav`, otherwise, the client interface drives `user_tx_dav`.

Table 3–3. FIFO or Client MAC Interface Signals (Part 2 of 2)

Signal Name	Dir	Description
`user_tx_err`	I	Marks the current client packet as errored. This signal must be asserted by the client when it asserts the `user_tx_eop`.
`user_tx_read`	O	This signal indicates that the Tx MAC is ready to accept data. It is used to apply backpressure. This signal is deasserted by the MAC when it is inserting CRC or IPG bytes. The client must stop sending data one clock cycle after this signal is deasserted. When this signal is asserted, the client may drive data to the MAC on the next clock cycle. Refer to Figure 3–10.

Figure 3–10 illustrates the timing for the client or FIFO Tx interface.

Figure 3–10. Client or FIFO MAC Tx Interface Timing



3.2.3. MAC – PHY Tx Interface

The PHY side of MAC interfaces implements the XGMII protocol as defined by IEEE 802.3 2005 standard. The standard XGMII implementation consists of 4-lane (32-bit) wide data bus transitioning at both edges of a 156.25 MHz clock. However, the Altera IP uses a single data rate (SDR) version of this interface, when connecting a MAC to an internal PHY.

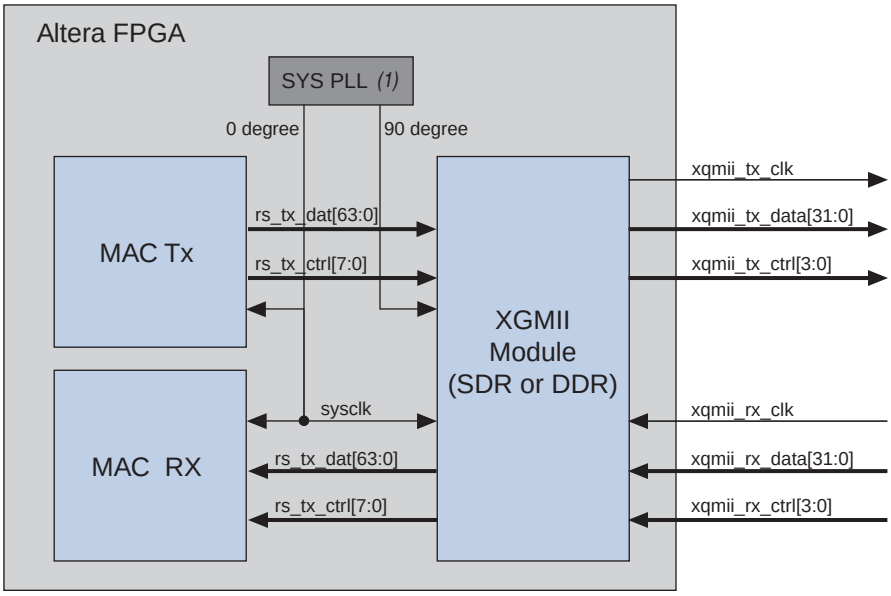
When you parameterize the 10-Gbps Ethernet IP core to include both Altera MAC and integrated PHY, the MAC-PHY Tx interface is an SDR version of XGMII. When you configure the IP core to connect the Altera MAC to an external PHY, the standard DDR XGMII interface is generated.

In this document, both SDR and DDR interfaces are generally referred to as XGMII interfaces.

3.2.3.1. Standard DDR XGMII Interface

The DDR XGMII interface consists of a 32-bit data, 4-bit control bus, and a source synchronous clock. Figure 3–11 illustrates this interface.

Figure 3–11. Tx and Rx XGMII Interface



Notes to Figure 3–11:

(1) The SYS PLL is not part of the 10-Gbps Ethernet IP core.

(2) The `sysclk` and the 90° phase shifted clock should come from the same PLL to ensure 0 ppm difference.

Table 3–4 describes the signals that comprise this interface.

Table 3–4. Tx XGMII Interface

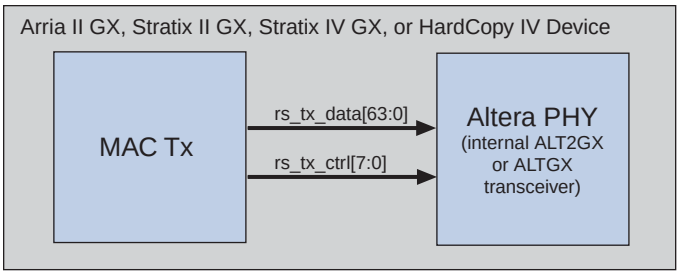
Signal Name	Dir	Description
`xgmii_tx_clk`	O	Clock with an additional output that has a 90° phase lag with respect to `xgmii_tx_data` and `xgmii_tx_ctrl` signals.
`xgmii_tx_data[31:0]`	O	4-lane data bus carrying bytes[7:0] of MAC Tx module changing value on both edges of `xgmii_tx_clk`.
`xgmii_tx_ctrl[3:0]`	O	4-bit signal that indicates when a control octet is present on the corresponding `xgmii_tx_data` lane.

3.2.3.2. SDR XGMII Tx Interface

This interface consists of a 64-bit data bus and 8-bit control bus operating at 156.25 MHz. The data bus carries the MAC frame; the most significant byte occupies the least significant lane.

Figure 3–12 illustrates the SDR XGMII interface for the Tx datapath.

Figure 3–12. SDR XGMII Interface



Note to Figure 3–12:

(1) The SDR interface is synchronous to the Tx PHY clock.

Table 3–5 describes the signals that comprise the SDR XGMII Tx interface.

Table 3–5. Tx XGMII SDR interface

Signal Name	Dir	Description
`rs_tx_data[63:0]`	O	8-lane data bus carrying bytes[7:0] of the MAC Tx module
`rs_tx_ctrl[7:0]`	O	8-bit signal where each bit indicates when a control octet is present on the corresponding `rs_tx_data` byte lane.

3.2.3.3. Error Conditions on Tx Datapath

During frame transmission, if an error condition is detected, the MAC sends |E| control characters on the data lanes to signal an error to the PHY. The MAC restarts the frame when the client sends a new frame with a start of frame indication.

The only error condition defined for the Tx datapath is FIFO underflow. (The FIFO can be read when it does not have data, which may occur when the MAC tries to prefetch data.) However, an error occurs if the FIFO deasserts the valid signal before the end of a packet. In this case, the MAC suffers from data starvation; there is no buffer in the MAC to compensate. Because the IEEE 802.3 standard requires continuos transmission, the packet is terminated with an |E|. The remainder of the packet is read when available and data is ignored, until the next start of packet.

3.2.3.4. Order of Transmission

Bytes are transmitted starting with the preamble and ending with the FCS in accordance with the IEEE 802.3 standard. All individual header bytes are transmitted LSB first except the length/type, VLAN tag, VLAN info, and pause quanta, which are transmitted MSB first. The data bus is 64-bits wide; the first byte of the destination address is carried on bits [63:56]. The FCS is transmitted with the CRC[32] bit first. The address arrives in the same order as it was received. The client side of Tx interface bus is big endian. Table 3–6 shows the fields of a Tx packet.

Table 3–6. Byte Order on the Avalon-ST Interface Lanes

	Destination Address (DA) *(Note 1)*						Source Address (SA)						Type/ Length		Data (D)		
Octet	**5**	**4**	**3**	**2**	**1**	**0**	**5**	**4**	**3**	**2**	**1**	**0**	**1**	**0**	**00**	**...**	**NN**
Bit	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]	[15:8]	[7:0]	MSB[7:0]	...	LSB[7:0]

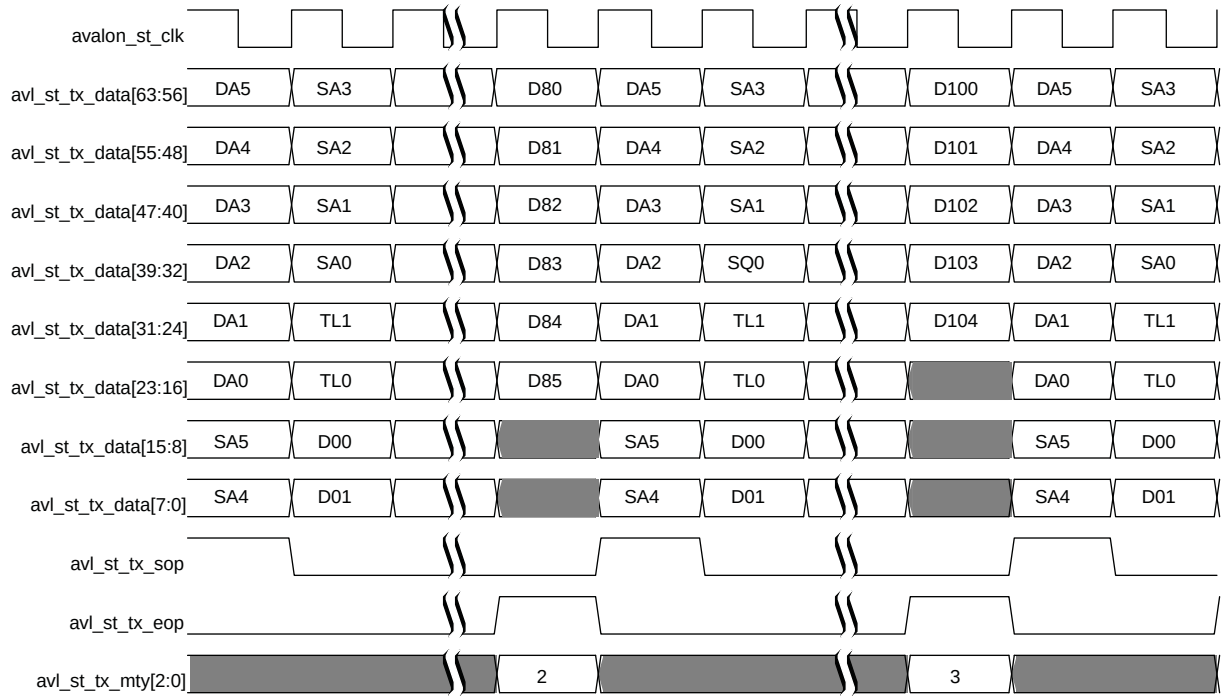
Note to Table 3–6:

(1) Destination Address[40] = type bit, broadcast/multicast bit.

(2) Destination Address[41] = Locally administrated address bit

For example, the destination MAC address includes the following six octets AC-DE-48-00-00-80, the first octet transmitted (octet 0 of the MAC address described in 802.3) is AC and the last octet transmitted (octet 7 of the MAC address) is 80. The first bit transmitted is the low-order bit of AC, a zero. The last bit transmitted is the high order bit of 80, a one. Figure 3–13 further illustrates how the octets of the client frame are transferred over the Tx datapath. As Table 3–6 and Figure 3–13 illustrate, 0xAC is driven on `DA5[47:40]` and 0x80 is driven on `DA0[7:0]`.

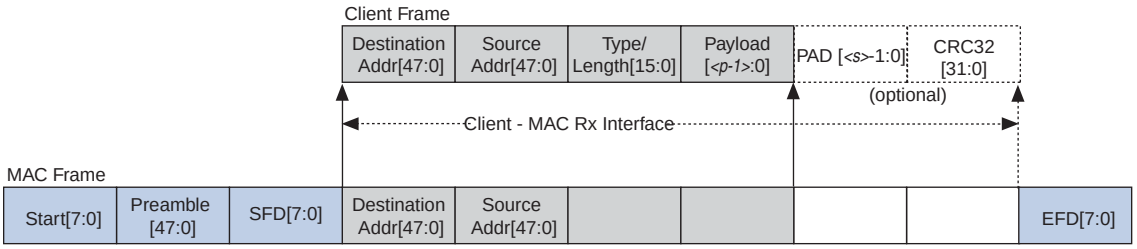
Figure 3–13. Octet Transmission on the Avalon-ST SIgnals



3.2.4. Receive Data Path

The Rx MAC receives Ethernet frames from the PHY and forwards the payload with relevant header bytes to the client after performing some MAC functions on header bytes. Some of the header bytes are optionally stripped from the frame before forwarding. Figure 3–14 illustrates the typical flow of frame through the MAC Rx.

Figure 3–14. Flow of Frame through the MAC Rx



Notes to Figure 3–14:

(1) <*p*> = payload size = 0–1500 bytes.

(2) <*s*> = pad bytes = 0–46 bytes.

The following section describe the functions performed by the Rx MAC.

3.2.4.1. Preamble Processing

The preamble sequence is Start, six preamble bytes, and SFD. If this sequence is incorrect the frame is ignored. The start word must be on receive lane 0 (most significant byte). The IP core uses the SFD byte (0xD5) to identify the last byte of the preamble. The MAC Rx looks for the Start, six preamble bytes and SFD. The MAC Rx removes all Start, SFD and IFG bytes from accepted frames. To be accepted the preamble must be preceded by a column of Idle `||I||` or an ordered set (for example, a local fault).

3.2.4.2. Payload Padding Removal

If the incoming frame includes padding, it is removed before forwarding the frame to Rx client. Padding removal may be optional depending on the payload length and the value of the `PAD_EN` bit in the `command_config` register. The IP core removes the padding, prior to sending the frames to the Rx FIFO, when the `PAD_EN` bit is set to 1 and the payload length is less than the following values for the different frame types:

- 46 bytes for basic frames
- 42 bytes for VLAN tagged frames
- 38 bytes for stacked VLAN tagged frames

If the `PAD_EN` bit is set to 0, complete frames including the padding are forwarded to the client.

3.2.4.3. CRC Checking

The 32-bit CRC field is received in the order: X^{32}, X^{30}, . . . X^{1}, and X^{0} , where X^{32} is the MSB of FCS field and occupies the LSB position on first FCS byte field. If a CRC-32 error is detected, the MAC Rx marks the frame invalid by asserting the `avl_st_rx_err`.

3.2.4.4. Rx CRC Forwarding

The CRC-32 field is forwarded to the client interface if the `CRC_FWD` bit of the `command_config` register is set.

Pad removal must be disabled by setting `PAD_EN` to 0 when enabling CRC forwarding to ensure client does not get incorrect CRC errors.

3.2.4.5. Address Checking

The Rx MAC supports all three types of addresses:

- Unicast—Specifies a destination address is a unicast (individual) address. Bit 0 is 0.
- Multicast—Specifies a destination address is a multicast or group address. Bit 0 is 1.
- Broadcast—Specifies a broadcast address when all 48 bits in the destination address are 1. The Rx MAC accepts a broadcast frames if enabled by the `BROAD_FILTER_ENA` configuration bit in the `command_config`. This bit is enabled by default. If the broadcast address is not accepted, the broadcast frame is discarded.

Unicast and multicast frames are only accepted by Rx MAC if its destination address matches either the primary address (configured in the `mac_0` and `mac_1` registers) or any of the supplementary addressees (configured in the `smac_0_0`, `smac_0_1`, `smac_1_0`, `smac_1_1`, `smac_2_0`, `smac_2_1`, `smac_3_0`, and `smac_3_1`registers). If the promiscuous mode of operation is enabled, address checking is omitted and all received client data frames are accepted.

3.2.4.6. Frame Length/Type Checking

This header field represents either the length or type of the received frame. If it represents the length, the MAC Rx ensures that the frame length conforms to the minimum and maximum constraints. The length of all frames must be at least 64 bytes. The maximum length of frames is configured in `frm_length` register. The following maximum values are defined for different frame types:

- The basic frames—The value in `frm_length` register.
- The VLAN tagged frames—The value in `frm_length` register plus four.
- The stacked VLAN tagged frames—The value in `frm_length` register plus eight.

When length checking is enabled, if the actual length of the frame received does not match the frame length field, the MAC asserts the `avl_st_rx_err`, indicating that received frame has an error.

Although the MAC supports a maximum frame size of 64 KBytes, the XAUI PCS only supports 16 KByte frames with a clock tolerance of ±100 ppm, as defined in the IEEE 802.3 standard. For the PCS to run frames of up to 64 KBytes, the clocks must be of identical frequency (0 ppm difference), therefore must originate from the same clock source.

The error detection capability of CRC32 code degrades if the Ethernet frame length exceeds 11,455 bytes. For more information refer to the *Extended Frame Sizes for Next Generation Ethernets* white paper available on the Pittsburgh Supercomputing Center website.

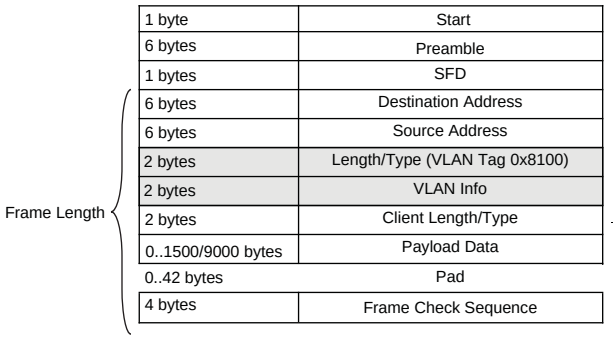
3.2.4.7. Payload Length Checking

The MAC Rx keeps track of the actual frame payload length as it receives a frame. The actual frame payload length is checked against the length/type or client length/type field. You can disable the payload length checking by setting the `NO_LGTH_CHECK` bit in the `command_config` register. For basic frames, VLAN frames and stacked VLAN frames, the payload range is 0x2E–0x5DC. For VLAN and VLAN stacked frames the total length of the header and payload is four or eight bytes greater, respectively. If the actual frame payload length violates the specification, the MAC asserts `avl_st_rx_err`, indicating that received frame has an error.

3.2.4.8. VLAN Frame Processing

A value of 0x8100 in the type/length field of a basic frame indicates an VLAN frame, and the next two bytes carry the tag information. The MAC Rx asserts `avl_st_vlan_tag` to mark VLAN frames. There are no other processing differences between basic and VLAN frames. Figure 3–15 illustrates the VLAN tagged frame format.

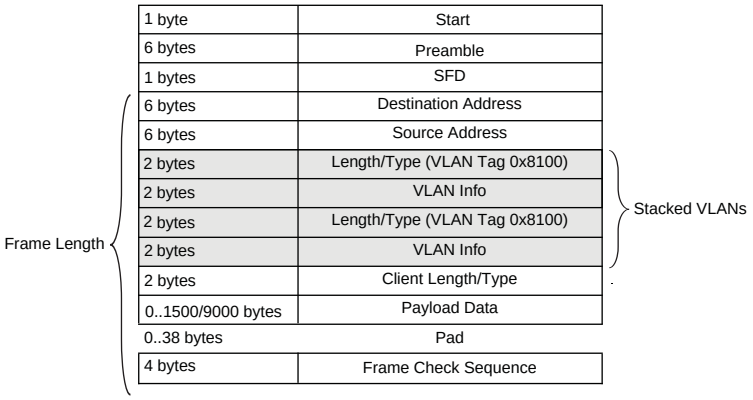
Figure 3–15. VLAN Tagged Frame Format



3.2.4.9. Stacked VLAN Frame Processing

A value of 0x8100 in the type/length field of a basic frame indicates an VLAN frame, and the next two bytes carry the stacked tag information. The MAC Rx asserts `avl_st_vlan_tag` and `avl_st_vlan_vlan_tag` to mark VLAN frames. There are no other processing differences between basic and VLAN frames. Figure 3–15 illustrates the stacked VLAN tagged frame format.

Figure 3–16. Stacked VLAN Tagged Frame Format



3.2.4.10. Pause Frame Termination

Pause frames are terminated within the receive engine; they are not forwarded to the receive FIFO unless the control register `PAUSE_FWD` bit is set to one in which case the pause is processed and forwarded. The MAC determines if a pause frame is valid by checking its CRC and frame length. The MAC extracts pause quanta (time to pause the Tx traffic) in a valid pause frame and forwards them to the transmit engine. Invalid pause frames are ignored. The flow control using pause frames is explained in "Congestion and Flow Control Using Pause Frames" on page 3–25.

3.2.4.11. Inter Packet Gap

IPG octets received are removed by the MAC Rx and are not forwarded to the client.

3.2.4.12. Pause Ignore

When this bit is set to one, the Pause frames are not processed, so that the MAC Tx traffic is not affected by the valid pause frames.

3.2.4.13. Control Frame Enable

In normal operation, the Rx MAC terminates control frames other than pause frames. If `CNTL_FRM_ENA` is set to one, the Rx MAC passes the control frame (but not pause frames) to the Avalon-ST interface. Pause frames have their own control register.

The MAC processes pause frame control as control frames and not pause frames. The Rx MAC terminates them in the same manner as other control frames unless the `CNTL_FRM_ENA` bit specifies otherwise.

3.2.4.14. MAC – PHY Rx Interface

Figure 3–17 illustrates the SDR XGMII interface for the Rx datapath.

Figure 3–17. SDR Rx XGMII Interface

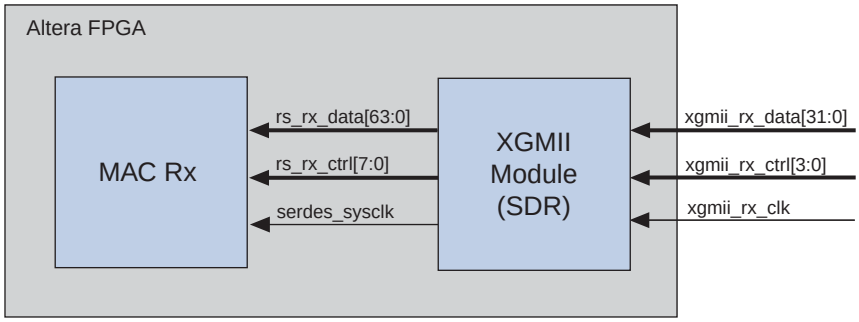


Table 3–7 describes the signals that comprise the SDR XGMII Rx interface.

Table 3–7. Rx XGMII SDR interface

Signal Name	Dir	Description
`rs_rx_data[63:0]`	O	8-lane data bus carrying bytes[7:0] from the PHY to the MAC.
`rs_rx_ctrl[7:0]`	O	8-bit signal from the PHY that indicates when a control octet is present on the corresponding `rs_rx_data` lane

3.2.4.15. DDR Rx XGMII Interface

Table 3–8 describes the signals that comprise the standard DDR XGMII Rx interface.

Table 3–8. Rx XGMII SDR interface

Signal Name	Dir	Description
`xgmii_rx_clk`	I	XGMII receive clock running at 156.25MHz. This clock should arrive shifted 90° with respect to data and control.
`xgmii_rx_data[31:0]`	I	4-lane data bus carrying bytes[7:0] from the Rx MAC, changing value on both edges of `xgmii_rx_clk`.
`xgmii_rx_ctrl[3:0]`	I	4-bit signal indicating when a control byte is present on corresponding `xgmii_rx_data` lane.

3.2.4.16. Client Side Interfaces of the Rx Datapath

The Rx interfaces employ the Avalon-ST interface. You can include an optional FIFO to buffer data between MAC and the client, or connect the MAC directly to the client.

3.2.4.17.Client FIFO Interface

In this configuration, the Rx MAC module is connected to the client through the FIFO. Figure 3–18 illustrates the signals that comprise this interface.

Figure 3–18. Client FIFO Interface Diagram

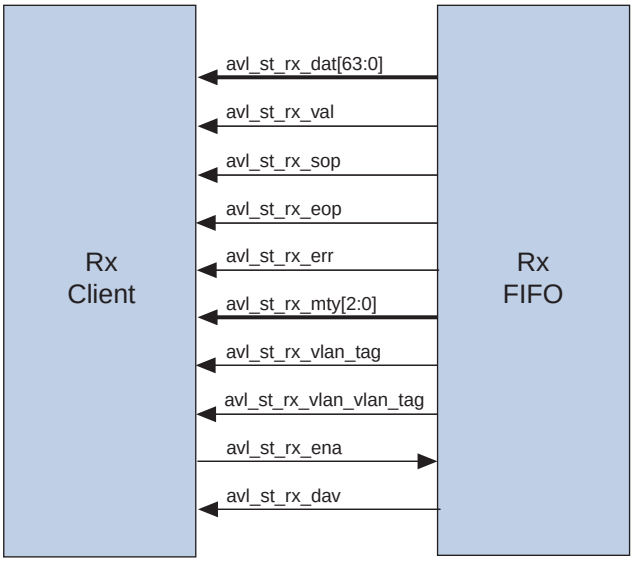


Table 3–9 describes the signals that comprise the standard DDR XGMII Rx interface.

Table 3–9. Client Rx FIFO Interface (Part 1 of 2)

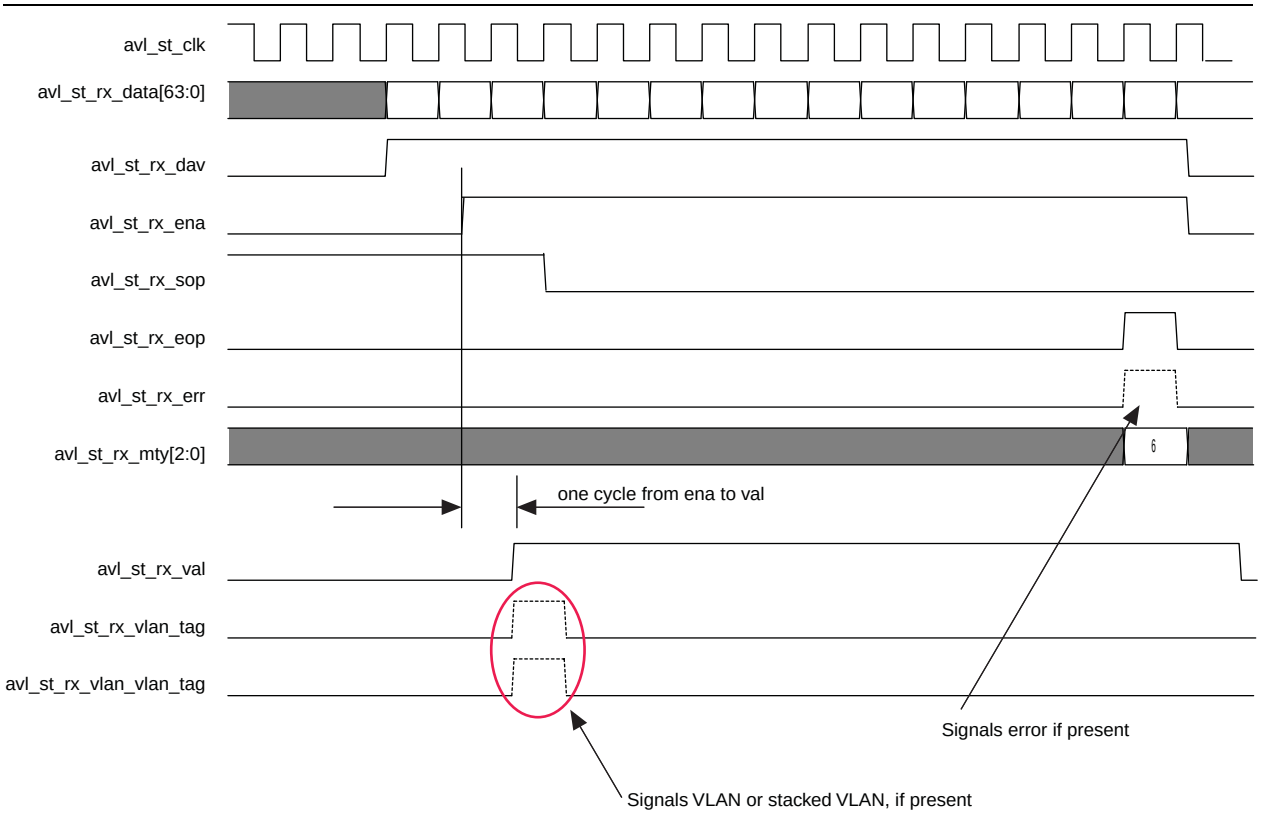
Signal Name	Dir	Description
`avl_st_rx_dat[63:0]`	O	64-bit client or source data for transmission.
`avl_st_rx_val`	O	When asserted indicates that the rest of the signals in the interface are valid.
`avl_st_rx_sop`	O	Asserted for one cycle to indicate the start of a MAC packet.
`avl_st_rx_eop`	O	Asserted for one cycle to indicate the end of a MAC packet.
`avl_st_rx_mty[2:0]`	O	Specifies how many bytes of `avl_st_rx_dat[63:0]` are empty when `avl_st_rx_eop` is asserted as follows: Value — Valid Data Bits 0 — `avl_st_rx_dat[63:0]` 1 — `avl_st_rx_dat[63:8]` 2 — `avl_st_rx_dat[63:16]` 3 — `avl_st_rx_dat[63:24]` 4 — `avl_st_rx_dat[63:32]` 5 — `avl_st_rx_dat[63:40]` 6 — `avl_st_rx_dat[63:48]` 7 — `avl_st_rx_dat[63:56]`
`avl_st_rx_dav`	O	Indicates that the FIFO has data for client's consumption.
`avl_st_rx_ena`	I	Read signal from the client indicating that it is ready to accept data from the FIFO.
`avl_st_rx_vlan_tag`	O	Indicates the received frame is a VLAN tagged frame.

Table 3–9. Client Rx FIFO Interface (Part 2 of 2)

Signal Name	Dir	Description
avl_st_rx_vlan_vlan_tag	O	Indicates the received frame is a stacked VLAN tagged frame.
avl_st_rx_err	O	Marks the current client packet as errored. This signal is asserted in conjunction with avl_st_rx_eop.

Figure 3–19 illustrates the timing for this interface.

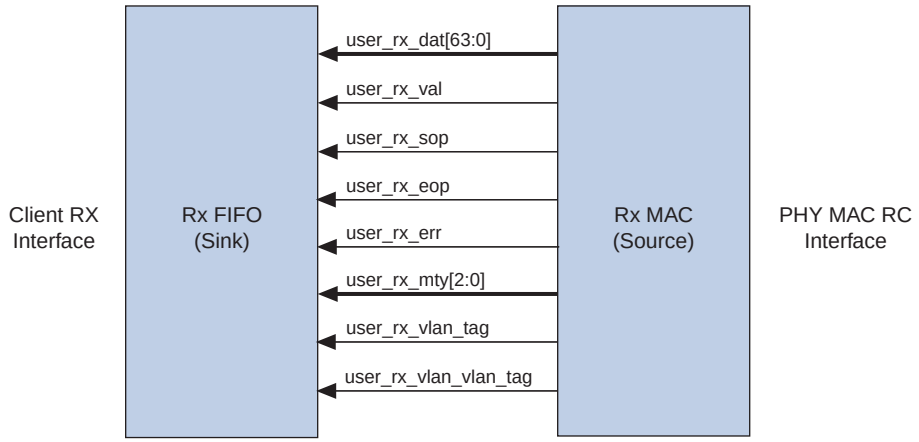
Figure 3–19. FIFO Client Rx Interface Timing



3.2.4.18. MAC FIFO Interface

Figure 3–20 illustrates the MAC FIFO Rx interface.

Figure 3–20. RX MAC FIFO Interface



The Rx FIFO interface does not include a backpressure signal to stop the flow of data from the Rx MAC. The sink must be able to consume data continuously from the MAC.

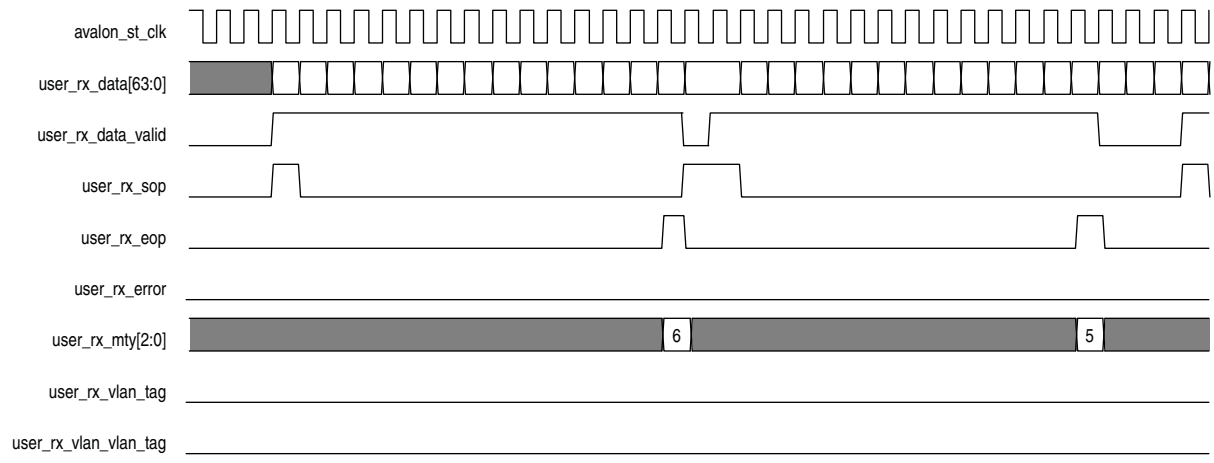
Table 3–10 describes the signals that comprise the client Rx FIFO interface.

Table 3–10. Client Rx FIFO Interface

Signal Name	Dir	Description
`user_rx_dat[63:0]`	O	64-bit client or source data for transmission.
`user_rx_val`	O	When asserted indicates that the rest of the signals in the interface are valid.
`user_rx_sop`	O	Asserted for one cycle to indicate the start of a packet.
`user_rx_eop`	O	Asserted for one cycle to indicate the end of a packet.
`user_rx_mty[2:0]`	O	Specifies how many bytes of `user_rx_dat[63:0]` are empty when `user_rx_eop` is asserted as follows: Value — Valid Data Bits 0 — `user_rx_dat[63:0]` 1 — `user_rx_dat[63:8]` 2 — `user_rx_dat[63:16]` 3 — `user_rx_dat[63:24]` 4 — `user_rx_dat[63:32]` 5 — `user_rx_dat[63:40]` 6 — `user_rx_dat[63:48]` 7 — `user_rx_dat[63:56]`
`user_rx_error`	O	Indicates the current packet has an error. Asserted in conjunction with the `user_rx_eop` signal.
`user_rx_vlan_tag`	I	Indicates that the received frame is a VLAN tagged frame.
`user_rx_vlan_vlan_tag`	I	Indicates that the received frame is a stacked VLAN tagged frame.

Figure 3–21 illustrates the timing for this interface.

Figure 3–21. FIFO MAC Rx Interface Timing Diagram



3.2.4.19. Error Conditions on Rx Datapath

The Rx MAC indicates error conditions by asserting `avl_st_rx_err`. The following error conditions are defined:

- Received frame terminated early or with an error
- Received frame has a CRC error
- Received frame length does not match length field
- Error characters received from PHY
- Rx link fault, receive frame is too short (less than 64 bytes) or too long (longer than the maximum specified length)

If your design includes a FIFO in store and forward mode, `setting the FIFO_ERR_DIS` bit to 1 in the `command_config` register discards the errored packets in the FIFO.

3.3. ECC Options

The 10-Gbps Ethernet IP core ECC feature implements single-bit error correction and double-bit error detection (SECDED).

The ALTECC megafunction performs the ECC encoding and decoding for this IP core. You can find additional information about this megafunction in the *ALTECC (Error Correction Code: Encoder/Decoder* section of the *Integer Arithmetic Megafunctions User Guide*.

The ECC feature changes FIFO entry widths as shown in Table 3–11.

Table 3–11. Receive and Transmit FIFO Entries With and Without ECC Bits

Information Type	Width Without ECC (bits)	Number of ECC Bits	Width with ECC (bits)
Data	64	8	72
Control	8	5	13

In the rate-matching FIFO in the Soft XAUI PCS, the ECC feature changes FIFO entry widths as shown in Table 3–12.

Table 3–12. Soft XAUI Rate-Matching FIFO Entries With and Without ECC Bits

Information Type	Width Without ECC (bits)	Number of ECC Bits	Width with ECC (bits)
Data	64	8	72
Control	18	6	24

In the per-lane deskew FIFOs in the Soft XAUI PCS, the ECC feature changes FIFO entry widths as shown in Table 3–13.

Table 3–13. Soft XAUI PCS Deskew FIFO Entries With and Without ECC Bits

Information Type	Width Without ECC (bits)	Number of ECC Bits	Width with ECC (bits)
Data	22	6	28

ECC-detected errors are indicated on the following three output signals:

- `ecc_sbe`—Single-bit error detected and corrected
- `ecc_mbe`—Multiple-bit error, more specifically a double-bit error, detected
- `ecc_packet_dropped`—Packet was dropped from Rx or Tx FIFO

These signals are asserted for one Avalon-MM clock cycle to indicate a relevant error. A packet is dropped if an ECC-detected multiple-bit error obscures a valid SOP or EOP indication. Technically, two packets are merged, resulting in one incorrect packet in place of two correct packets. In addition, when the IP core detects a multiple-bit error in a packet, it flags an error by asserting the `err` bit with the packet in the EOP cycle.

The Soft XAUI PCS does not report packets dropped, but a packet start or end in a XAUI FIFO can be corrupted. If a multiple-bit error is detected, the IP core inserts a disparity error in the FIFO entry, which affects two columns. The IDLE conversion state machine translates the disparity error to a local fault. If an error occurs during a frame, the MAC terminates the frame and asserts the `ecc_mbe` signal.

Turning on the ECC feature instantiates a comprehensive set of error insertion registers to support error insertion for ECC testing, and a set of statistics counters that software can read. For more information about these registers, refer to "ECC Monitoring and Testing" on page 3–44.

3.4. Congestion and Flow Control Using Pause Frames

The 10-Gbps Ethernet MAC provides flow control to control congestion at the local or remote link partner. When either link partner devices experience congestion, the respective transmit control sends pause frames. The pause frame instructs the remote transmitter to stop sending data for the duration specified by the congested receiver.

When a device receives the XOFF pause control frame, it stops transmitting frames to the link partner for a period equal to the pause quanta of the incoming pause frame. The pause quanta can be configured in the pause quanta register of the device sending XOFF frames. If the pause frame is received in the middle of a frame transmission, the transmitter finishes sending the current frame and then suspends transmission for a period specified by the pause quanta. Data transmission resumes when a pause frame with quanta of zero is received or when the timer has expired. The pause quanta received overrides any counter currently stored. When more than one pause quanta is sent, the value of the pause is set to the last quanta received.

XOFF pause frames stop the remote transmitter. XON pause frames let remote transmitter resume data transmission. Figure 3–22 illustrates these frames.

Figure 3–22. The XOFF and XONN Pause Frames *(Note 1)*

XOFF Frame
START[7:0]
PREAMBLE[47:0]
SFD[7:0]
DESTINATION ADDRESS[47:0] = 0x010000C28001
SOURCE ADDR[47:0]
TYPE[15:0] 0x0808
OPCODE[15:0] - 0X0001
PAUSE QUANTA[15:0] = 0xP1, 0xP2 *(2)*
PAD[355:0]
CRC[31:0]

XON Frame
START[7:0]
PREAMBLE[47:0]
SFD[7:0]
DESTINATION ADDRESS[47:0] = 0x010000C28001
SOURCE ADDR[47:0]
TYPE[15:0] 0x0808
OPCODE[15:0] - 0X0001
PAUSE QUANTA[15:0] = 0x00000000
PAD[355:0]
CRC[31:0]

Note to Figure 3–22:

(1) One pause quanta fraction is equivalent to 512 bit times, which equates to 512/64 (the width of the MAC data bus), or 8 cycles for the system clock.

(2) The bytes `P1` and `P2` are filled with the value configured in the `pause_quant` register.

3.4.1. Conditions Triggering XOFF Frame Transmission

The Tx MAC transmits XOFF frames when one of the following conditions occurs:

- Local device congestion (Rx FIFO almost full on)—When the local Rx FIFO asserts the `rx_almost_full` flag, XOFF pause frames are sent to the remote device. Setting the config register bit `NO_PAUSE_FIFO` disables this behavior. This feature is only available if you choose to include the optional FIFO in your design.

  ☞ If you use this mode be sure to size the FIFO to meet the IEEE 802.3 latency requirements.

- Client requests XOFF transmission—A client can explicitly request that XOFF frames be sent using the `xoff_request`. When this signal is asserted for one clock cycle, an XOFF frame is sent to the Ethernet network when the current frame transmission completes.
- Host (software) requests XOFF transmission—Setting the `XOFF_GEN` bit in the `command_config` register triggers a request that an XOFF frame be sent.

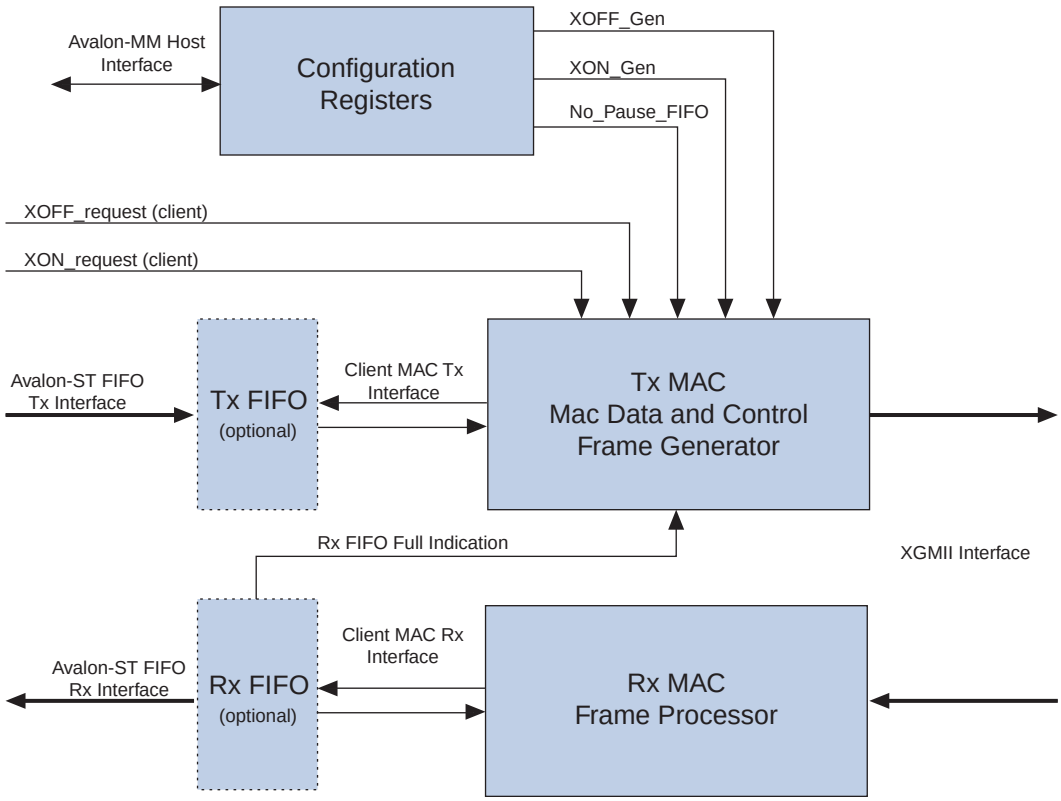
3.4.2. Conditions Triggering XON frame transmission

The Tx MAC transmits XON frames when one of the following conditions occurs:

- Local device congestion clears (FIFO not full)—When the local Rx FIFO deasserts the `rx_almost_full` flag, XON pause frames are sent to the remote device, indicating congestion has ended. Setting the `NO_PAUSE_FIFO` bit of the `command_config` register disables this behavior.
- Client requests XON transmission—A client can explicitly request that XON frames be sent using the `xon_request`. When this signal is asserted for one clock cycle, an XON frame is sent to the Ethernet network.
- Host (software) requests XON transmission—Setting the `XON_GEN` bit in the `command_config` register triggers a request that an XON frame be sent.

Figure 3–23 provides a block diagram of the pause transmission logic.

Figure 3–23. Block Diagram of Pause Transmission Logic

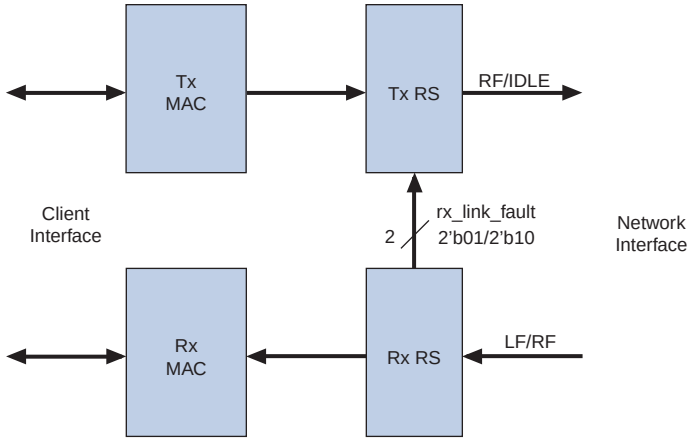


3.4.3. Link Fault Signaling

Altera's 10-Gbps Ethernet MAC includes a reconciliation sublayer (RS) that handles local and remote faults. The RS functionality is included in the MAC module. When the local PHY reports a local fault by sending 0x9c00000001 control characters, the RS starts sending the remote fault signal (0x9c0000002) to the PHY which is eventually received by the remote link partner. Similarly, when the local device receives a remote fault control character, the RS transmits idle control characters. Once normal data is received by the Rx, RS, and MAC/RS modules, the Tx module resumes sending Ethernet frames carrying client data.

When local or remote fault signals are received by local RS, it sets the appropriate bit in the `rx_link_fault` status register. The msb records the remote fault status and lsb records the local fault status. These status bits are not sticky. Once a fault condition disappears and normal data traffic is received for more than 127 columns, these fault status bits are cleared. When the RS starts sending Remote Fault/IDLE, the data sent by the Mac Tx is ignored. All data sent by the Tx is lost. Figure 3–24 illustrates the fault signaling.

Figure 3–24. Fault Signalling



Note to Figure 3–24:

(1) Tx RS sends a remote fault (RF) or idle respectively when Rx RS receives a local fault (LF) or remote fault (RF).

3.4.4. Modes of Operations

You can configure the Altera MAC datapath for different modes of operation. The following sections describe these options.

3.4.4.1. Regular Mode

This is the normal mode of operation when the Altera MAC transmits and receives data to and from a remote link partner MAC. It is also called the filtered or non-promiscuous mode of operation. In this mode, MAC Tx performs all enabled functions on client data and transmits on the physical media. The Rx MAC performs address filtering, various header checking, and control frame termination as per the IEEE 802.3 standard. The following functions are available in the regular mode:

- Tx Operations
  - Start, preamble, and SFD insertion
  - Payload padding
  - Source address overwrite by MAC (optional)
  - CRC insertion
- Rx Operations
  - Start, preamble, and SFD striping
  - Payload padding removal
  - CRC checking
  - CRC stripping
  - Address checking

3.4.4.2. Promiscuous Mode

In promiscuous mode, most of the MAC functions, especially on the Rx datapath are ignored; the data is passed on to the client after stripping the preamble and SFD. The following functions are available in promiscuous mode:

- Tx Operations
  - Add preamble and SFD
  - Payload padding
  - CRC insertion
- Rx Operations
  - Preamble, SFD, and IPG stripping
  - CRC check plus forwarding to the client.
  - No Payload padding removal

Table 3–14 lists the settings required to run in promiscuous mode.

Table 3–14. Promiscuous Mode Settings (Part 1 of 2)

Configuration Bit	Value	Description
`promis_en`	1	Enables promiscuous operation in the receiver. Destination MAC address is not filtered.
`pad_en`	0	Disables pad removal from Rx frame.
`crc_fwd`	1	Receiver forwards the CRC field as received from the network to the client or user application. Receiver always checks the CRC field and increments the respective statistics counter and sets the receive frame error bit.
`pause_fwd`	1	Receiver forwards the pause frames to the client or user application.
`pause_ignore`	0	Receiver ignores received pause frames.
`tx_addr_ins`	0	Transmitter does not modify and does not insert the source MAC address into the transmit frame from user application.
`ctrl_frm_ena`	1	Receiver accepts and forwards to client all control frames except those for flow control.

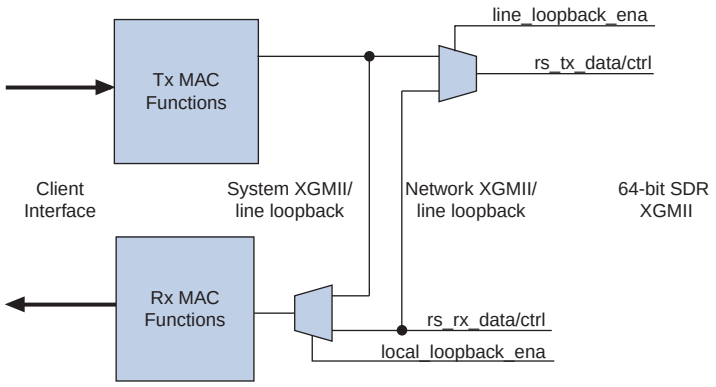
Table 3–14. Promiscuous Mode Settings (Part 2 of 2)

Configuration Bit	Value	Description
`no_length_check`	1	Receiver does not check the received frame payload length against the length/type field of the frame.
`broad_fltr_ena`	0	Broadcast filtering enable. When set to 1, received broadcast frames are filtered out.
`frm_length`	`Max Size`	Ensures all frames are forwarded to the client.

3.4.4.3. Loopback Modes

You can configure the 10-Gbps Ethernet IP core in various loopback modes to facilitate debugging and testing. Figure 3–25 illustrates the loopback modes available.

Figure 3–25. Loopback Modes



3.4.4.4. System XGMII Loopback/Local XGMII Loopback

When enabled by the `LOCAL_LOOP_ENA` bit of the `command_config` register, the MAC Rx starts accepting data and control transmitted by MAC Tx at the SDR XGMII interface. This loopback essentially takes client data through the MAC Tx and returns it to the client via MAC Rx. Transition to and from this loopback mode takes effect when the current frame completes. When enabled, idle characters are transmitted to the remote link partner.

3.4.4.5. Network XGMII Loopback/Remote XGMII Loopback/Line Loopback

When enabled by the `LINE_LOOP_ENA` bit of the `command_config` register, Ethernet traffic received from remote device is looped back to it at the XGMII interface. To use this mode, complete the following steps:

1. Stop transmitting data.
2. Put the IP core into line loopback mode by completing the following steps:
   a. Disable the MAC Tx in the configuration register.
   b. Turn on line loopback mode.
   c. After completing loopback testing, reenable the MAC Tx.
3. Resume transmission.

The MAC RX receives idles while the line loopback is enabled.

3.4.4.6. Local and Line Loopback used simultaneously

You can use both local and line loopbacks simultaneously. In this mode, the MAC Tx data is sent to the MAC Rx data while the line Rx data is forwarded to the MAC Tx data.

3.5. Software Programming Interface

Host software can access the configuration and statistics registers over the Avalon-MM interface. The Avalon-MM interface employes a standard memory-mapped protocol that is typically used for control and status updates. The following are key features of this interface:

- Host software uses a parallel address bus, read, write, write data, and read data signals to perform its write and read operations.
- A client can insert wait states, if the read or write operation cannot to be completed in the next clock cycle.
- The 32-bit register interface is accessed using byte addresses.

For a more information about the Avalon-MM protocol refer to the *Avalon Interface Specifications*.

Table 3–15 lists the signals that comprise this interface.

Table 3–15. Avalon-MM Control and Status Register Interface

Signal	Direction	Description
`avalon_clk`	I	Clock input.
`avalon_reset_n`	I	Active low reset.
`avalon_address[8:0]`	I	9 bit register word address. The word address is formed by right-shifting the lower two bits of the byte address provided by an 8-bit host. For example, a byte address of 0x5C becomes a word address of 0x17.
`avalon_write`	I	Write request from the master.
`avalon_writedata[31:0]`	I	Write data.
`avalon_waitrequest`	O	Output to request wait cycles.
`avalon_read`	I	Read request from the master,
`avalon_readdata[31:0]`	O	Data read from the slave register.

Figure 3–26 illustrates the timing for reads on the Avalon-MM interface.

Figure 3–26. Typical Avalon-MM Interface TIming for Multiple Reads

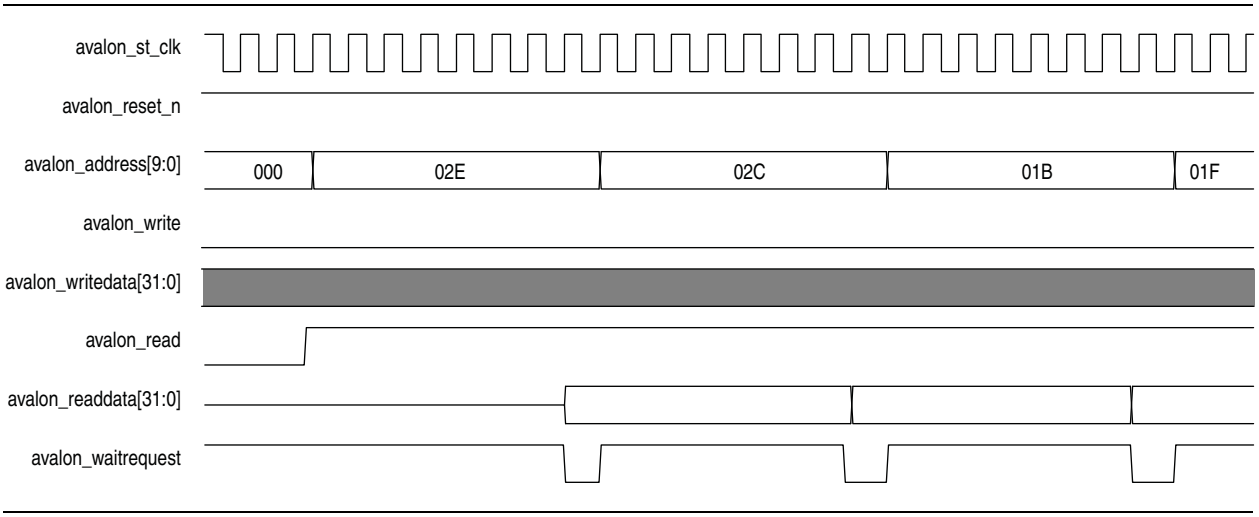
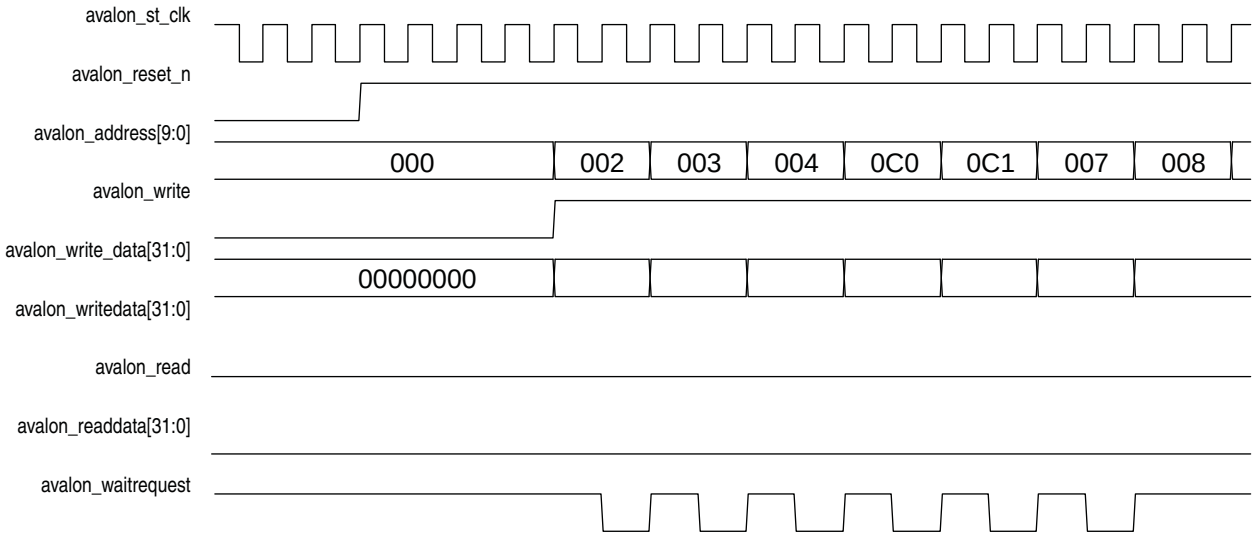


Figure 3–27 illustrates the timing for writes on the Avalon-MM interface.

Figure 3–27. Typical Avalon-MM Interface TIming for Multiple Writes



3.6. Register Descriptions

This section defines the control interface register map. Table 3–16 shows each usable register, except for the ECC feature management registers which are described in "ECC Monitoring and Testing" on page 3–44. In Table 3–16, the HW reset, SW reset and access columns provide the following information:

- The HW reset column specifies the value after hardware reset, which is controlled by the `reset` signal.

- The SW reset column specifies the value or influence after a software reset, which is controlled by the `SW_RESET` bit in the `command_config` register.
  - "—" indicates that `reset` is not relevant to this register and has no influence.
  - "X" indicates that the value is unknown, which is typical for memory-based registers.
- The access column indicates whether you can only read a register (RO), write it (WO) or read and write it (RW).

You must perform a software reset before you write to a non-ECC register, then re-enable the design.

For more information about how to access 64-bit registers, refer to "64-Bit Statistics Counters" on page 3–43.

Table 3–16. Control Interface Register Map (Part 1 of 7)

Address Offset	Name	Description	Access	HW Reset	SW Reset
`0x000`	`rev`	Revision. This register is divided into two 16-bit fields: ■ Bits 15:0: IP core revision, set to `0x0702` ■ Bit 31:16: Customer specific revision, set to `0` during IP core configuration. This field is controlled by the parameter `CUST_VERSION` defined in the top level generated for the 10-Gbps Ethernet IP core instance.	RO	0x00000901	—
`0x004`	`scratch`	Scratch register. Provides a memory location for user applications to test the device memory operation.	RW	0	—
`0x008`	`command_config`	Command register. The host processor uses this register to control and configure the IP core.	RW	0	bit 0 = 0 bit 1 = 0 Others not modified
`0x00C`	`mac_0`	32-bit primary address word 0: bits 0–31 of the primary address. Bit 0 maps to bit 0 of the address, bit 1 maps to bit 1 of the address, and so on.	RW	0	—
`0x010`	`mac_1`	32-bit primary address word 1: bits 32–47 of the primary address. Bits 16–31 are reserved. Bit 0 maps to bit 32 of the address.	RW	0	—

Table 3–16. Control Interface Register Map (Part 2 of 7)

Address Offset	Name	Description	Access	HW Reset	SW Reset
0x014	frm_length	14-bit maximum frame length. The receive logic uses this value to check frames. Typical value is 1518. Bits 14 –31 are reserved.	RW	1518	—
0x018	pause_quant	16-bit pause quanta. The pause quanta is used in each pause frame sent to a remote Ethernet device, in increments of 512 Ethernet bit times. Bits 16 –31 are reserved.	RW	0	—
0x01C	rx_almost_empty_off	FIFO almost empty off threshold. Bits 12 –31 are unused.	RW	0	—
0x020	rx_almost_empty_on	FIFO almost empty on threshold. The number writeable bits depends on the FIFO size. For example, if the FIFO has 1024 bytes, 10 bits are writeable. The bottom 3 bits are ignored so that a threshold of 0x100 and 0x102 is the same.	RW	0	—
0x024	tx_almost_empty_off	FIFO almost empty off threshold. The number writeable bits depends on the FIFO size. The bottom 3 bits are ignored so that a threshold of 0x100 and 0x102 is the same.	RW	0	—
0x028	tx_almost_empty_on	FIFO almost empty on threshold. The number writeable bits depends on the FIFO size. The bottom 3 bits are ignored so that a threshold of 0x100 and 0x102 is the same.	RW	0	—
0x02C	rx_almost_full_off	FIFO almost-full threshold. The number writeable bits depends on the FIFO size. The bottom 3 bits are ignored so that a threshold of 0x100 and 0x102 is the same.	RW	0	—
0x030	rx_almost_full_on	FIFO almost-full threshold. The number writeable bits depends on the FIFO size. The bottom 3 bits are ignored so that a threshold of 0x100 and 0x102 is the same.	RW	0	—
0x034	tx_almost_full_off	Transmit FIFO almost-full threshold. The number writeable bits depends on the FIFO size. The bottom 3 bits are ignored so that a threshold of 0x100 and 0x102 is the same.	RW	0	—
0x038	tx_almost_full_on	Transmit FIFO almost-full threshold. The number writeable bits depends on the FIFO size. The bottom 3 bits are ignored so that a threshold of 0x100 and 0x102 is the same.	RW	0	—

Table 3–16. Control Interface Register Map (Part 3 of 7)

Address Offset	Name	Description	Access	HW Reset	SW Reset
0x03C	mdio_addr0	MDIO address of PHY device 0. ■ 4:0 are the PHY address (clause 22) or the device address (clause 45) ■ 12:8 are the port address (clause 45 only) ■ 31:16 are the register address (clause 45 only). To read and write for clause 45, the read and write must be done at address 0x320	RW	0	—
0x040	Unused	Unused.	RO	0	—
0x044 – 0x054	Reserved	Reserved.	RO	0	—
0x058	Reserved	Reserved.	RO	0	—
0x05C	tx_ipg_length	Minimum IPG. Valid values are 8–252 bytes. If this register is set to an invalid value, it defaults to 12 byte-times which is a typical value of minimum IPG. Bits 5–31 are reserved and set to read-only value 0.	RW	12	—
0x060 0x064	aMacID	This register is wired to mac_0 and mac_1 addresses respectively.	RO	0	—
0x068	aFramesTransmittedOK_0	See Table 3–18 on page 3–41.	RO	0	0
0x06C	aFramesReceivedOK_0	See Table 3–18 on page 3–41.	RO	0	0
0x070	aFrameCheckSequence Errors	See Table 3–18 on page 3–41.	RO	0	0
0x074	aAlignmentErrors	See Table 3–18 on page 3–41.	RO	0	0
0x078	aOctetsTransmittedOK_0	See Table 3–18 on page 3–41.	RO	0	0
0x07C	aOctetsReceivedOK_0	See Table 3–18 on page 3–41.	RO	0	0
0x080	aTxPAUSEMACCtrlFrames	See Table 3–18 on page 3–41.	RO	0	0
0x084	aRxPAUSEMACCtrlFrames	See Table 3–18 on page 3–41.	RO	0	0
0x088	ifInErrors	See Table 3–19 on page 3–41.	RO	0	0
0x08C	ifOutErrors	See Table 3–19 on page 3–41.	RO	0	0
0x090	ifInUcastPkts	See Table 3–19 on page 3–41.	RO	0	0
0x094	ifInMulticastPkts	See Table 3–19 on page 3–41.	RO	0	0
0x098	ifInBroadcastPkts	See Table 3–19 on page 3–41.	RO	0	0
0x09C	ifOutDiscards	See Table 3–21 on page 3–43.	RO	0	0
0x0A0	ifOutUcastPkts	See Table 3–19 on page 3–41.	RO	0	0
0x0A4	ifOutMulticastPkts	See Table 3–19 on page 3–41.	RO	0	0
0x0A8	ifOutBroadcastPkts	See Table 3–19 on page 3–41.	RO	0	0

Table 3–16. Control Interface Register Map (Part 4 of 7)

Address Offset	Name	Description	Access	HW Reset	SW Reset
0x0AC	etherStatsDropEvents	See Table 3–19 on page 3–41.	RO	0	0
0x0B0	etherStatsOctets_0	See Table 3–19 on page 3–41.	RO	0	0
0x0B4	etherStatsPkts_0	See Table 3–20 on page 3–42.	RO	0	0
0x0B8	etherStatsUndersize Pkts	See Table 3–20 on page 3–42.	RO	0	0
0x0BC	etherStatsOversizePkts	See Table 3–20 on page 3–42.	RO	0	0
0x0C0	etherStatsPkts64Octets	See Table 3–20 on page 3–42.	RO	0	0
0x0C4	etherStatsPkts65to127 Octets	See Table 3–20 on page 3–42.	RO	0	0
0x0C8	etherStatsPkts128to255 Octets	See Table 3–20 on page 3–42.	RO	0	0
0x0CC	etherStatsPkts256to511 Octets	See Table 3–20 on page 3–42.	RO	0	0
0x0D0	etherStatsPkts512to102 3Octets	See Table 3–20 on page 3–42.	RO	0	0
0x0D4	etherStatsPkts1024to15 18Octets	See Table 3–20 on page 3–42.	RO	0	0
0x0D8	etherStatsPkts1519toX Octets	See Table 3–20 on page 3–42.	RO	0	0
0x0DC	etherStatsJabbers	See Table 3–20 on page 3–42.	RO	0	0
0x0E0	etherStatsFragments	See Table 3–20 on page 3–42.	RO	0	0
0x0E4	Reserved	Unused.	RO	0	—
0x0E8	linkFaultDetect	Link remote and local fault detection according to IEEE 802.3ae: ■ Bit[0] =1'b1 and indicates a local fault has been detected ■ Bit[1] = 1'b1 and indicates a remote fault has been detected ■ Bit [1:0] = 2'b00 and indicates link is OK All other bits are currently unused.	RO	0	—

Table 3–16. Control Interface Register Map (Part 5 of 7)

Address Offset	Name	Description	Access	HW Reset	SW Reset
0x200 – 0x27C	PHY Device 0 Internal Registers	Registers 0–31 within PHY device 0 connected to the MDIO PHY management interface. Reading or writing immediately causes the corresponding MDIO transaction to read or write the underlying PHY device register. The register at address offset 0x200 corresponds to register 0 of PHY device 0. The register at address offset 0x204 corresponds to register 1 of PHY device 0. For all registers, bits 15:0 are significant. Bits 31:16 should be written with 0 and ignored on read.	RW	0	—
0x280 – 0x2FC	Reserved	Reserved.	RO	0	—
0x300	smac_0_0	Supplemental address 0, bits 31:0. Register bit 0 maps to bit 0 of the address, bit 1 maps to bit 1 of the address, and so on.	RW	0	—
0x304	smac_0_1	Supplemental address 0, bits 47:32. Register bit 0 maps to bit 32 of the address. Register bits 30:16 are reserved. register bit 31 enables address: ■ 0 to disable ■ 1 to enable	RW	0	—
0x308	smac_1_0	Supplemental address 1, bits 31:0. Register bit 0 maps to bit 0 of the address, bit 1 maps to bit 1 of the address, and so on.	RW	0	—
0x30C	smac_1_1	Supplemental address 1, bits 47:32. Register bit 0 maps to bit 32 of the address. Register bits 30:16 are reserved. register bit 31 enables address: ■ 0 to disable ■ 1 to enable	RW	0	—
0x310	smac_2_0	Supplemental address 2, bits 31:0. Register bit 0 maps to bit 0 of the address, bit 1 maps to bit 1 of the address, and so on.	RW	0	—

Table 3–16. Control Interface Register Map (Part 6 of 7)

Address Offset	Name	Description	Access	HW Reset	SW Reset
0x314	smac_2_1	Supplemental address 2, bits 47:32. Register bit 0 maps to bit 32 of the address. Register bits 30:16 are reserved. register bit 31 enables address: ■ 0 to disable ■ 1 to enable	RW	0	—
0x318	smac_3_0	Supplemental address 3, bits 31:0. Register bit 0 maps to bit 0 of the address, bit 1 maps to bit 1 of the address, and so on.	RW	0	—
0x31C	smac_3_1	Supplemental address 3, bits 47:32. Register bit 0 maps to bit 32 of the address. Register bits 30:16 are reserved. register bit 31 enables address: ■ 0 to disable ■ 1 to enable	RW	0	—
0x320	MDIO clause 45	Reading or writing to this address tells the MAC to perform an MDIO read or write following the IEEE 802.3 specification. The address of the register, device and port are specified address 0x03C	RW	—	—
0x324	aFramesTransmittedOK_1	Packet and byte/octet counts for both transmitter and receiver are 64 bits wide. For more information about how to access 64-bit registers, see "64-Bit Statistics Counters" on page 3–43. See also Table 3–18 on page 3–41 and Table 3–20 on page 3–42.	RO	0	0
0x328	aFramesReceivedOK_1				
0x32C	aOctetsTransmittedOK_1				
0x330	aOctetsReceivedOK_1				
0x334	etherStatsOctets_1				
0x338	etherStatsPkts_1				
0x33C	Reserved	Reserved.	—	—	—
0x340	ALTGX status0	Status of the transceivers (XAUI only). ■ 0 pll_locked ■ 1 rx_channelaligned ■ 7:4 rx_freqlocked ■ 11:8 rx_pll_locked For more information, refer to the "Stratix IV Transceiver Architecture" chapter in the *Stratix IV Device Handbook*.	RO	—	—

Table 3–16. Control Interface Register Map (Part 7 of 7)

Address Offset	Name	Description	Access	HW Reset	SW Reset
0x344	ALTGX status1	Status of the transceivers (XAUI only). ■ 7:0 rx_disperr ■ 15:8 rx_errdetect ■ 23:16 rx_patterndetect ■ 31:24 rx_syncstatus For more information, refer to the "Stratix IV Transceiver Architecture" chapter in the *Stratix IV Device Handbook*.	RO	—	—
0x348	Shadow MDIO	Stores the last read value from the MDIO access.	—	—	—

3.6.1. Command_Config Register

Table 3–17 describes the function of each bit and field in the command_config register.

Table 3–17. Command_Config Register Bit Descriptions (Part 1 of 3)

Bit(s)	Bit Name	Access	Description
0	TX_ENA	RW	Transmit enable. Setting this bit to 1 enables the transmit datapath. This bit is cleared following a hardware or software reset. Refer to the SW_RESET bit description.
1	RX_ENA	RW	Receive enable. Setting this bit to 1 enables the receive datapath. This bit is cleared following a hardware or software reset. See the the SW_RESET bit description.
2	XON_GEN	RW	Pause frames generation. When this bit is set to 1, the IP core generates a pause frame with a pause quanta of 0, independent of the receive FIFO status.
3	Reserved	—	Reserved.
4	PROMIS_EN	RW	Promiscuous enable. Setting this bit to 1 enables promiscuous operation in which the destination address of the receive frame is not checked.
5	PAD_EN	RW	Pad enable. Setting this bit to 1 enables pad removal in receive frames. The IP core removes receive frame padding before forwarding the frames to the user application. Transmit frames are always padded and this bit has no effect.
6	CRC_FWD	RW	Receive CRC forwarding. ■ If this bit is set to 1, the IP core forwards the CRC field to the user application. ■ If this bit is set to 0, the IP core removes the CRC field from the frame before forwarding the frame to the user application. ■ This bit is ignored if the PAD_EN bit is 1. In this case, the IP core checks the CRC field and removes it from the frame before forwarding the frame to the user application.

Table 3–17. Command_Config Register Bit Descriptions (Part 2 of 3)

Bit(s)	Bit Name	Access	Description
7	PAUSE_FWD	RW	Receive pause frame forwarding. Terminates or forwards pause frames. ■ If this bit is set to 1, the IP core forwards pause frames to the user application. ■ If this bit is set to 0, the IP core terminates and discards pause frames.
8	PAUSE_IGNORE	RW	Ignore pause frame quanta. ■ Setting this bit to 1 causes the IP core to ignore received pause frames. ■ Setting this bit to 0 causes the transmit process to stop for an amount of time specified in the pause quanta within the pause frame.
9	TX_ADDR_INS	RW	Set address on transmit. ■ If this bit is set to 1, the IP core overwrites the transmit frame source address with the address configured in the `mac_0` and `mac_1` registers, or in any of the supplemental address registers. ■ If this bit is set to 0, the IP core does not modify the source address.
12:10	Reserved	—	Reserved.
13	SW_RESET	RW	Software reset command. Setting this bit to 1 causes the IP core to disable the transmit and receive logic, flush the receive FIFO, and reset the statistics counters. This bit is automatically cleared when the software reset sequence completes.
14	Reserved	—	Reserved.
15	LOCAL_LOOP_ENA	RW	Local loopback enable. Setting this bit to 1 enables loopback. Frames sent through the transmitter interface are looped back into the receiver interface at the XGMII before the PCS. The transition from normal operation to loopback and from loopback to normal operation only occurs between frames. If the IP core is transmitting a packet, the transition occurs after the EOP and when the frame is complete. Idle characters are transmitted to the Ethernet when in local loopback.
21:16	Reserved	—	Reserved.
22	XOFF_GEN	RW	Pause frame generation. If this bit is set to 1, the IP core generates a pause frame with the pause quanta set to the value configured in the `pause_quant` register, independent of the receive FIFO status.
23	CNTL_FRM_ENA	RW	Receive control frame enable bit. ■ If this bit is set to 1, the receive control frames with any opcode other than `0x0001` are accepted and forwarded to the Avalon-ST interface. ■ If this bit is set to 0, the receive control frames with any opcode other than `0x0001` are discarded.
24	NO_LGTH_CHECK	RW	Receive payload length check disable. ■ If this bit is set to 0, the IP core checks the actual payload length of received frames against the length/type field in the received frames. ■ No checking is done if this bit is set to 1.
25	Reserved	—	Reserved.
26	FIFO_ERR_DISC	RW	Enable discard in FIFO. In store forward mode only, discards error and overflow frames if set to 1.

Table 3–17. Command_Config Register Bit Descriptions (Part 3 of 3)

Bit(s)	Bit Name	Access	Description
27	LINE_LOOP_ENA	RW	Line loopback enable. Set to 1 to loopback the receiver traffic onto the transmitter path. To get correct loopback frames, you should first stop transmitting data, put the IP core into line loopback mode and then resume transmission.
28	BROAD_FILTER_ ENA	RW	Broadcast filtering enable. 0 lets the receive broadcast frames through; 1 filters them out.
29	INS_CRC_ERR	RW	Insert one CRC error in the next transmit frame.To generate fault, set to 1. An error is inserted in the next packet or possibly the second packet after the bit is set due to latency to propagate the error. The bit self clears.
30	NO_PAUSE_FIFO	RW	Disable FIFO pause. When set to 1, the receiver FIFO buffer does not trigger the generation of pause frames on the transmitter path. Otherwise, pause frames are triggered depending on thresholds.
31	CNT_RESET	WC	Self-clearing counter reset command. Setting this bit to 1 clears the statistics counters. This bit is automatically cleared when the counter reset sequence is completed.

3.6.2. Software Reset

A software application can reset the IP core by setting the `SW_RESET` bit in the `command_config` register to 1. During a software reset, the IP core clears all statistics registers, flushes both the Tx and Rx FIFOs, and disables the transmitter and receiver by setting the `TX_ENA` and `RX_ENA` bits in the `command_config` register to 0.

The value of configuration registers, such as the address and FIFO thresholds are preserved. The `SW_RESET` bit is cleared automatically when the software reset ends. For more information about the reset signal, refer to "Command_Config Register" on page 3–38.

3.6.3. Statistics Block

The statistics block accumulates statistics required in *IEEE802.3 Basic, Mandatory and Recommended Management Information Packages, IEEE 802.3ah, Clause 30*.

In addition, the IP core provides all signals to generate the applicable objects of the *Management Information Base (MIB, MIB-II)* according to *IETF RFC2665 and Remote Network Monitoring (RMON)* according to *IETF RFC 2819 for SNMP Managed Environments*.

The statistics block implements the applicable parts of the *RMON (RFC 2819), MIB (RFC 3635)*, and *MIB (RFC 2863)*. The sections and tables below describe in detail the statistics counters that were listed in Table 3–16 on page 3–32 and the statistics group to which they belong.

3.6.4. IEEE 802.3 Management Packages

Table 3–18 lists the resources available to implement the IEEE 802.3 mandatory management packages defined in the *Ethernet Standard 802.3 Clause 30* for the managed objects `oMacEntity` and `oPauseEntity`. Table 3–19 describes objects and attributes only applicable to the IP core; other objects and attributes are derived from your application or higher layers.

Table 3–18. IEEE 802.3 oEntity and oPauseEntity Managed Object Support

IEEE802.3 Attribute	IEEE Management Packages			Description
	Basic	Mandatory	Recommended (Optional)	
oEntity Managed Object Support				
`aMACID`	X	—	—	The address.
`aFramesTransmittedOK`	—	X	—	Number of frames transmitted without error including pause frames.
`aFramesReceivedOK`	—	X	—	Number of frames received without error including pause frames.
`aFrameCheckSequence Errors`	—	X	—	Number of frames received with a CRC error.
`aAlignmentErrors`	—	X	—	Frame received with an alignment error.
`aOctetsTransmittedOK`	—	—	X	Sum of payload and padding octets of frames transmitted without error.
`aOctetsReceivedOK`	—	—	X	Sum of payload and padding octets of frames received without error.
oPauseEntity Managed Object Support				
`aPAUSEMACCtrlFrames Transmitted`	—	—	X	Number of transmitted pause frames.
`aPAUSEMACCtrlFrames Received`	—	—	X	Number of received pause frames.

For more information about attributes and objects, refer to *IEEE 802.3 Standard Clause 30*.

3.6.5. IETF Management Information Base—(MIB, MIB-II) Objects Support

The Internet Engineering Task Force (IETF) Request for Comments 2665 (RFC2665) defines the Management Information Base (MIB, MIB-II) objects for the Ethernet-like interface types. *RFC 2665* details the MIB (MIB-II) objects for Ethernet interfaces, which are defined in a more generic manner within *RFC 2863*.

Table 3–19. IETF MIB (MIB-II) Objects Support (Part 1 of 2)

MIB Object Name	Description
`ifInUcastPkts`	Number of valid received unicast frames.
`ifInMulticastPkts`	Number of valid received multicast frames (without pause).
`ifInBroadcastPkts`	Number of valid received broadcast frames.
`ifOutUcastPkts`	Number of valid transmitted unicast frames.
`ifOutMulticastPkts`	Number of valid transmitted multicast frames.
`ifOutBroadcastPkts`	Number of valid transmitted broadcast frames.

Table 3–19. IETF MIB (MIB-II) Objects Support (Part 2 of 2)

MIB Object Name	Description
`ifInErrors`	Number of frames received with one of the following errors: ■ FIFO overflow error ■ CRC error ■ Length error ■ Terminated frame with an error or error during a frame
`ifOutErrors`	Number of frames transmitted with one of the following errors: ■ FIFO overflow error ■ FIFO underflow error ■ User application defined error

3.6.5.1.IETF Remote Network Monitoring Support

The *IETF RFC 2819* defines objects for managing remote network monitoring devices. These objects are usually implemented in a dedicated device (monitor/probe) for traffic monitoring and analysis within a network segment. Such a probe usually samples the values in a periodic manner to give relative usage estimations rather than absolute values. Table 3–20 lists the defined objects. The remote monitoring (RMON) MIB counts good and bad packets, using the following definitions:

- Good packets (valid frames)—Good packets are error-free packets that have a valid frame length. A valid frame length is defined as between 64 bytes long and the value set in the `frm_length` register. The length does not include framing bits (preamble, SFD) but includes the FCS field.
- Bad packets (invalid frames)—Bad packets are packets that have proper framing and are therefore recognized as packets, but contain errors within the packet or have an invalid length. On the Ethernet, bad packets have a valid preamble and SFD, but have a bad CRC, or are either shorter than 64 bytes or longer than and the value set in the `frm_length` register, or contain a control character such as |E| in the frame.

Table 3–20. IEFT RMON MIB Object Support (Part 1 of 2)

MIB Object Name	Support
`etherStatsDropEvents`	Counts the number of dropped frames due to internal errors of the client. Occurs when FIFO overflow condition persists.
`etherStatsOctets`	Total number of bytes received. Good and bad frames.
`etherStatsPkts`	Total number of frames received. Counts good and bad frames.
`etherStatsUndersizePkts`	Number of frames received with less than 64 bytes, (good and bad frames are counted).
`etherStatsOversizePkts`	Incremented with each well-formed frame that exceeds the valid maximum programmed frame length, (good and bad frames are counted).
`etherStatsPkts64Octets`	Incremented when a frame of 64 bytes length is received (good and bad frames are counted).
`etherStatsPkts65to127Octets`	Frames (good and bad) with 65–127 bytes.
`etherStatsPkts128to255Octets`	Frames (good and bad) with 128–255 bytes.
`etherStatsPkts256to511Octets`	Frames (good and bad) with 256–511 bytes.

Table 3–20. IEFT RMON MIB Object Support (Part 2 of 2)

MIB Object Name	Support
`etherStatsPkts512to1023Octets`	Frames (good and bad) with 512–1023 bytes.
`etherStatsPkts1024to1518Octets`	Frames (good and bad) with 1024–1518 bytes.
`etherStatsPkts1519toXOctets`	Frames (good and bad) with 1519 to maximum frame size defined address.
`etherStatsJabbers`	Frame too long with CRC error.
`etherStatsFragments`	Frame too short with CRC error.

3.6.6. Software Derived MIB Objects

To extend the management information base, you can derive the following counters and objects using the management or driver software, based on the counters available.

Table 3–21 describes three derived IETF MIB (MIB-II) objects.

Table 3–21. Derived IETF MIB (MIB-II) Objects

MIB Object	Description
`ifInOctets`	Sum of bytes received except preamble (for example, header, payload, pad and FCS) of all valid received frames. = 18 × `aFramesReceivedOK` + `aOctetsReceivedOK`
`ifOutOctets`	Sum of bytes transmitted except preamble (for example, header, payload, pad and FCS) of all valid transmitted frames. = 18 × `aFramesTransmittedOK` + `aOctetsTransmittedOK`
`ifOutDiscards`	Number of frames with an error which was discarded in the FIFO. This number includes frames that would overflow the FIFO.

Table 3–22 describes three derived IETF RMON MIB objects.

Table 3–22. Derived IETF RMON MIB Objects

Object	Support
`etherStatsBroadcastPkts`	Any valid frame with Broadcast address: = `ifInBroadcastPkts`
`etherStatsMulticastPkts`	Any valid multicast frame, including pause frames: = `ifInMulticastPkts` + `aPAUSEMACCtrlFramesReceived`
`etherStatsCRCAlignErrors`	Incremented when frames of correct length but with CRC error are received: = `aFrameCheckSequenceErrors`

3.6.7. 64-Bit Statistics Counters

To access a 64-bit wide statistics counter, follow these steps:

1. Read the lower 32 bits. The upper 32 bits are cached.
2. When reading the upper 32 bits, the cached value is read.

If you perform a read access to an address other than the upper 32 bits after reading the lower part, the cached value may be lost. If an upper 64-bit register is read without accessing the lower 32 bits first, the stored value is given (not the cached value).

3.6.8. ECC Monitoring and Testing

The 10-Gbps Ethernet IP core ECC feature implements single-bit error correction and double-bit error detection (SECDED). The ECC management registers support ECC monitoring and testing by error injection. Single-event memory errors are infrequent, and extremely unlikely to occur during testing without error injection. The ECC management registers support software testing of the optional ECC feature, and also provide cumulative statistics counters for ECC-detected errors.

The following sections describe the ECC management registers and how the IP core implements ECC testing based on the values in these registers.

3.6.8.1. ECC Management Registers

If you turn on **ECC Protected RAMs** in the MegaWizard interface, and the MAC is instantiated, the ECC testing registers and ECC statistics counters are implemented in the 10-Gbps Ethernet IP core. Because the ECC management registers are available only if the MAC is instantiated, they are not available in Soft XAUI only mode. However, in Soft XAUI only mode, special input signals allow software to specify Soft XAUI error injection to the IP core. Therefore, in this case you can test the ECC feature, even though the IP core does not maintain ECC testing registers or cumulative statistics information.

The ECC statistics counters are cleared upon read (RC). Their implementation is based on an assumption that single-event errors are rare. The statistics registers might not provide an accurate count if two errors occur less than 10 clock cycles apart, or if two different blocks attempt to increment a register simultaneously.

Table 3–23 provides a memory map for the ECC management registers. Table 3–24 through Table 3–41 describe the registers that support software testing of the ECC feature, and Table 3–42 through Table 3–51 describe the ECC statistics counters.

Table 3–23. ECC Management Registers Memory Map (Part 1 of 2)

Address	Name	Expanded Name
0x350	`Reserved`	
0x354	`ECC_FIFO_INS`	FIFO Errors Insert
0x358	`ERR_FIFO_TX_DATA_ECC_0`	Tx FIFO Data Errors Word 0
0x35C	`ERR_FIFO_TX_DATA_ECC_1`	Tx FIFO Data Errors Word 1
0x360	`ERR_FIFO_TX_DATA_ECC_2`	Tx FIFO Data Errors Word 2
0x364	`ERR_FIFO_TX_CTRL_ECC`	Tx FIFO Control Errors
0x368	`ERR_FIFO_RX_DATA_ECC_0`	Rx FIFO Data Errors Word 0
0x36C	`ERR_FIFO_RX_DATA_ECC_1`	Rx FIFO Data Errors Word 1
0x370	`ERR_FIFO_RX_DATA_ECC_2`	Rx FIFO Data Errors Word 2
0x374	`ERR_FIFO_RX_CTRL_ECC`	Rx FIFO Control Errors
0x378	`Reserved`	
0x37C	`ECC_XAUI_INS`	Soft XAUI PCS Errors Insert

Table 3–23. ECC Management Registers Memory Map (Part 2 of 2)

Address	Name	Expanded Name
0x380	ERR_XAUI_DESKEW_CHAN0	XAUI Errors Deskew Lane 0
0x384	ERR_XAUI_DESKEW_CHAN1	XAUI Errors Deskew Lane 1
0x388	ERR_XAUI_DESKEW_CHAN2	XAUI Errors Deskew Lane 2
0x38C	ERR_XAUI_DESKEW_CHAN3	XAUI Errors Deskew Lane 3
0x390	ERR_XAUI_RATE_DATA_ECC_0	XAUI Rate FIFO Data Errors Word 0
0x394	ERR_XAUI_RATE_DATA_ECC_1	XAUI Rate FIFO Data Errors Word 1
0x398	ERR_XAUI_RATE_DATA_ECC_2	XAUI Rate FIFO Data Errors Word 2
0x39C	ERR_XAUI_RATE_CTRL_ECC	XAUI Rate FIFO Control Errors
0x3A0–0x3FC	Reserved	
0x400	ECC_COUNT_FIFO_TX_SBE	Recovered (corrected) Tx FIFO ECC Errors
0x404	ECC_COUNT_FIFO_TX_MBE	Fatal (detected but uncorrectable) Tx FIFO ECC Errors
0x408	ECC_COUNT_FIFO_RX_SBE	Recovered Rx FIFO ECC Errors
0x40C	ECC_COUNT_FIFO_RX_MBE	Fatal Rx FIFO ECC Errors
0x410	ECC_COUNT_XAUI_SBE	Recovered XAUI ECC Errors
0x414	ECC_COUNT_XAUI_MBE	Fatal XAUI ECC Errors
0x418	ECC_COUNT_TOTAL_SBE	Total Recovered (corrected) ECC Errors
0x41C	ECC_COUNT_TOTAL_MBE	Total Fatal (detected but uncorrectable) ECC Errors
0x420	FIFO_TX_DROP_COUNT	Tx FIFO ECC Packets Dropped
0x424	FIFO_RX_DROP_COUNT	Rx FIFO ECC Packets Dropped

Table 3–24. ECC_FIFO_INS—FIFO Errors Insert—Offset: 0x354

Bits	Access	Function	HW Reset Value
[31:4]	RW	Reserved.	0x0
[3]		Insert Rx control error. Clears after the error is inserted.	0
[2]		Insert Rx datapath error. Clears after the error is inserted.	0
[1]		Insert Tx control error. Clears after the error is inserted.	0
[0]		Insert Tx datapath error. Clears after the error is inserted.	0

Table 3–25. ERR_FIFO_TX_DATA_ECC_0—Tx FIFO Data Errors Word 0—Offset: 0x358

Bits	Access	Function	HW Reset Value
[31:0]	RW	Errors inserted in bits [31:0] of the Tx FIFO data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–26. ERR_FIFO_TX_DATA_ECC_1—Tx FIFO Data Errors Word 1—Offset: 0x35C

Bits	Access	Function	HW Reset Value
[31:0]	RW	Errors inserted in bits [63:32] of the Tx FIFO data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–27. ERR_FIFO_TX_DATA_ECC_2—Tx FIFO Data Errors Word 2—Offset: 0x360

Bits	Access	Function	HW Reset Value
[31:8]	RW	Reserved.	0x0
[7:0]		Errors inserted in bits [71:64] of the Tx FIFO data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–28. ERR_FIFO_TX_CTRL_ECC—Tx FIFO Control Errors—Offset: 0x364

Bits	Access	Function	HW Reset Value
[31:13]	RW	Reserved.	0x0
[12:0]		Errors inserted in the 13-bit Tx FIFO control+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–29. ERR_FIFO_RX_DATA_ECC_0—Rx FIFO Data Errors Word 0—Offset: 0x368

Bits	Access	Function	HW Reset Value
[31:0]	RW	Errors inserted in bits [31:0] of the Rx FIFO data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–30. ERR_FIFO_RX_DATA_ECC_1—Rx FIFO Data Errors Word 1—Offset: 0x36C

Bits	Access	Function	HW Reset Value
[31:0]	RW	Errors inserted in bits [63:32] of the Rx FIFO data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–31. ERR_FIFO_RX_DATA_ECC_2—Rx FIFO Data Errors Word 2—Offset: 0x370

Bits	Access	Function	HW Reset Value
[31:8]	RW	Reserved.	0x0
[7:0]		Errors inserted in bits [71:64] of the Rx FIFO data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–32. ERR_FIFO_RX_CTRL_ECC—Rx FIFO Control Errors—Offset: 0x374

Bits	Access	Function	HW Reset Value
[31:13]	RW	Reserved.	0x0
[12:0]		Errors inserted in the 13-bit Rx FIFO control+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–33. ECC_XAUI_INS—Soft XAUI PCS Errors Insert—Offset: 0x37C

Bits	Access	Function	HW Reset Value
[31:6]	RW	Reserved.	0x0
[5]		Insert rate-matching control error. Clears after the error is inserted.	0
[4]		Insert rate-matching datapath error. Clears after the error is inserted.	0
[3]		Insert deskew channel 3 error. Clears after the error is inserted.	0
[2]		Insert deskew channel 2 error. Clears after the error is inserted.	0
[1]		Insert deskew channel 1 error. Clears after the error is inserted.	0
[0]		Insert deskew channel 0 error. Clears after the error is inserted.	0

Table 3–34. ERR_XAUI_DESKEW_CHAN0—XAUI Errors Deskew Lane 0—Offset: 0x380

Bits	Access	Function	HW Reset Value
[31:28]	RW	Reserved.	0x0
[27:0]		Errors inserted in bits [27:0] of the deskew lane 0 data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–35. ERR_XAUI_DESKEW_CHAN1—XAUI Errors Deskew Lane 1—Offset: 0x384

Bits	Access	Function	HW Reset Value
[31:28]	RW	Reserved.	0x0
[27:0]		Errors inserted in bits [27:0] of the deskew lane 1 data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–36. ERR_XAUI_DESKEW_CHAN2—XAUI Errors Deskew Lane 2—Offset: 0x388

Bits	Access	Function	HW Reset Value
[31:28]	RW	Reserved.	0x0
[27:0]		Errors inserted in bits [27:0] of the deskew lane 2 data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–37. ERR_XAUI_DESKEW_CHAN3—XAUI Errors Deskew Lane 3—Offset: 0x38C

Bits	Access	Function	HW Reset Value
[31:28]	RW	Reserved.	0x0
[27:0]		Errors inserted in bits [27:0] of the deskew lane 3 data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–38. ERR_XAUI_RATE_DATA_ECC_0—XAUI Rate FIFO Data Errors Word 0—Offset: 0x390

Bits	Access	Function	HW Reset Value
[31:0]	RW	Errors inserted in bits [31:0] of the rate-matching FIFO data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–39. ERR_XAUI_RATE_DATA_ECC_1—XAUI Rate FIFO Data Errors Word 1—Offset: 0x394

Bits	Access	Function	HW Reset Value
[31:0]	RW	Errors inserted in bits [63:32] of the rate-matching FIFO data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–40. ERR_XAUI_RATE_DATA_ECC_2—XAUI Rate FIFO Data Errors Word 2—Offset: 0x398

Bits	Access	Function	HW Reset Value
[31:8]	RW	Reserved	0x0
[7:0]	RW	Errors inserted in bits [71:64] of the rate-matching FIFO data+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–41. ERR_XAUI_RATE_CTRL_ECC—XAUI Rate FIFO Control Errors—Offset: 0x39C

Bits	Access	Function	HW Reset Value
[31:24]	RW	Reserved	0x0
[23:0]	RW	Errors inserted in the 24-bit rate-matching FIFO control+ECC. For each bit position, the value 1 indicates an error is inserted, and the value 0 indicates no error is inserted.	0x0

Table 3–42. ECC_COUNT_FIFO_TX_SBE—Recovered Tx FIFO ECC Errors—Offset: 0x400

Bits	Access	Function	HW Reset Value
[31:8]	RC	Reserved	0x0
[7:0]		Number of single-bit errors detected and corrected in Tx FIFO. This register saturates at the value 255; after it reaches 255, it maintains this value until read. Reading this register resets it automatically.	0x0

Table 3–43. ECC_COUNT_FIFO_TX_MBE—Fatal Tx FIFO ECC Errors—Offset: 0x404

Bits	Access	Function	HW Reset Value
[31:8]	RC	Reserved	0x0
[7:0]		Number of multiple-bit errors detected in Tx FIFO. These errors are not corrected. This register saturates at the value 255; after it reaches 255, it maintains this value until read. Reading this register resets it automatically.	0x0

Table 3–44. ECC_COUNT_FIFO_RX_SBE—Recovered Rx FIFO ECC Errors—Offset: 0x408

Bits	Access	Function	HW Reset Value
[31:8]	RC	Reserved	0x0
[7:0]		Number of single-bit errors detected and corrected in Rx FIFO. This register saturates at the value 255; after it reaches 255, it maintains this value until read. Reading this register resets it automatically.	0x0

Table 3–45. ECC_COUNT_FIFO_RX_MBE—Fatal Rx FIFO ECC Errors—Offset: 0x40C

Bits	Access	Function	HW Reset Value
[31:8]	RC	Reserved	0x0
[7:0]		Number of multiple-bit errors detected in Rx FIFO. These errors are not corrected. This register saturates at the value 255; after it reaches 255, it maintains this value until read. Reading this register resets it automatically.	0x0

Table 3–46. ECC_COUNT_XAUI_SBE—Recovered XAUI ECC Errors—Offset: 0x410

Bits	Access	Function	HW Reset Value
[31:8]	RC	Reserved	0x0
[7:0]		Number of single-bit errors detected and corrected in Soft XAUI PCS. This register saturates at the value 255; after it reaches 255, it maintains this value until read. Reading this register resets it automatically.	0x0

Table 3–47. ECC_COUNT_XAUI_MBE—Fatal XAUI ECC Errors—Offset: 0x414

Bits	Access	Function	HW Reset Value
[31:8]	RC	Reserved	0x0
[7:0]		Number of multiple-bit errors detected in Soft XAUI PCS. These errors are not corrected. This register saturates at the value 255; after it reaches 255, it maintains this value until read. Reading this register resets it automatically.	0x0

Table 3–48. ECC_COUNT_TOTAL_SBE—Total Recovered ECC Errors—Offset: 0x418

Bits	Access	Function	HW Reset Value
[31:8]	RC	Reserved	0x0
[7:0]		Total number of single-bit errors detected and corrected. This register saturates at the value 255; after it reaches 255, it maintains this value until read. Reading this register resets it automatically.	0x0

Table 3–49. ECC_COUNT_TOTAL_MBE—Total Fatal ECC Errors—Offset: 0x41C

Bits	Access	Function	HW Reset Value
[31:8]	RC	Reserved	0x0
[7:0]		Total number of multiple-bit errors detected. These errors are not corrected. This register saturates at the value 255; after it reaches 255, it maintains this value until read. Reading this register resets it automatically.	0x0

Table 3–50. FIFO_TX_DROP_COUNT—Tx FIFO ECC Packets Dropped—Offset: 0x420

Bits	Access	Function	HW Reset Value
[31:8]	RC	Reserved	0x0
[7:0]		Number of ECC-detected packets dropped from Tx FIFO. This register saturates at the value 255; after it reaches 255, it maintains this value until read. Reading this register resets it automatically.	0x0

Table 3–51. FIFO_RX_DROP_COUNT—Rx FIFO ECC Packets Dropped—Offset: 0x424

Bits	Access	Function	HW Reset Value
[31:8]	RC	Reserved	0x0
[7:0]		Number of ECC-detected packets dropped from Rx FIFO. This register saturates at the value 255; after it reaches 255, it maintains this value until read. Reading this register resets it automatically.	0x0

3.6.9. ECC Testing

This section describes how to test the ECC feature. This section assumes the following definitions:

- Error mask registers—Includes all the ECC management registers except for the `ECC_FIFO_INS`, `ECC_XAUI_INS`, and statistics registers.
- Error insertion registers—Includes the `ECC_FIFO_INS` and `ECC_XAUI_INS` registers.

To test the ECC feature in your 10-Gbps Ethernet IP core, perform the following steps:

1. Reset the ECC management registers. All registers reset to the value 0.
2. Write values to the error mask registers to indicate the bit errors you wish to insert.
3. Set the appropriate bits in the error insertion registers.
4. Run a test sequence that includes sending a packet.
5. Monitor the `ecc_sbe`, `ecc_mbe`, and `ecc_packet_dropped` signals to ensure the IP core detected the inserted errors correctly.
6. Repeat steps 2 to 4 until you are satisfied you have verified the ECC implementation.

When you enable the insertion of a bit error by performing steps 2 and 3 in the preceding instructions, the IP core implements error insertion differently for the Rx and Tx FIFOs than for the Soft XAUI PCS FIFOs. In the case of the Rx and Tx FIFOs, if you enable insertion during an idle cycle, the IP core inserts the error on the first cycle of the next packet written to the FIFO. In the case of the Soft XAUI PCS FIFOs, because data is written to the FIFOs continuously, the IP core inserts errors when requested. In both cases, the error insertion bit clears within a few clock cycles after the error is inserted.

Inserting a bit error in a Soft XAUI PCS FIFO affects two or three columns, which could be in two different clock cycles.

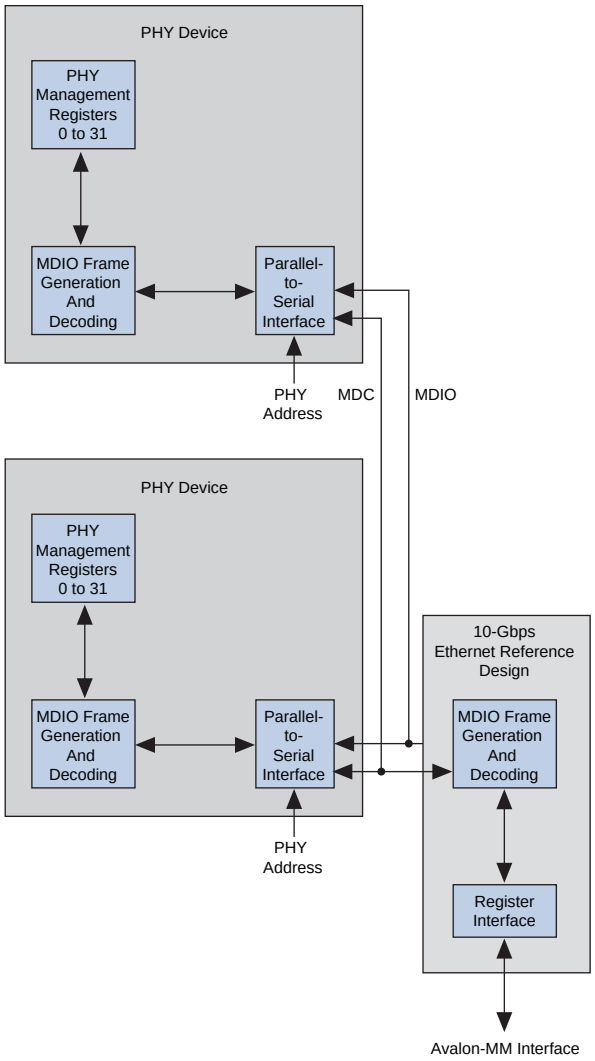
The Soft XAUI PCS rate-matching FIFO might skip a write during rate compensation. In that case, a bit error is inserted in the following write. A bit error inserted during an idle cycle might cause the MAC to ignore the next packet or insert an error in the previous packet. The Soft XAUI PCS cannot signal a dropped packet using the `ecc_packet_dropped` signal.

3.6.10. MDIO External PHY Management Interface

The MDIO interface is a two-wire standard management interface that implements a standardized method to access the external PHY device management registers. The IP core MDIO interface supports *IEEE 802.3 Clause 22* (see Figure 3–28) and *Clause 45*.

The PHY device MDIO registers are mapped in the register space and can be read and written from the Avalon-MM interface. The IP core provides the flexibility to access PHY devices with the MDIO address set to any legal value.

Figure 3–28. MDIO Interface (Clause 22)



3.6.10.1.MDIO Frame Format (Clause 22)

The MDIO master controller communicates with the slave PHY device using frames. A complete frame is 64-bits long and consists of 32-bit preamble, 14-bit command, 2-bit bus direction change, and 16-bit data. Each bit is transferred on the rising edge of the MDIO clock (MDC). The PHY management interface supports the standard MDIO specification (*IEEE803.2 Clause 22*). Table 3–52 illustrates the MDIO formats.

Table 3–52. MDIO Frame Formats (Read/Write)

Type	Command							
	PRE	ST	OP	PHYAD	REGAD	TA	Data	Idle
Read	1 ... 1	01	10	AAAAA	RRRRR	Z0	DDDDDDDDDDDDDDDD	Z
Write	1 ... 1	01	01	AAAAA	RRRRR	10	DDDDDDDDDDDDDDDD	Z

Table 3–53 describes the fields of the MDIO frame (Clause 22).

Table 3–53. MDIO Frame Field Descriptions—Clause 22

Name	Description
PRE	Preamble. 32 bits of logical 1 sent prior to every transaction.
ST	Start indication. Standard MDIO (Clause 22): `0b01`.
OP	The opcode defines whether a read or write operation is performed: ■ `0b10`: a read operation is performed. ■ `0b01`: a write operation is performed.
PHYAD	The PHY device address (PHYAD). Up to 32 devices can be addressed. For PHY device 0, the `Addr1` field is set to the value configured in the `mdio_addr0` register.
REDAD	Register address. Each PHY can have up to 32 registers.
TA	Turnaround time. Two bit times are reserved for read operations to switch the data bus from write to read for read operations. The PHY device presents its register contents in the data phase and drives the bus from the 2nd bit of the turnaround phase.
`Data`	16-bit data written to or read from the PHY device.
`Idle`	Between frames, the MDIO data signal is tristated.

3.6.10.2.MDIO Frame Format (Clause 45)

The MDIO master controller communicates with the slave PHY device using frames. A complete frame is 64-bits long and consists of 32-bit preamble, 14-bit command, 2-bit bus direction change, and 16-bit data. Each bit is transferred on the rising edge of the MDIO clock (MDC). The PHY management interface supports the standard MDIO specification (*IEEE803.2 Clause 45*). Table 3–54 describes the Clause 45 formats.

Table 3–54. MDIO Frame Formats—Clause 45

Type	Command							
	PRE	ST	OP	PRTAD	DEVAD	TA	Address/Data	Idle
Address	1 ... 1	00	00	PPPPP	EEEEE	10	AAAAAAAAAAAAAAAA	Z
Write	1 ... 1	00	01	PPPPP	EEEEE	10	DDDDDDDDDDDDDDDD	Z
Read	1 ... 1	00	11	PPPPP	EEEEE	Z0	DDDDDDDDDDDDDDDD	Z

Table 3–55 describes the fields of the MDIO frame (Clause 45).

Table 3–55. MDIO Frame Field Descriptions—Clause 45

Name	Description
PRE	Preamble. 32 bits of logical 1 sent prior to every transaction.
ST	The start of frame for indirect access cycles is indicated by the <00> pattern. This pattern assures a transition from the default one and identifies the frame as an indirect access. Frames that contain the ST=01 pattern defined in Clause 22 are ignored by the devices specified in Clause 45.
OP	The operation code field indicates the type of transaction being performed by the frame. ■ 00 indicates that the frame payload contains the address of the register to access. ■ 01 indicates that the frame payload contains data to be written to the register whose address was provided in the previous address frame. ■ 11 indicates that the frame is a read operation.
PRTAD	The port address (PRTAD). The port address is 5 bits, allowing 32 unique port addresses. The first port address bit to be transmitted and received is the MSB of the address. A station management entity must have a prior knowledge of the appropriate port address for each port to which it is attached, whether connected to a single port or to multiple ports.
DEVAD	The device address is 5 bits, allowing 32 unique MDIO manageable devices (MMDs) per port. The first device address bit transmitted and received is the MSB of the address.
TA	The turnaround time is a 2-bit time spacing between the device address field and the data field of a management frame to avoid contention during a read transaction. For a read or post-read-increment-address transaction, both the station management entity (STA) and the MMD remain in a high-impedance state for the first bit time of the turnaround. The MMD drives a 0 during the second bit time of the turnaround of a read or postread-increment-address transaction. During a write or address transaction, the STA drives a 1 for the first bit time of the turnaround and a 0 for the second bit time of the turnaround.
Address/ Data	The address/data field is 16 bits. For an address cycle, it contains the address of the register to be accessed on the next cycle. For the data cycle of a write frame, the field contains the data to be written to the register. For a read or post-read-increment-address frame, the field contains the contents of the register. The first bit transmitted and received is bit 15.
Idle	The idle condition on MDIO is a high-impedance state. All three state drivers are disabled and the MMDs pullup resistor pulls the MDIO line to a one.

3.6.10.3.MDIO Registers

The host processor can access the MDIO registers of a PHY device via an Avalon-MM interface which are mapped in the address space. Each PHY device has 32 registers.

The IP core supports both Clause 22 and Clause 45, but electrical restrictions require you to choose a single clause for your design.

For Clause 22, follow these steps:

1. Set up the `mdio_addr0` register at address `0x03C`, where:
   - Bits 31:5 are unused
   - Bits 4:0 are `PHYAD`
2. Read or write to addresses `0x200` to `0x27C` for direct access to the 32 register addresses present (`REGAD`).

For Clause 45, follow these steps:

1. Set up the `mdio_addr0` register at address `0x03C`, where:
   - Bits 31:16 are the `Address`
   - Bits 12:8 are the `PRTAD`
   - Bits 4:0 are the `DEVAD`
2. Read or write the register of the device of the port selected by reading or writing to address `0x320`.

Post-read increment is not supported.

3.6.10.4. MDIO Clock Generation

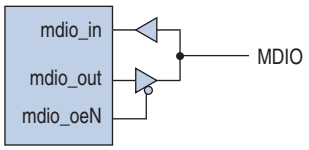
The management data clock (MDC) is generated from the Avalon-MM interface clock signal, `clk`.

The division factor must be defined such that the MDC frequency does not exceed 2.5 MHz.

3.6.10.5. MDIO Buffer Connection

Figure 3–29 illustrates the buffers you can implement for the MDIO tristate bus.

Figure 3–29. MDIO Buffer Connection



3.7. 10-Gbps Ethernet PHY

This section provides a high-level description of various PHY options.

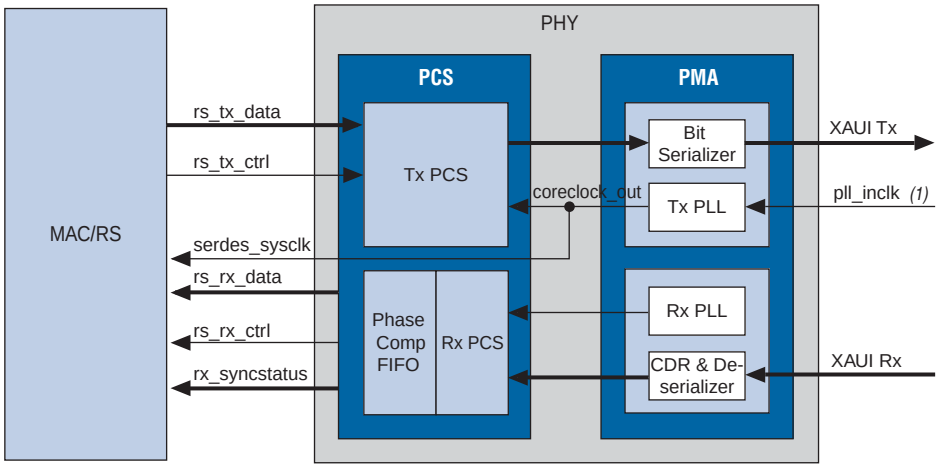
3.7.1. 10GBASE-X PHY

You can use the 10GBASE-X PHY to connect a 10-Gbps Ethernet MAC to the physical media over a four-lane XAUI electrical interface running at 3.125 Gbps. This PHY consists of a 10GBASE-X PCS and PMA. Figure 3–30 illustrates this configuration. As this figure shows, the PHY consists of two main parts: a PMA that interfaces to physical media and the PCS that handles the encoding and decoding necessary to transport the 10-Gbps Ethernet data on and off the device.

The Altera PHY includes the Altera Transceiver (ALTGXB or ALT2GXB) that includes the PMA, a selectable hard or soft PCS, and a reset controller block. The reset control block performs the reset sequence recommended for the ALTGXB megafunction.

For more information on reset refer to the *Reset Control and Power Down* chapter in Volume 2 of the *Stratix IV Device Handbook*, the "Reset Control and Power Down" section in Section I of the *Stratix II GX Transceiver User Guide*, or *Reset Control and Power Down* in Volume 2 of the *Arria II GX Device Handbook*.

Figure 3–30. 10GBASE-X PHY/Ethernet MAC Block Diagram



Note to Figure 3–30:

(1) This clock is the reference clock. It is called `pll_inclk` in the PHY (xaui_10g) module, but is called `sysclk` in the top-level design.

When you select a PHY that includes a hard PCS and PMA, the PHY provides `sysclk` to the MAC. A PLL in Tx PMA generates `sysclk`. The 64-bit SDR XGMII interface between MAC and PHY is synchronous to this clock. The PCS drives a set of signals, including `rx_syncstatus`, that reflect the status of the PCS and PMA. Their status is recorded in the `ALTGX status0` and `ALTGX status1` registers.

Table 3–56 describes the signals of the XAUI interface that interface to the physical media. (For more information about the MAC PHY interface, refer to "MAC Functional Description" on page 3–2.)

Table 3–56. Top-Level Transceiver Signals (Part 1 of 2)

Signal Name	Dir	Description
`cal_blk_clk`	I	Calibration clock. This clock runs the calibration block for the analog blocks. The frequency of this clock should be between 10–125 MHz.
`xaui_tx_data[3:0]`	O	Tx XAUI link.
`xaui_rx_data[3:0]`	I	Rx XAUI link.
`serdes_reset_n`	I	Reset for the PHY.
`serdes_sysclk`	O	Clock synchronous with the MAC-PHY 64-bit SDR XGMII interface. This clock also serves as the `sysclk` for the MAC.
`reconfig_clk`	I	Clock used for the reconfiguration block. The reconfig dynamically configures the PMA to change signal characteristics. More details can be found in appropriate device handbook. This input is optional for Stratix II GX but is required for Stratix IV GX and Arria II GX. The frequency range of this clock is 37.5–50 MHz for XAUI because it requires a Tx/Rx dual-mode transceiver channel.
`reconfig_togxb[<n>:0]` *(Note 1)*	I	3-bit configuration input for the PMA. The width depends on the device and also whether a soft or hard XAUI PCS has been selected.

Table 3–56. Top-Level Transceiver Signals (Part 2 of 2)

Signal Name	Dir	Description
`reconfig_fromgxb[<n>:0]` *(Note 2)*	O	1-bit output from PMA. The width depends on the device and also whether soft or hard XAUI PCS has been selected.

Note to Table 3–56:

(1) <*n*> = 3 for Stratix II GX, 68 for Stratix IV GX in the soft XAUI implementation, and 17 for the hard XAUI implementation.

(2) <n> = 3 for Stratix II GX, 4 for Arria II GX and Stratix IV GX.

For additional details about the transceiver signals, refer to the "Stratix II GX ALT2GXB Ports List" in the *Stratix II GX Transceiver User Guide*, the "Transceiver Port List" in Volume 2 of the *Stratix IV GX Device Handbook*, or the "Transceiver Port List" in Volume 2 of the *Arria II GX Device Handbook*.

3.7.2. PMA Reconfiguration

For Stratix II GX devices with XAUI, if you turn on **Use external reconfiguration block**, you can modify analog properties of the transceiver with an external reconfiguration block (ALT2GXB_RECONFIG). For Stratix IV or Arria II GX devices using XAUI, you must include an external reconfiguration block, so this option is always on. For Arria GX devices, this option is not available. When you use an external reconfiguration block, you must select the starting channel number.

For more information on the ALT2GXB_RECONFIG megafunction, refer to the *Stratix II GX ALT2GXB_RECONFIG Megafunction User Guide*. For more information about the ALTGX_RECONFIG megafunction refer to *AN 558: Implementing Dynamic Reconfiguration in Arria II GX Devices*.

The XAUI reconfiguration instantiates the transceiver reconfiguration block. This block converts Avalon-MM instructions into a format supported by the transceiver configuration module. In this design example, the relationship between the system clock and the reconfiguration clock is critical.

3.7.2.1. PHY Loopback

The 10-Gbps Ethernet IP core does not provide a loopback at the PMA interface. If this feature is critical to your design, you can create a **MAC only** option and hard XAUI block separately. The soft XAUI block does not support loopback at the PMA interface.

3.7.2.2. Reset Controller

The XAUI PHY architecture requires the various resets to be asserted in a specific sequence. Refer to the device handbook for your device for details. The 10-Gbps Ethernet IP core provides an example of correct reset sequencing.

3.7.3. External PHYs

If you chose the **MAC only** 10-Gbps Ethernet variation, you can instantiate either a PCS Base-X that you have developed or a Base-R PCS in your design and connect this PCS to the MAC that Altera provides.

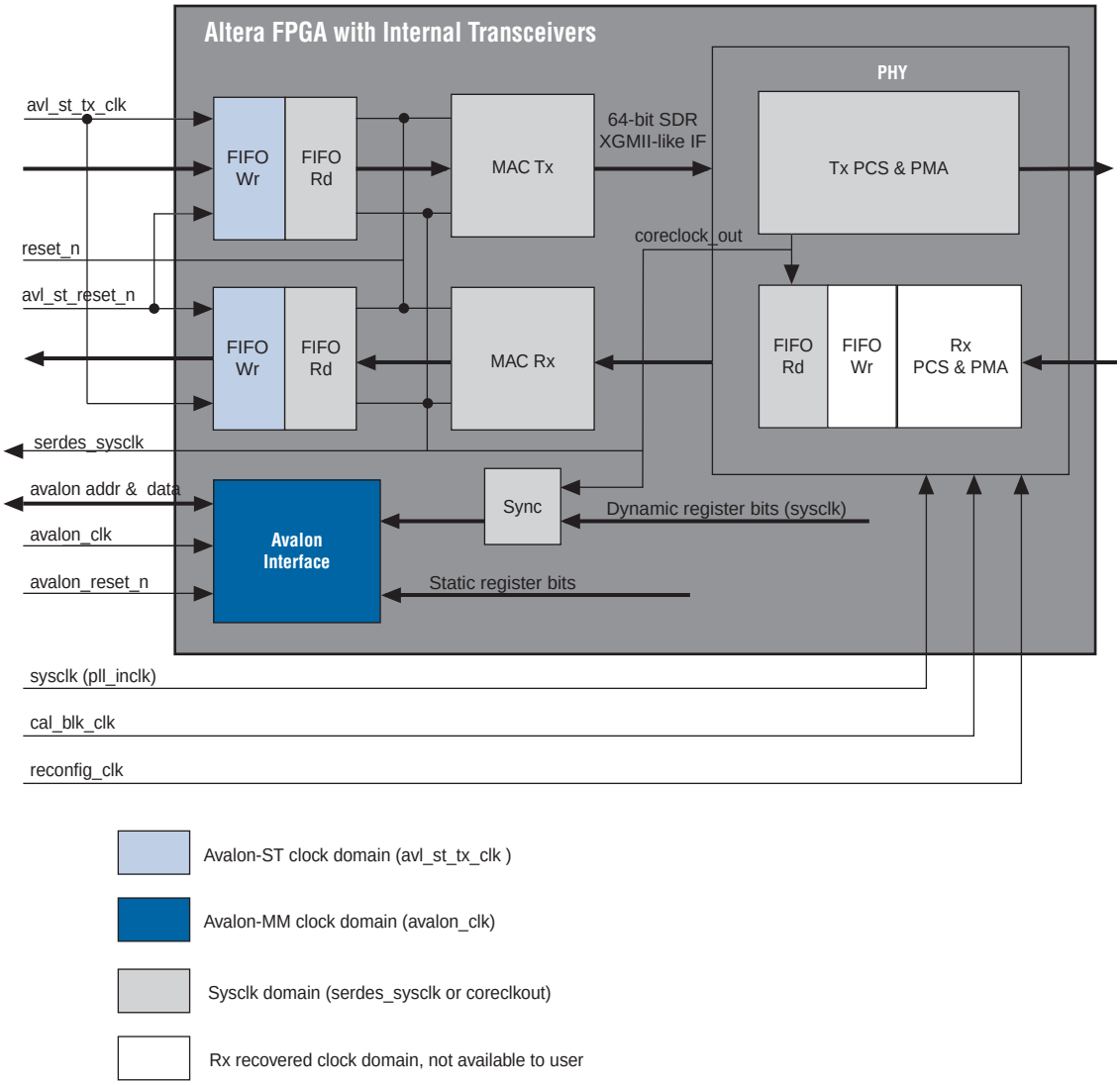
3.8. Clocks and Reset

The clocking depends upon the variant you specify. The IP core includes at least three clock domains. A fourth clock domain controls the ALTGX_RECONFIG megafunction. The following sections illustrate the clock domains for the different configurations.

3.8.1. MAC, PHY Resets and Clocks

In this configuration, the PHY provides a clock that can be used for MAC and MAC side of the FIFO logic, as Figure 3–31 illustrates.

Figure 3–31. MAC, PHY Resets and Clocks

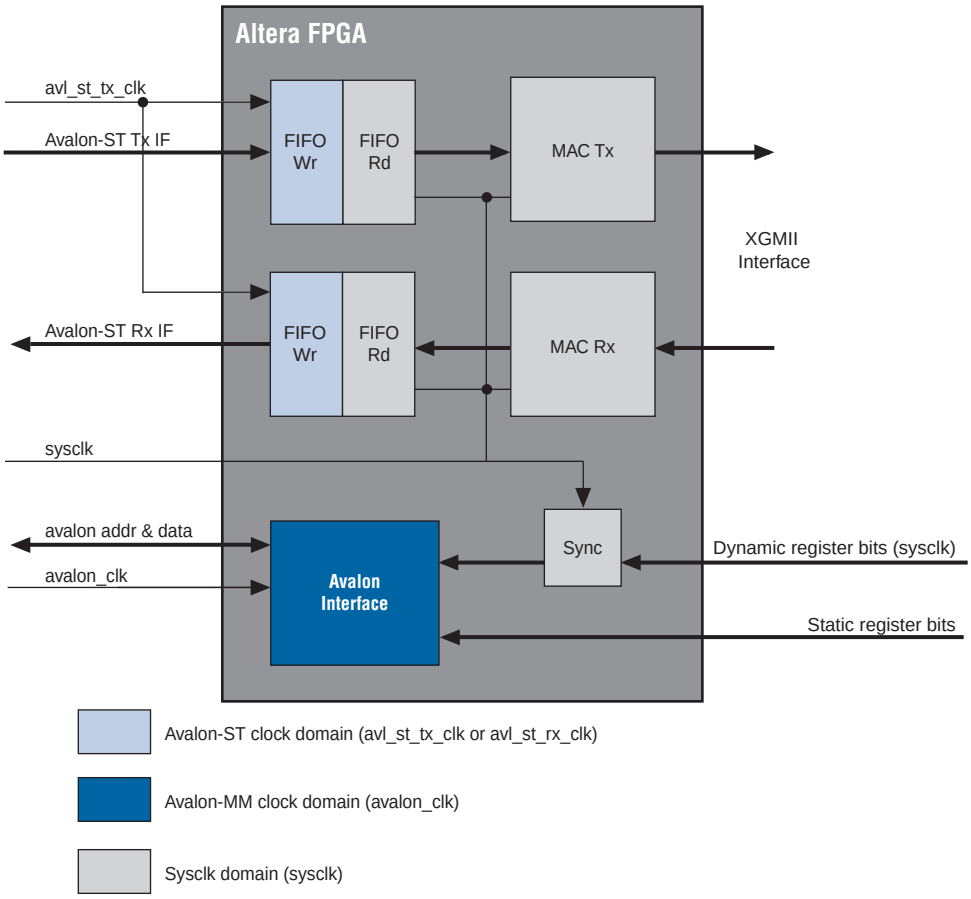


3.8.2. MAC Only Clocks

In this configuration the application logic provides the clock for the MAC and MAC side of the FIFO logic.

The reset connections for Figure 3–32 are the same as in Figure 3–31. The are omitted in Figure 3–32 for clarity.

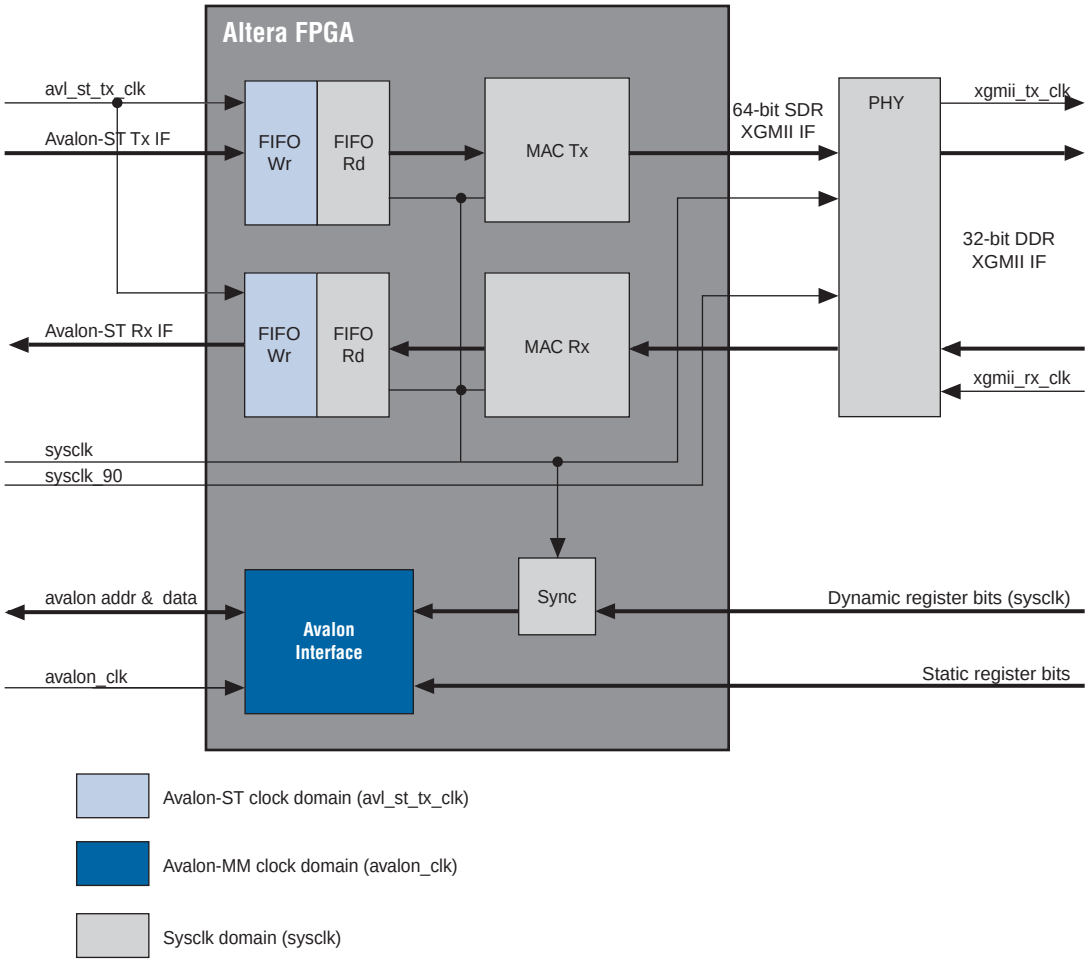
Figure 3–32. MAC Only Clocks



3.8.3. MAC XGMII Clocks

In this configuration the application logic provides the system clock for the MAC, MAC side of the FIFO logic, and the XGMII transmit data and clock interface as Figure 3–33 illustrates.

Figure 3–33. MAC XGMII Clocks



3.8.4. Clock Signals

Table 3–57 describes the signals that comprise the clock interface for the various configurations.

Table 3–57. Clock Signals (Part 1 of 2)

Signal	Description
`sysclk`	156.25 MHz system clock for the state machines and datapath. Can be shared with other 10 Gbps Ethernet IP cores in the same device. You must provide this clock.
`sysclk_90`	The `sysclk` phase-shifted by 90 degrees. This clock ensures that the transmitter clock and the Tx data are 90 degrees apart in an XGMII interface. In your design, this clock must be derived from the `sysclk` or form the same source as `sysclk`.

Table 3–57. Clock Signals (Part 2 of 2)

Signal	Description
`avalon_clk`	The Avalon-MM clock that controls the Avalon-MM interface used for statistics and configuration. This clock can be shared with other modules, an SOPC system, or can be tied to the system clock.
`avl_st_clk`	Avalon-ST clock input for the datapath, which can be different if a FIFO is present. This clock can be tied to the system clock.
`xgmii_rx_clk`	Captures the data arriving on the XGMII Rx interface, a PLL and local clock are required. The incoming clock is shifted by 90 degrees to capture the data.
`xgmii_tx_clk`	Clock that accompanies `xgmii_tx_data` and `xgmii_tx_ctrl.` It is shifted by 90° with respect to `xgmii_tx_data` and `xgmii_tx_ctrl`.
`serdes_sysclk`	XAUI clock. When using the XAUI block, the transceiver module has its own PLL to derive the required output clocks. You can feed both the Tx and Rx data clock, which are connected to the system clock, and the clock domain crossing from network clock to system clock occurs within the transceiver block.
`phy_mdc`	MDIO clock. The frequency of the MDIO must be less than 2.5 MHz and is derived from the Avalon-MM clock. the MDIO clock cannot be shared between modules.
`cal_blk_clk`	SERDES calibration clock. The frequency of this clock should be between 10–125 MHz.

For the definition of `reconfig_clk`, refer to "Top-Level Transceiver Signals" on page 3–55.

Copyright © 2011 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

I.S. EN ISO 9001