# Triple-Speed Ethernet MegaCore Function

# User Guide

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

Document last updated for Altera Complete Design Suite version: 13.0
Document publication date: May 2013

Feedback Subscribe

UG-01008-4.0

# Contents

## Chapter 1. About This MegaCore Function



## Chapter 2. Getting Started with Altera IP Cores



## Chapter 3. Parameter Settings



## Chapter 4. Functional Description

## Appendix A. MegaCore Evaluation



## Appendix B. Ethernet Frame Format



## Appendix C. Simulation Parameters



## Appendix D. Time-of-Day Clock



## Appendix E. ToD Synchronizer



## Appendix F. Packet Classifier



## Additional Information

# 1. About This MegaCore Function

The Altera® Triple-Speed Ethernet MegaCore® function is a configurable IP core that complies with the IEEE 802.3 standard. The IP core was tested and successfully validated by the University of New Hampshire (UNH) interoperability lab. It combines the features of a 10/100/1000-Mbps Ethernet media access controller (MAC) and 1000BASE-X/SGMII physical coding sublayer (PCS) with an optional physical medium attachment (PMA).

## Device Family Support

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

**Table 1–1.** Altera IP Core Device Support Levels

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

Table 1–2 shows the level of support offered by the Triple-Speed Ethernet MegaCore function for each Altera device family.

**Table 1–2.** Device Family Support (Part 1 of 2)

| Device Family | Support |
|---|---|
| Arria® II (GX/GZ) | Final |
| Arria GX | Final |
| Arria V | Refer to the *What's New in Altera IP* page of the Altera website. |
| Cyclone II® | Final |
| Cyclone III | Final |
| Cyclone III LS | Final |
| Cyclone IV (E/GX) | Final |
| Cyclone V | Refer to the *What's New in Altera IP* page of the Altera website. |
| HardCopy® II | Compilation |
| HardCopy III | Compilation |
| HardCopy IV | Compilation |
| Stratix® II | Final |
| Stratix II GX | Final |

**Table 1–2.** Device Family Support (Part 2 of 2)

| Device Family | Support |
|---|---|
| Stratix III | Final |
| Stratix IV | Final |
| Stratix IV GT | Final |
| Stratix V | Refer to the *What's New in Altera IP* page of the Altera website. |
| Other device families | No support |

# Features

- Complete triple-speed Ethernet IP: 10/100/1000-Mbps Ethernet MAC, 1000BASE-X/SGMII PCS, and embedded PMA.
- Successful validation from the University of New Hampshire (UNH) InterOperability Lab.
- 10/100/1000-Mbps Ethernet MAC features:
  - Multiple variations: 10/100/1000-Mbps Ethernet MAC in full duplex, 10/100-Mbps Ethernet MAC in half duplex, 10/100-Mbps or 1000-Mbps small MAC (resource-efficient variant), and multiport MAC that supports up to 24 ports.
  - Support for basic, VLAN, stacked VLAN, and jumbo Ethernet frames. Also supports control frames including pause frames.
  - Optional internal FIFO buffers, depth from 64 bytes to 256 Kbytes.
- Optional statistics counters.1000BASE-X/SGMII PCS features:
  - Compliance with Clause 36 of the IEEE standard 802.3.
  - Optional embedded PMA implemented with serial transceiver or LVDS I/O and soft CDR in Altera devices that support this interface at 1.25-Gbps data rate.
  - Support for auto-negotiation as defined in Clause 37.
  - Support for connection to 1000BASE-X PHYs. Support for 10BASE-T, 100BASE-T, and 1000BASE-T PHYs if the PHYs support SGMII.
- MAC interfaces:
  - Client side—8-bit or 32-bit Avalon® Streaming (Avalon-ST)
  - Network side—medium independent interface (MII), gigabit medium independent interface (GMII), or reduced gigabit medium independent interface (RGMII) on the network side. Optional loopback on these interfaces.
  - Optional management data I/O (MDIO) master interface for PHY device management.

- PCS interfaces:
  - Client side—MII or GMII
  - Network side—ten-bit interface (TBI) for PCS without PMA; 1.25-Gbps serial interface for PCS with PMA implemented with serial transceiver or LVDS I/O and soft CDR in Altera devices that support this interface at 1.25-Gbps data rate.
- Programmable features via 32-bit configuration registers:
  - FIFO buffer thresholds.
  - Pause quanta for flow control.
  - Source and destination MAC addresses.
  - Address filtering on receive, up to 5 unicast and 64 multicast MAC addresses.
  - Promiscuous mode—receive frame filtering is disabled in this mode.
  - Frame length—in MAC only variation, up to 64 Kbytes including jumbo frames. In all variants containing 1000BASE-X/SGMII PCS, the frame length is up to 10 Kbytes.
  - Optional auto-negotiation for the 1000BASE-X/SGMII PCS.
- Optional IEEE 1588v2 feature for 1000-Mbps Ethernet MAC with SGMII PCS and embedded serial PMA variation operating without internal FIFO in full-duplex mode.

# 10/100/1000 Ethernet MAC Versus Small MAC

Table 1–3 compares the features of the 10/100/1000 Ethernet MAC with the small MAC.

**Table 1–3.** Feature Comparison between 10/100/1000 Ethernet MAC and Small MAC

| Feature | 10/100/1000 Ethernet MAC | Small MAC |
|---|---|---|
| Speed | Triple speed (10/100/1000 Mbps) | 10/100 Mbps or 1000 Mbps |
| External interfaces | MII/GMII or RGMII | MII only for 10/100 Mbps small MAC, GMII or RGMII for 1000 Mbps small MAC |
| Control interface registers | Fully programmable | Limited programmable options. The following options are fixed:<br>■ Maximum frame length is fixed to 1518. Jumbo frames are not supported.<br>■ FIFO buffer thresholds are set to fixed values.<br>■ Store and forward option is not available.<br>■ Interpacket gap is set to 12.<br>■ Flow control is not supported; pause quanta is not in use.<br>■ Checking of payload length is disabled.<br>■ Supplementary MAC addresses are disabled.<br>■ Padding removal is disabled.<br>■ Sleep mode and magic packet detection is not supported. |
| Synthesis options | Fully configurable | Limited configurable options. The following options are NOT available:<br>■ Flow control<br>■ VLAN<br>■ Statistics counters<br>■ Multicast hash table<br>■ Loopback<br>■ TBI and 1.25 Gbps serial interface<br>■ 8-bit wide FIFO buffers |

# High-Level Block Diagrams

Figure 1–1 to Figure 1–4 show the high-level block diagrams of the different variations of the Triple-Speed Ethernet MegaCore function.

**Figure 1–1.** 10/100/1000-Mbps Ethernet MAC

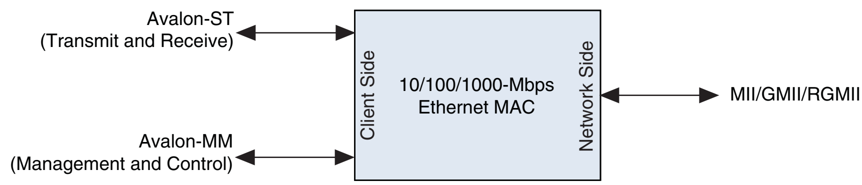



**Figure 1–2.** Multi-port MAC

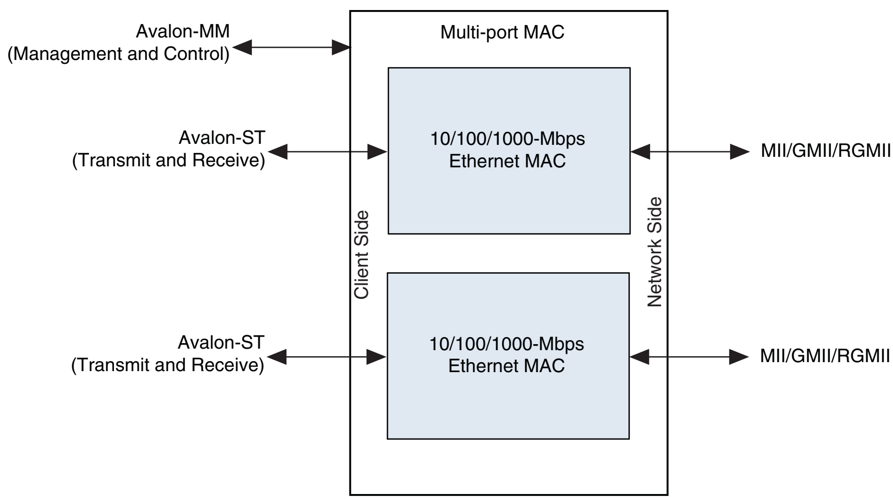



**Figure 1–3.** 10/100/1000-Ethernet MAC and 1000BASE-X/SGMII PCS with Optional PMA

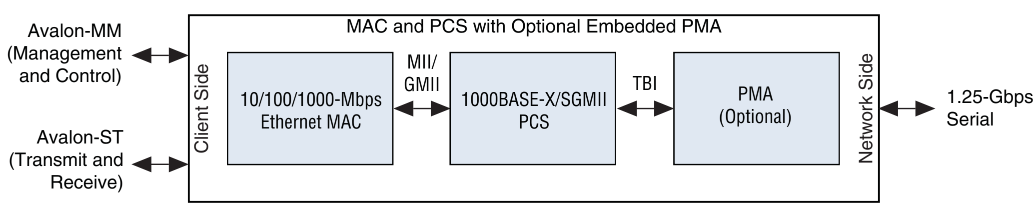



**Figure 1–4.** 1000BASE-X/SGMII PCS with Optional PMA

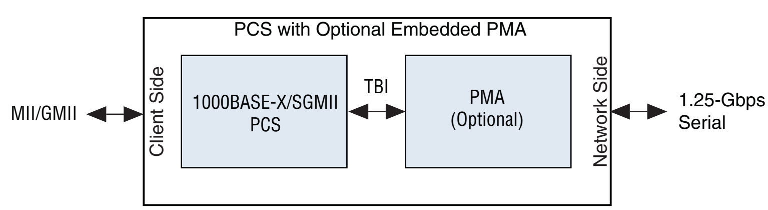
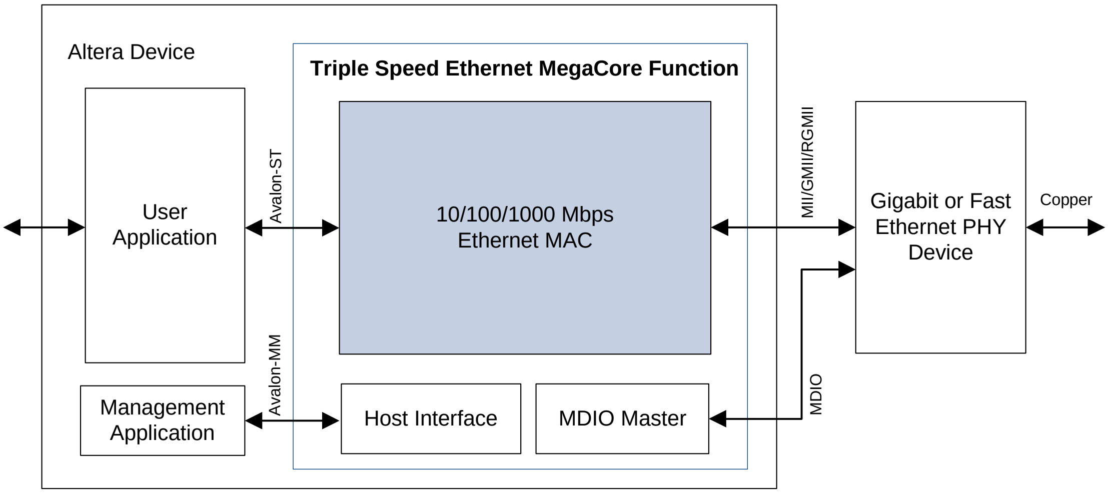


**Figure 1–5.** Stand-Alone 10/100/1000 Mbps Ethernet MAC

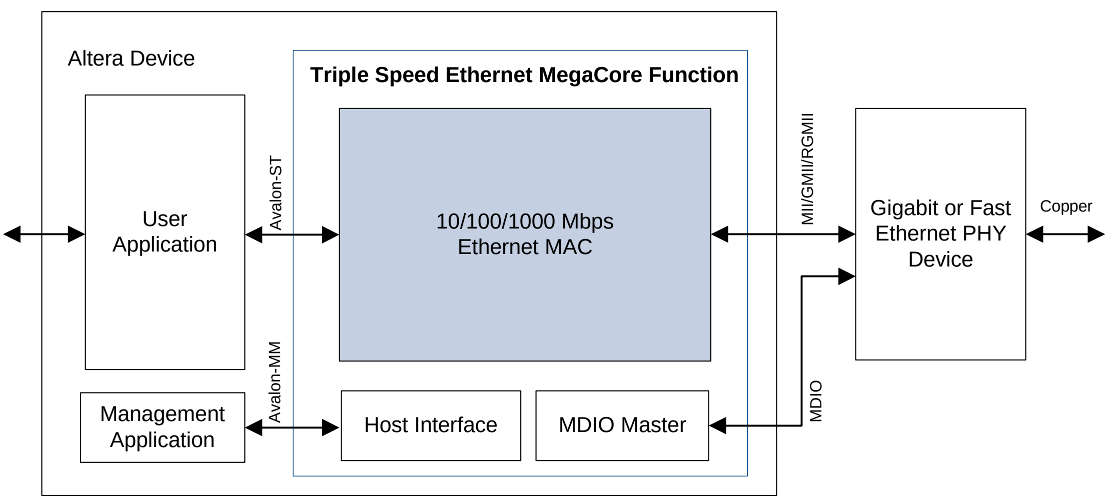



# Example Applications

This section shows example applications of the different variations of the Triple-Speed Ethernet MegaCore function.

The 10/100/1000-Gbps Ethernet MAC only variation can serve as a bridge between the user application and standard fast or gigabit Ethernet PHY devices. Figure 1–6 shows an example application using this variation for a copper network.

**Figure 1–6.** Stand-Alone 10/100/1000 Mbps Ethernet MAC




When configured to include the 1000BASE-X/SGMII PCS function, the MegaCore function can seamlessly connect to any industry standard gigabit Ethernet PHY device via a TBI. Alternatively, when the 1000BASE-X/SGMII PCS function is configured to include an embedded PMA, the MegaCore function can connect directly to a gigabit interface converter (GBIC), small form-factor pluggable (SFP) module, or an SGMII PHY.

Figure 1–7 shows an example application using the Triple-Speed Ethernet MegaCore function with 1000BASE-X and PMA. The PMA block connects to an off-the-shelf GBIC or SFP module to communicate directly over the optical link.

**Figure 1–7.** 10/100/1000 Mbps Ethernet MAC and 1000BASE-X PCS with Embedded PMA

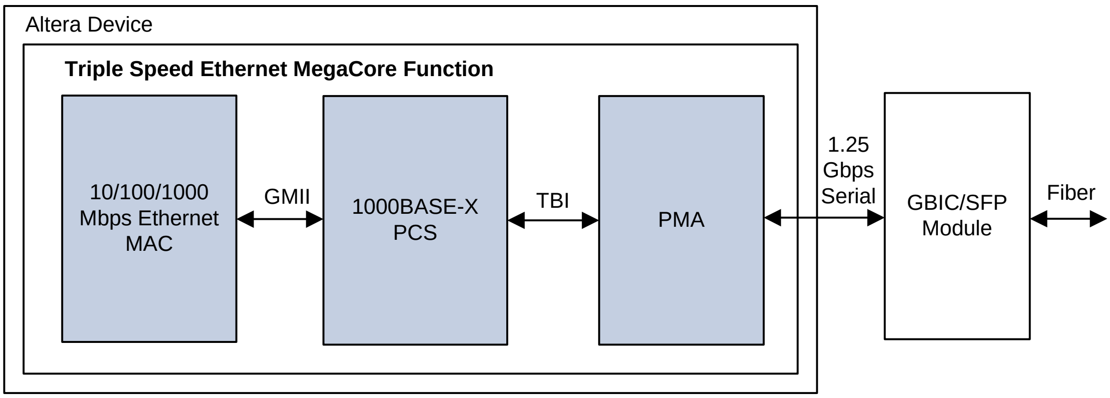
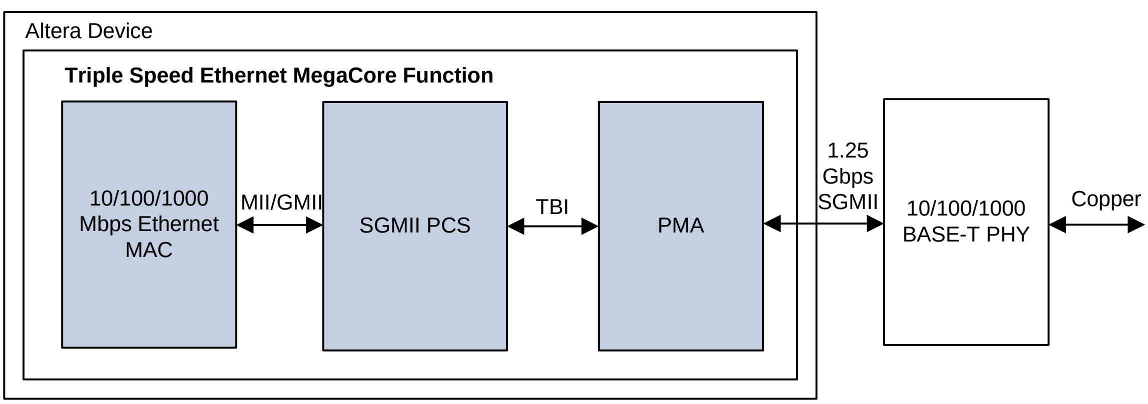



Figure 1–8 shows a similar configuration in which the PCS function is configured to operate in SGMII mode and acts as a GMII-to-SGMII bridge. In this case, the transceiver I/O connects to an off-the-shelf Ethernet PHY that supports SGMII (10BASE-T, 100BASE-T, or 1000BASE-T Ethernet PHY).

**Figure 1–8.** 10/100/1000 Mbps Ethernet MAC and SGMII PCS with Embedded PMA—GMII/MII to 1.25-Gbps Serial Bridge Mode




# MegaCore Verification

For each release, Altera verifies the Triple-Speed Ethernet MegaCore function through extensive simulation and internal hardware verification in various Altera device families. The University of New Hampshire (UNH) InterOperability Lab also successfully verified the MegaCore function prior to its release.

Altera used a highly parameterizeable transaction-based testbench to test the following aspects of the MegaCore function:

- Register access
- MDIO access

- Frame transmission and error handling
- Frame reception and error handling
- Ethernet frame MAC address filtering
- Flow control
- Retransmission in half-duplex

Altera has also validated the Triple-Speed Ethernet MegaCore function in both optical and copper platforms using the following development kits:

- Altera Nios II Development Kit, Cyclone II Edition (2C35)
- Altera Stratix III FPGA Development Kit
- Altera Stratix IV FPGA Development Kit
- Quad 10/100/1000 Marvell PHY
- MorethanIP 10/100 and 10/100/1000 Ethernet PHY Daughtercards

## Optical Platform

In the optical platform, the 10/100/1000 Mbps Ethernet MAC, 1000BASE-X/SGMII PCS, and PMA functions are instantiated.

The FPGA application implements the Ethernet MAC, the 1000BASE-X PCS, and an internal system using Ethernet connectivity. This internal system retrieves all frames received by the MAC function and returns them to the sender by manipulating the MAC address fields, thus implementing a loopback. A direct connection to an optical module is provided through an external SFP optical module. Certified 1.25 GBaud optical SFP transceivers are Finisar 1000BASE-SX FTLF8519P2BNL, Finisar 1000BASE-LX FTRJ-1319-3, and Avago Technologies AFBR-5710Z.

## Copper Platform

In the copper platform, Altera tested the Triple-Speed Ethernet MegaCore function with an external 1000BASE-T PHY devices. The MegaCore function is connected to the external PHY device using MII, GMII, RGMII, and SGMII, in conjunction with the 1000BASE-X/SGMII PCS and PMA functions.

A 10/100/1000 Mbps Ethernet MAC and an internal system are implemented in the FPGA. The internal system retrieves all frames received by the MAC function and returns them to the sender by manipulating the MAC address fields, thus implementing a loopback. A direct connection to an Ethernet link is provided through a combined MII to an external PHY module. Certified 1.25 GBaud copper SFP transceivers are Finisar FCMJ-8521-3, Methode DM7041, and Avago Technologies ABCU-5700RZ.

# Performance and Resource Utilization

The $f_{MAX}$ of the configurations in the following tables is more than 125 MHz.

Table 1–4 provides the estimated resource utilization and performance of the Triple-Speed Ethernet MegaCore function for the Arria II GX device family. The estimates are obtained by compiling the Triple-Speed Ethernet MegaCore function using the Quartus II software targeting an Arria II GX (EP2AGX260EF29I3) device with speed grade -3.

**Table 1–4.** Arria II GX Performance and Resource Utilization

| MegaCore Function | Settings | FIFO Buffer Size (Bits) | Combinational ALUTs | Logic Registers | Memory (M9K Blocks/ M144K Blocks/ MLAB Bits) |
|---|---|---|---|---|---|
| 10/100/1000-Mbps Ethernet MAC | RGMII<br>All MAC options enabled<br>Full and half-duplex modes supported | 2048x32 | 3357 | 3947 | 26/0/1828 |
| 8-port 10/100/1000-Mbps Ethernet MAC | MII/GMII<br>All MAC options enabled<br>Full and half-duplex modes supported | — | 20201 | 22292 | 32/0/14624 |
| 1000BASE-X/SGMII PCS | 1000BASE-X | — | 624 | 661 | 0/0/0 |
| | 1000BASE-X<br>SGMII bridge enabled<br>PMA block (GXB) | — | 1191 | 1214 | 1/0/160 |

Table 1–5 provides the estimated resource utilization and performance of the Triple-Speed Ethernet MegaCore function for the Stratix IV device family. The estimates are obtained by compiling the Triple-Speed Ethernet MegaCore function using the Quartus II software targeting a Stratix IV GX (EP4SGX530NF45C4) device with speed grade -4.

**Table 1–5.** Stratix IV Performance and Resource Utilization (Part 1 of 2)

| MegaCore Function | Settings | FIFO Buffer Size (Bits) | Combinational ALUTs | Logic Registers | Memory (M9K Blocks/ M144K Blocks/ MLAB Bits) |
|---|---|---|---|---|---|
| 10/100 Mbps Small MAC | MII<br>Full and half-duplex modes supported | 2048x32 | 1410 | 2127 | 12/1/1408 |
| | MII<br>All MAC options enabled | 2048x32 | 1157 | 1894 | 12/1/128 |
| 1000Mbps Small MAC | GMII<br>All MAC options enabled | 2048x32 | 1160 | 1827 | 12/1/176 |
| | RGMII<br>All MAC options enabled | 2048x32 | 1170 | 1861 | 12/1/176 |

**Table 1–5.** Stratix IV Performance and Resource Utilization (Part 2 of 2)

| MegaCore Function | Settings | FIFO Buffer Size (Bits) | Combinational ALUTs | Logic Registers | Memory (M9K Blocks/ M144K Blocks/ MLAB Bits) |
|---|---|---|---|---|---|
| 10/100/1000-Mbps Ethernet MAC | MII/GMII Full and half-duplex modes supported | — | 2721 | 3395 | 0/0/3364 |
| | | 2048x8 | 3201 | 3977 | 8/0/3620 |
| | | 2048x32 | 3345 | 4425 | 12/1/3364 |
| | MII/GMII All MAC options enabled | 2048x32 | 3125 | 3994 | 12/1/2084 |
| | RGMII All MAC options enabled | 2048x32 | 3133 | 4021 | 12/1/2084 |
| 12-port 10/100/1000-Mbps Ethernet MAC | MII/GMII All MAC options enabled | — | 27215 | 34372 | 0/0/25008 |
| 24-port 10/100/1000-Mbps Ethernet MAC | | — | 54123 | 68404 | 0/0/50016 |
| 1000BASE-X/SGMII PCS | 1000BASE-X | — | 624 | 661 | 0/0/0 |
| | 1000BASE-X SGMII bridge enabled | — | 808 | 986 | 2/0/0 |
| | 1000BASE-X SGMII bridge enabled PMA block (LVDS_IO) | — | 819 | 1057 | 2/0/0 |
| | 1000BASE-X SGMII bridge enabled PMA block (GXB) | — | 1189 | 1212 | 1/0/160 |
| 10/100/1000-Mbps Ethernet MAC and 1000BASE-X/SGMII PCS | All MAC options enabled SGMII bridge enabled | 2048×32 | 3971 | 4950 | 14/1/2084 |

Table 1–6 provides the estimated resource utilization and performance of the Triple Speed Ethernet MegaCore function for the Cyclone IV device family. The estimates are obtained by compiling the Triple-Speed Ethernet MegaCore function using the Quartus II software targeting a Cyclone IV GX (EP4CGX150DF27C7) device with speed grade -7.

**Table 1–6.** Cyclone IV GX Performance and Resource Utilization (Part 1 of 2)

| MegaCore Function | Settings | FIFO Buffer Size (Bits) | Logic Elements | Logic Registers | Memory (M9K Blocks/ Mi44K Blocks/ MLAB Bits) |
|---|---|---|---|---|---|
| 1000Mbps Small MAC | RGMII Only full-duplex mode supported | 2048x32 | 2161 | 1699 | 24/0/0 |
| 10/100/1000-Mbps Ethernet MAC | MII/GMII Full and half-duplex modes supported | 2048x32 | 5614 | 3666 | 31/0/0 |

**Table 1–6.** Cyclone IV GX Performance and Resource Utilization (Part 2 of 2)

| MegaCore Function | Settings | FIFO Buffer Size (Bits) | Logic Elements | Logic Registers | Memory (M9K Blocks/ Mi44K Blocks/ MLAB Bits) |
|---|---|---|---|---|---|
| 4-port 10/100/ 1000-Mbps Ethernet MAC | MII/GMII<br>All MAC options enabled<br>Full and half-duplex modes supported | — | 17017 | 10612 | 36/0/0 |
| 1000BASE-X/SGMII PCS | 1000BASE-X | — | 1149 | 661 | 0/0/0 |
| | 1000BASE-X<br>SGMII bridge enabled<br>PMA block (GXB) | — | 2001 | 1127 | 2/0/0 |

Table 1–7 provides the estimated resource utilization and performance of the Triple-Speed Ethernet MegaCore function for the Stratix V device family. The estimates are obtained by compiling the Triple-Speed Ethernet MegaCore function using the Quartus II software targeting a Stratix V GX (5SGXMA7N3F45C3) device with speed grade -3.

**Table 1–7.** Stratix V Performance and Resource Utilization (Part 1 of 2)

| MegaCore Function | Settings | FIFO Buffer Size (Bits) | Combinational ALUTs | Logic Registers | Memory (M20K Blocks/ MLAB Bits) |
|---|---|---|---|---|---|
| 10/100 Mbps Small MAC | MII<br>Full and half-duplex modes supported | 2048x32 | 1425 | 1967 | 11/1208 |
| | MII<br>All MAC options enabled | 2048x32 | 1155 | 1767 | 11/0 |
| 1000Mbps Small MAC | GMII<br>All MAC options enabled | 2048x32 | 1174 | 1670 | 10/176 |
| | RGMII<br>All MAC options enabled | 2048x32 | 1184 | 1704 | 10/176 |
| 10/100/1000-Mbps Ethernet MAC | MII/GMII<br>Full and half-duplex modes supported | — | 2754 | 2938 | 6/1664 |
| | | 2048x8 | 3228 | 3525 | 10/1920 |
| | | 2048x32 | 3358 | 3694 | 16/1664 |
| | MII/GMII<br>All MAC options enabled | 2048x32 | 3142 | 3495 | 16/384 |
| | RGMII<br>All MAC options enabled | 2048x32 | 3155 | 3522 | 16/384 |
| 12-port 10/100/1000-Mbps Ethernet MAC | MII/GMII<br>All MAC options enabled | — | 27360 | 29272 | 72/4608 |
| 24-port 10/100/1000-Mbps Ethernet MAC | | — | 54415 | 58204 | 144/9216 |

**Table 1–7.** Stratix V Performance and Resource Utilization (Part 2 of 2)

| MegaCore Function | Settings | FIFO Buffer Size (Bits) | Combinational ALUTs | Logic Registers | Memory (M20K Blocks/ MLAB Bits) |
|---|---|---|---|---|---|
| 1000BASE-X/SGMII PCS | 1000BASE-X | — | 621 | 661 | 0/0 |
| | 1000BASE-X<br>SGMII bridge enabled | — | 804 | 986 | 2/0 |
| | 1000BASE-X<br>SGMII bridge enabled<br>PMA block (LVDS_IO) | — | 820 | 1056 | 2/0 |
| | 1000BASE-X<br>SGMII bridge enabled<br>PMA block (GXB) | — | 1845 | 1550 | 2/160 |
| 10/100/1000-Mbps Ethernet MAC and 1000BASE-X/SGMII PCS | All MAC options enabled<br>SGMII bridge enabled | 2048×32 | 3971 | 4451 | 18/384 |
| | Default MAC option<br>SGMII bridge enabled<br>IEEE 1588v2 feature enabled | 0 | 5062 | 5318 | 4/1536 |

# Release Information

Table 1–8 provides information about this release of the Triple-Speed Ethernet MegaCore function.

**Table 1–8.** Triple-Speed Ethernet MegaCore Function Release Information

| Item | Description |
|---|---|
| Version | 13.0 |
| Release Date | May 2013 |
| Ordering Code | IP-TRIETHERNET |
| Product ID(s) | 00BD (Triple-Speed Ethernet MegaCore function)<br>0104 (IEEE 1588v2) |
| Vendor ID(s) | 6AF7 |

For more information about this release, refer to the *MegaCore IP Library Release Notes and Errata*.

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

# 2. Getting Started with Altera IP Cores

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

## Installation and Licensing

The Altera IP Library is distributed with the Quartus II software and downloadable from the Altera website (www.altera.com).

Figure 2–1 shows the directory structure after you install an Altera IP core, where *<path>* is the installation directory. The default installation directory on Windows is **C:\altera**<*version number*>; on Linux it is **/opt/altera**<*version number*>**.**

**Figure 2–1.** IP core Directory Structure

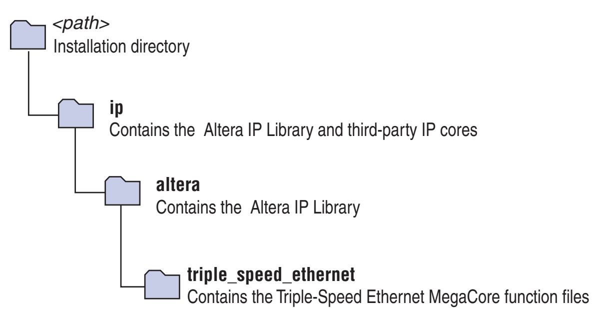



Figure 2–1 shows the directory structure

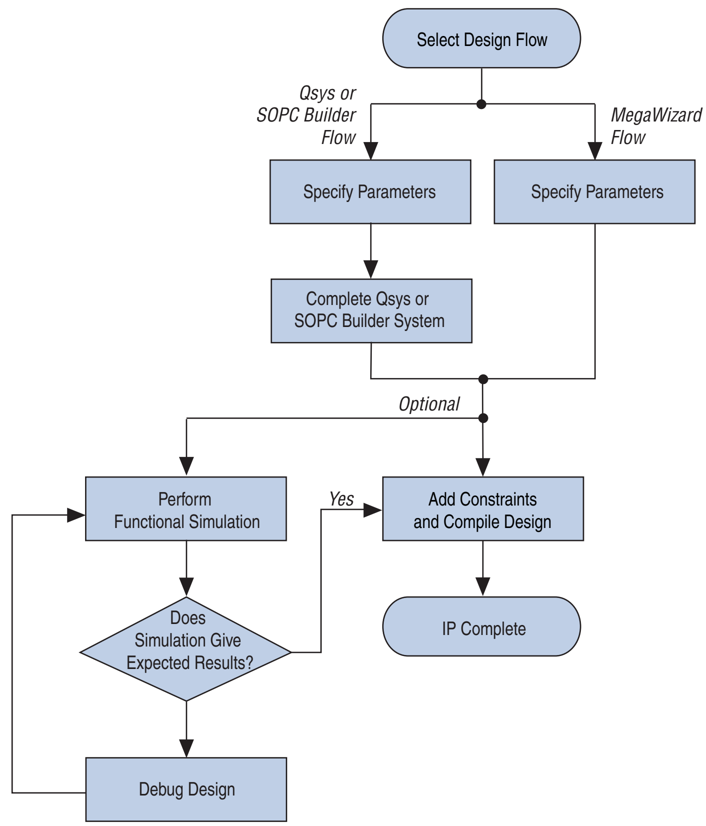
You can evaluate an IP core in simulation and in hardware until you are satisfied with its functionality and performance. Some IP cores require that you purchase a license for the IP core when you want to take your design to production. After you purchase a license for an Altera IP core, you can request a license file from the Altera Licensing page of the Altera website and install the license on your computer. For additional information, refer to *Altera Software Installation and Licensing*.

# Design Flows

You can use the following flow(s) to parameterize Altera IP cores:

- MegaWizard™ Plug-In Manager Flow
- Qsys Flow

**Figure 2–2.** Design Flows *(1)*




**Note to Figure 2–2:**

(1) Altera IP cores may or may not support the Qsys and SOPC Builder design flows.

The MegaWizard Plug-In Manager flow offers the following advantages:

- Allows you to parameterize an IP core variant and instantiate into an existing design
- For some IP cores, this flow generates a complete example design and testbench

The Qsys flow offers the following additional advantages over SOPC Builder:

- Provides visualization of hierarchical designs
- Allows greater performance through interconnect elements and pipelining
- Provides closer integration with the Quartus II software

# MegaWizard Plug-In Manager Flow

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

## Creating a New Quartus II Project

From the Windows Start menu, choose **Programs > Altera > Quartus II** *<version>* to run the Quartus II software. Alternatively, you can use the Quartus II Web Edition software.

You need to create a new Quartus II project with the **New Project Wizard**, which specifies the working directory for the project, assigns the project name, and designates the name of the top-level design entity. On the File menu, click **New Project Wizard**.

To create a new project, follow the steps in Table 2–1. Click **Next** to move to the following pages.

**Table 2–1. Creating a New Quartus II Project**

| Step | Page | Field and Entry | Description |
|---|---|---|---|
| — | New Project Wizard: Introduction | N/A | The introduction page is not displayed if you have turned it off previously. |
| 1 | New Project Wizard: Directory, Name, Top-Level Entity | Working directory: **c:\altera\projects\tse_project** | Set the working directory name. The entry is an example for the directory name. |
| | | Project name: **project** | Set the project name. This walkthrough uses **project** for the project name. |
| | | Top-level entity: **project** | Set the top-level design entity name. The Quartus II software automatically specifies a top-level design entity that has the same name as the project. This walkthrough assumes that the names are the same. |
| 2 | New Project Wizard: Add Files *(1)* | File name: *<path>***\ip\altera\triple_speed_ethernet** | Select the existing design files (if any) you want to include in the project. *<path>* is the directory in which you installed the SerialLite III Streaming MegaCore function. |
| 3 | New Project Wizard: Family & Device Settings | Device Family | Select the device family and specific device you want to target for compilation. |
| 4 | EDA Tool Settings | Tool Name | Select the EDA tools you want to use with the Quartus II software to develop your project. |

**Note to Table 2–1:**

(1) You must add the user libraries if you installed the MegaCore IP Library in a different directory from where you installed the Quartus II software.

The last page in the **New Project Wizard** shows the summary of your settings. Click **Finish** to complete the Quartus II project creation.

## Launching MegaWizard Plug-In Manager

To launch the MegaWizard Plug-In Manager in the Quartus II software, follow these steps:

1. On the Tools menu, click **MegaWizard Plug-In Manager**.
2. Select **Create a new custom megafunction variation** and click **Next**.
3. Select your target device family from the list.
4. Expand the **Interfaces > Ethernet** folder and click **Triple-Speed Ethernet** *<version>*.
5. Select the output file type for your design—AHDL, VHDL, or Verilog HDL.
6. Affix a variation name for the MegaCore function output files <*project path*>\<*variation name*>. The MegaWizard Plug-In Manager shows the project path that you specified in the **New Project Wizard**.
7. Click **Next** to display the Triple-Speed Ethernet MegaCore function parameter editor.
8. Set the parameters for the MegaCore function.

For more information about the MegaCore function parameters, refer to Chapter 3, Parameter Settings or the "*Documentation*" button in the MegaWizard parameter editor.

After you have parameterized the MegaCore function, you can also generate a design example, in addition to generating the MegaCore component files. Refer to "Generating a Design Example or Simulation Model" section.

Click **Yes** if you are prompted to add the Quartus II IP File (.qip) to the current Quartus II project.

You can now integrate your custom IP core instance in your design, simulate, and compile. While integrating your IP core instance into your design, you must make appropriate pin assignments. You can create a virtual pin to avoid making specific pin assignments for top-level signals while you are simulating and not ready to map the design to hardware.

For information about the Quartus II software, including virtual pins and the MegaWizard Plug-In Manager, refer to Quartus II Help.

## Generating a Design Example or Simulation Model

Check the **Generate Example Design** option to create a functional simulation model (design example that includes a testbench). The testbench and the automated script are located in the <*variation name*>**_testbench** directory.

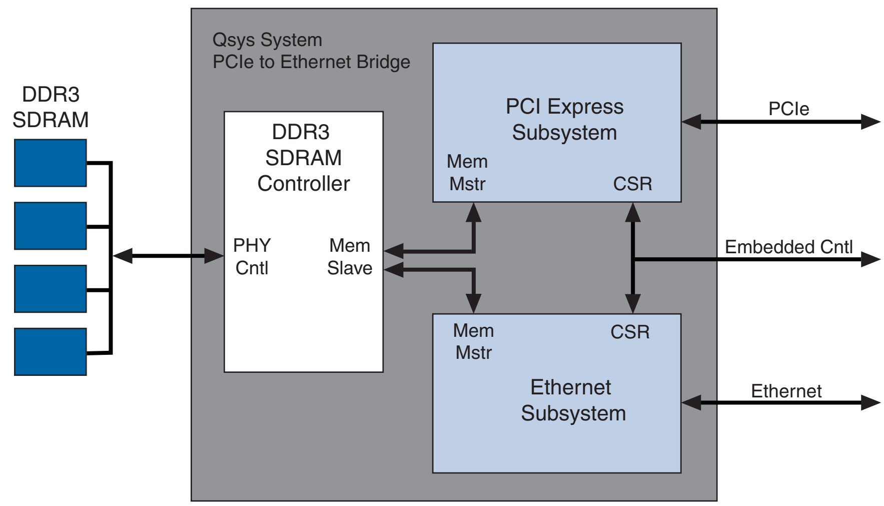
Generating a design example can increase processing time.

For more information about the MegaCore function simulation model, refer to Chapter 10, Testbench.

# Qsys System Integration Tool Design Flow

You can use the Qsys system integration tool to build a system that includes your customized IP core. You easily can add other components and quickly create a Qsys system. Qsys automatically generates HDL files that include all of the specified components and interconnections. In Qsys, you specify the connections you want. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device. Qsys generates Verilog HDL simulation models for the IP cores that comprise your system. Figure 2–3 shows a high level block diagram of an example Qsys system.

**Figure 2–3.** Example Qsys System




For more information about the Qsys system interconnect, refer to the *Qsys Interconnect* chapter in volume 1 of the *Quartus II Handbook* and to the *Avalon Interface Specifications*.

For more information about the Qsys tool and the Quartus II software, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook* and to Quartus II Help.

## Specify Parameters

To specify parameters for your IP core using the Qsys flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **Qsys**.
3. On the **Component Library** tab, expand the **Interface Protocols > Ethernet** directory and double-click the name of your IP core to add it to your system. The relevant parameter editor appears.

4. Specify the required parameters in all tabs in the Qsys tool. For detailed explanations of these parameters, refer to Chapter 3, Parameter Settings or the "*Documentation*" button in the parameter editor.

   ☞ If your design includes external memory interface IP cores, you must turn on **Generate power of two bus widths** on the **PHY Settings** tab when parameterizing those cores.

   ☞ Some IP cores provide preset parameters for specific applications. If you wish to use preset parameters, click the arrow to expand the **Presets** list, select the desired preset, and then click **Apply**. To modify preset settings, in a text editor modify the **<***installation directory***>/ip/altera/alt_mem_if_interfaces/alt_mem_if_<***memory_protocol***>_emif/alt_mem_if_<***memory_protocol***>_mem_model.qprs** file.

5. Click **Finish** to complete the IP core instance and add it to the system.

## Complete the Qsys System

To complete the Qsys system, follow these steps:

1. Add and parameterize any additional components.
2. Connect the components using the Connection panel on the **System Contents** tab.
3. In the **Export** column, enter the name of any connections that should be a top-level Qsys system port.
4. If you intend to simulate your Qsys system, on the **Generation** tab, select the simulation model type under **Simulation** to generate desired simulation files.
5. If your system is not part of a Quartus II project and you want to generate synthesis RTL files, select the design file type and turn on **Create block symbol file (.bsf)**.
6. Set the path name for the output directory.
7. Click **Generate** to generate the system. Qsys generates the system and produces the <*system name*>**.qip** file that contains the assignments and information required to process the IP core or system in the Quartus II Compiler.
8. In the Quartus II software, click **Add/Remove Files in Project** and add the **.qip** file to the project.
9. Compile your project in the Quartus II software.

☞ To ensure that the **memory** and **oct** interfaces are exported to the top-level RTL file, be careful not to accidentally rename or delete either of these interfaces in the **Export** column of the **System Contents** tab.

## Simulate the System

During system generation, Qsys generates a functional simulation model—or design example that includes a testbench—which you can use to simulate your system in any Altera-supported simulation tool.

For information about the following topics, refer to their respective documents:

- the latest Altera-supported simulation tools, refer to the *Quartus II Software Release Notes*.
- simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.
- simulating Qsys systems, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook*.

# Generated Files

Table 2–2 lists the files generated in your project directory. The type of files generated and their names vary depending on the custom variation of the MegaCore function you created.

**Table 2–2.** Generated Files (Part 1 of 2)

| File Name | Description |
|---|---|
| <*variation_name*>**.v** or<br><*variation_name*>**.vhd** | A MegaCore function variation file, which defines a VHDL or Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside your design. Include this file when compiling your design in the Quartus II software. |
| <*variation name*>**.bsf** | Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor. |
| <*variation name*>**.qip** and<br><*variation name*>**.sip** | Contains Quartus II project information for your MegaCore function variations. |
| <*variation name*>**.cmp** | A VHDL component declaration file for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore. |
| <*variation name*>**.spd** | Simulation Package Descriptor file. Specifies the files required for simulation. |
| Testbench Files (in <*variation_name*>**_testbench** folder) | |
| **README.txt** | Read me file for the testbench design. |
| **generate_sim.qpf** and<br>**generate_sim.qsf** | Dummy Quartus II project and project setting file. Use this to start the Quartus II in the correct directory to launch the **generate_sim_verilog.tcl** and **generate_sim_vhdl.tcl** files. |
| **generate_sim_verilog.tcl** and<br>**generate_sim_vhdl.tcl** | A Tcl script to generate the DUT VHDL or Verilog HDL simulation model for use in the testbench. |
| **/testbench_vhdl/**<*variation_name*>**/**<*variation_name*>**_tb.vhd** or<br>**/testbench_verilog/**<*variation_name*>**/**<*variation_name*>**_tb.v** | VHDL or Verilog HDL testbench that exercises your MegaCore function variation in a third party simulator. |

**Table 2–2.** Generated Files (Part 2 of 2)

| File Name | Description |
|---|---|
| **/testbench_vhdl/**<*variation_name*>**/run_** <*variation_name*>**_tb.tcl** or<br>**/testbench_verilog/**<*variation_name*>**/run_** <*variation_name*>**_tb.tcl** | A Tcl script for use with the ModelSim simulation software. |
| **/testbench_vhdl/**<*variation_name*>**/** <*variation_name*>**_wave.do** or<br>**/testbench_verilog/**<*variation_name*>**/** <*variation_name*>**_wave.do** | A signal tracing macro script used with the ModelSim simulation software to display testbench signals. |
| **/testbench_vhdl/models** or<br>**/testbench_verilog/models** | A directory containing VHDL and Verilog HDL models of the Ethernet generators and monitors used by the generated testbench. |

## Design Constraint File No Longer Generated

For the new Triple-Speed Ethernet MegaCore function created using the Quartus II software ACDS 13.0, the Quartus II software no longer generate the <*variation_name*>**_constraints.tcl** file that contains the necessary constraints for the compilation of your MegaCore Function variation. Table 2–3 lists the recommended Quartus II pin assignments that you can set in your design.

**Table 2–3. Recommended Quartus II Pin Assignments**

| Quartus II Pin Assignment | Assignment Value | Description | Design Pin |
|---|---|---|---|
| FAST_INPUT_REGISTER | ON | To optimize I/O timing for MII, GMII and TBI interface. | MII, GMII, RGMII, TBI input pins |
| FAST_OUTPUT_REGISTER | ON | To optimize I/O timing for MII, GMII and TBI interface. | MII, GMII, RGMII, TBI output pins |
| IO_STANDARD | 1.4-V PCML or 1.5-V PCML | I/O standard for GXB serial input and output pins. | GXB transceiver serial input and output pins |
| IO_STANDARD | LVDS | I/O standard for LVDS/IO serial input and output pins. | LVDS/IO transceiver serial input and output pins |
| GLOBAL_SIGNAL | Global clock | To assign clock signals to use the global clock network. Use this setting to guide the Quartus II in the fitter process for better timing closure. | ■ `ref_clk` for MAC and PCS with LVDS/IO (with internal FIFO)<br>■ `clk` and `reset` pins for MAC only (without internal FIFO)<br>■ `clk` and `ref_clk` input pins for MAC and PCS with transceiver (without internal FIFO) |
| GLOBAL_SIGNAL | Regional clock | To assign clock signals to use the regional clock network. Use this setting to guide the Quartus II in the fitter process for better timing closure. | ■ `rx_clk <n>` and `tx_clk <n>` input pins for MAC only using MII/GMII interface (without internal FIFO)<br>■ `rx_clk <n>` input pin for MAC only using RGMII interface (without internal FIFO) |
| GLOBAL_SIGNAL | OFF | To prevent a signal to be used as a global signal. | Signals for Arria V devices:<br>■ *reset_ff_wr and *reset_ff_rd<br>■ *\| altera_tse_reset_synchronizer_chain_out |

# 3. Parameter Settings

You customize the Triple-Speed Ethernet MegaCore function by specifying parameters using the Triple-Speed Ethernet MegaWizard interface, launched from either the MegaWizard Plug-in Manager or Qsys in the Quartus II software. The customization enables specific core features during synthesis and generation.

This chapter describes the parameters and how they affect the behavior of the MegaCore function. Each section corresponds to a page in the **Parameter Settings** tab in the Triple-Speed Ethernet MegaWizard interface.

## Core Configuration

Table 3–1 describes the core configuration parameters.

**Table 3–1.** Core Configuration Parameters (Part 1 of 2)

| Name | Value | Description |
|---|---|---|
| **Core Variation** | ■ **10/100/1000 Mb Ethernet MAC**<br>■ **10/100/1000 Mb Ethernet MAC with 1000BASE-X/SGMII PCS**<br>■ **1000BASE-X/SGMII PCS only**<br>■ **1000 Mb Small MAC**<br>■ **10/100 Mb Small MAC** | Determines the primary blocks to include in the variation. |
| **Interface** | ■ **MII**<br>■ **GMII**<br>■ **RGMII**<br>■ **MII/GMII** | Determines the Ethernet-side interface of the MAC block.<br>■ **MII**—The only option available for 10/100 Mb Small MAC core variations.<br>■ **GMII**—Available only for 1000 Mb Small MAC core variations.<br>■ **RGMII**—Available for 10/100/1000 Mb Ethernet MAC and 1000 Mb Small MAC core variations.<br>■ **MII/GMII**—Available only for 10/100/1000 Mb Ethernet MAC core variations. If this is selected, media independent interface (MII) is used for the 10/100 interface, and gigabit media independent interface (GMII) for the gigabit interface. |
| **Use internal FIFO** | On/Off | If this parameter is turned on, internal FIFO buffers are included in the core. You can only include internal FIFO buffers in single-port MACs. |
| **Number of ports** | **1**, **4**, **8**, **12**, **16**, **20**, and **24** | Specifies the number of Ethernet ports supported by the IP core. This parameter is enabled if the parameter **Use internal FIFO** is turned off. A multiport MAC does not support internal FIFO buffers. |

**Table 3–1.** Core Configuration Parameters (Part 2 of 2)

| Name | Value | Description |
|---|---|---|
| **Transceiver type** | ■ **None**<br>■ **LVDS I/O**<br>■ **GXB** | This option is only available for variations that include the PCS block.<br>■ **None**—the PCS block does not include an integrated transceiver module. The PCS block implements a ten-bit interface (TBI) to an external SERDES chip.<br>■ **LVDS I/O** or **GXB**—the MegaCore function includes an integrated transceiver module to implement a 1.25 Gbps transceiver. Respective GXB module is included for target devices with GX transceivers. For target devices with LVDS I/O including Soft-CDR such as Stratix III, the ALTLVDS module is included. |

# Ethernet MAC Options

Table 3–2 describes the MAC options. These options are enabled when your variation includes the MAC function. In small MACs, only the following options are available:

- **Enable MAC 10/100 half duplex support** (10/100 Small MAC variations)
- **Align packet headers to 32-bit boundary** (10/100 and 1000 Small MAC variations)

**Table 3–2.** MAC Options Parameters (Part 1 of 2)

| Name | Value | Description |
|---|---|---|
| Ethernet MAC Options | | |
| **Enable MAC 10/100 half duplex support** | On/Off | Turn on this option to include support for half duplex operation on 10/100 Mbps connections. |
| **Enable local loopback on MII/GMII/RGMII** | On/Off | Turn on this option to enable local loopback on the MAC's MII, GMII, or RGMII interface. If you turn on this option, the loopback function can be dynamically enabled or disabled during system operation via the MAC configuration register. |
| **Enable supplemental MAC unicast addresses** | On/Off | Turn on this option to include support for supplementary destination MAC unicast addresses for fast hardware-based received frame filtering. |
| **Include statistics counters** | On/Off | Turn on this option to include support for simple network monitoring protocol (SNMP) management information base (MIB) and remote monitoring (RMON) statistics counter registers for incoming and outgoing Ethernet packets.<br>By default, the width of all statistics counters are 32 bits. |
| **Enable 64-bit statistics byte counters** | On/Off | Turn on this option to extend the width of selected statistics counters— `aOctetsTransmittedOK`, `aOctetsReceivedOK`, and `etherStatsOctets`—to 64 bits. |

**Table 3–2.** MAC Options Parameters (Part 2 of 2)

| Name | Value | Description |
|---|---|---|
| **Include multicast hashtable** | On/Off | Turn on this option to implement a hash table, a fast hardware-based mechanism to detect and filter multicast destination MAC address in received Ethernet packets. |
| **Align packet headers to 32-bit boundary** | On/Off | Turn on this option to include logic that aligns all packet headers to a 32-bit boundary. This helps reduce software overhead processing in realignment of data buffers.<br>This option is only available for MAC variations with 32 bits wide internal FIFO buffers.<br>You must turn on this option if you intend to use the Triple-Speed Ethernet MegaCore function with the Interniche TCP/IP protocol stack. |
| **Enable full-duplex flow control** | On/Off | Turn on this option to include the logic for full-duplex flow control that includes pause frames generation and termination. |
| **Enable VLAN detection** | On/Off | Turn on this option to include the logic for VLAN and stacked VLAN frame detection. When turned off, the MAC does not detect VLAN and staked VLAN frames. The MAC forwards these frames to the user application without processing them. |
| **Enable magic packet detection** | On/Off | Turn on this option to include logic for magic packet detection (Wake-on LAN). |
| **MDIO Module** | | |
| **Include MDIO module (MDC/MDIO)** | On/Off | Turn on this option if you want to access external PHY devices connected to the MAC function. When turned off, the core does not include the logic or signals associated with the MDIO interface. |
| **Host clock divisor** | — | Clock divisor to divide the MAC control interface clock to produce the MDC clock output on the MDIO interface. The default value is 40.<br>For example, if the MAC control interface clock frequency is 100 MHz and the desired MDC clock frequency is 2.5 MHz, a host clock divisor of 40 should be specified.<br>Altera recommends that the division factor is defined such that the MDC frequency does not exceed 2.5 MHz. |

# FIFO Options

Table 3–3 describes the FIFO options, which are enabled only for MAC variations that include internal FIFO buffers.

**Table 3–3.** FIFO Options Parameters

| Name | Value | Parameter |
|---|---|---|
| **Width** | | |
| **Width** | **8 Bits** and **32 Bits** | Determines the data width in bits of the transmit and receive FIFO buffers. |
| **Depth** | | |
| **Transmit** | Between 64 and 64K | Determines the depth of the internal FIFO buffers. |
| **Receive** | | |

# Timestamp Options

Table 3–3 describes the timestamp options.

**Table 3–4.** Timestamp Options Parameters

| Name | Value | Parameter |
|---|---|---|
| **Timestamp** | | |
| **Enable timestamping** | On/Off | Turn on this parameter to enable time stamping on the transmitted and received frames. |
| **Enable PTP 1-step clock** | On/Off | Turn on this parameter to insert timestamp on PTP messages for 1-step clock based on the TX Timestamp Insert Control interface.<br>This parameter is disabled if you do not turn on **Enable timestamping**. |
| **Timestamp fingerprint width** | — | Use this parameter to set the width in bits for the timestamp fingerprint on the TX path. The default value is 4 bits. |

# PCS/Transceiver Options

Table 3–5 describes the PCS/Transceiver options, which are enabled only if your variation includes the PCS function.

**Table 3–5.** PCS/Transceiver Options Parameters (Part 1 of 2)

| Name | Value | Parameter |
|---|---|---|
| **PCS Options** | | |
| **PHY ID (32 bit)** | — | Configures the PHY ID of the PCS block. |
| **Enable SGMII bridge** | On/Off | Turn on this option to add the SGMII clock and rate-adaptation logic to the PCS block. If your application only requires 1000BASE-X PCS, turning off this option reduces resource usage.<br>In Cyclone IV GX devices, `REFCLK[0,1]` and `REFCLK[4,5]` cannot connect directly to the GCLK network. If you enable the SGMII bridge, you must connect `ref_clk` to an alternative dedicated clock input pin. |
| **Transceiver Options**—apply only to variations that include GXB transceiver blocks | | |
| **Export transceiver powerdown signal** | On/Off | This option is not supported in Stratix V, Arria V, Arria V GZ, and Cyclone V devices.<br>Turn on this option to export the powerdown signal of the GX transceiver to the top-level of your design. Powerdown is shared among the transceivers in a quad. Therefore, turning on this option in multiport Ethernet configurations maximizes efficient use of transceivers within the quad.<br>Turn off this option to connect the powerdown signal internally to the PCS control register interface. This connection allows the host processor to control the transceiver powerdown in your system.<br>For UNH-IOL certification purposes, you must set the embedded transceiver to use 7-bit word alignment pattern length to recognize the comma character found in /K28.1/, /K28.5/, and /K28.7/. Use the MegaWizard Plug-in Manager to edit the megafunction and change the default word alignment setting to 7 bits. |

**Table 3–5.** PCS/Transceiver Options Parameters (Part 2 of 2)

| Name | Value | Parameter |
|---|---|---|
| **Enable transceiver dynamic reconfiguration** | On/Off | This option is always turned on in devices other than Arria GX and Stratix II GX. When this option is turned on, the MegaCore function includes the dynamic reconfiguration signals.<br>For designs targeting devices other than Stratix V and Arria V, Altera recommends that you instantiate the ALTGX_RECONFIG megafunction and connect the megafunction to the dynamic reconfiguration signals to enable offset cancellation.<br>For Stratix V and Arria V designs, Altera recommends that you instantiate the Transceiver Reconfiguration Controller megafunction and connect the megafunction to the dynamic reconfiguration signals to enable offset cancellation. The transceivers in the Stratix V and Arria V designs are configured with Altera Custom PHY IP core. The Custom PHY IP core require two reconfiguration interfaces for external reconfiguration controller. For more information on the reconfiguration interfaces required, refer to the *Altera Transceiver PHY IP Core User Guide* and the respective device handbook.<br>For more information about quad sharing considerations, refer to “Sharing PLLs in Devices with GIGE PHY” on page 8–7. |
| **Starting channel number** | **0 – 284** | Specifies the channel number for the GXB transceiver block. In a multiport MAC, this parameter specifies the channel number for the first port. Subsequent channel numbers are in four increments.<br>In designs with multiple instances of GXB transceiver block (multiple instances of Triple-Speed Ethernet IP core with GXB transceiver block or a combination of Triple-Speed Ethernet IP core and other IP cores), Altera recommends that you set a unique starting channel number for each instance to eliminate conflicts when the GXB transceiver blocks share a transceiver quad.<br>This option is not supported in Stratix V devices. For Stratix V devices, the channel numbers depends on the dynamic reconfiguration controller. |

☞ Refer to the respective device handbook for more information on dynamic reconfiguration in Altera devices.

## 10/100/1000 Ethernet MAC

The Altera 10/100/1000 Ethernet MAC function handles the flow of data between user applications and Ethernet network through an internal or external Ethernet PHY. Altera offers the following MAC variations:

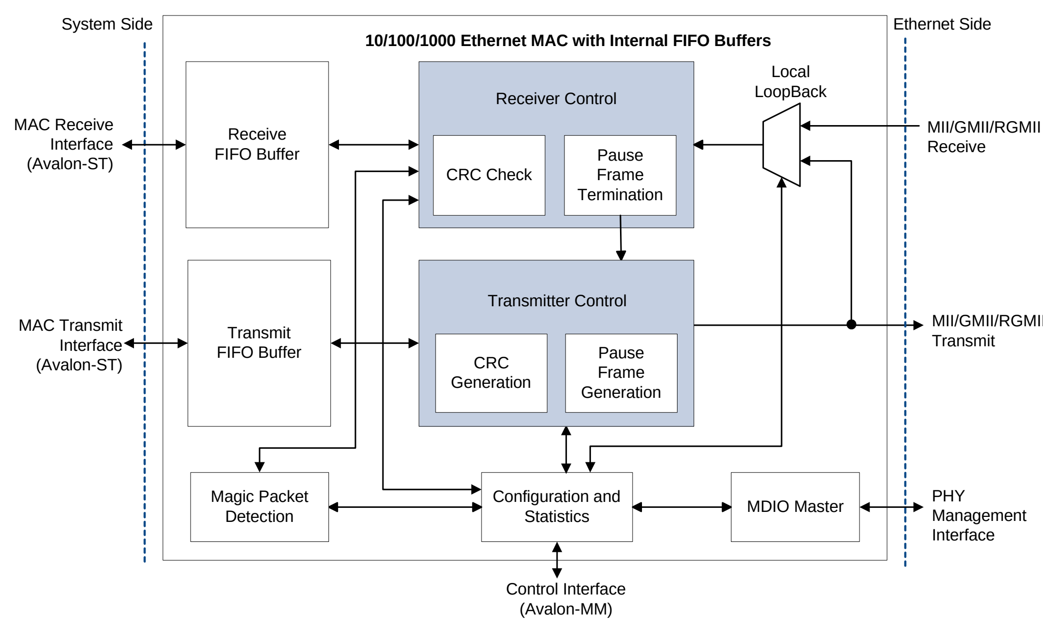
- Variations with internal FIFO buffers—supports only single port.
- Variations without internal FIFO buffers—supports up to 24 ports and the ports can operate at different speeds.
- Small MAC—provides basic functionalities of a MAC function using minimal resources. Refer to Table 1–3 on page 1–4 for a feature comparison between the 10/100/1000 Ethernet MAC and small MAC.

The MAC function supports the following Ethernet frames: basic, VLAN and stacked VLAN, jumbo, and control frames. For more information about these frame formats, refer to "Ethernet Frame Format" on page B–1.

### Architecture

Figure 4–1 shows a block diagram of the MAC variation with internal FIFO buffers.

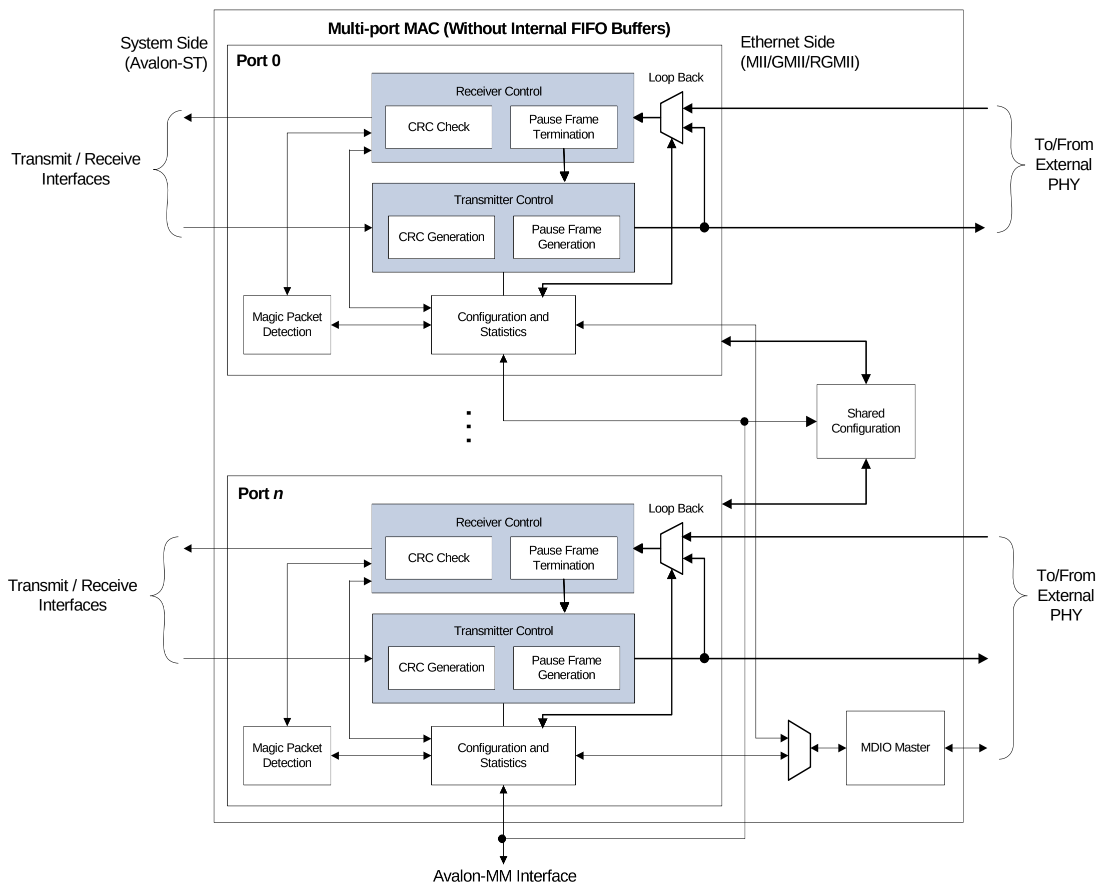
**Figure 4–1.** 10/100/1000 Ethernet MAC With Internal FIFO Buffers



The FIFO buffers, which you can configure to 8 or 32 bits wide, store the transmit and receive data. The buffer width determines the data width on the Avalon-ST receive and transmit interfaces. You can configure the FIFO buffers to operate in cut-through or store-and-forward mode using the `rx_section_full` and `tx_section_full` registers. For more information about these registers, refer to "MAC Configuration Register Space" on page 6–1.

Figure 4–2 shows a block diagram of the multiport MAC variation without internal FIFO buffers.

**Figure 4–2.** Multiport MAC Without Internal FIFO Buffers




In a multiport MAC, the instances share the MDIO master and some configuration registers. You can use the Avalon-ST Multi-Channel Shared Memory FIFO core in SOPC Builder and Qsys to store the transmit and receive data.

# Interfaces

The MAC function implements the following interfaces:

- Avalon-ST on the system side.
  - Avalon-ST sink port on transmit with the following properties:
    - Fixed data width, 8 bits, in MAC variations without internal FIFO buffers; configurable data width, 8 or 32 bits, in MAC variations with internal FIFO buffers.
    - Packet support using start-of-packet (SOP) and end-of-packet (EOP) signals, and partial final packet signals.
    - Error reporting.
    - Variable-length ready latency specified by the `tx_almost_full` register. For more information about the register, refer to Table 4–7 on page 4–15.
  - Avalon-ST source port on receive with the following properties:
    - Fixed data width of 8 bits in MAC variations without internal FIFO buffers; configurable data width, 8 or 32 bits, in MAC variations with internal FIFO buffers.
    - Backpressure is supported only in MAC variations with internal FIFO buffers. Transmission stops when the level of the FIFO buffer reaches the respective programmable thresholds.
    - Packet support using SOP and EOP signals, and partial final packet signals.
    - Error reporting.
    - Ready latency is zero in MAC variations without internal FIFO buffers. In MAC variations with internal FIFO buffers, the ready latency is two.
- Media independent interfaces on the network side—select MII, GMII, or RGMII by setting the **Interface** option on the **Core Configuration** page (see "Core Configuration" on page 3–1) or the `ETH_SPEED` bit in the `command_config` register (see "MAC Configuration Register Space" on page 6–1).
- Control interface—an Avalon-MM slave port that provides access to 256 32-bit configuration and status registers, and statistics counters. This interface supports the use of `waitrequest` to stall the interconnect fabric for as many cycles as required.
- PHY management interface—implements the standard MDIO specification, IEEE 803.2 standard Clause 22, to access the PHY device management registers. This interface supports up to 32 PHY devices.

MAC variations without internal FIFO buffers implement the following additional interfaces:

- FIFO status interface—an Avalon-ST sink port that streams in the fill level of an external FIFO buffer. Only MAC variations without internal buffers implement this interface.
- Packet classification interface—an Avalon-ST source port that streams out receive packet classification information. Only MAC variations without internal buffers implement this interface.

For more information about the Avalon interfaces, refer to the *Avalon Interface Specifications*.

Refer to “Interface Signals” on page 7–1 for pinout diagrams and signal descriptions.

## Transmit Datapath

On the transmit path, the MAC function accepts frames from a user application and constructs Ethernet frames before forwarding them to the PHY. Depending on the MAC configuration, the MAC function could perform the following tasks: realigns the payload, modifies the source address, calculates and appends the CRC-32 field, and inserts interpacket gap (IPG) bytes. In half-duplex mode, the MAC function also detects collision and attempts to retransmit frames when a collision occurs. The following conditions trigger transmission:

- In MAC variations with internal FIFO buffers:
  - Cut-through mode—transmission starts when the level of the FIFO level hits the transmit section-full threshold.
  - Store and forward mode—transmission starts when a full packet is received.
- In MAC variations without internal FIFO buffers, transmission starts as soon as data is available on the Avalon-ST transmit interface.

For more information about Ethernet frame format, refer to Appendix B.

### IP Payload Re-alignment

If you turn the **Align packet headers to 32-bit boundaries** option, the MAC function removes the additional two bytes from the beginning of Ethernet frames. See “IP Payload Alignment” on page 4–11 for more information about IP payload alignment.

### Address Insertion

By default, the MAC function retains the source address received from the user application. You can configure the MAC function to replace the source address with the primary MAC address or any of the supplementary addresses by setting the `TX_ADDR_INS` bit in the `command_config` register to 1. The `TX_ADDR_SEL` bits in the `command_config` register determines the address selection. For more information about the bits, refer to Table 6–3 on page 6–6.

### Frame Payload Padding

The MAC function inserts padding bytes (`0x00`) when the payload length does not meet the minimum length required:

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

### CRC-32 Generation

To turn on CRC-32 generation, you must set the `OMIT_CRC` bit in the `tx_cmd_stat` register to 0 and send the frame to the MAC function with the `ff_tx_crc_fwd` signal deasserted.

The following equation shows the CRC polynomial, as specified in the IEEE 802.3 standard:

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

The 32-bit CRC value occupies the FCS field with $X^{31}$ in the least significant bit of the first byte. The CRC bits are thus transmitted in the following order: $X^{31}$, $X^{30}$,..., $X^{1}$, $X^{0}$.

### Interpacket Gap Insertion

In full-duplex mode, the MAC function maintains the minimum number of IPG configured in the `tx_ipg_length` register between transmissions. You can configure the minimum IPG to any value between 64 and 216 bit times, where 64 bit times is the time it takes to transmit 64 bits of raw data on the medium.

In half-duplex mode, the MAC function constantly monitors the line. Transmission starts only when the line has been idle for a period of 96 bit times and any backoff time requirements have been satisfied. In accordance with the standard, the MAC function begins to measure the IPG when the `m_rx_crs` signal is deasserted.

### Collision Detection in Half-Duplex Mode

Collision occurs only in a half-duplex network. It occurs when two or more nodes transmit concurrently. The PHY device asserts the `m_rx_col` signal to indicate collision.

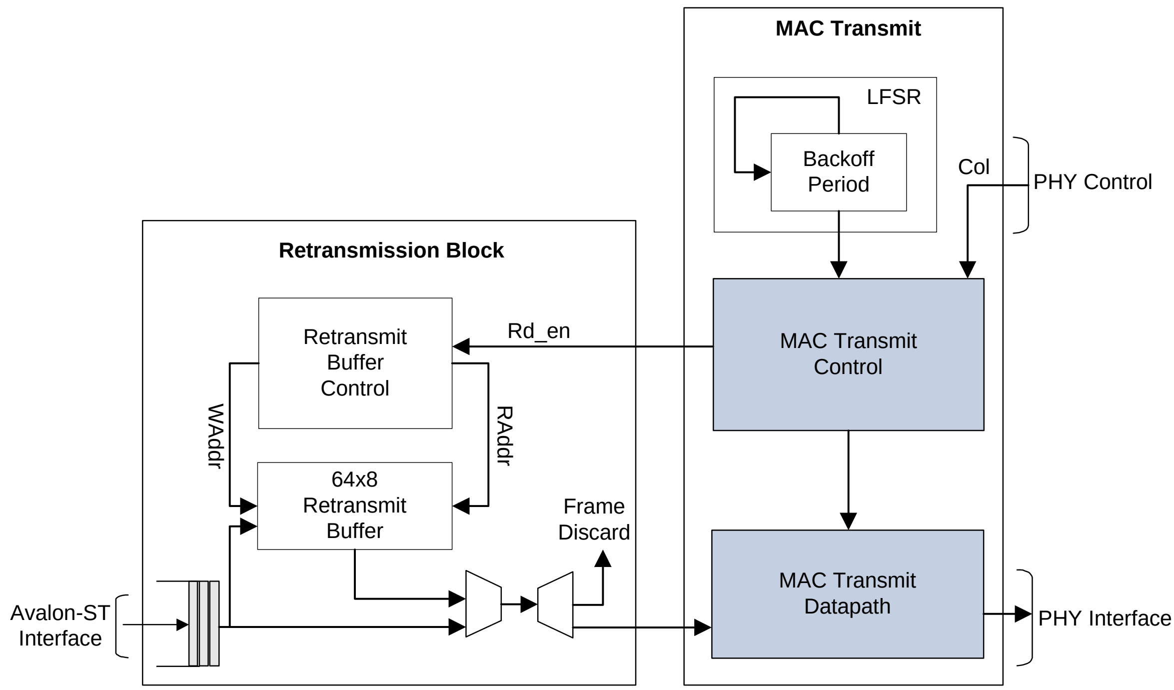
When the MAC function detects collision during transmission, it stops the transmission and sends a 32-bit jam pattern instead. A jam pattern is a fixed pattern, 0x648532A6, and is not compared to the CRC of the frame. The probability of a jam pattern to be identical to the CRC is very low, 0.532%.

If the MAC function detects collision while transmitting the preamble or SFD field, it sends the jam pattern only after transmitting the SFD field, which subsequently results in a minimum of 96-bit fragment.

If the MAC function detects collision while transmitting the first 64 bytes, including the preamble and SFD fields, the MAC function waits for an interval equal to the backoff period and then retransmits the frame. The frame is stored in a 64-byte retransmit buffer. The backoff period is generated from a pseudo-random process, truncated binary exponential backoff.

Figure 4–3 shows frame retransmission.

**Figure 4–3.** Frame Retransmission




The backoff time is a multiple of slot times. One slot is equal to a 512 bit times period. The number of the delay slot times, before the *Nth* retransmission attempt, is chosen as a uniformly distributed random integer in the following range:

$0 \leq r < 2^k$

$k = \min(n, N)$, where *n* is the number of retransmissions and *N = 10*.

For example, after the first collision, the backoff period, in slot time, is 0 or 1. If a collision occurs during the first retransmission, the backoff period, in slot time, is 0, 1, 2, or 3.

The maximum backoff time, in 512 bit times slots, is limited by *N* set to 10 as specified in the IEEE Standard 802.3.

If collision occurs after 16 consecutive retransmissions, the MAC function reports an excessive collision condition by setting the `EXCESS_COL` bit in the `command_config` register to 1, and discards the current frame from the transmit FIFO buffer.

In networks that violate standard requirements, collision may occur after the transmission of the first 64 bytes. If this happens, the MAC function stops transmitting the current frame, discards the rest of the frame from the transmit FIFO buffer, and resumes transmitting the next available frame.

## Receive Datapath

The MAC function receives Ethernet frames from the network via a PHY and forwards the payload with relevant frame fields to the user application after performing checks, filtering invalid frames, and removing the preamble and SFD.

For more information about Ethernet frame format, refer to Appendix B.

### Preamble Processing

The MAC function uses the SFD (0xD5) to identify the last byte of the preamble. If an SFD is not found after the seventh byte, the MAC function rejects the frame and discards it.

The IEEE standard specifies that frames must be separated by an interpacket gap (IPG) of at least 96 bit times. The MAC function, however, can accept frames with an IPG of less than 96 bit times; at least 48 and 64 bit times in RGMII/GMII (1000 Mbps operation) and RGMII/MII (10/100 Mbps operation) respectively.

The MAC function removes the preamble and SFD fields from valid frames.

### Collision Detection in Half-Duplex Mode

In half-duplex mode, the MAC function checks for collisions during frame reception. When collision is detected during the reception of the first 64 bytes, the MAC function discards the frame if the RX_ERR_DISC bit is set to 1. Otherwise, the MAC function forwards the frame to the user application with error.

### Address Checking

The MAC function can accept frames with the following address types:

- Unicast address—bit 0 of the destination address is 0.
- Multicast address—bit 0 of the destination address is 1.
- Broadcast address—all 48 bits of the destination address are 1.

The MAC function always accepts broadcast frames. If promiscuous mode is enabled (PROMIS_EN bit in the command_config register = 1), the MAC function omits address filtering and accepts all frames.

#### Unicast Address Checking

When promiscuous mode is disabled, the MAC function only accepts unicast frames if the destination address matches any of the following addresses:

- The primary address, configured in the registers mac_0 and mac_1
- The supplementary addresses, configured in the following registers: smac_0_0/smac_0_1, smac_1_0/smac_1_1, smac_2_0/smac_2_1 and smac_3_0/smac_3_1

Otherwise, the MAC function discards the frame.

### Multicast Address Resolution

You can use either a software program running on the host processor or a hardware multicast address resolution engine to resolve multicast addresses. Address resolution using a software program can affect the system performance, especially in gigabit mode.

The MAC function uses a 64-entry hash table in the register space, multicast hash table, to implement the hardware multicast address resolution engine as shown in Figure 4–4. The host processor must build the hash table according to the specified algorithm. A 6-bit code is generated from each multicast address by XORing the address bits as shown in Table 4–1 and Table 4–2 on page 4–9. This code represents the address of an entry in the hash table. Write one to the most significant bit in the table entry. All multicast addresses that hash to the address of this entry are valid and accepted. For more information about the multicast hash table, refer to "MAC Configuration Register Space" on page 6–1.

You can choose to generate the 6-bit code from all 48 bits of the destination address by setting the MHASH_SEL bit in the command_config register to 0, or from the lower 24 bits by setting the MHASH_SEL bit to 1. The latter option is provided if you want to omit the manufacturer's code, which typically resides in the upper 24 bits of the destination address, when generating the 6-bit code.

**Figure 4–4.** Hardware Multicast Address Resolution Engine

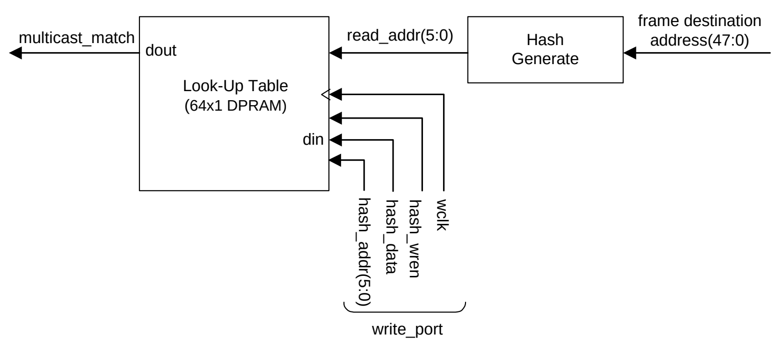



Table 4–1 shows the algorithm for generating the 6-bit code from the entire destination address.

**Table 4–1.** Hash Code Generation—Full Destination Address

| Hash Code Bit | Value |
|---|---|
| 0 | xor multicast MAC address bits 7:0 |
| 1 | xor multicast MAC address bits 15:8 |
| 2 | xor multicast MAC address bits 23:16 |
| 3 | xor multicast MAC address bits 31:24 |
| 4 | xor multicast MAC address bits 39:32 |
| 5 | xor multicast MAC address bits 47:40 |

Table 4–2 shows the algorithm for generating the 6-bit code from the lower 24 bits of the destination address.

**Table 4–2.** Hash Code Generation—Lower 24 Bits of Destination Address

| Hash Code Bit | Value |
|---|---|
| 0 | `xor` multicast MAC address bits 3:0 |
| 1 | `xor` multicast MAC address bits 7:4 |
| 2 | `xor` multicast MAC address bits 11:8 |
| 3 | `xor` multicast MAC address bits 15:12 |
| 4 | `xor` multicast MAC address bits 19:16 |
| 5 | `xor` multicast MAC address bits 23:20 |

The MAC function checks each multicast address received against the hash table, which serves as a fast matching engine, and a match is returned within one clock cycle. If there is no match, the MAC function discards the frame.

All multicast frames are accepted if all entries in the hash table are one.

### Frame Type Validation

The MAC function checks the length/type field to determine the frame type:

- Length/type < 0x600—the field represents the payload length of a basic Ethernet frame. The MAC function continues to check the frame and payload lengths.
- Length/type >= 0x600—the field represents the frame type.
  - Length/type = 0x8100—VLAN or stacked VLAN tagged frames. The MAC function continues to check the frame and payload lengths, and asserts the following signals:
    - for VLAN frames, `rx_err_stat[16]` in MAC variations with internal FIFO buffers or `pkt_class_data[1]` in MAC variations without internal FIFO buffers
    - for stacked VLAN frames, `rx_err_stat`[17] in MAC variations with internal FIFO buffers or `pkt_class_data[0]` in MAC variations without internal FIFO buffers.
  - Length/type = 0x8088—control frames. The next two bytes, the Opcode field, indicate the type of control frame.
    - For pause frames (Opcode = 0x0001), the MAC function continues to check the frame and payload lengths. For valid pause frames, the MAC function proceeds with pause frame processing (refer to "Remote Device Congestion" on page 4–17). The MAC function forwards pause frames to the user application only when the `PAUSE_FWD` bit in the `command_config` register is set to 1.
    - For other types of control frames, the MAC function accepts the frames and forwards them to the user application only when the `CNTL_FRM_ENA` bit in the `command_config` register is set to 1.
- For other field values, the MAC function forwards the receive frame to the user application.

### Payload Pad Removal

You can turn on padding removal by setting the PAD_EN bit in the command_config register to 1. The MAC function removes the padding, prior to forwarding the frames to the user application, when the payload length is less than the following values for the different frame types:

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

When padding removal is turned off, complete frames including the padding are forwarded to the Avalon-ST receive interface.

### CRC Checking

The following equation shows the CRC polynomial, as specified in the IEEE 802.3 standard:

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

The 32-bit CRC value occupies the FCS field with $X^{31}$ in the least significant bit of the first byte. The CRC bits are thus received in the following order: $X^{31}$, $X^{30}$,..., $X^{1}$, $X^{0}$.

If the MAC function detects CRC-32 error, it marks the frame invalid by asserting the following signals:

- rx_err[2] in MAC variations with internal FIFO buffers
- data_rx_error[1] in MAC variations without internal FIFO buffers

The MAC function discards frames with CRC-32 error if the RX_ERR_DISC bit in the command_config register is set to 1.

The MAC function forwards the CRC-32 field to the user application if the CRC_FWD and PAD_EN bits in the command_config register are 1 and 0 respectively. Otherwise, the CRC-32 field is removed from the frame.

### Length Checking

The MAC function checks the frame and payload lengths of basic, VLAN tagged, and stacked VLAN tagged frames.

The frame length must be at least 64 (0x40) bytes and not exceed the following maximum value for the different frame types:

- Basic frames—the value specified in the frm_length register
- VLAN tagged frames—the value specified in the frm_length register plus four
- Stacked VLAN tagged frames—the value specified in the frm_length register plus eight

To prevent FIFO buffer overflow, the MAC function truncates the frame if it is more than 11 bytes longer than the allowed maximum length.

For frames of a valid length, the MAC function continues to check the payload length if the `NO_LGTH_CHECK` bit in the `command_config` register is set to 0. The MAC function keeps track of the payload length as it receives a frame, and checks the length against the length/type field in basic MAC frames or the client length/type field in VLAN tagged frames. The payload length is valid if it satisfies the following conditions:

- The actual payload length matches the value in the length/type or client length/type field.
- Basic frames—the payload length is between 46 (0x2E)and 1536 (0x0600) bytes, excluding 1536.
- VLAN tagged frames—the payload length is between 42 (0x2A)and 1536 (0x0600), excluding 1536.
- Stacked VLAN tagged frames—the payload length is between 38 (0x26) and 1536 (0x0600), excluding 1536.

If the frame or payload length is not valid, the MAC function asserts one of the following signals to indicate length error:

- `rx_err[1]` in MACs with internal FIFO buffers.
- `data_rx_error[0]` in MACs without internal FIFO buffers.

### Frame Writing

The MegaCore function removes the preamble and SFD fields from the frame. The CRC field and padding bytes may be removed depending on the configuration.

For MAC variations with internal FIFO buffers, the MAC function writes the frame to the internal receive FIFO buffers.For MAC variations without internal FIFO buffers, it forwards the frame to the Avalon-ST receive interface.

MAC variations without internal FIFO buffers do not support backpressure on the Avalon-ST receive interface. In this variation, if the receiving component is not ready to receive data from the MAC function, the frame gets truncated with error and subsequent frames are also dropped with error.

### IP Payload Alignment

The network stack makes frequent use of the IP addresses stored in Ethernet frames. When you turn on the **Align packet headers to 32-bit boundaries** option, the MAC function aligns the IP payload on a 32-bit boundary by adding two bytes to the beginning of Ethernet frames. The padding of Ethernet frames are determined by the registers `tx_cmd_stat` and `rx_cmd_stat` on transmit and receive, respectively.

Table 4–3 illustrates the structure of a non-IP aligned Ethernet frame.

**Table 4–3.** 32-Bit Interface Data Structure — Non-IP aligned

| 31...24 | 23...16 | 15...8 | 7...0 |
|---|---|---|---|
| Byte 0 | Byte 1 | Byte 2 | Byte 3 |
| Byte 4 | Byte 5 | Byte 6 | Byte 7 |

Table 4–4 illustrates the structure of an IP aligned Ethernet frame.

**Table 4–4.** 32-Bit Interface Data Structure — IP aligned

| 31...24 | 23...16 | 15...8 | 7...0 |
|---|---|---|---|
| padded with zeros | | Byte 0 | Byte 1 |
| Byte 2 | Byte 3 | Byte 4 | Byte 5 |

## Transmit and Receive Latencies

Altera uses the following definitions for the transmit and receive latencies:

- Transmit latency is the number of clock cycles the MAC function takes to transmit the first bit on the network-side interface (MII/GMII/RGMII) after the bit was first available on the Avalon-ST interface.
- Receive latency is the number of clock cycles the MAC function takes to present the first bit on the Avalon-ST interface after the bit was received on the network-side interface (MII/GMII/RGMII).

Table 4–5 shows the transmit and receive nominal latencies in various modes. The FIFO buffer thresholds are set to the typical values specified in this user guide when deriving the latencies. See “Register Map” on page 6–3 for the threshold values.

**Table 4–5.** Transmit and Receive Nominal Latency

| MAC Configuration | Latency (Clock Cycles) *(1)* | |
|---|---|---|
| | Transmit | Receive |
| **MAC with Internal FIFO Buffers** *(2)* | | |
| GMII in cut-through mode | 32 | 110 |
| MII in cut-through mode | 41 | 218 |
| RGMII in gigabit and cut-through mode | 33 | 113 |
| RGMII in 10/100 Mbps and cut-through mode | 42 | 221 |
| **MAC without Internal FIFO Buffers** *(3)* | | |
| GMII | 11 | 37 |
| MII | 22 | 77 |
| RGMII in gigabit mode | 12 | 40 |
| RGMII in10/100 Mbps | 23 | 80 |

**Notes to Table 4–5:**

(1) The clocks in all domains are running at the same frequency.

(2) The data width is set to 32 bits.

(3) The data width is set to 8 bits.

## FIFO Buffer Thresholds

For MAC variations with internal FIFO buffers, you can change the operations of the FIFO buffers, and manage potential FIFO buffer overflow or underflow by configuring the following thresholds:

- Almost empty
- Almost full
- Section empty
- Section full

These thresholds are defined in bytes for 8-bit wide FIFO buffers and in words for 32-bit wide FIFO buffers.The FIFO buffer thresholds are configured via the registers.

### Receive Thresholds

Figure 4–5 shows the thresholds of the receive FIFO buffer.

**Figure 4–5.** Receive FIFO Thresholds

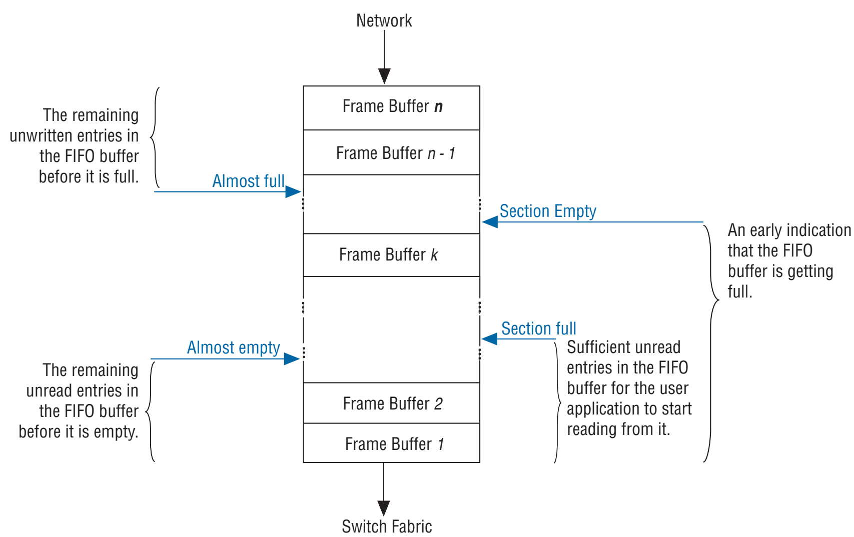


Table 4–6 describes the receive thresholds.

**Table 4–6.** Receive Thresholds

| Threshold | Register Name | Description |
|---|---|---|
| Almost empty | rx_almost_empty | The number of unread entries in the FIFO buffer before the buffer is empty. When the level of the FIFO buffer reaches this threshold, the MAC function asserts the ff_rx_a_empty signal. The MAC function stops reading from the FIFO buffer and subsequently stops transferring data to the user application to avoid buffer underflow.<br><br>When the MAC function detects an EOP, it transfers all data to the user application even if the number of unread entries is below this threshold. |
| Almost full | rx_almost_full | The number of unwritten entries in the FIFO buffer before the buffer is full. When the level of the FIFO buffer reaches this threshold, the MAC function asserts the ff_rx_a_full signal. If the user application is not ready to receive data (ff_rx_rdy = 0), the MAC function performs the following operations:<br>■ Stops writing data to the FIFO buffer.<br>■ Truncates received frames to avoid FIFO buffer overflow.<br>■ Asserts the rx_err[0] signal when the ff_rx_eop signal is asserted.<br>■ Marks the truncated frame invalid by setting the rx_err[3] signal to 1.<br><br>If the RX_ERR_DISC bit in the command_config register is set to 1 and the section-full (rx_section_full) threshold is set to 0, the MAC function discards frames with error received on the Avalon-ST interface. |
| Section empty | rx_section_empty | An early indication that the FIFO buffer is getting full. When the level of the FIFO buffer hits this threshold, the MAC function generates an XOFF pause frame to indicate FIFO congestion to the remote Ethernet device. When the FIFO level goes below this threshold, the MAC function generates an XON pause frame to indicate its readiness to receive new frames.<br><br>To avoid data loss, you can use this threshold as an early warning to the remote Ethernet device on the potential FIFO buffer congestion before the buffer level hits the almost-full threshold. The MAC function truncates receive frames when the buffer level hits the almost-full threshold. |
| Section full | rx_section_full | The section-full threshold indicates that there are sufficient entries in the FIFO buffer for the user application to start reading from it. The MAC function asserts the ff_rx_dsav signal when the buffer level hits this threshold.<br><br>Set this threshold to 0 to enable store and forward on the receive datapath. In the store and forward mode, the ff_rx_dsav signal remains deasserted. The MAC function asserts the ff_rx_dval signal as soon as a complete frame is written to the FIFO buffer. |

### Transmit Thresholds

Figure 4–6 shows the thresholds of the transmit FIFO buffer.

**Figure 4–6.** Transmit FIFO Thresholds

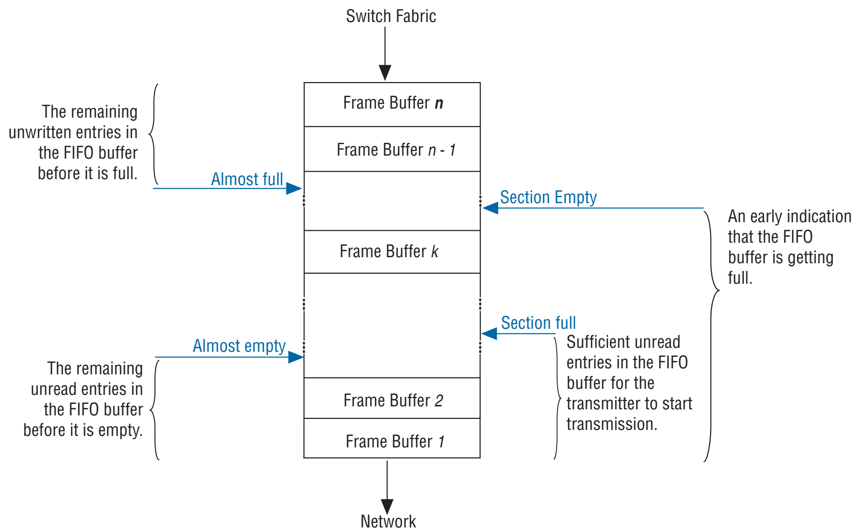



Table 4–7 describes the transmit thresholds.

**Table 4–7.** Transmit Thresholds

| Threshold | Register Name | Description |
|---|---|---|
| Almost empty | `tx_almost_empty` | The number of unread entries in the FIFO buffer before the buffer is empty. When the level of the FIFO buffer reaches this threshold, the MAC function asserts the `ff_tx_a_empty` signal. The MAC function stops reading from the FIFO buffer and sends the Ethernet frame with GMII / MII/ RGMII error to avoid FIFO underflow. |
| Almost full | `tx_almost_full` | The number of unwritten entries in the FIFO buffer before the buffer is full. When the level of the FIFO buffer reaches this threshold, the MAC function asserts the `ff_tx_a_full` signal. The MAC function deasserts the `ff_tx_rdy` signal to backpressure the Avalon-ST transmit interface. |
| Section empty | `tx_section_empty` | An early indication that the FIFO buffer is getting full. When the level of the FIFO buffer reaches this threshold, the MAC function deasserts the `ff_tx_septy` signal. This threshold can serve as a warning about potential FIFO buffer congestion. |
| Section full | `tx_section_full` | This threshold indicates that there are sufficient entries in the FIFO buffer to start frame transmission.<br>Set this threshold to 0 to enable store and forward on the transmit path. When you enable the store and forward mode, the MAC function forwards each frame as soon as it is completely written to the transmit FIFO buffer. |

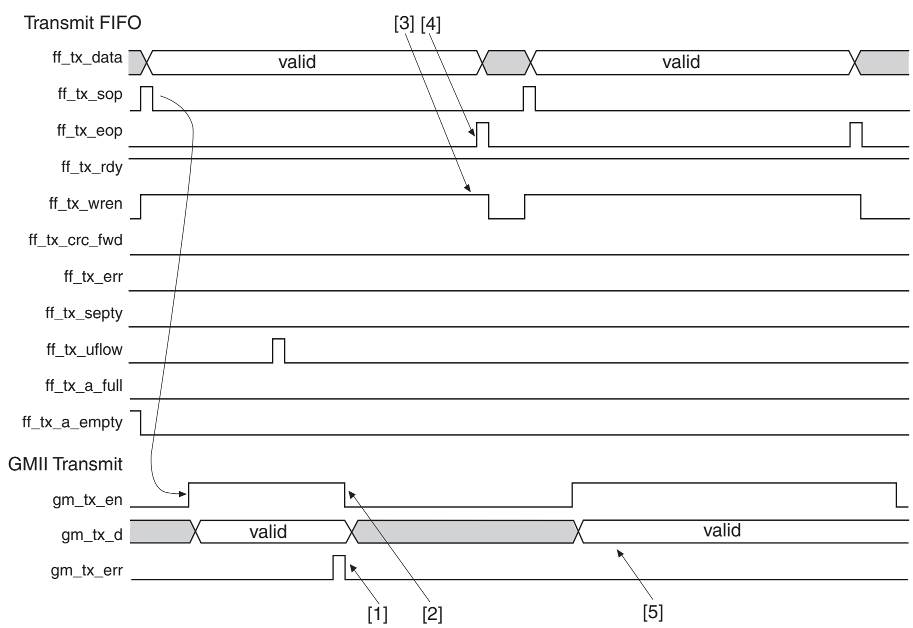
### Transmit FIFO Buffer Underflow

If the transmit FIFO buffer hits the almost-empty threshold during transmission and the FIFO buffer does not contain the end-of-packet indication, the MAC function stops reading data from the FIFO buffer and initiates the following actions:

1. The MAC function asserts the RGMII/GMII/MII error signals (`tx_control/gm_tx_err/m_tx_err`) to indicate that the fragment transferred is not valid.
2. The MAC function deasserts the RGMII/GMII/MII transmit enable signals (`tx_control/gm_tx_en/m_tx_en`) to terminate the frame transmission.
3. After the underflow, the user application completes the frame transmission.
4. The transmitter control discards any new data in the FIFO buffer until the end of frame is reached.
5. The MAC function starts to transfer data on the RGMII/GMII/MII when the user application sends a new frame with an SOP.

Figure 4–7 illustrates the FIFO buffer underflow protection algorithm for gigabit Ethernet system.

**Figure 4–7.** Transmit FIFO Buffer Underflow




## Congestion and Flow Control

In full-duplex mode, the MAC function implements flow control to manage the following types of congestion:

- Remote device congestion—the receiving device experiences congestion and requests the MAC function to stop sending data.
- Receive FIFO buffer congestion—when the receive FIFO buffer is almost full, the MAC function sends a pause frame to the remote device requesting the remote device to stop sending data.

- Local device congestion—any device connected to the MAC function, such as a processor, can request the remote device to stop data transmission.

For more information about pause frame format, refer to “Pause Frame Format” on page B–2.

### Remote Device Congestion

When the MAC function receives an XOFF pause frame and the `PAUSE_IGNORE` bit in the `command_config` register is set to 0, the MAC function completes the transfer of the current frame and stops transmission for the amount of time specified by the pause quanta in 512 bit times increments. Transmission resumes when the timer expires or when the MAC function receives an XON frame.

You can configure the MAC function to ignore pause frames by setting the `PAUSE_IGNORE` bit in the `command_config` register is set to 1.

### Receive FIFO Buffer and Local Device Congestion

Pause frames generated are compliant to the IEEE Standard 802.3 annex 31A & B. The MAC function generates pause frames when the level of the receive FIFO buffer hits a level that can potentially cause an overflow, or at the request of the user application. The user application can trigger the generation of an XOFF pause frame by setting the `XOFF_GEN` bit in the `command_config` register to 1 or asserting the `xoff_gen` signal.

For MAC variations with internal FIFO buffers, the MAC function generates an XOFF pause frame when the level of the FIFO buffer reaches the section-empty threshold (`rx_section_empty`). If transmission is in progress, the MAC function waits for the transmission to complete before generating the pause frame. The fill level of an external FIFO buffer is obtained via the Avalon-ST receive FIFO status interface.

When generating a pause frame, the MAC function fills the pause quanta bytes P1 and P2 with the value configured in the `pause_quant` register. The source address is set to the primary MAC address configured in the `mac_0` and `mac_1` registers, and the destination address is set to a fixed multicast address, 01-80-C2-00-00-01 (0x010000c28001).

The MAC function automatically generates an XON pause frame when the FIFO buffer section-empty flag is deasserted and the current frame transmission is completed. The user application can trigger the generation of an XON pause frame by clearing the `XOFF_GEN` bit and signal, and subsequently setting the `XON_GEN` bit to 1 or asserting the `XON_GEN` signal.

When generating an XON pause frame, the MAC function fills the pause quanta (payload bytes P1 and P2) with 0x0000 (zero quanta). The source address is set to the primary MAC address configured in the `mac_0` and `mac_1` registers and the destination address is set to a fixed multicast address, 01-80-C2-00-00-01 (0x010000c28001).

In addition to the flow control mechanism, the MAC function prevents an overflow by truncating excess frames. The status bit, `rx_err[3]`, is set to 1 to indicate such errors. The user application should subsequently discard these frames by setting the `RX_ERR_DISC` bit in the `command_config` register to 1.

## Magic Packets

A magic packet can be a unicast, multicast, or broadcast packet which carries a defined sequence in the payload section. Magic packets are received and acted upon only under specific conditions, typically in power-down mode.

The defined sequence is a stream of six consecutive 0xFF bytes followed by a sequence of 16 consecutive unicast MAC addresses. The unicast address is the address of the node to be awakened.

The sequence can be located anywhere in the magic packet payload and the magic packet is formed with a standard Ethernet header, optional padding and CRC.

### Sleep Mode

You can only put a node to sleep (set `SLEEP` bit in the `command_config` register to 1 and deassert the `magic_sleep_n` signal) if magic packet detection is enabled (set the `MAGIC_ENA` bit in the `command_config` register to 1).

☞ Altera recommends that you do not put a node to sleep if you disable magic packet detection.

Network transmission is disabled when a node is put to sleep. The receiver remains enabled, but it ignores all traffic from the line except magic packets to allow a remote agent to wake up the node. In the sleep mode, only `etherStatsPkts` and `etherStatsOctets` count the traffic statistics.

### Magic Packet Detection

Magic packet detection wakes up a node that was put to sleep. The MAC function detects magic packets with any of the following destination addresses:

- Any multicast address
- A broadcast address
- The primary MAC address configured in the `mac_0` and `mac_1` registers
- Any of the supplementary MAC addresses configured in the following registers if they are enabled: `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`

When the MAC function detects a magic packet, the `WAKEUP` bit in the `command_config` register is set to 1, and the `etherStatsPkts` and `etherStatsOctets` statistics registers are incremented.

Magic packet detection is disabled when the `SLEEP` bit in the `command_config` register is set to 0. Setting the `SLEEP` bit to 0 also resets the `WAKEUP` bit to 0 and resumes the transmit and receive operations.

## Local Loopback

You can enable local loopback on the MII/GMII/RGMII of the MAC function to exercise the transmit and receive paths. If you enable local loopback, use the same clock source for both the transmit and receive clocks. If you use different clock sources, ensure that the difference between the transmit and receive clocks is less than ±100 ppm.

To enable local loopback, perform the following steps:

1. Initiate software reset by setting the `SW_RESET` bit in `command_config` register to 1.

   Software reset disables the transmit and receive operations, flushes the internal FIFOs, and clears the statistics counters. The `SW_RESET` bit is automatically cleared upon completion.

2. When software reset is complete, enable local loopback on the MAC's MII/GMII/RGMII by setting the `LOOP_ENA` bit in `command_config` register to 1.
3. Enable transmit and receive operations by setting the `TX_ENA` and `RX_ENA` bits in `command_config` register to 1.
4. Initiate frame transmission.
5. Compare the statistics counters `aFramesTransmittedOK` and `aFramesReceivedOK` to verify that the transmit and receive frame counts are equal.
6. Check the statistics counters `ifInErrors` and `ifOutErrors` to determine the number of packets transmitted and received with errors.
7. To disable loopback, initiate a software reset and set the `LOOP_ENA` bit in `command_config` register to 0.

## Reset

A hardware reset resets all logic. A software reset only disables the transmit and receive paths, clears all statistics registers, and flushes the receive FIFO buffer. The values of configuration registers, such as the MAC address and thresholds of the FIFO buffers, are preserved during a software reset.

When you trigger a software reset, the MAC function sets the `TX_ENA` and `RX_ENA` bits in the `command_config` register to 0 to disable the transmit and receive paths. However, the transmit and receive paths are only disabled when the current frame transmission and reception complete.

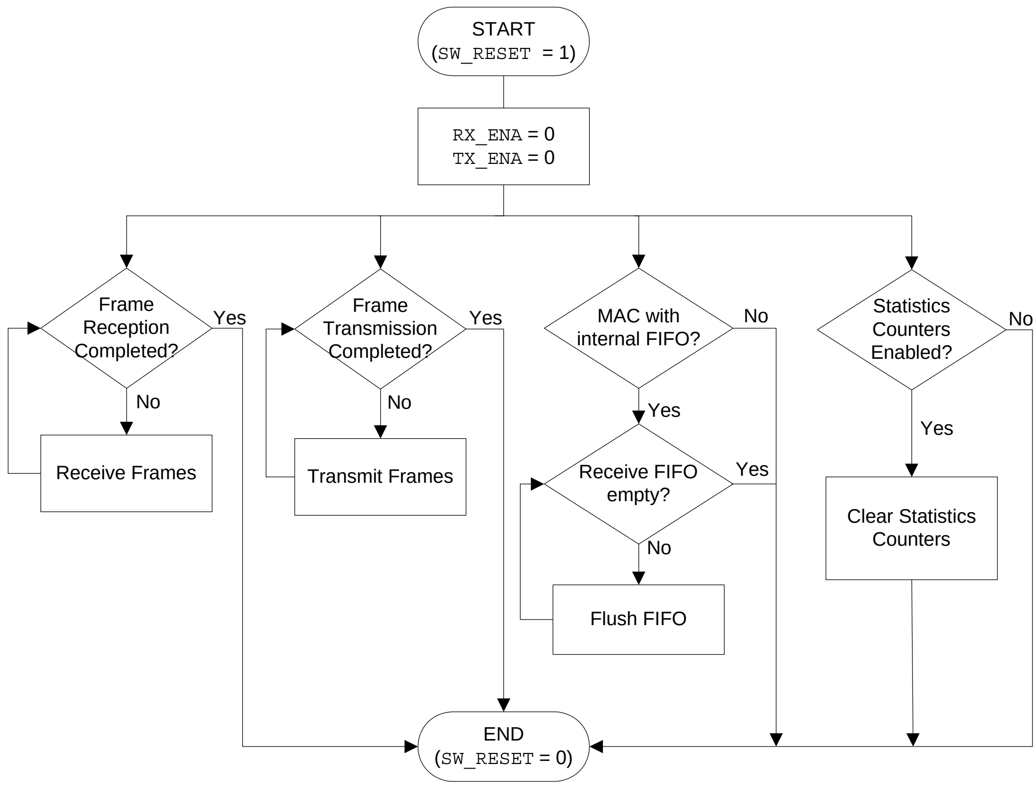
Do the following steps to trigger a reset:

- To trigger a hardware reset, assert the `reset` signal.
- To trigger a software reset, set the `SW_RESET` bit in the `command_config` register to 1. The `SW_RESET` bit is cleared automatically when the software reset ends.

☞ Altera recommends that you perform a software reset and wait for the software reset sequence to complete before changing the MAC operating speed and mode (full/half duplex). If you want to change the operating speed or mode without changing other configurations, preserve the `command_config` register before performing the software reset and restore the register after the changing the MAC operating speed or mode.

Figure 4–8 shows the sequence of a software reset.

**Figure 4–8.** Software Reset Sequence




If the `SW_RESET` bit is 1 when the line clocks are not available (for example, cable is disconnected), the statistics registers may not be cleared. The `read_timeout` register is then set to 1 to indicate that the statistics registers were not cleared.

## PHY Management (MDIO)

This module implements the standard MDIO specification, IEEE 803.2 standard Clause 22, to access the PHY device management registers, and supports up to 32 PHY devices.

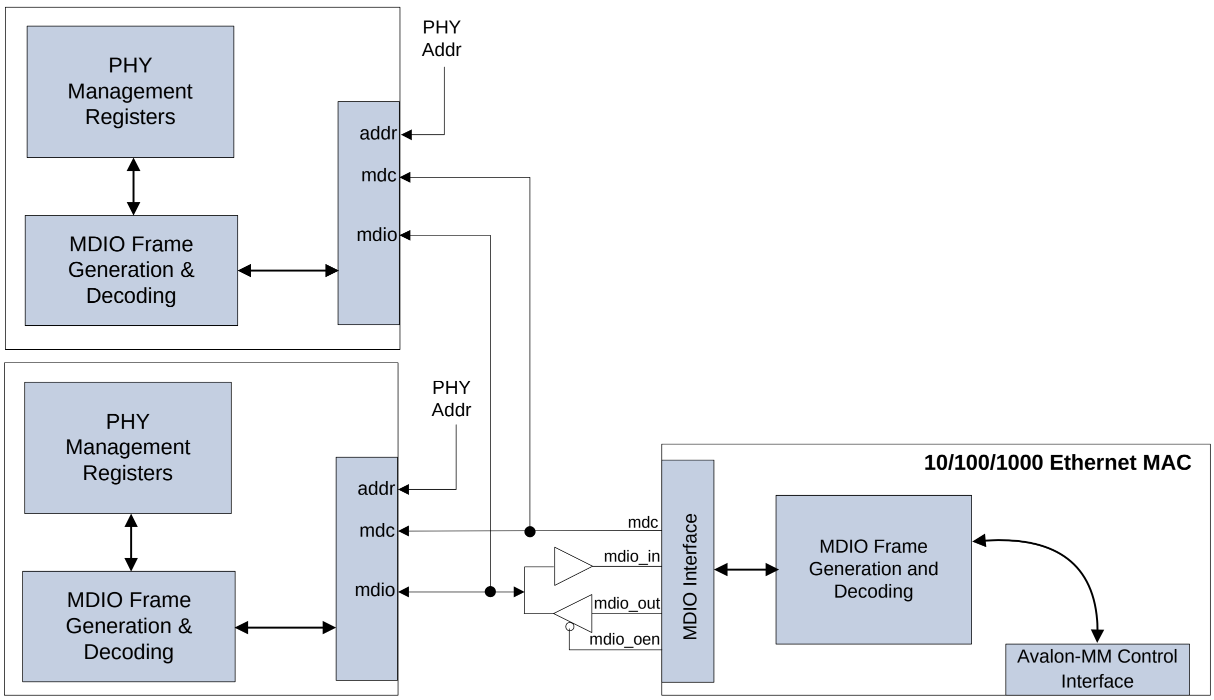
To access each PHY device, write the PHY address to the MDIO register (`mdio_addr0/1`) followed by the transaction data (MDIO Space 0/1). For faster access, the MAC function allows up to two PHY devices to be mapped in its register space at any one time. Subsequent transactions to the same PHYs do not require writing the PHY addresses to the register space thus reducing the transaction overhead. You can access the MDIO registers via the Avalon-MM interface. Refer to “MAC Configuration Register Space” on page 6–1 for more information about the MDIO registers.

For more information about the registers of a PHY device, refer to the specification provided with the device.

### MDIO Connection

Figure 4–9 shows the MDIO connection.

**Figure 4–9.** MDIO Interface




### MDIO Frame Format

The MDIO master communicates with the slave PHY device using MDIO 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`.

Table 4–8 lists the field settings for MDIO transactions.

**Table 4–8.** MDIO Frame Formats (Read/Write)

| Type | PRE | Command | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | ST<br>MSB LSB | OP<br>MSB LSB | Addr1<br>MSB LSB | Addr2<br>MSB LSB | TA | Data<br>MSB LSB | Idle |
| Read | 1 ... 1 | 01 | 10 | xxxxx | xxxxx | Z0 | xxxxxxxxxxxxxxxx | Z |
| Write | 1 ... 1 | 01 | 01 | xxxxx | xxxxx | 10 | xxxxxxxxxxxxxxxx | Z |

Table 4–9 describes the fields of an MDIO frame.

**Table 4–9.** MDIO Frame Field Descriptions (Part 1 of 2)

| Name | Description |
|---|---|
| PRE | Preamble. 32 bits of logical 1 sent prior to every transaction. |
| ST | Start indication. Standard MDIO (Clause 22): 0b01. |
| OP | Opcode. Defines the transaction type. |

**Table 4–9.** MDIO Frame Field Descriptions (Part 2 of 2)

| Name | Description |
|---|---|
| Addr1 | 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. For PHY device 1, the Addr1 field is set to the value configured in the `mdio_addr1` register. |
| Addr2 | 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 tri-stated. |

## Connecting MAC to External PHYs

The MAC function implements a flexible network interface—MII for 10/100-Mbps interfaces, RGMII or GMII for 1000-Mbps interfaces—that you can use in multiple applications. This section provides the guidelines for implementing the following network applications:

- Gigabit Ethernet operation
- Programmable 10/100 Ethernet operation
- Programmable 10/100/1000 Ethernet operation

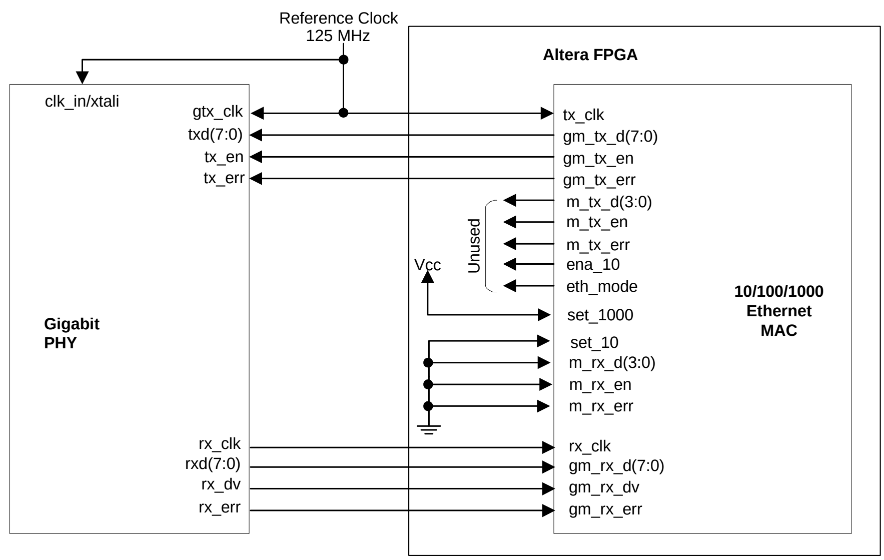
### Gigabit Ethernet

You can connect gigabit Ethernet PHYs to the MAC function via GMII or RGMII. On the receive path, connect the 125-MHz clock provided by the PHY device to the MAC clock, `rx_clk`. On transmit, drive a 125-MHz clock to the PHY GMII or RGMII. Connect a 125-MHz clock source to the MAC transmit clock, `tx_clk`.

A technology specific clock driver is required to generate a clock centered with the GMII or RGMII data from the MAC. The clock driver can be a PLL, a delay line or a DDR flip-flop.

Figure 4–10 shows how gigabit Ethernet PHYs are connected to the MAC function via GMII.

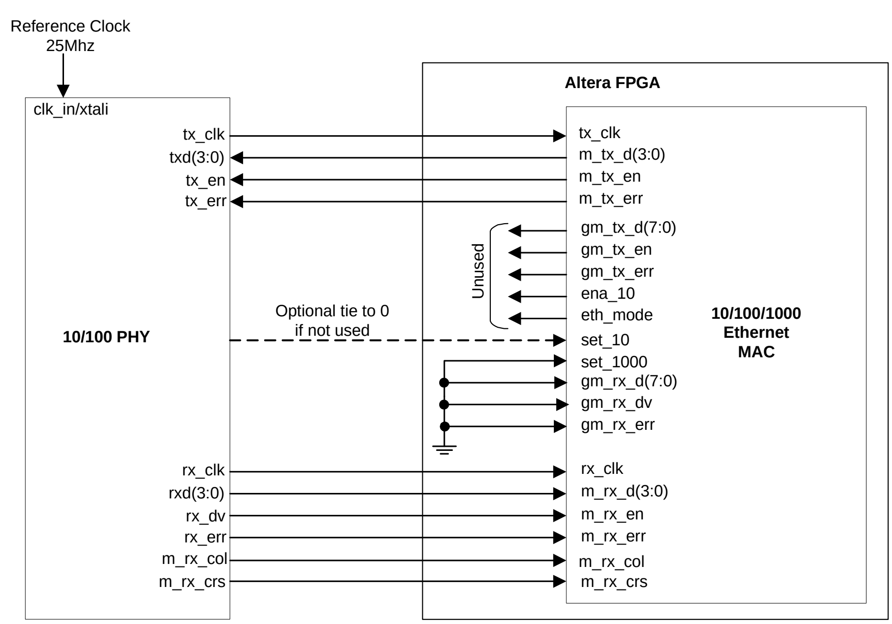
**Figure 4–10.** Gigabit PHY to MAC via GMII




## Programmable 10/100 Ethernet

Connect 10/100 Ethernet PHYs to the MAC function via MII. On the receive path, connect the 25-MHz (100 Mbps) or 2.5-MHz (10 Mbps) clock provided by the PHY device to the MAC clock, `rx_clk`. On the transmit path, connect the 25 MHz (100 Mbps) or a 2.5 MHz (10 Mbps) clock provided by the PHY to the MAC clock, `tx_clk`. Figure 4–11 shows the required connection for programmable 10/100 Ethernet operation.

**Figure 4–11.** 10/100 PHY Interface



## Programmable 10/100/1000 Ethernet Operation

Typically, 10/100/1000 Ethernet PHY devices implement a shared interface that you connect to a 10/100-Mbps MAC via MII/RGMII or to a gigabit MAC via GMII/RGMII.

On the receive path, connect the clock provided by the PHY device (2.5 MHz, 25 MHz or 125 MHz) to the MAC clock, `rx_clk`. The PHY interface is connected to both the MII (active PHY signals) and GMII of the MAC function.

On the transmit path, standard programmable PHY devices operating in 10/100 mode generate a 2.5 MHz (10 Mbps) or a 25 MHz (100 Mbps) clock. In gigabit mode, the PHY device expects a 125-MHz clock from the MAC function. Because the MAC function does not generate a clock output, an external clock module is introduced to drive the 125 MHz clock to the MAC function and PHY devices. In 10/100 mode, the clock generated by the MAC to the PHY can be tri-stated.

During transmission, the MAC control signal `eth_mode` selects either MII or GMII. The MAC function asserts the `eth_mode` signal when the MAC function operates in gigabit mode, which subsequently drives the MAC GMII to the PHY interface. The `eth_mode` signal is deasserted when the MAC function operates in 10/100 mode. In this mode, the MAC MII is driven to the PHY interface.

Figure 4–12 shows the required connection for programmable 10/100/1000 Ethernet operation via MII/GMII.

**Figure 4–12.** 10/100/1000 PHY Interface via MII/GMII

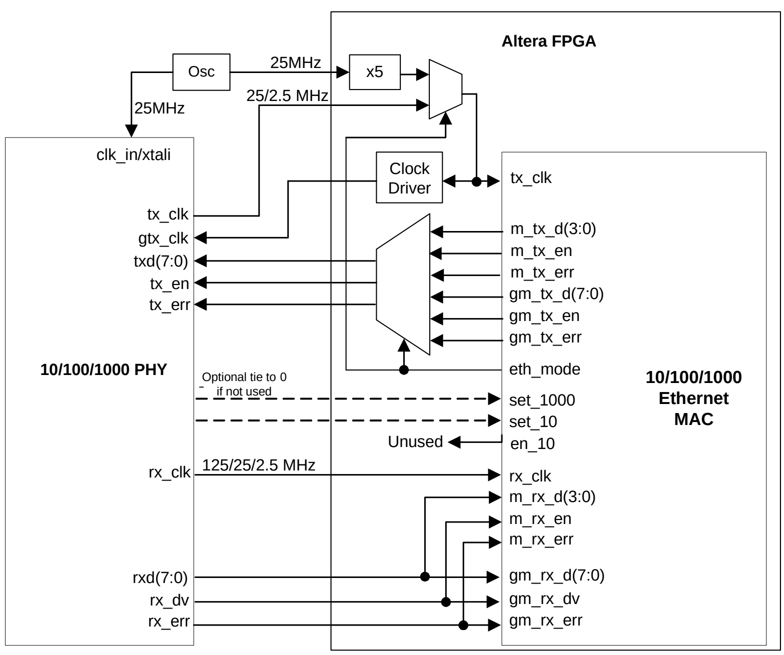
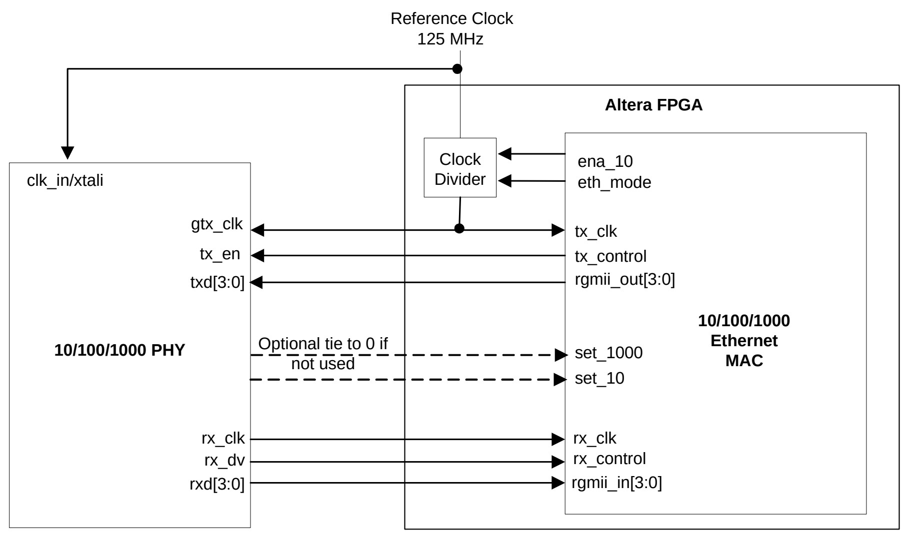


Figure 4–13 shows the required connection for programmable 10/100/1000 Ethernet operation via RGMII.

**Figure 4–13.** 10/100/1000 PHY Interface via RGMII



# 1000BASE-X/SGMII PCS With Optional Embedded PMA

The Altera 1000BASE-X/SGMII PCS function implements the functionality specified by IEEE 802.3 Clause 36. The PCS function is accessible via MII (SGMII) or GMII (1000BASE-X/SGMII). The PCS function interfaces to an on- or off-chip SERDES component via the industry standard ten-bit interface (TBI).

You can configure the PCS function to include an embedded physical medium attachment (PMA) with a a serial transceiver or LVDS I/O and soft CDR. The PMA interoperates with an external physical medium dependent (PMD) device, which drives the external copper or fiber network. The interconnect between Altera and PMD devices can be TBI or 1.25 Gbps serial.

The PCS function supports the following external PHYs:

- 1000 BASE-X PHYs as is.
- 10BASE-T, 100BASE-T and 1000BASE-T PHYs if the PHYs support SGMII.

## Architecture

Figure 4–14 shows a block diagram of the PCS function.

**Figure 4–14.** 1000BASE-X/SGMII PCS

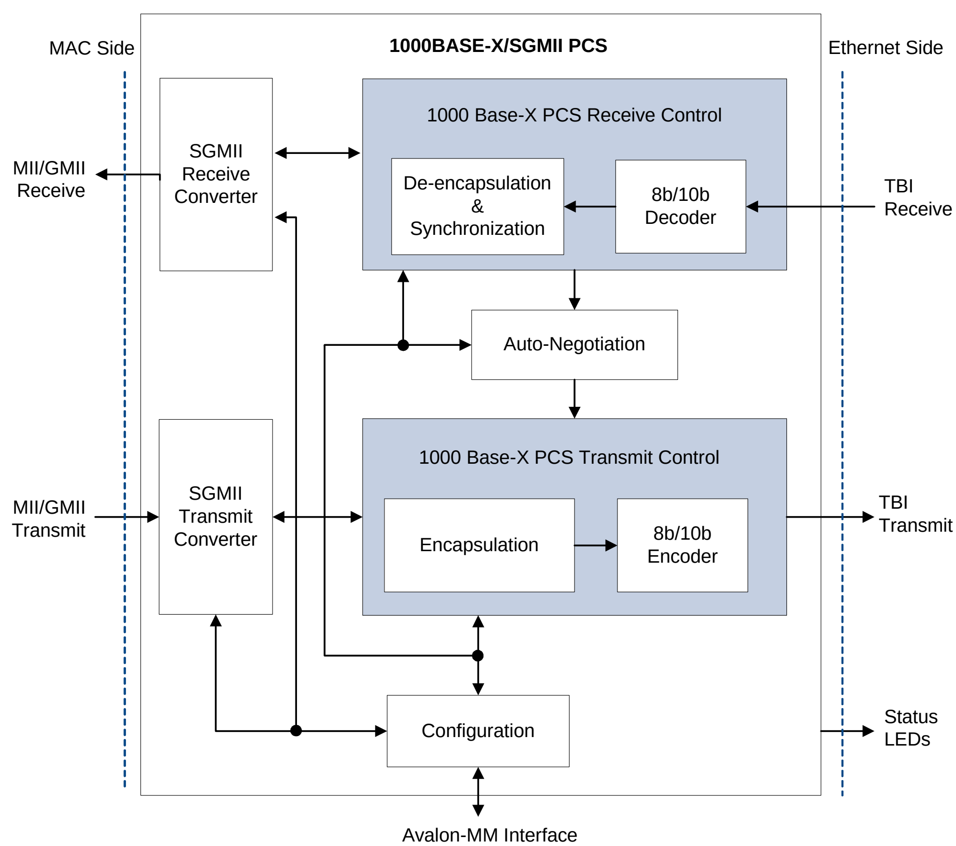
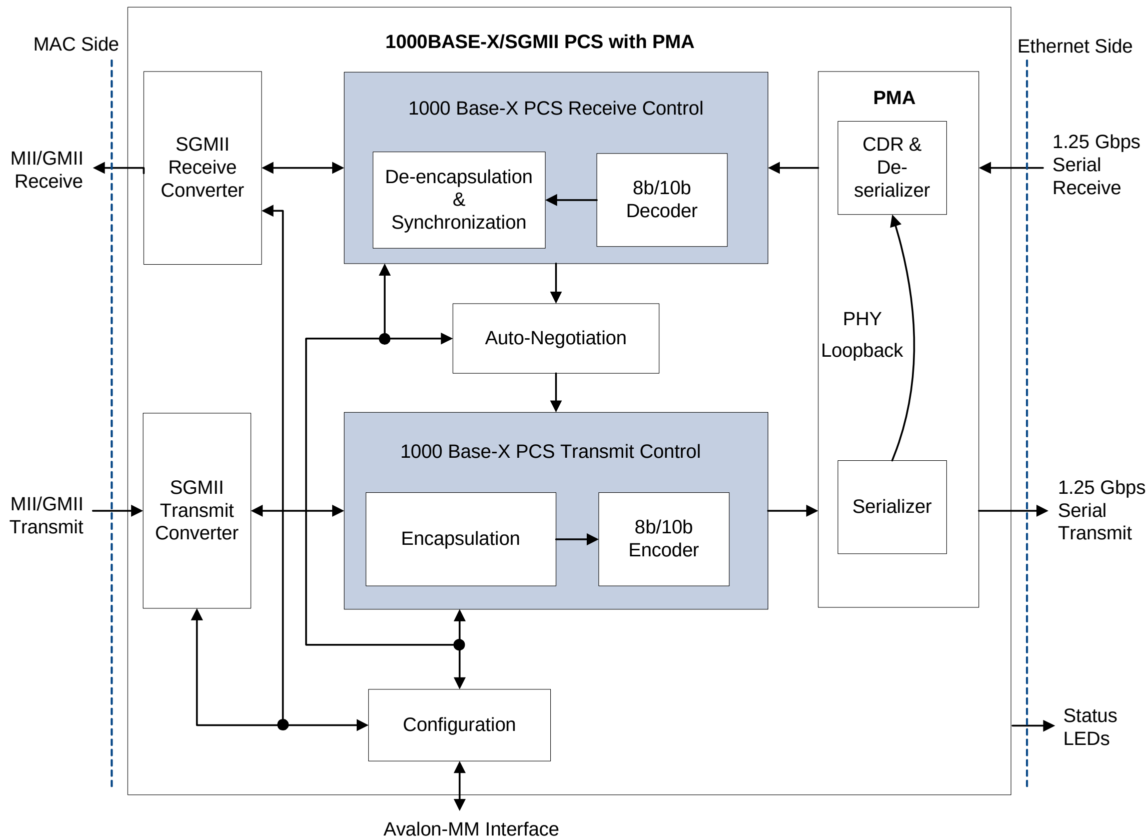


Figure 4–15 shows a block diagram of the PCS function with an embedded PMA.

**Figure 4–15.** 1000BASE-X/SGMII PCS with PMA




# Transmit Operation

This section describes the transmit operation, which includes frame encapsulation and encoding.

## Frame Encapsulation

The PCS function replaces the first preamble byte in the MAC frame with the start of frame /S/ symbol. Then, the PCS function encodes the rest of the bytes in the MAC frame with standard 8B/10B encoded characters. After the last FCS byte, the PCS function inserts the end of frame sequence, /T/ /R/ /R/ or /T/ /R/, depending on the number of character transmitted. Between frames, the PCS function transmits /I/ symbols.

If the PCS function receives a frame from the MAC function with an error (`gm_tx_err` asserted during frame transmission), the PCS function encodes the error by inserting a /V/ character.

## 8b/10b Encoding

The 8B/10B encoder maps 8-bit words to 10-bit symbols to generate a DC balance and ensure disparity of the stream with a maximum run length of 5.

## Receive Operation

This section describes the receive operation, which includes comma detection, decoding, de-encapsulation, synchronization, and carrier sense.

Configurations with PCS and embedded PMA can only support frame lengths longer than 16 Kbytes if the difference between the input reference clock and recovered clock is zero ppm.

### Comma Detection

The comma detection function searches for the 10-bit encoded comma character, K28.1/K28.5/K28.7, in consecutive samples received from PMA devices. When the K28.1/K28.5/K28.7 comma code group is detected, the PCS function realigns the data stream on a valid 10-bit character boundary. A standard 8b/10b decoder can subsequently decodes the aligned stream.

The comma detection function restarts the search for a valid comma character if the receive synchronization state machine loses the link synchronization.

### 8b/10b Decoding

The 8b/10b decoder performs the disparity checking to ensure DC balancing and produces a decoded 8-bit stream of data for the frame de-encapsulation function.

### Frame De-encapsulation

The frame de-encapsulation state machine detects the start of frame when the /I/ /S/ sequence is received and replaces the /S/ with a preamble byte (0x55). It continues decoding the frame bytes and transmits them to the MAC function. The /T/ /R/ /R/ or the /T/ /R/ sequence is decoded as an end of frame.

A /V/ character is decoded and sent to the MAC function as frame error. The state machine decodes sequences other than /I/ /I/ (Idle) or /I/ /S/ (Start of Frame) as wrong carrier.

During frame reception, the de-encapsulation state machine checks for invalid characters. When the state machine detects invalid characters, it indicates an error to the MAC function.

### Synchronization

The link synchronization constantly monitors the decoded data stream and determines if the underlying receive channel is ready for operation. The link synchronization state machine acquires link synchronization if the state machine receives three code groups with comma consecutively without error.

When link synchronization is acquired, the link synchronization state machine counts the number of invalid characters received. The state machine increments an internal error counter for each invalid character received and incorrectly positioned comma character. The internal error counter is decremented when four consecutive valid characters are received. When the counter reaches 4, the link synchronization is lost.

The PCS function drives the `led_link` signal to 1 when link synchronization is acquired. This signal can be used as a common visual activity check using a board LED.

### Carrier Sense

The carrier sense state machine detects an activity when the link synchronization is acquired and when the transmit and receive encapsulation or de-encapsulation state machines are not in the idle or error states.

The carrier sense state machine drives the `mii_rx_crs` and `led_crs` signals to 1 when it detects an activity. The `led_crs` signal can be used as a common visual activity check using a board LED.

### Collision Detection

A collision happens when non-idle frames are received from the PHY and transmitted to the PHY simultaneously. Collisions can be detected only in SGMII and half-duplex mode.

When a collision happens, the collision detection state machine drives the `mii_rx_col` and `led_col` signals to 1. You can use the `led_col` signal as a visual check using a board LED.

## Transmit and Receive Latencies

Altera uses the following definitions for the transmit and receive latencies for the PCS function with an embedded PMA:

- Transmit latency is the time the PCS function takes to transmit the first bit on the PMA-PCS interface after the bit was first available on the MAC side interface (MII/GMII).
- Receive latency is the time the PCS function takes to present the first bit on the MAC side interface (MII/GMII) after the bit was received on the PMA-PCS interface.

Table 4–10 shows the transmit and receive latencies for the PCS function with an embedded PMA.

☞ These latencies are derived from a simulation. For transceiver latency, refer to the transceiver handbook of the respective device family.

**Table 4–10.** PCS Transmit and Receive Latency (Part 1 of 2)

| PCS Configuration | Latency (ns) | |
|---|---|---|
| | **Transmit** | **Receive** |
| **PCS with GX transceivers** | | |
| 10-Mbps SGMII | 3368 | 2489 |
| 100-Mbps SGMII | 488 | 335 |
| 1000-Mbps SGMII | 184 | 135 |
| 1000BASE-X | 24 | 40 |
| **PCS with LVDS Soft-CDR I/O** | | |
| 10-Mbps SGMII | 3600 | 2344 |
| 100-Mbps SGMII | 440 | 344 |

**Table 4–10.** PCS Transmit and Receive Latency (Part 2 of 2)

| PCS Configuration | Latency (ns) | |
|---|---|---|
| | Transmit | Receive |
| 1000-Mbps SGMII | 192 | 184 |
| 1000BASE-X | 40 | 104 |

## SGMII Converter

You can enable the SGMII converter by setting the SGMII_ENA bit in the if_mode register to 1. When enabled and the USE_SGMII_AN bit in the if_mode register is set to 1, the SGMII converter is automatically configured with the capabilities advertised by the PHY. Otherwise, Altera recommends that you configure the SGMII converter with the SGMII_SPEED bits in the if_mode register.

In 1000BASE-X mode, the PCS function always operates in gigabit mode and data duplication is disabled.

### Transmit

In gigabit mode, the PCS and MAC functions must operate at the same rate. The transmit converter transmits each byte from the MAC function once to the PCS function.

In 100-Mbps mode, the transmit converter replicates each byte received by the PCS function 10 times. In 10 Mbps, the transmit converter replicates each byte transmitted from the MAC function to the PCS function 100 times.

### Receive

In gigabit mode, the PCS and MAC functions must operate at the same rate. The transmit converter transmits each byte from the PCS function once to the MAC function.

In 100-Mbps mode, the receive converter transmits one byte out of 10 bytes received from the PCS function to the MAC function. In 10-Mbps, the receive converter transmits one byte out of 100 bytes received from the PCS function to the MAC function.

## Auto-Negotiation

Auto-negotiation is an optional function that can be started when link synchronization is acquired during system start up. To start auto-negotiation automatically, set the AUTO_NEGOTIATION_ENABLE bit in the PCS control register to 1. During auto-negotiation, the PCS function advertises its device features and exchanges them with a link partner device.

If the SGMII_ENA bit in the if_mode register is set to 0, the PCS function operates in 1000BASE-X. Otherwise, the operating mode is SGMII. The following sections describe the auto-negotiation process for each operating mode.

For more information about PCS registers, refer to "PCS Configuration Register Space" on page 6–15.

☞ When simulating your design, you can disable auto-negotiation to reduce the simulation time. If you enable auto-negotiation in your design, set the `link_timer` time to a smaller value to reduce the auto-negotiation link timer in the simulation.

### 1000BASE-X Auto-Negotiation

When link synchronization is acquired, the PCS function starts sending a /C/ sequence (configuration sequence) to the link partner device with the advertised register set to 0x00. The sequence is sent for a time specified in the PCS `link_timer` register mapped in the PCS register space.

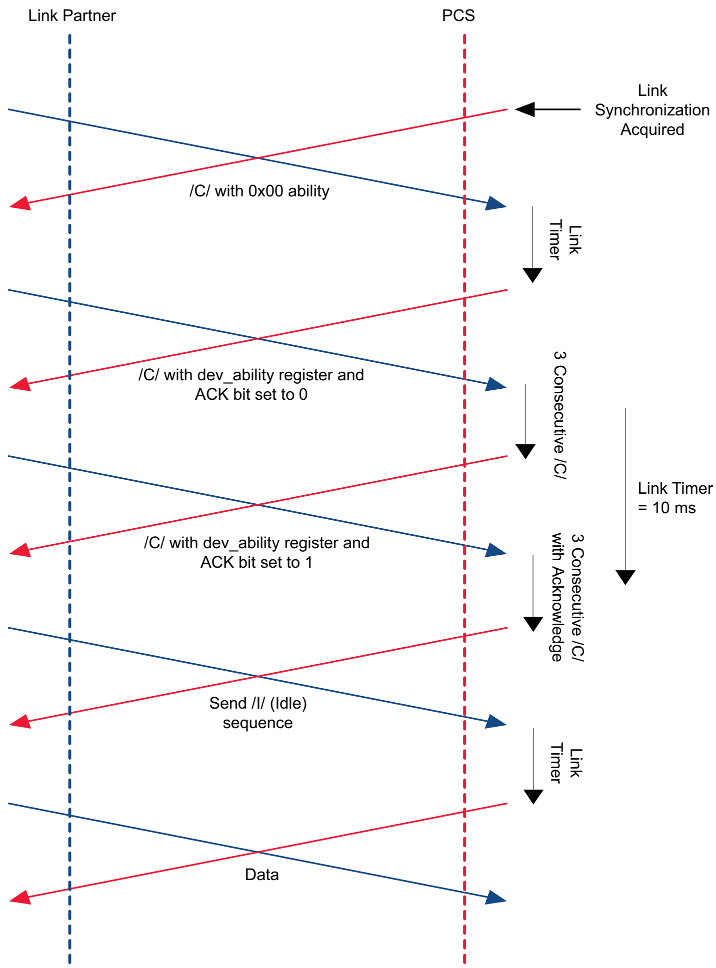
When the `link_timer` time expires, the PCS `dev_ability` register is advertised, with the `ACK` bit set to 0 for the link partner. The auto-negotiation state machine checks for three consecutive /C/ sequences received from the link partner.

The auto-negotiation state machine then sets the `ACK` bit to 1 in the advertised `dev_ability` register and checks if three consecutive /C/ sequences are received from the link partner with the `ACK` bit set to 1.

Auto-negotiation waits for the value configured in the `link_timer` register to ensure no more consecutive /C/sequences are received from the link partner. The auto-negotiation is successfully completed when three consecutive idle sequences are received after the link timer expires.

Figure 4–16 shows the sequence of the activities.

**Figure 4–16.** Auto-negotiation Simplified




Once auto-negotiation is successfully completed, the ability advertised by the link partner device is available in the `partner_ability` register and the `AUTO_NEGOTIATION_COMPLETE` bit in the `status` register is set to 1.

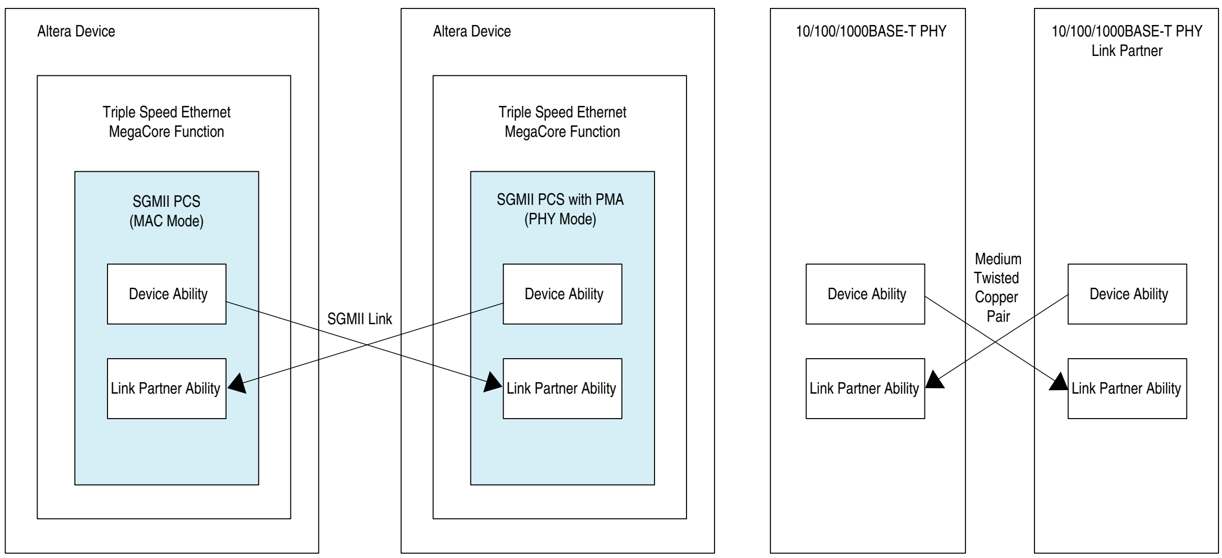
The PCS function restarts auto-negotiation when link synchronization is lost and re-acquired, or when you set the `RESTART_AUTO_NEGOTIATION` bit in the PCS `control` register to 1.

## SGMII Auto-Negotiation

In SGMII mode, the capabilities of the PHY device are advertised and exchanged with a link partner PHY device.

Figure 4–17 shows the possible application of SGMII auto-negotiation in MAC mode and PHY mode.

**Figure 4–17.** SGMII Auto-Negotiation in MAC Mode and PHY Mode



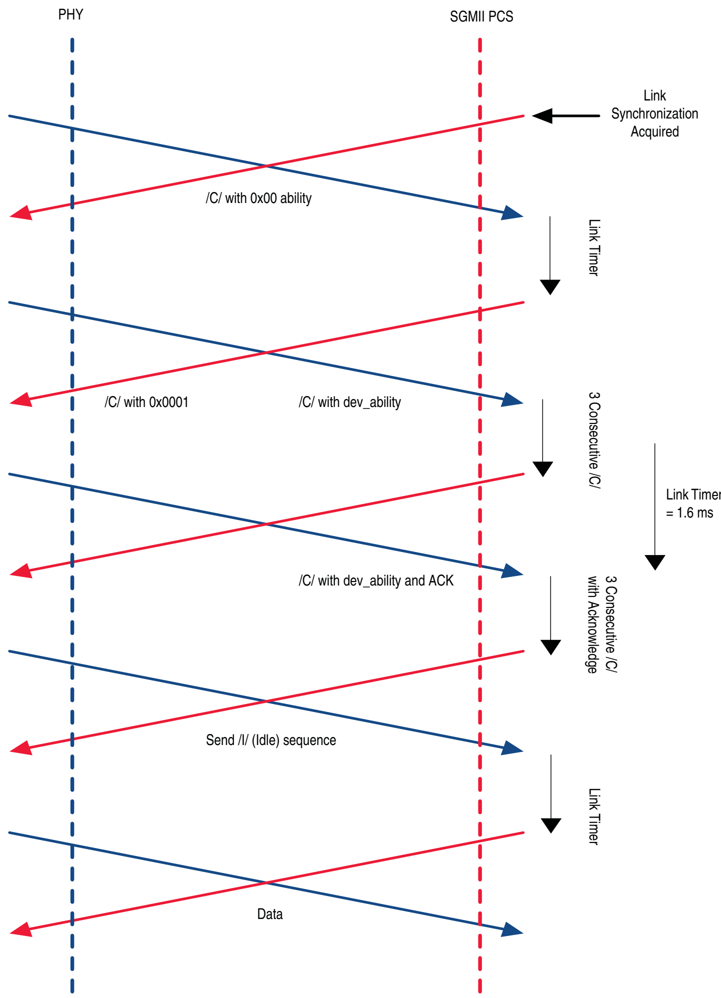

If the SGMII_ENA and USE_SGMII_AN bits in the if_mode register are 1, the PCS function is automatically configured with the capabilities advertised by the PHY device once the auto-negotiation completes.

If the SGMII_ENA bit is 1 and the USE_SGMII_AN bit is 0, the PCS function can be configured with the SGMII_SPEED and SGMII_DUPLEX bits in the if_mode register.

If the SGMII_ENA bit is 1 and the SGMII_AN_MODE bit is 1 (SGMII PHY Mode auto-negotiation is enabled) the speed and duplex mode resolution will be resolved based on the value that you set in the dev_ability register once auto negotiation is done. You should use set to the PHY mode if you want to advertise the link speed and duplex mode to the link partner.

Figure 4–16 shows the sequence of the activities.

**Figure 4–18.** SGMII Auto-negotiation



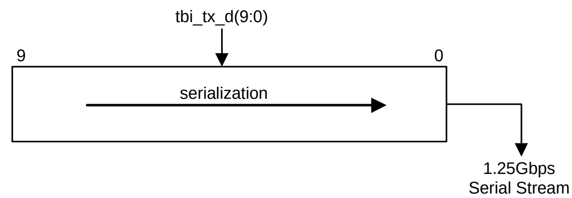

For more information, refer to *CISCO Serial-GMII Specifications*.

## Ten-bit Interface

In PCS variations with embedded PMA, the PCS function implements a TBI to an external SERDES.

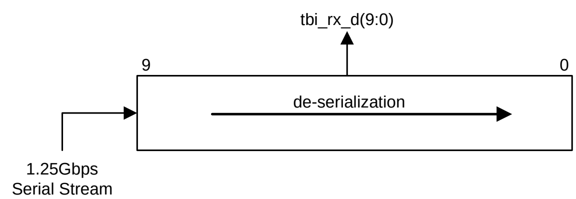
On transmit, the SERDES must serialize `tbi_tx_d[0]`, the least significant bit of the TBI output bus first and `tbi_tx_d[9]`, the most significant bit of the TBI output bus last to ensure the remote node receives the data correctly, as Figure 4–19 illustrates.

**Figure 4–19.** SERDES Serialization Overview




On receive, the SERDES must serialize the TBI least significant bit first and the TBI most significant bit last, as Figure 4–20 illustrates.

**Figure 4–20.** SERDES De-Serialization Overview



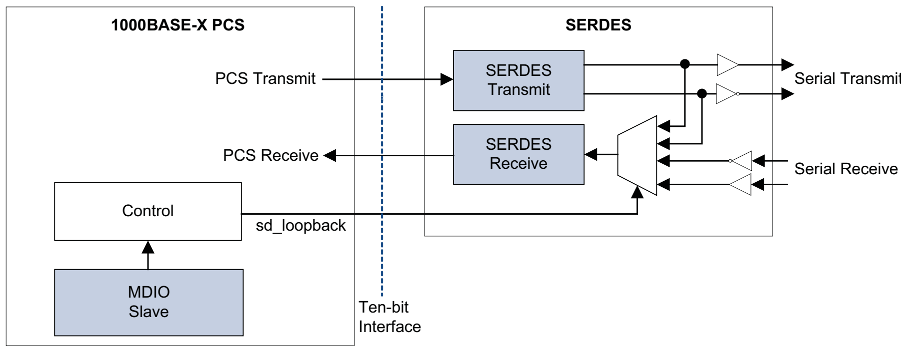

## PHY Loopback

In PCS variations with embedded PMA targeting devices with GX transceivers, you can enable loopback on the serial interface to test the PCS and embedded PMA functions in isolation of the PMD. To enable loopback, set the `sd_loopback` bit in the PCS `control` register to 1.

☞ The serial loopback option is not supported in Cyclone IV devices with GX transceiver.

Figure 4–20 illustrates the serial loopback.

**Figure 4–21.** Serial Loopback

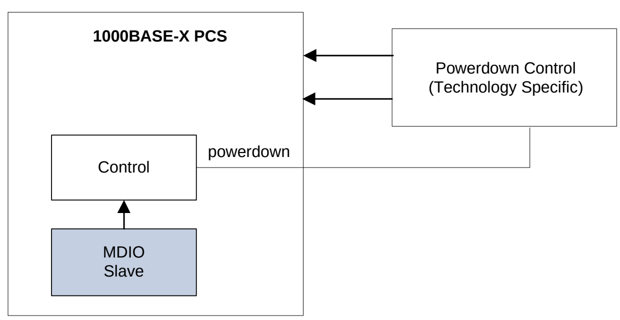



## PHY Power-Down

Power-down is controlled by the `POWERDOWN` bit in the PCS `control` register. When the system management agent enables power-down, the PCS function drives the `powerdown` signal, which can be used to control a technology specific circuit to switch off the PCS function clocks to reduce the application activity.

When the PHY is in power-down state, the PCS function is in reset and any activities on the GMII transmit and the TBI receive interfaces are ignored. The management interface remains active and responds to management transactions from the MAC layer device.

**Figure 4–22.** Power-Down



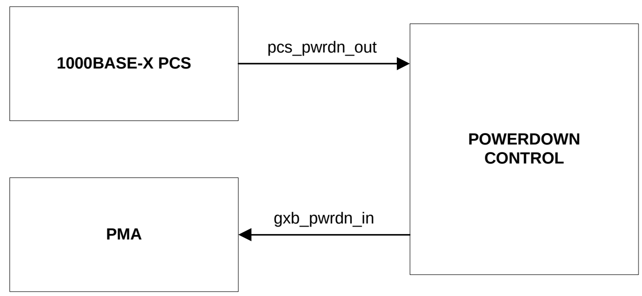

### Power-Down in PCS Variations with Embedded PMA

In PCS variations with embedded PMA targeting devices with GX transceivers, the power-down signal is internally connected to the power-down of the GX transceiver. In these devices, the power-down functionality is shared across quad-port transceiver blocks. Ethernet designs must share a common `gbx_pwrdn_in` signal to use the same quad-port transceiver block.

For designs targeting devices other than Stratix V, you can export the power-down signals to implement your own power-down logic to efficiently use the transceivers within a particular transceiver quad. Turn on the **Export transceiver powerdown signal** parameter to export the signals. For more information about the parameter, refer to “PCS/Transceiver Options” on page 3–5.

Figure 4–23 shows the power-down control with exported power-down signal.

**Figure 4–23.** Power-Down with Export Transceiver Power-Down Signal



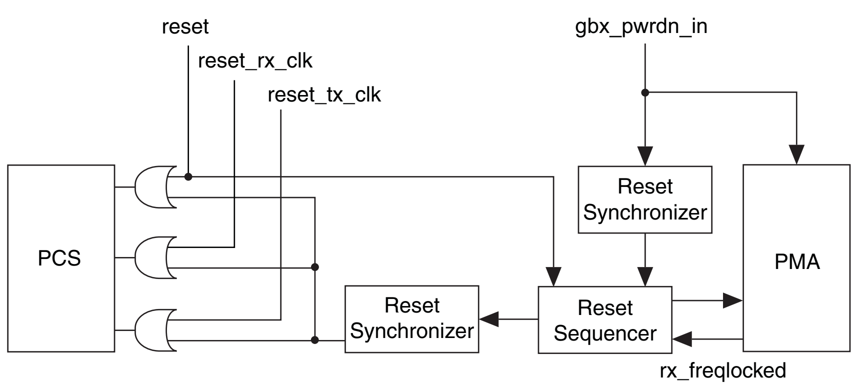

## Reset

A hardware reset resets all logic synchronized to the respective clock domains whereas a software reset only resets the PCS state machines, comma detection function, and 8B10B encoder and decoder. To trigger a hardware reset on the PCS, assert the respective reset signals: `reset_reg_clk`, `reset_tx_clk`, and `reset_rx_clk`. To trigger a software reset, set the `RESET` bit in the `control` register to 1.

In PCS variations with embedded PMA, assert the respective reset signals or the power-down signal to trigger a hardware reset. You must assert the `reset` signal subsequent to asserting the `reset_rx_clk`, `reset_tx_clk`, or `gbx_pwrdn_in` signal. The reset sequence is also initiated when the active-low `rx_freqlocked` signal goes low.

Figure 4–24 shows the reset distribution in the PCS with embedded PMA variation.

**Figure 4–24.** Reset Distribution in PCS with Embedded PMA



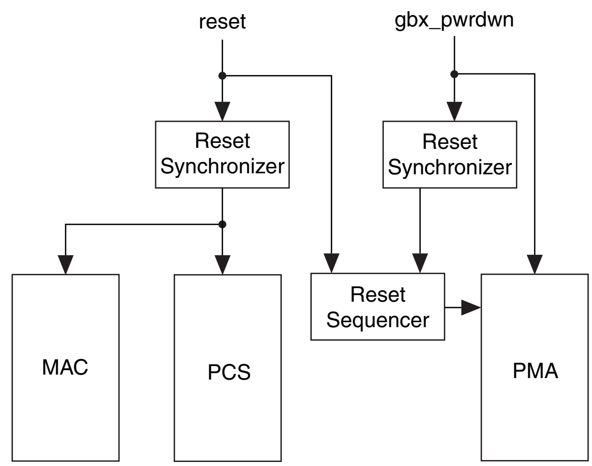

☞ For more information about the `rx_freqlocked` signal and transceiver reset, refer to the transceiver handbook of the respective device family.

Assert the `reset` or `gxb_pwrdn_in` signals to perform a hardware reset on MAC with PCS and embedded PMA variation.

You must assert the `reset` signal for at least three clock cycles.

Figure 4–25 shows the reset distribution in the MAC with PCS and embedded PMA variation.

**Figure 4–25.** Reset Distribution in MAC with PCS and Embedded PMA




# Altera IEEE 1588v2 Feature

The Altera IEEE 1588v2 feature provides timestamp for receive and transmit frames in the Triple-Speed Ethernet MegaCore function designs. The feature consists of Precision Time Protocol (PTP). PTP is a layer-3 protocol that accurately synchronizes all real time-of-day clocks in a network to a master clock.

## Supported Configurations

The Triple-Speed Ethernet MegaCore functions support the IEEE 1588v2 feature only in the following configurations:

- 1000-Mbps MAC with SGMII PCS and embedded serial PMA
- 10/100/1000-Mbps MAC with 1000BASE-X/SGMII PCS
- No FIFO
- Full-duplex mode

## Features

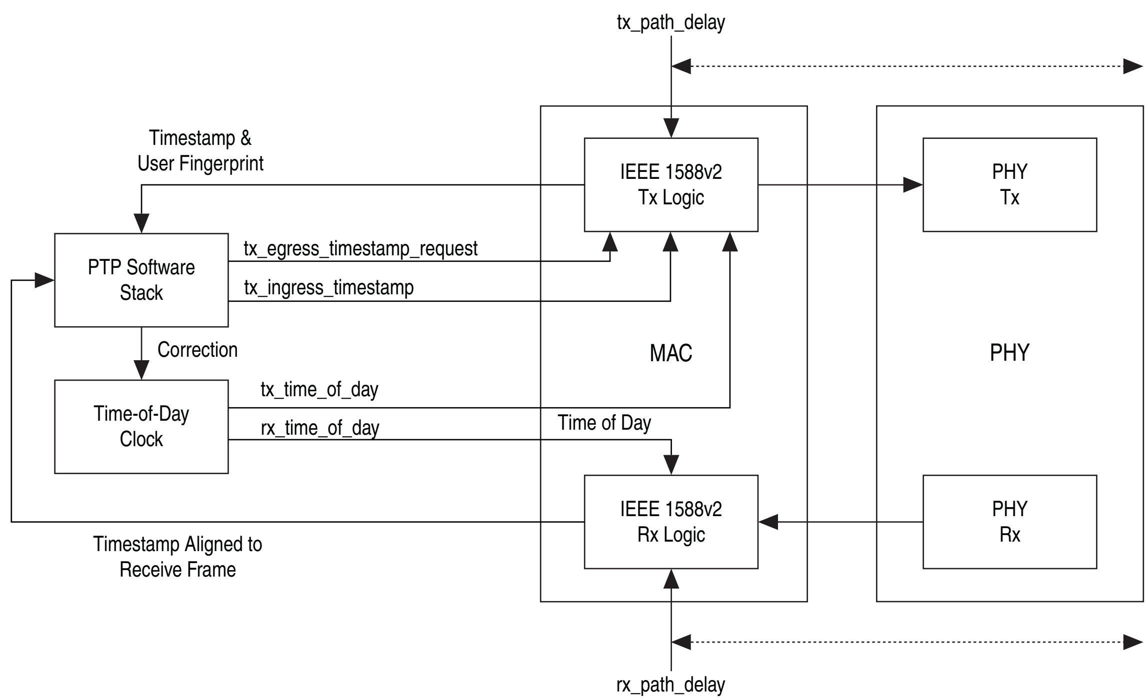
This section describes the features of the IEEE 1588v2.

- Supports 4 types of PTP clock on the transmit datapath:
  - Master and slave ordinary clock
  - Master and slave boundary clock
  - End-to-end (E2E) transparent clock
  - Peer-to-peer (P2P) transparent clock
- Supports PTP message types:
  - PTP event messages—Sync, Delay_Req, Pdelay_Req, and Pdelay_Resp.
  - PTP general messages—Follow_Up, Delay_Resp, Pdelay_Resp_Follow_Up, Announce, Management, and Signaling.
- Supports simultaneous 1-step and 2-step clock synchronizations on the transmit datapath.
  - 1-step clock synchronization—The MAC function inserts accurate timestamp in Sync PTP message or updates the correction field with residence time.
  - 2-step clock synchronization—The MAC function provides accurate timestamp and the related fingerprint for all PTP message.
- Supports the following PHY operating speed accuracy:
  - random error:
    - 10Mbps—NA
    - 100Mbps—timestamp accuracy of ± 5 ns
    - 1000Mbps—timestamp accuracy of ± 2 ns
  - static error—timestamp accuracy of ± 3 ns
- Supports IEEE 802.3, UDP/IPv4, and UDP/IPv6 transfer protocols for the PTP frames.
- Supports untagged, VLAN tagged, Stacked VLAN Tagged PTP frames, and any number of MPLS labels.
- Supports configurable register for timestamp correction on both transmit and receive datapaths.
- Supports Time-of-Day (ToD) clock that provides a stream of 64-bit and 96-bit timestamps. For more information about the ToD clock, refer to refer to Appendix D, Time-of-Day Clock.

## Architecture

Figure 4–26 shows the overview of the IEEE 1588v2 feature.

**Figure 4–26.** Overview of the IEEE 1588v2 Feature *(Note 1)*




**Note to Figure 4–26:**

(1) This figure shows only the datapaths related to the IEEE 1588v2 feature.

## Transmit Datapath

The IEEE 1588v2 feature supports 1-step and 2-step clock synchronizations on the transmit datapath.

- For 1-step clock synchronization,
  - Timestamp insertion depends on the PTP device and message type.
  - The MAC function inserts a timestamp in the Sync PTP message if the PTP clock operates as ordinary or boundary clock.
  - Depending on the PTP device and message type, the MAC function updates the residence time in the correction field of the PTP frame when the client asserts `tx_etstamp_ins_ctrl_residence_time_update`. The residence time is the difference between the egress and ingress timestamps.
  - For PTP frames encapsulated using the UDP/IPv6 protocol, the MAC function performs UDP checksum correction using extended bytes in the PTP frame.
  - The MAC function re-computes and re-inserts CRC-32 into the PTP frames after each timestamp or correction field insertion.
- For 2-step clock synchronization, the MAC function returns the timestamp and the associated fingerprint for all transmit frames when the client asserts `tx_egress_timestamp_request_valid`.

Table 4–11 summarizes the timestamp and correction field insertions for various PTP messages in different PTP clocks.

**Table 4–11.** Timestamp and Correction Insertion for 1-Step Clock Synchronization

| PTP Message | Ordinary Clock | | Boundary Clock | | E2E Transparent Clock | | P2P Transparent Clock | |
|---|---|---|---|---|---|---|---|---|
| | Insert Timestamp | Insert Correction | Insert Timestamp | Insert Correction | Insert Timestamp | Insert Correction | Insert Timestamp | Insert Correction |
| Sync | Yes *(1)* | No | Yes *(1)* | No | No | Yes *(2)* | No | Yes *(2)* |
| Delay_Req | No | No | No | No | No | Yes *(2)* | No | Yes *(2)* |
| Pdelay_Req | No | No | No | No | No | Yes *(2)* | No | No |
| Pdelay_Resp | No | Yes *(1)*, *(2)* | No | Yes *(1)*, *(2)* | No | Yes *(2)* | No | Yes *(1)*, *(2)* |
| Delay_Resp | No | No | No | No | No | No | No | No |
| Follow_Up | No | No | No | No | No | No | No | No |
| Pdelay_Resp_ Follow_Up | No | No | No | No | No | No | No | No |
| Announce | No | No | No | No | No | No | No | No |
| Signaling | No | No | No | No | No | No | No | No |
| Management | No | No | No | No | No | No | No | No |

**Notes to Table 4–11:**

(1) Applicable only when 2-step flag in `flagField` of the PTP frame is 0.

(2) Applicable when you assert `tx_ingress_timestamp_valid`.

## Receive Datapath

In the receive datapath, the IEEE 1588v2 feature provides a timestamp for all receive frames. The timestamp is aligned with the `avalon_st_rx_startofpacket` signal.

## Frame Format

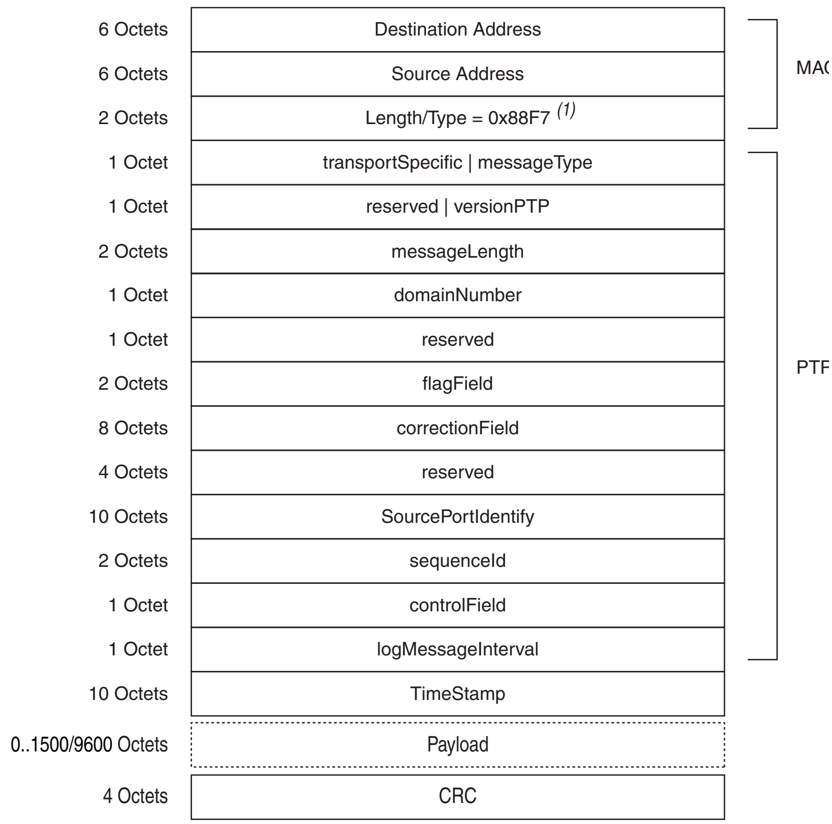
The MAC function, with the IEEE 1588v2 feature, supports PTP frame transfer for the following transport protocols:

- IEEE 802.3
- UDP/IPv4
- UDP/IPv6

## PTP Frame in IEEE 802.3

Figure 4–27 shows the format of the PTP frame encapsulated in IEEE 802.3.

**Figure 4–27.** PTP Frame in IEEE 8002.3



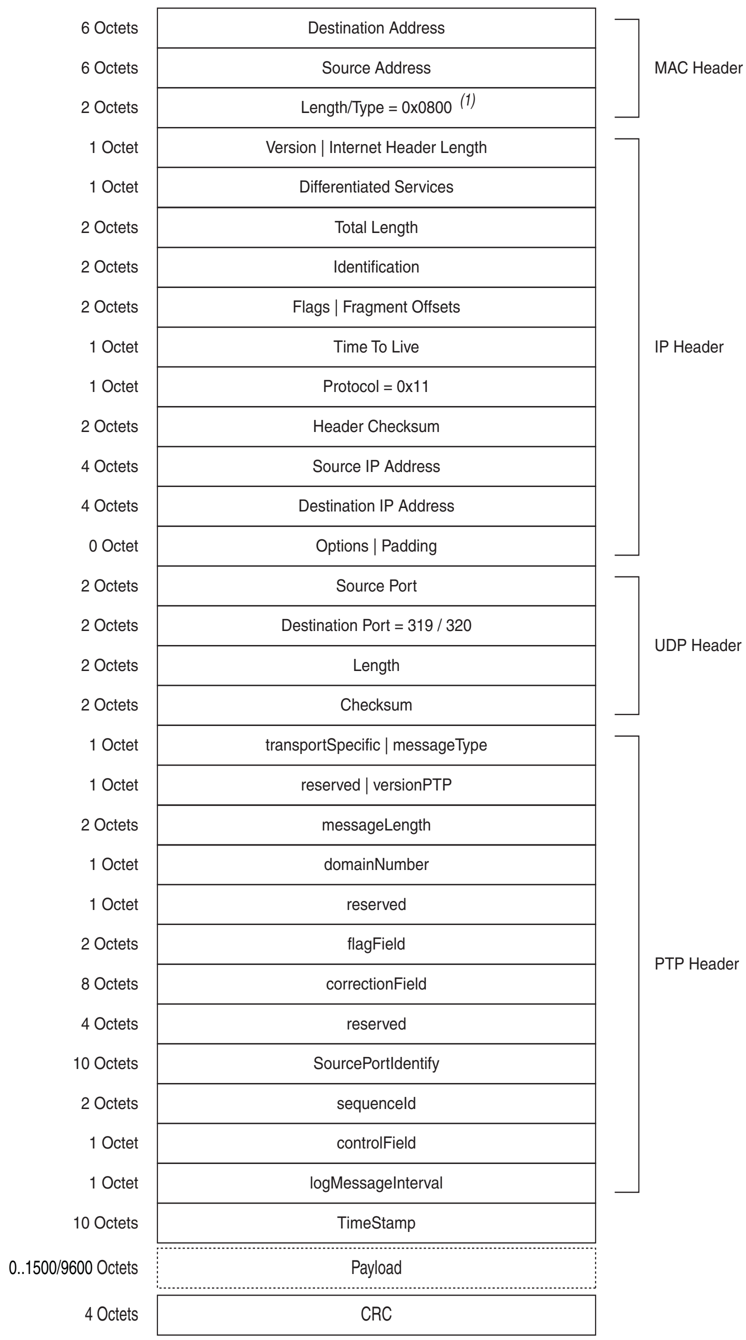

**Note to Figure 4–27:**

(1) For frames with VLAN or Stacked VLAN tag, add 4 or 8 octets offsets before the length/type field.

## PTP Frame over UDP/IPv4

Figure 4–28 shows the format of the PTP frame encapsulated in UDP/IPv4. Checksum calculation is optional for the UDP/IPv4 protocol. The 1588v2 Tx logic should set the checksum to zero.

**Figure 4–28.** PTP Frame over UDP/IPv4



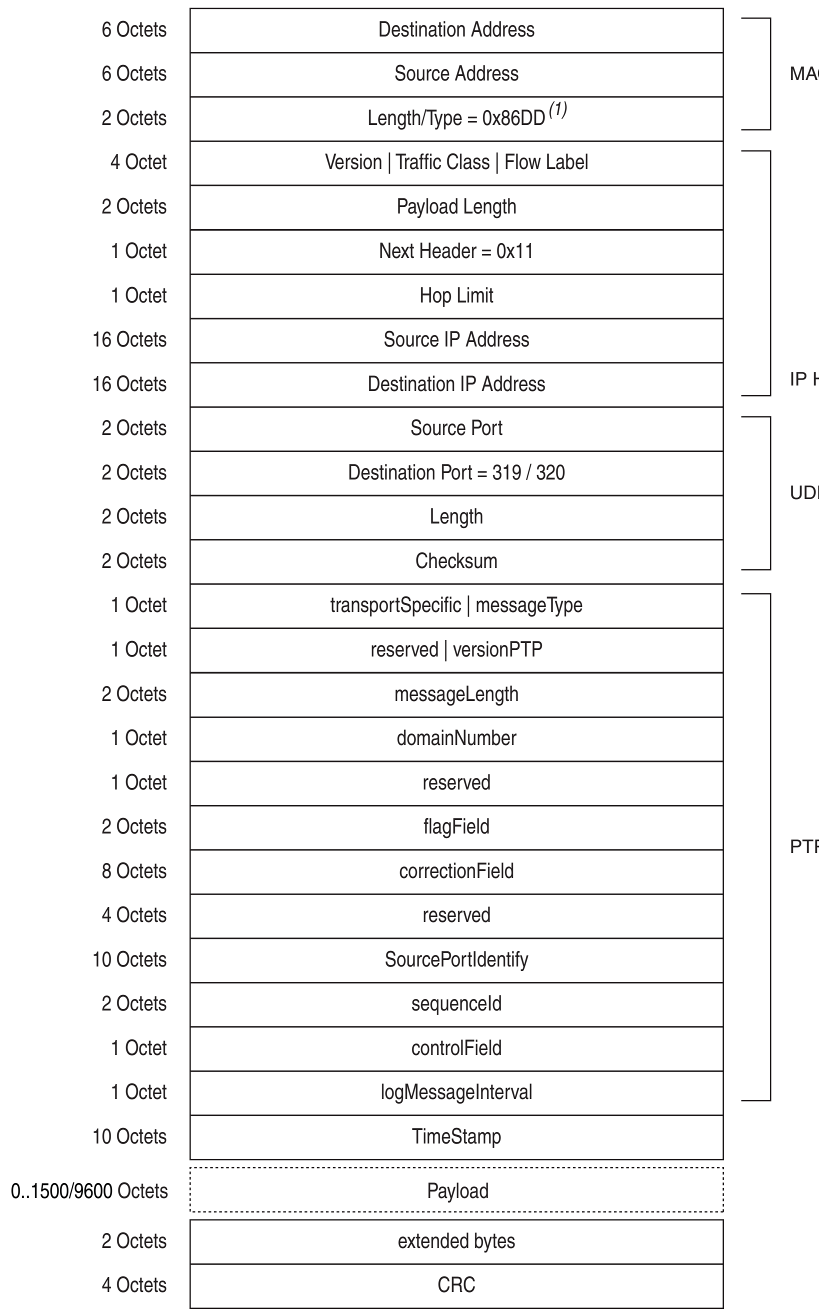

**Note to Figure 4–28:**

(1) For frames with VLAN or Stacked VLAN tag, add 4 or 8 octets offsets before the length/type field.

### PTP Frame over UDP/IPv6

Figure 4–29 shows the format of the PTP frame transported over the UDP/IPv6 protocol. Checksum calculation is mandatory for the UDP/IPv6 protocol. You must extend 2 bytes at the end of the UDP payload of the PTP frame. The MAC function modifies the extended bytes to ensure that the UDP checksum remains uncompromised.

**Figure 4–29.** PTP Frame over UDP/IPv6




**Note to Figure 4–29:**

(1) For frames with VLAN or Stacked VLAN tag, add 4 or 8 octets offsets before the length/type field.

# 5. Triple-Speed Ethernet with IEEE 1588v2 Design Example

This section describes the design example for the Triple-Speed Ethernet using the IEEE 1588v2 feature, the testbench, and its components.

## Software and Hardware Requirements

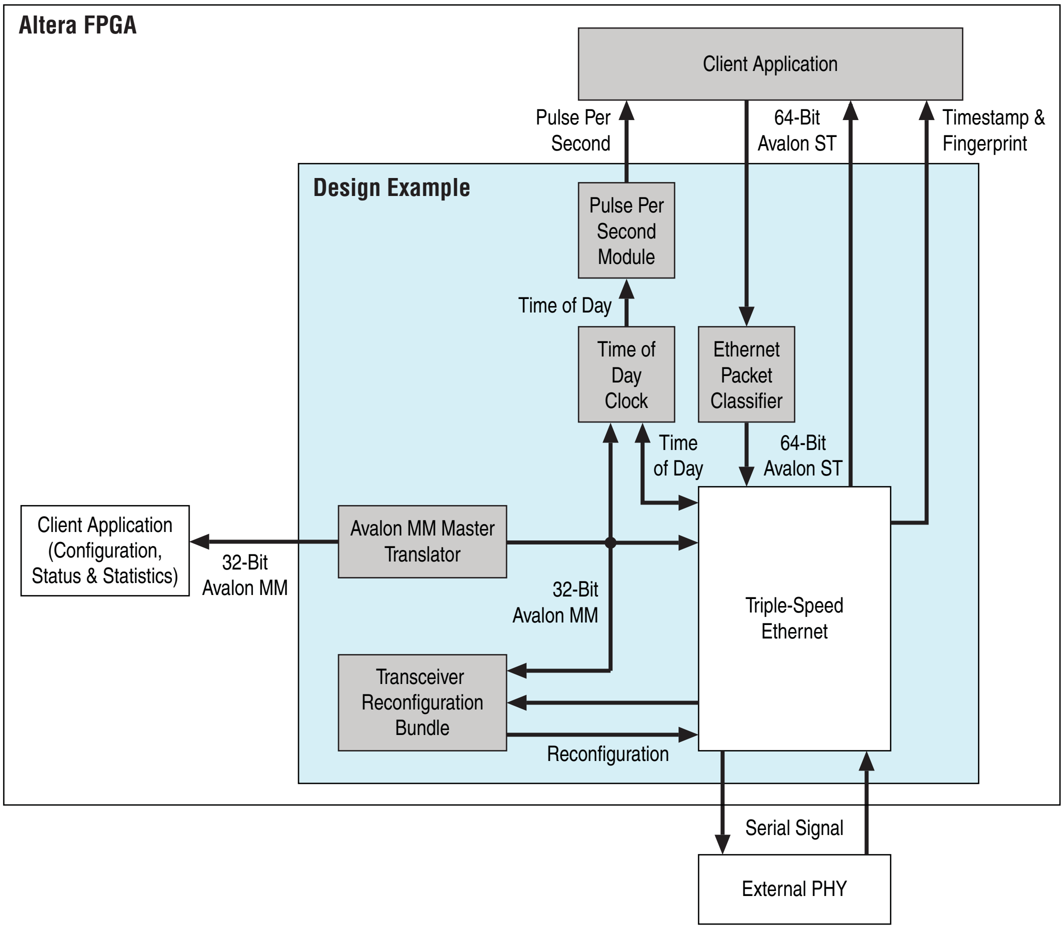
Altera uses the following hardware and software to test the Triple-Speed Ethernet with IEEE 1588v2 design example and testbench:

- Altera Complete Design Suite 13.0
- Arria V GT FPGA Development Kit
- Stratix V GX FPGA Development Kit
- ModelSim-SE 10.0b or higher

# Triple-Speed Ethernet with IEEE 1588v2 Design Example Components

Figure 5–1 shows the block diagram of a design example of the Triple-Speed Ethernet with IEEE 1588v2 feature.

**Figure 5–1. Triple-Speed Ethernet MAC with IEEE 1588v2 Design Example Block Diagram**




The Triple-Speed Ethernet with IEEE 1588v2 design example comprises the following components:

- Triple-Speed Ethernet design that has the following parameter settings:
  - 10/100/1000 Mbps Ethernet MAC with 1000BASE-X/SGMII PCS
  - SGMII bridge enabled
  - Used GXB transceiver block
  - Number of port = 1
  - Timestamping enabled
  - PTP 1-step clock enabled
  - Timestamp fingerprint width = 4
  - Internal FIFO not used

- Transceiver Reconfiguration Controller—dynamically calibrates and reconfigures the features of the PHY IP cores.
- Ethernet Packet Classifier—decodes the packet type of incoming PTP packets and returns the decoded information to the Triple-Speed Ethernet MAC.
- Ethernet ToD Clock—provides 64-bits and/or 96-bits time-of-day to TX and RX of Triple-Speed Ethernet MAC.
- Pulse Per Second Module—returns pulse per second (pps) to user.
- Avalon-MM Master Translator—provides access to the registers of the following components through the Avalon-MM interface:
  - Triple-Speed Ethernet MAC
  - Transceiver Reconfiguration Controller
  - ToD Clock

## Base Addresses

Table 5–1 lists the design example components that you can reconfigure to suit your verification objectives. To reconfigure the components, write to their registers using the base addresses listed in the table and the register offsets described in the components' user guides.

**Table 5–1. Base Addresses of Triple-Speed Ethernet MAC with IEEE 1588v2 Design Example Components**

| Component | Base Address |
|---|---|
| Triple-Speed Ethernet | 0x0000 |
| Time of Day Clock | 0x1000 |
| Transceiver Reconfiguration Controller | 0x2000 |

# Triple-Speed Ethernet MAC with IEEE 1588v2 Design Example Files

Figure 5–2 shows the directory structure for the Triple-Speed Ethernet MAC with IEEE 1588v2 design examples and testbenches.

**Figure 5–2. Triple-Speed Ethernet MAC with IEEE 1588v2 Design Example Folders**

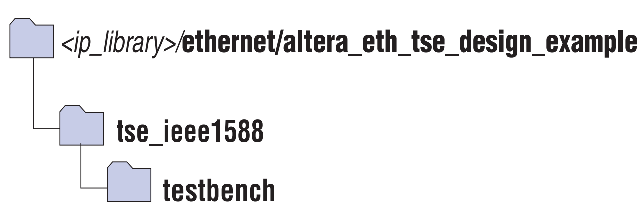


Table 5–2 lists the files in the **..\tse_ieee1588** directory.

**Table 5–2. Triple-Speed Ethernet MAC with IEEE 1588v2 Design Example Files**

| File Name | Description |
|---|---|
| **tse_1588_top.v** | The top-level entity file of the design example for verification in hardware. |
| **tse_1588_top.sdc** | The Quartus II SDC constraint file for use with the TimeQuest timing analyzer. |
| **tse_1588.qsys** | A Qsys file for the Triple-Speed Ethernet design example with IEEE 1588v2 option enabled. |
| **tb_run_simulation.tcl** | Tcl script to run testbench simulation. |

# Creating a New Triple-Speed Ethernet MAC with IEEE 1588v2 Design

You can use the Quartus II software to create a new Triple-Speed Ethernet MAC with IEEE 1588v2 design. Altera provides a Qsys design example file that you can customize to facilitate the development of your Triple-Speed Ethernet MAC with IEEE 1588v2 design.

To create the design, perform the following steps:

1. Launch the Quartus II software and open the **tse_1588.top.v** file from the project directory.
2. Launch Qsys from the Tools menu and open the **tse_1588.qsys** file. By default, the design example targets the Stratix V device family. To change the target family, click on the **Project Settings** tab and select the desired device from the **Device family** list.
3. Turn off the additional module under the **Use** column if your design does not require it. This action disconnects the module from the Triple-Speed Ethernet MAC with IEEE 1588v2 system.
4. Double-click on **triple_speed_ethernet_0** to launch the parameter editor.
5. Specify the required parameters in the parameter editor.
6. Click **Finish**.
7. On the **Generation** tab, select either a Verilog HDL or VHDL simulation model and make sure that the **Create HDL design files for synthesis** option is turned on.
8. Click **Generate** to generate the simulation and synthesis files.

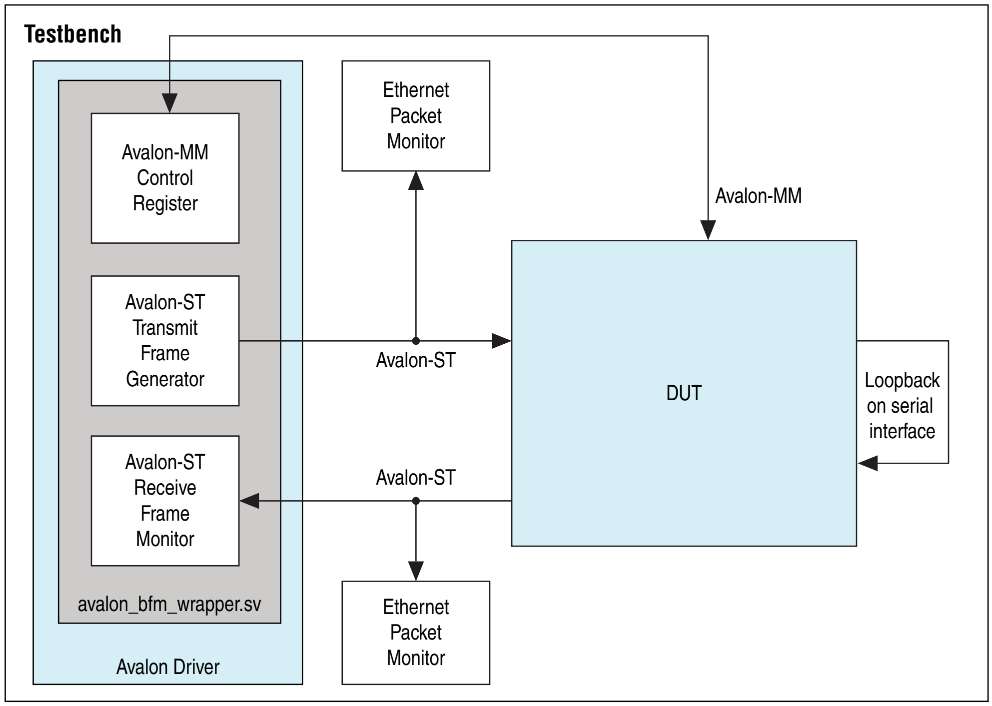
# Triple-Speed Ethernet with IEEE 1588v2 Testbench

Altera provides testbench for you to verify the Triple-Speed Ethernet with IEEE 1588v2 design example. The following sections describe the testbench, its components, and use.

# Triple-Speed Ethernet with IEEE 1588v2 Testbench

The testbench operates in loopback mode. Figure 5–3 shows the flow of the packets in the design example.

**Figure 5–3. Testbench Block Diagram**




# Triple-Speed Ethernet with IEEE 1588v2 Testbench Components

The testbenches comprise the following modules:

- Device under test (DUT)—the design example.
- Avalon driver—uses Avalon-ST master bus functional models (BFMs) to exercise the transmit and receive paths. The driver also uses the master Avalon-MM BFM to access the Avalon-MM interfaces of the design example components.
- Packet monitors—monitors the transmit and receive datapaths, and displays the frames in the simulator console.

## Triple-Speed Ethernet with IEEE 1588v2 Testbench Files

The <*ip library*>/**ethernet/altera_eth_tse_design_example/tse_ieee1588/testbench** directory contains the testbench files.

Table 5–3 describes the files that implement the Triple-Speed Ethernet with IEEE 1588v2 testbench.

**Table 5–3. Triple-Speed Ethernet with IEEE 1588v2 Testbench Files**

| File Name | Description |
|---|---|
| **avalon_bfm_wrapper.sv** | A wrapper for the Avalon BFMs that the **avalon_driver.sv** file uses. |
| **avalon_driver.sv** | A SystemVerilog HDL driver that utilizes the BFMs to exercise the transmit and receive path, and access the Avalon-MM interface. |
| **avalon_if_params_pkg.sv** | A SystemVerilog HDL testbench that contains parameters to configure the BFMs. Because the configuration is specific to the DUT, you must not change the contents of this file. |
| **avalon_st_eth_packet_monitor.sv** | A SystemVerilog HDL testbench that monitors the Avalon-ST transmit and receive interfaces. |
| **default_test_params_pkg.sv** | A SystemVerilog HDL package that contains the default parameter settings of the testbench. |
| **eth_mac_frame.sv** | A SystemVerilog HDL class that defines the Ethernet frames. The **avalon_driver.sv** file uses this class. |
| **eth_register_map_params_pkg.sv** | A SystemVerilog HDL package that maps addresses to the Avalon-MM control registers. |
| **ptp_timestamp.sv** | A SystemVerilog HDL class that defines the timestamp in the testbench. |
| **tb_testcase.sv** | A SystemVerilog HDL testbench file that controls the flow of the testbench. |
| **tb_top.sv** | The top-level testbench file. This file includes the customized Triple-Speed Ethernet MAC, which is the device under test (DUT), a client packet generator, and a client packet monitor along with other logic blocks. |
| **wave.do** | A signal tracing macro script for use with the ModelSim simulation software to display testbench signals. |

## Triple-Speed Ethernet with IEEE 1588v2 Testbench Simulation Flow

Upon a simulated power-on reset, each testbench performs the following operations:

1. Initializes the DUT by configuring the following options through the Avalon-MM interface:
   - Configures the MAC. In the MAC, sets the transmit primary MAC address to EE-CC-88-CC-AA-EE, sets the speed to 1000 Mbps, enables TX and RX MAC, enables pad removal at receive, sets IPG to 12, and sets maximum packet size to 1518.
   - Configures PCS and SGMII interface to 1000BASE-X.
   - Configures Timestamp Unit in the MAC, by setting periods and path delay adjustments of the clocks.
   - Configures ToD clock by loading a predefined time value.
   - Configures clock mode of Packet Classifier to Ordinary Clock mode.
2. Starts packet transmission with different clock mode. The testbench sends a total of three packets:
   - 1-step PTP Sync message over Ethernet
   - 1-step PTP Sync message over UDP/IPv4 with VLAN tag
   - 2-step PTP Sync message over UDP/IPv6 with stacked VLAN tag
3. Configures clock mode of Packet Classifier to End-to-end Transparent Clock mode.
4. Starts packet transmission. The testbench sends a total of three packets:
   - 1-step PTP Sync message over Ethernet
   - 1-step PTP Sync message over UDP/IPv4 with VLAN tag
   - 2-step PTP Sync message over UDP/IPv6 with stacked VLAN tag
5. Ends transmission.

## Simulating Triple-Speed Ethernet with IEEE 1588v2 Testbench with ModelSim Simulator

To use the ModelSim simulator to simulate the testbench design, follow these steps:

1. Copy the respective design example directory to your preferred project directory: **tse_ieee1588** from <*ip library*>/**ethernet/altera_eth_tse_design_example**.
2. Launch Qsys from the Tools menu and open the **tse_1588.qsys** file.
3. On the **Generation** tab, select either a Verilog HDL or VHDL simulation model.
4. Click **Generate** to generate the simulation and synthesis files.
5. Run the following command to set up the required libraries, to compile the generated IP Functional simulation model, and to exercise the simulation model with the provided testbench:

```
do tb_run.tcl↵
```

## MAC Configuration Register Space

Table 6–1 provides an overview of the MAC register space. Use the registers to configure the different aspects of the MAC function and retrieve its status and statistics counters.

In multiport MACs, a contiguous register space is allocated for all ports and accessed via the Avalon-MM control interface. For example, if the register space base address for the first port is 0x00, the base address for the next port is 0x100 and so forth. The registers that are shared among the instances occupy the register space of the first port. Updating these registers in the register space of other ports has no effect on the configuration.

**Table 6–1.** Overview of MAC Register Space (Part 1 of 2)

| Dword Offset | Section | Description |
|---|---|---|
| 0x00 – 0x17 | Base Configuration | Base registers to configure the MAC function. At the minimum, you must configure the following functions:<br>■ Primary MAC address (mac_0/mac_1)<br>■ Enable transmit and receive paths (TX_ENA and RX_ENA bits in the command_config register)<br>The following registers are shared among all instances of a multiport MAC:<br>■ rev<br>■ scratch<br>■ frm_length<br>■ pause_quant<br>■ mdio_addr0 and mdio_addr1<br>■ tx_ipg_length<br>For more information about the base configuration registers, refer to "Base Configuration Registers (Dword Offset 0x00 – 0x17)" on page 6–3. |
| 0x18 – 0x38 | Statistics Counters | Counters collecting traffic statistics. For more information about the statistics counters, refer to "Statistics Counters (Dword Offset 0x18 – 0x38)" on page 6–9. |
| 0x3A | Transmit Command | Transmit and receive datapaths control register. For more information about these registers, see "Transmit and Receive Command Registers (Dword Offset 0x3A – 0x3B)" on page 6–12. |
| 0x3B | Receive Command | |
| 0x3C – 0x3E | Extended Statistics Counters | Upper 32 bits of selected statistics counters. These registers are used if you turn on the option to use extended statistics counters. For more information about these counters, refer to "Statistics Counters (Dword Offset 0x18 – 0x38)" on page 6–9. |
| 0x3F | Reserved | Unused. |

**Table 6–1.** Overview of MAC Register Space (Part 2 of 2)

| Dword Offset | Section | Description |
|---|---|---|
| 0x40 – 0x7F | Multicast Hash Table | 64-entry write-only hash table to resolve multicast addresses. Only bit 0 in each entry is significant. When you write a 1 to a dword offset in the hash table, the MAC accepts all multicast MAC addresses that hash to the value of the address (bits 5:0). Otherwise, the MAC rejects the multicast address. This table is cleared during reset.<br>Hashing is not supported in 10/100 and 1000 Mbps Small MAC core variations. |
| 0x80 – 0x9F | MDIO Space 0<br>or PCS Function Configuration | MDIO Space 0 and MDIO Space 1 map to registers 0 to 31 of the PHY devices whose addresses are configured in the `mdio_addr0` and `mdio_addr1` registers respectively. For example, register 0 of PHY device 0 maps to dword offset 0x80, register 1 maps to dword offset 0x81 and so forth.<br>Reading or writing to MDIO Space 0 or MDIO Space 1 immediately triggers a corresponding MDIO transaction to read or write the PHY register. Only bits [15:0] of each register are significant. Write 0 to bits [31:16] and ignore them on reads.<br>If your variation does not include the PCS function, you can use MDIO Space 0 and MDIO Space 1 to map to two PHY devices.<br>If your MAC variation includes the PCS function, the PCS function is always device 0 and its configuration registers ("PCS Configuration Register Space" on page 6–15) occupy MDIO Space 0. You can use MDIO Space 1 to map to a PHY device. |
| 0xA0 – 0xBF | MDIO Space 1 | |
| 0xC0 – 0xC7 | Supplementary Address | Supplementary unicast addresses. For more information about these addresses, refer to "Supplementary Address (Dword Offset 0xC0 – 0xC7)" on page 6–13. |
| 0xC8 – 0xCF | Reserved *(1)* | Unused. |
| 0xD0 – 0xD6 | IEEE 1588v2 Feature | Registers to configure the IEEE 1588v2 feature. For more information about these registers, refer to "IEEE 1588v2 Feature (Dword Offset 0xD0 – 0xD6)" on page 6–14. |
| 0xD7 – 0xFF | Reserved *(1)* | Unused. |

**Note to Table 6–1:**

(1) Altera recommends that you set all bits in the reserved registers to 0 and ignore them on reads.

## Base Configuration Registers (Dword Offset 0x00 – 0x17)

Table 6–2 lists the base registers you can use to configure the MAC function.

A software reset does not reset these registers except the first two bits (TX_ENA and RX_ENA = 0) in the command_config register.

**Table 6–2.** Register Map (Part 1 of 4)

| Dword Offset | Name | R/W | Description | HW Reset |
|---|---|---|---|---|
| 0x00 | rev | RO | ■ Bits[15:0]—Set to the current version of the MegaCore function.<br>■ Bits[31:16]—Customer specific revision, specified by the CUST_VERSION parameter defined in the top-level file generated for the instance of the MegaCore function. These bits are set to 0 during the configuration of the MegaCore function. | *<IP version number>* |
| 0x01 | scratch *(1)* | RW | Scratch register. Provides a memory location for you to test the device memory operation. | 0 |
| 0x02 | command_config | RW | MAC configuration register. Use this register to control and configure the MAC function. The MAC function starts operation as soon as the transmit and receive enable bits in this register are turned on. Altera, therefore, recommends that you configure this register last.<br>See "Command_Config Register (Dword Offset 0x02)" on page 6–6 for the bit description. | 0 |
| 0x03 | mac_0 | RW | 6-byte MAC primary address. The first four most significant bytes of the MAC address occupy mac_0 in reverse order. The last two bytes of the MAC address occupy the two least significant bytes of mac_1 in reverse order.<br>For example, if the MAC address is 00-1C-23-17-4A-CB, the following assignments are made:<br>mac_0 = 0x17231c00<br>mac_1 = 0x0000CB4a<br>Ensure that you configure these registers with a valid MAC address if you disable the promiscuous mode (PROMIS_EN bit in command_config = 0). | 0 |
| 0x04 | mac_1 | RW | | 0 |
| 0x05 | frm_length | RW/ RO | ■ Bits[15:0]—16-bit maximum frame length in bytes. The MegaCore function checks the length of receive frames against this value. Typical value is 1518.<br>In 10/100 and 1000 Small MAC core variations, this register is RO and the maximum frame length is fixed to 1518.<br>■ Bits[31:16]—unused. | 1518 |

**Table 6–2.** Register Map (Part 2 of 4)

| Dword Offset | Name | R/W | Description | HW Reset |
|---|---|---|---|---|
| 0x06 | `pause_quant` | RW | ■ Bits[15:0]—16-bit pause quanta. Use this register to specify the pause quanta to be sent to remote devices when the local device is congested. The MegaCore function sets the pause quanta (P1, P2) field in pause frames to the value of this register.<br>10/100 and 1000 Small MAC core variations do not support flow control.<br>■ Bits[31:16]—unused. | 0 |
| 0x07 | `rx_section_empty` | RW/<br>RO | Variable-length section-empty threshold of the receive FIFO buffer. Use the depth of your FIFO buffer to determine this threshold. This threshold is typically set to (FIFO Depth – 16).<br>Set this threshold to a value that is below the `rx_almost_full` threshold and above the `rx_section_full` or `rx_almost_empty` threshold.<br>In 10/100 and 1000 Small MAC core variations, this register is RO and the register is set to a fixed value of (FIFO Depth – 16). | 0 |
| 0x08 | `rx_section_full` | RW/<br>RO | Variable-length section-full threshold of the receive FIFO buffer. Use the depth of your FIFO buffer to determine this threshold.<br>For cut-through mode, this threshold is typically set to 16. Set this threshold to a value that is above the `rx_almost_empty` threshold.<br>For store-and-forward mode, set this threshold to 0.<br>In 10/100 and 1000 Small MAC core variations, this register is RO and the register is set to a fixed value of 16. | 0 |
| 0x09 | `tx_section_empty` | RW/<br>RO | Variable-length section-empty threshold of the transmit FIFO buffer. Use the depth of your FIFO buffer to determine this threshold. This threshold is typically set to (FIFO Depth – 16).<br>Set this threshold to a value below the `rx_almost_full` threshold and above the `rx_section_full` or `rx_almost_empty` threshold.<br>In 10/100 and 1000 Small MAC core variations, this register is RO and the register is set to a fixed value of (FIFO Depth – 16). | 0 |
| 0x0A | `tx_section_full` | RW/<br>RO | Variable-length section-full threshold of the transmit FIFO buffer. Use the depth of your FIFO buffer to determine this threshold.<br>For cut-through mode, this threshold is typically set to 16. Set this threshold to a value above the `tx_almost_empty` threshold.<br>For store-and-forward mode, set this threshold to 0.<br>In 10/100 and 1000 Small MAC core variations, this register is RO and the register is set to a fixed value of 16. | 0 |

**Table 6–2.** Register Map (Part 3 of 4)

| Dword Offset | Name | R/W | Description | HW Reset |
|---|---|---|---|---|
| 0x0B | `rx_almost_empty` | RW/ RO | Variable-length almost-empty threshold of the receive FIFO buffer. Use the depth of your FIFO buffer to determine this threshold.<br>Due to internal pipeline latency, you must set this threshold to a value greater than 3. This threshold is typically set to 8.<br>In 10/100 and 1000 Small MAC core variations, this register is RO and the register is set to a fixed value of 8. | 0 |
| 0x0C | `rx_almost_full` | RW/ RO | Variable-length almost-full threshold of the receive FIFO buffer. Use the depth of your FIFO buffer to determine this threshold.<br>Due to internal pipeline latency, you must set this threshold to a value greater than 3. This threshold is typically set to 8.<br>In 10/100 and 1000 Small MAC core variations, this register is RO and the register is set to a fixed value of 8. | 0 |
| 0x0D | `tx_almost_empty` | RW/ RO | Variable-length almost-empty threshold of the transmit FIFO buffer. Use the depth of your FIFO buffer to determine this threshold.<br>Due to internal pipeline latency, you must set this threshold to a value greater than 3. This threshold is typically set to 8.<br>In 10/100 and 1000 Small MAC core variations, this register is RO and the register is set to a fixed value of 8. | 0 |
| 0x0E | `tx_almost_full` | RW/ RO | Variable-length almost-full threshold of the transmit FIFO buffer. Use the depth of your FIFO buffer to determine this threshold.<br>You must set this register to a value greater than or equal to 3. A value of 3 indicates 0 ready latency; a value of 4 indicates 1 ready latency, and so forth. Because the maximum ready latency on the Avalon-ST interface is 8, you can only set this register to a maximum value of 11. This threshold is typically set to 3.<br>In 10/100 and 1000 Small MAC core variations, this register is RO and the register is set to a fixed value of 3. | 0 |
| 0x0F | `mdio_addr0` | RW | ■ Bits[4:0]—5-bit PHY address. Set these registers to the addresses of any connected PHY devices you want to access. The `mdio_addr0` and `mdio_addr1` registers contain the addresses of the PHY whose registers are mapped to MDIO Space 0 and MDIO Space 1 respectively.<br>■ Bits[31:5]—unused. Set to read-only value of 0. | 0 |
| 0x10 | `mdio_addr1` | RW | | 1 |
| 0x11 | `holdoff_quant` | RW | ■ Bit[15:0]—16-bit holdoff quanta. When you enable the flow control, use this register to specify the gap between consecutive XOFF requests.<br>■ Bits[31:16]—unused. | 0xFFFF |
| 0x12 – 0x16 | Reserved | — | — | 0 |

**Table 6–2.** Register Map (Part 4 of 4)

| Dword Offset | Name | R/W | Description | HW Reset |
|---|---|---|---|---|
| 0x17 | `tx_ipg_length` | RW | ■ Bits[4:0]—minimum IPG. Valid values are between 8 and 26 byte-times. If this register is set to an invalid value, it defaults to 12 byte-times, which is a typical value of the minimum IPG.<br>In 10/100 and 1000 Small MAC core variations, this register is RO and the register is set to a fixed value of 12.<br>■ Bits[31:5]—unused. Set to read-only value 0. | 0 |

**Note to Table 6–2:**

(1) Register is not available in 10/100 and 1000 Small MAC variations.

## Command_Config Register (Dword Offset 0x02)

Figure 6–1 shows the fields in the `command_config` register.

**Figure 6–1.** Command_Config Register

| 31 | 30 ... 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 ... 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNT_RESET | RESERVED | RX_ERR_DISC | ENA_10 | NO_LGTH_CHECK | CTRL_FRM_ENA | XOFF_GEN | WAKEUP | SLEEP | MAGIC_ENA | TX_ADDR_SEL | LOOP_ENA | MHASH_SEL | SW_RESET | LATE_COL | EXCESS_COL | HD_ENA | TX_ADDR_INS | PAUSE_IGNORE | PAUSE_FWD | CRC_FWD | PAD_EN | PROMIS_EN | ETH_SPEED | XON_GEN | RX_ENA | TX_ENA |

Table 6–3 describes each field in the `command_config` register.

☞ At the minimum, you must configure the `TX_ENA` and `RX_ENA` bits to 1 to start the MAC operations. When configuring the `command_config` register, Altera recommends that you configure the `TX_ENA` and `RX_ENA` bits the last because the MAC function immediately starts its operations once these bits are set to 1.

**Table 6–3.** Command_Config Register Field Descriptions (Part 1 of 4)

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 0 | `TX_ENA` | RW | Transmit enable. Set this bit to 1 to enable the transmit datapath. Self-clearing reset bit. |
| 1 | `RX_ENA` | RW | Receive enable. Set this bit to 1 to enable the receive datapath. Self-clearing reset bit. |
| 2 | `XON_GEN` | RW | Pause frame generation. When you set this bit to 1, the MAC function generates a pause frame with a pause quanta of 0, independent of the status of the receive FIFO buffer. |

**Table 6–3.** Command_Config Register Field Descriptions (Part 2 of 4)

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 3 | ETH_SPEED | RW | Ethernet speed control.<br>■ Set this bit to 1 to enable gigabit Ethernet operation. The set_1000 signal is masked and does not affect the operation.<br>■ If you set this bit to 0, gigabit Ethernet operation is enabled only if the set_1000 signal is asserted. Otherwise, the MAC function operates in 10/100 Mbps Ethernet mode.<br>When the MAC operates in gigabit mode, the eth_mode signal is asserted. This bit is not available in the small MAC variation. |
| 4 | PROMIS_EN | RW | Promiscuous enable. Set this bit to 1 to enable promiscuous mode. In this mode, the MAC function receives all frames without address filtering. |
| 5 | PAD_EN | RW | Padding removal on receive. Set this bit to 1 to remove padding from receive frames before the MAC function forwards the frames to the user application. This bit has no effect on transmit frames.<br>This bit is not available in the small MAC variation. |
| 6 | CRC_FWD | RW | CRC forwarding on receive.<br>■ Set this bit to 1 to forward the CRC field to the user application.<br>■ Set this bit to 0 to remove the CRC field from receive frames before the MAC function forwards the frame to the user application.<br>■ The MAC function ignores this bit when it receives a padded frame and the PAD_EN bit is 1. In this case, the MAC function checks the CRC field and removes the checksum and padding from the frame before forwarding the frame to the user application. |
| 7 | PAUSE_FWD | RW | Pause frame forwarding on receive.<br>■ Set this bit to 1 to forward receive pause frames to the user application.<br>■ Set this bit to 0 to terminate and discard receive pause frames. |
| 8 | PAUSE_IGNORE | RW | Pause frame processing on receive.<br>■ Set this bit to 1 to ignore receive pause frames.<br>■ Set this bit to 0 to process receive pause frames. The MAC function suspends transmission for an amount of time specified by the pause quanta. |
| 9 | TX_ADDR_INS | RW | MAC address on transmit.<br>■ Set this bit to 1 to overwrite the source MAC address in transmit frames received from the user application with the MAC primary or supplementary address configured in the registers. The TX_ADDR_SEL bit determines the address selection.<br>■ Set this bit to 0 to retain the source MAC address in transmit frames received from the user application. |
| 10 | HD_ENA | RW | Half-duplex enable.<br>■ Set this bit to 1 to enable half-duplex.<br>■ Set this bit to 0 to enable full-duplex.<br>■ The MAC function ignores this bit if you set the ETH_SPEED bit to 1. |

**Table 6–3.** Command_Config Register Field Descriptions (Part 3 of 4)

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 11 | EXCESS_COL | RO | Excessive collision condition.<br>■ The MAC function sets this bit to 1 when it discards a frame after detecting a collision on 16 consecutive frame retransmissions.<br>■ The MAC function clears this bit following a hardware or software reset. See the SW_RESET bit description. |
| 12 | LATE_COL | RO | Late collision condition.<br>■ The MAC function sets this bit to 1 when it detects a collision after transmitting 64 bytes and discards the frame.<br>■ The MAC function clears this bit following a hardware or software reset. See the SW_RESET bit description. |
| 13 | SW_RESET | RW | Software reset. Set this bit to 1 to trigger a software reset. The MAC function clears this bit when it completes the software reset sequence.<br>When reset is triggered, the MAC function completes the current transmission or reception, and subsequently disables the transmit and receive logic, flushes the receive FIFO buffer, and resets the statistics counters. |
| 14 | MHASH_SEL | RW | Hash-code mode selection for multicast address resolution.<br>■ Set this bit to 0 to generate the hash code from the full 48-bit destination address.<br>■ Set this bit to 1 to generate the hash code from the lower 24 bits of the destination MAC address. |
| 15 | LOOP_ENA | RW | Local loopback enable. Set this bit to 1 to enable local loopback on the RGMII/GMII/MII of the MAC. The MAC function sends transmit frames back to the receive path.<br>This bit is not available in the small MAC variation. |
| 18 – 16 | TX_ADDR_SEL[2:0] | RW | Source MAC address selection on transmit. If you set the TX_ADDR_INS bit to 1, the value of these bits determines the MAC address the MAC function selects to overwrite the source MAC address in frames received from the user application.<br>■ 000 = primary address configured in the mac_0 and mac_1 registers.<br>■ 100 = supplementary address configured in the smac_0_0 and smac_0_1 registers.<br>■ 101 = supplementary address configured in the smac_1_0 and smac_1_1 registers.<br>■ 110 = supplementary address configured in the smac_2_0 and smac_2_1 registers.<br>■ 111 = supplementary address configured in the smac_3_0 and smac_3_1 registers. |
| 19 | MAGIC_ENA | RW | Magic packet detection. Set this bit to 1 to enable magic packet detection.<br>This bit is not available in the small MAC variation. |
| 20 | SLEEP | RW | Sleep mode enable. When the MAGIC_ENA bit is 1, set this bit to 1 to put the MAC function to sleep and enable magic packet detection.<br>This bit is not available in the small MAC variation. |

**Table 6–3.** Command_Config Register Field Descriptions (Part 4 of 4)

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 21 | WAKEUP | RO | Node wake-up request. Valid only when the MAGIC_ENA bit is 1.<br>■ The MAC function sets this bit to 1 when a magic packet is detected.<br>■ The MAC function clears this bit when the SLEEP bit is set to 0. |
| 22 | XOFF_GEN | RW | Pause frame generation. Set this bit to 1 to generate a pause frame independent of the status of the receive FIFO buffer. The MAC function sets the pause quanta field in the pause frame to the value configured in the pause_quant register. |
| 23 | CNTL_FRM_ENA | RW | MAC control frame enable on receive.<br>■ Set this bit to 1 to accept control frames other than pause frames (opcode = 0x0001) and forward them to the user application.<br>■ Set this bit to 0 to discard control frames other than pause frames. |
| 24 | NO_LGTH_CHECK | RW | Payload length check on receive.<br>■ Set this bit to 0 to check the actual payload length of receive frames against the length/type field in receive frames.<br>■ Set this bit to 1 to omit length checking.<br>This bit is not available in the small MAC variation |
| 25 | ENA_10 | RW | 10-Mbps interface enable. Set this bit to 1 to enable the 10-Mbps interface. The MAC function asserts the ena_10 signal when you enable the 10-Mbps interface. You can also enable the 10-Mbps interface by asserting the set_10 signal. |
| 26 | RX_ERR_DISC | RW | Erroneous frames processing on receive.<br>■ Set this bit to 1 to discard erroneous frames received. This applies only when you enable store and forward operation in the receive FIFO buffer by setting the rx_section_full register to 0.<br>■ Set this bit to 0 to forward erroneous frames to the user application with rx_err[0] asserted. |
| 27 | DISABLE_READ_TIMEOUT | RW | Set this bit to 1 to disable MAC configuration register read timeout. |
| 28 – 30 | Reserved | — | — |
| 31 | CNT_RESET | RW | Statistics counters reset. Set this bit to 1 to clear the statistics counters. The MAC function clears this bit when the reset sequence completes. |

## Statistics Counters (Dword Offset 0x18 – 0x38)

Table 6–4 describes the read-only registers that collect the statistics on the transmit and receive datapaths. A hardware reset clears these registers; a software reset also clears these registers except aMacID.

The register description uses the following definitions:

- Good frame—error-free frames with valid frame length.
- Error frame—frames that contain errors or whose length is invalid.
- Invalid frame—frames that are not addressed to the MAC function. The MAC function drops this frame.

**Table 6–4.** Statistics Counters (Part 1 of 2)

| Dword Offset | Name | R/W | Description |
|---|---|---|---|
| 0x18 – 0x19 | aMacID | RO | The MAC address. This register is wired to the primary MAC address in the mac_0 and mac_1 registers. |
| 0x1A | aFramesTransmittedOK | RO | The number of frames that are successfully transmitted including the pause frames. |
| 0x1B | aFramesReceivedOK | RO | The number of frames that are successfully received including the pause frames. |
| 0x1C | aFrameCheck SequenceErrors | RO | The number of receive frames with CRC error. |
| 0x1D | aAlignmentErrors | RO | The number of receive frames with alignment error. |
| 0x1E | aOctetsTransmittedOK | RO | The number of data and padding octets that are successfully transmitted.<br>This register contains the lower 32 bits of the aOctetsTransmittedOK counter. The upper 32 bits of this statistics counter reside at the dword offset 0x0F. |
| 0x1F | aOctetsReceivedOK | RO | The number of data and padding octets that are successfully received.<br>The lower 32 bits of the aOctetsReceivedOK counter. The upper 32 bits of this statistics counter reside at the dword offset 0x3D. |
| 0x20 | aTxPAUSEMACCtrlFrames | RO | The number of pause frames transmitted. |
| 0x21 | aRxPAUSEMACCtrlFrames | RO | The number received pause frames received. |
| 0x22 | ifInErrors | RO | The number of errored frames received. |
| 0x23 | ifOutErrors | RO | The number of transmit frames with one the following errors:<br>■ FIFO overflow error.<br>■ FIFO underflow error.<br>■ Errors defined by the user application |
| 0x24 | ifInUcastPkts | RO | The number of valid unicast frames received. |
| 0x25 | ifInMulticastPkts | RO | The number of valid multicast frames received. The count does not include pause frames. |
| 0x26 | ifInBroadcastPkts | RO | The number of valid broadcast frames received. |
| 0x27 | ifOutDiscards | — | This statistics counter is not in use.<br>The MAC function does not discard frames that are written to the FIFO buffer by the user application. |
| 0x28 | ifOutUcastPkts | RO | The number of valid unicast frames transmitted. |
| 0x29 | ifOutMulticastPkts | RO | The number of valid multicast frames transmitted, excluding pause frames. |
| 0x2A | ifOutBroadcastPkts | RO | The number of valid broadcast frames transmitted. |
| 0x2B | etherStatsDropEvents | RO | The number of frames that are dropped due to MAC internal errors when FIFO buffer overflow persists. |
| 0x2C | etherStatsOctets | RO | The total number of octets received. This count includes both good and errored frames.<br>This register is the lower 32 bits of etherStatsOctets. The upper 32 bits of this statistics counter reside at the dword offset 0x3E. |

**Table 6–4.** Statistics Counters (Part 2 of 2)

| Dword Offset | Name | R/W | Description |
|---|---|---|---|
| 0x2D | etherStatsPkts | RO | The total number of good and errored frames received. |
| 0x2E | etherStatsUndersizePkts | RO | The number of frames received with length less than 64 bytes. This count does not include errored frames. |
| 0x2F | etherStatsOversizePkts | RO | The number of frames received that are longer than the value configured in the frm_length register. This count does not include errored frames. |
| 0x30 | etherStatsPkts64Octets | RO | The number of 64-byte frames received. This count includes good and errored frames. |
| 0x31 | etherStatsPkts65to127Octets | RO | The number of received good and errored frames between the length of 65 and 127 bytes. |
| 0x32 | etherStatsPkts128to255Octets | RO | The number of received good and errored frames between the length of 128 and 255 bytes. |
| 0x33 | etherStatsPkts256to511Octets | RO | The number of received good and errored frames between the length of 256 and 511 bytes. |
| 0x34 | etherStatsPkts512to1023Octets | RO | The number of received good and errored frames between the length of 512 and 1023 bytes. |
| 0x35 | etherStatsPkts1024to1518Octets | RO | The number of received good and errored frames between the length of 1024 and 1518 bytes. |
| 0x36 | etherStatsPkts1519toXOctets | RO | The number of received good and errored frames between the length of 1519 and the maximum frame length configured in the frm_length register. |
| 0x37 | etherStatsJabbers | RO | Too long frames with CRC error. |
| 0x38 | etherStatsFragments | RO | Too short frames with CRC error. |
| 0x39 | Reserved | — | Unused |
| **Extended Statistics Counters (0x3C – 0x3E)** | | | |
| 0x3C | msb_aOctetsTransmittedOK | RO | Upper 32 bits of the respective statistics counters. By default all statistics counters are 32 bits wide. These statistics counters can be extended to 64 bits by turning on the **Enable 64-bit byte counters** parameter. |
| 0x3D | msb_aOctetsReceivedOK | RO | |
| 0x3E | msb_etherStatsOctets | RO | |

## Transmit and Receive Command Registers (Dword Offset 0x3A – 0x3B)

Table 6–5 describes the registers that determine how the MAC function processes transmit and receive frames. A software reset does not change the values in these registers.

**Table 6–5.** Transmit and Receive Command Registers

| Dword Offset | Name | R/W | Description |
|---|---|---|---|
| 0x3A | tx_cmd_stat | RW | Specifies how the MAC function processes transmit frames. When you turn on the **Align packet headers to 32-bit boundaries** option, this register resets to 0x00040000 upon a hardware reset. Otherwise, it resets to 0x00.<br>■ Bits 0 to 16—unused.<br>■ Bit 17 (OMIT_CRC)—Set this bit to 1 to omit CRC calculation and insertion on the transmit path. The user application is therefore responsible for providing the correct data and CRC. This bit, when set to 1, always takes precedence over the ff_tx_crc_fwd signal.<br>■ Bit 18 (TX_SHIFT16)—Set this bit to 1 if the frames from the user application are aligned on 32-bit boundary. For more information, refer to "IP Payload Re-alignment" on page 4–4.<br>This setting applies only when you turn on the **Align packet headers to 32-bit boundary** option and in MAC variations with 32-bit internal FIFO buffers. Otherwise, reading this bit always return a 0.<br>In MAC variations without internal FIFO buffers, this bit is a read-only bit and takes the value of the **Align packet headers to 32-bit boundary** option.<br>■ Bits 19 to 31—unused. |
| 0x3B | rx_cmd_stat | RW | Specifies how the MAC function processes receive frames. When you turn on the **Align packet headers to 32-bit boundaries** option, this register resets to 0x02000000 upon a hardware reset. Otherwise, it resets to 0x00.<br>■ Bits 0 to 24—unused.<br>■ Bit 25 (RX_SHIFT16)—Set this bit to 1 to instruct the MAC function to align receive frames on 32-bit boundary. For more information on frame alignment, refer to "IP Payload Alignment" on page 4–11.<br>This setting applies only when you turn on the **Align packet headers to 32-bit boundary** option and in MAC variations with 32-bit internal FIFO buffers. Otherwise, reading this bit always return a 0.<br>In MAC variations without internal FIFO buffers, this bit is a read-only bit and takes the value of the **Align packet headers to 32-bit boundary** option.<br>■ Bits 26 to 31—unused. |

## Supplementary Address (Dword Offset 0xC0 – 0xC7)

Table 6–6 describes the MAC supplementary address registers. A software reset has no impact on these registers. MAC supplementary addresses are not available in 10/100 and 1000 Small MAC variations.

**Table 6–6.** Supplementary Address Registers

| Dword Offset | Name | R/W | Description | HW Reset |
|---|---|---|---|---|
| 0xC0 | smac_0_0 | RW | You can specify up to four 6-byte supplementary addresses:<br>■ smac_0_0/1<br>■ smac_1_0/1<br>■ smac_2_0/1<br>■ smac_3_0/1<br>Map the supplementary addresses to the respective registers in the same manner as the primary MAC address. Refer to the description of mac_0 and mac_1.<br>The MAC function uses the supplementary addresses for the following operations:<br>■ to filter unicast frames when the promiscuous mode is disabled (refer to "Command_Config Register (Dword Offset 0x02)" on page 6–6 for the description of the PROMIS_EN bit).<br>■ to replace the source address in transmit frames received from the user application when address insertion is enabled (refer to "Command_Config Register (Dword Offset 0x02)" on page 6–6 for the description of the TX_ADDR_INS and TX_ADDR_SEL bits).<br>If you do not require the use of supplementary addresses, configure them to the primary address. | 0 |
| 0xC1 | smac_0_1 | | | |
| 0xC2 | smac_1_0 | | | |
| 0xC3 | smac_1_1 | | | |
| 0xC4 | smac_2_0 | | | |
| 0xC5 | smac_2_1 | | | |
| 0xC6 | smac_3_0 | | | |
| 0xC7 | smac_3_1 | | | |

## IEEE 1588v2 Feature (Dword Offset 0xD0 – 0xD6)

Table 6–7 describes the MAC register space for the IEEE 1588v2 feature.

**Table 6–7.** IEEE 1588v2 MAC Registers

| Dword Offset | Name | R/W | Description | HW Reset |
|---|---|---|---|---|
| 0xD0 | tx_period | RW | Clock period for timestamp adjustment on the transmit datapath. The period register is multiplied by the number of stages separating actual timestamp and the GMII bus.<br>■ Bits 0 to 15: Period in fractional nanoseconds (TX_PERIOD_FNS).<br>■ Bits 16 to 19: Period in nanoseconds (TX_PERIOD_NS).<br>■ Bits 20 to 31: Not used.<br>The default value for the period is 0. For 125-MHz clock, set this register to 8 ns. | 0x0 |
| 0xD1 | tx_adjust_fns | RW | Static timing adjustment in fractional nanoseconds for outbound timestamps on the transmit datapath.<br>■ Bits 0 to 15: Timing adjustment in fractional nanoseconds.<br>■ Bits 16 to 31: Not used. | 0x0 |
| 0xD2 | tx_adjust_ns | RW | Static timing adjustment in nanoseconds for outbound timestamps on the transmit datapath.<br>■ Bits 0 to 15: Timing adjustment in nanoseconds.<br>■ Bits 16 to 23: Not used. | 0x0 |
| 0xD3 | rx_period | RW | Clock period for timestamp adjustment on the receive datapath. The period register is multiplied by the number of stages separating actual timestamp and the GMII bus.<br>■ Bits 0 to 15: Period in fractional nanoseconds (RX_PERIOD_FNS).<br>■ Bits 16 to 19: Period in nanoseconds (RX_PERIOD_NS).<br>■ Bits 20 to 31: Not used.<br>The default value for the period is 0. For 125-MHz clock, set this register to 8 ns. | 0x0 |
| 0xD4 | rx_adjust_fns | RW | Static timing adjustment in fractional nanoseconds for outbound timestamps on the receive datapath.<br>■ Bits 0 to 15: Timing adjustment in fractional nanoseconds.<br>■ Bits 16 to 31: Not used. | 0x0 |
| 0xD5 | rx_adjust_ns | RW | Static timing adjustment in nanoseconds for outbound timestamps on the receive datapath.<br>■ Bits 0 to 15: Timing adjustment in nanoseconds.<br>■ Bits 16 to 23: Not used. | 0x0 |

## IEEE 1588v2 Feature PMA Delay

Table 6–8 describes the PMA digital and analog delay for the IEEE 1588v2 feature and the register timing adjustment.

**Table 6–8.** IEEE 1588v2 Feature PMA Delay

| Delay | Device | Timing Adjustment (1) | |
|---|---|---|---|
| | | TX register | RX register |
| Digital | Stratix V or Arria V GZ | 53 UI | 26 UI |
| | Arria V GX, Arria V GT, or Arria V SoC | 42 UI | 44 UI |
| Analog | Stratix V | -1.1 ns | 1.75 ns |
| | Arria V | -1.1 ns | 1.75 ns |

**Note to Table 6–8:**

(1) 1 UI is equivalent to 800 ps.

# PCS Configuration Register Space

This section describes the PCS registers. Use the registers to configure the PCS function or retrieve its status.

In MAC and PCS variations, the PCS registers occupy the MAC register space and you access these registers via the MAC 32-bit Avalon-MM control interface. PCS registers are 16 bits wide, they therefore occupy only the lower 16 bits and the upper 16 bits are set to 0. The offset of the first PCS register in this variation is mapped to dword offset 0x80.

If you instantiate the IP core using the MegaWizard Plug-in Manager flow, use word addressing to access the register spaces. When you instantiate MAC and PCS variations, map the PCS registers to the respective dword offsets in the MAC register space by adding the PCS word offset to the offset of the first PCS. For example,

- In PCS only variation, you can access the `if_mode` register at word offset 0x14.
- In MAC and PCS variations, map the `if_mode` register to the MAC register space:
  - Offset of the first PCS register = 0x80
  - `if_mode` word offset = 0x14
  - `if_mode` dword offset = 0x80 + 0x14 = 0x94

If you instantiate the MAC and PCS variation using the SOPC Builder and Qsys systems, access the register spaces using byte addressing. Convert the dword offsets to byte offsets by multiplying the dword offsets by 4. For example,

- For MAC registers:
  - `comand_config` dword offset = 0x02
  - `comand_config` byte offset = 0x02 × 4 = 0x08

- For PCS registers, map the registers to the dword offsets in the MAC register space before you convert the dword offsets to byte offsets:
  - `if_mode` word offset = 0x14
  - `if_mode` dword offset = 0x80 + 0x14 = 0x94
  - `if_mode` byte offset = 0x94 × 4 = 0x250

Table 6–9 describes the PCS configuration registers.

**Table 6–9.** PCS Configuration Registers (Part 1 of 2)

| Word Offset | Register Name | R/W | Description |
|---|---|---|---|
| 0x00 | `control` | RW | PCS control register. Use this register to control and configure the PCS function. For the bit description, see Table 6–10 on page 6–18. |
| 0x01 | `status` | RO | Status register. Provides information on the operation of the PCS function. |
| 0x02 | `phy_identifier` | RO | 32-bit PHY identification register. This register is set to the value of the **PHY ID** parameter. Bits 31:16 are written to word offset 0x02. Bits 15:0 are written to word offset 0x03. |
| 0x03 | | | |
| 0x04 | `dev_ability` | RW | Use this register to advertise the device abilities to a link partner during auto-negotiation. In SGMII MAC mode, the PHY does not use this register during auto-negotiation. For the register bits description in 1000BASE-X and SGMII mode, see Table 6–12 on page 6–20 and Table 6–14 on page 6–22. |
| 0x05 | `partner_ability` | RO | Contains the device abilities advertised by the link partner during auto-negotiation. For the register bits description in 1000BASE-X and SGMII mode, refer to Table 6–12 on page 6–20 and Table 6–14 on page 6–22, respectively. |
| 0x06 | `an_expansion` | RO | Auto-negotiation expansion register. Contains the PCS function capability and auto-negotiation status. |
| 0x07 | `device_next_page` | RO | The PCS function does not support these features. These registers are always set to 0x0000 and any write access to the registers is ignored. |
| 0x08 | `partner_next_page` | | |
| 0x09 | `master_slave_cntl` | | |
| 0x0A | `master_slave_stat` | | |
| 0x0B – 0x0E | Reserved | — | — |
| 0x0F | `extended_status` | RO | The PCS function does not implement extended status registers. |
| **Specific Extended Registers** | | | |
| 0x10 | `scratch` | RW | Scratch register. Provides a memory location to test register read and write operations. |
| 0x11 | `rev` | RO | The PCS function revision. Always set to the current version of the MegaCore function. |

**Table 6–9.** PCS Configuration Registers (Part 2 of 2)

| Word Offset | Register Name | R/W | Description |
|---|---|---|---|
| 0x12 | link_timer | RW | 21-bit auto-negotiation link timer. Set the link timer value from 0 to 16 ms in 8 ns steps (125 MHz clock periods). The reset value sets the link timer to 10 ms.<br>Bits 15:0 are written to word offset 0x12. Bit 0 of word offset 0x12 is always set to 0, thus any value written to it is ignored.<br>Bits 20:16 are written to word offset 0x13. The remaining bits are reserved and always set to 0. |
| 0x13 | | | |
| 0x14 | if_mode | RW | Interface mode. Use this register to specify the operating mode of the PCS function; 1000BASE-X or SGMII. |
| 0x15 | disable_read_timeout | RW | ■ Bit[0]—Set this bit to 1 to disable PCS register read timeout.<br>■ Bits[31:1]—unused. Set to read-only value 0. |
| 0x16 | read_timeout | RO | ■ Bit[0]—PCS register read timeout indication. Valid only when disable_read_timeout is set to 0. This bit is cleared after it is read.<br>The PCS function sets this bit to 0 when the register read ends normally; and sets this bit to 1 when the register read ends with a timeout.<br>■ Bits[31:1]—unused. |
| 0x17 – 0x1F | Reserved | — | — |

## Control Register (Word Offset 0x00)

Table 6–10 describes each field in the PCS control register.

**Table 6–10.** PCS Control Register Bit Descriptions

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 0:4 | Reserved | — | — |
| 5 | UNIDIRECTIONAL_ENABLE | RW | Enables the unidirectional function. This bit depends on bit 12. When bit 12 is one, this bit is ignored.<br>When bit 12 is zero, bit 5 indicates the unidirectional function:<br>■ A value of 1 enables transmit from media independent interface regardless of whether the PHY has determined that a valid link has been established.<br>■ A value of 0 enables transmit from media independent interface only when the PHY has determined that a valid link has been established.<br>The reset value of this bit is zero. |
| 6, 13 | SPEED_SELECTION | RO | Indicates the operating mode of the PCS function. Bits 6 and 13 are set to 1 and 0 respectively. This combination of values represent gigabit mode. |
| 7 | COLLISION_TEST | RO | The PCS function does not support half-duplex mode. This bit is always set to 0. |
| 8 | DUPLEX_MODE | RO | The PCS function only supports full-duplex mode. This bit is always set to 1. |
| 9 | RESTART_AUTO_NEGOTIATION | RW | Set this bit to 1 to restart the auto-negotiation sequence. For normal operation, set this bit to 0 (reset value). |
| 10 | ISOLATE | RW | Set this bit to 1 to isolate the PCS function from the MAC layer device. For normal operation, set this bit to 0 (reset value). |
| 11 | POWERDOWN | RW | Set this bit to 1 to power down the transceiver quad. The PCS function then asserts the powerdown signal to indicate the state it is in. |
| 12 | AUTO_NEGOTIATION_ENABLE | RW | Set this bit to 1 (reset value) to enable auto-negotiation. |
| 14 | LOOPBACK | RW | PHY loopback. Set this bit to 1 to implement loopback in the GX transceiver. For normal operation, set this bit to 0 (reset value). This bit is ignored if reduced ten-bit interface (RTBI) is implemented.<br>This feature is supported in all device families except the Cyclone IV GX device families. |
| 15 | RESET | RW | Self-clearing reset bit. Set this bit to 1 to generate a synchronous reset pulse which resets all the PCS function state machines, comma detection function, and 8b/10b encoder and decoder. For normal operation, set this bit to 0 (asynchronous reset value). |

## Status Register (Word Offset 0x01)

Table 6–11 describes each field in the `status` register. These bits are read only.

**Table 6–11.** Status Register Bit Descriptions

| Bit | Name | R/W | Description |
|---|---|---|---|
| 0 | EXTENDED_CAPABILITY | RO | A value of 1 indicates that the PCS function supports extended registers. |
| 1 | JABBER_DETECT | — | Unused. Always set to 0. |
| 2 | LINK_STATUS | RO | A value of 1 indicates that a valid link is established. A value of 0 indicates an invalid link.<br>If the link synchronization is lost, a 0 is latched. |
| 3 | AUTO_NEGOTIATION_ABILITY | RO | A value of 1 indicates that the PCS function supports auto-negotiation. |
| 4 | REMOTE_FAULT | — | Unused. Always set to 0. |
| 5 | AUTO_NEGOTIATION_COMPLETE | RO | A value of 1 indicates the following status:<br>■ The auto-negotiation process is completed.<br>■ The auto-negotiation control registers are valid. |
| 6 | MF_PREAMBLE_SUPPRESSION | — | Unused. Always set to 0. |
| 7 | UNIDIRECTIONAL_ABILITY | RO | A value of 1 indicates that the PCS is able to transmit from MII/GMII regardless of whether the PCS has established a valid link. |
| 8 | EXTENDED_STATUS | — | Unused. Always set to 0. |
| 9 | 100BASET2_HALF_DUPLEX | RO | The PCS function does not support 100Base-T2, 10-Mbps, 100BASE-X, and 100Base-T4 operation. Always set to 0. |
| 10 | 100BASET2_FULL_DUPLEX | | |
| 11 | 10MBPS_HALF_DUPLEX | | |
| 12 | 10MBPS_FULL_DUPLEX | | |
| 13 | 100BASE-X_HALF_DUPLEX | | |
| 14 | 100BASE-X_FULL_DUPLEX | | |
| 15 | 100BASE-T4 | | |

## Dev_Ability and Partner_Ability Registers (Word Offset 0x04 – 0x05)

The definition of each field in the `partner_ability` registers depends on the mode in which the PCS function operates.

In this mode, the definition of the fields in the `dev_ability` register are the same as the fields in the `partner_ability` register. The contents of these registers are valid only when the auto-negotiation completes (`AUTO_NEGOTIATION_COMPLETE` bit in the `status` register = 1).

### 1000BASE-X

Table 6–12 describes each field in the dev_ability and partner_ability registers in 1000BASE-X mode.

**Table 6–12.** Dev_Ability and Partner_Ability Registers Bits Description in 1000BASE-X

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 0:4 | Reserved | — | Always set these bits to 0. |
| 5 | FD | RW/RO *(1)(2)* | Full-duplex mode enable. A value of 1 indicates support for full duplex. |
| 6 | HD | | Half-duplex mode enable. A value of 1 indicates support for half duplex. |
| 7 | PS1 | | Pause support.<br>■ PS1=0 / PS2=0: Pause is not supported.<br>■ PS1=0 / PS2=1: Asymmetric pause toward link partner.<br>■ PS1=1 / PS2=0: Symmetric pause.<br>■ PS1=1/ PS2=1: Pause is supported on transmit and receive. |
| 8 | PS2 | | |
| 9:11 | Reserved | — | Always set these bits to 0. |
| 12 | RF1 | RW/RO *(1)(2)* | Remote fault condition:<br>■ RF1=0 / RF2=0: No error, link is valid (reset condition).<br>■ RF1=0 / RF2=1: Offline.<br>■ RF1=1 / RF2=0: Failure condition.<br>■ RF1=1 / RF2=1: Auto-negotiation error. |
| 13 | RF2 | | |
| 14 | ACK | | Acknowledge. A value of 1 indicates that the device has received three consecutive matching ability values from its link partner. |
| 15 | NP | | Next page. In dev_ability register, this bit is always set to 0. |

**Notes to Table 6–12:**

(1) All bits in the dev_ability register have RW access.

(2) All bits in the partner_ability register are read-only.

### SGMII MAC Mode Auto Negotiation

When the SGMII mode and the SGMII MAC mode auto-negotiation are enabled, the Triple-Speed Ethernet IP core ignores the value in the `dev_ability` register and automatically sets the value to `16'h4001` as specified in the SGMII specification for SGMII auto-negotiation.

When the auto-negotiation is complete, the Triple-Speed Ethernet IP core speed and the duplex mode will be resolved based on the value in the `partner_ability` register. The `partner_ability` register is received from the link partner during the auto-negotiation process.

Table 6–13 describes each field in the `partner_ability` register in SGMII MAC mode.

**Table 6–13.** Partner_Ability Register Bits Description in SGMII MAC Mode

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 0:9 | Reserved | — | — |
| 10:11 | `COPPER_SPEED(1:0)` | RO | Link partner interface speed:<br>■ 00: copper interface speed is 10 Mbps<br>■ 01: copper interface speed is 100 Mbps<br>■ 10: copper interface speed is 1 gigabit<br>■ 11: reserved |
| 12 | `COPPER_DUPLEX_STATUS` | RO | Link partner duplex capability:<br>■ 1: copper interface is capable of operating in full-duplex mode<br>■ 0: copper interface is capable of operating in half-duplex mode |
| 13 | Reserved | — | — |
| 14 | `ACK` | RO | Acknowledge. A value of 1 indicates that the link partner has received 3 consecutive matching ability values from the device. |
| 15 | `COPPER_LINK_STATUS` | RO | Copper link partner status:<br>■ 1: copper interface link is up<br>■ 0: copper interface link is down |

### SGMII PHY Mode Auto Negotiation

When the SGMII mode and the SGMII PHY mode auto-negotiation is enabled, set the `dev_ability` register before the auto-negotiation process so that the link partner can identify the copper speed, duplex status, and link status.

When the auto-negotiation is complete, Triple-Speed Ethernet IP core speed and the duplex mode will be resolved based on the value that you set in the `dev_ability` register. You can get the value for the `dev_ability` register from the system level where the Triple-Speed Ethernet IP core is integrated. If the IP core is integrated in the system level with another IP that resolves the copper speed and duplex information, use these values to set the `dev_ability` register.

Table 6–14 describes each field in the `dev_ability` register in SGMII PHY mode.

**Table 6–14.** Dev_Ability Register Bits Description in SGMII PHY Mode

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 0:9 | Reserved | — | Always set bit 0 to 1 and bits1–9 to 0. |
| 10:11 | `SPEED(1:0)` | RW | Link partner interface speed:<br>■ 00: copper interface speed is 10 Mbps<br>■ 01: copper interface speed is 100 Mbps<br>■ 10: copper interface speed is 1 gigabit<br>■ 11: reserved |
| 12 | `COPPER_DUPLEX_STATUS` | RW | Link partner duplex capability:<br>■ 1: copper interface is capable of operating in full-duplex mode<br>■ 0: copper interface is capable of operating in half-duplex mode<br>1 Gbps speed does not support half-duplex mode. |
| 13 | Reserved | — | Always set this bit to 0. |
| 14 | `ACK` | RO | Acknowledge. Value as specified in the IEEE 802.3z standard. |
| 15 | `COPPER_LINK_STATUS` | RW | Copper link partner status:<br>■ 1: copper interface link is up<br>■ 0: copper interface link is down |

## An_Expansion Register (Word Offset 0x06)

Table 6–15 describes each field in the `an_expansion` register.

**Table 6–15.** An_Expansion Register Description

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 0 | `LINK_PARTNER_AUTO_NEGOTIATION_ABLE` | RO | A value of 1 indicates that the link partner supports auto-negotiation. The reset value is 0. |
| 1 | `PAGE_RECEIVE` | RO | A value of 1 indicates that a new page is received with new partner ability available in the register `partner_ability`. The bit is set to 0 (reset value) when the system management agent performs a read access. |

**Table 6–15.** An_Expansion Register Description

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 2 | NEXT_PAGE_ABLE | — | Unused. Always set to 0. |
| 3:15 | Reserved | — | — |

## If_Mode Register (Word Offset 0x14)

Table 6–16 describes each field in the if_mode register.

**Table 6–16.** IF_Mode Register Description

| Bit(s) | Name | R/W | Description |
|---|---|---|---|
| 0 | SGMII_ENA | RW | Determines the PCS function operating mode. Setting this bit to 1 enables SGMII mode. Setting this bit to 0 enables 1000BASE-X gigabit mode. |
| 1 | USE_SGMII_AN | RW | This bit applies only to SGMII mode. Setting this bit to 1 causes the PCS function to be configured with the link partner abilities advertised during auto-negotiation. If this bit is set to 0, it is recommended for the PCS function to be configured with the SGMII_SPEED and SGMII_DUPLEX bits. |
| 2:3 | SGMII_SPEED[1:0] | RW | SGMII speed. When the PCS function operates in SGMII mode (SGMII_ENA = 1) and programed not to be automatically configured (USE_SGMII_AN = 0), set the speed as follows:<br>■ 00: 10 Mbps<br>■ 01: 100 Mbps<br>■ 10: Gigabit<br>■ 11: Reserved<br>These bits are ignored when SGMII_ENA is 0 or USE_SGMII_AN is 1. These bits are only valid if you only enable the SGMII mode and not the auto-negotiation mode. |
| 4 | SGMII_DUPLEX | RW | SGMII half-duplex mode. Setting this bit to 1 enables half duplex for 10/100 Mbps speed. This bit is ignored when SGMII_ENA is 0 or USE_SGMII_AN is 1. These bits are only valid if you only enable the SGMII mode and not the auto-negotiation mode. |
| 5 | SGMII_AN_MODE | RW | SGMII auto-negotiation mode:<br>■ 1: enable SGMII PHY mode<br>■ 0: enable SGMII MAC mode<br>This bit resets to 0, which defaults to SGMII MAC mode. |
| 6:15 | Reserved | — | — |

# Register Initialization

The Triple-Speed Ethernet MegaCore function supports various types of interface commonly used by the following Ethernet solutions:

- MII/GMII
- RGMII
- 10-bit Interface
- SGMII
- 1000BASE-X
- Management Data Input/Output (MDIO) for external PHY register configuration

When using the Triple-Speed Ethernet MegaCore function with an external interface, you must understand the requirements and initialize the registers.

Register initialization mainly performed in the following configurations:

- External PHY Initialization using MDIO (Optional)
- PCS Configuration Register Initialization
- MAC Configuration Register Initialization

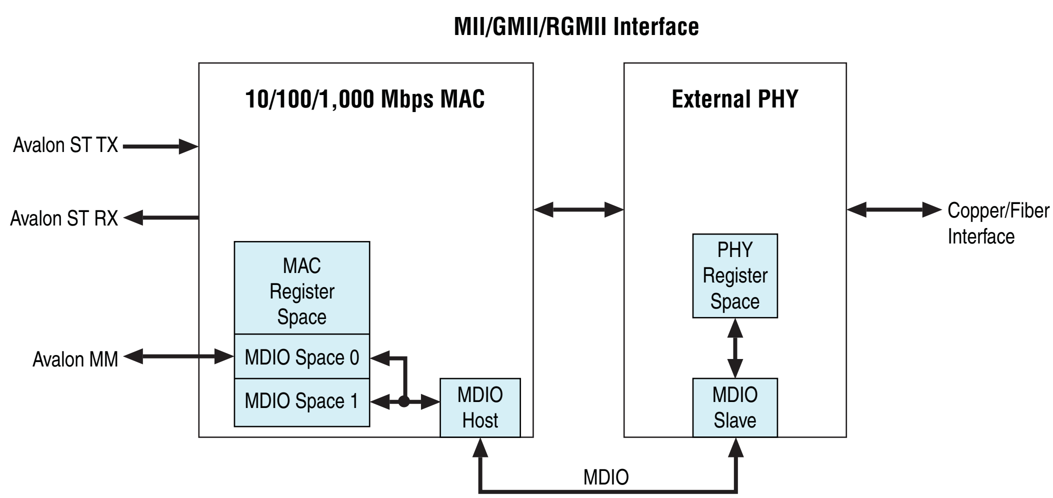
This section discusses the register initialization for the following examples of the Ethernet system using different MAC interfaces with recommended initialization sequences:

- Triple-Speed Ethernet System with MII/GMII or RGMII
- Triple-Speed Ethernet System with SGMII
- Triple-Speed Ethernet System with 1000BASE-X Interface

## Triple-Speed Ethernet System with MII/GMII or RGMII

Figure 6–2 shows the Triple-Speed Ethernet System with MII/GMII or RGMII with register initialization recommendation.

**Figure 6–2.** Triple-Speed Ethernet System with MII/GMII or RGMII



Use the following recommended initialization sequences for the example in Figure 6–2.

1. External PHY Initialization using MDIO

```
//Assume the External PHY Address is 0x0A

mdio_addr0 = 0x0A

//External PHY Register will Map to MDIO Space 0

Read/write to MDIO space 0 (dword offset 0x80 - 0x9F) = Read/write to
PHY Register 0 to 31
```

2. MAC Configuration Register Initialization

   a. Disable MAC Transmit and Receive Datapath

      Disable the MAC transmit and receive datapath before performing any changes to configuration.

```
//Set TX_ENA and RX_ENA bit to 0 in Command Config Register

Command_config Register = 0x00802220


//Read the TX_ENA and RX_ENA bit is set 0 to ensure TX and RX path is
disable

Wait Command_config Register = 0x00802220
```

   b. MAC FIFO Configuration

```
Tx_section_empty = Max FIFO size - 16

Tx_almost_full = 3

Tx_almost_empty = 8

Rx_section_empty = Max FIFO size - 16

Rx_almost_full = 8

Rx_almost_empty = 8

//Cut Throught Mode, Set this Threshold to 0 to enable Store and Forward
Mode

Tx_section_full = 16

//Cut Throught Mode, Set this Threshold to 0 to enable Store and Forward
Mode

Rx_section_full = 16
```

   c. MAC Address Configuration

```
//MAC address is 00-1C-23-17-4A-CB

mac_0 = 0x17231C00

mac_1 = 0x0000CB4A
```

   d. MAC Function Configuration

```
//Maximum Frame Length is 1518 bytes

Frm_length = 1518

//Minimum Inter Packet Gap is 12 bytes

Tx_ipg_length = 12

//Maximum Pause Quanta Value for Flow Control
```

```
Pause_quant = 0xFFFF
//Set the MAC with the following option:
// 100Mbps, User can get this information from the PHY status/PCS status
//Full Duplex, User can get this information from the PHY status/PCS
status
//Padding Removal on Receive
//CRC Removal
//TX MAC Address Insertion on Transmit Packet
//Select mac_0 and mac_1 as the source MAC Address
Command_config Register = 0x00800220
```

Figure 6–3 shows the settings for the `command_config` register.

**Figure 6–3.** Command_Config Register Settings

| 31 | 30..27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18..16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNT_RESET | Reserved | RX_ERR_DISC | ENA_10 | NO_LGTH_CHECK | CTRL_FRM_ENA | XOFF_GEN | WAKEUP | SLEEP | MAGIC_ENA | TX_ADDR_SEL | LOOP_ENA | MHASH_SEL | SW_RESET | LATE_COL | EXCESS_COL | HD_ENA | TX_ADDR_INS | PAUSE_IGNORE | PAUSE_FWD | CRC_FWD | PAD_EN | PROMIS_EN | ETH_SPEED | XON_GEN | RX_ENA | TX_ENA |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

e. Reset MAC

Altera recommends that you perform a software reset when there is a change in the MAC speed or duplex. The MAC software reset bit self-clears when the software reset is complete.

```
//Set SW_RESET bit to 1
Command_config Register = 0x00802220
Wait  Command_config Register = 0x00800220
```

f. Enable MAC Transmit and Receive Datapath

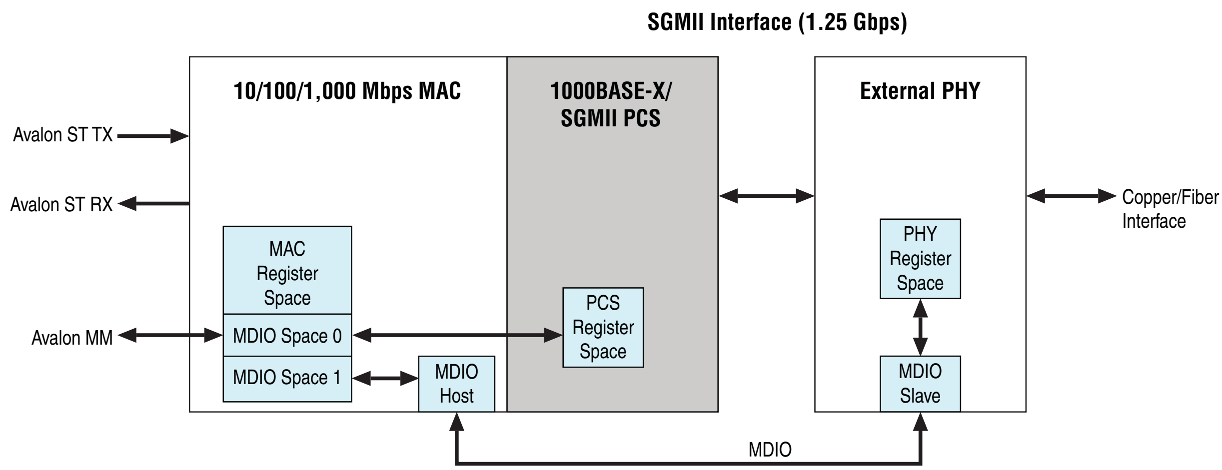
```
//Set TX_ENA and RX_ENA to 1 in Command Config Register
Command_config Register = 0x00802223


//Read the TX_ENA and RX_ENA bit is set 1 to ensure TX and RX path is
enable
Wait Command_config Register = 0x00802223
```

## Triple-Speed Ethernet System with SGMII

Figure 6–4 shows the Triple-Speed Ethernet System with SGMII with register initialization recommendation.

**Figure 6–4.** Triple-Speed Ethernet System with SGMII




Use the following recommended initialization sequences for the example in Figure 6–4.

1. External PHY Initialization using MDIO

Refer to step 1 in "Triple-Speed Ethernet System with MII/GMII or RGMII" on page 6–24.

2. PCS Configuration Register Initialization

   a. Set Auto Negotiation Link Timer

```
//Set Link timer to 1.6ms for SGMII
link_timer (address offset 0x12) = 0x0D40
Link_timer (address offset 0x13) = 0x03
```

   b. Configure SGMII

```
//Enable SGMII Interface and Enable SGMII Auto Negotiation
//SGMII_ENA = 1, USE_SGMII_AN = 1
if_mode = 0x0003
```

   c. Enable Auto Negotiation

```
//Enable Auto Negotiation
//AUTO_NEGOTIATION_ENA = 1, Bit 6,8,13 can be ignore
PCS Control Register = 0x1140
```

   d. PCS Reset

```
//PCS Software reset is recommended where there any configuration
changed
//RESET = 1
```

```
PCS Control Register = 0x9140
Wait PCS Control Register RESET bit is clear
```

3. MAC Configuration Register Initialization

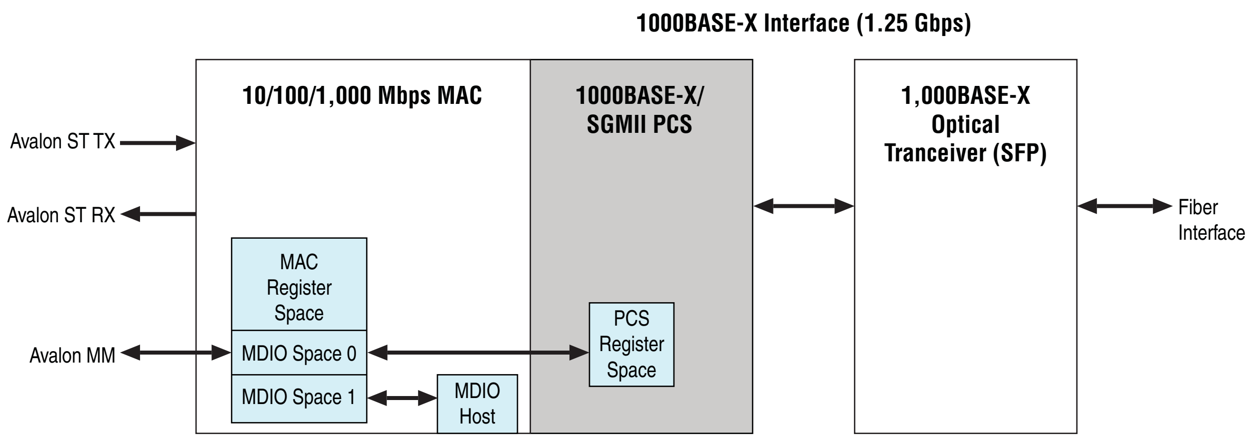
Refer to step 2 in "Triple-Speed Ethernet System with MII/GMII or RGMII" on page 6–24.

If 1000BASE-X/SGMII PCS is initialized, set the `ETH_SPEED` (bit 3) and `ENA_10` (bit 25) in `command_config` register to 0. If half duplex is reported in the PHY/PCS status register, set the `HD_ENA` (bit 10) to 1 in `command_config` register.

## Triple-Speed Ethernet System with 1000BASE-X Interface

Figure 6–5 shows the Triple-Speed Ethernet System with 1000BASE-X interface with register initialization recommendation.

**Figure 6–5.** Triple-Speed Ethernet System with 1000BASE-X Interface




Use the following recommended initialization sequences for the example in Figure 6–5.

1. External PHY Initialization using MDIO

Refer to step 1 in "Triple-Speed Ethernet System with MII/GMII or RGMII" on page 6–24.

2. PCS Configuration Register Initialization

   a. Set Auto Negotiation Link Timer

```
//Set Link timer to 10ms for 1000BASE-X
link_timer (address offset 0x12) = 0x12D0
link_timer (address offset 0x13) = 0x13
```

b. Configure SGMII

```
//1000BASE-X/SGMII PCS is default in 1000BASE-X Mode
//SGMII_ENA = 0, USE_SGMII_AN = 0
if_mode = 0x0000
```

c. Enable Auto Negotiation

```
//Enable Auto Negotiation
//AUTO_NEGOTIATION_ENA = 1, Bit 6,8,13 is Read Only
PCS Control Register = 0x1140
```

d. PCS Reset

```
//PCS Software reset is recommended where there any configuration
changed
//RESET = 1
PCS Control Register = 0x9140
Wait PCS Control Register RESET bit is clear
```

3. MAC Configuration Register Initialization

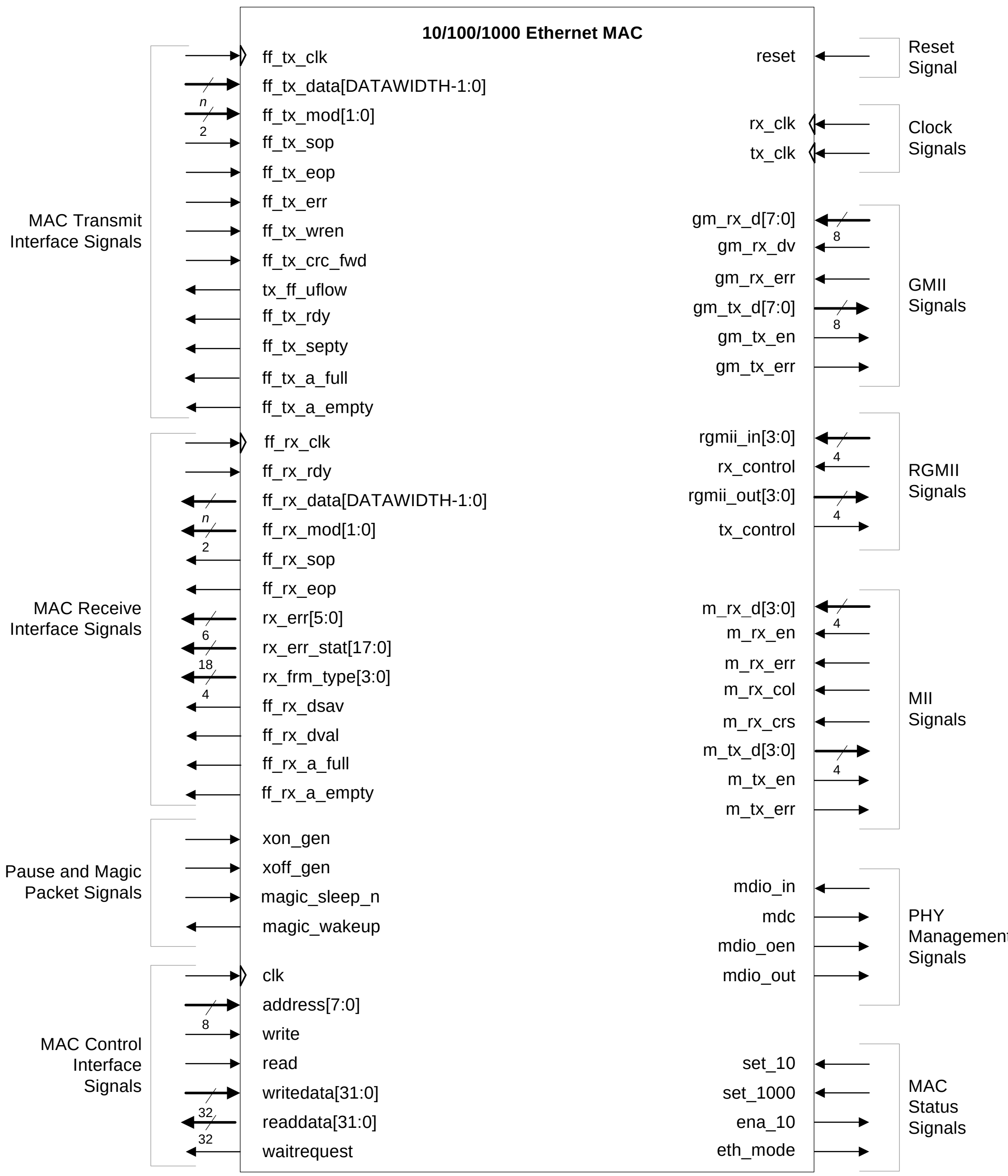
Refer to step 2 in "Triple-Speed Ethernet System with MII/GMII or RGMII" on page 6–24.

If 1000BASE-X/SGMII PCS is initialized, set the `ETH_SPEED` (bit 3) and `ENA_10` (bit 25) in `command_config` register to 0. If half duplex is reported in the PHY/PCS status register, set the `HD_ENA` (bit 10) to 1 in `command_config` register.

## 10/100/1000 Ethernet MAC Signals

Figure 7–1 shows all I/O signals of the 10/100/1000 Ethernet MAC function with internal FIFO buffers.

**Figure 7–1.** 10/100/1000 Ethernet MAC Signals



## Clock and Reset Signal

Data transfers on the MAC Ethernet-side interface are synchronous to the receive and transmit clocks. Table 7–1 describes the use of these clock signals.

**Table 7–1.** GMII/RGMII/MII Clock Signal

| Name | I/O | Description |
|---|---|---|
| tx_clk | I | GMII / RGMII/ MII transmit clock. Provides the timing reference for all GMII / MII transmit signals. The values of gm_tx_d[7:0] , gm_tx_en, gm_tx_err, and of m_tx_d[3:0], m_tx_en, m_tx_err are valid on the rising edge of tx_clk. |
| rx_clk | I | GMII /RGMII/ MII receive clock. Provides the timing reference for all rx related signals. The values of gm_rx_d[7:0], gm_rx_dv, gm_rx_err, and of m_rx_d[3:0], m_rx_en, m_rx_err are valid on the rising edge of rx_clk. |

Table 7–2 describes the reset signal.

**Table 7–2.** Reset Signal

| Name | I/O | Description |
|---|---|---|
| reset | I | Assert this signal to reset all logic in the MAC and PCS control interface. The signal must be asserted for at least three clock cycles. |

## MAC Control Interface Signals

The MAC control interface is an Avalon-MM slave port that provides access to the register space. Table 7–3 describes the signals that constitute the MAC control interface.

**Table 7–3.** MAC Control Interface Signals

| Name | Avalon-MM Signal Type | I/O | Description |
|---|---|---|---|
| clk | clk | I | Register access reference clock. Set the signal to a value less than or equal to 125 MHz. |
| write | write | I | Register write enable. |
| read | read | I | Register read enable. |
| address[7:0] | address | I | 32-bit word-aligned register address. |
| writedata[31:0] | writedata | I | Register write data. Bit 0 is the least significant bit. |
| readdata[31:0] | readdata | O | Register read data. Bit 0 is the least significant bit. |
| waitrequest | waitrequest | O | Register interface busy. Asserted during register read or register write access; deasserted when the current register access completes. |

## MAC Status Signals

Table 7–4 describes all MAC status signals which allow you to set the transfer mode of the Ethernet-side interface.

**Table 7–4.** MAC Status Signals

| Name | I/O | Description |
|---|---|---|
| eth_mode | O | Ethernet mode. This signal is set to 1 when the MAC function is configured to operate at 1000 Mbps; set to 0 when it is configured to operate at 10/100 Mbps. |
| ena_10 | O | 10 Mbps enable. This signal is set to 1 to indicate that the PHY interface should operate at 10 Mbps. Valid only when the eth_mode signal is set to 0. |
| set_1000 | I | Gigabit mode selection. Can be driven to 1 by an external device, for example a PHY device, to set the MAC function to operate in gigabit. When set to 0, the MAC is set to operate in 10/100 Mbps. This signal is ignored when the ETH_SPEED bit in the command_config register is set to 1. |
| set_10 | I | 10 Mbps selection. Can be driven to 1 by an external device, for example a PHY device, to indicate that the MAC function is connected to a 10-Mbps PHY device. When set to 0, the MAC function is set to operate in 100-Mbps or gigabit mode. This signal is ignored when the ETH_SPEED or ENA_10 bit in the command_config register is set to 1. The ENA_10 bit has a higher priority than this signal. |

## MAC Receive Interface Signals

Table 7–5 describes all interface signals associated with the MAC receive interface.

**Table 7–5.** MAC Receive Interface Signals (Part 1 of 2)

| Name | Avalon-ST Signal Type | I/O | Description |
|---|---|---|---|
| **Avalon-ST Signals** | | | |
| ff_rx_clk | clk | I | Receive clock. All signals on the Avalon-ST receive interface are synchronized on the rising edge of this clock. Set this clock to the frequency required to get the desired bandwidth on this interface. This clock can be completely independent from rx_clk. |
| ff_rx_dval | valid | O | Receive data valid. When asserted, this signal indicates that the data on the following signals are valid: ff_rx_data[(DATAWIDTH -1):0], ff_rx_sop, ff_rx_eop, rx_err[5:0], rx_frm_type[3:0], and rx_err_stat[17:0]. |
| ff_rx_data [(DATAWIDTH-1):0] | data | O | Receive data. When DATAWIDTH is 32, the first byte received is ff_rx_data[31:24] followed by ff_rx_data[23:16] and so forth. |
| ff_rx_mod[1:0] | empty | O | Receive data modulo. Indicates invalid bytes in the final frame word:<br>■ 11: ff_rx_data[23:0] is not valid<br>■ 10: ff_rx_data[15:0] is not valid<br>■ 01: ff_rx_data[7:0] is not valid<br>■ 00: ff_rx_data[31:0] is valid<br>This signal applies only when DATAWIDTH is set to 32. |

**Table 7–5.** MAC Receive Interface Signals (Part 2 of 2)

| Name | Avalon-ST Signal Type | I/O | Description |
|---|---|---|---|
| ff_rx_sop | startofpacket | O | Receive start of packet. Asserted when the first byte or word of a frame is driven on ff_rx_data[(DATAWIDTH-1):0]. |
| ff_rx_eop | endofpacket | O | Receive end of packet. Asserted when the last byte or word of frame data is driven on ff_rx_data[(DATAWIDTH-1):0]. |
| ff_rx_rdy | ready | I | Receive application ready. Assert this signal on the rising edge of ff_rx_clk when the user application is ready to receive data from the MAC function. |
| rx_err[5:0] | error | O | Receive error. Asserted with the final byte in the frame to indicate that an error was detected when receiving the frame. See Table 7–7 on page 7–5 for the bit description. |
| **Component-Specific Signals** | | | |
| ff_rx_dsav | — | O | Receive frame available. When asserted, this signal indicates that the internal receive FIFO buffer contains some data to be read but not necessarily a complete frame. The user application may want to start reading from the FIFO buffer.<br>This signal remains deasserted in the store and forward mode. |
| rx_frm_type[3:0] | — | O | Frame type. See Table 7–6 on page 7–5 for the bit description. |
| ff_rx_a_full | — | O | Asserted when the FIFO buffer reaches the almost-full threshold. |
| ff_rx_a_empty | — | O | Asserted when the FIFO buffer goes below the almost-empty threshold. |
| rx_err_stat[17:0] | — | O | rx_err_stat[17]: One indicates that the receive frame is a stacked VLAN frame.<br>rx_err_stat[16]: One indicates that the receive frame is either a VLAN or stacked VLAN frame.<br>rx_err_stat[15:0]: The value of the length/type field of the receive frame. |

Table 7–6 describes each bit of the rx_frm_type signal.

**Table 7–6.** rx_frm_type Bit Description

| Bit | Description |
|---|---|
| 3 | Indicates VLAN frames. Asserted with ff_rx_sop and remains asserted until the end of the frame. |
| 2 | Indicates broadcast frames. Asserted with ff_rx_sop and remains asserted until the end of the frame. |
| 1 | Indicates multicast frames. Asserted with ff_rx_sop and remains asserted until the end of the frame. |
| 0 | Indicates unicast frames. Asserted with ff_rx_sop and remains asserted until the end of the frame. |

Table 7–7 describes each bit of the rx_err signal.

**Table 7–7.** rx_err Bit Description

| Bit | Description |
|---|---|
| 5 | Collision error. Asserted when the frame was received with a collision. |
| 4 | Corrupted receive frame caused by PHY or PCS error. Asserted when the error is detected on the MII/GMII/RGMII. |
| 3 | Truncated receive frame. Asserted when the receive frame is truncated due to an overflow in the receive FIFO buffer. |
| 2 *(1)* | CRC error. Asserted when the frame is received with a CRC-32 error. This error bit applies only to frames with a valid length. See "Length Checking" on page 4–10. |
| 1 *(1)* | Invalid length error. Asserted when the receive frame has an invalid length as defined by the IEEE Standard 802.3. For more information on frame length, refer to "Length Checking" on page 4–10. |
| 0 | Receive frame error. Indicates that an error has occurred. It is the logical OR of rx_err[5:1]. |

**Note to Table 7–7:**

(1) Bits 1 and 2 are not mutually exclusive. Ignore CRC error rx_err[2] signal if it is asserted at the same time as the invalid length error rx_err[1] signal.

## MAC Transmit Interface Signals

Table 7–8 describes all signals associated with the MAC transmit interface.

**Table 7–8.** MAC Transmit Interface Signals (Part 1 of 2)

| Name | Avalon-ST Signal Type | I/O | Description |
|---|---|---|---|
| **Avalon-ST Signals** | | | |
| ff_tx_clk | clk | I | Transmit clock. All transmit signals are synchronized on the rising edge of this clock.<br>Set this clock to the required frequency to get the desired bandwidth on the Avalon-ST transmit interface. This clock can be completely independent from tx_clk. |
| ff_tx_wren | valid | I | Transmit data write enable. Assert this signal to indicate that the data on the following signals are valid: ff_tx_data[(DATAWIDTH-1):0], ff_tx_sop, and ff_tx_eop.<br>In cut-through mode, keep this signal asserted throughout the frame transmission. Otherwise, the frame is truncated and forwarded to the Ethernet-side interface with an error. |

**Table 7–8.** MAC Transmit Interface Signals (Part 2 of 2)

| Name | Avalon-ST Signal Type | I/O | Description |
|---|---|---|---|
| ff_tx_data [(DATAWIDTH-1):0] | data | I | Transmit data. DATAWIDTH can be either 8 or 32 depending on the FIFO data width configured. When DATAWIDTH is 32, the first byte transmitted is ff_tx_data[31:24] followed by ff_tx_data[23:16] and so forth. |
| ff_tx_mod[1:0] | empty | I | Transmit data modulo. Indicates invalid bytes in the final frame word:<br>■ 11: ff_tx_data[23:0] is not valid<br>■ 10: ff_tx_data[15:0] is not valid<br>■ 01: ff_tx_data[7:0] is not valid<br>■ 00: ff_tx_data[31:0] is valid<br>This signal applies only when DATAWIDTH is set to 32. |
| ff_tx_sop | startofpacket | I | Transmit start of packet. Assert this signal when the first byte in the frame (the first byte of the destination address) is driven on ff_tx_data. |
| ff_tx_eop | endofpacket | I | Transmit end of packet. Assert this signal when the last byte in the frame (the last byte of the FCS field) is driven on ff_tx_data. |
| ff_tx_err | error | I | Transmit frame error. Assert this signal with the final byte in the frame to indicate that the transmit frame is invalid. The MAC function forwards the invalid frame to the GMII with an error. |
| ff_tx_rdy | ready | O | MAC ready. When asserted, the MAC function is ready to accept data from the user application. |
| **Component-Specific Signals** | | | |
| ff_tx_crc_fwd | — | I | Transmit CRC insertion. Set this signal to 0 when ff_tx_eop is set to 1 to instruct the MAC function to compute a CRC and insert it into the frame. If this signal is set to 1, the user application is expected to provide the CRC. |
| tx_ff_uflow | — | O | Asserted when an underflow occurs on the transmit FIFO buffer. |
| ff_tx_septy | — | O | Deasserted when the FIFO buffer is filled to or above the section-empty threshold defined in the tx_section_empty register. User applications can use this signal to indicate when to stop writing to the FIFO buffer and initiate backpressure. |
| ff_tx_a_full | — | O | Asserted when the transmit FIFO buffer reaches the almost- full threshold. |
| ff_tx_a_empty | — | O | Asserted when the transmit FIFO buffer goes below the almost-empty threshold. |

## Pause and Magic Packet Signals

The pause and magic packet signals are component-specific signals. Table 7–9 describes these signals.

**Table 7–9.** Pause and Magic Packet Signals

| Name | I/O | Description |
|---|---|---|
| xon_gen | I | Assert this signal for at least 1 tx_clk clock cycle to trigger the generation of a pause frame with a 0 pause quanta. The MAC function generates the pause frame independent of the status of the receive FIFO buffer.<br>This signal is not in use in the following conditions:<br>■ Ignored when the xon_gen bit in the command_config register is set to 1.<br>■ Absent when the **Enable full duplex flow control** option is turned off. |
| xoff_gen | I | Assert this signal for at least one tx_clk clock cycle to trigger the generation of a pause frame with a pause quanta configured in the pause_quant register. The MAC function generates the pause frame independent of the status of the receive FIFO buffer.<br>This signal is not in use in the following conditions:<br>■ Ignored if the xoff_gen bit in the command_config register is set to 1.<br>■ Absent when the **Enable full duplex flow control** option is turned off. |
| magic_sleep_n | I | Assert this active-low signal to put the node into a power-down state.<br>If magic packets are supported (the MAGIC_ENA bit in the command_config register is set to 1), the receiver logic stops writing data to the receive FIFO buffer and the magic packet detection logic is enabled. Setting this signal to 1 restores the normal frame reception mode.<br>This signal is present only if the **Enable magic packet detection** option is turned on. |
| magic_wakeup | O | If the MAC function is in the power-down state, the MAC function asserts this signal to indicate that a magic packet has been detected and the node is requested to restore its normal frame reception mode.<br>This signal is present only if the **Enable magic packet detection** option is turned on. |

## MII/GMII/RGMII Signals

Table 7–10 describes the MII/GMII/RGMII signals.

**Table 7–10.** GMII/RGMII/MII Signals (Part 1 of 2)

| Name | I/O | Description |
|---|---|---|
| **GMII Transmit** | | |
| gm_tx_d[7:0] | I | GMII transmit data bus. |
| gm_tx_en | O | Asserted to indicate that the data on the GMII transmit data bus is valid. |
| gm_tx_err | O | Asserted to indicate to the PHY that the frame sent is invalid. |
| **GMII Receive** | | |
| gm_rx_d[7:0] | I | GMII receive data bus. |

**Table 7–10.** GMII/RGMII/MII Signals (Part 2 of 2)

| Name | I/O | Description |
|---|---|---|
| gm_rx_dv | I | Assert this signal to indicate that the data on the GMII receive data bus is valid. Keep this signal asserted during frame reception, from the first preamble byte until the last byte of the CRC field is received. |
| gm_rx_err | I | The PHY asserts this signal to indicate that the receive frame contains errors. |
| **RGMII Transmit** | | |
| rgmii_out[3:0] | O | RGMII transmit data bus. Drives gm_tx_d[3:0] on the positive edge of tx_clk and gm_tx_d[7:4] on the negative edge of tx_clk. |
| tx_control | O | Control output signal. Drives gm_tx_en on the positive edge of tx_clk and a logical derivative of (gm_tx_en XOR gm_tx_err) on the negative edge of tx_clk. |
| **RGMII Receive** | | |
| rgmii_in[3:0] | I | RGMII receive data bus. Expects gm_rx_d[3:0] on the positive edge of rx_clk and gm_rx_d[7:4] on the negative edge of rx_clk. |
| rx_control | I | RGMII control input signal. Expects gm_rx_dv on the positive edge of rx_clk and a logical derivative of (gm_rx_dv XOR gm_rx_err) on the negative edge of rx_clk. |
| **MII Transmit** | | |
| m_tx_d[3:0] | O | MII transmit data bus. |
| m_tx_en | O | Asserted to indicate that the data on the MII transmit data bus is valid. |
| m_tx_err | O | Asserted to indicate to the PHY device that the frame sent is invalid. |
| **MII Receive** | | |
| m_rx_d[3:0] | I | MII receive data bus. |
| m_rx_en | I | Assert this signal to indicate that the data on the MII receive data bus is valid. Keep this signal asserted during frame reception, from the first preamble byte until the last byte of the CRC field is received. |
| m_rx_err | I | The PHY asserts this signal to Indicate that the receive frame contains errors. |
| **MII PHY Status** | | |
| m_rx_col | I | Collision detection. The PHY asserts this signal to indicate a collision during frame transmission. This signal is not used in full- duplex or gigabit mode. |
| m_rx_crs | I | Carrier sense detection. The PHY asserts this signal to indicate that it has detected transmit or receive activity on the Ethernet line. This signal is not used in full-duplex or gigabit mode. |

## PHY Management Signals

Table 7–11 describes the PHY management signals.

**Table 7–11.** PHY Management Interface Signals

| Name | I/O | Description |
|---|---|---|
| mdio_in | I | Management data input. |
| mdio_out | O | Management data output. |

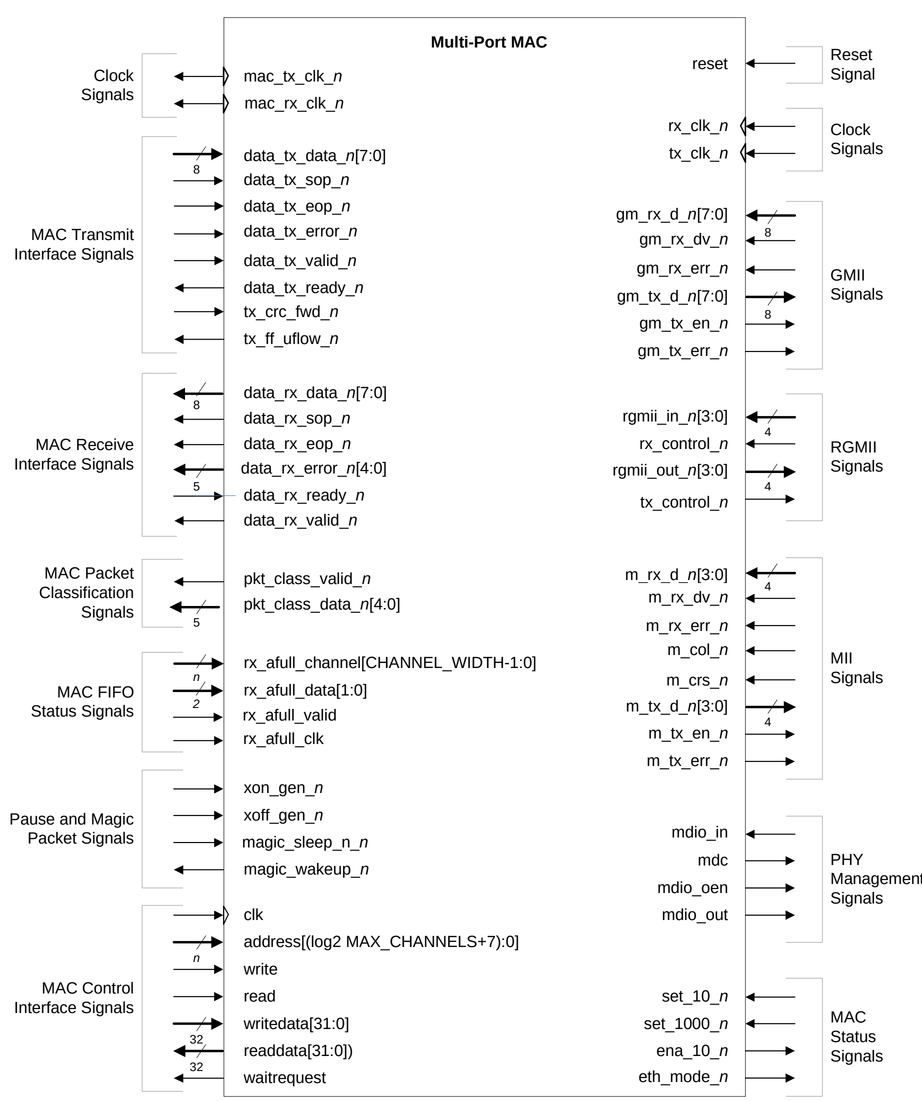
**Table 7–11.** PHY Management Interface Signals

| Name | I/O | Description |
|---|---|---|
| mdio_oen | O | An active-low signal that enables mdio_in or mdio_out. "MDIO Interface" on page 4–21 shows the MDIO connection. |
| mdc | O | Management data clock. Generated from the Avalon-MM interface clock signal, clk. Specify the division factor using the **Host clock divisor** parameter such that the frequency of this clock does not exceed 2.5 MHz. For more information about the parameters, refer to "Ethernet MAC Options" on page 3–2.<br>A data bit is shifted in/out on each rising edge of this clock. All fields are shifted in and out starting from the most significant bit. |

## 10/100/1000 Multiport Ethernet MAC Signals

Figure 7–2 shows all I/O signals of the 10/100/1000 multiport Ethernet MAC function, a MAC variation without internal FIFO buffers.

**Figure 7–2.** Multiport Ethernet MAC Signals



### Clock and Reset Signals

Table 7–12 describes the clock signals.

**Table 7–12.** Clock Signals

| Name | Avalon-ST Signal Type | I/O | Description |
|---|---|---|---|
| mac_rx_clk | clk | O | Receive MAC clock (2.5/25/125 MHz) for the Avalon-ST receive data and receive packet classification interfaces. |
| mac_tx_clk | clk | O | Transmit MAC clock (2.5/25/125 MHz) for the Avalon-ST transmit data interface. |

For more information about the rest of the clock and reset signals, refer to "Clock and Reset Signal" on page 7–2.

### MAC Receive Interface Signals

Table 7–13 describes all signals associated with the MAC receive interface.

**Table 7–13.** MAC Receive Interface Signals

| Name | Avalon-ST Signal Type | I/O | Description |
|---|---|---|---|
| data_rx_valid_*n* | valid | O | Receive data valid. When asserted, this signal indicates that the data on the following signals are valid: data_rx_data_*n*, data_rx_sop_*n*, data_rx_eop_*n*, and data_rx_error_*n*. |
| data_rx_data_*n*[7:0] | data | O | Receive data. |
| data_rx_sop_*n* | startofpacket | O | Receive start of packet. Asserted when the first byte or word of a frame is driven on data_rx_data_*n*. |
| data_rx_eop_*n* | endofpacket | O | Receive end of packet. Asserted when the last byte or word of frame data is driven on data_rx_data_*n*. |
| data_rx_ready_*n* | ready | I | Receive application ready. Assert this signal on the rising edge of data_rx_clk_*n* when the user application is ready to receive data from the MAC function.<br>If the user application is not ready to receive data, the packet is dropped or truncated with an error. |
| data_rx_error_*n*[4:0] | error | O | Receive error. Asserted with the final byte in the frame to indicate that an error was detected when receiving the frame. For the description of each bit, refer to the description of bits 5 to 1 in Table 7–7 on page 7–5. Bit 4 of this signal maps to bit 5 in the table and so forth. |

## MAC Transmit Interface Signals

Table 7–14 describes all signals associated with the MAC transmit interface.

**Table 7–14.** MAC Transmit Interface Signals

| Name | Avalon-ST Signal Type | I/O | Description |
|---|---|---|---|
| **Avalon-ST Signals** | | | |
| data_tx_valid_*n* | valid | I | Transmit data valid. Assert this signal to indicate that the data on the following signals are valid: data_tx_data_*n*, data_tx_sop_*n*, data_tx_eop_*n*, and data_tx_error_*n*. |
| data_tx_data_*n*[7:0] | data | I | Transmit data. |
| data_tx_sop_*n* | startofpacket | I | Transmit start of packet. Assert this signal when the first byte in the frame is driven on data_tx_data_*n*. |
| data_tx_eop_*n* | endofpacket | I | Transmit end of packet. Assert this signal when the last byte in the frame (the last byte of the FCS field) is driven on data_tx_data_*n*. |
| data_tx_error_*n* | error | I | Transmit frame error. Assert this signal with the final byte in the frame to indicate that the transmit frame is invalid. The MAC function then forwards the frame to the GMII with error. |
| data_tx_ready_*n* | ready | O | MAC ready. When asserted, this signal indicates that the MAC function is ready to accept data from the user application. |
| **Component-Specific Signal** | | | |
| tx_crc_fwd_*n* | — | I | Transmit CRC insertion. Assert this active-low signal when data_tx_eop_*n* is asserted for the MAC function to compute the CRC and insert it into the frame. Otherwise, the user application is expected to provide the CRC. |

## MAC Packet Classification Signals

The MAC packet classification interface is an Avalon-ST source port which streams out receive packet classifications. Table 7–15 describes the packet classification signals.

**Table 7–15.** MAC Packet Classification Signals

| Name | Avalon-ST Signal Type | I/O | Description |
|---|---|---|---|
| pkt_class_valid_*n* | valid | O | When asserted, this signal indicates that classification data is valid. |
| pkt_class_data_*n*[4:0] | data | O | Classification presented at the beginning of each packet:<br>■ Bit 4—Set to 1 for unicast frames.<br>■ Bit 3—Set to 1 for broadcast frames.<br>■ Bit 2—Set to 1 for multicast frames.<br>■ Bit 1—Set to 1 for VLAN frames.<br>■ Bit 0—Set to 1 for stacked VLAN frames. |

## MAC FIFO Status Signals

The MAC FIFO status interface is an Avalon-ST sink port which streams in information on the fill level of the external FIFO buffer to the MAC function. Table 7–16 describes the FIFO status signals.

**Table 7–16.** MAC FIFO Status Signals

| Signal Name | Avalon-ST Signal Type | I/O | Description |
|---|---|---|---|
| rx_afull_valid_*n* | valid | I | Assert this signal to indicate that the fill level of the external FIFO buffer, rx_afull_data_*n*[1:0],is valid. |
| rx_afull_data_*n*[1:0] | data | I | Carries the fill level of the external FIFO buffer:<br>■ rx_afull_data_*n*[1]—Set to 1 if the external receive FIFO buffer reaches the initial warning level indicating that it is almost full. Upon detecting this, the MAC function generates pause frames.<br>■ rx_afull_data_*n*[0]—Set to 1 if the external receive FIFO buffer reaches the critical level before it overflows. The FIFO buffer can be considered overflow if this bit is set to 1 in the middle of a packet transfer. |
| rx_afull_channel [(CHANNEL_WIDTH-1):0] | channel | I | The port number the status applies to. |
| rx_afull_clk | clk | I | The clock that drives the MAC FIFO status interface. |

## MAC Status Signals

For more information about the MAC status signals, refer to "MAC Status Signals" on page 7–3.

## Pause and Magic Packets Signals

For more information about pause and magic packet signals, refer to "Pause and Magic Packet Signals" on page 7–7.

## MII/GMII/RGMII Signals

For more information about the MII/GMII/RGMII signals, refer to "MII/GMII/RGMII Signals" on page 7–7.

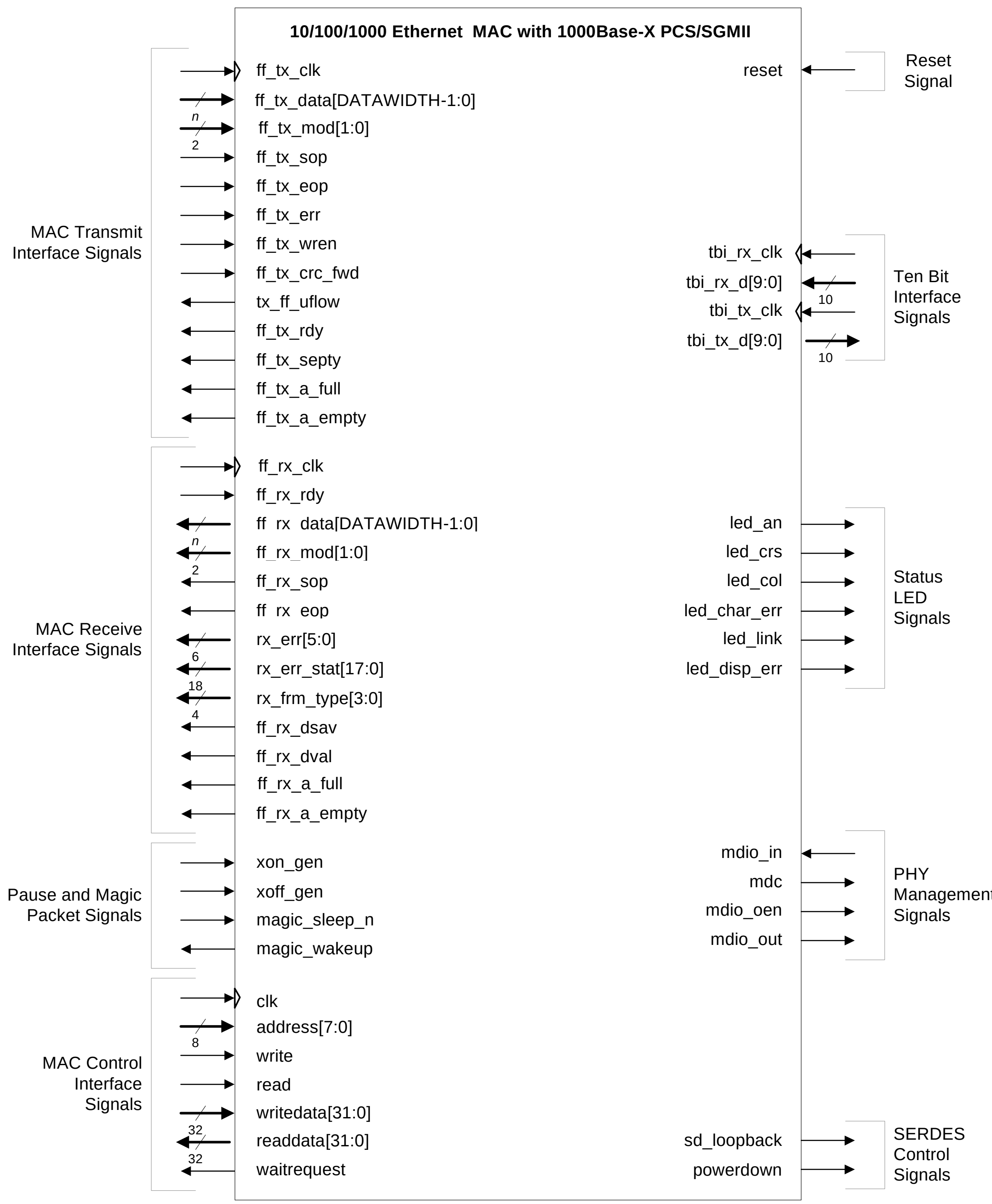
## PHY Management Signals

For more information about the PHY management signals, refer to "PHY Management Signals" on page 7–8.

# 10/100/1000 Ethernet MAC with 1000BASE-X/SGMII PCS Signals

Figure 7–3 shows all I/O signals of the 10/100/1000 Ethernet MAC, a MAC variation with internal FIFO buffers, with 1000BASE-X/SGMII PCS.

**Figure 7–3.** 10/100/1000 Ethernet MAC with 1000BASE-X/SGMII PCS Signals



## Reset Signals

For more information about the reset signal, refer to "Clock and Reset Signal" on page 7–2.

## MAC Control Interface Signals

For more information about MAC system-side signals, refer to "MAC Control Interface Signals" on page 7–2.

## MAC Receive Interface Signals

For more information about MAC system-side signals, refer to "MAC Receive Interface Signals" on page 7–3.

## MAC Transmit Interface Signals

For more information about MAC system-side signals, refer to "MAC Transmit Interface Signals" on page 7–5.

## TBI Interface Signals

If the core variation does not include an embedded PMA, the PCS block provides a 125-MHz ten-bit interface (TBI) to an external SERDES chip. Table 7–17 lists the PCS signals to an external SERDES chip.

**Table 7–17.** TBI Interface Signals for External SERDES Chip

| Name | I/O | Description |
|---|---|---|
| tbi_tx_d(9:0) | O | TBI transmit data. The PCS function transmits data on this bus synchronous to tbi_tx_clk. |
| tbi_tx_clk | I | 125-MHz TBI transmit clock from external SERDES, typically sourced by the local reference clock oscillator. |
| tbi_rx_clk | I | 125-MHz TBI receive clock from external SERDES, typically sourced by the line clock recovered from the encoded line stream. |
| tbi_rx_d[9:0] | I | TBI receive data. This bus carries the data from the external SERDES. Synchronize the bus with tbi_rx_clk. The data can be arbitrary aligned. |

## Status LED Control Signals

Table 7–18 lists the status LED control signals.

**Table 7–18.** Status LED Interface Signals (Part 1 of 2)

| Name | I/O | Description |
|---|---|---|
| led_link | O | When asserted, this signal indicates a successful link synchronization. |
| led_crs | O | When asserted, this signal indicates some activities on the transmit and receive paths. When deasserted, it indicates no traffic on the paths. |
| led_col | O | When asserted, this signal indicates that a collision was detected during frame transmission. This signal is always deasserted when the PCS function operates in standard 1000BASE-X mode or in full-duplex mode when SGMII is enabled. |
| led_an | O | Auto-negotiation status. The PCS function asserts this signal when an auto-negotiation completes. |

**Table 7–18.** Status LED Interface Signals (Part 2 of 2)

| Name | I/O | Description |
|---|---|---|
| led_char_err | O | 10-bit character error. Asserted for one tbi_rx_clk cycle when an erroneous 10-bit character is detected. |
| led_disp_err | O | 10-bit running disparity error. Asserted for one tbi_rx_clk cycle when a disparity error is detected. A running disparity error indicates that more than the previous and perhaps the current received group had an error. |

## SERDES Control Signals

Table 7–19 describes the functionality of the SERDES control signals.

**Table 7–19.** SERDES Control Signal

| Name | I/O | Description |
|---|---|---|
| powerdown | O | Power-down enable. Asserted when the PCS function is in power-down mode; deasserted when the PCS function is operating in normal mode. This signal is implemented only when an external SERDES is used. |
| sd_loopback | O | SERDES Loopback Control. Asserted when the PCS function operates in loopback mode. You can use this signal to configure an external SERDES device to operate in loopback mode. |

## Pause and Magic Packet Signals

For more information about pause and magic packet signals, refer to "Pause and Magic Packet Signals" on page 7–7.

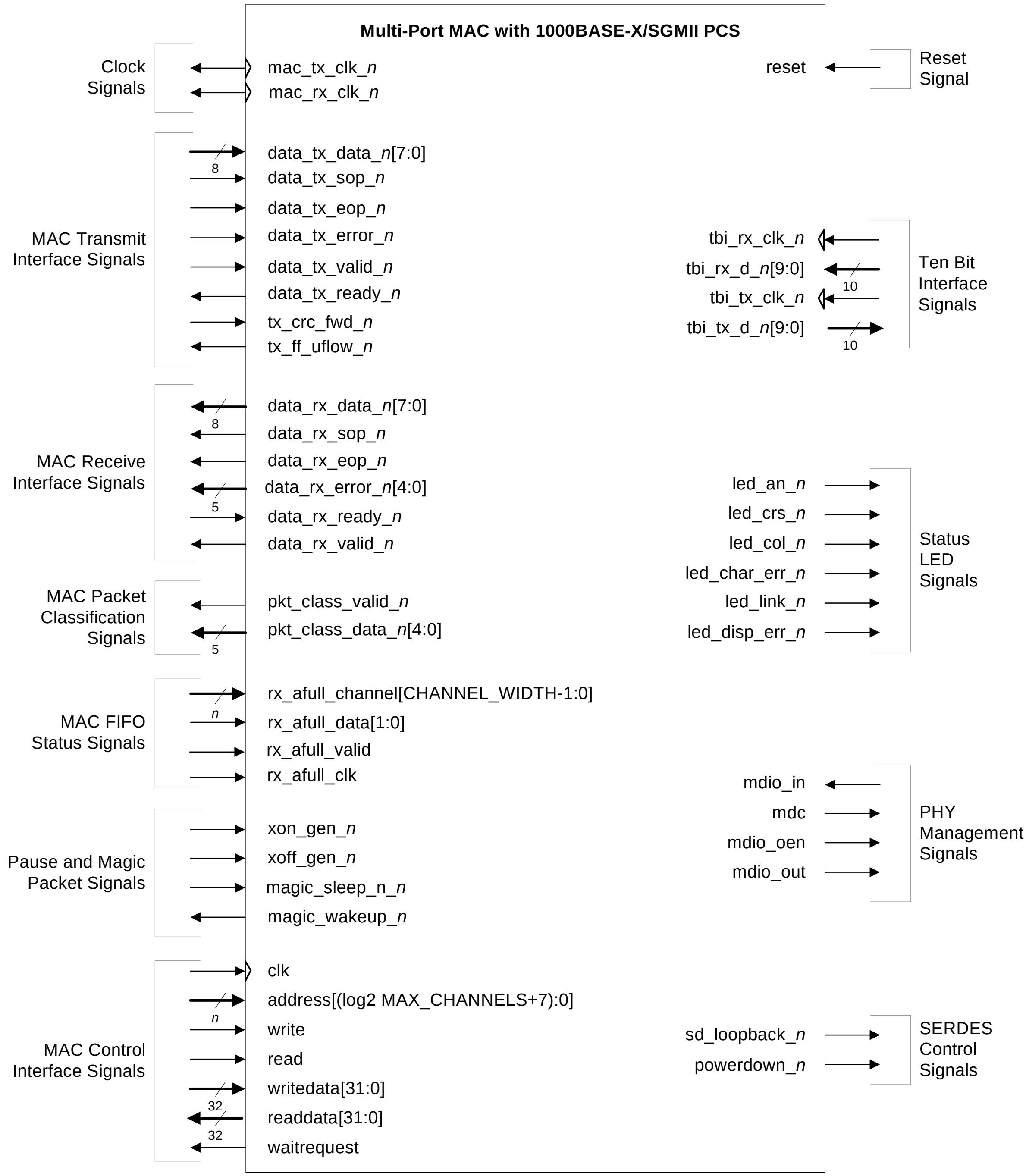
## PHY Management Signals

For more information about PHY Management signals, refer to Table 7–11 on page 7–8.

## 10/100/1000 Multiport Ethernet MAC with 1000BASE-X/SGMII PCS Signals

Figure 7–4 shows all I/O signals of the 10/100/1000 multiport Ethernet MAC, a MAC variation without internal FIFO buffers, with 1000BASE-X/SGMII PCS.

**Figure 7–4.** Multiport Ethernet MAC with 1000BASE-X/SGMII PCS Signals



For more information on the signals, refer to the respective sections shown in Table 7–20.

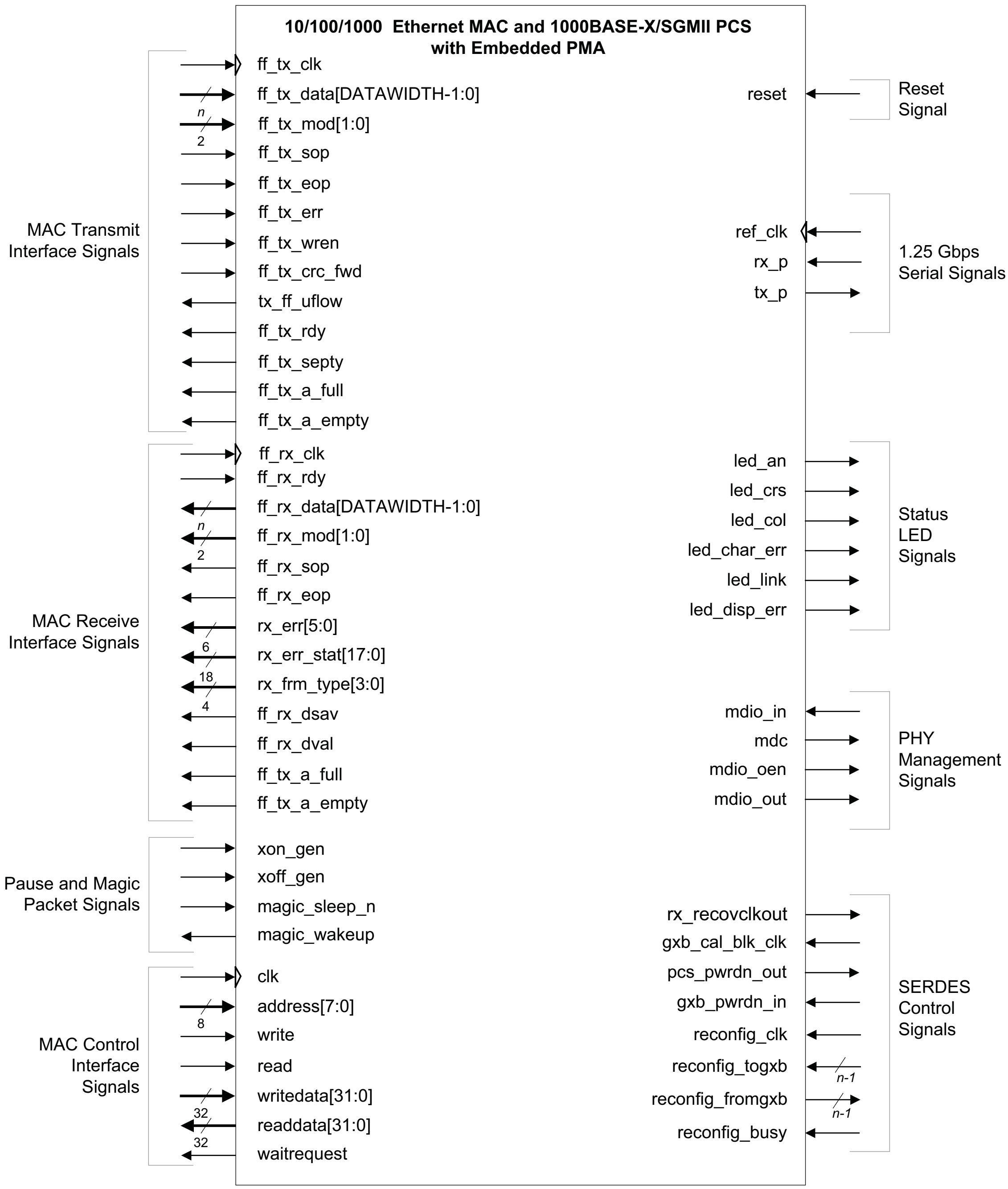
**Table 7–20.** References

| Interface Signal | Section |
|---|---|
| Clock and reset signals | "Clock and Reset Signal" on page 7–2 |
| MAC control interface | "MAC Control Interface Signals" on page 7–2 |
| MAC transmit interface | "MAC Transmit Interface Signals" on page 7–5 |
| MAC receive interface | "MAC Receive Interface Signals" on page 7–3 |
| MAC packet classification signals | "MAC Packet Classification Signals" on page 7–11 |
| MAC FIFO status signals | "MAC FIFO Status Signals" on page 7–12 |
| Pause and magic packet signals | "Pause and Magic Packet Signals" on page 7–7 |
| PHY management signals | "PHY Management Signals" on page 7–8 |
| Ten-bit interface | "TBI Interface Signals" on page 7–14 |
| Status LED signals | "Status LED Control Signals" on page 7–14 |
| SERDES control signals | "SERDES Control Signals" on page 7–15 |

## 10/100/1000 Ethernet MAC with 1000BASE-X/SGMII PCS and Embedded PMA Signals

Figure 7–5 shows all I/O signals of the 10/100/1000 Ethernet MAC, a MAC variation with internal FIFO buffers, and 1000BASE-X/SGMII PCS with an embedded PMA.

**Figure 7–5.** 10/100/1000 Ethernet MAC and 1000BASE-X/SGMII PCS With Embedded PMA Signals




**Note to Figure 7–5:**

(1) The SERDES control signals are present in variations targeting devices with GX transceivers. For Stratix II GX and Arria GX devices, the reconfiguration signals—`reconfig_clk`, `reconfig_togxb`, and `reconfig_fromgxb`—are included only when the option, **Enable transceiver dynamic reconfiguration**, is turned on. The reconfiguration signals—`gxb_cal_blk_clk`, `pcs_prwdwn_out`, `gxb_pwrdn_in`, `reconfig_clk`, and `reconfig_busy`—are not present in variations targeting Stratix V devices with GX transceivers.

### Reset Signals

For more information about the reset signal, refer to “Clock and Reset Signal” on page 7–2.

### MAC Control Interface Signals

For more information about MAC system-side signals, refer to “MAC Control Interface Signals” on page 7–2.

### MAC Receive Interface Signals

For more information about MAC system-side signals, refer to “MAC Receive Interface Signals” on page 7–3.

### MAC Transmit Interface Signals

For more information about MAC system-side signals, refer to “MAC Transmit Interface Signals” on page 7–5.

### 1.25 Gbps Serial Interface

If the variant includes an embedded PMA, the PMA provides a 1.25-GHz serial interface. Table 7–21 lists the MDI signals.

**Table 7–21.** 1.25 Gbps MDI Interface Signals

| Name | I/O | Description |
|---|---|---|
| ref_clk | I | 125 MHz local reference clock oscillator. |
| rx_p | I | Serial Differential Receive Interface. |
| tx_p | O | Serial Differential Transmit Interface. |

### Pause and Magic Packet Signals

For more information about pause and magic packet signals, refer to “Pause and Magic Packet Signals” on page 7–7.

### PHY Management Signals

For more information about PHY Management Signals, refer to Table 7–11 on page 7–8.

### Status LED Control Signals

For more information about Status LED Control Signals, refer to “Status LED Control Signals” on page 7–14.

## SERDES Control Signals

Table 7–22 describes the functionality of the SERDES control signals. These signals apply only to PMA blocks implemented in devices with GX transceivers.

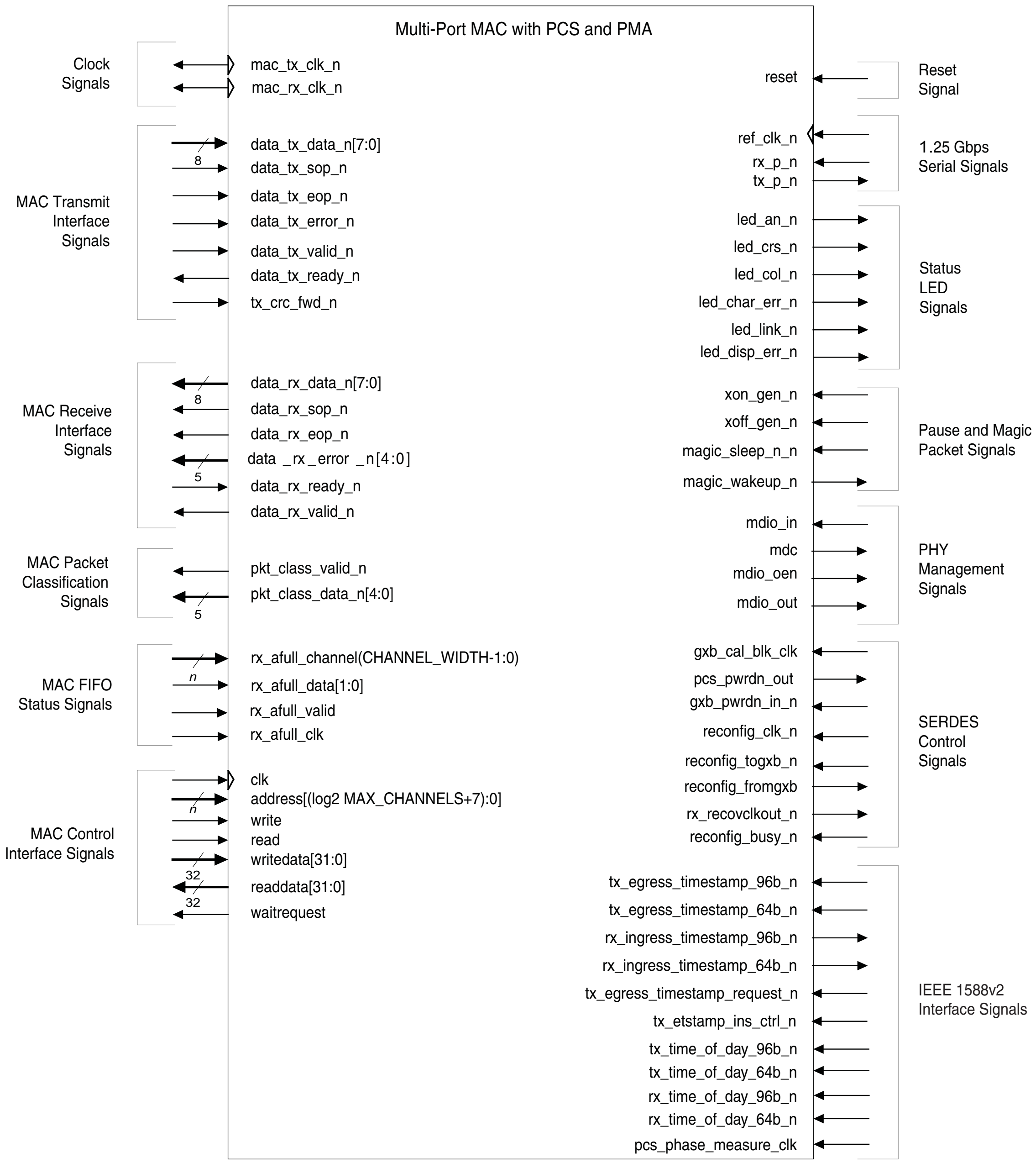
**Table 7–22.** SERDES Control Signal

| Name | I/O | Description |
|---|---|---|
| `rx_recovclkout` | O | Recovered clock from the PMA block. |
| `pcs_pwrdn_out` | O | Power-down status. Asserted when the PCS function is in power-down mode; deasserted when the PCS function is operating in normal mode. This signal is implemented only when an internal SERDES is used with the option to export the power-down signal.<br>This signal is not present in PMA blocks implemented in Stratix V devices with GX transceivers. |
| `gxb_pwrdn_in` | I | Power-down enable. Assert this signal to power down the transceiver quad block. This signal is implemented only when an internal SERDES is used with the option to export the power-down signal.<br>This signal is not present in PMA blocks implemented in Stratix V devices with GX transceivers. |
| `gxb_cal_blk_clk` | I | Calibration block clock for the ALT2GXB module (SERDES). This clock is typically tied to the 125 MHz `ref_clk`. Only implemented when an internal SERDES is used.<br>This signal is not present in PMA blocks implemented in Stratix V devices with GX transceivers. |
| `reconfig_clk` | I | Reference clock for the dynamic reconfiguration controller. If you use a dynamic reconfiguration controller in your design to dynamically control the transceiver, both the reconfiguration controller and the MegaCore function require this clock. This clock must operate between 37.5–50 MHz. Tie this clock low if you are not using an external reconfiguration controller.<br>This signal is not present in PMA blocks implemented in Stratix V devices with GX transceivers. |
| `reconfig_togxb[n:0]` | I | Driven from an external dynamic reconfiguration controller. Supports the selection of multiple transceiver channels for dynamic reconfiguration.<br>For PMA blocks implemented in Stratix V devices with GX transceivers, the bus width is [139:0]. For more information about the bus width for PMA blocks implemented in each device, refer to the *Dynamic Reconfiguration* chapter of the respective device handbook. |
| `reconfig_fromgxb[n:0]` | O | Connects to an external dynamic reconfiguration controller. The bus identifies the transceiver channel whose settings are being transmitted to the reconfiguration controller. Leave this bus disconnected if you are not using an external reconfiguration controller.<br>For more information about the bus width for PMA blocks implemented in each device, refer to the *Dynamic Reconfiguration* chapter of the respective device handbook. |
| `reconfig_busy` | I | Driven from an external dynamic reconfiguration controller. This signal will indicate the busy status of the dynamic reconfiguration controller during offset cancellation. Tie this signal to 1'b0 if you are not using an external reconfiguration controller.<br>This signal is not present in PMA blocks implemented in Stratix V devices with GX transceivers. |

## 10/100/1000 Multiport Ethernet MAC with 1000BASE-X/SGMII PCS and Embedded PMA

Figure 7–6 shows all I/O signals of the 10/100/1000 multiport Ethernet MAC with IEEE 1588v2 feature, a variation of MAC without internal FIFOs, with 1000BASE-X/SGMII PCS and embedded PMA function.

**Figure 7–6.** Multiport Ethernet MAC with IEEE 1588v2, 1000BASE-X/SGMII PCS and Embedded PMA Signals




**Note to Figure 7–6:**

(1) The SERDES control signals are present in variations targeting devices with GX transceivers. For Stratix II GX and Arria GX devices, the reconfiguration signals—reconfig_clk, reconfig_togxb, and reconfig_fromgxb—are included only when the **Enable transceiver dynamic reconfiguration** option is turned on. The reconfiguration signals—`gxb_cal_blk_clk`, `pcs_pwrdwn_out`, `gxb_pwrdn_in`, `reconfig_clk`, and `reconfig_busy`—are not present in variations targeting Stratix V devices with GX transceivers.

For more information on the signals, refer to the respective sections listed in Table 7–23.

**Table 7–23.** References

| Interface Signal | Section |
|---|---|
| Clock and reset signals | "Clock and Reset Signal" on page 7–2 |
| MAC control interface | "MAC Control Interface Signals" on page 7–2 |
| MAC transmit interface | "MAC Transmit Interface Signals" on page 7–5 |
| MAC receive interface | "MAC Receive Interface Signals" on page 7–3 |
| MAC packet classification signals | "MAC Packet Classification Signals" on page 7–11 |
| MAC FIFO status signals | "MAC FIFO Status Signals" on page 7–12 |
| Pause and magic packet signals | "Pause and Magic Packet Signals" on page 7–7 |
| PHY management signals | "PHY Management Signals" on page 7–8 |
| 1.25 Gbps Serial Signals | "1.25 Gbps Serial Interface" on page 7–19 |
| Status LED signals | "Status LED Control Signals" on page 7–14 |
| SERDES control signals | "SERDES Control Signals" on page 7–15 |
| IEEE 1588v2 RX Timestamp Signals | "IEEE 1588v2 RX Timestamp Signals" on page 7–22 |
| IEEE 1588v2 TX Timestamp Signals | "IEEE 1588v2 TX Timestamp Signals" on page 7–23 |
| IEEE 1588v2 TX Timestamp Request Signals | "IEEE 1588v2 TX Timestamp Request Signals" on page 7–24 |
| IEEE 1588v2 TX Insert Control Timestamp Signals | "IEEE 1588v2 TX Insert Control Timestamp Signals" on page 7–25 |
| IEEE 1588v2 ToD Clock Interface Signals | "IEEE 1588v2 Time-of-Day (ToD) Clock Interface Signals" on page 7–27 |

## IEEE 1588v2 RX Timestamp Signals

Table 7–24 describes the RX timestamp signals for the IEEE 1588v2 feature.

**Table 7–24.** IEEE 1588v2 RX Timestamp Interface Signals

| Signal | Direction | Width | Description |
|---|---|---|---|
| `rx_ingress_timestamp_96b_data_`*n* | Output | 96 | Carries the ingress timestamp on the receive datapath. Consists of 48-bit seconds field, 32-bit nanoseconds field, and 16-bit fractional nanoseconds field.<br>The MAC presents the timestamp for all receive frames and asserts this signal in the same clock cycle it asserts `rx_ingress_timestamp_96b_valid`. |
| `rx_ingress_timestamp_96b_valid` | Output | 1 | When asserted, this signal indicates that `rx_ingress_timestamp_96b_data` contains valid timestamp.<br>For all receive frame, the MAC asserts this signal in the same clock cycle it receives the start of packet (`avalon_st_rx_startofpacket` is asserted). |

**Table 7–24.** IEEE 1588v2 RX Timestamp Interface Signals

| Signal | Direction | Width | Description |
|---|---|---|---|
| `rx_ingress_timestamp_64b_data` | Output | 64 | Carries the ingress timestamp on the receive datapath. Consists of 48-bit nanoseconds field and 16-bit fractional nanoseconds field.<br>The MAC presents the timestamp for all receive frames and asserts this signal in the same clock cycle it asserts `rx_ingress_timestamp_64b_valid`. |
| `rx_ingress_timestamp_64b_valid` | Output | 1 | When asserted, this signal indicates that `rx_ingress_timestamp_64b_data` contains valid timestamp.<br>For all receive frame, the MAC asserts this signal in the same clock cycle it receives the start of packet (`avalon_st_rx_startofpacket` is asserted). |

## IEEE 1588v2 TX Timestamp Signals

Table 7–25 describes the TX timestamp signals for the IEEE 1588v2 feature.

**Table 7–25.** IEEE 1588v2 TX Timestamp Interface Signals

| Signal | Direction | Width | Description |
|---|---|---|---|
| `tx_egress_timestamp_96b_data_`*n* | Input | 96 | A transmit interface signal. This signal requests timestamp of frames on the TX path. The timestamp is used to calculate the residence time.<br>Consists of 48-bit seconds field, 32-bit nanoseconds field, and 16-bit fractional nanoseconds field. |
| `tx_egress_timestamp_96b_valid` | Input | 1 | A transmit interface signal. Assert this signal to indicate that a timestamp is obtained and a timestamp request is valid for the particular frame.<br>Assert this signal in the same clock cycle as the start of packet (`avalon_st_tx_startofpacket` is asserted). |
| `tx_egress_timestamp_96b_fingerprint` | Input | *n* | Configurable width fingerprint that returns with correlated timestamps.<br>The signal width is determined by the **TSTAMP_FP_WIDTH** parameter (default parameter value is 4). |
| `tx_egress_timestamp_64b_data` | Input | 64 | A transmit interface signal. This signal requests timestamp of frames on the TX path. The timestamp is used to calculate the residence time.<br>Consists of 48-bit nanoseconds field and 16-bit fractional nanoseconds field. |

**Table 7–25.** IEEE 1588v2 TX Timestamp Interface Signals

| Signal | Direction | Width | Description |
|---|---|---|---|
| `tx_egress_timestamp_64b_valid` | Input | 1 | A transmit interface signal. Assert this signal to indicate that a timestamp is obtained and a timestamp request is valid for the particular frame.<br>Assert this signal in the same clock cycle as the start of packet (`avalon_st_tx_startofpacket` or `avalon_st_tx_startofpacket_n` is asserted). |
| `tx_egress_timestamp_64b_fingerprint` | Input | *n* | Configurable width fingerprint that returns with correlated timestamps.<br>The signal width is determined by the **TSTAMP_FP_WIDTH** parameter (default parameter value is 4). |

## IEEE 1588v2 TX Timestamp Request Signals

Table 7–25 describes the TX timestamp request signals for the IEEE 1588v2 feature.

**Table 7–26.** IEEE 1588v2 TX Timestamp Request Signals

| Signal | Direction | Width | Description |
|---|---|---|---|
| `tx_egress_timestamp_request_valid_`*n* | Input | 1 | Assert this signal when a user-defined `tx_egress_timestamp` is required for a transmit frame.<br>Assert this signal in the same clock cycle as the start of packet (`avalon_st_tx_startofpacket` or `avalon_st_tx_startofpacket_n` is asserted). |
| `tx_egress_timestamp_request_fingerprint` | Input | *n* | Use this bus to specify fingerprint for the user-defined `tx_egress_timestamp`. The fingerprint is used to identify the user-defined timestamp.<br>The signal width is determined by the **TSTAMP_FP_WIDTH** parameter (default parameter value is 4).<br>The value of this signal is mapped to `user_fingerprint`.<br>This signal is only valid when you assert `tx_egress_timestamp_request_valid`. |

### IEEE 1588v2 TX Insert Control Timestamp Signals

Table 7–27 describes the TX insert control timestamp signals for the IEEE 1588v2 feature.

**Table 7–27.** IEEE 1588v2 TX Insert Control Timestamp Interface Signals (Part 1 of 2)

| Signal | Direction | Width | Description |
|---|---|---|---|
| tx_etstamp_ins_ctrl_timestamp_insert_*n* | Input | 1 | Assert this signal to insert egress timestamp into the associated frame.<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |
| tx_etstamp_ins_ctrl_timestamp_format | Input | 1 | Timestamp format of the frame, which the timestamp inserts.<br>0: 1588v2 format (48-bits second field + 32-bits nanosecond field + 16-bits correction field for fractional nanosecond)<br>Required offset location of timestamp and correction field.<br>1: 1588v1 format (32-bits second field + 32-bits nanosecond field)<br>Required offset location of timestamp.<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |
| tx_etstamp_ins_ctrl_residence_time_update | Input | 1 | Assert this signal to add residence time (egress timestamp –ingress timestamp) into correction field of PTP frame.<br>Required offset location of correction field.<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |
| tx_etstamp_ins_ctrl_ingress_timestamp_96b[] | Input | 96 | 96-bit format of ingress timestamp.<br>(48 bits second + 32 bits nanosecond + 16 bits fractional nanosecond).<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |
| tx_etstamp_ins_ctrl_ingress_timestamp_64b[] | Input | 64 | 64-bit format of ingress timestamp.<br>(48-bits nanosecond + 16-bits fractional nanosecond).<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |

**Table 7–27.** IEEE 1588v2 TX Insert Control Timestamp Interface Signals (Part 2 of 2)

| Signal | Direction | Width | Description |
|---|---|---|---|
| tx_etstamp_ins_ctrl_residence_time_calc_format | Input | 1 | Format of timestamp to be used for residence time calculation.<br>0: 96-bits (96-bits egress timestamp - 96-bits ingress timestamp).<br>1: 64-bits (64-bits egress timestamp - 64-bits ingress timestamp).<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |
| tx_etstamp_ins_ctrl_checksum_zero | Input | 1 | Assert this signal to set the checksum field of UDP/IPv4 to zero.<br>Required offset location of checksum field.<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |
| tx_etstamp_ins_ctrl_checksum_correct | Input | 1 | Assert this signal to correct UDP/IPv6 packet checksum, by updating the checksum correction, which is specified by checksum correction offset.<br>Required offset location of checksum correction.<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |
| tx_etstamp_ins_ctrl_offset_timestamp | Input | 1 | The location of the timestamp field, relative to the first byte of the packet.<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |
| tx_etstamp_ins_ctrl_offset_correction_field[] | Input | 16 | The location of the correction field, relative to the first byte of the packet.<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |
| tx_etstamp_ins_ctrl_offset_checksum_field[] | Input | 16 | The location of the checksum field, relative to the first byte of the packet.<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |
| tx_etstamp_ins_ctrl_offset_checksum_correction[] | Input | 16 | The location of the checksum correction field, relative to the first byte of the packet.<br>Assert this signal in the same clock cycle as the start of packet (avalon_st_tx_startofpacket is asserted). |

## IEEE 1588v2 Time-of-Day (ToD) Clock Interface Signals

Table 7–28 describes the ToD clock interface signals for the IEEE 1588v2 feature.

**Table 7–28.** IEEE 1588v2 ToD Clock Interface Signals

| Signal | Direction | Width | Description |
|---|---|---|---|
| tx_time_of_day_96b_data_*n* | Input | 96 | Use this bus to carry the time-of-day from external ToD module to 96-bit MAC TX clock.<br>Consists of 48 bits seconds field, 32 bits nanoseconds field, and 16 bits fractional nanoseconds field |
| rx_time_of_day_96b_data | Input | 96 | Use this bus to carry the time-of-day from external ToD module to 96-bit MAC RX clock.<br>Consists of 48 bits seconds field, 32 bits nanoseconds field, and 16 bits fractional nanoseconds field |
| tx_time_of_day_64b_data | Input | 64 | Use this bus to carry the time-of-day from external ToD module to 64-bit MAC TX clock.<br>Consists of 48-bit nanoseconds field and 16-bit fractional nanoseconds field |
| rx_time_of_day_64b_data | Input | 64 | Use this bus to carry the time-of-day from external ToD module to 64-bit MAC RX clock.<br>Consists of 48-bit nanoseconds field and 16-bit fractional nanoseconds field |

## IEEE 1588v2 PCS Phase Measurement Clock Signal

Table 7–28 describes the PCS phase measurement clock signal for the IEEE 1588v2 feature.

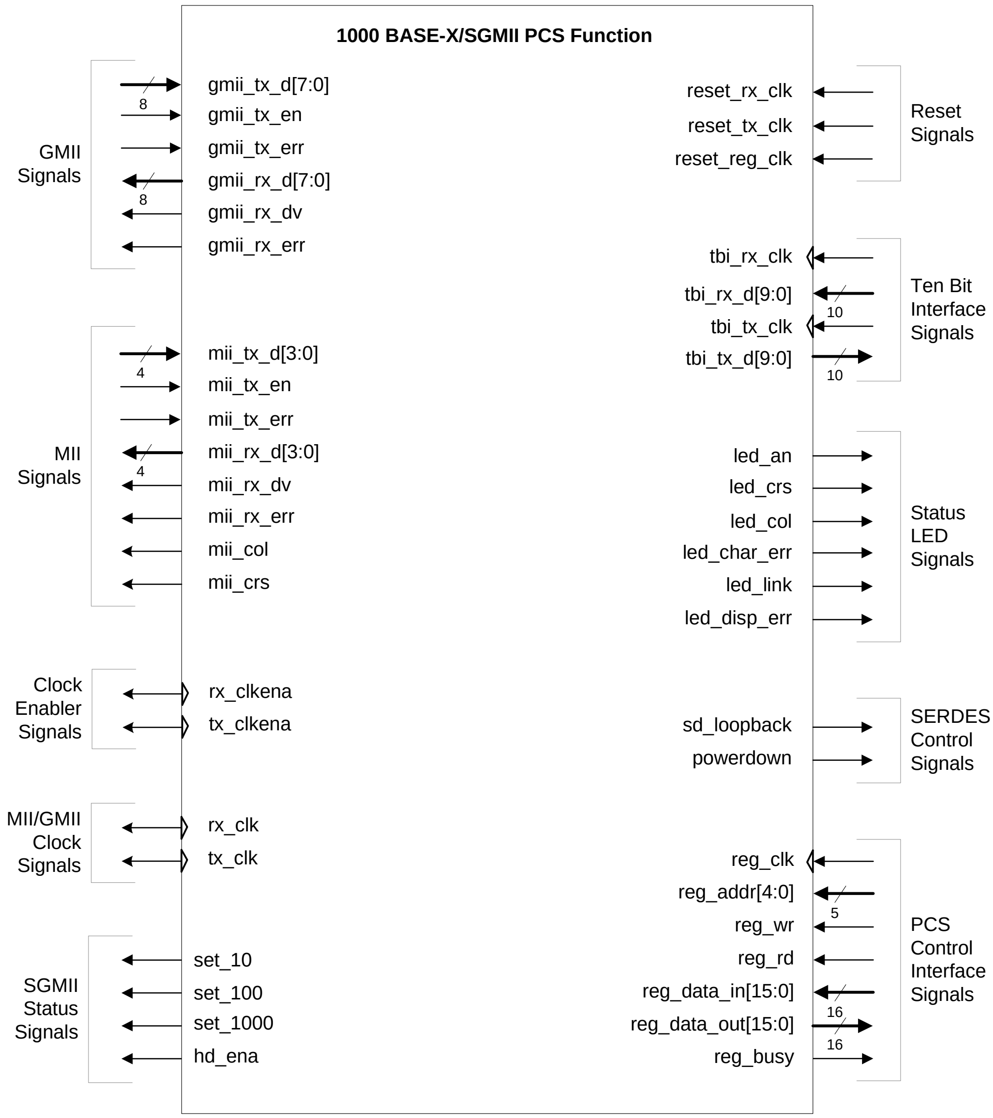
**Table 7–29.** IEEE 1588v2 PCS Phase Measurement Clock Signal

| Signal | Direction | Width | Description |
|---|---|---|---|
| pcs_phase_measure_clk | Input | 1 | Sampling clock to measure the latency through the PCS FIFO buffer. The recommended frequency is 80 MHz. |

# 1000BASE-X/SGMII PCS Signals

Figure 7–7 shows all I/O signals of the 1000BASE-X/SGMII PCS function.

**Figure 7–7.** 1000BASE-X/SGMII PCS Function Signals




**Note to Figure 7–7:**

(1) The clock enabler signals are present only in SGMII mode.

## PCS Control Interface Signals

Table 7–30 describes the signals that constitute the PCS control interface.

**Table 7–30.** Register Interface Signals

| Name | Avalon-MM Signal Type | I/O | Description |
|---|---|---|---|
| reg_clk | clk | I | Register access reference clock. Set the signal to a value less than or equal to 125-MHz. |
| reset_reg_clk | reset | I | Active-high reset signal for reg_clk clock domain. |
| reg_wr | write | I | Register write enable. |
| reg_rd | read | I | Register read enable. |
| reg_addr[4:0] | address | I | 16-bit word-aligned register address. |
| reg_data_in[15:0] | writedata | I | Register write data. Bit 0 is the least significant bit. |
| reg_data_out[15:0] | readdata | O | Register read data. Bit 0 is the least significant bit. |
| reg_busy | waitrequest | O | Register interface busy. Asserted during register read or register write. A value of 0 indicates that the read or write is complete. |

## Reset Signals

Table 7–31 describes the reset signals.

**Table 7–31.** Reset Signals

| Name | I/O | Description |
|---|---|---|
| reset_rx_clk | I | Active-high reset signal for PCS rx_clk clock domain. Assert this signal to reset the logic synchronized by rx_clk. |
| reset_tx_clk | I | Active-high reset signal for PCS tx_clk clock domain. Assert this signal to reset the logic synchronized by tx_clk. |

## MII/GMII Clocks and Clock Enablers

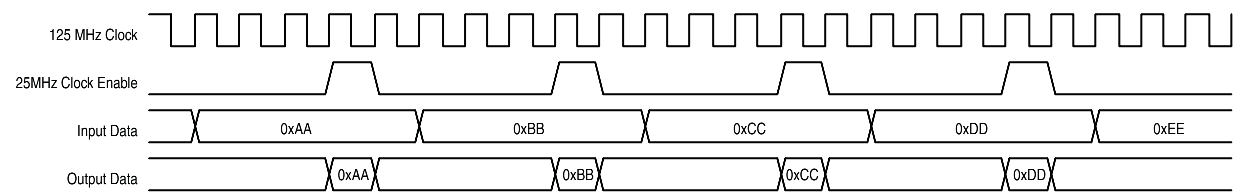
Data transfers on the MII/GMII interface are synchronous to the receive and transmit clocks. Table 7–32 describes these clock signals.

**Table 7–32.** MAC Clock Signals

| Name | I/O | Description |
|---|---|---|
| rx_clk | O | Receive clock. This clock is derived from the TBI clock tbi_rx_clk and set to 125 MHz. |
| tx_clk | O | Transmit clock. This clock is derived from the TBI clock tbi_tx_clk and set to 125 MHz. |
| rx_clkena | O | Receive clock enabler. In SGMII mode, this signal enables rx_clk. |
| tx_clkena | O | Transmit clock enabler. In SGMII mode, this signal enables tx_clk. |

Figure 7–8 shows the behavior of the clock enabler signal.

**Figure 7–8.** Clock Enabler Signal




## GMII

Table 7–33 describes the GMII transmit and receive signals.

**Table 7–33.** GMII Signals

| Name | I/O | Description |
|---|---|---|
| **GMII Transmit Interface** | | |
| `gmii_tx_d[7:0]` | I | GMII transmit data bus. |
| `gmii_tx_en` | I | Assert this signal to indicate that the data on `gmii_tx_d[7:0]` is valid. |
| `gmii_tx_err` | I | Assert this signal to indicate to the PHY device that the current frame sent is invalid. |
| **GMII Receive Interface** | | |
| `gmii_rx_d[7:0]` | O | GMII receive data bus. |
| `gmii_rx_dv` | O | Asserted to indicate that the data on `gmii_rx_d[7:0]` is valid. Stays asserted during frame reception, from the first preamble byte until the last byte in the CRC field is received. |
| `gmii_rx_err` | O | Asserted by the PHY to indicate that the current frame contains errors. |

## MII

Table 7–34 describes the MII transmit and receive signals.

**Table 7–34.** MII Signals (Part 1 of 2)

| Name | I/O | Description |
|---|---|---|
| **MII Transmit Interface** | | |
| `mii_tx_d[3:0]` | I | MII transmit data bus. |
| `mii_tx_en` | I | Assert this signal to indicate that the data on `mii_tx_d[3:0]` is valid. |
| `mii_tx_err` | I | Assert this signal to indicate to the PHY device that the frame sent is invalid. |
| **MII Receive Interface** | | |
| `mii_rx_d[3:0]` | O | MII receive data bus. |
| `mii_rx_dv` | O | Asserted to indicate that the data on `mii_rx_d[3:0]` is valid. The signal stays asserted during frame reception, from the first preamble byte until the last byte of the CRC field is received. |
| `mii_rx_err` | O | Asserted by the PHY to indicate that the current frame contains errors. |

**Table 7–34.** MII Signals (Part 2 of 2)

| Name | I/O | Description |
|---|---|---|
| mii_col | Out | Collision detection. Asserted by the PCS function to indicate that a collision was detected during frame transmission. |
| mii_crs | Out | Carrier sense detection. Asserted by the PCS function to indicate that a transmit or receive activity is detected on the Ethernet line. |

## SGMII Status Signals

The SGMII status signals provide status information to the PCS block. When the PCS is instantiated standalone, these signals are inputs to the MAC and serve as interface control signals for that block. Table 7–35 describes the SGMII status signals.

**Table 7–35.** SGMII Status Signals

| Name | I/O | Description |
|---|---|---|
| set_1000 | O | Gigabit mode enabled. In 1000BASE-X, this signal is always set to 1. In SGMII, this signal is set to 1 if one of the following conditions is met:<br>■ the USE_SGMII_AN bit is set to 1 and a gigabit link is established with the link partner, as decoded from the partner_ability register<br>■ the USE_SGMII_AN bit is set to 0 and the SGMII_SPEED bit is set to 10 |
| set_100 | O | 100 -Mbps mode enabled. In 1000BASE-X, this signal is always set to 0. In SGMII, this signal is set to 1 if one of the following conditions is met:<br>■ the USE_SGMII_AN bit is set to 1 and a 100Mbps link is established with the link partner, as decoded from the partner_ability register<br>■ the USE_SGMII_AN bit is set to 0 and the SGMII_SPEED bit is set to 01 |
| set_10 | O | 10 -Mbps mode enabled. In 1000BASE-X, this signal is always set to 0. In SGMII, this signal is set to 1 if one of the following conditions is met:<br>■ the USE_SGMII_AN bit is set to 1 and a 10Mbps link is established with the link partner, as decoded from the partner_ability register<br>■ the USE_SGMII_AN bit is set to 0 and the SGMII_SPEED bit is set to 00 |
| hd_ena | O | Half-duplex mode enabled. In 1000BASE-X, this signal is always set to 0. In SGMII, this signal is set to 1 if one of the following conditions is met:<br>■ the USE_SGMII_AN bit is set to 1 and a half-duplex link is established with the link partner, as decoded from the partner_ability register<br>■ the USE_SGMII_AN bit is set to 0 and the SGMII_DUPLEX bit is set to 1 |

## TBI Interface Signals for External SERDES Chip

For more information about TBI Interface signals, refer to "TBI Interface Signals" on page 7–14.

## Status LED Control Signals

For more information about Status LED signals, refer to "Status LED Control Signals" on page 7–14.

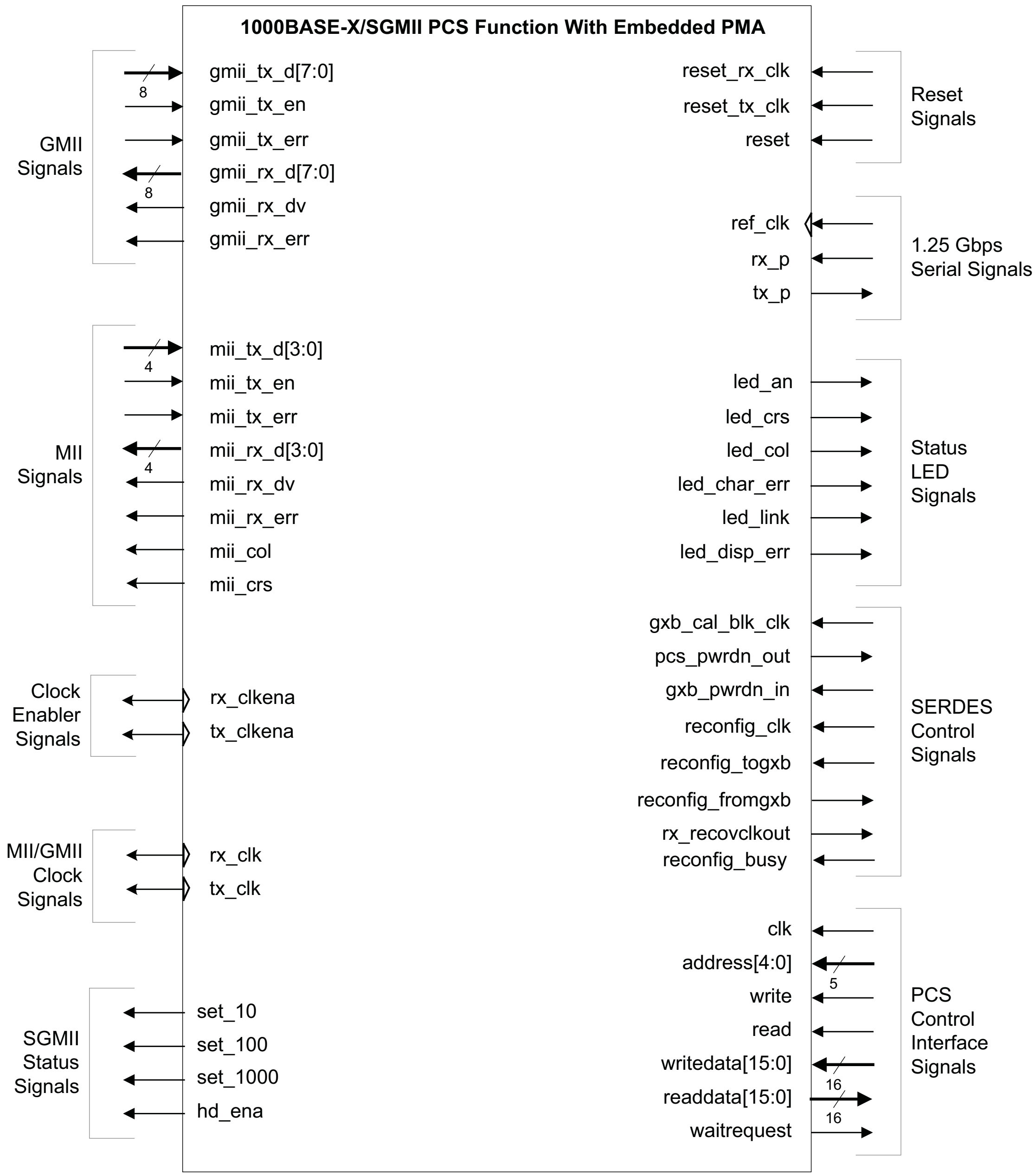
## SERDES Control Signals

For more information about SERDES Control signals, refer to"SERDES Control Signals" on page 7–15.

## 1000BASE-X/SGMII PCS and PMA Signals

Figure 7–9 shows all I/O signals of the 1000BASE-X/SGMII PCS function with an embedded PMA.

**Figure 7–9.** 1000BASE-X/SGMII PCS Function and PMA Signals




**Notes to Figure 7–9:**

(1) The clock enabler signals are present only in SGMII mode.

(2) The SERDES control signals are present in variations targeting devices with GX transceivers. For Stratix II GX and Arria GX devices, the reconfiguration signals—`reconfig_clk`, `reconfig_togxb`, and `reconfig_fromgxb`—are included only when the option, **Enable transceiver dynamic reconfiguration**, is turned on. The reconfiguration signals—`gxb_cal_blk_clk`, `pcs_pwrdwn_out`, `gxb_pwrdn_in`, `reconfig_clk`, and `reconfig_busy`—are not present in variations targeting Stratix V devices with GX transceivers.

For more information on the signals, refer to the respective sections shown in Table 7–36.

**Table 7–36.** References

| Interface Signal | Section |
|---|---|
| Reset signals | "Reset Signals" on page 7–14 |
| MII/GMII clocks and clock enablers | "MII/GMII Clocks and Clock Enablers" on page 7–29 |
| PCS control interface | "PCS Control Interface Signals" on page 7–29 |
| GMII signals | "GMII" on page 7–30 |
| MII signals | "MII" on page 7–30 |
| SGMII status signals | "SGMII Status Signals" on page 7–31 |
| 1.25 Gbps Serial Signals | "1.25 Gbps Serial Interface" on page 7–19 |
| Status LED signals | "Status LED Control Signals" on page 7–14 |
| SERDES control signals | "SERDES Control Signals" on page 7–15 |

# Timing

This section shows the timing on the Triple-Speed Ethernet transmit and receive interfaces as well as the timestamp signals for the IEEE 1588v2 feature.

For information on Avalon-MM control interface timing, refer to *Figure 3-3* in *Avalon Memory-Mapped Interfaces* chapter of *Avalon Interface Specifications*.

## Avalon-ST Receive Interface

Figure 7–10 shows the receive operation for MAC core variations with internal FIFO buffers.

**Figure 7–10.** Receive Operation—MAC With Internal FIFO Buffers

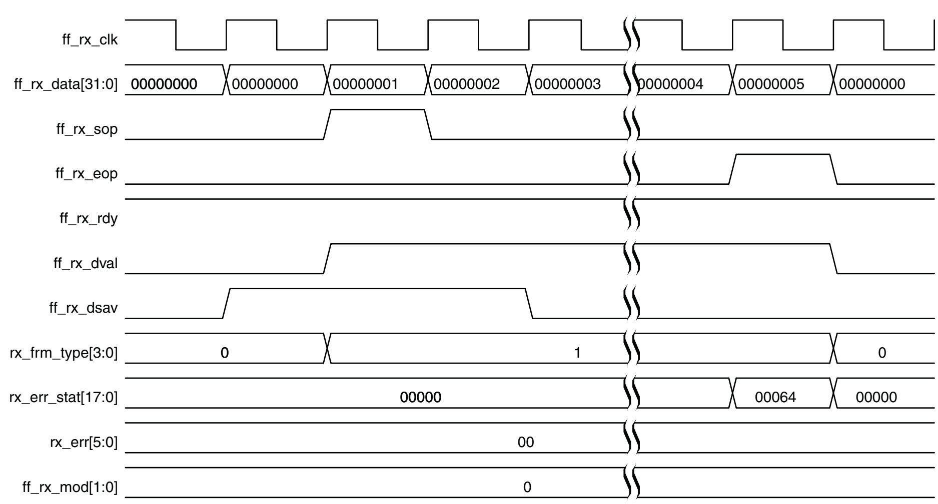


Figure 7–11 shows the receive operation for MAC core variations without internal FIFO buffers.

**Figure 7–11.** Receive Operation—MAC Without Internal FIFO Buffers

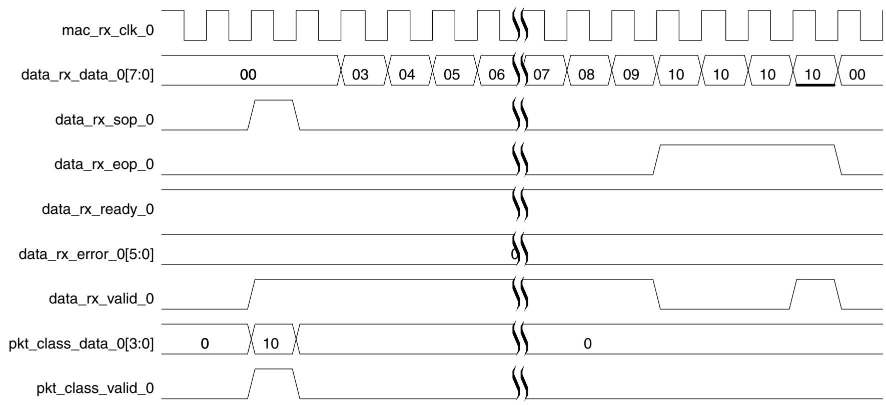



Figure 7–12 depicts an invalid length error during a receive operation for MAC core variations with internal FIFO buffers.

**Figure 7–12.** Invalid Length Error—MAC With Internal FIFO Buffer

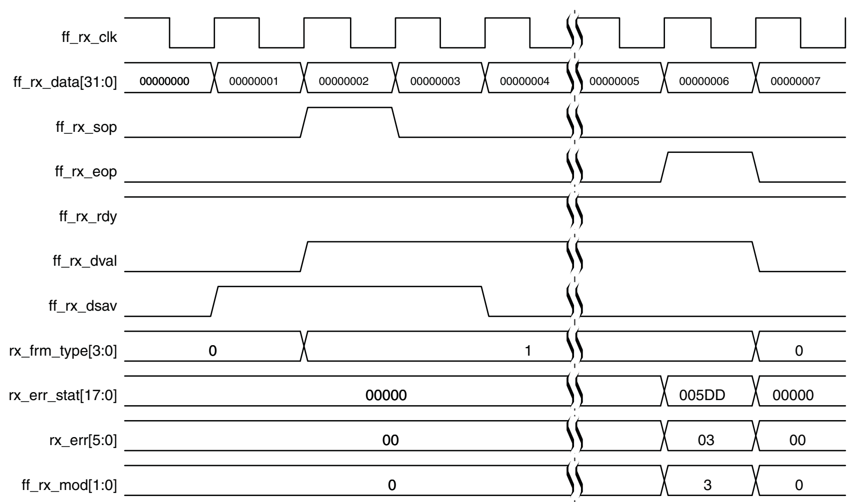
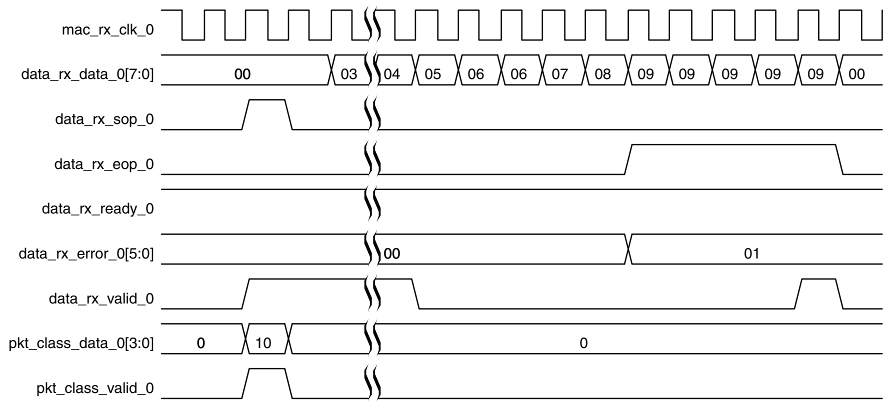


Figure 7–13 depicts an invalid length error during a receive operation for MAC core variations without internal FIFO buffers.

**Figure 7–13.** Invalid Length Error—MAC Without Internal FIFO Buffers




## Avalon-ST Transmit Interface

Figure 7–14 shows the transmit operation for MAC core variations with internal FIFO buffers.

**Figure 7–14.** Transmit Operation—MAC With Internal FIFO Buffers

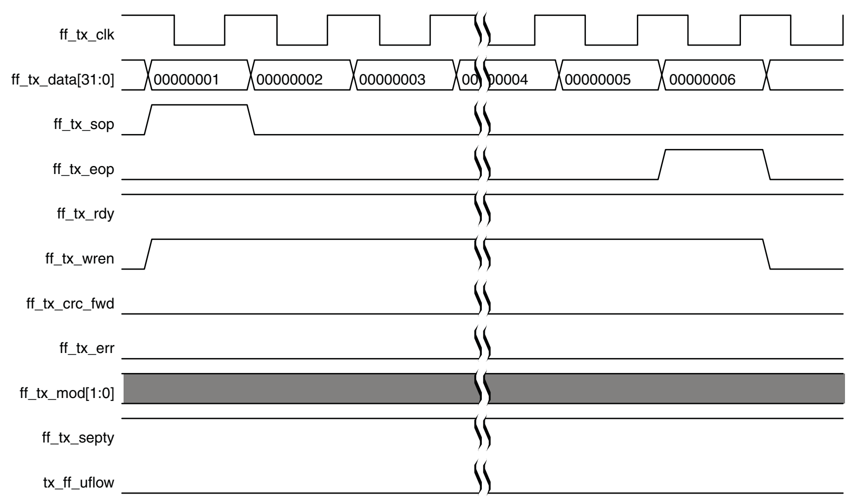
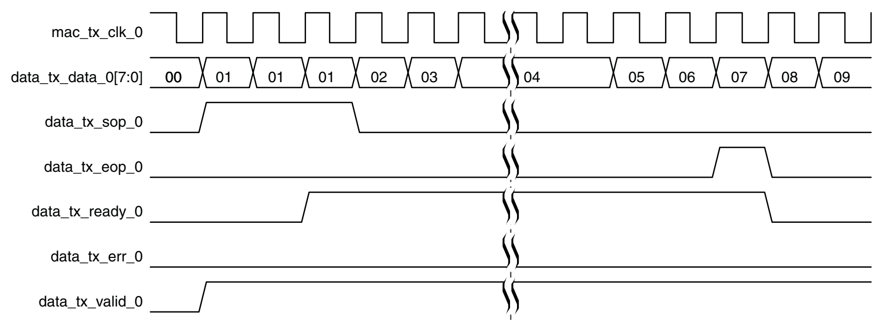


Figure 7–15 shows the transmit operation for MAC core variations without internal FIFO buffers.

**Figure 7–15.** Transmit Operation—MAC Without Internal FIFO Buffers




## GMII Transmit

On transmit, all data transfers are synchronous to the rising edge of `tx_clk`. The GMII data enable signal `gm_tx_en` is asserted to indicate the start of a new frame and remains asserted until the last byte of the frame is present on `gm_tx_d[7:0]` bus. Between frames, `gm_tx_en` remains deasserted.

If a frame is received on the Avalon-ST interface with an error (asserted with `ff_tx_eop`), the frame is subsequently transmitted with the GMII `gm_tx_err` error signal at any time during the frame transfer.

## GMII Receive

On receive, all signals are sampled on the rising edge of `rx_clk`. The GMII data enable signal `gm_rx_dv` is asserted by the PHY to indicate the start of a new frame and remains asserted until the last byte of the frame is present on the `gm_rx_d[7:0]` bus. Between frames, `gm_rx_dv` remains deasserted.

If the PHY detects an error on the frame received from the line, the PHY asserts the GMII error signal, `gm_rx_err`, for at least one clock cycle at any time during the frame transfer.

A frame received on the GMII interface with a PHY error indication is subsequently transferred on the Avalon-ST interface with the error signal `rx_err[0]` asserted.

## RGMII Transmit

On transmit, all data transfers are synchronous to both edges of `tx_clk`. The RGMII control signal `tx_control` is asserted to indicate the start of a new frame and remains asserted until the last upper nibble of the frame is present on the `rgmii_out[3:0]` bus. Between frames, `tx_control` remains deasserted. Figure 7–16 and Figure 7–17 show the timing diagrams of RGMII transmit in 10/100 Mbps and gigabit mode respectively.

**Figure 7–16.** RGMII Transmit in 10/100 Mbps

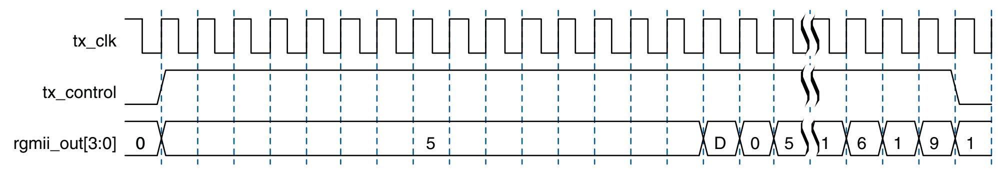
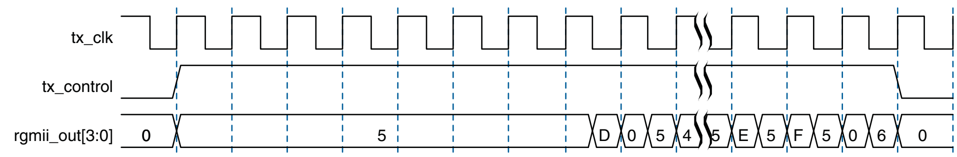
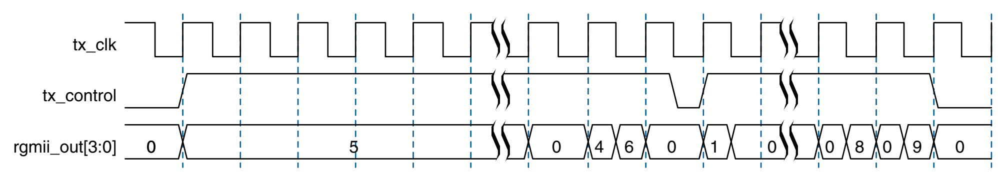



**Figure 7–17.** RGMII Transmit in Gigabit Mode




If a frame is received on the Avalon-ST interface with an error (`ff_tx_err` asserted with `ff_tx_eop`), the frame is subsequently transmitted with the RGMII `tx_control` error signal (at the falling edge of `tx_clk`) at any time during the frame transfer. Figure 7–18 shows the timing diagram of RGMII transmit when an error occurs.

**Figure 7–18.** RGMII Transmit with Error in 1000 Mbps




## RGMII Receive

On receive all signals are sampled on both edges of `rx_clk`. The RGMII control signal `rx_control` is asserted by the PHY to indicate the start of a new frame and remains asserted until the last upper nibble of the frame is present on `rgmii_in[3:0]` bus. Between frames, `rx_control` remains deasserted. Figure 7–19 and Figure 7–20 show the timing diagrams of RGMII receive in 10/100 Mbps and 1000 Mbps respectively.

**Figure 7–19.** RGMII Receive in 10/100 Mbps

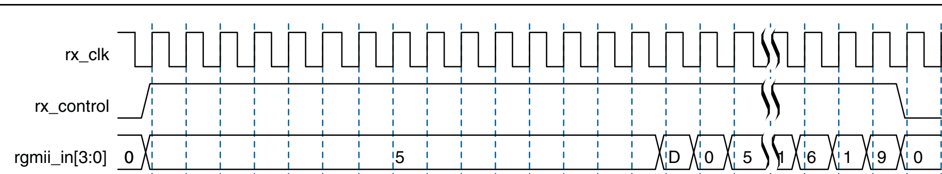
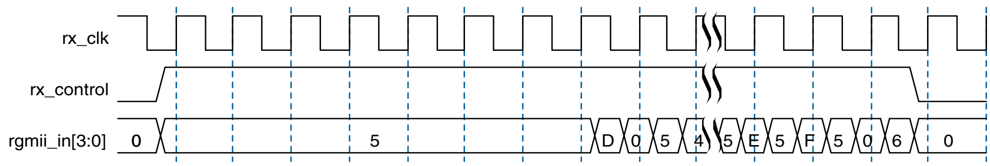
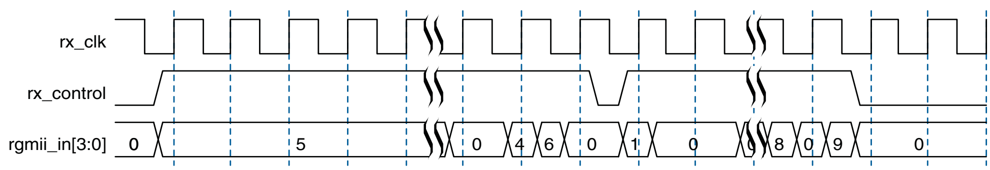


**Figure 7–20.** RGMII Receive in 1000 Mbps




A frame received on the RGMII interface with a PHY error indication is subsequently transferred on the Avalon-ST interface with the error signal `rx_err[0]` asserted. Figure 7–21 shows the timing diagram of RGMII receive when an error occurs.

**Figure 7–21.** RGMII Receive with Error in Gigabit Mode




The current implementation of the RGMII receive interface expects a positive-delay `rx_clk` relative to the receive data (the clock comes after the data).

## MII Transmit

On transmit, all data transfers are synchronous to the rising edge of `tx_clk`. The MII data enable signal, `m_tx_en`, is asserted to indicate the start of a new frame and remains asserted until the last byte of the frame is present on `m_tx_d[3:0]` bus. Between frames, `m_tx_en` remains deasserted.

If a frame is received on the FIFO interface with an error (`ff_tx_err` asserted) the frame is subsequently transmitted with the MII error signal `m_tx_err` for one clock cycle at any time during the frame transfer.

## MII Receive

On receive, all signals are sampled on the rising edge of `rx_clk`. The MII data enable signal `m_rx_en` is asserted by the PHY to indicate the start of a new frame and remains asserted until the last byte of the frame is present on `m_rx_d[3:0]` bus. Between frames, `m_rx_en` remains deasserted.

If the PHY detects an error on the frame received from the line, the PHY asserts the MII error signal, `m_rx_err`, for at least one clock cycle at any time during the frame transfer.

A frame received on the MII interface with a PHY error indication is subsequently transferred on the FIFO interface with the error signal `rx_err[0]` asserted.

## IEEE 1588v2 Timestamp

The following timing diagrams show the timestamp of frames observed on TX path for the IEEE 1588v2 feature.

Figure 7–22 shows the TX timestamp signals for the IEEE 1588v2 feature in a one step operation.

**Figure 7–22.** Egress Timestamp Insert for IEEE 1588v2 PTP Packet Encapsulated in IEEE 802.3

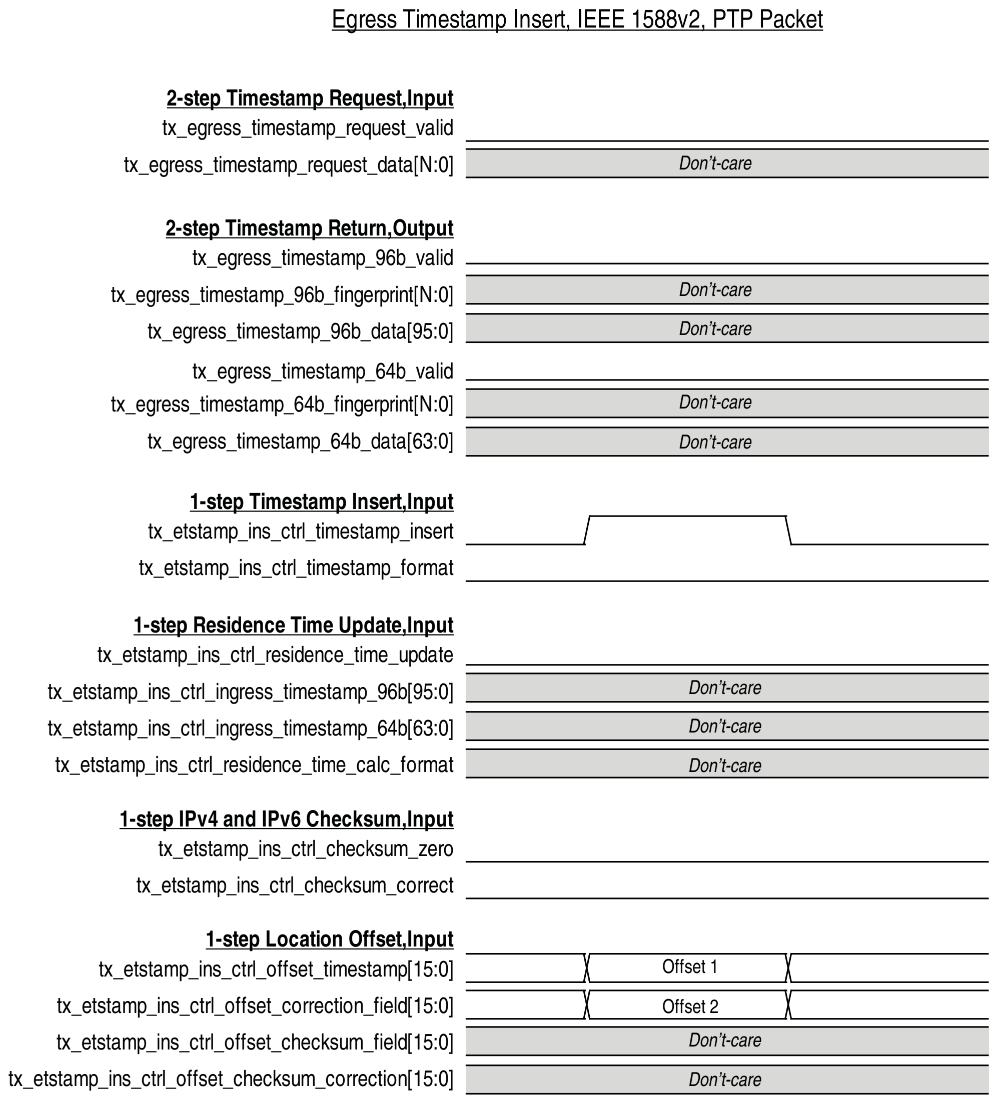


Figure 7–23 shows the TX timestamp signals for the first type of egress correction field update, where the residence time is calculated by subtracting 96 bit ingress timestamp from 96 bit egress timestamp. The result is updated in the correction field of the PTP frame encapsulated over UDP/IPv4.

**Figure 7–23.** Type 1 Egress Correction Field Update

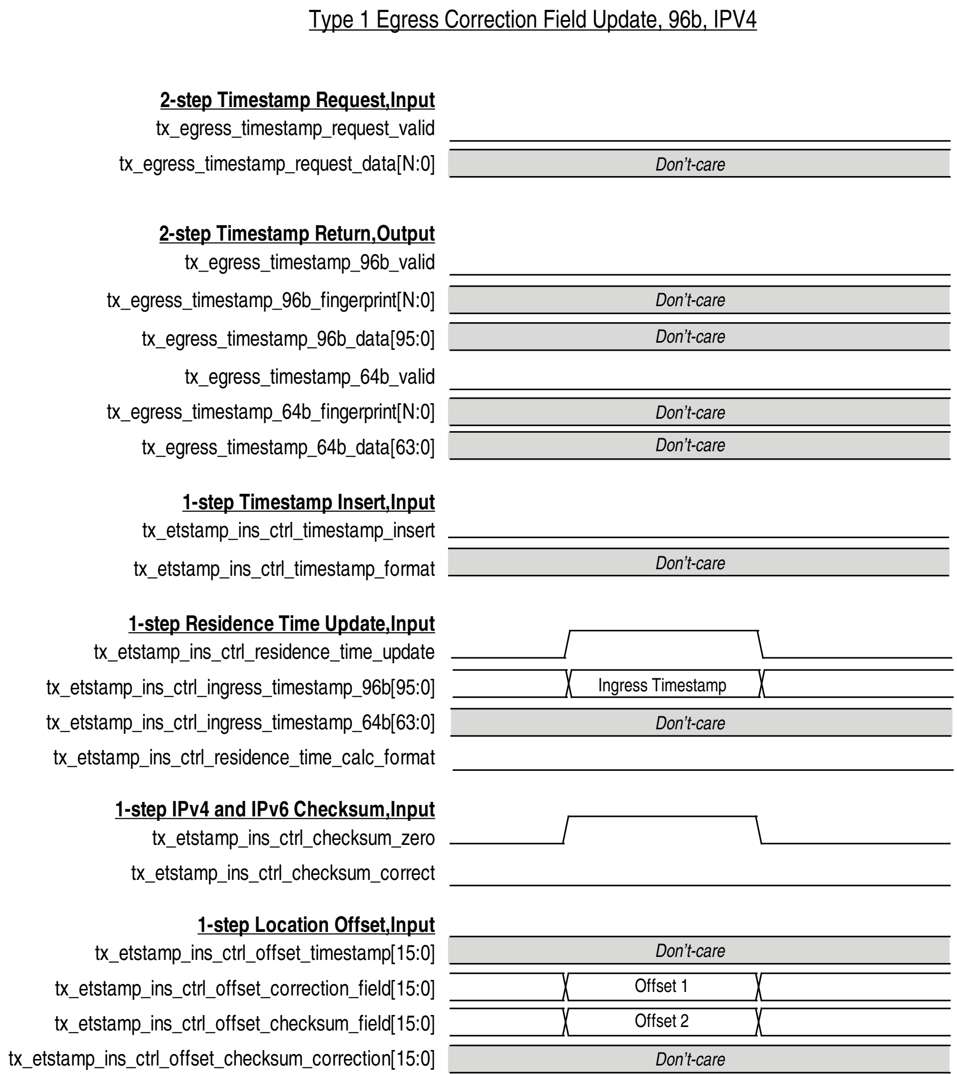


Figure 7–24 shows the TX timestamp signals for the second type of egress correction field update, where the 64 bit ingress timestamp has been pre-subtracted from the correction field at the ingress port. At the egress port, the 64 bit egress timestamp is added into the correction field and the correct residence time is updated in the correction field. This is the example of PTP frame encapsulated over UPD/IPV6.

**Figure 7–24.** Type 2 Egress Correction Field Update

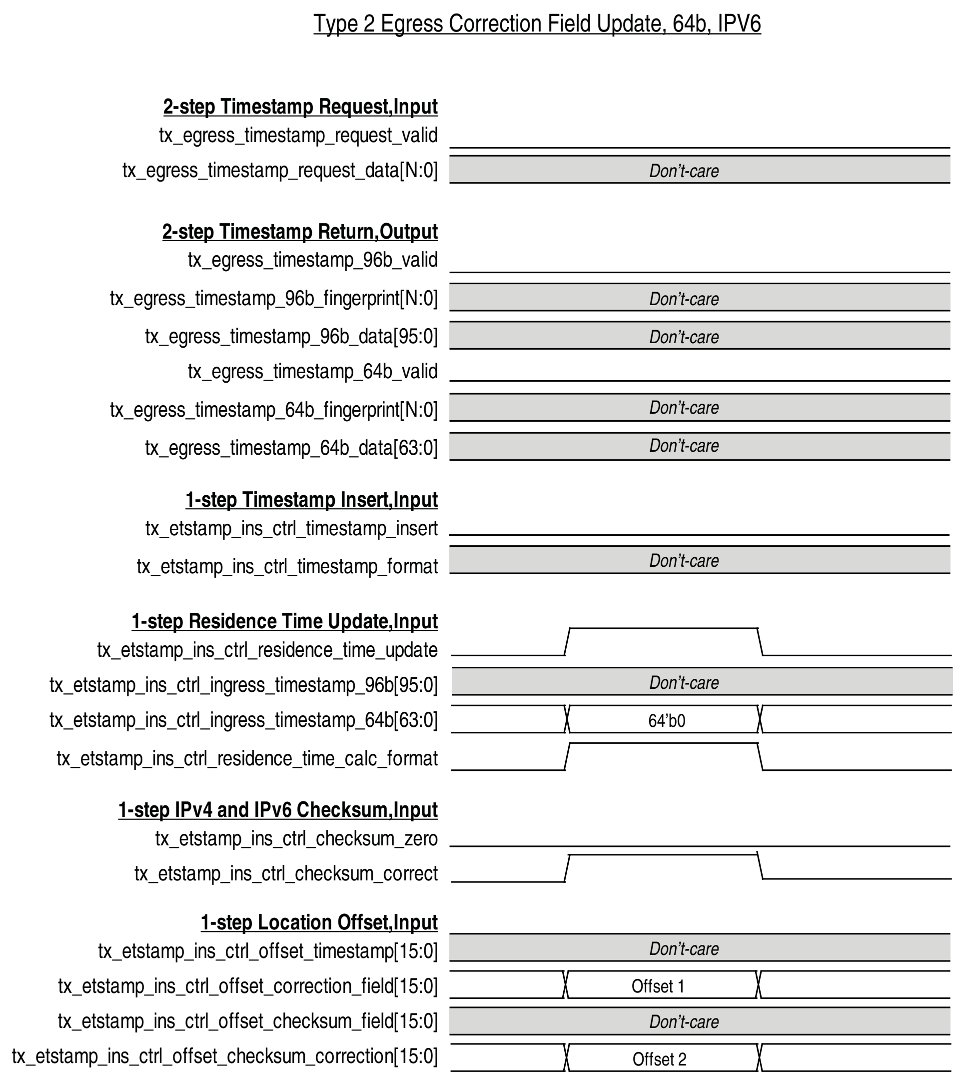
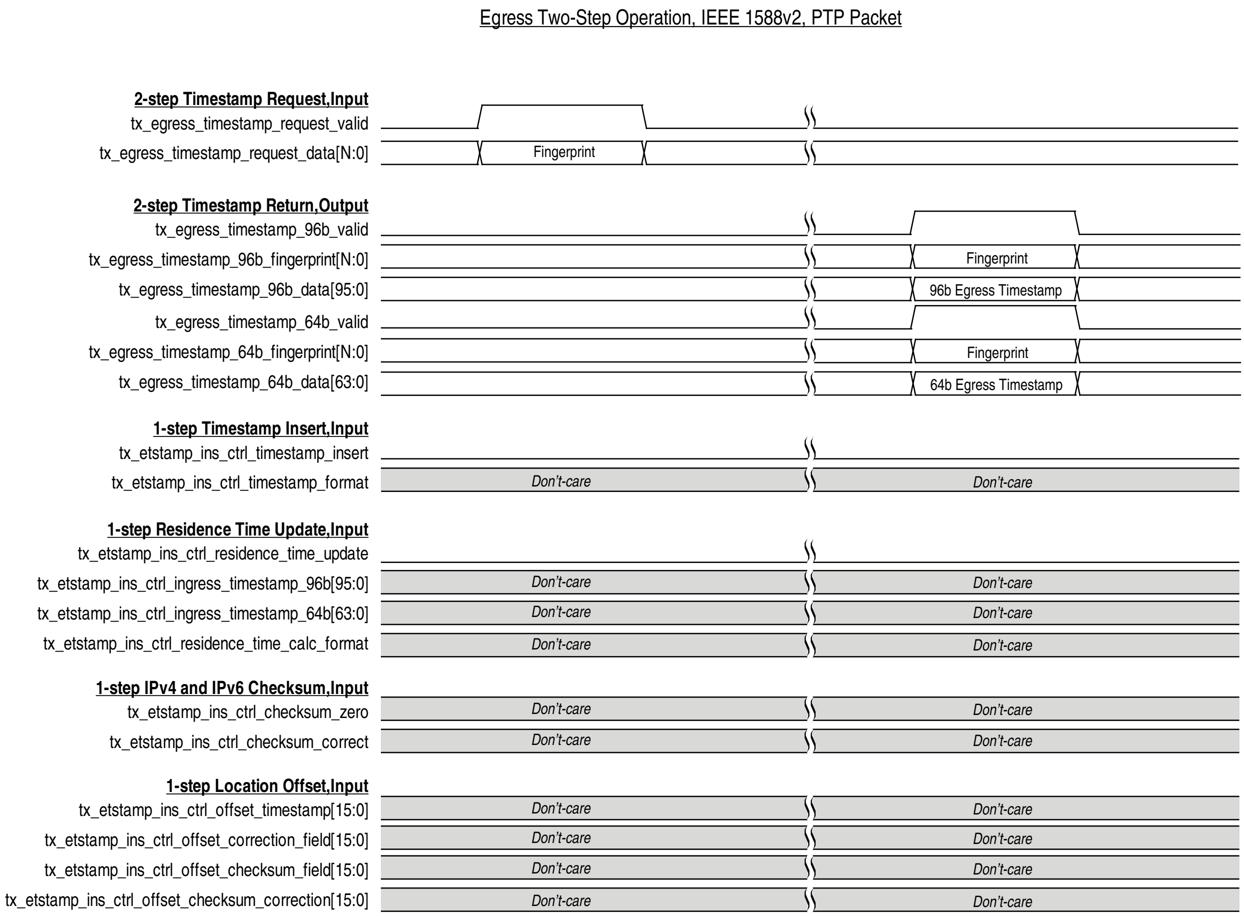


Figure 7–24 shows the TX timestamp signals for the IEEE 1588v2 feature in a two step operation.

**Figure 7–25.** Egress Two-Step Operation



## Optimizing Clock Resources in Multiport MAC with PCS and Embedded PMA

The following factors determine the total number of global and regional clock resources required by your system:

- Configuration of the Triple-Speed Ethernet MegaCore function and the blocks it contains
- PCS operating mode (SGMII or 1000BASE-X)
- PMA technology implemented in the target device
- Number of clocks that can share a single source
- Number of PMAs required in the design
- ALTGX megafunction operating mode

You can use the same clock source to drive clocks that are visible at the top-level design, thus reducing the total number of clock sources required by the entire design. Table 8–1 lists the clock and reset signals that are visible at the top-level design for each possible configuration.

**Table 8–1.** Clock Signals Visible at Top-Level Design

| Clocks | Configurations (1) | | |
|---|---|---|---|
| | MAC Only | MAC and PCS | MAC and PCS with PMA |
| rx_recovclkout | — | — | Yes |
| ref_clk | — | — | Yes |
| clk | Yes | Yes | Yes |
| ff_tx_clk | Yes | Yes | Yes |
| ff_rx_clk | Yes | Yes | Yes |
| tx_clk | Yes | No | No |
| rx_clk | Yes | No | No |
| tbi_rx_clk | — | Yes | No |
| tbi_tx_clk | — | Yes | No |
| gxb_cal_blk_clk (2) | — | — | Yes |
| reconfig_clk | — | — | Yes |

**Notes to Table 8–1:**

(1) Yes indicates that the clock is visible at the top-level design.
No indicates that the clock is not visible at the top-level design.
— indicates that the clock is not applicable for the given configuration.

(2) Applies to GX transceiver.

## MAC and PCS With GX Transceivers

In configurations that contain the MAC, PCS, and GX transceivers, you have the following options in optimizing clock resources:

- Utilize the same reset signal for all MAC instances if you do not require a separate reset for each instance.
- Utilize the same reference clock for all PMA quads
- Utilize the same clock source to drive the reference clock, FIFO transmit and receive clocks, and system clocks, if these clocks run at the same frequency.

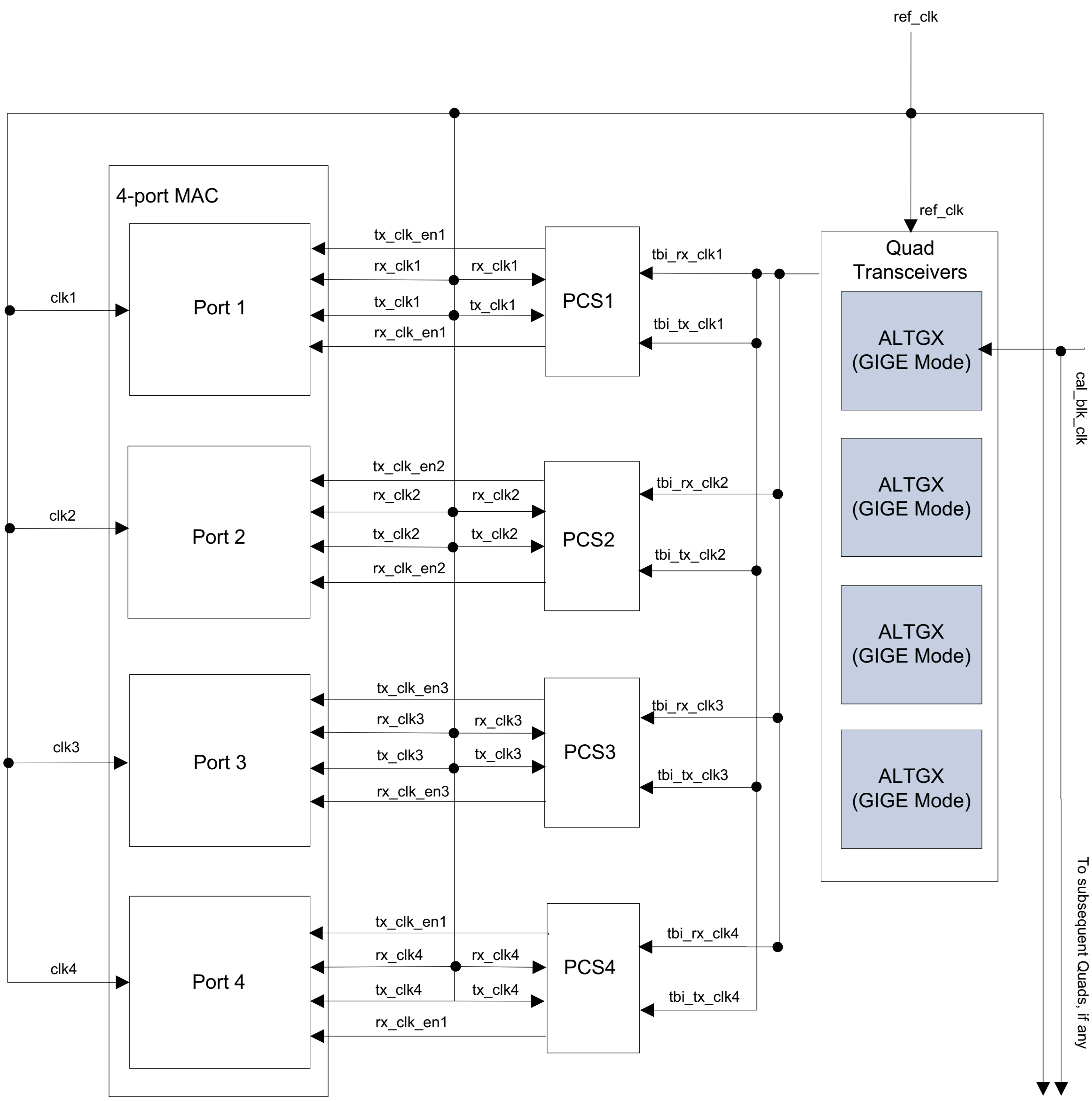
The Quartus II software automatically optimizes the TBI transmit clocks. Only one clock source drives the TBI transmit clocks from each PMA quad.

The calibration clock (`gxb_cal_blk_clk`) calibrates the termination resistors in all transceiver channels in a device. As there is only one calibration circuit in each device, one clock source suffices.

☞ If you do not constrain the PLL inputs and outputs in your design, add `derive_pll_clocks` in the timing constraint file to ensure that the TimeQuest timing analyzer automatically creates derived clocks for the PLL outputs.

Figure 8–1 shows the optimal clock distribution scheme you can achieve in configurations that contain the 10/100/1000 Ethernet MAC, SGMII PCS, and GX transceivers.

**Figure 8–1.** Clock Distribution in MAC and SGMII PCS with GXB Configuration—Optimal Case



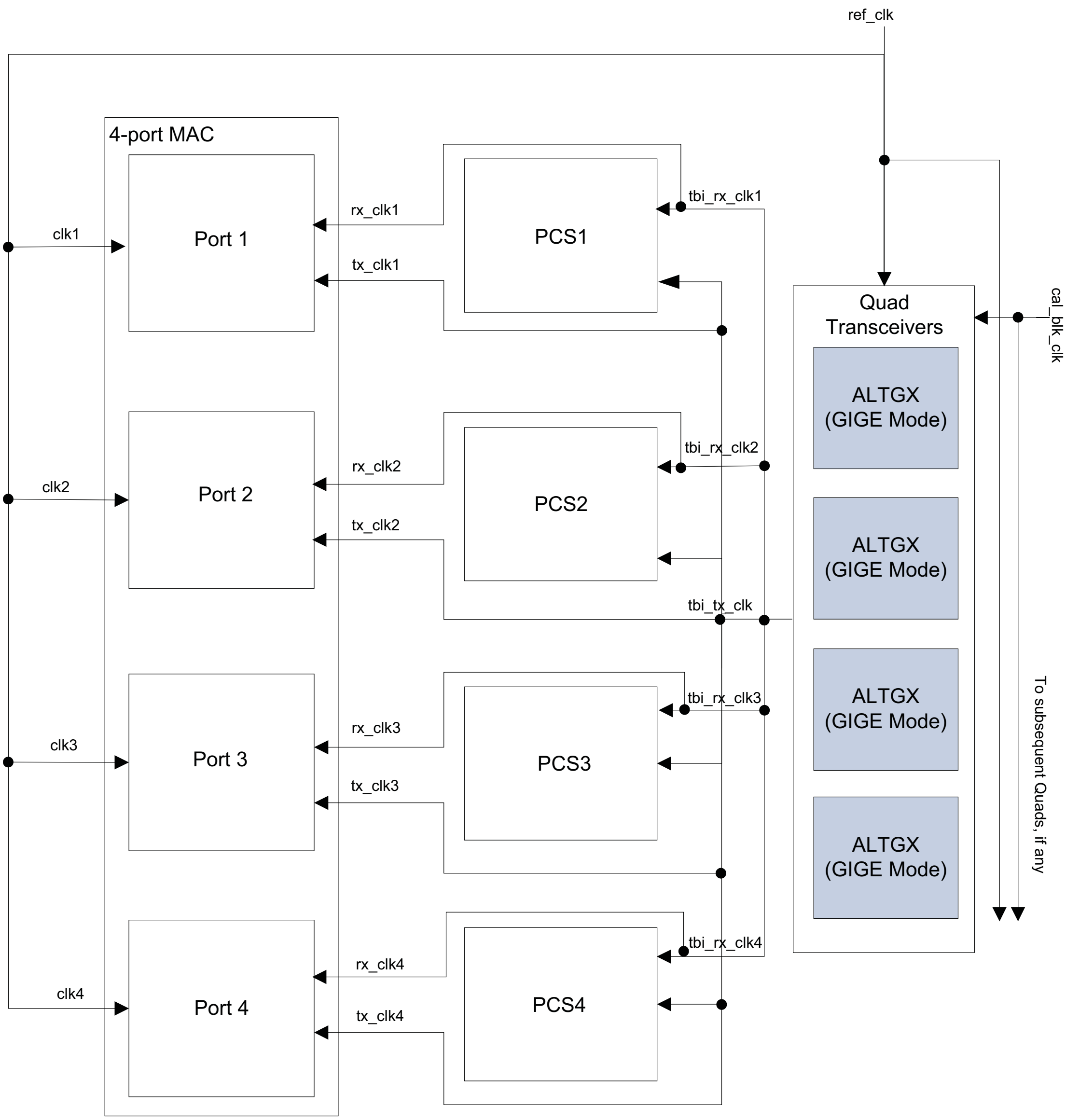

**Note to Figure 8–1:**

(1) The PMA layer in devices with GX transceivers uses ALTGX megafunctions.

Figure 8–2shows the optimal clock distribution scheme you can achieve in configurations that contain the 10/100/1000 Ethernet MAC, 1000Base-X PCS, and GX transceivers.

In addition to the aforementioned optimization options, the TBI transmit and receive clocks can be used to drive the MAC transmit and receive clocks, respectively.

**Figure 8–2.** Clock Distribution in MAC and 1000BASE-X PCS with GXB Configuration—Optimal Case




**Note to Figure 8–2:**

(1) The PMA layer in devices with GX transceivers uses ALTGX megafunctions.

## MAC and PCS With LVDS Soft-CDR I/O

In configurations that contain the MAC, PCS, and LVDS Soft-CDR I/O, you have the following options in optimizing clock resources:

- Utilize the same reset signal for all MAC instances if you do not require a separate reset for each instance.
- Utilize the same clock source to drive the reference clock, FIFO transmit and receive clocks, and system clocks, if these clocks run at the same frequency.

Figure 8–3 shows the optimal clock distribution scheme you can achieve in configurations that contain the MAC, SGMII PCS and LVDS Soft-CDR I/O.

**Figure 8–3.** Clock Distribution in MAC and SGMII PCS with LVDS Configuration—Optimal Case

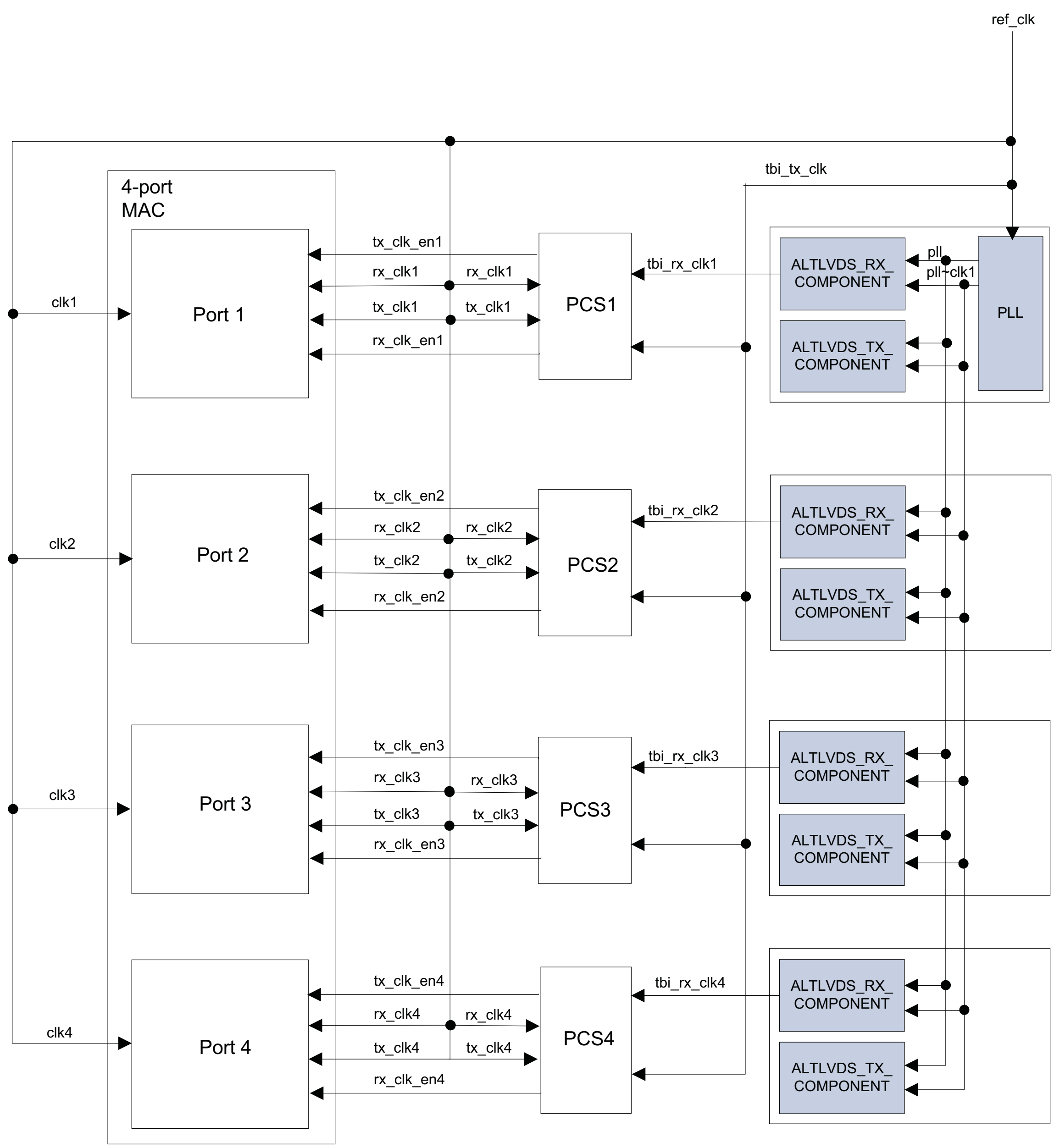
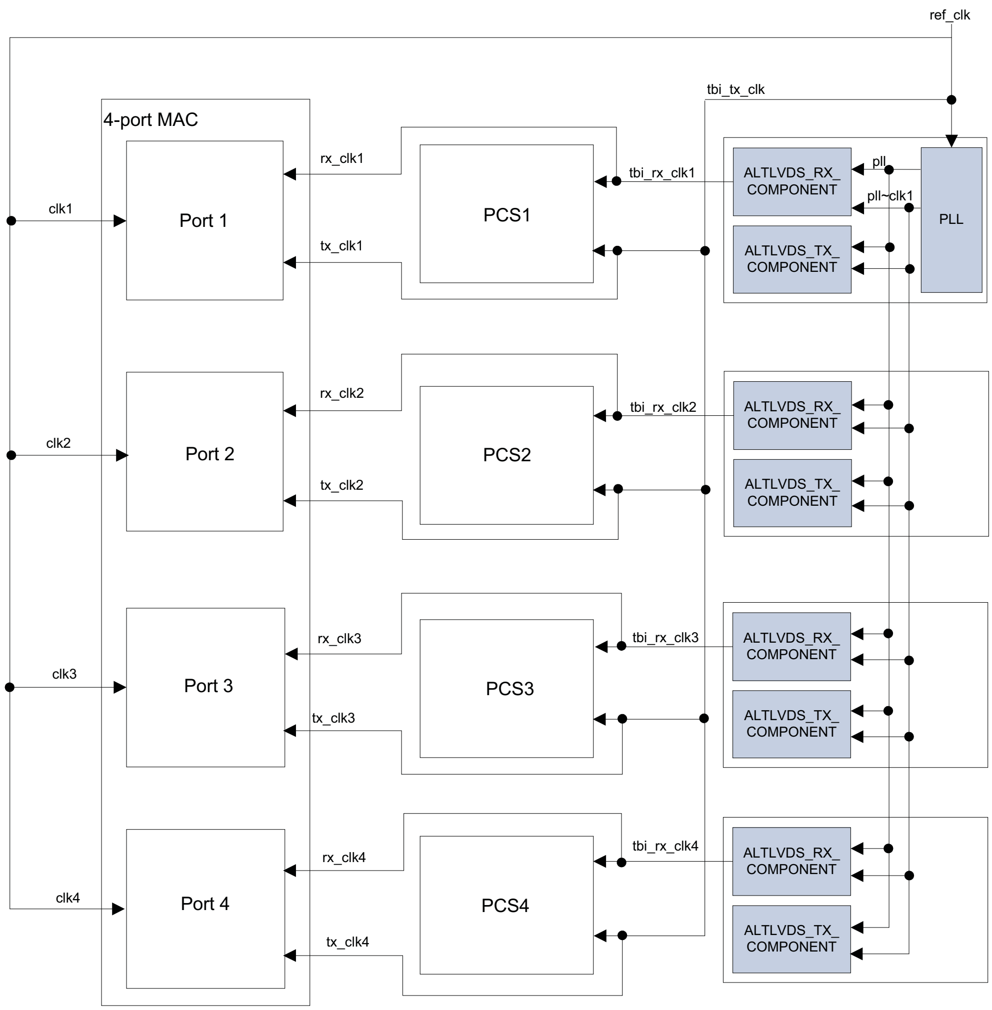


Figure 8–4 shows the optimal clock distribution scheme you can achieve in configurations that contain the MAC, 1000BASE-X PCS, and LVDS Soft-CDR I/O.

**Figure 8–4.** Clock Distribution in MAC and 1000BASE-X PCS with LVDS Configuration—Optimal Case



# Sharing PLLs in Devices with LVDS Soft-CDR I/O

For designs that contain multiple instances of MAC and PCS with PMA or PCS with PMA variation targeting devices with LVDS soft-CDR I/O, you can optimize resource utilization by sharing the PLLs.

The Quartus II software merges the PLLs for these instances if you implement the following items in your design:

- Connect the reference clock of each instance to the same source.
- Place the LVDS I/O pins on the same side of the FPGA.

# Sharing PLLs in Devices with GIGE PHY

For Cyclone V designs that contain multiple instances of MAC and PCS with PMA or PCS with PMA variation targeting devices with GIGE PHY, you can share the PLLs by placing the associated signals (`tx_p`, `rx_p`, and `ref_clk`) to the same I/O block of transceiver bank through pin assignment. Additionally, the `rx_recovclkout` clock must be buffered by two levels of inverter in the top level module so that it can be fitted to the general I/O pins.

# Sharing Transceiver Quads

For designs that contain multiple PMA blocks targeting Altera device families with GX transceivers, you can combine the transceiver channels in the same quad. To share the same transceiver quad, the transceiver channels must have the same dynamic reconfiguration setting. In other words, you must turn on dynamic reconfiguration capabilities in all channels in a quad even though you only intend to use these capabilities in some of the channels.

The dynamic reconfiguration is always turned on in devices other than Arria GX and Stratix II GX. When the dynamic reconfiguration is turned on in designs targeting devices other than Stratix V, Altera recommends that you connect the dynamic reconfiguration signals to the ALTGX_RECONFIG megafunction.

In Stratix V devices, Altera recommends that you connect the dynamic reconfiguration signals to the Transceiver Reconfiguration Controller megafunction. For transceiver quad sharing between Triple-Speed Ethernet IP core and other IP cores that target Stratix V devices, reset signal for all the cores must be from the same source.

☞ Refer to the respective device handbook for more information on dynamic reconfiguration signals in Altera devices.

# Migrating From Old to New User Interface For Existing Designs

In Quartus II software ACDS 13.0 release, the old Triple-Speed Ethernet MegaCore function user interface is deprecated. Existing Triple-Speed Ethernet designs generated prior to the ACDS 13.0 release can still load properly in ACDS 13.0. However, starting from ACDS 13.1 release, the old Triple-Speed Ethernet interface and the design generated using the old interface will not be supported.

☞ You need to manually migrate your design to the new user interface. Reopening and saving the existing design created with the old user interface will not automatically convert the design to the new user interface.

To migrate your design to the new user interface, launch the Quartus II software ACDS 13.0 or higher, create a new project, and specify the parameters as described in "Design Flows" on page 2–2.

# 9. Timing Constraints

Altera provides timing constraint files (**.sdc**) to ensure that the Triple-Speed Ethernet MegaCore function meets the design timing requirements in Altera devices. The files constraints the false paths and multi-cycle paths in the Triple-Speed Ethernet Megacore function. The timing constraints files are specified in the *<variation_name>***.qip** file and is automatically included in the Quartus II project files.

You may need to add timing constraints that are external to the MegaCore function. The following sections describe the procedure to create the timing constraint file.

## Creating Clock Constraints

After you generate and integrate the Triple-Speed Ethernet MegaCore function into the system, you need to create a timing constraints file to specify the clock constraint requirement.

You can specify the clock requirement in the timing constraint file using the following command:

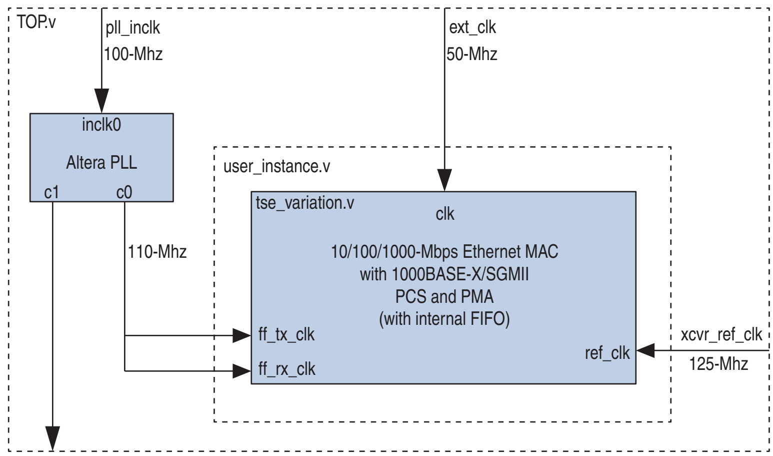
```
create_clock
     -period <period value>
     [-name <clock name>]
     [-waveform <edge list>]
     [-add]
     <source objects>
```

For example, for a new clock named "`reg_clk`", with a 50 MHz clock targeted to the top level input port "`clk`", enter the following command line:

```
create_clock -name "reg_clk" -period "50 MHz" [get_ports "clk"]
```

Figure 9–1 shows an example of how you can create a timing constraint file to constrain the Triple-Speed Ethernet MegaCore function clocks.

**Figure 9–1.** Triple-Speed Ethernet Timing Constraint Example




The example in Figure 9–1 consists of the following Verilog modules:

- **TOP.v**—The top level design module which contains an Altera PLL and a user-defined instance. The top level input clocks consist of `pll_inclk`, `ext_clk`, and `xcvr_ref_clk`.
- **user_instance.v**—The user-defined instance that instantiates the Triple-Speed Ethernet MegaCore function.
- **tse_variation.v**—A Triple-Speed Ethernet MegaCore function variation. This example uses a 10/100/1000-Mbps Ethernet MAC with an internal FIFO buffer, a 1000BASE-X/SGMII PCS, and an embedded PMA.

The frequency for the PLL clock input, `inclk0`, is 100 MHz, and the frequency for the PLL clock output, `c0`, is 110 MHz. The Triple-Speed Ethernet MAC Avalon-ST clocks, `ff_tx_clk` and `ff_rx_clk`, use `c0` as the clock source. The input clock frequency for the transceiver reference clock, `xcvr_ref_clk`, is 125 MHz.

Example 9–1 shows the Triple-Speed Ethernet MegaCore function timing constraint file.

**Example 9–1.** Triple-Speed Ethernet MegaCore Function Timing Constraints File

```
# PLL clock input, 100 MHz
create_clock -name pll_inclk -period 10.000 [get_ports {pll_inclk}]

# ext_clk, 50 MHz
create_clock -name ext_clk -period 20.000 [get_ports {pll_ext_clk}]

# xcvr_ref_clk, 125 MHz
create_clock -name xcvr_ref_clk -period 8.000 [get_ports {xcvr_ref_clk}]

# Derive PLL generated output clocks.
derive_pll_clocks
```

# Recommended Clock Frequency

Table 9–1 lists the recommended clock input frequency for each Triple-Speed Ethernet MegaCore function variant.

**Table 9–1.** Recommended Clock Input Frequency For Each MegaCore Function Variant (Part 1 of 2)

| MegaCore Function Variant | Clock | Recommended Frequency (MHz) |
|---|---|---|
| 10/100/1000-Mbps Ethernet MAC (with Internal FIFO buffers) | CLK | 50–100 |
| | TX_CLK | 125 |
| | RX_CLK | 125 |
| | FF_TX_CLK | 100 |
| | FF_RX_CLK | 100 |
| 10/100/1000-Mbps Ethernet MAC (without Internal FIFO buffers) | CLK | 50–100 |
| | TX_CLK <N> | 125 |
| | RX_CLK <N> | 125 |
| | RX_AFULL_CLK | 100 |
| 10/100/1000-Mbps Ethernet MAC with 1000BASE-X/SGMII PCS (with Internal FIFO buffers) | CLK | 50–100 |
| | FF_TX_CLK | 100 |
| | FF_RX_CLK | 100 |
| | TBI_TX_CLK | 125 |
| | TBI_RX_CLK | 125 |
| | REF_CLK | 125 |
| | RECONFIG_CLK *(1)* | 37.5–50 |
| | GXB_CAL_BLK_CLK | 125 |

**Table 9–1.** Recommended Clock Input Frequency For Each MegaCore Function Variant (Part 2 of 2)

| MegaCore Function Variant | Clock | Recommended Frequency (MHz) |
|---|---|---|
| 10/100/1000-Mbps Ethernet MAC with 1000BASE-X/SGMII PCS (without Internal FIFO buffers) | CLK | 50–100 |
| | RX_AFULL_CLK | 100 |
| | TBI_TX_CLK <N> | 125 |
| | TBI_RX_CLK <N> | 125 |
| | REF_CLK | 125 |
| | RECONFIG_CLK <N> *(1)* | 37.5–50 |
| | GXB_CAL_BLK_CLK | 125 |
| 1000BASE-X/SGMII PCS only | CLK | 50–100 |
| | REF_CLK | 125 |
| | TBI_TX_CLK | 125 |
| | TBI_RX_CLK | 125 |

**Note to Table 9–1:**

(1) This signal is only applicable to all device family prior to the 28-nm devices, which consists of the Stratix V, Arria V, Arria V GZ, and Cyclone V devices.

# 10. Testbench

You can use the testbench provided with the Triple-Speed Ethernet MegaCore function to exercise your custom MegaCore function variation. The testbench includes the following features:

- Easy-to-use simulation environment for any standard HDL simulator.
- Simulation of all basic Ethernet packet transactions.
- Open source Verilog HDL and VHDL testbench files.

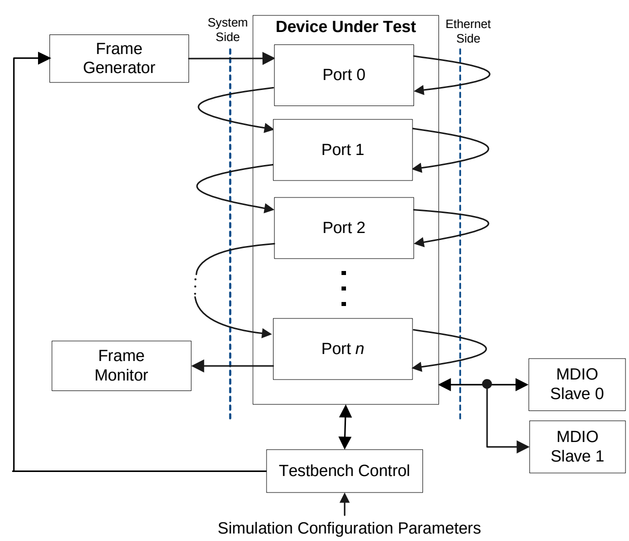
The provided testbench applies only to custom MegaCore function variations created using the MegaWizard Plug-in Manager flow.

For more information about the generated testbench files, refer to "Generated Files" on page 2–7.

## Architecture

Figure 10–1 illustrates the testbench architecture for the Triple-Speed Ethernet MegaCore function.

**Figure 10–1.** Triple-Speed Ethernet Testbench Architecture




## Components

The testbench comprises the following modules:

- Device under test (DUT)—Your custom MegaCore function variation
- Avalon-ST Ethernet frame generator—Simulates a user application connected to the MAC system-side interface. It generates frames on the Avalon-ST transmit interface.

- Avalon-ST Ethernet frame monitor—Simulates a user application receiving frames from the MAC system-side interface. It monitors the Avalon-ST receive interface and decodes all data received.
- MII/RGMII/GMII Ethernet frame generator—Simulates a MAC function that sends frames to the PCS function.
- MII/RGMII/GMII Ethernet frame monitor—Simulates a MAC function that receives frames from the PCS function and decodes them.
- MDIO slaves—Simulates a PHY management interface. It responds to an MDIO master transactor.
- Clock and reset generator.

Table 10–1 lists the interfaces, frame generator and frame monitor for each configuration.

**Table 10–1.** Testbench Components

| Configuration | System-Side Interface | Ethernet-Side Interface | Frame Generator | Frame Monitor |
|---|---|---|---|---|
| MAC only | Avalon-ST | GMII/MII/RGMII | Avalon-ST Frame Generator | Avalon-ST Frame Monitor |
| MAC with PCS | Avalon-ST | TBI | Avalon-ST Frame Generator | Avalon-ST Frame Monitor |
| MAC with PCS and embedded PMA | Avalon-ST | 1.25 Gbps | Avalon-ST Frame Generator | Avalon-ST Frame Monitor |
| PCS only | GMII/MII | TBI | GMII/MII Frame Generator | GMII/MII Frame Monitor |
| PCS with embedded PMA | GMII/MII | 1.25 Gbps | GMII/MII Frame Generator | GMII/MII Frame Monitor |

# Verification

The testbench is self-checking and determines the success of a simulation by verifying the frames received. It also checks for any errors detected by the frame monitors. The testbench does not verify the IEEE statistics generated by the MAC layer. Simulation fails only if the testbench is not able to detect deliberately inserted errors. At the end of a simulation, the testbench displays messages in the simulator console indicating its results.

The testbench verifies the following functionality:

- Transmit and receive datapaths are functionally correct.
- Ethernet frames generated by the frame generator are received by the frame monitor.
- Additional checks for configurations that contain the MAC function:
  - Correct CRC-32 is inserted.
  - Short frames are padded up to at least 64 bytes in length.
  - Untagged received frames of size greater than the maximum frame length are truncated to the maximum frame length with additional bytes up to 12.
  - CRC-32 is optionally discarded before the frames are received by the traffic monitor.

- Additional checks for configurations that contain the PCS function with optional embedded PMA:
  - Transmit frames generated by the frame generator are correctly encapsulated.
  - Received frames are de-encapsulated before they are forwarded to the frame monitor.

# Configuration

The testbench is configured, by default, to operate in loopback mode. Frames sent through the transmit path are looped back into the receive path.

Separate data paths can be configured for single-channel MAC with internal FIFO buffers. In this configuration, the MII/GMII Ethernet frame generator is enabled and the testbench control block simulates independent yet complete receive and transmit datapaths.

You can also customize other aspects of the testbench using the testbench simulation parameters. For more information on the testbench simulation parameters, refer to Appendix C, Simulation Parameters.

The device under test is configured with the following default settings:

- Link speed is set to Gigabit except for configurations that contain Small MAC. For Small MACs, the default speed is 100 Mbps.
- Five Ethernet frames of payload length 100, 101, 102, 103 and 104 bytes are transmitted to the system-side interface and looped back on the ethernet-side interface.
- Default settings for the MAC function:
  - The `command_config` register is set to 0x0408003B.
  - Promiscuous mode is enabled.
  - The maximum frame length, register `frm_length,` is configured to 1518.
  - For a single-channel MAC with internal FIFO buffers, the transmit FIFO buffer is set to start data transmission as soon as its level reaches `tx_section_full`. The receive FIFO buffer is set to begin forwarding Ethernet frames to the Avalon-ST receive interface when its level reaches `rx_section_full`.
- Default setting for the PCS function:
  - The `if_mode` register is set to 0x0000.
  - Auto-negotiation between the local PHY and remote link PHY is bypassed.

# Test Flow

The testbench performs the following operations upon a simulated power-on reset:

- Initializes the DUT registers.
- Starts transmission. For a single-channel MAC with internal FIFO buffers, clears the FIFOs.
- Ends transmission and checks the following elements to determine that the simulation is successful:
  - No Ethernet protocol errors detected.
  - Ethernet frames generated and transmitted are received by the frame monitor.

# Simulation Model

This section describes the step-by-step instructions for generating the simulation model and simulating your design using the ModelSim simulator or other simulators.

## Generate the Simulation Model

The generated design example includes both Verilog HDL and VHDL testbench files for the device under test (DUT)—your custom MegaCore function variation.

To generate a Verilog functional simulation model, you can use the command prompt and run the **quartus_sh -t generate_sim_verilog.tcl** file. Alternatively, perform the following steps:

1. Launch the Quartus II software and browse to the <*variation name*>**_testbench** directory.
2. Open the **generate_sim.qpf** file from the project directory.
3. On the **Tools** menu, select **Tcl Scripts...** and select the **generate_sim_verilog.tcl** file.
4. Click **Run**.

To generate a VHDL functional simulation model, you can use the command prompt and run the **quartus_sh -t generate_sim_vhdl.tcl** file. Alternatively, perform the following steps:

1. Launch the Quartus II software and browse to the <*variation name*>**_testbench** directory.
2. Open the **generate_sim.qpf** file from the project directory.
3. On the **Tools** menu, select **Tcl Scripts...** and browse to the **generate_sim_vhdl.tcl** file.
4. Click **Run**.

## Simulate the IP Core

You can simulate your IP core variation with the functional simulation model and the testbench or example design generated with your IP core. The functional simulation model and testbench files are generated in a project subdirectory. This directory may also include scripts to compile and run the testbench.

For a complete list of models or libraries required to simulate your IP core, refer to the scripts provided with the testbench in "Simulation Model Files" on page 10–6.

To use the ModelSim® simulation software to simulate the testbench design, follow these steps:

1. For Verilog testbench design:
   a. Browse to the following project directory:
      <*variation name*>**_testbench/testbench_verilog/**<*variation name*>
   b. Run the following command to set up the required libraries, to compile the generated IP Functional simulation model, and to exercise the simulation model with the provided testbench:

      `run_<variation_name>_tb.tcl↵`

2. For VHDL testbench design:
   a. Browse to the following project directory:
      <*variation name*>**_testbench/testbench_vhdl/**<*variation name*>
   b. Run the following command to set up the required libraries, to compile the generated IP Functional simulation model, and to exercise the simulation model with the provided testbench:

      `run_<variation_name>_tb.tcl↵`

For more information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

CAUTION: Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

## Simulation Model Files

Previously, the Triple-Speed Ethernet MegaCore function generates a <*variation_name*>**.vho** or <*variation_name*>**.vo** file for VHDL or Verilog HDL IP functional simulation model.

For the new Triple-Speed Ethernet MegaCore function created in Quartus II ACDS 13.0, the simulation model will be generated using the industrial standard IEEE simulation encryption.

Table 10–2 lists the scripts available for you to compile the simulation model files in a standalone flow.

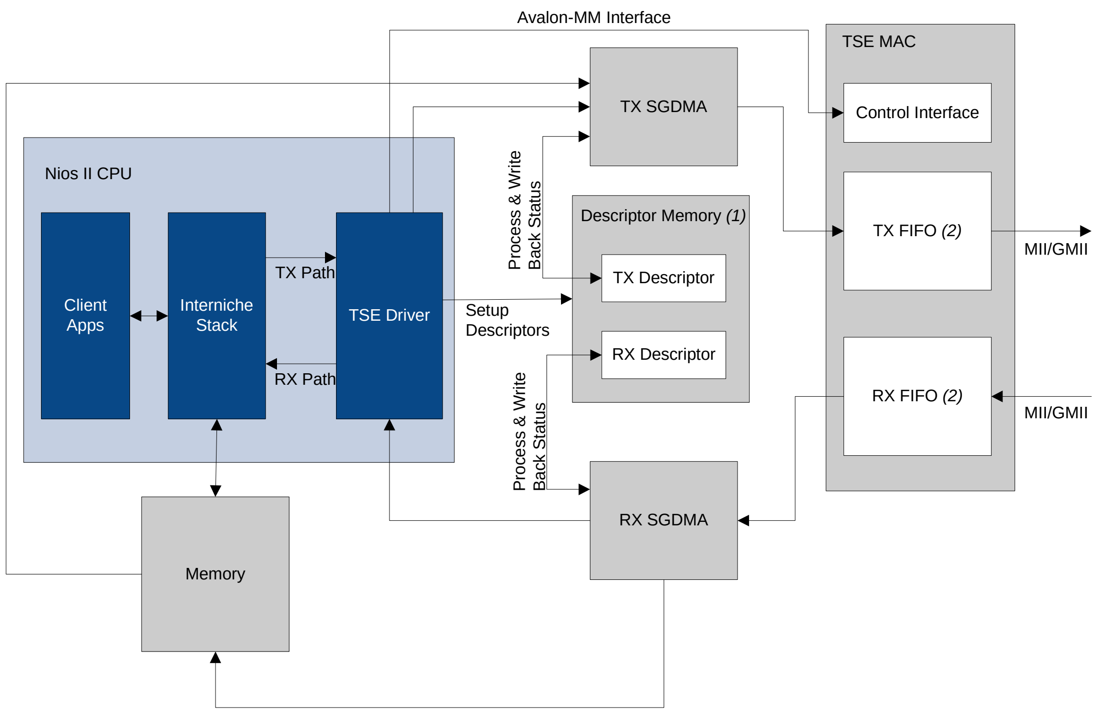
**Table 10–2.** Simulation Model Files

| Directory Name | Description |
|---|---|
| <*variation_name*>**_sim/mentor/** | Contains a ModelSim script **msim_setup.tcl** to set up and run a simulation. |
| <*variation_name*>**_sim/synopsys/vcs** | Contains a shell script **vcs_setup.sh** to set up and run a VCS® simulation. |
| <*variation_name*>**_sim/synopsys/vcsmx** | Contains a shell script **vcsmx_setup.sh** and **synopsys_sim.setup** to set up and run a VCS MX simulation. |
| <*variation_name*>**_sim/mentor/cadence** | Contains a shell script **ncsim_setup.sh** and other setup files to set up and run an NCSIM simulation. |

## Driver Architecture

Figure 11–1 illustrates the architecture of the Triple-Speed Ethernet software driver.

**Figure 11–1.** Triple-Speed Ethernet Software Driver Architecture




**Notes to Figure 11–1:**

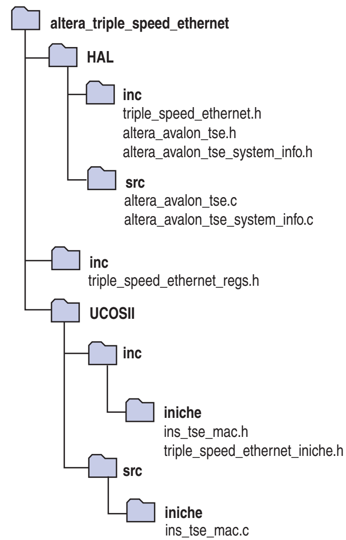
(1) The first *n* bytes are reserved for SGDMA descriptors, where *n* = (Total number of descriptors + 3) × 32. Applications must not use this memory region.

(2) For MAC variations without internal FIFO buffers, the transmit and receive FIFOs are external to the MAC function.

# Directory Structure

Figure 11–2 shows the directory structure of the **altera_triple_speed_ethernet** directory, and the relevant header and source files it contains.

**Figure 11–2.** Directory Structure




# PHY Definition

The software driver only supports the following PHYs by default:

- National DP83848C (10/100 Mbps)
- National DP83865 (10/100/1000 Mbps)
- Marvell 88E1111 (10/100/1000 Mbps)
- Marvell 88E1145 (Quad PHY, 10/100/1000 Mbps).

You can extend the software driver to support other PHYs by defining the PHY profile using the structure `alt_tse_phy_profile` and adding it to the system using the function `alt_tse_phy_add_profile()`. For each PHY instance, use the structure `alt_tse_system_phy_struct` to define it and the function `alt_tse_system_add_sys()` to add the instance to the system.

The software driver automatically detects the PHY's operating mode and speed if the PHY conforms to the following specifications:

- One bit to specify duplex and two consecutive bits (the higher bit being the most significant bit) to specify the speed in the same extended PHY specific register.
- The speed bits are set according to the convention shown in Table 11–1.

**Table 11–1.** PHY Speed Bit Values

| Speed (Mbps) | PHY Speed Bits | |
|---|---|---|
| | **MSB** | **LSB** |
| 1000 | 1 | 0 |
| 100 | 0 | 1 |
| 10 | 0 | 0 |

For PHYs that don't conform to the aforementioned specifications, you can write a function to retrieve the PHY's operating mode and speed, and set the field `*link_status_read` in the PHY data structure to your function's address.

You can also execute a function to initialize a PHY profile or a PHY instance by setting the function pointer (`*phy_cfg` and `*tse_phy_cfg`) in the respective structures to the function's address.

Example 11–1 and Example 11–2 shows PHY profile and instance data structures.

**Example 11–1.** PHY Profile Structure

```
typedef struct alt_tse_phy_profile_struct{ /* PHY profile */

    /*The name of the PHY*/
    char name[80];

    /*Organizationally Unique Identififier*/
    alt_u32 oui;

    /*PHY model number*/
    alt_u8 model_number;

    /*PHY revision number*/
    alt_u8 revision_number;

    /*The location of the PHY Specific Status Register*/
    alt_u8 status_reg_location;

    /*The location of the Speed Status bit in the PHY Specific Status
    Register*/
    alt_u8 speed_lsb_location;

    /*The location of the Duplex Status bit in the PHY Status Specific

    Register*/
    alt_u8 duplex_bit_location;

    /*The location of the Link Status bit in PHY Status Specific
    Register*/
    alt_u8 link_bit_location;

    /*PHY initialization function pointer—profile specific*/
    alt_32 (*phy_cfg)(np_tse_mac *pmac);

    /*Pointer to the function that reads and returns 32-bit link status.Possible status:
    full duplex (bit 0 = 1), half duplex (bit 0 = 0),gigabit (bit 1 = 1),
    100Mbps (bit 2 = 1), 10Mbps (bit 3 = 1),invalid speed (bit 16 = 1).*/
    alt_u32 (*link_status_read)(np_tse_mac *pmac);

} alt_tse_phy_profile;
```

**Example 11–2.** PHY Instance Structure

```
typedef struct alt_tse_system_phy_struct { /* PHY instance */

    /* PHY's MDIO address */
   alt_32tse_phy_mdio_address;
   /* PHY initialization function pointer—instance specific */
    alt_32 (*tse_phy_cfg)(np_tse_mac *pmac);

} alt_tse_system_phy;
```

## Using Multiple SG-DMA Descriptors

To successfully use multiple SG-DMA descriptors in your application, make the following modifications:

- Set the value of the constant ALTERA_TSE_SGDMA_RX_DESC_CHAIN_SIZE in **altera_avalon_tse.h** to the number of descriptors optimal for your application. The default value is 1 and the maximum value is determined by the constant NUMBIGBUFFS. For TCP applications, Altera recommends that you use the default value.
- Increase the amount of memory allocated for the Interniche stack.

  The memory space for the Interniche stack is allocated using the Interniche function pk_alloc(). Although user applications and other network interfaces such as LAN91C111 can share the memory space, Altera recommends that you use this memory space for only one purpose, that is storing unprocessed packets for the Triple-Speed Ethernet MegaCore function. Each SG-DMA descriptor used by the device driver consumes a buffer size of 1536 bytes (defined by the constant BIGBUFSIZE) in the memory space. To achieve reasonable performance and to avoid memory exhaustion, add a new constant named NUMBIGBUFS to your application and set its value using the following guideline:

  NUMBIGBUFS = <*current value*> + <*number of SG-DMA descriptors*>

  By default, the constant NUMBIGBUFS is set to 30 in **ipport.h**. If you changed the default value in the previous release of the MegaCore function to optimize performance and resource usage, use the modified value to compute the new value of NUMBIGBUFS.

# Using Jumbo Frames

To use jumbo frames, set the `frm_length` register to 9600 and edit the files and definitions listed in Table 11–2.

**Table 11–2.** Jumbo Frames Definitions

| File | Definition |
|---|---|
| **ip\altera\ethernet\altera_eth_tse\src\software\lib\UCOSII\inc\iniche\altera_eth_tse_iniche.h** | `#define ALTERA_TSE_PKT_INIT_LEN 8206`<br>`#define ALTERA_TSE_MAX_MTU_SIZE 8192`<br>`#define ALTERA_TSE_MIN_MTU_SIZE 14` |
| **ip\altera\ethernet\altera_eth_tse\src\software\lib\HAL\inc\altera_avalon_tse.h** | `#define ALTERA_TSE_MAC_MAX_FRAME_LENGTH 8196` *(1)* |
| *<BSP project directory>* **\iniche\src\h\nios2\ipport.h** | `#ifndef BIGBUFSIZE`<br>`#define BIGBUFSIZE 1536`<br>`#endif` |

**Note to Table 11–2:**

(1) The maximum value for `ALTERA_TSE_MAC_MAX_FRAME_LENGTH` is defined by the `frm_length` register.

# API Functions

This section describes each provided API function in alphabetical order.

## alt_tse_mac_get_common_speed()

| | |
|---|---|
| **Prototype:** | `alt_tse_mac_get_common_speed(np_tse_mac *pmac)` |
| **Thread-safe:** | No. |
| **Available from ISR:** | No. |
| **Include:** | <**altera_avalon_tse.h**> |
| **Description:** | The `alt_tse_mac_get_common_speed()` obtains the common speed supported by the PHYs connected to a multiport MAC and remote link partners. |
| **Parameter:** | `pmac`—A pointer to the base of the MAC control interface. |
| **Return:** | `TSE_PHY_SPEED_1000` if the PHYs common speed is 1000 Mbps.<br>`TSE_PHY_SPEED_100` if the PHYs common speed is 100 Mbps.<br>`TSE_PHY_SPEED_10` if the PHYs common speed is 10 Mbps.<br>`TSE_PHY_SPEED_NO_COMMON` if there isn't a common speed among the PHYs. |
| **See also:** | `alt_32 alt_tse_mac_set_common_speed()` |

## alt_tse_mac_set_common_speed()

| | |
|---|---|
| **Prototype:** | `alt_tse_mac_set_common_speed(np_tse_mac *pmac, alt_32 common_speed)` |
| **Thread-safe:** | No. |
| **Available from ISR:** | No. |
| **Include:** | <**altera_avalon_tse.h**> |
| **Description:** | The `alt_tse_mac_set_common_speed()` sets the speed of a multiport MAC and the PHYs connected to it. |
| **Parameter:** | `pmac`—A pointer to the base of the MAC control interface.<br>`common_speed`—The speed to set. |
| **Return:** | `TSE_PHY_SPEED_1000` if the PHYs common speed is 1000 Mbps.<br>`TSE_PHY_SPEED_100` if the PHYs common speed is 100 Mbps.<br>`TSE_PHY_SPEED_10` if the PHYs common speed is 10 Mbps.<br>`TSE_PHY_SPEED_NO_COMMON` if there isn't a common speed among the PHYs. The current speed of the MAC and PHYs is not changed. |
| **See also:** | `alt_32 alt_tse_mac_get_common_speed()` |

## alt_tse_phy_add_profile()

| | |
|---|---|
| **Prototype:** | `alt_tse_phy_add_profile(alt_tse_phy_profile *phy)` |
| **Thread-safe:** | No. |
| **Available from ISR:** | No. |
| **Include:** | <**altera_avalon_tse.h**> |
| **Description:** | The `alt_tse_phy_add_profile()` function adds a new PHY to the PHY profile. Use this function if you want to use PHYs other than Marvell 88E1111, Marvell Quad PHY 88E1145, National DP83865, and National DP83848C. |
| **Parameter:** | `phy`—A pointer to the PHY structure. |
| **Return:** | `ALTERA_TSE_MALLOC_FAILED` if the operation is not successful. Otherwise, the index of the newly added PHY is returned. |

## alt_tse_system_add_sys()

| | |
|---|---|
| **Prototype:** | `alt_tse_system_add_sys(alt_tse_system_mac *psys_mac, alt_tse_system_sgdma *psys_sgdma, alt_tse_system_desc_mem *psys_mem, alt_tse_system_shared_fifo *psys_shared_fifo, alt_tse_system_phy *psys_phy)` |
| **Thread-safe:** | No. |
| **Available from ISR:** | No. |
| **Include:** | <**system.h**><br><**altera_avalon_tse.h**><br><**altera_avalon_tse_system_info.h**> |
| **Description:** | The `alt_tse_system_add_sys()` function defines the TSE system's components: MAC, scatter-gather DMA, memory, FIFO and PHY. This needs to be done for each port in the system. |
| **Parameter:** | `psys_mac`—A pointer to the MAC structure.<br>`psys_sgdma`—A pointer to the scatter-gather DMA structure.<br>`psys_mem`—A pointer to the memory structure.<br>`psys_shared_fifo`—A pointer to the FIFO structure.<br>`psys_phy`—A pointer to the PHY structure. |
| **Return:** | `SUCCESS` if the operation is successful.<br>`ALTERA_TSE_MALLOC_FAILED` if the operation fails.<br>`ALTERA_TSE_SYSTEM_DEF_ERROR` if one or more of the definitions are incorrect, or empty. |

## triple_speed_ethernet_init()

**Prototype:** `error_t triple_speed_ethernet_init(alt_niche_dev *p_dev)`

**Thread-safe:** No.

**Available from ISR:** No.

**Include:** <**triple_speed_ethernet_iniche.h**>

**Description:** The `triple_speed ethernet_init()` function opens and initializes the Triple-Speed Ethernet driver. Initialization involves the following operations:

- Set up the NET structure of the MAC device instance.
- Configure the MAC PHY Address.
- Register and open the SGDMA RX and TX Module of the MAC device instance.
- Enable the SGDMA RX interrupt and register it to the Operating System.
- Register the SGDMA RX callback function.
- Obtains the PHY Speed of the MAC.
- Set up the Ethernet MAC Register settings for the Triple-Speed Ethernet driver operation.
- Set up the initial descriptor chain to start the SGDMA RX operation.

**Parameter:** `p_dev`—A pointer to the Triple-Speed Ethernet device instance.

**Return:** `SUCCESS` if the Triple-Speed Ethernet driver is successfully initialized.

**See also:** `tse_mac_close()`

## tse_mac_close()

**Prototype:** `int tse_mac_close(int iface)`

**Thread-safe:** No.

**Available from ISR:** No.

**Include:** <**triple_speed_ethernet_iniche.h**>

**Description:** The `tse_mac_close()` closes the Triple-Speed Ethernet driver by performing the following operations:

- Configure the admin and operation status of the NET structure of the Triple-Speed Ethernet driver instance to `ALTERA_TSE_ADMIN_STATUS_DOWN`.
- De-register the SGDMA RX interrupt from the operating system.
- Clear the `RX_ENA` bit in the `command_config` register to disable the RX datapath

**Parameter:** `iface`—The index of the MAC interface. This argument is reserved for configurations that contain multiple MAC instances.

**Return:** `SUCCESS` if the close operations are successful.
An error code if de-registration of SGDMA RX from the operating system failed.

**See also:** `triple_speed_ethernet_init()`

## tse_mac_raw_send()

| | |
|---|---|
| **Prototype:** | `int tse_mac_raw_send(NET net, char *data, unsigned data_bytes)` |
| **Thread-safe:** | No. |
| **Available from ISR:** | No. |
| **Include:** | <**triple_speed_ethernet_iniche.h**> |
| **Description:** | The `tse_mac_raw_send()` function sends Ethernet frames data to the MAC function. It validates the arguments to ensure the data length is greater than the ethernet header size specified by `ALTERA_TSE_MIN_MTU_SIZE`. The function also ensures the SGDMA TX engine is not busy prior to constructing the descriptor for the current transmit operation.<br>Upon successful validations, this function calls the internal API, `tse_mac_sTxWrite`, to initiate the synchronous SGDMA transmit operation on the current data buffer. |
| **Parameter:** | `net`—The NET structure of the Triple-Speed Ethernet MAC instance.<br>`data`—A data pointer to the base of the Ethernet frame data, including the header, to be transmitted to the MAC. The data pointer is assumed to be word-aligned.<br>`data_bytes`—The total number of bytes in the Ethernet frame including the additional padding bytes as specified by `ETHHDR_BIAS`. |
| **Return:** | `SUCCESS` if the current data buffer is successfully transmitted.<br>`SEND_DROPPED` if the number of data bytes is less than the Ethernet header size.<br>`ENP_RESOURCE` if the SGDMA TX engine is busy. |

## tse_mac_setGMII mode()

| | |
|---|---|
| **Prototype:** | `int tse_mac_setGMIImode(np_tse_mac *pmac)` |
| **Thread-safe:** | No. |
| **Available from ISR:** | No. |
| **Include:** | <**triple_speed_ethernet_iniche.h**> |
| **Description:** | The `tse_mac_setGMIImode()` function sets the MAC function operation mode to Gigabit (GMII). The settings of the `command_config` register are restored at the end of the function. |
| **Parameter:** | `pmac`—A pointer to the MAC control interface base address. |
| **Return:** | `SUCCESS` |
| **See also:** | `tse_mac_setMIImode()` |

## tse_mac_setMIImode()

| | |
|---|---|
| **Prototype:** | `int tse_mac_setMIImode(np_tse_mac *pmac)` |
| **Thread-safe:** | No. |
| **Available from ISR:** | No. |
| **Include:** | <**triple_speed_ethernet_iniche.h**> |
| **Description:** | The `tse_mac_setMIImode()` function sets the MAC function operation mode to MII (10/100). The settings of the `command_config` register are restored at the end of the function. |
| **Parameter:** | `pmac`—A pointer to the MAC control interface base address. |
| **Return:** | `SUCCESS` |
| **See also:** | `tse_mac_setGMIImode()` |

## tse_mac_SwReset()

| | |
|---|---|
| **Prototype:** | `int tse_mac_SwReset(np_tse_mac *pmac)` |
| **Thread-safe:** | No. |
| **Available from ISR:** | No. |
| **Include:** | <**triple_speed_ethernet_iniche.h**> |
| **Description:** | The `tse_mac_SwReset()` performs a software reset on the MAC function. A software reset occurs with some latency as specified by `ALTERA_TSE_SW_RESET_TIME_OUT_CNT`. The settings of the `command_config` register are restored at the end of the function. |
| **Parameter:** | `pmac`—A pointer to the MAC control interface base address. |
| **Return:** | `SUCCESS` |

# Constants

Table 11–3 lists all constants defined for the MAC registers manipulation and provides links to detailed descriptions of the registers. It also list the constants that define the MAC operating mode and timeout values.

**Table 11–3.** Constants Mapping (Part 1 of 3)

| Constant | Value | Description |
|---|---|---|
| ALTERA_TSE_DUPLEX_MODE_DEFAULT | 1 | 0: Half-duplex<br>1: Full-duplex |
| ALTERA_TSE_MAC_SPEED_DEFAULT | 0 | 0: 10 Mbps<br>1: 100 Mbps<br>2: 1000 Mbps |
| ALTERA_TSE_SGDMA_RX_DESC_CHAIN_SIZE | 1 | The number of SG-DMA descriptors required for the current operating mode. |
| ALTERA_CHECKLINK_TIMEOUT_THRESHOLD | 1000000 | The timeout value when the MAC tries to establish a link with a PHY. |
| ALTERA_AUTONEG_TIMEOUT_THRESHOLD | 250000 | The auto-negotiation timeout value. |
| **Command_Config Register** (“Command_Config Register (Dword Offset 0x02)” on page 6–6) | | |
| ALTERA_TSEMAC_CMD_TX_ENA_OFST | 0 | Configures the `TX_ENA` bit. |
| ALTERA_TSEMAC_CMD_TX_ENA_MSK | 0x1 | |
| ALTERA_TSEMAC_CMD_RX_ENA_OFST | 1 | Configures the `RX_ENA` bit. |
| ALTERA_TSEMAC_CMD_RX_ENA_MSK | 0x2 | |
| ALTERA_TSEMAC_CMD_XON_GEN_OFST | 2 | Configures the `XON_GEN` bit. |
| ALTERA_TSEMAC_CMD_XON_GEN_MSK | 0x4 | |
| ALTERA_TSEMAC_CMD_ETH_SPEED_OFST | 3 | Configures the `ETH_SPEED` bit. |
| ALTERA_TSEMAC_CMD_ETH_SPEED_MSK | 0x8 | |
| ALTERA_TSEMAC_CMD_PROMIS_EN_OFST | 4 | Configures the `PROMIS_EN` bit. |
| ALTERA_TSEMAC_CMD_PROMIS_EN_MSK | 0x10 | |
| ALTERA_TSEMAC_CMD_PAD_EN_OFST | 5 | Configures the `PAD_EN` bit. |
| ALTERA_TSEMAC_CMD_PAD_EN_MSK | 0x20 | |

**Table 11–3.** Constants Mapping (Part 2 of 3)

| Constant | Value | Description |
|---|---|---|
| ALTERA_TSEMAC_CMD_CRC_FWD_OFST | 6 | Configures the CRC_FWD bit. |
| ALTERA_TSEMAC_CMD_CRC_FWD_MSK | 0x40 | |
| ALTERA_TSEMAC_CMD_PAUSE_FWD_OFST | 7 | Configures the PAUSE_FWD bit. |
| ALTERA_TSEMAC_CMD_PAUSE_FWD_MSK | 0x80 | |
| ALTERA_TSEMAC_CMD_PAUSE_IGNORE_OFST | 8 | Configures the PAUSE_IGNORE bit. |
| ALTERA_TSEMAC_CMD_PAUSE_IGNORE_MSK | 0x100 | |
| ALTERA_TSEMAC_CMD_TX_ADDR_INS_OFST | 9 | Configures the TX_ADDR_INS bit. |
| ALTERA_TSEMAC_CMD_TX_ADDR_INS_MSK | 0x200 | |
| ALTERA_TSEMAC_CMD_HD_ENA_OFST | 10 | Configures the HD_ENA bit. |
| ALTERA_TSEMAC_CMD_HD_ENA_MSK | 0x400 | |
| ALTERA_TSEMAC_CMD_EXCESS_COL_OFST | 11 | Configures the EXCESS_COL bit. |
| ALTERA_TSEMAC_CMD_EXCESS_COL_MSK | 0x800 | |
| ALTERA_TSEMAC_CMD_LATE_COL_OFST | 12 | Configures the LATE_COL bit. |
| ALTERA_TSEMAC_CMD_LATE_COL_MSK | 0x1000 | |
| ALTERA_TSEMAC_CMD_SW_RESET_OFST | 13 | Configures the SW_RESET bit. |
| ALTERA_TSEMAC_CMD_SW_RESET_MSK | 0x2000 | |
| ALTERA_TSEMAC_CMD_MHASH_SEL_OFST | 14 | Configures the MHASH_SEL bit. |
| ALTERA_TSEMAC_CMD_MHASH_SEL_MSK | 0x4000 | |
| ALTERA_TSEMAC_CMD_LOOPBACK_OFST | 15 | Configures the LOOP_ENA bit. |
| ALTERA_TSEMAC_CMD_LOOPBACK_MSK | 0x8000 | |
| ALTERA_TSEMAC_CMD_TX_ADDR_SEL_OFST | 16 | Configures the TX_ADDR_SEL bits (bits 16 - 18). |
| ALTERA_TSEMAC_CMD_TX_ADDR_SEL_MSK | 0x70000 | |
| ALTERA_TSEMAC_CMD_MAGIC_ENA_OFST | 19 | Configures the MAGIC_ENA bit. |
| ALTERA_TSEMAC_CMD_MAGIC_ENA_MSK | 0x80000 | |
| ALTERA_TSEMAC_CMD_SLEEP_OFST | 20 | Configures the SLEEP bit. |
| ALTERA_TSEMAC_CMD_SLEEP_MSK | 0x100000 | |
| ALTERA_TSEMAC_CMD_WAKEUP_OFST | 21 | Configures the WAKEUP bit. |
| ALTERA_TSEMAC_CMD_WAKEUP_MSK | 0x200000 | |
| ALTERA_TSEMAC_CMD_XOFF_GEN_OFST | 22 | Configures the XOFF_GEN bit. |
| ALTERA_TSEMAC_CMD_XOFF_GEN_MSK | 0x400000 | |
| ALTERA_TSEMAC_CMD_CNTL_FRM_ENA_OFST | 23 | Configures the CNTL_FRM_ENA bit. |
| ALTERA_TSEMAC_CMD_CNTL_FRM_ENA_MSK | 0x800000 | |
| ALTERA_TSEMAC_CMD_NO_LENGTH_CHECK_OFST | 24 | Configures the NO_LENGTH_CHECK bit. |
| ALTERA_TSEMAC_CMD_NO_LENGTH_CHECK_MSK | 0x1000000 | |
| ALTERA_TSEMAC_CMD_ENA_10_OFST | 25 | Configures the ENA_10 bit. |
| ALTERA_TSEMAC_CMD_ENA_10_MSK | 0x2000000 | |
| ALTERA_TSEMAC_CMD_RX_ERR_DISC_OFST | 26 | Configures the RX_ERR_DISC bit. |
| ALTERA_TSEMAC_CMD_RX_ERR_DISC_MSK | 0x4000000 | |

**Table 11–3.** Constants Mapping (Part 3 of 3)

| Constant | Value | Description |
|---|---|---|
| ALTERA_TSEMAC_CMD_CNT_RESET_OFST | 31 | Configures the CNT_RESET bit. |
| ALTERA_TSEMAC_CMD_CNT_RESET_MSK | 0x80000000 | |
| **Tx_Cmd_Stat Register** (“Transmit and Receive Command Registers (Dword Offset 0x3A – 0x3B)” on page 6–12) | | |
| ALTERA_TSEMAC_TX_CMD_STAT_OMITCRC_OFST | 17 | Configures the OMIT_CRC bit. |
| ALTERA_TSEMAC_TX_CMD_STAT_OMITCRC_MSK | 0x20000 | |
| ALTERA_TSEMAC_TX_CMD_STAT_TXSHIFT16_OFST | 18 | Configures the TX_SHIFT16 bit. |
| ALTERA_TSEMAC_TX_CMD_STAT_TXSHIFT16_MSK | 0x40000 | |
| **Rx_Cmd_Stat Register** (“Transmit and Receive Command Registers (Dword Offset 0x3A – 0x3B)” on page 6–12) | | |
| ALTERA_TSEMAC_RX_CMD_STAT_RXSHIFT16_OFST | 25 | Configures the RX_SHIFT16 bit |
| ALTERA_TSEMAC_RX_CMD_STAT_RXSHIFT16_MSK | 0x2000000 | |

# A. MegaCore Evaluation

## OpenCore Plus Evaluation

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

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP℠ megafunction) within your system.
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily.
- Generate time-limited device programming files for designs that include megafunctions.
- Program a device and verify your design in hardware.

You only need to purchase a license for the megafunction when you are completely satisfied with its functionality and performance, and want to take your design to production.

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

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

### OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation supports the following two operation modes:

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

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

For MegaCore functions, the untethered timeout is 1 hour; the tethered timeout value is indefinite.

Your design stops working after the hardware evaluation time expires, and some signals are forced low. The following signals are forced low in configurations that contain the 10/100/1000 Ethernet MAC function: `ff_rx_data`, `ff_rx_mod`, `ff_rx_eop`, `ff_rx_sop`, `ff_rx_dval`, `rx_err`, `gm_tx_d`, `gm_tx_en`, `gm_tx_err`, `m_tx_d`, `m_tx_en`, and `m_tx_err`.

In configurations that contain the 1000BASE-X/SGMII PCS function, the following signals are forced low:
gmii_rx_d, gmii_rx_dv, gmii_rx_err, mii_rx_d, mii_rx_dv, mii_rx_err, mii_rx_col, mii_rx_crs, and tbi_tx_d.

# B. Ethernet Frame Format

## Basic Frame Format

Figure B–1 shows the format of a basic Ethernet frame.

**Figure B–1.** MAC Frame Format

| | | |
|---|---|---|
| | 7 octets | PREAMBLE |
| | 1 octet | SFD |
| Frame length | 6 octets | DESTINATION ADDRESS |
| Frame length | 6 octets | SOURCE ADDRESS |
| Frame length | 2 octets | LENGTH/TYPE |
| Frame length | 0..1500/9600 octets | PAYLOAD DATA |
| Frame length | 0..46 octets | PAD |
| Frame length | 4 octets | FRAME CHECK SEQUENCE |
| | | EXTENSION (half duplex only) |

A basic Ethernet frame comprises the following fields:

- Preamble—a maximum of 7-octet fixed value of 0x55.
- Start frame delimiter (SFD)—a 1-octet fixed value of 0xD5 which marks the beginning of a frame.
- Destination and source addresses—6 octets each. The least significant byte is transmitted first.
- Length or type—a 2-octet value equal to or greater than 1536 (0x600) indicates a type field. Otherwise, this field contains the length of the payload data. The most significant byte of this field is transmitted first.
- Payload Data and Pad—variable length data and padding.
- Frame check sequence (FCS)—a 4-octet cyclic redundancy check (CRC) value for detecting frame errors during transmission.
- An extension field—Required only for gigabit Ethernet operating in half-duplex mode. The MAC function does not support this implementation.

## VLAN and Stacked VLAN Frame Format

The extension of a basic MAC frame is a virtual local area network (VLAN) tagged frame, which contains an additional 4-byte field for the VLAN tag and information between the source address and length/type fields. VLAN tagging is defined by the IEEE Standard 802.1Q. VLAN tagging can identify and separate many groups' network traffic from each other in enterprise and metro networks. Each VLAN group can consist of many users with varied MAC address in different geographical locations of a network. VLAN tagging increases and scales the network performance and add privacy and safety to various groups and customers' network traffic.

VLAN tagged frames have a maximum length of 1522 bytes, excluding the preamble and the SFD fields. Figure B–2 shows the format of a VLAN tagged frame.

**Figure B–2.** VLAN Tagged MAC Frame Format

| | |
|---|---|
| 7 octets | PREAMBLE |
| 1 octet | SFD |
| 6 octets | DESTINATION ADDRESS |
| 6 octets | SOURCE ADDRESS |
| 2 octets | LENGTH/TYPE (VLAN Tag 0x8100) |
| 2 octets | VLAN info |
| 2 octets | CLIENT LENGTH/TYPE |
| 0..1500/9600 octets | PAYLOAD DATA |
| 0..42 octets | PAD |
| 4 octets | FRAME CHECK SEQUENCE |
| | EXTENSION (half duplex only) |

Frame length

In metro Ethernet applications, which require more scalability and security due to the sharing of an Ethernet link by many service providers, MAC 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 client length/type fields, as illustrated in Figure B–3.

**Figure B–3.** Stacked VLAN Tagged MAC Frame Format

| | |
|---|---|
| 7 octets | PREAMBLE |
| 1 octet | SFD |
| 6 octets | DESTINATION ADDRESS |
| 6 octets | SOURCE ADDRESS |
| 2 octets | LENGTH/TYPE (VLAN Tag 0x8100) |
| 2 octets | VLAN info |
| 2 octets | LENGTH/TYPE (VLAN Tag 0x8100) |
| 2 octets | VLAN info |
| 2 octets | CLIENT LENGTH/TYPE |
| 0..1500/9600 octets | PAYLOAD DATA |
| 0..38 octets | PAD |
| 4 octets | FRAME CHECK SEQUENCE |
| | EXTENSION (half duplex only) |

Frame length

Stacked VLANs

# Pause Frame Format

A pause frame is generated by the receiving device to indicate congestion to the emitting device. If flow control is supported, the emitting device should stop sending data upon receiving pause frames.

Figure B–4 shows the format of pause frames. The length/type field has a fixed value of 0x8808, followed by a 2-octet opcode field of 0x0001. A 2-octet pause quanta is defined in the second and third bytes of the frame payload (P1 and P2). The pause quanta, P1, is the most significant byte. A pause frame has no payload length field, and is always padded with 42 bytes of 0x00.

**Figure B–4.** Pause Frame Format

| | | |
|---|---|---|
| 7octets | PREAMBLE | |
| 1 octet | SFD | |
| 6 octets | DESTINATION ADDRESS | |
| 6 octets | SOURCE ADDRESS | |
| 2 octets | TYPE (0x8808) | |
| 2 octets | OPCODE (0X0001) | |
| 2 octets | PAUSE QUANTA (P1, P2) | Payload |
| 42 octets | PAD | |
| 4 octets | CRC | |

## Pause Frame Generation

When you turn on the **Enable full-duplex flow control** option, pause frame generation is triggered by the following events:

- RX FIFO fill level hits the `rx_section_empty` threshold.
- XOFF register write.
- XON register write.
- XOFF I/O pin (`xoff_gen`) assertion.
- XON I/O pin (`xon_gen`) assertion.

If the RX FIFO buffer is almost full, the MAC function triggers the pause frame generation to the remote Ethernet device.

If the local Ethernet device needs to generate pause frame via XOFF or XON register write or I/O pin assertion, it is recommended to set the `rx_section_empty` register to a larger value to avoid non-deterministic result.

Table B–1 summarizes the pause frame generation based on the above events.

**Table B–1.** Pause Frame Generation

| Register Write or I/O Pin Assertion *(1)* | | |
|---|---|---|
| **XOFF_GEN** | **XON_GEN** | **Description** |
| 1 | 0 | If the `XOFF_GEN` bit is set to 1, the XOFF pause frames are continuously generated and sent to the MII/GMII TX interface until the `XOFF_GEN` bit is cleared. |
| 0 | 1 | If the `XON_GEN` bit is set to 1, the XON pause frames are continuously generated and sent to the MII/GMII TX interface until the `XON_GEN` bit is cleared. |
| 1 | 1 | This event is not recommended as it will produce non-deterministic result. |

**Note to Table B–1:**

(1) Set the XON and XOFF registers to 0 when you use the I/O pin to generate the pause frame and vice versa.

## Functionality Configuration Parameters

You can use the parameters in Table C–1 to enable or disable specific functionality in the MAC and PCS.

**Table C–1.** MegaCore Functionality Configuration Parameters (Part 1 of 2)

| Parameter | Description | Default |
|---|---|---|
| **Supported in configurations that contain the 10/100/1000 Ethernet MAC** | | |
| ETH_MODE | 10: Enables MII.<br>100: Enables MII.<br>1000: Enables GMII. | 1000 |
| HD_ENA | Sets the HD_ENA bit in the command_config register. See Table 6–3 on page 6–6. | 0 |
| TB_MACPAUSEQ | Sets the pause_quant register. See Table 6–2 on page 6–3. | 15 |
| TB_MACIGNORE_PAUSE | Sets the PAUSE_IGNORE bit in the command_config register. See Table 6–3 on page 6–6. | 0 |
| TB_MACFWD_PAUSE | Sets the PAUSE_FWD bit in the command_config register. See Table 6–3 on page 6–6. | 0 |
| TB_MACFWD_CRC | Sets the CRC_FWD bit in the command_config register. See Table 6–3 on page 6–6. | 0 |
| TB_MACINSERT_ADDR | Sets the ADDR_INS bit in the command_config register. See Table 6–3 on page 6–6. | 0 |
| TB_PROMIS_ENA | Sets the PROMIS_EN bit in the command_config register. See Table 6–3 on page 6–6. | 1 |
| TB_MACPADEN | Sets the PAD_EN bit in the command_config register. See Table 6–3 on page 6–6. | 1 |
| TB_MACLENMAX | Maximum frame length. | 1518 |
| TB_IPG_LENGTH | Sets the tx_ipg_length register. See Table 6–2 on page 6–3. | 12 |
| TB_MDIO_ADDR0 | Sets the mdio_addr0 register. See Table 6–2 on page 6–3. | 0 |
| TB_MDIO_ADDR1 | Sets the mdio_addr1 register. See Table 6–2 on page 6–3. | 1 |
| TX_FIFO_AE | Sets the tx_almost_empty register. See Table 6–2 on page 6–3. | 8 |
| TX_FIFO_AF | Sets the tx_almost_full register. See Table 6–2 on page 6–3. | 10 |
| RX_FIFO_AE | Sets the rx_almost_empty register. See Table 6–2 on page 6–3. | 8 |
| RX_FIFO_AF | Sets the rx_almost_full register. See Table 6–2 on page 6–3. | 8 |
| TX_FIFO_SECTION_EMPTY | Sets the tx_section_empty register. See Table 6–2 on page 6–3. | 16 |

**Table C–1.** MegaCore Functionality Configuration Parameters (Part 2 of 2)

| Parameter | Description | Default |
|---|---|---|
| TX_FIFO_SECTION_FULL | Sets the tx_section_full register. See Table 6–2 on page 6–3. | 16 |
| RX_FIFO_SECTION_EMPTY | Sets the rx_section_empty register. See Table 6–2 on page 6–3. | 0 |
| RX_FIFO_SECTION_FULL | Sets the rx_section_full register. See Table 6–2 on page 6–3. | 16 |
| MCAST_TABLEN | Specifies the first *n* addresses from MCAST_ADDRESSLIST from which multicast address is selected. | 9 |
| MCAST_ADDRESSLIST | A list of multicast addresses. | 0x887654332211<br>0x886644352611<br>0xABCDEF012313<br>0x92456545AB15<br>0x432680010217<br>0xADB589215439<br>0xFFEACFE3434B<br>0xFFCCDDAA3123<br>0xADB358415439 |
| **Supported in configurations that contain the 1000BASE-X/SGMII PCS** | | |
| TB_SGMII_ENA | Sets the SGMII_ENA bit in the if_mode register. See Table 6–16 on page 6–23. | 0 |
| TB_SGMII_AUTO_CONF | Sets the USE_GMII_AN bit in the if_mode register. See Table 6–16 on page 6–23. | 0 |

# Test Configuration Parameters

You can use the parameters in Table C–2 to create custom test scenarios.

**Table C–2.** Test Configuration Parameters (Part 1 of 2)

| Parameter | Description | Default |
|---|---|---|
| **Supported in configurations that contain the 10/100/1000 Ethernet MAC** | | |
| TB_RXFRAMES | Enables local loopback on the Ethernet side (GMII/MII/RGMII). The value must always be set to 0. | 0 |
| TB_TXFRAMES | Specifies the number of frames to be generated by the Avalon-ST Ethernet frame generator. | 5 |
| TB_RXIPG | IPG on the receive path. | 12 |
| TB_ENA_VAR_IPG | 0: A constant IPG, TB_RXIPG, is used by the GMII/RGMII/MII Ethernet frame generator.<br>1: Enables variable IPG on the receive path. | 0 |
| TB_LENSTART | Specifies the payload length of the first frame generated by the frame generators. The payload length of each subsequent frame is incremented by the value of TB_LENSTEP. | 100 |
| TB_LENSTEP | Specifies the payload length increment. | 1 |
| TB_LENMAX | Specifies the maximum payload length generated by the frame generators. If the payload length exceeds this value, it wraps around to TB_LENSTART. This parameter can be used to test frame length error by setting it to a value larger than the value of TB_MACLENMAX. | 1500 |

**Table C–2.** Test Configuration Parameters (Part 2 of 2)

| Parameter | Description | Default |
|---|---|---|
| TB_ENA_PADDING | 0: Disables padding.<br>1: If the length of frames generated by the GMII/RGMII/MII Ethernet frame generator is less than the minimum frame length (64 bytes), the generator inserts padding bytes to the frames to make up the minimum length. | 1 |
| TB_ENA_VLAN | 0: Only basic frames are generated.<br>1: Enables VLAN frames generation. This value specifies the number of basic frames generated before a VLAN frame is generated followed by a stacked VLAN frame. | 0 |
| TB_STOPREAD | Specifies the number of packets to be read from the receive FIFO before reading is suspended. You can use this parameter to test FIFO overflow and flow control. | 0 |
| TB_HOLDREAD | Specifies the number of clock cycles before the Avalon-ST monitor stops reading from the receive FIFO. | 1000 |
| TB_TX_FF_ERR | 0: Normal behavior.<br>1: Drives the Avalon-ST error signal high to simulate erroneous frames transmission. | 0 |
| TB_TRIGGERXOFF | Specifies the number of clock cycles from the start of simulation before the xoff_gen signal is driven. | 0 |
| TB_TRIGGERXON | Specifies the number of clock cycles from the start of simulation before the xon_gen signal is driven high. | 0 |
| RX_COL_FRM | Specifies which frame is received with collision. Valid in fast Ethernet and half-duplex mode only. | 0 |
| RX_COL_GEN | Specifies which nibble within the frame collision occurs. | 0 |
| TX_COL_FRM | Specifies which frame is transmitted with a collision. Valid in fast Ethernet and half-duplex mode only. | 0 |
| TX_COL_GEN | Specifies which nibble within the frame collision occurs on the transmit path. | 0 |
| TX_COL_NUM | Specifies the number of consecutive collisions during retransmission. | 0 |
| TX_COL_DELAY | Specifies the delay, in nibbles, between collision and retransmission. | 0 |
| TB_PAUSECONTROL | 0: GMII frame generator does not respond to pause frames.<br>1: Enables flow control in the GMII frame generator. | 1 |
| TB_MDIO_SIMULATION | Enable / Disable MDIO simulation. | 0 |
| **Supported in configurations that contain the 1000BASE-X/SGMII PCS** | | |
| TB_SGMII_HD | 0: Disables half-duplex mode.<br>1: Enables half-duplex mode. | 0 |
| TB_SGMII_1000 | 0: Disables gigabit operation.<br>1: Enables gigabit operation. | 1 |
| TB_SGMII_100 | 0: Disables 100 Mbps operation.<br>1: Enables 100 Mbps operation. | 0 |
| TB_SGMII_10 | 0: Disables 10 Mbps operation.<br>1: Enables 10 Mbps operation. | 0 |
| TB_TX_ERR | 0: Disables error generation.<br>1: Enables error generation. | 0 |

# D. Time-of-Day Clock

The Time-of-Day (ToD) clock provides a stream of timestamps for the IEEE 1588v2 feature.

## Features

- Provides a stream of 64-bit and 96-bit timestamps.
  - The 64-bit timestamp has 48-bit nanosecond field and 16-bit fractional nanosecond field.
  - The 96-bit timestamp has 48-bit second field, 32-bit nanosecond field, and 16-bit fractional nanosecond field.
- Runs at 125-MHz for the Triple-Speed Ethernet MegaCore function.
- Supports coarse adjustment and fine adjustments through clean frequency adjustment.
- Supports period adjustment for frequency control using the `Period` register.
- Supports offset adjustment using the `AdjustPeriod` register.

## Device Family Support

Table D–1 shows the level of support offered by the Time-of-Day clock for each Altera device family.

**Table D–1.** Device Family Support

| Device Family | Support |
|---|---|
| Arria V GX/GT/GZ/SoC | Preliminary |
| Cyclone V GX/GT/SoC | Preliminary |
| Stratix V GX/GT | Preliminary |
| Other device families | No support |

## Performance and Resource Utilization

Table D–2 provides the estimated resource utilization and performance of the TOD clock for the Stratix V device family. The estimates are obtained by compiling the Triple-Speed Ethernet MegaCore function using the Quartus II software targeting a Stratix V GX (5SGXMA7N3F45C3) device with speed grade -3.

**Table D–2.** Stratix V Performance and Resource Utilization

| MegaCore Function | Settings | FIFO Buffer Size (Bits) | Combinational ALUTs | Logic Registers | Memory (M20K Blocks/ MLAB Bits) |
|---|---|---|---|---|---|
| TOD Clock | Default | 0 | 378 | 1,120 | 0/0 |

# Parameter Setting

Table D–3 describes the Time-of-Day clock configuration parameters.

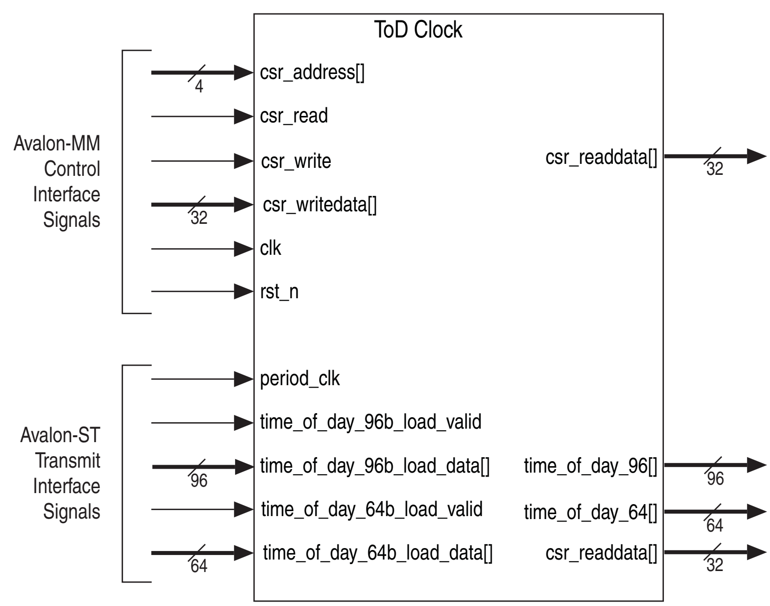
**Table D–3.** ToD Configuration Parameters

| Name | Value | Description |
|---|---|---|
| DEFAULT_NSEC_PERIOD | Between 0 and 0x000F | 4-bit value that defines the reset value for PERIOD_NS. For Triple-Speed Ethernet MegaCore function, set the value to 0x0008.<br>The default value is 0x0006. |
| DEFAULT_FNSEC_PERIOD | Between 0 and 0xFFFF | 16-bit value that defines the reset value for PERIOD_FNS. For Triple-Speed Ethernet MegaCore function, set the value to 0x0000.<br>The default value is 0x6666. |
| DEFAULT_NSEC_ADJPERIOD | Between 0 and 0x000F | 4-bit value that defines the reset value for ADJPERIOD_NS.<br>The default value is 0x0006. |
| DEFAULT_FNSEC_ADJPERIOD | Between 0 and 0xFFFF | 16-bit value that defines the reset value for PERIOD_FNS.<br>The default value is 0x6666. |

# Time-of-Day Clock Interface Signals

Figure D–1 shows the interface signals for the ToD clock.

**Figure D–1.** Time-of-Day Clock Interface Signals



## Avalon-MM Control Interface Signals

Table D–4 describes the Avalon-MM control interface signals for the ToD clock.

**Table D–4.** Avalon-MM Control Interface Signals for ToD Clock

| Signal | Direction | Width | Description |
|---|---|---|---|
| csr_address[] | Input | 2 | Use this bus to specify the register address you want to read from or write to. |
| csr_read | Input | 1 | Assert this signal to request a read. |
| csr_readdata[] | Output | 32 | Carries the data read from the specified register. |
| csr_write | Input | 1 | Assert this signal to request a write. |
| csr_writedata[] | Input | 32 | Carries the data to be written to the specified register. |
| clk | Input | 1 | Register access reference clock. |
| rst_n | Input | 1 | Assert this active low signal to reset the ToD clock. |

## Avalon-ST Transmit Interface Signals

Table D–5 describes the Avalon Streaming (Avalon-ST) transmit interface signals for the ToD clock.

**Table D–5.** Avalon-ST Transmit Interface Signals for ToD Clock (Part 1 of 2)

| Signal | Direction | Width | Description |
|---|---|---|---|
| time_of_day_64b[] | Output | 64 | Timestamp from the ToD clock<br>■ Bits 0 to 15: 16-bit fractional nanosecond field<br>■ Bits 16 to 63: 48-bit nanosecond field |
| time_of_day_96b[] | Output | 96 | Timestamp from the ToD clock<br>■ Bits 0 to 15: 16-bit fractional nanosecond field<br>■ Bits 16 to 47: 32-bit nanosecond field<br>■ Bits 48 to 95: 48-bit second field |
| time_of_day_96b_load_valid | Input | 1 | Indicates that the synchronized ToD is valid. Every time you assert this signal, the synchronized ToD is loaded into the ToD clock. Assert this signal for only one clock cycle. |
| time_of_day_96b_load_data[] | Input | 96 | Loads 96-bit synchronized ToD from master ToD clock to slave ToD clock within 1 clock cycle.<br>■ Bits 0 to 15: 16-bit fractional nanosecond field<br>■ Bits 16 to 63: 32-bit nanosecond field<br>■ Bits 64 to 95: 48-bit second field |
| time_of_day_64b_load_valid | Input | 1 | Indicates that the synchronized ToD is valid. Every time you assert this signal, the synchronized ToD is loaded into the ToD clock. Assert this signal for only one clock cycle. |

**Table D–5.** Avalon-ST Transmit Interface Signals for ToD Clock (Part 2 of 2)

| Signal | Direction | Width | Description |
|---|---|---|---|
| `time_of_day_64b_load_data[]` | Input | 64 | Loads 64-bit synchronized ToD from master ToD clock to slave ToD clock within 1 clock cycle.<br>■ Bits 0 to 15: 16-bit fractional nanosecond field<br>■ Bits 16 to 63: 48-bit nanosecond field |
| `period_clk` | Input | 1 | Clock for the ToD clock. The clock must be in the same clock domain as `tx_time_of_day` and `rx_time_of_day` in the MAC function. |
| `period_rst_n` | Input | 1 | Assert this signal to reset `period_clk` to the same clock domain as `tx_time_of_day` and `rx_time_of_day` in the MAC function. |

# Time-of-Day Clock Configuration Register Space

Table D–6 describes the ToD clock register space.

**Table D–6.** Time-of-Day Clock Registers (Part 1 of 2)

| Dword Offset | Name | R/W | Description | HW Reset |
|---|---|---|---|---|
| 0x00 | `SecondsH` | RW | ■ Bits 0 to 15: High-order 16-bit second field.<br>■ Bits 16 to 31: Not used. | 0x0 |
| 0x01 | `SecondsL` | RW | Bits 0 to 32: Low-order 32-bit second field. | 0x0 |
| 0x02 | `NanoSec` | RW | Bits 0 to 32: 32-bit nanosecond field. | 0x0 |
| 0x03 | Reserved | — | Reserved for future use | — |
| 0x04 | `Period` | RW | The period for the frequency adjustment.<br>■ Bits 0 to 15: Period in fractional nanosecond (`PERIOD_FNS`).<br>■ Bits 16 to 19: Period in nanosecond (`PERIOD_NS`).<br>■ Bits 20 to 31: Not used.<br>The default value for the period depends on the $f_{MAX}$ of the MAC function. For example, if $f_{MAX}$ = 125-MHz, the period is 8-ns (`PERIOD_NS` = 0x0008 and `PERIOD_FNS` = 0x0000). | *n* |
| 0x05 | `AdjustPeriod` | RW | The period for the offset adjustment.<br>■ Bits 0 to 15: Period in fractional nanosecond (`ADJPERIOD_FNS`).<br>■ Bits 16 to 19: Period in nanosecond (`ADJPERIOD_NS`).<br>■ Bits 20 to 31: Not used. | 0x0 |
| 0x06 | `AdjustCount` | RW | ■ Bits 0 to 19: The number of `AdjustPeriod` clock cycles used during offset adjustment.<br>■ Bits 20 to 31: Not used. | 0x06 |

**Table D–6.** Time-of-Day Clock Registers (Part 2 of 2)

| Dword Offset | Name | R/W | Description | HW Reset |
|---|---|---|---|---|
| 0x1C | `DriftAdjust` | RW | The drift of ToD adjusted periodically by adding a correction value as configured in this register space.<br>■ Bits 0 to 15: Adjustment value in fractional nanosecond (`DRIFT_ADJUST_FNS`). This value is added into the current ToD during the adjustment. The default value is 0.<br>■ Bits 16 to 19: Adjustment value in nanosecond (`DRIFT_ADJUST_NS`). This value is added into the current ToD during the adjustment. The default value is 0.<br>■ Bits 20 to 32: Not used. | 0x0 |
| 0x20 | `DriftAdjustRate` | RW | The count of clock cycles for each ToD's drift adjustment to take effect.<br>■ Bits 0 to 15: The number of clock cycles (`ADJUST_RATE`). The ToD adjustment happens once after every period in number of clock cycles as indicated by this register space.<br>■ Bits 16 to 32: Not used. | 0x0 |

## Adjusting Time-of-Day Clock Drift

You can use the `DriftAdjust` and `DriftAdjustRate` registers to correct any drift in the ToD clock.

In the case of a ToD for 10G with period of 6.4ns, the nanosecond field is converted directly to `PERIOD_NS` while the fractional nanosecond need to be multiplied with $2^{16}$ or 65536 in order to convert to `PERIOD_FNS`. This results in 0x6 `PERIOD_NS` and 0x6666.4 `PERIOD_FNS`.

`PERIOD_NS` only accepts 0x6666 and ignores 0x0000.4, which in turn would cause some inaccuracy in the configured period. This inaccuracy will cause the ToD to drift from the actual time as much as 953.67ns after a period of 1 second. You would notice that after every 5 cycles, 0x0000.4 accumulates to 0x0002. If the TOD is able to add 0x0002 of fractional nanosecond into the ToD once after every period of 5 cycles, then it will correct the drift.

Therefore, for the 10G case, `DRIFT_ADJUST_NS` is now configured to 0x0, `DRIFT_ADJUST_FNS` is configured to 0x0002 and `ADJUST_RATE` is configured to 0x5.

# E. ToD Synchronizer

The ToD Synchronizer provides a high accuracy synchronization of time of day from a master ToD clock to a slave ToD clock. This synchronizer provides more user flexibility for your design.

The IEEE 1588v2 specifies multiple type of PTP devices, which include the following clocks:

- ordinary clock
- boundary clock
- transparent clock
- peer to peer transparent clock

Some of these PTP devices, boundary clock for example, consists of multiple ports that act as master or slave in the IEEE 1588v2 system. All these ports may share a common system clock or have its own individual clock. If every port has an individual ToD running on its own clock, then you must implement a method to instantiate one ToD clock as the master and the rest of the ToD clocks synchronized to this master ToD clock.

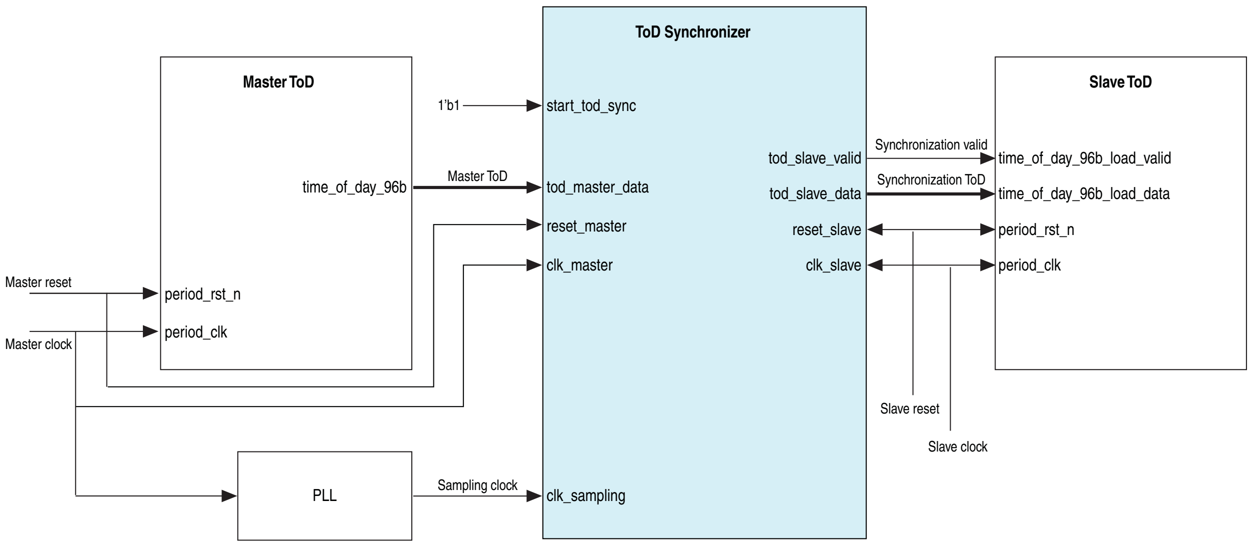
For this purpose, Altera provides the ToD synchronizer module. This module synchronizes a master ToD and a slave ToD in the following conditions:

- Master and slave ToD clock are in the same frequency (within 125 MHz to 156.25 MHz) but different phase.
- Master and slave ToD clock are in the same frequency (within 125 MHz to 156.25 MHz) but different PPM.
- Master and slave ToD clock are in different frequencies of either 125 MHz or 156.25 MHz.

# Block Diagram

Figure E–1 shows the connections between the ToD Synchronizer, master ToD, slave ToD, and sampling clock PLL.

**Figure E–1. Connection between ToD Synchronizer, Master ToD, Slave ToD, and Sampling Clock PLL**




The ToD Synchronizer block diagram comprises the following components::

- Master TOD clock domain—consists of three interfaces: `clk_master`, `reset_master`, and `tod_master_data`.
- Slave TOD clock domain—consists of five interfaces: `clk_slave`, `reset_slave`, `tod_slave_valid`, `tod_slave_data`, and `start_tod_sync`.
- Sampling clock PLL—consists of the `clk_domain` interface.

The Tod Synchronizer module synchronizes the master ToD clock domain with the slave ToD clock domain. The dual-clock FIFO in the Tod Synchronizer block takes in the time of day from the master ToD clock domain and transfers it to the slave ToD clock domain. The slave ToD then will load the synchronized time of day into its own internal counter, which then increments based on the new value.

As the ToD transfer is in progress, the master ToD domain keeps incrementing. When the ToD reaches the slave ToD clock domain and is ready to be loaded, it is much slower than the master ToD. To achieve high accuracy synchronization, the latency caused by the transfer must be reflected in the synchronized ToD.

The sampling clock PLL (`clk_sampling`) samples the FIFO fill level and calculates the latency through the FIFO. For better accuracy, the sampling clock must be derived from the master (`clk_master`) or slave (`clk_slave`) clock domain using a PLL.

If you use the recommended sampling clock frequency, the Tod Synchronizer module takes 64 clock cycles of sampling clock for every newly synchronized ToDto be valid at the output port.

Altera recommends that you use the following sampling clock frequencies:

- 1G master and slave—(64/63)*125MHz
- 10G master and slave—(64/63)*156.25MHz
- 1G master and 10G slave—(16/63)*125MHz or (64/315)*156.25MHz
- 10G master and 1G slave—(16/63)*125MHz or (64/315)*156.25MHz

Table E–1 shows the settings to achieve the recommended factors for Stratix V PLL.

**Table E–1. Settings to Achieve The Recommended Factors for Stratix V PLL**

| Settings | 64/63 | 16/63 | 64/315 |
|---|---|---|---|
| M-Counter | 64 | 16 | 64 |
| N-Counter | 21 | 03 | 21 |
| C-Counter | 03 | 21 | 15 |

# ToD Synchronizer Parameter Settings

Table E–2 describes the ToD Synchronizer configuration parameters.

**Table E–2. ToD Synchronizer Configuration Parameters**

| Name | Value | Description |
|---|---|---|
| `TOD_MODE` | Between 0 and 1 | Value that defines the time of day format that this block is synchronizing.<br>The default value is 1.<br>1: 96-bits format (32 bits seconds, 48 bits nanosecond and 16 bits fractional nanosecond)<br>0: 64-bits format (48 bits nanosecond and 16 bits fractional nanoseconds). |
| `SYNC_MODE` | Between 0 and 2 | Value that defines types of synchronization.<br>The default value is 1.<br>0: Master clock frequency is 125MHz (1G) while slave is 156.25MHz (10G).<br>1: Master clock frequency is 156.25MHz (10G) while slave is 125MHz (1G).<br>2: Master and slave are same in the same frequency; can be in different ppm or phase. When you select this mode, specify the period of master and slave through the `PERIOD_NSEC` and `PERIOD_FNSEC` parameters. |

**Table E–2. ToD Synchronizer Configuration Parameters**

| Name | Value | Description |
|---|---|---|
| PERIOD_NSEC | Between 0 and 4’hF | A 4-bit value that defines the reset value for a nanosecond of period.<br>The default value is 4'h6 to capture 6.4ns for 156.25MHz frequency. For 125MHz frequency (1G), set this parameter to 4'h8. |
| PERIOD_FNSEC | Between 0 and 16’hFFFF | A 4-bit value that defines the reset value for a fractional nanosecond of period.<br>The default value is 16'h6666 to capture 0.4ns of 6.4ns for 156.25MHz frequency. For 125MHz frequency (1G), set this parameter to 16'h0. |

# ToD Synchronizer Signals

## Common Clock and Reset Signals

Table E–1 describes the common clock and reset signals for the ToD Synchronizer.

**Table E–1. Clock and Reset Signals for the ToD Synchronizer**

| Signal | Direction | Width | Description |
|---|---|---|---|
| clk_master | Input | 1 | Clock from master ToD domain. |
| reset_master | Input | 1 | Reset signal that is synchronized to the master ToD clock domain. |
| clk_slave | Input | 1 | Clock from slave ToD domain. |
| reset_slave | Input | 1 | Reset signal that is synchronized to the slave ToD clock domain. |
| clk_sampling | Input | 1 | Sampling clock to measure the latency across the ToD Synchronizer. |

## Interface Signals

Table E–2 lists the interface signals for the ToD Synchronizer.

**Table E–2. Interface Signals for the ToD Synchronizer (Part 1 of 2)**

| Signal | Direction | Width | Description |
|---|---|---|---|
| start_tod_sync | Input | 1 | Assert this signal to start the ToD synchronization process. When this signal is asserted, the synchronization process continues and the time of day from the master ToD clock domain will be repeatedly synchronized with the slave ToD clock domain. |
| tod_master_data | Input | 1 | This signal carries the 64-bit or 96-bit format data for the time of day from the master ToD. The width of this signal is determined by the TOD_MODE parameter. |

**Table E–2. Interface Signals for the ToD Synchronizer (Part 2 of 2)**

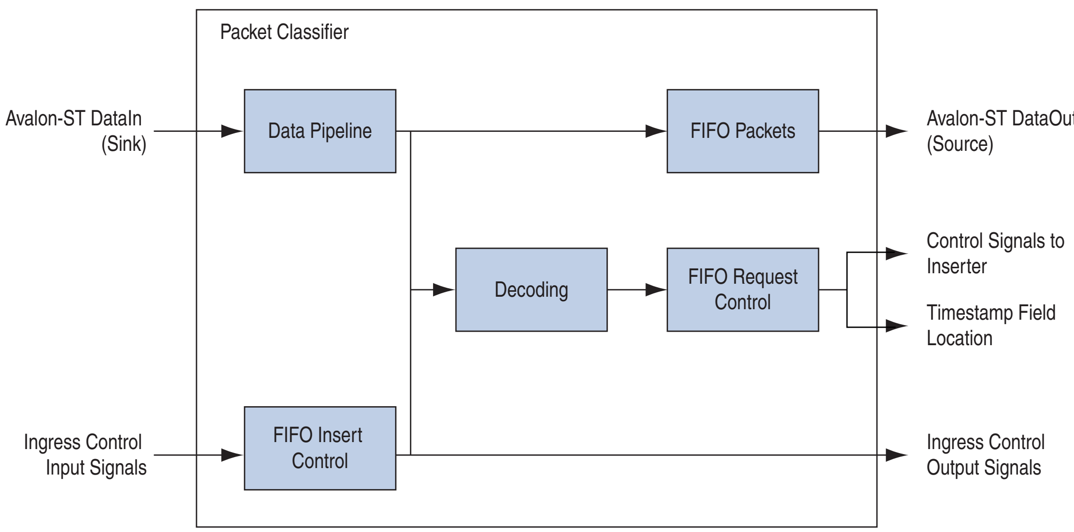
| Signal | Direction | Width | Description |
|---|---|---|---|
| tod_slave_valid | | | This signal indicates that the tod_data_slave signal is valid and ready to be loaded into the slave ToD clock in the following cycle.<br>This signal will only be high for 1 cycle every time a new time of day is successfully synchronized to the slave clock domain. |
| tod_slave_data[n-1:0] | Input | 1 | This signal carries the 64-bit or 96-bit format synchronized time of day that is ready to be loaded into the slave clock domain. The width of this signal is determined by the TOD_MODE parameter.<br>The synchronized time of day will be 1 slave clock period bigger than the master ToD because it takes 1 slave clock cycle to load this data into the slave ToD. |

The Packet Classifier decodes the packet types of incoming PTP packets and returns the decoded information aligned with SOP to the Triple-Speed Ethernet MAC with IEEE 1588v2 feature.

## Block Diagram

Figure F–1 shows the block diagram for the Packet Classifier.

**Figure F–1. Packet Classifier Block Diagram**




The Packet Classifier block diagram comprises the following components:

- Data Pipeline—holds the data frame up to a specified number of cycles. The number of cycles is determined by the largest length type field.
- FIFO Packets—holds the Avalon-ST frame data.
- FIFO Insert Control—the ingress control input bus that includes the signals required for decoding logics and signals to the MAC that is required to be aligned with SOP.
- FIFO Request Control—contains decoded data such as control signals to inserter and timestamp field locations.
- Decoding—Decodes packet types of incoming PTP packets and returns the decoded data to be stored in the FIFO request control block.

# Packet Classifier Signals

## Common Clock and Reset Signals

Table F–1 describes the common clock and reset signals for the Packet Classifier.

**Table F–1. Clock and Reset Signals for the Packet Classifier**

| Signal | Direction | Width | Description |
|---|---|---|---|
| clk | Input | 1 | 156.25-MHz register access reference clock. |
| reset | Input | 1 | Assert this signal to reset the clock. |

## Avalon-ST Interface Signals

Table F–2 lists the Avalon-ST DataIn (sink) interface signals for the Packet Classifier.

**Table F–2. Avalon-ST DataIn Interface Signals for the Packet Classifier**

| Signal | Direction | Width | Description |
|---|---|---|---|
| data_sink_sop | Input | 1 | The Avalon-ST input frames. |
| data_sink_eop | Input | 1 | |
| data_sink_valid | Input | 1 | |
| data_sink_ready | Output | 1 | |
| data_sink_data | Input | 64 | |
| data_sink_empty | Input | 3 | |
| data_sink_error | Input | 1 | |

Table F–3 lists the Avalon-ST DataOut (source) interface signals for the Packet Classifier.

**Table F–3. Avalon-ST DataOut Interface Signals for the Packet Classifier**

| Signal | Direction | Width | Description |
|---|---|---|---|
| data_src_sop | Input | 1 | The Avalon-ST output frames. |
| data_src_eop | Input | 1 | |
| data_src_valid | Input | 1 | |
| data_src_ready | Output | 1 | |
| data_src_data | Input | 64 | |
| data_src_empty | Input | 3 | |
| data_src_error | Input | 1 | |

## Ingress Control Signals

Table F–4 describes the ingress control signals for the Packet Classifier.

**Table F–4. Ingress Control Signals for the Packet Classifier (Part 1 of 2)**

| Signal | Direction | Width | Description |
|---|---|---|---|
| tx_etstamp_ins_ctrl_in_ingress_timestamp_96b | Input | 96 | 96-bit format of ingress timestamp that holds data so that the output can align with the start of an incoming packet. |
| tx_etstamp_ins_ctrl_in_ingress_timestamp_64b | Input | 64 | 64-bit format of ingress timestamp that holds data so that the output can align with the start of an incoming packet. |
| tx_etstamp_ins_ctrl_out_ingress_timestamp_96b | Output | 96 | 96-bit format of ingress timestamp that holds data so that the output can align with the start of an outgoing packet. |
| tx_etstamp_ins_ctrl_out_ingress_timestamp_64b | Output | 64 | 64-bit format of ingress timestamp that holds data so that the output can align with the start of an outgoing packet. |
| tx_egress_timestamp_request_in_valid | Input | 1 | Assert this signal when timestamp is required for the particular frame. This signal must be aligned to the start of an incoming packet. |
| tx_egress_timestamp_request_in_fingerprint | Input | 4 | A width-configurable fingerprint that correlates timestamps for incoming packets. |
| tx_egress_timestamp_request_out_valid | Output | 1 | Assert this signal when timestamp is required for the particular frame. This signal must be aligned to the start of an outgoing packet. |
| tx_egress_timestamp_request_out_fingerprint | Output | 4 | A width-configurable fingerprint that correlates timestamps for outgoing packets. |
| clock mode | Input | 2 | Determines the clock mode.<br>00: Ordinary clock<br>01: Boundary clock<br>10: End to end transparent clock<br>11: Peer to peer transparent clock |
| pkt_with_crc | Input | 1 | Indicates whether or not a packet contains CRC.<br>1: Packet contains CRC<br>0: Packet does not contain CRC |
| tx_etstamp_ins_ctrl_in_residence_time_update | Input | 1 | Indicates the update for residence time.<br>1: Allows update for residence time based on decoded results.<br>0: Prevents update for residence time. When this signal is deasserted, tx_etstamp_ins_ctrl_out_residence_time_update also gets deasserted. |

**Table F–4. Ingress Control Signals for the Packet Classifier (Part 2 of 2)**

| Signal | Direction | Width | Description |
|---|---|---|---|
| tx_etstamp_ins_ctrl_in_residence_time_calc_format | Input | 1 | Format of the timestamp to be used for calculating residence time. This signal must be aligned to the start of an incoming packet.<br>1: 64-bit timestamp format<br>0: 96-bit timestamp format |
| tx_etstamp_ins_ctrl_out_residence_time_calc_format | Output | 1 | Format of the timestamp to be used for calculating residence time. This signal must be aligned to the start of an outgoing packet.<br>1: 64-bit timestamp format<br>0: 96-bit timestamp format |

## Control Insert Signals

Table F–5 describes the control insert signals for the Packet Classifier. These signals must be aligned to the start of a packet.

**Table F–5. Control Insert Signals for the Packet Classifier**

| Signal | Direction | Width | Description |
|---|---|---|---|
| tx_etstamp_ins_ctrl_out_checksum_zero | Output | 1 | Assert this signal to set the checksum field. |
| tx_etstamp_ins_ctrl_out_checksum_correct | Output | 1 | Assert this signal to correct the packet checksum by updating the checksum correction specified by tx_etstamp_ins_ctrl_out_offset_checksum_correction. |
| tx_etstamp_ins_ctrl_out_timestamp_format | Output | 1 | The timestamp format of the frame where the timestamp is inserted. |
| tx_etstamp_ins_ctrl_out_timestamp_insert | Output | 1 | Assert this signal to insert timestamp into the associated frame. |
| tx_etstamp_ins_ctrl_out_residence_time_update | Output | 1 | Assert this signal to add the residence time into the correction field of the PTP frame. |

## Timestamp Field Location Signals

Table F–6 describes the timestamp field location signals for the Packet Classifier. These signals must be aligned to the start of a packet.

**Table F–6. Timestamp Field Location Signals for the Packet Classifier**

| Signal | Direction | Width | Description |
|---|---|---|---|
| tx_etstamp_ins_ctrl_out_offset_timestamp | Output | 16 | Indicates the location of the timestamp field. |
| tx_etstamp_ins_ctrl_out_offset_correction_field | Output | 16 | Indicates the location of the correction field. |
| tx_etstamp_ins_ctrl_out_offset_checksum_field | Output | 16 | Indicates the location of the checksum field. |
| tx_etstamp_ins_ctrl_out_offset_checksum_correction | Output | 16 | Indicates the location of the checksum corrector field. |

This chapter provides additional information about the document and Altera.

## Document Revision History

The following table lists the revision history for this document.

| Date | Version | Changes |
|---|---|---|
| May 2013 | 13.0 | ■ Updated the MegaWizard Plug-In Manager flow in Chapter 2, Getting Started with Altera IP Cores.<br>■ Added information about generating a design example and simulation testbench in the "MegaWizard Plug-In Manager Flow" on page 2–3.<br>■ Updated the list of Quartus II generated files in Table 2–2.<br>■ Added information about the recommended pin assignments Table 2–3 (design constraint file is no longer generated).<br>■ Updated the MegaCore parameter names and description in Chapter 3, Parameter Settings<br>■ Updated the IEEE 1588v2 feature list in Chapter 4, Functional Description.<br>■ Updated the SGMII auto-negotiation description in Chapter 4, Functional Description.<br>■ Added information about the IEEE 1588v2 feature PMA delay in Table 6–8.<br>■ Updated the Multiport Ethernet MAC with IEEE 1588v2, 1000BASE-X/SGMII PCS and Embedded PMA Signals in Figure 7–6.<br>■ Updated the IEEE 1588v2 timestamp signal names in Chapter 7, Interface Signals.<br>■ Added timing diagrams for IEEE 1588v2 timestamp signals in Chapter 7, Interface Signals.<br>■ Added a section about migrating existing design to the Quartus II software new TSE user interface in Chapter 8, Design Considerations.<br>■ Updated Chapter 9, Timing Constraints, to describe the new timing constraint files and the recommended clock input frequency for each MegaCore Function variant.<br>■ Added information about the simulation model files generated using IEEE simulation encryption in "Simulation Model Files" on page 10–6<br>■ Updated the jumbo frames file directory in Table 11–2.<br>■ Updated the ToD configuration parameters in Table D–3 and ToD interface signals in Figure D–1, Table D–5 and Table D–6..<br>■ Added information to describe the ToD's drift adjustment in "Adjusting Time-of-Day Clock Drift" on page D–5.<br>■ Added Appendix E, ToD Synchronizer and Appendix F, Packet Classifier.<br>■ Removed SOPC Builder information. |

| Date | Version | Changes |
|---|---|---|
| January 2013 | 12.1 | ■ Added the Altera IEEE 1588v2 Feature section in Chapter 4.<br>■ Added information for the following GUI parameters: **Enable timestamping**, **Enable PTP 1-step clock**, and **Timestamp fingerprint width** in "Timestamp Options" on page 3–4.<br>■ Added MAC registers with IEEE 1588v2 feature in Table 6–7.<br>■ Added IEEE 1588v2 feature signals in Table 7–24 through Table 7–28.<br>■ Added Chapter 5, Triple-Speed Ethernet with IEEE 1588v2 Design Example.<br>■ Added Appendix D, Time-of-Day Clock. |
| June 2012 | 12.0 | ■ Added support for Cyclone V.<br>■ Updated the Congestion and Flow Control section in Chapter 4.<br>■ Added Register Initialization section in Chapter 5.<br>■ Added `holdoff_quant` register description in Table 5-2.<br>■ Added `UNIDIRECTIONAL_ENABLE` bit description in Table 5-9.<br>■ Revised and moved the section on Timing Constraint to a new chapter (Chapter 8).<br>■ Added information about how to customize the SDC file in Chapter 8.<br>■ Added Pause Frame Generation section in Appendix B. |
| November 2011 | 11.1 | ■ Added support for Arria V.<br>■ Revised the Device Family Support section in Chapter 1.<br>■ Added `disable_read_timeout` and `read_timeout` registers at address 0x15 and 0x16 in Table 6-2. |
| June 2011 | 11.0 | ■ Updated support for Cyclone IV GX, Cyclone III LS, Aria II GZ, HardCopy IV GX/E and HardCopy III E devices.<br>■ Revised Performance and Resource Utilization section in Chapter 1.<br>■ Updated Chapter 3 to include Qsys System Integration Tool Design Flow.<br>■ Added Transmit and Receive Latencies section in Chapter 4.<br>■ Updated all MAC register address to dbyte addressing. |
| December 2010 | 10.1 | ■ Added support for Arria II GZ<br>■ Added a new parameter, Starting Channel Number.<br>■ Streamlined the contents and document organization. |
| August 2010 | 10.0 | ■ Added support for Stratix V.<br>■ Revised the nomenclature of device support types.<br>■ Added chapter 5, Design Considerations. Moved the Clock Distribution section to this chapter and renamed it to Optimizing Clock Resources in Multiport MAC and PCS with Embedded PMA. Added sections on PLL Sharing and Transceiver Quad Sharing.<br>■ Updated the description of Enable transceiver dynamic reconfiguration. |
| November 2009 | 9.1 | ■ Added support for Cyclone IV, Hardcopy III, and Hardcopy IV, and updated support for Hardcopy II to full.<br>■ Updated chapter 1 to include a feature comparison between 10/100/1000 Ethernet MAC and small MAC.<br>■ Updated chapter 4 to revise the 10/100/1000 Ethernet MAC description, Length checking, Reset, and Control Interface sections. |

| Date | Version | Changes |
|---|---|---|
| March 2009 | 9.0 | ■ Added support for Arria II GX.<br>■ Updated chapter 3 to include a new parameter that enables wider statistics counters.<br>■ Updated chapter 4 to reflect support for different speed in multiport MACs and gated clocks elimination.<br>■ Updated chapter 6 to reflect enhancements made on the device drivers. |
| November 2008 | 8.1 | ■ Updated Chapters 3 and 4 to add description on dynamic reconfiguration.<br>■ Updated Chapter 6 to include a procedure to add unsupported PHYs. |
| May 2008 | 8.0 | ■ Revised the performance tables and device support.<br>■ Updated Chapters 3 and 4 to include information on MAC with multi ports and without internal FIFOs.<br>■ Revised the clock distribution section in Chapter 4.<br>■ Reorganized Chapter 5 to remove redundant information and to include the new testbench architecture.<br>■ Updated Chapter 6 to include new public APIs. |
| October 2007 | 7.2 | ■ Updated Chapter 1 to reflect new device support.<br>■ Updated Chapters 3 and 4 to include information on Small MAC. |
| May 2007 | 7.1 | ■ Added Chapters 2, 3, 5 and 6.<br>■ Updated contents to reflect changes and enhancements in the current version. |
| March 2007 | 7.0 | Updated signal names and description. |
| December 2006 | 6.1 | ■ Global terminology changes: 1000BASE-X PCS/SGMII to 1000BASE-X/SGMII PCS, host side or client side to internal system side, HD to half-duplex.<br>■ Initial release of document on Web. |
| December 2006 | 6.1 | Initial release of document on DVD. |

# How to Contact Altera

For the most up-to-date information about Altera® products, see the following table.

| Contact *(1)* | Contact Method | Address |
|---|---|---|
| Technical support | Website | www.altera.com/support |
| Technical training | Website | www.altera.com/training |
| | Email | custrain@altera.com |
| Altera literature services | Email | literature@altera.com |
| Non-technical support (General) | Email | nacomp@altera.com |
| (Software Licensing) | Email | authorization@altera.com |

**Note:**

(1) You can also contact your local Altera sales office or sales representative.

# Typographic Conventions

The following table shows the typographic conventions that this document uses.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Indicates command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. |
| **bold type** | Indicates directory names, project names, disk drive names, file names, file name extensions, and software utility names. For example, **\qdesigns** directory, **d:** drive, and **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Indicates document titles. For example, *AN 519: Stratix IV Design Guidelines.* |
| *Italic type* | Indicates variables. For example, *n* + 1.<br>Variable names are enclosed in angle brackets (< >). For example, <*file name*> and <*project name*>**.pof** file. |
| Initial Capital Letters | Indicates keyboard keys and menu names. For example, Delete key and the Options menu. |
| "Subheading Title" | Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions." |
| `Courier type` | Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. Active-low signals are denoted by suffix `n`. For example, `resetn`.<br>Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`.<br>Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`). |
| 1., 2., 3., and<br>a., b., c., and so on. | Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| | Bullets indicate a list of items when the sequence of the items is not important. |
| | The hand points to information that requires special attention. |
| CAUTION | A caution calls attention to a condition or possible situation that can damage or destroy the product or your work. |
| WARNING | A warning calls attention to a condition or possible situation that can cause you injury. |
| | The angled arrow instructs you to press Enter. |
| | The feet direct you to more information about a particular topic. |
| | The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents. |
| | The feedback icon allows you to submit feedback to Altera about the document. Methods for collecting feedback vary as appropriate for each document. |