10-Gbps Ethernet MAC MegaCore Function

User Guide

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

UG-01083-3.0

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

Feedback Subscribe

Contents

Chapter 1. About This IP Core

Chapter 2. Getting Started with Altera IP Cores

Chapter 3. Design Examples and Testbench

Chapter 4. Functional Description

Chapter 5. Registers

Chapter 6. Interface Signals

Chapter 7. Design Considerations

Appendix A. Frame Format

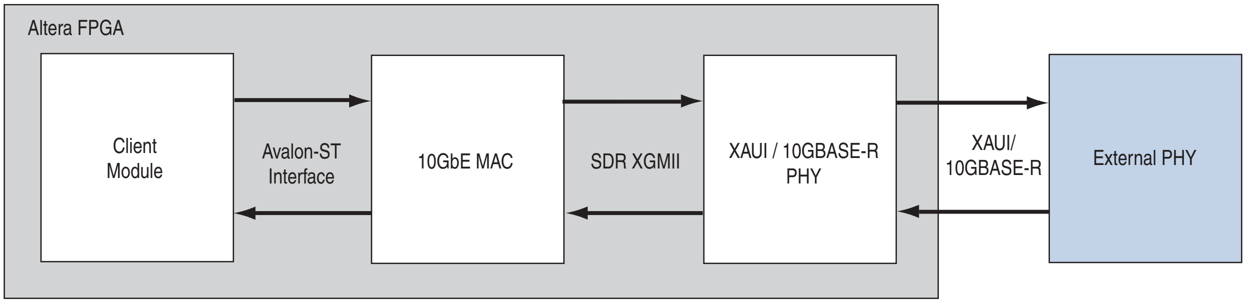
Additional Information

1. About This IP Core

The 10-Gbps Ethernet (10GbE) Media Access Controller (MAC) IP core is a configurable component that implements the IEEE 802.3-2005 specification. The IP core uses the Avalon® Streaming (Avalon-ST) interface on the client side and the single data rate (SDR) XGMII on the network side. To build a complete 10GbE subsystem in an Altera® device and connect it to an external device, you can use the 10GbE MAC IP core with an Altera PHY IP core such as a soft XAUI PHY in FPGA fabric, hard silicon-integrated XAUI PHY, or a 10GBASE-R PHY.

Figure 1–1 illustrates a system with the 10GbE MAC IP core.

Figure 1–1. Typical Application of 10GbE MAC



1.1. Features

The 10GbE MAC supports the following features:

- Avalon-ST 64-bit wide client interface running at 156.25 MHz with 10-Gbps full-duplex line rate.
- Direct interface to 64-bit SDR XGMII running at 156.25 MHZ.
- Virtual local area network (VLAN) and stacked VLAN tagged frames filtering as specified by IEEE 802.IQ and 802.1ad (Q-in-Q) standards respectively.
- Optional cyclic redundancy code (CRC)-32 computation and insertion on the transmit datapath; CRC checking on the receive datapath with optional forwarding of the frame check sequence (FCS) field to the client application.
- Checking of receive frames for FCS error, undersized and oversized frames, and payload length error.
- Deficit idle counter (DIC) for optimized performance with average inter-packet gap (IPG) for LAN applications.
- Optional statistics collection on the transmit and receive datapaths.
- Packets termination when the transmit datapath receives incomplete packets.
- Programmable maximum length of transmit and receive frames up to 64 Kbytes (KB).
- Programmable promiscuous (transparent) mode.

- Optional Ethernet flow control and priority-based flow control (PFC) using pause frames with programmable pause quanta. The PFC supports up to 8 priority queues.
- Optional padding termination on the receive datapath and insertion on the transmit datapath.
- Design examples with optional loopback and testbench for design verification.
- Optional preamble passthrough mode on the transmit and receive datapaths. The preamble passthrough mode allows you to define the preamble in the client frame.

1.2. Programmable datapath option to allow separate instantiation of MAC Tx block, MAC Rx block, or both MAC Tx and MAC Rx blocks.Device Family Support

MegaCore functions provide the following support for Altera device families:

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

Table 1–1 shows the level of support offered by the 10GbE MAC to each Altera device family.

Table 1–1. Device Family Support

Device Family	Support
Arria® GX	Final
Arria II GX	Final
Arria II GZ	Final
Arria V GT	Preliminary
Cyclone® IV GX	Final
HardCopy® IV	Companion
Stratix® II GX	Final
Stratix III	Final
Stratix IV	Final

Table 1–1. Device Family Support

Device Family	Support
Stratix V	Preliminary
Other device families	No support

1.3. IP Core Verification

To ensure compliance with the IEEE specification, Altera performs extensive validation of the 10GbE MAC IP core. Validation includes both simulation and hardware testing.

1.3.1. Simulation Environment

Altera performs the following tests in the simulation environment:

- Directed tests that test all types and sizes of transaction layer packets and all bits of the configuration space.
- Error injection tests that inject errors in the link, transaction layer packets, and data link layer packets, and check for the proper response from the IP core.
- Random tests that test a wide range of traffic patterns across one or more virtual channels.

1.3.2. Compatibility Testing Environment

Altera has performed significant hardware testing of the 10GbE MAC IP core to ensure a reliable solution. The IP core has been tested with Arria GX, Arria II GX, Cyclone IV GX, Stratix II GX, Stratix III, Stratix IV, Stratix IV GX, and Stratix V devices and soft XAUI PHYs. They have passed all interoperability tests with a wide selection of motherboards and test equipment. In addition, Altera internally tests every release with motherboards from a variety of manufacturers.

1.4. Performance and Resource Utilization

Table 1–2 provides the estimated performance and resource utilization of the 10GbE MAC for the Cyclone IV device family. The estimates are obtained by compiling the 10GbE MAC with the Quartus II software targeting a Cyclone IV (EP4CGX110DF31C7) device with speed grade –7.

To achieve your timing requirement in Quartus II, Altera recommends that you use multiple seeds in the Design Space Explorer to find the optimal Fitter settings for your design, follow the Timing Optimization Advisor's recommendations, apply the Speed Optimization Technique and use the LogicLock regions.

Table 1–2. Cyclone IV Performance and Resource Utilization

Settings	Logic Elements	Registers	Memory Block (M9K)	f_{MAX} (MHz)
All options disabled	4,424	3,245	2	>156.25
All options enabled with memory-based statistics counters	11,845	8,355	11	>156.25

Table 1–3 provides the estimated performance and resource utilization of the 10GbE MAC for the Stratix IV device family. The estimates are obtained by compiling the 10GbE MAC with the Quartus II software targeting a Stratix IV GX (EP4SGX70HF35C2) device with speed grade –2.

Table 1–3. Stratix IV Performance and Resource Utilization

Settings	Combinational ALUTs	Logic Registers	Memory Block (M9K)	f_{MAX} (MHz)
All options disabled	1,954	3,157	0	>156.25
All options enabled with memory-based statistics counters	5,684	8,349	7	>156.25
All options enabled with register-based statistics counters	8,135	10,117	3	>156.25

Table 1–4 provides the estimated performance and resource utilization of the 10GbE MAC for the Stratix V device family. The estimates are obtained by compiling the 10GbE MAC with the Quartus II software targeting a Stratix V GX (5SGXEA7H3F35C3) device with speed grade –3.

Table 1–4. Stratix V Performance and Resource Utilization

Settings	Combinational ALUTs	Dedicated Logic Registers	Memory Block (M20K)	f_{MAX} (MHz)
All options disabled	2,001	3,077	0	>156.25
All options enabled with memory-based statistics counters	5,772	8,197	7	>156.25
All options enabled with register-based statistics counters	8,202	9,965	3	>156.25

2. Getting Started with Altera IP Cores

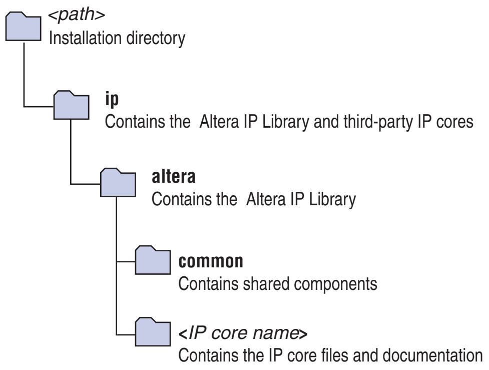
This chapter provides a general overview of the Altera IP core design flow to help you quickly get started with any Altera IP core. The Altera IP Library is installed as part of the Quartus II installation process. You can select and parameterize any Altera IP core from the library. Altera provides an 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.

2.1. 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


<path>
Installation directory
ip
Contains the Altera IP Library and third-party IP cores
altera
Contains the Altera IP Library
common
Contains shared components
<IP core name>
Contains the IP core files and documentation

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*.

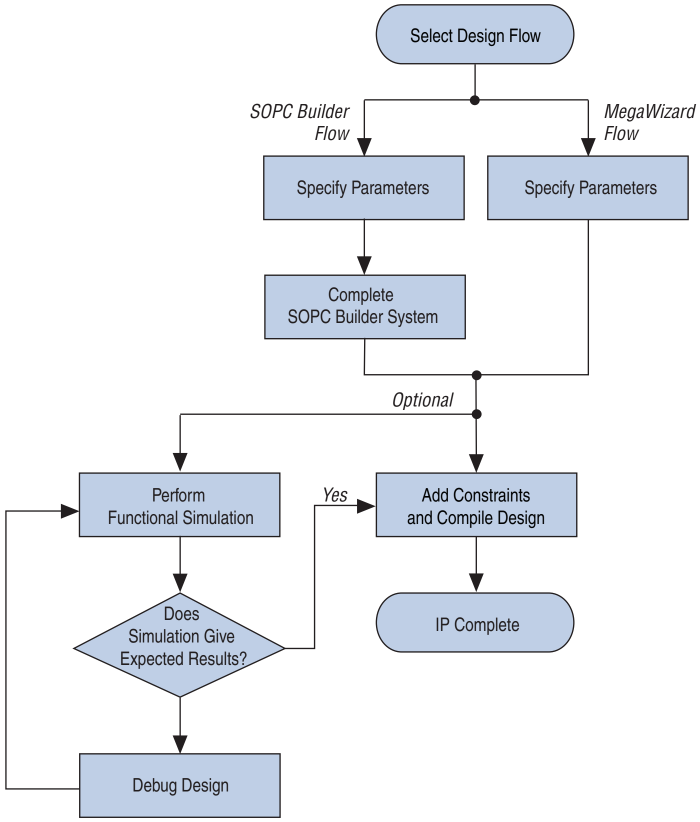
2.2. Design Flows

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

- MegaWizard Plug-In Manager Flow

Figure 2–2 shows the design flows.

Figure 2–2. Design Flows


Select Design Flow
SOPC Builder Flow
MegaWizard Flow
Specify Parameters
Specify Parameters
Complete SOPC Builder System
Optional
Perform Functional Simulation
Yes
Add Constraints and Compile Design
Does Simulation Give Expected Results?
IP Complete
Debug Design

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

- Allows you to parameterize an IP core variant and instantiate into an existing design

2.3. MegaWizard Plug-In Manager Flow

The MegaWizard Plug-In Manager flow allows you to customize the 10GbE MAC IP core and manually integrate the function into your design.

2.3.1. Specifying Parameters

To specify the 10GbE MAC IP core parameters with the MegaWizard Plug-In Manager, follow these steps:

1. Open an existing Quartus II project or create a new project using the **New Project Wizard** available from the File menu.

2. In the Quartus II software, launch the **MegaWizard Plug-in Manager** from the Tools menu, and follow the prompts in the MegaWizard Plug-In Manager interface to create or edit a custom IP core variation.
3. In the **Installed Plug-Ins** list on page 2a of the MegaWizard Plug-In Manager interface, expand the **Interfaces** folder and then the **Ethernet** folder. Select **Ethernet 10G MAC**. Specify the type and name of the output file you want to create.
4. Specify the parameters on the **Parameter Settings** pages. For detailed explanations of these parameters, refer to "10GbE MAC Parameter Settings" on page 2–8.
5. Specify appropriate options in the wizard to generate a simulation model.

   ☞ Altera IP supports a variety of simulation models, including simulation-specific IP functional simulation models and encrypted RTL models. These are all cycle-accurate models. The models allow for fast functional simulation of your IP core instance using industry-standard VHDL or Verilog HDL simulators.

   For more information about functional simulation models for 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.

6. Click **Finish**. The parameter editor generates the top-level HDL code for the 10GbE MAC IP core and a simulation directory which includes files for simulation.

   ☞ The **Finish** button may be unavailable until all parameterization errors listed in the messages window are corrected.

7. Upon clicking **Finish**, a dialog box appears to allow you to skip the example design file generation. The default setting is to generate the example design which creates a `list_of_files.tcl` file in the ***<design_name>*_sim** project directory. If you turn off **Generate Example Design**, the Quartus II skips the example design generation stage and does not create the `list_of_files.tcl` file.

   ☞ For the 10GbE MAC IP core, no example design is generated regardless of whether the **Generate Example Design** check box is turned on or off. You can leverage on the design example system provided in the Qsys component library.

8. Click **Yes** if you are prompted to add the Quartus II IP File (**.qip**) to the current Quartus II project. You can also turn on **Automatically add Quartus II IP Files to all projects**.

You can now integrate your custom 10GbE MAC IP core instance in your design, simulate, and compile.

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

2.3.2. Simulate the IP Core

You can simulate the 10GbE MAC IP core with the functional simulation model generated by the Quartus II software. To perform a successful simulation of the 10GbE MAC IP core using the MegaWizard Plug-In Manager flow, you are required to compile all files listed in the <*project directory*>/<*variation name*>**_sim** output file. Otherwise, the simulation may fail.

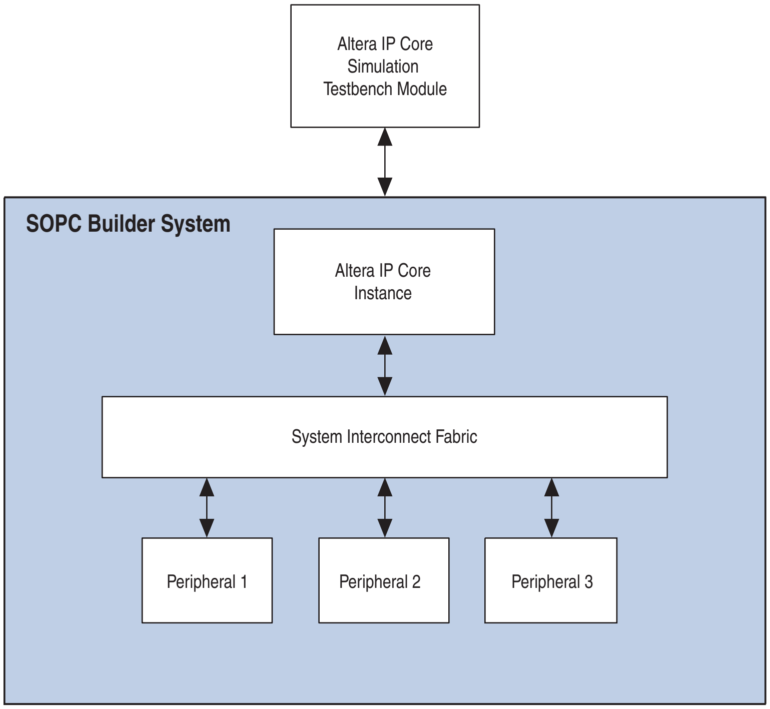
For more information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

2.4. SOPC Builder Design Flow

You can use SOPC Builder to build a system that includes your customized IP core. You easily can add other components and quickly create an SOPC Builder system. SOPC Builder automatically generates HDL files that include all of the specified components and interconnections. SOPC Builder defines default connections, which you can modify. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device.

Figure 2–3 shows a block diagram of an example SOPC Builder system.

Figure 2–3. SOPC Builder System



For more information about system interconnect fabric, refer to the *System Interconnect Fabric for Memory-Mapped Interfaces* and *System Interconnect Fabric for Streaming Interfaces* chapters in the *SOPC Builder User Guide* and to the *Avalon Interface Specifications*.

For more information about SOPC Builder and the Quartus II software, refer to the *SOPC Builder Features* and *Building Systems with SOPC Builder* sections in the *SOPC Builder User Guide* and to Quartus II Help.

2.4.1. Specify Parameters

To specify the 10GbE MAC IP core parameters in the SOPC Builder flow, follow these steps:

1. Open an existing Quartus II project or create a new project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **SOPC Builder**.
3. For a new system, specify the system name and language.
4. On the **System Contents** tab, expand the **Interfaces Protocols** list and then the **Ethernet** list. Double-click **Ethernet 10G MAC** to add it to your system. The relevant parameter editor appears.
5. Specify the required parameters in the parameter editor. For detailed explanations of these parameters, refer to "10GbE MAC Parameter Settings" on page 2–8.
6. Click **Finish** to complete the IP core instance and add it to the system.

2.4.2. Complete the SOPC Builder System

To complete the SOPC Builder system, follow these steps:

1. Add and parameterize any additional components. Some IP cores include a complete SOPC Builder system design example.
2. Use the Connection panel on the **System Contents** tab to connect the components.
3. By default, clock names are not displayed. To display clock names in the **Module Name** column and the clocks in the **Clock** column in the **System Contents** tab, click **Filters** to display the **Filters** dialog box. In the **Filter** list, click **All.**
4. If you intend to simulate your SOPC builder system, on the **System Generation** tab, turn on **Simulation. Create project simulator files** to generate simulation files for your system.
5. Click **Generate** to generate the system. SOPC Builder 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.
6. In the Quartus II software, click **Add/Remove Files in Project** on the Project menu and add the **.qip** file to the project.
7. Compile your design in the Quartus II software.

2.4.3. Simulate the System

During system generation, you can specify whether SOPC Builder generates a simulation model and testbench for the entire system, which you can use to easily simulate your system in any of Altera's supported simulation tools. SOPC Builder also generates a set of ModelSim® Tcl scripts and macros that you can use to compile the testbench and plain-text RTL design files that describe your system in the ModelSim simulation software.

For information about the latest Altera-supported simulation tools, refer to the *Quartus II Software Release Notes*.

For information about simulating SOPC Builder systems, refer to the *SOPC Builder User Guide and AN 351: Simulating Nios II Embedded Processor Designs*.

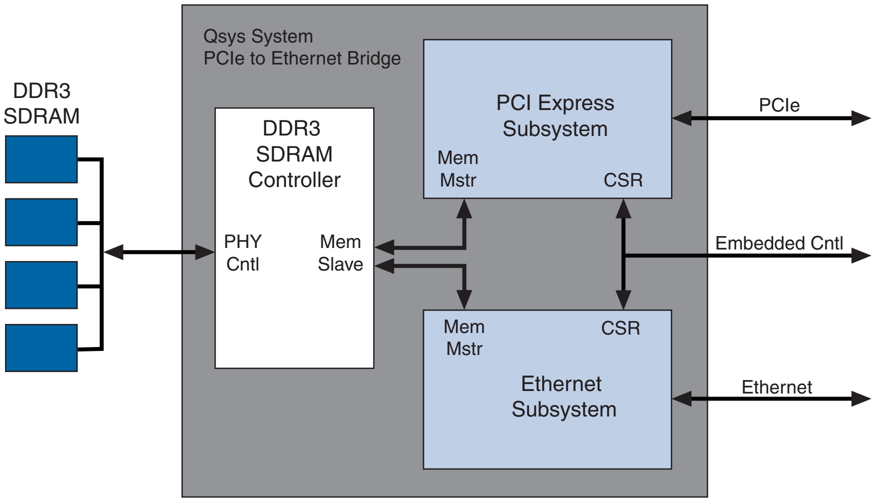
For general information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

2.5. 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.

Figure 2–4 shows a high level block diagram of an example Qsys system.

Figure 2–4. 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.

2.5.1. Specify Parameters

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

1. Open an existing Quartus II project or create a new 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 **Interfaces Protocols** list and then the **Ethernet** list. Double-click **Ethernet 10G MAC** to add it to your system. The relevant parameter editor appears.
4. Specify the required parameters in the Qsys tool. For detailed explanations of these parameters, refer to “10GbE MAC Parameter Settings” on page 2–8.
5. Click **Finish** to complete the IP core instance and add it to the system.

2.5.2. 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 Connections panel on the **System Contents** tab.
3. In the **Export As** 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, turn on one or more options under **Simulation** to generate desired simulation files.
5. If you want to generate synthesis RTL files, turn on **Create HDL design files for synthesis**.
6. 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.
7. In the Quartus II software, click **Add/Remove Files in Project** on the Project menu and add the **.qip** file to the project.
8. Compile your design in the Quartus II software.

2.5.3. Simulate the System

During system generation, Qsys generates a functional simulation model which you can use to simulate your system easily in any Altera-supported simulation tool.

For information about the latest Altera-supported simulation tools, refer to the *Quartus II Software Release Notes*.

For general information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

For information about simulating Qsys systems, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook*.

2.6. 10GbE MAC Parameter Settings

You customize the 10GbE MAC by specifying the parameters on the MegaWizard Plug-in Manager, SOPC Builder, or Qsys in the Quartus II software. Table 2–1 describes the parameters and how they affect the behavior of the IP core.

Table 2–1. 10GbE Parameters

Parameter	Description
Preamble Passthrough Mode	Turn on this parameter to include the logic to implement the preamble passthrough mode. To use the preamble passthrough mode, you must turn on this parameter and set the `tx_preamble_control`, `rx_lane_decoder_preamble_control`, and `rx_preamble_inserter_control` registers to 1 to enable the mode.
Priority-based Flow Control (PFC)	Turn on this parameter to include the logic to implement PFC. Refer to "Priority-Based Flow Control" on page 4–16 for more information on PFC and its operations.

Table 2–1. 10GbE Parameters

Parameter	Description
Number of PFC Priorities	Indicates the number of PFC priority levels that the 10GbE MAC IP core supports. The valid range is from 2 to 8. This option is enabled only if you turn on the **Priority-based Flow Control (PFC)** parameter.
Datapath Option	Use this parameter to select the datapath option that determines the MAC variation to instantiate. By default, the **TX & RX** option is selected. The default datapath instantiates the MAC Tx and MAC Rx blocks. Selecting **TX Only** instantiates the MAC Tx block; selecting **RX Only** instantiates the MAC Rx block.
Supplementary Address	Turn on this parameter to include the logic to implement supplementary addresses. To use supplementary addresses, you must turn on this parameter and enable supplementary addresses by setting the `EN_SUPP0/1/2/3` bits in the `rx_frame_control` register to 1.
CRC on Transmit Path	Turn on this parameter to include the logic to calculate and insert CRC on the transmit datapath. To compute and insert CRC on the transmit datapath, you must turn on this parameter and enable CRC insertion by setting the `tx_crcins_control`[1] register bit to 1.
Statistics Collection	Turn on this parameter to include the logic that collects statistics on the transmit and receive datapaths.
Statistics Counters	When you turn on **Statistics Collection**, the default implementation of the statistics counters is **Memory-based**. Use **Memory-based** statistics counters to free up the logic elements (the MAC does not clear the statistic counters after the counters are read); **Register-based** statistics counters to free up the memory (the MAC clears the statistic counters after the counters are read). Register-based statistics counters are not supported for Cyclone IV GX and Arria GX devices.

3. Design Examples and Testbench

Altera provides the following design examples and testbenches to help you get started with the 10GbE MAC IP core and use the core in your design:

- 10GbE MAC with XAUI PHY
- 10GbE MAC with 10GBASE-R PHY

This chapter describes the design examples, their architecture and components, and functional verification using the provided testbench and in the hardware.

☞ XAUI PHY and 10GBASE-R PHY do not support older devices such as Arria GX, Stratix II, and Stratix III.

3.1. Software and Hardware Requirements

Altera uses the following hardware and software to test the design examples and testbenches:

- Quartus II software 12.0
- Stratix IV GX FPGA development kit (for XAUI PHY)
- Transceiver Signal Integrity development kit, Stratix IV GT Edition (for 10GBASE-R PHY)
- ModelSim®-AE 6.6c, ModelSim-SE 6.6c or higher

For more information on the development kits, refer to the following documents:

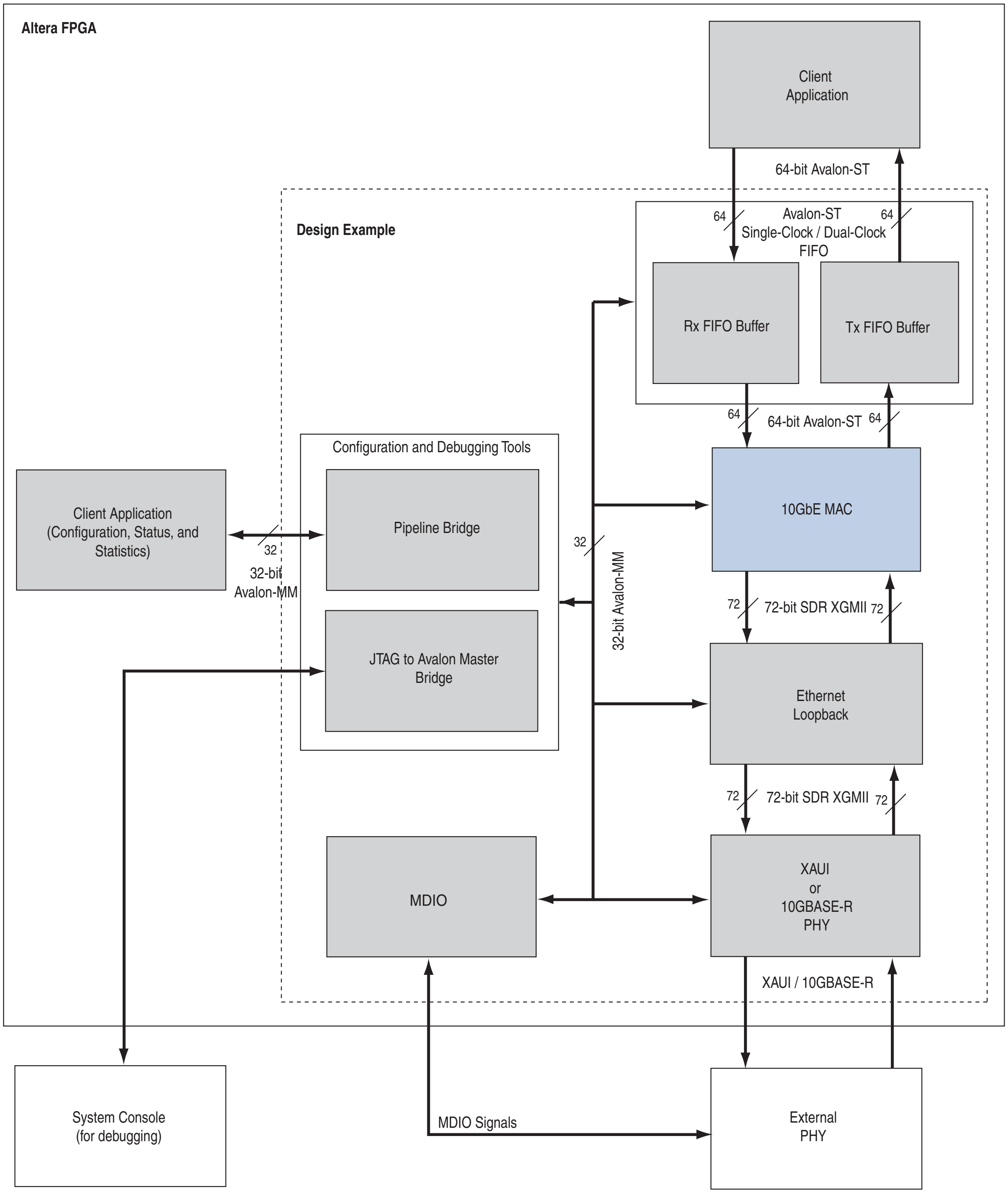
- *Stratix IV GX Development Kit User Guide*
- *Stratix IV GX Development Kit Reference Manual*
- *Transceiver Signal Integrity Development Kit, Stratix IV GT Edition User Guide*
- *Transceiver Signal Integrity Development Kit, Stratix IV GT Edition Reference Manual*

3.2. Design Example

You can use the 10GbE MAC IP core design example to simulate a complete 10GbE design in an Altera FPGA. You can compile the design example using the simulation files generated by the Quartus II software and program the targeted Altera device after a successful compilation.

Figure 3–1 shows the architecture of the design examples.

Figure 3–1. Design Example Architecture



3.2.1. Components

The design example comprises the following components:

- 10GbE Ethernet MAC—the MAC IP core with default settings. This IP core includes memory-based statistics counters.
- XAUI PHY or 10GBASE-R PHY—the PHY IP core with default settings. The XAUI PHY is set to **Hard XAUI** by default.
- Ethernet Loopback— the loopback module, which is enabled by default, provides a mechanism for you to verify the functionality of the MAC and PHY. Refer to Section 3.2.1.1, Ethernet Loopback Module for more information about this module.
- Rx and Tx FIFO buffers—Avalon-ST Single-Clock or Dual-Clock FIFO cores that buffer receive and transmit data between the MAC and client. These FIFO buffers are 64 bits wide and 512 bits deep. The default configuration is Avalon-ST Single-Clock FIFO which operates in store and forward mode and can be configured to provide packet-based flushing capabilities when an error occurs.
- Configuration and debugging tools—provides access to the registers of the following components via the Avalon Memory-Mapped (Avalon-MM) interface: MAC, MDIO, Ethernet loopback, PHY, and FIFO buffers. The provided testbench includes an Avalon driver which uses the pipeline bridge to access the registers. You can use the system console to access the registers via the JTAG to Avalon Master Bridge core when verifying the design in the hardware.

To learn more about the components, refer to the respective documents:

- XAUI PHY and 10GBASE-R PHY, refer to *Altera Transceiver PHY IP Core User Guide*.
- Avalon-ST Single-Clock or Dual-Clock FIFO, JTAG to Avalon Master Bridge, and MDIO cores, refer to *Embedded Peripherals IP User Guide*.
- Pipeline bridge, refer to *Avalon Memory-Mapped Bridges* in volume 4 of the *Quartus II Handbook*.
- System Console, refer to *Analyzing and Debugging Designs with the System Console* in volume 3 of the *Quartus II Handbook*.

3.2.1.1. Ethernet Loopback Module

You can enable one the following loopback types:

- Local loopback—turn on this loopback to verify the functionality of the MAC during simulation. When you enable the local loopback, the Ethernet loopback module takes the transmit frame from the MAC Tx and loops it back to the receive datapath. During this cycle, the loopback module also forwards the frame to the PHY. While the local loopback is turned on, the loopback module ignores any frame it receives from the PHY.

- Line loopback—turn on this loopback to verify the functionality of the PHY when verifying the design example in hardware. When you enable the line loopback, the Ethernet loopback module takes the XGMII signal received from the external PHY and loops it back to the transmit datapath. During this cycle, the loopback module also forwards the XGMII signal to the MAC. While the line loopback is turned on, the loopback module ignores any frame it receives from the MAC.

Table 3–1 describes the registers you can use to enable or disable the desired loopback.

Table 3–1. Loopback Registers

Byte Offset	Register	Description
0x00	`line loopback`	Set this register to 1 to enable line loopback; 0 to disable it.
0x04	Reserved	—
0x08	`local loopback`	Set this register to 1 to enable local loopback; 0 to disable it.

3.2.1.2. Base Addresses

Table 3–2 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. Refer to Table 3–1 for the Ethernet loopback registers.

Table 3–2. Base Addresses of Design Example Components

Component	Base Address
10GbE MAC	0x000
XAUI or 10GBASE-R PHY	0x40000
MDIO	0X10000
Ethernet loopback	0x10200
Rx FIFO (Avalon-ST Single-Clock FIFO)	0x10400
Tx FIFO (Avalon-ST Single-Clock FIFO)	0x10600

3.2.2. Files

Figure 3–2 shows the directory structure for the design examples and testbenches. The **..\csr_script** directory contains the design example script files.

Figure 3–2. Design Example Folders

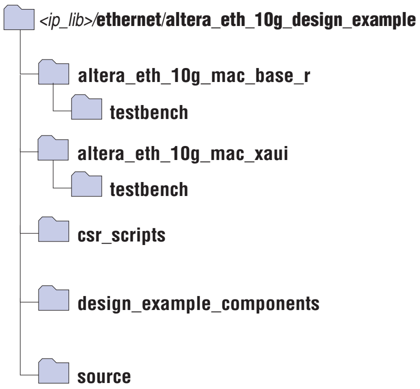

<ip_lib>/ethernet/altera_eth_10g_design_example
altera_eth_10g_mac_base_r
testbench
altera_eth_10g_mac_xaui
testbench
csr_scripts
design_example_components
source

Table 3–3 lists the design example files. For the description of testbench files, refer to Table 3–5 on page 3–9.

Table 3–3. Design Example Files

File Name	Description
setup_proj.tcl	A Tcl script that creates a new Quartus II project and sets up the project environment for your design example.
altera_eth_10g_design_mac_xaui.qsys	A Qsys file for the 10GbE MAC and XAUI PHY design example. The PHY is set to hard XAUI by default.
altera_eth_10g_design_mac_xaui_sv.qsys	A Qsys file for the 10GbE MAC and XAUI PHY design example with the Quartus II software targeting the Stratix V device. The PHY is set to hard XAUI by default.
altera_eth_10g_design_mac_base_r.qsys	A Qsys file for the 10GbE MAC and 10GBASE-R PHY design example.
altera_eth_10g_design_mac_base_r_sv.qsys	A Qsys file for the 10GbE MAC and 10GBASE-R PHY design example with the Quartus II software targeting the Stratix V device.
setup_SIVGX230C2ES.tcl	A Tcl script that sets the pin assignments and I/O standards for the Stratix IV GX FPGA development board. Use this Tcl script for the 10GbE MAC with XAUI PHY design example.
setup_EP4S100G5H40I3.tcl	A Tcl script that sets the pin assignments and I/O standards for the Stratix IV GT Signal Integrity development board. Use this Tcl script for the 10GbE MAC with 10GBASE-R PHY design example.
top.sdc	The Quartus II SDC constraint file for use with the TimeQuest timing analyzer.
top.v	The top-level entity file of the design example for verification in hardware.
top_sv.v	The top-level entity file of the design example—with the Quartus II software targeting the Stratix V device—for verification in hardware.
common.tcl	A Tcl script that contains basic functions based on the system console APIs to access the registers through the Avalon-MM interface.
config.tcl	A Tcl script that configures the design example.
csr_pkg.tcl	A Tcl script that maps address to the Avalon-MM control registers. The script contains APIs which is used by **config.tcl** and **show_stats.tcl**.
show_stats.tcl	A Tcl script that displays the MAC statistics counters.
altera_eth_10g_design_example_hw.tcl	A hardware Tcl script that contains the composition of the Ethernet system.

3.2.3. Creating a New 10GbE Design

You can use the Quartus II software to create a new 10GbE design. Altera provides a customizable Qsys design example file to facilitate the development of your 10GbE design. Follow these steps to create the design:

1. Copy the respective design example directory to your preferred project directory: **altera_eth_10g_mac_xaui** or **altera_eth_10g_mac_base_r** from *<ip library>*/**ethernet**/**altera_eth_10g_design_example**.
2. Launch the Quartus II software and open the **top.v** file from the project directory.
3. Open the Quartus II Tcl Console window by pointing to **Utility Windows** on the View menu and then selecting **Tcl Console**. In the Quartus II Tcl Console window, type the following command to set up the project environment:

   ```
   source setup_proj.tcl↵
   ```

4. Load the pin assignments and I/O standards for the development board:
   - For the 10GbE MAC with XAUI PHY design example, type the following command:

     ```
     source setup_SIVGX230C2ES.tcl↵
     ```

     This command assigns the XAUI serial interface to the pins that are connected to the HSMC Port A of the Stratix IV GX development board.
   - For the 10GbE MAC with 10GBASE-R design example, type the following command:

     ```
     source setup_EP4S100G2F40I1.tcl↵
     ```

     This command assigns the 10GBASE-R serial interface to the pins that are connected to the SMA connectors (J38 to J41) of the Stratix IV GT development board.

   For more information about the development boards, refer to the respective reference manuals: *Stratix IV GX Development Kit Reference Manual* or *Transceiver Signal Integrity Development kit, Stratix IV GT Edition Reference Manual*.

5. Launch Qsys from the Tools menu and open the **altera_eth_10g_mac_base_r.qsys** or **altera_eth_10g_mac_xaui.qsys** file. For design targeting the Stratix V device family, use the **altera_eth_10g_mac_base_r_sv.qsys** or **altera_eth_10g_mac_xaui_sv.qsys** file.

   ☞ By default, the design example targets the Stratix IV device family. To change the target device family, click on the **Project Settings** tab and select the desired device from the **Device family** list.

6. Turn off the additional module under the **Use** column if your design does not require them. This action disconnects the module from the 10GbE system.
7. Double-click **eth_10g_design_example_0** to launch the parameter editor.
8. Specify the required parameters in the parameter editor. For detailed explanations of these parameters, refer to "Parameter Settings" on page 3–7.

9. Click **Finish**.
10. On the **Generation** tab, select either a verilog or a VHDL simulation model and make sure that the **Create HDL design files for synthesis** option is turned on.
11. Click **Generate** to generate the simulation and synthesis files.

3.2.4. Parameter Settings

You can customize the 10GbE design example by specifying the parameters using the parameter editor. Table 3–4 describes the these parameters.

Table 3–4. Design Example Parameters

Name	Value	Description
Configuration		
MDIO	**MDIO** **None**	Specifies whether the Ethernet system requires a MDIO core to access the external PHY device management registers for configuration and management purposes.
PHY IP	**XAUI PHY** **10GBase-R PHY** **None**	Specifies which protocol-specific PHY IP core to use for the Ethernet system. For XAUI PHY, you can choose to implement the system in soft or hard logic.
FIFO	**Avalon-ST Single Clock FIFO** **Avalon-ST Dual Clock FIFO** **Avalon-ST Single Clock FIFO + Avalon-ST Dual Clock FIFO** **None**	Specifies which FIFO buffer to use for the Ethernet system. The Avalon-ST Single Clock FIFO operates with a common clock for the input and output ports while the Avalon-ST Dual Clock FIFO operates with independent clocks for the input and output ports. You cannot enable a different FIFO option for Tx datapath and Rx datapath. If you select **Avalon-ST Single Clock FIFO**, the design includes single clock FIFO at both the Tx and Rx datapath.

The parameter values you select on the configuration tab corresponds with other tabs that requires further parameterization. You should only parameterize the components that you select and omit the others. Editing the unselected component parameters may cause the system generation to fail.

For more information about the parameter settings of other components, refer to the respective documents:

- 10GbE MAC, refer to "10GbE MAC Parameter Settings" on page 2–8.
- Avalon-ST Single-Clock or Dual-Clock FIFO and MDIO core, refer to *Embedded Peripherals IP User Guide*.
- XAUI PHY and 10GBASE-R PHY, refer to *Altera Transceiver PHY IP Core User Guide*.

3.3. Testbenches

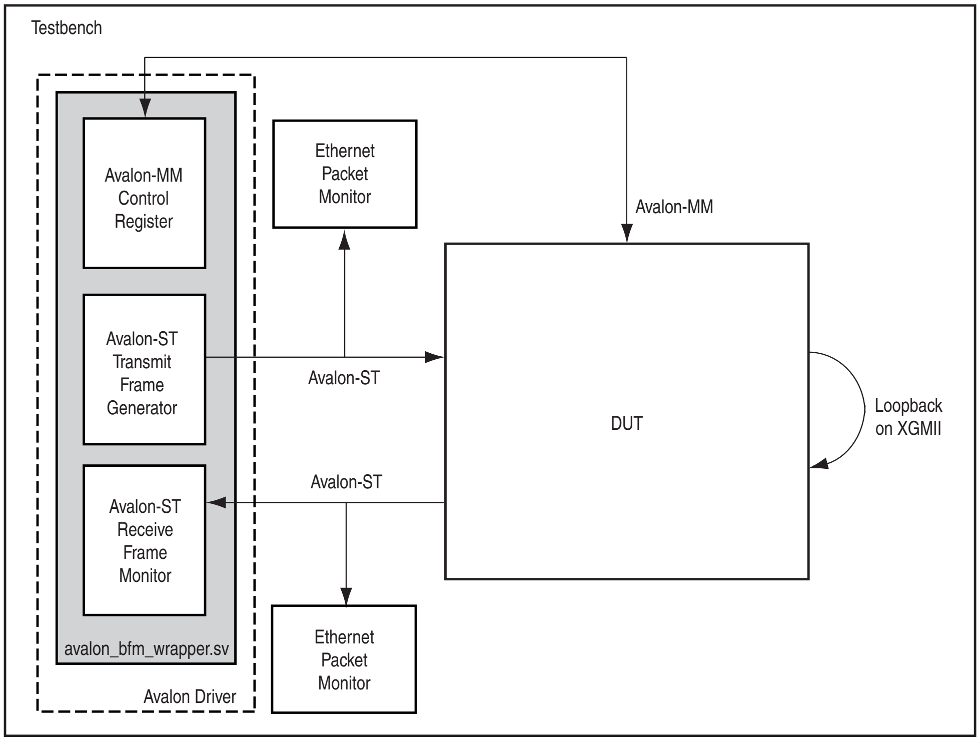
Altera provides testbenches for you to verify the design examples. The following sections in this document describe the architecture of the testbenches, their components, and use.

3.3.1. Architecture

The testbenches operate in loopback mode. Frames sent through the transmit path loops back into the receive path.

Figure 3–3 illustrates the architecture of the testbenches.

Figure 3–3. Testbench Architecture



3.3.2. Components

The testbenches comprise the following modules:

- Device under test (DUT)—the design example. Refer to Figure 3–1 on page 3–2 for the architecture of the design example.
- Avalon driver—uses Avalon-ST bus functional models (BFMs) to exercise the transmit and receive paths. The driver also utilizes the 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.

3.3.3. Files

The following directories contain the testbench files which are in clear text:

- 10GbE MAC and XAUI PHY testbench—<*ip library*>/**ethernet/altera_eth_10g_design_example/altera_eth_10g_mac_xaui/testbench**
- 10GbE MAC and 10GBASE-R PHY testbench— <*ip library*>/**ethernet/altera_eth_10g_design_example/altera_eth_10g_mac_base_r/testbench**

Table 3–5 describes the files that implement the testbenches.

Table 3–5. 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.
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.
tb_run.tcl	A Tcl script that starts a simulation session in the ModelSim simulation software.
tb.sv	The top-level testbench file. This file includes the customized 10GbE 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.

3.3.4. Simulation Flow

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

1. Initializes the DUT by configuring the following options via the Avalon-MM interface:
   a. In the MAC, enables address insertion on the transmit path and sets the transmit primary MAC address to EE-CC-88-CC-AA-EE.
   b. In the Tx and Rx FIFO buffers (Avalon-ST Single Clock FIFO core), enables drop on error.
2. Starts packet transmission. The testbench sends a total of eight packets:
   a. 64-byte basic Ethernet frame
   b. Pause frame
   c. 1518-byte VLAN frame
   d. 1518-byte basic Ethernet frame
   e. 64-byte stacked VLAN frame
   f. 500-byte VLAN frame
   g. Pause frame
   h. 1518-byte stacked VLAN frame
3. Ends the transmission and displays the MAC statistics in the transcript pane.

3.3.5. Simulating the Testbench with the 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: **altera_eth_10g_mac_xaui** or **altera_eth_10g_mac_base_r** from <*ip library*>/**ethernet**/**altera_eth_10g_design_example**.
2. The design example and testbench files are set to read only. Altera recommends that you turn off the read-only attribute of all design example and testbench files.
3. Launch the Quartus II software and open the **top.v** file from the project directory.
4. Open the Quartus II Tcl Console window by pointing to **Utility Windows** on the View menu and then selecting **Tcl Console**. In the Quartus II Tcl Console window, type the following command to set up the project environment:

   ```
   source setup_proj.tcl↵
   ```
5. Launch Qsys from the Tools menu and open **altera_eth_10g_mac_base_r.qsys** or **altera_eth_10g_mac_xaui.qsys** in the File menu.
6. For the 10GbE MAC with XAUI design example, the default setting of the XAUI PHY is **Hard XAUI**. Follow these steps if you want to set the PHY to **Soft XAUI**:
   a. Double-click the XAUI PHY module to open the parameter editor.
   b. On the **General Options** tab, select **Soft XAUI** for **XAUI Interface Type**.
7. On the **Generation** tab, select verilog simulation model.

8. Click **Generate** to generate the system.Launch the ModelSim simulator software.
9. Change the working directory to *<project directory>/<design example directory>*/**testbench** in the **File** menu.
10. Run the following command to set up the required libraries, compile the generated IP Functional simulation model, and exercise the simulation model with the provided testbench:

    ```
    do tb_run.tcl↵
    ```

    The ModelSim transcript pane in Main window displays messages from the testbench reflecting the current task being performed.

Upon a successful simulation, the simulator displays the following `RX Statistics` and `TX Statistics`:

```
# framesOK = 8
# framesErr = 0
# framesCRCErr = 0
# octetsOK = 5138
# pauseMACCtrlFrames = 2
# ifErrors = 0
# unicastFramesOK = 4
# unicastFramesErr = 0
# multicastFramesOK = 1
# multicastFramesErr = 0
# broadcastFramesOK = 1
# broadcastFramesErr = 0
# etherStatsOctets = 5310
# etherStatsPkts = 8
# etherStatsUndersizePkts = 0
# etherStatsOversizePkts = 0
# etherStatsPkts64Octets = 4
# etherStatsPkts65to127Octets = 0
# etherStatsPkts128to255Octets = 0
# etherStatsPkts256to511Octet = 1
# etherStatsPkts512to1023Octets = 0
# etherStatsPkts1024to1518Octets = 3
# etherStatsPkts1519OtoXOctets = 0
# etherStatsFragments = 0
# etherStatsJabbers = 0
# etherStatsCRCErr = 0
# unicastMACCtrlFrames = 1
# multicastMACCtrlFrames = 1
# broadcastMACCtrlFrames = 0
```

3.3.6. Enabling Local Loopback

You can turn on local loopback to verify the functionality of the MAC during simulation. Follow these steps to enable local loopback:

1. Open the **tb.sv** file.
2. Insert the command `U_AVALON_DRIVER.avalon_mm_csr_wr(offset,value)` where offset is the sum of the base address of the loopback module and the register offset, and value is the value to write to the register. Set value to 1 to enable local loopback; 0 to disable it. Altera recommends that you insert the command after the command that configures the Rx FIFO. For example, the following code segment enables local loopback:

```
// Configure the RX FIFO
U_AVALON_DRIVER.avalon_mm_csr_wr(RX_FIFO_DROP_ON_ERROR_ADDR,RX_FIFO_DROP_ON_ERROR);

// Read the configured registers
U_AVALON_DRIVER.avalon_mm_csr_rd(RX_FIFO_DROP_ON_ERROR_ADDR, readdata);
$display("RX FIFO Drop on Error Enable      = %0d", readdata[0]);

U_AVALON_DRIVER.avalon_mm_csr_wr(32'h948, 1)
```

3. Run the following command again to reconfigure the loopback module, set up the required libraries, compile the generated IP Functional simulation model, and exercise the simulation model:

   `do tb_run.tcl`↵

3.3.7. Simulation Timing Diagrams

Figure 3–4 shows the reset and initial configuration sequence. The first read or write transaction must be at least one clock cycle after the `csr_reset_reset_n` signal completes.

Figure 3–4. Reset and Configuration

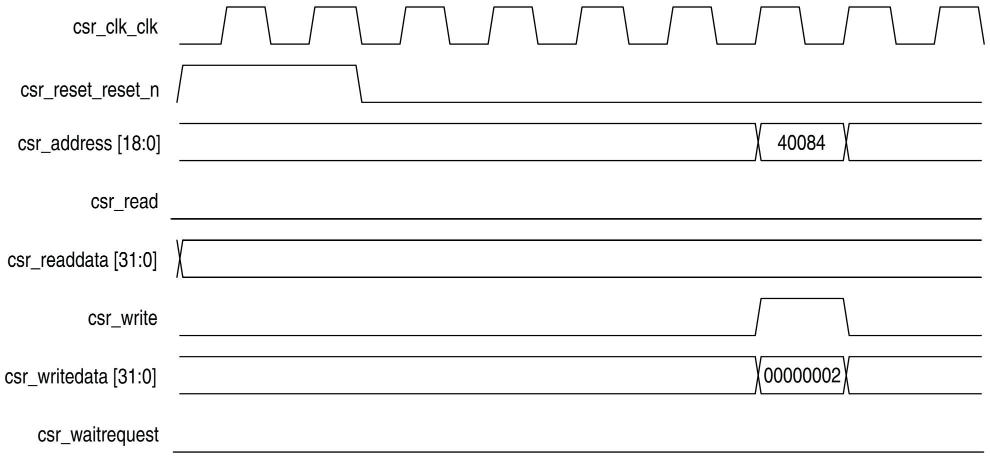
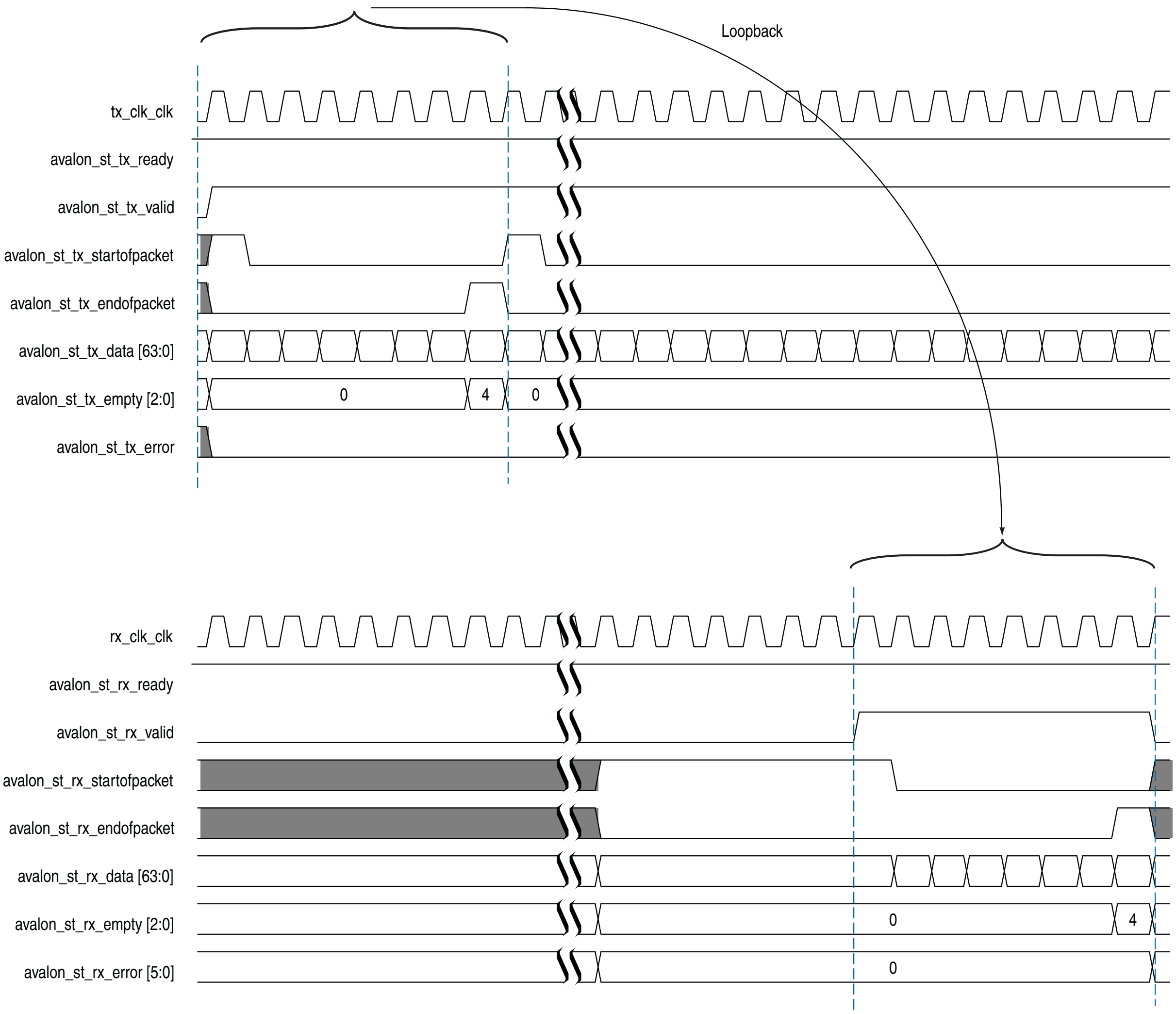


Figure 3–5 shows the transmission of the first 60-byte frame upon a successful reset and initial configuration. The same frame is looped back to the receive datapath.

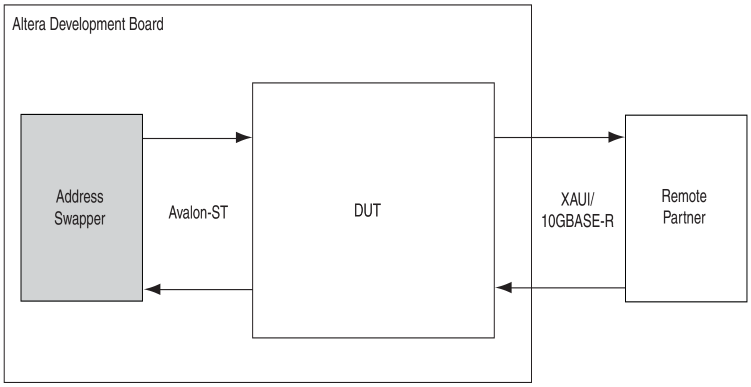
Figure 3–5. Frame Transmission and Reception



3.4. Design Example Compilation and Verification in Hardware

Figure 3–6 shows the components in the top-level file provided with the design example.

Figure 3–6. Top-Level Components



The address swapper swaps the destination address and source address in the receive frame before sending the frame onto the transmit path. You must connect the DUT—design example—to a remote partner that generates, transmits, and receives frames.

3.4.1. Compiling the Design

You can use the Quartus II software to compile the design example and program the targeted Altera device after a successful compilation.

Follow these steps to compile the design and program the device:

1. Copy the respective design example directory to your preferred project directory: **altera_eth_10g_mac_xaui** or **altera_eth_10g_mac_base_r** from <*ip library*>/**ethernet**/**altera_eth_10g_design_example**.
2. Launch the Quartus II software and open **top.v** from the project directory.
3. Open the Quartus II Tcl Console window by pointing to **Utility Windows** on the View menu then clicking **Tcl Console**. In the Quartus II Tcl Console window, type the following command to set up the project environment:

```
source setup_proj.tcl↵
```

4. Load the pin assignments and I/O standards for the development board:
   - For the 10GbE MAC with XAUI PHY design example, type the following command:

     ```
     source setup_SIVGX230C2ES.tcl↵
     ```

     This command assigns the XAUI serial interface to the pins that are connected to the HSMC Port A of the Stratix IV GX development board.
   - For the 10GbE MAC with 10BASE-R design example, type the following command:

     ```
     source setup_EP4S100G2F40I1.tcl↵
     ```

     This command assigns the 10GBASE-R serial interface to the pins that are connected to the SMA connectors (J38 to J41) of the Stratix IV GT development board.

   For more information about the development boards, refer to the respective reference manuals: *Stratix IV GX Development Kit Reference Manual* or *Transceiver Signal Integrity Development kit, Stratix IV GT Edition Reference Manual*.

5. Launch Qsys from the Tools menu and open **altera_eth_10g_mac_base_r.qsys** or **altera_eth_10g_mac_xaui.qsys**.
6. For the 10GbE MAC with XAUI PHY design example, the default setting of the PHY is **Hard XAUI**. Follow these steps if you want to set the PHY to **Soft XAUI**:

   a. Double-click the XAUI PHY module to open the parameter editor.

   b. On the **General Options** tab, select **Soft XAUI** for **XAUI Interface Type**.
7. Click **Save** on the File menu.
8. On the **Generation** tab, turn on **Create Synthesis RTL Files**.
9. Click **Generate** to generate the system.
10. Click **Start Compilation** on the Processing menu to compile the design example.
11. Upon a successful compilation, click **Programmer** on the Tools menu to program the device.

For more information about device programming, refer to *Quartus II Programmer* in volume 3 of the *Quartus II Handbook*.

☞ If you are not using the Stratix IV GX FPGA development board or the Transceiver Signal Integrity development board, Stratix IV GT Edition, modify **setup_proj.tcl** and **setup_SIVGX230C2ES.tcl** or **setup_EP4S100G2F40I1.tcl** to suit your hardware.

3.4.2. Verifying the Design in Hardware

After programming the targeted Altera device, follow these steps to verify your design and collect the statistics:

1. Copy the **csr_scripts** directory to the design example directory.
2. Launch Qsys and access the **System Console** by clicking **System Console** on the Tools menu.
3. Change the working directory to <*project directory*>/**csr_scripts**.
4. Type the following command to configure the design example:

   ```
   source config.tcl↵
   ```

5. Start frame transmission on your remote partner to exercise the datapaths.
6. Type the following command to read and view the statistics:

   ```
   source show_stats.tcl↵
   ```

☞ The `config.tcl` and `show_stats.tcl` scripts support only one USB-Blaster connection.

3.4.3. Debugging

You can use the system console to perform the following tasks for debugging purposes:

- Reconfigure the design example components and retrieve the registers during runtime by following these steps:

  a. Create a new Tcl script.

  b. Add the following commands:

  ```
  source common.tcl

  # establishes the connection
  open_jtag

  # use rd32 to retrieve the register value
  # base address = base address of the component
  # offset = byte offset of the register
  rd32 <base address> 0 <offset>

  # use wr32 to configure the register
  # base address = base address of the component
  # offset = byte offset of the register
  # value = value to be written to the register
  wr32 <base address> 0 <offset> <value>

  # closes the connection
  close_jtag
  ```

  Save and close the Tcl script and type the following command:

  ```
  source <script>.tcl↵
  ```

- Retrieve and view the statistics counters by typing the following command:

  ```
  source show_stats.tcl↵
  ```

- Turn on the line loopback to verify the functionality of the XAUI/10GBASE-R PHY by following these steps:

  a. Edit the script **config.tcl**.

  b. Add the command `write_line_loopback(value)` immediately after the command that establishes the JTAG connection. Set the argument value, to 1 to enable line loopback; 0 to disable line loopback. For example, the following codes enable line loopback:

  ```
  open_jtag
  write_line_loopback 1
  ```

  c. Save and close **config.tcl**, and type the following command:

  ```
  source config.tcl↵
  ```

For more information on the System Console, refer to *Analyzing and Debugging Designs with the System Console* in volume 3 of the *Quartus II Handbook*.

3.4.4. 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 byte on the network-side interface (XGMII SDR) 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 byte on the Avalon-ST interface after the bit was received on the network-side interface (XGMII SDR).

Table 3–6 shows the transmit and receive nominal latencies of the design example.

Table 3–6. Transmit and Receive Latencies of the Design Example

Configuration	Latency (Clock Cycles) *(1) (2)*	
	Transmit (with respect to Tx clock)	Receive (with respect to Rx clock)
MAC and Ethernet loopback	10	13
DC FIFO	6	6
SC FIFO	10	10
Soft XAUI PHY	41 *(3)*	
Hard XAUI PHY	24 *(3)*	
Soft 10GBASE-R PHY	56 *(3)*	

Notes to Table 3–6:

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

(2) The latency values are based on the assumption that there is no backpressure on the Avalon-ST Tx and Rx interface.

(3) Total latency for both transmit and receive in this design example targeting the Stratix IV device family.

3.4.5. Performance and Resource Utilization

Table 3–7 provides the estimated performance and resource utilization of the design example obtained by compiling the design with the Quartus II software targeting the Stratix IV GX (EP4SGX230KF40C2ES) device with speed grade –2.

Table 3–7. Stratix IV Performance and Resource Utilization

Components	Combinational ALUTs	Memory ALUTs	Logic Registers	Memory Block (M9K)	f_{MAX} (MHz)
MAC	4,105	17	4,642	8	≥156.25
Loopback	272	0	175	4	≥156.25
Rx SC FIFO	231	0	210	5	≥156.25
Tx SC FIFO	213	0	210	4	≥156.25
Hard XAUI PHY	1,837	0	1,124	0	≥156.25
MDIO	120	0	133	0	≥156.25
JTAG Master	492	0	433	1	≥156.25
Address Swapper	66	0	71	0	≥156.25
FIFO to Pause Adapter	2	0	4	0	≥156.25
Qsys Fabric	422	0	634	1	≥156.25
Total Resource Utilization	7,760	17	7,636	23	≥156.25

4. Functional Description

The 10GbE MAC IP core handles the flow of data between a client and Ethernet network through a 10-Gbps Ethernet PHY. On the transmit path, the MAC accepts client frames and constructs Ethernet frames by inserting various control fields, such as checksums before forwarding them to the PHY. Similarly, on the receive path, the MAC accepts Ethernet frames via a PHY, performs checks, and removes the relevant fields before forwarding the frames to the client. You can configure the MAC to collect statistics on both transmit and receive paths.

This chapter describes the 10GbE MAC IP core, its architecture, interfaces, data paths, registers, and interface signals.

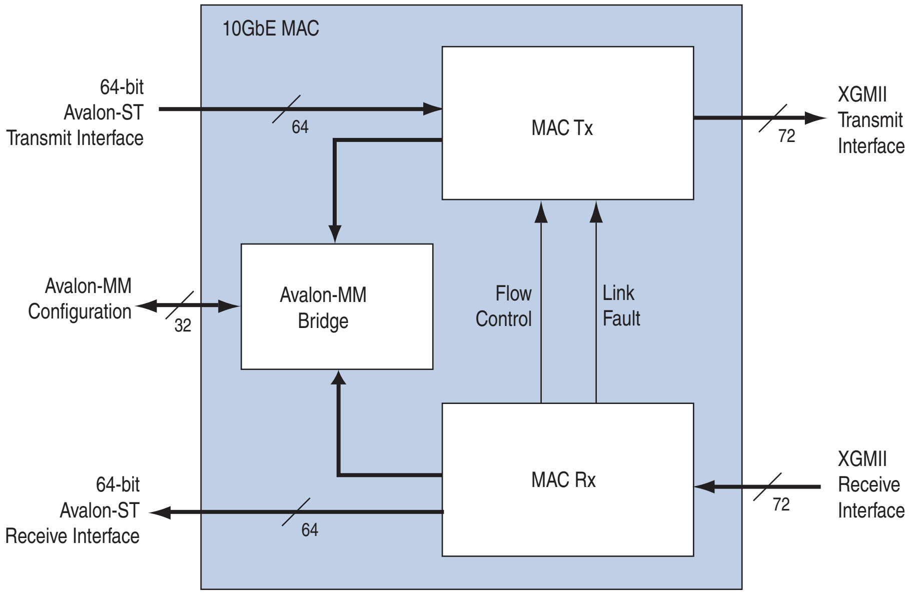
4.1. Architecture

The 10GbE MAC IP core is a composition of three blocks: MAC receiver (MAC Rx), MAC transmitter (MAC Tx), and Avalon-MM bridge. The MAC Rx and MAC Tx handle data flow between the client and Ethernet network. Each of these blocks include a 64-bit wide Avalon-ST interface on the client side and a single data rate (SDR) XGMII on the network side.

The Avalon-MM bridge provides a single interface to all Avalon-MM interfaces within the MAC, which allows a host to access 32-bit configuration and status registers, and statistics counters.

Figure 4–1 shows a block diagram of the 10GbE MAC IP core.

Figure 4–1. 10GbE MAC IP Core Block Diagram



4.2. Interfaces

The 10GbE MAC includes the following interfaces:

- Avalon-ST transmit and receive interface on the client side
- SDR XGMII transmit and receive interface on the network side
- Avalon-MM control and status register interface

4.2.1. Avalon-ST Interface

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

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

In the MAC, the Avalon-ST interface acts as a sink in the transmit datapath and source in the receive datapath. These interfaces are 64 bits wide and support packets, backpressure, and error. The ready latency on these interfaces is 0 and the MAC expects the `empty` signal to contain a valid value.

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

4.2.2. SDR XGMII

The network-side interface of the MAC implements the SDR version of the XGMII protocol. The SDR XGMII consists of 64-bit data bus and 8-bit control bus operating at 156.25 MHz. The data bus carries the MAC frame; the most significant byte occupies the least significant lane.

4.2.3. Avalon-MM Control and Status Register Interface

The Avalon-MM control and status register interface is an Avalon-MM slave port. This interface uses byte addressing which provides host access to 32-bit configuration and status registers, and statistics counters.

4.3. Frame Types

The MAC supports the following frame types:

- Basic Ethernet frames, including jumbo frames.
- VLAN and stacked VLAN frames.
- Control frames, which include pause and PFC frames.

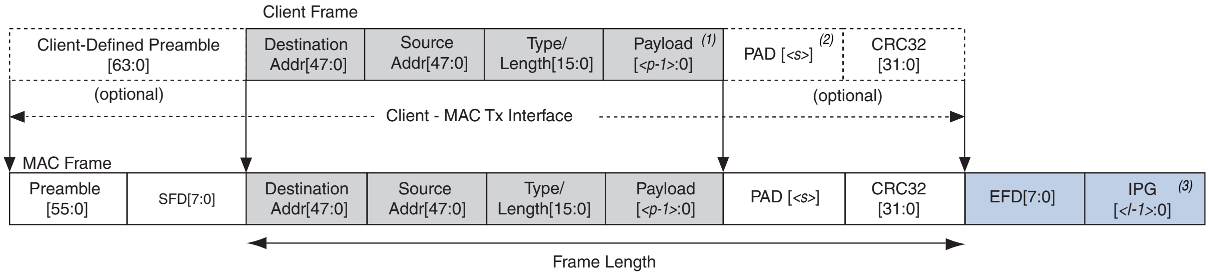
Refer to Appendix A, Frame Format for the frame formats and fields.

4.4. Transmit Datapath

The MAC Tx receives the client payload data with the destination and source addresses, and appends various control fields. Depending on the MAC configuration, the MAC Tx could perform the following tasks: pads the payload to satisfy the minimum Ethernet frame payload of 64 bytes, calculates and appends the CRC-32 field, modifies the source address, inserts inter-packet gap bytes, and accepts client-defined preamble bytes.

Figure 4–2 shows the typical flow of frame through the MAC Tx.

Figure 4–2. Typical Client Frame at Transmit Interface



Notes to Figure 4–2:

(1) <*p*> = payload size = 0–1500 bytes
(2) <*s*> = padding bytes = 0–46 bytes
(3) <*l*> = number of IPG bytes

4.4.1. Frame Payload Padding

The MAC Tx inserts pad bytes (0x00) into transmit frames when the payload length doesn't meet the minimum length required:

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

You can, however, disable pad bytes insertion by setting the `tx_padins_control` register to 0. If disabled, the MAC forwards the frames to the receiver without checking the frame length. You can check whether the frame is undersized by referring to the statistics collected.

4.4.2. Address Insertion

By default, the MAC Tx retains the source address received from the client. You can configure the MAX Tx to replace the source address with the primary MAC address specified in the `tx_addrins_macaddr0` and `tx_addrins_macaddr1` registers by setting the bit `tx_addrins_control[0]` to 1.

4.4.3. Frame Check Sequence (CRC-32) Insertion

The MAC Tx computes and inserts CRC-32 checksum into transmit frames. The MAC Tx computes the CRC-32 checksum over the frame bytes that include the source address, destination address, length, data, and pad bytes. The CRC checksum computation excludes the preamble, SFD, and FCS bytes.

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

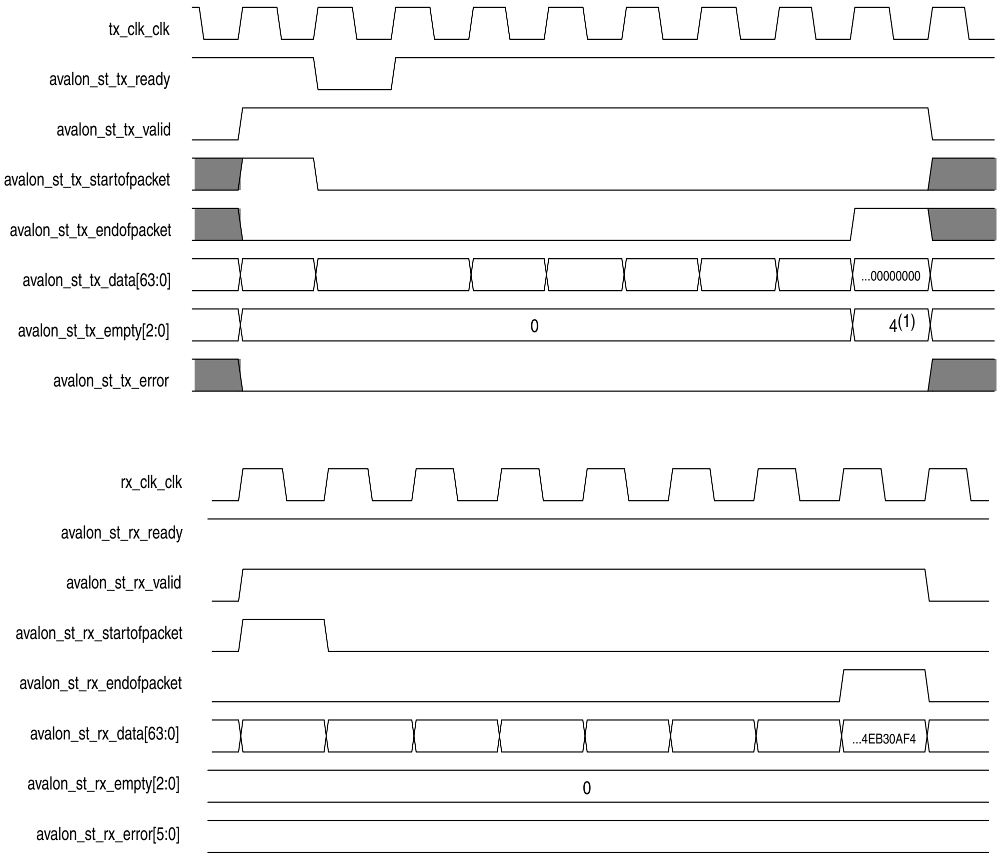
$$\mathrm{FCS(X)} = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X^{1} + 1$$

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}.

You can disable this function by setting the bit `tx_crcins_control`[1] to 0. You can also choose to omit the logic for CRC computation and insertion to save resources. When you disable or omit the CRC computation and insertion, the MAC does not append the CRC bits to the automatically generated pause frames.

Figure 4–3 on page 4–5 shows the timing diagram of the Avalon-ST transmit and receive interface where the FCS insertion function is on. The MAC Tx receives the frame without CRC-32 checksum and inserts CRC-32 checksum (4EB00AF4) into the frame. The frame is then loopback to the receive datapath with the `avalon_st_rx_data[63:0]` containing the CRC-32 checksum.

Figure 4–3. Avalon-ST Transmit and Receive Interface with CRC-32 Checksum Insertion

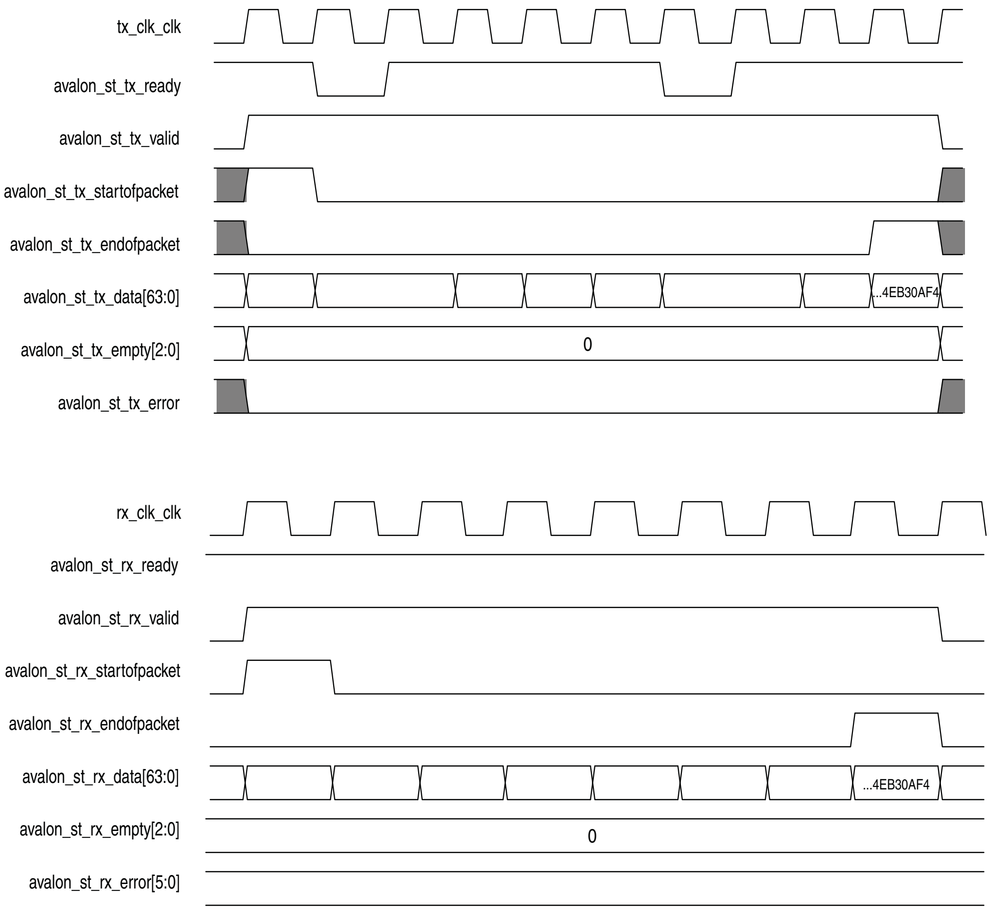


Note to Figure 4–3:

(1) This value (which varies depending on the frame size) indicates the number of symbols that are empty during the cycles that mark the end of a frame.

Figure 4–4 shows the timing diagram of the Avalon-ST transmit and receive interface where the FCS insertion function is off. The MAC Tx receives the frame which contains a CRC-32 checksum (4EB00AF4) and forwards the frame without performing CRC computation. The frame with the same CRC-32 field is then loopback to the receive datapath.

Figure 4–4. Avalon-ST Transmit and Receive Interface with CRC-32 Computation Disabled



4.4.4. XGMII Encapsulation

The MAC Tx inserts 7-byte preamble, 1-byte SFD and 1-byte EFD (0xFD) into frames received from the client. When you enable the preamble passthrough mode, the MAC Tx accepts 8-byte client-defined preamble in the frames received from the client and inserts a 1-byte EFD into the frames. For XGMII encapsulation, the first byte of the preamble data is converted to a 1-byte START (0xFB).

An underflow could occur on the Avalon-ST transmit interface. An underflow occurs when the `avalon_st_tx_valid` signal is deasserted in the middle of frame transmission. When this happens, the MAC Tx inserts an error character |E| into the frame and forwards the frame to the XGMII.

4.4.5. Inter-Packet Gap Generation and Insertion

The MAC Tx maintains an average IPG between transmit frames as required by the IEEE 802.3 Ethernet standard. The average IPG is maintained at 96 bit times (12 byte times) using the deficit idle count (DIC). The MAC Tx's decision to insert or delete idle bytes depends on the value of the DIC; the DIC is bounded between a minimum value of zero and maximum value of three. Averaging the IPG ensures that the MAC utilizes the maximum available bandwidth.

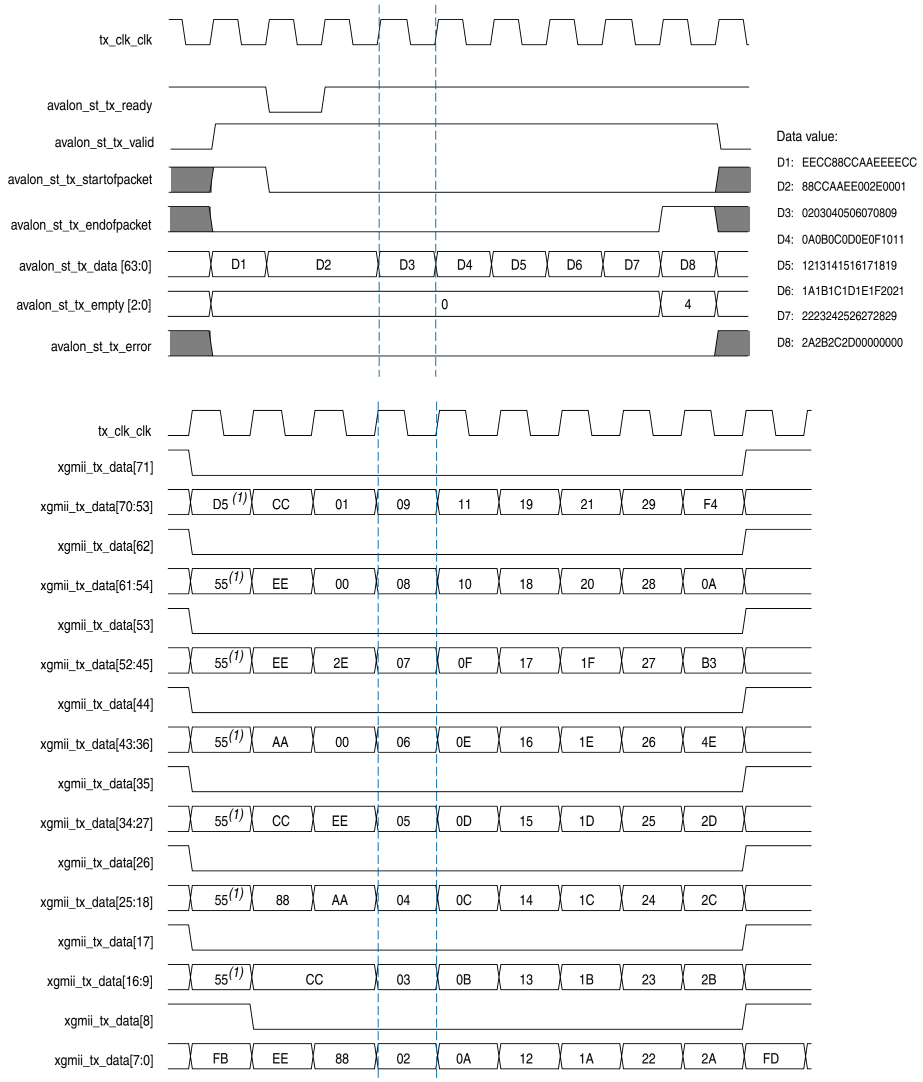
4.4.6. SDR XGMII Transmission

To comply with the IEEE 802.3 Clause 46 Ethernet standard, the MAC Tx ensures the following when transmitting frames on the SDR XGMII:

- Aligns the first byte of the frame to either lane 0 or lane 4 of the interface.
- Performs endian conversion. Transmit frames received from the client on the Avalon-ST interface are big endian. Frames transmitted on the SDR XGMII are little endian; the MAC Tx therefore transmits frames on this interface from the least significant byte.

Figure 4–5 shows the timing for the transmit frames on the Avalon-ST interface and the SDR XGMII. By comparing the data value in D3, the SDR XGMII performs endian conversion by transmitting the frames from the least significant byte.

Figure 4–5. Endian Conversion



Note to Figure 4–5:

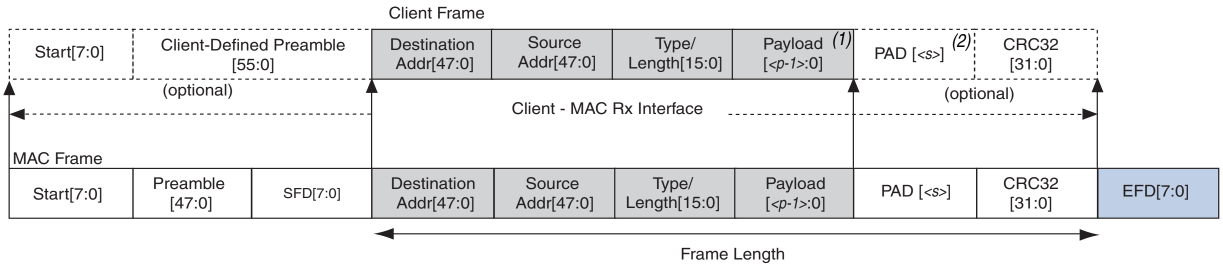
(1) In the preamble passthrough mode, the MAC Tx frame starts with a 1-byte START and a 7-byte client-defined preamble.

4.5. Receive Datapath

The MAC Rx receives Ethernet frames from the SDR XGMII and forwards the payload with relevant frame fields to the client after performing checks and filtering invalid frames. Some frame fields are optionally removed from the frame before MAC Rx forwards the frame to the client.

Figure 4–6 shows the typical flow of frame through the MAC Rx.

Figure 4–6. Typical Client Frame at Receive Interface



Notes to Figure 4–6:

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

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

(3) In the preamble passthrough mode, the MAC Rx frame starts with a 1-byte START and a 7-byte client-defined preamble.

4.5.1. XGMII Decapsulation

In the receive datapath, the MAC Rx decodes the data lanes coming through the SDR XGMII. The MAC Rx expects the first byte of the receive frame to be in either lane 0 (most significant byte) or lane 4. The receive frame must also be preceded by a column of idle bytes or an ordered set such as a local fault. A receive frame that does not satisfy these conditions is invalid and the MAC Rx drops the frame.

The MAC Rx then checks the sequence of the frame. The frame must begin with a 1-byte START, 6-byte preamble, and 1-byte SFD. Otherwise, the MAC Rx considers the frame invalid and drops it. For all valid frames, the MAC Rx removes the START, preamble, SFD, and EFD bytes and ensures that the first byte of the frame aligns to byte 0.

When you enable the preamble passthrough mode, the MAC Rx only checks for the following conditions: the frame begins with a 1-byte START and the minimum length of the frame including the START and client-defined preamble is 12 bytes. For frames that do not fulfill these conditions, the MAC Rx considers the frames invalid and drops them. For all valid frames, the MAC Rx removes the EFD byte and ensures that the first byte of the frame aligns to byte 0. The MAC Rx forwards the START and client-defined preamble to the client.

4.5.2. Frame Check Sequence (CRC-32) Checking

The CRC polynomial, as specified in the IEEE 802.3 Standard, is shown in the following equation:

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

The 32-bit CRC field is received in the following order: X^{31}, X^{30},..., X^{1}, X^{0}, where X^{31} is the MSB of FCS field and occupies the LSB position on first FCS byte field.

If a CRC-32 error is detected, the MAC Rx marks the frame invalid by setting `avalon_st_rx_error[1]` to 1 and forwards the frame to the client.

4.5.3. Address Checking

The MAC Rx 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 Rx always accepts broadcast frames. By default, it also receives all unicast and multicast frames unless configured otherwise in the `EN_ALLUCAST` and `EN_ALLMCAST` bits of the `rx_frame_control` register.

When the `EN_ALLUCAST` bit is set to 0, the MAC Rx filters unicast frames received. The MAC Rx accepts only unicast frames if the destination address matches the primary MAC address specified in the `rx_frame_addr0` and `rx_frame_addr1` registers. If any of the supplementary address bits are set to 1 (`EN_SUPP0/1/2/3` in the `rx_framedecoder_control` register), the MAC Rx also checks the destination address against the supplementary addresses in use.

When the `EN_ALLMCAST` bit is set to 0, the MAC Rx drops all multicast frames. This condition doesn't apply to global multicast pause frames.

4.5.4. Frame Type Checking

The MAC Rx 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 Rx 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 Rx continues to check the frame and payload lengths.
  - Length/type = 0x8088—Control frames. The next two bytes are the Opcode field which indicates the type of control frame. For pause frames (Opcode = 0x0001) and PFC frames (Opcode = 0x0101), the MAC Rx proceeds with pause frame processing (refer to "Congestion and Flow Control" on page 4–13). By default, the MAC Rx drops all control frames. If configured otherwise (`FWD_CONTROL` bit in the `rx_framedecoder_control` register = 1), the MAC Rx forwards control frames to the client.
  - For other field values, the MAC Rx forwards the receive frame to the client.

4.5.5. Length Checking

The MAC Rx 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—The value in the `rx_frame_maxlength` register.
- VLAN tagged—The value in the `rx_frame_maxlength` register plus four bytes.
- Stacked VLAN tagged—The value in the `rx_frame_maxlength` register plus eight bytes.

The MAC Rx keeps track of the actual payload length as it receives a frame and checks the actual payload length against the length/type or client length/type field. The payload length must be between 46 (0x2E) and 1500 (0x5DC). For VLAN and VLAN stacked frames, the minimum payload length is 42 (0x2A) or 38 (0x26) respectively and not exceeding the maximum value of 1500 (0x5DC).

The MAC Rx does not drop frames with invalid length. For the following length violations, the MAC Rx sets the corresponding error bit to 1:

- `avalon_st_rx_error[2]`—Undersized frame
- `avalon_st_rx_error[3]`—Oversized frame
- `avalon_st_rx_error[4]`—Invalid payload length, the actual payload length doesn't match the value of the length/type field

4.5.6. CRC-32 and Pad Removal

By default, the MAC Rx forwards receive frames to the client without removing pad bytes from the frames. You can, however, configure the MAC Rx to remove pad bytes by setting the bit `rx_padcrc_control`[1] to 1. When the bit is set to 1, the MAC Rx removes the pad bytes as well as the CRC-32 field from receive frames before forwarding the frames to the client.

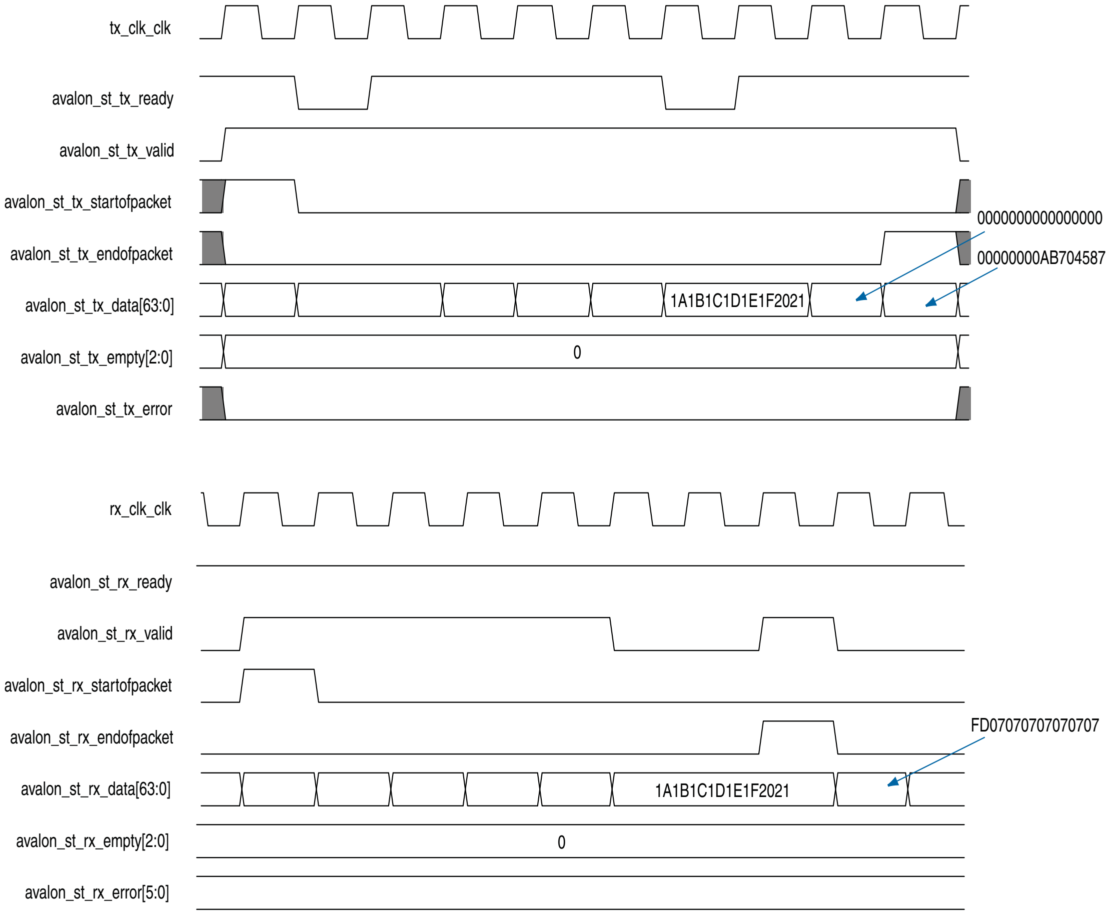
The MAC Rx removes pad bytes from receive frames whose payload length is less than the following values for the different frame types:

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

To retain the CRC-32 field, set the `rx_padcrc_control` register to 0.

Figure 4–7 on page 4–12 shows the timing for the Avalon-ST transmit and receive interface where the MAC Tx receives a frame with pad bytes and CRC-32 field inserted. The MAC Rx removes the pad bytes and CRC-32 field from the receive frame when the `rx_padcrc_control`[1] bit is set to 1.

Figure 4–7. Avalon-ST Transmit and Receive Interface with Pad Bytes and CRC-32 Field Removed



4.5.7. Overflow Handling

When an overflow occurs on the client side, the client can backpressure the Avalon-ST receive interface by deasserting the `avalon_st_rx_ready` signal. If an overflow occurs in the middle of frame transmission, the MAC Rx truncates the frame and the MAC Rx sets the error bit, `avalon_st_rx_error[5]`, to 1 to indicate an overflow. If frame transmission is not in progress when an overflow occurs, the MAC Rx drops the frame.

4.6. 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 byte on the network-side interface (XGMII SDR) 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 byte on the Avalon-ST interface after the bit was received on the network-side interface (XGMII SDR).

Table 4–1 shows the transmit and receive nominal latencies of the MAC.

Table 4–1. Transmit and Receive Latencies of the MAC

MAC Configuration	Latency (Clock Cycles) *(1) (2)*	
	Transmit (with respect to Tx clock)	Receive (with respect to Rx clock)
MAC only	10	12

Notes to Table 4–1:

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

(2) The latency values are based on the assumption that there is no backpressure on the Avalon-ST Tx and Rx interface.

4.7. Congestion and Flow Control

The flow control, as specified by IEEE 802.3 Annex 31B, is a mechanism to manage congestion at the local or remote partner. When the receiving device experiences congestion, it sends an XOFF pause frame to the emitting device to instruct the emitting device to stop sending data for a duration specified by the congested receiver. Data transmission resumes when the emitting device receives an XON pause frame (pause quanta = zero) or when the timer expires.

The PFC, as specified by IEEE 802.1Qbb, is a similar mechanism that manages congestion based on priority levels. The PFC supports up to 8 priority queues. When the receiving device experiences congestion on a priority queue, it sends a PFC frame requesting the emitting device to stop transmission on the priority queue for a duration specified by the congested receiver. When the receiving device is ready to receive transmission on the priority queue again, it sends a PFC frame instructing the emitting device to resume transmission on the priority queue.

☞ Ensure that only one type of flow control is enabled at any one time.

4.7.1. IEEE 802.3 Flow Control

This section describes the pause frame reception and transmission in the IEEE 802.3 flow control. To use the IEEE 802.3 flow control, set the following registers:

1. On the transmit datapath:
   - Set `tx_pfc_priority_enable` to 0 to disable the PFC.
   - Set `tx_pauseframe_enable` to 1 to enable the IEEE 802.3 flow control.
2. On the receive datapath:
   - Set `rx_pfc_control` to 1 to disable the PFC.
   - Set the `IGNORE_PAUSE` bit in the `rx_decoder_control register` to 0 to enable the IEEE 802.3 flow control.

4.7.1.1. Pause Frame Reception

When the MAC receives an XOFF pause frame, it stops transmitting frames to the remote partner for a period equal to the pause quanta field of the pause frame. If the MAC receives a pause frame in the middle of a frame transmission, the MAC finishes sending the current frame and then suspends transmission for a period specified by the pause quanta. The MAC resumes transmission when it receives an XON pause frame or when the timer expires. The pause quanta received overrides any counter currently stored. When the remote partner sends more than one pause quanta, the MAC sets the value of the pause to the last quanta it received from the remote partner. You have the option to configure the MAC to ignore pause frames and continue transmitting frames by setting the `IGNORE_PAUSE` bit in the `rx_decoder_control` register to 1.

4.7.1.2. Pause Frame Transmission

The MAC provides the following two methods for the client or connecting device to trigger pause frame transmission:

- `avalon_st_pause_data` signal—You can connect this 2-bit signal to a FIFO buffer or a client. Setting `avalon_st_pause_data[1]` to 1 triggers the transmission of XOFF pause frames; setting `avalon_st_pause_data[0]` to 1 triggers the transmission of XON pause frames.

  If pause frame transmission is triggered when the MAC is generating a pause frame, the MAC ignores the incoming request and completes the generation of the pause frame. Upon completion, if the `avalon_st_pause_data` signal remains asserted, the MAC generates a new pause frame and continues to do so until the signal is deasserted.

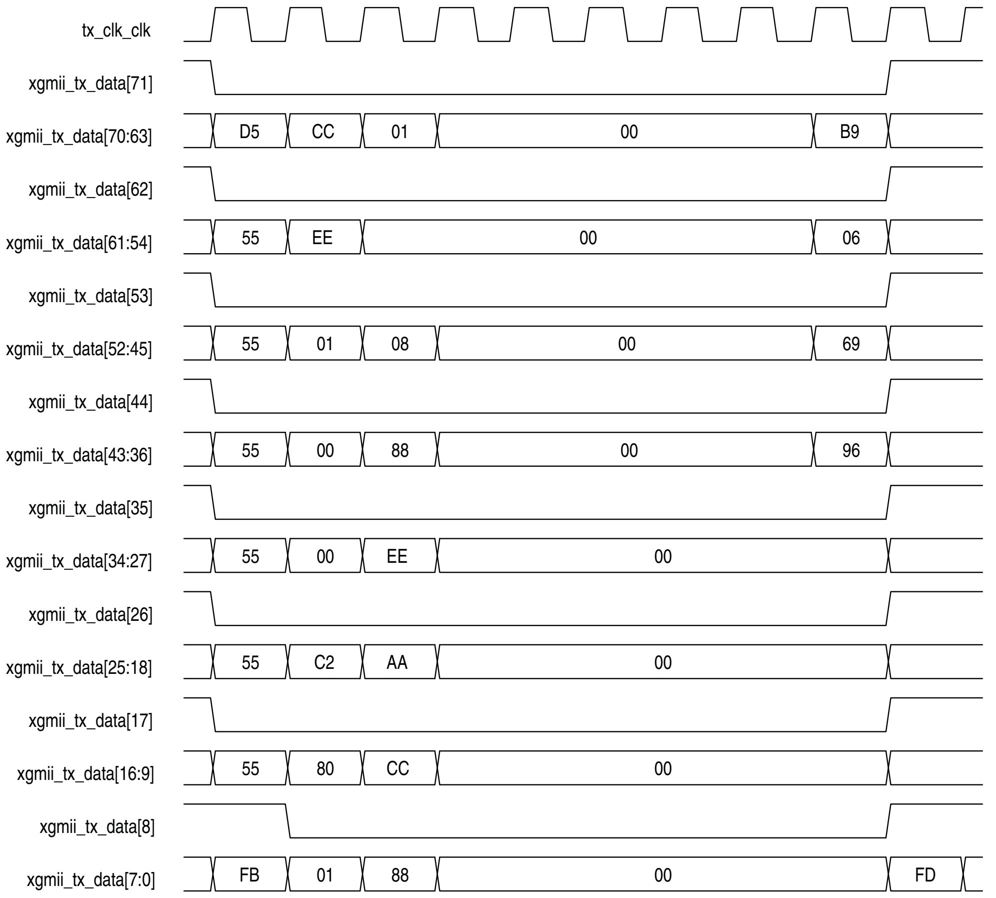
- `tx_pauseframe_control` register—A host (software) can set this register to trigger pause frames transmission. Setting `tx_pauseframe_control[1]` to 1 triggers the transmission of XOFF pause frames; setting `tx_pauseframe_control[0]` to 1 triggers the transmission of XON pause frames. The register clears itself after the request is executed.

You can configure the pause quanta in the `tx_pauseframe_quanta` register. The MAC sets the pause quanta field in XOFF pause frames to this register value.

☞ The `tx_pauseframe_control` register takes precedence over the `avalon_st_pause_data` signal.

Figure 4–8 shows the transmission of an XON pause frame. The MAC sets the destination address field to the global multicast address, 01-80-C2-00-00-01 (0x010000c28001) and the source address to the MAC primary address configured in the `tx_addrins_macaddr0` and `tx_addrins_madaddr1` registers.

Figure 4–8. XON Pause Frame Transmission



4.7.2. Priority-Based Flow Control

This section describes the PFC frame reception and transmission. Follow these steps to use the PFC:

1. Turn on the **Priority-based Flow Control (PFC)** parameter and specify the number of priority levels using the **Number of PFC Priorities** parameter. You can specify between 2 to 8 PFC priority levels.
2. Set the following registers.
   - On the transmit datapath:
     - Set `tx_pauseframe_enable` to 0 to disable the IEEE 802.3 flow control.
     - Set `tx_pfc_priority_enable[n]` to 1 to enable the PFC for priority queue *n*.
   - On the receive datapath:
     - Set the `IGNORE_PAUSE` bit in the `rx_decoder_control register` to 1 to disable the IEEE 802.3 flow control.
     - Set the `PFC_IGNORE_PAUSE_n` bit in the `rx_pfc_control` register to 0 to enable the PFC.
3. Connect the `avalon_st_tx_pfc_gen_data` signal to the corresponding Rx client logic and the `avalon_st_rx_pfc_pause_data` signal to the corresponding Tx client logic.
4. You have the option to configure the MAC Rx to forward the PFC frame to the client by setting the `FWD_PFC` bit in the `rx_pfc_control` register to 1. By default, the MAC Rx drops the PFC frame after processing it.

4.7.2.1. PFC Frame Reception

When the MAC Rx receives a PFC frame from the remote partner, it asserts the `avalon_st_rx_pfc_pause_data[n]` signal if Pause Quanta *n* is valid (Pause Quanta Enable [*n*] = 1) and greater than 0. The client suspends transmission from the Tx priority queue *n* for the period specified by Pause Quanta *n*. If the MAC Rx asserts the `avalon_st_rx_pfc_pause_data[n]` signal in the middle of a client frame transmission for the Tx priority queue *n*, the client finishes sending the current frame and then suspends transmission for the queue.

When the MAC Rx receives a PFC frame from the remote partner, it deasserts the `avalon_st_rx_pfc_pause_data[n]` signal if Pause Quanta *n* is valid (Pause Quanta Enable [*n*] = 1) and equal to 0. The MAC Rx also deasserts this signal when the timer expires. The client resumes transmission for the suspended Tx priority queue when the `avalon_st_rx_pfc_pause_data[n]` signal is deasserted.

When the remote partner sends more than one pause quanta for the Tx priority queue *n*, the MAC Rx sets the pause quanta *n* to the last pause quanta received from the remote partner.

For more information on the PFC pause frame, refer to Appendix A.4, Priority-Based Flow Control Frame.

4.7.2.2. PFC Frame Transmission

PFC frame generation is triggered through the avalon_st_tx_pfc_gen_data signal. Set the respective bits to generate XOFF or XON requests for the priority queues. Refer to Table 6–7 on page 6–17 for more information about the signal.

For XOFF requests, you can configure the pause quanta for each priority queue using the pfc_pause_quanta_*n* registers. For an XOFF request for priority queue *n*, the MAC Tx sets bit *n* in the Pause Quanta Enable field to 1 and the Pause Quanta *n* field to the value of the pfc_pause_quanta_*n* register. You can also configure the gap between successive XOFF requests for a priority queue using the pfc_holdoff_quanta_*n* register. Refer to Table 5–2 on page 5–2 for more information about these registers.

For XON requests, the MAC Tx sets the pause quanta to 0.

4.8. Error Handling (Link Fault)

The 10GbE MAC includes a reconciliation sublayer (RS) located between the MAC and the XGMII that handles local and remote faults.

When the local PHY reports a local fault (0x9c000001), the RS Rx sets link_fault_status_xgmii_rx_data to 01. The RS Tx starts sending the remote fault signal (0x9c000002) to the PHY, which is eventually received by the remote partner.

When the local PHY receives a remote fault signal, the RS Rx sets link_fault_status_xgmii_rx_data to 10. The RS Tx transmits IDLE signal (07070707). When the RS Tx starts sending the remote fault or IDLE signal, all data sent by the MAC Tx is lost.

If the client and the remote partner both receive valid data in more than 127 columns, the RS Rx sets link_fault_status_xgmii_rx_data to 00.

Figure 4–9 shows the fault signaling.

Figure 4–9. Fault Signaling

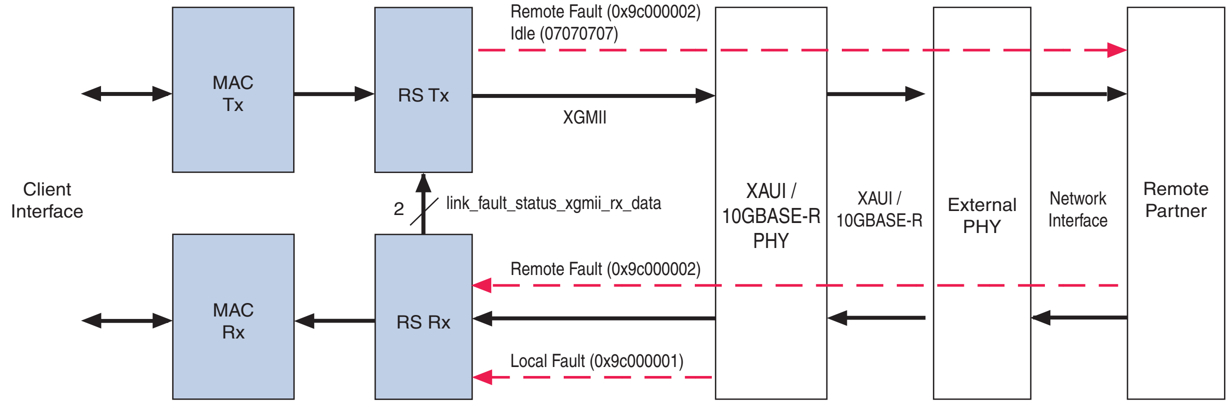
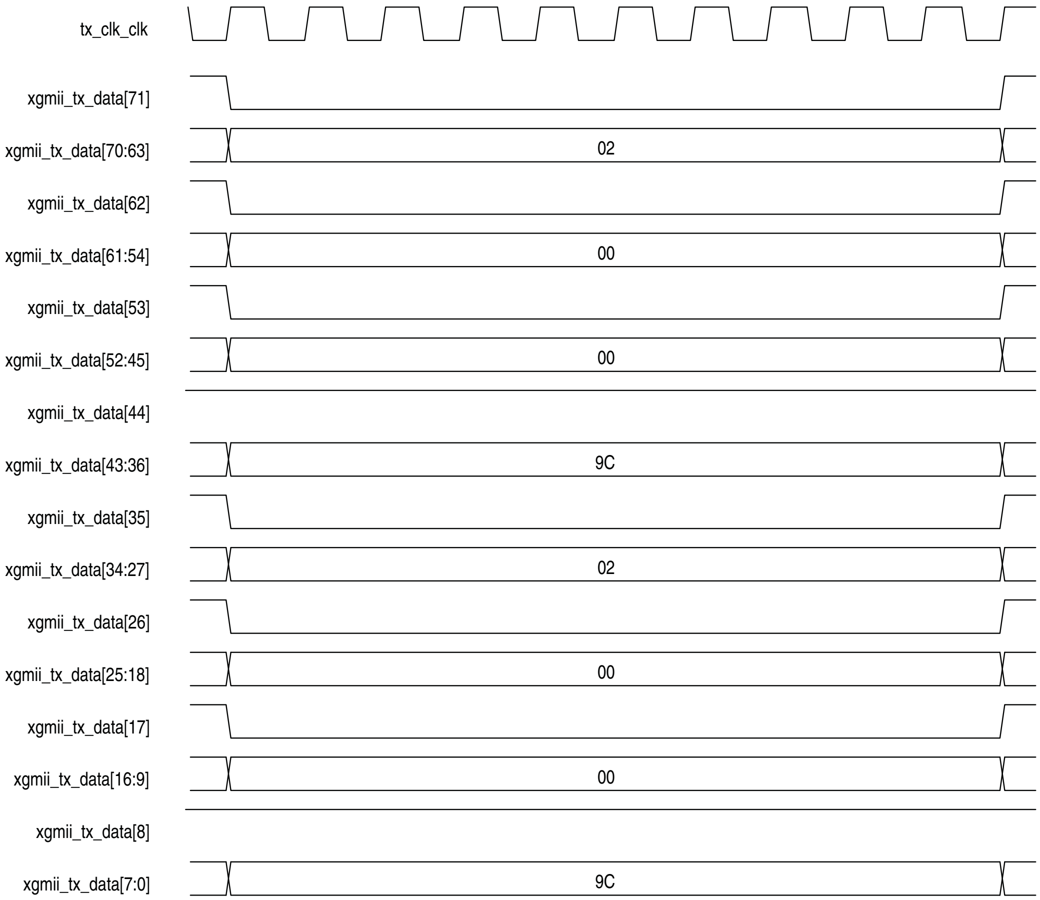


Figure 4–10 shows the timing for the XGMII Tx interface transmitting the remote fault signal (0x9c000002).

Figure 4–10. XGMII Tx interface Transmitting Remote Fault Signal



When you instantiate the MAC Rx only variation, connect the `link_fault_status_xgmii_rx_data` signal to the corresponding Rx client logic to handle the link fault. Similarly, when you instantiate the MAC Tx only variation, connect the `link_fault_status_xgmii_tx_data` signal to the corresponding Tx client logic. For more information on the signals, refer to "SDR XGMII Signals" on page 6–9.

5. Registers

This section defines the MAC registers. The statistics collected on the transmit and receive datapaths are categorized as good, error, or invalid frames.

- Good frame—Error-free frames with a valid frame length.
- Error frame—Frames that contain errors or with an invalid frame length.
- Invalid frame—Frames that are not addressed to the MAC. It may or may not contain error within the frame or have an invalid frame length. The MAC drops invalid frames.

When you select the MAC Rx only variation, the register offsets from 0x000 to 0x3FFF are available for Rx status and configuration registers. Similarly, when you select the MAC Tx only variation, the register offsets from 0x4000 to 0x7FFF are available for Tx status and configuration registers. All status and configuration registers are as defined in Table 5–2 on page 5–2.

CAUTION Altera recommends accessing only the available register spaces in the MAC Rx only variation or the MAC Tx only variation. Accessing unavailable register spaces may cause the MAC to lock the Avalon-MM bus.

☞ Altera has updated all register address for the 10GbE MAC IP core as part of register map expansion to accommodate new registers. Table 5–1 summarizes the changes.

Table 5–1. Summary of Register Address Expansion

Component Name	Previous Address Range (ACDS Version 10.0, 10.1)	New Address Range (ACDS Version 11.0 Onwards)
RX Datapath		
RX Packet Transfer	0x000:0x00F	0x000:0x0FF
RX Pad/CRC Remover	0x010:0x01F	0x100:0x1FF
RX CRC Checker	0x020:0x0FF	0x200:0x2FF
RX Packet Overflow	0x180:0x1FF	0x300:0x3FF
RX Preamble Control	—	0x400:0x4FF
RX Lane Decoder	—	0x500:0x1FFF
RX Frame Decoder	0x100:0x17F	0x2000:0x2FFF
RX Statistics Counters	0x200:0x3FF	0x3000:0x3FFF
TX Datapath		
TX Packet Transfer	0x400:0x40F	0x4000:0x40FF
TX Pad Inserter	0x410:0x41F	0x4100:0x41FF
TX CRC Inserter	0x420:0x45F	0x4200:0x42FF
TX Packet Underflow	0x580:0x5FF	0x4300:0x43FF
TX Preamble Control	—	0x4400:0x44FF
TX Pause Frame Control and Generator	0x460:0x47F	0x4500:0x45FF

Table 5–1. Summary of Register Address Expansion

Component Name	Previous Address Range (ACDS Version 10.0, 10.1)	New Address Range (ACDS Version 11.0 Onwards)
TX PFC Generator	—	0x4600:0x47FF
TX Address Inserter	0x480:0x4FF	0x4800:0x5FFF
TX Frame Decoder	0x500:0x57F	0x6000:0x6FFF
TX Statistics Counters	0x600:0x7FF	0x7000:0x7FFF

☞ If you instantiate the IP core using the MegaWizard Plug-in Manager flow, use double word (dword) addressing to access the register spaces. Convert the byte offsets to dword offsets by dividing the byte offsets by 4. For example,

- rx_padcrc_control byte offset = 0x100
- rx_padcrc_control word offset = 0x100 ÷ 4 = 0x040

5.1. MAC Registers

Table 5–2 shows the MAC registers.

Table 5–2. MAC Registers (Part 1 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
RX Packet Transfer (0x000:0x0FF)				
0x000	rx_transfer_control	RW	0x0	Receive path enable. ■ Bit 0 configures the receive path. 0—Enables the receive path. 1—Disables the receive path and drops all receive frames. ■ Bits 1 to 31 are not used.
0x004	rx_transfer_status	RO	0x0	■ Bit 0 indicates the status of the receive path. 0—The receive path is enabled. 1—The receive path is disabled. ■ Bits 1 to 31 are not used.
0x008 – 0x0FF	Reserved	—	—	Reserved for future use.

Table 5–2. MAC Registers (Part 2 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
RX Pad/CRC Remover (0x100:0x1FF)				
0x100	`rx_padcrc_control`	RW	0x1	Padding and CRC removal (through the `avalon_st_rx_data` signal). ■ Bit 0 configures CRC removal. 0—Retains the CRC field in receive packets. 1—Removes the CRC field from receive packets. ■ Bit 1 configures padding and CRC removal. 0—Retains the padding bytes and CRC field. 1—Removes the padding bytes and CRC field from receive packets. The setting of this bit takes precedence over bit 0. ■ Bits 2 to 31 are not used.
0x104 – 0x1FF	Reserved	—	—	Reserved for future use.
RX CRC Checker (0x200:0x2FF)				
0x200	`rx_crccheck_control`	RW	0x2	CRC checking: ■ Bit 0—Always set this bit to 0. ■ Bit 1 configures CRC checking. 0—Ignores the CRC field. 1—Checks the CRC field. ■ Bits 2 to 31 are not used.
0x204 – 0x2FF	Reserved	—	—	Reserved for future use.
RX Packet Overflow (0x300:0x3FF)				
0x300	`rx_pktovrflow_error`	RO	0x0	36-bit error counter that collects the number of receive frames that are truncated when FIFO buffer overflow persists: ■ The first 32 bits occupy the register at offset 0x300. ■ The last 4 bits occupy the first four bits of the register at offset 0x304. Bits 4 to 31 are unused. The counter will be cleared when the last 4 bits have been read. If only the first 32 bits are read, the counter will not be cleared.
0x304			0x0	

Table 5–2. MAC Registers (Part 3 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
0x308	`rx_pktovrflow_etherStatsDropEvents`	RO	0x0	36-bit error counter that collects the number of receive frames that are dropped when FIFO buffer overflow persists: ■ The first 32 bits occupy the register at offset 0x308. ■ The last 4 bits occupy the first four bits of the register at offset 0x38C. Bits 4 to 31 are unused. The counter will be cleared when the last 4 bits have been read. If only the first 32 bits are read, the counter will not be cleared.
0x30C			0x0	
0x310 – 0x3FF	Reserved	—	—	Reserved for future use.
RX Preamble Control (0x400:0x4FF)				
0x400	`rx_lane_decoder_preamble_control`	RW	0x0	■ Bit 0 determines whether or not the client-defined preamble is forwarded to the client frame. 0—Removes the client-defined preamble from the receive frame. 1—Forwards the client-defined preamble to the client. ■ Bits 1 to 31 are not used.
0x404 – 0x4FF	Reserved	—	—	Reserved for future use.
RX Lane Decoder (0x500:0x1FFF)				
0x500	`rx_preamble_inserter_control`	RW	0x0	■ Bit 0 enables the preamble passthrough mode on the receive datapath. 0—Disables the preamble passthrough mode. 1—Enables the preamble passthrough mode. ■ Bits 1 to 31 are not used. For more information on the XGMII decapsulation in the preamble passthrough mode, refer to "XGMII Decapsulation" on page 4–9.
0x504 – 0x1FFF	Reserved	—	—	Reserved for future use.
RX Frame Decoder (0x2000:0x2FFF)				
0x2000	`rx_frame_control`	RW	0x3	Specifies valid frame types, pause frames handling, and use of supplementary addresses. Refer to "Rx_frame_control Register" on page 5–15 for the bit description.

Table 5–2. MAC Registers (Part 4 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
0x2004	`rx_frame_maxlength`	RW	1518	■ Bits 0 to 15 specify the maximum allowable frame length. The MAC asserts the `avalon_st_rx_error[3]` signal when the length of the receive frame exceeds the value of this register. ■ Bits 16 to 31 are not used.
0x2008	`rx_frame_addr0`	RW	0x0	6-byte primary MAC address. You must map the address to the registers in the following manner: ■ `rx_frame_addr0` = Last four bytes of the address ■ `rx_frame_addr1`[0:15]= First two bytes of the address. Bits 16 to 31 are not used. Example: If the primary MAC address is 00-1C-23-17-4A-CB, set `rx_frame_addr0` to 0x23174ACB and `rx_frame_addr1` to 0x0000001C. The IP core uses the primary MAC address to filter unicast frames when the `en_allucast` bit of the `rx_frame_control` register is set to 0.
0x200C	`rx_frame_addr1` *(1)*	RW	0x0	
0x2010	`rx_frame_spaddr0_0`	RW	0x0	You can specify up to four 6-byte supplementary addresses: ■ `rx_framedecoder_spaddr0_0/1` ■ `rx_framedecoder_spaddr1_0/1` ■ `rx_framedecoder_spaddr2_0/1` ■ `rx_framedecoder_spaddr3_0/1` You must map the supplementary addresses to the respective registers in the same manner as the primary MAC address. Refer to the description of `rx_frame_addr0` and `rx_frame_addr1`. The IP core uses the supplementary addresses to filter unicast frames when the following conditions are set: ■ The use of the supplementary addresses are enabled using the respective bits in the `rx_frame_control` register (refer to “Rx_frame_control Register” on page 5–15). ■ The `en_allucast` bit of the `rx_frame_control` register is set to 0.
0x2014	`rx_frame_spaddr0_1` *(1)*	RW	0x0	
0x2018	`rx_frame_spaddr1_0`	RW	0x0	
0x201C	`rx_frame_spaddr1_1` *(1)*	RW	0x0	
0x2020	`rx_frame_spaddr2_0`	RW	0x0	
0x2024	`rx_frame_spaddr2_1` *(1)*	RW	0x0	
0x2028	`rx_frame_spaddr3_0`	RW	0x0	
0x202C	`rx_frame_spaddr3_1` *(1)*	RW	0x0	
0x2060	`rx_pfc_control`	RW	0x1	PFC enable for the priority queues on the receive datapath. Refer to “Rx_pfc_control Register” on page 5–16 for the bit description.

Table 5–2. MAC Registers (Part 5 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
0x2064 – 0x2FFF	Reserved	—	—	Reserved for future use.
TX Packet Transfer (0x4000:0x40FF)				
0x4000	tx_transfer_control	RW	0x0	Backpressure enable. ■ Bit 0 configures transmit transfer control. 0—Enables transmit transfer datapath. 1—Disables transmit transfer datapath on the Avalon-ST transmit interface. The IP core deasserts the avalon_st_tx_ready signal. ■ Bits 1 to 31 are not used.
0x4004	tx_transfer_status	RO	0x0	■ Bit 0 indicates if transmit transfer datapath is enabled on the Avalon-ST transmit interface. 0—Transmit transfer datapath is disabled. 1—Transmit transfer datapath is enabled.. ■ Bits 1 to 31 are not used.
0x4008 – 0x40FF	Reserved	—	—	Reserved for future use.
TX Pad Inserter (0x4100:0x41FF)				
0x4100	tx_padins_control	RW	0x1	■ Bit 0 indicates padding insertion. 0—No effect on transmit frames. 1—Inserts padding bytes into transmit frames until the frame length reaches 60 bytes. To achieve the minimum 64 bytes, ensure that the CRC field is inserted (refer to tx_crcins_control). ■ Bits 1 to 31 are not used.
0x4104 – 0x41FF	Reserved	—	—	Reserved for future use.
TX CRC Inserter (0x4200:0x42FF)				
0x4200	tx_crcins_control	RW	0x3	CRC insertion. ■ Bit 0—Always set this bit to 1. ■ Bit 1 configures CRC insertion. 0—Disables CRC insertion. 1—Computes CRC and inserts it into transmit frames. ■ Bits 2 to 31 are not used.
0x4204 – 0x42FF	Reserved	—	—	Reserved for future use.

Table 5–2. MAC Registers (Part 6 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
TX Packet Underflow (0x4300:0x43FF)				
0x4300 0x4304	tx_pktunderflow_error	RO	0x0	36-bit error counter that collects the number of transmit frames that are truncated when FIFO buffer underflow persists. ■ The first 32 bits occupy the register at offset 0x580. ■ The last 4 bits occupy the first four bits of the register at offset 0x584. Bits 4 to 31 are not used. The counter will be cleared when the last 4 bits have been read. If only the first 32 bits are read, the counter will not be cleared.
0x4308 – 0x43FF	Reserved	—	—	Reserved for future use.
TX Preamble Control (0x4400:0x44FF)				
0x4400	tx_preamble_control	RW	0x0	■ Bit 0 configures the preamble passthrough mode in the transmit datapath. 0—Set to 1 to disable the preamble passthrough mode. 1—Set to 1 to enable the preamble passthrough mode. MAC Tx identifies the first 8-bytes of the client frame as client-defined preamble. ■ Bits 1 to 31 are not used.
0x4404 – 0x44FF	Reserved	—	—	Reserved for future use.
TX Pause Frame Control and Generator (0x4500:0x45FF)				
0x4500	tx_pauseframe_control	RW	0x0	IEEE 802.3 pause frame generation. ■ Bit 0 configures the generation of XON pause frames. 0—Disables pause frame generation. 1—Generates a pause frame with a pause quanta value of 0. ■ Bit 1 configures the generation of XOFF pause frames. 0—Disables pause frame generation. 1—Generates a pause frame using the pause quanta specified in the tx_pauseframe_quanta register. ■ Bits 2 to 31 are not used. If both bits 0 and 1 are set to 1 simultaneously, the IP core does not generate any pause frames.
0x4504	tx_pauseframe_quanta	RW	0x0	16-bit pause quanta. The IP core uses this value when it generates XOFF pause frames. Bits 16 to 31 are reserved.

Table 5–2. MAC Registers (Part 7 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
0x4508	`tx_pauseframe_enable`	RW	0x1	IEEE 802.3 pause frame generation process. ■ Bit 0 configures the process to generate the IEEE 802.3 pause frame. 0—Disables pause frame generation process. 1—Enables pause frame generation process. ■ Bits 1 to 31 are not used.
0x450C – 0x45FF	Reserved	—	—	Reserved for future use.
TX PFC Generator (0x4600:0x47FF)				
0x4600	`pfc_pause_quanta_0`	RW	0x0	`pfc_pause_quanta_`*n* specifies the pause length for priority queue *n*. The pause length is in unit of pause quanta, where 1 pause quanta = 512 bits time.
0x4604	`pfc_pause_quanta_1`	RW	0x0	
0x4608	`pfc_pause_quanta_2`	RW	0x0	
0x460C	`pfc_pause_quanta_3`	RW	0x0	
0x4610	`pfc_pause_quanta_4`	RW	0x0	
0x4614	`pfc_pause_quanta_5`	RW	0x0	
0x4618	`pfc_pause_quanta_6`	RW	0x0	
0x461C	`pfc_pause_quanta_7`	RW	0x0	
0x4620 – 0x463F	Reserved	—	—	Reserved for future use.
0x4640	`pfc_holdoff_quanta_0`	RW	0x0	`pfc_holdoff_quanta_`*n* specifies the gap between consecutive XOFF requests for priority queue *n*. The gap is in unit of holdoff quanta, where 1 holdoff quanta = 512 bits time.
0x4644	`pfc_holdoff_quanta_1`	RW	0x0	
0x4648	`pfc_holdoff_quanta_2`	RW	0x0	
0x464C	`pfc_holdoff_quanta_3`	RW	0x0	
0x4650	`pfc_holdoff_quanta_4`	RW	0x0	
0x4654	`pfc_holdoff_quanta_5`	RW	0x0	
0x4658	`holdoff_quanta_6`	RW	0x0	
0x465C	`holdoff_quanta_7`	RW	0x0	
0x4660 – 0x467F	Reserved	—	—	Reserved for future use.
0x4680	`tx_pfc_priority_enable`	RW	0x0	Enables PFC for a priority queue on the transmit datapath. ■ Bit 0 to 7: Setting bit *n* in this register enables PFC for priority queue *n*. For example, setting tx_pfc_priority_enable[0] enables PFC for priority queue 0. ■ Bits 8 to 31 are not used.
0x4684 – 0x47FF	Reserved	—	—	Reserved for future use.
TX Address Inserter (0x4800:0x49FF)				

Table 5–2. MAC Registers (Part 8 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
0x4800	`tx_addrins_control`	RW	0x0	Address insertion on the transmit datapath. ■ Bit 0 configures address insertion. 0—Disables address insertion on the transmit datapath. 1—Overwrites the source address field of transmit frames with the address configured in the `tx_addrins_macaddr0` and `tx_addrins_macaddr1` registers. ■ Bits 1 to 31 are not used.
0x4804	`tx_addrins_macaddr0`	RW	0x0	6-byte MAC address. You must map the address to the registers in the following manner: ■ `tx_addrins_macaddr0` = Last four bytes of the address ■ `tx_addrins_macaddr1`[0:15] = First two bytes of the address. Bits 16 to 31 are not used. Example: If the primary MAC address is 00-1C-23-17-4A-CB, set `tx_addrins`_macaddr0 to 0x23174ACB and `tx_addrins`_macaddr1 to 0x0000001C. The IP core writes this address to the source address field of transmit frames when address insertion on transmit is enabled (refer to description of `tx_addrins_control`).
0x4808	`tx_addrins_macaddr1`	RW	0x0	
0x480C – 0x5FFF	Reserved	—	—	Reserved for future use.
TX Frame Decoder (0x6000:0x6FFF)				
0x6000	Reserved	—	—	Reserved for future use.
0x6004	`tx_frame_maxlength`	RW	1518	■ Bits 0 to 15 specifies the maximum allowable frame length for the statistic counter. The MAC asserts the `avalon_st_txstatus_error[1]` signal when the length of the transmit frame exceeds the value of this register and flags it as oversized frame. The value of this register does not affect the allowable frame size that can be sent through the Tx path. ■ Bits 16 to 31 are not used.
0x6008 – 0x6FFF	Reserved	—	—	Reserved for future use.

Table 5–2. MAC Registers (Part 9 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
RX Statistics Counters (0x3000:0x3FFF)—Collect statistics on the receive path. Prefixed with rx_. (1) TX Statistics Counters (0x7000:0x7FFF)—Collect statistics on the transmit path. Prefixed with tx_.				
0x3000	`rx_stats_clr`	RWC	0x0	■ Bit 0—Set this register to 1 to clear all statistics counters for the receive path. ■ Bits 1 to 31 are not used.
0x7000	`tx_stats_clr`	RWC	0x0	■ Bit 0—Set this register to 1 to clear all statistics counters for the transmit path. ■ Bits 1 to 31 are not used.
0x3004	Reserved	—	—	Reserved for future use.
0x7004				
0x3008	`rx_stats_framesOK`	RO	0x0	■ Bit 0—The number of frames that are successfully received or transmitted, including control frames. ■ 36-bit width register: ■ 0x3008 and 0x7008 = bits [31:0] ■ 0x300C and 0x700C = bits [35:32]
0x300C				
0x7008	`tx_stats_framesOK`			
0x700C				
0x3010	`rx_stats_framesErr` (2)	RO	0x0	■ Bit 0—The number of errored frames that are received or transmitted, including control frames. ■ 36-bit width register: ■ 0x3010 and 0x7010 = bits [31:0] ■ 0x3014 and 0x7014 = bits [35:32]
0x3014				
0x7010	`tx_stats_framesErr` (2)			
0x7014				
0x3018	`rx_stats_framesCRCErr`	RO	0x0	■ Bit 0—The number of receive or transmit frames with only CRC error. ■ 36-bit width register: ■ 0x3018 and 0x7018 = bits [31:0] ■ 0x301C and 0x701C = bits [35:32]
0x301C				
0x7018	`tx_stats_framesCRCErr`			
0x701C				
0x3020	`rx_stats_octetsOK`	RO	0x0	■ Bit 0—The number of data and padding octets that are successfully received or transmitted, including control frames.
0x3024				
0x7020	`tx_stats_octetsOK`			
0x7024				
0x3028	`rx_stats_pauseMACCtrl Frames`	RO	0x0	■ Bit 0—The number of valid pause frames received or transmitted. ■ 36-bit width register: ■ 0x3028 and 0x7028 = bits [31:0] ■ 0x302C and 0x702C = bits [35:32]
0x302C				
0x7028	`tx_stats_pauseMACCtrl Frames`			
0x702C				
0x3030	`rx_stats_ifErrors`	RO	0x0	■ Bit 0—The number of errored and invalid frames received or transmitted. ■ 36-bit width register: ■ 0x3030 and 0x7030 = bits [31:0] ■ 0x3034 and 0x7034 = bits [35:32]
0x3034				
0x7030	`tx_stats_ifErrors`			
0x7034				

Table 5–2. MAC Registers (Part 10 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
0x3038	`rx_stats_unicast FramesOK`	RO	0x0	■ Bit 0—The number of good unicast frames that are successfully received or transmitted, excluding control frames. ■ 36-bit width register: ■ 0x3038 and 0x7038 = bits [31:0] ■ 0x303C and 0x703C = bits [35:32]
0x303C				
0x7038	`tx_stats_unicast FramesOK`			
0x703C				
0x3040	`rx_stats_unicast FramesErr` (2)	RO	0x0	■ Bit 0—The number of errored unicast frames received or transmitted, excluding control frames. ■ 36-bit width register: ■ 0x3040 and 0x7040 = bits [31:0] ■ 0x3044 and 0x7044 = bits [35:32]
0x3044				
0x7040	`tx_stats_unicast FramesErr` (2)			
0x7044				
0x3048	`rx_stats_multicast FramesOK`	RO	0x0	■ Bit 0—The number of good multicast frames that are successfully received or transmitted, excluding control frames. ■ 36-bit width register: ■ 0x3048 and 0x7048 = bits [31:0] ■ 0x304C and 0x704C = bits [35:32]
0x304C				
0x7048	`tx_stats_multicast FramesOK`			
0x704C				
0x3050	`rx_stats_multicast FramesErr` (2)	RO	0x0	■ Bit 0—The number of errored multicast frames received or transmitted, excluding control frames. ■ 36-bit width register: ■ 0x3050 and 0x7050 = bits [31:0] ■ 0x3054 and 0x7054 = bits [35:32]
0x3054				
0x7050	`tx_stats_multicast FramesErr` (2)			
0x7054				
0x3058	`rx_stats_broadcast FramesOK`	RO	0x0	■ Bit 0—The number of good broadcast frames received or transmitted, excluding control frames. ■ 36-bit width register: ■ 0x3058 and 0x7058 = bits [31:0] ■ 0x305C and 0x705C = bits [35:32]
0x305C				
0x7058	`tx_stats_broadcast FramesOK`			
0x705C				
0x3060	`rx_stats_broadcast FramesErr` (2)	RO	0x0	■ Bit 0—The number of errored broadcast frames received or transmitted, excluding control frames. ■ 36-bit width register: ■ 0x3060 and 0x7060 = bits [31:0] ■ 0x3064 and 0x7064 = bits [35:32]
0x3064				
0x7060	`tx_stats_broadcast FramesErr` (2)			
0x7064				
0x3068	`rx_stats_etherStats Octets`	RO	0x0	■ Bit 0—The total number of octets received or transmitted. This count includes good, errored, and invalid frames.
0x306C				
0x7068	`tx_stats_etherStats Octets`			
0x706C				

Table 5–2. MAC Registers (Part 11 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
0x3070	`rx_stats_etherStatsPkts`	RO	0x0	■ Bit 0—The total number of good, errored, and invalid frames received or transmitted. ■ 36-bit width register: ■ 0x3070 and 0x7070 = bits [31:0] ■ 0x3074 and 0x7074 = bits [35:32]
0x3074				
0x7070	`tx_stats_etherStatsPkts`			
0x7074				
0x3078	`rx_stats_etherStats UndersizePkts`	RO	0x0	■ Bit 0—The number of undersized frames (frame length less than 64 bytes, including the CRC field) received or transmitted. ■ 36-bit width register: ■ 0x3078 and 0x7078 = bits [31:0] ■ 0x307C and 0x707C = bits [35:32]
0x307C				
0x7078	`tx_stats_etherStats UndersizePkts`			
0x707C				
0x3080	`rx_stats_etherStats OversizePkts`	RO	0x0	■ Bit 0—The number of oversized frames (frame length more than `rx_frame_maxlength`, including the CRC field) received. ■ 36-bit width register: ■ 0x3080 = bits [31:0] ■ 0x3084 = bits [35:32]
0x3084				
0x7080	`tx_stats_etherStats OversizePkts`	RO	0x0	■ Bit 0—The number of oversized frames (frame length more than `tx_frame_maxlength`, including the CRC field) transmitted. ■ 36-bit width register: ■ 0x7080 = bits [31:0] ■ 0x7084 = bits [35:32]
0x7084				
0x3088	`rx_stats_etherStats Pkts64Octets`	RO	0x0	■ Bit 0—The number of 64-byte receive or transmit frames, including the CRC field but excluding the preamble and SFD bytes. This count includes good, errored, and invalid frames. ■ 36-bit width register: ■ 0x3088 and 0x7088 = bits [31:0] ■ 0x308C and 0x708C = bits [35:32]
0x308C				
0x7088	`tx_stats_etherStats Pkts64Octets`			
0x708C				
0x3090	`rx_stats_etherStats Pkts65to127Octets`	RO	0x0	■ Bit 0—The number of receive or transmit frames between the length of 65 and 127 bytes, including the CRC field but excluding the preamble and SFD bytes. This count includes good, errored, and invalid frames. ■ 36-bit width register: ■ 0x3090 and 0x7090 = bits [31:0] ■ 0x3094 and 0x7094 = bits [35:32]
0x3094				
0x7090	`tx_stats_etherStats Pkts65to127Octets`			
0x7094				

Table 5–2. MAC Registers (Part 12 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
0x3098 0x309C 0x7098 0x709C	`rx_stats_etherStatsPkts128to255Octets` `tx_stats_etherStatsPkts128to255Octets`	RO	0x0	■ Bit 0—The number of receive or transmit frames between the length of 128 and 255 bytes, including the CRC field but excluding the preamble and SFD bytes. This count includes good, errored, and invalid frames. ■ 36-bit width register: ■ 0x3098 and 0x7098 = bits [31:0] ■ 0x309C and 0x709C = bits [35:32]
0x30A0 0x30A4 0x70A0 0x70A4	`rx_stats_etherStatsPkts256to511Octets` `tx_stats_etherStatsPkts256to511Octets`	RO	0x0	■ Bit 0—The number of receive or transmit frames between the length of 256 and 511 bytes, including the CRC field but excluding the preamble and SFD bytes. This count includes good, errored, and invalid frames. ■ 36-bit width register: ■ 0x30A0 and 0x70A0 = bits [31:0] ■ 0x30A4 and 0x70A4 = bits [35:32]
0x30A8 0x30AC 0x70A8 0x70AC	`rx_stats_etherStatsPkts512to1023Octets` `tx_stats_etherStatsPkts512to1023Octets`	RO	0x0	■ Bit 0—The number of receive or transmit frames between the length of 512 and 1,023 bytes, including the CRC field but excluding the preamble and SFD bytes. This count includes good, errored, and invalid frames. ■ 36-bit width register: ■ 0x30A8 and 0x70A8 = bits [31:0] ■ 0x30AC and 0x70AC = bits [35:32]
0x30B0 0x30BC 0x70B0 0x70B4	`rx_stats_etherStatPkts1024to1518Octets` `tx_stats_etherStatPkts1024to1518Octets`	RO	0x0	■ Bit 0—The number of receive or transmit frames between the length of 1,024 and 1,518 bytes, including the CRC field but excluding the preamble and SFD bytes. This count includes good, errored, and invalid frames. ■ 36-bit width register: ■ 0x30B0 and 0x70B0 = bits [31:0] ■ 0x30B4 and 0x70B4 = bits [35:32]
0x30B8 0x30BC 0x70B8 0x70BC	`rx_stats_etherStatsPkts1519toXOctets` `tx_stats_etherStatsPkts1519toXOctets`	RO	0x0	■ Bit 0—The number of receive or transmit frames equal or more than the length of 1,519 bytes, including the CRC field but excluding the preamble and SFD bytes. This count includes good, errored, and invalid frames. ■ 36-bit width register: ■ 0x30B8 and 0x70B8= bits [31:0] ■ 0x30BC and 0x70BC = bits [35:32]

Table 5–2. MAC Registers (Part 13 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
0x30C0	`rx_stats_etherStats Fragments`	RO	0x0	■ Bit 0—The total number of receive or transmit frames with length less than 64 bytes and CRC error. This count includes errored and invalid frames. ■ 36-bit width register: ■ 0x30C0 and 0x70C0 = bits [31:0] ■ 0x30C4 and 0x70C4 = bits [35:32]
0x30C4				
0x70C0	`tx_stats_etherStats Fragments`			
0x70C4				
0x30C8	`rx_stats_etherStats Jabbers`	RO	0x0	■ Bit 0—The number of oversized receive frames (frame length more than `rx_frame_maxlength`) with CRC error. This count includes invalid frame types. ■ 36-bit width register: ■ 0x30C8 = bits [31:0] ■ 0x30CC = bits [35:32]
0x30CC				
0x70C8	`tx_stats_etherStats Jabbers`	RO	0x0	■ Bit 0—The number of oversized transmit frames (frame length more than 1,518 bytes) with CRC error. This count includes invalid frame types. ■ 36-bit width register: ■ 0x70C8 = bits [31:0] ■ 0x70CC = bits [35:32]
0x70CC				
0x30D0	`rx_stats_etherStats CRCErr`	RO	0x0	■ Bit 0—The number of receive frames between the length of 64 and the value configured in the `rx_frame_maxlength` register with CRC error. This count includes errored and invalid frames. ■ 36-bit width register: ■ 0x30D0 = bits [31:0] ■ 0x30D4 = bits [35:32]
0x30D4				
0x70D0	`tx_stats_etherStats CRCErr`	RO	0x0	■ Bit 0—The number of transmit frames between the length of 64 and 1,518 bytes. This count includes errored and invalid frames. ■ 36-bit width register: ■ 0x70D0 = bits [31:0] ■ 0x70D4 = bits [35:32]
0x70D4				
0x30D8	`rx_stats_unicastMAC CtrlFrames`	RO	0x0	■ Bit 0—The number of valid unicast control frames received or transmitted. ■ 36-bit width register: ■ 0x30D8 and 0x70D8 = bits [31:0] ■ 0x30DC and 0x70DC = bits [35:32]
0x30DC				
0x70D8	`tx_stats_unicastMAC CtrlFrames`			
0x70DC				

Table 5–2. MAC Registers (Part 14 of 14)

Byte Offset	Register Name	Access	Reset Value	Description
0x30E0	`rx_stats_multicastMAC CtrlFrames`	RO	0x0	■ Bit 0—The number of valid multicast control frames received or transmitted. ■ 36-bit width register: ■ 0x30E0 and 0x70E0 = bits [31:0] ■ 0x30E4 and 0x70E4 = bits [35:32]
0x30E4				
0x70E0	`tx_stats_multicastMAC CtrlFrames`			
0x70E4				
0x30E8	`rx_stats_broadcastMAC CtrlFrames`	RO	0x0	■ Bit 0—The number of valid broadcast control frames received or transmitted. ■ 36-bit width register: ■ 0x30E8 and 0x70E8 = bits [31:0] ■ 0x30EC and 0x70EC = bits [35:32]
0x30EC				
0x70E8	`tx_stats_broadcastMAC CtrlFrames`			
0x70EC				
0x30F0	`rx_stats_PFCMACCtrlFrames`	RO	0x0	■ Bit 0—The number of valid PFC frames received or transmitted. ■ 36-bit width register: ■ 0x30F0 and 0x70F0 = bits [31:0] ■ 0x30F4 and 0x70F4 = bits [35:32]
0x30F4				
0x70F0	`tx_stats_PFCMACCtrlFrames`			
0x70F4				
0x30F8 – 0x3FFF	Reserved	—	—	Reserved for future use.
0x70F8 – 0x7FFF				

Notes to Table 5–2:

(1) When you read the statistic counters, read the LSB before reading the MSB. For example, when you read `rx_stats_PFCMACCtrlFrames`, read the register offset 0x30F0 before reading the register offset 0x30F4.

(2) If you set the statistics counters to memory-based implementation, the number of undersized frames received or transmitted is not incremented for this register. This is due to the limited processing time when undersized frames are received or transmitted.

5.1.1. Rx_frame_control Register

Table 5–3 describes the function of each field in the `rx_framedecoder_control` register.

Table 5–3. Rx_framedecoder_control Register (Part 1 of 2)

Bit	Field Name	Width	Access	Reset Value	Description
0	`EN_ALLUCAST`	1	RW	0x1	0—Drops unicast receive frames using the primary MAC addresses. 1—Accepts all unicast receive frames. Setting this register and the `EN_ALLMCAST` register to 1, enables the MAC to go on promiscuous (transparent) mode.
1	`EN_ALLMCAST`	1	RW	0x1	0—Drops all multicast frames. 1—Accepts all multicast frames. Setting this register and the `EN_ALLUCAST` register to 1, enables the MAC to go on promiscuous (transparent) mode.
2	Reserved	1	—	—	Reserved for future use.

Table 5–3. Rx_framedecoder_control Register (Part 2 of 2)

Bit	Field Name	Width	Access	Reset Value	Description
3	`FWD_CONTROL`	1	RW	0x0	When you turn on the **Priority-based Flow Control (PFC)** parameter, this bit affects all control frames except the IEEE 802.3 pause frames and PFC frames. Otherwise, this bit affects all control frames except the IEEE 802.3 pause frames. 0—Drops control frames. 1—Forwards control frames to the client.
4	`FWD_PAUSE`	1	RW	0x0	0—Drops IEEE 802.3 pause frame after processing them. 1—Forwards IEEE 802.3 pause frames to the client.
5	`IGNORE_PAUSE`	1	RW	0x0	0—Suspends transmission for the value specified by the pause quanta in the IEEE 802.3 pause frame received. 1—Ignores IEEE 802.3 pause frames.
6 – 15	Reserved	—	—	—	Reserved for future use.
16	`EN_SUPP0`	1	RW	0x0	0—Disables the use of supplementary address 0. 1—Enables the use of supplementary address 0.
17	`EN_SUPP1`	1	RW	0x0	0—Disables the use of supplementary address 1. 1—Enables the use of supplementary address 1.
18	`EN_SUPP2`	1	RW	0x0	0—Disables the use of supplementary address 2. 1—Enables the use of supplementary address 2.
19	`EN_SUPP3`	1	RW	0x0	0—Disables the use of supplementary address 3. 1—Enables the use of supplementary address 3.
20 – 31	Reserved	—	—	—	Reserved for future use.

5.1.2. Rx_pfc_control Register

Table 5–4 describes the function of each field in the `rx_pfc_control` register.

Table 5–4. Rx_pfc_control Register

Bit	Field Name	Width	Access	Reset Value	Description
0	`PFC_IGNORE_PAUSE_0`	1	RW	0x1	0—Suspends transmission for Tx priority queue *n* for the period specified by `pfc_pause_quanta_n`. 1—Ignores the PFC pause request for Tx priority queue *n*.
1	`PFC_IGNORE_PAUSE_1`	1	RW	0x1	
2	`PFC_IGNORE_PAUSE_2`	1	RW	0x1	
3	`PFC_IGNORE_PAUSE_3`	1	RW	0x1	
4	`PFC_IGNORE_PAUSE_4`	1	RW	0x1	
5	`PFC_IGNORE_PAUSE_5`	1	RW	0x1	
6	`PFC_IGNORE_PAUSE_6`	1	RW	0x1	
7	`PFC_IGNORE_PAUSE_7`	1	RW	0x1	
8 – 15	Reserved	—	—	—	Reserved for future use.

Table 5–4. Rx_pfc_control Register

Bit	Field Name	Width	Access	Reset Value	Description
16	FWD_PFC	1	RW	0x1	0—Drops the PFC frame after processing it. 1—Forwards the PFC frame to the client.
16 – 31	Reserved	—	—	—	Reserved for future use.

5.2. Register Initialization

Altera offers the following options for the 10G Ethernet solution with the 10G MAC IP core:

- 10G MAC with single data rate (SDR) XGMII
- 10G MAC with double data rate (DDR) XGMII
- 10G MAC with XAUI PHY IP
- 10G MAC with 10GBASE-R PHY IP

To learn more about the 10G MAC with SDR XGMII to DDR XGMII conversion, refer to "SDR XGMII to DDR XGMII Conversion" on page 7–1.

The 10G MAC is configured in promiscuous (transparent) mode at default or after a hard reset. In promiscuous mode, the 2.5G MAC does not perform any MAC address filtering and it is capable of transmitting and receiving all types of Ethernet frames.

Register initialization for the 10GbE MAC design example is mainly performed in the following configurations:

- External PHY Initialization Using MDIO (Optional)
- PHY Configuration Register Initialization
- Miscellaneous Configuration Register Initialization
- MAC Configuration Register Initialization

For more information about the 10GbE MAC design example, refer to the "Design Examples and Testbench"chapter.

To initialize the registers for the 10GbE MAC configuration, it is important for you to understand the usage of the addressing mode. This configuration uses the following addressing modes:

- 10G MAC IP Core—dword addressing
- 10G Ethernet design example—byte addressing

You can easily convert between dword and byte addressing by removing or adding two least significant bits (LSB) in the address. For example, if dword = 0x341, you can add two LSB bits to the byte address conversion to get byte address = 0xD04.

Use the following recommended register initialization sequences for 10GbE MAC design example:

1. External PHY initialization using MDIO

   This is only applicable when you require external PHY transceiver configuration.

```
//Assume:
//External PHY Address (Hardwired) (MDIO_PRTAD): 0x01
//External PHY Device Type (MDIO_DEVAD): 0x01
//External PHY Control Register address (MDIO_REGAD): 0x0000
//MDIO Base Address: 0x00010000
//MDIO Register Byte offset: 0x84 Byte Address, 0x00010084 = 0x00000104
//Read/Write to External PHY Control Register define in MDIO_REGAD
//MDIO Base Address: 0x00010000
//MDIO Register Byte offset: 0x80
Read/write to Byte Address, 0x00010080 = Read/write to PHY Control
Register (Device Address = 0x01, Register Address = 0x0000)
```

2. PHY configuration register initialization

   Altera provides various types of Ethernet PHY such as XAUI and 10GBASE-R PHY. By default, the PHY does not need any configuration register initialization. To ensure the transceiver PHY is operating properly, perform a hard reset to the PHY after a power-up sequence.

```
//Hard Reset the Altera Transceiver PHY
Asserted the phy_mgmt_reset input at least more than 3 phy_mgmt_clk
cycles.
De-assert the phy_mgmt_reset input to release the hard reset

//Wait for the Transceiver PHY Reset Sequence to Complete
Wait the tx_ready and rx_ready outputs = 1
```

Or

```
//Check the tx_read and rx_ready status through PHY Management Interface
//XAUI/10G BASE-R PHY Base Address: 0x00040000
//reset_status byte addres: 0x108
//reset_status bit 0 – tx_ready, bit 1 – rx_ready
Wait reset_status (address = 0x00040108) = 0x3
```

3. Miscellaneous configuration register initialization

   This is only applicable to the 10GbE MAC design example. The following components in the design example is categorized under the miscellaneous configuration register initialization:

- TX and RX single-clock FIFO/dual clock FIFO

   a. Setting for single-clock FIFO

```
//RX FIFO Base Address: 0x00010400
//TX FIFO Base Address: 0x00010600

//Enable Store and Forward Mode in RX Single-Clock FIFO
```

```
//cut_through_threshold byte address: 0x10
//Set this larger than 0 will enable Cut Through mode
cut_throught_threshold (address = 0x00010410) = 0x0

//Enable Store and Forward Mode in TX Single-Clock FIFO
//cut_through_threshold byte address: 0x10
//Set this larger than 0 will enable Cut through mode
cut_through_threshold (address = 0x00010610) = 0x0

//Enable FIFO Frame Drop On Error
//Drop on Error is NOT available in Cut Through Mode
//drop_on_error byte address: 0x14
//Set this to 0 will disable the drop on error
drop_on_error (address = 0x00010414) = 0x1

//Enable Drop On Error in TX Single Clock FIFO
//Drop on Error is NOT available in Cut Through Mode
//drop_on_error byte address: 0x14
//Set this to 0 will disable the drop on error
drop_on_error (address = 0x00010614) = 0x1
```

b. Setting for dual-clock FIFO

Because the drop on error and store and forward features are not supported, you are not required to perform any register initialization.

- Ethernet loopback

```
//Ethernet Loopback Base Address: 0x00010200

//Disable Line Loopback
//line_loopback byte address: 0x00
//Set this to 1 will enable the Line loopback
line_loopback (address = 0x00010200) = 0x0
//Disable Local Loopback
//local_loopback byte address: 0x08
//set this to 1 will eable the local loopback
local_loopback (address = 0x00010208) = 0x0
```

4. MAC configuration register initialization

The 10GbE MAC is configured as promiscuous mode by default; therefore it does not require any initialization to transmit and receive Ethernet frames. Use the following recommended initialization sequences for your configuration:

a. Disable MAC transmit and receive datapath

Disable the 10GbE MAC transmit and receive datapath before changing any configuration register.

```
//Disable the MAC Receive Path
//rx_transfer_control byte address: 0x000
rx_transfer_control (address = 0x00000000) = 0x1

//Disable the MAC Transmit Path
//tx_transfer_control byte address: 0x4000
tx_transfer_control (address = 0x00004000) = 0x1

//Check the MAC Transmit and Receive Path is disable
//rx_transfer_status byte address: 0x004
Wait rx_transfer_status (address = 0x00000004) = 0x1
//tx_transfer_status byte address: 0x4004
Wait tx_transfer_status (address = 0x00004004) = 0x1
```

b. MAC address configuration

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

//Configure the MAC Receive MAC Address
//rx_frame_addr0 byte address: 0x2008
//rx_frame_addr1 byte address: 0x200C
rx_frame_addr0 (address = 0x00002008) = 0x17231C00
rx_frame_addr1 (address = 0x0000200C) = 0x0000CB4A

//Configure the MAC Transmit MAC Address
//tx_addrins_macaddr0 byte address: 0x4804
//tx_addrins_macaddr1 byte address: 0x4808
tx_addrins_macaddr0 (address = 0x00004804) = 0x17231C00
tx_addrins_macaddr1 (address = 0x00004808) = 0x0000CB4A
```

c. MAC function configuration

```
//Maximum Frame Length is 1518 bytes
//rx_frame_maxlength byte address: 0x2004
rx_frame_maxlength (address = 0x00002004) = 1518

//tx_frame_maxlength byte address: 0x6004
tx_frame_maxlength (address = 0x00006004) = 1518

//Maximum Pause Quanta Value for Flow Control
//tx_pauseframe_quanta byte address: 0x4504
tx_pauseframe_quanta (address = 0x00004504) = 0xFFFF

//CRC and Padding Removal for MAC Receive
//rx_padcrc_control byte address: 0x0100
rx_padcrc_control (address = 0x00000100) = 0x3

//Padding Removal for MAC Transmit
//tx_padins_control byte address: 0x4100
tx_padins_control (address = 0x00004100) = 0x1

//CRC Removal for MAC Transmit
//tx_crcins_control byte address: 0x4200
tx_crcins_control (address = 0x00004200) = 0x3

//TX MAC Address Insertion on Transmit Frame
//tx_addrins_control byte address: 0x4800
tx_addrins_control (address = 0x00004800) = 0x1

//Configure the RX Frame Control Register
//Disable the promiscuous (transparent) mode by setting EN_ALLUCAST bit
to 0
//rx_frame_control byte address: 0x2000
rx_frame_control (address = 0x00002000) = 0x00000002
```

Figure 5–1 shows the settings for the `rx_frame_control` register.

Figure 5–1. Fx_frame_control Register Settings

30..20	19	18	17	16	15..6	5	4	3	2	1	0
Reserved	EN_SUPP3	EN_SUPP2	EN_SUPP1	EN_SUPP0	Reserved	IGNORE_PAUSE	FWD_PAUSE	FWD_CONTROL	Reserved	EN_ALLMCAST	EN_ALLUCAST
0..0	0	0	0	0	0..0	0	0	0	0	1	0

d. Enable MAC transmit and receive datapath.

```
//Enable the MAC Receive Path
//rx_transfer_control byte address: 0x000
rx_transfer_control (address = 0x00000000) = 0x0

//Enable the MAC Transmit Path
//tx_transfer_control byte address: 0x4000
tx_transfer_control (address = 0x00004000) = 0x0

//Check the Transmit and Receive Path is enable
//rx_transfer_status byte address: 0x004
Wait rx_transfer_status (address = 0x00000004) = 0x0

//tx_transfer_status byte address: 0x4004
Wait tx_transfer_status (address = 0x00004004) = 0x0
```

6. Interface Signals

This section describes the interface signals in all MAC variations.

Figure 6–1 shows the interface signals for the MAC Tx and Rx variation.

Figure 6–1. 10GbE MAC

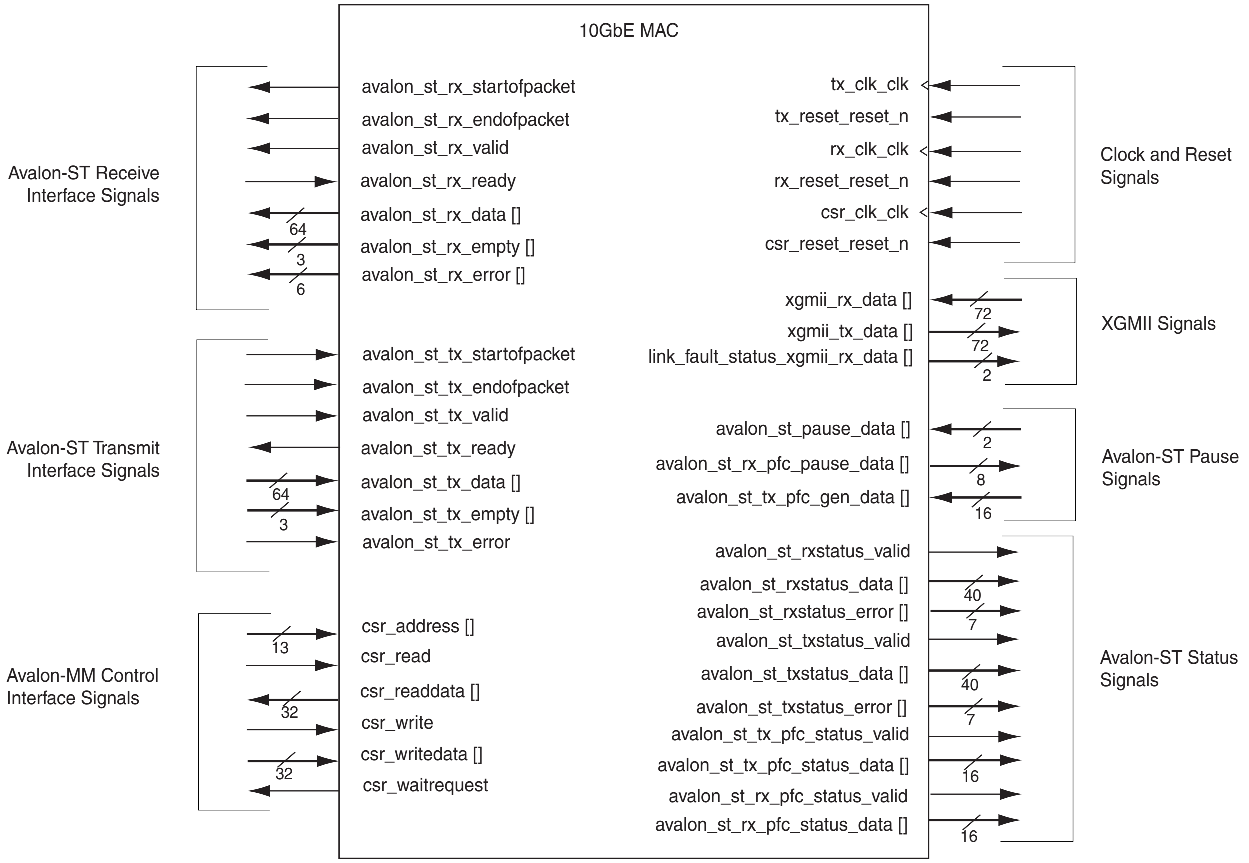


Figure 6–2 shows the interfaces signals for the MAC Tx only variation.

Figure 6–2. MAC Tx Only Variation

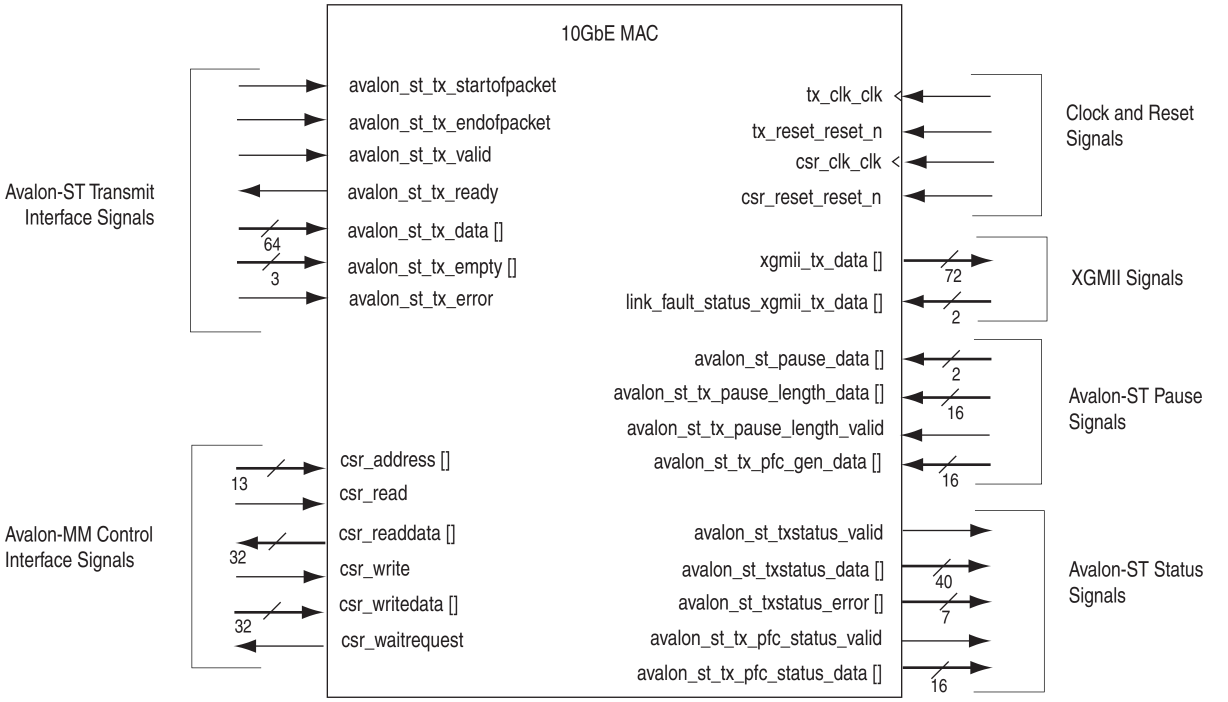
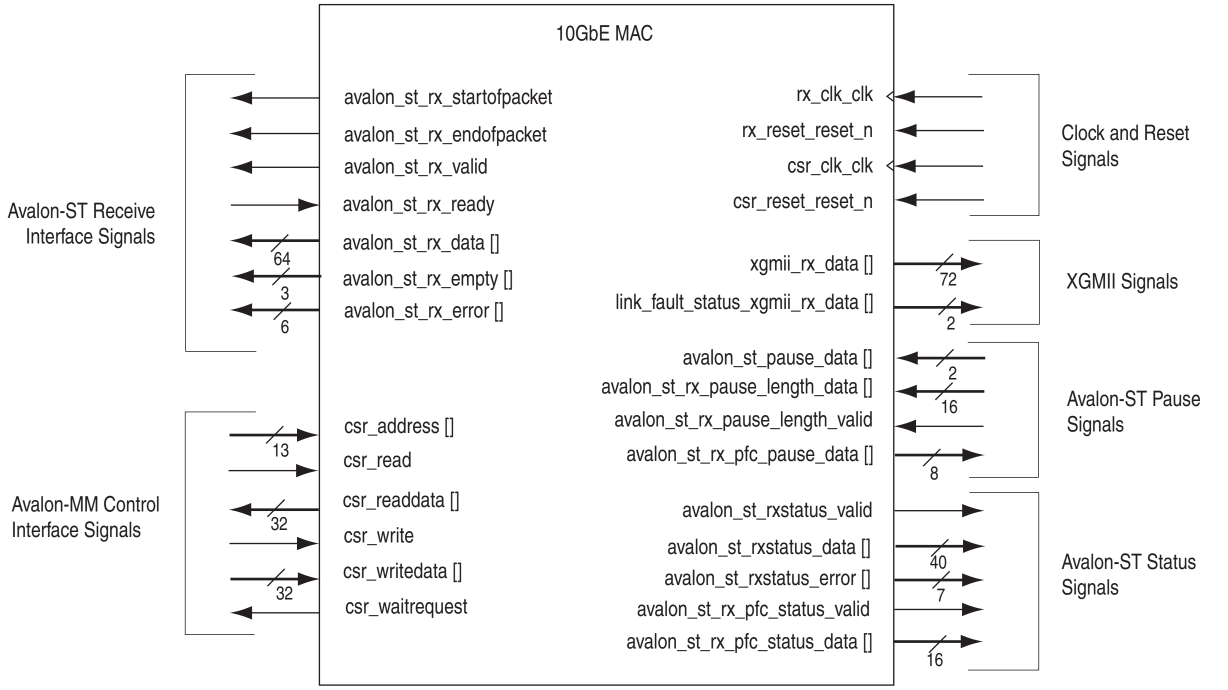


Figure 6–3 shows the interface signals for the MAC Rx only variation.

Figure 6–3. MAC Rx Only Variation



6.0.1. Clock and Reset Signals

The MAC operates in multiple clock domains. You can use different sources to drive the clock and reset interfaces. Refer to Table 6–1 on page 6–3 for the clock and timing requirements for the clock and reset interfaces.

Table 6–1 lists the MAC clock and reset signals.

Table 6–1. Common Clock and Reset Signals

Signal	Direction	Width	Description
tx_clk_clk *(1)*	Input	1	156.25-MHz transmit clock. Provides the timing reference for the Avalon-ST transmit interface.
tx_reset_reset_n	Input	1	An active-low asynchronous reset signal for the tx_clk_clk domain. The MAC function implements a reset synchronizer to generate a synchronous signal.
rx_clk_clk *(1)*	Input	1	156.25-MHz receive clock. Provides the timing reference for the Avalon-ST receive interface.
rx_reset_reset_n	Input	1	An active-low asynchronous reset signal for the rx_clk_clk domain. The MAC function implements a reset synchronizer to generate a synchronous signal.
csr_clk_clk	Input	1	Configuration clock for the control and status interface. The clock runs at 156.25-MHz or lower.
csr_reset_reset_n	Input	1	An active-low reset signal for the control and status interface.

Note to Table 6–1:

(1) You can use the same clock source for both tx_clk_clk and rx_clk_clk.

6.0.2. Avalon-ST Transmit and Receive Interface Signals

Table 6–2 describes the Avalon-ST transmit signals.

Table 6–2. Avalon-ST Transmit Signals

Signal	Direction	Width	Description
avalon_st_tx_startofpacket	Input	1	Assert this signal to indicate the beginning of the transmit packet.
avalon_st_tx_endofpacket	Input	1	Assert this signal to indicate the end of the transmit packet.
avalon_st_tx_valid	Input	1	Assert this signal to qualify the transmit data on the avalon_st_tx_data bus.
avalon_st_tx_ready	Output	1	When asserted, this signal indicates that the IP core is ready to accept data.
avalon_st_tx_data[]	Input	64	Carries the transmit data from the client.
avalon_st_tx_empty[]	Input	3	Use this signal to specify the number of bytes that are empty (not used) during cycles that contain the end of a packet.
avalon_st_tx_error	Input	1	Assert this signal to indicate the current receive packet contains errors.

Table 6–3 describes the Avalon-ST receive signals.

Table 6–3. Avalon-ST Receive Signals

Signal	Direction	Width	Description
avalon_st_rx_startofpacket	Output	1	When asserted, this signal indicates the beginning of the receive packet.
avalon_st_rx_endofpacket	Output	1	When asserted, this signal indicates the end of the receive packet.
avalon_st_rx_valid	Output	1	When asserted, this signal qualifies the receive data on the avalon_st_rx_data bus.
avalon_st_rx_ready	Input	1	Assert this signal when the client is ready to accept data.
avalon_st_rx_data[]	Output	64	Carries the receive data to the client.
avalon_st_rx_empty[]	Output	3	Contains the number of bytes that are empty (not used) during cycles that contain the end of a packet.
avalon_st_rx_error[]	Output	6	When set to 1, the respective bits in this signal indicate an error type in the receive frame: ■ Bit 0: PHY error—The XGMII Rx interface data bus (xgmii_rx_data) contains a control error character (FE). ■ Bit 1: CRC error—The calculated CRC value differs from the received CRC. ■ Bit 2: Undersized frame—The frame size is less than 64 bytes. ■ Bit 3: Oversized frame—The frame size is more than MAX_FRAME_SIZE. ■ Bit 4: Payload length error—The actual frame payload length differs from the length/type field. ■ Bit 5: Overflow error—The FIFO buffer is full while it is receiving signal from the MAC causing truncated receive frame. The IP core presents the error type on this bus in the same clock cycle it asserts avalon_st_rx_endofpacket and avalon_st_rx_valid.

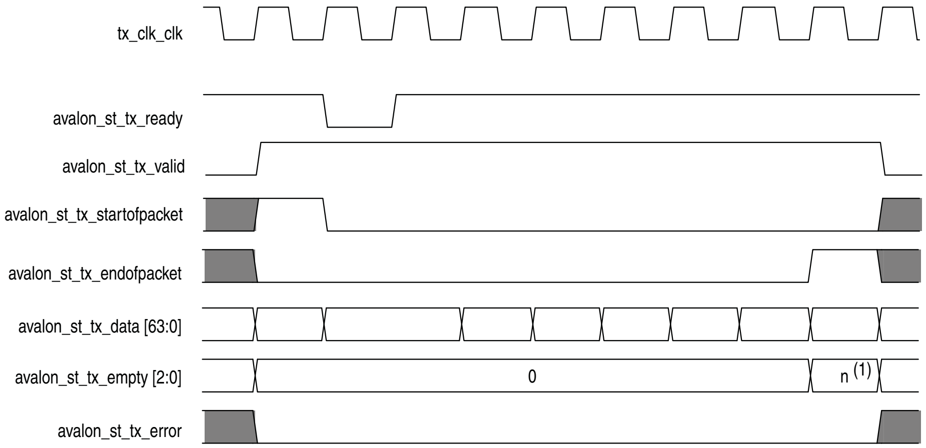
6.0.2.1. Timing Diagrams—Avalon-ST Transmit Interface

The diagrams in this section shows the timing and the mapping on the Avalon-ST transmit interface.

The client asserts the avalon_st_tx_startofpacket signal to indicate the beginning of the transmit packet. On the same rising edge of tx_clk_clk, the client asserts the avalon_st_tx_valid signal to qualify the transmit data on the avalon_st_tx_data[63:0] bus. At the end of the packet, the avalon_st_tx_empty [2:0] signal specifies the number of bytes that are empty.

Figure 6–4 shows the timing for the Avalon-ST transmit interface with a good frame.

Figure 6–4. Avalon-ST Transmit Interface



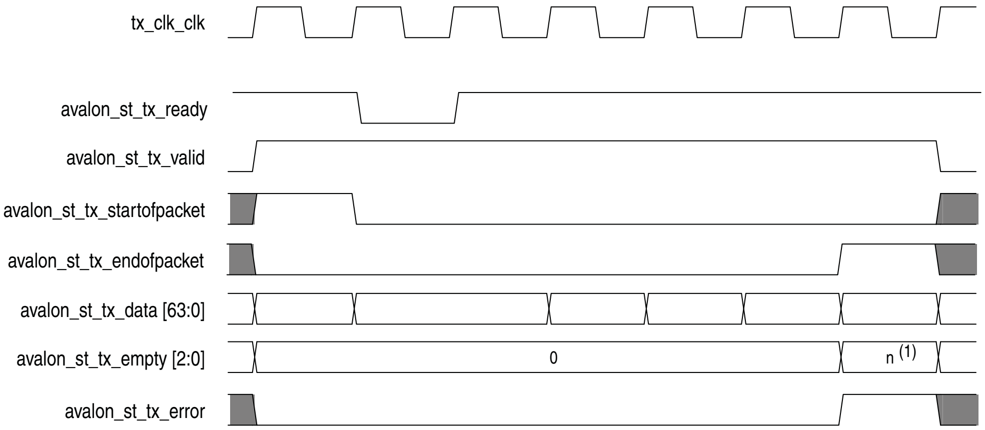
Note to Figure 6–4:

(1) *n* indicates the number of symbols that are empty during the cycles that mark the end of a frame.

When the client forwards an error frame to the Avalon-ST transmit interface, the client asserts the `avalon_st_tx_error` signal to indicate errors in the current frame. The `avalon_st_tx_error` signal is aligned with the `avalon_st_tx_endofpacket` signal.

Figure 6–5 shows the timing for the Avalon-ST transmit interface with an error frame.

Figure 6–5. Avalon-ST Transmit Interface with Error

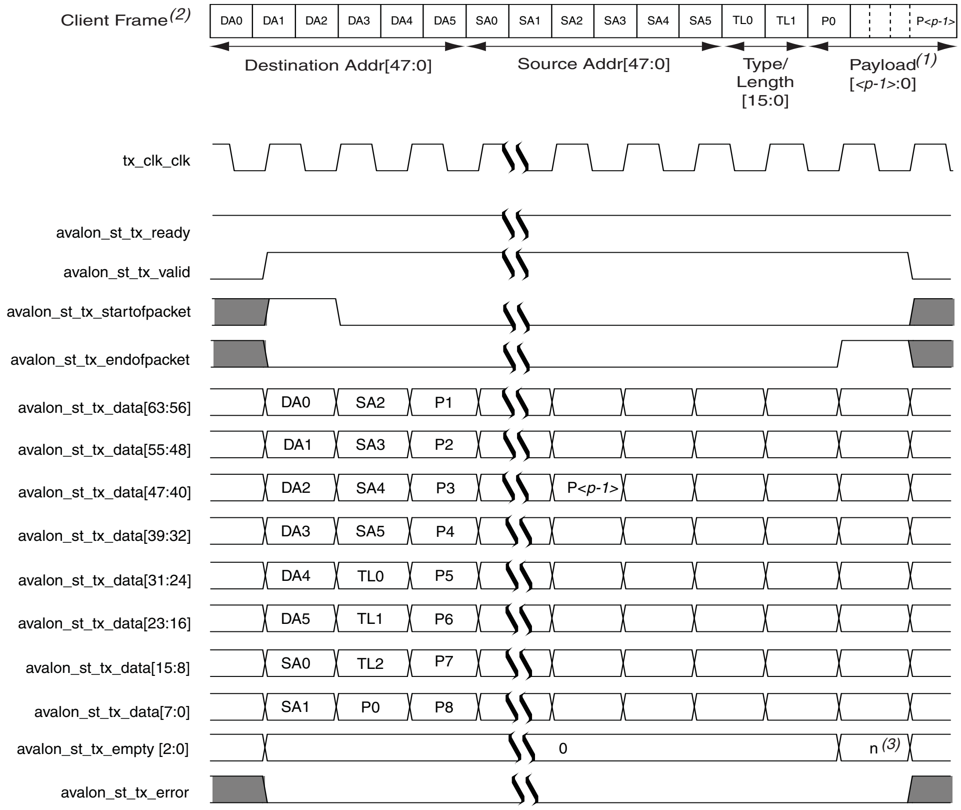


Note to Figure 6–5:

(1) *n* indicates the number of symbols that are empty during the cycles that mark the end of a frame.

Figure 6–6 shows the mapping of the client frame on the Avalon-ST transmit interface.

Figure 6–6. Mapping of Client Frame to Avalon-ST Transmit Interface



Notes to Figure 6–6:

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

(2) In the preamble passthrough mode, the client frame starts with an 8-byte client-defined preamble.

(3) *n* indicates the number of symbols that are empty during the cycles that mark the end of a frame.

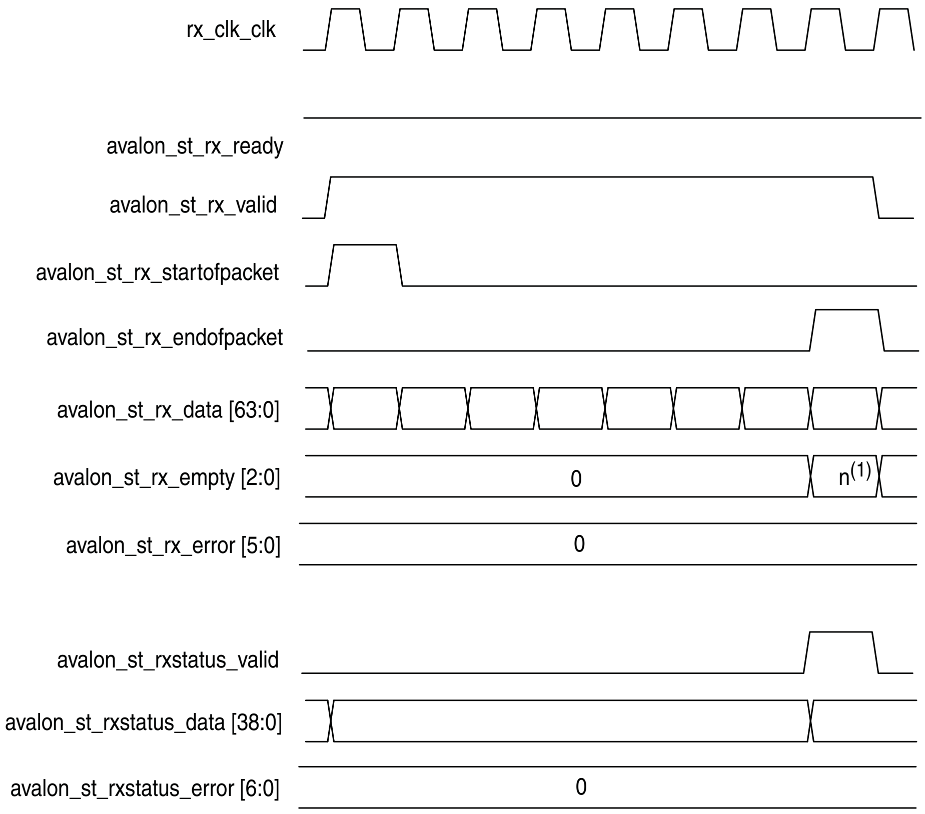
6.0.2.2. Timing Diagrams—Avalon-ST Receive Interface

The diagrams in this section show the timing on the Avalon-ST receive interface.

The Avalon-ST receive interface `avalon_st_rx_startofpacket` signal is asserted to indicate the start of a new frame. On the same rising edge of `rx_clk_clk`, the `avalon_st_rx_valid` signal is also asserted to qualify the transmit data on the `avalon_st_rx_data[63:0]` bus. The end of the receive packet is indicated by the `avalon_st_rx_endofpacket` signal.

Figure 6–7 shows the timing for the Avalon-ST receive interface with a good frame.

Figure 6–7. Avalon-ST Receive



Note to Figure 6–7:

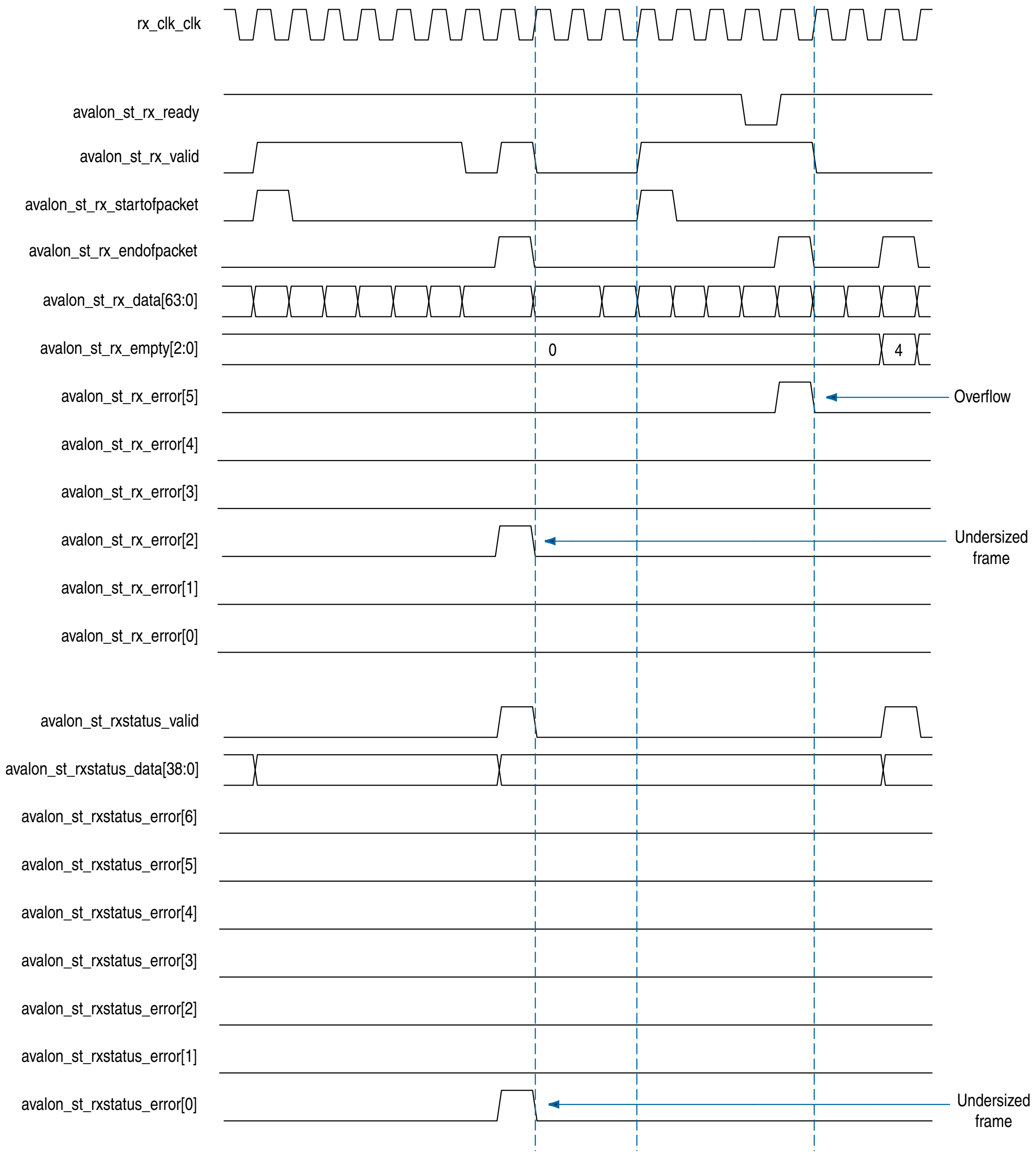
(1) *n* indicates the number of symbols that are empty during the cycles that mark the end of a frame.

When the MAC Rx receives an undersized frame, it sets the `avalon_st_rx_error[2]` bit to 1. When an overflow occurs, the MAC Rx sets the `avalon_st_rx_error[5]` bit to 1 and deasserts the `avalon_st_rx_ready` signal to backpressure the Avalon-ST receive interface. The error signals are sampled when `avalon_st_rx_endofpacket` and `avalon_st_rx_valid` signals are asserted.

For more information about the error signals in the Avalon-ST receive and status interface, refer to Table 6–3 on page 6–4 and Table 6–6 on page 6–12.

Figure 6–8 shows the reception of a 60-byte frame at the Avalon-ST receive interface when an error occurs with an overflow and undersized frame condition.

Figure 6–8. Avalon-ST Receive with Error (Overflow and Undersized Frame)



6.0.3. SDR XGMII Signals

Table 6–4 shows the SDR XGMII signals.

Table 6–4. SDR XGMII Signals

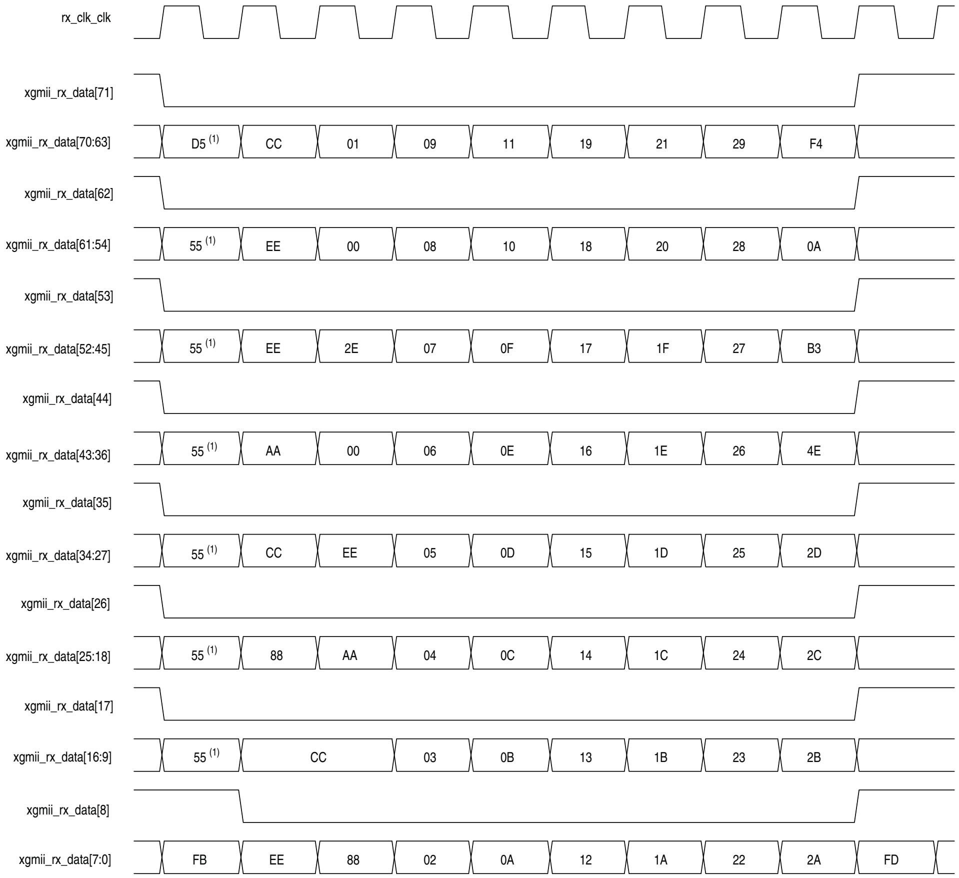
Signal	Direction	Width	Description
`xgmii_rx_data[]`	Input	72	8-lane data bus carrying 8-bit data and 1-bit control information. Lane 0 starts from the least significant bit. `xgmii_rx_data`[7:0] / `xgmii_tx_data`[7:0] = data `xgmii_rx_data`[8] / `xgmii_tx_data`[8] = control `xgmii_rx_data`[16:9] / `xgmii_tx_data`[16:9] = data `xgmii_rx_data`[17] / `xgmii_tx_data`[17] = control ... and so forth
`xgmii_tx_data[]`	Output	72	
`link_fault_status_xgmii_rx_data[]`	Output	2	Indicates the link fault status from the RS Rx to the RS Tx. ■ 00 = No link fault. ■ 01 = Local fault. ■ 10 = Remote fault. When you instantiate the MAC Rx only variation, connect this signal to the corresponding Tx client logic to handle the local and remote faults.
`link_fault_status_xgmii_tx_data[]`	Input	2	This signal is present only in the MAC Tx only variation. Connect this signal to the corresponding Rx client logic to handle the local and remote faults. Indicates the link fault status to the RS Tx. ■ 00 = No link fault ■ 01 = Local fault ■ 10 = Remote fault

6.0.3.1. Timing Diagrams—SDR XGMII

The diagrams in this section show the timing for the SDR XGMII.

Figure 6–9 shows the timing for the SDR XGMII Rx interface data bus.

Figure 6–9. SDR XGMII Rx Interface Data Bus



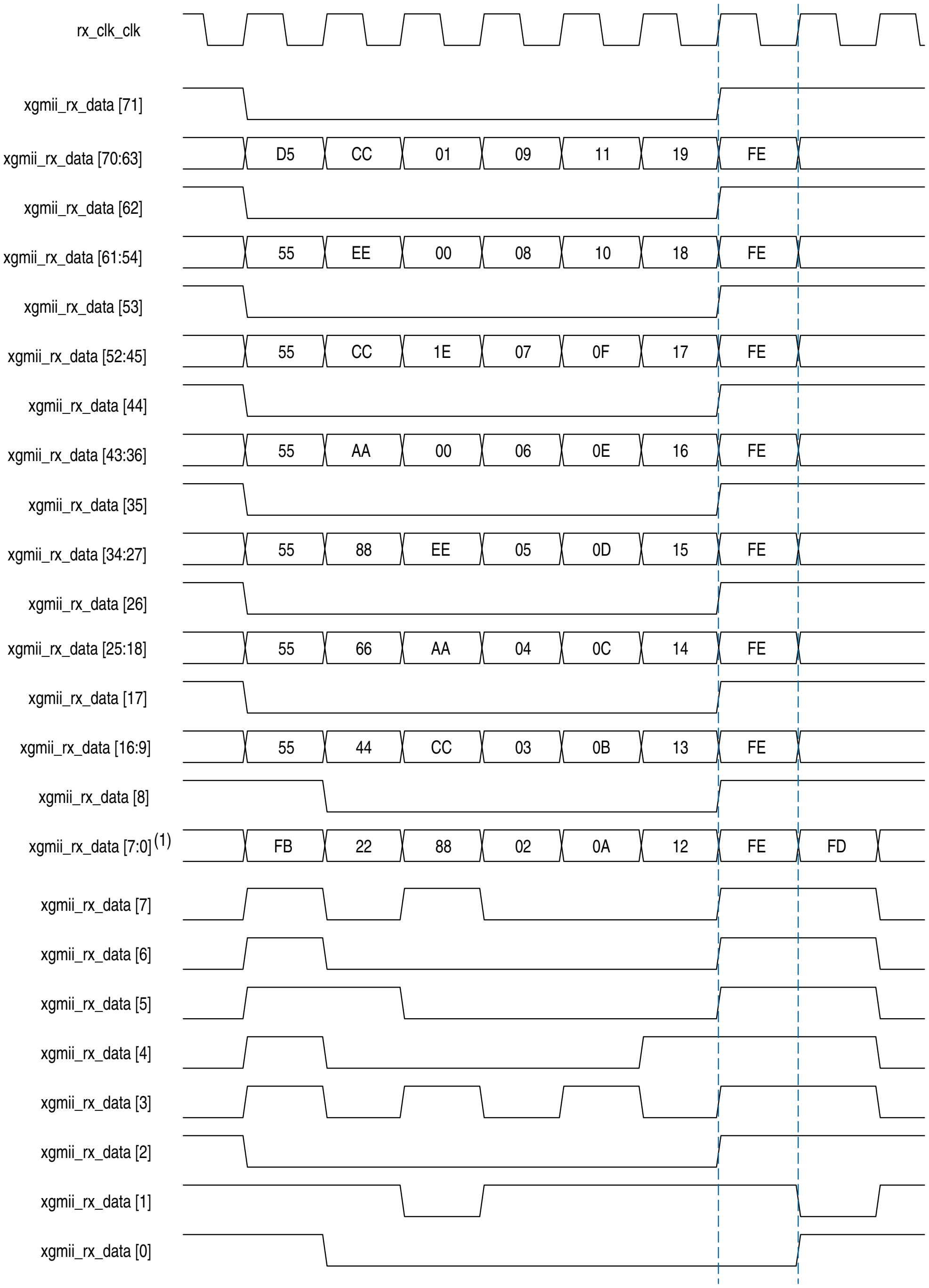
Note to Figure 6–9:

(1) In the preamble passthrough mode, the MAC Tx frame starts with a 1-byte START and a 7-byte client-defined preamble.

When an error occurs, the control bit signal is asserted and the data during that clock cycle is replaced by a control error character (FE).

Figure 6–10 shows the timing for the SDR XGMII Rx interface when an error occurs.

Figure 6–10. SDR XGMII Rx Interface with Error



Note to Figure 6–10:

(1) The `xgmii_rx_data[7:0]` bus is expanded to show the behavior of each signal when an error occurs.

6.0.4. Avalon-MM Programming Interface Signals

Table 6–5 describes the Avalon-MM programming interface signals.

Table 6–5. Avalon-MM CSR Interface Signals

Signal	Direction	Width	Description
`csr_address[]`	Input	13	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.
`csr_waitrequest`	Output	1	When asserted, this signal indicates that the IP core is busy and not ready to accept any read or write requests.

6.0.5. Avalon-ST Status and Pause Interface Signals

Table 6–6 describes the Avalon-ST status signals.

☞ Use the Avalon-ST status interface to obtain information and error status on receive frames only when the option to remove CRC and/or padding is disabled and no overflow occurs. When CRC and/or padding removal is enabled or when an overflow occurs (`avalon_st_rx_ready` is deasserted), obtain the same information using the statistics counters.

Table 6–6. Avalon-ST Status Interface Signals (Part 1 of 5)

Signal	Direction	Width	Description
`avalon_st_rxstatus_valid`	Output	1	When asserted, this signal indicates that `avalon_st_rxstatus_data[]` contains valid information about the receive frame. The IP core asserts this signal in the same clock cycle it receives the end of packet (`avalon_st_rx_endofpacket` is asserted).

Table 6–6. Avalon-ST Status Interface Signals (Part 2 of 5)

Signal	Direction	Width	Description
`avalon_st_rxstatus_data[]`	Output	40	Contains information about the receive frame: ■ Bits 0 to 15: Payload length. ■ Bits 16 to 31: Packet length. ■ Bit 32: When set to 1, indicates a stacked VLAN frame. ■ Bit 33: When set to 1, indicates a VLAN frame. ■ Bit 34: When set to 1, indicates a control frame. ■ Bit 35: When set to 1, indicates a pause frame. ■ Bit 36: When set to 1, indicates a broadcast frame. ■ Bit 37: When set to 1, indicates a multicast frame. ■ Bit 38: When set to 1, indicates a unicast frame. ■ Bit 39: When set to 1, indicates a PFC frame. The IP core presents the valid information on this bus in the same clock cycle it asserts `avalon_st_rxstatus_valid`. The information on this data bus is invalid when an overflow occurs or when CRC and/or padding removal is enabled.
`avalon_st_rxstatus_error[]`	Output	7	When set to 1, each bit of this signal indicates an error type in the receive frame. ■ Bit 0: Undersized frame. ■ Bit 1: Oversized frame. ■ Bit 2: Payload length error. ■ Bit 3: CRC error. ■ Bit 4: Unused. ■ Bit 5: Unused. ■ Bit 6: PHY error. The IP core presents the error status on this bus in the same clock cycle it asserts `avalon_st_rxstatus_valid`. The error status is invalid when an overflow occurs or when CRC and/or padding removal is enabled.
`avalon_st_txstatus_valid`	Output	1	When asserted, this signal indicates that `avalon_st_txstatus_data[]` contains valid information about the transmit frame.

Table 6–6. Avalon-ST Status Interface Signals (Part 3 of 5)

Signal	Direction	Width	Description
avalon_st_txstatus_data[]	Output	40	Contains information about the transmit frame: ■ Bits 0 to 15: Payload length. ■ Bits 16 to 31: Packet length. ■ Bit 32: When set to 1, indicates a stacked VLAN frame. ■ Bit 33: When set to 1, indicates a VLAN frame. ■ Bit 34: When set to 1, indicates a control frame. ■ Bit 35: When set to 1, indicates a pause frame. ■ Bit 36: When set to 1, indicates a broadcast frame. ■ Bit 37: When set to 1, indicates a multicast frame. ■ Bit 38: When set to 1, indicates a unicast frame. ■ Bit 39: When set to 1, indicates a PFC frame. The IP core asserts the valid information on this bus in the same clock cycle it asserts avalon_st_txstatus_valid. The information on this data bus is invalid when an overflow occurs or when CRC and/or padding insertion is enabled.
avalon_st_txstatus_error[]	Output	7	When set to 1, each bit of this signal indicates an error type in the receive frame. ■ Bit 0: Undersized frame. ■ Bit 1: Oversized frame. ■ Bit 2: Payload length error. ■ Bit 3: Unused. ■ Bit 4: Underflow. ■ Bit 5: Client error. ■ Bit 6: Unused. The IP core presents the error status on this bus in the same clock cycle it asserts avalon_st_txstatus_valid. The error status is invalid when an overflow occurs or when CRC and/or padding removal is enabled.
avalon_st_tx_pfc_status_valid *(1)*	Output	1	When asserted, this signal qualifies the data on the avalon_st_tx_pfc_status_data bus.

Table 6–6. Avalon-ST Status Interface Signals (Part 4 of 5)

Signal	Direction	Width	Description
avalon_st_tx_pfc_status_data[] *(1)*	Output	*n*	The signal width is determined by the **Number of PFC Priority** parameter, *n* = 2 x number priority queues enabled. ■ Bit 0: When asserted, this bit indicates that an XON request is transmitted for priority queue 0. ■ Bit 1: When asserted, this bit indicates that an XOFF request is transmitted for priority queue 0. ■ Bit 2: When asserted, this bit indicates that an XON request is transmitted for priority queue 1. ■ Bit 3: When asserted, this bit indicates that an XOFF request is transmitted for priority queue 1. ■ Bit 4:When asserted, this bit indicates that an XON request is transmitted for priority queue 2. ■ Bit 5: When asserted, this bit indicates that an XOFF request is transmitted for priority queue 2. ■ Bit 6: When asserted, this bit indicates that an XON request is transmitted for priority queue 3. ■ Bit 7: When asserted, this bit indicates that an XOFF request is transmitted for priority queue 3. ■ Bit 8: When asserted, this bit indicates that an XON request is transmitted for priority queue 4. ■ Bit 9: When asserted, this bit indicates that an XOFF request is transmitted for priority queue 4. ■ Bit 10: When asserted, this bit indicates that an XON request is transmitted for priority queue 5. ■ Bit 11: When asserted, this bit indicates that an XOFF request is transmitted for priority queue 5. ■ Bit 12: When asserted, this bit indicates that an XON request is transmitted for priority queue 6. ■ Bit 13: When asserted, this bit indicates that an XOFF request is transmitted for priority queue 6. ■ Bit 14: When asserted, this bit indicates that an XON request is transmitted for priority queue 7. ■ Bit 15: When asserted, this bit indicates that an XOFF request is transmitted for priority queue 7.
avalon_st_rx_pfc_status_valid *(1)*	Output	1	When asserted, this signal qualifies the data on the avalon_st_rx_pfc_status_data bus.

Table 6–6. Avalon-ST Status Interface Signals (Part 5 of 5)

Signal	Direction	Width	Description
avalon_st_rx_pfc_status_data[] *(1)*	Output	*n*	The signal width is determined by the **Number of PFC Priority** parameter, *n* = 2 x number priority queues enabled. ■ Bit 0: When asserted, this bit indicates that an XON request is received for priority queue 0. ■ Bit 1: When asserted, this bit indicates that an XOFF request is received for priority queue 0. ■ Bit 2: When asserted, this bit indicates that an XON request is received for priority queue 1. ■ Bit 3: When asserted, this bit indicates that an XOFF request is received for priority queue 1. ■ Bit 4:When asserted, this bit indicates that an XON request is received for priority queue 2. ■ Bit 5: When asserted, this bit indicates that an XOFF request is received for priority queue 2. ■ Bit 6: When asserted, this bit indicates that an XON request is received for priority queue 3. ■ Bit 7: When asserted, this bit indicates that an XOFF request is received for priority queue 3. ■ Bit 8: When asserted, this bit indicates that an XON request is received for priority queue 4. ■ Bit 9: When asserted, this bit indicates that an XOFF request is received for priority queue 4. ■ Bit 10: When asserted, this bit indicates that an XON request is received for priority queue 5. ■ Bit 11: When asserted, this bit indicates that an XOFF request is received for priority queue 5. ■ Bit 12: When asserted, this bit indicates that an XON request is received for priority queue 6. ■ Bit 13: When asserted, this bit indicates that an XOFF request is received for priority queue 6. ■ Bit 14: When asserted, this bit indicates that an XON request is received for priority queue 7. ■ Bit 15: When asserted, this bit indicates that an XOFF request is received for priority queue 7.

Note to Table 6–6:

(1) The signal is included only when you turn on the **Priority-based Flow Control (PFC)** parameter.

Table 6–7 describes the Avalon-ST flow control signals.

Table 6–7. Avalon-ST Flow Control Signals (Part 1 of 3)

Signal	Direction	Width	Description
`avalon_st_pause_data[]`	Input	2	Assert this signal to generate pause frames: ■ Bit 0: Set to 1 to generate an XON pause frame. ■ Bit 1: Set to 1 to generate an XOFF pause frame. You can also use the `tx_pauseframe_control` register to generate pause frames. The register takes precedence over this signal.
`avalon_st_rx_pfc_pause_data[]` *(1)*	Output	2-8	The signal width is determined by the **Number of PFC Priority** parameter, *n* = number priority queues enabled. The MAC Rx asserts bit *n* when the Pause Quanta *n* field in the PFC frame is valid (Pause Quanta Enable [*n*] = 1) and greater than 0. For each pause quanta unit, the MAC Rx asserts bit *n* for eight clock cycle. The MAC Rx deasserts bit *n* when the Pause Quanta *n* field in the PFC frame is valid (Pause Quanta Enable [*n*] = 1) and equal to 0. The MAC Rx also deasserts this signal when the timer expires.

Table 6–7. Avalon-ST Flow Control Signals (Part 2 of 3)

Signal	Direction	Width	Description
`avalon_st_tx_pfc_gen_data[]` *(1)*	Input	4-16	The signal width is determined by the **Number of PFC Priority** parameter, *n* = 2 x number priority queues enabled. ■ Bit 0: Set this bit to 1 to trigger an XON request for priority queue 0. ■ Bit 1: Set this bit to 1 to trigger an XOFF request for priority queue 0. ■ Bit 2: Set this bit to 1 to trigger an XON request for priority queue 1. ■ Bit 3: Set this bit to 1 to trigger an XOFF request for priority queue 1. ■ Bit 4:Set this bit to 1 to trigger an XON request for priority queue 2. ■ Bit 5: Set this bit to 1 to trigger an XOFF request for priority queue 2. ■ Bit 6: Set this bit to 1 to trigger an XON request for priority queue 3. ■ Bit 7: Set this bit to 1 to trigger an XOFF request for priority queue 3. ■ Bit 8: Set this bit to 1 to trigger an XON request for priority queue 4. ■ Bit 9: Set this bit to 1 to trigger an XOFF request for priority queue 4. ■ Bit 10: Set this bit to 1 to trigger an XON request for priority queue 5. ■ Bit 11: Set this bit to 1 to trigger an XOFF request for priority queue 5. ■ Bit 12: Set this bit to 1 to trigger an XON request for priority queue 6. ■ Bit 13: Set this bit to 1 to trigger an XOFF request for priority queue 6. ■ Bit 14: Set this bit to 1 to trigger an XON request for priority queue 7. ■ Bit 15: Set this bit to 1 to trigger an XOFF request for priority queue 7. If you simultaneously assert both bits corresponding to priority queue *n*, neither the XOFF request nor the XON request is generated.
`avalon_st_tx_pause_length_data[]`	Input	16	This signal is present only in the MAX Tx only variation. Specifies the pause duration when a pause frame is received on the Tx path. The pause length is in unit of pause quanta, where 1 pause length = 512 bits time.
`avalon_st_tx_pause_length_valid`	Input	1	This signal is present only in the MAX Tx only variation. When asserted, this signal qualifies the data on the `avalon_st_tx_pause_length_data` bus.

Table 6–7. Avalon-ST Flow Control Signals (Part 3 of 3)

Signal	Direction	Width	Description
`avalon_st_rx_pause_length_data[]`	Output	16	This signal is present only in the MAX Rx only variation. Specifies the pause duration when a pause frame is sent to the Tx path. The pause length is in unit of pause quanta, where 1 pause length = 512 bits time.
`avalon_st_rx_pause_length_valid`	Output	1	This signal is present only in the MAX Rx only variation. When asserted, this signal qualifies the data on the `avalon_st_rx_pause_length_data` bus.

Note to Table 6–7:

(1) The signal is present only when you turn on the **Priority-based Flow Control (PFC)** parameter.

7. Design Considerations

7.1. SDR XGMII to DDR XGMII Conversion

The MAC implements 64-bit SDR XGMII Tx and Rx interfaces with a frequency of 156.25 MHz. The XGMII as defined by IEEE 802.3-2005 standard is a 32-bit DDR interface with a frequency of 156.25 MHz.

If you want to use the MAC with a PHY IP core and connect it to an external device, convert the XGMII from 64-bit SDR (156.25 MHz) to 32-bit DDR (156.25 MHz) or vice versa by connecting the MAC to the Altera DDR I/O (ALTDDIO) megafunctions. The ALTDDIO megafunctions includes the following features:

- ALTDDIO_IN megafunction—Implements the Rx interface for DDR inputs to convert XGMII DDR to SDR frame format.
- ALTDDIO_OUT megafunction—Implements the Tx interface for DDR outputs to convert XGMII SDR to DDR frame format.

7.1.1. ALTDDIO_IN Megafunction Configuration

Use the MegaWizard Plug-in Manager to instantiate the ALTDDIO_IN megafunction and specify the initial parameters. Set the data bus width to 36 bits and apply the following signal connections:

- `xgmii_sdr[35:0]` to `dataout_l[35:0]`
- `xgmii_sdr[71:36]` to `dataout_h[35:0]`

7.1.2. ALTDDIO_OUT Megafunction Configuration

Use the MegaWizard Plug-in Manager to instantiate the ALTDDIO_OUT megafunction and specify the initial parameters. Set the data bus width to 36 bits and apply the following signal connections:

- `xgmii_sdr[35:0]` to `datain_l[35:0]`
- `xgmii_sdr[71:36]` to `datain_h[35:0]`

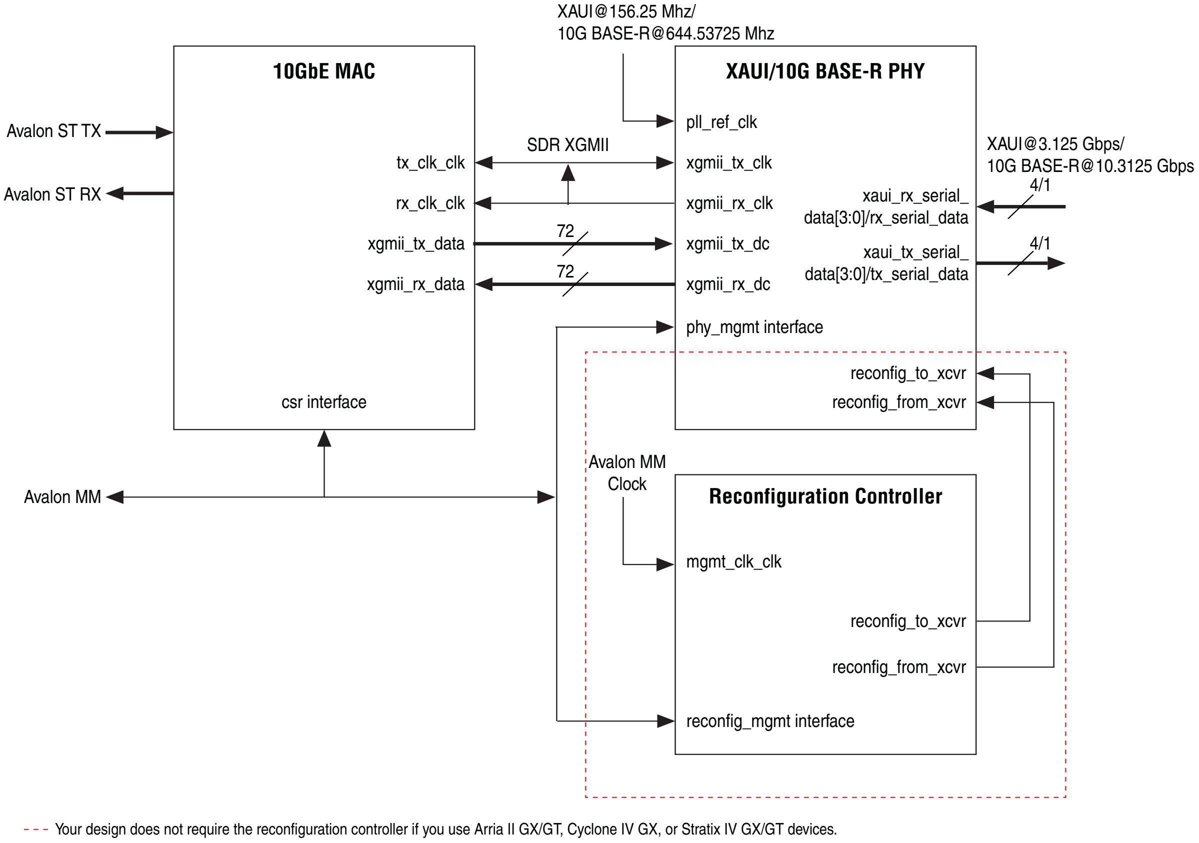
For more information about the ALTDDIO megafunction ports and parameters, refer to the *ALTDDIO Megafunction User Guide*.

7.2. 10GbE MAC and PHY Connection with XGMII

The XGMII is defined by the IEEE802.3 standard. XGMII is the standard interface between the MAC and PHY in the 10G Ethernet solution. Altera 10G MAC and PHY connect easily using the SDR XGMII interface.

Figure 7–1 shows an example of an SDR XGMII connection between the 10G MAC and PHY IP.

Figure 7–1. 10G MAC and PHY Connection with XGMII Interface

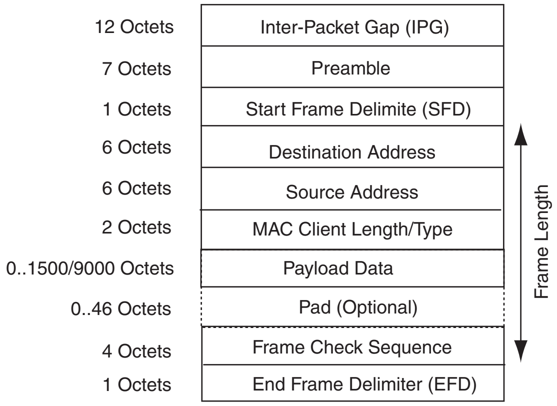


A. Frame Format

A.1. Ethernet Frame

Figure A–1 shows the Ethernet frame format.

Figure A–1. Ethernet Frame Format



12 Octets	Inter-Packet Gap (IPG)
7 Octets	Preamble
1 Octets	Start Frame Delimite (SFD)
6 Octets	Destination Address
6 Octets	Source Address
2 Octets	MAC Client Length/Type
0..1500/9000 Octets	Payload Data
0..46 Octets	Pad (Optional)
4 Octets	Frame Check Sequence
1 Octets	End Frame Delimiter (EFD)

The Ethernet frame comprises the following fields:

- Inter-packet gap (IPG)—an average inter-frame length of 12 octets and is represented with the Idle control character which consists of data value 0x07.
- Preamble—inserted by the MAC or the client. MAC-inserted preamble is a maximum of 7 octets of data with value 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.
- End frame delimiter (EFD)—a 1-octet fixed value of 0xFD which marks the end of a frame.

A.2. VLAN and Stacked VLAN Tagged MAC Frame

The extension of a basic frame is a VLAN tagged frame, which contains an additional VLAN tag field between the source address and length/type fields. VLAN tagging is defined by the IEEE 802.1Q standard. VLANs can identify and separate many groups' network traffic in enterprises and metro networks.VLAN tagged frames have a maximum length of 1522 bytes, excluding the preamble and the SFD bytes. In carrier Ethernet network applications based on IEEE 802.1ad provider bridge standard (QinQ) for scaling the network, frames can be tagged with two consecutive VLAN tags (stacked VLAN). Stacked VLAN frames contain an additional 8-byte field between the source address and length/type fields.

Figure A–2 shows the VLAN frame format.

Figure A–2. VLAN Frame Format

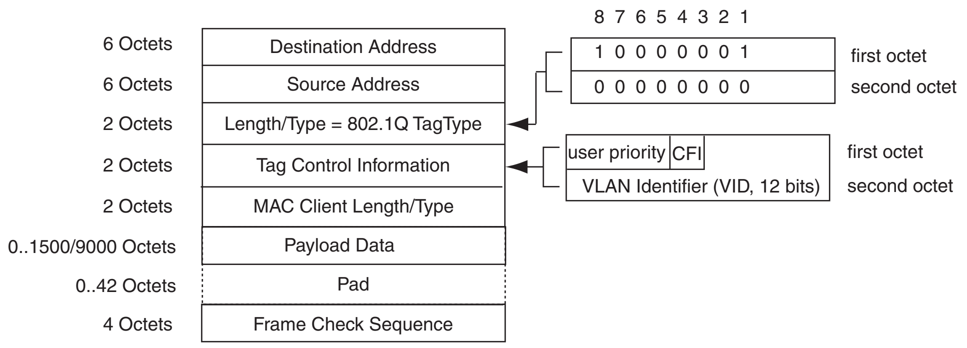
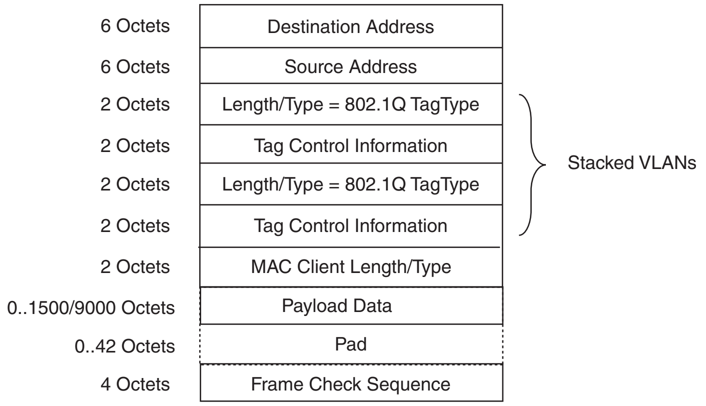


Figure A–3 shows the stacked VLAN frame format.

Figure A–3. Stacked VLAN Frame Format



A.3. Pause Frame

Figure A–4 shows the format of pause frames.

Figure A–4. Pause Frame Format

1 Octet	Start[7:0]
6 Octets	Preamble[47:0]
1 Octet	SFD[7:0]
6 Octets	Destination Address[47:0] = 0x010000c28001
6 Octets	Source Address[47:0] = MAC Primary Address
2 Octets	Type[15:0] = 0x8808
2 Octets	Opcode[15:0] = 0x0001
2 Octets	Pause Quanta[15:0] = 0xP1, 0xP2 (XOFF: P1,P2 = tx_pauseframe_quanta; XON: P1, P2 = 0x0)
42 Octets	Reserved[335:0] = 0x0
4 Octets	Frame Check Sequence[31:0]

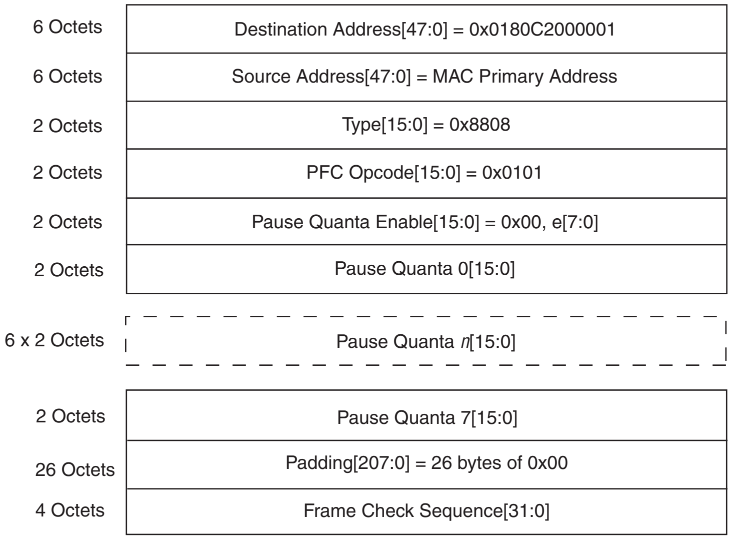
The length/type field has a fixed value of 0x8808, followed by a 2-byte opcode field of 0x0001. Subsequent two bytes define the pause quanta (P1 and P2); P1 is the most significant byte. For XOFF pause frames, the MAC sets the pause quanta field to the value of the `tx_pauseframe_quanta` register. For XON pause frames, the pause quanta is 0. One pause quanta fraction is equivalent to 512 bit times, which equates to 512/64 (the width of the MAC data bus), or 8 cycles for the system clock.

The MAC sets the destination address field to the global multicast address, 01-80-C2-00-00-01 (0x010000c28001) and the source address to the MAC primary address configured in the `tx_addrins_macaddr0` and `tx_addrins_madaddr1` registers. Pause frames have no payload length field, and is always padded with 42 bytes of 0x00.

A.4. Priority-Based Flow Control Frame

The PFC frame is an extension of the basic pause frame. It contains additional fields to enable priority queues and specify pause quanta for these queues. Figure A–5 shows the PFC frame format.

Figure A–5. PFC Frame Format



6 Octets	Destination Address[47:0] = 0x0180C2000001
6 Octets	Source Address[47:0] = MAC Primary Address
2 Octets	Type[15:0] = 0x8808
2 Octets	PFC Opcode[15:0] = 0x0101
2 Octets	Pause Quanta Enable[15:0] = 0x00, e[7:0]
2 Octets	Pause Quanta 0[15:0]
6 x 2 Octets	Pause Quanta *n*[15:0]
2 Octets	Pause Quanta 7[15:0]
26 Octets	Padding[207:0] = 26 bytes of 0x00
4 Octets	Frame Check Sequence[31:0]

The following are the additional fields in the PFC frame:

- PFC Opcode—a 2-octet fixed value of 0x0101.
- Pause Quanta Enable[15:0]—indicates the validity of the pause quanta fields. The upper byte of this field is unused. Each bit in the lower byte represents a priority queue. If bit *n* is set to 1, it indicates that pause quanta *n* is valid and should be acted upon.
- Pause Quanta *n*[15:0]—the pause quanta for priority queue *n*.

Additional Information

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes
July 2012	3.0	■ Added support for Arria V GT. ■ Revised the "Registers", "Interface Signals", and "Design Considerations" sections into individual chapters. ■ Added the "Register Initialization" section in Chapter 5. ■ Added the "10GbE MAC and PHY Connection with XGMII" section in Chapter 7.
May 2011	2.0	■ Added a new section "IP Core Verification" on page 1–3. ■ Revised the "Performance and Resource Utilization" section in Chapter 1. ■ Added new features option in the MAC parameter settings. ■ Updated the design example file directory structure in Table 3–3 on page 3–5. ■ Added two new sections in Chapter 3—"Creating a New 10GbE Design" on page 3–6 and "Parameter Settings" on page 3–7. ■ Added a new section "Transmit and Receive Latencies" on page 3–17. ■ Revised the "Performance and Resource Utilization" section in Chapter 3. ■ Updated the "Transmit Datapath" on page 4–3 and "Receive Datapath" on page 4–9 to describe the new preamble passthrough mode feature. ■ Updated the "Congestion and Flow Control" on page 4–13 to describe the new PFC feature. ■ Added a summary of register address expansion in Table 5–1. ■ Updated all register address and byte offset in table Table 5–2. ■ Revised Figure 6–1 and added two new figures (Figure 6–2 and Figure 6–3) to show the interface signals for Tx only and Rx only datapath. ■ Updated Table 6–2, Table 6–3, Table 6–4, Table 6–6 and Table 6–7 to describe the interface signals for preamble passthrough mode, datapath option, and PFC features. ■ Updated Figure 4–5, Figure 4–8, Figure 4–10, Figure 6–9, and Figure 6–10 to correct the bus signal names.

Date	Version	Changes
November 2010	1.2	■ Added new timing diagrams to the following sections: ■ "Frame Check Sequence (CRC-32) Insertion" on page 4–4 ■ "SDR XGMII Transmission" on page 4–7 ■ "CRC-32 and Pad Removal" on page 4–11 ■ "Pause Frame Transmission" on page 4–14 ■ "Error Handling (Link Fault)" on page 4–17 ■ "SDR XGMII to DDR XGMII Conversion" ■ Added Cyclone IV GX and Stratix III device family to the "Programmable datapath option to allow separate instantiation of MAC Tx block, MAC Rx block, or both MAC Tx and MAC Rx blocks.Device Family Support" section in Chapter 1 and updated Arria GX device family support from preliminary to final. ■ Revised the "Performance and Resource Utilization" section in Chapter 1. ■ Revised the "Ethernet Loopback Module" section in Chapter 3. ■ Revised the design simulation, compilation, and verification flow in Chapter 3. ■ Added a new section "Transmit and Receive Latencies" on page 4–13. ■ Updated Figure 4–9 on page 4–17. ■ Added frames types definition in "Registers" on page 5–1. ■ Corrected the register address of Rx statistics counters and Tx statistics counters in Table 5–2 on page 5–2. ■ Revised the description of reset signals in "MAC Tx Only Variation" on page 6–2.
August 2010	1.1	■ Revised the "Performance and Resource Utilization" section in Chapter 1. ■ Corrected signal name to `rx_clk_clk` in the "Mapping of Client Frame to Avalon-ST Transmit Interface" section in Chapter 6.
June 2010	1.0	Initial release.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

Contact *(1)*	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Non-technical support (General)	Email	nacomp@altera.com
(Software Licensing)	Email	authorization@altera.com

Note:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, *n* + 1. Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>***.pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
☞	The hand points to information that requires special attention.
?	A question mark directs you to a software help system with related information.
👣	The feet direct you to another document or website with related information.
⚠ CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
⚡ WARNING	A warning calls attention to a condition or possible situation that can cause you injury.
✉	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.