LogiCORE IP Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper v2.3

User Guide

UG800 April 24, 2012

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
3/1/11	1.0	Initial Xilinx release with AXI interface.
10/19/11	1.1	Updated for ISE Design Suite 13.3. Addition of 1000BASE-X PCS/PMA overclocked modes and 16-bit AXI4-Stream Interface.
4/24/12	1.2	Updated for ISE Design Suite 14.1. Core version 2.3.

Table of Contents

XILINX®

Chapter 1

Introduction

The Virtex®-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper (V6EMAC) comprises the 10/100/1000 Mb/s MAC and the 1000BASE-X PCS/PMA or SGMII IP Cores, which are fully-verified designs that support Verilog-HDL and VHDL. In addition, the example design provided with the core is in both Verilog and VHDL.

The V6EMAC provides wrappers around the Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper hard block (the TEMAC_SINGLE primitive) to provide new functionality, including a new address map. The TEMAC_SINGLE primitive is described in the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide* [Ref 1].

This chapter provides general information on the V6EMAC, including the recommended design experience, additional resources, technical support, and how to submit feedback to Xilinx.

System Requirements

Windows

- Windows XP Professional 32-bit/64-bit
- Windows Vista Business 32-bit/64-bit

Linux

- Red Hat Enterprise Linux WS v4.0 32-bit/64-bit
- Red Hat Enterprise Desktop v5.0 32-bit/64-bit (with Workstation Option)
- SUSE Linux Enterprise (SLE) desktop and server v10.1 32-bit/64-bit

Software

- ISE® software v14.1

About the Core

The V6EMAC is generated through the Xilinx CORE Generator™ software, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see the V6EMAC product page.

Recommended Design Experience

Although the V6EMAC is fully-verified, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, previous experience building high performance, pipelined FPGA designs using Xilinx implementation software and User Constraint Files (UCF) is recommended. Contact your local Xilinx representative for a closer review and estimation for your specific requirements.

Technical Support

For technical support, see support.xilinx.com/. Questions are routed to a team of engineers with expertise using the V6EMAC.

Xilinx provides technical support for use of this product as described in this manual, *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper v2.3 User Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the V6EMAC and the documentation supplied with the core.

Tri-Mode Ethernet MAC Core

For comments or suggestions about the core, submit a WebCase from www.xilinx.com/support/clearexpress/websupport.htm. Be sure to include the following information:

- Product name
- Core version number
- Configuration mode (10/100/1000 Mb/s, 1 Gb/s, 10/100 Mb/s)
- Explanation of your comments

Documentation

For comments or suggestions about the core documentation, submit a WebCase from www.xilinx.com/support/clearexpress/websupport.htm. Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

XILINX®

Chapter 2

Ethernet Overview

This chapter provides an overview of typical Ethernet applications and the Ethernet protocol.

This chapter contains the following sections:

- Typical Ethernet Application Overview
- Ethernet Protocol Overview

Typical Ethernet Application Overview

Typical applications for the V6EMAC include:

- Ethernet Switch or Router
- Ethernet Communications Port for an Embedded Processor

Ethernet Switch or Router

Figure 2-1 illustrates a typical application for a single V6EMAC. The PHY side of the core is connected to an off-the-shelf Ethernet PHY device, which performs the BASE-T standard at 1 Gb/s, 100 Mb/s, and 10 Mb/s speeds. The PHY device can be connected using any of the following supported interfaces: GMII/MII, RGMII, or SGMII.

The user side of the V6EMAC is connected to a FIFO to complete a single Ethernet port. This port is connected to a Switch or Routing matrix, which can contain several ports.

The CORE Generator™ tool provides an example design for the V6EMAC for any of the supported physical interfaces. A FIFO example is also generated, which can be used as the FIFO in the illustration, for a typical application.

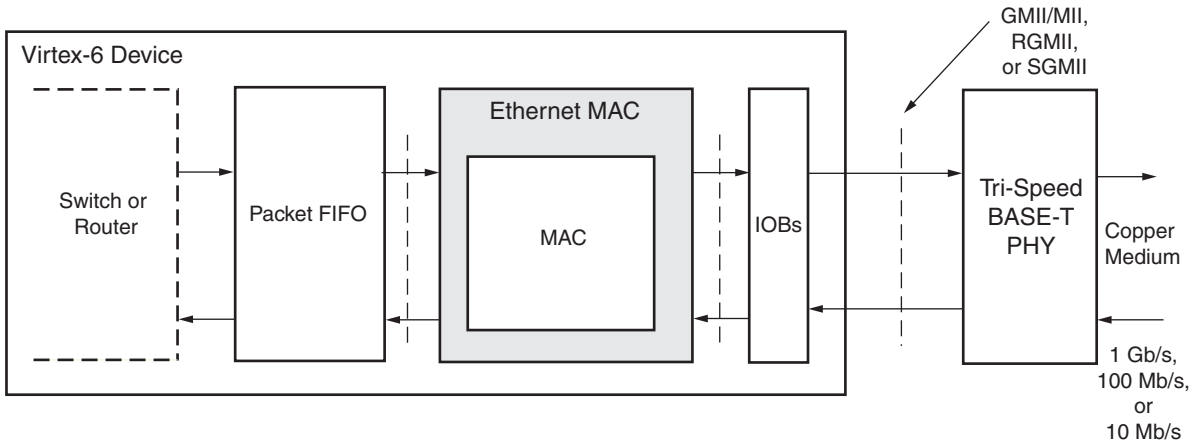

Figure 2-1: **Typical Application: Ethernet Switch or Router**

Ethernet Communications Port for an Embedded Processor

Figure 2-2 illustrates a typical application for a single V6EMAC. The PHY side of the core is connected to an off-the-shelf Ethernet PHY device, which performs the BASE-T standard at 1 Gb/s, 100 Mb/s, and 10 Mb/s speeds. The PHY device can be connected using any of the following supported interfaces: GMII/MII, RGMII, or SGMII.

The user side of the MAC is connected to a processor system through a processor DMA engine. This processor could be running a communications stack, such as the TCP/IP protocol. For applications such as this, see the Xilinx Platform Studio (XPS), Embedded Development Kit (EDK) IP portfolio. This portfolio contains additional IP to connect the AXI4-Stream interface of the MAC to the DMA port of a processor. The *AXI Ethernet Data Sheet* [Ref 2] describes the AXI Ethernet, which can be instantiated for an intended processor application.

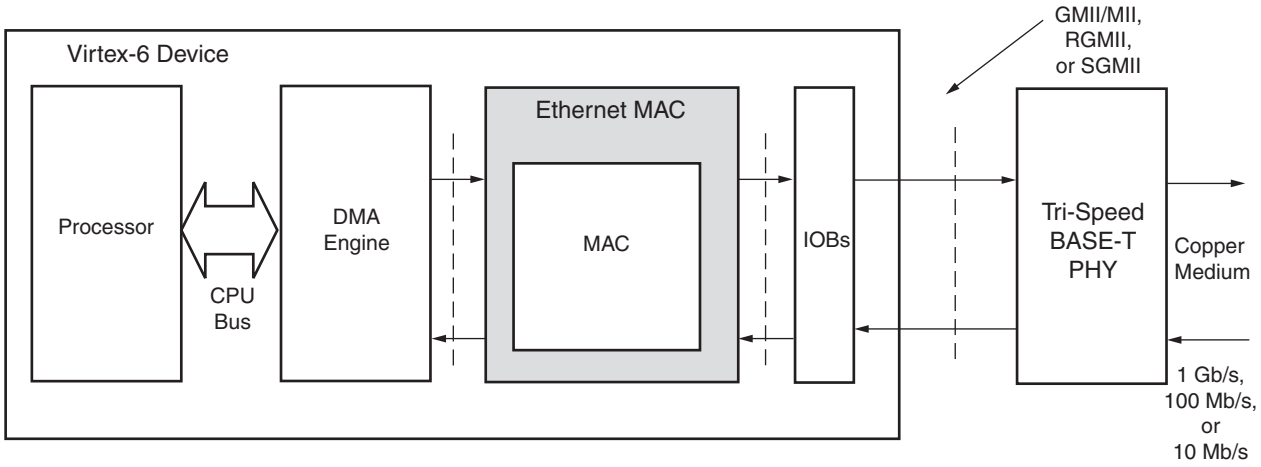

Figure 2-2: **Typical Application: Ethernet Communications Port for Embedded Processor**

Ethernet Protocol Overview

This section gives an overview of where the V6EMAC fits into an Ethernet system and provides a description of some basic Ethernet terminology.

Ethernet Sublayer Architecture

Figure 2-3 illustrates the relationship between the OSI reference model and the V6EMAC, as defined in *IEEE Std 802.3-2008* [Ref 4]. The grayed-in layers show the functionality that the V6EMAC handles. Figure 2-3 also shows where the supported physical interfaces fit into the architecture.

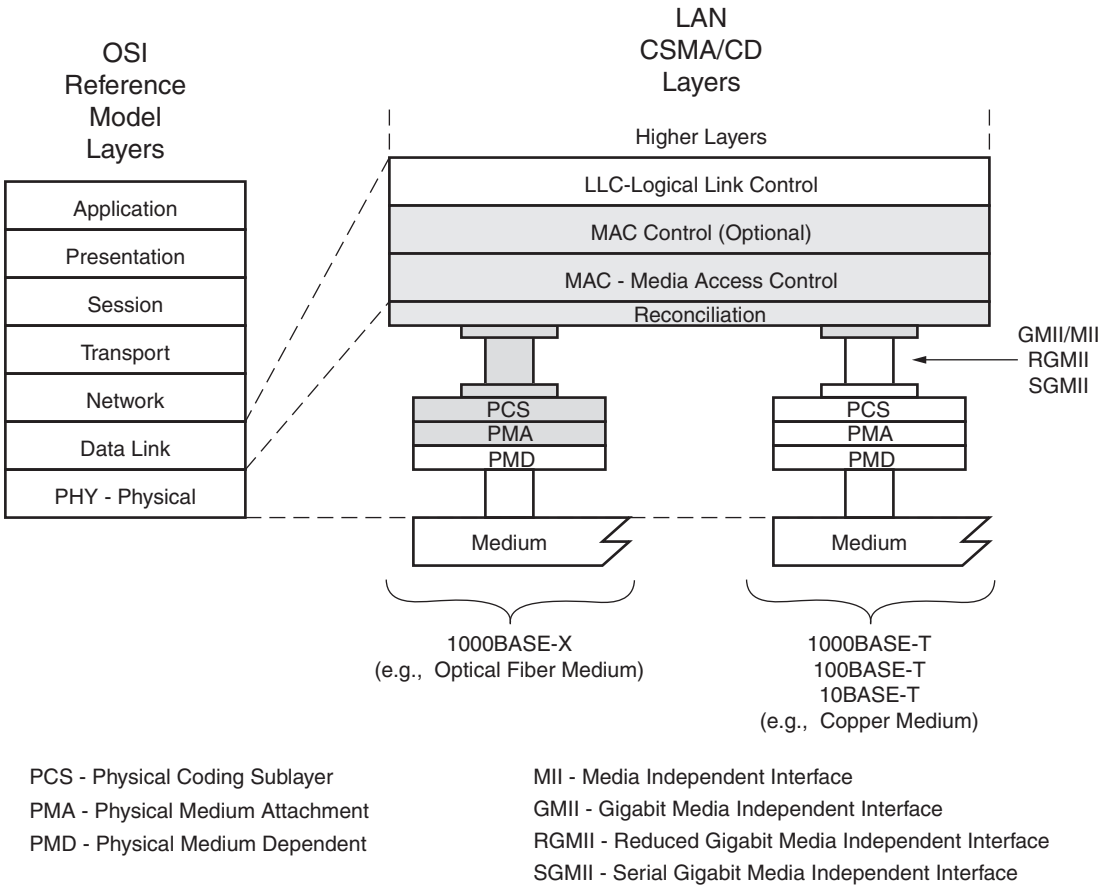

Figure 2-3: **IEEE Std 802.3-2008 Ethernet Model**

MAC and MAC CONTROL Sublayer

The V6EMAC is defined in *IEEE Std 802.3-2008* [Ref 4], clauses 2, 3, and 4. A MAC is responsible for the Ethernet framing protocols described in Ethernet Data Format and error detection of these frames. The MAC is independent of and can connect to any type of physical layer device.

The MAC Control sublayer is defined in *IEEE Std 802.3-2008* [Ref 4], clause 31. This provides real-time flow control manipulation of the MAC sublayer.

Both the MAC CONTROL and MAC sublayers are provided by the V6EMAC in all modes of operation.

Physical Sublayers PCS, PMA, and PMD

The combination of the Physical Coding Sublayer (PCS), the Physical Medium Attachment (PMA), and the Physical Medium Dependent (PMD) sublayer constitute the physical layers for the protocol. Two main physical standards are specified:

- **BASE-T** PHYs provide a link between the MAC and copper mediums. This functionality is not offered within the V6EMAC. However, external BASE-T PHY devices are readily available on the market. These can connect to the V6EMAC, using GMII/MII, RGMII or SGMII interfaces.
- **BASE-X** PHYs provide a link between the MAC and (usually) fiber optic mediums. The V6EMAC is capable of supporting the 1 Gb/s BASE-X standard.

Ethernet Data Format

Ethernet data is encapsulated in frames, as shown in Figure 2-4, for standard Ethernet frames. The fields in the frame are transmitted from left to right. The bytes within the fields are transmitted from left to right (from least significant bit to most significant bit unless specified otherwise). The V6EMAC can handle jumbo Ethernet frames where the data field can be much larger than 1500 bytes.

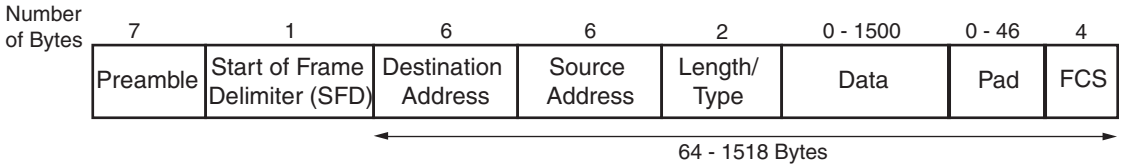

Figure 2-4: **Standard Ethernet Frame Format**

The V6EMAC can also accept VLAN frames. The VLAN frame format is shown in Figure 2-5. If the frame is a VLAN type frame, the V6EMAC accepts four additional bytes.

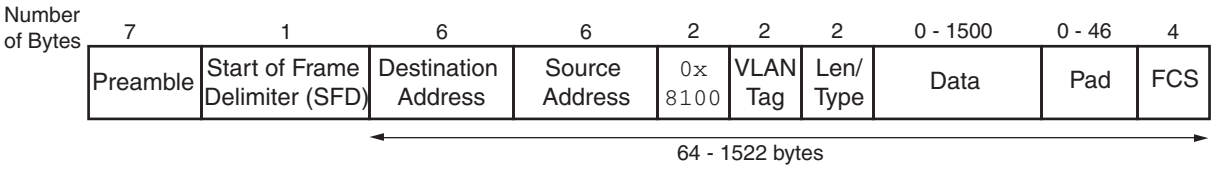

Figure 2-5: **Ethernet VLAN Frame Format**

Ethernet PAUSE/flow control frames can be transmitted and received by the V6EMAC. Figure 7-2, page 60 shows how a PAUSE/flow control frame differs from the standard Ethernet frame format.

The following subsections describe the individual fields of an Ethernet frame and some basic functionality of the V6EMAC.

Preamble

For transmission, this field is automatically inserted by the V6EMAC. The preamble field was historically used for synchronization and contains seven bytes with the pattern `0x55`, transmitted from left to right. For reception, this field is always stripped from the incoming frame, before the data is passed to the user. The V6EMAC can receive Ethernet frames, even if the preamble does not exist, as long as a valid start of frame delimiter is available.

Start of Frame Delimiter

The start of frame delimiter field marks the start of the frame and must contain the pattern 0xD5.

For transmission on the physical interface, this field is automatically inserted by the V6EMAC. For reception, this field is always stripped from the incoming frame before the data is passed to the user.

MAC Address Fields

MAC Address

The least significant bit of a MAC address determines if the address is an individual/unicast (0) or group/multicast (1) address. Multicast addresses are used to group logically related stations. The broadcast address (destination address field is all 1s) is a multicast address that addresses all stations on the LAN. The V6EMAC supports transmission and reception of unicast, multicast, and broadcast packets.

The address is transmitted in an Ethernet frame least significant bit first: so the bit representing an individual or group address is the first bit to appear in an address field of an Ethernet frame.

Destination Address

This MAC Address field is the first field of the Ethernet frame that is always provided in the packet data for transmissions and is always retained in the receive packet data. It provides the MAC address of the intended recipient on the network.

Source Address

This MAC Address field is the second field of the Ethernet frame that is always provided in the packet data for transmissions and is always retained in the receive packet data. It provides the MAC address of the frame's initiator on the network.

For transmission, the source address of the Ethernet frame should always be provided by the user because it is unmodified by the V6EMAC.

Length/Type

The value of this field determines if it is interpreted as a length or a type field, as defined by *IEEE Std 802.3-2008* [Ref 4]. A value of 1536 decimal or greater is interpreted by the V6EMAC as a type field.

When used as a length field, the value in this field represents the number of bytes in the following data field. This value does not include any bytes that can be inserted in the pad field following the data field.

A length/type field value of 0x8100 indicates that the frame is a VLAN frame, and a value of 0x8808 indicates a PAUSE MAC control frame.

For transmission, the V6EMAC does not perform any processing of the length/type field.

For reception, if this field is a length field, the V6EMAC receive engine interprets this value and removes any padding in the pad field (if necessary). If the field is a length field and length/type checking is enabled, the V6EMAC compares the length against the actual data field length and flags an error if a mismatch occurs. If the field is a type field, the V6EMAC ignores the value and passes it along with the packet data with no further processing. The length/type field is always retained in the receive packet data.

Data

The data field can vary from 0 to 1,500 bytes in length for a normal frame. The V6EMAC can handle jumbo frames of any length.

This field is always provided in the packet data for transmissions and is always retained in the receive packet data.

Pad

The pad field can vary from 0 to 46 bytes in length. This field is used to ensure that the frame length is at least 64 bytes in length (the preamble and SFD fields are not considered part of the frame for this calculation), which is required for successful CSMA/CD operation. The values in this field are used in the frame check sequence calculation but are not included in the length field value, if it is used. The length of this field and the data field combined must be at least 46 bytes. If the data field contains 0 bytes, the pad field is 46 bytes. If the data field is 46 bytes or more, the pad field has 0 bytes.

For transmission, this field can be inserted automatically by the V6EMAC or can be supplied by the client. If the pad field is inserted by the V6EMAC, the FCS field is calculated and inserted by the V6EMAC. If the pad field is supplied by the user, the FCS can be either inserted by the V6EMAC or provided by the user, as indicated by a configuration register bit.

For reception, if the length/type field has a length interpretation, any pad field in the incoming frame is not be passed to the user, unless the V6EMAC is configured to pass the FCS field on to the user.

FCS

The value of the FCS field is calculated over the destination address, source address, length/type, data, and pad fields using a 32-bit Cyclic Redundancy Check (CRC), as defined in *IEEE Std 802.3-2008* [Ref 4] para. 3.2.8:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x^{1} + x^{0}$$

The CRC bits are placed in the FCS field with the x^{31} term in the left-most bit of the first byte, and the x^{0} term is the right-most bit of the last byte (that is, the bits of the CRC are transmitted in the order x^{31}, x^{30},..., x^{1}, x^{0}).

For transmission, this field can be either inserted automatically by the V6EMAC or supplied by the user, as indicated by a configuration register bit. If this field is supplied by the user and the frame does not meet the minimum frame size of 64 bytes, the V6EMAC appends a pad field which results in a bad frame. This is not captured by the statistics vector for the frame.

For reception, the incoming FCS value is verified on every frame. If an incorrect FCS value is received, the V6EMAC indicates to the client that it has received a bad frame. The FCS field can either be passed on to the user or be dropped by the V6EMAC, as indicated by a configuration register bit.

Frame Transmission and Interframe Gap

Frames are transmitted over the Ethernet medium with an interframe gap, as specified by *IEEE Std 802.3-2008* [Ref 4]. For full-duplex systems, the minimum IFG is 96 bit times (9.6 µs for 10 Mb/s, 0.96 µs for 100 Mb/s, and 96 ns for 1 Gb/s), as defined by *IEEE Std 802.3-2008* [Ref 4]. For half-duplex systems, the minimum IFG is 208 bit times when using the RGMII physical interface, and 144 bit times when using the MII physical interface. The defined IFG is a minimum and can be increased with a resulting decrease in throughput.

The process for frame transmission is different for half-duplex and full-duplex systems.

Half-Duplex Frame Transmission

In a half-duplex system, the CSMA/CD media access method defines how two or more stations share a common medium.

1. Even when it has nothing to transmit, the V6EMAC monitors the Ethernet medium for traffic by watching the carrier sense signal (CRS) from the external PHY. Whenever the medium is busy (CRS = `1`), the V6EMAC defers to the passing frame by delaying any pending transmission of its own.
2. After the last bit of the passing frame (when the carrier sense signal changes from TRUE to FALSE), the V6EMAC starts the timing of the interframe gap.
3. The V6EMAC resets the interframe gap timer if the carrier sense becomes TRUE during the period defined by "interframe gap part 1 (IFG1)." *IEEE Std 802.3-2008* [Ref 4] states that this should be the first 2/3 of the interframe gap timing interval (64 bit times) but can be shorter and as small as zero. The purpose of this option is to support a possible brief failure of the carrier sense signal during a collision condition and is described in paragraph 4.2.3.2.1 of the IEEE standard.
4. The V6EMAC does not reset the interframe gap timer if carrier sense becomes TRUE during the period defined by "interframe gap part 2 (IFG2)" to ensure fair access to the bus. *IEEE Std 802.3-2008* [Ref 4] states that this should be the last 1/3 of the interframe gap timing interval.

If, after initiating a transmission, the message collides with the message of another station (COL = `1`), then each transmitting station intentionally continues to transmit (jam) for an additional predefined period (32 bit times) to ensure propagation of the collision throughout the system. The station remains silent for a random amount of time (back off) before attempting to transmit again.

A station can experience a collision during the beginning of its transmission (the collision window) before its transmission has had time to propagate to all stations on the bus. After the collision window has passed, a transmitting station has acquired the bus. Subsequent collisions (late collisions) are avoided because all other (properly functioning) stations are assumed to have detected the transmission and are deferring to it.

Full-Duplex Frame Transmission

In a full-duplex system, there is a point-to-point dedicated connection between two Ethernet devices, capable of simultaneous transmit and receive with no possibility of collisions. The V6EMAC does not use the carrier sense signal from the external PHY because the medium is not shared, and the V6EMAC only needs to monitor its own transmissions. After the last bit of an V6EMAC frame transmission, the V6EMAC starts the interframe gap timer and defers transmissions until the IFG count completes. The minimum value supported for the IFG delay is 96 bit times, or when IFG Adjustment is enabled, the greater of 64 bit times, and the value presented on `tx_ifg_delay`.

Generating the Core

The V6EMAC, which comprises the 10/100/1000 Mb/s, 1 Gb/s and 10/100 Mb/s IP cores, is generated through the Xilinx CORE Generator™ tool using a graphical user interface (GUI). This chapter describes the GUI options used to generate and customize the core.

GUI Interface: Interface Configuration

Figure 3-1 displays page 1 of the V6EMAC core customization screen.

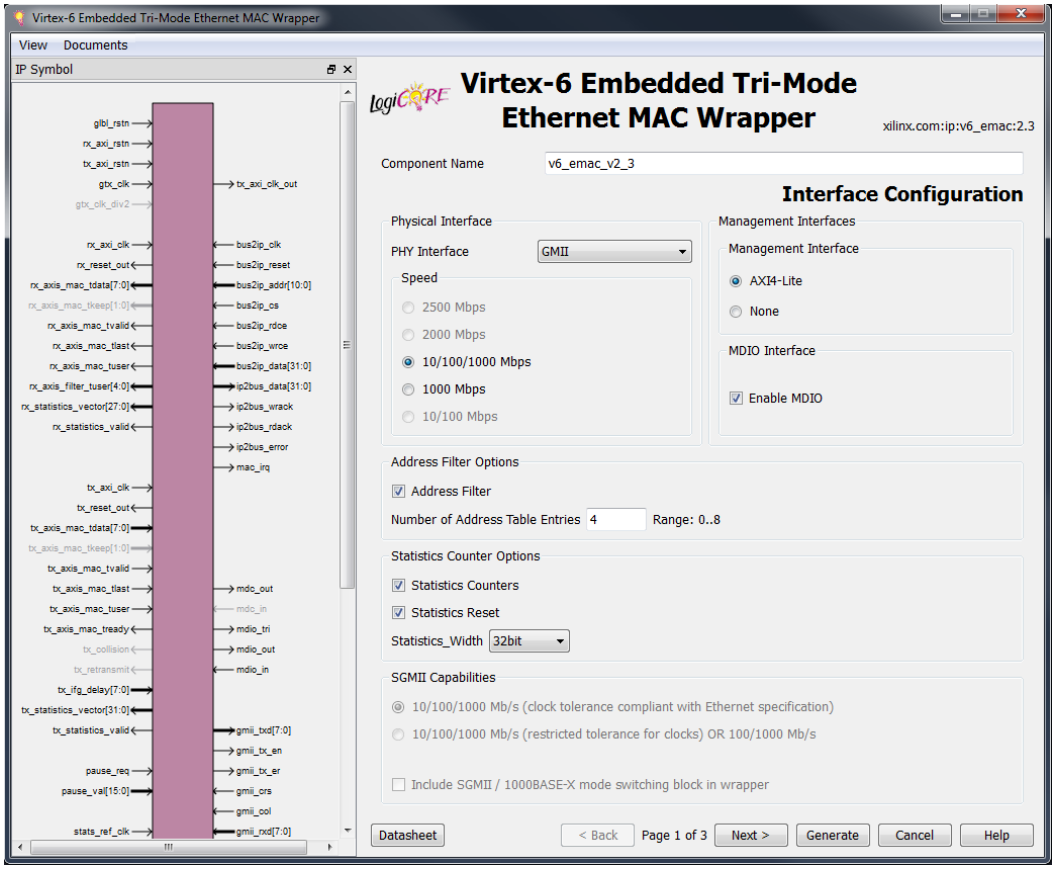

Figure 3-1: **Core Customization Screen**

For general help starting and using the CORE Generator tool on your system, see the documentation supplied with the Xilinx ISE® software, including the *CORE Generator Guide* at www.xilinx.com/support/software_manuals.htm.

Component Name

The component name is used as the base name of the output files generated for the core. Names must begin with a letter and must be composed from the following characters: a through z, 0 through 9 and "_".

Physical Interface

Five physical interface types are available for the core:

- GMII. The Gigabit Media Independent Interface (GMII) is defined by the IEEE802.3 specification; it can provide support for Ethernet operation at 10 Mb/s, 100 Mb/s and 1 Gb/s speeds.
- MII. The Media Independent Interface (MII) is defined by the IEEE802.3 specification; it can provide support for Ethernet operation at 10 Mb/s and 100 Mb/s speeds.
- RGMII. The Reduced Gigabit Media Independent Interface (RGMII) is, effectively, a Double Data Rate version of GMII; it can provide support for Ethernet operation at 10 Mb/s, 100 Mb/s and 1 Gb/s speeds.
- SGMII. The Serial Gigabit Media Independent Interface (SGMII) provides a single bit version of GMII through use of on-chip transceivers; it can provide support for Ethernet operation at 10 Mb/s, 100 Mb/s and 1 Gb/s speeds.
- 1000BASE-X PCS PMA. The 1000BASE-X physical standard, described in *IEEE Std 802.3-2008* [Ref 4], clauses 36 and 37, defines a physical sublayer that is usually connected to an optical fiber medium. Two common implementations currently exist: 1000BASE-LX and 1000BASE-SX (long and short wavelength laser), which can both be obtained by connecting the serial transceiver to a suitable GBIC or SFP optical transceiver.

The choice of physical interface determines the content of the example design delivered with the core where the external GMII, MII or RGMII is described in HDL. If either SGMII or 1000BASE-X PCS/PMA are selected then the core uses the internal *1000BASE-X PCS/PMA or SGMII* core to connect to a transceiver instantiated in the example design.

The default is to use GMII.

MAC Speed

The V6EMAC can provide support for 1 Gb/s speed only operation; 10 Mb/s and 100 Mb/s speed operation; full tri-speed operation (10 Mb/s, 100 Mb/s and 1 Gb/s speed capability); 2 Gb/s or 2.5 Gb/s overclocked operation when using the 1000BASE-X physical interface on supported devices.

The available choice for speed support selection is dependent on the chosen physical interface:

- If GMII or RGMII is selected, then Tri-speed operation and 1 Gb/s only operation are available for selection.
- If MII is selected, then only 10 Mb/s and 100 Mb/s operation is available.
- If SGMII is selected then Tri-speed operation, 1 Gb/s only operation and 10 Mb/s and 100 Mb/s are available for selection.
- If 1000BASE-X PCS/PMA is selected then 1Gb/s only operation, 2 Gb/s only operation, or 2.5 Gb/s only operation selections are available. When selected, the 2 Gb/s and 2.5 Gb/s overclocked modes use a 16-bit client interface for the core.

Management Interface

Select the AXI4-Lite option if you wish to include the optional Management Interface for V6EMAC configuration (see The Management Interface). If this option is not selected, then core attributes can be set using the Transmitter and Receiver Configuration page.

The default is to have the AXI4-Lite Management Interface.

Enable MDIO

When selected, the Management Data Input/Output (MDIO) option enables the MDIO ports on the V6EMAC to access the registers in the internal and external PHY. When the MDIO option is selected, an MDIO configuration screen appears before generating the core. When not selected, the MDIO configuration screen is not displayed. If the AXI4-Lite management interface option is selected then the MAC is configured as an MDIO master. If no management interface is selected then the MAC is an MDIO slave.

The default is to enable the MDIO.

Address Filter

It is possible to generate the core with a address filter, which prevents the reception of frames that are not matched by this MAC. This is most commonly used to identify packets which are addressed specifically to this MAC.

The default is to use the Address Filter.

Number of Address Table Entries

The Address Filter can be generated with a look-up table that holds up to 8 additional valid MAC frame match patterns. You can select an integer between 0 and 8 to define the number of match patterns that are present in the table.

The default is to use four table entries.

Statistics Counter

It is possible to generate the core with built in statistics counters. When used in conjunction with the vector decode block provided in the example design, these allow the collection of network traffic information.

The default is to use statistics counters.

Statistics Reset

The statistics counters are implemented using distributed memory and as such have no basic reset functionality. It is possible to include extra logic to ensure that all counters are reset to zero upon a reset.

The default is to include the statistics reset logic.

Statistics width

It is possible to select between 32 bit or 64 bit counter widths. This offers the ability to significantly extend the duration between counter rollover at the expense of logic. The default is to use 32 bit counters.

SGMII Capabilities

Select the desired mode of operation to enable the appropriate receive buffering.

- **10/100/1000 Mb/s (clock tolerance compliant with Ethernet specification)**. Default setting; provides the implementation using the Receiver Elastic Buffer in FPGA logic. This alternative Receiver Elastic Buffer utilizes a single block RAM to create a buffer twice as large as the one present in the serial transceiver, subsequently consuming extra logic resources. However, this default mode provides reliable SGMII operation under all conditions.
- **10/100/1000 Mb/s (restricted tolerance for clocks) or 100/1000 Mb/s**. Uses the receiver elastic buffer present in the serial transceivers. This is half the size and can potentially under- or overflow during SGMII frame reception at 10 Mb/s operation. However, there are logical implementations where this can be proven reliable and favored because of its lower logic utilization.
- **Include SGMII / 1000BASE-X PCS/PMA mode switching block in wrapper.** The V6EMAC supports switching between SGMII and 1000BASE-X modes of operation using configuration registers. If this option is selected, a block is additionally included in the wrappers, which facilitates mode switching based on a single, level input.

For detailed information about SGMII capabilities, see Chapter 12, Serial Gigabit Media Independent Interface (SGMII)

The default is to include the elastic buffer and not to allow SGMII/1000BASE-X switching.

GUI Interface: Transmitter and Receiver Configuration

The next configuration screen defines the default transmitter, receiver, and address filter configurations

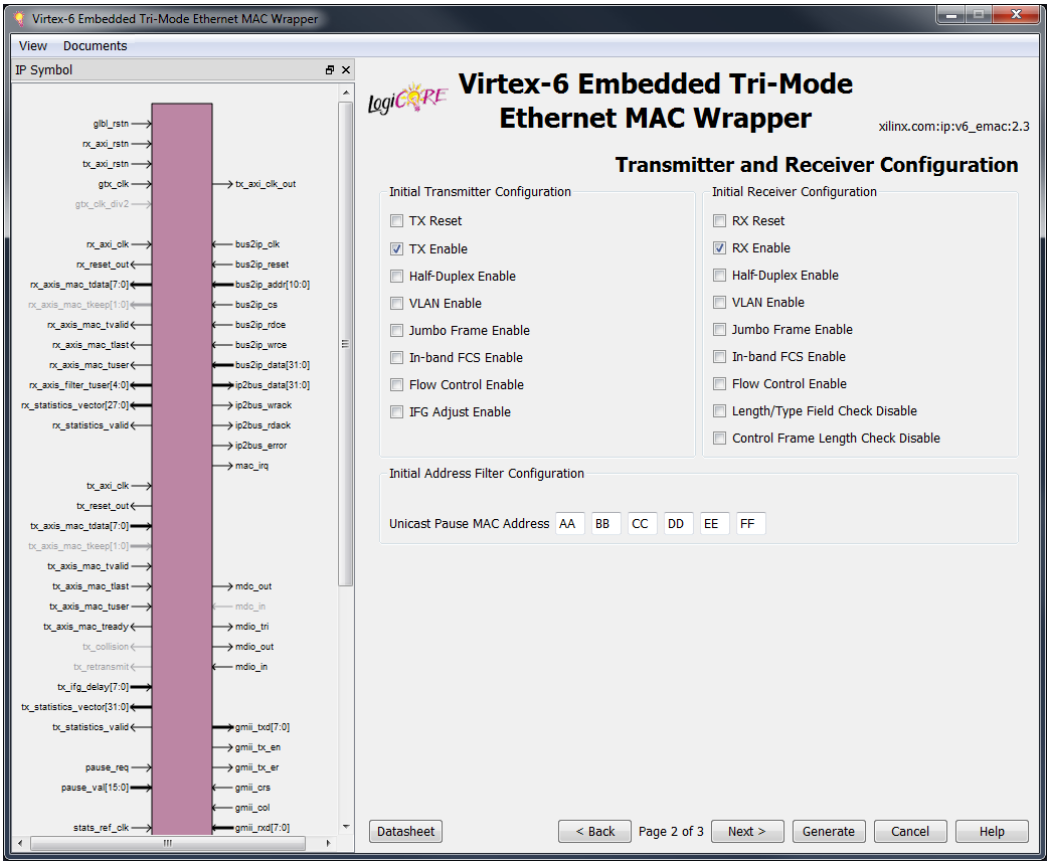

Figure 3-2: **Transmitter and Receiver Configuration**

Initial Transmitter Configuration

Transmitter configuration refers to the V6EMAC configuration registers located at `0x408`. Initial values for several bits of this register can be set using the GUI. Changes to the register bits can be written using a management interface access, if enabled. For more information, see The Management Interface in Chapter 8.

- **TX Reset**. When selected, places the transmitter in reset. When Management Interface is set to None, this option is not selected and cannot be changed.
- **TX Enable**. When selected, enables the transmitter. When Management Interface is set to None, this option is selected and cannot be changed.
- **Half-Duplex Enable**. When selected, the transmitter operates in half-duplex mode (applicable for MII, GMII, or RGMII physical interfaces at 10 or 100 Mb/s only). When not selected, the transmitter operates in full-duplex mode.
- **VLAN Enable**. When selected, the transmitter allows transmission of the VLAN-tagged frames, as specified in *IEEE Std 802.3-2008* [Ref 4].

- **Jumbo Frame Enable**. When selected, the transmitter sends frames greater than the maximum length specified in *IEEE Std 802.3-2008* [Ref 4]. When not selected, the transmitter sends only frames less than the specified maximum length.
- **In-band FCS Enable**. When selected, this bit causes the V6EMAC transmitter to expect that the FCS field is supplied by the user.
- **Flow Control Enable.** When selected, the transmission of flow control PAUSE frames is enabled.
- **IFG Adjust Enable**. When selected, the transmitter reads the value of `TX_IFG_DELAY` at the start of frame transmission and adjusts the IFG. This option is applicable for full-duplex operation only.

Receiver Configuration

Receiver configuration refers to the V6EMAC configuration registers located at `0x400`. Initial values for several bits of this register can be set using the GUI. Changes to the register bits can be written using a management interface access, if enabled. For more information, see The Management Interface in Chapter 8.

- **RX Reset**. When selected, places the receiver in reset. When Management Interface is set to None, this option is not selected and cannot be changed.
- **RX Enable**. When selected, enables the receiver. When Management Interface is set to None, this option is selected and cannot be changed.
- **Half-Duplex Enable**. When selected, the receiver operates in half-duplex mode (applicable for MII, GMII, or RGMII physical interfaces at 10 or 100 Mb/s only). When not selected, the receiver operates in full-duplex mode.
- **VLAN Enable**. When selected, the receiver accepts VLAN-tagged frames, as specified in *IEEE Std 802.3-2008* [Ref 4]. The maximum accepted payload length increases by four bytes.
- **Jumbo Frame Enable**. When selected, the V6EMAC receiver accepts frames over the maximum length specified in *IEEE Std 802.3-2008* [Ref 4]. When not selected, the receiver accepts only frames up to the specified maximum.
- **In-band FCS Enable**. When selected, the receiver passes the FCS field up to the user. When not selected, the FCS field is not passed to the user. In either case, the FCS of each frame is checked for correctness.
- **Flow Control Enable.** When selected, flow control PAUSE frames are received and acted upon.
- **Length/Type Field Check Disable**. When selected, disables the Length/Type field check on the frame.
- **Control Frame Length Check Disable**. When selected, control frames larger than the legal minimum frame size are accepted.

Initial Address Filter Configuration

- **Unicast Pause MAC Address.** The value entered by you is used by the V6EMAC to compare the destination address of any incoming flow control frames, and as the source address for any outbound flow control frames.

  The address is ordered for the least significant byte in the register to have the first byte transmitted or received, for example, a MAC address of `AA-BB-CC-DD-EE-FF` is entered as `FF-EE-DD-CC-BB-AA`.

GUI Interface: MDIO Configuration

The MDIO Configuration screen is only displayed if the 1000BASE-X PCS/PMA or SGMII PHY interface is selected and the Enable MDIO option is selected in the first EMAC configuration screen.

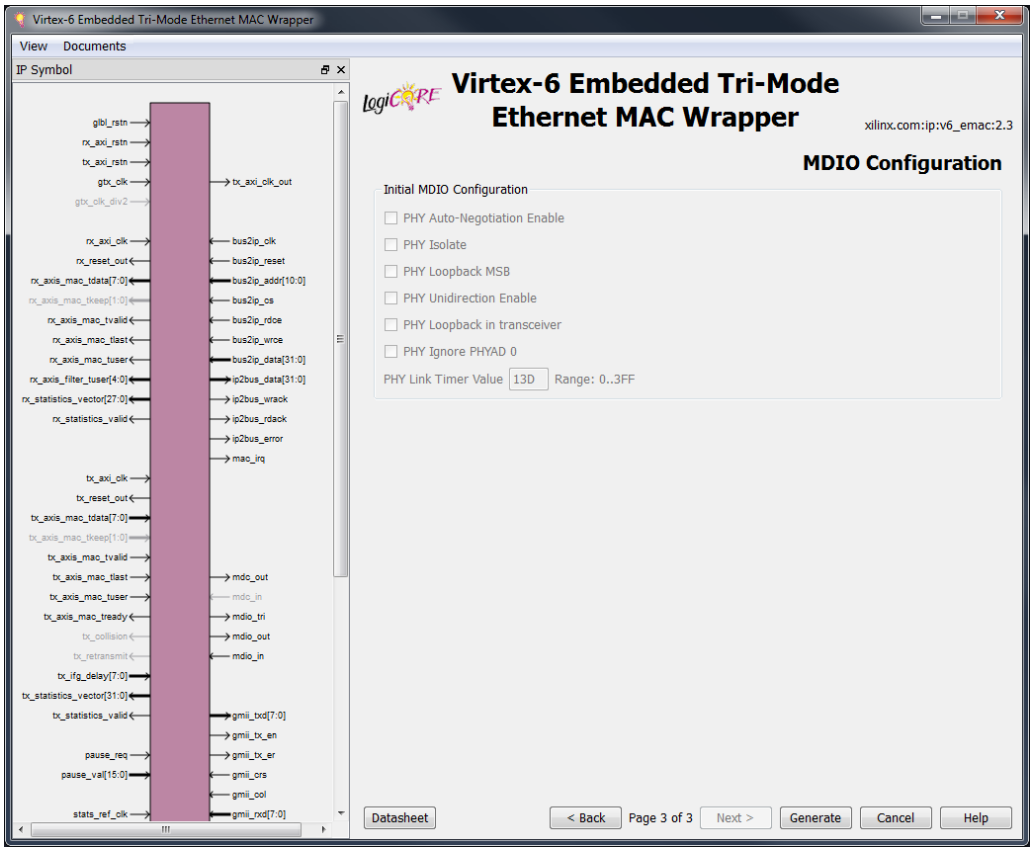

Figure 3-3: **MDIO Configuration**

Initial MDIO Configuration

PHY Auto-Negotiation Enable. If selected, Auto-Negotiation is enabled.

- **PHY Isolate**. If selected, the PHY is electrically isolated.
- **PHY Loopback MSB**. If selected, the PHY loopback is enabled.
- **PHY Unidirection Enable**. If selected, the PHY is capable of transmitting data regardless of whether a valid link has been established.
- **PHY Loopback in Transceiver**. If selected, loopback occurs in the serial transceiver. Otherwise loopback occurs in the V6EMAC.
- **PHY Ignore PHYAD 0**. If selected, the PHY ignores the MDIO broadcast address, PHYAD 0.
- **PHY Link Timer Value**. Programmable auto negotiation link timer value. See SGMII Auto-Negotiation Link Timer and 1000BASE-X Auto-Negotiation Link Timer for more details.

Parameter Values in the XCO File

An XCO file is produced by CORE Generator software whenever a core is customized and generated: it records all options used in the generation of the core, and lists these as parameters. Existing or manually created XCO files can be imported into a CORE Generator software project. XCO file parameter names and their values are identical to the names and values shown in the GUI, except that underscore characters (_) are used instead of spaces. The text in an XCO file is case insensitive. Table 3-1 shows the XCO file parameters and values, and summarizes the GUI defaults. The following is an example of the CSET parameters in an XCO file:

```
CSET component_name=v6_emac_v2_3
CSET physical_interface=GMII
CSET speed=Tri_speed
CSET axi_ipif=true
CSET management_interface=true
CSET mdio=true
CSET address_filter=true
CSET number_of_address_table_entries=4
CSET statistics_counters=true
CSET statistics_reset=true
CSET statistics_width=32bit
CSET phy_an_enable=false
CSET phy_ignore_adzero=false
CSET phy_isolate=false
CSET phy_link_timer_value=13D
CSET phy_loopback_in_gtp=false
CSET phy_loopback_msb=false
CSET phy_powerdown=false
CSET phy_reset=false
CSET phy_unidirection_enable=false
CSET tx_enable=true
CSET tx_flow_control_enable=false
CSET tx_half_duplex_enable=false
CSET tx_ifg_adjust_enable=false
CSET tx_in_band_fcs_enable=false
CSET tx_jumbo_frame_enable=false
CSET tx_reset=false
CSET tx_vlan_enable=false
CSET rx_ctrl_lencheck_disable=false
CSET rx_disable_length=false
CSET rx_enable=true
CSET rx_flow_control_enable=false
CSET rx_half_duplex_enable=false
CSET rx_in_band_fcs_enable=false
CSET rx_jumbo_frame_enable=false
CSET rx_reset=false
CSET rx_vlan_enable=false
CSET serial_mode_switch_enable=false
CSET sgmii_mode=No_clock
CSET unicast_pause_mac_address_1=AA
CSET unicast_pause_mac_address_2=BB
CSET unicast_pause_mac_address_3=CC
CSET unicast_pause_mac_address_4=DD
CSET unicast_pause_mac_address_5=EE
CSET unicast_pause_mac_address_6=FF
CSET address_filter_enable=false
CSET client_side_data_width=8_bit
CSET clock_enable=true
```

Table 3-1: **XCO File Values and Default Values**

Parameter	XCO File Values	Default GUI Setting
component_name	ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _	v6_emac_v2_3
physical_interface	One of the following keywords: mii, gmii, rgmii, sgmii, 1000BASE_X_PCS_PMA	gmii
speed	One of the following keywords: tri_speed, 1000_Mbps, 10_100_Mbps	tri_speed
axi_ipif	Always set to TRUE	TRUE
management_interface	One of the following keywords: TRUE, FALSE	TRUE
mdio	One of the following keywords: TRUE, FALSE	TRUE
address_filter	One of the following keywords: TRUE, FALSE	TRUE
number_of_address_table_entries	Integer in the range 0 - 8	4
statistics_counters	One of the following keywords: TRUE, FALSE	TRUE
statistics_reset	One of the following keywords: TRUE, FALSE	TRUE
statistics_width	One of the following keywords: 32bit, 64bit	32bit
PHY_Reset	One of the following keywords: TRUE, FALSE	FALSE
PHY_AN_Enable	One of the following keywords: TRUE, FALSE	FALSE
PHY_Isolate	One of the following keywords: TRUE, FALSE	FALSE
PHY_Powerdown	One of the following keywords: TRUE, FALSE	FALSE
PHY_Loopback_MSB	One of the following keywords: TRUE, FALSE	FALSE
PHY_Unidirection_Enable	One of the following keywords: TRUE, FALSE	FALSE
PHY_Loopback_in_GTP	One of the following keywords: TRUE, FALSE	FALSE
PHY_Ignore_AdZero	One of the following keywords: TRUE, FALSE	FALSE
PHY_Link_Timer_Value	Value in the range 0 - 3ff	13d

Table 3-1: **XCO File Values and Default Values *(Cont'd)***

Parameter	XCO File Values	Default GUI Setting
TX_Reset	One of the following keywords: TRUE, FALSE	FALSE
TX_Enable	One of the following keywords: TRUE, FALSE	TRUE
TX_Half_Duplex_Enable	One of the following keywords: TRUE, FALSE	FALSE
TX_VLAN_Enable	One of the following keywords: TRUE, FALSE	FALSE
TX_Jumbo_Frame_Enable	One of the following keywords: TRUE, FALSE	FALSE
TX_In_Band_FCS_Enable	One of the following keywords: TRUE, FALSE	FALSE
TX_Flow_Control_Enable	One of the following keywords: TRUE, FALSE	FALSE
TX_IFG_Adjust_Enable	One of the following keywords: TRUE, FALSE	FALSE
RX_Reset	One of the following keywords: TRUE, FALSE	FALSE
RX_Enable	One of the following keywords: TRUE, FALSE	TRUE
RX_Half_Duplex_Enable	One of the following keywords: TRUE, FALSE	FALSE
RX_VLAN_Enable	One of the following keywords: TRUE, FALSE	FALSE
RX_Jumbo_Frame_Enable	One of the following keywords: TRUE, FALSE	FALSE
RX_In_Band_FCS_Enable	One of the following keywords: TRUE, FALSE	FALSE
RX_Flow_Control_Enable	One of the following keywords: TRUE, FALSE	FALSE
RX_Disable_Length	One of the following keywords: TRUE, FALSE	FALSE
RX_Ctrl_Lencheck_Disable	One of the following keywords: TRUE, FALSE	FALSE
SGMII_Mode	One of the following keywords: Clock, no_clock	no_clock
Serial_Mode_Switch_Enable	One of the following keywords: TRUE, FALSE	FALSE
Unicast_Pause_MAC_Address_1	Value in the range 00 to FF	AA

Table 3-1: **XCO File Values and Default Values *(Cont'd)***

Parameter	XCO File Values	Default GUI Setting
Unicast_Pause_MAC_Address_2	Value in the range 00 to FF	BB
Unicast_Pause_MAC_Address_3	Value in the range 00 to FF	CC
Unicast_Pause_MAC_Address_4	Value in the range 00 to FF	DD
Unicast_Pause_MAC_Address_5	Value in the range 00 to FF	EE
Unicast_Pause_MAC_Address_6	Value in the range 00 to FF	FF
Client_Side_Data_Width	One of the following keywords: 8_bit, 16_bit	8_bit
Clock_Enable	One of the following keywords: TRUE, FALSE - set depending upon the physical interface.	TRUE
Address_Filter_Enable	One of the following keywords: TRUE, FALSE - no user control	FALSE

Output Generation

The output files generated from the CORE Generator tool are placed in the CORE Generator tool project directory. The list of output files includes

- the netlist file
- supporting CORE Generator software files
- release notes and other documentation
- subdirectories containing example design files
- scripts to run the core through the Xilinx back-end tools and to simulate the core using the Mentor Graphics ModelSim, Cadence IES, or Synopsys VCS simulators.

See Chapter 16, Quick Start Example Design for more information about the CORE Generator software output files and Chapter 17, Detailed Example Design for details on the HDL example design.

XILINX®

Chapter 4

Virtex-6 Embedded Tri-Mode Ethernet MAC Overview

This chapter introduces the V6EMAC architecture, including all interfaces and the major functional blocks.

This chapter contains the following sections:

Key Features

The key features of the V6EMAC are:

- Designed to the *IEEE Std 802.3-2008* specification
- Supports three separate speed options
  - 10/100/1000 Mb/s Ethernet MAC
  - 1 Gb/s Ethernet MAC
  - 10/100 Mb/s Ethernet MAC
- Configurable duplex operation (at 10/100Mb/s speeds only)
- Support for Media Independent Interface (MII), Gigabit Media Independent Interface (GMII), Reduced Gigabit Media Independent Interface (RGMII), Serial Gigabit Media Independent Interface (SGMII) or 1000BASE-X PCS/PMA.
- Management Data Input/Output (MDIO) interface to manage objects in the physical layer
- User-accessible raw statistic vector outputs
- Optional built in statistics counters
- Support for VLAN frames
- Configurable interframe gap (IFG) adjustment in full-duplex operation
- Configurable in-band Frame Check Sequence (FCS) field passing on both transmit and receive paths
- Auto padding on transmit and stripping on receive paths
- Fully memory mapped processor interface
- Configured and monitored through an AXI4-Lite Interface

- Configurable flow control through Ethernet MAC Control PAUSE frames; symmetrically or asymmetrically enabled
- Configurable support for jumbo frames of any length
- Configurable receive address/frame filter
- AXI4-Stream user interface

Note: Virtex®-6 devices support GMII and MII at 2.5V only.

Architecture Description

Figure 4-1 illustrates a block diagram of the core.

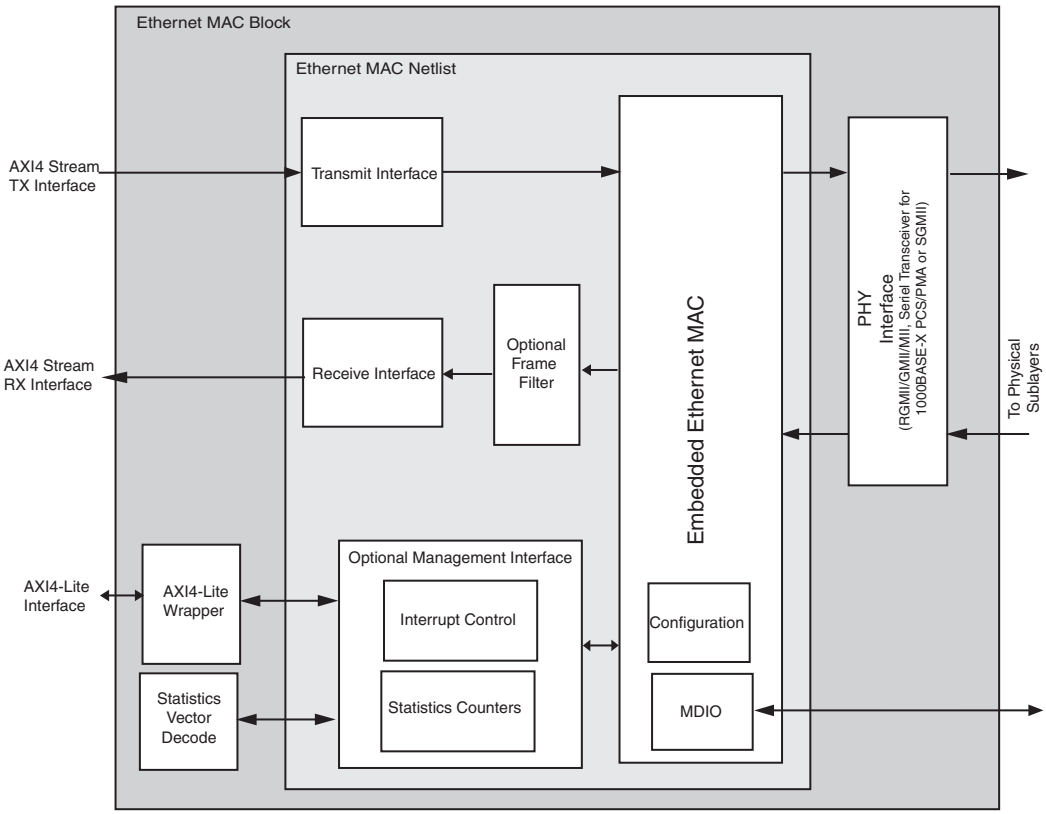

Figure 4-1: **Tri-Mode Ethernet MAC Block Diagram**

Core Components

The major functional blocks of the MAC are:

Ethernet MAC Block

The Ethernet MAC block is provided as part of the HDL example design and includes the basic blocks required to use the Ethernet MAC (V6EMAC) netlist. The Ethernet MAC Block should be instantiated in all designs that use the core.

AXI4-Lite Wrapper

The AXI4-Lite Wrapper allows the MAC netlist to be connected to an AXI4-Lite Interface and drives the V6EMAC netlist through a processor independent IPIF.

Statistics Vector Decode

The Statistics Vector Decode interprets the RX and TX statistics vectors supplied by the MAC netlist on a per frame basis and generates the Statistics counter increment controls. This code is provided as editable HDL to enable specific Statistics counter requirements to be met.

PHY Interface

The PHY Interface provides the required logic to interface to the PHY using RGMII, GMII, MII, SGMII or 1000BASE-X PCS/PMA.

When connecting to external PHY devices, either a parallel interface standard (GMII/MII or RGMII) can be used or one of the two supported serial standards (SGMII or 1000BASE-X PCS/PMA): these are selectable from the CORE Generator™ tool GUI (see Chapter 3, Generating the Core). In each case, either the required IOB logic or a serial transceiver is provided in the example design.

These external interfaces are described in the following chapters:

- Chapter 9, Media Independent Interface (MII)
- Chapter 10, Gigabit Media Independent Interface (GMII)
- Chapter 11, Reduced Gigabit Media Independent Interface (RGMII)
- Chapter 12, Serial Gigabit Media Independent Interface (SGMII)
- Chapter 13, 1000BASE-X PCS/PMA (GPCS)

Embedded Ethernet MAC

The Embedded Ethernet MAC, as shown in Figure 4-2, contains the key blocks to enable control of the PHY, whether Internal in the form of the PCS/PMA Sublayer or External, and provides the necessary protocol conversions to allow reception and transmission of data.

Further information about the EMAC is available in the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide* [Ref 1].

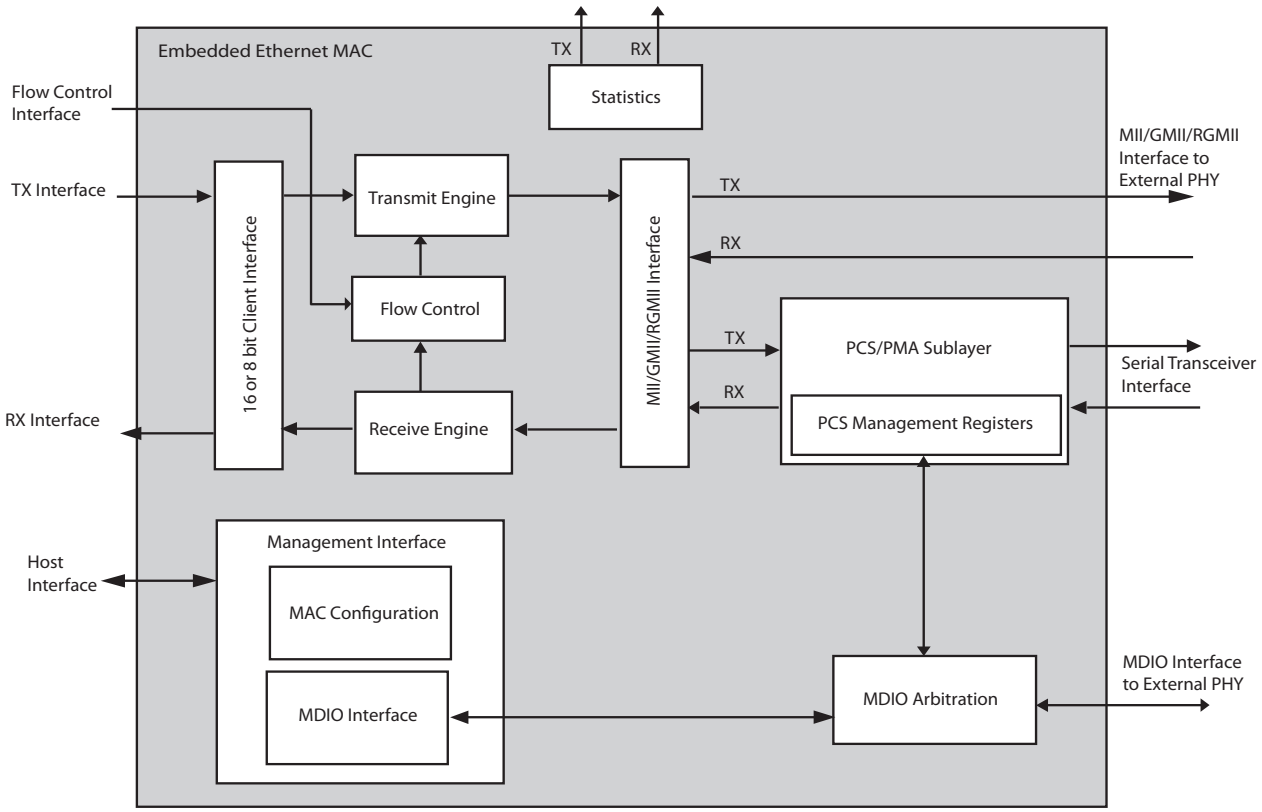

Figure 4-2: **Embedded Ethernet MAC Block Diagram**

Transmit Engine

The transmit engine takes data from the TX interface and converts it to GMII format. Preamble and frame check sequence fields are added and the data is padded if necessary. The transmit engine also provides the transmit statistics vector for each packet and transmits the pause frames generated by the flow control module.

Receive Engine

The receive engine takes the data from the GMII/MII interface and checks it for compliance to *IEEE Std 802.3-2008* [Ref 4]. Padding fields are removed and the RX interface is presented with the data along with a good/bad indication. The receive engine also provides the receive statistics vector for each received packet.

Flow Control

The flow control block is designed to *IEEE Std 802.3-2008* [Ref 4], clause 31. The MAC can be configured to send pause frames with a programmable pause value and to act on their reception. These two behaviors can be configured asymmetrically.

For further information, see Chapter 7, Flow Control.

16 or 8 bit Client Interface

To enable support of overclocking features for the 1000BASE-X physical interface, this allows the user data bus to increase to 16 bits.

MII/GMII/RGMII Interface

The Transmit and Receive Engines convert to a standard GMII format. This block implements the necessary logic to simplify the conversion to the required external interface.

PCS/PMA Sublayer

This is a key component of the Embedded EMAC and enables support of both SGMII and 1000BASE-X PCS/PMA serial standards. This block is described in more detail in Chapter 13, 1000BASE-X PCS/PMA (GPCS).

Management Interface

This block provides access to the various EMAC control/status registers and enables control of the MDIO master. The memory map as described by UG368 is superceded by that described in Address Map in Chapter 8.

MDIO Arbitor

The MDIO interface can operate as either a master or a slave depending upon the core configuration. The arbitor ensures the PCS Management Registers are available in either case.

Transmit Interface

The Transmit Interface takes data from the AXI4-Stream TX interface and converts it to the client interface format expected by the Embedded MAC (EMAC) core.

The transmitter user interface of the core is defined in Chapter 6, AXI4-Stream User Interface.

Receive Interface

The Receive Interface takes the data from the client interface provided by the EMAC Core and converts it to the required AXI4-Stream RX Interface.

The receiver user interface of the core is defined in Chapter 6, AXI4-Stream User Interface.

Management Interface

The optional Management Interface is a processor-independent interface with standard address, data, and control signals. It is used for the configuration and monitoring of the MAC and for access to the MDIO Interface, with automatic conversion to the EMAC's Host interface. It is supplied with a wrapper to interface to the industry standard AXI4-Lite. This interface is optional. If it is not present, the device attributes can be initialized using options in the GUI. See Chapter 8, Configuration and Status.

Address/Frame Filter

The V6EMAC can be implemented with an optional Address/Frame Filter. If the Address/Frame Filter is enabled, the device does not pass frames to the user that do not match with a configurable pattern.

Statistics Counters

The V6EMAC can be implemented with optional Statistics Counters.

Interrupt Controller

The V6EMAC can be implemented with an optional interrupt controller. This is present whenever the Management interface is set to AXI4-Lite and provides an interrupt output based on the MDIO status.

Core Interfaces

All ports of the netlist are internal connections in the Field Programmable Gate Array (FPGA) logic. An example HDL design, available in both VHDL and Verilog, is delivered with each core. The example design connects the core to a FIFO-based loopback example design and adds IOB flip-flops to the external signals of the GMII/MII (or RGMII).

All clock management logic is placed in this example design, allowing you more flexibility in implementation (for example, in designs using multiple cores).

All User Interfaces are described assuming the Block level of the example design.

User Interfaces

Transmitter Interface

Table 4-1 defines the AXI4-Stream transmit signals of the core, which are used to transmit data from the user to the core. Table 4-2 defines transmit sideband signals. See Transmitting Outbound Frames.

Table 4-1: **Transmit Interface AXI4-Stream Signal Pins**

Signal	Direction	Description
tx_axis_mac_tdata[7:0] or [15:0][(2)]	Input	Frame data to be transmitted.
tx_axis_mac_tkeep[1:0]	Input	Control signal for tx_axis_mac_tdata port. Indicates which bytes of tdata[15:0] are valid (16-bit AXI4- Stream Interface only).
tx_axis_mac_tvalid	Input	Control signal for tx_axis_mac_tdata port. Indicates the data is valid.
tx_axis_mac_tlast	Input	Control signal for tx_axis_mac_tdata port. Indicates the final transfer in a frame.
tx_axis_mac_tuser	Input	Control signal for tx_axis_mac_tdata port. Indicates an error condition, such as FIFO underrun, in the frame allowing the MAC to send an error to the PHY.
tx_axis_mac_tready	Output	Handshaking signal. Asserted when the current data on tx_axis_mac_tdata has been accepted and tx_axis_mac_tvalid is high. At 10/100Mb/s this is used to meter the data into the core at the correct rate.

Notes:

1. All signals are active-High
2. The 16-bit option is only available when using the 1000BASE-X PCS/PMA physical interface and the 2 or 2.5 Gb/s overclocking option.

Table 4-2: **Transmit Interface Sideband Signal Pins**

Signal	Direction	Description
tx_ifg_delay[7:0]	Input	Control signal for configurable interframe gap. See Interframe Gap Adjustment: Full-Duplex Mode Only for timing diagrams.
tx_collision	Output	Asserted by the MAC netlist to signal a collision on the medium and that any transmission in progress should be aborted. Always 0 when the MAC netlist is in full-duplex mode.
tx_retransmit	Output	When asserted at the same time as the tx_collision signal, this signals to the client that the aborted frame should be resupplied to the MAC netlist for retransmission. Always 0 when the MAC netlist is in full-duplex mode.
tx_statisitics_vector[31:0]	Output	A statistics vector that gives information on the last frame transmitted. See Transmitter Statistics Vector for vector contents.
tx_statistics_valid	Output	Asserted at end of frame transmission, indicating that the tx_statistics_vector is valid.

Notes:

1. All signals are active-High.

Receiver Interface

Table 4-3 describes the receive AXI4-Stream signals used by the core to transfer data to the user. Table 4-4 describes the related sideband interface signals.

Table 4-3: **Receive Interface AXI4-Stream Signal Pins**

Signal	Direction	Description
rx_axis_mac_tdata[7:0] or [15:0][(2)]	Output	Frame data received is supplied on this port.
rx_axis_mac_tkeep[1:0]	Output	Control signal for rx_axis_mac_tdata port. Indicates which bytes of tdata[15:0] are valid (16-bit AXI4- Stream Interface only).
rx_axis_mac_tvalid	Output	Control signal for the rx_axis_mac_tdata port. Indicates the data is valid.
rx_axis_mac_tlast	Input	Control signal for the rx_axis_mac_tdata port. Indicates the final transfer in the frame.
rx_axis_mac_tuser	Output	Control signal for rx_axis_mac_tdata. Asserted at end of frame reception to indicate that the frame had an error. See Normal Frame Reception. and Frame Reception with Errors.
rx_axis_filter_tuser[x:0]	Output	Per Frame filter tuser output. Can be used to send only data passed by a specific Frame filter.

Notes:

1. All signals are active-High.
2. The 16-bit option is only available when using the 1000BASE-X PCS/PMA physical interface and the 2 or 2.5 Gb/s overclocking option.

Table 4-4: **Receive Interface Sideband Signal Pins**

Signal	Direction	Description
rx_statistics_vector[27:0]	Output	Provides information about the last frame received. See Receiver Statistics Vector for the vector contents.
rx_statistics_valid	Output	Asserted at end of frame reception, indicating that the rx_statistics_vector is valid.

Notes:

1. All signals are active-High.

Flow Control Interface

Table 4-5 describes the signals used by the client to request a flow control action from the transmit engine. See Chapter 7, Flow Control.

Table 4-5: **Flow Control Interface Signal Pinout**

Signal	Direction	Description
pause_req	Input	Pause request: Upon request the MAC transmits a pause frame upon the completion of the current data packet. See Transmitting a Pause Control Frame.
pause_val[15:0]	Input	Pause value: inserted into the parameter field of the transmitted pause frame.

Notes:

1. All signals are active-High.

AXI4-Lite Signal Definition

Table 4-6 describes the optional signals used by the user to access the MAC netlist, including configuration, status and MDIO access. See The Management Interface.

Table 4-6: **Optional AXI4-Lite Signal Pinout**

Signal	Direction	Description
s_axi_aclk	Input	Clock for AXI4-Lite
s_axi_resetn	Input	Local reset for the clock domain
s_axi_awaddr[31:0]	Input	Write Address
s_axi_awvalid	Input	Write Address Valid
s_axi_awready	Output	Write Address ready
s_axi_wdata[31:0]	Input	Write Data
s_axi_wvalid	Input	Write Data valid
s_axi_wready	Output	Write Data ready
s_axi_bresp[1:0]	Output	Write Response
s_axi_bvalid	Output	Write Response valid

Table 4-6: **Optional AXI4-Lite Signal Pinout *(Cont'd)***

Signal	Direction	Description
s_axi_bready	Input	Write Response ready
s_axi_araddr[31:0]	Input	Read Address
s_axi_arvalid	Input	Read Address valid
s_axi_arready	Output	Read Address ready
s_axi_rdata[31:0]	Output	Read Data
s_axi_rresp[1:0]	Output	Read Response
s_axi_rvalid	Output	Read Data/Response Valid
s_axi_rready	Input	Read Data/Response ready

Clock, Reset and Control Signal Definition

Table 4-7 describes the reset signals, control signals and the clock signals that are available at the block level of the core. Any clock resources which can be shared amongst multiple cores are instantiated in the top level of the example design provided with the core.

Table 4-7: **Clock, Reset and Control Signals**

Signal	Direction	Description
glbl_rstn	Input	Active-Low asynchronous reset for entire core.
rx_axi_rstn	Input	Active-Low RX domain reset
tx_axi_rstn	Input	Active-Low TX domain reset
rx_reset	Output	Active-High RX software reset from MAC netlist
tx_reset	Output	Active-High TX software reset from MAC netlist
gtx_clk	Input	Global 125 MHz clock (or a higher frequency when using the overclocked 1000BASE-X physical interface)
clk125_out	Output	125 MHz clock output sourced from the transceiver. Only present when using either SGMII or 1000BASE-X PCS/PMA.
tx_mac_aclk	Output	Clock for the transmission of data on the physical interface. This clock should be used to clock the physical interface transmit circuitry and the TX AXI4-Stream transmit circuitry. This clock only exists in RGMII, GMII or MII. See the appropriate section: • Chapter 9, Media Independent Interface (MII) • Chapter 10, Gigabit Media Independent Interface (GMII) • Chapter 11, Reduced Gigabit Media Independent Interface (RGMII)

Table 4-7: **Clock, Reset and Control Signals *(Cont'd)***

Signal	Direction	Description
rx_mac_aclk	Output	Clock for the reception of data on the physical interface. This clock should be used to clock the physical interface receive circuitry and the RX AXI4-Stream receive circuitry. See the appropriate section: • Chapter 9, Media Independent Interface (MII) • Chapter 10, Gigabit Media Independent Interface (GMII) • Chapter 11, Reduced Gigabit Media Independent Interface (RGMII)
user_mac_aclk	Output	Clock for transmission and reception of data used by the AXI4-Stream logic when SGMII is selected.
clk_ds	Input	Clock source for the transceiver. This is a reference clock dedicated to the transceivers and can be shared amongst multiple TEMAC instances. This is only present when using either SGMII or 1000BASE-X PCS/PMA.
base_x_switch	Input	Only available when using Tri-speed SGMII with a management interface. This enables the core to dynamically switch between SGMII and 1000BASE-X PCS/PMA.

Physical Interface Signals

MDIO Signal Definition

Table 4-8 describes the MDIO (MII Management) interface signals of the core. The MDIO format is defined in *IEEE Std 802.3-2008* [Ref 4], clause 22. These signals are present whenever the optional Management Interface is used; they are typically connected to the MDIO port of a PHY device, either off-chip or an SoC-integrated core.

Table 4-8: **MDIO Interface Signal Pinout**

Signal	Direction	Description
mdc	Output	MDIO Management Clock: derived from s_axi_aclk on the basis of supplied configuration data when the optional Management Interface is used. This is present when the management interface is available and MDIO has been selected.
mdc_in	Input	MDIO Management Clock Input. This is present when the MDIO is selected but no management interface is present. In this case the MDIO acts as a slave allowing access to PCS Management registers.
mdio_i	Input	Input data signal for communication with PHY configuration and status. Tie high if unused.
mdio_o	Output	Output data signal for communication with PHY configuration and status.
mdio_t	Output	3-state control for MDIO signals; 0 signals that the value on MDIO_OUT should be asserted onto the MDIO bus.

PHY Interface Signal Definition

Table 4-9 through Table 4-11 describe the possible PHY interface standards supported, 1000BASE-X PCS/PMA, SGMII, RGMII, GMII and MII, which are typically attached to a PHY module, either off-chip or internally integrated. The RGMII is defined in *Reduced Gigabit Media Independent Interface (RGMII)*, version 2.0. The GMII is defined in *IEEE Std 802.3-2008* [Ref 4], clause 35. The MII is defined in *IEEE Std 802.3-2008* [Ref 4], clause 22. The SGMII is defined in the *Serial GMII specification v1.7*. The 1000BASE-X PCS/PMA is defined in *IEEE Std 802.3-2008* [Ref 4], clauses 36 and 37.

Table 4-9: **Optional GMII Interface Signal Pinout**

Signal	Direction	Description
gmii_tx_clk	Output	Clock to PHY
gmii_txd[7:0]	Output	Transmit data to PHY
gmii_tx_en	Output	Data Enable control signal to PHY
gmii_tx_er	Output	Error control signal to PHY
mii_tx_clk	Input	Clock from PHY (used for 10/100)
gmii_col	Input	Control signal from PHY
gmii_crs	Input	Control signal from PHY
gmii_rxd[7:0]	Input	Received data from PHY
gmii_rx_dv	Input	Data Valid control signal from PHY
gmii_rx_er	Input	Error control signal from PHY
gmii_rx_clk	Input	Clock from PHY

Table 4-10: **Optional MII Interface Signal Pinout**

Signal	Direction	Description
mii_tx_clk	Input	Clock from PHY
mii_txd[3:0]	Output	Transmit data to PHY
mii_tx_en	Output	Data Enable control signal to PHY
mii_tx_er	Output	Error control signal to PHY
mii_col	Input	Control signal from PHY
mii_crs	Input	Control signal from PHY
mii_rxd[3:0]	Input	Received data from PHY
mii_rx_dv	Input	Data Valid control signal from PHY
mii_rx_er	Input	Error control signal from PHY
mii_rx_clk	Input	Clock from PHY

Table 4-11: **Optional RGMII Interface Signal Pinout**

Signal	Direction	Description
rgmii_txd[3:0]	Output	Transmit data to PHY
rgmii_tx_ctl	Output	control signal to PHY
rgmii_txc	Output	Clock to PHY
rgmii_rxd[3:0]	Input	Received data from PHY
rgmii_rx_ctl	Input	Control signal from PHY
rgmii_rxc	Input	Clock from PHY
inband_link_status	Output	Link Status from PHY
inband_clock_speed	Output	Clock Speed from PHY
inband_duplex_status	Output	Duplex Status from PHY

Table 4-12: **Optional SGMII or 1000BASE-X PCS/PMA Interface Signal Pinout**

Signal	Direction	Description
txp	Output	Transmit data to PHY
txn	Output	Transmit data to PHY
rxp	Input	Received data from PHY
rxn	Input	Received data from PHY
phyad[4:0]	Input	Internal PHYaddress
resetdone	Output	Transceiver Reset complete
syncacqstatus	Output	Sync acquired status from PHY

XILINX®

Chapter 5

Designing with the Core

This chapter provides guidelines for creating designs using the V6EMAC. For more information about the example design included with the V6EMAC, see Chapter 16, Quick Start Example Design and Chapter 17, Detailed Example Design.

General Design Guidelines

This section describes the steps required to turn the V6EMAC into a fully-functioning design integrated with user application logic. It is important to recognize that not all designs require all the design steps defined in this chapter. The following sections discuss the design steps required for various implementations; follow the logic design guidelines carefully.

Design Steps

Generate the core using the CORE Generator™ software. See Chapter 3, Generating the Core.

Using the Example Design as a Starting Point

The core is delivered through the CORE Generator software with an HDL example design built around the core, allowing the functionality of the core to be demonstrated using either a simulation package or in hardware, if placed on a suitable board. Figure 5-1 is a block diagram of the example design. For more information about the example design, see Chapter 16, Quick Start Example Design.

The example design illustrates how to:

- Instantiate the core from HDL.
- Source and use the user-side interface ports of the core from application logic.
- Connect the physical-side interface of the core (GMII/MII or RGMII) to device IOBs creating an external interface. See the relevant physical interface chapters in this guide.
- Derive the clock logic, required for the core. See the relevant physical interface chapters in this guide:
  - Chapter 9, Media Independent Interface (MII)
  - Chapter 10, Gigabit Media Independent Interface (GMII)
  - Chapter 11, Reduced Gigabit Media Independent Interface (RGMII)

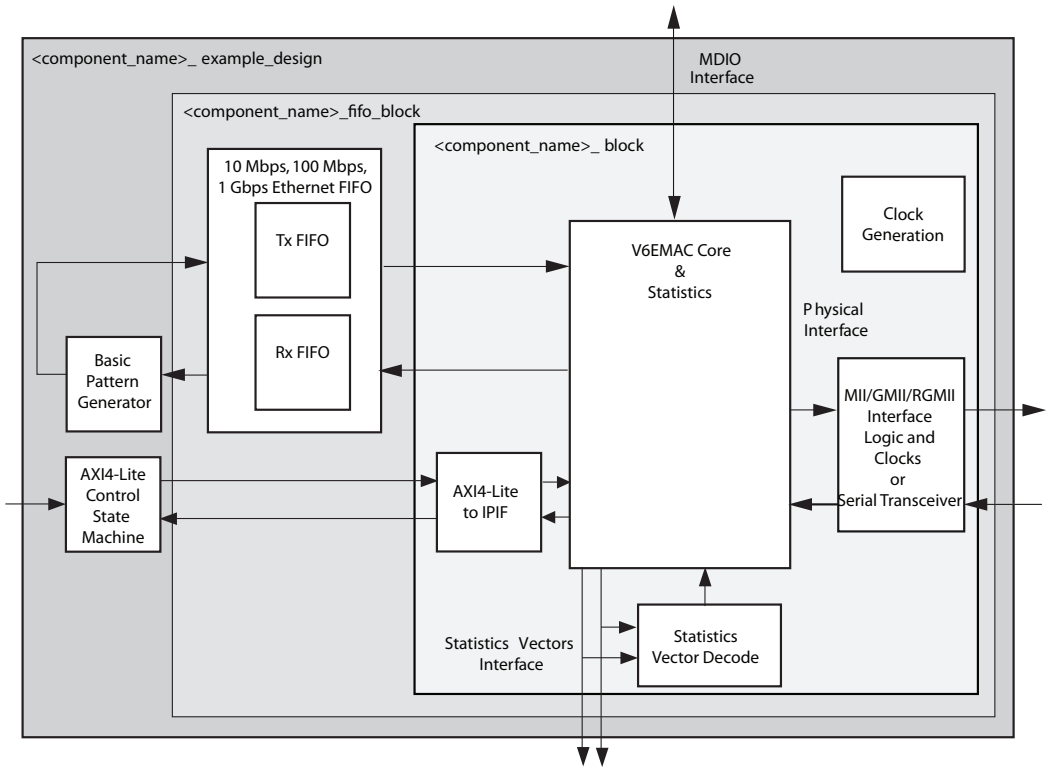

Figure 5-1: **Tri-Mode Ethernet MAC Core Example Design**

Using the example design as a starting point, you can do the following:

- Edit the HDL top level of the example design file to:
  - Change the clocking scheme.
  - Add/remove IOBs as required.
  - Replace the basic pattern generator logic with your specific application logic.
  - Adapt the 10 Mb/s, 100 Mb/s, 1 Gb/s Ethernet FIFO to suit your specific application (see 10 Mb/s / 100 Mb/s / 1 Gb/s Ethernet FIFO).
  - Remove the AXI4-Lite Control State machine and directly drive the AXI4-Lite bus from a processor.
- Synthesize the entire design.

  The Xilinx® Synthesis Tool (XST) script and Project file in the `/implement` directory can be adapted to include any HDL files you might want to add.
- Run the implement script in the `/implement` directory to create a top-level netlist for the design. The script can also run the Xilinx tools `map`, `par`, and `bitgen`, creating a bitstream that can be downloaded to a Xilinx device. SimPrim-based simulation models for the entire design are also produced by the implement scripts.
- Simulate the entire design using the demonstration test bench provided as a template in the `/simulation` directory.
- Download the bitstream to a target device.

Implementing the V6 Embedded Tri-Mode Ethernet MAC in Your Application

The example design can be studied as an example of how to do the following:

- Instantiate the core from HDL.
- Source and use the user-side interface ports of the core from application logic.
- Connect the physical-sideparallel interface of the core (GMII/MII or RGMII) to device IOBs to create an external interface.
- Connect the physical-side serial interface of the core to a serial transceiver to create an external interface.
- Derive the required clock logic.

After working with the example design and this User Guide, you can write your own HDL application, using single or multiple instances of the core.

You can synthesize the entire design using any synthesis tool. The core netlist is pre-synthesized and is delivered as an NGC netlist (which appears as a black box to synthesis tools).

Run the Xilinx tools `map`, `par`, and `bitgen` to create a bitstream that can be downloaded to a Xilinx device. Care must be taken to constrain the design correctly, and the UCF produced by the CORE Generator software should be used as the basis for the your own UCF. See Chapter 15, Implementing Your Design for more information.

You can simulate the entire design and download the bitstream to the target device.

Keep it Registered

To simplify timing and increase system performance in an FPGA design, keep all inputs and outputs registered between your application and the core. This means that all inputs and outputs from your application should come from, or connect to, a flip-flop. While registering signals might not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx tools to place-and-route the design.

Recognize Timing Critical Signals

The UCF provided with the example design identifies the critical signals and timing constraints that should be applied. See Chapter 14, Constraining the Core for more information.

Use Supported Design Flows

The core is pre-synthesized and delivered as an NGC netlist. The example implementation scripts provided use XST 14.1 as the synthesis tool for the HDL example design. Other synthesis tools can be used for your application logic. The core is always unknown to the synthesis tool and should appear as a black box. For post synthesis, only ISE® v14.1 tools are supported.

Make Only Allowed Modifications

You should not modify the core. Any modifications can have adverse effects on system timing and protocol compliance. Supported user configurations of the core can only be made by selecting the options in the CORE Generator tool when the core is generated. See Chapter 14, Constraining the Core.

XILINX®

Chapter 6

AXI4-Stream User Interface

This chapter provides a detailed description of the AXI4-Stream user-side interface. This interface must be used by the user-side logic to initiate frame transmission and accept frame reception to and from the core.

The definitions and abbreviations used in this chapter are described in Table 6-1.

Table 6-1: **Abbreviations Used in Timing Diagrams**

Abbreviation	Definition
DA	Destination address; 6 bytes
SA	Source address; 6 bytes
L/T	Length/type field; 2 bytes
FCS	Frame check sequence; 4 bytes

Receiving Inbound Frames

Received Ethernet frames are presented to the user logic on the receiver subset of the AXI4-Stream Interface. For port definition, see Receiver Interface.

All receiver signals are synchronous to the `rx_mac_aclk` clock, when present, or `user_mac_aclk` for SGMII and `gtx_clk` for 1000BASE-X PCS/PMA.

Normal Frame Reception

Figure 6-1 shows the timing of a normal inbound frame transfer. The user must be prepared to accept data at any time; there is no buffering within the MAC to allow for latency in the receive logic. After frame reception begins, data is transferred on consecutive validated cycles to the receive logic until the frame is complete. The MAC asserts the `rx_axis_mac_tlast` signal to indicate that the frame has completed with `rx_axis_mac_tuser` being used to indicate any errors.

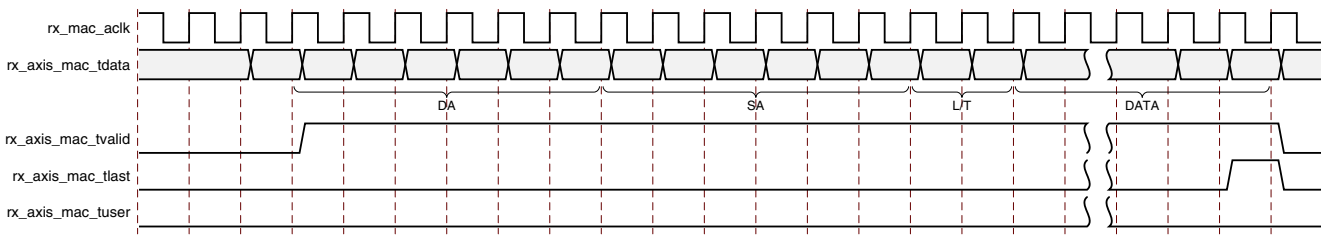

Figure 6-1: **Normal Frame Reception (8-bit Client Interface)**

Frame parameters (destination address, source address, length/type and optionally FCS) are supplied on the data bus according to the timing diagram. The abbreviations are described in Table 6-1.

If the length/type field in the frame has the length interpretation, and this indicates that the inbound frame has been padded to meet the Ethernet minimum frame size specification, then this padding is not passed to the user in the data payload. The exception to this is in the case where FCS passing is enabled. See User-Supplied FCS Passing.

When user-supplied FCS passing is disabled, `rx_axis_mac_tvalid`= 0 between frames for the duration of the padding field (if present), the FCS field, carrier extension (if present), the interframe gap following the frame, and the preamble field of the next frame. When user-supplied FCS passing is enabled, `rx_axis_mac_tvalid` = 0 between frames for the duration of carrier extension (if present), the interframe gap, and the preamble field of the following frame.

When the 16-bit client interface is used, frame reception is generally the same as described for the 8-bit case. However, the 16-bit client interface adds the `rx_axis_mac_tkeep[1:0]` bus and presents the received data two bytes per clock cycle. Figure 6-2 shows the timing of a normal inbound 16-bit interface frame transfer.

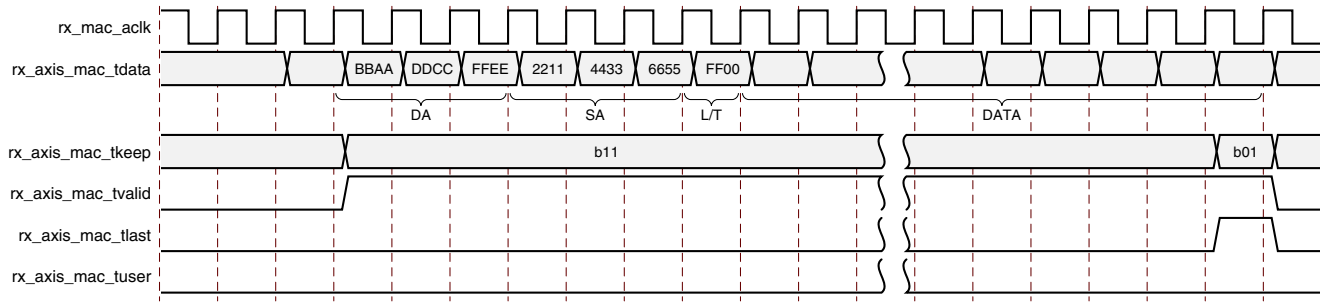

Figure 6-2: **Normal Frame Reception (16-bit Client Interface)**

With the 16-bit client interface, the 6-byte Destination Address and 6-byte Source Address field reception each take 3 cycles of `rx_mac_aclk`, while the two-byte Length/Type field is received in one cycle of `rx_mac_aclk`. The 16-bit interface presents the received data with the first byte received over the physical interface on `rx_axis_mac_tdata[7:0]`, and the next byte on `rx_axis_mac_tdata[15:8]`. For the example shown in Figure 6-2, the Destination Address was received over the physical interface in the order AA-BB-CC-DD-EE-FF, and the Length/Type field is 0x00FF (or 255 decimal). Figure 6-2 also shows an example of `rx_axis_mac_tkeep[1:0]` usage, with this example indicating only the `tdata[7:0]` byte is valid on the final data beat of the packet. Both `tkeep[1:0]` bits are always asserted during normal frame reception, except when `rx_axis_mac_tlast` is asserted. When `rx_axis_mac_tlast` is asserted, the only valid values of `tkeep[1:0]` are 11 or 01.

rx_axis_mac_tlast and rx_axis_mac_tuser Timing

Although Figure 6-1 illustrates the `rx_axis_mac_tlast` signal asserted immediately after a cycle containing valid data on `rx_axis_mac_tdata`, this is not usually the case. The `rx_axis_mac_tlast` and `rx_axis_mac_tuser` signals are asserted, along with the final byte of the transfer, only after all frame checks are completed. This is after the FCS field has been received (and after reception of carrier extension, if present). This is shown in Figure 6-3.

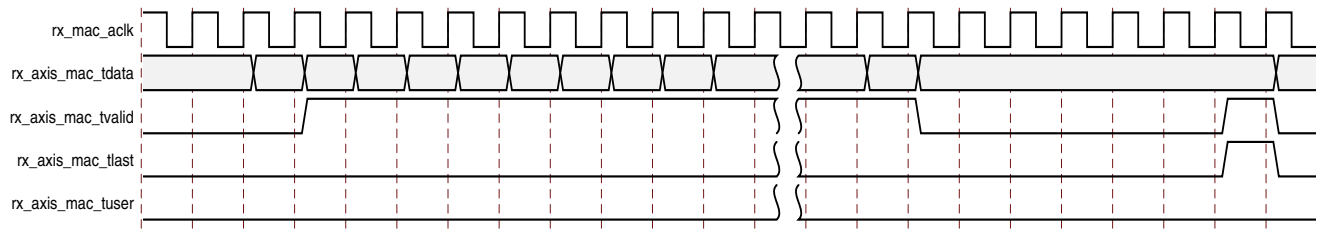

Figure 6-3: **Frame Reception TLast Timing**

Therefore, `rx_axis_mac_tlast` and possibly `rx_axis_mac_tuser` are asserted following frame reception at the beginning of the interframe gap.

Frame Reception with Errors

Figure 6-4 illustrates an unsuccessful frame reception (for example, a fragment frame or a frame with an incorrect FCS). In this case, the `rx_axis_mac_tuser` signal is asserted to the user at the end of the frame. It is then the responsibility of the user to drop the data already transferred for this frame.

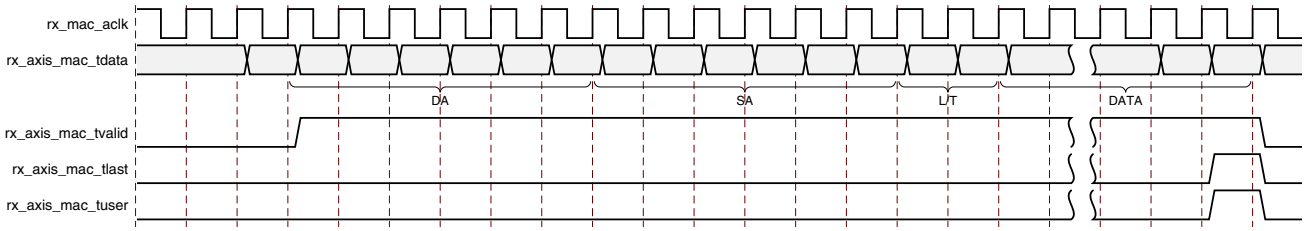

Figure 6-4: **Frame Reception with Error**

The following conditions cause the assertion of `rx_axis_mac_tuser`:

- FCS errors occur.
- Packets are shorter than 64 bytes (undersize or fragment frames).
- Jumbo frames are received when jumbo frames are not enabled.
- VLAN frames of length 1519-1522 are received when VLAN frames are not enabled.
- A value of 0x0000 to 0x002D is in the type/length field. In this situation, the frame should be padded to minimum length. If it is not padded to exactly minimum frame length, the frame is marked as bad (when length/type checking is enabled).
- A value of 0x002E to 0x0600 is in the type/length field, but the real length of the received frame does not match this value (when length/type checking is enabled).
- Any control frame that is received is not exactly the minimum frame length (unless control frame length checks are disabled: see Receiving a Pause Control Frame).
- An error is indicated on the phy interface at any point during frame reception.
- An error code is received in the 1-Gigabit frame extension field.
- A valid pause frame, addressed to the MAC, is received when flow control is enabled. See Overview of Flow Control for more information.
- A frame does not match against any of the enabled frame filters, if present.

- When in 1000BASE-X mode or SGMII mode, the following errors can also cause the assertion of `rx_axis_mac_tuser`:
  - Unrecognized 8B/10B code group received during the packet.
  - 8B/10B running disparity errors occur during the packet.
  - Unexpected K characters or sequences appearing in the wrong order/byte position.

User-Supplied FCS Passing

If the MAC core is configured to pass the FCS field to the user. It is handled as displayed in Figure 6-5.

In this case, any padding inserted into the frame to meet Ethernet minimum frame length specifications is left intact and passed to the user.

Even though the FCS is passed up to the user, it is also verified by the MAC core, and `rx_axis_mac_tuser` is asserted if the FCS check fails.

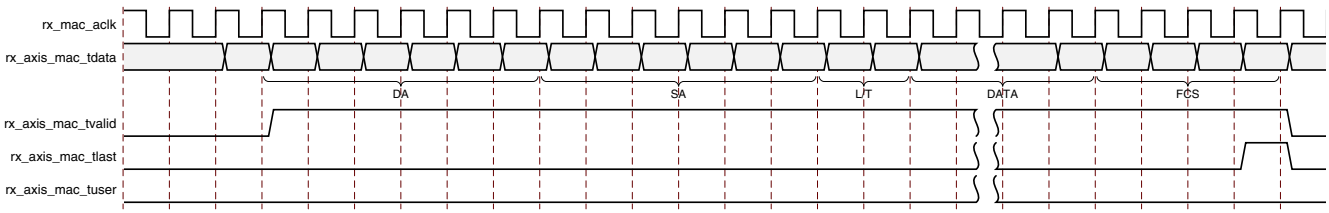

Figure 6-5: **Frame Reception with In-Band FCS Field**

VLAN Tagged Frames

The reception of a VLAN tagged frame can be seen in Figure 6-6. This frame is identified as being a VLAN frame by the inclusion of the VLAN type tag (81-00), located in the first two bytes following the Source Address. This is followed by the Tag Control Information bytes, V1 and V2. The length/type field after the tag control information is not checked for errors. More information on the interpretation of these bytes can be found in *IEEE Std 802.3-2008* [Ref 4].

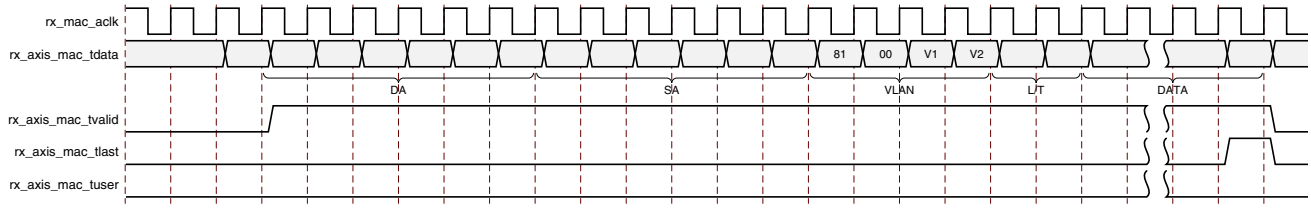

Figure 6-6: **Reception of a VLAN Tagged Frame**

Maximum Permitted Frame Length

The maximum legal length of a frame specified in *IEEE Std 802.3-2008* [Ref 4] is 1518 bytes for non-VLAN tagged frames. VLAN tagged frames can be extended to 1522 bytes. When jumbo frame handling is disabled and the core receives a frame which exceeds the maximum legal length, `rx_axis_mac_tuser` is asserted. When jumbo frame handling is

enabled, frames which are longer than the legal maximum are received in the same way as shorter frames.

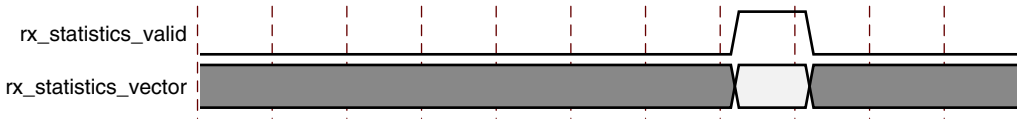

Figure 6-7: **Receiver Statistics Vector Timing**

Length/Type Field Error Checks

Enabled

Default operation is with the length/type error checking enabled. In this mode, the following checks are made on all frames received. If either of these checks fail, the frame is marked as bad.

- A value in the length/type field that is greater than or equal to decimal 46 but less than decimal 1536 (a Length interpretation) is checked against the actual data length received.
- A value in the length/type field that is less than decimal 46 is checked to see that the data field is padded to exactly 46 bytes (so that the resultant frame is minimum frame size: 64 bytes total in length).

Furthermore, if padding is indicated (the length/type field is less than decimal 46) and User-Supplied FCS Passing is disabled, then the length value in the length/type field is used to deassert `rx_axis_mac_tvalid` after the indicated number of data bytes so that the padding bytes are removed from the frame.

Disabled

When the length/type error checking is disabled and the length/type field has a length interpretation, the MAC does not check the length value against the actual data length received. A frame containing only this error is marked as good. However, if the length/type field is less than decimal 46, the MAC marks a frame as bad if it is not the minimum frame size of 64 bytes.

Furthermore, if padding is indicated and User-Supplied FCS Passing is disabled, then a length value in the length/type field is not used to deassert `rx_axis_mac_tvalid`. Instead `rx_axis_mac_tvalid` is deasserted before the start of the FCS field; in this way any padding is not removed from the frame.

Address/Frame Filter

If the optional address/frame filter is included in the core, the MAC is able to reject frames that do not match against a recognized pattern, that is, a specified destination address. If a frame is rejected within the destination address, the `rx_axis_mac_tvalid` signal is not asserted for the duration of the frame. The statistics vectors are still output with a valid pulse at the end of the rejected frame. If a frame is not rejected during the destination address then `rx_axis_mac_tvalid` is asserted as normal through the frame though the frame can still be rejected at a later point through the assertion of `rx_axis_mac_tuser` at the end of the frame.

This is described in more detail in Address/Frame Filter.

Receiver Statistics Vector

The statistics for the frame received are contained within the `rx_statistics_vector` output. Table 6-2 defines the bit field for the vector.

All bit fields, with the exception of BYTE_VALID are valid only when the `rx_statistics_valid` is asserted, as illustrated in Figure 6-7. BYTE_VALID is significant on every validated receiver cycle.

Table 6-2: **Bit Definition for the Receiver Statistics Vector**

emacclientrxstats	Name	Description
27	ADDRESS_MATCH	If the optional address filter is included in the core, this bit is asserted if the address of the incoming frame matches one of the stored or pre-set addresses in the address filter. If the address filter is omitted from the core or is configured in promiscuous mode, this line is held high.
26	ALIGNMENT_ERROR	Asserted at speeds less than 1 Gb/s if the frame contains an odd number of nibbles and the FCS for the frame is invalid.
25	LENGTH/TYPE Out of Range	If the length/type field contained a length value that did not match the number of MAC client data bytes received and the length/type field checks are enabled, then this bit is asserted. This bit is also asserted if the length/type field is less than 46, and the frame is not padded to exactly 64 bytes. This is independent of whether or not the length/type field checks are enabled.
24	BAD_OPCODE	Asserted if the previous frame was error-free and contained the special control frame identifier in the length/type field, but contained an opcode that is unsupported by the MAC (any opcode other than PAUSE).
23	FLOW_CONTROL_FRAME	Asserted if the previous frame was error-free, contained the special control frame identifier in the length/type field, contained a destination address that matched either the MAC Control multicast address or the configured source address of the MAC, contained the supported PAUSE opcode, and was acted upon by the MAC.
22	BYTE_VALID	Asserted if a MAC frame byte (destination address to FCS inclusive) is in the process of being received. This is valid on every clock cycle. Do not use this as an enable signal to indicate that data is present on emacclientrxd[7:0].
21	VLAN_FRAME	Asserted if the previous frame contained a VLAN identifier in the length/type field when receiver VLAN operation is enabled.
20	OUT_OF_BOUNDS	Asserted if the previous frame exceeded the specified *IEEE Std 802.3-2008* maximum legal length (see Maximum Permitted Frame Length). This is only valid if jumbo frames are disabled.
19	CONTROL_FRAME	Asserted if the previous frame contained the special control frame identifier in the length/type field.
18 down to 5	FRAME_LENGTH_COUNT	The length of the previous frame in number of bytes. The count stays at 16368 for any jumbo frames larger than this value.

Table 6-2: **Bit Definition for the Receiver Statistics Vector *(Cont'd)***

emacclientrxstats	Name	Description
4	MULTICAST_FRAME	Asserted if the previous frame contained a multicast address in the destination address field.
3	BROADCAST_FRAME	Asserted if the previous frame contained the broadcast address in the destination address field.
2	FCS_ERROR	Asserted if the previous frame received was correctly aligned but had an incorrect FCS value or the MAC detected error codes during frame reception.
1	BAD_FRAME(1)	Asserted if the previous frame received contained errors.
0	GOOD_FRAME(1)	Asserted if the previous frame received was error-free.

Notes:

1. If the length/type field error checks are disabled, a frame which has an actual data length that does not match the length/type field value is marked as a GOOD_FRAME providing no additional errors were detected. See Length/Type Field Error Checks for more information.

Transmitting Outbound Frames

Ethernet frames to be transmitted are presented to the user logic on the transmitter subset of the AXI4-Stream User-Side Interface. For port definition, see Transmitter Interface.

All transmitter signals are synchronous to the `tx_mac_aclk` clock if present or `gtx_clk` if not.

Normal Frame Transmission

The timing of a normal outbound frame transfer can be seen in Figure 6-8. When the user wants to transmit a frame, it places the first column of data onto the `tx_axis_mac_tdata` port and asserts a 1 onto `tx_axis_mac_tvalid`.

The MAC core accepts the first two bytes of data by asserting `tx_axis_mac_tready` and then waits until it is allowed to transmit; it then accepts the remainder of the frame. The user must be capable of supplying new data on the following cycle when data has been taken, indicated by the assertion of `tx_axis_mac_tready`.

The end of frame is signalled to the MAC core by asserting `tx_axis_mac_tlast` on the final byte of the frame. For maximum flexibility in switching applications, the Ethernet frame parameters (destination address, source address, length/type and optionally FCS) are encoded within the same data stream that the frame payload is transferred upon, rather than on separate ports.

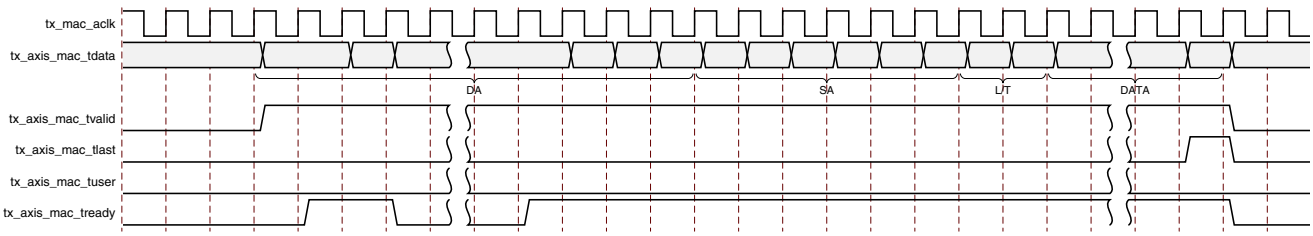

Figure 6-8: **Normal Frame Transmission Across 8-bit Client Interface**

When the 16-bit client interface is used, frame transmission is generally the same as described for the 8-bit case. However, the 16-bit client interface adds the `tx_axis_mac_tkeep[1:0]` bus and the user presents two data bytes per clock cycle. Figure 6-9 shows the timing of a normal outbound 16-bit interface frame transfer.

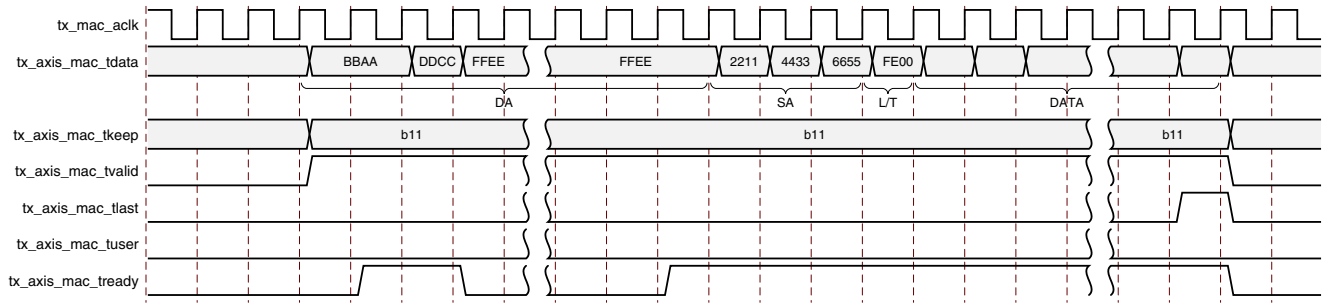

Figure 6-9: **Normal Frame Transmission Across 16-bit Client Interface**

With the 16-bit client interface, the 6-byte Destination Address and 6-byte Source Address field transmission each take 3 cycles of `tx_mac_aclk`, while the two-byte Length/Type field is transferred into the MAC in one cycle of `tx_mac_aclk`. The 16-bit interface expects the user data presented with the first byte to be transmitted over the physical interface on `tx_axis_mac_tdata[7:0]`, and the next byte on `tx_axis_mac_tdata[15:8]`. For the example shown in Figure 6-9, the Destination Address is transmitted over the physical interface in the order AA-BB-CC-DD-EE-FF, and the Length/Type field is 0x00FE (or 254 decimal). Figure 6-9 also shows an example of `tx_axis_mac_tkeep[1:0]` usage, with this example indicating both bytes on `tdata[15:0]` are valid on the final data beat of the packet. The `tkeep[1:0]` bits must always both be asserted during normal frame transmission, except when `tx_axis_mac_tlast` is asserted. When `tx_axis_mac_tlast` is asserted, the only valid values of `tkeep[1:0]` are 11 or 01.

Padding

When fewer than 46 bytes of data are supplied by the user to the MAC core, the transmitter module adds padding up to the minimum frame length. The exception to this is when the MAC core is configured for user-passed FCS; in this case the user must also supply the padding to maintain the minimum frame length.

User-Supplied FCS Passing

If the MAC core is configured to have the FCS field passed in by the user, the transmission timing is as depicted in Figure 6-10. In this case, it is the responsibility of the user to ensure that the frame meets the Ethernet minimum frame length requirements. If frame length requirements are not met, the core appends zeroes at the end of the supplied frame to meet the minimum frame length. Although this does not cause the Transmitter Statistics Vector to indicate a bad frame, it results in a frame error as received by the link partner MAC (due to the detection of an FCS error).

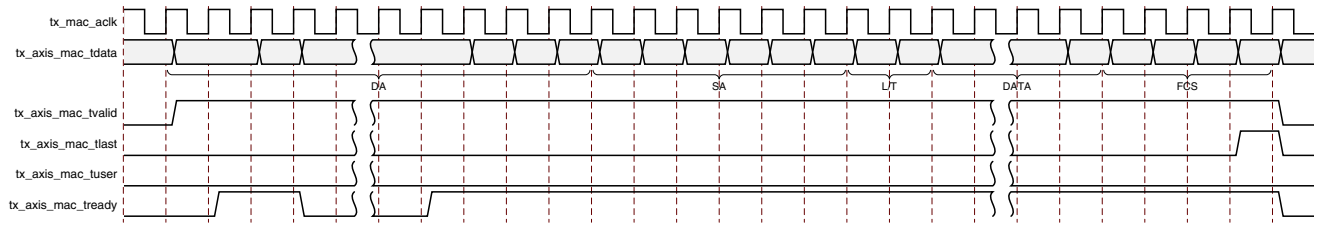

Figure 6-10: **Frame Transmission with Client-Supplied FCS**

User Error Indication

Figure 6-11 shows an example of the timing for an aborted transfer. This can occur, for example, if a FIFO connected to the AXI4-Stream TX interface empties before a frame is completed. When the user asserts `tx_axis_mac_tuser` during a frame transmission, the MAC core inserts an error code to corrupt the current frame and then falls back to idle transmission. It is the responsibility of the user to re-queue the aborted frame for transmission. It is also possible to abort a frame by deasserting `tx_axis_mac_tvalid` before the final byte of the frame. It is classed by the MAC as a frame underrun as it does not buffer the data and any gap is passed directly to the PHY, to avoid incorrect data being output. This is therefore classed as an implicit error condition and the frame is aborted.

The `tx_axis_mac_tuser` signal can be asserted at any time during active frame transmission. If it occurs prior to the MAC accepting the third byte of the frame, indicating it is actively transmitting to the PHY, it is possible to provide new frame data to the MAC and avoid the transmission of the aborted frame entirely. If this new data is not provided or arrives too late then a minimum sized errored frame is output.

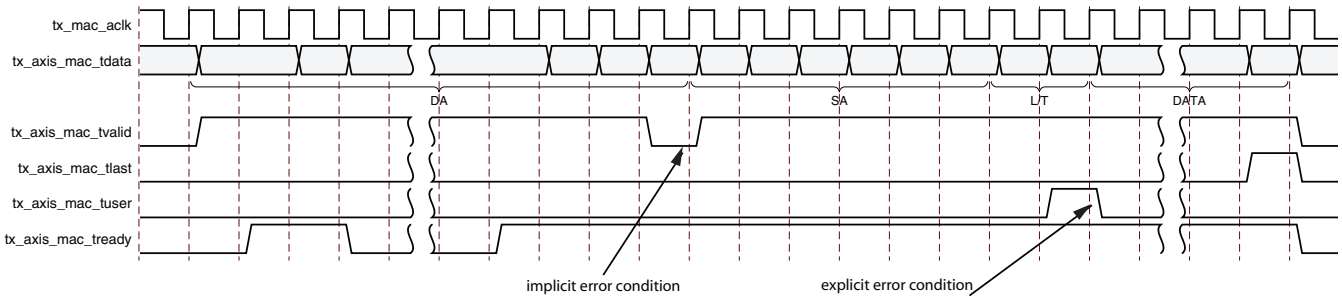

Figure 6-11: **Frame Transmission with Underrun**

Back-to-Back Transfers

Figure 6-12 shows the MAC user immediately ready to transmit a second frame of data following completion of its first frame. In this figure, the end of the first frame is shown on the left with the assertion of `tx_axis_mac_tlast`. On the cycle immediately following the final byte of the first frame, `tx_axis_mac_tvalid` remains high to indicate that the first byte of the destination address of the second frame is on `tx_axis_mac_tdata` awaiting transmission.

When the MAC core is ready to accept data, `tx_axis_mac_tready` is asserted and the transmission continues in the same manner as in the case of the single frame. The MAC core defers the assertion of `tx_axis_mac_tready` appropriately to comply with inter-packet gap requirements and flow control requests.

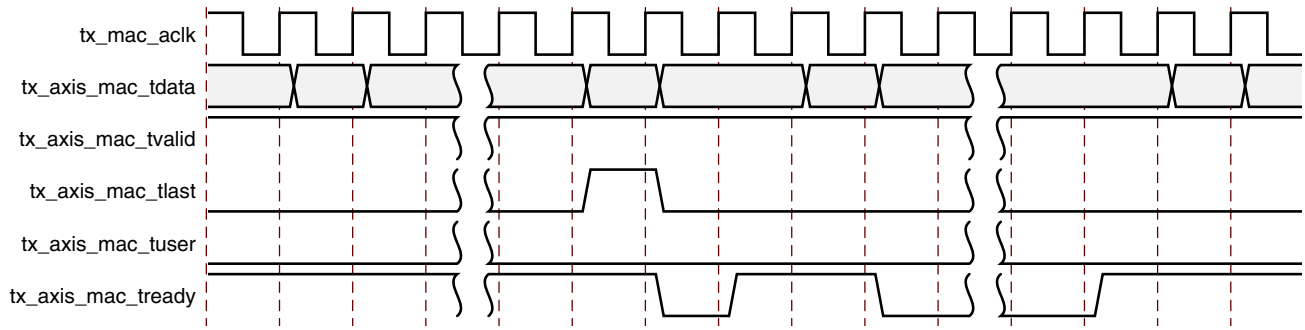

Figure 6-12: **Back-to-Back Frame Transmission**

VLAN Tagged Frames

Transmission of a VLAN tagged frame (if enabled) can be seen in Figure 6-13. The handshaking signals across the interface do not change; however, the VLAN type tag 81-00 must be supplied by the user to signify that the frame is VLAN tagged. The user also supplies the two bytes of Tag Control Information, V1 and V2, at the appropriate times in the data stream. More information on the contents of these two bytes can be found in *IEEE Std 802.3-2008* [Ref 4].

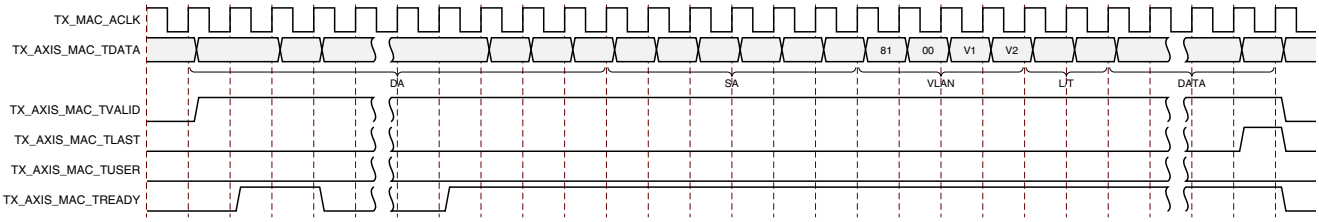

Figure 6-13: **Transmission of a VLAN Tagged Frame**

Maximum Permitted Frame Length

The maximum legal length of a frame specified in *IEEE Std 802.3-2008* [Ref 4] is 1518 bytes for non-VLAN tagged frames. VLAN tagged frames can be extended to 1522 bytes. When jumbo frame handling is disabled and the user attempts to transmit a frame which exceeds the maximum legal length, the MAC core inserts an error code to corrupt the current frame and the frame is truncated to the maximum legal length. When jumbo frame handling is enabled, frames which are longer than the legal maximum are transmitted error-free.

Frame Collisions: Half-Duplex Operation Only

In half-duplex Ethernet operation, which is only supported at 10/100 Mb/s in either MII/GMII or RGMII, collisions occur on the medium as a matter of course; this is how the arbitration algorithm is fulfilled. In the case of a collision, the MAC core signals to the user that data might need to be resupplied as follows.

- If there is a collision, the `tx_collision` signal is set to 1 by the MAC core. If a frame is in progress, the user must abort the transfer asserting `tx_axis_mac_tlast and tx_axis_mac_tuser`.
- If the `tx_retransmit` signal is 1 in the same cycle that the `tx_collision` signal is 1 the user must then resubmit the previous frame to the MAC core for retransmission;

`tx_axis_mac_tvalid` must be asserted to the MAC core within 7 cycles of the `tx_retransmit` signal: if `tx_axis_mac_tvalid` is asserted later than this, the MAC assumes that the frame is not retransmitted and the *number of retransmission attempts* counter within the MAC is reset. This case is illustrated in Figure 6-14.

If any frame presented to the user interface is shorter than the collision window (slot time) as defined in *IEEE Std 802.3-2008* [Ref 4], a retransmission request can occur after the end of the frame as observed on the user interface. Therefore, the user logic (which might have queued a subsequent frame for transmission) might then have to rewind back to the previous frame. In this case, the current frame has to be aborted by asserting `tx_axis_mac_tlast` in conjunction with `tx_axis_mac_tuser` and the previous frame data should be re-supplied on the `tx_axis_mac_tdata[7:0]` port within the same 7 cycles illustrated in Figure 6-14.

For reference only: the collision window (slot time) is 64 validated clock cycles when operating at 10 Mb/s and 100 Mb/s speeds (corresponding to 64-bytes of frame data).

- If the `tx_retransmit` signal is 0 in the same cycle that the `tx_collision` signal is 1 the number of retries for this frame has exceeded the Ethernet specification or the collision has been classed as late, and the frame should be dropped by the user. The user can then make any new frame available to the MAC for transmission without timing restriction. This case is illustrated in Figure 6-15.

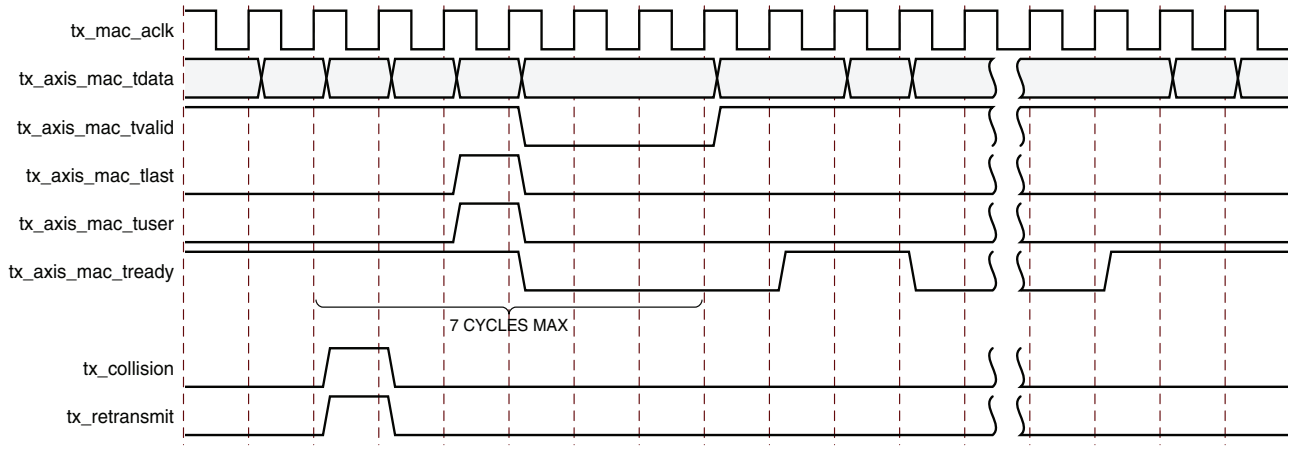

Figure 6-14: **Collision Handling: Frame Retransmission Required**

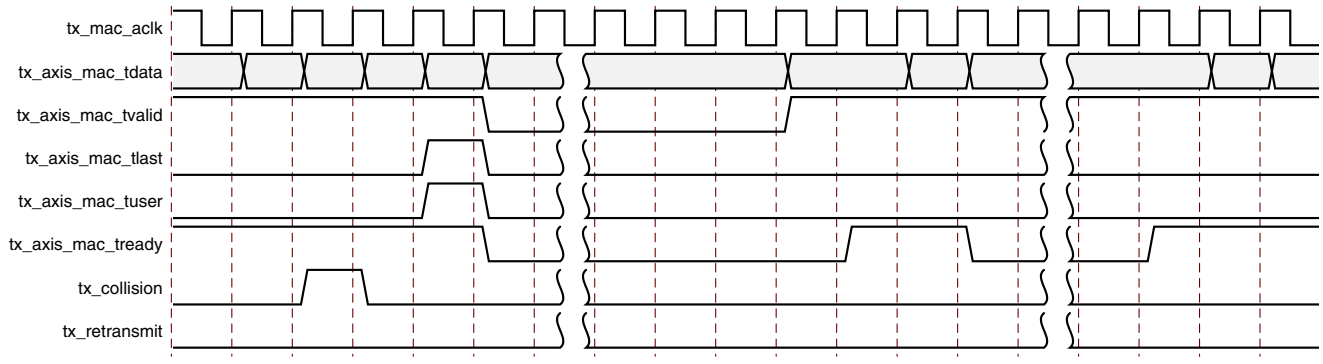

Figure 6-15: **Collision Handling: No Frame Retransmission Required**

Interframe Gap Adjustment: Full-Duplex Mode Only

A configuration bit in the transmitter control register allows you to control the length of the interframe gap transmitted by the MAC on the physical interface. If this function is selected, the MAC exerts back pressure on the user interface to delay the transmission of the next frame until the requested number of idle cycles has elapsed. The number of idle cycles is controlled by the value on the `tx_ifg_delay` port seen at the start of frame transmission on the user interface. Figure 6-16 shows the MAC operating in this mode.

The minimum interframe gap supported is 12 transmit clock cycles as specified in *IEEE Std 802.3-2008* [Ref 4]. This is the value used if either Interframe Gap Adjust Enable bit is set to 0 or IFG_DELAY is set to less than 12.

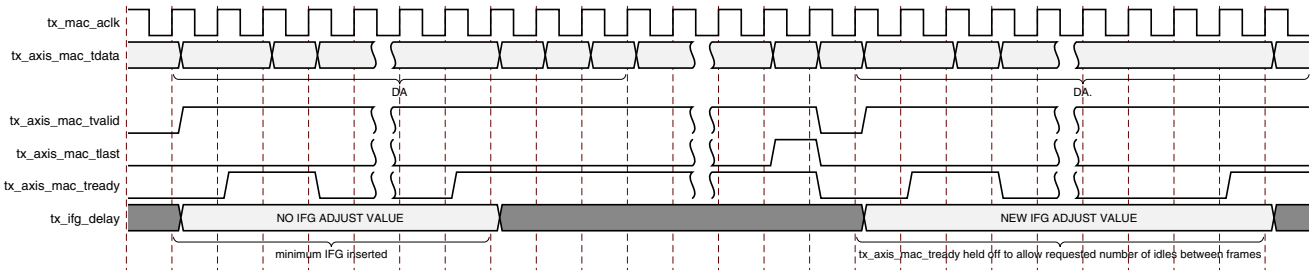

Figure 6-16: **Interframe Gap Adjustment**

Transmitter Statistics Vector

The statistics for the frame transmitted are contained within the `tx_statistics_vector` output. The bit field definition for the Vector is defined in Table 6-3. All bit fields, with the exception of BYTE_VALID are valid only when the `tx_statistics_valid` is asserted, as illustrated in Figure 6-17. BYTE_VALID is significant on every transmitter cycle that clock enable is high. `tx_statistics_vector` bits 28 down to 20 inclusive are for half-duplex only and are set to logic 0 when operating in full-duplex mode.

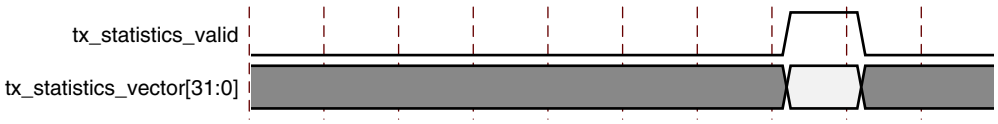

Figure 6-17: **Transmitter Statistics Vector Timing**

Table 6-3: **Bit Definition for the Transmitter Statistics Vector**

emacclienttxstats	Name	Description
31	PAUSE_FRAME_TRANSMITTED	Asserted if the previous frame was a pause frame that the MAC itself initiated in response to a pause_req assertion.
30	BYTE_VALID	Asserted if a MAC frame byte (DA to FCS inclusive) is in the process of being transmitted. This is valid on every clock cycle. Do not use this as an enable signal to indicate that data is present on (R)(G)MII_TXD.
29	Reserved	Returns logic 0.

Table 6-3: **Bit Definition for the Transmitter Statistics Vector *(Cont'd)***

emacclienttxstats	Name	Description
28 down to 25	TX_ATTEMPTS[3:0]	The number of attempts that have been made to transmit the previous frame. This is a 4-bit number: 0 should be interpreted as 1 attempt; 1 as 2 attempts, up until 15 as 16 attempts.
24	Reserved	Returns logic 0.
23	EXCESSIVE_COLLISION	Asserted if a collision has been detected on each of the last 16 attempts to transmit the previous frame.
22	LATE_COLLISION	Asserted if a late collision occurred during frame transmission.
21	EXCESSIVE_DEFERRAL	Asserted if the previous frame was deferred for an excessive amount of time as defined by the constant "maxDeferTime" in *IEEE Std 802.3-2008* [Ref 4].
20	TX_DEFERRED	Asserted if transmission of the frame was deferred.
19	VLAN_FRAME	Asserted if the previous frame contained a VLAN identifier in the length/type field when transmitter VLAN operation is enabled.
18 down to 5	FRAME_LENGTH_COUNT	The length of the previous frame in number of bytes. The count stays at 16368 for any jumbo frames larger than this value.
4	CONTROL_FRAME	Asserted if the previous frame had the special MAC Control Type code 88-08 in the length/type field.
3	UNDERRUN_FRAME	Asserted if the previous frame contained an underrun error.
2	MULTICAST_FRAME	Asserted if the previous frame contained a multicast address in the destination address field.
1	BROADCAST_FRAME	Asserted if the previous frame contained a broadcast address in the destination address field.
0	SUCCESSFUL_FRAME	Asserted if the previous frame was transmitted without error.

Chapter 7

Flow Control

This chapter describes the operation of the flow control logic of the V6EMAC. The flow control block is designed to clause 31 of *IEEE Std 802.3-2008* [Ref 4]. In full duplex mode the MAC can be configured to transmit pause requests and to act on their reception; these modes of operation can be independently enabled or disabled.

Overview of Flow Control

Flow Control Requirement

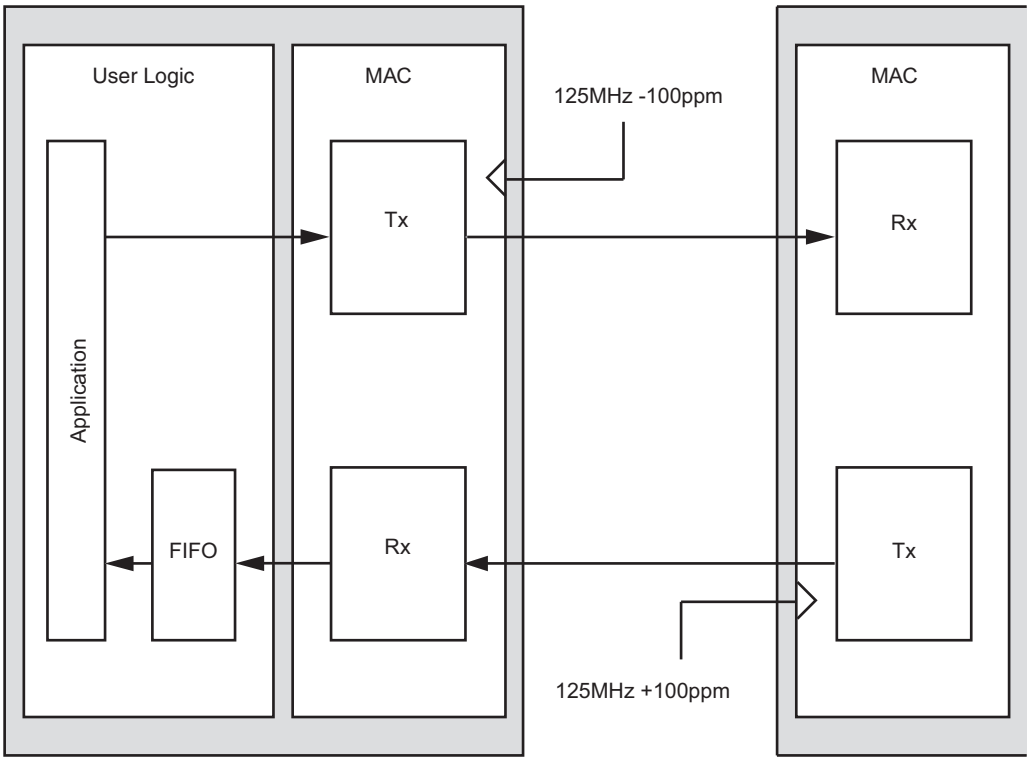

Figure 7-1: **The Requirement for Flow Control**

Figure 7-1 illustrates the requirement for Flow Control at 1 Gb/s. The MAC on the right side of the figure has a reference clock slightly faster than the nominal 125 MHz. The MAC on the left side of the figure has a reference clock slightly slower than the nominal 125 MHz. This results in the MAC on the left side of the figure not being able to match the full line rate of the MAC on the right side (due to clock tolerances). The MAC at the left is

illustrated as performing a loopback implementation, which results in the FIFO filling up over time. Without Flow Control, this FIFO eventually fills and overflows, resulting in the corruption or loss of Ethernet frames. Flow Control is one solution to this issue.

Flow Control Basics

A MAC can transmit a Pause Control frame to request that its link partner cease transmission for a specific period of time. For example, the left MAC in Figure 7-1 might initiate a pause request when its client FIFO (illustrated) reaches a nearly full state.

A MAC should respond to received Pause Control frames by ceasing transmission of frames for the period of time defined in the received pause control frame. For example, the right MAC of Figure 7-1 might cease transmission after receiving the Pause Control frame transmitted by the left MAC. In a well designed system, the right MAC ceases transmission before the client FIFO of the left MAC overflows to provide time to empty the FIFO to a safe level before resuming normal operation. This practice safeguards the system against FIFO overflow conditions and frame loss.

Pause Control Frames

Control frames are a special type of Ethernet frame defined in clause 31 of the *IEEE 802.3* standard. Control frames are identified from other frame types by a defined value placed into the length/type field (the MAC Control Type code). Figure 7-2 illustrates control frame format.

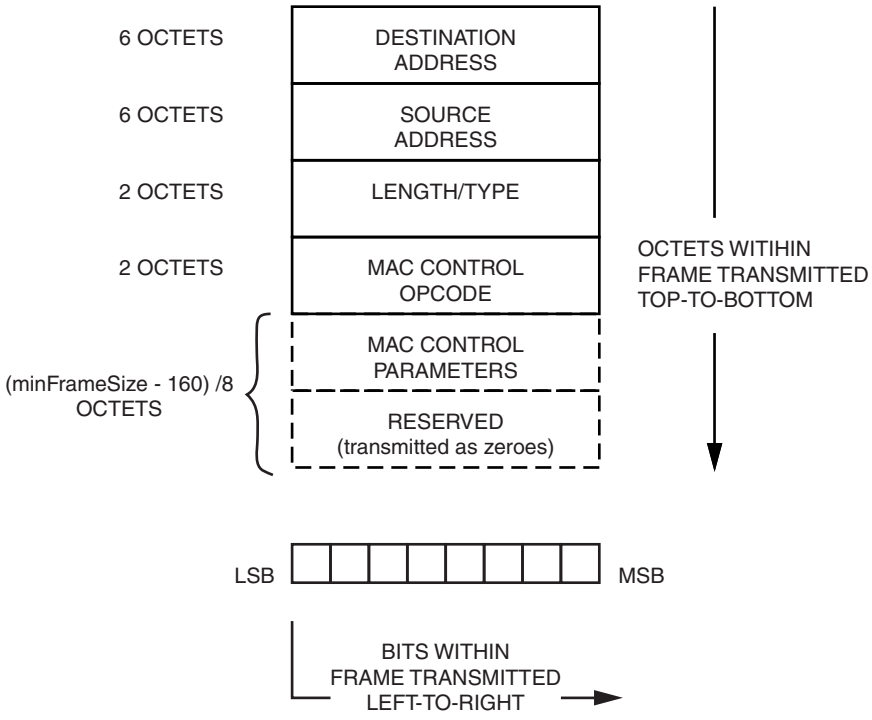

Figure 7-2: **MAC Control Frame Format**

A Pause Control frame is a special type of Control frame, identified by a defined value placed into the MAC Control opcode field.

Note: MAC Control OPCODES other than for Pause (Flow Control) frames have also been defined for Ethernet Passive Optical Networks.

The MAC Control Parameter field of the Pause Control frame contains a 16-bit field which contains a binary value directly relating to the duration of the pause. This defines the number of *pause_quantum* (512 bit times of the particular implementation). At 1 Gb/s, a single *pause_quantum* corresponds to 512 ns. At 100 Mb/s, a single *pause_quantum* corresponds to 5120 ns, and at 10 Mb/s, a single *pause_quantum* corresponds to 51200 ns.

Flow Control Operation of the V6EMAC

Transmitting a Pause Control Frame

Core-Initiated Pause Request

If the core is configured to support transmit flow control, the user can initiate a flow control frame by asserting `pause_req` while the pause value is on the `pause_val` bus. Figure 7-3 illustrates this timing.

Pause request signals are synchronous to either `tx_mac_aclk`, if present, or `gtx_clk`.

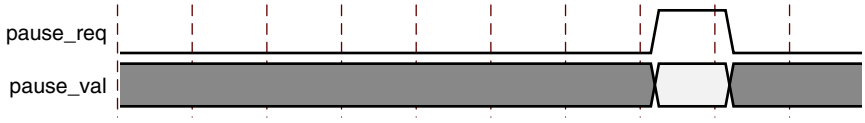

Figure 7-3: **Pause Request Timing**

This action causes the core to construct and transmit a Pause Control frame on the link with the following MAC Control frame parameters (see Figure 7-2):

- The destination address used is an *IEEE 802.3* globally assigned multicast address (which any Flow Control capable MAC responds to).
- The source address used is the configurable Pause Frame MAC Address.
- The value sampled from the `pause_val[15:0]` port at the time of the `pause_req` assertion is encoded into the MAC Control Parameter field to select the duration of the pause (in units of *pause_quantum*).

If the transmitter is currently inactive at the time of the pause request, this Pause Control frame is transmitted immediately. If the transmitter is currently busy, the current frame being transmitted is allowed to complete; the Pause Control frame then follows in preference to any pending user supplied frame.

A Pause Control frame initiated by this method is transmitted even if the transmitter itself has ceased transmission in response to receiving an inbound pause request.

Note: Only a single pause control frame request is stored by the transmitter. If the `pause_req` signal is asserted numerous times in a short time period (before the control pause frame transmission has had a chance to begin), only a single pause control frame is transmitted. The `pause_val[15:0]` value used is the most recent value sampled.

User-Initiated Pause Request

For maximum flexibility, flow control logic can be disabled in the core and alternatively implemented in the user logic connected to the core. Any type of Control frame can be transmitted through the core through the TX AXI4-Stream interface using the same transmission procedure as a standard Ethernet frame (see Transmitting Outbound Frames).

Receiving a Pause Control Frame

Core-Initiated Response to a Pause Request

An error-free Control frame is a received frame matching the format of Figure 7-2. It must pass all standard receiver frame checks (for example, FCS field checking); in addition, the control frame received must be exactly 64-bytes in length (from destination address through to the FCS field inclusive). This is minimum legal Ethernet MAC frame size and the defined size for control frames.

Any Control frame received that does not conform to these checks contains an error, and it is passed to the RX AXI4-Stream as an errored packet (`rx_axis_mac_tuser` asserted).

Pause Frame Reception Disabled

When pause control reception is disabled, an error-free control frame is received through the RX AXI4-Stream interface. In this way, the frame is passed to the user logic for interpretation (see User-Initiated Response to a Pause Request).

Pause Frame Reception Enabled

When pause control reception is enabled and an error-free frame is received by the MAC core, the following frame decoding functions are performed:

1. The destination address field is matched against the *IEEE 802.3* globally assigned control multicast address (01-80-C2-00-00-01) or the configurable Pause Frame MAC Address.
2. The length/type field is matched against the MAC Control Type code.
3. If the second match is TRUE, the OPCODE field contents are matched against the Ethernet MAC control OPCODE for pause frames.

If all the previously listed checks are TRUE, and the frame is of minimum legal size OR larger and control frame length checking is disabled, the 16-bit binary value in the MAC control parameters field of the control frame is then used to inhibit transmitter operation for the required number of *pause_quantum*. This inhibit is implemented by delaying the assertion of `tx_axis_mac_tready` at the TX AXI4-Stream interface until the requested pause duration has expired. Because the received pause frame has been acted upon, it is passed to the RX AXI4-Stream interface as an errored packet to indicate to the user that it can now be dropped.

If the second match is TRUE and the frame is not exactly 64 bytes in length (when control frame length checking is enabled), the reception of any frame is considered to be an invalid control frame. This frame is ignored by the flow control logic and passed to the RX AXI4-Stream interface as frame error. In this case, the frame is in error even if flow control is not enabled.

If any of the previously listed checks are FALSE, the frame is ignored by the Flow Control logic and passed up to the user logic for interpretation by marking it as a good frame. It is then the responsibility of the MAC user logic to decode, act on (if required) and drop this control frame.

Note: Any frame in which the length/type field contains the MAC Control Type in the length/type field should be dropped by the receiver user logic. All Control frames are indicated by `rx_statistics_vector` bit 19 (see Receiver Statistics Vector).

User-Initiated Response to a Pause Request

For maximum flexibility, flow control logic can be disabled in the core and alternatively implemented in the user logic connected to the core. Any type of error-free Control frame is then passed through the core without error. In this way, the frame is passed to the user for interpretation. It is then the responsibility of the user to drop this control frame and to act on it by ceasing transmission through the core, if applicable.

Flow Control Implementation Example

This explanation is intended to describe a simple (but crude) example of a Flow Control implementation to introduce the concept.

Consider the system illustrated in Figure 7-1. To summarize the example, the MAC on the left-hand side of the figure cannot match the full line rate of the right-hand MAC due to clock tolerances. Over time, the FIFO illustrated fills and overflows. The aim is to implement a Flow Control method which, over a long time period, reduces the full line rate of the right-hand MAC to average that of the lesser full line rate capability of the left-hand MAC.

Method

1. Choose a FIFO nearly full to occupancy threshold (7/8 occupancy is used in this description). When the occupancy of the FIFO exceeds this occupancy, initiate a single pause control frame with 0xFFFF used as the *pause_quantum* duration (0xFFFF is placed on `pause_val[15:0]`). This is the maximum pause duration. This causes the right-hand MAC to cease transmission and the FIFO of the left-hand MAC starts to empty.
2. Choose a second FIFO occupancy threshold (3/4 is used in this description). When the occupancy of the FIFO falls below this occupancy, initiate a second pause control frame with 0x0000 used as the *pause_quantum* duration (0x0000 is placed on `pause_val[15:0]`). This indicates a zero pause duration, and upon receiving this pause control frame, the right-hand MAC immediately resumes transmission (it does not wait for the original requested pause duration to expire). This pause control frame can therefore be considered a “pause cancel” command.

Operation

Figure 7-4 illustrates the FIFO occupancy over time.

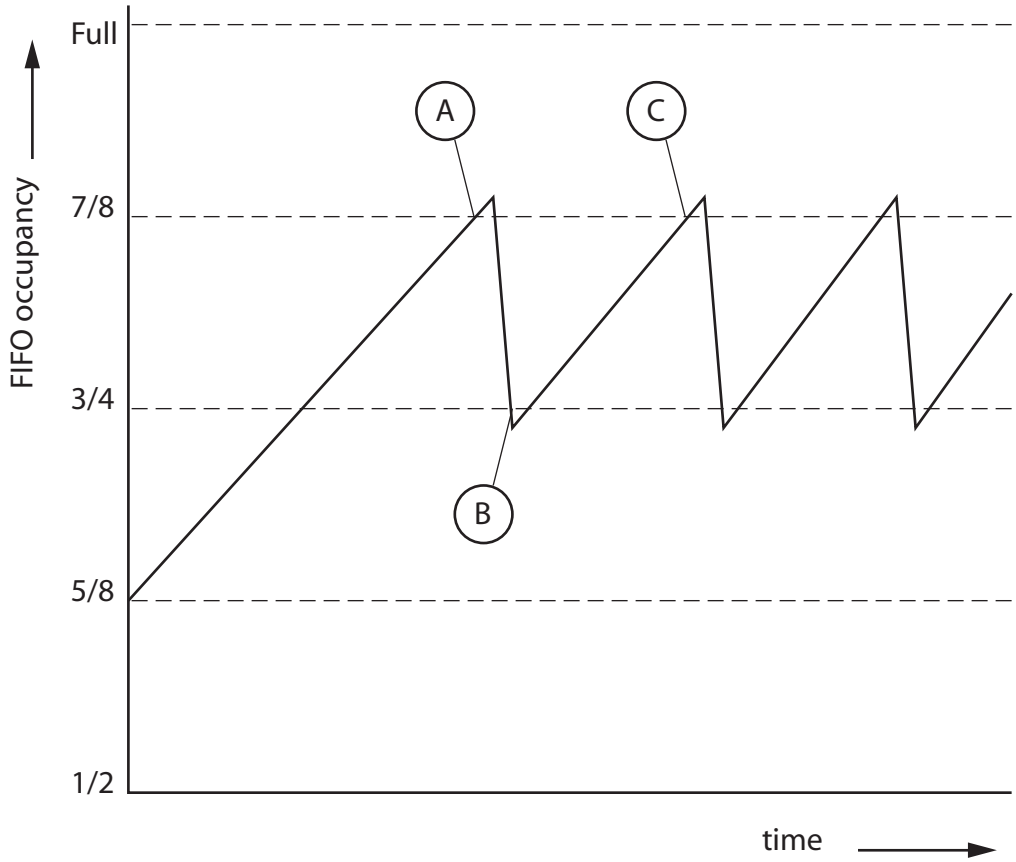

Figure 7-4: **Flow Control Implementation Triggered from FIFO Occupancy**

The following text describes the sequence of flow control operation in this given example.

1. The average FIFO occupancy of the left-hand MAC gradually increases over time due to the clock tolerances. At point A, the occupancy has reached the threshold of 7/8 occupancy. This triggers the maximum duration pause control frame request.
2. Upon receiving the pause control frame, the right-hand MAC ceases transmission.
3. After the right-hand MAC ceases transmission, the occupancy of the FIFO attached to the left-hand MAC rapidly empties. The occupancy falls to the second threshold of 3/4 occupancy at point B. This triggers the zero duration pause control frame request (the pause cancel command).
4. Upon receiving this second pause control frame, the right-hand MAC resumes transmission.
5. Normal operation resumes and the FIFO occupancy again gradually increases over time. At point C, this cycle of Flow Control repeats.

XILINX®

Chapter 8

Configuration and Status

This chapter provides general guidelines for configuring and monitoring the core, including a detailed description of the user-side management interface and registers present in the core.

The Management Interface

The management interface uses the industry standard AXI4-Lite to allow access to the MAC netlist. This interface is used for the following:

- Configuring the MAC core
- Configuring the Address/Frame Filter
- Configuring the Interrupts
- Accessing Statistics information
- Providing access to the MDIO interface

Table 8-1 describes the optional signals used by the user to access the MAC netlist.

Table 8-1: **Optional AXI4-Lite Signal Pinout**

Signal	Direction	Clock Domain	Description
s_axi_aclk	Input	N/A	Clock for AXI4-Lite
s_axi_resetn	Input	s_axi_aclk	Local reset for the clock domain
s_axi_awaddr[31:0]	Input	s_axi_aclk	Write Address
s_axi_awvalid	Input	s_axi_aclk	Write Address Valid
s_axi_awready	Output	s_axi_aclk	Write Address ready
s_axi_wdata[31:0]	Input	s_axi_aclk	Write Data
s_axi_wvalid	Input	s_axi_aclk	Write Data valid
s_axi_wready	Output	s_axi_aclk	Write Data ready
s_axi_bresp[1:0]	Output	s_axi_aclk	Write Response
s_axi_bvalid	Output	s_axi_aclk	Write Response valid
s_axi_bready	Input	s_axi_aclk	Write Response ready
s_axi_araddr[31:0]	Input	s_axi_aclk	Read Address
s_axi_arvalid	Input	s_axi_aclk	Read Address valid
s_axi_arready	Output	s_axi_aclk	Read Address ready

Table 8-1: **Optional AXI4-Lite Signal Pinout *(Cont'd)***

Signal	Direction	Clock Domain	Description
s_axi_rdata[31:0]	Output	s_axi_aclk	Read Data
s_axi_rresp[1:0]	Output	s_axi_aclk	Read Response
s_axi_rvalid	Output	s_axi_aclk	Read Data/Response Valid
s_axi_rready	Input	s_axi_aclk	Read Data/Response ready

Note: Only 11 out of the 32 address bits are used for decoding the read/write address

Figure 8-1 and Figure 8-2 show the basic AXI4-Lite transactions supported by the V6EMAC. Illegal accesses results in an error indication. See the *ARM® AMBA® AXI Protocol Specification v2.0* [Ref 7] for more information about this standard.

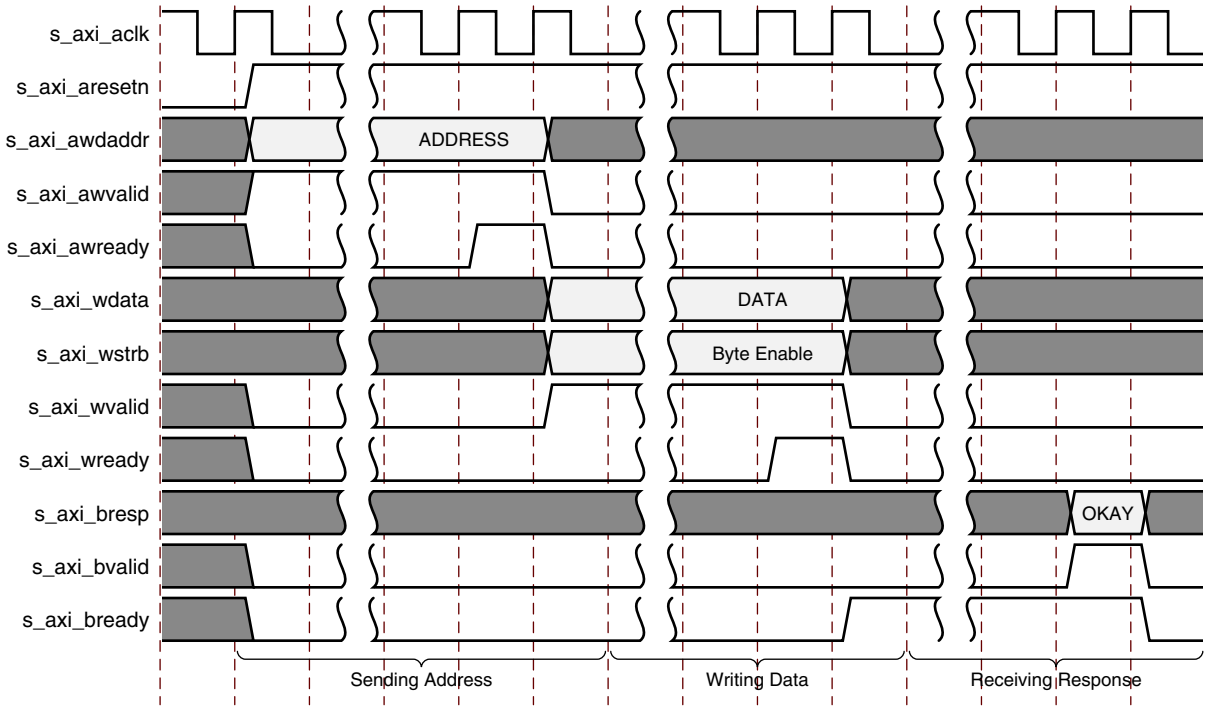

Figure 8-1: **Management Register Write Timing**

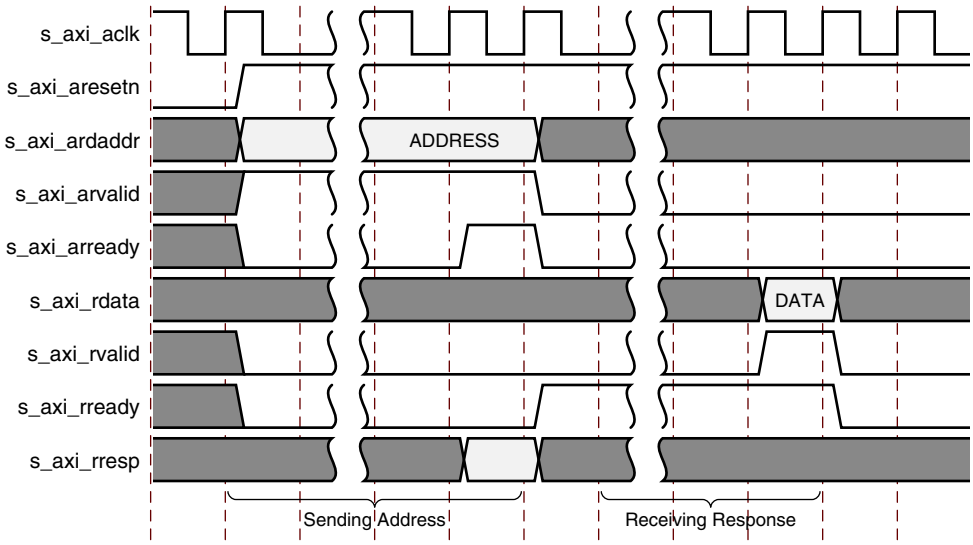

Figure 8-2: **Management Register Read Timing**

Address Map

The MAC address map, as shown in Table 8-2, has been updated to fully memory map all registers and provide the same interface in all cases. Where possible, gaps have been left to allow for future expansion. The Address Map has been split into 5 distinct functional blocks: Statistics, MAC Configuration, Interrupt Controller, Frame Filter and MDIO. Each of these register blocks and the individual registers are described in more detail in the following sections.

Table 8-2: **MAC Registers**

Address	Description
0x000-0x1FC	Reserved
0x200-0x3FC	Statistics Counters
0x400-0x4FC	MAC Configuration
0x500-0x5FC	MDIO Interface
0x600-0x6FC	Interrupt Controller
0x700-0x7FC	Address/Frame Filter

Statistics Counters

The Statistics counters, which are only available when the management interface is enabled, can be defined to be either 32 or 64-bits wide, with 32 bits being the default. When defined as 64-bits wide the counter values are captured across two registers. In all cases a read of the lower 32-bit value causes the upper 32 bits to be sampled. A subsequent read of the upper 32-bit location returns this sampled value. If an upper location for a different counter is read the access returns an error condition to indicate that the returned data value is incorrect.

Note: All statistics counters are read only. A write to any location is ignored and returns an error.

The Statistics counters can optionally be reset upon a global reset, with this being enabled using the GUI. They do not reset upon a read and wraparound when the maximum count value is reached. It is the responsibility of the user to ensure the counters are read frequently enough to guarantee a wraparound is not missed.

As the V6EMAC only targets families which have a LUT6 based architecture, the statistics counters are based around distributed memory. The V6EMAC always output RX and TX statistics vectors and, when the statistics counters are present, these are decoded in the Vector Decode block, provided in the Block level, which, in turn, provides the increment vector bus; this is described in more detail in the following section.

Increment Interface Overview

The Increment Interface has two main logical sections:

- A low-frequency increment component controlled by the increment vector input. This accommodates the majority of the statistical counters, which only increment at (or less frequently than) a standard minimum Ethernet frame period. These are decoded in the Vector Decode block and therefore can be edited by the user if desired though this is not recommended for the pre-defined counters.
- A high-frequency increment component. These are generated internally to the netlist and as such cannot be edited or monitored by the user. These are used to accommodate those counters which can increment on every cycle and these are captured in counters 0-3.

Low-Frequency Statistical Counters

The `increment_vector`[4:43] is an input bus signal which provides the predefined counters as described in Table 8-3 and 3 user defined counters. This accommodates the vast majority of the statistical counters which only increment at (or less frequently than) a minimum Ethernet frame period.

Figure 8-3 illustrates the `increment_vector` bus provided by the vector decode block. There is an increment bit for each counter from counter 4 upwards. A toggle on a particular increment bit causes the corresponding counter to increment. The mapping of the increment vector bits to the various register mapped counter is shown in Table 8-3.

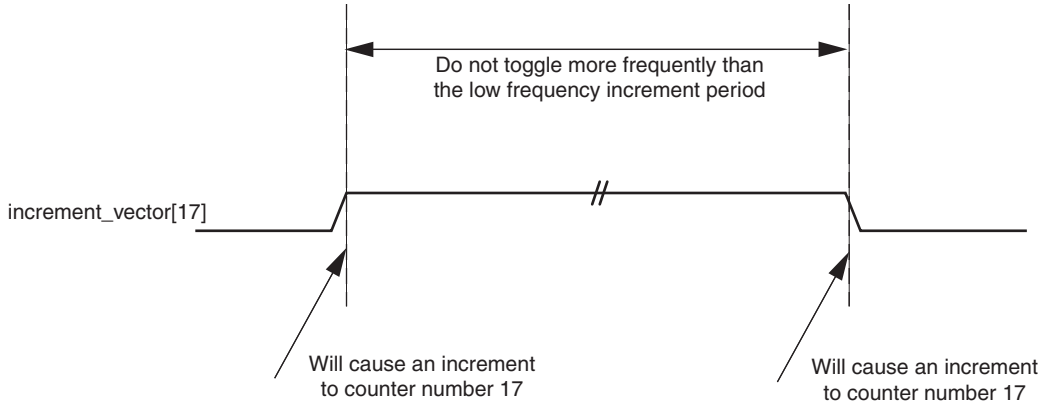

Figure 8-3: **Increment Vector Timing Diagram**

The `increment_vector` is input to the core and edge detection circuitry (toggle detection) is placed on each bit. The toggle detection circuitry is synchronous to `stats_ref_clk`.

Within the core, the current counter values are stored in distributed memory. The Statistics core accesses each of the counters within this memory in a round robin fashion and if an increment has been requested since the last access the relative value is incremented prior to being written back to memory.

Bandwidth Requirements

The frequency of `stats_ref_clk` is flexible but depends upon both the number of counters and the maximum frequency supported by the V6EMAC. The low-frequency increment vectors can update at a maximum rate of once per minimum sized Ethernet frame. This translates to 584ns when running at 1Gb/s (64 bytes of minimum Ethernet frame size, plus 1 byte of minimum received preamble, plus 8 bytes of minimum received interframe gap, at a byte rate of 1 byte per 8 ns).

With `stats_ref_clk` set to 125 MHz, 36 statistical counters can be safely updated between successive Ethernet frames (584 ns divided by the 8 ns clock period of `stats_ref_clk`, divided by 2 because a counter update requires two accesses). As this is less than the provided 44 counters extra decode logic is included to take advantage of the frame size counters one-hot status (that is, only one of the seven RX and one of the seven TX frame size counters can update on a per packet basis. The round robin function which controls which counter is being accessed only accesses the required frame size counter and skips the other 5.) This means that the 44 counters supported only actually require 32 counter accesses. However, this does mean that the `stats_ref_clk` should be at least as fast as the clock used for the maximum rate supported by the V6EMAC (125 MHz at 1 Gb/s or 12.5 MHz at 10/100 Mb/s). When the core is used with the 16-bit client interface (in 2 Gb/s or 2.5 Gb/s overclocked modes), the `stats_ref_clk` frequency must use the 2x clock rate. For 2.5 Gb/s this is 312.5 MHz, and for 2 Gb/s it is 250 MHz.

Table 8-3: **Statistics Counter Definitions**

Name	Increment Bit No.	Address	Description
Received bytes	NA	0x200-0x204	A count of bytes of frames received (destination address to frame check sequence inclusive).
Transmitted bytes	NA	0x208-0x20C	A count of bytes of frames transmitted (destination address to frame check sequence inclusive).
RX Undersize frames	NA	0x210-0x214	A count of the number of frames received that were fewer than 64 bytes in length but otherwise well formed.
RX Fragment frames	NA	0x218-0x21C	A count of the number of frames received that were fewer than 64 bytes in length and had a bad frame check sequence field.
RX 64 byte Frames	4	0x220-0x224	A count of error-free frames received 64 bytes in length.
RX 65-127 byte Frames	5	0x228-0x22C	A count of error-free frames received between 65 and 127 bytes in length.
RX 128-255 byte Frames	6	0x230-0x234	A count of error-free frames received between 128 and 255 bytes in length.
RX 256-511 byte Frames	7	0x238-0x23C	A count of error-free frames received between 256 and 511 bytes in length.
RX 512-1023 byte Frames	8	0x240-0x244	A count of error-free frames received between 512 and 1023 bytes in length.

Table 8-3: **Statistics Counter Definitions *(Cont'd)***

Name	Increment Bit No.	Address	Description
RX 1024-MaxFrameSize byte Frames	9	0x248-0x24C	A count of error-free frames received between 1024 bytes and the specified *IEEE Std 802.3-2008* maximum legal length.
RX Oversize Frames	10	0x250-0x254	A count of otherwise error-free frames received that exceeded the maximum legal frame length specified in *IEEE Std 802.3-2008* [Ref 4].
TX 64 byte Frames	11	0x258-0x25C	A count of error-free frames transmitted that were 64 bytes in length.
TX 65-127 byte Frames	12	0x260-0x264	A count of error-free frames transmitted between 65 and 127 bytes in length.
TX 128-255 byte Frames	13	0x268-0x26C	A count of error-free frames transmitted between 128 and 255 bytes in length.
TX 256-511 byte Frames	14	0x270-0x274	A count of error-free frames transmitted between 256 and 511 bytes in length.
TX 512-1023 byte Frames	15	0x278-0x27C	A count of error-free frames transmitted that were between 512 and 1023 bytes in length.
TX 1024-MaxFrameSize byte Frames	16	0x280-0x284	A count of error-free frames transmitted between 1024 and the specified *IEEE Std 802.3-2008* maximum legal length.
TX Oversize Frames	17	0x288-0x28C	A count of otherwise error-free frames transmitted that exceeded the maximum legal frame length specified in *IEEE Std 802.3-2008* [Ref 4].
RX Good Frames	18	0x290-0x294	A count of error-free frames received.
RX Frame Check Sequence Errors	19	0x298-0x29C	A count of received frames that failed the CRC check and were at least 64 bytes in length.
RX Good Broadcast Frames	20	0x2A0-0x2A4	A count of frames successfully received and directed to the broadcast group address.
RX Good Multicast Frames	21	0x2A8-0x2AC	A count of frames successfully received and directed to a non-broadcast group address.
RX Good Control Frames	22	0x2B0-0x2B4	A count of error-free frames received that contained the special control frame identifier in the length/type field.
RX Length/Type Out of Range	23	0x2B8-0x2BC	A count of frames received that were at least 64 bytes in length where the length/type field contained a length value that did not match the number of MAC client data bytes received. The counter also increments for frames in which the length/type field indicated that the frame contained padding but where the number of MAC client data bytes received was greater than 64 bytes (minimum frame size). The exception is when the Length/Type Error Checks are disabled in the chosen MAC, in which case this counter does not increment.
RX Good VLAN Tagged Frames	24	0x2C0-0x2C4	A count of error-free VLAN frames received. This counter only increments when the receiver is configured for VLAN operation.

Table 8-3: **Statistics Counter Definitions *(Cont'd)***

Name	Increment Bit No.	Address	Description
RX Good Pause Frames	25	0x2C8-0x2CC	A count of error-free frames received that contained the MAC Control type identifier 88-08 in the length/type field, contained a destination address that matched either the MAC Control multicast address or the configured source address of the MAC, contained the PAUSE opcode and were acted upon by the MAC.
RX Bad Opcode	26	0x2D0-0x2D4	A count of error-free frames received that contained the MAC Control type identifier 88-08 in the Length/Type field but were received with an opcode other than the PAUSE opcode.
TX Good Frames	27	0x2D8-0x2DC	A count of error-free frames transmitted.
TX Good Broadcast Frames	28	0x2E0-0x2E4	A count of error-free frames that were transmitted to the broadcast address.
TX Good Multicast Frames	29	0x2E8-0x2EC	A count of error-free frames that were transmitted to a group destination address other than broadcast.
TX Good Underrun Errors	30	0x2F0-0x2F4	A count of frames that would otherwise be transmitted by the core but could not be completed due to the assertion of TX_UNDERRUN during the frame transmission.
TX Good Control Frames	31	0x2F8-0x2FC	A count of error-free frames transmitted that contained the MAC Control Frame type identifier 88-08 in the length/type field.
TX Good VLAN Tagged Frames	32	0x300-0x304	A count of error-free VLAN frames transmitted. This counter only increments when the transmitter is configured for VLAN operation.
TX Good Pause Frames	33	0x308-0x30C	A count of error-free PAUSE frames generated and transmitted by the MAC in response to an assertion of pause_req.
TX Single Collision Frames	34	0x310-0x314	A count of frames involved in a single collision but subsequently transmitted successfully (half-duplex mode only).
TX Multiple Collision Frames	35	0x318-0x31C	A count of frames involved in more than one collision but subsequently transmitted successfully (half-duplex mode only).
TX Deferred	36	0x320-0x324	A count of frames whose transmission was delayed on its first attempt because the medium was busy (half-duplex mode only).
TX Late Collisions	37	0x328-0x32C	A count of the times that a collision has been detected later than one slot Time from the start of the packet transmission. A late collision is counted twice — both as a collision and as a late Collision (half-duplex mode only).
TX Excess collisions	38	0x330-0x334	A count of the frames that, due to excessive collisions, are not transmitted successfully (half-duplex mode only).

Table 8-3: **Statistics Counter Definitions *(Cont'd)***

Name	Increment Bit No.	Address	Description
TX Excess Deferral	39	0x338-0x33C	A count of frames that deferred transmission for an excessive period of time (half-duplex mode only).
TX Alignment Errors	40	0x340-0x344	Asserted for received frames of size 64-bytes and greater which contained an odd number of received nibbles and which also contained an invalid FCS field.

MAC Configuration

After the core is powered up and reset, the user can reconfigure some of the core parameters from their defaults, such as flow control support and RX/TX VLAN support. Configuration changes can be written at any time, however, both receiver and transmitter configuration changes only take effect during interframe gaps. The exceptions to this are the configurable soft resets, which take effect immediately.

Configuration of the MAC core is performed through a register bank accessed through the management interface. The configuration registers available in the core are detailed in Table 8-4.

Table 8-4: **Configuration Registers**

Address (Hex)	Description
0x400	Receiver Configuration Word 0
0x404	Receiver Configuration Word 1
0x408	Transmitter Configuration
0x40C	Flow Control Configuration
0x410	Ethernet MAC Mode configuration
0x414-0x41C	Reserved
0x420	RGMII/SGMII Configuration
0x424-0x4F4	Reserved
0x4F8	ID Register
0x4FC	Ability Register

The contents of each configuration register are shown in Table 8-5 through Table 8-12.

Table 8-5: **Receiver Configuration Word 0 (0x400)**

Bit	Default Value	Type	Description
31-0	All 0s	RW	**Pause frame MAC Source Address[31:0]:** This address is used by the MAC to match against the destination address of any incoming flow control frames. It is also used by the flow control block as the source address (SA) for any outbound flow control frames. The address is ordered so the first byte transmitted/received is the lowest positioned byte in the register; for example, a MAC address of AA-BB-CC-DD-EE-FF would be stored in Address[47:0] as 0xFFEEDDCCBBAA.

Table 8-6: **Receiver Configuration Word 1 (0x404)**

Bit	Default Value	Type	Description
15-0	All 0s	RW	**Pause frame MAC Source Address[47:32]:** See description in Table 8-5.
23-16	N/A	RO	Reserved
24	0	RW	**Control Frame Length Check Disable:** When this bit is set to 1 the core does not mark control frames as 'bad' if they are greater than the minimum frame length.
25	0	RW	**Length/Type Error Check Disable:** When this bit is set to 1 the core does not perform the length/type field error checks as described in Length/Type Field Error Checks. When this bit is set to 0 the length/type field checks is performed: this is normal operation.
26	0	RW	**Half Duplex:** (Applicable in 10/100 Mb/s only, for certain physical interfaces) If 1 the receiver operates in half- duplex mode. If 0 the receiver operates in full- duplex mode.
27	0	RW	**VLAN Enable:** When this bit is set to 1 VLAN tagged frames are accepted by the receiver.
28	1	RW	**Receiver Enable:** If set to 1 the receiver block is operational. If set to 0 the block ignores activity on the physical interface RX port.
29	0	RW	**In-band FCS Enable:** When this bit is 1 the MAC receiver passes the FCS field up to the client as described in User-Supplied FCS Passing. When it is 0 the client is not passed to the FCS. In both cases, the FCS is verified on the frame.
30	0	RW	**Jumbo Frame Enable:** When this bit is set to 1 the MAC receiver accepts frames over the specified *IEEE Std 802.3-2008* maximum legal length. When this bit is 0 the MAC only accepts frames up to the specified maximum.
31	0	RW	**Reset:** When this bit is set to 1 the receiver is reset. The bit then automatically reverts to 0. This reset also sets all of the receiver configuration registers to their default values.

Table 8-7: **Transmitter Configuration Word (0x408)**

Bit	Default Value	Type	Description
24-0	N/A	RO	Reserved
25	0	RW	**Interframe Gap Adjust Enable:** If 1 the transmitter reads the value on the port tx_ifg_delay at the start of frame transmission and adjusts the interframe gap following the frame accordingly (see Interframe Gap Adjustment: Full-Duplex Mode Only). If 0 the transmitter outputs a minimum interframe gap of at least twelve clock cycles, as specified in *IEEE Std 802.3-2008* [Ref 4].
26	0	RW	**Half Duplex:** (Applicable in 10/100 Mb/s only, for certain physical interfaces) If 1 the transmitter operates in half-duplex mode.
27	0	RW	**VLAN Enable:** When this bit is set to 1 the transmitter recognizes the transmission of VLAN tagged frames.
28	1	RW	**Transmit Enable:** When this bit is 1 the transmitter is operational. When it is 0 the transmitter is disabled.
29	0	RW	**In-band FCS Enable:** When this bit is 1 the MAC transmitter expects the FCS field to be passed in by the client as described in the User-Supplied FCS Passing. When this bit is 0 the MAC transmitter appends padding as required, computes the FCS and appends it to the frame.
30	0	RW	**Jumbo Frame Enable:** When this bit is set to 1 the MAC transmitter sends frames that are greater than the specified *IEEE Std 802.3-2008* maximum legal length. When this bit is 0 the MAC only sends frames up to the specified maximum.
31	0	RW	**Reset:** When this bit is set to 1 the transmitter is reset. The bit then automatically reverts to 0. This reset also sets all of the transmitter configuration registers to their default values.

Table 8-8: **Flow Control Configuration Word (0x40C)**

Bit	Default Value	Type	Description
28-0	N/A	RO	Reserved
29	1	RW	**Flow Control Enable (RX):** When this bit is 1 received flow control frames inhibits the transmitter operation as described in the Receiving a Pause Control Frame. When this bit is 0 received flow control frames are always passed up to the user.
30	1	RW	**Flow Control Enable (TX):** When this bit is 1 asserting the pause_req signal sends a flow control frame out from the transmitter as described in the Transmitting a Pause Control Frame. When this bit is 0, asserting the pause_req signal has no effect.
31	N/A	RO	Reserved

Table 8-9: **Ethernet MAC Mode Configuration Word (0x410)**

Bits	Default Value	Type	Description
8-0	EMAC_LINKTIMERVAL[8:0]	RW	Sets the programmable link timer value, for operation with 1000BASE-X or SGMII modes
23-9	0	RO	Reserved
24	EMAC_RX16BITCLIENT_ENABLE	RO	**RX 16 Bit AXI4-Stream Enable**: When this bit is 1, the AXI4-Stream receive data interface is 16 bits wide. When this bit is 0, the receive data interface is 8 bits wide. This bit is valid only when using 1000BASE-X PCS/PMA mode.
25	EMAC_TX16BITCLIENT_ENABLE	RO	**TX 16 Bit AXI4-Stream Enable**: When this bit is 1, the AXI4-Stream transmit data interface is 16 bits wide. When this bit is 0, the transmit data interface is 8 bits wide. This bit is valid only when using 1000BASE-X PCS/PMA mode.
26	EMAC_HOST_ENABLE	RO	**MAC Management Enable**: When this bit is 1, the management interface is enabled. When this bit is 0, the management interface is disabled.
27	EMAC_1000BASEX_ENABLE	RW	**1000BASE-X Enable**: When this bit is 1, the V6EMAC is configured in 1000BASE-X mode
28	EMAC_SGMII_ENABLE	RW	**SGMII Enable**: When this bit is 1, the V6EMAC is configured in SGMII mode
29	EMAC_RGMII_ENABLE	RO	**RGMII Enable**: When this bit is 1, the V6EMAC is configured in RGMII mode.
31-30	{EMAC_SPEED_MSB, EMAC_SPEED_LSB}	RW	MAC Speed Configuration 00 - 10 Mb/s 01 - 100 Mb/s 10 - 1 Gb/s 11 - N/A

Notes:

1. The setting of the Ethernet MAC Mode Configuration register is not affected by a reset.

Table 8-10: **RGMII/SGMII Configuration Word (0x420)**

Bits	Default Value	Type	Description
0	0	RO	**RGMII Link**: Valid in RGMII mode configuration only. When this bit is 1, the link is up. When this bit is 0, the link is down. This displays the link information from the PHY to the Ethernet MAC, encoded by GMII_RX_DV and GMII_RX_ER during the IFG.
1	0	RO	**RGMII Duplex Status**: Valid in RGMII mode configuration only. This bit is 0 for half-duplex and 1 for full-duplex. This displays the duplex information from the PHY to the V6EMAC, encoded by GMII_RX_DV and GMII_RX_ER during the IFG.
3-2	0	RO	**RGMII Speed**: Valid in RGMII mode configuration only. Link information from the PHY to the V6EMAC as encoded by GMII_RX_DV and GMII_RX_ER during the IFG. This two-bit vector is defined with the following values: • 10 = 1000 Mb/s • 01 = 100 Mb/s • 00 = 10 Mb/s • 11 = N/A
29-4	N/A	RO	Reserved
31-30	0	RO	**SGMII_Speed**: Valid in SGMII mode configuration only. This displays the SGMII speed information, as received by TX_CONFIG_REG[11:10] in the PCS/PMA register. This two-bit vector is defined with the following values: • 10 = 1000 Mb/s • 01 = 100 Mb/s • 00 = 10 Mb/s • 11 = N/A

Table 8-11: **ID Register (0x4F8)**

Bits	Default Value	Type	Description
7-0	0	RO	Patch Level (0- nopatch, 1-rev1 etc)
15-8	N/A	RO	Reserved
23-16	3	RO	Minor Rev
31-24	2	RO	Major Rev

Table 8-12: **Ability Register (0x4FC)**

Bits	Default Value	Type	Description
0	1(1)	RO	**10M Ability**: If set, the core is 10M capable
1	1(1)	RO	**100M Ability**: If set, the core is 100M capable
2	1(1)	RO	**1G Ability**: If set, the core is 1G capable
3-7	N/A	RO	Reserved
8	1(1)	RO	Statistics Counters available
9	1	RO	Half duplex capable
10	1(1)	RO	Frame Filter available
11-31	N/A	RO	Reserved

Notes:

1. Depends upon core abilities selected

MDIO Interface

The Management Interface is also used to access the MDIO Interface of the MAC core; this interface is used to access the Managed Information Block of the PHY components attached to the MAC core and is only available when the management interface is enabled.

Introduction to MDIO

MDIO Bus System

The Management Data Input/Output (MDIO) interface for access to Ethernet PHY devices for 1 Gb/s operation and slower speeds is defined in *IEEE 802.3*, clause 22. This two-wire interface consists of a clock (MDC) and a shared serial data line (MDIO). The maximum permitted frequency of MDC is set at 2.5 MHz.

Figure 8-4 illustrates an example MDIO bus system.

A V6EMAC is shown as the MDIO bus master (the Station Management (STA) entity).

Two PHY devices are shown connected to the same bus, both of which are MDIO slaves (MDIO Managed Device (MMD) entities).

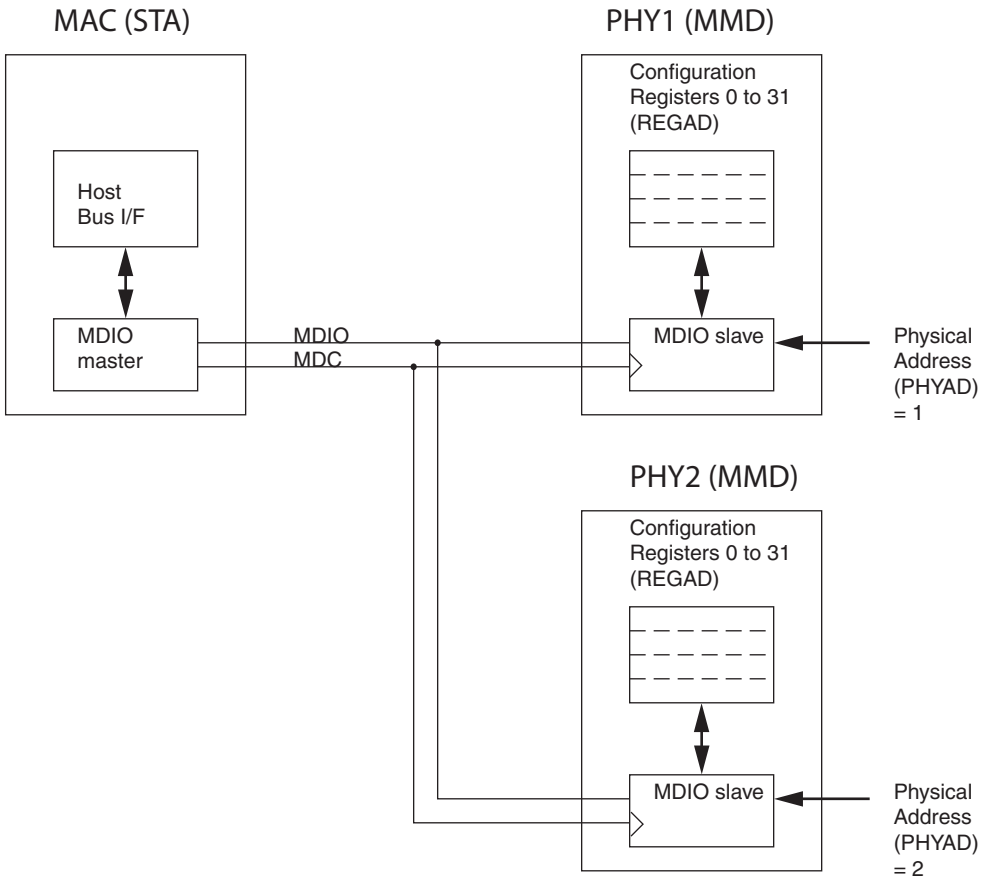

Figure 8-4: **A Typical MDIO-Managed System**

The MDIO bus system is a standardized interface for accessing the configuration and status registers of Ethernet PHY devices. In Figure 8-4, the Management Bus I/F of the V6EMAC is able to access the configuration and status registers of two PHY devices through the MDIO bus.

MDIO Transactions

All transactions, read or write, are initiated by the MDIO master. All MDIO slave devices, when addressed, must respond. MDIO transactions take the form of an MDIO frame, containing fields for transaction type, address and data. This MDIO frame is transferred across the MDIO wire synchronously to MDC. The abbreviations are used in this section are explained in Table 8-13.

Table 8-13: **Abbreviations and Terms**

Abbreviation	Term
PRE	Preamble
ST	Start of frame
OP	Operation code
PHYAD	Physical address
REGAD	Register address
TA	Turnaround

Write Transaction

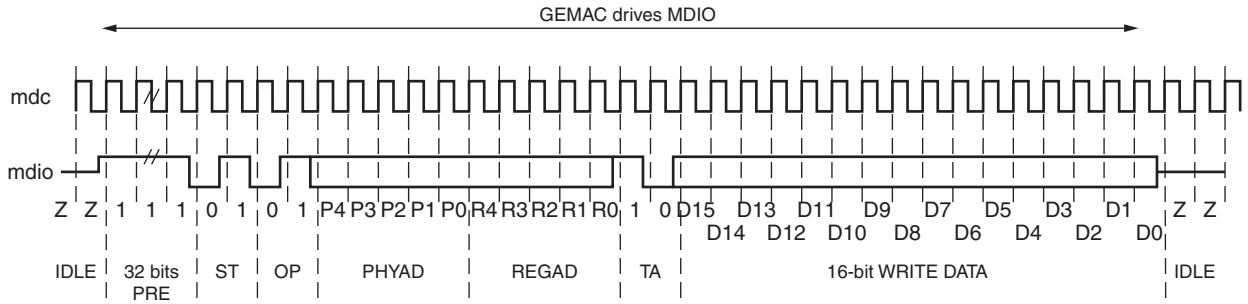

Figure 8-5: **MDIO Write Transaction**

Figure 8-5 shows a Write transaction across the MDIO; this is defined by OP=01. The addressed PHY (PHYAD) device takes the 16-bit word in the data field and writes it to the register at REGAD.

Read Transaction

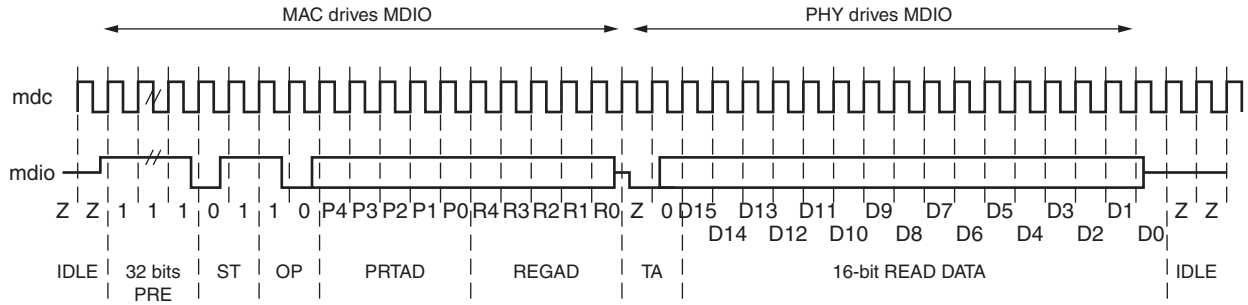

Figure 8-6: **MDIO Read Transaction**

Figure 8-6 shows a Read transaction; this is defined by OP=10. The addressed PHY (PHYAD) device returns the 16-bit word from the register at REGAD. For a fuller description of the operation of the MDIO Interface itself, see *IEEE Std 802.3-2008* [Ref 4].

MDIO Configuration and Control

Access to the MDIO interface through the management interface is entirely register mapped.

To perform an MDIO write the write data must first be written to the MDIO Write Data register, shown in Table 8-17, The MDIO write transaction is then initiated by a write to the MDIO Configuration word 1, shown in Table 8-16, with Initiate (bit 11) set to 0x1, OP (bits 15:14) set to 0x1 and the PHYAD and REGAD set according to the PHY and Register being accessed, see MDIO Addressing. This triggers the MDIO Ready bit to deassert and it remains deasserted until the MDIO transaction has completed.

To perform an MDIO read, the read transaction is initiated by a write to the MDIO Configuration Word 1, shown in Table 8-16, with Initiate (bit 11) set to 0x1, OP (bits 15:14) set to 0x2 and the PHYAD and REGAD set according to the PHY and Register being accessed, see MDIO Addressing. This triggers the MDIO Ready bit to deassert and it remains deasserted until the MDIO transaction has completed. When the MDIO Ready is re-asserted the read data is ready to be read from the MDIO Read data register, shown in Table 8-18.

Note: It is possible to either poll the MDIO Configuration register or the MDIO read data register to check the status of MDIO Ready. Alternatively, the MAC interrupt can be used (see Interrupt Controller).

A list of the MDIO registers is shown in Table 8-14.

Table 8-14: **MDIO Configuration Registers**

Address (Hex)	Description
0x500	MDIO Configuration Word 0
0x504	MDIO Configuration Word 1
0x508	MDIO TX Data
0x50C	MDIO RX Data

The contents of each configuration register are shown in Table 8-15 through Table 8-18.

Table 8-15: **MDIO Configuration word 0 (0x500)**

Bits	Default Value	Type	Description
5-0	0x0	RW	**Clock Divide[5:0]:** See Accessing PHY Configuration Registers, through the MDIO using the Management Interface
6	0x0	RW	**MDIO Enable:** When this bit is 1 the MDIO interface can be used to access attached PHY devices. When this bit is 0 the MDIO interface is disabled and the MDIO signals remain inactive. A write to this bit only takes effect if Clock Divide is set to a non-zero value.
31-7	N/A	RO	Reserved

Table 8-16: **MDIO Configuration Word 1 (0x504)**

Bits	Default Value	Type	Description
6-0	N/A	RO	Reserved
7	0x0	RO	**MDIO ready:** When set the MDIO is enabled and ready for a new transfer. This is also used to identify when a previous transaction has completed (for example, Read data is valid)

Table 8-16: **MDIO Configuration Word 1 (0x504) *(Cont'd)***

Bits	Default Value	Type	Description
10-8	N/A	RO	Reserved
11	0x0	WO	**Initiate:** Writing a 1 to this bit starts an MDIO transfer.
13-12	N/A	RO	Reserved
15-14	0x0	RW	**TX_OP:** This field controls the type of access performed when a one is written to initiate.
20-16	0x0	RW	**TX_REGAD:** This controls the register address being accessed. See Register Address (REGAD)
23-21	N/A	RO	Reserved
28-24	0x0	RW	**TX_PHYAD:** This controls the PHY address being accessed. See Physical Address (PHYAD)
31-29	N/A	RO	Reserved

Table 8-17: **MDIO Write Data (0x508)**

Bits	Default Value	Type	Description
15-0	0x0000	RW	Write Data
31-16	N/A	RO	Reserved

Table 8-18: **MDIO Read Data (0x50C)**

Bits	Default Value	Type	Description
15-0	0x0000	RO	**Read Data:** Only valid when MDIO ready is sampled high.
16	0x0	RO	**MDIO Ready:** This is a copy of bit 7 of the MDIO Control Word.
31-17	N/A	RO	Reserved

MDIO Addressing

MDIO addresses consists of two stages: physical address (PHYAD) and register address (REGAD).

Physical Address (PHYAD)

As shown in Figure 8-4, two PHY devices are attached to the MDIO bus. Each of these has a different physical address. To address the intended PHY, its physical address should be known by the MDIO master (in this case a V6EMAC) and placed into the PHYAD field of the MDIO frame (see MDIO Transactions).

The PHYAD field for an MDIO frame is a 5-bit binary value capable of addressing 32 unique addresses. However, every MDIO slave must respond to physical address 0.

This requirement dictates that the physical address for any particular PHY must not be set to 0 to avoid MDIO contention. Physical Addresses 1 through to 31, therefore, can be used to connect up to 31 PHY devices onto a single MDIO bus.

Physical Address 0 can be used to write a single command that is obeyed by all attached PHYs, such as a reset or power-down command.

Register Address (REGAD)

Having targeted a particular PHY using PHYAD, the individual configuration or status register within that particular PHY must now be addressed by placing the individual register address into the REGAD field of the MDIO frame (see MDIO Transactions).

The REGAD field for an MDIO frame is a 5-bit binary value capable of addressing 32 unique addresses. The first 16 of these registers (0 to 15) are defined by *IEEE Std 802.3-2008* [Ref 4]. The remaining 16 registers (16 to 31) are reserved for PHY vendors' own register definitions.

Accessing PHY Configuration Registers, through the MDIO using the Management Interface

The Management Interface is also used to access the MDIO interface, which operates as an MDIO master, of the core. The MDIO interface supplies a clock to the connected PHY, `mdc`, when the management interface is enabled. This clock is derived from the `s_axi_aclk` signal using the value in the `Clock Divide[4:0]` configuration register. The frequency of `mdc` is given by the following equation:

$$f_{MDC} = \frac{f_{s\_axi\_aclk}}{(1 + \text{Clock Divide}[4:0]) \times 2}$$

The frequency of `mdc` given by this equation should not exceed 2.5 MHz to comply with *IEEE Std 802.3-2008* [Ref 4] for this interface. To prevent `mdc` from being out of specification, the `Clock Divide[4:0]` value powers up at 00000, and while this value is in the register, it is impossible to enable the MDIO interface.

From the MDIO Interface block, the MDIO bus is internally connected within the Ethernet MAC to the MDIO Intersection block. Here the MDIO bus is split. It is internally connected to the 1000BASE-X PCS/PMA sublayer logic and connected to the V6EMAC MDIO ports (see MDIO Signal Definition, which can provide MDIO access to and from an external PHY device. The core must have been generated with MDIO enabled (and MDIO Enable must be set in the Management Configuration Register) to allow access to the internal MDIO registers and the external MDIO bus.

Accessing the Internal PCS/PMA Sublayer Management Registers

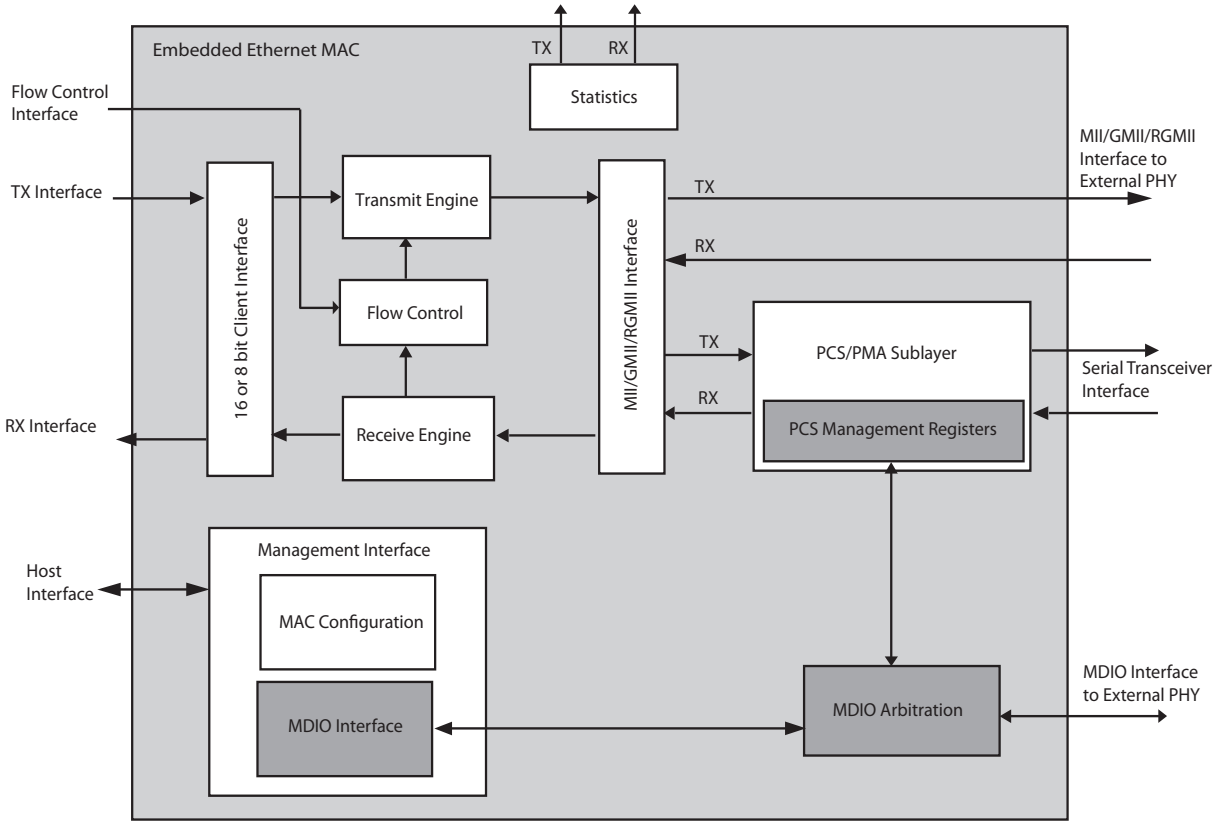

Figure 8-7: **MDIO Implementation of the V6EMAC**

The PCS/PMA sublayer logic, illustrated as a block in Figure 8-7, contains an MDIO slave, allowing access to its configuration and status registers. All connections are internal and active whenever MDIO operation is enabled.

The PHYAD of the internal PCS/PMA sublayer registers can be set with the PHYAD[4:0] port. To access the PCS/PMA sublayer registers, initiate an MDIO transaction using the matching physical address. The PCS/PMA sublayer also responds to Physical Address 0 (see MDIO Addressing), unless the PHY_IGNORE_ADZERO is set to TRUE in the GUI.

The PCS/PMA Management registers have different definitions depending on the mode of operation. See as appropriate:

- 1000BASE-X PCS/PMA Management Registers
- SGMII Management Registers

When no external PHY is present, the `mdio_i` port should be tied High and the `mdc_in` port tied Low.

Accessing the Management Registers of an External PHY using MDIO

Whenever an MDIO operation is enabled, connections can be made to the V6EMAC MDIO ports (see MDIO Signal Definition), which can provide MDIO access to and from an external PHY device. `mdio_i`, `mdio_o`, and `mdio_t` must be connected to a 3-state buffer to create the bidirectional wire, MDIO. This 3-state buffer can be either external to the

FPGA or internally integrated using an IOBUF with an appropriate I/O standard for the external PHY. This is illustrated in Figure 8-8.

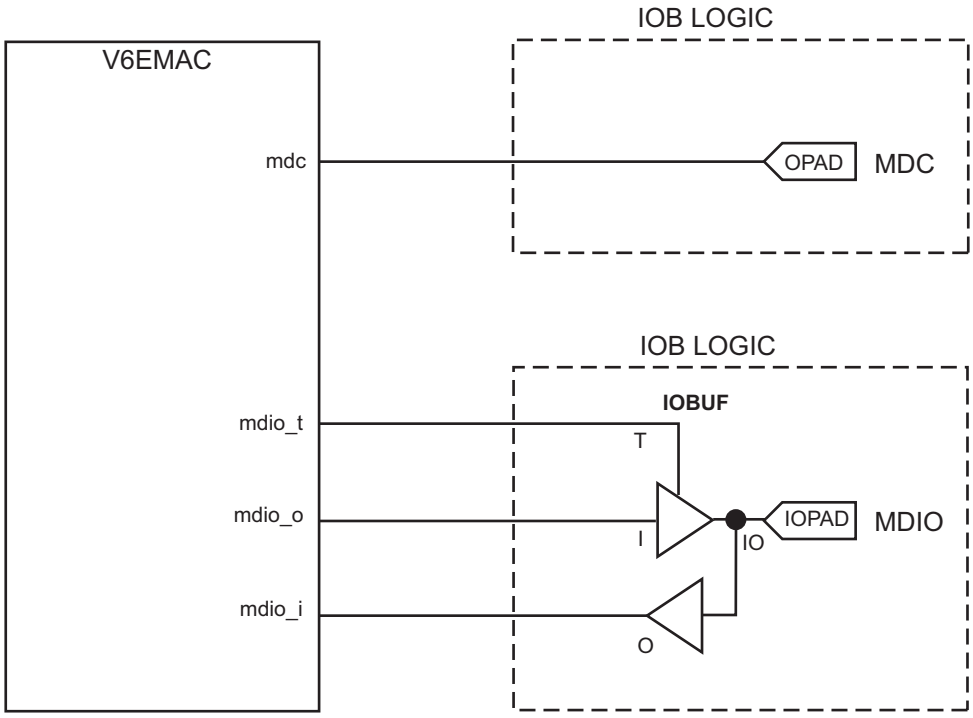

Figure 8-8: **External MDIO Interface**

In this mode of operation, the MDC clock is generated by the V6EMAC and obtained from the output port mdc; the input mdc_in port is unused and should be connected to logic 0.

To access the external PHY registers, initiate an MDIO transaction using the appropriate physical address. The PCS/PMA sublayer, even if it is not the physical interface, is still connected and responds to physical addresses 0 and the address matching the value of the PHYAD[4:0] port.

Accessing PCS/PMA Sublayer Management Registers using MDIO

Figure 8-9 shows the functional block diagram of the V6EMAC with the PCS management register, MDIO arbitration, and MDIO master blocks highlighted. This figure shows that the PCS/PMA sublayer registers can still be accessed using MDIO when the management interface is not in use.

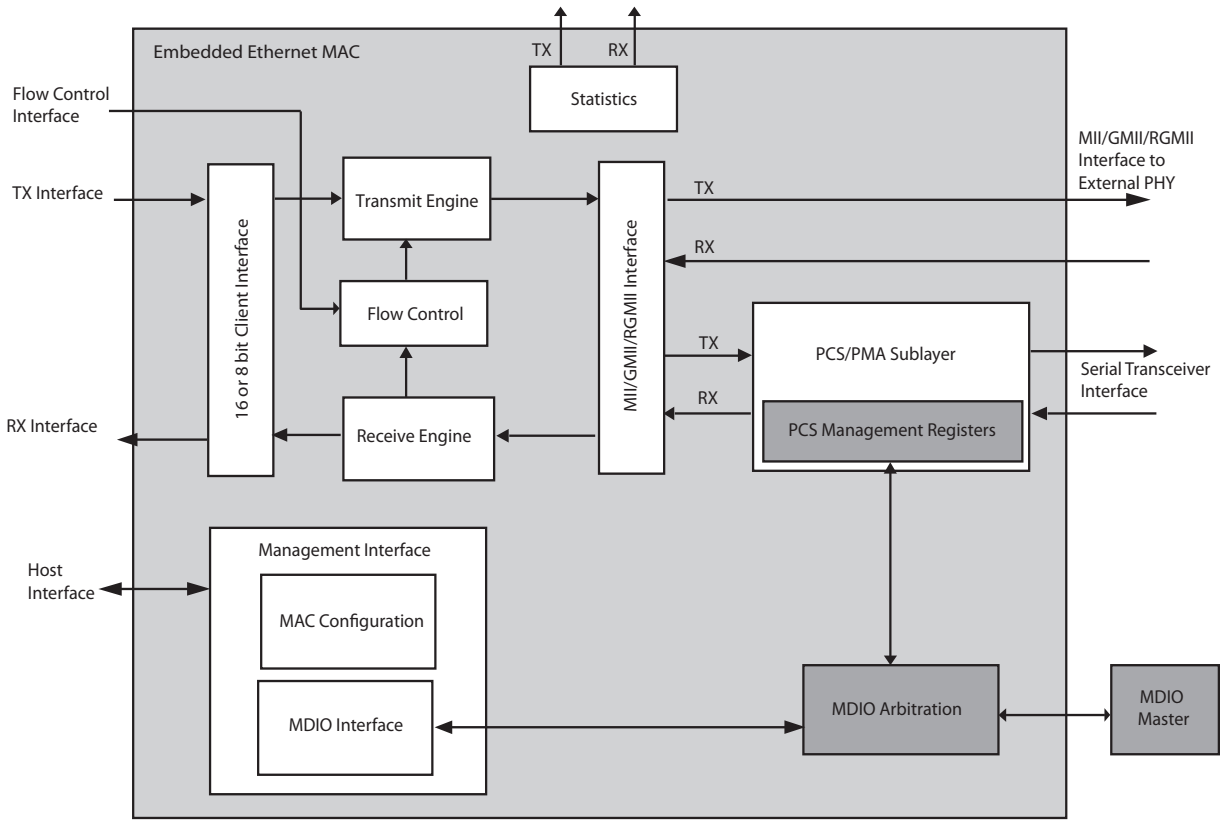

Figure 8-9: **MDIO access with no Management Interface**

In this situation, an MDIO master external to the V6EMAC must initiate the MDIO Transactions. This MDIO master can exist in FPGA logic or as an external device. In this situation, the MDC clock must be provided to the V6EMAC through the mdc_in port. The output signal mdc should be left unconnected.

The PCS/PMA Management registers have different definitions depending on the mode of operation. See as appropriate:

- 1000BASE-X PCS/PMA Management Registers
- SGMII Management Registers

1000BASE-X PCS/PMA Management Registers

The PCS/PMA sublayer block, shown in Figure 8-10, is contained within the Ethernet MAC. The PCS/PMA sublayer contains PCS Management registers, as listed in Table 8-19. These registers are described in further detail in Table 8-20 through to Table 8-31.

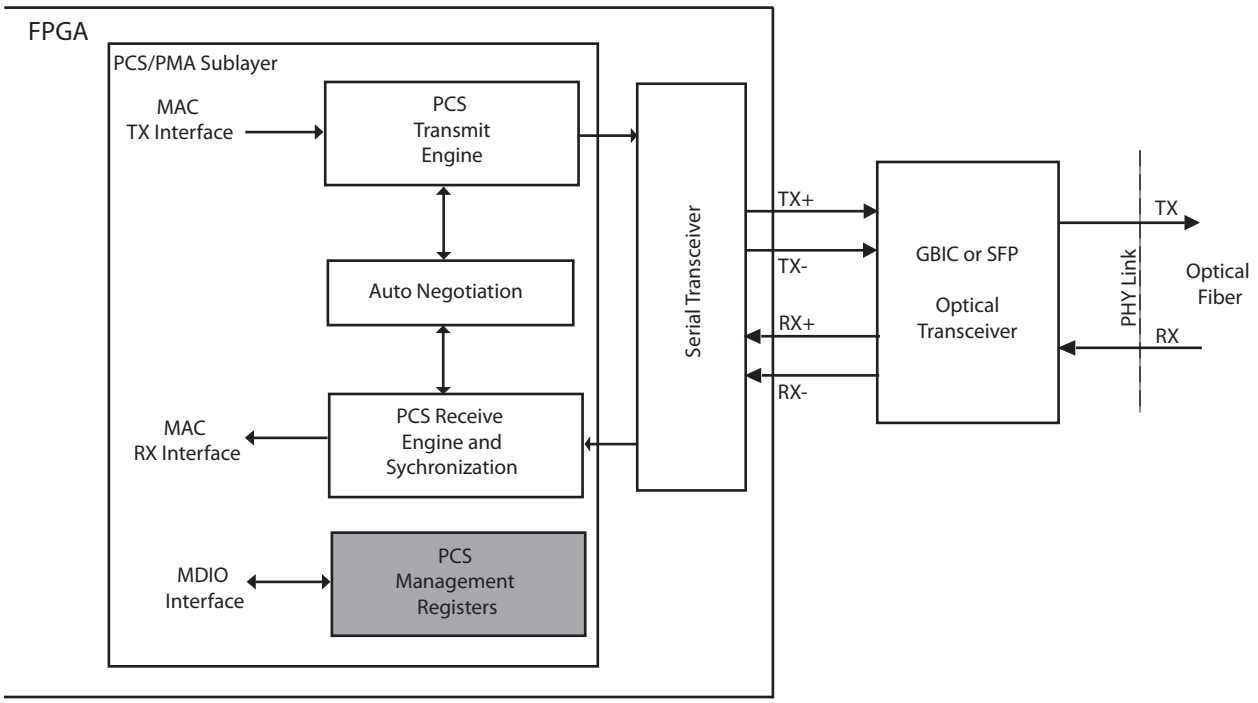

Figure 8-10: **1000BASE-X PCS/PMA Sublayer Logic and Physical connections**

Registers 0 through to 15 are defined in *IEEE Std 802.3-2008* [Ref 4]. These contain information relating to the operation of the 1000BASE-X PCS/PMA sublayer, including the status of the physical Ethernet link (PHY link). Additionally, these registers are directly involved in the operation of the 1000BASE-X auto-negotiation function that occurs between the V6EMAC and its link partner, which is the Ethernet device connected at the far end of the PHY link (see 1000BASE-X Auto-Negotiation).

Registers 16 and 17 are vendor-specific registers, defined by Xilinx.

Table 8-19: **Configuration Registers for 1000BASE-X PCS/PMA**

Register Address (REGAD)	Register name
0	Control Register (Register 0)
1	Status Register (Register 1)
2	PHY Identifier (Register 2)
3	PHY Identifier (Register 3)
4	Auto-Negotiation Advertisement Register (Register 4)
5	Auto-Negotiation Link Partner Ability Base Register (Register 5)
6	Auto-Negotiation Expansion Register (Register 6)

Table 8-19: **Configuration Registers for 1000BASE-X PCS/PMA *(Cont'd)***

Register Address (REGAD)	Register name
7	Auto-Negotiation Next Page Transmit Register (Register 7)
8	Auto-Negotiation Next Page Receive Register (Register 8)
15	Extended Status Register (Register 15)
16	Vendor Specific Register: Auto-Negotiation Interrupt Control Register (Register 16)
17	Vendor Specific Register: Loopback Control Register (Register 17)

Table 8-20: **Control Register (Register 0)**

Bit(s)	Name	Description	Type	Default Value
15	Reset	1=PCS/PMA reset. 0=Normal operation	RW (Self Clearing)	0
14	Loopback	1=Enable loopback mode 0=Disable loopback mode	RW	PHY_Loopback_MSB See Table 3-1, page 25
13	Speed Selection (LSB)	The V6EMAC always returns a 0 for this bit. Along with bit 6, speed selection of 1000 Mb/s is identified.	RO	0
12	Auto-Negotiation Enable	1=Enable auto-negotiation process 0=Disable auto-negotiation process	RW	PHY_AN_Enable See Table 3-1, page 25
11	Power Down	1=Power down, 0=Normal operation When set to 1, the serial transceiver is placed in a LOW power state. This bit requires a reset (see bit 15) to clear.	RW	0
10	Isolate	1=Electrically isolate the PHY from GMII. 0=Normal operation	RW	PHY_Isolate See Table 3-1, page 25
9	Restart Auto-Negotiation	1=Restart auto-negotiation process 0=Normal operation	RW (Self Clearing)	0
8	Duplex Mode	The V6EMAC always returns a 1 for this bit to signal full-duplex mode.	RO	1
7	Collision Test	The V6EMAC always returns a 0 for this bit to disable COL test.	RO	0
6	Speed Selection (MSB)	The V6EMAC always returns a 1 for this bit. Together with bit 13, speed selection of 1000 Mb/s is identified.	RO	1
5	Unidirectional Enable	Enable transmit regardless of whether a valid link has been established.	RW	PHY_Unidirection_Enable See Table 3-1, page 25
4:0	Reserved	Always returns 0s, writes ignored.	RO	0

Table 8-21: **Status Register (Register 1)**

Bit(s)	Name	Description	Type	Default Value
15	100BASE-T4	The V6EMAC always returns a 0 for this bit because 100BASE-T4 is not supported.	RO	0
14	100BASE-X Full Duplex	The V6EMAC always returns a 0 for this bit because 100BASE-X Full Duplex is not supported.	RO	0
13	100BASE-X Half Duplex	The V6EMAC always returns a 0 for this bit because 100BASE-X Half Duplex is not supported.	RO	0
12	10 Mb/s Full Duplex	The V6EMAC always returns a 0 for this bit because 10 Mb/s Full Duplex is not supported.	RO	0
11	10 Mb/s Half Duplex	The V6EMAC always returns a 0 for this bit because 10 Mb/s Half Duplex is not supported.	RO	0
10	100BASE-T2 Full Duplex	The V6EMAC always returns a 0 for this bit because 100BASE-T2 is not supported.	RO	0
9	100BASE-T2 Half Duplex	The V6EMAC always returns a 0 for this bit because 100BASE-T2 is not supported.	RO	0
8	Extended Status	The V6EMAC always returns a 1 for this bit, indicating the presence of the extended register (Register 15)	RO	1
7	Unidirectional Ability	Always returns a 1.	RO	1
6	MF Preamble Suppression	The V6EMAC always returns a 1 for this bit to indicate the support of management frame preamble suppression.	RO	1
5	Auto-Negotiation Complete	1=Auto-negotiation process completed 0=Auto-negotiation process not completed	RO	0
4	Remote Fault	1=Remote fault condition detected 0=No remote fault condition detected	RO (self clear on read)	0
3	Auto-Negotiation Ability	The V6EMAC always returns a 1 for this bit, indicating that the PHY is capable of auto-negotiation.	RO	1
2	Link status	1=PHY link is up 0=PHY link is down	RO (self clear on read)	0
1	Jabber Detect	The V6EMAC always returns a 0 for this bit because jabber detect is not supported.	RO	0
0	Extended Capability	The V6EMAC always returns a 0 for this bit because no extended register set is supported.	RO	0

Table 8-22: **PHY Identifier (Register 2)**

Bit(s)	Name	Description	Type	Default Value
15:0	Organizationally Unique Identifier (upper half word)	Organizationally Unique Identifier (OUI) from IEEE is 0x000A35. This register contains the upper half word (bits 3-18 = 0x0028)	RO	000000000101000

Table 8-23: **PHY Identifier (Register 3)**

Bit(s)	Name	Description	Type	Default Value
15:10	Organizationally Unique Identifier (lower bits)	Organizationally Unique Identifier (OUI) from IEEE is 0x000A35. This register contains the lower bits (bits 19-24 = 0x35)	RO	110101
9:4	Manufacturer's Model Number	Always returns 0.	RO	0
3:0	Recision Number	Always returns 0.	RO	0

Table 8-24: **Auto-Negotiation Advertisement Register (Register 4)**

Bit(s)	Name	Description	Type	Default Value
15	Next Page	1=Next page functionality is advertised	RW	0
14	Reserved	Always returns 0.	RO	0
13:12	Remote Fault	00=No error 01=Offline 10=Link failure 11=Auto-negotiation error	RW (Self clearing to 00 after auto-negotiation	00
11:9	Reserved	Always returns 0.	RO	0
8:7	Pause	00=No PAUSE 01=Symmetric PAUSE 10=Asymmetric PAUSE towards link partner 11=Both symmetric PAUSE and asymmetric pause towards link partner	RW	11
6	Half Duplex	The V6EMAC always returns a 0 for this bit because half duplex is not supported.	RO	0
5	Full Duplex	1= Full-duplex mode is advertised 0=Full-duplex mode is not advertised	RW	1
4:0	Reserved	Always returns 0.	RO	0

Table 8-25: **Auto-Negotiation Link Partner Ability Base Register (Register 5)**

Bit(s)	Name	Description	Type	Default Value
15	Next Page	1=Next page functionality is advertised	RW	0
14	Acknowledge	Used by the auto-negotiation function to indicate reception of a link partner's base or next page.	RO	0
13:12	Remote Fault	00=No error 01=Offline 10=Link failure 11=Auto-negotiation error	RO	00
11:9	Reserved	Always returns 0.	RO	0
8:7	Pause	00=No PAUSE 01=Symmetric PAUSE supported 10=Asymmetric PAUSE supported 11=Both symmetric PAUSE and asymmetric pause supported	RO	00
6	Half Duplex	1=Half-duplex mode is supported 0=Half-duplex mode is not supported	RO	0
5	Full Duplex	1= Full-duplex mode is supported 0=Full-duplex mode is not supported	RO	0
4:0	Reserved	Always returns 0.	RO	0

Table 8-26: **Auto-Negotiation Expansion Register (Register 6)**

Bit(s)	Name	Description	Type	Default Value
15:3	Reserved	Always returns 0.	RO	0
2	Next Page Able	The V6EMAC always returns a 1 for this bit because the device is next page able.	RO	1
1	Page Received	1=A new page is received 0=A new page is not received	RO (Self clear on read)	0
0	Reserved	Always returns 0.	RO	0

Table 8-27: **Auto-Negotiation Next Page Transmit Register (Register 7)**

Bit(s)	Name	Description	Type	Default Value
15	Next Page	1=Additional next page(s) follows. 0=Last page	RW	0
14	Reserved	Always returns 0.	RO	0
13	Message Page	1=Message page 0=Unformattted page	RW	1
12	Acknowledge 2	1=Complies with message 0-Cannot comply with message	RW	0
11	Toggle	Value toggles between subsequent pages.	RO	0
10:0	Message or Unformatted code Field	Message code field or unformatted page encoding as dictated by bit 13	RW	00000000001 (Null Message Code)

Table 8-28: **Auto-Negotiation Next Page Receive Register (Register 8)**

Bit(s)	Name	Description	Type	Default Value
15	Next Page	1=Additional next page(s) to follow. 0=Last page	RO	0
14	Acknowledge	Used by auto-negotiation function to indicate reception of a link partner's base or next page	RO	0
13	Message Page	1=Message page 0=Unformattted page	RO	0
12	Acknowledge 2	1=Complies with message 0-Cannot comply with message	RO	0
11	Toggle	Value toggles between subsequent next pages.	RO	0
10:0	Message or Unformatted code Field	Message code field or unformatted page encoding as dictated by bit 13	RO	0

Table 8-29: **Extended Status Register (Register 15)**

Bit(s)	Name	Description	Type	Default Value
15	1000BASE-X Full Duplex	The V6EMAC always returns a 1 for this bit because 1000BASE-X full duplex is supported.	RO	1
14	1000BASE-X Half Duplex	The V6EMAC always returns a 0for this bit because 1000BASE-X half duplex is not supported.	RO	0
13	1000BASE-T Full Duplex	The V6EMAC always returns a 0 for this bit because 1000BASE-T full duplex is not supported.	RO	0
12	1000BASE-T Half Duplex	The V6EMAC always returns a 0for this bit because 1000BASE-T half duplex is not supported.	RO	0
11:0	Reserved	Always returns 0s.	RO	0

Table 8-30: **Vendor Specific Register: Auto-Negotiation Interrupt Control Register (Register 16)**

Bit(s)	Name	Description	Type	Default Value
15:2	Reserved	Always returns 0s.	RO	0
1	Interrupt Status	1=Interrupt is asserted 0=Interrupt is not asserted If the interrupt is enabled, this bit is set upon the completion of an auto-negotiation cycle; it is cleared only by writing 0 to this bit. If the interrupt is disabled, this bit is reset to 0. The an_interrupt port is wired to this bit.	RW	0
0	Interrupt Enable	1=Interrupt is enabled 0=Interrupt is disabled	RW	1

Table 8-31: **Vendor Specific Register: Loopback Control Register (Register 17)**

Bit(s)	Name	Description	Type	Default Value
15:1	Reserved	Always returns 0s.	RO	0
0	Loopback Position	0=Loopback(when enabled) occurs in the V6EMAC directly before the interface to the serial transceiver. 1=Loopback(when enabled) occurs in the serial transceiver. Note: Loopback is enabled or disabled using Register 0 bit 14 (see Control Register (Register 0))	RW	PHY_Loopback in Transceiver See Table 3-1, page 25

SGMII Management Registers

Figure 8-11 shows the PCS/PMA sublayer block in an SGMII implementation. The PCS/PMA sublayer of the V6EMAC, when performing the SGMII standard, contains PCS Management registers as listed in Table 8-32. These registers are described in further detail in Table 8-33 through Table 8-44.

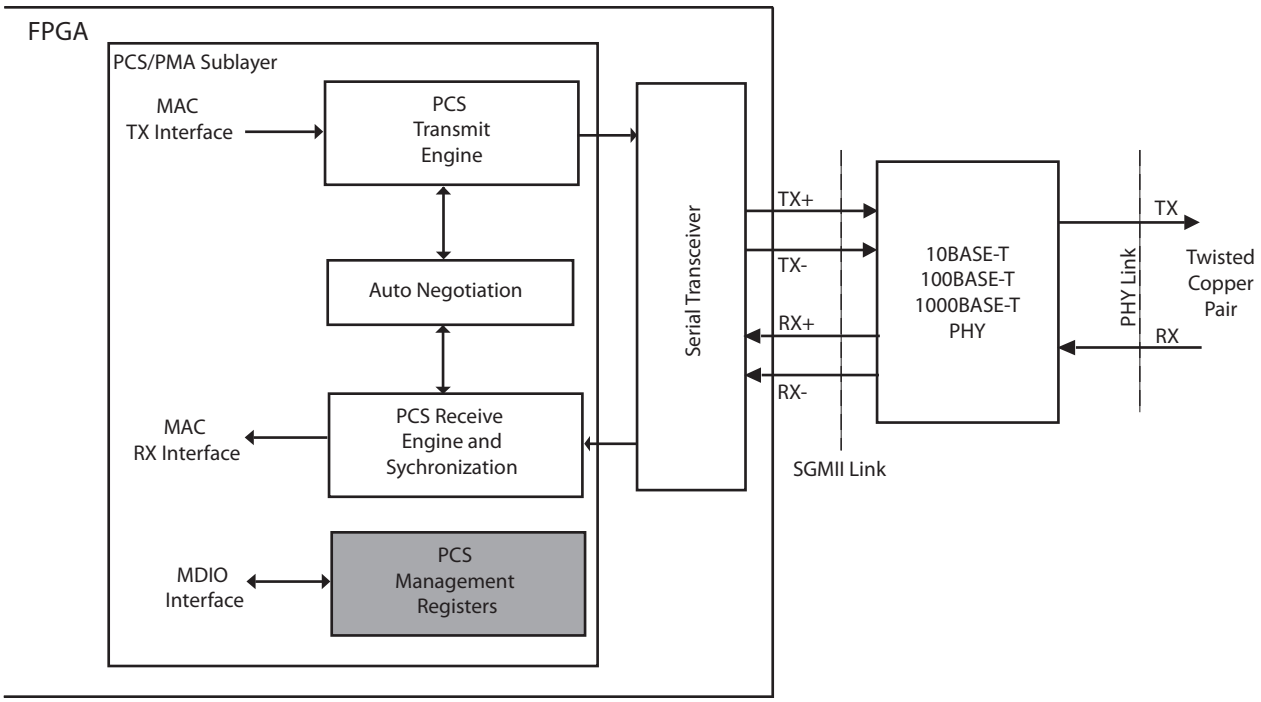

Figure 8-11: **SGMII Logic and Physical Connections**

These registers contain information relating to the operation of the system, including the status of both the SGMII link and the physical Ethernet link (PHY link). See Figure 8-11.

Additionally, these registers are directly involved in the operation of the SGMII auto-negotiation function which occurs between the V6EMAC and the external PHY device, illustrated as a Tri-speed BASE-T PHY in Figure 8-11 (see SGMII Auto-Negotiation).

Table 8-32: **SGMII Configuration Registers for 1000BASE-X PCS/PMA**

Register Address (REGAD)	Register name
0	SGMII Control Register (Register 0)
1	SGMII Status Register (Register 1)
2	PHY Identifier (Register 2)
3	PHY Identifier (Register 3)
4	SGMII Auto-Negotiation Advertisement Register (Register 4)
5	SGMII Auto-Negotiation Link Partner Ability Base Register (Register 5)
6	SGMII Auto-Negotiation Expansion Register (Register 6)
7	SGMII Auto-Negotiation Next Page Transmit Register (Register 7)
8	SGMII Auto-Negotiation Next Page Receive Register (Register 8)
15	SGMII Extended Status Register (Register 15)
16	SGMII Vendor Specific Register: Auto-Negotiation Interrupt Control Register (Register 16)
17	SGMII Vendor Specific Register: Loopback Control Register (Register 17)

Table 8-33: **SGMII Control Register (Register 0)**

Bit(s)	Name	Description	Type	Default Value
15	Reset	1=PCS/PMA reset. 0=Normal operation	RW (Self Clearing)	0
14	Loopback	1=Enable loopback mode 0=Disable loopback mode	RW	PHY_Loopback_MSB See Table 3-1, page 25
13	Speed Selection (LSB)	The V6EMAC always returns a 0 for this bit. Along with bit 6, speed selection of 1000 Mb/s is identified.	RO	0
12	Auto-Negotiation Enable	1=Enable SGMII auto-negotiation process 0=Disable SGMII auto-negotiation process	RW	PHY_AN_Enable See Table 3-1, page 25
11	Power Down	1=Power down, 0=Normal operation When set to 1, the serial transceiver is placed in a LOW power state. This bit requires a reset (see bit 15) to clear.	RW	0
10	Isolate	1=Isolate the SGMII logic from GMII. 0=Normal operation	RW	PHY_Isolate See Table 3-1, page 25
9	Restart Auto-Negotiation	1=Restart auto-negotiation process across the SGMII link 0=Normal operation	RW (Self Clearing)	0
8	Duplex Mode	The V6EMAC always returns a 1 for this bit to signal full-duplex mode.	RO	1

Table 8-33: **SGMII Control Register (Register 0) *(Cont'd)***

Bit(s)	Name	Description	Type	Default Value
7	Collision Test	The V6EMAC always returns a 0 for this bit to disable COL test.	RO	0
6	Speed Selection (MSB)	The V6EMAC always returns a 1 for this bit. Together with bit 13, speed selection of 1000 Mb/s is identified.	RO	1
5	Unidirectional Enable	Enable transmit regardless of whether a valid link has been established.	RW	PHY_Unidirection_Enable See Table 3-1, page 25
4:0	Reserved	Always returns 0s, writes ignored.	RO	0

Table 8-34: **SGMII Status Register (Register 1)**

Bit(s)	Name	Description	Type	Default Value
15	100BASE-T4	The V6EMAC always returns a 0 for this bit because 100BASE-T4 is not supported.	RO	0
14	100BASE-X Full Duplex	The V6EMAC always returns a 0 for this bit because 100BASE-X Full Duplex is not supported.	RO	0
13	100BASE-X Half Duplex	The V6EMAC always returns a 0 for this bit because 100BASE-X Half Duplex is not supported.	RO	0
12	10 Mb/s Full Duplex	The V6EMAC always returns a 0 for this bit because 10 Mb/s Full Duplex is not supported.	RO	0
11	10 Mb/s Half Duplex	The V6EMAC always returns a 0 for this bit because 10 Mb/s Half Duplex is not supported.	RO	0
10	100BASE-T2 Full Duplex	The V6EMAC always returns a 0 for this bit because 100BASE-T2 is not supported.	RO	0
9	100BASE-T2 Half Duplex	The V6EMAC always returns a 0 for this bit because 100BASE-T2 is not supported.	RO	0
8	Extended Status	The V6EMAC always returns a 1 for this bit, indicating the presence of the extended register (Register 15)	RO	1
7	Unidirectional Ability	Always returns a 1.	RO	1
6	MF Preamble Suppression	The V6EMAC always returns a 1 for this bit to indicate the support of management frame preamble suppression.	RO	1
5	Auto-Negotiation Complete	1=Auto-negotiation process completed 0=Auto-negotiation process not completed	RO	0
4	Remote Fault	1=Remote fault condition detected 0=No remote fault condition detected	RO (self clear on read)	0
3	Auto-Negotiation Ability	The V6EMAC always returns a 1 for this bit. indicating that the PHY is capable of auto-negotiation.	RO	1

Table 8-34: **SGMII Status Register (Register 1) *(Cont'd)***

Bit(s)	Name	Description	Type	Default Value
2	Link status	The link status of the V6EMAC with its external PHY across the SGMII link (see Figure 8-11) 1=SGMII link is up 0=SGMII link is down	RO (self clear on read)	0
1	Jabber Detect	The V6EMAC always returns a 0 for this bit because jabber detect is not supported.	RO	0
0	Extended Capability	The V6EMAC always returns a 0 for this bit because no extended register set is supported.	RO	0

Table 8-35: **PHY Identifier (Register 2)**

Bit(s)	Name	Description	Type	Default Value
15:0	Organizationally Unique Identifier (upper half word)	Organizationally Unique Identifier (OUI) from IEEE is 0x000A35. This register contains the upper half word (bits 3-18 = 0x0028)	RO	000000000101000

Table 8-36: **PHY Identifier (Register 3)**

Bit(s)	Name	Description	Type	Default Value
15:10	Organizationally Unique Identifier (lower bits)	Organizationally Unique Identifier (OUI) from IEEE is 0x000A35. This register contains the lower bits (bits 19-24 = 0x35)	RO	110101
9:4	Manufacturer's Model Number	Always returns 0.	RO	0
3:0	Recision Number	Always returns 0.	RO	0

Table 8-37: **SGMII Auto-Negotiation Advertisement Register (Register 4)**

Bit(s)	Name	Description	Type	Default Value
15:0	All bits	SGMII defined value sent from the MAC to the PHY	RO	000000000000001

Table 8-38: **SGMII Auto-Negotiation Link Partner Ability Base Register (Register 5)**

Bit(s)	Name	Description	Type	Default Value
15	PHY Link status	This refers to the link status of the external PHY device with its link partner across the PHY link (see Figure 8-11). 1-Link up 0=Link down	RO	1
14	Acknowledge	Used by the auto-negotiation function to indicate reception of a link partner's base or next page.	RO	0
13	Reserved	Always returns 0.	RO	0
12	Duplex mode	The resolved duplex mode that the external PHY device has auto-negotiated with its link partner across the PHY link (see Figure 8-11). 1=Full duplex 0=Half duplex	RO	0

Table 8-38: **SGMII Auto-Negotiation Link Partner Ability Base Register (Register 5) *(Cont'd)***

Bit(s)	Name	Description	Type	Default Value
11:10	Speed	The resolved operating speed that the external PHY device has auto-negotiated with its link partner across the PHY link (see Figure 8-11). 00=10MB/s 01=100Mb/s 10=1000Mb/s 11=Reserved	RO	00
9:1	Reserved	Always returns 0.	RO	0
0	Reserved	Always returns 1.	RO	1

Table 8-39: **SGMII Auto-Negotiation Expansion Register (Register 6)**

Bit(s)	Name	Description	Type	Default Value
15:3	Reserved	Always returns 0.	RO	0
2	Next Page Able	The V6EMAC always returns a 1 for this bit because the device is next page able.	RO	1
1	Page Received	1=A new page is received 0=A new page is not received	RO (Self clear on read)	0
0	Reserved	Always returns 0.	RO	0

Table 8-40: **SGMII Auto-Negotiation Next Page Transmit Register (Register 7)**

Bit(s)	Name	Description	Type	Default Value
15	Next Page	1=Additional next page(s) to follow. 0=Last page	RW	0
14	Reserved	Always returns 0.	RO	0
13	Message Page	1=Message page 0=Unformattted page	RW	1
12	Acknowledge 2	1=Complies with message 0=Cannot comply with message	RW	0
11	Toggle	Value toggles between subsequent pages.	RO	0
10:0	Message or Unformatted code Field	Message code field or unformatted page encoding as dictated by bit 13	RW	00000000001 (Null Message Code)

Table 8-41: **SGMII Auto-Negotiation Next Page Receive Register (Register 8)**

Bit(s)	Name	Description	Type	Default Value
15	Next Page	1=Additional next page(s) to follow. 0=Last page	RO	0
14	Acknowledge	Used by auto-negotiation function to indicate reception of a link partner's base or next page	RO	0
13	Message Page	1=Message page 0=Unformattted page	RO	0

Table 8-41: **SGMII Auto-Negotiation Next Page Receive Register (Register 8) *(Cont'd)***

Bit(s)	Name	Description	Type	Default Value
12	Acknowledge 2	1=Complies with message 0=Cannot comply with message	RO	0
11	Toggle	Value toggles between subsequent next pages.	RO	0
10:0	Message or Unformatted code Field	Message code field or unformatted page encoding as dictated by bit 13	RO	0

Table 8-42: **SGMII Extended Status Register (Register 15)**

Bit(s)	Name	Description	Type	Default Value
15	1000BASE-X Full Duplex	The V6EMAC always returns a 1 for this bit because 1000BASE-X full duplex is supported.	RO	1
14	1000BASE-X Half Duplex	The V6EMAC always returns a 0for this bit because 1000BASE-X half duplex is not supported.	RO	0
13	1000BASE-T Full Duplex	The V6EMAC always returns a 0 for this bit because 1000BASE-T full duplex is not supported.	RO	0
12	1000BASE-T Half Duplex	The V6EMAC always returns a 0for this bit because 1000BASE-T half duplex is not supported.	RO	0
11:0	Reserved	Always returns 0s.	RO	0

Table 8-43: **SGMII Vendor Specific Register: Auto-Negotiation Interrupt Control Register (Register 16)**

Bit(s)	Name	Description	Type	Default Value
15:2	Reserved	Always returns 0s.	RO	0
1	Interrupt Status	1=Interrupt is asserted 0=Interrupt is not asserted If the interrupt is enabled, this bit is set upon the completion of an auto-negotiation cycle; it is cleared only by writing 0 to this bit. If the interrupt is disabled, this bit is reset to 0. The an_interrupt port is wired to this bit.	RW	0
0	Interrupt Enable	1=Interrupt is enabled 0=Interrupt is disabled	RW	1

Table 8-44: **SGMII Vendor Specific Register: Loopback Control Register (Register 17)**

Bit(s)	Name	Description	Type	Default Value
15:1	Reserved	Always returns 0s.	RO	0
0	Loopback Position	0=Loopback(when enabled) occurs in the V6EMAC directly before the interface to the serial transceiver. 1=Loopback(when enabled) occurs in the serial transceiver. Note: Loopback is enabled or disabled using Register 0 bit 14 (see Control Register (Register 0))	RW	PHY_Loopback in Transceiver See Table 3-1, page 25

Interrupt Controller

An Interrupt Block is implemented in the V6MAC to assert an interrupt when a pending MDIO transaction has completed. Interrupt registers are shown in Table 8-45.

Table 8-45: **Interrupt Controller Configuration**

Address (Hex)	Default Value	Type	Description
0x600	0x00	RW	**Interrupt status Register**. Indicates the status of an interrupt. Any asserted interrupt can be cleared by directly writing a 0 to the concerned bit location.
0x610	0x00	RO	**Interrupt Pending Register**. Indicates the pending status of an interrupt. Bits in this register are only set when the corresponding bits in IER and ISR are set.
0x620	0x00	RW	**Interrupt Enable Register**. Indicates the enable state of an interrupt. Writing a 1 to any bit enables that particular interrupt.
0x630	0x00	WO	**Interrupt Clear Register**. Writing a 1 to any bit of this register clears that particular interrupt.

Bit 0 of all interrupt registers is used to indicate the MDIO Transaction complete interrupt. Bits [31:1] are Reserved.

Address/Frame Filter

The MAC can be configured with an optional Address/Frame Filter. This is available irrespective of the use of the management interface but has much reduced functionality if no management interface is present.

The Address/Frame Filter performs two functions:

- It checks if any received packet matches one of the predefined Destination Address values: Pause Address, Broadcast Address, User Defined Unicast Address and the special multicast Pause Address.
- Compares the first 64 bytes of a received packet against a user defined pattern.

In the case of the Destination Address compares the results are used in other blocks within the MAC, such as flow control and in the generation of statistics vectors.

The other function of the Address/Frame Filter is much more flexible and allows the user to specify any match pattern within the first 64 bytes whilst ignoring any other byte or bit values. This is extremely flexible as it enables packets to be filtered based on almost any header field or combination of header fields. Each Address/Frame Filter contains two 512 bit registers (64 bytes):

- Address/Frame Filter Value Register. this pattern is compared to the first 512 bits received in a frame with bit 0 being the first bit within a frame to be received.
- Address/Frame Filter Mask Value Register. Each bit within this register refers to the same bit number within the Address/Frame Filter Value Register. when a bit in the Mask Value Register is set to
  - logic 1, The same bit number within the Address/Frame Filter Value Register is compared with the respective bit in the received frame and must match if the overall Address/Frame Filter is to obtain a match.
  - logic 0, the same bit number within the Address/Frame Filter Value Register is not compared. This effectively turns the respective bit in the Address/Frame

Filter ValueRegister into a don't care bit: the overall Address/Frame Filter is capable of obtaining a match even if this bit does not match

This is described in more detail in Using the Address/Frame Filter.

The user can specify up to 8 Address/Frame filters in the MAC netlist and each is accessed through address mapped registers. However, because each filter requires 32 registers to access the pattern and mask values there is a control register which specifies which of the filters is being accessed with all filters being accessed through the same register set. When one or more Address/Frame filters are specified the `rx_axis_filter_tuser` output is available from the netlist. This is sized depending upon the number of filters selected and provides a direct pass/fail indication on a per filter basis. See Using the Address/Frame Filter for more detail.

Table 8-46 shows the available Address/Frame filter configuration registers.

Table 8-46: **Address/Frame Filter Configuration**

Address (Hex)	Description
0x700	Unicast Address word 0
0x704	Unicast Address word 1
0x708	Address/Frame filter Control
0x70C	Address/Frame filter Enable
0x710	Address/Frame filter value bytes 3-0
0x714	Address/Frame Filter value bytes 7-4
0x718	Address/Frame Filter value bytes 11-8
0x71C	Address/Frame Filter value bytes 15-12
0x720	Address/Frame Filter value bytes 19-16
0x724	Address/Frame Filter value bytes 23-20
0x728	Address/Frame Filter value bytes 27-24
0x72C	Address/Frame Filter value bytes 31-28
0x730	Address/Frame Filter value bytes 35-32
0x734	Address/Frame Filter value bytes 39-36
0x738	Address/Frame Filter value bytes 43-40
0x73C	Address/Frame Filter value bytes 47-44
0x740	Frame Filter value bytes 51-48
0x744	Frame Filter value bytes 55-52
0x748	Frame Filter value bytes 59-56
0x74C	Frame Filter value bytes 63-60
0x750	Frame filter mask value bytes 3-0
0x754	Frame Filter mask value bytes 7-4
0x758	Frame Filter mask value bytes 11-8
0x75C	Frame Filter mask value bytes 15-12

Table 8-46: **Address/Frame Filter Configuration *(Cont'd)***

Address (Hex)	Description
0x760	Frame Filter mask value bytes 19-16
0x764	Frame Filter mask value bytes 23-20
0x768	Frame Filter mask value bytes 27-24
0x76C	Frame Filter mask value bytes 31-28
0x770	Frame Filter mask value bytes 35-32
0x774	Frame Filter mask value bytes 39-36
0x778	Frame Filter mask value bytes 43-40
0x77C	Frame Filter mask value bytes 47-44
0x780	Frame Filter mask value bytes 51-48
0x784	Frame Filter mask value bytes 55-52
0x788	Frame Filter mask value bytes 59-56
0x78C	Frame Filter mask value bytes 63-60

The contents of each configuration Register is shown in Table 8-47 through Table 8-52.

Table 8-47: **Unicast Address (Word 0) (0x700)**

Bits	Default Value	Type	Description
31-0	0xFFFFFFFF	RW	**Address/Frame filter unicast address[31:0]**: This address is used by the MAC to match against the destination address of any incoming frames. The address is ordered so the first byte transmitted/received is the lowest positioned byte in the register; for example, a MAC address of AA-BB-CC-DD-EE-FF would be stored in Address[47:0] as 0xFFEEDDCCBBAA.

Table 8-48: **Unicast Address (Word 1) (0x704)**

Bits	Default Value	Type	Description
15-0	0xFFFF	RW	**Address/Frame filter unicast address[47:32]**: See description in Table 8-47.
31-16	N/A	RO	Reserved

Table 8-49: **Address/Frame Filter Control (0x708)**

Bits	Default Value	Type	Description
31	1	RW	**Promiscuous Mode:** If this bit is set to 1 the Address/Frame filter is set to operate in promiscuous mode. All frames are passed to the receiver client regardless of the destination address.
30-3	N/A	RO	Reserved
2-0	0	RW	**Filter Index:** All Address/Frame filters are mapped to the same location with the filter index specifying which physical filter is to be accessed.

Table 8-50: **Address/Frame Filter Enable (0x70C)**

Bits	Default Value	Type	Description
31-3	N/A	RO	Reserved
0	1	RW	**Filter Enable:** This enable relates to the physical Address/Frame Filter pointed to by the Filter index. If clear, the filter passes all packets.

Table 8-51: **Address/Frame Filter Value (0x710-0x74C)**

Bits	Default Value	Type	Description
31-0	bits 47:0 =1 All other =0	RW	**Filter Value** All filter value registers have the same format. The lower 31 bits of filter value, at address 0x710, relating to the Filter at physical Address/Frame Filter index, that is to be written to the address table. The value is ordered so that the first byte transmitted/received is the lowest positioned byte in the register; for example, a MAC address of AA-BB-CC-DD-EE-FF would be stored in Filter Value[47:0] as 0xFFEEDDCCBBAA. By default the Address/Frame filters are configured to match against the broadcast address.

Table 8-52: **Address/Frame Filter Mask Value (0x750-0x790)**

Bits	Default Value	Type	Description
31-0	bits 47:0 =1 All other =0	RW	**Mask Value**. All mask value registers have the same format. If a mask bit is set to 1 then the corresponding bit of the Filter Value is compared by the Address/Frame filter. For example, if a basic Destination address comparison was desired then bits 47:0 should be written to 1 and all other bits to 0.

Using the Address/Frame Filter

The Configurable Address/Frame filters can be used to perform simple Destination Address filtering, Multicast Group matching, Source Address matching, VLAN field matching. The use of the Mask register enables any field or combination of fields in the first 64 bytes to be isolated and matched.

The Address/Frame Filter Control Register, shown in Table 8-49, allows the user to enable or disable the Address/Frame Filter by setting the Promiscuous Mode Bit, which has the following functionality:

- when enabled, all good frames are marked as good
- when disabled, only frames which match one or more of the pre-defined Destination Address filters or the configurable Address/Frame Filters are marked good

When more than one Address/Frame Filter is generated it is necessary to write to the Address/Frame Filter Control Register to specify which Address/Frame Filter is being accessed prior to writing to any of the filter specific registers. It is then possible to enable each Address/Frame filter individually. While a particular filter is disabled it does not match any packets. It is recommended that Address/Frame Filters are disabled prior to updating the match pattern to avoid unexpected packets being accepted.

By default all Address/Frame filters are configured with all 1s in the bottom 48 bits in both the Address/Frame Filter Value registers and the Address/Frame Filter Mask Value registers, this results in a broadcast frame match, which has no effect as they are already accepted by the pre-defined Destination Address filters. When the Frame Filter Value and Mask Value have been updated to the desired values the Filter should be enabled.

In Figure 8-12, the reception of an error free frame that matches against filter 0 is shown. When one or more filters are generated, the `rx_axis_filter_tuser` bus is generated with one extra bit, for example if four filters are selected, `rx_axis_filter_tuser` would be a five bit bus. This extra bit (always the upper bit) is used to provide the 'else' case. It is asserted if any of the user defined filters match a frame. If using the `rx_axis_filter_tuser` outputs, this means the frame is serviced by a dedicated output and should therefore be dropped by the else output. This is shown in Figure 8-13 where a good frame has matched against a pre-defined Destination Address filter but failed to match any of the configurable filters.

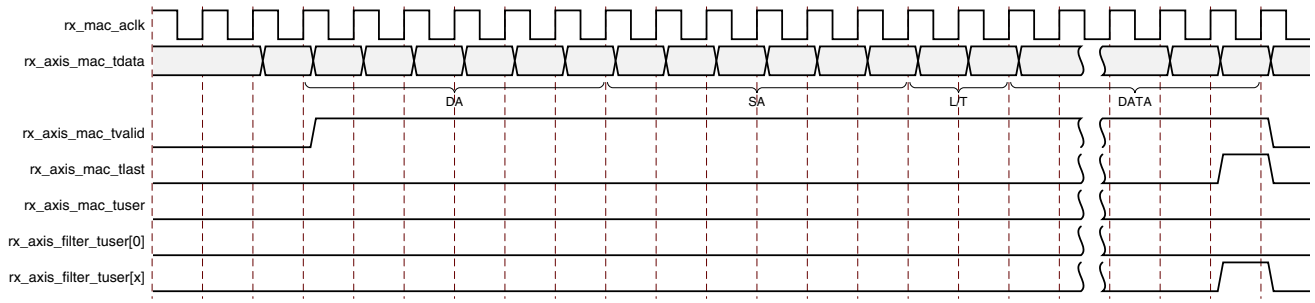

Figure 8-12: **Received Frame With a Match on Filter 0**

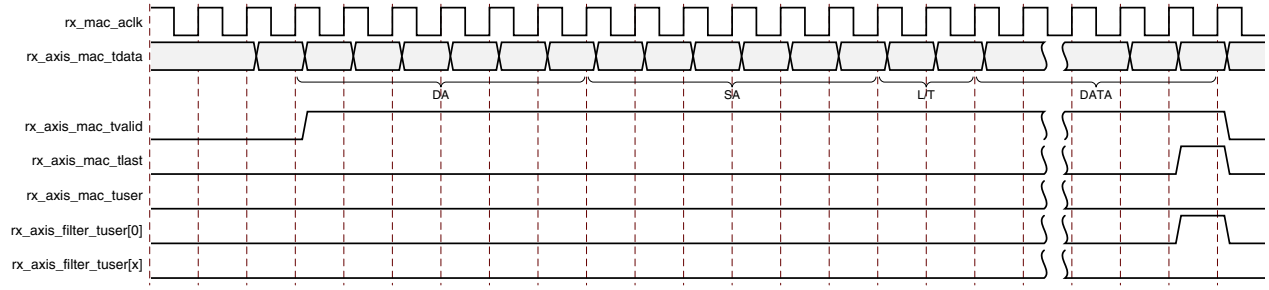

Figure 8-13: **Received Frame With No Match on Configurable Address/Frame Filters**

This extra bit allows the `rx_axis_filter_tuser` bus to be used to directly drive FIFOs for particular filter matches, such as VLAN priority. When the Frame Filter is configured in Promiscuous Mode the `rx_axis_filter_tuser` bits continue to operate as normal.

Priority FIFO Example

In Figure 8-14 the MAC is shown connected to priority FIFOs. In this case Address/Frame Filter 0 is set up to match on the VLAN Type and the VLAN Priority field with a value of 7. In a standard VLAN Ethernet frame, the VLAN type value of 0x8100 is found in bytes 13-14, with the priority field being the upper 3 bits of byte 15. This required the following register settings:

- Address/Frame Filter Value bytes 15-12 set to 0xE0008100
- Address/Frame Filter Mask Value bytes 15-12 set to 0xE0FFFF00
- All other Address/Frame Filter Mask Value bytes set to 0x0

In this case, the FIFO that is using `rx_axis_filter_tuser` is only passed good VLAN frames that have a priority field set to 7. The default FIFO, which is using `rx_axis_filter_tuser`[4] only accepts good frames which are either not VLAN frames or have a priority field with a value other than 7.

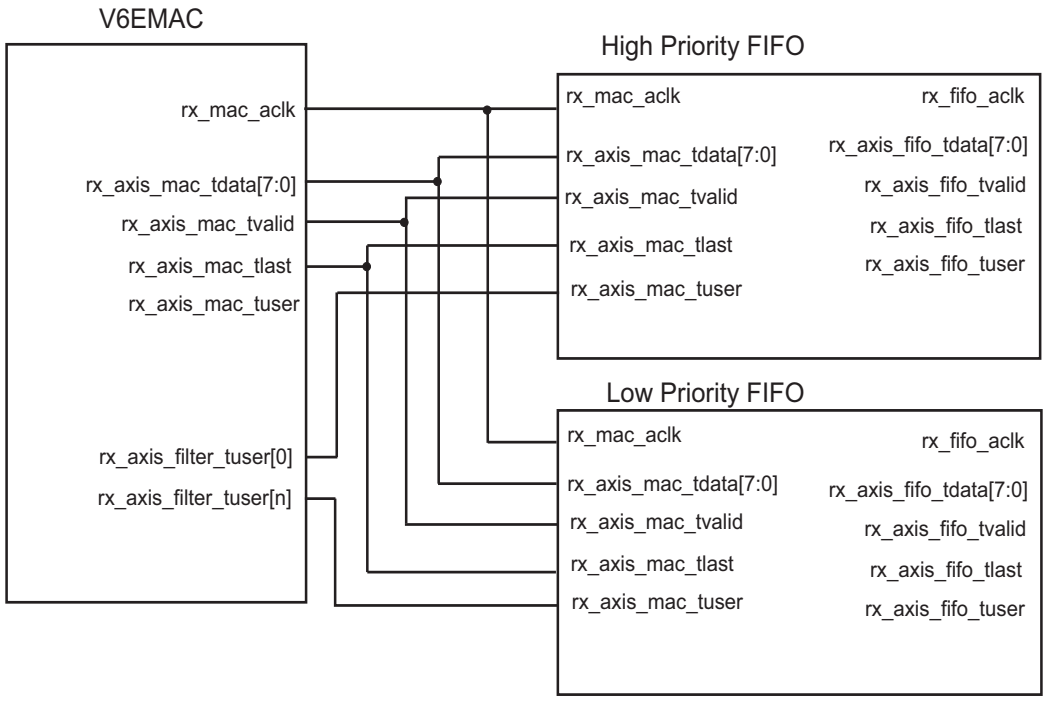

Figure 8-14: **Priority FIFO Connections**

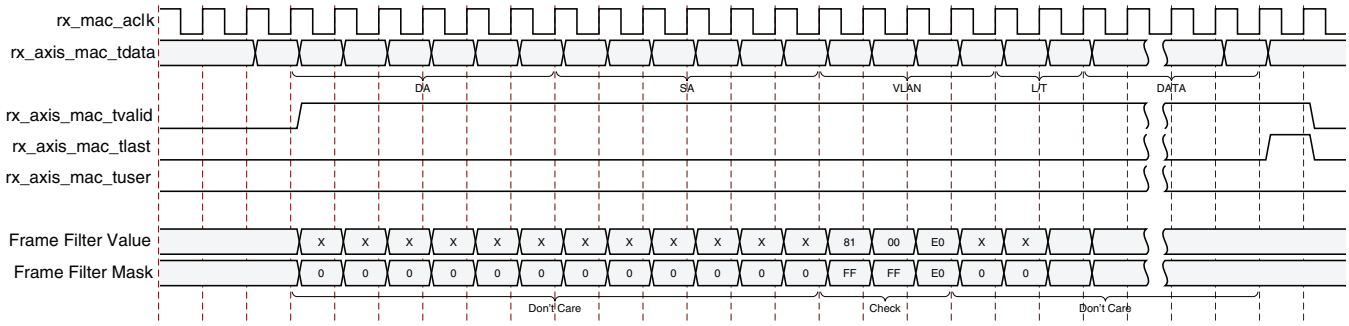

Figure 8-15: **Filtering of VLAN Frames with VLAN p of 7**

Address/Frame Filter with no Management Interface

When the Address/Frame filter is selected with no management interface only a subset of its functionality is available.

Because there is no user access to internal registers it is not possible to update the configurable Address/Frame filters, these are therefore not generated as part of the core. However, the basic Destination Address filtering is still available and enables the MAC to identify/filter the Broadcast address, a User supplied Pause/Unicast Address and the Special Pause Multicast Address.

In this configuration it is assumed that the User supplied Pause address is the same as the MAC Unicast address. A packet matching this filter is only treated as a pause frame if it meets all other criteria to identify a pause frame.

V6EMAC Configuration Settings

This section discusses unusual configuration options.

Half-Duplex Configuration Settings

When operating in MII/GMII or RGMII at 10 Mb/s or 100 Mb/s, the transmitter and receiver can be independently configured between full and half-duplex modes. This functionality is made available for full flexibility in unusual applications, for example, Ethernet protocol testers.

However, for legal and predictable behavior in Ethernet networks:

- Always configure transmitter and receiver duplex modes identically.

Half-Duplex and Flow Control Configuration Settings

The IEEE802.3 specification defines the flow control functionality only for full-duplex applications.

Configuration of the V6EMAC allows Flow Control functionality and Duplex mode to be configured independently. However, Flow Control is only enabled in full-duplex mode:

- When operating half-duplex mode, always disable Flow Control.
- When operating in full-duplex mode, Flow Control can optionally be enabled.

MAC Address Settings

When the core is generated with a Management Interface, the core contains a configurable Pause frame MAC Source Address (see MAC Configuration). This can also be set using the GUI regardless of the use of the management interface. This MAC Address is used by the flow control logic; received pause frames are matched against this address appearing in the Destination Address field; pause frames initiated by the core place this MAC Address into the Source Address field of a transmitted pause frame.

When the V6EMAC is generated with the optional Address/Frame filter, the core contains a configurable Unicast Address (see MAC Configuration). This address is used by the Address/Frame Filter to match against this address appearing in the Destination Address field of all received frames.

The core, for full flexibility, allows the Pause frame MAC Source Address and the Unicast Address (when present) to be configured independently. However, under standard network operating conditions:

- The Pause frame MAC Source Address should be set to the Unicast Address.

XILINX®

Chapter 9

Media Independent Interface (MII)

The Media Independent Interface (MII), defined in *IEEE Std 802.3-2008* [Ref 4], clause 22 is a parallel interface that connects a 10-Mb/s and/or 100-Mb/s capable MAC to the physical sublayers.

The Virtex®-6 FPGA Embedded Tri-Mode Ethernet MAC supports MII at 2.5 V only. Care must be taken to select a PHY that is electrically compatible with the Virtex-6 FPGA LVCMOS25 voltage standard. See the *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics* [Ref 8] for more information.

MII Transmitter Interface

The logic required to implement the MII transmitter logic is illustrated in Figure 9-1.

`mii_tx_clk` is provided by the external PHY device connected to the MII. As illustrated, this is placed onto global clock routing to provide the clock for all transmitter logic, both within the core and for the user side logic which connects to the TX AXI4-Stream I/F of the core. Alternatively the BUFG can be replaced with a BUFR. See the *Virtex-6 FPGA Configuration User Guide* [Ref 9] for BUFR usage guidelines. The frequency of this clock is 25 MHz at 100 Mb/s and 2.5 MHz at 10 Mb/s.

To match the user data rate, which uses an 8 bit datapath and the MII, which uses a 4 bit datapath, the TX AXI4-Stream interface is throttled, using `tx_axis_mac_tready`, under control of the MAC to limit data transfers to every other cycle.

Figure 9-1 also illustrates how to use the physical transmitter interface of the core to create an external MII. The signal names and logic shown in this figure exactly match those delivered with the example design. Figure 9-1 shows that the output transmitter signals are registered in device IOBs before driving them to the device pads.

MII Receiver Interface

The logic required to implement the MII receiver logic is also illustrated in Figure 9-1.

`mii_rx_clk` is provided by the external PHY device connected to the MII. As illustrated, this is placed onto global clock routing to provide the clock for all receiver logic, both within the core and for the user-side logic which connects to the RX AXI4-Stream I/F of the V6EMAC. Alternatively the BUFG can be replaced with a BUFR. See the *Virtex-6 FPGA Configuration User Guide* [Ref 9] for BUFR usage guidelines. The frequency of this clock is 25MHz at 100 Mb/s and 2.5 MHz at 10 Mb/s.

To match the user data rate, which uses an 8 bit datapath and the MII, which uses a 4 bit datapath, the RX AXI4-Stream interface is throttled, using `rx_axis_mac_tvalid`, under control of the MAC to limit data transfers to every other cycle.

Figure 9-1 also illustrates how to use the physical receiver interface of the core to create an external MII. The signal names and logic shown in this figure exactly match those delivered with the example design. Figure 9-1 shows that the input receiver signals are registered in device IOBs before routing them to the core.

Multiple Core Instances with the MII

Because both `mii_tx_clk` and `mii_rx_clk` are sourced by the external PHY device connected to the MII, it is not possible to share transmitter or receiver clock resources across multiple instantiations of the core. Each instance of the core requires its own independent clocking resources. Therefore the logic of Figure 9-1 must be duplicated for each instance of the core.

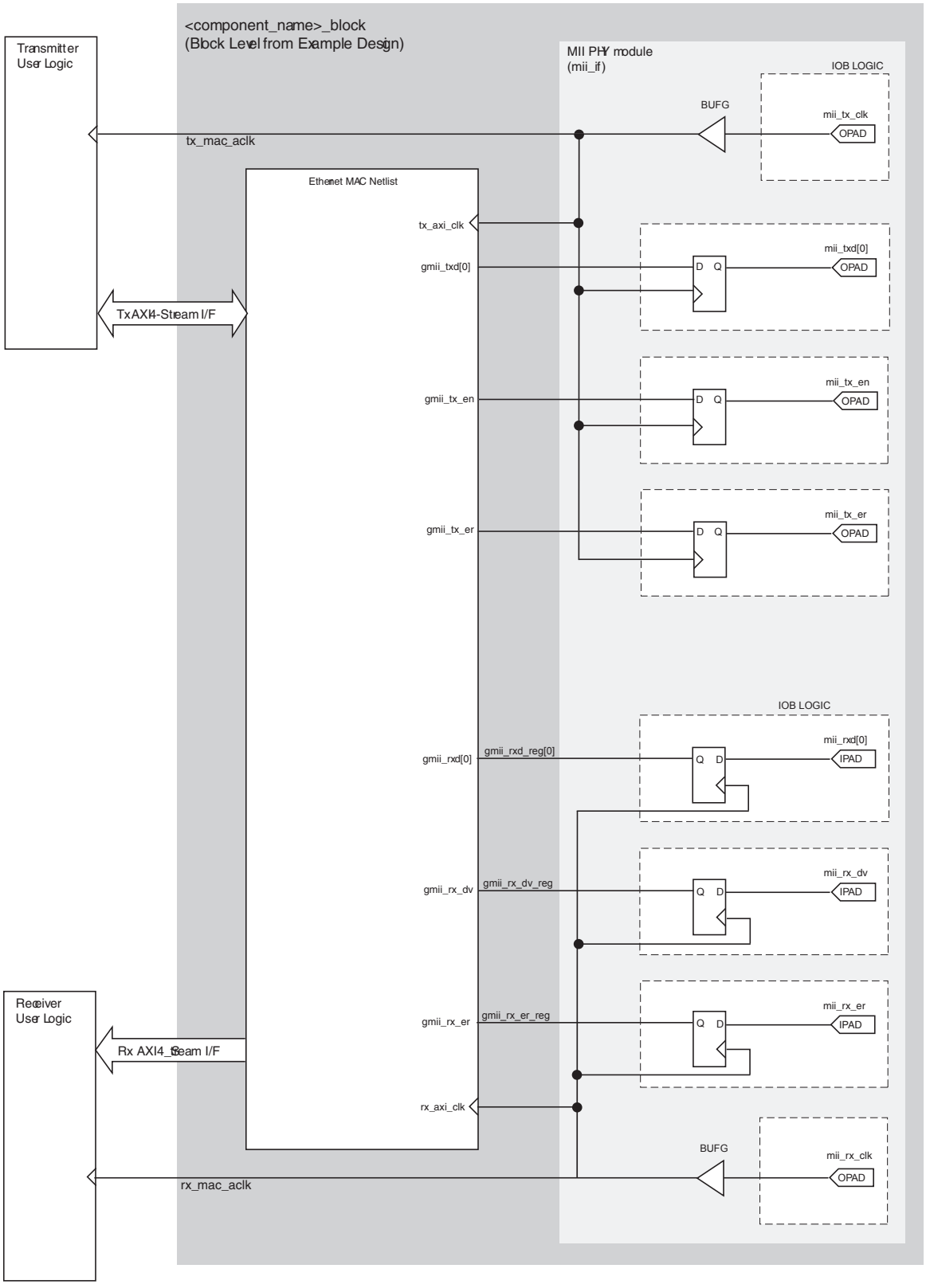

Figure 9-1: **MII Transmitter, Receiver and Clock Logic For All Devices**

XILINX®

Chapter 10

Gigabit Media Independent Interface (GMII)

The Media Independent Interface (MII), defined in *IEEE Std 802.3-2008* [Ref 4], clause 22, is a parallel interface that connects a 10-Mb/s and/or 100-Mb/s capable MAC to the physical sublayers. The Gigabit Media Independent Interface (GMII), defined in *IEEE Std 802.3-2008* [Ref 4], clause 35, is an extension of the MII used to connect a 1-Gb/s capable MAC to the physical sublayers. MII can be considered a subset of GMII, and as a result, GMII/MII can carry Ethernet traffic at 10 Mb/s, 100 Mb/s, and 1 Gb/s.

When GMII is selected it is possible to select either 10/100/1000 Mb/s or 1000 Mb/s. In either case the HDL example design provides

The Virtex®-6 FPGA Embedded Tri-Mode Ethernet MAC supports the GMII/MII physical interface at 2.5V only. Care must be taken to select a PHY that is electrically compatible with the Virtex-6 FPGA LVCMOS25 voltage standard. See the *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics* [Ref 8] for more information.

GMII Transmitter Interface

The logic required to implement the GMII transmitter logic is illustrated in Figure 10-1. `gtx_clk` is a user-supplied 125 MHz reference clock source for use at 1 Gb/s. `mii_tx_clk` is sourced by the external PHY device for use at 10 Mb/s and 100 Mb/s speeds. Consequently, for 10/100/1000 Mb/s support, a global clock multiplexer, a BUFGMUX, is used to switch the clock source depending on the operating speed. The output from this BUFGMUX provides the transmitter clock for the core and user logic as illustrated. If the core is generated with 1000 Mb/s only support then this BUFGMUX is replaced with a simple BUFG and the `mii_tx_clk` input can be ignored.

When operating at either 10Mb/s or 100 Mb/s, all user logic uses the AXI4-Stream interface handshaking to throttle the data to allow for the differing data widths between the 4-bit MII and the cores 8-bit user datapath.

Figure 10-1 also illustrates how to use the physical transmitter interface of the core to create an external GMII. The signal names and logic shown in this figure exactly match those delivered with the example design.

As shown in Figure 10-1, the output transmitter signals are registered in device IOBs before driving them to the device pads. The logic required to forward the transmitter clock for 1 Gb/s operation is also shown. This logic uses an IOB output Double-Data-Rate (DDR) register so that the clock signal produced incurs exactly the same delay as the data and control signals. This clock signal, `gmii_tx_clk`, is inverted with respect to `gtx_clk` so that the rising edge of `gmii_tx_clk` occurs in the centre of the data valid window, therefore maximizing setup and hold times across the interface.

The half-duplex signals `gmii_col` and `gmii_crs` are asynchronous to the transmit clock. These are routed through PADs and IOBs and then input to the core.

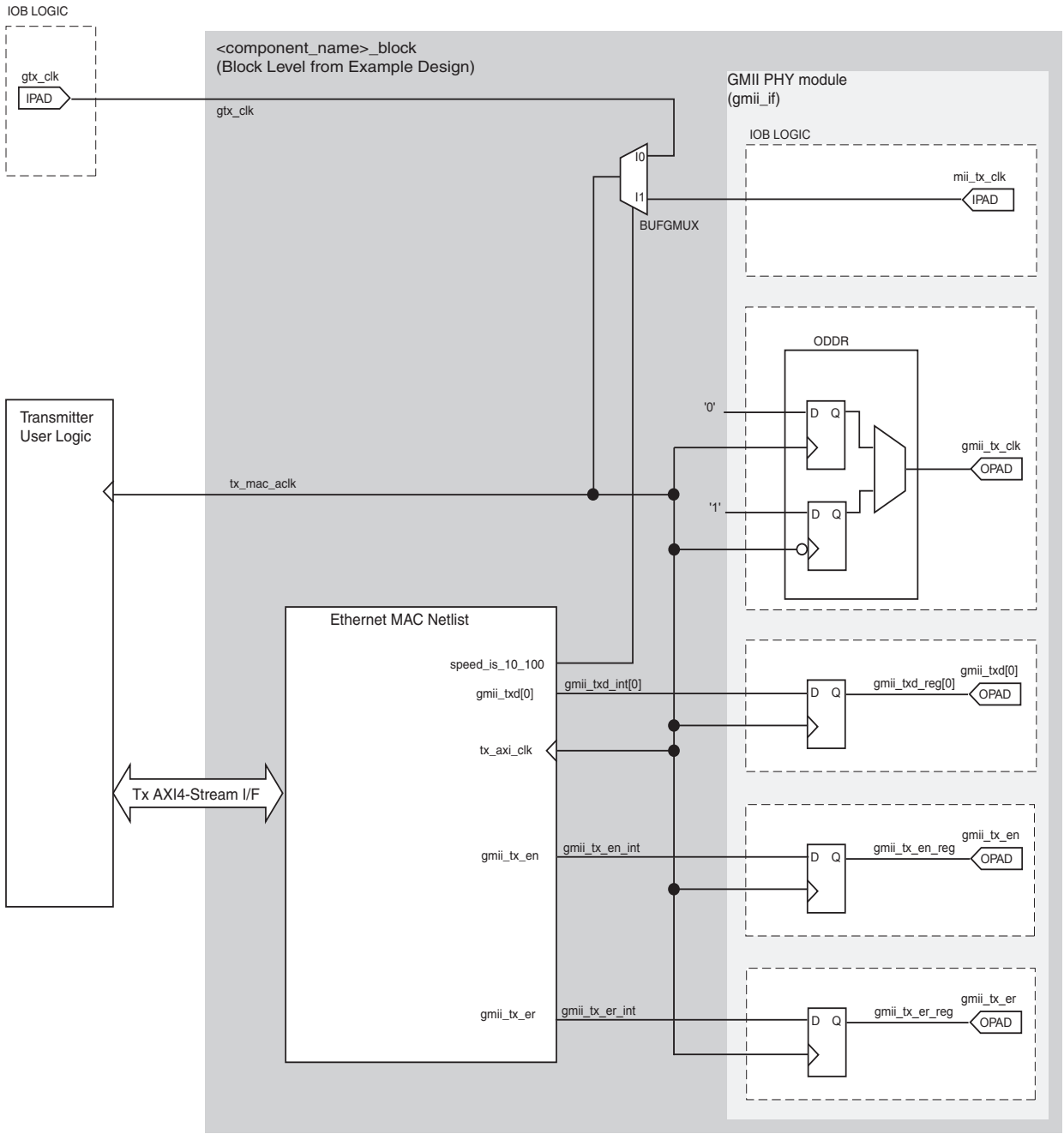

Figure 10-1: **GMII Transmitter Logic and Clock Logic**

GMII Receive Interface

In this implementation, a BUFIO is used to provide the lowest form of clock routing delay from input clock to input GMII Rx signal sampling at the device IOBs. Note, however, that this creates placement constraints: a BUFIO capable clock input pin must be selected, and all other input GMII Rx signals must be placed in the respective BUFIO region. For more information, see I/O Location Constraints

The input clock is also placed onto regional clock routing using the BUFR component as illustrated. This regional clock then provides the clock for all receiver logic, both within the core and for the user side logic which connects to the receiver AXI4-Stream I/F of the core.

Note: It is important to ensure that the TEMAC_SINGLE hard block is reachable by the BUFR net.

The IODELAY elements can be adjusted to fine-tune the setup and hold times at the GMII IOB input flip-flops. The delay is applied to the IODELAY element using constraints in the UCF; these can be edited if desired. See Chapter 14, Constraining the Core for more information.

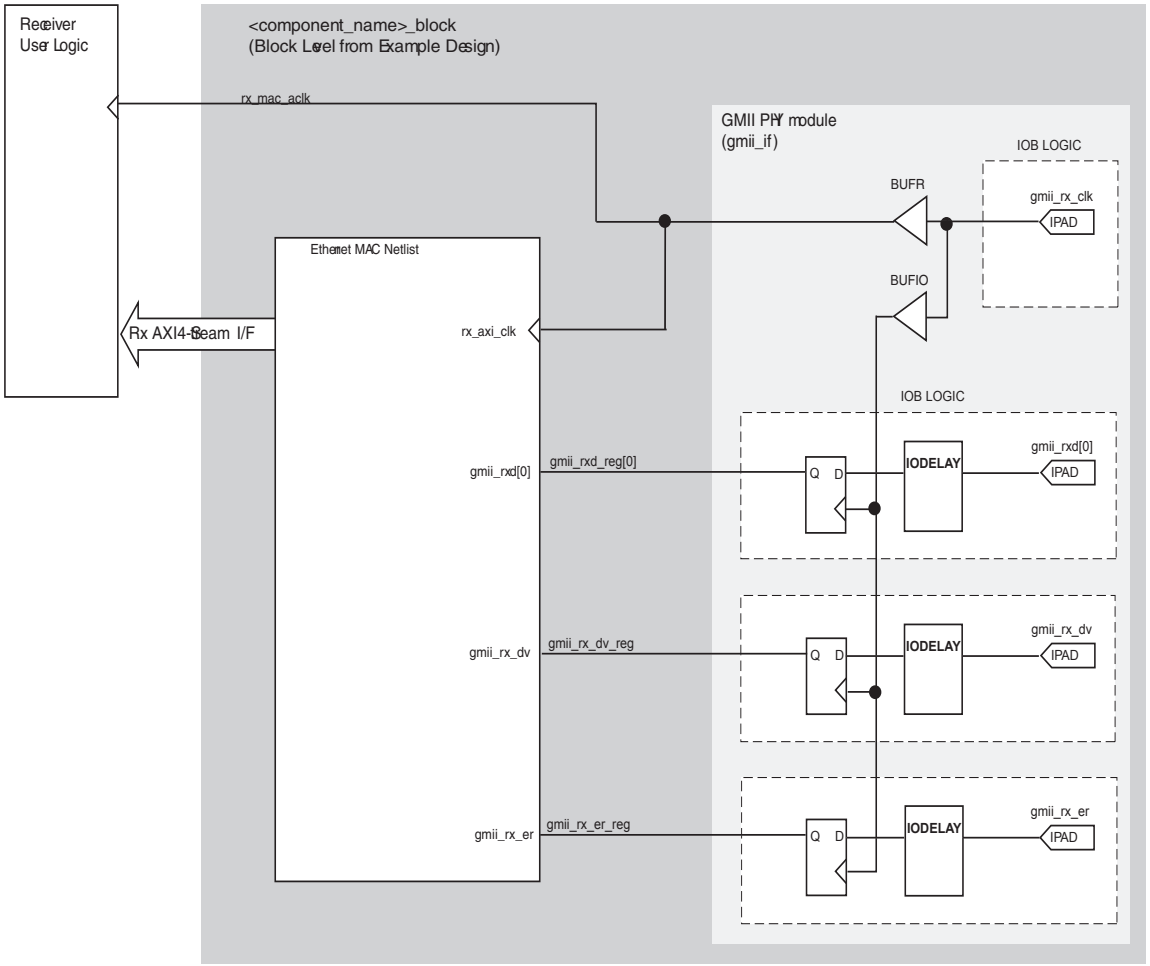

Figure 10-2: **GMII Receiver Logic and Clock Logic**

Clock Sharing across Multiple Cores

When multiple instances of the core are instantiated in a design, transmitter clock resources can be shared across all core instances; receiver clock resources cannot be shared and are independent for each core instance.

Figure 10-3 illustrates clock resource sharing across multiple instantiations of the core when using GMII at 1 Gb/s. For all instantiations, `gtx_clk` can be shared between multiple cores, resulting in a common clock domain across the device.

Figure 10-4 illustrates multiple instances of the core when 10/100/1000 Mb/s is selected. The 1 Gb/s transmitter reference clock source, `gtx_clk`, can be shared across all cores as illustrated. However, global transmitter and receiver clock resources cannot be shared and require independent BUFGMUX/BUFR elements as illustrated.

In both cases the receiver clocks cannot be shared. Each core is provided with its own local version of `gmii_rx_clk` from the connected external PHY device as illustrated.

Figure 10-3 and Figure 10-4 illustrate only two cores. However, more can be added using the same principal. This is done by instantiating the cores using the block level (from the example design) and sharing `gtx_clk` across all instantiations. The receiver clock, which cannot be shared, is unique for every instance of the core.

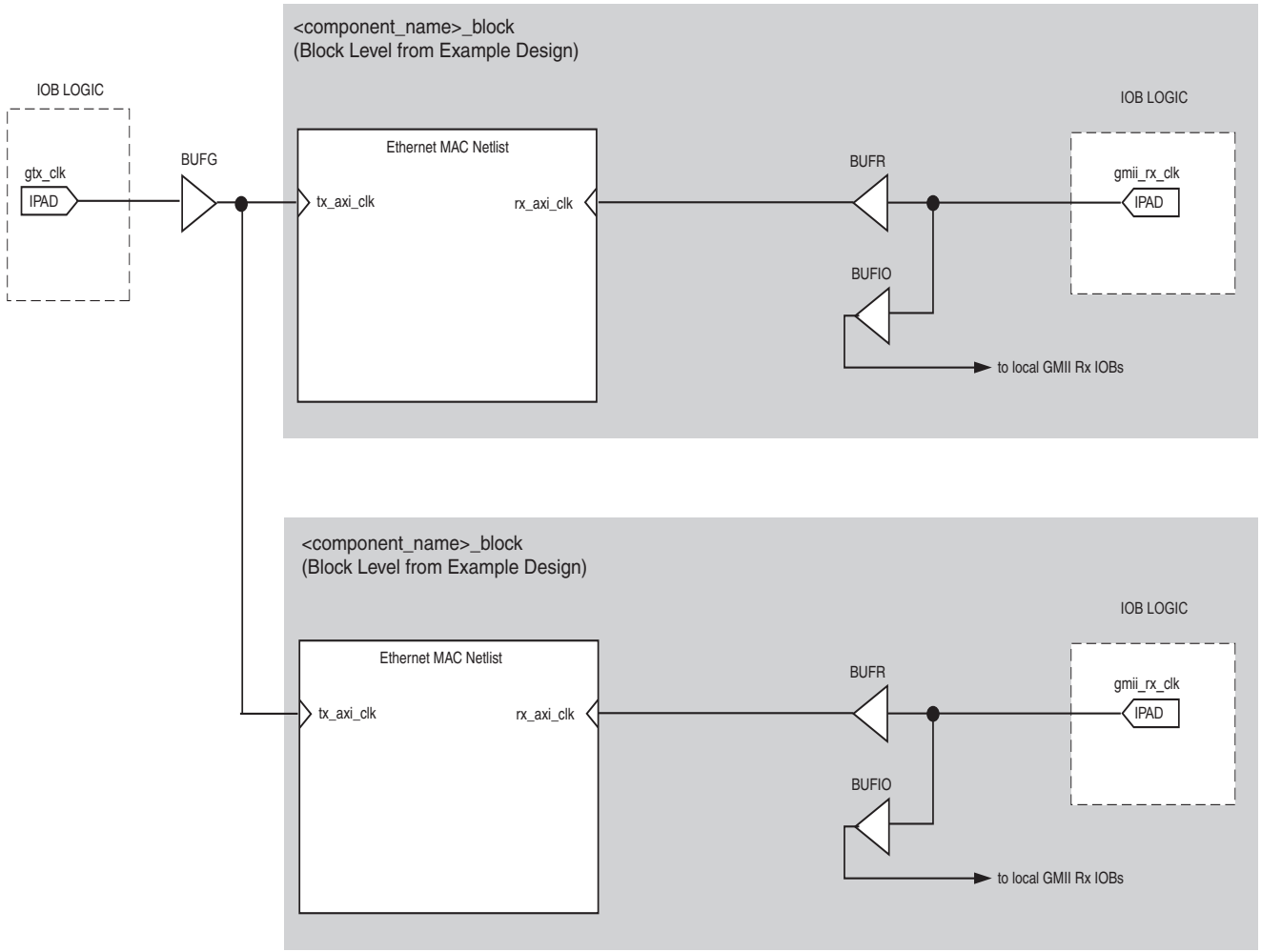

Figure 10-3: **Clock Resource Sharing for 1 Gb/s GMII**

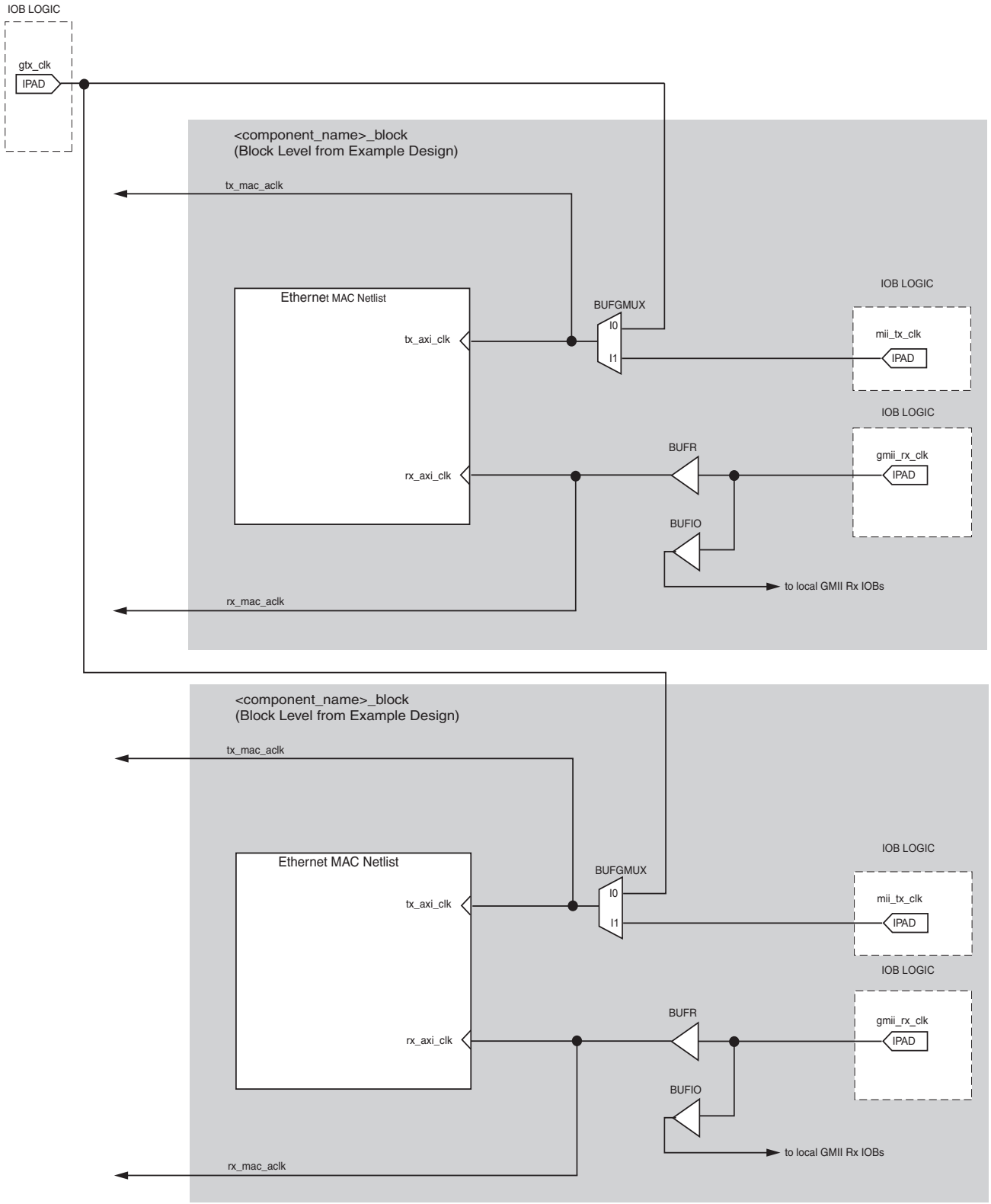

Figure 10-4: **Clock Resource Sharing for 10/100/1000 Mb/s GMII**

XILINX®

Chapter 11

Reduced Gigabit Media Independent Interface (RGMII)

The Reduced Gigabit Media Independent Interface (RGMII) is an alternative to the GMII/MII. RGMII can carry Ethernet traffic at 10 Mb/s, 100 Mb/s and 1 Gb/s and achieves a 50% reduction in the pin count compared with GMII; this is achieved with the use of double-data-rate (DDR) flip-flops. RGMII is therefore often favored over GMII by PCB designers.

Transmitter Logic

The logic required to implement the RGMII transmitter logic is illustrated in Figure 11-1. `gtx_clk` is a user-supplied 125 MHz reference clock source which is routed, through an IOB buffer, to the `gtx_clk` input of the V6EMAC. The V6EMAC drives a clock output, `tx_axi_clk_out`, which is placed onto global clock routing to provide the clock for all transmitter logic, both within the core and for the user side logic which connects to the transmitter AXI4-Stream I/F of the V6EMAC.

For RGMII, this `tx_axi_clk_out` is used to clock transmitter logic at all three Ethernet speeds. The data rate difference between the three speeds is compensated for as all User logic uses the AXI4-Stream interfaces built in handshaking to throttle the data appropriately, under control of the MAC Netlist. At 1 Gb/s, all external logic is clocked at 125 MHz. At 100 Mb/s and 10 Mb/s, the logic is clocked at 25 MHz and 2.5 MHz, respectively.

Figure 11-1 illustrates how to use the physical transmitter interface of the core to create an external RGMII. The signal names and logic shown in this figure exactly match those delivered with the example design. Figure 11-1 shows that the output transmitter signals are registered in device IOBs, using DDR registers, before driving them to the device pads.

The logic required to forward the transmitter clock is also shown. This logic uses an IOB output Double-Data-Rate (DDR) register so that the clock signal produced incurs exactly the same delay as the data and control signals. However, the clock signal is then routed though an output delay element (IODELAY) before connecting to the device pad. The result of this is to create a 2 ns delay, which places the `rgmii_txc` forwarded clock in the centre of the data valid window for forwarded RGMII data and control signals when operating at 1 Gb/s Ethernet speed.

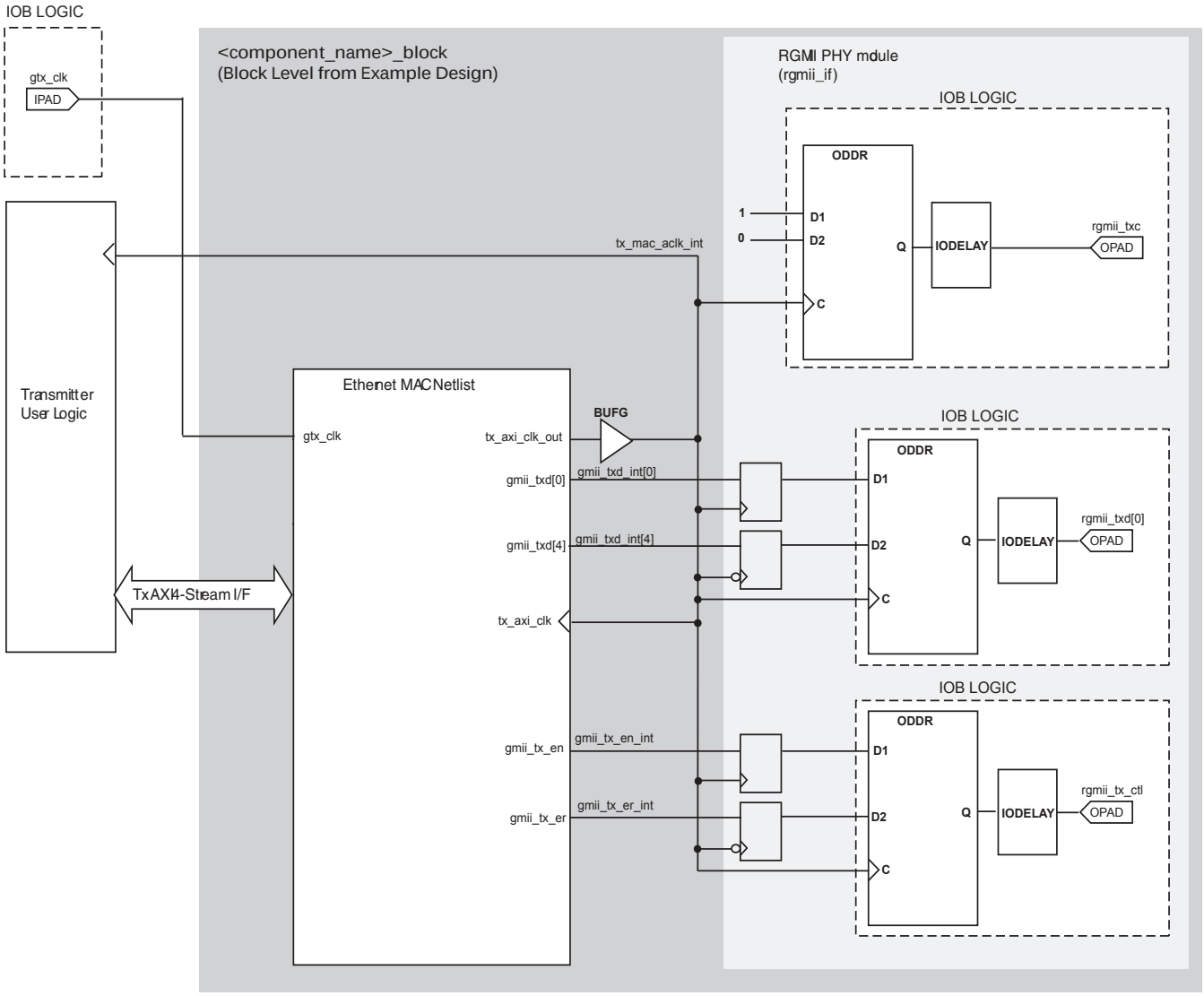

Figure 11-1: **RGMII Transmitter Logic and Clock Logic**

Receiver Logic

In this implementation, a BUFIO is used to provide the lowest form of clock routing delay from input clock to input RGMII Rx signal sampling at the device IOBs. Note, however, that this creates placement constraints: a BUFIO capable clock input pin must be selected, and all other input RGMII Rx signals must be placed in the respective BUFIO region. For more information, see I/O Location Constraints in Chapter 14.

The input clock is also placed onto regional clock routing using the BUFR component as illustrated. This regional clock then provides the clock for all receiver logic, both within the core and for the user side logic which connects to the receiver AXI4-Stream I/F of the core.

Note: It is important to ensure that the TEMAC_SINGLE hard block is reachable by the BUFR net.

The IODELAY elements can be adjusted to fine-tune the setup and hold times at the RGMII IOB input flip-flops. This meets input setup and hold constraints at all three Ethernet speeds. The delay is applied to the IODELAY element using constraints in the UCF; these can be edited if desired. See Chapter 14, Constraining the Core for more information.

All user logic uses the AXI4-Stream interface handshaking to throttle the data as required at the different speeds.

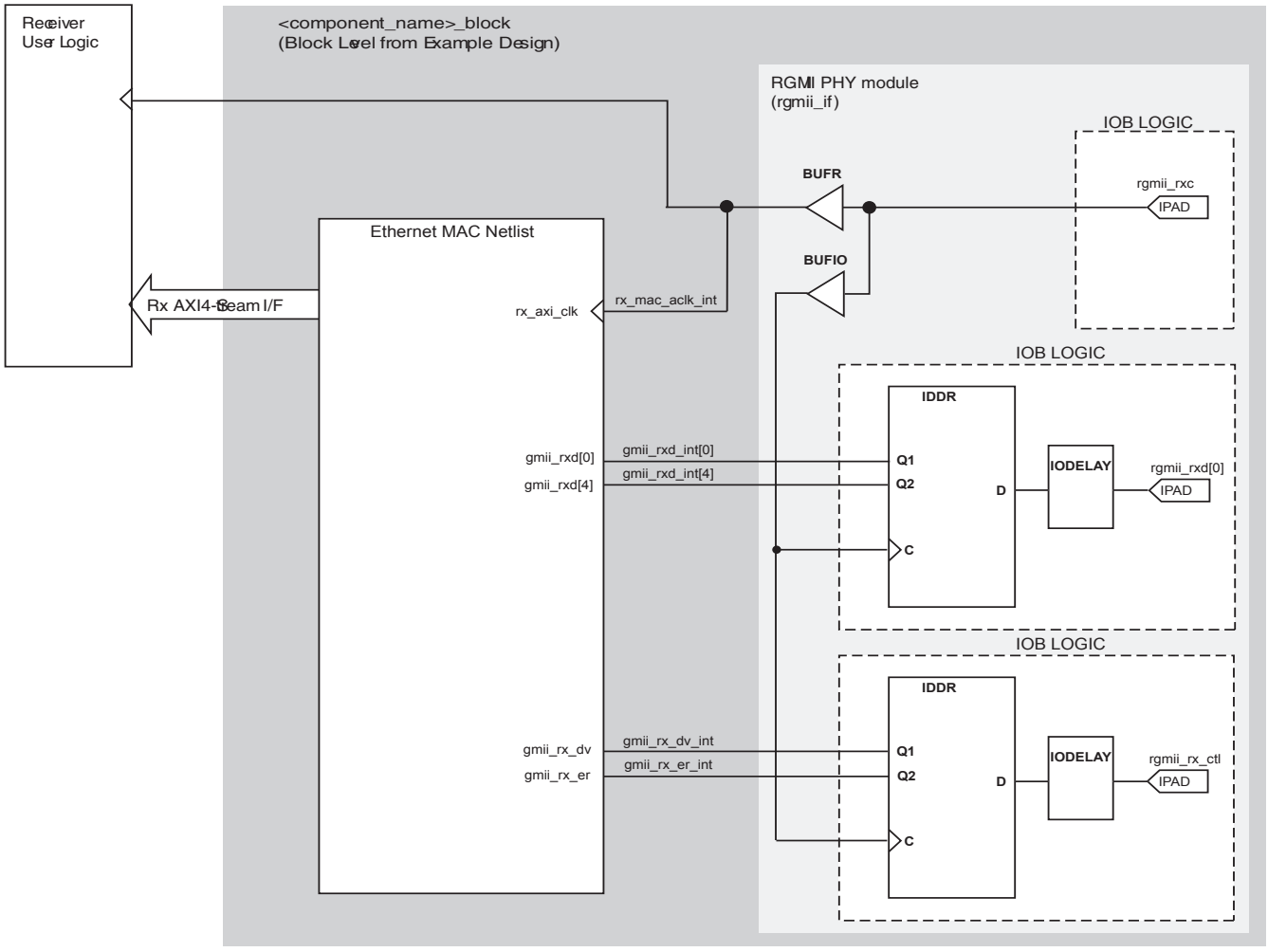

Figure 11-2: **RGMII Receiver Logic and Clock Logic**

Clock Resource Sharing

For all instantiations, `gtx_clk` can be shared between multiple cores; however, because the clock used for the transmit logic for each core, tx_mac_aclk_int, is output by the V6EMAC netlist, it is instance specific and cannot be shared.

The receiver clocks cannot be shared. Each core is provided with its own local version of `rgmii_rxc` from the connected external PHY device.

Chapter 12

Serial Gigabit Media Independent Interface (SGMII)

Ethernet MAC PCS/PMA Sublayer

Figure 12-1 shows the functional block diagram of the V6EMAC with the PCS/PMA sublayer block highlighted.

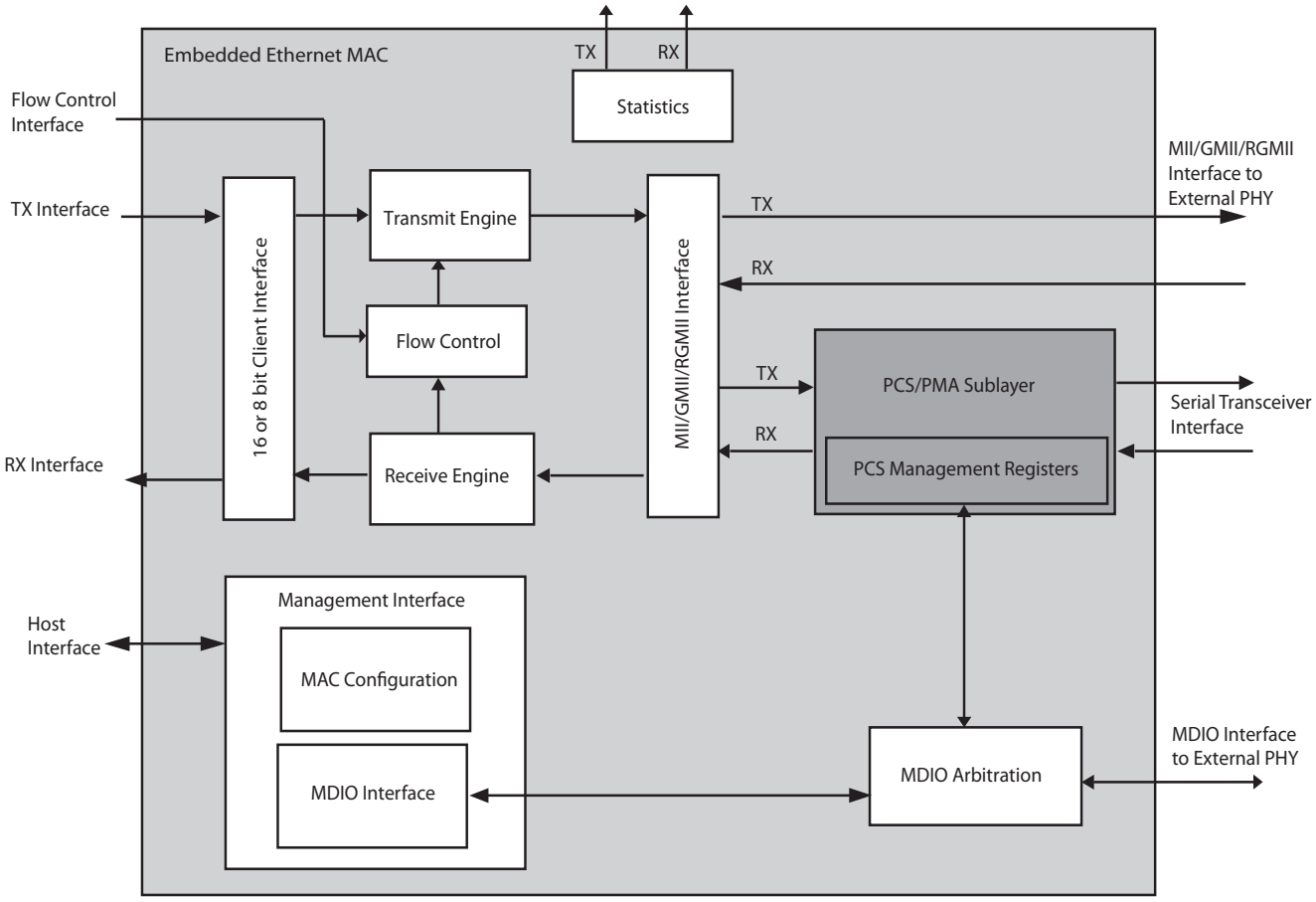

Figure 12-1: **PCS/PMA Sublayer Block Diagram in the V6EMAC**

The PCS/PMA sublayer extends the functionality of the V6EMAC, replacing the GMII/MII or RGMII parallel interfaces with an interface to a serial transceiver. The connected serial transceiver then provides the remaining functionality to accommodate the following two standards, both of which require the high-speed serial interface provided by the serial transceiver:

- Serial Gigabit Media Independent Interface (SGMII)
- 1000BASE-X PCS/PMA (GPCS)

Introduction to the SGMII Implementation

The Serial GMII (SGMII) is an alternative interface to the GMII/MII that converts the parallel interface of the GMII/MII into a serial format that is capable of carrying traffic at speeds of 10 Mb/s, 100 Mb/s, and 1 Gb/s. It radically reduces the I/O count and is, therefore, often favored by PCB designers.

The SGMII physical interface was defined by Cisco Systems. The data signals operate at a rate of 1.25 Gb/s. Due to the speed of these signals, differential pairs are used to provide signal integrity and minimize noise.

The source-synchronous clock signals defined in the specification are not implemented in the V6EMAC. Instead, the serial transceiver is used to transmit and receive the differential data at the required rate using clock data recovery (CDR). For more information on SGMII, see *Serial-GMII Specification (CISCO SYSTEMS, ENG-46158)* [Ref 6]. Figure 12-2 shows an expanded diagram of the PCS/PMA sublayer block. The V6EMAC is connected to a serial transceiver, which in turn connects to a Tri-Speed Ethernet BASE-T PHY device through the SGMII.

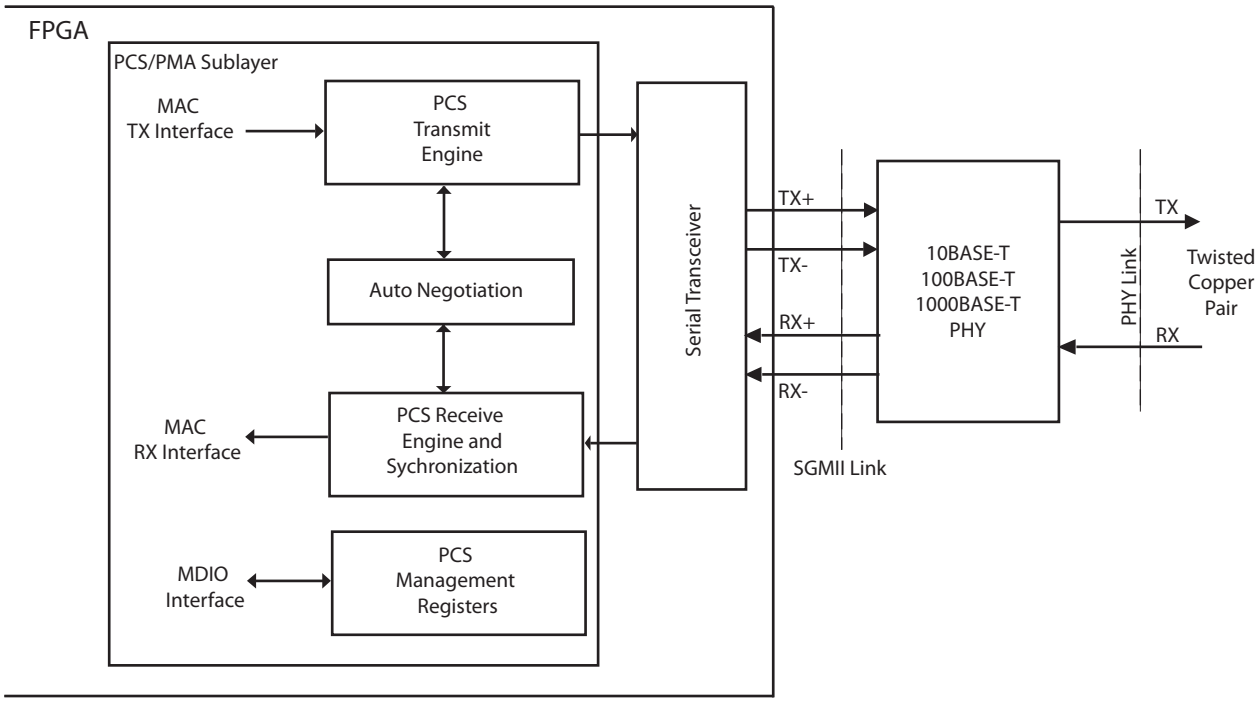

Figure 12-2: **Ethernet PCS/PMA Sublayer Extension**

The PCS/PMA sublayer contains its own functional blocks, which are illustrated and summarized in the following subsections.

PCS Transmit Engine

The PCS Transmit Engine encodes the data stream received from the V6EMAC into a sequence of ordered sets. The data is then transmitted to the serial transceiver, using an 8-bit datapath, which then provides 8B/10B encoding of this data and parallel to serial conversion. The output from the serial transceiver is a differential pair operating at a rate of 1.25 Gb/s.

PCS Receive Engine and Synchronization

The serial transceiver receives a serial differential pair operating at a rate of 1.25 Gb/s from the SGMII PHY. The serial transceiver, using CDR, performs word alignment on this serial data stream and then converts this to a parallel interface. This data is then 8B/10B decoded by the serial transceiver before presenting this to the PCS/PMA sublayer using an 8-bit datapath.

Within the PCS/PMA sublayer, the synchronization process performs analysis of valid/invalid received sequence ordered sets to determine if the link is in good order. The PCS receive engine then decodes these sequence ordered sets into a format that can then be presented to the standard receiver datapath within the V6EMAC.

Auto-Negotiation

The SGMII auto-negotiation function allows the V6EMAC to communicate with the attached PHY device. Auto-negotiation is controlled and monitored using a software function through the PCS Management Registers. See SGMII Auto-Negotiation.

PCS Management Registers

Configuration and status of the PCS/PMA sublayer, including access to and from the auto-negotiation function, is through the PCS Management Registers. These registers are accessed through the serial MDIO, see MDIO Interface.

For the SGMII standard, the configuration and status registers available are listed in SGMII Management Registers.

SGMII RX Elastic Buffer

The RX elastic buffer can be implemented in one of two ways:

- Using the buffer present in the serial transceivers
- Using a larger buffer that is implemented in the FPGA logic

This section describes the selection and implementation of two options in the following subsections:

Serial Transceiver Logic Using the RX Elastic Buffer

Serial Transceiver Logic Using the RX Elastic Buffer in FPGA Logic

RX Elastic Buffer Implementations

Selecting the Buffer Implementation from the GUI

The GUI for the V6EMAC provides two options under the heading of SGMII Capabilities. These options are:

- Option 1: 10/100/1000 Mb/s clock tolerance compliant with Ethernet specification

  For this option, if the FPGA and PHY clocks are independent and each has 100 ppm tolerance, an additional elastic buffer is required (100 slices and 1 block RAM per V6EMAC).

- Option 2: 10/100/1000 Mb/s (restricted tolerance for clocks) OR 100/1000 Mb/s

  For this option, if 10 Mb/s is not required OR it is required but the FPGA and PHY clocks are closely related (see the *Virtex-6 GTX Transceivers User Guide* [Ref 11]), the RX buffer can be used (saves 100 slices and 1 block RAM per V6EMAC).

Option 1, the default mode, provides the implementation using the RX elastic buffer in the FPGA logic. This alternative RX elastic buffer utilizes a single block RAM to create a buffer that is twice as large as the one present in the serial transceiver, therefore consuming extra logic resources. However, this default mode provides reliable SGMII operation.

Option 2 uses the RX elastic buffer in the serial transceivers. This buffer, which is half the size of the buffer in Option 1, can potentially underflow or overflow during SGMII frame reception at 10 Mb/s operation (see The FPGA RX Elastic Buffer Requirement). However, in logical implementations where this case is proven reliable, this option is preferred because of its lower logic utilization.

The FPGA RX Elastic Buffer Requirement

Figure 12-3 illustrates a simplified diagram of a common situation where the V6EMAC in SGMII mode is interfaced to an external PHY device. Separate oscillator sources are used for the FPGA and the external PHY. *IEEE Std 802.3-2008* [Ref 4] uses clock sources with a 100 ppm tolerance. In Figure 12-3, the clock source for the PHY is slightly faster than the clock source to the FPGA. Therefore, during frame reception, the RX elastic buffer (implemented in the serial transceiver in this example) starts to fill up.

Following frame reception in the interframe gap period, idles are removed from the received data stream to return the RX elastic buffer to half-full occupancy. This task is performed by the clock correction circuitry (see the *Virtex-6 GTX Transceivers User Guide* [Ref 11]).

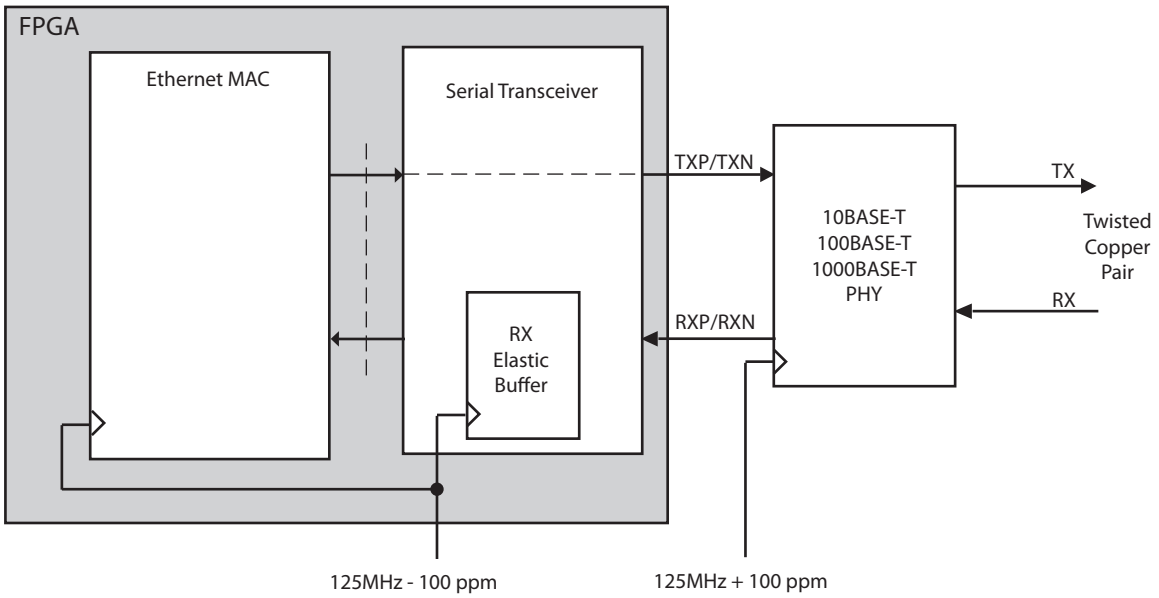

Figure 12-3: **SGMII Implementation using separate clock sources**

Assuming separate clock sources, each with 100 ppm tolerance, the maximum frequency difference between the two devices can be 200 ppm, which translates into a full clock period difference every 5000 clock periods.

Relating this information to an Ethernet frame, there is a single byte of difference every 5000 bytes of received frame data, causing the RX elastic buffer to either fill or empty by an occupancy of one.

The maximum Ethernet frame (non-jumbo) has a 1522-byte size for a VLAN frame:

- At 1 Gb/s operation, this size translates into 1522 clock cycles
- At 100 Mb/s operation, this size translates into 15220 clock cycles (because each byte is repeated 10 times
- At 10 Mb/s operation, this size translates into 152200 clock cycles (because each byte is repeated 100 times)

Considering the 10 Mb/s case, 152200/5000 = 31 FIFO entries are needed in the elastic buffer above and below the halfway point to ensure that the buffer does not underflow or overflow during frame reception. This assumes that frame reception begins when the buffer is exactly half full.

Serial Transceiver Elastic Buffer

The RX elastic buffer in the serial transceivers has 64 entries. However, the buffer cannot be assumed to be exactly half full at the start of frame reception. Additionally, the underflow and overflow thresholds are not exact.

Figure 6-26 illustrates the elastic buffer depths and thresholds of serial transceivers in Virtex®-6 devices. Each FIFO word corresponds to a single character of data (equivalent to a single byte of data following 8B/10B decoding).

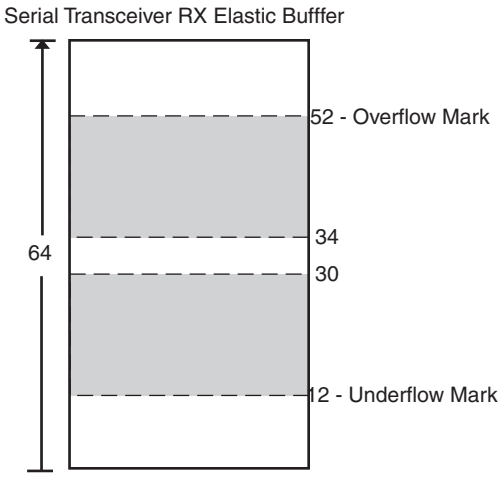

Figure 12-4: **Elastic Buffer Size for Serial Transceivers**

The shaded area represents the usable buffer for the duration of frame reception.

- If the buffer is filling during frame reception, then 52 – 34 = 18 FIFO locations are available before the buffer hits the overflow mark.
- If the buffer is emptying during reception, then 30 – 12 = 18 FIFO locations are available before the buffer hits the underflow mark.

This analysis assumes that the buffer is approximately at the half-full level at the start of the frame reception. There are, as illustrated, two locations of uncertainty above and below the exact half-full mark of 32. The uncertainty is a result of the clock correction decision, which is performed in a different clock domain.

Because there is a worst case scenario of one clock edge difference every 5,000 clock periods, the maximum number of clock cycles (bytes) that can exist in a single frame passing through the buffer before an error occurs is 5,000 x 18 = 90,000 bytes.

Table 12-1 translates this into maximum frame lengths at different V6EMAC speeds. The situation is worse at SGMII speeds lower than 1 Gb/s because bytes are repeated multiple times.

Table 12-1: **Maximum Frame sizes for Serial Transceivers RX Elastic buffers (100 ppm Clock Tolerance)**

Standard	Speed	Maximum Frame size
SGMII	1 Gb/s	90,000
SGMII	100 Mb/s	9,000
SGMII	10 Mb/s	900

FPGA Logic Elastic Buffer

To achieve reliable SGMII operation at 10 Mb/s (non-jumbo frames), the example design provides an RX elastic buffer implemented in the FPGA logic. It is twice the size of the serial transceiver's RX elastic buffer, and nominally provides 64 entries above and below the half-full threshold. This configuration can manage standard (non-jumbo) Ethernet frames at all three SGMII speeds.

Figure 12-5 illustrates alternative FPGA logic RX elastic buffer depth and thresholds. Each FIFO word corresponds to a single character of data (equivalent to a single byte of data following 8B/10B decoding). This buffer can optionally be used to replace the RX elastic buffers of the serial transceiver. See Serial Transceiver Logic Using the RX Elastic Buffer in FPGA Logic.

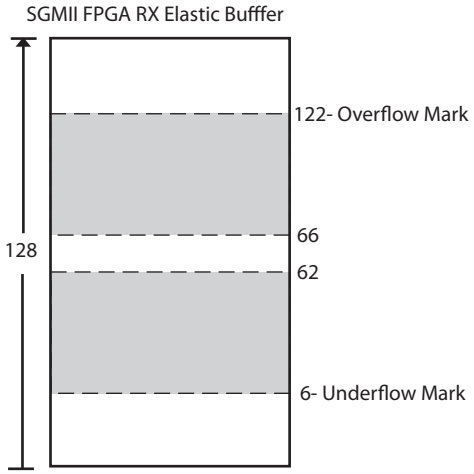

Figure 12-5: **Elastic Buffer Size of FPGA Logic Buffer**

The shaded area in Figure 12-5 represents the usable buffer availability for the duration of frame reception.

- If the buffer is filling during frame reception, then 122 – 66 = 56 FIFO locations are available before the buffer hits the overflow mark.
- If the buffer is emptying during reception, then 62 – 6 = 56 FIFO locations are available before the buffer hits the underflow mark.

This analysis assumes that the buffer is approximately at the half-full level at the start of the frame reception. As illustrated in Figure 12-5, there are two locations of uncertainty above and below the exact half-full mark of 64 as a result of the clock correction decision, which is based across an asynchronous boundary.

Because there is a worst-case scenario of one clock edge difference every 5,000 clock periods, the maximum number of clock cycles (bytes) that can exist in a single frame passing through the buffer before an error occurs is 5,000 x 56 = 280,000 bytes.

Table 12-2 translates this into maximum frame lengths at different V6EMAC speeds. The situation is worse at SGMII speeds lower than 1 Gb/s because bytes are repeated multiple times.

Table 12-2: **Maximum Frame Sizes for FPGA Logic buffers (100 ppm Clock Tolerance)**

Standard	Speed	Maximum Frame size
SGMII	1 Gb/s	280,000
SGMII	100 Mb/s	28,000
SGMII	10 Mb/s	2,800

Using the Serial Transceiver RX Elastic Buffer

The RX elastic buffer implemented in the serial transceiver can be used reliably when:

- 10 Mb/s operation is not required. Both 1 Gb/s and 100 Mb/s operate on standard Ethernet MAC frame sizes only.
- When the clocks are closely related (see Closely Related Clock Sources).

Designers are recommended to select the FPGA Logic RX elastic buffer implementation if they have any doubts to reliable operation.

Closely Related Clock Sources

Two cases are described with closely related clocks in SGMII mode:

- Case 1

  Figure 12-6 illustrates a simplified diagram of a common situation where the V6EMAC in SGMII mode is interfaced to an external PHY device. A common oscillator source is used for both the FPGA and the external PHY.

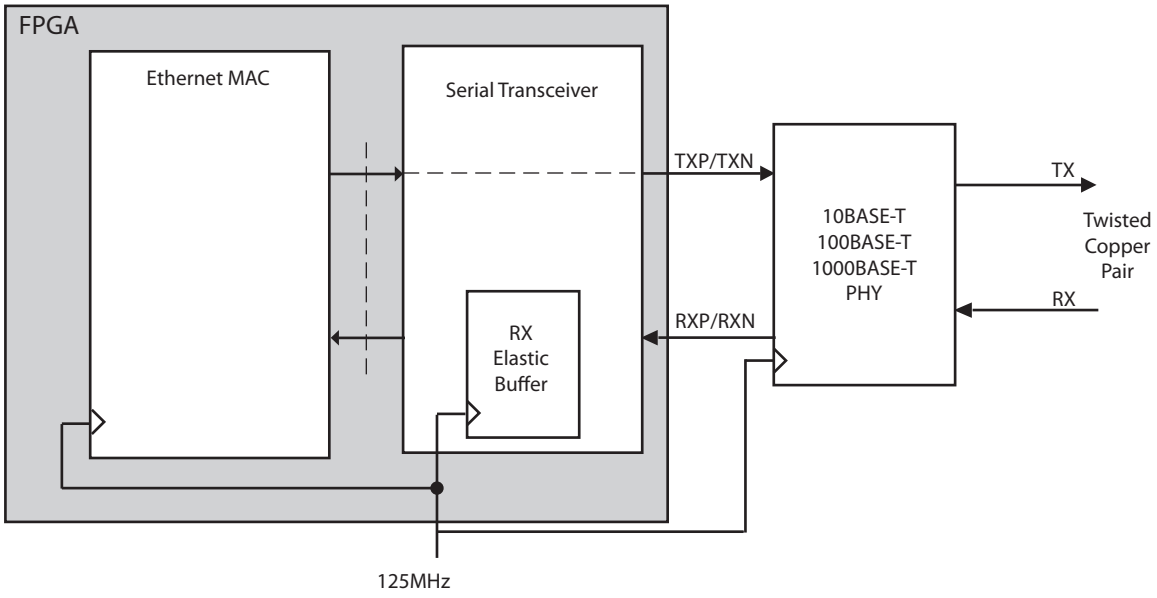

Figure 12-6: **SGMII Implementation Using Shared Clock Sources**

If the PHY device sources the receiver SGMII stream synchronously from the shared oscillator (see the PHY data sheet), the serial transceiver receives data at exactly the same rate as that used by the core. That is, the RX elastic buffer neither empties nor fills because the same frequency clock is on either side.

In this situation, the RX elastic buffer does not underflow or overflow, and the RX elastic buffer implementation in the serial transceiver is recommended to save logic resources.

- Case 2

  Using the case illustrated by Figure 12-6, assume that both clock sources used are 50 ppm. The maximum frequency difference between the two devices is 100 ppm, translating into a full clock period difference every 10,000 clock periods and resulting in a requirement for 16 FIFO entries above and below the half-full point. This case provides reliable operation with the serial transceiver RX elastic buffers. However, the designer must check the PHY data sheet to ensure that the PHY device sources the receiver SGMII stream synchronously to its reference oscillator.

Using the FPGA Logic Elastic Buffer

Figure 12-7 illustrates a simplified diagram of a situation where the V6EMAC in SGMII mode is interfaced to an external PHY device with an independent clock. The larger FPGA elastic buffer is used to cross clock domains, rather than the serial transceiver's elastic buffer.

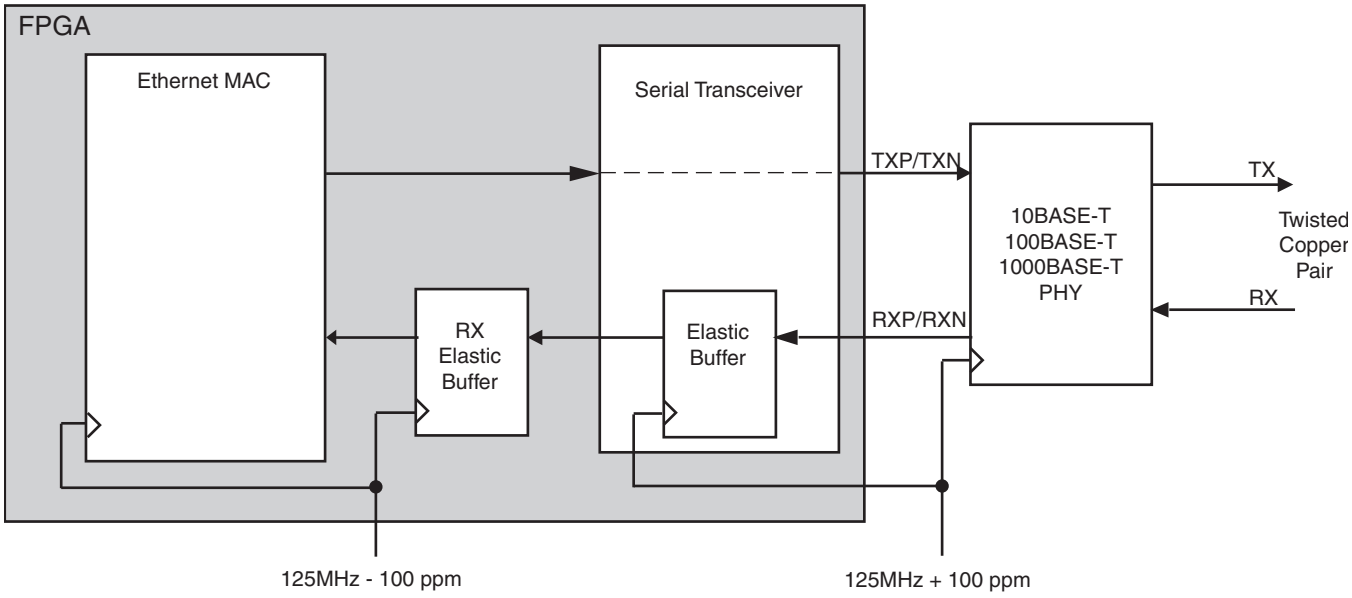

Figure 12-7: **SGMII Implementation Using a Logic Buffer**

Using the SGMII in this configuration eliminates the possibility of buffer error if the clocks are not tightly controlled enough to use the serial transceiver elastic buffer.

Serial Transceiver Logic Using the RX Elastic Buffer

Figure 12-8 shows the V6EMAC configured with SGMII as the physical interface. Connections to the Virtex-6 FPGA serial transceiver are illustrated.

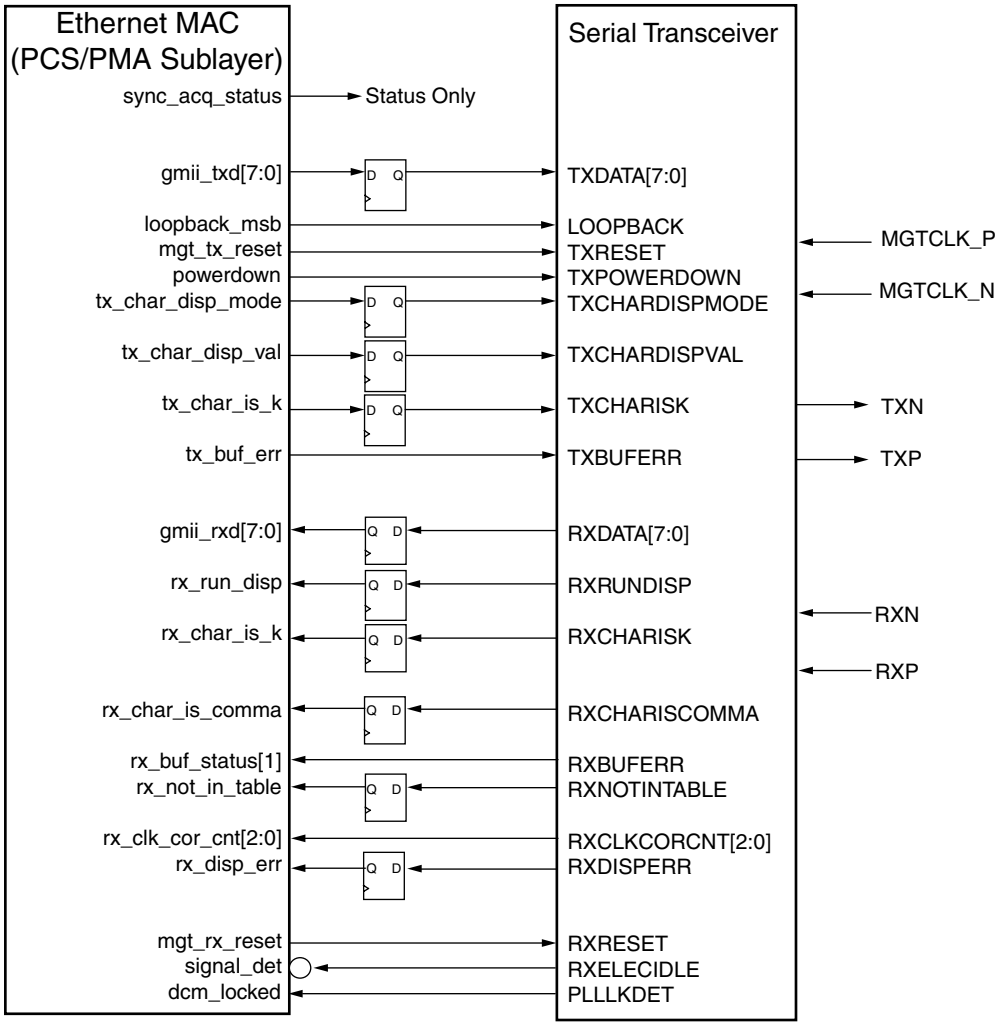

Figure 12-8: **V6EMAC Configured in SGMII Mode**

Serial Transceiver Logic Using the RX Elastic Buffer in FPGA Logic

The example design, delivered with the core in the CORE Generator™ tool, is split between two different hierarchical layers. The block level is designed so that it can be instantiated directly into customer designs. It provides the following functionality:

- Instantiates the core from HDL
- Connects the physical-side interface of the core to a Virtex-6 FPGA serial transceiver through the RX elastic buffer in FPGA logic

The V6EMAC is designed to integrate with the Virtex-6 FPGA serial transceiver. The connections and logic required between the V6EMAC and serial transceiver are illustrated in Figure 12-9. The signal names and logic shown in Figure 12-9 exactly match those delivered with the example design when the serial transceiver is used.

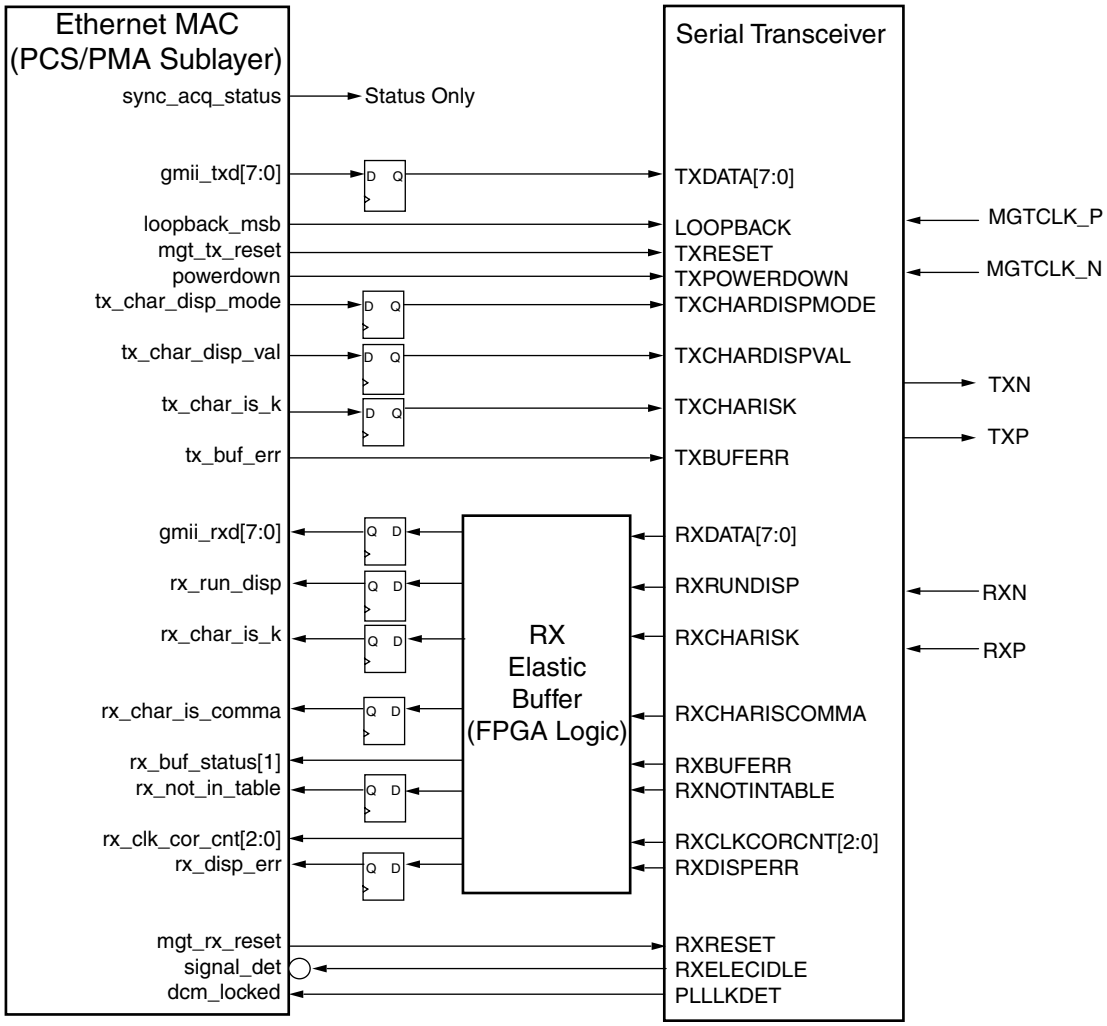

Figure 12-9: **SGMII Connection to a Serial Transceiver**

Figure 12-9 shows that the RX elastic buffer is implemented in the FPGA logic between the serial transceiver and the V6EMAC instead of in the serial transceiver. This alternative RX elastic buffer utilizes a single block RAM to create a buffer that is twice as large as the one present in the serial transceiver. This configuration, therefore, can handle larger frame sizes before clock tolerances accumulate and result in emptying or filling of the buffer. This is necessary for SGMII operation at 10 Mb/s, where each frame size is effectively 100 times larger than the same frame is at 1 Gb/s due to each byte being repeated 100 times.

SGMII Clock Management

Tri-Speed Operation

Figure 12-10 shows the clock management used with the SGMII interface without the FPGA logic elastic buffer. The CLKIN inputs to the serial transceiver must be connected to an external, high-quality differential reference clock with frequency of 125 MHz. A 125 MHz clock source is then provided to the FPGA logic from the TXOUTCLK output port of the serial transceiver. This is connected to global clock routing using a BUFG as illustrated. This clock should then be routed to the `gtx_clk` of the V6EMAC.

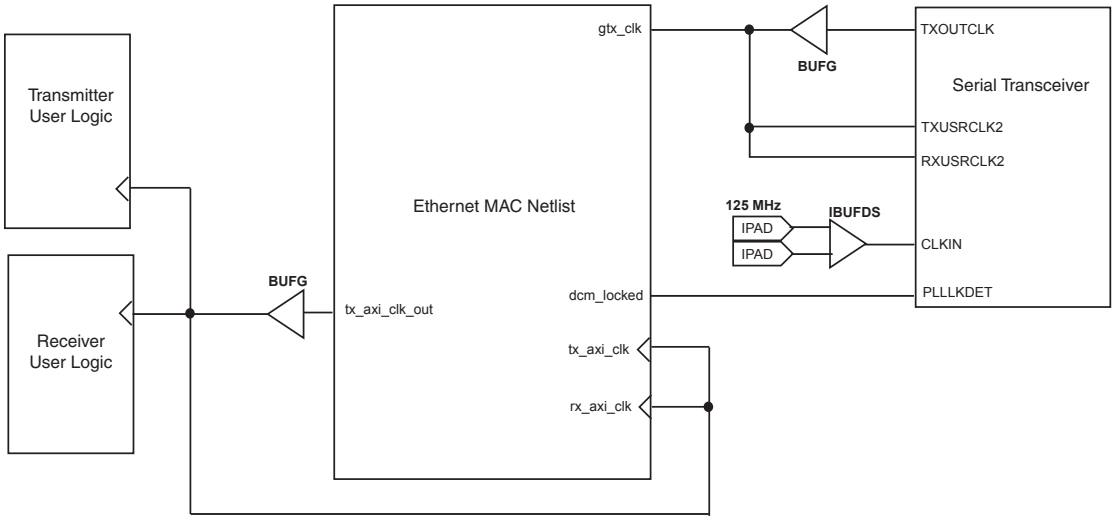

Figure 12-10: **SGMII Clock Management - Serial Transceiver RX Elastic Buffer**

When using the larger FPGA logic implementation of the RX elastic buffer, data is clocked out of the serial transceiver synchronously to RXRECCLK. This clock, placed on a BUFR component, drives RXUSRCLK2 and some of the logic in the RX elastic buffer, as illustrated in Figure 12-11.

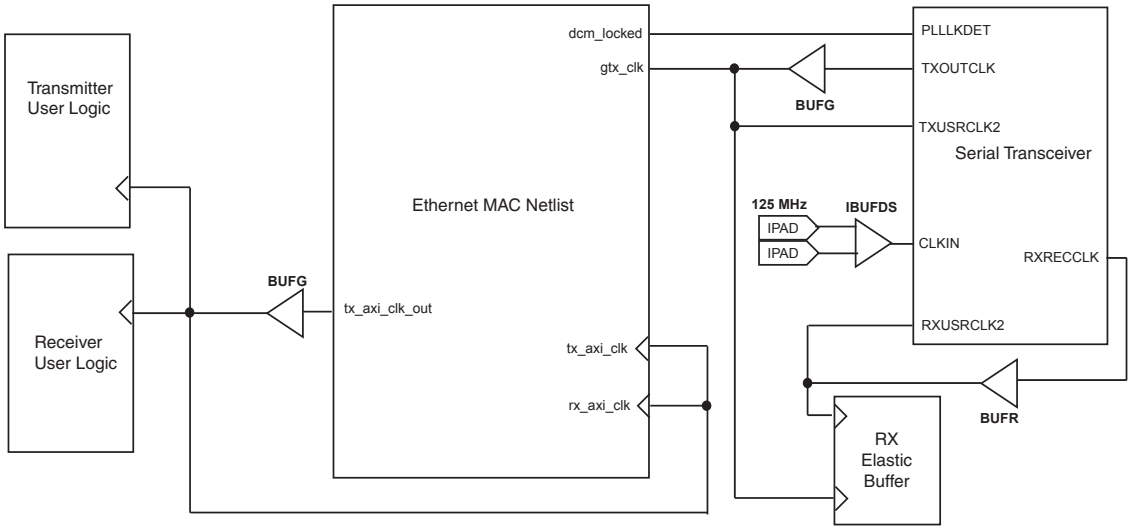

Figure 12-11: **Using the FPGA Fabric RX Elastic Buffer**

In both configurations, the PLLLKDET signal from the serial transceiver (indicating that its internal PLLs have locked) is routed to the *dcm_locked* input port of the V6EMAC, which ensures that the state machines of the V6EMAC are held in reset until the serial transceiver has locked and its clocks are running cleanly.

The `tx_axi_clk_out` is connected to a BUFG, which then provides the user clock to FPGA logic. It must also be routed back to the V6EMAC through the `rx_axi_clk` and `tx_axi_clk` input ports.

1 Gb/s Only Operation

When the SGMII is used only for 1 Gb/s speeds, further clocking optimization can be performed. The clock logic is then identical to that described in 1000BASE-X PCS/PMA Clock Management.

SGMII Auto-Negotiation

Overview of Operation

Figure 12-12 illustrates a simplified diagram of the V6EMAC instantiated within a Virtex-6 device. The only components shown are two of the PCS Management Registers, which are directly involved in the auto-negotiation process (see SGMII Management Registers). The corresponding registers of the connected devices are also shown.

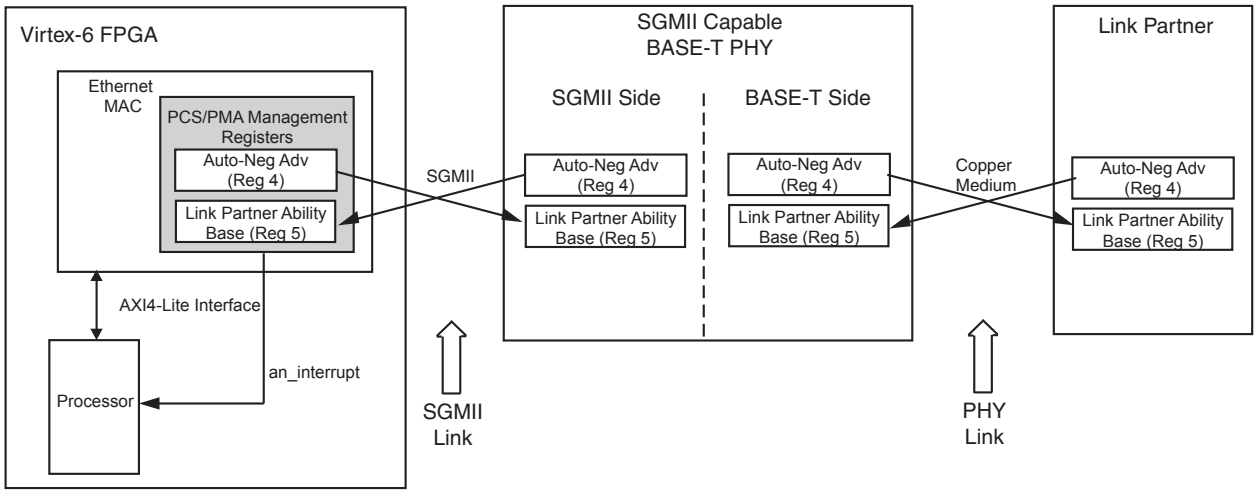

Figure 12-12: **SGMII Auto-Negotiation Overview**

The SGMII-capable PHY has two distinctive sides to auto-negotiation as illustrated:

- The PHY initially performs auto-negotiation with its link partner across the PHY link using the relevant auto-negotiation standard for the chosen medium (BASE-T auto-negotiation, illustrated in Figure 12-12, uses a copper medium). This resolves the operational speed and duplex mode with the link partner.
- The PHY then initiates a secondary auto-negotiation process with the V6EMAC across the SGMII link. This leverages the 1000BASE-X auto-negotiation specification described in 1000BASE-X Auto-Negotiation with only minor differences. This transfers the results of the initial PHY with link partner auto-negotiation across the SGMII, and this is the only auto-negotiation process observed by the V6EMAC.

The SGMII auto-negotiation function leverages the 1000BASE-X Auto-Negotiation function with the exception of:

- The duration of the link timer of the SGMII auto-negotiation decreases from 10 ms to 1.6 ms making the entire auto-negotiation cycle faster (see SGMII Auto-Negotiation Link Timer).
- The information exchanged now contains speed resolution in addition to duplex mode.

Under normal conditions, this completes the auto-negotiation information exchange. The results can be read from Table 8-38. The duplex mode and speed of the V6EMAC should then be configured by a software routine to match. This does not happen automatically within the V6EMAC. There are two methods by which a host processor can learn of the completion of an auto-negotiation cycle:

By polling the auto-negotiation completion bit (bit 5 in Table 8-34.)

By using the auto-negotiation interrupt port (see Auto-Negotiation Interrupt, page 181).

SGMII Auto-Negotiation Link Timer

The auto-negotiation link timer is used to time three phases of the auto-negotiation procedure. For the SGMII standard, this link timer is defined as having a duration of 1.6 ms.

The duration of the link timer used by the V6EMAC can be configured with the EMAC_LINKTIMERVAL[8:0] attribute or by writing to the appropriate management register (see Table 8-9). The duration of the timer is approximately equal to the binary value placed onto this attribute multiplied by 32.768 µs (4096 clock periods of the 125 MHz clock provided to the V6EMAC on `gtx_clk`). The accuracy of this link timer is within the range:

```
+0 to -32.768 µs
```

Therefore, for the SGMII standard, set the EMAC_LINKTIMERVAL[8:0] attribute to:

```
000110010 = 50 decimal
```

This corresponds to a timer duration of between 1.606 and 1.638 ms. This value can be reduced for simulation.

Auto-Negotiation Interrupt

The auto-negotiation function has an an_interrupt port. This port is designed to be used with common microprocessor bus architectures.

The operation of this port is enabled or disabled and cleared using the SGMII Management Registers.

- When disabled, this port is permanently driven Low.
- When enabled, this port is set to logic 1 following the completion of an auto-negotiation cycle. It remains High until cleared after a zero is written to bit 1 (Interrupt Status bit) of the SGMII Vendor Specific Register: Auto-Negotiation Interrupt Control Register (Register 16), page 95.

Loopback When Using the PCS/PMA Sublayer for SGMII

Figure 12-13 illustrates the loopback options when using the PCS/PMA sublayer. Two possible loopback positions are illustrated:

- Loopback in the V6EMAC. When placed into loopback, data is routed from the transmitter to the receiver path at the last possible point in the PCS/PMA sublayer, immediately before the serial transceiver interface. When placed into loopback, a constant stream of Idle code groups are transmitted through the serial transceiver. Loopback in this position allows test frames to be looped back within the V6EMAC without allowing them to be received by the link partner. The transmission of idles allows the link partner to remain in synchronization so that no fault is reported.
- Loopback in the serial transceiver. The serial transceiver can alternatively be switched into loopback by connecting the EMACPHYLOOPBACKMSB port of the V6EMAC to the loopback port of the serial transceiver as illustrated. The serial transceiver loopback routes data from the transmitter path to the receiver path within the serial transceiver. However, this data is also transmitted out of the serial transceiver, so any test frames used for a loopback test are received by the link partner.

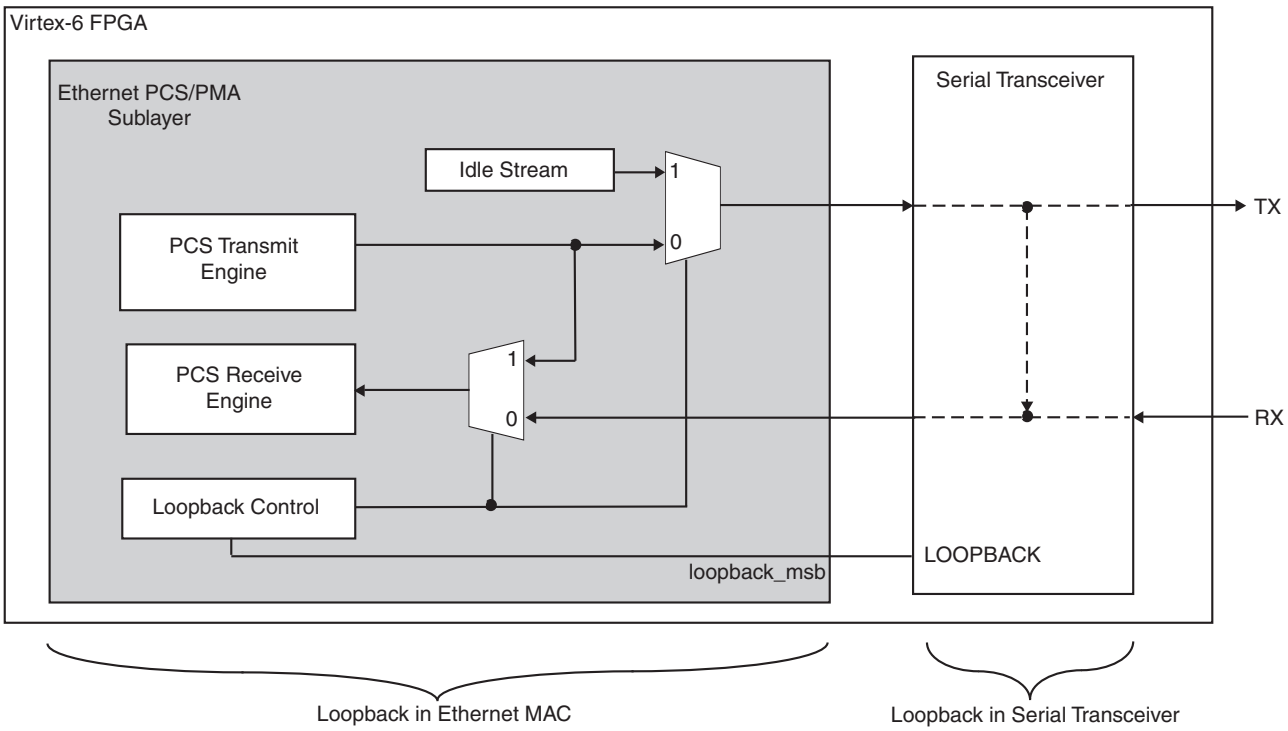

Figure 12-13: **Loopback when using the PCS/PMA Sublayer**

Switching Between SGMII and 1000BASE-X Standards

When the V6EMAC is configured for tri-speed operation with the SGMII physical interface (see SGMII Clock Management), it is possible to switch between SGMII and 1000BASE-X modes without reconfiguring the FPGA. This includes transitions from SGMII to 1000BASE-X and from 1000BASE-X to SGMII.

Mode switching is useful when using off-the-shelf PHY devices with the ability to perform both BASE-X and BASE-T standards, or when a single Virtex-6 FPGA bitstream can be used with either a BASE-X or BASE-T PHY device.

The default method for switching modes follows. This task can be simplified in some cases by using the `base_x_switch` input as detailed in Optional BASE-X Switching Control, which automates the tasks shown below. See Generating the Core for more information about the CORE Generator tool.

Performing a Mode Switch

The V6EMAC mode of operation is set by writing to the Ethernet MAC Mode Configuration Register through the AXI4-Lite interface (see Table 8-9, page 74). Table 12-3 indicates the possible settings.

Table 12-3: **Possible Mode Switch Settings**

Mode of Operation	Bit 28	Bit 27
Illegal Setting	0	0
1000BASE-X	0	1
SGMII	1	0
Illegal Setting	1	1

After the appropriate Ethernet MAC Mode Configuration Register bits have been written, the mode switch begins to take effect immediately.

The programmable link timer value is also set using the Ethernet MAC Mode Configuration Register, and it is recommended that the link timer value appropriate to the new mode of operation is written at the same time as the mode switch command. See 1000BASE-X PCS/PMA Management Registers.

The PCS/PMA Management Registers accessible through MDIO have different interpretations for the two standards. Following a mode switch, these registers should immediately be interpreted under the new standard of operation.

- For SGMII operation, PCS/PMA Management Registers should be interpreted according to the format in SGMII Management Registers.
- For 1000BASE-X operation, PCS/PMA Management Registers should be interpreted according to the format in 1000BASE-X PCS/PMA Management Registers.

When switching from SGMII to 1000BASE-X, the auto-negotiation advertisement register (Register 4) must be reprogrammed to the desired settings.

After the mode switch command and the new link timer value have been written, and the auto-negotiation advertisement register has been reprogrammed (if appropriate), the user must request an auto-negotiation restart by setting bit 9 in the appropriate control register. See Control Register (Register 0) and SGMII Control Register (Register 0), page 91 for details.

The mode switch process is finished following the successful completion of auto-negotiation. Other requirements and considerations for mode switching follow.

Optional BASE-X Switching Control

The V6EMAC can automate the process if the optional SGMII /1000BASE-X mode switching block is included. This option is available in the SGMII Capabilities section of the CORE Generator software when all three of the following selections are made:

- An SGMII physical interface
- 10/100/1000 Mb/s operation
- AXI4-Lite interface

Switching Modes and Speeds

The V6EMAC executes the necessary mode switching commands based on a single signal, `base_x_switch`, present as an input to the netlist. When `base_x_switch` is driven to logic '0', the V6EMAC operates according to the SGMII standard. When `base_x_switch` is driven to logic '1', the V6EMAC operates according to the 1000BASE-X standard.

When operating in SGMII mode (`base_x_switch` = 0), switching between 10, 100, and 1000 Mb/s is supported as usual. When operating in 1000BASE-X PCS/PMA mode (`base_x_switch` = 1), only 1000 Mb/s is supported. When switching from SGMII at 10/100 Mb/s to 1000BASE-X PCS/PMA, the speed is automatically changed to 1000 Mb/s.

The `base_x_switch` signal can be driven by logic internal to the FPGA, or by an external resource, such as a switch or jumper.

Operational Requirements

Dynamic switching during V6EMAC operation is not supported and can result in unknown behavior.

The `base_x_switch` signal must be driven during Ethernet MAC operation; a floating input can result in unknown behavior.

Requirements and Considerations for Mode Switching

To support switching between SGMII and 1000BASE-X standards, the V6EMAC must be configured for tri-speed operation with the SGMII physical interface. This configuration properly supports the 10/100/1000 Mb/s operation of the SGMII standard, in addition to supporting the 1000 Mb/s operation of the 1000BASE-X standard.

Mode switching is only allowed following initial power-up or after the deassertion of reset.

The V6EMAC must operate in 1000BASE-X mode to use an optical fiber medium. It must operate in SGMII mode to provide BASE-T functionality and use a twisted-pair copper medium.

Chapter 13

1000BASE-X PCS/PMA (GPCS)

Ethernet MAC PCS/PMA Sublayer

Figure 13-1 shows the functional block diagram of the V6EMAC with the PCS/PMA sublayer block highlighted.

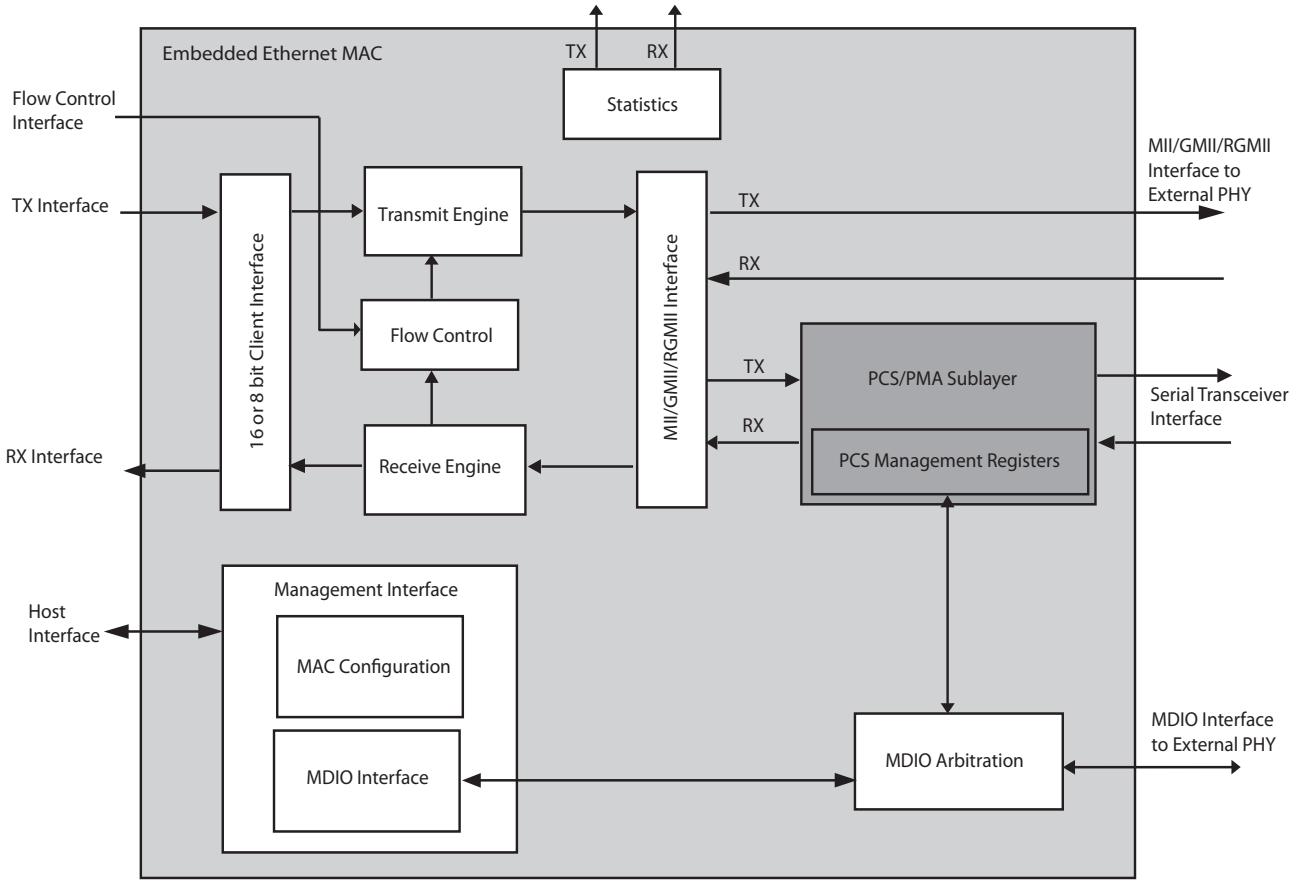

Figure 13-1: **PCS/PMA Sublayer Block Diagram in the V6EMAC**

The PCS/PMA sublayer extends the functionality of the V6EMAC, replacing the GMII/MII or RGMII parallel interfaces with an interface to a serial transceiver. The connected serial transceiver then provides the remaining functionality to accommodate the following two standards, both of which require the high-speed serial interface provided by the serial transceiver:

- Serial Gigabit Media Independent Interface (SGMII)
- 1000BASE-X PCS/PMA (GPCS)

Introduction to the 1000BASE-X PCS/PMA Implementation

The 1000BASE-X physical standard, described in *IEEE Std 802.3-2008* [Ref 4], clauses 36 and 37, defines a physical sublayer that is usually connected to an optical fiber medium. Two common implementations currently exist: 1000BASE-LX and 1000BASE-SX (long and short wavelength laser), which can both be obtained by connecting the serial transceiver to a suitable GBIC or SFP optical transceiver.

Figure 13-2 shows an expanded diagram of the PCS/PMA sublayer block. The PCS/PMA sublayer contains its own functional blocks, which are illustrated and summarized in the following subsections. The serial transceiver is shown connected to an external optical transceiver to complete the 1000BASE-X optical link.

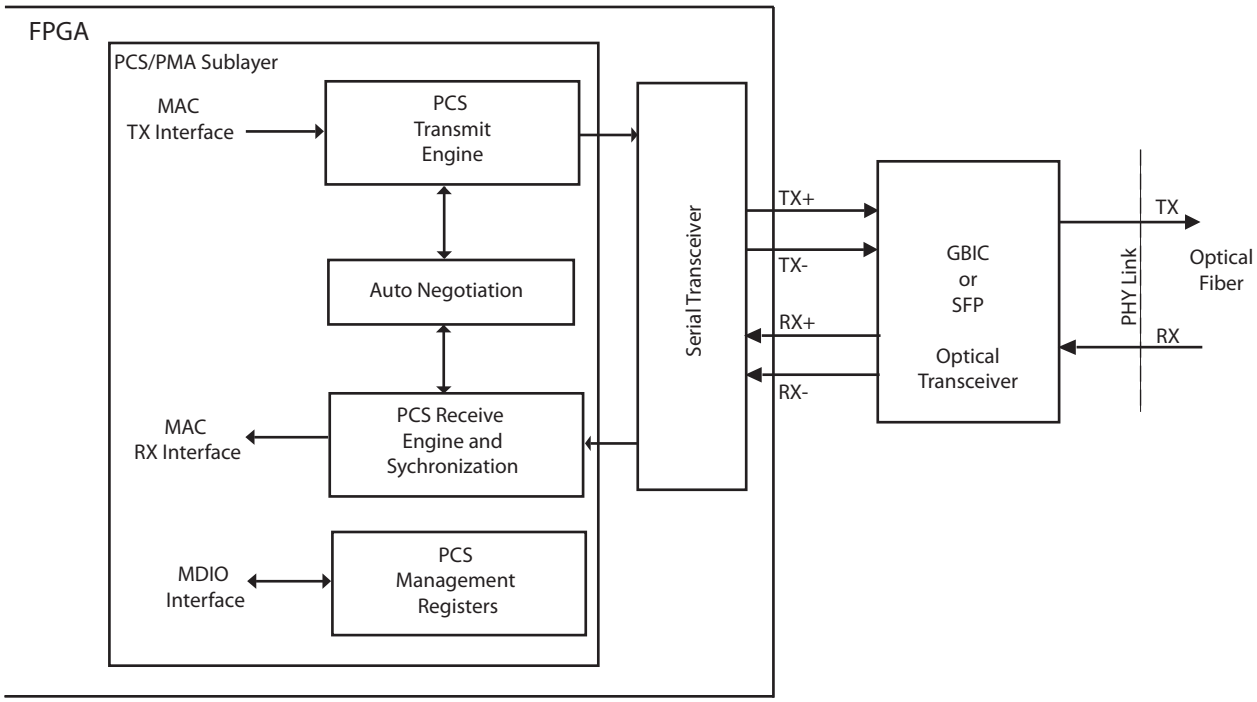

Figure 13-2: **Ethernet PCS/PMA Sublayer Extension**

PCS Transmit Engine

The PCS Transmit Engine encodes the data stream received from the MAC into a sequence of ordered sets as defined by the standard. The data is transmitted to the serial transceiver using an 8-bit datapath that then provides 8B/10B encoding of this data and parallel to serial conversion. The output from the serial transceiver is a differential pair operating at a rate of 1.25 Gb/s.

For more information on the PCS Transmit Engine, see the state diagrams of *IEEE Std 802.3-2008* [Ref 4] (Figures 36-5 and 36-6).

PCS Receive Engine and Synchronization

The serial transceiver receives a serial differential pair operating at a rate of 1.25 Gb/s from the optical transceiver. The serial transceiver, using CDR, performs word alignment on this serial data stream and then converts this to a parallel interface. This data is then 8B/10B decoded by the serial transceiver before presenting this to the PCS/PMA sublayer using an 8-bit datapath

Within the PCS/PMA sublayer, the synchronization process performs analysis of valid/invalid received sequence ordered sets to determine if the link is in good order. The PCS receive engine then decodes these sequence ordered sets into a format that can then be presented to the standard receiver datapath within the V6EMAC.

For more information on the synchronization process, see *IEEE Std 802.3-2008* [Ref 4] (Figure 36-9). For the PCS Receive Engine, see *IEEE Std 802.3-2008* [Ref 4] (Figures 36-7a and 36-7b).

Auto-Negotiation

The 1000BASE-X auto-negotiation function allows a device to advertise the supported modes of operation to a device at the remote end of the optical fiber (the link partner) and to detect corresponding operational modes that the link partner advertises.

Auto-negotiation is controlled and monitored using a software function through the PCS Management Registers. See 1000BASE-X Auto-Negotiation.

PCS Management Registers

Configuration and status of the PCS/PMA sublayer, including access to and from the auto-negotiation function, is through the PCS Management Registers. These registers are accessed through the serial MDIO. See MDIO Interface.

For the 1000BASE-X standard, the configuration and status registers available are listed in 1000BASE-X PCS/PMA Management Registers.

Using the SGMII in this configuration eliminates the possibility of buffer error if the clocks are not tightly controlled enough to use the serial transceiver elastic buffer.

V6EMAC to Serial Transceiver Connections

Figure 13-3 shows the V6EMAC configured with 1000BASE-X PCS/PMA as the physical interface. Connections to the Virtex®-6 FPGA serial transceiver are illustrated. To aid in timing closure, the V6EMAC might provide more than one register stage when configured for the 1000BASE-X physical interface with the 16-bit client interface.

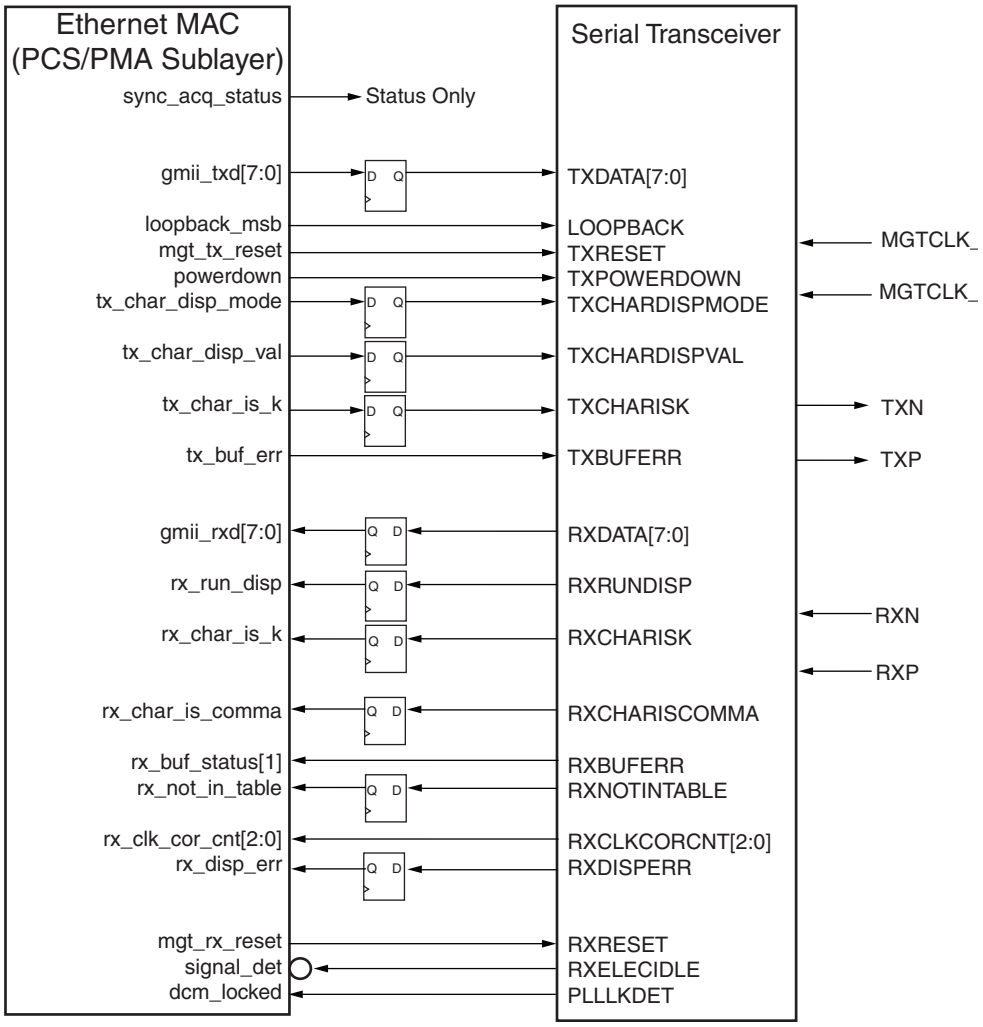

Figure 13-3: **V6EMAC Configured in 1000BASE-X PCS/PMA Mode**

1000BASE-X PCS/PMA Clock Management

1000BASE-X PCS/PMA with 8-bit Client Interface

Figure 13-4 shows the clock management used with the 1000BASE-X PCS/PMA interface in 8-bit client interface mode.

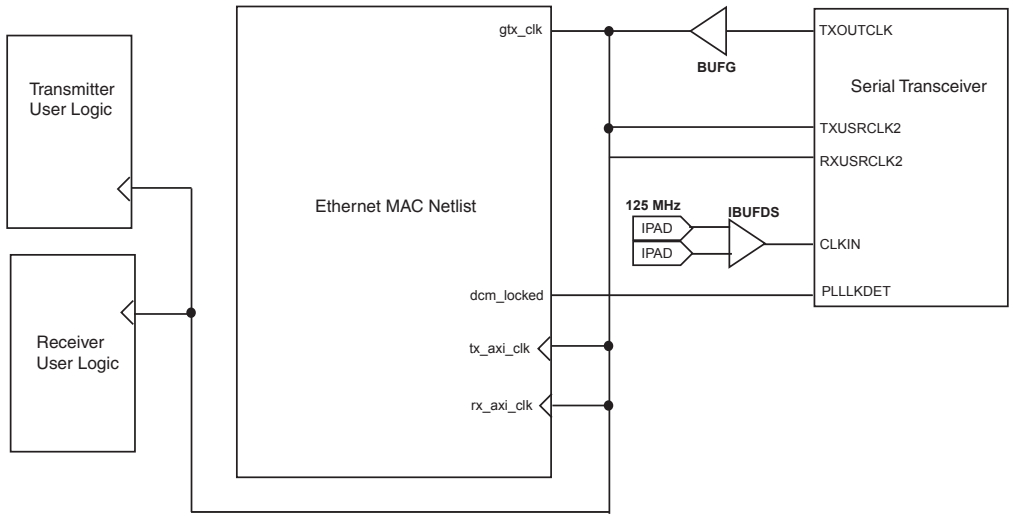

Figure 13-4: **1000BASE-X PCS/PMA (8-Bit Data Client) Clock Management**

The CLKIN inputs to the serial transceiver must be connected to an external, high-quality differential reference clock of frequency of 125 MHz. A 125 MHz clock source is then provided to the FPGA logic from the TXOUTCLK output port of the serial transceiver. This is connected to global clock routing using a BUFG as illustrated in Figure 13-4. This clock should then be routed to the `gtx_clk` port of the V6EMAC.

Additionally, this global clock can be used for both receiver and transmitter user logic, as illustrated in Figure 13-4, and it therefore must be routed back to the V6EMAC through the `rx_axi_clk` and `tx_axi_clk` input ports.

The PLLLKDET signal from the serial transceiver (indicating that its internal PLLs have locked) is routed to the `dcm_locked` input port of the V6EMAC. This ensures that the state machines of the V6EMAC are held in reset until the serial transceiver has locked, and its clocks are running cleanly.

1000BASE-X PCS/PMA with 16-bit Client Interface

Figure 13-5 shows the clock management used with the 1000BASE-X PCS/PMA interface and a 16-bit client data width. This mode supports overclocking at data rates of 2 Gb/s and 2.5 Gb/s based upon the combination of the serial transceiver attributes and MMCM attributes. Because IEEE Std 802.3-2005 specifies a 1000BASE-X rate of 1 Gb/s, overclocking should not be used for Ethernet-compliant designs; however, it can be useful for some proprietary implementations.

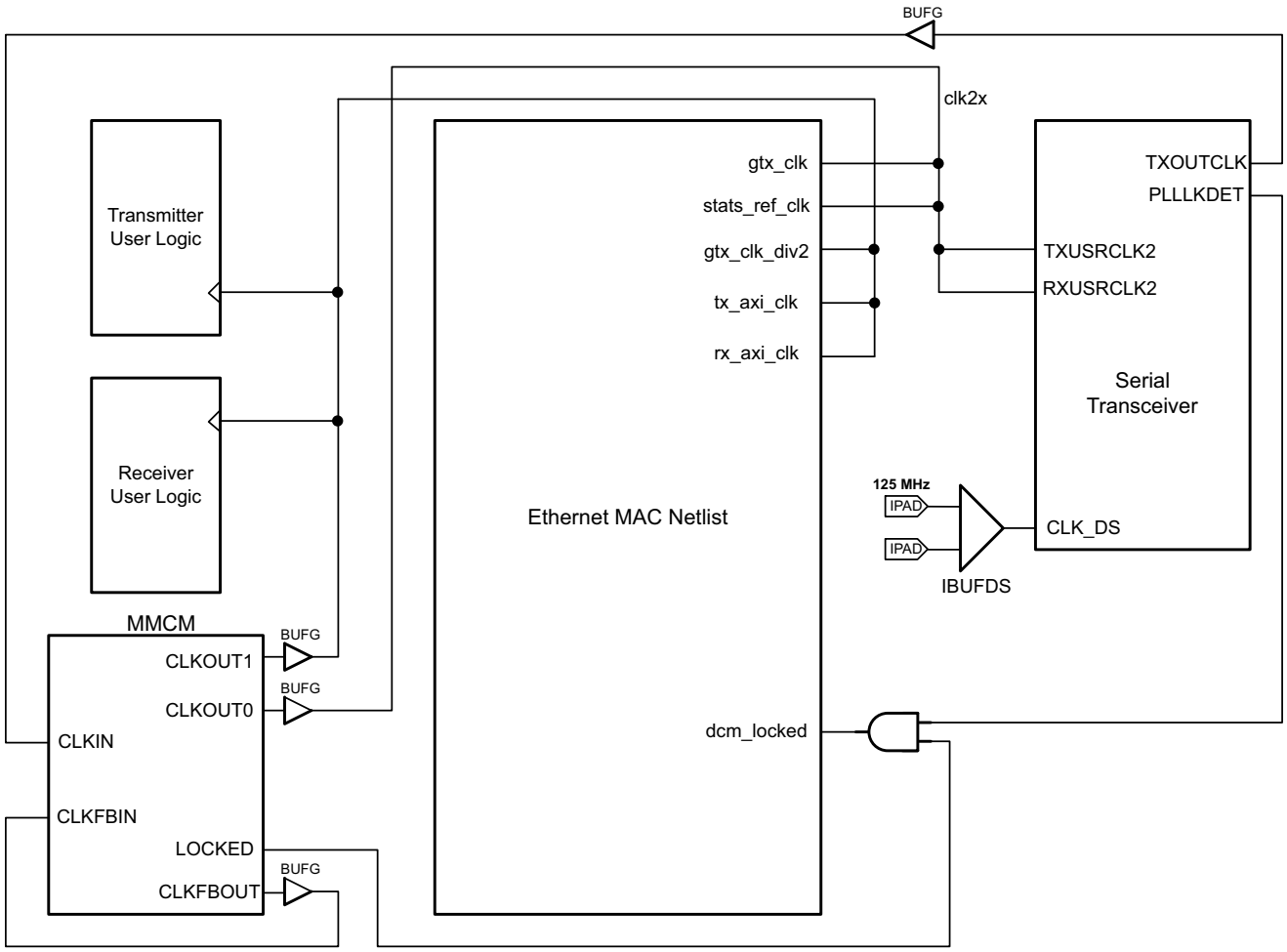

Figure 13-5: **1000BASE-X PCS/PMA (16-bit Data Client) Clock Management**

When operating with the 16-bit client interface, an additional clock (`gtx_clk_div2`) is input to the wrappers. This is used to clock the AXI4-Stream client interface and associated user logic. The frequency of `gtx_clk_div2` depends on the rate in use as follows:

- For 2000 Mb/s, `gtx_clk_div2` is 125 MHz.
- For 2500 Mb/s, `gtx_clk_div2` is 156.25 MHz.

The `gtx_clk` input requires a clock at twice the rate of `gtx_clk_div2`. This is used to clock the serial transceiver and the physical side of the Ethernet MAC. The `gtx_clk` input is generated by an MMCM, which is driven by the TXOUTCLK output of the transceiver and can be shared between multiple instantiations of the wrappers. The frequency of `gtx_clk` depends on the rate in use as follows:

- For 2000 Mb/s, `gtx_clk` is 250 MHz.
- For 2500 Mb/s, `gtx_clk` is 312.5 MHz.

For all overclocked cases, the serial transceiver reference clock (CLK_DS) should be supplied at 125 MHz. This means TXOUTCLK from the transceiver is also 125 MHz and is multiplied to the correct frequencies by the MMCM, as shown in the core Example Design.

While the use of a 125 MHz transceiver reference clock is the provided approach, it is also possible to supply a 156.25 MHz transceiver reference clock and still achieve the required

gtx_clk and gtx_clk_div2 frequencies described. This approach, which can result in different serial transceiver performance characteristics, requires manual modification to both the serial transceiver attributes and the MMCM attributes and is not provided by the V6EMAC.

1000BASE-X Auto-Negotiation

Overview of Operation

Figure 13-6 illustrates a simplified diagram of the V6EMAC instantiated within a Virtex-6 device. The only components shown are two of the PCS Management Registers that are directly involved in the auto-negotiation process (see 1000BASE-X PCS/PMA Management Registers). The corresponding registers of the connected device are also shown.

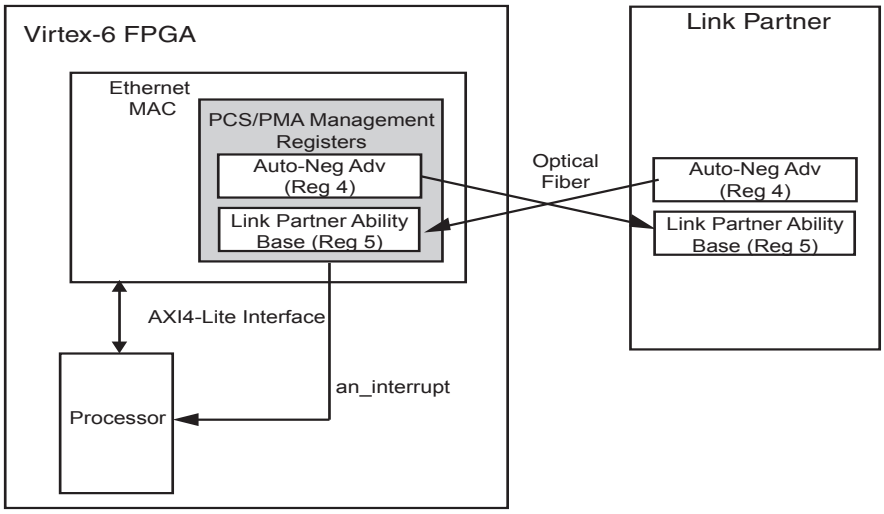

Figure 13-6: **1000BASE-X Auto-Negotiation Overview**

IEEE Std 802.3-2008 [Ref 4], clause 37 describes the 1000BASE-X auto-negotiation function. This function allows a device to advertise its supported modes of operation to a device at the remote end of a link segment (the link partner) and to detect corresponding operational modes advertised by the link partner. The operation of the 1000BASE-X auto-negotiation is summarized as follows:

- To enable auto-negotiation, both auto-negotiation (see Control Register (Register 0), page 85) and MDIO (see MDIO Configuration word 0 (0x500), page 78) must be enabled. If enabled, auto-negotiation starts automatically:
  - After power-up/reset
  - Upon loss of synchronization
  - Whenever the link partner initiates auto-negotiation
  - Whenever an auto-negotiation restart is requested (see Table 8-20)
  - When the PHY Reset bit is set in the PCS control register (see Table 8-20)

- During auto-negotiation, the contents of the Auto-Negotiation Advertisement Register (Register 4), page 87 are transferred to the link partner. This register can be written to through the management interface, and enables software control of the systems advertised abilities. Information provided in this register includes:
  - Fault condition signalling
  - Duplex mode
  - Flow control capabilities for the V6EMAC
- At the same time, the advertised abilities of the link partner are transferred into the Auto-Negotiation Link Partner Ability Base Register (Register 5), page 88 This includes the same information fields as in the Auto-Negotiation Advertisement Register (Register 4), page 87.
- Under normal conditions, this completes the auto-negotiation information exchange. The results can be read from the Auto-Negotiation Link Partner Ability Base Register (Register 5), page 88. The mode of the V6EMAC should then be configured by a software routine to match; this does not happen automatically within the V6EMAC. The two methods by which a host processor can learn of the completion of an auto-negotiation cycle are:
- By polling the auto-negotiation completion bit (bit 5 in Status Register (Register 1), page 86).
- By using the auto-negotiation interrupt port (see Auto-Negotiation Interrupt).

1000BASE-X Auto-Negotiation Link Timer

The auto-negotiation link timer is used to time three phases of the auto-negotiation procedure. For the 1000BASE-X standard, this link timer is defined as having a duration of between 10 and 20 ms. Both link partners wait until their own link timer and the partner's link timer expire before moving on to the next phase of the auto negotiation process. The complete process takes a minimum of three times the values of the biggest link timer value.

The duration of the link timer used by the V6EMAC can be configured with the EMAC_LINKTIMERVAL[8:0] attribute or by writing to the appropriate management register (see Ethernet MAC Mode Configuration Word (0x410), page 74). The duration of the timer is approximately equal to the binary value placed onto this attribute multiplied by 32.768 µs (4,096 clock periods of the 125 MHz clock provided to the V6EMAC on PHYEMACGTXCLK). The accuracy of this link timer is within the range:

```
+0 to -32.768 µs
```

Therefore, for the 1000BASE-X standard, the attribute EMAC_LINKTIMERVAL[8:0] is set to:

```
100111101 = 317 decimal
```

This setting corresponds to a timer duration of between 10.354 and 10.387 ms. This value can be reduced for simulation.

Auto-Negotiation Interrupt

The auto-negotiation function has an EMACCLIENTANINTERRUPT port. This port is designed to be used with common microprocessor bus architectures.

The operation of this port is enabled or disabled and cleared using the 1000BASE-X PCS/PMA Management Registers.

- When disabled, this port is permanently driven Low.
- When enabled, this port is set to logic 1 following the completion of an auto-negotiation cycle. It remains High until cleared after a zero is written to bit 1 (Interrupt Status bit) of the Vendor Specific Register: Auto-Negotiation Interrupt Control Register (Register 16), page 89.

Use of Clock Correction Sequences

The serial transceiver is configured by the Virtex-6 FPGA GTX Transceiver Wizard to perform clock correction. The output of the Transceiver Wizard is provided as part of the Ethernet MAC wrapper generated using the CORE Generator™ tool. Two different clock correction sequences can be employed:

- The mandatory clock correction sequence is the /I2/ ordered set; this is a two-byte code-group sequence formed from /K28.5/ and /D16.2/ characters. The /I2/ ordered set is present in the interframe gap. These sequences can therefore be removed or inserted by the transceiver's receiver elastic buffer without causing frame corruption.
- The default Transceiver Wizard configuration enables the CLK_COR_SEQ_2_USE attribute. In this case, the transceiver is also configured to perform clock correction on the /K28.5/D21.5/ sequence; these are the first two code-groups from the /C1/ ordered set (the /C1/ ordered set is four code-groups in length). Because there are no /I2/ ordered sets present during much of the auto-negotiation cycle, this provides a method of allowing clock correction to be performed during auto-negotiation. Because this form of clock correction inserts or removes two code groups into or from a four code-group sequence, this causes ordered-set fragments to be seen by the Ethernet MAC's auto-negotiation state machine. It is therefore important that the transceiver's RXCLKCORCNT[2:0] port be correctly connected to the Ethernet MAC; this indicates a clock correction event (and type) to the Ethernet MAC. Using this signal, the Ethernet MAC's state machine can interpret the clock-correction fragments, and the auto-negotiation function can complete cleanly.

  When the transceiver's CLK_COR_SEQ_2_USE attribute is not enabled, no clock correction can be performed during much of the auto-negotiation cycle. When this is the case, it is possible that the transceiver's receiver elastic buffer could underflow or overflow as asynchronous clock tolerances accumulate. This results in an elastic buffer error. It is therefore important that the transceiver's RXBUFSTATUS[2:0]port is correctly connected to the Ethernet MAC; this indicates a buffer error event to the Ethernet MAC. Using this signal, the Ethernet MAC's state machine can interpret the buffer error and the auto-negotiation function can complete cleanly.

The serial transceiver is by default configured to perform clock correction during the auto-negotiation cycle. If correctly connected as per the example design, the Ethernet MAC's state machine can correctly determine the transceiver's elastic buffer behavior, and auto-negotiation can complete cleanly.

Loopback When Using the PCS/PMA Sublayer for 1000BASE-X

Figure 13-7 illustrates the loopback options when using the PCS/PMA sublayer. Two possible loopback positions are illustrated:

- Loopback in the V6EMAC. When placed into loopback, data is routed from the transmitter to the receiver path at the last possible point in the PCS/PMA sublayer, immediately before the serial transceiver interface. When placed into loopback, a constant stream of Idle code groups are transmitted through the serial transceiver. Loopback in this position allows test frames to be looped back within the V6EMAC without allowing them to be received by the link partner. The transmission of idles allows the link partner to remain in synchronization so that no fault is reported.
- Loopback in the serial transceiver. The serial transceiver can alternatively be switched into loopback by connecting the EMACPHYLOOPBACKMSB port of the V6EMAC to the loopback port of the serial transceiver as illustrated. The serial transceiver loopback routes data from the transmitter path to the receiver path within the serial transceiver. However, this data is also transmitted out of the serial transceiver, so any test frames used for a loopback test are received by the link partner.

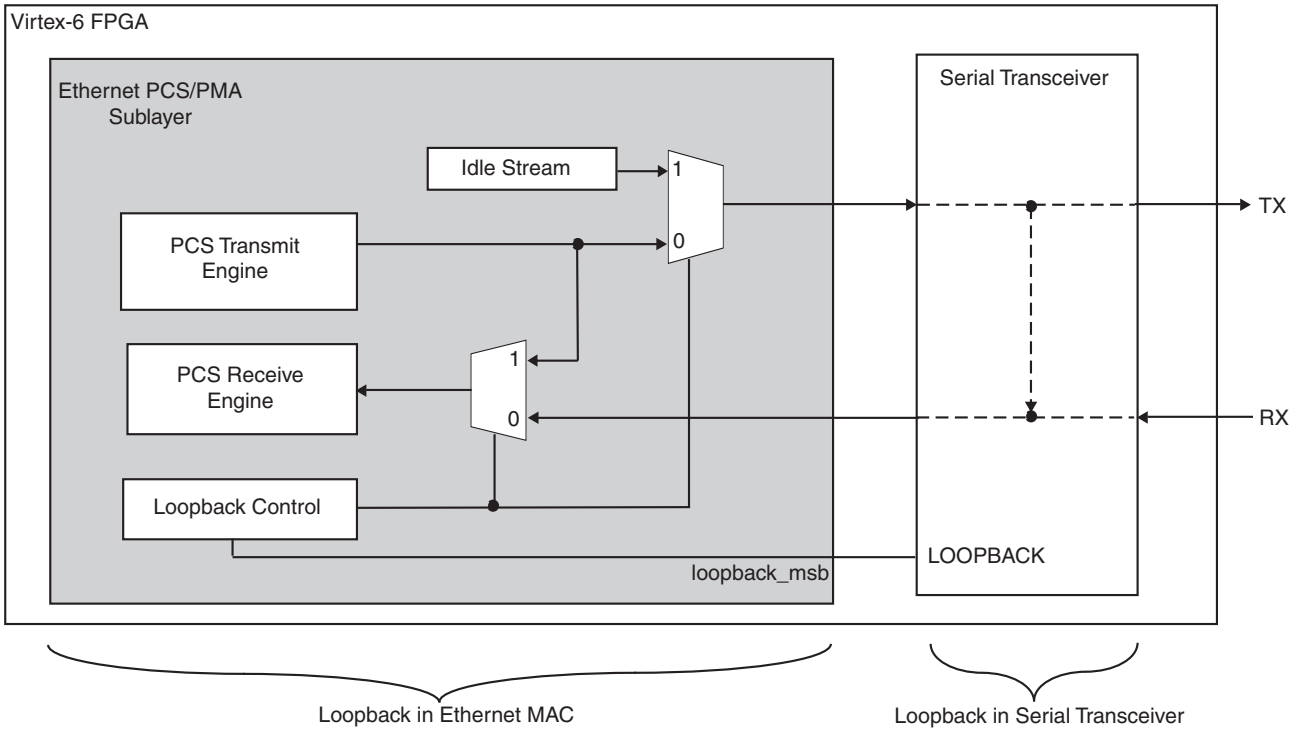

Figure 13-7: **Loopback when using the PCS/PMA Sublayer**

Loopback itself can be enabled or disabled by writing to the Control Register (Register 0), page 85. The loopback position can be controlled through the Vendor Specific Register: Loopback Control Register (Register 17), page 90. See 1000BASE-X PCS/PMA Management Registers for details. Alternatively, loopback can be controlled by setting the EMAC_PHYLOOPBACKMSB and EMAC_GTLOOPBACK attributes.

XILINX®

Chapter 14

Constraining the Core

This chapter defines the constraint requirements of the V6EMAC. An example UCF is provided with the HDL example design to provide samples of constraint requirements for the design. See Chapter 16, Quick Start Example Design and Chapter 17, Detailed Example Design.

I/O Location Constraints

Xilinx recommends the following guidelines to improve design timing using the Virtex®-6 FPGA Embedded Tri-Mode Ethernet MAC:

- If available, use dedicated global clock pins for the V6EMAC input clocks.
- Use the column of IOBs located closest to the V6EMAC block.
- For 1000BASE-X PCS/PMA or SGMII physical interface configurations, use the serial transceiver located closest to the V6EMAC block.

Regional Clocking Requirements

When implementing the V6EMAC with either a GMII or RGMII physical interface, regional clocking methodologies are used. For diagrams of specific clocking networks, see the appropriate configurations in Chapter 9, Media Independent Interface (MII) through Chapter 13, 1000BASE-X PCS/PMA (GPCS).

The regional clocking networks, which utilize BUFIO and BUFR clock buffers in conjunction with IODELAY resources, substantially improve timing and are often necessary to achieve timing closure on the physical interface. However, they also impose certain pinout requirements which are critical to the planning process.

Specifically, the following three requirements must be met to support the regional clocking methodologies specified for GMII and RGMII physical interface implementations:

- The receive-side physical interface clock (GMII_RX_CLK for GMII, or RGMII_RXC for RGMII) must be placed at a clock-capable I/O (CCIO) pin.
- All receive-side physical interface signals (see Table 14-1) must be placed at package pins that correspond to the same clock region as the receive-side physical interface clock.
- An available Embedded Tri-Mode Ethernet MAC block must be present in a clock region that is reachable by the regionally buffered physical interface clock net.

The ideal solution is to place all receive-side physical interface signals within a single clock region, which also contains an Embedded Tri-Mode Ethernet MAC block. The receive-side physical interface clock must be placed at a CCIO in that region.

Table 14-1: **Receive-side Physical Interface Signals with Regional Clocking Requirements**

GMII	RGMII
GMII-RX_CLK (to CCIO pin)	RGMII_RXC (to CCIO pin)
GMII_RXD[7:0]	RGMII_RXD[3:0]
GMII_RX_DV	RGMII_RX_CTL
GMII_RX_ER	
GMII_COL	
GMII_CRS	

Other solutions are possible. When configuring the V6EMAC with a GMII or RGMII physical interface, it is highly recommended to study regional clocking capabilities and requirements during the pinout planning process. See the *Virtex-6 FPGA Configuration User Guide* [Ref 9] for details.

Note: The example designs delivered by the CORE Generator™ software to accompany the core netlist contain I/O placement constraints. These are provided as an example only and should be edited for specific customer placements.

Timing Constraints

Timing Constraints are provided in the UCF delivered with the HDL example design for the core. The remainder of this chapter attempts to describe the constraints that are provided in this file; constraints relating to the core netlist and for the block level of the example design (GMII/MII and RGMII logic) are discussed.

However, because the core can be generated with a large number of clock logic variations, should any inconsistency arise between the UCF and constraints described in this chapter, then the constraints provided in the example design UCF should take precedence.

PERIOD(s) for Clock Nets

These are the primary clock input constraints.

The example design uses a MMCM to generate the core clocks required for all parallel interface implementations of the core (MII/GMII and RGMII).

The clock input to this MMCM is constrained for all implementations of the example design:

```
NET "clk_in_p" TNM_NET = "clk_in_p";
TIMESPEC "TS_v6emac_clk_in_p" = PERIOD "clk_in_p" 5.000 ns HIGH 50%
INPUT_JITTER 50.0ps;
```

Transmitter Clock Constrains

Depending on the selected physical interface, and the supported Ethernet speeds, a wide variation of required clock period constraint syntax exists. However, the UCF provided with the example design always provides the correct constraints for the generated example design and so this file should be used for reference. Do not relax these clock period constraints.

The following syntax provides an example. This is taken from a tri-speed capable design using GMII:

```
# Ethernet GTX_CLK high quality 125 MHz reference clock
NET "gtx_clk_bufg" TNM_NET = "ref_gtx_clk";
TIMEGRP "v6emac_clk_ref_gtx" = "ref_gtx_clk";
TIMESPEC "TS_v6emac_clk_ref_gtx" = PERIOD "v6emac_clk_ref_gtx" 8 ns
HIGH 50 %;

# Multiplexed 1 Gb/s, 10/100 Mb/s output inherits constraint from
GTX_CLK
NET "*tx_mac_aclk*" TNM_NET  = "clk_tx_mac";
TIMEGRP "v6emac_clk_ref_mux" = "clk_tx_mac";
TIMESPEC "TS_v6emac_clk_ref_mux" = PERIOD "v6emac_clk_ref_mux"
TS_v6emac_clk_ref_gtx HIGH 50%;
```

Receiver Clock Constraints

Depending on the selected physical interface, and the supported Ethernet speeds, a wide variation of required clock period constraint syntax exists. However, the UCF provided with the example design always provides the correct constraints for the generated example design and so this file should be used for reference. Do not relax these clock period constraints.

The following syntax provides an example. This is taken from a tri-speed capable design using GMII:

```
# Ethernet GMII PHY-side receive clock
NET "gmii_rx_clk" TNM_NET = "phy_clk_rx";
TIMEGRP "v6emac_clk_phy_rx" = "phy_clk_rx";
TIMESPEC "TS_v6emac_clk_phy_rx" = PERIOD "v6emac_clk_phy_rx" 7.5 ns HIGH 50 %;
```

IDELAYCTRL Reference Clock Constraints

The required reference clock is generated by the MMCM and the required constraints are inherited from this. However, if the MMCM is not used the following constraint must be used on the required `ref_clk` input. See the *Virtex-6 FPGA SelectIO Resources User Guide* [Ref 10] for IDELAYCTRL components and the supported reference clock frequency range.

```
NET "*refclk_bufg" TNM_NET  = "clk_ref_clk";
TIMESPEC "TS_ref_clk" = PERIOD "clk_ref_clk" 5000 ps HIGH 50 %;
```

Management Clock Constraints

Whenever the optional Management Interface is present in the core, the `s_axi_aclk` signal must be constrained to run at the desired frequency. This is set to 125 MHz in the example design.

```
NET "*s_axi_aclk"   TNM_NET  = "clk_axi";
TIMEGRP "v6emac_config_clk"  = "clk_axi";
TIMESPEC "TS_v6emac_config_clk" = PERIOD "v6emac_config_clk" 8 ns HIGH 50 %;
```

Constraints when Implementing an External GMII

The constraints defined in this section are implemented in the UCF for the GMII example design delivered with the core. Sections from this UCF are copied into the following descriptions to act as an example. These should be studied in conjunction with the HDL source code for the example design and with the description given in the relevant physical interface chapters in this guide.

GMII IOB Constraints

The following constraints target the flip-flops that are inferred in the top-level HDL file for the example design; constraints are set to ensure that these are placed in IOBs.

```
INST "*gmii_txd_reg*"    IOB = true;
INST "*gmii_tx_en_reg"   IOB = true;
INST "*gmii_tx_er_reg"   IOB = true;

INST "*rxd_to_mac*"    IOB = true;
INST "*rx_dv_to_mac"   IOB = true;
INST "*rx_er_to_mac"   IOB = true;
```

Virtex-6 devices only support GMII/MII at 2.5V and the device default SelectIO™ interface standard of LVCMOS25 is used. In addition, the example design provides pad locking on the GMII I/Os. These are provided as a guideline only; there are no specific I/O location constraints for this core.

GMII Input Setup/Hold Timing

Figure 14-1 and Table 14-2 illustrate the setup and hold time window for the input GMII signals. This is the worst-case data valid window presented to the FPGA pins.

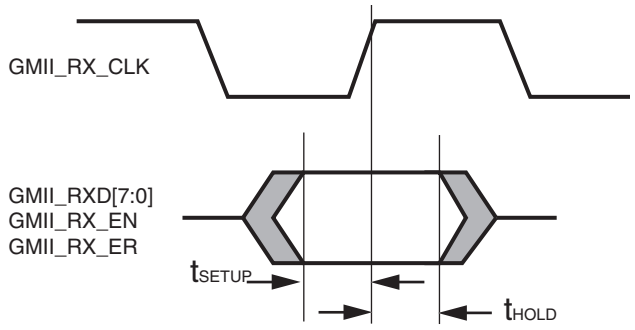

Figure 14-1: **Input GMII Timing**

Observe that there is a 2 ns data valid window which is presented across the GMII input bus. This must be correctly sampled by the FPGA devices.

Table 14-2: **Input GMII Timing**

Symbol	Min	Max	Units
t_{SETUP}	2.00	-	ns
t_{HOLD}	0.00	-	ns

The GMII example design uses BUFIO/BUFR routing on the clock and IODELAY components on the receiver data and control signals. A fixed tap delay can be applied to delay the data and control signals so that they are correctly sampled by the `gmii_rx_clk` clock at the IOB flip-flop, thereby meeting GMII setup and hold timing.

The following constraint shows an example of setting the delay value for one of these IODELAY components. All bits can be adjusted individually, if desired, to compensate for any PCB routing skew.

```
INST *gmii_interface/delay_gmii_rx_dv IDELAY_VALUE = 23;
```

The value of IDELAY_VALUE is preconfigured in the example designs to meet the setup and hold constraints for the example GMII pinout in the particular device. The setup/hold

timing which is achieved after place-and-route is reported in the data sheet section of the TRCE report (created by the implement script). See Understanding Timing Reports for GMII Setup/Hold Timing.

When IODELAY primitives are instantiated with a fixed delay attribute, an IDELAYCTRL component must be also instantiated to continuously calibrate the individual input delay elements. The IDELAYCTRL module requires a reference clock, which is assumed to be an input to the example design delivered by the CORE Generator software. The most efficient way to use the IDELAYCTRL module is to lock the placement of the instance to the clock region of the device where the IDELAY/IODELAY components are placed. To aid the tools in this process the IODELAY components are placed into an IODELAY_GROUP in the UCF. See the *Virtex-6 FPGA Configuration User Guide* [Ref 9] and code comments for more information. In addition, for all Virtex-6 family designs, the following UCF syntax is included:

```
INST "gmii_rxd<?>" TNM = “gmii_rx”;
INST "gmii_rx_dv"  TNM = “gmii_rx”;
INST "gmii_rx_er"  TNM = “gmii_rx”;

TIMEGRP "IN_GMII" OFFSET = IN  2 ns VALID 2 ns BEFORE "gmii_rx_clk" RISING;
```

This syntax causes the Xilinx implementation tools to analyze the input setup and hold constraints for the input GMII bus. If these constraints are not met, the tools report timing errors. However, the tools do NOT attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the IDELAY_VALUEs in the UCF.

Understanding Timing Reports for GMII Setup/Hold Timing

Setup and Hold results for the GMII input bus can be found in the data sheet section of the Timing Report. The results are self-explanatory and it is easy to see how they relate to Figure 14-1. Here follows an example report. The implementation requires 1.835 ns of setup: this is less than the 2 ns required and so there is slack. The implementation requires -0.226 ns of hold; this is less than the 0 ns required and so there is slack.

```
Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)
Setup/Hold to clock gmii_rx_clk
------------+------------+------------+-------------------+--------+
            |  Setup to  |  Hold to   |                   | Clock  |
Source      | clk (edge) | clk (edge) |Internal Clock(s)  | Phase  |
------------+------------+------------+-------------------+--------+
gmii_rx_dv  |    1.820(R)|   -0.281(R)| gmii_rx_clk_bufio |   0.000|
gmii_rx_er  |    1.770(R)|   -0.226(R)| gmii_rx_clk_bufio |   0.000|
gmii_rxd<0> |    1.821(R)|   -0.283(R)| gmii_rx_clk_bufio |   0.000|
gmii_rxd<1> |    1.833(R)|   -0.295(R)| gmii_rx_clk_bufio |   0.000|
gmii_rxd<2> |    1.790(R)|   -0.253(R)| gmii_rx_clk_bufio |   0.000|
gmii_rxd<3> |    1.789(R)|   -0.252(R)| gmii_rx_clk_bufio |   0.000|
gmii_rxd<4> |    1.834(R)|   -0.296(R)| gmii_rx_clk_bufio |   0.000|
gmii_rxd<5> |    1.829(R)|   -0.291(R)| gmii_rx_clk_bufio |   0.000|
gmii_rxd<6> |    1.793(R)|   -0.255(R)| gmii_rx_clk_bufio |   0.000|
gmii_rxd<7> |    1.835(R)|   -0.296(R)| gmii_rx_clk_bufio |   0.000|
------------+------------+------------+-------------------+--------+
```

Constraints when Implementing an External RGMII

The constraints defined in this section are implemented in the UCF for the example design delivered with the core. Sections from this UCF are copied into the following descriptions to act as an example. These should be studied in conjunction with the HDL source code for the example design and with the description given in the relevant physical interface chapters in this guide.

RGMII IOB Constraints

The RGMII v2.0 is a 1.5 V signal-level interface. The 1.5 VHSTL Class I SelectIO interface standard is used for RGMII interface pins. Use the following constraints with the device I/O Banking rules. The I/O slew rate can be set to 'fast' to ensure that the interface can meet setup and hold times.

```
INST "rgmii_txd<?>" IOSTANDARD = HSTL_I;
INST "rgmii_tx_ctl" IOSTANDARD = HSTL_I;
INST "rgmii_txc"    IOSTANDARD = HSTL_I;
INST "rgmii_rxd<?>" IOSTANDARD = HSTL_I;
INST "rgmii_rx_ctl" IOSTANDARD = HSTL_I;
INST "rgmii_rxc"    IOSTANDARD = HSTL_I;
```

In addition, the example design can provide pad locking on the RGMII. This is a provided as a guideline only; there are no specific I/O location constraints for this core.

RGMII Input Setup/Hold Timing

Figure 14-2 and Table 14-3 illustrate the setup and hold time window for the input RGMII signals. This is the worst-case data valid window presented to the FPGA pins.

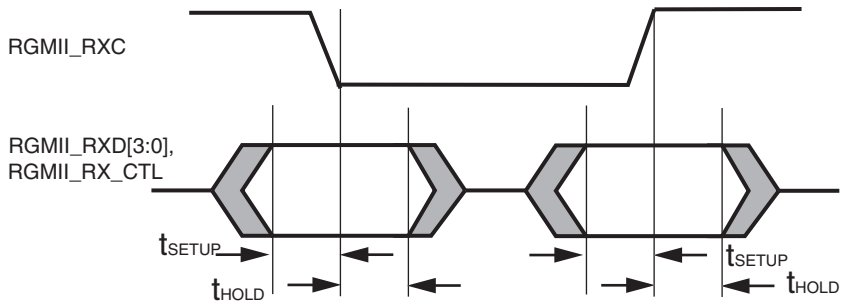

Figure 14-2: **Input RGMII Timing**

Observe that there is a 2 ns data valid window which is presented across the RGMII input bus. This must be correctly sampled on both clock edges by the FPGA devices.

Table 14-3: **Input RGMII Timing**

Symbol	Min	Typical	Units
t_{SETUP}	1.0	2.0	ns
t_{HOLD}	1.0	2.0	ns

For RGMII, the lower data bits, `rgmii_rxd[3:0]`, should be sampled internally on the rising edge of `rgmii_rxc`, and the upper data bits, `rgmii_rxd[7:4]`, should be sampled internally on the falling edge of `rgmii_rxc`.

The RGMII design uses an IODELAY component on the `rgmii_txc` transmitter output clock. A fixed tap delay is applied to move the rising edge of this clock to the center of the output data window. This is performed with the following UCF syntax:

```
INST "*v6emac_block*rgmii_interface*delay_rgmii_tx_clk" ODELAY_VALUE = 4;
```

The RGMII receiver design uses BUFIO/BUFR routing on the clock and IODELAY components on the data and control signals. A fixed tap delay can be applied to delay the data and control signals so that they are correctly sampled by the `rgmii_rxc` clock at the IOB IDDR registers, thereby meeting RGMII setup and hold timing.

The following constraint shows an example of setting the delay value for one of these IODELAY components. Data/Control bits can be adjusted individually, if desired, to compensate for any PCB routing skew.

```
INST "*v6emac_block*rgmii_interface*delay_rgmii_rx_ctl" IDELAY_VALUE = 12;
```

The value of IDELAY_VALUE is preconfigured in the example designs to meet the setup and hold constraints for the example RGMII pinout in the particular device. The setup/hold timing which is achieved after place-and-route is reported in the data sheet section of the TRCE report (created by the implement script). See Understanding Timing Reports for RGMII Setup/Hold Timing.

When IODELAY primitives are instantiated with a fixed delay attribute, an IDELAYCTRL component must be also instantiated to continuously calibrate the individual input delay elements. The IDELAYCTRL module requires a reference clock, which is assumed to be an input to the example design delivered by the CORE Generator software. The most efficient way to use the IDELAYCTRL module is to lock the placement of the instance to the clock region of the device where the IDELAY/IODELAY components are placed. To aid the tools all related IODELAY components are placed in a single IODELAY_GROUP in the UCF. See the *Virtex-6 FPGA User Guide* and code comments for more information.

In addition the following UCF syntax is included:

```
INST "rgmii_rxd<?>"  TNM = "rgmii_rx";
INST "rgmii_rx_ctl"  TNM = "rgmii_rx";
TIMEGRP "rgmii_rx" OFFSET = IN 1 ns VALID 2 ns BEFORE "rgmii_rxc" RISING;
TIMEGRP "rgmii_rx" OFFSET = IN 1 ns VALID 2 ns BEFORE "rgmii_rxc" FALLING;
```

This syntax causes the Xilinx implementation tools to analyze the input setup and hold constraints for the input RGMII bus. If these constraints are not met, the tools report timing errors. However, the tools do NOT attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the IDELAY_VALUE in the UCF.

Understanding Timing Reports for RGMII Setup/Hold Timing

Setup and Hold results for the RGMII input bus can be found in the data sheet section of the Timing Report. The results are self-explanatory and it is easy to see how they relate to Figure 14-2. Here follows an example for the RGMII report.

Each Input lists two sets of values: one corresponding to the falling edge of the clock and one to the rising edge. The first set listed corresponds to rising edge which occurs at time 0 ns.

Setup: the implementation requires 0.848 ns of setup to the rising edge and 0.848 ns to the falling edge: this is less than the 1 ns required and so there is slack.

Hold: the implementation requires 0.727 ns of hold after the rising edge and 0.724 ns after the falling edge; this is less than the 1ns required and so there is slack.

```
Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock rgmii_rxc
------------+------------+------------+------------------+--------+
            |Max Setup to|Max Hold to |                  | Clock  |
Source      | clk (edge) | clk (edge) |Internal Clock(s) | Phase  |
------------+------------+------------+------------------+--------+
rgmii_rx_ctl|    0.848(R)|    0.727(R)|rgmii_rx_clk_bufio|   0.000|
            |   -3.151(F)|    4.724(F)|rgmii_rx_clk_bufio|   4.000|
rgmii_rxd<0>|    0.826(R)|    0.746(R)|rgmii_rx_clk_bufio|   0.000|
            |   -3.173(F)|    4.743(F)|rgmii_rx_clk_bufio|   4.000|
rgmii_rxd<1>|    0.852(R)|    0.724(R)|rgmii_rx_clk_bufio|   0.000|
            |   -3.147(F)|    4.721(F)|rgmii_rx_clk_bufio|   4.000|
rgmii_rxd<2>|    0.856(R)|    0.720(R)|rgmii_rx_clk_bufio|   0.000|
            |   -3.143(F)|    4.717(F)|rgmii_rx_clk_bufio|   4.000|
rgmii_rxd<3>|    0.871(R)|    0.704(R)|rgmii_rx_clk_bufio|   0.000|
            |   -3.128(F)|    4.701(F)|rgmii_rx_clk_bufio|   4.000|
------------+------------+------------+------------------+--------+
```

XILINX®

Chapter 15

Implementing Your Design

This chapter describes how to simulate and implement your design containing the V6EMAC core.

Pre-implementation Simulation

The CORE Generator™ software generates a functional model of the core netlist to allow simulation of the block in the design phase of the project.

Using the Simulation Model

For information on setting up your simulator to use the functional model, see the *Xilinx Synthesis and Simulation Design Guide* [Ref 12], also included in your Xilinx software installation.

The model is provided in the CORE Generator software project directory.

VHDL

```
<component_name>.vhd
```

Verilog

```
<component_name>.v
```

This model can be compiled along with your code to simulate the overall system.

Synthesis

XST - VHDL

In the CORE Generator software project directory, there is a `<component_name>.vho` file that is a component and instantiation template for the core. Use this to help instance the core into your VHDL source. After your entire design is complete, create the following:

- An XST project file `top_level_module_name.prj` listing all your source code files
- An XST script file `top_level_module_name.scr` containing your required synthesis options

To synthesize the design, run:

```
$ xst -ifn top_level_module_name.scr
```

See the *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices* [Ref 13] for more information on creating project and synthesis script files, and running the `xst` program.

XST - Verilog

In the CORE Generator software project directory, locate the module declaration for the core at:

```
<component_name>/implement/<component_name>_mod.v
```

Use this module to help instance the core into your Verilog source.

After your entire design is complete, create

- An XST project file *top_level_module_name*`.prj` listing all your source code files. Be sure to include

  ```
  %XILINX%/verilog/src/iSE/unisim_comp.v
  ```

  and

  ```
  <component_name>/implement/component_name_mod.v
  ```

  as the first two files in the project list.
- An XST script file *top_level_module_name*`.scr` containing your required synthesis options

To synthesize the design, run

```
$ xst -ifn top_level_module_name.scr
```

See the *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices* [Ref 13] for more information on creating project and synthesis script files and running the `xst` program.

Implementation

Generating the Xilinx Netlist

To generate the Xilinx netlist, the ngdbuild tool is used to translate and merge the individual design netlists into a single design database, the NGD file. Also merged at this stage is the UCF for the design. An example of the ngdbuild command is:

```
$ ngdbuild -sd path_to_core_netlist -sd path_to_user_synth_results \
-uc top_level_module_name.ucf top_level_module_name
```

Mapping the Design

To map the logic gates of your design netlist into the CLBs and IOBs of the FPGA, run the `map` command. The `map` command writes out a physical design to an NCD file. An example of the `map` command is:

```
$ map -ol high -timing top_level_module_name \
-o top_level_module_name_map.ncd
```

Placing and Routing the Design

To place and route your design's logic components (mapped physical logic cells) contained within an NCD file in accordance with the layout and timing requirements specified

within the PCF file, the `par` command must be executed. The par command outputs the placed and routed physical design to an NCD file. An example of the par command is:

```
$ par -ol high -w top_level_module_name_map.ncd \
top_level_module_name.ncd mapped.pcf
```

Static Timing Analysis

To evaluate timing closure on a design and create a Timing Report file (TWR) derived from static timing analysis of the Physical Design file (NCD), the `trce` command must be executed. The analysis is typically based on constraints included in the optional PCF file. An example of the `trce` command is:

```
$ trce -o top_level_module_name.twr top_level_module_name.ncd \
mapped.pcf
```

Generating a Bitstream

To create the configuration bitstream (BIT) file based on the contents of a physical implementation file (NCD), the `bitgen` command must be executed. The BIT file defines the behavior of the programmed FPGA. An example of the `bitgen` command is:

```
$ bitgen -w top_level_module_name.ncd
```

Post-Implementation Simulation

The purpose of post-implementation simulation is to verify that the design as implemented in the FPGA works as expected.

Generating a Simulation Model

To generate a chip-level simulation netlist for your design, run the netgen command.

VHDL

```
$ netgen -sim -ofmt vhdl -ngm top_level_module_name_map.ngm \
-tm netlist top_level_module_name.ncd \
top_level_module_name_postimp.vhd
```

Verilog

```
$ netgen -sim -ofmt verilog -ngm top_level_module_name_map.ngm \
-tm netlist top_level_module_name.ncd \
top_level_module_name_postimp.v
```

Using the Model

For information on setting up your simulator to use the pre-implemented model, consult the Xilinx *Synthesis and Simulation Design Guide* [Ref 12], included in your Xilinx software installation.

Other Implementation Information

For more information about using the Xilinx implementation tool flow, including command line switches and options, see the Xilinx ISE® software manuals.

Chapter 16

Quick Start Example Design

This chapter provides instructions for generating the V6EMAC example design using the default parameters.

Overview

The V6EMAC example design consists of the following:

- A V6EMAC netlist
- An HDL example design
- A demonstration test bench to exercise the example design

Figure 16-1 shows the block diagram for the example design and the test bench provided with the V6EMAC. The example design has been tested with Xilinx ISE® software v14.1, Cadence Incisive Enterprise Simulator, Mentor Graphics ModelSim, and Synopsys VCS and VCS MX (see the ISE Design Suite 14: Release Notes Guide for the tool version).

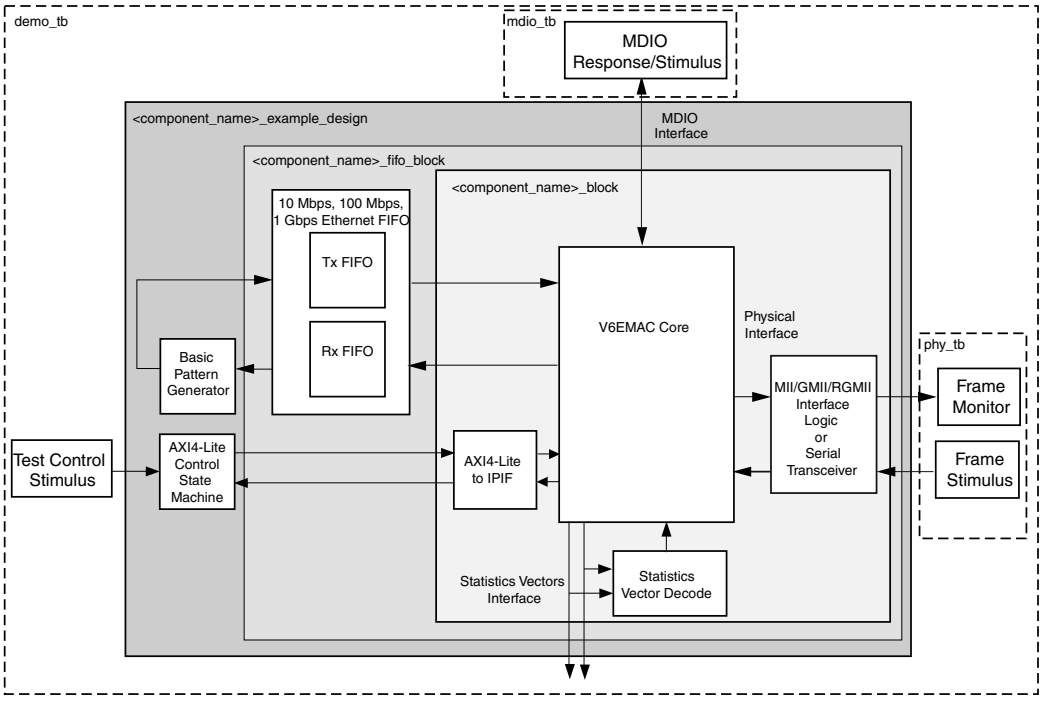

Figure 16-1: **Default Example Design and Test Bench**

Generating the Core

To begin using the V6EMAC example design, start by generating the core.

1. Start the CORE Generator™ software.

   For help starting and using the CORE Generator software, see the documentation supplied with the ISE software, including the *CORE Generator Guide* at www.xilinx.com/support/software_manuals.htm.

2. Choose File > New Project.
3. Do the following to set project options:
   - From Target Architecture, select a Virtex®-6 device.

   Note: If an unsupported silicon family is selected, the core does not appear in the taxonomy tree. For a list of supported architectures, see the *Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Data Sheet* [Ref 14].

   - For Design Entry, select either VHDL or Verilog; for Vendor, select Other.
4. After creating the project, locate the directory containing the core in the taxonomy tree. The project appears under one of the following:
   - Communications & Networking /Ethernet
   - Communications & Networking /Networking
   - Communications & Networking/Telecommunications
5. Double-click the core.
6. Click OK to exit the dialog box. The core customization screen appears (Figure 16-2).
7. In the Component Name field, enter a name for the core instance.
8. Accept the remaining defaults and click Generate to generate the core.
9. The core and its supporting files, including the example design, are generated in your project directory. For a detailed description of the design example files and directories, see Chapter 17, Detailed Example Design.

After the core is generated, a functional simulation directory is created, which contains scripts to simulate the core using the structural HDL models. See Functional Simulation.

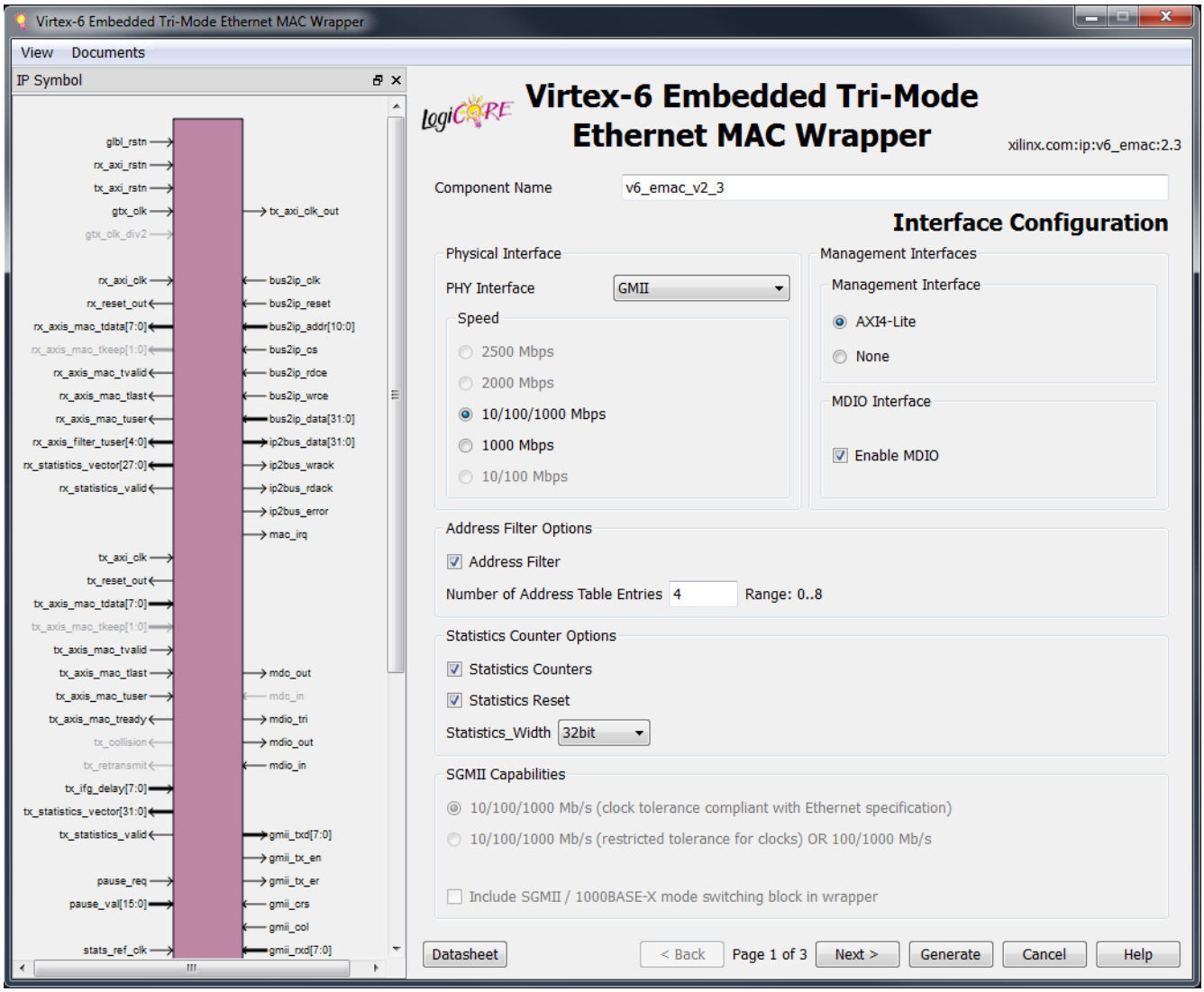

Figure 16-2: **Virtex-6 Embedded Tri-Mode Ethernet MAC Main Screen**

Implementing the Example Design

The netlist and HDL example design can be processed through the Xilinx implementation toolset. The generated output files include several scripts to assist you in running the Xilinx software.

Note: In the following examples, *<project_dir>* is the CORE Generator software project directory and *<component_name>* is the name entered in the customization dialog box.

Open a command prompt or shell in your project directory, then enter the following commands:

Linux

```
% cd <component_name>/implement
% ./implement.sh
```

Windows

```
ms-dos> cd <component_name>\implement
ms-dos> implement.bat
```

These commands execute a script that synthesizes, builds, maps, and place-and-routes the example design together with the core. The script also generates a gate-level model of the example design and a netlist for use in timing simulation. The resulting files are placed in the results directory, which is created by the implement script at run time.

Running the Simulation

Functional Simulation

To run the functional simulation, you must have the Xilinx Simulation Libraries compiled for your system. See "Compiling Xilinx Simulation Libraries (COMPXLIB)" in the *Xilinx Synthesis and Simulation Design Guide* [Ref 12] which can be obtained from: www.xilinx.com/support/software_manuals.htm.

Note: In the simulation examples that follow, <*project_dir*> is the CORE Generator software project directory, and <*component_name*> is the component name as entered in the core customization dialog box.

Virtex-6 Devices

Virtex-6 device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. When running VHDL simulations, a mixed-language license is required.

VHDL Simulation

To run a VHDL functional simulation:

1. Open a command prompt or shell and set the current directory to:
   `<project_dir>/<component_name>/simulation/functional`
2. Launch the simulation script:

   ```
   ModelSim: vsim -do simulate_mti.do
   IES:./simulate_ncsim.sh
   ```

The scripts compile the structural VHDL model, the example design files and the demonstration test bench, add some relevant signals to a wave window, then run the simulation to completion. Now you can review the simulation transcript and waveform to observe the operation of the core.

Verilog Simulation

To run a Verilog functional simulation:

1. Open a command prompt or shell and set the current directory to
   `<project_dir>/<component_name>/simulation/functional`
2. Launch the simulation script:

   ```
   ModelSim: vsim -do simulate_mti.do
   IES: ./simulate_ncsim.sh
   VCS: ./simulate_vcs.sh
   ```

The scripts compile the structural Verilog model, the example design files and the demonstration test bench, add some relevant signals to a wave window, then run the simulation to completion. Now you can review the simulation transcript and waveform to observe the operation of the core.

Timing Simulation

To run the gate-level simulation, you must have the Xilinx Simulation Libraries compiled for your system. See" Compiling Xilinx Simulation Libraries (COMPXLIB)" in the *Xilinx Synthesis and Verification Design Guide*, which can be obtained from www.xilinx.com/support/software_manuals.htm.

Note: In the simulation examples that follow, *<project_dir>* is the CORE Generator software project directory; *<component_name>* is the component name entered in the core customization dialog box.

VHDL Simulation

To run a VHDL timing simulation:

1. Open a command prompt or shell and set the current directory to `<project_dir>/<component_name>/simulation/timing`
2. Launch the simulation script:

```
ModelSim: vsim -do simulate_mti.do
IES: ./simulate_ncsim.sh
```

The scripts compile the gate-level model and the demonstration test bench, add some relevant signals to a wave window, then run the simulation to completion. Now you can review the simulation transcript and waveform to observe the operation of the core.

Verilog Simulation

To run a Verilog timing simulation:

1. Open a command prompt or shell and set the current directory to `<project_dir>/<component_name>/simulation/timing`
2. Launch the simulation script:

```
ModelSim: vsim -do simulate_mti.do
IES: ./implement_ncsim.sh
VCS: ./implement_vcs.sh
```

The scripts compile the gate-level model and the demonstration test bench, add some relevant signals to a wave window, then run the simulation to completion. Now you can review the simulation transcript and waveform to observe the operation of the core.

XILINX®

Chapter 17

Detailed Example Design

This chapter provides detailed information about the example design, including a description of files and the directory structure generated by the Xilinx® CORE Generator™ software, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

When the XC6VLX240T-FF1156-1 part is targeted in the CORE Generator software, the generated V6EMAC example design is targeted for use on the Xilinx ML605 evaluation board.

The example design includes a basic state machine which, using the AXI4-Lite interface, brings up the external PHY and MAC to allow basic frame transfer. A Simple Frame Generator is also included that can be used to turn the a particular board into a packet generator. Basic control of the state machine, allowing, for example, MAC speed change, is achieved using push buttons and DIP switches on the board. See the board specific sections in Targeting the Example Design to a Board.

Example Design Directory Structure.

- <project directory>
  Top-level project directory; name is user-defined.
  - <project directory>/<component name>
    Core release notes file
    - <component name>/doc
      Product documentation
    - <component name>/example design
      Verilog and VHDL design files
      - example design/common
        Files for general use in the example design
      - example design/axi_ipif
        Files for the AXI4-Lite to IPIF interface that is instanced in the Block level
      - example design/axi_lite
        Files for the AXI4-Lite control state machine which is instanced in the top level example design

- example design/fifo
  Files for the FIFO that is instanced in the FIFO Block level
- example design/pat_gen
  Files for the Basic Pattern Generator which is instanced in the top level example design
- example_design/physical
  Files for the physical interface of the MAC
- example_design/statistics
  Files for the statistics vector decode logic

<component name>/implement
Implementation script files

- implement/results
  Results directory, created after implementation scripts are run, and contains implement script results

<component name>/simulation
Simulation scripts

- simulation/functional
  Functional simulation files
- simulation/timing
  Timing simulation files

Directory and File Contents

The core directories and associated files are defined in the following sections.

<project directory>

The project directory contains all the CORE Generator software project files.

Table 17-1: **Project Directory**

Name	Description
	<project_dir>
<component_name>.ngc	Binary Xilinx implementation netlist. Describes how the core is to be implemented. Used as input to the Xilinx Implementation Tools.
<component_name>.v[hd]	VHDL or Verilog structural simulation model. File used to support functional simulation of a core. The model passes customized parameters to the generic core simulation model.
<component_name>.xco	As an output file, the XCO file is a log file which records the settings used to generate a particular core. An XCO file is generated by the CORE Generator software for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_flist.txt	Text file that defines all the output files produced when a customized core is generated using the CORE Generator software.
<component_name>.{veo|vho}	Verilog or VHDL template for the core. This can be copied into your design.

Back to Top

<project directory>/<component name>

The <component name> directory contains the release notes file provided with the core, which can include last-minute changes and updates.

Table 17-2: **Component Name Directory**

Name	Description
	<project_dir>/<component_name>
v6_emac_readme.txt	Core release notes file

Back to Top

<component name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 17-3: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
ds835_v6_emac.pdf	Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper Data Sheet
ug800_v6_emac.pdf	Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper User Guide

Back to Top

<component name>/example design

This directory (and subdirectories) contain all of the support files necessary for a VHDL or Verilog implementation of the example design. Table 17-4 defines the HDL files that are always present in this directory. Example designs for certain implementations can contain extra files for clock/clock enable generation logic.

See Example Design for more information.

Table 17-4: **Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_example_design.v[hd]	Top-level VHDL or Verilog file for the example design. This instantiates the FIFO block level along with the basic pattern generator block, providing a simple loopback function or frame generation.
<component_name>_example_design.ucf	User constraints file (UCF) for the core and the example design
<component_name>_fifo_block.v[hd]	Example design with an AXI4-Stream interface. This instantiates the block level V6EMAC wrapper together with a receive and a transmit FIFO.
<component_name>_block.v[hd]	Block-level V6EMAC wrapper containing the core and all clocking and physical interface circuitry
<component_name>_mod.v	Module declaration for the core instance in the example design, generated for Verilog core configurations.
clk_wiz.v[hd]	Simple clock generation block, generated for core configurations which use the MII, GMII, or RGMII physical interface

Back to Top

example design/common

This directory contains common files required by various levels of the example design.

Table 17-5: **Common Directory**

Name	Description
<project_dir>/<component_name>/example_design/common	
sync_block.v[hd]	Synchronizer module, used for passing signals across a clock domain.
reset_sync.v[hd]	Local reset synchronizer, used to create a synchronous reset output signal from an asynchronous input.

Back to Top

example design/axi_ipif

This directory contains the files for the AXI4_IPIF interface that is instanced in the Block level of the example design.

Table 17-6: **Axi_ipif Directory**

Name	Description
<project_dir>/<component_name>/example_design/axi_ipif	
axi4_lite_ipif_wrapper.v[hd]	Top Level wrapper for the AXI4-Lite to IPIF interface. This simplifies the required parameters to just the required base address.
axi_lite_ipif.v[hd]	AXI4-Lite IPIF wrapper block. converts from the industry standard AXI4-Lite to a simple IPIF interface.
slave_attachment.v[hd]	Required file for the AXI_Lite_IPIF block.
address_decoder.v[hd]	Required file for the AXI_Lite_IPIF block.
counter_f.v[hd]	Required file for the AXI_Lite_IPIF block.
pselect_f.v[hd]	Required file for the AXI_Lite_IPIF block.
ipif_pkg.vhd	Only required for VHDL projects. Provides entity declarations of the VHDL files required by this block.

Back to Top

example design/axi_lite

This directory contains the files for the AXI4-Lite controller that is instanced in the example design when the optional management interface is selected.

Table 17-7: **Axi_lite Directory**

Name	Description
<project_dir>/<component_name>/example_design/axi_lite	
axi_lite_sm.v[hd]	Simple state machine to bring up the PHY (if any) and MAC ready for frame transfer.

Back to Top

example design/fifo

This directory contains the files for the FIFO that is instanced in the fifo_block example design.

Table 17-8: **FIFO Directory**

Name	Description
<project_dir>/<component_name>/example_design/fifo	
tx_client_fifo_[8|16].v[hd]	Transmit FIFO. This takes data from the user in AXI4-Stream format, stores it and sends it to the MAC.
rx_client_fifo_[8|16].v[hd]	Receive FIFO. This reads in and stores data from the MAC before outputting it to the user in AXI4-Stream format.
ten_100_1G_eth_fifo.v[hd]	FIFO top level. This instantiates the transmit and receive FIFOs.

Back to Top

example design/pat_gen

This directory contains the files for the basic pattern generator that is instanced in the example design.

Table 17-9: **Pat_gen Directory**

Name	Description
<project_dir>/<component_name>/example_design/pat_gen	
basic_pat_gen.v[hd]	Top level for the basic pattern generator block
axi_mux.v[hd]	Simple mux to allow the choice between the axi_pat_gen or the AXI4-Stream RX datapath. Provides basic loopback functionality under control of a dedicated input.
axi_pipe.v[hd]	Simple axi4-stream pipeline stage
axi_pat_gen[_16].v[hd]	Simple pattern generator. Generates packets of a defined size/range of sizes with the (parameter) specified DA and SA fields. The frame content is simple incrementing data.
axi_pat_check[_16].v[hd]	Simple pattern checker. Checks packets for size, DA field, SA field, and data contents based on expected pattern from pattern generator.
address_swap[_16].v[hd]	Allows the frame sourced by the axi_mux block to have the DA and SA fields swapped. This is controlled using a dedicated input.

Back to Top

example_design/physical

This directory contains a file for the physical interface of the MAC. A GMII, MII or RGMII version is delivered by the CORE Generator software, depending on the selected option.

Table 17-10: **Physical Directory**

Name	Description
<project_dir>/<component_name>/example_design/physical	
mii_if.v[hd]	For MII only: all clocking and logic required to provide an MII physical interface
gmii_if.v[hd]	For GMII only: all clocking and logic required to provide a GMII physical interface
rgmii_v2_0_if.v[hd]	For RGMII only: all clocking and logic required to provide a RGMII v2.0 physical interface
v6_gtxwizard_top.v[hd]	For SGMII/1000BASE-X PCS/PMA only: top level wrapper for the GTX transceiver instantiation.
v6_gtxwizard_gtx.v[hd]	For SGMII/1000BASE-X PCS/PMA only: GTX transceiver instantiation.
v6_gtxwizard.v[hd]	For SGMII/1000BASE-X PCS/PMA only: GTX transceiver instantiation.
double_reset.v[hd]	For SGMII/1000BASE-X PCS/PMA only: Reset logic for GTX transceiver
rx_elastic_buffer.v[hd]	For SGMII only: FPGA logic-based elastic buffer.
v6_gtxwizard.xco	Xilinx Core Options file for GTX Serial Transceiver Wizard wrapper generation.

Back to Top

example_design/statistics

This directory contains the statistics counters decode logic which is required when the core is build with the statistics core included.

Table 17-11: **Statistics Directory**

Name	Description
<project_dir>/<component_name>/example_design/statistics	
vector_decode.v[hd]	This block translates between the MAC source statistics vectors and the required counter increment signals. This is provided to allow user customization of the counters.

Back to Top

<component name>/implement

This directory contains the support files necessary for implementation of the example design with the Xilinx tools. See Example Design. Execution of an implement script results in creation of the results directory and an xst project directory.

Table 17-12: **Implement Directory**

Name	Description
<project_dir>/<component_name>/implement	
implement.sh	Linux shell script that processes the example design through the Xilinx tool flow
implement.bat	Windows batch file that processes the example design through the Xilinx tool flow
xst.prj	XST project file for the example design; it enumerates all the HDL files that need to be synthesised.
xst.scr	XST script file for the example design

Back to Top

implement/results

This directory is created by the implement scripts and is used to run the example design files and the <component_name>.ngc file through the Xilinx implementation tools. On completion of an implement script, this directory contains the following files for timing simulation. Output files from the Xilinx implementation tools are also located in this directory.

Table 17-13: **Results Directory**

Name	Description
<project_dir>/<component_name>/implement/results	
routed.v[hd]	Back-annotated SIMPRIM based gate-level VHDL or Verilog design. Used for timing simulation.
routed.sdf	Timing information for simulation

Back to Top

<component name>/simulation

The simulation directory and the subdirectories below it provide the files necessary to test a VHDL or Verilog implementation of the example design.

Table 17-14: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
demo_tb.v[hd]	VHDL or Verilog demonstration test bench top level for the V6EMAC
mdio_tb.v[hd]	MDIO test bench component
phy_tb.v[hd]	PHY specific test bench component

Back to Top

simulation/functional

The functional directory contains functional simulation scripts provided with the core.

Table 17-15: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	ModelSim macro file that compiles the example design sources and the structural simulation model then runs the functional simulation to completion.
wave_mti.do	ModelSim macro file that opens a wave window and adds interesting signals to it. It is called by the simulate_mti.do macro file.
simulate_ncsim.sh	Cadence IES script file that compiles the example design sources and the structural simulation model and then runs the functional simulation to completion.
wave_ncsim.sv	Cadence IES macro file that opens a wave window and adds interesting signals to it.
simulate_vcs.sh	VCS script file that compiles the Verilog sources and runs the functional simulation to completion.
ucli_commands.key	This file is sourced by VCS at the start of simulation: it configures the simulator.
vcs_session.tcl	VCS macro file that opens a wave window and adds signals of interest to it. It is called by the simulate_vcs.sh script file.

Back to Top

simulation/timing

The timing directory contains timing simulation scripts provided with the core.

Table 17-16: **Timing Directory**

Name	Description
<project_dir>/<component_name>/simulation/timing	
`simulate_mti.do`	ModelSim macro file that compiles the VHDL gate-level model and demonstration test bench then runs the timing simulation to completion.
`wave_mti.do`	ModelSim macro file that opens a wave window and adds interesting signals to it. It is called used by the simulate_mti.do macro file.
`simulate_ncsim.sh`	Linux shell script that compiles the VHDL gate-level model and demonstration test bench and then runs the timing simulation to completion using Cadence IES.
`wave_ncsim.sv`	IES macro file that opens a wave window and adds interesting signals to it. It is used by the simulate_ncsim.sh script.
`simulate_vcs.sh`	VCS script file that compiles the Verilog sources and runs the timing simulation to completion.
`ucli_commands.key`	This file is sourced by VCS at the start of simulation: it configures the simulator.
`vcs_session.tcl`	VCS macro file that opens a wave window and adds signals of interest to it. It is called by the `simulate_vcs.sh` script file.

Back to Top

Implementation and Test Scripts

Implementation Scripts for Timing Simulation

When the CORE Generator software is run, an implement script is generated in the `<project_dir>/<component_name>/implement` directory. The implementation script is either a shell script or batch file that processes the example design through the Xilinx tool flow.

Linux

```
<project_dir>/<component_name>/implement/implement.sh
```

Windows

```
<project_dir>/<component_name>/implement/implement.bat
```

The implement script performs the following steps:

- The HDL example design is synthesized using XST.
- NGDBuild is run to consolidate the core netlist and the HDL example netlist into the NGD file containing the entire design.
- The design is mapped to the target technology.
- The design is place-and-routed on the target device.
- Static timing analysis is performed on the routed design using trce.
- A bitstream is generated.
- Netgen runs on the routed design to generate VHDL and Verilog netlists and timing information in the form of SDF files.

The Xilinx tool flow generates several output and report files. These are saved in the following directory which is created by the implement script:

```
<project_dir>/<component_name>/implement/results
```

Test Scripts For Functional Simulation

The test script that automates the simulation of the test bench is located in one of the following locations:

Mentor ModelSim

```
<project_dir>/<component_name>/simulation/functional/simulate_mti.do
```

Cadence IES

```
<project_dir>/<component_name>/simulation/functional/simulate_ncsim.sh
```

Synopsys VCS

```
<project_dir>/<component_name>/simulation/functional/simulate_vcs.sh
```

The test script performs the following tasks:

- Compiles the structural simulation model of the core
- Compiles the example design files
- Compiles the demonstration test bench

- Starts a simulation of the test bench with no timing information
- Opens a Wave window and adds some signals of interest
- Runs the simulation to completion

Test Scripts For Timing Simulation

The test script that automates the simulation of the test bench is located at the following locations:

Mentor ModelSim

```
<project_dir>/<component_name>/simulation/timing/simulate_mti.do
```

Cadence IES

```
<project_dir>/<component_name>/simulation/timing/simulate_ncsim.sh
```

Synopsys VCS

```
<project_dir>/<component_name>/simulation/timing/simulate_vcs.sh
```

The test script performs the following tasks:

- Compiles the gate-level model of the example design
- Compiles the demonstration test bench
- Starts a simulation of the test bench using timing information
- Opens a Wave window and adds some signals of interest
- Runs the simulation to completion

Example Design

HDL Example Design

Figure 17-1 illustrates the top-level design for the V6EMAC example design.

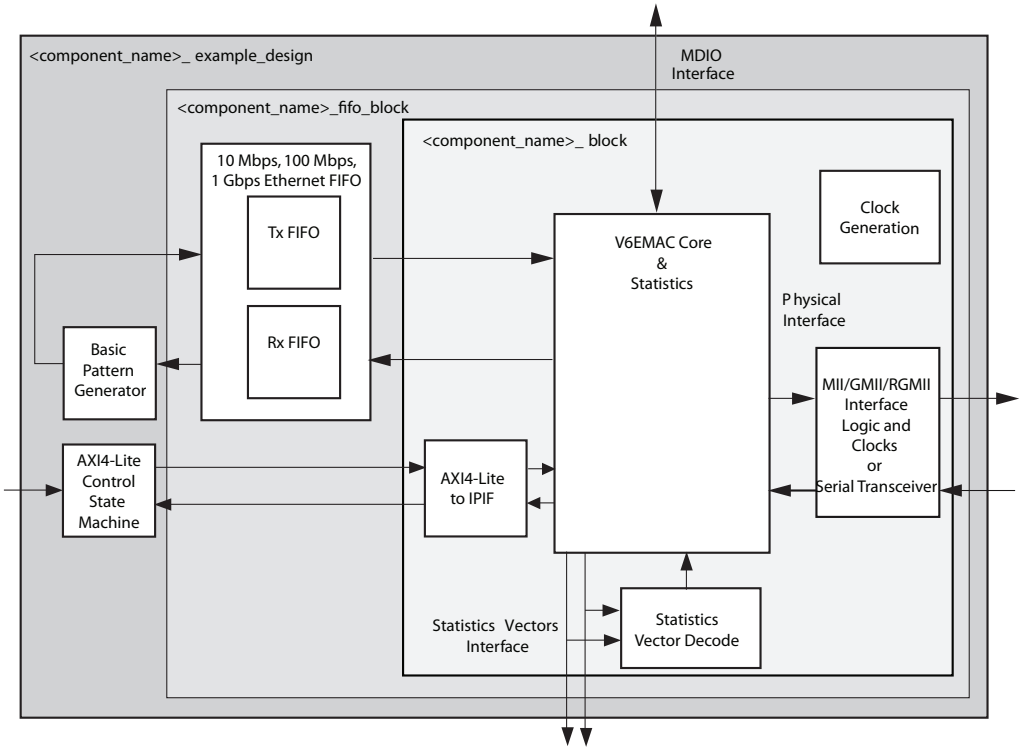

Figure 17-1: **HDL Example Design**

The top-level example design for the V6EMAC is defined in the following files:

```
<project_dir>/<component_name>/example_design/
<component_name>_example_design.v[hd]
```

The HDL example design contains the following:

- An instance of the V6EMAC
- Clock management logic, including MMCM and Global Clock Buffer instances, where required
- MII, GMII, RGMII, SGMII or 1000BASE-X PCS/PMA interface logic, including IOB and DDR registers instances, where required
- Statistics vector decode logic
- AXI4-Lite to IPIF interface logic
- User Transmit and Receive FIFOs with AXI4-Stream interfaces
- User Basic pattern generator module that performs address swapping and loopback
- A simple state machine to bring up the PHY (if any) and MAC ready for frame transfer

The HDL example design provides basic loopback functionality on the user side of the V6EMAC and connects the PHY interface to external IOBs, if appropriate. This allows the functionality of the core to be demonstrated either using a simulation package, as discussed in this guide, or in hardware, if placed on a suitable board. The simple state machine assumes standard PHY address and register content as per standard Xilinx demonstration boards.

10 Mb/s / 100 Mb/s / 1 Gb/s Ethernet FIFO

The 10 Mb/s/100 Mb/s/1 Gb/s Ethernet FIFO is described in the following files:

```
<project_dir>/<component_name>/example_design/fifo/ten_100_1g_eth_fifo.v[hd]
<project_dir>/<component_name>/example_design/fifo/tx_client_fifo_[8|16].v[hd]
<project_dir>/<component_name>/example_design/fifo/rx_client_fifo_[8|16].v[hd]
```

The 10 Mb/s/100 Mb/s/1 Gb/s Ethernet FIFO contains an instance of `tx_client_fifo` to connect to the MAC TX AXI4-Stream interface, and an instance of the `rx_client_fifo` to connect to the MAC RX AXI4-Stream interface. Both transmit and receive FIFO components implement an AXI4-Stream user interface, through which the frame data can be read/written. Figure 17-2 illustrates a straightforward frame transfer across the user side AXI4-Stream interface.

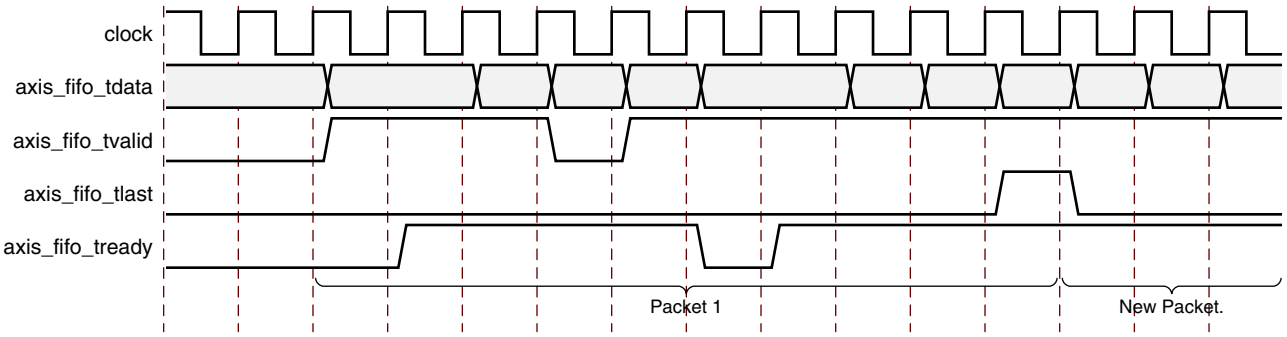

Figure 17-2: **Frame Transfer across AXI4-Stream Interface**

rx_client_fifo

The `rx_client_fifo` is built around two Dual Port Block RAMs, giving a total memory capacity of 4096 bytes. The receive FIFO writes in data received through the V6EMAC. If the frame is not errored, that frame is presented on the AXI4-Stream FIFO interface for reading by you, (in this case the `basic_pat_gen` module). If the frame is errored, that frame is dropped by the receive FIFO.

If the receive FIFO memory overflows, the frame currently being received is dropped, regardless of whether it is a good or bad frame, and the signal `rx_overflow` is asserted. Situations in which the memory can overflow are:

- The FIFO can overflow if the receiver clock is running at a faster rate than the transmitter clock or if the interpacket gap between the received frames is smaller than the interpacket gap between the transmitted frames. If this is the case, the tx FIFO is not able to read data from the rx FIFO as fast as it is being received.
- The FIFO size of 4096 bytes limits the size of the frames that it can store without error. If a frame is larger than 4000 bytes, the FIFO can overflow and data lost. It is therefore recommended that the example design is not used with the V6EMAC in jumbo frame mode for frames of larger than 4000 bytes.

tx_client_fifo

The `tx_client_fifo` is built around two Dual Port Block RAMs, giving a total memory capacity of 4096 bytes.

When a full frame has been written into the transmit FIFO, the FIFO presents data to the MAC transmitter. The MAC uses `tx_axis_mac_tready` to throttle the data until it has control of the medium.

If the FIFO memory fills up, the `tx_axis_fifo_tready` signal is used to halt the AXI4-Stream interface writing in data, until space becomes available in the FIFO. If the FIFO memory fills up but no full frames are available for transmission. For example if a frame larger than 4000 bytes is written into the FIFO, the FIFO asserts the `tx_overflow` signal and continues to accept the rest of the frame from you. The overflow frame is dropped by the FIFO. This ensures that the AXI4-Stream FIFO interface does not lock up.

Basic Pattern Generator Module

```
<project_dir>/<component_name>/example_design/pat_gen/basic_pat_gen.v[hd]
<project_dir>/<component_name>/example_design/pat_gen/axi_pat_gen[_16].v[hd]
<project_dir>/<component_name>/example_design/pat_gen/axi_mux.v[hd]
<project_dir>/<component_name>/example_design/pat_gen/axi_pipe.v[hd]
<project_dir>/<component_name>/example_design/pat_gen/address_swap[_16].v[hd]
```

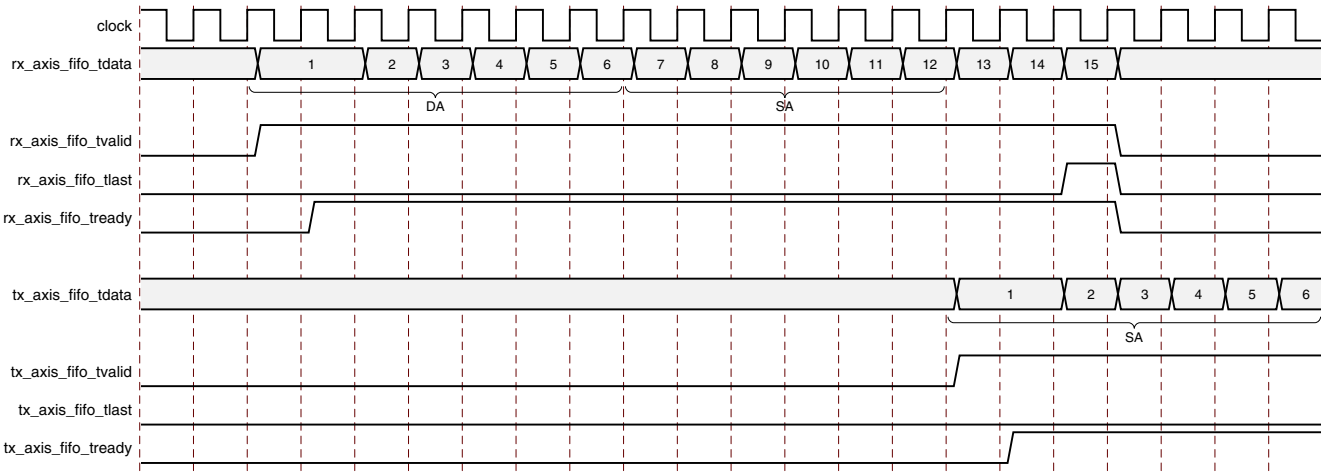

Figure 17-3: **Modification of Frame Data by Address Swap Module**

The basic pattern generator has two main functional modes. In loopback the data from the RX FIFO is passed to the address swap module and passed from their to the TX FIFO. The address swap module waits until both the DA and SA have been received before starting to send data on to the TX FIFO. If enabled, using a dedicated input, the module swaps the destination and source addresses of each frame as shown in Figure 17-3 to ensure that the outgoing frame destination address matches the source address of the link partner. The module transmits the frame control signals with an equal latency to the frame data.

The basic pattern generator can also operate as a simple pattern generator. In this case the data from the RX FIFO is ignored with frames being generated locally. These frames have fixed DA and SA fields (based on parameters) with the frame size incrementing between a supplied minimum and maximum value (also based on parameters). The frame content is a simple incrementing data pattern.

The choice between these two modes of operation is controlled using a dedicated input and can be changed dynamically.

AXI4-Lite Control State Machine

The AXI4-Lite state machine, which is present when the core is generated with AXI4-Lite support enabled, provides basic accesses to initialise the PHY and MAC to allow basic frame transfer.

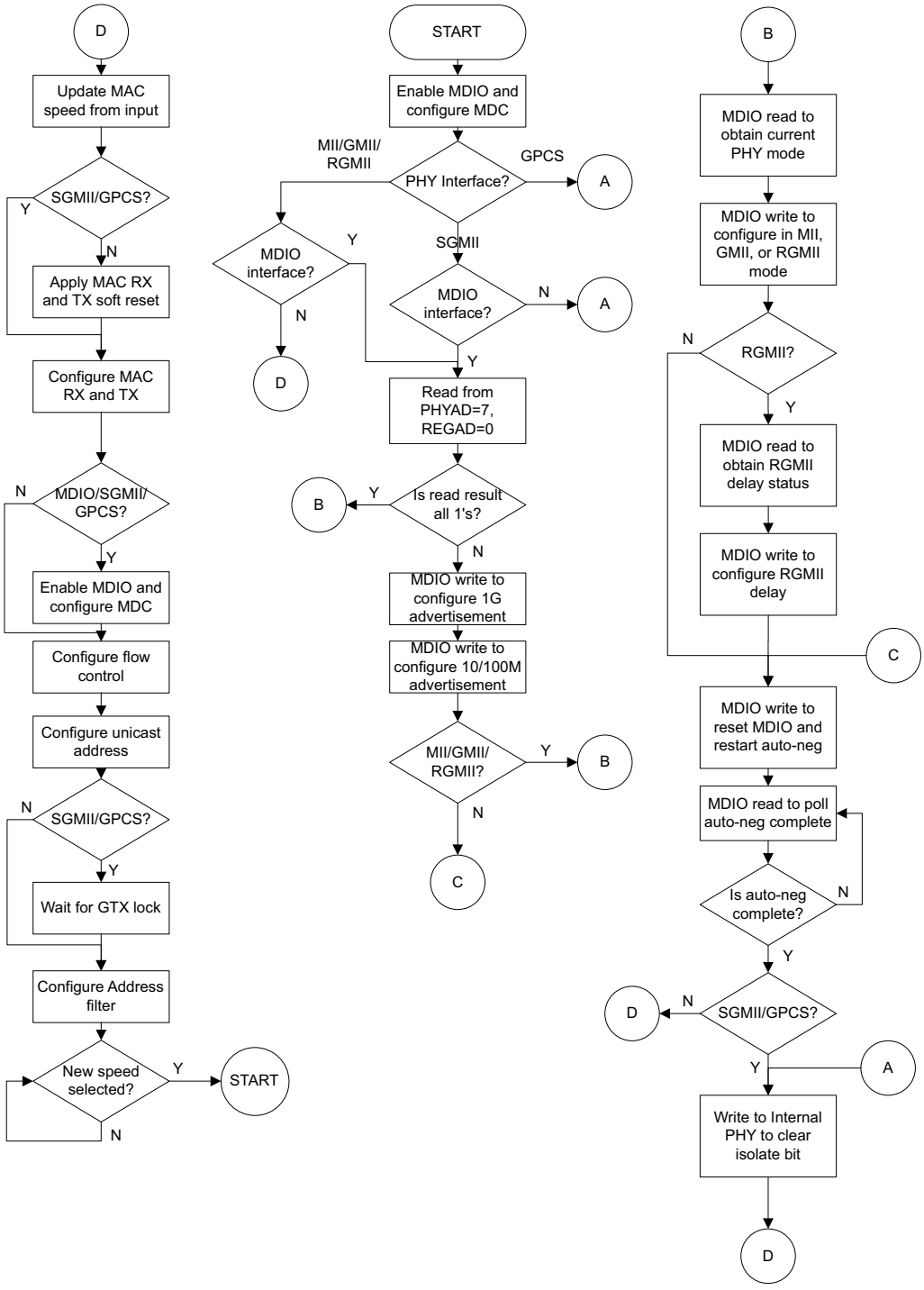

Figure 17-4: **State Machine Flow diagram**

Figure 17-4 shows the accesses performed by the state machine. After a reset, and allowing settling time for internal resets to complete, the state machine first writes to the MAC to enable the MDIO and configure the MDIO clock (this assumes an *s_axi_aclk* running at 100MHz). If an external MDIO interface is present an MDIO read is performed from PHYAD 7, which is the standard address used on Xilinx Demonstration Boards. If this returns all 1s it implies that no PHY exists at this location and further MDIO accesses are skipped.

This MDIO read enables the demonstration test bench to limit the number of MDIO accesses performed and reduce the run time of the simulations whilst still allowing the correct MDIO accesses to take place on a board. If the PHY is present, the MDIO read data has a value other than all 1s, the state machine then performs the necessary MDIO writes to configure the PHY speed advertisement as per the *mac_speed* inputs. If RGMII is selected a read-modify-write is performed (B) to select RGMII, avoiding the need to change jumper settings on the board. Finally the PHY is reset (C) and auto-negotiation restarted.

If either SGMII or 1000BASE-X PCS/PMA (GPCS) are used then the core contains an internal PHY which need to be enabled to allow data transfer. Therefore when auto negotiation completes (or directly after the initial MDIO configuration) a write is performed (A) to this PHY to initialise it.

The final step is to configure the MAC (B). First the MAC speed is updated, as per the *mac_speed* inputs. The MAC is then configured to disable flow control, initialize the unicast address and set the Frame Filter to promiscuous mode. If either SGMII or GPCs are used then the state machine wait for the GTX transceiver to lock. Finally the state machine sits and waits, if the update_speed input asserts it returns to the initial MDIO read state and the new *mac_speed* input is captured and applied.

With the state machine only applying a fixed core configuration, logic can be stripped during logic optimization. To avoid this, the state machine has a serial interface, *serial_command* and *serial_response*, which can be used to access any location and either perform a read or a write. This uncertainty prevents functions unused by the state machine from being stripped.

Demonstration Test Bench

Test Bench Functionality

The demonstration test bench is defined in the following files:

```
<project_dir>/<component_name>/simulation/demo_tb.v[hd]
```

The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself.

The test bench consists of the following:

- Clock generators
- The phy_tb component which connects to the appropriate PHY interface and both sources and monitors frames.
- A management block to control the speed selection
- An mdio_tb component containing a monitor and stimulus to check and respond to MDIO accesses, if a management interface is selected.

A control mechanism to manage the interaction of management, stimulus and monitor blocks.

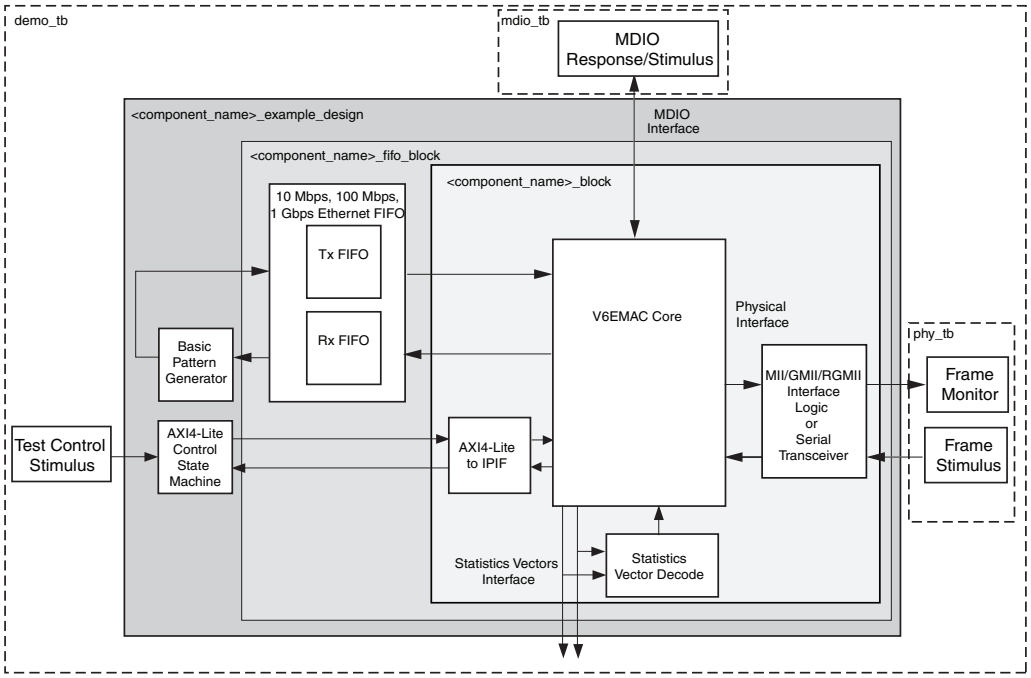

Figure 17-5: **Demonstration Test Bench**

Core with Management Interface

The demonstration test bench performs the following tasks:

1. Input clock signals are generated.
2. A reset is applied to the example design.
3. The required speed is selected using mac_speed and update_speed
4. The MDIO stimulus/response block responds to a read with all 1s - to indicate no PHY is present.
5. If a Serial Transceiver is present then a delay is added, followed by a sync check to ensure the GTX transceiver is ready. Otherwise, the test bench waits for the complete indication from the example design.
6. Four frames are pushed into the PHY receiver interface at the fastest MAC speed supported:
   - The first frame is a minimum length frame
   - The second frame is a type frame
   - The third frame is an errored frame
   - The fourth frame is a padded frame
7. The frames received at the PHY transmitter interface are checked against the stimulus frames to ensure data is the same. The monitor process takes into account the source/destination address field and FCS modifications resulting from the address swap module.
8. If either the 10/100/1000 Mb/s or 10/100 Mb/s configurations have been selected, mac_speed is updated to run at the next fastest available speed. This is 100 Mb/s or 10 Mb/s respectively. Update_speed is then pulsed.

9. The MDIO stimulus/response block responds to a read with all 1s - to indicate no PHY is present.
10. The same four frames are then sent to the PHY interface and checked against the stimulus frames.
11. If the 10/100/1000 Mb/s configuration has been selected, mac_speed is updated to run at 10 Mb/s. Update_speed is then pulsed.
12. The MDIO stimulus/response block responds to a read with all 1s - to indicate no PHY is present.
13. The same four frames are then sent to the PHY interface and checked against the stimulus frames.
14. If the BASE-X SWITCH feature is enabled then it is switched to the 1000BASE-X mode.
15. Steps 2 to 7 are then repeated
16. The BASE-X SWITCH is again switched back to SGMII and steps 2 to 13 are repeated.

Core with No Management Interface

The demonstration test bench performs the following tasks:

1. Input clock signals are generated
2. A reset is applied to the example design
3. The Core speed is selected - this is primarily to ensure the test bench generates data at the correct rate as the speed is set using an attribute and cannot be altered.
4. If either SGMII or GPCS are selected and the MDIO enabled then the mdio_tb drives the MDIO interface to initialise the internal PHY.
5. If a Serial Transceiver is in use then a delay is added and its sync status checked.
6. The stimulus block pushes four frames into the GMII/MII or RGMII receiver interface at the fastest speed supported by the selected configuration:
   - The first frame is a minimum-length frame
   - The second frame is a type frame
   - The third frame is an errored frame
   - The fourth frame is a padded frame
7. The frames received at the GMII/MII or RGMII transmitter interface are checked against the stimulus frames to ensure data is the same. The monitor process takes into account the source/destination address field and FCS modifications resulting from the address swap module.

Changing the Test Bench

Changing Frame Data

The contents of the frame data passed into the V6EMAC receiver can be changed by editing the DATA fields for each frame defined in the test bench. The test bench automatically calculates the new FCS field to pass into the V6EMAC, as well as calculating the new expected FCS value. Further frames can be added by defining a new frame of data and adjusting the stimulus and monitor processes to iterate around the correct number of frames.

Changing Frame Error Status

Errors can be inserted into any of the pre-defined frames by changing the error field to 1 in any column of that frame.

When an error is introduced into a frame, the `bad_frame` field for that frame must be set to disable the monitor checking for that frame.

The error currently written into the third frame can be removed by setting all error fields for the frame to 0 and unsetting the `bad_frame` field.

Targeting the Example Design to a Board

When the XC6VLX240T-FF1156-1 part is selected in the CORE Generator software, the generated V6EMAC example design is targeted for use on the Xilinx ML605 evaluation board. The UCF provided with the example design provides the required pin placements for the board. The board DIP switches, push buttons and LEDs are used to provide basic control over the V6EMAC functionality. This is described in more detail in Board Specific Features.

The example design also include certain features which are only used when it is targeted to a board, as described in Board Specific Features.

V6EMAC Configurations supported

There are some basic requirements for the example design to function correctly when targeted to the ML605 evaluation board. The V6EMAC must:

- Include an AXI4-Lite Management Interface
- Target the XC6VLX240T-FF1156-1 part

Board Specific Features

Pattern Generator

The *basic_pat_gen* module, which is instantiated at the top level of the example design, can be used as an optional source for the *address_swap* module. This module contains the *axi_pat_gen* module and the *address_swap* module along with a mux, pipeline and *axi_pat_check* module (described separately).

This pattern generator can be enabled/disabled using a DIP switch. When enabled, the data from the RX FIFO is flushed and the *axi_pat_gen* module drives the *address_swap* module's inputs. When disabled the RX FIFO is used as the data source for the *address_swap* module.

The pattern generator allows user modification of the Destination Address, Source Address, minimum frame size and maximum frame size through the use of parameters. When enabled it starts with the minimum frame size and after each frame is sent, increments the frame size until the maximum value is reached; it then starts again at the minimum frame size.

In all cases the Destination and Source address are as provided by the parameters, with the Type/Length field being dependant upon the frame size and the frame data being a decrementing count starting from the value in the type/length field. This should mean that the final data byte in all frames is 0x1. This is shown in Figure 17-6.

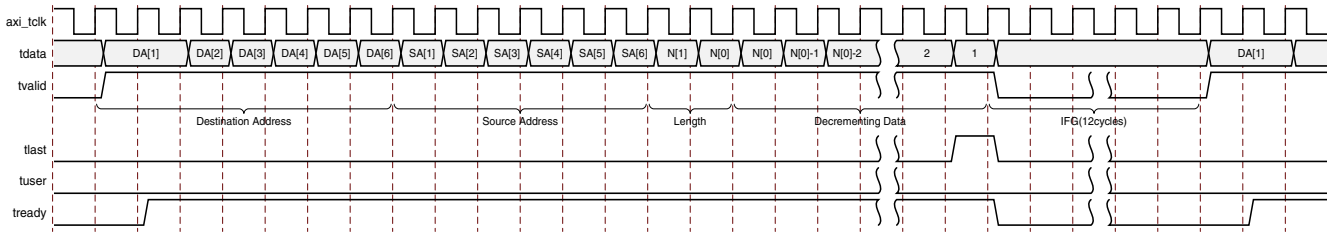

Figure 17-6: **Pattern Generator Frame Structure**

The provided pattern generator transmits frames at the maximum rate supported. In a loopback scenario, the ppm difference between the oscillators on the two boards can cause overflows in the slower board, resulting in errors. This is normally observed when the slower board is operating as the loopback board.

Pattern Checker

The axi_pat_check module, which is a submodule of the `basic_pat_gen`, provides a simple sanity check that data is being received correctly. It uses the same parameters as the `axi_pat_gen` module and therefore expects the same frame contents and frame size increments.

When enabled, the output from the RX FIFO is monitored. The first step is to identify where in the frame sequence the data is, which is done by capturing the value in the type/length field. When this is done, the following frame is expected to be incrementally bigger (unless at the wrap point). If an error is detected, an error is raised on the byte or bytes which mismatch, and the error condition is latched and displayed by a board LED. The pattern checker state machine then re-syncs to the data. One of the push buttons is used to clear this latched error state, enabling a feel for the frequency of errors if any.

Bring up sequence

When the example design is first targeted to the ML605, the following sequence is suggested to check the various features are working:

- Attach an Ethernet cable between the board and a PC with Wireshark (or similar) installed
- Select the desired speed using the DIP switches
- Push the update speed push-button
- Ensure the link status LEDs indicate a link and the expected speed
- Enable the pattern generator using the DIP switch
- Capture and check the received frames at the PC and ensure they have the expected data pattern

Note: Ensure that the ML605 jumpers and DIP switches are in the required position as described by the *ML605 Hardware User Guide* [Ref 15]. For V6EMAC SGMII configurations, jumpers J66 and J67 placement differs from other PHY interfaces.

After the basic features have been checked for operation, it can be desirable to check the operation of both the TX and RX datapaths. Two use models are considered:

1. Using an ML605 as both frame source and checker

   This approach requires a link partner, which can be either another ML605 configured with the same bitstream or some other loopback device.

   Bring up process:

   - Attach an Ethernet cable between the ML605 frame source and its link partner
   - Select the desired speed on both the ML605 (using the DIP switches) and the link partner; this must be the same setting
   - Configure the link partner to loopback data to the ML605 frame source. When the link partner is another ML605, this is done by disabling the pattern generator using the DIP switch.
   - Update the speed on both boards - on the ML605, by pushing the update speed push-button
   - Ensure the link status LEDs indicate a link and the expected speed
   - Enable the pattern checker on the ML605 frame source using the DIP switch
   - Enable the pattern generator on the ML605 frame source using the DIP switch
   - Ensure that the Link Status, TX, and RX LEDs all light up

Note: The pattern checker expects that the Destination Address and Source Address are in the same format as provided by the frame generator, that is, not swapped. If the link partner is another ML605, both boards should similarly enable or disable the address swap using the DIP switch. If the link partner is some other loopback device, the ML605 frame source should disable the address swap using the DIP switch.

2. Using two ML605 for point-to-point checking

   This approach requires two ML605 boards, each acting as a frame source in one direction and as a pattern checker in the other direction. This approach does not loop data back.

   Bring up process:

   - Attach an Ethernet cable between the ML605 boards
   - Select the desired speed using the DIP switches on both boards
   - Push the update speed push-button on both boards
   - Ensure the link status LEDs indicate a link and the expected speed
   - Disable the address swap using the DIP switch on both boards
   - Enable the pattern checker using the DIP switch on both boards
   - Enable the pattern generator using the DIP switch on both boards
   - Ensure that the Link Status, TX, and RX LEDs all light up
   - Push the reset checker error push-button on both boards as necessary to flush the RX FIFO and continue checking under the point-to-point configuration

In these or any other use model, pattern errors can result from a ppm difference between the boards leading to FIFO overflow. If this occurs when using a loopback configuration with two ML605s, try switching which board is used as the frame source.

ML605 Board

The UCF targets the ML605 when any Virtex®-6 LX part is selected.

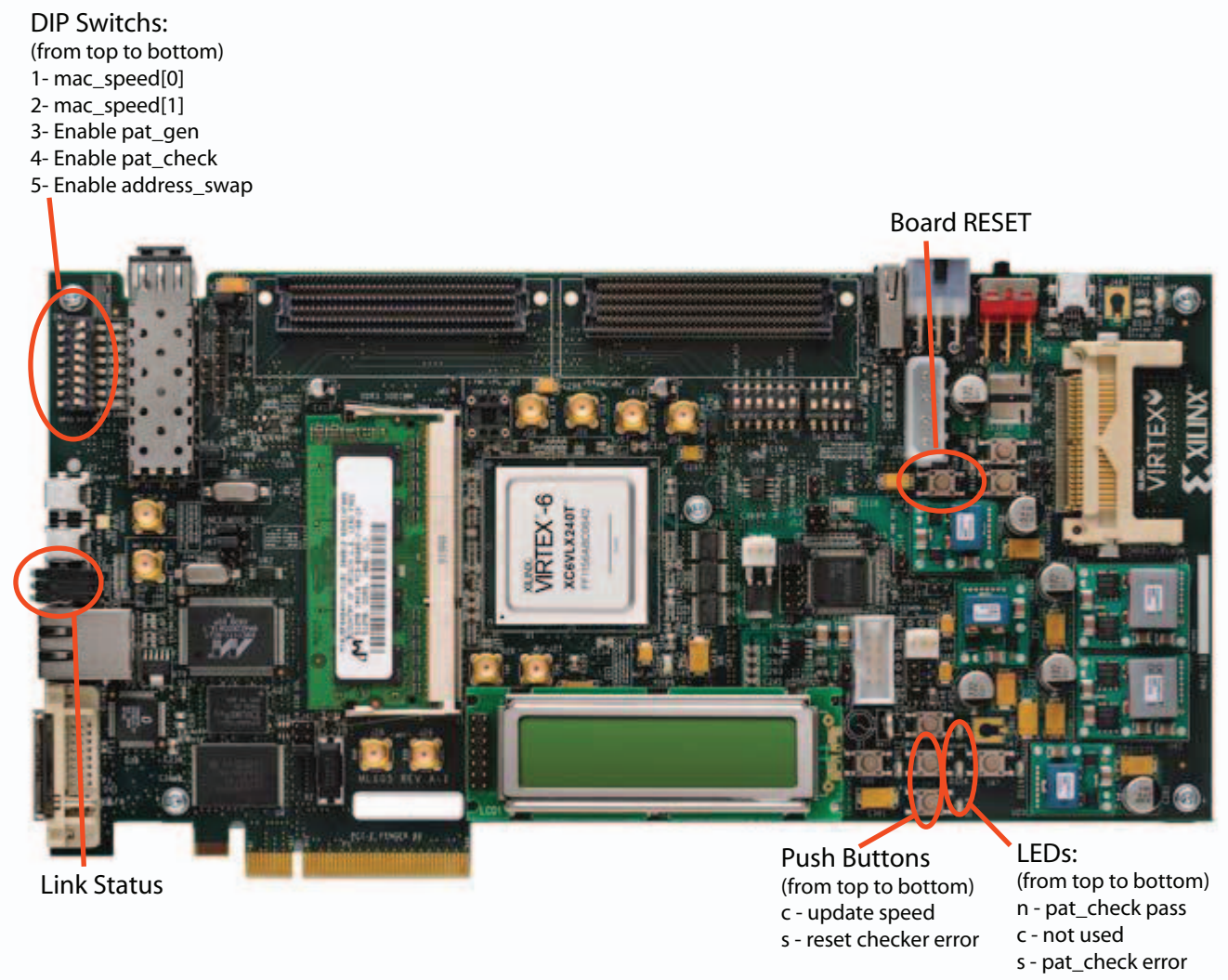

Figure 17-7: **ML605 Board Connectivity**

XILINX®

Appendix A

Calculating the MMCM Phase Shift or IODELAY Tap Setting

Two differing methods can be used by the core to meet input bus (GMII/MII or RGMII) setup and hold timing specifications. These are:

- MMCM Usage

  An MMCM can be used in the receiver clock path to meet the input setup and hold requirements when implementing GMII/MII and RGMII. This is not used by default as the core cannot support Half-Duplex operation when this scheme is used. See the relevant physical interface chapters in this guide.

- IODELAY Usage

  IODELAYs are used in a receiver clock path to meet the input setup and hold requirements when implementing GMII/MII and RGMII. See the relevant physical interface chapters in this guide.

MMCM Usage

MMCM Phase Shifting Requirements

When using an MMCM, a fixed-phase shift offset is applied to the receiver clock MMCM to skew the clock; this performs static alignment by using the receiver clock MMCM to shift the internal version of the receiver clock such that the input data is sampled at the optimum time. The ability to shift the internal clock in small increments is critical for sampling high-speed source synchronous signals. For statically aligned systems, the MMCM output clock phase offset (as set by the phase shift value) is a critical part of the system, as is the requirement that the PCB is designed with precise delay and impedance-matching for all the GMII/MII or RGMII receiver data bus and control signals.

You must determine the best MCM setting (phase shift) to ensure that the target system has the maximum system margin to perform across voltage, temperature, and process (multiple chips) variations. Testing the system to determine the best MMCM phase shift setting has the added advantage of providing a benchmark of the system margin based on the UI (unit interval or bit time). System margin is defined as the following:

System Margin (ps) = UI(ps) * (working phase shift range/128)

Finding the Ideal Phase Shift Value

Xilinx cannot recommend a singular phase shift value that is effective across all hardware families. Xilinx does not recommend attempting to determine the phase shift setting analytically. In addition to the clock-to-data phase relationship, other factors such as package flight time (package skew) and clock routing delays (internal to the device) affect the clock to data relationship at the sample point (in the IOB) and are difficult to characterize.

Xilinx recommends extensive investigation of the phase shift setting during hardware integration and debugging.

Perform a complete sweep of phase-shift settings during your initial system test. Use a test range which covers at least half of the clock period or 128 taps. This does not imply that 128 phase-shift values must be tested; increments of 4 (52, 56, 60, and so forth) correspond to roughly one MMCM tap at 125 MHz, and consequently provide an appropriate step size. Additionally, it is not necessary to characterize areas outside the working phase-shift range.

At the edge of the operating phase shift range, system behavior changes dramatically. In eight phase shift settings or less, the system can transition from no errors to exhibiting errors. Checking the operational edge at a step size of two (on more than one board) refines the typical operational phase shift range. When the range is determined, choose the average of the high and low working phase shift values as the default. During the production test, Xilinx recommends that you re-examine the working range at corner case operating conditions to determine whether any final adjustments to the final phase shift setting are needed.

You can use the FPGA Editor to generate the required test file set instead of resorting to multiple PAR runs. Performing the test on design files that differ only in phase shift setting prevents other variables from affecting the test results. FPGA Editor operations can even be scripted further, reducing the effort needed to perform this characterization.

IODELAY Usage

IODELAY Tap Setting Requirements

With this method, an IODELAY is used on either the clock or Data (or both) to adjust the Clock/Data relationship such that the input data is sampled at the optimum time. The ability to adjust this relationship in small increments is critical for sampling high-speed source synchronous signals. For statically aligned systems, the IODELAY Tap setting is a critical part of the system, as is the requirement that the PCB is designed with precise delay and impedance-matching for all the GMII/MII or RGMII receiver data bus and control signals.

You must determine the best IODELAY Tap setting to ensure that the target system has the maximum system margin to perform across voltage, temperature, and process (multiple chips) variations.

Finding the Ideal Tap Setting Value

Xilinx cannot recommend a singular tap value that is effective across all packages or placements. Xilinx does not recommend attempting to determine the tap setting analytically. In addition to the clock-to-data phase relationship, other factors such as package flight time (package skew) and clock routing delays (internal to the device) affect the clock to data relationship at the sample point (in the IOB) and are difficult to characterize.

Xilinx recommends extensive investigation of the tap setting during hardware integration and debugging. The tap settings provided in the example design constraint file are placeholders, and work successfully in back-annotated simulation of the example design.

Perform a complete sweep of tap settings during your initial system test. If possible use a test range which covers at least half of the clock period. This does not imply that all values must be tested as it might be simpler to use a large step size initially to identify a tighter range for a subsequent run. Additionally, it is not necessary to characterize areas outside the working range. If an IODELAY is used on both Clock and Data then ensure this test range covers both clock only and data only adjustments.

At the edge of the operating range, system behavior changes dramatically. In four tap settings or less, the system can transition from no errors to exhibiting errors. Checking the operational edge at a step size of two (on more than one board) refines the typical operational range. When the range is determined, choose the average of the high and low working values as the default. During the production test, Xilinx recommends that you re-examine the working range at corner case operating conditions to determine whether any final adjustments to the final setting are needed. Where IODELAYs are used on the data it might be necessary or beneficial to use slightly different values for each bit.

You can use the FPGA Editor to generate the required test file set instead of resorting to multiple PAR runs. Performing the test on design files that differ only in tap setting prevents other variables from affecting the test results. FPGA Editor operations can even be scripted further, reducing the effort needed to perform this characterization.

XILINX®

Appendix B

Virtex-6 Embedded Tri-Mode Ethernet MAC versus Soft TEMAC

This appendix describes the differences between the Embedded Tri-Mode Ethernet MAC blocks, available in Virtex®-6 devices, and the soft IP cores, Tri-Mode Ethernet MAC (TEMAC), provided by the CORE Generator™ tool.

Note: This guide only considers versions of the Virtex-6 Embedded Tri-Mode Ethernet MAC from v2.1 and higher, and the soft Tri-Mode Ethernet MAC v5.1 and higher.

The functionality provided by the Embedded Tri-Mode Ethernet MACs can be provided by linking together the Tri-Mode Ethernet MAC soft IP core and the Ethernet 1000BASE-X PCS/PMA or SGMII core. More details are available at:

www.xilinx.com/products/design_resources/conn_central/protocols/gigabit_ethernet.htm

There are, however, some differences in the operation of the Embedded Tri-Mode Ethernet MACs themselves, which have evolved over three device generations, and between the embedded MACs and the soft IP cores. These differences are detailed in the following sections.

Virtex-6 Device

Features Exclusive to the Embedded Tri-Mode Ethernet MAC

These features are exclusive to the Embedded Tri-Mode Ethernet MAC:

- Includes integrated SGMII and 1000BASE-X PCS/PMA functionality.
- Includes an RGMII/SGMII status register.
- Supports 1000BASE-X PCS/PMA overclocking at 2 Gb/s and 2.5 Gb/s and provides a 16-bit client interface for these configurations.

Features Exclusive to Soft IP Cores

These features are exclusive to soft 10/100/1000 Mb/s, 1000 Mb/s and 10/100 Mb/s IP cores:

- The soft Tri-Mode Ethernet MAC solution:
  - Supports 1 GB half-duplex mode for parallel physical interfaces.
  - Supports IFG adjustment down to 8 bytes if half-duplex support is added or 4 bytes if full-duplex only.
- The soft PCS/PMA core:
  - Supports the ten bit interface (TBI).
  - Supports SGMII over LVDS in the Virtex-6.
  - Outputs a status vector with bits that indicate:
    - The status of the link.
    - The status of the link synchronization state machine.
    - When the core is receiving /C/ ordered sets.
    - When the core is receiving /I/ ordered sets.
    - When the core is receiving invalid data.
- Soft PCS/PMA and Tri-Mode Ethernet MAC solution IP cores:
  - Can be connected together to provide a single Ethernet interface, similar to the solution provided by the Embedded Tri-Mode Ethernet MAC.
  - Can be configured by a vector when the management interface is not required. The vector signals equate to attributes in the Embedded Tri-Mode Ethernet MAC.
  - The Embedded Tri-Mode Ethernet MAC is available in the Virtex-6 FPGA.

XILINX®

Appendix C

Debugging Designs

This appendix defines a step-by-step debugging procedure to assist in the identification and resolution of any issues that might arise during each phase of the design process. It contains the following sections:

- Debug Tools
- Simulation Debug
- Implementation and Timing Errors
- Hardware Debug

If this appendix does not help to resolve the issue, see Additional Core Resources and Technical Support in Chapter 1 for additional support.

Debug Tools

There are many tools available to debug Ethernet MAC design issues. It is important to know which tools are useful for debugging various situations. This section references the following tools:

Example Design

The Virtex®-6 Embedded Tri-Mode Ethernet MAC Wrapper v2.3 comes with a synthesizable example design complete with functional and post-place and route simulation test benches. Information on the example design can be found in Chapter 17, Detailed Example Design.

ChipScope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. The ChipScope Pro tool allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information on the ChipScope Pro tool, see www.xilinx.com/tools/cspro.htm.

Available Reference Boards

The ML605 is a Xilinx development board supporting 10/100/1000 Mb/s Ethernet. The ML605 board can be used to prototype designs and establish that the core can communicate with the system.

The provided example design can, if generated with the correct part and core options, be targeted directly to the ML605. For more information see Targeting the Example Design to a Board in Chapter 17.

Link Analyzers

Link analyzers can be used to generate and analyze traffic for hardware debug and testing. Common link analyzers include:

- Spirent SmartBits
- IXIA brand 10/100/1000 Ethernet test chassis
- Wireshark (a free packet sniffer software application)

Simulation Debug

The simulation debug flow for ModelSim is shown in Figure C-1.

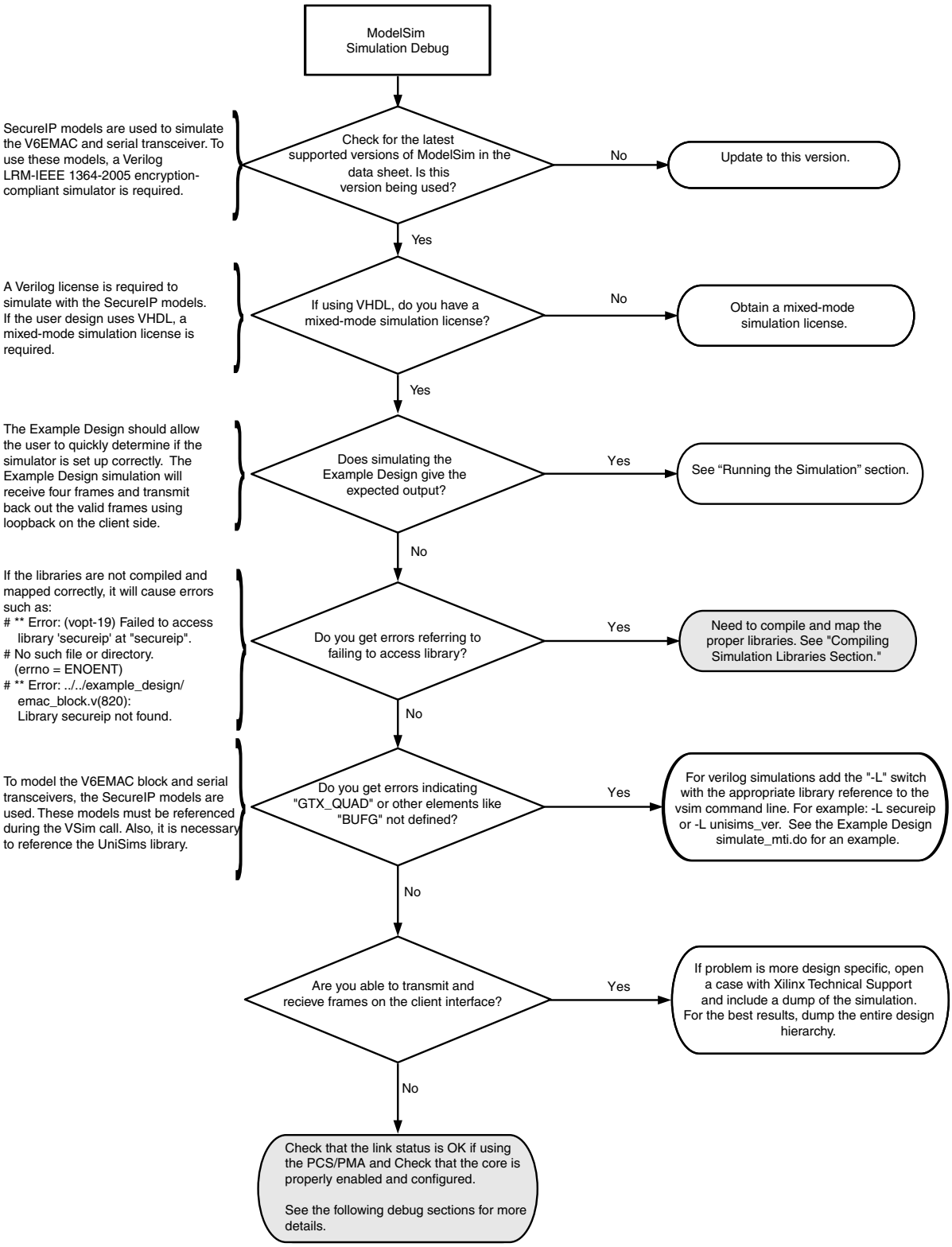

Figure C-1: **Simulation Debug Flow Chart**

Compiling Simulation Libraries

Compile the Xilinx simulation libraries, either by using the Xilinx Simulation Library Compilation Wizard, or by using the compxlib command line tool.

Xilinx Simulation Library Compilation Wizard

A GUI wizard provided as part of the Xilinx software can be launched to assist in compiling the simulation libraries by typing `compxlib` in the command prompt.

Compxlib

A compxlib command line can also be used to compile simulation libraries. This tool is delivered as part of the Xilinx software. For more information see the ISE® Software Manuals and specifically the *Command Line Tools Reference Guide* under the section titled compxlib.

Assuming the Xilinx and ModelSim environments are set up correctly, this is an example of compiling the SecureIP and UniSims libraries for Verilog into the current directory.

```
compxlib -s mti_se -arch virtex6 -l verilog -lib secureip -lib unisims
    -dir ./
```

There are many other options available for compxlib described in the *Command Line Tools Reference Guide*.

Compxlib produces a modelsim.ini file containing the library mappings. In ModelSim, to see the current library mappings, type `vmap` at the prompt. The mappings can be updated in the .ini file or to map a library at the ModelSim prompt type:

```
vmap [<logical_name>] [<path>]
```

For example:

```
vmap unisims_ver C:\my_unisim_lib
```

Implementation and Timing Errors

The example design provided with the Virtex-6 Embedded Tri-Mode Ethernet MAC Wrapper v2.3 comes complete with implementation scripts. For more details on using these scripts, see Implementation Scripts for Timing Simulation. If implementation or timing errors are encountered with the Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper, it is recommended to first try running the example design to see if the failures are seen there. If the failures do not exist in the example design, then differences between the example design and the design in which failures are seen can be compared.

Regional Clocking Errors in Map

When implementing the Ethernet MAC with either a GMII or RGMII physical interface, regional clocking methodologies are used. This means that there are the following requirements:

1. The receive-side physical interface clock (GMII_RX_CLK for GMII, or RGMII_RXC or RGMII) must be placed at a clock-capable I/O (CCIO) pin. If this requirement is not met, an error similar to the following one might be seen during implementation:

   ERROR:Place:839 - The component GMII_RX_CLK has been physically constrained to a location which is an invalid placement for this component.

2. All receive-side physical interface signals must be placed at package pins that correspond to the same clock region as the receive-side physical interface. If this requirement is not met, an error similar to the following might be seen during implementation:

   ERROR:Place:901 - IO Clock Net "gmii_rx_clk_bufio" cannot possibly be routed to component v6_emac_gmii_locallink_inst/v6_emac_gmii_block_inst/gmii/RXD_TO_MAC<2>" (placed in clock region "CLOCKREGION_X0Y1"), since it is too far away from source BUFIO "bufio_rx" (placed in clock region "CLOCKREGION_X1Y1"). The situation may be caused by user constraints, or the complexity of the design. Constraining the components related to the regional clock properly may guide the tool to find a solution.

3. An available Embedded Tri-Mode Ethernet MAC block must be present in a clock region that is reachable by the regionally buffered physical interface clock net. If this requirement is not met, an error similar to the following might be seen during implementation:

   ERROR:Place:905 - Components driven by Regional clock net <rx_clk_i> cannot be placed and routed because location constraints are causing the clock region rules to be violated. Regional Clock net <rx_clk_i> is being driven by BUFR <bufr_rx> locked to site "BUFR_X0Y10" Because of this location constraint, <rx_clk_i> can only drive clock regions "CLOCKREGION_X0Y5, CLOCKREGION_X0Y4".

   The following components driven by <rx_clk_i> have been locked to sites outside of these clock regions:
   v6_emac_gmii_locallink_inst/v6_emac_gmii_block_inst/v6_emac_gmii_inst/v6_emac (Locked Site: TEMAC_X0Y0 CLOCKREGION_X1Y2)

For more information on these requirements, see I/O Location Constraints.

Timing Failed for GMII/RGMII/MII OFFSET IN Constraint

To satisfy setup and hold requirements for these standards, fixed-mode IODELAYs are placed on the receive data and control signals when using the GMII, RGMII, or MII wrapper files. In the example design UCF, the fixed value delays are set based on the pinout used in the example design. With a different pinout, it might be required to adjust the fixed DELAY value to still meet the setup and hold requirements. For more details on how to adjust this delay to meet setup and hold requirements, see Chapter 14, Constraining the Core.

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The ChipScope tool is a valuable resource to use in hardware debug and the signal names mentioned in the following individual sections can be probed using the ChipScope tool for debugging the specific problems. Many of these common issues can also be applied to debugging design simulations. Details are provided on:

- General Checks
- Problems with Transmitting and Receiving Frames
- Link Bring-up Using 1000BASE-X or SGMII
- Problems with the MDIO
- Configuring the Ethernet MAC to the Correct Speed

General Checks

- Ensure that all the timing constraints for the core were properly incorporated from the example design delivered from the CORE Generator™ software and are met during place and route.
- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue.
- Ensure that all clock sources are active and clean. If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the LOCKED port.

Problems with Transmitting and Receiving Frames

Problems with data reception or transmission can be caused by a wide range of factors. The following list contains common causes to check for:

- Verify that the whole TEMAC block is not being held in reset. The whole block is held in reset if the main reset input is asserted or if `dcm_locked` is held low.
- Verify that both the receiver and transmitter are enabled and not being held in reset. For more information, see "Receiver and Transmitter Configuration Words" in MAC Configuration in Chapter 8.
- Verify that the Ethernet MAC is configured correctly and that the latest cores from the CORE Generator software or EDK are being used. Try running a simulation to check if the failure is hardware-specific.
- If using GMII or RGMII, check if setup and hold requirements are met using IDELAY components. For more information, see the section on debugging Implementation and Timing Errors.
- Verify that the link is up between the PHY and its link partner. If using 1000BASE-X or SGMII configurations of the Ethernet MAC, see the Link Bring-up Using 1000BASE-X or SGMII section for more details.
- If using an external PHY, is data received correctly if the PHY is put in loopback? If so, the issue might be on the link between the PHY and its link partner.
- Check if the address filter is enabled. If frames are not being received correctly, try disabling the address filter to ensure that the frame is not being dropped by the address filter. For more information, see Address/Frame Filter in Chapter 8.

- Verify that the Ethernet MAC has been configured to operate at the correct speed negotiated with the PHY. For more information, see the Configuring the Ethernet MAC to the Correct Speed section.
- Are received frames being dropped by user logic because `rx_axis_mac_tuser` is asserted? See Frame Reception with Errors in Chapter 6 for details on why frames are marked bad by the Ethernet MAC. The ChipScope tool can be inserted to get more details on the bad frames.
- Add the ChipScope tool to the design to look at the RX and TX AXI4-Stream and physical interface data signals, control signals and statistics vectors.

Link Bring-up Using 1000BASE-X or SGMII

Problems with Data Reception or Transmission

When no data is being received or transmitted:

- Ensure that a valid link has been established between the core and its link partner, either by auto-negotiation or manual configuration.
  - `syncacqstatus` should be high to indicate that the SYNC_ACQUIRED state from *IEEE Std 802.3-2008* [Ref 4], clause 36 state machine has been achieved.
  - If auto-negotiation is enabled, then PCS Status Register (Register 1) bit 5 should be read to verify that auto-negotiation has completed. The auto-negotiation interrupt output can also be used to verify that auto-negotiation has completed.

If no link has been established, see the topics discussed in the next section.

- Problems with Auto-Negotiation
- Problems in Obtaining a Link (Auto-Negotiation Disabled)

Note: Transmission through the core is not allowed unless a link has been established. This behavior can be overridden by setting the Unidirectional Enable attribute.

- Ensure that the Isolate state has been disabled.

By default, the Isolate state is set by the attribute PHY_ISOLATE. The Isolate state can be changed by writing to PCS Control Register (Register 0) or SGMII Control Register (Register 0) bit 10 after power-up. If the Isolate state is enabled, this results in no data transferred across the internal GMII interface between the PCS/PMA and MAC.

If data is being transmitted and received between the core and its link partner, but with a high rate of packet loss, see Problems with a High Bit Error Rate.

Problems with Auto-Negotiation

Determine whether auto-negotiation has completed successfully by doing one of the following.

- Poll the auto-negotiation completion bit 5 in Status Register (Register 1)
- Use the auto-negotiation interrupt port of the core (see Auto-Negotiation Interrupt)

If auto-negotiation is not completing:

1. Ensure that auto-negotiation is enabled in both the core and in the link partner (the device or test equipment connected to the core). Auto-negotiation cannot complete successfully unless both devices are configured to perform auto-negotiation. The auto-negotiation procedure requires that the auto-negotiation handshaking protocol between the core and its link partner, which lasts for several link timer periods, occurs

without a bit error. A detected bit error causes auto-negotiation to restart. Therefore, a link with an exceptionally high bit error rate might not be capable of completing auto-negotiation, or might lead to a long auto-negotiation period caused by the numerous restarts. If this appears to be the case, try the next step and see Problems with a High Bit Error Rate.

2. Try disabling auto-negotiation in both the core and the link partner and see if both devices report a valid link and are able to pass traffic. If they do, it proves that the core and link partner are otherwise configured correctly. If they do not pass traffic, see the next section, Problems in Obtaining a Link (Auto-Negotiation Disabled).

Problems in Obtaining a Link (Auto-Negotiation Disabled)

Determine whether the device has successfully obtained a link with its link partner by doing the following:

- Monitoring the state of `syncacqstatus`. If this is logic '1,' then synchronization, and therefore a link, has been established.
- Reading bit 2, Link Status, in Status Register (Register 1) when using the MDIO management interface.

If the devices have failed to form a link then do the following:

- Ensure that auto-negotiation is disabled in both the core and in the link partner (the device or test equipment connected to the core).
- Monitor the state of the `signal_detect` input to the core. This should either be:
  - connected to an optical module to detect the presence of light. Logic '1' indicates that the optical module is correctly detecting light; logic '0' indicates a fault. Therefore, ensure that this is driven with the correct polarity, or
  - tied to logic '1' (if not connected to an optical module).

Note: When `signal_detect` is set to logic '0,' this forces the receiver synchronization state machine of the core to remain in the loss of sync state.

- See the section, Problems with a High Bit Error Rate.

Serial Transceiver-Specific

- Ensure that the polarities of the TXN/TXP and RXN/RXP lines are not reversed. If they are, this can be fixed by using the TXPOLARITY and RXPOLARITY ports of the serial transceiver.
- Check that the serial transceiver is not being held in reset by monitoring the `mgt_tx_reset` and `mgt_rx_reset` signals between the core and the serial transceiver.
- Monitor the `rxbufstatus` signal when auto-negotiation is disabled. If this is being asserted, the elastic buffer in the receiver path of the serial transceiver is either underflowing or overflowing. This indicates a clock correction issue caused by differences between the transmitting and receiving ends. Check all clock management circuitry and clock frequencies applied to the core and to the serial transceiver.

Note: It is normal to see buffer errors during auto-negotiation because clock correction sequences are not sent during auto-negotiation. The PCS/PMA logic masks buffer errors during auto-negotiation and resets the RX buffer so that it recovers.

Problems with a High Bit Error Rate

Symptoms

The severity of a high-bit error rate can vary and cause any of the following symptoms:

- Failure to complete auto-negotiation when auto-negotiation is enabled.
- Failure to obtain a link when auto-negotiation is disabled in both the core and the link partner.
- High proportion of lost packets when passed between two connected devices that are capable of obtaining a link through auto-negotiation or otherwise. This can usually be accurately measured if the core is generated with the Ethernet Statistics counters.

Note: All bit errors detected by the PCS/PMA logic during frame reception show up as frame check sequence (FCS) errors in the Ethernet MAC statistics vector.

Debugging

- Compare the issue across several devices or PCBs to ensure that the issue is not a one-off case.
- Try using an alternative link partner or test equipment and then compare results.
- Try putting the core into loopback (both by placing the core into internal loopback, and by looping back the optical cable) and compare the behavior. The core should always be capable of auto-negotiating with itself and looping back its transmitter to receiver so direct comparisons can be made. If the core exhibits correct operation when placed into internal loopback, but not when loopback is performed using an optical cable, this might indicate a faulty optical module or a PCB issue.
- Try swapping the optical module on an erroneous device and repeat the tests.

Serial Transceiver-Specific Checks

Perform these additional checks when using a serial transceiver:

- Directly monitor the following ports of the serial transceiver by attaching error counters to them, or by triggering on them using the ChipScope tool or an external logic analyzer.

  RXDISPERR
  RXNOTINTABLE

  These signals should not be asserted over the duration of a few seconds, minutes or even hours. If they are frequently asserted, it might indicate an issue with the serial transceiver. See the *Virtex-6 GTX Transceivers User Guide* [Ref 11] for debugging serial transceiver issues.
- Place the serial transceiver into parallel or serial loopback.
  - If correct operation is seen in serial loopback, but not when loopback is performed using an optical cable, it might indicate a faulty optical module or issues on the PCB between the serial transceiver pins and the optical module.
  - If the core exhibits correct operation in serial transceiver parallel loopback but not in serial loopback, this might indicate a serial transceiver issue. See *Virtex-6 GTX Transceivers User Guide* [Ref 11] for more details.
- Minor bit error rates can be solved by adjusting the transmitter TXPREEMPHASIS, TXDIFFCTRL and TERMINATION_CTRL attributes of the serial transceiver.

Problems with the MDIO

See MDIO Interface in Chapter 8 for detailed information about performing MDIO transactions.

Things to check for:

- Ensure that the MDIO is driven properly and correctly terminated. Even if only using the internal MDIO interface, correct termination is needed to ensure the MDIO interface operates correctly.
- Check that the mdc clock is running and that the frequency is 2.5 MHz or less. If using the MDIO control registers to perform MDIO accesses, the MDIO interface does not work until the clock frequency is set with CLOCK_DIVIDE. The MDIO clock with a maximum frequency of 2.5 MHz is derived from the `s_axi_aclk` clock.
- Ensure that the TEMAC and PHY are not held in reset. Be sure to check the polarity of the reset to your external PHY. Many PHYs have an active-low reset.
- Read from a configuration register that does not have all 0s as a default. If all 0s are read back, the read was unsuccessful.
- If using the management interface to access the MDIO, check if the issue is just with the MDIO control registers or if there are also issues reading and writing MAC registers with the management interface.
- If accessing MDIO registers for the internal PCS/PMA, check that the PHYAD field placed into the MDIO frame matches the value placed on the `phyad`[4:0] port of the Ethernet MAC.
- If an external PHY is being used, check the PHY address. PHY address 0 is a global address for writing to all PHYs on the MDIO bus at the same time. If you have more than one PHY on the MDIO bus, you will have contention reading address 0. Unless the attribute EMAC_MDIO_IGNORE_PHYADZERO is enabled the internal PCS/PMA responds to address 0 if it is not held in reset. This is the case even if the TEMAC is not configured for a 1000BASE-X or SGMII interface.
- Has a simulation been run? Verify in simulation and/or a ChipScope tool capture that the waveform is correct for accessing the management interface for a MDIO read/write. The Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper example design test bench generated with 1000BASE-X or SGMII configurations performs MDIO writes to disable auto-negotiation.

Configuring the Ethernet MAC to the Correct Speed

When operating in tri-mode, the PHY negotiates the highest speed available with its link partner. The speed of the Ethernet MAC can be set by the user application after auto-negotiation completes by doing the following:

1. The user application can either monitor auto-negotiation interrupt from the external PHY or internal PCS/PMA, or poll for auto-negotiation, Status Register (Register 1) bit 5, to complete through MDIO.
2. When auto-negotiation completes the user application can read the MDIO auto-negotiation registers to obtain the negotiated speed.
3. The user application then needs to set this speed in the Ethernet MAC configuration registers using the host interface.

If auto-negotiation is disabled, the Ethernet MAC, PHY, and the PHY's link partner must all be set to the same speed.

XILINX®

Appendix D

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/company/terms.htm

References

1. Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC User Guide (UG368)
2. AXI Ethernet Data Sheet (DS759)
3. *AMBA AXI4-Stream Protocol Specification*
4. IEEE Std 802.3-2008
5. Reduced Gigabit Media Independent Interface (RGMII), version 2.0
6. Serial-GMII Specification (CISCO SYSTEMS, ENG-46158)
7. ARM AMBA AXI Protocol Specification v2.0
8. Virtex-6 FPGA Data Sheet: DC and Switching Characteristics (DS152)
9. Virtex-6 FPGA Configuration User Guide (UG360)
10. Virtex-6 FPGA SelectIO Resources User Guide (UG361)
11. Virtex-6 GTX Transceivers User Guide (UG366)
12. Xilinx Synthesis and Simulation Design Guide (UG626)
13. XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687)
14. Virtex-6 FPGA Embedded Tri-Mode Ethernet MAC Wrapper Data Sheet (DS835)
15. ML605 Hardware User Guide (UG534)

Additional Core Resources

For details and updates about the core, see the data sheet, available from the V6EMAC product page. From the document directory, available after generating the core, all product documentation, including the release notes, are available.

Related Xilinx Ethernet Products and Services

See the Ethernet Products and Services page at:

www.xilinx.com/products/design_resources/conn_central/protocols/gigabit_ethernet.htm