LogiCORE IP Tri-Mode Ethernet MAC v5.2

User Guide

UG777 January 18, 2012

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/01/11	1.1	Initial Xilinx Release
01/18/12	2.0	Added Ethernet AVB Endpoint information for ISE13.4 release

Table of Contents

XILINX®

Chapter 1

Introduction

The Tri-Mode Ethernet Media Access Controller (TEMAC) solution comprises the 10/100/1000 Mb/s, 1 Gb/s and 10/100 Mb/s IP (Intellectual Property) cores along with the optional Ethernet AVB Endpoint which are fully-verified designs that support Verilog-HDL (Hardware Description Language) and VHDL. In addition, the example design provided with the core is in both Verilog and VHDL. This chapter introduces the TEMAC solution and provides related information, including recommended design experience, additional resources, technical support, and submitting 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 v13.4

About the Core

The TEMAC solution 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 TEMAC product page and the Ethernet AVB Endpoint product page for that optional feature. For information about licensing the core, see Chapter 2, Licensing the Core.

Recommended Design Experience

Although the TEMAC core 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 TEMAC solution.

Xilinx provides technical support for use of this product as described in this manual, *Tri-Mode Ethernet MAC 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 TEMAC solution 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

Document

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

Licensing the Core

The Tri-Mode Ethernet MAC (TEMAC) solution consists of four Xilinx® LogiCORE™ IP cores. This chapter provides licensing instructions for the 10/100/1000 Mb/s Tri-Mode Ethernet MAC, 1 Gb/s Ethernet MAC, 10/100 Mb/s Ethernet MAC and the Ethernet AVB Endpoint.

You must obtain the appropriate licenses before using the cores in your designs. These IP cores are provided under the terms of the Xilinx LogiCORE IP Site License Agreement or Xilinx LogiCORE IP Project License Agreement. Purchase of the Tri-Mode Ethernet MAC core license includes licensing the 10/100/1000 Mb/s Tri-Mode, 1 Gb/s and the 10/100 Mb/s Ethernet MAC. Purchase of the 10/100 MAC core license only entitles full access to the 10/100 Mb/s IP. Purchase of the Ethernet AVB Endpoint license only entitles full access to the AVB Endpoint IP and the appropriate MAC license needs to be bought in addition. Purchase of a core entitles you to technical support and access to updates for a period of one year.

Before you Begin

This chapter assumes that you have installed all required software specified on the product page for this core.

License Options

The TEMAC solution provides three licensing options. After installing the required Xilinx ISE® software and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator™ tool. This key lets you assess core functionality with either the example design provided with the TEMAC solution, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place and route the design, evaluate timing, and perform back-annotated gate-level simulation of the TEMAC core and optional Ethernet AVB Endpoint core using the example design and the demonstration test bench provided with the core.

In addition, the license lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before *timing out* (ceasing to function) at which time it can be reactivated by reconfiguring the device.

Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including: Functional simulation support

- Back annotated gate-level simulation support
- Functional simulation support
- Full implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time outs

To obtain full access to the TEMAC when built with the optional AVB Endpoint, both the TEMAC and AVB Endpoint licenses need to be purchased.

Obtaining your License Key

This section contains information about obtaining a simulation, full system hardware, and full license keys.

Simulation License

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license, perform these steps:

1. Navigate to the TEMAC product page and optional Ethernet AVB Endpoint product page.
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE® software and IP Service Packs.

Obtaining a Full License

To obtain a Full license key, you must purchase licenses for the TEMAC and the optional Ethernet AVB Endpoint cores. After doing so, click the 'Access Core' link on the Xilinx.com IP core product page for further instructions.

Installing your License File

The Simulation Only Evaluation license key is provided with the ISE software CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

XILINX®

Chapter 3

Ethernet Overview

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

This chapter contains these sections:

- Typical Ethernet Application Overview
- Ethernet Protocol Overview

Typical Ethernet Application Overview

Typical applications for the Ethernet MAC include:

- Ethernet Switch or Router
- Ethernet Communications Port for an Embedded Processor
- Ethernet AVB Endpoint System

Ethernet Switch or Router

Figure 3-1 illustrates a typical application for a single Ethernet MAC. The Physical-side interface (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, by additionally using the Ethernet 1000BASE-X PCS/PMA or SGMII LogiCORE™, SGMII.

The user side of the Ethernet MAC 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 TEMAC solution 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.

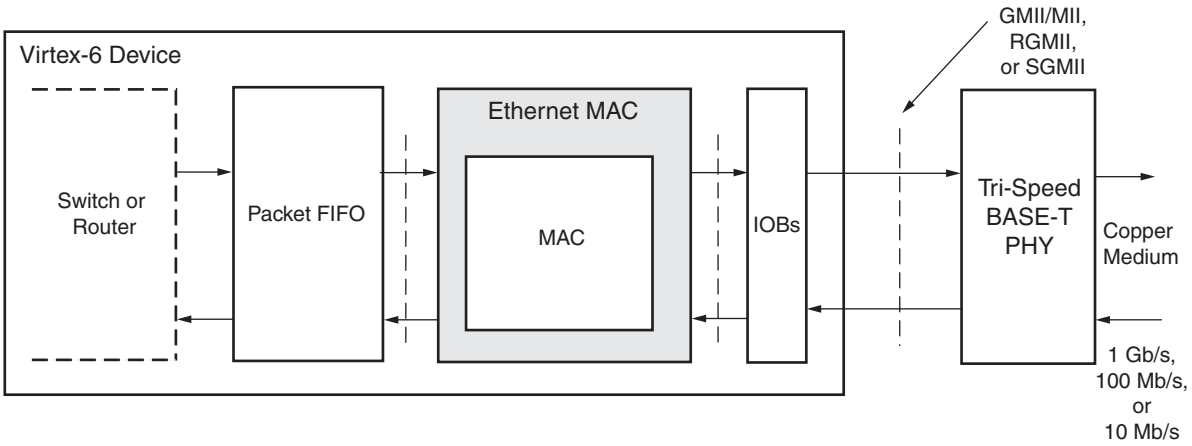

Figure 3-1: **Typical Application: Ethernet Switch or Router**

Ethernet Communications Port for an Embedded Processor

Figure 3-2 illustrates a typical application for a single Ethernet MAC. 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, by additionally using the Ethernet 1000BASE-X PCS/PMA or SGMII LogiCORE, 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 Transmission Control Protocol/Internet Protocol (TCP/IP). 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 user interface of the MAC to the DMA port of a processor. [Ref 6] describes the AXI ETHERNET, which can be instantiated for an intended processor application.

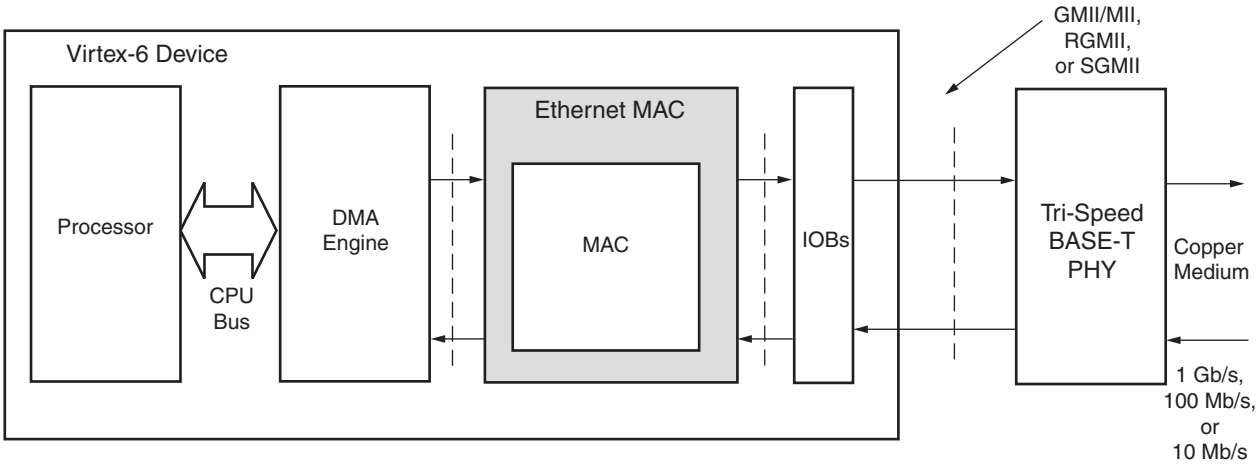

Figure 3-2: **Typical Application: Ethernet Communications Port for Embedded Processor**

Ethernet AVB Endpoint System

Figure 3-3 illustrates a typical implementation for the TEMAC(100/1000 Mb/s) core when the optional Ethernet AVB endpoint is included. Endpoint refers to a talker (for example,

DVD player) or listener (for example, TV set) device as opposed to an intermediate bridge function, which is not supported. In the implementation, the Tri-Mode Ethernet MAC core, with the AVB front end, is connected to an AVB-capable network.

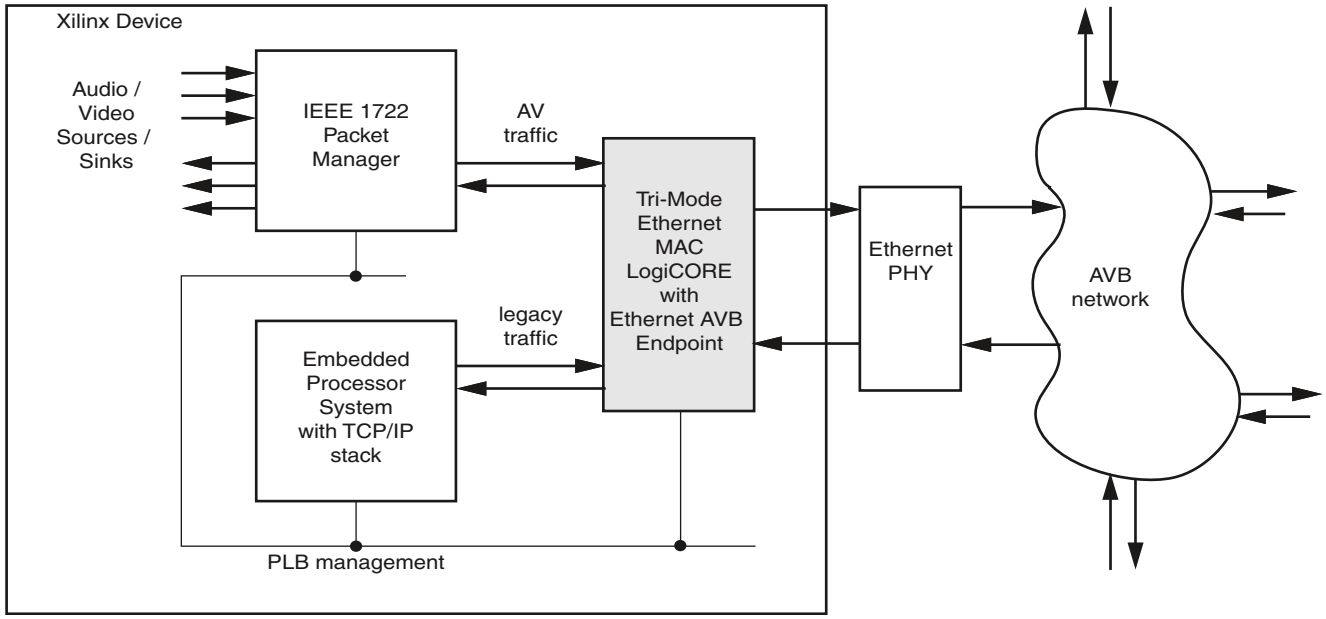

Figure 3-3: **Ethernet AVB Endpoint system**

Figure 3-3 illustrates that the Tri-Mode Ethernet MAC core with the Ethernet AVB Endpoint logic supports two main data interfaces at the user side:

1. The **AV traffic** interface is intended for the Quality of Service audio/video data. Illustrated are a number of audio/video sources (for example, a DVD player), and a number of audio/video sinks (for example, a TV set). The Ethernet AVB Endpoint gives priority to the **AV traffic** interface over the **legacy traffic** interface, as dictated by *IEEE 802.1Q* 75% bandwidth restrictions.
2. The **legacy traffic** interface is maintained for *best effort* Ethernet data: Ethernet as we know it today (for example, a PC surfing the internet). Wherever possible, priority is given to the **AV traffic** interface (as dictated by *IEEE 802.1Q* bandwidth restrictions), but a minimum of 25% of the total Ethernet bandwidth is always available for legacy Ethernet applications.

The **AV traffic** interface in Figure 3-3 is shown as interfacing to a 1722 Packet Manager block. The *IEEE1722* is also another standard which specifies the embedding of audio/video data streams into Ethernet Packets. The 1722 headers within these packets include presentation timestamp information. Contact Xilinx for an engineering solution and for more system-level information.

Ethernet Protocol Overview

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

Ethernet Sublayer Architecture

Figure 3-4 illustrates the relationship between the Open Systems Interconnection (OSI) reference model and the Ethernet MAC, as defined in [Ref 10]. The grayed-in layers show the functionality that the Ethernet MAC handles. Figure 3-4 also shows where the supported physical interfaces fit into the architecture.

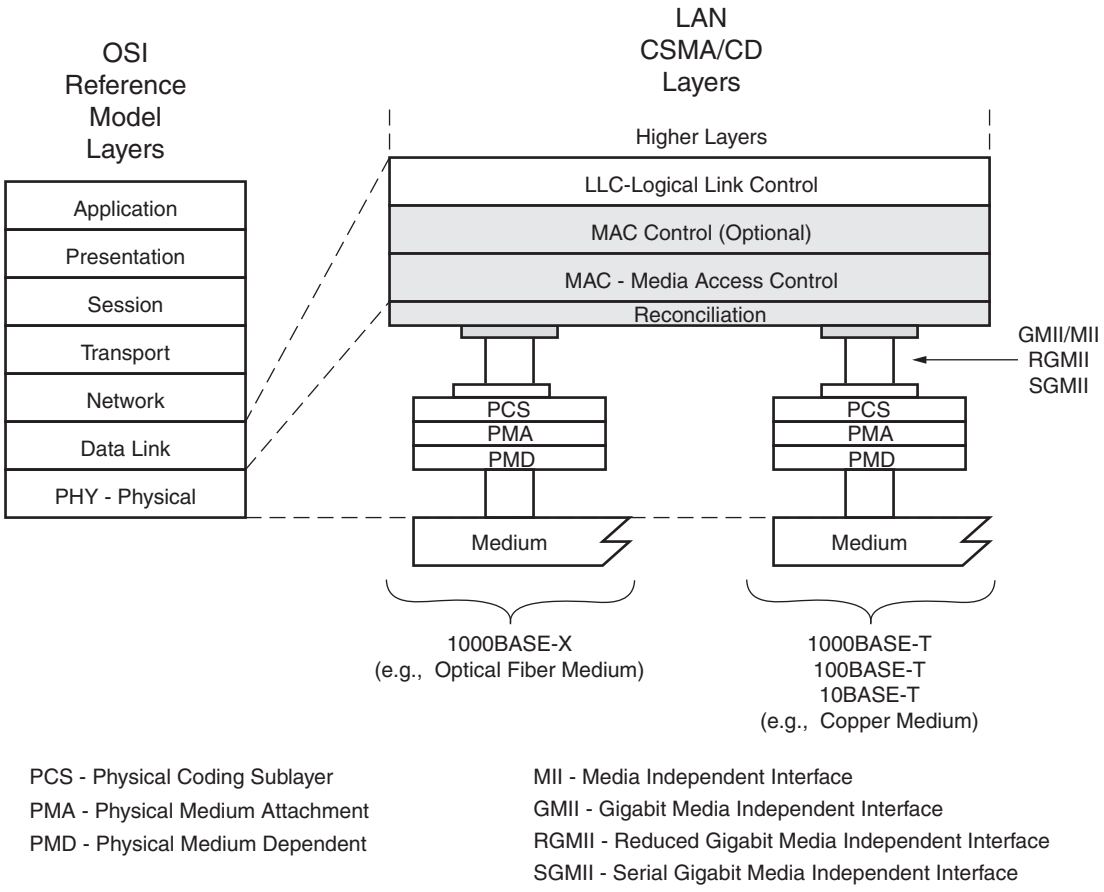

Figure 3-4: **IEEE Std 802.3-2008 Ethernet Model**

MAC and MAC CONTROL Sublayer

The Ethernet MAC is defined in IEEE Std 802.3-2008, 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, clause 31. This provides real-time flow control manipulation of the MAC sublayer.

Both the MAC CONTROL and MAC sublayers are provided by the Ethernet MAC 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 TEMAC. However, external BASE-T PHY devices are readily available on the market. These can connect to the Ethernet MAC, using GMII/MII, RGMII, or, by additionally using the Ethernet 1000BASE-X PCS/PMA or SGMII LogiCORE, SGMII interfaces.
- **BASE-X** PHYs provide a link between the MAC and (usually) fiber optic mediums. The TEMAC is capable of supporting the 1 Gb/s BASE-X standard; 1000BASE-X PCS and PMA sublayers can be offered by connecting the TEMAC to the Ethernet 1000BASE-X PCS/PMA or SGMII LogiCORE.

Ethernet Data Format

Ethernet data is encapsulated in frames, as shown in Figure 3-5, 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 Ethernet MAC can handle jumbo Ethernet frames where the data field can be much larger than 1500 bytes.

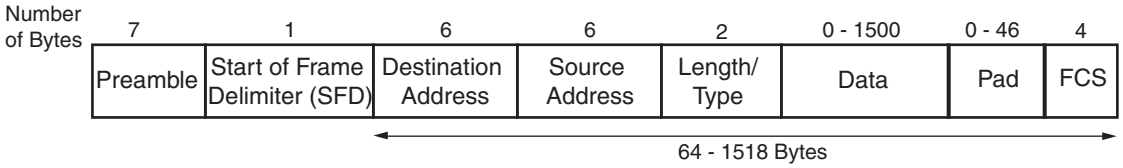

Figure 3-5: **Standard Ethernet Frame Format**

The Ethernet MAC can also accept Virtual LAN (VLAN) frames. The VLAN frame format is shown in Figure 3-6. If the frame is a VLAN type frame, the Ethernet MAC accepts four additional bytes.

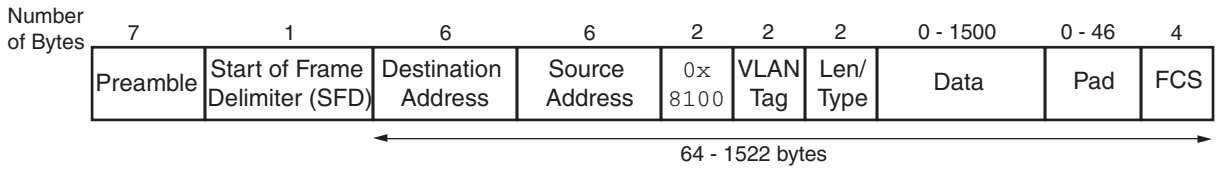

Figure 3-6: **Ethernet VLAN Frame Format**

Ethernet PAUSE/flow control frames can be transmitted and received by the Ethernet MAC. Figure 8-2, page 56 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 Ethernet MAC.

Preamble

For transmission, this field is automatically inserted by the Ethernet MAC. 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 Ethernet MAC 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 Ethernet MAC. 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 `1`s) is a multicast address that addresses all stations on the Local Area Network (LAN). The Ethernet MAC 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 TEMAC.

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. A value of 1536 decimal or greater is interpreted by the Ethernet MAC 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 Ethernet MAC does not perform any processing of the length/type field.

For reception, if this field is a length field, the Ethernet MAC 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 Ethernet MAC 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 Ethernet MAC 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 Ethernet MAC 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 Ethernet MAC or can be supplied by the user. If the pad field is inserted by the Ethernet MAC, the FCS field is calculated and inserted by the Ethernet MAC. If the pad field is supplied by the user, the FCS can be either inserted by the Ethernet MAC 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 Ethernet MAC 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 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 Ethernet MAC or supplied by the user, as indicated by a configuration register bit.

For reception, the incoming FCS value is verified on every frame. If an incorrect FCS value is received, the Ethernet MAC indicates to the user that it has received a bad frame. The FCS field can either be passed on to the user or be dropped by the Ethernet MAC, 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 the IEEE Std 802.3-2008, to be 96 bit times (9.6 µs for 10 Mb/s, 0.96 µs for 100 Mb/s, and

96 ns for 1 Gb/s). This value 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 Ethernet MAC 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 Ethernet MAC 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 Ethernet MAC starts the timing of the interframe gap.
3. The Ethernet MAC 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 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 Ethernet MAC 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 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 for 10/100 Mb/s) 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 Ethernet MAC does not use the carrier sense signal from the external PHY because the medium is not shared, and the Ethernet MAC only needs to monitor its own transmissions. After the last bit of an Ethernet MAC frame transmission, the Ethernet MAC starts the interframe gap timer and defers transmissions until the IFG count completes. The minimum value supported for the IFG depends on the TEMAC solution options and the current mode of operation. If the TEMAC solution has been built with half-duplex support then 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`. If the TEMAC solution has been built with only full-duplex support then the IFG delay is 96 bit times, or when IFG Adjustment is enabled, the greater of 32 bit times, and the value presented on `tx_ifg_delay`.

Chapter 4

Generating the Core

The TEMAC solution which comprises the 10/100/1000 Mb/s, 1 Gb/s and 10/100 Mb/s IP cores are 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

Figure 4-1 displays the TEMAC core customization screen.

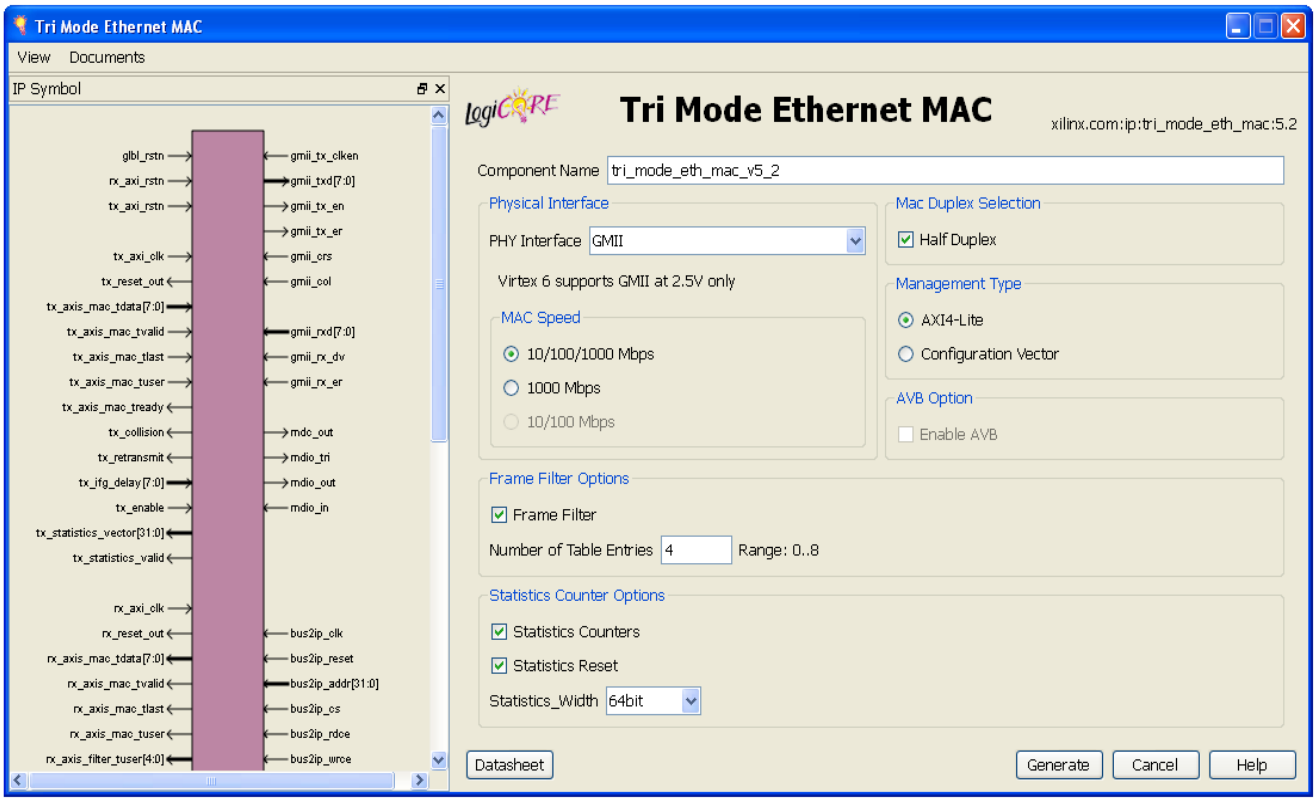

Figure 4-1: **Core Customization Screen**

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

Four 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.
- Internal. The core is generated with no physical interface ready for connection to an internal PHY such as the Ethernet 1000BASE-X PCS/PMA or SGMII LogiCORE.

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. There is no change in the core netlist between RGMII, GMII or Internal. If MII is selected then the physical interface datapath is reduced to 4 bits. The default is to use GMII.

MAC Speed

The TEMAC solution 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).

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 Internal is selected then only Tri-speed is available as the speed is under the control of the internal PHY.

Half Duplex

The TEMAC solution always provides support for full-duplex Ethernet. However, to provide half-duplex operation, further FPGA logic resources are required. Because many applications require only full-duplex support, the half-duplex logic is therefore optional.

When the core is generated with half-duplex logic, full or half-duplex operation can be selected using TEMAC configuration.

The default is to include half-duplex support.

When half-duplex is selected the AVB option is disabled.

Note: If a MMCM is used on the physical interface receive path to control the clock to data relationship then 1 G half-duplex is not supported.

Management Interface

Select the AXI4-Lite option if you wish to include the optional Management Interface for TEMAC configuration (see The Management Interface, page 77). If this option is not selected, the core is generated with a replacement configuration vector. If the AXI4-Lite Management Interface is not selected the AVB option is not available. The default is to have the AXI4-Lite Management Interface.

AVB Option

Select the Enable_AVB option if you wish the optional AVB Endpoint front end logic to be included.

- If half-duplex is selected the AVB option is disabled
- If the AXI4-Lite management interface is not selected the AVB option is disabled

If selected the fee-based Ethernet AVB Endpoint license is required in addition to the Tri-Mode Ethernet MAC license to enable core generation. The default is to not have the AVB Endpoint included.

Frame Filter

It is possible to generate the core with a frame 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 Frame Filter.

Number of Table Entries

The Frame Filter can be generated with a look-up table that holds up to eight 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 4 table entries.

Statistics Counters

It is possible to generate the core with built in statistics counters. The number of counters available is dependant upon the duplex setting of the core with full-duplex requiring 34 counters and half duplex requiring 44 counters. This option can only be selected when the core is configured with the AXI4-Lite Management Interface. The default is to include the Statistics counters

Statistics Reset

When the Statistics Counters are included it is possible to include logic to ensure the counters are cleared to zero upon a hardware reset. Without this logic the counter values persist over a reset and are only cleared upon device configuration. The default is to include the counter reset functionality

Statistics Width

The Statistics counters can be either 32 bits or 64 bits wide. This allows the user to control the frequency at which the counters must be polled to avoid information loss due to overflow. The default is to use 64-bit wide counters.

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 4-1 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=tri_mode_eth_mac_v5_2
CSET physical_interface=GMII
CSET mac_speed=Tri_speed
CSET enable_avb=false
CSET half_duplex=true
CSET management_interface=true
CSET frame_filter=true
CSET number_of_table_entries=4
CSET statistics_counters=true
CSET statistics_reset=true
CSET statistics_width=32bit
```

Table 4-1: **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 _	tri_mode_eth_mac_v5_2
physical_interface	One of the following keywords: mii, gmii, rgmii, internal	gmii
mac_speed	One of the following keywords: tri_speed, 1000_Mbps, 10_100_Mbps	tri_speed
enable_avb	One of the following keywords: true, false	false
half_duplex	One of the following keywords: true, false	true
management_interface	One of the following keywords: true, false	true
frame_filter	One of the following keywords: true, false	true
number_of_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	64bit

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 17, Quick Start Example Design, for details about the CORE Generator software output files and Chapter 18, Detailed Example Design, for details on the HDL example design.

XILINX®

Chapter 5

Tri-Mode Ethernet MAC Overview

This chapter introduces the TEMAC solution architecture including all interfaces and the major functional blocks.

This chapter contains these sections:

Key Features

The key features of the TEMAC solution are:

- Designed to the IEEE Std 802.3-2008 specification
- Supports four separate IP cores
  - 10/100/1000 Mb/s Ethernet MAC
  - 1 Gb/s Ethernet MAC
  - 10/100 Mb/s Ethernet MAC
  - Optional Ethernet AVB
- Configurable duplex operation
- Support for Media Independent Interface (MII), Gigabit Media Independent Interface (GMII), Reduced Gigabit Media Independent Interface (RGMII) and connection to the Ethernet 1000BASE-X PCS/PMA or SGMII LogiCORE™.
- Management Data Input/Output (MDIO) interface to manage objects in the physical layer
- User-accessible raw statistic vector outputs
- Optional built in statistics counters
- Optional built-in Ethernet AVB Endpoint designed to the following IEEE specifications
  - IEEE802.1AS

    Supports clock master functionality, clock slave functionality and the Best Master Clock Algorithm (BMCA)
  - IEEE802.1Qav

    Supports arbitration between different priority traffic and implements bandwidth policing
- 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 maximum frame length check
- Configurable receive frame filter
- AXI4-Stream user interface

Note: Virtex-6 devices support GMII and MII at 2.5 V only; Spartan®-6 devices support GMII and MII at 3.3 V or lower. For 7 Series families it depends on the type of IO used: HR I/O supports GMII or RGMII at 2.5 V whereas HP I/O only supports 1.8 V or lower. Despite this being the defined RGMII voltage most PHYs require 2.5V and therefore an external voltage converter is required to interoperate with any multistandard PHY when using 7 series HP I/O.

Architecture Description

Figure 5-1 illustrates a block diagram of the TEMAC and optional Ethernet AVB Endpoint cores.

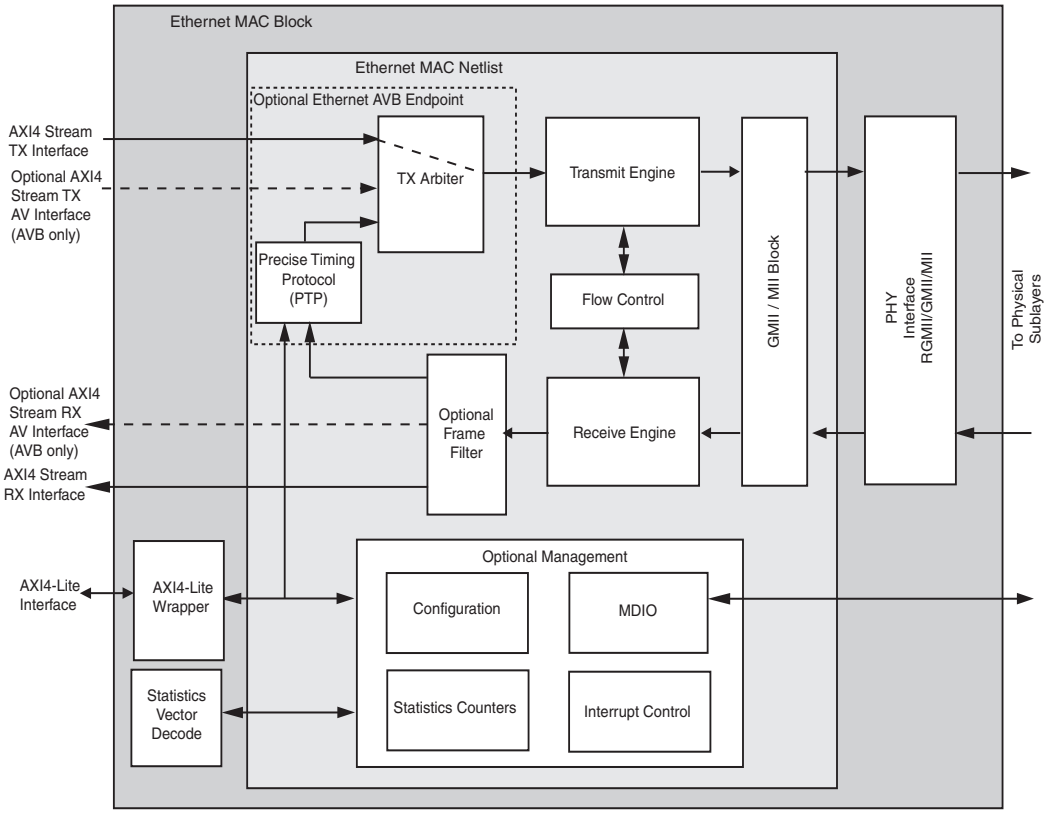

Figure 5-1: **Tri-Mode Ethernet MAC Block Diagram**

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 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 Ethernet MAC 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 either RGMII or GMII/MII.

When connecting to external PHY devices, either of GMII/MII or RGMII interfaces can be used: these are selectable from the CORE Generator™ tool GUI (see Chapter 4, Generating the Core). These external interfaces are described in these chapters:

- Chapter 11, Physical Interface for the 10 Mb/s and 100 Mb/s Only Ethernet MAC IP Core
- Chapter 12, Physical Interfaces for 1 Gb/s Only Ethernet MAC IP Core
- Chapter 13, Physical Interfaces for Tri-speed (10 Mb/s, 100 Mb/s and 1 Gb/s) Ethernet MAC IP Core

Alternatively, the core can be generated without the PHY interface to allow internal connection to the Ethernet 1000BASE-X PCS/PMA or SGMII core to provide Serial Gigabit Media Independent Interface (SGMII) or to perform the 1000BASE-X physical standard. See Chapter 15, Interfacing to Other Xilinx Ethernet Cores.

Ethernet AVB Endpoint

The TEMAC can be implemented with an optional Ethernet AVB endpoint which itself is made up of two key functional blocks, see below. When this functionality is not included the AXI4-Stream TX Data is passed directly to the transmit engine. The AXI4-Stream RX Data is always passed directly to the user, with the relative tuser signals being used to validate the data on the required interface. See Chapter 9, Ethernet AVB Endpoint.

Precise Timing Protocol (PTP)

The Precise Timing Protocol (PTP) block within the core provides the dedicated hardware to implement the *IEEE 802.1AS* specification. However, full functionality is only achieved using a combination of this hardware block coupled with functions provided by the

relevant software drivers (run on an embedded processor). For more information please refer to Precise Timing Protocol Packet Buffers in Chapter 9

TX Arbiter

Data for transmission over an AVB network can be obtained from three source types:

1. **AV Traffic.** For transmission from the AV Traffic interface of the core.
2. **Precise Timing Protocol (PTP) Packets**. Initiated by the software drivers using the dedicated hardware
3. **Legacy Traffic**. For transmission from the Legacy Traffic interface of the core.

The transmitter (TX) arbiter selects from these three sources in the following manner. If there is AV packet available and the programmed AV bandwidth limitation is not exceeded then the AV packet is transmitted, otherwise the TX arbiter checks to see if there are any PTP packets to be transmitted, otherwise if there is an available legacy packet then this is transmitted. To comply with the specifications, the AV Traffic Interface should not be configured to exceed 75%. The arbiter then polices this bandwidth restriction for the AV traffic and ensures that on average, it is never exceeded. Consequently, despite the AV traffic having a higher priority than the legacy traffic, there is always remaining bandwidth available to schedule legacy traffic. For more information see Chapter 9, Ethernet AVB Endpoint.

Transmit Engine

The transmit engine takes data from the AXI4-Stream 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.

The transmitter user interface of the core is defined in Chapter 7, AXI4-Stream User Interface.

Receive Engine

The receive engine takes the data from the GMII/MII interface and checks it for compliance to the *IEEE 802.3*. Padding fields are removed and the AXI4-Stream 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.

The receiver user interface of the core is defined in Chapter 7, AXI4-Stream User Interface.

Flow Control

The flow control block is designed to *IEEE 802.3-2008* 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 8, Flow Control.

GMII/MII Block

The GMII/MII interface, which only operates at speeds below 1 Gb/s, converts between the 4-bit data required by MII and the 8-bit data expected by the Receiver/Transmitter interfaces.

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. 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 can be configured using configuration vectors. See Chapter 10, Configuration and Status.

MDIO Interface

The optional MDIO interface can be written to and read from using the Management Interface. The MDIO is used to monitor and configure PHY devices. The MDIO interface is defined in *IEEE 802.3* clause 22.

Frame Filter

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

When the AVB Endpoint is included the Frame Filter is always present with three filters being dedicated to identifying AV or PTP data. In this case these filters are initialized to identify the default values for the various AVB specific frame fields. The number of filters selected by the user is in addition to these three.

Statistics Counters

The TEMAC solution can be implemented with optional Statistics Counters.

Core Interfaces

All ports of the netlist are internal connections in the Field Programmable Gate Array (FPGA) logic. An example HDL design, provided 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 (Input/Output Block) 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).

User Interfaces

Transmitter Interface

Table 5-1 defines the AXI4-Stream transmit signals of the core, which are used to transmit data from the user to the core. Table 5-2 defines transmit sideband signals. See Transmitting Outbound Frames, page 48.

Table 5-1: **Transmit Interface AXI4-Stream Signal Pins**

Signal	Direction	Description
tx_axis_mac_tdata[7:0]	Input	Frame data to be transmitted.
tx_axis_mac_tvalid	Input	Control signal for tx_axis_mac_tdata port. Indicates the data is valid.

Table 5-1: **Transmit Interface AXI4-Stream Signal Pins *(Cont'd)***

Signal	Direction	Description
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 10/100 Mb/s this is used to meter the data into the core at the correct rate.

Note: All signals are active High.

Table 5-2: **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, page 52 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 user 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, page 53 for vector contents.
tx_statistics_valid	Output	Asserted at end of frame transmission, indicating that the tx_statistics_vector is valid.

Note: All signals are active High.

Table 5-3 defines the optional AXI4-Stream AV transmit signals included when the AVB functionality is selected.

Table 5-3: **Transmit Interface AXI4-Stream AV Signal Pins**

Signal	Direction	Description
tx_axis_av_tdata[7:0]	Input	Frame data to be transmitted.
tx_axis_av_tvalid	Input	Control signal for tx_axis_av_tdata port. Indicates the data is valid.
tx_axis_av_tlast	Input	Control signal for tx_axis_av_tdata port. Indicates the final transfer in a frame.

Table 5-3: **Transmit Interface AXI4-Stream AV Signal Pins *(Cont'd)***

Signal	Direction	Description
tx_axis_av_tuser	Input	Control signal for tx_axis_av_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_av_tready	Output	Handshaking signal. Asserted when the current data on tx_axis_av_tdata has been accepted and tx_axis_av_tvalid is high. At 100 Mb/s this is used to meter the data into the core at the correct rate.

Note: All signals are active High

Receiver Interface

Table 5-4 describes the receive AXI4-Stream signals used by the core to transfer data to the user. Table 5-5 describes the related sideband interface signals.

Table 5-4: **Receive Interface AXI4-Stream Signal Pins**

Signal	Direction	Description
rx_axis_mac_tdata[7:0]	Output	Frame data received is supplied on this port.
rx_axis_mac_tvalid	Output	Control signal for the rx_axis_mac_tdata port. Indicates the data is valid.
rx_axis_mac_tlast	Output	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, page 43. and Frame Reception with Errors, page 44.
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.

Note: All signals are active High.

Table 5-5: **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, page 47 for the vector contents.
rx_statistics_valid	Output	Asserted at end of frame reception, indicating that the rx_statistics_vector is valid.

Note: All signals are active High.

Table 5-6 defines the optional AXI4-Stream AV receive signals included when the AVB functionality is selected.

Table 5-6: **Receive Interface AXI4-Stream AV Signal Pins**

Signal	Direction	Description
rx_axis_av_tdata[7:0]	Output	Frame data received is supplied on this port.
rx_axis_av_tvalid	Output	Control signal for the rx_axis_av_tdata port. Indicates the data is valid.
rx_axis_av_tlast	Output	Control signal for the rx_axis_av_tdata port. Indicates the final byte in the frame.
rx_axis_av_tuser	Output	Control signal for rx_axis_av_tdata. Asserted at end of frame reception to indicate that the frame had an error.

Note: All signals are active High.

Flow Control Interface

Table 5-7 describes the signals used by the user to request a flow control action from the transmit engine. See Chapter 8, Flow Control.

Table 5-7: **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, page 57.
pause_val[15:0]	Input	Pause value: inserted into the parameter field of the transmitted pause frame.

Note: All signals are active High.

AXI4-Lite Signal Definition

Table 5-8 describes the optional signals used by the user to access the MAC netlist, including configuration, status and MDIO access. See The Management Interface, page 77.

Table 5-8: **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

Table 5-8: **Optional AXI4-Lite Signal Pinout *(Cont'd)***

Signal	Direction	Description
s_axi_bvalid	Output	Write Response valid
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

Configuration Vector Signal Definition

Table 5-9 describes the configuration vectors, which use direct inputs to the core to replace the functionality of the MAC configuration bits when the Management Interface is not used.

Table 5-9: **Alternative to the Optional Management Interface: Configuration Vector Signal Pinout**

Signal	Direction	Description
rx_mac_config_vector[79:0]	Input	The RX Configuration Vector is used to replace the functionality of the MAC RX Configuration Registers when the Management Interface is not used.
tx_mac_config_vector[79:0]	Input	The TX Configuration Vector is used to replace the functionality of the MAC TX Configuration Registers when the Management interface is not used.

Note: All bits of the config vectors are registered on input but can be treated as asynchronous inputs.

Clock, Speed Indication, and Reset Signal Definition

Table 5-10 describes the reset signals, the clock signals that are input to the core, and the outputs that can be used to select between the three operating speeds. The clock signals are generated in the top-level wrapper provided with the core.

Table 5-10: **Clock and Speed Indication 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

Table 5-10: **Clock and Speed Indication Signals *(Cont'd)***

Signal	Direction	Description
tx_reset	Output	Active High TX software reset from MAC netlist
gtx_clk	Input	Global 125 MHz clock
rtc_clk	Input	Only available when the core is generated with AVB. Reference clock used to increment the Real Time Clock (RTC). The minimum frequency is 25 MHz. Xilinx recommends a 125 MHz clock source shared with gtx_clk.
tx_mac_aclk	Input	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 GMII or MII. See the appropriate section: • Chapter 11, Physical Interface for the 10 Mb/s and 100 Mb/s Only Ethernet MAC IP Core • Chapter 12, Physical Interfaces for 1 Gb/s Only Ethernet MAC IP Core • Chapter 13, Physical Interfaces for Tri-speed (10 Mb/s, 100 Mb/s and 1 Gb/s) Ethernet MAC IP Core
rx_mac_aclk	Input	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 11, Physical Interface for the 10 Mb/s and 100 Mb/s Only Ethernet MAC IP Core • Chapter 12, Physical Interfaces for 1 Gb/s Only Ethernet MAC IP Core • Chapter 13, Physical Interfaces for Tri-speed (10 Mb/s, 100 Mb/s and 1 Gb/s) Ethernet MAC IP Core
speedis100	Output	This output is asserted when the core is operating at 100 Mb/s. It is derived from either bits 30 and 31 of the MAC Speed Configuration register.If the Management Interface is not present, this is derived from configuration vector bits 65 and 66.
speedis10100	Output	This output is asserted when the core is operating at either 10 Mb/s or 100 Mb/s. It is derived from either bits 30 and 31 of the MAC Speed Configuration register. If the Management Interface is not present, this is derived from configuration vector bits 65 and 66.

Interrupt Signals

Table 5-11 describes the interrupt signals provided by the TEMAC core.

Table 5-11: **Interrupt Signals**

Signal	Direction	Description
mac_int	Output	This is the interrupt output from the interrupt controller. Currently the only interrupt source which can be configured is the mdio_ready signal. See Interrupt Controller in Chapter 10 for more information.
interrupt_ptp_rx	Output	Only available when the core is generated with AVB. This is asserted following the reception of any PTP packet by the RX PTP Packet Buffers. See RX PTP Packet Buffer in Chapter 9 for more information.
interrupt_ptp_tx	Output	Only available when the core is generated with AVB. This is asserted following the transmission of any PTP packet from the TX PTP Packet Buffers. See TX PTP Packet Buffer in Chapter 9 for more information.
interrupt_ptp_timer	Output	Only available when the core is generated with AVB. This interrupt asserts every 1/128 s as measured by the RTC. This acts as a timer for the PTP software algorithms. See Real Time Clock in Chapter 9 for more information.

Ethernet AVB Endpoint PTP signals

Table 5-12 defines the signals output from the core by the Precise Timing Protocol (PTP). These signals are only present when the AVB Endpoint is included in the TEMAC, are provided for reference only and can be used by an application.

Table 5-12: **AVB Specific Signals**

Signal	Direction	Description
rtc_nanosec_field	Output	This is the synchronised nanoseconds field from the RTC.
rtc_sec_field	Output	This is the synchronised seconds fields from the RTC.
clk8k	Output	This is an 8 kHz clock which is derived from, and synchronized in frequency, to the Real Time Clock. The period of this clock, 125 µs, can be useful in timing SR class measurement intervals.
rtc_nanosec_field_1722	Output	The IEEE1722 specification contains a different format for the Real Time Clock, provided here as an extra port. This is derived and is in sync with the IEEE802.1 AS Real Time Clock.

Physical Interface Signals

MDIO Signal Definition

Table 5-13 describes the MDIO (MII Management) interface signals of the core. The MDIO format is defined in [Ref 10], 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 5-13: **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.
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 5-14 through Table 5-16 describe the three possible interface standards supported, RGMII, GMII and MII, which are typically attached to a PHY module, either off-chip or internally integrated. The RGMII is defined in [Ref 14], the GMII is defined in [Ref 10], clause 35, and MII is defined in [Ref 10], clause 22.

Table 5-14: **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 5-15: **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 5-16: **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

XILINX®

Chapter 6

Designing with the Core

This chapter provides guidelines for creating designs using the TEMAC solution. For details about the example design included with the TEMAC solution, see Chapter 17, Quick Start Example Design and Chapter 18, Detailed Example Design.

General Design Guidelines

This section describes the steps required to turn the TEMAC solution 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 4, 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 6-1 is a block diagram of the example design. For details about the example design, see Chapter 17, 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 Physical Interface chapters in this document)
- Derive the clock logic, required for the core (See the Physical Interface chapters in this document).

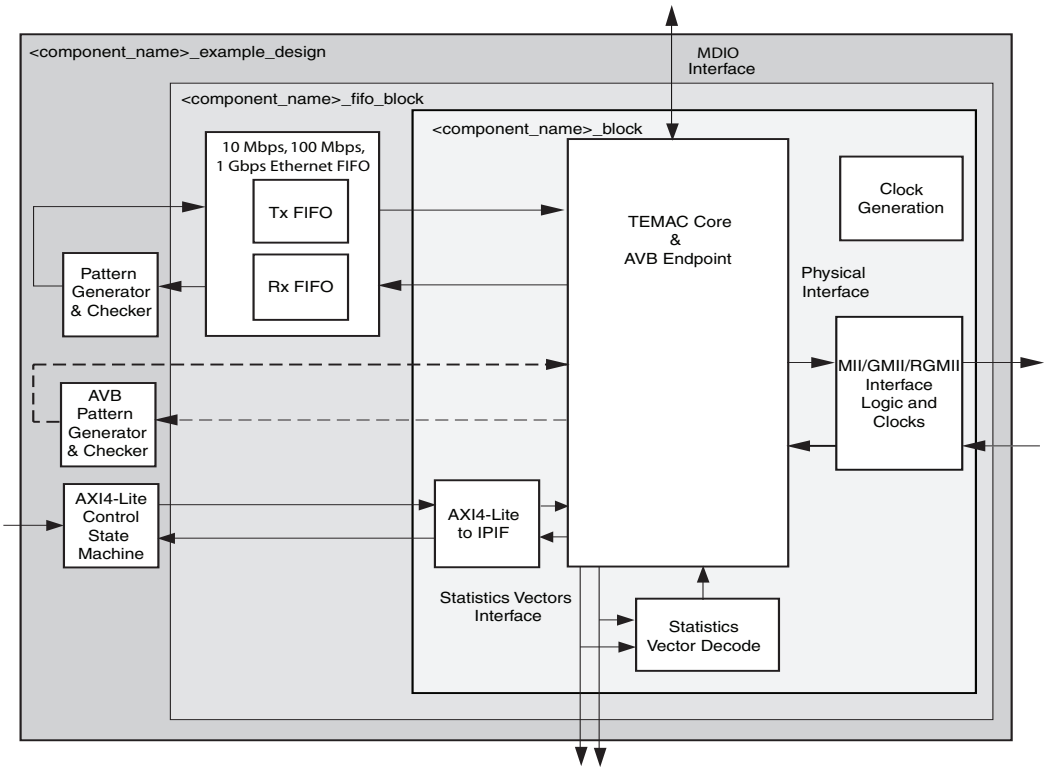

Figure 6-1: **Tri-Mode Ethernet MAC Core Example Design**

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, page 190).
  - 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 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 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-side interface of the core (GMII/MII or RGMII) to device IOBs 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 Native Generic Circuit (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 16, Implementing Your Design.

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.

Use Supported Design Flows

The core is pre-synthesized and delivered as an NGC netlist. The example implementation scripts provided use XST 13.4 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® v13.4 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 7

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 7-1.

Table 7-1: **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, page 31. All receiver signals are synchronous to the `rx_mac_aclk` clock.

Normal Frame Reception

Figure 7-1 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. When 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.

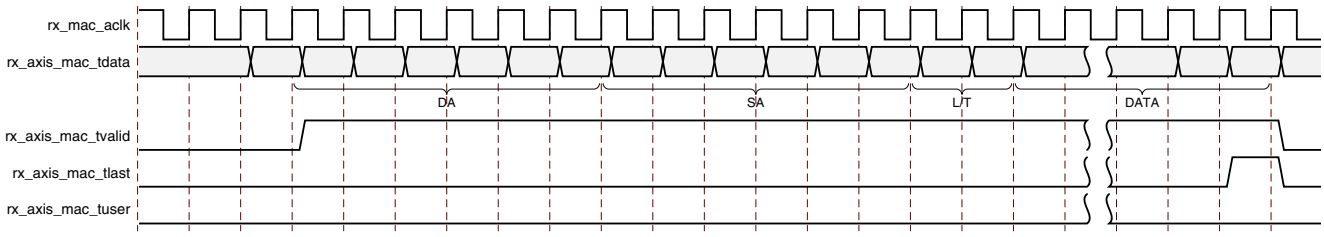

Figure 7-1: **Normal Frame Reception**

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 7-1.

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.

rx_axis_mac_tlast and rx_axis_mac_tuser Timing

Although Figure 7-1 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 7-2.

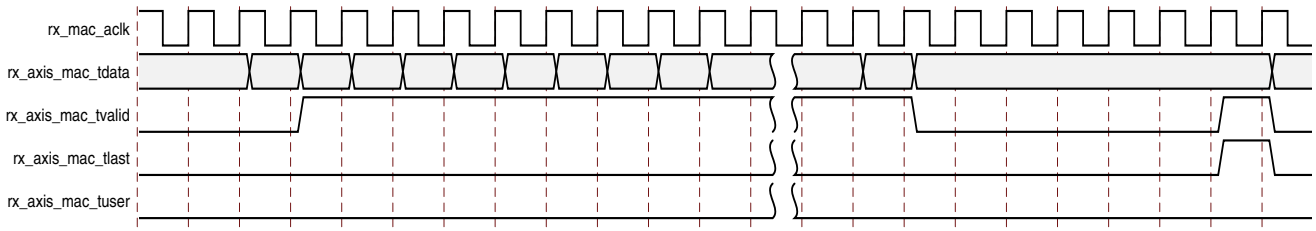

Figure 7-2: **Frame Reception TLast Timing**

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 7-3 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.

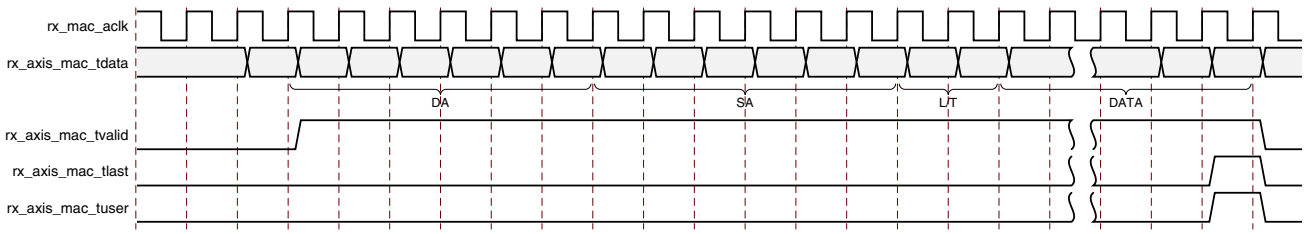

Figure 7-3: **Frame Reception with Error**

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 frame above the programmed Max Frame Size is received when Max frame length checking is 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, page 58).
- 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.
- A frame does not match against any of the enabled frame filters, if present.

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 7-4.

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.

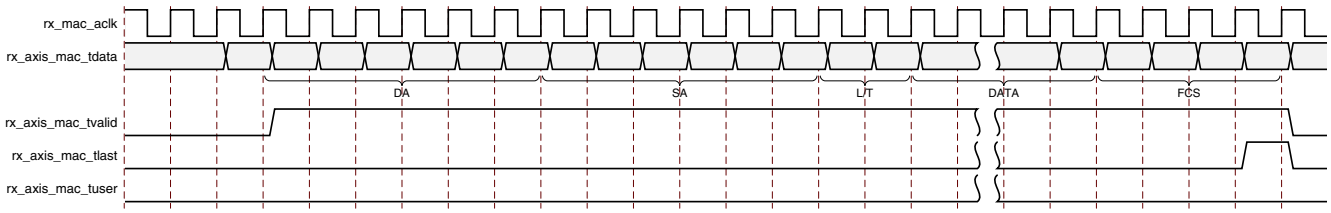

Figure 7-4: **Frame Reception with In-Band FCS Field**

VLAN Tagged Frames

The reception of a VLAN tagged frame can be seen in Figure 7-5. 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 the *IEEE 802.3-2008* standard.

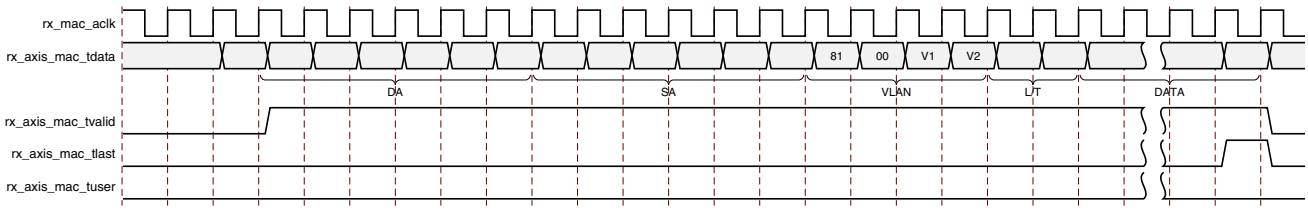

Figure 7-5: **Reception of a VLAN Tagged Frame**

Maximum Permitted Frame Length

The maximum legal length of a frame specified in *IEEE 802.3-2008* 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.

It is also possible to specify a different maximum frame size. If this is enabled and the frame exceeds the configured value then the frame is rejected, that is, `rx_axis_mac_tuser` is asserted at the end of the frame. In this case VLAN frames are not treated separately. If jumbo frame handling is enabled, that takes precedence and the configured value is ignored.

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.

Frame Filter

If the optional frame filter is included in the core, the MAC is able to reject frames that do not match against a recognized pattern i.e 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 Frame Filter, page 93.

Receiver Statistics Vector

The statistics for the frame received are contained within the rx_statistics_vector output. Table 7-2 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 7-6. BYTE_VALID is significant on every validated receiver cycle.

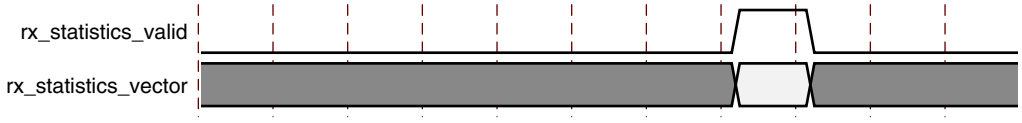

Figure 7-6: **Receiver Statistics Vector Timing**

Table 7-2: **Bit Definition for the Receiver Statistics Vector**

emacclient rxstats	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 below 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).

Table 7-2: **Bit Definition for the Receiver Statistics Vector *(Cont'd)***

emacclient rxstats	Name	Description
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 802.3-2008* maximum legal length (see Maximum Permitted Frame Length, page 46). 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.
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.

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, page 46.

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, page 29. 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 7-7. 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 TEMAC core accepts the first two bytes of data by asserting `tx_axis_mac_tready` and then waits until it is allowed to transmit and 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.

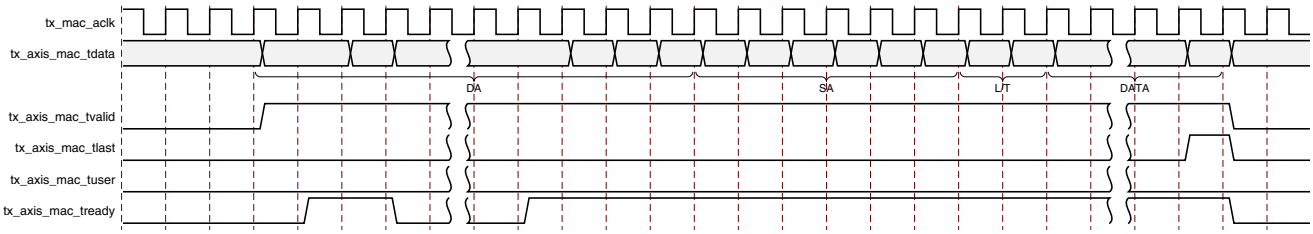

Figure 7-7: **Normal Frame Transmission Across User Interface**

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 7-8. 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 an errored frame as received by the link partner MAC (due to the detection of an FCS error).

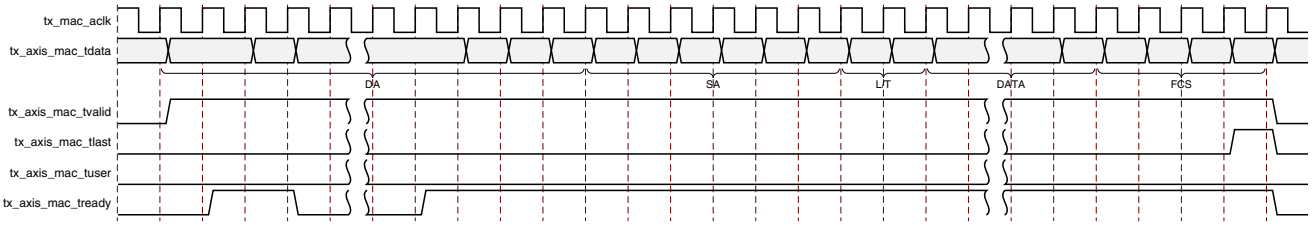

Figure 7-8: **Frame Transmission with User-Supplied FCS**

User Error Indication

Figure 7-9 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.

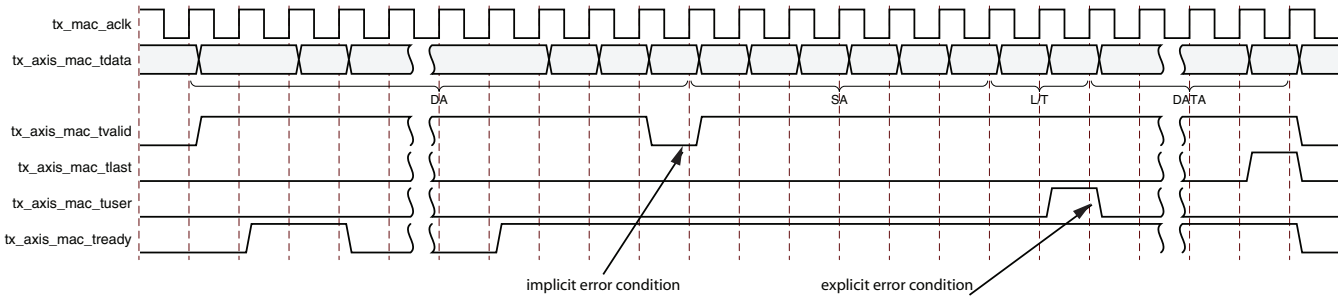

Figure 7-9: **Frame Transmission with Underrun**

Back-to-Back Transfers

Figure 7-10 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.

If the MAC core is operating at 1 Gb/s in half-duplex mode, the timing shown in Figure 7-10 is required to take advantage of frame bursting; the MAC core is only guaranteed to retain control of the medium if the `tx_axis_mac_tvalid` signal is high immediately after the end of the previous packet. For details on frame bursting, see *IEEE 802.3-2008*.

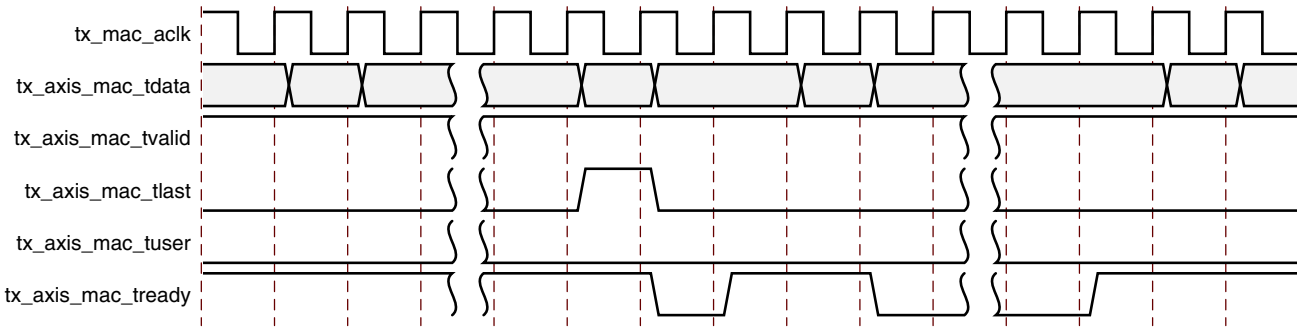

Figure 7-10: **Back-to-Back Frame Transmission**

VLAN Tagged Frames

Transmission of a VLAN tagged frame (if enabled) can be seen in Figure 7-11. 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 802.3-2008*.

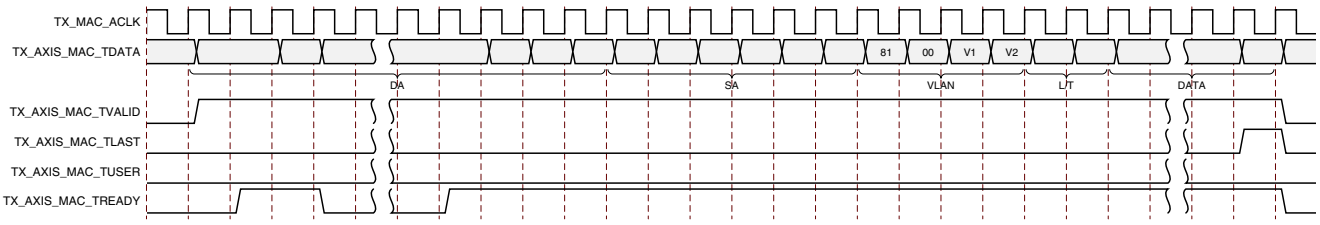

Figure 7-11: **Transmission of a VLAN Tagged Frame**

Maximum Permitted Frame Length

The maximum legal length of a frame specified in *IEEE 802.3-2008* 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.

It is also possible to specify a different maximum frame size. If this is enabled and the frame exceeds the configured value then the frame is corrupted. In this case VLAN frames are not treated separately. If jumbo frame handling is enabled, that takes precedence and the configured value is ignored.

Frame Collisions: Half-Duplex Operation Only

In half-duplex Ethernet operation, 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 6 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 7-12.

  If any frame presented to the user interface is shorter than the collision window (slot time) as defined in IEEE Std 802.3-2008, 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 8 cycles illustrated in Figure 7-12.

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), and 512 clock cycles when operating at 1 Gb/s speed (corresponding to 512-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 7-13.

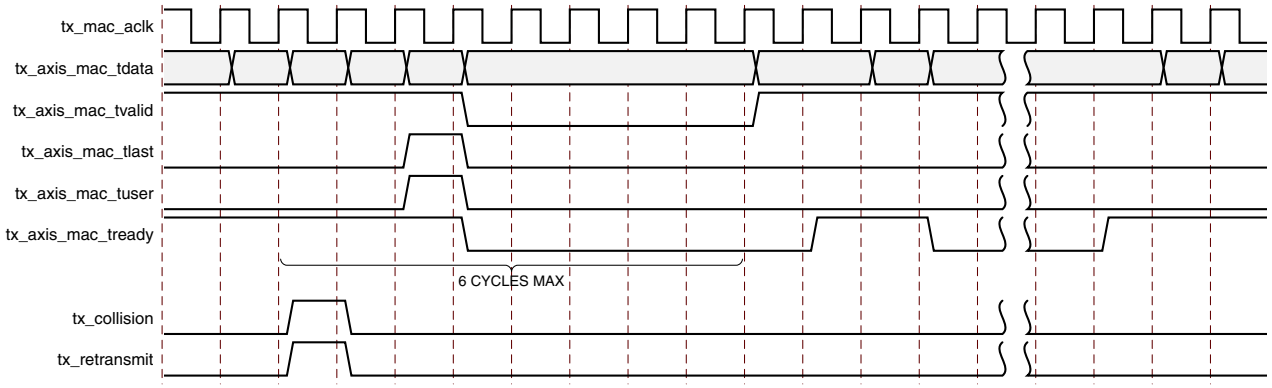

Figure 7-12: **Collision Handling: Frame Retransmission Required**

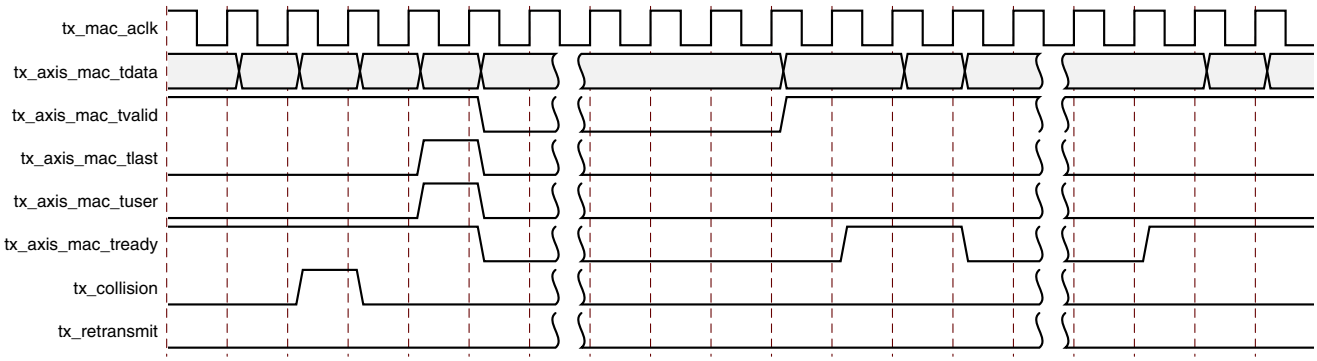

Figure 7-13: **Collision Handling: No Frame Retransmission Required**

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 7-14 shows the MAC operating in this mode.

The minimum interframe gap supported is dependent upon the support of half-duplex operation. If half duplex is supported, the minimum IFG possible is 8 transmit clock cycles. If the MAC only supports full-duplex operation then this reduces the minimum possible IFG to 4 transmit clock cycles. In both cases the interframe gap used when the **Interframe Gap Adjust Enable** bit is set to '0' is the minimum value as specified in the *IEEE 802.3-2008*. This corresponds to 12 transmit clock cycles on the GMMI/MII interface. The value on the `tx_ifg_delay` port must be larger than 4 or 8 to have an effect as described previously.

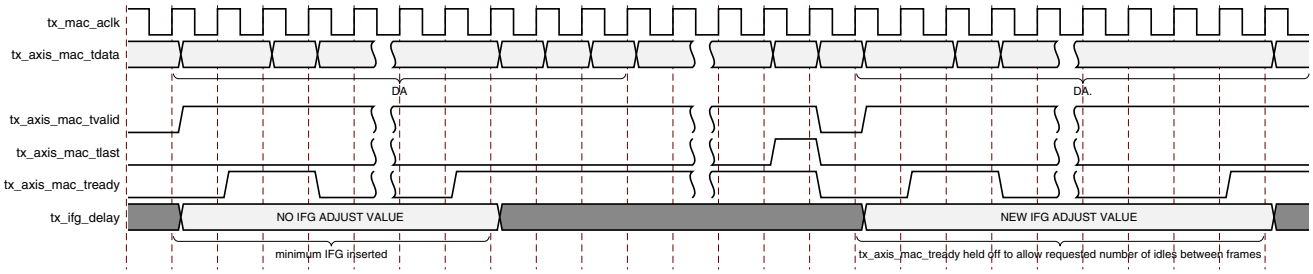

Figure 7-14: **Interframe Gap Adjustment**

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 7-3. All bit fields, with the exception of BYTE_VALID are valid only when the `tx_statistics_valid` is asserted, as illustrated in Figure 7-15. 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.

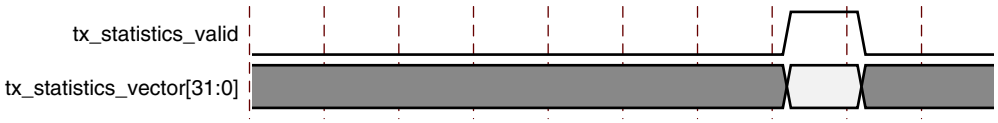

Figure 7-15: **Transmitter Statistics Vector Timing**

Table 7-3: **Bit Definition for the Transmitter Statistics Vector**

emacclient txstats	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.
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.

Table 7-3: **Bit Definition for the Transmitter Statistics Vector *(Cont'd)***

emacclient txstats	Name	Description
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 802.3-2008*.
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 8

Flow Control

This chapter describes the operation of the flow control logic of the TEMAC solution. The flow control block is designed to clause 31 of the *IEEE 802.3-2008* [Ref 10] standard. 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

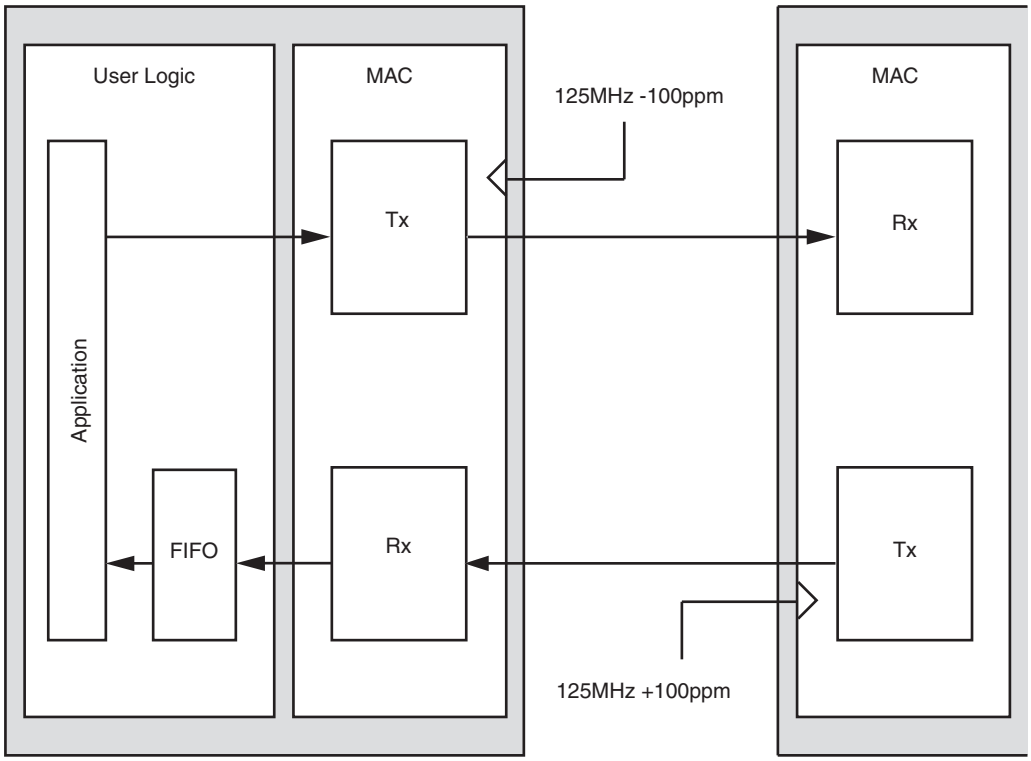

Figure 8-1: **Requirement for Flow Control**

Figure 8-1 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 8-1 can initiate a pause request when its user 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 8-1 can 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 user 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 8-2 illustrates control frame format.

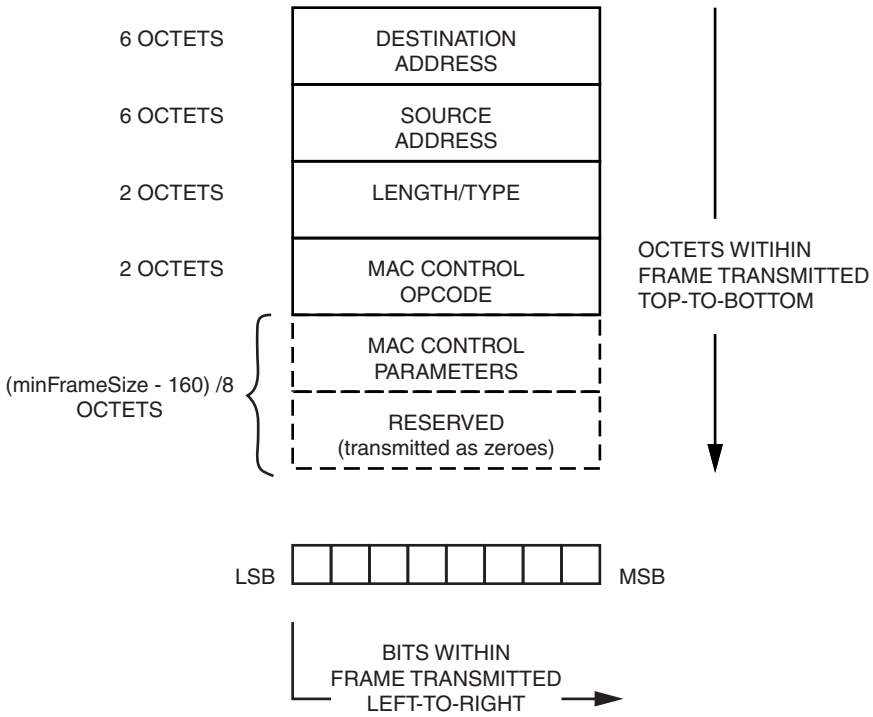

Figure 8-2: **MAC Control Frame Format**

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 TEMAC

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 8-3 illustrates this timing. Pause request signals are synchronous to the `gtx_clk` clock.

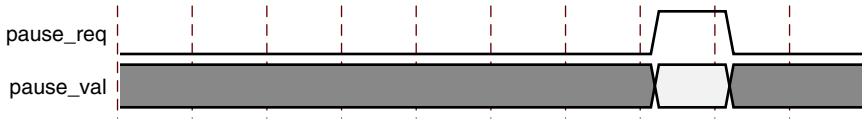

Figure 8-3: **Pause Request Timing**

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 8-2):

- 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, page 48).

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 8-2. 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, page 59).

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 an errored frame. In this case the frame is errored 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, page 47).

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 8-1. 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 8-4 illustrates the FIFO occupancy over time.

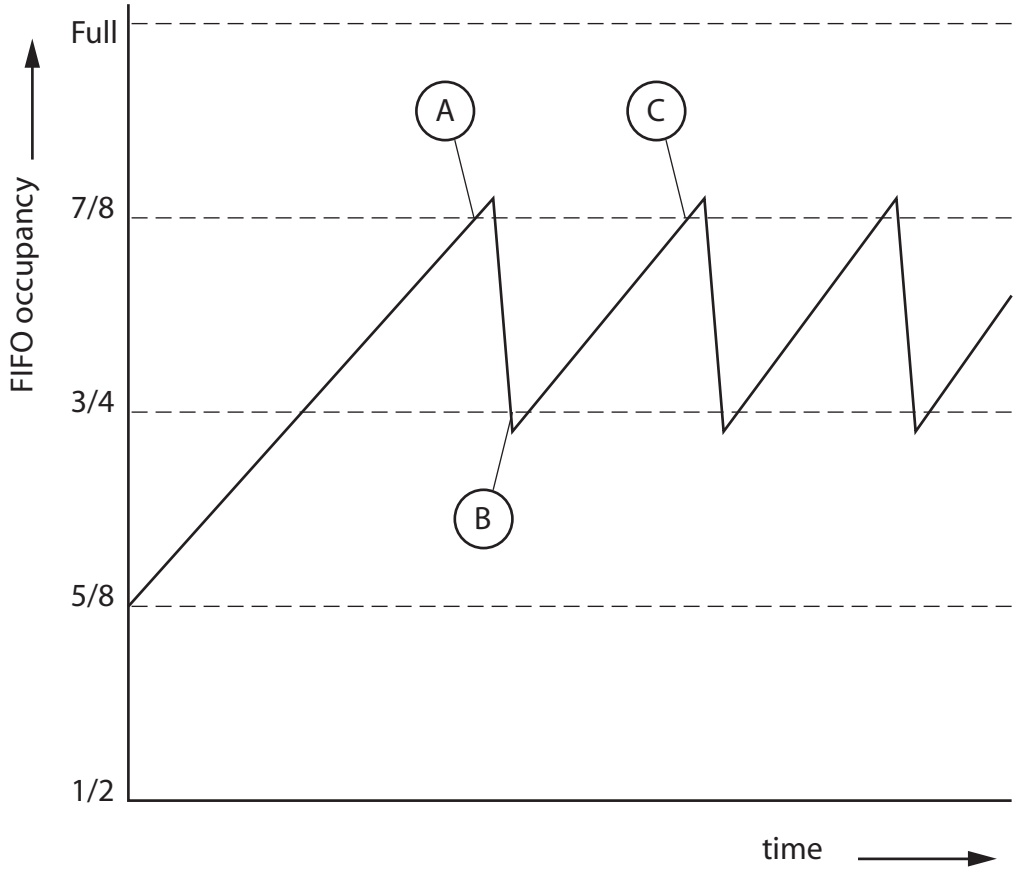

Figure 8-4: **Flow Control Implementation Triggered from FIFO Occupancy**

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.

Chapter 9

Ethernet AVB Endpoint

This chapter provides information about the key functional blocks which are introduced when the optional AVB Endpoint is included in the core.

Ethernet AVB Endpoint Transmission

As illustrated in Figure 9-1, data for transmission over an AVB network can be obtained from three types of sources:

1. **AV Traffic.** For transmission from the TX AV Traffic Interface of the core.
2. **Precise Timing Protocol (PTP) Packets**. Initiated by the software drivers using the dedicated hardware TX PTP Packet Buffer.
3. **Legacy Traffic**. For transmission from the TX Legacy Traffic Interface of the core.

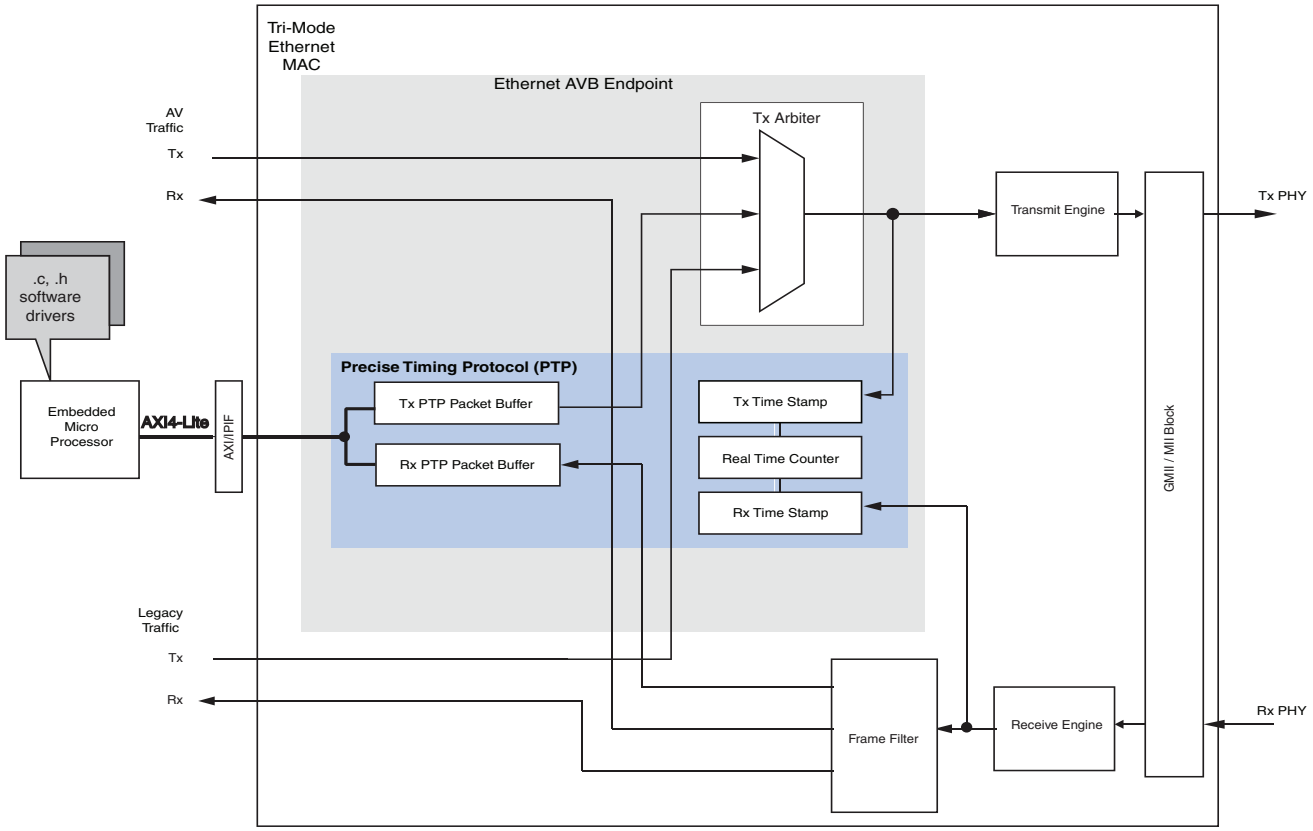

Figure 9-1: **Ethernet AVB Endpoint Data Path**

TX Legacy Traffic Interface

The legacy traffic interface is maintained for best effort Ethernet data: Ethernet as we know it today (for example, a PC surfing the internet). The signals forming the TX Legacy Traffic interface are defined in Table 5-1. The timing of this interface is as described in Transmitting Outbound Frames in Chapter 7, with the interface functionality limited to full-duplex, no jumbo support, VLAN frames enabled and flow control disabled.

TX AV Traffic Interface

The AV traffic interface is intended for the Quality of Service audio/video data. The Ethernet AVB Endpoint gives priority to the AV traffic interface over the legacy traffic interface, as dictated by IEEE 802.1Q 75% bandwidth restrictions. The signals forming the TX AV Traffic interface are defined in Table 5-3. The timing of this interface is exactly the same as for the TX Legacy Traffic with the only difference being how the `tvalid` signal is handled between frames.

In Figure 9-2, following the end of frame transmission, the `tx_axis_av_tvalid` signal is held high, which indicates to the TX Arbiter that another AV frame is queued. Unless the configurable bandwidth restrictions have been exceeded, this *parks* the TX Arbiter onto the AV traffic queue and the following frame can immediately be taken. However, if no further AV traffic frames are queued, the `tx_axis_av_tvalid` signal should be set to low immediately following the end of frame transmission. This then allows the TX Arbiter to schedule legacy traffic transmission (if any legacy frames are queued).

If, following the end of frame reception, the bandwidth allocation for AV traffic has been exceeded, the TX Arbiter switches to service the legacy traffic regardless of the state of the `tx_axis_av_tvalid` signal.

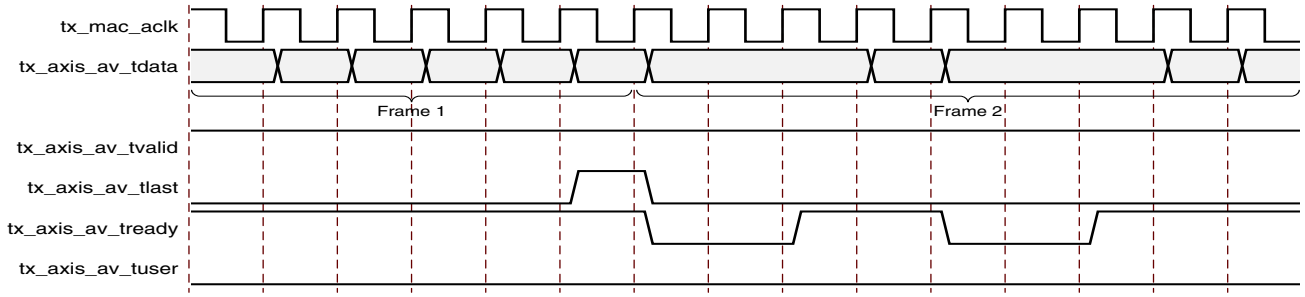

Figure 9-2: **TX AV Traffic Timing**

TX Arbiter

Overview

As illustrated in Figure 9-1, data for transmission over an AVB network can be obtained from three types of sources.

The transmitter (TX) arbiter selects from these three sources in the following manner.

- If there is AV data available and the programmed AV bandwidth limitation is not exceeded, then the AV packet is transmitted
- otherwise the TX arbiter checks to see if there are any PTP packets to be transmitted
- otherwise if there is an available legacy packet then this is transmitted.

The Ethernet AVB Endpoint uses configuration registers to set up the percentage of available Ethernet bandwidth reserved for AV traffic. To comply with the IEEE802.1Q specification these should not be configured to exceed 75%. The arbiter then polices this bandwidth restriction for the AV traffic and ensures that on average, it is never exceeded. Consequently, despite the AV traffic having a higher priority than the legacy traffic, there is always remaining bandwidth available to schedule legacy traffic.

The relevant configuration registers for programming the bandwidth percentage dedicated to AV traffic are defined in Chapter 10, Configuration and Status and are:

- TX Arbiter Send Slope Control Register
- TX Arbiter Idle Slope Control Register

These registers are defaulted to values which dedicate **up to** 75% of the overall bandwidth to the AV traffic. This is the maximum legal percentage that is defined in the *IEEE802.1* AVB standards.

In many implementations, it might be unnecessary to change these register values. Correct use of the `tx_axis_av_tvalid` signal, as defined in TX AV Traffic Interface, allows the TX Arbiter to share the bandwidth allocation efficiently between the AV and Legacy sources (even in the situations where the AV traffic requires less than 75% of the overall bandwidth).

However, for the cases that require less than 75% of the overall bandwidth, careful configuration can result in a *smoother* (less bursty) transmission of the AV traffic, which should prevent frame *bunching* across the AVB network.

Credit Based Traffic Shaping Algorithm

To enforce the bandwidth policing of the AV Traffic, a credit-based shaper algorithm has been implemented in the TX Arbiter. Figure 9-3 illustrates the basic operation of the algorithm and indicates how the TX Arbiter decides which Ethernet frame to transmit.

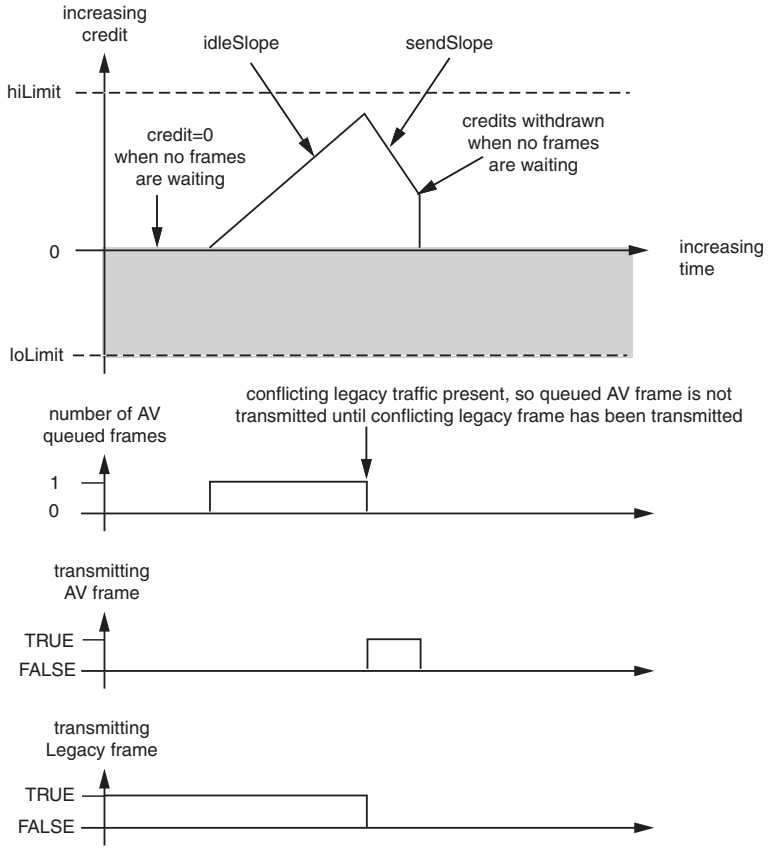

Figure 9-3: **Credit-based Shaper Operation**

Figure 9-3 illustrates the key features of the credit based algorithm, which are:

- The TX Arbiter schedules queued transmission from the TX AV Traffic Interface if the algorithm is in credit (greater or equal to 0).
- If there is less than 0 credit (not shown in Figure 9-3, but the credit can sink below 0), then the TX Arbiter does not allow AV traffic to be transmitted; legacy traffic, if queued, is scheduled instead.
- When no AV traffic is queued, any positive credit is lost and the credit is reset to 0.
- When AV traffic is queued, and until the time at which the TX Arbiter is able to schedule it (while waiting for an in-progress legacy frame to complete transmission), credit can be gained at a rate defined by the **idleSlope.**
- During AV traffic transmission, credit is removed at a rate defined by the **sendSlope.**
- The **hiLimit** and **loLimit** settings impose a fixed range on the possible values of credit. If the available credit hits one of these limits, it does not exceed, but saturates at the magnitude of that limit. These limits are fixed in the netlist to ensure that the interface is not used incorrectly.

The overall intention of the two settings **idleSlope** and **sendSlope** is to spread out the AV traffic transmission as evenly as possible over time, preventing periods of bursty AV

transmission surrounded by idle AV transmission periods. No further background information is provided in this document with regard to the credit-based algorithm. The remainder of this section describes the **idleSlope**, and **sendSlope** variables from the perspective of the TX Arbiter.

TX Arbiter Bandwidth Control

The configuration register settings, used for setting the cores local definitions of idleSlope and sendSlope, are described in general, and then from the point of view of a single example which describes the calculations made to set the register default values. This example dedicates up to 75% of the overall bandwidth to be reserved for the AV traffic (leaving at least 25% for the Legacy Traffic).

The calculations described are independent of Ethernet operating speed (no re-calculation is required when changing between Ethernet speeds of 1 Gb/s and 100 Mb/s).

idleSlope

The general equation is:

```
idleSlopeValue=(AV percentage / 100) x 8192
```

In this example, dedicating up to 75% of the total bandwidth to the AV traffic, we obtain:

```
idleSlopeValue=(75 / 100) x 8192 = 6144
```

The calculated value for the `idleSlopeValue` should be written directly to the TX Arbiter Send Slope Control Register. This provides a per-byte increment value when relating this to Legacy Ethernet frame transmission.

sendSlope

The general equation is:

```
sendSlopeValue=((100 - AV percentage) / 100) x 8192
```

In this example, dedicating up to 75% of the total bandwidth to the AV traffic, we obtain:

```
sendSlopeValue=((100 - 75) / 100) x 8192 = 2048
```

The calculated value for the `sendSlopeValue` should be written directly to the TX Arbiter Idle Slope Control Register. This provides a per-byte decrement value when relating this to AV Ethernet frame transmission.

Ethernet AVB Endpoint Reception

When the AVB Endpoint is present the Optional Frame Filter is always present with three dedicated filters for the identification of AVB specific frames. As shown in Figure 9-4 received data from an AVB network can be of three types:

- **Precise Timing Protocol (PTP) Packets**. Routed to the dedicated hardware RX PTP Packet Buffer which can be accessed by the software drivers. PTP packets are identified by searching for a specific MAC Destination Address and Type field.
- **AV Traffic**. Routed to the RX AV Traffic Interface of the core. These packets are identified by searching for MAC packets containing a MAC VLAN field with one of two possible configurable VLAN PCP and VID combinations.
- **Legacy Traffic:**. Routed to theRX Legacy Traffic Interface of the core. All packet types which are not identified as PTP or AV Traffic are considered legacy traffic.

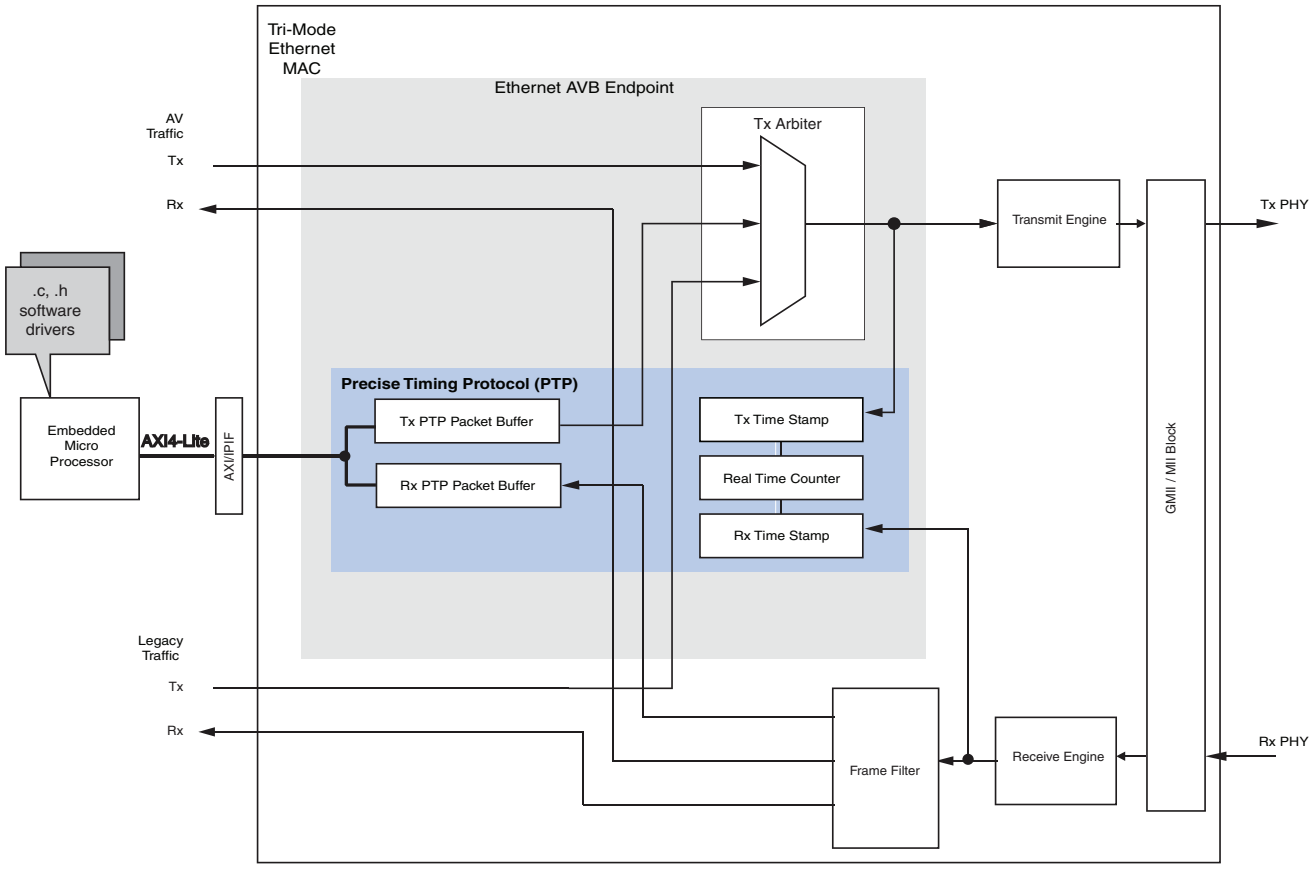

Figure 9-4: **Ethernet AVB Endpoint Data Path**

RX Legacy Traffic Interface

The signals forming the RX Legacy Traffic Interface are defined in Table 5-4. The timing of this interface is as described in Receiving Inbound Frames in Chapter 7, with the interface functionality limited to full-duplex, no jumbo support, VLAN frames enabled and no flow control.

RX AV Traffic Interface

The signals forming the RX AV Traffic Interface are defined in Table 5-6. The timing of this interface is exactly the same as for the RX Legacy Traffic (there is a one-to-one correspondence between signal names when the `axis_mac` is exchanged for `axis_av`).

The RX AV traffic is identified using the dedicated AVB Frame filters. These are only present when the AVB Endpoint is include in the TEMAC core and are initialized to match against the default AV VLAN p and VLAN Q values. These are described in more detail in Using the AVB Specific Frame Filters in Chapter 10.

Real Time Clock and Time Stamping

This chapter considers two of the logical components that are partially responsible for the AVB timing synchronization protocol.

- Real Time Clock
- Time Stamping Logic

These are both described in this chapter as they are closely related.

Real Time Clock

A significant component of the PTP network wide timing synchronization mechanism is the Real Time Clock (RTC), which provides the common time of the network. Every device on the network maintains its own local version.

The RTC is effectively a large counter which consists of a 32-bit nanoseconds field (the unit of this field is 1 nanosecond and this field counts the duration of exactly one second, then resets back to zero) and a 48-bit seconds field (the unit of this field is one second; this field increments when the nanosecond field saturates at 1 second). The seconds field only wraps around when its count fully saturates. The entire RTC is therefore designed never to wrap around in our lifetime. The RTC is summarized in Figure 9-5.

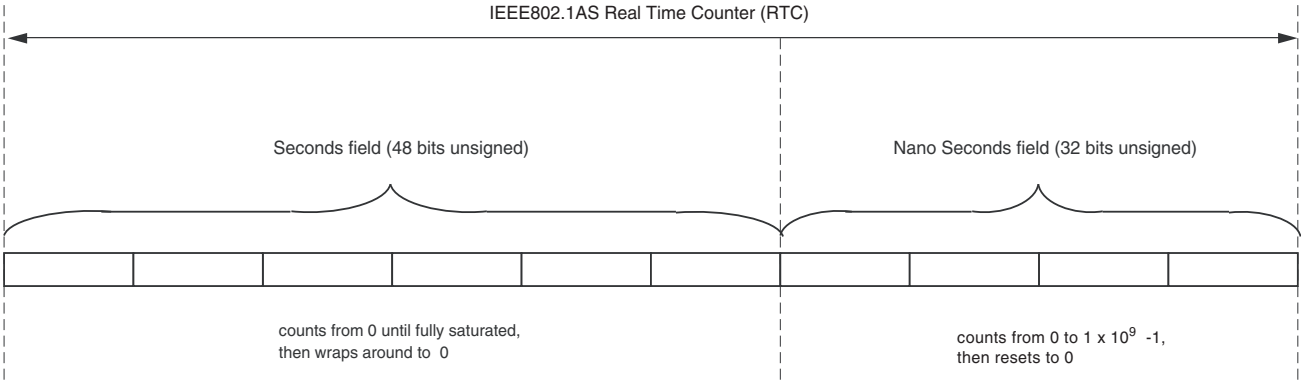

Figure 9-5: **Real Time Clock (RTC)**

Conceptually, the RTC is not related to the frequency of the clock used to increment it. A configuration register within the core provides a configurable increment rate for this counter: this increment register,RTC Increment Value Control Register, is for this reason programmed with the value of the RTC reference clock period which is being used to increment the RTC. The resolution of this increment register is very fine (in units of 1/1048576 ($1/2^{20}$) fraction of one nanosecond). Therefore, the RTC increment rate can be adjusted to a very fine degree of accuracy which provides the following features:

- The RTC can be incremented from any available clock frequency that is greater than the AVB standards defined minimum of 25 MHz. However, the faster the frequency of the clock, the smaller the step increment and the smoother the overall RTC increment rate. Xilinx recommends clocking the RTC logic at 125 MHz because this is a readily available clock source: this frequency significantly exceeds the minimum performance of the IEEE802.1AS specification.
- When acting as a clock slave, the rate adjustment of the RTC can be matched to that of the network clock master to an exceptional level of accuracy (by slightly increasing or decreasing the value within the RTC Increment Value Control Register). The software drivers (available separately) periodically calculate the increment rate error between themselves and the master, and update the RTC increment value accordingly.

The core also contains configuration registers, RTC Seconds Field Offset Control and RTC Nanoseconds Field Offset Control, which allow a large step change to be made to the RTC. This can be used to initialize the RTC, after power-up. It is also used to make periodic corrections, as required, by the software drivers when operating as a clock slave; however, if the increment rates are closely matched, these periodic step corrections will be small.

RTC Implementation

Increment of Nanoseconds Field

Figure 9-6 illustrates the implementation used to create the RTC nanoseconds field. This is performed by the use of an implementation specific 20-bit *sub-nanoseconds field* as illustrated. The nanoseconds and sub-nanoseconds fields can be considered to be concatenated together. All RTC logic within the core is synchronous to the RTC Reference Clock, `rtc_clk`.

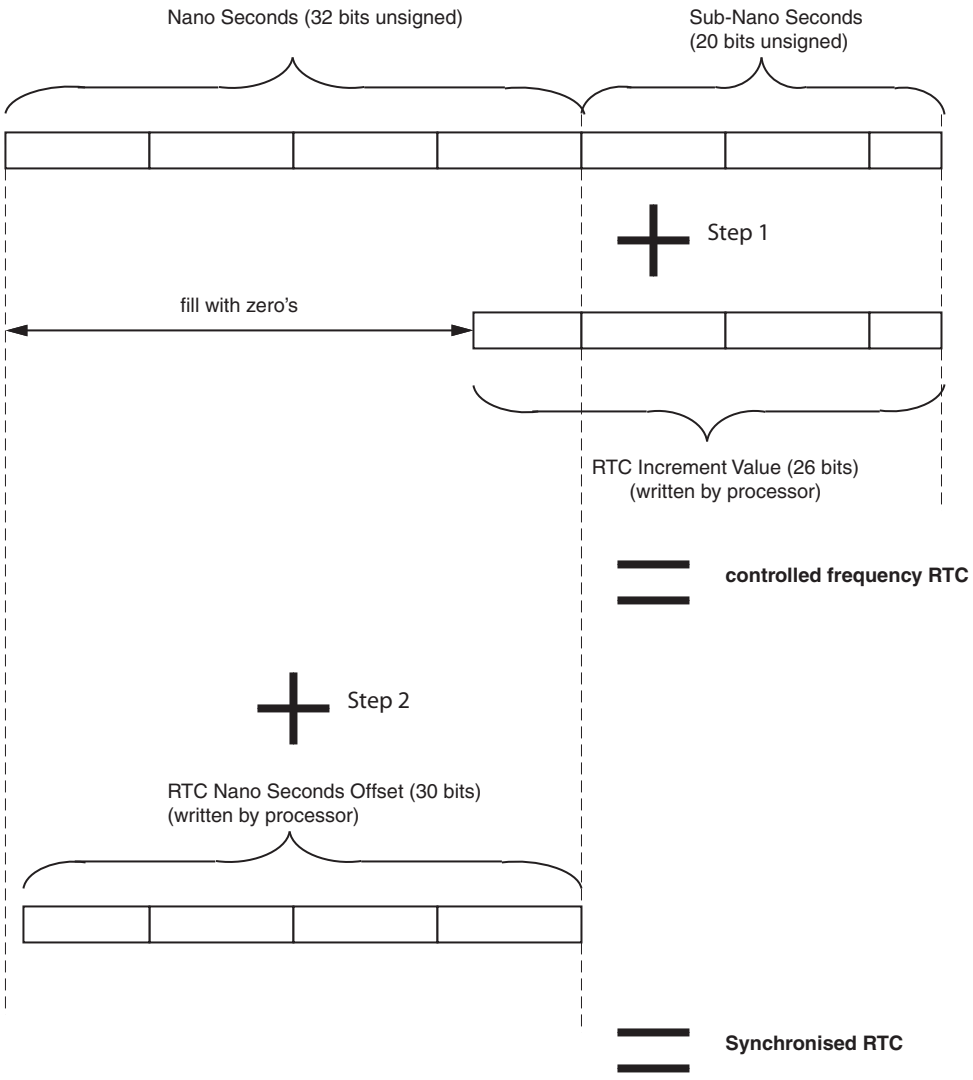

Figure 9-6: **Increment of Sub-nanoseconds and Nanoseconds Field**

There are two stages to the implementation:

1. Controlled Frequency RTC

The RTC Increment Value illustrated in Figure 9-6 is set directly from the RTC Increment Value Control Register. The upper 6 bits of this register align with the lower 6 bits of the RTC nanoseconds field. The lower 20-bits of the RTC Increment Value align with the 20-bit sub-nanoseconds field. It is assumed that the frequency of the RTC reference clock is known by the processor to enable the increment value to be programmed correctly. For example, if the RTC is being clocked from a 125 MHz clock source, a nominal increment value of 8 ns should be programmed (by writing the value 0x800000 into the RTC Increment Value Control Register). However, if the microprocessor determines that this clock is drifting with respect to the grand master clock, it can revise this nominal 8 ns up or down by a very fine degree of accuracy.

The 'step 1' addition illustrated in Figure 9-6 (of current counter value plus increment) occurs on every clock cycle of the RTC reference clock. The result from this addition forms the new value of the 'controlled frequency RTC' nanoseconds field. This controlled frequency RTC initializes to zero, following reset, and continues to increment smoothly on

every RTC reference clock cycle by the current value contained in the RTC Increment Value Control Register.

Figure 9-6 illustrates that 26 bits have been reserved for the Increment Value, the upper 6-bits of which overlap into the nanoseconds field. For this reason, the largest per-cycle increment = 1ns * 2^6 = 64 ns. The lowest clock period which is expected to increment this counter is 40 ns (corresponding to the 25 MHz MAC clock used at 100 Mb/s speeds). So this should satisfy all allowable clock periods.

2. Synchronized RTC

The value contained in the RTC Seconds Field Offset Control and RTC Nanoseconds Field Offset Control written by the microprocessor, is then applied to the free running 'controlled frequency RTC' counter. This is used by the microprocessor to:

- Initialize the power-up value of the Synchronized RTC.
- Apply step corrections to the Synchronized RTC (when a slave), based on the timing PTP packets received from the Grand Master Clock RTC.

The 'step 2' addition illustrated in Figure 9-6 (of controlled frequency RTC value plus offset) occurs on every clock cycle of the RTC reference clock. The result from this addition forms the new value of the Synchronized RTC nanoseconds field. It is this version of the RTC nanoseconds field which is made available as an output of the core - the `rtc_nanosec_field[31:0]` port.

Increment of the Seconds Field

The RTC seconds field is, conceptually, implemented in a similar way to the nanoseconds field. The seconds field should be incremented by a value of one whenever the synchronized RTC nanoseconds field saturates at one-second. The RTC Seconds Field Offset Control and RTC Nanoseconds Field Offset Control allow the software to make large step corrections to the seconds field in a similar manner. Again, the step correction capability can be used to either initialize the RTC counter following reset, or to synchronize the local RTC to that of the Grand Master Clock (when the local device is acting as a clock slave).

Clock Outputs Based on the Synchronized RTC Nanoseconds Field

The `clk8k` (8 kHz clock) output, derived from the Synchronized RTC, is provided as an output from the core. The synchronized RTC counter, unlike the controlled frequency version, has no long-term drift (assuming the provided software drivers are used correctly). Therefore, the `clk8k` signal is synchronized exactly to the network RTC frequency.

The 8 kHz clock is the period of the shortest class measurement interval for an SR class as specified in IEEE802.1Q. This clock could also be useful for external applications (for example, a 1722 implementation of the AV traffic).

Time Stamping Logic

Whenever a PTP packet, used with the Precise Timing Protocol (PTP), is transmitted or received (see Precise Timing Protocol Packet Buffers), a sample of the current value of the RTC is taken and made available for the software drivers to read. The hardware makes no distinction between frames carrying event or general PTP messages (as defined in IEEE 802.1AS); it always stores a timestamp value for ethernet frames containing the Ethertype specified for PTP messages.

This time stamping of packets is a key element of the tight timing synchronization across the AVB network wide RTC, and these samples must be performed in hardware for accuracy. The hardware in this core therefore samples and captures the local nanoseconds RTC field for every PTP frame transmitted or received. These captured time stamps are stored in the Precise Timing Protocol Packet Buffers alongside the relevant PTP frame, and are read and used by the PTP software drivers.

It is important to realize that is it actually the 'controlled frequency RTC' nanoseconds field which is sampled by the time stamping logic rather than the synchronized RTC (see Figure 9-6). This is important when operating as a clock slave: the controlled frequency RTC always acts as a smooth counter whereas the synchronized RTC might suffer from occasional step changes (whenever a new offset adjustment is periodically applied by the software drivers). These step changes, avoided by using the controlled frequency RTC, could otherwise lead to errors in the various PTP calculations which are performed by the software drivers.

Note: The software drivers can themselves obtain (when required) the local synchronized RTC value by summing the captured time stamp with the current nanoseconds offset value of the RTC Nanoseconds Field Offset Control (effectively performing the step 2 calculation of Figure 9-6 in software).

Time Stamp Sampling Position of MAC Frames

A time stamp value should be sampled at the beginning of the first symbol following the Start of Frame Delimiter (SFD) of the Ethernet MAC frame.

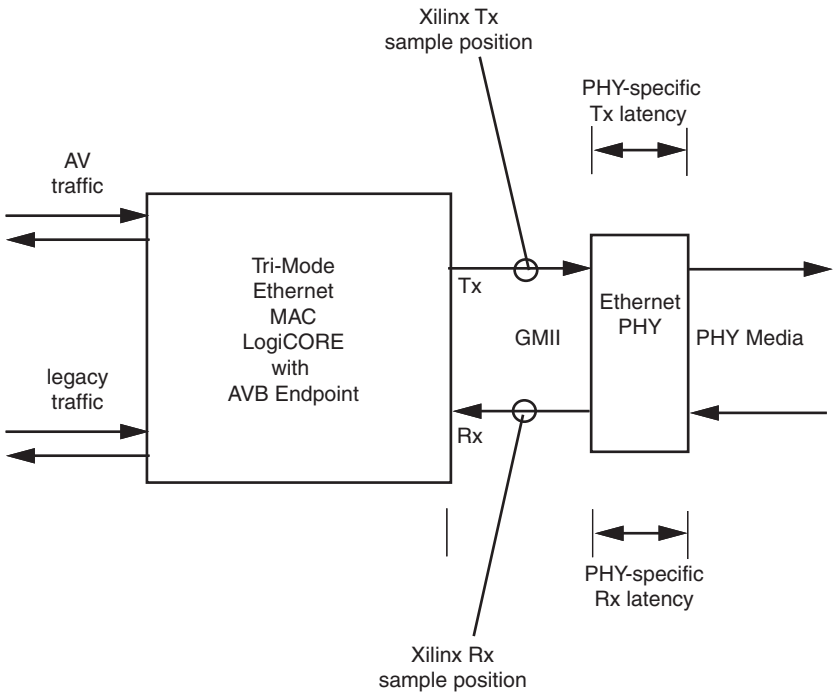

Figure 9-7: **Time Stamping Position**

Figure 9-7 illustrates the time stamp sampling position that is used by the core. Time stamps are taken after the MAC frame SFD is seen on the GMII.

Note: If the PHY specific latency values are available the software drivers can use them to adjust the timestamps and improve overall system accuracy.

IEEE1722 Real Time Clock Format

The IEEE1722 specification defines the *avbtp_timestamp* field. This is derived by sampling the IEEE802.1 AS Real Time Clock and converting the low order time to nanoseconds. This conversion is performed in the core and an alternative RTC, in the 1722 format, is output on the `rtc_nanosec_field_1722[31:0]` port.

This port contains a 32-bit word representing nanosecond values. Unlike the IEEE802.1 AS nanosecond field (which resets back to zero when it reaches 1 second), the IEEE1722 nanosecond field counts fully to 0xFFFFFFFF before wrapping around. The field therefore wraps around approximately every 4 seconds.

If the system is using the IEEE1722 functionality, this port can be sampled to create the *avbtp_timestamp* field. Otherwise this port can be ignored.

Precise Timing Protocol Packet Buffers

This chapter considers two of the logical components which are partly responsible for the AVB timing synchronization protocol.

- TX PTP Packet Buffer
- RX PTP Packet Buffer

These are both described in this chapter as they are closely related.

TX PTP Packet Buffer

The TX PTP packet buffer is illustrated in Figure 9-8. This packet buffer provides working memory to hold the PTP frames which are required for transmission. The software drivers, through the AXI4-Lite configuration bus, can read/modify/write the PTP frame contents, and whenever required, can request transmission of the appropriate PTP frames.

The PTP packet buffer is implemented in dual-port block RAM. Port A of the block RAM is connected to the configuration bus and all addresses in the buffer are read/writable. Port B of the block RAM is connected to the TX Arbiter module, allowing PTP frames to be read out of the block RAM and transmitted.

The TX PTP Packet Buffer is divided into eight identical buffer sections as illustrated. Each section contains 256 bytes, which are formatted as follows:

- the first byte, at address zero, contains a frame length field. This indicates how many bytes make up the PTP frame that is to be transmitted from this particular PTP buffer.
- The next seven bytes, from address 1 to 7, are reserved for future use.
- The PTP frame data itself is stored from address 8 onwards. The amount of addresses used is dependent on the indicated frame length field, which is different for each PTP frame type. Each PTP buffer provides a maximum of 244 bytes (more than that required for the largest PTP frame). Each PTP frame holds the entire MAC frame (with the exception of any required MAC padding or CRC - these are automatically inserted by the transmit logic) from the Destination Address field onwards.
- The top four addresses of each buffer, from address 0xFC to 0xFF are reserved for a time stamp field. At the beginning of PTP frame transmission from any of the eight buffers, the Time Stamping Logic samples the Real Time Clock. Following the end of PTP frame transmission, this captured timestamp is automatically written into this location to accompany the frame for which it was taken.

Despite the logic and formatting of each individual PTP buffer being identical, the block RAM is pre-initialized at device configuration to hold template copies of each of the PTP frames, as indicated in Figure 9-8. This shows that the first seven memory segments are in use. PTP Buffer number 8 is currently unused and could therefore be used by proprietary applications.

The TX PTP Packet Control Register is defined for the purpose of requesting which of the eight TX PTP Buffers are to be transmitted. It is possible to request more than a single frame at one time (indeed it is possible to request all 8). When more than one frame is requested, the TX PTP Buffer logic gives a priority order to the lowest PTP Buffer Number that has been requested.

The TX PTP Packet Control Register also contains a frame waiting field. This can be read by the software drivers to determine which of the previously requested PTP frames have been sent, and which are still queued.

Following transmission completion of each requested PTP frame, a dedicated interrupt signal, `interrupt_ptp_tx`, is generated by the core. On the assertion of the interrupt, the captured timestamp is already available in the upper four bytes of the buffer, and the `tx_packet` field of theTX PTP Packet Control Register indicates the most recently transmitted Buffer Number.

The software drivers, available from Xilinx, using the AXI4-Lite and dedicated interrupts, use this interface to, as defined by the IEEE802.1AS protocol, periodically update specific fields within the PTP packets, and request transmission of these packets.

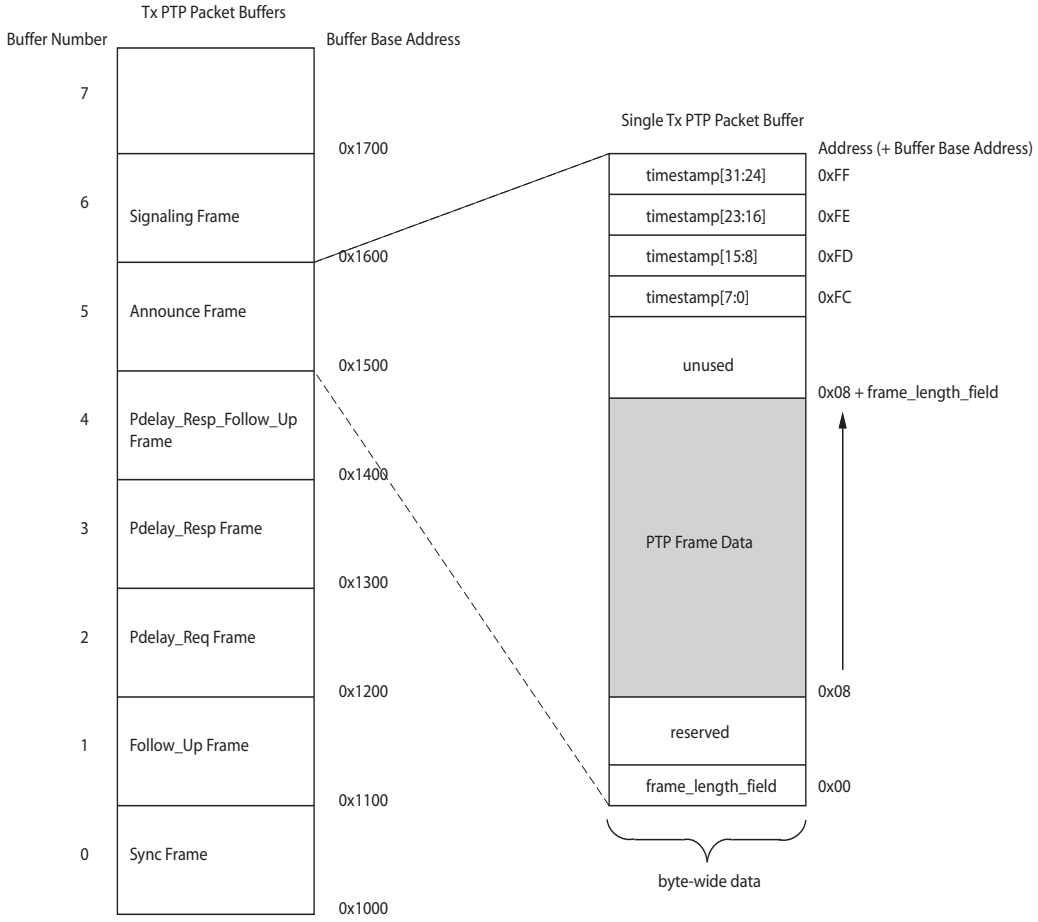

Figure 9-8: **TX PTP Packet Buffer Structure**

RX PTP Packet Buffer

The RX PTP packet buffer is illustrated in Figure 9-9. This provides working memory to hold each received PTP frame. The software drivers, using the AXI4-Lite configuration bus, can then read and decode the contents of the received PTP frames. The PTP packet buffer is implemented in dual-port block RAM. Port A of the block RAM is connected to the configuration bus and all addresses in the buffer can be read (writes are not allowed). Port B of the block RAM is connected to the PTP Frame Filter, which routes all received PTP frames into the RX PTP Packet Buffer. The RX PTP Packet Buffer is divided into sixteen identical buffer sections as illustrated. Each section contains 256 bytes, which are formatted as follows:

- The PTP frame data itself is stored from address 0 onwards: the entire MAC frame from the Destination Address onwards is written (with the exception of the FCS field which has been removed by the receive logic). The number of addresses used is dependent on the particular PTP frame size, which is different for each PTP frame type. Each PTP buffer provides a maximum of 252 bytes (more than that required for the largest PTP frame). Should an oversized PTP frame be received, the first 252 bytes is captured and stored - other bytes are lost.
- The top four addresses of each buffer, from address 0xFC to 0xFF are reserved for a timestamp field. At the beginning of PTP frame reception, the Time Stamping Logic samples the Real Time Clock. Following the end of PTP frame reception, this captured timestamp is automatically written into this location to accompany the frame for which it was taken.

Following reset, the first received PTP frame is written into Buffer Number 0. The next subsequent received PTP frame is written into the next available buffer - in this case number 1. This process continues with buffer number 2, 3, then 4, and so forth, being used. After receiving the 16th PTP frame (which would have been stored into buffer number 15), the count is reset, and then buffer number 0 is overwritten with the next received PTP frame. For this reason, at any one time, the RX PTP Packet Buffer is capable of storing the most recently received sixteen PTP frames. Following the completion of PTP frame reception, a dedicated interrupt signal, `interrupt_ptp_rx`, is generated by the core. On the assertion of the interrupt, the captured timestamp is already available in the upper four bytes of the buffer, and the `rx_packet` field of the RX PTP Packet Control Register indicates the most recently filled Buffer Number. The software drivers, available from Xilinx, using the AXI4-Lite and dedicated interrupt, use this interface to decode, and then act on, the received PTP packet information.

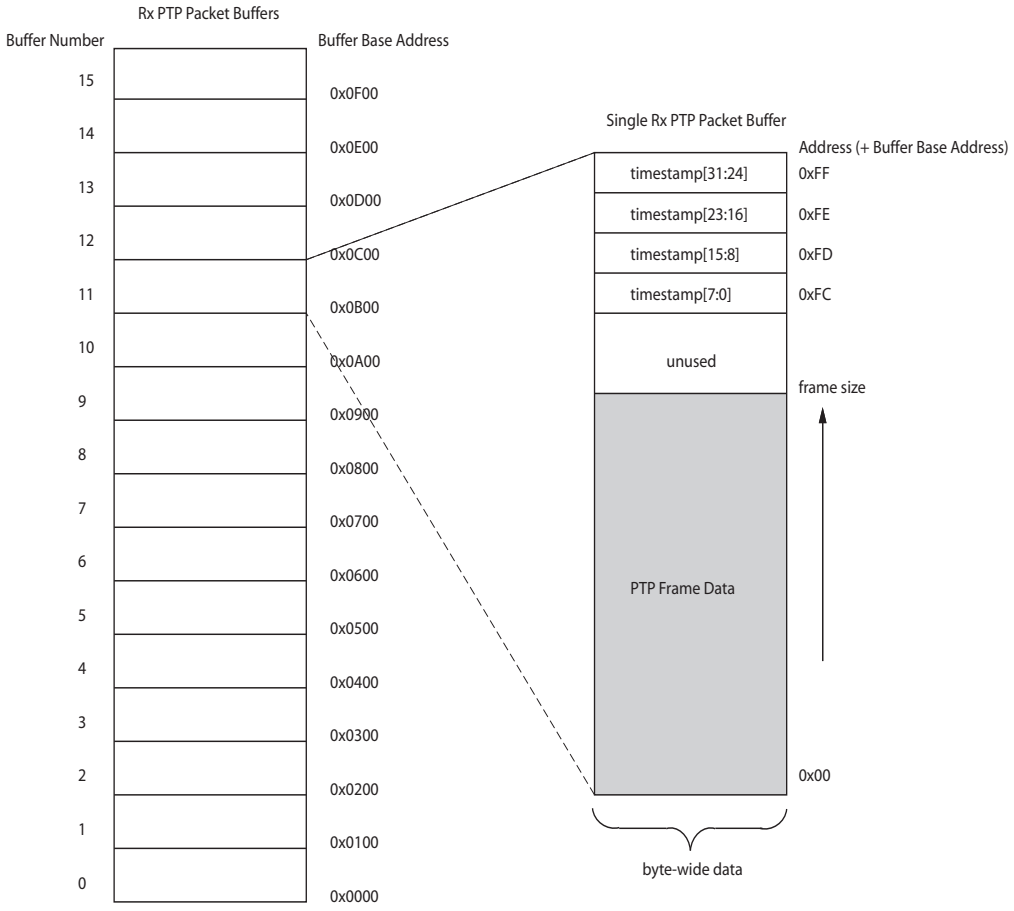

Figure 9-9: **RX PTP Packet Buffer**

XILINX®

Chapter 10

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. It also describes the alternative to the optional management interface which is the Configuration Vector. See the appropriate section:

- The Management Interface
- The Configuration Vector

The Management Interface

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

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

Table 10-1 describes the optional signals used by the user to access the MAC netlist.

Table 10-1: **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

Table 10-1: **Optional AXI4-Lite Signal Pinout *(Cont'd)***

Signal	Direction	Clock Domain	Description
s_axi_arvalid	Input	s_axi_aclk	Read Address valid
s_axi_arready	Output	s_axi_aclk	Read Address ready
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 10-1 and Figure 10-2 show the basic AXI4-Lite transactions supported by the MAC solution. Illegal accesses results in an error indication. See [Ref 1] for more information about this standard.

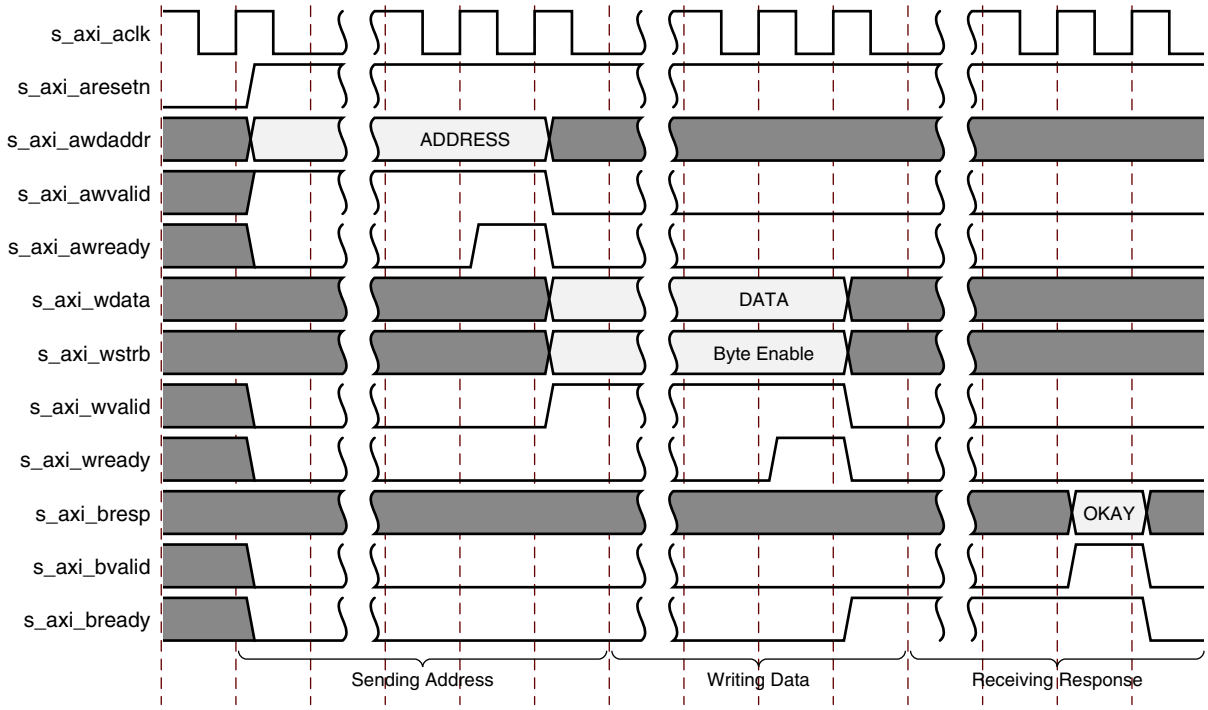

Figure 10-1: **Management Register Write Timing**

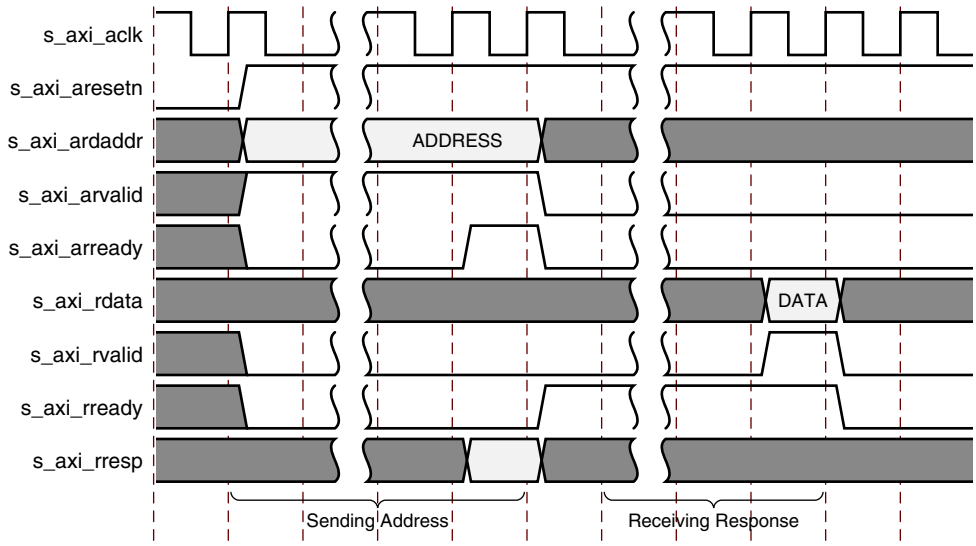

Figure 10-2: **Management Register Read Timing**

Address Map

The MAC address map, as shown in Table 10-2, 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 distinct functional blocks. Each of these register blocks and the individual registers is described in more detail in the following sections.

Table 10-2: **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	Frame Filter
0x800-0xFFFC	Reserved
0x10000-0x0FFC	RX PTP Packet Buffer Address Space
0x11000-0x117FC	TX PTP Packet Buffer Address Space
0x11800-0x11FFC	Reserved
0x12000-0x127FC	AVB Configuration
0x12800-0x13FFC	RTC Configuration

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 64 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 wrap around 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 TEMAC core only targets families which have a LUT6 based architecture, the statistics counters are based around distributed memory. The TEMAC core always outputs 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.

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 10-3 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 10-3 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 10-3.

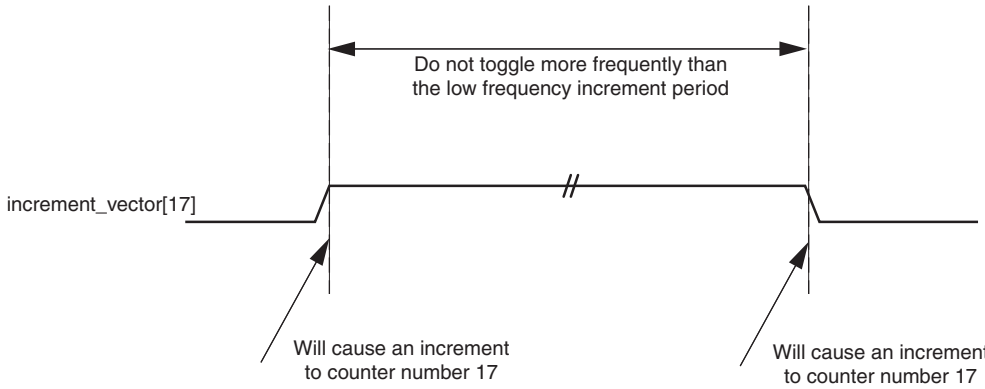

Figure 10-3: **Increment Vector Timing Diagram**

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 MAC. The low-frequency increment vectors can update at a maximum rate of once per minimum sized Ethernet frame. This translates to 584 ns when running at 1 Gb/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, we can safely update 36 statistical counters 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 the 44 counters supported only 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 MAC (125 MHz at 1 Gb/s or 12.5 MHz at 10/100 Mb/s).

Table 10-3: **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.
RX 1024-MaxFrameSize byte Frames	9	0x248-0x24C	A count of error-free frames received between 1024 bytes and the specified *IEEE 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 802.3-2008*.

Table 10-3: **Statistics Counter Definitions *(Cont'd)***

Name	Increment Bit No.	Address	Description
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 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 802.3-2008*.
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 user 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 user 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.
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.

Table 10-3: **Statistics Counter Definitions *(Cont'd)***

Name	Increment Bit No.	Address	Description
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).
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, users 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 10-4.

Table 10-4: **Configuration Registers**

Address (Hex)	Description
0x400	Receiver Configuration Word 0
0x404	Receiver Configuration Word 1
0x408	Transmitter Configuration
0x40C	Flow Control Configuration

Table 10-4: **Configuration Registers *(Cont'd)***

Address (Hex)	Description
0x410	Speed configuration
0x414	RX Max Frame Configuration
0x418	TX Max Frame Configuration
0x41C-0x4F4	Reserved
0x4F8	ID Register
0x4FC	Ability Register

The contents of each configuration register are shown in Table 10-5 through Table 10-13.

Table 10-5: **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 10-6: **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 10-5.
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, page 46. When this bit is set to '0,' the length/type field checks is performed: this is normal operation.
26	0	RW	**Half Duplex**: 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 user as described in User-Supplied FCS Passing, page 45. When it is '0,' the user is not passed 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 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 10-7: **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, page 52). If '0,' the transmitter outputs a minimum interframe gap of at least twelve clock cycles, as specified in IEEE 802.3-2008.
26	0	RW	**Half Duplex**: 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 user as described in the User-Supplied FCS Passing, page 49. 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 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 10-8: **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, page 58. 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, page 57. When this bit is '0', asserting the pause_req signal has no effect.
31	N/A	RO	Reserved

Table 10-9: **MAC Speed Configuration Word (0x410)**

Bits	Default Value	Type	Description
29-0	N/A	RO	Reserved
31-30	10	RW	**MAC Speed Configuration** 00 - 10 Mb/s 01 - 100 Mb/s 10 - 1 Gb/s When the TEMAC solution has been generated for only 1 Gb/s speed support, bits 31-30 are hard-coded to the value '10'. When the TEMAC solution has been generated for only 10 Mb/s and 100 Mb/s speed support, bits 31-30 only accept the values of '00' to configure for 10 Mb/s operation, or '01' to configure for 100 Mb/s operation

Note: The setting of the MAC Speed Configuration register is not affected by a reset.

Table 10-10: **RX Max Frame Configuration Word (0x414)**

Bits	Default Value	Type	Description
14-0	0x5EE	RW	**RX Max Frame Length**
15	N/A	RO	Reserved
16	0	RW	**RX Max Frame Enable:** When low, the MAC assumes use of the standard 1518/1522 depending upon the setting of **VLAN enable**. When high, the MAC allows frames up to **RX Max Frame Length** irrespective of the value of **VLAN enable**. If **Jumbo Enable** is set then this register has no effect.See Maximum Permitted Frame Length, page 46
31-17	N/A	RO	Reserved

Table 10-11: **TX Max Frame Configuration Word (0x418)**

Bits	Default Value	Type	Description
14-0	0x5EE	RW	**TX Max Frame Length**
15	N/A	RO	Reserved
16	0	RW	**TX Max Frame Enable:** When low the MAC assumes use of the standard 1518/1522 depending upon the setting of **VLAN enable**. When high the MAC allows frames up to **TX Max Frame Length** irrespective of the value of **VLAN enable**. If **Jumbo Enable** is set then this register has no effect. See Maximum Permitted Frame Length, page 51
31-17	N/A	RO	Reserved

Table 10-12: **ID Register (0x4F8)**

Bits	Default Value	Type	Description
7-0	0	RO	Patch Level (0x00- No Patch, 0x01 - Rev1...)
15-0	N/A	RO	Reserved
23-16	0x2	RO	Minor Revision
31-24	0x5	RO	Major Revision

Table 10-13: **Ability Register (0x4FC)**

Bits	Default Value	Type	Description
0	1*	RO	**10M Ability**: If set, the core is 10 M capable
1	1*	RO	**100M Ability**: If set, the core is 100 M capable
2	1*	RO	**1G Ability**: If set, the core is 1 G capable
3-7	N/A	RO	Reserved
8	1*	RO	Statistics Counters available
9	1*	RO	Half duplex capable
10	1*	RO	Frame Filter available
11	1*	RO	AVB Endpoint available
31-12	N/A	RO	Reserved

Note: * Depends upon core abilities selected

MDIO Interface

The Management Interface is also used to access the MDIO Interface of the TEMAC core; this interface is used to access the Managed Information Block of the PHY components attached to the TEMAC 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 Management Data Clock (MDC) and a shared serial data line (MDIO). The maximum permitted frequency of MDC is set at 2.5 MHz. Figure 10-4 illustrates an example MDIO bus system.

An Ethernet MAC 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).

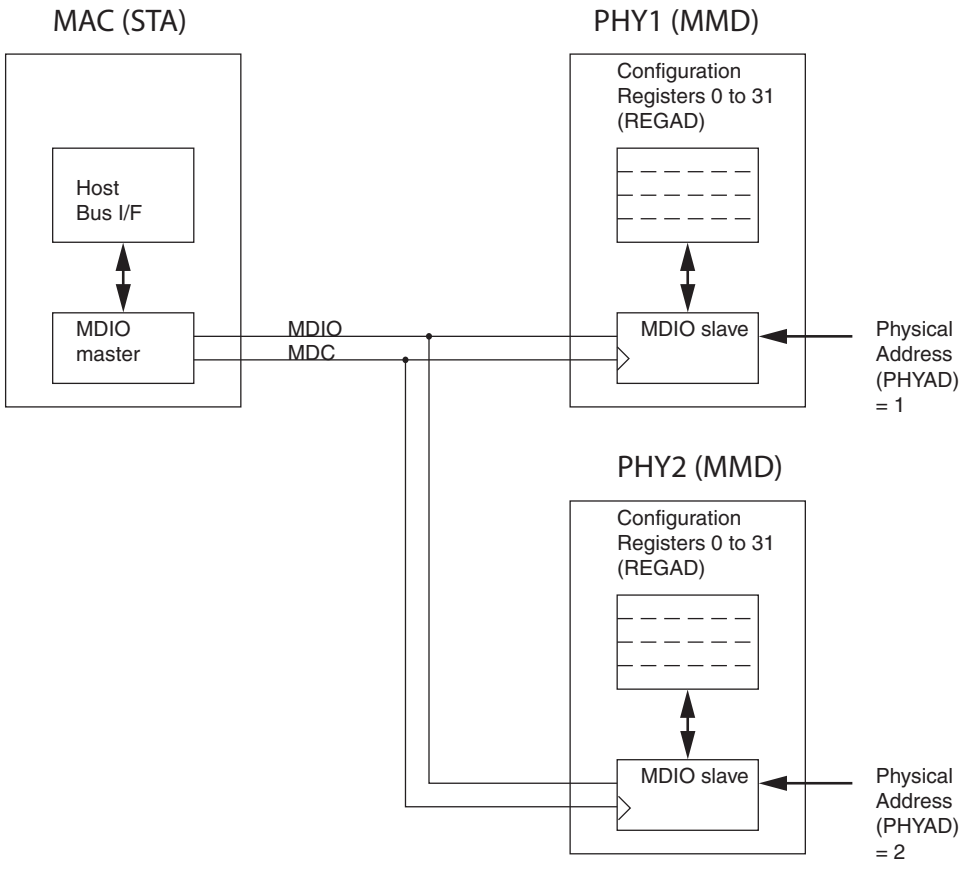

Figure 10-4: **A Typical MDIO-Managed System**

The MDIO bus system is a standardized interface for accessing the configuration and status registers of Ethernet PHY devices. In the example illustrated, the Management Bus Interface of the Ethernet MAC is able to access the configuration and status registers of two PHY devices using 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 10-14.

Table 10-14: **Abbreviations and Terms**

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

Write Transaction

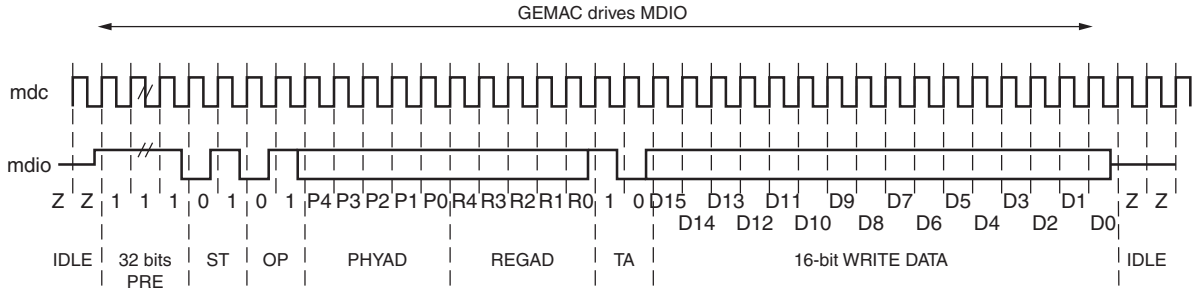

Figure 10-5: **MDIO Write Transaction**

Figure 10-5 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

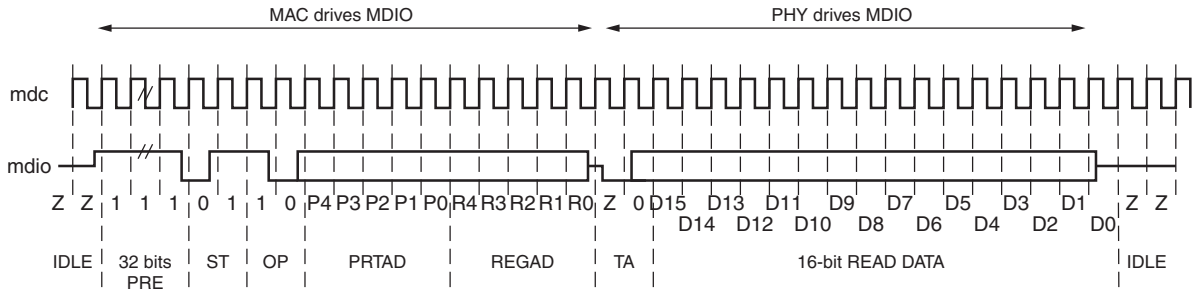

Figure 10-6: **MDIO Read Transaction**

Figure 10-6 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 details of the register map of PHY layer devices and a fuller description of the operation of the MDIO interface itself, see [Ref 10].

Connecting the TEMAC to an MDIO bus

Connecting the MDIO to an Internally Integrated PHY

The MDIO ports of the core (see Table 5-13) can be connected to the MDIO ports of an internally integrated physical layer device. For example, the MDIO ports of the Ethernet 1000BASE-X PCS/PMA or SGMII from Xilinx (see Chapter 15, Interfacing to Other Xilinx Ethernet Cores).

Connecting the MDIO to an External PHY

When the core is used to connect to an external PHY device using GMII/MII or RGMII, it is expected that the MDIO of the core is also connected to the external PHY. This allows the configuration registers of the PHY to be accessed through the Management interface of the core.

In this situation, `mdio_i`, `mdio_o` and `mdio_t` must be connected to a 3-state buffer to create a bidirectional wire, `mdio`. This 3-state buffer can be either external to the FPGA, or internally integrated by using an IOB IOBUF component with an appropriate SelectIO™ interface standard for the external PHY. (This is illustrated in Figure 10-7.)

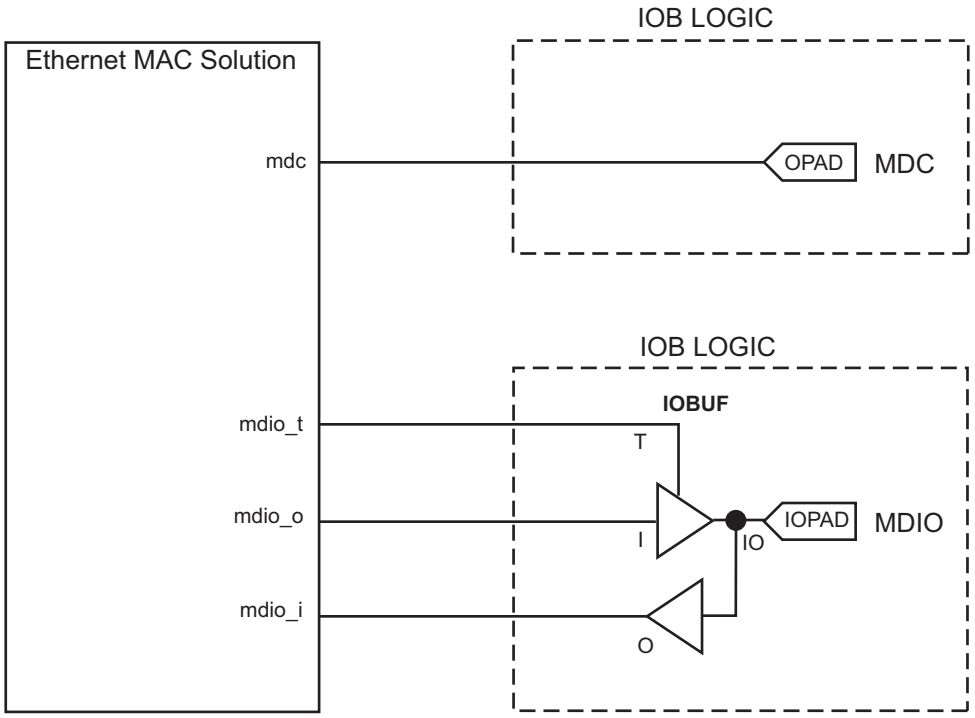

Figure 10-7: **External MDIO Interface**

Connecting the MDIO to an External and Internal PHY

The MDIO can connect to more than one device. If an internal PHY is present, for example the Ethernet 1000BASE-X PCS/PMA or SGMII LogiCORE™, performing as SGMII **and** it is desired to also connect the MDIO to an external PHY device, then an arbitration circuit is required. An example circuit is shown in Figure 10-8. Both PHY devices must be assigned a unique none zero physical address (PHYAD). This description is included only for completeness and this particular use case is not expected to be common.

The SGMII specification contains a method of transferring PHY configuration information across the SGMII link. Therefore all relevant PHY information can be obtained directly from the internal SGMII core.

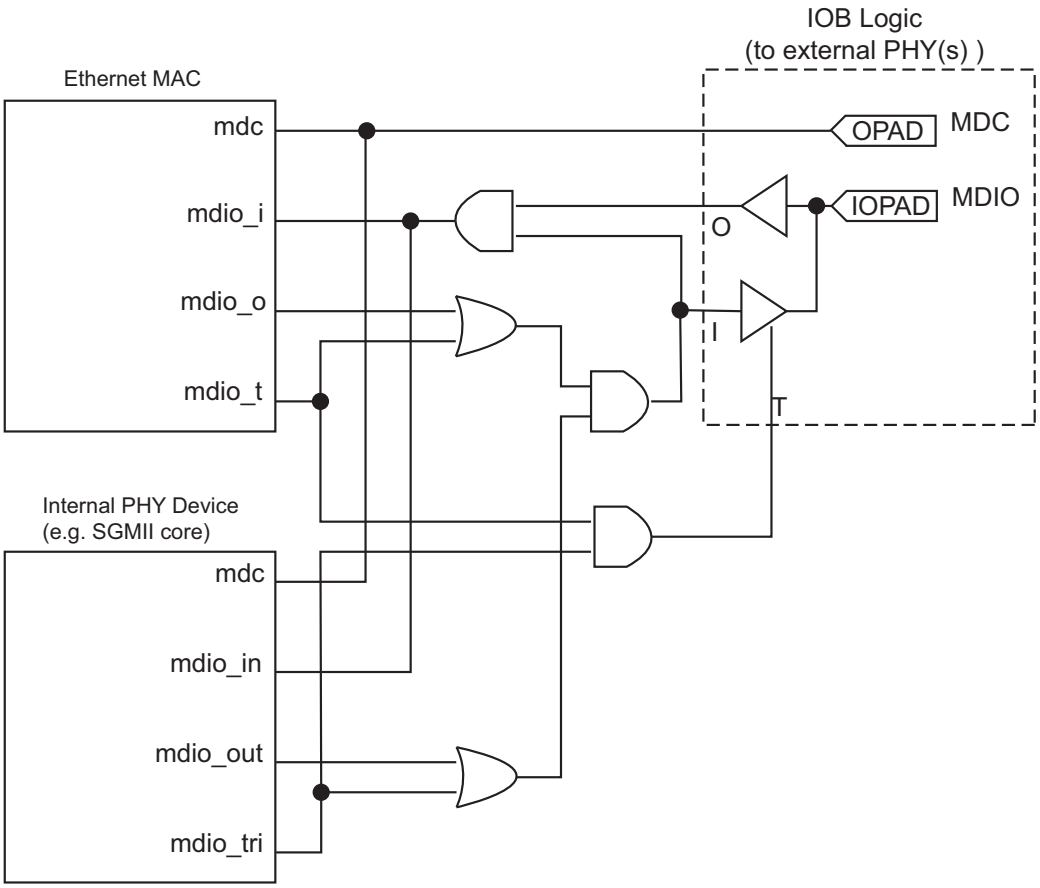

Figure 10-8: **Internal and External MDIO Interfaces**

Accessing PHY Configuration Registers, through MDIO using the Management Interface

The Management Interface is also used to access the MDIO interface of the core. The MDIO interface supplies a clock to the connected PHY, `mdc`. 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 Equation 10-1.

$$f_{MDC} = \frac{f_{s\_axi\_aclk}}{(1 + \texttt{Clock Divide[4:0]}) \times 2} \qquad \textit{Equation 10-1}$$

The frequency of `mdc` given by Equation 10-1 should not exceed 2.5 MHz to comply with the *IEEE 802.3-2008* specification 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.

For details of the register map of PHY layer devices and a fuller description of the operation of the MDIO interface itself, see *IEEE 802.3-2008*.

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 10-18, The MDIO write transaction is then initiated by a write to the MDIO Configuration word 1, shown in Table 10-17, 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. 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 10-17, 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. 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 10-19.

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 10-15.

Table 10-15: **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 10-16 through Table 10-19.

Table 10-16: **MDIO Configuration word 0 (0x500)**

Bits	Default Value	Type	Description
5-0	0x0	RW	**Clock Divide[5:0]:** See Accessing PHY Configuration Registers, through MDIO using the Management Interface, page 90
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 10-17: **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 10-17: **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.
23-21	N/A	RO	Reserved
28-24	0x0	RW	**TX_PHYAD:** This controls the PHY address being accessed.
31-29	N/A	RO	Reserved

Table 10-18: **MDIO Write Data (0x508)**

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

Table 10-19: **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

Interrupt Controller

An Interrupt Block is implemented in the Tri-Mode Ethernet MAC solution to assert an interrupt when a pending MDIO transaction has completed. Interrupt registers are shown in Table 10-20.

Table 10-20: **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. AVB Interrupts are handled through the dedicated AVB interrupt registers.

Frame Filter

The MAC can be configured with an optional Frame Filter. this is available irrespective of the use of the management interface but has much reduced functionality if no management interface is present, this usage model is described in The Configuration Vector.

The Frame Filter performs two functions:

- It checks any received packet matches of 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 comparisons, 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 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 Frame Filter contains two 512 bit registers (64 bytes):

- 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.
- Frame Filter Mask Value Register. Each bit within this register refers to the same bit number within the Frame Filter Value Register. when a bit in the Mask Value Register is set to
  - logic 1, The same bit number within the Frame Filter Value Register is compared with the respective bit in the received frame and must match if the overall Frame Filter is to obtain a match.
  - logic 0, the same bit number within the Frame Filter Value Register is not compared. This effectively turns the respective bit in the Frame Filter Value Register into a don't care bit: the overall Frame Filter is capable of obtaining a match even if this bit does not match.

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

The user can specify up to 8 frame filters in the MAC netlist, with an additional three being used if the AVB Endpoint is present. 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 1 or more 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, this does not include a AVB specific filters. See Using the Frame Filter for more detail.

Table 10-21 shows the available frame filter configuration registers.

Table 10-21: **Frame Filter Configuration**

Address (Hex)	Description
0x700	Unicast Address word 0
0x704	Unicast Address word 1
0x708	Frame filter Control

Table 10-21: **Frame Filter Configuration *(Cont'd)***

Address (Hex)	Description
0x70C	Frame filter Enable
0x710	Frame filter value bytes 3-0
0x714	Frame Filter value bytes 7-4
0x718	Frame Filter value bytes 11-8
0x71C	Frame Filter value bytes 15-12
0x720	Frame Filter value bytes 19-16
0x724	Frame Filter value bytes 23-20
0x728	Frame Filter value bytes 27-24
0x72C	Frame Filter value bytes 31-28
0x730	Frame Filter value bytes 35-32
0x734	Frame Filter value bytes 39-36
0x738	Frame Filter value bytes 43-40
0x73C	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
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 10-22 through Table 10-27.

Table 10-22: **Unicast Address (Word 0) (0x700)**

Bits	Default Value	Type	Description
31-0	unicast_address[31-0]	RW	**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 10-23: **Unicast Address (Word 1) (0x704)**

Bits	Default Value	Type	Description
15-0	unicast_address[47:32]	RW	**Frame filter unicast address[47:32]**: See description in Table 10-22.
31-16	N/A	RO	Reserved

Table 10-24: **Frame Filter Control (0x708)**

Bits	Default Value	Type	Description
31	1	RW	**Promiscuous Mode:** If this bit is set to '1,' the Frame filter is set to operate in promiscuous mode. All frames are passed to the receiver client regardless of the destination address.
30-9	N/A	RO	Reserved
8	0	RW	**AVB Select**: If the AVB Endpoint is present this is used to indicate that the filter to be selected is one of the three dedicated filters.
7-3	N/A	RO	Reserved
2-0	0	RW	**Filter Index:** All Frame filters are mapped to the same location with the filter index specifying which physical filter is to be accessed.

Table 10-25: **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 Frame Filter pointed to by the Filter index. If clear, the filter passes all packets.

Table 10-26: **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 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 frame filters are configured to match against the broadcast address.

Table 10-27: **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 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 Frame Filter

The Configurable Frame filters can be used to perform simple Destination Address filtering, Multicast Group matching, Source Address matching and 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 Frame Filter Control Register, shown in Table 10-24, allows the user to enable or disable the 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 Frame Filters are marked good.

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

By default all non-AVB Frame filters are configured with all '1's in the bottom 48 bits in both the Frame Filter Value registers and the 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. After the Frame Filter Value and Mask Value have been updated to the desired values the Filter should be enabled.

In Figure 10-9 the reception of an error free frame which 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 5-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 would mean the frame is being serviced by a dedicated output and should therefore be dropped by the else output. This is shown in Figure 10-10 where a good frame has matched against a pre-defined Destination Address filter but failed to match any of the configurable filters.

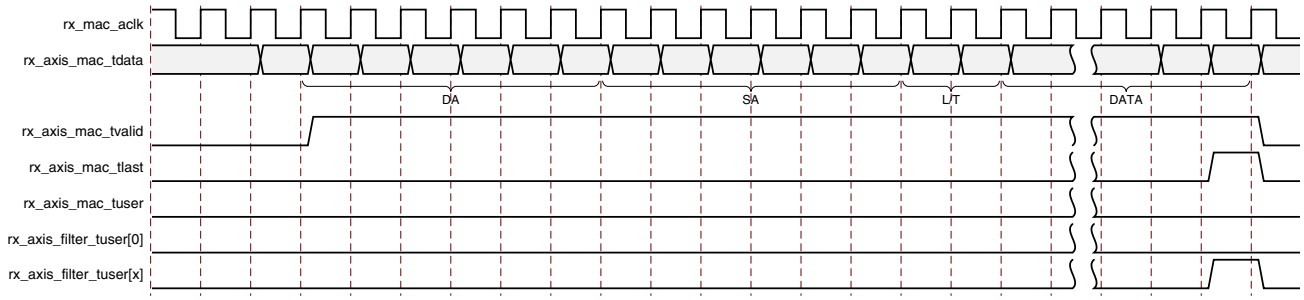

Figure 10-9: **Received Frame with a Match on Filter 0**

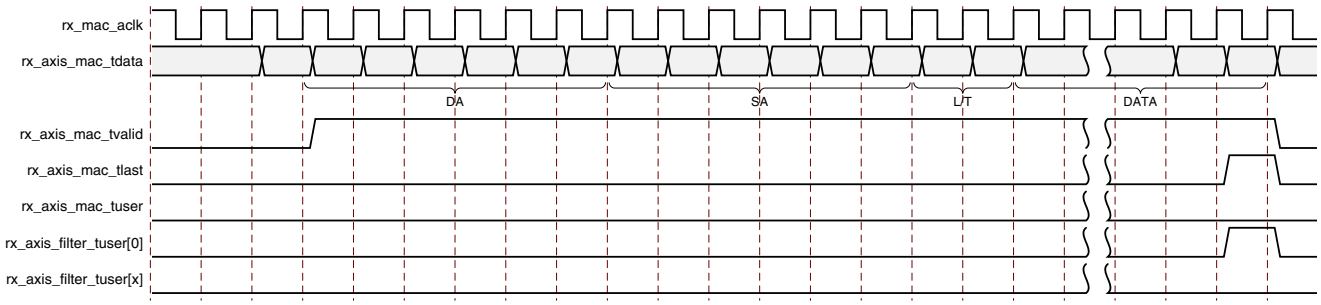

Figure 10-10: **Received Frame with No Match on Configurable Frame Filters**

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

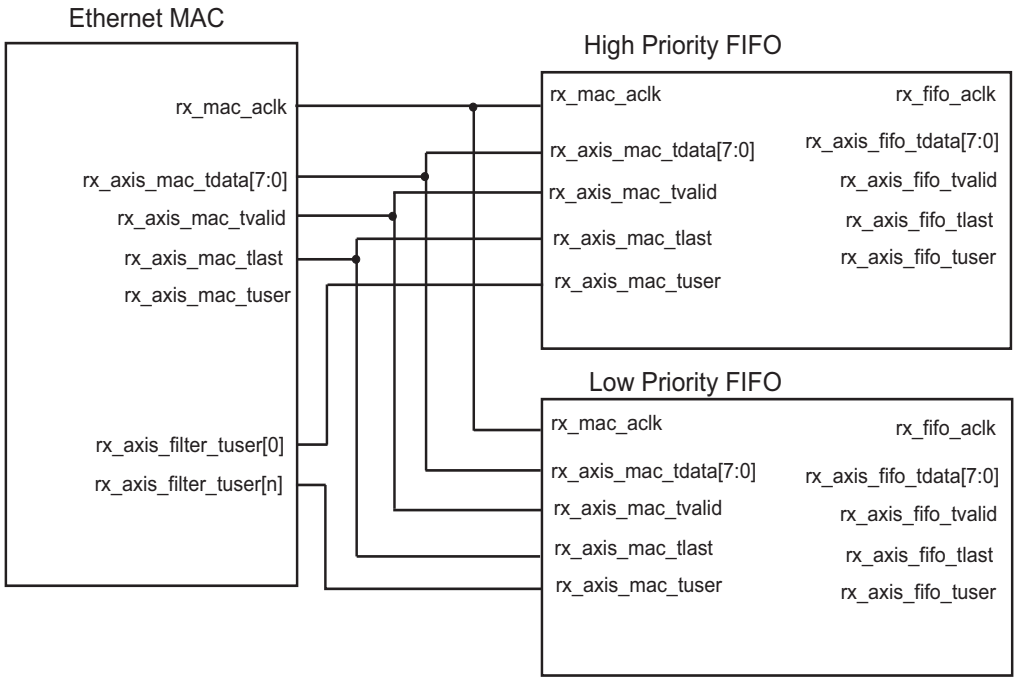

Figure 10-11: **Priority FIFO Connections**

In Figure 10-11 the MAC is shown connected to priority FIFOs. In this case 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 these register settings:

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

In this case the FIFO, which is using `rx_axis_filter_tuser`, is only passing good VLAN frames which 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. This is illustrated in Figure 10-12.

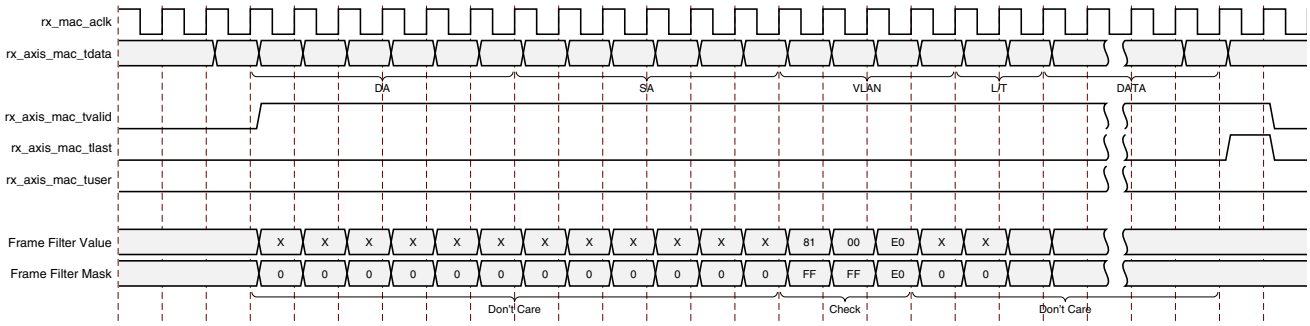

Figure 10-12: **Filtering of VLAN Frames with VLAN priority of 7**

Using the AVB Specific Frame Filters

The three AVB Frame filters are initialized to identify PTP frames and AV frames of both priorities. These initial values can be adjusted to changes the VLAN field values used by the software drivers as required.

The Frame Filter Control Register, shown in Table 10-24, allows the user to access each of the AVB Frame filters by setting bit 8: AVB Select. When set, bits 1:0 are used to specify the required AVB frame filter with the value of 3 being ignored.

PTP Frame Filter

The PTP Frame Filter is at AVB index 0 and is initialized to match the following:

Destination Address = 0x0E0000C28001
Type = 0xf788

This translates to the following register settings:

Table 10-28: **PTP Frame Filter Value (0x710-0x74C)**

Address	Default Value	Description
0x710	0x00C28001	First four bytes of the AVB Special address
0x714	0x00000E00	Final two bytes of the AVB Special Address and first two bytes of the source address
0x718	0x0	Final four bytes of the source address
0x71C	0x0000f788	Type field
0x720-0x74C	0x0	Not Used

Table 10-29: **PTP Frame Filter Mask Value (0x750-0x790)**

Address	Default Value	Description
0x750	0xffffffff	Match against the Destination address
0x754	0x0000ffff	Match against the destination address but not the source address.
0x758	0x0	Do not match against source address.
0x75C	0x0000ffff	Match against the Type field
0x760-0x790	0x0	Not Used

SR Classes A and B Frame Filters

Two frame filters are provided to identify frames belonging to either SR Class A or SR Class B. The SR Class A frame filter can be accessed by setting Filter Index to '1' in the Frame Filter Control register (see Table 10-24) and the SR Class B frame filter can be accessed by setting Filter Index to '2'.

The output of the two filters is combined so that a match against either filter allows AV traffic to pass. If only one SR Class is supported then either disable the undesired filter or set both filters to the same value.

The default behavior of the SR Class frame filters is to match both the default VLAN PCP and VLAN ID values for that SR Class, as follows:

SR Class A:

Type = 0x0081 (VLAN)

Type_info = 0x0260 (VLAN PCP= 3 VLAN ID= 2)

SR Class B:

Type = 0x0081 (VLAN)

Type_info = 0x0240 (VLAN PCP= 2VLAN ID= 2)

This translates to the following register settings:

Table 10-30: **SR Class A (index 1) Frame Filter Value (0x710-0x74C)**

Address	Default Value	Description
0x710-0x718	0x0	Not Used
0x71C	0x02600081	Match against VLAN field with PCP field set to 3 and ID field set to 2
0x720-0x74C	0x0	Not Used

Table 10-31: **SR Class A (index 1) Frame Filter Mask Value (0x750-0x790)**

Address	Default Value	Description
0x750-758	0x0	Not Used
0x75C	0xffffffff	Match against Type field and type info field
0x760-0x790	0x0	Not Used

Table 10-32: **SR Class B (index 2) Frame Filter Value (0x710-0x74C)**

Address	Default Value	Description
0x710-0x718	0x0	Not Used
0x71C	0x02400081	Match against VLAN field with PCP field set to 2 and ID field set to 2
0x720-0x74C	0x0	Not Used

Table 10-33: **SR CLass B (index 2) Frame Filter Mask Value (0x750-0x790)**

Address	Default Value	Description
0x750-758	0x0	Not Used
0x75C	0xffffffff	Match against Type field and type info field
0x760-0x790	0x0	Not Used

RX PTP Packet Buffer Address Space

The RX PTP Packet buffers are only available when the AVB functionality is included in the TEMAC core. The Address space of the RX PTP Packet Buffer is 4 Kbytes in total. This represents the size of a single FPGA block RAM pair (4 Kbytes). Every byte of this block RAM can be read.

This address space is divided equally into 16 separate buffers of 256 bytes, each of which is capable of storing a unique PTP frame. When received, a PTP frame is written into one of these buffers; then the buffer write pointer increments and points to the next buffer in preparation for subsequent PTP frame reception.

Within each buffer, the entire PTP frame is written in (from MAC Destination Address through to the last byte from the data field), starting at the base address of that buffer. Following PTP frame reception, the RX timestamp captured for that frame is written into the top 4 bytes of the buffer used. A list of the RX PTP Buffers is shown in Table 10-15.

Table 10-34: **RX PTP Buffers**

Address (Hex)	Description
0x10000-0x100FC	RX PTP Buffer 0
0x10100-0x101FC	RX PTP Buffer 1
0x10200-0x102FC	RX PTP Buffer 2
0x10300-0x103FC	RX PTP Buffer 3
0x10400-0x104FC	RX PTP Buffer 4
0x10500-0x105FC	RX PTP Buffer 5
0x10600-0x106FC	RX PTP Buffer 6
0x10700-0x107FC	RX PTP Buffer 7
0x10800-0x108FC	RX PTP Buffer 8
0x10900-0x109FC	RX PTP Buffer 9
0x10A00-0x10AFC	RX PTP Buffer 10
0x10B00-0x10BFC	RX PTP Buffer 11
0x10C00-0x10CFC	RX PTP Buffer 12
0x10D00-0x10DFC	RX PTP Buffer 13
0x10E00-0x10EFC	RX PTP Buffer 14
0x10F00-0x10FFC	RX PTP Buffer 15

TX PTP Packet Buffer Address Space

The TX PTP Packet buffers are only available when the AVB functionality is included in the TEMAC core.

The Address space of the TX PTP Packet Buffer is 2K in total, representing the size of a single FPGA block 18K RAM. Every byte of this block RAM is accessible by the CPU. This address space is divided equally into 8 separate buffers of 256 bytes, each of which is

capable of storing a unique PTP frame: 7 of these buffer locations are pre-initialized with standard PTP frame syntax; however, each byte can be modified if desired.

Within each single buffer, the initial byte is used as a length field, used to indicate to the core logic the number of bytes to be transmitted for that frame. An entire PTP frame (from MAC Destination Address through to the last byte from the data field) is then stored, starting at the 8th address of that particular buffer. Following PTP frame transmission from one of these buffers, the TX Timestamp captured for that frame is written into the top 4 bytes of the buffer just used. See TX PTP Packet Buffer for more details. A list of the TX PTP Buffers is shown in Table 10-35.

Table 10-35: **TX PTP Buffers**

Address (Hex)	Description
0x11000-0x110FC	TX PTP Buffer 0 - Initialized for a SYNC frame.
0x11100-0x111FC	TX PTP Buffer 1 - Initialized for a Follow up frame.
0x11200-0x112FC	TX PTP Buffer 2 - Initialized for a Pdelay request frame.
0x11300-0x113FC	TX PTP Buffer 3 - Initialized for a Pdelay response frame.
0x11400-0x114FC	TX PTP Buffer 4 - Initialized for a Pdelay response follow up frame.
0x11500-0x115FC	TX PTP Buffer 5 - Initialized for an Announce frame.
0x11600-0x116FC	TX PTP Buffer 6 - Initialized for a Signaling frame.
0x11700-0x117FC	TX PTP Buffer 7

AVB Configuration

The AVB configuration registers are only present when the AVB is included in the TEMAC core. These registers are used by the software drivers to control the AVB functionality.

A list of the AVB Configuration registers is shown in Table 10-15.

Table 10-36: **AVB Configuration Registers**

Address (Hex)	Description
0x12000	TX PTP Packet Buffer Control Register
0x12004	RX PTP Packet Control Register
0x12008	Reserved
0x1200C	TX Arbiter Send Slope control Register
0x12010	TX Arbiter Idle Slope control Register
0x12014-0x127FC	Reserved

TX PTP Packet Control Register

Table 10-37 defines associated control register of the TX PTP Packet Buffers, used by the software to request the transmission of the PTP frames.

Table 10-37: **TX PTP Packet Buffer Control Register (0x12000)**

Bits	Default Value	Type	Description
7-0	0	WO	**tx_send_frame Bits**. The TX PTP Packet Buffer is split into 8 regions of 256 bytes, each of which can contain a separate PTP frame. There is 1 tx_send_frame bit for each of the 8 regions. Each bit, when written to '1', causes a request to be made to the TX Arbiter. When access is granted the frame contained within the respected region is transmitted. If read, always returns 0.
15-8	0	RO	**tx_frame_waiting Indication**. The TX PTP Packet Buffer is split into 8 regions of 256 bytes, each of which can contain a separate PTP frame. There is 1 tx_frame_waiting bit for each of the 8 regions. Each bit, when logic 1, indicates that a request has been made for frame transmission to the TX Arbiter, but that a grant has not yet occurred. When the frame has been successfully transmitted, the bit is set to logic 0. This bit allows the microprocessor to run off a polling implementation as opposed to the Interrupts.
18-16	0	RO	**tx_packet**. Indicates the number (block RAM bin position) of the most recently transmitted PTP packet.
31-19	0	RO	Unused

Note: A read or a write to this register clears the interrupt_ptp_tx interrupt (asserted after each successful PTP packet transmission).

RX PTP Packet Control Register

Table 10-38 defines the associated control register of the RX PTP Packet Buffers used by the software to monitor the position of the most recently received PTP frame.

Table 10-38: **RX PTP Packet Buffer Control Register (0x12004)**

Bits	Default Value	Type	Description
0	0	WO	**rx_clear**. When written with a '1,' forces the buffer to empty, in practice moving the write address to the same value as the read address. If read, always returns 0.
7-1	0	RO	Unused
11-8	0	RO	**rx_packet**. Indicates the number (block RAM bin position) of the most recently received PTP packet.
31-12	0	RO	Unused

Note: A read or a write to this register clears the `i`nterrupt_ptp_rx interrupt (asserted after each successful PTP packet reception).

TX Arbiter Send Slope Control Register

The SendSlope variable is defined in IEEE802.1Q to be the rate of change of credit, in bits per second, when the value of credit is decreasing (during AV packet transmission). Together with TX Arbiter Idle Slope Control Register, RTC Nanoseconds Field Offset Control and RTC Seconds Field Offset Control these registers define the maximum limit of the bandwidth reserved for AV traffic, as enforced by the TX Arbiter. The default values allow the maximum bandwidth proportion of 75% for the AV traffic. See the IEEE specification for further information.

Table 10-39: **TX Arbiter Send Slope Control Register (0x1200C)**

Bits	Default Value	Type	Description
31-20	0	RO	Unused
19-0	2048	R/W	The value of sendSlope

TX Arbiter Idle Slope Control Register

The `idleSlope` variable is defined in *IEEE802.1Q* to be the rate of change of credit, in bits per second, when the value of credit is increasing (whenever there is no AV packet transmission). Together with TX Arbiter Send Slope Control Register, RTC Nanoseconds Field Offset Control and RTC Seconds Field Offset Control these registers define the maximum limit of the bandwidth reserved for AV traffic: this is enforced by the TX Arbiter. The default values allow the maximum bandwidth proportion of 75% for the AV traffic. See the IEEE specification for further information.

Table 10-40: **TX Arbiter Idle Slope Control Register (0x12010)**

Bits	Default Value	Type	Description
31-20	0	RO	Unused
19-0	6144	R/W	The value of idleSlope

RTC Configuration

The RTC configuration registers are only present when the AVB is included in the TEMAC core. These registers are used by the software drivers to control the RTC functionality.

A list of the RTC Configuration registers is shown in Table 10-15.

Table 10-41: **RTC Configuration Registers**

Address (Hex)	Description
0x12800	RTC Nanoseconds Field Offset
0x12804	Reserved
0x12808	RTC Seconds Field Offset [31:0]
0x1280C	RTC Seconds Field Offset [47:32]
0x12810	RTC Increment Value Control Register
0x12814	Current RTC Nanoseconds Value
0x12818	Current RTC Seconds Value Bits [31:0]
0x1281C	Current RTC Seconds Value Bits [47:32]
0x12820	RTC Interrupt Clear Register
0x12824	RTC Phase Adjustment Register
0x12828-0x13FFC	Reserved

RTC Nanoseconds Field Offset Control

Table 10-42 describes the offset control register for the nanoseconds field of the RTC used to force step changes into the counter. When in PTP clock master mode, this can be used to set the initial value following power-up. When in PTP clock slave mode, the software drivers use this register to implement the periodic step corrections.

This register and the registers defined in Table 10-43 and in Table 10-44 are linked. These three offset values are loaded into the RTC counter logic simultaneously following a write to this nanosecond offset register.

Table 10-42: **RTC Nanoseconds Field Offset (0x12800)**

Bits	Default Value	Type	Description
29-0	0	R/W	**30-bit offset value for the RTC nano seconds**. Used by the microprocessor to initialize the RTC, then afterwards to perform the regular RTC corrections (when in slave mode).
31-30	0	RO	Unused

RTC Seconds Field Offset Control

Table 10-43 describes the offset control register for the lower 32-bits of seconds field of the RTC, used to force step changes into the counter. When in PTP clock master mode, this can be used to set the initial value following power-up. When in PTP clock slave mode, the software drivers use this register to implement the periodic step corrections.

This register and the registers defined in Table 10-42 and in Table 10-44 are linked. These three offset values are loaded into the RTC counter logic simultaneously following a write to the nanosecond offset register defined in Table 10-42.

Table 10-43: **Seconds Field Offset Bits [31:0] (0x12808)**

Bits	Default Value	Type	Description
31-0	0	R/W	**32-bit offset value for the RTC seconds field (bits 31-0)**. Used by the microprocessor to initialize the RTC, then afterwards to perform the regular RTC corrections (when in slave mode).

Table 10-44 describes the offset control register for the upper 16-bits of seconds field of the RTC, used to force step changes into the counter. When in PTP clock master mode, this can be used to set the initial value following power-up. When in PTP clock slave mode, the software drivers use this register to implement the periodic step corrections.

This register and the registers defined in Table 10-42 and in Table 10-43 are linked. These three offset values are loaded into the RTC counter logic simultaneously following a write to the nanosecond offset register defined in Table 10-42.

Table 10-44: **Seconds Field Offset Bits [47:32] (0x1280C)**

Bits	Default Value	Type	Description
15-0	0	R/W	**16-bit offset value for the RTC seconds field (bits 47-32)**. Used by the microprocessor to initialize the RTC, then afterwards to perform the regular RTC corrections (when in slave mode).
31-16	0	RO	Unused

RTC Increment Value Control Register

Table 10-45 describes the RTC Increment Value Control Register, which provides a configurable increment rate for the RTC counter. This increment register should take the value of the clock period being used to increment the RTC; however, the resolution of this increment register is very fine (in units of 1/1048576 ($1/2^{20}$) fraction of one nanosecond) and for this reason the RTC increment rate can be adjusted to a very fine degree of accuracy, providing the following features:

- The RTC can be incremented from any available clock frequency that is greater than the IEEE802.1AS defined minimum of 25 MHz.
- When acting as a clock slave, the rate adjustment of the RTC can be matched to that of the network clock master to an exceptional level of accuracy.

Table 10-45: **RTC Increment Value Control Register (0x12810)**

Bits	Default Value	Type	Description
25-0	0	R/W	Per rtc_clk clock period Increment Value for the RTC.
31-26	0	RO	Unused

Current RTC Value Registers

Table 10-46 describes the nanoseconds field value register for the nanoseconds field of the RTV. When read, this returns the latest value of the counter. This register and the registers defined in Table 10-47 and in Table 10-48 are linked. When this nanoseconds value register is read, the entire RTC (including the seconds field) is sampled.

Table 10-46: **Current RTC Nanoseconds Value (0x12814)**

Bits	Default Value	Type	Description
29-0	0	RO	Current Value of the synchronized RTC nanoseconds field. Note: A read from this register samples the entire RTC counter (synchronized) so that the Epoch and Seconds field are held static for a subsequent read.
31-30	0	RO	Unused

Table 10-47 describes the lower 32-bits of the seconds value register for the seconds field of the RTC. When read, this returns the latest value of the counter. This register and the registers defined in Table 10-46 and in Table 10-48 are linked. When the nanoseconds value register is read (see Table 10-46), the entire RTC is sampled.

Table 10-47: **Current RTC Seconds Field Value bits [31:0] (0x12818)**

Bits	Default Value	Type	Description
31-0	0	RO	Sampled Value of the synchronized RTC Seconds field (bits 31-0).

Table 10-48 describes the upper 16-bits of the seconds value register for the seconds field of the RTC. When read, this returns the latest value of the counter. This register and the registers defined in Table 10-46 and in Table 10-47 are linked. When the nanoseconds value register is read (see Table 10-46), the entire RTC is sampled.

Table 10-48: **Current RTC Seconds Field Value Bits [47:32] (0x1281C)**

Bits	Default Value	Type	Description
15-0	0	RO	Sampled Value of the synchronized RTC Seconds field (bits 47-32).
32-16	0	RO	Unused

RTC Interrupt Clear Register

Table 10-49 describes the control register defined for the `interrupt_ptp_timer` signal, the periodic interrupt signal which is raised by the RTC.

Table 10-49: **RTC Interrupt Clear Register (0x12820)**

Bits	Default Value	Type	Description
0	0	WO	Write ANY value to bit 0 of this register to clear the interrupt_ptp_timer Interrupt signal. This bit always returns 0 on read.
31-1	0	RO	Unused

Phase Adjustment Register

Table 10-50 describes the Phase Adjustment Register which has units of nanoseconds. This value is added to the synchronized value of the RTC nanoseconds field, and the RTC timing signals are then derived from the result. This phase offset is therefore applied to the `clk8k` signal.

As an example, writing the value of the decimal 62500 (half of an 8 kHz clock period) to this register would invert the `clk8k` signal with respect to a value of 0. For this reason, this register can provide fine grained phase alignment of these signals to a 1 ns resolution.

Table 10-50: **RTC Phase Adjustment Register (0x12824)**

Bits	Default Value	Type	Description
29-0	0	R/W	ns value relating to the phase offset for all RTC derived timing signals (clk8k).
31-30	0	RO	Unused

The Configuration Vector

If the Optional Management interface is omitted from the core, all of the relevant configuration signals are brought out of the core. These signals are bundled into the `rx_configuration_vector` and the `tx_configuration_vector` signals. The bit mapping of these signals are defined in Table 10-51 and Table 10-52.

You can permanently set the vector bits to logic 0 or 1 or change the configuration vector signals at any time; however, with the exception of the reset signals, they do not take effect until the current frame has completed transmission or reception.

Table 10-51: **tx_configuration_vector Bit Definitions**

Bit(s)	Description
0	**Transmitter Reset.** When this bit is '1', the MAC transmitter is held in reset. This signal is an input to the reset circuit for the transmitter block.
1	**Transmitter Enable.** When this bit is set to '1', the transmitter is operational. When set to '0', the transmitter is disabled.
2	**Transmitter VLAN Enable.** When this bit is set to '1', the transmitter allows the transmission of VLAN tagged frames up to 1522 bytes in size.
3	**Transmitter In-Band FCS Enable.** When this bit is '1', the MAC transmitter expects the FCS field and any padding to take the frame up to 64 bytes to be passed in by the user as described in User-Supplied FCS Passing, page 49. When it is '0', the MAC transmitter appends padding as required, compute the FCS and append it to the frame.
4	**Transmitter Jumbo Frame Enable.** When this bit is '1', the MAC transmitter allows frames larger than the maximum legal frame length specified in *IEEE 802.3-2008* to be sent. When set to '0', the maximum frame size is dependant upon the setting of *Transmitter Max Frame Enable* and *Transmitter Max Frame Length*.
5	**Transmitter Flow Control Enable.** When this bit is '1', asserting the pause_req signal causes the MAC core to send a flow control frame out from the transmitter as described in Transmitting a Pause Control Frame, page 57. when this bit is '0', asserting the pause_req signal has no effect.
6	**Transmitter Half Duplex** If '1,' the transmitter operates in half-duplex mode. If '0,' the transmitter operates in full-duplex mode. If the TEMAC solution has been generated without half-duplex support, this input to the core is unused.
7	Reserved
8	**Transmitter Interframe Gap Adjust Enable.** If '1,' and the MAC is set to operate in full-duplex mode, then the transmitter reads the value of the tx_ifg_delay port and set the Interframe Gap accordingly. If '0,' the transmitter always inserts at least the legal minimum interframe gap.
11:9	Reserved
13:12	**Transmitter Speed Configuration** 00 - 10 Mb/s 01 - 100 Mb/s 10 - 1 Gb/s When the TEMAC solution is generated for only 1 Gb/s speed support, these inputs are unused. When the TEMAC solution is generated for only 10 Mb/s or 100 Mb/s speed support, only bit 12 is used to differentiated the speed: bit 13 is unused. ***Caution!*** Issue the core with a system-wide reset following a speed change.
14	**Transmitter Max Frame Enable.** When this bit is set to '1' and *Transmitter Jumbo Frame Enable* is set to '0', the MAC transmitter allows frames larger than the maximum legal frame length specified in IEEE 802.3-2008 to be sent, provided they are smaller than the size specified in *Transmitter Max Frame Length*. This is described in Maximum Permitted Frame Length, page 51. When set to '0', the MAC transmitter only allows frames up to the legal maximum to be sent.
15	Reserved

Table 10-51: **tx_configuration_vector Bit Definitions *(Cont'd)***

Bit(s)	Description
31:16	**Transmitter Max Frame Size[15:0].** This specifies the maximum frame size supported when *Transmitter Max Frame Enable* is set to '1' and *Transmitter Jumbo Frame Enable* is set to '0'. This should always be set to 1518 or more.
79:32	**Transmitter Pause Frame Source Address[47:0].** This MAC Address is used by the MAC core to match against the destination address of any incoming flow control frames, and as the source address for any outbound flow control frames. The bits in this vector field are ordered so that the least significant bit of the MAC Address (IEEE802.3 definition) is stored in the least significant bit of this vector field. Consequently, bit 0 of this field differentiates between an individual or group (multicast) address. The transmission order within a MAC frame is to send the least significant bit of the MAC Address first. Consequently, bits 7-0 of this vector field represent the first byte to appear in frame transmission

Table 10-52: **rx_configuration_vector Bit Definitions**

Bit(s)	Description
0	**Receiver Reset.** When this bit is '1', the MAC receiver is held in reset. This signal is an input to the reset circuit for the receiver block.
1	**Receiver Enable.** When this bit is set to '1', the receiver is operational. When set to '0', the receiver is disabled.
2	**Receiver VLAN Enable.** When this bit is set to '1', the receiver allows the reception of VLAN tagged frames up to 1522 bytes in size.
3	**Receiver In-Band FCS Enable.** When this bit is '1', the MAC receiver pass the FCS field to the user as described in User-Supplied FCS Passing, page 45. When it is '0', the MAC receiver does not pass the FCS field. In both cases, the FCS field is verified on the frame.
4	**Receiver Jumbo Frame Enable.** When this bit is '1', the MAC receiver passes frames larger than the maximum legal frame length specified in IEEE 802.3-2008. When set to '0', the maximum frame size is dependant upon the setting of *Receiver Max Frame Enable* and *Receiver Max Frame Length*.
5	**Receiver Flow Control Enable.** When this bit is '1', received flow control frames inhibit the transmitter operation as described in Receiving a Pause Control Frame, page 58. When it is '0', received flow frames are passed up to the user.
6	**Receiver Half Duplex** If '1,' the receiver operates in half-duplex mode. If '0,' the receiver operates in full-duplex mode. If the TEMAC has been generated without half-duplex support then this input to the core is unused.
7	Reserved
8	**Receiver Length/Type Error Check Disable** When this bit is '1,' the core does not perform the length/type field error checks as described in Length/Type Field Error Checks, page 46. When it is set to '0,' the length/type field checks are performed; this is normal operation.
9	**Receiver 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.
10	Reserved
11	**Promiscuous Mode:** When this bit is set to '1,' the Frame filter is set to operate in promiscuous mode. All frames are passed to the receiver client regardless of the destination address.

Table 10-52: **rx_configuration_vector Bit Definitions *(Cont'd)***

Bit(s)	Description
13:12	**Receiver Speed Configuration** 00 - 10 Mb/s 01 - 100 Mb/s 10 - 1 Gb/s When the TEMAC solution is generated for only 1 Gb/s speed support, these inputs are unused. When the TEMAC solution is generated for only 10 Mb/s or 100 Mb/s speed support, only bit 12 is used to differentiated the speed: bit 13 is unused. ***Caution!*** Issue the core with a system-wide reset following a speed change.
14	**Receiver Max Frame Enable.** When this bit is set to '1' and *Receiver Jumbo Frame Enable* is set to '0', the MAC receiver passes frames larger than the maximum legal frame length specified in IEEE 802.3-2008, provided they are smaller than the size specified in *Receiver Max Frame Length*. This is described in Maximum Permitted Frame Length, page 46. When set to '0', the MAC receiver only passes frames up to the legal maximum.
15	Reserved
31:16	**Receiver Max Frame Size[15:0].** This specifies the maximum frame size supported when *Receiver Max Frame Enable* is set to '1' and *Receiver Jumbo Frame Enable* is set to '0'. This should always be set to 1518 or more.
79:32	**Receiver Pause Frame Source Address[47:0].** This MAC Address is used by the MAC core to match against the destination address of any incoming flow control frames, and as the source address for any outbound flow control frames. The bits in this vector field are ordered so that the least significant bit of the MAC Address (IEEE802.3 definition) is stored in the least significant bit of this vector field. Consequently, bit 0 of this field differentiates between an individual or group (multicast) address. The reception order within a MAC frame is to receive the least significant bit of the MAC Address first. Consequently, bits 7-0 of this vector field represent the first byte to appear in frame reception.

Frame Filter

When the 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 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.

TEMAC Configuration Settings

This section discusses unusual configuration options. These can be set by either method (Management Interface or the Configuration Vector).

Half-Duplex Configuration Settings

When the core is generated with half-duplex capability, 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 TEMAC allows Flow Control functionality and Duplex mode to be configured independently. However, Flow Control is enabled only 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

Under all core generation settings, the core contains a configurable Pause frame MAC Source Address (see MAC Configuration) and the use of configuration vectors allows this to be set independently for RX and TX. 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 TEMAC solution is generated with the optional frame filter, the core contains a configurable Unicast Address (see MAC Configuration). This is used by the 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.

The core, when generated without a management interface, has independent RX and TX Pause Frame MAC source Address control; these should be set to the same value.

AVB Endpoint

When enabling AVB Endpoint operation, disable flow control and jumbo mode and use only in full-duplex mode.

Note: The AVB Endpoint is only available if the AXI4-Lite management interface is included and therefore there is no configuration vector control available.

XILINX®

Chapter 11

Physical Interface for the 10 Mb/s and 100 Mb/s Only Ethernet MAC IP Core

The HDL example design supplied with the 10 Mb/s or 100 Mb/s only IP core, provides an MII interface. This is typically used to connect the MAC to an external PHY device.

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

Virtex®-6 devices support MII at 2.5 V only; Spartan®-6 devices support MII at 3.3 V or lower. For 7 Series families it depends on the type of I/O used: HR I/O supports MII at 2.5 V whereas HP I/O only supports 1.8 V or lower and therefore an external voltage converter is required to interoperate with any multi-standard PHY.

MII Transmitter Interface

The logic required to implement the MII transmitter logic is illustrated in Figure 11-1.

`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 interface of the core. Alternatively, for devices containing regional clock resources, the BUFG of Figure 11-1 can be replaced with a BUFR primitive.

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 11-1 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 11-1 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 11-1.

`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 interface of the TEMAC. Alternatively, for devices containing regional clock resources, the BUFG of Figure 11-1 can be replaced with a BUFR primitive.

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 11-1 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 11-1 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 both 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 11-1 must be duplicated for each instance of the core.

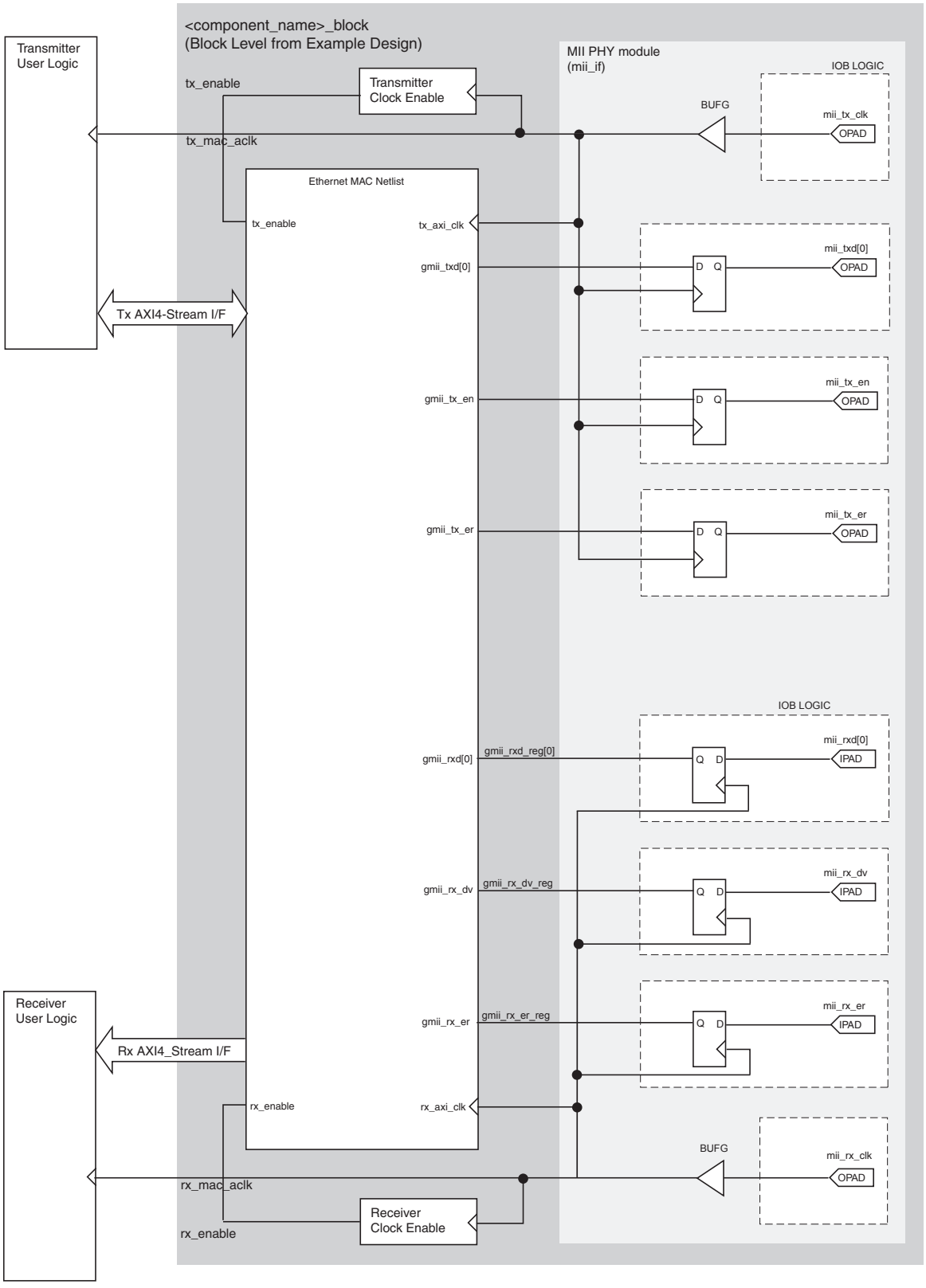

Figure 11-1: **MII Transmitter, Receiver and Clock Logic For All Devices**

XILINX®

Chapter 12

Physical Interfaces for 1 Gb/s Only Ethernet MAC IP Core

The HDL example design supplied with the core, for 1 Gb/s only operation provides either a GMII or RGMII interface. These are typically used to connect the MAC to an external PHY device.

The Gigabit Media Independent Interface (GMII), defined in IEEE Std 802.3-2008, clause 35, is used to connect a 1 Gb/s capable MAC to the physical sublayers.

Virtex®-6 devices support GMII at 2.5 V only; Spartan®-6 devices support GMII at 3.3 V or lower. For 7 Series families it depends on the type of I/O used: HR I/O support s GMII at 2.5 V whereas HP I/O only supports 1.8V or lower. and therefore an external voltage converter is required to interoperate with any multi-standard PHY for GMII.

The Reduced Gigabit Media Independent Interface (RGMII) is an alternative to the GMII and achieves a 50% reduction in the pin count compared with GMII. Therefore, this is often favored over GMII by Printed Circuit Board (PCB) designers. This configuration is achieved with the use of double-data-rate (DDR) flip-flops.

Virtex-6 devices support RGMII at 2.5 V or lower; Spartan-6 devices support RGMII at 3.3 V or lower. For 7 Series families it depends on the type of I/O used: HR I/O supports RGMII at 2.5 V whereas HP I/O only supports 1.8 V or lower. Despite this being the defined RGMII voltage most PHYs require 2.5 V and therefore an external voltage converter is required to interoperate with any multi-standard PHY for RGMII.

See the appropriate section:

- Gigabit Media Independent Interface (GMII)
- Reduced Gigabit Media Independent Interface (RGMII)

Gigabit Media Independent Interface (GMII)

GMII Transmitter Interface

The logic required to implement the GMII transmitter logic is illustrated in Figure 12-1. `gtx_clk` is a user-supplied 125 MHz reference clock source. 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 interface of the core.

Figure 12-1 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 for a Virtex-6 device when the GMII is selected. If other

families are chosen, equivalent primitives specific to that family are used in the example design.

Figure 12-1 shows that the output transmitter signals are registered in device IOBs 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.

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.

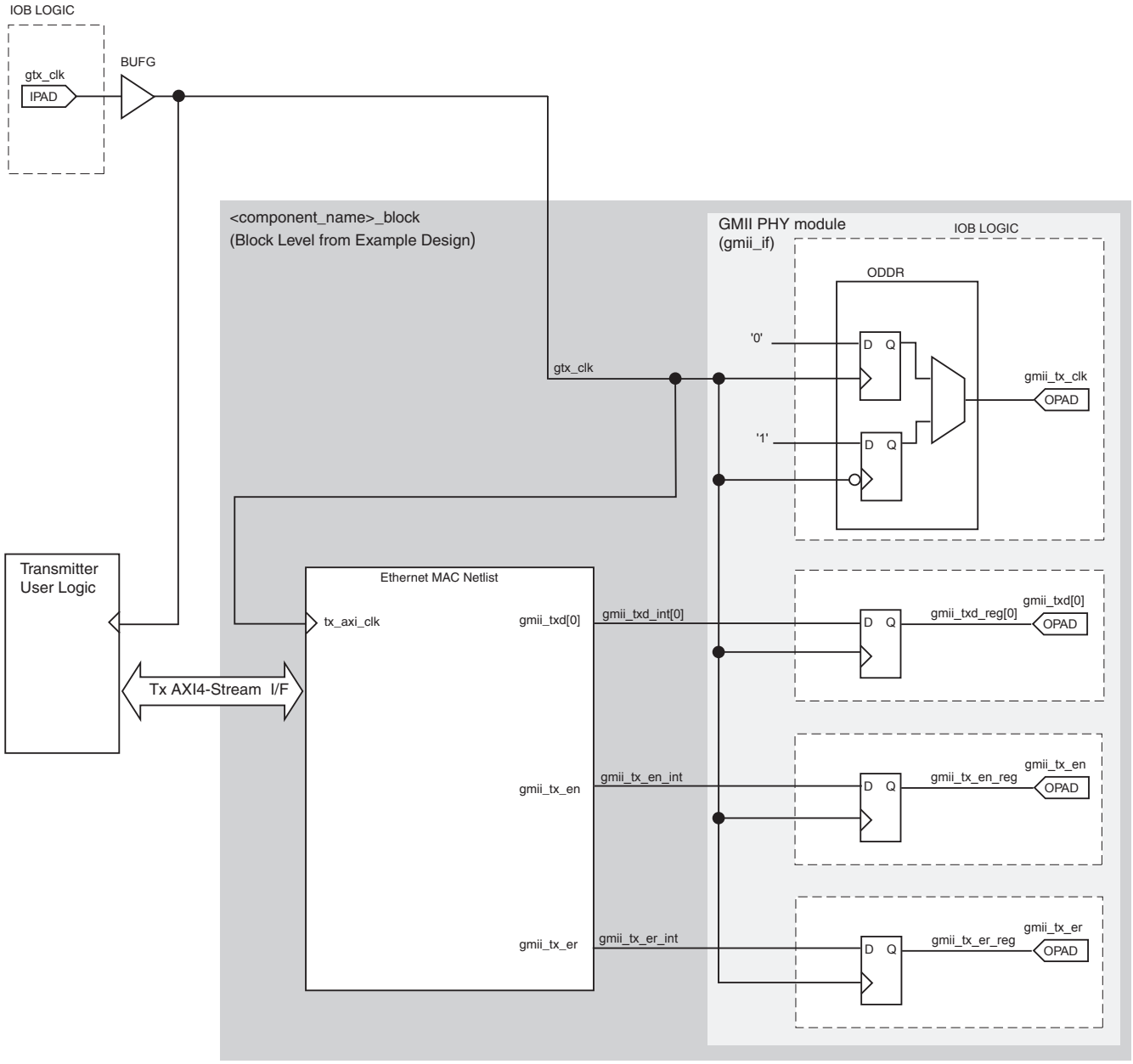

Figure 12-1: **GMII Transmitter Logic and Clock Logic**

GMII Receive Interface

The logic required to implement the GMII receiver logic is described in the following sections. Logical implementation is different for different device families. See the specific family section:

- Virtex-7, Kintex-7, Artix-7 and Virtex-6 Devices
- Spartan-6 Devices

Virtex-7, Kintex-7, Artix-7 and Virtex-6 Devices

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. 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. The relevant family *User Guide* should be consulted.

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 interface of the core. 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.

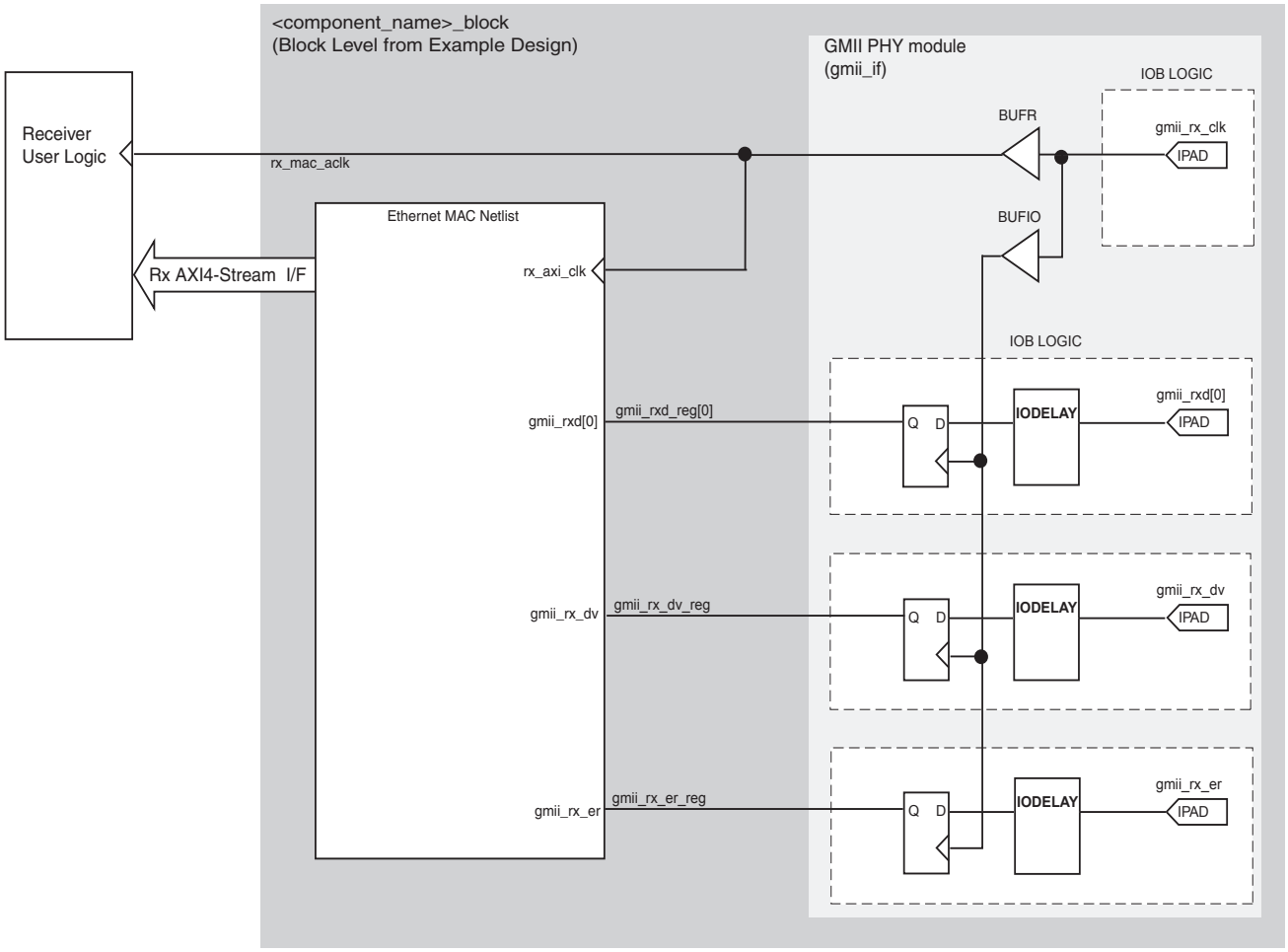

Figure 12-2: **GMII Receiver Logic and Clock Logic for 7 Series and Virtex-6 Devices**

Spartan-6 Devices

In this implementation, a BUFIO2 is used to provide the lowest form of clock routing delay from input clock to input GMII RX signal sampling at the device IOBs. This creates placement constraints: a BUFIO2 capable clock input pin must be selected, and all other input GMII RX signals must be placed in the respective BUFIO2 region. See [Ref 5].

The input clock is also placed onto global clock routing using the BUFG component as illustrated. This 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 interface of the core.

The IODELAY2 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 IODELAY2 element using constraints in the UCF; these can be edited if desired. See Chapter 14, Constraining the Core.

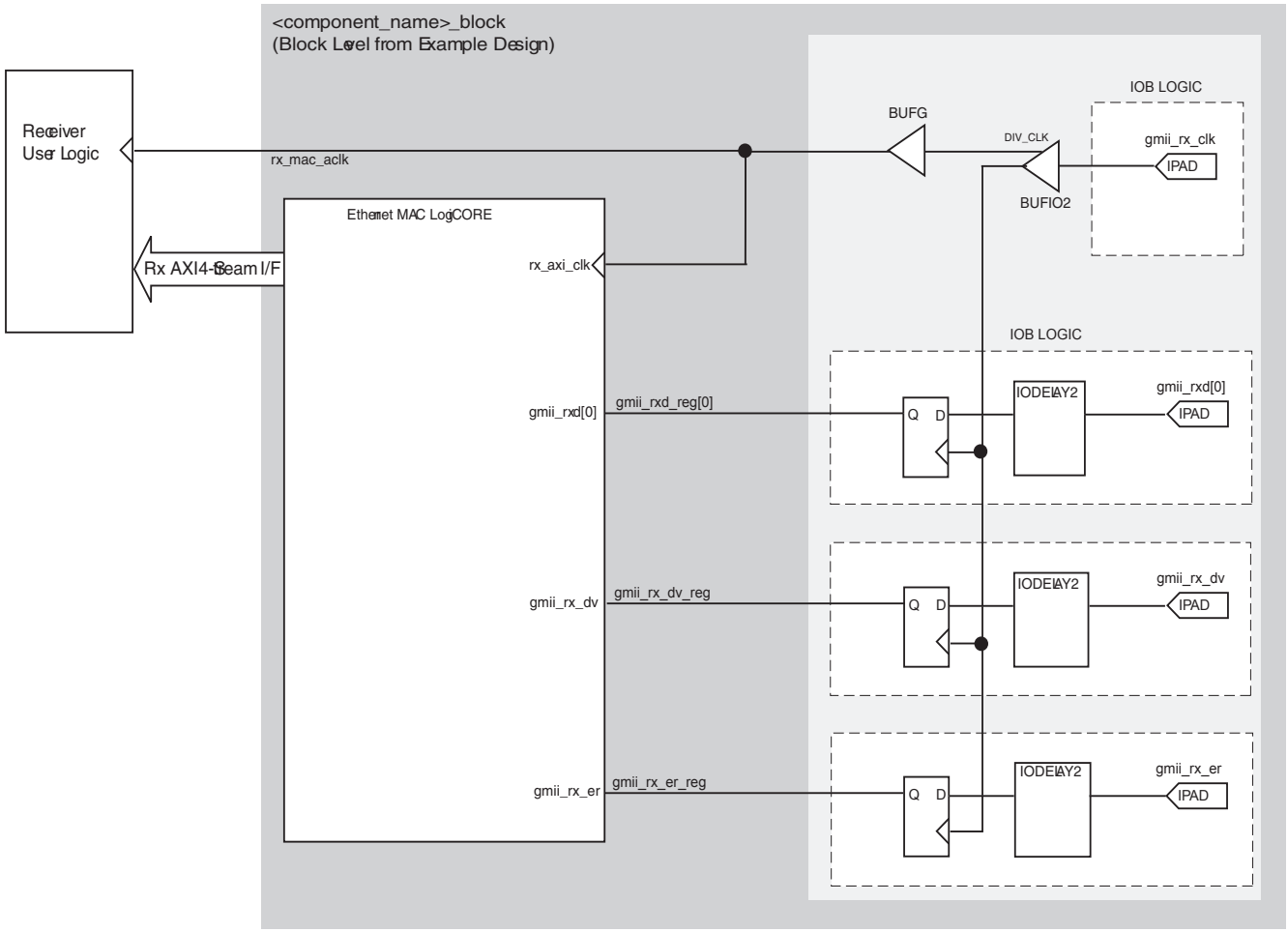

Figure 12-3: **GMII Receiver Logic and Clock Logic for Spartan-6 Devices**

Clock Sharing across Multiple Cores with GMII for 1 Gb/s Operation

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. See the appropriate section:

- Clock Resource Sharing in 7 Series and Virtex-6 Devices
- Clock Resource Sharing in Spartan-6 Devices

Clock Resource Sharing in 7 Series and Virtex-6 Devices

Figure 12-4 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.

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 12-4 illustrates 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.

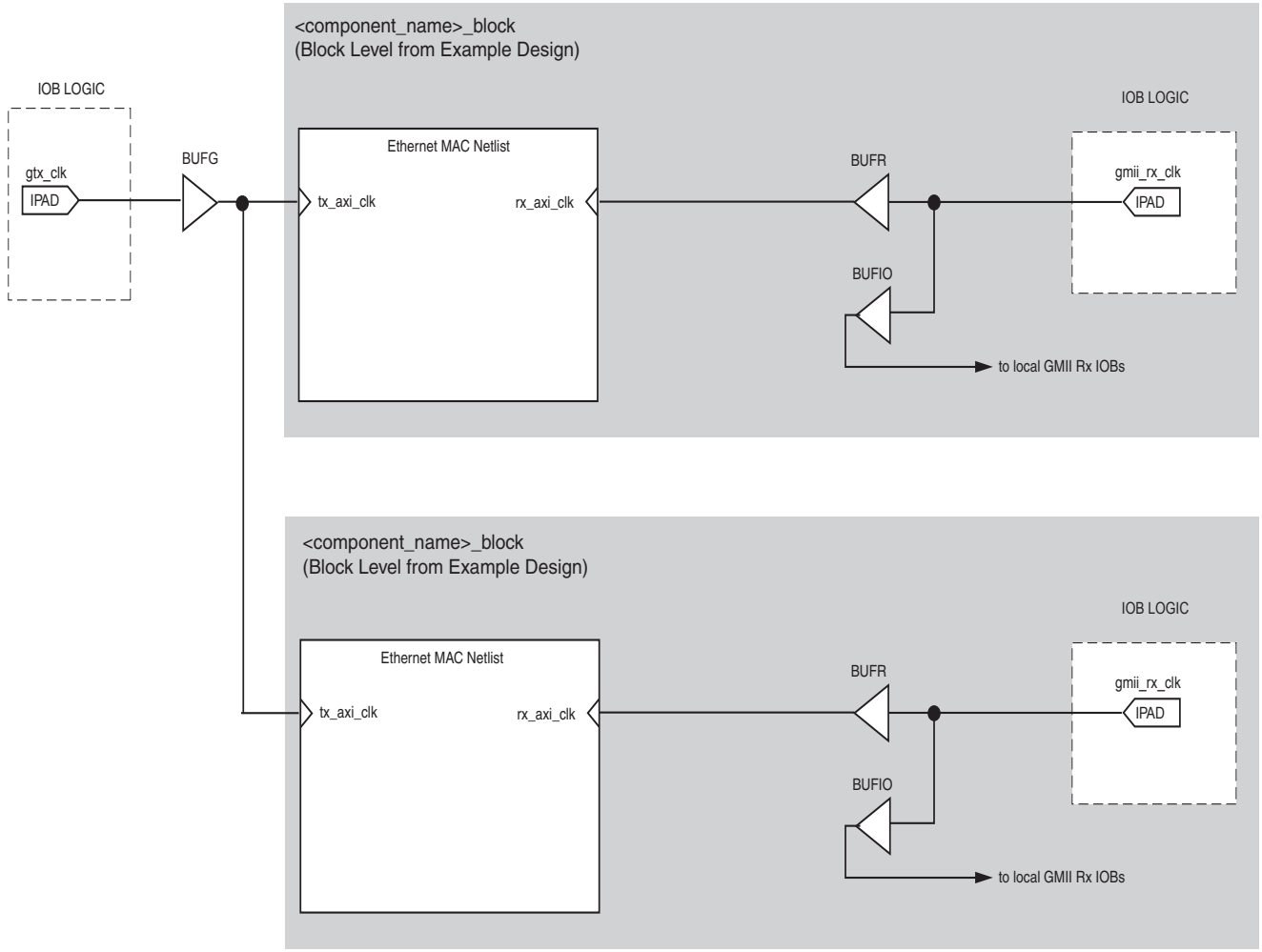

Figure 12-4: **Clock Resource Sharing for 1 Gb/s GMII in 7 Series and Virtex-6 Devices**

Clock Resource Sharing in Spartan-6 Devices

Figure 12-5 illustrates clock resource sharing across multiple instantiations of the core when using GMII at 1 Gb/s in Spartan-6 devices. For all instantiations, `gtx_clk` can be shared between multiple cores, resulting in a common clock domain across the device.

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 12-5 illustrates 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.

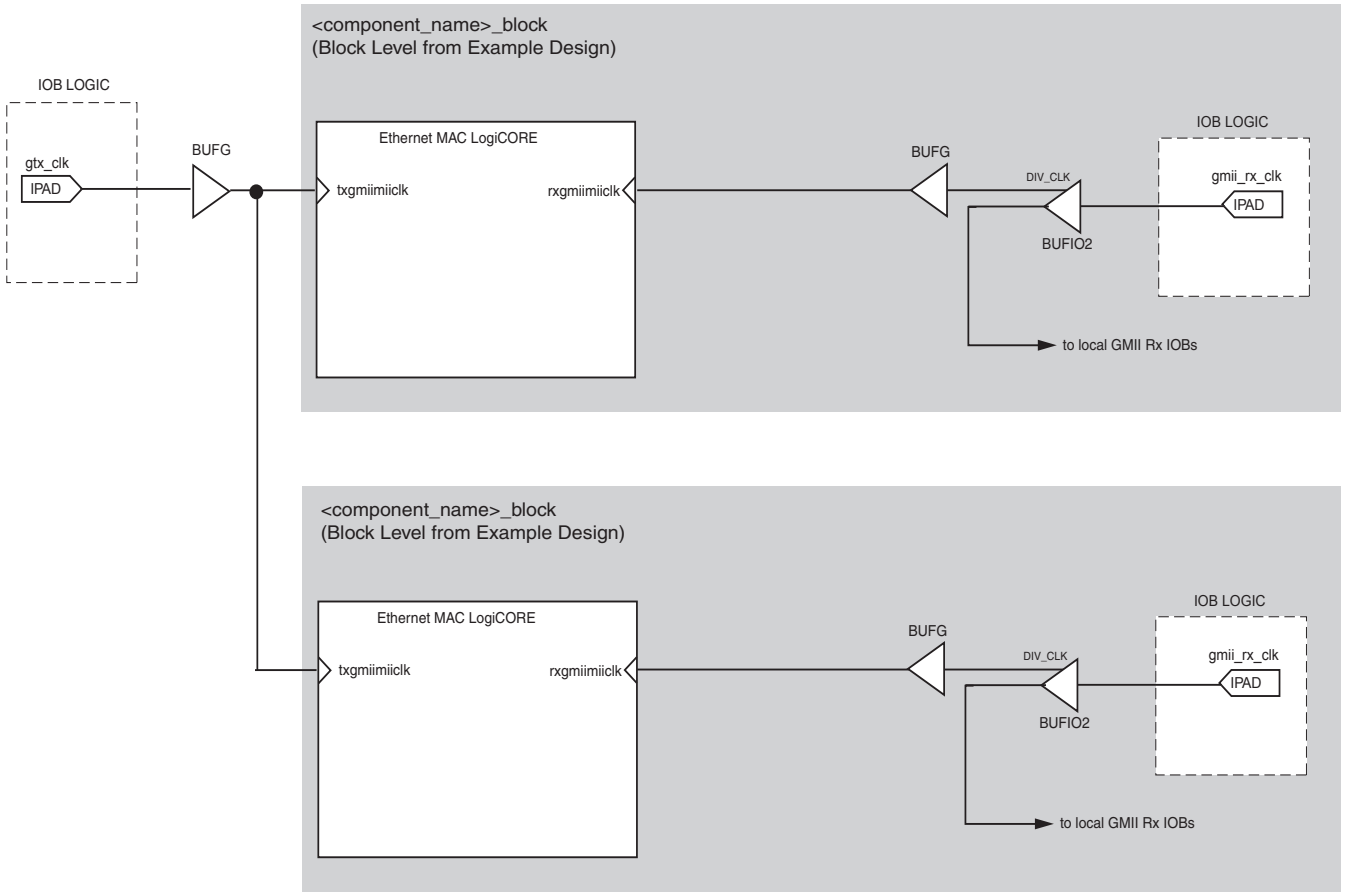

Figure 12-5: **Clock Resource Sharing for 1-Gb/s GMII in Spartan-6 Devices**

Reduced Gigabit Media Independent Interface (RGMII)

The logic required to implement the RGMII logic is described in the following sections. Logical implementation is different for different device families. See the specific family section:

- 7 Series and Virtex-6 Devices
- Spartan-6 Devices

7 Series and Virtex-6 Devices

Transmitter Logic for 7 Series using HP I/O and Virtex-6

The logic required to implement the RGMII transmitter logic is illustrated in Figure 12-6. `gtx_clk` is a user-supplied 125 MHz reference clock source 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 interface of the core.

Figure 12-6 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 12-6 shows that the output transmitter signals are registered in device IOBs, using DDR registers, before driving them to the device pads.

Note: Virtex-6 devices support RGMII at 2.5 V or lower; For 7 Series families it depends on the type of I/O used: HR I/O supports RGMII at 2.5 V whereas HP I/O only supports 1.8 V or lower. Despite this being the defined RGMII voltage most PHYs require 2.5V and therefore an external voltage converter is required to interoperate with any multi-standard PHY when using 7 Series HP I/O.

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.

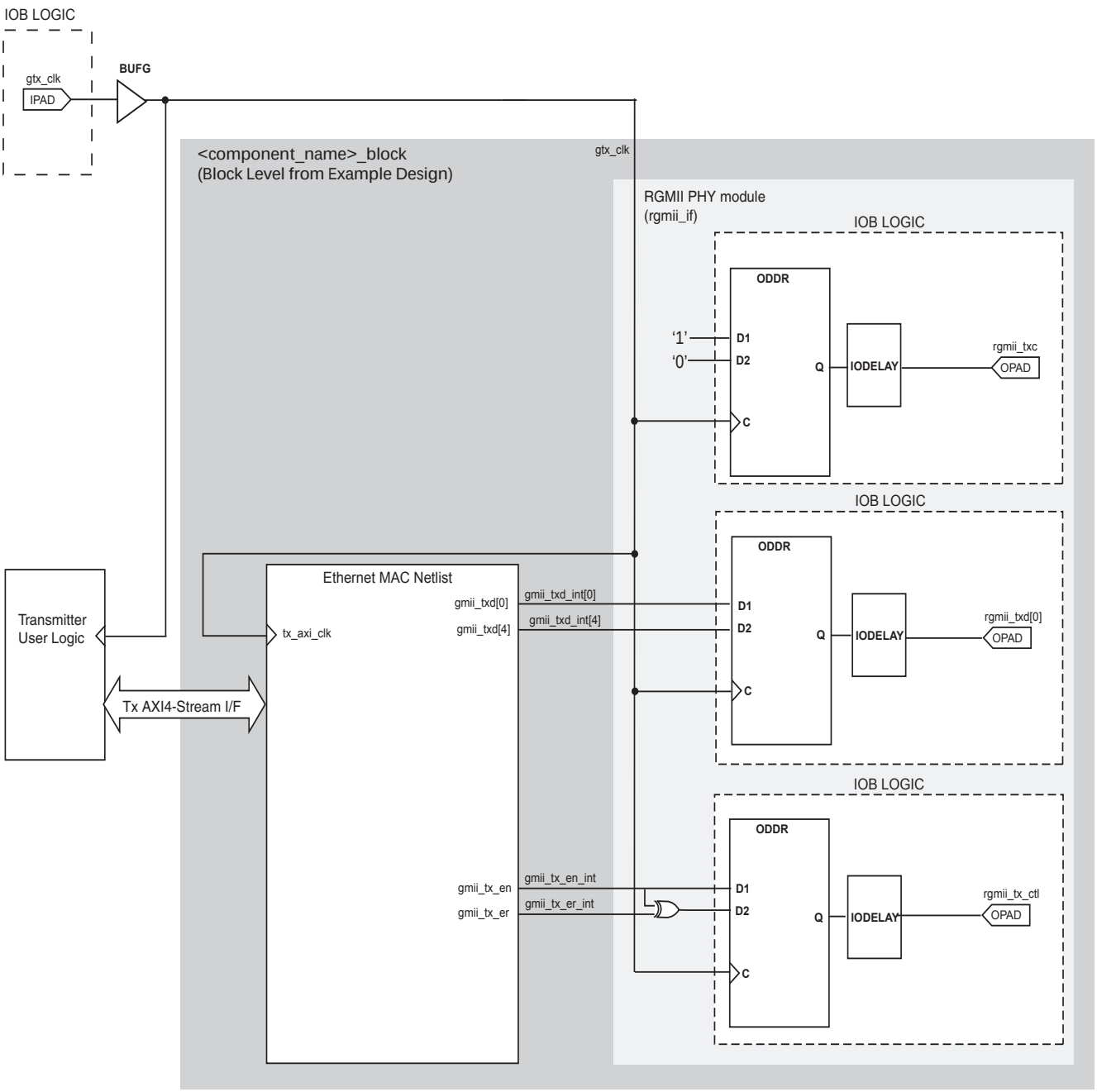

Figure 12-6: **RGMII Transmitter Logic and Clock Logic for 7 Series using HP I/O and Virtex-6 Devices**

Transmitter Logic for 7 Series using HR I/O

HR I/O do not include ODELAY components and another method is required to introduce the required 2 ns offset between the clock and data.

The logic required to implement the RGMII transmitter logic is illustrated in Figure 12-6. `gtx_clk and gtx_clk90` are user-supplied 125 MHz reference clock sources with gtx_clk90 having a 90º phase shift with respect to `gtx_clk`. These are placed onto global clock routing to provide the clocks for all transmitter logic. `Gtx_clk` is used as for the RGMII data and control and both within the core and for the user side logic which connects to the transmitter AXI4-Stream interface of the core. `Gtx_clk90` is used for the RGMII clock only.

Figure 12-6 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 12-6 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 uses the 90º phase shifted version of the clock. 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.

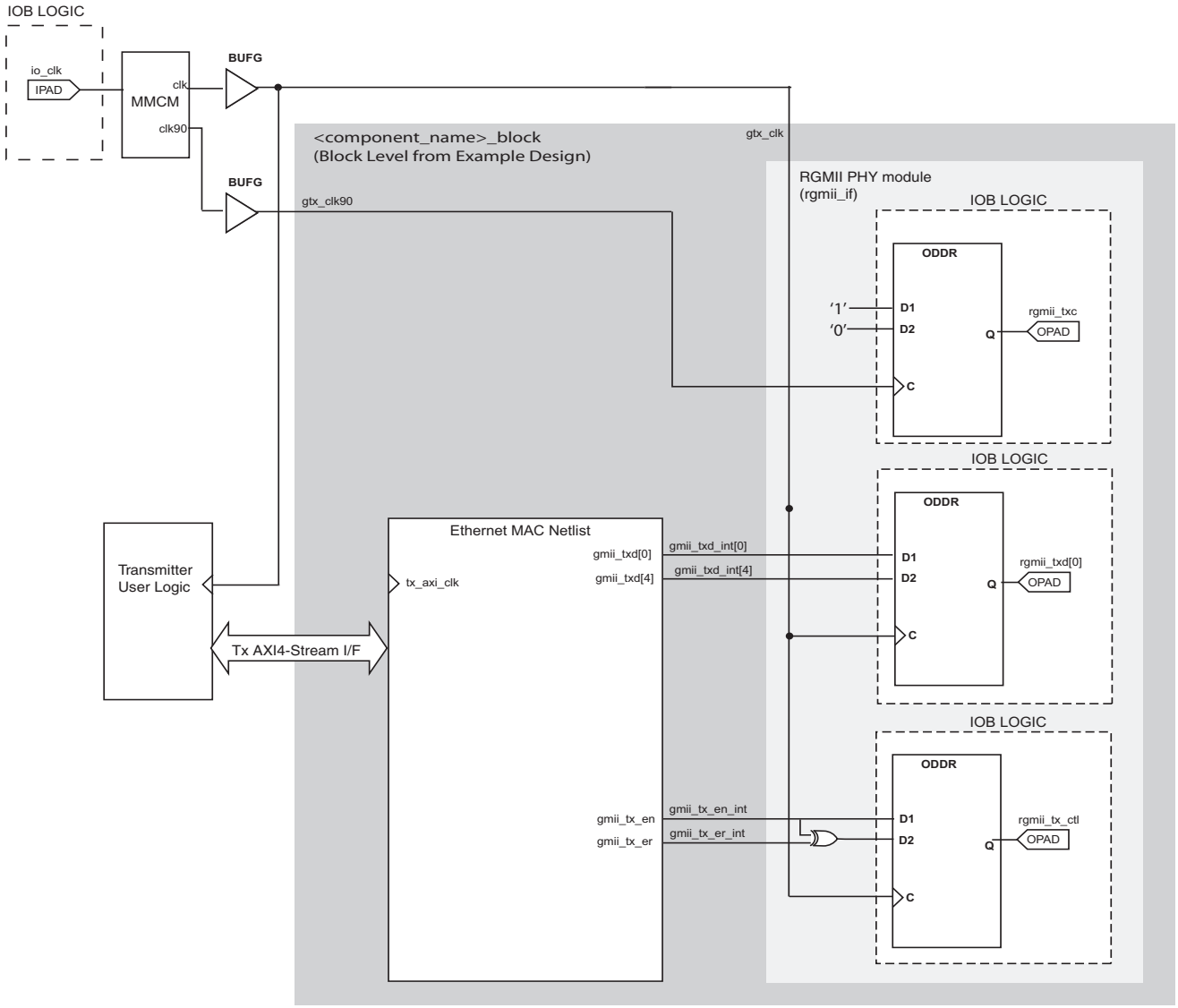

Figure 12-7: **RGMII Transmitter Logic and Clock Logic for 7 Series using HR I/O**

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. 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. The relevant family *User Guide* should be consulted.

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 interface of the core.

The IODELAY elements can be adjusted to fine-tune the setup and hold times at the RGMII 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.

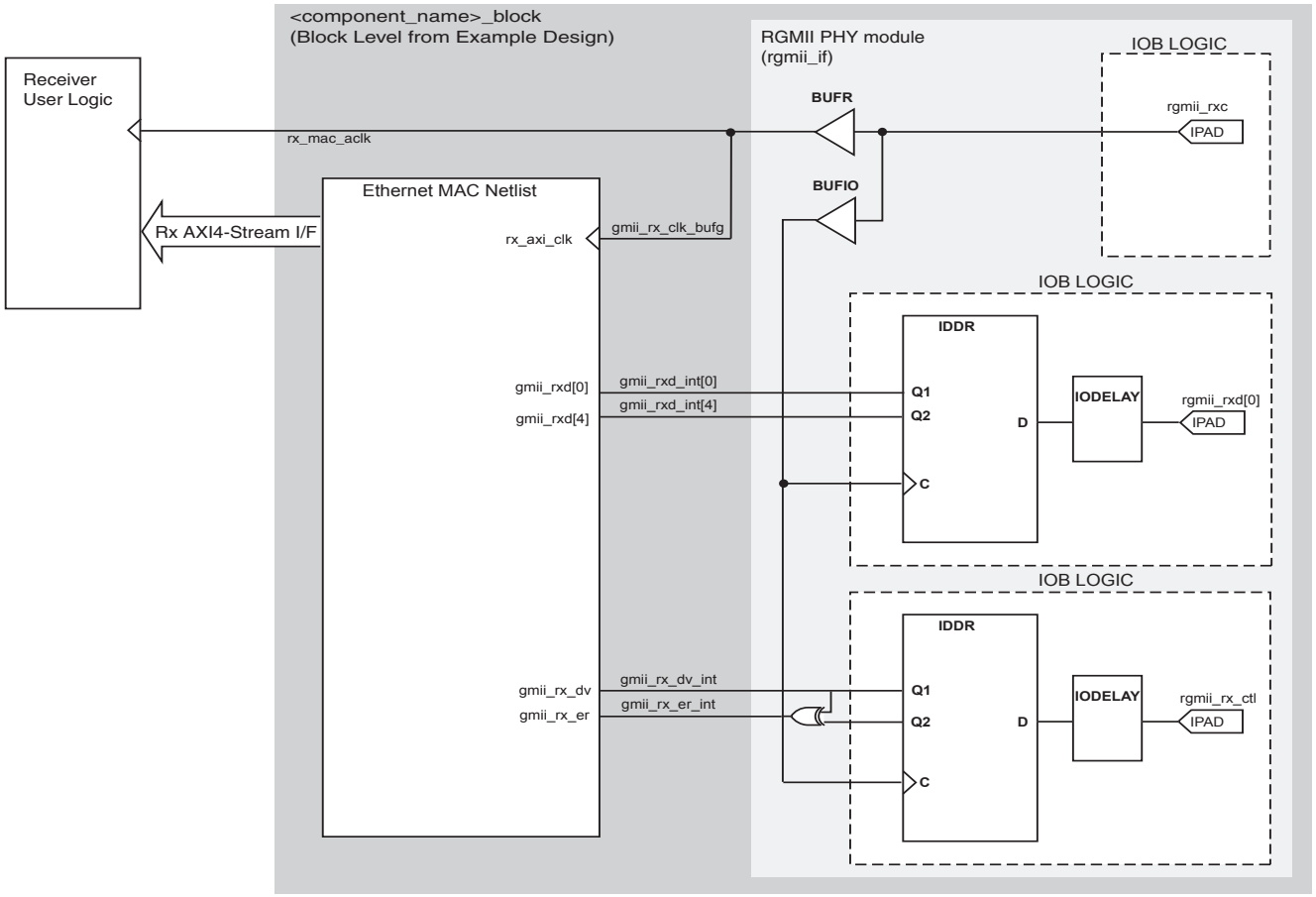

Figure 12-8: **RGMII Receiver Logic and Clock Logic for Virtex-7, Kintex-7 and Virtex-6 Devices**

Clock Resource Sharing

Figure 12-9 illustrates clock resource sharing across multiple instantiations of the core when using RGMII at 1 Gb/s in 7 Series devices using HP I/O and Virtex-6 devices. Figure 12-10 illustrates clock resource sharing across multiple instantiations of the core when using RGMII at 1 Gb/s in 7 Series devices using HR I/O. For all instantiations, `gtx_clk, and gtx_clk90 where present,` can be shared between multiple cores, resulting in a common clock domain across the device.

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

Figure 12-9 and Figure 12-10 illustrates 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.

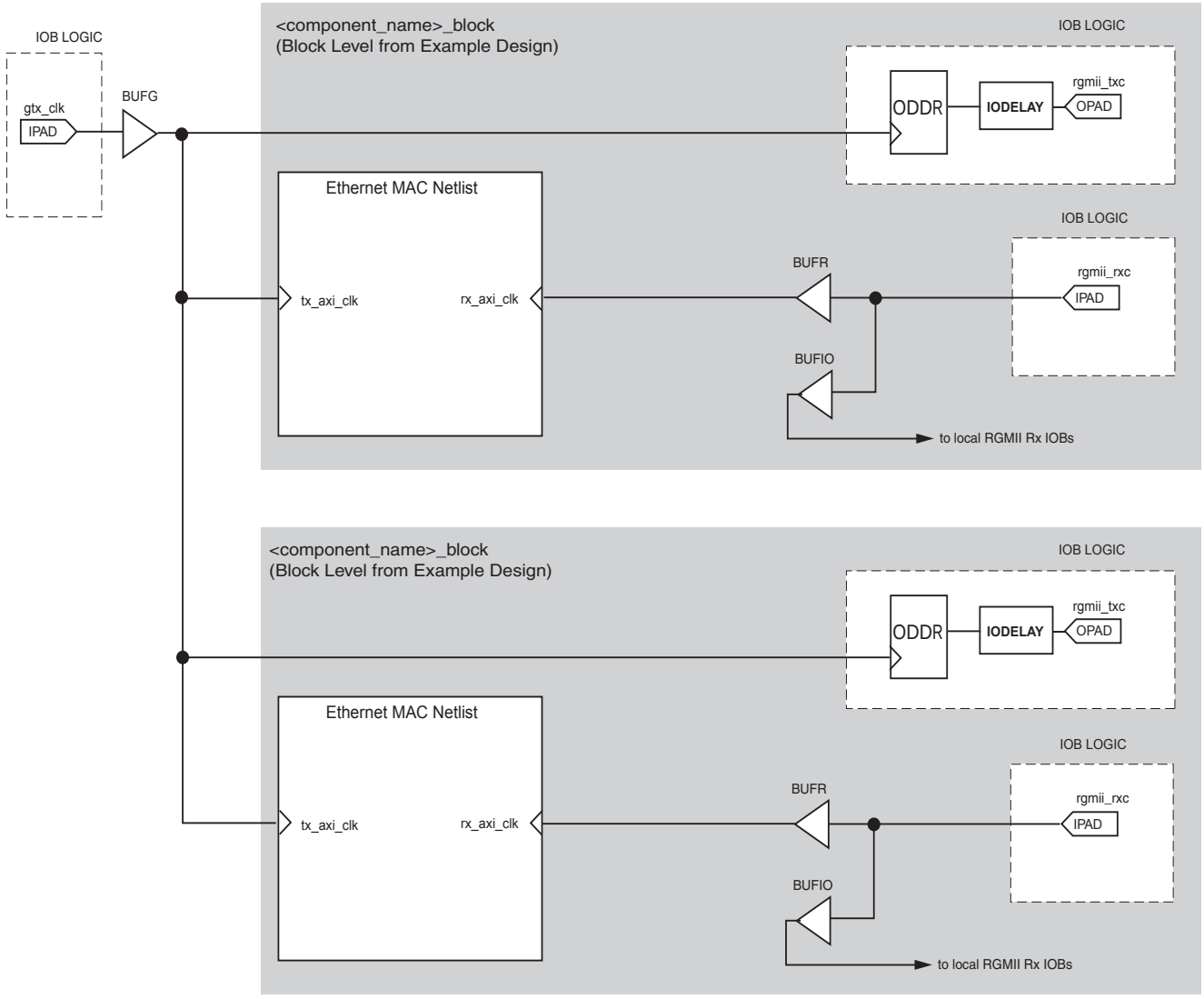

Figure 12-9: **Clock Resource Sharing for 1-Gb/s RGMII in Virtex, Kintex-7 using HP I/O and Virtex-6 Devices**

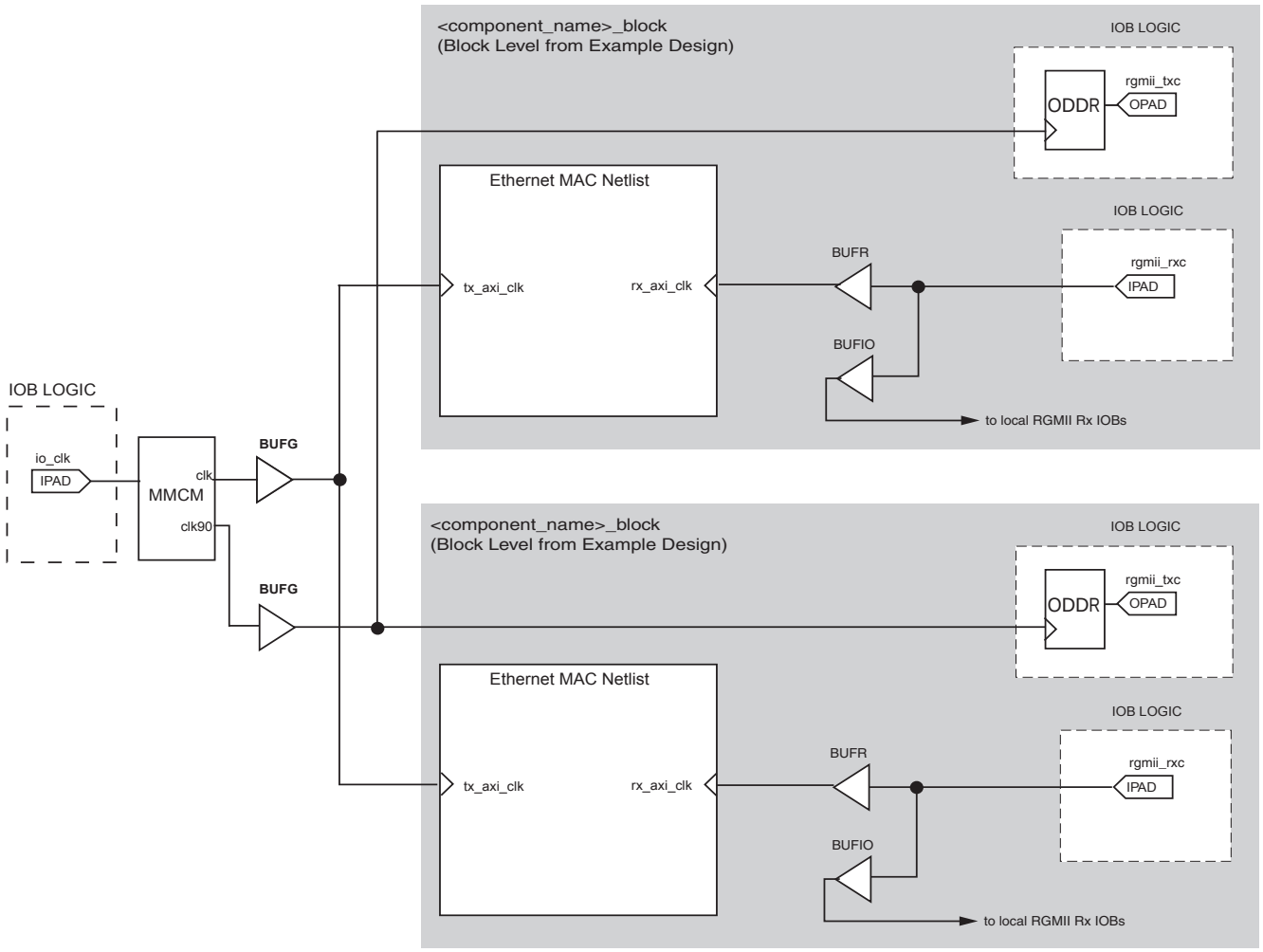

Figure 12-10: **Clock Resource Sharing for 1-Gb/s RGMII in 7 Series using HR I/O**

Spartan-6 Devices

Transmitter Logic

The logic required to implement the RGMII transmitter logic is illustrated in Figure 12-11. `gtx_clk` is a user-supplied 125 MHz reference clock source 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 interface of the core.

Figure 12-11 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 12-11 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 (IODELAY2) 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.

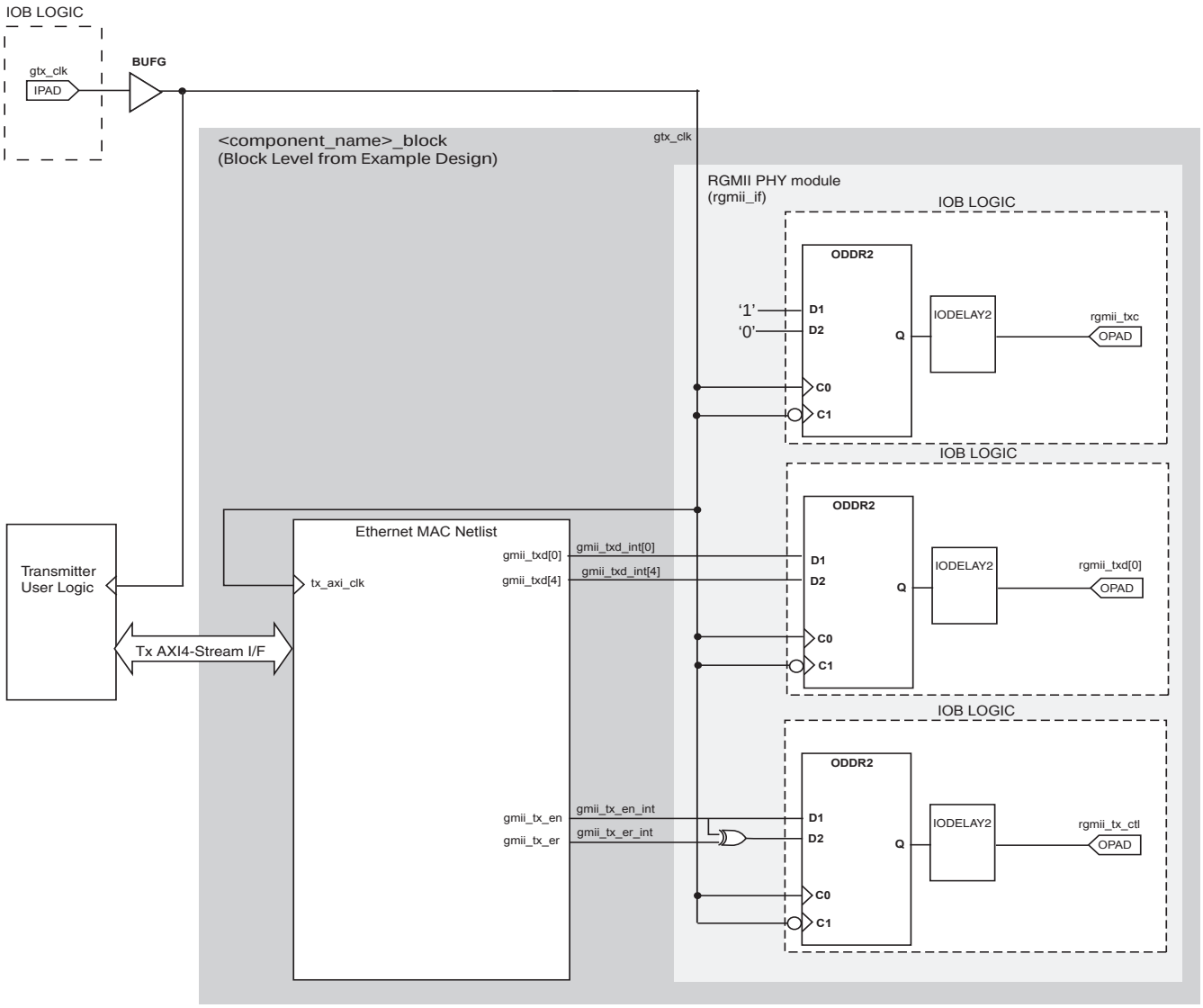

Figure 12-11: **RGMII Transmitter Logic and Clock Logic for Spartan-6 Devices**

Receiver Logic

In Spartan-6 devices, a BUFG is used on the `rgmii_rxc clock` path with IODELAY2s on the datapaths as illustrated in Figure 12-12 to meet the RGMII input setup and hold requirements. This logic is implemented by the example design delivered with the core (all signal names and logic match).

The tap delays of the individual IODELAY2s can then be adjusted to fine-tune the setup and hold times of the input RGMII receiver signals which are sampled at the RGMII IOB flip-flops; a fixed tap delay is applied to the IODELAY2s using the example UCF for the example design.See Chapter 14, Constraining the Core.

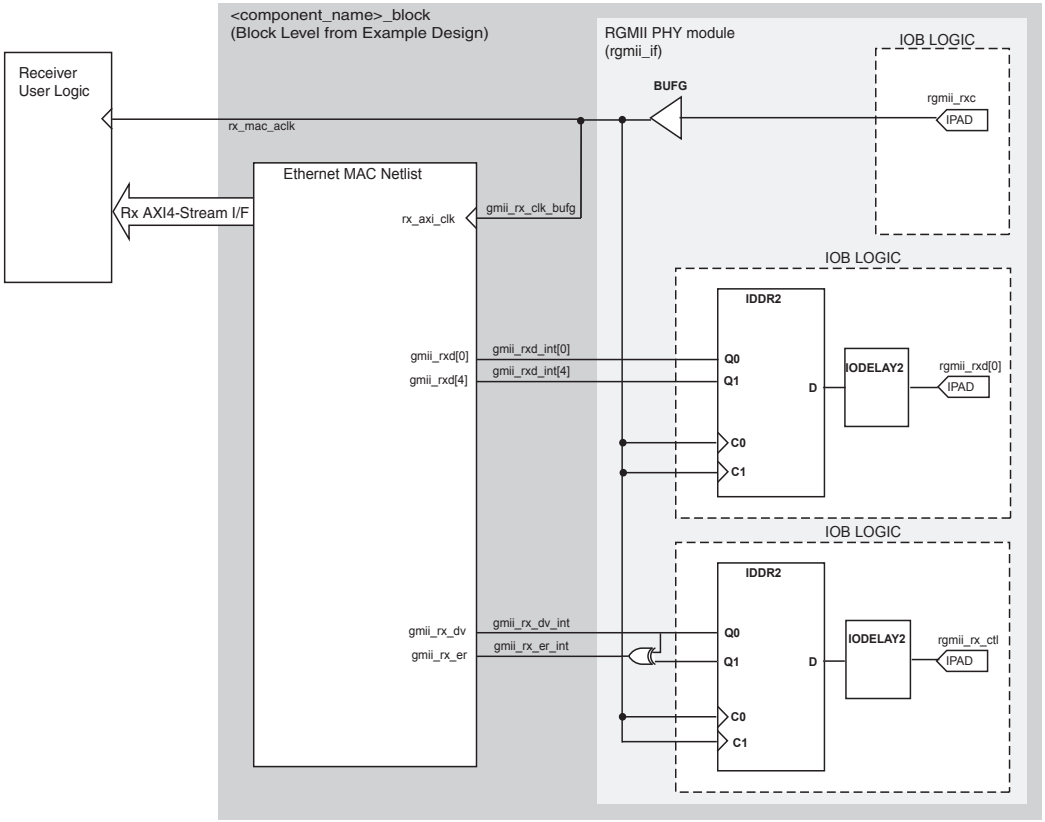

Figure 12-12: **RGMII Receiver Logic and Clock Logic for Spartan-6 Devices**

Clock Resource Sharing

Figure 12-13 illustrates clock resource sharing across multiple instantiations of the core when using RGMII at 1 Gb/s in Spartan-6 devices. For all instantiations, `gtx_clk` can be shared between multiple cores, resulting in a common clock domain across the device.

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

Figure 12-13 illustrates 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.

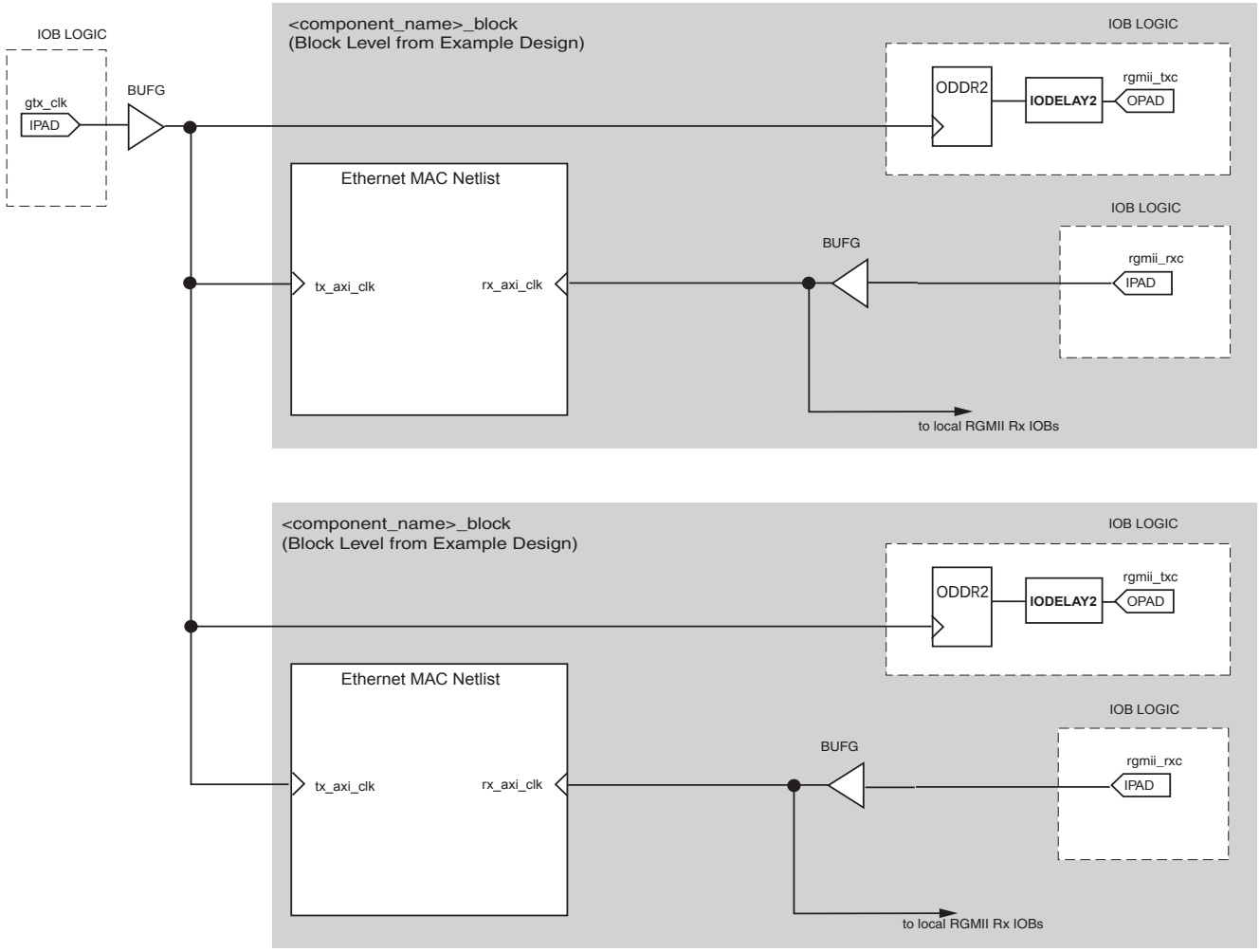

Figure 12-13: **Clock Resource Sharing for 1 Gb/s RGMII in Spartan-6 Devices**

XILINX®

Chapter 13

Physical Interfaces for Tri-speed (10 Mb/s, 100 Mb/s and 1 Gb/s) Ethernet MAC IP Core

The HDL example design supplied with the core, for Tri-speed operation, provides either a GMII or RGMII interface. These are typically used to connect the MAC to an external PHY device.

The Media Independent Interface (MII), defined in [Ref 10], 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 [Ref 10], clause 35, is an extension of the MII and is 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.

Virtex®-6 devices support GMII at 2.5 V only; Spartan®-6 devices support GMII at 3.3 V or lower. For 7 Series families it depends on the type of I/O used: HR I/O supports GMII at 2.5 V whereas HP I/O only supports 1.8 V or lower and therefore an external voltage converter is required to interoperate with any multi-standard PHY for GMII.

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. A further advantage of the RGMII implementation is that, unlike GMII/MII, clock resources for the transmitter can be shared across multiple core instances. This results in significant clock resource savings when implementing multiple cores in a design.

Virtex®-6 devices support RGMII at 2.5 V or lower; Spartan®-6 devices support RGMII at 3.3 V or lower. For 7 Series families it depends on the type of I/O used: HR I/O supports RGMII at 2.5 V whereas HP I/O only supports 1.8 V or lower. Despite this being the defined RGMII voltage most PHYs require 2.5 V and therefore an external voltage converter is required to interoperate with any multi-standard PHY for RGMII.

See the appropriate section:

- Gigabit Media Independent Interface (GMII)
- Reduced Gigabit Media Independent Interface (RGMII)

Gigabit Media Independent Interface (GMII)

GMII Transmitter Interface

The logic required to implement the GMII transmitter logic is illustrated in Figure 13-1. `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 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.

Closely linked to the clock logic is the use of the `tx_enable` clock enable derivation. This must be provided to the MAC Netlist. 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 13-1 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 for a Virtex-6 device when the GMII is selected. If other families are chosen, equivalent primitives specific to that family are used in the example design.

As shown in Figure 13-1, 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.

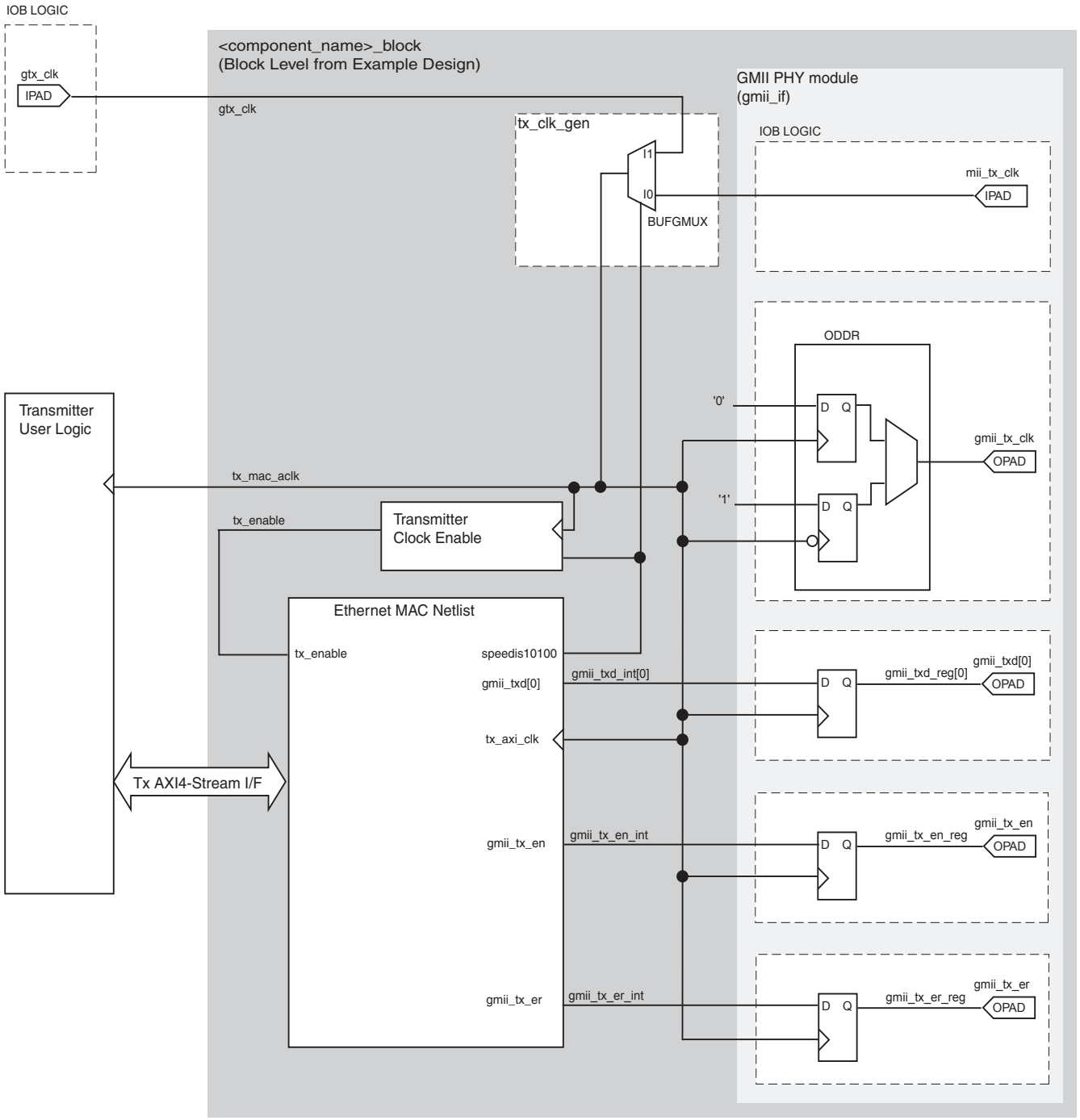

Figure 13-1: **GMII Transmitter Logic and Clock Logic**

GMII Receive Interface

The logic required to implement the GMII receiver logic is described in the following sections. Logical implementation varies for different device families. See the specific family section:

- 7 Series and Virtex-6 Devices
- Spartan-6 Devices

7 Series and Virtex-6 Devices

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. However, 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. The respective family *User Guide* should be consulted.

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 interface of the core.

The IODELAY elements can be adjusted to fine-tune the setup and hold times at the GMII 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.

Closely linked to the clock logic is the use of the `rx_enable` clock enable derivation. This must be provided to the MAC Netlist. 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 core's 8-bit user datapath.

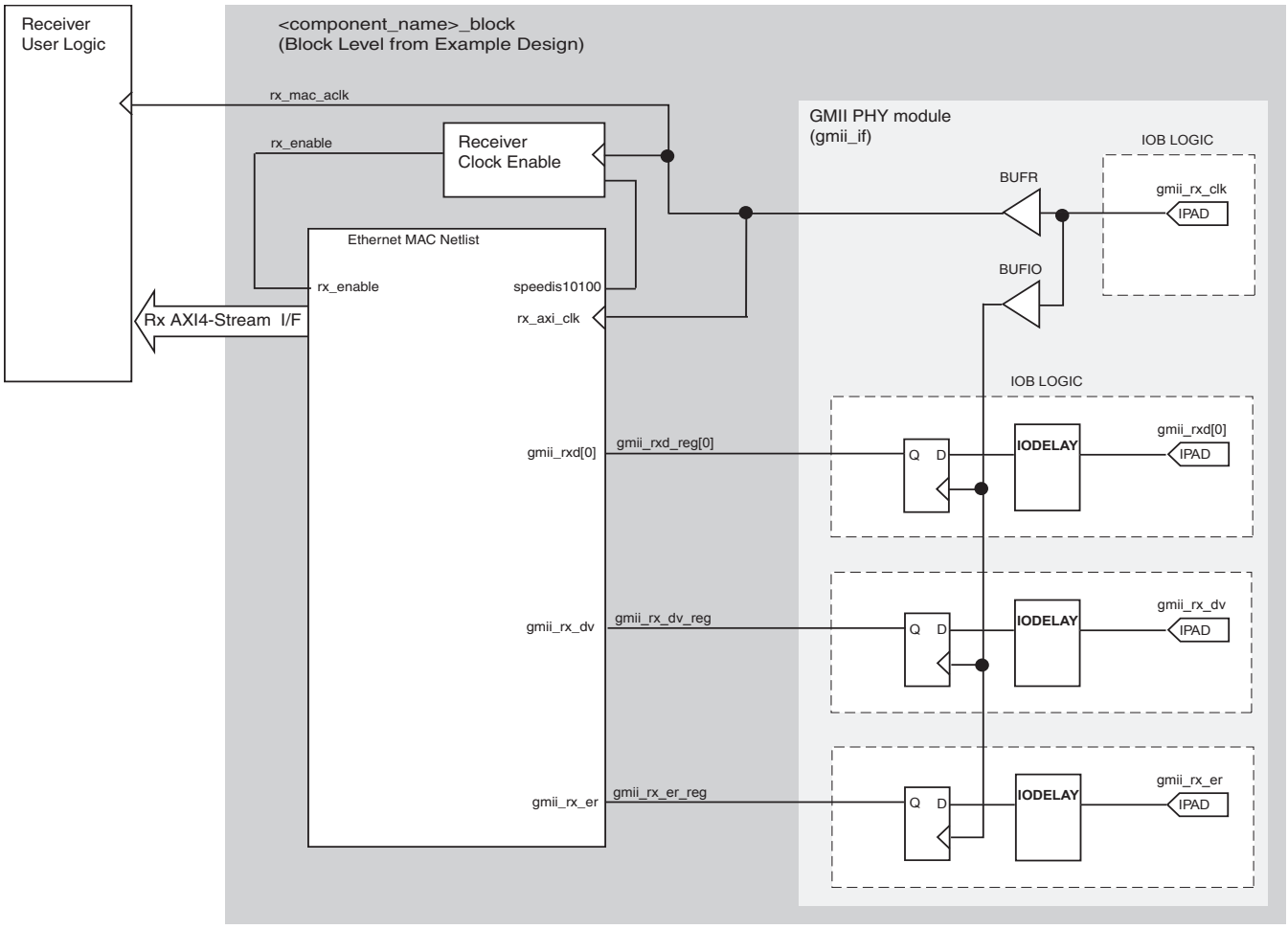

Figure 13-2: **GMII Receiver Logic and Clock Logic for 7 Series and Virtex-6 Devices**

Spartan-6 Devices

In this implementation, a BUFIO2 is used to provide the lowest form of clock routing delay from input clock to input GMII RX signal sampling at the device IOBs. However, this creates placement constraints; a BUFIO2 capable clock input pin must be selected, and all other input GMII RX signals must be placed in the respective BUFIO2 region. The *Spartan-6 FPGA User Guide* should be consulted.

The input clock is also placed onto global clock routing using the BUFG component as illustrated. This 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 interface of the core.

The IODELAY2 elements can be adjusted to fine-tune the setup and hold times at the GMII IOB input flip-flops. This meets input setup and hold constraints at all three ethernet speeds. The delay is applied to the IODELAY2 element using constraints in the UCF; these can be edited if desired. See Chapter 14, Constraining the Core.

Closely linked to the clock logic is the use of the `rx_enable` clock enable derivation. This must be provided to the MAC Netlist. 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.

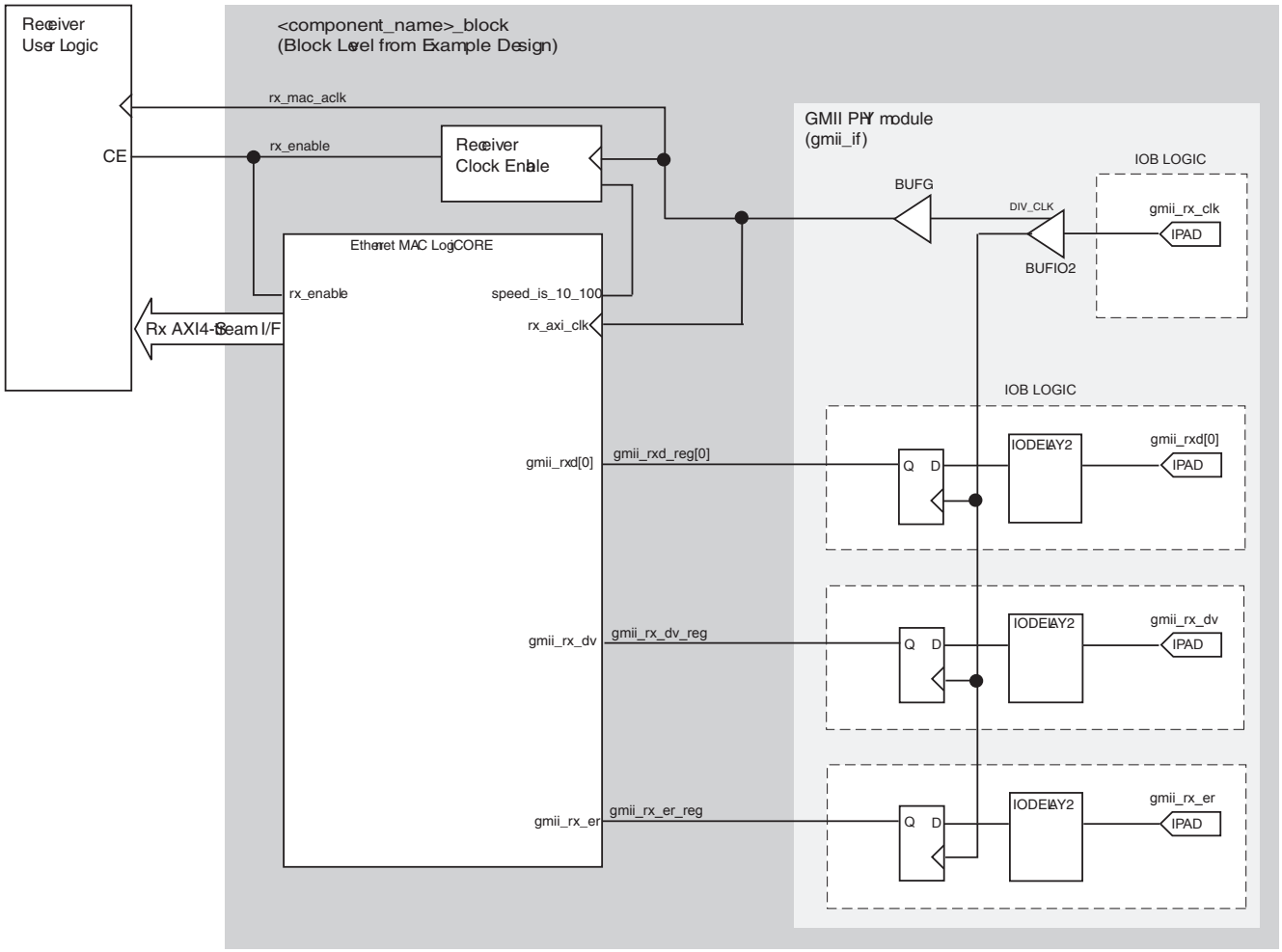

Figure 13-3: **GMII Receiver Logic and Clock Logic for Spartan-6 Devices**

Multiple Core Instantiations

Because both `mii_tx_clk` and `gmii_rx_clk` are both sourced by the external PHY device connected to the GMII/MII, it is not possible to share global transmitter or receiver clock resources across multiple instantiations of the core. Each instance of the core requires its own endpoint clocking resources. See the appropriate section:

- Multiple Core Instances in 7 Series and Virtex-6 Devices
- Multiple Core Instances in Spartan-6 Devices

Note: RGMII provides a more optimal solution because it does allow transmitter clock resources to be shared. See Reduced Gigabit Media Independent Interface (RGMII).

Multiple Core Instances in 7 Series and Virtex-6 Devices

Figure 13-4 illustrates multiple instances of the core in Virtex-7, Kintex-7, Artix-7 and Virtex-6 devices. 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.

Figure 13-4 illustrates 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 the `gtx_clk` clock source across all instantiations.

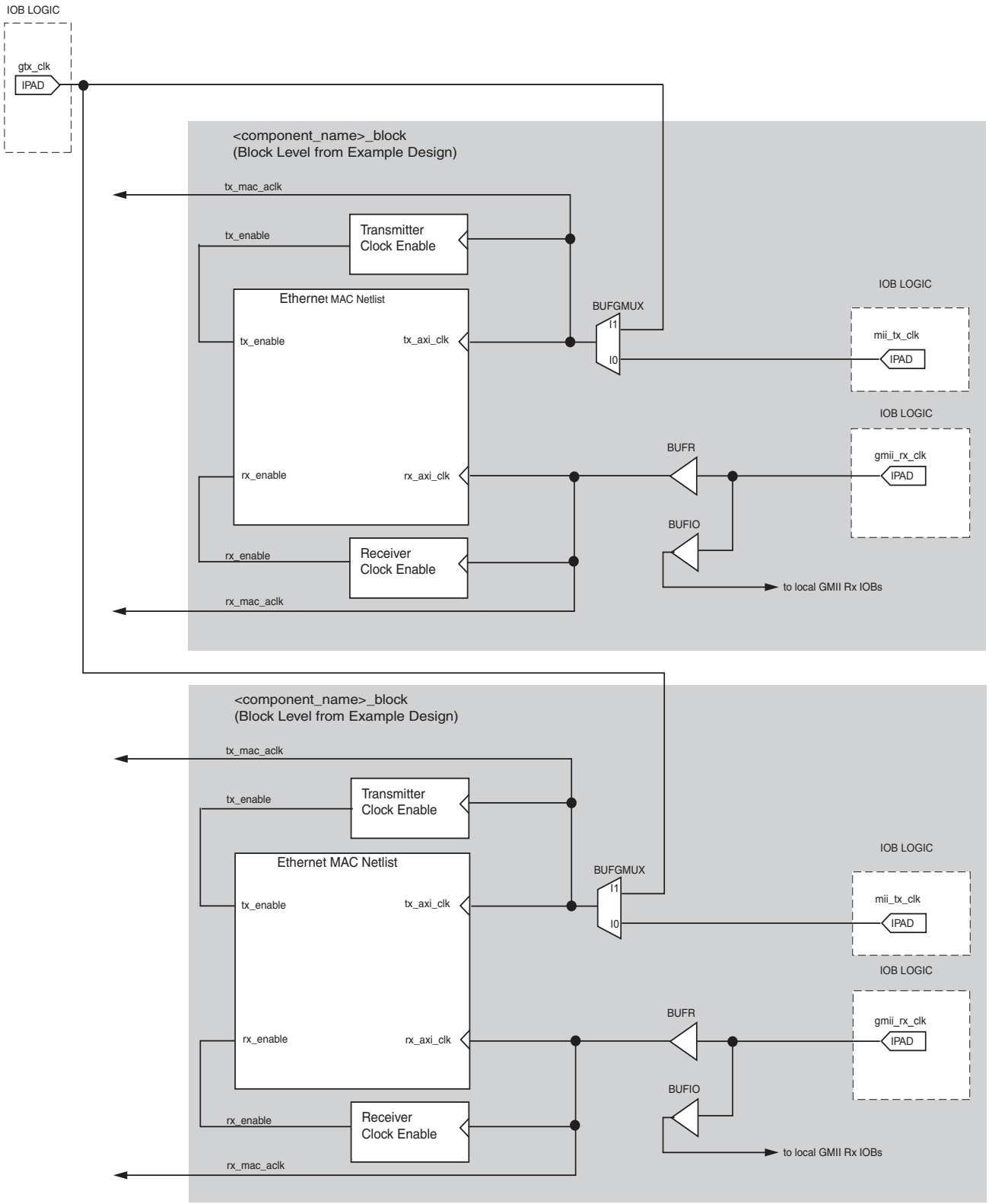

Figure 13-4: **Clock Resource Sharing for 1 Gb/s GMII in 7 Series and Virtex-6 Devices**

Multiple Core Instances in Spartan-6 Devices

Figure 13-5 illustrates multiple instances of the core in Spartan-6 devices. 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.

Figure 13-5 illustrates 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 the `gtx_clk` clock source across all instantiations.

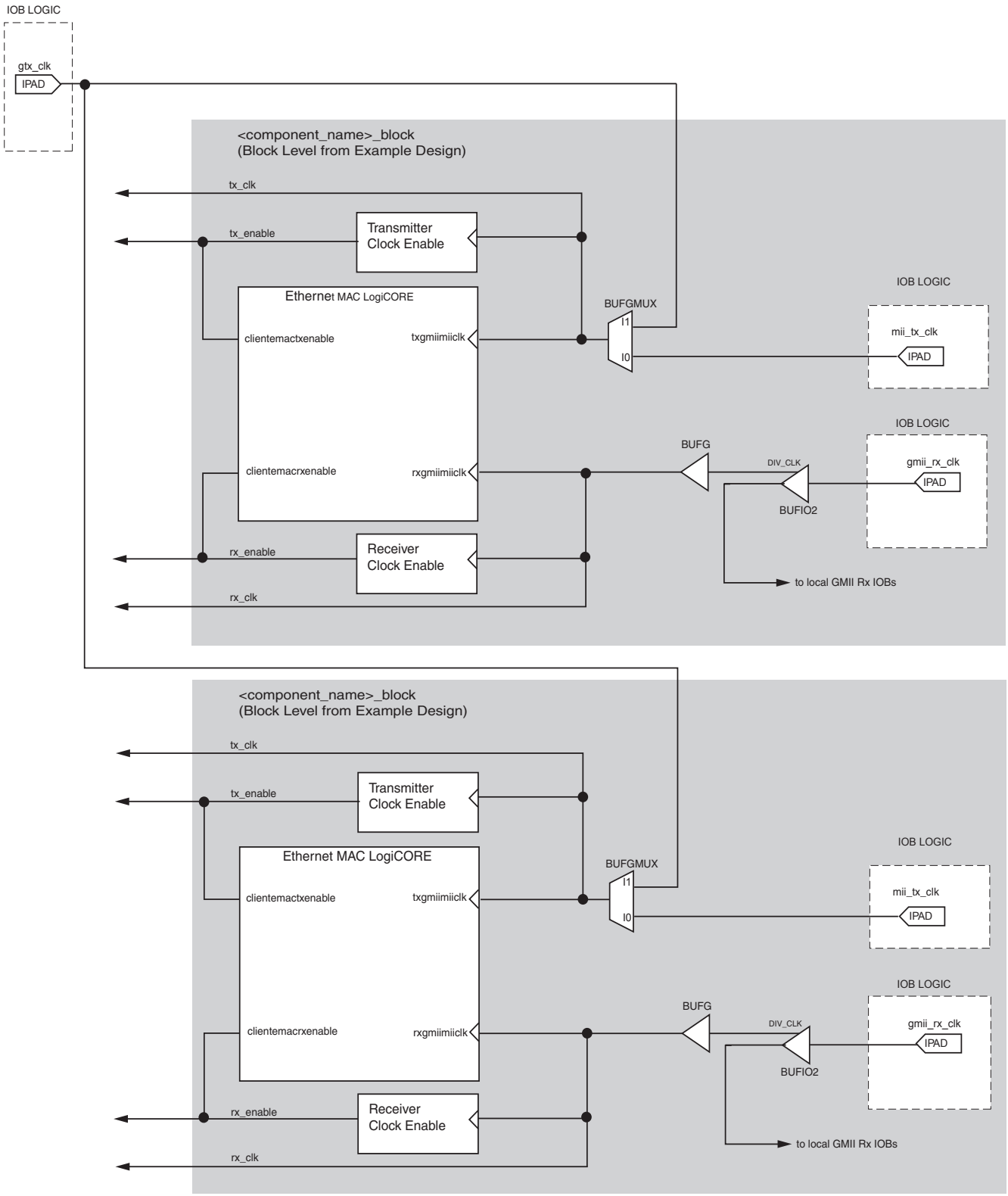

Figure 13-5: **Clock Resource Sharing for 1 Gb/s GMII in Spartan-6 Devices**

Reduced Gigabit Media Independent Interface (RGMII)

The logic required to implement the RGMII logic is described in the next sections. Logical implementation varies for different device families. See the specific device section:

- 7 Series and Virtex-6 Devices
- Spartan-6 Devices

7 Series and Virtex-6 Devices

Transmitter Logic for 7 Series using HP I/O and Virtex-6

The logic required to implement the RGMII transmitter logic is illustrated in Figure 13-6. `gtx_clk` is a user-supplied 125 MHz reference clock source 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 interface of the TEMAC.

For RGMII, this global 125 MHz is used to clock transmitter logic at all three ethernet speeds. The data rate difference between the three speeds is compensated for by the transmitter clock enable logic (the `enable_gen` module from the example design describes the required logic). The derived `tx_enable` signal must be supplied to the MAC Netlist. All User logic uses the AXI4-Stream interfaces built in handshaking to throttle the data appropriately, under control of the MAC Netlist. At all speeds the MAC expects the User logic to supply/accept new data after each validated clock cycle. The generated `tx_enable` signal is always high at 1 Gb/s, high for one in ten cycles at 100 Mb/s and high for one in a hundred cycles at 10 Mb/s. The advantage of this approach is that it allows common transmitter global clocks to be shared across any number of instantiated cores.

Figure 13-6 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 13-6 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. At 10 Mb/s and 100 Mb/s speeds, the `enable_gen` module toggles the DDR input signals at the required frequency so that the forwarded `rgmii_txc` clock is always of the correct frequency for the forwarded data.

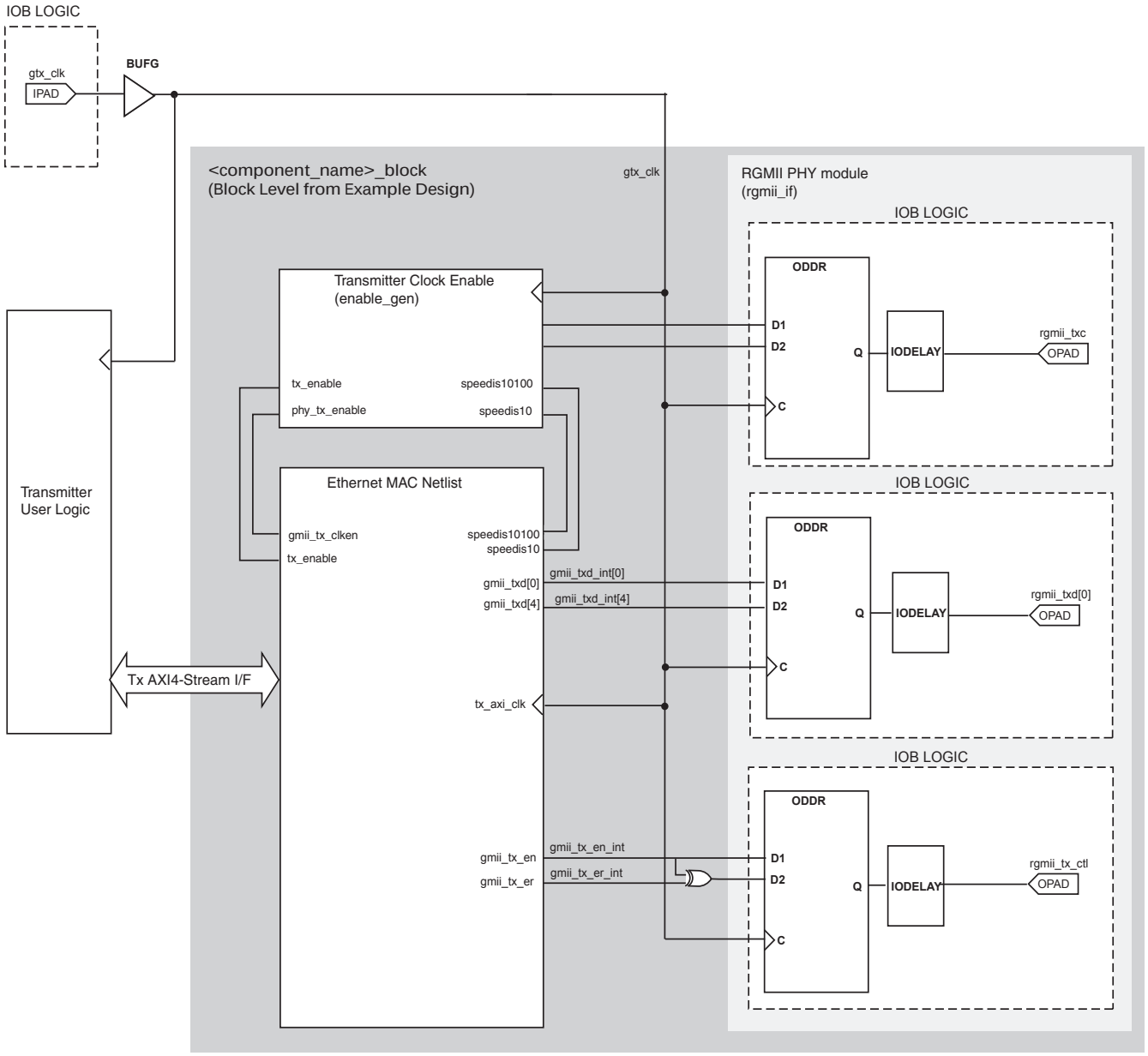

Figure 13-6: **RGMII Transmitter Logic and Clock Logic for Virtex-7, Kintex-7 and Virtex-6 Devices**

Transmitter Logic for 7 Series using HR I/O

HR I/O do not include ODELAY components and another method is required to introduce the required 2 ns offset between the clock and data. The logic required to implement the RGMII transmitter logic is illustrated in Figure 13-7. `gtx_clk and gtx_clk90` are user-supplied 125 MHz reference clock sources with `gtx_clk90` having a 90º phase shift with respect to `gtx_clk`. These are placed onto global clock routing to provide the clocks for all transmitter logic. `gtx_clk` is used for the RGMII data and control is also the clock source for the transmit datapath of the core. `gtx_clk90` is used for the RGMII clock only.

For RGMII, this global 125 MHz is used to clock transmitter logic at all three ethernet speeds. The data rate difference between the three speeds is compensated for by the transmitter clock enable logic (the `enable_gen` module from the example design

describes the required logic). The derived `tx_enable` signal must be supplied to the MAC Netlist. All User logic uses the AXI4-Stream interfaces built in handshaking to throttle the data appropriately, under control of the MAC Netlist. At all speeds the MAC expects the User logic to supply/accept new data after each validated clock cycle. The generated `tx_enable` signal is always high at 1 Gb/s, high for one in ten cycles at 100 Mb/s and high for one in a hundred cycles at 10 Mb/s. The advantage of this approach is that it allows common transmitter global clocks to be shared across any number of instantiated cores.

Figure 13-7 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 13-7 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 uses the 90 degree phase shifted version of the clock. 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. At 10 Mb/s and 100 Mb/s speeds, the `enable_gen` module toggles the DDR input signals at the required frequency so that the forwarded `rgmii_txc` clock is always of the correct frequency for the forwarded data.

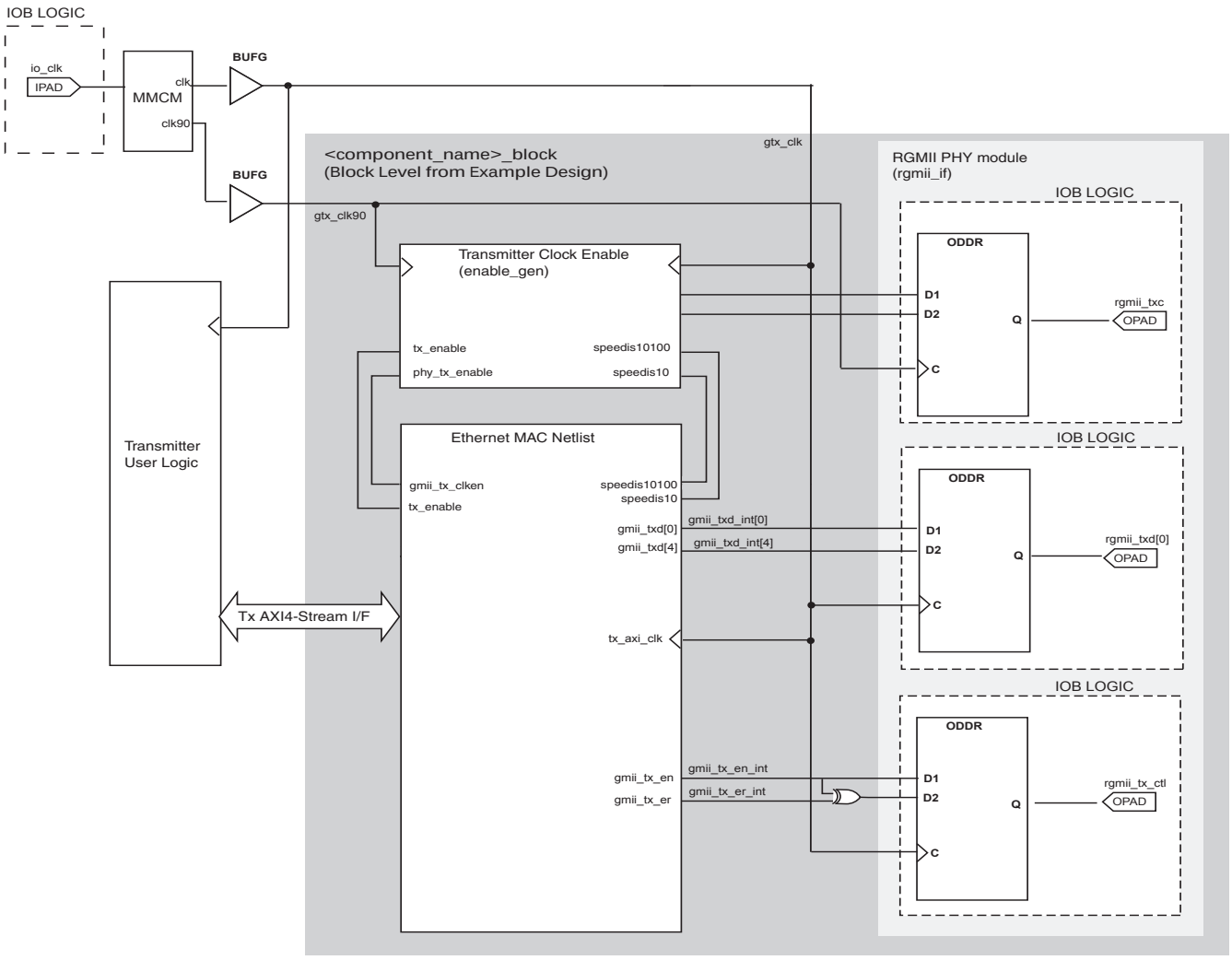

Figure 13-7: **RGMII Transmitter Logic and Clock Logic for 7 Series devices using HR I/O**

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. However, 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. The relevant family *User Guide* should be consulted.

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 interface of the core.

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.

Closely linked to the clock logic is the use of the `rx_enable` clock enable derivation. This must be provided to the MAC Netlist. All User logic uses the AXI4-Stream interface handshaking to throttle the data as required at the different speeds.

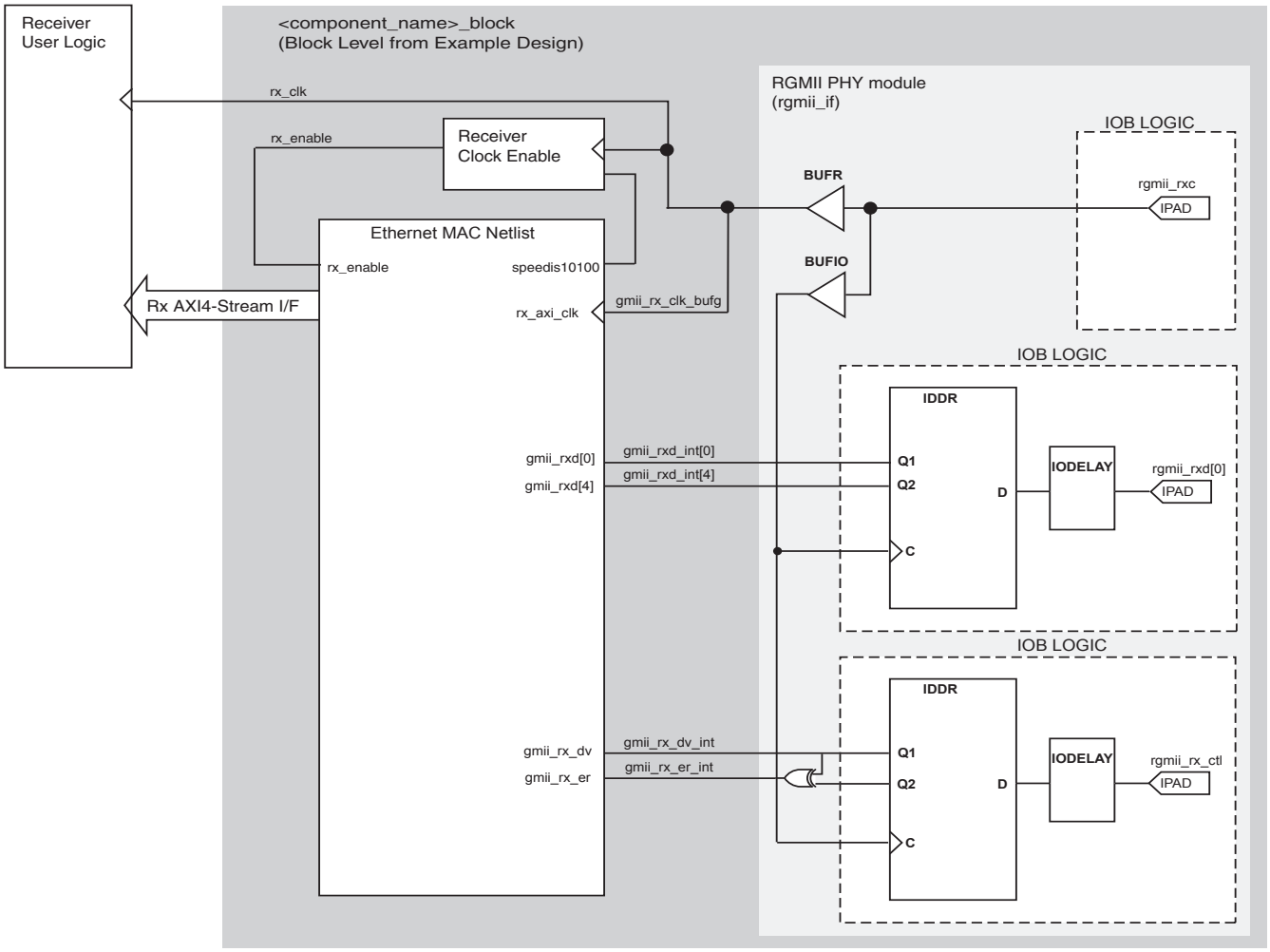

Figure 13-8: **RGMII Receiver Logic and Clock Logic for Virtex-7, Kintex-7 and Virtex-6 Devices**

Clock Resource Sharing

Figure 13-9 illustrates clock resource sharing across multiple instantiations of the core when using RGMII in 7 Series devices using HP I/O and Virtex-6 devices. Figure 13-10 illustrates clock resource sharing across multiple instantiations of the core when using RGMII in 7 Series devices using HR I/O. For all instantiations, `gtx_clk` and gtx_clk90, where present, can be shared between multiple cores, resulting in a common clock domain across the device. The receiver clocks cannot be shared. Each core is provided with its own local version of `rgmii_rxc` from the connected external PHY device as illustrated.

Figure 13-9 and Figure 13-10 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 and gtx_clk90,` if required, across all instantiations. The receiver clock, which cannot be shared, is unique for every instance of the core.

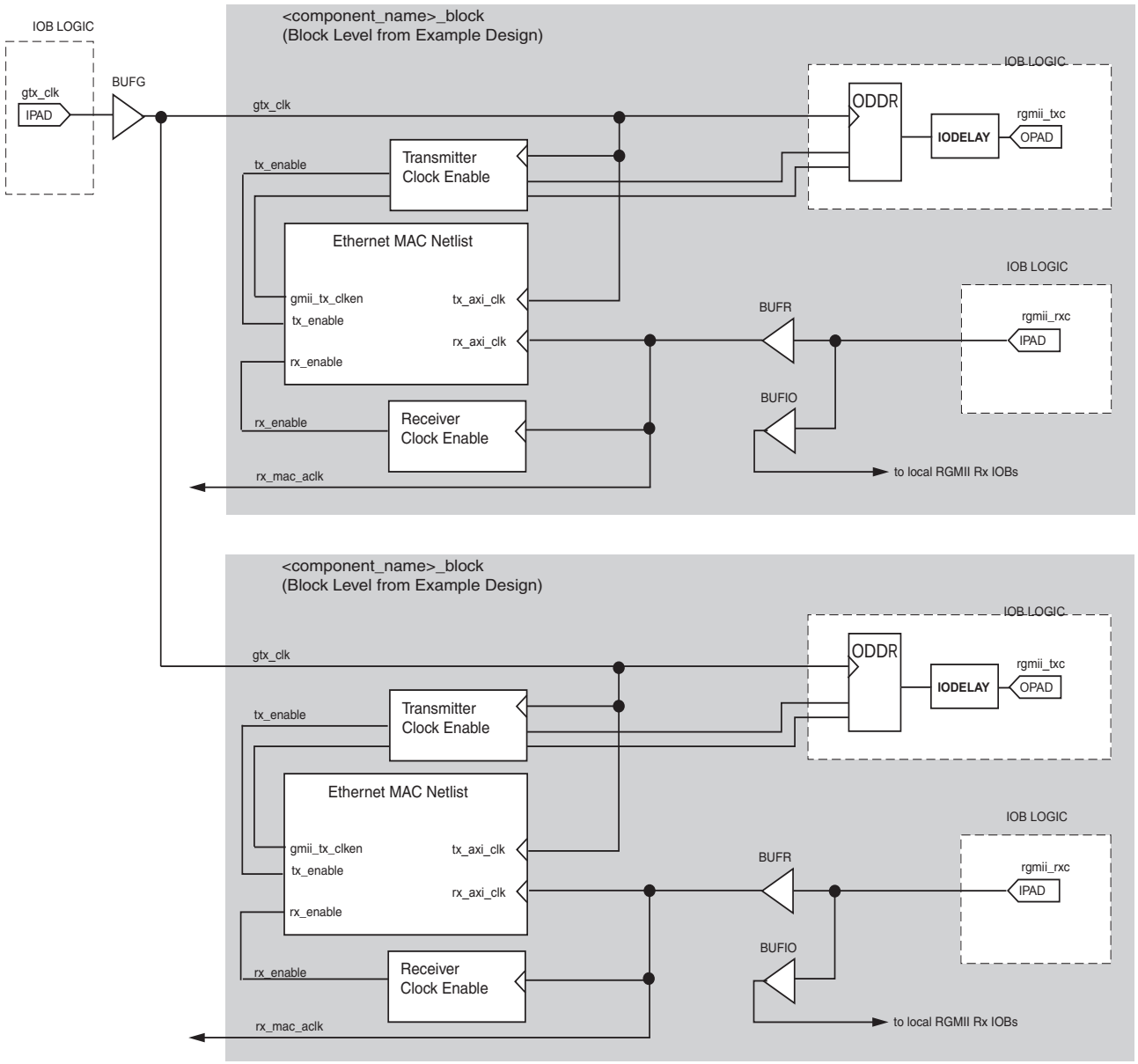

Figure 13-9: **Clock Resource Sharing for 1-Gb/s RGMII in 7 Series using HP I/O and Virtex-6 Devices**

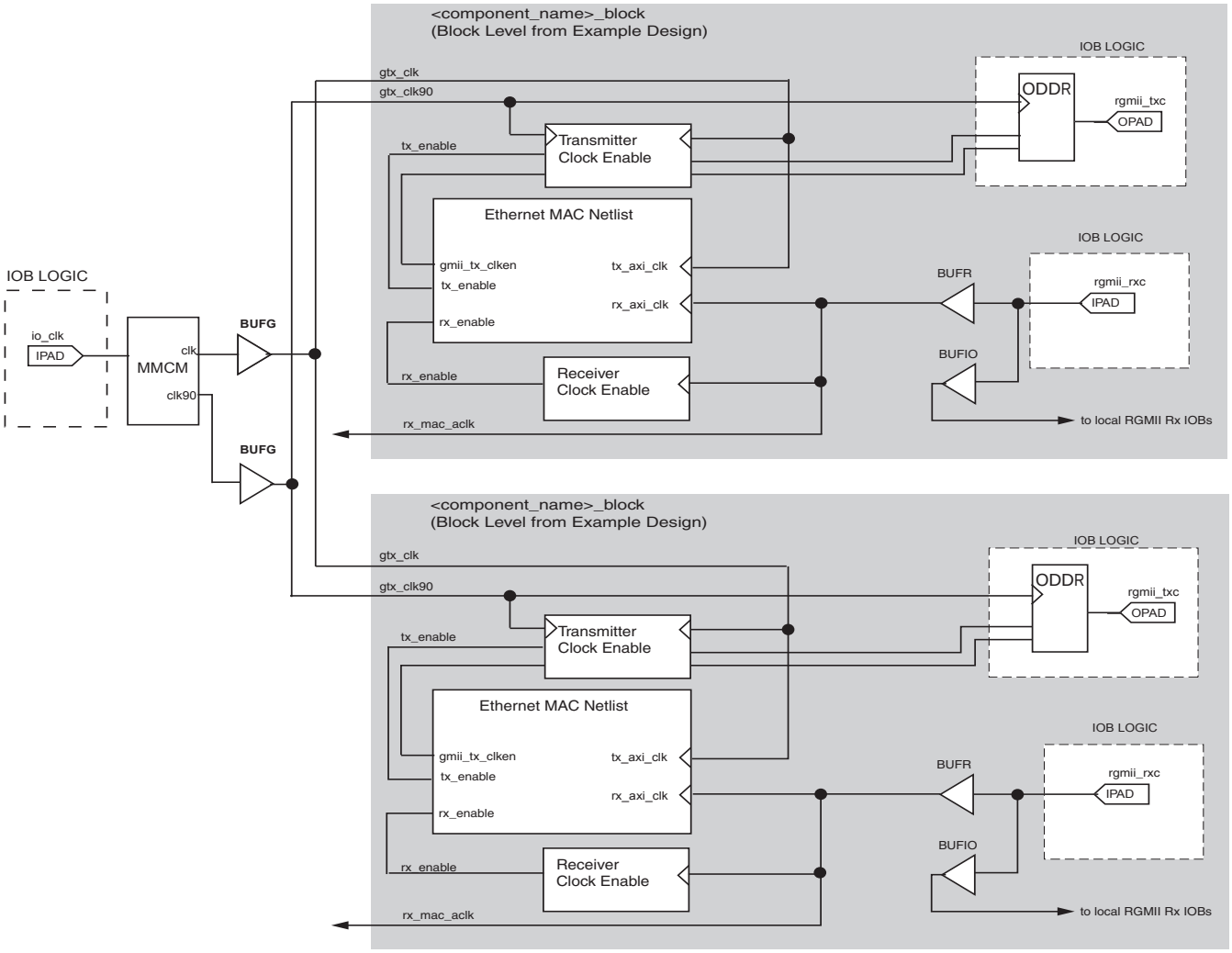

Figure 13-10: **Clock Resource Sharing for 1-Gb/s RGMII in 7 Series devices using HR I/O**

Spartan-6 Devices

Transmitter Logic

The logic required to implement the RGMII transmitter logic is illustrated in Figure 13-11. `gtx_clk` is a user-supplied 125 MHz reference clock source 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 interface of the core.

For RGMII, this global 125 MHz is used to clock transmitter logic at all three ethernet speeds. The data rate difference between the three speeds is compensated for by the transmitter clock enable logic (the enable_gen module from the example design describes the required logic). The derived tx_enable signal must be supplied to the MAC Netlist. All User logic uses the AXI4-Stream interfaces built in handshaking to throttle the data appropriately, under control of the MAC Netlist. At all speeds the MAC expects the User logic to supply/accept new data after each validated clock cycle. The generated tx_enable signal is always high at 1 Gb/s, high for one in ten cycles at 100 Mb/s and high for one in a hundred cycles at 10 Mb/s. The advantage of this approach is that it allows common transmitter global clocks to be shared across any number of instantiated cores.

Figure 13-11 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 13-11 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 (IODELAY2) 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. At 10 Mb/s and 100 Mb/s speeds, the `enable_gen` module toggles the DDR input signals at the required frequency so that the forwarded rgmii_txc clock is always of the correct frequency for the forwarded data.

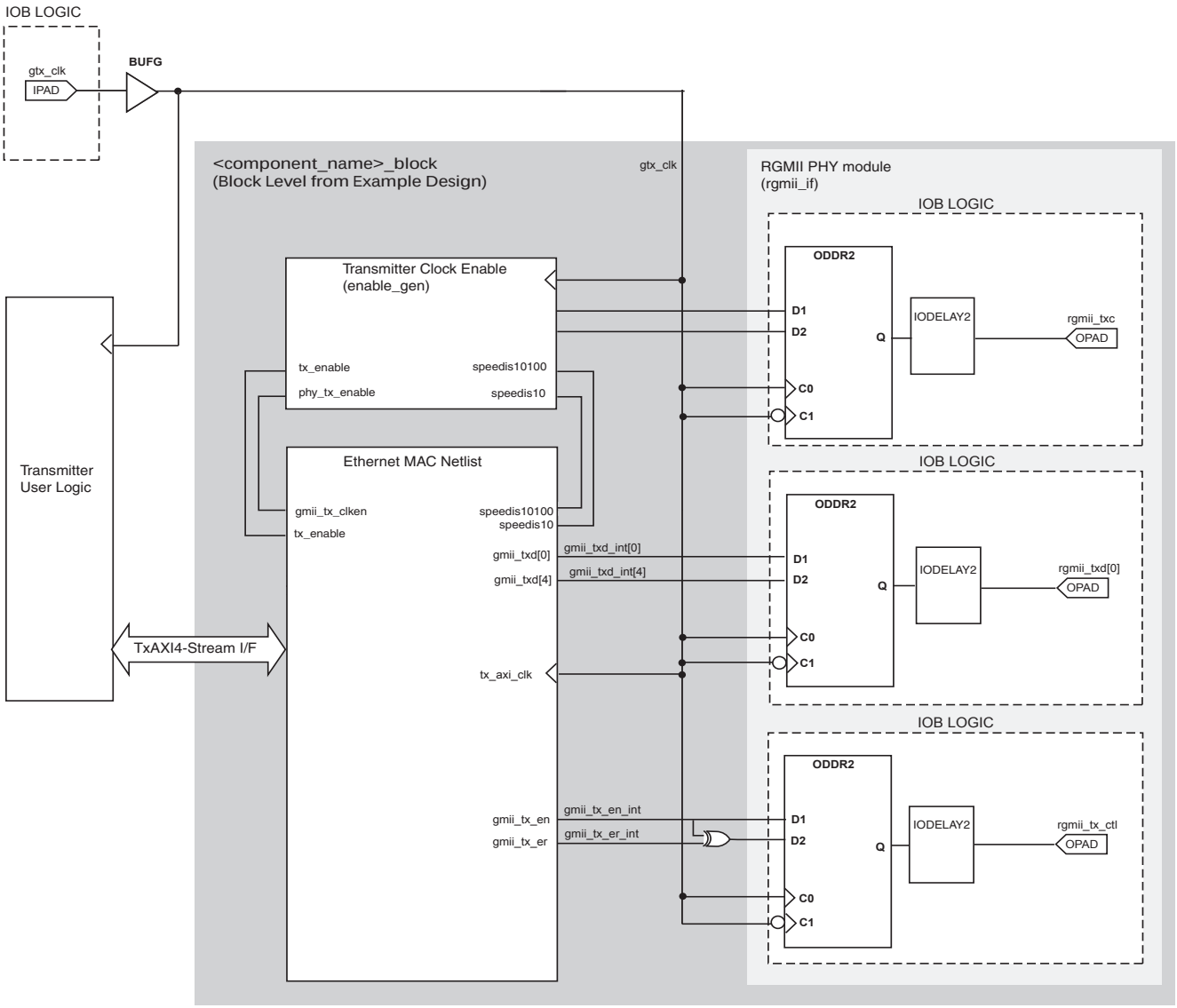

Figure 13-11: **RGMII Transmitter Logic and Clock Logic for Spartan-6 Devices**

Receiver Logic

In Spartan-6 devices, a BUFG is used on the `rgmii_rxc` clock path with IODELAY2s on the datapaths as illustrated in Figure 13-12 to meet the RGMII input setup and hold requirements. This logic is implemented by the example design delivered with the core (all signal names and logic match).

The tap delays of the individual IODELAY2s can then be adjusted to fine-tune the setup and hold times of the input RGMII receiver signals which are sampled at the RGMII IOB flip-flops; a fixed tap delay is applied to the IODELAY2s using the example UCF for the example design. See Chapter 14, Constraining the Core.

Closely linked to the clock logic is the use of the `rx_enable` clock enable derivation. This must be provided to the MAC Netlist. All User logic uses the AXI4-Stream interface handshaking to throttle the data as required at the different speeds.

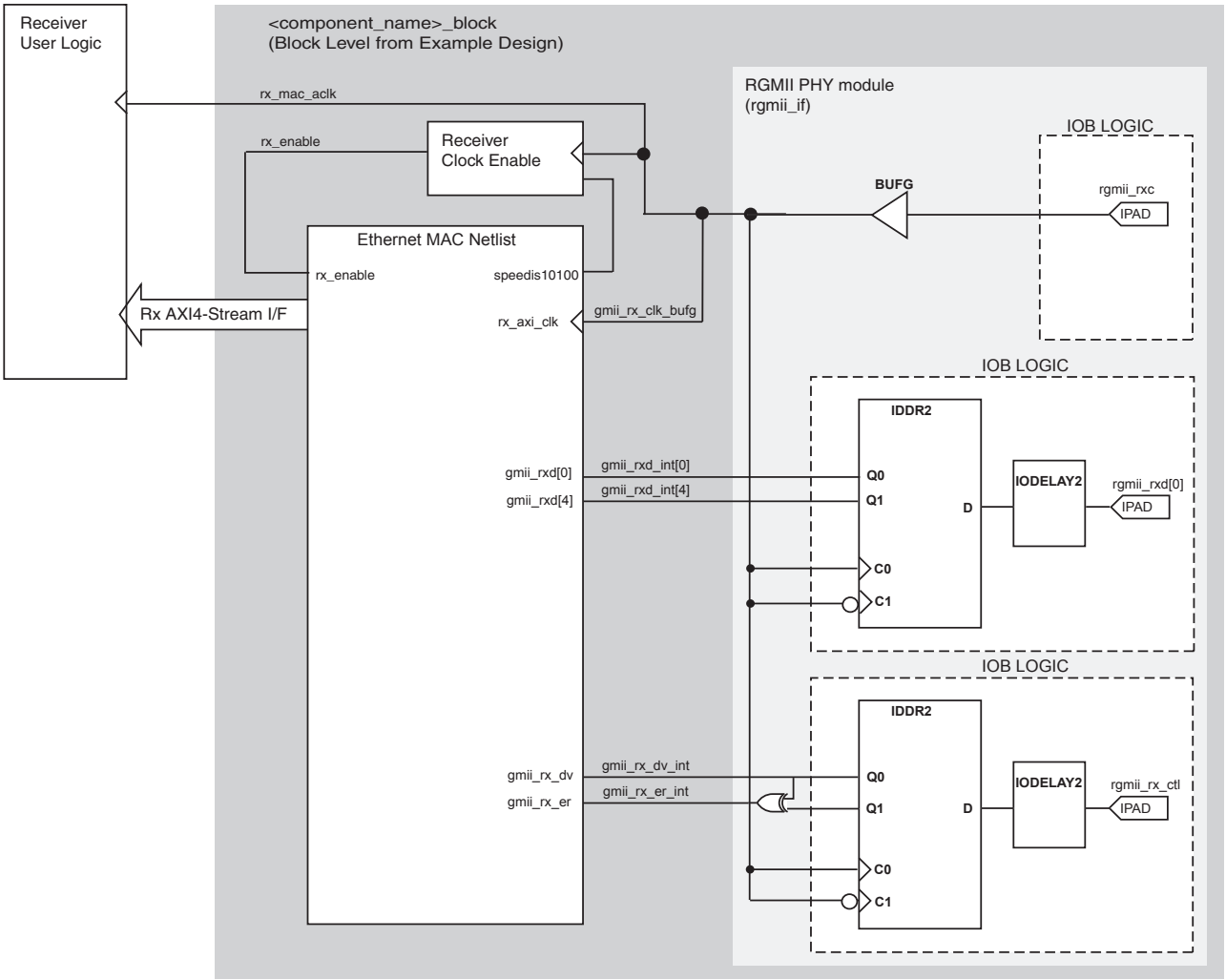

Figure 13-12: **RGMII Receiver Logic and Clock Logic for Spartan-6 Devices**

Clock Resource Sharing

Figure 13-13 illustrates clock resource sharing across multiple instantiations of the core when using RGMII in Spartan-6 devices. For all instantiations, `gtx_clk` can be shared between multiple cores, resulting in a common clock domain across the device.

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

Figure 13-13 illustrates 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.

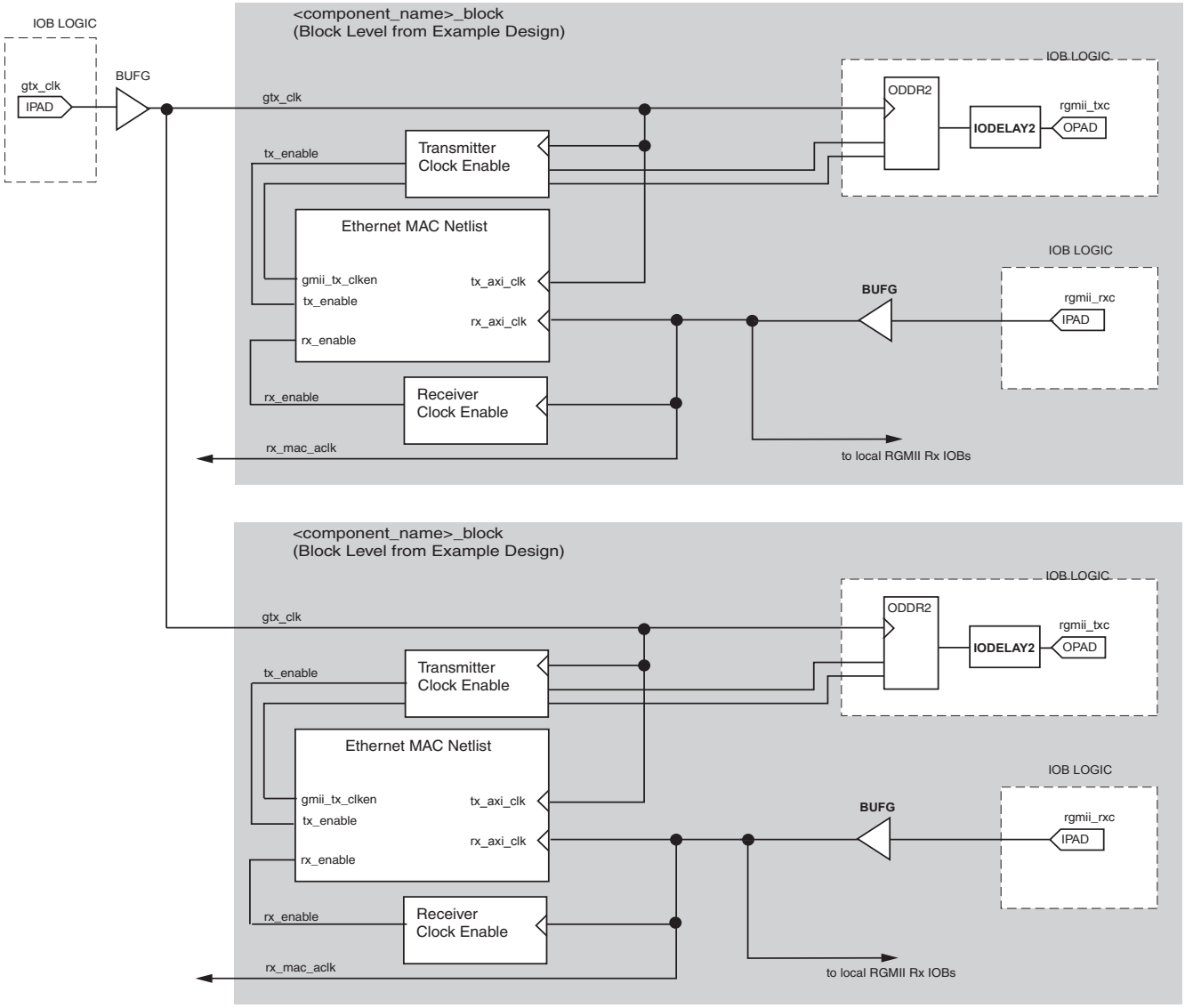

Figure 13-13: **Clock Resource Sharing for 1 Gb/s RGMII in Spartan-6 Devices**

XILINX®

Chapter 14

Constraining the Core

This chapter defines the constraint requirements of the TEMAC solution. An example UCF is provided with the HDL example design to provide samples of constraint requirements for the design. See Chapter 17, Quick Start Example Design and Chapter 18, Detailed Example Design.

Device, Package, and Speed Grade Selection

The core can be implemented in Virtex®-7, Kintex™-7, Artix™-7, Spartan®-6 and Virtex-6 devices with these attributes:

- Large enough to accommodate the core
- Contains a sufficient number of IOBs
- Device has a supported speed grade:

Table 14-1: **Supported Speed Grades**

Device Family	Speed Grade
Virtex-7	-1 or faster
Kintex-7	-1 or faster
Artix-7	-1 or faster
Virtex-6	-1 or faster
Virtex-6 Lower Power	-1L or faster
Spartan-6	-2 or faster

Note: Check for additional restrictions and limitations in [Ref 2].

I/O Location Constraints

For Virtex-7, Kintex-7 and Artix-7 FPGAs there are two types of I/O available, though both are not supported in most parts: HR I/O support MII, GMII or RGMII at 2.5 V whereas HP I/O only supports 1.8 V or lower. Despite this being the defined RGMII voltage most PHYs require 2.5 V and therefore an external voltage converter is required to interoperate with any multi-standard PHY for MII, GMII and RGMII. Whilst there are no specific I/O location constraints required for the Tri-Mode Ethernet MAC care must be taken to understand the type of I/O being targeted and the board implications.

For Spartan-6 and Virtex-6 devices no specific I/O location constraints are required. However, ensure that you use the correct type of dedicated clock input pins for clock

inputs and to abide by the device bank rules when using SelectIO™ interface logic at different voltage standards.

Additionally, when employing BUFIO, BUFIO2 and BUFR regional clock routing, ensure that a BUFIO/BUFIO2 capable clock input pin is selected for input clock sources, and that all related input synchronous data signals are placed in the respective BUFIO/BUFIO2 region. The device User Guides should be consulted.

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.

Placement Constraints

The example designs delivered by the CORE Generator™ software to accompany the core netlist can contain placement information for the global clock buffers and Mixed-Mode Clock Managers (MMCM). These are provided as an example only and can be removed or 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 precedent.

PERIOD(s) for Clock Nets

Transmitter Clock Constrains

Depending on the selected device family, 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.

Transmitter clock period constraints are always provided after the following comment heading in the UCF:

```
############################################################
# TX Clock period Constraints                              #
############################################################
```

The following syntax provides an example. This is taken from a tri-speed capable Virtex-7 FPGA design using GMII:

```
# Transmitter clock period constraints: do not relax
NET "clk_in_p" TNM_NET = "clk_in_p";
TIMESPEC "TS_clk_in_p" = PERIOD "clk_in_p" 5 ns HIGH 50% INPUT_JITTER 50.0ps;

NET "gtx_clk_bufg" TNM_NET = "clk_gtx";
TIMESPEC "TS_gtx_clk"      = PERIOD "clk_gtx" 8000 ps HIGH 50 %;
```

```
NET "*tx_mac_aclk*"              TNM_NET  = "clk_tx_mac";
TIMESPEC "TS_tx_clk_gmii"  = PERIOD "clk_tx_mac" 8000 ps HIGH 50 %;
```

Receiver Clock Constraints

Depending on the selected device family, 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.

Receiver clock period constraints are always provided after the following comment heading in the UCF:

```
############################################################
# RX Clock period Constraints                              #
############################################################
```

The following syntax provides an example. This is taken from a tri-speed capable Virtex-7 FPGA design using GMII:

```
# Receiver clock period constraints: do not relax
NET "gmii_rx_clk"          TNM_NET  = "clk_rx";
TIMESPEC "TS_rx_clk"       = PERIOD "clk_rx" 8000 ps HIGH 50 %;
```

IDELAYCTRL Reference Clock Constraints

For Virtex-7, Kintex-7 and Virtex-6 devices, an additional constraint my be required in the UCF for the IDELAYCTRL reference clock. This is not required in the generated example design as the relevant clock constraint is inherited from the MMCM. This clock is constrained to run at 200 MHz. See the device *User Guide* 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 NOT required in the generated example design as it is inherited from the MMCM as *TS_clock_generator_clkout1*. This is set to 100 MHz in the example design.

```
NET "*s_axi_aclk TNM_NET    = "clk_axi";
TIMESPEC "TS_axi_clk" = PERIOD "clk_axi" 8000 ps HIGH 50 % PRIORITY 10;
```

MDIO Logic

The MDIO logic is synchronous to `s_axi_aclk`, but data only changes at the MDC output rate:; nominally this is set to a frequency of 2.5 MHz. Every flip-flop in the MDIO logic is clocked with `s_axi_aclk`, but is sent a clock enable pulse at the MDC frequency. To prevent this logic being over constrained by the `s_axi_aclk` period, the relevant flip-flops for the MDIO logic can be grouped together and constrained at a multiple of the `s_axi_aclk` period. The Priority setting ensures this constraint takes precedence over the standard `s_axi_aclk` period constraint (which is given a priority of 10).

The UCF syntax which follows targets the MDIO logic flip-flops and groups them together. Reduced clock period constraints are then applied.

```
############################################################
# MDIO Constraints: do not edit                            #
############################################################
# Place the MDIO logic in its own timing groups
INST "*trimac_core*MANIFGEN?MANAGEN?PHY?ENABLE_REG"    TNM =
"mdio_logic";
INST "*trimac_core*MANIFGEN?MANAGEN?PHY?READY_INT"     TNM =
"mdio_logic";
INST "*trimac_core*MANIFGEN?MANAGEN?PHY?STATE_COUNT*"  TNM = FFS
"mdio_logic";
INST "*trimac_core*MANIFGEN?MANAGEN?PHY?MDIO_TRISTATE" TNM =
"mdio_logic";
INST "*trimac_core*MANIFGEN?MANAGEN?PHY?MDIO_OUT"      TNM =
"mdio_logic";
TIMESPEC "TS_mdio" = PERIOD "mdio_logic" "TS_clock_generator_clkout1" *
40 PRIORITY 0;
```

Timespecs for Critical Logic

Signals must cross clock domains at certain points in the core. These are described in the following section.

Flow Control

Pause requests are received and decoded in the receiver clock domain and must be transferred into the transmitter domain to pause the transmitter. Therefore, the following constraints must always be applied:

```
# Flow Control logic reclocking - control signal is synchronized
INST "*trimac_core?BU2?U0?FLOW?RX_PAUSE?PAUSE_REQ_TO_TX" TNM="flow_rx_to_tx";
INST "*trimac_core?BU2?U0?FLOW?RX_PAUSE?PAUSE_VALUE_TO_TX*"
TNM="flow_rx_to_tx";
TIMESPEC "TS_flow_rx_to_tx" = FROM "flow_rx_to_tx" TO "clk_tx_gmii" 8000 ps
DATAPATHONLY;
```

Configuration Logic

When the optional Management Interface is used with the core (see The Management Interface, page 77), configuration information is written synchronously to `s_axi_aclk`. Receiver configuration data must be transferred onto the receiver clock domain for use with the receiver; transmitter configuration data must be transferred onto the transmitter clock domain for use with the transmitter. The following UCF syntax targets this logic and a timing ignore attribute (TIG) is applied. It does not matter when configuration changes take place; the latest configurations are sampled at the beginning of new frames by both the receiver and transmitter.

Note: clock_generator_clkout1 is the auto-defined group generated by the MMCM and covers the s_axi_aclk domain.

```
TIMEGRP "ffs_except_axi" = FFS EXCEPT "clock_generator_clkout1" "mdio_logic";
TIMESPEC "TS_config_to_all" = FROM "clock_generator_clkout1" TO
"ffs_except_axi" TIG;
```

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 Physical Interface chapters within 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*"     IOB = true;
INST "*gmii_tx_en"    IOB = true;
INST "*gmii_tx_er"    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 2.5 V and the device default SelectIO™ interface standard of LVCMOS25 is used. In Spartan-6 devices, GMII/MII by default is supported at 3.3 V and the UCF can be updated to contain the following syntax, the targeted demonstration platforms use 2.5 V PHY devices and the UCF therefore specifies LVCMOS25. Use this syntax together with the device I/O Banking rules. for Virtex-7 and Kintex-7 devices see the relevant FPGA data sheet [Ref 4].

```
INST "gmii_txd<?>"      IOSTANDARD = LVTTL;
INST "gmii_tx_en"       IOSTANDARD = LVTTL;
INST "gmii_tx_er"       IOSTANDARD = LVTTL;

INST "gmii_rxd<?>"      IOSTANDARD = LVTTL;
INST "gmii_rx_dv"       IOSTANDARD = LVTTL;
INST "gmii_rx_er"       IOSTANDARD = LVTTL;

INST "gmii_tx_clk"      IOSTANDARD = LVTTL;
INST "gmii_rx_clk"      IOSTANDARD = LVTTL;
```

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.

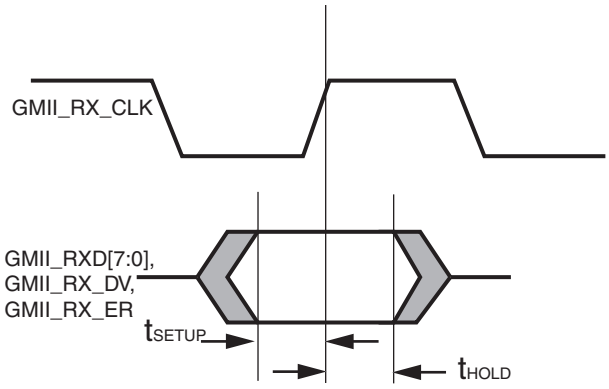

Figure 14-1: **Input GMII Timing**

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

7 Series and Virtex-6 Devices

In 7 Series Devices there are two types of I/O available, High Performance (HP) and High Range (HR). The HR I/O are perfectly suited to use for GMII as they support both the required frequency and the required voltage. However, HR I/O are only available on a subset of Virtex-7 devices, most Kintex-7 devices and all Artix-7 devices and are therefore not guaranteed to be available. HP I/O, however, are available on all Virtex-7 and Kintex-7 devices but are limited to 1.8V or lower operation. Therefore if HP I/O is used an external voltage converter is required on all GMII I/O (as well as any other PHY related signals).

The GMII example design uses BUFIO/BUFR routing on the clock and IODELAY components on the receiver data and control signals for 7 Series and Virtex-6 devices. 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 = 26;
```

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 all related IODELAY components and the related IDELAYCTRL are placed into a common IODELAY_GROUP. See the *Virtex-6 FPGA User Guide* and code comments for details.

In addition, for all 7 Series and Virtex-6 family designs, the following UCF syntax is included:

```
############################################################
# For Setup and Hold time analysis on GMII inputs          #
############################################################

# Identify GMII RX Pads only.
# This prevents setup/hold analysis being performed on false inputs,
# for example, the configuration_vector inputs.
INST "gmii_rxd<?>"                    TNM = IN_GMII;
INST "gmii_rx_er"                     TNM = IN_GMII;
INST "gmii_rx_dv"                     TNM = IN_GMII;
```

```
TIMEGRP "IN_GMII" OFFSET = IN  2 ns VALID 2 ns BEFORE "gmii_rx_clk";
```

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.

Spartan-6 Devices

The GMII example design uses BUFIO2/BUFG routing on the clock and IODELAY2 components on the receiver data and control signals for Spartan-6 devices. 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 IODELAY2 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 = 26;
```

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.

In addition, for all Spartan-6 FPGA designs, the following UCF syntax is included:

```
############################################################
# For Setup and Hold time analysis on GMII inputs          #
############################################################

# Identify GMII RX Pads only.
# This prevents setup/hold analysis being performed on false inputs,
# for example, the configuration_vector inputs.
INST "gmii_rxd<?>"                    TNM = IN_GMII;
INST "gmii_rx_er"                     TNM = IN_GMII;
INST "gmii_rx_dv"                     TNM = IN_GMII;

TIMEGRP "IN_GMII" OFFSET = IN  2 ns VALID 2 ns BEFORE "gmii_rx_clk";
```

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)|rx_gmii_mii_clk_int|   0.000|
gmii_rx_er  |    1.770(R)|   -0.226(R)|rx_gmii_mii_clk_int|   0.000|
gmii_rxd<0> |    1.821(R)|   -0.283(R)|rx_gmii_mii_clk_int|   0.000|
gmii_rxd<1> |    1.833(R)|   -0.295(R)|rx_gmii_mii_clk_int|   0.000|
gmii_rxd<2> |    1.790(R)|   -0.253(R)|rx_gmii_mii_clk_int|   0.000|
gmii_rxd<3> |    1.789(R)|   -0.252(R)|rx_gmii_mii_clk_int|   0.000|
gmii_rxd<4> |    1.834(R)|   -0.296(R)|rx_gmii_mii_clk_int|   0.000|
gmii_rxd<5> |    1.829(R)|   -0.291(R)|rx_gmii_mii_clk_int|   0.000|
gmii_rxd<6> |    1.793(R)|   -0.255(R)|rx_gmii_mii_clk_int|   0.000|
gmii_rxd<7> |    1.835(R)|   -0.296(R)|rx_gmii_mii_clk_int|   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 Physical Interface chapters within this Guide.

RGMII IOB Constraints

All Families

The RGMII v2.0 is a 1.5 V signal-level interface. The 1.5 V HSTL (High-Speed Transistor Logic) 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 is set to fast to ensure that the interface can meet setup and hold times.

Note: The targeted demonstration platforms use PHY devices which require 2.5 V, the UCF therefore sets the IOSTANDARD to LVCMOS25.

```
INST "rgmii_txd<?>"  IOSTANDARD = HSTL_I;
INST "rgmii_tx_ctl"  IOSTANDARD = HSTL_I;
INST "rgmii_rxd<?>"  IOSTANDARD = HSTL_I;
INST "rgmii_rx_ctl"  IOSTANDARD = HSTL_I;

INST "rgmii_txc"     IOSTANDARD = HSTL_I;
INST "rgmii_rxc"     IOSTANDARD = HSTL_I;

INST "rgmii_txd<?>   SLEW = FAST;
INST "rgmii_tx_ctl"  SLEW = FAST;
INST "rgmii_txc"     SLEW = FAST;
```

In addition, the example design provides pad locking on the RGMII for several families. 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.

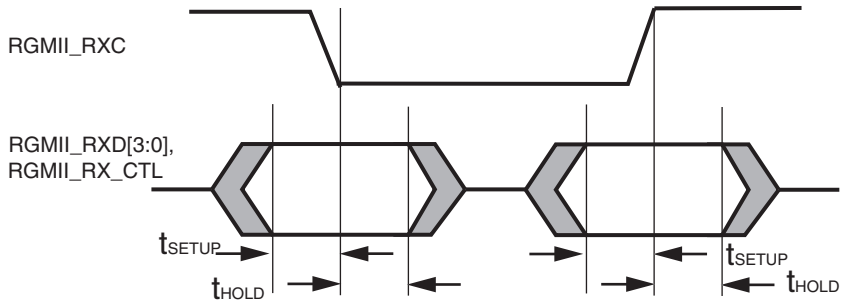

Figure 14-2: **Input RGMII Timing**

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`.

7 Series Devices

RGMII requires an offset between the transmit clock and data. In Spartan-6 and Virtex-6 devices this is achieved using a built in output delay. In 7 Series devices the output delay component is only available on High Performance (HP) I/O which are limited to operating at 1.8V or lower. Despite RGMII being defined as a 1.8V standard, the majority of PHYs supporting it are multi-standard and require it to run at 2.5V.

To be able to run at 2.5V we are limited to either using High Range (HR) I/O, which are only available on a subset of Virtex-7 devices, most Kintex-7 devices and all Artix-7 devices, or using an off-chip voltage converter. 7 Series HR I/O do not have the output delay functionality and therefore require different logic to implement the required transmit clock/data offset. See the appropriate section depending upon the type of I/O used.

7 Series Devices using HR I/O

The RGMII design requires a 90º phase shifted version off `gtx_clk` to be available. In the provided example design this is an output from the Clock Generator.

The 90º phase shifted clock, `gtx_clk90`, is used to generate the transmit clock, with the normal clock, `gtx_clk`, being used for the data and control generation.

The RGMII receiver design uses BUFIO/BUFR routing on the clock and IDELAY 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 IDELAY components. Data/Control bits can be adjusted individually, if desired, to compensate for any PCB routing skew.

```
INST *gmii_interface/delay_rgmii_rx_ctl IDELAY_VALUE = 20;
```

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 IDELAY 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 components are placed. To aid the tools in this all related IDELAY components and the related IDELAYCTRL are placed into a common IODELAY_GROUP. See the code comments for details.

In addition, for all designs, the following UCF syntax is included:

```
############################################################
# For Setup and Hold time analysis on RGMII inputs         #
############################################################

# Identify RGMII RX Pads only.
# This prevents setup/hold analysis being performed on false inputs,
# for example, the configuration_vector inputs.
INST "rgmii_rxd<?>"                    TNM = IN_RGMII;
INST "rgmii_rx_ctl"                    TNM = IN_RGMII;

# Define data valid window with respect to the clock rising edge.
# The spec states that, worst case, the data is valid 1 ns before the
clock edge.
# The worst case it to provide 1 ns hold time (a 2ns window in total)
TIMEGRP "IN_RGMII" OFFSET = IN 1 ns VALID 2 ns BEFORE "rgmii_rxc"
"RISING";

# Define data valid window with respect to the clock falling edge.
TIMEGRP "IN_RGMII" 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.

7 Series using HP I/O and Virtex-6 Devices

Note: HP I/O only supports operation at 1.8V or lower and this either requires an external voltage converter for 2.5V PHYs or a dedicated RGMII PHY supporting 1.8V.

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. The following UCF syntax is an example:

```
INST "*delay_rgmii_tx_clk" ODELAY_VALUE = 26;
```

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 *gmii_interface/delay_rgmii_rx_ctl IDELAY_VALUE = 20;
```

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 in this all related IODELAY components and the related IDELAYCTRL are placed into a common IODELAY_GROUP. See the *Virtex-6 FPGA User Guide* and code comments for details.

In addition, for all 7 Series and Virtex-6 family designs, the following UCF syntax is included:

```
##################################################################
# For Setup and Hold time analysis on RGMII inputs         #
##################################################################

# Identify RGMII RX Pads only.
# This prevents setup/hold analysis being performed on false inputs,
# for example, the configuration_vector inputs.
INST "rgmii_rxd<?>"                    TNM = IN_RGMII;
INST "rgmii_rx_ctl"                    TNM = IN_RGMII;

# Define data valid window with respect to the clock rising edge.
# The spec states that, worst case, the data is valid 1 ns before the
clock edge.
# The worst case it to provide 1 ns hold time (a 2ns window in total)
TIMEGRP "IN_RGMII" OFFSET = IN 1 ns VALID 2 ns BEFORE "rgmii_rxc"
"RISING";

# Define data valid window with respect to the clock falling edge.
TIMEGRP "IN_RGMII" 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.

Spartan-6 Devices

The RGMII design uses an IODELAY2 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 "*delay_rgmii_tx_clk" ODELAY_VALUE = 26;
```

The RGMII receiver design uses direct BUFG routing on the clock with IODELAY2 components on the control and datapaths. A fixed tap delay is applied to move the control/data in relation to the clock to provide the maximum setup/hold for the interface at the data/control IOB IDDR2 registers.

The following constraint shows an example of setting the IODELAY2 tap delay. The ideal value for this is found through hardware testing.

```
INST "*delay_rgmii_rxd*"             IDELAY_VALUE = 8;
```

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.

In addition, for all Spartan-6 FPGA designs, the following UCF syntax is included:

```
############################################################
# For Setup and Hold time analysis on RGMII inputs         #
############################################################

# Identify RGMII RX Pads only.
# This prevents setup/hold analysis being performed on false inputs,
# for example, the configuration_vector inputs.
INST "rgmii_rxd<?>"                    TNM = IN_RGMII;
INST "rgmii_rx_ctl"                    TNM = IN_RGMII;

# Define data valid window with respect to the clock rising edge.
# The spec states that, worst case, the data is valid 1 ns before the
clock edge.
# The worst case it to provide 1 ns hold time (a 2ns window in total)
TIMEGRP "IN_RGMII" OFFSET = IN 1 ns VALID 2 ns BEFORE "rgmii_rxc"
"RISING";

# Define data valid window with respect to the clock falling edge.
# TIMEGRP "IN_RGMII" 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 report. Each Input lists two sets of values: one corresponding to the falling edge of the clock and one to the rising edge.

Setup: the first set listed corresponds to falling edge, which occurs at time 4 ns. The implementation requires 0.648 ns of setup to the falling edge and 0.661 ns to the rising edge; this is less than the 1ns required and so there is slack.

Hold: the implementation requires 0.300 ns of hold after the falling edge and 0.316 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
------------+------------+------------+---------------------+--------+
            |  Setup to  |  Hold to   |                     | Clock  |
Source      | clk (edge) | clk (edge) |Internal Clock(s)    | Phase  |
------------+------------+------------+---------------------+--------+
rgmii_rx_ctl|   -3.352(R)|    4.300(R)|not_rgmii_rx_clk_bufg|   4.938|
            |    0.661(R)|    0.284(R)|rgmii_rx_clk_bufg    |   0.938|
rgmii_rxd<0>|   -3.384(R)|    4.332(R)|not_rgmii_rx_clk_bufg|   4.938|
            |    0.629(R)|    0.316(R)|rgmii_rx_clk_bufg    |   0.938|
rgmii_rxd<1>|   -3.348(R)|    4.296(R)|not_rgmii_rx_clk_bufg|   4.938|
            |    0.665(R)|    0.280(R)|rgmii_rx_clk_bufg    |   0.938|
rgmii_rxd<2>|   -3.360(R)|    4.308(R)|not_rgmii_rx_clk_bufg|   4.938|
            |    0.653(R)|    0.292(R)|rgmii_rx_clk_bufg    |   0.938|
rgmii_rxd<3>|   -3.428(R)|    4.382(R)|not_rgmii_rx_clk_bufg|   4.938|
            |    0.585(R)|    0.366(R)|rgmii_rx_clk_bufg    |   0.938|
------------+------------+------------+---------------------+--------+
```

XILINX®

Chapter 15

Interfacing to Other Xilinx Ethernet Cores

Ethernet 1000BASE-X PCS/PMA or SGMII Core

The Ethernet MAC core can be integrated in a single device with the Ethernet 1000BASE-X PCS/PMA or SGMII core to provide either:

- A MAC with an SGMII interface to an external PHY device. SGMII can support either tri-speed (10/100/1000 Mb/s) designs or 1 Gb/s only designs.
- A 1 Gb/s Ethernet MAC core with 1000BASE-X PCS/PMA sublayer functionality: this is a 1 Gb/s only PHY standard which is most commonly used for a fibre optic medium.

For more details on the Xilinx Ethernet 1000BASE-X PCS/PMA or SGMII core, see the product page at:

www.xilinx.com/products/ipcenter/DO-DI-GMIITO1GBSXPCS.htm

[Ref 3] provides the information required to connect the two cores together in any supported configuration.

XILINX®

Chapter 16

Implementing Your Design

This chapter describes how to simulate and implement your design containing the Ethernet MAC 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 [Ref 7], 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 details 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 details 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 Native Generic Database (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 (Configurable Logic Blocks) and IOBs of the FPGA, run the `map` command. The `map` command writes out a physical design to an Native Circuit Description (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 Physical Constraints File (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 Wizard Report (TWR) file 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 Verification Design Guide*, included in your Xilinx software installation.

Other Implementation Information

For details about using the Xilinx implementation tool flow, including command line switches and options, see the Xilinx ISE® software manuals.

XILINX®

Chapter 17

Quick Start Example Design

This chapter provides instructions for generating the TEMAC example design using the default parameters.

Overview

The TEMAC example design consists of the following:

- A TEMAC solution netlist
- An HDL example design
- A demonstration test bench to exercise the example design

Figure 17-1 shows the block diagram for the example design and the test bench provided with the TEMAC solution. The example design has been tested with Xilinx® ISE® software v13.4, Cadence Incisive Enterprise Simulator (IES), Mentor Graphics ModelSim, and Synopsys VCS and VCS MX(1).

1. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

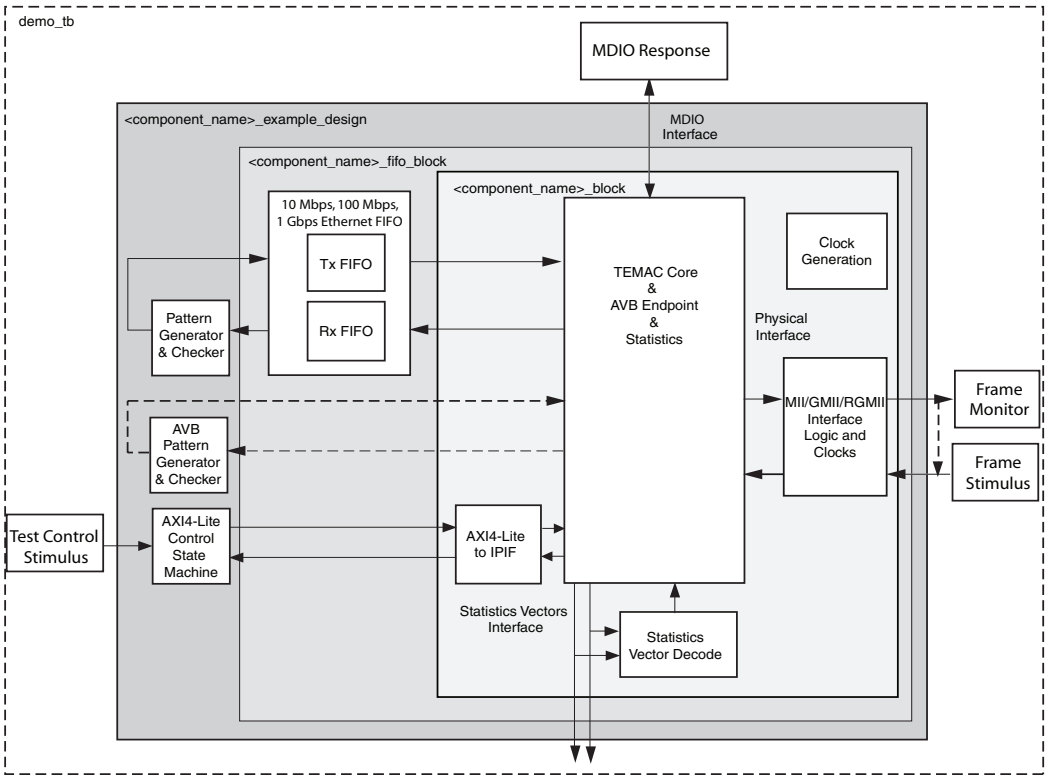

Figure 17-1: **Default Example Design and Test Bench**

Generating the Core

To begin using the TEMAC example design, start by generating the core.

To generate 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 [Ref 8].
2. Choose File > New Project.
3. Do the following to set project options:
   - From Target Architecture, select an FPGA family that supports the core, for example 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 [Ref 2].
   - 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. A warning can appear regarding the limitations of the simulation-only evaluation license.

6. Click OK to exit the dialog box. The core customization screen appears.

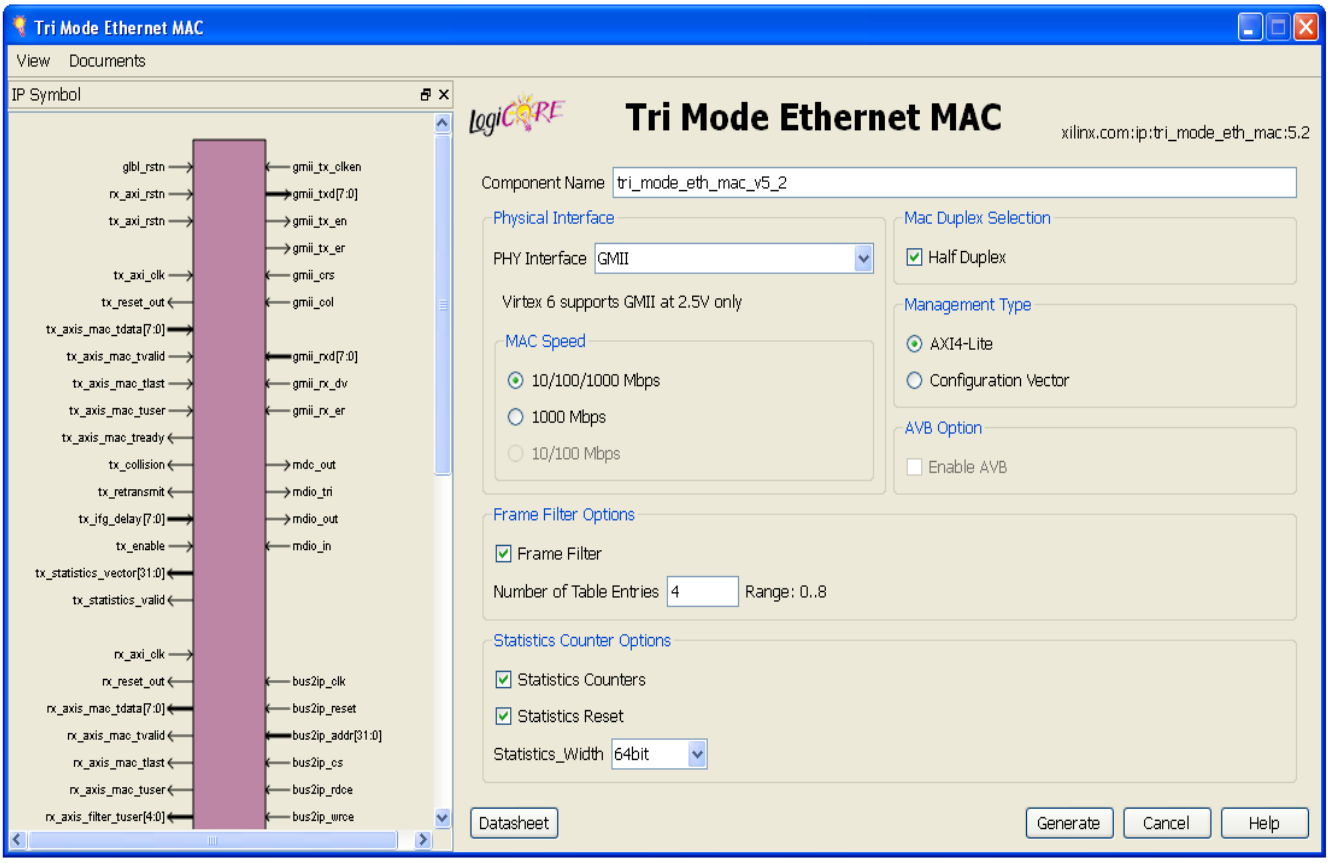

Figure 17-2: **Tri-Mode Ethernet MAC Main Screen**

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 18, 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, page 176.

Implementing the Example Design

If the core is generated with a Hardware Evaluation or a Full license, 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 *COMPXLIB* in [Ref 9].

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.

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. At this point, 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. At this point, you can review the simulation transcript and waveform to observe the operation of the core.

Timing Simulation

If the core is generated with a Hardware Evaluation or a Full license, then Timing Simulations are supported. Ensure that the example design has been implemented before attempting a Timing Simulation.

To run the gate-level simulation, you must have the Xilinx Simulation Libraries compiled for your system. See *COMPXLIB* in [Ref 9].

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. At this point, 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. At this point, you can review the simulation transcript and waveform to observe the operation of the core.

XILINX®

Chapter 18

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.

The example design, under certain core configurations, is intended to be directly targetable to key Xilinx Demonstration Platforms, the current supported boards being:

- Spartan®-6 FPGA boards
  - SP601 Board
  - SP605 Board
- Virtex®-6 FPGA boards
  - ML605 Board
- Kintex™-7 FPGA boards
  - KC705 Board
- Virtex®-7 and Artix™-7 FPGA Boards
  - No boards are supported at this time

The example design includes a basic state machine which, through the AXI4-Lite interface, brings up the external PHY and MAC to allow basic frame transfer.

A Simple Frame Generator and Frame Checker are also included which can be used to turn a particular board into a packet generator with any received data optionally being checked. If the TEMAC is generated with the Optional AVB Endpoint another frame generator and frame checker are included to exercise the AV datapath.

Loopback functionality is provided as either MAC RX to TX loopback, where the loopback logic becomes the packet source in place of the packet generator, or PHY TX to RX loopback, with the loopback replacing the demonstration test bench stimulus and checker. Basic control of the state machine, allowing 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, page 199

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/control
    Files for the Configuration vector 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

Directory and File Contents

The core directories and their associated files are defined in the following sections.

Note: The implement and timing simulation directories are only present when the core is generated with a Full System Hardware Evaluation license or a Full license.

<project directory>

The project directory contains all the CORE Generator software project files.

Table 18-1: **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 18-2: **Component Name Directory**

Name	Description
<project_dir>/<component_name>	
tri_mode_eth_mac_readme.txt	Core release notes file

Back to Top

<component name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 18-3: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
ds818_tri_mode_eth_mac.pdf	Tri-Mode Ethernet MAC Data Sheet
ug777_tri_mode_eth_mac.pdf	Tri-Mode Ethernet MAC 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 18-4 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, page 189 for details.

Table 18-4: **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 TEMAC wrapper together with a receive and a transmit FIFO.
<component_name>_block.v[hd]	Block-level TEMAC wrapper containing the core and all clocking and physical interface circuitry
<component_name>_mod.v	Verilog module declaration for the core instance in the example design
tx_clk_gen.v[hd]	Simple TX clock generation block
<component_name>_example_design.xdc	Not Used

Back to Top

example design/common

This directory contains common files required by various levels of the example design.

Table 18-5: **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 18-6: **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.
slace_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 18-7: **Axi_lite Directory**

Name	Description
<project_dir>/<component_name>/example_design/axi_lite	

Table 18-7: **Axi_lite Directory *(Cont'd)***

Name	Description
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/control

This directory contains the files for the config vector controller that is instanced in the example design when no management interface is selected.

Table 18-8: **Control Directory**

Name	Description
<project_dir>/<component_name>/example_design/control	
config_vector_sm.v[hd]	Simple state machine to drive the configuration vectors to allow 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 18-9: **FIFO Directory**

Name	Description
<project_dir>/<component_name>/example_design/fifo	
tx_client_fifo.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.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 18-10: **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

Table 18-10: **Pat_gen Directory *(Cont'd)***

Name	Description
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.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.
address_swap.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 18-11: **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

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 18-12: **Statistics Directory**

Name	Description
<project_dir>/<component_name>/example_design/statistics	

Table 18-12: **Statistics Directory *(Cont'd)***

Name	Description
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 is only available when the core is generated with a Hardware Evaluation or a Full license.

This directory contains the support files necessary for implementation of the example design with the Xilinx tools. See Example Design, page 189. Execution of an implement script results in creation of the results directory and an xst project directory.

Table 18-13: **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
example_design_xst.xcf	Constraints file automatically used by XST
planAhead_rdn.sh	Not Used
planAhead_rdn.bat	Not Used
planAhead_rdn.tcl	Not Used

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 18-14: **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 18-15: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
demo_tb.v[hd]	VHDL or Verilog demonstration test bench for the TEMAC solution

Back to Top

simulation/functional

The functional directory contains functional simulation scripts provided with the core.

Table 18-16: **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.

Table 18-16: **Functional Directory *(Cont'd)***

Name	Description
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

Implementation and Test Scripts

Implementation Scripts

When the CORE Generator software is run with a Full System Hardware license or a Full license, 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 functional simulation flow is available with any license type, and 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

Example Design

HDL Example Design

Figure 18-1 illustrates the top-level design for the TEMAC solution example design.

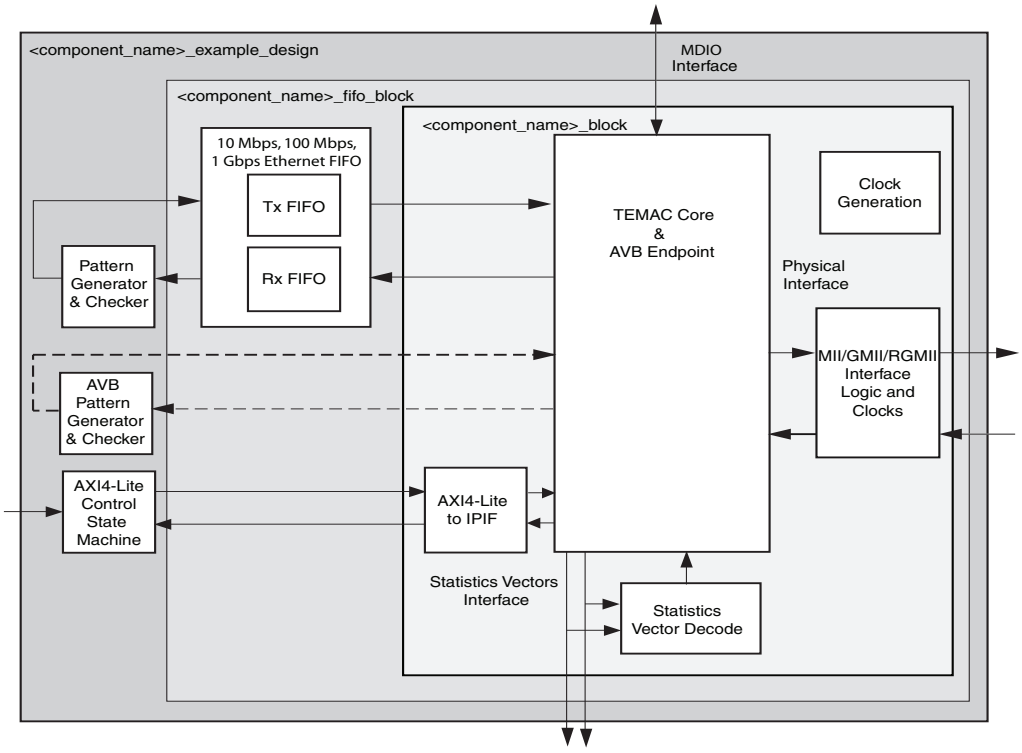

Figure 18-1: **HDL Example Design**

The top-level example design for the TEMAC solution 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 TEMAC solution
- Clock management logic, including MMCM and Global Clock Buffer instances, where required
- MII, GMII or RGMII 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 contains a frame generator and frame checker plus loopback logic.
- User AVB pattern generator module providing a second frame generator and frame checker for designs including the AVB Endpoint.
- 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 TEMAC solution and connects the GMII/RGMII interface to external IOBs, it can also operate as a pattern generator with data being optionally looped back externally, on the PHY side, and automatically checked.

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.v[hd
]
```

```
<project_dir>/<component_name>/example_design/fifo/rx_client_fifo.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 18-2 illustrates a straightforward frame transfer across the user side AXI4-Stream interface

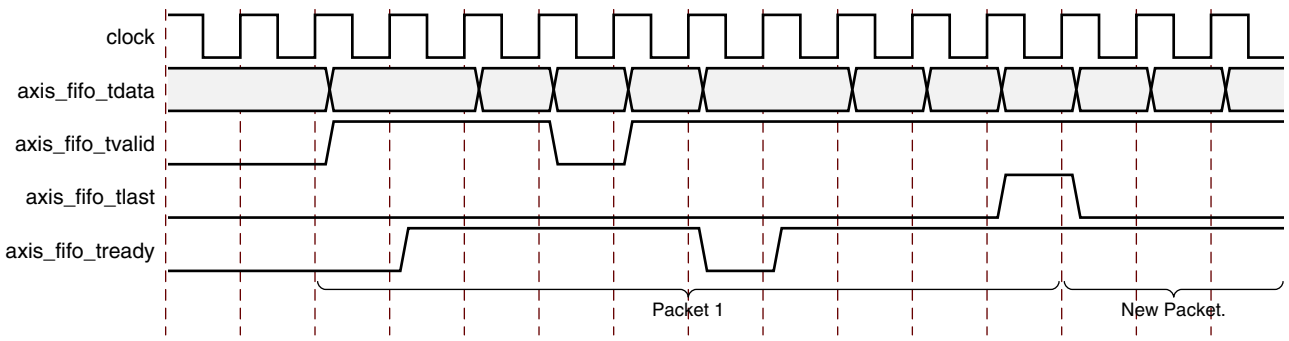

Figure 18-2: **Frame Transfer across AXI4-Stream Interface**

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 TEMAC core. If the frame is not errored, that frame is presented on the AXI4-Stream FIFO interface for reading by the user, (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 inter-packet 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 is then lost. It is therefore recommended that the example design is not used with the TEMAC solution 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.v[hd]
<project_dir>/<component_name>/example_design/pat_gen/axi_pat_check.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.v[hd]
```

The basic pattern generator has two main functional modes: loopback and generator. In loopback the data from the RX FIFO is passed to the address swap module and passed from there to the TX FIFO. In Generator mode the TX data is provided by the Pattern Generator, with RX Data being optionally checked by the pattern Checker.

Address Swap

The address swap module waits until both the DA and SA have been received before starting to send data on to the TX FIFO.

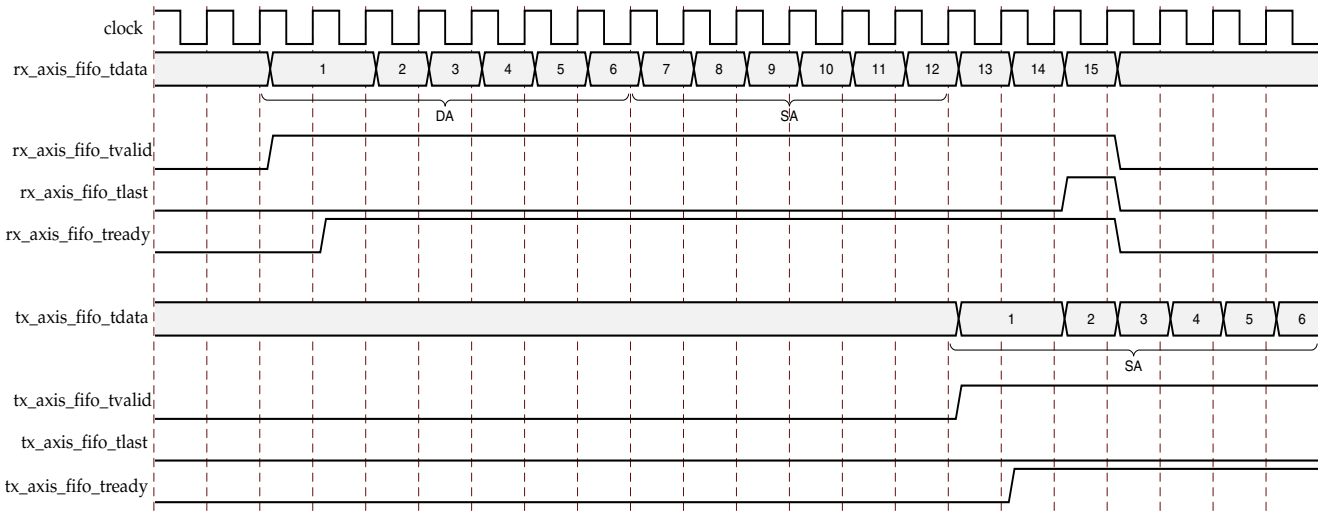

Figure 18-3: **Modification of Frame Data by Address Swap Module**

If enabled, the module swaps the destination and source addresses of each frame as shown in Figure 18-3 to ensure that the outgoing frame destination address matches the source address of the link partner, otherwise the DA and SA are left untouched. The module transmits the frame control signals with an equal latency to the frame data

Pattern Generator

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` modules inputs.

The pattern generator allows user modification of the Destination Address, Source Address, minimum frame size and maximum frame size using parameters. When enabled, using a dedicated input mapped to a DIP switch on a board, 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 18-4.

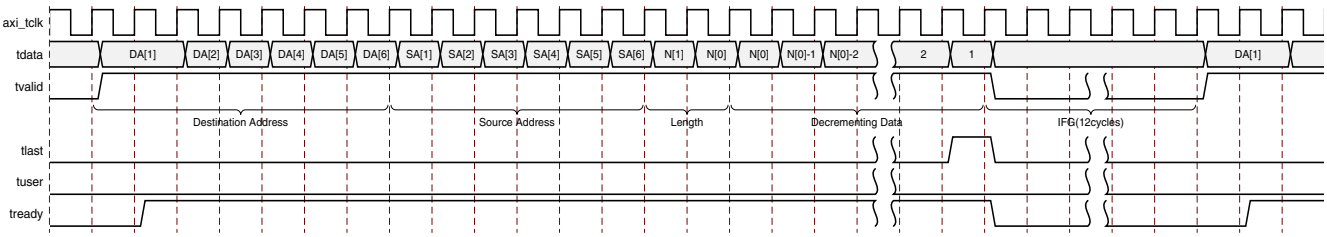

Figure 18-4: **Pattern Generator Frame Structure**

In a loopback scenario (using a second board as the loopback) 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. To avoid this issue the data rate provided by the pattern generator is throttled to just below the selected line rate.

Pattern Checker

The `axi_pat_check` module 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. Because the Frame data might or might not have the DA and SA swapped the pattern checker allows either value to be in either location.

When enabled, using a dedicated input which uses a board DIP switch, the output from the RX_FIFO is monitored. The first step is to identify where in the frame sequence the data is, this is done by capturing the value in the type/length field. After this is done the following frame is expected to be incrementally bigger (unless we happen to be 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 output to a dedicated output, this is displayed using a board LED. The pattern checker state machine then re-syncs to the data. A dedicated input, connected to one of the push buttons, is used to clear this latched error state, enabling a feel for the frequency of errors if any.

The pattern checker also provides a simple activity monitor. This toggles a dedicated output, which flashes a board LED, to indicate that RX Data is being received correctly. This ensures that the lack of a detected error is not just due to all frames being dropped.

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.

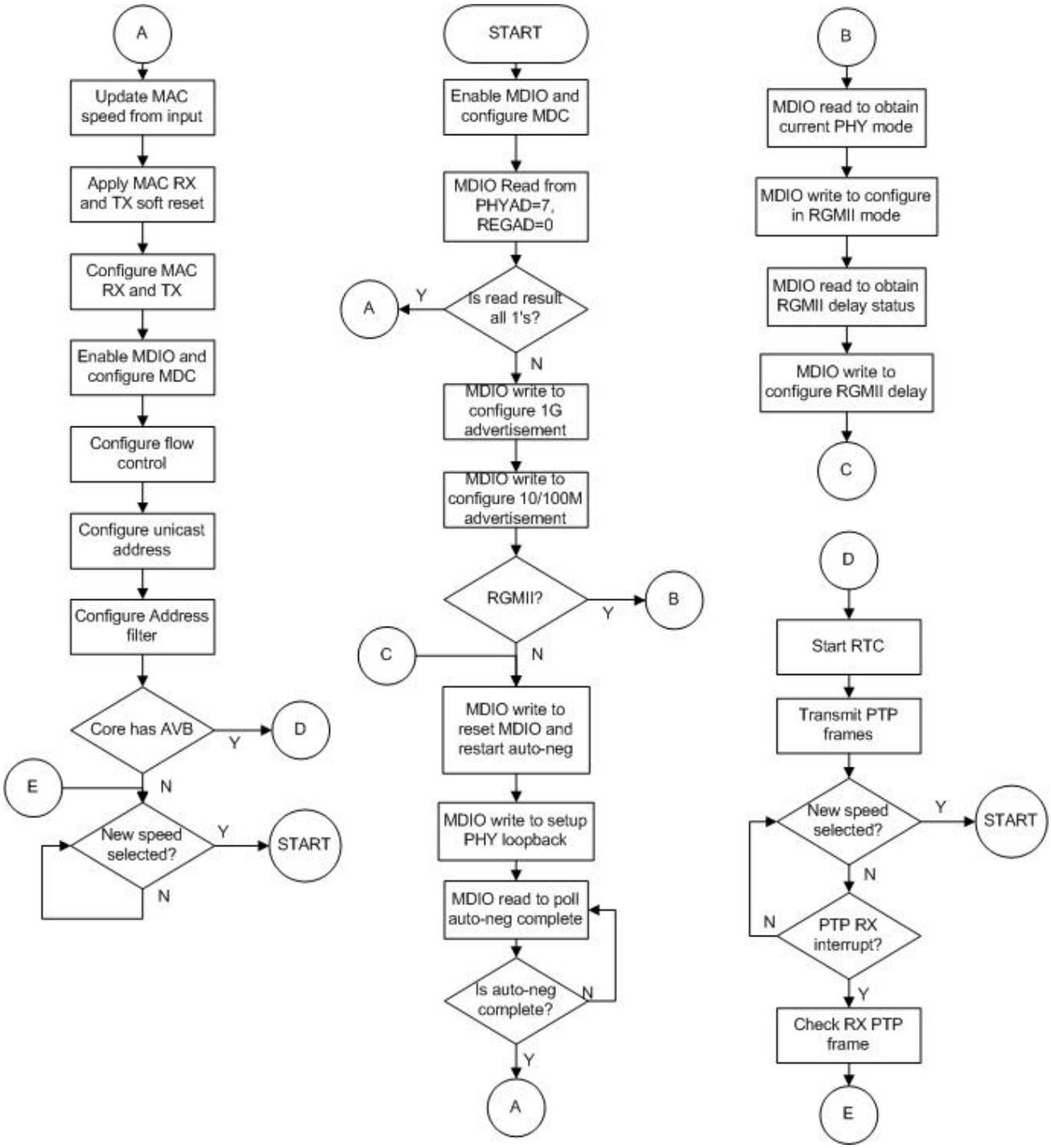

Figure 18-5: **State Machine Flow Diagram**

Figure 18-5 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 100 MHz). An MDIO read is then performed from PHYAD 7, which is the standard address used on Xilinx Demonstration Boards. If this returns all 1's 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 1's; 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 to select RGMII, avoiding the need to change jumper settings on the board. Finally the PHY is reset and auto-negotiation restarted. After auto-negotiation completes the MAC speed is updated, as per the `mac_speed` inputs. The MAC is then configured to disable flow control, initialise the unicast address and set the Frame Filter to promiscuous mode. If the AVB Endpoint logic is present then the RTC is started and PTP frames both transmitted and received (this assumes an external loopback is in use). 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

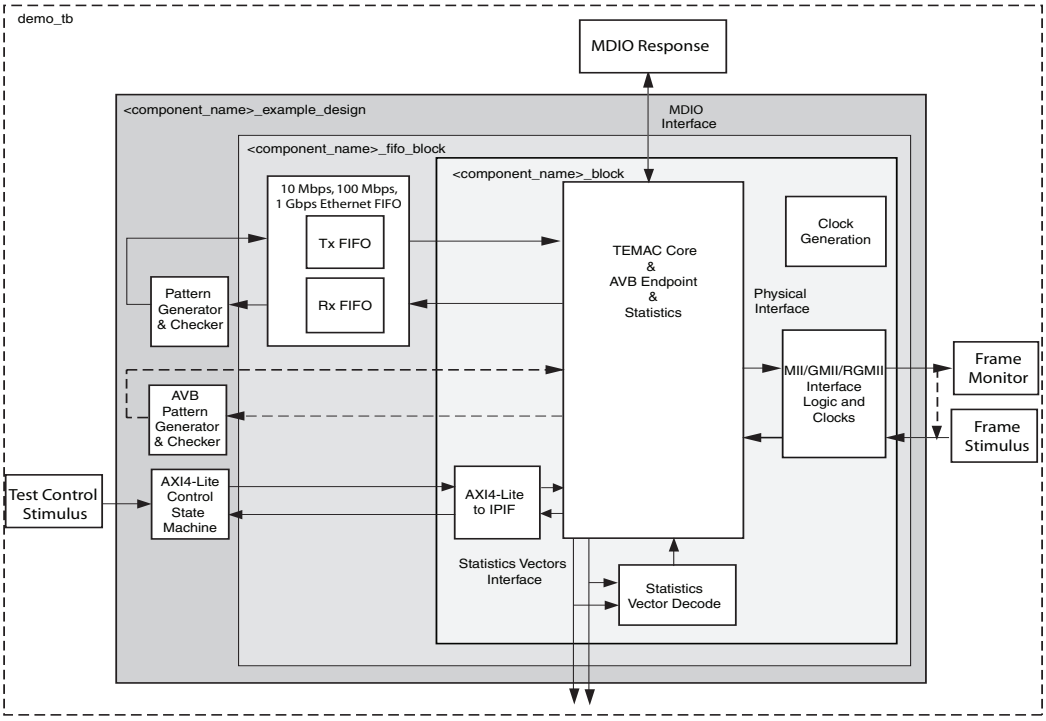

Figure 18-6: **Demonstration Test Bench**

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. It has two modes of operation, DEMO and Built-in Self Test (BIST), with BIST being the default mode when the AVB Endpoint is included. The mode can be changed using a parameter.

The test bench consists of the following:

- Clock generators
- DEMO: A stimulus block that connects to the GMII/MII or RGMII receiver interface of the example design
- DEMO: A monitor block to check data returned through the GMII/MII or RGMII transmitter interface
- BIST: A simple loopback from the GMII/MII or RGMII transmit interface to the receiver.
- BIST (AVB only): A basic AV data bandwidth monitor.
- A management block to control the speed selection
- An MDIO monitor/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

Core with Management Interface

DEMO mode

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The required speed is selected using mac_speed and update_speed
- The MDIO stimulus/response block responds to a read with all 1's - to indicate no PHY is present.
- Four frames are pushed into the GMII/MII or RGMII 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
- 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.
- If either the Tri-speed or MII 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.
- The MDIO stimulus/response block responds to a read with all 1's - to indicate no PHY is present.
- The same four frames are then sent to the MII/GMII or RGMII interface and checked against the stimulus frames.
- If the Tri-speed configuration has been selected, mac_speed is updated to run at 10 Mb/s. `update_speed` is then pulsed.
- The MDIO stimulus/response block responds to a read with all 1's - to indicate no PHY is present.
- The same four frames are then sent to the MII/GMII or RGMII interface and checked against the stimulus frames.
- For the Tri-speed configuration, the speed is then changed back to 1 Gb/s and the same four frames are sent and checked for a final time. This tests the speed switching between 1 Gb/s and 10/100 Mb/s in both directions.

BIST mode

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The required speed is selected using mac_speed and `update_speed`
- The MDIO stimulus/response block responds to a read with all 1's - to indicate no PHY is present.
- If the AVB Endpoint is included the AXI4-Lite state machine requests two TX PTP frames and checks they are received

- The pattern generator(s) and checker(s) are enabled
- The simulation runs for a fixed duration, allowing a large number of frames to pass.
- Any detected errors or lack of RX activity are reported as errors
- If the AVB Endpoint is included the bandwidth of the two data streams is reported.

Core with No Management Interface

DEMO Mode

The demonstration test bench performs the following tasks:

- Input clock signals are generated
- A reset is applied to the example design
- The required speed is selected using mac_speed and update_speed
- 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
- 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.
- If either the Tri-speed or MII 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.
- The same four frames are then sent to the MII/GMII or RGMII interface and checked against the stimulus frames.
- If the Tri-speed configuration has been selected, mac_speed is updated to run at 10 Mb/s. `update_speed` is then pulsed. The same four frames are then sent to the MII or RGMII interface and checked against the stimulus frames.
- For the Tri-speed configuration, the speed is then changed back to 1 Gb/s and the same four frames are sent and checked for a final time. This tests the speed switching between 1 Gb/s and 10/100 Mb/s in both directions.

BIST mode

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The required speed is selected using `mac_speed` and `update_speed`
- The pattern generator and checker are enabled
- The simulation runs for a fixed duration, allowing a large number of frames to pass.
- Any detected errors or lack of RX activity are reported as errors

Changing the Test Bench

The Demonstration test bench defaults to DEMO mode for all implementations which do not include the AVB Endpoint, when it is included the default is BIST mode.

The mode is set using a parameter in the `demo_tb.v[hd]` with the alternative mode being commented out. To change mode, uncomment the desired mode and either remove or comment the undesired one.

DEMO Mode

Changing Frame Data

The contents of the frame data passed into the TEMAC 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 GEMAC, as well as calculating the new expected FCS value. Further frames can be added by defining a new frame of data.

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.

BIST Mode

In BIST mode the data is provided by the Basic Pattern Generator Module. This allows a degree of control over the frames generated using the module parameters:

```
DEST_ADDR
SRC_ADDR
MAX_SIZE
MIN_SIZE
ENABLE_VLAN
VLAN_ID
VLAN_PRIORITY
```

The pattern generator does not have an error injection capability.

Targeting the Example Design to a Board

For each supported device, there are certain TEMAC solution configurations which can be targeted directly to the Xilinx connectivity board for that device. The UCF included with the example design provides the required pin placements for the specific board. In each case the board DIP switches, push buttons and LEDs are used to provide basic control over the MAC functionality. This is described in more detail in the board specific sections.

TEMAC Solution Configurations Supported

There are some basic requirements for the example design to function correctly when targeted at a board. The TEMAC must:

- Include an AXI4_Lite Management interface
- Target the relevant part for the specific board - see the board specific section.

Bring Up Sequence

When the example design is first targeted at a board the following sequence is suggested to check the various features are working, this is common for all boards:

- First Attach an Ethernet cable between the board and a PC installed with wireshark or similar.
- Select the desired speed using the DIP switches
- Push the update speed pushbutton
- Ensure the link status LEDs show 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.

To utilise the pattern checker and check the both datapaths two boards are required.

Board A: Operates as a frame source and optionally checker.

Board B: This board operates as a simple loopback board.

Bring up process:

- First attach an Ethernet cable between the two boards.
- Select the desired speed on both boards - this must be the same setting
- Push the update speed button on both boards
- Check the link status LEDs show the correct speed
- Enable the pattern generator on Board A, ensure it is disabled on Board B
- Check the Link Status RX/TX LEDs all light up
- If desired the Pattern checker can be enabled on both boards or just Board A.
- Ensure the RX activity LED is flashing

SP601 Board

The UCF targets the SP601 when any Spartan-6 device apart from the 45T is used.

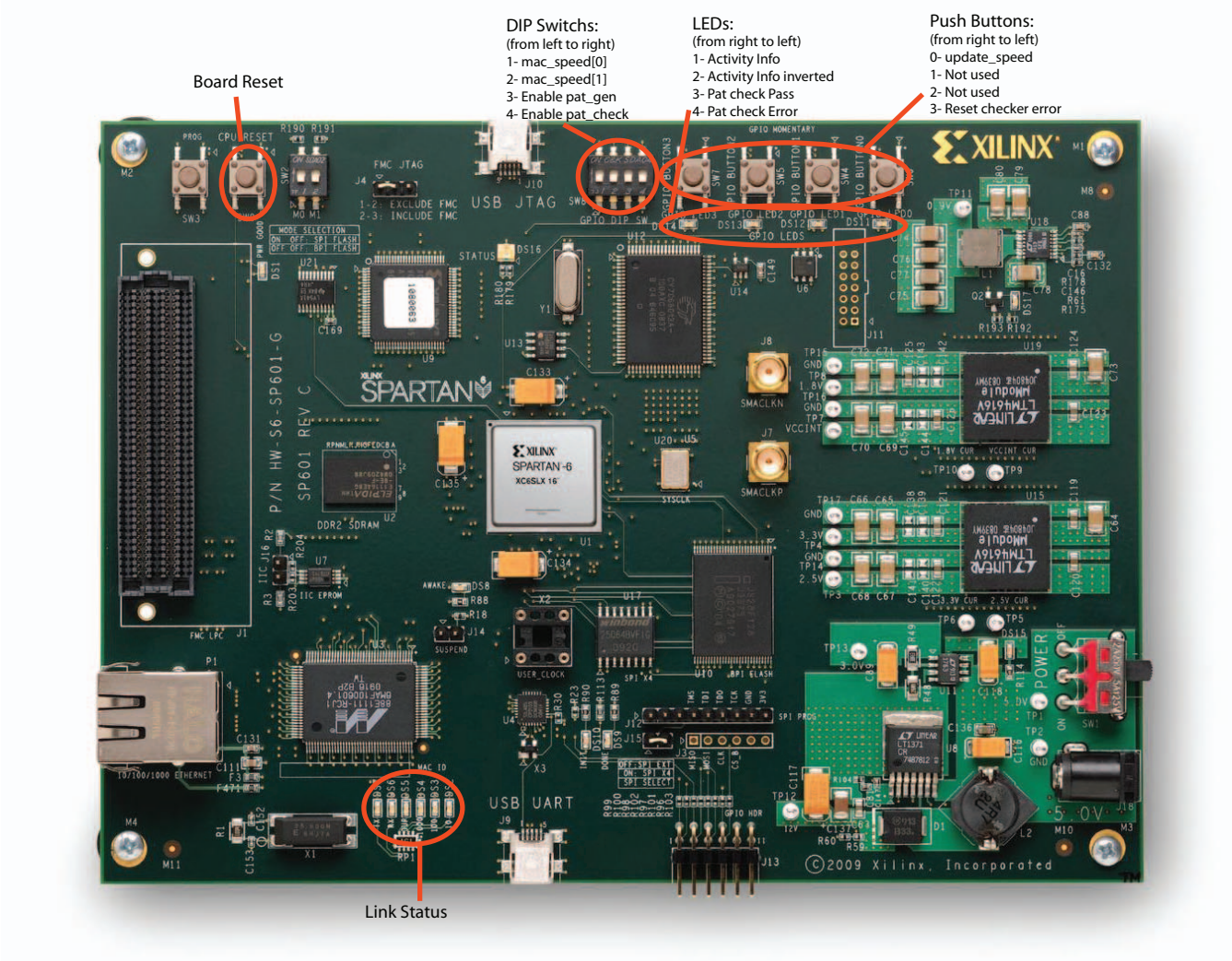

Figure 18-6: **SP601 Board Connectivity**

SP605 Board

The UCF targets this board if the Spartan-6 45T part is selected.

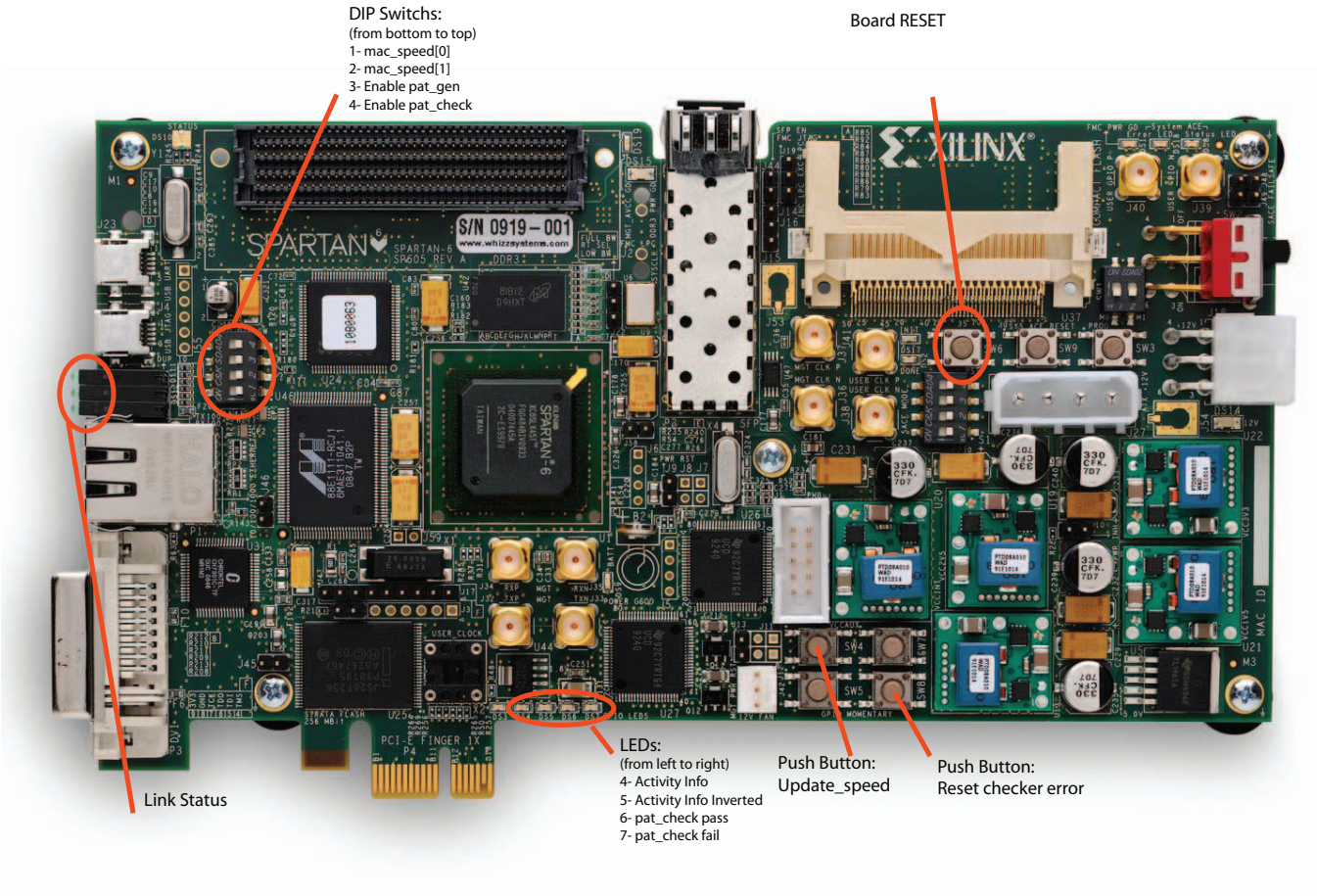

Figure 18-7: **SP605 Board Connectivity**

ML605 Board

The UCF targets the ML605 when any Virtex-6 LX FPGA part is selected.

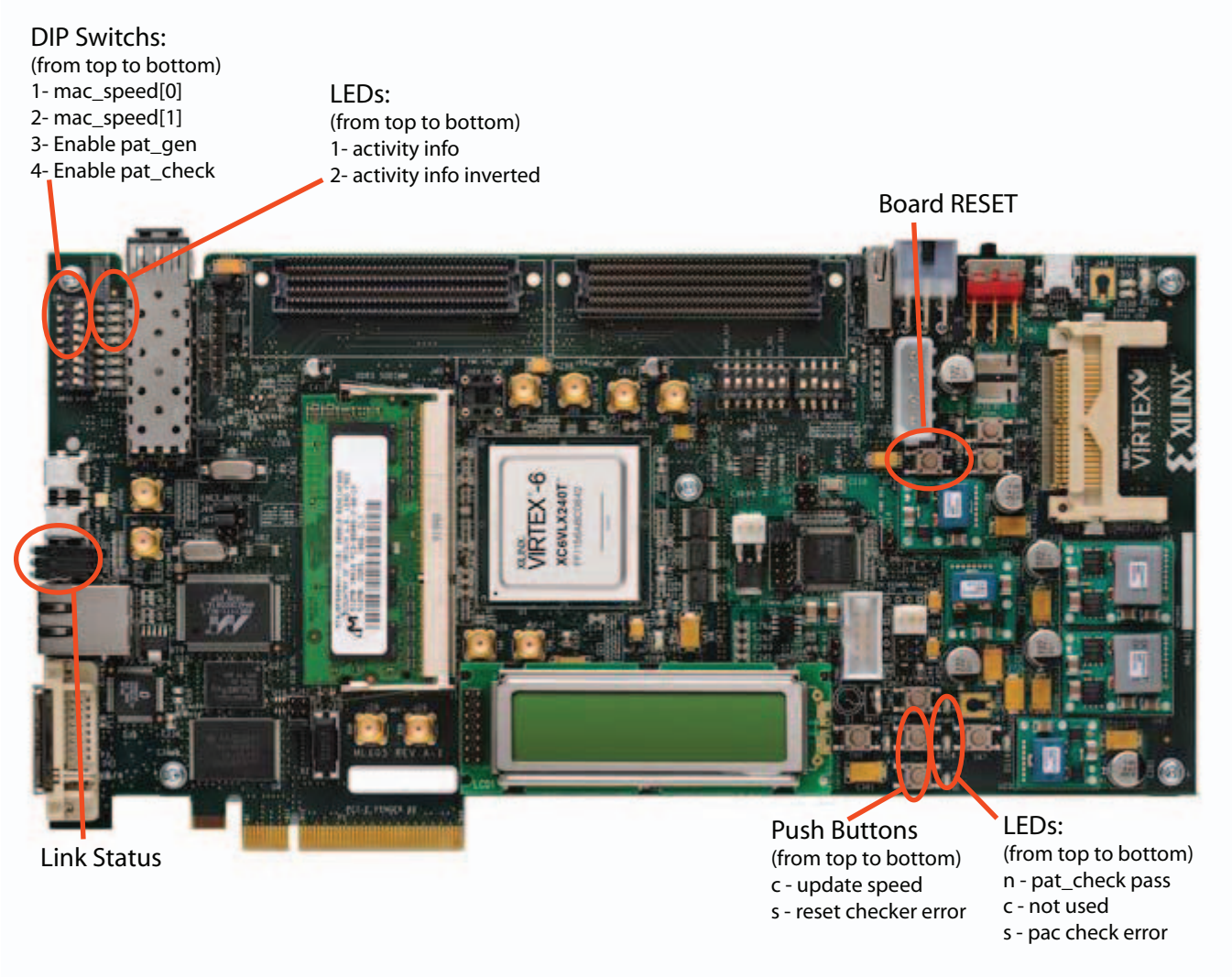

Figure 18-8: **ML605 Board Connectivity**

KC705 Board

The UCF targets the KC705 when any Kintex-7 FPGA part is selected.

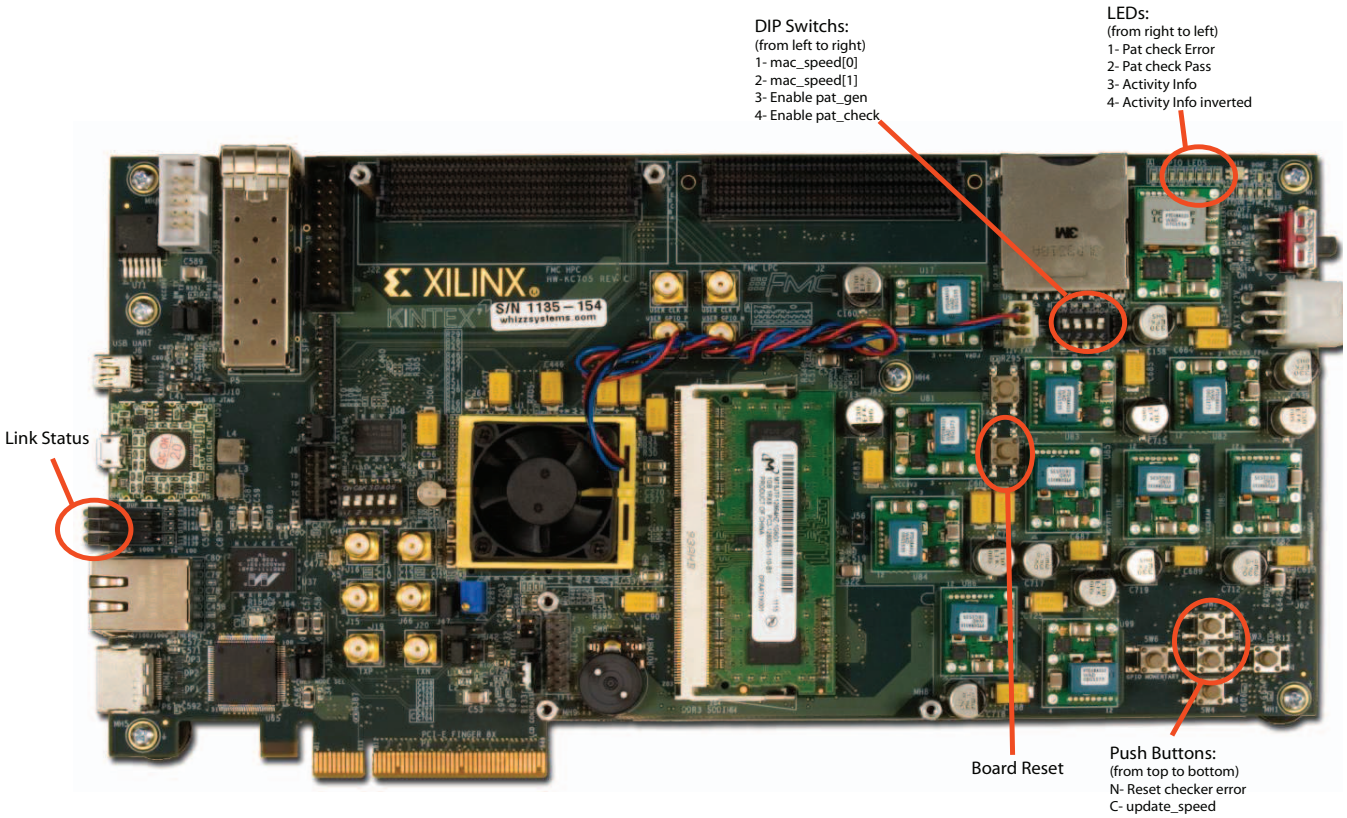

Figure 18-9: **KC705 Board Connectivity**

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

  A 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 in any of the supported families. See the 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 using the core in Virtex®-7, Kintex™-7, Artix-7, Virtex-6 and Spartan®-6 devices (see the Physical Interface chapters in this Guide).

MMCM Usage

MMCM Phase Shifting Requirements

When using a 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 MMCM 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 empirically. 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. The phase shift 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 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. After 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 hardware families. Xilinx does not recommend attempting to determine the tap setting empirically. 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. After 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

Differences between the Embedded Tri-Mode Ethernet MACs and the Soft TEMAC Solution IP Core

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), solutions provided by the CORE Generator™ tool.

Note: This guide only considers the V6 Embedded Tri-Mode Ethernet MAC v2.1 and later and the Soft Tri-Mode Ethernet MAC v5.2 and later.

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 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.

Features Exclusive to Soft 10/100/1000 Mb/s, 1000 Mb/s and 10/100 Mb/s IP Cores

These features are exclusive to soft 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.
  - Includes the optional AVB Endpoint front end.
- The soft PCS/PMA core:
  - Supports the ten bit interface (TBI).
  - 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

Translating to AXI Tri-Mode Ethernet MAC

As of Tri-Mode Ethernet MAC v5.1 onwards the MAC uses an optional AXI4-Lite interface for the configuration, AXI4-Stream for data transfer, and has an entirely new memory map.

This Appendix describes the differences between the legacy interfaces used in prior versions and those in TEMAC version 5.1 and onwards.

Host Interface to AXI4-Lite

The management interface uses the industry standard AXI4-Lite to allow access to the MAC netlist. This interface replaces the Host interface for these operations:

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

As these four features are now accessible using a single interface they have been combined into a single memory map. Table C-1 lists the AXI4-Lite registers and, if appropriate, their legacy host locations. For more details see Chapter 10, Configuration and Status.

Table C-1: **AXI4-LITE/Host Address Map Comparison**

AXI4-Lite Address	Legacy Host Address	Name	Notes
0x200-0x204	0x001	Received bytes	Upper and lower words are separately addressed - but need to be linked, that is, upper access must follow lower access.
0x208-0x20C	0x000	Transmitted bytes	
0x210-0x214	0x002	RX Undersize frames	
0x218-0x21C	0x003	RX Fragment frames	
0x220-0x224	0x004	RX 64 byte Frames	
0x228-0x22C	0x005	RX 65-127 byte Frames	
0x230-0x234	0x006	RX 128-255 byte Frames	
0x238-0x23C	0x007	RX 256-511 byte Frames	
0x240-0x244	0x008	RX 512-1023 byte Frames	

Table C-1: **AXI4-LITE/Host Address Map Comparison *(Cont'd)***

AXI4-Lite Address	Legacy Host Address	Name	Notes
0x248-0x24C	0x009	RX 1024-MaxFrameSize byte Frames	Upper and lower words are separately addressed - but need to be linked, that is, upper access must follow lower access.
0x250-0x254	0x00A	RX Oversize Frames	
0x258-0x25C	0x00B	TX 64 byte Frames	
0x260-0x264	0x00C	TX 65-127 byte Frames	
0x268-0x26C	0x00D	TX 128-255 byte Frames	
0x270-0x274	0x00E	TX 256-511 byte Frames	
0x278-0x27C	0x00F	TX 512-1023 byte Frames	
0x280-0x284	0x010	TX 1024-MaxFrameSize byte Frames	
0x288-0x28C	0x011	TX Oversize Frames	
0x290-0x294	0x012	RX Good Frames	
0x298-0x29C	0x013	RX Frame Check Sequence Errors	
0x2A0-0x2A4	0x014	RX Good Broadcast Frames	
0x2A8-0x2AC	0x015	RX Good Multicast Frames	
0x2B0-0x2B4	0x016	RX Good Control Frames	
0x2B8-0x2BC	0x017	RX Length/Type Out of Range	
0x2C0-0x2C4	0x018	RX Good VLAN Tagged Frames	
0x2C8-0x2CC	0x019	RX Good Pause Frames	
0x2D0-0x2D4	0x01A	RX Bad Opcode	
0x2D8-0x2DC	0x01B	TX Good Frames	
0x2E0-0x2E4	0x01C	TX Good Broadcast Frames	
0x2E8-0x2EC	0x01D	TX Good Multicast Frames	
0x2F0-0x2F4	0x01E	TX Good Underrun Errors	
0x2F8-0x2FC	0x01F	TX Good Control Frames	
0x300-0x304	0x020	TX Good VLAN Tagged Frames	
0x308-0x30C	0x021	TX Good Pause Frames	
0x310-0x314	0x022	TX Single Collision Frames	
0x318-0x31C	0x023	TX Multiple Collision Frames.	
0x320-0x324	0x024	TX Deferred	
0x328-0x32C	0x025	TX Late Collisions	
0x330-0x334	0x026	TX Excess collisions	
0x338-0x33C	0x027	TX Excess Deferral	
0x340-0x344	0x028	TX Alignment Errors	

Table C-1: **AXI4-LITE/Host Address Map Comparison *(Cont'd)***

AXI4-Lite Address	Legacy Host Address	Name	Notes
0x348-0x3FC	NA	Reserved	
0x400	0x200-0x23F	Receiver Configuration Word 0	No change
0x404	0x240-0x27F	Receiver Configuration Word 1	No change
0x408	0x280-0x2BF	Transmitter Configuration	No change
0x40C	0x2C0-0x2FF	Flow Control Configuration	No change
0x410	0x300-0x31F	Speed configuration	No change
0x414	NA	RX Max Frame Configuration	New feature
0x418	NA	TX Max Frame Configuration	New feature
0x41C-0x4F4	NA	Reserved	
0x4F8	NA	ID Register	New feature
0x4FC	NA	Ability Register	New feature
0x500	0x340-0x37F	MDIO Configuration Word 0	No Change
0x504	NA	MDIO Configuration Word 1	MDIO is fully address-mapped and MDIO Ready can be either polled or setup as an interrupt. See MDIO Interface in Chapter 10 for more information.
0x508	NA	MDIO TX Data	
0x50C	NA	MDIO RX Data	
0x510-0x5FC	NA	Reserved	
0x600	NA	Interrupt status Register.	New feature. See Interrupt Controller in Chapter 10 for more information.
0x604-0x60C	NA	Reserved	
0x610	NA	Interrupt Pending Register.	
0x614-0x61C	NA	Reserved	
0x620	NA	Interrupt Enable Register.	
0x624-0x62C	NA	Reserved	
0x630	NA	Interrupt Clear Register.	
0x634-0x6FC	NA	Reserved	
0x700	0x380-0x383	Unicast Address word 0	No Change
0x704	0x384-0x387	Unicast Address word 1	No Change
0x708	0x390-0x3BF	Frame filter Control	Additional control to select which filter is being addressed
0x70C	NA	Frame filter Enable	New Feature - by default filter 0 is enabled. See Frame Filter in Chapter 10 for more information.
0x710	0x388-0x38B	Frame filter value bytes 3-0	This register accesses the filter specified by the frame filter control register.

Table C-1: **AXI4-LITE/Host Address Map Comparison *(Cont'd)***

AXI4-Lite Address	Legacy Host Address	Name	Notes
0x714	0x38C-0x38F	Frame Filter value bytes 7-4	By default only the bottom 16 bits are used (controlled through the mask registers). The upper 16 bits provide extended filter capabilities.
0x718-0x74C	NA	Frame Filter value bytes 63-8	
0x750-0x78C	NA	Frame filter mask value bytes 63-0	By default only the destination address is compared.
0x790-0x7FC	NA	Reserved	

Table C-2 describes the optional signals used to access the MAC netlist.

Table C-2: **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
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

With the legacy host interface each of these four key features had different access models as shown in Figure C-3 to Figure C-6. With the move to AXI4-Lite and a single memory map the same format and timing is used for all accesses as shown in Figure C-1 to Figure C-2.

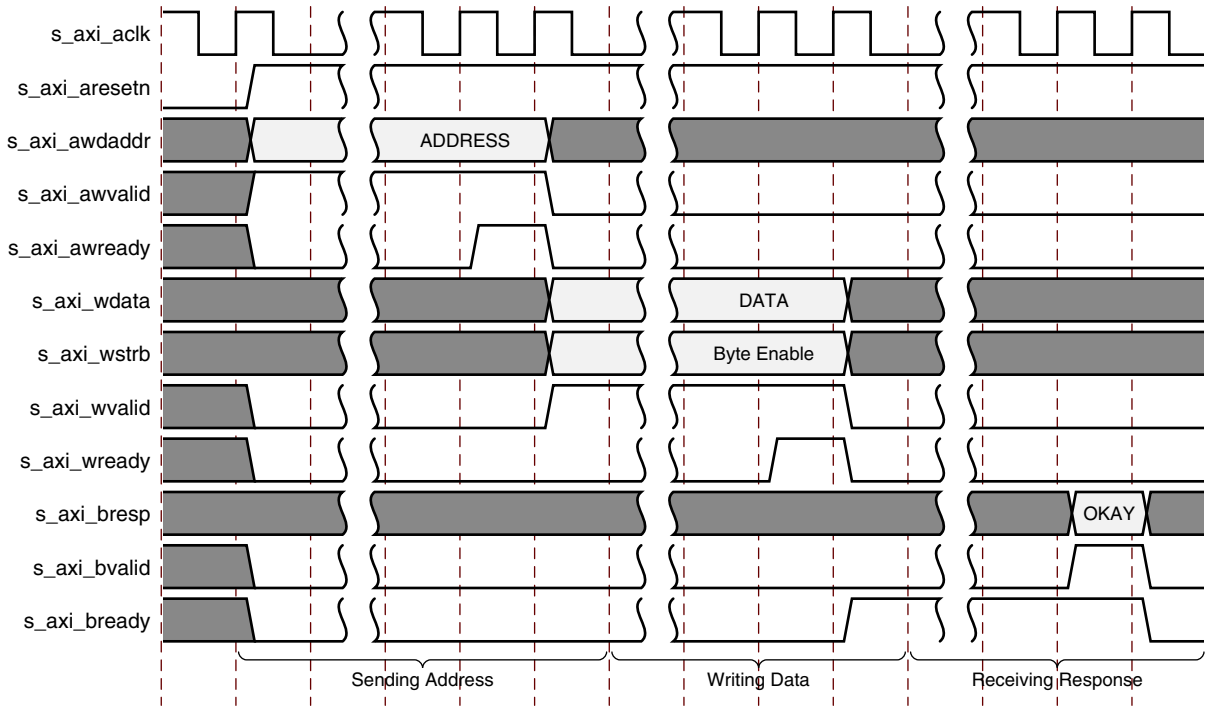

Figure C-1: **AXI4-Lite Write**

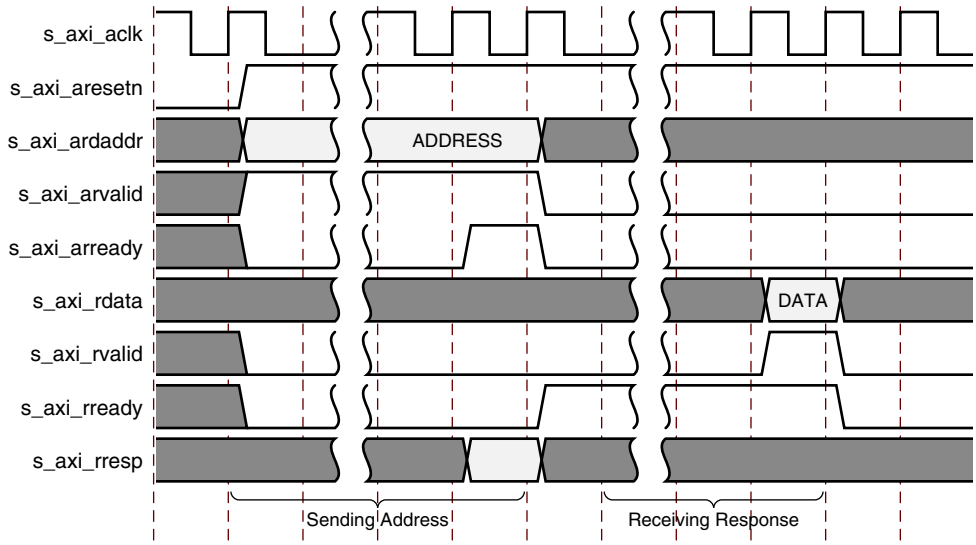

Figure C-2: **AXI4-Lite Read**

Host MAC Configuration

Host MAC configuration writes require the `hostmiimsel` to be driven low and `hostaddr[9]` to be driven high. A read or write is controlled by the `hostopcode` upper bit. `host_req` and `host_rdy` are not used.

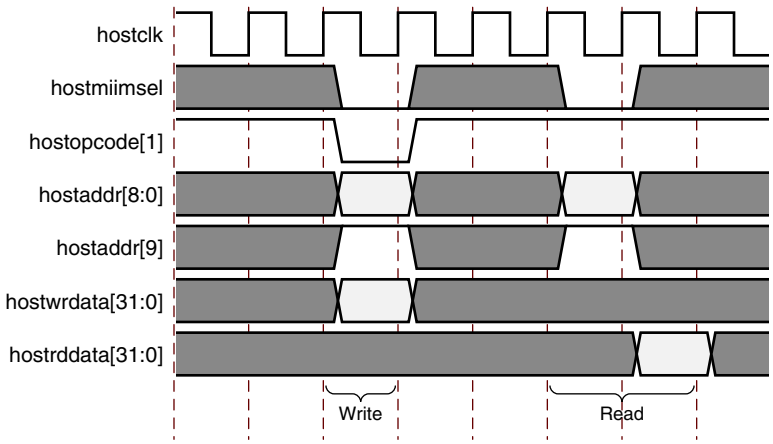

Figure C-3: **Host Interface MAC Configuration Read/Write**

AXI4-Lite accesses use a standard memory-mapped access as shown in Figure C-1 and Figure C-2. See MAC Configuration in Chapter 10 for more information.

Host Filter Access

Host filter accesses are controlled using a write to the filter control register at 0x18C (in the legacy core - an equivalent register does not exist from version 5.1 onwards). For writes the new filter data is split over two writes with the first setting the lower 32-bits of the 48-bit address and the control write setting the upper 16-bits of the 48-bit address and specifying the filter that is to be updated. For a read, the control register is written to specify a read and the filter to be read and the 48-bit address is returned over the following two cycles.

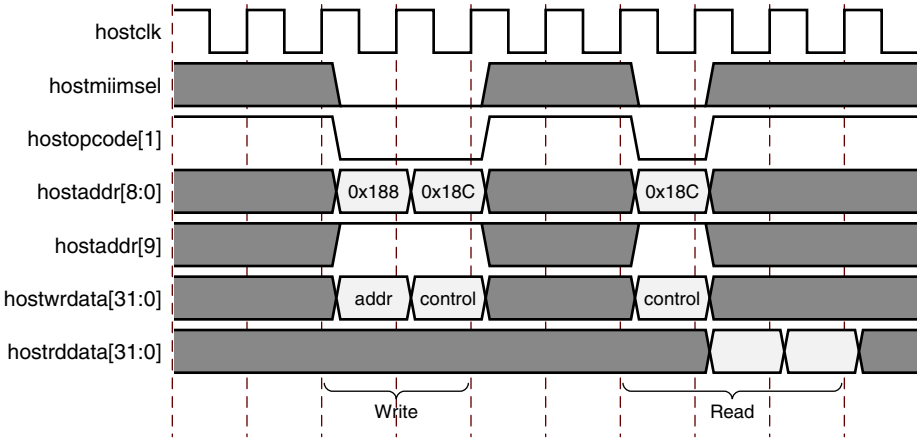

Figure C-4: **Address Filter Read/Write**

AXI4-Lite accesses use a standard memory-mapped access as shown in Figure C-1 and Figure C-2. However, the updated frame filter provides significantly more filtering capabilities covering the entire first 64 bytes of any frame. The Frame filter Control register (0x708) is therefore used to specify which of the available filters is being accessed by accesses in the range 0x70C to 0x790. To access a particular filter:

- Write to the Frame filter control register(0x708) to select the Filter
- Write or read to the specified filter register

See Frame Filter in Chapter 10 for more information.

Host Statistics Read

Host Statistics reads make use of the host_req to initialize an access, with hostmiimsel and hostaddr[9] being low to indicate a statistics access is required.

The stats specific host_stats_lsw_rdy and host_stats_msw_rdy output indicate when valid stats data is present on the read data bus. This data is always presented 6 cycles after the request is initiated.

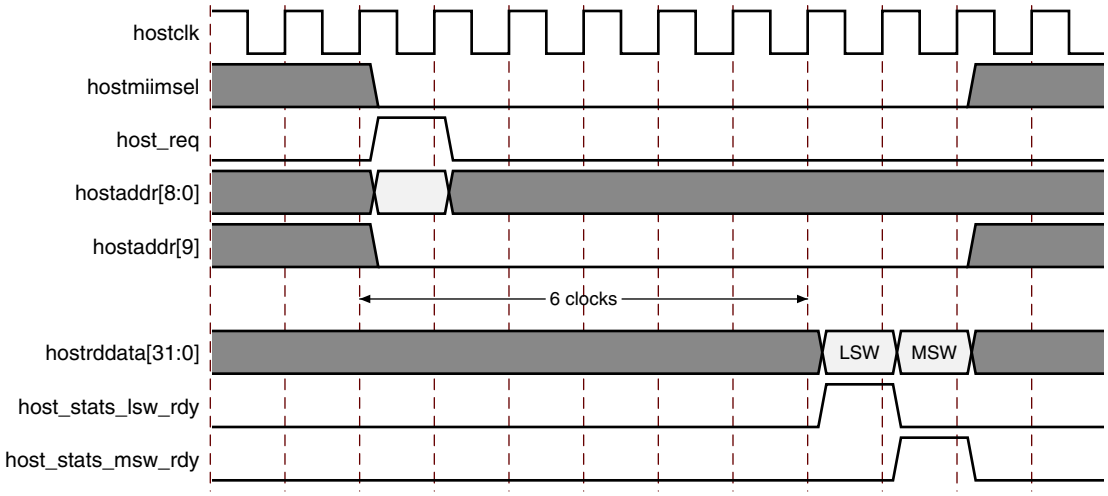

Figure C-5: **Statistics Read**

AXI4-Lite accesses use a standard memory-mapped access as shown in Figure C-1 and Figure C-2. However, there is a read-order requirement if the full 64-bit statistics value is desired. In all cases the lowest location must first be read, this causes the upper 32-bit value to be captured. The following read, to the statistics block, can either be to the related upper 32-bit location OR to another statistics counter's lower 32-bit location. A read to another counter's upper 32-bit location results in an error. See Statistics Counters in Chapter 10 for more detail.

Host MDIO Read/Write

Host MDIO accesses are initialized with the assertion of host_req when hostmiimsel is high. The values presented on the hostopcode and hostaddr inputs then specify the type of access and its location. When a write is required, the hostwrdata must also be valid in this cycle. When the access has started, the MDIO specific host_rdy output drops low, remaining in this state until the access is complete. When a read has been requested the hostrddata becomes valid at this point.

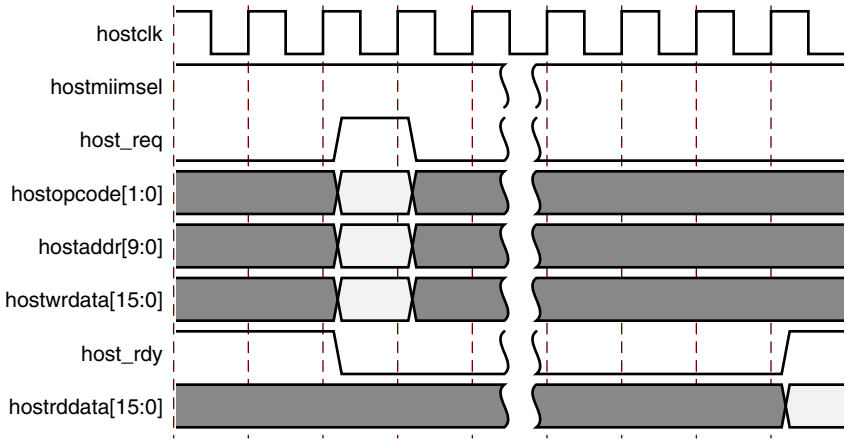

Figure C-6: **MDIO Access**

AXI4-Lite accesses use a standard memory-mapped access as shown in Figure C-1 and Figure C-2. MDIO accesses use a mailbox that does not hold up the bus for the duration of the access. For a write the write data must be first set to the desired value by writing to the write data register at 0x508; the write access is then initialized by writing to the MDIO control register at 0x504 with the access location and type being specified. See MDIO Interface in Chapter 10.

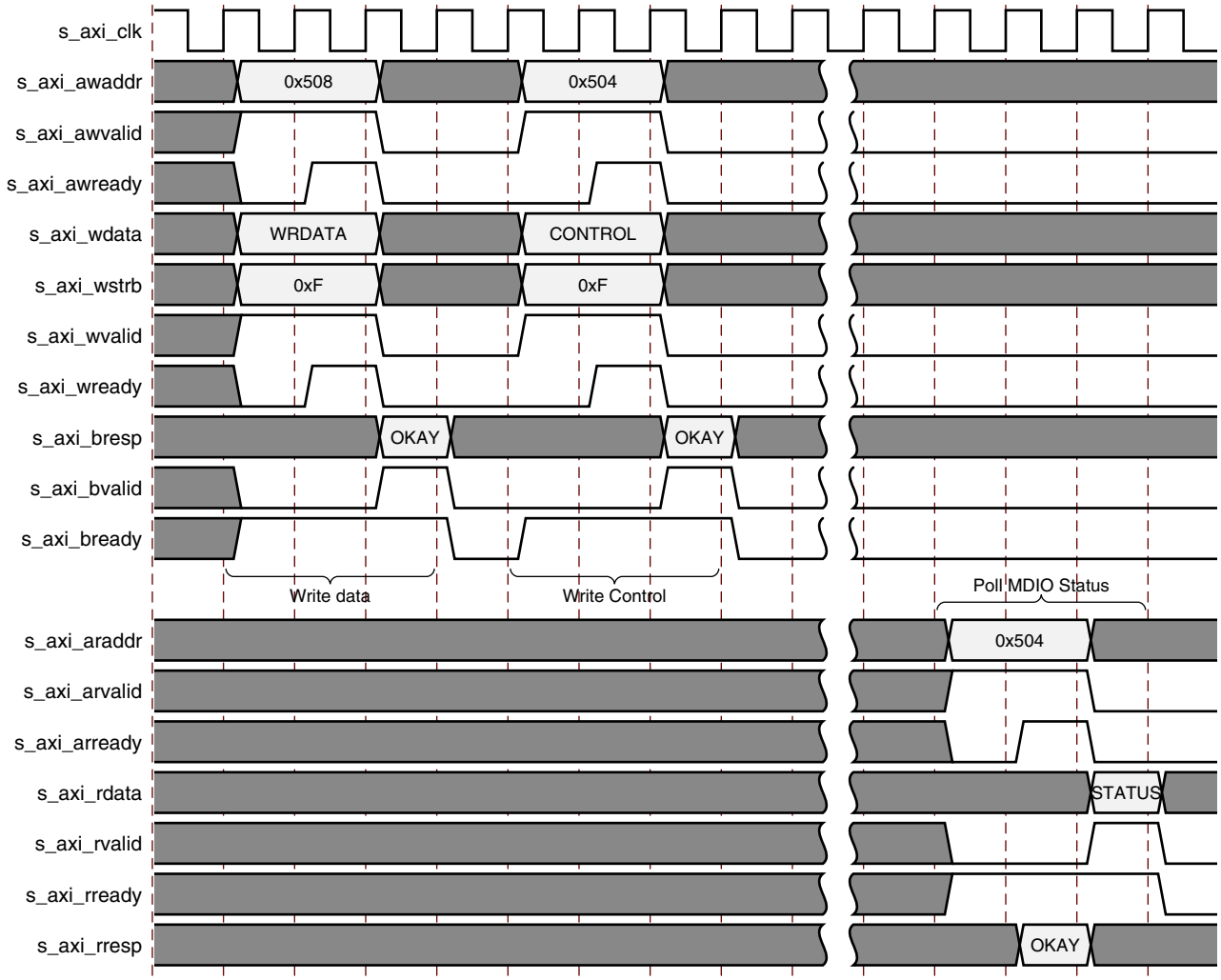

Figure C-7: **AXI4-Lite MDIO Write Access**

For a read only, the MDIO control register write is required.

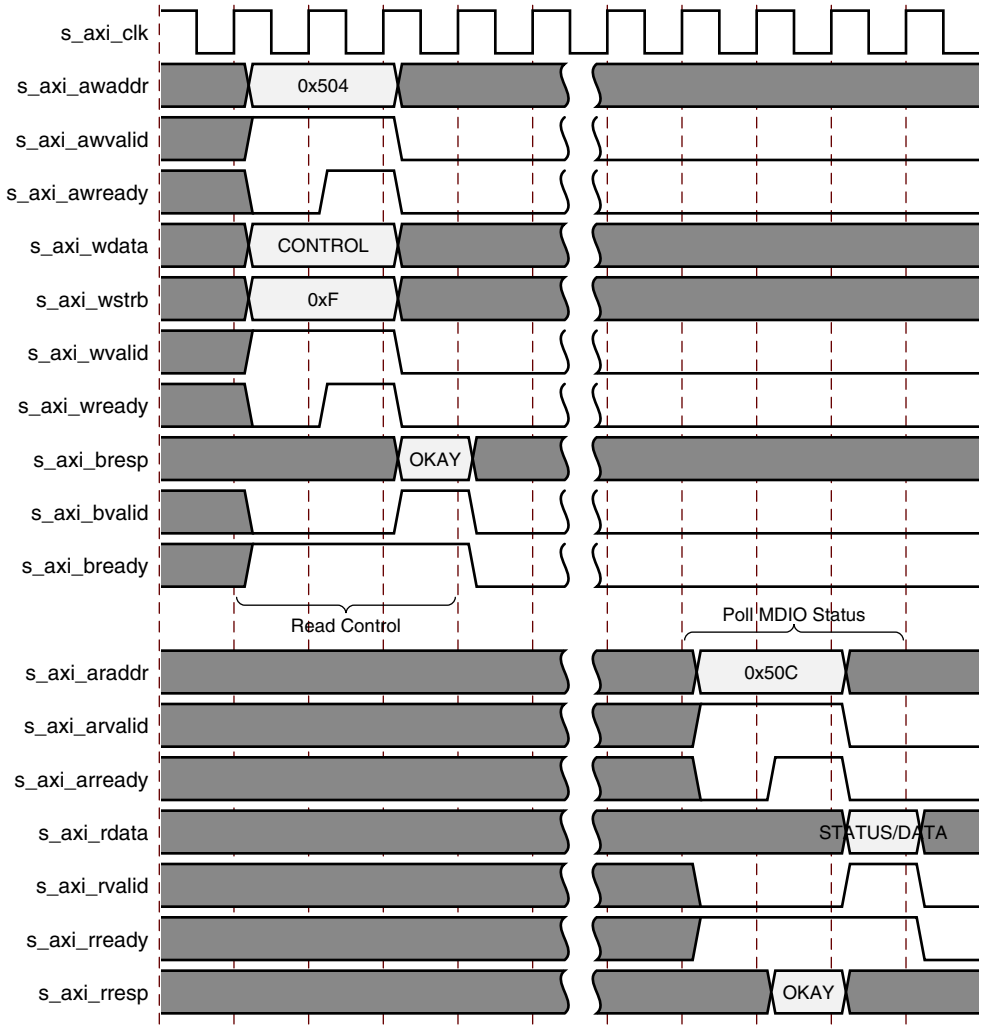

Figure C-8: **AXI4-Lite MDIO Read Access**

In both cases the same methods can be used to identify if an mdio transaction has completed. Either poll the `mdio_ready` status in either the MDIO control register (0x504) or the MDIO Read data register (0x50C) or setup the interrupt controller to provide an interrupt. After the MDIO transaction is complete, a read results in valid data in the MDIO Read data register.

Client Interface to AXI4-Stream

The following tables show the RX and TX AXI4-Stream signals.

Table C-3: **TX AXI4-Stream Signal Pinout**

Signal	Direction	Clock Domain	Description
tx_mac_clk	Input	N/A	Clock for AXI4-Stream
tx_axis_mac_tdata	Input	tx_mac_clk	Data
tx_axis_mac_tvalid	Input	tx_mac_clk	Data Valid
tx_axis_mac_tlast	input	tx_mac_clk	Final transfer of frame
tx_axis_mac_tuser	Input	tx_mac_clk	Explicit Error indication
tx_axis_mac_tready	output	tx_mac_clk	MAC ready for data

Table C-4: **RX AXI4-Stream Signal Pinout**

Signal	Direction	Clock Domain	Description
rx_mac_clk	Input	N/A	Clock for AXI4-Stream
rx_axis_mac_tdata	Output	rx_mac_clk	Data
rx_axis_mac_tvalid	Output	rx_mac_clk	Data Valid
rx_axis_mac_tlast	Output	rx_mac_clk	final transfer of frame
rx_axis_mac_tuser	Output	rx_mac_clk	Frame good/bad indication

TX Client Interface vs TX AXI4-Stream

The TX client interface requires the use of an acknowledge from the MAC to identify when a data transfer can continue; this is shown in Figure C-9.

The key requirement is that the first byte of a frame is presented to the MAC, with `txdvld` high, and then held until `tx_ack` is asserted. The MAC expects new data on the following cycle, and on each valid cycle after that until the end of the frame. The deassertion of `txdvld` identifies the end of the frame.

If data cannot be made available at the required rate, the user is expected to assert `tx_underrun` to ensure the frame is errored. For lower rates, where data is only not required on every physical cycle, it is expected that a clock enable is used to control the data rate.

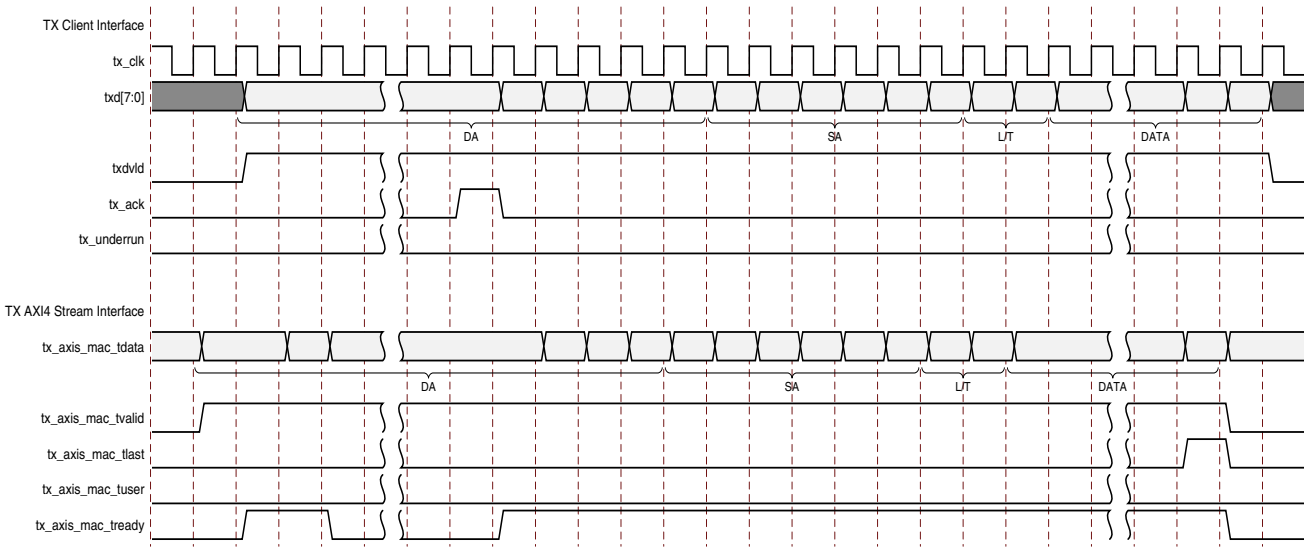

Figure C-9: **TX Client access vs TX AXI4-Stream**

The TX AXI4-Stream access is also shown in Figure C-9. Because there is no built-in FIFO to allow throttling of frame data, the requirement still exists to always provide data to the MAC when requested. However, AXI4-Stream uses a standard ready/valid handshake throughout the frame and requires the final byte of the frame to be identified with `tlast`.

The TX AXI4-Stream interface allows for both implicit and explicit error insertion. In the case of frame underrun, the valid would be dropped mid-frame and, if `tlast` was not asserted on the previous cycle, this would implicitly create an error. Deasserting the `tuser` input allows an error to be forced under direct user control.

In the case of lower rates, the is no difference in the required user logic as `tready` is used to actively control the data throughput.

RX Client Interface vs RX AXI4-Stream

The RX Client interface, shown in Figure C-10, outputs data as received from the PHY, with a frame good or frame bad indication being set when the frame is complete. For lower rates a clock enable is used to control the data throughput.

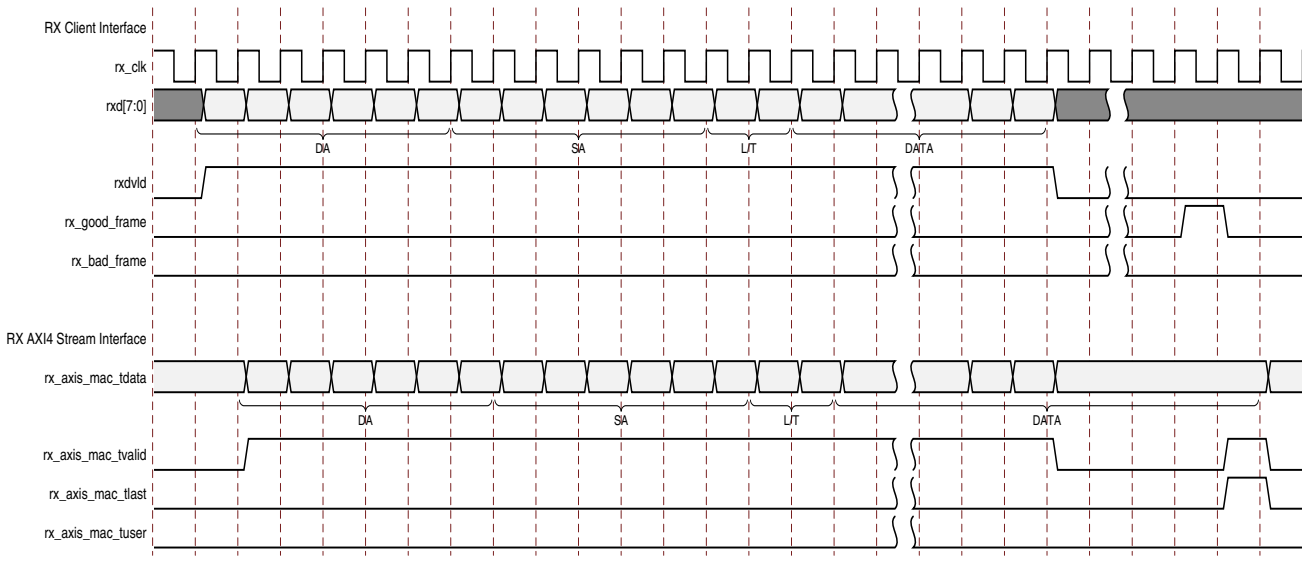

Figure C-10: **RX Client Transfer vs RX AXI4-Stream**

The RX AXI4-Stream interface, also shown in Figure C-10, is almost identical to the RX Client interface with the main difference being the use of `tlast` to identify the final byte of the frame. Unlike the TX interface, no ready is used or required, and it is assumed by the MAC that data can be received at full-line rate if required.

The `tuser` output is used to identify if a frame is good or bad, and this is only valid on the cycle `tlast` is asserted. Because the frame can optionally strip the frame CRC, and this is checked prior to marking a frame as good/bad, the final byte of the frame can be extended, with `tvalid` deasserted, until this check has been performed.

For lower data rates `tvalid` is used to control the data throughput.

LocalLink to AXI4-Stream Translation

The example design FIFO was previously provided with a LocalLink interface. This has also been converted to AXI-Stream. Because the LocalLink interface uses handshaking to transfer data it is almost identical in all but signal names, see Table C-5:

Table C-5: **LocalLink to AXI4-Stream**

LocalLink Name	AXI4-Stream Name	Difference
data	tdata	Name change only
eof_n	tlast	Name change; tlast is the inverse of eof_n
dst_rdy_n	tready	Name change; tready is the inverse of dst_rdy_n
sof_n		No direct equivalent
src_rdy_n		No direct equivalent
	tvalid	Generated from sof_n and src_rdy_n. tvalid can only be high when valid frame data is present.

Figure C-11 shows a LocalLink transfer and the associated AXI4-Stream signals. This is identical for both TX and RX.

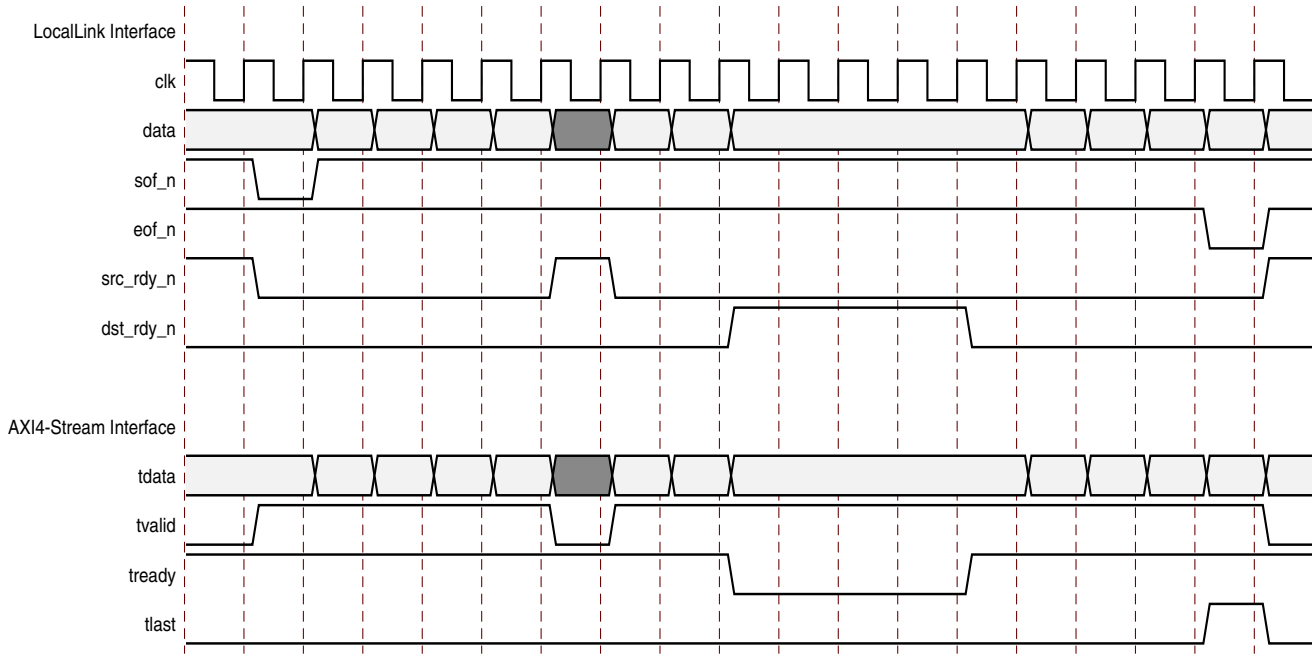

Figure C-11: **LocalLink vs AXI4-Stream**

XILINX®

Appendix D

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

http://www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

Additional Core Resources

For details and updates about the core, see the data sheet, available from the TEMAC 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

References

1. AMBA AXI4-Stream Protocol Specification
2. LogiCORE IP Tri-Mode Ethernet MAC v5.2 Data Sheet (DS818)
3. Ethernet 1000BASE-X PCS/PMA or SGMII User Guide (UG155)
4. 7 Series Data Sheets
5. Spartan-6 FPGA Clocking Resources User Guide (UG382)
6. AXI Ethernet Data Sheet (DS759)
7. Xilinx Synthesis and Simulation Design Guide
8. Xilinx ISE Design Suite Documentation
9. Xilinx Command Line Tools User Guide (UG628)
10. IEEE 802.3-2008
11. IEEE 802.1 AS
12. IEEE 802.1 Q-2011
13. IEEE 802.1BA-2011
14. Reduced Gigabit Media Independent Interface (RGMII), version 2.0