LogiCORE™ IP Ethernet 1000BASE-X PCS/PMA or SGMII v10.3

User Guide

UG155 September 16, 2009

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Doc Version	Revision
09/30/04	1.0	Initial Xilinx release.
04/28/05	2.0	Updated to Xilinx tools 7.1i SP2, support for Virtex®-4 Rocket IO.
01/18/06	3.0	Updated to Xilinx tools 8.1i SP1 for 7.0 release, added new chapter for dynamic switching.
07/13/06	4.0	Updated to core version 7.1; Xilinx tools 8.2i.
10/23/06	5.0	Updated to core version 8.0, support for Virtex-5 LXT and Spartan®-3A families.
02/15/07	6.0	Updated to core version 8.1, Xilinx tools 9.1i.
08/08/07	7.0	Updated to core version 9.0, Xilinx tools 9.2i.
03/24/08	8.0	Updated to core version 9.1, Xilinx tools 10.1.
04/24/09	9.0	Updated to core version 10.1, Xilinx tools 11.1, support for Virtex-5 TXT and Virtex-6 families
06/24/09	10.0	Updated to core version 10.2, Xilinx tools 11.2, support for Spartan-6 family
09/16/09	11.0	Updated to core version 10.3, Xilinx tools 11.3.

Table of Contents

Chapter 4: Designing with the Core

Chapter 5: Using the Client-Side GMII Data Path

Chapter 6: The Ten-Bit Interface

Chapter 7: 1000BASE-X with Transceivers

Chapter 8: SGMII / Dynamic Standards Switching with Transceivers

Chapter 9: Configuration and Status

Chapter 10: Auto-Negotiation

Chapter 11: Dynamic Switching of 1000BASE-X and SGMII Standards

Chapter 12: Constraining the Core

Chapter 13: Interfacing to Other Cores

Chapter 14: Special Design Considerations

Chapter 15: Implementing the Design

Appendix A: Core Verification, Compliance, and Interoperability

Appendix B: Core Latency

Appendix C: Calculating the DCM Fixed Phase Shift Value

Appendix D: 1000BASE-X State Machines

Appendix E: Rx Elastic Buffer Specifications

Appendix F: Debugging Guide

Schedule of Figures

Chapter 1: Introduction

Chapter 2: Core Architecture

Chapter 3: Generating and Customizing the Core

Chapter 4: Designing with the Core

Chapter 5: Using the Client-Side GMII Data Path

Chapter 6: The Ten-Bit Interface

Chapter 7: 1000BASE-X with Transceivers

Chapter 8: SGMII / Dynamic Standards Switching with Transceivers

Chapter 9: Configuration and Status

Chapter 10: Auto-Negotiation

Chapter 11: Dynamic Switching of 1000BASE-X and SGMII Standards

Chapter 12: Constraining the Core

Chapter 13: Interfacing to Other Cores

Chapter 14: Special Design Considerations

Chapter 15: Implementing the Design

Appendix A: Core Verification, Compliance, and Interoperability

Appendix B: Core Latency

Appendix C: Calculating the DCM Fixed Phase Shift Value

Appendix D: 1000BASE-X State Machines

Appendix E: Rx Elastic Buffer Specifications

Appendix F: Debugging Guide

Schedule of Tables

Chapter 1: Introduction

Chapter 2: Core Architecture

Chapter 3: Generating and Customizing the Core

Chapter 4: Designing with the Core

Chapter 5: Using the Client-Side GMII Data Path

Chapter 6: The Ten-Bit Interface

Chapter 7: 1000BASE-X with Transceivers

Chapter 8: SGMII / Dynamic Standards Switching with Transceivers

Chapter 9: Configuration and Status

Chapter 10: Auto-Negotiation

Chapter 11: Dynamic Switching of 1000BASE-X and SGMII Standards

Chapter 12: Constraining the Core

About This Guide

The *LogiCORE™ IP Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* provides information about generating a Xilinx Ethernet 1000BASE-X PCS/PMA or SGMII core, customizing and simulating the core using the provided example design, and running the design files through implementation using the Xilinx tools.

Guide Contents

This guide contains the following information.

- Preface, "About This Guide" introduces the organization and purpose of this guide, a list of additional resources, and the conventions used in this document.
- Chapter 1, "Introduction" describes the core and related information, including recommended design experience, additional documentation resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Core Architecture" provides an overview of the core including all interfaces and major functional blocks.
- Chapter 3, "Generating and Customizing the Core." describes the Graphical User Interface (GUI) options used to generate and customize the core.
- Chapter 4, "Designing with the Core" provides general guidelines for creating designs with the core.
- Chapter 5, "Using the Client-Side GMII Data Path" provides general guidelines for creating designs using client side GMII of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Chapter 6, "The Ten-Bit Interface" provides general design guidelines when using the Ten-Bit Interface (TBI) as the Physical Side of the core.
- Chapter 7, "1000BASE-X with Transceivers" provides general design guidelines when using the 1000BASE-X standard with the device-specific transceiver as the physical side of the core.
- Chapter 8, "SGMII / Dynamic Standards Switching with Transceivers" provides general design guidelines when using either the SGMII standard, or the Dynamic Switching option (between 1000BASE-X and SGMII standards). These options always use a device-specific transceiver as the physical interface.
- Chapter 9, "Configuration and Status" provides general guidelines for configuring and monitoring the core, including a detailed description of the management registers present in the core.
- Chapter 10, "Auto-Negotiation" provides guidelines for Auto-Negotiation function of the core.

- Chapter 11, "Dynamic Switching of 1000BASE-X and SGMII Standards" provides general guidelines for using the core to perform dynamic standards switching between 1000BASE-X and SGMII.
- Chapter 12, "Constraining the Core" defines the constraint requirements of the core.
- Chapter 13, "Interfacing to Other Cores" describes additional design considerations associated with implementing the core with the Tri-Mode Ethernet MAC core.
- Chapter 14, "Special Design Considerations" describes additional design considerations associated with implementing the core.
- Chapter 15, "Implementing the Design"describes how to simulate and implement your design containing the core.
- Appendix A, "Core Verification, Compliance, and Interoperability" describes how the core was verified.
- Appendix B, "Core Latency" defines the latency of the core.
- Appendix C, "Calculating the DCM Fixed Phase Shift Value" instructs you how to calculate the system timing requirements when using DCMs with the core.
- Appendix D, "1000BASE-X State Machines" serves as a reference for the basic operation of the 1000BASE-X *IEEE 802.3* clause 36 transmitter and receiver state machines.
- Appendix E, "Rx Elastic Buffer Specifications" describes the depth of the Rx Elastic Buffers which are available with the core. The size of the buffer is related to the maximum frame size which the core can accommodate.
- Appendix F, "Debugging Guide" provides information for debugging the core within a system.

Additional Resources

To find additional documentation, see the Xilinx website at:

www.xilinx.com/support/documentation/index.htm

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

www.xilinx.com/support/mysupport.htm

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File →Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**`lowpwr =`**{**`on`**|**`off`**}
Vertical bar |	Separates items in a list of choices	**`lowpwr =`**{**`on`**|**`off`**}
Angle brackets < >	User-defined variable or in code samples	<directory name>

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Convention	Meaning or Use	Example
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	**usr_teof_n** is active low.

Online Document

The following conventions are used in this document.

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. See "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

List of Acronyms

The following table describes acronyms used in this manual.

Acronym	Spelled Out
CLB	Configurable Logic Block
DCM	Digital Clock Manager
DDR	Double Data Rate
DRP	Dynamic Reconfiguration Port
DSP	Digital Signal Processor
FCS	Frame Check Sequence
FIFO	First In First Out
FPGA	Field Programmable Gate Array.
Gbps	Gigabits per second
GMII	Gigabit Media Independent Interface
GUI	Graphical User Interface
HDL	Hardware Description Language
IO	Input/Output
IOB	Input/Output Block
IP	Intellectual Property
ISE®	Integrated Software Environment

Acronym	Spelled Out
IUS	Incisive Unified Simulator (Cadence)
MAC	Media Access Controller
Mbps	Megabits per second
MMD	MDIO Managed Device
MDIO	Management Data Input/Output
MGT	Multi-Gigabit Transceiver
MHz	Mega Hertz
ms	milliseconds
NCD	Native Circuit Description
NGC	Native Generic Circuit
NGD	Native Generic Database
ns	nanoseconds
PAR	Place and Route
PCB	Printed Circuit Board
PCF	Physical Constraints File
PCS	Physical Coding Sublayer
PHY	physical-side interface
PMA	Physical Medium Attachment
PMD	Physical Medium Dependent
SA	Source Address
SFD	Start of Frame Delimiter
SGMII	Serial Gigabit Media Independent Interface
STA	Station Management Entity
TBI	Ten-Bit-Interface
TWR	Timing Wizard Report
UCF	User Constraints File
VHDL	VHSIC Hardware Description Language (VHSIC an acronym for Very High-Speed Integrated Circuits).
VLAN	Virtual LAN (Local Area Network)
XCO	Xilinx CORE Generator™ core source file
XST	Xilinx Synthesis Technology

XILINX®

Chapter 1

Introduction

The Ethernet 1000BASE-X PCS/PMA or SGMII core is a fully verified solution that supports Verilog HDL and VHDL. In addition, the example design provided with the core supports both Verilog and VHDL.

This chapter introduces the Ethernet 1000BASE-X PCS/PMA or SGMII core and provides related information, including recommended design experience, additional resources, technical support, and methods for submitting feedback to Xilinx.

About the Core

The Ethernet 1000BASE-X PCS/PMA or SGMII core is a Xilinx CORE Generator™ IP software core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see the Ethernet 100BASE-X PCS/PMA product page. For information about system requirements and licensing options, see Chapter 2, "Licensing the Core," in the *Getting Started Guide*.

Designs Using Transceivers

Transceivers are defined by device family in the following way:

- For Virtex®--4 devices, RocketIO™ Multi-Gigabit transceivers (MGT)
- For Virtex-5 LXT and SXT devices, RocketIO GTP transceivers; Virtex-5 FXT and TXT devices, RocketIO GTX transceivers
- For Virtex-6 devices, GTX transceivers
- For Spartan®-6 devices, GTP transceivers

Recommended Design Experience

Although the Ethernet 1000BASE-X PCS/PMA or SGMII core is a fully-verified solution, 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.

Additional Core Resources

For detailed information and updates about the Ethernet 1000BASE-X PCS/PMA or SGMII core, see the following documents, located on the Xilinx Ethernet 100BASE-X PCS/PMA product page.

- *Ethernet 1000BASE-X PCS/PMA or SGMII Data Sheet*
- *Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started Guide*

After generating the core, the following documents are available in the document directory:

- *Ethernet 1000BASE-X PCS/PMA or SGMII Release Notes*
- *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide*

Related Xilinx Ethernet Products and Services

For information about all Xilinx Ethernet solutions, see www.xilinx.com/products/design_resources/conn_central/protocols/gigabit_ethernet.htm.

Specifications

- IEEE 802.3
- *Serial-GMII Specification* (CISCO SYSTEMS, ENG-46158)

Technical Support

To obtain technical support specific to the Ethernet 1000BASE-X PCS/PMA or SGMII core, visit www.support.xilinx.com. Questions are routed to a team of engineers with expertise using the Ethernet 1000BASE-X PCS/PMA or SGMII core.

Xilinx provides technical support for use of this product as described in the *Ethernet 1000BASE-X PCS/PMA or SGMII User Guide* and the *Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started 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 Ethernet 1000BASE-X PCS/PMA or SGMII core and the documentation supplied with the core.

Ethernet 1000BASE-X PCS/PMA or SGMII Core

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

- Product name
- Core version number
- Explanation of your comments

Document

For comments or suggestions about this document, please submit a WebCase from www.support.xilinx.com. 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

Core Architecture

This chapter describes the architecture of the Ethernet 1000BASE-X PCS/PMA or SGMII core, including all interfaces and major functional blocks.

System Overview

Ethernet 1000BASE-X PCS/PMA or SGMII Using A Transceiver

The Ethernet 1000BASE-X PCS/PMA or SGMII core provides the functionality to implement the 1000BASE-X PCS and PMA sublayers or used to provide a GMII to SGMII bridge when used with a device-specific transceiver.

Transceivers are defined in the following way:

- Virtex®-4 devices, RocketIO™ Multi-Gigabit transceivers (MGT)
- For Virtex-5 LXT and SXT FPGAs, RocketIO GTP transceivers; Virtex-5 FXT and TXT FPGA, RocketIO GTX transceiver
- For Virtex-6 FPGAs, GTX transceivers
- For Spartan®-6 FPGAs, GTP transceivers

The core interfaces to a device-specific transceiver, providing some of the PCS layer functionality such as 8B/10B encoding/decoding, the PMA SERDES, and clock recovery. Figure 2-1 illustrates the remaining PCS sublayer functionality, and also shows the major functional blocks of the core.

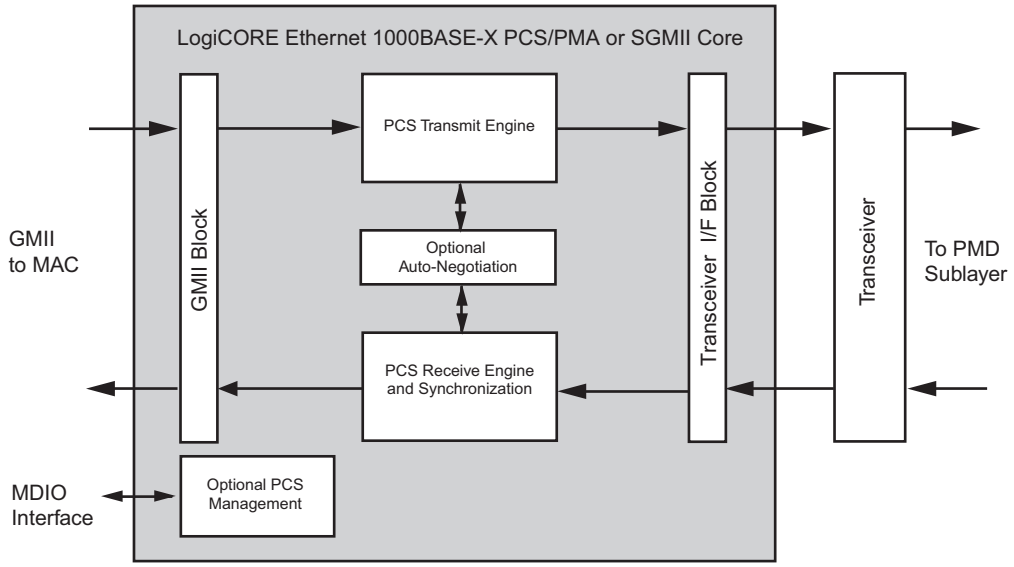


Figure 2-1: **Functional Block Diagram Using Device-Specific Transceiver**

GMII Block

A client-side GMII is provided with the core, which can be used as an internal interface for connection to an embedded Media Access Controller (MAC) or other custom logic. Alternatively, the GMII may be routed to device IOBs to provide an external (off chip) GMII. Virtex-6 devices support GMII at 2.5V only. Please see the *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics* for more information. Virtex-5, Virtex-4, Spartan-6 and Spartan-3 devices support GMII at 3.3V or lower.

PCS Transmit Engine

The PCS transmit engine converts the GMII data octets into a sequence of ordered sets by implementing the state diagrams of *IEEE 802.3* (figures 36-5 and 36-6). See Appendix D, "1000BASE-X State Machines."

PCS Receive Engine and Synchronization

The synchronization process implements the state diagram of *IEEE 802.3* (figure 36-9). The PCS receive engine converts the sequence of ordered sets to GMII data octets by implementing the state diagrams of *IEEE 802.3* (figures 36-7a and 36-7b). See Appendix D, "1000BASE-X State Machines."

Optional Auto-Negotiation Block

IEEE 802.3 clause 37 describes the 1000BASE-X Auto-Negotiation function that allows a device to advertise the modes of operation that it supports to a device at the remote end of a link segment (link partner), and to detect corresponding operational modes that the link partner may be advertising.

Auto-Negotiation is controlled and monitored through the PCS Management Registers. See Chapter 10, "Auto-Negotiation."

Optional PCS Management Registers

Configuration and status of the core, including access to and from the optional Auto-Negotiation function, uses the 1000BASE-X PCS Management Registers defined in *IEEE 802.3* clause 37. These registers are accessed through the serial Management Data Input/Output Interface (MDIO), defined in *IEEE 802.3* clause 22, as if it were an externally connected PHY.

The PCS Management Registers may be omitted from the core when the core is performing the 1000BASE-X standard. In this situation, configuration and status of the core is made possible with the use of an alternative configuration vector and a status signal.

When the core is performing the SGMII standard, the PCS Management Registers become mandatory and information in the registers takes on a different interpretation. For more information, see "Management Registers" in Chapter 9.

Transceiver Interface Block

The Transceiver Interface Block enables the core to connect to a Virtex-4, Virtex-5, Virtex-6 or Spartan-6 FPGA GTP transceiver.

Ethernet 1000BASE-X PCS/PMA or SGMII with Ten-Bit-Interface

The Ethernet 1000BASE-X PCS/PMA or SGMII core, when used with the Ten-Bit Interface (TBI), allows you to implement only the 1000BASE-X PCS sublayer.

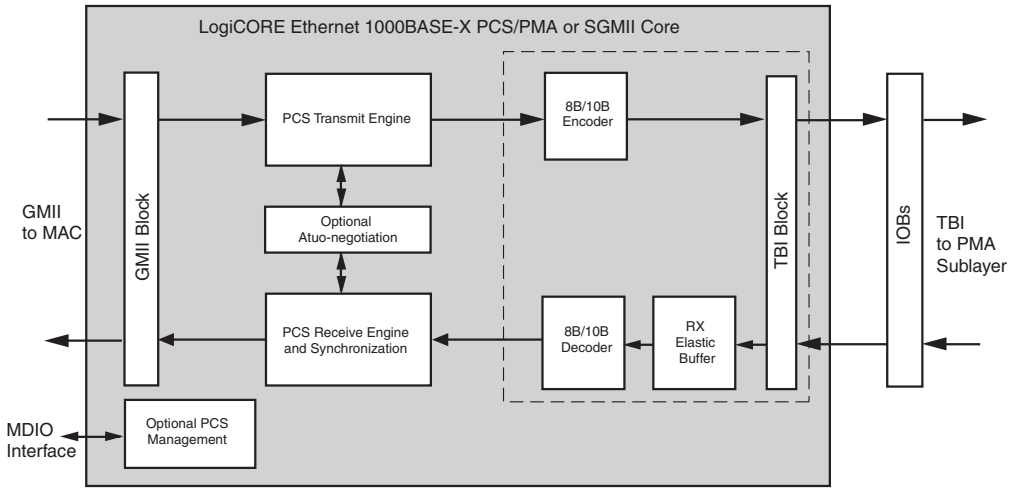


Figure 2-2: **Functional Block Diagram with a Ten-Bit Interface**

The optional TBI can be used in place of the device-specific transceiver to provide a parallel interface for connection to an external PMA SERDES device. In this implementation, additional logic blocks are required to replace some of the device-specific transceiver functionality. These are shown in the surrounded by the dotted line box in Figure 2-2 and are described in the following sections. The other blocks are described previously in this document.

Virtex-6 devices support TBI at 2.5V only. Please see the *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics* for more information. Virtex-5, Virtex-4, Spartan-6 and Spartan-3 devices support TBI at 3.3V or lower.

8B/10B Encoder

8B10B encoding, as defined in *IEEE 802.3* (Tables 36-1a to 36-1e and Table 36-2), is implemented in a block SelectRAM™ memory, configured as ROM, and used as a large look-up table.

8B/10B Decoder

8B10B decoding, as defined in *IEEE 802.3* (Table 36-1a to 36-1e and Table 36-2), is implemented in a block SelectRAM memory, configured as ROM, and used as a large look-up table.

Receiver Elastic Buffer

The Receiver Elastic Buffer enables the 10-bit parallel TBI data, received from the PMA sublayer synchronously to the TBI receiver clocks, to be transferred onto the cores internal 125 MHz clock domain. It is an asynchronous FIFO implemented in internal RAM. The Receiver Elastic Buffer attempts to maintain a constant occupancy by inserting or removing Idle sequences as necessary. This causes no corruption to the frames of data.

TBI Block

The core provides a TBI interface that should be routed to device IOBs to provide an off-chip TBI.

Core Interfaces

All ports of the core are internal connections in FPGA fabric. An HDL example design (delivered with the core) connects the core, where appropriate, to a device-specific transceiver, and/or add IBUFs, OBUFs, and IOB flip-flops to the external signals of the GMII and TBI. IOBs are added to the remaining unconnected ports to take the example design through the Xilinx implementation software.

All clock management logic is placed in this example design, allowing you more flexibility in implementation (such as designs using multiple cores). This example design is provided in both VHDL and Verilog. For more information, see the *Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started Guide*.

Figure 2-3 shows the pinout for the Ethernet 1000BASE-X PCS/PMA or SGMII core using a device-specific transceiver *with* the optional PCS Management Registers. The signals shown in the Auto-Negotiation box included only when the core includes the Auto-Negotiation functionality. For more information, see Chapter 3, "Generating and Customizing the Core."

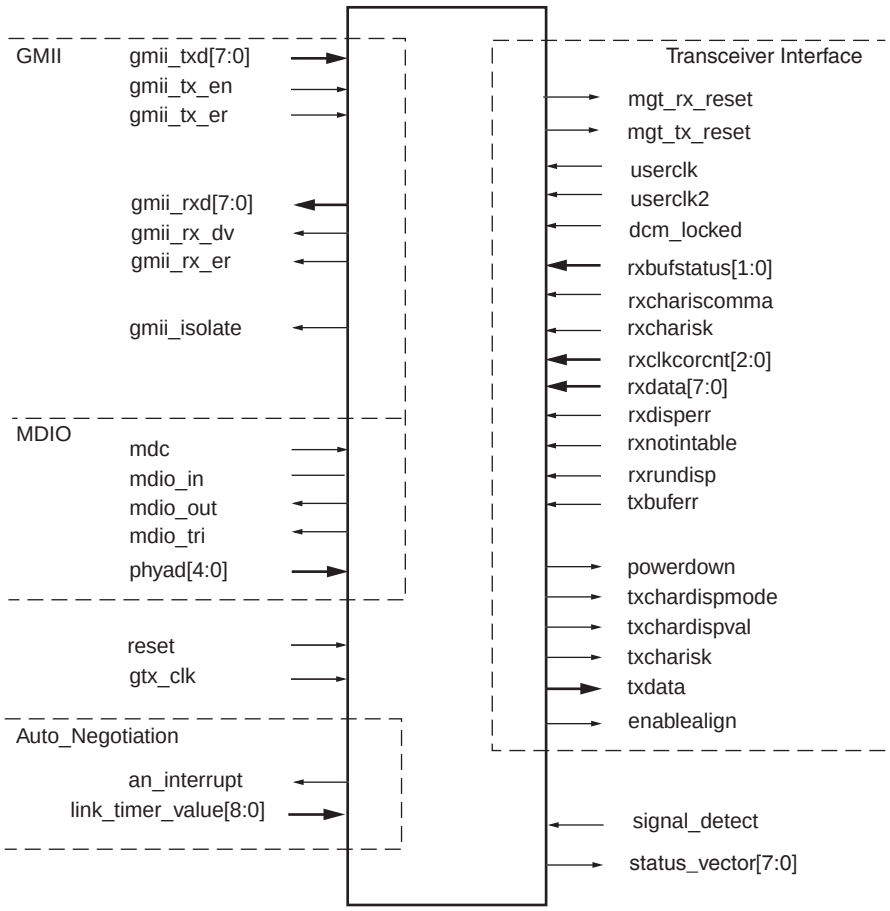


Figure 2-3: **Component Pinout Using a Device-Specific Transceiver *with* PCS Management Registers**

Figure 2-4 shows the pinout for the Ethernet 1000BASE-X PCS/PMA or SGMII core using a device-specific transceiver *without* the optional PCS Management Registers

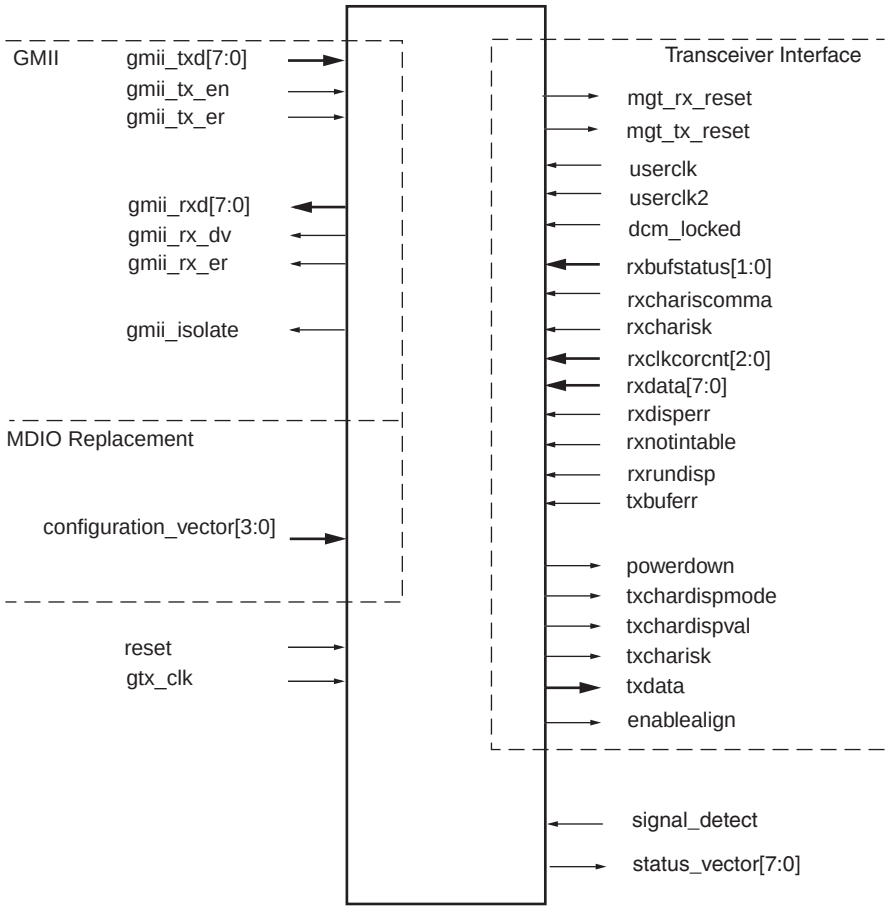


Figure 2-4: **Component Pinout Using a Device-Specific Transceiver *without* PCS Management Registers**

Figure 2-5 shows the pinout for the Ethernet 1000BASE-X PCS/PMA or SGMII core when using the TBI with optional PCS Management Registers. The signals shown in the Auto-Negotiation box are included only when the core includes the Auto-Negotiation functionality (see Chapter 3, "Generating and Customizing the Core.").

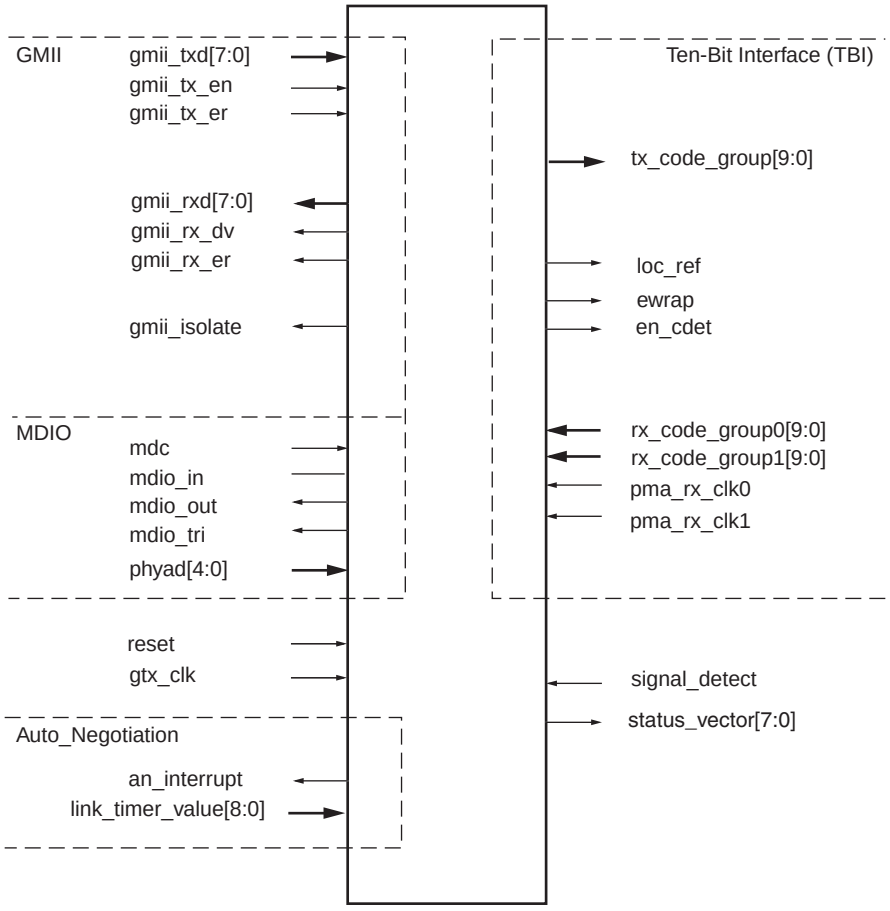


Figure 2-5: **Component Pinout Using the Ten-Bit Interface *with* PCS Management Registers**

Figure 2-6 shows the pinout for the Ethernet 1000BASE-X PCS/PMA or SGMII core when using a TBI without the optional PCS Management Registers.

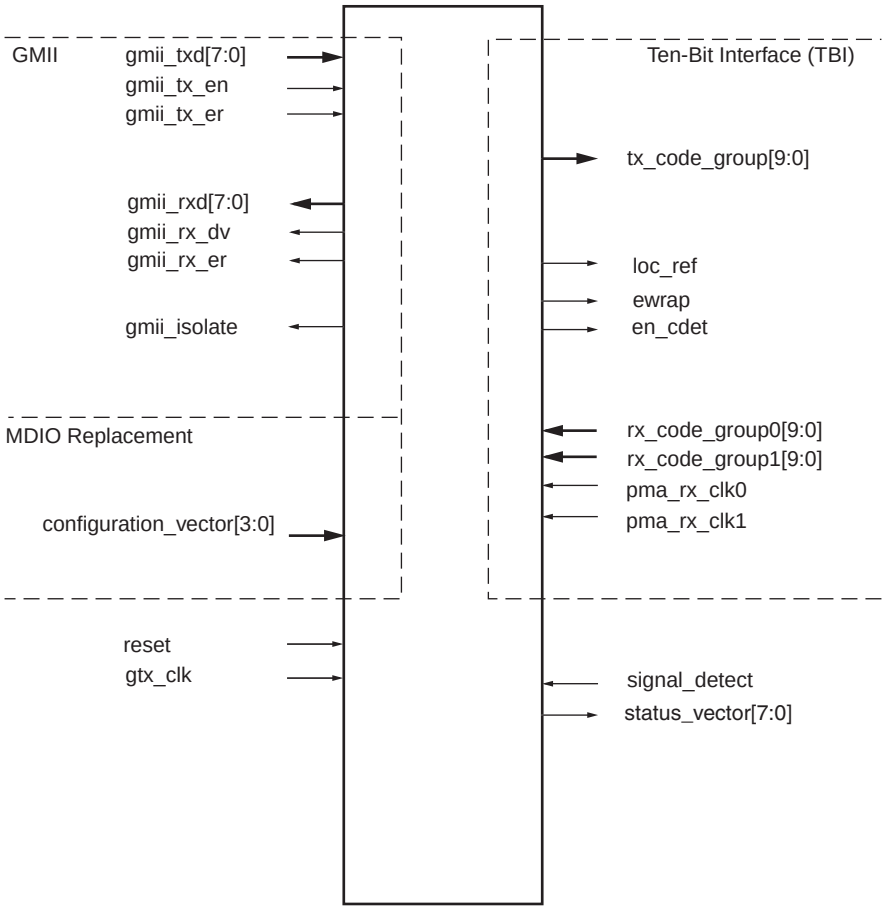


Figure 2-6: **Component Pinout Using Ten-Bit Interface *without* PCS Management Registers**

Figure 2-7 shows the pinout for the Ethernet 1000BASE-X PCS/PMA or SGMII core using the optional dynamic switching logic (between 1000BASE-X and SGMII standards). This mode is shown used with a device-specific transceiver interface. For more information, see Chapter 11, "Dynamic Switching of 1000BASE-X and SGMII Standards."

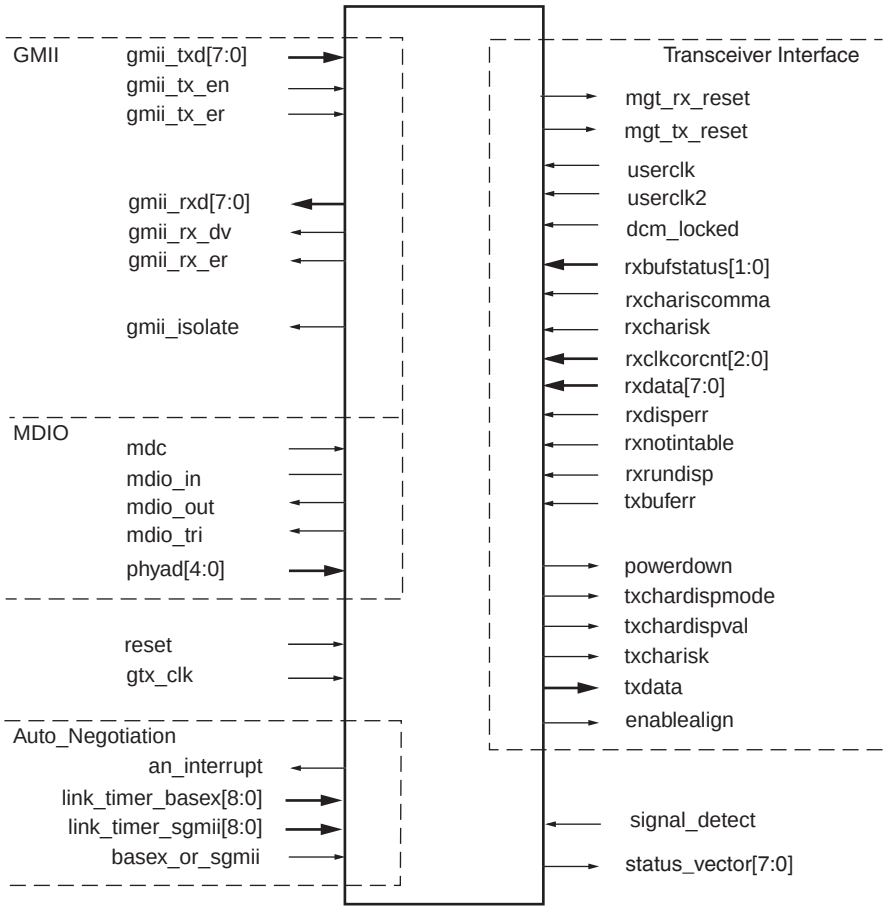


Figure 2-7: **Component Pinout with the Dynamic Switching Logic**

Client Side Interface

GMII Pinout

Table 2-1 describes the GMII-side interface signals of the core common to all parameterizations of the core. These are typically attached to an Ethernet MAC, either off-chip or internally integrated. The HDL example design delivered with the core connects these signals to IOBs to provide a place-and-route example.

For more information, see "Designing with the Client-side GMII for the 1000BASE-X Standard" in Chapter 5.

Table 2-1: **GMII Interface Signal Pinout**

Signal	Direction	Description
gmii_txd[7:0][1]	Input	GMII Transmit data from MAC.
gmii_tx_en[1]	Input	GMII Transmit control signal from MAC.
gmii_tx_er[1]	Input	GMII Transmit control signal from MAC.
gmii_rxd[7:0][2]	Output	GMII Received data to MAC.
gmii_rx_dv[2]	Output	GMII Received control signal to MAC.
gmii_rx_er[2]	Output	GMII Received control signal to MAC.
gmii_isolate[2]	Output	IOB Tri-state control for GMII Isolation. Only of use when implementing an External GMII as illustrated by the example design HDL.

1. When the Transmitter Elastic Buffer is present, these signals are synchronous to gmii_tx_clk. When the Transmitter Elastic Buffer is omitted, see Note 2.
2. These signals are synchronous to the internal 125 MHz reference clock of the core. This is userclk2 when the core is used with the device-specific transceiver; gtx_clk when the core is used with TBI.

Common Signal Pinout

Table 2-2 describes the remaining signals common to all parameterizations of the core.

Table 2-2: **Other Common Signals**

Signal	Direction	Description
reset	Input	Asynchronous reset for the entire core. Active High. Clock domain is not applicable.
signal_detect	Input	Signal direct from PMD sublayer indicating the presence of light detected at the optical receiver. If set to '1,' indicates that the optical receiver has detected light. If set to '0,' this indicates the absence of light. If unused this signal should be set to '1'to enable correct operation the core. Clock domain is not applicable.

Table 2-2: **Other Common Signals *(Continued)***

Signal	Direction	Description
status_vector[7:0][1]	Output	Bit[0]: **Link Status** Indicates the status of the link. • When high, the link is valid: synchronization of the link has been obtained **and** Auto-Negotiation (if present and enabled) has successfully completed. • When low, a valid link has not been established. Either link synchronization has failed or Auto-Negotiation (if present and enabled) has failed to complete. • When auto-negotiation is enabled this signal is identical to Status Register Bit 1.2: Link Status. • When auto-negotiation is disabled this signal is identical to status_vector Bit[1]. Bit[1]: **Link Synchronization** Indicates the state of the synchronization state machine (IEEE802.3 figure 36-9) which is based on the reception of valid 8B10B code groups. This signal is similar to Bit[0] (Link Status), but is NOT qualified with Auto-Negotiation.
		• When high, link synchronization has been obtained and in the synchronization state machine, `sync_status` = OK. • When low, synchronization has failed. Bit[2]: **RUDI(/C/)** The core is receiving /C/ ordered sets (Auto-Negotiation Configuration sequences). Bit[3]: **RUDI(/I/)** The core is receiving /I/ ordered sets (Idles). Bit[4]: **RUDI(INVALID)** The core has received invalid data whilst receiving/C/ or /I/ ordered set. See "status_vector[7:0] signals" in Chapter 5 for more information. Bit[5]: **RXDISPERR** The core has received a running disparity error during the 8B10B decoding function. Bit[6]: **RXNOTINTABLE** The core has received a code group which is not recognized from the 8B10B coding tables. Bit[7]: **PHY Link Status (SGMII mode only)** When operating in SGMII mode, this bit represents the link status of the external PHY device attached to the other end of the SGMII link (high indicates that the PHY has obtained a link with its link partner; low indicates that is has not linked with its link partner). When operating in 1000BASE-X mode this bit will remain low and should be ignored.

1. These signals are synchronous to the internal 125 MHz reference clock of the core. This is userclk2 when the core is used with the device-specific transceiver; this is gtx_clk when the core is used with TBI.

MDIO Management Interface Pinout (Optional)

Table 2-3 describes the optional MDIO interface signals of the core used to access the PCS Management Registers. These signals are typically connected to the MDIO port of a MAC device, either off-chip or to an internally integrated MAC core. For more information, see "Management Registers" in Chapter 9.

Table 2-3: **Optional MDIO Interface Signal Pinout**

Signal	Direction	Clock Domain	Description
mdc	Input	N/A	Management clock (<= 2.5 MHz).
mdio__in[1]	Input	mdc	Input data signal for communication with MDIO controller (for example, an Ethernet MAC). Tie high if unused.
mdio_out[1]	Output	mdc	Output data signal for communication with MDIO controller (for example, an Ethernet MAC).
mdio_tri[1]	Output	mdc	Tri-state control for MDIO signals; '0' signals that the value on mdio_out should be asserted onto the MDIO interface.
phyad[4:0]	Input	N/A	Physical Address of the PCS Management register set. It is expected that this signal will be tied off to a logical value.

1. These signals can be connected to a Tri-state buffer to create a bidirectional `mdio` signal suitable for connection to an external MDIO controller (for example, an Ethernet MAC).

Configuration Vector (Optional)

Table 2-4 shows the alternative to the optional MDIO Management Interface, the configuration vector. See "Optional Configuration Vector" in Chapter 9.

Table 2-4: **Optional Configuration and Status Vectors**

Signal	Direction	Description
configuration_vector[3:0][1]	Input	Bit[0]: Reserved (currently unused) Bit[1]: **Loopback Control** • When the core with device-specific transceiver is used, the core is placed in internal loopback mode. • With the TBI version, Bit 1 is connected to ewrap. When set to '1,' this indicates to the external PMA module to enter loopback mode. Bit[2]: **Power Down** • When the device-specific transceiver is used (when set to '1'), the MGT is placed in a low power state. A reset must be applied to clear. • With the TBI version this bit is unused. Bit[3]: **Isolate** When set to '1,' the GMII should be electrically isolated. When set to '0,' normal operation is enabled.

1. This signal is synchronous to the internal 125 MHz reference clock of the core. This is `userclk2` when the core is used with the device-specific transceiver; this is `gtx_clk` when the core is used with TBI.

Auto-Negotiation Signal Pinout

Table 2-5 describes the signals present when the optional Auto-Negotiation functionality is present. For more information, see Chapter 10, "Auto-Negotiation."

Table 2-5: **Optional Auto-Negotiation Interface Signal Pinout**

Signal	Direction	Description
link_timer_value[8:0][1]	Input	Used to configure the duration of the Auto-Negotiation Link Timer period. The duration of this timer is set to the binary number input into this port multiplied by 4096 clock periods of the 125 MHz reference clock (8 ns). It is expected that this signal will be tied off to a logical value. This port is replaced when using the dynamic switching mode.
an_interrupt[1]	Output	Active high interrupt to signal the completion of an Auto-Negotiation cycle. This interrupt can be enabled/disabled and cleared by writing to the appropriate PCS Management Register.

1. These signals are synchronous to the internal 125 MHz reference clock of the core. This is userclk2 when the core is used with the device-specific transceiver; this is gtx_clk when the core is used with TBI.

Dynamic Switching Signal Pinout

Table 2-6 describes the signals present when the optional Dynamic Switching mode (between 1000BASE-X and SGMII standards) is selected. In this case, the MDIO (Table 2-3) and device-specific transceiver (Table 2-7) interfaces are always present.

Table 2-6: **Optional Dynamic Standard Switching Signals**

Signal	Direction	Description
link_timer_basex[8:0][1]	Input	Used to configure the duration of the Auto-Negotiation Link Timer period when performing the 1000BASE-X standard. The duration of this timer is set to the binary number input into this port multiplied by 4096 clock periods of the 125 MHz reference clock (8 ns). It is expected that this signal will be tied off to a logical value.

Table 2-6: **Optional Dynamic Standard Switching Signals**

Signal	Direction	Description
link_timer_sgmii[8:0][1]	Input	Used to configure the duration of the Auto-Negotiation Link Timer period when performing the SGMII standard. The duration of this timer is set to the binary number input into this port multiplied by 4096 clock periods of the 125 MHz reference clock (8 ns). It is expected that this signal will be tied off to a logical value.
basex_or_sgmii[1]	Input	Used as the reset default to select the standard. It is expected that this signal will be tied off to a logical value. '0' signals that the core will come out of reset operating as 1000BASE-X. '1' signals that the core will come out of reset operating as SGMII. **Note**: The standard can be set following reset through the MDIO Management.

1. Clock domain is userclk2.

Physical Side Interface

1000BASE-X PCS with PMA Using Device-Specific Transceiver Signal Pinout (Optional)

Table 2-7 describes the optional interface to the device-specific transceiver. The core is connected to a device-specific transceiver in the appropriate HDL example design delivered with the core. For more information, see:

- Chapter 7, "1000BASE-X with Transceivers"
- Chapter 8, "SGMII / Dynamic Standards Switching with Transceivers"

Table 2-7: **Optional Transceiver Interface Pinout**

Signal	Direction	Description
mgt_rx_reset[1]	Output	Reset signal issued by the core to the device-specific transceiver receiver path. Connect to RXRESET signal of device-specific transceiver.
mgt_tx_reset[1]	Output	Reset signal issued by the core to the device-specific transceiver transmitter path. Connect to TXRESET signal of device-specific transceiver.
userclk	Input	Also connected to TXUSRCLK and RXUSRCLK of the device-specific transceiver. Clock domain is not applicable.
userclk2	Input	Also connected to TXUSRCLK2 and RXUSRCLK2 of the device-specific transceiver. Clock domain is not applicable.

Table 2-7: **Optional Transceiver Interface Pinout *(Continued)***

Signal	Direction	Description
dcm_locked	Input	A DCM may be used to derive userclk and userclk2. This is implemented in the HDL design example delivered with the core. The core will use this input to hold the device-specific transceiver in reset until the DCM obtains lock. Clock domain is not applicable.
rxbufstatus[1:0][1]	Input	Connect to device-specific transceiver signal of the same name.
rxchariscomma[1]	Input	Connects to device-specific transceiver signal of the same name.
rxcharisk[1]	Input	Connects to device-specific transceiver signal of the same name.
rxclkcorcnt[2:0][1]	Input	Connect to device-specific transceiver signal of the same name.
rxdata[7:0][1]	Input	Connect to device-specific transceiver signal of the same name.
rxdisperr[1]	Input	Connects to device-specific transceiver signal of the same name.
rxnotintable[1]	Input	Connects to device-specific transceiver signal of the same name.
rxrundisp[1]	Input	Connects to device-specific transceiver signal of the same name.
txbuferr[1]	Input	Connects to device-specific transceiver signal of the same name.
powerdown[1]	Output	Connects to device-specific transceiver signal of the same name.
txchardispmode[1]	Output	Connects to device-specific transceiver signal of the same name.
txchardispval[1]	Output	Connects to device-specific transceiver signal of the same name.
txcharisk[1]	Output	Connects to device-specific transceiver signal of the same name.
txdata[7:0][1]	Output	Connect to device-specific transceiver signal of the same name.
enablealign[1]	Output	Allows the transceivers to serially realign to a comma character. Connects to ENMCOMMAALIGN and ENPCOMMAALIGN of the device-specific transceiver.

1. When the core is used with a device-specific transceiver, `userclk2` is used as the 125 MHz reference clock for the entire core.

1000BASE-X PCS with TBI Pinout

Table 2-8 describes the optional TBI signals, used as an alternative to the device-specific transceiver receiver interface. The appropriate HDL example design delivered with the core connects these signals to IOBs to provide an external TBI suitable for connection to an off-chip PMA SERDES device. When the core is used with the TBI, gtx_clk is used as the 125 MHz reference clock for the entire core. For more information, see Chapter 6, "The Ten-Bit Interface."

Table 2-8: **Optional TBI Interface Signal Pinout**

Signal	Direction	Clock Domain	Description
gtx_clk	Input	N/A	Clock signal at 125 MHz. Tolerance must be within *IEEE 802.3* specification.
tx_code_group[9:0]	Output	gtx_clk	10-bit parallel transmit data to PMA Sublayer (SERDES).
loc_ref	Output	N/A	Causes the PMA sublayer clock recovery unit to lock to pma_tx_clk. This signal is currently tied to Ground.
ewrap	Output	gtx_clk	When '1,' this indicates to the external PMA SERDES device to enter loopback mode. When '0,' this indicates normal operation
rx_code_group0[9:0]	Input	pma_rx_clk0	10-bit parallel received data from PMA Sublayer (SERDES). This is synchronous to pma_rx_clk0.
rx_code_group1[9:0]	Input	pma_rx_clk1	10-bit parallel received data from PMA Sublayer (SERDES). This is synchronous to pma_rx_clk1.
pma_rx_clk0	Input	N/A	Received clock signal from PMA Sublayer (SERDES) at 62.5 MHz.
pma_rx_clk1	Input	N/A	Received clock signal from PMA Sublayer (SERDES) at 62.5 MHz. This is 180 degrees out of phase with pma_rx_clk0.
en_cdet	Output	gtx_clk	Enables the PMA Sublayer to perform comma realignment. This is driven from the PCS Receive Engine during the *Loss-Of-Sync* state.

Chapter 3

Generating and Customizing the Core

The Ethernet 1000BASE-X PCS/PMA or SGMII core is generated using the CORE Generator™ software. This chapter describes the GUI options used to generate and customize the core.

GUI Interface

Figure 3-1 displays the Ethernet 1000BASE-X PCS/PMA or SGMII customization screen, used to set core parameters and options. For help starting and using CORE Generator software on your system, see the documentation included with the ISE® software, including the *CORE Generator Guide*, at www.xilinx.com/support/software_manuals.htm.

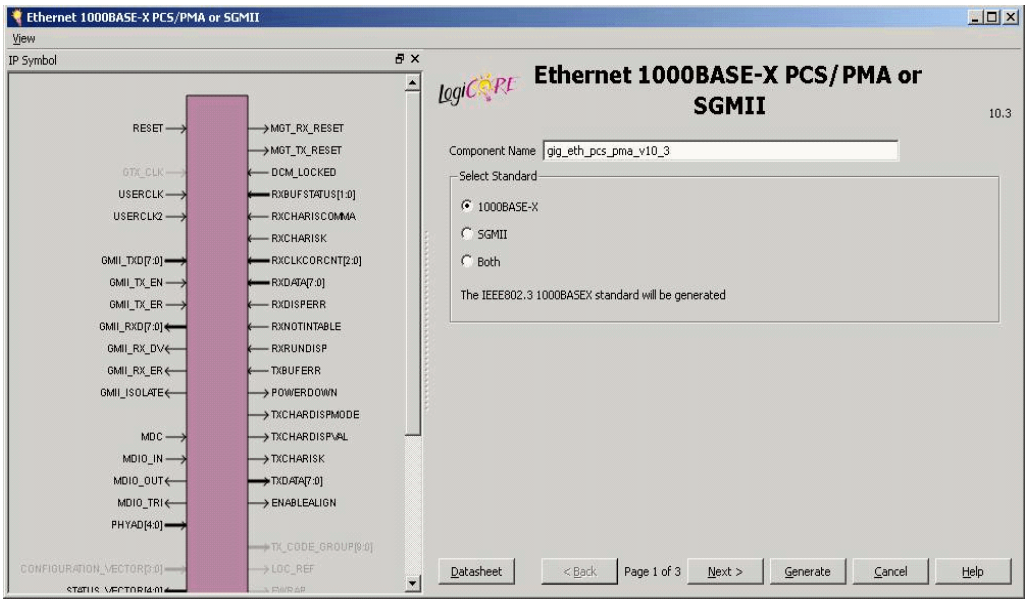


Figure 3-1: **Core Customization Screen**

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 "_."

Select Standard

Select from the following standards for the core:

- **1000BASE-X**. 1000BASE-X Physical Coding Sublayer (PCS) functionality is designed to the IEEE 802.3 specification. Depending on the choice of physical interface, the functionality may be extended to include the 1000BASE-X Physical Medium Attachment (PMA) sublayer. Default setting.
- **SGMII**. Provides the functionality to provide a Gigabit Media Independent Interface (GMII) to Serial-GMII (SGMII) bridge, as defined in the Serial-GMII Specification (Cisco Systems, ENG-46158). SGMII may be used to replace GMII at a much lower pin count and for this reason often favored by PCB designers.
- **Both** (a combination of 1000BASE-X and SGMII). Combining the 1000BASE-X and SGMII standards lets you dynamically configure the core to switch between 1000BASE-X and SGMII standards. The core can be switched by writing through the MDIO Management Interface. For more information, see Chapter 9, "Configuration and Status."

Core Functionality

Figure 3-2 displays the Ethernet 1000BASE-X PCS/PMA or SGMII functionality screen.

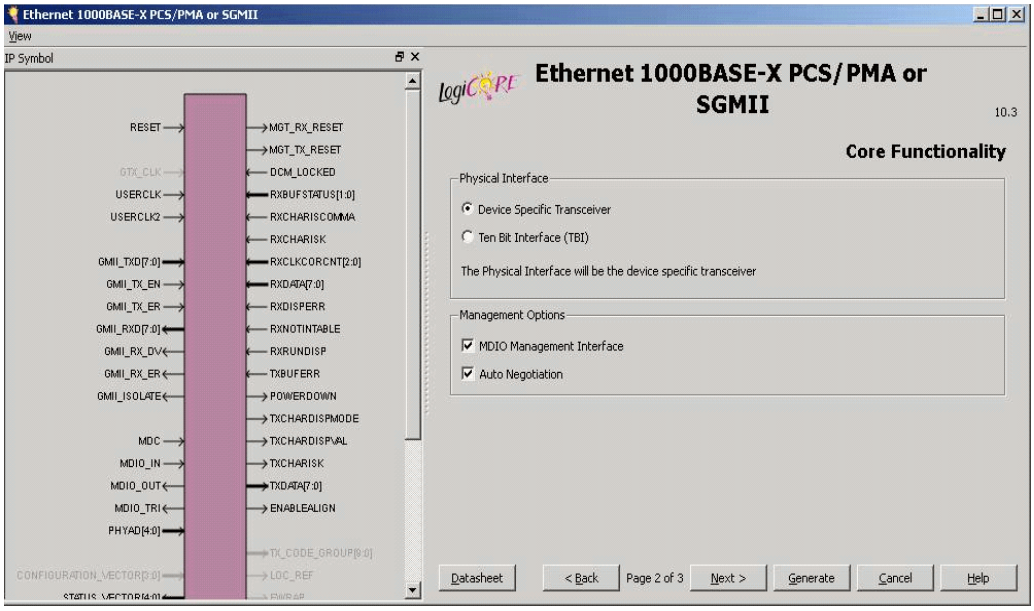


Figure 3-2: **1000BASE-X Standard Options Screen**

Physical Interface

Depending on the target architecture, two physical interface options are available for the core.

- **Transceiver**. Uses a transceiver specific to the selected device family to extend the 1000BASE-X functionality to include both PCS and PMA sub-layers. For this reason, it is available only for Virtex®-4 FX, Virtex-5 LXT, Virtex-5 SXT, Virtex-5 FXT and Virtex-5 TXT, Spartan®-6 LXT and selective Virtex-6 devices. For additional information, see "Transceiver Logic" in Chapter 7.
- **Ten Bit Interface (TBI)**. Available in all supported families and provides 1000BASE-X or SGMII functionality with a parallel TBI used to interface to an external SERDES. For more information, see "Ten-Bit-Interface Logic" in Chapter 6. Default setting.

MDIO Management Interface

Select this option to include the MDIO Management Interface to access the PCS Configuration Registers. See "MDIO Management Interface" in Chapter 9.

If this option is not selected, the core is generated with a replacement configuration vector. See "Optional Configuration Vector" in Chapter 9. The Management Interface is selected by default.

Auto-Negotiation

Select this option to include Auto-Negotiation functionality with the core, available only if the core includes the optional Management Interface. For more information, see Chapter 10, "Auto-Negotiation." The default is to include Auto-Negotiation.

SGMII/Dynamic Standard Switching Elastic Buffer Options

The SGMII/Dynamic Standard Switching Options screen, used to customize the Ethernet 1000BASE-X PCS/PMA or SGMII core, is *only* displayed if either SGMII or Both is selected in the Select Standard section of the initial customization screen, and *only* if the device-specific transceiver is selected as the Physical Standard.

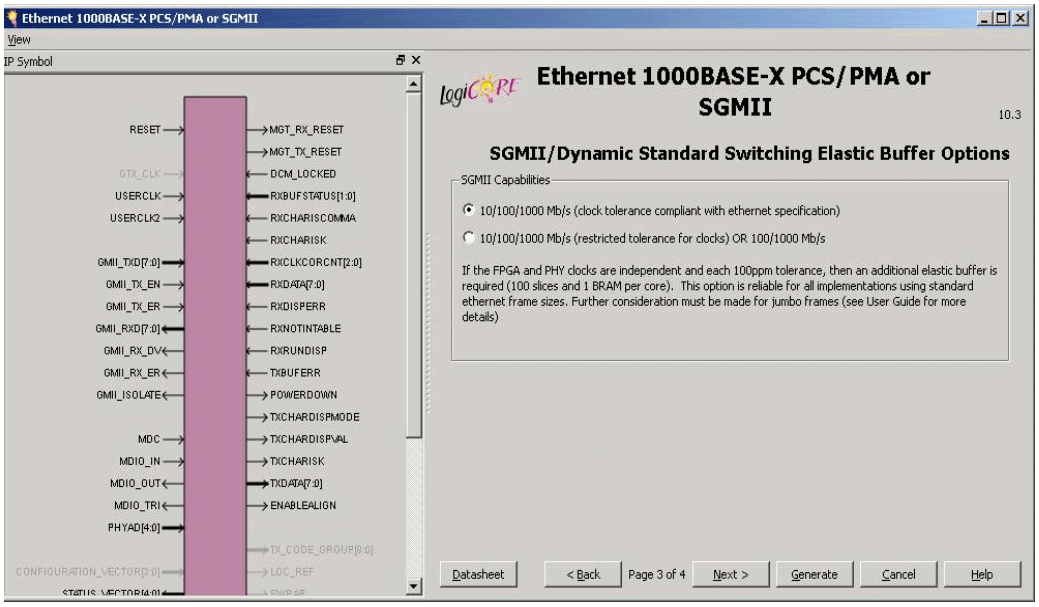


Figure 3-3: **SGMII/Dynamic Standard Switching Options Screen**

This screen lets you select the Receiver Elastic Buffer type to be used with the core. Before selecting this option, see "Receiver Elastic Buffer Implementations" in Chapter 8.

Transceiver Tile Configuration

The Transceiver Tile Configuration screen is only displayed if the transceiver interface is used with selective Virtex-4, Virtex-5 and Spartan-6 device families.

Transceivers for Virtex-4 FX, Virtex-5 and Spartan-6 device families are available in tiles, each tile consisting of a pair of transceivers. The Transceiver Tile Selection has no effect on the functionality of the core netlist, but determines the functionality of the example design delivered with the core.

Depending on the option selected, the example design instantiates a single core netlist and does one of the following:

- **MGT A (0)**. Connects to device-specific transceiver A
- **MGT B (1)**. Connects to device-specific transceiver B

Both MGTs. Two instantiations of the core are created in the example design and connected to both device-specific transceiver A and B.

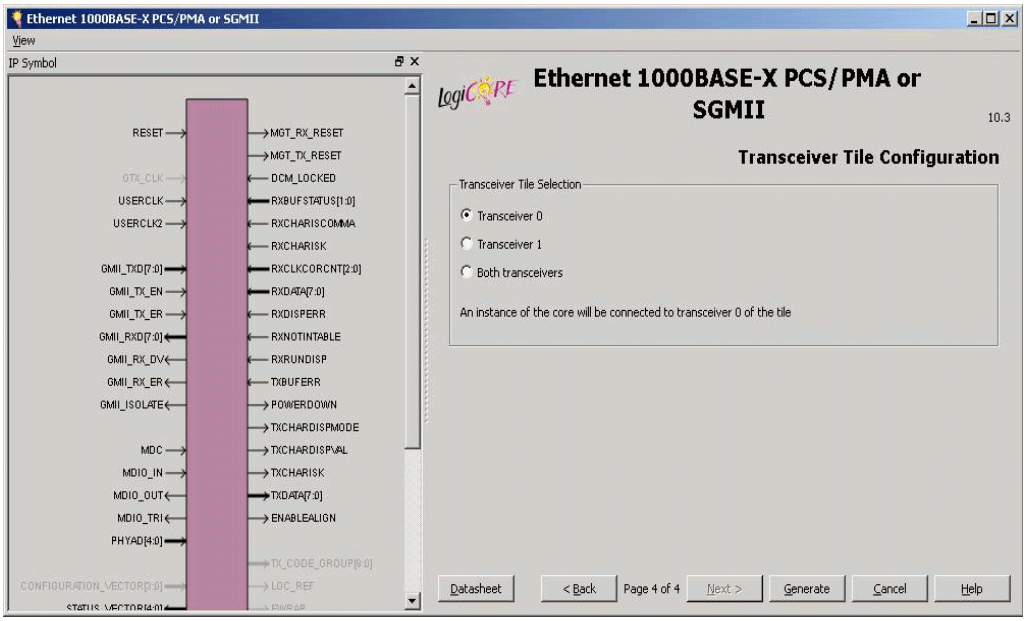


Figure 3-4: **Transceiver Tile Configuration Screen**

Parameter Values in the XCO File

XCO file parameters are used to run the CORE Generator software from the command line. XCO file parameter names and their values are similar to the names and values shown in the GUI, except that underscore characters (_) may be used instead of spaces. The text in an XCO file is not case sensitive.

Table 3-1 describes the XCO file parameters, values and summarizes the GUI defaults. The following is an example of the CSET parameters in an XCO file:

```
CSET component_name=gig_eth_pcs_pma_v10_3
CSET standard=1000BASEX
CSET physical_interface=TBI
CSET management_interface=true
CSET auto_negotiation=true
CSET sgmii_mode=10_100_1000
CSET RocketIO_tile=A
```

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

Parameter	XCO File Values	Default GUI Setting
component_name	ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _	gig_eth_pcs_pma_v10_3
standard	One of the following keywords: 1000BASEX, SGMII, Both	1000BASEX
physical_interface	One of the following keywords: TBI, RocketIO	TBI
management_interface	One of the following keywords: true, false	true
auto_negotiation	One of the following keywords: true, false	true
sgmii_mode	One of the following keywords: 10_100_1000, 100_1000 • 10_100_1000 corresponds to “10/100/1000 Mbps (clock tolerance compliant with Ethernet specification)“ • 100_1000 corresponds to “10/100/1000 Mbps (restricted tolerance for clocks) OR 100/1000 Mbps“	10_100_1000
RocketIO_tile	One of the following keywords: A, B, Both	A

Output Generation

The files output by the CORE Generator software are placed in the CORE Generator software project directory and include the following:

- The netlist file for the core
- Supporting CORE Generator software files
- Release notes and documentation
- Subdirectories containing an HDL example design
- Scripts to run the core through the back-end tools and simulate the core using either Mentor Graphics ModelSim, Cadence IUS, and Synopsys simulators

See the *Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started Guide* for a complete description of the CORE Generator software output files, simulation requirements, and detailed information about the HDL example design.

XILINX®

Chapter 4

Designing with the Core

This chapter provides information about creating your own designs using the Ethernet 1000BASE-X PCS/PMA or SGMII core. Design guidelines, as well as the variety of implementations presented, are based on the example design delivered with the core. See the *Xilinx Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started Guide* for information about the example design delivered with the core.

Not all implementations require all of the design steps defined in this chapter. Carefully follow the provided logic design guidelines to ensure success.

Design Overview

An HDL example design built around the core is provided through the CORE Generator™ software and allows for a demonstration of core functionality using either a simulation package or in hardware if placed on a suitable board. Four implementations of the core, based on the provided example design, are illustrated in the following sections.

- "1000BASE-X Standard Using Device-Specific Transceiver Example Design"
- "1000BASE-X Standard with TBI Example Design"
- "SGMII Standard Using a Device-Specific Transceiver Example Design"
- "SGMII Standard with TBI Transceiver Example Design"

1000BASE-X Standard Using Device-Specific Transceiver Example Design

Figure 4-1 illustrates the example design in 1000BASE-X mode using the following:

- Virtex®-4 FPGA RocketIO™ MGT transceiver
- Virtex-5 FPGA RocketIO GTP transceiver
- Virtex-5 FPGA RocketIO GTX transceiver
- Virtex-6 FPGA GTX transceiver
- Spartan®-6 FPGA GTP transceiver

As illustrated, the example is split between two hierarchical layers. The block level is designed so that it can be instantiated directly into your design and performs the following functions:

- Instantiates the core from HDL
- Connects the physical-side interface of the core to a device-specific transceiver

The top level of the example design creates a specific example that can be simulated, synthesized, implemented, and if required, placed on a suitable board and demonstrated in hardware. The top level of the example design performs the following functions:

- Instantiates the block level from HDL
- Derives the clock management logic for a device-specific transceiver and the core
- Implements an external GMII

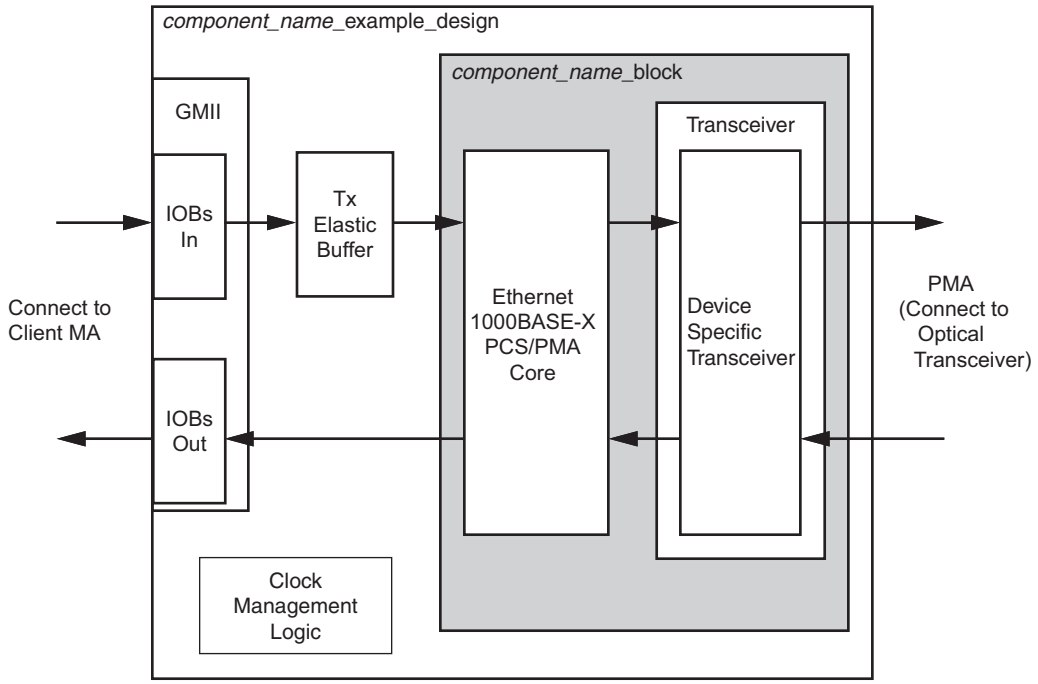


Figure 4-1: **1000BASE-X Standard Using a Device-Specific Transceiver**

1000BASE-X Standard with TBI Example Design

Figure 4-2 illustrates the example design in 1000BASE-X mode using a TBI. As illustrated, the example is split between two hierarchical layers. The block level is designed so that it can be instantiated directly into customer designs and performs the following functions:

- Instantiates the core from HDL
- Connects the physical-side interface of the core to device IOBs, creating an external TBI. See Chapter 6, "The Ten-Bit Interface."

The top level of the example design creates a specific example that can be simulated, synthesized, implemented, and if required, placed on a suitable board and demonstrated in hardware. The top level of the example design performs the following functions:

- Instantiates the block level from HDL
- Derives the clock management logic for the core
- Implements an external GMII

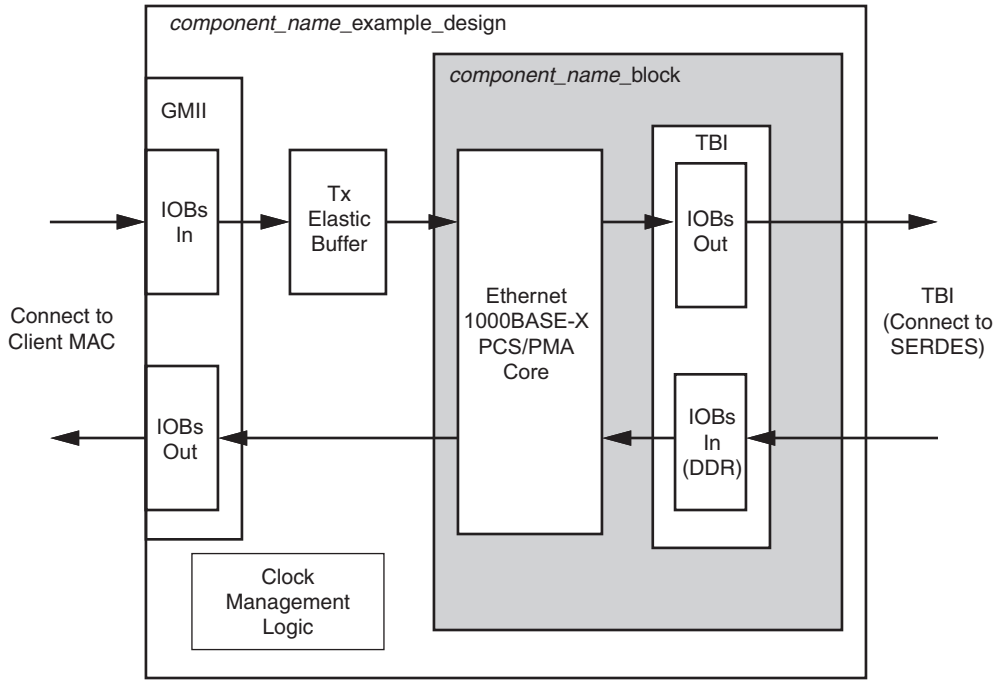


Figure 4-2: **Example Design 1000BASE-X Standard Using TBI**

SGMII Standard Using a Device-Specific Transceiver Example Design

Figure 4-3 illustrates the example design in SGMII mode using the following:

- Virtex-4 FPGA RocketIO MGT transceiver
- Virtex-5 FPGA RocketIO GTP transceiver
- Virtex-5 FPGA RocketIO GTX transceiver
- Virtex-6 FPGA GTX transceiver
- Spartan-6 FPGA GTP transceiver

This is also the example design created when the Dynamic Switching capability between SGMII and 1000BASE-X standards is present. As illustrated, the example is split between two hierarchical layers. The block level is designed so that it can be instantiated directly into customer designs and performs the following functions:

- Instantiates the core from HDL
- Connects the physical-side interface of the core to a device-specific transceiver
- Connects the client side GMII of the core to an SGMII Adaptation Module, which provides the functionality to operate at speeds of 1 Gbps, 100 Mbps and 10 Mbps

The top level of the example design creates a specific example which can be simulated, synthesized and implemented. The top level of the example design performs the following functions:

- Instantiates the block level from HDL
- Derives the clock management logic for device-specific transceiver and the core
- Implements an external GMII-style interface

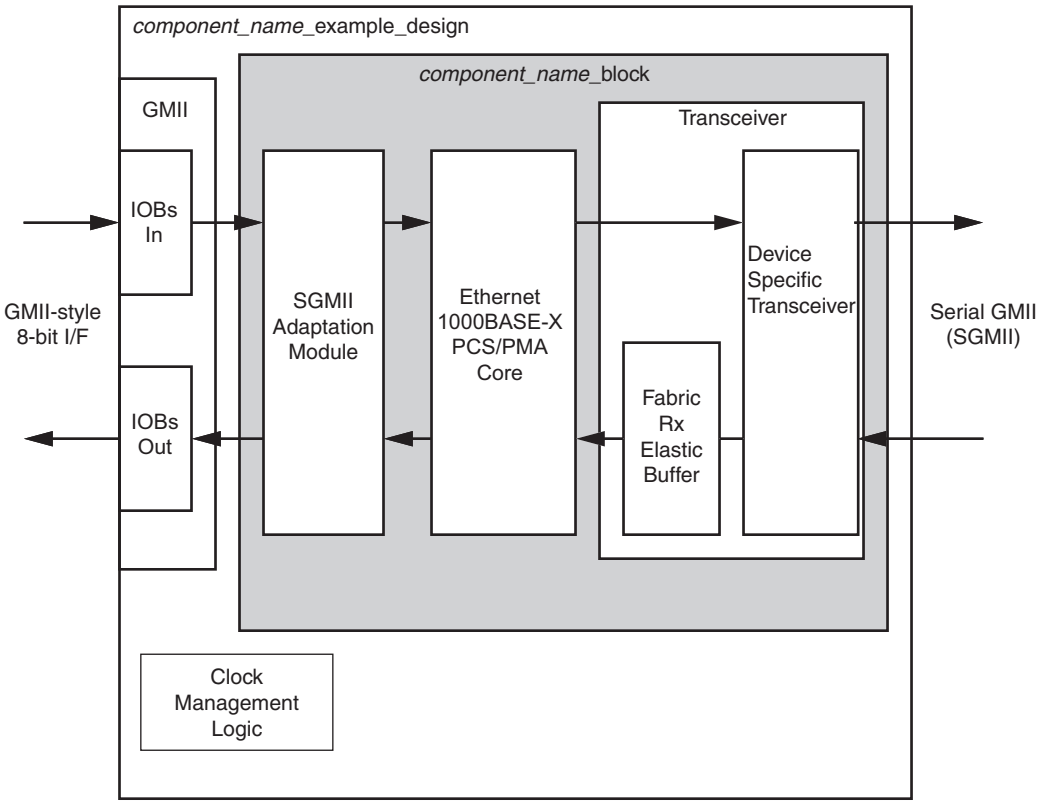


Figure 4-3: **Example Design Performing the SGMII Standard**

SGMII Standard with TBI Transceiver Example Design

Figure 4-3 illustrates the example design with the SGMII standard using a TBI. This is also the example design created when the Dynamic Switching capability between SGMII and 1000BASE-X standards is present. As illustrated, the example is split between two hierarchical layers. The block level is designed so that it can be instantiated directly into customer designs and performs the following functions:

- Instantiates the core from HDL
- Connects the physical-side interface of the core to device IOBs, creating an external TBI. See Chapter 6, "The Ten-Bit Interface."
- Connects the client side GMII of the core to an SGMII Adaptation Module, which provides the functionality to operate at speeds of 1 Gbps, 100 Mbps and 10 Mbps

The top level of the example design creates a specific example which can be simulated, synthesized and implemented. The top level of the example design performs the following functions:

- Instantiates the block level from HDL
- Derives the clock management logic for the core
- Implements an external GMII-style interface

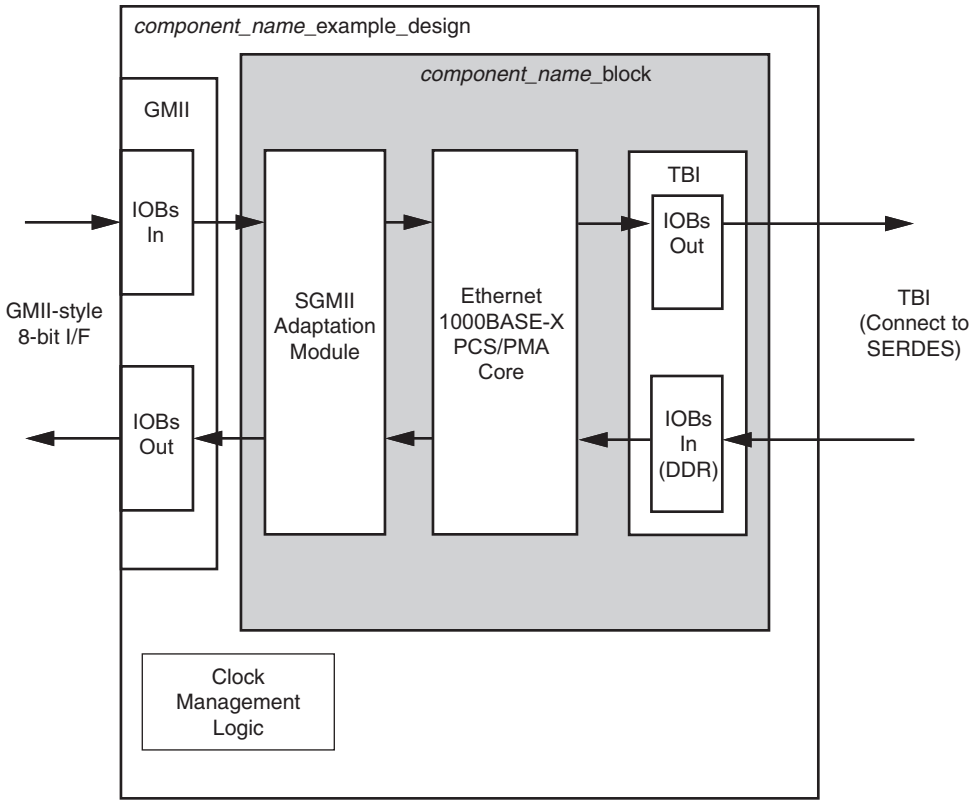


Figure 4-4: **Example Design Performing the SGMII Standard**

Design Guidelines

Generate the Core

Generate the core using the CORE Generator software, as described in Chapter 3, "Generating and Customizing the Core."

Examine the Example Design Provided with the Core

Before implementing the core in your application, examine the example design provided with the core to identify the steps that can be performed:

- Edit the HDL top level of the example design file to change the clocking scheme, add or remove IOBs as required, and replace the GMII IOB logic with user-specific application logic (for example, an Ethernet MAC).
- Synthesize the entire design.

  The Xilinx Synthesis Tool (XST) script and Project file in the `/implement/vhdl` (or `/implement/verilog`) directory may be adapted to include any added user HDL files.
- Run the implement script in the `/implement` directory to create a top-level netlist for the design.

  The script may also run the Xilinx tools **`map`**, **`par`**, and **`bitgen`** to create 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 in `/test/vhdl (or /test/verilog)` as a template.
- Download the bitstream to a target device.

Implement the Ethernet 1000BASE-X PCS/PMA or SGMII Core in Your Application

Before implementing your application, examine the example design delivered with the core for information about the following:

- Instantiating the core from HDL
- Connecting the physical-side interface of the core (device-specific transceiver or TBI)
- Deriving the clock management logic

It is expected that the block level module from the example design will be instantiated directly into customer designs rather than the core netlist itself. The block level contains the core and a completed physical interface.

Write an HDL Application

After reviewing the example design delivered with the core, write an HDL application that uses single or multiple instances of the block level module for the Ethernet 1000BASE-X PCS/PMA or SGMII core. Client-side interfaces and operation of the core are described in Chapter 5, "Using the Client-Side GMII Data Path." See the following information for additional details:

- Using the Ethernet 1000BASE-X PCS/PMA or SGMII core in conjunction with the Tri-Mode Ethernet MAC core in Chapter 13, "Interfacing to Other Cores".

Synthesize your Design

Synthesize your entire design using the desired synthesis tool. The Ethernet 1000BASE-X PCS/PMA or SGMII core is pre-synthesized and delivered as an NGC netlist—for this reason, it appears as a black box to synthesis tools.

Create a Bitstream

Run the Xilinx tools **`map`**, **`par`**, and **`bitgen`** to create a bitstream that can be downloaded to a Xilinx device. The UCF produced by the CORE Generator software should be used as the basis for your UCF and care must be taken to constrain the design correctly. See Chapter 12, "Constraining the Core" for more information.

Simulate and Download your Design

After creating a bitstream that can be downloaded to a Xilinx device, simulate the entire design and download it to the desired device.

Know the Degree of Difficulty

An Ethernet 1000BASE-X PCS/PMA or SGMII core is challenging to implement in any technology and as such, all Ethernet 1000BASE-X PCS/PMA or SGMII core applications require careful attention to system performance requirements. Pipelining, logic mapping, placement constraints, and logic duplication are all methods that help boost system performance.

Review Table 4-1 to determine the relative level of difficulty associated with different designs. This relates to meeting the required system clock frequency of 125 MHz for the core.

Table 4-1: **Degree of Difficulty for Various Implementations**

Device Family	Difficulty
Virtex-4	Easy
Virtex-5	Easy
Virtex-6	Easy
Spartan-3	Difficult
Spartan-3E	Difficult
Spartan-3A	Difficult
Spartan-6	Difficult

Keep it Registered

To simplify timing and to increase system performance in an FPGA design, keep all inputs and outputs registered between the user application and the core. All inputs and outputs from the user application should come *from*, or connect *to*, a flip-flop. While registering signals may 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 for the core identifies the critical signals and the timing constraints that should be applied. See Chapter 12, "Constraining the Core"for more information.

Use Supported Design Flows

The core is pre-synthesized and is delivered as an NGC netlist. The example implementation scripts provided currently uses ISE® v11.3 tools as the synthesis tool for the HDL example design delivered with the core. Other synthesis tools may be used for the user application logic. The core will always be unknown to the synthesis tool and should appear as a black box. Post synthesis, only ISE v11.3 tools are supported.

Make Only Allowed Modifications

The Ethernet 1000BASE-X PCS/PMA or SGMII core should not be modified. Modifications may have adverse effects on system timing and protocol compliance. Supported user configurations of the Ethernet 1000BASE-X PCS/PMA or SGMII core can only be made by the selecting the options from within the CORE Generator software when the core is generated. See Chapter 3, "Generating and Customizing the Core."

Chapter 5

Using the Client-Side GMII Data Path

This chapter provides general guidelines for creating designs using client-side GMII of the Ethernet 1000BASE-X PCS/PMA or SGMII core.

Designing with the Client-side GMII for the 1000BASE-X Standard

It is not within the scope of this document to define the Gigabit Media Independent Interface (GMII)— see clause 35 of the *IEEE 802.3* specification for information about the GMII. Timing diagrams and descriptions are provided only as an informational guide.

GMII Transmission

This section includes figures that illustrate GMII transmission. In these figures the clock is not labeled. The source of this clock signal varies, depending on the options selected when the core is generated. For more information on clocking, see Chapters 6, 7 and 8.

Normal Frame Transmission

Normal outbound frame transfer timing is illustrated in Figure 5-1. This figure shows that an Ethernet frame is proceeded by an 8-byte preamble field (inclusive of the Start of Frame Delimiter (SFD)), and completed with a 4-byte Frame Check Sequence (FCS) field. This frame is created by the MAC connected to the other end of the GMII. The PCS logic itself does not recognize the different fields within a frame and will treat any value placed on `gmii_txd[7:0]` within the `gmii_tx_en` assertion window as data.

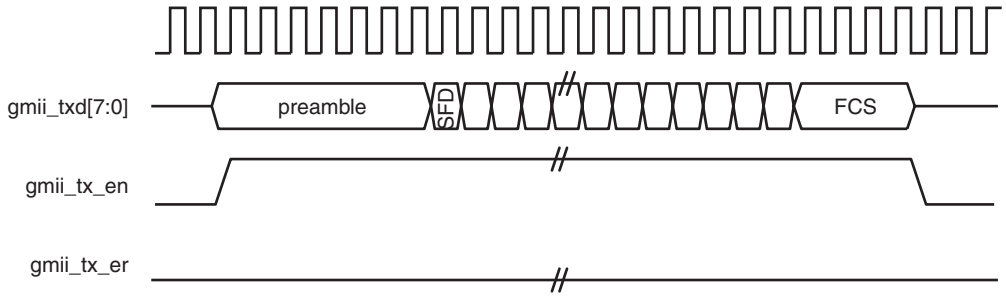


Figure 5-1: **GMII Normal Frame Transmission**

Error Propagation

A corrupted frame transfer is illustrated in Figure 5-2. An error may be injected into the frame by asserting `gmii_tx_er` at any point during the `gmii_tx_en` assertion window. The core ensures that all errors are propagated through both transmit and receive paths so that the error is eventually detected by the link partner.

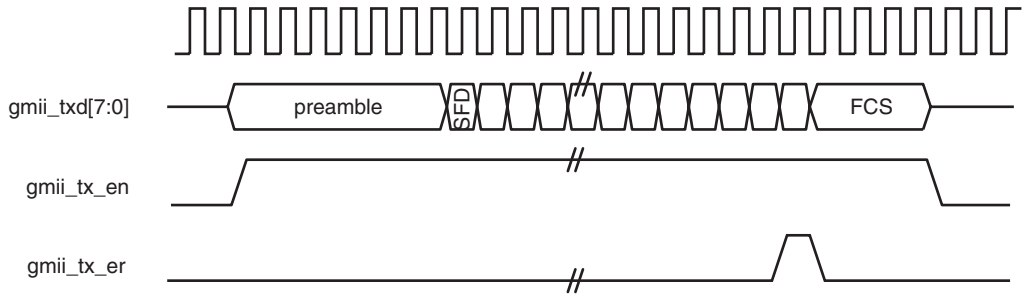


Figure 5-2: **GMII Error Propagation Within a Frame**

GMII Reception

This section includes figures that illustrate GMII reception. In these figures the clock is not labelled. The source of this clock signal will vary, depending on the options used when the core is generated. For more information on clocking, see Chapters 6, 7 and 8.

Normal Frame Reception

The timing of normal inbound frame transfer is illustrated in Figure 5-3. This shows that Ethernet frame reception is proceeded by a preamble field. The *IEEE 802.3* specification (see clause 35) allows for up to all of the seven preamble bytes that proceed the Start of Frame Delimiter (SFD) to be lost in the network. The SFD will always be present in well-formed frames.

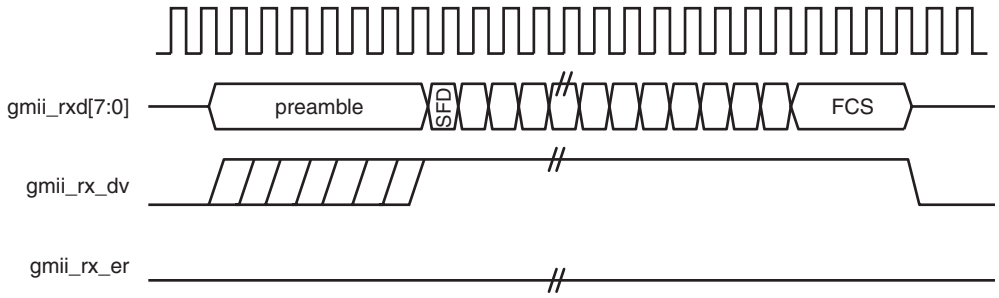


Figure 5-3: **GMII Normal Frame Reception**

Normal Frame Reception with Extension Field

In accordance with the *IEEE 802.3*, clause 36, state machines for the 1000BASE-X PCS, `gmii_rx_er` may be driven high following reception of the end frame in conjunction with `gmii_rxd[7:0]` containing the hexadecimal value of 0x0F to signal carrier extension. This is illustrated in Figure 5-4. See Appendix D, "1000BASE-X State Machines" for more information.

This is not an error condition and may occur even for full-duplex frames.

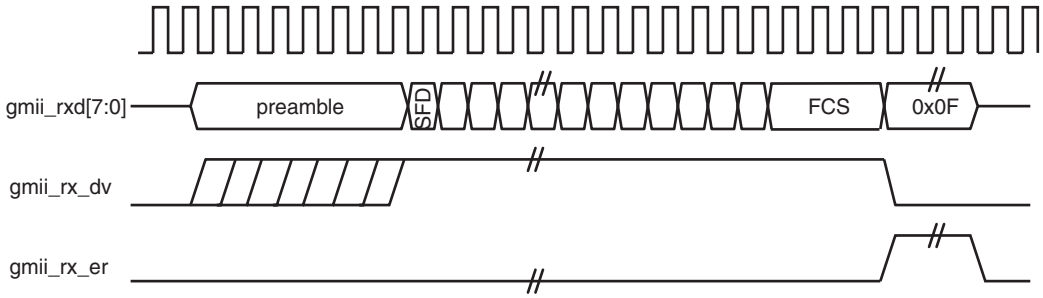


Figure 5-4: **GMII Normal Frame Reception with Carrier Extension**

Frame Reception with Errors

The signal `gmii_rx_er` when asserted within the assertion window signals that a frame was received with a detected error (Figure 5-5). In addition, a late error may also be detected during the Carrier Extension interval. This is indicated by `gmii_rxd[7:0]` containing the hexadecimal value 0x1F, also illustrated in Figure 5-5.

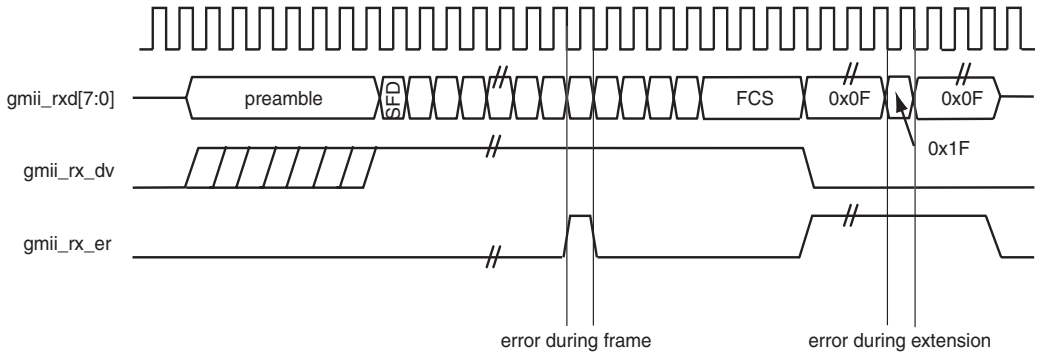


Figure 5-5: **GMII Frame Reception with Errors**

False Carrier

Figure 5-6 illustrates the GMII signaling for a False Carrier condition. False Carrier is asserted by the core in response to certain error conditions, such as a frame with a corrupted start code, or for random noise.

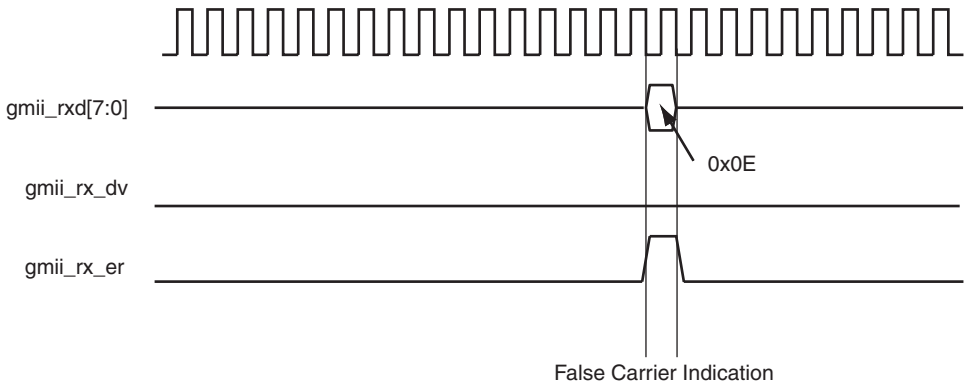


Figure 5-6: **False Carrier Indication**

status_vector[7:0] signals

Bit[0]: Link Status

This signal indicates the status of the link. This information is duplicated in the optional PCS Management Registers, if present ("Status Register (Register 1)," bit 1.2). However, this would always serve a useful function as a Link Status LED.

When high, the link is valid: synchronization of the link has been obtained **and** Auto-Negotiation (if present and enabled) has completed.

When low, a valid link has not been established. Either link synchronization has failed or Auto-Negotiation (if present and enabled) has failed to complete.

Bit[1]: Link Synchronization

This signal indicates the state of the synchronization state machine (IEEE802.3 figure 36-9). This signal is similar to Bit[0] (Link Status), but is NOT qualified with Auto-Negotiation.

When high, link synchronization has been obtained.

When low, synchronization has failed.

Bit[7]: PHY Link Status (SGMII mode only)

When operating in SGMII mode, this bit represents the link status of the external PHY device attached to the other end of the SGMII link. **However, this bit is only valid after successful completion of Auto-Negotiation across the SGMII link**. If SGMII Auto-Negotiation is disabled, then the status of this bit should be ignored.

- When high, the PHY has obtained a link with its link partner;
- When low, the PHY has not linked with its link partner.

When operating in 1000BASE-X mode this bit will remain low and should be ignored

Bits[6:2]: Code Group Reception Indicators

These signals indicate the reception of particular types of group, as defined in the following subsections. Figure 5-7 illustrates the timing of these signals, showing that they are aligned with the code groups themselves, as they appear on the output `gmii_rxd[7:0]` port.

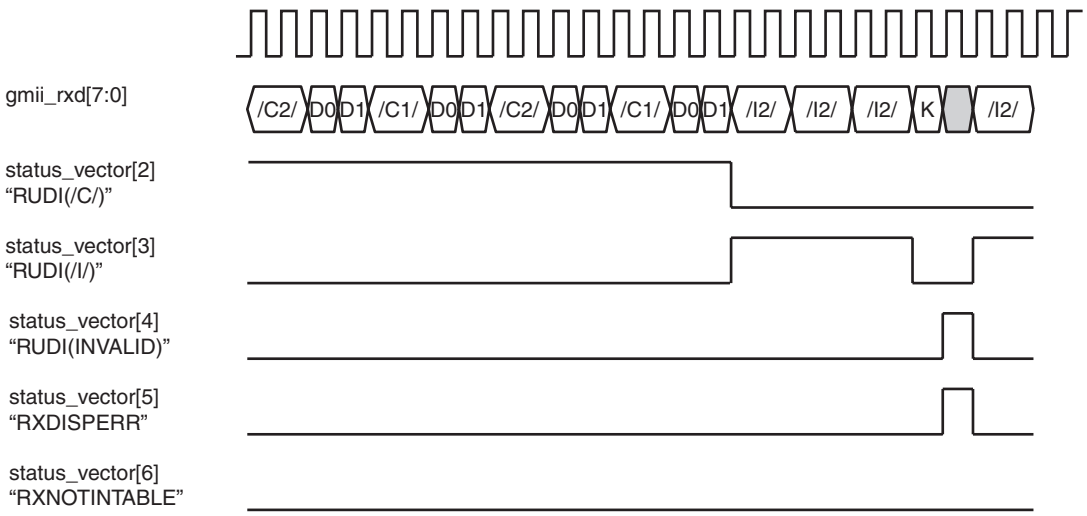


Figure 5-7: **status_vector[4:2] timing**

Bit[2]: RUDI(/C/)

The core is receiving /C/ ordered sets (Auto-Negotiation Configuration sequences) as defined in IEEE802.3 clause 36.2.4.10.

Bit[3]: RUDI(/I/)

The core is receiving /I/ ordered sets (Idles) as defined in IEEE802.3 clause 36.2.4.12.

Bit[4]: RUDI(INVALID)

The core has received invalid data whilst receiving/C/ or /I/ ordered set as defined in IEEE802.3 clause 36.2.5.1.6. This may be caused, for example, by bit errors occurring in any clock cycle of the /C/ or /I/ ordered set: Figure 5-7 illustrates an error occurring in the second clock cycle of an /I/ idle sequence.

Bit[5]: RXDISPERR

The core has received a running disparity error during the 8B10B decoding function. Figure 5-7 illustrates a running disparity error occurring in the second clock cycle of an /I/ idle sequence.

Bit[6]: RXNOTINTABLE

The core has received a code group which is not recognized from the 8B10B coding tables. If this error is detected, the timing of the RXNOTINTABLE signal would be identical to that of the RXDISPERR signal illustrated in Figure 5-7.

Designing with Client-side GMII for the SGMII Standard

Overview

When the core is generated for the SGMII standard, changes are made to the core that affect the PCS Management Registers and the Auto-Negotiation function (see "Select Standard" in Chapter 3). However, the data path through both transmitter and receiver sections of the core remains unchanged.

GMII Transmission

1 Gigabit per Second Frame Transmission

The timing of normal outbound frame transfer is illustrated in Figure 5-8. At 1 Gbps speed, the operation of the transmitter GMII signals remains identical to that described in "Designing with the Client-side GMII for the 1000BASE-X Standard."

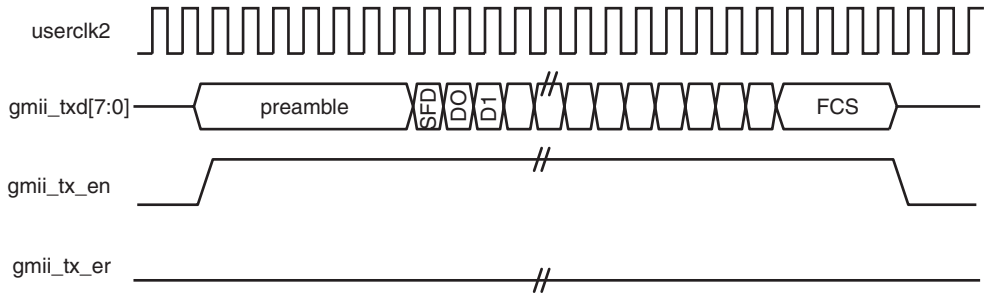


Figure 5-8: **GMII Frame Transmission at 1 Gbps**

100 Megabit per Second Frame Transmission

The operation of the core remains unchanged. It is the responsibility of the client logic (for example, an Ethernet MAC) to enter data at the correct rate. When operating at a speed of 100 Mbps, every byte of the MAC frame (from preamble field to the Frame Check Sequence field, inclusive) should each be repeated for 10 clock periods to achieve the desired bit rate, as illustrated in Figure 5-9. It is also the responsibility of the client logic to ensure that the interframe gap period is legal for the current speed of operation.

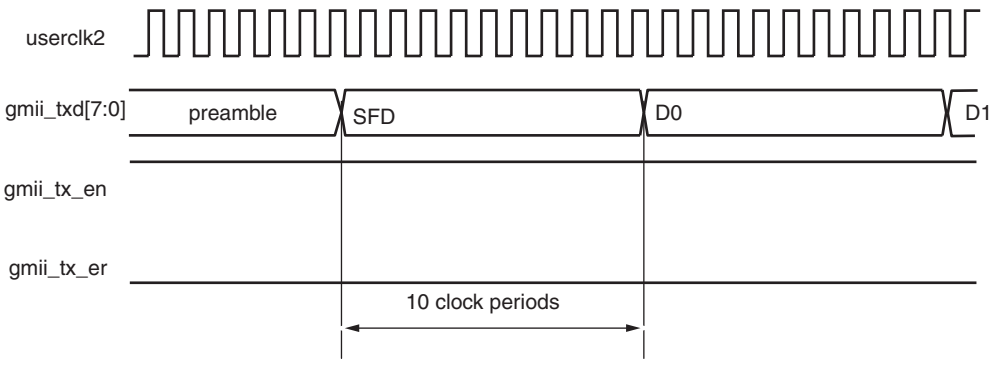


Figure 5-9: **GMII Data Transmission at 100 Mbps**

10 Megabit per Second Frame Transmission

The operation of the core remains unchanged. It is the responsibility of the client logic (for example, an Ethernet MAC), to enter data at the correct rate. When operating at a speed of 10 Mbps, every byte of the MAC frame (from destination address to the frame check sequence field, inclusive) should each be repeated for 100 clock periods to achieve the desired bit rate. It is also the responsibility of the client logic to ensure that the interframe gap period is legal for the current speed of operation.

GMII Reception

1 Gigabit per Second Frame Reception

The timing of normal inbound frame transfer is illustrated in Figure 5-10. At 1 Gbps speed, the operation of the receiver GMII signals remains identical to that described in "Designing with the Client-side GMII for the 1000BASE-X Standard" in Chapter 5.

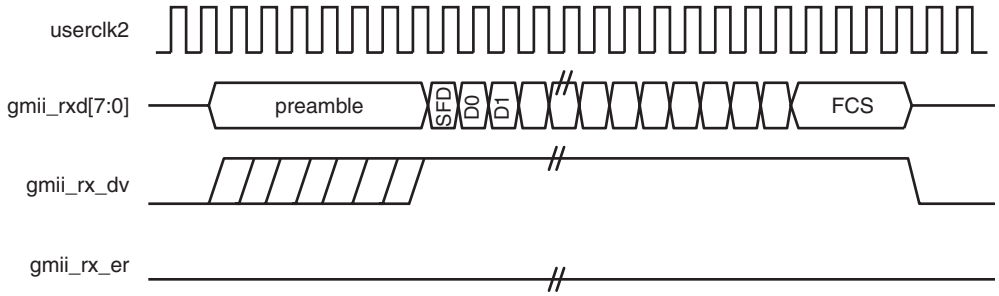


Figure 5-10: **GMII Frame Reception at 1 Gbps**

100 Megabit per Second Frame Reception

The operation of the core remains unchanged. When operating at a speed of 100 Mbps, every byte of the MAC frame (from destination address to the frame check sequence field, inclusive) is repeated for 10 clock periods to achieve the desired bit rate. See Figure 5-11. It is the responsibility of the client logic, for example an Ethernet MAC, to sample this data correctly.

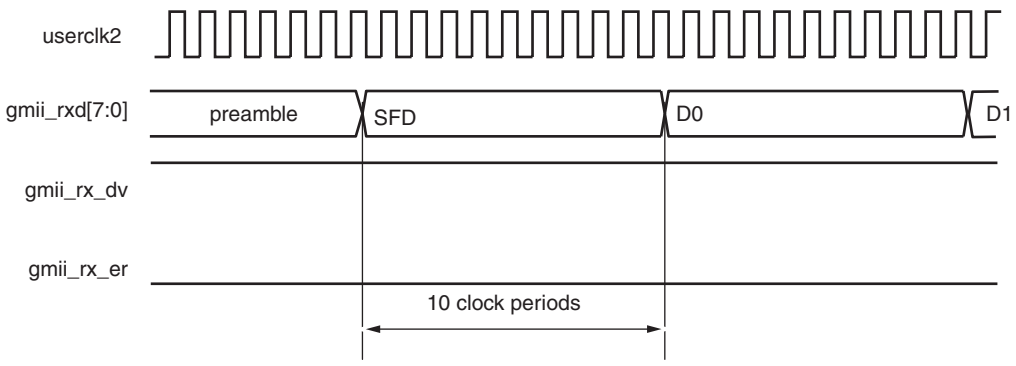


Figure 5-11: **GMII Data Reception at 100 Mbps**

10 Megabit per Second Frame Reception

The operation of the core remains unchanged. When operating at a speed of 10 Mbps, every byte of the MAC frame (from destination address to the frame check sequence field, inclusive) is repeated for 100 clock periods to achieve the desired bit rate. It is the responsibility of the client logic (for example, an Ethernet MAC) to sample this data correctly.

Using the GMII as an Internal Connection

The client-side GMII of the core may be used to connect to an internally integrated Media Access Controller. For details, see "Integrating for 1 Gbps Only Speed Capability," page 212 and "Integration for Tri-speed Capability," page 221.

Implementing External GMII

Virtex®-6 devices support GMII at 2.5V only. Please see the *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics* for more information. Virtex-5, Virtex-4, Spartan®-6 and Spartan-3 devices support GMII at 3.3V or lower.

GMII Transmitter Logic

When implementing an external GMII, the GMII transmitter signals will be synchronous to their own clock domain. The core must be used with a Transmitter Elastic Buffer to transfer these GMII transmitter signals onto the cores internal 125 MHz reference clock (`gtx_clk` when using the TBI; `userclk2` when using the device-specific transceiver). A Transmitter Elastic Buffer is provided for the 1000BASE-X standard by the example design provided with the core. See the *Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started Guide* for more information.

Spartan-3, Spartan-3E, Spartan-3A/3A DSP and Virtex-4 Devices

A DCM must be used on the `gmii_tx_clk` clock path, as illustrated in Figure 5-12. This is performed by the top-level example design delivered with the core (all signal names and logic match Figure 5-12). This DCM circuitry may optionally be used in other families.

Phase-shifting should then be applied to the DCM to fine-tune the setup and hold times at the GMII IOB input flip-flops. The fixed phase shift is applied to the DCM with the example UCF for the example design. See "Constraints When Implementing an External GMII" in Chapter 12.

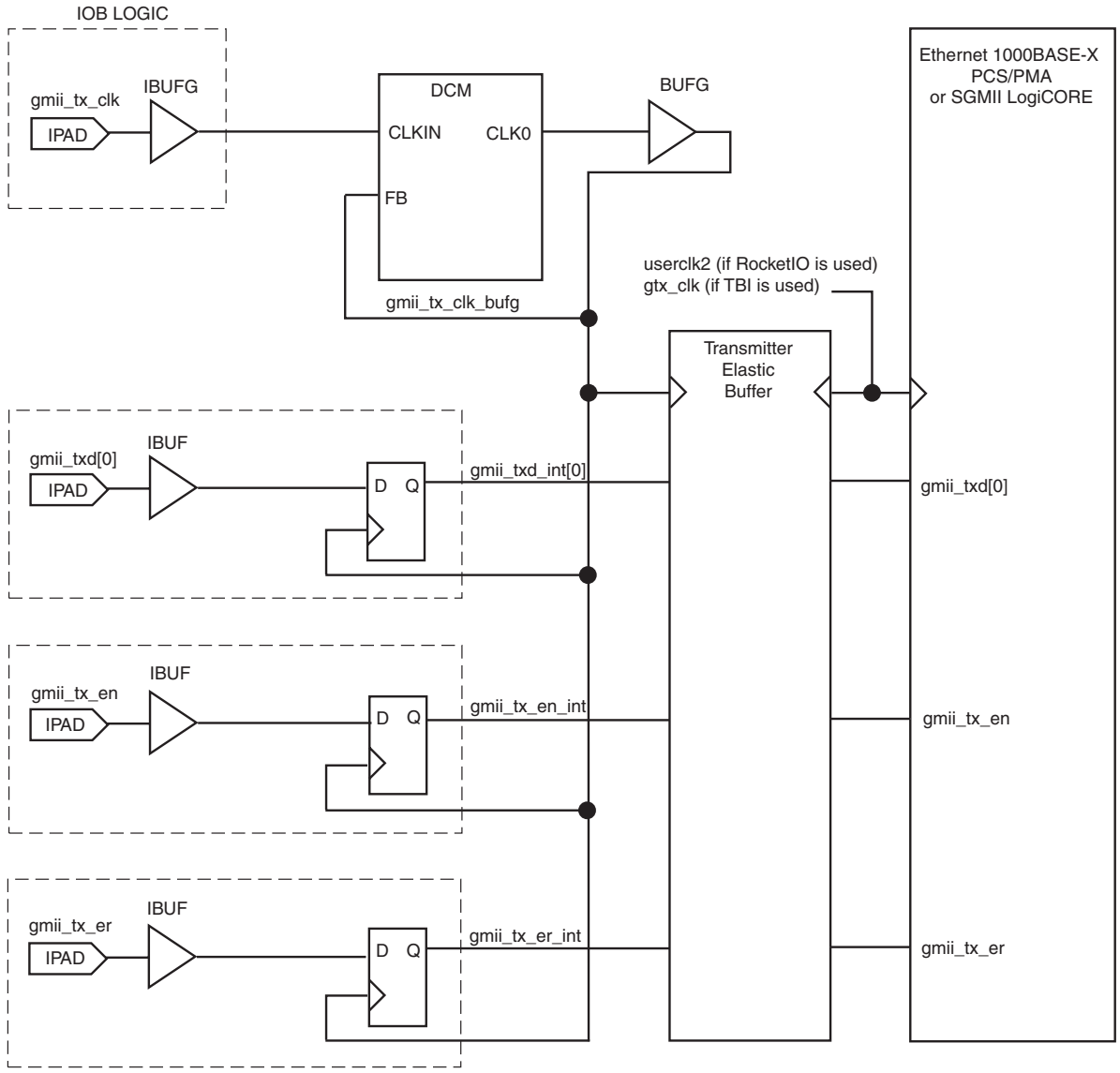


Figure 5-12: **External GMII Transmitter Logic for Spartan-3, Spartan-3E, Spartan-3A/3A DSP and Virtex-4 Devices**

Virtex-5 Devices

Three possible solutions are:

1. For Virtex-5 devices, a DCM may be used on the `gmii_tx_clk` clock path, using global clock routing, and illustrated in Figure 5-12 for the Spartan-3 and Virtex-4 family.
2. Input Delay Elements may be used on the GMII data and clock path, using global clock routing (not illustrated). This implementation was provided as the default example design for Virtex-5 devices in versions of this core prior to version 10.1.
3. Using a combination of IODELAY elements on the data, and using BUFIO and BUFR regional clock routing for the `gmii_tx_clk` input clock, as illustrated in Figure 5-13.

   The design for case 3 provides a simpler solution than the DCM logic of case 1 and provides better input setup and hold time margins than case 2. It has therefore been chosen as the default example design from version 10.1 of the core onwards.

   In this implementation, a BUFIO is used to provide the lowest form of clock routing delay from input clock to input GMII Tx signal sampling at the device IOBs. Please note, however, that this creates placement constraints: a BUFIO capable clock input pin must be selected, and all other input GMII Tx signals must be placed in the respective BUFIO region. The *Virtex-5 FPGA User Guide* should be consulted.

   The clock is then placed onto regional clock routing using the BUFR component and the input GMII Tx data immediately resampled as illustrated.

   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 "Constraints When Implementing an External GMII" in Chapter 12 for more information.

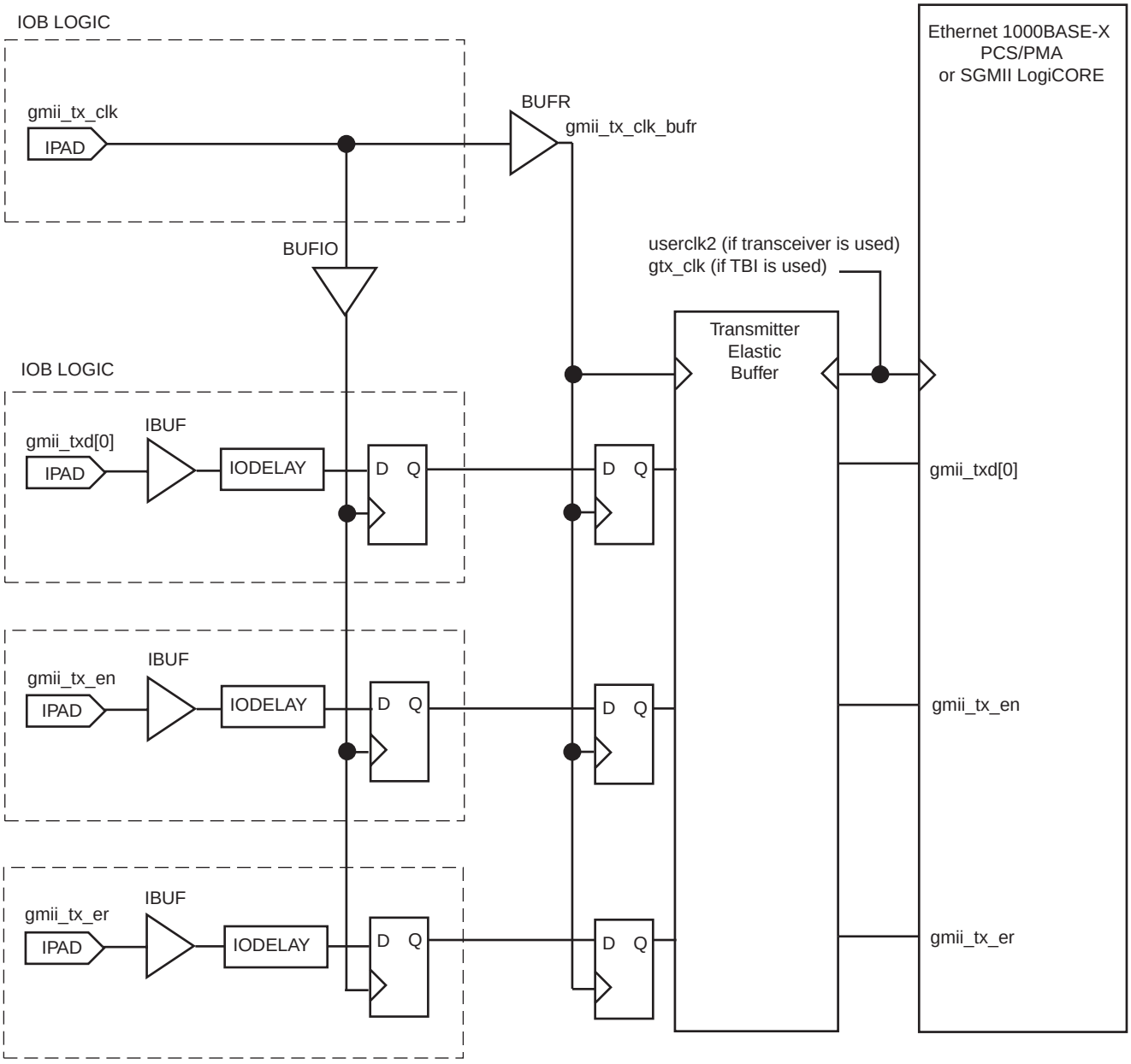


Figure 5-13: **External GMII Transmitter Logic for Virtex-5 and Virtex-6 Devices**

Virtex-6 Devices

Two possible solutions are:

1. For Virtex-6 devices, a MMCM may be used on the `gmii_tx_clk` clock path, using global clock routing. This is as illustrated in Figure 5-12 for the Spartan-3 and Virtex-4 family; simply replace the DCM for a MMCM.
2. Using a combination of IODELAY elements on the data, and using BUFIO and BUFR regional clock routing for the `gmii_tx_clk` input clock, as illustrated in Figure 5-13.

   The design for case 2 provides a simpler solution than that of case 1. It has therefore been chosen as the default example design for 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 Tx signal sampling at the device IOBs. Please note, however, that this creates placement constraints: a BUFIO capable clock input pin must be selected, and all other input GMII Tx signals must be placed in the respective BUFIO region. The *Virtex-6 FPGA User Guide* should be consulted.

   The clock is then placed onto regional clock routing using the BUFR component and the input GMII Tx data immediately resampled as illustrated.

   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 "Constraints When Implementing an External GMII" in Chapter 12 for more information.

Spartan-6 Devices

Three possible solutions are:

1. For Spartan-6 devices, a MMCM may be used on the `gmii_tx_clk` clock path, using global clock routing, and illustrated in Figure 5-12 for the Spartan-3 and Virtex-4 family.
2. Using a combination of IODELAY elements on the data, and using BUFIO2 and BUFG global clock routing for the `gmii_tx_clk` input clock, as illustrated in Figure 5-14.

   The design for case 2 provides a simpler solution than that of case 1. It has therefore been chosen as the default example design for 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 Tx signal sampling at the device IOBs. Please note, however, that this creates placement constraints: a BUFIO capable clock input pin must be selected, and all other input GMII Tx signals must be placed in the respective BUFIO2 region. The *Spartan-6 FPGA User Guide* should be consulted.

   The clock is then placed onto global clock routing using the BUFG component and the input GMII Tx data immediately resampled as illustrated.

   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 "Constraints When Implementing an External GMII" in Chapter 12 for more information.

   .

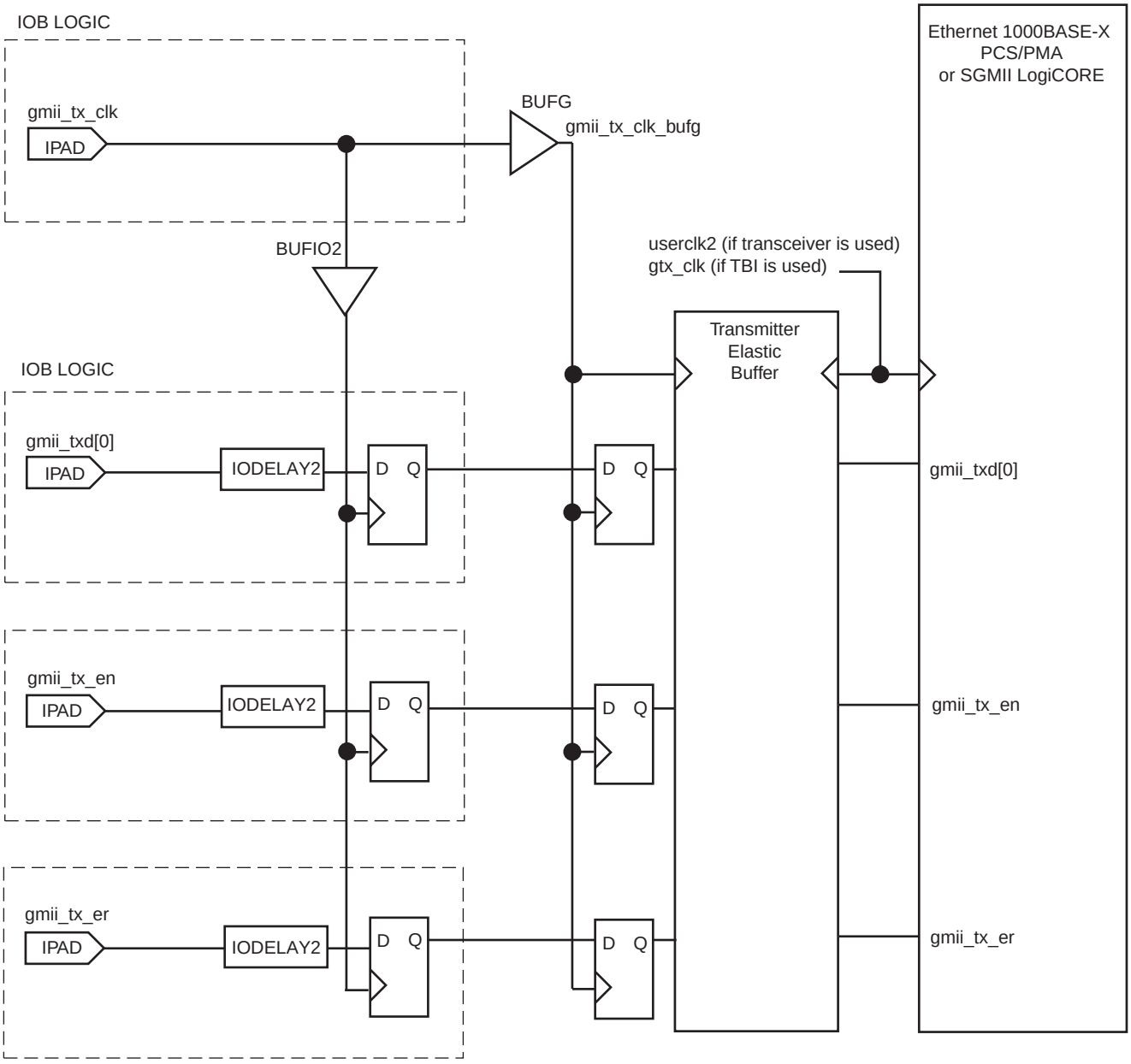


Figure 5-14: **External GMII Transmitter Logic for Spartan-6 Devices**

GMII Receiver Logic

Figure 5-15 illustrates an external GMII receiver created in a Virtex-5 family device. The signal names and logic shown in the figure exactly match those delivered with the example design when the GMII is selected. If other families are selected, equivalent primitives and logic specific to that family is automatically used in the example design.

Figure 5-15 also shows that the output receiver signals are registered in device IOBs before driving them to the device pads. The logic required to forward the receiver GMII clock is also shown. This 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_rx_clk`, is inverted so that the rising edge of `gmii_rx_clk` occurs in the center of the data valid window, which maximizes setup and hold times across the interface. All receiver logic is synchronous to a single clock domain.

The clock name varies depending on the CORE Generator™ software options. When used with the device-specific transceiver, the clock name is `userclk2`; when used with the TBI, the clock name is `gtx_clk`. For more information on clocking, see Chapters 6, 7 and 8.

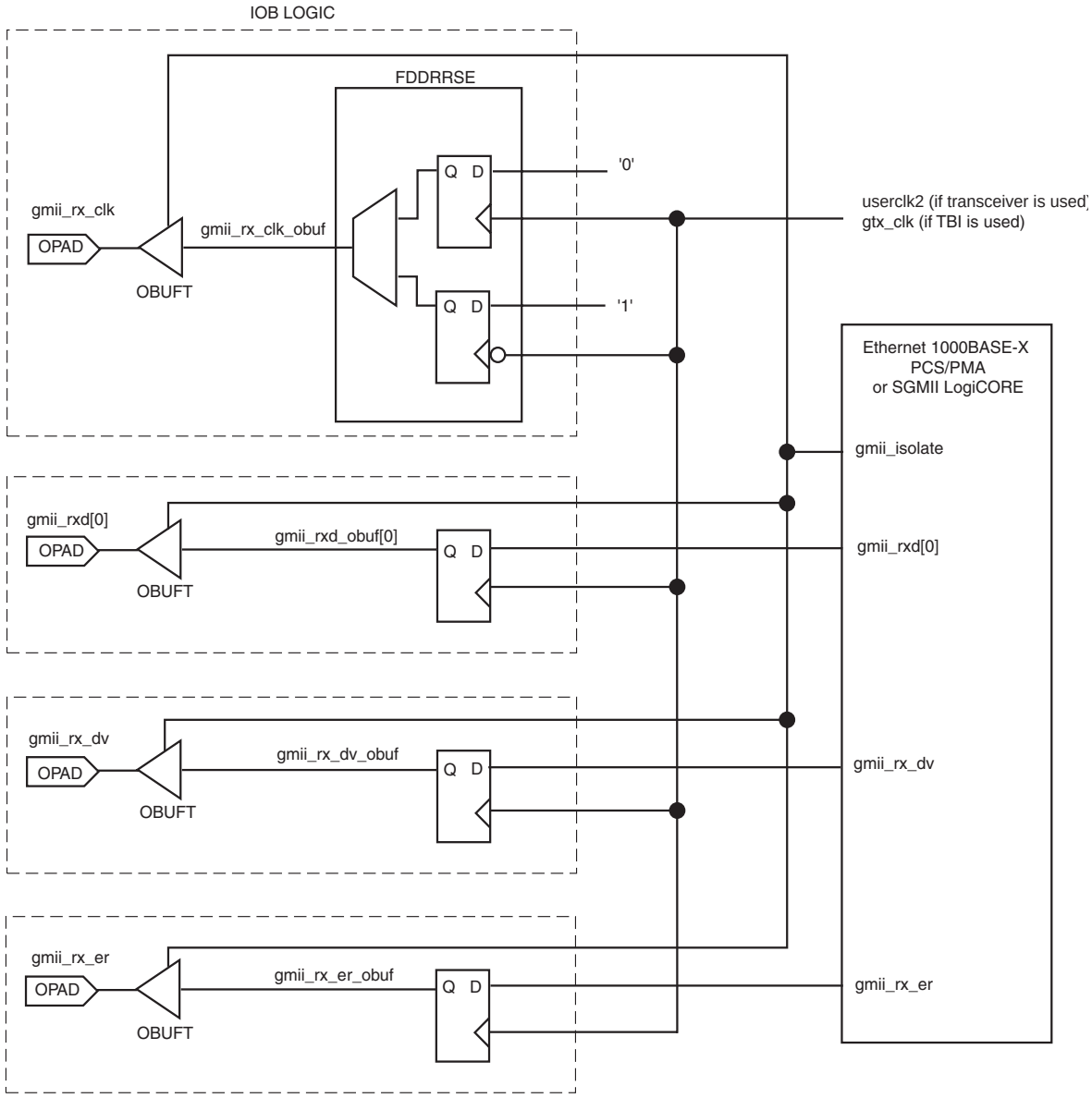


Figure 5-15: **External GMII Receiver Logic**

XILINX®

Chapter 6

The Ten-Bit Interface

This chapter provides general guidelines for creating 1000BASE-X, SGMII or Dynamic Standards Switching designs using the Ten-Bit Interface (TBI). An explanation of the TBI logic in all supported device families is provided, as well as examples in which multiple instantiations of the core are required. Whenever possible, clock sharing should occur to save device resources.

Virtex®-6 devices support TBI at 2.5V only. Please see the *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics* for more information. Virtex-5, Virtex-4, Spartan®-6 and Spartan-3 devices support TBI at 3.3V or lower.

Ten-Bit-Interface Logic

The example design delivered with the core is split between two hierarchical layers, as illustrated in Figure 4-2. The block level is designed so that it can be instantiated directly into customer designs and provides the following functionality:

- Instantiates the core from HDL
- Connects the physical-side interface of the core to device IOBs, creating an external TBI

The TBI logic implemented in the block level is illustrated in all the figures in this chapter.

Transmitter Logic

Figure 6-1 illustrates the use of the physical transmitter interface of the core to create an external TBI in a Virtex-5 family device. The signal names and logic shown exactly match those delivered with the example design when TBI is chosen. If other families are chosen, equivalent primitives and logic specific to that family will automatically be used in the example design.

Figure 6-1 shows that the output transmitter data path signals are registered in device IOBs before driving them to the device pads. The logic required to forward the transmitter clock is also shown. The 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, `pma_tx_clk`, is inverted with respect to `gtx_clk` so that the rising edge of `pma_tx_clk` occurs in the center of the data valid window to maximize setup and hold times across the interface.

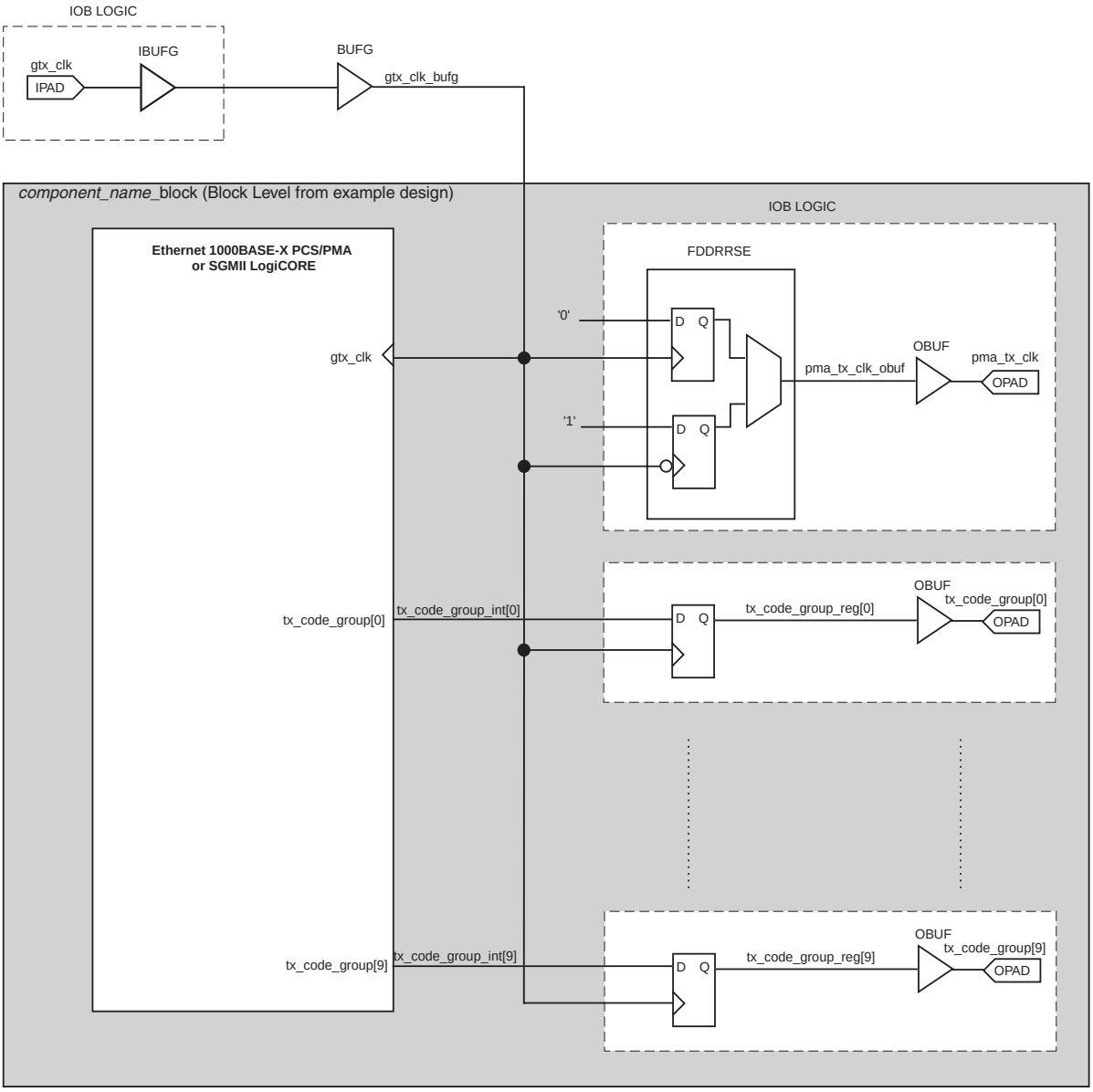


Figure 6-1: **Ten-Bit Interface Transmitter Logic**

Receiver Logic

Introduction

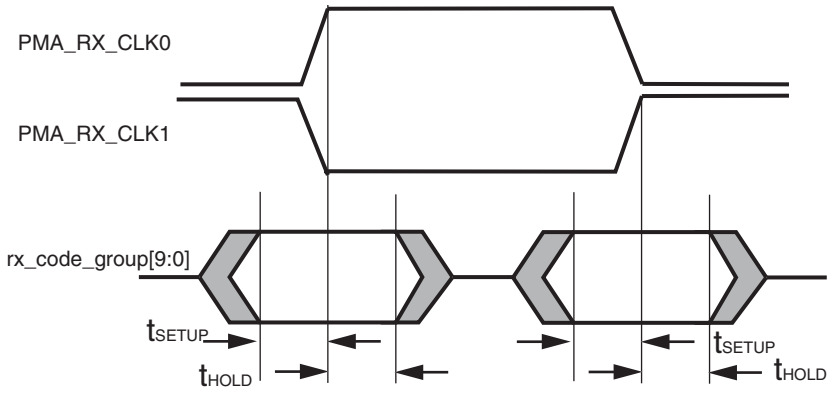


Figure 6-2: **Input TBI timing**

Figure 6-2 illustrates the input timing for the TBI interface as defined in IEEE802.3 clause 36 (see also "TBI Input Setup/Hold Timing" in Chapter 12 for further information).

The important point to note here is that the input TBI data bus, `rx_code_group[9:0]`, is synchronous to two clock sources: `pma_rx_clk0` and `pma_rx_clk1`. As defined by the standard, the TBI data should be sampled alternatively on the rising edge of `pma_rx_clk0`, then `pma_rx_clk1`. Minimum setup and hold constraints are specified and apply to both clock sources.

In the IEEE802.3 specification, there is no exact requirement that `pma_rx_clk0` and `pma_rx_clk1` be exactly 180 degrees out of phase with each other, so the safest approach is to use both `pma_rx_clk0` and `pma_rx_clk1` clocks as the specification intends. This is at the expense of clocking resources.

However, the data sheet for a particular external SERDES device which connects to the TBI may well specify that this is the case: that `pma_rx_clk0` and `pma_rx_clk1` are exactly 180 degrees out of phase. If this is the case then the TBI receiver clock logic may be simplified by ignoring the `pma_rx_clk1` clock altogether, and simply using both the rising and falling edges of `pma_rx_clk0`.

For this reason, the following sections describe two different alternatives methods for implementing the TBI receiver clock logic: one which uses both `pma_rx_clk0` and `pma_rx_clk1` clock, and a second which only uses `pma_rx_clk0` (but both rising and falling edges). Please select the method carefully by referring to the data sheet of the external SERDES.

The example designs provided with the core will only provide one of these methods (which vary on a family by family basis). However, the example design HDL can easily be edited to convert to the alternative method.

Spartan-3, Spartan-3E and Spartan-3A Devices

Method 1: Using Both pma_rx_clk0 and pma_rx_clk1 (Provided by the Example Design)

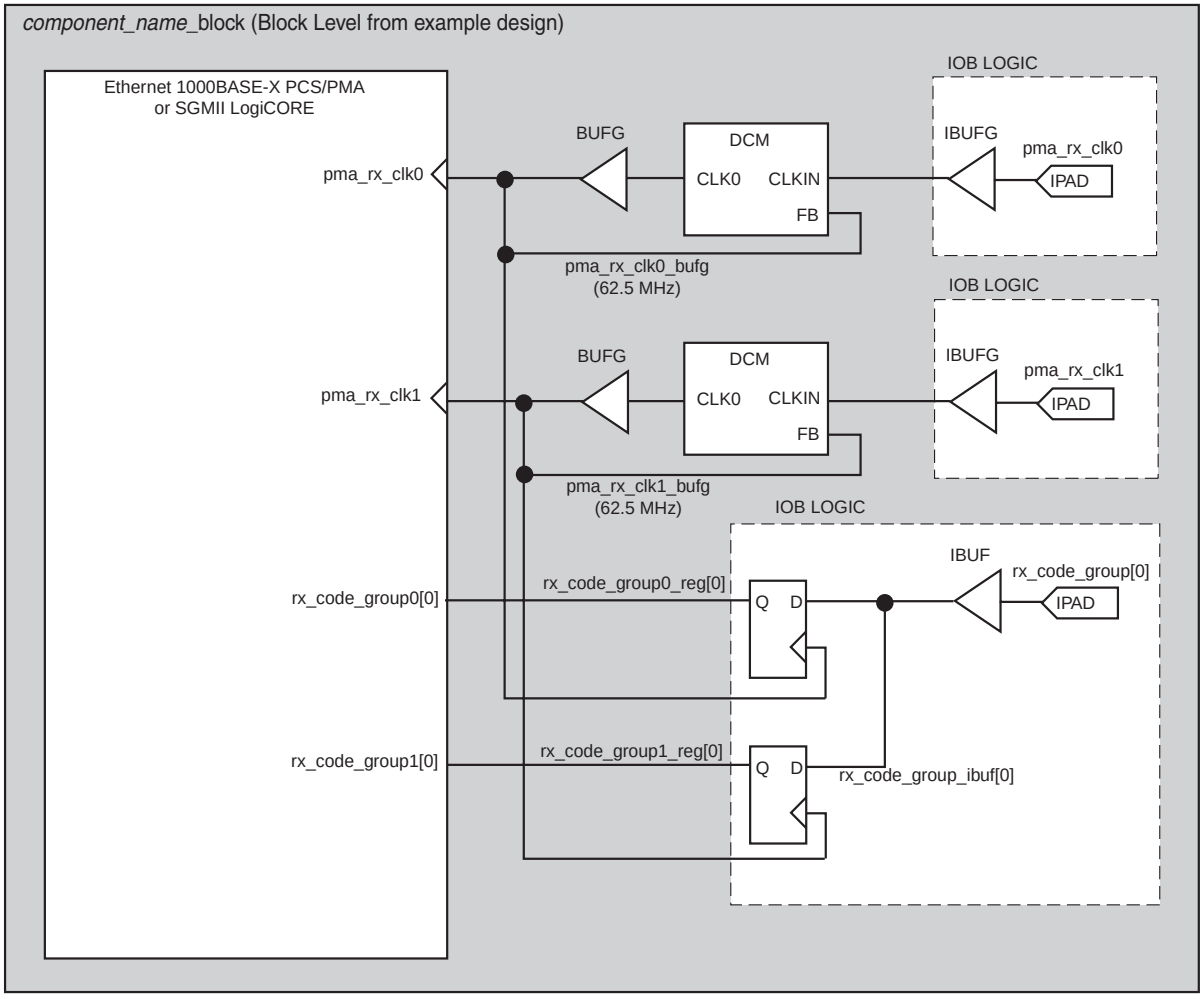


Figure 6-3: **TBI Receiver Logic for Spartan-3, Spartan-3E, and Spartan-3A Devices (Example Design)**

This is the implementation provided by the example design for the Spartan-3 families. This uses the `pma_rx_clk0` and `pma_rx_clk1` clocks as intended by the TBI specification. Please contrast this with Method 2 which can save on clock resources *if* the external SERDES devices guarantees that it provides `pma_rx_clk0` and `pma_rx_clk1` exactly 180 degrees out of phase with each other.

In this implementation, a DCM is used on both the `pma_rx_clk0` and `pma_rx_clk1` clock paths (see Figure 6-3). Phase shifting should then be applied to the DCM to fine-tune the setup and hold times at the TBI IOB input flip-flops. Fixed phase shift is applied to the DCM using constraints in the example UCF for the example design. See "Ten-Bit Interface Constraints" in Chapter 12 for more information.

Method 2: An Alternative Using Only pma_rx_clk0

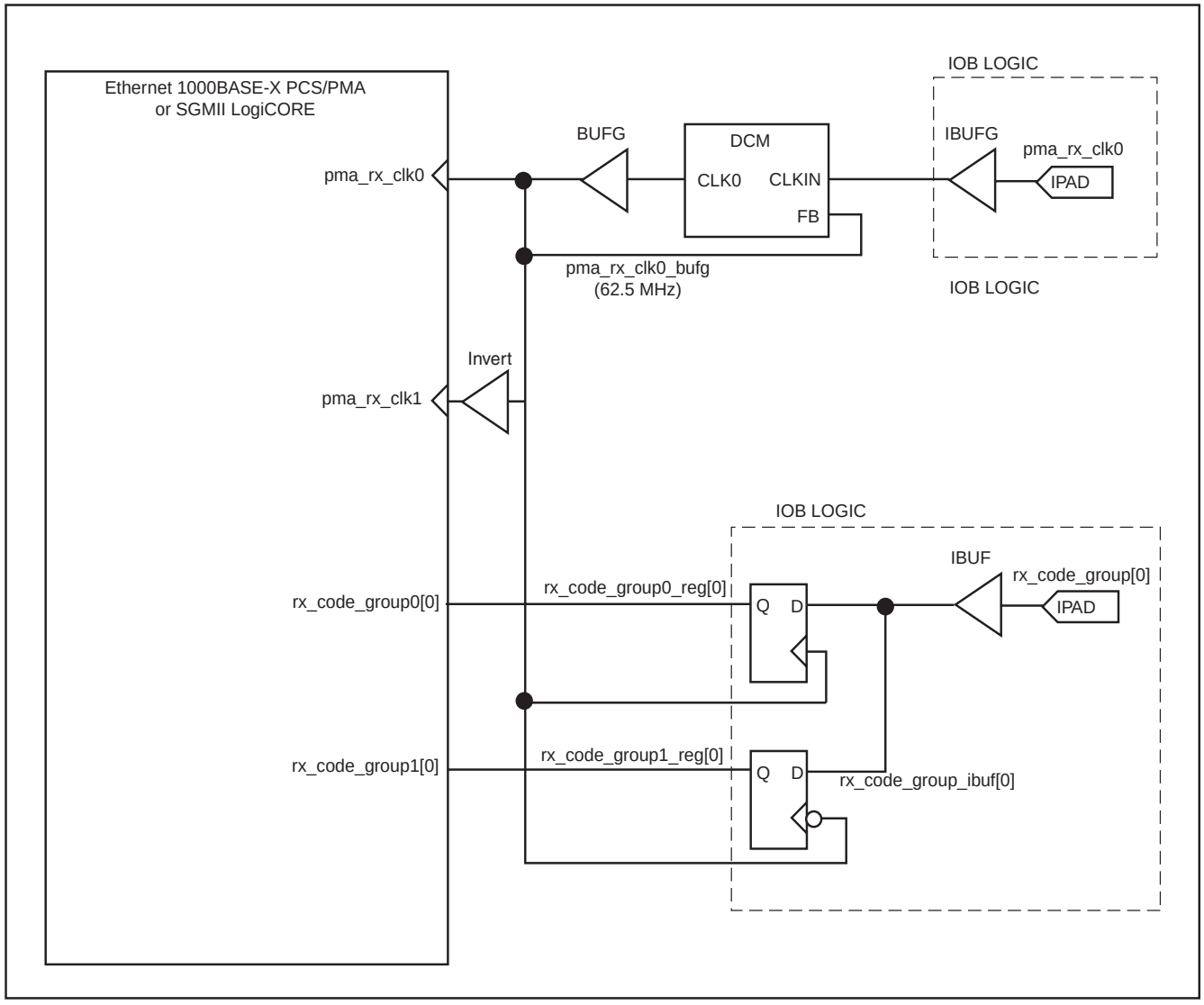


Figure 6-4: **TBI Receiver Logic for Spartan-3, Spartan-3E, and Spartan-3A Devices**

In this implementation, the falling edge of `pma_rx_clk0` is used instead of `pma_rx_clk1` (see Figure 6-4).

The DCM is used on the `pma_rx_clk0` clock path. Phase shifting should then be applied to the DCM to fine-tune the setup and hold times at the `rx_code_group[9:0]` IOB input flip-flops.

The clock derived from the DCM should be inverted, as illustrated, before routing it to the `pma_rx_clk1` input of the core. This will not create a clock on local routing. Instead the tools will use local clock inversion directly at the clock input of the flip-flops that this clock is routed to.

> ***Caution!*** This logic relies on `pma_rx_clk0` and `pma_rx_clk1` being exactly 180 degrees out of phase with each other since the falling edge of `pma_rx_clk0` is used in place of `pma_rx_clk1`. See the data sheet for the attached SERDES to verify that this is the case.

Virtex-4 Devices

Method 1: Using Only pma_rx_clk0 (Provided by the Example Design)

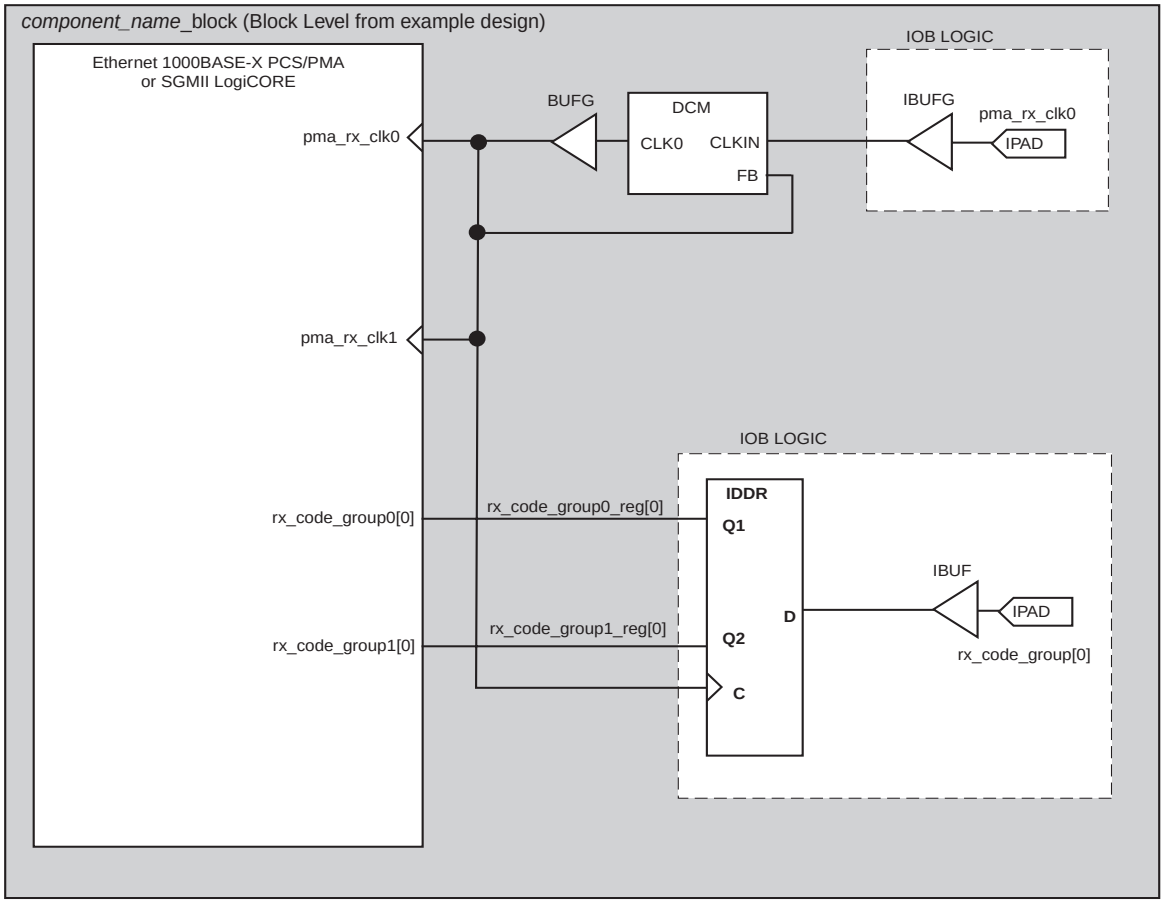


Figure 6-5: **Ten-Bit Interface Receiver Logic - Virtex-4 Device (Example Design)**

The Virtex-4 FPGA logic used by the example design delivered with the core is illustrated in Figure 6-5. This shows a Virtex-4 device `IDDR` primitive used with the `DDR_CLK_EDGE` attribute set to `SAME_EDGE` (see the *Virtex-4 FPGA User Guide*). This uses local inversion of `pma_rx_clk0` within the IOB logic to receive the `rx_code_group[9:0]` data bus on both the rising and falling edges of `pma_rx_clk0`. The `SAME_EDGE` attribute causes the `IDDR` to output both `Q1` and `Q2` data on the rising edge of `pma_rx_clk0`.

For this reason, `pma_rx_clk0` can be routed to both `pma_rx_clk0` and `pma_rx_clk1` clock inputs of the core as illustrated.

> ***Caution!*** This logic relies on `pma_rx_clk0` and `pma_rx_clk1` being exactly 180 degrees out of phase with each other since the falling edge of `pma_rx_clk0` is used in place of `pma_rx_clk1`. See the data sheet for the attached SERDES to verify that this is the case.

The DCM is used on the `pma_rx_clk0` clock path. Phase shifting should then be applied to the DCM to fine-tune the setup and hold times at the `rx_code_group[9:0]` IOB input flip-flops. Fixed phase shift is applied to the DCM using constraints in the example UCF for the example design. See "Ten-Bit Interface Constraints" in Chapter 12 for more information.

Method 2: An Alternative Using both pma_rx_clk0 and pma_rx_clk1

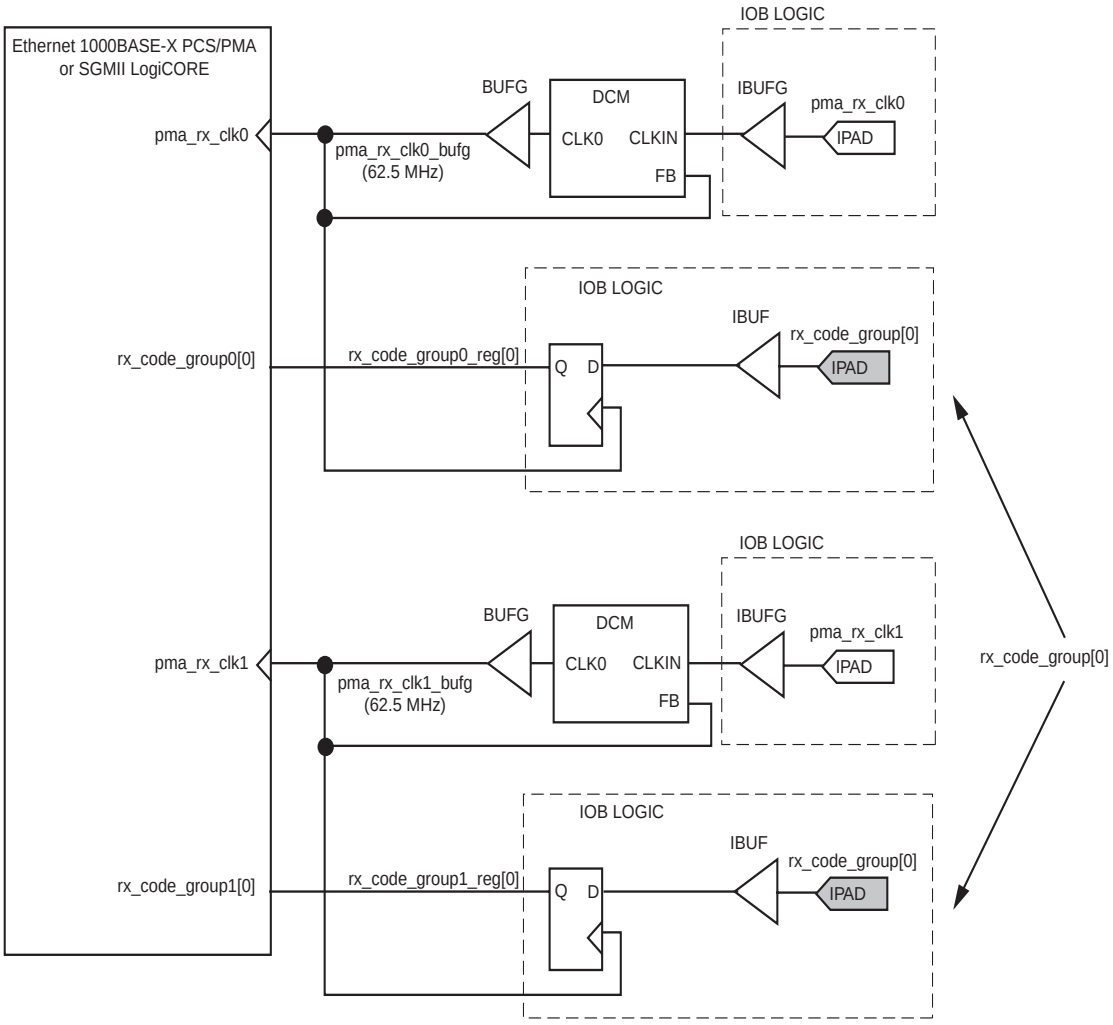


Figure 6-6: **Alternate Ten-Bit Interface Receiver Logic for Virtex-4 Devices**

This logic from method 1 relies on `pma_rx_clk0` and `pma_rx_clk1` being exactly 180 degrees out of phase with each other since the falling edge of `pma_rx_clk0` is used in place of `pma_rx_clk1`. See the data sheet for the attached SERDES to verify that this is the case. If not, then the logic of Figure 6-6 illustrates an alternative implementation where both `pma_rx_clk0` and `pma_rx_clk1` are used as intended. Each bit of `rx_code_group[9:0]` must be routed to two separate device pads.

In this implementation, a DCM is used on both the `pma_rx_clk0` and `pma_rx_clk1` clock paths (see Figure 6-6). Phase shifting should then be applied to the DCMs to fine-tune the setup and hold times at the TBI IOB input flip-flops.

Virtex-5 Devices

Method 1: Using Only pma_rx_clk0 (Provided by the Example Design)

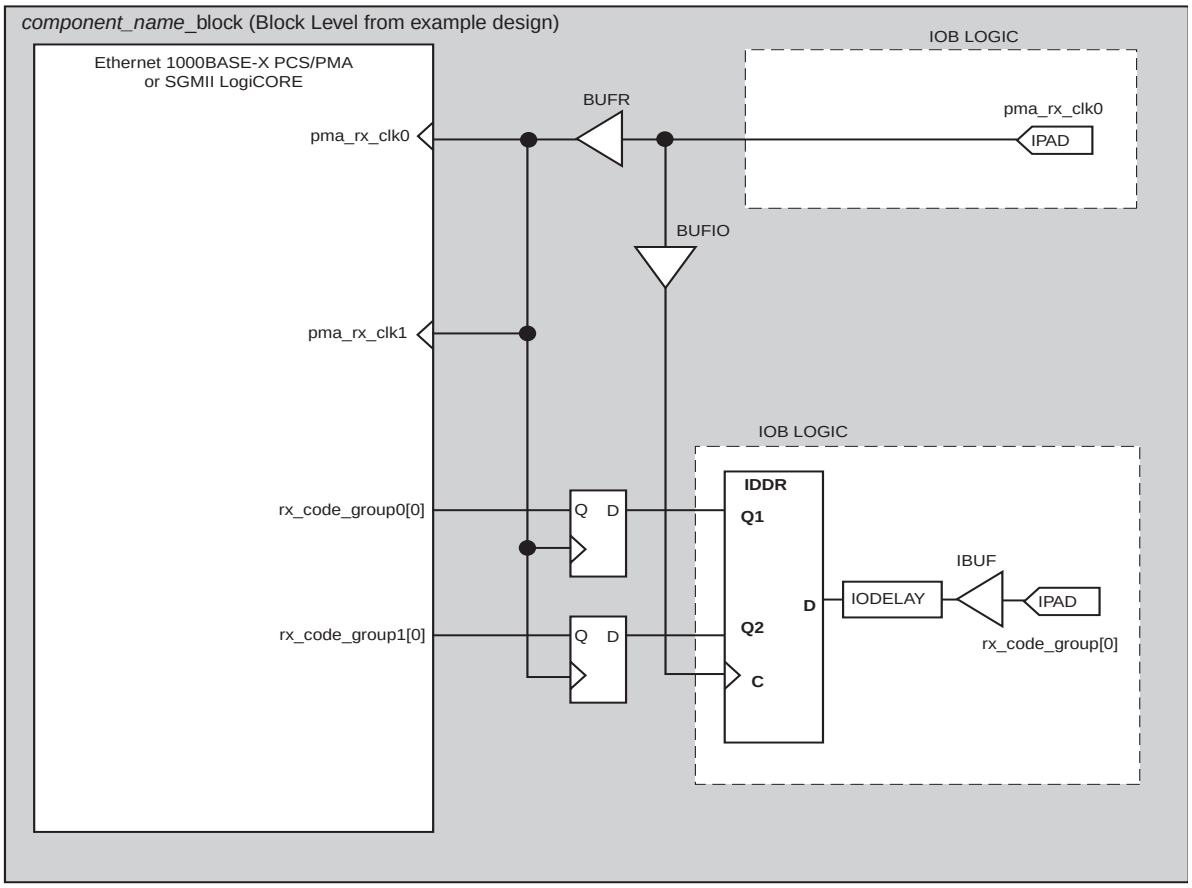


Figure 6-7: **Ten-Bit Interface Receiver Logic - Virtex-5 Device (Example Design)**

The Virtex-5 FPGA logic used by the example design delivered with the core is illustrated in Figure 6-7. This shows a Virtex-5 device `IDDR` primitive used with the `DDR_CLK_EDGE` attribute set to `SAME_EDGE` (see the *Virtex-5 FPGA User Guide*). This uses local inversion of `pma_rx_clk0` within the IOB logic to receive the `rx_code_group[9:0]` data bus on both the rising and falling edges of `pma_rx_clk0`. The `SAME_EDGE` attribute causes the `IDDR` to output both `Q1` and `Q2` data on the rising edge of `pma_rx_clk0`.

For this reason, `pma_rx_clk0` can be routed to both `pma_rx_clk0` and `pma_rx_clk1` clock inputs of the core as illustrated.

> ***Caution!*** This logic relies on `pma_rx_clk0` and `pma_rx_clk1` being exactly 180 degrees out of phase with each other because the falling edge of `pma_rx_clk0` is used in place of `pma_rx_clk1`. See the data sheet for the attached SERDES to verify that this is the case.

Setup and Hold is achieved using a combination of IODELAY elements on the data, and using BUFIO and BUFR regional clock routing for the `pma_rx_clk0` input clock, as illustrated in Figure 6-7.

This design provides a simpler solution than the DCM logic required for Virtex-4 devices (see Figure 6-5). It has therefore been chosen as the example design from version 10.1 of the core onwards. However, the Virtex-4 approach could alternatively be adopted.

In the Figure 6-7 implementation, a BUFIO is used to provide the lowest form of clock routing delay from input clock to input `rx_code_group[9:0]` signal sampling at the device IOBs. Please note, however, that this creates placement constraints: a BUFIO capable clock input pin must be selected for `pma_rx_clk0`, and all `rx_code_group[9:0]` input signals must be placed in the respective BUFIO region. The *Virtex-5 FPGA User Guide* should be consulted.

The clock is then placed onto regional clock routing using the BUFR component and the input `rx_code_group[9:0]` data immediately resampled as illustrated.

The IODELAY elements can be adjusted to fine-tune the setup and hold times at the TBI IOB input flip-flops. The delay is applied to the IODELAY element using constraints in the UCF; these can be edited if desired. See "Ten-Bit Interface Constraints" in Chapter 12 for more information

Method 2: An Alternative Using Both pma_rx_clk0 and pma_rx_clk1

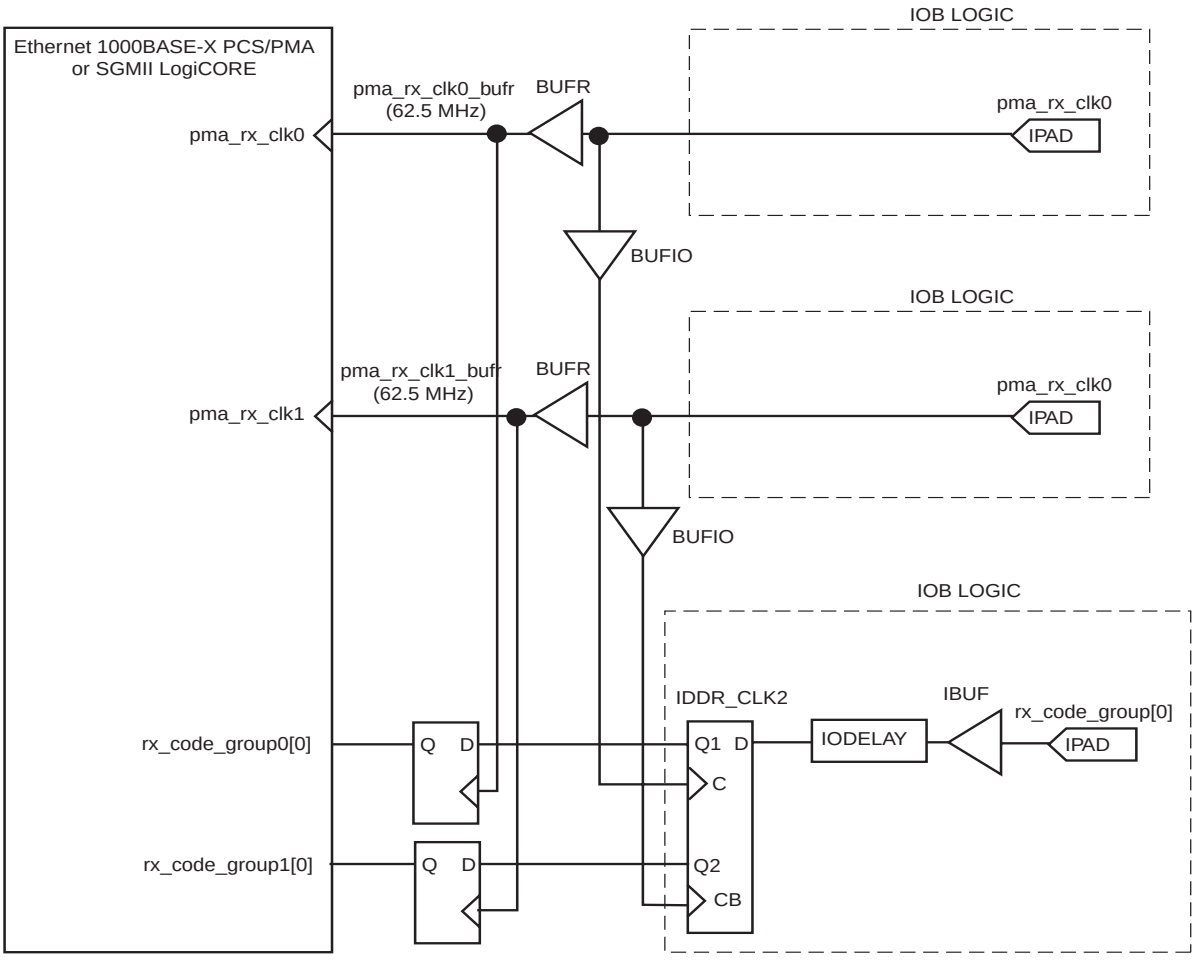


Figure 6-8: **Alternate Ten-Bit Interface Receiver Logic - Virtex-5 Devices**

The logic from method 1 relies on `pma_rx_clk0` and `pma_rx_clk1` being exactly 180 degrees out of phase with each other because the falling edge of `pma_rx_clk0` is used in place of `pma_rx_clk1`. See the data sheet for the attached SERDES to verify that this is the case. If not, the logic of Figure 6-8 illustrates an alternate implementation where both `pma_rx_clk0` and `pma_rx_clk1` are used as intended.

In this method, the logic used on pma_rx_clk0 in Figure 6-7 is duplicated for pma_rx_clk1. A IDDR_CLK2 primitive replaces the IDDR primitive: this contains two clock inputs as illustrated.

Virtex-6 Devices

Method 1: Using Only pma_rx_clk0 (Provided by the Example Design)

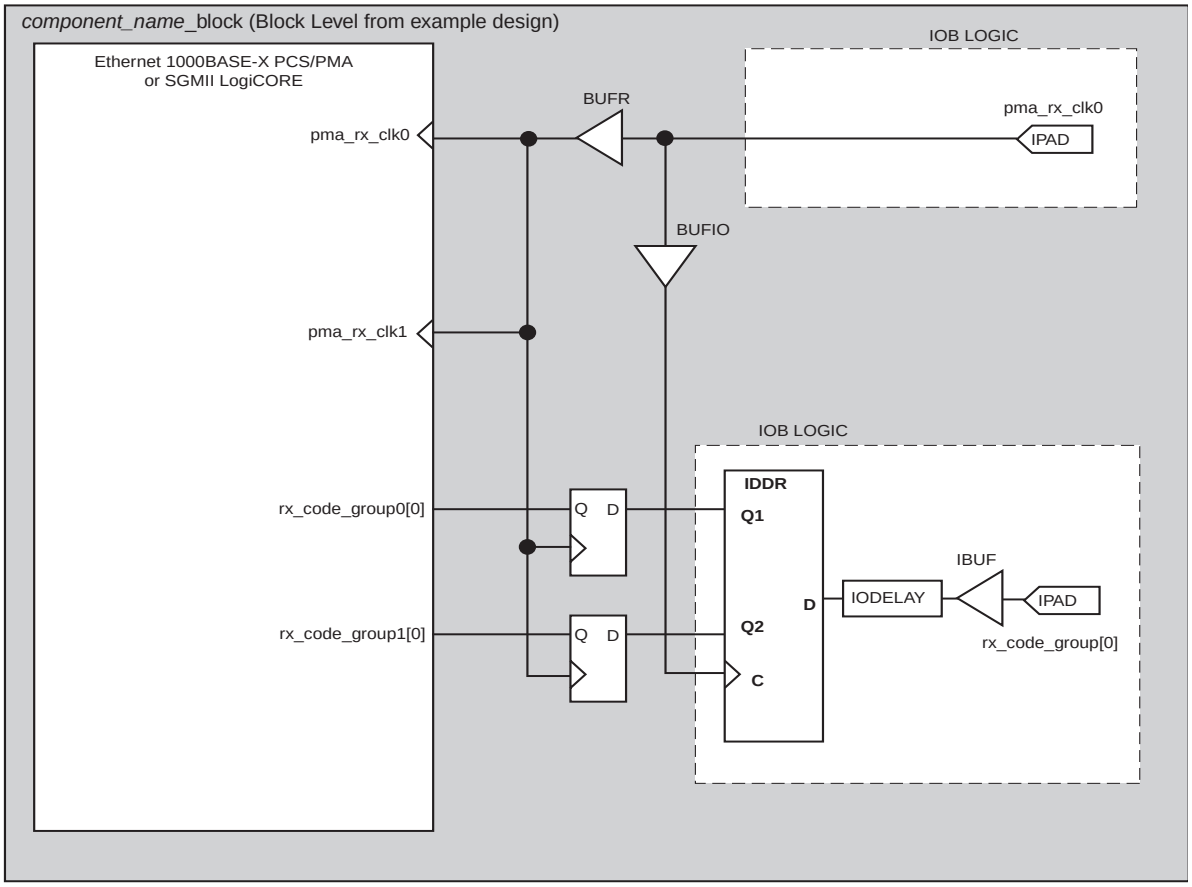


Figure 6-9: **Ten-Bit Interface Receiver Logic - Virtex-6 Device (Example Design)**

The Virtex-6 FPGA logic used by the example design delivered with the core is illustrated in Figure 6-7. This shows a Virtex-6 device IDDR primitive used with the DDR_CLK_EDGE attribute set to SAME_EDGE (see the *Virtex-6 FPGA User Guide*). This uses local inversion of pma_rx_clk0 within the IOB logic to receive the rx_code_group[9:0] data bus on both the rising and falling edges of pma_rx_clk0. The SAME_EDGE attribute causes the IDDR to output both Q1 and Q2 data on the rising edge of pma_rx_clk0.

For this reason, pma_rx_clk0 can be routed to both pma_rx_clk0 and pma_rx_clk1 clock inputs of the core as illustrated.

> ***Caution!*** This logic relies on pma_rx_clk0 and pma_rx_clk1 being exactly 180 degrees out of phase with each other because the falling edge of pma_rx_clk0 is used in place of pma_rx_clk1. See the data sheet for the attached SERDES to verify that this is the case.

Setup and Hold is achieved using a combination of IODELAY elements on the data, and using BUFIO and BUFR regional clock routing for the pma_rx_clk0 input clock, as illustrated in Figure 6-9.

This design provides a simpler solution than the DCM logic required for Virtex-4 devices. It has therefore been chosen as the example design for the Virtex-6 family. However, the Virtex-4 approach could alternatively be adopted: simply replace the DCM with a MMCM module (see Figure 6-5).

In the Figure 6-9 implementation, a BUFIO is used to provide the lowest form of clock routing delay from input clock to input `rx_code_group[9:0]` signal sampling at the device IOBs. Please note, however, that this creates placement constraints: a BUFIO capable clock input pin must be selected for `pma_rx_clk0`, and all `rx_code_group[9:0]` input signals must be placed in the respective BUFIO region. The *Virtex-6 FPGA User Guide* should be consulted.

The clock is then placed onto regional clock routing using the BUFR component and the input `rx_code_group[9:0]` data immediately resampled as illustrated.

The IODELAY elements can be adjusted to fine-tune the setup and hold times at the TBI IOB input flip-flops. The delay is applied to the IODELAY element using constraints in the UCF; these can be edited if desired. See "Ten-Bit Interface Constraints" in Chapter 12 for more information

Method 2: An Alternative Using Both pma_rx_clk0 and pma_rx_clk1

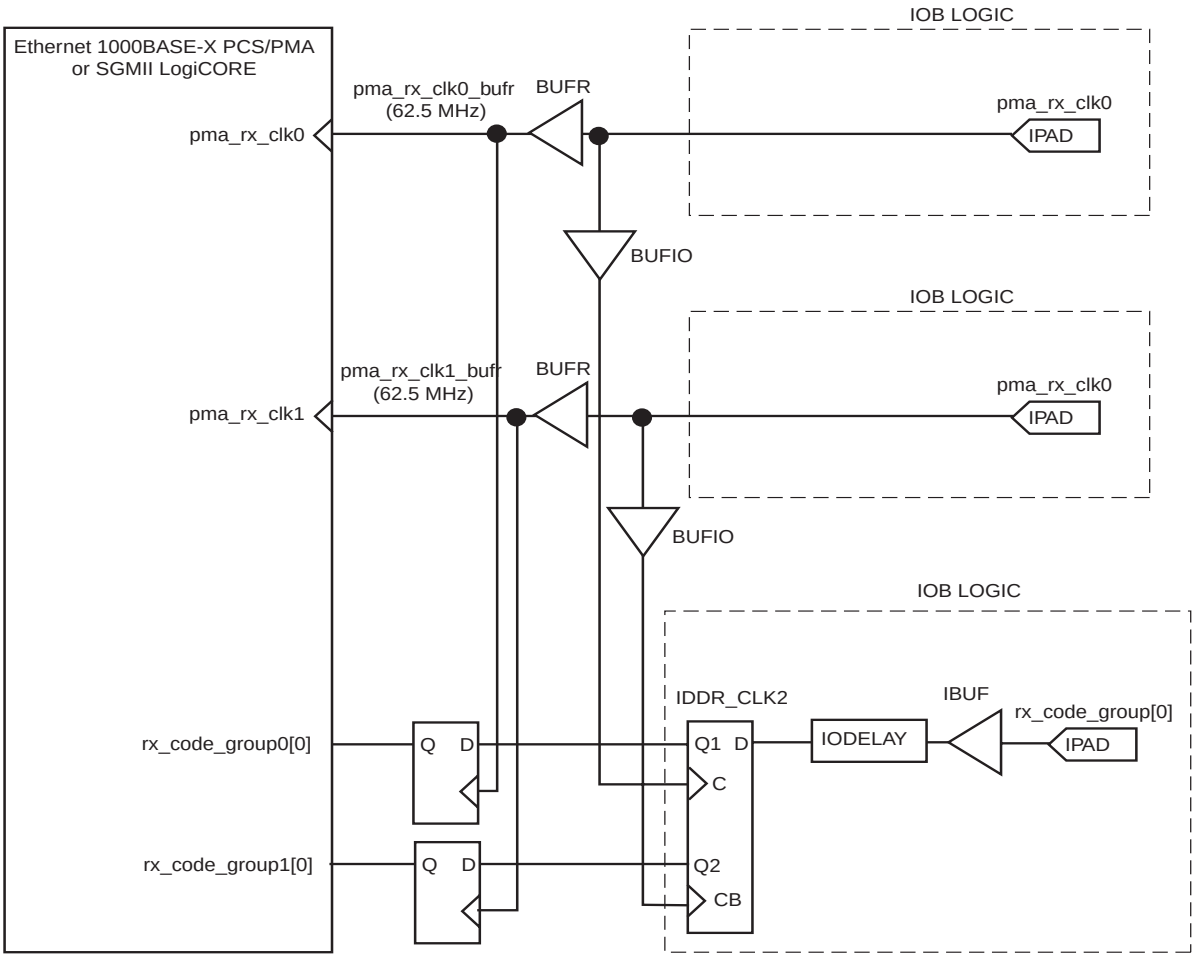


Figure 6-10: **Alternate Ten-Bit Interface Receiver Logic - Virtex-6 Devices**

This logic from method 1 relies on `pma_rx_clk0` and `pma_rx_clk1` being exactly 180 degrees out of phase with each other because the falling edge of `pma_rx_clk0` is used in place of `pma_rx_clk1`. See the data sheet for the attached SERDES to verify that this is the case. If not, the logic of Figure 6-10 illustrates an alternate implementation where both `pma_rx_clk0` and `pma_rx_clk1` are used as intended. Each bit of `rx_code_group[9:0]` must be routed to two separate device pads.

In this method, the logic used on `pma_rx_clk0` in Figure 6-9 is duplicated for `pma_rx_clk1`. A `IDDR_CLK2` primitive replaces the `IDDR` primitive: this contains two clock inputs as illustrated.

Spartan-6 Devices

Method 1: Using Only pma_rx_clk0 (Provided by the Example Design)

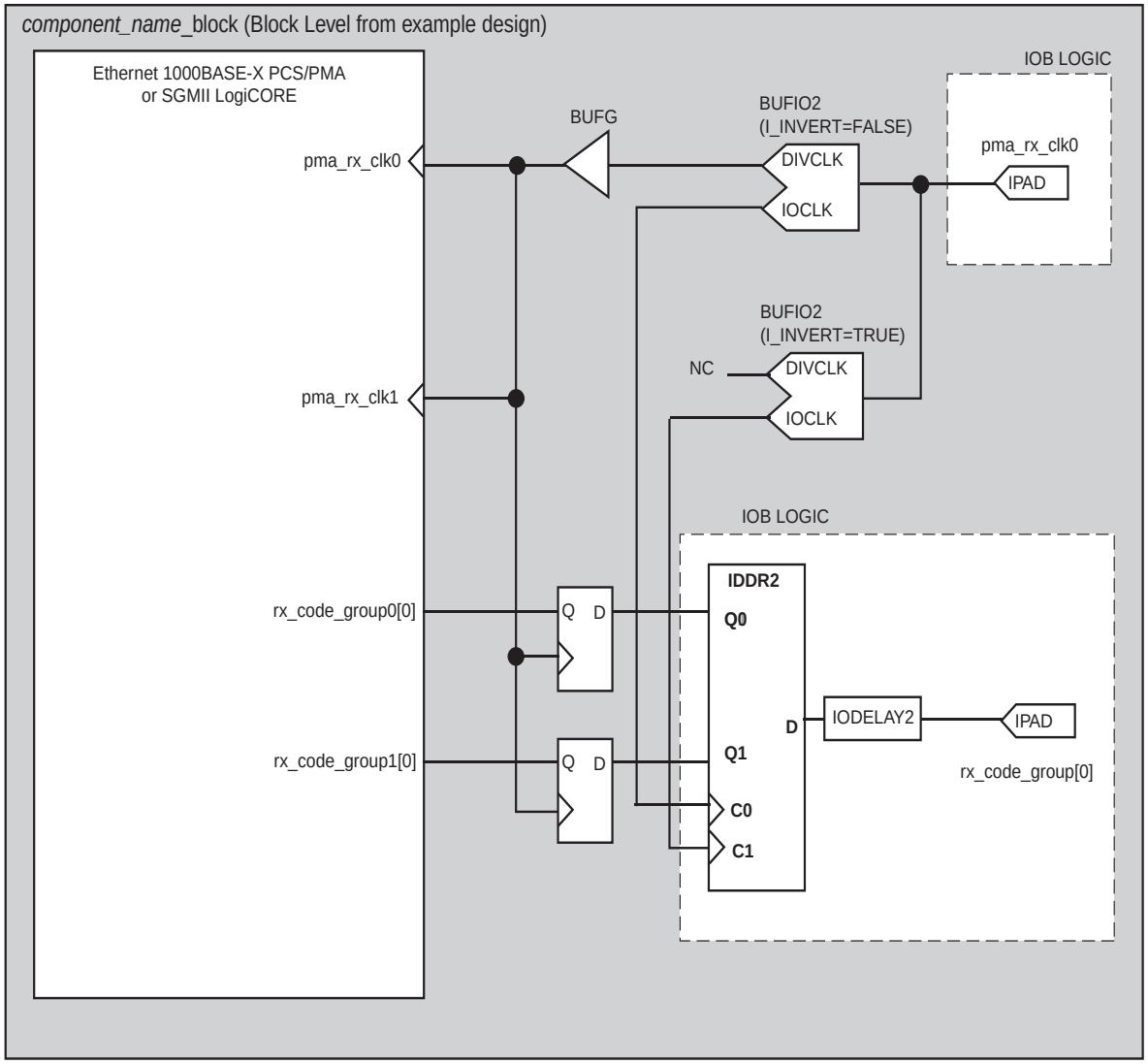


Figure 6-11: **Ten-Bit Interface Receiver Logic - Spartan-6 Device (Example Design)**

The Spartan-6 FPGA logic used by the example design delivered with the core is illustrated in Figure 6-11. This figure shows a Spartan-6 device IDDR2 primitive used with the DDR_ALIGNMENT attribute set to C0 (see the *Spartan-6 FPGA User Guide*). This DDR_ALIGNMENT attribute causes the IDDR2 to output both Q1 and Q2 data on the rising edge of pma_rx_clk0.

For this reason, pma_rx_clk0 can be routed to both pma_rx_clk0 and pma_rx_clk1 clock inputs of the core as illustrated.

> ***Caution!*** This logic relies on pma_rx_clk0 and pma_rx_clk1 being exactly 180 degrees out of phase with each other because the falling edge of pma_rx_clk0 is used in place of pma_rx_clk1. See the data sheet for the attached SERDES to verify that this is the case.

Setup and Hold is achieved using a combination of IODELAY2 elements on the data, and using BUFIO2 elements and BUFG global clock routing for the pma_rx_clk0 input clock, as illustrated in Figure 6-11.

This design provides a simpler solution than the DCM logic required for Virtex-4 devices. It has therefore been chosen as the example design for the Virtex-6 family. However, the Virtex-4 approach could alternatively be adopted: simply replace the DCM with a MMCM module (see Figure 6-5).

In the Figure 6-11 implementation, two BUFIO2s are used to provide the lowest form of clock routing delay from input clock to input rx_code_group[9:0] signal sampling at the device IOBs. One BUFIO2 element is used for the rising edge logic; no clock inversion is performed and the DIVCLK output will connect to the BUFG to provide global clock routing; the IOCLK output of this BUFIO2 is routed to the IDDR2 primitive to sample data on the rising edge. The second BUFIO2 element is configured to invert the clock; the IOCLK output is routed to the IDDR2 to effectively sample the data on the falling edge position of pma_rx_clk0. The DIVCLK output of this BUFIO2 is not used and is left unconnected.

The IODELAY2 elements can be adjusted to fine-tune the setup and hold times at the TBI IOB input flip-flops. The delay is applied to the IODELAY element using constraints in the UCF; these can be edited if desired. See "Ten-Bit Interface Constraints" in Chapter 12 for more information.

Please note, however, that this logic creates placement constraints; rx_code_group[9:0] input signals must be placed in the respective half-bank region for the two BUFIO2 elements in use. The *Spartan-6 FPGA User Guide* should be consulted.

Method 2: An Alternative Using Both pma_rx_clk0 and pma_rx_clk1

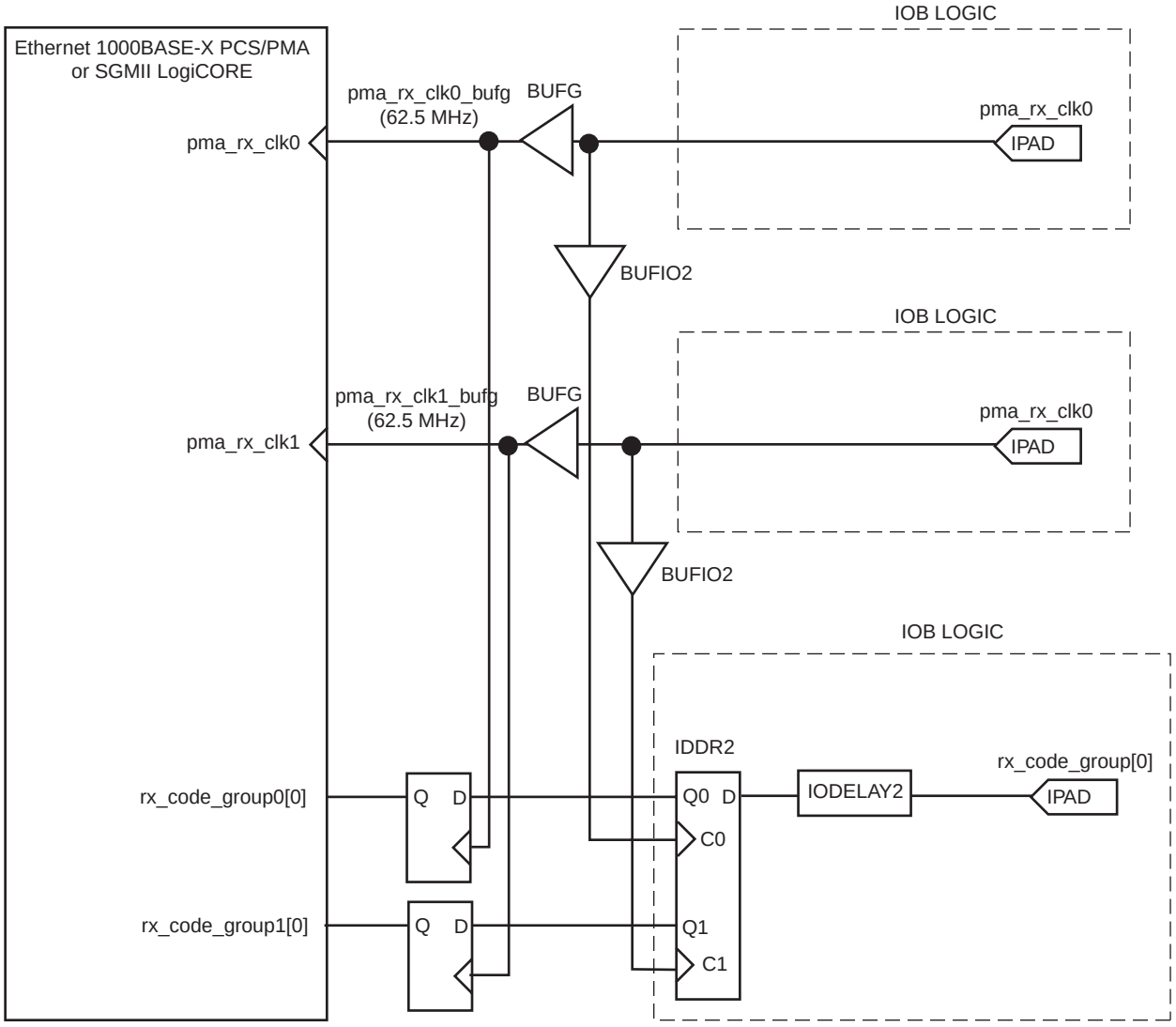


Figure 6-12: **Alternate Ten-Bit Interface Receiver Logic - Spartan-6 Devices**

This logic from method 1 relies on `pma_rx_clk0` and `pma_rx_clk1` being exactly 180 degrees out of phase with each other because the falling edge of `pma_rx_clk0` is used in place of `pma_rx_clk1`. See the data sheet for the attached SERDES to verify that this is the case. If not, the logic of Figure 6-12 illustrates an alternate implementation where both `pma_rx_clk0` and `pma_rx_clk1` are used as intended. Each bit of `rx_code_group[9:0]` must be routed to two separate device pads.

In this method, the logic used on `pma_rx_clk0` in Figure 6-11 is duplicated for `pma_rx_clk1`.

In the figure, a simplified view of the BUFIO2 elements are provided. The connected output of each BUFIO is the IOCLK port. Other BUFIO2 output ports are unused and unconnected.

Clock Sharing across Multiple Cores with TBI

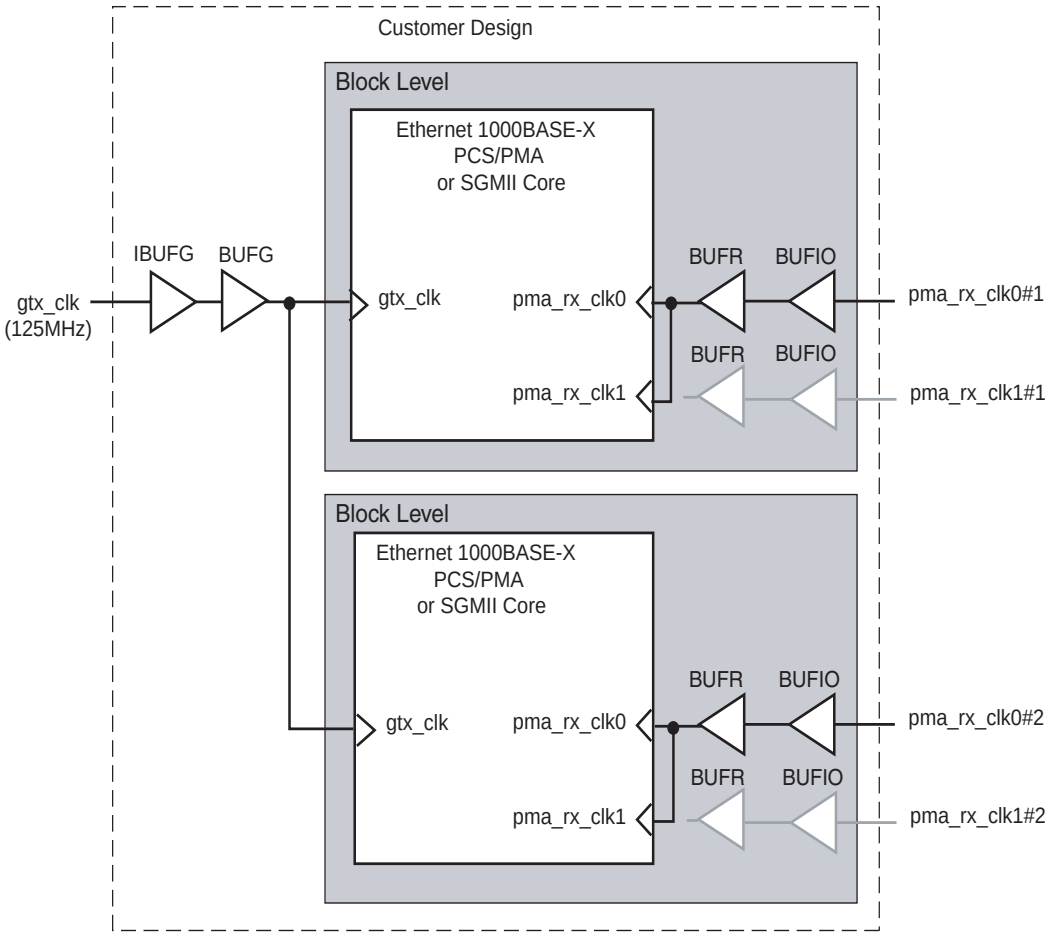


Figure 6-13: **Clock Management, Multiple Core Instances with Ten-Bit Interface**

Figure 6-13 illustrates sharing clock resources across multiple instantiations of the core when using the TBI. For all implementations, `gtx_clk` may be shared between multiple cores, resulting in a common clock domain across the device.

The receiver clocks `pma_rx_clk0` and `pma_rx_clk1` (if used) cannot be shared. Each core will be provided with its own versions of these receiver clocks from its externally connected SERDES. Figure 6-13 illustrates the receiver clock logic used for the Virtex-5 family of Figure 6-7. See "Receiver Logic," page 81, for a description of alternative receiver clock logic for Virtex-5 and for other device families.

Figure 6-13 illustrates only two cores. However, more can be added using the same principle. 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 logic cannot be shared and must be unique for every instance of the core.

Chapter 7

1000BASE-X with Transceivers

This chapter provides general guidelines for creating 1000BASE-X designs that use transceivers for Virtex®-4, Virtex-5, Virtex-6 and Spartan®-6 devices. Information about transceiver and core logic in all supported device families is provided, as well as information about designs requiring multiple instantiations of the core. Clock sharing should occur whenever possible to save device resources.

Transceiver Logic

The example is split between two discrete hierarchical layers, as illustrated in Figure 4-1. The block level is designed so that it can be instantiated directly into customer designs and provides the following functionality:

- Instantiates the core from HDL
- Connects the physical-side interface of the core to a Virtex-4, Virtex-5 or Virtex-6 FPGA transceiver

The logic implemented in the block level is illustrated in all the figures in this chapter.

Virtex-4 FX Devices

The core is designed to integrate with the Virtex-4 FPGA RocketIO™ MGT transceiver. Figure 7-1 illustrates the connections and logic required between the core and MGT—the signal names and logic in the figure precisely match those delivered with the example design when an MGT is used.

Note: A small logic shim (included in the *block*-level wrapper) is required to convert between the port differences between the Virtex-5 and Virtex-4 FPGA RocketIO transceivers.

The MGT clock distribution in Virtex-4 devices is column-based and consists of multiple MGT tiles (each tile contains two MGTs). For this reason, the MGT wrapper delivered with the core always contains two MGT instantiations, even if only a single MGT is in use. Figure 7-1 illustrates a single MGT tile for clarity.

A `GT11CLK_MGT` primitive is also instantiated to derive the reference clocks required by the MGT column-based tiles. See the *Virtex-4 RocketIO Multi-Gigabit Transceiver User Guide* (UG076) for information about layout and clock distribution.

The 250 MHz reference clock from the `GT11CLK_MGT` primitive is routed to the MGT, configured to internally synthesize a 125 MHz clock. This is output on the `TXOUTCLK1` port of the MGT and after placed onto global clock routing, can be used by all core logic. This clock is input back into the MGT on the user interface clock ports `rxusrclk2` and `txusrclk2`. With the attribute settings applied to the MGT from the example design, the `txusrclk` and `rxusrclk` ports are derived internally within the MGT using the internal clock dividers and do not need to be provided from the FPGA fabric.

The Virtex-4 FX FPGA RocketIO MGT transceivers require the inclusion of a calibration block in the fabric logic; the example design provided with the core instantiates calibration blocks as required. Calibration blocks require a clock source of between 25 to 50 MHz that is shared with the Dynamic Reconfiguration Port (DRP) of the MGT, which is named `dclk` in the example design. See Xilinx Answer Record 22477 for more information.

Figure 7-1 also illustrates the `TX_SIGNAL_DETECT` and `RX_SIGNAL_DETECT` ports of the calibration block, which should be driven to indicate whether or not dynamic data is being transmitted and received through the MGT (see Virtex-4 Errata). However, `RX_SIGNAL_DETECT` is connected to the `signal_detect` port of the example design. `signal_detect` is intended to be connected to the optical transceiver to indicate the presence of light. When light is detected, the optical transceiver provides dynamic data to the Rx ports of the MGT. When no light is detected, the calibration block switches the MGT into loopback to force dynamic data through the MGT receiver path.

> ***Caution!*** `signal_detect` is an optional port in the *IEEE 802.3* specification. If this is not used, the `RX_SIGNAL_DETECT` port of the calibration block must be driven by an alternative method. Please see *XAPP732* for more information.

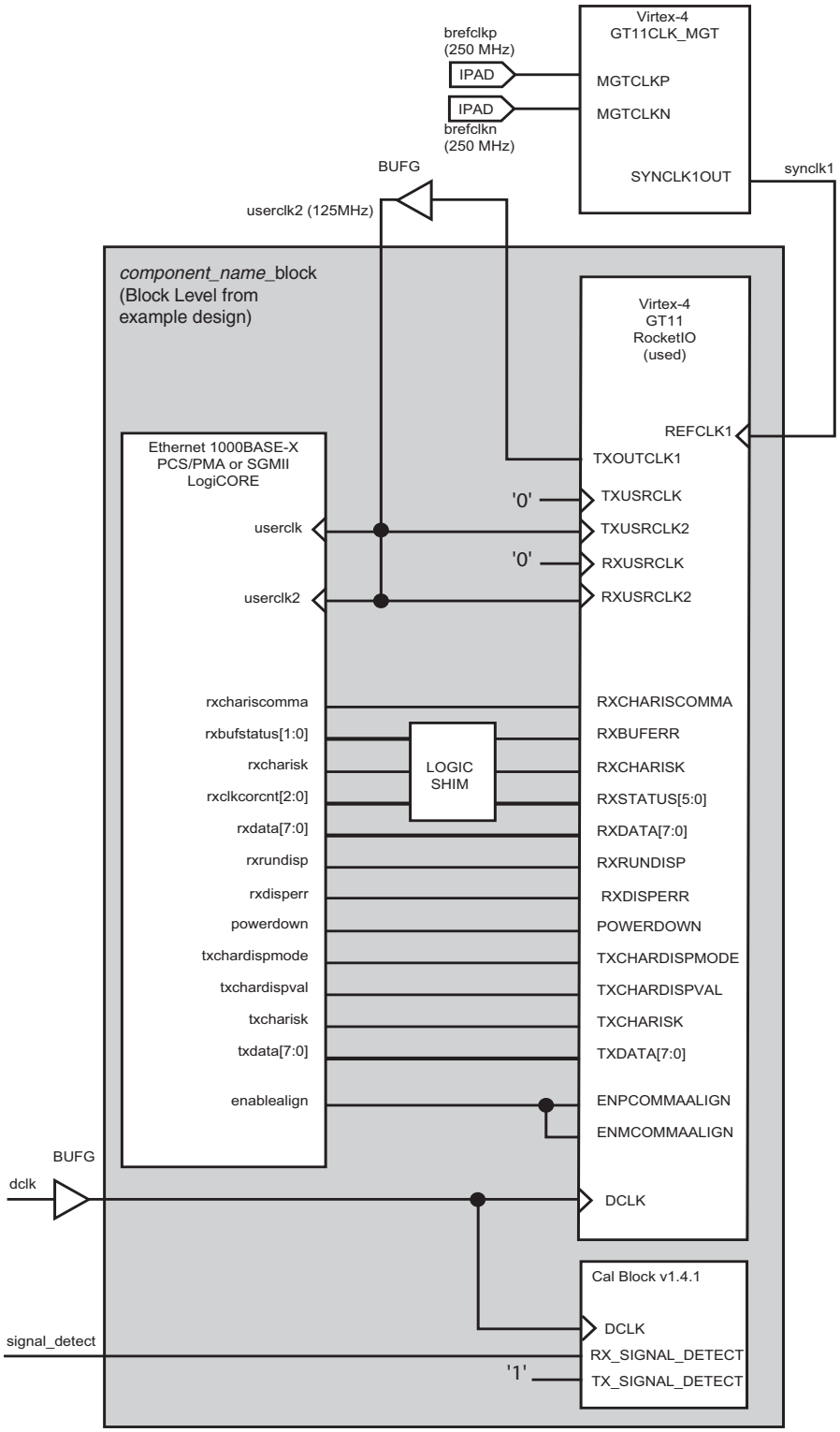


Figure 7-1: **1000BASE-X Connection to Virtex-4 FPGA RocketIO MGT Transceiver**

Virtex-5 LXT and SXT Devices

The core is designed to integrate with the Virtex-5 FPGA RocketIO GTP transceiver. Figure 7-2 illustrates the connections and logic required between the core and the GTP transceiver—the signal names and logic in the figure precisely match those delivered with the example design when a GTP transceiver is used.

A GTP tile consists of a pair of transceivers. For this reason, the GTP transceiver wrapper delivered with the core always contains two GTP instantiations, even if only a single GTP transceiver tile is in use. Figure 7-2 illustrates a single GTP transceiver tile.

The 125 MHz differential reference clock is routed directly to the GTP transceiver. The GTP transceiver is configured to output a version of this clock on the `REFCLKOUT` port and after placement onto global clock routing, can be used by all core logic. This clock is input back into the GTP transceiver on the user interface clock ports `rxusrclk`, `rxusrclk2`, `txusrclk`, and `txusrclk2`.

See also "Virtex-5 FPGA RocketIO GTP Transceivers for 1000BASE-X Constraints," page 192.

Virtex-5 FPGA RocketIO GTP Transceiver Wizard

The two wrapper files immediately around the GTP transceiver pair, `RocketIO_wrapper_gtp_tile` and `RocketIO_wrapper_gtp` (see Figure 7-2), are generated from the *RocketIO GTP Wizard*. These files apply all the gigabit Ethernet attributes. Consequently, these files can be regenerated by customers and therefore be easily targeted at ES or Production silicon. This core targets production silicon.

The CORE Generator™ software log file (XCO file) which was created when the *RocketIO GTP Wizard* project was generated is available in the following location:

```
<project_directory>/<component_name>/example_design/transceiver/
RocketIO_wrapper_gtp.xco
```

This file can be used as an input to the CORE Generator software to regenerate the device-specific RocketIO transceiver wrapper files. The XCO file itself contains a list of all of the GTP Wizard attributes which were used. For further information, please see the *Virtex-5 FPGA RocketIO GTP Wizard Getting Started Guide* (UG188) and the *CORE Generator Guide*, at www.xilinx.com/support/software_manuals.htm

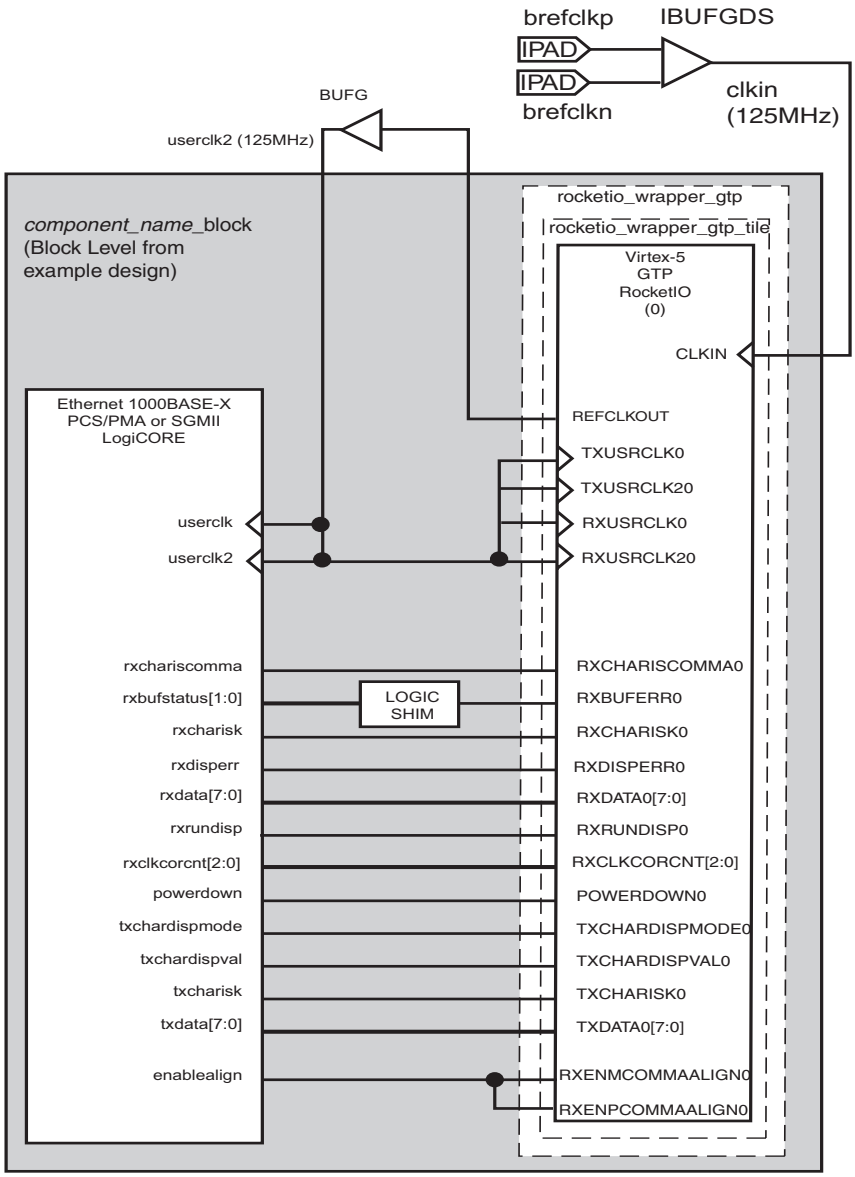


Figure 7-2: **1000BASE-X Connection to Virtex-5 FPGA RocketIO GTP Transceivers**

Virtex-5 FXT and TXT Devices

The core is designed to integrate with the Virtex-5 FPGA RocketIO GTX transceiver. Figure 7-3 illustrates the connections and logic required between the core and the GTX transceiver—the signal names and logic in the figure precisely match those delivered with the example design when a GTX transceiver is used.

A GTX tile consists of a pair of transceivers. For this reason, the GTX transceiver wrapper delivered with the core always contains two GTX instantiations, even if only a single GTX transceiver tile is in use. Figure 7-3 illustrates a single GTX transceiver tile.

The 125 MHz differential reference clock is routed directly to the GTX transceiver. The GTX transceiver is configured to output a version of this clock on the `REFCLKOUT` port; this is then routed to a DCM via a BUFG (global clock routing).

From the DCM, the `CLK0` port (125 MHz) is placed onto global clock routing and can be used as the 125 MHz clock source for all core logic; this clock is also input back into the GTX transceiver on the user interface clock ports `rxusrclk2` and `txusrclk2`.

From the DCM, the `CLKDV` port (62.5 MHz) is placed onto global clock routing and is input back into the GTX transceiver on the user interface clock ports `rxusrclk` and `txusrclk`.

See also "Virtex-5 FPGA RocketIO GTX Transceivers for 1000BASE-X Constraints," page 193.

Virtex-5 FPGA RocketIO GTX Wizard

The two wrapper files immediately around the GTX transceiver pair, `RocketIO_wrapper_gtx_tile` and `RocketIO_wrapper_gtx` (see Figure 7-3), are generated from the *RocketIO GTX Wizard*. These files apply all the gigabit Ethernet attributes. Consequently, these files can be regenerated by customers and therefore be easily targeted at ES or Production silicon. This core targets production silicon.

The CORE Generator software log file (XCO file) which was created when the *RocketIO GTX Wizard* project was generated is available in the following location:

```
<project_directory>/<component_name>/example_design/transceiver/
RocketIO_wrapper_gtx.xco
```

This file can be used as an input to the CORE Generator software to regenerate the device-specific RocketIO transceiver wrapper files. The XCO file itself contains a list of all of the GTX Wizard attributes which were used. For further information, please see the *Virtex-5 FPGA RocketIO GTX Wizard Getting Started Guide* and the *CORE Generator Guide*, at www.xilinx.com/support/software_manuals.htm

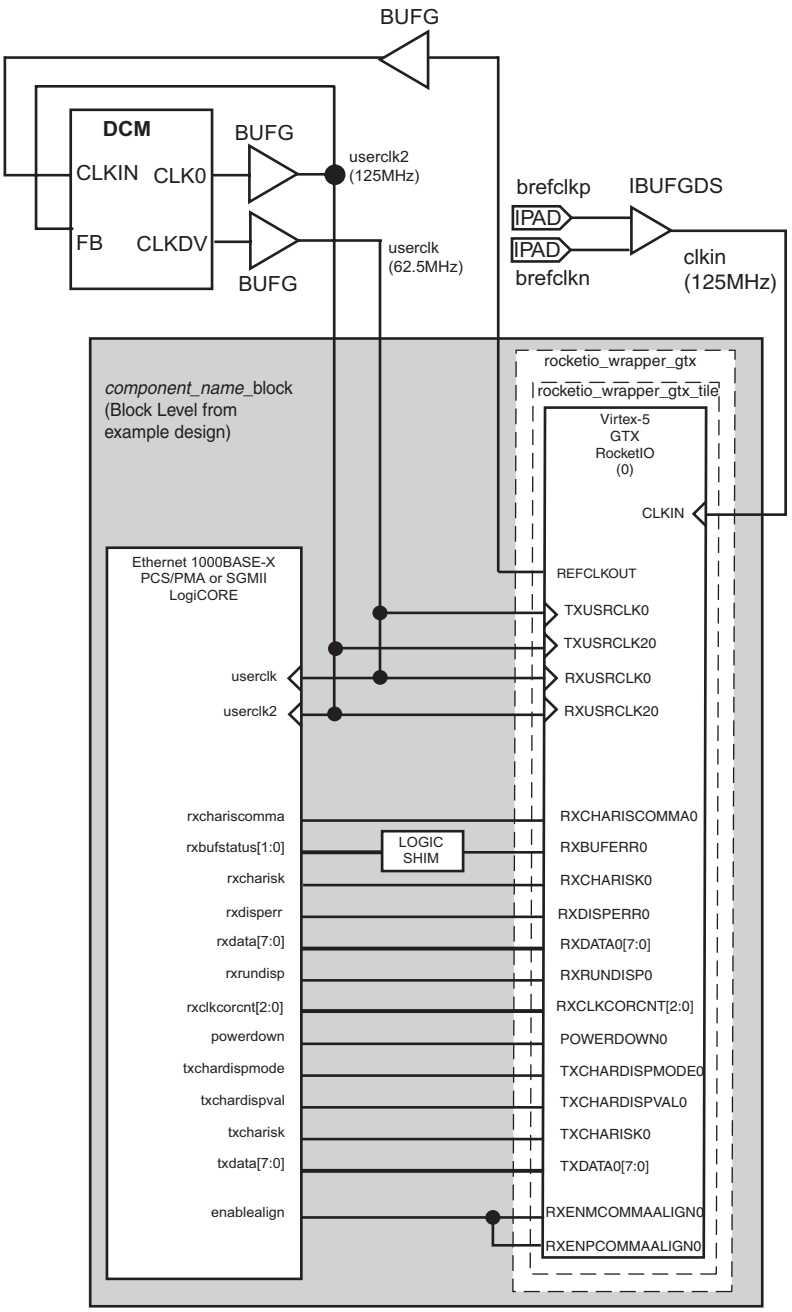


Figure 7-3: **1000BASE-X Connection to Virtex-5 FPGA RocketIO GTX Transceivers**

Virtex-6 Devices

The core is designed to integrate with the Virtex-6 FPGA GTX transceiver. Figure 7-4 illustrates the connections and logic required between the core and the GTX transceiver—the signal names and logic in the figure precisely match those delivered with the example design.

The 125 MHz differential reference clock is routed directly to the GTX transceiver from the specialized IBUFDS_GTXE1 primitive. The GTX transceiver is configured to output a version of this clock on the `TXOUTCLK` port and after placement onto global clock routing, can be used by all core logic. This clock is input back into the GTX transceiver on the user interface clock ports `rxusrclk2` and `txusrclk2`. The `rxusrclk` and `txusrclk` clocks will be derived internally and can be grounded.

Virtex-6 FPGA GTX Transceiver Wizard

The two wrapper files immediately around the GTX transceiver.

`gtx_wrapper_gtx` and `gtx_wrapper` (see Figure 7-4), are generated from the *Virtex-6 FPGA GTX Transceiver Wizard*. These files apply all the gigabit Ethernet attributes. Consequently, these files can be regenerated by customers and therefore be easily targeted at silicon/device versions.

The CORE Generator software log file (XCO file) which was created when the *Virtex-6 FPGA GTX Transceiver Wizard* project was generated is available in the following location:

```
<project_directory>/<component_name>/example_design/transceiver/
gtx_wrapper_gtx.xco
```

This file can be used as an input to the CORE Generator software to regenerate the transceiver wrapper files. The XCO file itself contains a list of all of the Wizard attributes which were used. For further information, please see the *Virtex-6 FPGA GTX Transceiver Wizard Getting Started Guide* and the *CORE Generator Guide*, at www.xilinx.com/support/software_manuals.htm

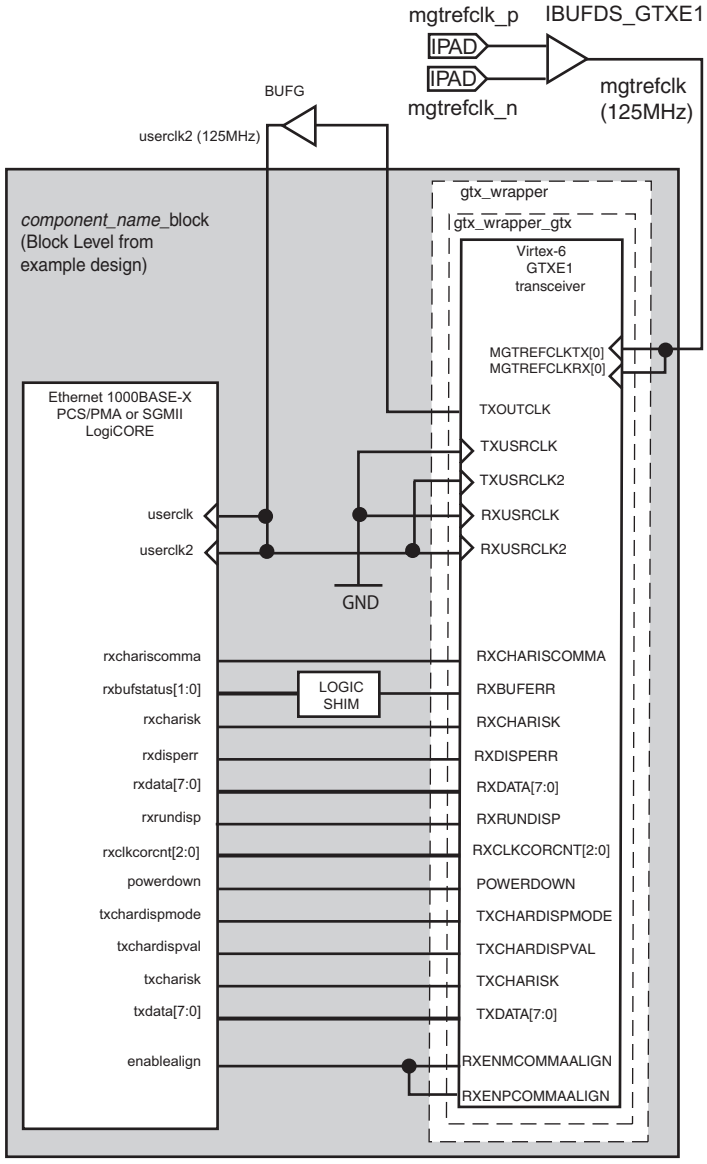


Figure 7-4: **1000BASE-X Connection to Virtex-6 FPGA GTX Transceiver**

Spartan-6 LXT Devices

The core is designed to integrate with the Spartan-6 FPGA GTP transceiver. Figure 7-5 illustrates the connections and logic required between the core and the GTP transceiver—the signal names and logic in the figure precisely match those delivered with the example design when a GTP transceiver is used.

A GTP tile consists of a pair of transceivers. For this reason, the GTP transceiver wrapper delivered with the core always contains two GTP instantiations, even if only a single GTP transceiver tile is in use. Figure 7-5 illustrates a single GTP transceiver tile.

The 125 MHz differential reference clock is routed directly to the GTP transceiver. The GTP transceiver is configured to output a version of this clock on the `GTPCLKOUT` port and after placement through a BUFIO2 and BUFG onto global clock routing, can be used by all core logic. This clock is input back into the GTP transceiver on the user interface clock ports `rxusrclk`, `rxusrclk2`, `txusrclk`, and `txusrclk2`.

See also "Spartan-6 FPGA GTP Transceivers for 1000BASE-X Constraints," page 196.

Spartan-6 FPGA GTP Transceiver Wizard

The two wrapper files immediately around the GTP transceiver pair, `gtp_wrapper_tile` and `gtp_wrapper` (see Figure 7-5), are generated from the Spartan-6 FPGA *GTP Wizard*. These files apply all the gigabit Ethernet attributes. Consequently, these files can be regenerated by customers and therefore be easily targeted at ES or Production silicon. This core targets production silicon.

The CORE Generator software log file (XCO file) which was created when the *GTP Wizard* project was generated is available in the following location:

```
<project_directory>/<component_name>/example_design/transceiver/
gtp_wrapper.xco
```

This file can be used as an input to the CORE Generator software to regenerate the device-specific transceiver wrapper files. The XCO file itself contains a list of all of the GTP Wizard attributes that were used. For further information, please see the *Spartan-6 FPGA GTP Wizard Getting Started Guide* and the *CORE Generator Guide*, at www.xilinx.com/support/software_manuals.htm

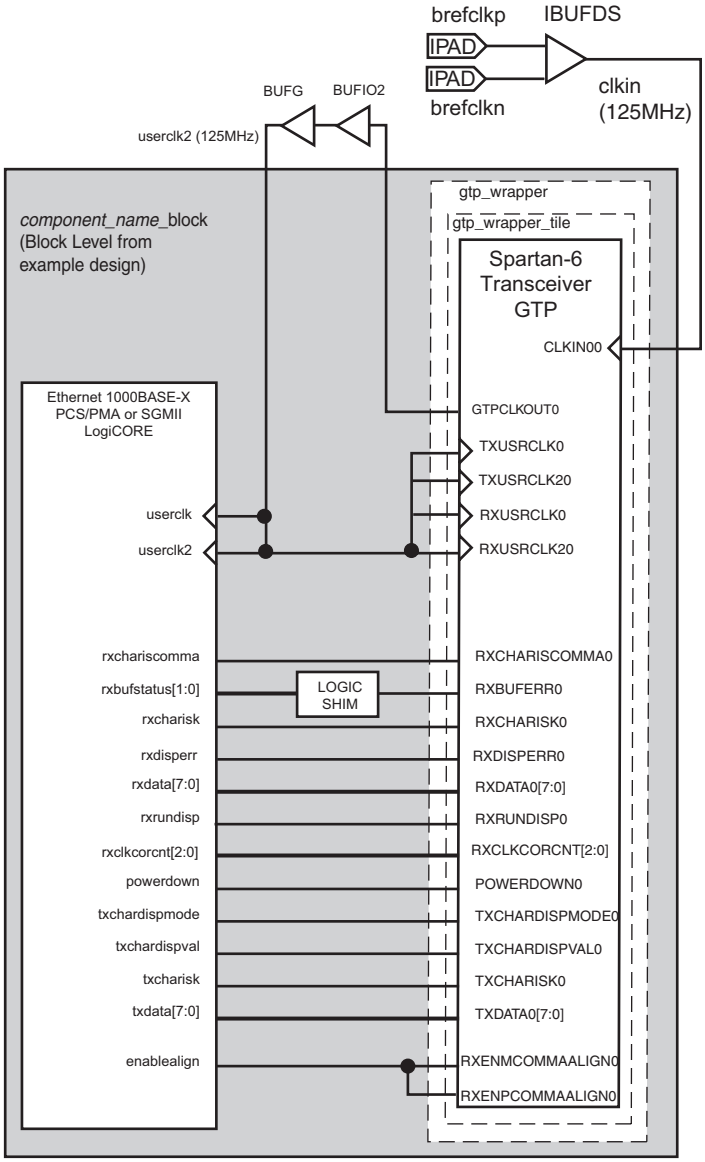


Figure 7-5: **1000BASE-X Connection to Spartan-6 FPGA GTP Transceivers**

Clock Sharing Across Multiple Cores with Transceivers

Virtex-4 FX Devices

Figure 7-6 illustrates sharing clock resources across multiple instantiations of the core when using MGTs. The example design, when using the Virtex-4 family, can be generated to connect either a single instance of the core, or connect a pair of core instances to the transceiver pair present in an MGT tile. Figure 7-6 shows two instantiations of the block level, where each block contains a pair of cores, subsequently illustrating clock sharing between four cores in total.

More cores can be added by continuing to instantiate extra block-level modules. Share clocks only between the MGTs in a single column. For each column, use a single `brefclk_p` and `brefclk_n` differential clock pair and connect this to a `GT11CLK_MGT` primitive. The clock output from this primitive should be shared across all used RocketIO transceiver tiles in the column. See the *Virtex-4 RocketIO Multi-Gigabit Transceiver User Guide* (UG076) for more information.

To provide the 125 MHz clock for all core instances, select a `TXOUTCLK1` port from any MGT. This can be routed onto global clock routing using a BUFG as illustrated, and shared between all cores and MGTs in the column. Although not illustrated in Figure 7-6, `dclk` (the clock used for the calibration blocks and for the Dynamic Reconfiguration Port (DRP) of the MGTs) can also be shared.

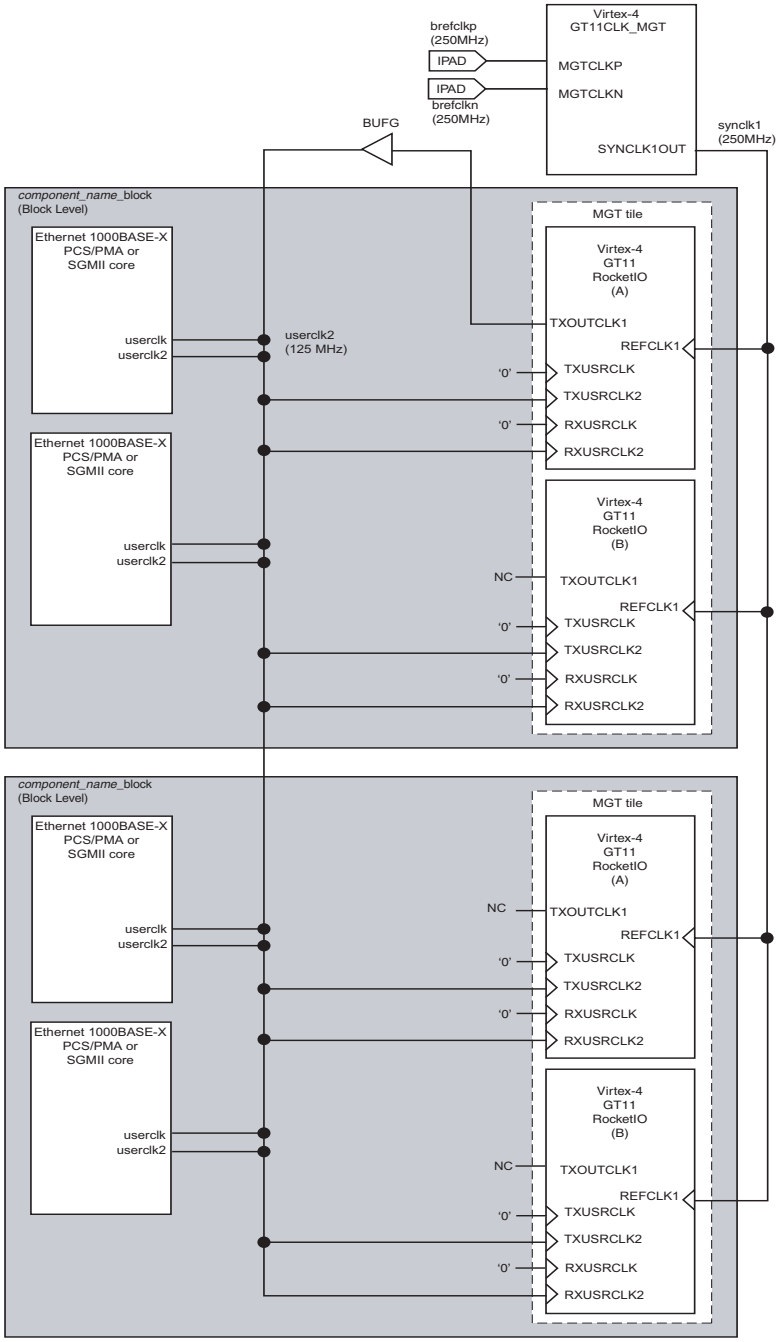


Figure 7-6: **Clock Management - Multiple Core Instances, MGTs for 1000BASE-X**

Virtex-5 LXT and SXT Devices

Figure 7-7 illustrates sharing clock resources across multiple instantiations of the core when using Virtex-5 FPGA RocketIO GTP transceivers.

The example design can be generated to connect either a single instance of the core or connect a pair of core instances to the transceiver pair present in a GTP tile. Figure 7-7 illustrates two instantiations of the block level, and each block level contains a pair of cores, consequently illustrating clock sharing between a total of four cores.

Additional cores can be added by continuing to instantiate extra block level modules. Share the `brefclk_p` and `brefclk_n` differential clock pair. See the *Virtex-5 FPGA RocketIO GTP Transceiver User Guide* (UG196) for more information.

To provide the 125 MHz clock for all core instances, select a `REFCLKOUT` port from any GTP transceiver. This can be routed onto global clock routing using a BUFG as illustrated and shared between all cores and GTP transceivers.

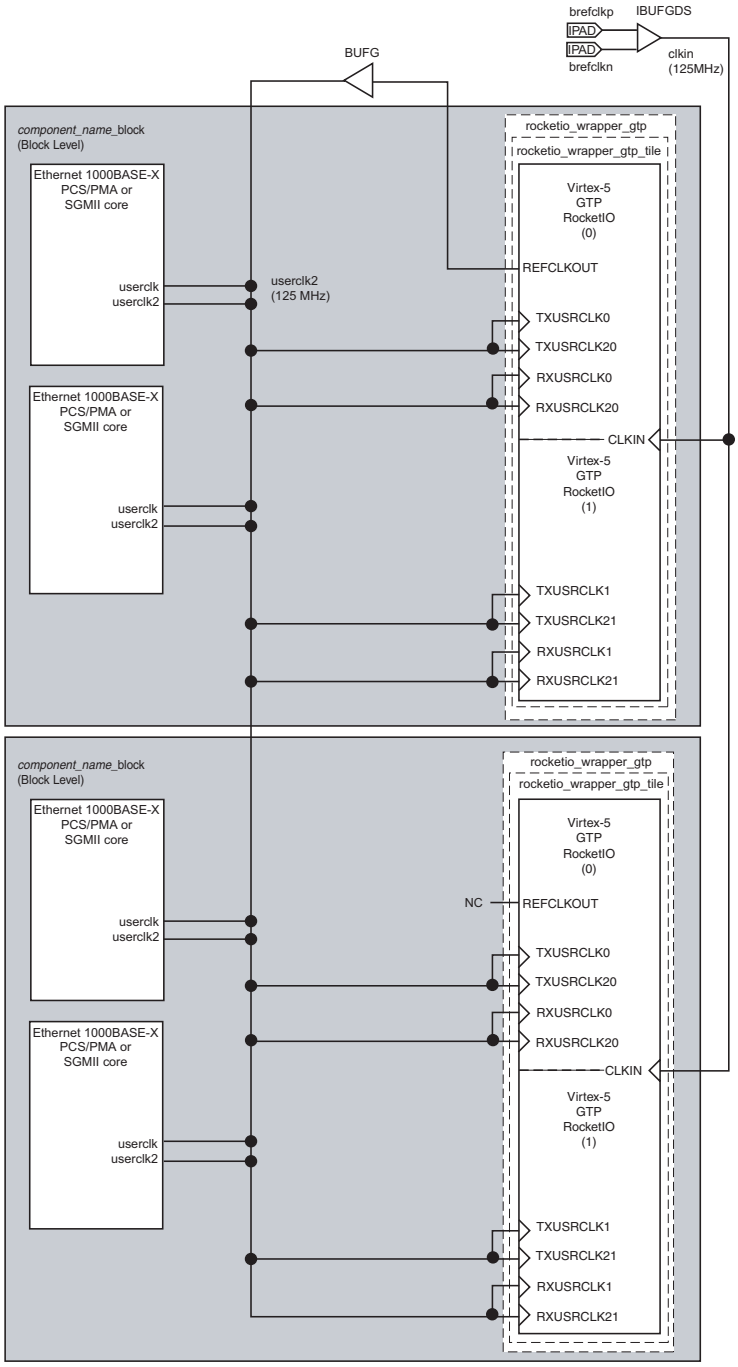


Figure 7-7: **Clock Management - Multiple Core Instances, Virtex-5 FPGA RocketIO GTP Transceivers for 1000BASE-X**

Virtex-5 FXT and TXT Devices

Figure 7-8 illustrates sharing clock resources across multiple instantiations of the core when using Virtex-5 FPGA RocketIO GTX transceivers.

The example design can be generated to connect either a single instance of the core or connect a pair of core instances to the transceiver pair present in a GTX tile. Figure 7-8 illustrates two instantiations of the block level, and each block level contains a pair of cores, consequently illustrating clock sharing between a total of four cores.

Additional cores can be added by continuing to instantiate extra block level modules. Share the `brefclk_p` and `brefclk_n` differential clock pair. See the *Virtex-5 FPGA RocketIO GTX Transceiver User Guide* for more information.

To provide the FPGA fabric clocks for all core instances, select a `REFCLKOUT` port from any GTX transceiver and route this to a single DCM via a BUFG (global clock routing). The `CLK0` (125 MHz) and `CLKDV` (62.5 MHz) outputs from this DCM, placed onto global clock routing using BUFGs, can be shared across all core instances and GTX transceivers as illustrated.

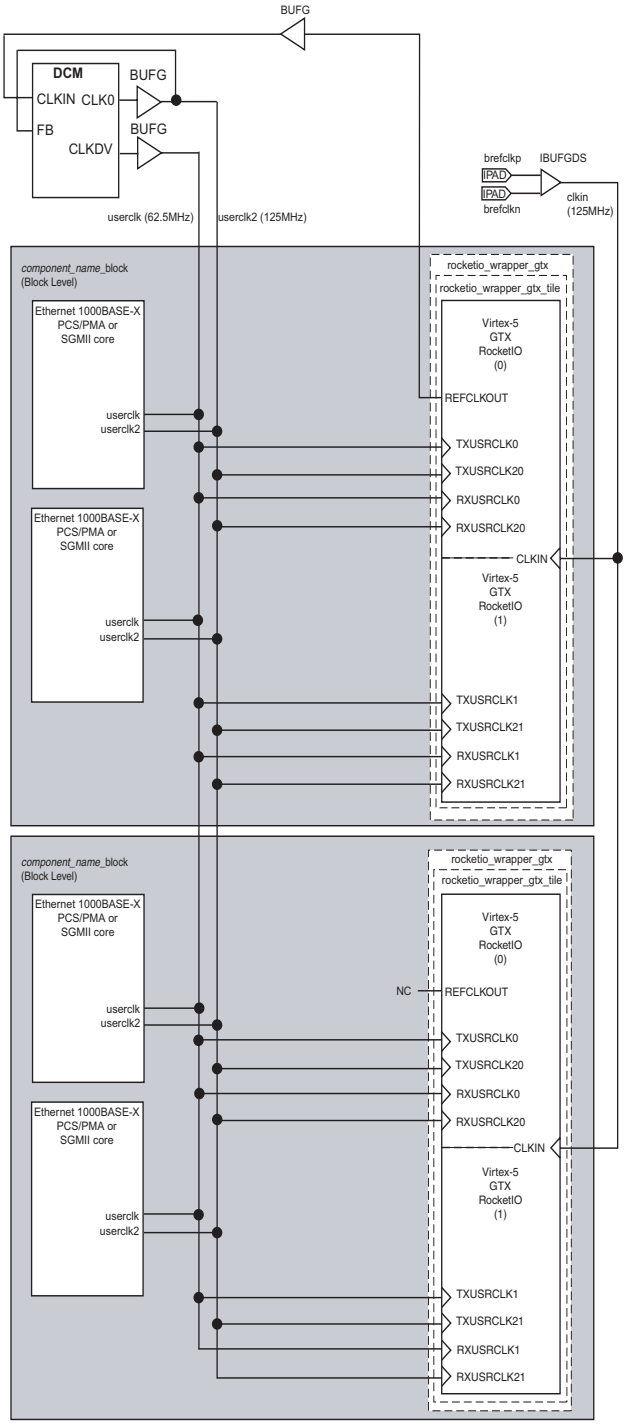


Figure 7-8: **Clock Management - Multiple Core Instances, Virtex-5 FPGA RocketIO GTX Transceivers for 1000BASE-X**

Virtex-6 Devices

Figure 7-9 illustrates sharing clock resources across two instantiations of the core when using Virtex-6 FPGA GTX transceivers. Additional cores can be added by continuing to instantiate extra block level modules.

Share the `mgtrefclk_p` and `mgtrefclk_n` differential clock pair clock source across all of the transceivers in use. To provide the 125 MHz clock for all core instances, select a `TXOUTCLK` port from any GTX transceiver. This can be routed onto global clock routing using a BUFG as illustrated and shared between all cores and GTX transceivers.

See the *Virtex-6 GTX Transceiver User Guide* for more information on GTX clock resources.

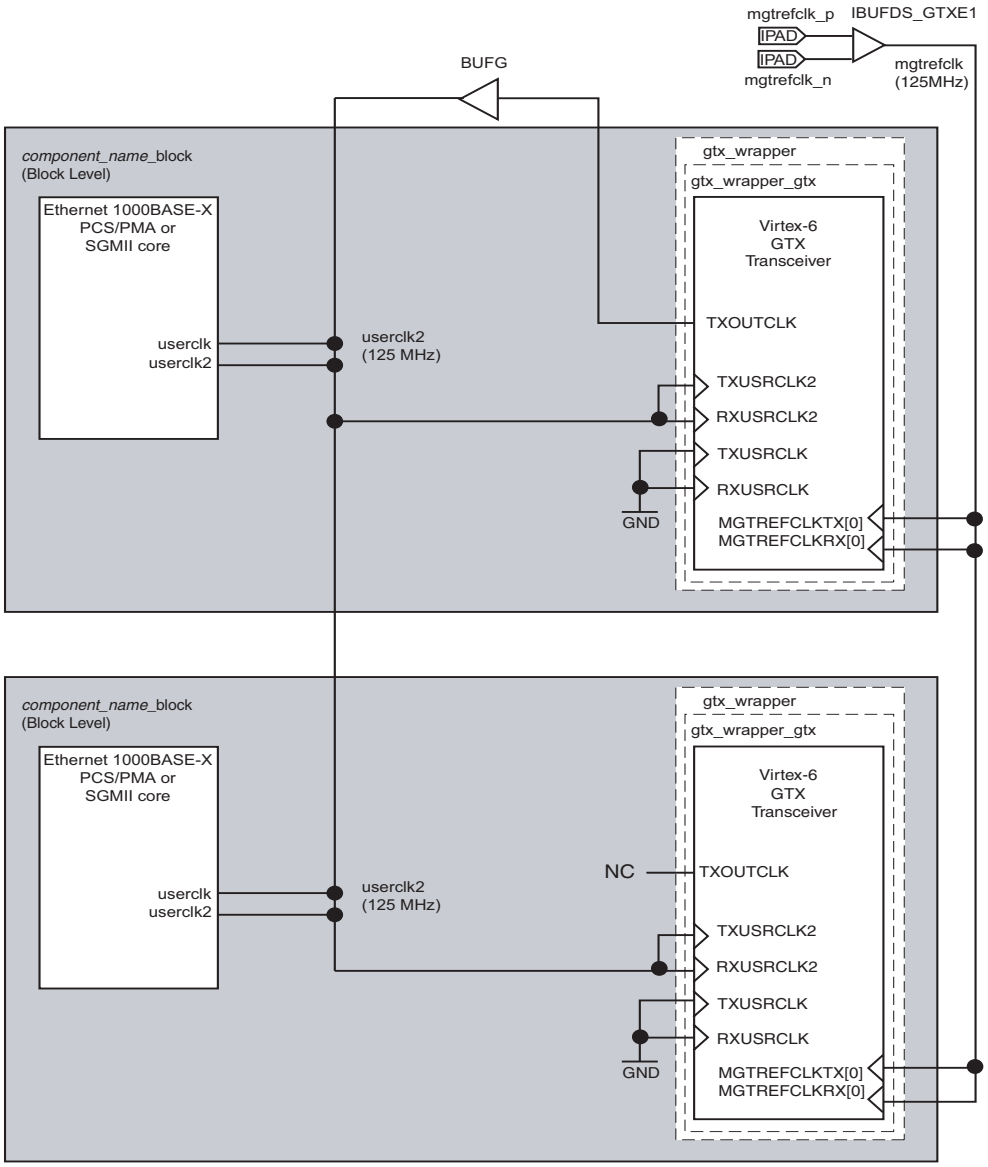


Figure 7-9: **Clock Management - Multiple Core Instances, Virtex-6 FPGA GTX Transceivers for 1000BASE-X**

Spartan-6 LXT Devices

Figure 7-10 illustrates sharing clock resources across multiple instantiations of the core when using Spartan-6 FPGA GTP transceivers.

The example design can be generated to connect either a single instance of the core or connect a pair of core instances to the transceiver pair present in a GTP tile. Figure 7-10 illustrates two instantiations of the block level, and each block level contains a pair of cores, consequently illustrating clock sharing between a total of four cores.

Additional cores can be added by continuing to instantiate extra block level modules. Share the `brefclk_p` and `brefclk_n` differential clock pair. See the *Spartan-6 FPGA GTP Transceiver User Guide* for more information.

To provide the 125 MHz clock for all core instances, select a `GTPCLKOUT` port from any GTP transceiver. This can be routed onto global clock routing using a BUFIO2 and BUFG as illustrated and shared between all cores and GTP transceivers.

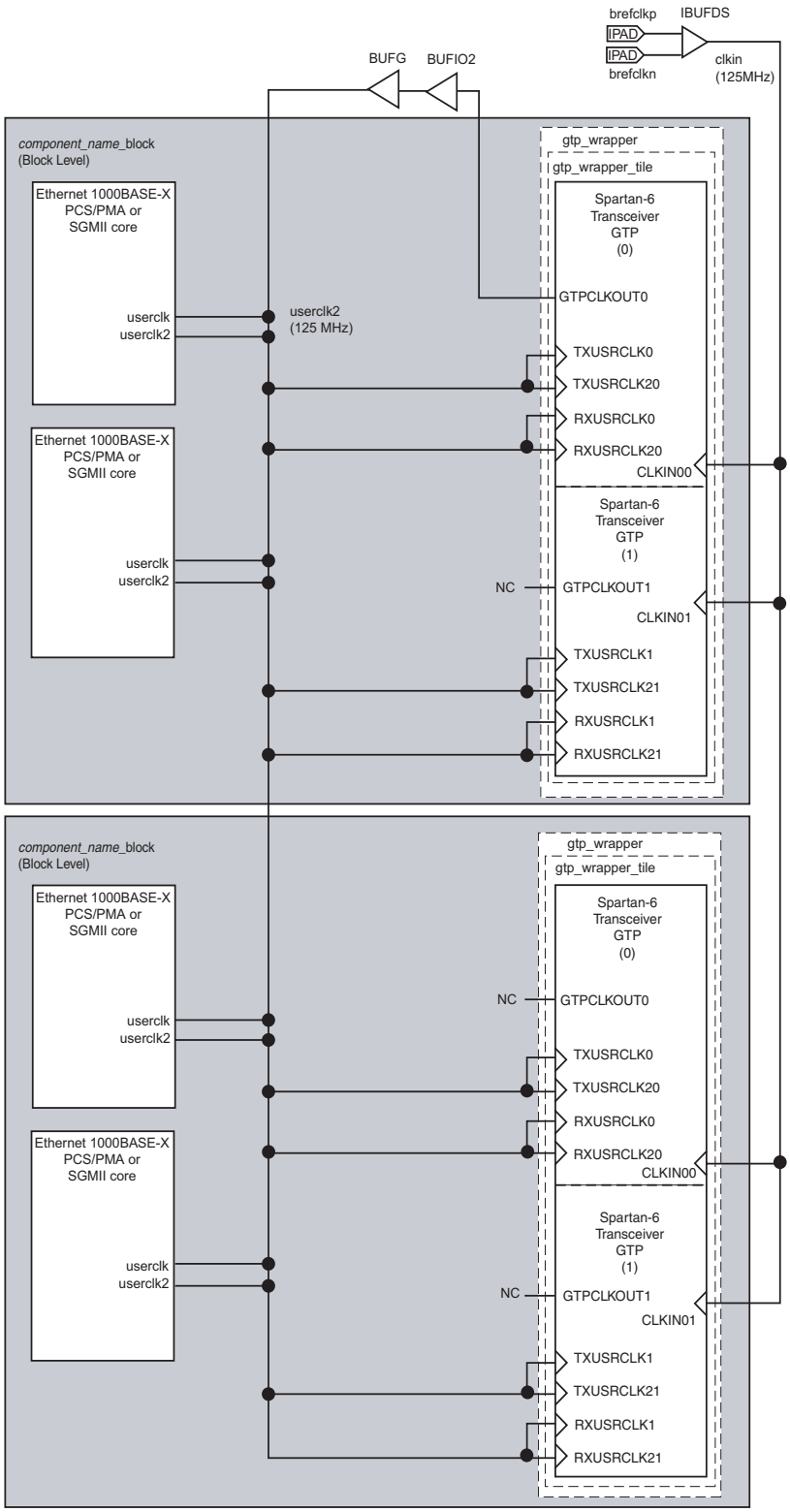


Figure 7-10: **Clock Management - Multiple Core Instances, Spartan-6 FPGA GTP Transceivers for 1000BASE-X**

XILINX®

Chapter 8

SGMII / Dynamic Standards Switching with Transceivers

This chapter provides general guidelines for creating SGMII designs, and designs capable of switching between 1000BASE-X and SGMII standards (Dynamic Standards Switching), using a device-specific transceiver. Throughout this chapter, any reference to SGMII also applies to the Dynamic Standards Switching implementation.

The chapter begins with an explanation of the two Receiver Elastic Buffer implementations: one implementation uses the buffer present in the device-specific transceivers, and the other uses a larger buffer, implemented in the FPGA fabric.

After selecting the Rx Elastic Buffer implementation type, an explanation of the device-specific transceiver and core logic in all supported device families is provided in the following sections:

- "Transceiver Logic Using the Rx Elastic Buffer," page 118
- "Transceiver Logic with the Fabric Rx Elastic Buffer," page 119

Instances where multiple instantiations of the core are required when using the fabric Receiver Elastic Buffer are then presented. Clock sharing should occur whenever possible to save device resources.

Receiver Elastic Buffer Implementations

Selecting the Buffer Implementation from the GUI

The GUI provides two SGMII Capability options:

- 10/100/1000 Mbps (clock tolerance compliant with Ethernet specification)
- 10/100/1000 Mbps (restricted tolerance for clocks) OR 100/1000 Mbps

The first option, 10/100/1000 Mbps (clock tolerance compliant with Ethernet specification) is the default and provides the implementation using the Receiver Elastic Buffer in FPGA fabric. This alternative Receiver Elastic Buffer uses a single block RAM to create a buffer twice as large as the one present in the device-specific transceiver, for this reason consuming extra logic resources. However, this default mode is reliable for all implementations using standard Ethernet frame sizes. Further consideration must be made for jumbo frames.

The second option, 10/100/1000 Mbps (restricted tolerance for clocks) or 100/1000 Mbps, uses the receiver elastic buffer present in the device-specific transceivers. This is half the size and can potentially underflow or overflow during SGMII frame reception at 10 Mbps operation (see the next section). However, there are logical implementations where this can be reliable and has the benefit of lower logic utilization.

The Requirement for the FPGA Fabric Rx Elastic Buffer

Figure 8-1 illustrates a simplified diagram of a common situation where the core, in SGMII mode, is interfaced to an external PHY device. Separate oscillator sources are used for the FPGA and the external PHY. The Ethernet specification uses clock sources with a tolerance of 100ppm. In Figure 8-1, the clock source for the PHY is slightly faster than the clock source to the FPGA. For this reason, during frame reception, the receiver elastic buffer (shown here as implemented in the device-specific transceiver) starts to fill.

Following frame reception, in the interframe gap period, idles are removed from the received data stream to return the Rx Elastic Buffer to half-full occupancy. This is performed by the clock correction circuitry (see the device-specific transceiver User Guide for the targeted device).

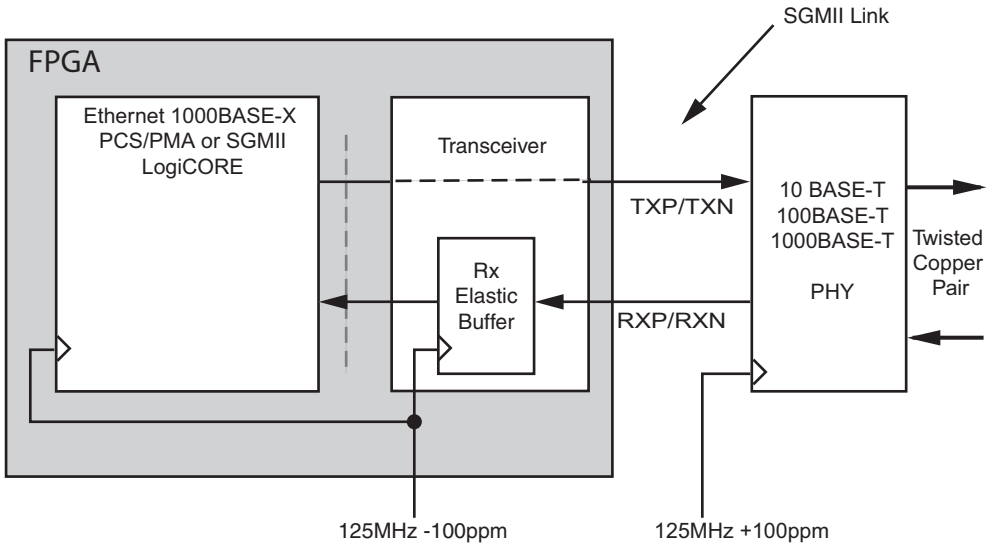


Figure 8-1: **SGMII Implementation using Separate Clock Sources**

Analysis

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

Relating this to an Ethernet frame, there will be a single byte of difference every 5000 bytes of received frame data, and this will cause the Rx Elastic Buffer to either fill or empty by an occupancy of one.

The maximum Ethernet frame size (non-jumbo) is 1522 bytes for a VLAN frame.

- At 1 Gbps operation, this translates into 1522 clock cycles.
- At 100 Mbps operation, this translates into 15220 clock cycles (as each byte is repeated 10 times).
- At 10 Mbps operation, this translates into 152200 clock cycles (as each byte is repeated 100 times).

Considering the 10 Mbps case, we would need 152200/5000 = 31 FIFO entries in the Elastic Buffer above and below the half way point to guarantee that the buffer will not under or overflow during frame reception. This assumes that frame reception begins when the buffer is exactly half full.

The size of the Rx Elastic Buffer in the device-specific transceivers is 64 entries. However, we cannot assume that the buffer is exactly half full at the start of frame reception. Additionally, the underflow and overflow thresholds are not exact (see Appendix E, "Rx Elastic Buffer Specifications" for more information).

To guarantee reliable SGMII operation at 10 Mbps (non-jumbo frames), the device-specific transceiver Elastic Buffer must be bypassed and a larger buffer implemented in the FPGA fabric. The fabric buffer, provided by the example design, is twice the size of the device-specific transceiver alternative. This has been proven to cope with standard (none jumbo) Ethernet frames at all three SGMII speeds.

Appendix E, "Rx Elastic Buffer Specifications" provides further information about all Rx Elastic Buffers used by the core. Information about the reception of jumbo frames is also provided.

The Transceiver Rx Elastic Buffer

The Elastic Buffer in the device-specific transceiver can be used reliably when the following conditions are met:

- 10 Mbps operation is not required. Both 1 Gbps and 100 Mbps operation can be guaranteed.
- When the clocks are closely related (see the following section).

If there is any doubt, select the FPGA fabric Rx Elastic Buffer Implementation.

Closely Related Clock Sources

Case 1

Figure 8-2 illustrates a simplified diagram of a common situation where the core, in SGMII mode, is interfaced to an external PHY device. A common oscillator source is used for both the FPGA and the external PHY.

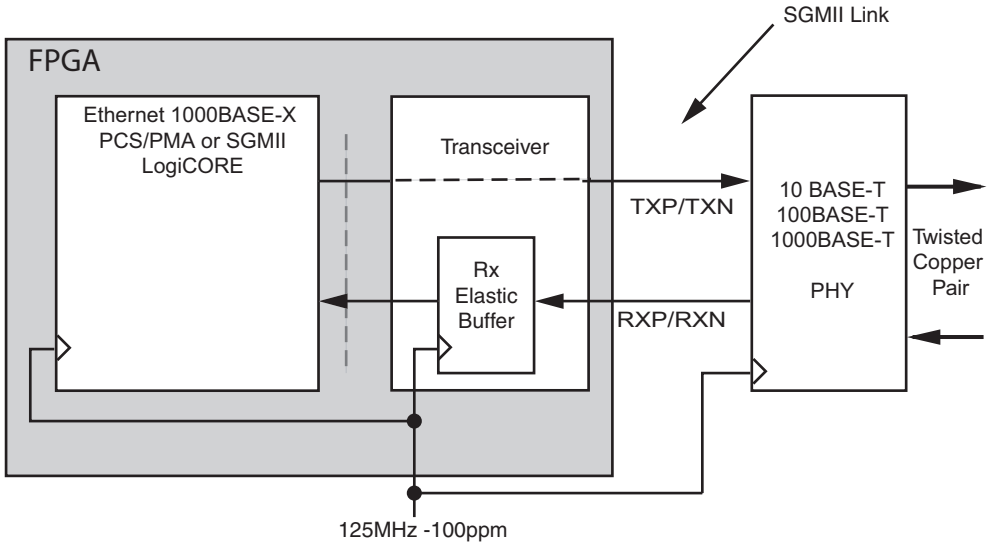


Figure 8-2: **SGMII Implementation using Shared Clock Sources**

If the PHY device sources the receiver SGMII stream synchronously from the shared oscillator (check PHY data sheet), the device-specific transceiver will receive data at exactly the same rate as that used by the core. The receiver elastic buffer will neither empty nor fill, having the same frequency clock on either side.

In this situation, the receiver elastic buffer will not under or overflow, and the elastic buffer implementation in the device-specific transceiver should be used to save logic resources.

Case 2

Consider again the case illustrated in Figure 8-1 with the following exception; assume that the clock sources used are both 50 ppm. Now the maximum frequency difference between the two devices is 100 ppm. It can be shown that this translates into a full clock period difference every 10000 clock periods, resulting in a requirement for 16 FIFO entries above and below the half-full point. This provides reliable operation with the device-specific transceiver Rx Elastic Buffers. Again, however, check the PHY data sheet to ensure that the PHY device sources the receiver SGMII stream synchronously to its reference oscillator.

Transceiver Logic Using the Rx Elastic Buffer

When the device-specific transceiver Rx Elastic Buffer implementation is selected, the connections between the core and the device-specific transceiver as well as all clock circuitry in the system are identical to the 1000BASE-X implementation. For a detailed explanation, see Chapter 7, "1000BASE-X with Transceivers."

Transceiver Logic with the Fabric Rx Elastic Buffer

The example design delivered with the core is split between two hierarchical layers, as illustrated in Figure 4-3. The block level is designed so to be instantiated directly into customer designs and provides the following functionality:

- Instantiates the core from HDL
- Connects the physical-side interface of the core to a Virtex®-4, Virtex-5, Virtex-6 or Spartan®-6 FPGA transceiver via the fabric Rx Elastic Buffer

The logic implemented in the block level is illustrated in all figures throughout the remainder of this chapter.

Virtex-4 Devices for SGMII or Dynamic Standards Switching

The core is designed to integrate with the Virtex-4 FPGA MGT. The connections and logic required between the core and MGT transceiver are illustrated in Figure 8-3–the signal names and logic in the figure precisely match those delivered with the example design when an MGT transceiver is used.

Note: A small logic shim (included in the "block" level wrapper) is required to convert between the port differences between the Virtex-5 and Virtex-4 FPGA MGTs. This is not illustrated in Figure 8-3.

The MGT clock distribution in Virtex-4 devices is column-based and consists of multiple MGT tiles (that contain two MGTs each). For this reason, the MGT transceiver wrapper delivered with the core always contains two MGT instantiations, even if only a single MGT is in use. Figure 8-3 illustrates only a single MGT for clarity.

A `GT11CLK_MGT` primitive is also instantiated to derive the reference clocks required by the MGT column-based tiles. See the *Virtex-4 FPGA RocketIO Multi-Gigabit Transceiver User Guide* (UG076) for more information about layout and clock distribution.

The 250 MHz reference clock from the `GT11CLK_MGT` primitive is routed to the MGT, which is configured to internally synthesize a 125 MHz clock. This is output on the `TXOUTCLK1` port of the MGT and once placed onto global clock routing, can be used by all core logic. This clock is input back into the MGT on the user interface clock port `txusrclk2`. With the attribute settings applied to the MGT from the example design, the `txusrclk` port is derived internally within the MGT using the internal clock dividers and does not need to be provided from the FPGA fabric.

It can be seen from Figure 8-3 that the Rx Elastic Buffer is implemented in the FPGA fabric between the MGT and the core. This replaces the Rx Elastic Buffer in the MGT (which is bypassed).

This alternative Receiver Elastic Buffer uses a single block RAM to create a buffer twice as large as the one present in the MGT. It is able to cope with larger frame sizes before clock tolerances accumulate and result in emptying or filling of the buffer. This is necessary to guarantee SGMII operation at 10 Mbps where each frame size is effectively 100 times larger than the same frame would be at 1 Gbps because each byte is repeated 100 times (see "Designing with Client-side GMII for the SGMII Standard," page 68).

In bypassing the MGT Rx Elastic Buffer, data is clocked out of the MGT synchronously to `rxrecclk1`. This clock can be placed on a BUFR component and is used to synchronize the transfer of data between the MGT and the Elastic Buffer, as illustrated in Figure 8-3. See also "Virtex-4 FPGA RocketIO MGT Transceivers for SGMII or Dynamic Standards Switching Constraints," page 192.

The MGT transceivers require a calibration block to be included in the fabric logic. The example design provided with the core instantiates calibration blocks as required.

Calibration blocks require a clock source of between 25 to 50 MHz, which is shared with the Dynamic Reconfiguration Port (DRP) of the MGT, named `dclk` in the example design. See Xilinx Answer Record 22477 for more information.

Figure 8-3 also illustrates the `TX_SIGNAL_DETECT` and `RX_SIGNAL_DETECT` ports of the calibration block, which should be driven to indicate whether or not dynamic data is being transmitted and received through the MGT (see Virtex-4 Errata). However, `RX_SIGNAL_DETECT` is connected to the `signal_detect` port of the example design. `signal_detect` is intended to indicate to the core that valid data is being received. When not asserted, the calibration block will switch the MGT into loopback to force dynamic data through the MGT receiver path.

> ***Caution!*** The PHY connected via SGMII may always provide dynamic SGMII data (when powered up). If not, and if `signal_detect` is not present, the `RX_SIGNAL_DETECT` port of the calibration block must be driven by an alternative method. See *XAPP732* for more information.

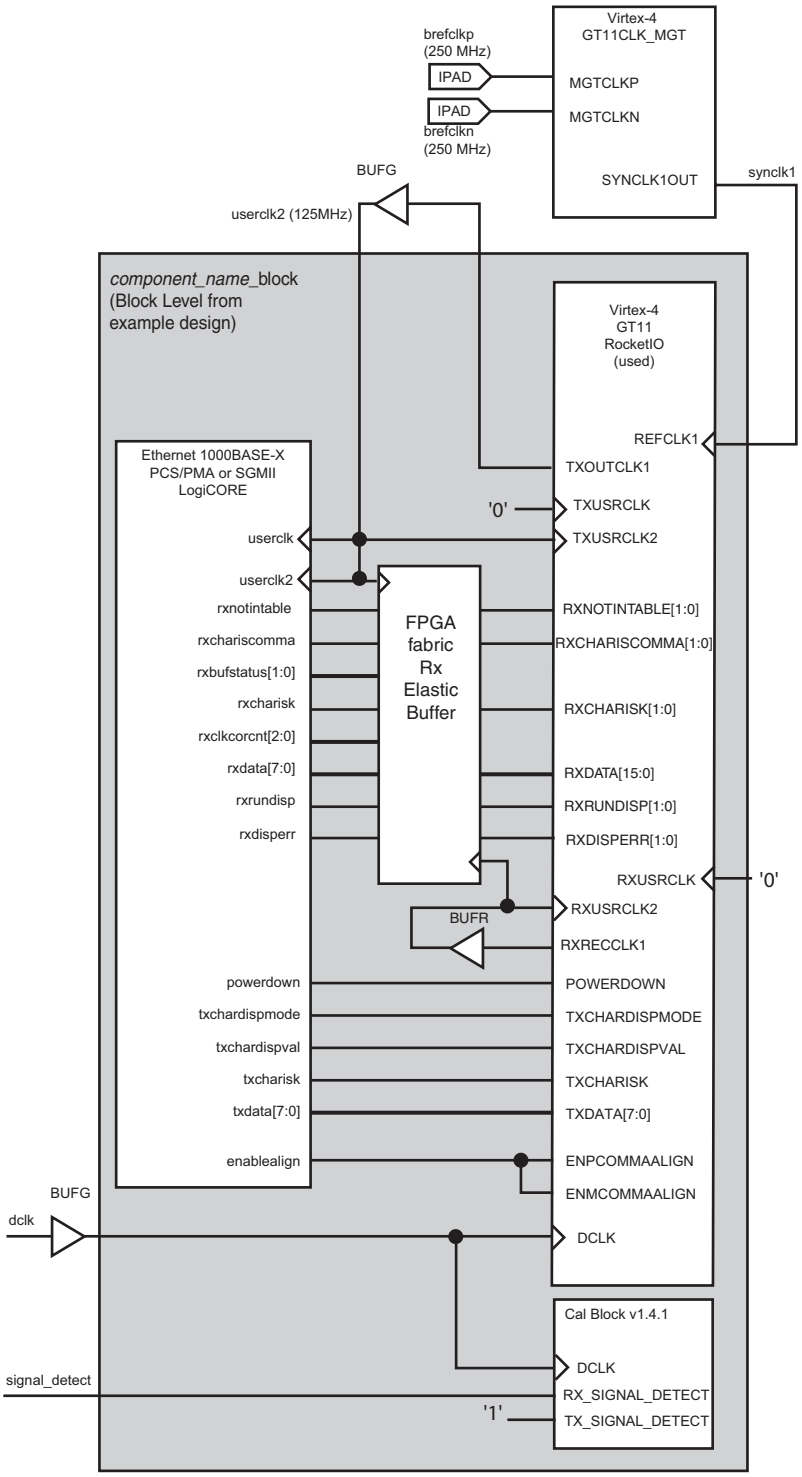


Figure 8-3: **SGMII Connection to a Virtex-4 FPGA Rocket IO MGT**

Virtex-5 LXT or SXT Devices for SGMII or Dynamic Standards Switching

The core is designed to integrate with the Virtex-5 FPGA RocketIO™ GTP transceiver. The connections and logic required between the core and GTP transceiver are illustrated in Figure 8-4–the signal names and logic in the figure precisely match those delivered with the example design when a GTP transceiver is used.

A GTP tile consists of a pair of transceivers. For this reason, the GTP transceiver wrapper delivered with the core will always contain two GTP transceiver instantiations, even if only a single GTP is in use. Figure 8-4 illustrates only a single GTP transceiver for clarity.

The 125 MHz differential reference clock is routed to the GTP transceiver, which is configured to output a version of this clock on the `REFCLKOUT` port, and once placed onto global clock routing can be used by all core logic. This clock is input back into the GTP transceiver on the user interface clock port `txusrclk` and `txusrclk2`.

It can be seen from Figure 8-4 that the Rx Elastic Buffer is implemented in the FPGA fabric between the GTP transceiver and the core; this replaces the Rx Elastic Buffer in the GTP transceiver.

This alternative Receiver Elastic Buffer uses a single block RAM to create a buffer twice as large as the one present in the GTP transceiver. It is able to cope with larger frame sizes before clock tolerances accumulate and result in emptying or filling of the buffer. This is necessary to guarantee SGMII operation at 10 Mbps where each frame size is effectively 100 times larger than the same frame would be at 1 Gbps because each byte is repeated 100 times (see "Designing with Client-side GMII for the SGMII Standard," page 68).

With this fabric Rx Elastic Buffer implementation, data is clocked out of the GTP transceiver synchronously to `rxrecclk0`. This clock can be placed on a BUFR component and is used to synchronize the transfer of data between the GTP and the Elastic Buffer, as illustrated in Figure 8-4. See also "Virtex-5 FPGA RocketIO GTP Transceivers for SGMII or Dynamic Standards Switching Constraints," page 193.

Virtex-5 FPGA RocketIO Transceiver GTP Wizard

The two wrapper files immediately around the GTP transceiver pair, `RocketIO_wrapper_gtp_tile` and `RocketIO_wrapper_gtp` (see Figure 8-4), are generated from the *RocketIO GTP Wizard*. These files apply all the gigabit Ethernet attributes. Consequently, these files can be regenerated by customers and therefore be easily targeted at ES or Production silicon. This core targets production silicon.

The CORE Generator™ software log file (XCO file) which was created when the *RocketIO GTP Wizard* project was generated is available in the following location:

```
<project_directory>/<component_name>/example_design/transceiver/
RocketIO_wrapper_gtp.xco
```

This file can be used as an input to the CORE Generator software to regenerate the device-specific RocketIO transceiver wrapper files. The XCO file itself contains a list of all of the GTP Wizard attributes which were used. For further information, please see the *Virtex-5 RocketIO GTP Wizard Getting Started Guide* (UG188) and the *CORE Generator Guide*, at www.xilinx.com/support/software_manuals.htm

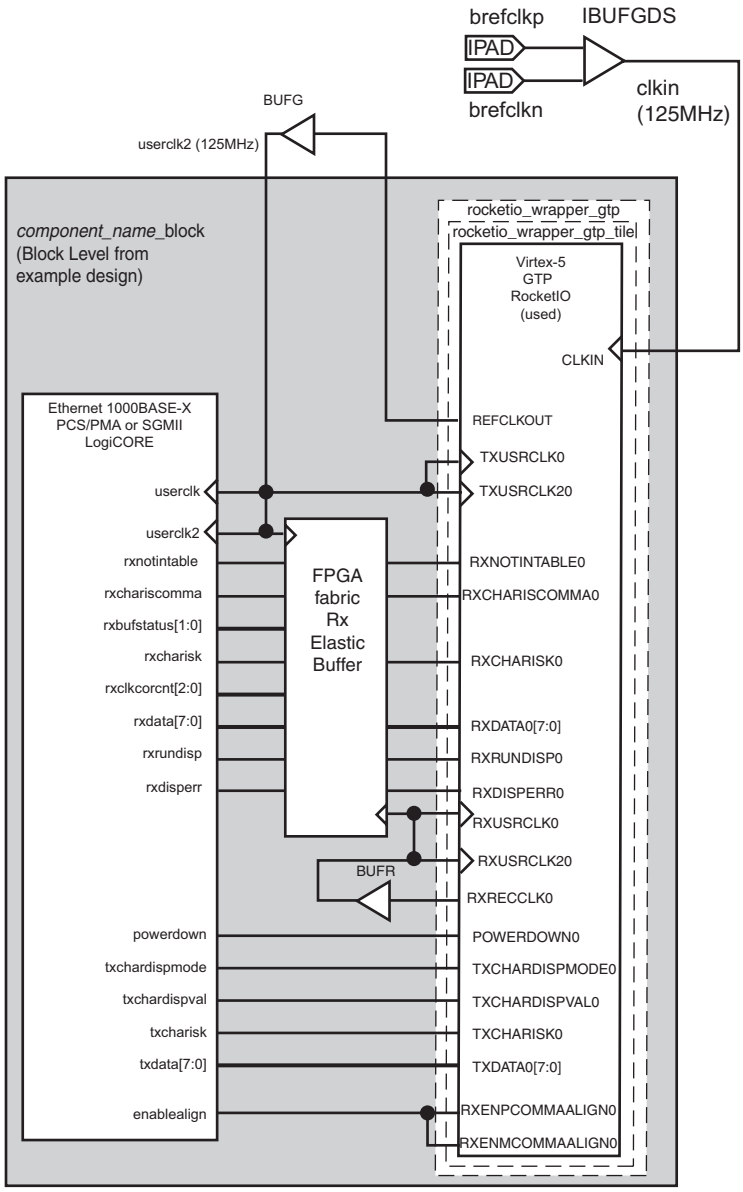


Figure 8-4: **SGMII Connection to a Virtex-5 FPGA RocketIO GTP Transceiver**

Virtex-5 FXT and TXT Devices for SGMII or Dynamic Standards Switching

The core is designed to integrate with the Virtex-5 FPGA RocketIO GTX transceiver. The connections and logic required between the core and GTX transceiver are illustrated in Figure 8-5–the signal names and logic in the figure precisely match those delivered with the example design when a GTX transceiver is used.

A GTX tile consists of a pair of transceivers. For this reason, the GTX transceiver wrapper delivered with the core will always contain two GTX transceiver instantiations, even if only a single GTX is in use. Figure 8-5 illustrates only a single GTX transceiver for clarity.

The 125 MHz differential reference clock is routed directly to the GTX transceiver. The GTX transceiver is configured to output a version of this clock on the `REFCLKOUT` port; this is then routed to a DCM via a BUFG (global clock routing).

From the DCM, the `CLK0` port (125 MHz) is placed onto global clock routing and can be used as the 125 MHz clock source for all core logic; this clock is also input back into the GTX transceiver on the user interface clock port `txusrclk2`.

From the DCM, the `CLKDV` port (62.5 MHz) is placed onto global clock routing and is input back into the GTX transceiver on the user interface clock port `txusrclk`.

It can be seen from Figure 8-5 that the Rx Elastic Buffer is implemented in the FPGA fabric between the GTX transceiver and the core; this replaces the Rx Elastic Buffer in the GTX transceiver.

This alternative Receiver Elastic Buffer uses a single block RAM to create a buffer twice as large as the one present in the GTX transceiver. It is able to cope with larger frame sizes before clock tolerances accumulate and result in emptying or filling of the buffer. This is necessary to guarantee SGMII operation at 10 Mbps where each frame size is effectively 100 times larger than the same frame would be at 1 Gbps because each byte is repeated 100 times (see "Designing with Client-side GMII for the SGMII Standard," page 68).

With this fabric Rx Elastic Buffer implementation, data is clocked out of the GTX transceiver synchronously to `rxrecclk0` (62.5 MHz) on a 16-bit interface. This clock can be placed on a BUFR component and is used to synchronize the transfer of data between the GTX and the Elastic Buffer, as illustrated in Figure 8-5. See also "Virtex-5 FPGA RocketIO GTX Transceivers for SGMII or Dynamic Standards Switching Constraints," page 194.

Virtex-5 FPGA RocketIO GTX Wizard

The two wrapper files immediately around the GTX transceiver pair, `RocketIO_wrapper_gtx_tile` and `RocketIO_wrapper_gtx` (see Figure 8-5), are generated from the *RocketIO GTP Wizard*. These files apply all the gigabit Ethernet attributes. Consequently, these files can be regenerated by customers and therefore be easily targeted at ES or Production silicon. This core targets production silicon.

The CORE Generator software log file (XCO file) which was created when the *RocketIO GTX Wizard* project was generated is available in the following location:

```
<project_directory>/<component_name>/example_design/transceiver/
RocketIO_wrapper_gtx.xco
```

This file can be used as an input to the CORE Generator software to regenerate the device-specific RocketIO transceiver wrapper files. The XCO file itself contains a list of all of the GTX Wizard attributes which were used. For further information, please see the *Virtex-5 FPGA RocketIO GTX Wizard Getting Started Guide* and the *CORE Generator Guide*, at www.xilinx.com/support/software_manuals.htm

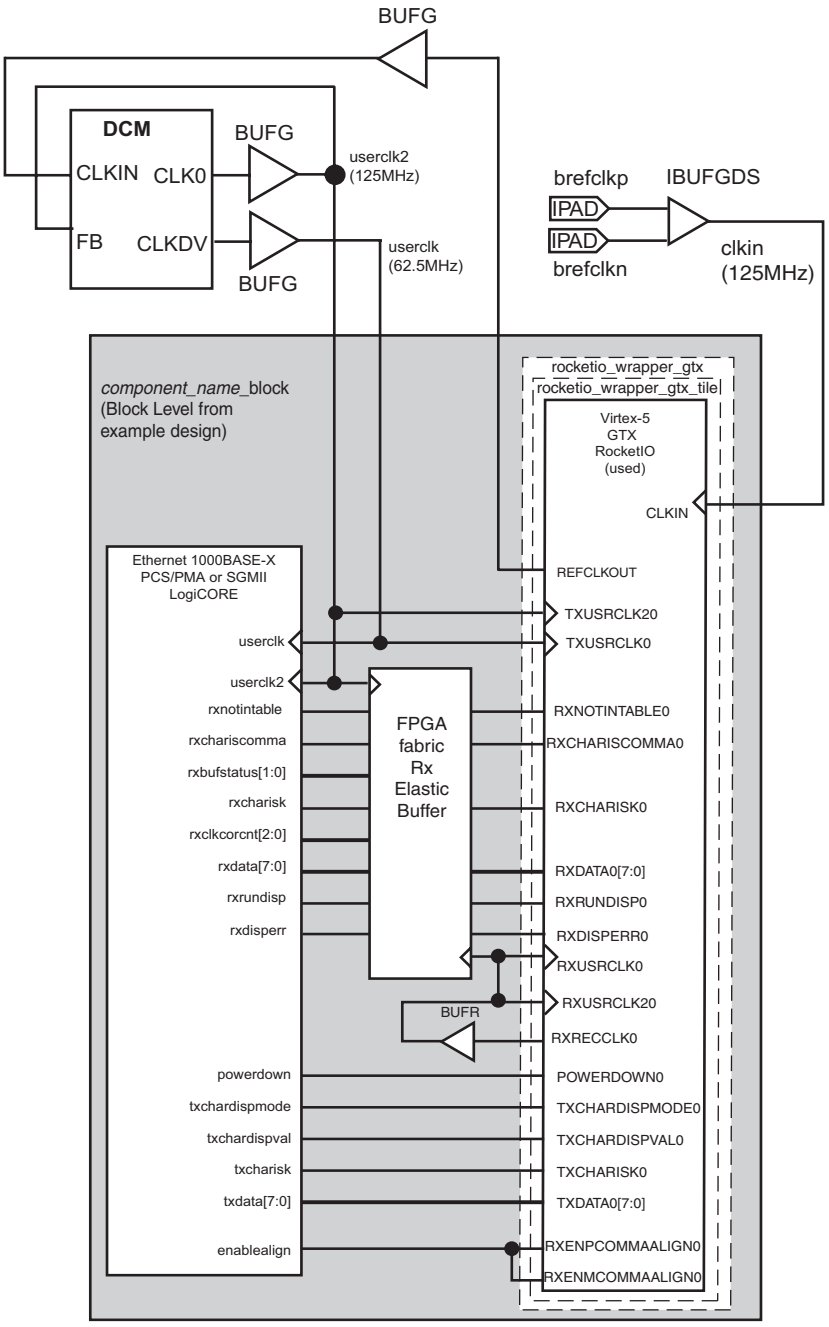


Figure 8-5: **SGMII Connection to a Virtex-5 FPGA RocketIO GTX Transceiver**

Virtex-6 Devices for SGMII or Dynamic Standards Switching

The core is designed to integrate with the Virtex-6 FPGA GTX transceiver. The connections and logic required between the core and GTP transceiver are illustrated in Figure 8-6–the signal names and logic in the figure precisely match those delivered with the example design when a Virtex-6 FPGA GTX transceiver is used.

The 125 MHz differential reference clock is routed to the GTX transceiver, which is configured to output a version of this clock on the `TXOUTCLK` port, and once placed onto global clock routing can be used by all core logic. This clock is input back into the GTX transceiver on the user interface clock port `txusrclk2`. The `txusrclk` clock signal will be derived internally in the GTX and so can be connected to ground.

It can be seen from Figure 8-6 that the Rx Elastic Buffer is implemented in the FPGA fabric between the GTX transceiver and the core; this replaces the Rx Elastic Buffer in the GTX transceiver.

This alternative Receiver Elastic Buffer uses a single block RAM to create a buffer twice as large as the one present in the GTX transceiver. It is able to cope with larger frame sizes before clock tolerances accumulate and result in emptying or filling of the buffer. This is necessary to guarantee SGMII operation at 10 Mbps where each frame size is effectively 100 times larger than the same frame would be at 1 Gbps because each byte is repeated 100 times (see "Designing with Client-side GMII for the SGMII Standard," page 68).

With this fabric Rx Elastic Buffer implementation, data is clocked out of the GTX transceiver synchronously to `RXRECCLK`. This clock can be placed on a BUFR component and is used to synchronize the transfer of data between the GTX and the Elastic Buffer, as illustrated in Figure 8-6. See also "Virtex-6 FPGA GTX Transceivers for SGMII or Dynamic Standards Switching Constraints," page 195.

Virtex-6 FPGA GTX Transceiver Wizard

The two wrapper files immediately around the GTX transceiver, `gtx_wrapper_gtx` and `gtx_wrapper` (see Figure 8-6), are generated from the *Virtex-6 FPGA GTX Transceiver Wizard*. These files apply all the gigabit Ethernet attributes. Consequently, these files can be regenerated by customers and therefore be easily targeted at silicon/device versions.

The CORE Generator software log file (XCO file) which was created when the *Virtex-6 FPGA GTX Transceiver Wizard* project was generated is available in the following location:

```
<project_directory>/<component_name>/example_design/transceiver/
gtx_wrapper_gtx.xco
```

This file can be used as an input to the CORE Generator software to regenerate the device-specific transceiver wrapper files. The XCO file itself contains a list of all of the Wizard attributes which were used. For further information, please see the *Virtex-6 FPGA GTX Transceiver Wizard Getting Started Guide* and the *CORE Generator Guide*, at www.xilinx.com/support/software_manuals.htm

.

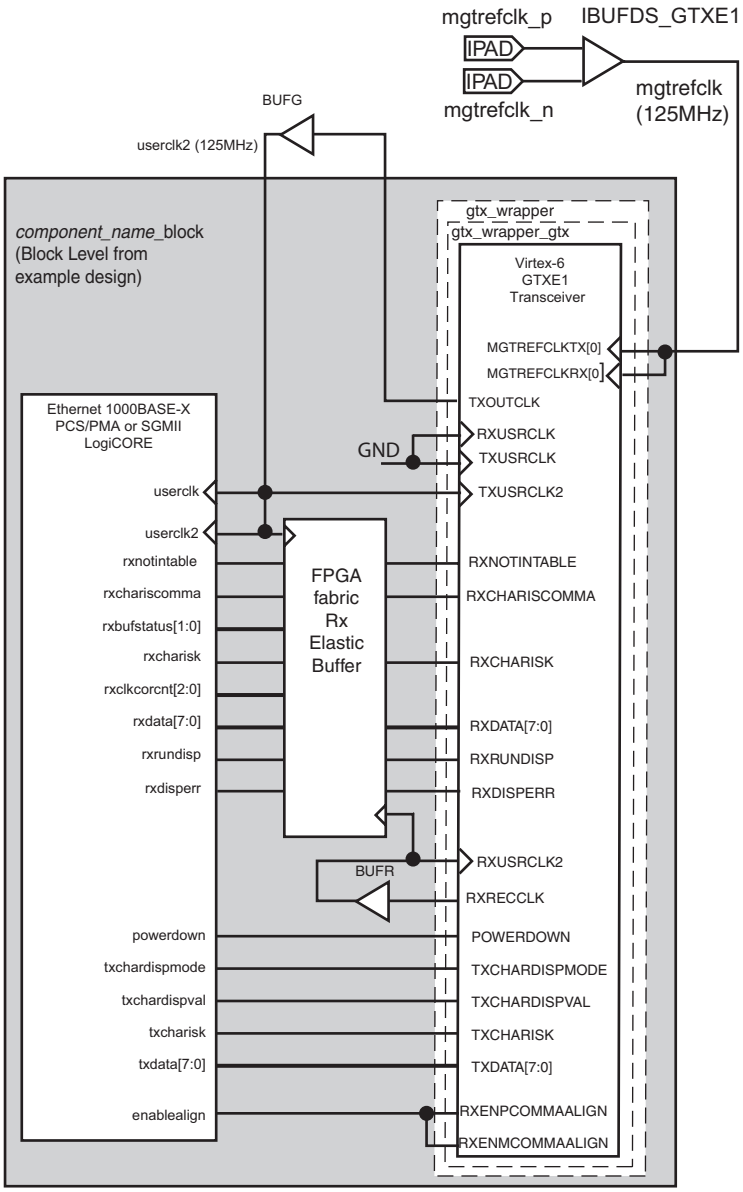


Figure 8-6: **SGMII Connection to a Virtex-6 FPGA GTX Transceiver**

Spartan-6 LXT Devices for SGMII or Dynamic Standards Switching

The core is designed to integrate with the Spartan-6 FPGA GTP transceiver. The connections and logic required between the core and GTP transceiver are illustrated in Figure 8-7. The signal names and logic in the figure precisely match those delivered with the example design when a GTP transceiver is used.

A GTP tile consists of a pair of transceivers. For this reason, the GTP transceiver wrapper delivered with the core will always contain two GTP transceiver instantiations, even if only a single GTP is in use. Figure 8-7 illustrates only a single GTP transceiver for clarity.

The 125 MHz differential reference clock is routed to the GTP transceiver, which is configured to output a version of this clock on the `GTPCLKOUT` port, then routed through a BUFIO2 and BUFG to place onto global clock routing where it can be used by all core logic. This clock is input back into the GTP transceiver on the user interface clock port `txusrclk` and `txusrclk2`.

It can be seen from Figure 8-7 that the Rx Elastic Buffer is implemented in the FPGA fabric between the GTP transceiver and the core; this replaces the Rx Elastic Buffer in the GTP transceiver.

This alternative Receiver Elastic Buffer uses a single block RAM to create a buffer twice as large as the one present in the GTP transceiver. It is able to cope with larger frame sizes before clock tolerances accumulate and result in emptying or filling of the buffer. This is necessary to guarantee SGMII operation at 10 Mbps where each frame size is effectively 100 times larger than the same frame would be at 1 Gbps because each byte is repeated 100 times (see "Designing with Client-side GMII for the SGMII Standard," page 68).

With this fabric Rx Elastic Buffer implementation, data is clocked out of the GTP transceiver synchronously to `rxrecclk0`. This clock can be placed on a BUFG component and is used to synchronize the transfer of data between the GTP and the Elastic Buffer, as illustrated in Figure 8-4. See also "Spartan-6 FPGA GTP Transceivers for SGMII or Dynamic Standards Switching Constraints," page 197.

Spartan-6 FPGA Transceiver GTP Wizard

The two wrapper files immediately around the GTP transceiver pair, `gtp_wrapper_tile` and `gtp_wrapper` (see Figure 8-7), are generated from the *GTP Wizard*. These files apply all the gigabit Ethernet attributes. Consequently, these files can be regenerated by customers and therefore be easily targeted at ES or Production silicon. This core targets production silicon.

The CORE Generator software log file (XCO file) which was created when the *GTP Wizard* project was generated is available in the following location:

```
<project_directory>/<component_name>/example_design/transceiver/
gtp_wrapper.xco
```

This file can be used as an input to the CORE Generator software to regenerate the device-specific transceiver wrapper files. The XCO file itself contains a list of all of the GTP Wizard attributes which were used. For further information, please see the *Spartan-6 GTP Wizard Getting Started Guide* and the *CORE Generator Guide*, at www.xilinx.com/support/software_manuals.htm

.

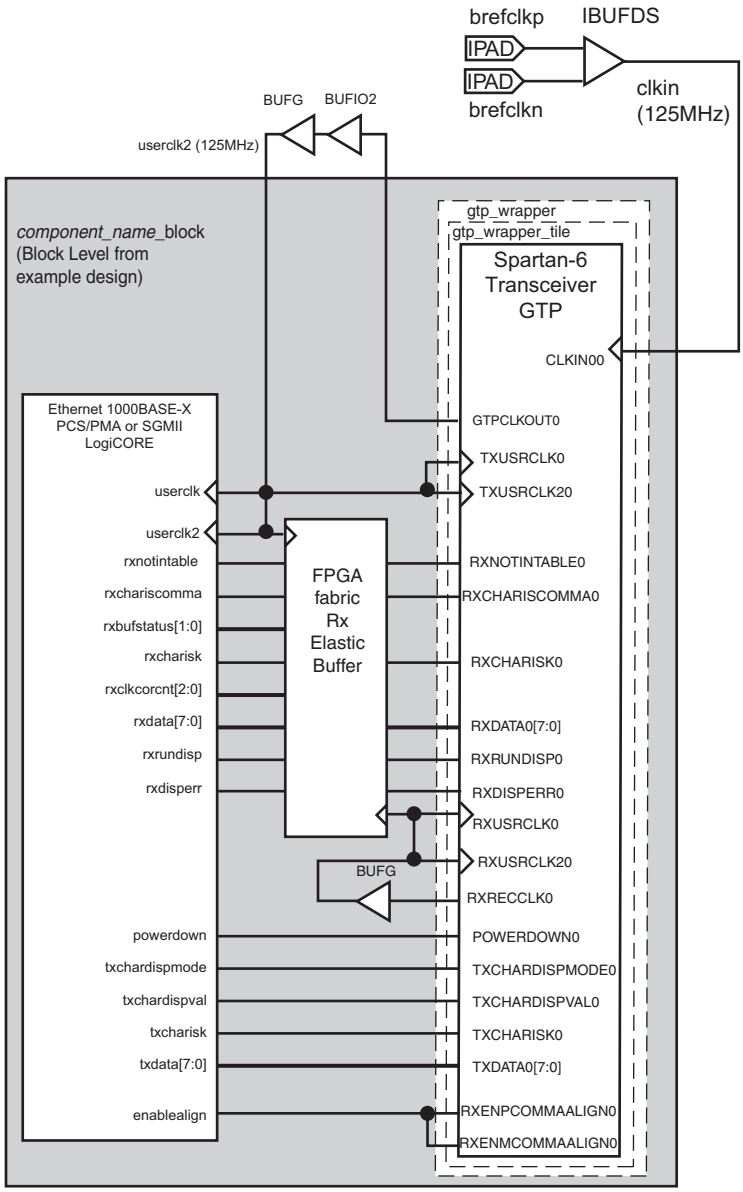


Figure 8-7: **SGMII Connection to a Spartan-6 FPGA GTP Transceiver**

Clock Sharing - Multiple Cores with Transceivers, Fabric Elastic Buffer

Virtex-4 FX Devices

Figure 8-8 illustrates sharing clock resources across multiple instantiations of the core when using the Virtex-4 FPGA RocketIO MGT transceiver. The example design, when using the Virtex-4 family, can be generated to connect either a single instance of the core, or connect a pair of core instances to the transceiver pair present in a MGT tile. Figure 8-8 illustrates two instantiations of the block level, and each block level contains a pair of cores, illustrating clock sharing between four cores.

More cores can be added by continuing to instantiate extra block level modules. Share clocks only between the MGTs in a single column. For each column, use a single `brefclk_p` and `brefclk_n` differential clock pair and connect this to a `GT11CLK_MGT` primitive. The clock output from this primitive should be shared across all used MGT tiles in the column. See the *Virtex-4 RocketIO Multi-Gigabit Transceiver User Guide* for more information.

To provide the 125 MHz clock for all core instances, select a `TXOUTCLK1` port from any MGT. This can be routed onto global clock routing using a BUFG as illustrated, and shared between all cores and MGTs in the column.

Each MGT and core pair instantiated has its own independent clock domain synchronous to `RXRECCLK1` which is placed on regional clock routing using a BUFR, as illustrated in Figure 8-8–these cannot be shared across multiple MGTs. Although not illustrated in Figure 8-8, `dclk` (the clock used for the calibration blocks and for the Dynamic Reconfiguration Port (DRP) of the MGTs) can also be shared.

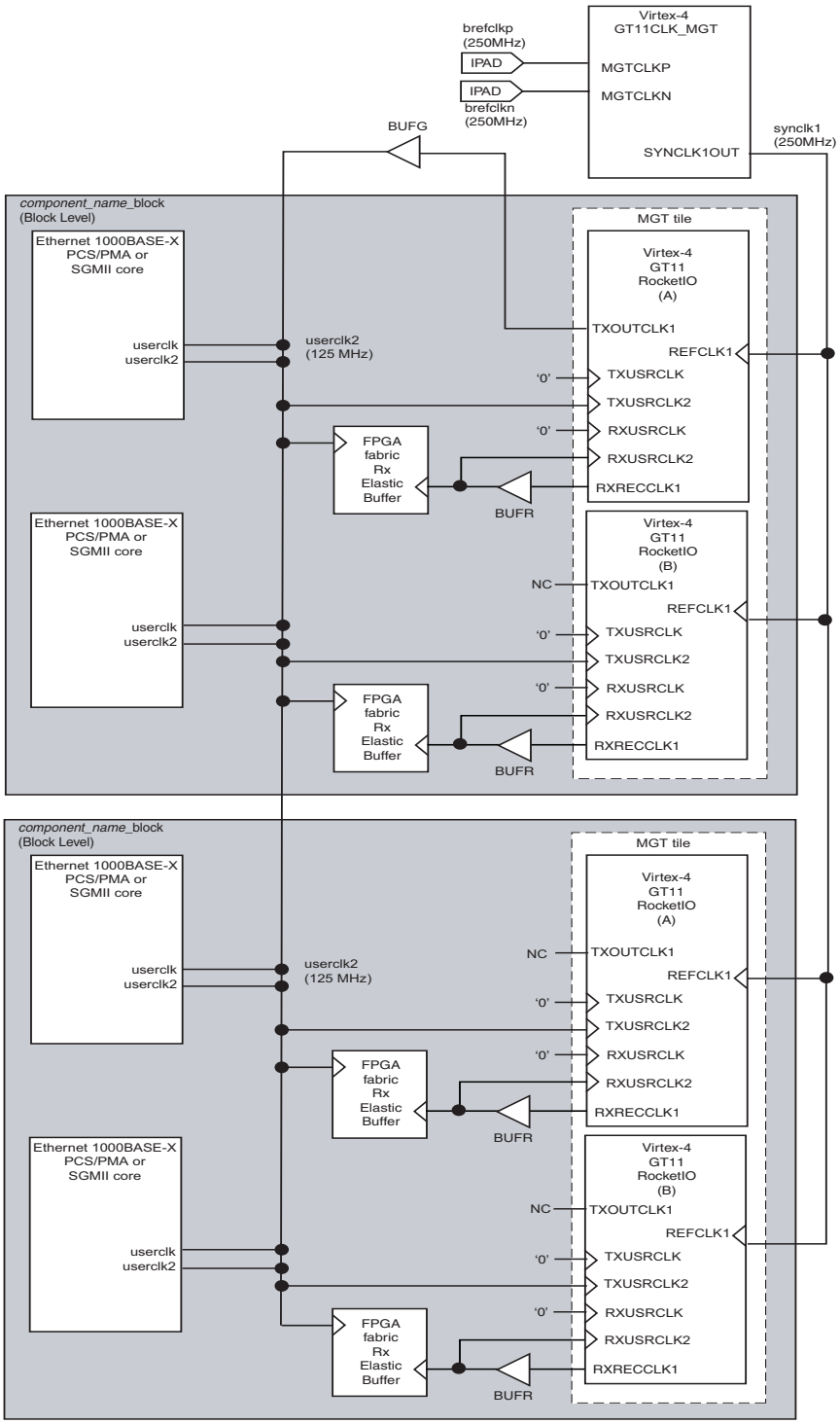


Figure 8-8: **Clock Management with Multiple Core Instances with Virtex-4 FPGA MGTs for SGMII**

Virtex-5 LXT and SXT Devices

Figure 8-9 illustrates sharing clock resources across multiple instantiations of the core when using the Virtex-5 FPGA RocketIO GTP transceiver. The example design can be generated to connect either a single instance of the core, or connect a pair of core instances to the transceiver pair present in a GTP transceiver tile. Figure 8-9 illustrates two instantiations of the block level, and each block level contains a pair of cores. Figure 8-9 illustrates clock sharing between four cores.

More cores can be added by instantiating extra block level modules. Share the `brefclk_p` and `brefclk_n` differential clock pairs. See the *Virtex-5 RocketIO GTP Transceiver User Guide* for more information.

To provide the 125 MHz clock for all core instances, select a `REFCLKOUT` port from any GTP transceiver. This can be routed onto global clock routing using a BUFG as illustrated and shared between all cores and GTP transceivers in the column.

Each GTP and core pair instantiated has its own independent clock domains synchronous to `RXRECCLK0` and `RXRECCLK1`. These are placed on regional clock routing using a BUFR, as illustrated in Figure 8-9, and cannot be shared across multiple GTP transceivers.

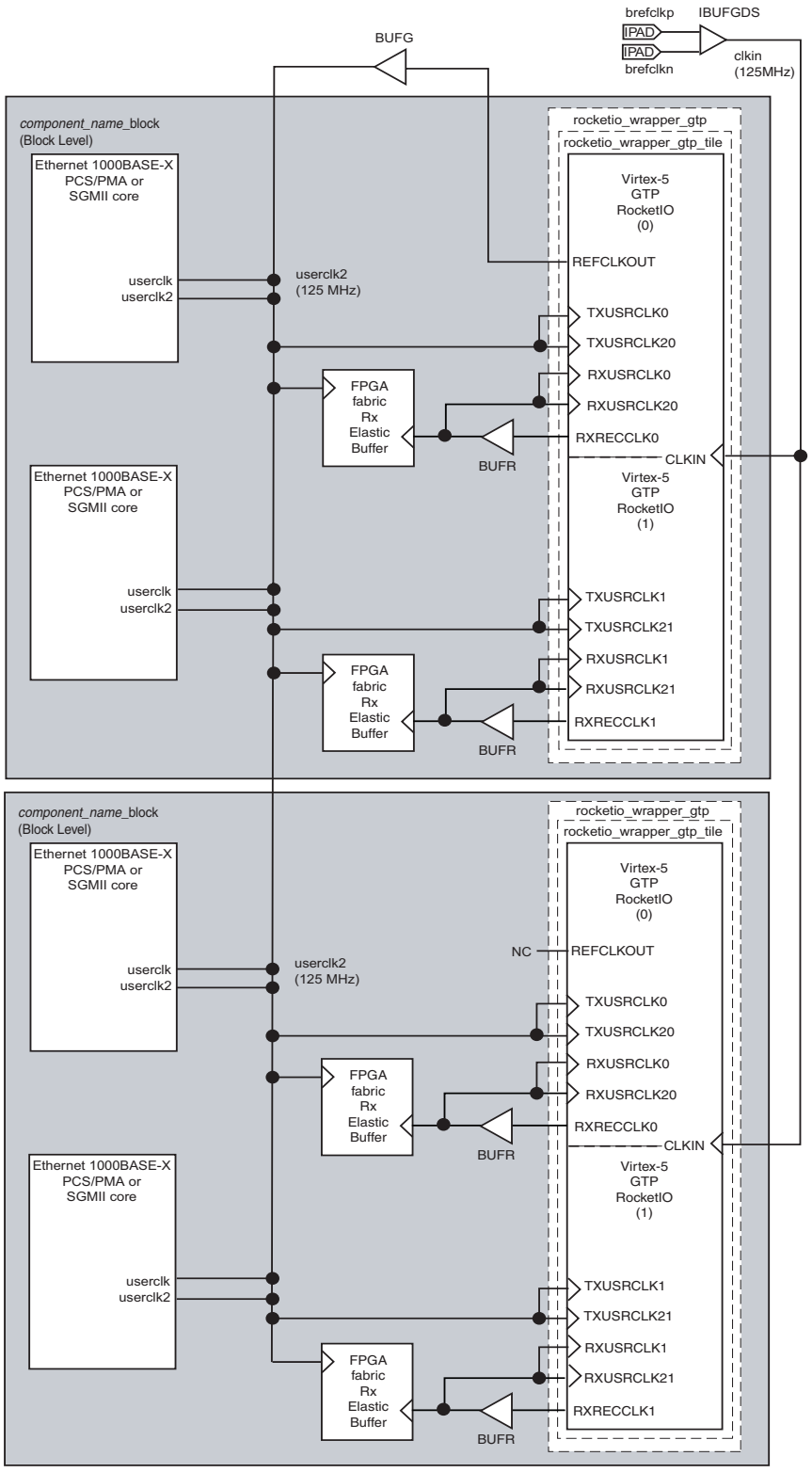


Figure 8-9: **Clock Management with Multiple Core Instances with Virtex-5 FPGA RocketIO GTP Transceivers for SGMII**

Virtex-5 FXT and TXT Devices

Figure 8-10 illustrates sharing clock resources across multiple instantiations of the core when using the Virtex-5 FPGA RocketIO GTX transceiver. The example design can be generated to connect either a single instance of the core, or connect a pair of core instances to the transceiver pair present in a GTX transceiver tile. Figure 8-10 illustrates two instantiations of the block level, and each block level contains a pair of cores. Figure 8-10 illustrates clock sharing between four cores.

More cores can be added by instantiating extra block level modules. Share the `brefclk_p` and `brefclk_n` differential clock pairs. See the *Virtex-5 RocketIO GTX Transceiver User Guide* for more information.

To provide the FPGA fabric clocks for all core instances, select a `REFCLKOUT` port from any GTX transceiver and route this to a single DCM via a BUFG (global clock routing). The `CLK0` (125 MHz) and `CLKDV` (62.5 MHz) outputs from this DCM, placed onto global clock routing using BUFGs, can be shared across all core instances and GTX transceivers as illustrated.

Each GTX and core pair instantiated has its own independent clock domains synchronous to `RXRECCLK0` and `RXRECCLK1`. These are placed on regional clock routing using a BUFR, as illustrated in Figure 8-10, and cannot be shared across multiple GTX transceivers.

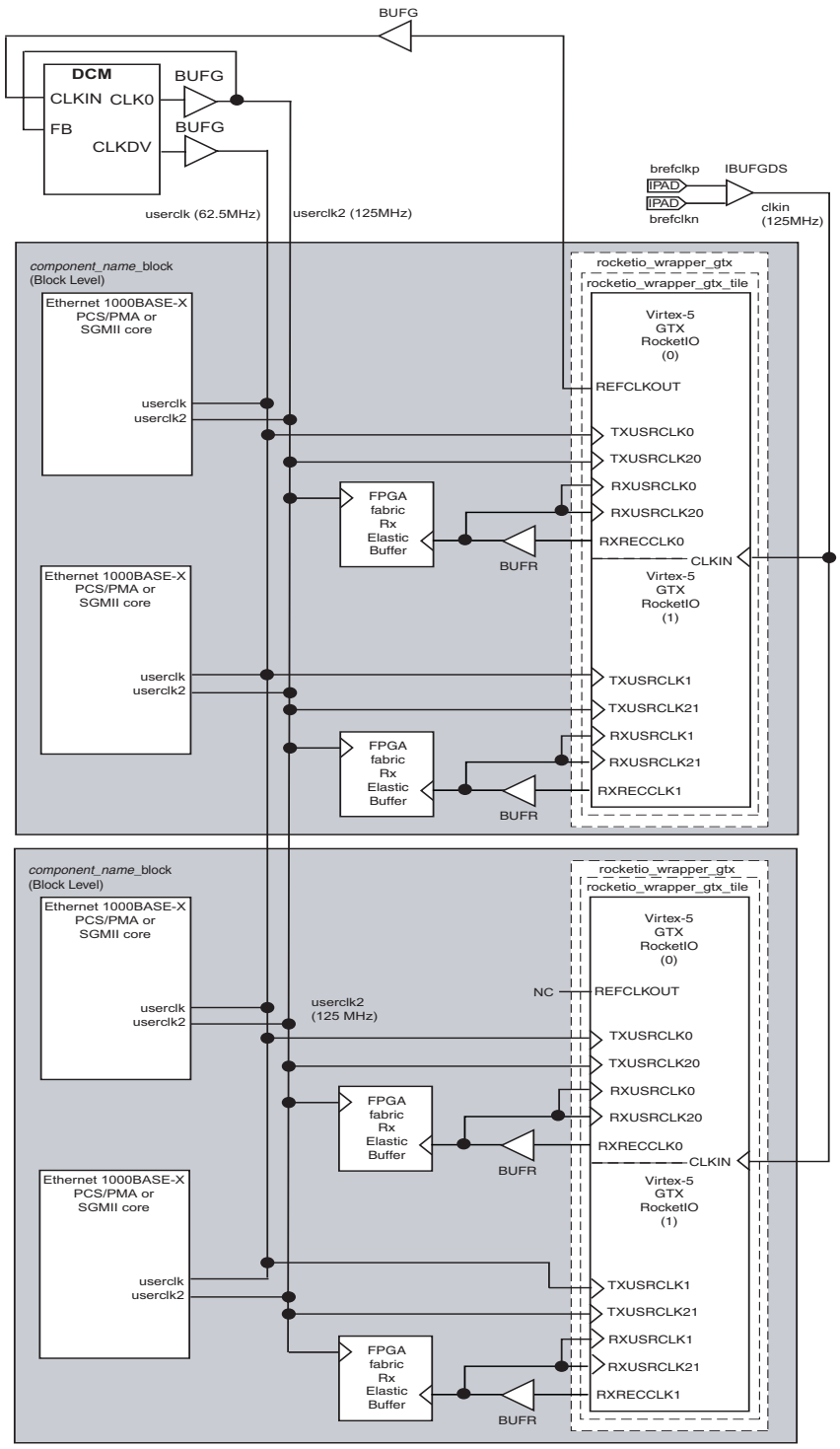


Figure 8-10: **Clock Management with Multiple Core Instances with Virtex-5 FPGA RocketIO GTX Transceivers for SGMII**

Virtex-6 Devices

Figure 8-11 illustrates sharing clock resources across two instantiations of the core when using the Virtex-6 FPGA GTX transceivers. Further cores can be added by instantiating extra block level modules.

Share the `mgtrefclk_p` and `mgtrefclk_n` differential clock pair clock source across all of the transceivers in use. To provide the 125 MHz clock for all core instances, select a `TXOUTCLK` port from any GTX transceiver. This can be routed onto global clock routing using a BUFG as illustrated and shared between all cores and GTX transceivers.

Each GTX and core pair instantiated has its own independent clock domains synchronous to `RXRECCLK`. These are placed on regional clock routing using a BUFR, as illustrated in Figure 8-11, and cannot be shared across multiple GTX transceivers.

See the *Virtex-6 FPGA GTX Transceiver User Guide* for more information on GTX clock resources.

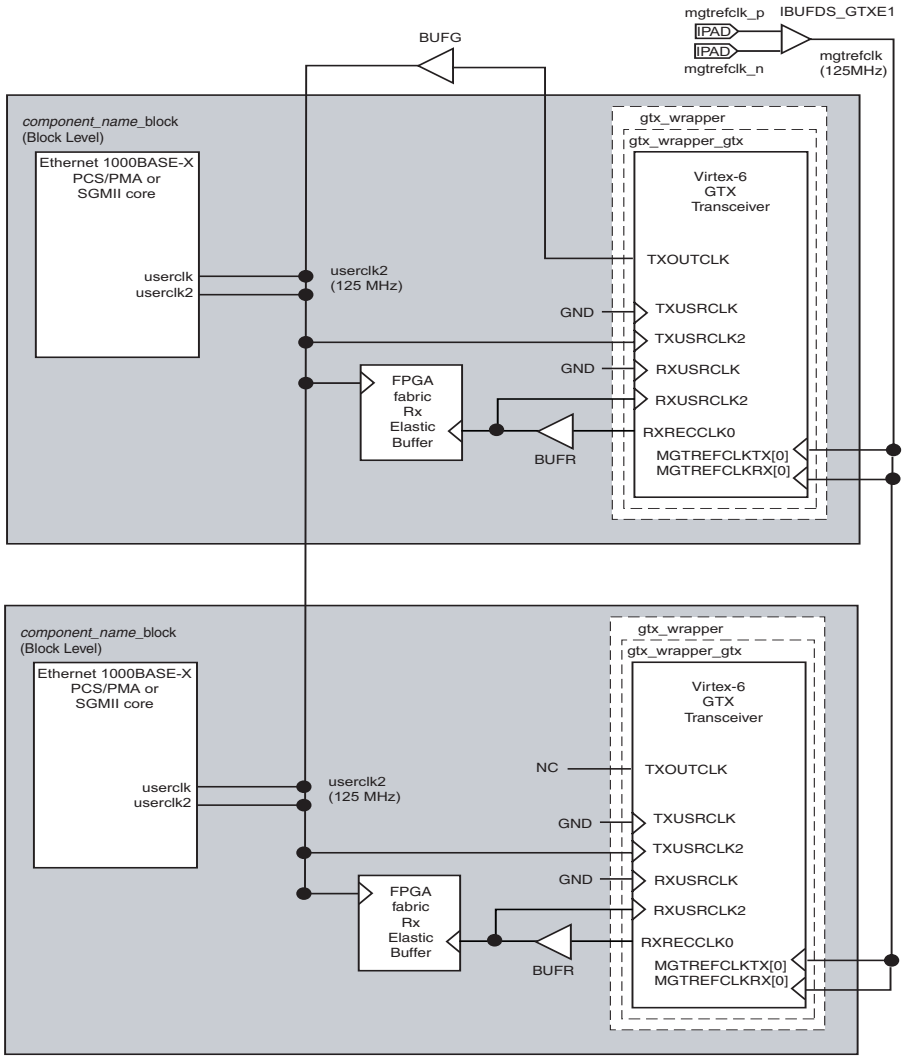


Figure 8-11: **Clock Management with Multiple Core Instances with Virtex-6 FPGA GTX Transceivers for SGMII**

Spartan-6 LXT Devices

Figure 8-12 illustrates sharing clock resources across multiple instantiations of the core when using the Spartan-6 FPGA GTP transceiver. The example design can be generated to connect either a single instance of the core, or connect a pair of core instances to the transceiver pair present in a GTP transceiver tile. Figure 8-12 illustrates two instantiations of the block level, and each block level contains a pair of cores. Figure 8-12 illustrates clock sharing between four cores.

More cores can be added by instantiating extra block level modules. Share the `brefclk_p` and `brefclk_n` differential clock pairs. See the *Spartan-6 FPGA GTP Transceiver User Guide* for more information.

To provide the 125 MHz clock for all core instances, select a GTP`CLKOUT` port from any GTP transceiver. This can be routed onto global clock routing using a BUFIO2 and BUFG as illustrated and shared between all cores and GTP transceivers in the column.

Each GTP and core pair instantiated has its own independent clock domains synchronous to `RXRECCLK0` and `RXRECCLK1`. These are placed on global clock routing using a BUFG, as illustrated in Figure 8-12, and cannot be shared across multiple GTP transceivers.

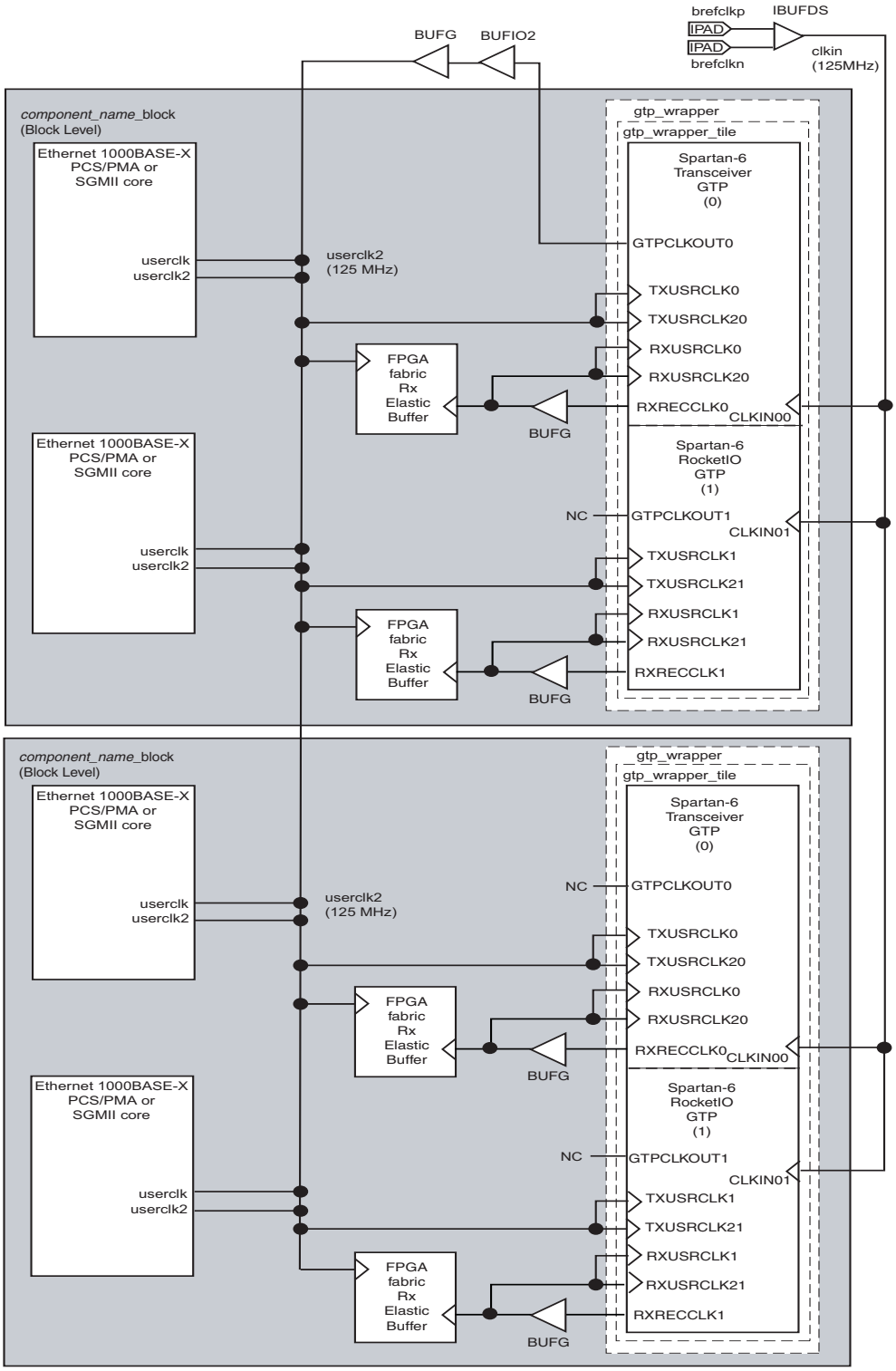


Figure 8-12: **Clock Management with Multiple Core Instances with Spartan-6 FPGA GTP Transceivers for SGMII**

XILINX®

Chapter 9

Configuration and Status

This chapter provides general guidelines for configuring and monitoring the Ethernet 1000BASE-X PCS/PMA or SGMII core, including a detailed description of the core management registers. It also describes Configuration Vector and status signals, an alternative to using the optional MDIO Management Interface.

MDIO Management Interface

When the optional MDIO Management Interface is selected, configuration and status of the core is achieved by the Management Registers accessed through the serial Management Data Input/Output Interface (MDIO). See "MDIO Management Interface" in Chapter 3 for more information.

MDIO Bus System

The MDIO interface for 1 Gbps operation (and slower speeds) is defined in *IEEE 802.3*, clause 22. This two-wire interface consists of a clock (MDC) and a shared serial data line (MDIO). The maximum permitted frequency of MDC is set at 2.5 MHz.

Figure 9-1 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).

.

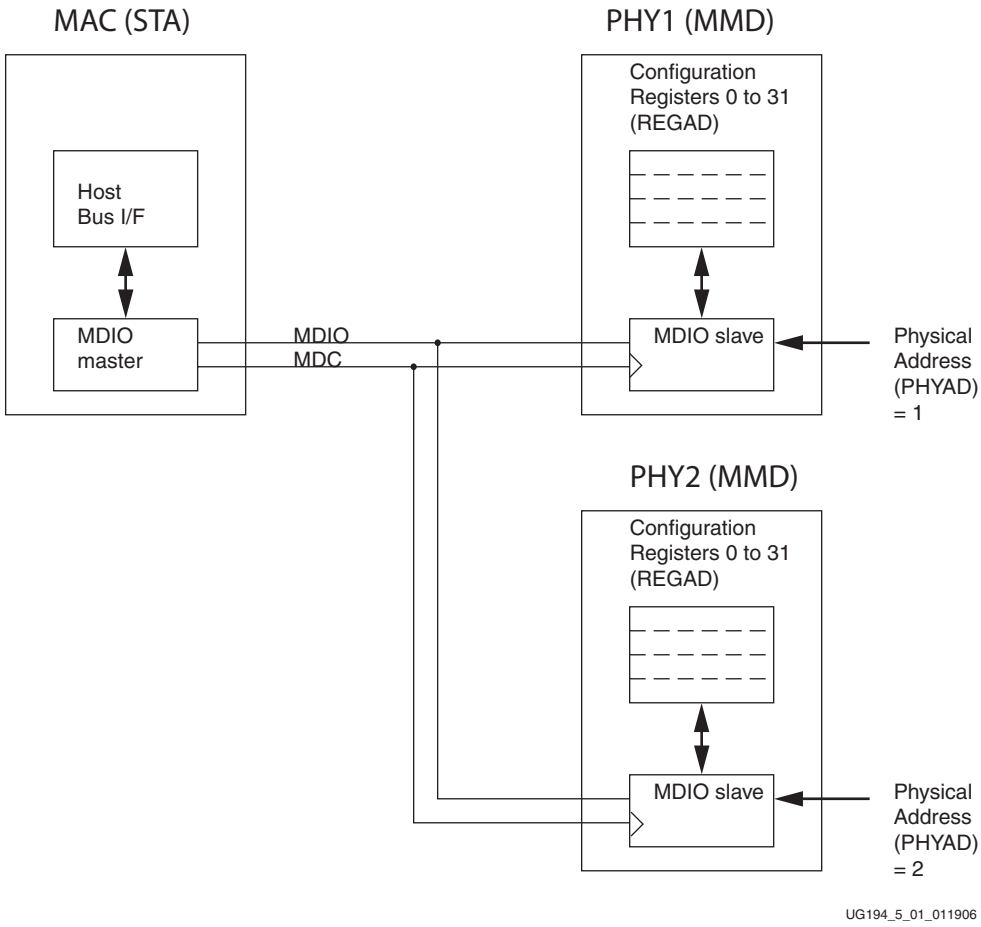


Figure 9-1: **A Typical MDIO-Managed System**

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 Host Bus I/F of the Ethernet MAC is able to access the configuration and status registers of two PHY devices via 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 9-1.

Table 9-1: **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 9-2 shows a write transaction across the MDIO, defined as OP="01." The addressed PHY device (with physical address PHYAD) takes the 16-bit word in the Data field and writes it to the register at REGAD.

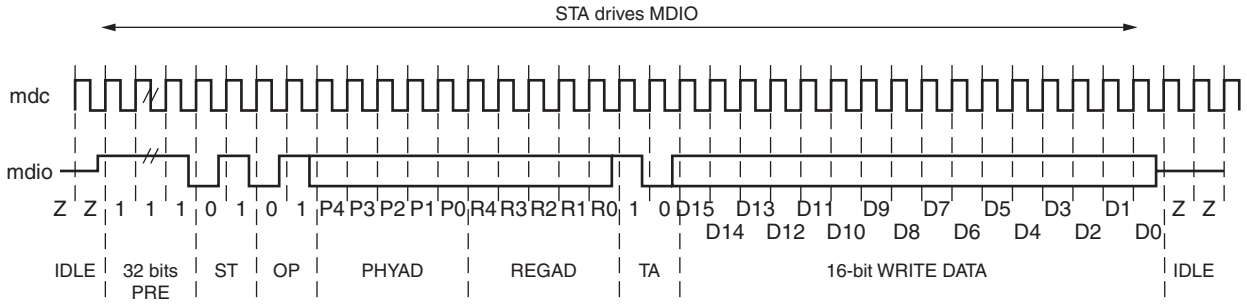


Figure 9-2: **MDIO Write Transaction**

Read Transaction

Figure 9-3 shows a read transaction, defined as OP="10." The addressed PHY device (with physical address PHYAD) takes control of the MDIO wire during the turnaround cycle and then returns the 16-bit word from the register at REGAD.

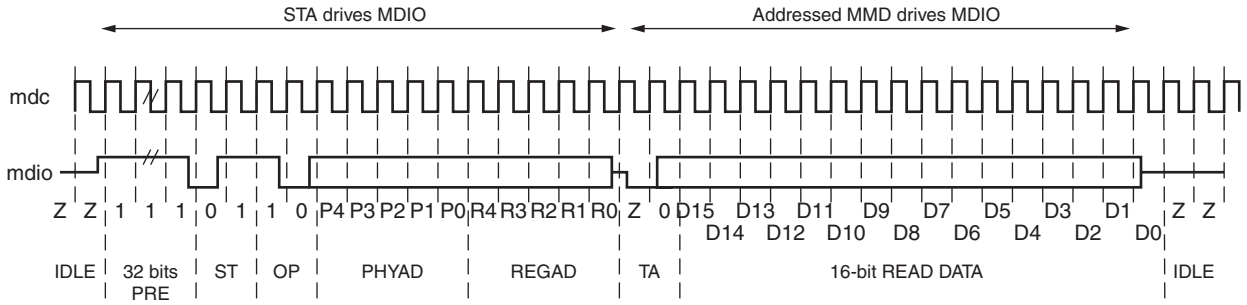


Figure 9-3: **MDIO Read Transaction**

MDIO Addressing

MDIO Addresses consists of two stages: Physical Address (PHYAD) and Register Address (REGAD).

Physical Address (PHYAD)

As shown in Figure 9-1, two PHY devices are attached to the MDIO bus. Each of these has a different physical address. To address the intended PHY, its physical address should be known by the MDIO master (in this case an Ethernet MAC) and placed into the PHYAD field of the MDIO frame (see "MDIO Transactions").

The PHYAD field for an MDIO frame is a 5-bit binary value capable of addressing 32 unique addresses. However, every MDIO slave must respond to physical address 0. This requirement dictates that the physical address for any particular PHY must not be set to 0 to avoid MDIO contention. Physical Addresses 1 through to 31 can be used to connect up to 31 PHY devices onto a single MDIO bus.

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

Register Address (REGAD)

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

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

For details of the register map of PHY layer devices and a more extensive description of the operation of the MDIO Interface, see *IEEE 802.3-2002*.

Connecting the MDIO to an Internally Integrated STA

The MDIO ports of the Ethernet 1000BASE-X PCS/PMA or SGMII core can be connected to the MDIO ports of an internally integrated Station Management (STA) entity, such as the MDIO port of the Tri-Mode Ethernet MAC core (see Chapter 13, "Interfacing to Other Cores")

Connecting the MDIO to an External STA

Figure 9-4 shows the MDIO ports of the Ethernet 1000BASE-X PCS/PMA or SGMII core connected to the MDIO of an external STA entity. In this situation, `mdio_in`, `mdio_out`, and `mdio_tri` must be connected to a Tri-State buffer to create a bidirectional wire, `mdio`. This Tri-State buffer can either be external to the FPGA or internally integrated by using an IOB `IOBUF` component with an appropriate SelectIO™ interface standard suitable for the external PHY.

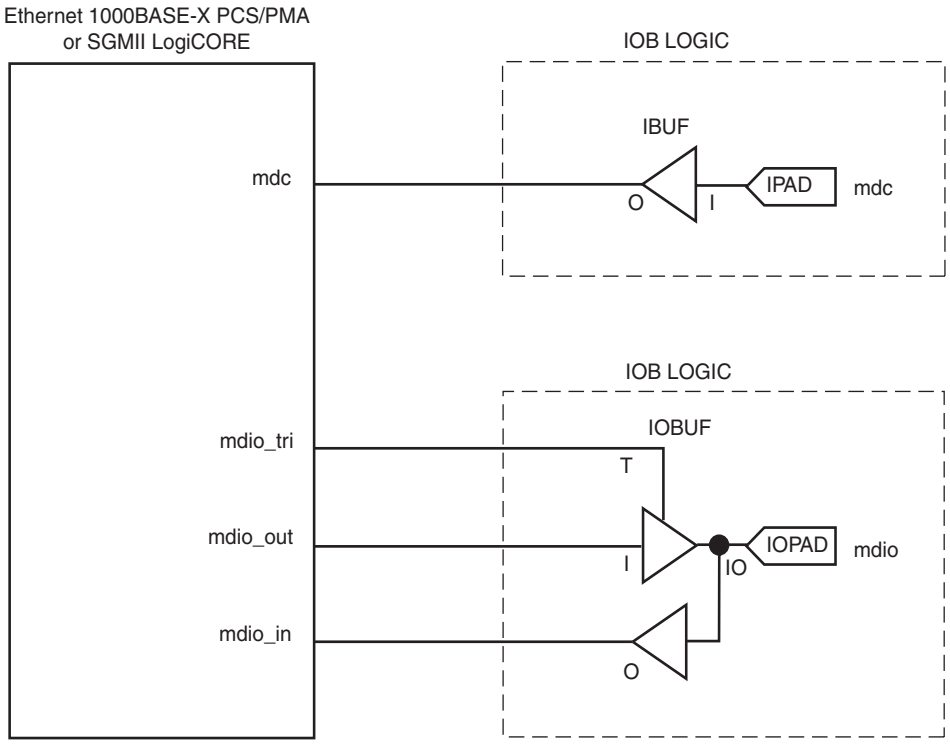


Figure 9-4: **Creating an External MDIO Interface**

Management Registers

The contents of the Management Registers can be accessed using the REGAD field of the MDIO frame. Contents will vary depending on the CORE Generator™ software options, and are defined in the following sections in this guide.

- 1000BASE-X Standard Using the Optional Auto-Negotiation
- 1000BASE-X Standard Without the Optional Auto-Negotiation
- SGMII Standard Using the Optional Auto-Negotiation
- SGMII Standard without the Optional Auto-Negotiation
- Both 1000BASE-X and SGMII Standards

1000BASE-X Standard Using the Optional Auto-Negotiation

More information on the 1000BASE-X PCS Registers can be found in clause 37 and clause 22 of the *IEEE 802.3* specification. Registers at undefined addresses are read-only and return 0s.

Table 9-2: **MDIO Registers for 1000BASE-X with Auto-Negotiation**

Register Address	Register Name
0	Control Register
1	Status Register
2,3	PHY Identifier
4	Auto-Negotiation Advertisement Register
5	Auto-Negotiation Link Partner Ability Base Register
6	Auto-Negotiation Expansion Register
7	Auto-Negotiation Next Page Transmit Register
8	Auto-Negotiation Next Page Receive Register
15	Extended Status Register
16	Vendor Specific: Auto-Negotiation Interrupt Control

Register 0: Control Register

MDIO Register 0: Control Register

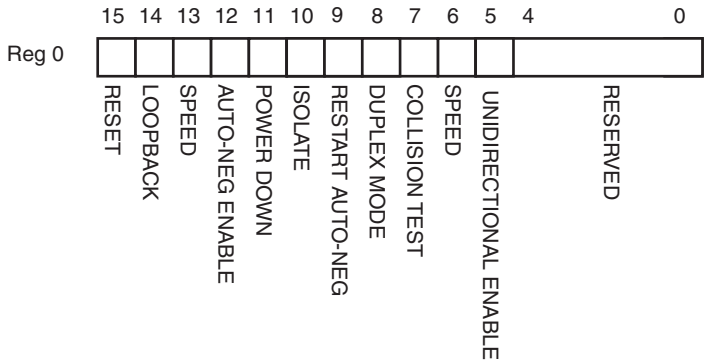


Table 9-3: **Control Register (Register 0)**

Bit(s)	Name	Description	Attributes	Default Value
0.15	Reset	1 = Core Reset 0 = Normal Operation	Read/write Self clearing	0
0.14	Loopback	1 = Enable Loopback Mode 0 = Disable Loopback Mode When used with a device-specific transceiver, the core is placed in internal loopback mode. With the TBI version, Bit 1 is connected to `ewrap`. When set to '1,' indicates to the external PMA module to enter loopback mode. See "Loopback," page 234.	Read/write	0
0.13	Speed Selection (LSB)	Always returns a 0 for this bit. Together with bit 0.6, speed selection of 1000 Mbps is identified	Returns 0	0
0.12	Auto-Negotiation Enable	1 = Enable Auto-Negotiation Process 0 = Disable Auto-Negotiation Process	Read/write	1
0.11	Power Down	1 = Power down 0 = Normal operation With the PMA option, when set to '1' the device-specific transceiver is placed in a low-power state. This bit requires a reset (see bit 0.15) to clear. With the TBI version this register bit has no effect.	Read/ write	0

Table 9-3: **Control Register (Register 0)** ***(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
0.10	Isolate	1 = Electrically Isolate PHY from GMII 0 = Normal operation	Read/write	1
0.9	Restart Auto-Negotiation	1 = Restart Auto-Negotiation Process 0 = Normal Operation	Read/write Self clearing	0
0.8	Duplex Mode	Always returns a '1' for this bit to signal Full-Duplex Mode.	Returns 1	1
0.7	Collision Test	Always returns a '0' for this bit to disable COL test.	Returns 0	0
0.6	Speed Selection (MSB)	Always returns a '1' for this bit. Together with bit 0.13, speed selection of 1000 Mbps is identified.	Returns 1	1
0.5	Unidirectional Enable	Enable transmit regardless of whether a valid link has been established.	Read/ write	0
0.4:0	Reserved	Always return 0s, writes ignored.	Returns 0s	00000

Register 1: Status Register

MDIO Register 1: Status Register

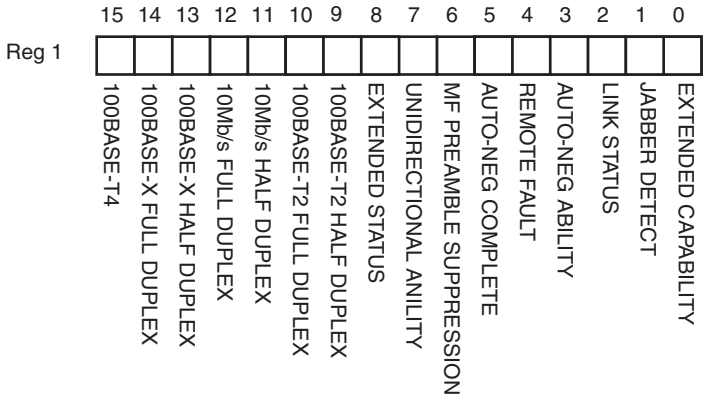


Table 9-4: **Status Register (Register 1)**

Bit(s)	Name	Description	Attributes	Default Value
1.15	100BASE-T4	Always returns a '0' as 100BASE-T4 is not supported.	Returns 0	0
1.14	100BASE-X Full Duplex	Always returns a '0' as 100BASE-X full duplex is not supported.	Returns 0	0
1.13	100BASE-X Half Duplex	Always returns a '0' as 100BASE-X half duplex is not supported.	Returns 0	0
1.12	10 Mbps Full Duplex	Always returns a '0' as 10 Mbps full duplex is not supported.	Returns 0	0
1.11	10 Mbps Half Duplex	Always returns a '0' as 10 Mbps half duplex is not supported	Returns 0	0
1.10	100BASE-T2 Full Duplex	Always returns a '0' as 100BASE-T2 full duplex is not supported.	Returns 0	0
1.9	100BASE-T2 Half Duplex	Always returns a '0' as 100BASE-T2 Half Duplex is not supported.	Returns 0	0
1.8	Extended Status	Always returns a '1' to indicate the presence of the Extended Register (Register 15).	Returns 1	1
1.7	Unidirectional Ability	Always returns a '1,' writes ignored	Returns 1	1
1.6	MF Preamble Suppression	Always returns a '1' to indicate that Management Frame Preamble Suppression is supported.	Returns 1	1
1.5	Auto- Negotiation Complete	1 = Auto-Negotiation process completed 0 = Auto-Negotiation process not completed	Read only	0
1.4	Remote Fault	1 = Remote fault condition detected 0 = No remote fault condition detected	Read only Self-clearing on read	0
1.3	Auto- Negotiation Ability	Always returns a '1' for this bit to indicate that the PHY is capable of Auto-Negotiation.	Returns 1	1
1.2	Link Status	1 = Link is up 0 = Link is down (or has been down) Latches '0' if Link Status goes down. Clears to current Link Status on read. See the following Link Status section for further details.	Read only Self clearing on read	0
1.1	Jabber Detect	Always returns a '0' for this bit since Jabber Detect is not supported.	Returns 0	0
1.0	Extended Capability	Always returns a '0' for this bit since no extended register set is supported.	Returns 0	0

Link Status

When high, the link is valid and has remained valid since this register was last read; synchronization of the link has been obtained and Auto-Negotiation (if enabled) has completed.

When low, either:

- a valid link has not been established: link synchronization has failed or Auto-Negotiation (if enabled) has failed to complete.
- OR, link synchronization was lost at some point since this register was previously read. However, the current link status may be good. **Therefore read this register a 2nd time to get confirmation of the current link status**.

Regardless of whether Auto-Negotiation is enabled or disabled, there can be some delay to the deassertion of Link Status following the loss of synchronization of a previously successful link. This is due to the Auto-Negotiation state machine which requires that synchronization is lost for an entire link timer duration before changing state. For more information, see the 802.3 specification (the *an_sync_status* variable).

Registers 2 and 3: PHY Identifiers

Registers 2 and 3: PHY Identifiers

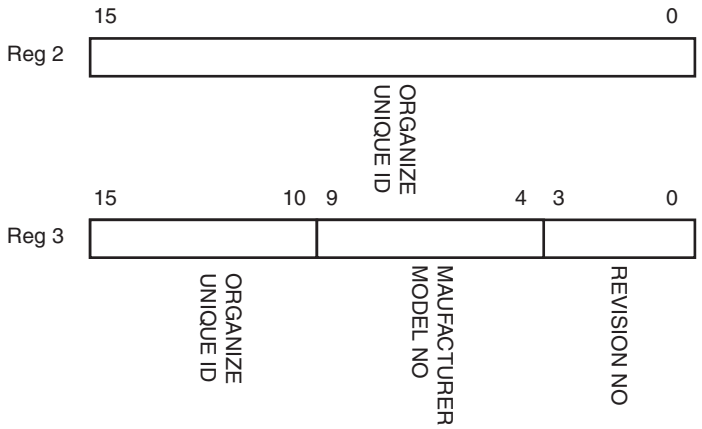


Table 9-5: **PHY Identifier (Registers 2 and 3)**

Bit(s)	Name	Description	Attributes	Default Value
2.15:0	Organizationally Unique Identifier	Always return 0s	returns 0s	0000000000000000
3.15:10	Organizationally Unique Identifier	Always return 0s	returns 0s	000000
3.9:4	Manufacturer model number	Always return 0s	returns 0s	000000
3.3:0	Revision Number	Always return 0s	returns 0s	0000

Register 4: Auto-Negotiation Advertisement

MDIO Register 4: Auto-Negotiation Advertisement

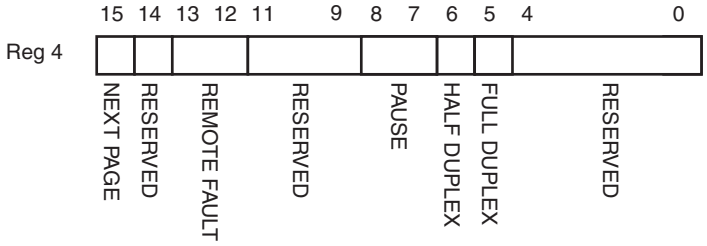


Table 9-6: **Auto-Negotiation Advertisement Register (Register 4)**

Bit(s)	Name	Description	Attributes	Default Value
4.15	Next Page	1 = Next Page functionality is advertised 0 = Next Page functionality is not advertised	read/write	0
4.14	Reserved	Always returns '0,' writes ignored	returns 0	0
4.13:12	Remote Fault	00 = No Error 01 = Offline 10 = Link Failure 11 = Auto-Negotiation Error	read/write self clearing to 00 after auto-negotiation	00
4.11:9	Reserved	Always return 0s, writes ignored	returns 0	0
4.8:7	Pause	00 = No PAUSE 01 = Symmetric PAUSE 10 = Asymmetric PAUSE towards link partner 11 = Both Symmetric PAUSE and Asymmetric PAUSE towards link partner	read/write	11
4.6	Half Duplex	Always returns a '0' for this bit since Half Duplex Mode is not supported	returns 0	0
4.5	Full Duplex	1 = Full Duplex Mode is advertised 0 = Full Duplex Mode is not advertised	read/write	1
4.4:0	Reserved	Always return 0s , writes ignored	returns 0s	00000

Register 5: Auto-Negotiation Link Partner Base

MDIO Register 5: Auto-Negotiation Link Partner Base

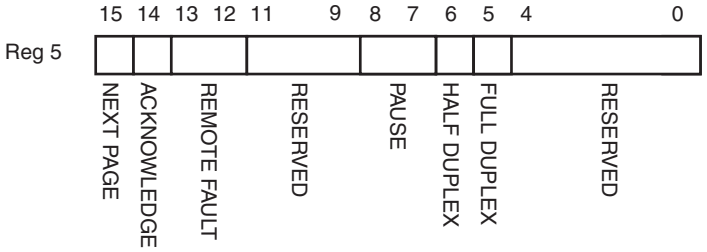


Table 9-7: **Auto-Negotiation Link Partner Ability Base Register (Register 5)**

Bit(s)	Name	Description	Attributes	Default Value
5.15	Next Page	1 = Next Page functionality is supported 0 = Next Page functionality is not supported	read only	0
5.14	Acknowledge	Used by Auto-Negotiation function to indicate reception of a link partner's base or next page	read only	0
5.13:12	Remote Fault	00 = No Error 01 = Offline 10 = Link Failure 11 = Auto-Negotiation Error	read only	00
5.11:9	Reserved	Always return 0s	returns 0s	000
5.8:7	Pause	00 = No PAUSE 01 = Symmetric PAUSE 10 = Asymmetric PAUSE towards link partner 11 = Both Symmetric PAUSE and Asymmetric PAUSE supported	read only	00
5.6	Half Duplex	1 = Half Duplex Mode is supported 0 = Half Duplex Mode is not supported	read only	0
5.5	Full Duplex	1 = Full Duplex Mode is supported 0 = Full Duplex Mode is not supported	read only	0
5.4:0	Reserved	Always return 0s	returns 0s	00000

Register 6: Auto-Negotiation Expansion

MDIO Register 6: Auto-Negotiation Expansion

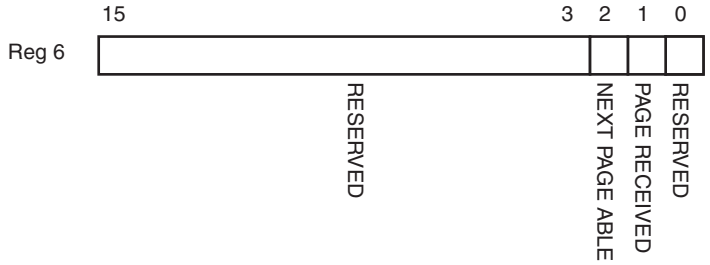


Table 9-8: **Auto-Negotiation Expansion Register (Register 6)**

Bit(s)	Name	Description	Attributes	Default Value
6.15:3	Reserved	Always returns 0s	returns 0s	0000000000000
6.2	Next Page Able	Always returns a '1' for this bit since the device is Next Page Able	returns 1	1
6.1	Page Received	1 = A new page has been received 0 = A new page has not been received	read only self clearing on read	0
6.0	Reserved	Always returns 0s	returns 0s	0000000

Register 7: Next Page Transmit

MDIO Register 7: Next Page Transmit

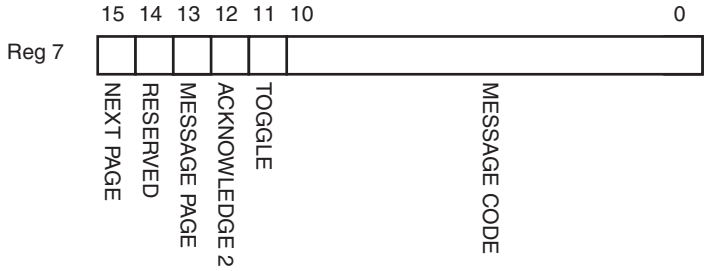


Table 9-9: **Auto-Negotiation Next Page Transmit (Register 7)**

Bit(s)	Name	Description	Attributes	Default Value
7.15	Next Page	1 = Additional Next Page(s) will follow 0 = Last page	read/ write	0
7.14	Reserved	Always returns '0'	returns 0	0
7.13	Message Page	1 = Message Page 0 = Unformatted Page	read/ write	1
7.12	Acknowledge 2	1 = Comply with message 0 = Cannot comply with message	read/ write	0
7.11	Toggle	Value toggles between subsequent Next Pages	read only	0
7.10:0	Message / Unformatted Code Field	Message Code Field or Unformatted Page Encoding as dictated by 7.13	read/ write	00000000001 (Null Message Code)

Register 8: Next Page Receive

MDIO Register 8: Next Page Receive

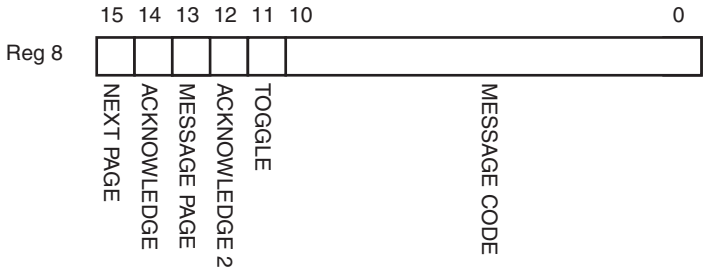


Table 9-10: **Auto-Negotiation Next Page Receive (Register 8)**

Bit(s)	Name	Description	Attributes	Default Value
8.15	Next Page	1 = Additional Next Page(s) will follow 0 = Last page	read only	0
8.14	Acknowledge	Used by Auto-Negotiation function to indicate reception of a link partner's base or next page	read only	0
8.13	Message Page	1 = Message Page 0 = Unformatted Page	read only	0
8.12	Acknowledge 2	1 = Comply with message 0 = Cannot comply with message	read only	0
8.11	Toggle	Value toggles between subsequent Next Pages	read only	0
8.10:0	Message / Unformatted Code Field	Message Code Field or Unformatted Page Encoding as dictated by 8.13	read only	00000000000

Register 15: Extended Status

MDIO Register 15: Extended Status Register

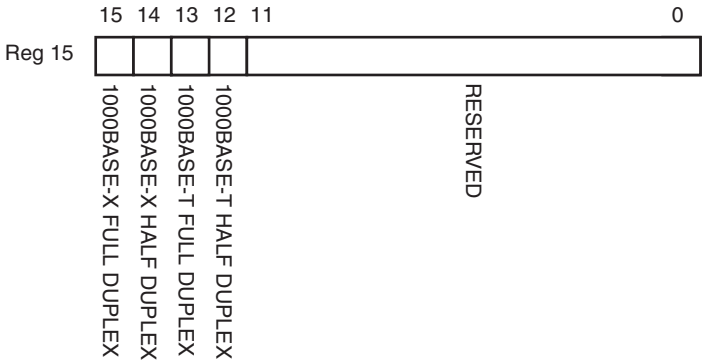


Table 9-11: **Extended Status Register (Register 15)**

Bit(s)	Name	Description	Attributes	Default Value
15.15	1000BASE-X Full Duplex	Always returns a '1' for this bit since 1000BASE-X Full Duplex is supported	returns 1	1
15.14	1000BASE-X Half Duplex	Always returns a '0' for this bit since 1000BASE-X Half Duplex is not supported	returns 0	0
15.13	1000BASE-T Full Duplex	Always returns a '0' for this bit since 1000BASE-T Full Duplex is not supported	returns 0	0
15.12	1000BASE-T Half Duplex	Always returns a '0' for this bit since 1000BASE-T Half Duplex is not supported	returns 0	0
15:11:0	Reserved	Always return 0s	returns 0s	000000000000

Register 16: Vendor-Specific Auto-Negotiation Interrupt Control

MDIO Register 16: Vendor Specific Auto-Negotiation Interrupt Control

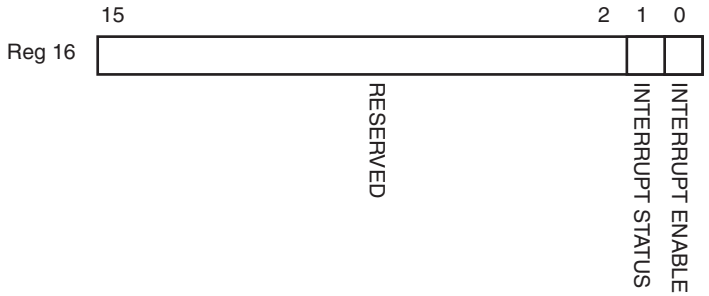


Table 9-12: **Vendor Specific Register: Auto-Negotiation Interrupt Control Register (Register 16)**

Bit(s)	Name	Description	Attributes	Default Value
16.15:2	Reserved	Always return 0s	returns 0s	00000000000000
16.1	Interrupt Status	1 = Interrupt is asserted 0 = Interrupt is not asserted If the interrupt is enabled, this bit is asserted on the completion of an Auto-Negotiation cycle; it is only cleared by writing '0' to this bit. If the Interrupt is disabled, the bit is set to '0.' NOTE: the `an_interrupt` port of the core is wired to this bit.	read/ write	0
16.0	Interrupt Enable	1 = Interrupt enabled 0 = Interrupt disabled	read/ write	1

1000BASE-X Standard Without the Optional Auto-Negotiation

It is not the intention of this document to fully describe the 1000BASE-X PCS Registers. See clauses 37 and 22 of the *IEEE 802.3* Specification for further information.

Registers at undefined addresses are read-only and return 0s.

Table 9-13: **MDIO Registers for 1000BASE-X without Auto-Negotiation**

Register Address	Register Name
0	Control Register
1	Status Register
2,3	PHY Identifier
15	Extended Status Register

Register 0: Control Register

MDIO Register 0: Control Register

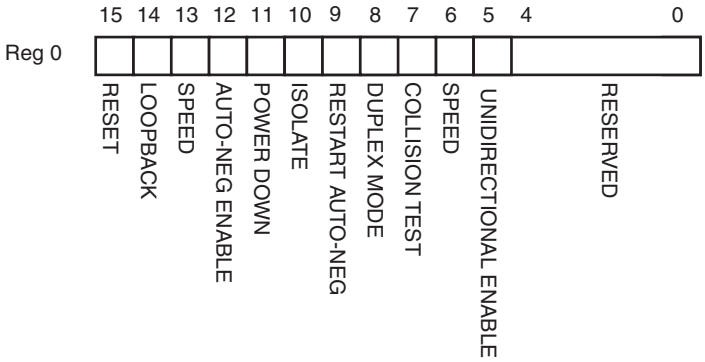


Table 9-14: **Control Register (Register 0)**

Bit(s)	Name	Description	Attributes	Default Value
0.15	Reset	1 = PCS/PMA reset 0 = Normal Operation	read/write self clearing	0
0.14	Loopback	1 = Enable Loopback Mode 0 = Disable Loopback Mode When used with a device-specific transceiver, the core is placed in internal loopback mode. With the TBI version, Bit 1 is connected to `ewrap`. When set to '1' indicates to the external PMA module to enter loopback mode. See "Loopback," page 234.	read/write	0
0.13	Speed Selection (LSB)	Always returns a 0 for this bit. Together with bit 0.6, speed selection of 1000 Mbps is identified.	returns 0	0
0.12	Auto-Negotiation Enable	Ignore this bit because Auto-Negotiation is not included.	read/ write	1
0.11	Power Down	1 = Power down 0 = Normal operation With the PMA option, when set to '1' the device-specific transceiver is placed in a low- power state. This bit requires a reset (see bit 0.15) to clear. With the TBI version this register bit has no effect.	read/ write	0
0.10	Isolate	1 = Electrically Isolate PHY from GMII 0 = Normal operation	read/write	1

Table 9-14: **Control Register (Register 0) *(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
0.9	Restart Auto-Negotiation	Ignore this bit because Auto-Negotiation is not included.	read/ write	0
0.8	Duplex Mode	Always returns a '1' for this bit to signal Full-Duplex Mode.	returns 1	1
0.7	Collision Test	Always returns a '0' for this bit to disable COL test.	returns 0	0
0.6	Speed Selection (MSB)	Always returns a '1' for this bit. Together with bit 0.13, speed selection of 1000 Mbps is identified	returns 1	1
0.5	Unidirectional Enable	Ignore this bit because Auto-Negotiation is not included.	read/ write	0
0.4:0	Reserved	Always return 0s , writes ignored.	returns 0s	00000

Register 1: Status Register

MDIO Register 1: Status Register

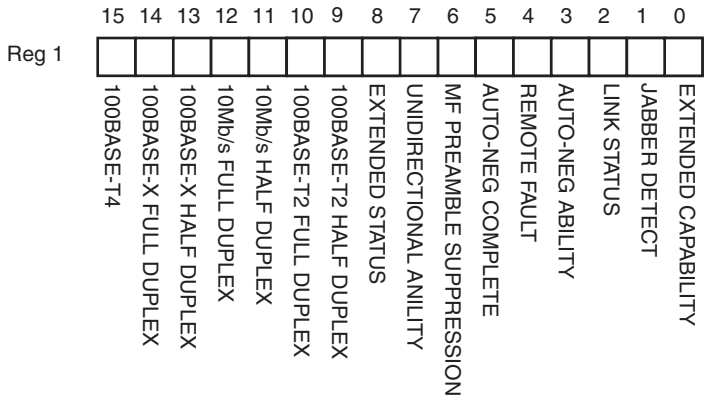


Table 9-15: **Status Register (Register 1)**

Bit(s)	Name	Description	Attributes	Default Value
1.15	100BASE-T4	Always returns a '0' for this bit since 100BASE-T4 is not supported	returns 0	0
1.14	100BASE-X Full Duplex	Always returns a '0' for this bit since 100BASE-X Full Duplex is not supported	returns 0	0
1.13	100BASE-X Half Duplex	Always returns a '0' for this bit since 100BASE-X Half Duplex is not supported	returns 0	0
1.12	10 Mbps Full Duplex	Always returns a '0' for this bit since 10 Mbps Full Duplex is not supported	returns 0	0
1.11	10 Mbps Half Duplex	Always returns a '0' for this bit since 10 Mbps Half Duplex is not supported	returns 0	0

Table 9-15: **Status Register (Register 1)** ***(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
1.10	100BASE-T2 Full Duplex	Always returns a '0' for this bit since 100BASE-T2 Full Duplex is not supported	returns 0	0
1.9	100BASE-T2 Half Duplex	Always returns a '0' for this bit since 100BASE-T2 Half Duplex is not supported	returns 0	0
1.8	Extended Status	Always returns a '1' for this bit to indicate the presence of the Extended Register (Register 15)	returns 1	1
1.7	Unidirectional Ability	Always returns 1, writes ignored	returns 1	1
1.6	MF Preamble Suppression	Always returns a '1' for this bit to indicate that Management Frame Preamble Suppression is supported	returns 1	1
1.5	Auto- Negotiation Complete	Ignore this bit because Auto-Negotiation is not included.	returns 1	1
1.4	Remote Fault	Always returns a '0' for this bit because Auto-Negotiation is not included.	returns 0	0
1.3	Auto- Negotiation Ability	Ignore this bit because Auto-Negotiation is not included.	returns 0	0
1.2	Link Status	1 = Link is up 0 = Link is down Latches '0' if Link Status goes down. Clears to current Link Status on read. See the following Link Status section for further details.	read only self clearing on read	0
1.1	Jabber Detect	Always returns a '0' for this bit since Jabber Detect is not supported	returns 0	0
1.0	Extended Capability	Always returns a '0' for this bit since no extended register set is supported	returns 0	0

Link Status

When high, the link is valid and has remained valid since this register was last read; synchronization of the link has been obtained.

When low, either:

- a valid link has not been established; link synchronization has failed.
- OR, link synchronization was lost at some point since this register was previously read. However, the current link status may be good. **Therefore read this register a 2nd time to get confirmation of the current link status**.

Registers 2 and 3: Phy Identifier

MDIO Registers 2 and 3: PHY Identifier

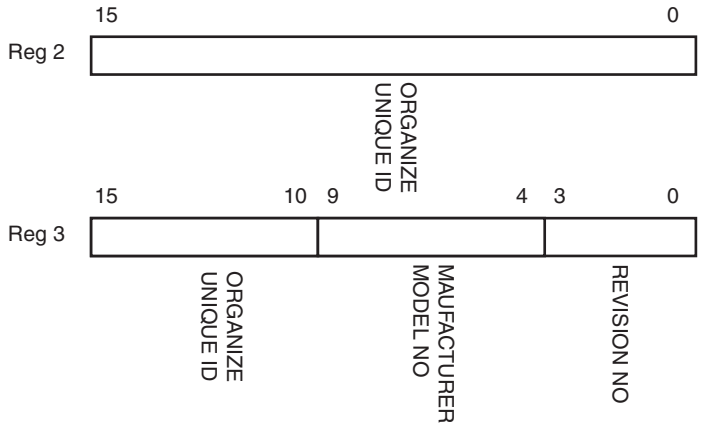


Table 9-16: **PHY Identifier (Registers 2 and 3)**

Bit(s)	Name	Description	Attributes	Default Value
2.15:0	Organizationally Unique Identifier	Always return 0s	returns 0s	0000000000000000
3.15:10	Organizationally Unique Identifier	Always return 0s	returns 0s	000000
3.9:4	Manufacturer model number	Always return 0s	returns 0s	000000
3.3:0	Revision Number	Always return 0s	returns 0s	0000

Register 15: Extended Status

MDIO Register 15: Extended Status

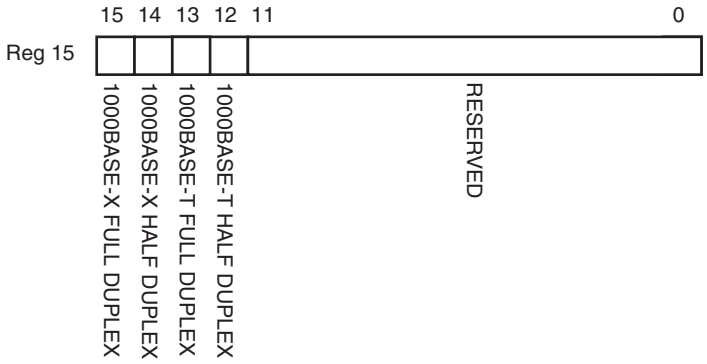


Table 9-17: **Extended Status (Register 15)**

Bit(s)	Name	Description	Attributes	Default Value
15.15	1000BASE-X Full Duplex	Always returns a '1' since 1000BASE-X Full Duplex is supported	returns 1	1
15.14	1000BASE-X Half Duplex	Always returns a '0' since 1000BASE-X Half Duplex is not supported	returns 0	0
15.13	1000BASE-T Full Duplex	Always returns a '0' since 1000BASE-T Full Duplex is not supported	returns 0	0
15.12	1000BASE-T Half Duplex	Always returns a '0' since 1000BASE-T Half Duplex is not supported	returns 0	0
15:11:0	Reserved	Always return 0s	returns 0s	0000000000 00

SGMII Standard Using the Optional Auto-Negotiation

The registers provided for SGMII operation in this core are adaptations of those defined in *IEEE 802.3* clauses 37 and 22. In an SGMII implementation, two different types of links exist. They are the SGMII link between the MAC and PHY (SGMII link) and the link across the Ethernet Medium itself (Medium). See Figure 10-2.

Information regarding the state of both of these links is contained within the following registers. Where applicable, the abbreviations *SGMII link* and *Medium* are used in the register descriptions. Registers at undefined addresses are read-only and return 0s.

Table 9-18: **MDIO Registers for 1000BASE-X with Auto-Negotiation**

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

Register 0: SGMII Control

MDIO Register 0: SGMII Control

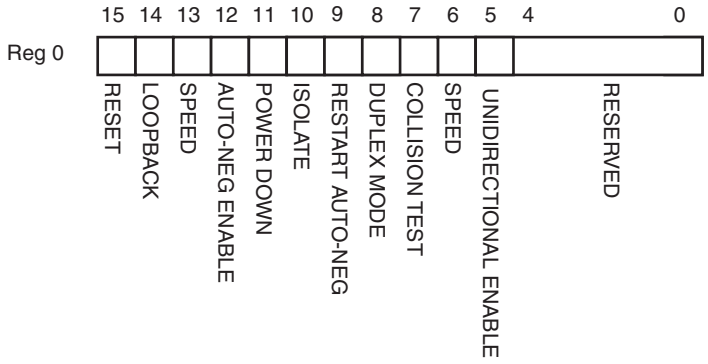


Table 9-19: **SGMII Control (Register 0)**

Bit(s)	Name	Description	Attributes	Default Value
0.15	Reset	1 = Core Reset 0 = Normal Operation	read/write self clearing	0
0.14	Loopback	1 = Enable Loopback Mode 0 = Disable Loopback Mode When used with a device-specific transceiver, the core is placed in internal loopback mode. With the TBI version, Bit 1 is connected to `ewrap`. When set to '1' indicates to the external PMA module to enter loopback mode. See "Loopback," page 234.	read/write	0
0.13	Speed Selection (LSB)	Always returns a '0' for this bit. Together with bit 0.6, speed selection of 1000 Mbps is identified	returns 0	0
0.12	Auto-Negotiation Enable	1 = Enable SGMII Auto-Negotiation Process 0 = Disable SGMII Auto-Negotiation Process	read/write	1
0.11	Power Down	1 = Power down 0 = Normal operation With the PMA option, when set to '1' the device-specific transceiver is placed in a low-power state. This bit requires a reset (see bit 0.15) to clear. With the TBI version this register bit has no effect.	read/ write	0
0.10	Isolate	1 = Electrically Isolate SGMII logic from GMII 0 = Normal operation	read/write	1
0.9	Restart Auto-Negotiation	1 = Restart Auto-Negotiation Process across SGMII link 0 = Normal Operation	read/write self clearing	0
0.8	Duplex Mode	Always returns a '1' for this bit to signal Full-Duplex Mode	returns 1	1
0.7	Collision Test	Always returns a '0' for this bit to disable COL test	returns 0	0
0.6	Speed Selection (MSB)	Always returns a '1' for this bit. Together with bit 0.13, speed selection of 1000 Mbps is identified	returns 1	1

Table 9-19: **SGMII Control (Register 0) *(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
0.5	Unidirectional Enable	Enable transmit regardless of whether a valid link has been established	read/ write	0
0.4:0	Reserved	Always return 0s , writes ignored	returns 0s	00000

Register 1: SGMII Status

MDIO Register 1: SGMII Status

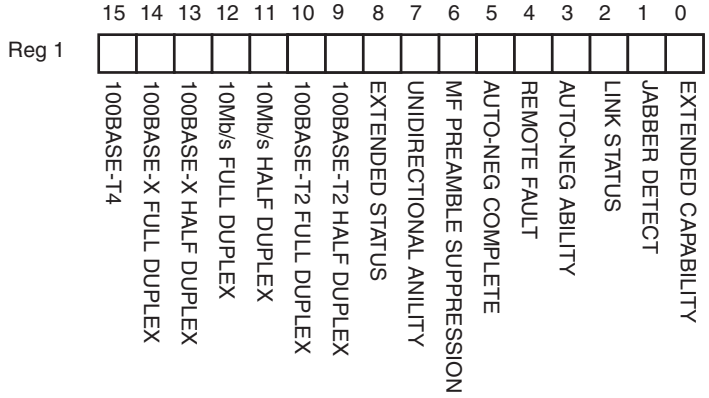


Table 9-20: **SGMII Status (Register 1)**

Bit(s)	Name	Description	Attributes	Default Value
1.15	100BASE-T4	Always returns a '0' for this bit because 100BASE-T4 is not supported	returns 0	0
1.14	100BASE-X Full Duplex	Always returns a '0' for this bit because 100BASE-X Full Duplex is not supported	returns 0	0
1.13	100BASE-X Half Duplex	Always returns a '0' for this bit because 100BASE-X Half Duplex is not supported	returns 0	0
1.12	10 Mbps Full Duplex	Always returns a '0' for this bit because 10 Mbps Full Duplex is not supported	returns 0	0
1.11	10 Mbps Half Duplex	Always returns a '0' for this bit because 10 Mbps Half Duplex is not supported	returns 0	0
1.10	100BASE-T2 Full Duplex	Always returns a '0' for this bit because 100BASE-T2 Full Duplex is not supported	returns 0	0
1.9	100BASE-T2 Half Duplex	Always returns a '0' for this bit because 100BASE-T2 Half Duplex is not supported	returns 0	0
1.8	Extended Status	Always returns a '1' for this bit to indicate the presence of the Extended Register (Register 15)	returns 1	1

Table 9-20: **SGMII Status (Register 1)** ***(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
1.7	Unidirectional Ability	Always returns '1,' writes ignored	returns 1	1
1.6	MF Preamble Suppression	Always returns a '1' for this bit to indicate that Management Frame Preamble Suppression is supported	returns 1	1
1.5	Auto- Negotiation Complete	1 = Auto-Negotiation process completed across SGMII link 0 = Auto-Negotiation process not completed across SGMII link	read only	0
1.4	Remote Fault	1 = A fault on the Medium has been detected 0 = No fault of the Medium has been detected	read only self clearing on read	0
1.3	Auto- Negotiation Ability	Always returns a '1' for this bit to indicate that the SGMII core is capable of Auto-Negotiation	returns 1	1
1.2	SGMII Link Status	1 = SGMII Link is up 0 = SGMII Link is down Latches '0' if SGMII Link Status goes down. Clears to current SGMII Link Status on read. See the following Link Status section for further details.	read only self clearing on read	0
1.1	Jabber Detect	Always returns a '0' for this bit since Jabber Detect is not supported	returns 0	0
1.0	Extended Capability	Always returns a '0' for this bit because no extended register set is supported	returns 0	0

Link Status

When high, the link is valid and has remained valid since this register was last read: synchronization of the link has been obtained and Auto-Negotiation (if enabled) has completed.

When low, either:

- a valid link has not been established; link synchronization has failed or Auto-Negotiation (if enabled) has failed to complete.
- OR, link synchronization was lost at some point since this register was previously read. However, the current link status may be good. **Therefore read this register a 2nd time to get confirmation of the current link status**.

Regardless of whether Auto-Negotiation is enabled or disabled, there can be some delay to the deassertion of Link Status following the loss of synchronization of a previously successful link. This is due to the Auto-Negotiation state machine which requires that synchronization is lost for an entire link timer duration before changing state. For more information, see the 802.3 specification (the *an_sync_status* variable).

Registers 2 and 3: PHY Identifier

MDIO Registers 2 and 3: PHY Identifier

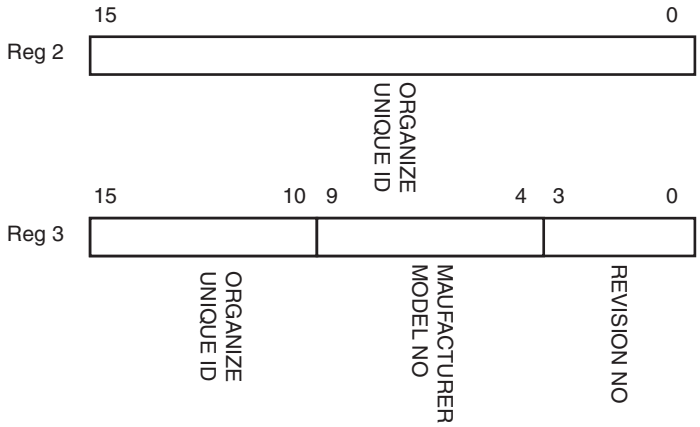


Table 9-21: **PHY Identifier (Registers 2 and 3)**

Bit(s)	Name	Description	Attributes	Default Value
2.15:0	Organizationally Unique Identifier	Always return 0s	returns 0s	0000000000000000
3.15:10	Organizationally Unique Identifier	Always return 0s	returns 0s	000000
3.9:4	Manufacturer model number	Always return 0s	returns 0s	000000
3.3:0	Revision Number	Always return 0s	returns 0s	0000

Register 4: SGMII Auto-Negotiation Advertisement

MDIO Register 4: SGMII Auto-Negotiation Advertisement

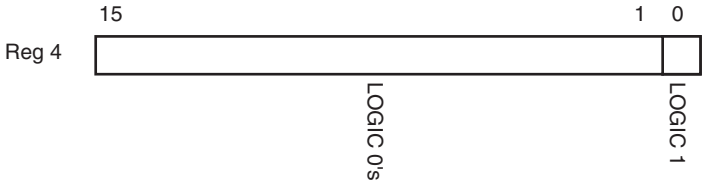
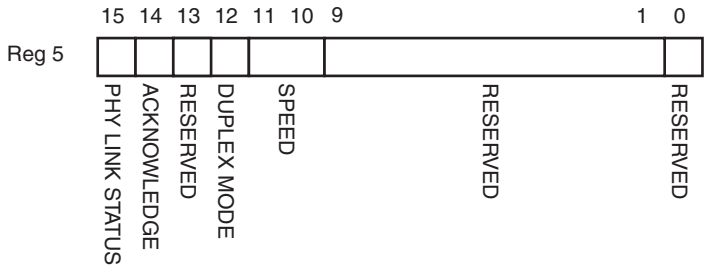


Table 9-22: **SGMII Auto-Negotiation Advertisement (Register 4)**

Bit(s)	Name	Description	Attributes	Default Value
4.15:0	All bits	SGMII defined value sent from the MAC to the PHY	read only	0000000000000001

Register 5: SGMII Auto-Negotiation Link Partner Ability

MDIO Register 5: SGMII Auto-Negotiation Link Partner Ability



The Auto-Negotiation Ability Base Register (Register 5) contains information related to the status of the link between the PHY and its physical link partner across the Medium.

Table 9-23: **SGMII Auto-Negotiation Link Partner Ability Base (Register 5)**

Bit(s)	Name	Description	Attributes	Default Value
5.15	PHY Link Status	This refers to the link status of the PHY with its link partner across the Medium. 1 = Link Up 0 = Link Down	read only	1
5.14	Acknowledge	Used by Auto-Negotiation function to indicate reception of a link partner's base or next page	read only	0
5.13	Reserved	Always returns '0,' writes ignored	returns 0	0
5.12	Duplex Mode	1= Full Duplex 0 = Half Duplex	read only	0
5.11:10	Speed	11 = Reserved 10 = 1 Gbps 01 = 100 Mbps 00 = 10 Mbps	read only	00
5.9:1	Reserved	Always return 0s	returns 0s	000000000
5:0	Reserved	Always returns '1'	returns 1	1

Register 6: SGMII Auto-Negotiation Expansion

MDIO Register 6: SGMII Auto-Negotiation Expansion

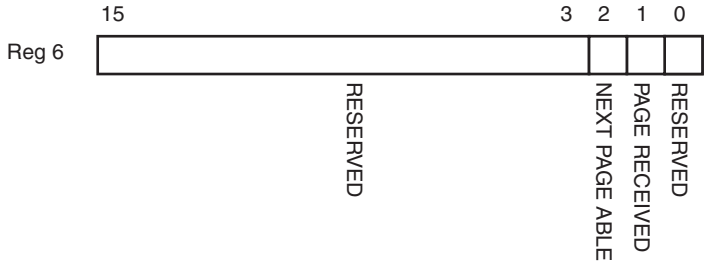


Table 9-24: **SGMII Auto-Negotiation Expansion (Register 6)**

Bit(s)	Name	Description	Attributes	Default Value
6.15:3	Reserved	Always return 0s	returns 0s	0000000000000
6.2	Next Page Able	Always returns a '1' for this bit since the device is Next Page Able	returns 1	1
6.1	Page Received	1 = A new page has been received 0 = A new page has not been received	read only self clearing on read	0
6.0	Reserved	Always return 0s	returns 0s	0000000

Register 7: SGMII Auto-Negotiation Next Page Transmit

MDIO Register 7: SGMII Auto-Negotiation Next Page Transmit

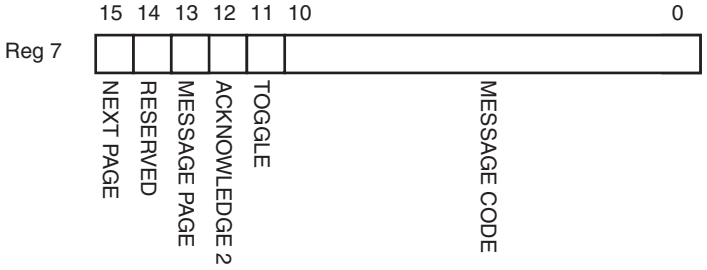


Table 9-25: **SGMII Auto-Negotiation Next Page Transmit (Register 7)**

Bit(s)	Name	Description	Attributes	Default Value
7.15	Next Page	1 = Additional Next Page(s) will follow 0 = Last page	read/ write	0
7.14	Reserved	Always returns '0'	returns 0	0
7.13	Message Page	1 = Message Page 0 = Unformatted Page	read/ write	1
7.12	Acknowledge 2	1 = Comply with message 0 = Cannot comply with message	read/ write	0
7.11	Toggle	Value toggles between subsequent Next Pages	read only	0
7.10:0	Message / Unformatted Code Field	Message Code Field or Unformatted Page Encoding as dictated by 7.13	read/ write	00000000001 (Null Message Code)

Register 8: SGMII Next Page Receive

MDIO Register 8: SGMII Next Page Receive

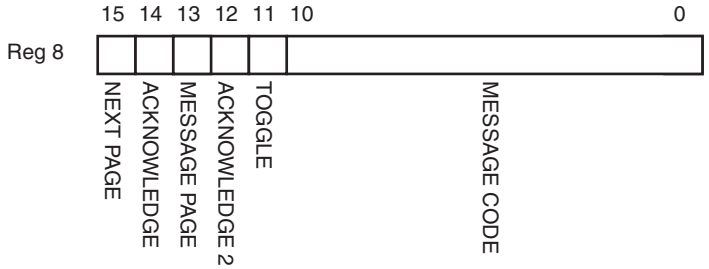


Table 9-26: **SGMII Auto-Negotiation Next Page Receive (Register 8)**

Bit(s)	Name	Description	Attributes	Default Value
8.15	Next Page	1 = Additional Next Page(s) will follow 0 = Last page	read only	0
8.14	Acknowledge	Used by Auto-Negotiation function to indicate reception of a link partner's base or next page	read only	0
8.13	Message Page	1 = Message Page 0 = Unformatted Page	read only	0
8.12	Acknowledge 2	1 = Comply with message 0 = Cannot comply with message	read only	0
8.11	Toggle	Value toggles between subsequent Next Pages	read only	0
8.10:0	Message / Unformatted Code Field	Message Code Field or Unformatted Page Encoding as dictated by 8.13	read only	00000000000

Register 15: SGMII Extended Status

MDIO Register 15: SGMII Extended Status

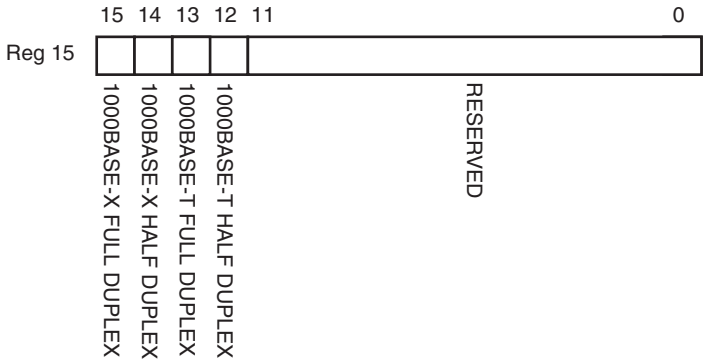


Table 9-27: **SGMII Extended Status Register (Register 15)**

Bit(s)	Name	Description	Attributes	Default Value
15.15	1000BASE-X Full Duplex	Always returns a '1' for this bit since 1000BASE-X Full Duplex is supported	returns 1	1
15.14	1000BASE-X Half Duplex	Always returns a '0' for this bit since 1000BASE-X Half Duplex is not supported	returns 0	0
15.13	1000BASE-T Full Duplex	Always returns a '0' for this bit since 1000BASE-T Full Duplex is not supported	returns 0	0
15.12	1000BASE-T Half Duplex	Always returns a '0' for this bit since 1000BASE-T Half Duplex is not supported	returns 0	0
15:11:0	Reserved	Always return 0s	returns 0s	000000000000

Register 16: SGMII Auto-Negotiation Interrupt Control

MDIO Register 16: SGMII Auto-Negotiation Interrupt Control

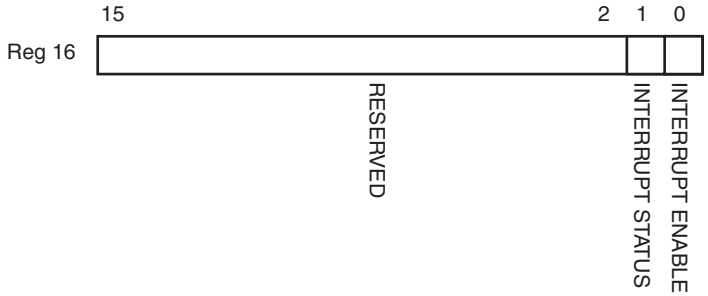


Table 9-28: **SGMII Auto-Negotiation Interrupt Control (Register 16)**

Bit(s)	Name	Description	Attributes	Default Value
16.15:2	Reserved	Always return 0s	returns 0s	00000000000000
16.1	Interrupt Status	1 = Interrupt is asserted 0 = Interrupt is not asserted If the interrupt is enabled, this bit is asserted on completion of an Auto-Negotiation cycle across the SGMII link; it is only cleared by writing '0' to this bit. If the Interrupt is disabled, the bit is set to '0.' NOTE: The an_interrupt port of the core is wired to this bit.	read/ write	0
16.0	Interrupt Enable	1 = Interrupt enabled 0 = Interrupt disabled	read/ write	1

SGMII Standard without the Optional Auto-Negotiation

The Registers provided for SGMII operation in this core are adaptations of those defined in *IEEE 802.3* clauses 37 and 22. In an SGMII implementation, two different types of links exist. They are the SGMII link between the MAC and PHY (SGMII link) and the link across the Ethernet Medium itself (Medium). See Figure 10-2. Information about the state of the SGMII link is available in registers that follow.

The state of the link across the Ethernet Medium itself is not directly available when SGMII Auto-Negotiation is not present. For this reason, the status of the link and the results of the PHYs Auto-Negotiation (for example, Speed and Duplex mode) must be obtained directly from the management interface of connected PHY module. Registers at undefined addresses are read-only and return 0s.

Table 9-29: **MDIO Registers for 1000BASE-X with Auto-Negotiation**

Register Address	Register Name
0	SGMII Control Register
1	SGMII Status Register
2,3	PHY Identifier
4	SGMII Auto-Negotiation Advertisement Register
15	SGMII Extended Status Register

Register 0: SGMII Control

MDIO Register 0: SGMII Control

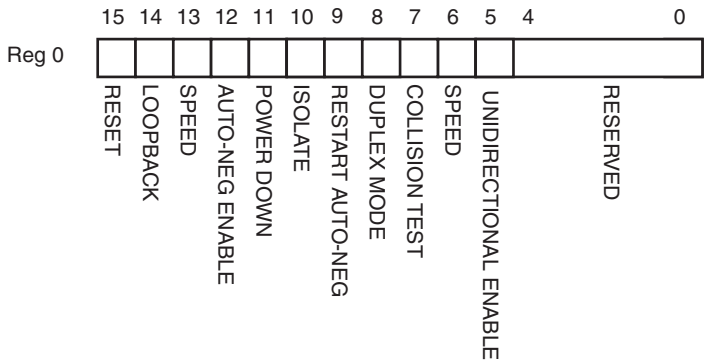


Table 9-30: **SGMII Control (Register 0)**

Bit(s)	Name	Description	Attributes	Default Value
0.15	Reset	1 = Core Reset 0 = Normal Operation	read/write self clearing	0
0.14	Loopback	1 = Enable Loopback Mode 0 = Disable Loopback Mode When used with a device-specific transceiver, the core is placed in internal loopback mode. With the TBI version, Bit 1 is connected to `ewrap`. When set to '1' indicates to the external PMA module to enter loopback mode. See "Loopback," page 234.	read/write	0
0.13	Speed Selection (LSB)	Always returns a '0' for this bit. Together with bit 0.6, speed selection of 1000 Mbps is identified	returns 0	0
0.12	Auto-Negotiation Enable	1 = Enable SGMII Auto-Negotiation Process 0 = Disable SGMII Auto-Negotiation Process	read/write	1
0.11	Power Down	1 = Power down 0 = Normal operation With the PMA option, when set to '1' the device-specific transceiver is placed in a low-power state. This bit requires a reset (see bit 0.15) to clear. With the TBI version this register bit has no effect.	read/ write	0
0.10	Isolate	1 = Electrically Isolate SGMII logic from GMII 0 = Normal operation	read/write	1
0.9	Restart Auto-Negotiation	1 = Restart Auto-Negotiation Process across SGMII link 0 = Normal Operation	read/write self clearing	0
0.8	Duplex Mode	Always returns a '1' for this bit to signal Full-Duplex Mode	returns 1	1
0.7	Collision Test	Always returns a '0' for this bit to disable COL test	returns 0	0
0.6	Speed Selection (MSB)	Always returns a '1' for this bit. Together with bit 0.13, speed selection of 1000 Mbps is identified	returns 1	1

Table 9-30: **SGMII Control (Register 0) *(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
0.5	Unidirectional Enable	Enable transmit regardless of whether a valid link has been established	read/ write	0
0.4:0	Reserved	Always return 0s , writes ignored	returns 0s	00000

Register 1: SGMII Status

MDIO Register 1: SGMII Status

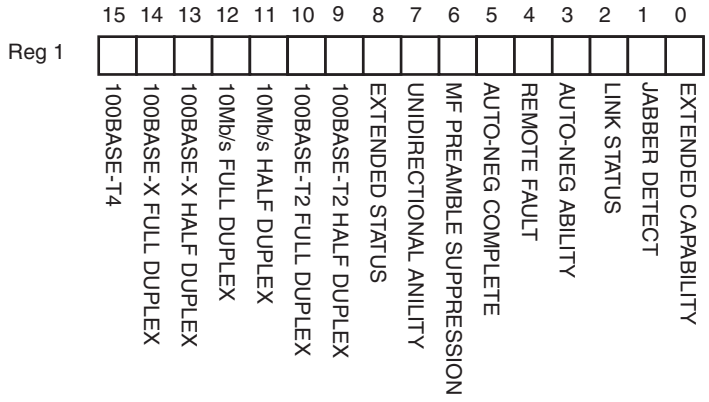


Table 9-31: **SGMII Status (Register 1)**

Bit(s)	Name	Description	Attributes	Default Value
1.15	100BASE-T4	Always returns a '0' for this bit because 100BASE-T4 is not supported	returns 0	0
1.14	100BASE-X Full Duplex	Always returns a '0' for this bit because 100BASE-X Full Duplex is not supported	returns 0	0
1.13	100BASE-X Half Duplex	Always returns a '0' for this bit because 100BASE-X Half Duplex is not supported	returns 0	0
1.12	10 Mbps Full Duplex	Always returns a '0' for this bit because 10 Mbps Full Duplex is not supported	returns 0	0
1.11	10 Mbps Half Duplex	Always returns a '0' for this bit because 10 Mbps Half Duplex is not supported	returns 0	0
1.10	100BASE-T2 Full Duplex	Always returns a '0' for this bit because 100BASE-T2 Full Duplex is not supported	returns 0	0
1.9	100BASE-T2 Half Duplex	Always returns a '0' for this bit because 100BASE-T2 Half Duplex is not supported	returns 0	0
1.8	Extended Status	Always returns a '1' for this bit to indicate the presence of the Extended Register (Register 15)	returns 1	1

Table 9-31: **SGMII Status (Register 1) *(Continued)***

Bit(s)	Name	Description	Attributes	Default Value
1.7	Unidirectional Ability	Always returns '1,' writes ignored	returns 1	1
1.6	MF Preamble Suppression	Always returns a '1' for this bit to indicate that Management Frame Preamble Suppression is supported	returns 1	1
1.5	Auto- Negotiation Complete	Ignore this bit because Auto-Negotiation is not included.	returns 1	0
1.4	Remote Fault	Ignore this bit because Auto-Negotiation is not included	returns 0	0
1.3	Auto- Negotiation Ability	Ignore this bit because Auto-Negotiation is not included	returns 0	0
1.2	SGMII Link Status	1 = SGMII Link is up 0 = SGMII Link is down Latches '0' if SGMII Link Status goes down. Clears to current SGMII Link Status on read. See the following Link Status section for further details.	read only self clearing on read	0
1.1	Jabber Detect	Always returns a '0' for this bit since Jabber Detect is not supported	returns 0	0
1.0	Extended Capability	Always returns a '0' for this bit because no extended register set is supported	returns 0	0

Link Status

When high, the link is valid and has remained valid since this register was last read: synchronization of the link has been obtained.

When low, either:

- a valid link has not been established; link synchronization has failed.
- OR, link synchronization was lost at some point since this register was previously read. However, the current link status may be good. **Therefore read this register a second time to get confirmation of the current link status**.

Registers 2 and 3: PHY Identifier

MDIO Registers 2 and 3: PHY Identifier

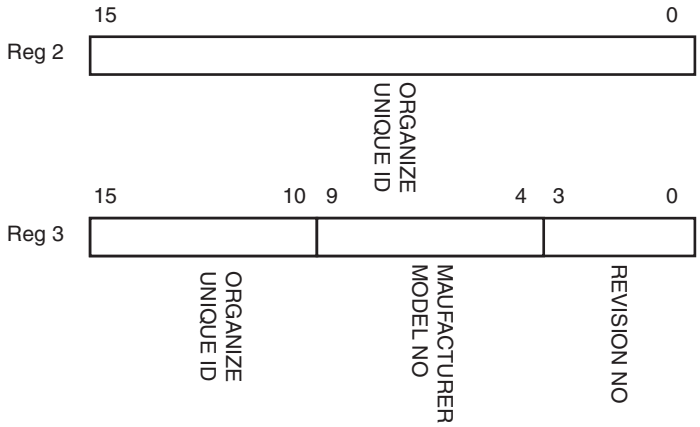


Table 9-32: **PHY Identifier (Registers 2 and 3)**

Bit(s)	Name	Description	Attributes	Default Value
2.15:0	Organizationally Unique Identifier	Always return 0s	returns 0s	0000000000000000
3.15:10	Organizationally Unique Identifier	Always return 0s	returns 0s	000000
3.9:4	Manufacturer model number	Always return 0s	returns 0s	000000
3.3:0	Revision Number	Always return 0s	returns 0s	0000

Register 4: SGMII Auto-Negotiation Advertisement

MDIO Register 4: SGMII Auto-Negotiation Advertisement

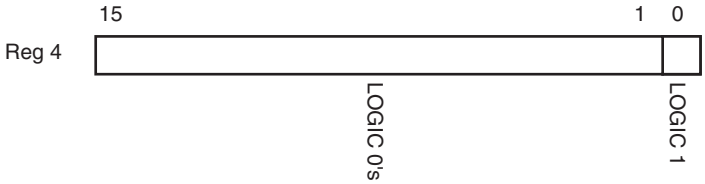


Table 9-33: **SGMII Auto-Negotiation Advertisement (Register 4)**

Bit(s)	Name	Description	Attributes	Default Value
4.15:0	All bits	Ignore this register because Auto-Negotiation is not included	read only	0000000000000001

Register 15: SGMII Extended Status

MDIO Register 15: SGMII Extended Status

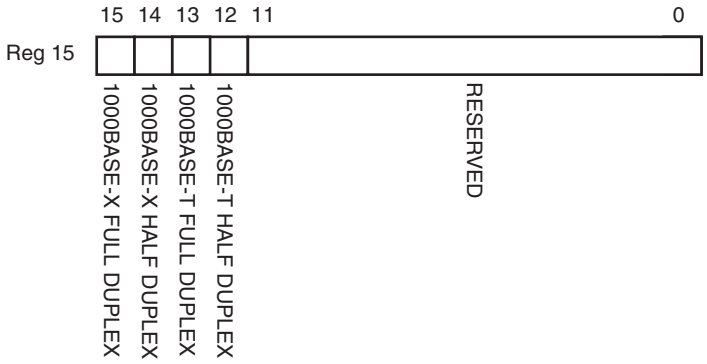


Table 9-34: **SGMII Extended Status Register (Register 15)**

Bit(s)	Name	Description	Attributes	Default Value
15.15	1000BASE-X Full Duplex	Always returns a '1' for this bit since 1000BASE-X Full Duplex is supported	returns 1	1
15.14	1000BASE-X Half Duplex	Always returns a '0' for this bit since 1000BASE-X Half Duplex is not supported	returns 0	0
15.13	1000BASE-T Full Duplex	Always returns a '0' for this bit since 1000BASE-T Full Duplex is not supported	returns 0	0
15.12	1000BASE-T Half Duplex	Always returns a '0' for this bit since 1000BASE-T Half Duplex is not supported	returns 0	0
15:11:0	Reserved	Always return 0s	returns 0s	000000000000

Both 1000BASE-X and SGMII Standards

Table 9-35 describes register 17, the vendor-specific Standard Selection Register. This register is only present when the core is generated with the capability to dynamically switch between 1000BASE-X and SGMII standards. See "Select Standard" in Chapter 3.

When this Register is configured to perform the 1000BASE-X standard, Registers 0 to 16 should be interpreted as per "1000BASE-X Standard Using the Optional Auto-Negotiation," or "1000BASE-X Standard Without the Optional Auto-Negotiation."

When this Register is configured to perform the SGMII standard, Registers 0 to 16 should be interpreted as per "SGMII Standard Using the Optional Auto-Negotiation," or "SGMII Standard without the Optional Auto-Negotiation." This register may be written to at any time. See Chapter 11, "Dynamic Switching of 1000BASE-X and SGMII Standards" for more information.

Register 17: Vendor-Specific Standard Selection Register

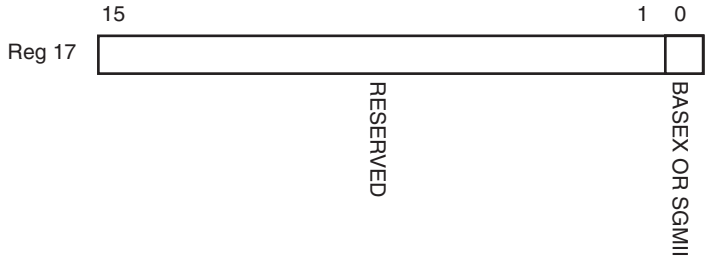


Figure 9-5: **Dynamic Switching (Register 17)**

Table 9-35: **Vendor-specific Register: Standard Selection Register (Register 17)**

Bit(s)	Name	Description	Attributes	Default Value
17.15:1	Reserved	Always return 0s	Returns 0s	000000000000000
16.0	Standard	0 = Core will perform the 1000BASE-X standard. Registers 0 to 16 will behave as per "1000BASE-X Standard Using the Optional Auto-Negotiation" 1= Core will perform the SGMII standard. Registers 0 to 16 will behave as per "SGMII Standard Using the Optional Auto-Negotiation".	read/write	Determined by the basex_or_sgmii port

Optional Configuration Vector

If "MDIO Management Interface" is omitted, relevant configuration signals are brought out of the core. These signals are bundled into the CONFIGURATION_VECTOR signal as defined in Table 9-36.

These signals may be changed by the user application at any time. The *Clock Domain* heading denotes the clock domain the configuration signal is registered in before use by the core. It is not necessary to drive the signal from this clock domain.

Table 9-36: **Optional Configuration and Status Vectors**

Signal	Direction	Clock Domain	Description
configuration_vector [3:0]	Input	See note 1	Bit[0]: Reserved (currently unused) Bit[1]: **Loopback Control** • When used with a device-specific transceiver, the core is placed in internal loopback mode. • With the TBI version, Bit 1 is connected to ewrap. When set to '1,' this indicates to the external PMA module to enter loopback mode. See "Loopback," page 234. Bit[2]: **Power Down** • When a device-specific transceiver is used, a setting of '1' places the device-specific transceiver in a low-power state. A reset must be applied to clear. • With the TBI version, this bit is unused. Bit[3]: **Isolate** • When set to '1,' the GMII should be electrically isolated. • When set to '0,' normal operation is enabled.

1. Signals are synchronous to the internal 125 MHz reference clock of the core; this is userclk2 when used with a device-specific transceiver; gtx_clk when used with TBI.

Chapter 10

Auto-Negotiation

This chapter provides general guidelines for using the Auto-Negotiation function of the Ethernet 1000BASE-X PCS/PMA or SGMII core. Auto-Negotiation is controlled and monitored through the PCS Management Registers and is only available when the optional MDIO Management Interface is present. For more information, see Chapter 9, "Configuration and Status."

Overview of Operation

For either standard, when considering Auto-Negotiation between two connected devices, it must be remembered that:

- Auto-Negotiation must be either enabled in **both** devices, or:
- Auto-Negotiation must be disabled in **both** devices.

1000BASE-X Standard

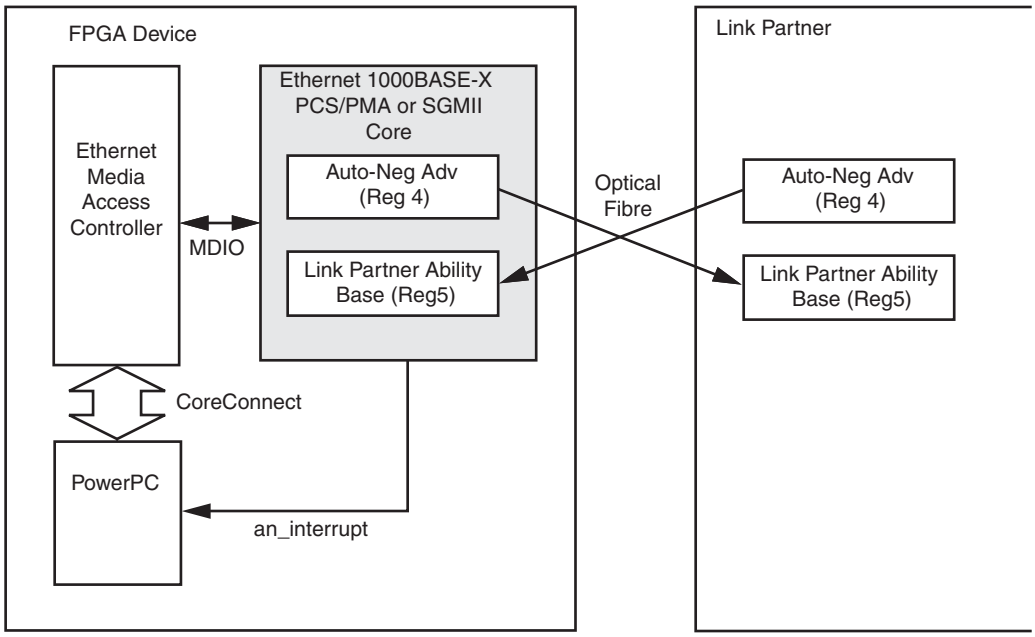


Figure 10-1: **1000BASE-X Auto-Negotiation Overview**

IEEE 802.3 clause 37 describes the 1000BASE-X Auto-Negotiation function that allows a device to advertise the modes of operation that it supports to a device at the remote end of a link segment (the link partner) and to detect corresponding operational modes that the link partner advertises. Figure 10-1 illustrates the operation of 1000BASE-X Auto-Negotiation.

The following describes typical operation when Auto-Negotiation is enabled.

1. Auto-Negotiation starts automatically when any of the following conditions are met.
   - Power-up/reset
   - Upon loss of synchronization
   - The link partner initiates Auto-Negotiation
   - An Auto-Negotiation Restart is requested (See "Control Register (Register 0)," page 145.)
2. During Auto-Negotiation, the contents of the Auto-Negotiation Advertisement Register are transferred to the link partner.

   This register is writable through the MDIO, therefore enabling software control of the systems advertised abilities. See "Auto-Negotiation Advertisement Register (Register 4)," page 149 for more information.

   Information provided in this register includes:

   - Fault Condition signaling
   - Duplex Mode
   - Flow Control capabilities for the attached MAC.
3. The advertised abilities of the Link Partner are simultaneously transferred into the Auto-Negotiation Link Partner Ability Base Register.

   This register contains the same information as in the Auto-Negotiation Advertisement Register. See "Auto-Negotiation Link Partner Ability Base Register (Register 5)," page 150 for more information.
4. Under normal conditions, this completes the Auto-Negotiation information exchange.

   It is now the responsibility of system management (for example, software running on an embedded PowerPC® or MicroBlaze™ processor) to complete the cycle. The results of the Auto-Negotiation should be read from Auto-Negotiation Link Partner Ability Base Register. Other networking components, such as an attached Ethernet MAC, should be configured accordingly. See "Auto-Negotiation Link Partner Ability Base Register (Register 5)" for more information.

   There are two methods that a host processor uses to learn of the competition of an Auto-Negotiation cycle:

   - Polling the Auto-Negotiation completion bit 1.5 in the Status Register (Register 1).
   - Using the Auto-Negotiation interrupt port of the core (see "Using the Auto-Negotiation Interrupt," page 184).

SGMII Standard

Figure 10-2 illustrates the operation of SGMII Auto-Negotiation. Additional information about SGMII Standard Auto-Negotiation is provided in the following sections.

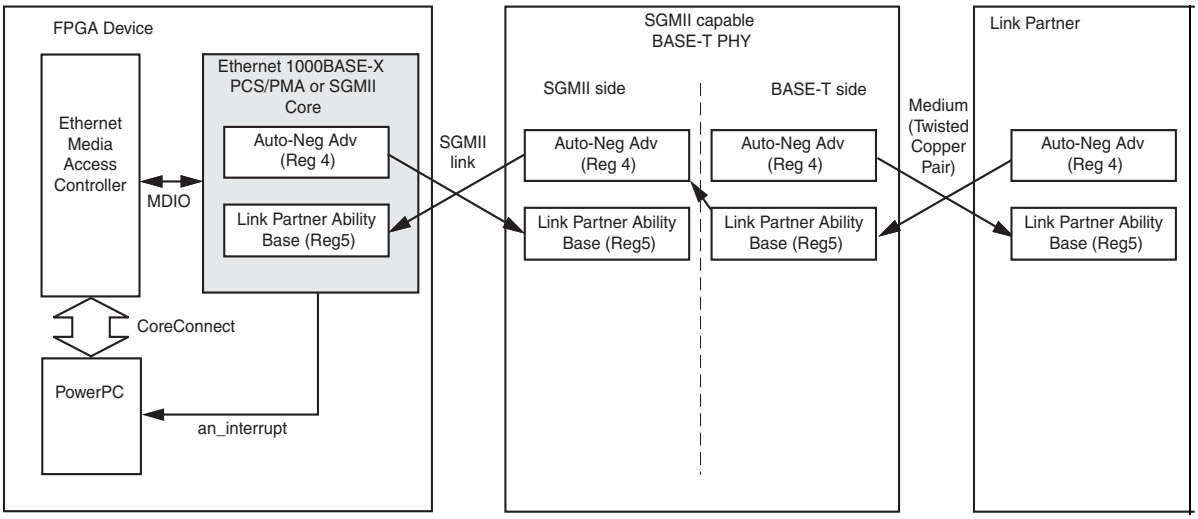


Figure 10-2: **SGMII Auto-Negotiation**

The SGMII capable PHY has two distinctive sides to Auto-Negotiation.

- The PHY performs Auto-Negotiation with its link partner using the relevant Auto-Negotiation standard for the chosen medium (BASE-T Auto-Negotiation is illustrated in Figure 10-2, using a twisted copper pair as its medium). This resolves the operational speed and duplex mode with the link partner.
- The PHY then passes the results of the Auto-Negotiation process with the link partner to the Ethernet 1000BASE-X PCS/PMA or SGMII core (in SGMII mode), by leveraging the 1000BASE-X Auto-Negotiation specification described in "1000BASE-X Auto-Negotiation Overview," page 181. This transfers the results of the Link Partner Auto-Negotiation across the SGMII and is the only Auto-Negotiation observed by the core.

This SGMII Auto-Negotiation function, summarized previously, leverages the 1000BASE-X PCS/PMA Auto-Negotiation function but contains two differences.

- The duration of the Link Timer of the SGMII Auto-Negotiation is shrunk from 10 ms to 1.6 ms so that the entire Auto-Negotiation cycle is much faster. See "Setting the Configurable Link Timer," page 184.
- The information exchanged is different and now contains speed resolution in addition to duplex mode. See "MDIO Register 5: SGMII Auto-Negotiation Link Partner Ability," page 166.

There are no other differences and dealing with the results of Auto-Negotiation can be handled as described previously in "1000BASE-X Auto-Negotiation Overview."

Setting the Configurable Link Timer

The optional Auto-Negotiation function has a Link Timer (`link_timer[8:0]`) port. This port sets the period of the Auto-Negotiation Link Timer. This port should be permanently tied to a logical binary value, and a binary value should be placed on this port. The duration of the timer is approximately equal to the binary value multiplied by 32.768 microseconds (4,96 clock periods of the 125 MHz clock provided to the core). See "Auto-Negotiation Signal Pinout" in Chapter 2.

Note: See Chapter 11, "Dynamic Switching of 1000BASE-X and SGMII Standards" for details of programming the Auto-Negotiation Link Timer when performing dynamic switching between 1000BASE-X and SGMI Standards.

The accuracy of this Link Timer is within the following range.

```
+0 to -32.768 microseconds
```

1000BASE-X Standard

The Link-Timer is defined as having a duration somewhere between 10 and 20 milliseconds. The example design delivered with the core sets the binary value as follows:

```
100111101 = 317 decimal
```

This corresponds to a duration of between 10.354 and 10.387 milliseconds.

SGMII Standard

The Link-Timer is defined as having a duration of 1.6 milliseconds. The example design delivered with the core sets the binary value to

```
000110010 = 50 decimal
```

This corresponds to a duration of between 1.606 and 1.638 milliseconds.

Simulating Auto-Negotiation

Auto-Negotiation requires a minimum of three link timer periods for completion. If simulating the Auto-Negotiation procedure, setting the `link_timer[8:0]` port to a low value will greatly reduce the simulation time required to complete Auto-Negotiation.

Using the Auto-Negotiation Interrupt

The Auto-Negotiation function has an `an_interrupt` port. This is designed to be used with common microprocessor bus architectures (for example, the CoreConnect bus interfacing to a MicroBlaze processor or the Virtex®-5 FXT FPGA embedded IBM PowerPC processor). For more information, see "Auto-Negotiation Signal Pinout" in Chapter 2.

The operation of this port is enabled or disabled and cleared via the MDIO Register 16, the Vendor-specific Auto-Negotiation Interrupt Control Register.

- When disabled, this port is permanently tied to logic 0.
- When enabled, this port will be set to logic 1 following the completion of an Auto-Negotiation cycle. It will remain high until it is cleared by writing 0 to bit 16.1 (Interrupt Status bit) of the "MDIO Register 16: Vendor Specific Auto-Negotiation Interrupt Control," page 155.

Chapter 11

Dynamic Switching of 1000BASE-X and SGMII Standards

This chapter provides general guidelines for using the core to perform dynamic standards switching between 1000BASE-X and SGMII. The core only provides this capability if generated with the appropriate option, as described in Chapter 3, "Generating and Customizing the Core."

Typical Application

Figure 11-1 illustrates a typical application for the Ethernet 1000BASE-X PCS/PMA or SGMII core with the ability to dynamically switch between 1000BASE-X and SGMII standards.

The FPGA is shown connected to an external, off-the-shelf PHY with the ability to perform both BASE-X and BASE-T standards.

- The core must operate in 1000BASE-X mode to use the optical fibre.
- The core must operate in SGMII mode to provide BASE-T functionality and use the twisted copper pair.

The GMII of the Ethernet 1000BASE-X PCS/PMA or SGMII core is shown connected to an embedded Ethernet Media Access Controller (MAC), for example the Tri-Mode Ethernet MAC core from Xilinx.

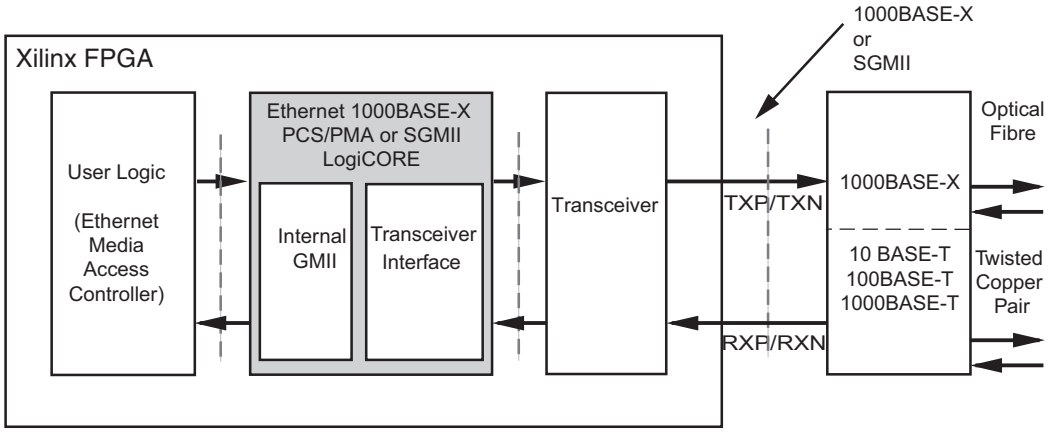


Figure 11-1: **Typical Application for Dynamic Switching**

Operation of the Core

Selecting the Power-On / Reset Standard

The external port of the core, `basex_or_sgmii` (see "Dynamic Switching Signal Pinout" in Chapter 2), will select the default standard of the core as follows:

- Tie to logic '0' in the core instantiation. The core powers-up and comes out of a reset cycle operating in the 1000BASE-X standard.
- Tie to logic '1' in the core instantiation. The core powers-up and comes out of a reset cycle operating in the SGMII standard.

The `basex_or_sgmii` port of the core can be dynamically driven. In this configuration, it is possible to drive a logical value onto the port, followed by a core reset cycle to switch the core to the desired standard. However, it is expected that the standard will be switched through the MDIO Management Registers.

Switching the Standard Using MDIO

The 1000BASE-X or SGMII standard of the core can be switched at any time by writing to the "Vendor-specific Register: Standard Selection Register (Register 17)." Following completion of this write, the MDIO Management Registers will immediately switch.

- Core set to 1000BASE-X standard. Management Registers 0 through 16 should be interpreted according to "1000BASE-X Standard Using the Optional Auto-Negotiation," page 144.
- Core set to SGMII standard. Management Registers 0 through 16 should be interpreted according to "SGMII Standard Using the Optional Auto-Negotiation," page 161.

Auto-Negotiation State Machine

- Core set to the 1000BASE-X standard. The Auto-Negotiation state machine operates as described in "1000BASE-X Standard," page 184.
- Core set to perform the SGMII standard. The Auto-Negotiation state machine operates as described in "SGMII Standard," page 184.
- Standard is switched during an Auto-Negotiation sequence. The Auto-Negotiation state machine will not immediately switch standards, but attempt to continue to completion at the original standard.
- Switching the standard using MDIO. This does not cause Auto-Negotiation to automatically restart. Xilinx recommends that after switching to a new standard using a MDIO write, immediately perform the following:
  - If you have switched to the 1000BASE-X standard, reprogram the Auto-Negotiation Advertisement Register (Register 4) to the desired settings.
  - For either standard, restart the Auto-Negotiation sequence by writing to bit 0.9 of the MDIO Control Register (Register 0).

Setting the Auto-Negotiation Link Timer

As described in "Auto-Negotiation" in Chapter 10, the duration of the Auto-Negotiation Link Timer differs with the 1000BASE-X and the SGMII standards. To provide configurable link timer durations for both standards, the following ports are available. These ports replace the `link_timer_value[8:0]` port that is used when the core is generated for a single standard.

- **`link_timer_basex[8:0]`** The value placed on this port is sampled at the beginning of the Auto-Negotiation cycle by the Link Timer when the core is set to perform the 1000BASE-X standard.
- **`link_timer_sgmii[8:0]`** The value placed on this port is sampled at the beginning of the Auto-Negotiation cycle by the Link Timer when the core is set to perform the SGMII standard.

Both ports follow the same rules that are described in "Setting the Configurable Link Timer," page 184.

XILINX®

Chapter 12

Constraining the Core

This chapter defines the constraint requirements of the Ethernet 1000BASE-X PCS/PMA or SGMII core. An example UCF is provided with the HDL example design for the core to implement the constraints defined in this chapter.

See the *Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started Guide* for a complete description of the CORE Generator™ software output files and for details on the HDL example design.

Required Constraints

Device, Package, and Speedgrade Selection

The Ethernet 1000BASE-X PCS/PMA or SGMII core can be implemented in Virtex®-4, Virtex-5, Spartan®-6, Spartan-3, Spartan-3E, Spartan-3A/3AN and Spartan-3 DSP devices. When selecting a device, be aware of the following considerations:

- Device must be large enough to accommodate the core.
- Device must contain a sufficient number of IOBs.
- –4 speed grade for Spartan-3, Spartan-3E, Spartan-3A/3AN/3A DSP devices
- –10 speed grade for Virtex-4 devices
- -1 speed grade for Virtex-5 and Virtex-6 devices
- -2 speed grade for Spartan-6 devices
- The transceiver is only supported in Virtex-4 FX, Virtex-5 LXT, Virtex-5 SXT, and Virtex-5 FXT and TXT FPGAs, Spartan-6 LXT and Virtex-6 devices.

I/O Location Constraints

No specific I/O location constraints required.

However, when employing BUFIO and BUFR regional clock routing (Virtex-5, Virtex-6 and Spartan-6 devices), please ensure that a BUFIO capable clock input pin is selected for input clock sources, and that all related input synchronous data signals are placed in the respective BUFIO region. The device User Guide should be consulted.

Placement Constraints

No specific placement constraints required.

Virtex-4 FPGA MGT Transceivers for 1000BASE-X Constraints

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from the UCF are copied into the following descriptions to serve as examples and should be studied in conjunction with the HDL source code for the example design. See also "Virtex-4 FX Devices" in Chapter 7.

Clock Period Constraints

The clock `txoutclk` is provided by the MGT for use in the FPGA fabric. It is connected to global clock routing to produce the `usrclk2` signal. This is the main 125 MHz clock used by all core logic and must be constrained.

`DCLK` is a clock with a frequency between 25 and 50 MHz, which must be provided to the Dynamic Reconfiguration Port and to the calibration block of the MGT. In the example design, this is constrained to 50 MHz.

The following UCF syntax shows these constraints being applied.

```
#************************************************************
# PCS/PMA Clock period Constraints: please do not relax     *
#************************************************************

NET "userclk2" TNM_NET = "userclk2";
TIMESPEC "TS_userclk2" = PERIOD "userclk2" 8 ns HIGH 50 %;

NET "dclk" TNM_NET = "dclk";
TIMESPEC "TS_dclk" = PERIOD "dclk" 20 ns HIGH 50 %;
```

Setting MGT Transceiver Attributes

The Virtex-4 FPGA MGT device has many attributes. These attributes are set directly from HDL source code for the transceiver wrapper file delivered with the example design. These are in the file `transceiver.vhd` (for VHDL design entry) or `transceiver.v` (for Verilog design entry). See the *Ethernet 1000BASE-X PCS/PMA or SGMII Getting Started Guide* for a detailed description of the example design provided with the core.

This HDL transceiver wrapper file was initially created using Architecture Wizard. See the *Virtex-4 RocketIO Multi-Gigabit Transceiver User Guide* (UG076) for a description of available attributes.

MGT Placement Constraints

The following UCF syntax illustrates the MGT placement contraints for the example design. Because Virtex-4 FPGA MGTs are always available in pairs, two MGTs are always instantiated in the example design, even if one is inactive.

```
#***********************************************************
# Example Rocket I/O placement                             *
#***********************************************************
# Lock down the REFCLK pins:
NET  brefclk_p   LOC = F26;
NET  brefclk_n   LOC = G26;

# Lock down the GT11 pair and GT11 clock module
INST "core_wrapper/RocketIO/GT11_1000X_A" LOC = GT11_X0Y5;
INST "core_wrapper/RocketIO/GT11_1000X_B" LOC = GT11_X0Y4;
INST "GT11CLK_MGT_INST" LOC = GT11CLK_X0Y3;

# Lock down the RocketIO pins:
NET  "rxp0" LOC = J26;
NET  "rxn0" LOC = K26;
NET  "txp0" LOC = M26;
NET  "txn0" LOC = N26;
NET  "rxp1" LOC = U26;
NET  "rxn1" LOC = V26;
NET  "txp1" LOC = P26;
NET  "txn1" LOC = R26;
```

Virtex-4 FPGA RocketIO MGT Transceivers for SGMII or Dynamic Standards Switching Constraints

All the constraints described in the section "Virtex-4 FPGA MGT Transceivers for 1000BASE-X Constraints." In addition, if the FPGA Fabric Rx Elastic Buffer is selected, an extra clock period constraint of 16 ns is required for `rxrecclk1`.

With the MGT Rx Elastic Buffer bypassed, `rxrecclk1` is provided by the MGT to the FPGA fabric for the recovered receiver data signals leaving the transceiver. This data is then written into the replacement Rx Elastic Buffer implemented in the FPGA fabric. See "Virtex-4 Devices for SGMII or Dynamic Standards Switching," page 119.

The following UCF syntax shows the necessary constraint being applied to GT11 A.

```
#************************************************************
# PCS/PMA Clock period Constraints for the GT11 A           *
# recovered clock: please do not relax                      *
#************************************************************

NET "core_wrapper/RocketIO/rxrecclk10" TNM_NET = "rxrecclk10";
TIMESPEC "ts_rxrecclk10" = PERIOD "rxrecclk10" 16 ns;
```

Virtex-5 FPGA RocketIO GTP Transceivers for 1000BASE-X Constraints

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from the UCF are copied into the following descriptions to serve as examples, and should be studied with the HDL source code for the example design. See also "Virtex-5 LXT and SXT Devices" in Chapter 7.

Clock Period Constraints

The `clkin` clock is provided to the GTP transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained.

The `refclkout` clock is provided by the GTP for use in the FPGA fabric, which is then connected to global clock routing to produce the `usrclk2` signal. This is the main 125 MHz clock used by all core logic and must be constrained.

The following UCF syntax shows these constraints being applied.

```
#************************************************************
# PCS/PMA Clock period Constraints: please do not relax     *
#************************************************************

NET "*clkin" TNM_NET = "clkin";
TIMESPEC "TS_clkin" = PERIOD "clkin" 8 ns HIGH 50 %;

NET "*refclkout" TNM_NET = "refclkout";
TIMESPEC "TS_refclkout" = PERIOD "refclkout" 8 ns HIGH 50 %;
```

Setting GTP Transceiver Attributes

The Virtex-5 FPGA RocketIO™ GTP transceiver has many attributes that are set directly from HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the `RocketIO_wrapper_gtp_tile.vhd` file (for VHDL design entry) or the `RocketIO_wrapper_gtp_tile.v` file (for Verilog design entry); these files were generated using the GTP Transceiver Wizard - to change the attributes, re-run the Wizard. See "Virtex-5 FPGA RocketIO GTP Transceiver Wizard" in Chapter 7.

Virtex-5 FPGA RocketIO GTP Transceivers for SGMII or Dynamic Standards Switching Constraints

If the core is generated to use the GTP Rx Elastic Buffer, all of the constraints apply, as defined in "Virtex-5 FPGA RocketIO GTP Transceivers for 1000BASE-X Constraints". However, if the FPGA Fabric Rx Elastic Buffer is selected, an extra clock period constraint of 8 ns is required for `rxrecclk`: with the GTP Rx Elastic Buffer bypassed, `rxrecclk` is provided by the GTP transceiver to the FPGA fabric for the recovered receiver data signals leaving the transceiver. This data is then written into the replacement Rx Elastic Buffer implemented in the FPGA fabric. See "Virtex-5 LXT or SXT Devices for SGMII or Dynamic Standards Switching," page 122 for more information about this logic.

The following UCF syntax shows the necessary constraint being applied to the `rxrecclk` signal sourced from GTP 0.

```
#*********************************************************
# PCS/PMA Clock period Constraints for the GTP 0          *
# recovered clock: please do not relax                    *
#*********************************************************

NET "core_wrapper/RocketIO/rxrecclk0" TNM_NET = "rxrecclk0";
TIMESPEC "ts_rxrecclk0" = PERIOD "rxrecclk0" 8 ns;
```

Setting GTP Transceiver Attributes

Additionally, if the FPGA Fabric Rx Elastic Buffer is selected, then the attributes of the Virtex-5 FPGA RocketIO GTP transceiver which are set directly from HDL source code do differ from the standard case. These can be found in the `RocketIO_wrapper_gtp_tile.vhd` file (for VHDL design entry) or the `RocketIO_wrapper_gtp_tile.v` file (for Verilog design entry); these files were generated using the GTP RocketIO Wizard - to change the attributes, re-run the Wizard. See "Virtex-5 FPGA RocketIO Transceiver GTP Wizard" in Chapter 8.

Virtex-5 FPGA RocketIO GTX Transceivers for 1000BASE-X Constraints

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from the UCF are copied into the following descriptions to serve as examples, and should be studied with the HDL source code for the example design. See also "Virtex-5 FXT and TXT Devices" in Chapter 7.

Clock Period Constraints

The `clkin` clock is provided to the GTX transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained.

The `refclkout` clock is provided by the GTX for use in the FPGA fabric–this is the main 125 MHz clock reference source for the FPGA fabric and should be constrained. This is then connected to a DCM. The ports `CLK0` (125 MHz) and `CLKDV` (62.5 MHz) of this DCM are then placed onto global clock routing to produce the `usrclk2` and `usrclk` clock signals respectively. The Xilinx tools will trace the `refclkout` constraint through the DCM and automatically generate clock period constraints for the DCM output clocks. So constraints `usrclk2` and `usrclk` do not need to be manually applied.

The following UCF syntax shows these constraints being applied.

```
#***********************************************************
# PCS/PMA Clock period Constraints: please do not relax    *
#***********************************************************

NET "*clkin" TNM_NET = "clkin";
TIMESPEC "TS_clkin" = PERIOD "clkin" 8 ns HIGH 50 %;

NET "*refclkout" TNM_NET = "refclkout";
TIMESPEC "TS_refclkout" = PERIOD "refclkout" 8 ns HIGH 50 %;
```

Setting GTX Transceiver Attributes

The Virtex-5 FPGA RocketIO GTX transceiver has many attributes that are set directly from HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the `RocketIO_wrapper_gtx_tile.vhd` file (for VHDL design entry) or the `RocketIO_wrapper_gtx_tile.v` file (for Verilog design entry): these files were generated using the GTX Transceiver Wizard - to change the attributes, re-run the Wizard. See "Virtex-5 FPGA RocketIO GTX Wizard" in Chapter 7.

Virtex-5 FPGA RocketIO GTX Transceivers for SGMII or Dynamic Standards Switching Constraints

If the core is generated to use the GTX Rx Elastic Buffer, then all of the constraints documented in "Virtex-5 FPGA RocketIO GTX Transceivers for 1000BASE-X Constraints", apply.

However, if the FPGA Fabric Rx Elastic Buffer is selected, then an extra clock period constraint of 16 ns is required for `rxrecclk`: with the GTX Rx Elastic Buffer bypassed, `rxrecclk` is provided by the GTX transceiver to the FPGA fabric for the recovered receiver data signals leaving the transceiver. This data is then written into the replacement Rx Elastic Buffer implemented in the FPGA fabric. See "Virtex-5 FXT and TXT Devices for SGMII or Dynamic Standards Switching," page 124 for more information about this logic.

The following UCF syntax shows the necessary constraint being applied to the `rxrecclk` signal sourced from GTX 0.

```
#***********************************************************
# PCS/PMA Clock period Constraints for the GTP/X 0          *
# recovered clock: please do not relax                      *
#***********************************************************

NET "core_wrapper/RocketIO/rxrecclk0" TNM_NET = "rxrecclk0";
TIMESPEC "ts_rxrecclk0" = PERIOD "rxrecclk0" 16 ns;
```

Setting GTX Transceiver Attributes

Additionally, if the FPGA Fabric Rx Elastic Buffer is selected, then the attributes of the Virtex-5 FPGA RocketIO GTX transceiver which are set directly from HDL source code do differ from the standard case. These can be found in the `RocketIO_wrapper_gtx_tile.vhd` file (for VHDL design entry) or the `RocketIO_wrapper_gtx_tile.v` file (for Verilog design entry); these files were generated using the GTX RocketIO Wizard - to change the attributes, re-run the Wizard. See "Virtex-5 FPGA RocketIO GTX Wizard" in Chapter 8.

Virtex-6 FPGA GTX Transceivers for 1000BASE-X Constraints

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from the UCF are copied into the following descriptions to serve as examples, and should be studied with the HDL source code for the example design. See also "Virtex-6 Devices" in Chapter 7.

Clock Period Constraints

The `mgtrefclk` clock is provided to the GTX transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained.

The `txoutclk` clock is provided by the GTX for use in the FPGA fabric, which is then connected to global clock routing to produce the `usrclk2` signal. This is the main 125 MHz clock used by all core logic and must be constrained.

The following UCF syntax shows these constraints being applied.

```
#***********************************************************
# PCS/PMA Clock period Constraints: please do not relax    *
#***********************************************************

NET "mgtrefclk" TNM_NET = "mgtrefclk";
TIMESPEC "ts_mgtrefclk" = PERIOD "mgtrefclk" 8 ns HIGH 50 %;

NET "*txoutclk" TNM_NET = "txoutclk";
TIMESPEC "TS_txoutclk" = PERIOD "txoutclk" 8 ns HIGH 50 %;
```

Setting Virtex-6 FPGA GTX Transceiver Attributes

The Virtex-6 FPGA GTX transceiver has many attributes that are set directly from HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the `gtx_wrapper_gtx.vhd` file (for VHDL design entry) or the `gtx_wrapper_gtx.v` file (for Verilog design entry); these files were generated using the Virtex-6 FPGA GTX Transceiver Wizard - to change the attributes, re-run the Wizard. See "Virtex-6 FPGA GTX Transceiver Wizard" in Chapter 7.

Virtex-6 FPGA GTX Transceivers for SGMII or Dynamic Standards Switching Constraints

If the core is generated to use the Virtex-6 FPGA GTX Rx Elastic Buffer, all of the constraints apply, as defined in "Virtex-6 FPGA GTX Transceivers for 1000BASE-X Constraints". However, if the FPGA Fabric Rx Elastic Buffer is selected, an extra clock period constraint of 8 ns is required for `rxrecclk`: with the GTX Rx Elastic Buffer unused, `RXRECCLK` is provided by the GTX transceiver to the FPGA fabric for the recovered receiver data signals leaving the transceiver. This data is then written into the replacement Rx Elastic Buffer implemented in the FPGA fabric. See "Virtex-6 Devices for SGMII or Dynamic Standards Switching," page 126 for more information about this logic.

The following UCF syntax shows the necessary constraint being applied to the `RXRECCLK` signal sourced from the GTX.

```
#***********************************************************
# PCS/PMA Clock period Constraints for the GTP 0           *
# recovered clock: please do not relax                     *
#***********************************************************

NET "core_wrapper/gtx/RXRECCLK" TNM_NET = "rxrecclk";
TIMESPEC "ts_rxrecclk" = PERIOD "rxrecclk" 8 ns;
```

Setting Virtex-6 FPGA GTX Transceiver Attributes

Additionally, if the FPGA Fabric Rx Elastic Buffer is selected, then the attributes of the Virtex-6 FPGA GTX transceiver, which are set directly from HDL source code, do differ from the standard case. These can be found in the `gtx_wrapper_gtx.vhd` file (for VHDL design entry) or the `gtx_wrapper_gtx.v` file (for Verilog design entry): these files were generated using the Virtex-6 FPGA GTX Wizard - to change the attributes, re-run the Wizard. See "Virtex-6 FPGA GTX Transceiver Wizard" in Chapter 8.

Spartan-6 FPGA GTP Transceivers for 1000BASE-X Constraints

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from the UCF are copied into the following descriptions to serve as examples, and should be studied with the HDL source code for the example design. See also "Spartan-6 LXT Devices" in Chapter 7.

Clock Period Constraints

The `clkin` clock is provided to the GTP transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained.

The `refclkout` clock is provided by the GTP for use in the FPGA fabric, which is then connected to global clock routing to produce the `usrclk2` signal. This is the main 125 MHz clock used by all core logic and must be constrained.

The following UCF syntax shows these constraints being applied.

```
#***********************************************************
# PCS/PMA Clock period Constraints: please do not relax    *
#***********************************************************

NET "*clkin" TNM_NET = "clkin";
TIMESPEC "TS_clkin" = PERIOD "clkin" 8 ns HIGH 50 %;

NET "*gtpclkout" TNM_NET = "gtpclkout";
TIMESPEC "TS_gtpclkout" = PERIOD "gtpclkout" 8 ns HIGH 50 %;
```

Setting Spartan-6 FPGA GTP Transceiver Attributes

The Spartan-6 FPGA GTP transceiver has many attributes that are set directly from HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the `gtp_wrapper_tile.vhd` file (for VHDL design entry) or the `gtp_wrapper_tile.v` file (for Verilog design entry): these files were generated using the Spartan-6 FPGA GTP Transceiver Wizard. To change the attributes, re-run the Wizard. See "Spartan-6 FPGA GTP Transceiver Wizard" in Chapter 7.

Spartan-6 FPGA GTP Transceivers for SGMII or Dynamic Standards Switching Constraints

If the core is generated to use the GTP Rx Elastic Buffer, all of the constraints apply, as defined in "Spartan-6 FPGA GTP Transceivers for 1000BASE-X Constraints". However, if the FPGA Fabric Rx Elastic Buffer is selected, an extra clock period constraint of 8 ns is required for `rxrecclk`: with the GTP Rx Elastic Buffer bypassed, `rxrecclk` is provided by the GTP transceiver to the FPGA fabric for the recovered receiver data signals leaving the transceiver. This data is then written into the replacement Rx Elastic Buffer implemented in the FPGA fabric. See "Spartan-6 LXT Devices for SGMII or Dynamic Standards Switching," page 128 for more information about this logic.

The following UCF syntax shows the necessary constraint being applied to the `rxrecclk` signal sourced from GTP 0.

```
#*********************************************************
# PCS/PMA Clock period Constraints for the GTP 0          *
# recovered clock: please do not relax                    *
#*********************************************************

NET "core_wrapper/gtp/rxrecclk0" TNM_NET = "rxrecclk0";
TIMESPEC "ts_rxrecclk0" = PERIOD "rxrecclk0" 8 ns;
```

Setting Spartan-6 FPGA GTP Transceiver Attributes

Additionally, if the FPGA Fabric Rx Elastic Buffer is selected, then the attributes of the Virtex-5 FPGA GTP transceiver which are set directly from HDL source code do differ from the standard case. These can be found in the `gtp_wrapper_tile.vhd` file (for VHDL design entry) or the `gtp_wrapper_tile.v` file (for Verilog design entry): these files were generated using the Spartan-6 FPGA GTP Wizard. To change the attributes, re-run the Wizard. See "Spartan-6 FPGA Transceiver GTP Wizard" in Chapter 8.

Ten-Bit Interface Constraints

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from this UCF have been copied into the descriptions in this section to serve as examples, and should be studied with the HDL source code for the example design. See also Chapter 6, "The Ten-Bit Interface."

Clock Period Constraints

The clocks provided to `pma_rx_clk0` and `pma_rx_clk1` must be constrained for a clock frequency of 62.5 MHz. The clock provided to `gtx_clk` must be constrained for a clock frequency of 125 MHz. The following UCF syntax shows the constraints being applied to the example design.

```
############################################################
# TBI Clock period Constraints: please do not relax        #
############################################################
NET "pma_rx_clk0" TNM_NET = "pma_rx_clk0";
TIMESPEC "ts_pma_rx_clk0" = PERIOD "pma_rx_clk0" 16000 ps HIGH 50 %;

NET "pma_rx_clk1" TNM_NET = "pma_rx_clk1";
TIMESPEC "ts_pma_rx_clk1" = PERIOD "pma_rx_clk1" 16000 ps HIGH 50 %;
NET "gtx_clk_bufg" TNM_NET = "clk_tx";
TIMESPEC "ts_tx_clk" = PERIOD "clk_tx" 8000 ps HIGH 50 %;
```

Period constraints should be applied to cover signals in to and out of the block memory based 8B/10B encoder and decoder.

```
# Constrain between flip-flops and the Block Memory for the 8B10B
encoder and decoder
INST "gig_eth_pcs_pma_core/BU2/U0/PCS_OUTPUT/DECODER/LOOK_UP_TABLE"
TNM = "codec8b10b";
INST "gig_eth_pcs_pma_core/BU2/U0/PCS_OUTPUT/ENCODER/LOOK_UP_TABLE"
TNM = "codec8b10b";
TIMESPEC "ts_ffs_to_codec8b10b" = FROM FFS TO "codec8b10b" 8000 ps;
TIMESPEC "ts_codec8b10b_to_ffs" = FROM "codec8b10b" TO FFS 8000 ps;
```

Ten-Bit Interface 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 "tx_code_group_reg*"    IOB = true;
INST "ewrap_reg"             IOB = true;
INST "en_cdet_reg"           IOB = true;
INST "rx_code_group0_reg*"   IOB = true;
INST "rx_code_group1_reg*"   IOB = true;
```

Note: For Virtex-4, Virtex-5, Virtex-6 and Spartan-6 devices, the example design will directly instantiate IOB DDR components and the previous constraints are not included.

Virtex-6 devices support TBI at 2.5V only and the device default SelectIO™ standard of LVCMOS25 is used. Please see the *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics* for more information. In Virtex-5, Virtex-4, Spartan-6 and Spartan-3 devices support is 3.3V by default and the UCF will contain the following syntax. Use this syntax together with the device IO Banking rules.

```
INST "tx_code_group<?>" IOSTANDARD = LVTTL;
INST "pma_tx_clk"       IOSTANDARD = LVTTL;

INST "rx_code_group<?>" IOSTANDARD = LVTTL;
INST "pma_rx_clk0"      IOSTANDARD = LVTTL;

INST "loc_ref"          IOSTANDARD = LVTTL;
INST "ewrap"            IOSTANDARD = LVTTL;
INST "en_cdet"          IOSTANDARD = LVTTL;
```

In addition, the example design provides pad locking on the TBI for several families. This is included as a guideline only, and there are no specific I/O location constraints for this core.

TBI Input Setup/Hold Timing

Input TBI Timing Specification

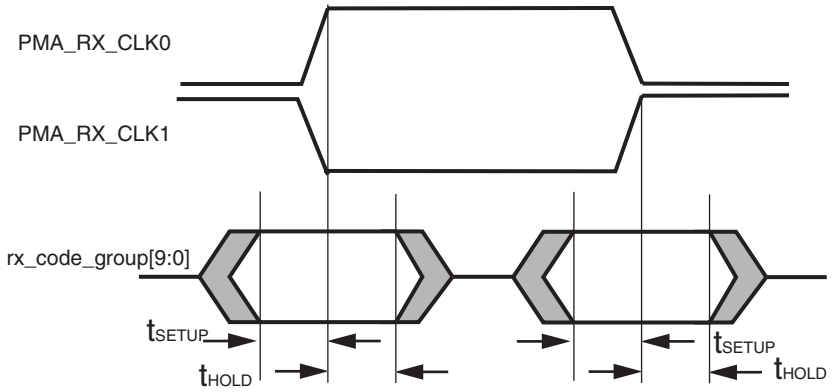


Figure 12-1: **Input TBI timing**

Figure 12-1 and Table 12-1 illustrate the setup and hold time window for the input TBI signals. These specify the worst-case data valid window presented to the FPGA device pins. There is only a 2 ns data valid window of guaranteed data presented across the TBI input bus. This must be correctly sampled by the FPGA devices.

Table 12-1: **Input TBI Timing**

Symbol	Min	Max	Units
t_{SETUP}	2.00	-	ns
t_{HOLD}	0.00	-	ns

Spartan-3, Spartan-3E, and Spartan-3A Devices

Figure 6-3, page 82 illustrates the TBI input logic provided by the example design for the Spartan-3 class family. DCMs are used on the `pma_rx_clk0` and `pma_rx_clk1` clock paths as illustrated. Phase-shifting is then applied to the DCMs to align the resultant clocks so that they correctly sample the 2 ns TBI data valid window at the input DDR flip-flops.

The fixed phase shift is applied to the DCMs using the following UCF syntax.

```
INST "core_wrapper/tbi_rx_clk0_dcm" CLKOUT_PHASE_SHIFT = FIXED;
INST "core_wrapper/tbi_rx_clk0_dcm" PHASE_SHIFT = -10;
INST "core_wrapper/tbi_rx_clk0_dcm" DESKEW_ADJUST = 0;

INST "core_wrapper/tbi_rx_clk1_dcm" CLKOUT_PHASE_SHIFT = FIXED;
INST "core_wrapper/tbi_rx_clk1_dcm" PHASE_SHIFT = -10;
INST "core_wrapper/tbi_rx_clk1_dcm" DESKEW_ADJUST = 0;
```

The values of `PHASE_SHIFT` are preconfigured in the example designs to meet the setup and hold constraints for the example TBI 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).

For customers fixing their own pinout, the setup and hold figures reported in the TRCE report can be used to initially setup the approximate DCM phase shift values. Appendix C, "Calculating the DCM Fixed Phase Shift Value" describes a more accurate method for fixing the phase shift by using hardware measurement of a unique PCB design.

Virtex-4 Devices

Figure 6-5, page 84 illustrates the TBI input logic provided by the example design for the Virtex-4 family. A DCM is used on the `pma_rx_clk0` clock path as illustrated. Phase-shifting is then applied to the DCM to align the resultant clock so it will correctly sample the 2 ns TBI data valid window at the input DDR flip-flops.

The fixed phase shift is applied to the DCM using the following UCF syntax.

```
INST "core_wrapper/tbi_rx_clk0_dcm" CLKOUT_PHASE_SHIFT = FIXED;
INST "core_wrapper/tbi_rx_clk0_dcm" PHASE_SHIFT = -35;
INST "core_wrapper/tbi_rx_clk0_dcm" DESKEW_ADJUST = 0;
```

The value of `PHASE_SHIFT` is preconfigured in the example designs to meet the setup and hold constraints for the example TBI 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).

For customers fixing their own pinout, the setup and hold figures reported in the TRCE report can be used to initially setup the approximate DCM phase shift values. Appendix C, "Calculating the DCM Fixed Phase Shift Value" describes a more accurate method for fixing the phase shift by using hardware measurement of a unique PCB design.

In addition, for Virtex-4 designs, the following UCF syntax is included:

```
#------------------------------------------------------------
# To check (analyze) TBI Rx Input Setup/Hold Timing        -
#------------------------------------------------------------
NET "rx_code_group<?>" OFFSET = IN 2 ns VALID 2 ns BEFORE "pma_rx_clk0"
RISING;
NET "rx_code_group<?>" OFFSET = IN 2 ns VALID 2 ns BEFORE "pma_rx_clk0"
FALLING;
```

This syntax causes the Xilinx implementation tools to analyze the input setup and hold constraints for the input TBI bus. If these constraints are not met then the tools will report timing errors. However, the tools will NOT attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the DCM PHASE_SHIFT value in the UCF.

Virtex-5 Devices

Figure 6-7, page 86 illustrates the TBI input logic provided by the example design for the Virtex-5 family. IODELAY elements are instantiated on the TBI data input path as illustrated. Fixed tap delays are applied to these IODELAY elements to delay the `rx_code_group[9:0]` bus so that data is correctly sampled at the IOB IDDR registers, thereby meeting TBI input setup and hold timing constraints.

The number of tap delays are applied using the following UCF syntax.

```
#------------------------------------------------------------
# To Adjust TBI Rx Input Setup/Hold Timing
#------------------------------------------------------------
INST "core_wrapper/tbi_rx_data_bus[9].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[8].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[7].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[6].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[5].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[4].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[3].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[2].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[1].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[0].delay_tbi_rx_data" IDELAY_VALUE
= "20";
```

The number of tap delays are preconfigured in the example designs to meet the setup and hold constraints for the example TBI 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 Setup/Hold Timing."

In addition, for Virtex-5 FPGA designs, the following UCF syntax is included:

```
#------------------------------------------------------------
# To check (analyze) TBI Rx Input Setup/Hold Timing        -
#------------------------------------------------------------
NET "rx_code_group<?>" OFFSET = IN 2 ns VALID 2 ns BEFORE "pma_rx_clk0"
RISING;
NET "rx_code_group<?>" OFFSET = IN 2 ns VALID 2 ns BEFORE "pma_rx_clk0"
FALLING;
```

This syntax will cause the Xilinx implementation tools to analyze the input setup and hold constraints for the input TBI bus. If these constraints are not met then the tools will report timing errors. However, the tools will NOT attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the number of tap delays for the IODELAY elements in the UCF.

Virtex-6 Devices

Figure 6-9, page 88 illustrates the TBI input logic provided by the example design for the Virtex-6 family. IODELAY elements are instantiated on the TBI data input path as illustrated. Fixed tap delays are applied to these IODELAY elements to delay the `rx_code_group[9:0]` bus so that data is correctly sampled at the IOB IDDR registers, thereby meeting TBI input setup and hold timing constraints.

The number of tap delays are applied using the following UCF syntax.

```
#------------------------------------------------------------
# To Adjust TBI Rx Input Setup/Hold Timing
#------------------------------------------------------------
INST "core_wrapper/tbi_rx_data_bus[9].delay_tbi_rx_data" IDELAY_VALUE
= "5";
INST "core_wrapper/tbi_rx_data_bus[8].delay_tbi_rx_data" IDELAY_VALUE
= "5";
INST "core_wrapper/tbi_rx_data_bus[7].delay_tbi_rx_data" IDELAY_VALUE
= "5";
INST "core_wrapper/tbi_rx_data_bus[6].delay_tbi_rx_data" IDELAY_VALUE
= "5";
INST "core_wrapper/tbi_rx_data_bus[5].delay_tbi_rx_data" IDELAY_VALUE
= "5";
INST "core_wrapper/tbi_rx_data_bus[4].delay_tbi_rx_data" IDELAY_VALUE
= "5";
INST "core_wrapper/tbi_rx_data_bus[3].delay_tbi_rx_data" IDELAY_VALUE
= "5";
INST "core_wrapper/tbi_rx_data_bus[2].delay_tbi_rx_data" IDELAY_VALUE
= "5";
INST "core_wrapper/tbi_rx_data_bus[1].delay_tbi_rx_data" IDELAY_VALUE
= "5";
INST "core_wrapper/tbi_rx_data_bus[0].delay_tbi_rx_data" IDELAY_VALUE
= "5";
```

The number of tap delays are preconfigured in the example designs to meet the setup and hold constraints for the example TBI 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 Setup/Hold Timing."

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

```
#------------------------------------------------------------
# To check (analyze) TBI Rx Input Setup/Hold Timing        -
#------------------------------------------------------------
NET "rx_code_group<?>" OFFSET = IN 2 ns VALID 2 ns BEFORE "pma_rx_clk0"
RISING;
NET "rx_code_group<?>" OFFSET = IN 2 ns VALID 2 ns BEFORE "pma_rx_clk0"
FALLING;
```

This syntax will cause the Xilinx implementation tools to analyze the input setup and hold constraints for the input TBI bus. If these constraints are not met then the tools will report timing errors. However, the tools will NOT attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the number of tap delays for the IODELAY elements in the UCF.

Spartan-6 Devices

Figure 6-11, page 90 illustrates the TBI input logic provided by the example design for the Spartan-6 family. IODELAY2 elements are instantiated on the TBI data input path as illustrated. Fixed tap delays are applied to these IODELAY2 elements to delay the `rx_code_group[9:0]` bus so that data is correctly sampled at the IOB IDDR registers, thereby meeting TBI input setup and hold timing constraints.

The number of tap delays are applied using the following UCF syntax.

```
#------------------------------------------------------------
# To Adjust TBI Rx Input Setup/Hold Timing
#------------------------------------------------------------
INST "core_wrapper/tbi_rx_data_bus[9].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[8].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[7].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[6].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[5].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[4].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[3].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[2].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[1].delay_tbi_rx_data" IDELAY_VALUE
= "20";
INST "core_wrapper/tbi_rx_data_bus[0].delay_tbi_rx_data" IDELAY_VALUE
= "20";
```

The number of tap delays are preconfigured in the example designs to meet the setup and hold constraints for the example TBI 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 Setup/Hold Timing."

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

```
#------------------------------------------------------------
# To check (analyze) TBI Rx Input Setup/Hold Timing        -
#------------------------------------------------------------
NET "rx_code_group<?>" OFFSET = IN 2 ns VALID 2 ns BEFORE "pma_rx_clk0"
RISING;
NET "rx_code_group<?>" OFFSET = IN 2 ns VALID 2 ns BEFORE "pma_rx_clk0"
FALLING;
```

This syntax will cause the Xilinx implementation tools to analyze the input setup and hold constraints for the input TBI bus. If these constraints are not met then the tools will report timing errors. However, the tools will NOT attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the number of tap delays for the IODELAY elements in the UCF.

Constraints When Implementing an External GMII

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from this UCF have been copied into the following examples, and should be studied in conjunction with the HDL source code for the example design. See also the section, "Implementing External GMII," page 70.

Clock Period Constraints

When implementing an external GMII, the Transmitter Elastic Buffer delivered with the example design (or similar logic) must be used. The input transmitter GMII signals are then synchronous to their own clock domain (`gmii_tx_clk` is used in the example design). This clock must be constrained for a clock frequency of 125 MHz. The following UCF syntax shows the necessary constraints being applied to the example design.

```
############################################################
# GMII Clock period Constraints: please do not relax       #
############################################################
NET "gmii_tx_clk_bufg" TNM_NET = "gmii_tx_clk";
TIMESPEC "ts_gmii_tx_clk" = PERIOD "gmii_tx_clk" 8000 ps HIGH 50 %;
```

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.

```
############################################################
# GMII Transmitter Constraints:  place flip-flops in IOB   #
############################################################
INST "gmii_txd*"  IOB = true;
INST "gmii_tx_en" IOB = true;
INST "gmii_tx_er" IOB = true;

############################################################
# GMII Receiver Constraints:  place flip-flops in IOB      #
############################################################
INST "gmii_rxd_obuf*"  IOB = true;
INST "gmii_rx_dv_obuf" IOB = true;
INST "gmii_rx_er_obuf" IOB = true;
```

Virtex-6 devices support GMII at 2.5V only and the device default SelectIO standard of LVCMOS25 is used. Please see the *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics* for more information. In Virtex-5, Virtex-4, Spartan-6 and Spartan-3 devices, GMII by default is supported at 3.3V and the UCF will contain the following syntax. Use this syntax together with the device IO Banking rules.

```
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 for several families. This is a provided as a guideline only; there are no specific I/O location constraints for this core.

GMII Input Setup/Hold Timing

Input GMII timing specification

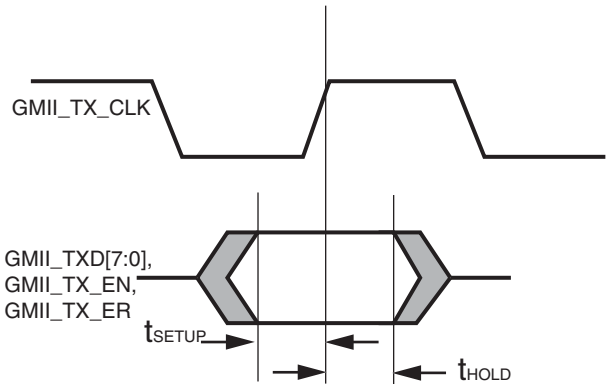


Figure 12-2: **Input GMII timing**

Figure 12-2 and Table 12-2 illustrate the setup and hold time window for the input GMII signals. These are the worst-case data valid window presented to the FPGA device pins.

Observe that there is, in total, a 2 ns data valid window of guaranteed data which is presented across the GMII input bus. This must be correctly sampled by the FPGA devices.

Table 12-2: **Input GMII Timing**

Symbol	Min	Max	Units
t_{SETUP}	2.00	-	ns
t_{HOLD}	0.00	-	ns

Spartan-3, Spartan-3E, and Spartan-3A Devices

Figure 5-12 illustrates the GMII input logic which is provided by the example design for the Spartan-3 class family. A DCM must be used on the `gmii_tx_clk` clock path as illustrated. Phase-shifting is then applied to the DCM to align the resultant clock so that it will correctly sample the 2ns GMII data valid window at the input flip-flops.

The fixed phase shift is applied to the DCM using the following UCF syntax.

```
INST "gmii_tx_dcm" CLKOUT_PHASE_SHIFT = FIXED;
INST "gmii_tx_dcm" PHASE_SHIFT = -20;
INST "gmii_tx_dcm" DESKEW_ADJUST = 0;
```

The value of `PHASE_SHIFT` 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).

For customers fixing their own pinout, the setup and hold figures reported in the TRCE report can be used to initially setup the approximate DCM phase shift. Appendix C, "Calculating the DCM Fixed Phase Shift Value" describes a more accurate method for fixing the phase shift by using hardware measurement of a unique PCB design.

Virtex-4 Devices

Figure 5-12 illustrates the GMII input logic provided by the example design for the Virtex-4 family. A DCM must be used on the `gmii_tx_clk` clock path as illustrated. Phase-shifting is then applied to the DCM to align the resultant clock so that it will correctly sample the 2 ns GMII data valid window at the input flip-flops.

The fixed phase shift is applied to the DCM using the following UCF syntax.

```
INST "gmii_tx_dcm" CLKOUT_PHASE_SHIFT = FIXED;
INST "gmii_tx_dcm" PHASE_SHIFT = -20;
INST "gmii_tx_dcm" DESKEW_ADJUST = 0;
```

The value of `PHASE_SHIFT` 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).

For customers fixing their own pinout, the setup and hold figures reported in the TRCE report can be used to initially setup the approximate DCM phase shift. Appendix C, "Calculating the DCM Fixed Phase Shift Value" describes a more accurate method for fixing the phase shift by using hardware measurement of a unique PCB design.

In addition, for Virtex-4 FPGA designs, the following UCF syntax is included:

```
#------------------------------------------------------------
# To check (analyze) GMII Tx Input Setup/Hold Timing       -
#------------------------------------------------------------
INST "gmii_txd*"  TNM = IN_GMII;
INST "gmii_tx_en" TNM = IN_GMII;
INST "gmii_tx_er" TNM = IN_GMII;

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

This syntax will cause the Xilinx implementation tools to analyze the input setup and hold constraints for the input GMII bus. If these constraints are not met then the tools will report timing errors. However, the tools will NOT attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the DCM PHASE_SHIFT value in the UCF

Virtex-5 Devices

Figure 5-13 illustrates the GMII input logic provided by the example design for the Virtex-5 family. IODELAY elements are instantiated on the GMII data input path as illustrated. Fixed tap delays are applied to these IODELAY elements to delay the GMII input data signals so that data is correctly sampled at the IOB IDDR registers, thereby meeting GMII input setup and hold timing constraints.

The number of tap delays are applied using the following UCF syntax.

```
#------------------------------------------------------------
# To Adjust GMII Tx Input Setup/Hold Timing                 -
#------------------------------------------------------------
INST "delay_gmii_tx_en"  IDELAY_VALUE = "20";
INST "delay_gmii_tx_er"  IDELAY_VALUE = "20";

INST "gmii_data_bus[7].delay_gmii_txd"  IDELAY_VALUE = "20";
INST "gmii_data_bus[6].delay_gmii_txd"  IDELAY_VALUE = "20";
INST "gmii_data_bus[5].delay_gmii_txd"  IDELAY_VALUE = "20";
INST "gmii_data_bus[4].delay_gmii_txd"  IDELAY_VALUE = "20";
INST "gmii_data_bus[3].delay_gmii_txd"  IDELAY_VALUE = "20";
INST "gmii_data_bus[2].delay_gmii_txd"  IDELAY_VALUE = "20";
INST "gmii_data_bus[1].delay_gmii_txd"  IDELAY_VALUE = "20";
INST "gmii_data_bus[0].delay_gmii_txd"  IDELAY_VALUE = "20";
```

The number of tap delays are 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 Setup/Hold Timing."

In addition, for Virtex-5 FPGA designs, the following UCF syntax is included:

```
#------------------------------------------------------------
# To check (analyze) GMII Tx Input Setup/Hold Timing        -
#------------------------------------------------------------
INST "gmii_txd*"  TNM = IN_GMII;
INST "gmii_tx_en" TNM = IN_GMII;
INST "gmii_tx_er" TNM = IN_GMII;

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

This syntax will cause the Xilinx implementation tools to analyze the input setup and hold constraints for the input GMII bus. If these constraints are not met then the tools will report timing errors. However, the tools will NOT attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the number of tap delays for the IODELAY elements in the UCF.

Virtex-6 Devices

Figure 5-14 illustrates the GMII input logic provided by the example design for the Virtex-6 family. IODELAY elements are instantiated on the GMII data input path as illustrated. Fixed tap delays are applied to these IODELAY elements to delay the GMII input data signals so that data is correctly sampled at the IOB IDDR registers, thereby meeting GMII input setup and hold timing constraints.

The number of tap delays are applied using the following UCF syntax.

```
#------------------------------------------------------------
# To Adjust GMII Tx Input Setup/Hold Timing                 -
#------------------------------------------------------------
INST "delay_gmii_tx_en"  IDELAY_VALUE = "5";
INST "delay_gmii_tx_er"  IDELAY_VALUE = "5";

INST "gmii_data_bus[7].delay_gmii_txd"  IDELAY_VALUE = "5";
INST "gmii_data_bus[6].delay_gmii_txd"  IDELAY_VALUE = "5";
INST "gmii_data_bus[5].delay_gmii_txd"  IDELAY_VALUE = "5";
INST "gmii_data_bus[4].delay_gmii_txd"  IDELAY_VALUE = "5";
INST "gmii_data_bus[3].delay_gmii_txd"  IDELAY_VALUE = "5";
INST "gmii_data_bus[2].delay_gmii_txd"  IDELAY_VALUE = "5";
INST "gmii_data_bus[1].delay_gmii_txd"  IDELAY_VALUE = "5";
INST "gmii_data_bus[0].delay_gmii_txd"  IDELAY_VALUE = "5";
```

The number of tap delays are 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 Setup/Hold Timing."

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

```
#------------------------------------------------------------
# To check (analyze) GMII Tx Input Setup/Hold Timing        -
#------------------------------------------------------------
INST "gmii_txd*"  TNM = IN_GMII;
INST "gmii_tx_en" TNM = IN_GMII;
INST "gmii_tx_er" TNM = IN_GMII;

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

This syntax will cause the Xilinx implementation tools to analyze the input setup and hold constraints for the input GMII bus. If these constraints are not met then the tools will report timing errors. However, the tools will NOT attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the number of tap delays for the IODELAY elements in the UCF.

Spartan-6 Devices

Figure 5-15 illustrates the GMII input logic provided by the example design for the Spartan-6 family. IODELAY2 elements are instantiated on the GMII data input path as illustrated. Fixed tap delays are applied to these IODELAY2 elements to delay the GMII input data signals so that data is correctly sampled at the IOB IDDR registers, thereby meeting GMII input setup and hold timing constraints.

The number of tap delays are applied using the following UCF syntax.

```
#------------------------------------------------------------
# To Adjust GMII Tx Input Setup/Hold Timing                 -
#------------------------------------------------------------
INST "delay_gmii_tx_en"  IDELAY_VALUE = "10";
INST "delay_gmii_tx_er"  IDELAY_VALUE = "10";

INST "gmii_data_bus[7].delay_gmii_txd"  IDELAY_VALUE = "10";
INST "gmii_data_bus[6].delay_gmii_txd"  IDELAY_VALUE = "10";
INST "gmii_data_bus[5].delay_gmii_txd"  IDELAY_VALUE = "10";
INST "gmii_data_bus[4].delay_gmii_txd"  IDELAY_VALUE = "10";
INST "gmii_data_bus[3].delay_gmii_txd"  IDELAY_VALUE = "10";
INST "gmii_data_bus[2].delay_gmii_txd"  IDELAY_VALUE = "10";
INST "gmii_data_bus[1].delay_gmii_txd"  IDELAY_VALUE = "10";
INST "gmii_data_bus[0].delay_gmii_txd"  IDELAY_VALUE = "10";
```

The number of tap delays are 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 Setup/Hold Timing."

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

```
#------------------------------------------------------------
# To check (analyze) GMII Tx Input Setup/Hold Timing        -
#------------------------------------------------------------
INST "gmii_txd*"  TNM = IN_GMII;
INST "gmii_tx_en" TNM = IN_GMII;
INST "gmii_tx_er" TNM = IN_GMII;

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

This syntax will cause the Xilinx implementation tools to analyze the input setup and hold constraints for the input GMII bus. If these constraints are not met then the tools will report timing errors. However, the tools will NOT attempt to automatically correct the timing in the case of failure. These must be corrected manually by changing the number of tap delays for the IODELAY elements in the UCF.

Understanding Timing Reports for Setup/Hold Timing

Setup and Hold results for the TBI or GMII input busses for the following devices are defined in the Data Sheet Report section of the Timing Report. The results are self-explanatory and show an obvious correlation and relationship to Figure 12-1 and Figure 12-2.

The following example shows the GMII report from a Spartan-3A DSP device. The implementation requires 1.531 ns of setup (this is less than the 2 ns required, to allow for slack). The implementation requires -0.125 ns of hold (this is less than the 0 ns required, to allow for slack).

```
Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock gmii_tx_clk
------------+------------+------------+------------------+--------+
            |  Setup to  |  Hold to   |                  | Clock  |
Source      | clk (edge) | clk (edge) |Internal Clock(s) | Phase  |
------------+------------+------------+------------------+--------+
gmii_tx_en  |    1.531(R)|   -0.141(R)|gmii_tx_clk_bufg  |   0.000|
gmii_tx_er  |    1.531(R)|   -0.141(R)|gmii_tx_clk_bufg  |   0.000|
gmii_txd<0> |    1.531(R)|   -0.141(R)|gmii_tx_clk_bufg  |   0.000|
gmii_txd<1> |    1.525(R)|   -0.135(R)|gmii_tx_clk_bufg  |   0.000|
gmii_txd<2> |    1.531(R)|   -0.141(R)|gmii_tx_clk_bufg  |   0.000|
gmii_txd<3> |    1.525(R)|   -0.135(R)|gmii_tx_clk_bufg  |   0.000|
gmii_txd<4> |    1.515(R)|   -0.125(R)|gmii_tx_clk_bufg  |   0.000|
gmii_txd<5> |    1.515(R)|   -0.125(R)|gmii_tx_clk_bufg  |   0.000|
gmii_txd<6> |    1.520(R)|   -0.130(R)|gmii_tx_clk_bufg  |   0.000|
gmii_txd<7> |    1.520(R)|   -0.130(R)|gmii_tx_clk_bufg  |   0.000|
------------+------------+------------+------------------+--------+
```

XILINX®

Chapter 13

Interfacing to Other Cores

The 1000BASE-X PCS/PMA or SGMII core can be integrated in a single device with the Tri-Mode Ethernet MAC core to extend the system functionality to include the MAC sublayer. The Tri-Mode Ethernet MAC core provides support for operation at 10 Mbps, 100 Mbps, and 1 Gbps.

A description of the latest available IP update containing the Tri-Mode Ethernet MAC core and instructions can be found in the Tri-Mode Ethernet MAC product Web page:

www.xilinx.com/systemio/temac/index.htm

> ***Caution!*** The Tri-Mode Ethernet MAC should always be configured for full-duplex operation when used with the 1000BASE-X PCS/PMA or SGMII core. This constraint is due to the increased latency introduced by the 1000BASE-X PCS/PMA or SGMII core. With half-duplex operation, the MAC response to collisions will be late, violating the CDMA protocol.

The Tri-Mode Ethernet MAC core can be generated with a choice of supported speeds. Please see the following sections as applicable:

- "Integrating for 1 Gbps Only Speed Capability"
- "Integration for Tri-speed Capability"

Integrating for 1 Gbps Only Speed Capability

In this section, it is assumed that the Tri-Mode Ethernet MAC core is generated with only 1 Gbps ethernet speed and full-duplex only support. This will provide the most optimal solution.

Integration of the Tri-Mode Ethernet MAC to Provide 1000BASE-X PCS with TBI

Figure 13-1 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in 1000BASE-X mode with the parallel TBI) to the Tri-Mode Ethernet MAC core.

Features of this configuration include:

- Direct internal connections are made between the GMII interfaces between the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected to that of the Tri-Mode Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the embedded Receiver Elastic Buffer in the Ethernet 1000BASE-X PCS/PMA, the entire GMII is synchronous to a single clock domain. Therefore, `gtx_clk` is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the Tri-Mode Ethernet MAC core operates in the same clock domain.

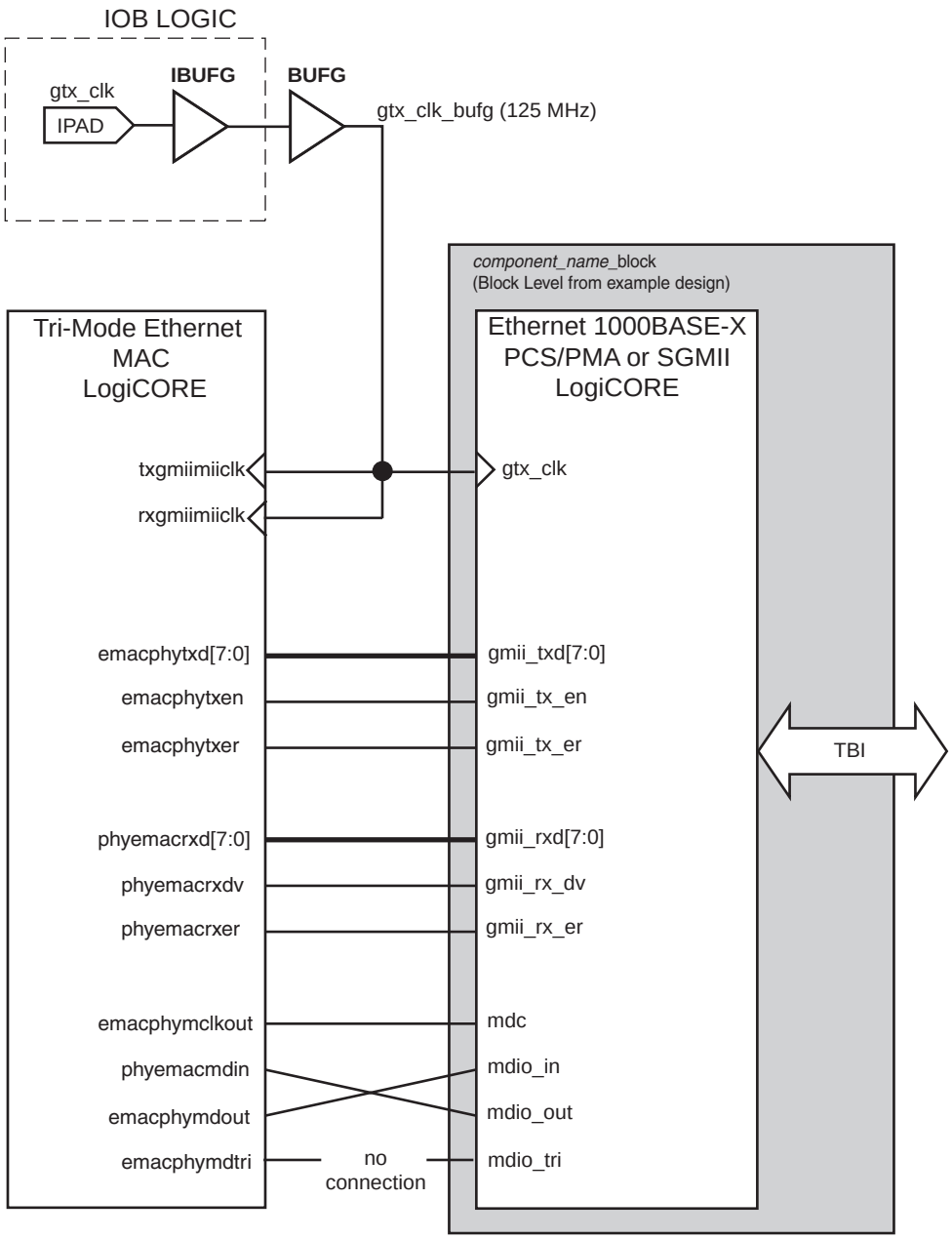


Figure 13-1: **Tri-Mode Ethernet MAC Extended to Include 1000BASE-X PCS with TBI**

Integration of the Tri-Mode Ethernet MAC to Provide 1000BASE-X Using Transceivers

Virtex-4 Devices

Figure 13-2 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in 1000BASE-X mode) to the Tri-Mode Ethernet MAC core.

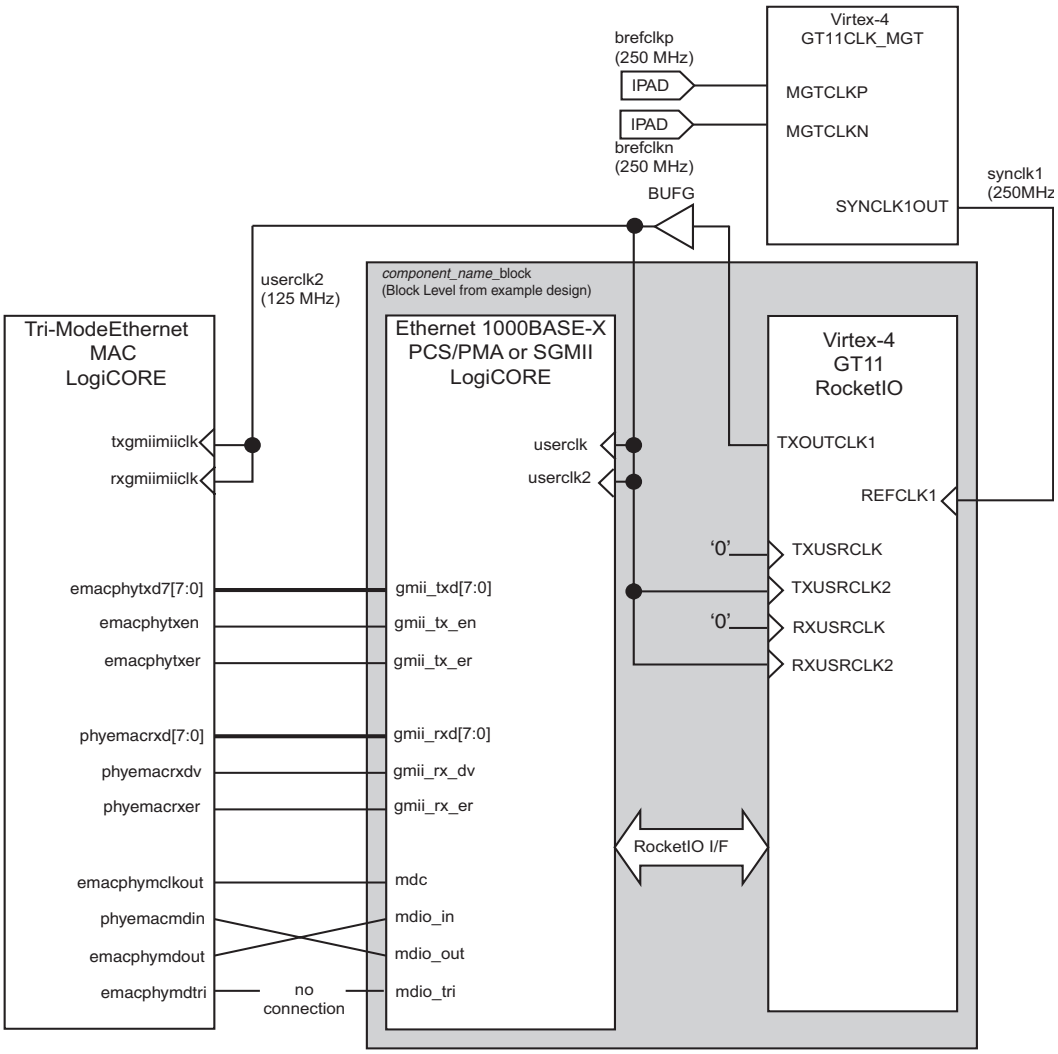


Figure 13-2: **Tri-Mode Ethernet MAC Extended to Include 1000BASE-X PCS and PMA Using a Virtex-4 FPGA RocketIO™ MGT Transceiver**

Features of this configuration include:

- Direct internal connections are made between the GMII interfaces between the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected up to that of the Tri-Mode Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the embedded Receiver Elastic Buffer in the MGT, the entire GMII is synchronous to a single clock domain. Therefore `userclk2` is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the 1 Tri-Mode Ethernet MAC core now operate in the same clock domain.

Virtex-5 LXT and SXT Devices

Figure 13-3 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in 1000BASE-X mode) to the Tri-Mode Ethernet MAC core.

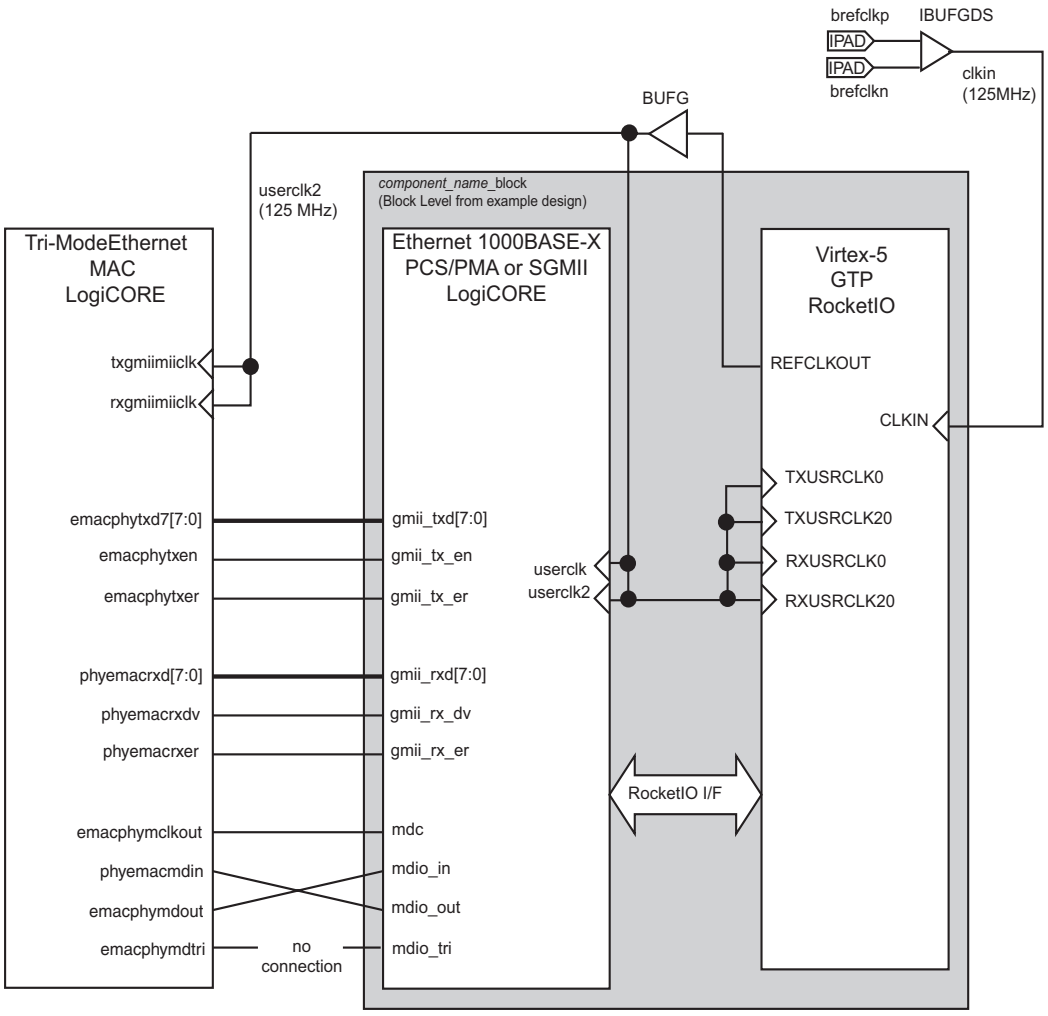


Figure 13-3: **Tri-Mode Ethernet MAC Extended to Include 1000BASE-X PCS and PMA Using a Virtex-5 FPGA RocketIO GTP Transceiver**

Features of this configuration include:

- Direct internal connections are made between the GMII interfaces between the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected up to that of the Tri-Mode Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the embedded Receiver Elastic Buffer in the GTP transceiver, the entire GMII is synchronous to a single clock domain. Therefore `userclk2` is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the Tri-Mode Ethernet MAC core now operate in the same clock domain.

Virtex-5 FXT and TXT Devices

Figure 13-4 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in 1000BASE-X mode) to the Tri-Mode Ethernet MAC core.

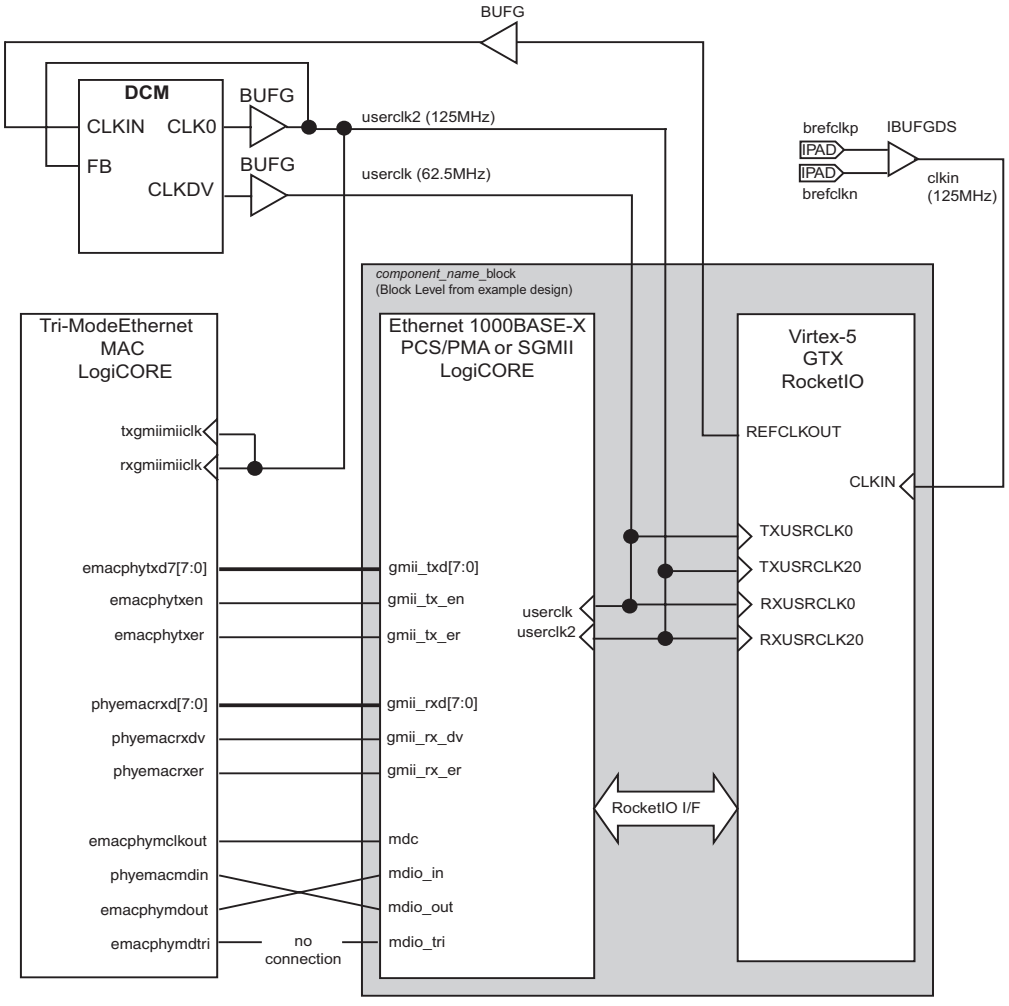


Figure 13-4: **Tri-Mode Ethernet MAC Extended to Include 1000BASE-X PCS and PMA Using a Virtex-5 FPGA RocketIO GTX Transceiver**

Features of this configuration include:

- Direct internal connections are made between the GMII interfaces between the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected up to that of the Tri-Mode Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the embedded Receiver Elastic Buffer in the GTX transceiver, the entire GMII is synchronous to a single clock domain. Therefore `userclk2` is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the Tri-Mode Ethernet MAC core now operate in the same clock domain.

Virtex-6 Devices

Figure 13-5 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in 1000BASE-X mode) to the Tri-Mode Ethernet MAC core.

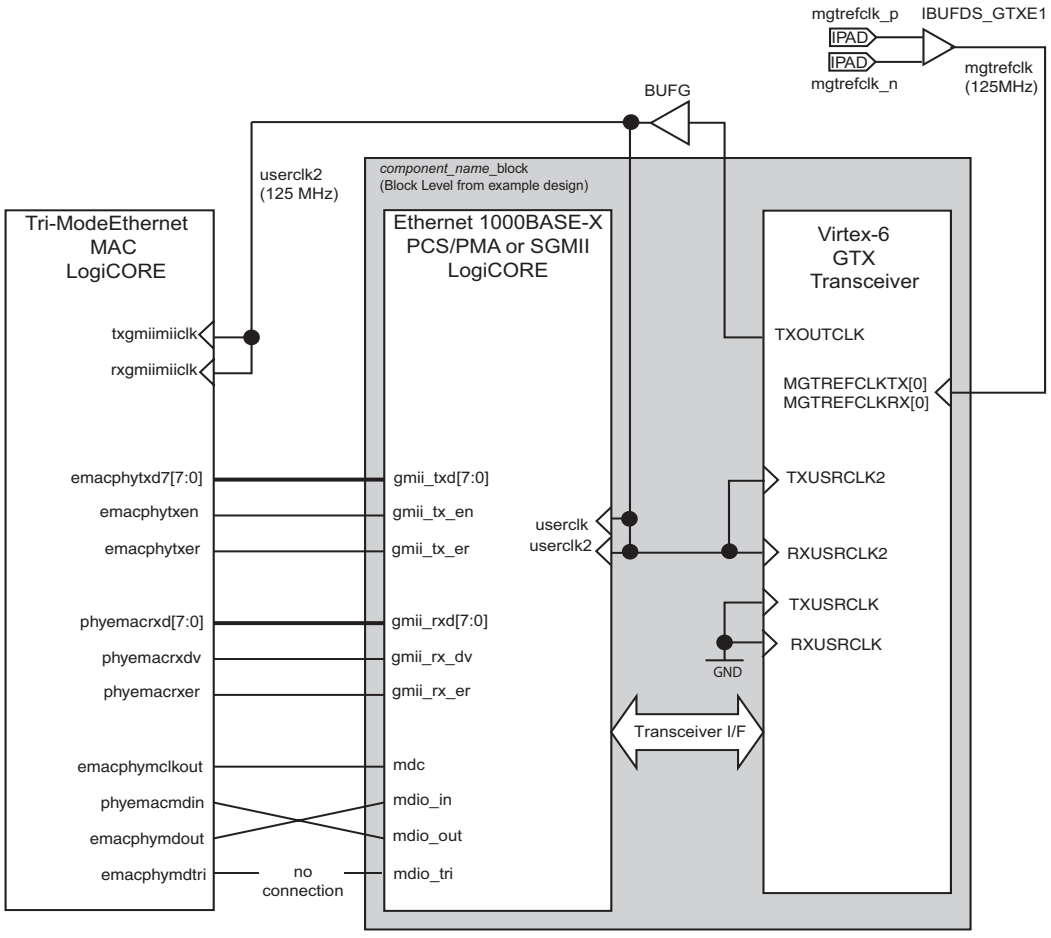


Figure 13-5: **Tri-Mode Ethernet MAC Extended to Include 1000BASE-X PCS and PMA Using a Virtex-6 FPGA GTX Transceiver**

Features of this configuration include:

- Direct internal connections are made between the GMII interfaces between the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected up to that of the Tri-Mode Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the embedded Receiver Elastic Buffer in the transceiver, the entire GMII is synchronous to a single clock domain. Therefore `userclk2` is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the Tri-Mode Ethernet MAC core now operate in the same clock domain.

Spartan®-6 LXT Devices

Figure 13-6 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in 1000BASE-X mode) to the Tri-Mode Ethernet MAC core.

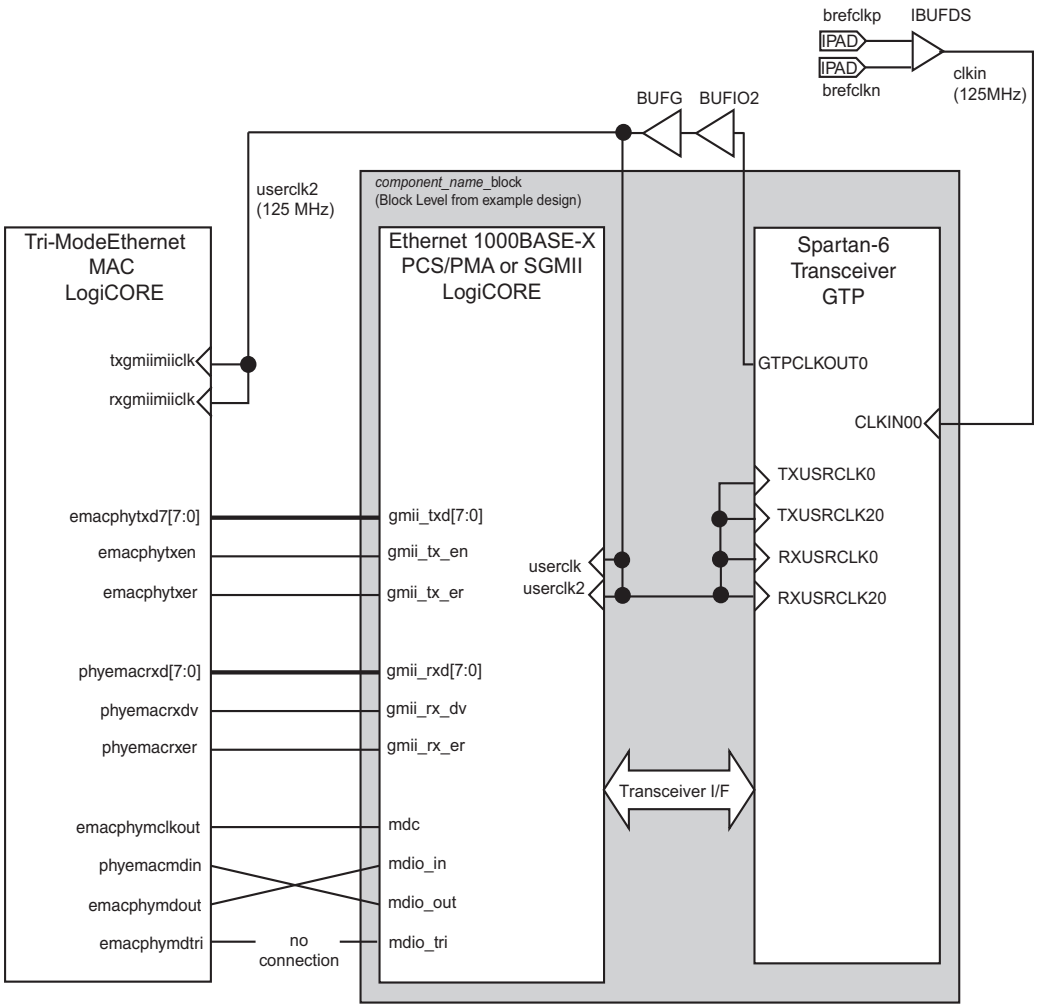


Figure 13-6: **Tri-Mode Ethernet MAC Extended to Include 1000BASE-X PCS and PMA Using a Spartan-6 FPGA GTP Transceiver**

Features of this configuration include:

- Direct internal connections are made between the GMII interfaces between the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected up to that of the Tri-Mode Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the embedded Receiver Elastic Buffer in the GTP transceiver, the entire GMII is synchronous to a single clock domain. Therefore `userclk2` is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the Tri-Mode Ethernet MAC core now operate in the same clock domain.

Integration of the Tri-Mode Ethernet MAC to Provide SGMII (or Dynamic Switching) Functionality - 1 Gbps Only Operation

The connections required to provide SGMII functionality are identical to the connections required for either "Integration of the Tri-Mode Ethernet MAC to Provide 1000BASE-X PCS with TBI" or "Integration of the Tri-Mode Ethernet MAC to Provide 1000BASE-X Using Transceivers", depending upon the chosen physical interface. The only difference is that the Ethernet 1000BASE-X PCS/PMA or SGMII core is generated with the SGMII or Dynamic switching option.

Note: When operating at 1 Gbps speed only, the Rx Elastic Buffer internal to the GTP transceiver should be used to save device resources. Additionally, when operating at 1 Gbps only, the SGMII Adaptation Module instantiated from within the block level of the example design is not required and can be removed.

Integration for Tri-speed Capability

In this section, it is assumed that the Tri-Mode Ethernet MAC core is generated for Tri-speed operation and full-duplex only support. This will provide the most optimal solution.

This section assumes only SGMII or Dynamic switching operation. For 1000BASE-X designs, please see "Integrating for 1 Gbps Only Speed Capability".

Integration of the Tri-Mode Ethernet MAC to Provide SGMII (or Dynamic Switching) Functionality with TBI

Figure 13-7 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in SGMII mode with the TBI) to the Tri-Mode Ethernet MAC core. The following is a description of the functionality.

- The SGMII Adaptation module, provided in the example design for the Ethernet 1000BASE-X PCS/PMA or SGMII core when generated to the SGMII standard, can be used to interface the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected to that of the Tri-Speed Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the Receiver Elastic Buffer in the core, the entire GMII (transmitter and receiver paths) is synchronous to a single clock domain. Therefore, the `txcoreclk` and `rxcoreclk` inputs of the Tri-Speed Ethernet MAC core can always be driven from the same clock source.

Figure 13-7 illustrates the Tri-Mode Ethernet MAC core generated with the optional clock enable circuitry. This is the most efficient way to connect the two cores together in terms of clock resource usage and so is recommended. See the *Tri-Mode Ethernet MAC User Guide* for more information.

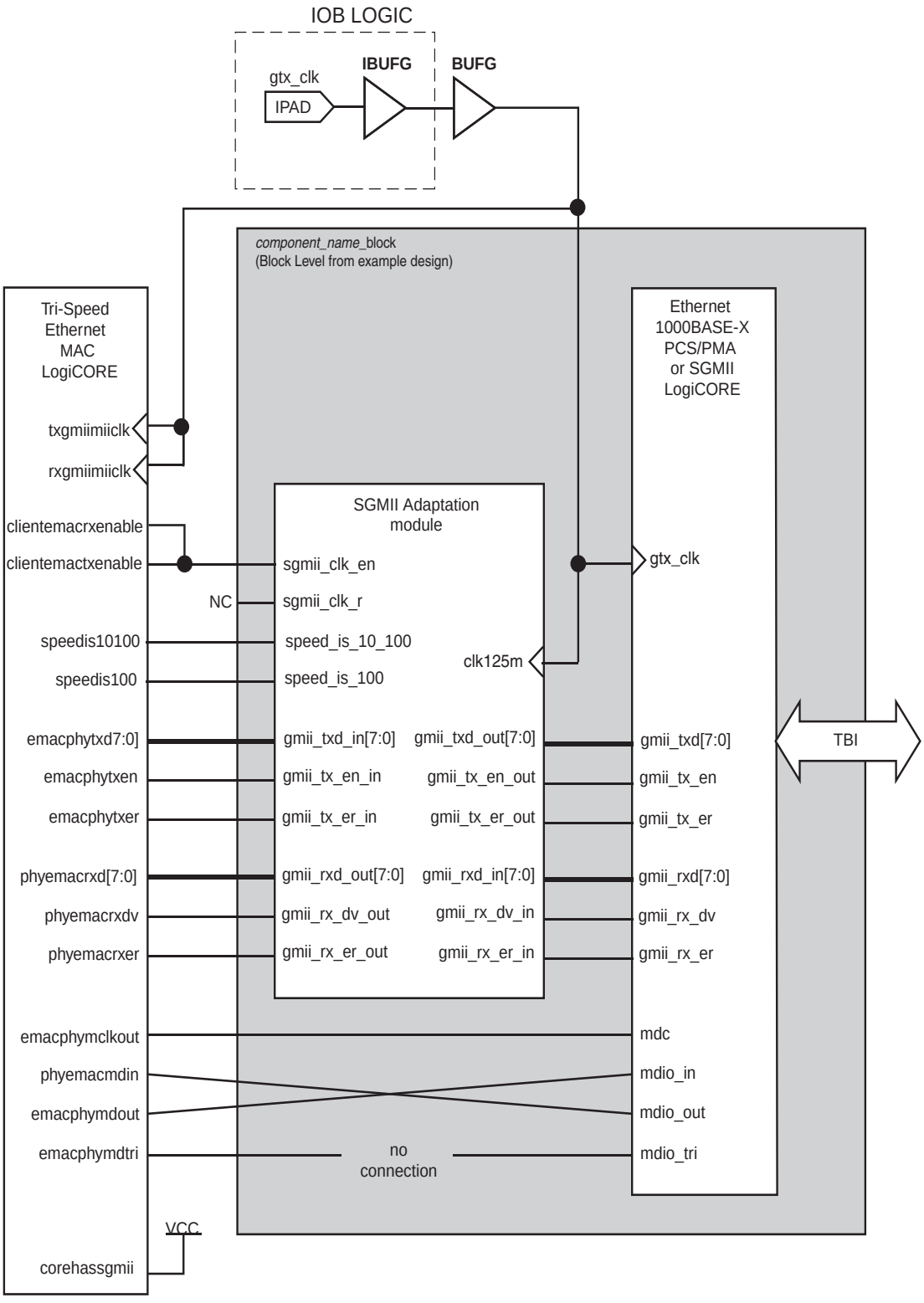


Figure 13-7: **Tri-Speed Ethernet MAC Extended to use an SGMII with TBI**

Integration of the Tri-Mode Ethernet MAC Using Transceivers

Virtex-4 Devices

Figure 13-8 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in SGMII mode with the Virtex®-4 FPGA MGT) to the Tri-Mode Ethernet MAC core.

The following conditions apply.

- The SGMII Adaptation module, as provided in the example design for the Ethernet 1000BASE-X PCS/PMA or SGMII core, when generated to the SGMII standard can be used to interface the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected up to that of the Tri-Speed Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the Receiver Elastic Buffer, the entire GMII (transmitter and receiver paths) is synchronous to a single clock domain. Therefore the `txcoreclk` and `rxcoreclk` inputs of the Tri-Speed Ethernet MAC core can always be driven from the same clock source. The entire design is synchronous to the 125 MHz reference clock derived from the `CLK2X180` output of the DCM.

Figure 13-8 illustrates the Tri-Mode Ethernet MAC core generated with the optional clock enable circuitry. This is the most efficient way to connect the two cores together in terms of clock resource usage and so is recommended. See the *Tri-Mode Ethernet MAC User Guide* for more information.

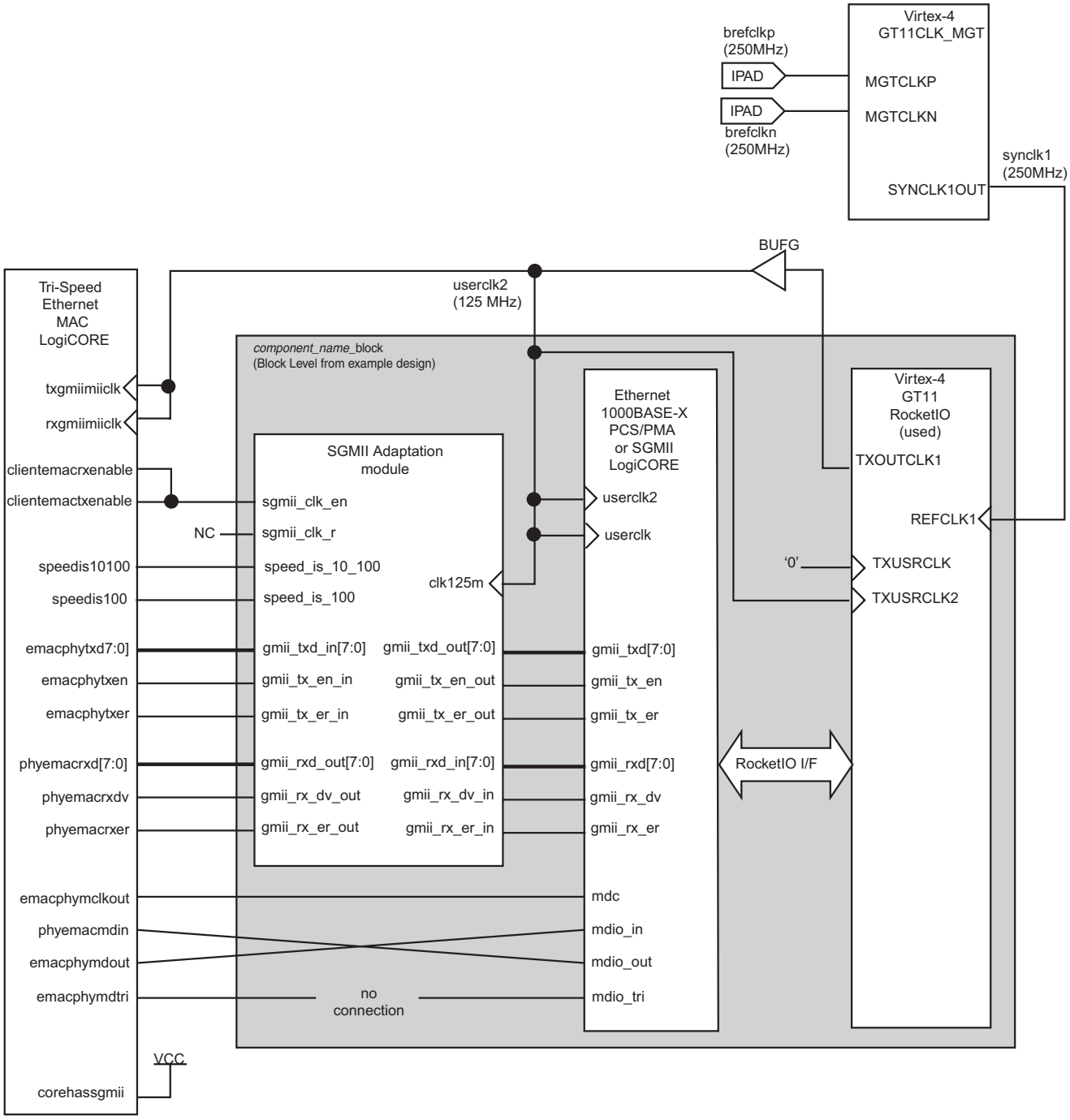


Figure 13-8: **Tri-Speed Ethernet MAC Extended to Use an SGMII in a Virtex-4 FPGA**

Virtex-5 LXT and SXT Devices

Figure 13-9 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in SGMII mode with the Virtex-5 FPGA GTP) to the Tri-Mode Ethernet MAC core.

The following conditions apply.

- The SGMII Adaptation module, as provided in the example design for the Ethernet 1000BASE-X PCS/PMA or SGMII core when generated to the SGMII standard, can be used to interface the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected up to that of the Tri-Speed Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the embedded Receiver Elastic Buffer in the GTP transceiver, the entire GMII is synchronous to a single clock domain. Therefore `userclk2` is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the Tri-Mode Ethernet MAC core now operate in the same clock domain.

Figure 13-9 illustrates the Tri-Mode Ethernet MAC core generated with the optional clock enable circuitry. This is the most efficient way to connect the two cores together in terms of clock resource usage and so is recommended. See the *Tri-Mode Ethernet MAC User Guide* for more information.

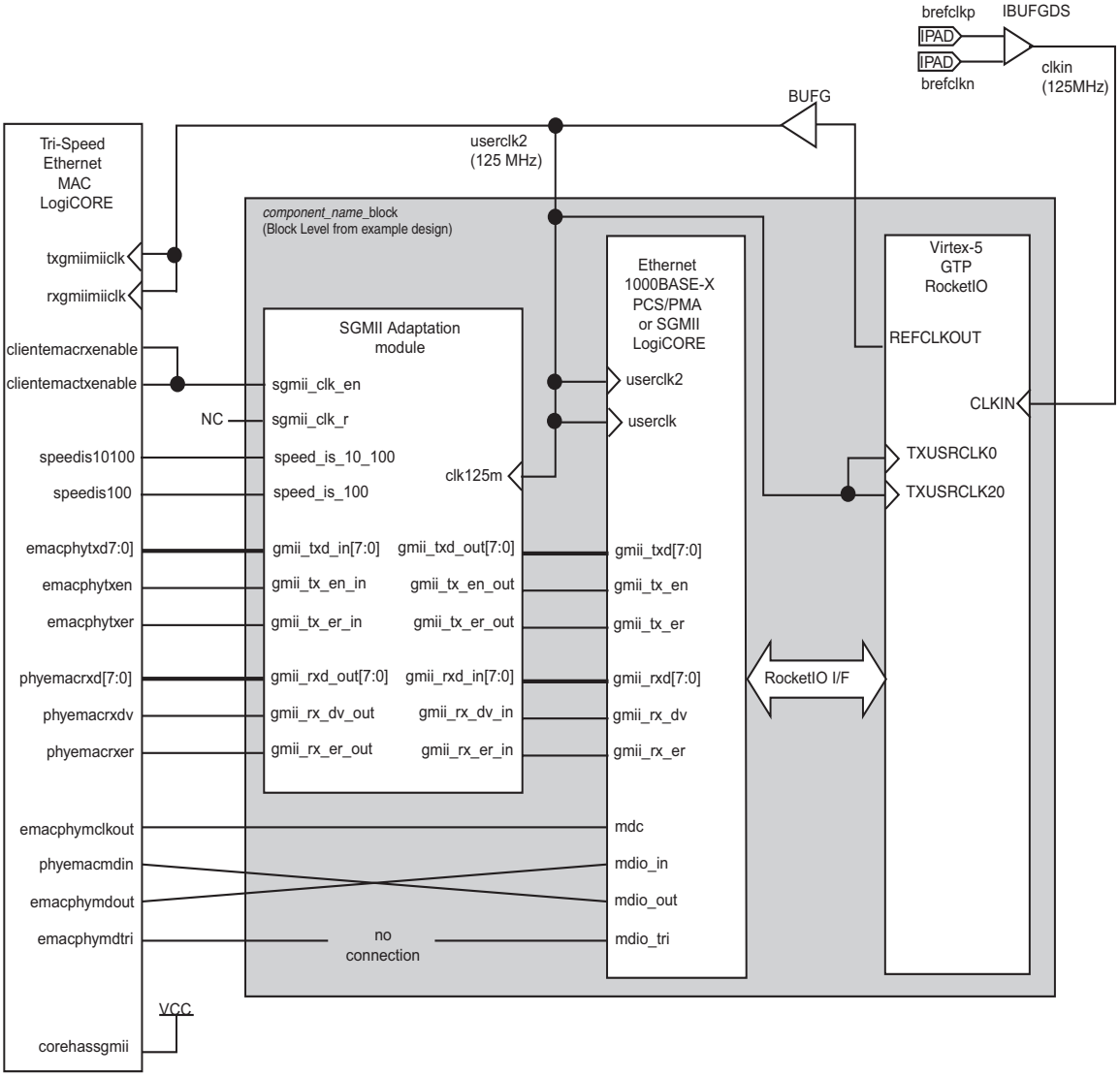


Figure 13-9: **Tri-Speed Ethernet MAC Extended to Use an SGMII in a Virtex-5 LXT/SXT Device**

Virtex-5 FXT and TXT Devices

Figure 13-10 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in SGMII mode with the Virtex-5 FPGA GTX) to the Tri-Mode Ethernet MAC core.

The following conditions apply.

- The SGMII Adaptation module, as provided in the example design for the Ethernet 1000BASE-X PCS/PMA or SGMII core when generated to the SGMII standard, can be used to interface the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected up to that of the Tri-Speed Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the Receiver Elastic Buffer, the entire GMII (transmitter and receiver paths) is synchronous to a single clock domain. Therefore the `txcoreclk` and `rxcoreclk` inputs of the Tri-Speed Ethernet MAC core can always be driven from the same clock source. The entire design is synchronous to the 125 MHz reference clock derived from the `CLK2X180` output of the DCM.

Figure 13-10 illustrates the Tri-Mode Ethernet MAC core generated with the optional clock enable circuitry. This is the most efficient way to connect the two cores together in terms of clock resource usage and so is recommended. See the *Tri-Mode Ethernet MAC User Guide* for more information.

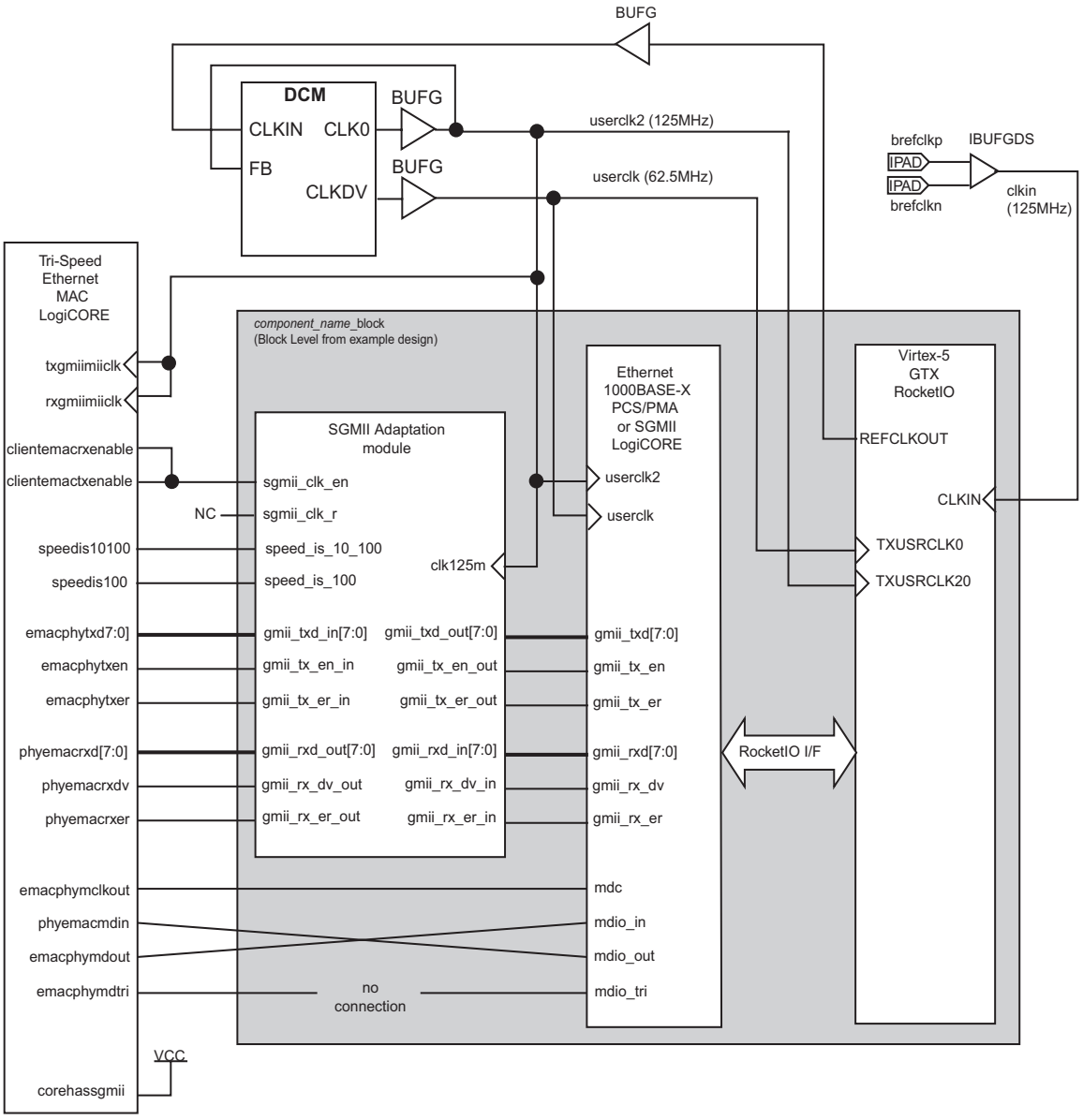


Figure 13-10: **Tri-Speed Ethernet MAC Extended to Use an SGMII in a Virtex-5 FXT and TXT Device**

Virtex-6 Devices

Figure 13-11 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in SGMII mode with the Virtex-6 FPGA GTX) to the Tri-Mode Ethernet MAC core.

The following conditions apply.

- The SGMII Adaptation module, as provided in the example design for the Ethernet 1000BASE-X PCS/PMA or SGMII core when generated to the SGMII standard, can be used to interface the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected up to that of the Tri-Speed Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the embedded Receiver Elastic Buffer, the entire GMII is synchronous to a single clock domain. Therefore `userclk2` is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the Tri-Mode Ethernet MAC core now operate in the same clock domain.

See also the *Tri-Mode Ethernet MAC User Guide* for more information.

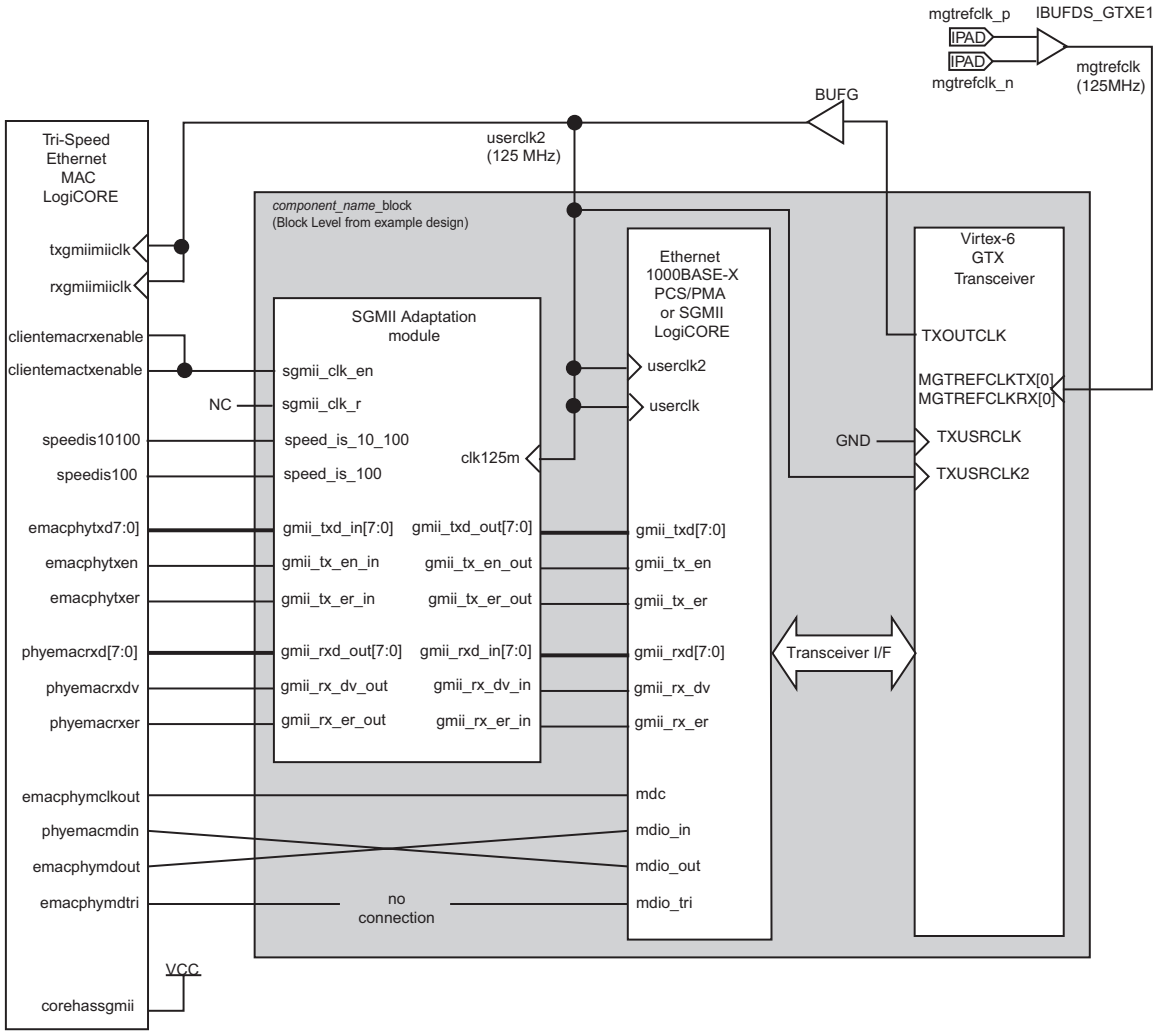


Figure 13-11: **Tri-Speed Ethernet MAC Extended to use an SGMII in Virtex-6 Devices**

Spartan-6 LXT Devices

Figure 13-12 illustrates the connections and clock management logic required to interface the Ethernet 1000BASE-X PCS/PMA or SGMII core (when used in SGMII mode with the Spartan-6 FPGA GTP) to the Tri-Mode Ethernet MAC core.

The following conditions apply.

- The SGMII Adaptation module, as provided in the example design for the Ethernet 1000BASE-X PCS/PMA or SGMII core when generated to the SGMII standard, can be used to interface the two cores.
- If both cores have been generated with the optional management interface, the MDIO port can be connected up to that of the Tri-Speed Ethernet MAC core, allowing the MAC to access the embedded configuration and status registers of the Ethernet 1000BASE-X PCS/PMA or SGMII core.
- Due to the embedded Receiver Elastic Buffer in the GTP transceiver, the entire GMII is synchronous to a single clock domain. Therefore `userclk2` is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the Tri-Mode Ethernet MAC core now operate in the same clock domain.

Figure 13-12 illustrates the Tri-Mode Ethernet MAC core generated with the optional clock enable circuitry. This is the most efficient way to connect the two cores together in terms of clock resource usage and so is recommended. See the *Tri-Mode Ethernet MAC User Guide* for more information.

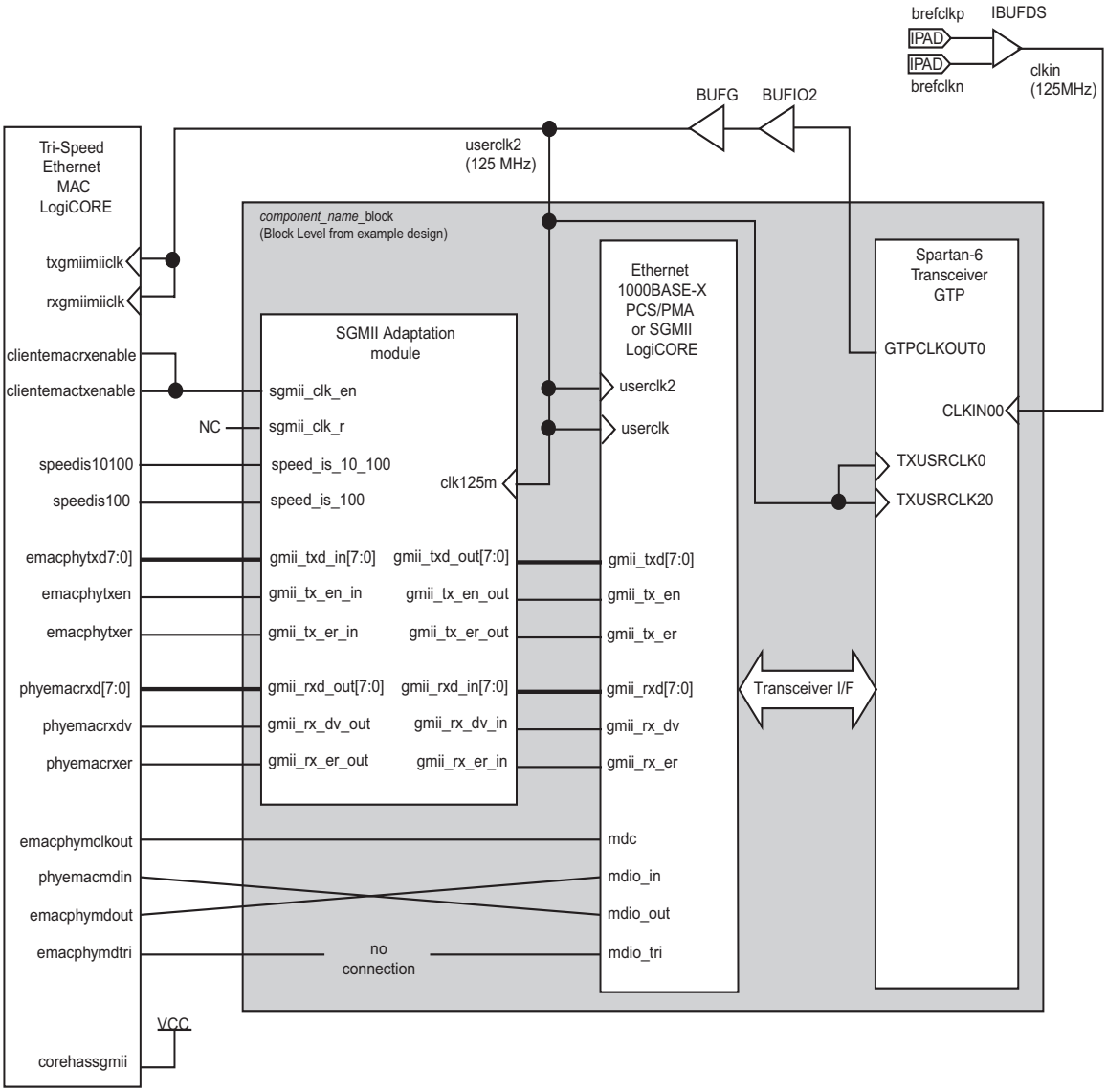


Figure 13-12: **Tri-Speed Ethernet MAC Extended to Use an SGMII in a Spartan-6 LXT Device**

XILINX®

Chapter 14

Special Design Considerations

This chapter describes the unique design considerations associated with implementing the Ethernet 1000BASE-X PCS/PMA or SGMII core.

Power Management

No power management considerations are recommended for the Ethernet 1000BASE-X PCS/PMA or SGMII core when using it with the TBI. When using the Ethernet 1000BASE-X PCS/PMA or SGMII core with a Virtex®-5, Virtex-6 or Spartan®-6 device, the transceiver may be placed in a low-power state in either of the following ways:

- Writing to the PCS Configuration Register 0 (if using the core with the optional Management Interface). The low-power state can only be removed by issuing the core with a reset. This reset can be achieved either by writing to the software reset bit in the PCS Configuration Register 0, or by driving the core `reset` port.
- Asserting the Power Down bit in the `configuration_vector` (if using the core without the optional Management Interface). The low-power state can only be removed by issuing the core with a reset by driving the `reset` port of the core.

Startup Sequencing

IEEE 802.3 clause 22.2.4.1.6 states that by default, a PHY should power-up in an isolate state (electrically isolated from the GMII).

- If you are using the core with the optional Management Interface, it is necessary to write to the PCS Configuration Register 0 to take the core out of the isolate state.
- If using the core without the optional Management interface, it is the responsibility of the client to ensure that the isolate input signal in the `configuration_vector` is asserted at power-on.

Loopback

This section details the implementation of the loopback feature. Loopback mode is enabled or disabled by either the "MDIO Management Interface," page 139, or by the "Optional Configuration Vector," page 179.

Core with the TBI

There is no physical loopback path in the core. Placing the core into loopback has the effect of asserting logic 1 on the `ewrap` signal of the TBI (see "1000BASE-X PCS with TBI Pinout," page 43). This instructs the attached PMA SERDES device to enter loopback mode as illustrated in Figure 14-1.

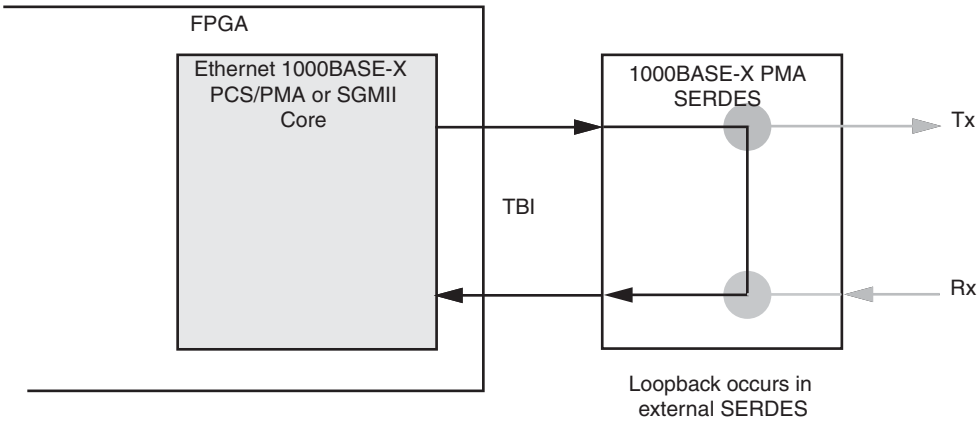


Figure 14-1: **Loopback Implementation Using the TBI**

Core with Transceiver

The loopback path is implemented in the core as illustrated in Figure 14-2. When placed into loopback, the data is routed from the transmitter path to the receiver path at the last possible point in the core. This point is immediately before the device-specific transceiver interface. When placed in loopback, the core creates a constant stream of Idle code groups that are transmitted through the MGT or GTP transceiver in accordance with the *IEEE 802.3* specification.

Earlier versions (before v5.0) of the core implemented loopback differently. The serial loopback feature of the device-specific transceiver was used by driving the `LOOPBACK[1:0]` port of the device-specific (MGT or GTP) transceiver. This is no longer the case, and the `loopback[1:0]` output port of the core is now permanently set to logic "00." However, for debugging purposes, the `LOOPBACK[1:0]` input port of the device-specific transceiver may be directly driven by the user logic to place it in either parallel or serial loopback mode.

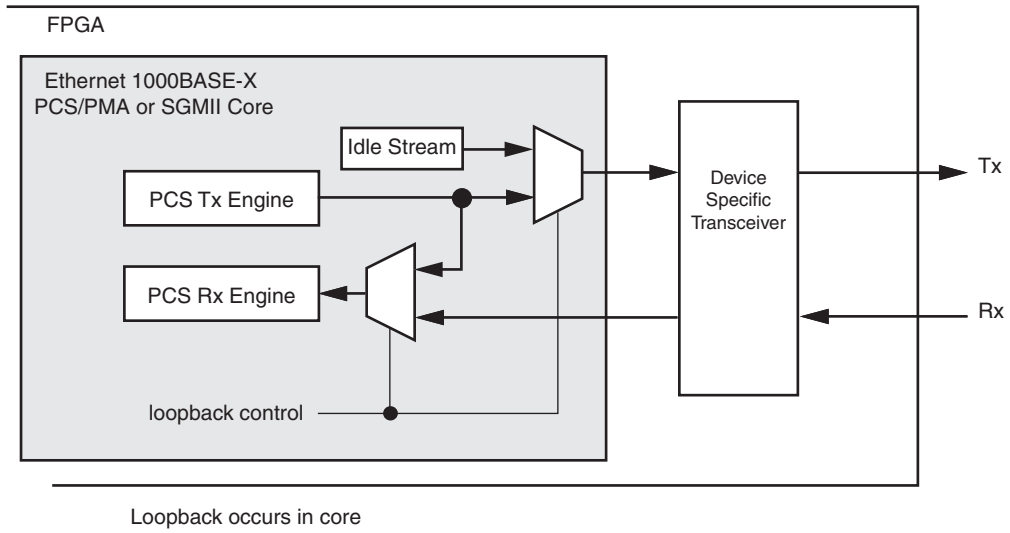


Figure 14-2: **Loopback Implementation When Using the Core with Device-Specific Transceivers**

XILINX®

Chapter 15

Implementing the Design

This chapter describes how to simulate and implement your design containing the Ethernet 1000BASE-X PCS/PMA or SGMII core.

Pre-implementation Simulation

A functional model of the Ethernet 1000BASE-X PCS/PMA or SGMII core netlist is generated by the CORE Generator™ software to allow simulation of the core in the design phase of the project.

Using the Simulation Model

For information about setting up your simulator to use the pre-implemented model, please consult the Xilinx *Synthesis and Verification Design Guide*, included in your Xilinx software installation.

The model is provided in the CORE Generator software project directory.

VHDL Design Entry

`<component_name>.vhd`

Verilog Design Entry

`<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 Ethernet 1000BASE-X PCS/PMA or SGMII core into your VHDL source.

After the entire design is complete, create the following:

- An XST project file *top_level_module_name*`.prj` listing all the user 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 more information on creating project and synthesis script files, and running the `xst` program.

XST - Verilog

There is a module declaration for the Ethernet 1000BASE-X PCS/PMA or SGMII core in the CORE Generator software project directory:

`<component_name>/implement/<component_name>_mod.v`

Use this module to help instance the Ethernet 1000BASE-X PCS/PMA or SGMII core into your Verilog source.

After the entire design is complete, do the following:

- Generate an XST project file *top_level_module_name*`.prj` listing all user source code files.

  Make sure to include the following as the first two files in the project list.

  ```
  %XILINX%/verilog/src/iSE/unisim_comp.v
  ```

  and

  ```
  <component_name>/implement/component_name_mod.v
  ```

- Generate 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 more information on creating project and synthesis script files, and running the `xst` program.

Implementation

Generating the Xilinx Netlist

To generate the Xilinx netlist, the ngdbuild tool is used to translate and merge the individual design netlists into a single design database—the NGD file. Also merged at this stage is the UCF for the design. An example of the `ngdbuild` command is:

```
$ ngdbuild -sd path_to_core_netlist -sd path_to_user_synth_results \
   -uc top_level_module_name.ucf top_level_module_name
```

Mapping the Design

To map the logic gates of the user design netlist into the CLBs and IOBs of the FPGA, run the `map` command. The `map` command writes out a physical design to an NCD file. An example of the `map` command is:

```
$ map -o top_level_module_name_map.ncd top_level_module_name.ngd \
   top_level_module_name.pcf
```

Placing and Routing the Design

The `par` command must be executed to place and route your design logic components (mapped physical logic cells) within an NCD file, in accordance with the layout and timing requirements specified within the PCF file. The `par` command outputs the placed and routed physical design to an NCD file.

An example of the `par` command is:

```
$ par top_level_module_name_map.ncd top_level_module_name.ncd \
    top_level_module_name.pcf
```

Static Timing Analysis

The `trce` command must be executed to evaluate timing closure on a design and create a Timing Report file (TWR) that is derived from static timing analysis of the Physical Design file (NCD). 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 \
    top_level_module_name.pcf
```

Generating a Bitstream

The `bitgen` command must be executed to create the configuration bitstream (BIT) file based on the contents of a physical implementation file (NCD). 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, the netgen command must be run.

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 about setting up your simulator to use the pre-implemented model, please consult the Xilinx *Synthesis and Verification Design Guide*, included in your Xilinx software installation.

In addition, use the following guidelines to determine the simulator type required:

Designs incorporating a device-specific transceiver require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. Currently supported simulators are:

- Mentor Graphics ModelSim v6.4b and above
- Cadence IUS v8.1 -s009 and above
- Synopsys 2008.09 and above

For VHDL simulation, a mixed HDL license is required.

Other Implementation Information

For more information about using the Xilinx implementation tool flow, including command line switches and options, consult the software manuals provided with the Xilinx ISE® software.

XILINX®

Appendix A

Core Verification, Compliance, and Interoperability

Verification

The Ethernet 1000BASE-X PCS/PMA or SGMII core has been verified with extensive simulation and hardware verification.

Simulation

A highly parameterizable transaction based test bench was used to test the core. Testing included the following:

- Register Access
- Loss of Synchronization
- Auto-Negotiation and error handling
- Frame Transmission and error handling
- Frame Reception and error handling
- Clock Compensation in the Elastic Buffers

Hardware Verification

The core has been tested in a variety of hardware test platforms at Xilinx to represent different parameterizations, including the following:

- The core with device-specific transceiver and performing the 1000BASE-X standard was tested with the Tri-Mode Ethernet MAC core from Xilinx.

  This follows the architecture shown in Figure 13-3. A test platform was built around these cores, including a back-end FIFO capable of performing a simple ping function, and a test pattern generator. Software running on the embedded PowerPC® processor was used to provide access to all configuration and status registers. Version 3.0 of this core was taken to the University of New Hampshire Interoperability Lab (UNH IOL) where conformance and interoperability testing was performed.

- The core with device-specific transceiver (all supported families) and performing the SGMII standard was tested with the Tri-speed Ethernet MAC core from Xilinx.

  This was connected to an external PHY capable of performing 10BASE-T, 100BASE-T and 1000BASE-T. The system was tested at all three speeds, following the architecture shown in Figure 13-9 and included the PowerPC processor based test platform.

XILINX®

Appendix B

Core Latency

Core Latency

The standalone core does not meet all the latency requirements specified in *IEEE 802.3* due to the latency of the Elastic Buffers in both TBI and device-specific transceiver versions. However, the core may be used for backplane and other applications where strict adherence to the IEEE latency specification is not required.

Where strict adherence to the *IEEE 802.3* specification is required, the core may be used with an Ethernet MAC core that is within the IEEE specified latency for a MAC sublayer. For example, when the core is connected to the Xilinx Tri-Mode Ethernet MAC core, the system as a whole is compliant with the overall *IEEE 802.3* latency specifications.

Latency for 1000BASE-X PCS with TBI

The following measurements are for the core only, and do not include any IOB registers or the Transmitter Elastic Buffer added in the example design.

Transmit Path Latency

As measured from a data octet input into `gmii_txd[7:0]` of the transmitter side GMII until that data appears on `tx_code_group[9:0]` on the TBI interface, the latency through the core in the transmit direction is 5 clock periods of `gtx_clk`.

Receive Path Latency

Measured from a data octet input into the core on `rx_code_group0[9:0]` or `rx_code_group1[9:0]` from the TBI interface (until that data appears on `gmii_rxd[7:0]` of the receiver side GMII), the latency through the core in the receive direction is equal to 16 clock periods of `gtx_clk`, plus an additional number of clock cycles equal to the current value of the Receiver Elastic Buffer.

The Receiver Elastic Buffer is 32 words deep. The nominal occupancy will be at half-full, thereby creating a nominal latency through the receiver side of the core equal to 16 + 16= 32 clock cycles of `gtx_clk`.

Latency for 1000BASE-X PCS and PMA Using a Transceiver

These measurements are for the core only–they do not include the latency through the Virtex®-4 FPGA MGT, Virtex-5 FPGA GTP, Virtex-5 FPGA GTX RocketIO™ transceiver, Virtex-6 FPGA GTX transceiver, Spartan®-6 FPGA GTP transceiver or the Transmitter Elastic Buffer added in the example design.

Transmit Path Latency

As measured from a data octet input into `gmii_txd[7:0]` of the transmitter side GMII (until that data appears on `txdata[7:0]` on the MGT interface), the latency through the core in the transmit direction is 4 clock periods of `userclk2`.

Receive Path Latency

As measured from a data octet input into the core on `rxdata[7:0]` from the MGT interface (until that data appears on `gmii_rxd[7:0]` of the receiver side GMII), the latency through the core in the receive direction is 6 clock periods of `userclk2`.

Latency for SGMII

When performing the SGMII standard, the core latency figures are identical to the Latency for 1000BASE-X PCS and PMA using the MGT. Again these figures do not include the latency through the MGT or any Elastic Buffers added in the example design.

XILINX®

Appendix C

Calculating the DCM Fixed Phase Shift Value

Requirement for DCM Phase Shifting

A DCM is used in the clock path to meet the input setup and hold requirements when using the core with a TBI (see Chapter 6, "The Ten-Bit Interface") and with an external GMII implementation in Spartan®-3, Spartan-3E, Spartan-3A/3AN/3A DSP devices (see "Spartan-3, Spartan-3E, Spartan-3A/3A DSP and Virtex-4 Devices," page 71).

In these cases, a fixed phase shift offset is applied to the DCM to skew the clock. This will initiate a static alignment by using the clock DCM to shift the internal version of the clock so that its edges are centered on the data eye at the IOB DDR flip-flops. The ability to shift the internal clock in small increments is critical for sampling high-speed source synchronous signals such as TBI and GMII. For statically aligned systems, the DCM 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/TBI data bus and control signals.

Determine the best DCM setting (phase shift) to ensure that the target system has the maximum system margin required to perform across voltage, temperature, and process (multiple chips) variations. Testing the system to determine the best DCM 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:

System Margin (ps) = UI(ps) * (working phase shift range/128)

Finding the Ideal Phase Shift Value for Your System

Xilinx cannot recommend a singular phase shift value that is effective across all hardware platforms. 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 UCF is a placeholder, and works 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 DCM 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 fewer, 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. Once 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 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.

XILINX®

Appendix D

1000BASE-X State Machines

This appendix is intended to serve as a reference for the basic operation of the 1000BASE-X *IEEE 802.3* clause 36 transmitter and receiver state machines.

Introduction

Table D-1 illustrates the Ordered Sets defined in *IEEE 802.3* clause 36. These code group characters are inserted by the PCS Transmit Engine into the transmitted data stream, encapsulating the Ethernet frames indicated via the GMII transmit signals.

The PCS Receive Engine performs the opposite function; it uses the Ordered Sets to detect the Ethernet frames and from them creates the GMII receive signals.

Cross reference Table D-1 with the remainder of this Appendix. See *IEEE 802.3* clause 36 for further information on these Orders Sets.

Table D-1: **Defined Ordered Sets**

Code	Ordered_Set	No. of Code-Groups	Encoding
/C/	**Configuration**		Alternating /C1/ and /C2/
/C1/	Configuration 1	4	/K28.5/D21.5/Config_Reg[1]
/C2/	Configuration 2	4	/K28.5/D2.2/Config_Reg[1]
/I/	**IDLE**		Correcting /I1/, Preserving /I2/
/I1/	IDLE_1	2	/K28.5/D5.6/
/I2/	IDLE_2	2	/K28.5/D16.2/
	Encapsulation		
/R/	Carrier_Extend	1	/K23.7/
/S/	Start_of_Packet	1	/K27.7/
/T/	End_of_Packet	1	/K29.7/
/V/	Error_Propagation	1	/K30.7/

1. Two data code-groups representing the Config_Reg value (contains Auto-Negotiation information)

Start of Frame Encoding

The Even Transmission Case

Figure D-1 illustrates the translation of GMII encoding into the code-group stream performed by the PCS Transmit Engine. This stream is transmitted out of the core, either serially using the device-specific transceiver or in parallel across the TBI.

It is important to note that the encoding of Idle periods /I2/ is constructed from a couple of code groups—the /K28.5/ character (considered the *even* position) and the /D16.2/ character (considered the *odd* position). In this example, the assertion of the `gmii_tx_en` signal of the GMII occurs in the even position. In response, the state machines insert a Start of Packet code group /S/ following the Idle (in the *even* position). This is inserted in place of the first byte of the frame preamble field.

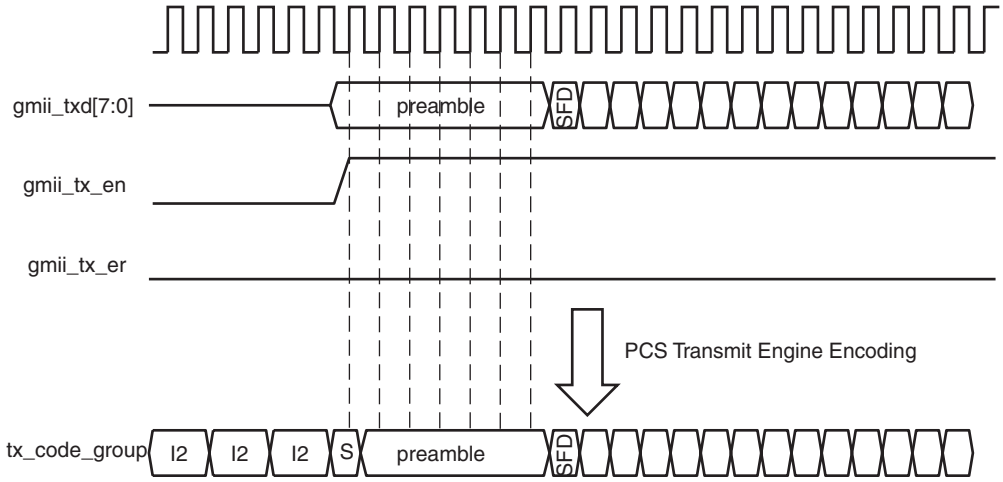


Figure D-1: **1000BASE-X Transmit State Machine Operation (Even Case)**

Reception of the Even Case

Figure D-2 illustrates the reception of the in-bound code-group stream, received either serially using the device-specific transceiver, or in parallel across the TBI, and translation of this code-group stream into the receiver GMII. This is performed by the PCS Receive Engine.

The Start of Packet code group /S/ is replaced with a preamble byte. This results in the restoration of the full preamble field.

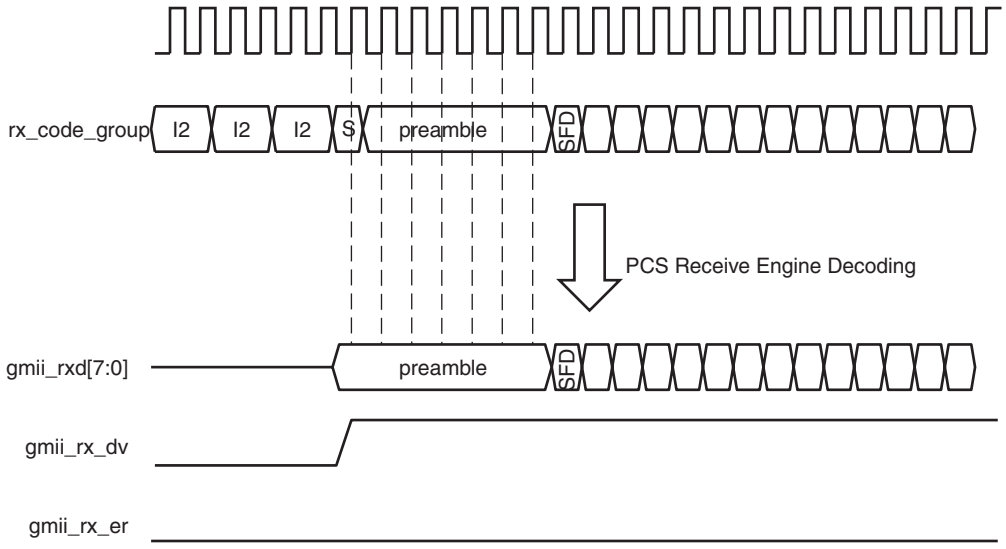


Figure D-2: **1000BASE-X Reception State Machine Operation (Even Case)**

The Odd Transmission Case

Figure D-3 illustrates the translation of GMII encoding into the code-group stream performed by the PCS Transmit Engine; this stream is transmitted out of the core, either serially using the device-specific transceiver, or in parallel across the TBI.

In this example, the assertion of the `gmii_tx_en` signal of the GMII occurs in the *odd* position; in response, the state machines are unable to immediately insert a Start-Of-Packet code group /S/ as the Idle character must first be completed. The Start of Packet code group /S/ is therefore inserted (in the *even* position) after completing the Idle. This results in the /D16.2/ character of the Idle /I2/ sequence being inserted in place of the first byte of the preamble field, and the Start-Of-Packet /S/ being inserted in place of the second byte of preamble as illustrated.

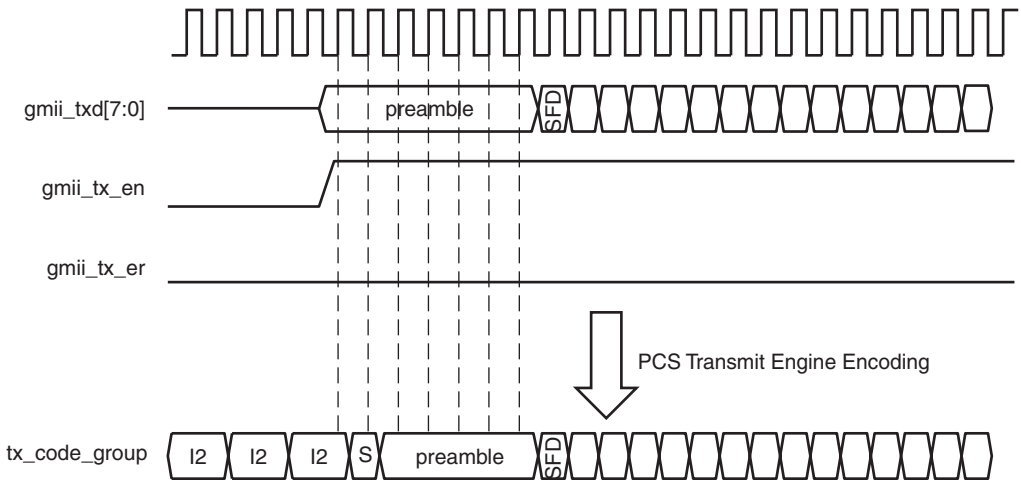


Figure D-3: **1000BASE-X Transmit State Machine Operation (Odd Case)**

Reception of the Odd Case

Figure D-4 illustrates the reception of the in-bound code-group stream, received either serially using the device-specific transceiver, or in parallel across the TBI, and translation of this code-group stream into the receiver GMII. This is performed by the PCS Receive Engine.

The Start of Packet code group /S/ is again replaced with a preamble byte. However, the first preamble byte of the original transmit GMII (see Figure D-3) frame (which was replaced with the /D16.2/ character to complete the Idle sequence), has not been replaced. This has resulted in a single byte of preamble loss across the system.

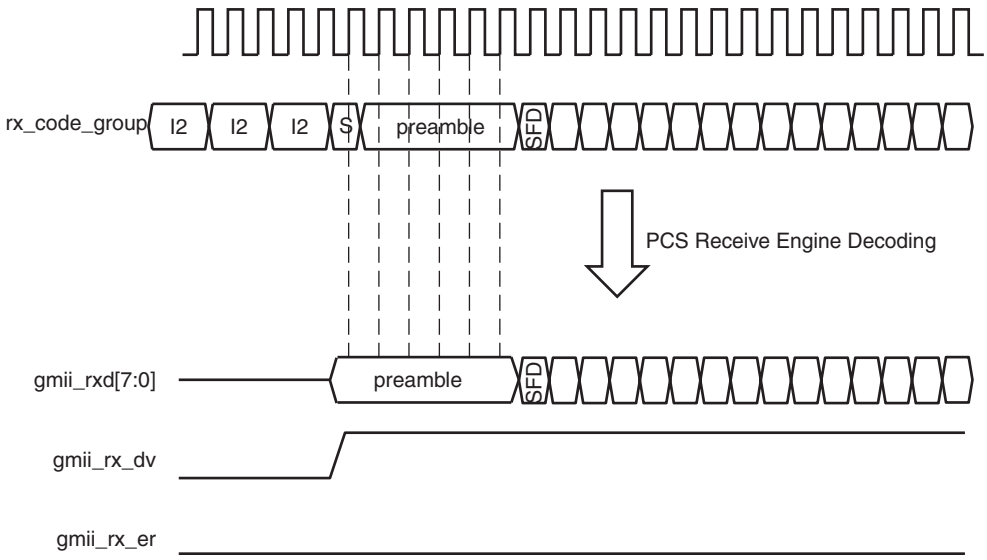


Figure D-4: **1000BASE-X Reception State Machine Operation (Odd Case)**

Preamble Shrinkage

As previously described, a single byte of preamble can be lost across the 1000BASE-X system (the actual loss occurs in the 1000BASE-X PCS transmitter state machine).

- There is no specific statement for this preamble loss in the *IEEE 802.3-2002* specification; the preamble loss falls out as a consequence of the state machines (see figures 36-5 and 36-6 in the *IEEE 802.3-2002* specification).
- *IEEE 802.3ah-2004* does, however, specifically state in clause 65.1.3.2.1:

  "NOTE 1 – The 1000BASE-X PCS transmit function replaces the first octet of preamble with the /S/ code-group or it discards the first octet and replaces the second octet of preamble with the /S/ code-group. This decision is based upon the even or odd alignment of the PCS transmit state diagram (see Figure 36-5)."

End of Frame Encoding

The Even Transmission Case

Figure D-5 illustrates the translation of GMII encoding into the code-group stream performed by the PCS Transmit Engine. This stream is transmitted out of the core, either serially using the device-specific transceiver or in parallel across the TBI.

In response to the deassertion of `gmii_tx_en`, an End of Packet code group /T/ is immediately inserted. The even and odd alignment described in "Start of Frame Encoding" persists throughout the Ethernet frame. If the /T/ character occurs in the even position (the frame contained an even number of bytes starting from the /S/ character), then this is followed with a single Carrier Extend code group /R/. This allows the /K28.5/ character of the following Idle code group to be aligned to the even position.

Note: The first Idle to follow the frame termination sequence will be a /I1/ if the frame ended with positive running disparity or a /I2/ if the frame ended with negative running disparity. This is illustrated as the shaded code group.

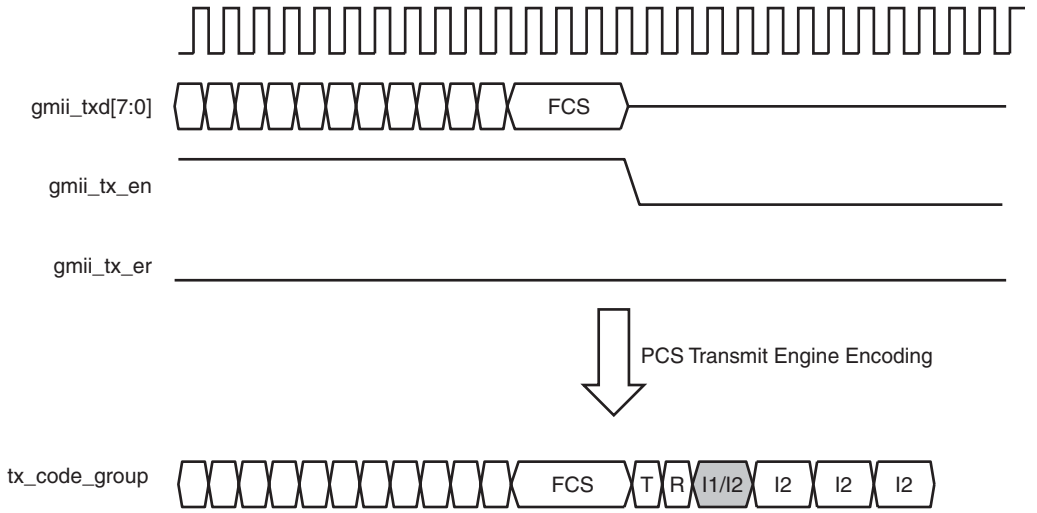


Figure D-5: **1000BASE-X Transmit State Machine Operation (Even Case)**

Reception of the Even Case

Figure D-6 illustrates the reception of the in-bound code-group stream, received either serially using the device-specific transceiver, or in parallel across the TBI, and translation of this code-group stream into the receiver GMII. This is performed by the PCS Receive Engine.

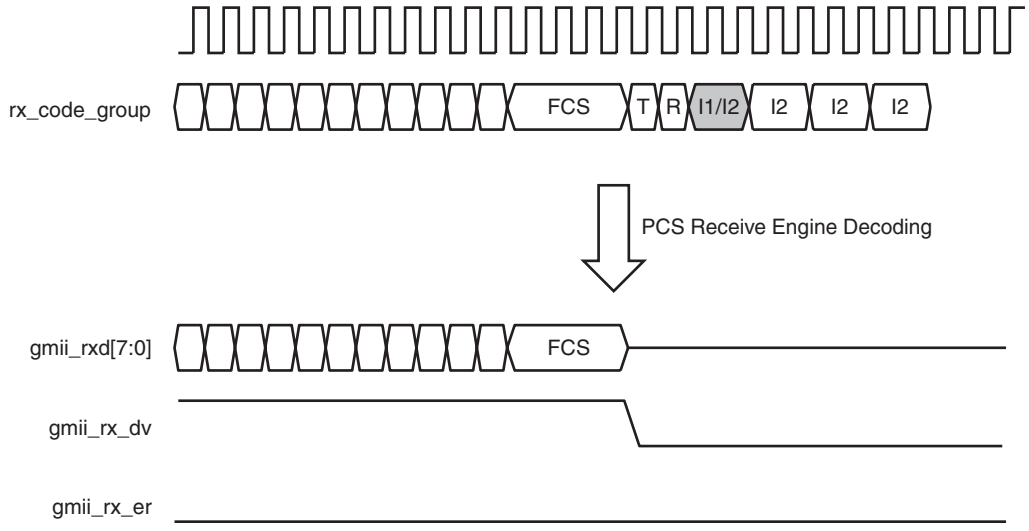


Figure D-6: **1000BASE-X Reception State Machine Operation (Even Case)**

The Odd Transmission Case

Figure D-7 illustrates the translation of GMII encoding into the code-group stream performed by the PCS Transmit Engine; this stream is transmitted out of the core, either serially using the device-specific transceiver, or in parallel across the TBI.

In response to the deassertion of `gmii_tx_en`, an End of Packet code group /T/ is immediately inserted. The even and odd alignment described in "Start of Frame Encoding" persists throughout the Ethernet frame. If the /T/ character occurs in the odd position (the frame contained an odd number of bytes starting from the /S/ character), then this is followed with two Carrier Extend code groups /R/. This allows the /K28.5/ character of the following Idle code group to be aligned to the even position.

Note: The first Idle to follow the frame termination sequence will be a /I1/ if the frame ended with positive running disparity or a /I2/ if the frame ended with negative running disparity. This is illustrated as the shaded code group.

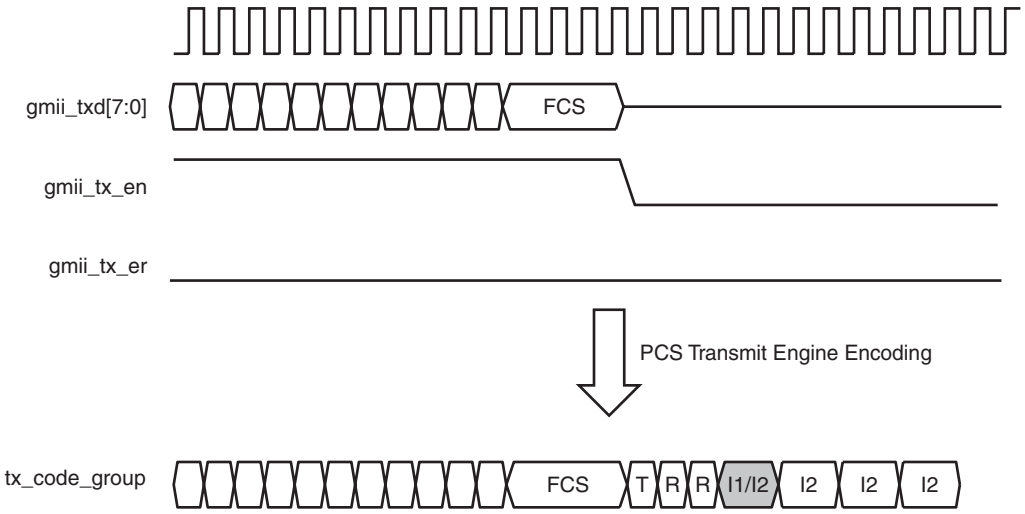


Figure D-7: **1000BASE-X Transmit State Machine Operation (Even Case)**

Reception of the Odd Case

Figure D-8 illustrates the reception of the in-bound code-group stream, received either serially using the device-specific transceiver, or in parallel across the TBI, and translation of this code-group stream into the receiver GMII. This is performed by the PCS Receive Engine.

As defined in *IEEE 802.3* figure 36-7b, the combined /T/R/R/ sequence results in the GMII encoding of Frame Extension. This occurs for a single clock cycle following the end of frame reception; the `gmii_rx_er` signal is driven high and the frame extension code of 0x0F is driven onto `gmii_rxd[7:0]`. This occurs even in full-duplex mode.

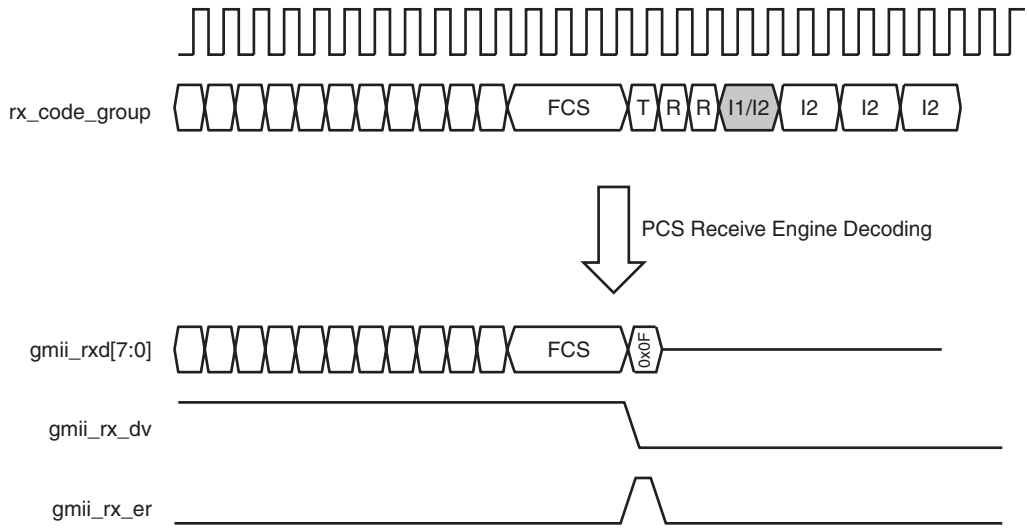


Figure D-8: **1000BASE-X Reception State Machine Operation (Odd Case)**

Appendix E

Rx Elastic Buffer Specifications

This appendix is intended to serve as a reference for the Rx Elastic Buffer sizes used in the core and the related maximum frame sizes that can be used without causing a buffer underflow or overflow error.

Throughout this appendix, all analyses are based on 100 ppm clock tolerances on both sides of the elastic buffer (200 ppm total difference). This corresponds to the Ethernet clock tolerance specification.

Introduction

The need for an Rx Elastic Buffer is illustrated in "The Requirement for the FPGA Fabric Rx Elastic Buffer" in Chapter 8. The analysis included in this chapter shows that for standard Ethernet clock tolerances (100 ppm) there can be a maximum difference of one clock edge every 5000 clock periods of the nominal 125 MHz clock frequency.

This slight difference in clock frequency on either side of the buffer will accumulate and either start to fill or empty the Rx Elastic Buffer over time. The Rx Elastic buffer copes with this by performing clock correction during the interframe gaps by either inserting or removing Idle characters. The Rx Elastic Buffer will always attempt to restore the buffer occupancy to the half full level during an interframe gap. See "Clock Correction," page 262.

Rx Elastic Buffers: Depths and Maximum Frame Sizes

Transceiver Rx Elastic Buffers

Figure E-1 illustrates the transceiver Rx Elastic Buffer depths and thresholds in Virtex®-4 FX, Virtex-5 LXT, SXT, FXT and TXT families, Spartan®-6 LXT family. Each FIFO word corresponds to a single character of data (equivalent to a single byte of data following 8B10B decoding).

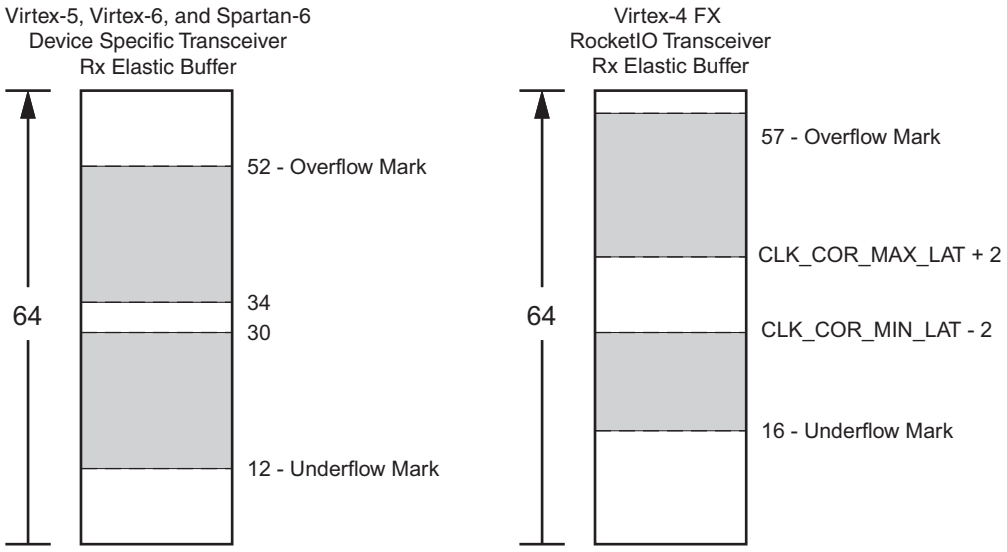


Figure E-1: **Elastic Buffer Sizes for all Transceiver Families**

Virtex-5, Virtex-6 and Spartan-6 Devices

Consider the example, where the shaded area represents the usable buffer availability for the duration of frame reception.

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

This analysis assumes that the buffer is approximately at the half-full level at the start of the frame reception. As illustrated, there are two locations of uncertainty, above and below the exact half-full mark of 32, resulting from the clock correction decision, and is based across an asynchronous boundary.

Because there is a worst-case scenario of one clock edge difference every 5000 clock periods, the maximum number of clock cycles (bytes) that can exist in a single frame passing through the buffer before an error occurs is:

```
5000 x 18 = 90000 bytes
```

Table E-1 translates this into maximum frame size at different Ethernet speeds. At SGMII speeds lower than 1 Gbps, performance is diminished because bytes are repeated multiple times (see "Designing with Client-side GMII for the SGMII Standard" in Chapter 5).

Table E-1: **Maximum Frame Sizes: Transceiver Rx Elastic Buffers (100ppm Clock Tolerance)**

Standard / Speed	Maximum Frame Size
1000BASE-X (1 Gbps only)	90000
SGMII (1 Gbps)	90000
SGMII (100 Mbps)	9000
SGMII (10 Mbps)	900

Virtex-4 FX Device

Consider the Virtex-4 FX device case also illustrated in Figure E-1. The thresholds are different to that of other families, but the overall size of the buffer is the same. Instead of the half full point, there are configurable clock correction thresholds. During the interframe gap, clock correction will attempt to restore the occupancy to within these two thresholds.

However, by setting both `CLK_COR_MAX_LAT` and `CLK_COR_MIN_LAT` thresholds to the same value, symmetrically between overflow and underflow marks, it is possible to obtain the same figures as for other families. For this reason, by adjusting the threshold attributes accordingly, Table E-1 is also applicable.

SGMII Fabric Rx Elastic Buffer

Figure E-2 illustrates the alternative FPGA fabric Rx Elastic Buffer depth and thresholds in Virtex-4 FX, Virtex-5 LXT, Virtex-5 SXT, Virtex-6 and Spartan-6 LXT device families. Each FIFO word corresponds to a single character of data (equivalent to a single byte of data following 8B10B decoding). This buffer can optionally be used to replace the Rx Elastic Buffers of the transceiver (see "Receiver Elastic Buffer Implementations" in Chapter 8).

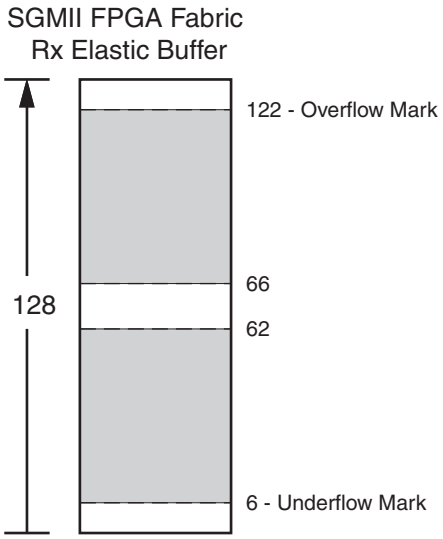


Figure E-2: **Elastic Buffer Size for all Transceiver Families**

The shaded area of Figure E-2 represents the usable buffer availability for the duration of frame reception.

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

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

Because there is a worst-case scenario of one clock edge difference every 5000 clock periods, the maximum number of clock cycles (bytes) that can exist in a single frame passing through the buffer before an error occurs is:

```
5000 x 56 = 280000 bytes
```

Table E-2 translates this into maximum frame size at different Ethernet speeds. At SGMII speeds lower than 1 Gbps, performance is diminished because bytes are repeated multiple times (see "Designing with Client-side GMII for the SGMII Standard" in Chapter 5).

Table E-2: **Maximum Frame Sizes: Fabric Rx Elastic Buffers (100ppm Clock Tolerance)**

Standard / Speed	Maximum Frame Size
1000BASE-X (1 Gbps only)	280000
SGMII (1 Gbps)	280000
SGMII (100 Mbps)	28000
SGMII (10 Mbps)	2800

TBI Rx Elastic Buffer

For SGMII / Dynamic Switching

The Rx Elastic Buffer used for the SGMII or Dynamic Standards Switching is identical to the method use in "SGMII Fabric Rx Elastic Buffer."

For 1000BASE-X

Figure E-3 illustrates the Rx Elastic Buffer depth and thresholds when using the Ten-Bit-Interface with the 1000BASE-X standard. This buffer is intentionally smaller than the equivalent buffer for SGMII/Dynamic Switching; because a larger size is not required, the buffer is kept smaller to save logic and keep latency low. Each FIFO word corresponds to a single character of data (equivalent to a single byte of data following 8B10B decoding).

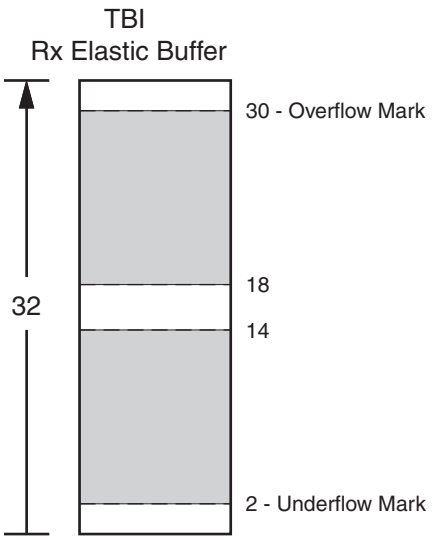


Figure E-3: **TBI Elastic Buffer Size for All Families**

The shaded area of Figure E-3 represents the usable buffer availability for the duration of frame reception.

- If the buffer is filling during frame reception, then there are 30-18 = 12 FIFO locations available before the buffer reaches the overflow mark.
- If the buffer is emptying during reception, then there are 14-2 = 12 FIFO locations available before the buffer reaches the underflow mark.

This analysis assumes that the buffer is approximately at the half-full level at the start of the frame reception. As illustrated, there are two locations of uncertainty above and below the exact half-full mark of 16. This is as a result of the clock correction decision, and is based across an asynchronous boundary.

Since there is a worst-case scenario of 1 clock edge difference every 5000 clock periods, the maximum number of clock cycles (bytes) that can exist in a single frame passing through the buffer before an error occurs is:

```
5000 x 12 = 60000 bytes
```

This translates into a maximum frame size of 60000 bytes.

Clock Correction

The calculations in all previous sections assumes that the Rx Elastic Buffers are restored to approximately half occupancy at the start of each frame. This is achieved by the elastic buffer performing clock correction during the interframe gaps either by inserting or removing Idle characters as required.

- If the Rx Elastic Buffer is emptying during frame reception, there are no restrictions on the number of Idle characters that can be inserted due to clock correction. The occupancy will be restored to half full and the assumption holds true.
- If the Rx Elastic Buffer is filling during frame reception, Idle characters need to be removed. Restrictions that need to be considered are described in the following sections.

Idle Character Removal at 1Gbps (1000BASE-X and SGMII)

The minimum number of clock cycles that may be presented to an Ethernet receiver, according to the *IEEE 802.3* specification, is 64-bit times at any Ethernet speed. At 1 Gbps 1000BASE-X and SGMII, this corresponds to 8 bytes (8 clock cycles) of interframe gap. However, an interframe gap consists of a variety of code groups, namely /T/, /R/, /I1/ and /I2/ characters (please see Appendix D, "1000BASE-X State Machines"). Of these, only /I2/ can be used as clock correction characters.

In a minimum interframe gap at 1 Gbps, we can only assume that two /I2/ characters are available for removal. This corresponds to 4 bytes of data.

Looking at this from another perspective, 4 bytes of data will need to be removed in an elastic buffer (which is filling during frame reception) for a frame which is 5000 x 4 = 20000 bytes in length. So if the frame being received is 20000 bytes in length or shorter, at 1 Gbps, we can assume that the occupancy of the elastic buffer will always self correct to half full before the start of the subsequent frame.

For frames which are longer than 20000 bytes, the assumption that the elastic buffer will be restored to half full occupancy does not hold true. For example, for a long stream of 250000 byte frames, each separated by a minimum interframe gap, the Rx Elastic Buffer will eventually fill and overflow. This is despite the 250000 byte frame length being less than the maximum frame size calculated in the "Rx Elastic Buffers: Depths and Maximum Frame Sizes" section.

However, since the legal maximum frame size for Ethernet frames is 1522 bytes (for a VLAN frame), idle character removal restrictions are not usually an issue.

Idle Character Removal at 100 Mbps (SGMII)

At SGMII, 100 Mbps, each byte is repeated 10 times. This also applies to the interframe gap period. For this reason, the minimum of 8 bytes for the 1 Gbps case corresponds to a minimum of 80 bytes for the 100 Mbps case.

Additionally, the majority of characters in this 80-byte interframe-gap period are going to be the /I2/ clock correction characters. Because of the clock correction circuitry design, a minimum of 20 /I2/ code groups will be available for removal. This translates into 40 bytes, giving a maximum run size of 40 x 5000 = 200000 bytes. Because each byte at 100 Mbps is repeated ten times, this corresponds to an Ethernet frame size of 20000 bytes, the same size as the 1 Gbps case.

So in summary, at 100 Mbps, for any frame size of 20000 bytes or less, it can still be assumed that the Elastic Buffer will return to half full occupancy before the start of the next frame. However, a frame size of 20000 is larger than can be received in the device-specific transceiver Elastic Buffer (see "Rx Elastic Buffers: Depths and Maximum Frame Sizes"). Only the SGMII fabric Rx Elastic buffer is large enough.

Idle Character Removal at 10 Mbps (SGMII)

Using a similar argument to the 100 Mbps case, it can be shown that clock correction circuitry can also cope with a frame size up to 20000 bytes. However, this is larger than the maximum frame size for any Elastic Buffer provided with the core (see "Rx Elastic Buffers: Depths and Maximum Frame Sizes").

Maximum Frame Sizes for Sustained Frame Reception

Sustained frame reception refers to the maximum size of frames which can be continuously received when each frame is separated by a minimum interframe gap.

The size of frames that can be reliably received is dependent on the two considerations previously introduced in this appendix:

- The size of the Elastic Buffer, see"Rx Elastic Buffers: Depths and Maximum Frame Sizes"
- The number of clock correction characters present in a minimum interframe gap, (see "Clock Correction")

Table E-3 summarizes the maximum frame sizes for sustained frame reception when used with the different Rx Elastic Buffers provided with the core. All frame sizes are provided in bytes.

Table E-3: **Maximum Frame Size: (Sustained Frame Reception) Capabilities of the Rx Elastic Buffers**

Ethernet Standard and Speed	Rx Elastic Buffer Type		
	TBI	Transceiver	SGMII Fabric Buffer
1000BASE-X (1 Gbps)	20000 (limited by clock correction)	20000 (limited by clock correction)	20000 (limited by clock correction)
SGMII 1 Gbps	20000 (limited by clock correction)	20000 (limited by clock correction)	20000 (limited by clock correction)
SGMII 100 Mbps	20000 (limited by clock correction)	9000 (limited by buffer size)	20000 (limited by clock correction)
SGMII 10 Mbps	2800 (limited by buffer size)	900 (limited by buffer size)	2800 (limited by buffer size)

Jumbo Frame Reception

A jumbo frame is an Ethernet frame which is deliberately larger than the maximum sized Ethernet frame allowed in the *IEEE 802.3* specification. The size of jumbo frames that can be reliably received is identical to the frame sizes defined in "Maximum Frame Sizes for Sustained Frame Reception," page 264.

XILINX®

Appendix F

Debugging Guide

This appendix provides assistance for debugging the core within a system. For additional help, contact Xilinx by submitting a WebCase at support.xilinx.com/.

General Checks

- Ensure that all the timing constraints for the core were met during Place and Route.
- Does it work in timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue.
- Ensure that all clock sources are clean. If using DCMs in the design, ensure that all DCMs have obtained lock by monitoring the `LOCKED` port.

Problems with the MDIO

- Ensure that the MDIO is driven properly. See "MDIO Management Interface," page 139 for detailed information about performing MDIO transactions.
- Check that the `mdc` clock is running and that the frequency is 2.5 MHz or less.
- Read from a configuration register that does not have all 0s as a default. If all 0s are read back, the read was unsuccessful. Check that the PHYAD field placed into the MDIO frame matches the value placed on the `phyad[4:0]` port of the core.

Problems with Data Reception or Transmission

When no data is being received or transmitted:

- Ensure that a valid link has been established between the core and its link partner, either by Auto-Negotiation or Manual Configuration: `status_vector[0]` and `status_vector[1]` should both be high. If no link has been established, see the topics discussed in the next section.
  - "Problems with Auto-Negotiation"
  - "Problems in Obtaining a Link (Auto-Negotiation Disabled)"

  Note: Transmission through the core is not allowed unless a link has been established. This behavior can be overridden by setting the Unidirectional Enable bit.
- Ensure that the Isolate state has been disabled.

  By default, the Isolate state is enabled after power-up. For an external GMII, the PHY will be electrically isolated from the GMII; for an internal GMII, it will behave as if it is isolated. This results in no data transfer across the GMII. See "Startup Sequencing," page 233 for more information.

If data is being transmitted and received between the core and its link partner, but with a high rate of packet loss, see "Problems with a High Bit Error Rate."

Problems with Auto-Negotiation

Determine whether Auto-Negotiation has completed successfully by doing one of the following.

- Poll the Auto-Negotiation completion bit 1.5 in "Status Register (Register 1)"
- Use the Auto-Negotiation interrupt port of the core (see "Using the Auto-Negotiation Interrupt," page 184)

If Auto-Negotiation is not completing:

1. Ensure that Auto-Negotiation is enabled in *both* the core and in the link partner (the device or test equipment connected to the core). Auto-Negotiation cannot complete successfully unless both devices are configured to perform Auto-Negotiation.

   The Auto-Negotiation procedure requires that the Auto-Negotiation handshaking protocol between the core and its link partner, which lasts for several link timer periods, occur without a bit error. A detected bit error will cause Auto-Negotiation to go back to the beginning and restart. Therefore, a link with an exceptionally high bit error rate may not be capable of completing Auto-Negotiation, or may lead to a long Auto-Negotiation period caused by the numerous Auto-Negotiation restarts. If this appears to be the case, try the next step and see "Problems with a High Bit Error Rate."
2. Try disabling Auto-Negotiation in both the core and the link partner and see if both devices report a valid link and are able to pass traffic. If they do, it proves that the core and link partner are otherwise configured correctly. If they do not pass traffic, see the next section, "Problems in Obtaining a Link (Auto-Negotiation Disabled)."

Problems in Obtaining a Link (Auto-Negotiation Disabled)

Determine whether the device has successfully obtained a link with its link partner by doing the following:

- Reading bit 1.2, Link Status, in "Status Register (Register 1)" when using the optional MDIO management interface (or look at `status_vector[1]`).
- Monitoring the state of `status_vector[0]`. If this is logic '1,' then synchronization, and therefore a link, has been established. See "Bit[0]: Link Status."

If the devices have failed to form a link then do the following:

- Ensure that Auto-Negotiation is disabled in *both* the core and in the link partner (the device or test equipment connected to the core).
- Monitor the state of the `signal_detect` signal input to the core. This should either be:
  - connected to an optical module to detect the presence of light. Logic '1' indicates that the optical module is correctly detecting light; logic '0' indicates a fault. Therefore, ensure that this is driven with the correct polarity.
  - Signal must be tied to logic '1' (if not connected to an optical module).

    Note: When signal_detect is set to logic '0,' this forces the receiver synchronization state machine of the core to remain in the loss of sync state.
  - See the following section, "Problems with a High Bit Error Rate."

Transceiver Specific

When using a device-specific transceiver, perform these additional checks:

- Ensure that the polarities of the TXN/TXP and RXN/RXP lines are not reversed. If they are, this can be easily fixed by using the TXPOLARITY and RXPOLARITY ports of the device-specific transceiver.
- Check that the device-specific transceiver is not being held in reset by monitoring the mgt_tx_reset and mgt_rx_reset signals between the core and the device-specific transceiver. If these are asserted then this indicates that the PMA PLL circuitry in the device-specific transceiver has not obtained lock; please check the PLL Lock signals output from the device-specific transceiver.
- Monitor the RXBUFERR signal when Auto-Negotiation is disabled. If this is being asserted, the Elastic Buffer in the receiver path of the device-specific transceiver is either under or overflowing. This indicates a clock correction problem caused by differences between the transmitting and receiving ends. Check all clock management circuitry and clock frequencies applied to the core and to the device-specific transceiver.

Problems with a High Bit Error Rate

Symptoms

The severity of a high-bit error rate can vary and cause any of the following symptoms:

- Failure to complete Auto-Negotiation when Auto-Negotiation is enabled.
- Failure to obtain a link when Auto-Negotiation is disabled in both the core and the link partner.
- High proportion of lost packets when passed between two connected devices that are capable of obtaining a link through Auto-Negotiation or otherwise. This can usually be accurately measured if the Ethernet MAC attached to the core contains statistic counters.

  Note: All bit errors detected by the 1000BASE-X PCS/PMA logic during frame reception will show up as Frame Check Sequence Errors in an attached Ethernet MAC.

Debugging

- Compare the problem across several devices or PCBs to ensure that the problem is not a one-off case.
- Try using an alternative link partner or test equipment and then compare results.
- Try putting the core into loopback (both by placing the core into internal loopback, and by looping back the optical cable) and compare the behavior. The core should always be capable of Auto-Negotiating with itself and looping back with itself from transmitter to receiver so direct comparisons can be made. If the core exhibits correct operation when placed into internal loopback, but not when loopback is performed via an optical cable, this may indicate a faulty optical module or a PCB problem.
- Try swapping the optical module on a misperforming device and repeat the tests.

Transceiver Specific Checks

Perform these additional checks when using a device-specific transceiver:

- Directly monitor the following ports of the device-specific transceiver by attaching error counters to them, or by triggering on them using the Chipscope™ tool or an external logic analyzer.

  `RXDISPERR`

  `RXNOTINTABLE`

  These signals should not be asserted over the duration of a few seconds, minutes or even hours. If they are frequently asserted, it may indicate a problem with the device-specific transceiver. Consult Answer Record 19699 for debugging device-specific transceiver issues.

- Place the device-specific transceiver into parallel or serial loopback.
  - If the core exhibits correct operation in device-specific transceiver serial loopback, but not when loopback is performed via an optical cable, it may indicate a faulty optical module.
  - If the core exhibits correct operation in device-specific transceiver parallel loopback but not in serial loopback, this may indicate a device-specific transceiver problem. See Answer Record 19699 for details.
- A mild form of bit error rate may be solved by adjusting the transmitter `TX_PREEMPHASIS`, `TX_DIFF_CTRL` and `TERMINATION_IMP` attributes of the device-specific transceiver.