Ethernet 1000BASE-X PCS/PMA or SGMII v14.3

LogiCORE IP Product Guide

Vivado Design Suite

PG047 October 1, 2014

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The Ethernet 1000BASE-X PCS/PMA or Serial Gigabit Media Independent Interface (SGMII) core provides a flexible solution for connection to an Ethernet Media Access Controller (MAC) or other custom logic. It supports two standards: the 1000BASE-X Physical Coding Sublayer (PCS) and Physical Medium Attachment (PMA) operation, as defined in the IEEE 802.3-2008 standard and the Gigabit Media Independent Interface (GMII) to Serial-GMII (SGMII) bridge or SGMII to GMII bridge, as defined in the *Serial-GMII Specification V1.7* (CISCO SYSTEMS, ENG-46158) [Ref 1]

Dynamic switching between 1000BASE-X and SGMII standards is also supported

Features

- Supported physical interfaces for 1000BASE-X and SGMII standards
- Integrated device-specific transceiver interface using one of the following:
- Support for SGMII over Select Input/Output (I/O) Low Voltage Differential Signaling (LVDS) in UltraScale architecture, Zynq-7000, and 7 series devices
- Configured and monitored through MDIO
- 1000BASE-X and SGMII Auto-Negotiation supported
- Support for full duplex only

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale™ Architecture, Zynq®-7000 All Programmable SoC, 7 Series
Supported User Interfaces	GMII(2)
Resources	See Table 2-6 through Table 2-21.
Provided with Core	
Design Files	Encrypted RTL
Example Designs	Verilog and VHDL
Test Bench	Demonstration Test Bench
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Verilog and VHDL
Supported S/W Driver	NA
Tested Design Flows(3)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx, Inc.@ www.xilinx.com/support	

1. For a complete list of supported devices, see the Vivado IP catalog. For supported family configurations see Table 2-1. For supported speed grades see Speed Grades.
2. MII is supported only when used with EMAC0/EMAC1 present in Zynq-7000 AP SoC processor subsystem (PS).
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

This product guide provides information for generating a 1000BASE-X Physical Coding Sublayer/Physical Medium Attachment (PCS/PMA) or a Serial Gigabit Media Independent Interface (SGMII) core, customizing and simulating the core using the provided example design, and running the design files through implementation using the Xilinx tools. The two standards supported are sufficiently similar to be supported in the same core.

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

For detailed information about the core, see the Ethernet 100BASE-X PCS/PMA or SGMII product page.

Applications

The core can be used for applications using the Ethernet 1000BASE-X standard or the SGMII standard.

Ethernet 1000BASE-X

Figure 1-1 shows a typical application for the core meeting the 1000BASE-X standard using a device-specific transceiver to provide the Physical Coding Sublayer (PCS) and Physical Medium Attachment (PMA) sublayers for 1 Gigabit Ethernet.

- The PMA is connected to an external off-the-shelf Gigabit Interface Converter (GBIC) or Small Form-Factor Pluggable (SFP) optical transceiver to complete the Ethernet port.
- The GMII of the Ethernet 1000BASE-X PCS/PMA is connected to an embedded Ethernet Media Access Controller (MAC), for example, the Xilinx Tri-Mode Ethernet MAC core in all supported devices or Ethernet MAC (EMAC0 or EMAC1) present in the Zynq®-7000 AP SoC processor subsystem (PS).

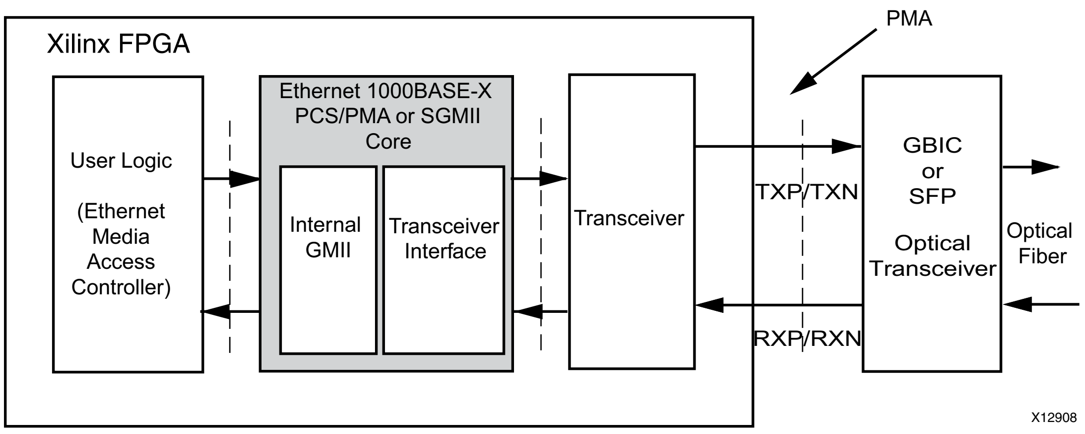


Figure 1-1: **Typical 1000BASE-X Application**

SGMII

The core can operate in two SGMII modes:

GMII to SGMII Bridge

Figure 1-2 shows a typical application for the core, where the core is providing a GMII to SGMII bridge using a device-specific transceiver to provide the serial interface.

- The device-specific transceiver is connected to an external off-the-shelf Ethernet PHY device that also supports SGMII. (This can be a tri-mode PHY providing 10BASE-T, 100BASE-T, and 1000BASE-T operation.)
- The core GMII interface is connected to an embedded Ethernet MAC, for example, the Xilinx Tri-Mode Ethernet MAC core (in supported devices) or Ethernet MAC (EMAC0 or EMAC1) present in the Zynq-7000 AP SoC processor subsystem (PS).

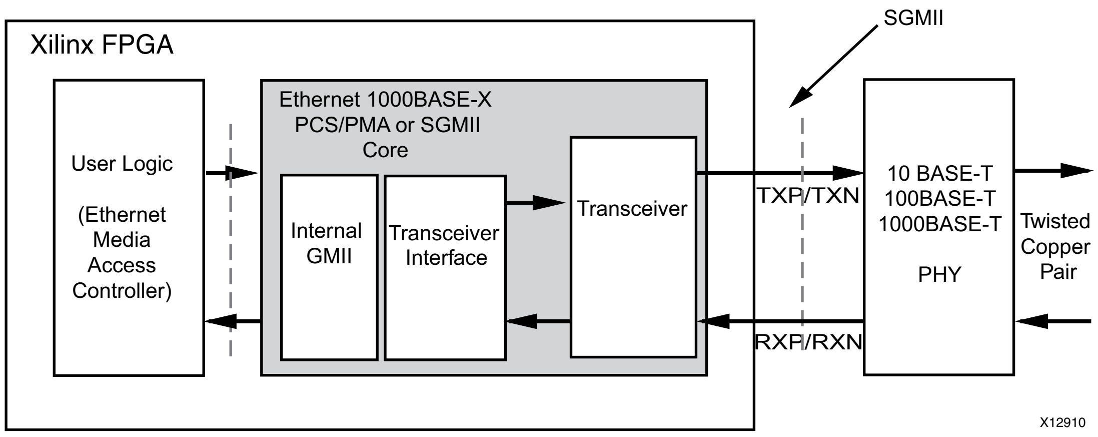


Figure 1-2: **Typical Application for GMII to SGMII Bridge Mode**

SGMII to GMII Bridge

Figure 1-3 shows a typical application for the core, where the core is providing a SGMII to GMII bridge using a device-specific transceiver to provide the serial interface.

- The device-specific transceiver is connected to an external off-the-shelf Ethernet MAC device that also supports SGMII. (This can be the Xilinx Tri-Mode Ethernet MAC core connected to 1000BASE-X PCS/PMA or SGMII core operating in GMII to SGMII mode)
- The GMII core interface is connected to a tri-mode PHY providing 10BASE-T, 100BASE-T, and 1000BASE-T operation.

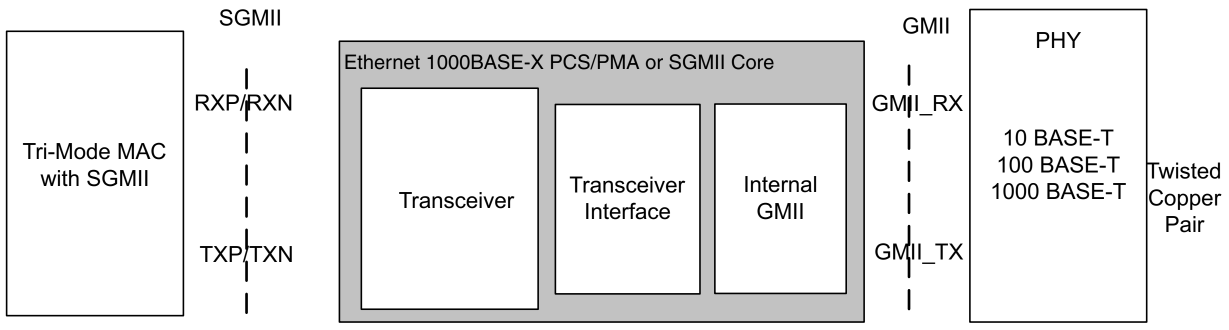


Figure 1-3: **Typical Application for SGMII to GMII Bridge Mode**

1000BASE-X PCS/PMA or SGMII Using a Device-Specific Transceiver

Using the core with a device-specific transceiver provides the functionality to implement the 1000BASE-X PCS and PMA sublayers. Alternatively, it can be used to provide a GMII to SGMII bridge.

The core interfaces to a device-specific transceiver, which provides some of the PCS layer functionality such as 8B/10B encoding/decoding, the PMA Serializer/Deserializer (SerDes), and clock recovery. Figure 1-4 shows the PCS sublayer functionality and the major functional blocks of the core. A description of the functional blocks and signals is provided in subsequent sections.

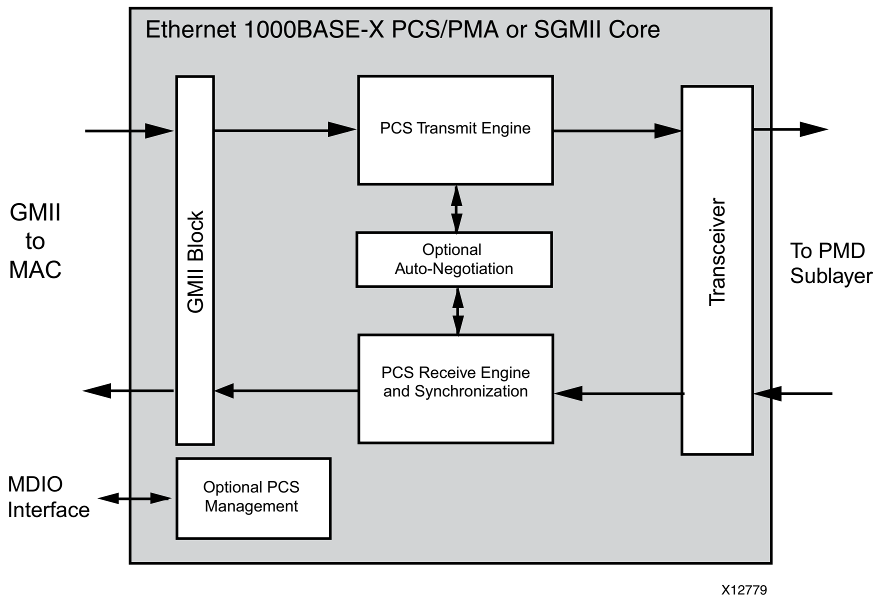


Figure 1-4: **Core Block Diagram Using a Device-Specific Transceiver**

GMII Block

The core provides a client-side GMII. This can be used as an internal interface for connection to an integrated Ethernet MAC or other custom logic. Alternatively, the core GMII can be routed to device Input/Output Blocks (IOBs) to provide an off-chip GMII.

Zynq-7000 and 7 series devices support GMII at 3.3V or lower only in certain parts and packages.

Note: See the *7 Series FPGAs SelectIO Resources User Guide* (UG471) [Ref 2] for I/O voltages supported by the devices. Some devices do not support 3.3V on pads. See the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 3].

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 802.3-2008 (Figures 36-5 and 36-6).

PCS Receive Engine and Synchronization

The synchronization process implements the state diagram of IEEE 802.3-2008 (Figure 36-9). The PCS receive engine converts the sequence of ordered sets to GMII data octets by implementing the state diagrams of IEEE 802.3-2008, Figures 36-7a and 36-7b.

Optional Auto-Negotiation Block

IEEE 802.3-2008 clause 37 describes the 1000BASE-X Auto-Negotiation function that allows a device to advertise the supported modes of operation to a device at the remote end of a link segment (link partner), and to detect corresponding operational modes that the link partner might be advertising. Auto-Negotiation is controlled and monitored through the PCS management registers.

Optional PCS Management Registers

Configuration and status of the core, including access to and from the optional Auto-Negotiation function, is performed with the 1000BASE-X PCS management registers as defined in IEEE 802.3-2008 clause 37. These registers are accessed through the MDIO, defined in IEEE 802.3-2008 clause 22, as if it were an externally connected PHY.

The PCS management registers can be omitted from the core. In this situation, configuration and status is made possible by using configuration vector and status signals. The configuration interface is provided to program Control register (Register 0) and Auto-Negotiation advertisement register (Register 4) independent of the MDIO interface. Bits corresponding to Remote fault and Pause in Register 5 are also part of Status vector.

When the core is using the SGMII standard, the register information is defined differently.

Transceiver

The transceiver is a device-specific transceiver which provides PCS layer functionality such as 8B/10B encoding/decoding and PMA parallel to serial/serial to parallel conversion to interface to the PMD.

Synchronous SGMII over LVDS

The core can fully support SGMII using standard LVDS SelectIO technology logic resources. This enables direct connection to external PHY devices without the use of an FPGA Transceiver. This implementation is shown in Figure 1-5.

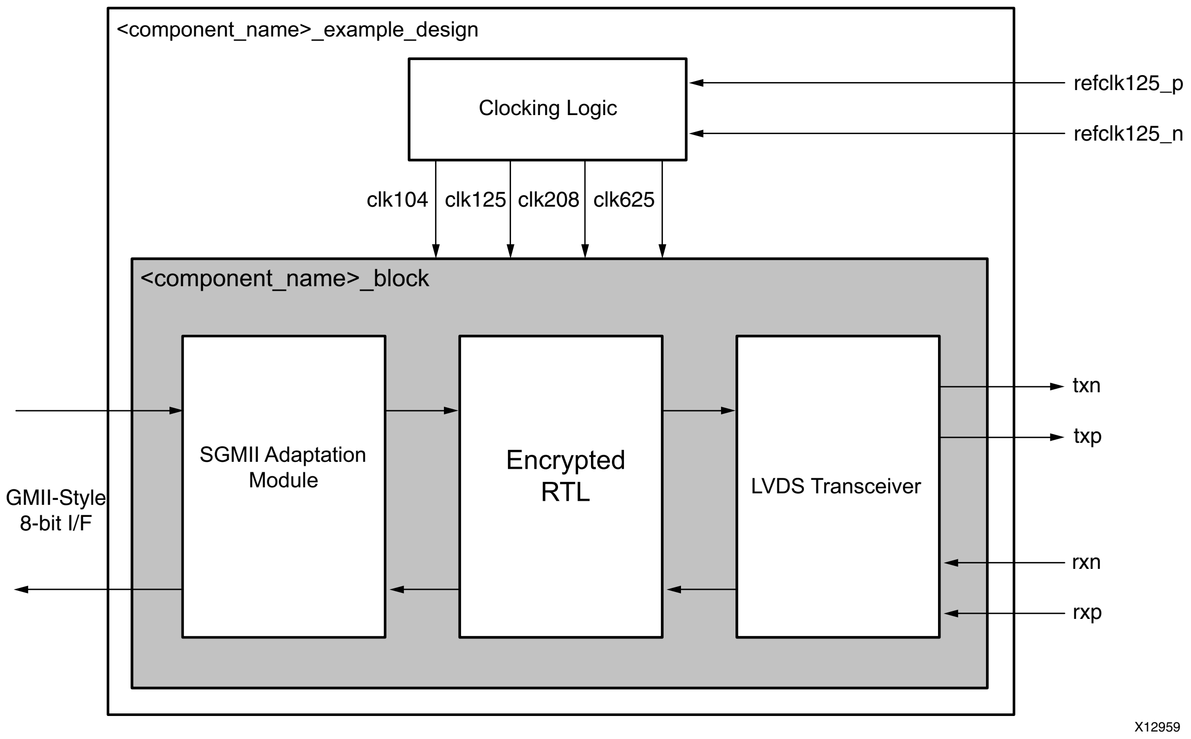


Figure 1-5: **Core Block Diagram with Standard SelectIO Technology Support for SGMII**

Note: For UltraScale™ architecture devices clocking logic generates 125, 312.5 and 625 MHz clocks respectively. Frequencies shown in Figure 1-5 are applicable for 7 series devices.

The core netlist in this implementation is identical to that of Figure 1-4 and all core netlist blocks are identical to those described in 1000BASE-X PCS/PMA or SGMII Using a Device-Specific Transceiver.

As shown in Figure 1-5, the Hardware Description Language (HDL) example design for this implementation provides additional logic to form the LVDS transceiver. The LVDS transceiver block fully replaces the functionality otherwise provided by an UltraScale architecture or 7 series FPGA GTP/GTX/GTH transceiver. This is only possible at a serial line rate of 1.25 Gb/s. The following subsections describe the design requirements.

SGMII Only

The interface implemented using this method supports SGMII between the FPGA and an external PHY device; the interface cannot directly support 1000BASE-X.

Supported Devices

- Kintex-7 devices, -2 speed grade or faster for devices with HR Banks or -1 speed grade or faster for devices with HP banks.
- Virtex®-7 devices, -2 speed grade or faster for devices with HR Banks or -1 speed grade or faster for devices with HP banks.

- Artix®-7 devices, -2 speed grade or faster.
- Zynq-7000 Devices, -2 speed grade or faster for XC7Z010/20 devices and -1 speed grade or faster for XC7Z030/45/100 devices.
- For UltraScale architecture devices, any I/O supporting a maximum of 1.25 Gb/s should support SGMII over device LVDS. See the performance characteristics of I/O banks in the UltraScale architecture device in the device data sheet.

Recommended for Chip-to-Chip Copper Implementations Only

This interface supports an SGMII link between the FPGA and an external PHY device across a single PCB; keep the SGMII copper signal lengths to a minimum.

SGMII Using Asynchronous Oversampling over 7 Series FPGAs LVDS

See *LVDS 4x Asynchronous Oversampling Using 7 Series FPGAs* (XAPP523) [Ref 4] for information about 7 series devices using asynchronous oversampling.

SGMII Only

The interface implemented using this asynchronous oversampling method supports SGMII between the FPGA and an external PHY device; the interface cannot directly support 1000BASE-X.

Receiver UI Specification

IMPORTANT: *The DRU must have at least two valid sampling points per data bit, requiring 0.5 UI of opening. The settings of the FPGA add 0.125 UI of requirement making a total opening requirement at the receiver of 0.625 UI.*

Recommended for Chip-to-Chip Copper Implementations Only

This interface supports an SGMII link between the FPGA and an external PHY device across a single PCB; keep the SGMII copper signal lengths to a minimum.

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

When used with the Ten-Bit Interface (TBI), the core provides the functionality to implement the 1000BASE-X PCS sublayer (or to provide SGMII support) with use of an external SerDes.

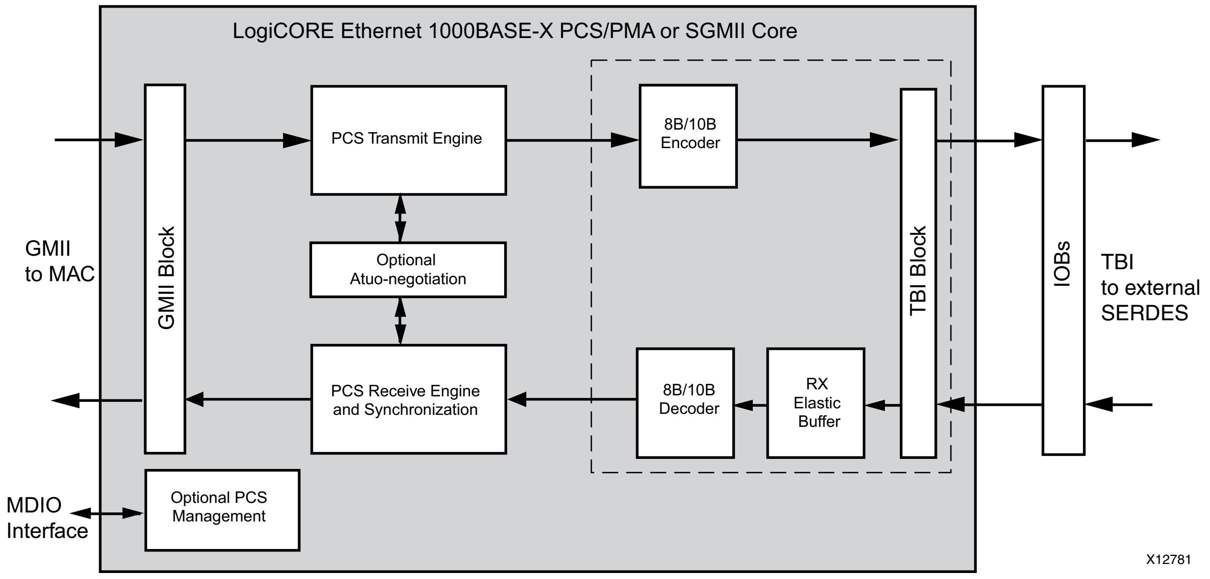


Figure 1-6: **Core Block Diagram with TBI**

The optional TBI is used in place of the device-specific transceiver to provide a parallel interface for connection to an external PMA SerDes device, allowing an alternative implementation for families without device-specific transceivers. In this implementation, additional logic blocks are required in the core to replace some of the device-specific transceiver functionality. These blocks are surrounded by a dashed line (see Figure 1-6). Other blocks are identical to those previously defined.

UltraScale architecture, Zynq-7000, Artix-7 and Virtex-7 devices do not support the TBI. Kintex-7 devices support TBI at 3.3 V or lower.

8B/10B Encoder

8B/10B encoding, as defined in IEEE 802.3-2008 specification, 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

8B/10B decoding, as defined in IEEE 802.3-2008 specification, 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.

Receive Elastic Buffer

The receive 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 core internal 125 MHz clock domain.

The receive elastic buffer is an asynchronous First In First Out (FIFO) implemented in internal RAM. The operation of the receive 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, which should be routed to device IOBs to provide an off-chip TBI.

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 Field Programmable Gate Array (FPGA) designs using Xilinx implementation software with the Xilinx Design Constraints (XDC) is recommended.

Contact your local Xilinx representative for a closer review and estimation for your specific requirements.

Licensing and Ordering Information

This Xilinx® LogiCORE IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Figure 2-1 shows the 1 Gigabit Ethernet PCS and PMA sublayers provided by this core, which are part of the Ethernet architecture. The part of this architecture, from the Ethernet MAC to the right, is defined in the IEEE 802.3-2008 specification. Figure 2-1 also shows where the supported interfaces fit into the architecture.

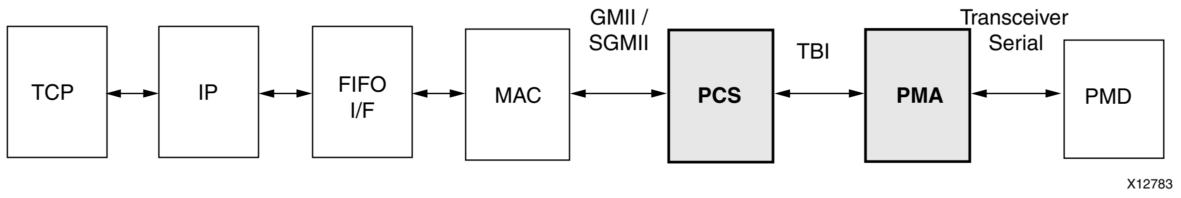


Figure 2-1: **Overview of Ethernet Architecture**

MAC

The Ethernet Media Access Controller (MAC) is defined in IEEE 802.3-2008, clauses 2, 3, and 4. A MAC is responsible for the Ethernet framing protocols and error detection of these frames. The MAC is independent of, and can connect to, any type of physical layer device.

GMII/SGMII

The Gigabit Media Independent Interface (GMII), a parallel interface connecting a MAC to the physical sublayers (PCS, PMA, and PMD), is defined in IEEE 802.3-2008, clause 35. For a MAC operating at a speed of 1 Gb/s, the full GMII is used; for a MAC operating at a speed of 10 Mb/s or 100 Mb/s, the GMII is replaced with a Media Independent Interface (MII) that uses a subset of the GMII signals.

The Serial-GMII (SGMII) is an alternative interface to the GMII/MII that converts the parallel interface of the GMII/MII into a serial format capable of carrying traffic at speeds of 10 Mb/s, 100 Mb/s, and 1 Gb/s. This radically reduces the I/O count and for this reason is often preferred by Printed Circuit Board (PCB) designers. The SGMII specification is closely related to the 1000BASE-X PCS and PMA sublayers, which enables it to be offered in this core.

Physical Coding Sublayer

The Physical Coding Sublayer (PCS) for 1000BASE-X operation is defined in IEEE 802.3-2008, clauses 36 and 37, and performs these operations:

- Encoding (and decoding) of GMII data octets to form a sequence of ordered sets
- 8B/10B encoding (and decoding) of the sequence ordered sets
- 1000BASE-X Auto-Negotiation for information exchange with the link partner

Ten Bit Interface

The Ten-Bit Interface (TBI), defined in IEEE 802.3-2008 clause 36 is a parallel interface connecting the PCS to the PMA and transfers the 8B/10B encoded sequence-ordered sets. The TBI should be used with an external SerDes device to implement the PMA functionality.

Physical Medium Attachment

The Physical Medium Attachment (PMA) for 1000BASE-X operation, defined in IEEE 802.3-2008 clause 36, performs the following:

- Serialization (and deserialization) of code-groups for transmission (and reception) on the underlying serial Physical Medium Dependent (PMD)
- Recovery of the clock from the 8B/10B-coded data supplied by the PMD

The device-specific transceivers provide the serial interface required to connect the PMD.

Physical Medium Dependent

The Physical Medium Dependent (PMD) sublayer is defined in IEEE 802.3-2008 clause 38 for 1000BASE-LX and 1000BASE-SX (long and short wavelength laser). This type of PMD is provided by the external GBIC or SFP optical transceivers. An alternative PMD for 1000BASE-CX (short-haul copper) is defined in IEEE 802.3-2008 clause 39.

Standards

- Ethernet Standard 802.3-2008 Clauses 22, 35, 36 and 38 [Ref 5]
- *Serial-GMII Specification V1.7* (CISCO SYSTEMS, ENG-46158) [Ref 1]

Performance

This section details the performance information for various core configurations.

Maximum Frequencies

The core operates at 125 MHz.

Core Latency

The stand-alone core does not meet all the latency requirements specified in IEEE 802.3-2008 because of the latency of the elastic buffers in both TBI and device-specific transceiver versions. However, the core can be used for backplane and other applications where strict adherence to the IEEE latency specification is not required.

Where strict adherence to the IEEE 802.3-2008 specification is required, the core can 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 802.3-2008 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 TX 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 receive elastic buffer.

The receive 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/PMA Using a Transceiver

These measurements are for the core only; they do not include the latency through the device-specific transceiver or the TX 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 serial transceiver 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 serial transceiver 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 six clock periods of `userclk2`.

Latency for SGMII

When implementing the SGMII standard, the core latency figures are identical to the latency for the core using the serial transceiver. These figures do not include the latency through the serial transceiver or any elastic buffers added in the example design.

Throughput

The core operates at a full lane rate of 1.25 Gb/s.

Voltage Requirements

Virtex®-7 devices support GMII at 3.3 V or lower only in certain parts and packages; see the *7 Series FPGAs SelectIO Resources User Guide* (UG471) [Ref 2]. Kintex®-7 devices support TBI and GMII at 3.3 V or lower. Artix®-7 and Zynq®-7000 devices support GMII at 3.3 V or lower.

Speed Grades

UltraScale™ architecture, Zynq-7000, Virtex-7, Kintex-7, and Artix-7 devices support speed grade -1 and faster for GT transceiver interface.

For the SGMII LVDS interface, see Synchronous SGMII over LVDS.

Resource Utilization

Resources required for this core have been estimated for the UltraScale architecture, Zynq-7000, Virtex-7, Kintex-7, and Artix-7 devices. These values were generated using the Vivado® Design Suite.

Table 2-1: **Core Family Support**

Device Family	**LogiCORE IP Functionality**							
	1000BASE-X		**GMII to SGMII Bridge or SGMII to GMII Bridge**				**1000BASE-X and SGMII Standards with Dynamic Switching**	
	TBI	**Device Specific Transceiver**	**TBI**	**Device Specific Transceiver**	**Synchronous LVDS SelectIO**	**Asynchronous LVDS SelectIO**	**TBI**	**Device Specific Transceiver**
UltraScale	No	Yes	No	Yes	Yes	No	No	Yes
Zynq-7000	No	Yes	No	Yes	Yes	No	No	Yes
Virtex-7	No	Yes	No	Yes	Supported in -2 speed grade and faster parts for HR banks; -1 speed grade and faster for HP banks	Available through XAPP523 [Ref 4]	No	Yes
Kintex-7	Yes	Yes	Yes	Yes	Supported in -2 speed grade and faster parts for HR banks; -1 speed grade and faster for HP banks	Available through XAPP523 [Ref 4]	Yes	Yes
Artix-7	No	Yes	No	Yes	Supported in -2 speed grades and faster.	No	No	Yes

Zynq-7000, Virtex-7, Kintex-7, and Artix-7 families contain six input LUTs. Utilization figures are obtained by implementing the block-level wrapper for the core. This wrapper is part of the example design and connects the core to the selected physical interface.

BUFG Usage

- BUFG usage does not consider multiple instantiations of the core, where clock resources can often be shared.
- BUFG usage does not include the reference clock required for IDELAYCTRL. This clock source can be shared across the entire device and is not core specific.

UltraScale Architecture Devices

Table 2-2 shows the device utilization for the 1000BASE-X standard for UltraScale architecture devices.

Table 2-2: **Device Utilization for the 1000BASE-X Standard**

Parameter Values			Device Resources			
Physical Interface / **Transceiver**	**MDIO Interface**	**Auto- Negotiation**	**CLBs**	**LUTs**	**FFs**	**BUFG_GTs**
Yes	Yes	Yes	188	695	1151	2
Yes	Yes	No	150	503	950	2
Yes	No	Yes	168	584	1059	2
Yes	No	No	139	445	882	2

Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `independent_clock`, `userclk`, `userclk2`, `rxuserclk`, and `rxuserclk2`. These BUFGs can be shared across multiple instances of the core.

Table 2-3 shows the device utilization for the GMII to SGMII or SGMII to GMII bridge for UltraScale architecture devices.

Table 2-3: **Device Utilization for the GMII to SGMII or SGMII to GMII Bridge**

Parameter Values			Device Resources			
Physical Interface / **Transceiver**	**MDIO Interface**	**Auto- Negotiation**	**CLBs**	**LUTs**	**FFs**	**BUFG_GTs**
Yes	Yes	Yes	198	667	1330	2
Yes	Yes	No	177	560	1162	2
Yes	No	Yes	190	597	1249	2
Yes	No	No	151	499	1095	2

Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `independent_clock`, `userclk`, `userclk2`, `rxuserclk`, and `rxuserclk2`. These BUFGs can be shared across multiple instances of the core.

Table 2-4 shows the device utilization for the 1000BASE-X and SGMII standards with dynamic switching for UltraScale architecture devices.

Table 2-4: **Device Utilization for 1000BASE-X and SGMII with Dynamic Switching**

Parameter Values			Device Resources			
Physical Interface / **Transceiver**	**MDIO Interface**	**Auto- Negotiation**	**CLBs**	**LUTs**	**FFs**	**BUFG_GTs**
Yes	Yes	Yes	210	719	1364	2
Yes	Yes	No	181	553	1163	2
Yes	No	Yes	198	632	1272	2
Yes	No	No	151	499	1095	2

Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `independent_clock`, `userclk`, `userclk2`, `rxuserclk`, and `rxuserclk2`. These BUFGs can be shared across multiple instances of the core.

Table 2-5 shows the device utilization for the SGMII standard over LVDS for UltraScale architecture devices.

Table 2-5: **Device Utilization for SGMII Standard over LVDS Interface**

Parameter Values			Device Resources			
Physical Interface / **LVDS**	**MDIO Interface**	**Auto- Negotiation**	**CLBs**	**LUTs**	**FFs**	**BUFGs**
Yes	Yes	Yes	161	650	743	0
Yes	Yes	No	139	550	593	0
Yes	No	Yes	155	584	662	0
Yes	No	No	130	502	526	0

Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `clk125`, `clk312` and `clk625`.
3. BUFGs specified can be shared across multiple instances of the core depending on the implementation.

Zynq-7000 Devices

Table 2-6 shows the device utilization for the 1000BASE-X standard for Zynq-7000 devices.

Table 2-6: **Device Utilization for the 1000BASE-X Standard**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	376	604	896	0
Yes	No	Yes	No	299	400	682	0
Yes	No	No	Yes	330	496	824	0
Yes	No	No	No	279	376	660	0

Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `txoutclk`, `independent_clock`, `userclk`, and `userclk2`. These BUFGs can be shared across multiple instances of the core.

Table 2-7 shows the device utilization for the GMII to SGMII or SGMII to GMII bridge for Zynq-7000 devices.

Table 2-7: **Device Utilization for the GMII to SGMII or SGMII to GMII Bridge**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	424	573	1027	0
Yes	No	Yes	No	300	387	679	0
Yes	No	No	Yes	342	471	778	0
Yes	No	No	No	312	447	738	0

Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `txoutclk`, `independent_clock`, `userclk`, and `userclk2`. These BUFGs can be shared across multiple instances of the core.
3. Additional BUFGs can be added for `rxoutclk`. Alternately a BUFMR and BUFR in series can be used. A BUFG is added by default if you select **Include Shared Logic in Core**, otherwise you can manually instantiate the BUFGs. This is mandatory when the fabric elastic buffer is used.

Table 2-8 shows the device utilization for the 1000BASE-X and SGMII standards with dynamic switching for Zynq-7000 devices.

Table 2-8: **Device Utilization for 1000BASE-X and SGMII with Dynamic Switching**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	444	698	1027	0
Yes	No	Yes	No	339	493	821	0
Yes	No	No	Yes	386	595	924	0
Yes	No	No	No	312	447	738	o

Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `txoutclk`, `independent_clock`, `userclk`, and `userclk2`.These BUFGs can be shared across multiple instances of the core.
3. Additional BUFGs can be added for `rxoutclk`. Alternately a BUFMR and BUFR in series can be used. A BUFG is added by default if you select **Include Shared Logic in Core**; otherwise you can manually instantiate the BUFGs. This is mandatory when the fabric elastic buffer is used.

Table 2-9 shows the device utilization for the SGMII standard over LVDS for Zynq-7000 devices.

Table 2-9: **Device Utilization for SGMII Standard over LVDS Interface**

Parameter Values			Device Resources			
Physical Interface	MDIO Interface	Auto- Negotiation	Slices	LUTs	FFs	BUFGs
LVDS						
Yes	Yes	Yes	395	919	939	0
Yes	Yes	No	334	781	772	0
Yes	No	Yes	354	829	836	0
Yes	No	No	285	732	707	0

Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `clk125`, `clk208`, `clk104`, and `clk625`.
3. BUFGs specified can be shared across multiple instances of the core depending on the implementation.

Virtex-7 Devices

Table 2-10 shows the device utilization for the 1000BASE-X standard for Virtex-7 devices.

Table 2-10: **Device Utilization for the 1000BASE-X Standard**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	370	604	896	0
Yes	No	Yes	No	288	400	682	0
Yes	No	No	Yes	349	491	794	0
Yes	No	No	No	267	358	630	0

Notes:
1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `independent_clock`, `userclk` and `userclk2`.
3. These BUFGs can be shared across multiple instances of the core.

Table 2-11 shows the device utilization for the GMII to SGMII or SGMII to GMII bridge for Virtex-7 devices.

Table 2-11: **Device Utilization for the GMII to SGMII or SGMII to GMII Bridge**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	340	566	851	0
Yes	No	Yes	No	306	487	790	0
Yes	No	No	Yes	354	564	894	0
Yes	No	No	No	286	446	738	0

Notes:
1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `independent_clock`, `userclk`, and `userclk2`. These BUFGs can be shared across multiple instances of the core.
3. Additional BUFGs can be added for `rxoutclk`. Alternately a BUFMR and BUFR in series can be used. A BUFG is added by default if you select **Include Shared Logic in Core**; otherwise you can manually instantiate the BUFGs. This is mandatory when the fabric elastic buffer is used.

Table 2-12 shows the device utilization for the 1000BASE-X and SGMII standards with dynamic switching for Virtex-7 devices.

Table 2-12: **Device Utilization for 1000BASE-X and SGMII Standards with Dynamic Switching**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	286	446	738	0
Yes	No	Yes	No	254	386	650	0
Yes	No	No	Yes	378	594	924	0
Yes	No	No	No	296	457	768	0

Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `independent_clock`, `userclk`, and `userclk2`. These BUFGs can be shared across multiple instances of the core.
3. Additional BUFGs can be added for `rxoutclk`. Alternately a BUFMR and BUFR in series can be used. A BUFG is added by default if you select **Include Shared Logic in Core**; otherwise you can manually instantiate the BUFGs. This is mandatory when the fabric elastic buffer is used.

Table 2-13 shows the device utilization for the SGMII standard over LVDS for Virtex-7 devices.

Table 2-13: **Device Utilization for SGMII Standard over LVDS Interface**

Parameter Values			Device Resources			
Physical Interface	MDIO Interface	Auto- Negotiation	Slices	LUTs	FFs	BUFGs
LVDS						
Yes	Yes	Yes	434	930	952	0
Yes	Yes	No	333	782	772	0
Yes	No	Yes	373	837	849	0
Yes	No	No	304	738	720	0

Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `clk125`, `clk208`, `clk104`, and `clk625`.
3. BUFGs specified can be shared across multiple instances of the core depending on the implementation.

Kintex-7 Devices

Table 2-14 shows the device utilization for the 1000BASE-X standard using device-specific transceivers or the TBI for Kintex-7 devices.

Table 2-14: **Device Utilization for the 1000BASE-X Standard**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	371	604	896	0
Yes	No	Yes	No	296	404	712	0
Yes	No	No	Yes	320	495	824	0
Yes	No	No	No	252	362	630	0
No	Yes	Yes	Yes	309	493	706	0
No	Yes	Yes	No	180	310	481	0
No	Yes	No	Yes	227	379	588	0
No	Yes	No	No	165	256	429	0

Notes:
1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `txoutclk`, `independent_clock`, `userclk`, and `userclk2`. These BUFGs can be shared across multiple instances of the core.
3. For TBI mode two BUFGs are required to drive `refclk` and `gtx_clk`. Notes [(1)]-[(2)] are not applicable in that case.

Table 2-15 shows the device utilization for the GMII to SGMII or SGMII to GMII bridge using device-specific transceivers or TBI for Kintex-7 devices.

Table 2-15: **Device Utilization for the GMII to SGMII or SGMII to GMII Bridge**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	358	566	851	0
Yes	No	Yes	No	350	490	820	0
Yes	No	No	Yes	347	565	894	0
Yes	No	No	No	313	457	768	0
No	Yes	Yes	Yes	256	484	684	0
No	Yes	Yes	No	184	313	471	0
No	Yes	No	Yes	215	385	565	0

Table 2-15: **Device Utilization for the GMII to SGMII or SGMII to GMII Bridge *(Cont'd)***

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
No	Yes	No	No	165	274	419	0

Notes:
1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `txoutclk`, `independent_clock`, `userclk`, and `userclk2`. These BUFGs can be shared across multiple instances of the core.
3. Additional BUFGs can be added for `rxoutclk`. Alternately a BUFMR and BUFR in series can be used. BUFG is added by default if you select **Include Shared Logic in Core**; otherwise you can manually instantiate the BUFGs. This is mandatory when the fabric elastic buffer is used.
4. For TBI mode two BUFGs are required to drive `refclk` and `gtx_clk`. Notes (1)-(3) are not applicable in this case.

Table 2-16 shows the device utilization for the 1000BASE-X and SGMII standards with dynamic switching for Kintex-7 devices.

Table 2-16: **Device Utilization for 1000BASE-X and SGMII with Dynamic Switching**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	406	612	911	0
Yes	No	Yes	No	297	389	680	0
Yes	No	No	Yes	390	595	924	0
Yes	No	No	No	320	448	738	0
No	Yes	Yes	Yes	273	534	714	0
No	Yes	Yes	No	178	317	472	0
No	Yes	No	Yes	215	418	595	0
No	Yes	No	No	164	276	419	0

Notes:
1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `txoutclk`, `independent_clock`, `userclk`, and `userclk2`. These BUFGs can be shared across multiple instances of the core.
3. Additional BUFGs can be added for `rxoutclk`. Alternately a BUFMR and BUFR in series can be used. BUFG is added by default if you select **Include Shared Logic in Core**; otherwise you can manually instantiate the BUFGs. This is mandatory when the fabric elastic buffer is used.
4. For TBI mode two BUFGs are required to drive `refclk` and `gtx_clk`. Notes 1-3 are not applicable in this case.

Table 2-17 shows the device utilization for the SGMII standard over LVDS for Kintex-7 devices.

Table 2-17: **Device Utilization for SGMII Standard over LVDS Interface**

Parameter Values			Device Resources			
Physical Interface	MDIO Interface	Auto- Negotiation	Slices	LUTs	FFs	BUFGs
LVDS						
Yes	Yes	Yes	400	918	939	0
Yes	Yes	No	332	774	759	0
Yes	No	Yes	344	828	836	0
Yes	No	No	311	739	720	0

Notes:
1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `clk125`, `clk208`, `clk104`, and `clk625`.
3. BUFGs specified can be shared across multiple instances of the core depending on the implementation.

Artix-7 Devices

Table 2-18 shows the device utilization for the 1000BASE-X standard for Artix-7 devices.

Table 2-18: **Device Utilization for the 1000BASE-X Standard**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	398	632	926	0
Yes	No	Yes	No	314	430	744	0
Yes	No	No	Yes	357	518	824	0
Yes	No	No	No	283	396	692	0

Notes:
1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `txoutclk`, `independent_clock`, `userclk`, and `userclk2`.
3. These BUFGs can be shared across multiple instances of the core.

Table 2-19 shows the device utilization for the GMII to SGMII or SGMII to GMII bridge for Artix-7 devices.

Table 2-19: **Device Utilization for the GMII to SGMII or SGMII to GMII Bridge**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	377	596	913	0
Yes	No	Yes	No	279	413	711	0
Yes	No	No	Yes	337	497	810	0
Yes	No	No	No	322	471	768	0

Notes:
1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `txoutclk`, `independent_clock`, `userclk`, and `userclk2`.
3. BUFGs specified in 2 can be shared across multiple instances of the core.
4. Additional BUFGs can be added for `rxoutclk`. Alternately a BUFMR and BUFR in series can be used. BUFG is added by default if you select **Include Shared Logic in Core**; otherwise you can manually instantiate the BUFGs. This is mandatory when the fabric elastic buffer is used

Table 2-20 shows the device utilization for the 1000BASE-X and SGMII standards with dynamic switching for Artix-7 devices.

Table 2-20: **Device Utilization for 1000BASE-X and SGMII with Dynamic Switching**

Parameter Values				Device Resources			
Physical Interface		MDIO Interface	Auto-Negotiation	Slices	LUTs	FFs	BUFGs
Transceiver	TBI						
Yes	No	Yes	Yes	402	628	911	0
Yes	No	Yes	No	320	411	680	0
Yes	No	No	Yes	337	497	810	0
Yes	No	No	No	341	478	800	0

Notes:
1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `txoutclk`, `independent_clock`, `userclk`, and `userclk2`.These BUFGs can be shared across multiple instances of the core.
3. Additional BUFGs can be added for `rxoutclk`. Alternately a BUFMR and BUFR in series can be used. BUFG is added by default if you select **Include Shared Logic in core**; otherwise you can manually instantiate the BUFGs. This is mandatory when the fabric elastic buffer is used.

Table 2-21 shows the device utilization for the SGMII standard over LVDS for Artix-7 devices.

Table 2-21: **Device Utilization for SGMII Standard over LVDS Interface**

Parameter Values			Device Resources			
Physical Interface	**MDIO Interface**	**Auto- Negotiation**	**Slices**	**LUTs**	**FFs**	**BUFGs**
LVDS						
Yes	Yes	Yes	387	919	939	0
Yes	Yes	No	346	781	772	0
Yes	No	Yes	376	830	836	0
Yes	No	No	301	731	707	0

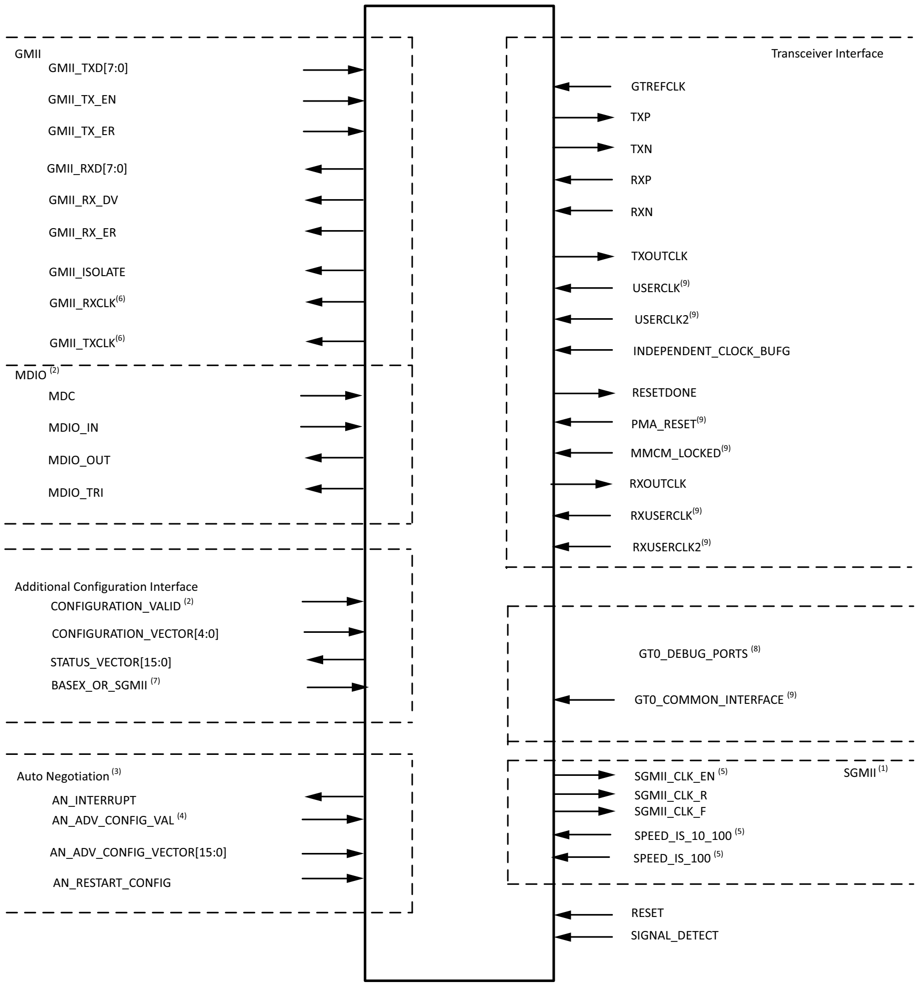
Notes:

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFGs are required to drive `clk125`, `clk208`, `clk104`, and `clk625`.
3. BUFGs specified can be shared across multiple instances of the core depending on the implementation.

The clocking logic is only required once for multiple SGMII cores.

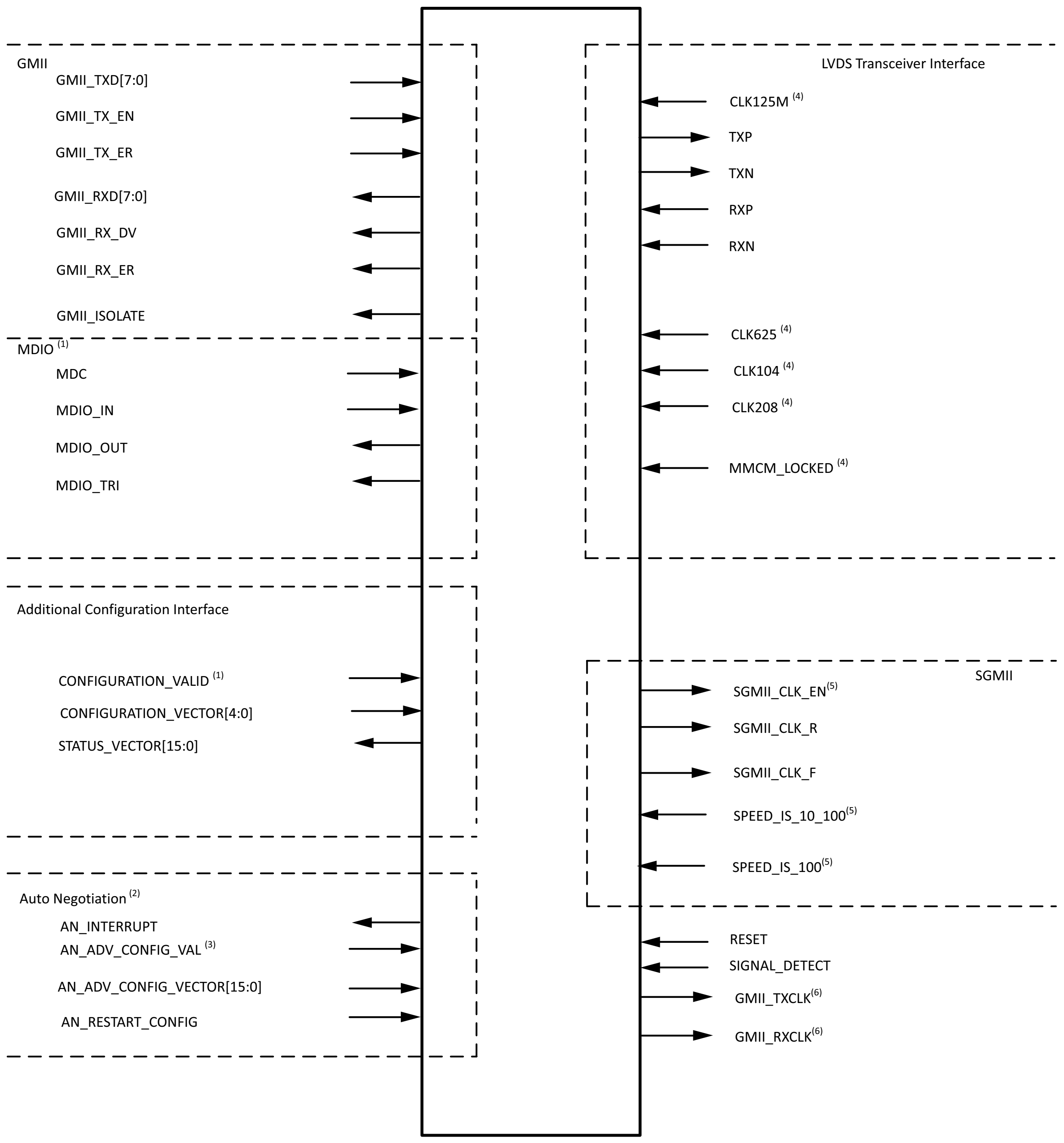
Port Descriptions

The pinouts for the core in using the optional transceiver, using SGMII over LVDS and using the TBI are shown in Figure 2-2, Figure 2-3 and Figure 2-4. The ports are described in this section.



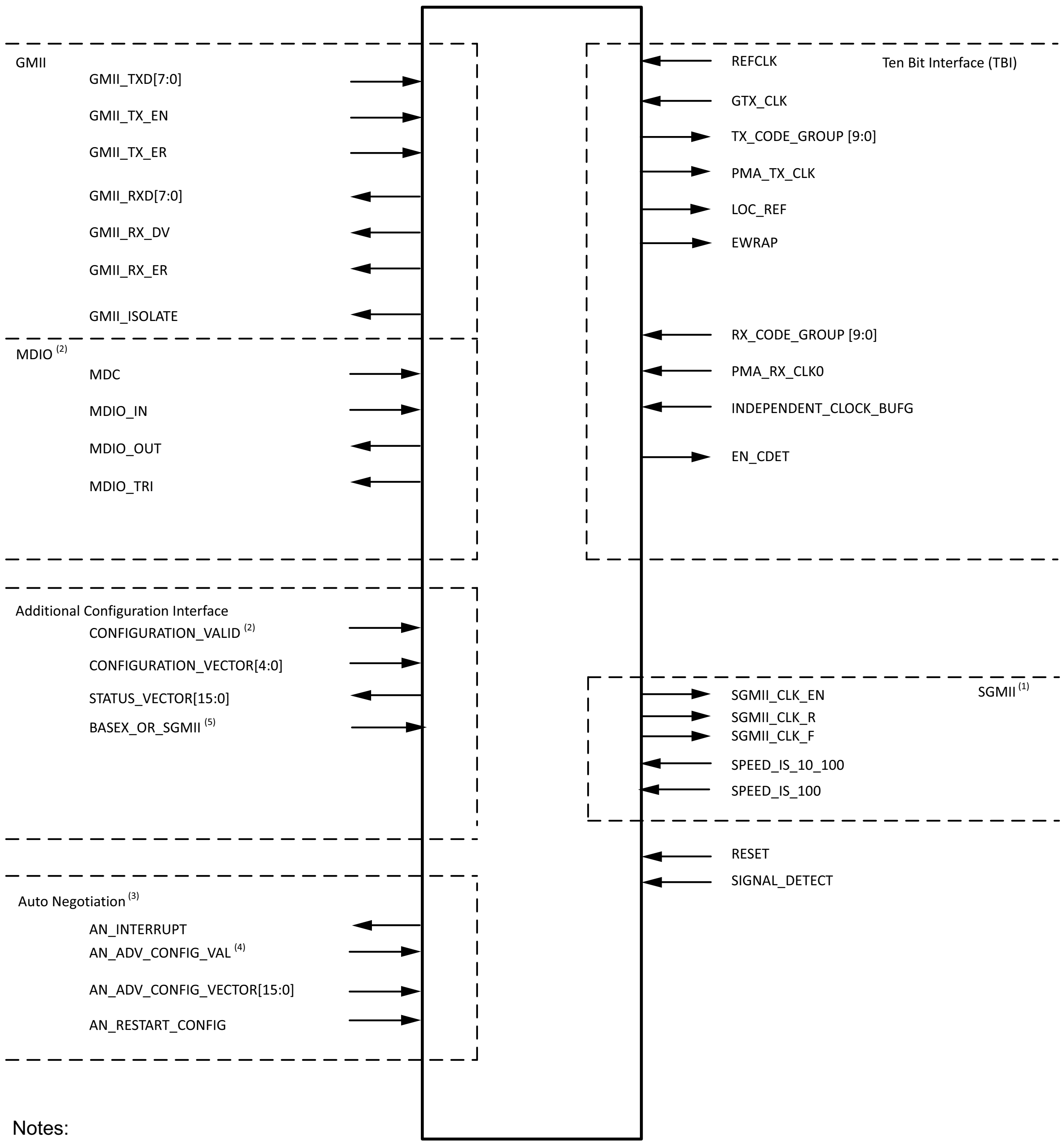
Notes:
1) Pins are visible only if SGMII is enabled.
2) Pins are visible only if MDIO is enabled.
3) Pins are visible only if Auto Negotiation is enabled.
4) Pins are visible only if Auto Negotiation and MDIO are enabled.
5) Pins are visible only if generated to interface with TEMAC .
6) Pins are visible only if generated to interface with GEM .These pins are a part of GMII interface
7) Pin is visible if a) auto -negotiation and dynamic switching are enabled OR b) dynamic switching and MDIO are enabled .
8) Pins are visible only if Transceiver Debug is enabled.
9) Direction of the pins change if shared logic is a part of core. Direction given above are in case when shared logic is a part of example design .

Figure 2-2: **Pinout in 1000BASE-X or SGMII or Dynamic Switching Mode Using Transceivers**



Notes:
1) Pins are visible only if MDIO is enabled.
2) Pins are visible only if Auto-Negotiation is enabled.
3) Pins are visible only if Auto Negotiation and MDIO are enabled.
4) Direction of the pins change when shared logic is a part of core. Diagram shows the case when shared logic is a part of Example Design.
5) Pins are visible only in case TEMAC is selected as interface.
6) Pins are visible only in case GEM is selected as interface. These pins are a part of GMII interface.

Figure 2-3: **Pinout for SGMII over LVDS Mode**



Notes:

1) Pins are visible only if SGMII is enabled.
2) Pins are visible only if MDIO is enabled.
3) Pins are visible only if Auto Negotiation is enabled.
4) Pins are visible only if Auto Negotiation and MDIO are enabled.
5) Pin is visible if a) auto -negotiation and dynamic switching are enabled OR b) dynamic switching and MDIO are enabled .

Figure 2-4: **Pinout Using TBI Mode**

GMII Ports

Table 2-22 describes the core GMII interface ports common to all core configurations. These are typically attached to an Ethernet MAC, either off-chip or internally integrated. The HDL block level design delivered with the core connects these signals to IOBs.

For more information, see Using the Client-Side GMII Datapath.

Table 2-22: **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 3-state control for GMII Isolation. Only of use when implementing an External GMII as shown by the block level design HDL.

Notes:

1. When the TX elastic buffer is present, these signals are synchronous to `gmii_tx_clk`. When the TX elastic buffer is omitted, see [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.

MDIO Management Interface Ports

Table 2-23 describes the optional MDIO interface signals of the core that are 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 internal MAC core. For more information, see Management Registers.

Table 2-23: **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	3-state control for MDIO signals; 0 signals that the value on mdio_out should be asserted onto the MDIO interface.

Table 2-23: **Optional MDIO Interface Signal Pinout *(Cont'd)***

Signal	Direction	Clock Domain	Description
phyad[4:0]	Input	N/A	Physical Address of the PCS management register set. The PHY Address value entered in the Vivado IDE percolates to this signal.

1. These signals can be connected to a 3-state buffer to create a bidirectional mdio signal suitable for connection to an external MDIO controller (for example, an Ethernet MAC)

Reset Ports

Table 2-24: **Reset Signals Pinout**

Signal	Direction	Clock Domain	Description
reset	Input	n/a	Asynchronous reset for the entire core. Active-High
reset_done	Input	userclk	Marks the completion of the gtwizard reset sequence. In cases where the transceiver is not present, this signal is tied to 1.

Dynamic Switching Signal Port

Table 2-25 describes the signals present when the optional dynamic switching mode (between 1000BASE-X and SGMII standards) is selected. In this case, the MDIO (Table 2-23) and device-specific Transceiver Ports are always present.

Table 2-25: **Optional Dynamic Switching Signals**

Signal	Direction	Description
basex_or_sgmii[(1)]	Input	Used as the reset default to select the standard. Tie this signal to 0 or 1. 0: core comes out of reset operating in 1000BASE-X. 1: core comes out of reset operating in SGMII. ***Note:*** The standard can be set following reset through the MDIO management.

1. Clock domain is `userclk2`.

DRP and 1588 Support Ports

Table 2-26 and Table 2-27 describe the signals for supporting 1588. These interfaces are available only when this core is used in conjunction with the Tri-Mode Ethernet MAC core (TEMAC).

See the *LogiCORE IP 7 Series FPGAs Transceivers Wizard* (PG168) [Ref 6] for more details on the DRP signals.

Table 2-26: **DRP Interface to Transceiver Ports**

Signal	Direction	Description
drp_dclk	In	DRP interface clock, tied to userclk
drp_req(1)	Out	DRP request
drp_gnt(1)	In	DRP grant
drp_den(1)	Out	DRP enable signal
drp_dwe(1)	Out	DRP write enable
drp_drdy(1)	In	Indicates DRP operation is complete
drp_daddr[8:0](1)	Out	DRP address
drp_di[15:0](1)	Out	DRP data from transceiver
drp_do[15:0](1)	In	DRP data to transceiver

1. Signals are synchronous to `userclk`.

Table 2-27: **1588 Ports**

Signal	Direction	Description
systemtimer_s_field[47:0]	In	1588 System timer seconds value
systemtimer_ns_field[31:0]	In	1588 System timer nanoseconds value
rxphy_s_field[47:0]	Out	1588 timer PHY correction seconds value
rxphy_ns_field[31:0]	Out	1588 timer PHY correction nanoseconds value
rxrecclk	In	RX recovered clock from transceiver

Configuration and Status Vector Ports

Table 2-28 describe the ports that are used to configure and monitor the core if the MDIO interface is not used.

Table 2-28: **Alternative to Optional Management Interface - Vector Signal Pinout**

Signal(1)	Direction	Description
status_vector[15:0]	Output	See Table 2-75 for bit description
configuration_vector[4:0]	Input	Additional interface to program management Register 0 irrespective of the optional MDIO interface. See Table 2-73 for bit description.

Table 2-28: **Alternative to Optional Management Interface - Vector Signal Pinout *(Cont'd)***

Signal[1]	Direction	Description
configuration_valid	Input	This signal is valid only when the MDIO interface is present. The rising edge of this signal is the enable signal to overwrite the Register 0 contents that were written from the MDIO interface. For triggering a fresh update of Register 0 through *configuration_vector*, this signal should be deasserted and then reasserted.
an_adv_config_vector[15:0]	Input	Auto-Negotiation: this interface is used to program Register 4, irrespective of MDIO interface. For more information, see Auto-Negotiation. See Table 2-74 for bit description.
an_adv_config_val	Input	This signal is valid only when the MDIO interface is present. The rising edge of this signal is the enable signal to overwrite the Register 4 contents that were written from the MDIO interface. For triggering a fresh update of Register 4 through *an_adv_config_vector*, this signal should be deasserted and then reasserted.
an_restart_config	Input	This signal is valid only when AN is present. The rising edge of this signal is the enable signal to overwrite Bit 9 or Register 0. For triggering a fresh AN Start, this signal should be deasserted and then reasserted.
an_interrupt	Output	When the MDIO module is selected through the Vivado IDE interface, this signal indicates an active-High interrupt for Auto-Negotiation cycle completion which needs to be cleared though MDIO. This interrupt can be enabled/disabled and cleared by writing to the appropriate PCS management register. When the MDIO module is not selected, this signal indicates AN Complete, which is asserted as long as the Auto-Negotiation is complete and AN is not restarted and cannot be cleared.

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

TBI Ports

Table 2-29 describes the optional TBI signals, used as an alternative to the transceiver interfaces. The appropriate HDL block level design delivered with the core connects these signals to IOBs to provide an external TBI suitable for connection to an off-device 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 The Ten-Bit Interface.

Table 2-29: **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 802.3-2008 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.

Table 2-29: **Optional TBI Interface Signal Pinout *(Cont'd)***

Signal	Direction	Clock Domain	Description
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° 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.

Transceiver Ports

Table 2-30 shows the transceiver interface ports for the case when Shared Logic is included in the example design.

Table 2-30: **Transceiver Interface with Shared Logic in the Example Design**

Signal	Direction	Description
gtrefclk	Input	125 MHz reference clock from IBUFDS to the transceiver
txp	Output	Transmit differential
txn	Output	Transmit differential
rxp	Input	Receive differential
rxn	Input	Receive differential
txoutclk	Output	txoutclk from transceiver
userclk	Input	Also connected to txusrclk of the device-specific transceiver. Clock domain is not applicable.
userclk2	Input	Also connected to txusrclk2 of the device-specific transceiver. Clock domain is not applicable.
rxoutclk	Output	rxoutclk from transceiver
rxuserclk	Input	Also connected to rxusrclk of the device-specific transceiver. Clock domain is not applicable.
rxuserclk2	Input	Also connected to rxusrclk2 of the device-specific transceiver. Clock domain is not applicable.
independent_clock_bufg(1)	Input	Stable clock in transceiver and also as control clock for IDELAYCTRL.
resetdone	Output	Indication that reset sequence of the transceiver is complete

Table 2-30: **Transceiver Interface with Shared Logic in the Example Design *(Cont'd)***

Signal	Direction	Description
pma_reset	Input	Hard reset synchronized to independent_clock_bufg.
mmcm_locked	Input	Indication from the MMCM that the outputs are stable.
gmii_txclk	Output	Applicable only when GEM is selected as the interface type. This is the looped back version of userclk2 in BASE-X mode and is the same as sgmii_clk_r in SGMII modes.
gmii_rxclk	Output	Same as gmii_txclk.
GT COMMON Clock Interface		
gt0_pll0outclk_in	Input	Valid only for Artix-7 families. Indicates out clock from PLL0 of GT Common
gt0_pll0outrefclk_in	Input	Valid only for Artix-7 families. Indicates reference out clock from PLL0 of GT Common
gt0_pll1outclk_in	Input	Valid only for Artix-7 families. Indicates out clock from PLL1 of GT Common
gt0_pll1outrefclk_in	Input	Valid only for Artix-7 families. Indicates reference out clock from PLL1 of GT Common
gt0_pll0lock_in	Input	Valid only for Artix-7 families. Indicates out PLL0 of GT Common has locked
gt0_pll0refclklost_in	Input	Valid only for Artix-7 families. Indicates out reference clock for PLL0 of GT Common is lost
gt0_pll0reset_out	Output	Valid only for Artix-7 families. Reset for PLL of GT Common from reset fsm in GT Wizard
gt0_qplloutclk_in	Input	Valid only for non Artix-7 families. Indicates out clock from PLL of GT Common
gt0_qplloutrefclk_in	Input	Valid only for non Artix-7 families. Indicates reference out clock from PLL of GT Common

1. The example design assumes `independent_clock_bufg` to be 200 MHz. If it is different, the period should be changed in the `<component_name>_gtwizard.v[hd]` file with the parameter STABLE_CLOCK_PERIOD = 5. This is the period of the stable clock driving this state-machine; units are ns. Also, make sure that if the design is using IDELAYCTRL then the value given to this clock is
   a) either within the range of the `refclk` value specified for IDELAYCTRL. OR
   b) Some different clock is used as the reference clock for IDELAYCRTL.

   Changing the value for STABLE_CLOCK_PERIOD to anything other than 5 may require changing the WAIT_TIMEOUT_2ms counter value in the `<component_name>_tx_startup_fsm.v[hd]` and `<component_name>_rx_startup_fsm.v[hd]` files to provide appropriate delays to the FSMs for completion of its startup sequence.

   The valid range for stable clock period is 4 to 250 ns. The core has only been tested with 5 ns (that is, 200 MHz). For UltraScale architecture devices this must be 300 MHz because it is internally divided by 6 and a generated 50 MHz clock is given to the gtwizard in the case where the transceiver control and status ports are disabled and shared logic is in the core.

Table 2-31 describes the interface to the transceiver when Shared Logic is included in the core.

Table 2-31: **Transceiver Interface with Shared Logic in the Core**

Signal	Direction	Description
gtrefclk_p	Input	125 MHz differential reference clock to IBUFDS
gtrefclk_p	Input	125 MHz differential reference clock to IBUFDS
gtrefclk_out	Output	125 MHz reference clock from IBUFDS
txp	Output	Transmit differential
txn	Output	Transmit differential
rxp	Input	Receive differential
rxn	Input	Receive differential
userclk_out	Output	Also connected to txusrclk of the device-specific transceiver. Clock domain is not applicable.
userclk2_out	Output	Also connected to txusrclk2 of the device-specific transceiver. Clock domain is not applicable.
rxuserclk_out	Output	Also connected to rxusrclk of the device-specific transceiver. Clock domain is not applicable.
rxuserclk2_out	Output	Also connected to `rxusrclk2` of the device-specific transceiver. Clock domain is not applicable.
independent_clock_bufg(1)	Input	Stable clock in transceiver and also as control clock for IDELAYCTRL.
resetdone	Output	Indication that reset sequence of the transceiver is complete.
pma_reset_out	Output	Hard reset synchronized to independent_clock_bufg.
mmcm_locked_out	Output	Indication from the MMCM that the outputs are stable.
gmii_txclk	Output	Applicable only when GEM is selected as the interface type. This is the looped back version of userclk2 in BASE-X mode and the same as sgmii_clk_r in SGMII modes.
gmii_rxclk	Output	Same as gmii_txclk.
GT COMMON Clock Interface		
gt0_pll0outclk_out	Output	Valid only for Artix-7 families. Indicates out clock from PLL0 of GT Common.
gt0_pll0outrefclk_out	Output	Valid only for Artix-7 families. Indicates reference out clock from PLL0 of GT Common.
gt0_pll1outclk_out	Output	Valid only for Artix-7 families. Indicates out clock from PLL1 of GT Common.
gt0_pll1outrefclk_out	Output	Valid only for Artix-7 families. Indicates reference out clock from PLL1 of GT Common.
gt0_pll0lock_out	Output	Valid only for Artix-7 families. Indicates out PLL0 of GT Common has locked.
gt0_pll0refclklost_out	Output	Valid only for Artix7 families. Indicates out reference clock for PLL0 of GT Common is lost.
gt0_qplloutclk_out	Output	Valid only for non Artix-7 families. Indicates out clock from PLL of GT Common.

Table 2-31: **Transceiver Interface with Shared Logic in the Core *(Cont'd)***

Signal	Direction	Description
gt0_qplloutrefclk_out	Output	Valid only for non Artix-7 families. Indicates reference out clock from PLL of GT Common.

1. The example design assumes `independent_clock_bufg` to be 200 MHz. If it is different, the period should be changed in the `<component_name>_gtwizard.v[hd]` file with the parameter STABLE_CLOCK_PERIOD = 5. This is the period of the stable clock driving this state-machine; units are ns. Also, make sure that if the design is using IDELAYCTRL, then the value given to this clock is
a) either within the range of the `refclk` value specified for IDELAYCTRL. OR
b) Some different clock is used as reference clock for IDELAYCRTL.

   Changing the value for STABLE_CLOCK_PERIOD to anything other than 5 might require changing the WAIT_TIMEOUT_2ms counter value in the `<component_name>_tx_startup_fsm.v[hd]` and `<component_name>_rx_startup_fsm.v[hd]` files to provide appropriate delays to the FSMs for completion of its startup sequence.

   The valid range for stable clock period is 4 to 250 ns. The core has only been tested with 5 ns (that is, 200 MHz). For UltraScale architecture devices this must be 300 MHz because it is internally divided by 6 and a generated 50 MHz clock is given to the gtwizard in the case where the transceiver control and status ports are disabled and shared logic is in the core.

SGMII Ports

Table 2-32 describes the SGMII interface ports.

Table 2-32: **SGMII Interface Ports**

Signal	Direction	Description
sgmii_clk_en	Output	Clock for GMII transmit data
sgmii_clk_f	Output	Differential clock for GMII transmit data
sgmii_clk_r	Output	Differential clock for GMII transmit data
speed_is_10_100	Input	Speed control for controlling operating speed of SGMII interface
speed_is_100	Input	Speed control for controlling operating speed of SGMII interface

SGMII over LVDS Transceiver Interface Ports

Table 2-33 shows the physical side interface ports for SGMII over LVDS when Shared Logic is included in the Example Design.

Table 2-33: **Physical Side Interface Ports for SGMII over LVDS - Shared Logic in Example Design**

Signal	Direction	Description
clk125m	Input	125 MHz reference clock from IBUFDS.
txp	Output	Transmit differential
txn	Output	Transmit differential
rxp	Input	Receive differential
rxn	Input	Receive differential

Table 2-33: **Physical Side Interface Ports for SGMII over LVDS - Shared Logic in Example Design**

clk104	Input	104 MHz clock derived from 125MHz input differential clock
clk208	Input	208 MHz clock derived from 125MHz input differential clock
clk625	Input	625 MHz clock derived from 125MHz input differential clock
mmcm_locked	Input	Indication from the MMCM that the outputs are stable

Table 2-34 describes the physical side interface ports when the core is configured with SMGII over LVDS when Shared Logic is included in the core.

Table 2-34: **Physical Side Interface Ports for SGMII over LVDS with Shared Logic in the Core**

Signal	Direction	Description
refclk125_p	Input	Differential 125MHz clock synchronous to incoming SGMII serial data
refclk125_n	Input	Differential 125Mhz clock synchronous to incoming SGMII serial data
clk125_out	Output	Single ended 125 MHz clock.
clk625_out	Output	625 MHz clock
clk208_out	Output	208 MHz clock
clk104_out	Output	104 MHz clock
rst_125_out	Output	Output reset synchronous to 125 MHz clock.
mmcm_locked_out	Output	MMCM locked indication.
txp	Output	Transmit differential
txn	Output	Transmit differential
rxp	Input	Receive differential
rxn	Input	Receive differential

Note: The signal `eye_mon_wait_time` is given a lower value for ease in simulation. Actual implementation can tie it to 12'hFFF.

Transceiver Control and Status Debug Ports

Table 2-35 and Table 2-36 show the optional ports that, if enabled, allow the monitoring and control of some transceiver ports. When not selected, these ports are tied to their default values.

IMPORTANT: The Dynamic Reconfiguration Port is only available if this option is selected. Driving the DRP interface should be done only after assertion of the `gt0_rxresetdone_out` signal which indicates the completion of RX reset sequence.

Table 2-35: **Transceiver Control and Status Ports (7 Series and Zynq-7000 Devices)**

Signal	Direction	Description
gt0_drp_addr_in[8:0]	Input	DRP address bus
gt0_drpi_in[15:0]	Input	Data bus for writing configuration data to the transceiver.
gt0_drpo_out[15:0]	Output	Data bus for reading configuration data from the transceiver.
gt0_drprdy_out	Output	Indicates operation is complete for write operations and data is valid for read operations.
gt0_drpwe_in	Input	DRP write enable
gt0_drpclk_in	Input	DRP Clock
gt0_rxchariscomma_out[3:0]	Output	GT Status
gt0_rxcharisk_out[3:0]	Output	
gt0_rxbyteisaligned_out	Output	
gt0_rxbyterealign_out	Output	
gt0_rxcommadet_out	Output	
gt0_txdiffctrl_in[3:0]	Input	GT TX Driver
gt0_txpostcursor_in[4:0]	Input	
gt0_txprecursor_in[4:0]	Input	
gt0_txpolarity_in	Input	GT Polarity
gt0_rxpolarity_in	Input	
gt0_txprbssel_in[2:0]	Input	GT PRBS
gt0_txprbsforceerr_in	Input	
gt0_rxprbscntreset_in	Input	
gt0_rxprbserr_out	Output	
gt0_rxprbssel_in[2:0]	Input	
gt0_loopback_in[2:0]	Input	GT Loopback
gt0_txresetdone_out	Output	GT Status
gt0_rxresetdone_out	Output	
gt0_rxdisperr_out[3:0]	Output	
gt0_rxnotintable_out	Output	
gt0_eyescanreset_in[3:0]	Input	GT Eye Scan
gt0_eyescandataerror_out	Output	
gt0_eyescantrigger_in	Input	
gt0_rxcdrhold_in	Input	GT CDR
gt0_rxcdrlock_out	Output	

Table 2-35: **Transceiver Control and Status Ports (7 Series and Zynq-7000 Devices) *(Cont'd)***

Signal	Direction	Description
gt0_rxlpmen_in	Input	GT GTX/GTH RX Decision Feedback Equalizer (DFE)
gt0_rxdfelpmreset_in	Input	
gt0_rxdfeagcovrden_in	Input	
gt0_rxmonitorout_out[6:0]	Output	
gt0_rxmonitorsel_in[1:0]	Input	
gt0_txpmareset_in	Input	GT TX-PMA Reset
gt0_txpcsreset_in	Input	GT TX-PCS Reset
gt0_rxpmareset_in	Input	GT RX-PMA Reset
gt0_rxpcsreset_in	Input	GT RX-PCS Reset
gt0_rxbufreset_in	Input	GT receive elastic buffer Reset
gt0_rxpmaresetdone_out	Output	GT PMA resetdone indication
gt0_txbufstatus_out[1:0]	Output	GT TX Buffer status
gt0_rxbufstatus_out[2:0]	Output	GT RX Buffer status
gt0_dmonitorout_out[16:0]	Output	GT Status. If width differs for particular family then LSBs valid.
gt0_rxlpmreset_in	Input	RX LPM reset. Valid only for GTP.
gt0_rxlpmhfoverden_in	Input	RX LPM-HF override enable. Valid only for GTP.

Table 2-36: **Transceiver Control and Status Ports (UltraScale Architecture Devices)**

Signal	Direction	Description
gt_drp_addr_in[8:0]	Input	DRP address bus
gt_drpi_in[15:0]	Input	Data bus for writing configuration data to the transceiver.
gt_drpo_out[15:0]	Output	Data bus for reading configuration data from the transceiver.
gt_drprdy_out	Output	Indicates operation is complete for write operations and data is valid for read operations.
gt_drpwe_in	Input	DRP write enable.
gt_drpclk_in	Input	DRP Clock. For UltraScale architecture devices this must be 50 MHz.
gt_rxcommadet_out	Output	
gt_txdiffctrl_in[3:0]	Input	GT TX Driver
gt_txpostcursor_in[4:0]	Input	
gt_txprecursor_in[4:0]	Input	
gt_txpolarity_in	Input	GT Polarity
gt_rxpolarity_in	Input	

Table 2-36: **Transceiver Control and Status Ports (UltraScale Architecture Devices) *(Cont'd)***

Signal	Direction	Description
gt_txprbssel_in[2:0]	Input	GT PRBS
gt_txprbsforceerr_in	Input	
gt_rxprbscntreset_in	Input	
gt_rxprbserr_out	Output	
gt_rxprbssel_in[2:0]	Input	
gt_loopback_in[2:0]	Input	GT Loopback
gt_txresetdone_out	Output	GT Status
gt_rxresetdone_out	Output	
gt_rxdisperr_out[3:0]	Output	
gt_rxnotintable_out	Output	
gt_eyescanreset_in[3:0]	Input	GT Eye Scan
gt_eyescandataerror_out	Output	
gt_eyescantrigger_in	Input	
gt_rxcdrhold_in	Input	GT CDR
gt_rxcdrlock_out	Output	
gt_rxlpmen_in	Input	GT GTX/GTH RX Decision Feedback Equalizer (DFE)
gt_rxdfelpmreset_in	Input	
gt_txpmareset_in	Input	GT TX-PMA Reset
gt_txpcsreset_in	Input	GT TX-PCS Reset
gt_rxpmareset_in	Input	GT RX-PMA Reset
gt_rxpcsreset_in	Input	GT RX-PCS Reset
gt_rxbufreset_in	Input	GT Receive Elastic Buffer Reset
gt_rxpmaresetdone_out	Output	GT PMA resetdone indication
gt_txbufstatus_out[1:0]	Output	GT TX Buffer status
gt_rxbufstatus_out[2:0]	Output	GT RX Buffer status
gt_dmonitorout_out[16:0]	Output	GT Status

Register Space

This section provides general guidelines for configuring and monitoring the 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).

MDIO Bus System

The MDIO interface for 1 Gb/s operation (and slower speeds) is defined in IEEE 802.3-2008, clause 22. Figure 2-5 shows an example MDIO bus system. This two-wire interface consists of a clock (MDC) and a shared serial data line (MDIO). The maximum permitted frequency of Management Data Clock (MDC) is set at 2.5 MHz. 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).

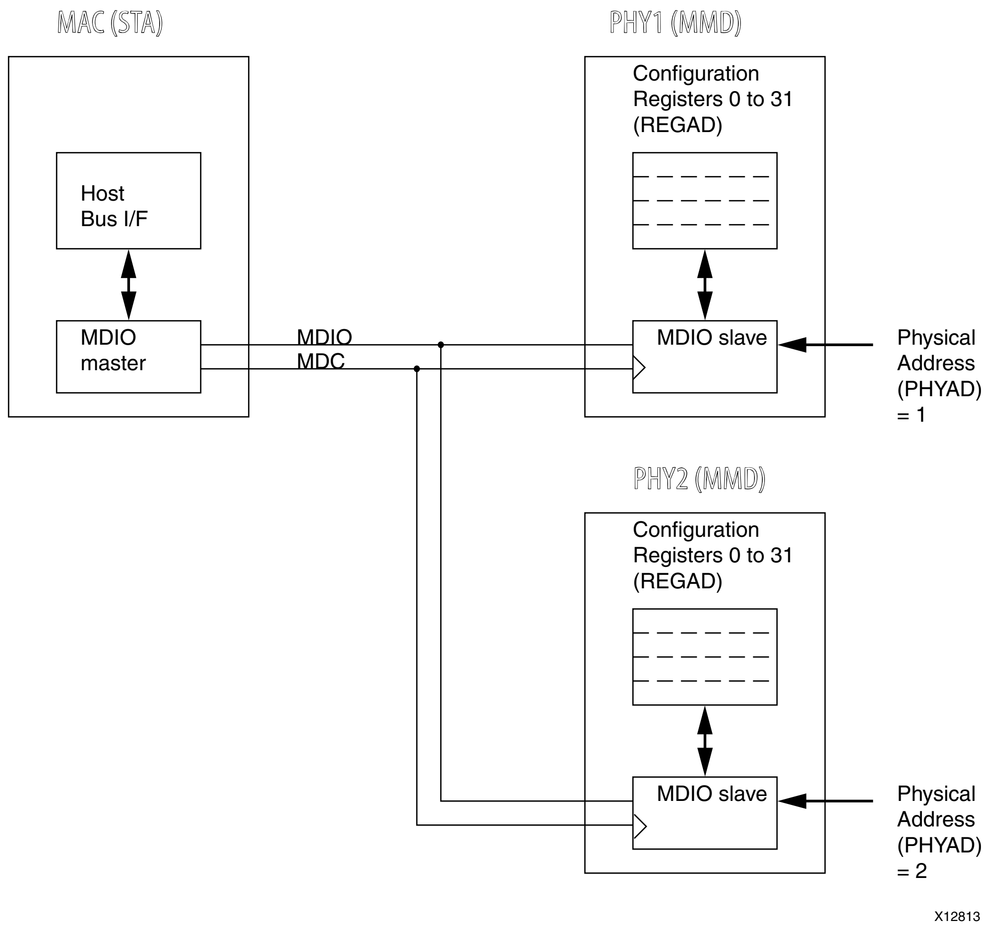

MAC (STA)
Host Bus I/F
MDIO master
MDIO
MDC
PHY1 (MMD)
Configuration Registers 0 to 31 (REGAD)
MDIO slave
Physical Address (PHYAD) = 1
PHY2 (MMD)
Configuration Registers 0 to 31 (REGAD)
MDIO slave
Physical Address (PHYAD) = 2
X12813

Figure 2-5: **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 shown, the Management Host Bus I/F of the Ethernet MAC is able to access the configuration and status registers of two PHY devices using the MDIO bus.

MDIO Transactions

All transactions, read or write, are initiated by the MDIO master. All MDIO slave devices, when addressed, must respond. MDIO transactions take the form of an MDIO frame, containing fields for transaction type, address and data. This MDIO frame is transferred across the MDIO wire synchronously to MDC. The abbreviations are used in this section are explained in Table 2-37.

Table 2-37: **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 2-6 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.

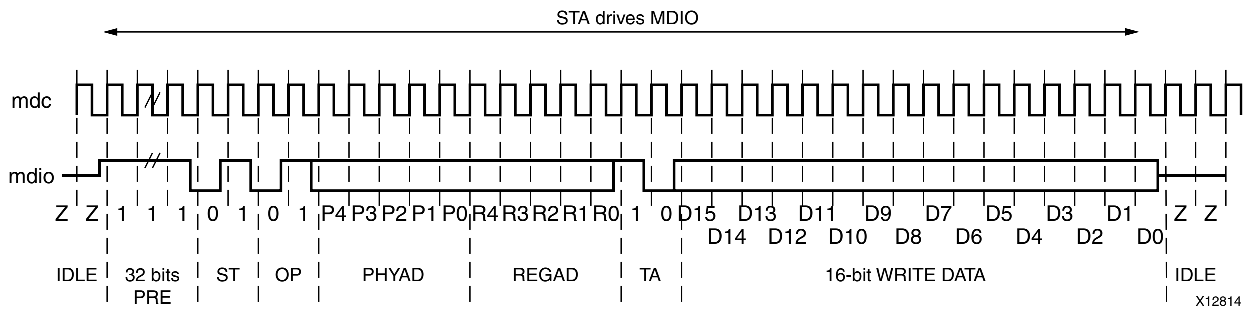

STA drives MDIO
mdc
mdio
Z Z 1 1 1 0 1 0 1 P4 P3 P2 P1 P0 R4 R3 R2 R1 R0 1 0 D15 D13 D11 D9 D7 D5 D3 D1 Z Z
D14 D12 D10 D8 D6 D4 D2 D0
IDLE
32 bits PRE
ST
OP
PHYAD
REGAD
TA
16-bit WRITE DATA
IDLE
X12814

Figure 2-6: **MDIO Write Transaction**

Read Transaction

Figure 2-7 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.

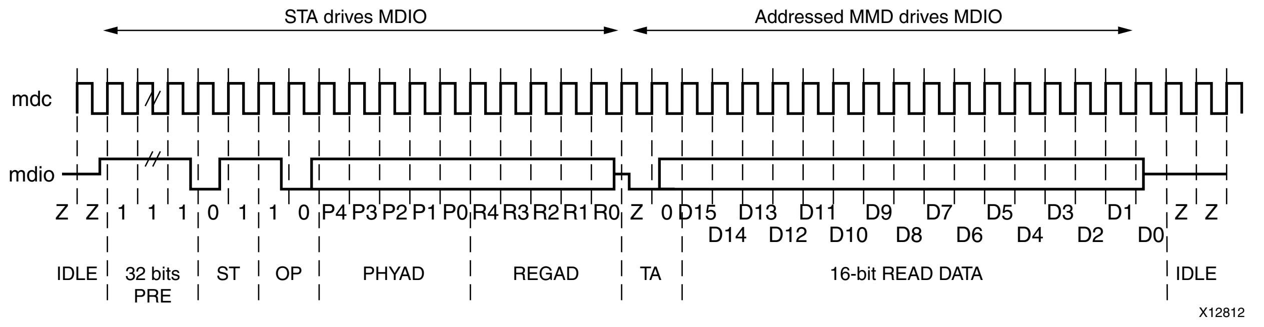

STA drives MDIO
Addressed MMD drives MDIO
mdc
mdio
Z Z 1 1 1 0 1 1 0 P4 P3 P2 P1 P0 R4 R3 R2 R1 R0 Z 0 D15 D13 D11 D9 D7 D5 D3 D1 Z Z
D14 D12 D10 D8 D6 D4 D2 D0
IDLE
32 bits PRE
ST
OP
PHYAD
REGAD
TA
16-bit READ DATA
IDLE
X12812

Figure 2-7: **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 2-5, 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 802.3-2008. 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 802.3-2008.

Connecting the MDIO to an Internally Integrated STA

The MDIO ports of the 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 Interfacing to Other Cores).

Connecting the MDIO to an External STA

Figure 2-8 shows the MDIO ports of the 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 3-state buffer to create a bidirectional wire, `mdio`. This 3-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.

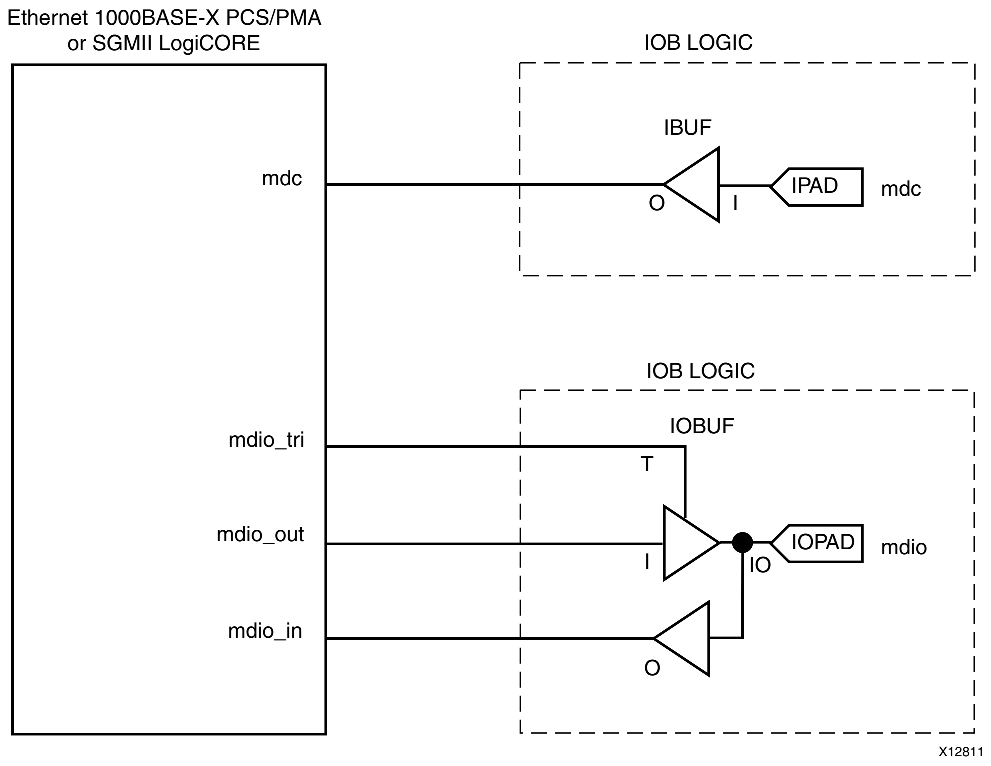


Figure 2-8: **Creating an External MDIO Interface**

Management Registers

The contents of the management registers can be accessed using the REGAD field of the MDIO frame. Contents vary depending on the IP catalog tool options, and are defined in the following sections in this chapter.

The core can be reset three ways: reset, DCM_LOCKED and soft reset. All of these methods reset all the registers to their default values.

1000BASE-X Standard Using Optional Auto-Negotiation

More information on the 1000BASE-X PCS registers can be found in clause 22 and clause 37 of the IEEE 802.3-2006 specification. Registers at undefined addresses are read-only and return 0s. The core can be reset three ways: reset, DCM_LOCKED and soft reset. All of these methods reset all the registers to the default values.

Table 2-38: **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

Note: In the following register definitions, R/W is Read/Write, RO is Read Only.

Register 0: Control Register

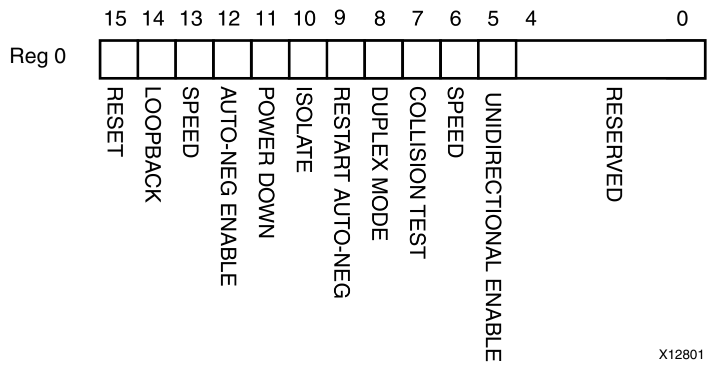


Figure 2-9: **MDIO Register 0: Control Register**

This register can also be programmed using the optional configuration interface.

Table 2-39: **Control Register (Register 0)**

Bits	Name	Description	Attributes	Default Value
0.15	Reset	1 = Core Reset 0 = Normal Operation	R/W 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.	R/W	0
0.13	Speed Selection (LSB)	Always returns a 0 for this bit. Together with bit 0.6, speed selection of 1000 Mb/s is identified	Returns 0	0
0.12	Auto-Negotiation Enable	1 = Enable Auto-Negotiation Process 0 = Disable Auto-Negotiation Process	R/W	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.	R/W	0
0.10	Isolate	1 = Electrically Isolate PHY from GMII 0 = Normal operation	R/W	1
0.9	Restart Auto-Negotiation	1 = Restart Auto-Negotiation Process 0 = Normal Operation	R/W 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 Mb/s is identified.	Returns 1	1
0.5	Unidirectional Enable	Enable transmit regardless of whether a valid link has been established. This feature is only possible if Auto-Negotiation Enable bit 0.12 is disabled	R/W	0
0.4:0	Reserved	Always return 0s, writes ignored.	Returns 0s	00000

Register 1: Status Register

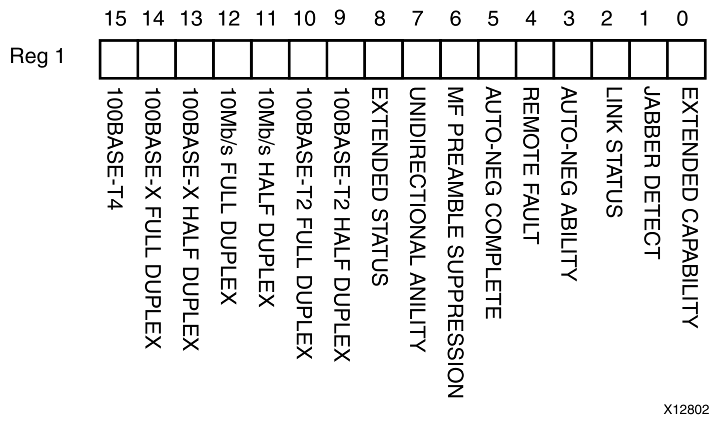


Figure 2-10: **MDIO Register 1: Status Register**

Table 2-40: **Status Register (Register 1)**

Bits	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 Mb/s Full Duplex	Always returns a 0 as 10 Mb/s full duplex is not supported.	Returns 0	0
1.11	10 Mb/s Half Duplex	Always returns a 0 as 10 Mb/s 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	RO	0
1.4	Remote Fault	1 = Remote fault condition detected 0 = No remote fault condition detected	RO Self clearing on read	0

Table 2-40: **Status Register (Register 1)** ***(Cont'd)***

Bits	Name	Description	Attributes	Default Value
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.	RO Self clearing on read	0
1.1	Jabber Detect	Always returns a 0 for this bit because 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 after this register was last read; synchronization of the link has been obtained and Auto-Negotiation (if enabled) has completed and the reset sequence of the transceiver (if present) 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 after this register was previously read. However, the current link status might be good. *Therefore read this register a second 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

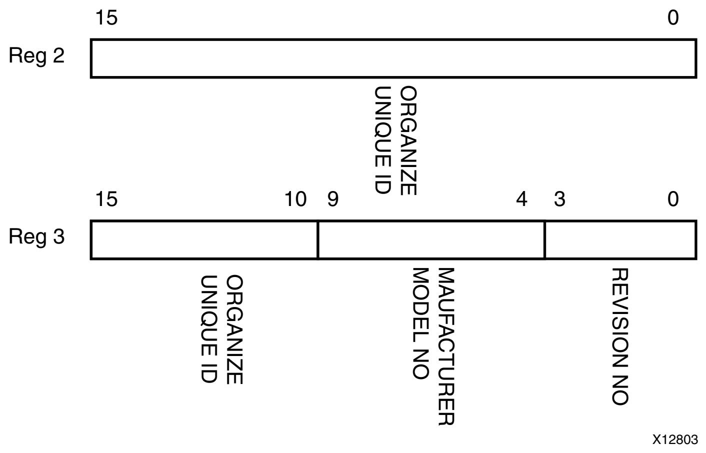


Figure 2-11: **Registers 2 and 3: PHY Identifiers**

Table 2-41: **PHY Identifier (Registers 2 and 3)**

Bits	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

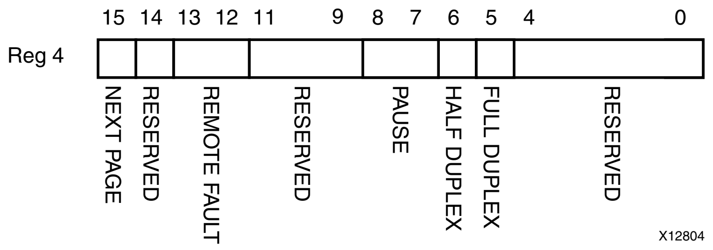


Figure 2-12: **MDIO Register 4: Auto-Negotiation Advertisement**

This register can also be programmed using the Optional Auto-Negotiation Configuration interface.

Table 2-42: **Auto-Negotiation Advertisement Register (Register 4)**

Bits	Name	Description	Attributes	Default Value
4.15	Next Page	Core currently does not support Next Page. Can be enabled, if requested. Writes ignored.	R/W	0
4.14	Reserved	Always returns 0, writes ignored	Returns 0	0

Table 2-42: **Auto-Negotiation Advertisement Register (Register 4)** ***(Cont'd)***

Bits	Name	Description	Attributes	Default Value
4.13:12	Remote Fault	00 = No Error 01 = Offline 10 = Link Failure 11 = Auto-Negotiation Error	R/W 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	R/W	11
4.6	Half Duplex	Always returns a 0 for this bit because 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	R/W	1
4.4:0	Reserved	Always return 0s, writes ignored	Returns 0s	00000

Register 5: Auto-Negotiation Link Partner Base

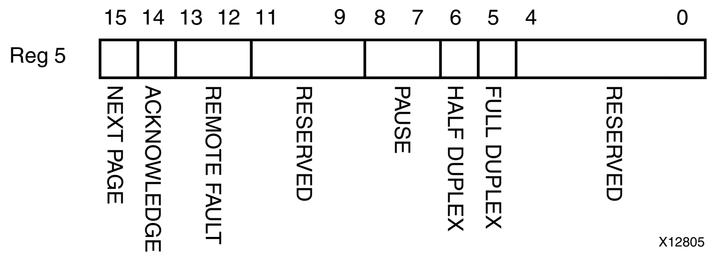


Figure 2-13: **MDIO Register 5: Auto-Negotiation Link Partner Base**

Table 2-43: **Auto-Negotiation Link Partner Ability Base Register (Register 5)**

Bits	Name	Description	Attributes	Default Value
5.15	Next Page	1 = Next Page functionality is supported 0 = Next Page functionality is not supported	RO	0
5.14	Acknowledge	Used by Auto-Negotiation function to indicate reception of a link partner base or next page	RO	0
5.13:12	Remote Fault	00 = No Error 01 = Offline 10 = Link Failure 11 = Auto-Negotiation Error	RO	00
5.11:9	Reserved	Always return 0s	Returns 0s	000

Table 2-43: **Auto-Negotiation Link Partner Ability Base Register (Register 5) *(Cont'd)***

Bits	Name	Description	Attributes	Default Value
5.8:7	Pause	00 = No PAUSE 01 = Symmetric PAUSE 10 = Asymmetric PAUSE towards link partner 11 = Both Symmetric PAUSE and Asymmetric PAUSE supported	RO	00
5.6	Half Duplex	1 = Half Duplex Mode is supported 0 = Half Duplex Mode is not supported	RO	0
5.5	Full Duplex	1 = Full Duplex Mode is supported 0 = Full Duplex Mode is not supported	RO	0
5.4:0	Reserved	Always return 0s	Returns 0s	00000

Register 6: Auto-Negotiation Expansion

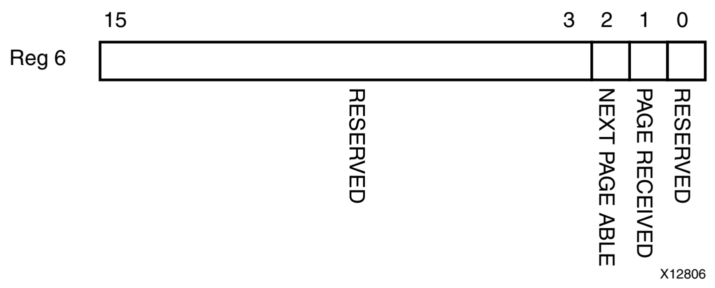


Figure 2-14: **MDIO Register 6: Auto-Negotiation Expansion**

Table 2-44: **Auto-Negotiation Expansion Register (Register 6)**

Bits	Name	Description	Attributes	Default Value
6.15:3	Reserved	Always returns 0s	Returns 0s	0000000000000
6.2	Next Page Able	This bit is ignored as the core currently does not support next page. This feature can be enabled on request.	Returns 1	1
6.1	Page Received	1 = A new page has been received 0 = A new page has not been received	RO Self clearing on read	0
6.0	Reserved	Always returns 0s	Returns 0s	0000000

Register 7: Next Page Transmit

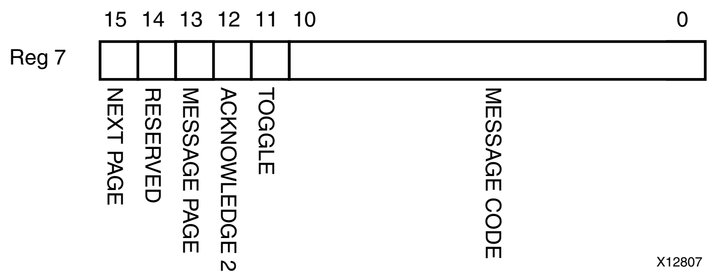


Figure 2-15: **MDIO Register 7: Next Page Transmit**

Table 2-45: **Auto-Negotiation Next Page Transmit (Register 7)**

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

1. This register returns zeros because the core currently does not support Next Page. This feature can be enabled on request.

Register 8: Next Page Receive

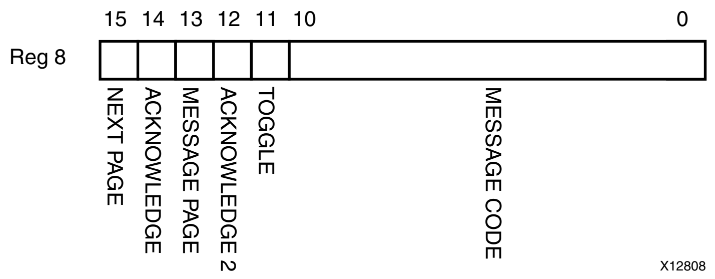


Figure 2-16: **MDIO Register 8: Next Page Receive**

Table 2-46: **Auto-Negotiation Next Page Receive (Register 8)**

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

Register 15: Extended Status

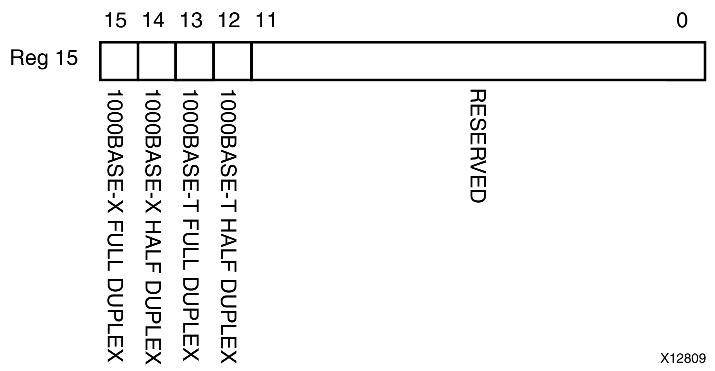


Figure 2-17: **MDIO Register 15: Extended Status Register**

Table 2-47: **Extended Status Register (Register 15)**

Bits	Name	Description	Attributes	Default Value
15.15	1000BASE-X Full Duplex	Always returns a 1 for this bit because 1000BASE-X Full Duplex is supported	Returns 1	1
15.14	1000BASE-X Half Duplex	Always returns a 0 for this bit because 1000BASE-X Half Duplex is not supported	Returns 0	0
15.13	1000BASE-T Full Duplex	Always returns a 0 for this bit because 1000BASE-T Full Duplex is not supported	Returns 0	0
15.12	1000BASE-T Half Duplex	Always returns a 0 for this bit because 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

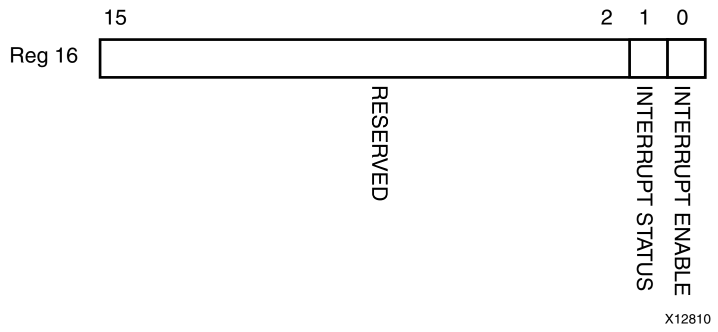


Figure 2-18: **MDIO Register 16: Vendor Specific Auto-Negotiation Interrupt Control**

Table 2-48: **Vendor Specific Register: Auto-Negotiation Interrupt Control Register (Register 16)**

Bits	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.	R/W	0
16.0	Interrupt Enable	1 = Interrupt enabled 0 = Interrupt disabled	R/W	1

1000BASE-X Standard Without Optional Auto-Negotiation

It is not in the scope of this document to fully describe the 1000BASE-X PCS registers. See clauses 22 and 37 of the IEEE 802.3-2008 specification for further information.

Registers at undefined addresses are read-only and return 0s. The core can be reset three ways: reset, DCM_LOCKED and soft reset. All of these methods reset all the registers to the default values.

Table 2-49: **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

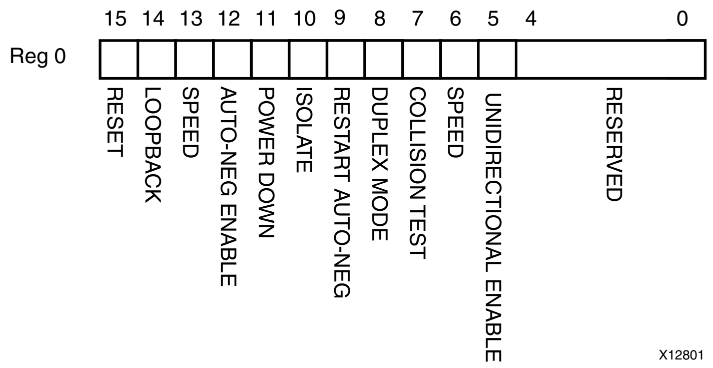


Figure 2-19: **MDIO Register 0: Control Register**

This register can also be programmed using the Optional Configuration interface.

Table 2-50: **Control Register (Register 0)**

Bits	Name	Description	Attributes	Default Value
0.15	Reset	1 = PCS/PMA reset 0 = Normal Operation	R/W 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.	R/W	0
0.13	Speed Selection (LSB)	Always returns a 0 for this bit. Together with bit 0.6, speed selection of 1000 Mb/s is identified.	Returns 0	0
0.12	Auto-Negotiation Enable	Ignore this bit because Auto-Negotiation is not included.	R/W	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.	R/W	0
0.10	Isolate	1 = Electrically Isolate PHY from GMII 0 = Normal operation	R/W	1

Table 2-50: **Control Register (Register 0) *(Cont'd)***

Bits	Name	Description	Attributes	Default Value
0.9	Restart Auto-Negotiation	Ignore this bit because Auto-Negotiation is not included.	R/W	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 1for this bit. Together with bit 0.13, speed selection of 1000 Mb/s is identified	Returns 1	1
0.5	Unidirectional Enable	Enables transmit irrespective of receive. Unidirectional feature is enabled automatically when this bit is set because optional Auto-Negotiation is not present.	R/W	0
0.4:0	Reserved	Always return 0s, writes ignored.	Returns 0s	00000

Register 1: Status Register

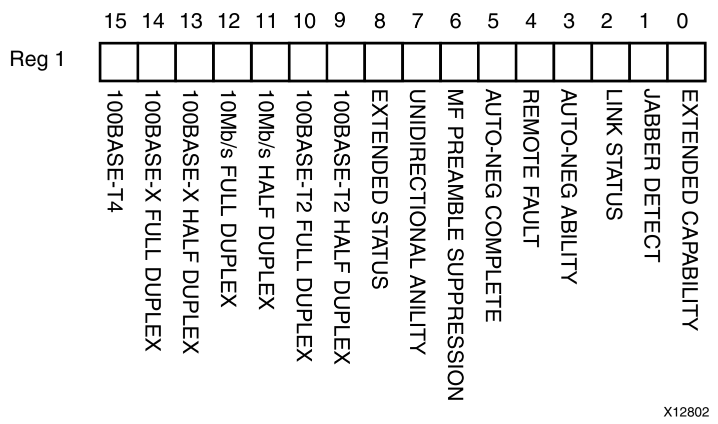


Figure 2-20: **MDIO Register 1: Status Register**

Table 2-51: **Status Register (Register 1)**

Bits	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 Mb/s Full Duplex	Always returns a 0 for this bit because 10 Mb/s Full Duplex is not supported	Returns 0	0

Table 2-51: **Status Register (Register 1)** ***(Cont'd)***

Bits	Name	Description	Attributes	Default Value
1.11	10 Mb/s Half Duplex	Always returns a 0 for this bit because 10 Mb/s 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
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.	RO Self clearing on read	0
1.1	Jabber Detect	Always returns a 0 for this bit because 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 after this register was last read; synchronization of the link has been obtained and the reset sequence of the transceiver (if present) has completed.

When Low, either:

- A valid link has not been established; link synchronization has failed.

  OR

- Link synchronization was lost at some point after this register was previously read. However, the current link status might be good. *Therefore read this register a second time to get confirmation of the current link status.*

Registers 2 and 3: PHY Identifier

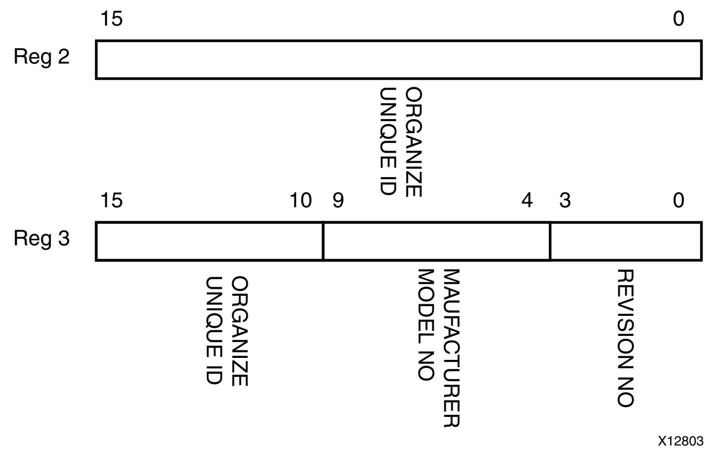


Figure 2-21: **MDIO Registers 2 and 3: PHY Identifier**

Table 2-52: **PHY Identifier (Registers 2 and 3)**

Bits	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

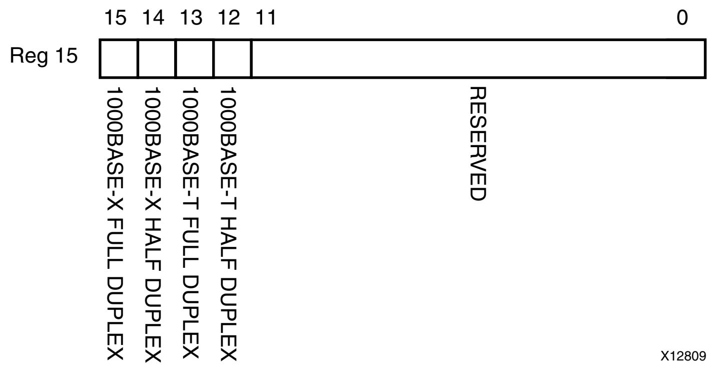


Figure 2-22: **MDIO Register 15: Extended Status**

Table 2-53: **Extended Status (Register 15)**

Bits	Name	Description	Attributes	Default Value
15.15	1000BASE-X Full Duplex	Always returns a 1 because 1000BASE-X Full Duplex is supported	Returns 1	1
15.14	1000BASE-X Half Duplex	Always returns a 0 because 1000BASE-X Half Duplex is not supported	Returns 0	0
15.13	1000BASE-T Full Duplex	Always returns a 0 because 1000BASE-T Full Duplex is not supported	Returns 0	0
15.12	1000BASE-T Half Duplex	Always returns a 0 because 1000BASE-T Half Duplex is not supported	Returns 0	0
15:11:0	Reserved	Always return 0s	Returns 0s	000000000000

SGMII Standard Using Optional Auto-Negotiation

The registers provided for SGMII operation in this core are adaptations of those defined in clauses 22 and 37 of the IEEE 802.3-2008 specification. 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 3-48, page 151.

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. The core can be reset three ways: reset, DCM_LOCKED and soft reset. All of these methods reset all the registers to the default values.

Table 2-54: **MDIO Registers for SGMII 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

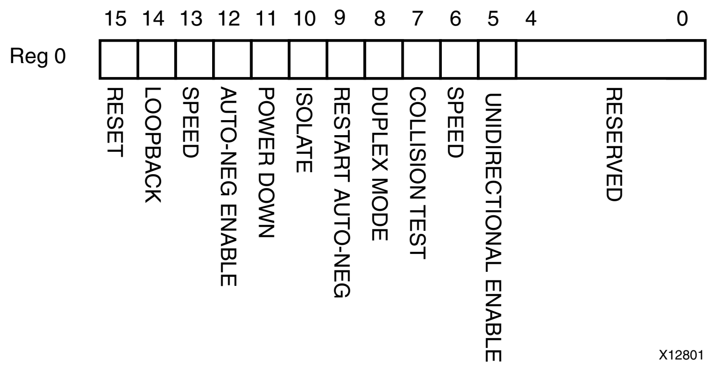


Figure 2-23: **MDIO Register 0: SGMII Control**

This register can also be programmed using the Optional Configuration interface.

Table 2-55: **SGMII Control (Register 0)**

Bits	Name	Description	Attributes	Default Value
0.15	Reset	1 = Core Reset 0 = Normal Operation	R/W 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.	R/W	0
0.13	Speed Selection (LSB)	11 = Reserved 10 = 1 Gb/s 01 = 100 Mb/s 00 = 10 Mb/s Zynq-7000 AP SoC PS Gigabit Ethernet Controller mode, identifies with bit 0.13 of Control register specified in IEEE 802.3-2008. Returns 0 in any other mode, together with bit 0.6, speed selection of 1000 Mb/s is identified	R/W in Zynq-7000 AP SoC PS Gigabit Ethernet Controller mode. Returns 0 in any other mode	0
0.12	Auto-Negotiation Enable	1 = Enable SGMII Auto-Negotiation Process 0 = Disable SGMII Auto-Negotiation Process	R/W	1

Table 2-55: **SGMII Control (Register 0)** ***(Cont'd)***

Bits	Name	Description	Attributes	Default Value
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.	R/W	0
0.10	Isolate	1 = Electrically Isolate SGMII logic from GMII 0 = Normal operation	R/W	1
0.9	Restart Auto-Negotiation	1 = Restart Auto-Negotiation Process across SGMII link 0 = Normal Operation	R/W 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)	11 = Reserved 10 = 1 Gb/s 01 = 100 Mb/s 00 = 10 Mb/s Zynq-7000 AP SoC PS Gigabit Ethernet Controller mode, identifies with bit 0.6 of Control register specified in IEEE 802.3-2008. Returns 1 in any other mode, together with bit 0.13, speed selection of 1000 Mb/s is identified	R/W in Zynq-7000 AP SoC PS Gigabit Ethernet Controller mode. Returns 1 in any other mode	1
0.5	Unidirectional Enable	Enable transmit regardless of whether a valid link has been established. This feature is only possible if Auto-Negotiation Enable bit 0.12 is disabled.	R/W	0
0.4:0	Reserved	Always return 0s; writes ignored	Returns 0s	00000

Register 1: SGMII Status

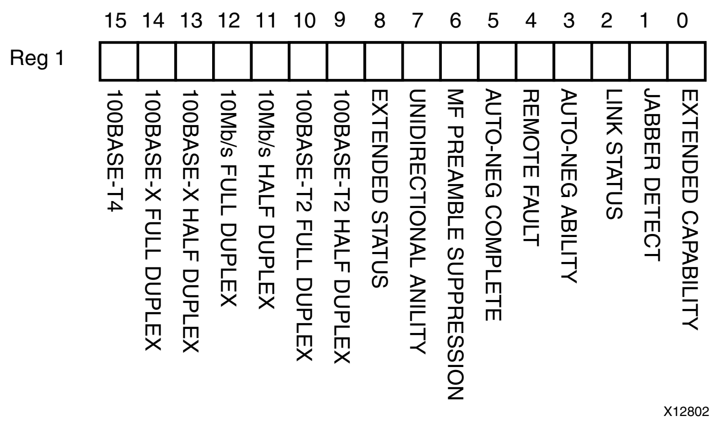


Figure 2-24: **MDIO Register 1: SGMII Status**

Table 2-56: **SGMII Status (Register 1)**

Bits	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 Mb/s Full Duplex	Always returns a 0 for this bit because 10 Mb/s Full Duplex is not supported	Returns 0	0
1.11	10 Mb/s Half Duplex	Always returns a 0 for this bit because 10 Mb/s 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
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	RO	0
1.4	Remote Fault	1 = A fault on the Medium has been detected 0 = No fault of the Medium has been detected	RO Self clearing on read	0

Table 2-56: **SGMII Status (Register 1)** ***(Cont'd)***

Bits	Name	Description	Attributes	Default Value
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.	RO Self clearing on read	0
1.1	Jabber Detect	Always returns a 0 for this bit because 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 after this register was last read: synchronization of the link has been obtained and Auto-Negotiation (if enabled) has completed and the reset sequence of the transceiver (if present) 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 when the register was previously read. However, the current link status might be good. *Therefore read this register a second 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

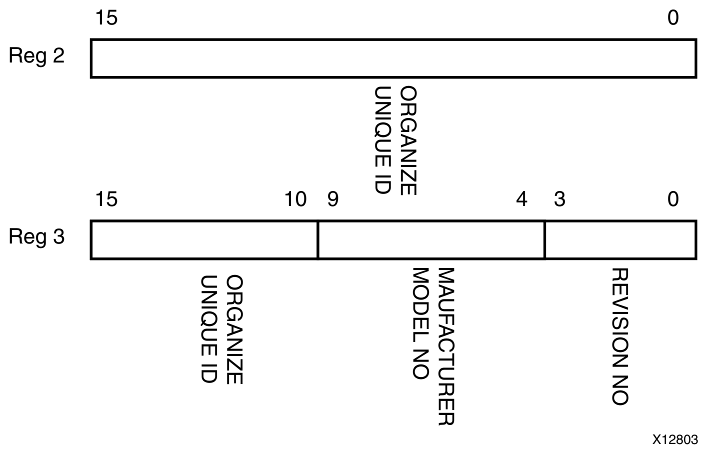


Figure 2-25: **MDIO Registers 2 and 3: PHY Identifier**

Table 2-57: **PHY Identifier (Registers 2 and 3)**

Bits	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

MAC Mode of Operation

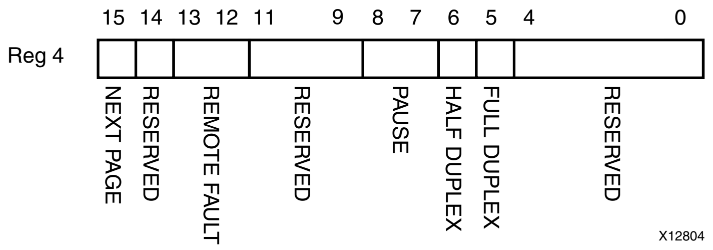


Figure 2-26: **MDIO Register 4: SGMII Auto-Negotiation Advertisement**

This register can also be programmed using the Optional Auto-Negotiation Configuration interface.

Table 2-58: **SGMII Auto-Negotiation Advertisement (Register 4)**

Bits	Name	Description	Attributes	Default Value
4.15:0	All bits	SGMII defined value sent from the MAC to the PHY	RO	0000000000000001

PHY Mode of Operation

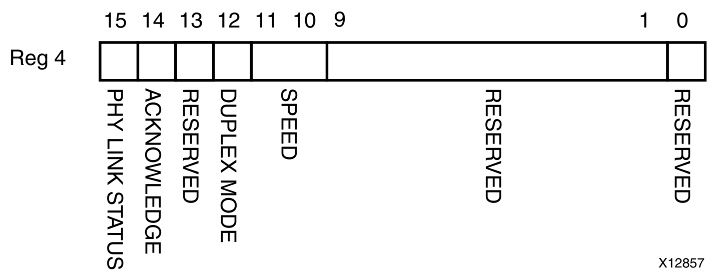


Figure 2-27: **MDIO Register 4: SGMII Auto-Negotiation Advertisement**

Table 2-59: **SGMII Auto-Negotiation Advertisement in PHY Mode (Register 4)**

Bits	Name	Description	Attributes	Default Value
4.15	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	R/W	0
4.14	Acknowledge	Used by Auto-Negotiation function to indicate reception of a link partner base or next page	R/W	0
4.13	Reserved	Always returns 0, writes ignored	Returns 0	0
4.12	Duplex Mode	1= Full Duplex 0 = Half Duplex	R/W	0
4.11:10	Speed	11 = Reserved 10 = 1 Gb/s 01 = 100 Mb/s 00 = 10 Mb/s	R/W	00
4.9:1	Reserved	Always return 0s	Returns 0s	000000000
4:0	Reserved	Always returns 1	Returns 1	1

Register 5: SGMII Auto-Negotiation Link Partner Ability

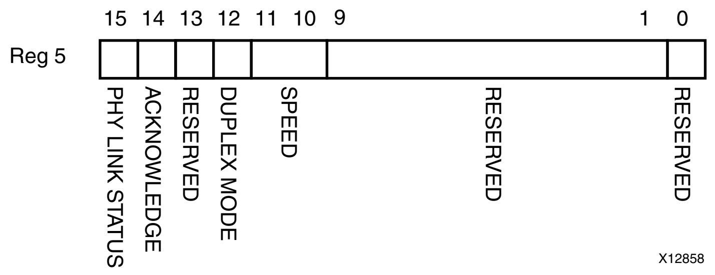


Figure 2-28: **MDIO Register 5: SGMII Auto-Negotiation Link Partner Ability**

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 2-60: **SGMII Auto-Negotiation Link Partner Ability Base (Register 5)**

Bits	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	RO	1
5.14	Acknowledge	Used by Auto-Negotiation function to indicate reception of a link partner base or next page	RO	0
5.13	Reserved	Always returns 0 writes ignored	Returns 0	0
5.12	Duplex Mode	1= Full Duplex 0 = Half Duplex	RO	0
5.11:10	Speed	11 = Reserved 10 = 1 Gb/s 01 = 100 Mb/s 00 = 10 Mb/s	RO	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

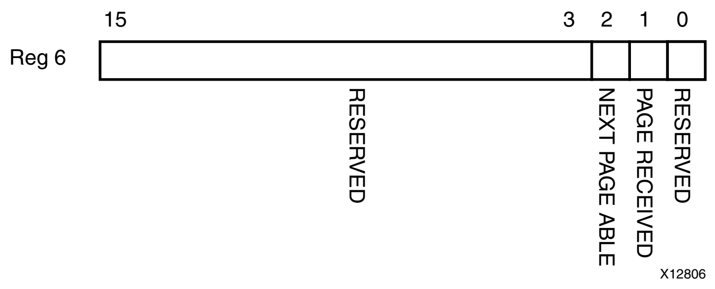


Figure 2-29: **MDIO Register 6: SGMII Auto-Negotiation Expansion**

Table 2-61: **SGMII Auto-Negotiation Expansion (Register 6)**

Bits	Name	Description	Attributes	Default Value
6.15:3	Reserved	Always return 0s	Returns 0s	0000000000000
6.2	Next Page Able	This bit is ignored as the core currently does not support next page. This feature can be enabled on request.	Returns 1	1
6.1	Page Received	1 = A new page has been received 0 = A new page has not been received	RO Self clearing on read	0
6.0	Reserved	Always return 0s	Returns 0s	0000000

Register 7: SGMII Auto-Negotiation Next Page Transmit

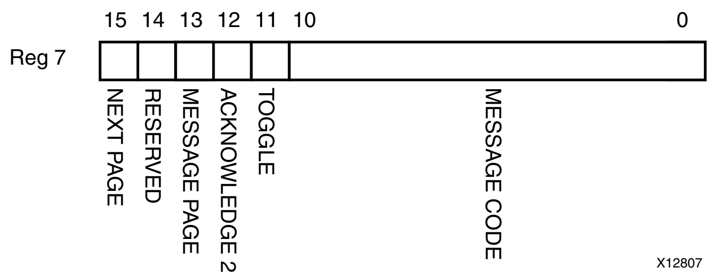


Figure 2-30: **MDIO Register 7: SGMII Auto-Negotiation Next Page Transmit**

Table 2-62: **SGMII Auto-Negotiation Next Page Transmit (Register 7)**

Bits	Name	Description	Attributes	Default Value[1]
7.15	Next Page	1 = Additional Next Page(s) will follow 0 = Last page	R/W	0
7.14	Reserved	Always returns 0	Returns 0	0
7.13	Message Page	1 = Message Page 0 = Unformatted Page	R/W	0

Table 2-62: **SGMII Auto-Negotiation Next Page Transmit (Register 7)**

Bits	Name	Description	Attributes	Default Value[1]
7.12	Acknowledge 2	1 = Comply with message 0 = Cannot comply with message	R/W	0
7.11	Toggle	Value toggles between subsequent Next Pages	RO	0
7.10:0	Message / Unformatted Code Field	Message Code Field or Unformatted Page Encoding as dictated by 7.13	R/W	00000000001 (Null Message Code)

1. This register returns zeros because the core does not support Next Page. The feature can be enabled, if requested.

Register 8: SGMII Next Page Receive

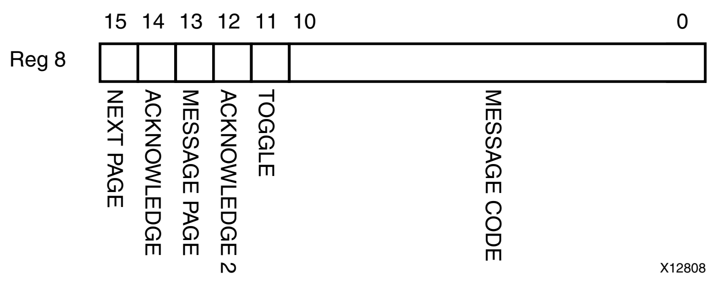


Figure 2-31: **MDIO Register 8: SGMII Next Page Receive**

Table 2-63: **SGMII Auto-Negotiation Next Page Receive (Register 8)**

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

Register 15: SGMII Extended Status

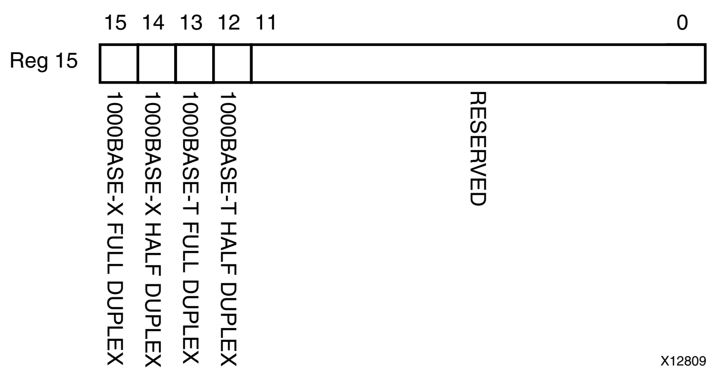


Figure 2-32: **MDIO Register 15: SGMII Extended Status**

Table 2-64: **SGMII Extended Status Register (Register 15)**

Bits	Name	Description	Attributes	Default Value
15.15	1000BASE-X Full Duplex	Always returns a 1 for this bit because 1000BASE-X Full Duplex is supported	Returns 1	1
15.14	1000BASE-X Half Duplex	Always returns a 0 for this bit because 1000BASE-X Half Duplex is not supported	Returns 0	0
15.13	1000BASE-T Full Duplex	Always returns a 0 for this bit because 1000BASE-T Full Duplex is not supported	Returns 0	0
15.12	1000BASE-T Half Duplex	Always returns a 0 for this bit because 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

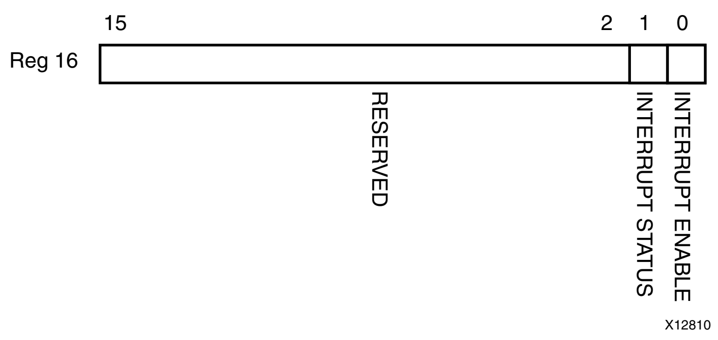


Figure 2-33: **MDIO Register 16: SGMII Auto-Negotiation Interrupt Control**

Table 2-65: **SGMII Auto-Negotiation Interrupt Control (Register 16)**

Bits	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. The an_interrupt port of the core is wired to this bit.	R/W	0
16.0	Interrupt Enable	1 = Interrupt enabled 0 = Interrupt disabled	R/W	1

SGMII Standard without Optional Auto-Negotiation

The registers provided for SGMII operation in this core are adaptations of those defined in clauses 22 and 37 of the IEEE 802.3-2008 specification. 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 3-48. Information about the state of the SGMII link is available in registers that follow.

IMPORTANT: *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.*

The core can be reset three ways: reset, DCM_LOCKED and soft reset. All of these methods reset all the registers to the default values.

Table 2-66: **MDIO Registers for SGMII 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

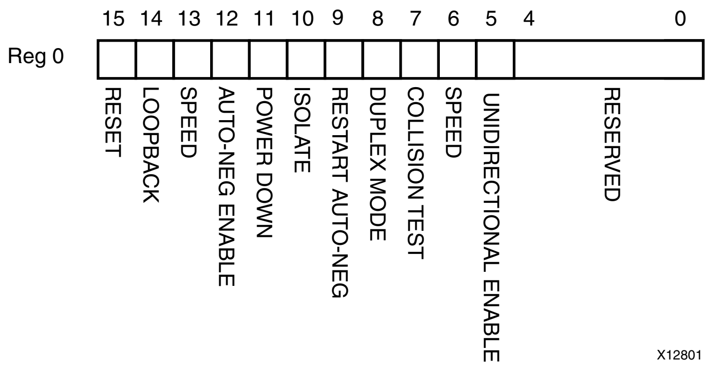


Figure 2-34: **MDIO Register 0: SGMII Control**

This register can also be programmed using the Optional Configuration interface.

Table 2-67: **SGMII Control (Register 0)**

Bits	Name	Description	Attributes	Default Value
0.15	Reset	1 = Core Reset 0 = Normal Operation	R/W 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.	R/W	0
0.13	Speed Selection (LSB)	Zynq-7000 AP SoC PS Gigabit Ethernet Controller mode, identifies with bit 0.6 of Control register specified in IEEE 802.3-2008. Returns 1 in any other mode, together with bit 0.13, speed selection of 1000 Mb/s is identified	R/W in Zynq-7000 AP SoC PS Gigabit Ethernet Controller mode. Returns 0 in any other mode	0
0.12	Auto-Negotiation Enable	1 = Enable SGMII Auto-Negotiation Process 0 = Disable SGMII Auto-Negotiation Process	R/W	1

Table 2-67: **SGMII Control (Register 0)** *(Cont'd)*

Bits	Name	Description	Attributes	Default Value
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.	R/W	0
0.10	Isolate	1 = Electrically Isolate SGMII logic from GMII 0 = Normal operation	R/W	1
0.9	Restart Auto-Negotiation	1 = Restart Auto-Negotiation Process across SGMII link 0 = Normal Operation	R/W 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)	Zynq-7000 AP SoC PS Gigabit Ethernet Controller mode, identifies with bit 0.6 of Control register specified in IEEE 802.3-2008. Returns 1 in any other mode, together with bit 0.13, speed selection of 1000 Mb/s is identified	R/W in Zynq-7000 AP SoC PS Gigabit Ethernet Controller mode. Returns 1 in any other mode	1
0.5	Unidirectional Enable	Enable transmit regardless of whether a valid link has been established	R/W	0
0.4:0	Reserved	Always return 0s, writes ignored	Returns 0s	00000

Register 1: SGMII Status

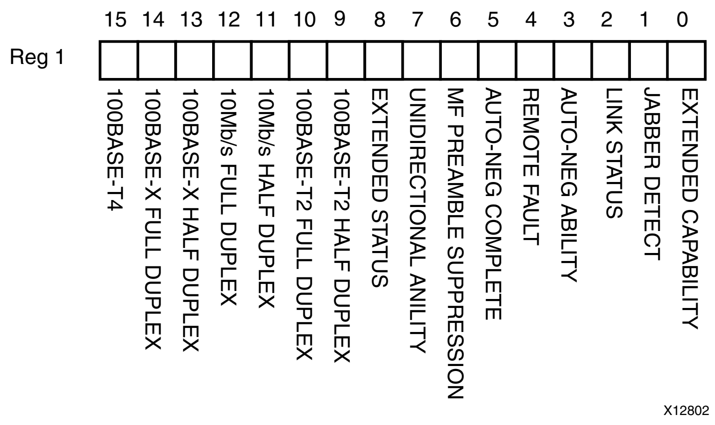


Figure 2-35: **MDIO Register 1: SGMII Status**

Table 2-68: **SGMII Status (Register 1)**

Bits	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 Mb/s Full Duplex	Always returns a 0 for this bit because 10 Mb/s Full Duplex is not supported	Returns 0	0
1.11	10 Mb/s Half Duplex	Always returns a 0 for this bit because 10 Mb/s 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
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.	RO Self clearing on read	0
1.1	Jabber Detect	Always returns a 0 for this bit because 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 after this register was last read; synchronization of the link has been obtained and the reset sequence of the transceiver (if present) has completed.

When Low, either:

- A valid link has not been established; link synchronization has failed.

  OR

- Link synchronization was lost at some point when this register was previously read. However, the current link status might be good. *Therefore read this register a second time to get confirmation of the current link status.*

Registers 2 and 3: PHY Identifier

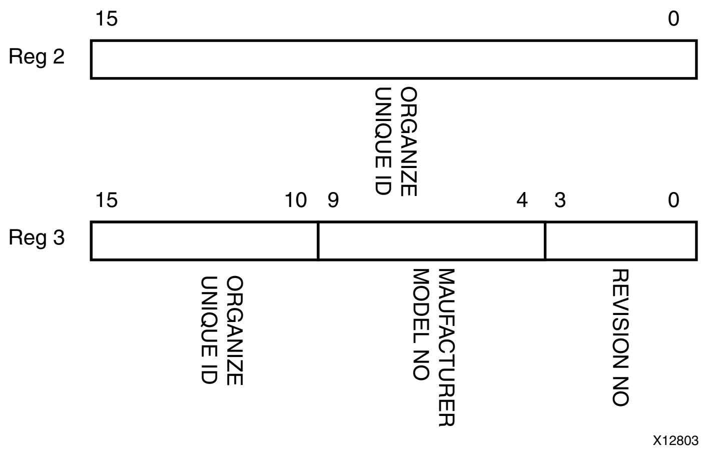


Figure 2-36: **MDIO Registers 2 and 3: PHY Identifier**

Table 2-69: **PHY Identifier (Registers 2 and 3)**

Bits	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

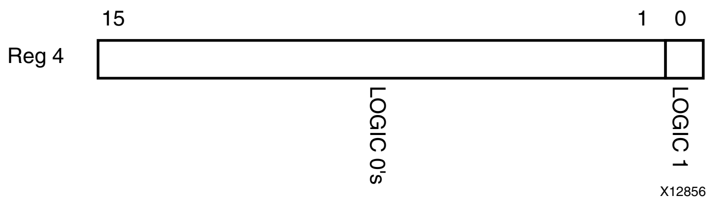


Figure 2-37: **MDIO Register 4: SGMII Auto-Negotiation Advertisement**

Table 2-70: **SGMII Auto-Negotiation Advertisement (Register 4)**

Bits	Name	Description	Attributes	Default Value
4.15:0	All bits	Ignore this register because Auto-Negotiation is not included	RO	0000000000000001

Register 15: SGMII Extended Status

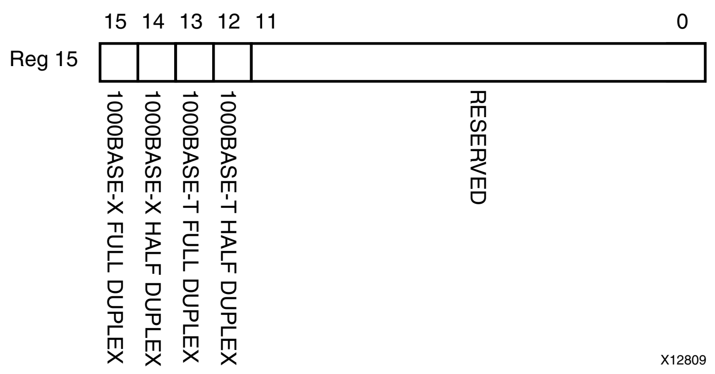


Figure 2-38: **MDIO Register 15: SGMII Extended Status**

Table 2-71: **SGMII Extended Status Register (Register 15)**

Bits	Name	Description	Attributes	Default Value
15.15	1000BASE-X Full Duplex	Always returns a 1 for this bit because 1000BASE-X Full Duplex is supported	Returns 1	1
15.14	1000BASE-X Half Duplex	Always returns a 0 for this bit because 1000BASE-X Half Duplex is not supported	Returns 0	0
15.13	1000BASE-T Full Duplex	Always returns a 0 for this bit because 1000BASE-T Full Duplex is not supported	Returns 0	0
15.12	1000BASE-T Half Duplex	Always returns a 0 for this bit because 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 2-72 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. The component name is used as the base name of the output files generated for the core. See Select Standard in Chapter 4.

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

Standard Without Optional Auto-Negotiation. This register can be written to at any time. See Dynamic Switching of 1000BASE-X and SGMII Standards for more information.

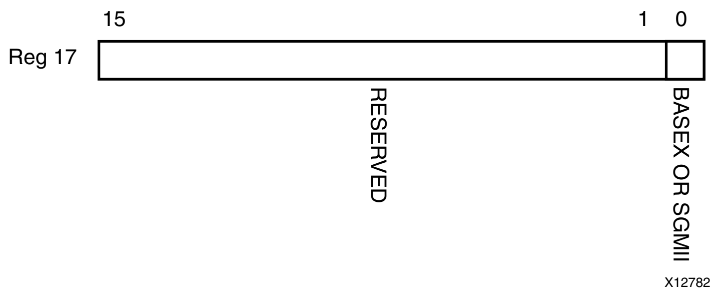


Figure 2-38: **Dynamic Switching (Register 17)**

Table 2-72: **Vendor-specific Register: Standard Selection Register (Register 17)**

Bits	Name	Description	Attributes	Default Value
17.15:1	Reserved	Always return 0s	Returns 0s	000000000000000
16.0	Standard	0 = Core performs to the 1000BASE-X standard. Registers 0 to 16 behave as per 1000BASE-X Standard Using Optional Auto-Negotiation 1= Core performs to the SGMII standard. Registers 0 to 16 behave as per SGMII Standard Using Optional Auto-Negotiation.	R/W	Determined by the basex_or_sgmii port

Configuration and Status Vectors

Additional signals are brought out of the core to program Register 0 independent of the MDIO management interface. These signals are bundled into the `configuration_vector` signal as defined in Table 2-73.

Signals are also brought out of the core to program Register 4 independent of the MDIO management interface. These signals are bundled into `an_adv_config_vector` as defined in Table 2-74. Status signals are also brought out of the core to `status_vector` as defined in Table 2-75.

Table 2-73: **Configuration Vector**

Bits	Description
0	**Unidirectional Enable.** When set to 1, Enable Transmit irrespective of state of RX (802.3ah). When set to 0, Normal operation
1	**Loopback Control.** When the core with a device-specific transceiver is used, this places the core into internal loopback mode. With the TBI version, Bit 1 is connected to ewrap. When set to 1, this signal indicates to the external PMA module to enter loopback mode.
2	**Power Down,** When the Zynq-7000, Virtex-7, Kintex-7, and Artix-7device transceivers are used and set to 1, the device-specific transceiver is placed in a low-power state. A reset must be applied to clear. With the TBI version this bit is unused.

Table 2-73: **Configuration Vector *(Cont'd)***

Bits	Description
3	**Isolate.** When set to 1, the GMII should be electrically isolated. When set to 0, normal operation is enabled.
4	**Auto-Negotiation Enable**. This signal is valid only if the AN module is enabled through the IP catalog. When set to 1, the signal enables the AN feature. When set to 0, AN is disabled.

Table 2-74: **Auto-Negotiation Vector**

Bits	Description[1]
0	For 1000BASE-X-Reserved. For SGMII- Always 1
4:1	Reserved
5	For 1000BASE-X- Full Duplex 1 = Full Duplex Mode is advertised 0 = Full Duplex Mode is not advertised For SGMII: Reserved
6	Reserved
8:7	For 1000BASE-X- Pause 0 0 = No Pause 0 1 = Symmetric Pause 1 0 = Asymmetric Pause towards link partner 1 1 = Both Symmetric Pause and Asymmetric Pause towards link partner For SGMII - Reserved
9	Reserved
11:10	For 1000BASE-X- Reserved For SGMII- Speed 1 1 = Reserved 1 0 = 1000 Mb/s 0 1 = 100 Mb/s 0 0 = 10 Mb/s
13:12	For 1000BASE-X- Remote Fault 0 0 = No Error 0 1 = Offline 1 0 = Link Failure 1 1 = Auto-Negotiation Error For SGMII- Bit[13]: Reserved Bit[12]: Duplex Mode 1 = Full Duplex 0 = Half Duplex
14	For 1000BASE-X- Reserved For SGMII- Acknowledge

Table 2-74: **Auto-Negotiation Vector *(Cont'd)***

Bits	Description[1]
	For 1000BASE-X- Reserved For SGMII- PHY Link Status 1 = Link Up 0 = Link Down

1. In SGMII operating in MAC Mode, the AN_ADV register is hard wired internally to "0x4001" and this bus has no effect. For 1000BASE-X and SGMII operating in PHY mode, the AN_ADV register is programmed by this bus as specified for the following bits.

Table 2-75: **Status Vector**

Bits	Description
0	**Link Status**. This signal 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 and the reset sequence of the transceiver (if present) 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. 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]. In this case, either of the bits can be used.
1	**Link Synchronization**. This signal indicates the state of the synchronization state machine (IEEE802.3 figure 36-9) which is based on the reception of valid 8B/10B 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.
2	**RUDI(/C/)**. The core is receiving /C/ ordered sets (Auto-Negotiation Configuration sequences) as defined in IEEE 802.3-2008 clause 36.2.4.10.
3	**RUDI(/I/)**. The core is receiving /I/ ordered sets (Idles) as defined in IEEE 802.3-2008 clause 36.2.4.12.
4	**RUDI(INVALID)**. The core has received invalid data while receiving/C/ or /I/ ordered set set as defined in IEEE 802.3-2008 clause 36.2.5.1.6. This can be caused, for example, by bit errors occurring in any clock cycle of the /C/ or /I/ ordered set.
5	**RXDISPERR.** The core has received a running disparity error during the 8B/10B decoding function.
6	**RXNOTINTABLE.** The core has received a code group which is not recognized from the 8B/10B coding tables.
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). The value reflected is Link Partner Base AN Register 5 bit 15 in SGMII MAC mode and the Advertisement Ability register 4 bit 15 in PHY mode. 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 operating in 1000BASE-X mode, this bit remains Low and should be ignored.

Table 2-75: **Status Vector *(Cont'd)***

Bits	Description
9:8	**Remote Fault Encoding.** This signal indicates the remote fault encoding (IEEE802.3 table 37-3). This signal is validated by bit 13 of status_vector and is only valid when Auto-Negotiation is enabled. In 1000BASE-X mode these values reflected Link Partner Base AN Register 5 bits [13:12]. This signal has no significance when the core is in SGMII mode with PHY side implementation and indicates 00. In MAC side implementation of the core the signal takes the value 10 to indicate the remote fault (Link Partner Base AN Register 5 bit 15 (Link bit) is 0).
11:10	**SPEED.** This signal indicates the speed negotiated and is only valid when Auto-Negotiation is enabled. In 1000BASE-X mode these bits are hard wired to 10 but in SGMII mode the signals encoding is as shown below. The value reflected is Link Partner Base AN Register 5 bits [11:10] in MAC mode and the Advertisement Ability register 4 bits [11:10] in PHY mode. 1 1 = Reserved 1 0 = 1000 Mb/s 0 1 = 100 Mb/s 0 0 = 10 Mb/s
12	**Duplex Mode.** This bit indicates the Duplex mode negotiated with the link partner. Indicates bit 5 of Link Partner Base AN register 5 in 1000BASE-X mode; otherwise bit 12 in SGMII mode (In SGMII MAC mode it is register bit 5.12, in SGMII PHY mode 4.12). 1 = Full Duplex 0 = Half Duplex
13	**Remote Fault**. When this bit is logic one, it indicates that a remote fault is detected and the type of remote fault is indicated by `status_vector bits[9:8]`. This bit reflects MDIO register bit 1.4. ***Note:*** This bit is only deasserted when a MDIO read is made to status register (register1). This signal has no significance in SGMII PHY mode.
15:14	**Pause.** These bits reflect the bits [8:7] of Register 5 (Link Partner Base AN register). These bits are valid only in 1000BASE-X mode and have no significance in SGMII mode of operation. 0 0 = No Pause 0 1 = Symmetric Pause 1 0 = Asymmetric Pause towards Link partner 1 1 = Both Symmetric Pause and Asymmetric Pause towards link partner

Designing with the Core

This chapter includes guidelines and additional information to make designing with the core easier.

General Design Guidelines

The following sections provide some design guidelines.

Understand the Core Features and Interfaces

Chapter 1, Overview introduced the features and interfaces that are present in the logic of the core netlist. This chapter assumes a working knowledge of the IEEE802.3-2008 Ethernet specification, in particular the Gigabit Ethernet 1000BASE-X sections: clauses 34 through to 37.

Customize and Generate the Core

Generate the core with your desired options using the IP catalog, as described in Customizing and Generating the Core.

Examine the Example Design Provided with the Core

An HDL example design built around the core is provided through the Vivado® design tools that allow for a demonstration of core functionality using either a simulation package or in hardware if placed on a suitable board.

Multiple different example designs are provided depending upon the core customization:

- 1000BASE-X with Transceiver Example Design
- SGMII/Dynamic Switching Using a Transceiver Example Design
- SGMII over LVDS Example Design
- 1000BASE-X with TBI Example Design
- SGMII/Dynamic Switching with TBI Example Design

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

1. Edit the HDL top level of the example design file to change the clocking scheme, add or remove Input/Output Blocks (IOBs) as required, and replace the GMII IOB logic with user-specific application logic (for example, an Ethernet MAC).
2. Synthesize the entire design.
3. Implement the entire design. After implementation is complete you can also create a bitstream that can be downloaded to a Xilinx device.
4. Download the bitstream to a target device.

Implement the 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 core. Client-side interfaces and operation of the core are described in Using the Client-Side GMII Datapath. See the following information for additional details: using the core in conjunction with the TEMAC core in Interfacing to Other Cores.

Synthesize your Design and Create a Bitstream

Synthesize your entire design using the desired synthesis tool.

IMPORTANT: *Care must be taken to constrain the design correctly; the constraints provided with the core should be used as the basis for your own. See the constraint chapters in the Vivado Design Suite chapters as appropriate.*

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 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.

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 might not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx tools to place and route the design.

Recognize Timing Critical Signals

The constraints provided with the example design for the core identifies the critical signals and the timing constraints that should be applied. See Constraining the Core (Vivado design tools) for more information.

Make Only Allowed Modifications

The core should not be modified. Modifications can have adverse effects on system timing and protocol compliance. Supported user configurations of the core can only be made by selecting the options from within the IP catalog when the core is generated. See Customizing and Generating the Core for Vivado Design Suite.

Shared Logic

Up to version 13.0 of the core, the RTL hierarchy for the core was fixed. This resulted in some difficulty because shareable clocking and reset logic needed to be extracted from the core example design for use with a single instance, or multiple instances of the core.

Shared logic is a feature that provides a more flexible architecture that works both as a standalone core and as a part of a larger design with one or more core instances. This minimizes the amount of HDL modifications required, but at the same time retains the flexibility to address more uses of the core.

The new level of hierarchy is called `<component_name>_support`. Figure 3-1 and Figure 3-2 show two hierarchies where the shared logic block is contained either in the core or in the example design. In these figures, <component_name> is the name of the generated core. The difference between the two hierarchies is the boundary of the core. It is controlled using the **Shared Logic** option in the Vivado IDE (see Figure 4-7).

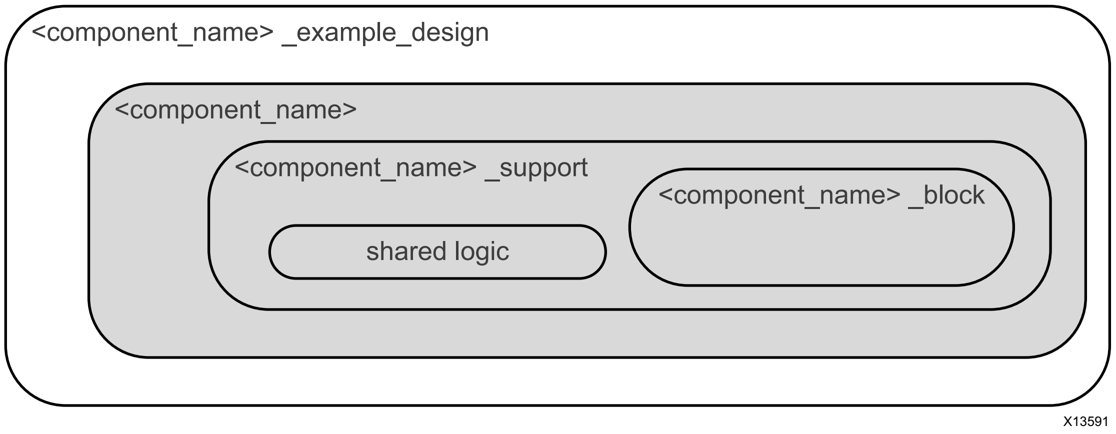


Figure 3-1: **Shared Logic Included in Core**

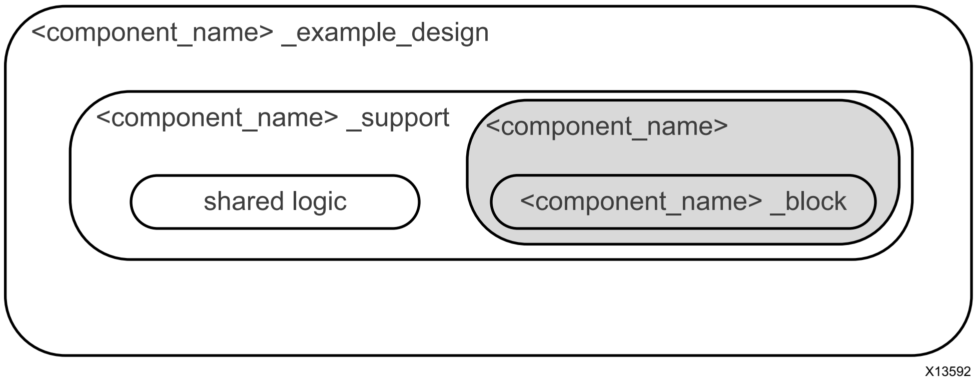


Figure 3-2: **Shared Logic Included in Example Design**

Clocking

For clocking frequencies for the Vivado Design Suite, see Clock Frequencies.

For clocking information on the client interface in SGMII mode, see Clock Generation.

For clocking information on the PHY interface, see the following:

- For TBI Clocking, see The Ten-Bit Interface.
- For 1000BASE-X, see 1000BASE-X with Transceivers.
- For SGMII and Dynamic Switching, see SGMII/Dynamic Switching with Transceivers.
- For System Synchronous SGMII over LVDS, see SGMII LVDS Clocking Logic.

Resets

Due to the number of clock domains in this IP core, the reset structure is not simple and involves several separate reset regions, with the number of regions dependent on the core configuration.

Reset Structure with Transceiver

Figure 3-3 shows the most common reset structure for the core connected to the serial or LVDS transceiver. The grayed out region indicates the logic that is activated under certain conditions based on the core configuration.

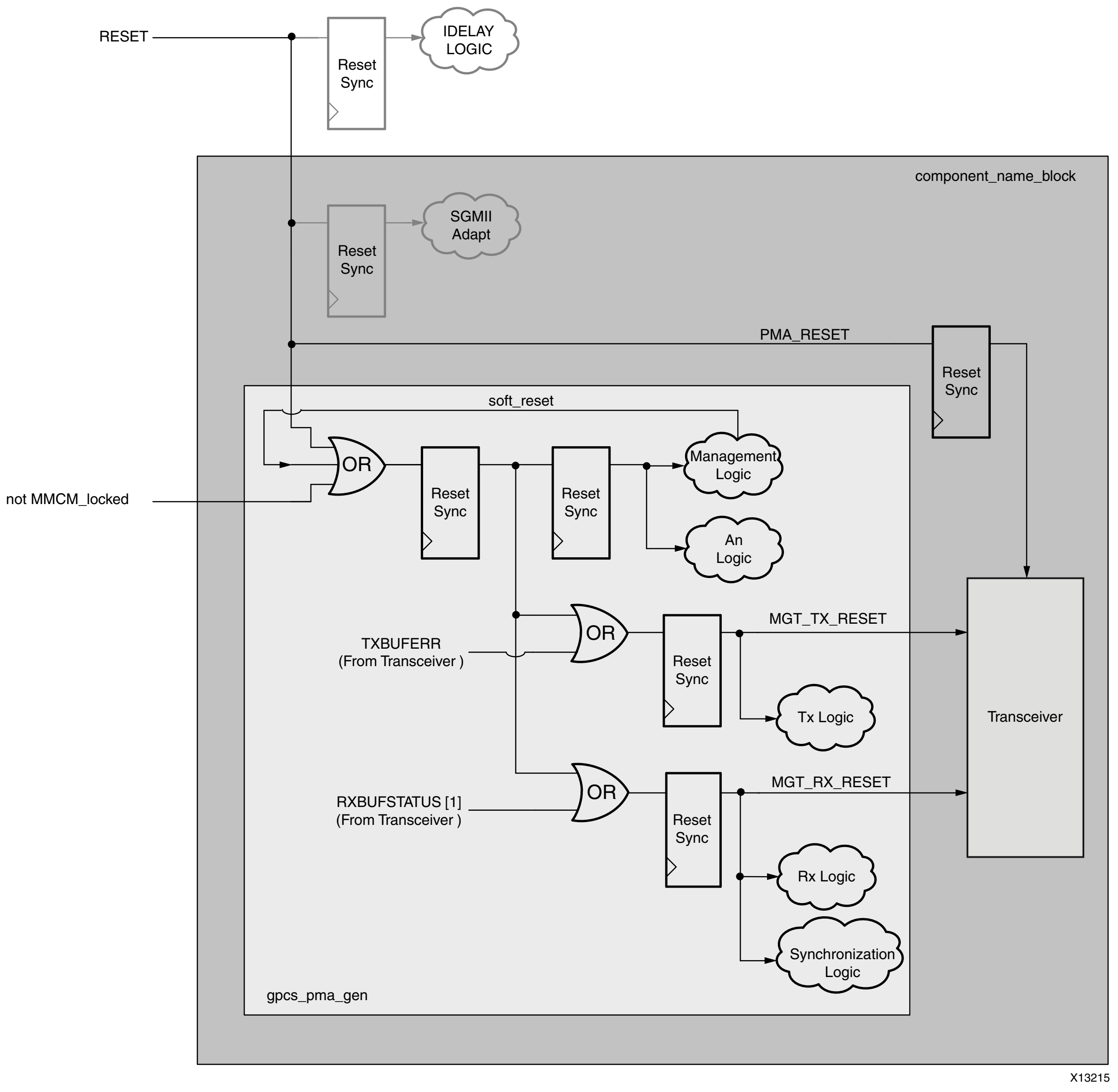


Figure 3-3: **Reset Structure for Core with Transceiver**

Reset Structure with TBI

Figure 3-4 shows the most common reset structure for the core with TBI. The grayed out region indicates the logic that is activated under certain conditions based on the core configuration.

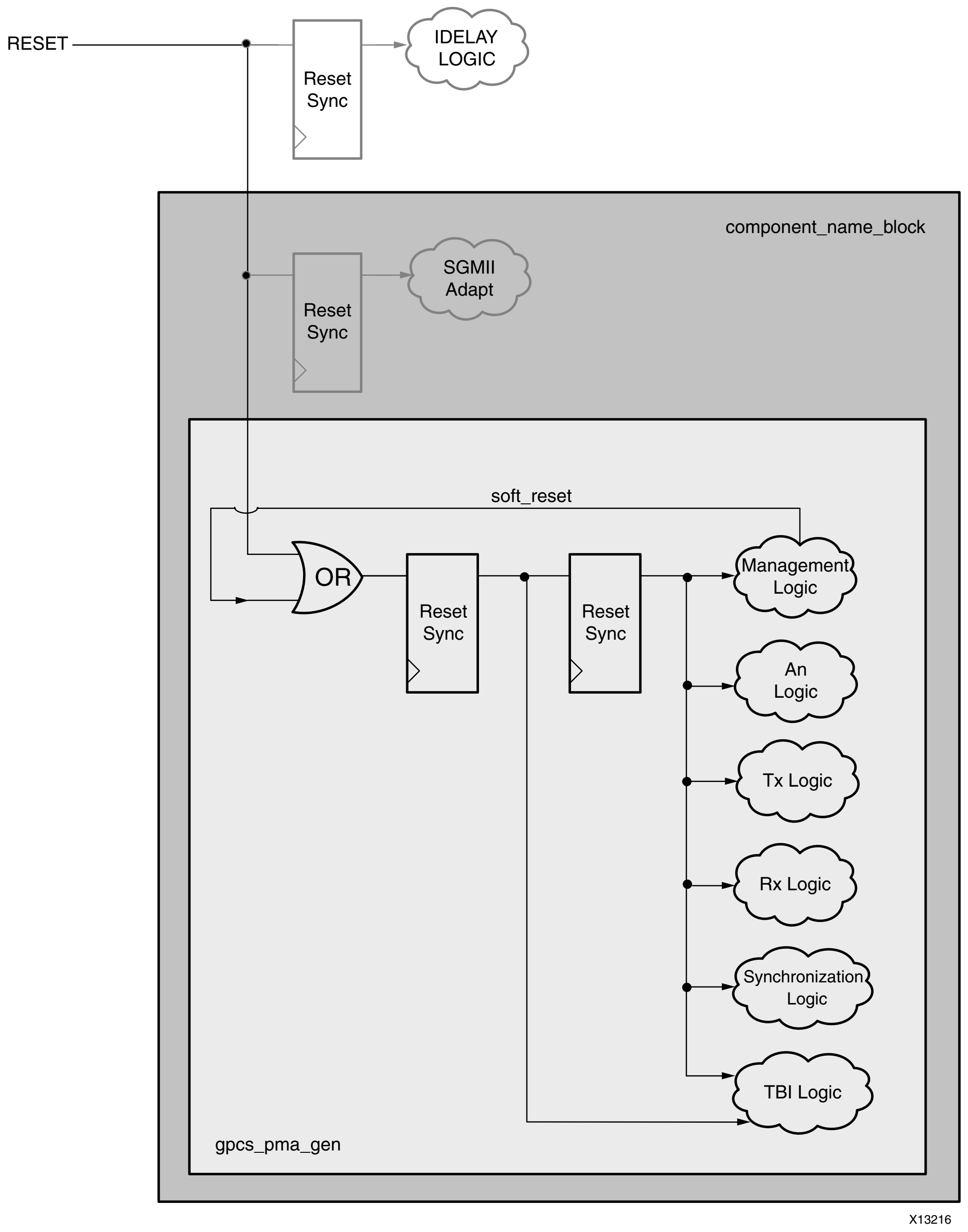


Figure 3-4: **Reset Structure for Core with TBI**

1000BASE-X with Transceivers

This section provides general guidelines for creating 1000BASE-X designs for device-specific transceivers.

For information on the example design see 1000BASE-X with Transceiver Example Design.

Transceiver Logic

The core is designed to integrate with 7 series and Zynq®-7000 FPGA transceivers. Figure 3-5 shows the transceiver connections for Virtex®-7, Kintex®-7 and Zynq-7000 devices. Virtex-7 devices support both the GTX and GTH transceivers; Kintex-7 and Zynq-7000 devices support the GTX transceiver. For Artix®-7devices, the transceiver is connected as shown in Figure 3-6; the supported transceiver is a GTP transceiver.

Figure 3-5 shows the connections and logic required between the core and the transceiver—the signal names and logic in the figure precisely match those delivered with the example design when a 7 series FPGA transceiver is used.

The 125 MHz differential reference clock is routed directly to the 7 series FPGA transceiver. The transceiver is configured to output a version of this clock (62.5 MHz) on the `txoutclk` port; this is then routed to a MMCM. From the MMCM, the `clkout1` port (62.5 MHz) is placed onto global clock routing and is input back into the transceiver on the user interface clock ports `rxusrclk`, `rxusrclk2`, `txusrclk`, and `txusrclk2`. The `clkout0` port (125 MHz) of the MMCM is placed onto global clock routing and can be used as the 125 MHz clock source for all core logic. The clocking logic is included in a separate module `<component_name>_clocking` which is instantiated in the `<component_name>_support` module.

The two wrapper files immediately around the transceiver pair, `gtwizard` and `gtwizard_gt` (Figure 3-5), are generated from the 7 series FPGA Transceiver wizard. These files apply all the gigabit Ethernet attributes. Consequently, these files can be regenerated by customers. See Regeneration of 7 Series/Zynq-7000 Transceiver Files for more information.

A 500 ns wait time for reset is generated with respect to the system clock input in the `<component_name>_gtwizard_init.v[hd]` module. The STABLE_CLOCK_PERIOD attribute in this file has to be set to the period of the system clock.

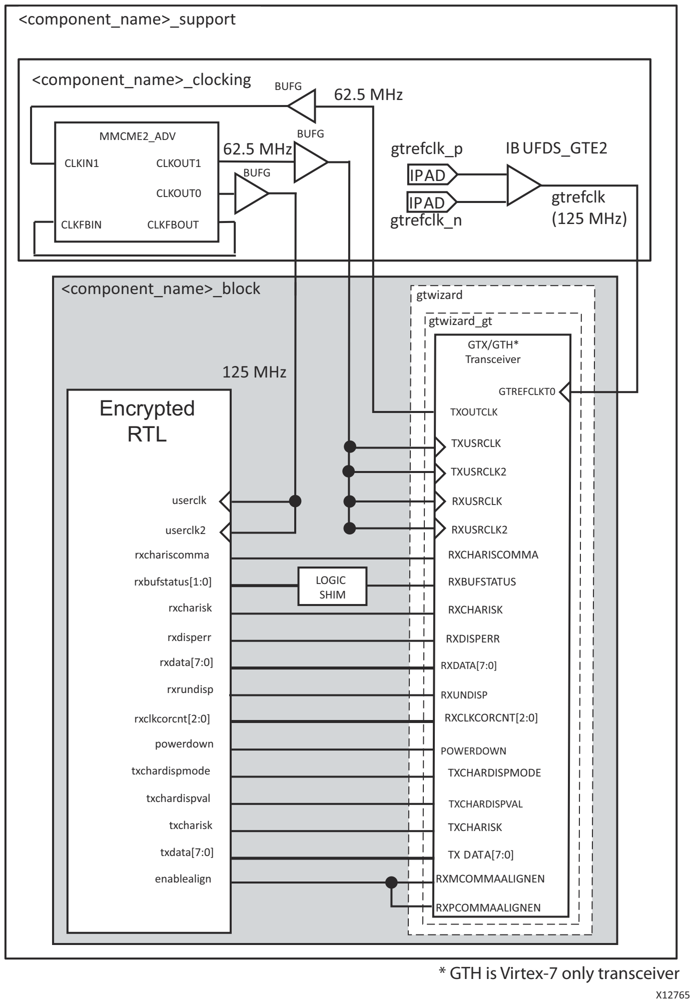


Figure 3-5: **1000BASE-X Transceiver Connections (Virtex-7, Kintex-7, Zynq-7000)**

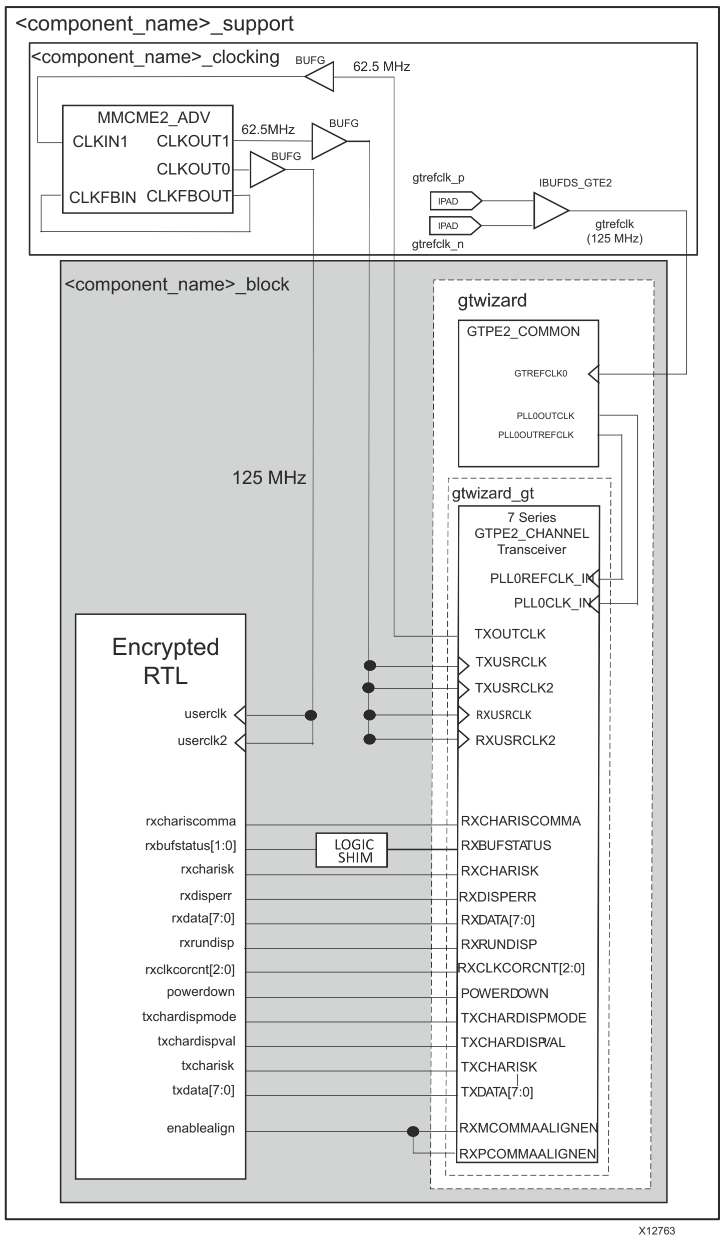


Figure 3-6: **1000BASE-X Transceiver Connections (Artix-7)**

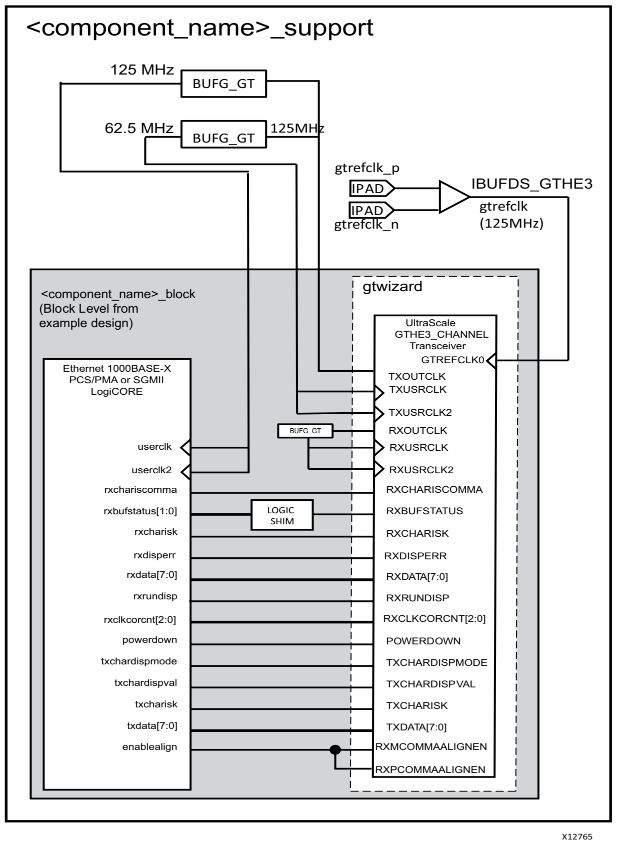


Figure 3-7: **1000BASE-X Transceiver Connections (UltraScale Architecture)**

Clock Sharing Across Multiple Cores with Transceivers

One instance of the core is generated with the **Include Shared Logic in Core** option. This instance contains all the clocking logic that can be shared. The remaining instances can be generated using the **Include Shared Logic in Example Design** option.

Figure 3-8 shows sharing clock resources across two instantiations of the core for Virtex-7, Kintex-7 and Zynq-7000 device transceivers; Figure 3-9 shows the connections for Artix-7 devices. Table 3-1 shows example connections when connecting an instance generated with **Include Shared Logic in Core** to an instance generated using **Include Shared Logic in Example Design**.

Additional cores can be added by continuing to instantiate extra block level modules. To provide the FPGA logic clocks for all core instances, select a `txoutclk` port from any transceiver and route this to a single MMCM. The `clkout0` (125 MHz) and `clkout1` (62.5 MHz) outputs from this MMCM, placed onto global clock routing using BUFGs, can be shared across all core instances and transceivers as shown.

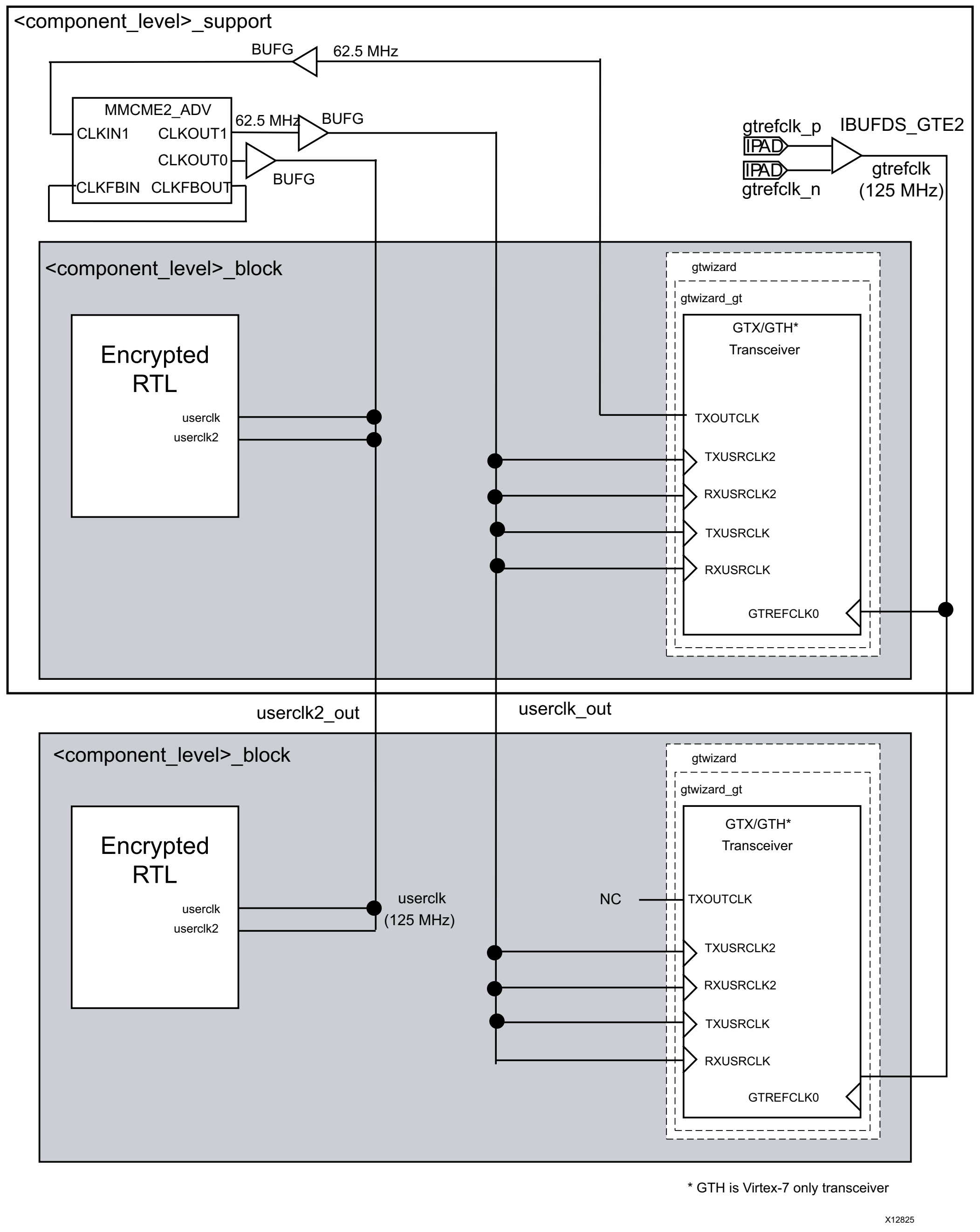


Figure 3-8: **Clock Management-Multiple Core Instances (VIrtex-7, Kintex-7, Zynq-7000)**

For UltraScale architecture devices the MMCM is not required because the `txoutclk` generated is 125 MHz and can be used to generate the 62.5 MHz (`txuserclk`) clock using the BUFG_GT. The same `txoutclk` can be used by other instances of the core by using the BUFG_GT as shown in Figure 3-17.

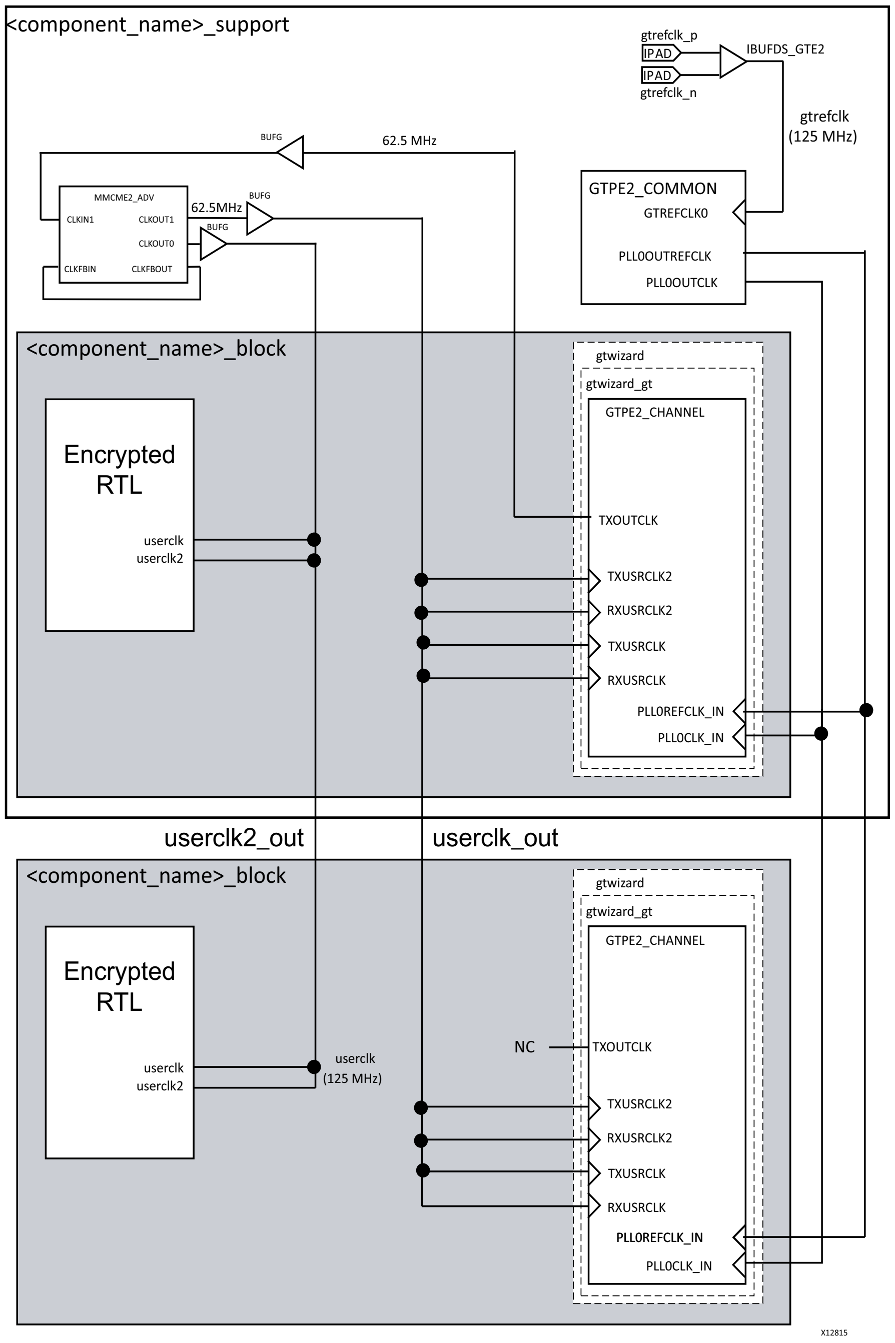


Figure 3-9: **Clock Management-Multiple Core Instances (Artix-7)**

Table 3-1: **Shared Signals Connectivity**

From Instance Using Shared Logic in Core	To Instance Using Shared Logic in Example Design	Design Considerations
gtrefclk_out	gtrefclk	GTREFCLK can be shared up one quad and down one quad
userclk_out	userclk	This can be shared only when TXOUTCLK(and hence gtrefclk) of both instances are synchronous; otherwise this should be connected to the TXOUTCLK port of the same instance. This clock frequency is 62.5 MHz.
userclk2_out	userclk2	This can be shared only when TXOUTCLK (and hence gtrefclk) of both the instances are synchronous; otherwise this should be connected to TXOUTCLK multiplied by two of the same instance. This clock frequency is 125 MHz.
rxuserclk_out	rxuserclk	This can be shared only when RXOUTCLK (recovered clock) of both the channels are synchronous; otherwise this should be connected to the RXOUTCLK port of the same instance. This clock frequency is 62.5 MHz.
rxuserclk2_out	rxuserclk2	This can be shared only when RXOUTCLK (recovered clock) of both the channels are synchronous. Otherwise this should be connected to the RXOUTCLK port of the same instance. This clock frequency is 62.5 MHz.
pma_reset_out	pma_reset	
mmcm_locked_out	mmcm_locked	Provided both the instances use the same MMCM.
Transceiver Specific Signals		
GTH/GTX		
gt0_qplloutclk_out	gt0_qplloutclk_in	Common block output connections for cores in same Quad(1)
gt0_qplloutrefclklost_out	gt0_qplloutrefclklost_in	
GTP		
gt0_pll0outclk_out	gt0_pll0outclk_in	Common block output connections for cores in same Quad(1)
gt0_pll0outrefclk_out	gt0_pll0outrefclk_in	
gt0_pll1outclk_out	gt0_pll1outclk_in	
gt0_pll1outrefclk_out	gt0_pll1outrefclk_in	
gt0_pll0lock_out	gt0_pll0lock_in	
gt0_pll0refclklost_out	gt0_pll0refclklost_in	

1. If instances are using more than one QUAD then an extra common block can be instantiated for each extra quad and connected to each core in that quad.

Transceiver Files

Transceiver Wrapper

This device-specific transceiver wrapper is instantiated from the block-level HDL file of the example design and is described in the following files:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/transceiver/<component_name>_transceiver.v[hd]
```

This file instances output source files from the transceiver wizard (used with Gigabit Ethernet 1000BASE-X attributes).

Zynq-7000 and 7 Series Device Transceiver Wizard Files

For Zynq-7000 and 7 series devices, the transceiver wrapper file directly instantiates device-specific transceiver wrapper files created from the serial transceiver wizard. These files tie off (or leave unconnected) unused I/O for the transceiver and apply the 1000BASE-X attributes. The files can be edited/tailored by re-running the wizard (see Regeneration of 7 Series/Zynq-7000 Transceiver Files) and swapping these files.

Support Level

The following files describe the support level for the core. The files can be found in `/synth` directory if shared logic in core is selected or `/example_design/support` if shared logic in the example design is selected.

```
/synth/<component_name>_support.v[hd] or
/example_design/support/<component_name>_support.v[hd]
```

The <component_name>_support module instantiates idelayctrl, clocking and reset modules.

```
/synth/<component_name>_idelayctrl.v[hd] or
/example_design/support/<component_name>_idelayctrl.v[hd]

/synth/<component_name>_clocking.v[hd] or
/example_design/support/<component_name>_clocking.v[hd]

/synth/<component_name>_resets.v[hd] or
/example_design/support/<component_name>_resets.v[hd]
```

Block Level

The block level is designed so that it can be instantiated directly into your design and performs the following functions:

- Instantiates the core level HDL
- Instantiates shared logic if **Shared Logic in the Core** is selected (see Shared Logic for more information)
- Connects the physical-side interface of the core to a device-specific transceiver

SGMII/Dynamic Switching with Transceivers

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

For information about the SGMII example design see SGMII/Dynamic Switching Using a Transceiver Example Design.

Receive Elastic Buffer

This section describes the two receive elastic buffer implementations; one implementation uses the buffer present in the device-specific transceivers, and the other uses a larger buffer, implemented in FPGA logic. If the latter option is selected, the buffer in the device-specific transceiver is bypassed.

The receive elastic buffer if present (see Receive Elastic Buffer) is described in the following files:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/transceiver/<component_name>_rx_elastic_buffer.v[hd]
```

If the transceiver buffer is bypassed, the buffer implemented in the FPGA logic is instantiated from within the transceiver wrapper. This alternative receive elastic buffer uses a single block RAM to create a buffer twice as large as the one present in the device-specific transceiver, and is able to cope with larger frame sizes before clock tolerances accumulate and result in an emptying or filling of the buffer.

Selecting the Buffer Implementation from the Vivado Integrated Design Environment

The Vivado Integrated Design Environment (IDE) provides two SGMII capability options:

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

The first option, 10/100/1000 Mb/s (clock tolerance compliant with Ethernet specification) is the default and provides the implementation using the receive elastic buffer in FPGA logic. This alternative receive elastic buffer uses a single block RAM to create a buffer twice as large as the one present in the device-specific transceiver, thus taking 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 Mb/s (restricted tolerance for clocks) or 100/1000 Mb/s, uses the receive 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 Mb/s operation. However, there are logical implementations where this can be reliable and has the benefit of lower logic utilization.

Requirement for the Receive Elastic Buffer

Figure 3-10 shows 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 100 ppm. In Figure 3-10, the clock source to the PHY is slightly faster than the clock source to the FPGA. For this reason, during frame reception, the receive 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 receive 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).

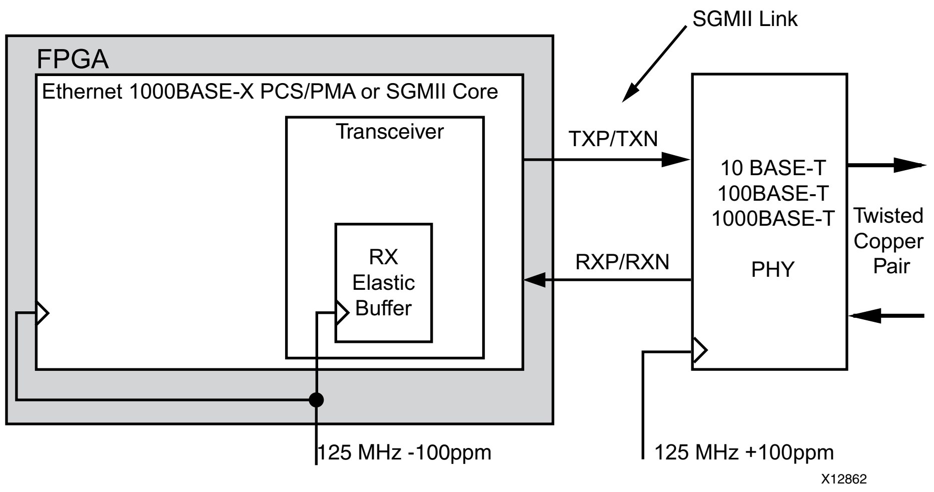


Figure 3-10: **SGMII Implementation using Separate Clock Sources**

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 is a single byte of difference for every 5000 bytes of received frame data, which causes the receive elastic buffer to either fill or empty by an occupancy of one.

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

- At 1 Gb/s operation, this translates into 1522 clock cycles.

- At 100 Mb/s operation, this translates into 15220 clock cycles (as each byte is repeated 10 times).
- At 10 Mb/s operation, this translates into 152200 clock cycles (as each byte is repeated 100 times).

Considering the 10 Mb/s case, you need 152200/5000 = 31 FIFO entries in the elastic buffer above and below the half way point to guarantee that the buffer does not under or overflow during frame reception. This assumes that frame reception begins when the buffer is exactly half full.

The size of the receive elastic buffer in the device-specific transceivers is 64 entries. However, you 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 D, Receive Elastic Buffer Specifications for more information).

To guarantee reliable SGMII operation at 10 Mb/s (non-jumbo frames), the device-specific transceiver elastic buffer must be bypassed and a larger buffer implemented in the FPGA logic. The FPGA logic 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 D, Receive Elastic Buffer Specifications provides further information about all receive elastic buffer used by the core. Information about the reception of jumbo frames is also provided.

Transceiver Receive Elastic Buffer

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

- 10 Mb/s operation is not required. Both 1 Gb/s and 100 Mb/s operation can be guaranteed.
- When the clocks are closely related (see Closely Related Clock Sources).

If there is any doubt, select the FPGA logic receive elastic buffer implementation.

Closely Related Clock Sources

Case 1

Figure 3-11 shows 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.

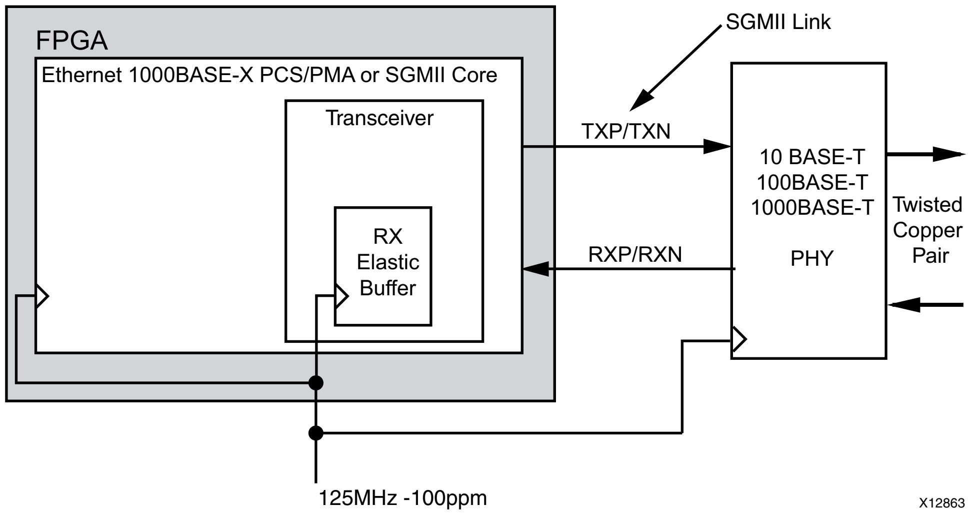


Figure 3-11: **SGMII Implementation using Shared Clock Sources**

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

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

Closely Related Clock Sources

Case 2

Consider again the case shown in Figure 3-10 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 receive 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.

Logic Using the Transceiver Receive Elastic Buffer

When the device-specific transceiver receive 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 1000BASE-X with Transceivers.

Transceiver Logic with the FPGA Logic Receive Elastic Buffer

The example design delivered with the core is shown in Figure 5-2. The block level is designed so to be instantiated directly into customer designs and connects the physical-side interface of the core to an UltraScale™ architecture, Virtex-7, Kintex-7, Artix-7 or Zynq-7000 device transceiver through the FPGA logic receive elastic buffer

Note: The optional transceiver Control and Status ports are not shown here. These ports have been brought up to the <component_name> module level

Transceiver Logic

The core is designed to integrate with 7 series and Zynq-7000 FPGA transceiver. The connections and logic required between the core and transceiver are shown in Figure 3-12 for Virtex-7. Kintex-7 and Zynq-7000 devices; the connections for Artix-7 devices are shown in Figure 3-13; the signal names and logic in the figure precisely match those delivered with the example design when a transceiver is used.

The 125 MHz differential reference clock is routed directly to the transceiver. The transceiver is configured to output 62.5 MHz clock on the `txoutclk` port; this is then routed to an MMCM through a BUFG (global clock routing. From the MMCM, the `clkout0` port (125 MHz) is placed onto global clock routing and can be used as the 125 MHz clock source for all core logic.

From the MMCM, the `clkout1` port (62.5 MHz) is placed onto global clock routing and is input back into the transceiver on the user interface clock port `txusrclk` and `txusrclk2`. The clocking logic is included in a separate module `<component_name>_clocking` which is instantiated in the `<component_name>_support` module.

It can be seen from Figure 3-12/Figure 3-13 that the receive elastic buffer is implemented in the FPGA logic between the transceiver and the core; this replaces the receive elastic buffer in the transceiver.

This alternative receive elastic buffer uses a single block RAM to create a buffer twice as large as the one present in the 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 Mb/s where each frame size is effectively 100 times larger than the same frame would be at 1 Gb/s because each byte is repeated 100 times (see Using the Client-Side GMII for the SGMII Standard).

With this FPGA logic receive elastic buffer implementation, data is clocked out of the transceiver synchronously to `rxoutclk`. This clock can be placed on a BUFMR followed by a BUFR (Virtex-7 only) or a BUFG (Kintex-7, Artix-7and Zynq-7000) and is used to synchronize the transfer of data between the transceiver and the elastic buffer, as shown in Figure 3-12/Figure 3-13.

The two wrapper files around the GTX/GTH transceiver, `gtwizard_gt` and `gtwizard` are generated from the 7 series FPGA transceiver wizard. These files apply all the gigabit

Ethernet attributes. Consequently, these files can be regenerated by customers. See Regeneration of 7 Series/Zynq-7000 Transceiver Files for more information.

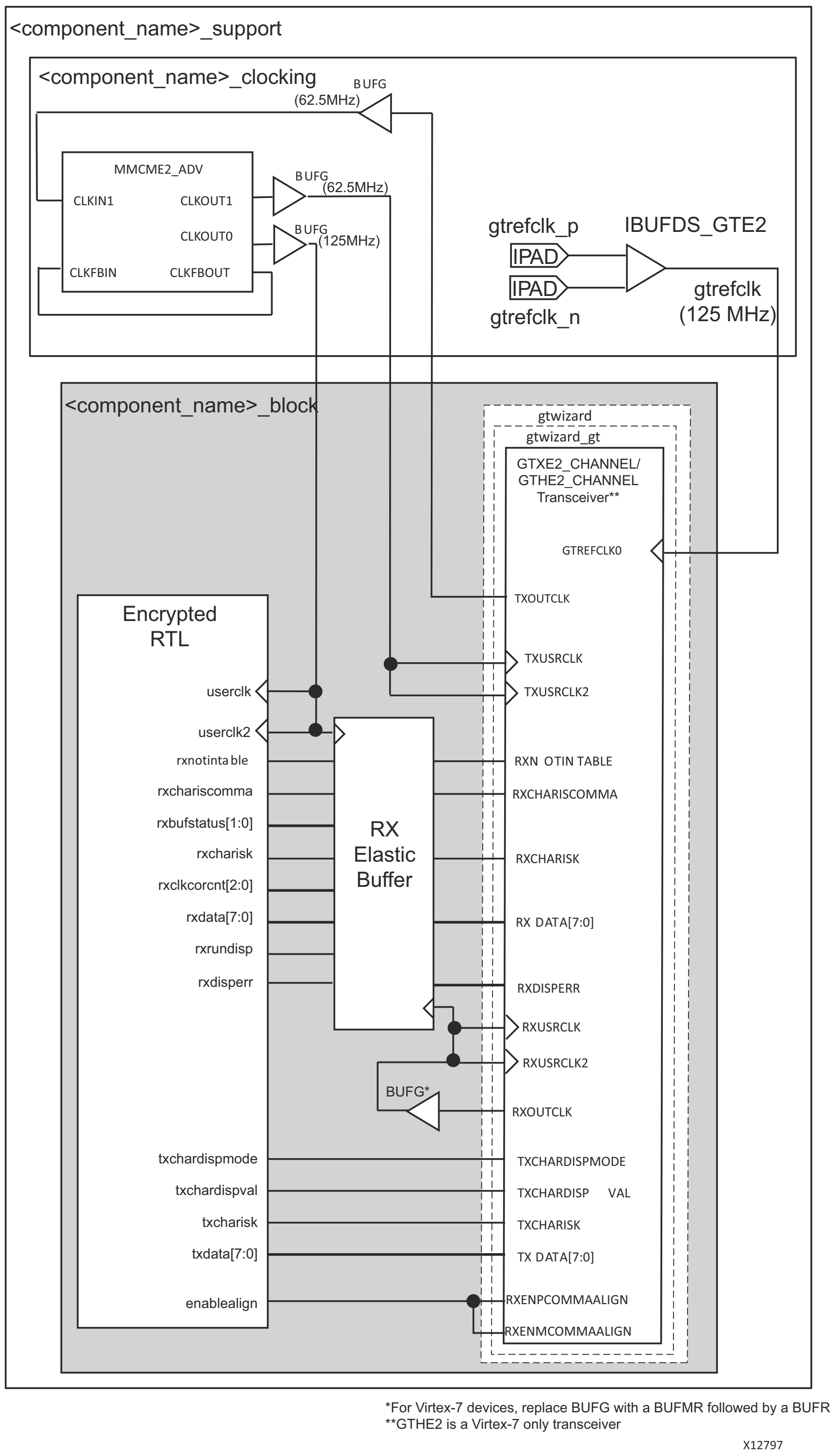


Figure 3-12: **SGMII Transceiver Connection s (VIrtex-7, Kintex-7, Zynq-7000)**

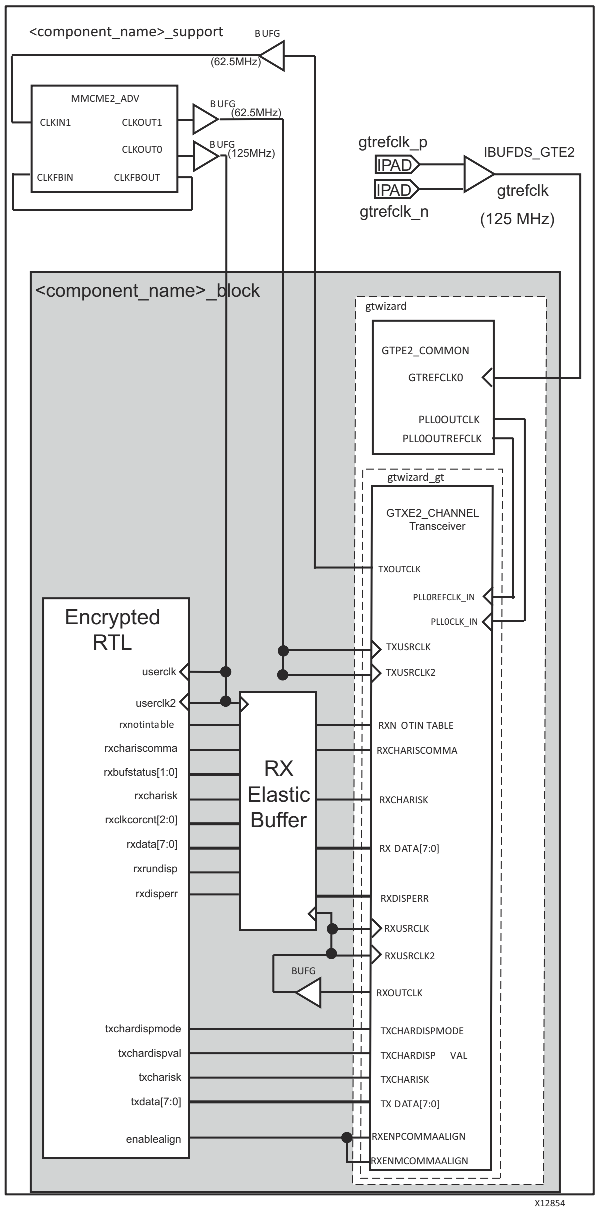


Figure 3-13: **SGMII Transceiver Connections (Artix-7)**

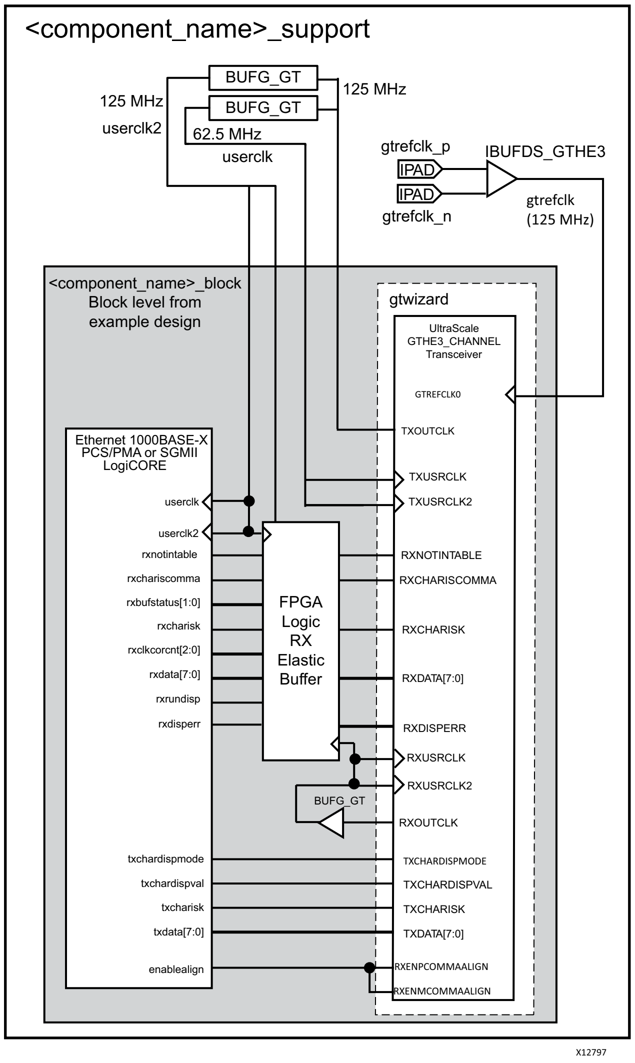


Figure 3-14: **SGMII Transceiver Connections (UltraScale Architecture)**

Clock Sharing Across Multiple Cores with Transceivers and FPGA Logic Elastic Buffer

One instance of the core is generated with the **Include Shared Logic in Core** option. This instance contains all the clocking logic that can be shared. The remaining instances can be generated using the **Include Shared Logic in Example Design** option.

The clocking logic for `rxoutclk` can only be shared if it is known that the transceiver and core pairs across pcs-pma instances are synchronous. In this case the receive clock outputs of clocking module can be used.

Figure 3-15 shows sharing clock resources across two instantiations of the core for Virtex-7, Kintex-7 and Zynq-7000 device transceivers; Figure 3-16 shows the connections for Artix-7 devices. Table 3-1 shows example connections when connecting an instance generated with **Include Shared Logic in Core** to an instance generated using **Include Shared Logic in**

Example Design. Additional cores can be added by continuing to instantiate extra block level modules and sharing the `gtrefclk_p` and `gtrefclk_n` differential clock pairs. See the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 7] or the *7 Series FPGAs GTP Transceivers User Guide* (UG482) [Ref 8]for more information.

To provide the FPGA logic clocks for all core instances, select a `txoutclk` port from any transceiver and route this to a single MMCM. The `clkout0` (125 MHz) and `clkout1` (62.5 MHz) outputs from this MMCM, placed onto global clock routing using BUFGs, can be shared across all core instances and transceivers as shown.

Each transceiver and core pair instantiated has its own independent clock domains synchronous to `rxoutclk`. These are placed on BUFMR followed by regional clock routing using a BUFR and cannot normally be shared across multiple GTX/GTH transceivers.

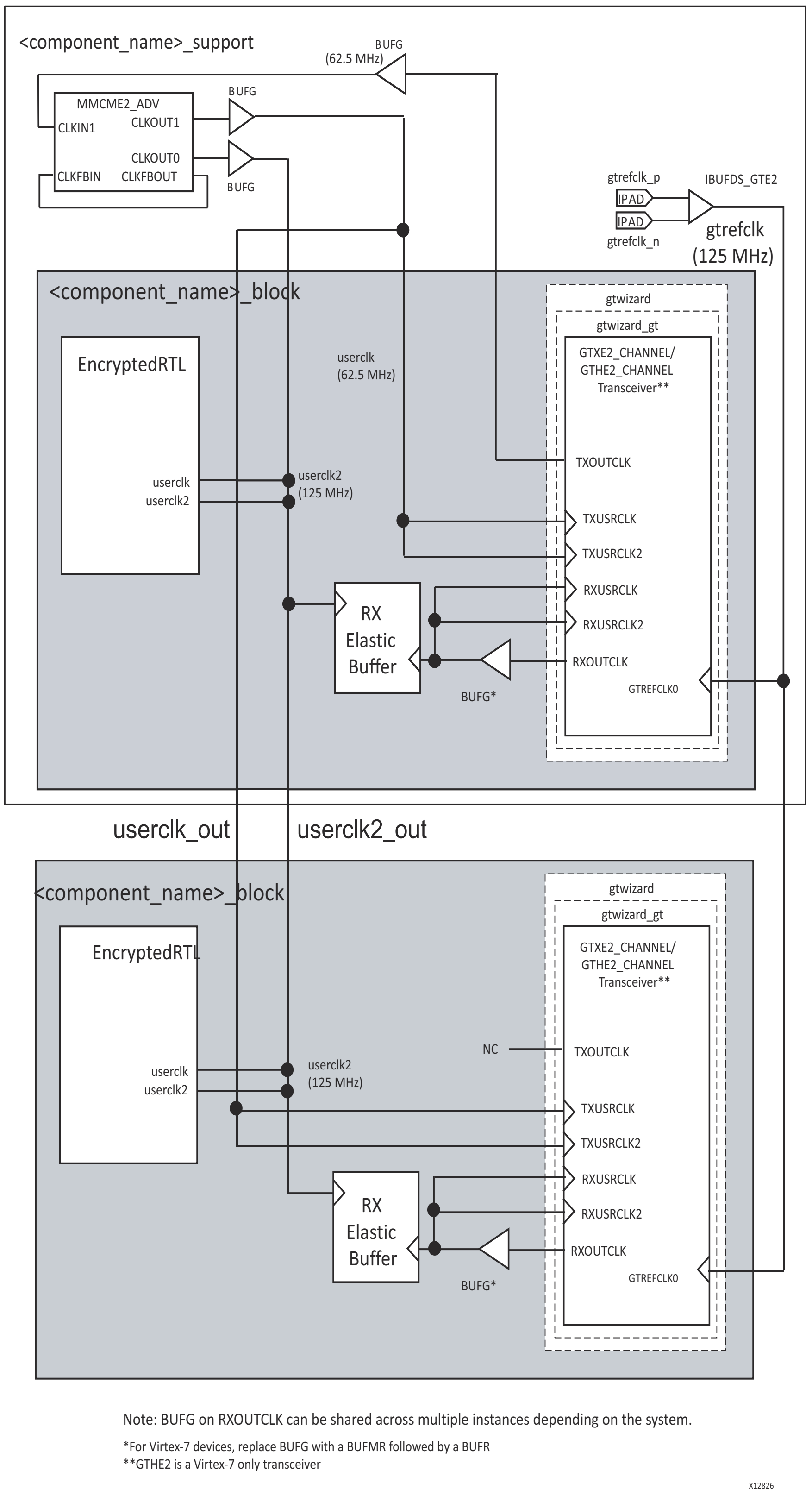
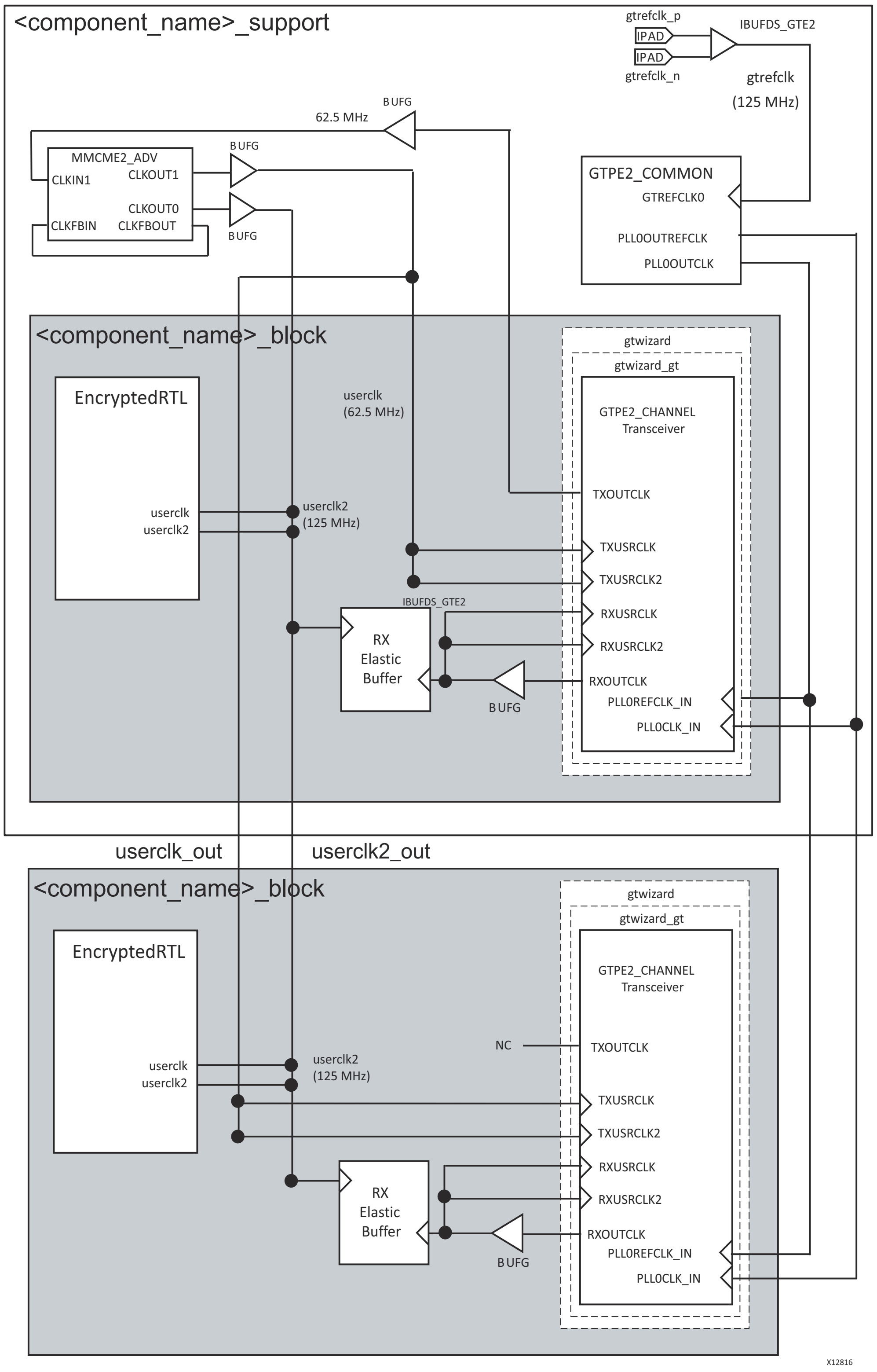


Figure 3-15: **Clock Management with Multiple Core Instances for SGMII (VIrtex-7, Kintex-7, Zynq-7000)**



Note: BUFG on RXOUTCLK can be shared across multiple instances depending on the system.

Figure 3-16: **Clock Management with Multiple Core Instances for SGMII (Artix-7)**

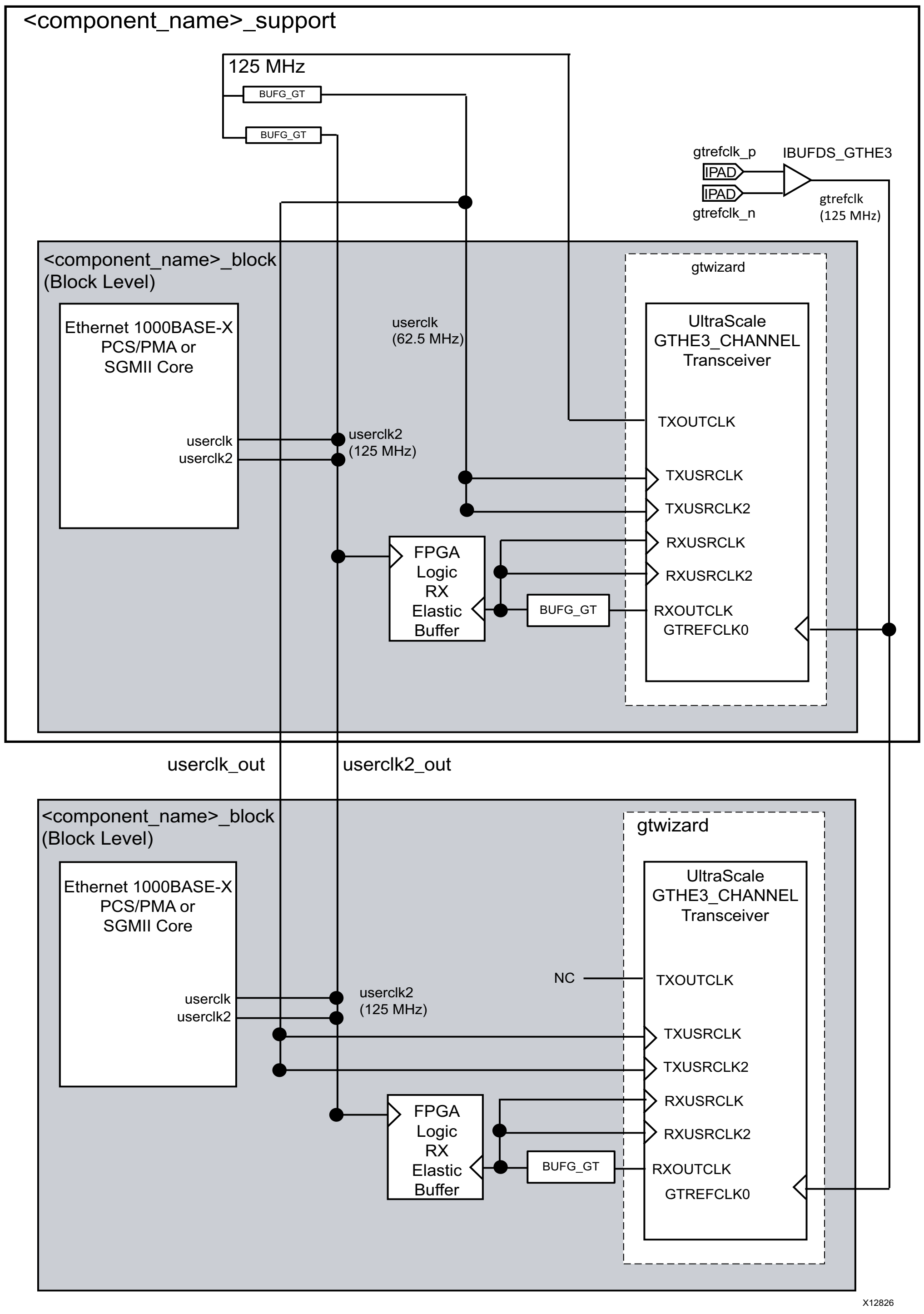


Figure 3-17: **Clock Management with Multiple Core Instances for SGMII (UltraScale)**

Transceiver Files

Transceiver Wrapper

This device-specific transceiver wrapper is instantiated from the block-level HDL file of the example design and is described in the following files:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/transceiver/<component_name>_transceiver.v[hd]
```

This file instances output source files from the transceiver wizard (used with Gigabit Ethernet 1000BASE-X attributes).

Zynq-7000 and 7 Series Device Transceiver Wizard Files

For Zynq-7000, Virtex-7, Kintex-7, and Artix-7 devices, the transceiver wrapper file directly instantiates device-specific transceiver wrapper files created from the serial transceiver wizard. These files tie off (or leave unconnected) unused I/O for the transceiver, and apply the 1000BASE-X attributes. The files can be edited/tailored by re-running the wizard (see Regeneration of 7 Series/Zynq-7000 Transceiver Files) and swapping these files.

Support Level

The following files describe the block level for the core. The files can be found in `/synth` directory if shared logic in core is selected or `/example_design/support` if shared logic in the example design is selected.

```
/synth/<component_name>_support.v[hd] or
/example_design/support/<component_name>_support.v[hd]
```

<component_name>_support module instantiates idelayctrl, clocking and reset modules.

```
/synth/<component_name>_idelayctrl.v[hd] or
/example_design/support/<component_name>_idelayctrl.v[hd]

/synth/<component_name>_clocking.v[hd] or
/example_design/support/<component_name>_clocking.v[hd]

/synth/<component_name>_resets.v[hd] or
/example_design/support/<component_name>_resets.v[hd]
```

Block Level

The block level is designed so that it can be instantiated directly into your design and performs the following functions:

- Instantiates the core level HDL
- Instantiates shared logic if shared logic in the core is selected (see Shared Logic for more information)

- Connects the core physical-side interface to a device-specific transceiver
- An SGMII adaptation module containing:
  - The clock management logic required to enable the SGMII example design to operate at 10 Mb/s, 100 Mb/s, and 1 Gb/s.
  - GMII logic for both transmitter and receiver paths; the GMII style 8-bit interface is run at 125 MHz for 1 Gb/s operation; 12.5 MHz for 100 Mb/s operation; 1.25 MHz for 10 Mb/s operation.

SGMII Adaptation Module

The SGMII Adaptation Module is described in the following files:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
<component_name>/synth/sgmii_adapt/
```

```
<component_name>sgmii_adapt.v[hd]
<component_name>clk_gen.v[hd]
<component_name>johnson_cntr.v[hd]
<component_name>tx_rate_adapt.v[hd]
<component_name>rx_rate_adapt.v[hd]
```

The GMII of the core always operates at 125 MHz. The core makes no differentiation between the three speeds of operation; it always effectively operates at 1 Gb/s. However, at 100 Mb/s, every data byte run through the core should be repeated 10 times to achieve the required bit rate; at 10 Mb/s, each data byte run through the core should be repeated 100 times to achieve the required bit rate. Dealing with this repetition of bytes is the function of the SGMII adaptation module and its component blocks.

The SGMII adaptation module and component blocks are described in detail in the Chapter 3, Additional Client-Side SGMII Logic.

SGMII over LVDS

This section provides the guidelines for creating SGMII interfaces using Zynq-7000, 7 series, and UltraScale architecture devices.

7 Series and Zynq-7000 Device LVDS

This section provides guidelines for creating synchronous SGMII designs using Zynq-7000 and 7 series device LVDS. Supported devices are shown in Table 3-2. This mode enables direct connection to external PHY devices without the use of an FPGA transceiver. An example implementation is shown in Figure 3-18.

Table 3-2: **Devices Supporting SGMII over LVDS**

Family	Supported Devices
Zynq-7000	-2 speed grade or faster for XC7Z010/20 devices and -1 speed grade or faster for XC7Z030/45/100 devices
Virtex-7	-2 speed grade or faster for devices with HR Banks or -1 speed grade or faster for devices with HP banks
Kintex-7	-2 speed grade or faster for devices with HR Banks or -1 speed grade or faster for devices with HP banks
Artix-7	-2 speed grade or faster

For information about the SGMII over LVDS example design see SGMII over LVDS Example Design.

A detailed understanding of 7 series FPGA Clocking Resources and SelectIO Resources is useful to understand the core operation. See the *7 Series FPGAs SelectIO Resources User Guide* (UG471) [Ref 2] and *7 Series FPGA Clocking Resources User Guide* (UG472) [Ref 9].

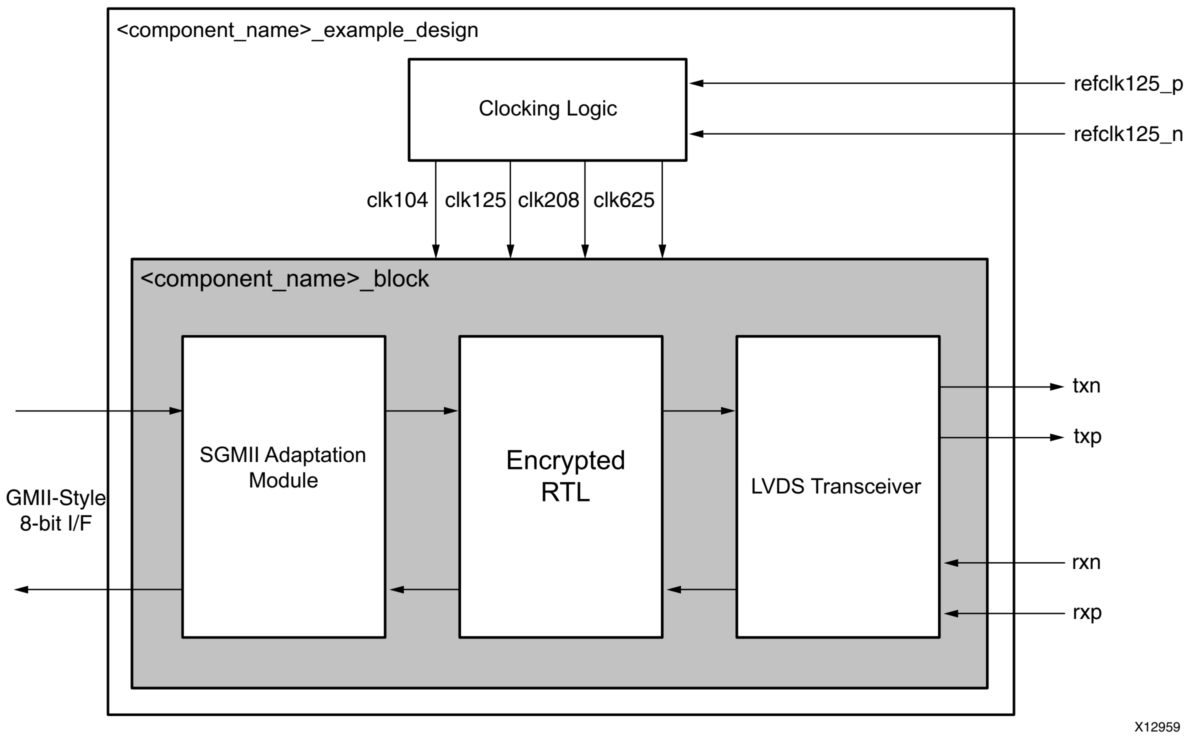


Figure 3-18: **Example Design for SGMII over LVDS Solution (7 Series and Zynq-7000)**

Design Requirements

Timing closure of this interface is challenging; perform the steps described in Layout and Placement.

SGMII Only

The interface implemented using this method supports SGMII between the FPGA and an external PHY device; the interface cannot directly support 1000BASE-X.

SGMII LVDS Clocking Logic

The SGMII LVDS solution is a synchronous implementation where an external clock is provided to the design. In the example design this clock is assumed to be a 125 MHz differential clock.

This 125 MHz differential clock is fed to IBUFDS and the output drives the input of MMCM. The MMCM is used to generate multiple clocks of 208, 625, 125, and 104 MHz. The clocking logic is included in a separate module <component_name>_clocking which is instantiated in the <component_name>_support module.

The 208 MHz clock from MMCM is given to the IDELAYCTRL module which calibrates IDELAY and ODELAY using the user-supplied REFCLK. The system clock of 200 MHz can also be used as a clock input to IDELAYCTRL module instead of the 208 MHz MMCM output clock. See details about IDELAYCTRL in the *7 Series FPGAs SelectIO Resources User Guide* (UG471) [Ref 2].

Typical usage of synchronous LVDS solution involves multiple instances of the LVDS solution with a single clocking block. One instance of the core is generated with the **Include Shared Logic in Core** option. This instance contains all the clocking logic that can be shared. The remaining instances can be generated using the **Include Shared Logic in Example Design** option. Figure 3-19 shows the clocking logic. Table 3-3 lists the clocks in the design and their description.

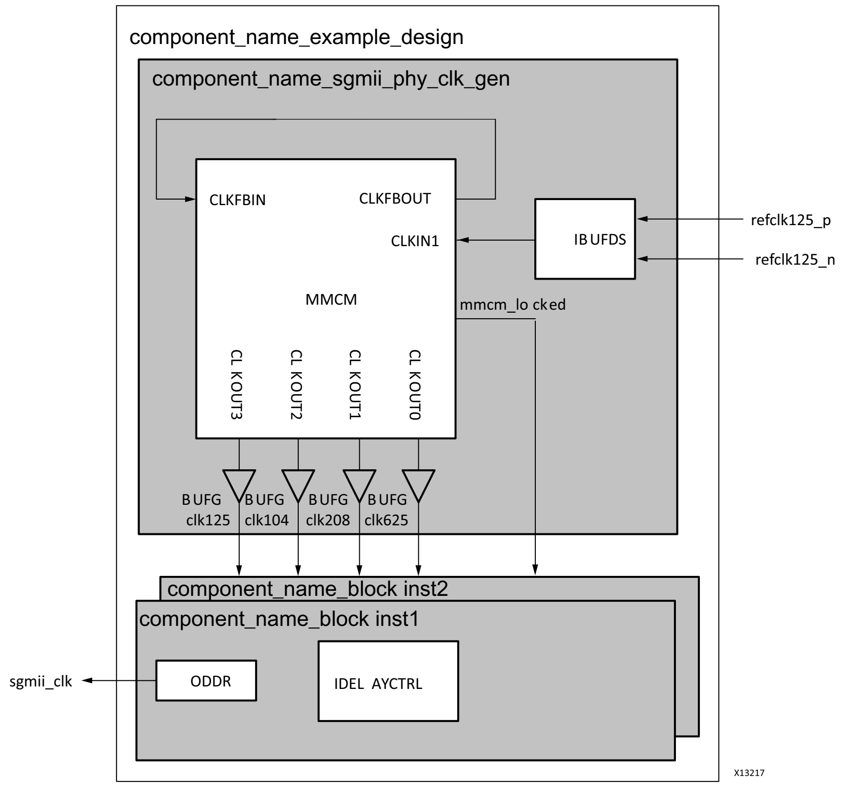


Figure 3-19: **Synchronous LVDS Implementation Clocking Logic Using BUFGs**

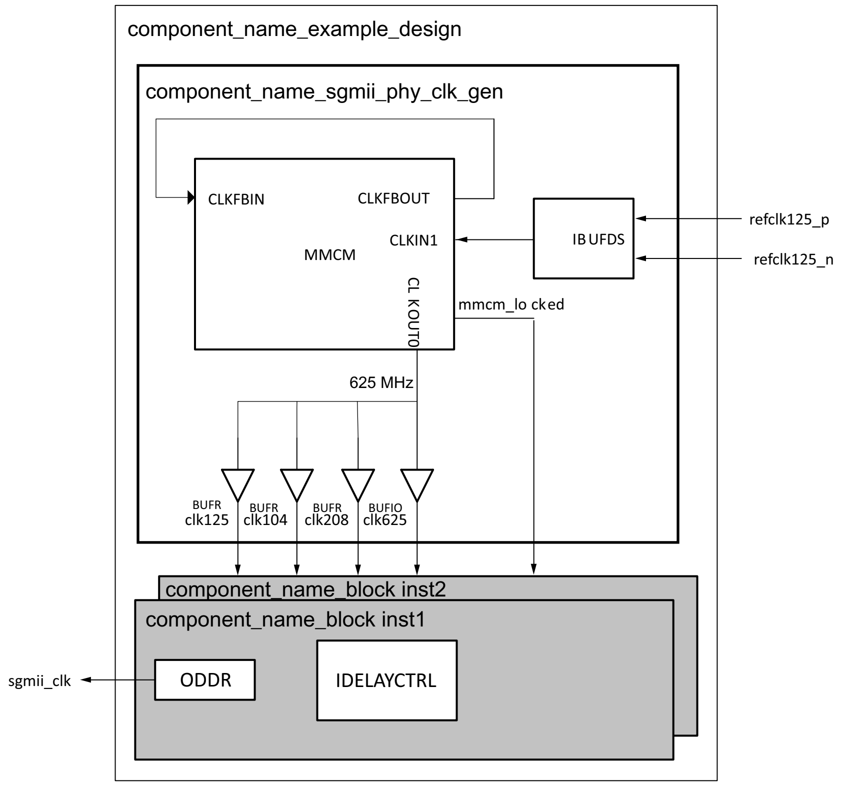


Figure 3-20: **Synchronous LVDS Implementation Clocking Logic Using BUFR/BUFIO**

Important notes relating to Figure 3-19 and Figure 3-20:

- The 125 MHz clock output from IBUFDS that is routed to the `clkin1` pin of the MMCM should enter the FPGA on a global clock pin. This enables the clock signal to be routed to the device MMCM module using dedicated clock routing. The clock source should conform to Ethernet specifications (100 ppm accuracy).
- Figure 3-20 shows that a BUFIO can be used for the 625 MHz clock and a BUFR for the other three MMCM clock outputs. BUFR DIVIDE should be used to divide the MMCM output clock down from 625 MHz for clk125, clk104, and clk208 to keep clock skew low.
- A BUFH can also be used on all four MMCM clock outputs in Figure 3-19. IDELAYCTRL reference clock cannot be fed from the MMCM output with the BUFH if IDELAYs need to span multiple clock regions.
- The OSERDES primitives used by the LVDS transceiver must use a 625 MHz clock source to provide the cleanest possible serial output. This necessitates that the Output Serializer/Deserializer (OSERDES) parallel clock (`clkdiv`) must be provided from a 208 MHz clock buffer that is derived from the same MMCM or PLL. This requirement is used to satisfy the parallel to serial clock phase relationships within the OSERDES primitives.
  See the *7 Series FPGAs SelectIO Resources User Guide* (UG471) [Ref 2] and *7 Series FPGA Clocking Resources User Guide* (UG472) [Ref 9].
- An IDELAY Controller module is provided in the Example Design module for use with the IDELAYs required on the receiver input serial path. This is provided with a 208 MHz clock source from MMCM. The 200 MHz system clock can also be used instead of the 208 MHz clock from MMCM.

Table 3-3 provides a description of the core clocks.

Table 3-3: **Design Clocks**

Clock	Input/Generated/Output	Description
refclk125_p	Differential input clock	Differential clock input to FPGA, synchronous to the incoming serial data.
refclk125_n	Differential input clock	Differential clock input to FPGA, synchronous to the incoming serial data.
clk125_ibuf	125 MHz input clock	Clock derived from incoming differential clock by IBUFGDS.This is the input clock for MMCM.
sgmii_clk	Output Clock to MAC	Clock for client MAC. This clock is derived from sgmii_clk_r and sgmii_clk_f using ODDR primitive.
clk104	Generated by MMCM/BUFR	This clock is used in eye monitor and phy calibration modules to process 12-bit wide data.

Table 3-3: ***(Cont'd)*Design Clocks *(Cont'd)***

Clock	Input/Generated/Output	Description
clk208	Generated by MMCM/BUFR	On transmitter path OSERDES takes 6-bit parallel data at this frequency and converts it to serial data.Similarly on receiver path ISERDES converts serial data into 6 bit parallel data at 208 MHz.Later 6 bit data is converted into 10-bit data through gearbox. Clock also drives the IDELAYCTRL primitive.
clk625	Generated by MMCM	Used by ISERDES and OSERDES modules for input data sampling and parallel to serial conversion respectively.
clk125	Generated by MMCM/BUFR	Used inside the design as main clock. The PCS/PMA core and SGMII adaptation modules work at this clock.
sgmii_clk_r	Generated in SGMII adapter	125 MHz or 12.5 MHz or 1.25 MHz depending on data rate.
sgmii_clk_f	Generated in SGMII adapter	125 MHz or 12.5 MHz or 1.25 MHz depending on data rate.

Layout and Placement

A hands-on approach is required for placing this design. The steps provided are a useful guide, but other knowledge is assumed. To aid with these guidelines in this mode, see the *7 Series FPGAs SelectIO Resources User Guide* (UG471) [Ref 2] and *7 Series FPGA Clocking Resources User Guide* (UG472) [Ref 9]. A working knowledge of the Xilinx Vivado Design Suiteis also useful to locate particular clock buffers and slices.

Use the following guidelines:

- Select an I/O Bank in your chosen device for use with for your transmitter and receiver SGMII ports; see SGMII LVDS Clocking Logic.
- A single IDELAYCTRL is instantiated by the Block Level for use with a single I/O Bank. This primitive needs to be associated with the various IODELAYE2 elements used in that I/O Bank.

The following XDC syntax achieves this in the example design provided for the Kintex-7 device XC7K325T:

```
set_property PACKAGE_PIN AD12 [get_ports refclk125_p]
set_property PACKAGE_PIN AD11 [get_ports refclk125_n]
set_property IOSTANDARD LVDS [get_ports refclk125_n]
set_property IOSTANDARD LVDS [get_ports refclk125_p]

set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property PACKAGE_PIN Y29 [get_ports reset]

set_property PACKAGE_PIN Y23 [get_ports rxp]
set_property PACKAGE_PIN Y24 [get_ports rxn]
set_property IOSTANDARD LVDS_25 [get_ports rxn]
set_property IOSTANDARD LVDS_25 [get_ports rxp]

set_property PACKAGE_PIN L25 [get_ports txp]
set_property PACKAGE_PIN K25 [get_ports txn]
set_property IOSTANDARD LVDS_25 [get_ports txn]
set_property IOSTANDARD LVDS_25 [get_ports txp]
```

LVDS Transceiver for 7 Series and Zynq-7000 Devices

The LVDS transceiver block fully replaces the functionality otherwise provided by a 7 series device transceiver. This is only possible at a serial line rate of 1.25 Gb/s. Figure 3-21 shows a block diagram of the LVDS transceiver for Zynq-7000 and 7 series devices. This is split up into several sub-blocks which are described in further detail in the following sections.

On the transmitter path, data sourced by the core netlist is routed through the 8B/10B Encoder to translate the 8-bit code groups into 10-bit data. The 10-bit data is then passed through the 10B6B Gearbox and the parallel data is then clocked out serially at a line rate of 1.25 Gb/s.

The receiver path has additional complexity. Serial data received at 1.25 Gb/s is routed in parallel to two IODELAYs and ISERDES. Logic is provided to find the correct sampling point in the eye monitor and Phy calibration blocks.

The 6-bit parallel data is fed to the 6B10B gearbox which converts it into 10-bit parallel data. Having recovered parallel data from the serial stream, the Comma Alignment module, next on the receiver path, detects specific 8B/10B bit patterns (commas) and uses these to realign the 10-bit parallel data to contain unique 8B/10B code groups. These code groups are then routed through the 8B/10B Decoder module to obtain the unencoded 8-bit code groups that the core netlist can accept.

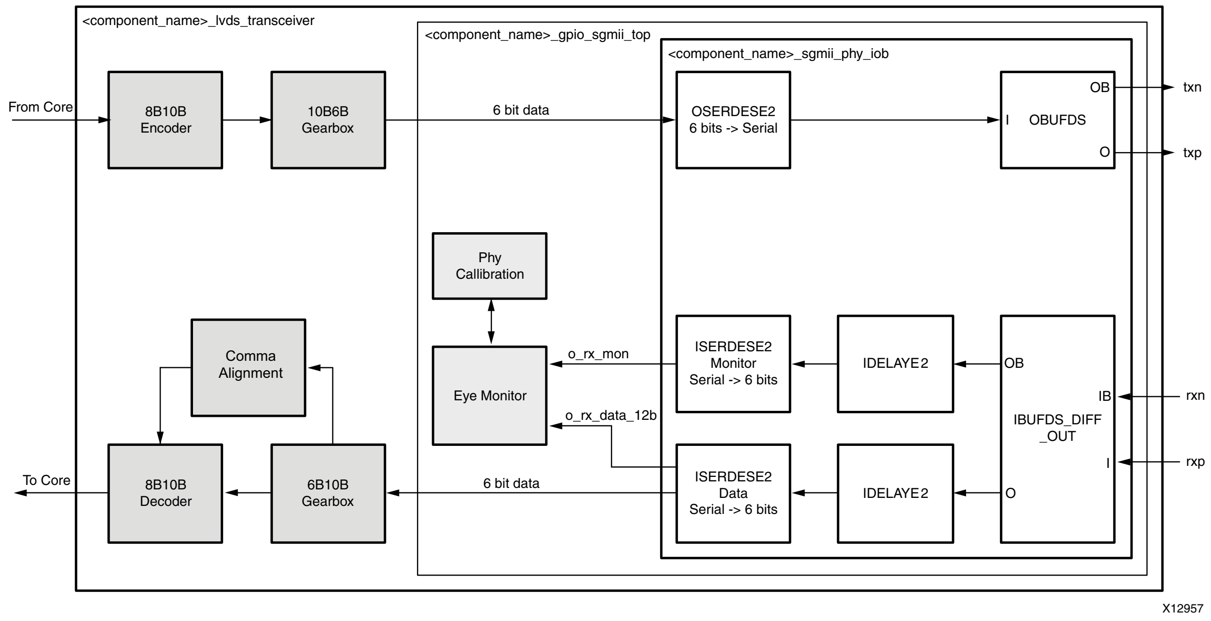


Figure 3-21: **LVDS Transceiver Block Level for 7 Series and Zynq-7000 Devices**

The following files describe the top level of the hierarchal levels of the LVDS transceiver:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/lvds_transceiver/<component_name>_lvds_transceiver.v[hd]
```

8B/10B Encoder

The implemented 8B/10B coding scheme is an industry standard, DC-balanced, byte-oriented transmission code ideally suited for high-speed local area networks and serial data links. As such, the coding scheme is used in several networking standards, including Ethernet. The 8B/10B Encoder block is taken from Xilinx Application Note, *Parameterizable 8B/10B Encoder* (XAPP1122) [Ref 10] provides two possible approaches: a choice of a block RAM-based implementation or a LUT-based implementation. The SGMII LVDS example design uses the LUT-based implementation, but XAPP1122 can be used to swap this for the block RAM-based approach if this better suits device logic resources.

The following files describe the 8B/10B Encoder:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/lvds_transceiver/
    <component_name>_encode_8b10b_pkg.v[hd]
    <component_name>_encode_8b10b_lut_base.v[hd]
```

OSERDES

The OSERDES primitive (actually a MASTER-SLAVE pair of primitives) is used in a standard mode; 6-bit input parallel data synchronous to a 208 MHz global clock buffer source (BUFG) is clocked into the OSERDES. Internally within the OSERDES, the data is serialized and output at a rate of 1.25 Gb/s. The clock source used for the serial data is a 625 MHz clock source using a BUFG global clock buffer at double data rate.

- The 625 MHz BUFG and 208 MHz BUFG clocks for serial and parallel data are both derived from the same MMCM so there is no frequency drift.
- The use of the BUFG global clock buffer for the parallel clock is a requirement of the OSERDES; when using a BUFG clock for serial data, a BUFG clock source, derived from the same MMCM source, must be used for the parallel data to satisfy clock phase alignment constraints within the OSERDES primitives.

Gearbox 10b6b

This module is used to convert 10-bit data at 125 MHz to 6-bit data at 208 MHz. This data is then given to OSERDES for serialization.

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/lvds_transceiver/<component_name>_gearbox_10b_6b.v[hd]
```

IODELAYs and ISERDES

This logic along with eye monitor and phy calibration is used to convert incoming serial data into 6 bit parallel data. See IODELAYs and ISERDES in the *7 Series FPGAs SelectIO Resources User Guide* (UG471) [Ref 2] for more information on these primitives.

Eye Monitor and PHY Calibration

Both these modules have state machines and work in conjunction to find the right sampling point for receive data coming from ISERDES. These modules work on 12-bit wide data at 104 MHz frequency. This data is the 6-bit parallel data (at 208 MHz) sampled at 104 MHz. Eye monitor monitors the N-node IDELAY to determine the margin of current P-node (data) IDELAY tap value.

The following file describes the eye monitor functionality:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/lvds_transceiver/<component_name>_sgmii_eye_monitor.v[hd]
```

PHY calibration module uses the eye monitor block to determine the optimal rx-data IDELAY sampling point. The following file describes the phy calibration functionality:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/
<component_name>/synth/lvds_transceiver/<component_name>_sgmii_phy_calibration.v[hd]
```

Gearbox 6b10b

This module is used to convert 6-bit data recovered from ISERDES at 208 MHz to 10-bit data at 125 MHz to be used by Comma Alignment and 8B/10B Decoder modules. Also it implements bitslip logic based on input from comma alignment module.

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/synth
/lvds_transceiver/<component_name>_gearbox_6b_10b.v[hd]
```

Comma Alignment

Data received by comma alignment block is in parallel form, but the bits of the parallel bus have not been aligned into correct 10-bit word boundaries. By detecting a unique 7-bit serial sequence known as a 'comma' (however the commas can fall across the 10-bit parallel words), the comma alignment logic controls bit shifting of the data so as to provide correct alignment to the data leaving the module.

The bitslip input of the gearbox_6b_10b is driven by the comma alignment module state machine, so the actual bit shift logic is performed by the gearbox_6b_10b. In 8B/10B encoding, both +ve and -ve bit sequences exist for each defined code group. The comma alignment logic is able to detect and control realignment on both +ve and -ve comma versions.

The following files describe the Comma Alignment block:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/lvds_transceiver/<component_name>_sgmii_comma_alignment.v[hd]
```

8B/10B Decoder

The implemented 8B/10B coding scheme is an industry-standard, DC-balanced, byte-oriented transmission code ideally suited for high-speed local area networks and serial data links. As such, the coding scheme is used in several networking standards, including Ethernet. The 8B/10B Decoder block is taken from Xilinx Application Note, *Parameterizable 8B/10B Encoder* (XAPP1122) [Ref 10] provides two possible approaches: a choice of a block RAM-based implementation or a LUT-based implementation.

The SGMII LVDS example design uses the LUT-based implementation, but XAPP1112 can be used to swap this for the block RAM-based approach if this better suits device logic resources.

The following files describe the 8B/10B Decoder:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/lvds_transceiver/
    <component_name>_decode_8b10b_pkg.v[hd]
    <component_name>_decode_8b10b_lut_base.v[hd]
```

GPIO SGMII TOP

This module is a hierarchical top including the eye monitor, phy calibration modules, and the SGMII PHY IOB functionality. See Figure 3-21 for a detailed block diagram for LVDS transceiver.

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/lvds_transceiver/<component_name>_gpio_sgmii_top.v[hd]
```

SGMII PHY IOB

This module is a hierarchical top including the ISERDES, OSERDES, and IDELAY modules. See Figure 3-21 for a detailed block diagram for LVDS transceiver.

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/synth
/lvds_transceiver/<component_name>_sgmii_phy_iob.v[hd]
```

UltraScale Architecture Device LVDS

This section provides general guidelines for creating synchronous SGMII designs using the UltraScale architecture device LVDS. This enables direct connection to external PHY devices without the use of an FPGA transceiver.

To benefit from a detailed understanding of UltraScale architecture clocking resources and SelectIO Resources, see *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 3] and *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 11].

Design Requirements

SGMII Only

The interface implemented using this method supports SGMII between the FPGA and an external PHY device; the interface cannot directly support 1000BASE-X.

Supported Devices

See the performance characteristics of the I/O banks of the UltraScale architecture device in the device data sheet. Any I/O supporting a maximum 1.25 Gb/s should support this feature.

SGMII LVDS Clocking Logic

The SGMII LVDS solution is a synchronous implementation where an external clock is provided to the design. In the example design this clock is assumed to be a 125 MHz differential clock. Clocking logic is similar to 7 series implementation.

The differences in clocking logic compared to 7 series implementation follow:

- `Clk208` and `Clk104` are not required by the UltraScale architecture device implementation of SGMII over LVDS design.
- `Clk312` (312.5 MHz) is required by the UltraScale architecture device implementation of SGMII over LVDS design.

Layout and Placement

A hands-on approach is required for placing this design. The steps provided are a useful guide, but other knowledge is assumed. To aid with these guidelines in this mode, see the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 3] and *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 11].

Use the following guidelines:

- Select an I/O Bank in your chosen device for use with for your transmitter and receiver SGMII ports; see SGMII LVDS Clocking Logic.
- A single IDELAYCTRL is instantiated by the block level for use with a single I/O Bank. This primitive needs to be associated with the various IODELAYE3 elements used in that I/O Bank.

Note: In the example design constraint file (XDC), replace the example LOC placement constraint with the placement constraint relevant to your implementation.

LVDS Transceiver for UltraScale Architecture Devices

The LVDS transceiver block fully replaces the functionality otherwise provided by an UltraScale architecture device transceiver. This is only possible at a serial line rate of 1.25 Gb/s. The LVDS is split up into serveral sub-blocks which are described in further detail in the following sections.

Figure 3-22 shows a block diagram of the LVDS transceiver for UltraScale architecture devices. This is split up into several sub-blocks which are described in further detail in the following sections.

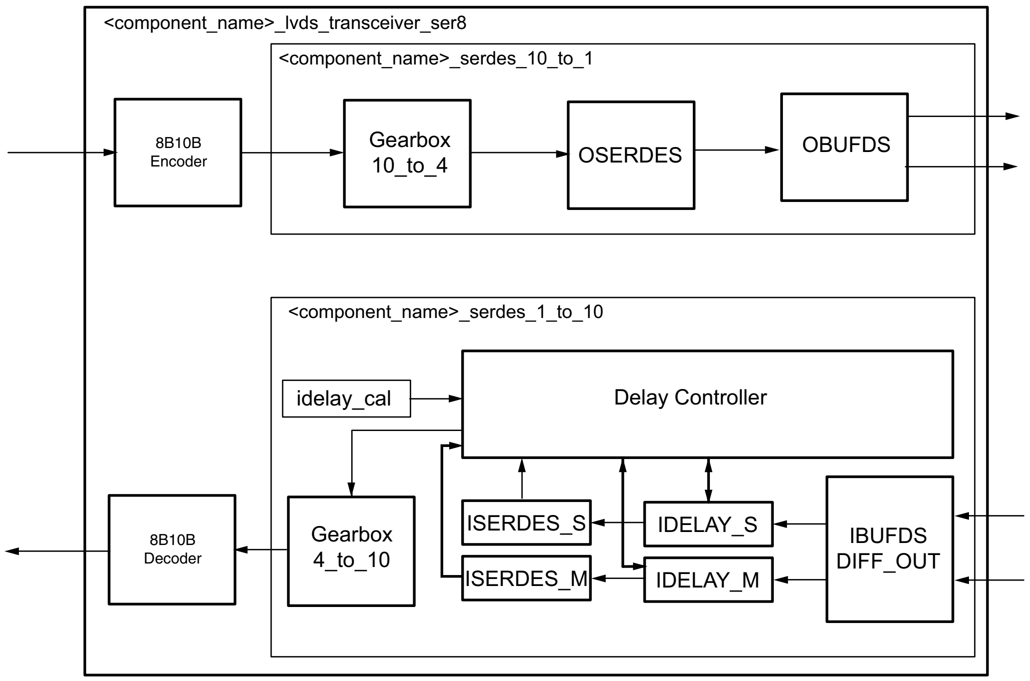


Figure 3-22: **LVDS Transceiver Block Level for UltraScale Architecture Devices**

On the transmitter path, data sourced by the core netlist is routed through the 8B/10B Encoder to translate the 8-bit code groups into 10-bit data. The 10-bit data is then passed through the 10B4B Gearbox and the 4-bit parallel data is then clocked out serially at a line rate of 1.25 Gb/s.

The receiver path has additional complexity. Serial data received at 1.25 Gb/s is routed in parallel to two IODELAYE3 and ISERDESE3. The LVDS transceiver block uses the UltraScale architecture device OSERDESE3, IODELAYE3s and ISERDESE3 elements. See the *UltraScale Architecture SelectIO Resources User Guide* (UG571) [Ref 3] for a description of these elements. Logic is provided to find the correct sampling point in the delay controller block.

Then parallel data is fed to the 4B10B gearbox which converts it into 10-bit parallel data. Having recovered parallel data from the serial stream, comma alignment and detection is done on the parallel data. Receiver uses these to realign the 10-bit parallel data to contain unique 8B/10B code groups. These code groups are then routed through the 8B/10B Decoder module to obtain the unencoded 8-bit code groups that the core netlist can accept.

The following files describe the top level of the hierarchal levels of the LVDS transceiver:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/sgmii_lvds_transceiver/<component_name>_lvds_transceiver_ser8.v
```

Note: Transceiver functionality is implemented only in Verilog except for the 8B/10B encoder and 10B/8B decoder modules which are project setting specific.

Delay Controller

This module controls delays on a per bit basis, It controls the delay values for IDELAYE3s and hence the sampling point for incoming receive data.

The following file describes the delay controller:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/sgmii_lvds_transceiver/<component_name>_delay_controller_wrap.v
```

Serdes_1_to_10_ser8

This module converts 1-bit serial data to 10-bits parallel data. It instantiates the I/O logic cells (IDELAYE3, ISERDES), delay controller and 4-bit to 10-bit gearbox functionality.

The following file describes the serdes 1 to 10 logic:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/sgmii_lvds_transceiver/<component_name>_serdes_1_to_10_ser8.v
```

Serdes_10_to_1_ser8

This module converts 10-bit parallel data to 1-bit serial data. It instantiates the I/O logic cells (ODELAYE3, OSERDES) and 10-bit to 4-bit gearbox functionality.

The following file describes the serdes 10 to 1 logic:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/sgmii_lvds_transceiver/
<component_name>_serdes_10_to_1_ser8.v
```

Gearbox_10_to_4_ser8

Converts 10-bit data clocked at 125 MHz to 4-bits data clocked at 312.5 MHz. This 4-bit parallel data is then presented to OSERDES which converts it into serial stream of 1.25 Gb/s.

The following file describes the gearbox 10 to 4 bit logic:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/sgmii_lvds_transceiver/
<component_name>_gearbox_10_to_4_ser8.v
```

Gearbox_4_to_10_ser8

Converts 4-bit data clocked at 312.5 MHz from ISERDES to 10-bit parallel data clocked at 125 MHz. This data is then presented to the 10b/8b decoder.

The following file describes the gearbox 4 to 10 bit logic:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/sgmii_lvds_transceiver/
<component_name>_gearbox_4_to_10_ser8.v
```

SGMII Adaptation Module

The SGMII Adaptation Module is described in the following files:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/sgmii_adapt/
    <component_name>_sgmii_adapt.v[hd]
    <component_name>_clk_gen.v[hd]
    <component_name>_johnson_cntr.v[hd]
    <component_name>_tx_rate_adapt.v[hd]
    <component_name>_rx_rate_adapt.v[hd]
```

The GMII of the core always operates at 125 MHz. The core does not differentiate between the three speeds of operation; it always effectively operates at 1 Gb/s. However, at 100 Mb/s, every data byte run through the core should be repeated 10 times to achieve the required bit rate; at 10 Mb/s, each data byte run through the core should be repeated 100 times to achieve the required bit rate. Dealing with this repetition of bytes is the function of the SGMII adaptation module and its component blocks. The SGMII adaptation module and component blocks are described in detail in the Additional Client-Side SGMII Logic.

Support Level

The following files describe the support level for the core. The files can be found in `/synth` directory if shared logic in core is selected or `/example_design/support` if shared logic in example design is selected.

```
/synth/<component_name>_support.v[hd] or
/example_design/support/<component_name>_support.v[hd]
```

The <component_name>_support module instantiates idelayctrl, clocking and reset modules.

```
/synth/<component_name>_idelayctrl.v[hd] or
/example_design/support/<component_name>_idelayctrl.v[hd]

/synth/<component_name>_clocking.v[hd] or
/example_design/support/<component_name>_clocking.v[hd]

/synth/<component_name>_resets.v[hd] or
/example_design/support/<component_name>_resets.v[hd]
```

Block Level

The block level connects together all of the components for a single SGMII port. These are:

- A core netlist (introduced in 1000BASE-X PCS/PMA or SGMII Using a Device-Specific Transceiver in Chapter 1).
- The LVDS Transceiver for UltraScale Architecture Devices, connected to the PHY side of the core netlist, to perform the SerDes functionality using the Synchronous LVDS Method. Containing:

    - Functionality for I/O functionality and gearbox modules in transmit and receive path for data width conversion.
    - Functionality to find the right sampling point using eye monitor and phy calibration modules.
- The SGMII Adaptation Module top level, connected to the Ethernet MAC (GMII) side of the core netlist, containing:
    - The clock management logic required to enable the SGMII example design to operate at 10 Mb/s, 100 Mb/s, and 1 Gb/s.
    - GMII logic for both transmitter and receiver paths; the GMII style 8-bit interface is run at 125 MHz for 1 Gb/s operation; 12.5 MHz for 100 Mb/s operation; 1.25 MHz for 10 Mb/s operation.

The Ten-Bit Interface

This section provides general guidelines for creating 1000BASE-X, SGMII or Dynamic Switching designs using the Ten-Bit Interface (TBI).

For information on the ten-bit example design see Chapter 5, Example Design.

Note: Kintex-7 devices support TBI at 3.3V or lower.

Ten-Bit Interface Logic

This section provides an explanation of the TBI physical interface logic in all supported families. This section is common to both 1000BASE-X and SGMII implementations.

Transmitter Logic

Figure 3-23 shows the use of the physical transmitter interface of the core to create an external TBI. 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 are automatically used in the example design.

Figure 3-23 shows that the output transmitter datapath 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.

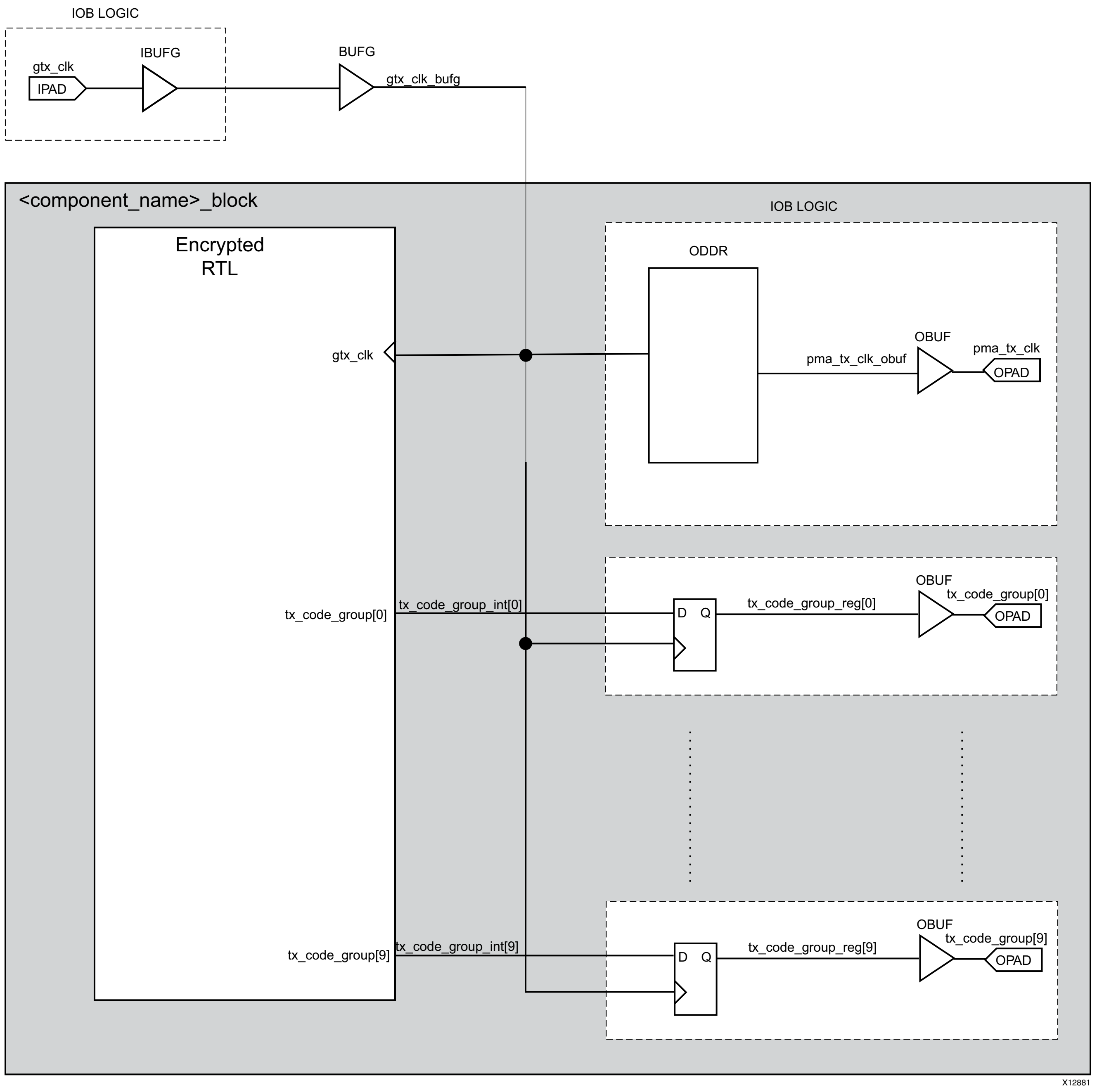


Figure 3-23: **Ten-Bit Interface Transmitter Logic**

Receiver Logic

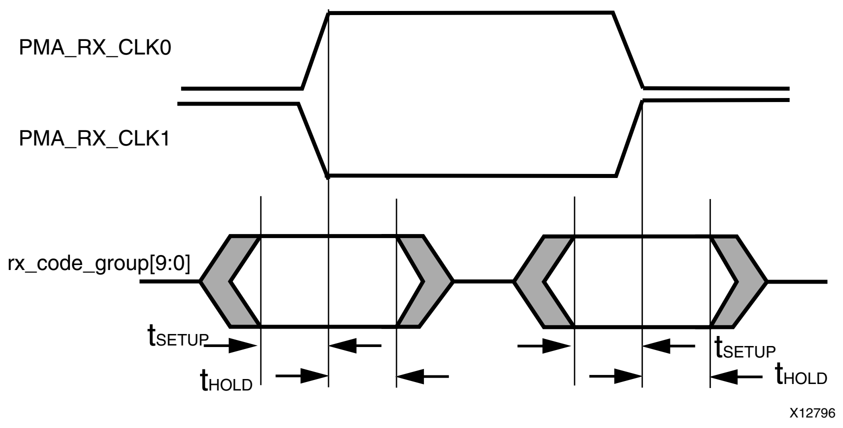


Figure 3-24: **Input TBI timing**

Figure 3-24 shows the input timing for the TBI interface as defined in IEEE802.3-2008 clause 36.

IMPORTANT: *The important point 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 IEEE802.3-2008 specification, there is no exact requirement that `pma_rx_clk0` and `pma_rx_clk1` be exactly 180° 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 that connects to the TBI might well specify that this is the case; that `pma_rx_clk0` and `pma_rx_clk1` are exactly 180° out of phase. If this is the case, the TBI receiver clock logic can 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). Select the method carefully by referring to the data sheet of the external SerDes.

The example design provided with the core only gives one of these methods (which vary on a family-by-family basis). However, the example HDL design can be edited to convert to the alternative method. See the following two methods for a Kintex-7 device.

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

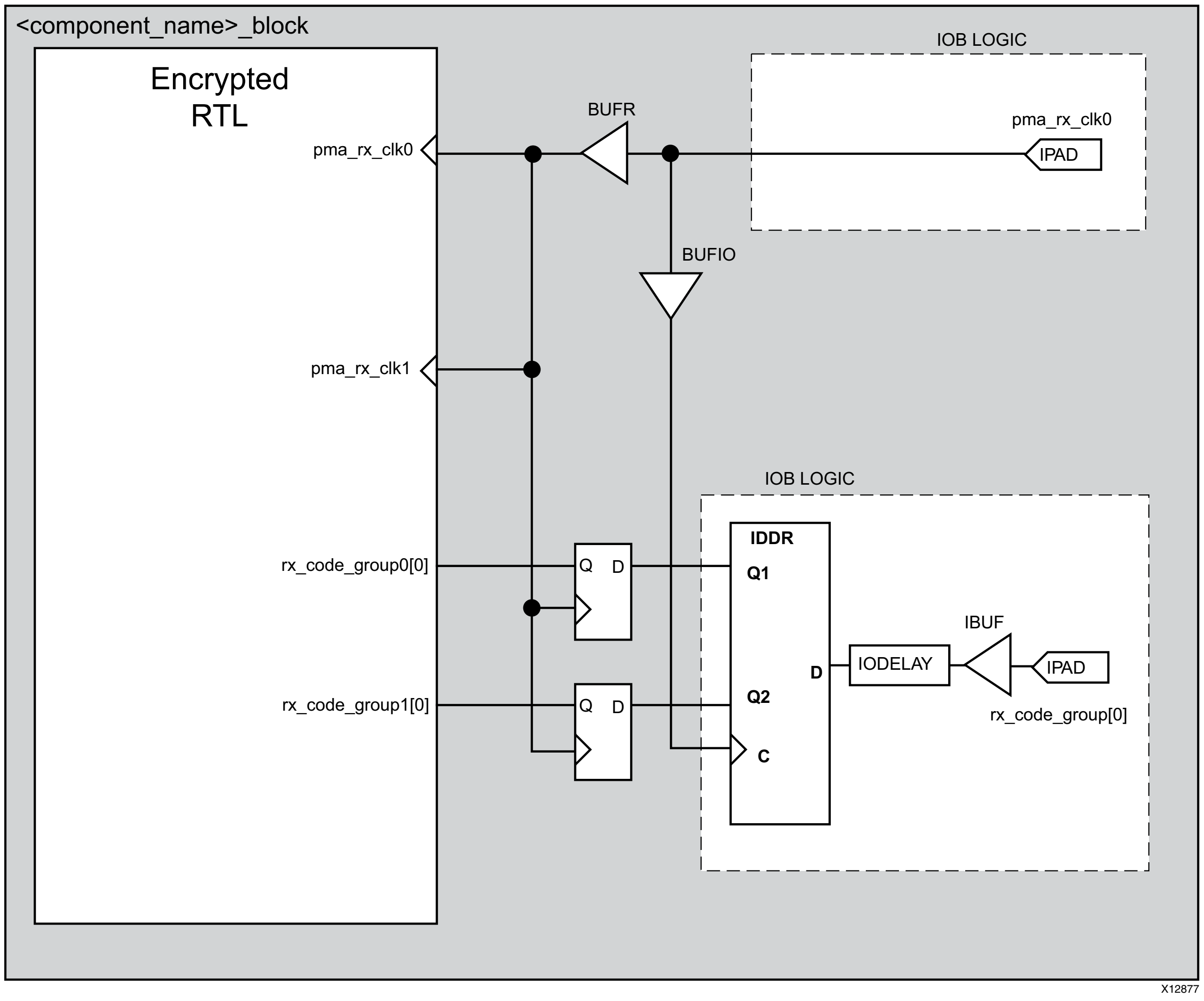


Figure 3-25: **Ten-Bit Interface Receiver Logic - Kintex-7 Devices (Example Design)**

The FPGA logic used by the example design delivered with the core is shown in Figure 3-25. This shows an `IDDR` primitive used with the DDR_CLK_EDGE attribute set to SAME_EDGE. 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 shown.

CAUTION! *This logic relies on pma_rx_clk0 and pma_rx_clk1 being exactly 180° 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 shown in Figure 3-25.

In the Figure 3-25 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. However, 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. See the FPGA user guides for more information.

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 shown.

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 XDC; these can be edited if required.

Method 2: Alternative Using Both pma_rx_clk0 and pma_rx_clk1

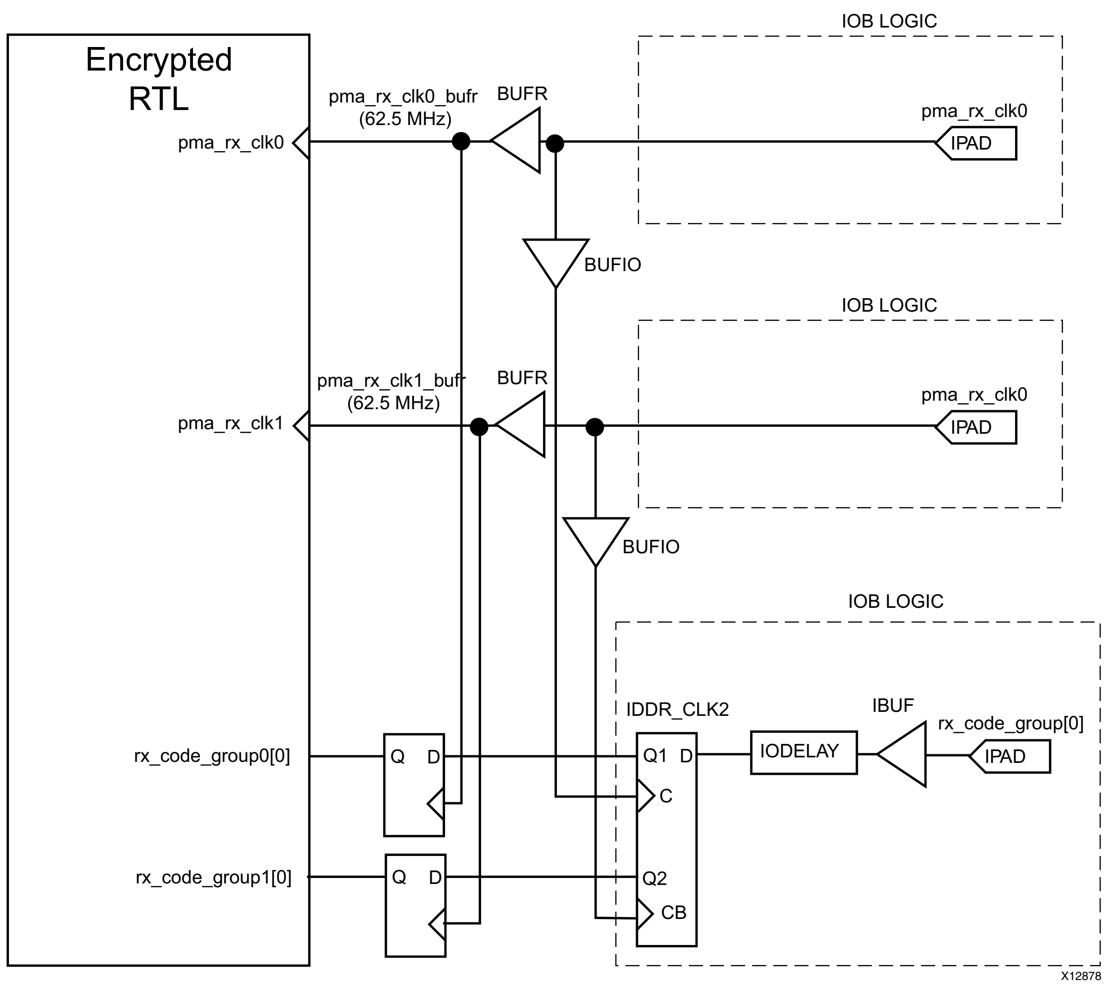


Figure 3-26: **Alternate TBI Receiver Logic - Kintex-7 Devices**

This logic from Method 1 relies on `pma_rx_clk0` and `pma_rx_clk1` being exactly 180° 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 3-26 shows 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 3-25 is duplicated for `pma_rx_clk1`. An IDDR_CLK2 primitive replaces the IDDR primitive; this contains two clock inputs as shown.

Clock Sharing across Multiple Cores with the TBI

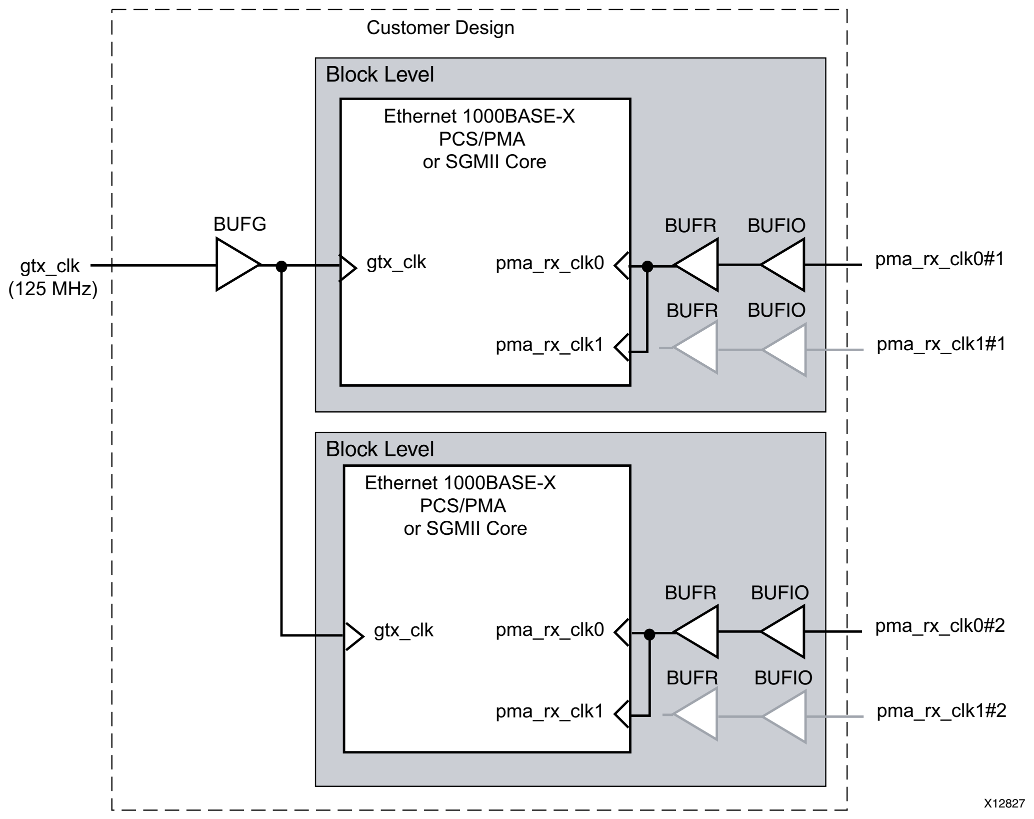


Figure 3-27: **Clock Management, Multiple Core Instances with the TBI**

Figure 3-27 shows sharing clock resources across multiple instantiations of the core when using the TBI. For all implementations, `gtx_clk` can 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 is provided with its own versions of these receiver clocks from its externally connected SerDes.

Figure 3-27 shows 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.

Block Level

The block level is designed so that it can be instantiated directly into customer designs and performs the following functions:

- Instantiates the core level HDL
- Connects the physical-side interface of the core to device IOBs, creating an external TBI TBI, including IOB and DDR registers instances, where required

For SGMII/Dynamic Switching with a TBI the block level also has an SGMII Adaptation Module containing:

  - The clock management logic required to enable the SGMII example design to operate at 10 Mb/s, 100 Mb/s, and 1 Gb/s.
  - GMII logic for both transmitter and receiver paths; the GMII style 8-bit interface is run at 125 MHz for 1 Gb/s operation; 12.5 MHz for 100 Mb/s operation; 1.25 MHz for 10 Mb/s operation.

The block level HDL connects the TBI of the core to external IOBs (the most useful part of the example design) and should be instantiated in all customer designs that use the core.

The file location for the SGMII Adaptation Module is described in SGMII Adaptation Module. The SGMII adaptation module and component blocks are described in detail in Additional Client-Side SGMII Logic.

Using the Client-Side GMII Datapath

This section provides general guidelines for using the client-side GMII of the core. In most applications, the client-side GMII is expected to be used as an internal interface, connecting to either:

- Proprietary customer logic

  This section describes the GMII-styled interface that is present on the netlist of the core. This interface operates identically for both 1000BASE-X and SGMII standards.

  The section then also focuses on additional optional logic (which is provided by the example design delivered with the core when SGMII mode is selected). This logic enhances the internal GMII-styled interface to support 10 Mb/s and 100 Mb/s Ethernet speeds in addition to the nominal 1 Gb/s speed of SGMII.

- The IP catalog core Tri-Mode Ethernet MAC

  The 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. See Interfacing to Other Cores.

- Ethernet MACs (ENET0/ENET1) in the Zynq-7000 AP SoC processor subsystem

  The core can be integrated with ENET0 or ENET1 through the EMIO interface. See Interfacing to Other Cores.

In rare applications, the client-side GMII datapath can be used as a true GMII, to connect externally off-chip across a PCB. The extra logic required to create a true external GMII is detailed in Appendix E, Implementing External GMII.

Using 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 802.3-2008 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 shown in Figure 3-28. 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 treats any value placed on `gmii_txd[7:0]` within the `gmii_tx_en` assertion window as data.

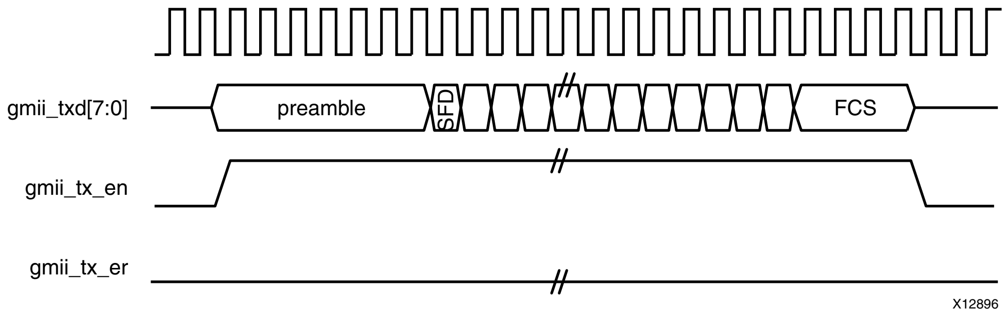


Figure 3-28: **GMII Normal Frame Transmission**

Error Propagation

A corrupted frame transfer is shown in Figure 3-29. An error can 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.

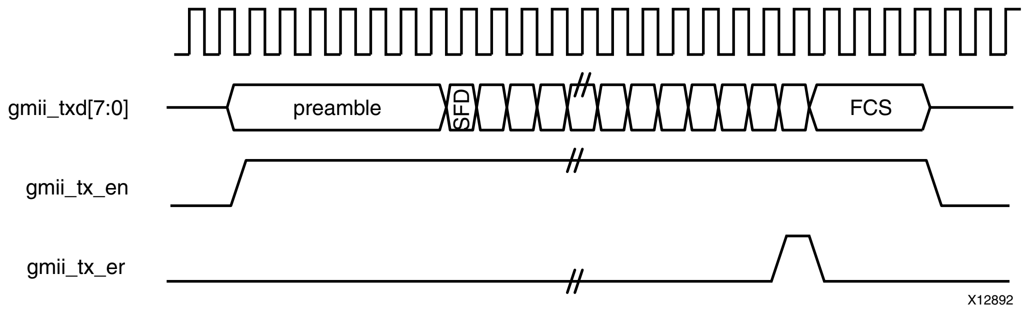


Figure 3-29: **GMII Error Propagation Within a Frame**

GMII Reception

This section includes figures that illustrate GMII reception. In these figures the clock is not labeled. The source of this clock signal 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 shown in Figure 3-30. This shows that Ethernet frame reception is proceeded by a preamble field. The *IEEE 802.3-2008* specification (see clause 35) [Ref 5] 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 is always present in well-formed frames.

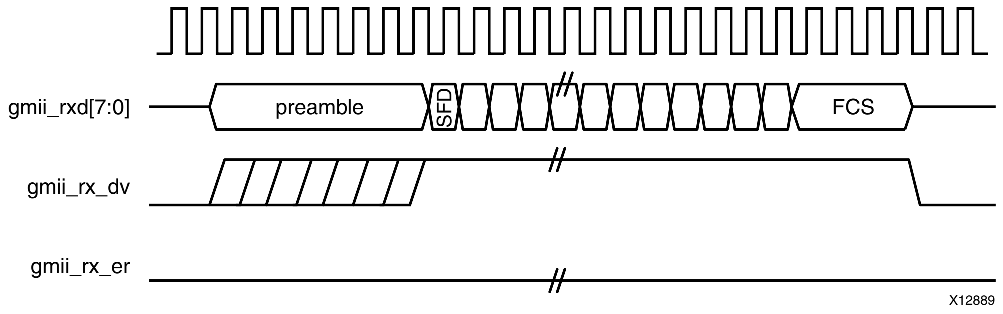


Figure 3-30: **GMII Normal Frame Reception**

Normal Frame Reception with Extension Field

In accordance with the IEEE 802.3-2008, clause 36 [Ref 5], state machines for the 1000BASE-X PCS, `gmii_rx_er` can 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 shown in Figure 3-31. See Appendix C, 1000BASE-X State Machines for more information.

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

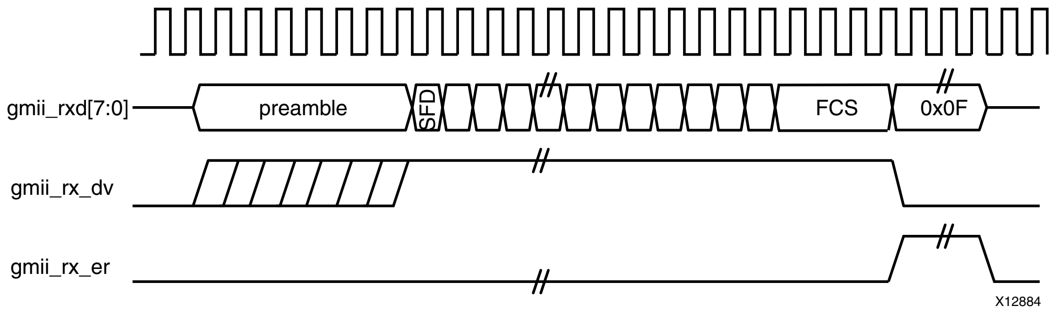


Figure 3-31: **GMII Normal Frame Reception with Carrier Extension**

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 3-32). In addition, a late error can also be detected during the Carrier Extension interval. This is indicated by gmii_rxd[7:0] containing the hexadecimal value 0x1F, also shown in Figure 3-32.

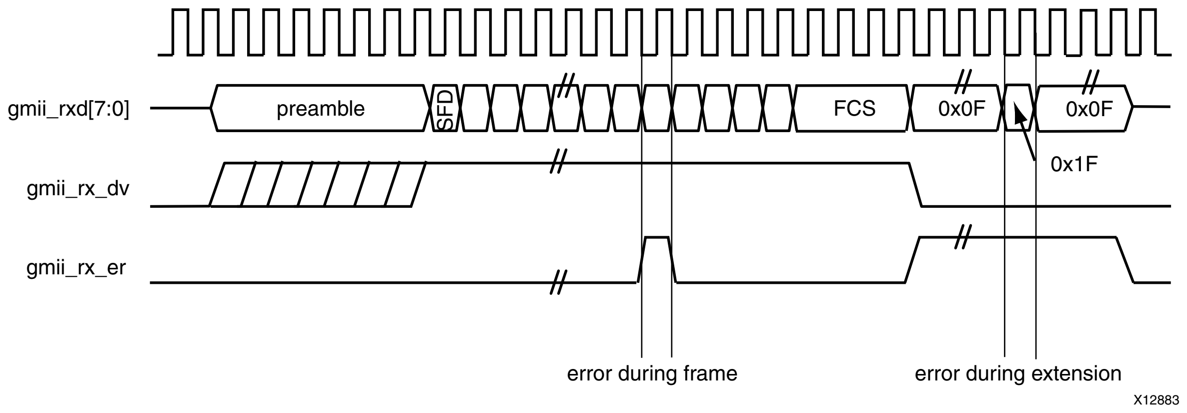


Figure 3-32: **GMII Frame Reception with Errors**

False Carrier

Figure 3-33 shows 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.

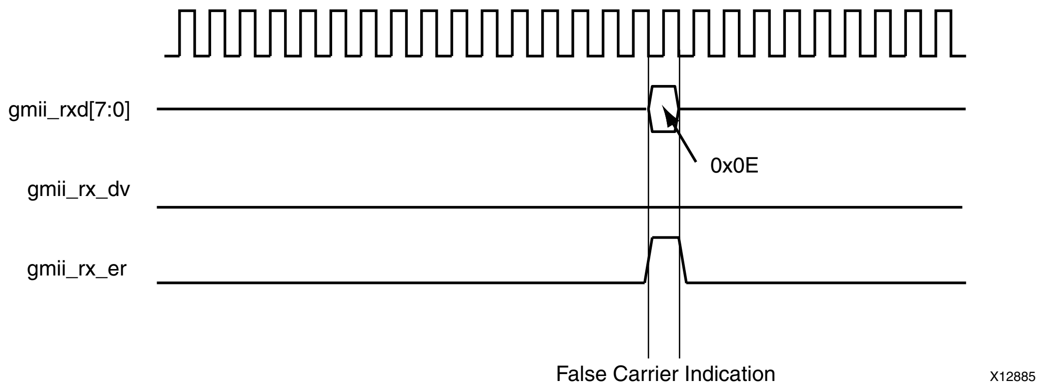


Figure 3-33: **False Carrier Indication**

status_vector[15:0] Signals

Figure 3-34 shows an error occurring in the second clock cycle of an /I/ idle sequence.See Table 2-75 for the status_vector bit definitions.

Bits[6:2]: Code Group Reception Indicators

These signals indicate the reception of particular types of groups, as defined in Table 2-75. Figure 3-34 shows 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

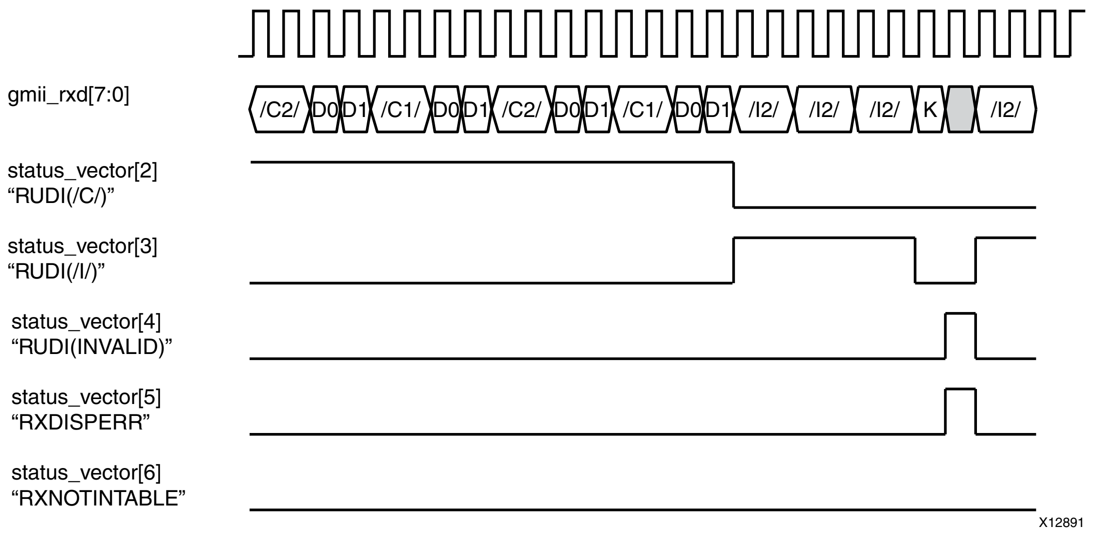

gmii_rxd[7:0]
/C2/
D0
D1
/C1/
D0
D1
/C2/
D0
D1
/C1/
D0
D1
/I2/
/I2/
/I2/
K
/I2/
status_vector[2] "RUDI(/C/)"
status_vector[3] "RUDI(/I/)"
status_vector[4] "RUDI(INVALID)"
status_vector[5] "RXDISPERR"
status_vector[6] "RXNOTINTABLE"
X12891

Figure 3-34: **status_vector[4:2] timing**

Figure 3-34 shows an error occurring in the second clock cycle of an /I/ idle sequence. RXDISPERR shows this as a running disparity error occurring in the second clock cycle of an /I/ idle sequence. **RXNOTINTABLE** means that the core has received a code group that is not recognized from the 8B/10B coding tables. If this error is detected, the timing of the `rxnotintable` signal would be identical to that of the `rxdisperr` signal shown in Figure 3-34.

Using the Client-Side GMII for the SGMII Standard

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 4). However, the datapath through both transmitter and receiver sections of the core remains unchanged.

GMII Transmission

1 Gb/s Frame Transmission

The timing of normal outbound frame transfer is shown in Figure 3-35. At 1 Gb/s speed, the operation of the transmitter GMII signals remains identical to that described in Using the Client-Side GMII for the 1000BASE-X Standard.

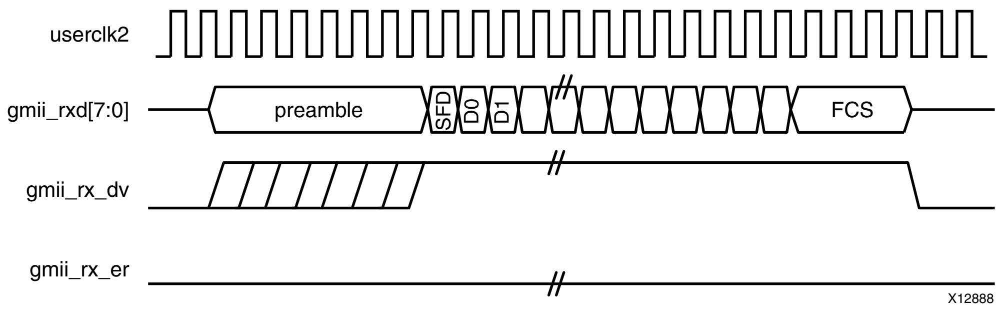


Figure 3-35: **GMII Frame Transmission at 1 Gb/s**

100 Mb/s 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 Mb/s, every byte of the MAC frame (from preamble to the Frame Check Sequence field, inclusive) should each be repeated for 10 clock periods to achieve the desired bit rate, as shown in Figure 3-36. It is also the responsibility of the client logic to ensure that the interframe gap period is legal for the current speed of operation. Only when the core is connected to ENET0/1 in the Zynq-7000 AP SoC processor subsystem, the core will take care of converting the 4-bit MII interface to 8 bit as required by the core. In all other cases the core expects 8 bits from client logic.

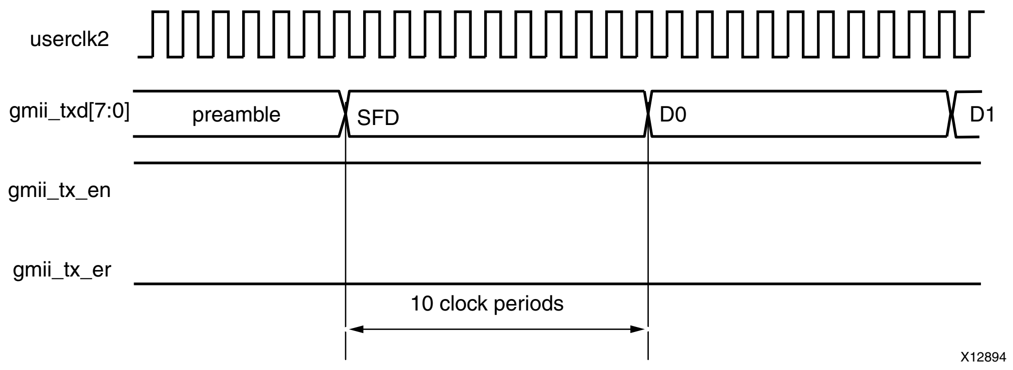


Figure 3-36: **GMII Data Transmission at 100 Mb/s**

10 Mb/s 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 Mb/s, every byte of the MAC frame (from preamble 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. Only when the core is connected to ENET0/1 in the Zynq-7000 AP SoC processor subsystem, the core will take care of converting the 4-bit MII interface to 8 bit as required by the core. In all other cases the core expects 8 bits from client logic.

GMII Reception

1 Gb/s Frame Reception

The timing of normal inbound frame transfer is shown in Figure 3-37. At 1 Gb/s speed, the operation of the receiver GMII signals remains identical to that described in Using the Client-Side GMII for the 1000BASE-X Standard.

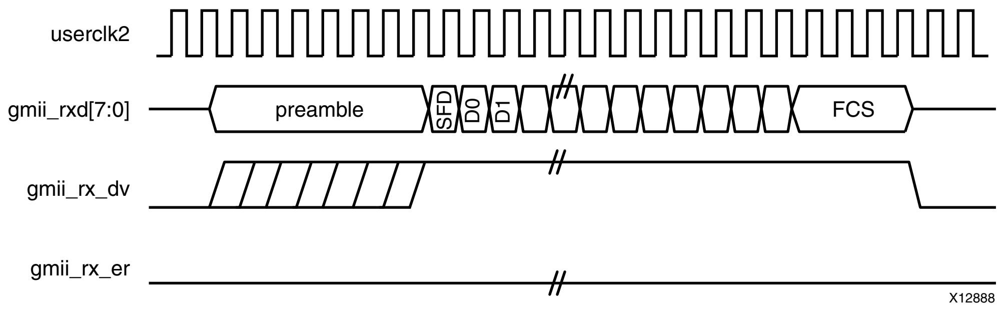


Figure 3-37: **GMII Frame Reception at 1 Gb/s**

100 Mb/s Frame Reception

The operation of the core remains unchanged. When operating at a speed of 100 Mb/s, every byte of the MAC frame (from preamble to the frame check sequence field, inclusive) is repeated for 10 clock periods to achieve the desired bit rate. See Figure 3-38. Only when the core is connected to ENET0/1 in the Zynq-7000 AP SoC processor subsystem, the core will take care of converting the 8 bit from the core to 4-bit MII interface. In other cases, it is the responsibility of the client logic, for example an Ethernet MAC, to sample this data correctly.

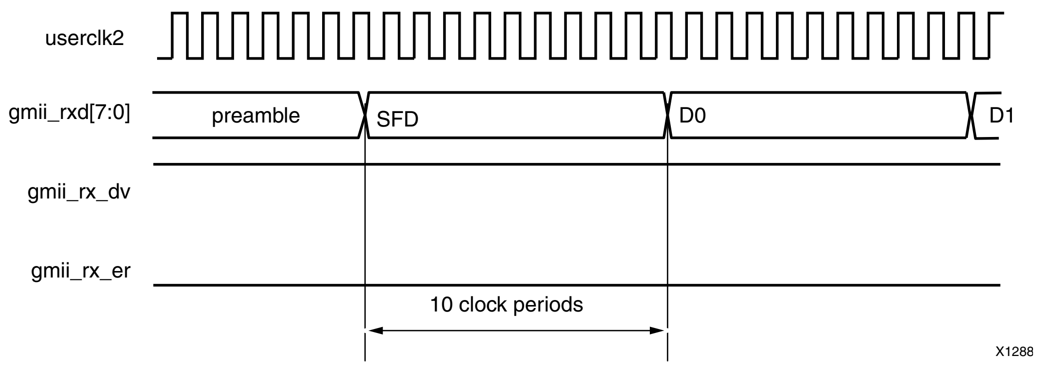


Figure 3-38: **GMII Data Reception at 100 Mb/s**

10 Mb/s Frame Reception

The operation of the core remains unchanged. When operating at a speed of 10 Mb/s, every byte of the MAC frame (from preamble to the frame check sequence field, inclusive) is repeated for 100 clock periods to achieve the desired bit rate. Only when the core is connected to ENET0/1 in the Zynq-7000 AP SoC processor subsystem, the core will take care of converting the 8 bit from the core to 4-bit MII interface. In other cases, it is the responsibility of the client logic (for example, an Ethernet MAC) to sample this data correctly.

Additional Client-Side SGMII Logic

When the core is generated in SGMII or Dynamic Switching mode, the block level of the core contains the SGMII Adaptation Module (this is shown in Figure 3-39 for a core using a device specific transceiver as the physical interface). This SGMII adaptation module is described in the remainder of this section.

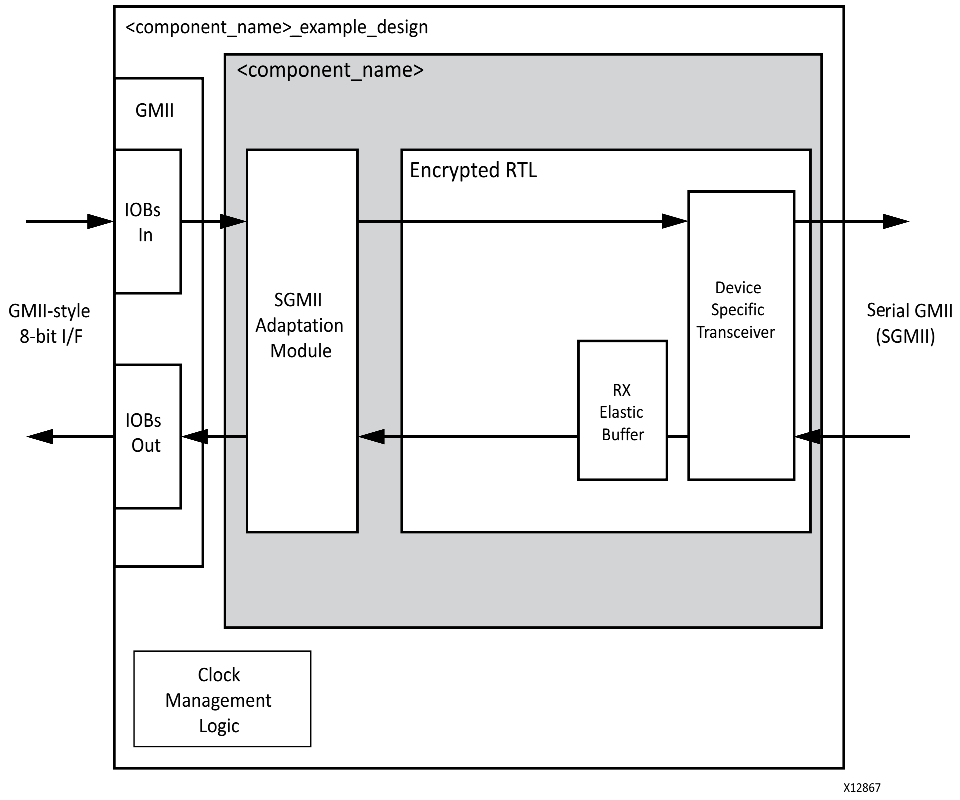


Figure 3-39: **Block Level Diagram of an SGMII Example Design**

Because the GMII of the core always operates at 125 MHz, the core does not differentiate between the three SGMII speeds of operation, it always effectively operates at 1 Gb/s. However, as described in Using the Client-Side GMII for the 1000BASE-X Standard, at 100 Mb/s, every data byte run through the core is repeated ten times to achieve the required bit rate; similarly, at 10 Mb/s, each data byte run through the core is repeated 100 times to achieve the required bit rate. Dealing with this repetition of bytes is the function of the SGMII adaptation module.

The SGMII adaptation module (Figure 3-40) creates a GMII-style interface that drives/samples the GMII data and control signals at the following frequencies:

- 125 MHz when operating at a speed of 1 Gb/s (with no repetition of data bytes)
- 12.5 MHz at a speed of 100 Mb/s (each data byte is repeated and run through the core 10 times)
- 1.25 MHz at a speed of 10 Mb/s (each data byte is repeated and run through the core 100 times)

When the core is connected to ENET0/1 in the Zynq-7000 AP SoC processor subsystem, the SGMII adaptation module performs the additional function of converting the 8 bits from the core to a 4-bit MII interface and vice versa. The function of the SGMII adaptation module is therefore to create a proprietary interface that is based on GMII (true GMII only operates at a clock frequency of 125 MHz for an ethernet line rate of 1.25 Gb/s). This interface then allows a straightforward internal connection to an Ethernet MAC core when operating in MAC mode or the GMII can be brought out on pads to connect to an external PHY when the core operates in PHY mode. For example, the SGMII adaptation module can be used to interface the core, operating in SGMII configuration with MAC mode of operation, to the Xilinx Tri-Mode Ethernet MAC core directly (see Interfacing to Other Cores). The GMII interface of the SGMII adaptation module can brought out to the pads and connected to an external PHY module that converts GMII to a Physical Medium Dependent (PMD) signal when the core is operating in SGMII configuration and PHY mode of operation.

SGMII Adaptation Module Top Level

The SGMII adaptation module is described in several hierarchical submodules as shown in Figure 3-40. These submodules are each described in separate HDL files and are described in the following sections.

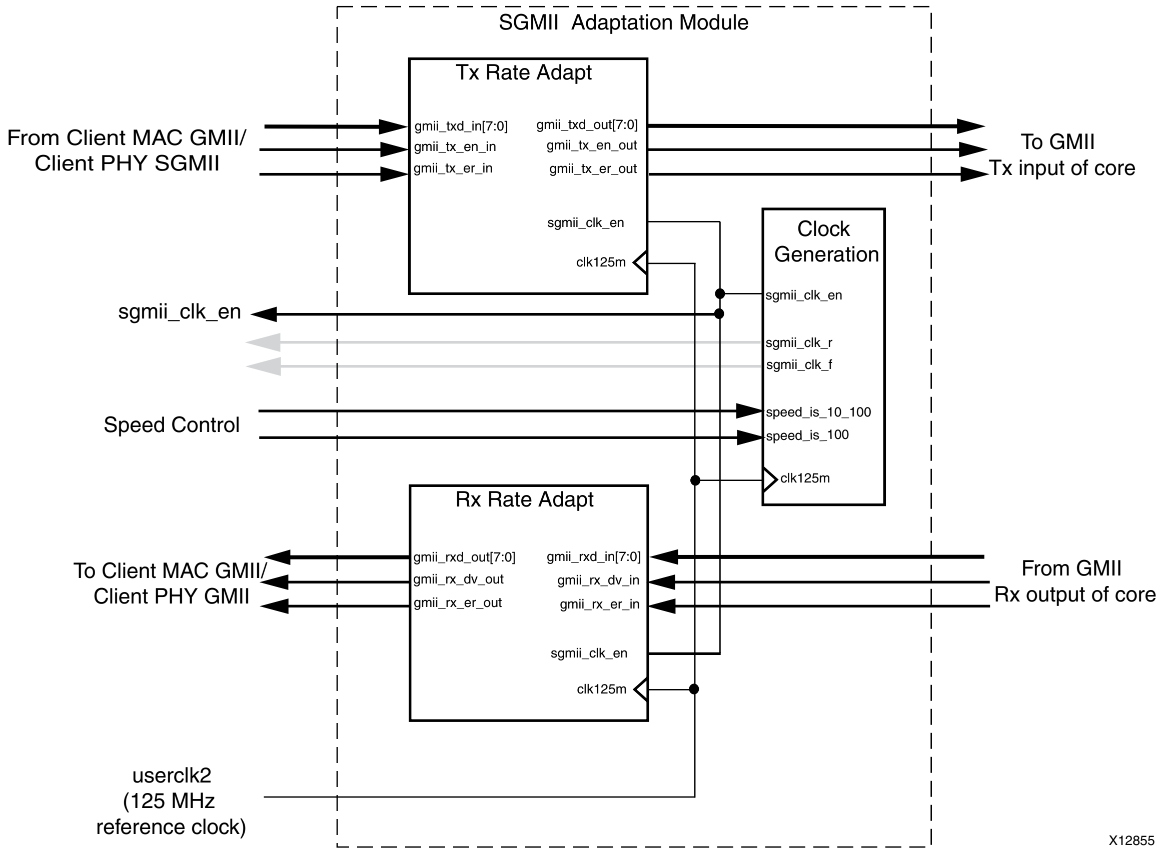


Figure 3-40: **SGMII Adaptation Module**

Transmitter Rate Adaptation Module

Interfacing with Client Proprietary Logic/ IP Catalog Tri-Mode Ethernet MAC

This module accepts transmitter data from the GMII-style interface from the attached client MAC/External PHY, and samples the input data on the 125 MHz reference clock, `clk125m`. This sampled data can then be connected directly to the input GMII of the Ethernet 1000BASE-X PCS/PMA, or SGMII netlist. The 1 Gb/s and 100 Mb/s cases are shown in Figure 3-41.

At all speeds, the client MAC/External PHY logic should drive the GMII transmitter data synchronously to the rising edge of the 125 MHz reference clock while using `sgmii_clk_en` (derived from the Clock Generation module) as a clock enable. The frequency of this clock enable signal ensures the correct data rate and correct data sampling between the two devices.

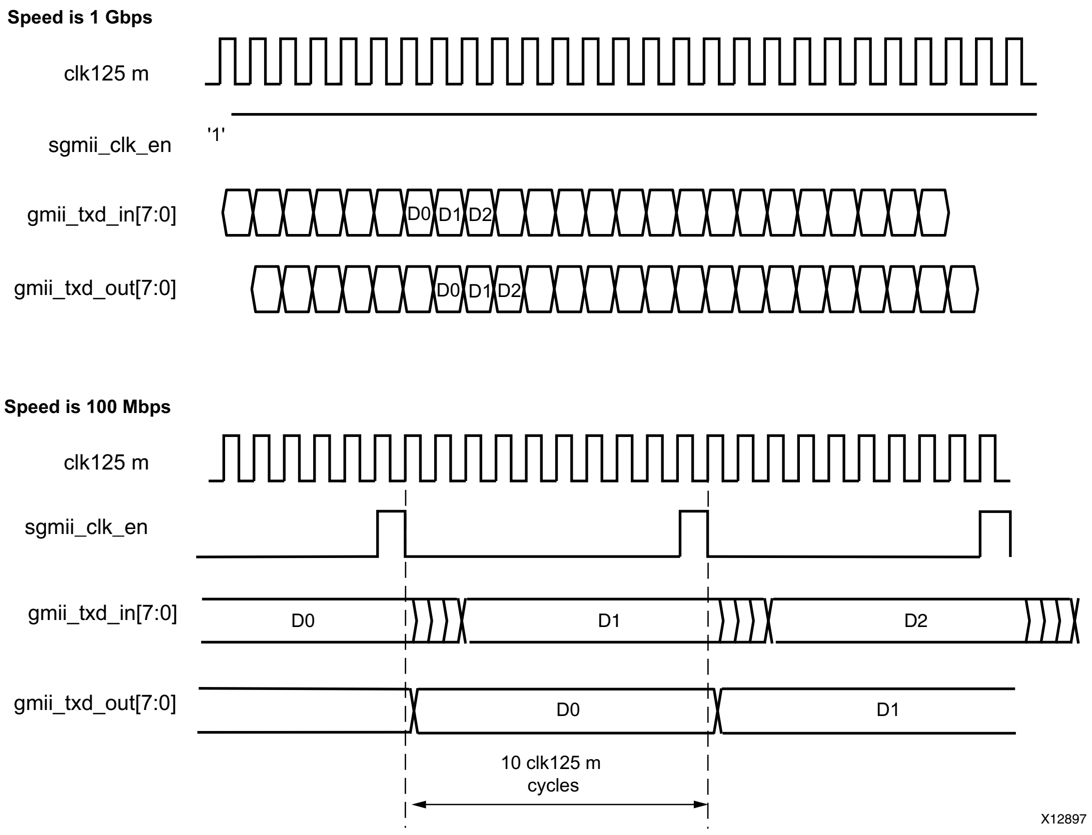


Figure 3-41: **Transmitter Rate Adaptation Module Data Sampling**

Interfacing with ENET0/1 in Zynq-7000 Device PS

When the speed is 1Gb/s, the data is received on the 125 MHz clock (`clk125m`). When a speed of 10/100 Mb/s is selected, 4 bits of MII are received on the LSB 4 bits of the GMII interface. This interface is converted to 8 bits by sampling with `sgmii_ddr_clk_en` (internally derived from the Clock Generation module).

This 8-bit interface should drive the GMII transmitter data synchronously to the rising edge of the 125 MHz reference clock while using `sgmii_clk_en` (internally derived from the Clock Generation module) as a clock enable. It is possible that the SFD could have been skewed across two separate bytes, so 8-bit Start of Frame Delimiter (SFD) code is detected, and if required, it is realigned across the 8-bit datapath. The 1 Gb/s and 100 Mb/s cases are shown in Figure 3-42.

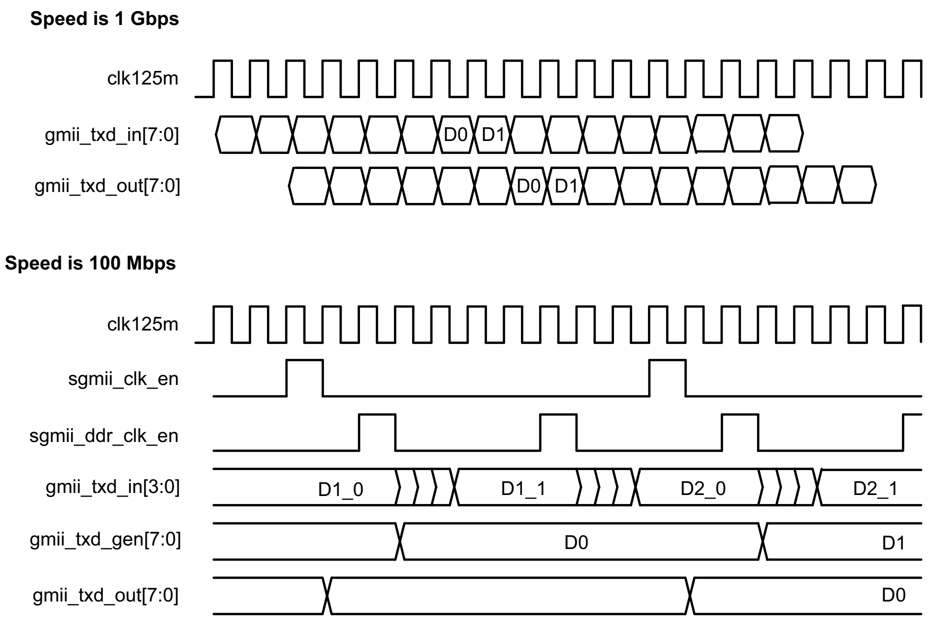


Figure 3-42: **Transmitter Rate Adaptation Module Data Sampling**

Receiver Rate Adaptation Module

Interfacing with Client Proprietary Logic/IP Catalog Tri-Mode Ethernet MAC

This module accepts received data from the Ethernet 1000BASE-X PCS or SGMII core. This data is sampled and sent out of the GMII receiver interface for the attached client MAC/External PHY. The 1 Gb/s and 100 Mb/s cases are shown in Figure 3-43.

At 1 Gb/s, the data is valid on every clock cycle of the 125 MHz reference clock (`clk125m`). Data received from the core is clocked straight through the Receiver Rate Adaptation module.

At 100 Mb/s, the data is repeated for a 10 clock period duration of `clk125m`; at 10 Mb/s, the data is repeated for a 100 clock period duration of `clk125m`. The Receiver Rate Adaptation Module samples this data using the `sgmii_clk_en` clock enable.

The Receiver Rate Adaptation module also performs a second function that accounts for the latency inferred in Figure 3-43. The 8-bit Start of Frame Delimiter (SFD) code is detected, and if required, it is realigned across the 8-bit datapath of `gmii_rxd_out[7:0]` before being presented to the attached client MAC. It is possible that this SFD could have been skewed across two separate bytes by MACs operating on a 4-bit datapath.

At all speeds, the client MAC/External PHY logic should sample the GMII receiver data synchronously to the rising edge of the 125 MHz reference clock while using `sgmii_clk_en` (derived from the Clock Generation module) as a clock enable. The frequency of the `sgmii_clk_en` clock enable signal ensures the correct data rate and correct data sampling between the two devices.

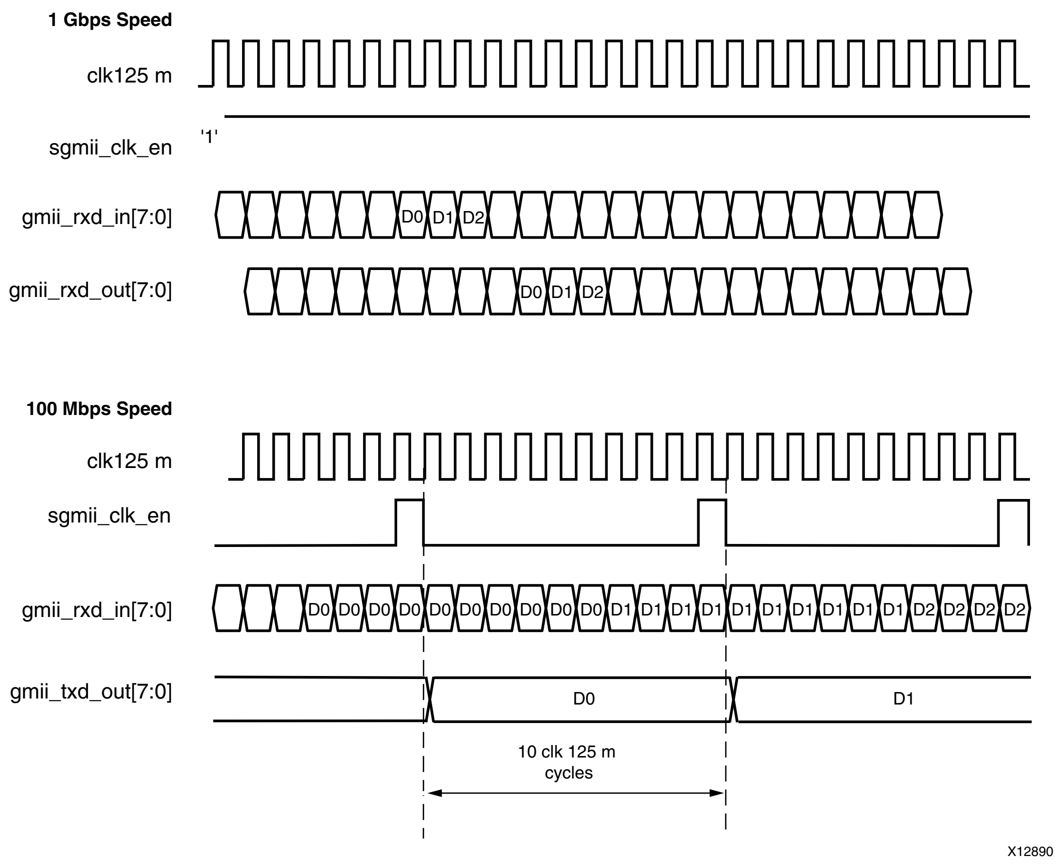


Figure 3-43: **Receiver Rate Adaptation Module Data Sampling**

Interfacing with ENET0/1 in Zynq-7000 Device PS

This module accepts received data from the core. This data is sampled and sent out of the GMII receiver interface for the attached external PHY. The 1 Gb/s and 100 Mb/s cases are shown in Figure 3-44.

At 1 Gb/s the data is valid on every clock cycle of the 125 MHz reference clock (`clk125m`). Data received from the core is clocked straight through the Receiver Rate Adaptation module.

At 100 Mb/s, the data is repeated for a 10 clock period duration of `clk125m`; at 10 Mb/s, the data is repeated for a 100 clock period duration of `clk125m`. The Receiver Rate Adaptation Module samples this data using the `sgmii_clk_en` clock enable generated internally in clock generation module. Then the lower half of the byte is sent on the LSB 4 bits of `gmii_rxd_out[3:0]` followed by the upper nibble. This operation is done on `sgmii_ddr_clk_en` generated internally in clock generation module.

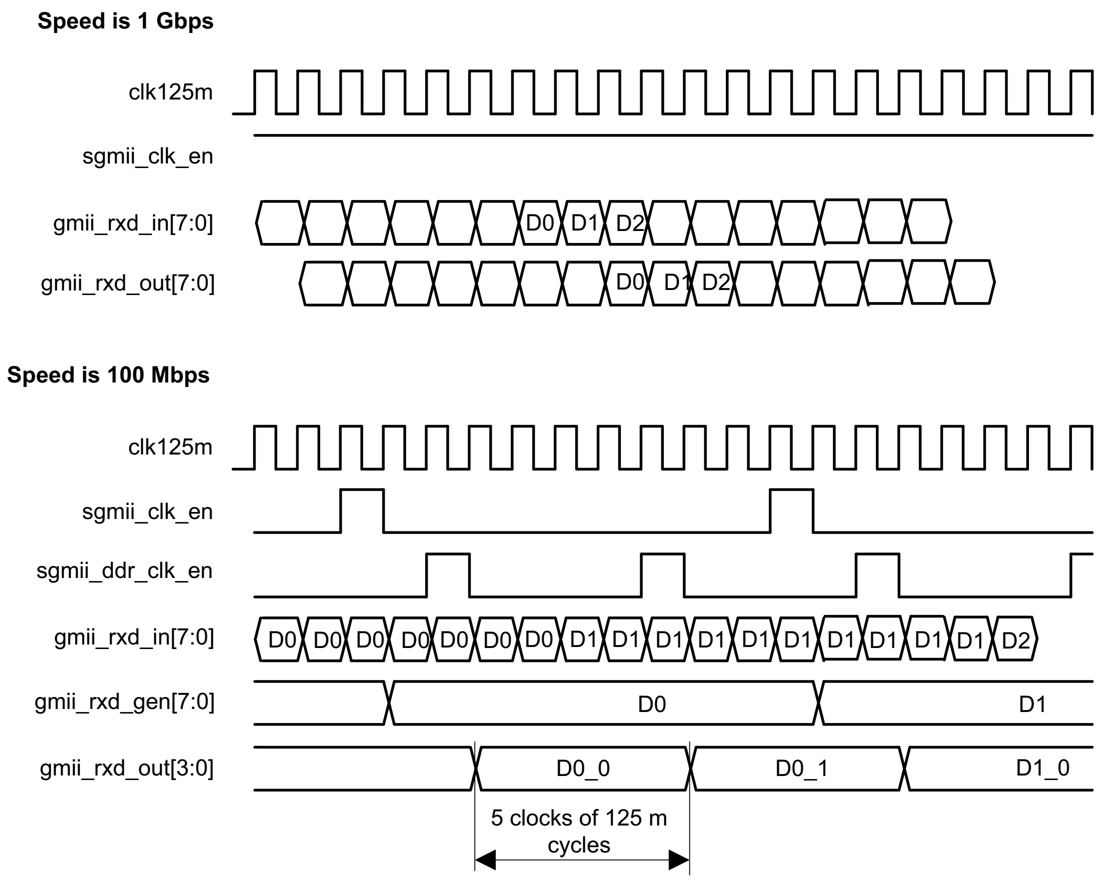


Figure 3-44: **Receiver Rate Adaptation Module Data Sampling**

Clock Generation

Interfacing with Client Proprietary Logic/IP Catalog Tri-Mode Ethernet MAC

This module creates the `sgmii_clk_en` clock enable signal for use throughout the SGMII adaptation module. Clock enabled frequencies are:

- 125 MHz at an operating speed of 1 Gb/s
- 12.5 MHz at an operating speed of 100 Mb/s
- 1.25 MHz at an operating speed of 10 Mb/s

Figure 3-45 shows the output clock enable signal for the Clock Generation module at 1 Gb/s and 100 Mb/s speeds.

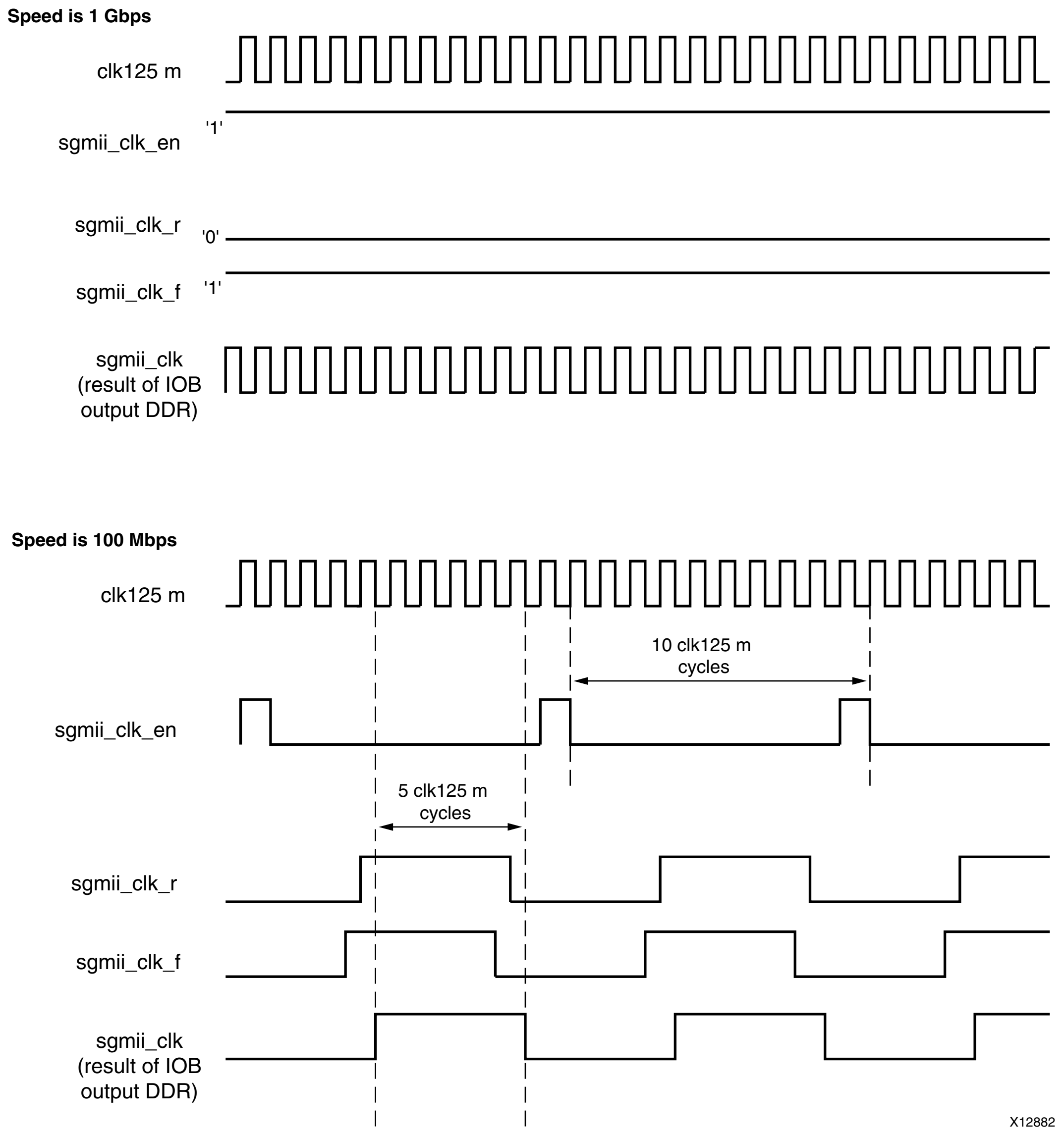


Figure 3-45: **Clock Generator Output Clocks and Clock Enable**

Figure 3-45 also shows the formation of the `sgmii_clk_r` and `sgmii_clk_f` signals. These are used only in the example design delivered with the core, where they are routed to a device IOB DDR output register. This provides SGMII clock forwarding at the correct frequency; these signal can be ignored when connecting the core and SGMII Adaptation module to internal logic.

Interfacing with ENET0/1 in Zynq-7000 Device PS

This module creates the `sgmii_clk_en`, `sgmii_ddr_clk_en` clock enable signals for use throughout the SGMII adaptation module. Figure 3-46 shows the clock enable signal for the Clock Generation module at 1Gb/s and 100 Mb/s speeds.

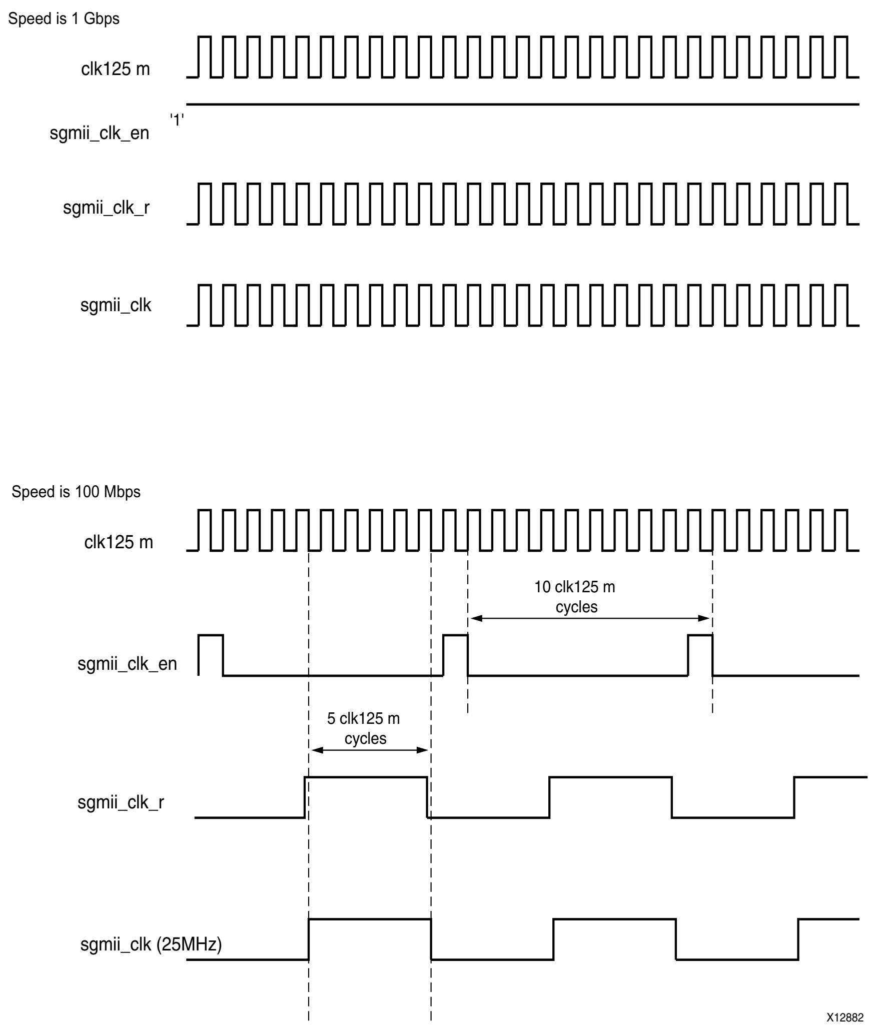


Figure 3-46: **Clock Enable Signal for the Clock Generation Module**

Figure 3-46 also shows the formation of the `sgmii_clk_r` signal. `sgmii_clk_r` should be connected to the `sgmii_clk` port of ENET0/1 core generated.

This provides SGMII clock forwarding at the correct frequency.

Note:

1. `sgmii_clk_f` signal is not used in this case.
2. `sgmii_clk_en` is not given as an output but used internally within SGMII adaptation module.

The `sgmii_clk_r` frequencies for various modes of operation are:

- 125 MHz at an operating speed of 1 Gb/s
- 25 MHz at an operating speed of 100 Mb/s
- 2.5 MHz at an operating speed of 10 Mb/s

Auto-Negotiation

This section provides general guidelines for using the auto-negotiation function of the core. Auto-Negotiation is controlled and monitored through the PCS management registers. For more information, see Register Space in Chapter 2.

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

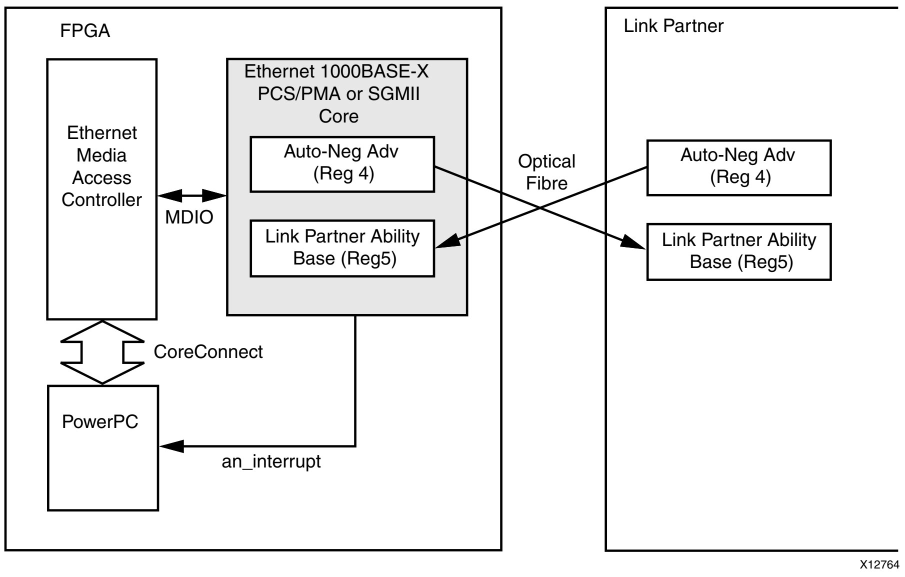

FPGA
Ethernet Media Access Controller
Ethernet 1000BASE-X PCS/PMA or SGMII Core
Auto-Neg Adv (Reg 4)
Link Partner Ability Base (Reg5)
MDIO
CoreConnect
PowerPC
an_interrupt
Optical Fibre
Link Partner
Auto-Neg Adv (Reg 4)
Link Partner Ability Base (Reg5)
X12764

Figure 3-47: **1000BASE-X Auto-Negotiation Overview**

IEEE 802.3-2008 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 3-47 shows 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 Register 0: Control Register and `an_restart_config` in Table 2-28.)
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 Register 4: Auto-Negotiation Advertisement for more information.

   This register is also writable through dedicated interface signal `an_adv_config_vector`. If optional MDIO is present, the additional signal `an_adv_config_valid` quantifies the contents of `an_adv_config_vector`. See definitions of `an_adv_config_vector` and `an_adv_config_valid` in Table 2-28 for more information.

   Information provided in this register includes:
   - Fault Condition signaling
   - Duplex Mode
   - Flow Control capabilities for the attached Ethernet 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 Register 5: Auto-Negotiation Link Partner Base for more information. Remote Fault and pause status bits of this register are also provided in status_vector.
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. OR by reading the remote_fault and pause status bits of status_vector if MDIO is not present. Other networking components, such as an attached Ethernet MAC, should be configured accordingly. See Register 5: Auto-Negotiation Link Partner Base for more information.

There are two methods that a host processor uses to learn of the completion 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).

SGMII Standard

Using the SGMII MAC Mode to Interface to an External BASE-T PHY with SGMII Interface

Figure 3-48 shows the operation of SGMII auto-negotiation as described in Overview of Operation. Additional information about SGMII Standard auto-negotiation is provided in the following sections.

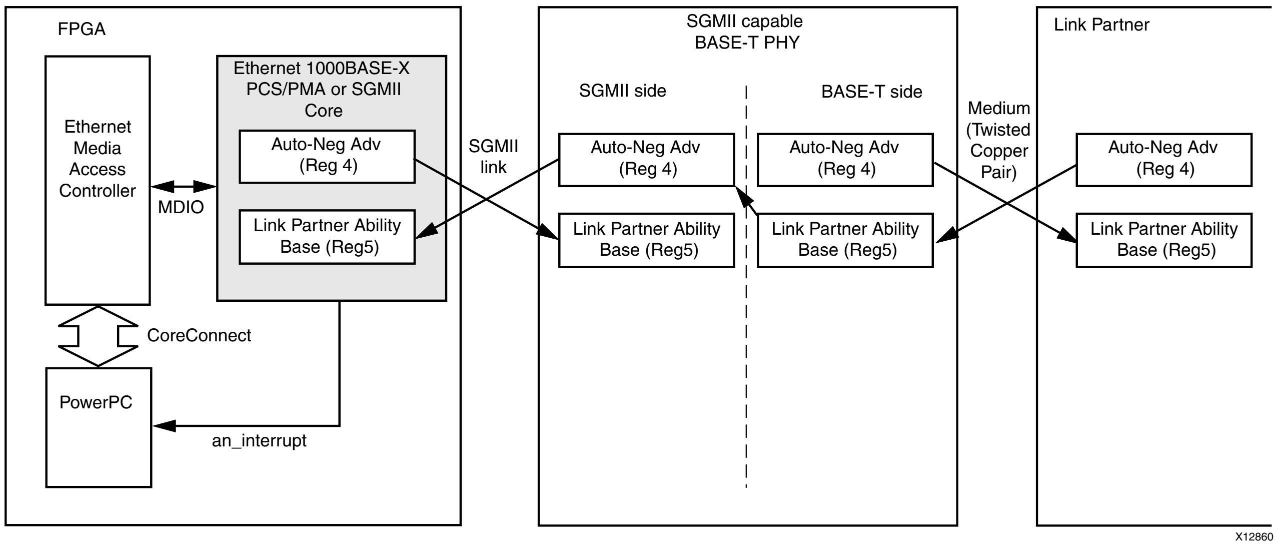

FPGA
Ethernet Media Access Controller
Ethernet 1000BASE-X PCS/PMA or SGMII Core
Auto-Neg Adv (Reg 4)
Link Partner Ability Base (Reg5)
MDIO
CoreConnect
PowerPC
an_interrupt
SGMII link
SGMII capable BASE-T PHY
SGMII side
BASE-T side
Auto-Neg Adv (Reg 4)
Link Partner Ability Base (Reg5)
Auto-Neg Adv (Reg 4)
Link Partner Ability Base (Reg5)
Medium (Twisted Copper Pair)
Link Partner
Auto-Neg Adv (Reg 4)
Link Partner Ability Base (Reg5)
X12860

Figure 3-48: **SGMII Auto-Negotiation in MAC Mode**

The SGMII capable PHY has two distinctive sides to auto-negotiation.

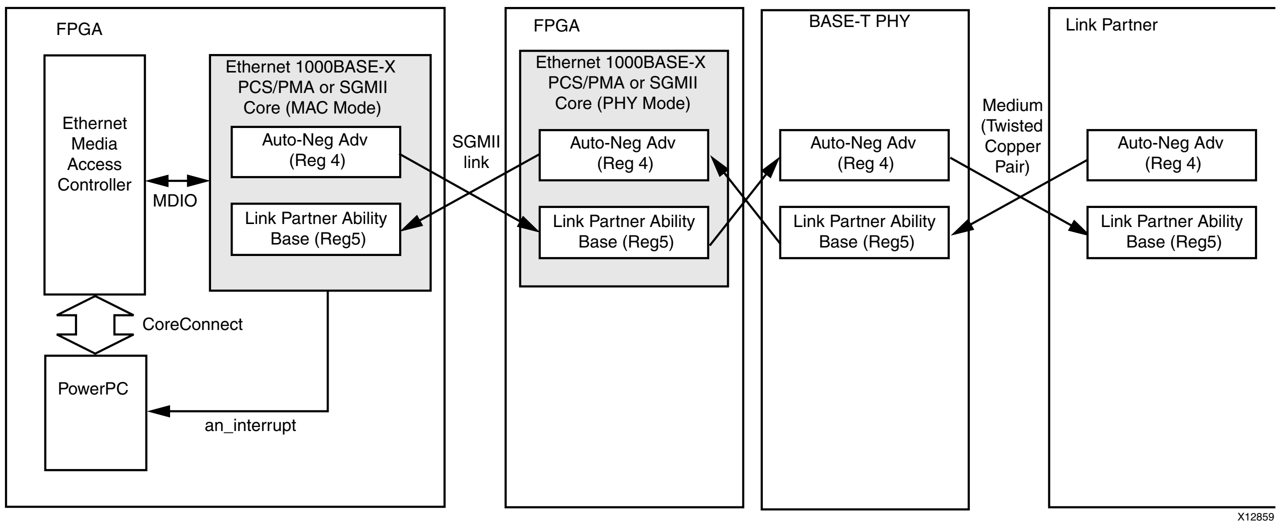
- The PHY performs auto-negotiation with its link partner using the relevant auto-negotiation standard for the chosen medium (BASE-T auto-negotiation is shown in Figure 3-48, using a twisted copper pair as its medium). This resolves the operational speed and duplex mode with the link partner.
- The PHY then passes the results of the auto-negotiation process with the link partner to the core (in SGMII mode), by leveraging the 1000BASE-X auto-negotiation specification described in Figure 3-47. 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.
- The information exchanged is different and now contains speed resolution in addition to duplex mode. See Register 5: Auto-Negotiation Link Partner Base. Speed and Duplex status bits of this register are also provided in status_vector.
- There are no other differences and dealing with the results of auto-negotiation can be handled as described previously in Figure 3-47.

Using Both the SGMII MAC Mode and SGMII PHY Mode Configurations to interface to an External BASE-T PHY with a GMII interface

Figure 3-49 shows the operation of SGMII auto-negotiation. Additional information about SGMII Standard auto-negotiation is provided in the following sections.



FPGA
Ethernet Media Access Controller
Ethernet 1000BASE-X PCS/PMA or SGMII Core (MAC Mode)
Auto-Neg Adv (Reg 4)
Link Partner Ability Base (Reg5)
MDIO
CoreConnect
PowerPC
an_interrupt
SGMII link
FPGA
Ethernet 1000BASE-X PCS/PMA or SGMII Core (PHY Mode)
Auto-Neg Adv (Reg 4)
Link Partner Ability Base (Reg5)
BASE-T PHY
Auto-Neg Adv (Reg 4)
Link Partner Ability Base (Reg5)
Medium (Twisted Copper Pair)
Link Partner
Auto-Neg Adv (Reg 4)
Link Partner Ability Base (Reg5)
X12859

Figure 3-49: **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 shown in Figure 3-49, using a twisted copper pair as its medium). This resolves the operational speed and duplex mode with the link partner. The BASE-T PHY transfers the link partner abilities though the MDIO interface to the core (in SGMII configuration and PHY mode).
- The core (in SGMII configuration and PHY mode) then passes the results of the auto-negotiation process to the core (in SGMII configuration and MAC mode), by leveraging the 1000BASE-X auto-negotiation specification described in Overview of Operation. 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.
- The information exchanged is different and now contains speed resolution in addition to duplex mode. See Register 5: Auto-Negotiation Link Partner Base. There are no other differences and dealing with the results of auto-negotiation can be handled as described previously in Overview of Operation.

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).

The operation of this port is enabled or disabled and cleared through 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 is set to logic 1 following the completion of an auto-negotiation cycle. It remains High until it is cleared by writing 0 to bit 16.1 (Interrupt Status bit) of the Register 16: Vendor-Specific Auto-Negotiation Interrupt Control.

Clock Correction Sequences in Device-Specific Transceivers (1000BASE-X Standard)

The device-specific transceivers are configured by the appropriate transceiver wizard to perform clock correction. The output of the transceiver wizard is provided as part of the example design. Two different clock correction sequences can be employed:

1. The mandatory clock correction sequence is the /I2/ ordered set; this is a two byte code-group sequence formed from /K28.5/ and /D16.2/ characters. The /I2/ ordered-set is present in the inter-frame-gap. These sequences can therefore be removed or inserted by the transceiver receive elastic buffer without causing frame corruption.
2. The default transceiver wizard configuration for the device-specific transceivers varies across device families. Some of the transceiver wizards enable the CLK_COR_SEQ_2_USE attribute. When this is the case, the transceiver is also configured to perform clock correction on the /K28.5/D21.5/ sequence; this is the first two code-groups from the /C1/ ordered set (the /C1/ ordered-set is four code-groups in length).

   Because there are no /I2/ ordered-sets present during much of the auto-negotiation cycle, this provides a method of allowing clock correction to be performed during auto-negotiation.

Because this form of clock correction inserts or removes two-code groups into or from a four-code group sequence, this causes ordered-set fragments to be seen by the cores auto-negotiation state machine. It is therefore important that the transceivers `rxclkcorcnt[2:0]` port is correctly wired up to the core netlist; this indicates a clock correction event (and type) to the core. Using this signal, the cores state machine can interpret the clock-correction fragments and the auto-negotiation function can complete cleanly.

When the device-specific transceivers CLK_COR_SEQ_2_USE attribute is not enabled, no clock correction can be performed during much of the auto-negotiation cycle. When this is the case, it is possible that the transceivers receive elastic buffer could underflow or overflow as asynchronous clock tolerances accumulate. This results in an elastic buffer error. It is therefore important that the transceivers `rxbufstatus[2:0]` port is correctly wired up to the core netlist; this indicates a buffer error event to the core. Using this signal, the cores state machine can interpret the buffer error and the auto-negotiation function can complete cleanly.

Conclusion

The device-specific transceivers can be configured to optionally perform clock correction during the auto-negotiation cycle, and their default configuration varies from family to family. Regardless, if correctly connected, as per the example design, the core state machine can determine the transceivers elastic buffer behavior and auto-negotiation will complete cleanly.

Dynamic Switching of 1000BASE-X and SGMII Standards

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

Typical Application

Figure 3-50 shows a typical application for the 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 fiber.

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

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

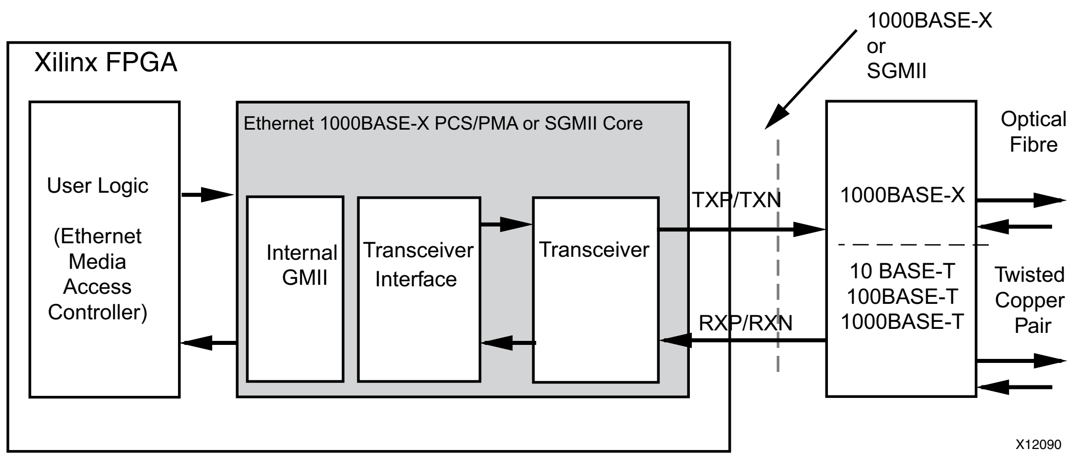


Figure 3-50: **Typical Application for Dynamic Switching**

Operation of the Core

Selecting the Power-On / Reset Standard

The external port of the core, `basex_or_sgmii` (see Dynamic Switching Signal Port), selects 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 Dynamic Switching Register 17 (see Table 2-72). Following completion of this write, the MDIO management registers immediately switch.

- Core set to 1000BASE-X standard. Management registers 0 through 16 should be interpreted according to 1000BASE-X Standard Using Optional Auto-Negotiation.
- Core set to SGMII standard. Management registers 0 through 16 should be interpreted according to SGMII Standard Using Optional Auto-Negotiation.

Auto-Negotiation State Machine

- Core set to the 1000BASE-X standard. The auto-negotiation state machine operates as described in 1000BASE-X Standard.
- Core set to perform the SGMII standard. The auto-negotiation state machine operates as described in SGMII Standard.
- Standard is switched during an auto-negotiation sequence. The auto-negotiation state machine does 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 an 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).

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 (v5.1 and later) to extend the system functionality to include the Ethernet MAC sublayer. The Tri-Mode Ethernet MAC core provides support for operation at 10 Mb/s, 100 Mb/s, and 1 Gb/s.

Note: The Tri-Mode Ethernet MAC core is abbreviated to the TEMAC core in this section.

A description of the latest available IP update containing the TEMAC core and instructions can be found in the Tri-Mode Ethernet MAC product web page.

CAUTION! *The TEMAC core 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 Code-Division Multiple Access (CDMA) protocol.

The TEMAC core v8.1 supports UltraScale architecture, Zynq-7000, Virtex-7, Kintex-7, and Artix-7 devices.

The 1000BASE-X PCS/PMA or SGMII core can also be integrated in a single device with the either of the Ethernet MAC (ENET0/ENET1) instances in the Zynq-7000 device processor subsystem to extend the system functionality to include the Ethernet MAC sublayer. ENET0/1 MACs provide support for operation at 10 Mb/s, 100 Mb/s, and 1 Gb/s.

Integration of the TEMAC for 1000BASE-X Operation

In this section, it is assumed that the TEMAC core is generated with only 1 Gb/s Ethernet speed and full-duplex only support. This provides the optimal solution.

Ten-Bit Interface Implementation

Figure 3-51 shows the connections and clock management logic required to interface the 1000BASE-X PCS/PMA or SGMII core (used in 1000BASE-X mode with the TBI) to the TEMAC core.

IMPORTANT: *The TEMAC core must be generated with the "interface" variable set as "Internal" for interfacing to the 1000BASE-X PCS/PMA or SGMII 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 TEMAC core, allowing the MAC to access the embedded configuration and status registers of the 1000BASE-X PCS/PMA or SGMII core.
- Due to the embedded receive elastic buffer in the 1000BASE-X PCS/PMA or SGMII core, 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 TEMAC core operates in the same clock domain.

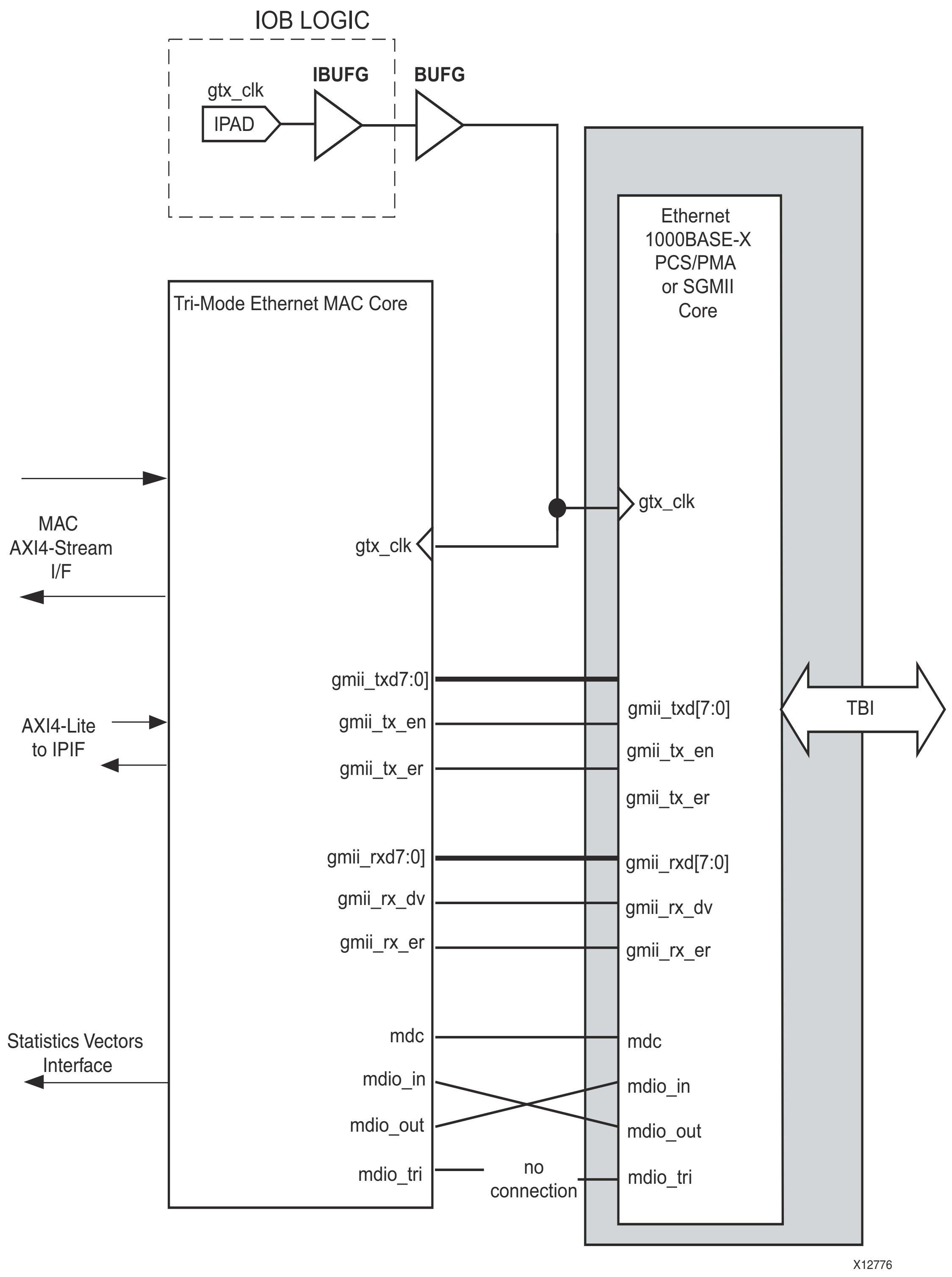


Figure 3-51: **Core with TBI Connected to TEMAC Core**

Transceiver Implementation

Figure 3-52 shows the connections and clock management logic required to interface the core (in 1000BASE-X mode) to the TEMAC core for Zynq-7000, Virtex-7 and Kintex-7 devices. Figure 3-53 shows the same interface for Artix-7 devices.

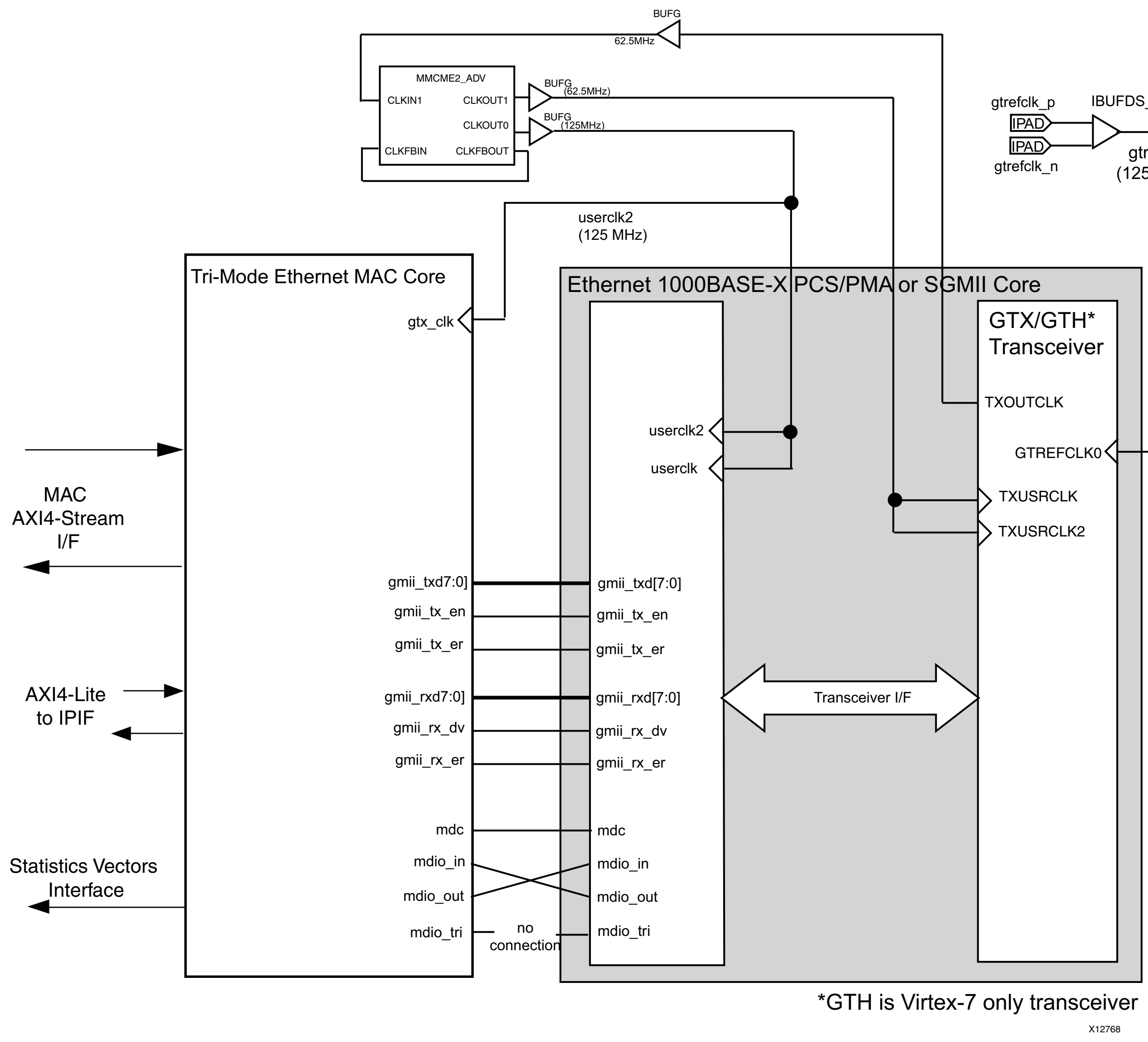


Figure 3-52: **Core Using GTX/GTH Transceivers Connected to the TEMAC Core**

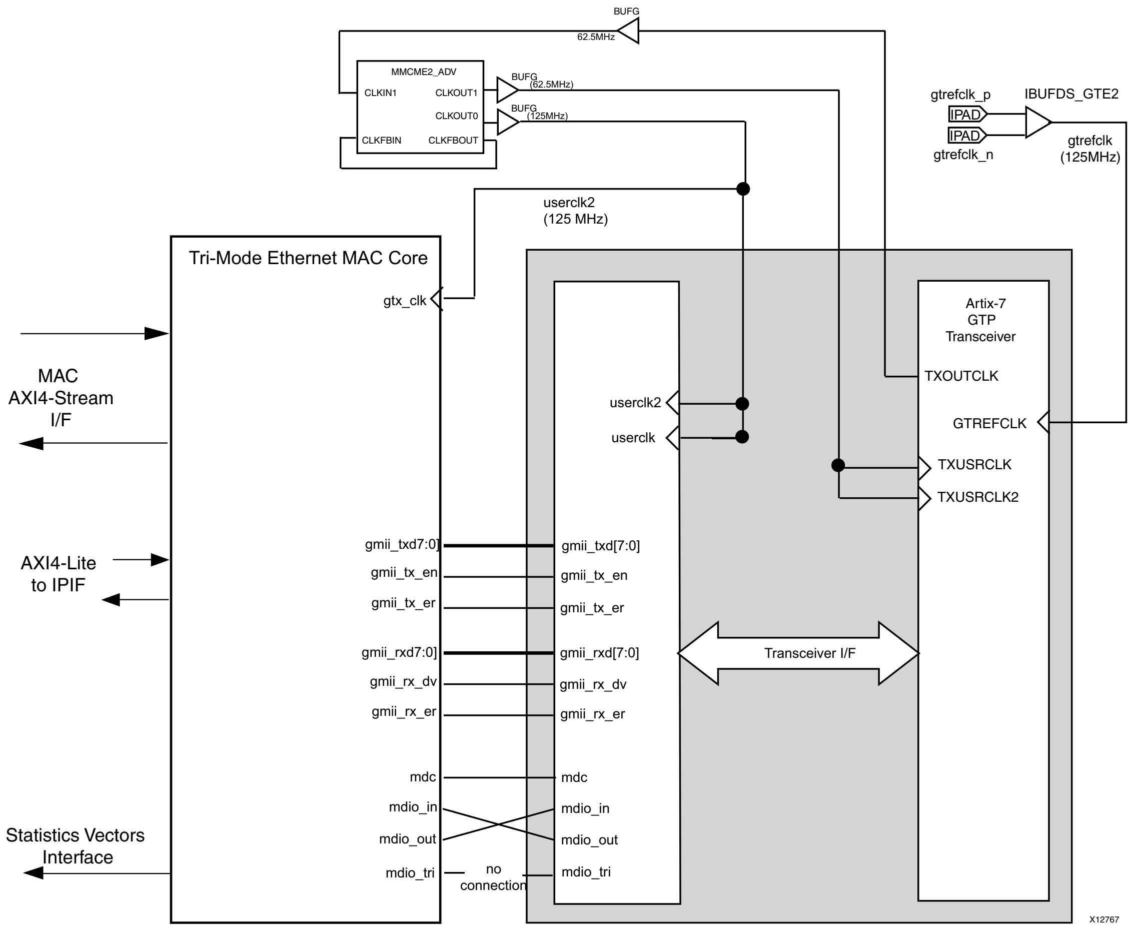


Figure 3-53: **Core Using Artix-7 Transceiver Connected to the TEMAC Core**

Features of this configuration include:

- Observe that the block level of the TEMAC is instantiated. This provides the MAC with extra functionality that is not provided by the TEMAC core netlist. When using the MAC to connect the 1000BASE-X core, the “Internal” PHY interface mode must be selected from the TEMAC Vivado IDE prior to core generation. See the *LogiCORE IP Tri-Mode Ethernet MAC Product Guide* (PG051) [Ref 12].
- 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 TEMAC core, allowing the MAC to access the embedded configuration and status registers of the 1000BASE-X PCS/PMA or SGMII core.
- Because of the embedded receive 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.

Integration of the TEMAC for Tri-speed SGMII Operation

In this section, it is assumed that the TEMAC core is generated for tri-speed operation and full-duplex only support. This provides the most optimal solution.

This section assumes only SGMII or Dynamic switching operation and MAC mode configuration. PHY mode configuration of SGMII is used to interface to a external PHY device. For SGMII in PHY mode configuration, see SGMII/Dynamic Switching with TBI Example Design and SGMII/Dynamic Switching with Transceivers. For 1000BASE-X only designs, see Transceiver Implementation.

Ten-Bit Interface Implementation

Figure 3-54 shows the connections and clock management logic required to interface the core (in SGMII mode with the TBI) to the TEMAC core.

IMPORTANT: *The TEMAC core must be generated with "interface" variable set as "Internal" for interfacing with 1000BASE-X PCS/PMA or SGMII core.*

Features of this configuration include:

- The SGMII Adaptation module, provided in the example design for the 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 TEMAC core, allowing the MAC to access the embedded configuration and status registers of the 1000BASE-X PCS/PMA or SGMII core.

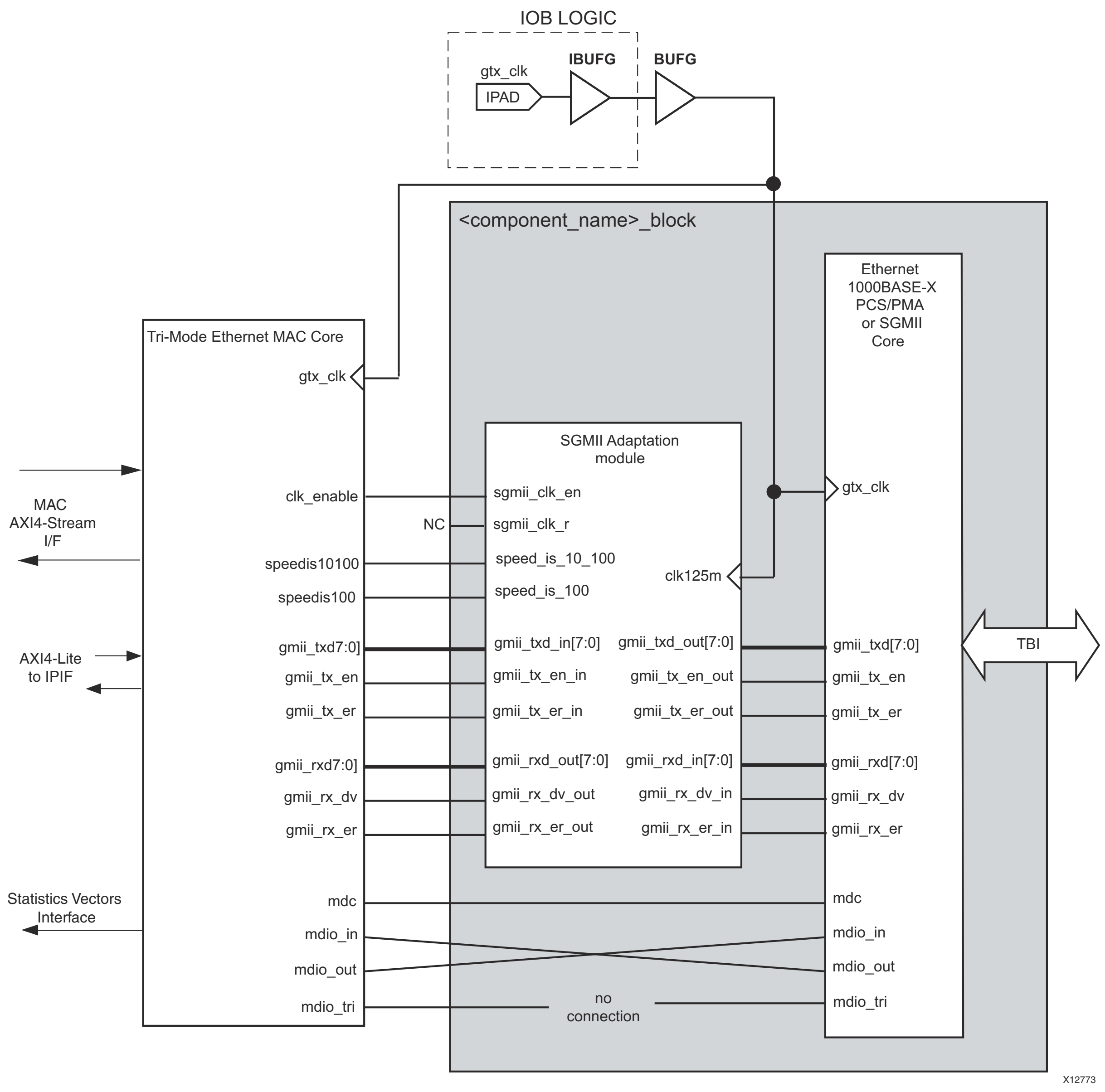


Figure 3-54: **Core Using TBI Connected to the TEMAC Core**

Transceiver Implementation

Figure 3-55 shows the connections and clock management logic required to interface the core (in SGMII Configuration and MAC mode with the GTX/GTH transceiver) to the TEMAC core for Zynq-7000, Virtex-7 and Kintex-7 devices. Figure 3-56 shows the same interface for Artix-7 devices.

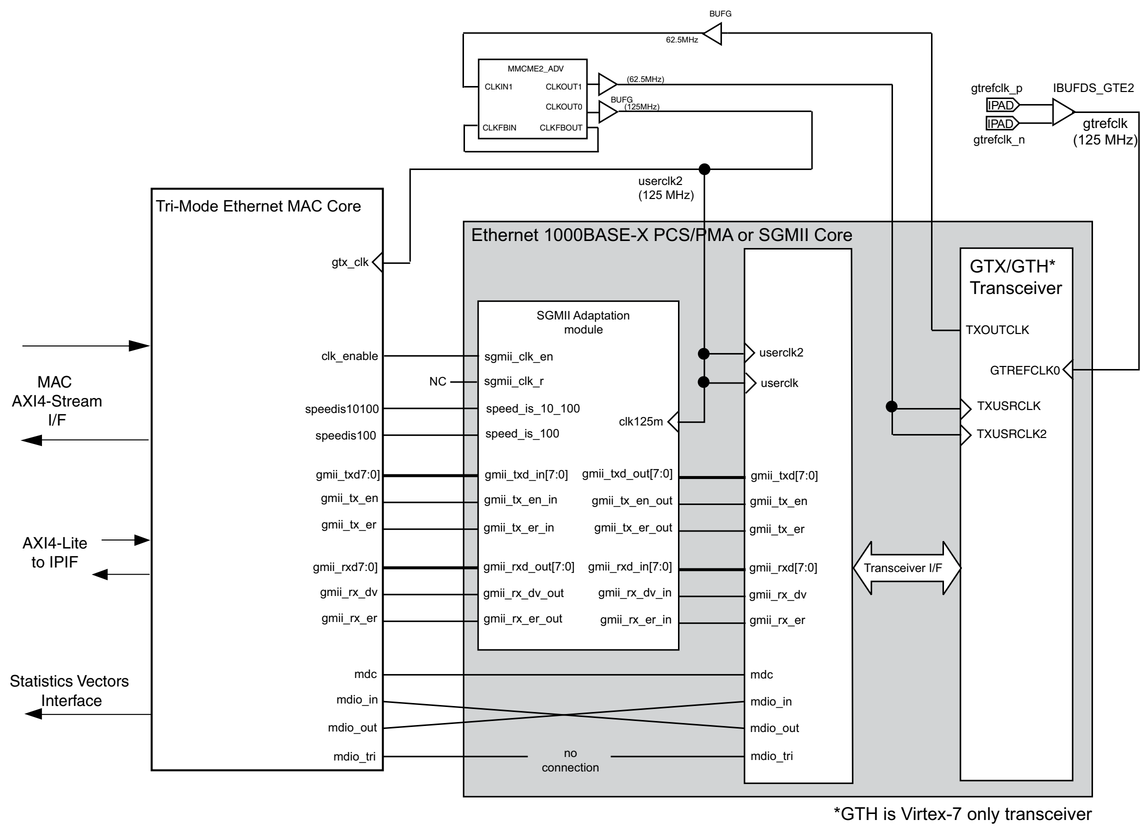


Figure 3-55: **Core Using SGMII with the GTX/GTH Transceiver Connected to the TEMAC Core**

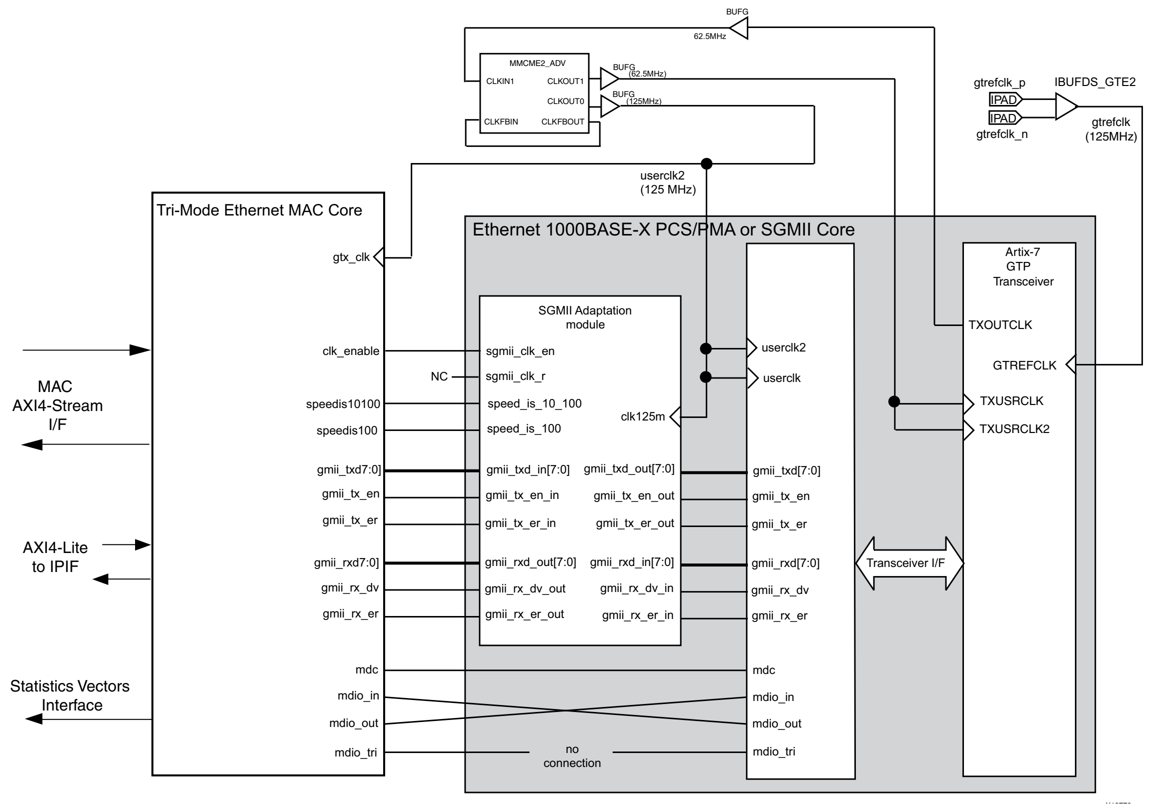


Figure 3-56: **Core Using SGMII with Artix-7 Transceiver Connected to the TEMAC Core**

Features of this configuration include:

- Observe that the block level of the TEMAC is instantiated. This provides the MAC with extra functionality that is not provided by the TEMAC core netlist. When using the MAC to connect the 1000BASE-X core, the "Internal" PHY interface mode must be selected from the TEMAC Vivado IDE prior to core generation. See the *LogiCORE IP Tri-Mode Ethernet MAC Product Guide* (PG051) [Ref 12].
- The SGMII Adaptation module, as provided in the example design for the core when generated to the SGMII standard and MAC mode, 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 TEMAC core, allowing the MAC to access the embedded configuration and status registers of the 1000BASE-X PCS/PMA or SGMII core.
- Because of the receive elastic buffer, the entire GMII (transmitter and receiver paths) 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 TEMAC core now operate in the same clock domain.

Integration of the TEMAC Core Using Sync SGMII over LVDS

Figure 3-57 shows the connections and clock management logic required to interface the core (in Sync SGMII over LVDS) to the TEMAC core. The block level of the Example Design should be taken from the example design and instantiated for connection to the TEMAC core. Connections from a unique TEMAC core to SGMII port are identical and are shown in Figure 3-57.

The following conditions apply to each connected TEMAC core and SGMII port pair:

- The SGMII Adaptation module, as provided in the example design for the 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 TEMAC core, allowing the MAC to access the embedded configuration and status registers of the 1000BASE-X PCS/PMA or SGMII core.
- clk125 is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the TEMAC core now operate in the same clock domain. This is the clock derived by MMCM and IBUFDS from differential reference clock.

Figure 3-57 shows a TEMAC core generated with the optional clock enable circuitry. This is recommended as the best way to connect the two cores together for efficient use of clock resources. See the *LogiCORE IP Tri-Mode Ethernet MAC Product Guide* (PG051) [Ref 12] for more information.

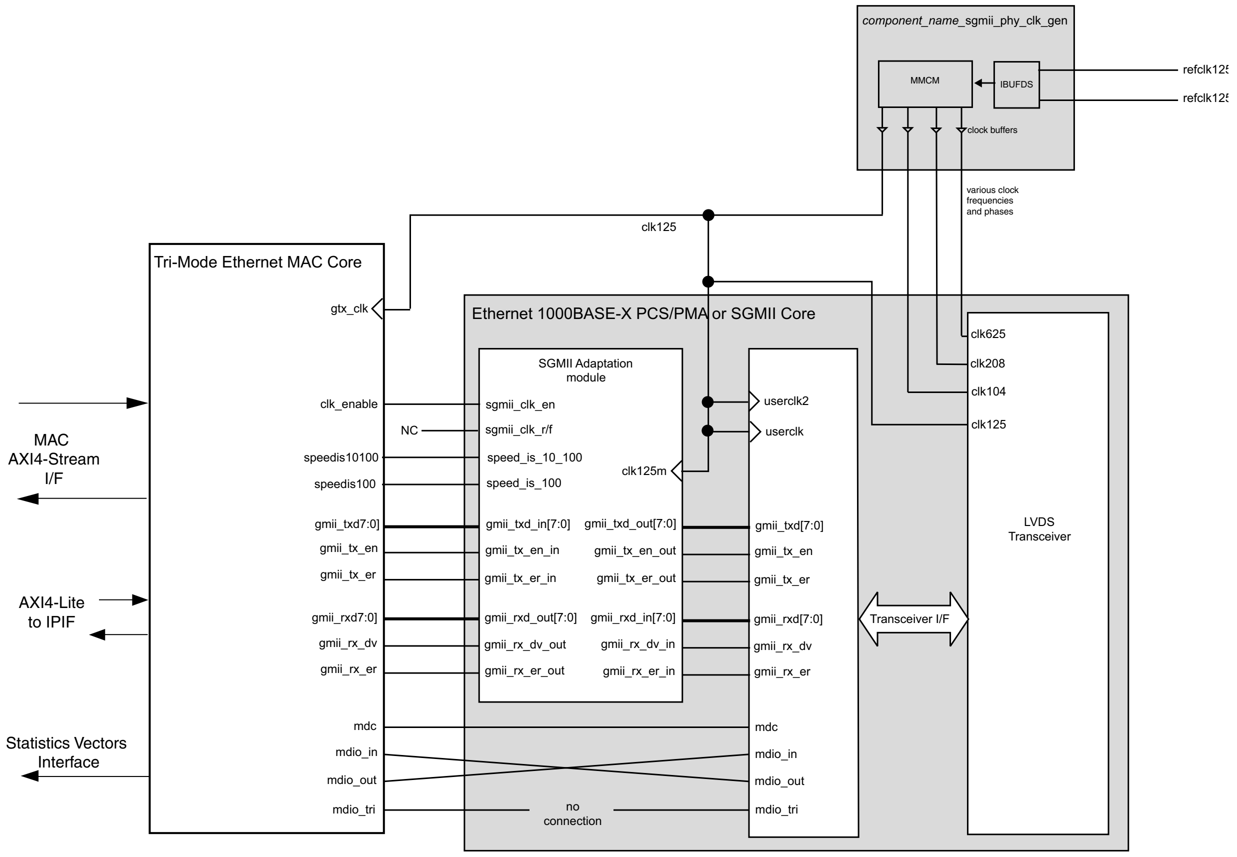


Figure 3-57: **Core Using SGMII over LVDS Connected to the TEMAC Core**

Integration of the Zynq-7000 Device PS ENET0/1 for 1000BASE-X Operation

Figure 3-58 shows the connections and clock management logic required to interface the core (in 1000BASE-X mode) to the Zynq-7000 device PS ENET0/1.

Features of this configuration include:

- Direct internal connections are made between the GMII interfaces between the ENET0/1 and 1000BASE-X PCS/PMA or SGMII core.
- The MDIO port can be connected, allowing the Ethernet MAC to access the embedded configuration and status registers of the 1000BASE-X PCS/PMA or SGMII core.
- Because of the embedded receive 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 ENET0/1 and 1000BASE-X PCS/PMA or SGMII core, and the transmitter and receiver logic of the Zynq-7000 device PS ENET0/1 now operate in the same clock domain.

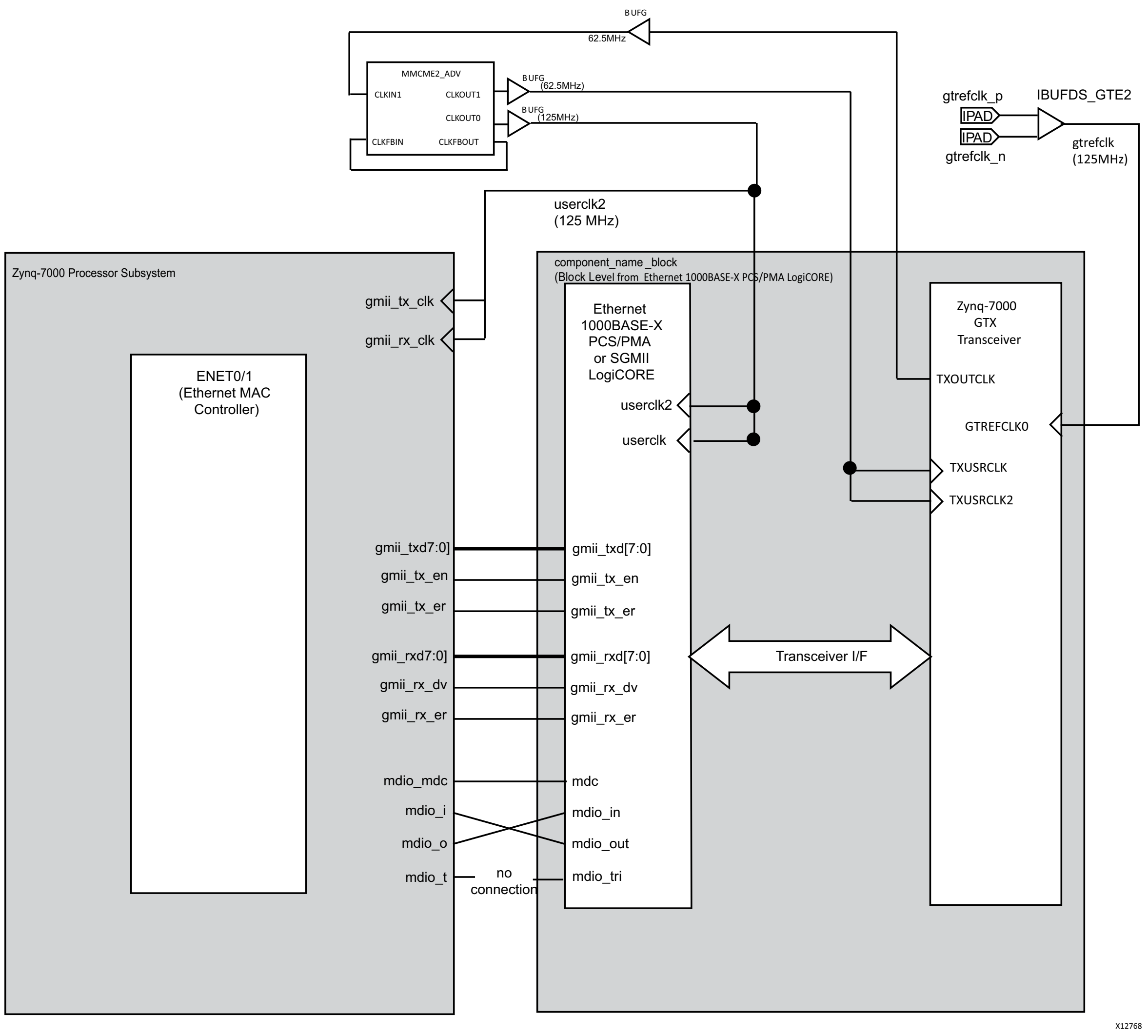


Figure 3-58: **ENET0/1 Extended to Include 1000BASE-X PCS/PMA Using Device Transceiver**

Integration of the Zynq-7000 Device PS ENET0/1 for Tri-speed SGMII Operation

Integration of the Zynq-7000 Device PS ENET0/1 Using Device Specific Transceivers

Figure 3-59 shows the connections and clock management logic required to interface the core (in SGMII Configuration and MAC mode with the 7 series FPGA transceiver) to the Zynq-7000 device PS ENET0/1.

Features of this configuration include:

- The SGMII Adaptation module, as provided in the example design for the core when generated to the SGMII standard and MAC mode, can be used to interface the two cores.
- The MDIO port can be connected up to that of the Zynq-7000 device ENET0/1, allowing the MAC to access the embedded configuration and status registers of the 1000BASE-X PCS/PMA or SGMII core.
- Because of the receive elastic buffer, the entire GMII (transmitter and receiver paths) 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 Zynq-7000 device PS ENET0/1 now operate in the same clock domain.

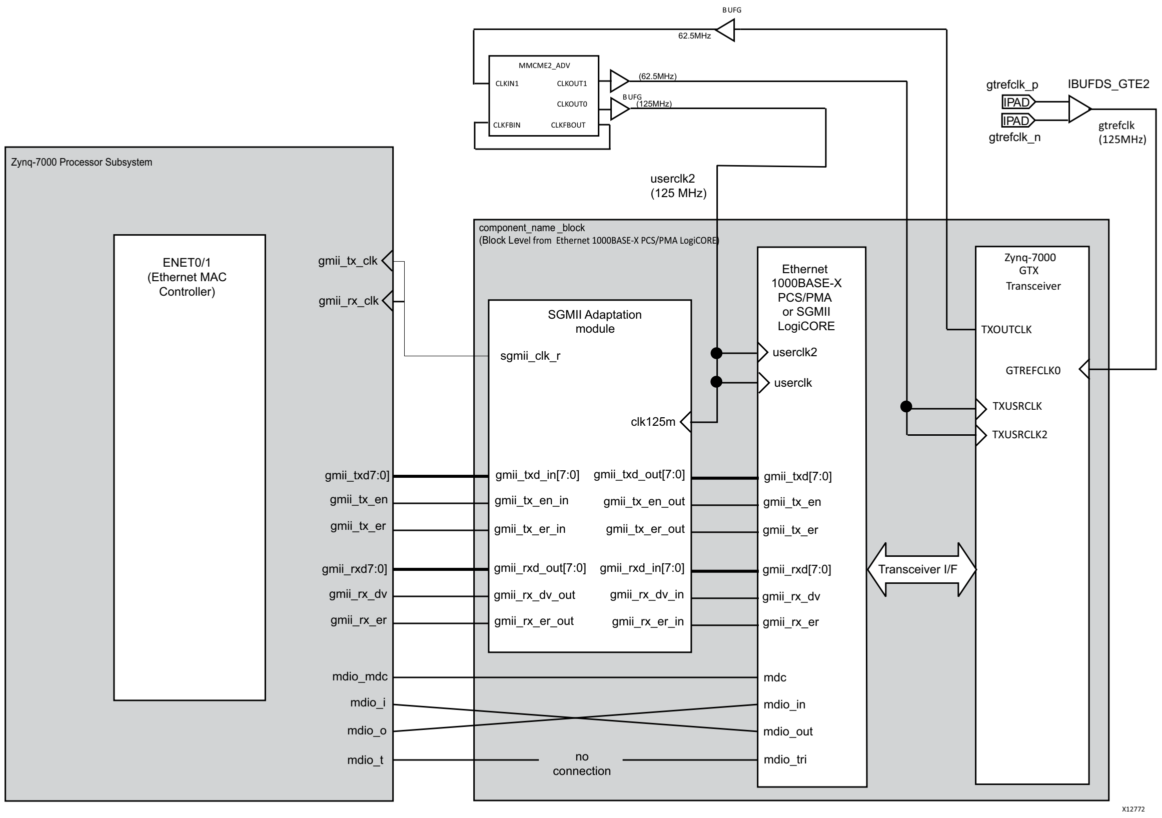


Figure 3-59: **Zynq-7000 Device ENET0/1 Extended to Include SGMII Using GTX Transceiver**

Integration of the TEMAC Core Using Sync SGMII over LVDS

Figure 3-60 shows the connections and clock management logic required to interface the core (in Sync SGMII over LVDS) to the Zynq-7000 device PS ENET0/1. The following conditions apply to each connected the Zynq-7000 device PS ENET0/1 and SGMII port pair:

- The SGMII Adaptation module, as provided in the example design for the core when generated to the SGMII standard, can be used to interface the two cores.

- The MDIO port can be connected up to that of the Zynq-7000 device PS ENET0/1, allowing the MAC to access the embedded configuration and status registers of the 1000BASE-X PCS/PMA or SGMII core.
- `clk125` is used as the 125 MHz reference clock for both cores, and the transmitter and receiver logic of the Zynq-7000 device PS ENET0/1 now operate in the same clock domain. This is the clock derived by MMCM and IBUFDS from differential reference clock.

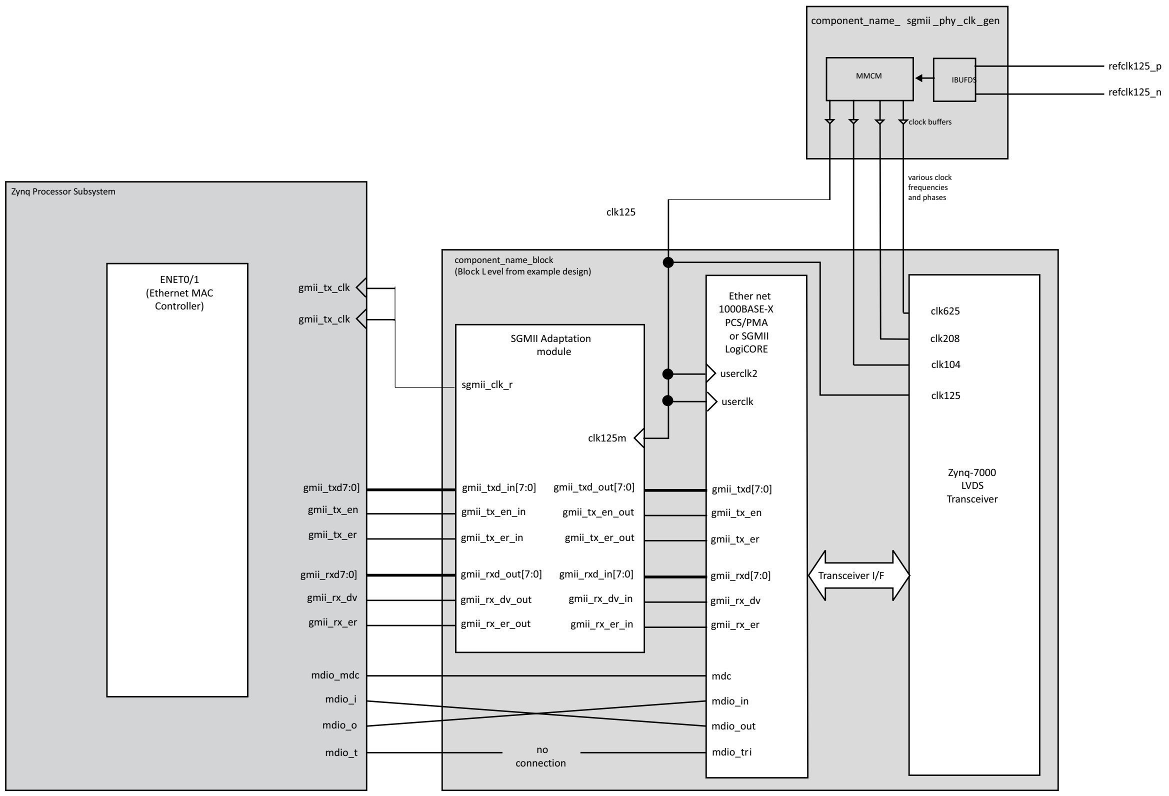


Figure 3-60: **Zynq-7000 Device ENET0/1 Extended to Include SGMII Using Zynq-7000 Device Synchronous LVDS**

Special Design Considerations

This section describes the unique design considerations associated with implementing the core.

Power Management

No power management considerations are recommended for the core when using it with the TBI. When using the core with a Zynq-7000, Virtex-7, Kintex-7 or Artix-7 device, the transceiver can 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.

Start-up Sequencing

IEEE 802.3-2008 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 in Chapter 2 or by the Configuration and Status Vectors in Chapter 2.

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 TBI Ports). This instructs the attached PMA SerDes device to enter loopback mode as shown in Figure 3-61.

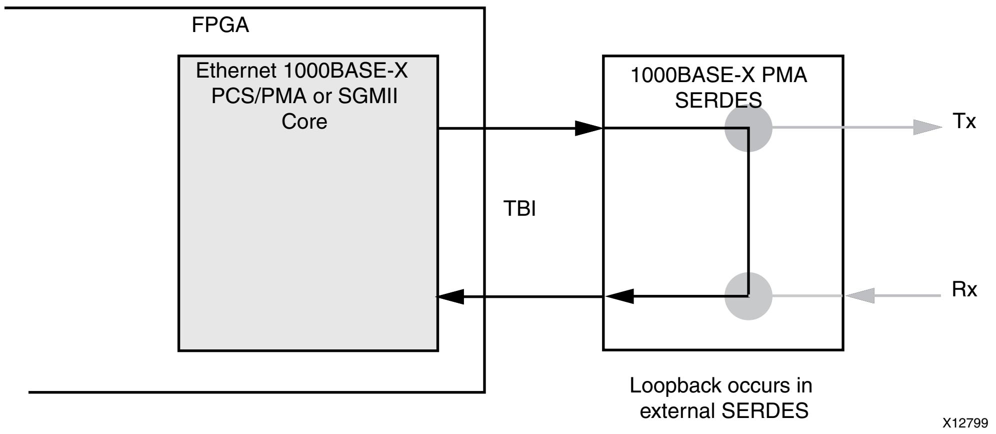

FPGA
Ethernet 1000BASE-X PCS/PMA or SGMII Core
TBI
1000BASE-X PMA SERDES
Tx
Rx
Loopback occurs in external SERDES
X12799

Figure 3-61: **Loopback Implementation Using the TBI**

Core with Transceiver

The loopback path is implemented in the core as shown in Figure 3-62. 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 (or LVDS transceiver) interface. When placed in loopback, the core creates a constant stream of Idle code groups that are transmitted through the serial or GTP transceiver in accordance with the IEEE 802.3-2008 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 (serial 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 can be directly driven by the user logic to place it in either parallel or serial loopback mode.

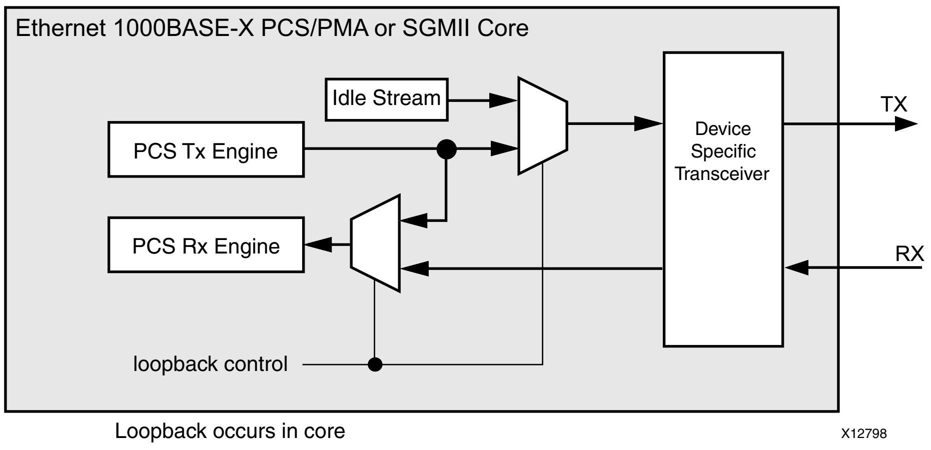


Figure 3-62: **Loopback Implementation When Using the Core with Device-Specific Transceivers**

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows in the IP Integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16]

Customizing and Generating the Core

The 1000BASE-X PCS/PMA or SGMII core is generated using the IP catalog. This section describes the Vivado IDE options used to generate and customize the core.

Impact of IP Integrator on Vivado IDE Options in the IP core

If you are customizing and generating the core in the Vivado IP Integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 13] for detailed information. Vivado IDE might auto-compute certain configuration values when validating or generating the design, as noted in this section. You can view the parameter value after successful completion of `validate_bd_design` command.

FREQ_HZ,PHASE and CLK_DOMAIN properties are propagated from `gt0_drpclk_in` to rest of the signals of DRP interface when DRP interfaces is selected (by enabling transceiver debug signals).

Vivado IDE for Zynq-7000 Devices

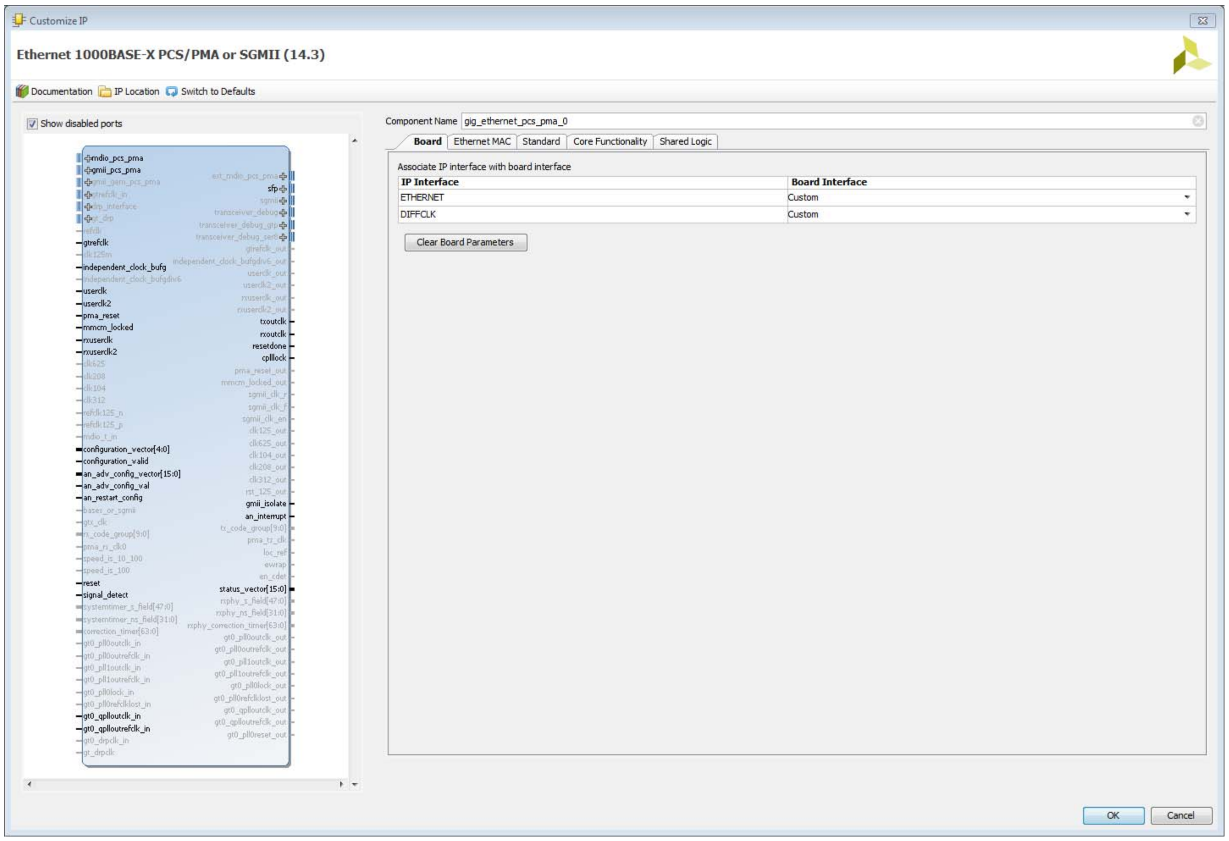
You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the tool bar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 15].

Note: Figures in this section are illustrations of the Vivado IDE. This layout might vary from the current version.

Figure 4-1 displays the option to enable the additional board support flow with the IP. This option is available only when the project is created by selecting a board from the list of boards available.



Ethernet 1000BASE-X PCS/PMA or SGMII (14.3)

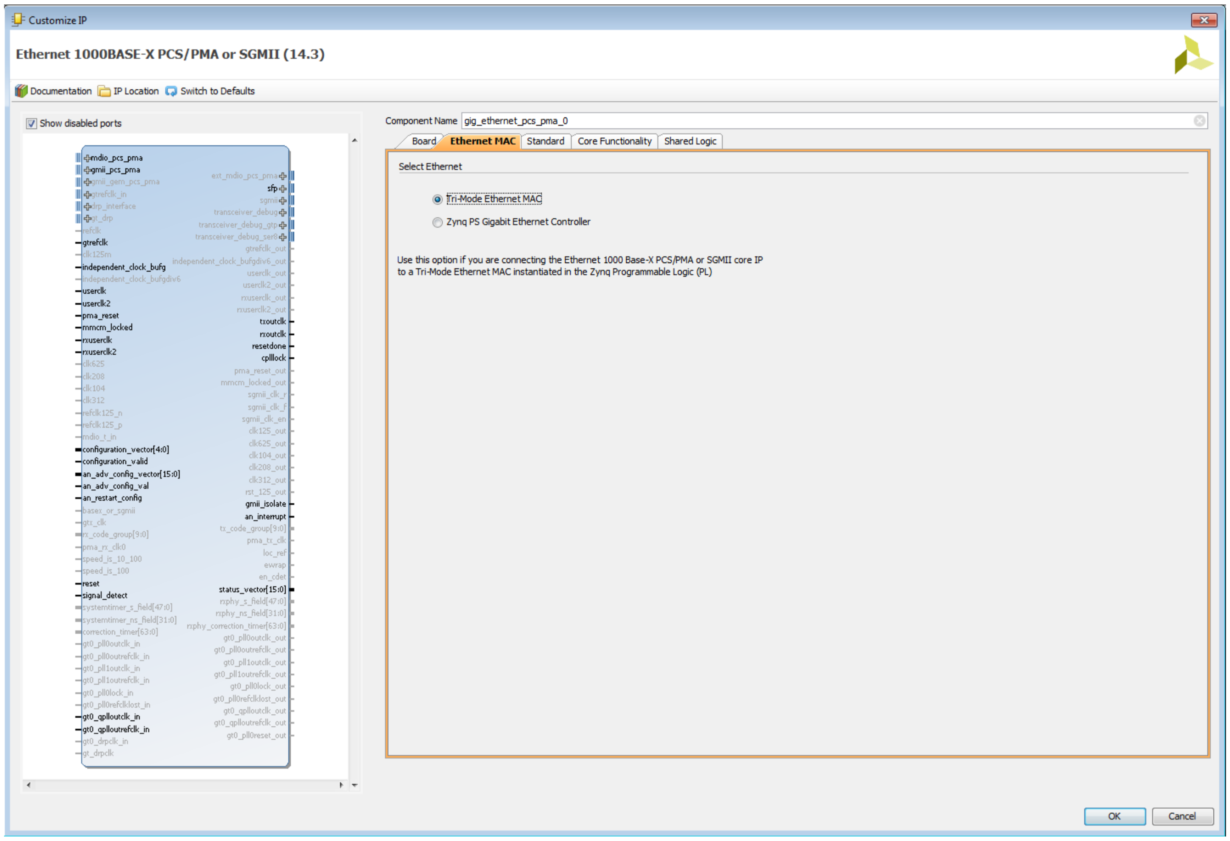
Component Name gig_ethernet_pcs_pma_0

Associate IP interface with board interface

IP Interface	Board Interface
ETHERNET	Custom
DIFFCLK	Custom

Figure 4-1: **Board Tab**

Figure 4-2 displays the Ethernet MAC selection screen. This screen is visible only for Zynq®-7000 devices.



Ethernet 1000BASE-X PCS/PMA or SGMII (14.3)

Component Name gig_ethernet_pcs_pma_0

Select Ethernet

[✓ Tri-Mode Ethernet MAC] Zynq PS Gigabit Ethernet Controller

Use this option if you are connecting the Ethernet 1000 Base-X PCS/PMA or SGMII core IP to a Tri-Mode Ethernet MAC instantiated in the Zynq Programmable Logic (PL)

Figure 4-2: **Core Customization Screen for Zynq-7000 Devices**

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 "_."

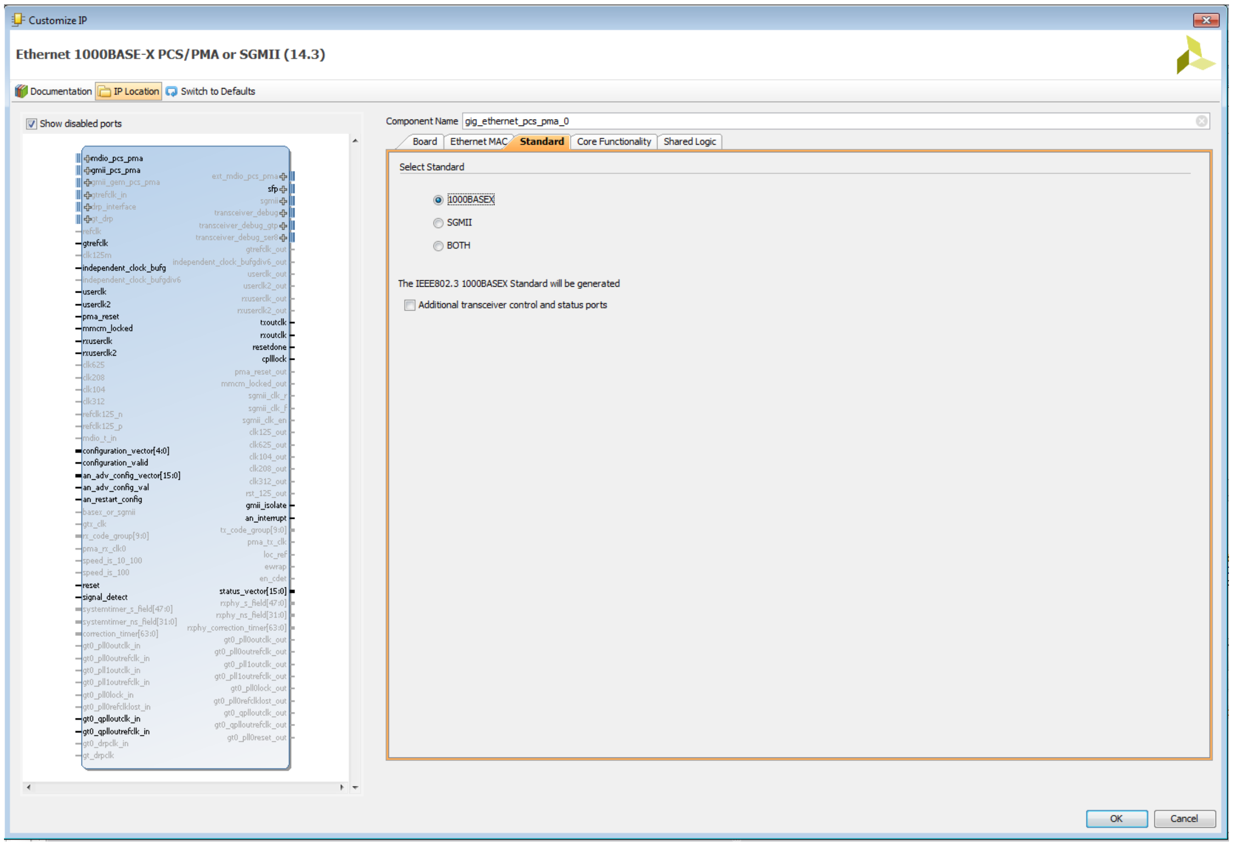
Select Ethernet

Select from the following Ethernet MACs.

- **Tri-Mode Ethernet MAC**. This option is used if the Ethernet 1000BASE-X PCS/PMA or SGMII core is interfaced with Tri-mode Ethernet MAC instantiated in the Zynq-7000 device Programmable Logic (PL)
- **Zynq-PS Gigabit Ethernet Controller**. This option is used if the Ethernet 1000BASE-X PCS/PMA or SGMII core is interfaced with Ethernet MAC (EMAC) present in the Zynq-7000 device processor subsystem (PS). Ethernet 1000BASE-X PCS/PMA or SGMII core and EMAC are connected through the EMIO interface.

Vivado IDE for UltraScale Architecture and 7 Series Devices

Figure 4-3 displays the Ethernet 1000BASE-X PCS/PMA or SGMII customization screen, used to set core parameters and options.



Ethernet 1000BASE-X PCS/PMA or SGMII (14.3)

[✓] Show disabled ports

Component Name: gig_ethernet_pcs_pma_0

Select Standard

[✓ 1000BASEX] SGMII BOTH

The IEEE802.3 1000BASEX Standard will be generated

[ ] Additional transceiver control and status ports

Figure 4-3: **Core Customization Screen**

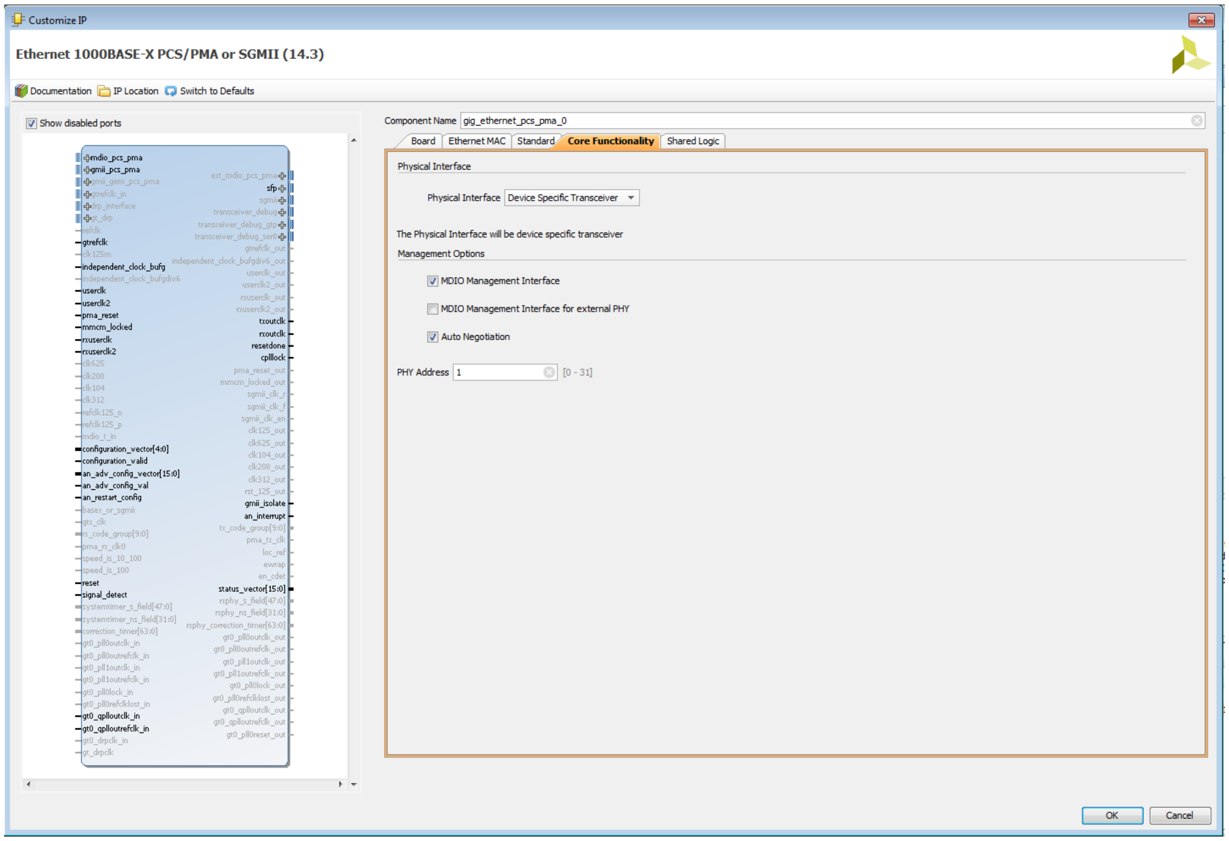
Select Standard

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 can be extended to include the 1000BASE-X Physical Medium Attachment (PMA) sublayer. Default setting.
- SGMII. Provides the functionality to provide a GMII to SGMII bridge, as defined in the *Serial-GMII Specification V1.7* (Cisco Systems, ENG-46158). SGMII can be used to replace GMII at a much lower pin count and for this reason is often favored by Printed Circuit Board (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 MDIO Management Interface.

Core Functionality

Figure 4-4 displays the Ethernet 1000BASE-X PCS/PMA or SGMII functionality screen.



Component Name gig_ethernet_pcs_pma_0

Physical Interface

Physical Interface: Device Specific Transceiver

The Physical Interface will be device specific transceiver

Management Options

- [✓] MDIO Management Interface
- [ ] MDIO Management Interface for external PHY
- [✓] Auto Negotiation

PHY Address 1 [0 - 31]

Figure 4-4: **1000BASE-X Standards Options Screen**

Physical Interface

Depending on the target architecture, up to three physical interface options are available for the core.

- **Device Specific Transceiver**. Uses a transceiver specific to the selected family to extend the 1000BASE-X functionality to include both PCS and PMA sub-layers. It is available for UltraScale™ architecture, Zynq-7000, Virtex®-7, Kintex-7 and Artix®-7 devices. For additional information, see 1000BASE-X with Transceivers.
- **Ten Bit Interface (TBI)**. Provides 1000BASE-X or SGMII functionality with a parallel TBI used to interface to an external Serializer/Deserializer (SerDes.) For more information, see The Ten-Bit Interface. This is available for Kintex-7 devices.
- **LVDS Serial**. Virtex-7 and Kintex-7 devices, -2 speed grade or faster for devices with HR Banks and -1 speed grade or faster for devices with HP Banks for performing the SGMII Standard. Artix-7 devices, -2 speed grade or higher, can fully support SGMII using standard LVDS SelectIO™ technology logic resources. Zynq-7000 devices, -2

speed grade or faster for XC7Z010/20 devices and -1 speed grade or faster for XC7Z030/45/100 devices, can fully support SGMII using standard LVDS SelectIO™ technology logic resources. This enables direct connection to external PHY devices without the use of an FPGA transceiver.

MDIO Management Interface

Select this option to include the MDIO management Interface to access the PCS Configuration registers. See MDIO Management Interface. An additional configuration vector interface is provided to write into management Registers 0 and 4. See Configuration and Status Vectors in Chapter 2.

Auto-Negotiation

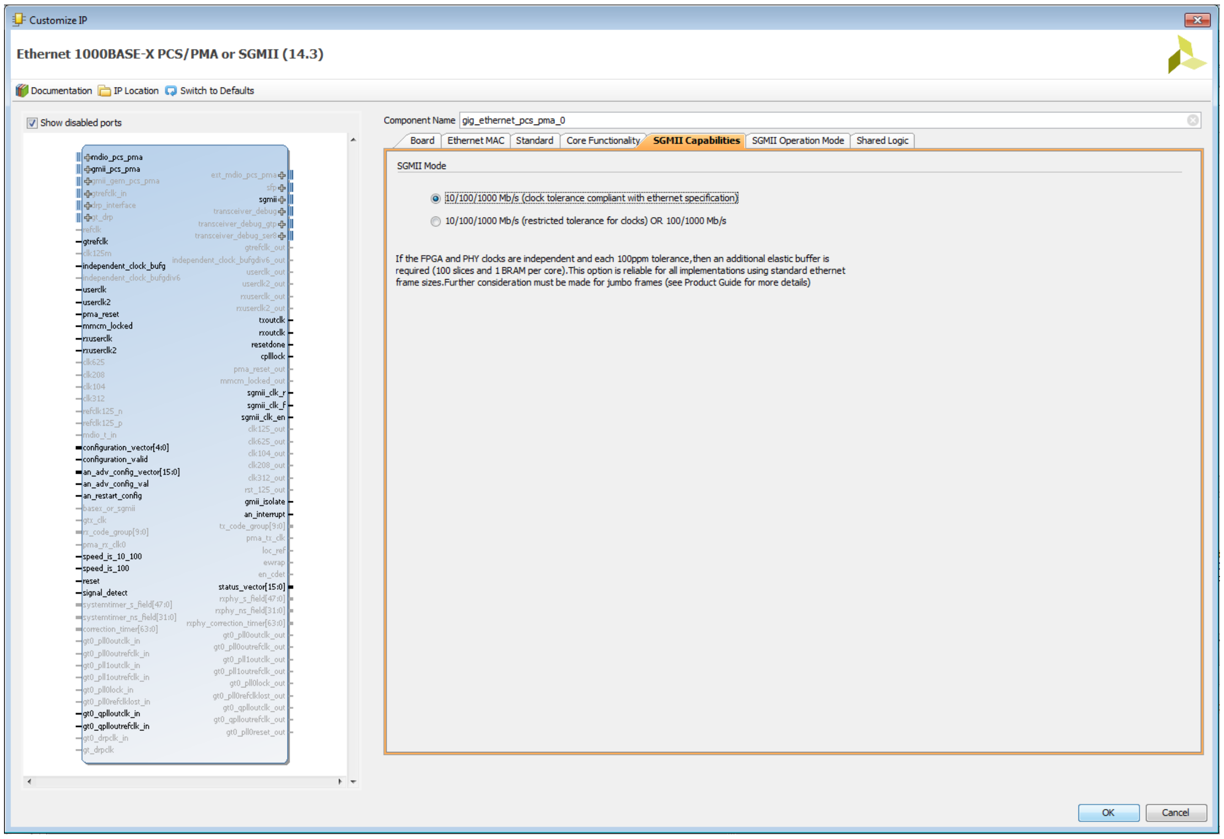
Select this option to include auto-negotiation functionality with the core. For more information, see Auto-Negotiation. The default is to include auto-negotiation.

PHY Address

PHY Address of the core. The values of PHY Address must be in range 0 to 31.

SGMII/Dynamic Switching Elastic Buffer Options

The SGMII/Dynamic 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.



Component Name: gig_ethernet_pcs_pma_0

SGMII Mode

[✓ 10/100/1000 Mb/s (clock tolerance compliant with ethernet specification)]
10/100/1000 Mb/s (restricted tolerance for clocks) OR 100/1000 Mb/s

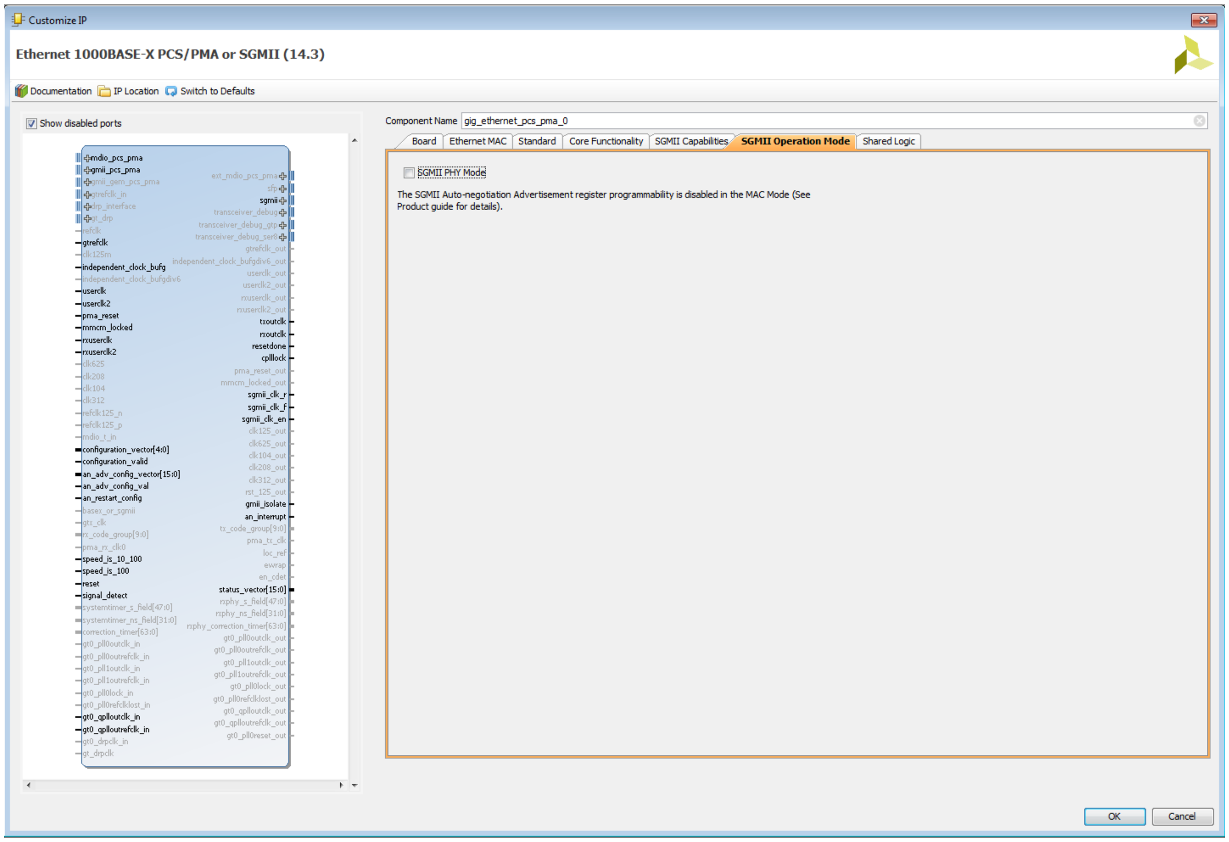
If the FPGA and PHY clocks are independent and each 100ppm tolerance,then an additional elastic buffer is required (100 slices and 1 BRAM per core).This option is reliable for all implementations using standard ethernet frame sizes.Further consideration must be made for jumbo frames (see Product Guide for more details)

Figure 4-5: **SGMII Dynamic Switching Options**

This screen lets you select the receive elastic buffer type to be used with the core. Before selecting this option, see Receive Elastic Buffer.

SGMII/Dynamic Switching Mode of Operation

The SGMII/Dynamic Switching Mode screen, used to customize the core, is only displayed if either SGMII or Both is selected in the Select Standard section of the initial customization screen.



Component Name gig_ethernet_pcs_pma_0

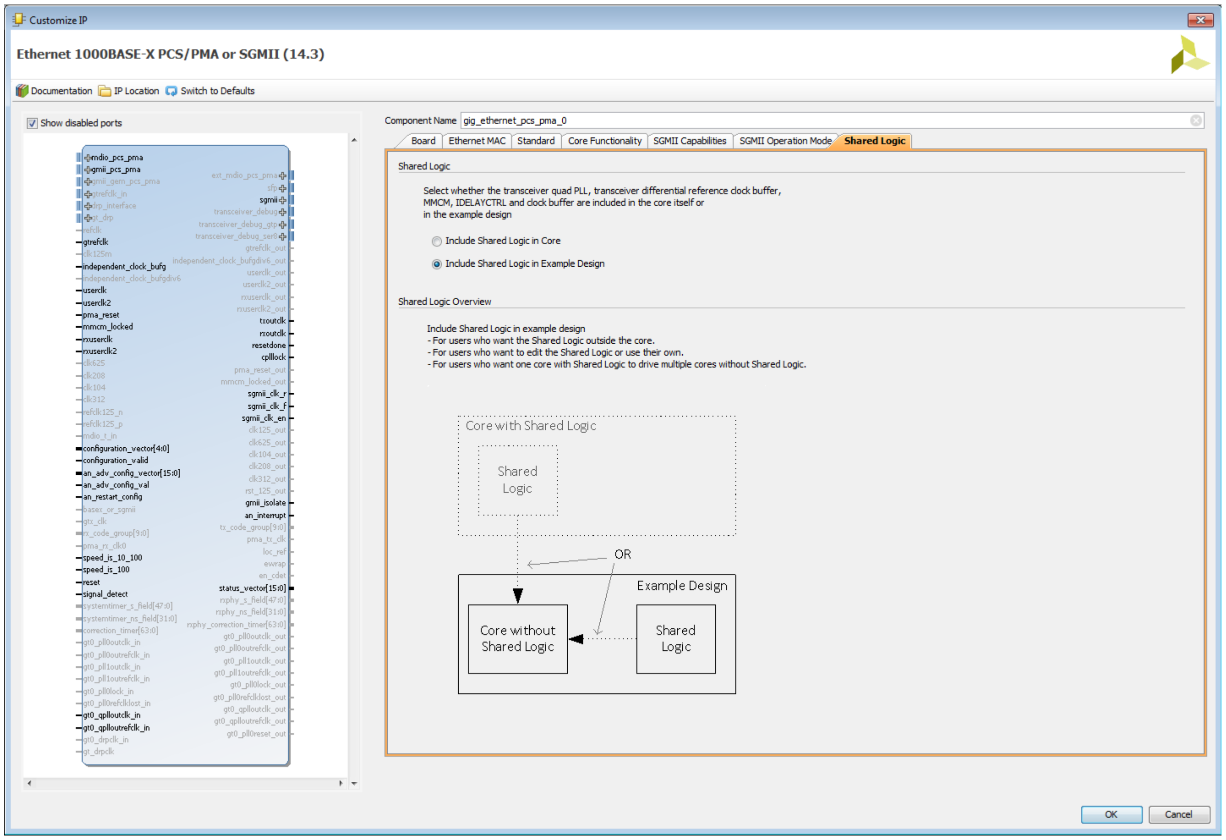
☐ SGMII PHY Mode

The SGMII Auto-negotiation Advertisement register programmability is disabled in the MAC Mode (See Product guide for details).

Figure 4-6: **SGMII Operation Mode Options**

This screen lets you select the core to operate in the PHY mode or Media Access Controller (MAC) mode.

Figure 4-7 displays the shared logic placement options available in the IP core.



Component Name: gig_ethernet_pcs_pma_0

Shared Logic

Select whether the transceiver quad PLL, transceiver differential reference clock buffer, MMCM, IDELAYCTRL and clock buffer are included in the core itself or in the example design

Include Shared Logic in Core

[✓ Include Shared Logic in Example Design]

Shared Logic Overview

Include Shared Logic in example design
- For users who want the Shared Logic outside the core.
- For users who want to edit the Shared Logic or use their own.
- For users who want one core with Shared Logic to drive multiple cores without Shared Logic.

Figure 4-7: **Shared Logic Placement Options**

Regeneration of 7 Series/Zynq-7000 Transceiver Files

Transceiver files for 7 series and Zynq-7000 devices can be generated using transceiver wizards by generating the GT Wizard IP using the following configuration:

- For BASE-X or SGMII without fabric elastic buffer mode: Select the **Protocol** as **gigabit ethernet CC** and generate the GT Wizard IP.
- For SGMII with fabric elastic buffer/ Dynamic Switching modes: Select **Protocol** as **gigabit ethernet noCC** and generate the GT Wizard IP.

The files delivered can include some or all of the following:

```
project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/transceiver/

    <component_name>_gtwizard_init.v[hd]

    <component_name>_tx_startup_fsm.v[hd]

    <component_name>_gtwizard_multi_gt.v[hd]

    <component_name>_rx_startup_fsm.v[hd]
```

```
<component_name>_gtwizard_gtrxreset_seq.v[hd]
<component_name>_gtwizard.v[hd]
<component_name>_gtwizard_gt.v[hd]
<component_name>_gtwizard_gtp_rxpmarst_seq_vhd.v[hd]
<component_name>_gtwizard_gtp_rxrate_seq.v[hd]
```

Output Generation

The Ethernet 1000BASE-X PCS/PMA or SGMII solution delivers files into several filegroups. By default the filegroups necessary for use of the core or opening the IP Example design are generated when the core is generated. If additional filegroups are required these can be selected using the generate option. The filegroups generated can be seen in the IP Sources tab of the Sources window where they are listed for each IP in the project. The filegroups available for the Ethernet 1000BASE-X PCS/PMA or SGMII solution are described in the following subsections.

For more detail, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

Examples

Includes all source required to be able to open and implement the IP example design project. That is, example design HDL and the example design xdc file.

Examples Simulation

Includes all source required to be able to simulate the IP example design project. This is the same list of HDL as the Examples filegroup with the addition of the demonstration test bench HDL.

Synthesis

Includes all synthesis sources required by the core. For the Ethernet 1000BASE-X PCS/PMA or SGMII solution this is a mix of both encrypted and unencrypted source. Only the unencrypted sources are visible.

Simulation

Includes all simulation sources required by the core. Simulation of the Ethernet 1000BASE-X PCS/PMA or SGMII solution at the core level is not supported without the addition of a test bench (not supplied). Simulation of the example design is supported.

When EXAMPLE_SIMULATION is set, the link timer for Auto-Negotiation is pre-loaded with a smaller value.

For SGMII over LVDS, setting EXAMPLE_SIMULATION pre-loads the eye_mon_wait_time counter to a lower value to decrease the simulation time.

Note: EXAMPLE_SIMULATION generic is provided in all modes to reduce simulation time. In simulation, the value of EXAMPLE_SIMULATION should be 1. In implementation, the value of EXAMPLE_SIMULATION should be 0. To change the EXAMPLE_SIMULATION attribute you need to give following command before the generation of output products:

set_property CONFIG.EXAMPLE_SIMULATION {1} [get_ips <component_name>**]**

IMPORTANT: *EXAMPLE_SIMULATION generic should be set to 1 only for simulation; for synthesis this should be reset to 0.*

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite. It defines the constraint requirements of the Ethernet 1000BASE-X PCS/PMA or SGMII solution.

Required Constraints

The Ethernet 1000BASE-X PCS/PMA or SGMII solution is provided with a core level XDC file. This provides constraints for the core that are expected to be applied in all instantiations of the core. This XDC file, named <component name>.xdc, can be found in the IP Sources tab of the Sources window in the Synthesis file group.

An example XDC is also provided with the HDL example design to provide the board level constraints. This is specific to the example design and, as such, is only expected to be used as a template for the user design. See Chapter 5, Example Design. This XDC file, named `<component name>_example_design.xdc`, is found in the IP Sources tab of the Sources window in the Examples file group.

The core level XDC file inherits some constraints from the example design XDC file. In any system it is expected that you will also provide an XDC file to constrain the logic in which the Ethernet 1000BASE-X PCS/PMA or SGMII solution is instantiated.

Device, Package, and Speed Grade Selections

The core can be implemented in UltraScale architecture, Zynq-7000, Virtex-7, Kintex-7 and Artix-7 devices. The modes supported for specific devices are described in Table 2-1.

Clock Frequencies

The Ethernet 1000BASE-X PCS/PMA or SGMII solution has a variable number of clocks with the precise number required being dependent upon the specific parameterization. As the core targets various transceiver options, there are associated clock frequency requirements.

Table 4-1: **Clock Frequencies**

Clock Name	Parametrization	Frequency Requirement
gtrefclk	Present if serial transceiver is used	125 MHz
txoutclk	Present if serial transceiver is used	62.5 or 125 MHz depending on serial transceiver used
userclk	Present if serial transceiver is used	62.5 or 125 MHz depending on serial transceiver used
userclk2	Present if serial transceiver is used	125 MHz
sgmii_clk	Present in SGMII Mode	1.25 MHz, 12.5 MHz or 125 MHz
gtx_clk	Present in TBI Mode	125 MHz
pma_tx_clk	Present in TBI Mode	125 MHz
pma_rx_clk	Present in TBI Mode	125 MHz
clk625	Present in LVDS Mode	625 MHz
clk208	Present in LVDS Mode	208 MHz
clk104	Present in LVDS Mode	104 MHz

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

There are no specific I/O standard/placement requirements on most interfaces. Depending upon the device family, part and package chosen there are two types of I/O available for use. HP I/O is intended for support of high-speed interfaces and as such is limited to 1.8 V support. HP I/O support both Input and Output Delays components. HR I/O is intended for interfaces with higher voltage requirements and has a more limited supported frequency range. HR I/O only supports Input Delay components.

Both MII and GMII are 3.3 V standards. However the majority of PHYs are multi-standard and operate at either 2.5 V or 3.3 V and this is also true of the PHYs selected for Xilinx development boards. This means that for most applications the physical interfaces are restricted to either using HR I/O, where available, or HP I/O with an external voltage converter to translate between 1.8 V and the minimum level required by the PHY of 2.5 V.

IMPORTANT: *For any board design it is very important to identify which type of I/O is available/being used.*

In most of the applications the GMII interface of the core is interfaced to Xilinx TEMAC core in the FPGA, which means that no IP standard/placement is required for that interface.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 16].

All simulation sources are included that are required by the core. Simulation of PCS-PMA at the core level is not supported without the addition of a test bench (not supplied). Simulation of the example design is supported.

Note: EXAMPLE_SIMULATION generic is provided in all modes to reduce simulation time. In simulation, the value of EXAMPLE_SIMULATION should be 1. In implementation, the value of EXAMPLE_SIMULATION should be 0. To change the EXAMPLE_SIMULATION attribute you need to run the following command before the generation of output products:

```
set_property CONFIG.EXAMPLE_SIMULATION {1} [get_ips <component_name>]
```

IMPORTANT: *EXAMPLE_SIMULATION generic should be set to 1 only for simulation; for synthesis this should be reset to 0.*

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 14].

All synthesis sources are included that are required by the core. This is a mix of both encrypted and unencrypted source. Only the unencrypted sources are visible and optionally editable by using the **Unlink IP Vivado** option.

Example Design

The example designs provided with the core are described in detail in this chapter. For information about the Demonstration Test Bench, see Chapter 6, Test Bench.

For all the example designs described in this chapter the file locations for the top level and block level VHDL and Verilog example designs are as follows. The contents of the files, which are different, depending on the core configuration, are described in the respective sections.

The following files describe the top level for the core:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
example_design/<component_name>_example_design.v[hd]
```

The following files describe the block level example design for the core:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
example_design/<component_name>.v[hd]
```

1000BASE-X with Transceiver Example Design

Figure 5-1 shows the example design for the core using a device-specific transceiver (UltraScale™ architecture, 7 series or Zynq®-7000).

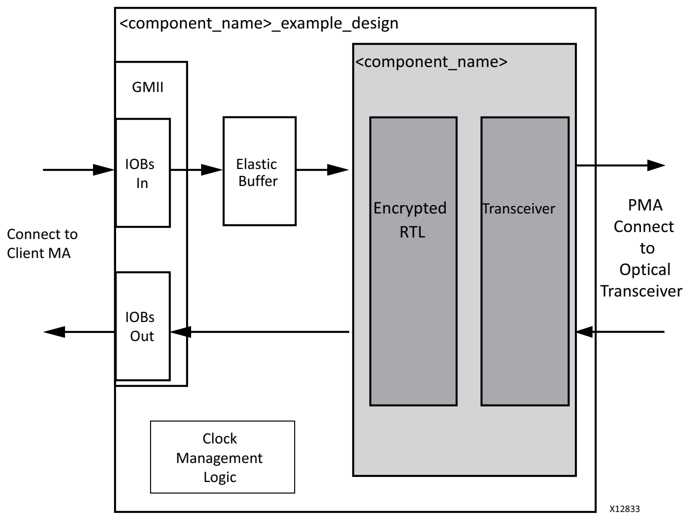


Figure 5-1: **Core Example Design HDL Using a Device-Specific Transceiver**

The top level of the example design (`<component_name>_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 HDL
- Instantiates shared logic if shared logic in the example design is selected (see Shared Logic for more information)
- A transmitter elastic buffer
- GMII interface logic, including IOB instances

Note: The optional transceiver control and status ports are not shown here. These ports have been brought up to the <component_name> module level.

Transmitter Elastic Buffer

The transmitter elastic buffer is described in the following files:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
example_design/<component_name>_tx_elastic_buffer.v[hd]
```

When the GMII is used externally (as in this example design), the GMII transmit signals (inputs to the core from a remote MAC at the other end of the interface) are synchronous to a clock that is likely to be derived from a different clock source to the core. For this reason, GMII transmit signals must be transferred into the core main clock domain before they can be used by the core and device-specific transceiver. This is achieved with the TX elastic buffer, an asynchronous FIFO implemented in distributed RAM. The operation of the elastic buffer is to attempt to maintain a constant occupancy by inserting or removing any idle sequences. This causes no corruption to the frames of data.

When the GMII is used as an internal interface, it is expected that the entire interface will be synchronous to a single clock domain, and the transmitter elastic buffer should be discarded.

SGMII/Dynamic Switching Using a Transceiver Example Design

Figure 5-2 shows an example design for top level HDL for the core in SGMII (or dynamic switching) mode using a device-specific transceiver (UltraScale architecture, 7 series, or Zynq-7000 devices).

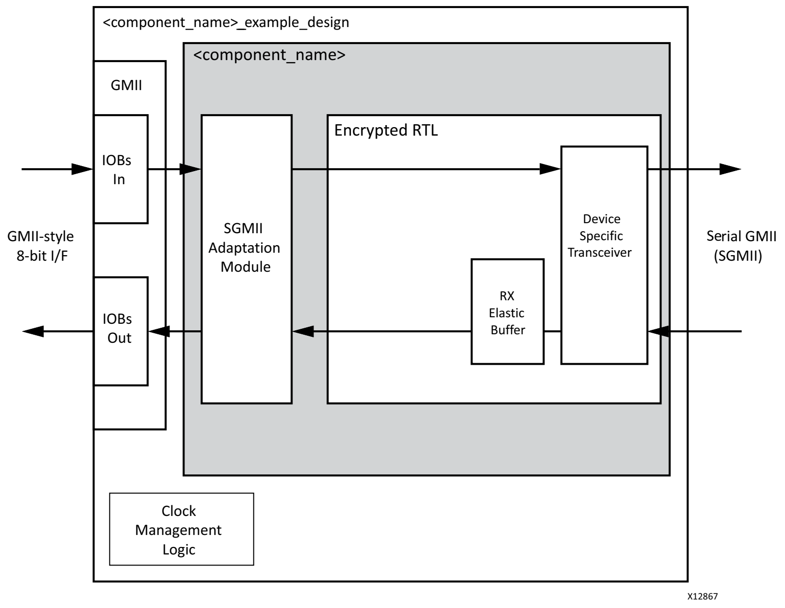


Figure 5-2: **Example Design HDL for the Core in SGMII Mode Using a Device-Specific Transceiver**

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 HDL
- Instantiates shared logic if shared logic in the example design is selected (see Shared Logic for more information)
- Clock management logic for the core and the device-specific transceiver, including DCM (if required) and Global Clock Buffer instances
- External GMII logic, including IOB and DDR register instances, where required

SGMII over LVDS Example Design

Figure 3-18 shows the HDL example design that is provided for the SGMII over Zynq-7000 and 7 series device LVDS implementation. The top level of the example design creates a specific example that can be simulated, synthesized and implemented.

The core netlist in this implementation remains identical to that of 1000BASE-X PCS/PMA or SGMII Using a Device-Specific Transceiver.

Top Level

The example design HDL top level performs the following:

- Instantiates the block level HDL
- Instantiates shared logic if shared logic in the example design is selected (see Shared Logic for more information)
- External GMII logic, including IOB and DDR register instances, where required. This module adds I/O logic to the GMII of the SGMII ports. This is included only to create a standalone design that can be implemented in an FPGA and simulated in both functional and timing simulation for the purposes of providing a complete SGMII design example. Discard this level of hierarchy and instantiate the block level in your own design.

1000BASE-X with TBI Example Design

Figure 5-3 shows the example design for a top level HDL with a 10-bit interface (TBI).

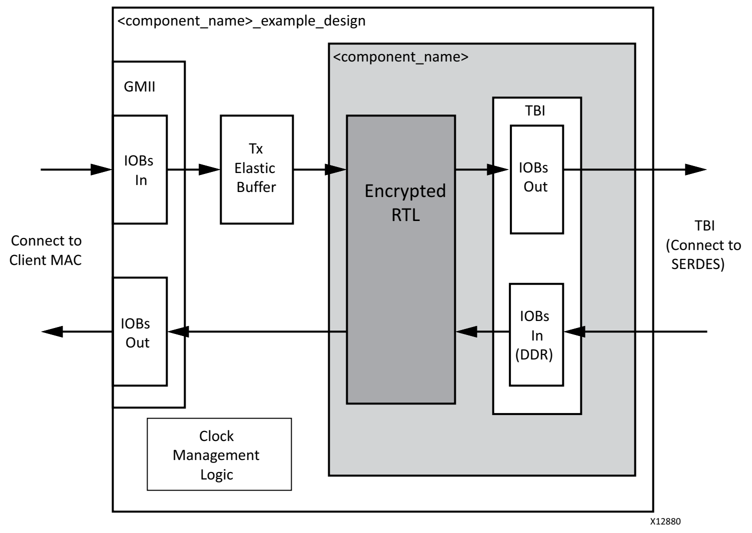


Figure 5-3: **Example Design HDL for the Ethernet 1000BASE-X PCS with TBI**

Top Level

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 HDL
- Clock management logic, including DCM and Global Clock Buffer instances, where required
- A transmitter elastic buffer
- GMII interface logic, including IOB and DDR registers instances, where required

The example design HDL top level connects the GMII of block level to external IOBs. This allows the functionality of the core to be demonstrated using a simulation package as described in this guide. The example design can also be synthesized and placed on a suitable board and demonstrated in hardware, if required.

Transmitter Elastic Buffer

The transmitter elastic buffer is described in the following files:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/
<component_name>/example_design/<component_name>_tx_elastic_buffer.v[hd]
```

When the GMII is used externally (as in this example design), the GMII transmit signals (inputs to the core from a remote Ethernet MAC at the other end of the interface) are synchronous to a clock, which is likely to be derived from a different clock source to the core. For this reason, GMII transmit signals must be transferred into the core main clock domain before they can be used by the core. This is achieved with the TX elastic buffer, an asynchronous FIFO implemented in distributed RAM. The operation of the elastic buffer is to attempt to maintain a constant occupancy by inserting or removing Idle sequences as necessary. This causes no corruption to the frames of data.

When the GMII is used as an internal interface, it is expected that the entire interface is synchronous to a single clock domain, and the TX elastic buffer should be discarded.

SGMII/Dynamic Switching with TBI Example Design

Figure 5-4 shows an example design for the top level HDL for the core in SGMII (or dynamic switching) mode with the TBI.

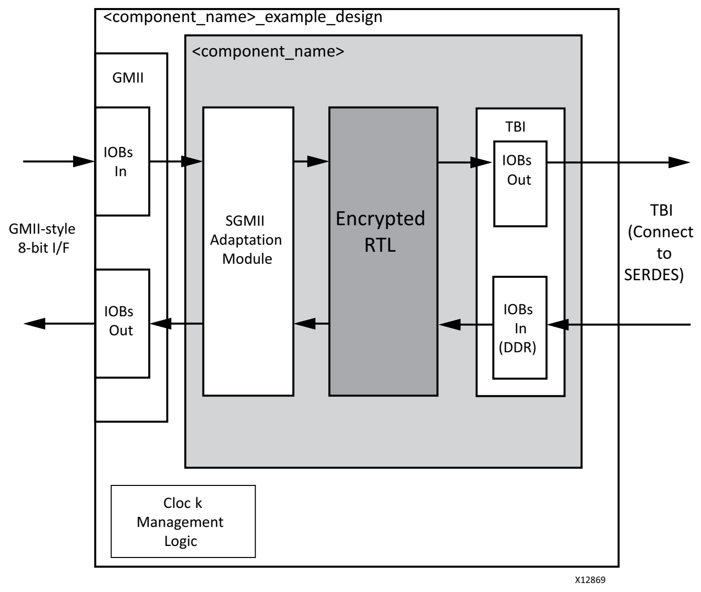


Figure 5-4: **Example Design HDL for the Core in SGMII Mode with TBI**

Top Level

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 HDL
- An instance of the SGMII block level
- Derives the clock management logic, including DCM and Global Clock Buffer instances, where required
- Implements external GMII logic, including IOB and DDR register instances, where required

The example design HDL top level connects the GMII of the block level to external IOBs. This allows the functionality of the core to be demonstrated using a simulation package.

Test Bench

This chapter contains information about the demonstration test bench provided in the Vivado® Design Suite.

Figure 6-1 shows the demonstration test bench for the core using the TBI, a device-specific transceiver or the LVDS transceiver. The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core.

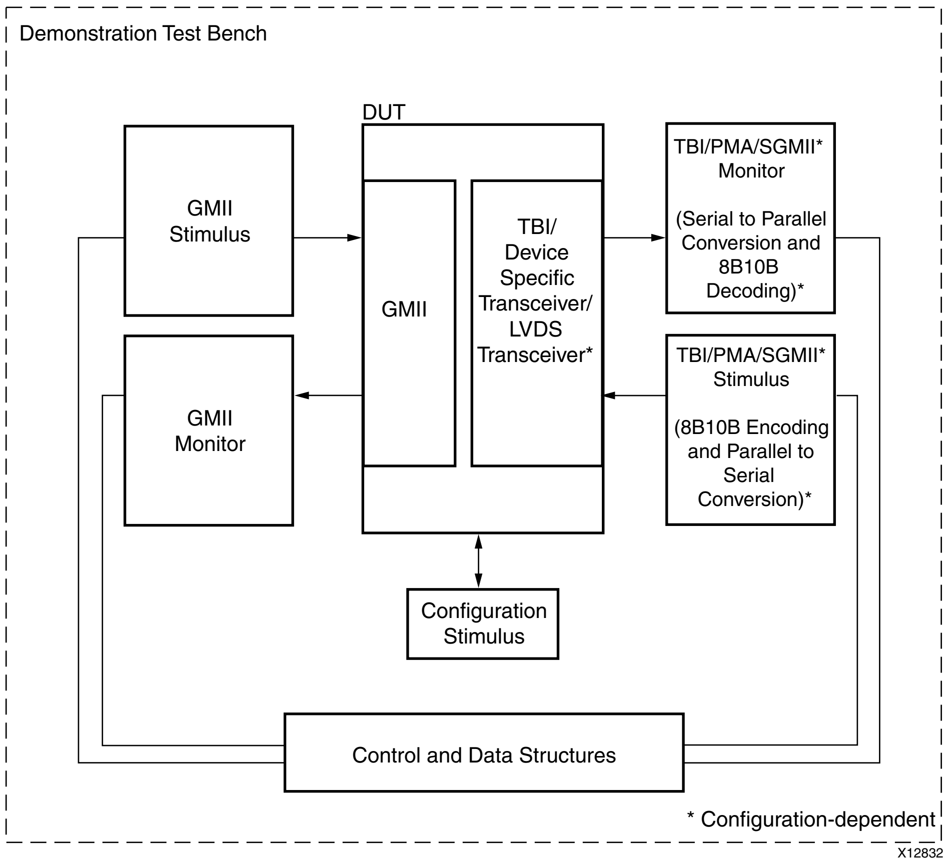

Demonstration Test Bench
DUT
GMII Stimulus
GMII Monitor
GMII
TBI/ Device Specific Transceiver/ LVDS Transceiver*
TBI/PMA/SGMII* Monitor
(Serial to Parallel Conversion and 8B10B Decoding)*
TBI/PMA/SGMII* Stimulus
(8B10B Encoding and Parallel to Serial Conversion)*
Configuration Stimulus
Control and Data Structures
* Configuration-dependent
X12832

Figure 6-1: **Demonstration Test Bench**

The top level test bench entity instantiates the example design for the core, which is the Device Under Test (DUT). A stimulus block is also instantiated and clocks, resets, and test bench semaphores are created. The following files describe the top level of the demonstration test bench:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
simulation/demo_tb.v[hd]
```

The stimulus block entity, instantiated from within the test bench top level, creates the Ethernet stimulus in the form of four Ethernet frames, which are injected into the GMII and PHY interfaces of the DUT. The output from the DUT is also monitored for errors. The following files describe the stimulus block of the demonstration test bench.

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
simulation/stimulus_tb.v[hd]
```

Together, the top level test bench file and the stimulus block combine to provide the full test bench functionality, described in the sections that follow.

Core with MDIO Interface

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- The core is configured through the MDIO interface by injecting an MDIO frame into the example design. This disables auto-negotiation (if present) and takes the core out of the Isolate state.
- Four frames are injected into the GMII transmitter by the GMII stimulus block.
    - the first frame is a minimum length frame
    - the second frame is a type frame
    - the third frame is an errored frame
    - the fourth frame is a padded frame
- The data at the TBI/transceiver transmitter interface is converted to 10-bit parallel data (for the transceiver only, not when the TBI is used), then 8B/10B decoded. The resulting frames are checked by the TBI/PMA/SGMII Monitor against the stimulus frames injected into the GMII transmitter to ensure data integrity.
- The same four frames are generated by the TBI/PMA/SGMII Stimulus block. These are 8B/10B encoded, converted to serial data (for the transceivers only, not when the TBI is used), and injected into the TBI/transceiver receiver interface.
- Data frames received at the GMII receiver are checked by the GMII Monitor against the stimulus frames injected into the TBI/transceiver receiver to ensure data integrity.

Core without MDIO Interface

The demonstration test bench performs the following tasks:

- Input clock signals are generated.

- A reset is applied to the example design.
- The core is configured using the Configuration Vector to take the core out of the Isolate state.
- Four frames are injected into the GMII transmitter by the GMII stimulus block.
    - the first frame is a minimum length frame
    - the second frame is a type frame
    - the third frame is an errored frame
    - the fourth frame is a padded frame
- The data at the TBI/transceiver transmitter interface is converted to 10-bit parallel data (for the transceiver only, not when the TBI is used), then 8B/10B decoded. The resultant frames are checked by the TBI/PMA/SGMII Monitor against the stimulus frames injected into the GMII transmitter to ensure data integrity.
- The same four frames are generated by the TBI/PMA/SGMII Stimulus block. These are 8B/10B encoded, converted to serial data (for the transceivers only, not when the TBI is used) and injected into the TBI/transceiver receiver interface.
- Data frames received at the GMII receiver are checked by the GMII Monitor against the stimulus frames injected into the TBI/transceiver receiver to ensure data integrity.

Customizing the Test Bench

This section provides information about making modifications to the demonstration test bench files.

Changing Frame Data

You can change the contents of the four frames used by the demonstration test bench by changing the *data* and *valid* fields for each frame defined in the stimulus block. Frames can be added by defining a new frame of data. Any modified frames are automatically updated in both stimulus and monitor functions.

Changing Frame Error Status

Errors can be inserted into any of the predefined frames in any position by setting the *error* field to 1 in any column of that frame. Injected errors are automatically updated in both stimulus and monitor functions.

Changing the Core Configuration

The configuration of the core used in the demonstration test bench can be altered.

CAUTION! *Certain configurations of the core can cause the test bench to fail or cause processes to run indefinitely. For example, the demonstration test bench does not auto-negotiate with the design example. Determine the configurations that can safely be used with the test bench.*

If the MDIO interface option has been selected, the core can be reconfigured by editing the injected MDIO frame in the demonstration test bench top level. Management Registers 0 and 4 can additionally be written though `configuration_vector[4:0]` and `an_adv_config_vector[15:0]` interface signals respectively. If the MDIO interface option has not been selected, the core can be reconfigured by modifying the configuration vector directly.

Changing the Operational Speed

SGMII can be used to carry Ethernet traffic at 10 Mb/s, 100 Mb/s or 1 Gb/s. By default, the demonstration test bench is configured to operate at 1 Gb/s. The speed of both the example design and test bench can be set to the desired operational speed by editing the following settings, recompiling the test bench, then running the simulation again.

1 Gb/s Operation

```
set speed_is_10_100 to logic 0
```

100 Mb/s Operation

```
set speed_is_10_100 to logic 1
set speed_is_100 to logic 1
```

10 Mb/s Operation

```
set speed_is_10_100 to logic 1
set speed_is_100 to logic 0
```

Verification, Compliance, and Interoperability

This appendix provides details about how this IP core was tested for compliance. The 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

UltraScale™ architecture, Zynq®-7000, and 7 series device designs incorporating a device-specific transceiver require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. For VHDL simulation, a mixed hardware description language (HDL) license is required.

For UltraScale architecture devices a mixed simulation license is required.

Hardware Testing

The core has been tested on many hardware test platforms at Xilinx to represent different parameterizations, including the following:

- The core, used with a device-specific transceiver and using the 1000BASE-X standard, has been tested with the Xilinx Tri-Mode Ethernet MAC core, which follows the architecture shown in Figure 1-1. A test platform was built around these cores, including a backend FIFO capable of performing a simple ping function, and a test pattern generator. Software running on the embedded PowerPC® processor provided 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, used with a device-specific transceiver and using the SGMII standard, has been tested with the LogiCORE™ IP Tri-Mode Ethernet MAC core. This was connected to an external PHY capable of performing 10BASE-T, 100BASE-T, and 1000BASE-T, and the system was tested at all three speeds. This follows the architecture shown in Figure 1-2 and also includes the PowerPC-based processor test platform described previously.

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information on migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 17].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Port Changes from v14.1 to v14.2

Ports Added

The ports in Table B-1 were added to the core (non-shared logic).

Table B-1: **Ports Added**

In/Out	Port Name and Width	Description	What to do
Output	ext_mdio_t	This is the MDIO 3-state control signal for the IOBUF to drive the external MDIO interface.	Enabled only when selected through Vivado IDE or enabled during IP generation in batch mode through the command line configuration option Ext_Management_Interface {true}.
Output	ext_mdio_o	This is the MDIO output signal for the IOBUF to drive the external MDIO interface.	Enabled only when selected through Vivado IDE or enabled during IP generation in batch mode through the command line configuration option Ext_Management_Interface {true}.
Input	ext_mdio_i	This is the MDIO input signal for the IOBUF to drive the external MDIO interface.	Enabled only when selected through Vivado IDE or enabled during IP generation in batch mode through the command line configuration option Ext_Management_Interface {true}.
Output	ext_mdc	This is the mdc to drive the external MDIO interface.	Enabled only when selected through Vivado IDE or enabled during IP generation in batch mode through the command line configuration option Ext_Management_Interface {true}.
Input	mdio_t_in	This is the MDIO 3-state input that should be driven through the TEMAC or GEM.	Enabled only when selected through GUI or enabled during IP generation in batch mode through the command line configuration option Ext_Management_Interface {true}.
Input	clk312	312 MHz clock	Enabled only in SGMII over LVDS mode for UltraScale™ architecture devices. See SGMII over LVDS for more details.
Input	Independent_ clock_bufgdiv6	50 MHz clock	This is a new clock added that is required by the UltraScale architecture gtwizard. The frequency of this clock must 50 MHz. This clock is visible only when the transceiver control and status ports are disabled and shared logic is present in the example design.
Output	Independent_clock_ bufgdiv6_out	50 MHz clock output	This clock is visible only when the transceiver control and status ports are disabled and shared logic is present in the core. This is an independent clock that is divided by 6. The independent clock must 300 MHz in this case.

Ports Functionality Changed

The functionality of the ports in Table B-2 were changed or corrected as described.

Table B-2: **Ports Functionality Changed**

In/Out	Port Name and Width	Description	What has changed
Output	rxuserclk_out	rxuserclk output	This port is enabled when shared logic is in the core and SGMII/BASE-X modes are selected. It was earlier derived from `txoutclk` in the case of BASE-X mode; now it is the same as `rxoutclk` in the BASE-X and SGMII modes.
Output	rxuserclk2_out	rxuserclk2 output	This port is enabled when shared logic is in the core and SGMII/BASE-X modes are selected. It was earlier derived from `txoutclk` in the case of BASE-X mode; now it is the same as `rxoutclk` in BASE-X and SGMII modes.
Output	resetdone	Reset done indication from the core	Previously this indication was not correct and used to indicate reset done much before completion of the transceiver reset sequence. This has been corrected and indicates the actual reset done of the TX and RX reset sequences.

Port Changes from v14.0 to v14.1

Ports Added

The ports in Table B-3 and Table B-4 were added to the Transceiver Debug feature of the core.

Table B-3: **Ports Added for 7 Series and Zynq Devices**

Signal	Direction	Description
gt0_txpmareset_in	Input	GT TX-PMA Reset
gt0_txpcsreset_in	Input	GT TX-PCS Reset
gt0_rxpmareset_in	Input	GT RX-PMA Reset
gt0_rxpcsreset_in	Input	GT RX-PCS Reset
gt0_rxbufreset_in	Input	GT receive elastic buffer Reset
gt0_rxpmaresetdone_out	Output	GT PMA resetdone indication
gt0_txbufstatus_out[1:0]	Output	GT TX Buffer status
gt0_rxbufstatus_out[2:0]	Output	GT RX Buffer status
gt0_dmonitorout_out[16:0]	Output	GT Status
gt0_rxlpmreset_in	Input	RX LPM reset. Valid only for GTP.
gt0_rxlpmhfoverden_in	Input	RX LPM-HF override enable. Valid only for GTP.

Table B-4: **Ports Added for UltraScale Architecture Devices**

Signal	Direction	Description
gt_drp_addr_in[8:0]	Input	DRP address bus
gt_drpi_in[15:0]	Input	Data bus for writing configuration data to the transceiver.
gt_drpo_out[15:0]	Output	Data bus for reading configuration data from the transceiver.
gt_drprdy_out	Output	Indicates operation is complete for write operations and data is valid for read operations.
gt_drpwe_in	Input	DRP write enable.
gt_drpclk_in	Input	DRP Clock
gt_rxcommadet_out	Output	
gt_txdiffctrl_in[3:0]	Input	GT TX Driver
gt_txpostcursor_in[4:0]	Input	
gt_txprecursor_in[4:0]	Input	
gt_txpolarity_in	Input	GT Polarity
gt_rxpolarity_in	Input	
gt_txprbssel_in[2:0]	Input	GT PRBS
gt_txprbsforceerr_in	Input	
gt_rxprbscntreset_in	Input	
gt_rxprbserr_out	Output	
gt_rxprbssel_in[2:0]	Input	
gt_loopback_in[2:0]	Input	GT Loopback
gt_txresetdone_out	Output	GT Status
gt_rxresetdone_out	Output	
gt_rxdisperr_out[3:0]	Output	
gt_rxnotintable_out	Output	
gt_eyescanreset_in[3:0]	Input	GT Eye Scan
gt_eyescandataerror_out	Output	
gt_eyescantrigger_in	Input	
gt_rxcdrhold_in	Input	GT CDR
gt_rxcdrlock_out	Output	
gt_rxlpmen_in	Input	GT GTX/GTH RX Decision Feedback Equalizer (DFE)
gt_rxdfelpmreset_in	Input	
gt_txpmareset_in	Input	GT TX-PMA Reset
gt_txpcsreset_in	Input	GT TX-PCS Reset
gt_rxpmareset_in	Input	GT RX-PMA Reset
gt_rxpcsreset_in	Input	GT RX-PCS Reset
gt_rxbufreset_in	Input	GT receive elastic buffer Reset
gt_rxpmaresetdone_out	Output	GT PMA resetdone indication

Table B-4: **Ports Added for UltraScale Architecture Devices *(Cont'd)***

Signal	Direction	Description
gt_txbufstatus_out[1:0]	Output	GT TX Buffer status
gt_rxbufstatus_out[2:0]	Output	GT RX Buffer status
gt_dmonitorout_out[16:0]	Output	GT Status

Ports Removed

The ports in Table B-5 were removed under the Transceiver Debug feature of the core.

Note: Ports were renamed/removed only for UltraScale architecture devices to retain backward compatibility for 7 series and Zynq®-7000 devices.

Table B-5: **Ports Removed for UltraScale Devices**

Signal	Direction	Description
gt0_drp_addr_in[8:0]	Input	DRP address bus
gt0_drpi_in[15:0]	Input	Data bus for writing configuration data to the transceiver.
gt0_drpo_out[15:0]	Output	Data bus for reading configuration data from the transceiver.
gt0_drprdy_out	Output	Indicates operation is complete for write operations and data is valid for read operations.
gt0_drpwe_in	Input	DRP write enable
gt0_drpclk_in	Input	DRP Clock
gt0_rxchariscomma_out[3:0]	Output	GT Status
gt0_rxcharisk_out[3:0]	Output	
gt0_rxbyteisaligned_out	Output	
gt0_rxbyterealign_out	Output	
gt0_rxcommadet_out	Output	
gt0_txdiffctrl_in[3:0]	Input	GT TX Driver
gt0_txpostcursor_in[4:0]	Input	
gt0_txprecursor_in[4:0]	Input	
gt0_txpolarity_in	Input	GT Polarity
gt0_rxpolarity_in	Input	
gt0_txprbssel_in[2:0]	Input	GT PRBS
gt0_txprbsforceerr_in	Input	
gt0_rxprbscntreset_in	Input	
gt0_rxprbserr_out	Output	
gt0_rxprbssel_in[2:0]	Input	
gt0_loopback_in[2:0]	Input	GT Loopback

Table B-5: **Ports Removed for UltraScale Devices *(Cont'd)***

Signal	Direction	Description
gt0_txresetdone_out	Output	GT Status
gt0_rxresetdone_out	Output	
gt0_rxdisperr_out[3:0]	Output	
gt0_rxnotintable_out	Output	
gt0_eyescanreset_in[3:0]	Input	GT Eye Scan
gt0_eyescandataerror_out	Output	
gt0_eyescantrigger_in	Input	
gt0_rxcdrhold_in	Input	GT CDR
gt0_rxcdrlock_out	Output	
gt0_rxlpmen_in	Input	GT GTX/GTH RX Decision Feedback Equalizer (DFE)
gt0_rxdfelpmreset_in	Input	
gt0_rxdfeagcovrden_in	Input	
gt0_rxmonitorout_out[6:0]	Output	
gt0_rxmonitorsel_in[1:0]	Input	

Port Changes from v13.0 to v14.0

In the 14.0 version of the core, there have been several changes that make the core pin-incompatible with the previous version(s). These changes were required as part of the general one-off hierarchical changes to enhance the customer experience and are not likely to occur again.

Shared Logic

As part of the hierarchical changes to the core, it is now possible to have the core itself include all of the logic that can be shared between multiple cores, which was previously exposed in the example design for the core.

If you are updating a previous version to the 14.0 version with shared logic, there is no simple upgrade path; it is recommended to consult the Shared Logic sections of this document for more guidance.

Ports Removed

The ports in Table B-6 were removed from the core (non-shared logic).

Table B-6: **Ports Removed**

Port Name and Width	In/Out	Description	What to Do
link_timer_value[8:0]	Input	Used to configure the duration of the Auto-Negotiation function Link Timer. 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). This port is replaced when using the dynamic switching mode.	This is done to ease IP integration of IP. This can be pre-loaded with a lower value by modifying the EXAMPLE_SIMULATION parameter within the IP.
link_timer_basex[8:0]	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).	This is done to ease IP integration of IP. This can be pre-loaded with a lower value by modifying the EXAMPLE_SIMULATION parameter within the IP.
link_timer_sgmii[8:0]	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).	This is done to ease IP integration of IP. This can be pre-loaded with a lower value by modifying EXAMPLE_SIMULATION parameter within the IP.

Generic Removed

The following generic in Table B-7 was removed from the core (non-shared logic).

Table B-7: **Generic Removed**

Generic Name	Description	What to Do
EXAMPLE_SIMULATION	EXAMPLE_SIMULATION generic is provided in all modes to reduce simulation time.	This generic has been removed from the top level to support dcp flow which necessitates removals of generics from the top level wrapper. This generic still exists in wrappers underneath. This can also be controlled during generation of the core. See Port Descriptions in Chapter 2 for guidelines for controlling the same.

Ports Added

The ports in Table B-8 were added to the core (non-shared logic).

Table B-8: **Ports Added (non-shared Logic)**

Port Name and Width	In/Out	Description	What to do
rxoutclk	Output	`rxoutclk` from the transceiver	This was previously connected internally to clocking elements and routed to `rxuserclk` and `rxuserclk2`. This can be left open if `rxoutclk` can be shared across instances or if not should drive clocking elements.
rxuserclk	Input	Signal from the shared logic block to the transceiver	If `rxoutclk` can be shared across instances, connect O/P of shared logic block. If not, connect to rxoutclk after passing through additional clocking elements.
rxuserclk2	Input	Signal from the shared logic block to the transceiver	If `rxoutclk` can be shared across instances, connect O/P of shared logic block. If not, connect to rxoutclk after passing through additional clocking elements.
gt0_pll0outclk_in	Input	Valid only for Artix®-7 families. Indicates out clock from PLL0 of GT Common.	Should be connected to signal of same name from GT Common.
gt0_pll0outrefclk_in	Input	Valid only for Artix-7 families. Indicates reference out clock from PLL0 of GT Common.	Should be connected to signal of same name from GT Common.
gt0_pll1outclk_in	Input	Valid only for Artix-7 families. Indicates out clock from PLL1 of GT Common.	Should be connected to signal of same name from GT Common.
gt0_pll1outrefclk_in	Input	Valid only for Artix-7 families. Indicates reference out clock from PLL1 of GT Common.	Should be connected to signal of same name from GT Common
gt0_pll0lock_in	Input	Valid only for Artix-7 families. Indicates out PLL0 of GT Common has locked.	Should be connected to signal of same name from GT Common.
gt0_pll0refclklost_in	Input	Valid only for Artix-7 families. Indicates out reference clock for PLL0 of GT Common is lost.	Should be connected to signal of same name from GT Common.
gt0_pll0reset_out	Output	Valid only for Artix-7 families. Reset for PLL of GT Common from reset fsm in GT Wizard	Should be connected to signal of same name from GT Common or can be left open if not needed.

Table B-8: **Ports Added (non-shared Logic) *(Cont'd)***

Port Name and Width	In/Out	Description	What to do
gt0_qplloutclk_in	Input	Valid only for non Artix-7 families. Indicates out clock from PLL of GT Common.	Should be connected to signal of same name from GT Common.
gt0_qplloutrefclk_in	Input	Valid only for non Artix-7 Families. Indicates reference out clock from PLL of GT Common.	Should be connected to signal of same name from GT Common.

The ports in Table B-9 were added to the core, but only if the transceiver debug feature was requested during core customization. Consult the relevant transceiver user guide for more information on using these control/status ports.

Table B-9: **Ports Added for Transceiver Debug Feature**

Port Name and Width	In/Out	Description	What to do(1)
gt0_rxchariscomma_out[1:0]	Output	RX Character is Comma indication	LSB is valid in all cases other than 1588 mode where both the bits are valid.
gt0_rxcharisk_out[1:0]	Output	RX Character is K indication	LSB is valid in all cases other than 1588 mode where both the bits are valid.
gt0_rxbyteisaligned_out	Output	RX Byte is aligned indication	
gt0_rxbyterealign_out	Output	RX Byte is realigned indication	
gt0_rxcommadet_out	Output	RX Comma is detected indication	
gt0_txpolarity	Input	Switch the sense of the TXN/P pins	
gt0_txdiffctrl[3:0]	Input	Can be used to tune the transceiver TX waveform	
gt0_txprecursor[4:0]	Input	Can be used to tune the transceiver TX waveform	
gt0_txpostcursor[4:0]	Input	Can be used to tune the transceiver TX waveform	
gt0_rxpolarity	Input	Switch the sense of the RXN/P pins	
gt0_txprbssel_in[2:0]	Input	TX Pattern Generator control signals to test signal integrity	
gt0_txprbsforceerr_in	Input	TX Pattern Generator control signals to test signal integrity	
gt0_rxprbscntreset_in	Input	RX Pattern Checker reset	
gt0_rxprbserr_out	Output	RX Pattern Checker error output	
gt0_rxprbssel_in	Input	RX Pattern Checker control signals to test signal integrity	
gt0_loopback_in[2:0]	Input	Loopback within transceiver	

Table B-9: **Ports Added for Transceiver Debug Feature *(Cont'd)***

Port Name and Width	In/Out	Description	What to do[1]
gt0_txresetdone_out	Output	Transmitter Reset Done	
gt0_rxresetdone_out	Output	Receiver Reset Done	
gt0_rxdisperr_out[1:0]	Output	Indicates there is disparity error in received data	LSB is valid in all cases other than 1588 mode where both the bits are valid.
gt0_rxnotintable_out[1:0]	Output	Indicates received 10 bit pattern was not found in 8B/10B decode table	LSB is valid in all cases other than 1588 mode where both the bits are valid.
gt0_eyescanreset	Input	Reset the EYE Scan logic	
gt0_eyescantrigger	Input	Trigger the EYE Scan logic	
gt0_eyescandataerror	Output	Signals an error during Eye Scan	
gt0_rxcdrhold	Input	Freeze the CDR loop	
gt0_rxlpmhfhold_in	Input	GTP transceiver low-power mode signal	
gt0_rxlpmlfhold_in	Input	GTP transceiver low-power mode signal	
gt0_rxmonitorout_out[6:0]	Output	GTX/GTH transceiver RX DFE Signal	
gt0_rxmonitorsel_in[1:0]	Input	GTX/GTH transceiver RX DFE Signal	

1. Drive this signal according to the relevant transceiver user guide. If DRP interface was unused in previous revisions, then generate without the Transceiver Debug feature.

Ports Moved

The ports in Table B-10 were moved under the Transceiver Debug feature of the core (non-shared logic). If these signals were used in the previous version, then the Transceiver Debug feature needs to be enabled and the appropriate signals mapped and remaining signals tied off to default values.

Table B-10: **Ports Moved (non-shared logic)**

Port Name and Width	In/Out	Description	What to do
gt0_drpdo_out, gt0_drprdy_out	Outputs	These signals come from the transceiver and should be connected either to an external arbiter or to the signals described in the following row.	If there is no external arbiter, connect these signals directly to the associated signals. If the interface is not used, the signals can be left open.
gt0_drpen_in, gt0_drpwe_in, gt0_drpaddr_in[8:0], gt0_drpdi_in[15:0], gt0_drpclk_in	Inputs	These signals go to the transceiver, either from an external arbiter or from the signals described in the preceding row.	If there is no external arbiter, connect these signals directly to the associated core signals. If the interface is not used, tie off the signals to ground and gt0_drpclk_in to txusrclk2.

1000BASE-X State Machines

This appendix is intended to serve as a reference for the basic operation of the 1000BASE-X IEEE 802.3-2008 clause 36 transmitter and receiver state machines.

Introduction

Table C-1 shows the Ordered Sets defined in IEEE 802.3-2008 clause 36 [Ref 5]. These code group characters are inserted by the PCS Transmit Engine into the transmitted data stream, encapsulating the Ethernet frames indicated by 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 C-1 with the remainder of this Appendix. See IEEE 802.3-2008 clause 36 [Ref 5] for further information on these Orders Sets.

Table C-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 C-1 shows 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.

IMPORTANT: *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.

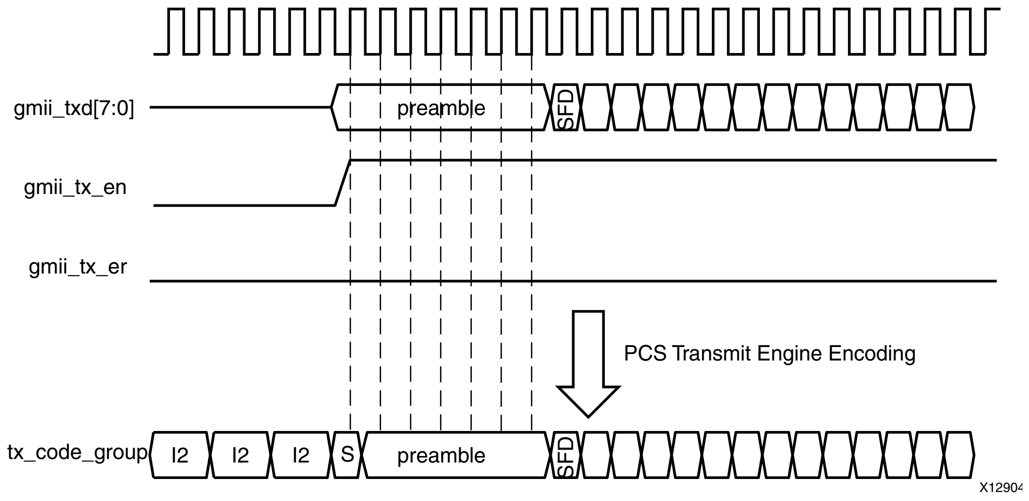


Figure C-1: **1000BASE-X Transmit State Machine Operation (Even Case)**

Reception of the Even Case

Figure C-2 shows 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.

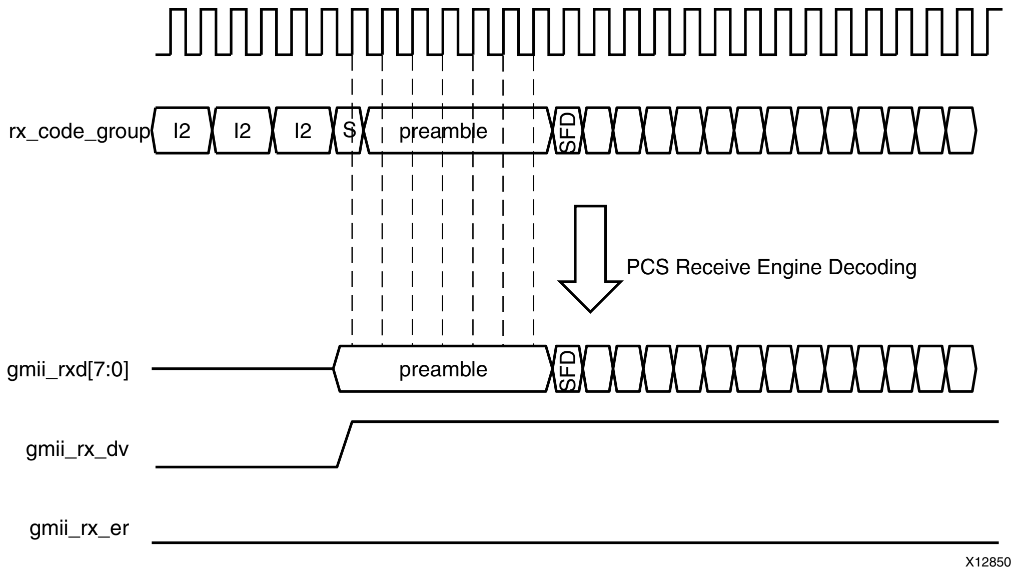


Figure C-2: **1000BASE-X Reception State Machine Operation (Even Case)**

The Odd Transmission Case

Figure C-3 shows 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 shown.

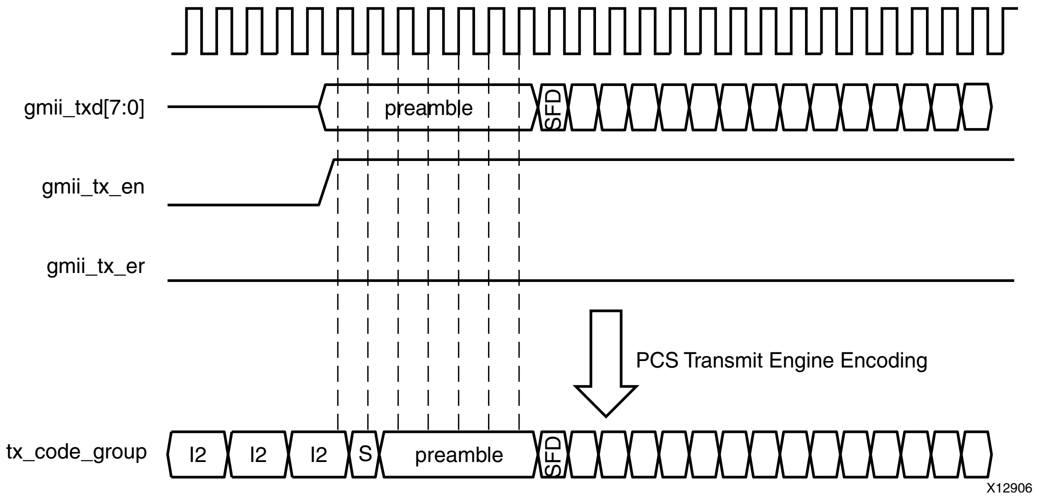


Figure C-3: **1000BASE-X Transmit State Machine Operation (Odd Case)**

Reception of the Odd Case

Figure C-4 shows 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 C-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.

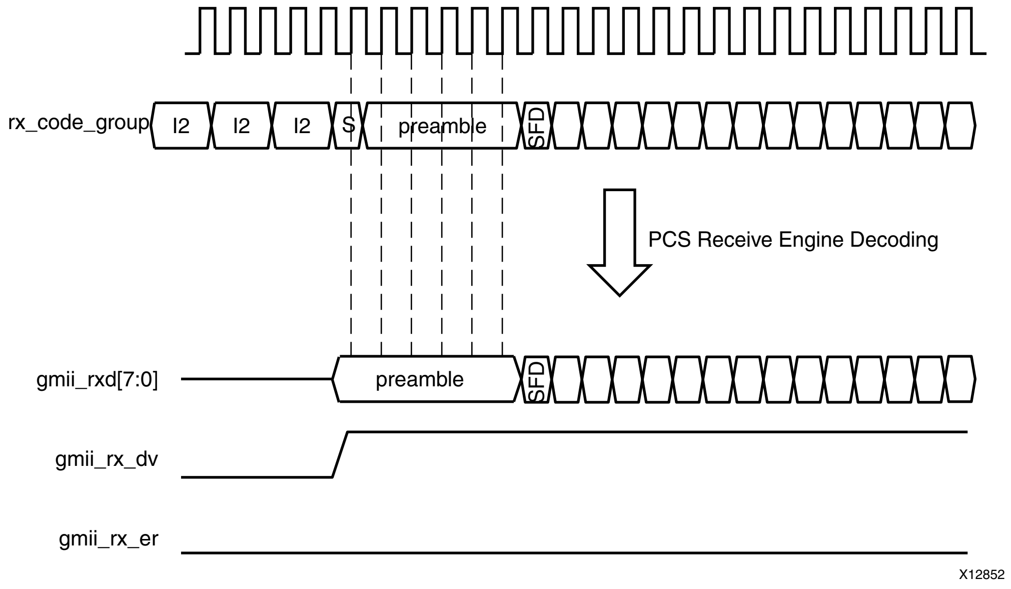


Figure C-4: **1000BASE-X Reception State Machine Operation (Odd Case)**

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 802.3-2008 specification; the preamble loss falls out as a consequence of the state machines (see figures 36-5 and 36-6 in the IEEE 802.3-2008 specification).
- *IEEE 802.3ah-2004* does, however, specifically state in clause 65.1.3.2.1:

Note: 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 C-5 shows 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 shown as the shaded code group.

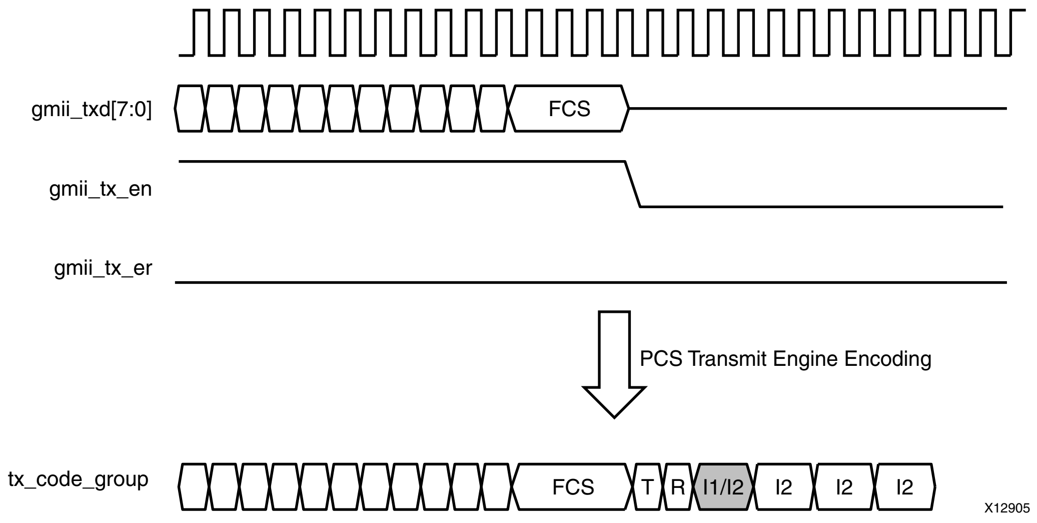


Figure C-5: **1000BASE-X Transmit State Machine Operation (Even Case)**

Reception of the Even Case

Figure C-6 shows 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.

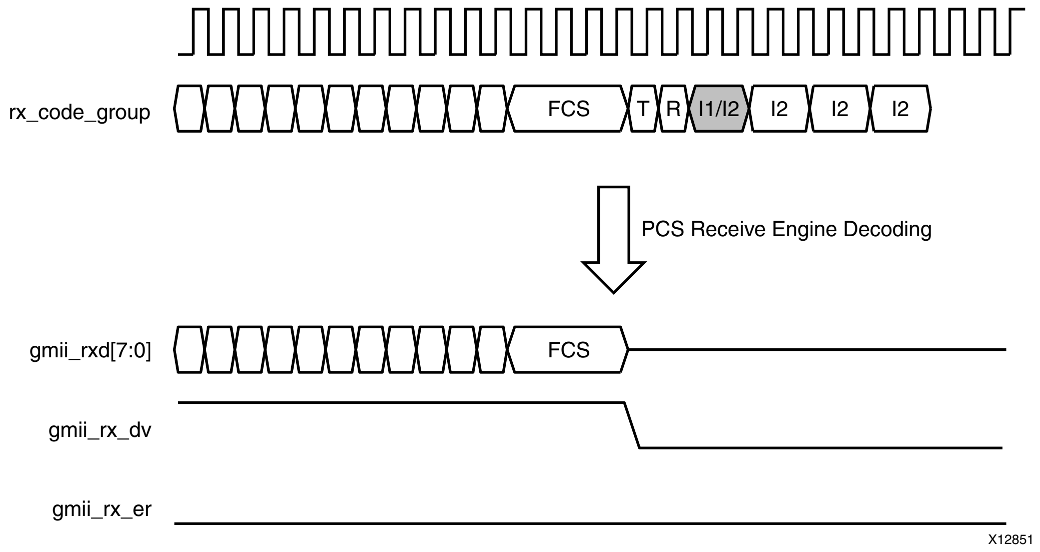


Figure C-6: **1000BASE-X Reception State Machine Operation (Even Case)**

The Odd Transmission Case

Figure C-7 shows 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 shown as the shaded code group.

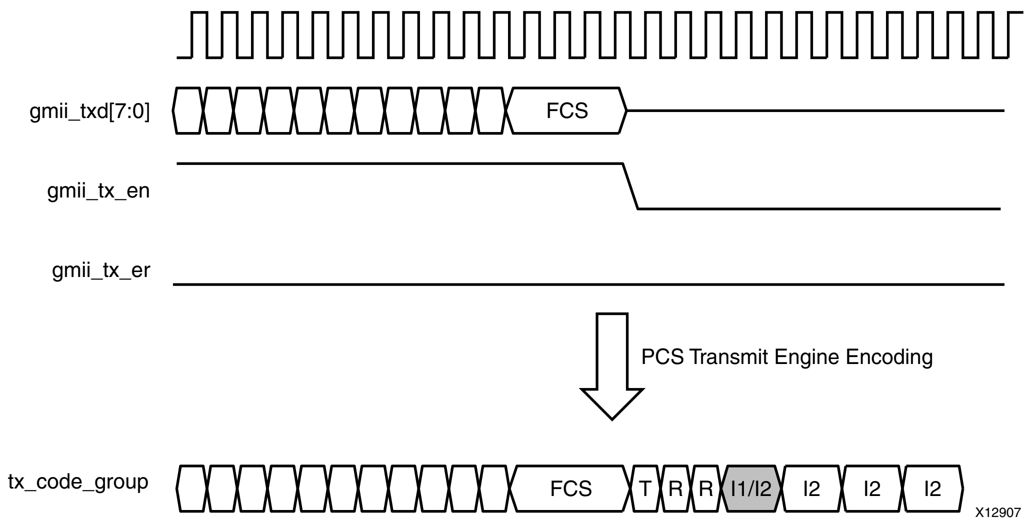


Figure C-7: **1000BASE-X Transmit State Machine Operation (Even Case)**

Reception of the Odd Case

Figure C-8 shows 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 802.3-2008 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.

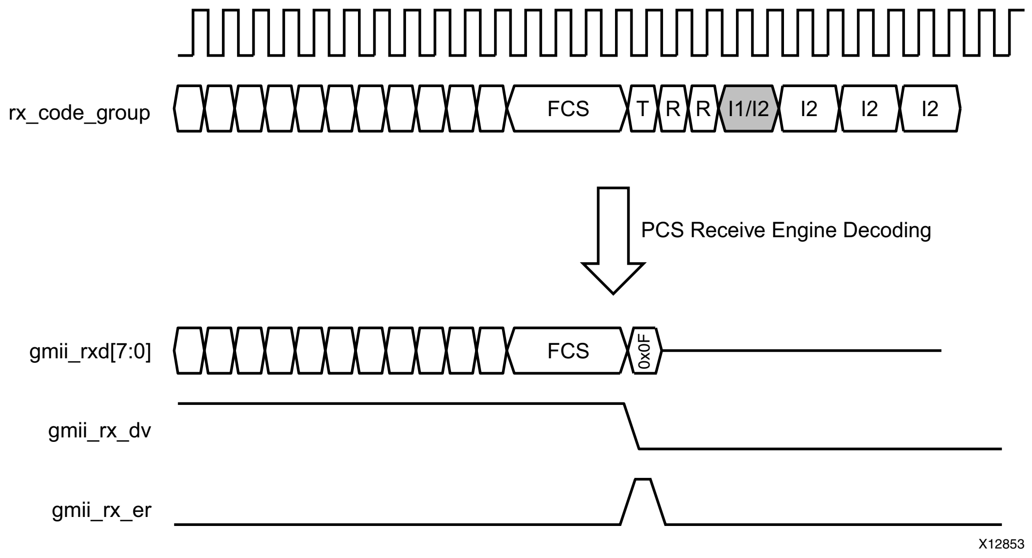


Figure C-8: **1000BASE-X Reception State Machine Operation (Odd Case)**

Receive Elastic Buffer Specifications

This appendix is intended to serve as a reference for the receive 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 receive elastic buffer is shown in Requirement for the Receive Elastic Buffer. The analysis included in this appendix 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 accumulates and either starts to fill or empties the receive elastic buffer over time. The receive elastic buffer copes with this by performing clock correction during the interframe gaps by either inserting or removing Idle characters. The receive elastic buffer always attempts to restore the buffer occupancy to the half full level during an interframe gap. See Clock Correction.

Receive Elastic Buffers: Depths and Maximum Frame Sizes

Device Specific Transceiver Receive Elastic Buffers

Figure D-1 shows the transceiver receive elastic buffer depths and thresholds in UltraScale™ architecture, Zynq®-7000, and 7 series families. Each FIFO word corresponds to a single character of data (equivalent to a single byte of data following 8B/10B decoding).

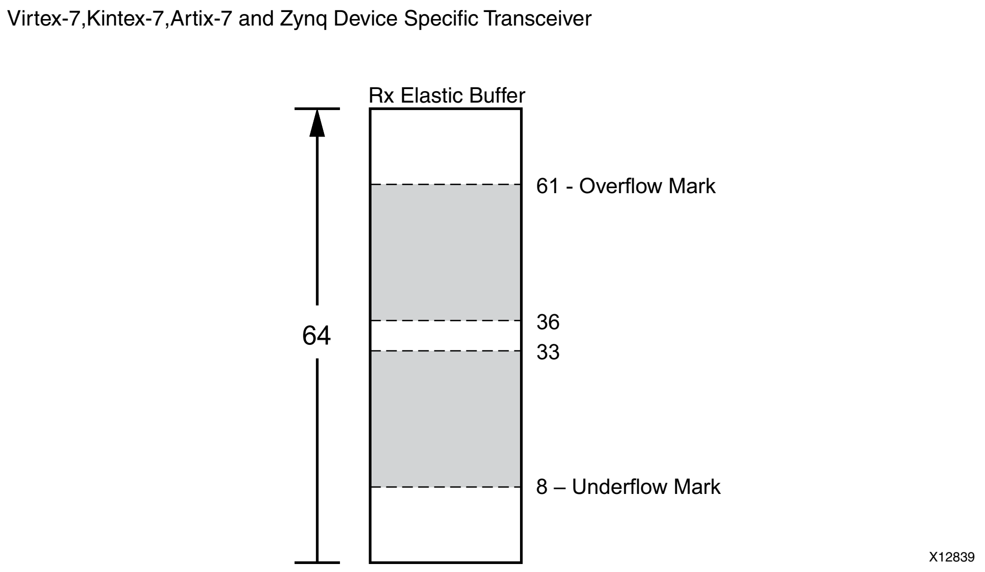


Figure D-1: **Elastic Buffer Sizes for all Transceiver Families**

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 61 - 36 = 25 FIFO locations available before the buffer reaches the overflow mark.
- If the buffer is emptying during reception, then there are 33-8 = 25 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 shown, 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 25=125000 bytes
```

Table D-1 translates this into maximum frame size at different Ethernet speeds. At SGMII speeds lower than 1 Gb/s, performance is diminished because bytes are repeated multiple times (see Using the Client-Side GMII for the SGMII Standard).

Table D-1: **Maximum Frame Sizes: Transceiver Receive Elastic Buffers (100ppm Clock Tolerance)**

Standard / Speed	Maximum Frame Size
1000BASE-X (1 Gb/s only)	125000
SGMII (1 Gb/s)	125000

Table D-1: **Maximum Frame Sizes: Transceiver Receive Elastic Buffers (100ppm Clock Tolerance)**

Standard / Speed	Maximum Frame Size
SGMII (100 Mb/s)	12500
SGMII (10 Mb/s)	1250

SGMII FPGA Logic Receive Elastic Buffer

Figure D-2 shows the FPGA logic receive elastic buffer depth. This logic elastic buffer is used with the core when:

- Performing SGMII over LVDS.
- This buffer can optionally be used to replace the receive elastic buffers of the transceiver when performing SGMII or Dynamic Switching (see Receive Elastic Buffer).

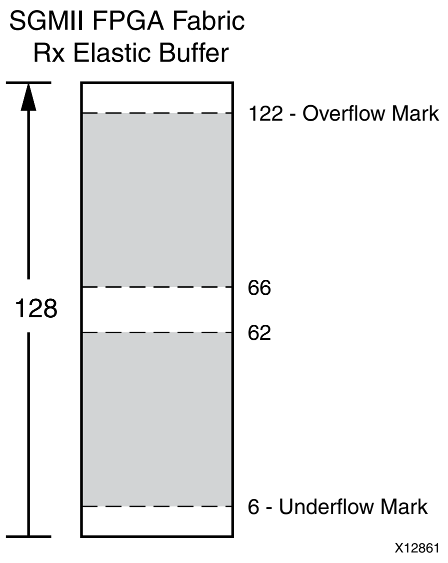


Figure D-2: **Elastic Buffer Size for all Transceiver Families**

The shaded area of Figure D-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 shown, 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 D-2 translates this into maximum frame size at different Ethernet speeds. At SGMII speeds lower than 1 Gb/s, performance is diminished because bytes are repeated multiple times. See Using the Client-Side GMII for the SGMII Standard.

Table D-2: **Maximum Frame Sizes: FPGA logic Receive Elastic Buffers (100 ppm Clock Tolerance)**

Standard / Speed	Maximum Frame Size
1000BASE-X (1 Gb/s only)	280000
SGMII (1 Gb/s)	280000
SGMII (100 Mb/s)	28000
SGMII (10 Mb/s)	2800

TBI Receive Elastic Buffer

For SGMII / Dynamic Switching

The receive elastic buffer used for the SGMII or Dynamic Switching is identical to the method used in SGMII FPGA Logic Receive Elastic Buffer.

For 1000BASE-X

Figure D-3 shows the receive 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 8B/10B decoding).

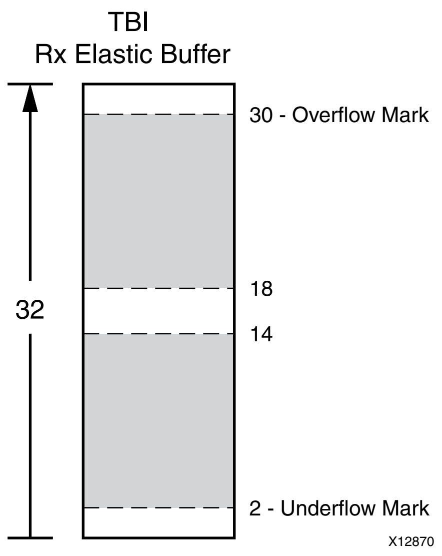


Figure D-3: **TBI Elastic Buffer Size for All Families**

The shaded area of Figure D-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 shown, 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.

Because 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 receive 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 receive 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 receive 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 1 Gb/s (1000BASE-X and SGMII)

The minimum number of clock cycles that can be presented to an Ethernet receiver, according to the *IEEE 802.3-2008* specification, is 64-bit times at any Ethernet speed. At 1 Gb/s 1000BASE-X and SGMII, this corresponds to 8 bytes (8 clock cycles) of interframe gap. However, an interframe gap consists of many code groups, namely /T/, /R/, /I1/ and /I2/ characters (see Appendix C, 1000BASE-X State Machines). Of these, only /I2/ can be used as clock correction characters.

In a minimum interframe gap at 1 Gb/s, you 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 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 Gb/s, you 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 that 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 receive 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 Receive Elastic Buffers: Depths and Maximum Frame Sizes section.

However, because 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 Mb/s (SGMII)

At SGMII, 100 Mb/s, 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 Gb/s case corresponds to a minimum of 80 bytes for the 100 Mb/s 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 Mb/s is repeated ten times, this corresponds to an Ethernet frame size of 20000 bytes, the same size as the 1 Gb/s case.

So in summary, at 100 Mb/s, 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 Receive Elastic Buffers: Depths and Maximum Frame Sizes). Only the SGMII FPGA Logic receive elastic buffer is large enough.

Idle Character Removal at 10 Mb/s (SGMII)

Using a similar argument to the 100 Mb/s 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 Receive 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 Receive Elastic Buffers: Depths and Maximum Frame Sizes
- The number of clock correction characters present in a minimum interframe gap, (see Clock Correction)

Table D-3 summarizes the maximum frame sizes for sustained frame reception when used with the different receive elastic buffers provided with the core. All frame sizes are provided in bytes.

Table D-3: **Maximum Frame Size: Capabilities of the Receive Elastic Buffers**

Ethernet Standard and Speed	Receive Elastic Buffer Type		
	TBI	Device Specific Transceiver	SGMII FPGA Logic Buffer (optional for use with device-specific transceivers)
1000BASE-X (1 Gb/s)	20000 (limited by clock correction)	20000 (limited by clock correction)	20000 (limited by clock correction)
SGMII 1 Gb/s	20000 (limited by clock correction)	20000 (limited by clock correction)	20000 (limited by clock correction)
SGMII 100 Mb/s	20000 (limited by clock correction)	9000 (limited by buffer size)	20000 (limited by clock correction)
SGMII 10 Mb/s	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 802.3-2008* 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.

Implementing External GMII

In certain applications, the client-side GMII datapath can be used as a true GMII to connect externally off-device across a PCB. This external GMII functionality is included in the HDL example design delivered with the core by the IP catalog for 1000BASE-X designs. The extra logic required to accomplish this is described in this appendix.

Note: Virtex®-7 devices support GMII at 3.3 V or lower only in certain parts and packages; see the Virtex-7 Device Documentation. Zynq®-7000, Kintex®-7, and Artix®-7 devices support GMII at 3.3V or lower.

GMII Transmitter Logic (Zynq-7000 and 7 Series Devices)

When implementing an external GMII, the GMII transmitter signals are synchronous to their own clock domain. The core must be used with a TX elastic buffer to transfer these GMII transmitter signals onto the core 125 MHz reference clock (`gtx_clk` when using the TBI; `userclk2` when using the device-specific transceiver). A TX elastic buffer is provided for the 1000BASE-X standard in the example design.

Use a combination of IODELAY elements on the data, and use BUFIO and BUFR regional clock routing for the `gmii_tx_clk` input clock

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. 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. See the device data sheets for more information.

The clock is then placed onto regional clock routing using the BUFR component and the input GMII TX data immediately resampled. 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 XDC; these can be edited if required.

GMII Receiver Logic

Figure E-1 shows an external GMII receiver created in a 7 series 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 E-1 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 configuration 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 The Ten-Bit Interface, and 1000BASE-X with Transceivers.

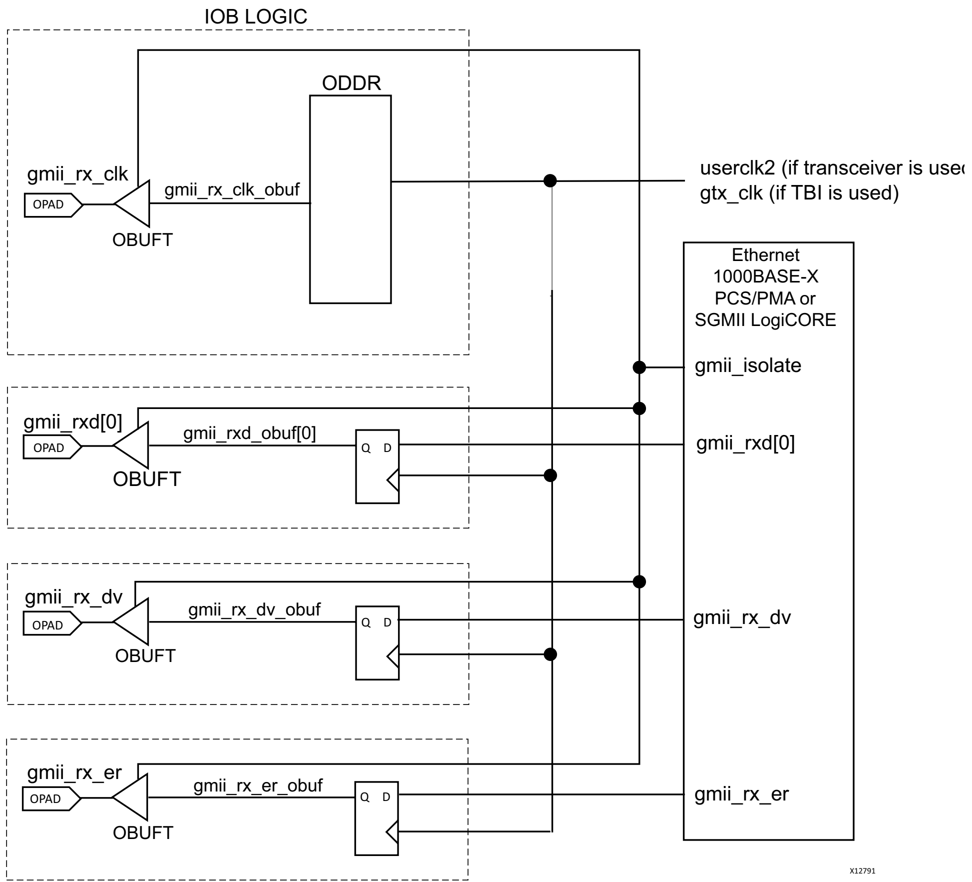


Figure E-1: **External GMII Receiver Logic**

Appendix F

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the 1000BASE-X PCS/PMA or SGMII core, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the 1000BASE-X PCS/PMA or SGMII core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips. See Xilinx Ethernet IP Solution Center

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are shown in the following bulleted list, and can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the 1000BASE-X PCS/PMA or SGMII Core

AR: 54667

Contacting Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Additional Resources.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

Note: Access to WebCase is not available in all cases. Log in to the WebCase tool to see your specific support options.

Debug Tools

There are many tools available to address 1000BASE-X PCS/PMA or SGMII core design issues. It is important to know which tools are useful for debugging various situations.

Vivado Lab Tools

Vivado® lab tools insert logic analyzer and virtual I/O cores directly into your design. Vivado lab tools also allow you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 18].

Reference Boards

Various Xilinx development boards support 1000BASE-X PCS/PMA or SGMII core. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series FPGA evaluation boards
  - KC705
  - VC707

Simulation Debug

The simulation debug flow for Questa® SIM is shown in Figure F-1. A similar approach can be used with other simulators.

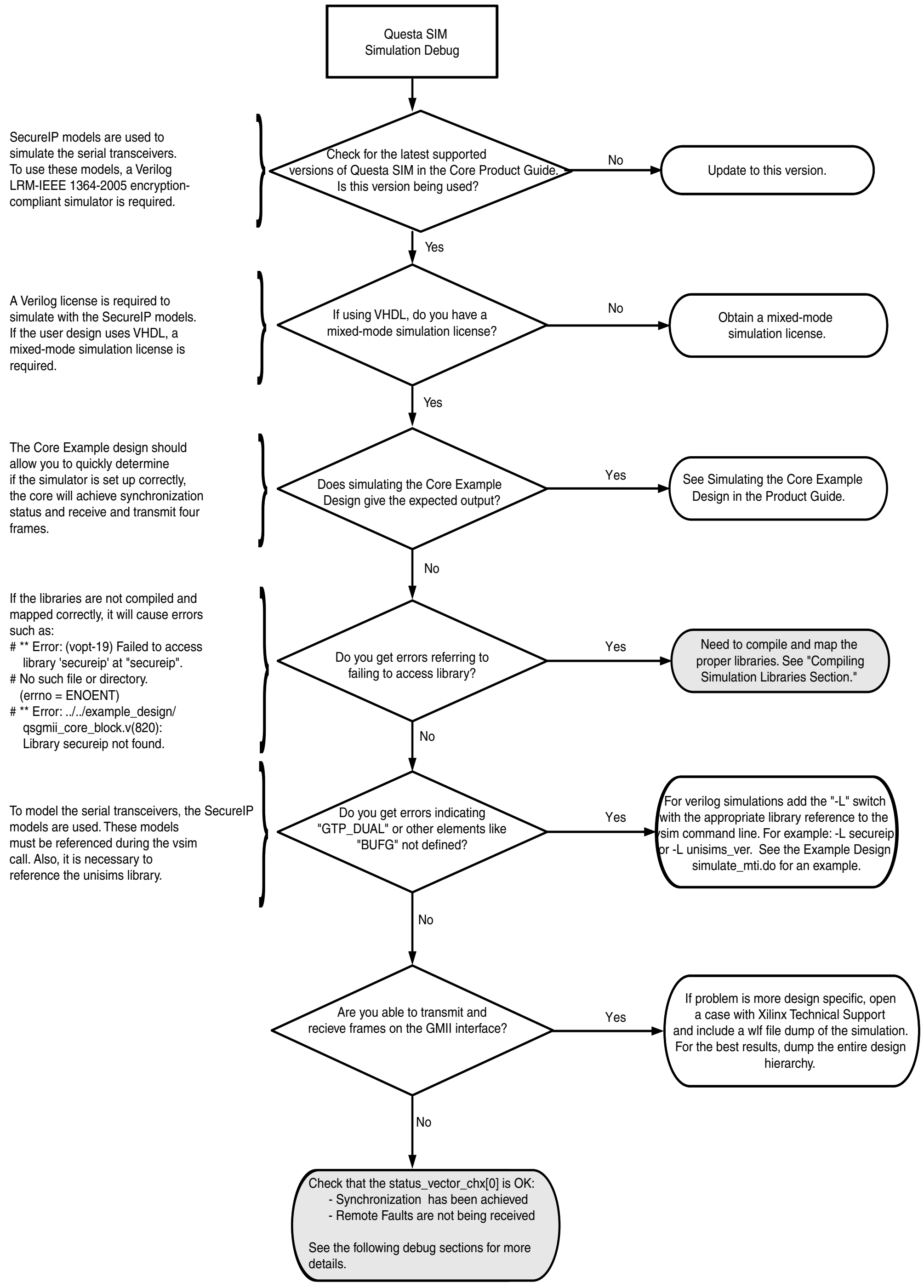

Questa SIM Simulation Debug
SecureIP models are used to simulate the serial transceivers. To use these models, a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator is required.
Check for the latest supported versions of Questa SIM in the Core Product Guide. Is this version being used?
No
Update to this version.
Yes
A Verilog license is required to simulate with the SecureIP models. If the user design uses VHDL, a mixed-mode simulation license is required.
If using VHDL, do you have a mixed-mode simulation license?
No
Obtain a mixed-mode simulation license.
Yes
The Core Example design should allow you to quickly determine if the simulator is set up correctly, the core will achieve synchronization status and receive and transmit four frames.
Does simulating the Core Example Design give the expected output?
Yes
See Simulating the Core Example Design in the Product Guide.
No
If the libraries are not compiled and mapped correctly, it will cause errors such as:
# ** Error: (vopt-19) Failed to access library 'secureip' at "secureip".
# No such file or directory. (errno = ENOENT)
# ** Error: ../../example_design/qsgmii_core_block.v(820): Library secureip not found.
Do you get errors referring to failing to access library?
Yes
Need to compile and map the proper libraries. See "Compiling Simulation Libraries Section."
No
To model the serial transceivers, the SecureIP models are used. These models must be referenced during the vsim call. Also, it is necessary to reference the unisims library.
Do you get errors indicating "GTP_DUAL" or other elements like "BUFG" not defined?
Yes
For verilog simulations add the "-L" switch with the appropriate library reference to the vsim command line. For example: -L secureip or -L unisims_ver. See the Example Design simulate_mti.do for an example.
No
Are you able to transmit and recieve frames on the GMII interface?
Yes
If problem is more design specific, open a case with Xilinx Technical Support and include a wlf file dump of the simulation. For the best results, dump the entire design hierarchy.
No
Check that the status_vector_chx[0] is OK:
- Synchronization has been achieved
- Remote Faults are not being received
See the following debug sections for more details.

Figure F-1: **Simulation Debug Flow**

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado lab tools are a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado lab tools for debugging the specific problems.

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.
- If Clock Data Recovery (CDR) is not done on the board, increase RX_CDRLOCK_TIME parameter in the `gtwizard_init` file. This value is silicon-specific. The value given by default is a typical value and can be increased to the maximum TDLOCK value as specified in the device datasheet.
- For BASE-X/SGMII modes in UltraScale architecture devices, if the transceiver is not coming out of the reset sequence, check the following:
  a. If the Transceiver Control and status is enabled, check the DRP clock frequency. The frequency should be 50 MHz.
  b. If (a) is not applicable and shared logic is in the core, the independent clock frequency must be 300 MHz because in this case the independent clock is used to generate a 50 MHz clock required by the gtwizard FSM.
  c. If (a) is not applicable and shared logic is in the example design, the `independent_clock_bufgdiv6` must be fed with a 50 MHz clock.

Problems with the MDIO

- Ensure that the MDIO is driven properly. See MDIO Management Interface 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]` 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.

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 Start-up Sequencing 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 Special Design Considerations.

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 Register 1: Status Register
- Use the auto-negotiation interrupt port of the core (see Using the Auto-Negotiation Interrupt).

If Auto-Negotiation is not completing:

1. Ensure that auto-negotiation is enabled in *both* the core and in the link partner (the device or test equipment connected to the core). Auto-Negotiation cannot complete successfully unless both devices are configured to perform auto-negotiation.

   The auto-negotiation procedure requires that the auto-negotiation handshaking protocol between the core and its link partner, which lasts for several link timer periods, occur without a bit error. A detected bit error causes auto-negotiation to go back to the beginning and restart.
   Therefore, a link with an exceptionally high bit error rate might not be capable of completing auto-negotiation, or might lead to a long auto-negotiation period caused by the numerous 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 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 MDIO Register 1: Status register, (see Register 1: Status Register) 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 in Table 2-75.

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 Problems with a High Bit Error Rate in a subsequent section.

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 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; 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 issue caused by differences between the transmitting and receiving ends. Check all clock management circuitry and clock frequencies applied to the core and to the 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 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 through an optical cable, this can indicate a faulty optical module or a PCB problem.
- Try swapping the optical module on a misperforming device and repeat the tests.

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 Vivado lab tools or an external logic analyzer.

  ```
  rxdisperr
  rxnotintable
  ```

  These signals should not be asserted over the duration of a few seconds, minutes or even hours. If they are frequently asserted, it might indicate 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 by an optical cable, it might indicate a faulty optical module.
    - If the core exhibits correct operation in device-specific transceiver parallel loopback but not in serial loopback, this can indicate a device-specific transceiver problem. See Answer Record 19699 for details.
- A mild form of bit error rate might be solved by adjusting the transmitter TX_PREEMPHASIS, TX_DIFF_CTRL and TERMINATION_IMP attributes of the device-specific transceiver.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. *Serial-GMII Specification V1.7* (CISCO SYSTEMS, (ENG-46158))
2. *7 Series FPGAs SelectIO™ Resources User Guide* (UG471)
3. *UltraScale Architecture SelectIO Resources User Guide* (UG571)
4. *LVDS 4x Asynchronous Oversampling Using 7 Series FPGAs* (XAPP523)
5. *Ethernet Standard 802.3-2008 Clauses 22, 35, 36 and 38* (Part 3)
6. *LogiCORE IP 7 Series FPGAs Transceivers Wizard Product Guide* (PG168)
7. *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)
8. *7 Series FPGAs GTP Transceivers User Guide* (UG482)
9. *7 Series FPGA Clocking Resources User Guide* (UG472)
10. *Parameterizable 8B/10B Encoder* (XAPP1122)
11. *UltraScale Architecture Clocking Resources User Guide* (UG572)
12. *LogiCORE IP Tri-Mode Ethernet MAC Product Guide* (PG051)
13. *Vivado® Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
14. *Vivado Design Suite User Guide: Designing with IP* (UG896)
15. *Vivado Design Suite User Guide: Getting Started* (UG910)
16. *Vivado Design Suite User Guide - Logic Simulation* (UG900)
17. *ISE® to Vivado Design Suite Migration Guide* (UG911)
18. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)

19. *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585)

Additional Core Resources

After generating the core, the *Ethernet 1000BASE-X PCS/PMA or SGMII Release Notes* are available in the document directory

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 802.3-2008*
- *Serial-GMII Specification V1.7* (CISCO SYSTEMS, ENG-46158)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/01/2014	14.3	• Added 1588(PTP) GTH transceiver support for UltraScale architecture. • Document re-structured • Added information on shared logic for cases using device-specific transceiver
04/02/2014	14.2	• Added SGMII over LVDS for UltraScale architecture devices. • Modified status_vector(0) and LINK_STATUS register to take care of reset sequence completion of transceivers. • Updated screen displays in chapter 13. • Added `reset_done` signal to several figures. • Added External MDIO feature. • Modified ambiguous text for BUFG usage in 7 series device SGMII over LVDS.
12/18/2013	14.1	• Added UltraScale™ architecture support. • Added 1588(PTP) GTH transceiver support in the core. • Updated screen displays in Chapter 13.

10/02/2013	14.0	• Removed link timer value ports from block_wrapper • Enhanced support for IP Integrator. • Reduced warnings in synthesis and simulation. • Updated clock synchronizers to improve Mean Time Between Failures (MTBF) for metastability. • Added optional transceiver control and status ports. • Added Vivado IDE option to include or exclude shareable logic resources in the core. • Added new board Vivado IDE tab for targeting evaluation boards.
06/19/2013	13.0	• Revision number advanced to 13.0 to align with core version number 13.0. • Added Zynq-7000 AP SoC EMAC support. • Added 1588 (PTP) support in the core. • Modified PHYAD to be a GUI option instead of block level port. • Updated Figures 2-2, 2-3, 2-6, 2-7, 2-8, 2-9, 13-1, 13-2, 13-3, and 13-4.
03/20/2013	2.0	• Updated to core version 12.0. • Removed all material related to devices not supported by the Vivado Design Suite. • Removed all material related to ISE® Design Suite, CORE Generator™ tools, and UCF. • Updated 7 series FPGA transceivers diagrams. • Added Zynq support for SGMII over LVDS feature.
12/18/2012	1.2	• Updated for 14.4 and 2012.4. Updated to core version 11.5. • Updated Debugging appendix. • Added new information about Artix®-7 FPGAs throughout the guide • Added XCI file information. • Added statement about wait time for Vivado Design Suite use with transceiver wizards. • Updated Figures 6-8, 6-9, 6-10, 6-17, 7-2, and G-1. • Added XDC information.
10/16/2012	1.1	Updated for 14.3 and 2012.3. Added Gigabit Ethernet EDK application for Zynq®-7000 devices.
07/25/2012	1.0	Initial Xilinx release in product guide format. This document is based on the following documents: • *LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII v11.3 Product Guide* • *LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII v11.3 Data Sheet*

Please Read: Important Legal Notices