



# **1G/2.5G Ethernet PCS/PMA or SGMII LogiCORE IP Product Guide (PG047)**

## Introduction

- Features
- IP Facts

## Overview

- Navigating Content by Design Process
- Core Overview
- Applications
- Recommended Design Experience
- Licensing and Ordering

## Product Specification

- MAC
- GMII and 1G or 2.5G SGMII
- Physical Coding Sublayer
- Ten Bit Interface
- Physical Medium Attachment
- Physical Medium Dependent
- Standards
- Performance
- Voltage Requirements
- Speed Grades
- Resource Utilization
- Port Descriptions
- Register Space

## Designing with the Core

- General Design Guidelines
- Shared Logic
- Clocking
- Resets
- 1000BASE-X or 2500BASE-X with Transceivers
- SGMII/Dynamic Switching with Transceivers
- Synchronous SGMII over LVDS
- Asynchronous 1000BASE-X/SGMII over LVDS
- The Ten-Bit Interface
- Using the Client-Side GMII Datapath
- Auto-Negotiation
- Dynamic Switching of 1000BASE-X and SGMII
- Interfacing to Other Cores
- Special Design Considerations

## Design Flow Steps

- Customizing and Generating the Core
- Constraining the Core
- Simulation
- Synthesis and Implementation

## Example Design

- 1000BASE-X or 2500BASE-X with Transceiver Example Design
- SGMII/Dynamic Switching Using a Transceiver Example Design
- Synchronous SGMII over LVDS Example Design (Applicable for Non-Versal Devices)
- Asynchronous 1000BASE-X/SGMII over LVDS (Applicable for Non-Versal Devices)
- Asynchronous 1000BaseX/SGMII over LVDS Example Design (Versal Devices)
- 1000BASE-X with TBI Example Design
- SGMII/Dynamic Switching with TBI Example Design

## Test Bench

- Core with MDIO Interface
- Core without MDIO Interface
- Customizing the Test Bench

## Verification, Compliance, and Interoperability

- Simulation
- Hardware Testing

## Upgrading

- Migrating to the Vivado Design Suite
- Upgrading in the Vivado Design Suite

## 1000BASE-X State Machines

- Introduction
- Start of Frame Encoding
- End of Frame Encoding

## Receive Elastic Buffer Specifications

- Introduction
- Receive Elastic Buffers: Depths and Maximum Frame Sizes
- Clock Correction
- Maximum Frame Sizes for Sustained Frame
- Jumbo Frame Reception

## **Implementing External GMII**

- GMII Transmitter Logic (Zynq 7000 and 7 Series)
- GMII Receiver Logic

## **Debugging**

- Finding Help with AMD Adaptive Computing Solutions
- Debug Tools
- Simulation Debug
- Hardware Debug

## **Additional Resources and Legal Notices**

- Support Resources
- Finding Additional Documentation
- References
- Revision History
- Please Read: Important Legal Notices

# Introduction

The AMD LogiCORE™ IP 1G/2.5G Ethernet 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 and 2500BASE-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 \(CISCO SYSTEMS, ENG-46158\)](#). Dynamic switching between 1000BASE-X and SGMII standards is also supported.

# Features

The AMD LogiCORE™ IP 1G/2.5G Ethernet PCS/PMA or SGMII core provides the following capabilities:

- Supported physical interfaces for 1000BASE-X and 2500BASE-X, SGMII, or 2.5G SGMII standards
- Integrated device-specific transceiver interface
- Support for 1000BASE-X or SGMII over High-Speed Select Input/Output (I/O) Low Voltage Differential Signaling (LVDS) inZynq 7000 , UltraScale, UltraScale+, and 7 series devices
- Support for Asynchronous Low Voltage Differential Signaling (LVDS) solution with Advanced IO Wizard module for AMD Versal™ adaptive SoC devices.
- Configured and monitored through MDIO
- 1000BASE-X or 2500BASE-X and SGMII Auto-Negotiation supported
- Support for full duplex only
- Support preamble shrinkage for 2.5G core speed

# IP Facts

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family <sup>1</sup>	AMD Versal™ Adaptive SoC, AMD UltraScale+™ , AMD UltraScale™ , Zynq UltraScale+ MPSoC,Zynq 7000 SoC, 7 series FPGAs
Supported User Interfaces	GMII <sup>2</sup>
Resources	<a href="#">Performance and Resource Utilization</a>
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog and VHDL
Test Bench	Demonstration Test Bench
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Verilog and VHDL
Supported S/W Driver	See AXI Ethernet (TEMAC + 1000BASE-X/SGMII) for 1G driver support
Tested Design Flows <sup>3</sup>	
Design Entry	AMD Vivado™ Design Suite
Simulation	For supported simulators, see the <i>Vivado Design Suite User Guide: Release Notes, Installation, and Licensing</i> ( <a href="#">UG973</a> ).
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: <a href="#">54667</a>
All Vivado IP Change Logs	Master Vivado IP Change Logs: <a href="#">72775</a>
<a href="#">Support Web Page</a>	
1. For a complete list of supported devices, see the AMD Vivado™ IP catalog. For supported family configurations see <a href="#">Resource Utilization</a> . For supported speed grades see <a href="#">Speed Grades</a> .	

LogiCORE IP Facts Table
<p>2. MII is supported only when used with EMAC0/EMAC1 present in the Zynq 7000 SoC and Zynq UltraScale+ MPSoC processing subsystem (PS).</p> <p>3. For the supported versions of third-party tools, see the <i>Vivado Design Suite User Guide: Release Notes, Installation, and Licensing</i> (UG973).</p>

# Overview

## Navigating Content by Design Process

AMD Adaptive Computing documentation is organized around a set of standard design processes to help you find relevant content for your current development task. You can access the AMD Versal™ adaptive SoC design processes on the [Design Hubs](#) page. You can also use the [Design Flow Assistant](#) to better understand the design flows and find content that is specific to your intended design needs.

### Hardware, IP, and Platform Development

Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the AMD Vivado™ timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:

- [Port Descriptions](#)
- [Register Space](#)
- [Clocking](#)
- [Resets](#)
- [Customizing the Test Bench](#)
- [Example Design](#)

## Core Overview

This product guide provides information for generating a 1000BASE-X or 2500BASE-X Ethernet Physical Coding Sublayer/Physical Medium Attachment (PCS/PMA) or a Serial Gigabit Media Independent Interface (SGMII) or 2.5G SGMII core, customizing and simulating the core using the provided example design, and running the design files through implementation using AMD tools. The two standards supported are sufficiently similar to be supported in the same core. The 1G/2.5G Ethernet PCS/PMA or SGMII IP core is a fully-verified solution that supports Verilog Hardware Description Language (HDL) and Very-High Speed Integrated Circuit (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 1G/2.5G Ethernet PCS/PMA or SGMII [product page](#).

## Applications

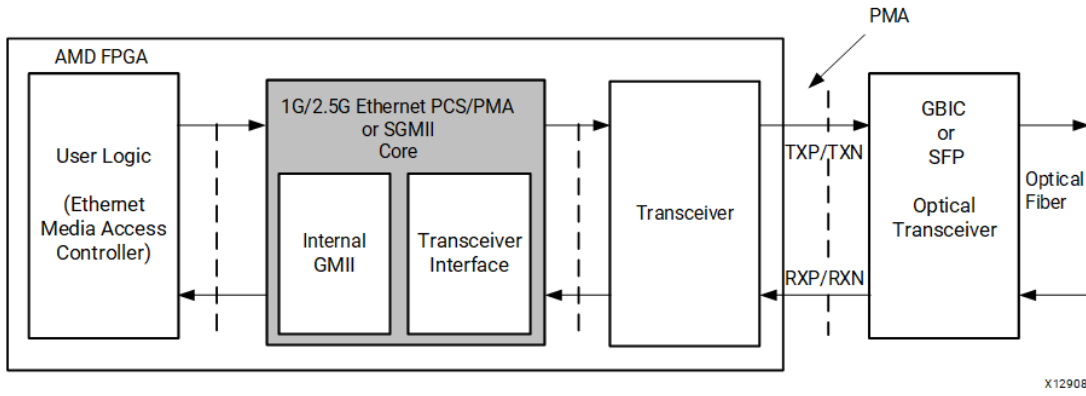
The core can be used for applications using the Ethernet 1000BASE-X, 2500BASE-X, SGMII or 2.5G SGMII standards.

### Ethernet 1000BASE-X or 2500BASE-X

The following figure shows a typical application for the core meeting the 1000BASE-X or 2500BASE-X standard using a device-specific transceiver to provide the Physical Coding Sublayer (PCS) and Physical Medium Attachment (PMA) sublayers for 1/2.5 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 or 2500BASE-X PCS/PMA is connected to an embedded Ethernet Media Access Controller (MAC), for example, the AMD Tri-Mode Ethernet MAC core in all supported devices or Ethernet MAC (EMAC0 or EMAC1) present in the AMD Zynq™ 7000 SoC or Gigabit Ethernet MAC (GEM0 or GEM1) present in Zynq AMD UltraScale+™ processing subsystem (PS) and AMD Versal™ CIPS. The core does not support 2500BASE-X and 2.5G SGMII when the core is generated to interface with the Ethernet MAC (EMAC0 or EMAC1) present in the AMD Zynq™ 7000 SoC or Gigabit Ethernet MAC present in Zynq AMD UltraScale+™ PS or AMD Versal™ CIPS.

**Figure: Typical 1000BASE-X or 2500BASE-X Application**



## 1G or 2.5G SGMII

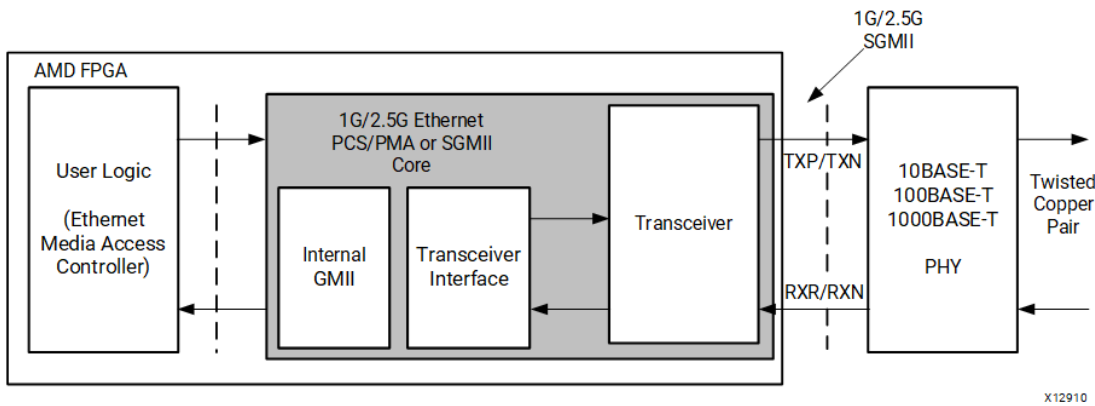
The core can operate in two SGMII modes:

### GMII to SGMII Bridge

The following figure 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 1G or 2.5G SGMII. (This can be a tri-mode PHY providing 10BASE-T, 100BASE-T, and 1000BASE-T operation for 1G.)
- The core GMII interface is connected to an embedded Ethernet MAC, for example, the AMD Tri-Mode Ethernet MAC core (in supported devices) or Ethernet MAC (EMAC0 or EMAC1) present in the AMD Zynq™ 7000 SoC or PS or Gigabit Ethernet MAC (GEM0 or GEM1) present in Zynq AMD UltraScale+™ or AMD Versal™ CIPS. The core does not support 2500BASE-X and 2.5G SGMII when the core is generated to interface with the Ethernet MAC (EMAC0 or EMAC1) present in the AMD Zynq™ 7000 SoC or Gigabit Ethernet MAC present in Zynq AMD UltraScale+™ PS or AMD Versal™ CIPS.

**Figure: Typical Application for GMII to SGMII Bridge Mode**

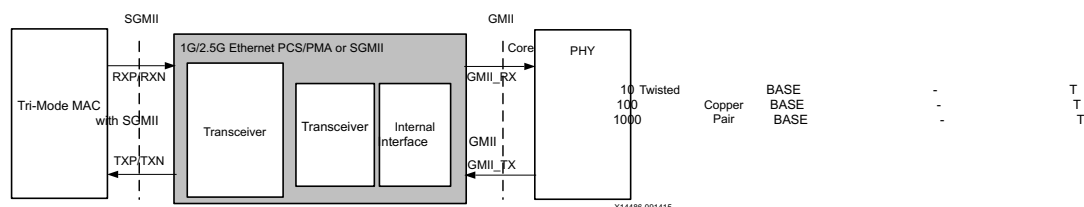


### SGMII to GMII Bridge

The following figure 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. Operation in 2.5G SGMII mode is similar to 1G SGMII mode. Data rates of 25 Mbps or 250 Mbps are not supported for the 2.5G SGMII interface.

- The device-specific transceiver is connected to an external off-the-shelf Ethernet MAC device that also supports 1G or 2.5G SGMII. (This can be the AMD Tri-Mode Ethernet MAC core connected to the 1G/2.5G Ethernet PCS/PMA or SGMII core operating in GMII to SGMII mode).
- The GMII core interface is connected to a tri-mode or 2.5G PHY, providing 10BASE-T, 100BASE-T, and 1000BASE-T operation.

**Figure: Typical Application for SGMII to GMII Bridge Mode**

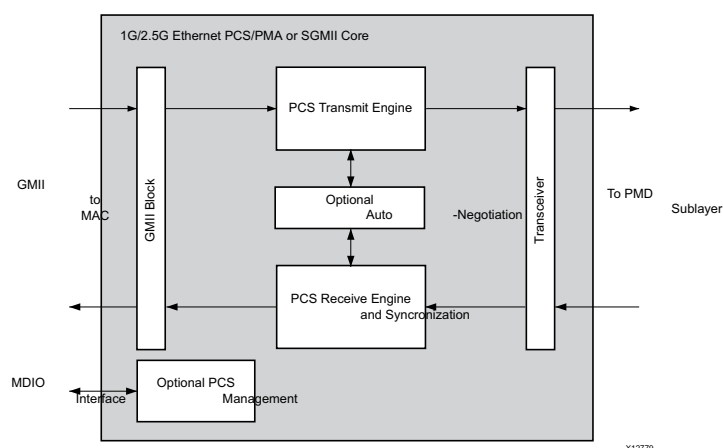


## 1G/2.5G Ethernet PCS/PMA or SGMII Using a Device-Specific Transceiver

Using the core with a device-specific transceiver provides the functionality to implement the 1000BASE-X or 2500BASE-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. The following figure 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: 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.

Some devices do not support 3.3V on pads. See the *7 Series FPGAs SelectIO Resources User Guide (UG471)* or the *UltraScale Architecture SelectIO Resources User Guide (UG571)* for the respective device.

### 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. 2500BASE-X follows the same Auto-Negotiation function as defined in IEEE 802.3-2008 clause 37.

### 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. 2500BASE-X uses the same PCS management registers as 1000BASE-X PCS.

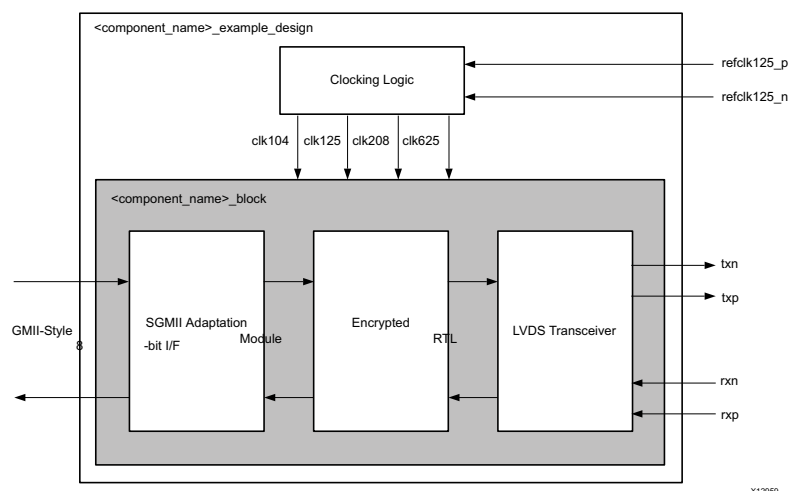
## Transceiver

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 Using Component Mode

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 the following figure. The core does not support 2.5G SGMII modes for the LVDS physical interface. Synchronous LVDS mode is not supported for AMD Versal™ devices.

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



**Note:** For UltraScale devices, clocking logic generates 125, 312.5, and 625 MHz clocks respectively. Frequencies shown in the previous figure are applicable for 7 series devices.

The core netlist in this implementation is identical to that of the previous figure and all core netlist blocks are identical to those described in [1G/2.5G Ethernet PCS/PMA or SGMII Using a Device-Specific Transceiver](#).

As shown in the previous figure, 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 or 7 series FPGA transceiver. This is only possible at a serial line rate of 1.25 Gbps. The following subsections describe the design requirements.

## 1G 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

- AMD Kintex™ 7 devices, -2 speed grade or faster for devices with HR Banks or -1 speed grade or faster for devices with HP banks.
- AMD Virtex™ 7 devices, -2 speed grade or faster for devices with HR Banks or -1 speed grade or faster for devices with HP banks.
- AMD 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 devices, any I/O supporting a maximum of 1.25 Gbps should support SGMII over device LVDS. See the performance characteristics of I/O banks in the UltraScale device in the device data sheet.

**Recommended:** For Chip-to-Chip Copper Implementations, 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.

## Asynchronous SGMII/1000BASE-X Over LVDS

The core supports Asynchronous SGMII/1000BASE-X over LVDS using High Speed SelectIO logic resources in Native Mode for UltraScale / UltraScale+ devices.

For Versal devices, the Asynchronous SGMII/1000Base-X over LVDS is using the Advanced IO Wizard IP as a subcore module. This



enables direct connection to an external PHY without using the FPGA transceiver. 2.5G data rates are not supported in this mode.

## Supported Devices

Asynchronous SGMII/1000BASE-X over LVDS is supported in Versal, UltraScale, and UltraScale+ devices only.

**Note:** Asynchronous LVDS mode is not supported for the following Versal devices.

- xcvp1002 and xcvp1052: The I/O Banks available on devices with 'nfvi1369' package do not meet the requirements for the placement of Advanced I/O Wizard.
- xcvn3716

## SGMII Using Asynchronous Oversampling over 7 Series FPGAs

See the application note ([XAPP523](#)) for information about 7 series devices using asynchronous oversampling. The application note does not support 2.5G SGMII modes for the LVDS physical interface.

### 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 or 2500BASE-X.

### Receiver UI Specification

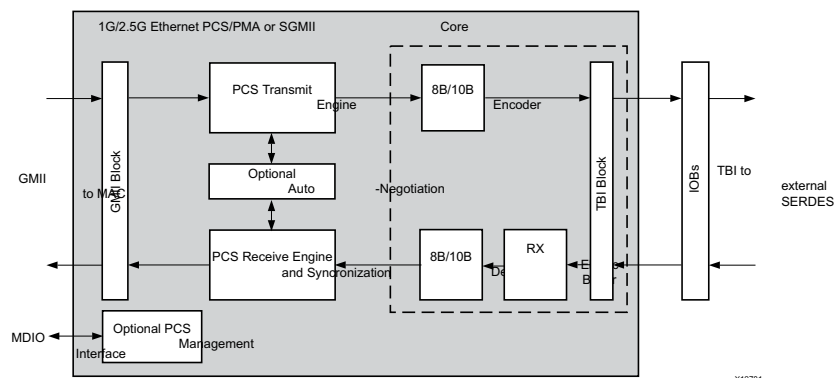
**!! Important:** The DRU must have at least two valid sampling points per data bit, requiring 0.5 UI of opening. The settings of the FPGA add 0.125 UI of requirement making a total opening requirement at the receiver of 0.625 UI.

**Recommended:** For Chip-to-Chip Copper Implementations, 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.

## 1G Ethernet 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. TBI mode is not supported for 2500BASE-X or 2.5 SGMII data rates.

**Figure: 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 the above figure). Other blocks are identical to those previously defined.

Versal, UltraScale+, Zynq UltraScale+ MPSoC, UltraScale, Zynq 7000, AMD Artix™ 7, Zynq 7000, and AMD Virtex™ 7 devices do not support the TBI. AMD 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 1G/2.5G Ethernet 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 AMD implementation software with the Xilinx Design Constraints (XDC) is recommended.

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

## Licensing and Ordering

This AMD LogiCORE™ IP module is provided at no additional cost with the AMD Vivado™ Design Suite under the terms of the [End User License](#).

This AMD LogiCORE™ IP module is provided under the terms of the [Core License Agreement](#). The module is shipped as part of the AMD Vivado™ Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. To generate a full license, visit the [product licensing web page](#). Evaluation licenses and hardware timeout licenses might be available for this core. Contact your [local sales representative](#) for information about pricing and availability.

---

**Note:** To verify that you need a license, check the License column of the IP Catalog. Included means that a license is included with the AMD Vivado™ Design Suite; Purchase means that you have to purchase a license to use the core.

---

For more information about this core, visit the product web page.

Information about other AMD LogiCORE™ IP modules is available at the [Intellectual Property](#) page. For information about pricing and availability of other AMD LogiCORE IP modules and tools, contact your [local sales representative](#).

#### License Checkers

If the IP requires a license key, the key must be verified. The AMD Vivado™ design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with an error. License checkpoints are enforced by the following tools:

- Vivado Synthesis
- Vivado Implementation
- write\_bitstream (Tcl command)

---

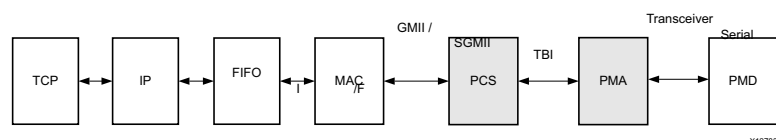
**!! Important:** IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.

---

## Product Specification

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

**Figure: 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 and 1G or 2.5G 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 or 2.5 Gbps, the full GMII is used; for a MAC operating at a speed of 10 Mbps or 100 Mbps, 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 Mbps, 100 Mbps, and 1 Gbps or 2.5Gbps. 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 or 2500BASE-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 or 2500BASE-X Auto-Negotiation for information exchange with the link partner
- 2500BASE-X PCS operation is similar to 1000BASE-X operation as defined in IEEE 802.3-2008, clauses 36 and 37

## 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. This mode is not supported for 2.5 Gbps data rates.

## 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
- 2500BASE-X PMA operation is similar to 1000BASE-X operation as defined in IEEE 802.3-2008, clauses 36

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.

- 2500BASE-X PMD operation is similar to 1000BASE-X operation as defined in IEEE 802.3-2008

## Standards

- Ethernet Standard 802.3-2008 Clauses 22, 35, 36, and 3
- Serial-GMII Specification V1.7 (CISCO SYSTEMS, ENG-46158)

## Performance

This section details the performance information for various core configurations.

### Maximum Frequencies

The core operates at 125 MHz for the 1 Gbps data rate (1.25Gbps line rate) and 312.5 MHz at 2.5 Gbps data rates (3.125 Gbps line rate) in modes having device transceivers.

### 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 AMD 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 or 2500BASE-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 Gbps or 3.125 Gbps depending on the data rate selected in Vivado IDE.

## Voltage Requirements

AMD 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](#)). AMD Kintex™ 7 devices support TBI and GMII at 3.3 V or lower. AMD Artix™ 7 and AMD Zynq™ 7000 devices support GMII at 3.3 V or lower.

## Speed Grades

AMD UltraScale+™, AMD UltraScale™, Zynq 7000 SoC, and 7 series devices support speed grade -1 and faster for GT transceiver interface.

For information about the SGMII 1000BASE-X LVDS interface, see [Synchronous SGMII over LVDS](#) and [Asynchronous 1000BASE-X/SGMII over LVDS](#).

## Resource Utilization

For full details about performance and resource use, visit the [Performance and Resource Utilization](#).

The following tables show the supported interfaces per device family for 1G and 2.5G rates.

**Table: 1G Core Family Support**

Device Family	LogiCORE IP Functionality								
	TBI	Device Specific Transceivers	LVDS Select IO	Device specific Transceivers	LVDS Select IO	Device specific Transceivers	LVDS Select IO	Device specific Transceivers	LVDS Select IO
	1000BASE-X			GMII to SGMII Bridge or SGMII to GMII Bridge			1000BASE-X and SGMII Standalone		
Versal	No	Yes	Yes	No	Yes	No	Yes	No	Yes
UltraScale	No	Yes	Yes	No	Yes	Yes	Yes	No	Yes
UltraScale+	No	Yes	Yes	No	Yes	No (Asynchronous solution can be used for Synchronous implementations)	Yes	No	Yes
Zynq 7000	No	Yes	No	No	Yes	Yes (Supported in -2 speed grades and faster for Z-7010, Z-7015, and Z-7020 devices. For other Zynq devices speed grade -1 and above are supported.)	Available through XAPP523	No	Yes
Virtex 7	No	Yes	No	No	Yes	Supported in -2 speed grade and faster parts for HR banks; -1 speed grade and faster for HP banks	Available through XAPP523	No	Yes
Kintex 7	Yes	Yes	No	Yes	Yes	Supported in -2 speed grade and faster parts for HR banks; -1 speed grade and faster for HP banks	Available through XAPP523	Yes	Yes
Artix 7	No	Yes	No	No	Yes	Supported in -2 speed grades and faster.	No	No	Yes
Spartan 7	No	No	No	No	No	Supported in -2 speed grades and faster.	No	No	No

**Table: 2.5G Core Family Support**

Device Family	LogiCORE IP Functionality				
	TBI/LVDS Select IO	Device specific Transceivers	Any Interface	TBI/LVDS Select IO	Device specific Transceivers
	2500BASE-X		2500BASE-X and 2.5G SGMII	2.5G SGMII to SGMII Bridging or 2.5G SGMII to GMII	
Versal	No	Yes	No	No	Yes
UltraScale	No	Yes	No	No	Yes
UltraScale+	No	Yes	No	No	Yes
Zynq 7000	No	Yes (excluding devices having only GTP transceivers with	No	No	Yes (excluding devices having only GTP transceivers with

Device Family	LogiCORE IP Functionality				
	TBI/LVDS SELECT IO		Device specific Transceiver	Device Interface	TBI/LVDS Select IO
	2500BASE-X		2500BASE-X and 2.5G SGMII	2.5G SGMII to SGMII Bridging	2.5G SGMII to SGMII Bridging
	-1 speed grades)				-1 speed grades)
Virtex 7	No	Yes	No	No	Yes
Kintex 7	No	Yes	No	No	Yes
Artix 7	No	Yes (for -2, -2L, and -3 speed grades)	No	No	Yes (for -2, -2L, and -3 speed grades)
Spartan 7	No	No	No	No	No

BUFG Usage

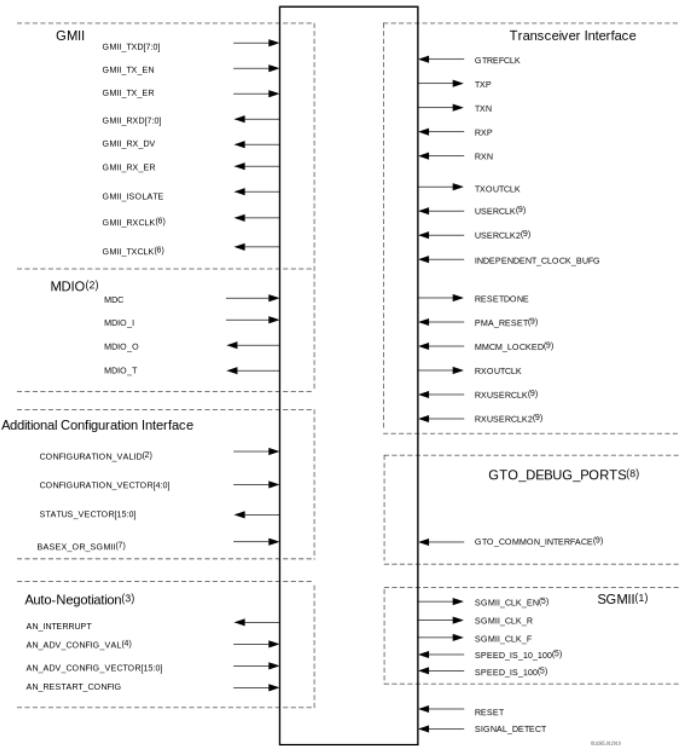
BUFG usage does not consider multiple instantiations of the core, where clock resources can often be shared.

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.

Port Descriptions

The core pinout for 1000BASE-X, 2500BASE-X, SGMII, or 2.5G SGMII or Dynamic Switching in the core using transceivers is shown in the following figure.

Figure: Core Pinout with Transceiver in the Core

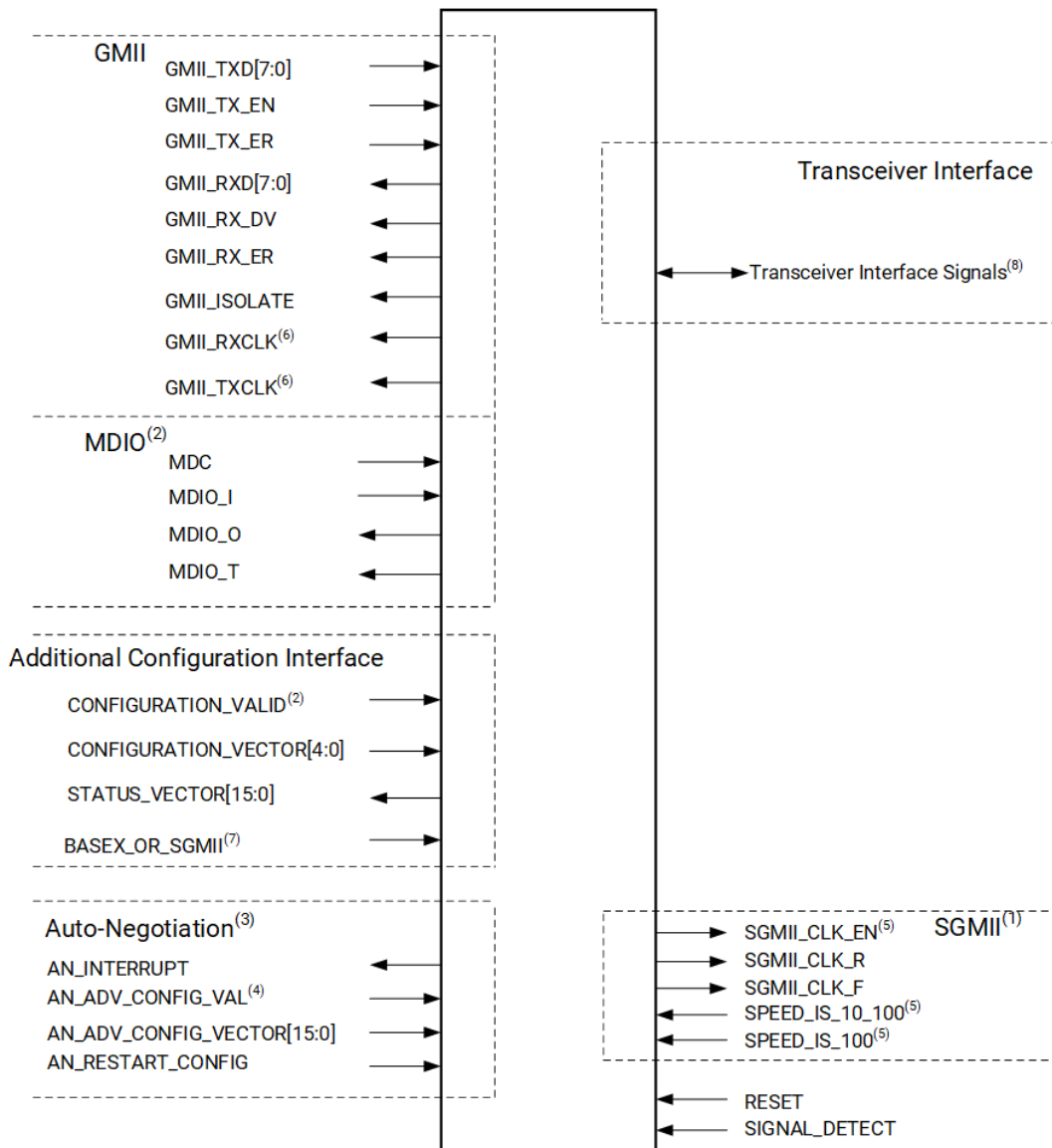


**Note:**

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 when TEMAC selected as the interface ( speed\_is\_10\_100 and speed\_is\_100 inputs are not applicable for 2.5 Gbps mode.).
6. Pins are visible only when Gigabit Ethernet MAC (GEM) selected as the interface. These pins are part of the GMII interface.
7. Pins are 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 pins changes if shared logic is part of the core. The direction shown here is when shared logic is part of the example design.

The core pinout for 1000 BASE-X, 2500BASE-X, SGMII, or 2.5G SGMII or Dynamic Switching using the transceiver in the example design is shown in the following figure.

**Figure: Core Pinout with Transceiver in Example Design**



X16216-012919

**Note:**

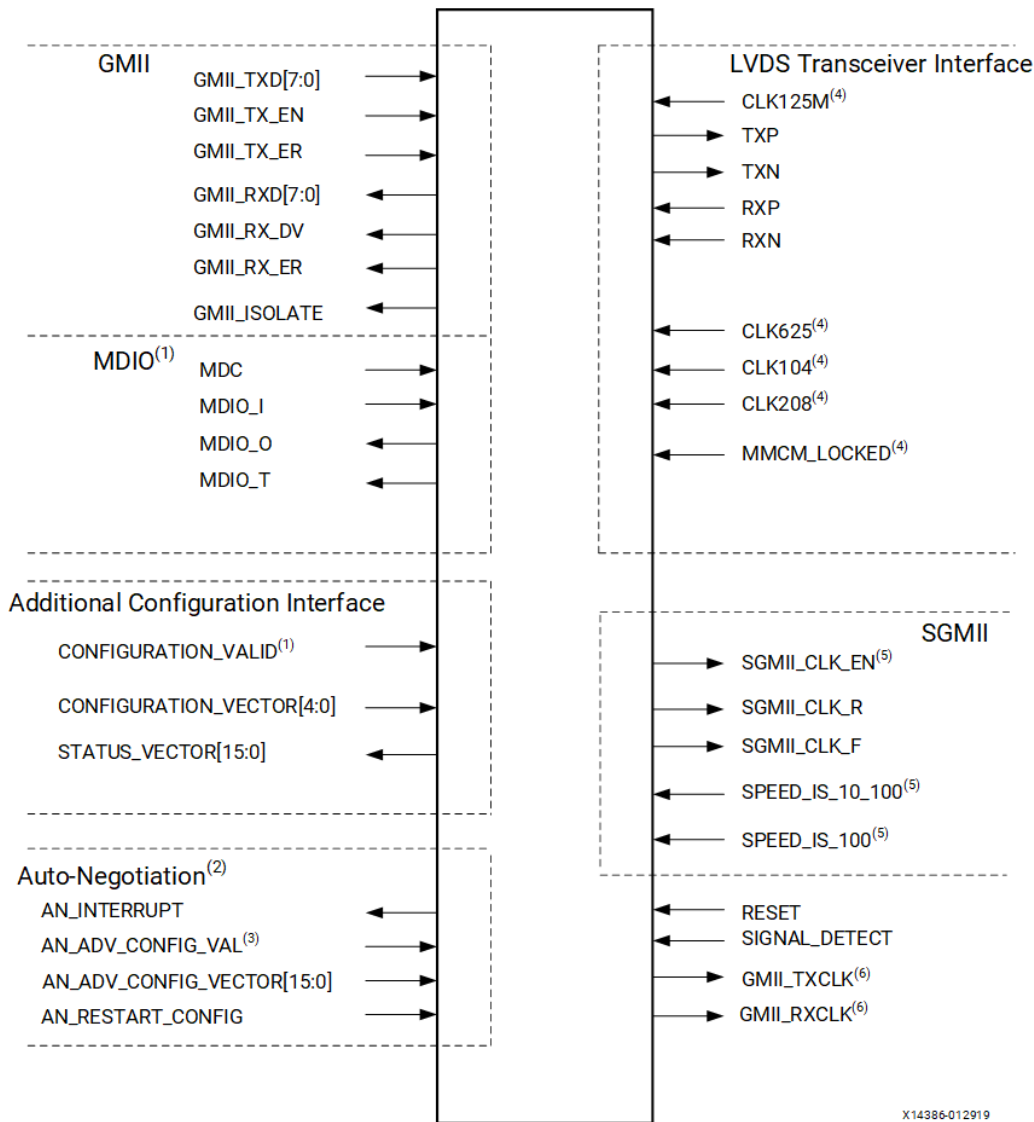
1. Pins are visible only if SGMII is enabled.
2. Pins are visible only if MDIO is enabled.
3. Pins are visible only if Auto-Negotiation is enabled.
4. Pins are visible only if Auto-Negotiation and MDIO are enabled.
5. Pins are visible only when TEMAC selected as the interface ( speed\_is\_10\_100 and speed\_is\_100 inputs are not applicable for 2.5

Gbps mode.).

6. Pins are visible only when Gigabit Ethernet MAC (GEM) selected as the interface. These pins are part of the GMII interface.
7. Pins are visible if (a) Auto-Negotiation and dynamic switching are enabled OR (b) dynamic switching and MDIO are enabled.
8. See [Table 2](#) for signals.

The core pinout for SGMII over LVDS is shown in the following figure.

**Figure: Pinout for SGMII over LVDS Mode**



#### Note:

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 if (a) Auto-Negotiation and dynamic switching are enabled OR (b) dynamic switching and MDIO are enabled.

#### GMII Ports

The following table 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: GMII Interface Signal Pinout**

Direction	Description
GMII_TXD[7:0]	GMII Transmit data from MAC.
GMII_TX_EN	GMII Transmit Enable.
GMII_TX_ER	GMII Transmit Error.
GMII_RXD[7:0]	GMII Receive data to MAC.
GMII_RX_DV	GMII Receive Data Valid.
GMII_RX_ER	GMII Receive Error.
GMII_ISOLATE	GMII Isolate.



Signal	Description
<b>gmii_tx_ctl</b> 1	GMII Transmit control signal from MAC.
<b>gmii_tx_ctl</b> 1	GMII Transmit control signal from MAC.
<b>gmii_rx_ctl</b> 2	GMII Receive control signal to MAC.
<b>gmii_rx_ctl</b> 2	GMII Receive control signal to MAC.
<b>gmii_rx_ctl</b> 2	GMII Receive control signal to MAC.
<b>gmii_isolate</b> 2	GMII Isolate control for GMII Isolation. Only of use when implementing an External GMII as shown by the block level design HDL.
<ol style="list-style-type: none"> <li>When the TX elastic buffer is present, these signals are synchronous to <code>gmii_tx_clk</code>. When the TX elastic buffer is omitted, see <a href="#">2</a>.</li> <li>These signals are synchronous to the internal 125 MHz reference clock of the core. This is <code>userclk2</code> when the core is used with the device-specific transceiver; <code>gtx_clk</code> when the core is used with TBI.</li> </ol>	

## MDIO Management Interface Ports

The following table 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: Optional MDIO Interface Signal Pinout**

Signal	Description
<b>mdio_clk</b> A	Management clock ( $\leq 2.5$ MHz).
<b>mdio_in</b> 1	Input data signal for communication with MDIO controller (for example, an Ethernet MAC). Tie High if unused.
<b>mdio_out</b> 1	Output data signal for communication with MDIO controller (for example, an Ethernet MAC).
<b>mdio_ctl</b> 1	Output control for MDIO signals; A value of 0 indicates that the value on <code>mdio_out</code> should be asserted onto the MDIO interface.
<b>mdio_addr</b> A	Physical Address of the PCS management register set.
<ol style="list-style-type: none"> <li>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).</li> </ol>	

## Reset Ports

**Table: Reset Signals Pinout**

Signal	Description
<b>reset</b> a	Asynchronous reset for the entire core. Active-High
<b>mdio_ctl</b> 1	Indicates completion of the gtwizard reset sequence. In cases where the transceiver is not present, this signal is tied to 1.

## Dynamic Switching Signal Port

The following table describes the signals present when the optional dynamic switching mode (between 1000BASE-X and SGMII

standards) is selected. In this case, the MDIO ([MDIO Management Interface Ports](#)) and device-specific [Transceiver Ports](#) are always present.

**Table: Optional Dynamic Switching Signals**

Signal	Description
<a href="#">userclk2</a>	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.
1. Clock domain is userclk2.	

### DRP and 1588 Support Ports

The following tables 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 *7 Series FPGAs Transceivers Wizard LogiCORE IP Product Guide* ([PG168](#)) for more details on the DRP signals.

**Table: DRP Interface to Transceiver Ports**

Signal	Description
<a href="#">drp_clk</a>	DRP interface clock, tied to gtrefclk_bufg for 7 series/Zynq devices and independent_clock_bufg for UltraScale+/UltraScale devices.
<a href="#">drp_req</a>	DRP request
<a href="#">drp_grnt</a>	DRP grant
<a href="#">drp_ena</a>	DRP enable signal
<a href="#">drp_wre</a>	DRP write enable
<a href="#">drp_done</a>	Indicates DRP operation is complete
<a href="#">drp_addr[8:0]</a>	DRP address
<a href="#">drp_data[15:0]</a>	DRP data from transceiver
<a href="#">drp_data[5:0]</a>	DRP data to transceiver
1. Signals are synchronous to gtrefclk for 7 series devices and independent_clock for UltraScale+/UltraScale devices.	

**Table: 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
correction_timer[63:0]	In	Correction timer
rxphy_correction_timer[63:0]	Out	Modified correction timer

### Configuration and Status Vector Ports

The following table describe the ports that are used to configure and monitor the core if the MDIO interface is not used.

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

Signal	Direction	Description
status_vector[0:15:0]	Input	See <a href="#">Table 3</a> for bit description.
configuration_vector[4:0]	Input	Additional interface to program management Register 0 irrespective of the optional MDIO interface. See <a href="#">Table 1</a> for bit 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 <a href="#">Auto-Negotiation</a> . See <a href="#">Table 2</a> for bit description.
an_adv_config_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 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 of 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

The following table 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 [Asynchronous LVDS Transceiver for Versal devices](#).

**Table: Optional TBI Interface Signal Pinout**

Signal	Domain	Description
gtx_clk	A	Signal at 125 MHz. Tolerance must be within IEEE 802.3-2008 specification.
tx_data[0:15:0]	A	Output data to PMA Sublayer (SerDes).
tx_clk	A	Output of the PMA sublayer clock recovery unit to lock to pma_tx_clk. This signal is tied to ground.
loopback	A	When 1, this indicates to the external PMA SerDes device to enter loopback mode. When 0, this indicates normal operation.
rx_data[0:15:0]	A	Input data from PMA Sublayer (SerDes). This is synchronous to pma_rx_clk0.
rx_data[16:31:0]	A	Input data from PMA Sublayer (SerDes). This is synchronous to pma_rx_clk1.
pma_rx_clk0	A	Input clock signal from PMA Sublayer (SerDes) at 62.5 MHz.
pma_rx_clk1	A	Input clock signal from PMA Sublayer (SerDes) at 62.5 MHz. This is 180° out of phase with pma_rx_clk0.

Signal	Direction	Description
gtrfclk	Input	Provides the PMA Sublayer to perform comma realignment. This is driven from the PCS Receive Engine during the <i>Loss-Of-Sync</i> state.

### Transceiver Ports

The following table shows the transceiver interface ports for the case when Shared Logic is included in the example design and the transceiver is in the core.

**Table: Transceiver Interface - Transceiver in Core**

Signal	Direction	Description
gtrefclk	Input	125 MHz reference clock from IBUFDS to the transceiver for 7 series and Zynq devices. Selectable in GUI for UltraScale+/UltraScale devices.
gtrefclk_bufg	Input	Reference clock for transceiver which is passed through a BUFG used to drive logic. This is applicable only for 7 series and Zynq devices.
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 <a href="#">1</a> , <a href="#">2</a> , <a href="#">3</a>	Input	Stable clock in transceiver and also as control clock for IDELAYCTRL. GT Reset controller free running clock in Versal devices.
resetdone	Output	Indication that reset sequence of the transceiver is complete
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 sgmmi_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

Signal	Direction	Description
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 Finite State Machine (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 nonArtix 7 families. Indicates reference out clock from PLL of GT Common

- For 7 series and Zynq 7000 devices 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 (in ns). Also, make sure that if the design is using IDELAYCTRL then the value given to this clock is either
  - within the range of the refclk value specified for IDELAYCTRL or
  - a different clock is used as the reference clock for IDELAYCTRL.
 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 the stable clock period is 4 to 250 ns. The core has only been tested with 5 ns (that is, 200 MHz).

2. For UltraScale+/UltraScale devices when the transceiver debug signals are not selected then this is selectable in the Vivado IDE or through the parameter, DrpClkRate. The clock should be free-running and the range for this clock is 6.25-62.5 MHz for the 1 Gbps data rate and 6.25-156.25 for the 2.5 Gbps data rate. When the transceiver debug signals are selected then this can be any frequency depending on the UltraScale+/UltraScale gtwizard recommendation. In this case DrpClkRate corresponds to the Drp Clock port input. Its recommended to keep drp clock and the independent clock to be the same in this case because the core has been tested only for this combination.

- For AMD Versal™ devices, the independent clock frequency need not be the same as GT's APB3 clock frequency. Hence, both the clocks can be provided in accordance with their specifications.

The following table describes the interface to the transceiver when Shared Logic is included in the Example design and the transceiver is outside the core.

**Table: Transceiver Interface - Transceiver Outside Core**

Signal	Direction	Description
gtwiz_userclk_tx_active_out	Output	Connect to gtwiz_u of GT wizard
gtwiz_reset_clk_freerun_out	Output	Connect to gtwiz_re of GT wizard
gtwiz_reset_tx_datapath_out	Output	Connect to gtwiz_re of GT wizard

Signal	Direction	Description
gtwiz_reset_rx_datapath_out	Output	Connected to gtwiz_reset_rx_datapath_out of GT wizard.
gtwiz_reset_all_out	Output	Connected to gtwiz_reset_all_out of GT wizard.
gtwiz_userclk_rx_active_out	Output	Connected to gtwiz_userclk_rx_active_out of GT wizard.
gtwiz_reset_tx_pll_and_datapath_out	Output	Connected to gtwiz_reset_tx_pll_and_datapath_out of GT wizard.
gtwiz_reset_rx_pll_and_datapath_out	Output	Connected to gtwiz_reset_rx_pll_and_datapath_out of GT wizard.
gtwiz_reset_tx_done_in	Input	Connected to gtwiz_reset_tx_done_in of GT wizard.
gtwiz_reset_rx_done_in	Input	Connected to gtwiz_reset_rx_done_in of GT wizard.
gtwiz_buffbypass_rx_reset_out	Output	Connected to gtwiz_buffbypass_rx_reset_out of GT wizard. Valid when SGMII with fabric elastic buffer or when RX.

Signal	Direction	Descripti
		path is on rxuserc or the 1588 solution is selecte
gtwiz_buffbypass_rx_start_user_out	Output	Connec to gtwiz_b of GT wizard. Valid when SGMII with fabric elastic buffer or when RX path is on rxuserc or the 1588 solution is selecte
gtwiz_buffbypass_rx_done_in	Input	Connec to gtwiz_b of GT wizard. Valid when SGMII with fabric elastic buffer or when RX path is on rxuserc or the 1588 solution is

Signal	Direction	Description
		selected
gtwiz_buffbypass_tx_reset_out	Output	Connects to gtwiz_buffbypass_tx_reset_out of GT wizard. Valid only when the 1588 solution is selected.
gtwiz_buffbypass_tx_start_user_out	Output	Connects to gtwiz_buffbypass_tx_start_user_out of GT wizard. Valid only when the 1588 solution is selected.
gtwiz_buffbypass_tx_done_in	Input	Connects to gtwiz_buffbypass_tx_done_in of GT wizard. Valid only when the 1588 solution is selected.
rxpmaresetdone_in	Input	Connects to rxpmaresetdone_in of GT wizard.
txresetdone_in	Input	Connects to txresetdone_in of GT wizard.
rxresetdone_in	Input	Connects to rxresetdone_in of



Signal	Direction	Description
		GT wizard
rxmcommaalignen_out	Output	Connects to rxmcommaalignen of GT wizard
rxpcommaalignen_out	Output	Connects to rxpcommaalignen of GT wizard. For Versal devices this output is connected to gpi[15:0] input port of GTY/GTYP. For the bit position connect refer IP example design or AM002-versal-gty-transceiver user guide.
txelecidle_out	Output	Connects to txelecidle of GT wizard
txpd_out[1:0]	Output	Connects to txpd_in of GT wizard
rxpd_out[1:0]	Output	Connects to rxpd_in

Signal	Direction	Description
		of GT wizard
rxusrclk_out	Output	Connect to rxusrclk of GT wizard
rxusrclk2_out	Output	Connect to rxusrclk of GT wizard
txusrclk_out	Output	Connect to txusrclk of GT wizard
txusrclk2_out	Output	Connect to txusrclk of GT wizard
txctrl0_out[15:0]	Output	Connect to txctrl0 of GT wizard
txctrl1_out[15:0]	Output	Connect to txctrl1 of GT wizard
txctrl2_out[7:0]	Output	Connect to txctrl2 of GT wizard
gtwiz_userdata_tx_out[15:0]	Output	Connect to gtwiz_u of GT wizard

Signal	Direction	Description
rxctrl0_in[15:0]	Input	Connects to rxctrl0_in of GT wizard
rxctrl1_in[15:0]	Input	Connects to rxctrl1_in of GT wizard
rxctrl2_in[7:0]	Input	Connects to rxctrl2_in of GT wizard
rxctrl3_in[7:0]	Input	Connects to rxctrl3_in of GT wizard
rxclkcorcnt_in[1:0]	Input	Connects to rxclkcorcnt_in of GT wizard for 1000BASE-X mode or SGMII mode using GT elastic buffer; otherwise 0.
gtwiz_userdata_rx_in[15:0]	Input	Connects to gtwiz_userdata_rx_in of GT wizard
rxbufstatus_in[2:0]	Input	Connects to rxbufstatus_in for non-1588 designs and in

Signal	Direction	Description
		1000BASE-X mode or SGMII mode using GT elastic buffer; otherwise 0.
txbufstatus_in[1:0]	Input	Connects to txbufstatus for non-1588 designs; otherwise 0.
cpplllock_in	Input	Connects to cpplllock of GT wizard.
rx8b10ben_out	Output	Connects to rx8b10ben of GT wizard.
tx8b10ben_out	Output	Connects to tx8b10ben of GT wizard.
rxcommadeten_out	Output	Connects to rxcommadeten of GT wizard.

The following table describes the interface to the transceiver when Shared Logic is included in the core.

**Table: Transceiver Interface with Shared Logic in the Core**

Signal	Direction	Description
gtrefclk_p	Input	125 MHz differential reference clock to IBUFDS for

Signal	Direction	Descripti
		7 series and Zynq devices Selecta in GUI forUltra UltraSc: UltraSc: devices
gtrefclk_n	Input	125 MHz differen referenc clock to IBUFDS for 7 series and Zynq devices Selecta in GUI for UltraSc: UltraSc: devices
gtrefclk_out	Output	125 MHz referenc clock from IBUFDS for 7 series and Zynq devices Selecta in GUI for UltraSc: UltraSc: devices
gtrefclk_bufg_out	Output	Referen clock for transce which is passed through a BUFG

Signal	Direction	Description
		used to drive logic. This is applicable for 7 series and Zynq devices
txp	Output	Transmit differential
txn	Output	Transmit differential
rxp	Input	Receive differential
rxn	Input	Receive differential
userclk_out	Output	Also connect to txusrclk of the device-specific transceiver. Clock domain is not applicable
userclk2_out	Output	Also connect to txusrclk of the device-specific transceiver. Clock domain is not applicable
rxuserclk_out	Output	Also connect to rxusrclk of the device-specific transceiver

Signal	Direction	Description
		Clock domain is not applicable.
rxuserclk2_out	Output	Also connects to rxusrclk2 of the device-specific transceiver. Clock domain is not applicable.
independent_clock_bufg <a href="#">1</a> , <a href="#">2</a>	Input	Stable clock in transceiver and also as control clock for IDELAYCTRL.
resetdone	Output	Indicates that reset sequence of the transceiver is complete.
pma_reset_out	Output	Hard reset synchronizes to independent clock.
mmcm_locked_out	Output	Indicates that the MMCM outputs are stable.
gmii_txclk	Output	Applicable only when GEM is selected.

Signal	Direction	Descripti
		as the interface type. This is the looped back version of userclk in BASE-X mode and the same as sgmiic 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 output clock from PLL0 of GT Common
gt0_pll0outrefclk_out	Output	Valid only for Artix 7 families. Indicates reference output clock from PLL0 of GT Common
gt0_pll1outclk_out	Output	Valid only



Signal	Direction	Description
		for Artix 7 families. Indicates output clock from PLL1 of GT Common.
gt0_pll1outrefclk_out	Output	Valid only for Artix 7 families. Indicates reference output clock from PLL1 of GT Common.
gt0_pll0lock_out	Output	Valid only for Artix 7 families. Indicates PLL0 of GT Common has locked.
gt0_pll0refclklost_out	Output	Valid only for Artix 7 families. Indicates output reference clock for PLL0 of GT Common is lost.
gt0_qplloutclk_out	Output	Valid only

Signal	Direction	Description
		for non Artix 7 families. Indicate out clock from PLL of GT Common
gt0_qplloutrefclk_out	Output	Valid only for non Artix 7 families. Indicate reference out clock from PLL of GT Common
<p>1. For 7 series and Zynq 7000 the example design assumes independent_clock_bufg to be 200 MHz. If it is different, the period should be changed in the &lt;component_name&gt;_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</p> <ol style="list-style-type: none"> <li>either within the range of the refclk value specified for IDELAYCTRL. OR</li> <li>Some different clock is used as reference clock for IDELAYCTRL.</li> </ol> <p>Changing the value for STABLE_CLOCK_PERIOD to anything other than 5 might require changing the WAIT_TIMEOUT_2ms counter value in the &lt;component_name&gt;_tx_startup_fsm.v[hd] and &lt;component_name&gt;_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).</p> <p>2. For UltraScale+/UltraScale devices this is selectable through the Vivado IDE or through parameter DrpClkRate. The clock should be free-running and the range for this clock is 6.25-62.5 MHz for 1 Gbps data rate and 6.25-156.25 for 2.5 Gbps data rate.</p>		

## SGMII Ports

The following table describes the SGMII interface ports.

**Table: SGMII Interface Ports**

Signal	Direction	Description
sgmii_clk_en	Output	Clock for GMII transmit data
sgmii_clk_f	Output	Differential clock for GMII

Signal	Direction	Description
		transmission data
sgmii_clk_r	Output	Differential clock for GMII transmission data
recclk_mmcm_reset	Output	MMCM reset for MMCM generating rxuserclk when RxGmii is output only when the clocking logic is a part of the example design and applicable only for 7 series devices with MMCM is used for clock multipli
sgmii_rx_clk_en	Output	Clock enable for GMII receive data when RxGmii is in SGMII mode.
sgmii_rx_clk_f	Output	Differential clock for

Signal	Direction	Descripti
		GMII receive data when RxGmii in SGMII mode.
sgmii_rx_clk_r	Output	Differen clock for GMII receive data when RxGmii in SGMII mode.
speed_is_10_100	Input	Speed control for control operati speed of SGMII interfac Not applicat for 2.5G SGMII. Valid Values: 0 for 1 Gbps 1 for 10/100 Mbps
speed_is_100	Input	Speed control for control operati speed of SGMII interfac Not applicat for 2.5G SGMII. Valid Values: 0

Signal	Direction	Description
		for 1 Gbps and 10 Mbps 1 for 100 Mbps

### Synchronous SGMII over LVDS Transceiver Interface Ports

The following table shows the physical side interface ports for SGMII over LVDS when Shared Logic is included in the Example Design.

**Table: 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	Transmi differen
txn	Output	Transmi differen
rxp	Input	Receive differen
rxn	Input	Receive differen
clk104	Input	104 MHz clock derived from 125MHz input differen clock
clk208	Input	208 MHz clock derived from 125MHz input differen clock
clk625	Input	625 MHz clock derived from 125MHz input differen

Signal	Direction	Description
mmcm_locked	Input	Indicates when the MMCM outputs are stable

The following table describes the physical side interface ports when the core is configured with SGMII over LVDS when Shared Logic is included in the core.

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

Signal	Direction	Description
refclk125_p	Input	Differential 125 MHz clock synchronous to incoming SGMII serial data
refclk125_n	Input	Differential 125 Mhz 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.

Signal	Direction	Description
mmcm_locked_out	Output	MMCM locked indicating
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.

### Asynchronous SGMII over LVDS Transceiver Interface Ports

The following tables show the physical side interface ports for Asynchronous SGMII over LVDS using Ultrascale Select IO logic or Versal Advanced IO Wizard when the shared logic is included in example design.

**Table: Ultrascale/Ultrascale+ Asynchronous LVDS Transceiver Ports**

Signal	Direction	Description
clk125m	Input	125 MHz Core clock
clk312	Input	312 MHz Referen clock which is used by the RX Datapath of LVDS Transce block
tx_bsc_rst	Input	Reset the BITSLIC Primitives controlling the TX Nibble
rx_bsc_rst	Input	Reset the BITSLIC Primitives controlling the RX Nibble

Signal	Direction	Description
tx_bs_rst	Input	Reset for TX_BITS primitive used in the TX nibble
rx_bs_rst	Input	Reset for RX_BITS primitive used in the RX nibble
tx_rst_dly	Input	Resets the internal delay lines used by TX_BITS primitive within the TX nibble
rx_rst_dly	Input	Resets the internal delay lines used by RX_BITS primitive within the RX nibble
tx_rdclk	Input	156.25 Mhz clock used by TX gearbox whose 8 bit data output is synchro to



Signal	Direction	Description
		this clock
tx_pll_clk	Input	PLL clock which is used to clock the Transmitter path BITSLIC Primitives
rx_pll_clk	Input	PLL clock which is used to clock the Receiver path BITSLIC Primitives
tx_bsc_en_vtc	Input	Enable Voltage Temperature and Process compensation for BITSLIC Primitives controlling the TX datapath
rx_bsc_en_vtc	Input	Enable Voltage Temperature and Process compensation for BITSLIC Primitives controlling the TX datapath
tx_bs_en_vtc	Input	Enable Voltage Temperature and Process compensation for

Signal	Direction	Description
		TX_BITS primitives
rx_btval[8:0]	Input	This value provided by the user is used to calculate the number of taps required to maintain a relative delay of 400 ps between the two RX_BITS primitives
tx_dly_ready	Output	Asserted High to indicate that delay line calibration is complete for TX_BITS primitives
rx_dly_ready	Output	Asserted High to indicate that delay line calibration is complete for RX_BITS primitives
tx_vtc_ready	Output	Asserted High to indicate

Signal	Direction	Description
		that the TX_BITS primitive is ready for Voltage Temper and Process comper
rx_vtc_ready	Output	Asserted High to indicate that the RX_BITS primitive is ready for Voltage Temper and Process comper

Table: Versal Asynchronous LVDS Transceiver Ports

Signal	Direction	Description
clk125m	Input	125 MHz core clock
pll_rst_in	Input	XPLL Reset
pll_clk_in	Input	XPLL Referen clock
ctrl_clk	Input	Referen clock used by the XPHY for RIU interfac Delay lines and Built in Self Calibrat (BISC).

Signal	Direction	Description
pll_locked_out	Output	Indicates whether the XPLL is Locked
dly_ready	Output	DLY_RE output port of the XPHY primitive. Indicates that delay line values can be changed
vtc_ready	Output	VTC_RE output port of the XPHY primitive. Indicates that XPHY is ready for voltage temper. compar

The following are the physical interface ports when the core is configured with the include shared logic in core option. The below ports are only applicable to Ultrascale and Ultrascale + devices.

**Table: Ultrascale/Ultrascale+ Asynchronous LVDS Transceiver Ports with Shared Logic Included in Core**

Signal	Direction	Description
tx_dly_ready<n>	Input	Asserted HIGH if TX nibble Delay line calibration is complete for the nth core instance

Signal	Direction	Description
rx_dly_ready<n>	Input	Asserted HIGH if RX nibble Delay line calibration is complete for the nth core instance
tx_vtc_ready<n>	Input	Asserted HIGH if TX nibble of the nth core instance is ready for Voltage Temperature and Process compensation
rx_vtc_ready<n>	Input	Asserted HIGH if RX nibble of the nth core instance is ready for Voltage Temperature and Process compensation
clk312_out	Output	312Mhz clock that is used by the RX datapath

Signal	Direction	Description
		of the LVDS Transceiver block. To be shared with other core instances
clk125_out	Output	125MHz Reference clock. To be shared with other core instances
rx_logic_reset	Output	To be combined with tx_logic_reset using an OR gate to reset other core instances
tx_logic_reset	Output	--
tx_locked	Output	LOCKED output of the PLL that is being used to clock the BISLICE primitive on the TX datapath
rx_locked	Output	LOCKED output of

Signal	Direction	Description
		the PLL that is being used to clock the BISLICE primitiv on the RX datapath
tx_bsc_rst_out	Output	To be connected to tx_bsc_ input port, of cores without shared logic
rx_bsc_rst_out	Output	To be connected to rx_bsc_ input port, of cores without shared logic
rx_bs_rst_out	Output	To be connected to rx_bs_r_ input port, of cores without shared logic
tx_bs_rst_out	Output	To be connected to tx_bs_r_ input port, of cores

Signal	Direction	Description
		without shared logic
tx_rst_dly_out	Output	To be connected to tx_rst_c input port, of cores without shared logic
rx_rst_dly_out	Output	To be connected to rx_rst_c input port, of cores without shared logic
tx_bsc_en_vtc_out	Output	To be connected to tx_bsc_ input port, of cores without shared logic
rx_bsc_en_vtc_out	Output	To be connected to rx_bsc_ input port, of cores without shared logic
tx_bsc_en_vtc_out	Output	To be connected to tx_bsc_ input port, of



Signal	Direction	Description
		cores without shared logic
rx_bsc_en_vtc_out	Output	To be connected to rx_bsc_ input port, of cores without shared logic
tx_pll_clk_out	Output	To be connected to tx_pll_c_ input port, of cores without shared logic
rx_pll_rst_out	Output	To be connected to rx_pll_c_ input port, of cores without shared logic
tx_rdclk_out	Output	156.25MHz reference clock for cores without shared logic.
rx_btval<n>[8:0]	Output	To be connected to rx_btva_ input port, of cores without shared

Signal	Direction	Description
		logic

RIU Interface Ports

RIU interface is used by BITSLICE\_CONTROL primitives in Ultrascale and Ultrascale+ devices as well as by the XPHY in Versal devices.

Table: RIU Interface Ports

Signal	Direction	Description
riu_clk	Input	Clock from interconnect logic. The RIU clock must be connected in for the BISC process to complete (ctrl_clk). Is used for clocking RIU in Versal devices.
riu_addr	Input	Register Address
riu_wr_data[15:0]	Input	Data write to register
riu_rd_data[15:0]	Output	Data read from register
riu_valid	Output	Asserts when the user has control over the RIU bus.

Signal	Direction	Description
riu_wr_en	Input	Active high register write enable.
riu_nibble_sel	Input	Nibble in byte select. This signal must be High to perform read/write to the nibble.

### Transceiver Control and Status Debug Ports

The following tables 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 input ports in the Transceiver Control And Status Interface must be driven in accordance with the appropriate GT user guide. Using the input signals listed in the following tables might result in unpredictable behavior of the IP core.

**!! 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: Transceiver Control and Status Ports (7 Series and Zynq 7000 Devices)**

Signal	Direction	Clock Domain	Description
gt0_drp_addr_in[8:0]	Input	gt0_drpclk_in	DRP address bus
gt0_drpi_in[15:0]	Input	gt0_drpclk_in	Data bus for writing configuration data to the transceiver.
gt0_drpo_out[15:0]	Output	gt0_drpclk_in	Data bus for reading configuration data from the transceiver.
gt0_drprdy_out	Output	gt0_drpclk_in	Indicates operation is complete for write operations and data is valid for read operations.
gt0_drpwe_in	Input	gt0_drpclk_in	DRP write enable
gt0_drpclk_in	Input	N/A	DRP Clock
gt0_rxchariscomma_out[1:0]	Output	userclk2 for non-1588 mode, rxuserclk2 for when 1588 enabled.	GT Status
gt0_rxcharisk_out[1:0]	Output	userclk2 for non-1588 mode, rxuserclk2 for when 1588 enabled.	
gt0_rxbyteisaligned_out	Output	rxuserclk2	

Signal	Direction	Clock Domain	Description
gt0_rxbyterealign_out	Output	rxuserclk2	GT TX Driver
gt0_rxcommadet_out	Output	rxuserclk2	
gt0_txdiffctrl_in[3:0]	Input	Asynchronous	
gt0_txpostcursor_in[4:0]	Input	Asynchronous	
gt0_txprecursor_in[4:0]	Input	Asynchronous	
gt0_txpolarity_in	Input	txusrclk2	GT Polarity
gt0_rxpolarity_in	Input	rxusrclk2	GT PRBS
gt0_txprbsel_in[2:0]	Input	txusrclk2	
gt0_txprbsforceerr_in	Input	txusrclk2	
gt0_rxprbscntreset_in	Input	rxusrclk2	
gt0_rxprbserr_out	Output	rxusrclk2	
gt0_rxprbsel_in[2:0]	Input	rxusrclk2	GT Loopback Loopback is not supported by the core when RxGmiiClkSrc=RXOUTCLK.
gt0_loopback_in[2:0]	Input	Asynchronous	
gt0_txresetdone_out	Output	txusrclk2	
gt0_rxresetdone_out	Output	rxusrclk2	
gt0_rxdisperr_out[3:0]	Output	userclk2 for non-1588 mode, rxuserclk2 for when 1588 enabled.	
gt0_rxnotintable_out [1:0]	Output	userclk2 for non-1588 mode, rxuserclk2 for when 1588 enabled.	GT Status
gt0_eyescanreset_in[3:0]	Input	Asynchronous	
gt0_eyes candataerror_out	Output	Asynchronous	
gt0_eyes cantrigger_in	Input	rxusrclk2	
gt0_rxcdrhold_in	Input	Asynchronous	GT CDR
gt0_rxcdrlock_out	Output	Asynchronous	GT GTX/GTH RX Decision Feedback Equalizer (DFE)
gt0_rxlpmen_in	Input	Asynchronous	
gt0_rxdfelpmreset_in	Input	Asynchronous	
gt0_rxdfeagcovrden_in	Input	rxusrclk2	
gt0_rxmonitorout_out[6:0]	Output	Asynchronous	
gt0_rxmonitorsel_in[1:0]	Input	Asynchronous	GT TX-PMA Reset
gt0_txpmareset_in	Input	Asynchronous	
gt0_txpcsreset_in	Input	Asynchronous	GT TX-PCS Reset
gt0_rxpmareset_in	Input	Asynchronous	GT RX-PMA Reset
gt0_rxpcsreset_in	Input	Asynchronous	GT RX-PCS Reset
gt0_rxbufreset_in	Input	Asynchronous	GT receive elastic buffer Reset
gt0_rxpmaresetdone_out	Output	Asynchronous	GT PMA resetdone indication

Signal	Direction	Clock Domain	Description
gt0_txbufstatus_out[1:0]	Output	txusrclk2	GT TX Buffer status
gt0_rxbufstatus_out[2:0]	Output	rxusrclk2	GT RX Buffer status
gt0_dmonitorout_out[16:0]	Output	Asynchronous	GT Status. If width differs for particular family then LSBs valid.
gt0_rxlpmreset_in	Input	Asynchronous	RX LPM reset. Valid only for GTP.
gt0_rxlpmhfoverden_in	Input	Asynchronous	RX LPM-HF override enable. Valid only for GTP.
gt0_txinhibit_in	Input	txusrclk2	Active-High signal forces TX output to steady state.

Table: Transceiver Control and Status Ports (UltraScale Devices)

Signal	Direction	Clock Domain	Description
gt_drp_addr_in[8:0]	Input	gt_drpclk_in	DRP address bus
gt_drpi_in[15:0]	Input	gt_drpclk_in	Data bus for writing configuration data to the transceiver.
gt_drpo_out[15:0]	Output	gt_drpclk_in	Data bus for reading configuration data from the transceiver.
gt_drprdy_out	Output	gt_drpclk_in	Indicates operation is complete for write operations and data is valid for read operations.
gt_drpwe_in	Input	gt_drpclk_in	DRP write enable.
gt_drpclk_in	Input	N/A	DRP Clock. For UltraScale+/ UltraScale devices this must be the same value as selected through the Vivado IDE or passed through the DrpClkRate parameter at generation time.
gt_rxcommadet_out	Output	rxusrclk2	GT TX Driver
gt_txdiffctrl_in[3:0]	Input	Asynchronous	
gt_txpostcursor_in[4:0]	Input	Asynchronous	
gt_txprecursor_in[4:0]	Input	Asynchronous	
gt_txpolarity_in	Input	txusrclk2	GT Polarity
gt_rxpolarity_in	Input	rxusrclk2	
gt_txprbsel_in[2:0]	Input	txusrclk2	GT PRBS
gt_txprbsforceerr_in	Input	txusrclk2	
gt_rxprbscntreset_in	Input	rxusrclk2	
gt_rxprbserr_out	Output	rxusrclk2	
gt_rxprbsel_in[2:0]	Input	rxusrclk2	
gt_loopback_in[2:0]	Input	Asynchronous	GT Loopback Loopback is not supported by the core when

Signal	Direction	Clock Domain	Description
			RxGmiiClkSrc=RXOUTCLK.
gt_txresetdone_out	Output	txusrclk2	GT Status
gt_rxresetdone_out	Output	rxusrclk2	
gt_rxdisperr_out[1:0]	Output	userclk2 for non-1588 mode, rxuserclk2 for when 1588 enabled.	
gt_rxnotintable_out[1:0]	Output	userclk2 for non-1588 mode, rxuserclk2 for when 1588 enabled.	
gt_eyescanreset_in[3:0]	Input	Asynchronous	GT Eye Scan
gt_eyes candataerror_out	Output	Asynchronous	
gt_eyes cantrigger_in	Input	rxusrclk2	
gt_rxcdrhold_in	Input	Asynchronous	GT CDR
gt_rxcdrlock_out	Output	Asynchronous	
gt_rxlpmen_in	Input	Asynchronous	GT GTX/GTH RX Decision Feedback Equalizer (DFE)
gt_rxdfelpmreset_in	Input	Asynchronous	
gt_txpmarereset_in	Input	Asynchronous	GT TX-PMA Reset
gt_txpcsreset_in	Input	Asynchronous	GT TX-PCS Reset
gt_rxpmarereset_in	Input	Asynchronous	GT RX-PMA Reset
gt_rxpcsreset_in	Input	Asynchronous	GT RX-PCS Reset
gt_rxbufreset_in	Input	Asynchronous	GT Receive Elastic Buffer Reset
gt_rxpmaresetdone_out	Output	Asynchronous	GT PMA resetdone indication
gt_txbufstatus_out[1:0]	Output	txusrclk2	GT TX Buffer status
gt_rxbufstatus_out[2:0]	Output	rxusrclk2	GT RX Buffer status
gt_dmonitorout_out[16:0]	Output	Asynchronous	GT Status
gt_txinhibit	Input	txusrclk2	Active-High signal forces TX output to steady state.
gt_pcsrsvdin	Input	Asynchronous	See the <i>UltraScale Architecture GTH Transceivers User Guide (UG576)</i> and the <i>AMD UltraScale™ FPGAs Transceivers Wizard LogiCORE IP Product Guide(PG182)</i> for details.
gt_cpllrefclkssel[2:0]	Input	Asynchronous	
gt_gtrefclk1	Input	N/A	

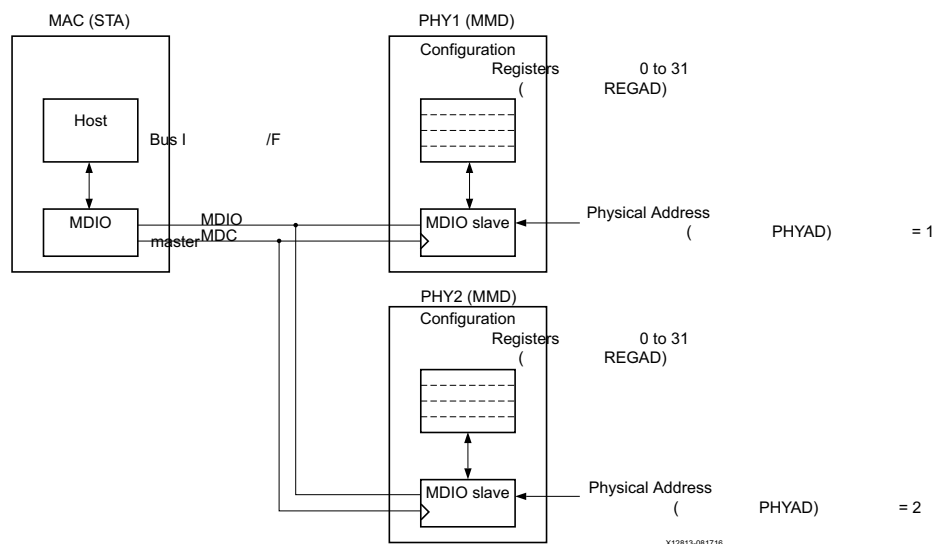
## Register Space

This section provides general guidelines for configuring and monitoring the core, including a detailed description of the core management registers. It also describes the 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).

The MDIO interface for 1 Gbps operation (and slower speeds) is defined in IEEE 802.3-2008, clause 22. The following figure 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).



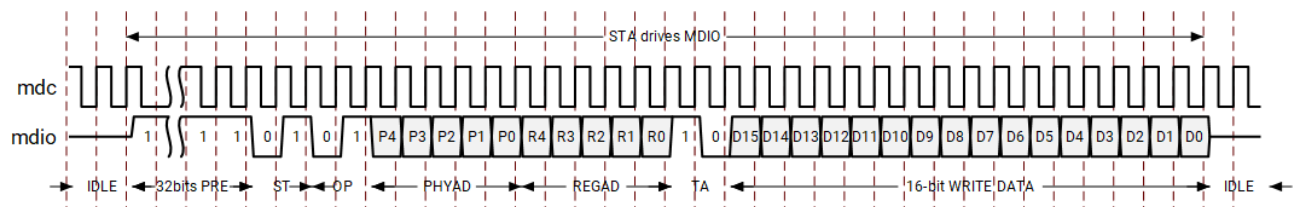
The MDIO bus system is a standardized interface for accessing the configuration and status registers of Ethernet PHY devices. In the example shown, the Management Host Bus I/F of the Ethernet MAC is able to access the configuration and status registers of two PHY devices using the MDIO bus.

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 used in this section are explained in the following table.

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

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

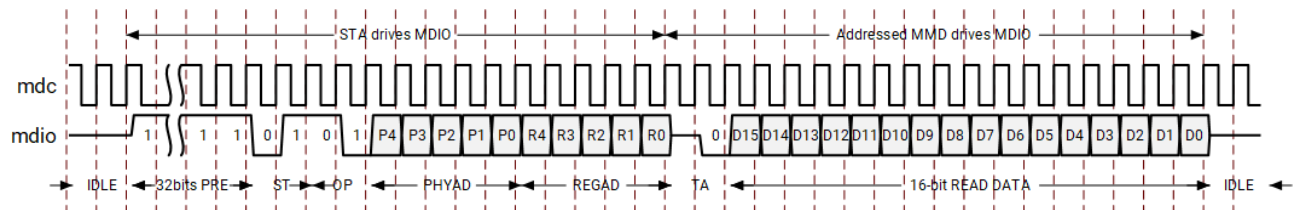
### Figure: MDIO Write Transaction



### Read Transaction

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

**Figure: MDIO Read Transaction**



### MDIO Addressing

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

#### Physical Address (PHYAD)

As shown in [MDIO Bus System](#), 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

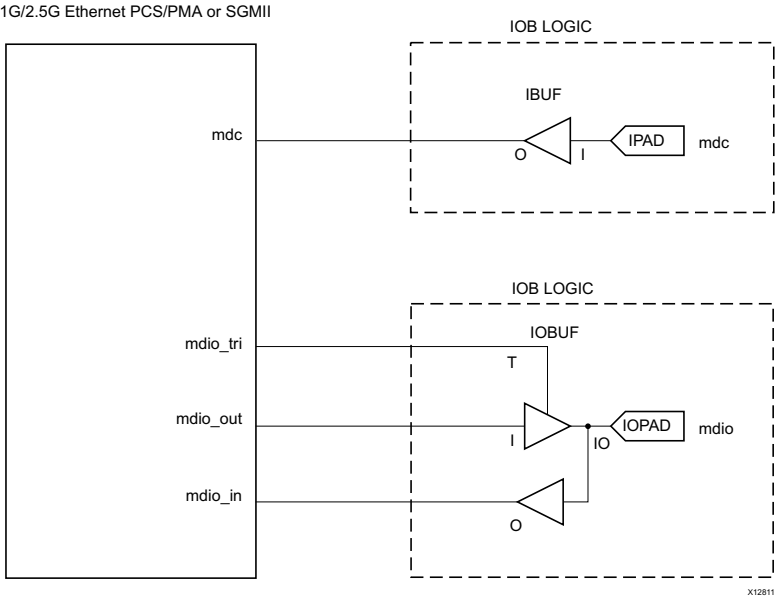
The MDIO ports of the core can be connected to the MDIO ports of an internally integrated Station Management (STA) entity, such as the MDIO port of the Tri-Mode Ethernet MAC core (see [Interfacing to Other Cores](#)).

### Connecting the MDIO to an External STA

The following figure shows the MDIO ports of the core connected to the MDIO of an external STA entity. In this situation, mdio\_i, mdio\_o, and mdio\_t must be connected to a 3-state buffer to create a bidirectional wire, mdio. This 3-state buffer can 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: 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 or 2500BASE-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 802.3-2008 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. For 2500BASE-X, the register definition is same as 1000BASE-X.

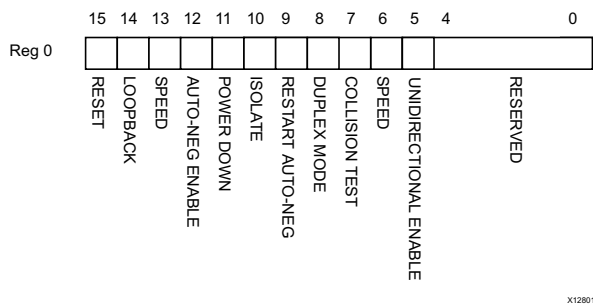
Table: MDIO Registers for 1000BASE-X or 2500BASE-X With Auto-Negotiation

Register Address	Register Name
Register 0: Control Register	
Register 1: Status Register	
Registers 2 and 3: PHY Identifiers	
Register 4: Auto-Negotiation Advertisement	
Register 5: Auto-Negotiation Link Partner 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: MDIO Register 0: Control Register



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

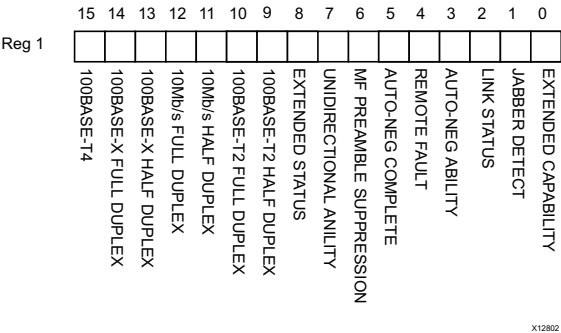
Table: Register 0 - Control Register

Name	Description	Default
Reset	Before Reset 0 = Normal Operation	R/ W Self clearing
Loopback	Enable Loopback Mode 0 = Disable Loopback Mode When used with a device-specific transceiver, the core is placed in internal loopback mode. <b>Note:</b> Loopback is not supported by the core when RxGmiiClkSrc=RXOUTCLK. In TBI mode, bit 1 is connected to the ewrap signal. When set to 1, indicates to the external PMA module to enter loopback mode. See <a href="#">Loopback</a> .	R/ W
Speed	BASE-X/2500BASE-X : Selection returns a 0 for this bit. Together with bit 0.6, a speed selection of 1000 Mbps is identified. In USB mode this bit along with bit 0.6, indicates a speed selection of 2500 Mbps. SGMII: 11 = Reserved 10 = 1 Gbps 01 = 100 Mbps 00 = 10 Mbps Zynq 7000, Zynq MPSoC and Zynq RFSoc PS Gigabit Ethernet Controller mode, identifies with bit 0.13 of Control register specified in IEEE 802.3-2008.	0000BA X/ 2500BA X : Returns 0 SGMII: R/ W in Zynq 7000, Zynq MPSoC and Zynq RFSoc PS Gigabit Ethernet Controller mode. Returns 0 in any other mode
Auto-Negotiation	optional auto-negotiation: Enable Auto-Negotiation process Disable Auto-Negotiation process Without optional auto-negotiation: Bit is reserved	R/ W
Power down	Power down Normal operation	R/ W

Bit	Description	Default
0.15	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. In TBI mode this register bit has no effect.	
0.14	<b>BASE-X/2500BASE-X :</b> 1 = Electrically isolate PHY from GMII <b>SGMII:</b> Electrically isolate SGMII logic from GMII 0 = Normal operation	R/ W
0.13	<b>Optional auto-negotiation:</b> Auto-Restart auto-negotiation process Negotiation operation <b>Without optional auto-negotiation:</b> Bit is reserved	R/ W Self clearing
0.12	Displays returns a 1 for this bit to signal Full-Duplex mode. Mode	Returns 1
0.11	Displays returns a 0 for this bit to disable COL test. Test	Returns 0
0.10	<b>BASE-X/2500BASE-X :</b> Selection returns a 1 for this bit. Together with bit 0.13, a speed selection of 1000 Mbps is identified. <b>(M25)G mode</b> this bit, along with bit 0.13, indicates a speed selection of 2500Mbps. <b>SGMII:</b> 11 = Reserved 10 = 1 Gbps 01 = 100 Mbps 00 = 10 Mbps Zynq 7000, Zynq MPSoC and Zynq RFSoc PS Gigabit Ethernet Controller mode, identifies with bit 0.6 of Control register specified in IEEE 802.3-2008.	1000BA X/ 2500BA X : Returns 1 <b>SGMII:</b> R/ W inZynq 7000 , Zynq MPSoC and Zynq RFSocP Gigabit Etherne Control mode. Returns 1 in any other mode
0.9	Indicates link direction 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
0.8	Always return 0s, writes ignored.	Reads 0s

Register 1: Status Register

Figure: MDIO Register 1: Status Register



**Table: Register 1 - Status Register**

Bits	Name	Description	Attributes	Default Value
1.15	100BASE-T4	Always returns a 0 because 100BASE-T4 is not supported.	Returns 0	0
1.14	100BASE-X Full Duplex	Always returns because 0 as 100BASE-X full duplex is not supported.	Returns 0	0
1.13	100BASE-X Half Duplex	Always returns a 0 because 100BASE-X half duplex is not supported.	Returns 0	0
1.12	10 Mbps Full Duplex	Always returns a 0 because 10 Mbps full duplex is not supported.	Returns 0	0
1.11	10 Mbps Half Duplex	Always returns a 0 because 10 Mbps half duplex is not supported	Returns 0	0
1.10	100BASE-T2 Full Duplex	Always returns a 0 because 100BASE-T2 full duplex is not supported.	Returns 0	0
1.9	100BASE-T2 Half Duplex	Always returns a 0 because 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	Using optional auto-negotiation: 1 = Auto-Negotiation process completed 0 = Auto-Negotiation process not completed	RO	0

Bits	Name	Description	Attributes	Default Value
		<i>Without optional auto-negotiation:</i> Ignore this bit.		
1.4	Remote Fault	<i>Using optional auto-negotiation:</i> 1 = Remote fault condition detected 0 = No remote fault condition detected <i>Without optional auto-negotiation:</i> Always returns a 0.	RO Self clearing on read	0
1.3	Auto- Negotiation Ability	<i>Using optional auto-negotiation:</i> Always returns a 1 for this bit to indicate that the PHY is capable of Auto-Negotiation. <i>Without optional auto-negotiation:</i> Ignore this bit.	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 <a href="#">Link Status</a> for further details.	RO	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: Registers 2 and 3: PHY Identifiers**

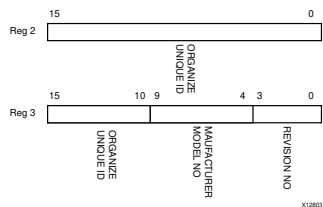


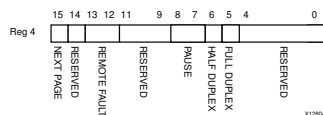
Table: Registers 2 and 3: PHY Identifier

Bits	Name	Description	Attributes	Default Value
2.15:0	Organizationally Unique Identifier (OUI) <sup>1</sup>	Returns OUI (0x005D03) (format as per IEEE specification).	RO	0000_0001_0111_0100
3.15:10			RO	000011
3.9:4	Manufacturer model number	Always return 0s	RO	000000
3.3:0	Revision Number	Always return 0s	RO	0000

1. The OUI is split across two registers.

Register 4: Auto-Negotiation Advertisement

Figure: MDIO Register 4: Auto-Negotiation Advertisement



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

Table: Register 4: Auto-Negotiation Advertisement Register

Bit	Name	Description	Default Value
15	Next Page	0 = does not support Next Page. It can be enabled, if requested. Writes ignored.	R/W
14	Reserved	Always returns 0, writes ignored	Returns 0
13	Link Error	0 = Link Error	R/W
12	Link Offline	0 = Link Offline	R/W
11	Link Failure	10 = Link Failure	Self clearing
10	Auto-Negotiation Error	11 = Auto-Negotiation Error	clearing to 00 after Auto-Negotiation
9	Reserved	Always return 0s, writes ignored	Returns 0
8	PAUSE	00 = No PAUSE	R/W
7	Symmetric PAUSE	01 = Symmetric PAUSE	
6	Asymmetric PAUSE towards link partner	10 = Asymmetric PAUSE towards link partner	
5	Both Symmetric PAUSE and Asymmetric PAUSE towards link partner	11 = Both Symmetric PAUSE and Asymmetric PAUSE towards link partner	
4	Half Duplex	Always returns a 0 for this bit because Half Duplex Mode is not supported	Returns 0
3	Full Duplex Mode is advertised	1 = Full Duplex Mode is advertised	R/W
2	Half Duplex Mode is not advertised	0 = Half Duplex Mode is not advertised <sup>1</sup>	R/W

Name	Description	Default
Ability to return 0s, writes ignored		00000000s
1. Even if the Full Duplex bit is set to 0, the core is assumed to be in Full Duplex, although it advertises the programmed ability.		

Register 5: Auto-Negotiation Link Partner Base

Figure: MDIO Register 5: Auto-Negotiation Link Partner Base

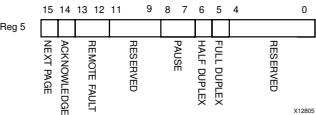


Table: Register 5: Auto-Negotiation Link Partner Ability Base Register

Name	Description	Default
Next Page functionality is supported	Next Page functionality is not supported	BO
Auto-Negotiation function to indicate reception of a link partner base or Next Page		BO
Error	Offline	BO
10 = Link Failure	11 = Auto-Negotiation Error	
Ability to return 0s		BO
No PAUSE	01 = Symmetric PAUSE	BO
10 = Asymmetric PAUSE towards link partner	11 = Both Symmetric PAUSE and Asymmetric PAUSE supported	
Half Duplex Mode is supported	Half Duplex Mode is not supported	BO
Full Duplex Mode is supported	Full Duplex Mode is not supported	BO
Ability to return 0s		BO

Register 6: Auto-Negotiation Expansion

Figure: MDIO Register 6: Auto-Negotiation Expansion

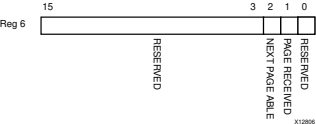


Table: Register 6: Auto-Negotiation Expansion Register

Name	Description	Default
Ability to return 0s		BO
Next Page Able	Next bit is ignored as the core does not support Next Page. This feature can be enabled on request.	Returns 1

Name	Description	Default
Bit 15 Page A new page has been received Bit 14 Received page has not been received		BO Self clearing on read
Bit 13 Acknowledge	Bit 12 Returns 0s	BO0000 0s

Register 7: Auto-Negotiation Next Page Transmit

Figure: MDIO Register 7: Next Page Transmit

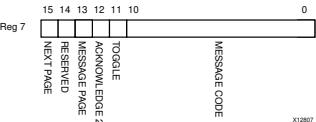


Table: Register 7: Auto-Negotiation Next Page Transmit

Name	Description	Default
Bit 15 Additional Next Page(s) will follow Bit 14 Last page		B/ W
Bit 13 Acknowledge	Bit 12 Returns 0	Returns 0
Bit 11 Message Page Bit 10 Unformatted Page		B/ W
Bit 9 Acknowledge with message Bit 8 Cannot comply with message		B/ W
Bit 7 Toggle	Bit 6 toggles between subsequent Next Pages	BO
Bit 5 Message/Code Field or Unformatted Page Encoding as dictated by 7.13 Unformatted Code Field		BO0000 Null Message Code)
1. This register returns zeros because the core does not support Next Page. This feature can be enabled on request.		

Register 8: Auto-Negotiation Next Page Receive

Figure: MDIO Register 8: Next Page Receive

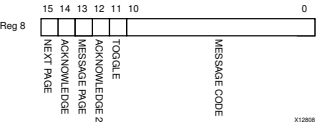


Table: Register 8: Auto-Negotiation Next Page Receive

Name	Description	Default
Bit 15 Additional Next Page(s) will follow Bit 14 Last page		BO
Bit 13 Acknowledge	Bit 12 Auto-Negotiation function to indicate reception of a link partner base or Next Page	BO
Bit 11 Message Page Bit 10 Unformatted Page		BO



Name	Description	Default
ACKnowledge with message 0 = Cannot comply with message		BO
Toggle	Toggles between subsequent Next Pages	BO
Message/Code Field or Unformatted Page Encoding as dictated by 8.13 Unformatted Code Field		BO0000

Register 15: Extended Status

Figure: MDIO Register 15: Extended Status Register

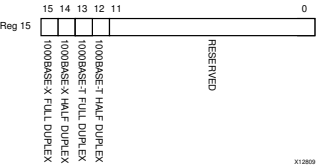


Table: Register 15 - Extended Status Register

Name	Description	Default
1000BASE-T Full Duplex is supported	Returns a 1 because 1000BASE-X Full Duplex is supported	Returns 1
1000BASE-T Half Duplex is not supported	Returns a 0 because 1000BASE-X Half Duplex is not supported	Returns 0
1000BASE-T Full Duplex is not supported	Returns a 0 because 1000BASE-T Full Duplex is not supported	Returns 0
1000BASE-T Half Duplex is not supported	Returns a 0 because 1000BASE-T Half Duplex is not supported	Returns 0
Reserved	Return 0s	BO0000 0s

Register 16: Vendor-Specific Auto-Negotiation Interrupt Control

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

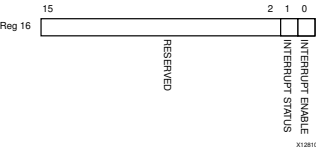



Table: Register 16 - Vendor Specific Register: Auto-Negotiation Interrupt Control Register

Name	Description	Default
Reserved	Return 0s	BO0000 0s
Interrupt is asserted		B/

Name	Description	Default
Status	<p>Interrupt is not asserted</p> <p>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.</p> <p>If the Interrupt is disabled, the bit is set to 0.</p> <hr/> <p> <b>Note:</b> The an_interrupt port of the core is wired to this bit.</p>	W
Enable	<p>Interrupt enabled</p> <p>Interrupt disabled</p>	R/ W

1000BASE-X or 2500BASE-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. For 2500BASE-X the register definition is same as 1000BASE-X.

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

Register Address	Register Name
0	<a href="#">Register 0: Control Register</a>
1	<a href="#">Register 1: Status Register</a>
2,3	<a href="#">Registers 2 and 3: PHY Identifiers</a>
15	<a href="#">Register 15: Extended Status</a>

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 [Using the SGMII MAC Mode to Interface to an External BASE-T PHY with SGMII Interface](#).

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. For 2.5G SGMII the register definition is similar to 1G SGMII. Speed selection bits in 2.5G mode are not relevant because the core supports only 2.5G.

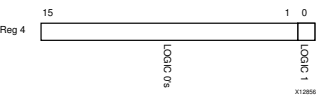
**Table: MDIO Registers for SGMII with Auto-Negotiation**

Register Address	Register Name
0	<a href="#">Register 0: Control Register</a>
1	<a href="#">Register 1: Status Register</a>
2,3	<a href="#">Registers 2 and 3: PHY Identifiers</a>
4	<a href="#">Register 4: Auto-Negotiation Advertisement</a>
5	<a href="#">Register 5: Auto-Negotiation Link Partner Base</a>
6	<a href="#">Register 6: Auto-Negotiation Expansion</a>
7	<a href="#">Register 7: Auto-Negotiation Next Page Transmit</a>
8	<a href="#">Register 8: Auto-Negotiation Next Page Receive</a>
15	<a href="#">Register 15: Extended Status</a>
16	<a href="#">Register 16: Vendor-Specific Auto-Negotiation Interrupt Control</a>

Register 4: SGMII Auto-Negotiation Advertisement

MAC Mode

Figure: MDIO Register 4: SGMII Auto-Negotiation Advertisement



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

Table: Register 4 - SGMII Auto-Negotiation Advertisement

Name	Description	Default
Advertisement	Defined value sent from the MAC to the PHY bits	000000

PHY Mode

Figure: MDIO Register 4: SGMII Auto-Negotiation Advertisement

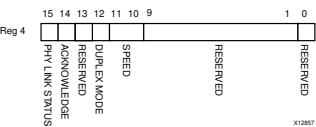
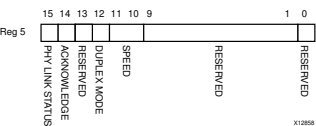


Table: Register 4: SGMII Auto-Negotiation Advertisement in PHY Mode

Name	Description	Default
Link Status	Refers to the link status of the PHY with its link partner across the Medium. Link Up Link Down	R/W
Acknowledge	Auto-Negotiation function to indicate reception of a link partner base or Next Page	R/W
Reserved	Reserved returns 0, writes ignored	Returns 0
Duplex Mode	Duplex Half Duplex	R/W
Speed	Reserved 10 = 1 Gbps 01 = 100 Mbps 00 = 10 Mbps	R/W
Reserved	Reserved return 0s	000000 0s
Reserved	Reserved returns 1	Returns 1

Register 5: SGMII Auto-Negotiation Link Partner Ability

Figure: 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: Register 5: SGMII Auto-Negotiation Link Partner Ability Base

Name	Description	Default
Link Status	Refers to the link status of the PHY with its link partner across the Medium.	RO

Name	Description	Default
Link Up		
Link Down		
Auto-Negotiation	Auto-Negotiation function to indicate reception of a link partner base or Next Page	BO
Reserved	Returns 0 writes ignored	Returns 0
Full Duplex		BO
Half Duplex		
Speed	Reserved 10 = 1 Gbps 01 = 100 Mbps 00 = 10 Mbps	BO
Reserved	Return 0s	00000000 0s
Reserved	Returns 1	Returns 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 [Using the SGMII MAC Mode to Interface to an External BASE-T PHY with SGMII Interface](#). Information about the state of the SGMII link is available in registers that follow. For 2.5G SGMII the register definition is similar to 1G SGMII. Speed selection bits in 2.5G mode are not relevant because the core supports only 2.5G.

**!! 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: MDIO Registers for SGMII with Auto-Negotiation

Register Address	Register Name
0	Register 0: Control Register
1	Register 1: Status Register
2,3	Registers 2 and 3: PHY Identifiers
15	Register 15: SGMII Extended Status

Register 15: SGMII Extended Status

Figure: MDIO Register 15: SGMII Extended Status

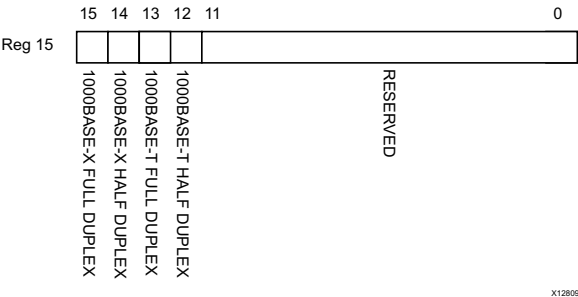


Table: Register 15 - SGMII Extended Status Register

Name	Description	Default
------	-------------	---------

Name	Description	Default
1000BASE-X Full Duplex	Returns a 1 for this bit because 1000BASE-X Full Duplex is supported	Returns 1
1000BASE-X Half Duplex	Returns a 0 for this bit because 1000BASE-X Half Duplex is not supported	Returns 0
1000BASE-T Full Duplex	Returns a 0 for this bit because 1000BASE-T Full Duplex is not supported	Returns 0
1000BASE-T Half Duplex	Returns a 0 for this bit because 1000BASE-T Half Duplex is not supported	Returns 0
Reserved	Return 0s	Return 0s

Both 1000BASE-X and SGMII Standards

The following table 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](#). Dynamic Switching between 2500BASE-X and 2.5G SGMII is not supported by the core.

When this register is configured to perform the 1000BASE-X standard, registers 0 to 16 should be interpreted as per [1000BASE-X or 2500BASE-X Standard Using Optional Auto-Negotiation](#) or [1000BASE-X or 2500BASE-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 or 2500BASE-X Standard without Optional Auto-Negotiation](#). This register can be written to at any time. See [Dynamic Switching of 1000BASE-X and SGMII](#) for more information.

Figure: Dynamic Switching (Register 17)

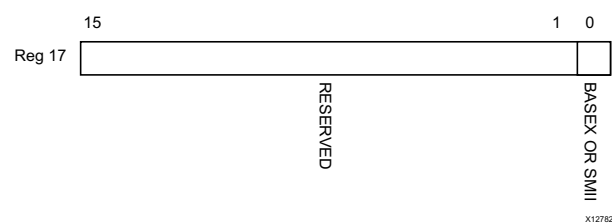


Table: Register 17 - Vendor-specific Register: Standard Selection Register

Name	Description	Default
Reserved	Return 0s	Return 0s
Standard	Performs to the 1000BASE-X standard. Registers 0 to 16 behave as per <a href="#">1000BASE-X or 2500BASE-X Standard Using Optional Auto-Negotiation</a> 1= Core performs to the SGMII standard. Registers 0 to 16 behave as per <a href="#">SGMII Standard Using Optional Auto-Negotiation</a> .	Determined by the basex_c 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 the following table.

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](#). Status signals are also brought out of the core to status\_vector as defined in [Table 3](#).

**Table: Configuration Vector**

Bits	Description
0	<i>Unidirectional Enable</i> . When set to 1, Enable Transmit irrespective of state of RX (802.3ah). When set to 0, Normal operation
1	<i>Loopback Control</i> . When the core with a device-specific transceiver is used, this places the core into internal loopback mode. In TBI mode bit 1 is connected to ewrap. When set to 1, this signal indicates to the external PMA module to enter loopback mode.
2	<i>Power Down</i> , When the Zynq 7000, Virtex 7, Kintex 7, and Artix 7 device 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. In TBI mode this bit is unused.
3	<i>Isolate</i> . When set to 1, the GMII should be electrically isolated. When set to 0, normal operation is enabled.
4	<i>Auto-Negotiation Enable</i> . 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: Auto-Negotiation Vector**

Bits	Description <sup>1</sup>
0	For 1000BASE-X or 2500BASE-X-Reserved. For SGMII- Always 1
4:1	Reserved
5	For 1000BASE-X or 2500BASE-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 or 2500BASE-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 or 2500BASE-X- Reserved For SGMII- Speed 1 1 = Reserved 1 0 = 1000 Mbps 0 1 = 100 Mbps 0 0 = 10 Mbps
13:12	For 1000BASE-X or 2500BASE-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 or 2500BASE-X- Reserved For SGMII- Acknowledge
15	For 1000BASE-X or 2500BASE-X- Reserved. Should be tied to 0 if the next page is disabled or not used.

Bits	Description <sup>1</sup>
	For SGMII- PHY Link Status 1 = Link Up 0 = Link Down
<p>1. In SGMII operating in MAC Mode, the AN_ADV register is hard wired internally to “0x01” and this bus has no effect. For 1000BASE-X or 2500BASE-X and SGMII operating in PHY mode, the AN_ADV register is programmed by this bus as specified for the following bits.</p>	

**Table: Status Vector**

Bits	Description
0	<i>Link Status</i> . 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	<i>Link Synchronization</i> . 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	<i>RUDI(/C/)</i> . The core is receiving /C/ ordered sets (Auto-Negotiation Configuration sequences) as defined in IEEE 802.3-2008 clause 36.2.4.10.
3	<i>RUDI(/I/)</i> . The core is receiving /I/ ordered sets (Idles) as defined in IEEE 802.3-2008 clause 36.2.4.12.
4	<i>RUDI(INVALID)</i> . The core has received invalid data while receiving/C/ or /I/ ordered 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	<i>RXDISPERR</i> . The core has received a running disparity error during the 8B/10B decoding function.
6	<i>RXNOTINTABLE</i> . The core has received a code group which is not recognized from the 8B/10B coding tables.
7	<i>PHY Link Status (SGMII mode only)</i> . 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 it 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.
9:8	<i>Remote Fault Encoding</i> . 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	<i>SPEED</i> . This signal indicates the speed negotiated and is only valid when Auto-Negotiation is enabled. In 1000BASE-X or 2500BASE-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.

Bits	Description
	1 1 = Reserved 1 0 = 1000 Mbps; 2500 Mbps in 2.5G mode 0 1 = 100 Mbps; reserved in 2.5G mode 0 0 = 10 Mbps; reserved in 2.5G mode
12	<i>Duplex Mode</i> . This bit indicates the Duplex mode negotiated with the link partner. Indicates bit 5 of Link Partner Base AN register 5 in 1000BASE-X or 2500BASE-X mode; otherwise bit 12 in SGMII mode. (In SGMII MAC and PHY mode it is register bit 5.12.) 1 = Full Duplex 0 = Half Duplex
13	<i>Remote Fault</i> . When this bit is logic one, it indicates that a remote fault is detected and the type of remote fault is indicated by <i>status_vector bits[9:8]</i> . This bit reflects MDIO register bit 1.4. <hr/> <b>Note:</b> This bit is only deasserted when a MDIO read is made to status register (register1). This signal has no significance in SGMII PHY mode or when MDIO is disabled.
15:14	<i>Pause</i> . These bits reflect the bits [8:7] of Register 5 (Link Partner Base AN register). These bits are valid only in 1000BASE-X or 2500BASE-X mode and have no significance in SGMII mode. 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 section includes guidelines and additional information to facilitate designing with the core.

### General Design Guidelines

The following sections provide some design guidelines.

#### Understand the Core Features and Interfaces

[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 AMD 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 or 2500BASE-X with Transceiver Example Design](#)
- [SGMII/Dynamic Switching Using a Transceiver Example Design](#)
- [Synchronous SGMII over LVDS Example Design \(Applicable for Non-Versal Devices\)](#)
- [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 an AMD 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 [Constraining the Core](#).

---

#### Simulate and Download Your Design

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

#### Know the Degree of Difficulty

A 1G/2.5G Ethernet 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 AMD 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](#) for more information.

#### Make Only Allowed Modifications

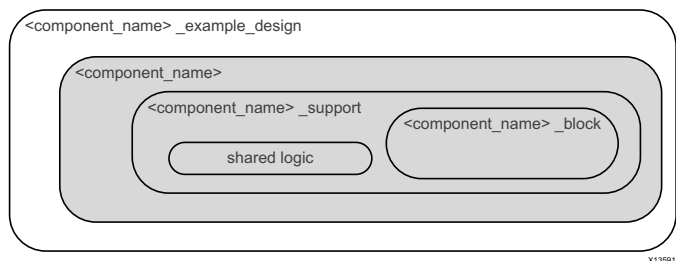
The core should not be modified. Modifications can have adverse effects on system timing and protocol compliance. Supported user configurations of the core can only be made by selecting the options from within the IP catalog when the core is generated. See [Customizing and Generating the Core](#).

## Shared Logic

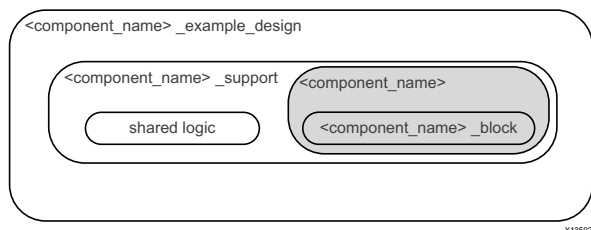
Up to version 13.0 of the core, the RTL hierarchy for the core was fixed. This resulted in some difficulty because shareable clocking and reset logic needed to be extracted from the core example design for use with a single instance, or multiple instances of the core. Shared logic is a feature that provides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

#### Figure: Shared Logic Included in Core

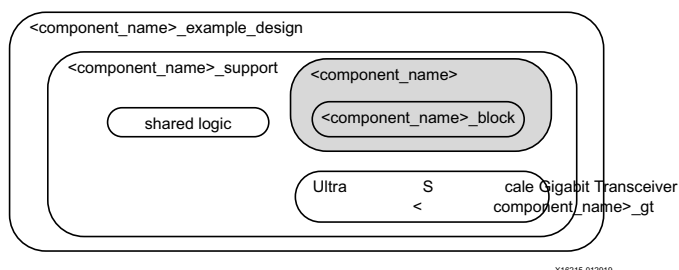


**Figure: Shared Logic Included in Example Design**



In version 15.2 of the core, an option was introduced under *Shared logic Included in Example Design* . The option *GT in Example design*, pulls out the Gigabit transceiver from the core and moves it to the `<component_name>_support` hierarchy. The Gigabit transceiver can be edited. The core provides a interface that can be connected one to one with the transceiver. The following figure shows this hierarchy.

**Figure: Shared Logic and Transceiver Included in Example Design**



## Clocking

For clocking frequencies for the Vivado Design Suite, see [Constraining the Core](#).

For clocking information on the client interface in SGMII mode, see [Constraining the Core](#).

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

- For Asynchronous LVDS Clocking, see [Asynchronous LVDS Transceiver for Versal devices](#).
- For 1000BASE-X or 2500BASE-X, see [1000BASE-X or 2500BASE-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.

The reset from the example design (which is the system reset from the board or any external source) is synchronized with the respective clock domain and then provided to the core, hence it is synchronous reset to the core.

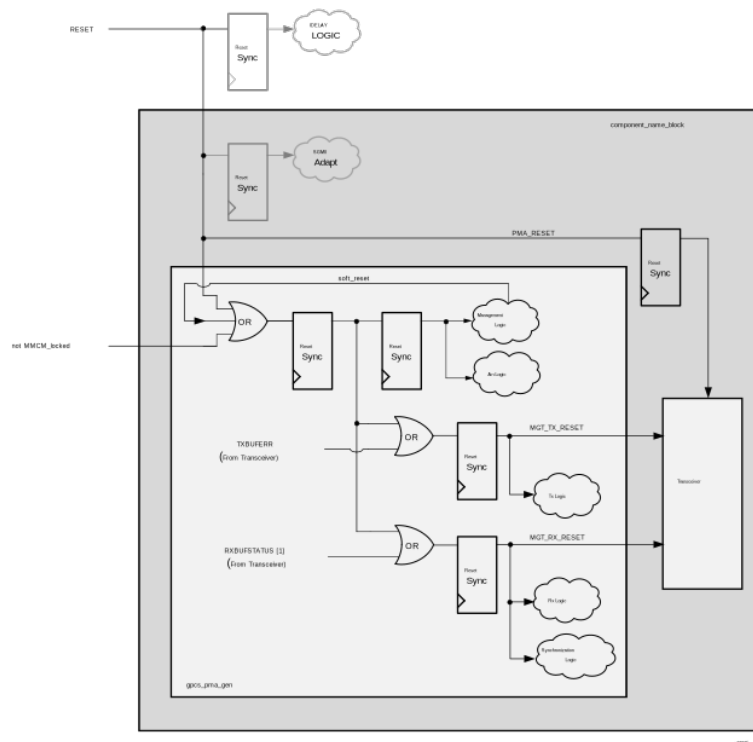
For LVDS Transceiver configuration, the `tx_logic_reset` and `rx_logic_reset` are generated using 156.25MHz clock by a control state machine. The above two resets are ORed and then synchronized with the core clock to 125 MHz and provided to the core as reset.

For GT Transceiver configuration, the reset is synchronized with `independent_clock_bufg` and provided to the core as reset.

### Reset Structure with Transceiver

The following figure 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.

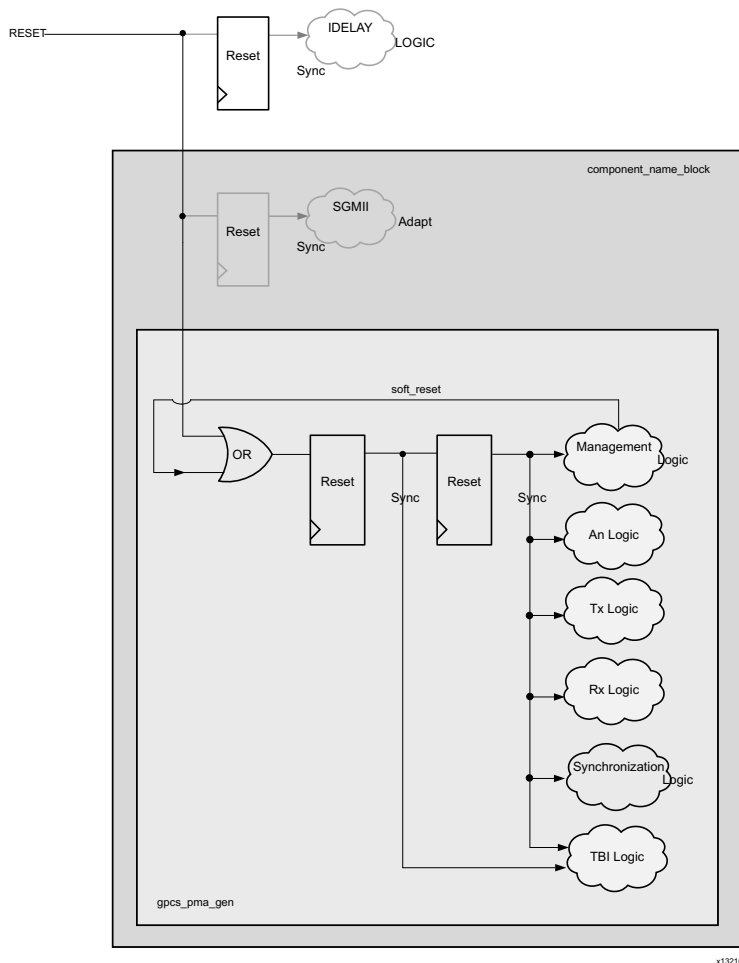
**Figure: Reset Structure for Core with Transceiver**



Reset Structure with TBI

The following figure 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: Reset Structure for Core with TBI



## 1000BASE-X or 2500BASE-X with Transceivers

This section provides general guidelines for creating 1000BASE-X designs for device-specific transceivers. 2500BASE-X follows the same structure and guidelines as for 1000BASE-X designs. 2500BASE-X is not supported for devices that have GTP transceivers. For information on the example design see [1000BASE-X or 2500BASE-X with Transceiver Example Design](#).

### Transceiver Logic for 7 Series and Zynq 7000 Devices

The core is designed to integrate with 7 series and Zynq 7000 FPGA transceivers. The following figure shows the transceiver connections for AMD Virtex™ 7, AMD Kintex™ 7 and Zynq 7000 devices. AMD Virtex™ 7 devices support both the GTX and GTH transceivers; AMD Kintex™ 7 and Zynq 7000 devices support the GTX transceiver. For AMD Artix™ 7 devices, the transceiver is connected as shown in the following figure; the supported transceiver is a GTP transceiver.

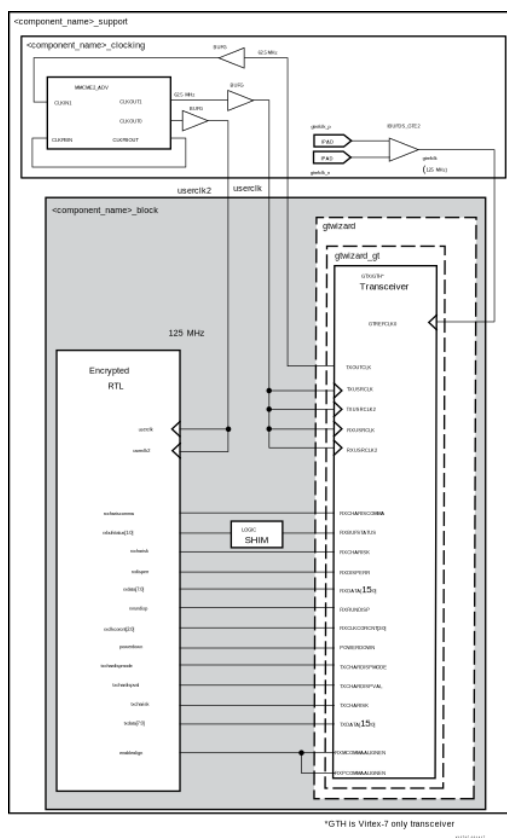
Both the following figures show the connectivity of the clocking logic with the encrypted core and transceiver channel. The internal signal name of GT\_CHANNEL or Encrypted RTL might not exactly correspond to the block level port names. For connectivity at the block level see [Port Descriptions](#). For shared logic connectivity guidelines see [Clock Sharing Across Multiple Cores with Transceivers](#). Note that in this case rxuserclk / rxuserclk2 is unused in the core.

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, 125 MHz for 2.5 Gbps) on the txoutclk port; this is then routed to a MMCM. From the MMCM, generated clocks (62.5 MHz, 125 MHz for 1 Gbps; 156.25 MHz, 312.5 MHz for 2.5 Gbps) are placed onto global clock routing and are input back into the transceiver on the user interface clock ports txusrc1k , and txusrc1k2. 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 as shown in the following figure, 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.

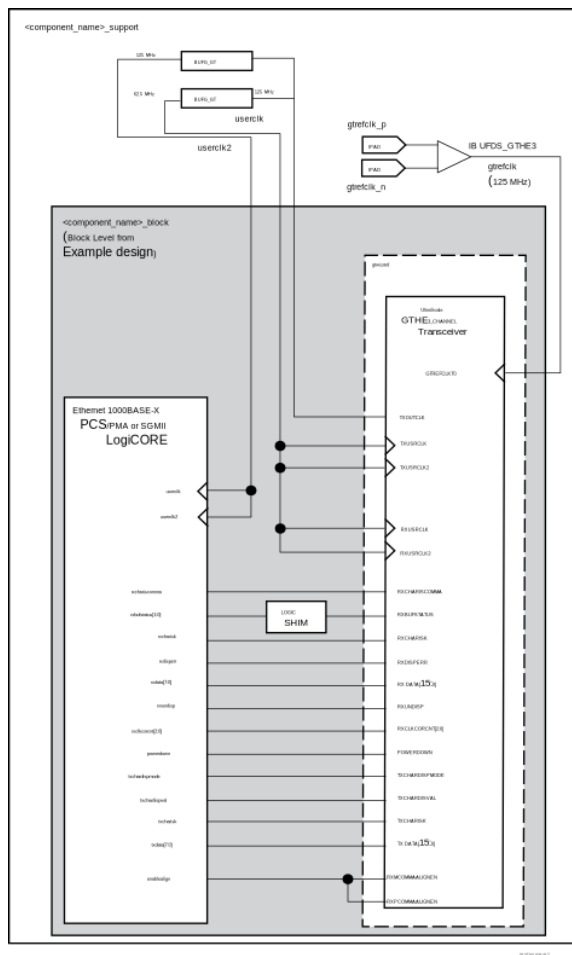
A 500 ns wait time for reset is generated with respect to the system clock input in the <component\_name>\_gtwizard\_init.v[hd] module. The STABLE\_CLOCK\_PERIOD attribute in this file has to be set to the period of the system clock.

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



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





## Transceiver Logic for Versal Devices

The Transceiver logic for Versal devices is similar to that of UltraScale and UltraScale+ with the GT IP always present in the example design.

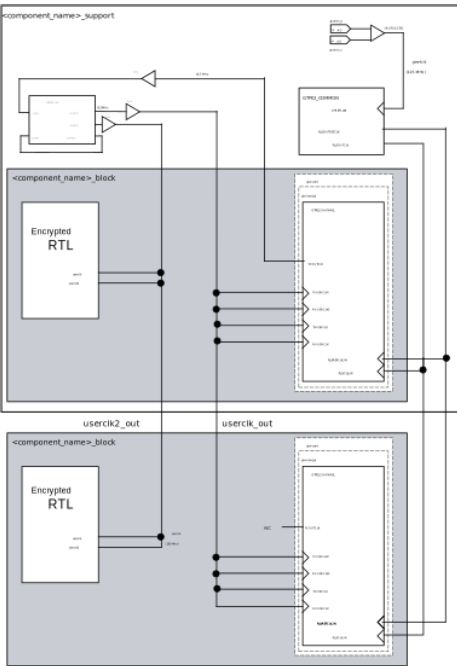
## Clock Sharing Across Multiple Cores with Transceivers

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.

The following figure shows sharing clock resources across two instantiations of the core for AMD Virtex™ 7, AMD Kintex™ 7 and Zynq 7000 device transceivers; [Figure 2](#) shows the connections for AMD Artix™ 7 devices. The following table 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*. The clock frequencies specified in the diagrams are for 1G mode.

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 for 1 Gbps, 312.5 MHz for 2.5 Gbps) and clkout1 (62.5 MHz for 1 Gbps, 156.25 MHz for 2.5 Gbps) outputs from this MMCM, placed onto global clock routing using BUFs, can be shared across all core instances and transceivers as shown.

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



For UltraScale+/UltraScale devices the MMCM is not required because the txoutclk generated is 125 MHz for 1 Gbps, 312.5 MHz for 2.5 Gbps and can be used to generate the 62.5 MHz for 1 Gbps, 156.25 MHz for 2.5 Gbps (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](#).

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

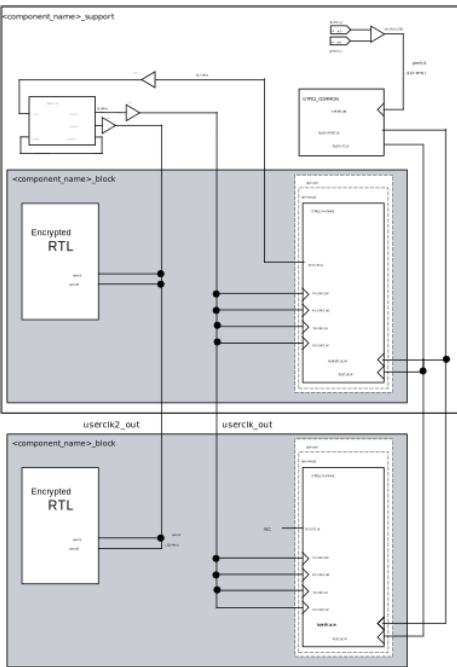


Table: Shared Signals Connectivity

Tools	Instance	Using Shared Logic	Example Design	Design Considerations
gtrefclk	can be shared	up one quad and down one quad		
gtrefclk	not configurable	only for 7 series and Zynq devices.		
txoutclk	not 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 (For 1G mode). This clock frequency is 62.5 MHz in 1 Gbps mode, 156.25 MHz in 2.5 Gbps mode. This clock can be used across the whole device and is used for any GT that has the same gtrefclk		
txuserclk	not shared	only when TXOUTCLK (and hence gtrefclk) of both the instances are synchronous; otherwise this should be connected to TXOUTCLK multiplied by two (for 1G mode) of the same instance. This clock frequency is 125 MHz in 1 Gbps mode, 312.5 MHz in 2.5 Gbps mode. This clock can be used across the whole device and used for any GT that has the same gtrefclk.		

Tools Instantiation Using Shared Logic Example Design Considerations
<b>rxoutclk_out</b> rxoutclk_out is 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 in 1 Gbps mode, 156.25 MHz in 2.5 Gbps mode.
<b>rxoutclk_refclk_out</b> rxoutclk_refclk_out is shared only when the RXOUTCLK signal (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 in 1 Gbps mode and 156.25 MHz in 2.5 Gbps mode. The frequency of this clock when RxGmiiClkSrc=RXOUTCLK is 125 MHz and 312.5 MHz respectively.
pma_reset_out
Provide clock to all instances use the same MMCM.
GTH/GTX Transceiver Specific Signals
gt0_pll0outclk_out gt0_pll0outrefclk_out gt0_pll0outrefclklost_out
GTP Transceiver Specific Signals
gt0_pll0outclk_out gt0_pll0outrefclk_out gt0_pll1outclk_out gt0_pll1outrefclk_out gt0_pll0lock_out gt0_pll0refclklost_out
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 or 2500BASE-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 or 2500BASE-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. Dynamic Switching between 2500BASE-X and 2.5 G SGMII is not supported by the core.

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 in UltraScale and UltraScale+ families. For 7 series devices the buffer in the device-specific transceiver is not bypassed; however it is read using the recovered clock.

The receive elastic buffer if present, 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 for 1G SGMII:

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

For 2.5G SGMII multiple speeds are not applicable.

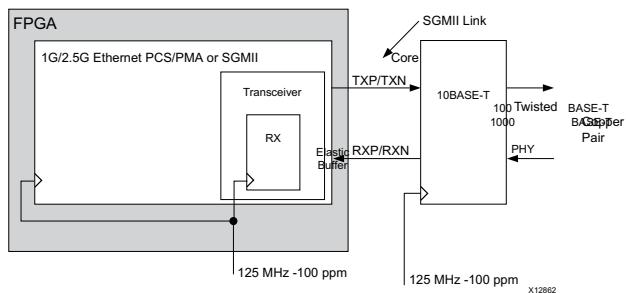
The first option, 10/100/1000 Mbps (clock tolerance compliant with Ethernet specification) is the default and provides the implementation using the receive elastic buffer in FPGA logic. This alternative receive elastic buffer uses a single block RAM to create a buffer twice as large as the one present 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 Mbps (restricted tolerance for clocks) or 100/1000 Mbps, 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 Mbps 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

The following figure 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 the following figure, 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). The receive elastic buffer also performs clock correction on C1,C2 configuration code words received during auto-negotiation. This is similar to the way that clock correction is done on C1 in the transceiver elastic buffer.

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

Considering the 10 Mbps case, 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 [Receive Elastic Buffer Specifications](#) for more information).

To guarantee reliable SGMII operation at 10 Mbps (non-jumbo frames), the device-specific transceiver elastic buffer must be bypassed and a larger buffer implemented in the FPGA 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.

[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 Mbps operation is not required. Both 1 Gbps and 100 Mbps operation can be guaranteed.
- When the clocks are closely related (see [Closely Related Clock Sources](#)).

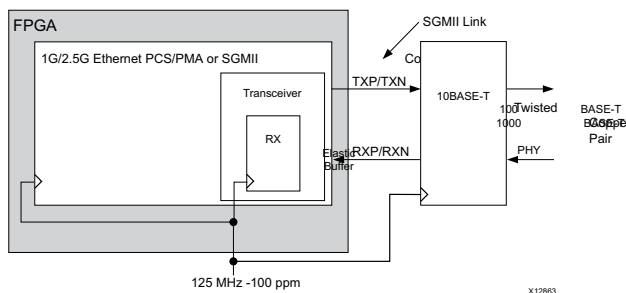
If there is any doubt, select the FPGA logic receive elastic buffer implementation.

#### Closely Related Clock Sources

##### Case 1

The following figure 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: 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 [Requirement for the Receive Elastic Buffer](#) 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 or 2500BASE-X with Transceivers](#).

## Transceiver Logic with the FPGA Logic Receive Elastic Buffer

The example design delivered with the core is shown in [SGMII/Dynamic Switching Using a Transceiver Example Design](#). The block level is designed so to be instantiated directly into customer designs and connects the physical-side interface of the core to an AMD UltraScale+™ /AMD UltraScale™, Virtex 7, Kintex 7 or 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 for 7 Series and Zynq 7000 Devices

The core is designed to integrate with 7 series and Zynq 7000 FPGA transceivers. The connections and logic required between the core and transceiver are shown in the following figure for Virtex 7. Kintex 7 and Zynq 7000 devices; the connections for Artix 7 devices are shown in [uon1664789800031.html#uon1664789800031\\_image\\_t5b\\_knl\\_hvb](#).

The following figures show the connectivity of clocking logic with an encrypted core and transceiver channel. The internal signal names of the GT\_CHANNEL or the encrypted RTL might not exactly correspond to the block level port names. For connectivity purposes at the block level see [Port Descriptions](#). For shared logic connectivity guidelines see [Clock Sharing Across Multiple Cores with Transceivers](#). The clock buffer on RXOUTCLK in the diagrams is a part of the clocking logic shown below for simplification.

The 125 MHz differential reference clock is routed directly to the transceiver. The transceiver is configured to output 62.5 MHz (125 MHz for 2.5G) 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 for 1G and 312.5 MHz for 2.5G) is placed onto global clock routing and can be used as the 125/312.5 MHz clock source for all core logic.

From the MMCM, the clkout1 port (62.5 MHz for 1G and 156.25 MHz for 2.5G) 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.

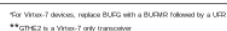
It can be seen from the following figures that the receive elastic buffer is implemented in the FPGA logic between the transceiver and the core; this replaces the receive elastic buffer in the transceiver.

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 Mbps where each frame size is effectively 100 times larger than the same frame would be at 1 Gbps because each byte is repeated 100 times (see [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 7 and Zynq 7000) and is used to synchronize the transfer of data between the transceiver and the elastic buffer, as shown in the following figures.

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: 1G SGMII Transceiver Connections (Virtex 7, Kintex 7, Zynq 7000)**



## Transceiver Logic For UltraScale+/UltraScale Devices

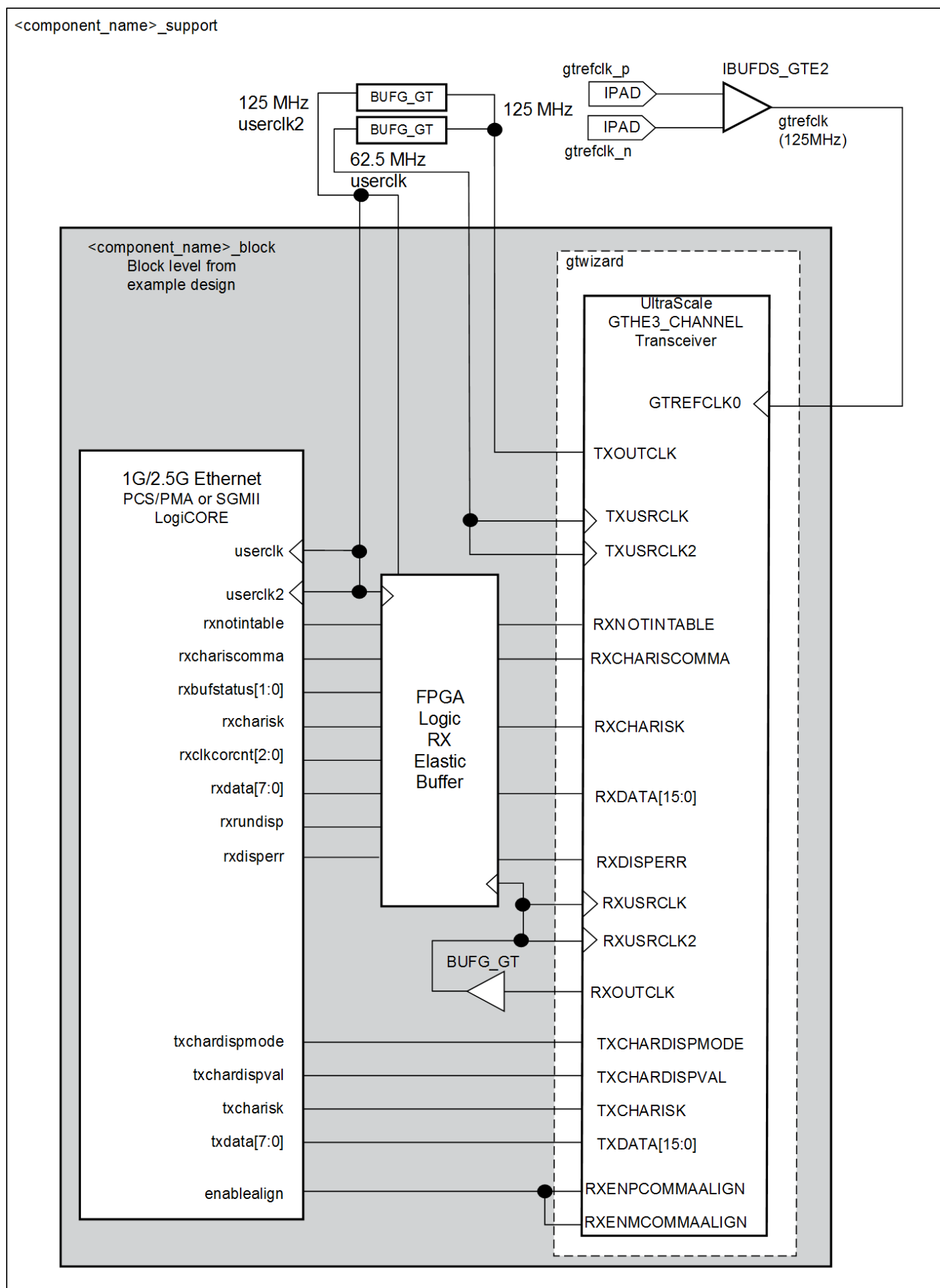
The following figure shows the connectivity of clocking logic with encrypted core and transceiver channel. The internal signal names of the GT\_CHANNEL or the encrypted RTL might not exactly correspond to the block level port names. For connectivity purposes at the block level see [Port Descriptions](#). For shared logic connectivity guidelines see [Clock Sharing Across Multiple Cores with Transceivers](#).

The clock buffer on RXOUTCLK in the diagrams is a part of the clocking logic shown below for simplification.

The differential reference clock selected through Vivado IDE is routed directly to the UltraScale+/UltraScale FPGA transceiver. The transceiver is configured to output a version of this clock (125 MHz for 1 G, 312.5 MHz for 2.5G) on the txoutclk port; txoutclk is then routed to a BUFG\_GT to generate *userclk* (62.5, 125MHz for 1G and 312.5, 156.25 MHz for 2.5G), *userclk2* (125 MHz for 1G, 312.5 MHz for 2.5G) and placed onto global clock routing. These clocks are input back into the transceiver on the user interface clock ports *usrclk* and *usrclk2*. The clocking logic is included in a separate module `<component_name>_clocking` which is instantiated in the `<component_name>_support` module.

Transceiver files are generated on runtime through UltraScale+/UltraScale gtwizard. See the *UltraScale Architecture GTH Transceivers User Guide* ([UG576](#)), the *UltraScale Architecture GTY Transceivers User Guide* ([UG578](#)), and the *FPGAs Transceivers Wizard LogiCORE IP Product Guide* ([PG182](#)) for details on the supporting logic and files generated.

### Figure: 1G SGMII Transceiver Connections (UltraScale Architecture)



X18829-031417

The Transceiver logic for Versal devices is similar to that of UltraScale and UltraScale+ with the GT IP always present in the example design.

## 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. Clock frequencies specified in the diagrams are for 1G mode.

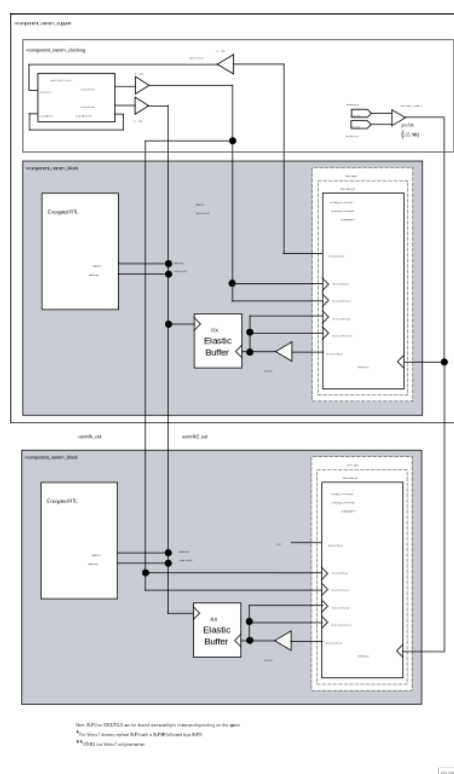
The clocking logic for rxoutclk can only be shared if it is known that the transceiver and core pairs across pcs-pma instances are synchronous. In this case the receive clock outputs of clocking module can be used.

The following figure shows sharing clock resources across two instantiations of the core for Virtex 7, Kintex 7 and Zynq 7000 device transceivers; [Figure 2](#) shows the connections for Artix 7 devices. [Clock Sharing Across Multiple Cores with Transceivers](#) 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 gtrfclk\_p and gtrfclk\_n differential clock pairs. See the *7 Series FPGAs GTX/GTH Transceivers User Guide (UG476)* or the *7 Series FPGAs GTP Transceivers User Guide (UG482)* for more information.

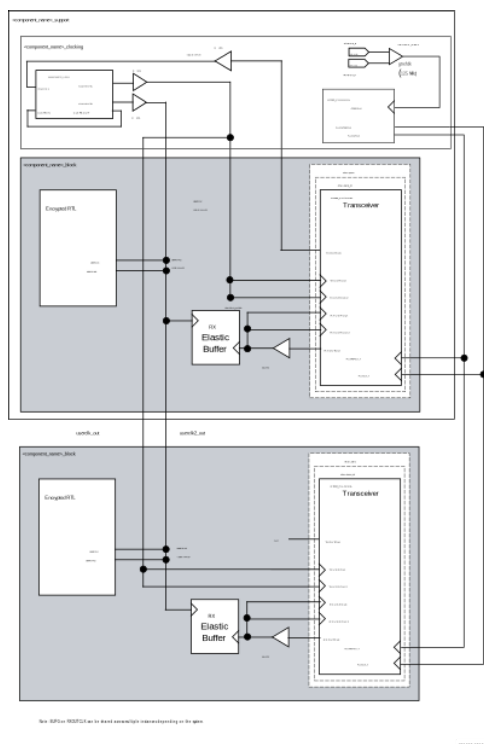
To provide the FPGA logic clocks for all core instances, select a txoutclk port from any transceiver and route this to a single MMCM. The clkout0 (125 MHz) and clkout1 (62.5 MHz) outputs from this MMCM, placed onto global clock routing using BUFs, can be shared across all core instances and transceivers 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: Clock Management with Multiple Core Instances for SGMII (Virtex 7, Kintex 7, Zynq 7000)**



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





## 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 Mbps, 100 Mbps, and 1 Gbps.
  - GMII logic for both transmitter and receiver paths; the GMII style 8-bit interface is run at 125 MHz for 1 Gbps operation; 12.5 MHz for 100 Mbps operation; 1.25 MHz for 10 Mbps 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 Gbps. However, at 100 Mbps, every data byte run through the core should be repeated 10 times to achieve the required bit rate; at 10 Mbps, 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](#).

## Synchronous SGMII over LVDS

This section provides the guidelines for creating SGMII interfaces using AMD Zynq™ 7000, 7 series and UltraScale devices. 2.5G SGMII is not supported for the LVDS physical interface.

### 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 the following table. This mode enables direct connection to external PHY devices without the use of an FPGA transceiver. An example implementation is shown in the following figure.

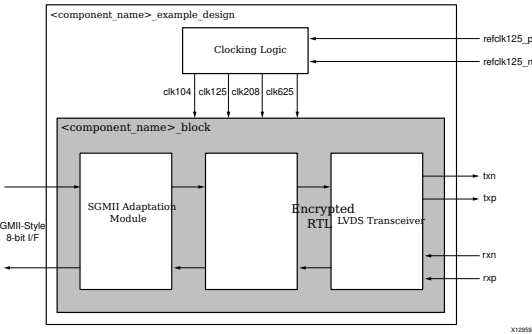
**Table: Devices Supporting SGMII over LVDS**

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

Family	Supported 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, Spartan 7	-2 speed grade or faster

For information about the SGMII over LVDS example design see [Synchronous SGMII over LVDS Example Design \(Applicable for Non-Versal Devices\)](#).  
 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)* and *7 Series FPGAs Clocking Resources User Guide (UG472)*.

**Figure: 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](#).

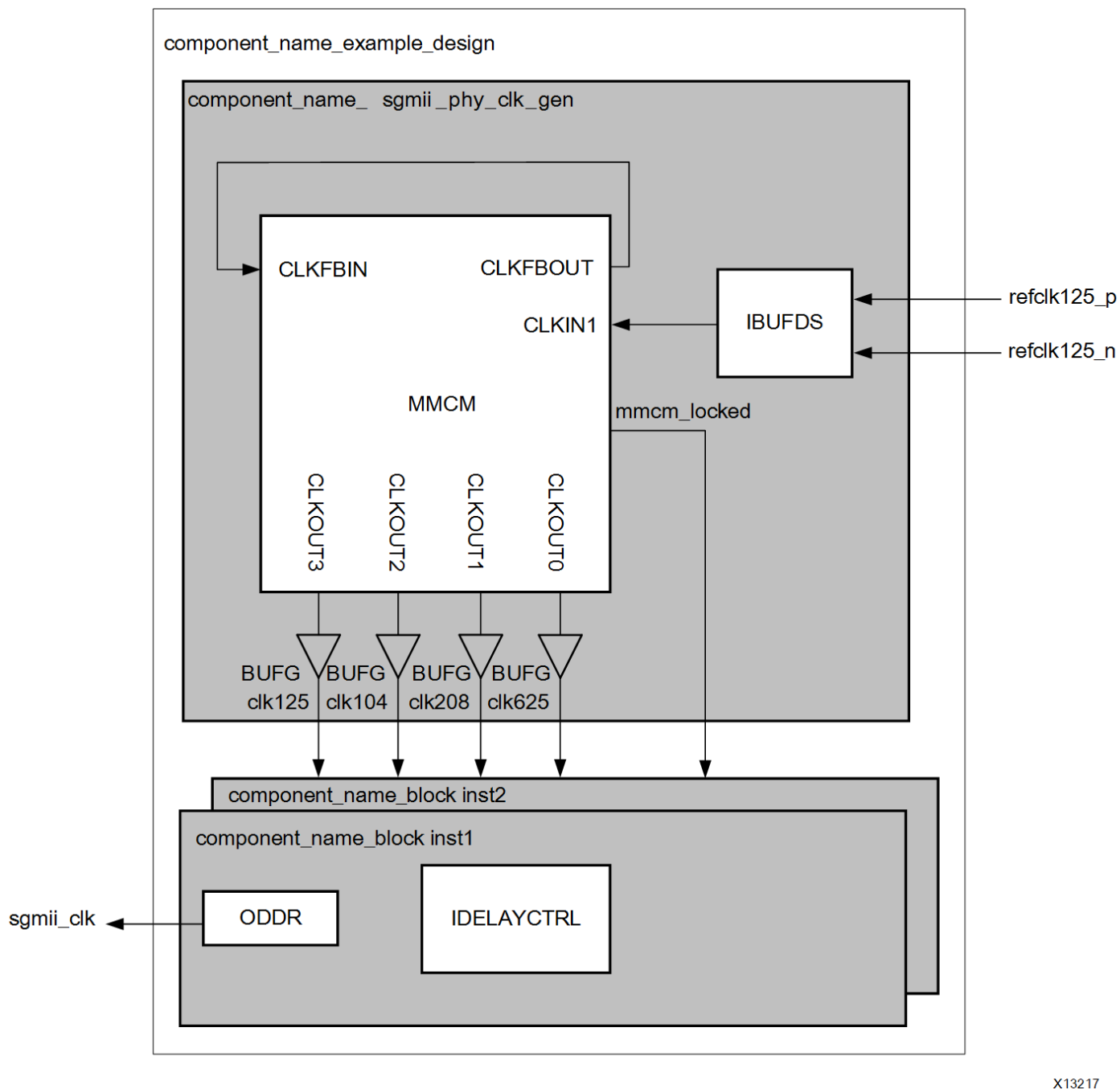
#### SGMII Only

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

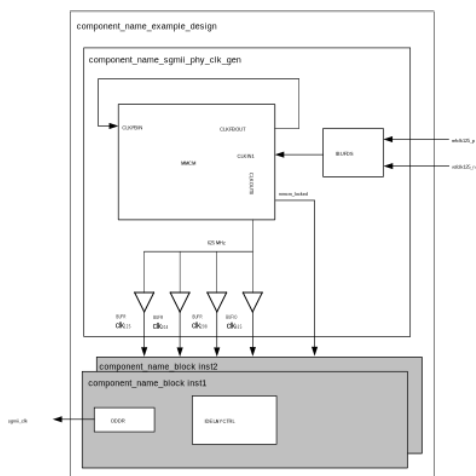
The SGMII LVDS solution is a synchronous implementation where an external clock is provided to the design. In the example design this clock is assumed to be a 125 MHz differential clock.  
 This 125 MHz differential clock is fed to IBUFDS and the output drives the input of MMCM. The MMCM is used to generate 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)*.  
 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. The following figure shows the clocking logic. The following table lists the clocks in the design and their description.

**Figure: Synchronous LVDS Implementation Clocking Logic Using BUFs**



X13217

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



Important notes relating to the previous two figures :

- The 125 MHz clock output from IBUFDS that is routed to the clk<sub>in1</sub> 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).
- The previous figure shows that a BUFIO can be used for the 625 MHz clock and a BUFR for the other three MMCM clock outputs. BUF<sub>IO</sub> DIVIDE should be used to divide the MMCM output clock down from 625 MHz for clk<sub>125</sub>, clk<sub>104</sub>, and clk<sub>208</sub> to keep clock skew low.
- A BU<sub>FH</sub> can also be used on all four MMCM clock outputs in [jge1665047219063.html#jge1665047219063\\_\\_image\\_lxg\\_bpl\\_hvb](#). IDELAYCTRL reference clock cannot be fed from the MMCM output with the BU<sub>FH</sub> 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 ( clk<sub>div</sub>) 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)* and *7 Series FPGAs Clocking Resources User Guide (UG472)*.
- 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.

The following table provides a description of the core clocks.

**Table: Design Clocks**

Input/Generated/Output	Description
Differential clock input to FPGA, synchronous to the incoming serial data.	
Differential clock input to FPGA, synchronous to the incoming serial data.	
125 MHz input clock	Derived from incoming differential clock by IBUFGDS.This is the input clock for MMCM.
Output Clock to MAC	Output clock client MAC. This clock is derived from sg <sub>mii</sub> _clk_r and sg <sub>mii</sub> _clk_f using ODDR primitive.
104 MHz	This clock is used in eye monitor and phy calibration modules to process 12-bit wide data. Generated by MMCM/BUFR
208 MHz	On transmitter path OSERDES takes 6-bit parallel data at this frequency and converts it to serial data.Similarly on receiver path BUFR converts serial data into 6 bit parallel data at 208 MHz.Later 6 bit data is converted into 10-bit data through gearbox. Clock generated by the IDELAYCTRL primitive. Generated by MMCM/BUFR
625 MHz	Generated by OSERDES and OSERDES modules for input data sampling and parallel to serial conversion respectively. Generated by MMCM
625 MHz	Used inside the design as main clock. The PCS/PMA core and SGMII adaptation modules work at this clock. Generated by MMCM/BUFR
SGMII adapter	Generated at 12.5 MHz or 1.25 MHz depending on data rate. Generated in SGMII adapter

Input/Generated/Output	Description
Generated	Generated for 12.5 MHz or 1.25 MHz depending on data rate.
Input	in
SGMII	SGMII
adapter	adapter

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)* and *7 Series FPGAs Clocking Resources User Guide (UG472)*. A working knowledge of the Vivado Design Suite is 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

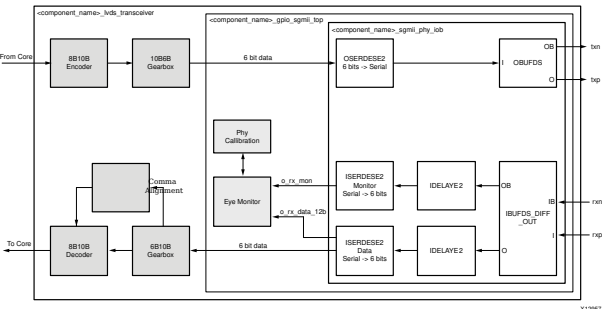
The LVDS transceiver block fully replaces the functionality otherwise provided by a 7 series device transceiver. This is only possible at a serial line rate of 1.25 Gbps. The following figure shows a block diagram of the LVDS transceiver for Zynq 7000 and 7 series devices. This is split up into several 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 Gbps.

The receiver path has additional complexity. Serial data received at 1.25 Gbps 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: 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 AMD Application Note, *Parameterizable 8B/10B Encoder* (XAPP1122) 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 Gbps. 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 RDES in the *7 series FPGAs SelectIOResources User Guide (UG471)* 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 bitflip 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 bitflip 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 AMD Application Note, *Parameterizable 8B/10B Encoder* (XAPP1122) 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 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 [LVDS Transceiver for 7 Series and Zynq 7000 Devices](#) 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 [LVDS Transceiver for 7 Series and Zynq 7000 Devices](#) 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 Device LVDS

This section provides general guidelines for creating synchronous SGMII designs using the UltraScale 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 clocking resources and SelectIO Resources, see *UltraScale Architecture SelectIO Resources User Guide* ([UG571](#)) and *UltraScale Architecture Clocking Resources User Guide* ([UG572](#)).

#### 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 device in the device data sheet. Any I/O supporting a maximum 1.25 Gbps 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:

- Internal clocks generated by the MMCM, Clk208 and Clk104 are not required by the UltraScale device implementation of SGMII over LVDS design.
- An additional clock Clk312 (312.5 MHz) generated internally by the MMCM is required by the UltraScale 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](#)) and *UltraScale*

Architecture Clocking Resources User Guide (UG572).

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:** The design contains a cascade of delay elements IDELAY (idelay\_cal) and ODELAY used for delay calibration. Ensure that you place these delay elements carefully when pin planning.

---

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

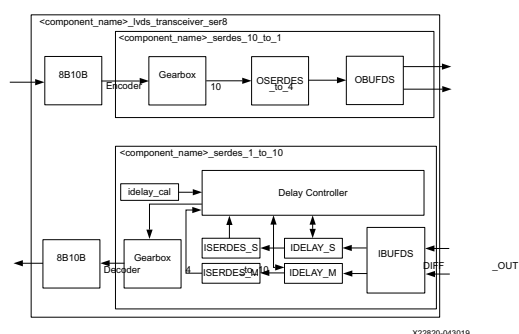
---

### Synchronous SGMII LVDS Transceiver for UltraScale Devices

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

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

**Figure: LVDS Transceiver Block Level for UltraScale 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 Gbps.

The receiver path has additional complexity. Serial data received at 1.25 Gbps is routed in parallel to two IODELAYE3 and ISERDESE3. The LVDS transceiver block uses the UltraScale device OSERDESE3, IODELAYE3s and ISERDESE3 elements. See the *UltraScale Architecture SelectIO Resources User Guide* (UG571) 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 Gbps.

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.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.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 Gbps. However, at 100 Mbps, every data byte run through the core should be repeated 10 times to achieve the required bit rate; at 10 Mbps, 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 [1G/2.5G Ethernet PCS/PMA or SGMII Using a Device-Specific Transceiver](#)).
- The [Synchronous SGMII LVDS Transceiver for UltraScale 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 Mbps, 100 Mbps, and 1 Gbps.
  - GMII logic for both transmitter and receiver paths; the GMII style 8-bit interface is run at 125 MHz for 1 Gbps operation; 12.5 MHz for 100 Mbps operation; 1.25 MHz for 10 Mbps operation.

## Asynchronous 1000BASE-X/SGMII over LVDS

This section provides the guidelines for creating asynchronous 1000BASE-X/SGMII interfaces using UltraScale/UltraScale+ device LVDS. This mode enables direct connection to external PHY devices without the use of an FPGA transceiver. 2.5G SGMII/2.5G BASE-X is unsupported for the LVDS physical interface.

**!! Important:** The Asynchronous 1000BASE-X/SGMII over LVDS implementation can fully support synchronous SGMII interfaces. This can be accomplished through the Clock Selection parameter in GUI.

In UltraScale devices, for the SGMII interface, an option is provided to choose between the Synchronous SGMII over LVDS solution that uses native components and the Asynchronous SGMII over LVDS solution that uses SelectIOtechnology using the parameter, EnableAsyncSGMII.

The synchronous SGMII over LVDS solution uses component mode I/Os such as ISERDES, OSERDES, IDELAY, ODELAY components whereas the asynchronous implementation uses native mode HSSIO components such as BITSlice and BITSlice\_CONTROL.

## Design Requirements

Timing closure of this interface can be challenging; perform the steps described in [Layout and Placement](#).

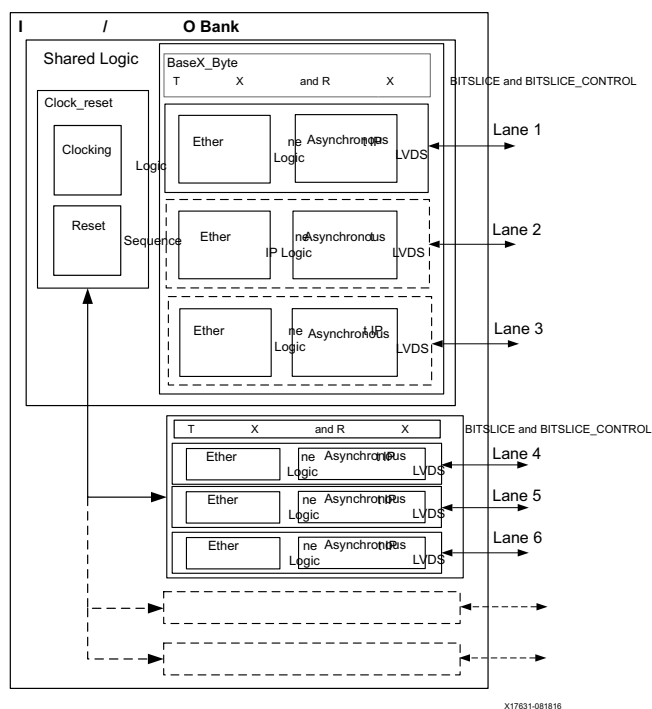
## Asynchronous LVDS Clocking and Reset Logic (Applicable for non Versal devices)

The asynchronous LVDS solution is an implementation where an external clock is provided to the design. This clock can be asynchronous to the incoming data stream within the limits specified in the Ethernet specifications. This 625 MHz differential clock is fed to an IBUFDS and the output drives the input of two PLLs. When implementing synchronous SGMII interfaces using this solution, the reference clock is user-selectable.

A PLL is used to generate multiple clocks at 312.5,125 MHz along with the 625,1250 MHz tx\_pll\_clock and rx\_pll\_clock driven to TX\_BITSLICE\_CONTROL and RX\_BITSLICE\_CONTROL. The clocking and reset logic is included in a separate module <component\_name>\_clock\_reset which is instantiated in the <component\_name>\_support module .

The core supports up to three instances of PCS/PMA lanes within a single BYTE\_GROUP. The core can be extrapolated to the full bank using multiple instances (up to four for an I/O bank) of three lane core instances. However one differential pair of clocks is required to drive the PLLs in the bank. This limits the maximum number of Ethernet links in an I/O bank to 11. This is shown in the following figure.

**Figure: Multi-Lane Asynchronous LVDS Core in a Full-Bank Design**



Typical use of the asynchronous 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 and reset logic that can be shared. The remaining instances can be generated using the Include Shared Logic in Example Design option. The reset logic contains the reset sequence required in native I/O mode. The reset sequence is used to calibrate the delays and provides the rx\_bt\_val value to the core logic. This value is used to calculate the number of taps required to maintain a relative difference of 400 ps between two RX\_BITSLICES.

## Asynchronous LVDS Clocking and Reset logic (Versal Devices)

Asynchronous LVDS solution for Versal implements the Advanced IO Wizard as a sub-core. An XPLL that is included within Advanced IO Wizard sub-core, is used to provide clocking to RX Gearbox, TX Gearbox and blocks present within the wizard. The XPLL generates clocks of frequencies 312.5 MHz and 1250Mhz, a 312.5Mhz clock output is connected to a BUFGE\_DIV to provide clocking to TX

The core supports up to three instances of PCS/PMA lanes to be used with a single instance of Advanced IO Wizard. As XPLL cannot be shared across the core instances, the maximum number of single/multi-lane cores supported per IO bank is 2. Shareable logic includes the clocking logic to generate `clk125m`, `pll_clk_in` and `ctrl_clk` along with an RIU based reset state machine. The shareable logic mentioned above is always included in the example design.

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 following figure shows sharing clock resources across three instantiations of the core. This can be extrapolated to a maximum of four instances within the same bank.

[illegible]

The steps provided are a useful guide, but additional knowledge is assumed. To help with the guidelines in this mode, see the *UltraScale Architecture SelectIO Resources User Guide (UG571)*. A working knowledge of the AMD Vivado Design Suite is also useful to locate particular clock buffers and slices.

- All the transmitter and receiver lanes should be within the same BYTE\_GROUP.
- All transmitter lanes should be within one nibble and receiver lanes should be within another nibble within the same byte group.
- Nibble selection and placement selection for lanes of the transmitter and receiver must match the Vivado IDE selection of placement constraints.
- The following I/O constraints related to EQUALIZATION, DQS\_BIAS, DIFF\_TERM, PRE\_EMPHASIS must be set appropriately for your application:

```
set property IOSTANDARD LVDS [get ports rxn]
```

```

set_property IOSTANDARD LVDS [get_ports rxp]
# Equalization can be set to EQ_LEVEL0-4 based on the loss in the channel. EQ_NONE is #an invalid option
set_property EQUALIZATION EQ_LEVEL0 [get_ports rxn]
set_property EQUALIZATION EQ_LEVEL0 [get_ports rxp]
#DQS_BIAS is to be set to TRUE if internal DC biasing is used - this is recommended. #If the signal is biased
externally on the board, should be set to FALSE
set_property DQS_BIAS TRUE [get_ports rxn]
set_property DQS_BIAS TRUE [get_ports rxp]
# DIFF_TERM is to be set to TERM_100 if internal Diff term is used - this is #recommended. If differential
termination is external on the board, should be set to #TERM_NONE
set_property DIFF_TERM_ADV TERM_100 [get_ports rxn]
set_property DIFF_TERM_ADV TERM_100 [get_ports rxp]
Transmit Pins
#LVDS_PRE_EMPHASIS can be set to TRUE/FALSE based on loss in the line if pre-emphasis #is desired or not. Note,
if PRE -emphasis is desired, ENABLE_PRE_EMPHASIS attribute #in TXBITSlice needs to be set to TRUE as well.
set_property LVDS_PRE_EMPHASIS FALSE [get_ports txn]
set_property LVDS_PRE_EMPHASIS FALSE [get_ports txp]
#IO standard has to be LVDS
set_property IOSTANDARD LVDS [get_ports txn]
set_property IOSTANDARD LVDS [get_ports txp]

```

## Lane Placement Parameters

### NumOfLanes

Number of PCS/PMA lanes to generate. This is applicable only for asynchronous 1000BASE-X/SGMII over LVDS.

### TxLane0\_Placement

This is the location of txp\_0/txn\_0 within TxNibble. Valid values are:

#### DIFF\_PAIR\_0

Indicates bitslice0 and bitslice1 to be used for the first lane.

#### DIFF\_PAIR\_1

Indicates bitslice2 and bitslice3 to be used for the first lane.

#### DIFF\_PAIR\_2

Indicates bitslice4 and bitslice5 to be used for the first lane.

### TxLane1\_Placement

This is the location of txp\_1/txn\_1 within TxNibble (if Number\_of\_Lanes is greater than 1). Valid values are:

#### DIFF\_PAIR\_0

Indicates bitslice0 and bitslice1 to be used for the second lane.

#### DIFF\_PAIR\_1

Indicates bitslice2 and bitslice3 to be used for the second lane.

#### DIFF\_PAIR\_2

Indicates bitslice4 and bitslice5 to be used for the second lane.

For the third lane the placement not selected for Lane 0 and Lane 1 is chosen.

### RxLane0\_Placement

This is the location of rxp\_0/rxn\_0 within RxNibble. Valid values are:

#### DIFF\_PAIR\_0

Indicates bitslice0 and bitslice1 to be used for the first lane.

#### DIFF\_PAIR\_1

Indicates bitslice2 and bitslice3 to be used for the first lane.

#### DIFF\_PAIR\_2

Indicates bitslice4 and bitslice5 to be used for the first lane.

### RxLane1\_Placement

This is the location of rxp\_1/rxn\_1 within RxNibble (if Number\_of\_Lanes is greater than 1). Valid values are:

#### DIFF\_PAIR\_0

Indicates bitslice0 and bitslice1 to be used for the second lane.

#### DIFF\_PAIR\_1

Indicates bitslice2 and bitslice3 to be used for the second lane.

#### DIFF\_PAIR\_2

Indicates bitslice4 and bitslice5 to be used for the second lane.

For third lane the placement not selected for Lane 0 and Lane 1 would be chosen.

**InstantiateBitslice0**

This is an indication to the logic to instantiate a Dummy BITSlice0.

If TRUE, then the core logic handles BITSlice0 insertion. If BITSlice0 insertion is required a dummy BITSlice0 is instantiated, and a dummy\_port\_in appears at the top level (depending on the RxNibbleBitslice0Used value). The dummy\_port\_in needs to be connected at the top level and the pin LOC'ed at the appropriate BITSlice0 location.

If FALSE, the IP logic does not instantiate BITSlice0.

Valid Values: TRUE/FALSE

**Note:** BITSlice0 instantiation is necessary in RxNibble if the pin is not used as per native mode use guidelines.

**RxNibbleBitslice0Used**

This is an indication to the core if the BITSlice0 of RX\_NIBBLE is used as the reference clock. This is enabled only when InstantiateBitslice0 is enabled.

If TRUE, the clock pins should be connected at the corresponding BITSlice0 location. Internal logic connects the clock to the BITSlice0 input. However, when Include Shared Logic in Example Design is selected the dummy\_port\_in should be connected to a single ended clock.

If FALSE, BITSlice0 is not used as a clock. Logic instantiates BITSlice0 based on the InstantiateBitslice0 value.

Valid Values: TRUE/FALSE

**Tx\_In\_Upper\_Nibble**

Indicates whether the TX is in the upper or lower nibble, If Tx\_In\_Upper\_Nibble=0 this means that the RX is in the upper nibble and the TX is in lower nibble and vice versa.

Valid Values: 0/1

**Layout and Placement in Versal Devices**

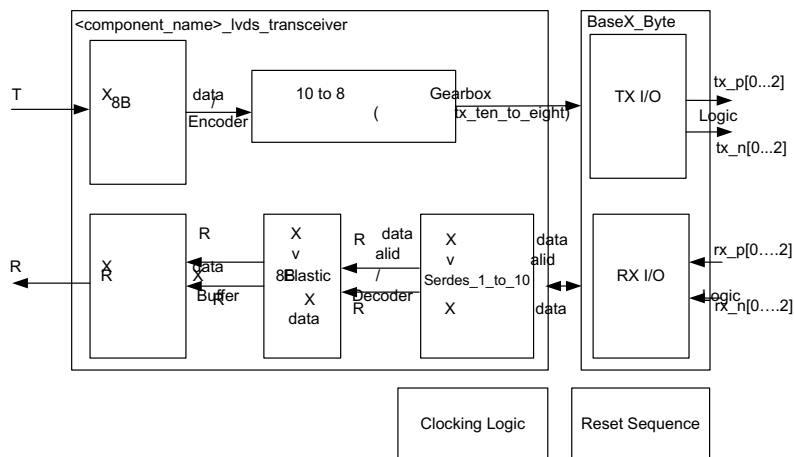
Advanced I/O planner is available for Versal devices during the I/O planning step. The IO bank and Nibble placement constraints can be chosen appropriately using the planner. In cases where the placement is sub-optimal, an error is thrown

**!! Important:** Ensure that you strictly adhere to the core constraints during core generation.

**Asynchronous LVDS Transceiver for UltraScale and Ultrascale+**

The LVDS transceiver block fully replaces the functionality otherwise provided by the device transceiver. This is only possible at a serial line rate of 1.25 Gbps. The following figure shows a block diagram of the asynchronous LVDS transceiver for UltraScale and UltraScale+ devices. This is split up into several sub-blocks which are described in further detail in the following sections.

**Figure: Asynchronous LVDS Transceiver Block**



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 10B8B Gearbox and the parallel data is then clocked out serially at a line rate of 1.25 Gbps.

The receiver path has additional complexity. Serial data received at 1.25 Gbps is routed in parallel to two RX\_BITSlices. By comparing 4-bit parallel data from these RX\_BITSlices the correct sampling point is computed for the incoming data stream and delays for RX\_BITSlices adjusted accordingly. Soft logic in the serdes\_1\_to\_10 module provides logic to find the correct sampling point of the received data by scanning the serial stream from the two parallel RX\_BITSlices.

The Serdes\_1\_to\_10 module provides 10-bit comma-aligned parallel data and a data valid signal at a higher clock rate of 312.5 MHz. The next block 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.

These 8-bit code groups are synchronous to 312.5 MHz local clock and are qualified with data valid. To convert this to the 125 MHz local clock, the clock correction buffer needs to be present which can insert or remove idles into or from the incoming data stream to

compensate for ppm difference between the far end clock and local clock. This ppm compensation is handled by the RX elastic buffer.

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

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

### 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 AMD Application Note, *Parameterizable 8B/10B Encoder* ([XAPP1122](#)) provides two possible approaches: a choice of a block RAM-based implementation or a LUT-based implementation. The LVDS example design uses the LUT-based implementation, but the application note can be used to show how to swap this for the block RAM-based approach if this better suits the 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  
<component_name>_encode_8b10b_lut_base.v
```

### tx\_ten\_to\_eight

This module is used to convert 10-bit data at 125 MHz to 8-bit data at 156.25 MHz. This data is then given to TX\_BITSLICE for serialization.

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/  
synth/lvds_transceiver/<component_name>_tx_ten_to_eight.v
```

### serdes\_1\_to\_10

This module adjusts input delays for the RX\_BITSLICE in order to sample the data in the middle of the data eye. Two RX\_BITSLICES are fed with the serial data but phase shifted with respect to each other by 400 ps (one half bit period). One of the RX\_BITSLICE is always kept at the boundary of serial data eye, ensuring that another RX\_BITSLICE is sampling the data at the center of the data eye. This module changes the delays of the RX\_BITSLICES dynamically while selecting the valid data from the two RX\_BITSLICES. This module is also responsible for comma alignment and gives out comma-aligned data at the interface along with a data valid signal, synchronous to the 312.5 MHz local clock.

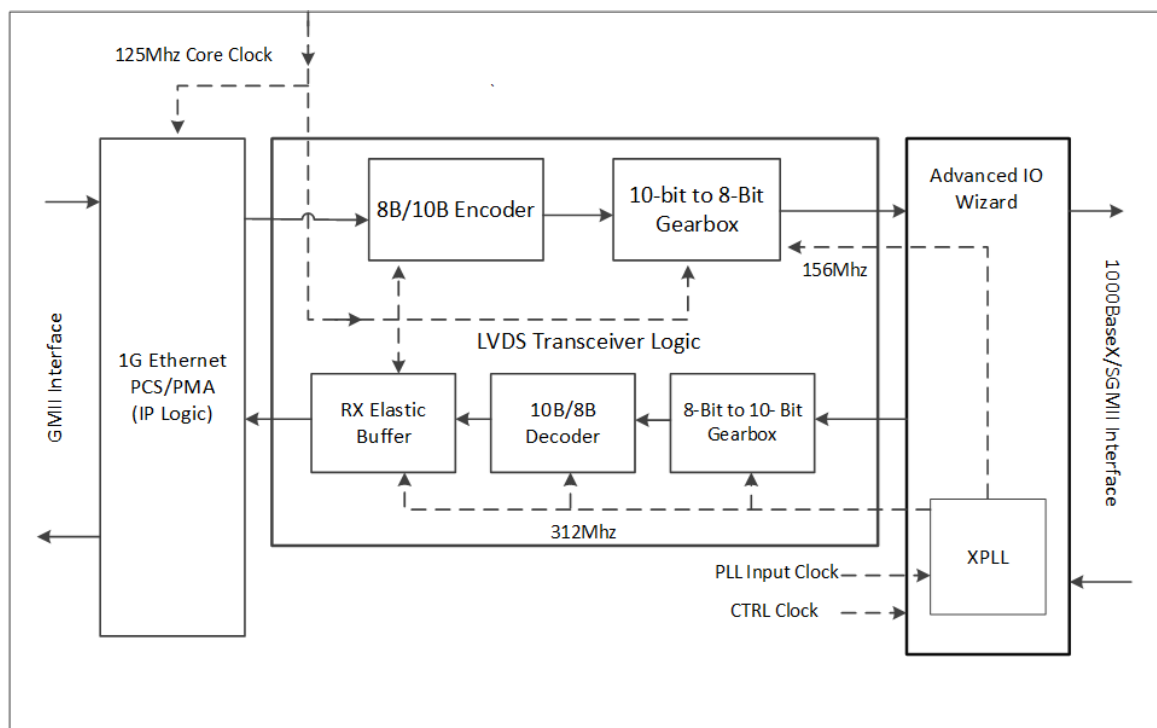
The following file describes the serdes\_1\_to\_10 functionality:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/  
synth/lvds_transceiver/<component_name>_serdes_1_to_10.v
```

### Asynchronous LVDS Transceiver for Versal devices

The LVDS Transceiver block replaces the functionality provided by a Device Transceiver but only at 1.25Gbps line rate. The further details of this block are as discussed below.

**Figure: Top-Level Hierarchy of Asynchronous LVDS Transceiver for Versal Devices**



X25378-061421

The Data Transmit logic is similar to that of UltraScale/UltraScale+ devices. The 8-bit parallel data from TX 10-bit to 8-bit gearbox, synchronous to 156.25Mhz clock is fed to the Advanced IO Wizard.

The Receiver path has been simplified as the Advanced IO Wizard sub-core detects the data sampling point through it's CDR logic and provides 8-bit parallel data synchronous to a 312.5Mhz clock, qualified with data valid. RX Gearbox converts 8-bit parallel data from the wizard, to 10-bit comma aligned parallel data at 312.5Mhz, qualified with data valid.

Data is then passed through a clock correction buffer and the 10b/8b decoder block in the same way as it is done for UltraScale/ UltraScale+ devices.

rx\_eight\_to\_ten

RX 8b/10b gearbox converts 8-bit data to 10-bit data and then performs comma alignment resulting in aligned data being output from this module. The data is synchronous to a 312.5Mhz clock and is qualified with data valid at all stages.

The above block, combined with the CDR logic that is built into Advanced IO wizard, replaces the function of serdes\_1\_to\_10 block present in UltraScale/UltraScale+ designs.

The following file describes the RX gearbox functionality:

synth/lvds\_transceiver/<component\_name>\_rx\_eight\_to\_ten.v

## The Ten-Bit Interface

This section provides general guidelines for creating 1000BASE-X, SGMII or Dynamic Switching designs using the Ten-Bit Interface (TBI). 2.5G is not supported in TBI mode.

For information on the ten-bit example design see [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.

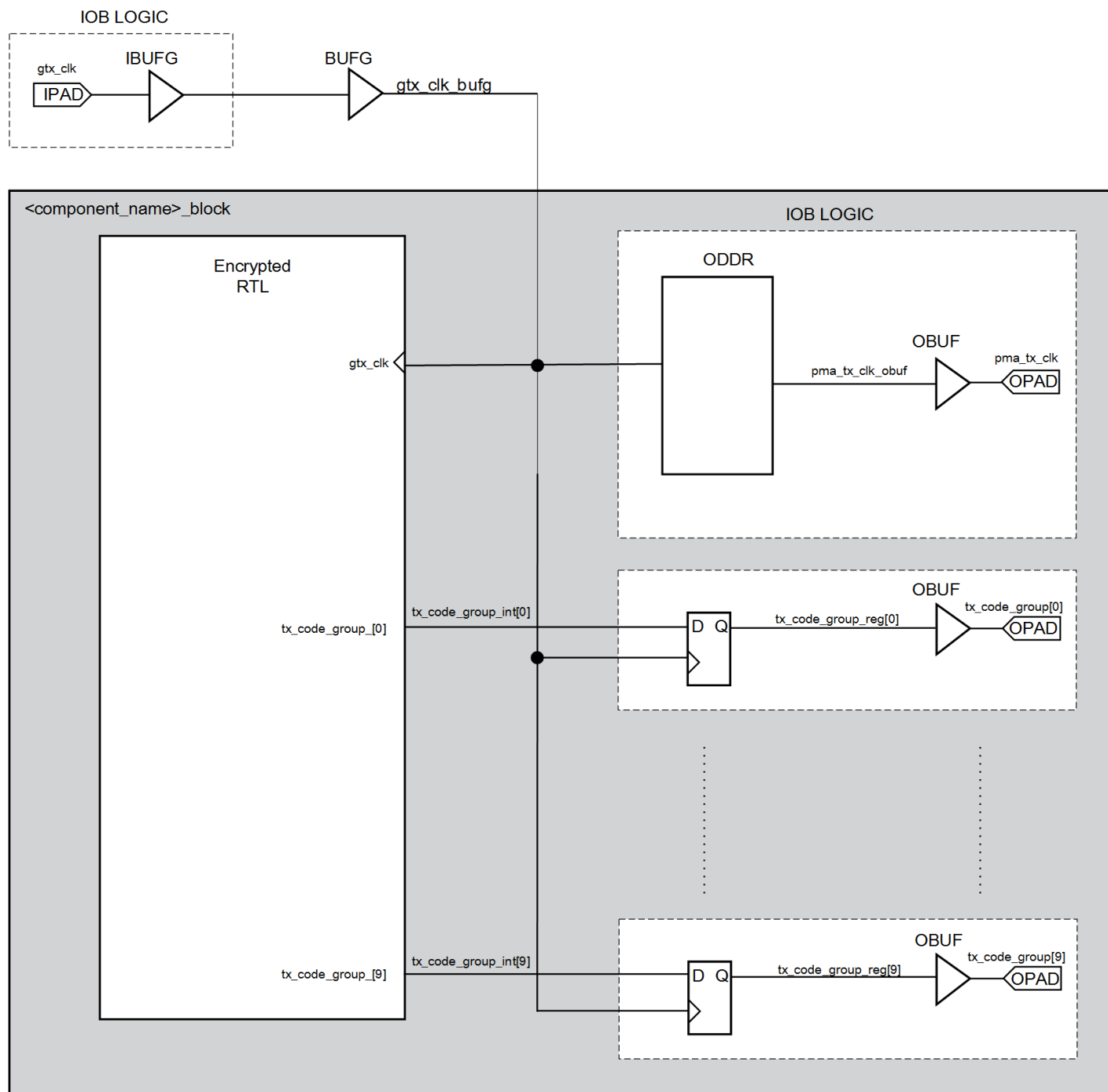
#### Transmitter Logic

The following figure 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.

The following figure 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.

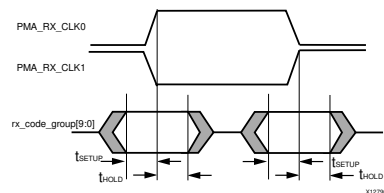
**Figure: Ten-Bit Interface Transmitter Logic**



X12881

Receiver Logic

**Figure: Input TBI timing**



The previous figure shows the input timing for the TBI interface as defined in IEEE802.3-2008 clause 36.

**Note:** 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.

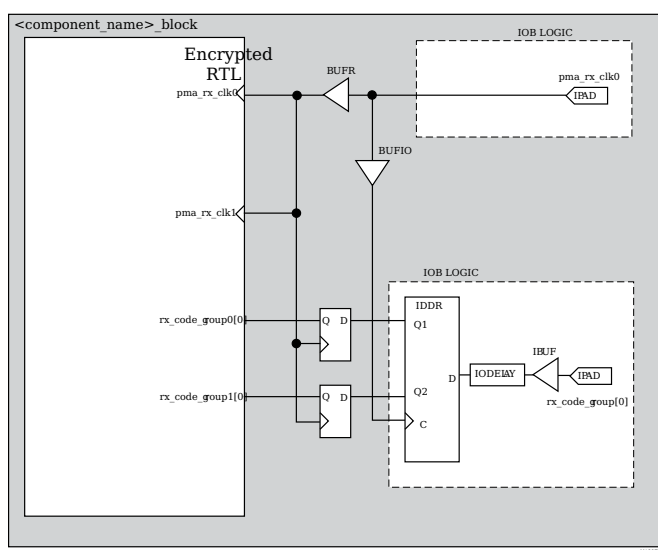
However, the data sheet for a particular external SerDes device that connects to the TBI might well specify that this is the case; that pma\_rx\_clk0 and pma\_rx\_clk1 are exactly 180° out of phase. If this is the case, the TBI receiver clock logic can be simplified by ignoring the pma\_rx\_clk1 clock altogether, and simply using both the rising and falling edges of pma\_rx\_clk0.

For this reason, the following sections describe two different alternative 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: Ten-Bit Interface Receiver Logic - Kintex 7 Devices (Example Design)**



The device logic used by the example design delivered with the core is shown in the previous figure. 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.

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 BUFI0 and BUFR regional clock routing for the pma\_rx\_clk0 input clock, as shown in the previous figure.

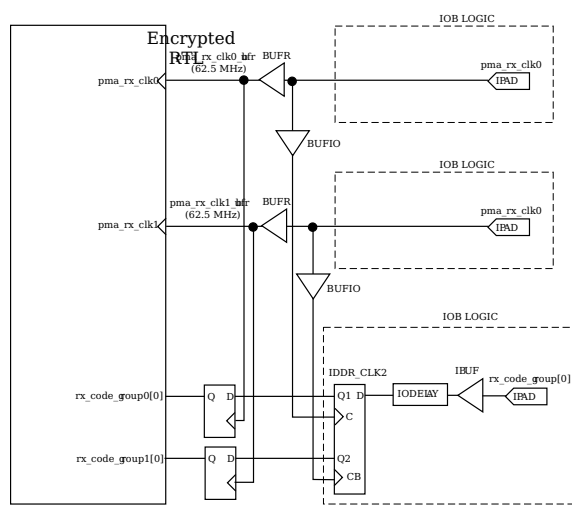
In the previous figure's implementation, a BUFI0 is used to provide the lowest form of clock routing delay from input clock to input rx\_code\_group[9:0] signal sampling at the device IOBs. However, this creates placement constraints; a BUFI0 capable clock input pin must be selected for pma\_rx\_clk0, and all rx\_code\_group[9:0] input signals must be placed in the respective BUFI0 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: Alternate TBI Receiver Logic - Kintex 7 Devices**

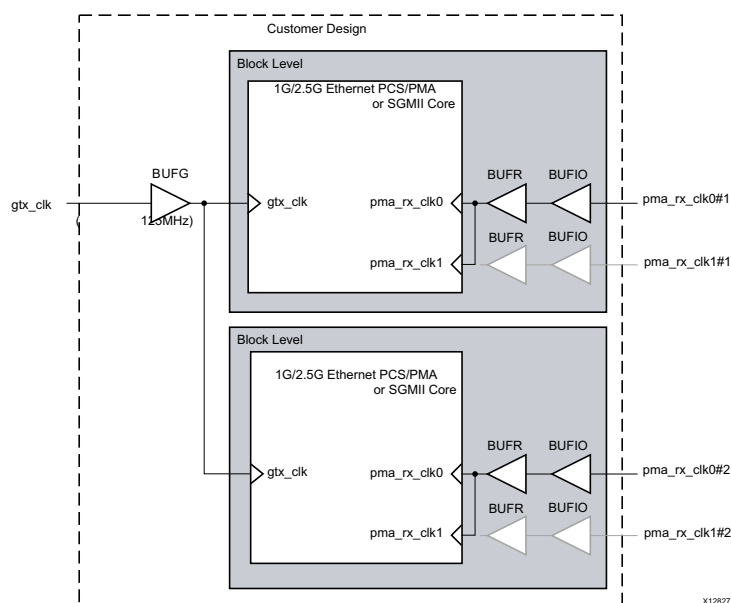


This logic from Method 1 relies on pma\_rx\_clk0 and pma\_rx\_clk1 being exactly 180° out of phase with each other because the falling edge of pma\_rx\_clk0 is used in place of pma\_rx\_clk1. See the data sheet for the attached SerDes to verify that this is the case. If not, the logic of the previous 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 1](#) 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: Clock Management, Multiple Core Instances with the TBI**



The above figure 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.

The figure 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 Mbps, 100 Mbps, and 1 Gbps.
- GMII logic for both transmitter and receiver paths; the GMII style 8-bit interface is run at 125 MHz for 1 Gbps operation; 12.5 MHz for 100 Mbps operation; 1.25 MHz for 10 Mbps 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 Mbps and 100 Mbps Ethernet speeds in addition to the nominal 1 Gbps 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 SoC processing subsystem  
The core can be integrated with ENET0 or ENET1 through the Extended multiplexed I/O (EMIO) interface. 2.5G mode is not applicable when interfacing to the Zynq PS. See [Interfacing to Other Cores](#).
- Gigabit Ethernet MAC (GEM0, GEM1) in Versal Control, Interfaces and Processing System.  
The core can be integrated to GEM0 and GEM1 peripherals of Versal CIPS through Extended Multiplexed I/O (EMIO) Interface. 2.5G speed is not applicable while interfacing with Versal CIPS.

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 [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. For 2.5G mode, the GMII interface is over-clocked at 312.5 MHz.

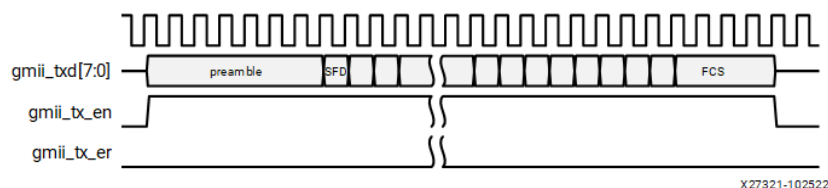
#### GMII Transmission

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 [Clocking](#).

##### Normal Frame Transmission

Normal outbound frame transfer timing is shown in the following figure. This figure shows that an Ethernet frame is preceded 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.

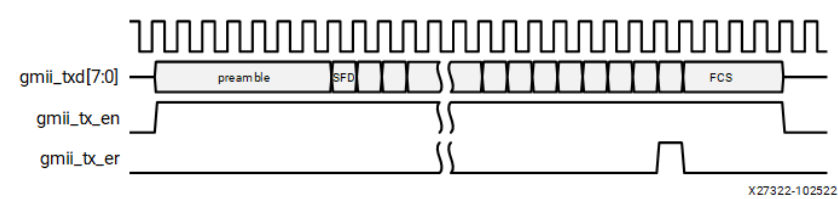
**Figure: GMII Normal Frame Transmission**



##### Error Propagation

A corrupted frame transfer is shown in the following figure. 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: GMII Error Propagation Within a Frame**



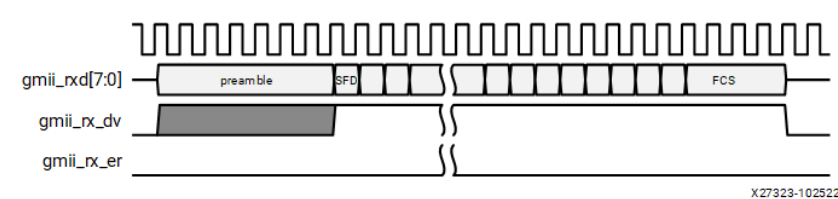
### GMII Reception

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 [Clocking](#).

#### Normal Frame Reception

The timing of normal inbound frame transfer is shown in the following figure. This shows that Ethernet frame reception is preceded by a preamble field. The *IEEE 802.3-2008* specification (see clause 35) allows for up to all of the seven preamble bytes that proceed the Start of Frame Delimiter (SFD) to be lost in the network. The SFD is always present in well-formed frames.

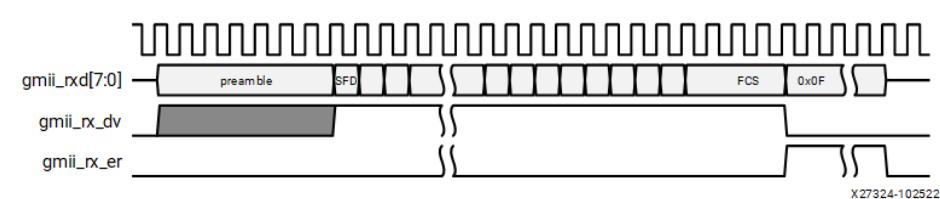
**Figure: GMII Normal Frame Reception**



#### Normal Frame Reception with Extension Field

In accordance with the *IEEE 802.3-2008*, clause 36 , 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 the following figure. See [1000BASE-X State Machines](#) for more information. This is not an error condition and can occur even for full-duplex frames.

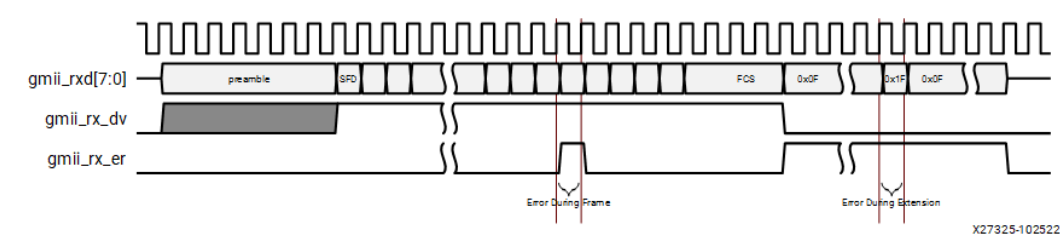
**Figure: GMII Normal Frame Reception with Carrier Extension**



#### 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 as shown in the following figure. 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 the following figure.

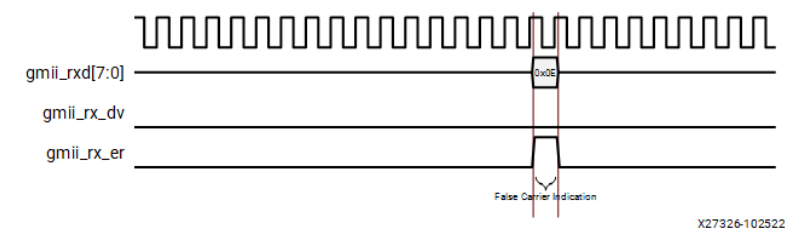
**Figure: GMII Frame Reception with Errors**



#### False Carrier

The following figure 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: False Carrier Indication**



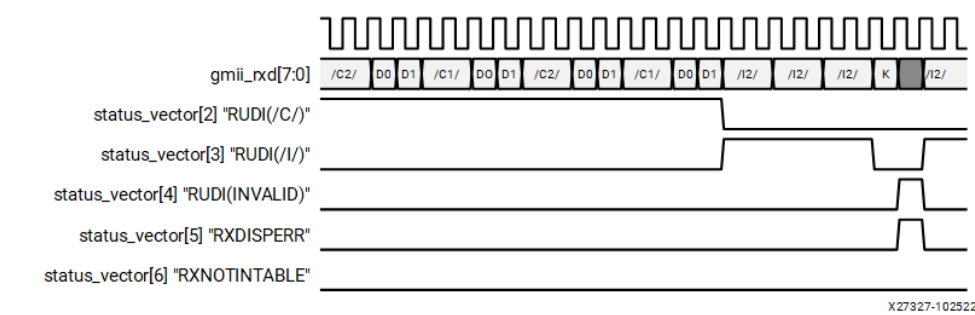
### status\_vector[15:0] Signals

The following figure shows an error occurring in the second clock cycle of an `/I/` idle sequence. See [Table 3](#) 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 3](#). The following figure 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.

**Figure: status\_vector[4:2] Timing**



The above figure 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 is identical to that of the `RXDISPERR` signal shown in the previous figure.

### Using the Client-Side GMII for the SGMII Standard

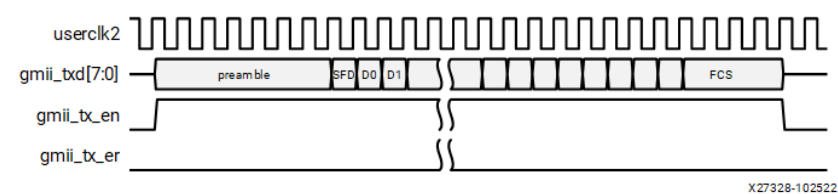
When the core is generated for the SGMII standard, changes are made to the core that affect the PCS management registers and the auto-negotiation function (see [Select Standard](#)). However, the datapath through both transmitter and receiver sections of the core remains unchanged. For the 2.5G mode the GMII interface is over-clocked at 312.5 MHz. 250 Mbps and 25 Mbps modes are not supported.

### GMII Transmission

#### 1/2.5 Gbps Frame Transmission

The timing of normal outbound frame transfer is shown in the following figure. At 1/2.5 Gbps speed, the operation of the transmitter GMII signals remains identical to that described in [Using the Client-Side GMII for the 1000BASE-X Standard](#).

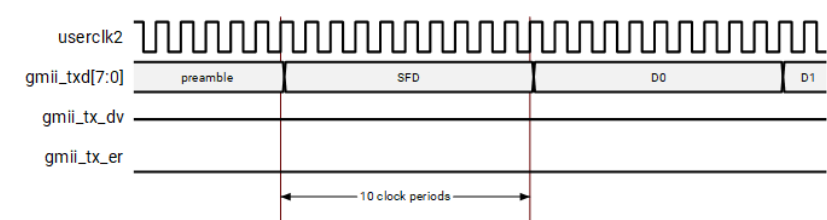
**Figure: GMII Frame Transmission at 1/2.5 Gbps**



#### 100 Mbps Frame Transmission

At 100 Mbps 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 100 Mbps, every byte of the MAC frame (from preamble to the Frame Check Sequence field, inclusive) should be repeated for 10 clock periods to achieve the desired bit rate, as shown in the following figure. 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 Ethernet MAC peripherals of the processing subsystem present in Zynq and Versal devices, the core takes care of converting the 4-bit MII interface to the 8 bits required by the core. In all other cases the core expects 8 bits from the client logic.

**Figure: GMII Data Transmission at 100 Mbps**



10 Mbps Frame Transmission

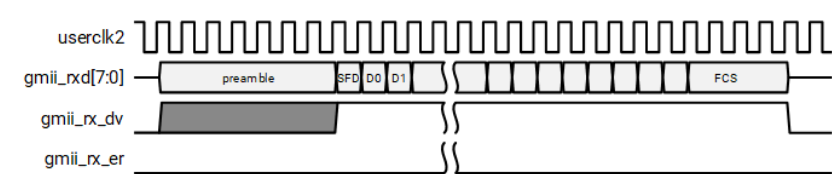
At 10 Mbps the operation of the core remains unchanged. It is the responsibility of the client logic (for example, an Ethernet MAC), to enter data at the correct rate. When operating at 10 Mbps, every byte of the MAC frame (from preamble to the frame check sequence field, inclusive) should be repeated 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 Ethernet MAC peripherals of the processing subsystem present in Zynq and Versal devices, the core takes care of converting the 4-bit MII interface to the 8 bits required by the core. In all other cases the core expects 8 bits from the client logic.

GMII Reception

1/2.5 Gbps Frame Reception

The timing of a normal inbound frame transfer is shown in the following figure. At 1/2.5 Gbps, the operation of the receiver GMII signals is identical to that described in [Using the Client-Side GMII for the 1000BASE-X Standard](#).

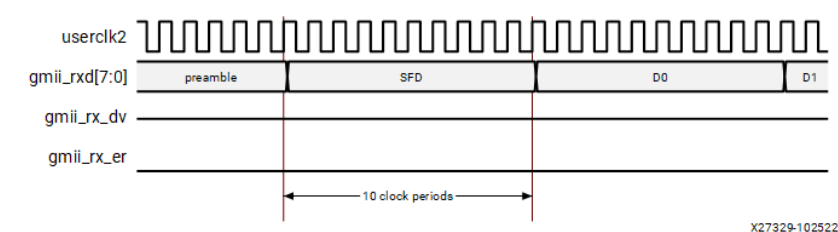
**Figure: GMII Frame Reception at 1/2.5 Gbps**



100 Mbps Frame Reception

At 100 Mbps, the operation of the core is unchanged. When operating at a speed of 100 Mbps, every byte of the MAC frame (from the preamble to the frame check sequence field, inclusive) is repeated for 10 clock periods to achieve the desired bit rate. See the following figure. Only when the core is connected to Ethernet MAC peripherals of the processing subsystem present in Zynq and Versal devices, 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: GMII Data Reception at 100 Mbps**



X27329-102522

10 Mbps Frame Reception

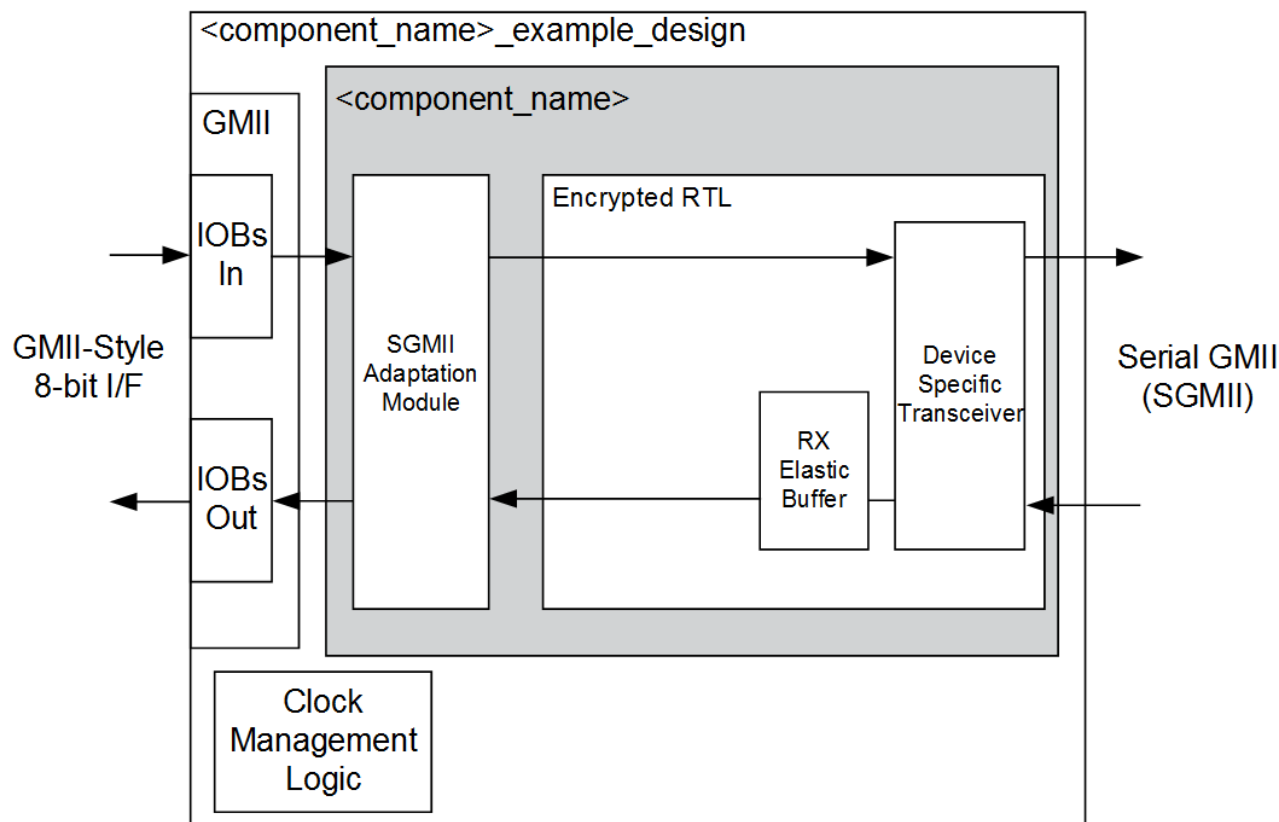
The operation of the core remains unchanged. When operating at a speed of 10 Mbps, every byte of the MAC frame (from preamble

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 Ethernet MAC peripherals of the processing subsystem present in Zynq and Versal devices, 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 the following figure 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: Block Level Diagram of an SGMII Example Design**



X22817-042919

Because the GMII of the core always operates at 125 MHz (312.5 MHz for the 2.5G data rate), the core does not differentiate between the three SGMII speeds of operation (for the 1 Gbps maximum data rate); it always effectively operates at 1/2.5 Gbps. However, as described in [Using the Client-Side GMII for the 1000BASE-X Standard](#), at 100 Mbps, every data byte run through the core is repeated ten times to achieve the required bit rate; similarly, at 10 Mbps, 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 (as shown in the following figure) creates a GMII-style interface that drives/samples the GMII data and control signals at the following frequencies:

- 125/312.5 MHz when operating at a speed of 1/2.5 Gbps (with no repetition of data bytes)
- 12.5 MHz at a speed of 100 Mbps (each data byte is repeated and run through the core 10 times)
- 1.25 MHz at a speed of 10 Mbps (each data byte is repeated and run through the core 100 times)

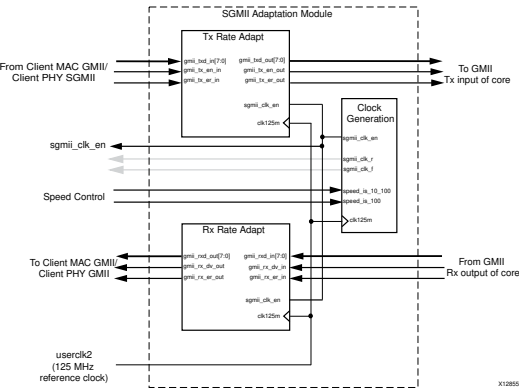
When the core is connected to Ethernet MAC peripherals of the processing subsystem present in Zynq and Versal devices, the SGMII adaptation module performs the additional function of converting the 8 bits from the core to a 4-bit MII interface and vice versa. The function of the SGMII adaptation module 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 Gbps). 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, to the AMD Tri-Mode Ethernet MAC core directly (see [Interfacing to Other Cores](#)). The GMII interface of the SGMII adaptation module can be 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.

### SGMII Adaptation Module Top Level

The SGMII adaptation module is described in several hierarchical submodules as shown in the following figure. These submodules are

each described in separate HDL files and are described in the following sections.

**Figure: SGMII Adaptation Module**

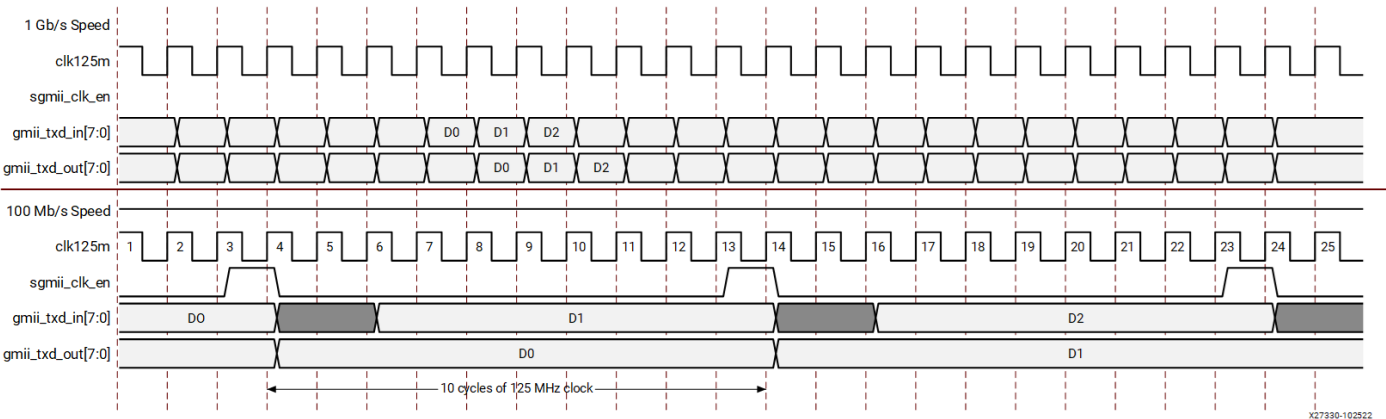


### Transmitter Rate Adaptation Module

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 1G/2.5G Ethernet PCS/PMA or SGMII netlist. The 1 Gbps and 100 Mbps cases are shown in the following figure. 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 the `sgmii_clk_en` signal (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: Transmitter Rate Adaptation Module Data Sampling**

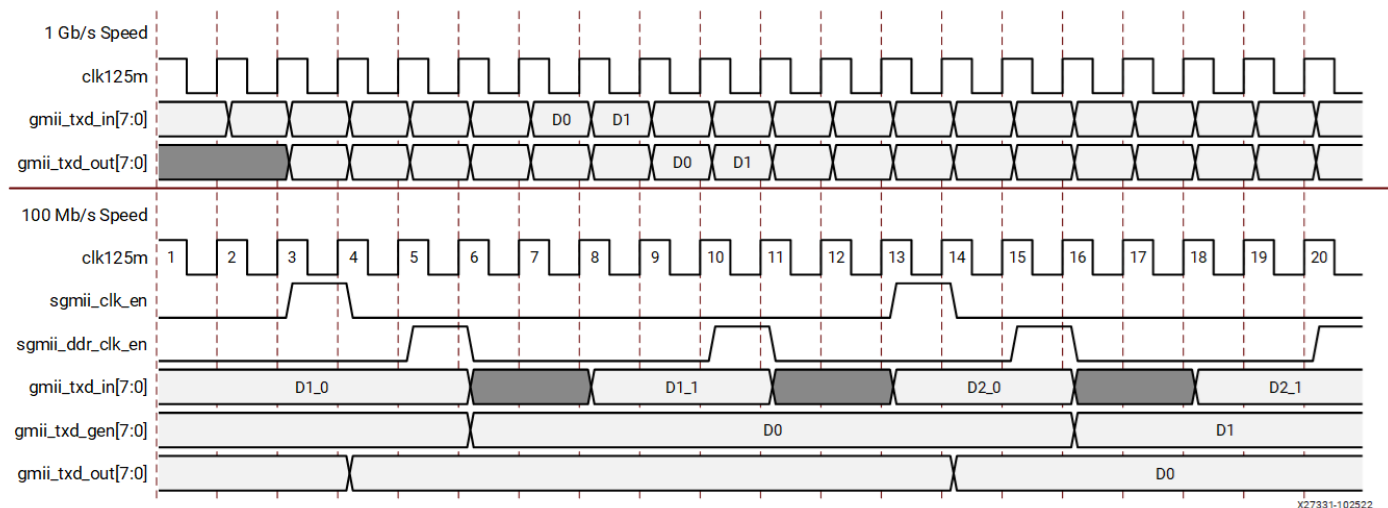


Interfacing with Ethernet MAC Peripherals in Device PS

When the speed is 1 Gbps, the data is received on the 125 MHz clock (`clk125m`). When a speed of 10/100 Mbps 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 the `sgmii_ddr_clk_en` signal (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 the `sgmii_clk_en` signal (internally derived from the Clock Generation module) as a clock enable. It is possible that the Start of Frame Delimiter (SFD) could have been skewed across two separate bytes, so the 8-bit SFD code is detected, and if required, is realigned across the 8-bit datapath. The 1 Gbps and 100 Mbps cases are shown in the following figure.

**Figure: 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 1G/2.5G Ethernet PCS/PMA or SGMII core. This data is sampled and sent out of the GMII receiver interface for the attached client MAC/External PHY. The 1 Gbps and 100 Mbps cases are shown in the following figure.

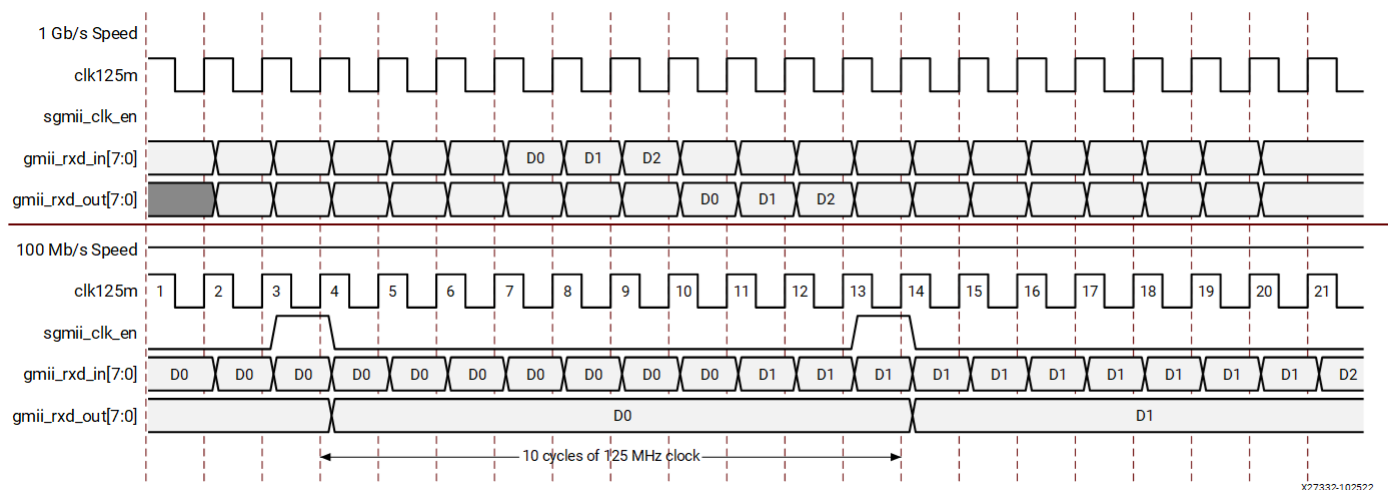
At 1 Gbps, the data is valid on every clock cycle of the 125 MHz reference clock (clk125m). Data received from the core is clocked straight through the Receiver Rate Adaptation module.

At 100 Mbps, the data is repeated for a 10 clock period duration of clk125m; at 10 Mbps, 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 the following figure. 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: Receiver Rate Adaptation Module Data Sampling**



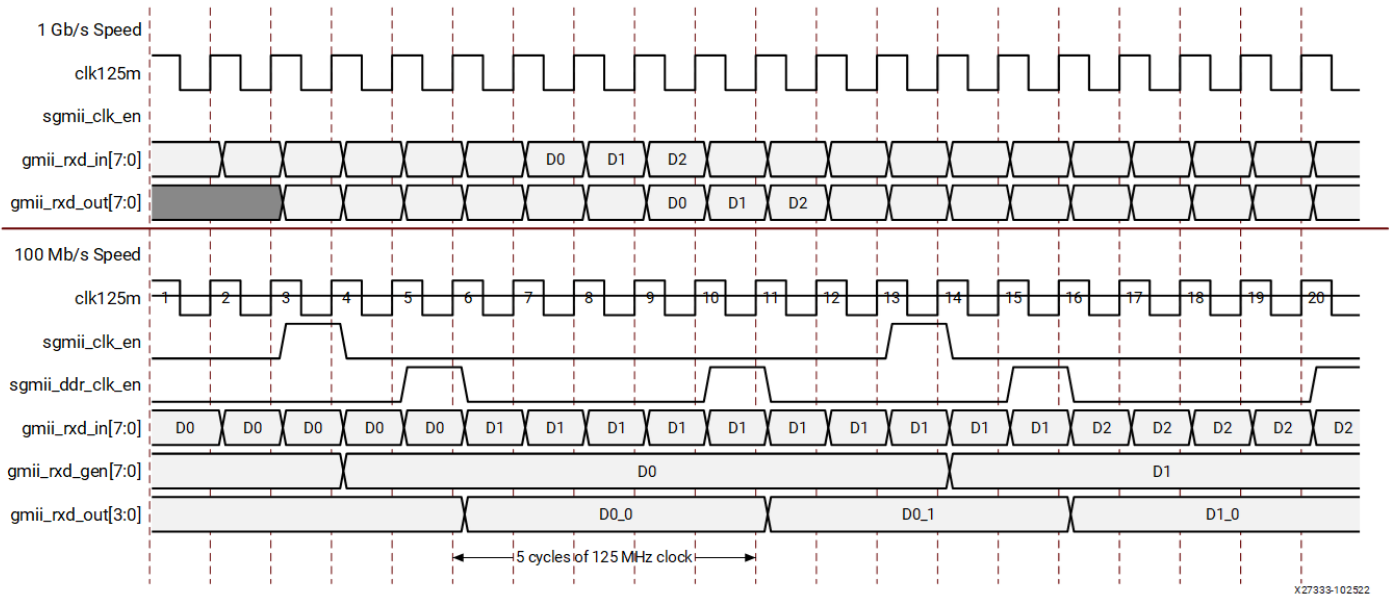
### Interfacing with Ethernet MAC Peripherals in Device PS

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 Gbps and 100 Mbps cases are shown in the following figure.

At 1 Gbps 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 Mbps, the data is repeated for a 10 clock period duration of clk125m; at 10 Mbps, 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 the clock generation module. Then the lower half of the byte is sent on the LSB four bits of gmii\_rxd\_out[3:0] followed by the upper nibble. This operation is done using the sgmii\_ddr\_clk\_en signal generated internally in the clock generation module.

**Figure: Receiver Rate Adaptation Module Data Sampling**



### Clock Generation

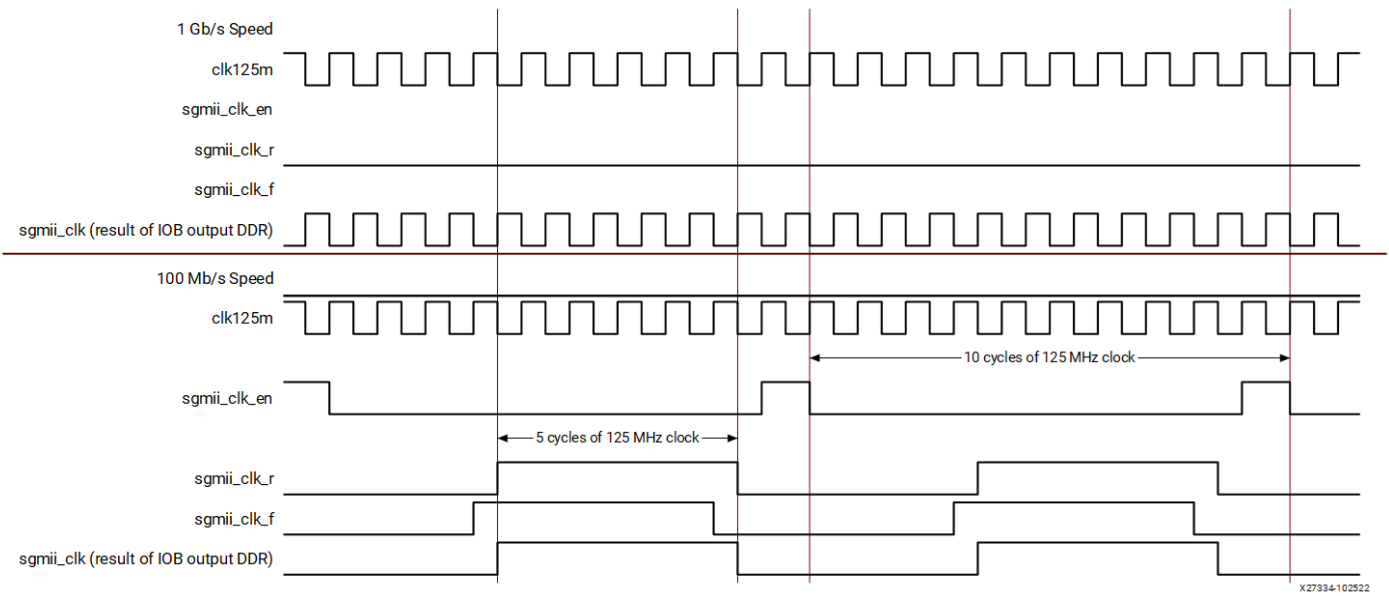
Interfacing with Client Proprietary Logic/IP Catalog Tri-Mode Ethernet MAC

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 Gbps
- 12.5 MHz at an operating speed of 100 Mbps
- 1.25 MHz at an operating speed of 10 Mbps

The following figure shows the output clock enable signal for the Clock Generation module at 1 Gbps and 100 Mbps speeds.

**Figure: Clock Generator Output Clocks and Clock Enable**

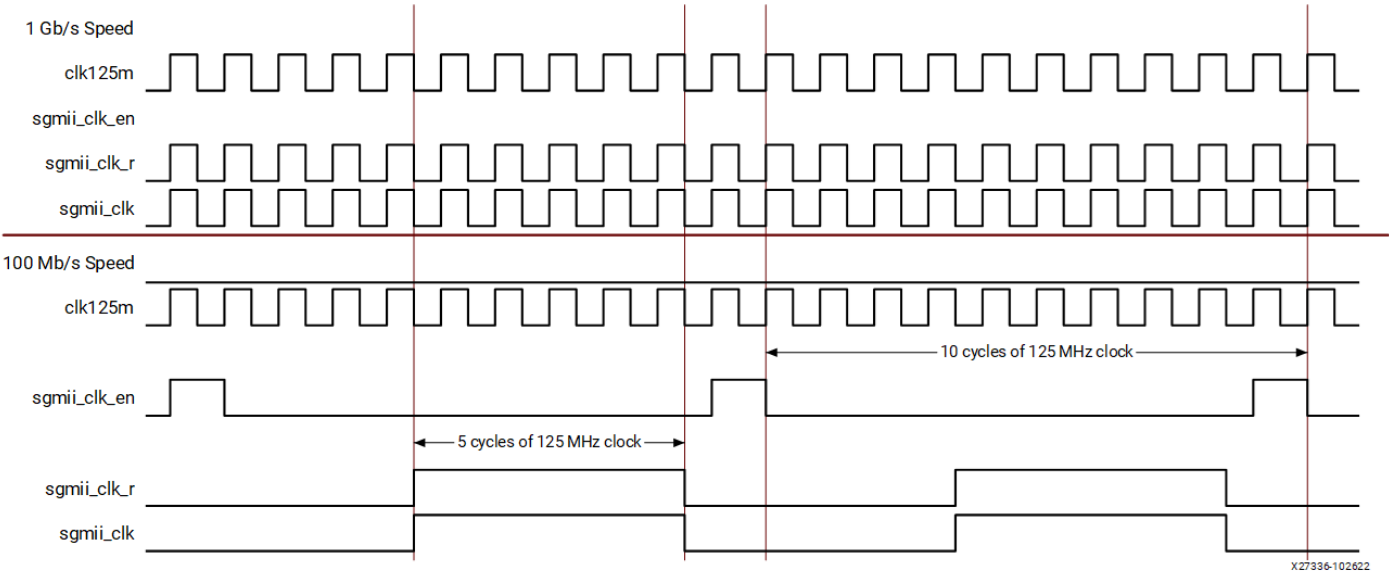


The above figure also shows the formation of the sgmii\_clk\_r and sgmii\_clk\_f signals. These are used only in the example design delivered with the core, where they are routed to a device IOB DDR output register. This provides SGMII clock forwarding at the correct frequency; these signal can be ignored when connecting the core and SGMII Adaptation module to internal logic.

Interfacing with Ethernet MAC Peripherals in Device PS

This module creates the sgmii\_clk\_en and sgmii\_ddr\_clk\_en clock enable signals for use throughout the SGMII adaptation module. The following figure shows the clock enable signal for the Clock Generation module at 1 Gbps and 100 Mbps speeds.

**Figure: Clock Enable Signal for the Clock Generation Module**



The above figure also shows the formation of the sgmmi\_clk\_r signal. sgmmi\_clk\_r is forwarded to the Ethernet Mac Peripheral through gmii\_tx\_clk port and gmii\_rx\_clk port. This provides SGMII clock forwarding at the correct frequency.

**Note:**

1. The sgmmi\_clk\_f signal is not used in this case.
2. The sgmmi\_clk\_en is not provided as an output but used internally within the SGMII adaptation module.

The sgmmi\_clk\_r frequencies for the different modes are:

- 125 MHz at an operating speed of 1 Gbps
- 25 MHz at an operating speed of 100 Mbps
- 2.5 MHz at an operating speed of 10 Mbps

## 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](#). For 2.5G mode Auto-Negotiation function is the same as the 1G mode. 250 Mbps and 25 Mbps modes are not supported for 2.5G SGMII.

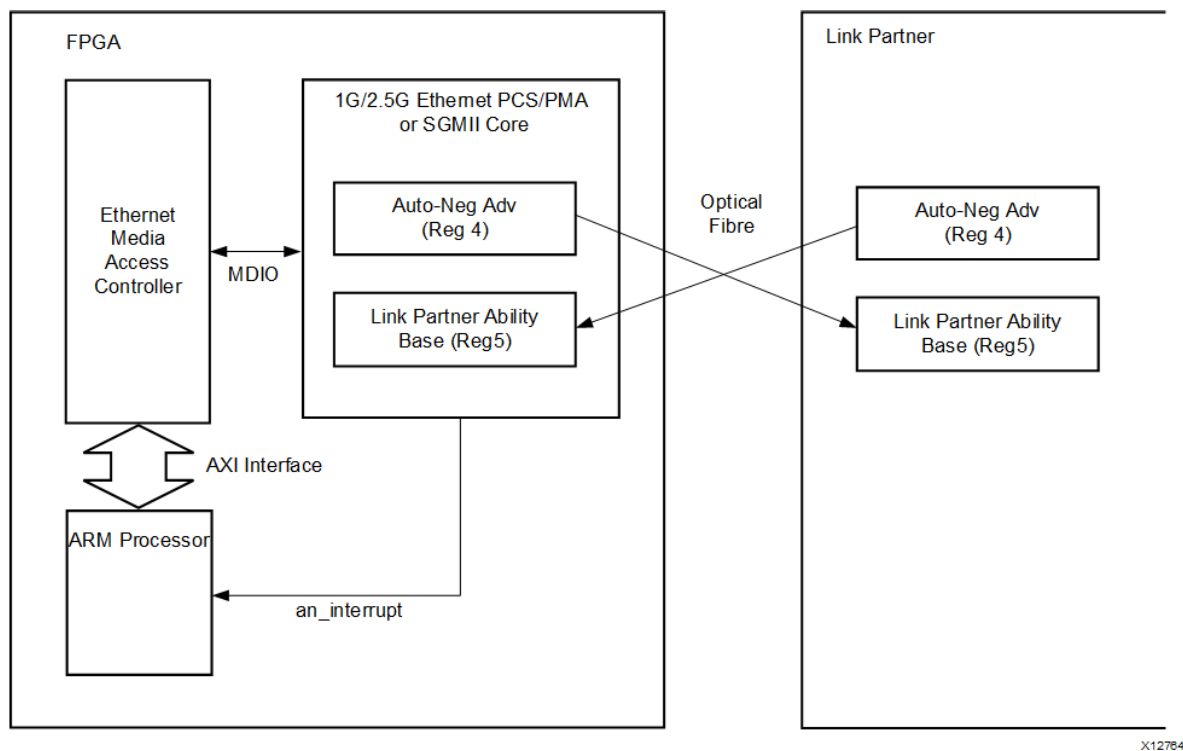
### Overview of Operation

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

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

### 1000BASE-X Standard

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



X12784

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. The previous figure 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 [Configuration and Status Vectors](#)).
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 the 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 [Configuration and Status Vectors](#) 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 Arm® 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](#)).

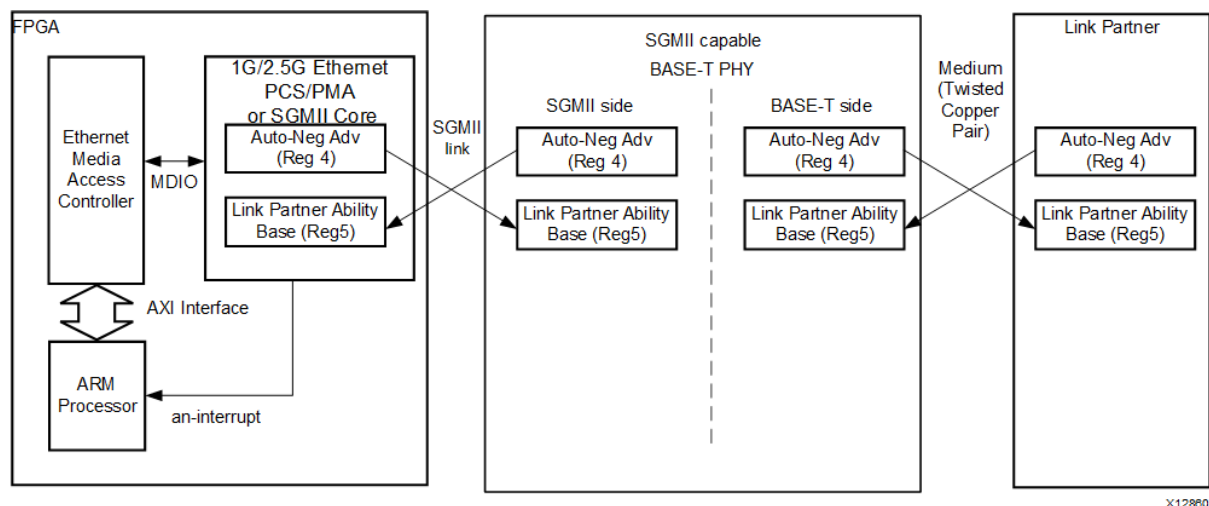
## SGMII Standard

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

The following figure 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.

**Figure: 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 the previous figure, 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 the previous figure. This transfers the results of the Link Partner auto-negotiation across the SGMII and is the only auto-negotiation observed by the core.

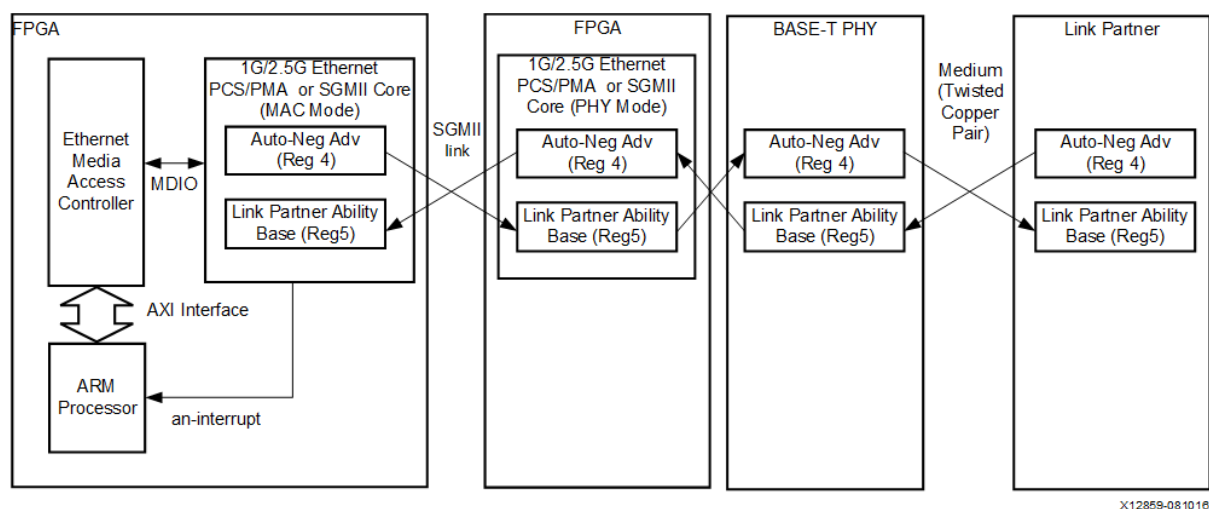
This SGMII auto-negotiation function, summarized previously, leverages the 1000BASE-X PCS/PMA auto-negotiation function but contains two differences.

- 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

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

The following figure shows the operation of SGMII auto-negotiation. Additional information about SGMII Standard auto-negotiation is provided in the following sections.

**Figure: 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 the previous figure, 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 through 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

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

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](#). Dynamic Switching between 2500BASE-X and 2.5G SGMII is not supported by the core.

### Typical Application

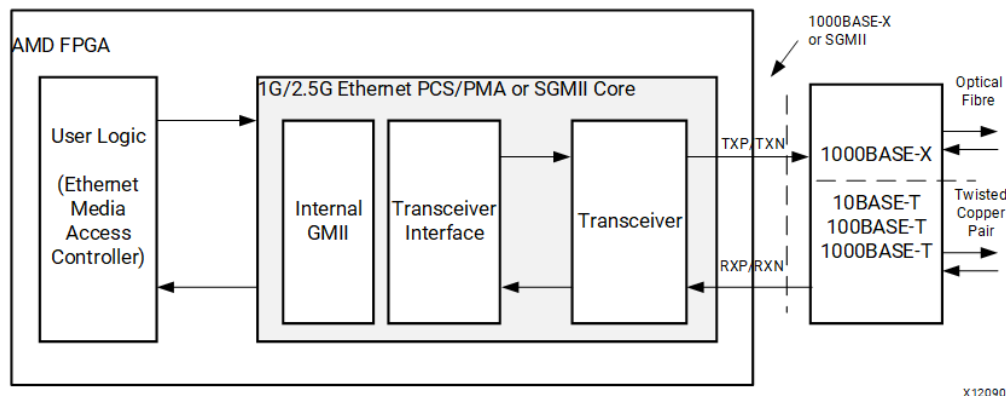
The following figure shows a typical application for the core with the ability to dynamically switch between 1000BASE-X and SGMII standards.

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 1G/2.5G Ethernet PCS/PMA or SGMII core is shown connected to an embedded Ethernet Media Access Controller (MAC), for example, the AMD Tri-Mode Ethernet MAC core.

**Figure: Typical Application for Dynamic Switching**



## Operation of the Core

### Selecting the Power-On/Reset Standard

The external port of the core, `basex_or_sgmmi` (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_sgmmi` 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 1G SGMII standard of the core can be switched at any time by writing to the Dynamic Switching Register 17 (see [Configuration and Status Vectors](#)). 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 or 2500BASE-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. AMD 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 1G/2.5G Ethernet 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 (from v9.0) core provides support for operation at 10 Mbps, 100 Mbps, 1 Gbps, and 2.5 Gbps.



**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 1G/2.5G Ethernet PCS/PMA or SGMII core. This constraint is due to the increased latency introduced by the 1G/2.5G Ethernet 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 v8.1 core supports UltraScale, Zynq 7000, Virtex 7, Kintex 7, and Artix 7 devices. The TEMAC v9.0 core supports 2.5G mode for UltraScale, UltraScale+, Versal, Zynq 7000 (except GTP-based devices), Virtex 7, and Kintex 7 devices.

The 1G/2.5G Ethernet PCS/PMA or SGMII core can also be integrated with a single device with either of the Ethernet MAC instances in the Zynq 7000 (ENET0/ENET1), Zynq UltraScale+, and Versal (GEM) device processing subsystem to extend the system functionality to include the Ethernet MAC sublayer. ENET0/1 MACs provide support for operation at 10 Mbps, 100 Mbps, and 1 Gbps.

## Integration of the TEMAC for 1000BASE-X or 2500BASE-X

In this section, it is assumed that the TEMAC core is generated with only 1 Gbps or 2.5 Gbps Ethernet speed and full-duplex only support. This provides the optimal solution.

### Ten-Bit Interface Implementation

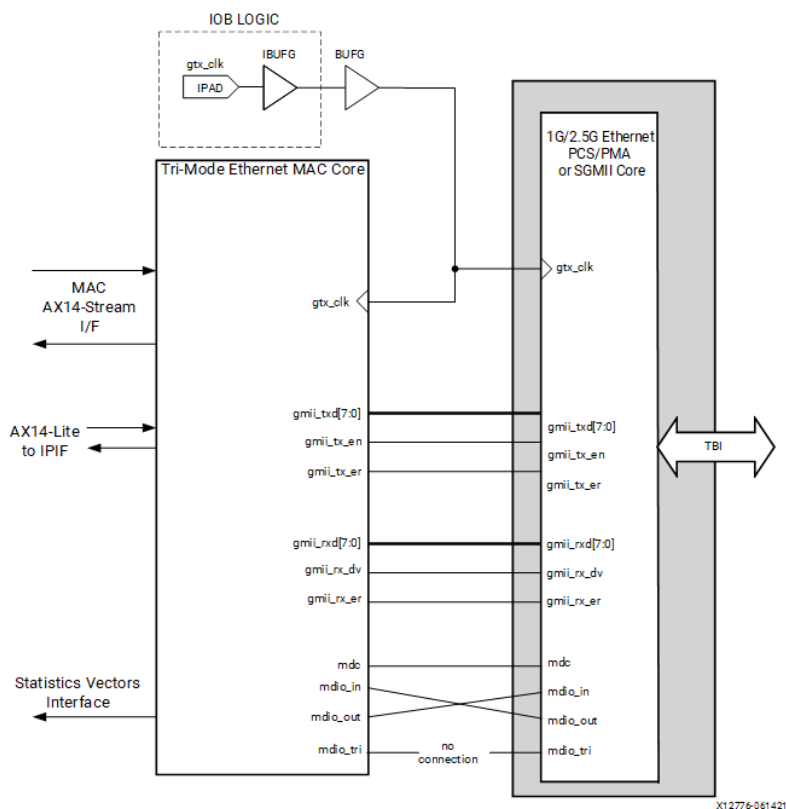
The following figure shows the connections and clock management logic required to interface the 1G/2.5G Ethernet PCS/PMA or SGMII core (used in 1000BASE-X mode with the TBI) to the TEMAC core.

**Important:** The TEMAC core must be generated with the “interface” variable set as “Internal” for interfacing to the 1G/2.5G Ethernet 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 1G/2.5G Ethernet PCS/PMA or SGMII core.
- Due to the embedded receive elastic buffer in the 1G/2.5G Ethernet 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: Core with TBI Connected to TEMAC Core**



### Transceiver Implementation



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



- 2024-03-20, 17:11

## 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 or 2500BASE-X only designs, see [Transceiver Implementation](#).

### Ten-Bit Interface Implementation

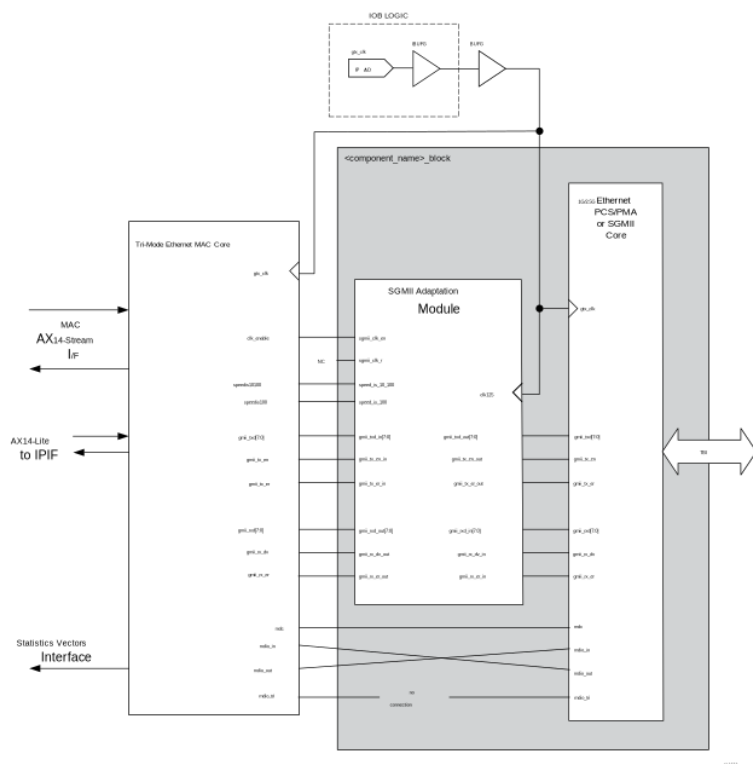
shows the connections and clock management logic required to interface the core (in SGMII mode with the TBI) to the TEMAC core. The 2.5G mode is not supported in TBI mode.

**!! Important:** The TEMAC core must be generated with “interface” variable set as “Internal” for interfacing with 1G/2.5G Ethernet 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 1G/2.5G Ethernet PCS/PMA or SGMII core.

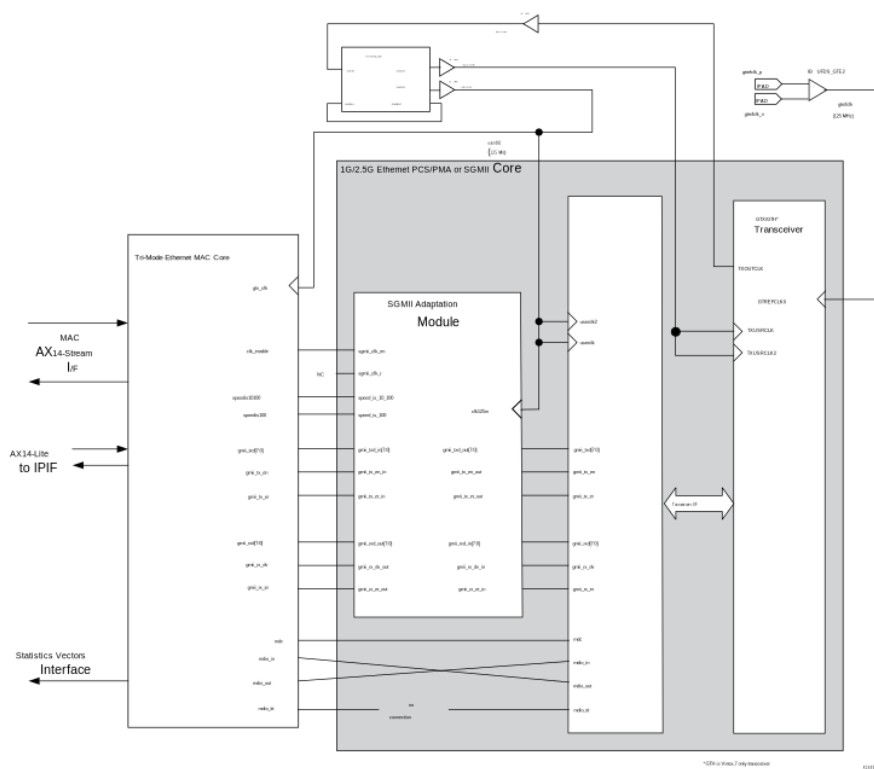
**Figure: Core Using TBI Connected to the TEMAC Core**



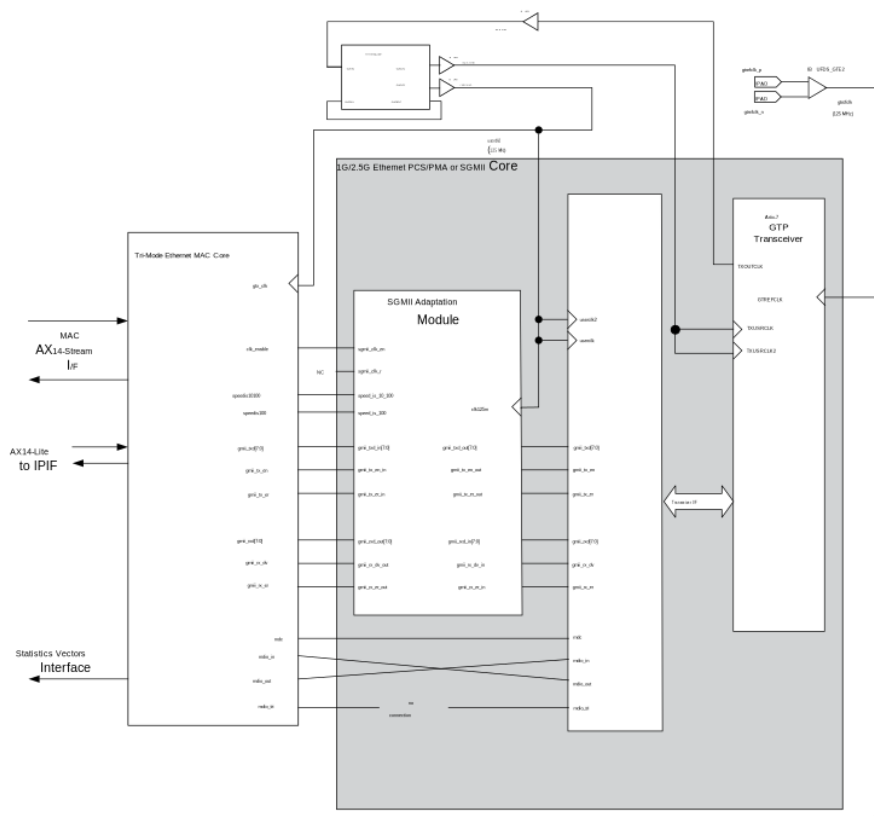
### Transceiver Implementation

The following figure 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. The next figure shows the same interface for Artix 7 devices. The transceiver implementation is similar in 2.5G mode.

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



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



- 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 *Tri-Mode Ethernet MAC LogiCORE IP Product Guide (PG051)*.
- 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 1G/2.5G Ethernet 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 Zynq 7000 Device PS ENET0/1 Using Sync SGMII over LVDS

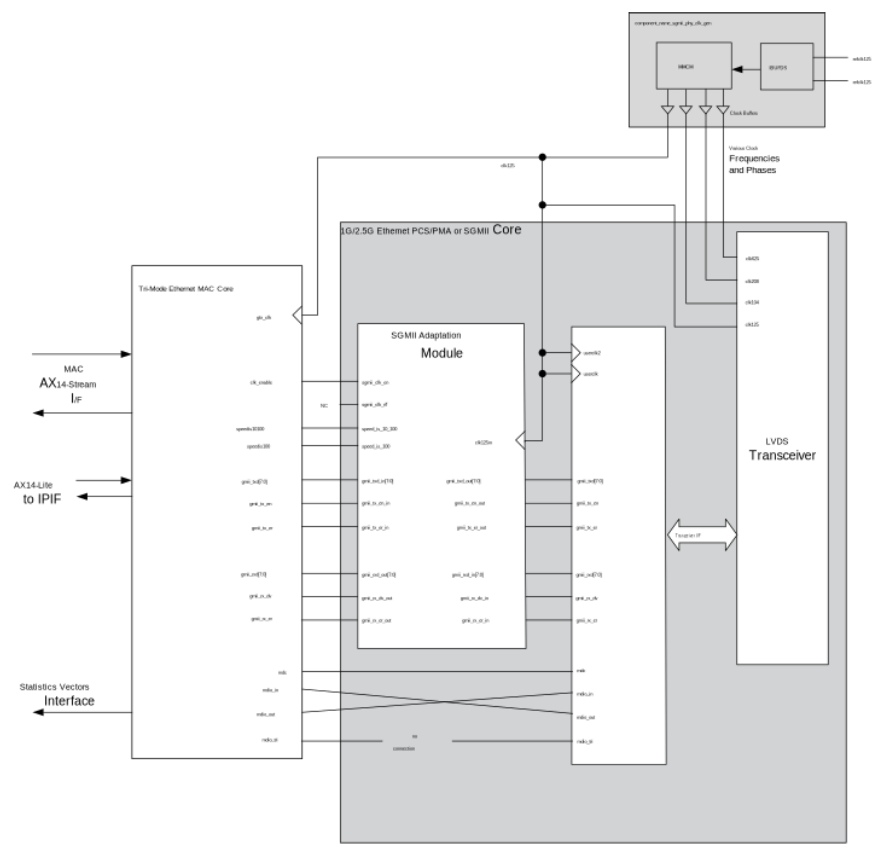
The following figure shows the connections and clock management logic required to interface the core (in Sync SGMII over LVDS) to the TEMAC core. The block level of the Example Design should be taken from the example design and instantiated for connection to the TEMAC core. Connections from a unique TEMAC core to SGMII port are identical and are shown in the following figure. The 2.5G mode is not supported in this case.

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 1G/2.5G Ethernet 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 1G/2.5G Ethernet 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.

The following figure 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 *Tri-Mode Ethernet MAC LogiCORE IP Product Guide (PG051)*.

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



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

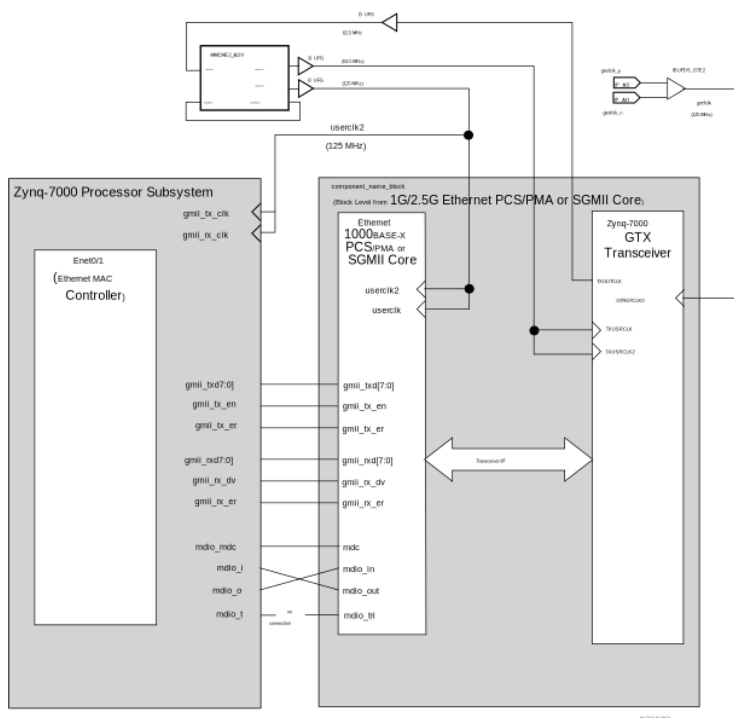
The following figure shows the connections and clock management logic required to interface the core (in 1000BASE-X mode) to the

Zynq 7000 device PS ENET0/1. The 2.5G mode is not supported in this case.

Features of this configuration include:

- Direct internal connections are made between the GMII interfaces between the ENET0/1 and 1G/2.5G Ethernet 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 1G/2.5G Ethernet 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 1G/2.5G Ethernet 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: ENET0/1 Extended to Include 1000BASE-X PCS/PMA Using Device Transceiver**



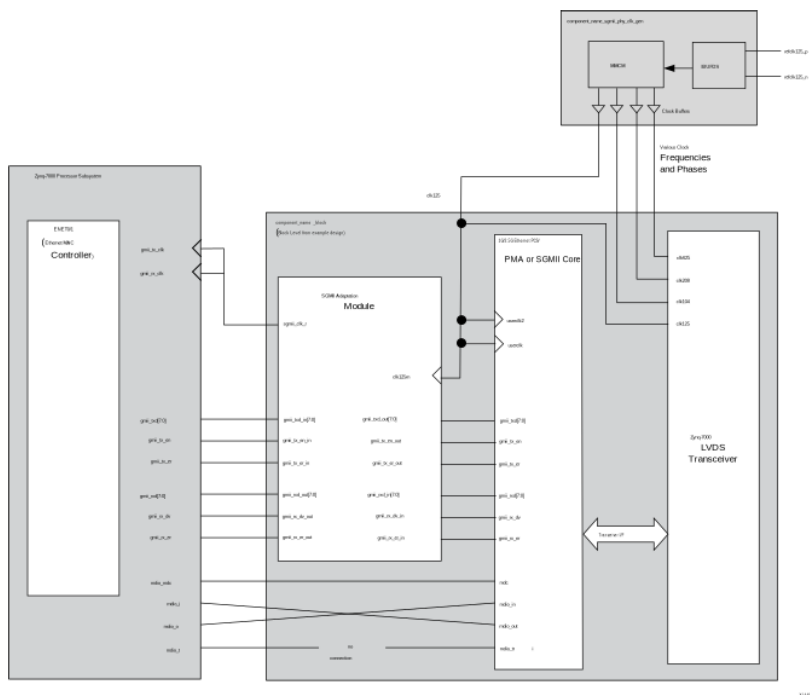
## Integration of the Zynq 7000 Device PS ENET0/1 for Tri-speed SGMII Operation

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

The following figure 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. The 2.5G mode is not supported in this case. 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 1G/2.5G Ethernet 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: Zynq 7000 Device ENET0/1 Extended to Include SGMII Using GTX Transceiver**



Integration of the TEMAC Core Using Sync SGMII over LVDS

The following figure 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 2.5G mode is not supported in this case. 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 1G/2.5G Ethernet 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.

## 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.

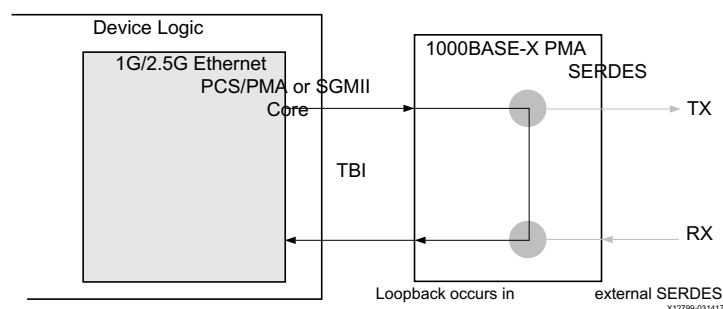
### Loopback

This section details the implementation of the loopback feature. Loopback mode is enabled or disabled by either the [MDIO Management Interface Ports](#) or by the [Configuration and Status Vector Ports](#).

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 the following figure.

**Figure: Loopback Implementation Using the TBI**



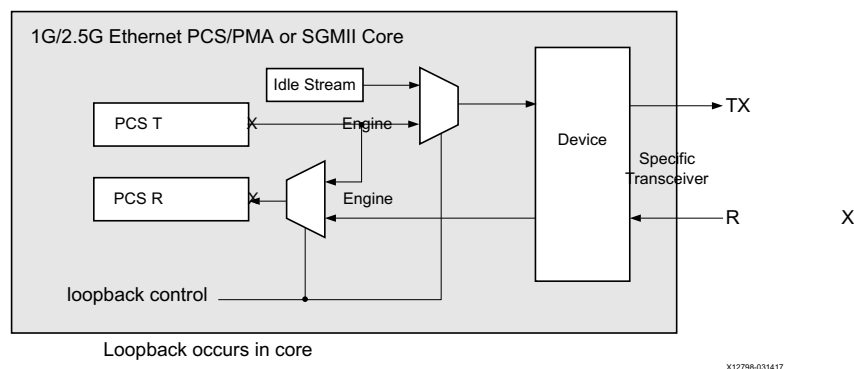
#### Core with Transceiver

The loopback path is implemented in the core as shown in the following figure. 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.

**Note:** Loopback is not supported by the core when RxGmiiClkSrc=RXOUTCLK.

**Figure: Loopback Implementation when Using the Core with Device-Specific Transceivers**



## Design Flow Steps

This section 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 AMD Vivado™ design flows and 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](#))
- *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
- *Vivado Design Suite User Guide: Getting Started* ([UG910](#))
- *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))

## Customizing and Generating the Core

This section includes information about using AMD tools to customize and generate the core in the AMD Vivado™ Design Suite. 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](#)) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console. 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 toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

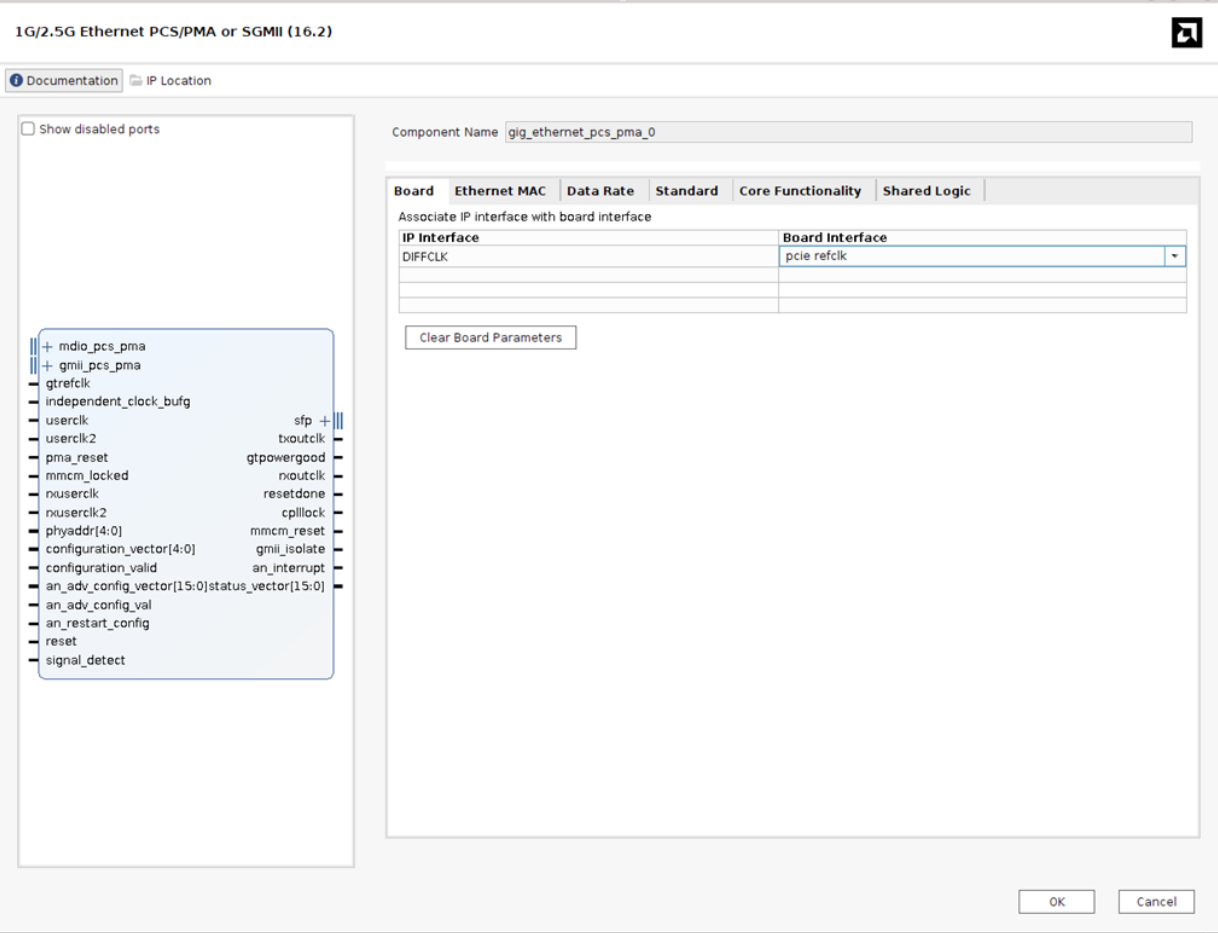
### Component Name

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 ".".

### Board Tab

The following figure shows the option to enable the additional board support flow with the core. This option is available only when the project is created by selecting a board from the list of non-Versal boards available.

**Figure: Board Tab**

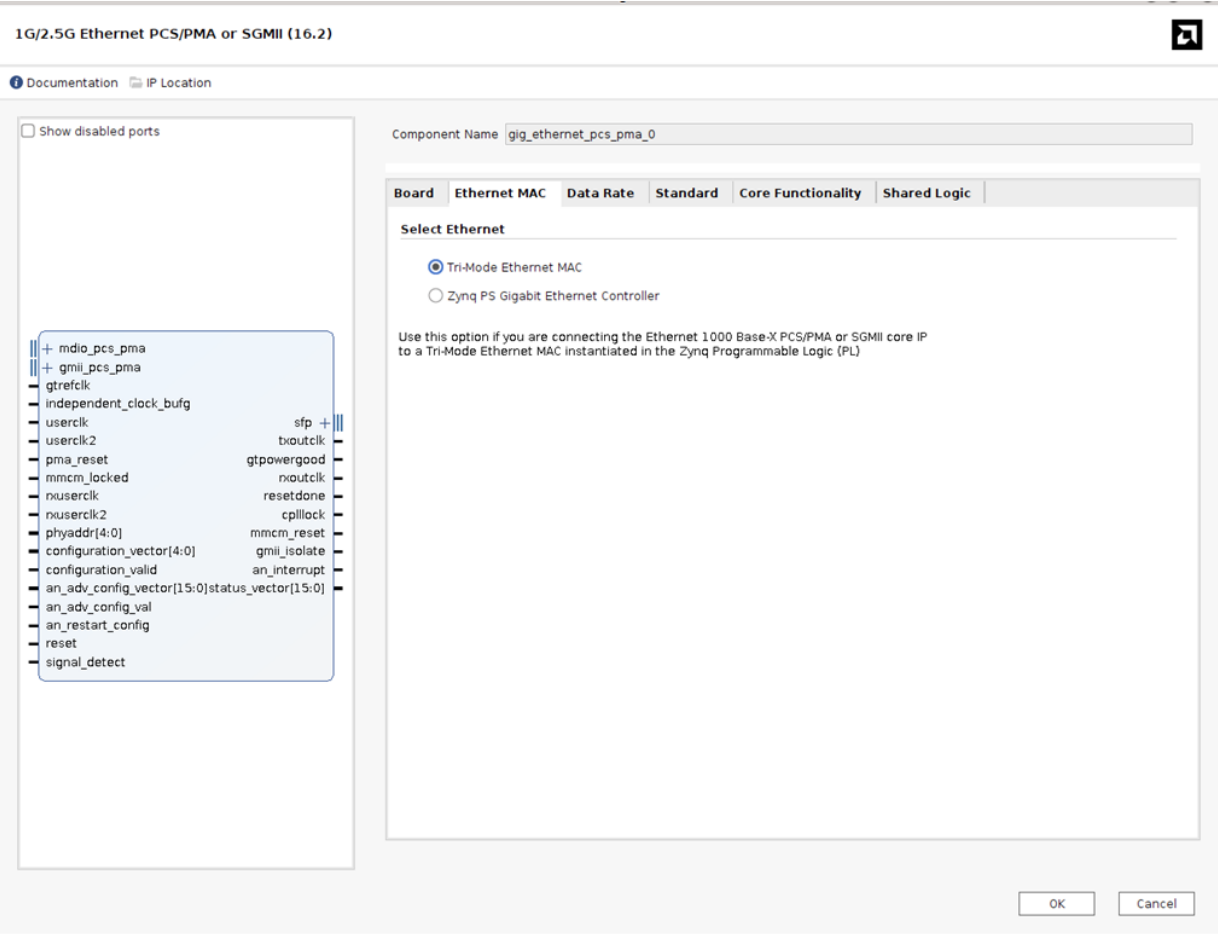


### Ethernet MAC Tab

The following figure shows the Ethernet MAC selection screen. This screen is visible only for AMD Zynq™ and Versal devices.

**Figure: Core Customization Screen for Zynq 7000 Devices - Ethernet MAC Tab**





### Select Ethernet

Select from the following Ethernet MACs:

#### Tri-Mode Ethernet MAC

This option is used if the core is interfaced with the Tri-mode Ethernet MAC instantiated in the device Programmable Logic (PL).

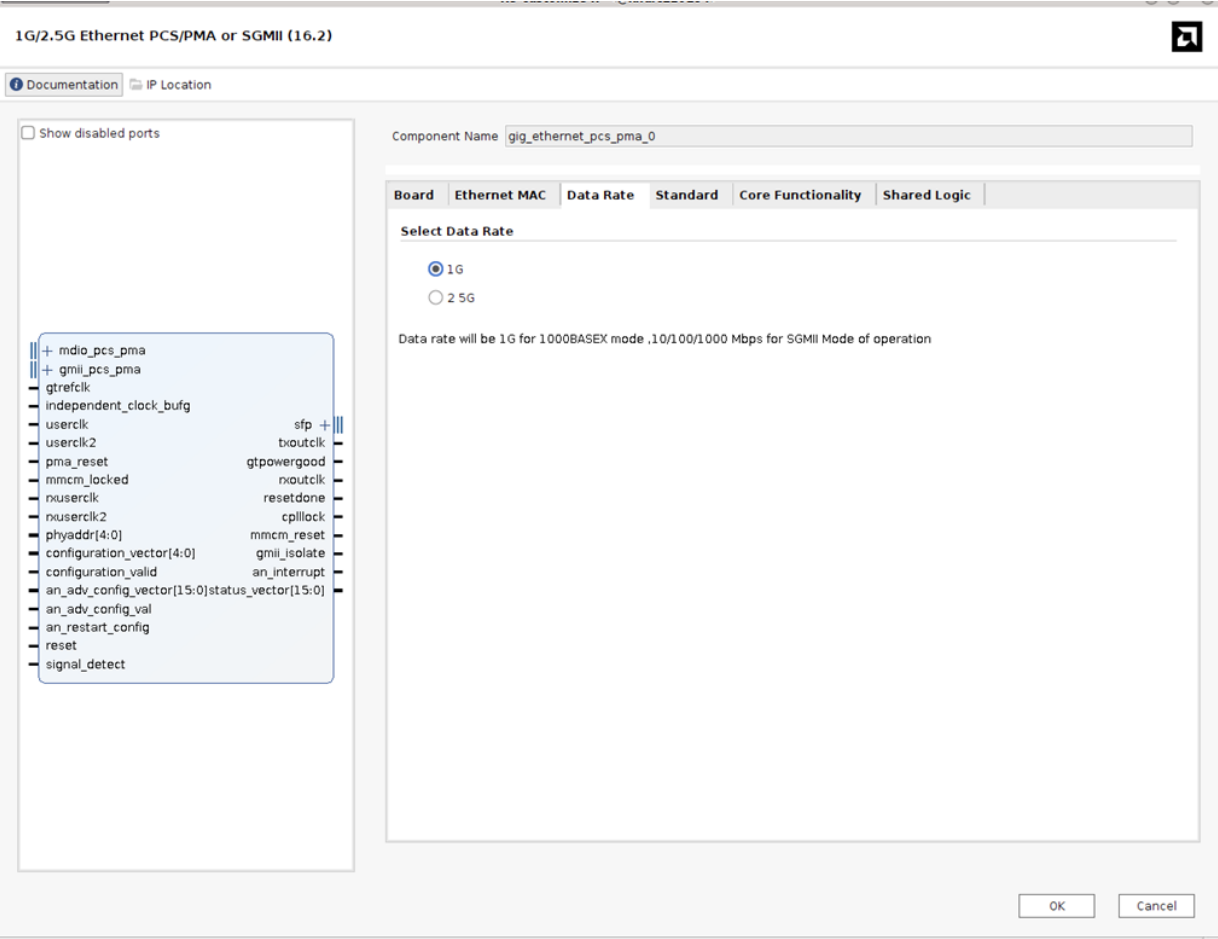
#### Zynq-PS Gigabit Ethernet Controller

This option is used if the core is interfaced with the Ethernet MAC (EMAC) present in the AMD Zynq™ device processor subsystem (PS). The core and EMAC are connected through the EMIO interface. The Same option is available for Versal devices as "Versal CIPS Gigabit Ethernet Controller".

### Data Rate Tab

The following figure shows the Data Rate tab in the core customization screen.

**Figure: Core Customization Screen - Data Rate Tab**



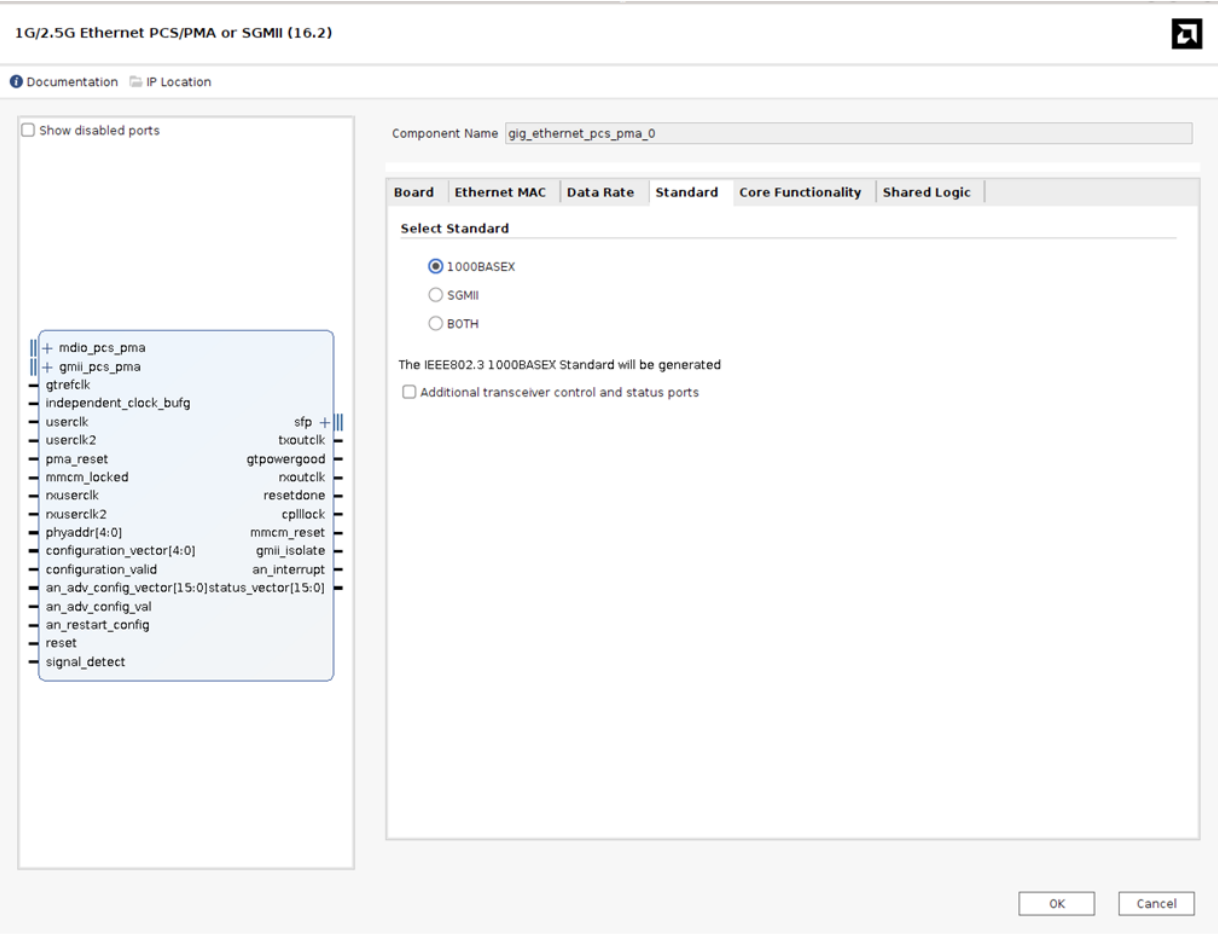
### Select Data Rate

Select the data rate to support 1G maximum speed or 2.5G speed.

### Standard Tab

The following figure shows the Standard tab in the core customization screen.

**Figure: Core Customization Screen - Standard Tab**



### 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.

### Core Functionality Tab

The following figure shows the Core Functionality tab in the core customization screen.

**Figure: Core Customization Screen - Core Functionality Tab**

1G/2.5G Ethernet PCS/PMA or SGMII (16.2)

Documentation IP Location

Show disabled ports

Component Name: `gig_ethernet_pcs_pma_0`

Board Ethernet MAC Data Rate Standard Core Functionality Shared Logic

**Physical Interface**

☒ Device Specific Transceiver  
☐ LVDS Serial

The Physical Interface will be device specific transceiver

**Transceiver options**

Reference Clock Frequency (MHz): 125  
 Transceiver Location: X0Y0  
 DRP Clock Frequency (MHz): 50.0

**Rx Gmii Clk Src**

Receive GMII Clock Source: TXOUTCLK

**Management Options**

☒ MDIO Management Interface  
☐ MDIO Management Interface for external PHY  
☒ Auto Negotiation

OK Cancel

## 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 AMD Versal™ Adaptive SoCs, AMD UltraScale+™ /AMD UltraScale™, Zynq 7000, AMD Virtex™ 7, Kintex 7 and AMD Artix™ 7 devices. For additional information, see [1000BASE-X or 2500BASE-X with Transceivers](#).

### Ten Bit Interface (TBI)

Provides 1000BASE-X, 2500BASE-X, SGMII or 2.5G SGMII functionality with a parallel TBI used to interface to an external Serializer/Deserializer (SerDes). This is available for Kintex 7 devices.

### LVDS Serial

AMD 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. AMD Artix™ 7 and Spartan 7 devices, -2 speed grade or higher, can fully support SGMII using standard LVDS SelectIO™ technology logic resources for AMD UltraScale™ /AMD UltraScale+™ devices. 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. For AMD Versal™ devices, the Advanced IO Wizard is used as a subcore to support the Async LVDS solution. For more information on supported AMD Versal™ devices, refer to [Asynchronous SGMII/1000BASE-X Over LVDS](#). This enables direct connection to external PHY devices without the use of an FPGA transceiver.

## Transceiver Options

### GT TYPE (GT\_Type)

Selects the type of transceiver (GTH/GTY). Applicable only for UltraScale and AMD UltraScale+™ devices with GTH and GTY transceivers. This option is enabled only for AMD UltraScale+™ /AMD UltraScale™ devices. For AMD Versal™ Devices, GTY or GTYP is automatically assigned based on the device.

Valid Values: GTH (Default) and GTY

### Reference Clock Frequency (MHz)

This specifies the GT reference clock rate. Valid values are selected in the drop-down menu.

### Transceiver Location

This specifies the GT Location. Valid values are selected in the drop-down menu.

#### **DRP Clock Frequency**

drpclock and independent clock frequency.

1. If the Transceiver Control and status is enabled, this corresponds to the DRP clock input. It is recommended to connect the independent clock to the same clock frequency.
2. If the Transceiver Control and status is disabled this corresponds to the independent clock.

Valid values:

- 6.25-62.5 MHz (MaxDataRate is 1G)
- 6.25-156.25 MHz (MaxDataRate is 2\_5G)

#### **LVDS Reference Clock Frequency**

This specifies the reference clock frequency for the synchronous SGMII LVDS solution. Valid values are selected through the drop-down menu. Applicable only for AMD UltraScale™ devices.

#### **Advanced IO PLL INPUT Clock**

Specifies the reference clock for the XPLL present in Advanced IO wizard sub-core. Valid values are 125, 156.25, 312.5 and 625. This option is shown instead of “LVDS Reference Clock Frequency” For AMD Versal™ devices.

#### **RX Gmii Clk Src**

##### **Receive GMII Clock Source**

Selects the clock source for the receive path and the GMII RX interface. RxGmiiClkSrc=RXOUTCLK is not supported for the 2.5G data rates. The fabric elastic buffer is not required when RXOUTCLK is selected because the RX datapath is synchronous to the recovered clock. Therefore, the SGMII Capabilities tab for SGMII speed selection is not required.

Valid values: TXOUTCLK (Default) and RXOUTCLK

#### **Management Options**

##### **MDIO Management Interface**

Select this option to include the MDIO management Interface to access the PCS Configuration registers. An additional configuration vector interface is provided to write into management Registers 0 and 4 (see [Configuration and Status Vector Ports](#)).

##### **MDIO Management Interface for external PHY**

Select this option to include an additional MDIO management interface to access the MDIO registers of the external PHY. This option is provided only when the MDIO management Interface is enabled.

##### **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.

---

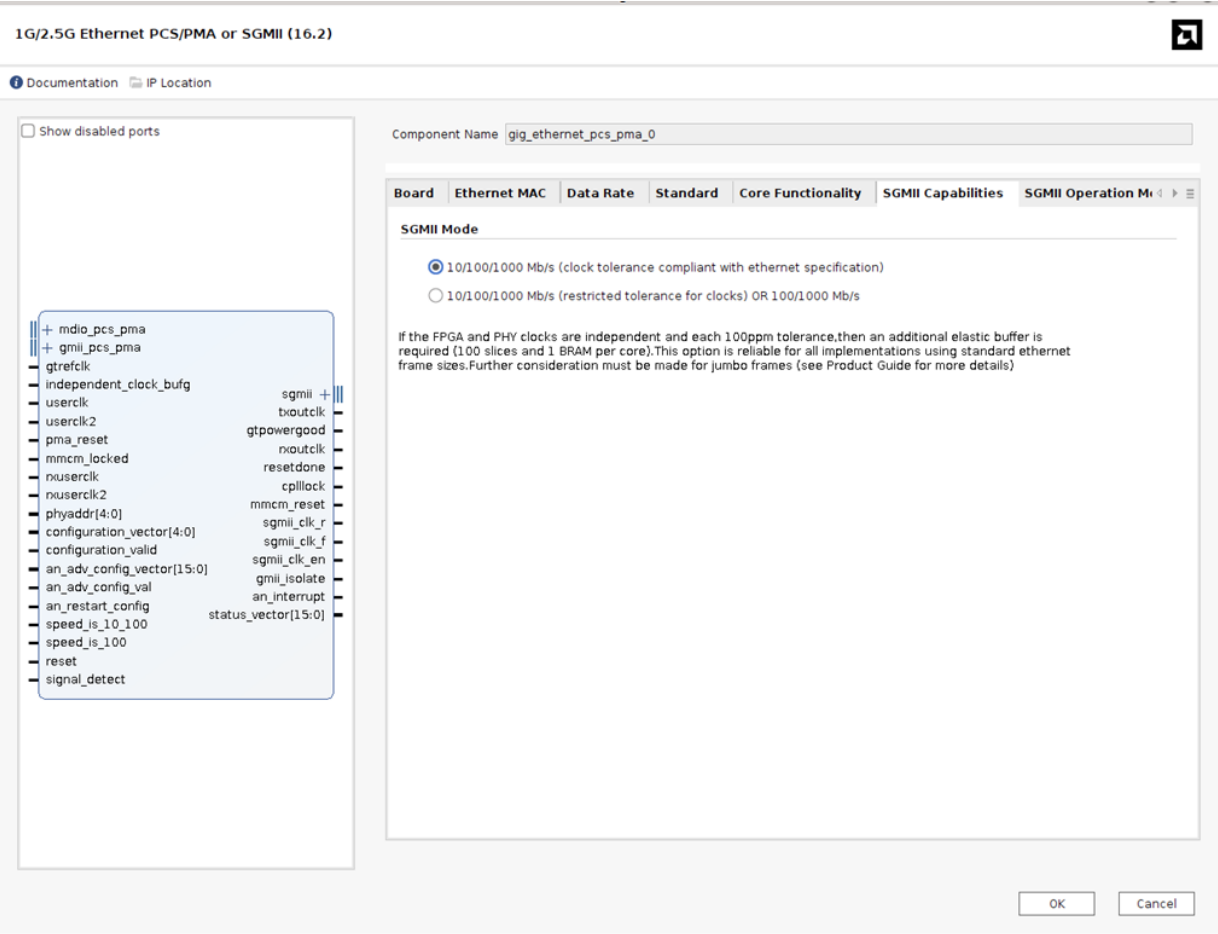
**!! Important:** For details on placement options for asynchronous LVDS see [Lane Placement Parameters](#).

---

#### **SGMII Capabilities Tab**

The SGMII Capabilities tab is only displayed if either SGMII or BOTH is selected in the Standard tab and only if the device-specific transceiver is selected as the Physical Interface in the Core Functionality tab.

#### **Figure: Core Customization Screen - SGMII Capabilities Tab**

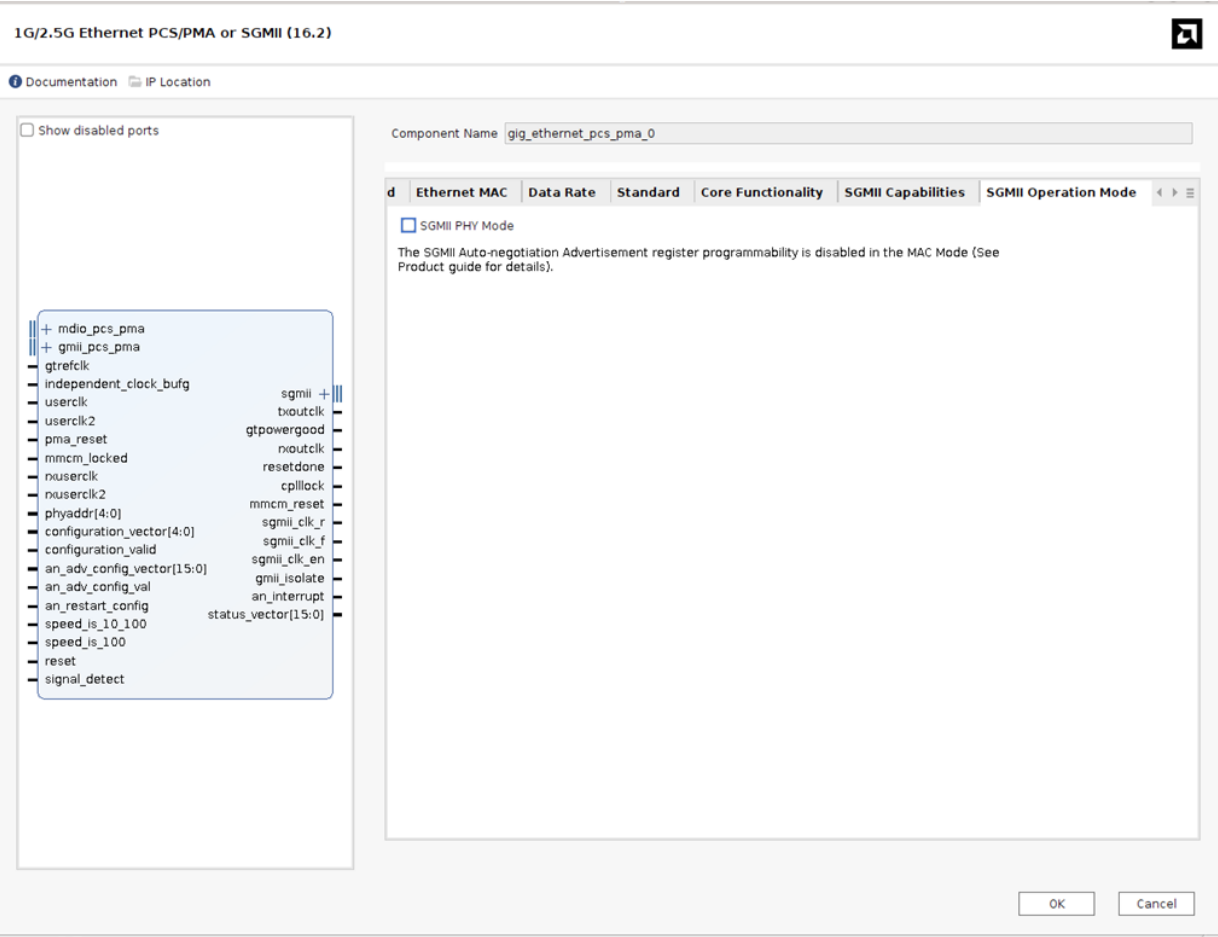


This screen lets you select the receive elastic buffer type to be used with the core. Before selecting this option, see [Receive Elastic Buffer](#).

SGMII Operation Mode Tab

The SGMII Operation Mode tab is only displayed if either SGMII or BOTH is selected in the Standard tab.

Figure: Core Customization Screen - SGMII Operation Mode Tab



This screen lets you select the core to operate in the PHY mode or Media Access Controller (MAC) mode.

Shared Logic Tab

This tab is available for non-Versal designs. The following figure shows the shared logic placement options available in the IP core.

Figure: Shared Logic Placement Options

1G/2.5G Ethernet PCS/PMA or SGMII (16.2)

Documentation
IP Location

Show disabled ports

+ mdio\_pcs\_pma
+ gmii\_pcs\_pma
gtrefclk
independent\_clock\_bufg
userclk
userclk2
pma\_reset
mmcm\_locked
nuserclk
nuserclk2
phyaddr[4:0]
configuration\_vector[4:0]
configuration\_valid
an\_adv\_config\_vector[15:0]
an\_adv\_config\_val
an\_restart\_config
speed\_is\_10\_100
speed\_is\_100
reset
signal\_detect

sgmii
txoutclk
txoutclk
cpillock
mmcm\_reset
sgmii\_clk\_r
sgmii\_clk\_f
sgmii\_clk\_en
gmii\_isolate
an\_interrupt
status\_vector[15:0]

Component Name
gig\_ethernet\_pcs\_pma\_0

Data Rate
Standard
Core Functionality
SGMII Capabilities
SGMII Operation Mode
Shared Logic

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

☐ GT 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.

Core with Shared Logic

Shared Logic

OR

Core without Shared Logic

Example Design

Shared Logic

## Shared Logic

Determines whether some shared clocking logic is included as part of the core or as part of the example design.

## GT in Example Design

This option is available only if Include Shared logic in Example Design is selected. This moves the transceiver from the core to the Support level instance. When this option is selected generation of additional transceiver control and status ports is disabled. This option is available only for UltraScale architecture designs.

**Note:** For Versal, the GT is always in the example design.

## 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 2.5G data rates select the TX and RX line rates to be 3.125 Gbps.

### For SGMII with fabric elastic buffer/ Dynamic Switching modes

Select Protocol as gigabit ethernet noCC and generate the GT Wizard IP. For 2.5G data rates select the TX and RX line rates to be 3.125 Gbps.

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]

136 of 180

2024-03-20, 17:11



## Output Generation

The 1G/2.5G Ethernet 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 created 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 1G/2.5G Ethernet PCS/PMA or SGMII solution are described in the following subsections.

### Examples

Includes all source required to be able to open and implement the IP example design project, that is, the 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. 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 core 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. When `EXAMPLE_SIMULATION` is set, the core also pre-loads the 3 ms counter for the 7 series transceiver in startup FSM with a smaller value to decrease the simulation time.

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:** In the Asynchronous SGMII/1000BASE-X over LVDS solution, `EXAMPLE_SIMULATION` should be 1 to simulate the design in post synthesis and post implementation simulations. This is because the RIU register that is used in the calibration sequence in the `clock_reset` module is not supported in simulation and always returns a 0 value when read. However `EXAMPLE_SIMULATION` should be set to 0 when the design is implemented in a device.

---

🔗 **Note:** The `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 use the following command before the generation of the 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.

---

For more detail, see the *Vivado Design Suite User Guide: Designing with IP* ([UG896](#)).

## User Parameters

The following table shows the relationship between the fields in the AMD Vivado™ IDE and the user parameters (which can be viewed in the Tcl Console).

**Table: Vivado IDE Parameter to User Parameter Relationship**

AMD Vivado™ IDE Parameter/Value	User Parameter/Value	Default Value
MDIO	ETHERNET_BOARD_INTERFACE	Custom
DIFFCLK	DIFFCLK_BOARD_INTERFACE	Custom
MDIO	MDIO_BOARD_INTERFACE	Custom
Standard	Standard	1000BASEX
Data Rate	MaxDataRate	1G
Physical Interface	Physical_Interface	Transceiver
Rx Gmii Clk Src	RxGmiiClkSrc	TXOUTCLK
MDIO Management Interface	Management_Interface	true

AMD Vivado™ IDE Parameter/Value	User Parameter/Value	Default Value
MDIO Management Interface for external PHY	Ext_Management_Interface	false
Auto Negotiation	Auto_Negotiation	true
SGMII Mode	SGMII_Mode	10_100_1000
SGMII PHY Mode	SGMII_PHY_Mode	false
Select Ethernet	EMAC_IF_TEMAC	TEMAC
Shared Logic	SupportLevel	Include_Shared_Logic_in_Example_Design
GT in Example Design	GTInEx	false
Additional transceiver control and status ports	TransceiverControl	false
Reference Clock Frequency (MHz)	RefClkRate	125
LVDS Reference Clock	LvdsRefClk	125
DRP Clock Frequency (MHz)	DrpClkRate	50.0
Number of Lanes	NumOfLanes	1
Tx In Upper Nibble	Tx_In_Upper_Nibble	1
TxLane0 Placement	TxLane0_Placement	DIFF_PAIR_0
RxLane0 Placement	RxLane0_Placement	DIFF_PAIR_0
TxLane1 Placement	TxLane1_Placement	DIFF_PAIR_1
RxLane1 Placement	RxLane1_Placement	DIFF_PAIR_1
EnableAsyncSGMII	EnableAsyncSGMII	FALSE
Transceiver Location	GT_Location	X0Y0
GT Type	GT_Type	GTH

## Constraining the Core

### Required Constraints

The 1G/2.5G Ethernet 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 [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 1G/2.5G Ethernet PCS/PMA or SGMII solution is instantiated.

### Device, Package, and Speed Grade Selections

The core can be implemented in Versal, UltraScale+, Zynq 7000 SoC, Virtex 7, Kintex 7, Artix 7, and Spartan-7 devices. The modes supported for specific devices are described in [Resource Utilization](#).

### Clock Frequencies

The 1G/2.5G Ethernet 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: Clock Frequencies

Clock Name	Frequency Requirement
025_0MHz if serial transceiver is used	025_0MHz or user selectable value for UltraScale+/UltraScale devices.
025_0MHz serial transceiver is used	025_0MHz 125 MHz depending on serial transceiver used for 1G data rate. For 2.5G data rates for 7 series and Zynq devices this clock frequency is 125 MHz, For UltraScale+/UltraScale devices the frequency is 312.5 MHz.
025_0MHz if serial transceiver is used	025_0MHz 156.25 MHz for 1G line rate. For 2.5G data rates this clock frequency is 156.25 MHz.
025_0MHz if serial transceiver is used	025_0MHz 312.5 MHz for 1G data rates. For 2.5G data rates this clock frequency is 312.5 MHz.
025_0MHz SGMII Mode	025_0MHz 12.5 MHz or 125 MHz for 1G data rates. For 2.5 data rates 312.5 MHz is applicable because the core supports only the 2.5 Gbps data rate.
025_0MHz serial transceiver is used.	025_0MHz Recovered clock by transceiver from the input serial data. for 1G data rate this frequency is 62.5 MHz. For 2.5G rate this is 156.25 MHz.
025_0MHz transceiver is used.	025_0MHz The clock is a looped back version of rxoutclk after passing through a BUFG. When the fabric elastic buffer is not used (1000BASE-X or 100_1000 SGMII modes) this is not used within the core. The BUFG can be replaced in clocking logic with a BUFR or BUFG. For 7 series devices, when RxGmiiClkSrc=RXOUTCLK, rxuserclk2 is twice the RXOUTCLK rate.
025_0MHz in TBI Mode	025_0MHz
025_0MHz in TBI Mode	025_0MHz
025_0MHz in TBI Mode	025_0MHz
025_0MHz in LVDS	025_0MHz

Characterization	Frequency Requirement
Mode	
200MHz in LVDS Mode	
200MHz in LVDS Mode	

## 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 AMD 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 AMD TEMAC core in the FPGA, which means that no IP standard/placement is required for that interface.

## Simulation

For comprehensive information about AMD Vivado™ simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).


---

**!! Important:** For cores targeting 7 series or Zynq 7000 devices, UNIFAST libraries are not supported. AMD IP is tested and qualified with UNISIM libraries only.


---

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> ]`

---

 **Note:** 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).

# Example Design

The example designs provided with the core are described in detail in this chapter. For information about the Demonstration Test Bench, see [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]
```

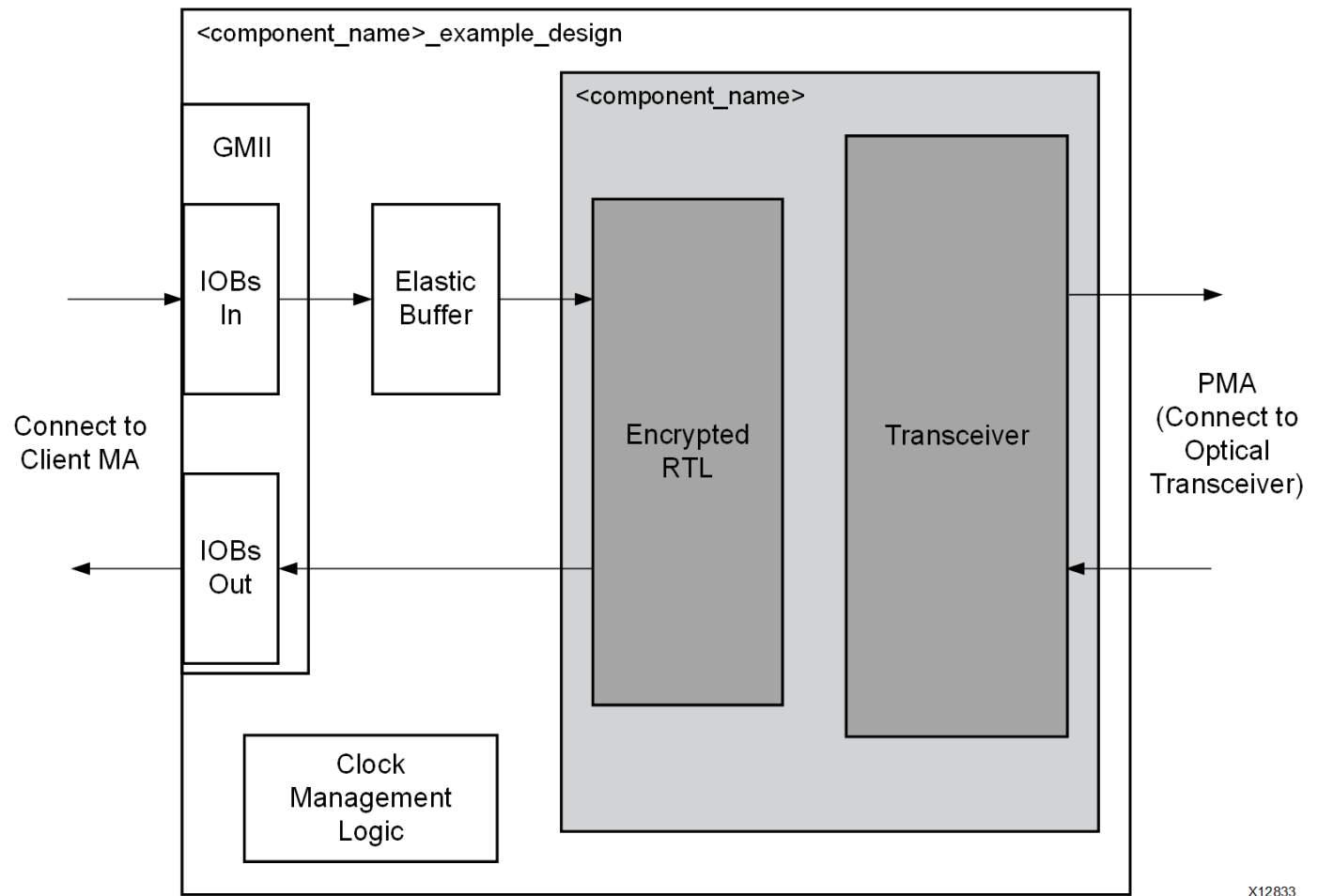
The following files describe the block level example design for the core:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/example_design/<component_name>.v[hd]
```

## 1000BASE-X or 2500BASE-X with Transceiver Example Design

shows the example design for the core using a device-specific transceiver (AMD UltraScale+™ /AMD UltraScale™ architecture, 7 series or AMD Zynq™ 7000).

**Figure: 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]
```

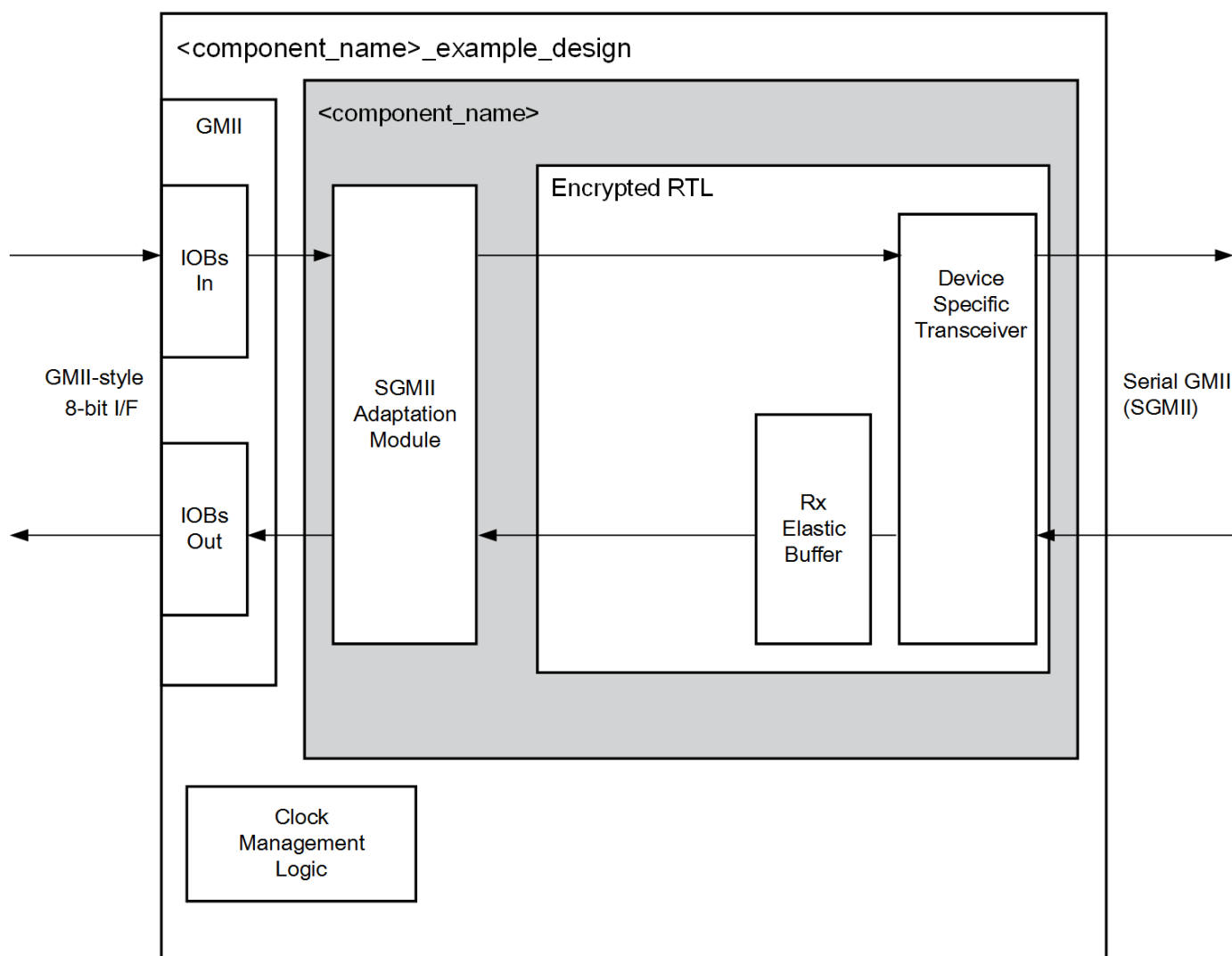
When the GMII is used externally (as in this example design), the GMII transmit signals (inputs to the core from a remote MAC at the other end of the interface) are synchronous to a clock that is likely to be derived from a different clock source to the core. For this reason, GMII transmit signals must 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

The following figure shows an example design for top level HDL for the core in SGMII (or dynamic switching) mode using a device-specific transceiver / architecture, , or devices). Dynamic switching is not supported in 2.5G mode. The 2.5G SGMII mode structure is the same as that of 1G SGMII.

**Figure: Example Design HDL in 1G or 2.5G SGMII Mode Using a Device-Specific Transceiver**



X12867

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

# Synchronous SGMII over LVDS Example Design (Applicable for Non-Versal Devices)

Figure 1 shows the HDL example design that is provided for the SGMII overZynq 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 2.5G mode is not supported in this case. The core netlist in this implementation is identical to that of [1G/2.5G Ethernet PCS/PMA or SGMII Using a Device-Specific Transceiver](#).

## Top Level

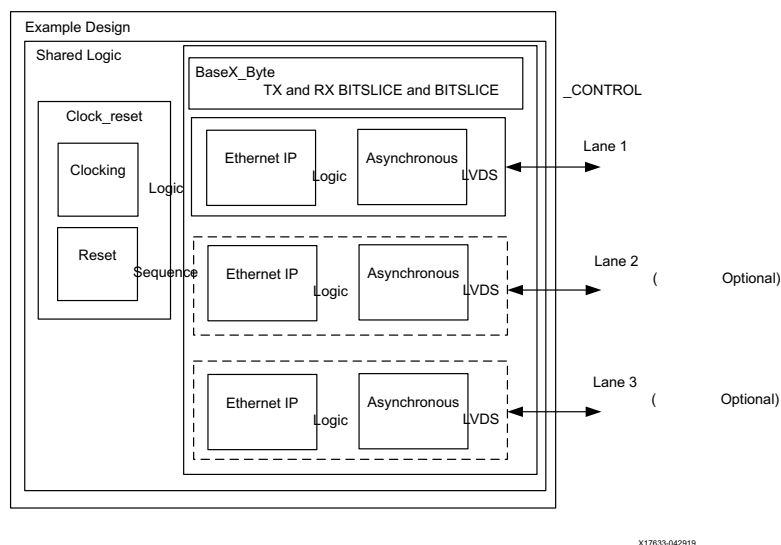
The example design HDL top level performs the following:

- Instantiates the block level HDL
- Instantiates shared logic if shared logic in the example design is selected (see [Shared Logic](#) for more information)
- Adds 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.

# Asynchronous 1000BASE-X/SGMII over LVDS (Applicable for Non-Versal Devices)

The following figure shows the HDL example design that is provided for the Asynchronous 1000BASE-X/SGMII over UltraScale and UltraScale+ device LVDS implementation. The top level of the example design creates a specific example that can be simulated, synthesized and implemented. The 2.5G mode is not supported in this case. The core netlist in this implementation is identical to that of [1G/2.5G Ethernet PCS/PMA or SGMII Using a Device-Specific Transceiver](#).

**Figure: Example Design for Asynchronous 1000BASE-X/SGMII over LVDS**



## Top Level

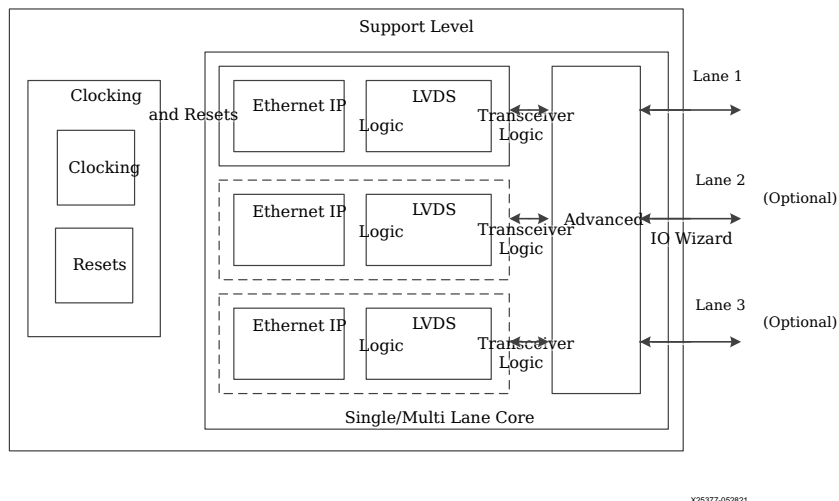
The example design HDL top level performs the following:

- Instantiates the 1-3 lanes block level HDL
- Instantiates shared logic if shared logic in the example design is selected (see [Shared Logic](#) for more information)

# Asynchronous 1000BaseX/SGMII over LVDS Example Design (Versal Devices)

The following figure shows The Example design provided for Asynchronous 1000BaseX/SGMII over Versal device LVDS Implementation. Logic to generate Input clocks (125 MHz Core Clock, PLL input clock and CTRL clock) for the core and the RIU reset state machine are provided in the example design.

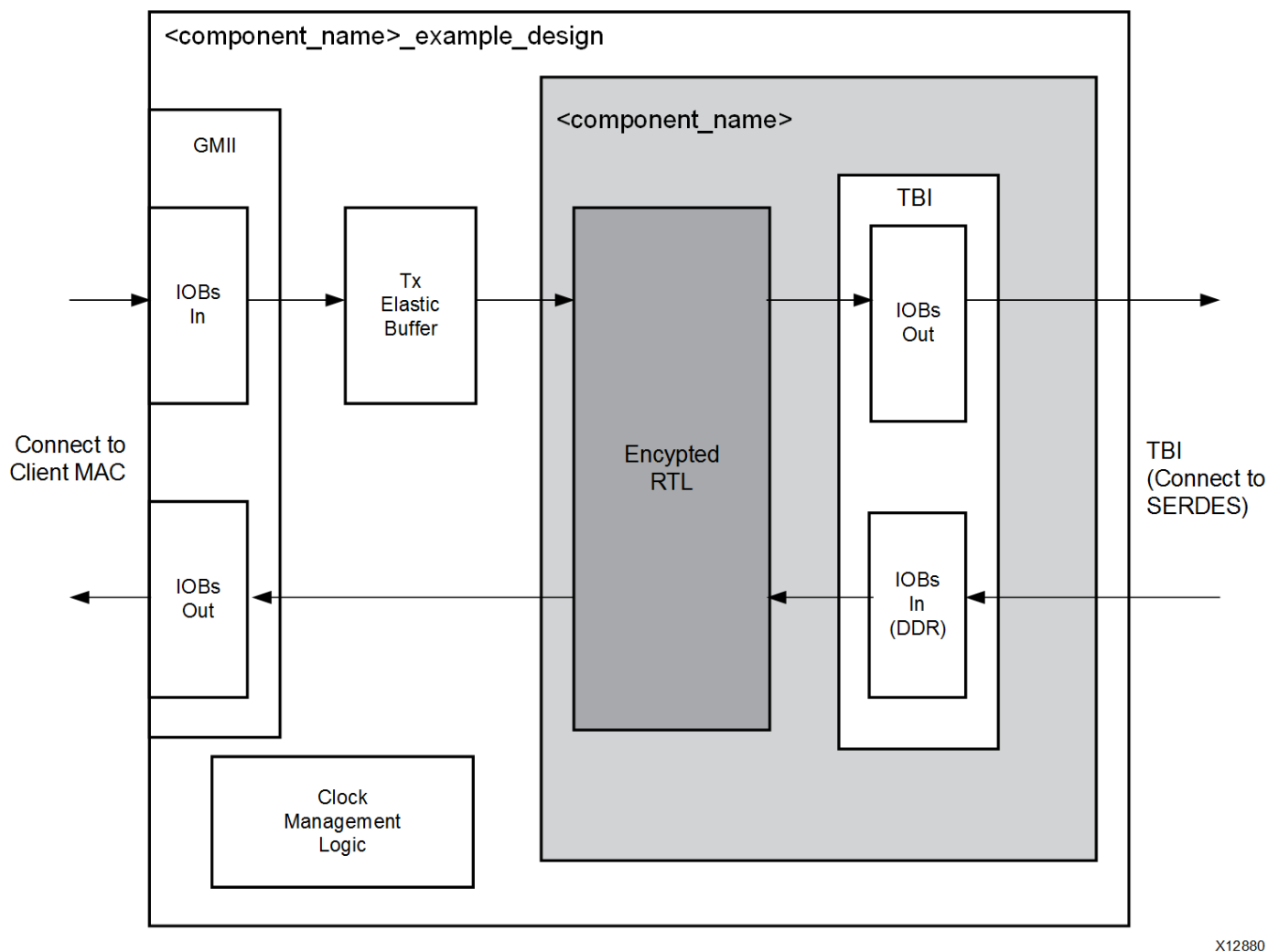
**Figure: Example design for Asynchronous SGMII/1000BaseX over LVDS (Versal)**



## 1000BASE-X with TBI Example Design

The following figure shows the example design for a top level HDL with a 10-bit interface (TBI). The 2.5G mode is not supported in this case.

**Figure: Example Design HDL for the Core 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]
```

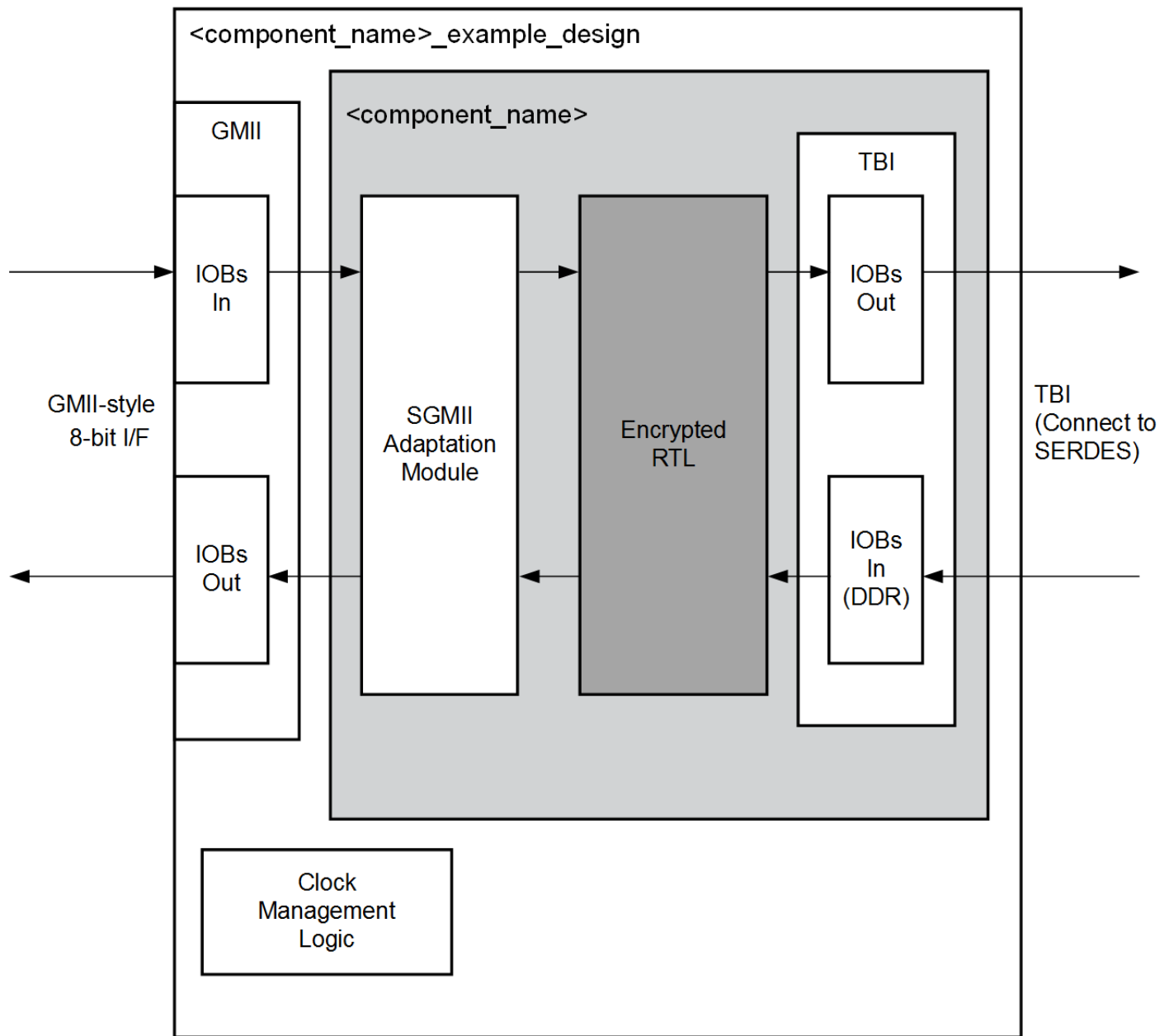
When the GMII is used externally (as in this example design), the GMII transmit signals (inputs to the core from a remote Ethernet MAC at the other end of the interface) are synchronous to a clock, which is likely to be derived from a different clock source to the core. For this reason, GMII transmit 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

The following figure shows an example design for the top level HDL for the core in SGMII (or dynamic switching) mode with the TBI. The 2.5G mode is not supported in this case.

**Figure: Example Design HDL for the Core in SGMII Mode with TBI**



X12869

## 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

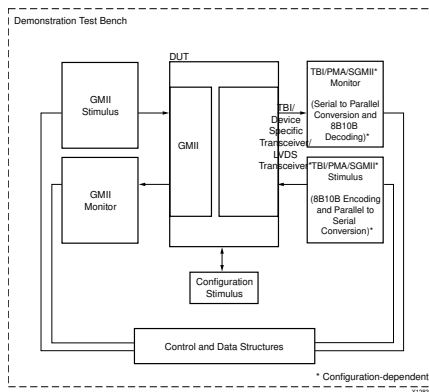
The example design HDL top level connects the GMII of the block level to external IOBs. This allows the functionality of the core to be demonstrated using a simulation package.

## Test Bench

This chapter contains information about the demonstration test bench provided in the AMD Vivado™ Design Suite.

The following figure 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.

**Figure: 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 through `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 Mbps, 100 Mbps, 1 Gbps, or 2.5 Gbps. By default, the demonstration test bench is configured to operate at 1 Gbps. 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. The speed bits are not applicable for 2.5G SGMII.

1 Gbps Operation

```
set speed_is_10_100 to logic 0
```

100 Mbps Operation

```
set speed_is_10_100 to logic 1
```

```
set speed_is_100 to logic 1
```

10 Mbps 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

AMD UltraScale+™, AMD UltraScale™, AMD Zynq™ 7000, and 7 series device designs incorporating a device-specific transceiver require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. For IP simulation, a mixed simulation license is required.

## Hardware Testing

The core has been tested on many hardware test platforms at AMD 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 AMD Tri-Mode Ethernet MAC core, which follows the architecture shown in [Ethernet 1000BASE-X or 2500BASE-X](#). 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 Arm® 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 AMD 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 [GMII to SGMII Bridge](#) and also includes the Arm based processor test platform described previously.

## Upgrading

This appendix contains information about migrating a design from ISE® to the AMD Vivado™ Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the AMD 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 AMD Vivado™ Design Suite, see the *ISE to Vivado Design Suite Migration Guide*([UG911](#)).

## 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 AMD Vivado™ Design Suite.

**Table: Ports Added in V16.1**

In/Out	Port Name	Description	What to do
Out	gtpowergood	See <i>UltraScale FPGAs Transceivers Wizard LogiCORE IP Product Guide</i> ( <a href="#">PG182</a> ) for AMD UltraScale™ and AMD UltraScale+™ usage guidelines of the signal.	Port is useful in AMD UltraScale device families when using 'gtrefclk' for other applications.

**Table: Parameters Added in V16.0**

Generic Name	Applicability	Description	Values
<del>Asynchronous</del> Asynchronous	Asynchronous 1000BASE-X/SGMII mode only	This indicates whether to instantiate BITSlice0 if BITSlice0 of RX_NIBBLE is unused.	True False
<del>Asynchronous</del> Asynchronous	Asynchronous 1000BASE-X/SGMII mode only	This is an indication to the core if the BITSlice0 of RX_NIBBLE is used for the clock.	True False
<del>Asynchronous</del> Asynchronous	Asynchronous 1000BASE-X/SGMII mode only	This is applicable only when SGMII is selected as physical interface. Setting TRUE enables options for setting reference clock frequencies.	Sync Async

**Table: Ports Added in V16.0**

In/Out	Port Name	Description	What to do
In	tx_dly_rdy_n <sup>1</sup>	TX delay ready from adjacent BYTE_GROUPS	For multiple core instantiations connect to the

In/Out	Port Name	Description	What to do
In	rx_dly_rdy_n <sup>1</sup>	RX delay ready from adjacent BYTE_GROUPS	appropriate port in the instance without Shared Logic in the Core.
In	tx_vtc_rdy_n <sup>1</sup>	TX VTC ready from adjacent BYTE_GROUPS	
In	rx_vtc_rdy_n <sup>1</sup>	RX VTC ready from adjacent BYTE_GROUPS	
1. n=1, 2, 3.			

### Ports Added from v15.2 to v16.0

The following table does not list the ports separately for multi-lane asynchronous SGMII/BASE-X over LVDS prefixed with `_[lane_number]` . The definition of such signals is the same as the older signal definition except for the prefixed lane number in the signal name.

**Table: Ports Added in V16.0**

In/Out	Port Name	Description	What to do
In	phyaddr[4:0]	PHY address for the core	This has been converted from a parameter in earlier releases to a port.
In	dummy_port_in	Applicable only for Asynchronous 1000BASE-X/SGMII over LVDS mode. Enabled based on placement options for the IP.	Bring this signal to the top level and LOC this port to BITLISCE0 location of the RX_NIBBLE.
Out	tx_dly_rdy	Delay ready indication from TX IO logic	For multiple core instantiations connect this output from the instance without shared logic to the tx_dly_rdy_n/rx_dly_rdy_n port of the shared logic in the core.
Out	rx_dly_rdy	Delay ready indication from RX IO logic	
Out	tx_vtc_rdy	VTC ready indication from TX IO logic	For multiple core instantiations connect this output from the instance without shared logic to the tx_vtc_rdy_n/rx_vtc_rdy_n port of the shared logic in the core.
Out	rx_vtc_rdy	VTC ready indication from RX IO logic	
Out	tx_logic_reset <sup>1</sup>	Reset for TX fabric logic	Can be kept open for the Ethernet application because enabling of the downstream logic can be handled by the core sync_status.
Out	rx_logic_reset <sup>1</sup>	Reset for RX fabric logic	
Out	rx_locked <sup>1</sup>	RX PLL locked indication	Can be kept open.
Out	tx_locked <sup>1</sup>	TX PLL locked indication	Can be kept open.
Out	tx_pll_clkout_phy_en		Port no longer available.
Out	rx_pll_clkout_phy_en		Port no longer available.
Out	tx_bsc_rst_out <sup>1</sup>	TX BITSlice control reset	For multiple core instantiations connect to the appropriate port in the instance without shared logic in the core.
Out	rx_bsc_rst_out <sup>1</sup>	RX BITSlice control reset	
Out	rx_rst_dly_out <sup>1</sup>	Reset to RX delay lines	
Out	tx_rst_dly_out <sup>1</sup>	Reset to TX delay lines	

In/Out	Port Name	Description	What to do
Out	tx_bsc_en_vtc_out <sup>1</sup>	Enable VTC indication to TX BITSlice control	
Out	rx_bsc_en_vtc_out <sup>1</sup>	Enable VTC indication to RX BITSlice control	
Out	tx_bs_en_vtc_out <sup>1</sup>	Enable VTC indication to TX BITSlice	
Out	rx_bs_en_vtc_out <sup>1</sup>	Enable VTC indication to RX BITSlice	
Out	riu_clk_out <sup>1</sup>	RIU clock from PLL	Can be kept open. For multiple core instantiations connect to the appropriate port in the instance without shared logic in the core.
Out	riu_addr_out[5:0] <sup>1</sup>	RIU address	For multiple core instantiations connect to the appropriate port in the instance without shared logic in the core.
Out	riu_wr_data_out[15:0] <sup>1</sup>	RIU write data	
Out	riu_wr_en_out <sup>1</sup>	RIU write enable	
Out	riu_nibble_sel_out[1:0] <sup>1</sup>	RIU Nibble select	
In	riu_rddata_n[15:0] <sup>1</sup> where N=1,2,3	RIU read data from other BYTE_GROUPS of the bank	
In	riu_valid_n <sup>1</sup> Where N=1,2,3	RIU valid from other BYTE_GROUPS of the bank	
In	riu_prsnt_n <sup>1</sup> Where N =1,2,3	RIU present indication from other BYTE_GROUPS of the bank	
Out	rx_btval_n[8:0] <sup>1</sup> Where n=1,2,3	Calibration output from reset sequence. This is the count value of DELAY corresponding to 800 ps.	
Out	clk_rst_debug_out[7:0]		Port no longer available.
Out	tx_pll_clk_out <sup>1</sup>	TX PLL clock to RX IO logic	For multiple core instantiations connect to the appropriate port without shared logic in core.
Out	rx_pll_clk_out <sup>1</sup>	RX PLL clock to RX IO logic	
Out	tx_rdclock_out <sup>1</sup>	Read clock from TX_BITSlice FIFO.	
In	rx_btval[8:0] <sup>2</sup>	Calibration output from reset sequence. This is the count value of DELAY corresponding to 800 ps.	For multiple core instantiations connect to the appropriate port from the instance with shared logic in the core.
In	tx_bsc_rst <sup>2</sup>	TX_BITSlice_CONTROL reset	
In	rx_bsc_rst <sup>2</sup>	RX_BITSlice_CONTROL reset	
In	tx_bs_rst <sup>2</sup>	TX_BITSlice reset	
In	rx_bs_rst <sup>2</sup>	RX_BITSlice reset	
In	tx_rst_dly <sup>2</sup>	Reset to TX delay lines	
In	rx_rst_dly <sup>2</sup>	Reset to RX delay lines	
In	tx_bsc_en_vtc <sup>2</sup>	Enable VTC to TX BITSlice control	

In/Out	Port Name	Description	What to do
In	rx_bsc_en_vtc <sup>2</sup>	Enable VTC to RX BITSlice control	
In	tx_bs_en_vtc <sup>2</sup>	Enable VTC to TX BITSlice	
In	rx_bs_en_vtc <sup>2</sup>	Enable VTC to RX BITSlice	
In	riu_clk <sup>2</sup>	RIU clock	
In	riu_addr[5:0] <sup>2</sup>	RIU address	
In	riu_wr_data[15:0] <sup>2</sup>	RIU write data	
In	riu_wr_en <sup>2</sup>	RIU write enable	
In	riu_nibble_sel[1:0] <sup>2</sup>	RIU Nibble select	
Out	riu_prsnt	RIU present	
Out	riu_valid <sup>2</sup>	RIU Valid	
Out	riu_rd_data[15:0] <sup>2</sup>	RIU Read data	
In	tx_pll_clk <sup>2</sup>	TX PLL clock to IO logic	
In	rx_pll_clk <sup>2</sup>	TX PLL clock to IO logic	
In	tx_rdcclk <sup>2</sup>	Read clock from TX_BITSlice FIFO.	
<div><div>1. This signal is applicable for asynchronous 1000BASE-X/SGMII over LVDS when Shared logic is in core.</div><div>2. This signal is applicable for asynchronous 1000BASE-X/SGMII over LVDS when shared logic is in example design.</div></div>			

#### Parameters Added from v15.2 to v16.0

The following table lists the parameters added to the core from v15.2 to v16.0.

**Table: Parameters Added in V16.0**

Generic Name	Applicability	Description	Values
NumOfLanes	Applicable in asynchronous 1000BASE-X/SGMII mode only	Number of lanes of PCS/PMA to be generated	1,2,3
Tx_In_Upper_Nibble	Applicable in asynchronous 1000BASE-X/SGMII mode only	See <a href="#">Lane Placement Parameters</a> for details	
TxLane0_Placement			
RxLane0_Placement			
TxLane1_Placement			
RxLane1_Placement			
EnableAsyncSGMII	Selection between synchronous SGMII solution based on component mode vs asynchronous LVDS SGMII solution based on Native mode for UltraScale.	Synchronous SGMII can be implemented using Asynchronous Solution. However Synchronous SGMII solution is given in order to maintain backward compatibility.	TRUE: Use Native mode asynchronous solution for SGMII over LVDS implementation. FALSE: Use Component mode synchronous solution for SGMII over LVDS implementation.
GT_Location	Applicable in UltraScale and UltraScale+ device families	Selection logic for GT location in terms of X and Y co-ordinates.	

#### Parameters Removed from v15.2 to v16.0



The following table lists the parameters that have been removed from v15.2 of the core to v16.0.

**Table: Parameters Removed in V16.0**

Generic Name	Applicability	Description
PHYADDR	Removed from all modes	This parameter has been converted into a port.

Ports Added from v15.1 to v15.2

**Table: Ports Added in v15.2**

In/Out	Port Name and Width	Description	What to do
Output	recclk_mmcm_reset	MMCM reset for MMCM generating rxuserclk, rxuserclk2, when RxGmiiClkSrc=RXOUTCLK. This is output only when the clocking logic is a part of the example design and applicable only for 7 series devices with MMCM is used for clock multiplication.	Connect to input signal of TEMAC when TEMAC is generated in internal mode and selecting RX clock source as RXOUTCLK.
Output	sgmii_rx_clk_r	Differential clock for GMII RX data when RxGmiiClkSrc=RXOUTCLK in SGMII mode.	Connect to input signal of TEMAC when TEMAC is generated in internal mode and selecting RX clock source as RXOUTCLK.
Output	sgmii_rx_clk_f	Differential clock for GMII RX data when RxGmiiClkSrc=RXOUTCLK in SGMII mode.	Connect to input signal of TEMAC when TEMAC is generated in internal mode and selecting RX clock source as RXOUTCLK.
Output	sgmii_rx_clk_en	Clock for GMII RX data	Connect to input signal of TEMAC when TEMAC is generated in internal mode and selecting RX clock source as RXOUTCLK.
All ports in		See the <i>UltraScale FPGAs Transceivers Wizard LogiCORE IP Product Guide</i> (PG182) for a description of the ports.	

Parameters Added from v15.1 to v15.2

The following table lists the parameters changes from v15.1 of the core to v15.2.

**Table: Parameters Added in v15.2**

Applicable Mode	Description	Values
<p><b>TXOUTCLK</b> Selects the clock for the receive side of the core as TXOUTCLK or RXOUTCLK. The fabric elastic buffer is not required when RXOUTCLK is selected because the RX datapath is synchronous to the recovered clock. Therefore, the <i>SGMII Capabilities</i> tab for SGMII speed selection is not required.</p> <p>the physical interface is the transceiver.</p>		TXOUTCLK (default) RXOUTCLK
<p><b>Include_shared_logic_in_Example_Design</b> If this parameter is selected, the Gigabit transceiver is pulled to shared level instance from the core.</p> <p>only when</p>		True False (default)

Applicable	Port Name	Description	Values
is selected for UltraScale and UltraScale+ devices.			
is applicable only for UltraScale and UltraScale+ devices with GTH and GTY transceiver types.			GTH (default) GTY

Ports Added from v15.0 to v15.1

The following table lists the ports added from v15.0 of the core to v15.1.

Table: Port Changes in v15.1

In/Out	Port Name and Width	Description	What to do
Input	gt_gtreclk1	Reference clock option for AMD UltraScale+™ /AMD UltraScale™ GT. This is enabled only when debug ports are enabled.	
Input	gt_cpllrefclkssel[2:0]	Reference clock select line for UltraScale+/UltraScale GT. This is enabled only when debug ports are enabled.	Default value is 3'b001 for gtrefclk0
Input	refclk625_p	LVDS reference clock when LvdsRefclk is selected as 625 MHz. <sup>1</sup>	
Input	refclk625_n	LVDS reference clock when LvdsRefclk is selected as 625 MHz. <sup>1</sup>	
Input	refclk156_25_p	LVDS reference clock when LvdsRefclk is selected as 156.25 MHz. <sup>1</sup>	
Input	refclk156_25_n	LVDS reference clock when LvdsRefclk is selected as 156.25 MHz. <sup>1</sup>	
1. Applicable only for UltraScale devices.			

Parameters Added from v15.0 to v15.1

The following table lists the port changes from v15.0 of the core to v15.1.

Table: Parameters Added in v15.1

Generic Name	Applicability	Description	Values
AppRefClk	Applicable only for LVDS modes.	Parameter to select reference clock for LVDS solution.	Valid values: 125 156.25 625

Port Changes from v14.3 to v15.0

Ports Added

Table: Ports Added in v15.0

Port Name and Width	Description	What to do
mmcm_reset	Reset for MMCM generating userclk, userclk2. This is output only when the clocking logic is a part of the example design.	This is the reset which is generated through 7 series gtwizard. For UltraScale devices this is same as cpllock. While connecting this to MMCM ensure that the pma_reset to the core is not dependent on mmcm_locked.
txpma_appld150	Only for UltraScale devices and is a part of transceiver debug interface.	See the UltraScale transceiver guide for more details.
txpma_appl	Only applicable only for UltraScale devices and is a part of transceiver debug interface.	Active-High signal forces TX output to a steady state. See the UltraScale transceiver guide for more details.
txpma_appl150	Only applicable only for 7 series devices and is a part of transceiver debug interface.	Active-High signal forces the TX output to a steady state. See the UltraScale transceiver guide for more details.
drpclk_bufg	drp clock for transceiver which is passed through a BUFG used to drive logic. This is applicable only for 7 series and Zynq devices, when shared logic is a part of example design.	drp clock is now connected to gtrefclk_bufg instead of userclk/userclk2 because of MMCM reset change, which makes userclk/userclk2 non-continuous until the MMCM is locked.
gtrefclk_bufg	drp clock for transceiver which is passed through a BUFG used to drive fabric logic. This is applicable only for 7 series and Zynq devices, when shared logic is a part of example design.	drp clock internally is now connected to gtrefclk_bufg instead of userclk/userclk2 because of MMCM reset change, which makes userclk/userclk2 non-continuous until the MMCM is locked.

Ports Removed

Table: Ports Removed in V15.0

Description	What to do
drpclk_bufg150 MHz free running clock.	drp clock is now connected to gtrefclk_bufg instead of userclk/userclk2 because of MMCM reset change, which makes userclk/userclk2 non-continuous until the MMCM is locked.
drpclk_bufg	drp clock is now connected to gtrefclk_bufg instead of userclk/userclk2 because of MMCM reset change, which makes userclk/userclk2 non-continuous until the MMCM is locked.

Parameters Added

Table: Ports Removed in V15.0

Description	What to do
drpclk_bufg150 MHz free running clock.	drp clock is now connected to gtrefclk_bufg instead of userclk/userclk2 because of MMCM reset change, which makes userclk/userclk2 non-continuous until the MMCM is locked.

Description	What to do
Independent clock required because an independent clock and hence drp clock are now selectable in GUI. 50 MHz free running clock.	

## Port Changes from v14.1 to v14.2

### Ports Added

The ports in the following table were added to the core (non-shared logic).

**Table: Ports Added**

In/Out	Port Name	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 AMD UltraScale™ devices. See for more details.
Input	Independent_clock_bufgdiv6	50 MHz clock	This is a new clock added that is required by the UltraScale gtwizard. The frequency of this clock must 50 MHz. This clock is visible only when the

In/Out	Port Name	Description	What to do
			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 the following table were changed or corrected as described.

Table: Ports Functionality Changed

Description	What has changed
<del>txoutclk</del> is enabled when shared logic is in the core and SGMII/BASE-X modes are selected. It was earlier derived from txoutclk outputcase of BASE-X mode; now it is the same as rxoutclk in the BASE-X and SGMII modes.	
<del>rxoutclk2</del> is enabled when shared logic is in the core and SGMII/BASE-X modes are selected. It was earlier derived from txoutclk outputcase of BASE-X mode; now it is the same as rxoutclk in BASE-X and SGMII modes.	
<del>ResetDone</del> 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 the following tables were added to the Transceiver Debug feature of the core.

Table: Ports Added for 7 seriesand 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: Ports Added for UltraScale 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_drpclkin	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_txprbsel_in[2:0]	Input	GT PRBS
gt_txprbsforceerr_in	Input	
gt_rxprbscntreset_in	Input	
gt_rxprbserr_out	Output	
gt_rxprbsel_in[2:0]	Input	
gt_loopback_in[2:0]	Input	GT Loopback <b>Note:</b> Loopback is not supported by the core when RxGmiiClkSrc=RXOUTCLK.
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_eyescanataerror_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

Signal	Direction	Description
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

#### Ports Removed

The ports in the following table were removed under the Transceiver Debug feature of the core.

**Table: 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_drpclock_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_txprbsel_in[2:0]	Input	GT PRBS
gt0_txprbsforceerr_in	Input	
gt0_rxprbscntreset_in	Input	
gt0_rxprbserr_out	Output	
gt0_rxprbsel_in[2:0]	Input	
gt0_loopback_in[2:0]	Input	GT Loopback  <b>Note:</b> Loopback is not supported by the core when RxGmiiClkSrc=RXOUTCLK.
gt0_txresetdone_out	Output	GT Status
gt0_rxresetdone_out	Output	
gt0_rxdisperr_out[3:0]	Output	
gt0_rxnotintable_out	Output	

Signal	Direction	Description
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 the following table were removed from the core (non-shared logic).

Table: Ports Removed

Port Name and Width	Description	What to Do
<del>Link_timer_configure[8:0]</del>	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.
<del>Link_timer_base[8:0]</del>	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.
<del>Link_timer_sgmr[8:0]</del>	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 the following table was removed from the core (non-shared logic).

Table: 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



Generic Name	Description	What to Do
		exists in wrappers underneath. This can also be controlled during generation of the core. See <a href="#">Port Descriptions</a> for guidelines for controlling the same.

#### Ports Added

The ports in the following table were added to the core (non-shared logic).

**Table: Ports Added (non-shared Logic)**

Port Name and Width	Description	What to do
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.
rxoutclk	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.
rxoutclk2	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.
rxoutclk	for AMD Artix™ 7 families. Indicates out clock from PLL0 of GT Common.	Should be connected to signal of same name from GT Common.
rxoutclk	for AMD Artix™ 7 families. Indicates reference out clock from PLL0 of GT Common.	Should be connected to signal of same name from GT Common.
rxoutclk	for AMD Artix™ 7 families. Indicates out clock from PLL1 of GT Common.	Should be connected to signal of same name from GT Common.
rxoutclk	for AMD Artix™ 7 families. Indicates reference out clock from PLL1 of GT Common.	Should be connected to signal of same name from GT Common
rxoutclk	for AMD Artix™ 7 families. Indicates out PLL0 of GT Common has locked.	Should be connected to signal of same name from GT Common.
rxoutclk	for AMD 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.
rxoutclk	for AMD 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.
rxoutclk	for AMD Artix™ 7 families. Indicates out clock from PLL of GT Common.	Should be connected to signal of same name from GT Common.
rxoutclk	for AMD 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 the following table were added to the core, but only if the transceiver debug feature was requested during core customization. See the relevant transceiver user guide for more information on using these control/status ports.

**Table: Ports Added for Transceiver Debug Feature**

Port Name and Width	In/Out	Description	What to do <sup>1</sup>
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	

Port Name and Width	In/Out	Description	What to do <sup>1</sup>
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_txprbsel_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_rxprbsel_in	Input	RX Pattern Checker control signals to test signal integrity	
gt0_loopback_in[2:0]	Input	<u>Loopback within transceiver</u> 🔔 <b>Note:</b> Loopback is not supported by the core when RxGmiiClkSrc=RXOUTCLK.	
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_eyes candataerror	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	

Port Name and Width	In/Out	Description	What to do <sup>1</sup>
1. Drive this signal according to the relevant transceiver user guide. If the DRP interface was unused in previous revisions, then generate the core without the Transceiver Debug feature.			

Ports Moved

The ports in the following table 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: Ports Moved (Non-Shared Logic)

Port Name and Width	Description	What to do
gt0_drpdo_output, gt0_drprdy_out	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	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 the1000BASE-X or 2500BASE-X IEEE 802.3-2008 clause 36 transmitter and receiver state machines.

## Introduction

The following table shows the Ordered Sets defined in IEEE 802.3-2008 clause 36. 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 with the remainder of this Appendix. See IEEE 802.3-2008 clause 36 for further information on these Orders Sets.

Table: 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 <sup>1</sup>
/C2/	Configuration 2	4	/K28.5/D2.2/Config_Reg <sup>1</sup>
/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/

Code	Ordered_Set	No. of Code-Groups	Encoding
1. Two data code-groups representing the Config_Reg value (contains auto-negotiation information).			

Start of Frame Encoding

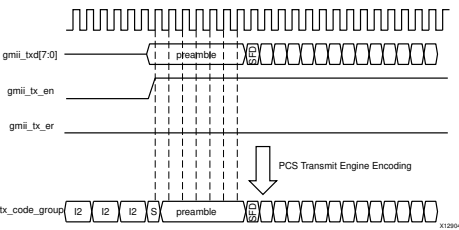
The Even Transmission Case

The following figure 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: 1000BASE-X Transmit State Machine Operation (Even Case)

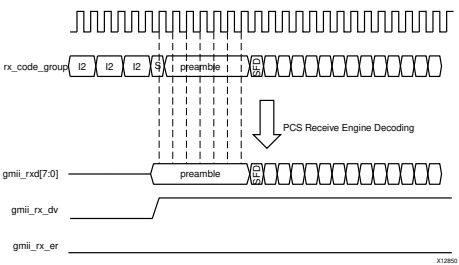


Reception of the Even Case

The following figure 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: 1000BASE-X Reception State Machine Operation (Even Case)

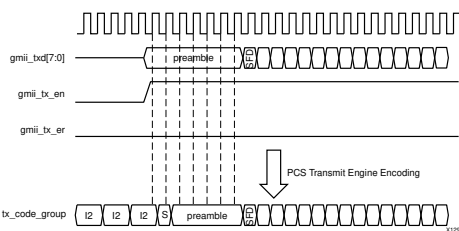


The Odd Transmission Case

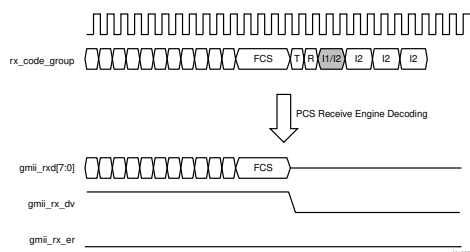
The following figure shows the translation of GMII encoding into the code-group stream performed by the PCS Transmit Engine; this stream is transmitted out of the core, either serially using the device-specific transceiver, or in parallel across the TBI.

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: 1000BASE-X Transmit State Machine Operation (Odd Case)





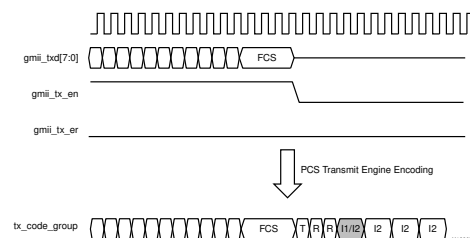


## The Odd Transmission Case

The following figure 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 `gmi_i_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: 1000BASE-X Transmit State Machine Operation (Even Case)**

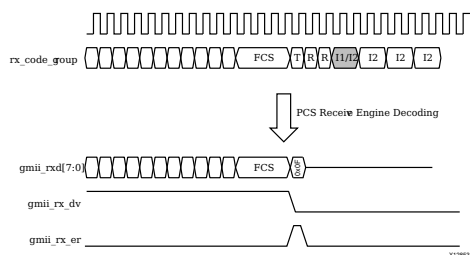


## Reception of the Odd Case

The following figure 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 `gmi_i_rx_er` signal is driven High and the frame extension code of `0x0F` is driven onto `gmi_i_rxd[7:0]`. This occurs even in full-duplex mode.

**Figure: 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

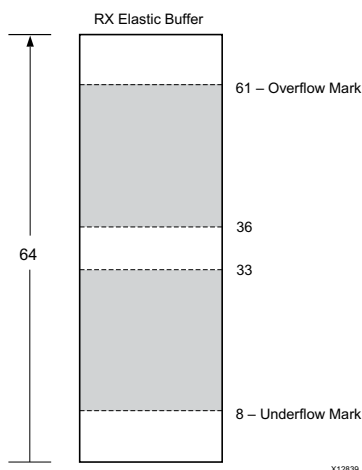
inserting or removing Idle characters. The receive elastic buffer always attempts to restore the buffer occupancy to the half full level during an interframe gap. See [Clock Correction](#).

## Receive Elastic Buffers: Depths and Maximum Frame Sizes

### Device Specific Transceiver Receive Elastic Buffers

The following figure shows the transceiver receive elastic buffer depths and thresholds in AMD UltraScale+™, AMD UltraScale™ architecture, AMD 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: 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:

$$5000 \times 25 = 125000 \text{ bytes}$$

The following figure translates this into maximum frame size at different Ethernet speeds. At SGMII speeds lower than 1 Gbps, performance is diminished because bytes are repeated multiple times (see [Using the Client-Side GMII for the SGMII Standard](#)).

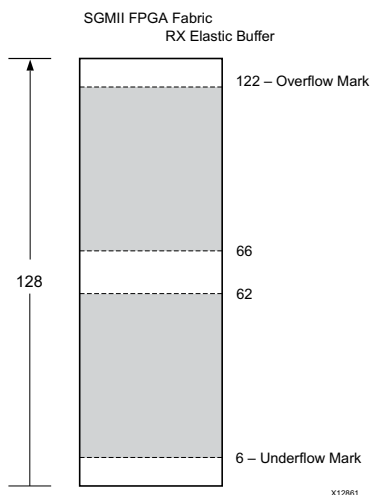
**Table: Maximum Frame Sizes: Transceiver Receive Elastic Buffers (100 ppm Clock Tolerance)**

Standard/Speed	Maximum Frame Size (Bytes)
1000BASE-X (1 Gbps only)	125000
SGMII (1 Gbps)	125000
SGMII (100 Mbps)	12500
SGMII (10 Mbps)	1250

### SGMII FPGA Logic Receive Elastic Buffer

The following figure shows the FPGA logic receive elastic buffer depth. 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: Elastic Buffer Size for All Transceiver Families**



The shaded area of the previous figure 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:

$$5000 \times 56 = 280000 \text{ bytes}$$

The following figure translates this into maximum frame size at different Ethernet speeds. At SGMII speeds lower than 1 Gbps, performance is diminished because bytes are repeated multiple times. See [Using the Client-Side GMII for the SGMII Standard](#).

**Table: Maximum Frame Sizes: FPGA logic Receive Elastic Buffers (100 ppm Clock Tolerance)**

Standard/Speed	Maximum Frame Size (Bytes)
1000BASE-X (1 Gbps only)	280000
SGMII (1 Gbps)	280000
SGMII (100 Mbps)	28000
SGMII (10 Mbps)	2800

## TBI Receive Elastic Buffer

For SGMII / Dynamic Switching

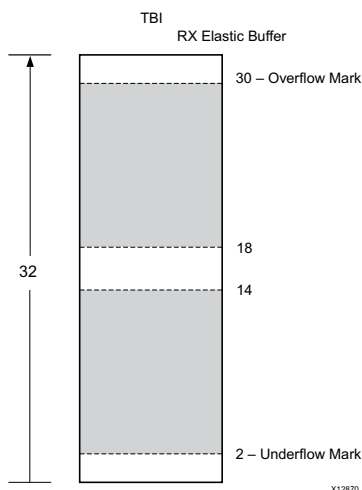
The receive elastic buffer used for the SGMII or Dynamic Switching is identical to the method used in [SGMII FPGA Logic Receive Elastic Buffer](#).

For 1000BASE-X

The following figure 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: TBI Elastic Buffer Size for All Families**





The shaded area of the previous figure 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:

$$5000 \times 12 = 60000 \text{ bytes}$$

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. The receive elastic buffer also performs clock correction on /C1/, /C2/ configuration code words received during auto-negotiation. This is similar to the way that clock correction is done on /C1/ in the transceiver elastic buffer.

- 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 Gbps (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 Gbps 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 [1000BASE-X State Machines](#)). Of these, only /I2/ can be used as clock correction characters.

In a minimum interframe gap at 1 Gbps, 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  $5000 \times 4 = 20000$  bytes in length. So if the frame being received is 20000 bytes in length or shorter, at 1 Gbps, 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 Mbps (SGMII)

At SGMII, 100 Mbps, each byte is repeated 10 times. This also applies to the interframe gap period. For this reason, the minimum of 8

bytes for the 1 Gbps case corresponds to a minimum of 80 bytes for the 100 Mbps case. Additionally, the majority of characters in this 80-byte interframe-gap period are going to be the /I2/ clock correction characters. Because of the clock correction circuitry design, a minimum of 20 /I2/ code groups will be available for removal. This translates into 40 bytes, giving a maximum run size of  $40 \times 5000 = 200000$  bytes. Because each byte at 100 Mbps is repeated ten times, this corresponds to an Ethernet frame size of 20000 bytes, the same size as the 1 Gbps case. So in summary, at 100 Mbps, for any frame size of 20000 bytes or less, it can still be assumed that the elastic buffer will return to half full occupancy before the start of the next frame. However, a frame size of 20000 is larger than can be received in the device-specific transceiver elastic buffer (see [Receive Elastic Buffers: Depths and Maximum Frame Sizes](#)). Only the SGMII FPGA Logic receive elastic buffer is large enough.

### Idle Character Removal at 10 Mbps (SGMII)

Using a similar argument to the 100 Mbps case, it can be shown that clock correction circuitry can also cope with a frame size up to 20000 bytes. However, this is larger than the maximum frame size for any elastic buffer provided with the core (see [Receive Elastic Buffers: Depths and Maximum Frame Sizes](#)).

## Maximum Frame Sizes for Sustained Frame

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](#)).

[Maximum Frame Sizes for Sustained Frame](#) 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: 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 1000BASE-X)
1000BASE-X (1 Gbps)	20000 (limited by clock correction)	20000 (limited by clock correction)	20000 (limited by clock correction)
SGMII 1 Gbps	20000 (limited by clock correction)	20000 (limited by clock correction)	20000 (limited by clock correction)
SGMII 100 Mbps	20000 (limited by clock correction)	9000 (limited by buffer size)	20000 (limited by clock correction)
SGMII 10 Mbps	2800 (limited by buffer size)	900 (limited by buffer size)	2800 (limited by buffer size)

## Jumbo Frame Reception

A jumbo frame is an Ethernet frame which is deliberately larger than the maximum sized Ethernet frame allowed in the *IEEE 802.3-2008* specification. The size of jumbo frames that can be reliably received is identical to the frame sizes defined in [Receive Elastic Buffers: Depths and Maximum Frame Sizes](#).

## 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 or 2500BASE-X designs. The extra logic required to accomplish this is described in this appendix.

**Note:** AMD Virtex™ 7 devices support GMII at 3.3 V or lower only in certain parts and packages; see the AMD Virtex™ 7 Device Documentation.AMD Zynq™ 7000, AMD Kintex™ 7, and AMD Artix™ 7 devices support GMII at 3.3V or lower.

## GMII Transmitter Logic (Zynq 7000 and 7 Series)

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 (312.5 MHz in 2.5G mode) 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 or 2500BASE-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.

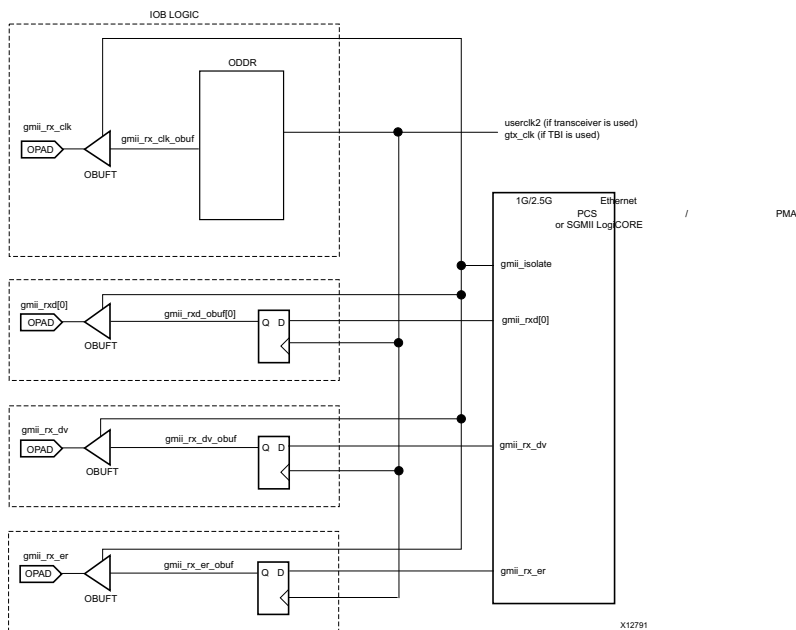
## GMII Receiver Logic

The following figure 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.

The following figure 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 [Asynchronous LVDS Transceiver for Versal devices](#), and [SGMII/Dynamic Switching with Transceivers](#).

**Figure: External GMII Receiver Logic**



## Debugging

This appendix includes details about resources available on the Support website and debugging tools.

If the IP requires a license key, the key must be verified. The AMD Vivado™ design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with an error. License checkpoints are enforced by the following tools:

- Vivado Synthesis
- Vivado Implementation
- `write_bitstream` (Tcl command)

**Note:** IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.

## Finding Help with AMD Adaptive Computing Solutions

To help in the design and debug process when using the core, the [Support web page](#) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. The [Community Forums](#) are also available where members can learn, participate, share, and ask questions about AMD Adaptive

Computing solutions.

## Documentation

This product guide is the main document associated with the core. This guide, along with documentation related to all products that aid in the design process, can be found on the [Support web page](#) or by using the AMD Adaptive Computing Documentation Navigator. Download the Documentation Navigator from the [Downloads page](#). 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.

For information about all AMD Ethernet solutions, see [https://www.xilinx.com/products/design\\_resources/conn\\_central/protocols/gigabit\\_ethernet.htm](https://www.xilinx.com/products/design_resources/conn_central/protocols/gigabit_ethernet.htm).

## Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with an AMD Adaptive Computing product. Answer Records are created and maintained daily to ensure that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main [Support web page](#). To maximize your search results, use 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 1G/2.5G Ethernet PCS/PMA or SGMII Core

AR [54667](#).

## Technical Support

AMD Adaptive Computing provides technical support on the [Community Forums](#) for this AMD LogiCORE™ IP product when used as described in the product documentation. AMD Adaptive Computing cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To ask questions, navigate to the [Community Forums](#).

## Debug Tools

There are many tools available to address design issues. It is important to know which tools are useful for debugging various situations.

### Vivado Design Suite Debug Feature

The AMD Vivado™ Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows 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 AMD devices. The Vivado logic analyzer is used to interact with the logic debug LogiCORE 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](#)).

## Reference Boards

Various AMD development boards support the core. These boards can be used to prototype designs and establish that the core can communicate with the system.

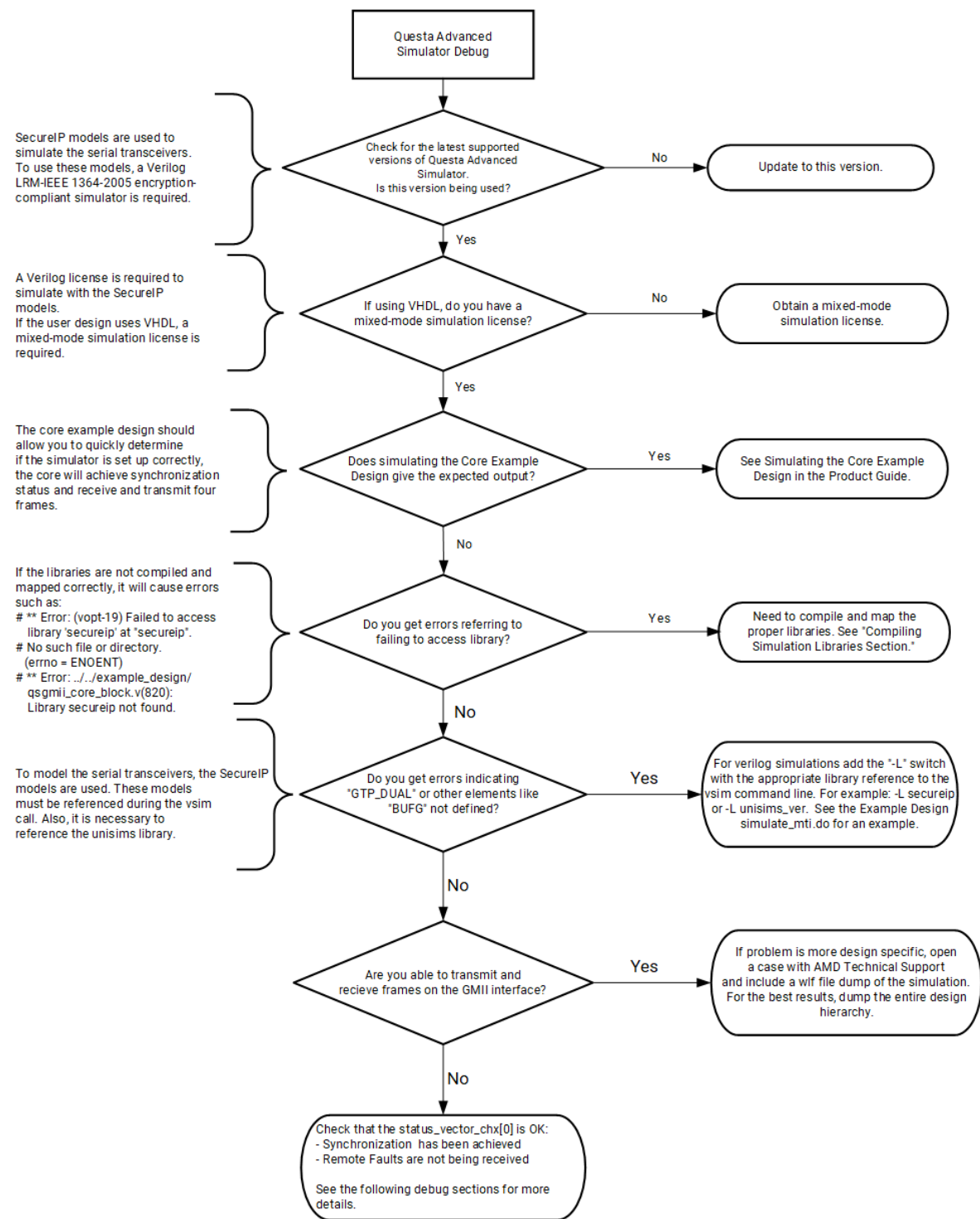
- 7 series FPGA evaluation boards

- KC705
- VC707

## Simulation Debug

The simulation debug flow for Mentor Graphics Questa Advanced Simulator is illustrated in the following figure. A similar approach can be used with other simulators.

**Figure: Simulation Debug Flow**



X16334-030916

## 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 AMD Vivado™ debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the debug feature 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 devices, if the transceiver is not coming out of the reset sequence, check the following:
  1. If the Transceiver Control and status is enabled, check the DRP clock frequency. The frequency should be exactly same as that selected through DrpClkRate. It is recommended to connect the independent clock to the same clock frequency.
  2. If (a) is not applicable the independent clock frequency must be exactly same as that selected through the Vivado IDE GUI or through the parameter DrpClkRate. The DRP clock internally is connected to the independent clock in this case.

## 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.
- 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 3](#).

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 or 2500BASE-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 debug feature 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

### Support Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Support](#).

### Finding Additional Documentation

#### Documentation Portal


The AMD Adaptive Computing Documentation Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Documentation Portal, go to <https://docs.xilinx.com>.

#### Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select Help > Documentation and Tutorials.
- On Windows, click the Start button and select Xilinx Design Tools > DocNav.
- At the Linux command prompt, enter docnav.

---

 **Note:** For more information on DocNav, refer to the *Documentation Navigator User Guide* ([UG968](#)).

---

### Design Hubs

AMD Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the Design Hubs View tab.
- Go to the [Design Hubs](#) web page.

### References

These documents provide supplemental material useful with this guide:

1. *FPGAs SelectIOResources User Guide* ([UG471](#))
2. *UltraScale Architecture SelectIO Resources User Guide* ([UG571](#))
3. *7 Series FPGAs Transceivers Wizard LogiCORE IP Product Guide* ([PG168](#))
4. *7 Series FPGAs GTX/GTH Transceivers User Guide* ([UG476](#))
5. *7 Series FPGAs GTP Transceivers User Guide* ([UG482](#))
6. *UltraScale Architecture GTH Transceivers User Guide* ([UG576](#))
7. *UltraScale Architecture GTY Transceivers User Guide* ([UG578](#))
8. *UltraScale FPGAs Transceivers Wizard LogiCORE IP Product Guide* ([PG182](#))
9. *7 Series FPGAs Clocking Resources User Guide* ([UG472](#))
10. *UltraScale Architecture Clocking Resources User Guide* ([UG572](#))
11. *Tri-Mode Ethernet MAC LogiCORE IP Product Guide* ([PG051](#))
12. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* ([UG994](#))



13. *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
14. *Vivado Design Suite User Guide: Getting Started* ([UG910](#))
15. *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))
16. *ISE to Vivado Design Suite Migration Guide*([UG911](#))
17. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
- 18.

## Training Resources

1. [Designing FPGAs Using the Vivado Design Suite 1 Training Course](#)
2. [Vivado Design Suite QuickTake Video Tutorials](#)

## Revision History

The following table shows the revision history for this document.

Section	Revision Summary
11/01/2023 Version 16.2	
<a href="#">Transceiver Logic for Versal Devices</a>	Added.
<a href="#">Asynchronous 1000BASE-X/SGMII over LVDS</a>	Updated.
<a href="#">IP Facts</a>	Updated.
05/24/2023 Version 16.2	
<a href="#">Asynchronous 1000BASE-X/SGMII over LVDS</a>	Updated the section
<a href="#">Synchronous SGMII over LVDS Using Component Mode</a>	Updated the section.
11/25/2022 Version 16.2	
<a href="#">Asynchronous SGMII/1000BASE-X Over LVDS</a>	Added a note.
07/08/2022 Version 16.2	
N/A	For Versal devices, the GEM support is added in the GUI.
06/16/2021 Version 16.2	
N/A	Async LVDS mode configuration is supported for Versal ACAP devices.
02/04/2021 Version 16.2	
N/A	Added information for Versal.
06/26/2020 Version 16.2	
<a href="#">Configuration and Status Vector Ports</a>	Updated an_restart_config signal description.
<a href="#">Customizing and Generating the Core</a>	Updated figures.
<a href="#">Resets</a>	Updated the section.
05/22/2019 Version 16.1	
<a href="#">Register 1: Status Register</a>	<ul style="list-style-type: none"> <li>Updated note #2 in <a href="#">Table 1</a> and <a href="#">Table 3</a>.</li> <li>Updated Attributes for Bit[2] in <a href="#">Register 1: Status Register</a>.</li> </ul>
11/14/2018 Version 16.1	
N/A	<ul style="list-style-type: none"> <li>Added preamble shrinkage bullet in <a href="#">Features</a>.</li> <li>Updated Device Specific Transceiver columns for Artix 7 in <a href="#">Table 2</a>.</li> </ul>
10/04/2017 Version 16.1	

Section	Revision Summary
N/A	Added 2500BASE-X support for Zynq 7000 devices in <a href="#">Table 2</a> .
06/07/2017 Version 16.1	
N/A	Added gtpowergood output port for UltraScale and UltraScale+ device families.
04/05/2017 Version 16.0	
<a href="#">Upgrading</a>	Updated tables in Appendix B (Migrating and Upgrading) for ports and parameters for Asynchronous 1000BASE-X/SGMII over LVDS for UltraScale and UltraScale+ device families
10/05/2016 Version 16.0	
16.0	<ul style="list-style-type: none"> <li>Added support for Asynchronous 1000BASE-X/SGMII over LVDS for UltraScale and UltraScale+ device families.</li> <li>Added support to provide GT_Location in Vivado IDE.</li> <li>Added support for Spartan-7 devices.</li> <li>Added support for clock correction on /C1/,/C2/ code groups in receive elastic buffer.</li> </ul>
04/06/2016 Version 15.2	
N/A	<ul style="list-style-type: none"> <li>Added support for GT in example design for UltraScale and UltraScale+ devices.</li> <li>Added support for RX GMII interface to work at RXOUTCLK.</li> <li>Added support for Virtex UltraScale+ family.</li> <li>Added selection option for GTH and GTY transceivers for UltraScale and UltraScale+ devices.</li> <li>Added support to interface with Gigabit Ethernet MAC of Zynq UltraScale+ devices.</li> </ul>
11/18/2015 Version 15.1	
N/A	Added support for UltraScale+ families.
09/30/2015 Version 15.1	
N/A	<ul style="list-style-type: none"> <li>Added 2.5G support for Artix devices (-2 and -3 speed grades).</li> <li>Added LvdsRefClk for LVDS reference clock selection for UltraScale devices.</li> <li>Added gt_gtrefclk1, gt_cp1lrefclkssel[2:0] in UltraScale debug signals.</li> <li>Updated Register 2 and Register 3 values.</li> </ul>
04/01/2015 Version 15.0	
15.0	<ul style="list-style-type: none"> <li>Added 2.5 Gbps support for 7 series devices (except Artix 7 and Zynq devices with Artix fabric) and UltraScale devices.</li> <li>Added options for gtrefclk and DRP/Free run clock selection for UltraScale devices.</li> <li>Added txinhibit to the transceiver debug signals.</li> <li>Added pcsrsvdin to the transceiver debug signals for UltraScale devices.</li> <li>Added mmcm_reset port for modes using transceivers.</li> </ul>
10/01/2014 Version 14.3	
N/A	

Section	Revision Summary
	<ul style="list-style-type: none"> <li>Added 1588(PTP) GTH transceiver support for UltraScale architecture.</li> <li>Document re-structured.</li> <li>Added information on shared logic for cases using device-specific transceiver.</li> </ul>
04/02/2014 Version 14.2	
N/A	<ul style="list-style-type: none"> <li>Added SGMII over LVDS for UltraScale devices.</li> <li>Modified status_vector(0) and LINK_STATUS register to take care of reset sequence completion of transceivers.</li> <li>Updated screen displays in chapter 13.</li> <li>Added reset_done signal to several figures.</li> <li>Added External MDIO feature.</li> <li>Modified ambiguous text for BUFG usage in 7 series device SGMII over LVDS.</li> </ul>
12/18/2013 Version 14.1	
N/A	<ul style="list-style-type: none"> <li>Added UltraScale architecture support.</li> <li>Added 1588(PTP) GTH transceiver support in the core.</li> <li>Updated screen displays in Chapter 13.</li> </ul>
10/02/2013 Version 14.0	
N/A	<ul style="list-style-type: none"> <li>Removed link timer value ports from block_wrapper</li> <li>Enhanced support for IP integrator.</li> <li>Reduced warnings in synthesis and simulation.</li> <li>Updated clock synchronizers to improve Mean Time Between Failures (MTBF) for metastability.</li> <li>Added optional transceiver control and status ports.</li> <li>Added Vivado IDE option to include or exclude shareable logic resources in the core.</li> <li>Added new board Vivado IDE tab for targeting evaluation boards.</li> </ul>
06/19/2013 Version 13.0	
N/A	<ul style="list-style-type: none"> <li>Revision number advanced to 13.0 to align with core version number 13.0.</li> <li>Added Zynq 7000 SoC EMAC support.</li> <li>Added 1588 (PTP) support in the core.</li> <li>Modified PHYAD to be a GUI option instead of block level port.</li> <li>Updated Figures 2-2, 2-3, 2-6, 2-7, 2-8, 2-9, 13-1, 13-2, 13-3, and 13-4.</li> </ul>
03/20/2013 Version 2.0	
N/A	<ul style="list-style-type: none"> <li>Updated to core version 12.0.</li> <li>Removed all material related to devices not supported by the Vivado Design Suite.</li> <li>Removed all material related to ISE® Design Suite, CORE Generator™ tools, and UCF.</li> <li>Updated 7 series FPGA transceivers diagrams.</li> <li>Added Zynq support for SGMII over LVDS feature.</li> </ul>
12/18/2012 Version 1.2	

Section	Revision Summary
Debugging	<ul style="list-style-type: none"> <li>Updated for 14.4 and 2012.4. Updated to core version 11.5.</li> <li>Updated Debugging appendix.</li> <li>Added new information about Zynq 7000 FPGAs throughout the guide</li> <li>Added XCI file information.</li> <li>Added statement about wait time for Vivado Design Suite use with transceiver wizards.</li> <li>Updated Figures 6-8, 6-9, 6-10, 6-17, 7-2, and G-1.</li> <li>Added XDC information.</li> </ul>
10/16/2012 Version 1.1	
N/A	<ul style="list-style-type: none"> <li>Updated for 14.3 and 2012.3.</li> <li>Added Gigabit Ethernet EDK application for Zynq 7000 devices.</li> </ul>
07/25/2012 Version 1.0	
N/A	<p>Initial Xilinx release in product guide format. This document is based on the following documents:</p> <ul style="list-style-type: none"> <li>LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII v11.3 Product Guide</li> <li>LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII v11.3 Data Sheet</li> </ul>

## Please Read: Important Legal Notices

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes. THIS INFORMATION IS PROVIDED "AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

## Copyright

© Copyright 2012-2023 Advanced Micro Devices, Inc. AMD, the AMD Arrow logo, Artix, Kintex, UltraScale+, Versal, Virtex, Vivado, Zynq, and combinations thereof are trademarks of Advanced Micro Devices, Inc. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the US and/or elsewhere. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.