USXGMII Ethernet Subsystem v2.0

Product Guide

Vivado Design Suite

PG251 (v2.0) December 4, 2025

Table of Contents

Introduction

The Universal Serial 10GE Media Independent Interface (USXGMII) IP Core implements an Ethernet Media Access Controller (MAC) with a mechanism to carry a single port of 10M, 100M, 1G, 2.5G, 5G, or 10GE over an IEEE 802.3 Clause 49 BASE-R physical coding sublayer/physical layer (PCS/PHY). The USXGMII IP core is delivered as encrypted register transfer level (RTL) through the AMD Vivado™ Design Suite targeted for AMD FPGAs.

Features

- Designed to meet the USXGMII specification EDCS-1467841 revision 1.4
- Supports 10M, 100M, 1G, 2.5G, 5G, or 10GE data rates over a 10.3125 Gb/s link
- Both media access control (MAC) and PCS/PMA functions are included
- Code replication/removal of lower rates onto the 10GE link
- Rate adaption onto user clock domain
- Low data path latency
- 32-bit AXI4-Stream interface for datapath
- Optional AXI4-Lite register interface
- Support for 802.3x and priority-based pause operation
- Detailed statistics gathering
- Support for custom preambles
- Supports deficit idle count (DIC)
- Supports Auto-Negotiation features

IP Facts

IP Facts Table	
Core Specifics	
Supported Device Family	AMD Versal™ adaptive SoC, AMD Virtex 7, AMD UltraScale™, AMD Kintex™ UltraScale™, AMD Virtex™ UltraScale+™, AMD Kintex™ UltraScale+™, AMD Zynq™ UltraScale+™ MPSoC, AMD Artix™ UltraScale+™
Supported User Interfaces	AXI4-Stream 32-bit Optional AXI4-Lite
Resources	Performance and Utilization Web Page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog
Test Bench	Verilog
Constraints File	XDC
Simulation Model	Verilog
Supported S/W Driver	Linux
Tested Design Flows	
Design Entry	Not Applicable
Simulation	For supported simulators, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).
Synthesis	Not Applicable
Support	
Release Notes and Known Issues	Master Answer Record: N/A
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Support web page	

Notes:

1. AMD recommends that you join the Ethernet Alliance to gain access to the USXGMII specification. For more information, visit NBASE-T Library.

Chapter 2

Core Overview

The USXGMII IP Core provides an architecture to convey a single port of Ethernet over a 10GE BASE-R link in a way that maximizes existing standards and thus reduces risk. The port can operate at an effective data rate of 10M, 100M, 1GE, 2.5GE, 5GE, or 10GE. The IP core is implemented with a low-latency 32-bit datapath.

Typical applications include connecting to an electrical Copper PHY (10GBASE-T) application-specific standard products (ASSPs).

This product guide contains an overview of the USXGMII IP core as well as example application, and licensing.

Navigating Content by Design Process

AMD documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers hardware, IP and Platform Development, including creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, evaluating the AMD Vivado™ timing, resource and power closure. This document also covers the hardware platform for system integration. Topics in this document that apply to this design process include:

- Port Descriptions
- Register Space
- Clocking and Resets
- Customizing and Generating the Subsystem
- Chapter 6: Example Design

Feature Summary

- Complete Ethernet MAC and PCS functions

- 32-bit serialize/deserializer (SerDes) interface using AMD UltraScale™, AMD UltraScale+™, and AMD Versal™ adaptive SoC GT transceiver operating with Asynchronous Gearbox enabled
- Pause Processing including IEEE std. 802.3 Annex 31D (Priority based Flow Control)
- Custom preamble
- Clause 37 Auto Negotiation for USXGMII
- Configurable speed from 10 Mb/s to 10 Gb/s to connect to 10G base-T PHY

Licensing and Ordering

This AMD LogiCORE™ IP module is provided at an additional charge with the AMD Vivado™ Design Suite under the terms of the End User License.

For more information, refer to the USXGMII product web page.

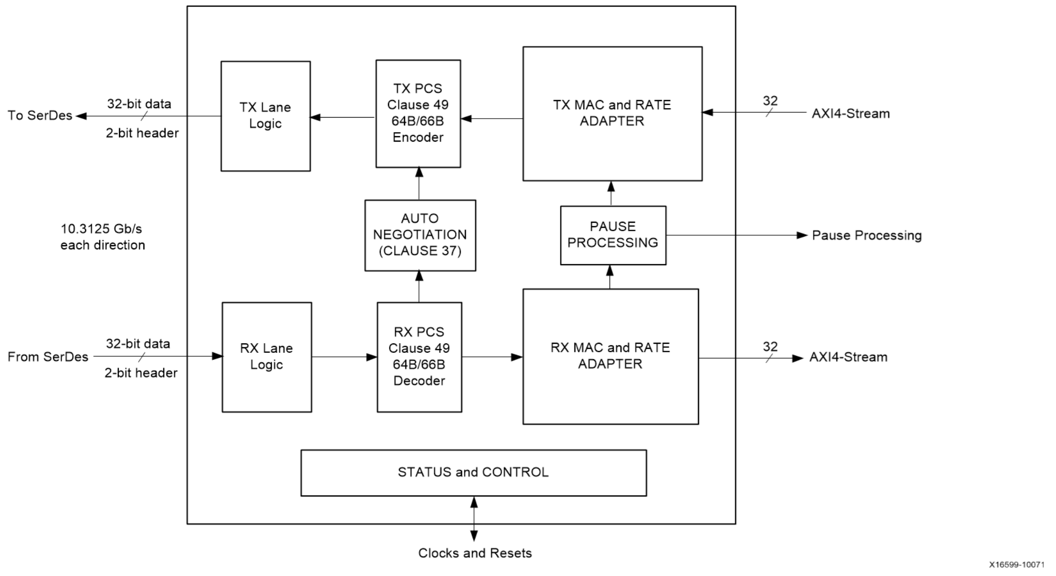
Information about other AMD LogiCORE™ IP modules is available at the Intellectual Property page. For information about pricing and availability of other AMD LogiCORE IP modules and tools, contact your local sales representative.

Chapter 3

Product Specification

The following figure shows the block diagram of the USXGMII IP, not including the GT transceiver.

Figure 1: **Block Diagram**



The USXGMII IP Core uses two data signals in each direction to convey frame data and link rate information between a single or multi-port PHY and the Ethernet MAC(s). The data signals operate at 10.3125 Gb/s (USXGMII/XFI), using clock data recovery (CDR) technology to recover the clock at the MAC and PHY serial interfaces. Due to the high speed of operation, each of these signal pairs is realized as differential pairs thus optimizing signal integrity while minimizing system noise.

The USXGMII core leverages the 64B/66B PCS defined in IEEE 802.3 Clause 49. Data replication is used to encapsulate lower Ethernet rates (10M, 100M, 1G, etc) into BASE-R 66b words for translation into a USXGMII stream via the Clause 49 PCS.

The PCS is unchanged with additional functionality being added via the "ordered set" mechanism defined by the IEEE Group.

TX Data Path

The 32-bit AXI4-Stream interface supports user-defined out-of-band preamble values, allowing USXGMII messages to be passed on packet preamble bytes.

On TX, every 32-bit word is replicated several times according to the rate selected by the auto-negotiation with the link partner, or by direct programming of the `ctl_usxgmii_rate[2:0]` port without auto-negotiation, when `ctl_umii_an_bypass` = 1 (see Port Descriptions for encoding). Code word replication is used to pad out the less than 10GE rates, and Start and Terminate words are modified according to the USXGMII specification for seamless decoding.

RX Datapath

On RX, replicated 32-bit words are discarded before the remaining unique packet data is passed on. The received preamble bytes are extracted from received packets and passed out alongside packet start of packets (SOPs).

As with the TX data path, when `ctl_umii_an_bypass` = 1, the USXGMII RX rate is determined by `ctl_usxgmii_rate[2:0]` (see Port Descriptions for encoding).

USXGMII at Lower Speeds

The following figures illustrate the start and termination of a packet transfer at 5 Gb/s.

Figure 2: **RX - Start of a Packet at 5 Gb/s**

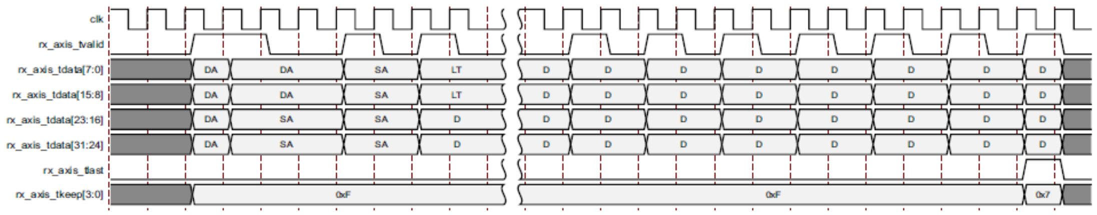
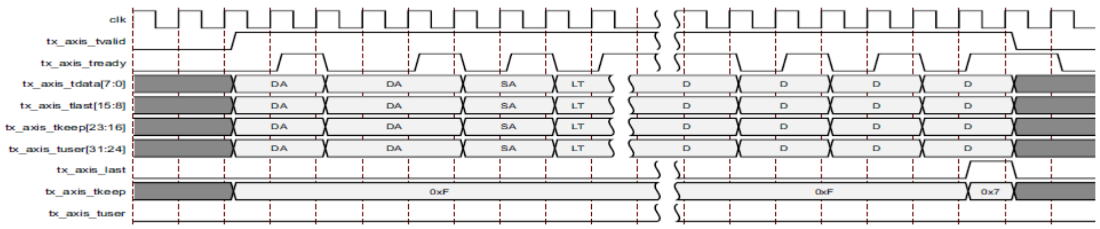


Figure 3: **TX - Start of a Packet at 5 Gb/s**



Each word of the preamble and payload is replicated 2/4/10/100/1000 times for 5G/2.5G/1G/100M/10M, respectively.

Auto-Negotiation

In the USXGMII IP Core, the auto-negotiation is based on clause 37 of IEEE 802.3. The operation conforms to the description provided by EDCS-1467841 NBASE-T Universal SXGMII: Copper PHY ERS.

The USXGMII core creates three new types of ordered sets, besides the existing local fault and the remote fault. The `UsxgmiiChannelInfo` carries the auto-negotiation information. The control register `ctl_umii_an_mr_adv_ability` maps to `UsxgmiiChannelInfo`. The other two are `UmiiLocalFault` and `UmiiRemoteFault`. This ordered set is used to transfer the 16-bit `UsxgmiiChannelInfo` word(s) between the link partners. The bit meanings are laid out in EDCS-1467841. Amidst this data is the desired rate that can be specified using bits 11:9 of `ctl_umii_an_mr_adv_ability`.

During auto-negotiation, each link partner continuously broadcasts the `UsxgmiiChannelInfo` word(s) to the respective link partners, and each link partner examines the received channel information message and compares that message to what was being broadcast.

Then, following the clause 37 protocol, the auto-negotiation circuit will idle the lanes for the link-time-out value, and then the circuit will set the negotiated rate and enable mission-mode.

Currently, the core provides an `autoneg_bypass`, similar in behavior to that of clause 73. When this is set, the `autoneg` function is disabled, and the operation is preset to mission mode at the data rate that is set by the `ctl_usxgmii_rate` input.

Currently, the auto-negotiation resolves the rate to the lowest requested rate. However, there is an ambiguity in EDCS-1467841 that does not specify how rates are to be resolved.

Clause 37 Auto-Negotiation Operation

Signal names are:

- `ctl_umii_an_mr_an_enable`
- `ctl_umii_an_mr_restart_an`
- `ctl_umii_an_mr_main_reset`
- `ctl_umii_an_bypass`

The `ctl_umii_an_mr_main_reset` is not the circuit reset input. It is the `mr_main_reset` as defined in clause 37.

When `ctl_umii_an_mr_main_reset` is asserted, the auto-negotiation (AN) will hold the state AN_ENABLE. In this state, if `ctl_umii_an_mr_an_enable` is set, the AN will configure the PCS for a 10G rate, and it will continuously send a zero `usx_config` word, but it will not do anything else until `ctl_umii_an_mr_main_reset` is cleared. If `ctl_umii_an_mr_an_enable` is not set, the AN will set the PCS to continuously send IDLES. The rate will be whatever was previously set. If there were no previous negotiations, the rate set will be 10G. The AN will hold in state AN_ENABLE until the `ctl_umii_an_mr_main_reset` signal is removed.

When `ctl_umii_an_mr_main_reset` is removed, then, if `ctl_umii_an_mr_an_enable` is set, the AN will proceed to state `AN_RESTART`, and it will begin to negotiate the link, as per clause 37. If `ctl_umii_an_mr_an_enable` is not set, the AN will go to state AN_DISABLE_LINK_OK. In this state, the PCS is configured in mission mode, (that is, ready for traffic), at whatever rate was previously negotiated, and the AN circuit holds there.

If the `ctl_umii_an_bypass` signal is set, it automatically clears the internal `mr_an_enable` signal into the AN circuit, regardless of the setting of the `ctl_umii_an_mr_an_enable` input. It also automatically sets the PCS rate to whatever the `ctl_usxgmii_rate` input is set to, and it sets the PCS to mission-mode. The `ctl_umii_an_bypass` signal will also override the above described behavior of the `ctl_umii_an_mr_main_reset`. That is, the PCS will be configured into mission mode as described regardless of how `ctl_umii_an_mr_main_reset` and `ctl_umii_an_mr_an_enable` are set.

If the `ctl_umii_an_bypass` signal is not set, and the `ctl_umii_an_mr_main_reset` is also not set, and if the `ctl_umii_an_mr_an_enable` is set, and then if `ctl_umii_an_mr_restart_an` is asserted, the AN will enter state AN_RESTART, where it will proceed to negotiate the link, as per clause 37. To signal another restart, the `ctl_umii_an_mr_restart_an` has to be cycled. That is, it has to be cleared and then set again. The restart is triggered on the positive transition of the `ctl_umii_an_mr_restart_an` signal.

Standards

The USXGMII IP conforms to the EDCS 1467841-NBASE-T USXGMII: Copper PHY ERS specification, revision 1.4 (AMD recommends that you join the NBASE-T Alliance to gain access to the USXGMII specification.

Performance and Resource Utilization

For transceiver latency, see *UltraScale Architecture GTY Transceivers User Guide* (UG578) and *Versal Adaptive SoC GTY and GTYP Transceivers Architecture Manual* (AM002) for information on the transceiver latency.

For full details about performance and resource utilization, visit the Performance and Utilization web page.

***Note*:** For AMD UltraScale™/UltraScale+ devices, the default configuration for USXGMII IP is with Auto-Negotiation enabled. So the utilization summary page shows with the Auto-Negotiation feature enabled. If Auto-Negotiation is bypassed in the example design, the total number of LUTs for default configuration is approximately 2400. For AMD Versal™ devices, the default configuration for USXGMII IP is with Auto-Negotiation disabled.

Latency

These measurements are for the core only; they do not include the latency through the transceiver. The latency through the transceiver can be obtained from the relevant user guide.

Transmit Path Latency

As measured from the input port `tx_axis_tdata[31:0]` of the transmitter side AXI4-Stream (until that data appears on tx_serdes_data0[31:0] on the transceiver interface), the latency through the core for AMD UltraScale™ devices is thirteen periods of the 312.5 MHz transmit clock.

Receive Path Latency

As measured from the input into the core on rx_serdes_data0[31:0] until the data appears on rx_axis_tdata[31:0] of the receiver side AXI4-Stream interface, the latency through the core in the receive direction is seventeen cycles of the 312.5 MHz receive clock.

Port Descriptions

The following tables list the ports for the USXGMII subsystem. These signals are usually found in the wrapper.v hierarchy. When the AXI register interface is included, some of these ports are accessed by means of the registers instead of the broadside bus.

Transceiver Interface

The following table shows the transceiver I/O ports for the USXGMII subsystem. Refer to Clocking and Resets for details regarding each clock domain.

Table 1: **Transceiver Interface**

Name	Direction	Clock Domain	Description
gtwiz_reset_tx_datapath_*	Input	Async	GT TX reset
gtwiz_reset_rx_datapath_*	Input	Async	GT RX reset

Table 1: **Transceiver Interface** *(cont'd)*

Name	Direction	Clock Domain	Description
gt_refclk_p	Input		Differential reference clock input for the SerDes, positive phase.
gt_refclk_n	Input		Differential reference clock input for the SerDes, negative phase.
gt_rxp_in_*	Input		Serial data from the line; positive phase of the differential signal.
gt_rxn_in_*	Input		Serial data from the line; negative phase of the differential signal.
gt_txp_out_*	Output		Serial data to the line; positive phase of the differential signal.
gt_txn_out_*	Output		Serial data to the line; negative phase of the differential signal.
dclk	Input		Free running clock to GT
gt_loopback_in_*	Input	Async	GT loopback input signal. Refer to the GT user guide. This port is available when the Include GT subcore in core option is selected in the GT Selection and Configuration tab.
gt_loopback_out_*	Output	Async	GT loopback output signal from AXI4-Lite register map. Refer to the GT user guide. This port is available when Include AXI4-Lite is selected from the Configuration tab and the Include GT subcore in the example design option is selected in the GT Selection and Configuration tab.
rxgearboxslip_in_*	Output	RXUSRCLK2	Rxgearboxslip signal from core to GT. This port is available when the Include GT subcore in the example design option is selected in the GT Selection and Configuration tab.
rxdatavalid_out_*	Input	RXUSRCLK2	RX data valid signal from GT to core. This port is available when the Include GT subcore in the example design option is selected in the GT Selection and Configuration tab.
rx_serdes_data_out_*	Input	RXUSRCLK2	RX data signal from GT to core. This port is available when the Include GT subcore in example design option is selected in the GT Selection and Configuration tab.
rxheader_out_*	Input	RXUSRCLK2	RX header signal from GT to core. This port is available when the Include GT subcore in the example design option is selected in the GT Selection and Configuration tab.
rxheadervalid_out_*	Input	RXUSRCLK2	RX header valid signal from GT to core. This port is available when the Include GT subcore in the example design option is selected in the GT Selection and Configuration tab.

Table 1: **Transceiver Interface** *(cont'd)*

Name	Direction	Clock Domain	Description
tx_serdes_data_in_*	Output	TXUSRCLK2	TX data signal from core to GT. This port is available when the Include GT subcore in example design option is selected in the GT Selection and Configuration tab.
txheader_in_*	Output	TXUSRCLK2	TX header signal from core to GT. This port is available when the Include GT subcore in the example design option is selected in the GT Selection and Configuration tab.
gtwiz_reset_clk_freerun_in_0[1]	Input		Free running input clock to core. Versal devices only. This port is available when the Control and Statistics interface is selected from the Configuration tab.
gtwiz_reset_all_in_0[1]	Input		`sys_reset` signal from the user. Versal devices only. This port is available when the Control and Statistics interface is selected from the Configuration tab.
gtpowergood_in_0[1]	Input		Refer to the *UltraScale Architecture GTH Transceivers User Guide* (UG576) the *UltraScale Architecture GTY Transceivers User Guide* (UG578) for the port description.
tx_pma_resetdone_0[1]	Input		TX PMA `resetdone` signal from GT to Core indicates lane0 status. (Versal devices only).
rx_pma_resetdone_0[1]	Input		RX PMA `resetdone` signal from GT to Core indicates lane0 status. (Versal devices only).
mst_tx_resetdone_0[1]	Input		TX master `resetdone` signal from GT to Core indicates lane0 status. (Versal devices only).
mst_rx_resetdone_0[1]	Input		RX master `resetdone` signal from GT to Core indicates lane0 status. (Versal devices only).
rx_resetdone_in_0[1]	Output		RX user ready output signal from Core to example design. (Versal devices only).
tx_resetdone_in_0[1]	Output		TX user ready output signal from Core to example design. (Versal devices only).
mst_tx_reset_0[1]	Output		TX master reset output signal from Core to GT of Lane0. (Versal devices only).
mst_tx_dp_reset_0[1]	Output		TX reset output signal from gt reset ip to the core (Versal devices only).
mst_rx_reset_0[1]	Output		RX master reset output signal from Core to GT of Lane0. (Versal devices only).
mst_rx_dp_reset_0[1]	Output		RX reset output signal from gt reset ip to the core (Versal devices only).
txuserrdy_out_0[1]	Output		TX user ready output signal from Core (reset Interface IP) to GT of Lane0. (Versal devices only).

Table 1: **Transceiver Interface** *(cont'd)*

Name	Direction	Clock Domain	Description
rxuserrdy_out_0[1]	Output		RX user ready output signal from Core (reset Interface IP) to GT of Lane0. (Versal devices only).

Notes:

1. When the USXGMII IP is generated with the GT Wizard Subsystem (gtwiz_versal IP), the GT Reset controller is internal to the GT Wizard Subsystem (gtwiz_versal IP). In this case, the GT Reset controller IP related ports from the USXGMII IP are not available.

AXI4-Stream Interfaces

The USXGMII IP provides a 32-bit AXI4-Stream interface for the TX and RX datapaths. The following subsections describe the clocks and resets and the interface signals in detail.

AXI4-Stream Clocks and Resets

The following table shows the clocks and reset signals of the AXI4-Stream interface.

Table 2: **AXI4-Stream Interface - Clock/Reset Signals**

Name	Direction	Clock Domain	Description
tx_clk_out	Output		Transmit clock for the AXI4-Stream interface. All signals between the USXGMII core and the user-side logic are synchronized to the positive edge of this signal. The AXI4-Stream clock is 312.5 MHz.
rx_clk_out	Output		Receive clock for the AXI4-Stream interface. All signals between the USXGMII IP core and the user-side logic are synchronized to the positive edge of this signal. The AXI4-Stream clock is 312.5 MHz. When the RX FIFO is included, this clock will cease to exist and the RX path will be synchronized to tx_clk_out.
tx_reset	Input	Async	Reset for the TX circuits. This signal is active-High (1=reset) and must be held High until clk is stable. The core handles synchronizing the tx_reset input to the appropriate clock domains within the core.

Table 2: **AXI4-Stream Interface - Clock/Reset Signals** *(cont'd)*

Name	Direction	Clock Domain	Description
rx_reset		Async	Reset for the RX circuits. This signal is active-High (1=reset) and must be held High until clk is stable. The core handles synchronizing the rx_reset input to the appropriate clock domains within the core.

The following table shows the AXI4-Stream transmit interface signals.

Table 3: **Transmit AXI4-Stream Interface**

Name	Direction	Clock Domain	Description
tx_axis_tready	Output	tx_clk_out	AXI4-Stream acknowledge signal to indicate to start the data transfer
tx_axis_tvalid	Input	tx_clk_out	AXI4-Stream Data Valid input
tx_axis_tdata[31:0]	Input	tx_clk_out	AXI4-Stream data (32-bit interface)
tx_axis_tlast	Input	tx_clk_out	AXI4-Stream signal indicating End of Ethernet packet.
tx_axis_tkeep [3:0]	Input	tx_clk_out	AXI4-Stream Data Control (4-bit interface)
tx_axis_tuser	Input	tx_clk_out	AXI4-Stream User sideband interface. Equivalent to the tx_errin signal. 1 indicates a bad packet has been transmitted. 0 indicates a good packet has been transmitted.
tx_unfout	Output	tx_clk_out	TX Underflow. When this signal is High, it indicates that there has not been a sufficient data transfer and the Ethernet interface will underflow. This must not be allowed to occur. You must ensure that you transfer data whenever tx_axis_tready is High until you reach the end of the Ethernet frame.

Transmit AXI4-Stream Interface

The following table shows the AXI4-Stream transmit interface signals.

Table 4: **AXI4-Stream Transmit Interface Signals**

Name	Direction	Clock Domain	Description
tx_axis_tready	Output	tx_clk_out	AXI4-Stream acknowledge signal to indicate to start the data transfer
tx_axis_tvalid	Input	tx_clk_out	AXI4-Stream Data Valid input
tx_axis_tdata[31:0]	Input	tx_clk_out	AXI4-Stream data (32-bit interface)
tx_axis_tlast	Input	tx_clk_out	AXI4-Stream signal indicating End of Ethernet packet.
tx_axis_tkeep [3:0]	Input	tx_clk_out	AXI4-Stream Data Control (4-bit interface)
tx_axis_tuser	Input	tx_clk_out	AXI4-Stream User sideband interface. Equivalent to the `tx_errin` signal. 1 indicates a bad packet has been transmitted. 0 indicates a good packet has been transmitted.
tx_unfout	Output	tx_clk_out	TX Underflow. When this signal is High, it indicates that there has not been a sufficient data transfer and the Ethernet interface will underflow. This must not be allowed to occur. You must ensure that you transfer data whenever `tx_axis_tready` is High until you reach the end of the Ethernet frame.

Data Lane Mapping

For transmit data `tx_axis_tdata[31:0]`, the port is logically divided into lane 0 to lane 3 as shown in the following table.

Table 5: **tx_axis_tdata Lane Mapping**

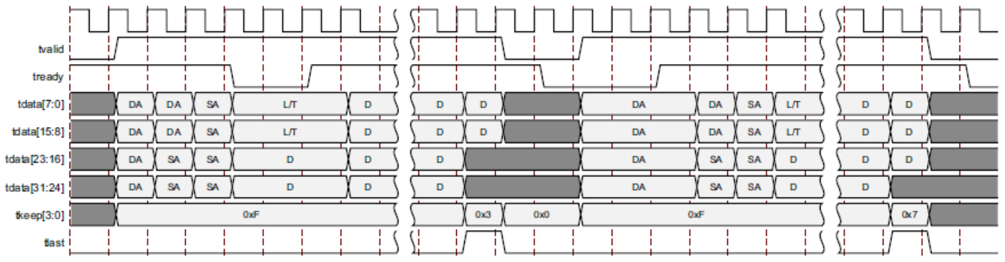
Lane/tx_axis_tkeep	tx_axis_tdata[31:0] bits
0	7:0
1	15:8
2	23:16
3	31:24

Normal Frame Transmission

The timing of a normal frame transfer is shown in the following figure. When the client wants to transmit a frame, it asserts `tx_axis_tvalid` and places the data and control in `tx_axis_tdata` and `tx_axis_tkeep` in the same clock cycle. When this data is accepted by the core, indicated by `tx_axis_tready` being asserted, the client must provide the next cycle of data. If `tx_axis_tready` is not asserted by the core, the client must hold the current valid data value until it is. The end of the packet is in dictated to the core by `tx_axis_tlast` asserted for one cycle. The bits of `tx_axis_tkeep` are set appropriately to indicate the number of valid bytes in the final data transfer. `tx_axis_tuser` is also asserted to indicate a bad packet.

After `tx_axis_tlast` is deasserted, any data and control is deemed invalid until `tx_axis_tvalid` is next asserted.

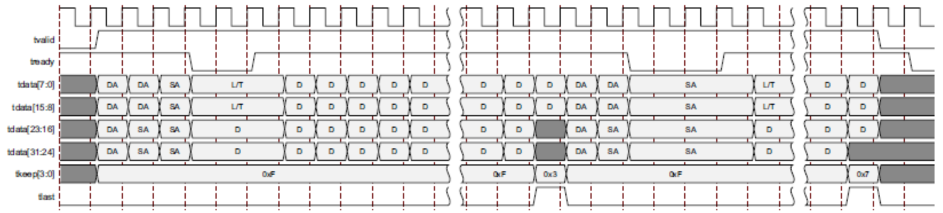
Figure 4: **Normal Frame Transfer - 32 bits**



Back to Back Continuous Transfers

Continuous data transfer on the transmit AXI4-Stream interface is possible, as the signal `tx_axis_tvalid` can remain continuously High, with packet boundaries defined solely by `tx_axis_tlast` asserted for the end of the Ethernet packet. However, the core can de-assert the `tx_axis_tready` acknowledgement signal to throttle the client data rate as required. See the following figure for details.

Figure 5: **Back-to-Back Continuous Transfer on Transmit Client Interface - 32-bit**



Aborting a Transmission

The aborted transfer of a packet on the client interface is called an underrun. This can happen if a FIFO in the AXI Transmit client interface empties before a frame is completed.

This is indicated to the core in one of two ways.

- An explicit error in which a frame transfer is aborted by asserting `tx_axis_tuser` High while `tx_axis_tlast` is High. See the figure on Frame Reception with Errors - 32 Bits.
- An implicit underrun in which a frame transfer is aborted by de-asserting `tx_axis_tvalid` without asserting `tx_axis_tlast`.

Receive AXI4-Stream Interface

The following table shows the AXI4-Stream receive interface signals.

Table 6: **AXI4-Stream receive interface signals.**

Name	Direction	Clock Domain	Description
rx_axis_tvalid	Output	rx_clk_out	AXI4-Stream Data Valid
rx_axis_tdata[31:0]	Output	rx_clk_out	AXI4-Stream Data to upper layer
rx_axis_tlast	Output	rx_clk_out	AXI4-Stream signal indicating an end of packet
rx_axis_tkeep[3:0]	Output	rx_clk_out	AXI4-Stream Data control to upper layer
rx_axis_tuser	Output	rx_clk_out	AXI4-Stream User Sideband interface 1 indicates a bad packet has been received 0 indicates a good packet has been received

Data Lane Mapping

For receive data `rx_axis_tdata`, the port is divided into lane 0 to lane 3. See the following table.

Table 7: **rx_axis_tdata Lane Mapping**

Lane/rx_axis_tkeep	rx_axis_tdata bits
0	7:0
1	15:8
2	23:16
3	31:24

Normal Frame Reception

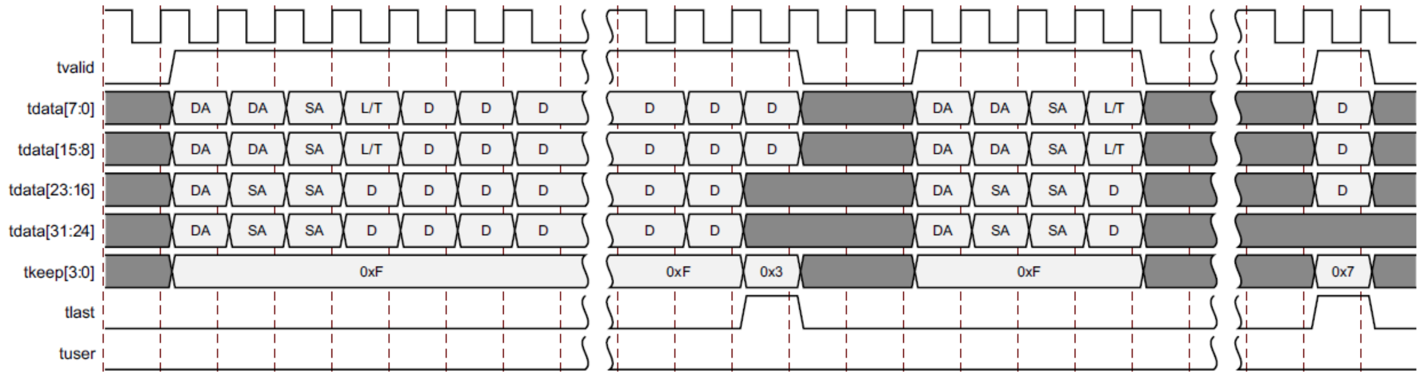
The timing of a normal inbound frame transfer is represented in the following figure. The client must be prepared to accept data at any time; there is no buffering within the core to allow for latency in the receive client. When frame reception begins, data is transferred on consecutive clock cycles to the receive client.

During frame reception, `rx_axis_tvalid` is asserted to indicate that valid frame data is being transferred to the client on `rx_axis_tdata`. All bytes are always valid throughout the frame, as indicated by all `rx_axis_tkeep` bits being set to 1, except during the final transfer of the frame when `rx_axis_tlast` is asserted. During this final transfer of data for a frame, `rx_axis_tkeep` bits indicate the final valid bytes of the frame using the mapping from above.

The valid bytes of the final transfer always lead out from `rx_axis_tdata[7:0]` `(rx_axis_tkeep[0])` because Ethernet frame data is continuous and is received least significant byte first.

The `rx_axis_tlast` is asserted and `rx_axis_tuser` is deasserted, along with the final bytes of the transfer, only after all the frame checks are completed. This is after the frame check sequence (FCS) field has been received. The core keeps the `rx_axis_tuser` signal deasserted to indicate that the frame was successfully received and that the frame must be analyzed by the client. This is also the end of the packet signaled by `rx_axis_tlast` asserted for one cycle.

Figure 6: **Normal Frame Reception – 32 Bits**



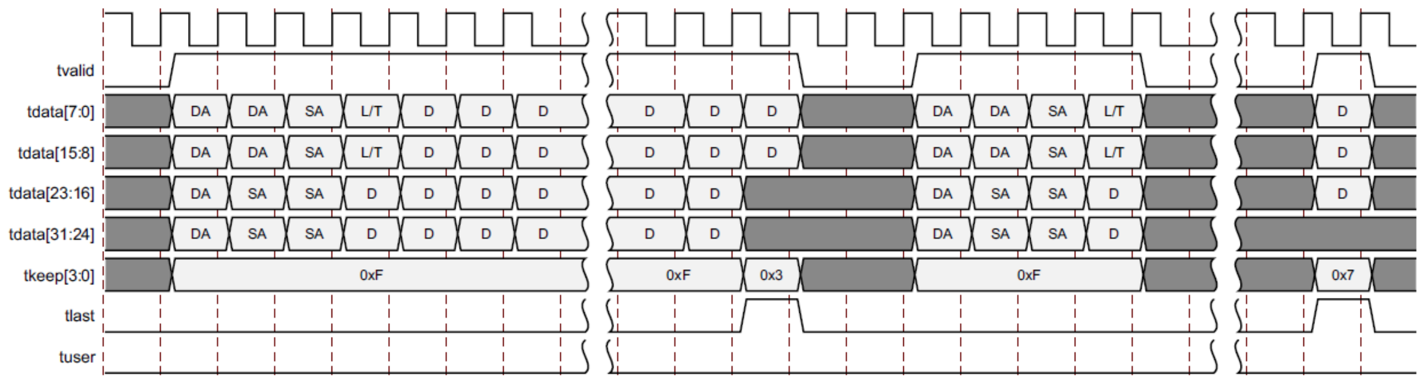
Frame Reception with Errors

The case of an unsuccessful frame reception (for example, a runt frame or a frame with an incorrect FCS) is shown in the following figure. In this case the bad frame is received and the signal `rx_axis_tuser` is asserted to the client at the end of the frame. It is the responsibility of the client to drop the data already transferred for this frame.

The following conditions cause the assertion of `rx_axis_tlast` along with `rx_axis_tuser` = 1 signifying a bad frame.

- FCS errors occur.
- Packets are shorter than 64 bytes (undersize or fragment frames).
- Frames of length greater than the maximum transmission unit (MTU) size programmed are received.
- Any control frame that is received is not exactly the minimum frame length.
- The XGMII data stream contains error codes.

Figure 7: **Frame Reception with Errors - 32 Bits**



AXI4-Stream Control and Status Ports

Table 8: **AXI4-Stream Interface – TX Path Control/Status Signals**

Name	Direction	Clock Domain	Description
ctl_tx_enable	Input	tx_clk_out	TX Enable. When sampled as a 1, this signal is used to enable the transmission of data. When sampled as a 0, only IDLEs are transmitted by the core. This input must not be set to 1 until the receiver it is sending data to is fully synchronized and ready to receive data. (that is, the receiver on the link partner is not sending a remote fault condition.) Otherwise, loss of data can occur. If this signal is set to 0 while a packet is being transmitted, the current packet transmission is completed and then the core stops transmitting any more packets.
ctl_tx_custom_preamble_enable	Input	tx_clk_out	When asserted, this signal enables the use of tx_preamblein as a custom preamble instead of inserting a standard preamble.
tx_preamblein[55:0]	Input	tx_clk_out	This is the custom preamble which is a separate input port rather than being in-line with the data. It needs to be valid during the start of the packet.

Table 8: **AXI4-Stream Interface – TX Path Control/Status Signals** *(cont'd)*

Name	Direction	Clock Domain	Description
ctl_tx_fcs_ins_enable	Input	tx_clk_out	Enable FCS insertion by the TX core. If set to 0, the core does not add FCS to the packet. If set to 1, the core calculates and adds FCS to the packet. This input cannot be dynamically changed between packets.
ctl_tx_send_lfi	Input	tx_clk_out	Transmit Local Fault Indication (LFI) code word. Takes precedence over Remote Fault Indication (RFI).
ctl_tx_send_rfi	Input	tx_clk_out	Transmit Remote Fault Indication (RFI) code word. If sampled as a 1, the TX path transmits only RFI code words. This input must be set to 1 until the RX path is fully synchronized and is ready to accept data from the link partner.
ctl_tx_send_idle	Input	tx_clk_out	Transmit IDLE code words. If sampled as a 1, the TX path only transmits IDLE code words. This input must be set to 1 when the partner is sending RFI code words.
ctl_tx_send_umii_lfi	Input	tx_clk_out	If set to 1, the TX MAC/PCS sends LFI code words onto the line.
ctl_tx_send_umii_rfi	Input	tx_clk_out	If UMII LFI code set to 1, the TX MAC/PCS sends UMII RFI code words onto the line.
ctl_tx_ignore_fcs	Input	tx_clk_out	Enable FCS error checking at the AXI4-Stream interface by the TX core. This input only has effect when `ctl_tx_fcs_ins_enable` is Low. If set to 0 and a packet with bad FCS is being transmitted, it is not binned as good. If set to 1, a packet with bad FCS is binned as good. The error is flagged on the signals stat_tx_bad_fcs and stomped_fcs and the packet is transmitted as it was received. Statistics are reported as if there was no FCS error.
ctl_usxgmii_rate[2:0]	Input	static	User selectable USXGMII rate, valid when `ctl_umii_an_bypass` = 1; otherwise the USXGMII rate is determined through auto-negotiation. 3’b000 = 10 Mb/s 3’b001 = 100 Mb/s 3’b010 = 1 Gb/s 3’b011 = 10 Gb/s 3’b100 = 2.5 Gb/s 3’b101 = 5 Gb/s

Table 8: **AXI4-Stream Interface – TX Path Control/Status Signals** *(cont'd)*

Name	Direction	Clock Domain	Description
ctl_umii_an_mr_adv_ability[15:0]	Input	tx_clk_out	This bus must be driven with the desired link advertised ability for the Auto-Negotiation function, that maps to the UsxgmiiChannelInfo message's config field. [15] = Link Status [14:13] = 2'b00 [12] = Duplex Mode. 1 = Full Duplex, 0=Half Duplex [11:9] = Speed; for more information, refer table f N-base-T_usxgmii rev1.4 [8] = 1'b0 (EEE capability; Not supported.) [7] = 1'b0 (EEE clock stop capability; Not supported) [6:1] = 0x0 Reserved [0] = USXGMII
ctl_umii_an_mr_an_enable	Input	tx_clk_out	Refer to Auto-Negotiation for the use of these signals in the auto-negotiation operation.
ctl_umii_an_mr_restart_an	Input	tx_clk_out	
ctl_umii_an_mr_main_reset	Input	tx_clk_out	
ctl_umii_an_link_timer_config[3:0]	Input	tx_clk_out	
ctl_umii_an_bypass	Input	tx_clk_out	
stat_umii_an_mr_an_complete	Output	rx_serdes_clk	USXGMII auto-negotiation complete status signals.
stat_umii_an_mr_lp_adv_ability[15:0]	Output	rx_serdes_clk	Auto-negotiation ability advertisement from Link Partner.
stat_umii_an_mr_np_able	Output	rx_serdes_clk	Indicates next page capability. The design not supporting next page capability. It is set to 0.

Table 9: **AXI4-Stream Interface – RX Path Control/Status Signals**

Name	Direction	Clock Domain	Description
ctl_rx_enable	Input	rx_clk_out	RX enable. For normal operation this input must be set to 1. When set to 0, after the RX completes the reception of the current packet (if any), it stops receiving packets by keeping the PCS from decoding incoming data. In this mode, there are no statistics reported and the AXI4-Stream interface is idle.
ctl_rx_custom_preamble_enable	Input	rx_clk_out	When asserted, this signal causes the side band of a packet presented on the AXI4-Stream to be the preamble as it appears on the line.
rx_preambleout[55:0]	Output	rx_clk_out	This is the preamble, and now a separate output instead of inline with data.

Table 9: **AXI4-Stream Interface – RX Path Control/Status Signals** *(cont'd)*

Name	Direction	Clock Domain	Description
ctl_rx_delete_fcs	Input	rx_clk_out	Enable FCS removal by the RX core. If set to 0, the core does not remove the FCS of the incoming packet. If set to 1, the core deletes the FCS to the received packet. FCS is not deleted for packets that are less than 5 bytes. This input only needs to be changed while the corresponding reset input is asserted.
ctl_rx_ignore_fcs	Input	rx_clk_out	Enable FCS error checking at the AXI4-Stream interface by the RX core. If set to 0, a packet received with an FCS error is indicated as an errored frame (rx_axis_tuser=1 when rx_axis_tlast=1). If set to 1, the core does not flag an FCS error at the AXI4-Stream Interface. The statistics are reported as if the packet is good. The signal stat_rx_bad_fcs, however, reports the error.
ctl_rx_max_packet_len[14:0]	Input	rx_clk_out	Any packet longer than this value is considered to be oversized. If a packet has a size greater than this value, it is truncated to this value and the `rx_axis_tuser` signal is asserted along with the `rx_axis_tlast` signal. ctl_rx_max_packet_len[14] is reserved and must be set to 0.
ctl_rx_min_packet_len[7:0]	Input	rx_clk_out	Any packet shorter than this value is considered to be undersized. If a packet has a size shorter than this value, the rx_axis_tuser signal is asserted along with the rx_axis_tlast signal.
ctl_rx_check_sfd	Input	rx_clk_out	When asserted, this input causes the MAC to check the Start of Frame Delimiter (SFD) of the received frame.
ctl_rx_check_preamble	Input	rx_clk_out	When asserted, this input causes the MAC to check the preamble of the received frame. When rx custom preamble is enabled, both ctl_rx_check_sfd, crl_rx_check_preamble are required to be disabled.
ctl_rx_force_resync	Input	rx_clk_out	RX force resynchronization input. This signal is used to force the RX path to reset and resynchronize. A value of 1 forces the reset operation. A value of 0 allows normal operation. This must normally be Low and must only be pulsed. (1 cycle minimum pulse)
stat_rx_local_fault	Output	rx_clk_out	This output is High when `stat_rx_internal_local_fault` or `stat_rx_received_local_fault` is asserted. This output is level sensitive.
stat_rx_umii_local_fault	Output	rx_clk_out	USXGMII received UMII local fault

Table 9: **AXI4-Stream Interface – RX Path Control/Status Signals** *(cont'd)*

Name	Direction	Clock Domain	Description
stat_rx_remote_fault	Output	rx_clk_out	Remote fault indication status. If this bit is sampled as 1, indicates a remote fault condition was detected. If this bit is sampled as 0, remote fault condition does not exist. This output is level sensitive.
stat_rx_umii_remote_fault	Output	rx_clk_out	USXGMII has received UMII Remote fault.
stat_rx_internal_local_fault	Output	rx_clk_out	High when an internal local fault is generated due to any of the following: test pattern generation or high bit error rate (BER).Remains High as long as the fault condition persists.
stat_rx_received_local_fault	Output	rx_clk_out	High when enough local fault words are received from the link partner to trigger a fault condition as specified by the IEEE fault state machine. Remains High as long as the fault condition persists.
stat_rx_block_lock	Output	rx_clk_out	Block lock status. A value of 1 indicates that block lock is achieved as defined in Clause 49.2.14 and management data input/output (MDIO) register 3.32.0. This output is level sensitive.
stat_rx_framing_err_valid	Output	rx_clk_out	Valid indicator for `stat_rx_framing_err`. When sampled as a 1, the value on `stat_rx_framing_err` is valid.
stat_rx_framing_err[2:0]	Output	rx_clk_out	The RX sync header bits framing error is a bus that indicates how many sync header errors were received. The value of the bus is only valid when the `stat_rx_framing_err_valid` is a 1. The values can be updated at any time and are intended to be used as increment values for sync header error counters.
stat_rx_hi_ber	Output	rx_clk_out	High Bit Error Rate (BER) indicator. When set to 1, the BER is too high as defined by IEEE Std. 802.3. Corresponds to management data input/output (MDIO) register bit 3.32.1as defined in Clause 49.2.14. This output is level sensitive.

Miscellaneous Status/Control Signals

The following table shows the miscellaneous status and control signals.

Table 10: **Miscellaneous Status/Control Signals**

Name	Direction	Clock Domain	Description
ctl_tx_test_pattern_enable	Input	tx_clk_out	Test pattern generation enable for the TX core. A value of 1 enables test mode. Corresponds to MDIO register bit 3.42.3 as defined in clause 45. Takes second precedence.

Table 10: **Miscellaneous Status/Control Signals** *(cont'd)*

Name	Direction	Clock Domain	Description
ctl_tx_test_pattern_select	Input	tx_clk_out	Corresponds to MDIO register bit 3.42.1 as defined in Clause 45
ctl_tx_data_pattern_select	Input	tx_clk_out	Corresponds to MDIO register bit 3.42.0 as defined in Clause 45
ctl_tx_test_pattern_seed_a[57:0]	Input	tx_clk_out	Corresponds to MDIO registers 3.34 through to 3.37 as defined in Clause 45
ctl_tx_test_pattern_seed_b[57:0]	Input	tx_clk_out	Corresponds to MDIO registers 3.38 through to 3.41 as defined in Clause 45
ctl_rx_test_pattern_enable	Input	rx_clk_out	Test pattern enable for the RX core. A value of 1 enables test mode. Corresponds to MDIO register bit 3.42.2 as defined in Clause 45. Takes second precedence.
ctl_rx_data_pattern_select	Input	rx_clk_out	Corresponds to MDIO register bit 3.42.0 as defined in Clause 45.
stat_rx_test_pattern_mismatch	Output	rx_clk_out	Test pattern mismatch increment. A non-zero value in any cycle indicates how many mismatches occurred for the test pattern in the RX core. This output is only active when `ctl_rx_test_pattern` is set to a 1. This output can be used to generate MDIO register as defined in Clause 45. This output is pulsed for one clock cycle.
ctl_local_loopback	Input	Async	Sets GT in near-end PMA loopback
ctl_rx_process_lfi	Input	rx_clk_out	When this input is set to 1, the RX core expects and processes LF control codes coming in from the transceiver. When set to 0, the RX core ignores LF control codes coming in from the transceiver.
stat_rx_valid_ctrl_code	Output	rx_clk_out	Indicates that a PCS block with a valid control code was received.

Statistics Interface Ports

The following tables shows the statistics interface ports for the RX and TX paths respectively.

Table 11: **Statistics Interface Ports – RX**

Name	Direction	Clock Domain	Description
stat_rx_bad_code	Output	rx_clk_out	Increment for 64B/66B code violations. This signal indicates that the RX PCS receive state machine is in the RX_E state as specified by IEEE Std 802.3. This output can be used to generate MDIO register as defined in Clause 45.
stat_rx_total_packets[1:0]	Output	rx_clk_out	Increment for the total number of packets received.

Table 11: **Statistics Interface Ports – RX** *(cont'd)*

Name	Direction	Clock Domain	Description
stat_rx_total_good_packets	Output	rx_clk_out	Increment for the total number of good packets received. This value is non-zero only when a packet is received completely and contains no errors.
stat_rx_total_bytes[3:0]	Output	rx_clk_out	Increment for the total number of bytes received.
stat_rx_total_good_bytes[13:0]	Output	rx_clk_out	Increment for the total number of good bytes received. This value is non-zero only when a packet is received completely and contains no errors.
stat_rx_packet_small	Output	rx_clk_out	Increment for all packets that are less than 64 bytes long. Packets that are less than 64 bytes are dropped.
stat_rx_jabber	Output	rx_clk_out	Increment for packets longer than ctl_rx_max_packet_len with bad FCS.
stat_rx_packet_large	Output	rx_clk_out	Increment for all packets that are more than 9215 bytes long.
stat_rx_oversize	Output	rx_clk_out	Increment for packets longer than ctl_rx_max_packet_len with good FCS.
stat_rx_undersize	Output	rx_clk_out	Increment for packets shorter than ctl_rx_min_packet_len with good FCS.
stat_rx_toolong	Output	rx_clk_out	Increment for packets longer than ctl_rx_max_packet_len with good and bad FCS.
stat_rx_fragment	Output	rx_clk_out	Increment for packets shorter than ctl_rx_min_packet_len with bad FCS.
stat_rx_packet_64_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain 64 bytes.
stat_rx_packet_65_127_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 65 and 127 bytes.
stat_rx_packet_128_255_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 128 and 255 bytes.
stat_rx_packet_256_511_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 256 and 511 bytes.
stat_rx_packet_512_1023_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 512 and 1023 bytes.
stat_rx_packet_1024_1518_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 1024 and 1518 bytes.
stat_rx_packet_1519_1522_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 1519 and 1522 bytes.
stat_rx_packet_1523_1548_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 1523 and 1548 bytes.
stat_rx_packet_1549_2047_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 1549 and 2047 bytes.
stat_rx_packet_2048_4095_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 2048 and 4095 bytes.
stat_rx_packet_4096_8191_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 4096 and 8191 bytes.
stat_rx_packet_8192_9215_bytes	Output	rx_clk_out	Increment for good and bad packets received that contain between 8192 and 9215 bytes.

Table 11: **Statistics Interface Ports – RX** *(cont'd)*

Name	Direction	Clock Domain	Description
stat_rx_bad_fcs [1:0]	Output	rx_clk_out	When this signal is positive, it indicates that the error detection logic has identified mismatches between the expected and received value of CRC32 in the received packet. When a CRC error is detected, the received packet is marked as containing an error and is sent with rx_axis_tuser asserted for the last transfer (the cycle with rx_axis_last asserted), unless ctl_rx_ignore_fcs is asserted. This signal is asserted for one clock cycle for each CRC32 error detected.
stat_rx_packet_bad_fcs	Output	rx_clk_out	Increment for packets between 64 and ctl_rx_max_packet_len bytes that have Frame Check Sequence (FCS) errors.
stat_rx_stomped_fcs [1:0]	Output	rx_clk_out	Stomped FCS indicator. The value on this bus indicates the packets received with a stomped FCS. A stomped FCS is defined as the bitwise inverse of the expected good FCS. This output is pulsed for one clock cycle to indicate the stomped condition. Pulses can occur in back-to-back cycles.
stat_rx_unicast	Output	rx_clk_out	Increment for good unicast packets.
stat_rx_multicast	Output	rx_clk_out	Increment for good multicast packets.
stat_rx_broadcast	Output	rx_clk_out	Increment for good broadcast packets.
stat_rx_vlan	Output	rx_clk_out	Increment for good 802.1Q tagged VLAN packets.
stat_rx_pause	Output	rx_clk_out	Increment for 802.3x MAC Pause Packet with good FCS.
stat_rx_user_pause	Output	rx_clk_out	Increment for priority based pause packets with good FCS.
stat_rx_inrangeerr	Output	rx_clk_out	Increment for packets with Length field error but with good FCS.
stat_rx_bad_preamble	Output	rx_clk_out	Increment for packets received with bad preamble. This signal indicates if the Ethernet packet received was preceded by a valid preamble. A value of 1 indicates that an invalid preamble was received.
stat_rx_bad_sfd	Output	rx_clk_out	Increment for packets received with bad SFD. This signal indicates if the Ethernet packet received was preceded by a valid SFD. A value of 1 indicates that an invalid SFD was received.
stat_rx_got_signal_os	Output	rx_clk_out	Signal OS indication. If this bit is sampled as a 1, it indicates that a signal OS word was received. Signal OS cannot be received in an Ethernet network.
stat_rx_truncated	Output	rx_clk_out	Packet truncation indicator. A value of 1 indicates that the current packet in flight is truncated due to its length exceeding ctl_rx_max_packet_len[14:0]. This output is pulsed for one clock cycle to indicate the truncated condition. Pulses can occur in back to back cycles.

Table 12: **Statistics Interface Ports - TX Path**

Name	Direction	Clock Domain	Description
stat_tx_total_packets	Output	tx_clk_out	Increment for total number of packets transmitted.
stat_tx_total_bytes[2:0]	Output	tx_clk_out	Increment for total number of bytes transmitted.
stat_tx_total_good_packets	Output	tx_clk_out	Increment for the total number of good packets transmitted.
stat_tx_total_good_bytes[13:0]	Output	tx_clk_out	Increment for the total number of good bytes transmitted. This is signal is non-zero only when a packet is transmitted completely and contains no errors.
stat_tx_packet_64_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain 64 bytes.
stat_tx_packet_65_127_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 65 and 127 bytes.
stat_tx_packet_128_255_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 128 and 255 bytes.
stat_tx_packet_256_511_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 256 and 511 bytes.
stat_tx_packet_512_1023_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 512 and 1023 bytes.
stat_tx_packet_1024_1518_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 1024 and 1518 bytes.
stat_tx_packet_1519_1522_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 1519 and 1522 bytes.
stat_tx_packet_1523_1548_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 1523 and 1548 bytes.
stat_tx_packet_1549_2047_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 1549 and 2047 bytes.
stat_tx_packet_2048_4095_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 2048 and 4095 bytes.
stat_tx_packet_4096_8191_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 4096 and 8191 bytes.
stat_tx_packet_8192_9215_bytes	Output	tx_clk_out	Increment for good and bad packets transmitted that contain between 8192 and 9215 bytes.
stat_tx_packet_small	Output	tx_clk_out	Increment for all packets that are less than 64 bytes long.
stat_tx_packet_large	Output	tx_clk_out	Increment for all packets that are more than 9215 bytes long.
stat_tx_unicast	Output	tx_clk_out	Increment for good unicast packets.
stat_tx_multicast	Output	tx_clk_out	Increment for good multicast packets.
stat_tx_broadcast	Output	tx_clk_out	Increment for good broadcast packets.
stat_tx_vlan	Output	tx_clk_out	Increment for good 802.1Q tagged VLAN packets.
stat_tx_pause	Output	tx_clk_out	Increment for 802.3x MAC Pause Packet with good FCS.
stat_tx_user_pause	Output	tx_clk_out	Increment for Priority based pause packets with good FCS.
stat_tx_bad_fcs	Output	tx_clk_out	Increment for packets greater than 64 bytes that have FCS errors.

Table 12: **Statistics Interface Ports - TX Path** *(cont'd)*

Name	Direction	Clock Domain	Description
stat_tx_frame_error	Output	tx_clk_out	Increment for packets with tx_axis_tuser set to indicate an End of Packet (EOP) abort.
stat_tx_local_fault	Output	tx_clk_out	A value of 1 indicates the transmit encoder state machine is in the TX_INIT state. This output is level sensitive.

Pause Interface

The following two tables shows the Pause interface I/O ports.

Table 13: **Pause Interface - Control Ports**

Name	Direction	Clock Domain	Description
ctl_rx_pause_enable[8:0]	Input	rx_clk_out	RX pause enable signal. This input is used to enable the processing of the pause quanta for the corresponding priority. This signal only affects the RX user interface and not the pause processing logic
ctl_tx_pause_enable[8:0]	Input	tx_clk_out	TX pause enable signal. This input is used to enable the processing of the pause quanta for the corresponding priority. This signal gates transmission of pause packets.

Table 14: **Pause Interface - TX Path**

Name	Direction	Clock Domain	Description
ctl_tx_pause_req[8:0]	Input	tx_clk_out	If a bit of this bus is set to 1, the core transmits a pause packet using the associated quanta value on the ctl_tx_pause_quanta[8:0][15:0] bus. If bit[8] is set to 1, a global pause packet is transmitted. All other bits cause a priority pause packet to be transmitted.

Table 14: **Pause Interface - TX Path** *(cont'd)*

Name	Direction	Clock Domain	Description
ctl_tx_resend_pause	Input	tx_clk_out	Re-transmit pending pause packets. When this input is sampled as 1, all pending pause packets are retransmitted as soon as possible (that is, after the current packet in flight is completed) and the retransmit counters are reset. This input must be pulsed to 1 for one cycle at a time.
ctl_tx_pause_quanta[8:0][15:0]	Input	tx_clk_out	These nine buses indicate the quanta to be transmitted for each of the eight priorities in priority based and global pause operations. The value for ctl_tx_pause_quanta[8] is used for global pause operation. All other values are used for priority based pause operation.
ctl_tx_pause_refresh_timer[8:0][15:0]	Input	tx_clk_out	These nine buses set the retransmission time of pause packets for each of the eight priorities in priority based pause operation and the global pause operation. The values for ctl_tx_pause_refresh_timer[8] are used for global pause operation. All other values are used for priority pause operation.
ctl_tx_da_gpp[47:0]	Input	tx_clk_out	Destination address for transmitting global pause packets.
ctl_tx_sa_gpp[47:0]	Input	tx_clk_out	Source address for transmitting global pause packets.
ctl_tx_ethertype_gpp[15:0]	Input	tx_clk_out	Ethertype for transmitting global pause packets.
ctl_tx_opcode_gpp[15:0]	Input	tx_clk_out	Opcode for transmitting global pause packets.
ctl_tx_da_ppp[47:0]	Input	tx_clk_out	Destination address for transmitting priority pause packets.
ctl_tx_sa_ppp[47:0]	Input	tx_clk_out	Source address for transmitting priority pause packets.
ctl_tx_ethertype_ppp[15:0]	Input	tx_clk_out	Ethertype for transmitting priority pause packets.
ctl_tx_opcode_ppp[15:0]	Input	tx_clk_out	Opcode for transmitting priority pause packets.

Table 14: **Pause Interface - TX Path** *(cont'd)*

Name	Direction	Clock Domain	Description
stat_tx_pause_valid[8:0]	Output	tx_clk_out	If a bit of this bus is set to 1, the core has transmitted a pause packets. If bit [8] is set to 1, a global pause packet is transmitted. All other bits cause a priority pause packet to be transmitted.

Table 15: **Pause Interface - RX**

Name	Direction	Clock Domain	Description
ctl_rx_pause_ack[8:0]	Input	rx_clk_out	Pause acknowledge signal. This bus is used to acknowledge the receipt of the pause frame from the user logic.
ctl_rx_check_ack	Input	rx_clk_out	Wait for acknowledge. IF this input is set to 1, the core uses the ctl_rx_pause_ack[8:0] bus for pause processing. If this input is set to 0, ctl_rx_pause_ack[8:0] is not used.
ctl_rx_enable_gcp	Input	rx_clk_out	A value of 1 enables global control packet processing.
ctl_rx_check_mcast_gcp	Input	rx_clk_out	A value of 1 enables global control multicast destination address processing.
ctl_rx_check_ucast_gcp	Input	rx_clk_out	A value of 1 enables global control unicast destination address processing.
ctl_rx_pause_da_ucast[47:0]	Input	rx_clk_out	Unicast destination address for pause processing.
ctl_rx_check_sa_gcp	Input	rx_clk_out	A value of 1 enables global control source address processing.
ctl_rx_pause_sa[47:0]	Input	rx_clk_out	Source address for pause processing.
ctl_rx_check_etype_gcp	Input	rx_clk_out	A value of 1 enables global control ethertype processing.
ctl_rx_etype_gcp[15:0]	Input	rx_clk_out	Ethertype field for global control processing
ctl_rx_check_opcode_gcp	Input	rx_clk_out	A value of 1 enables global control opcode processing.
ctl_rx_opcode_min_gcp[15:0]	Input	rx_clk_out	Minimum global control opcode value.
ctl_rx_opcode_max_gcp[15:0]	Input	rx_clk_out	Maximum global control opcode value.
ctl_rx_enable_pcp	Input	rx_clk_out	A value of 1 enables priority control packet processing.

Table 15: **Pause Interface - RX** *(cont'd)*

Name	Direction	Clock Domain	Description
ctl_rx_check_mcast_pcp	Input	rx_clk_out	A value of 1 enables priority control multicast destination address processing.
ctl_rx_check_ucast_pcp	Input	rx_clk_out	A value of 1 enables priority control unicast destination address processing.
ctl_rx_pause_da_mcast[47:0]	Input	rx_clk_out	Multicast destination address for pause processing.
ctl_rx_check_sa_pcp	Input	rx_clk_out	A value of 1 enables priority control source address processing.
ctl_rx_check_etype_pcp	Input	rx_clk_out	A value of 1 enables priority control ethertype processing.
ctl_rx_etype_pcp[15:0]	Input	rx_clk_out	Ethertype field for priority control processing.
ctl_rx_check_opcode_pcp	Input	rx_clk_out	A value of 1 enables priority control opcode processing.
ctl_rx_opcode_min_pcp[15:0]	Input	rx_clk_out	Minimum priority control opcode value.
ctl_rx_opcode_max_pcp[15:0]	Input	rx_clk_out	Maximum priority control opcode value.
ctl_rx_enable_gpp	Input	rx_clk_out	A value of 1 enables global pause packet processing.
ctl_rx_check_mcast_gpp	Input	rx_clk_out	A value of 1 enables global pause multicast destination address processing.
ctl_rx_check_ucast_gpp	Input	rx_clk_out	A value of 1 enables global pause unicast destination address processing.
ctl_rx_check_sa_gpp	Input	rx_clk_out	A value of 1 enables global pause source address processing.
ctl_rx_check_etype_gpp	Input	rx_clk_out	A value of 1 enables global pause ethertype processing.
ctl_rx_etype_gpp[15:0]	Input	rx_clk_out	Ethertype field for global pause processing.
ctl_rx_check_opcode_gpp	Input	rx_clk_out	A value of 1 enables global pause opcode processing.
ctl_rx_opcode_gpp[15:0]	Input	rx_clk_out	Global pause opcode value.
ctl_rx_enable_ppp	Input	rx_clk_out	A value of 1 enables priority pause packet processing.
ctl_rx_check_mcast_ppp	Input	rx_clk_out	A value of 1 enables priority pause multicast destination address processing.
ctl_rx_check_ucast_ppp	Input	rx_clk_out	A value of 1 enables priority pause unicast destination address processing.

Table 15: **Pause Interface - RX** *(cont'd)*

Name	Direction	Clock Domain	Description
ctl_rx_check_sa_ppp	Input	rx_clk_out	A value of 1 enables priority pause source address processing.
ctl_rx_check_etype_ppp	Input	rx_clk_out	A value of 1 enables priority pause ethertype processing.
ctl_rx_etype_ppp[15:0]	Input	rx_clk_out	Ethertype field for priority pause processing.
ctl_rx_check_opcode_ppp	Input	rx_clk_out	A value of 1 enables priority pause opcode processing.
ctl_rx_opcode_ppp[15:0]	Input	rx_clk_out	Priority pause opcode value.
ctl_rx_forward_control	Input	rx_clk_out	A value of 1 indicates that the core forwards control packets. A value of 0 causes core to drop control packets.
stat_rx_pause_valid [8:0]	Output	rx_clk_out	Indicates that a pause packet was received and the associated quanta on the stat_rx_pause_quanta[8:0][15:0] bus is valid and must be used for pause processing. If an 802.3x MAC Pause packet is received, bit [8] is set to 1.
stat_rx_pause_quanta[8:0][15:0]	Output	rx_clk_out	These nine buses indicate the quanta received for each of the eight priorities in priority based pause operation and global pause operation. If an 802.3x MAC Pause packet is received, the quanta is placed in value[8].
stat_rx_pause_req [8:0]	Output	rx_clk_out	Pause request signal. When the RX receives a valid pause frame, it sets the corresponding bit of this bus to 1 and keep it at 1 until the pause packet has been processed.

Register Space

The USXGMII IP Core can be optionally configured with AXI4-Lite registers to access the configuration and status signals.

AXI4-Lite Ports

Table 16: **AXI4-Lite Ports**

Name	Direction	Clock Domain	Description
s_axi_aclk	Input		AXI4-Lite Clock
s_axi_aresetn	Input		Asynchronous active-Low Reset
s_axi_awaddr[31:0]	Input	s_axi_aclk	Write Address Bus
s_axi_awvalid	Input	s_axi_aclk	Write Address Valid
s_axi_awready	Output	s_axi_aclk	Write address acknowledge
s_axi_wdata[31:0]	Input	s_axi_aclk	Write Data Bus
s_axi_wstrb[3:0]	Input	s_axi_aclk	Strobe signal for the data bus byte lane
s_axi_wvalid	Input	s_axi_aclk	Write data valid
s_axi_wready	Output	s_axi_aclk	Write data acknowledge
s_axi_bresp[1:0]	Output	s_axi_aclk	Write transaction response
s_axi_bvalid	Output	s_axi_aclk	Write response valid
s_axi_bready	Input	s_axi_aclk	Write response acknowledge
s_axi_araddr[31:0]	Input	s_axi_aclk	Read address bus
s_axi_arvalid	Input	s_axi_aclk	Read address valid
s_axi_arready	Output	s_axi_aclk	Read address acknowledge
s_axi_rdata[31:0]	Output	s_axi_aclk	Read data output
s_axi_rresp[1:0]	Output	s_axi_aclk	Read data response
s_axi_rvalid	Output	s_axi_aclk	Read data/response valid
s_axi_rready	Input	s_axi_aclk	Read data acknowledge
pm_tick	Input	s_axi_aclk	Top level signal to read statistics counters; requires MODE_REG[30] (that is, tick_reg_mode_sel) to be set to 0.

Additional information for the operation of the AXI4 bus is found in the AMD AXI Memory-Mapped Protocol version 1.8.

As noted previously, the top-level signal pm_tick can be used to read statistics counters instead of the configuration register TICK_REG. In this case, configuration register MODE_REG bit 30 must be set to 0. If set to 1, `tick_reg` is used to read the statistics counters.

Configuration Register Map

The configuration space provides software with the ability to configure the IP core for various use cases. Certain features are optional and the assigned register might not exist in a particular variant, in which case the applicable registers are considered RESERVED.

Table 17: **Configuration Register Map**

Hex Address	Register Name/Link to Description
0x0000	GT_RESET_REG: 0000
0x0004	RESET_REG: 0004
0x0008	MODE_REG: 0008
0x000C	CONFIGURATION_TX_REG1: 000C
0x0014	CONFIGURATION_RX_REG1: 0014
0x0018	CONFIGURATION_RX_MTU: 0018
0x0020	TICK_REG: 0020
0x0024	CONFIGURATION_REVISION_REG: 0024
0x0028	CONFIGURATION_TX_TEST_PAT_SEED_A_LSB: 0028
0x002C	CONFIGURATION_TX_TEST_PAT_SEED_A_MSB: 002C
0x0030	CONFIGURATION_TX_TEST_PAT_SEED_B_LSB: 0030
0x0034	CONFIGURATION_TX_TEST_PAT_SEED_B_MSB: 0034
0x0038	CONFIGURATION_1588_REG: 0038
0x0040	CONFIGURATION_TX_FLOW_CONTROL_REG1: 0040
0x0044	CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG1: 0044
0x0048	CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG2: 0048
0x004C	CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG3: 004C
0x0050	CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG4: 0050
0x0054	CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG5: 0054
0x0058	CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG1: 0058
0x005C	CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG2: 005C
0x0060	CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG3: 0060
0x0064	CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG4: 0064
0x0068	CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG5: 0068
0x006C	CONFIGURATION_TX_FLOW_CONTROL_PPP_ETYPE_OP_REG: 006C
0x0070	CONFIGURATION_TX_FLOW_CONTROL_GPP_ETYPE_OP_REG: 0070
0x0074	CONFIGURATION_TX_FLOW_CONTROL_GPP_DA_REG_LSB: 0074
0x0078	CONFIGURATION_TX_FLOW_CONTROL_GPP_DA_REG_MSB: 0078
0x007C	CONFIGURATION_TX_FLOW_CONTROL_GPP_SA_REG_LSB: 007C
0x0080	CONFIGURATION_TX_FLOW_CONTROL_GPP_SA_REG_MSB: 0080
0x0084	CONFIGURATION_TX_FLOW_CONTROL_PPP_DA_REG_LSB: 0084
0x0088	CONFIGURATION_TX_FLOW_CONTROL_PPP_DA_REG_MSB: 0088
0x008C	CONFIGURATION_TX_FLOW_CONTROL_PPP_SA_REG_LSB: 008C
0x0090	CONFIGURATION_TX_FLOW_CONTROL_PPP_SA_REG_MSB: 0090
0x0094	CONFIGURATION_RX_FLOW_CONTROL_REG1: 0094
0x0098	CONFIGURATION_RX_FLOW_CONTROL_REG2: 0098
0x009C	CONFIGURATION_RX_FLOW_CONTROL_PPP_ETYPE_OP_REG: 009C
0x00A0	CONFIGURATION_RX_FLOW_CONTROL_GPP_ETYPE_OP_REG: 00A0
0x00A4	CONFIGURATION_RX_FLOW_CONTROL_GCP_PCP_TYPE_REG: 00A4

Table 17: **Configuration Register Map** *(cont'd)*

Hex Address	Register Name/Link to Description
0x00A8	CONFIGURATION_RX_FLOW_CONTROL_PCP_OP_REG: 00A8
0x00AC	CONFIGURATION_RX_FLOW_CONTROL_GCP_OP_REG: 00AC
0x00B0	CONFIGURATION_RX_FLOW_CONTROL_DA_REG1_LSB: 00B0
0x00B4	CONFIGURATION_RX_FLOW_CONTROL_DA_REG1_MSB: 00B4
0x00B8	CONFIGURATION_RX_FLOW_CONTROL_DA_REG2_LSB: 00B8
0x00BC	CONFIGURATION_RX_FLOW_CONTROL_DA_REG2_MSB: 00BC
0x00C0	CONFIGURATION_RX_FLOW_CONTROL_SA_REG1_LSB: 00C0
0x00C4	CONFIGURATION_RX_FLOW_CONTROL_SA_REG1_MSB: 00C4
0x00C8	CONFIGURATION_USXGMII_AN: 00C8
0x0184	USER_REG_0: 0184
0x0188	USER_REG_1: 0188
0x0194	CONFIGURATION_TX_PTP_LATENCY_ADJUST_REG: 0194
0x0198	CONFIGURATION_RX_PTP_LATENCY_ADJUST_REG: 0198

Status Register Map

The status registers provide an indication of the health of the system. These registers are Read-Only and a read operation clears the register.

Status registers are cleared according to the following conditions.

- Applying `s_axi_aresetn` clears both TX and RX status registers
- When a particular status register is read (clear on read)
- Applying `rx_reset` clears the RX status registers only
- Applying `tx_reset` clears the TX status registers only

Table 18: **Status Register Map**

Hex Address	Register Name/Link to Description
0x0400	STAT_TX_STATUS_REG1: 0400
0x0404	STAT_RX_STATUS_REG1: 0404
0x0408	STAT_STATUS_REG1: 0408
0x040C	STAT_RX_BLOCK_LOCK_REG: 040C
0x0450	STAT_TX_FLOW_CONTROL_REG1: 0450
0x0454	STAT_RX_FLOW_CONTROL_REG1: 0454
0x0458	STAT_AN_STATUS_REG1: 0458
0x0494	STAT_RX_VALID_CTRL_CODE: 0494

Statistics Counters

The statistics counters provide histograms of the classification of traffic and error counts. These counters can be read either by a 1 on pm_tick or by writing a 1 to TICK_REG, depending on the value of MODE_REG[30] (that is, `tick_reg_mode_sel`). pm_tick will be used when MODE_REG[30] = 0 and TICK_REG will be used when MODE_REG[30] = 1 (1 =default).

The counters employ an internal accumulator. A write to the TICK_REG register causes the accumulated counts to be pushed to the readable STAT_*_MSB/LSB registers and simultaneously clear the accumulators. The STAT_*_MSB/LSB registers can then be read. In this way all values stored in the statistics counters represent a snap-shot over the same time interval.

The STAT_CYCLE_COUNT_MSB/LSB register contains a count of the number of RX core clock cycles between TICK_REG writes. This allows for easy time-interval based statistics.

Statistic counter registers are cleared according to the following conditions:

- Applying `s_axi_aresetn` clears both TX and RX statistics counter registers
- Applying PM Tick clears both TX and RX statistics counter registers
- Applying `rx_reset` clears the RX statistics counter registers only
- Applying `tx_reset` clears the TX statistics counter registers only

Table 19: **Statistics Counters**

Hex Address	Register Name/Link to Description
0x0500	STATUS_CYCLE_COUNT_LSB: 0500
0x0504	STATUS_CYCLE_COUNT_MSB: 0504
0x0648	STAT_RX_FRAMING_ERR_LSB: 0648
0x064C	STAT_RX_FRAMING_ERR_MSB: 064C
0x0660	STAT_RX_BAD_CODE_LSB: 0660
0x0664	STAT_RX_BAD_CODE_MSB: 0664
0x06A0	STAT_TX_FRAME_ERROR_LSB: 06A0
0x06A4	STAT_TX_FRAME_ERROR_MSB: 06A4
0x0700	STAT_TX_TOTAL_PACKETS_LSB: 0700
0x0704	STAT_TX_TOTAL_PACKETS_MSB: 0704
0x0708	STAT_TX_TOTAL_GOOD_PACKETS_LSB: 0708
0x070C	STAT_TX_TOTAL_GOOD_PACKETS_MSB: 070C
0x0710	STAT_TX_TOTAL_BYTES_LSB: 0710
0x0714	STAT_TX_TOTAL_BYTES_MSB: 0714
0x0718	STAT_TX_TOTAL_GOOD_BYTES_LSB: 0718
0x071C	STAT_TX_TOTAL_GOOD_BYTES_MSB: 071C
0x0720	STAT_TX_PACKET_64_BYTES_LSB: 0720
0x0724	STAT_TX_PACKET_64_BYTES_MSB: 0724

Table 19: **Statistics Counters** *(cont'd)*

Hex Address	Register Name/Link to Description
0x0728	STAT_TX_PACKET_65_127_BYTES_LSB: 0728
0x072C	STAT_TX_PACKET_65_127_BYTES_MSB: 072C
0x0730	STAT_TX_PACKET_128_255_BYTES_LSB: 0730
0x0734	STAT_TX_PACKET_128_255_BYTES_MSB: 0734
0x0738	STAT_TX_PACKET_256_511_BYTES_LSB: 0738
0x073C	STAT_TX_PACKET_256_511_BYTES_MSB: 073C
0x0740	STAT_TX_PACKET_512_1023_BYTES_LSB: 0740
0x0744	STAT_TX_PACKET_512_1023_BYTES_MSB: 0744
0x0748	STAT_TX_PACKET_1024_1518_BYTES_LSB: 0748
0x074C	STAT_TX_PACKET_1024_1518_BYTES_MSB: 074C
0x0750	STAT_TX_PACKET_1519_1522_BYTES_LSB: 0750
0x0754	STAT_TX_PACKET_1519_1522_BYTES_MSB: 0754
0x0758	STAT_TX_PACKET_1523_1548_BYTES_LSB: 0758
0x075C	STAT_TX_PACKET_1523_1548_BYTES_MSB: 075C
0x0760	STAT_TX_PACKET_1549_2047_BYTES_LSB: 0760
0x0764	STAT_TX_PACKET_1549_2047_BYTES_MSB: 0764
0x0768	STAT_TX_PACKET_2048_4095_BYTES_LSB: 0768
0x076C	STAT_TX_PACKET_2048_4095_BYTES_MSB: 076C
0x0770	STAT_TX_PACKET_4096_8191_BYTES_LSB: 0770
0x0774	STAT_TX_PACKET_4096_8191_BYTES_MSB: 0774
0x0778	STAT_TX_PACKET_8192_9215_BYTES_LSB: 0778
0x077C	STAT_TX_PACKET_8192_9215_BYTES_MSB: 077C
0x0780	STAT_TX_PACKET_LARGE_LSB: 0780
0x0784	STAT_TX_PACKET_LARGE_MSB: 0784
0x0788	STAT_TX_PACKET_SMALL_LSB: 0788
0x078C	STAT_TX_PACKET_SMALL_MSB: 078C
0x07B8	STAT_TX_BAD_FCS_LSB: 07B8
0x07BC	STAT_TX_BAD_FCS_MSB: 07BC
0x07D0	STAT_TX_UNICAST_LSB: 07D0
0x07D4	STAT_TX_UNICAST_MSB: 07D4
0x07D8	STAT_TX_MULTICAST_LSB: 07D8
0x07DC	STAT_TX_MULTICAST_MSB: 07DC
0x07E0	STAT_TX_BROADCAST_LSB: 07E0
0x07E4	STAT_TX_BROADCAST_MSB: 07E4
0x07E8	STAT_TX_VLAN_LSB: 07E8
0x07EC	STAT_TX_VLAN_MSB: 07EC
0x07F0	STAT_TX_PAUSE_LSB: 07F0
0x07F4	STAT_TX_PAUSE_MSB: 07F4
0x07F8	STAT_TX_USER_PAUSE_LSB: 07F8

Table 19: **Statistics Counters** *(cont'd)*

Hex Address	Register Name/Link to Description
0x07FC	STAT_TX_USER_PAUSE_MSB: 07FC
0x0808	STAT_RX_TOTAL_PACKETS_LSB: 0808
0x080C	STAT_RX_TOTAL_PACKETS_MSB: 080C
0x0810	STAT_RX_TOTAL_GOOD_PACKETS_LSB: 0810
0x0814	STAT_RX_TOTAL_GOOD_PACKETS_MSB: 0814
0x0818	STAT_RX_TOTAL_BYTES_LSB: 0818
0x081C	STAT_RX_TOTAL_BYTES_MSB: 081C
0x0820	STAT_RX_TOTAL_GOOD_BYTES_LSB: 0820
0x0824	STAT_RX_TOTAL_GOOD_BYTES_MSB: 0824
0x0828	STAT_RX_PACKET_64_BYTES_LSB: 0828
0x082C	STAT_RX_PACKET_64_BYTES_MSB: 082C
0x0830	STAT_RX_PACKET_65_127_BYTES_LSB: 0830
0x0834	STAT_RX_PACKET_65_127_BYTES_MSB: 0834
0x0838	STAT_RX_PACKET_128_255_BYTES_LSB: 0838
0x083C	STAT_RX_PACKET_128_255_BYTES_MSB: 083C
0x0840	STAT_RX_PACKET_256_511_BYTES_LSB: 0840
0x0844	STAT_RX_PACKET_256_511_BYTES_MSB: 0844
0x0848	STAT_RX_PACKET_512_1023_BYTES_LSB: 0848
0x084C	STAT_RX_PACKET_512_1023_BYTES_MSB: 084C
0x0850	STAT_RX_PACKET_1024_1518_BYTES_LSB: 0850
0x0854	STAT_RX_PACKET_1024_1518_BYTES_MSB: 0854
0x0858	STAT_RX_PACKET_1519_1522_BYTES_LSB: 0858
0x085C	STAT_RX_PACKET_1519_1522_BYTES_MSB: 085C
0x0860	STAT_RX_PACKET_1523_1548_BYTES_LSB: 0860
0x0864	STAT_RX_PACKET_1523_1548_BYTES_MSB: 0864
0x0868	STAT_RX_PACKET_1549_2047_BYTES_LSB: 0868
0x086C	STAT_RX_PACKET_1549_2047_BYTES_MSB: 086C
0x0870	STAT_RX_PACKET_2048_4095_BYTES_LSB: 0870
0x0874	STAT_RX_PACKET_2048_4095_BYTES_MSB: 0874
0x0878	STAT_RX_PACKET_4096_8191_BYTES_LSB: 0878
0x087C	STAT_RX_PACKET_4096_8191_BYTES_MSB: 087C
0x0880	STAT_RX_PACKET_8192_9215_BYTES_LSB: 0880
0x0884	STAT_RX_PACKET_8192_9215_BYTES_MSB: 0884
0x0888	STAT_RX_PACKET_LARGE_LSB: 0888
0x088C	STAT_RX_PACKET_LARGE_MSB: 088C
0x0890	STAT_RX_PACKET_SMALL_LSB: 0890
0x0894	STAT_RX_PACKET_SMALL_MSB: 0894
0x0898	STAT_RX_UNDERSIZE_LSB: 0898
0x089C	STAT_RX_UNDERSIZE_MSB: 089C

Table 19: **Statistics Counters** *(cont'd)*

Hex Address	Register Name/Link to Description
0x08A0	STAT_RX_FRAGMENT_LSB: 08A0
0x08A4	STAT_RX_FRAGMENT_MSB: 08A4
0x08A8	STAT_RX_OVERSIZE_LSB: 08A8
0x08AC	STAT_RX_OVERSIZE_MSB: 08AC
0x08B0	STAT_RX_TOOLONG_LSB: 08B0
0x08B4	STAT_RX_TOOLONG_MSB: 08B4
0x08B8	STAT_RX_JABBER_LSB: 08B8
0x08BC	STAT_RX_JABBER_MSB: 08BC
0x08C0	STAT_RX_BAD_FCS_LSB: 08C0
0x08C4	STAT_RX_BAD_FCS_MSB: 08C4
0x08C8	STAT_RX_PACKET_BAD_FCS_LSB: 08C8
0x08CC	STAT_RX_PACKET_BAD_FCS_MSB: 08CC
0x08D0	STAT_RX_STOMPED_FCS_LSB: 08D0
0x08D4	STAT_RX_STOMPED_FCS_MSB: 08D4
0x08D8	STAT_RX_UNICAST_LSB: 08D8
0x08DC	STAT_RX_UNICAST_MSB: 08DC
0x08E0	STAT_RX_MULTICAST_LSB: 08E0
0x08E4	STAT_RX_MULTICAST_MSB: 08E4
0x08E8	STAT_RX_BROADCAST_LSB: 08E8
0x08EC	STAT_RX_BROADCAST_MSB: 08EC
0x08F0	STAT_RX_VLAN_LSB: 08F0
0x08F4	STAT_RX_VLAN_MSB: 08F4
0x08F8	STAT_RX_PAUSE_LSB: 08F8
0x08FC	STAT_RX_PAUSE_MSB: 08FC
0x0900	STAT_RX_USER_PAUSE_LSB: 0900
0x0904	STAT_RX_USER_PAUSE_MSB: 0904
0x0908	STAT_RX_INRANGEERR_LSB: 0908
0x090C	STAT_RX_INRANGEERR_MSB: 090C
0x0910	STAT_RX_TRUNCATED_LSB: 0910
0x0914	STAT_RX_TRUNCATED_MSB: 0914
0x0918	STAT_RX_TEST_PATTERN_MISMATCH_LSB: 0918
0x091C	STAT_RX_TEST_PATTERN_MISMATCH_MSB: 091C

Register Descriptions

This section contains descriptions of the configuration registers. In the cases where the features described in the bit fields are not present in the IP core, the bit field reverts to RESERVED.

Configuration Registers

The following tables define the bit assignments for the configuration registers. Registers or bit fields within registers can be accessed for Read-Write (RW), Write-Only (WO), or Read-Only (RO). Default values shown are decimal values and take effect after a `s_axi_aresetn`.

A description of each signal is found in Port Descriptions.

GT_RESET_REG: 0000

Table 20: **GT_RESET_REG: 0000**

Bits	Default	Type	Signal
0	0	RW	gt_reset This is a clear on write register.
1	0	RW	ctl_gt_rx_reset
2	0	RW	ctl_gt_tx_reset

RESET_REG: 0004

Table 21: **RESET_REG: 0004**

Bits	Default	Type	Signal
0	0	RW	rx_serdes_reset
29	0	RW	tx_serdes_reset
30	0	RW	rx_reset
31	0	RW	tx_reset

MODE_REG: 0008

Table 22: **RESET_REG: 0008**

Bits	Default	Type	Signal
30	1	RW	tick_reg_mode_sel
31	0	RW	ctl_local_loopback Sets GT in near-end PMA loopback

CONFIGURATION_TX_REG1: 000C

Table 23: **CONFIGURATION_TX_REG1: 000C**

Bits	Default	Type	Signal
0	1	RW	ctl_tx_enable
1	1	RW	ctl_tx_fcs_ins_enable

Table 23: **CONFIGURATION_TX_REG1: 000C** *(cont'd)*

Bits	Default	Type	Signal
2	0	RW	ctl_tx_ignore_fcs
3	0	RW	ctl_tx_send_lfi
4	0	RW	ctl_tx_send_rfi
5	0	RW	ctl_tx_send_idle
6	0	RW	ctl_tx_send_umii_lfi
7	0	RW	ctl_tx_send_umii_rfi
15	0	RW	ctl_tx_test_pattern_enable
16	0	RW	ctl_tx_test_pattern_select
17	0	RW	ctl_tx_data_pattern_select
18	0	RW	ctl_tx_custom_preamble_enable

CONFIGURATION_RX_REG1: 0014

Table 24: **CONFIGURATION_RX_REG1: 0014**

Bits	Default	Type	Signal
0	1	RW	ctl_rx_enable
1	1	RW	ctl_rx_delete_fcs
2	0	RW	ctl_rx_ignore_fcs
3	0	RW	ctl_rx_process_lfi
4	1	RW	ctl_rx_check_sfd
5	1	RW	ctl_rx_check_preamble
6	0	RW	ctl_rx_force_resync
8	0	RW	ctl_rx_test_pattern_enable
9	0	RW	ctl_rx_data_pattern_select
11	0	RW	ctl_rx_custom_preamble_enable

CONFIGURATION_RX_MTU: 0018

Table 25: **CONFIGURATION_RX_MTU: 0018**

Bits	Default	Type	Signal
7:0	64	RW	ctl_rx_min_packet_len
30:16	9,600	RW	ctl_rx_max_packet_len

TICK_REG: 0020

Table 26: **TICK_REG: 0020**

Bits	Default	Type	Signal
0	0	WO	tick_reg

CONFIGURATION_REVISION_REG: 0024

Table 27: **CONFIGURATION_REVISION_REG: 0024**

Bits	Default	Type	Signal
7:0	1	RO	major_rev
15:8	1	RO	minor_rev
31:24	0	RO	patch_rev

CONFIGURATION_TX_TEST_PAT_SEED_A_LSB: 0028

Table 28: **CONFIGURATION_TX_TEST_PAT_SEED_A_LSB: 0028**

Bits	Default	Type	Signal
31:0	0	RW	ctl_tx_test_pattern_seed_a[31:0]

CONFIGURATION_TX_TEST_PAT_SEED_A_MSB: 002C

Table 29: **CONFIGURATION_TX_TEST_PAT_SEED_A_MSB: 002C**

Bits	Default	Type	Signal
25:0	0	RW	ctl_tx_test_pattern_seed_a[57:32]

CONFIGURATION_TX_TEST_PAT_SEED_B_LSB: 0030

Table 30: **CONFIGURATION_TX_TEST_PAT_SEED_B_LSB: 0030**

Bits	Default	Type	Signal
31:0	0	RW	ctl_tx_test_pattern_seed_b[31:0]

CONFIGURATION_TX_TEST_PAT_SEED_B_MSB: 0034

Table 31: **CONFIGURATION_TX_TEST_PAT_SEED_B_MSB: 0034**

Bits	Default	Type	Signal
25:0	0	RW	ctl_tx_test_pattern_seed_b[57:32]

CONFIGURATION_1588_REG: 0038

Table 32: **CONFIGURATION_1588_REG: 0038**

Bits	Default	Type	Signal
0	0	RW	ctl_tx_ptp_1step_enable
2	0	RW	ctl_ptp_transpclk_mode

CONFIGURATION_TX_FLOW_CONTROL_REG1: 0040

Table 33: **CONFIGURATION_TX_FLOW_CONTROL_REG1: 0040**

Bits	Default	Type	Signal
8:0	0	RW	ctl_tx_pause_enable

CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG1: 0044

Table 34: **CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG1: 0044**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_pause_refresh_timer0
31:16	0	RW	ctl_tx_pause_refresh_timer1

CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG2: 0048

Table 35: **CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG2: 0048**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_pause_refresh_timer2
31:16	0	RW	ctl_tx_pause_refresh_timer3

CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG3: 004C

Table 36: **CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG3: 004C**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_pause_refresh_timer4
31:16	0	RW	ctl_tx_pause_refresh_timer5

CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG4: 0050

Table 37: **CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG4: 0050**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_pause_refresh_timer6
31:16	0	RW	ctl_tx_pause_refresh_timer7

CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG5: 0054

Table 38: **CONFIGURATION_TX_FLOW_CONTROL_REFRESH_REG5: 0054**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_pause_refresh_timer8

CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG1: 0058

Table 39: **CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG1: 0058**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_pause_quanta0
31:16	0	RW	ctl_tx_pause_quanta1

CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG2: 005C

Table 40: **CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG2: 005C**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_pause_quanta2
31:16	0	RW	ctl_tx_pause_quanta3

CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG3: 0060

Table 41: **CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG3: 0060**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_pause_quanta4
31:16	0	RW	ctl_tx_pause_quanta5

CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG4: 0064

Table 42: **CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG4: 0064**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_pause_quanta6
31:16	0	RW	ctl_tx_pause_quanta7

CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG5: 0068

Table 43: **CONFIGURATION_TX_FLOW_CONTROL_QUANTA_REG5: 0068**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_pause_quanta8

CONFIGURATION_TX_FLOW_CONTROL_PPP_ETYPE_OP_REG: 006C

Table 44: **CONFIGURATION_TX_FLOW_CONTROL_PPP_ETYPE_OP_REG: 006C**

Bits	Default	Type	Signal
15:0	34824	RW	ctl_tx_ethertype_ppp
31:16	257	RW	ctl_tx_opcode_ppp

CONFIGURATION_TX_FLOW_CONTROL_GPP_ETYPE_OP_REG: 0070

Table 45: **CONFIGURATION_TX_FLOW_CONTROL_GPP_ETYPE_OP_REG: 0070**

Bits	Default	Type	Signal
15:0	34824	RW	ctl_tx_ethertype_gpp
31:16	1	RW	ctl_tx_opcode_gpp

CONFIGURATION_TX_FLOW_CONTROL_GPP_DA_REG_LSB: 0074

Table 46: **CONFIGURATION_TX_FLOW_CONTROL_GPP_DA_REG_LSB: 0074**

Bits	Default	Type	Signal
31:0	0	RW	ctl_tx_da_gpp[31:0]

CONFIGURATION_TX_FLOW_CONTROL_GPP_DA_REG_MSB: 0078

Table 47: **CONFIGURATION_TX_FLOW_CONTROL_GPP_DA_REG_MSB: 0078**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_da_gpp[47:32]

CONFIGURATION_TX_FLOW_CONTROL_GPP_SA_REG_LSB: 007C

Table 48: **CONFIGURATION_TX_FLOW_CONTROL_GPP_SA_REG_LSB: 007C**

Bits	Default	Type	Signal
31:0	0	RW	ctl_tx_sa_gpp[31:0]

CONFIGURATION_TX_FLOW_CONTROL_GPP_SA_REG_MSB: 0080

Table 49: **CONFIGURATION_TX_FLOW_CONTROL_GPP_SA_REG_MSB: 0080**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_sa_gpp[47:32]

CONFIGURATION_TX_FLOW_CONTROL_PPP_DA_REG_LSB: 0084

Table 50: **CONFIGURATION_TX_FLOW_CONTROL_PPP_DA_REG_LSB: 0084**

Bits	Default	Type	Signal
31:0	0	RW	ctl_tx_da_ppp[31:0]

CONFIGURATION_TX_FLOW_CONTROL_PPP_DA_REG_MSB: 0088

Table 51: **CONFIGURATION_TX_FLOW_CONTROL_PPP_DA_REG_MSB: 0088**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_da_ppp[47:32]

CONFIGURATION_TX_FLOW_CONTROL_PPP_SA_REG_LSB: 008C

Bits	Default	Type	Signal
31:0	0	RW	ctl_tx_sa_ppp[31:0]

CONFIGURATION_TX_FLOW_CONTROL_PPP_SA_REG_MSB: 0090

Table 52: **CONFIGURATION_TX_FLOW_CONTROL_PPP_SA_REG_MSB: 0090**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_sa_ppp[47:32]

CONFIGURATION_RX_FLOW_CONTROL_REG1: 0094

Table 53: **CONFIGURATION_RX_FLOW_CONTROL_REG1: 0094**

Bits	Default	Type	Signal
8:0	0	RW	ctl_rx_pause_enable
9	0	RW	ctl_rx_forward_control
10	0	RW	ctl_rx_enable_gcp
11	0	RW	ctl_rx_enable_pcp
12	0	RW	ctl_rx_enable_gpp
13	0	RW	ctl_rx_enable_ppp
14	0	RW	ctl_rx_check_ack

CONFIGURATION_RX_FLOW_CONTROL_REG2: 0098

Table 54: **CONFIGURATION_RX_FLOW_CONTROL_REG2: 0098**

Bits	Default	Type	Signal
0	0	RW	ctl_rx_check_mcast_gcp
1	0	RW	ctl_rx_check_ucast_gcp
2	0	RW	ctl_rx_check_sa_gcp
3	0	RW	ctl_rx_check_etype_gcp
4	0	RW	ctl_rx_check_opcode_gcp
5	0	RW	ctl_rx_check_mcast_pcp
6	0	RW	ctl_rx_check_ucast_pcp
7	0	RW	ctl_rx_check_sa_pcp
8	0	RW	ctl_rx_check_etype_pcp
9	0	RW	ctl_rx_check_opcode_pcp
10	0	RW	ctl_rx_check_mcast_gpp
11	0	RW	ctl_rx_check_ucast_gpp
12	0	RW	ctl_rx_check_sa_gpp
13	0	RW	ctl_rx_check_etype_gpp
14	0	RW	ctl_rx_check_opcode_gpp
15	0	RW	ctl_rx_check_mcast_ppp
16	0	RW	ctl_rx_check_ucast_ppp
17	0	RW	ctl_rx_check_sa_ppp
18	0	RW	ctl_rx_check_etype_ppp
19	0	RW	ctl_rx_check_opcode_ppp

CONFIGURATION_RX_FLOW_CONTROL_PPP_ETYPE_OP_REG: 009C

Table 55: **CONFIGURATION_RX_FLOW_CONTROL_PPP_ETYPE_OP_REG: 009C**

Bits	Default	Type	Signal
15:0	34,824	RW	ctl_rx_etype_ppp
31:16	257	RW	ctl_rx_opcode_ppp

CONFIGURATION_RX_FLOW_CONTROL_GPP_ETYPE_OP_REG: 00A0

Table 56: **CONFIGURATION_RX_FLOW_CONTROL_GPP_ETYPE_OP_REG: 00A0**

Bits	Default	Type	Signal
15:0	34,824	RW	ctl_rx_etype_gpp
31:16	1	RW	ctl_rx_opcode_gpp

CONFIGURATION_RX_FLOW_CONTROL_GCP_PCP_TYPE_REG: 00A4

Table 57: **CONFIGURATION_RX_FLOW_CONTROL_GCP_PCP_TYPE_REG: 00A4**

Bits	Default	Type	Signal
15:0	34,824	RW	ctl_rx_etype_gcp
31:16	34,824	RW	ctl_rx_etype_pcp

CONFIGURATION_RX_FLOW_CONTROL_PCP_OP_REG: 00A8

Table 58: **CONFIGURATION_RX_FLOW_CONTROL_PCP_OP_REG: 00A8**

Bits	Default	Type	Signal
15:0	257	RW	ctl_rx_opcode_min_pcp
31:16	257	RW	ctl_rx_opcode_max_pcp

CONFIGURATION_RX_FLOW_CONTROL_GCP_OP_REG: 00AC

Table 59: **CONFIGURATION_RX_FLOW_CONTROL_GCP_OP_REG: 00AC**

Bits	Default	Type	Signal
15:0	1	RW	ctl_rx_opcode_min_gcp
31:16	6	RW	ctl_rx_opcode_max_gcp

CONFIGURATION_RX_FLOW_CONTROL_DA_REG1_LSB: 00B0

Table 60: **CONFIGURATION_RX_FLOW_CONTROL_DA_REG1_LSB: 00B0**

Bits	Default	Type	Signal
31:0	0	RW	ctl_rx_pause_da_ucast[31:0]

CONFIGURATION_RX_FLOW_CONTROL_DA_REG1_MSB: 00B4

Table 61: **CONFIGURATION_RX_FLOW_CONTROL_DA_REG1_MSB: 00B4**

Bits	Default	Type	Signal
15:0	0	RW	ctl_rx_pause_da_ucast[47:32]

CONFIGURATION_RX_FLOW_CONTROL_DA_REG2_LSB: 00B8

Table 62: **CONFIGURATION_RX_FLOW_CONTROL_DA_REG2_LSB: 00B8**

Bits	Default	Type	Signal
31:0	0	RW	ctl_rx_pause_da_mcast[31:0]

CONFIGURATION_RX_FLOW_CONTROL_DA_REG2_MSB: 00BC

Table 63: **CONFIGURATION_RX_FLOW_CONTROL_DA_REG2_MSB: 00BC**

Bits	Default	Type	Signal
15:0	0	RW	ctl_rx_pause_da_mcast[47:32]

CONFIGURATION_RX_FLOW_CONTROL_SA_REG1_LSB: 00C0

Table 64: **CONFIGURATION_RX_FLOW_CONTROL_SA_REG1_LSB: 00C0**

Bits	Default	Type	Signal
31:0	0	RW	ctl_rx_pause_sa[31:0]

CONFIGURATION_RX_FLOW_CONTROL_SA_REG1_MSB: 00C4

Table 65: **CONFIGURATION_RX_FLOW_CONTROL_SA_REG1_MSB: 00C4**

Bits	Default	Type	Signal
15:0	0	RW	ctl_rx_pause_sa[47:32]

CONFIGURATION_USXGMII_AN: 00C8

Table 66: **CONFIGURATION_USXGMII_AN: 00C8**

Bits	Default	Type	Signal
0	0	RW	ctl_umii_an_bypass
4:1	000	RW	ctl_umii_an_link_timer_config
5	0	RW	ctl_umii_an_mr_an_enable
6	0	RW	ctl_umii_an_mr_main_reset
7	0	RW	ctl_umii_an_mr_restart_an
10:8	011	RW	ctl_usxgmii_rate • 000: 10 Mb/s • 001: 100 Mb/s • 010: 1 Gb/s • 011: 10 Gb/s • 100: 2.5 Gb/s • 101: 5 Gb/s
15:11	0	RO	Reserved for future use
16	1	RW	USXGMII
22:17	0	RW	Reserved for future use
23	0	RW	EEE clock stop capability; Not supported
24	0	RW	EEE capability; Not supported.
27:25	011	RW	Speed; refer to table f N-base-T_usxgmii rev1.4
28	1	RW	Duplex Mode. 1 = Full Duplex, 0=Half Duplex
29	0	RW	Reserved for future use
30	0	RO	AN acknowledge
31	0	RW	Link Status

USER_REG_0: 0184

Table 67: **USER_REG_0: 0184**

Bits	Default	Type	Signal
31:0	0	RW	user_reg0

USER_REG_1: 0188

Table 68: **USER_REG_1: 0188**

Bits	Default	Type	Signal
31:0	0	RW	user_reg1

CONFIGURATION_TX_PTP_LATENCY_ADJUST_REG: 0194

Table 69: **CONFIGURATION_TX_PTP_LATENCY_ADJUST_REG: 0194**

Bits	Default	Type	Signal
15:0	0	RW	ctl_tx_ptp_latency_adjust

CONFIGURATION_RX_PTP_LATENCY_ADJUST_REG: 0198

Table 70: **CONFIGURATION_RX_PTP_LATENCY_ADJUST_REG: 0198**

Bits	Default	Type	Signal
15:0	0	RW	ctl_rx_ptp_latency_adjust

Status Registers

The tables in this section define the bit assignments for the status registers. Some bits are sticky, that is, latching their value High or Low once set. This is indicated by the type LH (Latched High) or LL (Latched Low).

A description of each signal is found in Port Descriptions.

STAT_TX_STATUS_REG1: 0400

Table 71: **STAT_TX_STATUS_REG1: 0400**

Bits	Default	Type	Signal
0	0	RO LH	stat_tx_local_fault

STAT_RX_STATUS_REG1: 0404

Table 72: **STAT_RX_STATUS_REG1: 0404**

Bits	Default	Type	Signal
4	0	RO LH	stat_rx_hi_ber
5	0	RO LH	stat_rx_remote_fault
6	0	RO LH	stat_rx_local_fault
7	0	RO LH	stat_rx_internal_local_fault
8	0	RO LH	stat_rx_received_local_fault
9	0	RO LH	stat_rx_bad_preamble
10	0	RO LH	stat_rx_bad_sfd
11	0	RO LH	stat_rx_got_signal_os
12	0	RO LH	stat_rx_umii_local_fault
13	0	RO LH	stat_rx_umii_remote_fault

STAT_STATUS_REG1: 0408

Table 73: **STAT_STATUS_REG1: 0408**

Bits	Default	Type	Signal
4	0	RO LH	stat_tx_ptp_fifo_read_error
5	0	RO LH	stat_tx_ptp_fifo_write_error

STAT_RX_BLOCK_LOCK_REG: 040C

Table 74: **STAT_RX_BLOCK_LOCK_REG: 040C**

Bits	Default	Type	Signal
0	1	RO LL	stat_rx_block_lock

STAT_TX_FLOW_CONTROL_REG1: 0450

Table 75: **STAT_TX_FLOW_CONTROL_REG1: 0450**

Bits	Default	Type	Signal
8:0	0	RO LH	stat_tx_pause_valid

STAT_RX_FLOW_CONTROL_REG1: 0454

Table 76: **STAT_RX_FLOW_CONTROL_REG1: 0454**

Bits	Default	Type	Signal
8:0	0	RO LH	stat_rx_pause_req
17:9	0	RO LH	stat_rx_pause_valid

STAT_AN_STATUS_REG1: 0458

Table 77: **STAT_AN_STATUS_REG1: 0458**

Bits	Default	Type	Signal
15:0	0	RO	stat_umii_an_mr_lp_adv_ability
16	0	RO	stat_umii_an_mr_an_complete
17	0	RO	stat_umii_an_mr_np_able

Notes:

1. AN Status register is not clear on read.

STAT_RX_VALID_CTRL_CODE: 0494

Table 78: **STAT_RX_VALID_CTRL_CODE: 0494**

Bits	Default	Type	Signal
0	0	RO LH	stat_rx_valid_ctrl_code

STATUS_CYCLE_COUNT_LSB: 0500

Table 79: **STATUS_CYCLE_COUNT_LSB: 0500**

Bits	Default	Type	Signal
31:0	0	RO HIST	stat_cycle_count[31:0]

STATUS_CYCLE_COUNT_MSB: 0504

Table 80: **STATUS_CYCLE_COUNT_MSB: 0504**

Bits	Default	Type	Signal
15:0	0	RO HIST	stat_cycle_count[47:32]

STAT_RX_FRAMING_ERR_LSB: 0648

Table 81: **STAT_RX_FRAMING_ERR_LSB: 0648**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_framing_err_count[31:0]

STAT_RX_FRAMING_ERR_MSB: 064C

Table 82: **STAT_RX_FRAMING_ERR_MSB: 064C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_framing_err_count[48-1:32]

Statistics Counters

The following tables define the bit assignments for the statistics counters. Counters are 48 bits and require two 32-bit address spaces containing the MSB and LSB as indicated. The default value of all counters is 0. Counters are cleared when read by tick_reg (or pm_tick if so selected) but the register containing the count retains its value. Each counter saturates at FFFFFFFFFFFF (hex).

A description of each signal is found in Port Descriptions.

STAT_RX_FRAMING_ERR_MSB: 064C

Table 83: **STAT_RX_FRAMING_ERR_MSB: 064C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_framing_err_count[48-1:32]

STAT_RX_BAD_CODE_LSB: 0660

Table 84: **STAT_RX_BAD_CODE_LSB: 0660**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_bad_code_count[31:0]

STAT_RX_BAD_CODE_MSB: 0664

Table 85: **STAT_RX_BAD_CODE_MSB: 0664**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_bad_code_count[47:32]

STAT_TX_FRAME_ERROR_LSB: 06A0

Table 86: **STAT_TX_FRAME_ERROR_LSB: 06A0**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_frame_error_count[31:0]

STAT_TX_FRAME_ERROR_MSB: 06A4

Table 87: **STAT_TX_FRAME_ERROR_MSB: 06A4**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_frame_error_count[47:32]

STAT_TX_TOTAL_PACKETS_LSB: 0700

Table 88: **STAT_TX_TOTAL_PACKETS_LSB: 0700**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_total_packets_count[31:0]

STAT_TX_TOTAL_PACKETS_MSB: 0704

Table 89: **STAT_TX_TOTAL_PACKETS_MSB: 0704**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_total_packets_count[47:32]

STAT_TX_TOTAL_GOOD_PACKETS_LSB: 0708

Table 90: **STAT_TX_TOTAL_GOOD_PACKETS_LSB: 0708**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_total_good_packets_count[31:0]

STAT_TX_TOTAL_GOOD_PACKETS_MSB: 070C

Table 91: **STAT_TX_TOTAL_GOOD_PACKETS_MSB: 070C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_total_good_packets_count[47:32]

STAT_TX_TOTAL_BYTES_LSB: 0710

Table 92: **STAT_TX_TOTAL_BYTES_LSB: 0710**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_total_bytes_count[31:0]

STAT_TX_TOTAL_BYTES_MSB: 0714

Table 93: **STAT_TX_TOTAL_BYTES_MSB: 0714**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_total_bytes_count[47:32]

STAT_TX_TOTAL_GOOD_BYTES_LSB: 0718

Table 94: **STAT_TX_TOTAL_GOOD_BYTES_LSB: 0718**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_total_good_bytes_count[31:0]

STAT_TX_TOTAL_GOOD_BYTES_MSB: 071C

Table 95: **STAT_TX_TOTAL_GOOD_BYTES_MSB: 071C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_total_good_bytes_count[47:32]

STAT_TX_PACKET_64_BYTES_LSB: 0720

Table 96: **STAT_TX_PACKET_64_BYTES_LSB: 0720**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_64_bytes_count[31:0]

STAT_TX_PACKET_64_BYTES_MSB: 0724

Table 97: **STAT_TX_PACKET_64_BYTES_MSB: 0724**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_64_bytes_count[47:32]

STAT_TX_PACKET_65_127_BYTES_LSB: 0728

Table 98: **STAT_TX_PACKET_65_127_BYTES_LSB: 0728**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_65_127_bytes_count[31:0]

STAT_TX_PACKET_65_127_BYTES_MSB: 072C

Table 99: **STAT_TX_PACKET_65_127_BYTES_MSB: 072C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_65_127_bytes_count[47:32]

STAT_TX_PACKET_128_255_BYTES_LSB: 0730

Table 100: **STAT_TX_PACKET_128_255_BYTES_LSB: 0730**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_128_255_bytes_count[31:0]

STAT_TX_PACKET_128_255_BYTES_MSB: 0734

Table 101: **STAT_TX_PACKET_128_255_BYTES_MSB: 0734**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_128_255_bytes_count[47:32]

STAT_TX_PACKET_256_511_BYTES_LSB: 0738

Table 102: **STAT_TX_PACKET_256_511_BYTES_LSB: 0738**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_256_511_bytes_count[31:0]

STAT_TX_PACKET_256_511_BYTES_MSB: 073C

Table 103: **STAT_TX_PACKET_256_511_BYTES_MSB: 073C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_256_511_bytes_count[47:32]

STAT_TX_PACKET_512_1023_BYTES_LSB: 0740

Table 104: **STAT_TX_PACKET_512_1023_BYTES_LSB: 0740**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_512_1023_bytes_count[31:0]

STAT_TX_PACKET_512_1023_BYTES_MSB: 0744

Table 105: **STAT_TX_PACKET_512_1023_BYTES_MSB: 0744**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_512_1023_bytes_count[47:32]

STAT_TX_PACKET_1024_1518_BYTES_LSB: 0748

Table 106: **STAT_TX_PACKET_1024_1518_BYTES_LSB: 0748**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_1024_1518_bytes_count[31:0]

STAT_TX_PACKET_1024_1518_BYTES_MSB: 074C

Table 107: **STAT_TX_PACKET_1024_1518_BYTES_MSB: 074C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_1024_1518_bytes_count[47:32]

STAT_TX_PACKET_1519_1522_BYTES_LSB: 0750

Table 108: **STAT_TX_PACKET_1519_1522_BYTES_LSB: 0750**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_1519_1522_bytes_count[31:0]

STAT_TX_PACKET_1519_1522_BYTES_MSB: 0754

Table 109: **STAT_TX_PACKET_1519_1522_BYTES_MSB: 0754**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_1519_1522_bytes_count[47:32]

STAT_TX_PACKET_1523_1548_BYTES_LSB: 0758

Table 110: **STAT_TX_PACKET_1523_1548_BYTES_LSB: 0758**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_1523_1548_bytes_count[31:0]

STAT_TX_PACKET_1523_1548_BYTES_MSB: 075C

Table 111: **STAT_TX_PACKET_1523_1548_BYTES_MSB: 075C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_1523_1548_bytes_count[47:32]

STAT_TX_PACKET_1549_2047_BYTES_LSB: 0760

Table 112: **STAT_TX_PACKET_1549_2047_BYTES_LSB: 0760**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_1549_2047_bytes_count[31:0]

STAT_TX_PACKET_1549_2047_BYTES_MSB: 0764

Table 113: **STAT_TX_PACKET_1549_2047_BYTES_MSB: 0764**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_1549_2047_bytes_count[47:32]

STAT_TX_PACKET_2048_4095_BYTES_LSB: 0768

Table 114: **STAT_TX_PACKET_2048_4095_BYTES_LSB: 0768**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_2048_4095_bytes_count[31:0]

STAT_TX_PACKET_2048_4095_BYTES_MSB: 076C

Table 115: **STAT_TX_PACKET_2048_4095_BYTES_MSB: 076C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_2048_4095_bytes_count[47:32]

STAT_TX_PACKET_4096_8191_BYTES_LSB: 0770

Table 116: **STAT_TX_PACKET_4096_8191_BYTES_LSB: 0770**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_4096_8191_bytes_count[31:0]

STAT_TX_PACKET_4096_8191_BYTES_MSB: 0774

Table 117: **STAT_TX_PACKET_4096_8191_BYTES_MSB: 0774**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_4096_8191_bytes_count[47:32]

STAT_TX_PACKET_8192_9215_BYTES_LSB: 0778

Table 118: **STAT_TX_PACKET_8192_9215_BYTES_LSB: 0778**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_8192_9215_bytes_count[31:0]

STAT_TX_PACKET_8192_9215_BYTES_MSB: 077C

Table 119: **STAT_TX_PACKET_8192_9215_BYTES_MSB: 077C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_8192_9215_bytes_count[47:32]

STAT_TX_PACKET_LARGE_LSB: 0780

Table 120: **STAT_TX_PACKET_LARGE_LSB: 0780**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_large_count[31:0]

STAT_TX_PACKET_LARGE_MSB: 0784

Table 121: **STAT_TX_PACKET_LARGE_MSB: 0784**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_large_count[47:32]

STAT_TX_PACKET_SMALL_LSB: 0788

Table 122: **STAT_TX_PACKET_SMALL_LSB: 0788**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_packet_small_count[31:0]

STAT_TX_PACKET_SMALL_MSB: 078C

Table 123: **STAT_TX_PACKET_SMALL_MSB: 078C1***

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_packet_small_count[47:32]

STAT_TX_BAD_FCS_LSB: 07B8

Table 124: **STAT_TX_BAD_FCS_LSB: 07B8**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_bad_fcs_count[31:0]

STAT_TX_BAD_FCS_MSB: 07BC

Table 125: **STAT_TX_BAD_FCS_MSB: 07BC**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_bad_fcs_count[47:32]

STAT_TX_UNICAST_LSB: 07D0

Table 126: **STAT_TX_BAD_FCS_MSB: 07BC**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_unicast_count[31:0]

STAT_TX_UNICAST_MSB: 07D4

Table 127: **STAT_TX_UNICAST_MSB: 07D4**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_unicast_count[47:32]

STAT_TX_MULTICAST_LSB: 07D8

Table 128: **STAT_TX_MULTICAST_LSB: 07D8**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_multicast_count[31:0]

STAT_TX_MULTICAST_MSB: 07DC

Table 129: **STAT_TX_MULTICAST_MSB: 07DC**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_multicast_count[47:32]

STAT_TX_BROADCAST_LSB: 07E0

Table 130: **STAT_TX_BROADCAST_LSB: 07E0**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_broadcast_count[31:0]

STAT_TX_BROADCAST_MSB: 07E4

Table 131: **STAT_TX_BROADCAST_MSB: 07E4**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_broadcast_count[47:32]

STAT_TX_VLAN_LSB: 07E8

Table 132: **STAT_TX_VLAN_LSB: 07E8**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_vlan_count[31:0]

STAT_TX_VLAN_MSB: 07EC

Table 133: **STAT_TX_VLAN_MSB: 07EC**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_vlan_count[47:32]

STAT_TX_PAUSE_LSB: 07F0

Table 134: **STAT_TX_PAUSE_LSB: 07F0**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_pause_count[31:0]

STAT_TX_PAUSE_MSB: 07F4

Table 135: **STAT_TX_PAUSE_MSB: 07F4**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_pause_count[47:32]

STAT_TX_USER_PAUSE_LSB: 07F8

Table 136: **STAT_TX_USER_PAUSE_LSB: 07F8**

Bits	Default	Type	Signal
31:0	0	HIST	stat_tx_user_pause_count[31:0]

STAT_TX_USER_PAUSE_MSB: 07FC

Table 137: **STAT_TX_USER_PAUSE_MSB: 07FC**

Bits	Default	Type	Signal
15:0	0	HIST	stat_tx_user_pause_count[47:32]

STAT_RX_TOTAL_PACKETS_LSB: 0808

Table 138: **STAT_RX_TOTAL_PACKETS_LSB: 0808**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_total_packets_count[31:0]

STAT_RX_TOTAL_PACKETS_MSB: 080C

Table 139: **STAT_RX_TOTAL_PACKETS_MSB: 080C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_total_packets_count[47:32]

STAT_RX_TOTAL_GOOD_PACKETS_LSB: 0810

Table 140: **STAT_RX_TOTAL_GOOD_PACKETS_LSB: 0810**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_total_good_packets_count[31:0]

STAT_RX_TOTAL_GOOD_PACKETS_MSB: 0814

Table 141: **STAT_RX_TOTAL_GOOD_PACKETS_MSB: 0814**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_total_good_packets_count[47:32]

STAT_RX_TOTAL_BYTES_LSB: 0818

Table 142: **STAT_RX_TOTAL_BYTES_LSB: 0818**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_total_bytes_count[31:0]

STAT_RX_TOTAL_BYTES_MSB: 081C

Table 143: **STAT_RX_TOTAL_BYTES_MSB: 081C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_total_bytes_count[47:32]

STAT_RX_TOTAL_GOOD_BYTES_LSB: 0820

Table 144: **STAT_RX_TOTAL_GOOD_BYTES_LSB: 0820**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_total_good_bytes_count[31:0]

STAT_RX_TOTAL_GOOD_BYTES_MSB: 0824

Table 145: **STAT_RX_TOTAL_GOOD_BYTES_MSB: 0824**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_total_good_bytes_count[47:32]

STAT_RX_PACKET_64_BYTES_LSB: 0828

Table 146: **STAT_RX_PACKET_64_BYTES_LSB: 0828**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_64_bytes_count[31:0]

STAT_RX_PACKET_64_BYTES_MSB: 082C

Table 147: **STAT_RX_PACKET_64_BYTES_MSB: 082C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_64_bytes_count[47:32]

STAT_RX_PACKET_65_127_BYTES_LSB: 0830

Table 148: **STAT_RX_PACKET_65_127_BYTES_LSB: 0830**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_65_127_bytes_count[31:0]

STAT_RX_PACKET_65_127_BYTES_MSB: 0834

Table 149: **STAT_RX_PACKET_65_127_BYTES_MSB: 0834**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_65_127_bytes_count[47:32]

STAT_RX_PACKET_128_255_BYTES_LSB: 0838

Table 150: **STAT_RX_PACKET_128_255_BYTES_LSB: 0838**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_128_255_bytes_count[31:0]

STAT_RX_PACKET_128_255_BYTES_MSB: 083C

Table 151: **STAT_RX_PACKET_128_255_BYTES_MSB: 083C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_128_255_bytes_count[47:32]

STAT_RX_PACKET_256_511_BYTES_LSB: 0840

Table 152: **STAT_RX_PACKET_256_511_BYTES_LSB: 0840**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_256_511_bytes_count[31:0]

STAT_RX_PACKET_256_511_BYTES_MSB: 0844

Table 153: **STAT_RX_PACKET_256_511_BYTES_MSB: 0844**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_256_511_bytes_count[47:32]

STAT_RX_PACKET_512_1023_BYTES_LSB: 0848

Table 154: **STAT_RX_PACKET_512_1023_BYTES_LSB: 0848**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_512_1023_bytes_count[31:0]

STAT_RX_PACKET_512_1023_BYTES_MSB: 084C

Table 155: **STAT_RX_PACKET_512_1023_BYTES_MSB: 084C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_512_1023_bytes_count[47:32]

STAT_RX_PACKET_1024_1518_BYTES_LSB: 0850

Table 156: **STAT_RX_PACKET_1024_1518_BYTES_LSB: 0850**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_1024_1518_bytes_count[31:0]

STAT_RX_PACKET_1024_1518_BYTES_MSB: 0854

Table 157: **STAT_RX_PACKET_1024_1518_BYTES_MSB: 0854**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_1024_1518_bytes_count[47:32]

STAT_RX_PACKET_1519_1522_BYTES_LSB: 0858

Table 158: **STAT_RX_PACKET_1519_1522_BYTES_LSB: 0858**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_1519_1522_bytes_count[31:0]

STAT_RX_PACKET_1519_1522_BYTES_MSB: 085C

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_1519_1522_bytes_count[47:32]

STAT_RX_PACKET_1523_1548_BYTES_LSB: 0860

Table 159: **STAT_RX_PACKET_1523_1548_BYTES_LSB: 0860**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_1523_1548_bytes_count[31:0]

STAT_RX_PACKET_1523_1548_BYTES_MSB: 0864

Table 160: **STAT_RX_PACKET_1523_1548_BYTES_MSB: 0864**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_1523_1548_bytes_count[47:32]

STAT_RX_PACKET_1549_2047_BYTES_LSB: 0868

Table 161: **STAT_RX_PACKET_1549_2047_BYTES_LSB: 0868**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_1549_2047_bytes_count[31:0]

STAT_RX_PACKET_1549_2047_BYTES_MSB: 086C

Table 162: **STAT_RX_PACKET_1549_2047_BYTES_MSB: 086C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_1549_2047_bytes_count[47:32]

STAT_RX_PACKET_2048_4095_BYTES_LSB: 0870

Table 163: **STAT_RX_PACKET_2048_4095_BYTES_LSB: 0870**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_2048_4095_bytes_count[31:0]

STAT_RX_PACKET_2048_4095_BYTES_MSB: 0874

Table 164: **STAT_RX_PACKET_2048_4095_BYTES_MSB: 0874**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_2048_4095_bytes_count[47:32]

STAT_RX_PACKET_4096_8191_BYTES_LSB: 0878

Table 165: **STAT_RX_PACKET_4096_8191_BYTES_LSB: 0878**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_4096_8191_bytes_count[31:0]

STAT_RX_PACKET_4096_8191_BYTES_MSB: 087C

Table 166: **STAT_RX_PACKET_4096_8191_BYTES_MSB: 087C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_4096_8191_bytes_count[47:32]

STAT_RX_PACKET_8192_9215_BYTES_LSB: 0880

Table 167: **STAT_RX_PACKET_8192_9215_BYTES_LSB: 0880**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_8192_9215_bytes_count[31:0]

STAT_RX_PACKET_8192_9215_BYTES_MSB: 0884

Table 168: **STAT_RX_PACKET_8192_9215_BYTES_MSB: 0884**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_8192_9215_bytes_count[47:32]

STAT_RX_PACKET_LARGE_LSB: 0888

Table 169: **STAT_RX_PACKET_LARGE_LSB: 0888**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_large_count[31:0]

STAT_RX_PACKET_LARGE_MSB: 088C

Table 170: **STAT_RX_PACKET_LARGE_MSB: 088C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_large_count[47:32]

STAT_RX_PACKET_SMALL_LSB: 0890

Table 171: **STAT_RX_PACKET_SMALL_LSB: 0890**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_small_count[31:0]

STAT_RX_PACKET_SMALL_MSB: 0894

Table 172: **STAT_RX_PACKET_SMALL_MSB: 0894**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_small_count[47:32]

STAT_RX_UNDERSIZE_LSB: 0898

Table 173: **STAT_RX_UNDERSIZE_LSB: 0898**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_undersize_count[31:0]

STAT_RX_UNDERSIZE_MSB: 089C

Table 174: **STAT_RX_UNDERSIZE_MSB: 089C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_undersize_count[47:32]

STAT_RX_FRAGMENT_LSB: 08A0

Table 175: **STAT_RX_FRAGMENT_LSB: 08A0**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_fragment_count[31:0]

STAT_RX_FRAGMENT_MSB: 08A4

Table 176: **STAT_RX_FRAGMENT_MSB: 08A4**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_fragment_count[47:32]

STAT_RX_OVERSIZE_LSB: 08A8

Table 177: **STAT_RX_OVERSIZE_LSB: 08A8**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_oversize_count[31:0]

STAT_RX_OVERSIZE_MSB: 08AC

Table 178: **STAT_RX_OVERSIZE_MSB: 08AC**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_oversize_count[47:32]

STAT_RX_TOOLONG_LSB: 08B0

Table 179: **STAT_RX_TOOLONG_LSB: 08B0**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_toolong_count[31:0]

STAT_RX_TOOLONG_MSB: 08B4

Table 180: **STAT_RX_TOOLONG_MSB: 08B4**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_toolong_count[47:32]

STAT_RX_JABBER_LSB: 08B8

Table 181: **STAT_RX_JABBER_LSB: 08B8**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_jabber_count[31:0]

STAT_RX_JABBER_MSB: 08BC

Table 182: **STAT_RX_JABBER_MSB: 08BC**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_jabber_count[47:32]

STAT_RX_BAD_FCS_LSB: 08C0

Table 183: **STAT_RX_BAD_FCS_LSB: 08C0**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_bad_fcs_count[31:0]

STAT_RX_BAD_FCS_MSB: 08C4

Table 184: **STAT_RX_BAD_FCS_MSB: 08C4**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_bad_fcs_count[47:32]

STAT_RX_PACKET_BAD_FCS_LSB: 08C8

Table 185: **STAT_RX_PACKET_BAD_FCS_LSB: 08C8**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_packet_bad_fcs_count[31:0]

STAT_RX_PACKET_BAD_FCS_MSB: 08CC

Table 186: **STAT_RX_PACKET_BAD_FCS_MSB: 08CC**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_packet_bad_fcs_count[47:32]

STAT_RX_STOMPED_FCS_LSB: 08D0

Table 187: **STAT_RX_STOMPED_FCS_LSB: 08D0**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_stomped_fcs_count[31:0]

STAT_RX_STOMPED_FCS_MSB: 08D4

Table 188: **STAT_RX_STOMPED_FCS_MSB: 08D4**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_stomped_fcs_count[47:32]

STAT_RX_UNICAST_LSB: 08D8

Table 189: **STAT_RX_UNICAST_LSB: 08D8**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_unicast_count[31:0]

STAT_RX_UNICAST_MSB: 08DC

Table 190: **STAT_RX_UNICAST_MSB: 08DC**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_unicast_count[47:32]

STAT_RX_MULTICAST_LSB: 08E0

Table 191: **STAT_RX_MULTICAST_LSB: 08E0**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_multicast_count[31:0]

STAT_RX_MULTICAST_MSB: 08E4

Table 192: **STAT_RX_MULTICAST_MSB: 08E4**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_multicast_count[47:32]

STAT_RX_BROADCAST_LSB: 08E8

Table 193: **STAT_RX_BROADCAST_LSB: 08E8**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_broadcast_count[31:0]

STAT_RX_BROADCAST_MSB: 08EC

Table 194: **STAT_RX_BROADCAST_MSB: 08EC**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_broadcast_count[47:32]

STAT_RX_VLAN_LSB: 08F0

Table 195: **STAT_RX_VLAN_LSB: 08F0**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_vlan_count[31:0]

STAT_RX_VLAN_MSB: 08F4

Table 196: **STAT_RX_VLAN_MSB: 08F4**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_vlan_count[47:32]

STAT_RX_PAUSE_LSB: 08F8

Table 197: **STAT_RX_PAUSE_LSB: 08F8**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_pause_count[31:0]

STAT_RX_PAUSE_MSB: 08FC

Table 198: **STAT_RX_PAUSE_MSB: 08FC**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_pause_count[47:32]

STAT_RX_USER_PAUSE_LSB: 0900

Table 199: **STAT_RX_USER_PAUSE_LSB: 0900**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_user_pause_count[31:0]

STAT_RX_USER_PAUSE_MSB: 0904

Table 200: **STAT_RX_USER_PAUSE_MSB: 0904**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_user_pause_count[47:32]

STAT_RX_INRANGEERR_LSB: 0908

Table 201: **STAT_RX_INRANGEERR_LSB: 0908**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_inrangeerr_count[31:0]

STAT_RX_INRANGEERR_MSB: 090C

Table 202: **STAT_RX_INRANGEERR_MSB: 090C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_inrangeerr_count[47:32]

STAT_RX_TRUNCATED_LSB: 0910

Table 203: **STAT_RX_TRUNCATED_LSB: 0910**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_truncated_count[31:0]

STAT_RX_TRUNCATED_MSB: 0914

Table 204: **STAT_RX_TRUNCATED_MSB: 0914**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_truncated_count[47:32]

STAT_RX_TEST_PATTERN_MISMATCH_LSB: 0918

Table 205: **STAT_RX_TEST_PATTERN_MISMATCH_LSB: 0918**

Bits	Default	Type	Signal
31:0	0	HIST	stat_rx_test_pattern_mismatch_count[31:0]

STAT_RX_TEST_PATTERN_MISMATCH_MSB: 091C

Table 206: **STAT_RX_TEST_PATTERN_MISMATCH_MSB: 091C**

Bits	Default	Type	Signal
15:0	0	HIST	stat_rx_test_pattern_mismatch_count[47:32]

Chapter 4

Designing with the Subsystem

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

General Design Guidelines

Use the Example Design as a Starting Point

Each release is delivered as a complete reference design that includes a sample test bench for simulation. Transceivers are included, targeted to the particular AMD device requested for that release. In most cases, you need to re-assign the transceivers according to the device pinout specific to your board layout. You might also wish to generate new custom transceivers using the AMD Vivado™ Design Suite with characteristics suited to your board.

Know the Degree of Difficulty

USXGMII IP Core designs are challenging to implement in any technology, and the degree of difficulty is further influenced by:

- Maximum system clock frequency
- Targeted device architecture
- Nature of your application

All USXGMII core implementations need 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 increase system performance in an FPGA design, keep all inputs and outputs registered between your application and the core. This means that all inputs and outputs from your application should come from, or connect to a flip-flop. While registering signals cannot be possible for all paths, it simplifies timing analysis and makes it easier for the Vivado Design Suite to place and route the design.

Recognize Timing Critical Signals

The timing constraints file that is provided with the example design for the core identifies the critical signals and the timing constraints that should be applied.

Make Only Allowed Modifications

The USXGMII IP Core is not user-modifiable. Do not make any modifications because these modifications can have adverse effects on system timing and protocol functionality. You can submit supported user configurations of the USXGMII core to AMD Technical Support for implementation.

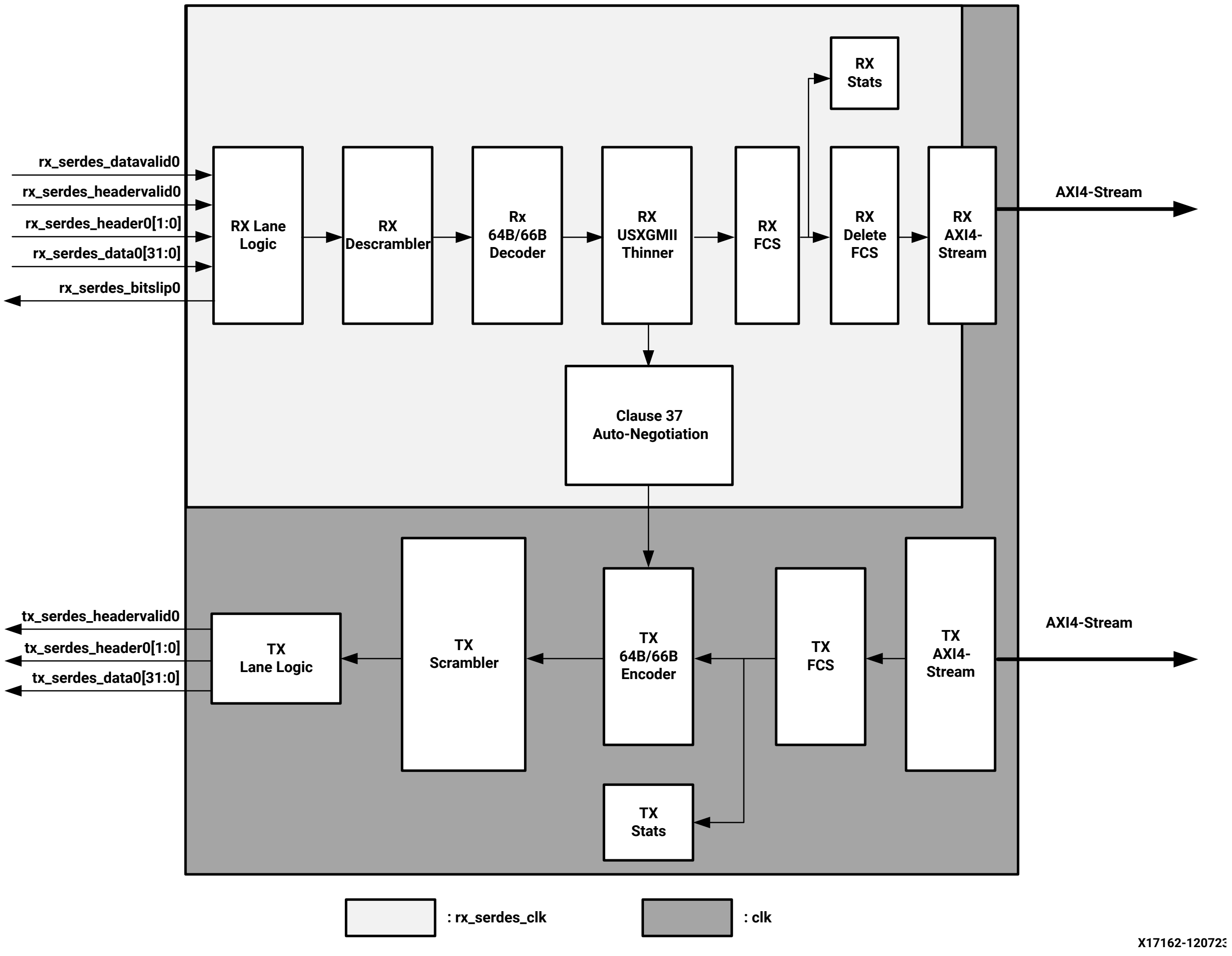
You are encouraged to modify the transceivers included with the example design. Use the latest GT Wizard which is part of the Vivado Design Suite. Some features that might need to be customized are the transceiver placement, reference clocks, and optional ports, among others.

Clocking and Resets

The 32-bit USXGMII IP core MAC is configured to operate its RX side on a single RX recovered clock (async gearbox option on the GTH or GTY transceiver for a bubble-less data path running with a 312.5 MHz clock).

Clocking and Reset

Figure 8: **Clocking the Reset**



The entire TX side and the AXI4-Stream interface run on a single 312.5 MHz clock. This clock is provided to user logic on the `tx_clk_out` port.

All signals between the USXGMII IP core and the user-side RX logic are synchronized to the positive edge of the `rx_clk_out` clock, which is set to a frequency of 312.5 MHz. When the RX FIFO is included, `rx_clk_out` will not be present and the RX path will be synchronized to `tx_clk_out`.

The IP core has a simple reset structure. The single `tx_reset` input is asserted High to reset the TX data path. This reset can be deasserted when the `clk` clock is stable.

The RX GT interface logic is reset by asserting the `rx_serdes_reset` signal. The `rx_reset` signal resets the internal RX data path. When the `clk` clock is stable, the `rx_reset` can be released. Following that when the `rx_serdes_clk` is stable the `rx_serdes_reset` can be released.

Support for IEEE Standard 1588v2

This section details the packet timestamping function of the USXGMII subsystem. The timestamping option must be specified at the time of generating the subsystem from the IP catalog or ordering the IP Core asynchronously.

Overview

Only two-step IEEE 1588v2 functionality is supported. When the core is generated with timestamping support, the two-step related ports are populated, and the core performs two-step time stamping functionality.

Ethernet frames are timestamped at both ingress and egress. The option can be used for implementing all kinds of IEEE 1588v2 clocks: Ordinary, Transparent, and Boundary. It can also be used for the generic timestamping of packets at the ingress and egress ports of a system. While this feature can be used for a variety of packet timestamping applications, the rest of this section assumes that you are also implementing the IEEE 1588v2 Precision Time Protocol (PTP).

IEEE 1588v2 defines a protocol for performing timing synchronization across a network. A 1588 network has a single master clock timing reference, usually selected through a best master clock algorithm. Periodically, this master samples its system timer reference counter, and transmits this sampled time value across the network using defined packet formats. This timer must be sampled (a timestamp) when the start of a 1588 timing packet is transmitted. Therefore, to achieve high synchronization accuracy over the network, accurate timestamps are required. If this sampled timer value (the timestamp) is placed into the packet that triggered the timestamp, this is known as one-step operation. Alternatively, the timestamp value can be placed into a follow up packet; this is known as two-step operation.

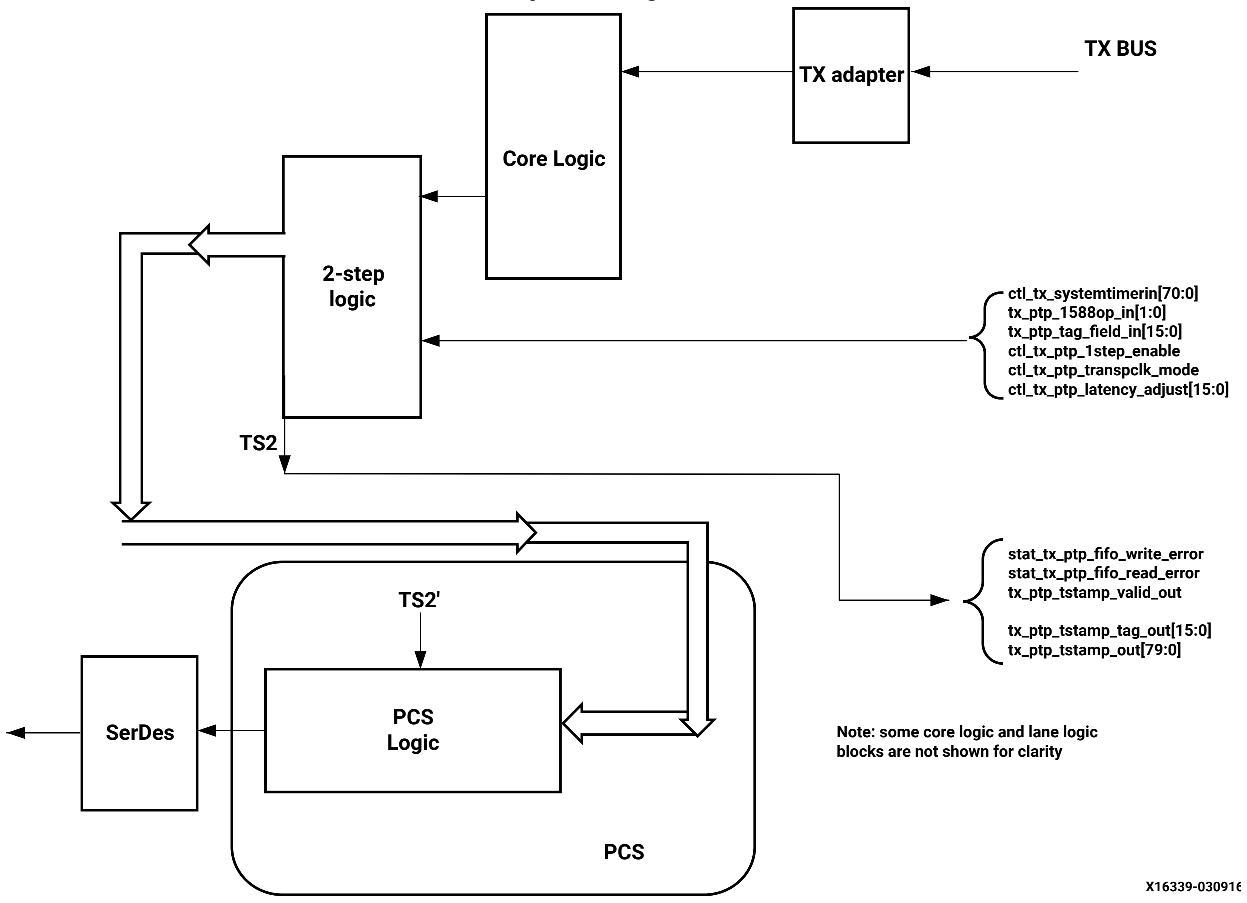
Other timing slave devices on the network receive these timing reference packets from the network timing master and attempt to synchronize their own local timer references to it. This mechanism relies on these Ethernet ports also taking timestamps (samples of their own local timer) when the 1588 timing packets are received. Further explanation of the operation of 1588 is out of scope of this document. This document now describes the 1588 hardware timestamping features of the subsystem.

The 1588 timer provided to the subsystem and the consequential timestamping taken from it are available in one of two formats:

- Time-of-Day (ToD) format: IEEE 1588-2008 format consisting of an unsigned 48-bit second field and a 32-bit nanosecond field.
- Correction Field format: IEEE 1588-2008 numerical format consisting of a 64-bit signed field representing nanoseconds multiplied by 216 (see IEEE 1588 clause 13.3.2.7). This timer must count from 0 through the full range up to 264 -1 before wrapping around.

Egress

Figure 9: **Egress**


TX BUS
TX adapter
Core Logic
2-step logic
ctl_tx_systemtimerin[70:0]
tx_ptp_1588op_in[1:0]
tx_ptp_tag_field_in[15:0]
ctl_tx_ptp_1step_enable
ctl_tx_ptp_transpclk_mode
ctl_tx_ptp_latency_adjust[15:0]
TS2
stat_tx_ptp_fifo_write_error
stat_tx_ptp_fifo_read_error
tx_ptp_tstamp_valid_out
tx_ptp_tstamp_tag_out[15:0]
tx_ptp_tstamp_out[79:0]
TS2'
SerDes
PCS Logic
Note: some core logic and lane logic blocks are not shown for clarity
PCS
X16339-030916

The diagram above shows the point in the TX datapath where the two-step operation is performed. The time stamp references are defined as follows:

- TS2: The output timestamp signal when a two-step operation is selected.
- TS2': The plane to which both timestamps are corrected.

TS2 has a correction applied so that it is referenced to the TS2' plane, depending on the value of the signal `ctl_tx_ptp_latency_adjust[15:0]`.

On the transmit side, a command field is provided by the client to the subsystem in parallel with the frame sent for transmission. This indicates, on a frame-by-frame basis, the 1588 function to perform (either no-operation or two-step). If using the ToD format, then for both one-step and two-step operation, the full captured 80-bit ToD timestamp is returned to the client logic using the additional ports defined in Port Descriptions. If using the Correction Field format, then for both one-step and two-step operation, the full captured 64-bit timestamp is returned to the client logic using the additional ports defined in Port Descriptions (with the upper bits of data set to zero as defined in the table).

Frame-by-Frame Timestamping Operation

The operational mode of the egress timestamping function is determined by the settings on the 1588 command port. The information contained within the command port indicates one of the following:

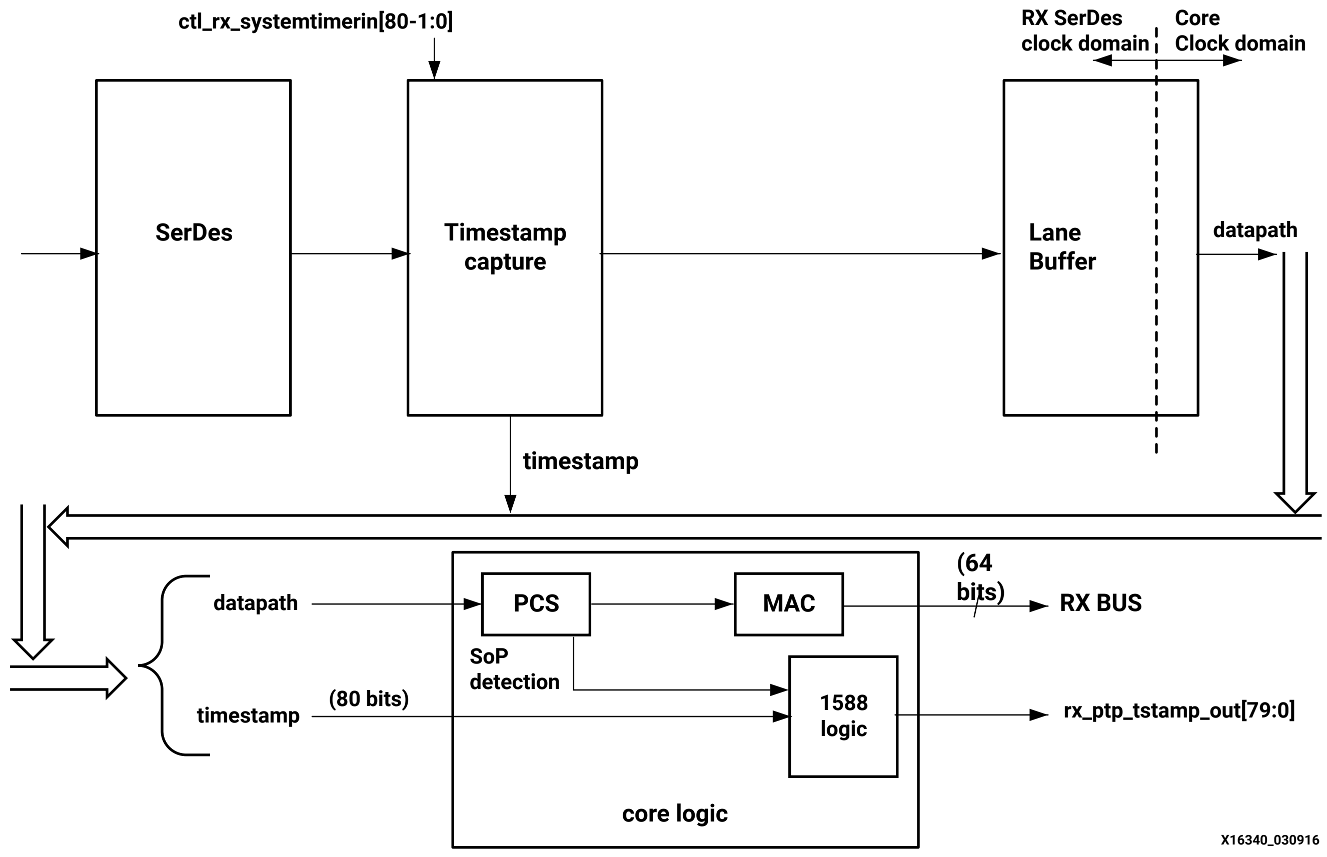
- No operation: the frame is not a PTP frame and no timestamp action needs to be taken.
- Two-step operation is required and a tag value (user-sequence ID) is provided as part of the command field; the frame must be timestamped, and the timestamp made available to the client logic, along with the provided tag value for the frame. The additional MAC transmitter ports provide this function.

Ingress

The ingress logic does not parse the ingress packets to search for 1588 precise timing protocol (PTP) frames. Instead, it takes a timestamp for every received frame and outputs this value to the user logic. The feature is always enabled, but the timestamp output can be ignored if you do not require this function.

Timestamps are filtered after the PCS decoder to retain only those timestamps corresponding to a Start of Packet (SOP). These 80-bit timestamps are output on the system side. The timestamp is valid during the S0P cycle and when ena_out = 1.

Figure 10: **Ingress**



Port Descriptions

The following table details the additional signals present when the packet timestamping feature is included.

Table 207: **1588v2 Port List and Descriptions**

Name	Direction	Clock Domain	Description
IEEE 1588 Interface – TX Path			
ctl_tx_systemtimerin[79:0]	Input	tx_serdes_clk	System timer input for the TX. In normal clock mode, the 32 LSBs carry nsec and the 48 MSBs carry seconds. In transparent clock mode, bit 63 is expected to be zero, bits 62:16 carry nanoseconds, and bits 15:0 carry fractional nanoseconds. Refer to IEEE 1588v2 for the representational definitions. This input must be in the TX SerDes clock domain.

Table 207: **1588v2 Port List and Descriptions** *(cont'd)*

Name	Direction	Clock Domain	Description
tx_ptp_tstamp_valid_out	Output	tx_clk_out	This bit indicates that a valid timestamp is being presented on the TX system interface.
tx_ptp_tstamp_tag_out[15:0]	Output	tx_clk_out	Tag output corresponding to tx_ptp_tag_field_in[15:0].
tx_ptp_tstamp_out[79:0]	Output	tx_clk_out	Timestamp for the transmitted packet SOP corresponding to the time at which it passed the capture plane. Time format same as timer input.
tx_ptp_1588op_in[1:0]	Input	tx_clk_out	The signal must be valid on the first cycle of the packet. • 2’b00 – No operation: no timestamp is taken and the frame is not modified. • 2’b10 – 2-step: a timestamp must be taken and returned to the client using the additional ports of 2-step operation. The frame itself is not modified. • 2’b01 and 2’b11 – Reserved: act as No operation.
ctl_tx_ptp_1step_enable	Input	tx_clk_out	This signal must be tied to 1’b0.
ctl_ptp_transpclk_mode	Input	tx_clk_out	When set to 1, this input places the timestamping logic into transparent clock mode.
tx_ptp_tag_field_in[15:0]	Input	tx_clk_out	The usage of this field is dependent on the 1588 operation. The signal must be valid on the first cycle of the packet. • For No operation, this field is ignored. • For two-step this field is a tag field. This tag value is returned to the client with the timestamp for the current frame using the additional ports of two-step operation. This tag value can be used by software to ensure that the timestamp can be matched with the PTP frame that it sent for transmission.

Table 207: **1588v2 Port List and Descriptions** *(cont'd)*

Name	Direction	Clock Domain	Description
ctl_tx_ptp_latency_adjust[31:0]	Input	tx_clk_out	This bus can be used to adjust the one-step T timestamp with respect to the two-step timestamp. Only the LSB 16-bits are used.
stat_tx_ptp_fifo_write_error	Output	tx_clk_out	Transmit PTP FIFO write error. A value of 1 on this status indicates that an error occurred during the PTP Tag write. A TX Path reset is required to clear the error.
stat_tx_ptp_fifo_read_error	Output	tx_clk_out	Transmit PTP FIFO read error. A value of 1 on this status indicates that an error occurred during the PTP Tag read. A TX Path reset is required to clear the error.
tx_ptp_rxtstamp_in	Input	tx_clk_out	At present this port is not used. Must be tied to 0x0.
IEEE 1588 Interface – RX Path			
ctl_rx_systemtimerin[79:0]	Input	rx_serdes_clk	System timer input for the RX. Same time format as the TX. This input must be in the same clock domain as the RX SerDes.
rx_ptp_tstamp_out[79:0]	Output	rx_clk_out	Timestamp for the received packet SOP corresponding to the time at which it passed the capture plane. This signal is valid on the first cycle of the packet.

Pause Processing

The USXGMII core provides a comprehensive mechanism for pause packet termination and generation. The TX and RX have independent interfaces for processing pause information as described in this section.

TX Pause Generation

You can request a pause packet to be transmitted using the `ctl_tx_pause_req[8:0]` and `ctl_tx_pause_enable[8:0]` input buses. Bit [8] corresponds to global pause packets and bits [7:0] correspond to priority pause packets.

Each bit of this bus must be held at a steady state for a minimum of 16 cycles before the next transition.

CAUTION! *Requesting both global and priority pause packets at the same time results in unpredictable behaviour and must be avoided.*

The contents of the pause packet are determined using the following input pins.

Global pause packets:

```
ctl_tx_da_gpp[47:0]
ctl_tx_sa_gpp[47:0]
ctl_tx_ethertype_gpp[15:0]
ctl_tx_opcode_gpp[15:0]
ctl_tx_pause_quanta8[15:0]
```

Priority pause packets:

```
ctl_tx_da_ppp[47:0]
ctl_tx_sa_ppp[47:0]
ctl_tx_ethertype_ppp[15:0]
ctl_tx_opcode_ppp[15:0]
ctl_tx_pause_quanta0[15:0]
ctl_tx_pause_quanta1[15:0]
ctl_tx_pause_quanta2[15:0]
ctl_tx_pause_quanta3[15:0]
ctl_tx_pause_quanta4[15:0]
ctl_tx_pause_quanta5[15:0]
ctl_tx_pause_quanta6[15:0]
ctl_tx_pause_quanta7[15:0]
```

The USXGMII core automatically calculates and adds the FCS to the packet. For priority pause packets the USXGMII core also automatically generates the enable vector based on the priorities that are requested.

To request a pause packet, you must set the corresponding bit of the `ctl_tx_pause_req[8:0]` and `ctl_tx_pause_enable[8:0]` bus to a 1 and keep it at 1 for the duration of the pause request (that is, if these inputs are set to 0, all pending pause packets are canceled). The USXGMII core transmits the pause packet immediately after the current packet in flight is completed.

IMPORTANT! *Each bit of this bus must be held at a steady state for a minimum of 16 cycles before the next transition.*

To retransmit pause packets, the USXGMII core maintains a total of nine independent timers; one for each priority and one for global pause. These timers are loaded with the value of the corresponding input buses. After a pause packet is transmitted the corresponding timer is loaded with the corresponding value of the `ctl_tx_pause_refresh_timer[8:0]` input bus. When a timer times out, another packet for that priority (or global) is transmitted as soon as the current packet in flight is completed. Additionally, you can manually force the timers to 0, and therefore force a retransmission, by setting the `ctl_tx_resend_pause` input to 1 for one clock cycle.

To reduce the number of pause packets for priority mode operation, a timer is considered timed out if any of the other timers time out. Additionally, while waiting for the current packet in flight to be completed, any new timer that times out or any new requests are merged into a single pause frame. For example, if two timers are counting down, and you send a request for a third priority, the two timers are forced to be timed out and a pause packet for all three priorities is sent as soon as the current in-flight packet (if any) is transmitted. Similarly, if one of the two timers times out without an additional request, both timers are forced to be timed out and a pause packet for both priorities is sent as soon as the current in-flight packet (if any) is transmitted.

You can stop pause packet generation by setting the appropriate bits of `ctl_tx_pause_req[8:0]` or `ctl_tx_pause_enable[8:0]` to 0.

RX Pause Termination

The USXGMII IP Core terminates global and priority pause frames and provides a simple hand-shaking interface to allow user logic to respond to pause packets.

Determining Pause Packets

There are three steps in determining pause packets:

1. Checks are performed to see if a packet is a global or a priority control packet.

   ***Note*:** Packets that pass step 1 are forwarded to you only if `ctl_rx_forward_control` is set to 1.
2. If step 1 passes, the packet is checked to determine if it is a global pause packet.
3. If step 2 fails, the packet is checked to determine if it is a priority pause packet.

For Step 1, the following pseudo code shows the checking function:

```
assign da_match_gcp = (!ctl_rx_check_mcast_gcp && !ctl_rx_check_ucast_gcp)
|| ((DA == ctl_rx_pause_da_ucast) && ctl_rx_check_ucast_gcp) || ((DA ==
48'h0180c2000001) && ctl_rx_check_mcast_gcp);
assign sa_match_gcp = !ctl_rx_check_sa_gcp || (SA == ctl_rx_pause_sa);
assign etype_match_gcp = !ctl_rx_check_etype_gcp || (ETYPE ==
ctl_rx_etype_gcp);
assign opcode_match_gcp = !ctl_rx_check_opcode_gcp || ((OPCODE >=
ctl_rx_opcode_min_gcp) && (OPCODE <= ctl_rx_opcode_max_gcp));
assign global_control_packet = da_match_gcp && sa_match_gcp &&
etype_match_gcp && opcode_match_gcp && ctl_rx_enable_gcp;
assign da_match_pcp = (!ctl_rx_check_mcast_pcp && !ctl_rx_check_ucast_pcp)
|| ((DA == ctl_rx_pause_da_ucast) && ctl_rx_check_ucast_pcp) || ((DA ==
ctl_rx_pause_da_mcast) && ctl_rx_check_mcast_pcp);
assign sa_match_pcp = !ctl_rx_check_sa_pcp || (SA == ctl_rx_pause_sa);
assign etype_match_pcp = !ctl_rx_check_etype_pcp || (ETYPE ==
ctl_rx_etype_pcp);
assign opcode_match_pcp = !ctl_rx_check_opcode_pcp || ((OPCODE >=
ctl_rx_opcode_min_pcp) && (OPCODE <= ctl_rx_opcode_max_pcp));
assign priority_control_packet = da_match_pcp && sa_match_pcp &&
etype_match_pcp && opcode_match_pcp && ctl_rx_enable_pcp;
assign control_packet = global_control_packet || priority_control_packet;
```

In the code above, DA is the destination address, SA is the source address, OPCODE is the opcode and ETYPE is the ethertype/length field that are extracted from the incoming packet.

For Step 2, the following pseudo code shows the checking function:

```
assign da_match_gpp = (!ctl_rx_check_mcast_gpp && !ctl_rx_check_ucast_gpp)
|| ((DA == ctl_rx_pause_da_ucast) && ctl_rx_check_ucast_gpp) || ((DA ==
48'h0180c2000001) && ctl_rx_check_mcast_gpp);
assign sa_match_gpp = !ctl_rx_check_sa_gpp || (SA == ctl_rx_pause_sa);
assign etype_match_gpp = !ctl_rx_check_etype_gpp || (ETYPE ==
ctl_rx_etype_gpp);
assign opcode_match_gpp = !ctl_rx_check_opcode_gpp || (OPCODE ==
ctl_rx_opcode_gpp);
assign global_pause_packet = da_match_gpp && sa_match_gpp &&
etype_match_gpp && opcode_match_gpp && ctl_rx_enable_gpp;
```

In the code above, DA is the destination address, SA is the source address, OPCODE is the opcode and ETYPE is the ethertype/length field that are extracted from the incoming packet.

For Step 3, the following pseudo code shows the checking function:

```
assign da_match_ppp = (!ctl_rx_check_mcast_ppp && !ctl_rx_check_ucast_ppp)
|| ((DA == ctl_rx_pause_da_ucast) && ctl_rx_check_ucast_ppp) || ((DA ==
ctl_rx_pause_da_mcast) && ctl_rx_check_mcast_ppp);
assign sa_match_ppp = !ctl_rx_check_sa_ppp || (SA == ctl_rx_pause_sa);
assign etype_match_ppp = !ctl_rx_check_etype_ppp || (ETYPE ==
ctl_rx_etype_ppp);
assign opcode_match_ppp = !ctl_rx_check_opcode_ppp || (OPCODE ==
ctl_rx_opcode_ppp);
assign priority_pause_packet = da_match_ppp && sa_match_ppp &&
etype_match_ppp && opcode_match_ppp && ctl_rx_enable_ppp;
```

In the code above, DA is the destination address, SA is the source address, OPCODE is the opcode and ETYPE is the ethertype/length field that are extracted from the incoming packet.

User Interface

A simple handshaking protocol is used to alert you of the reception of pause packets using the `ctl_rx_pause_enable[8:0]`, `stat_rx_pause_req[8:0]` and `ctl_rx_pause_ack[8:0]` buses. For these buses, bit [8] corresponds to global pause packets and bits [7:0] correspond to priority pause packets.

The following steps occur when a pause packet is received:

1. If the corresponding bit of `ctl_rx_pause_enable[8:0]` is 0, the quanta is ignored and the hard CMAC stays in step 1.

   ***Note*:** Otherwise, the corresponding bit of the `stat_rx_pause_req[8:0]` bus is set to 1, and the received quanta is loaded into a timer.

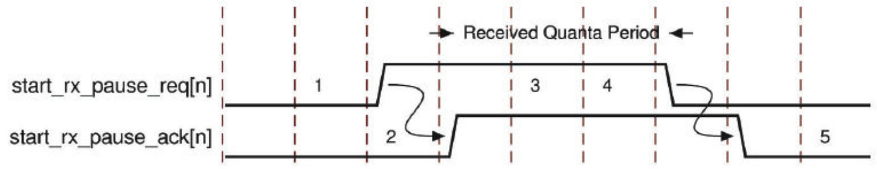
2. If `ctl_rx_check_ack` input is 1, the core waits for you to set the appropriate bit of the `ctl_rx_pause_ack[8:0]` bus to 1.

3. After you set the proper bit of `ctl_rx_pause_ack[8:0]` to 1, or if `ctl_rx_check_ack` is 0, the core starts counting down the timer.
4. When the timer times out, the core sets the appropriate bit of `stat_rx_pause_req[8:0]` back to 0.
5. If `ctl_rx_check_ack` input is 1, the operation is complete when you set the appropriate bit of `ctl_rx_pause_ack[8:0]` back to 0.

If you do not set the appropriate bit of `ctl_rx_pause_ack[8:0]` back to 0, the core deems the operation complete after 32 clock cycles.

These steps are demonstrated in the following figure with each step shown on the waveform.

Figure 11: **RX Pause Interface Example**



If at any time during Step 2 to Step 5 a new pause packet is received, the timer is loaded with the newly acquired quanta value and the process continues.

Chapter 5

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 Subsystem

This section includes information about using AMD tools to customize and generate the subsystem in the AMD Vivado™ Design Suite.

If you are customizing and generating the subsystem 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 Vivado IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

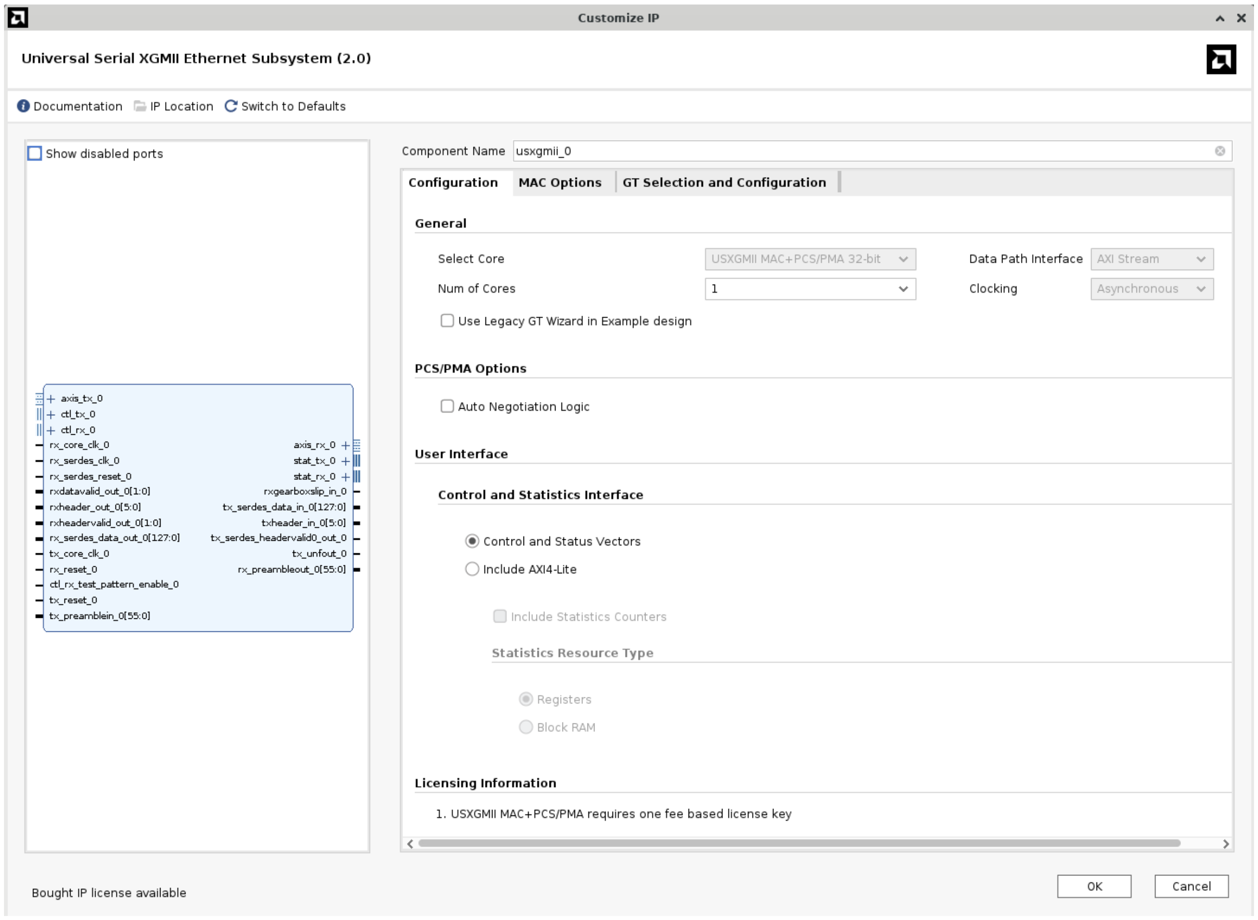
For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

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

Configuration Tab

The Configuration tab provides the basic core configuration options. Default values are pre-populated in all tabs.

Figure 12: **Configuration Tab (Versal Adaptive SoC)**



Universal Serial XGMII Ethernet Subsystem (2.0)

Show disabled ports

Component Name usxgmii_0

General

Select Core: USXGMII MAC+PCS/PMA 32-bit

Data Path Interface: AXI Stream

Num of Cores: 1

Clocking: Asynchronous

Use Legacy GT Wizard in Example design

PCS/PMA Options

Auto Negotiation Logic

User Interface

Control and Statistics Interface

[✓ Control and Status Vectors] Include AXI4-Lite

Include Statistics Counters

Statistics Resource Type

[✓ Registers] Block RAM

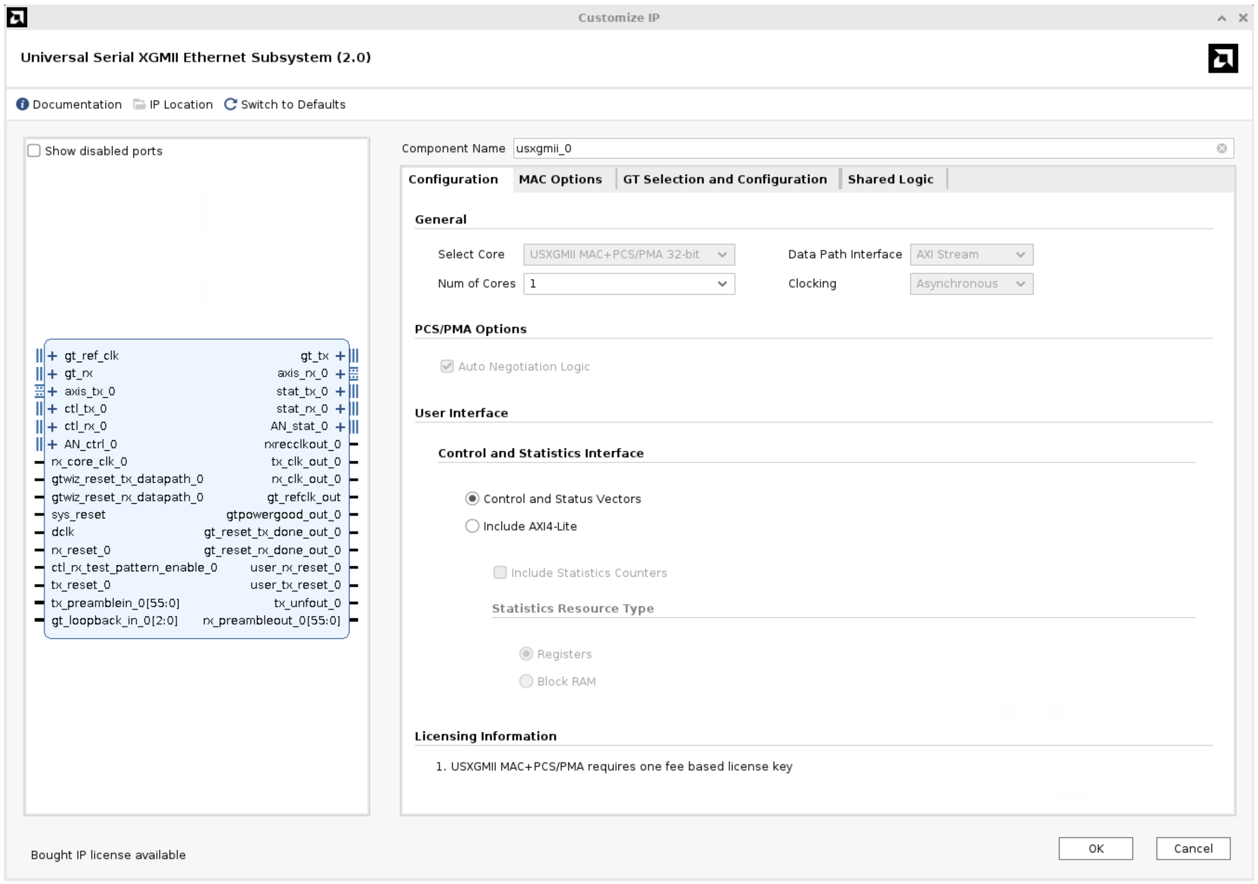
Licensing Information

1. USXGMII MAC+PCS/PMA requires one fee based license key

Bought IP license available

The following screenshot details the Configuration Tab for AMD UltraScale™ / AMD UltraScale+™.

Figure 13: **Configuration Tab (UltraScale/UltraScale+)**



Universal Serial XGMII Ethernet Subsystem (2.0)

Component Name: usxgmii_0

General

Select Core: USXGMII MAC+PCS/PMA 32-bit
Num of Cores: 1
Data Path Interface: AXI Stream
Clocking: Asynchronous

PCS/PMA Options

[✓] Auto Negotiation Logic

User Interface

Control and Statistics Interface

[● Control and Status Vectors]
○ Include AXI4-Lite

☐ Include Statistics Counters

Statistics Resource Type

[● Registers]
○ Block RAM

Licensing Information

1. USXGMII MAC+PCS/PMA requires one fee based license key

Bought IP license available

Table 208: **Configuration Options**

Option	Values	Default
General		
Select Core	USXGMII MAC+PCS/PMA 32-bit	USXGMII MAC+PCS/PMA 32-bit
Data Path Interface	AXI4-Stream [1]	AXI4-Stream
Num of Cores[6]	1 2 3 4	1
Clocking	Asynchronous[2]	Asynchronous
PCS/PMA Options		
Auto Negotiation Logic	Checked and Unchecked[3]	Checked
Control and Statistics Interface		
Control and Statistics interface	Control and Status Vectors Include AXI4-Lite	Control and Status Vectors
Include Statistics Counters	Checked and Unchecked[4]	Checked
Statistics Resource Type[5]	Registers, block RAM	Registers

Table 208: **Configuration Options** *(cont'd)*

Option	Values	Default
Use Legacy GT Wizard in Example Design[7]	Checked, Unchecked	Unchecked

Notes:

1. The AXI4-Stream interface is visible and is the only available option.
2. Asynchronous clocking is the only option available. In this mode TXOUTCLK sources the TXUSRCLK and RXOUTCLK sources the RXUSRCLK.
3. The USXGMII standard mandates that the auto negotiation logic is always enabled.

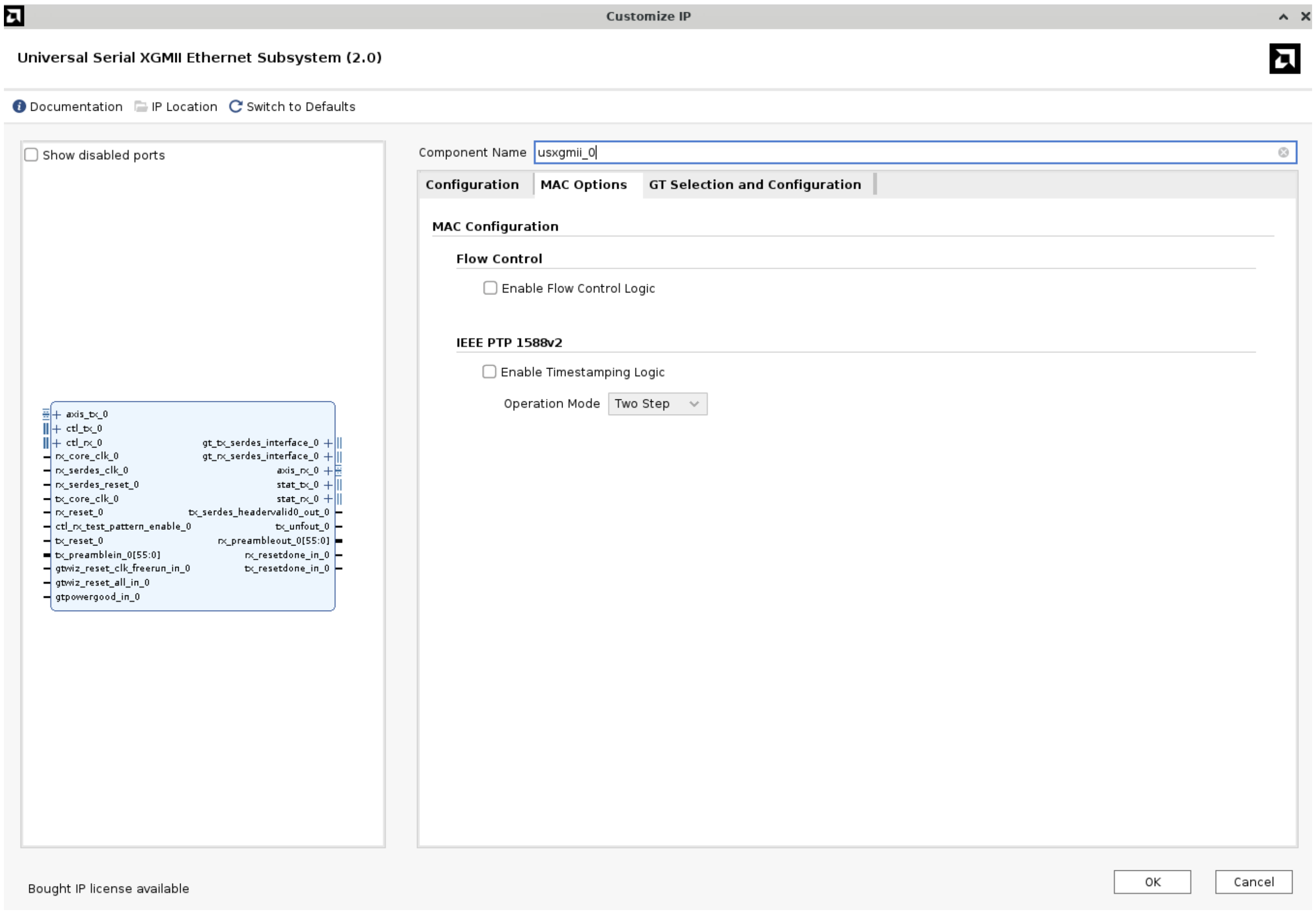
   For UltraScale/UltraScale+ devices, the auto negotiation logic is always enabled. You can assert the core input signal, `ctl_umii_an_bypass`, to bypass auto-negotiation logic and run the core at a fixed data-rate.

   For AMD Versal™ devices, the default configuration for USXGMII IP is with Auto-Negotiation disabled. Enable the Auto-Negotiation Logic option to enable auto-negotiation.
4. The Statistics Counters are available in the register map only when you enable the Include Statistics Counters option. Otherwise, the Statistics Counters are not available.
5. Statistics Resource Type block RAM option inclusion is planned for future release.
6. The Number of Cores will be 1 only for the Versal devices and multi-core support will not be supported as GT will always be in the example design/outside the core.
7. Select the required GT Wizard. For the legacy GT Wizard, enable the Use Legacy GT Wizard in Example Design option and disable it for the GT Wizard Subsystem (gtwiz_Versal IP) selection. This option only available on Versal devices.

MAC Options Tab

The GT Selection and Configuration tab enables you to configure the serial transceiver features of the core.

Figure 14: **MAC Options Tab**



Customize IP

Universal Serial XGMII Ethernet Subsystem (2.0)

Documentation | IP Location | Switch to Defaults

Show disabled ports

Component Name: usxgmii_0

Configuration | MAC Options | GT Selection and Configuration

MAC Configuration

Flow Control

- [ ] Enable Flow Control Logic

IEEE PTP 1588v2

- [ ] Enable Timestamping Logic

Operation Mode: Two Step

Bought IP license available

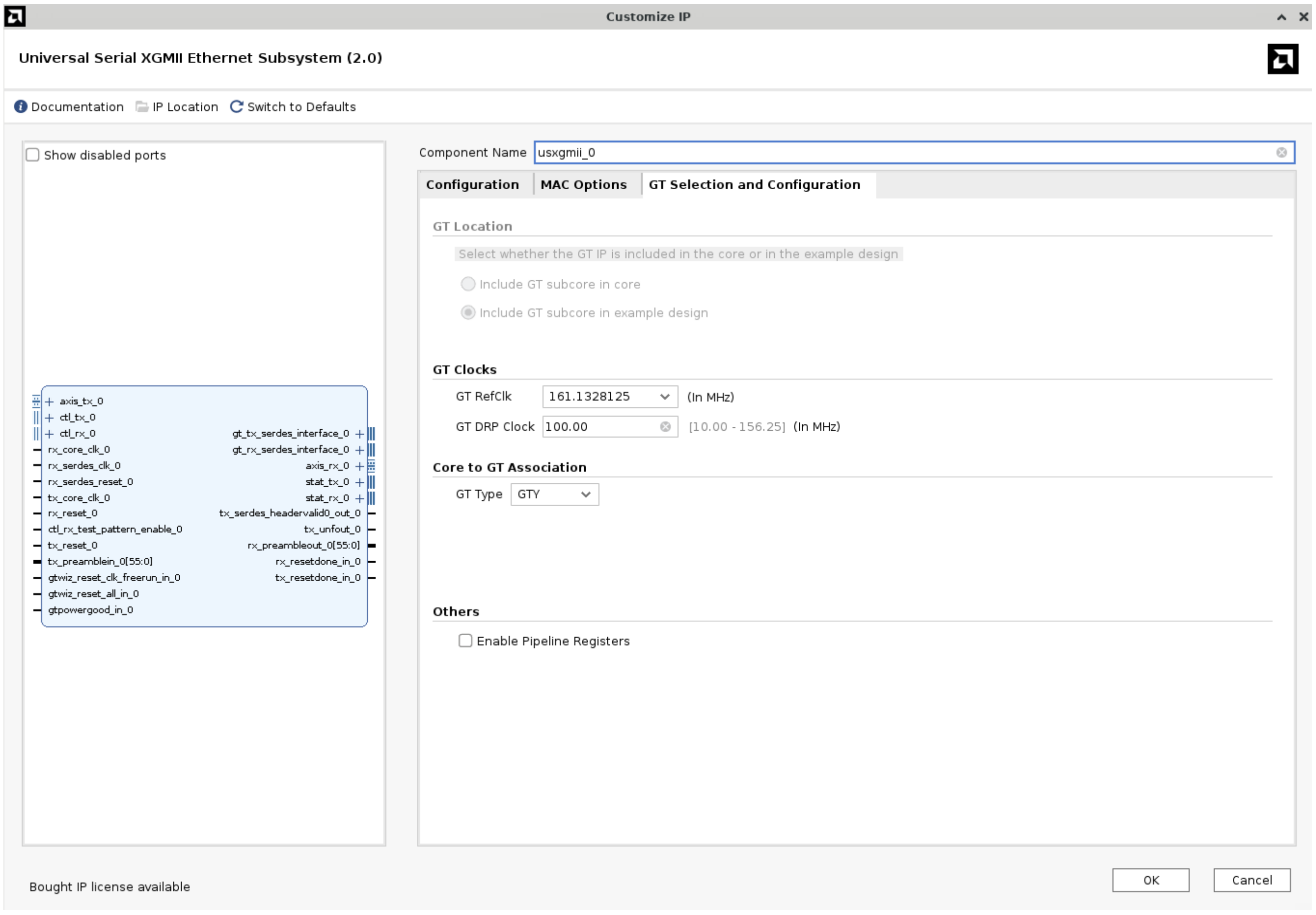
Table 209: **MAC Options**

Option	Values	Default
Flow Control		
Enable Flow Control Logic	Checked, Unchecked	Unchecked
IEEE PTP 1588v2		
Enable Timestamping Logic	Checked, Unchecked	Unchecked
Operation Mode	Two Step	Two Step

GT Selection and Configuration Tab

The GT Selection and Configuration tab enables you to configure the serial transceiver features of the core.

Figure 15: **GT Selection and Configuration Tab (Versal Adaptive SoC)**



Universal Serial XGMII Ethernet Subsystem (2.0)

☐ Show disabled ports

Component Name: usxgmii_0

GT Location

Select whether the GT IP is included in the core or in the example design

○ Include GT subcore in core

● Include GT subcore in example design

GT Clocks

GT RefClk: 161.1328125 (In MHz)

GT DRP Clock: 100.00 [10.00 - 156.25] (In MHz)

Core to GT Association

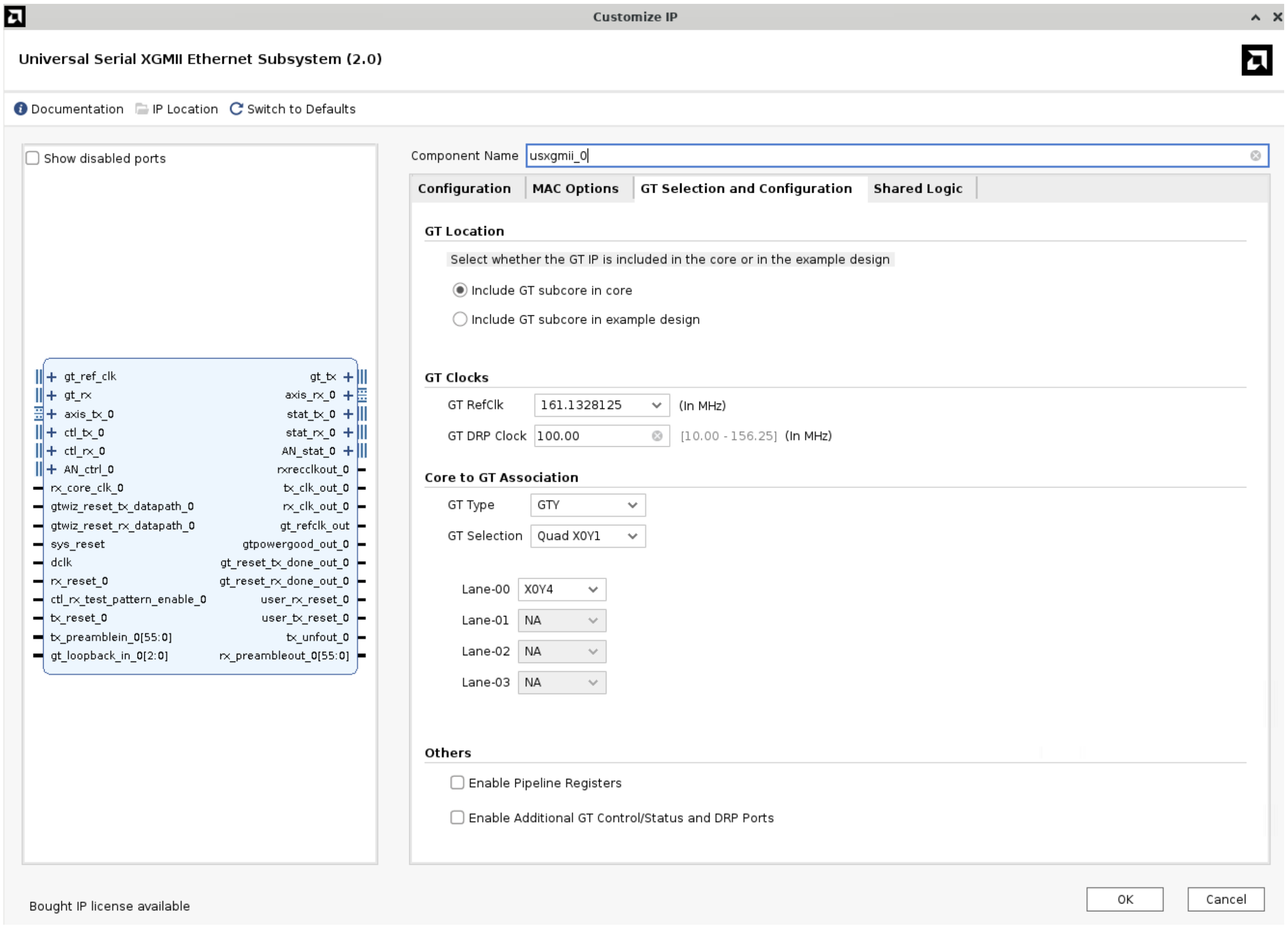
GT Type: GTY

Others

☐ Enable Pipeline Registers

Bought IP license available

Figure 16: **GT Selection and Configuration Tab (UltraScale/UltraScale+)**



Universal Serial XGMII Ethernet Subsystem (2.0)

Show disabled ports [ ]

Component Name: usxgmii_0

GT Location

Select whether the GT IP is included in the core or in the example design

[✓ Include GT subcore in core]

Include GT subcore in example design

GT Clocks

GT RefClk: 161.1328125 (In MHz)

GT DRP Clock: 100.00 [10.00 - 156.25] (In MHz)

Core to GT Association

GT Type: GTY

GT Selection: Quad X0Y1

Lane-00: X0Y4

Lane-01: NA

Lane-02: NA

Lane-03: NA

Others

[ ] Enable Pipeline Registers

[ ] Enable Additional GT Control/Status and DRP Ports

Bought IP license available

Table 210: **GT Selection and Configuration Options**

Option	Values	Default
GT Location		
Select whether the GT IP is included in the core or in the example design	Include GT subcore in core Include GT subcore in example design	Include GT subcore in core
GT Clocks		
GT RefClk (In MHz)	103.125 128.90625 156.25 161.1328125 206.25 257.8125 309.375 312.5 322.265625	161.1328125
GT DRP Clock (In MHz)	10 – 156.25 MHz	100

Table 210: **GT Selection and Configuration Options** *(cont'd)*

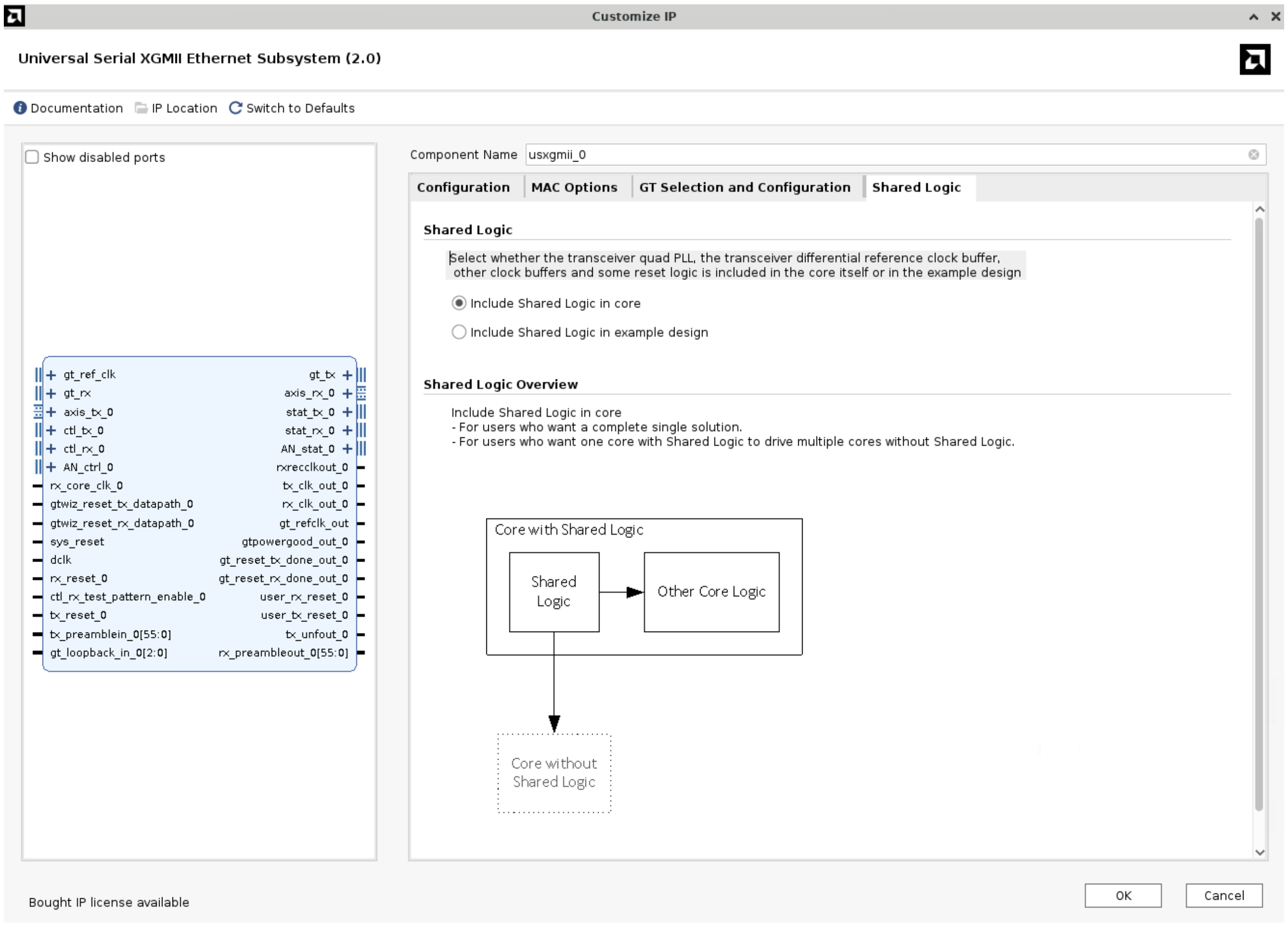
Option	Values	Default
Core to GT Association		
GT Type	GTY GTH	GTY
GT Selection	Options based on device/ package Quad groups. For example: Quad X0Y1 Quad X0Y2 Quad X0Y3 ...	Quad X0Y1
Lane-00 to Lane-03	Auto filled based on device/ package. For example, if Num of Core = 4, and GT Selection = Quad X0Y1, four lanes are: X0Y4 X0Y5 X0Y6 X0Y7	Checked
Others		
Enable Pipeline Registers	Checked, Unchecked	Unchecked
Enable Additional GT Control/ Status and DRP Ports	Checked, Unchecked	Unchecked

Shared Logic Tab

The Shared Logic tab allows you to use shared logic in either the core or the example design.

***Note*:** The Shared Logic tab is available only for AMD UltraScale™ and AMD UltraScale+™ devices.

Figure 17: **Shared Logic Tab (UltraScale/UltraScale+)**



Component Name: usxgmii_0

Shared Logic

Select whether the transceiver quad PLL, the transceiver differential reference clock buffer, other clock buffers and some reset logic is included in the core itself or in the example design

[✓ Include Shared Logic in core]

Include Shared Logic in example design

Shared Logic Overview

Include Shared Logic in core
- For users who want a complete single solution.
- For users who want one core with Shared Logic to drive multiple cores without Shared Logic.

Table 211: **Shared Logic Options**

Options	Default
Include Shared Logic in core Include Shared Logic in example design	Include Shared Logic in core

Output Generation

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

User Parameters

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

Table 212: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value
CORE USXGMII MAC+PCS/PMA 32-bit	CORE USXGMII MAC+PCS/PMA 32-bit	USXGMII MAC+PCS/PMA 32-bit
NUM_OF_CORES 1 2 3 4	NUM_OF_CORES 1 2 3 4	1
CLOCKING Asynchronous	CLOCKING Asynchronous	"Asynchronous"
DATA_PATH_INTERFACE AXI4-Stream	DATA_PATH_INTERFACE AXI4-Stream	"AXI4-Stream"
INCLUDE_AUTO_NEG_LOGIC True False	INCLUDE_AUTO_NEG_LOGIC 1 0	1
INCLUDE_AXI4_INTERFACE Control and Status Vectors Include AXI4-Lite	INCLUDE_AXI4_INTERFACE 0 1	0
INCLUDE_STATISTICS_COUNTER True False	INCLUDE_STATISTICS_COUNTER 0 1	1
ENABLE_FLOW_CONTROL_LOGIC True False	ENABLE_FLOW_CONTROL_LOGIC 0 1	0
ENABLE_TIME_STAMPING True False	ENABLE_TIME_STAMPING 0 1	0
PTP_OPERATION_MODE Two Step	PTP_OPERATION_MODE 2	2
PTP_CLOCKING_MODE Ordinary Clock Transparent Clock	PTP_CLOCKING_MODE 0 1	0
TX_LATENCY_ADJUST TRUE FALSE	TX_LATENCY_ADJUST 1 0	0
ENABLE_VLANE_ADJUST_MODE TRUE FALSE	ENABLE_VLANE_ADJUST_MODE 1 0	0
GT_LOCATION Include GT subcore in core Include GT subcore in example design	GT_LOCATION 1 0	1

Table 212: **Vivado IDE Parameter to User Parameter Relationship** *(cont'd)*

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value
GT_REF_CLK_FREQ 161.1328125 195.3125 201.4160156 257.8125 322.265625	GT_REF_CLK_FREQ 161.1328125 195.3125 201.4160156 257.8125 322.265625	161.1328125
GT_DRP_CLK 10 to 250	GT_DRP_CLK 10 to 250	100
GT_TYPE GTY GTH	GT_TYPE GTY GTH	GTY
GT_GROUP_SELECT (options based of device/package)	GT_GROUP_SELECT (options based of device/package)	X0Y0
LANE1_GT_LOC X0Y0 X0Y1 X0Y2 X0Y3	LANE1_GT_LOC X0Y0 X0Y1 X0Y2 X0Y3	
LANE2_GT_LOC X0Y1 X0Y2 X0Y3	LANE2_GT_LOC X0Y1 X0Y2 X0Y3	
LANE3_GT_LOC X0Y2 X0Y3	LANE3_GT_LOC X0Y2 X0Y3	
LANE4_GT_LOC X0Y3	LANE4_GT_LOC X0Y3	
ENABLE_PIPELINE_REG TRUE FALSE	ENABLE_PIPELINE_REG 1 0	0
ADD_GT_CNTRL_STS_PORTS TRUE FALSE	ADD_GT_CNTRL_STS_PORTS 1 0	0
INCLUDE_SHARED_LOGIC Include Shared Logic in core Include Shared Logic in example design	INCLUDE_SHARED_LOGIC 1 0	1
FAST_SIM_MODE TRUE FALSE	FAST_SIM_MODE 1 0	0

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Constraining the Subsystem

This section contains information about constraining the subsystem.

Required Constraints

The USXGMII IP core is delivered with the AMD XDC constraints file.

Device, Package, and Speed Grade Selections

The USXGMII IP core is available for all AMD UltraScale™, AMD UltraScale+™ and AMD Versal™ adaptive SoC devices and transceivers.

Clock Frequencies

The USXGMII IP core has two clocks. Both clocks are 312.5 MHz. The `clk` provides a clock to the TX and RX AXI4-Stream datapath. This clock must be the user interface clock generated by the GT TXUSRCLK output. The `rx_serdes_clk` is generated by the GTs RX interface as the RXUSRCLK output.

Clock Management

Not Applicable

Clock Placement

Not Applicable

Banking

Not Applicable

Transceiver Placement

Not Applicable

I/O Standard and Placement

Not Applicable

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).

Simulation Speed Up

The example design contains wait timers. A define SIM_SPEED_UP is available to improve simulation time by speeding up these wait times.

VCS

Use the vlogan option: +define+SIM_SPEED_UP.

ModelSim

Use the vlog option: +define+SIM_SPEED_UP.

Questa Advanced Simulator

Use the vlog option: +define+SIM_SPEED_UP.

Vivado Simulator

Use the xvlog option: -d SIM_SPEED_UP.

Xcelium Parallel Simulator

Use the xmvlog option: +define+SIM_SPEED_UP.

Riviera-PRO Simulator

Use the vlog option: +define+SIM_SPEED_UP.

Synthesis and Implementation

For details about synthesis and implementation, see *Vivado Design Suite User Guide: Designing with IP* (UG896).

Chapter 6

Example Design

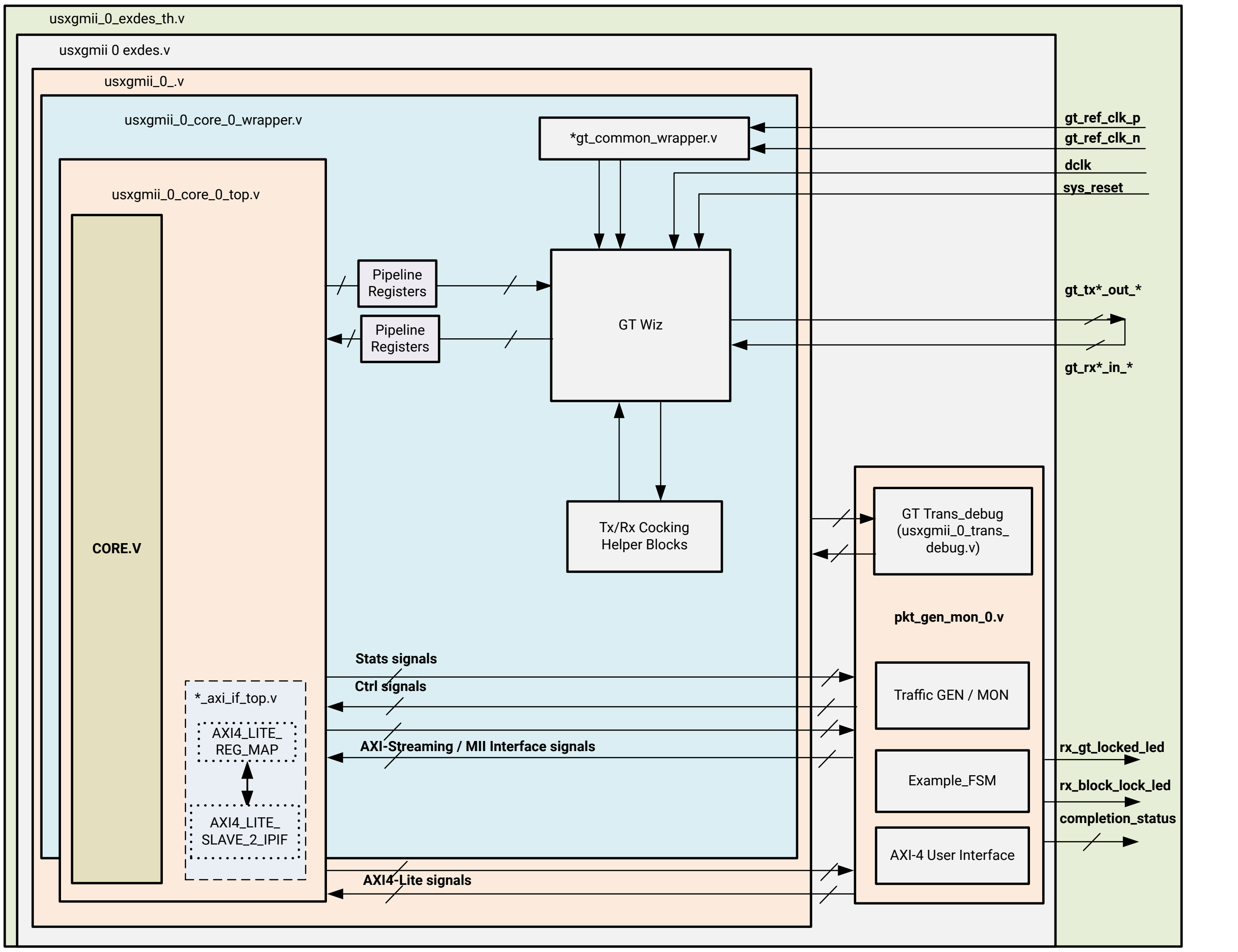
This chapter contains information about the example design provided in the AMD Vivado™ Design Suite when using the AMD Vivado™ Integrated Design Environment (IDE).

Overview

The following diagram shows the instantiation of various modules and their hierarchy for a single core configuration of usxgmii_0 example design when the GT (serial transceiver) is inside the IP core. (Serial Transceiver is always a part of the example design for AMD Versal™ adaptive SoC.)

Sync registers and pipeline registers are used for to synchronize the data between the core and the GT. Clocking helper blocks are used to generate the required clock frequency for the core.

Figure 18: **Single Core Example Design Hierarchy**



X00037-071425

AXI4-Stream is available for datapath interface and AXI4-Lite is available for control and statistics interface.

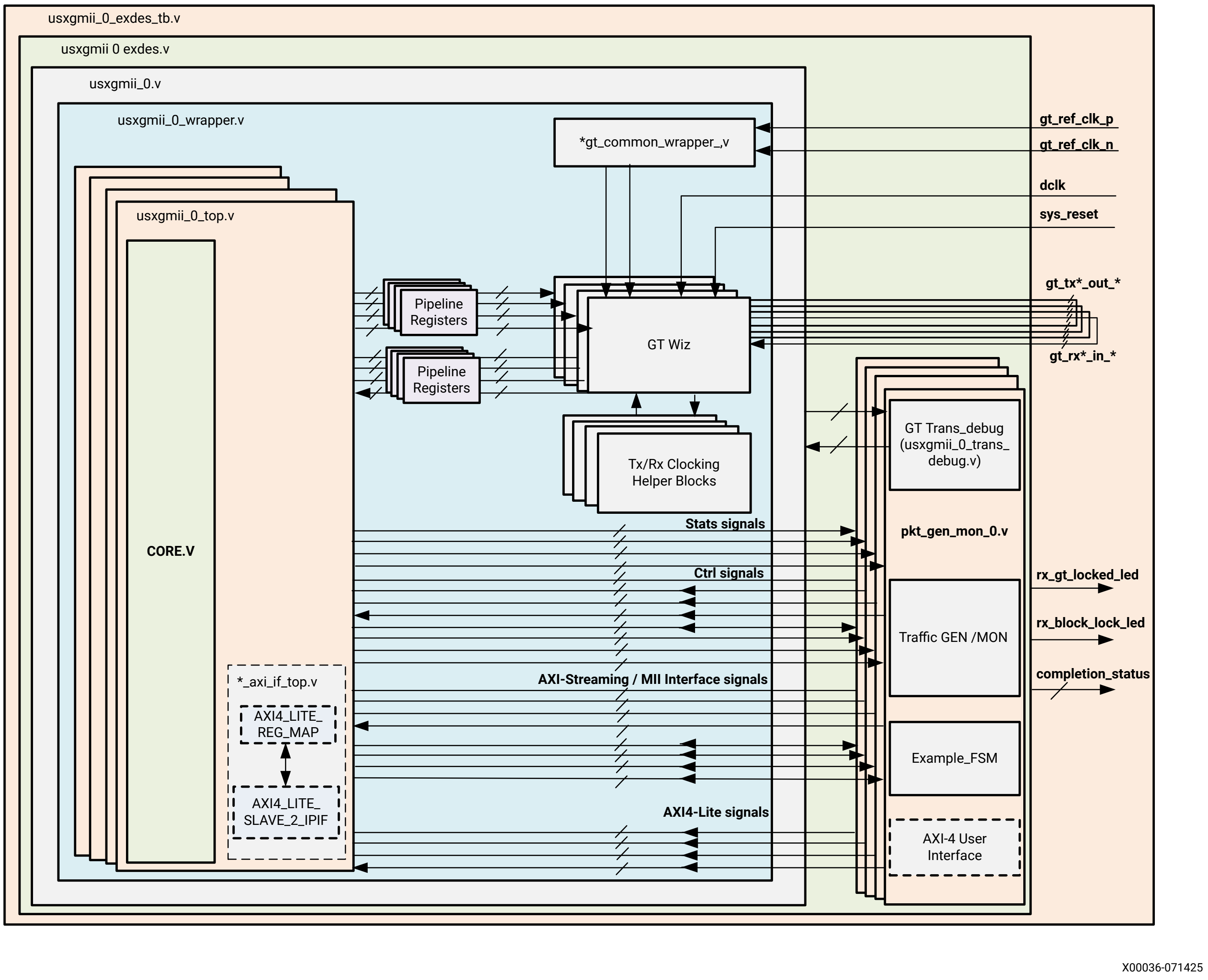
The `usxgmii_0_pkt_gen_mon` module is used to generate the data packets for sanity testing. The packet generation and checking is controlled by an FSM module.

The Example Design consists of these optional modules:

- usxgmii _0_trans_debug: This module is present in the example design when you enable the Additional GT Control and Status Ports check box from the GT Selection and Configuration tab in AMD Vivado™ IDE. This module brings out all the GT channel dynamic reconfiguration port (DRP) ports, and some control and status ports of the transceiver module out of the USXGMII core.
- Retiming registers: When you select the Enable Retiming Register option from the GT Selection and Configuration tab, it includes a single stage pipeline register between the core and the GT to ease timing, using the `gt_txusrclk2` and `gt_rxusrclk2` for TX and RX paths respectively. However, by default two-stage registering is done for the signals between the GT and the core.

The following diagram shows the instantiation of various modules and their hierarchy for the multiple core configuration of the usxgmii_0 example design.

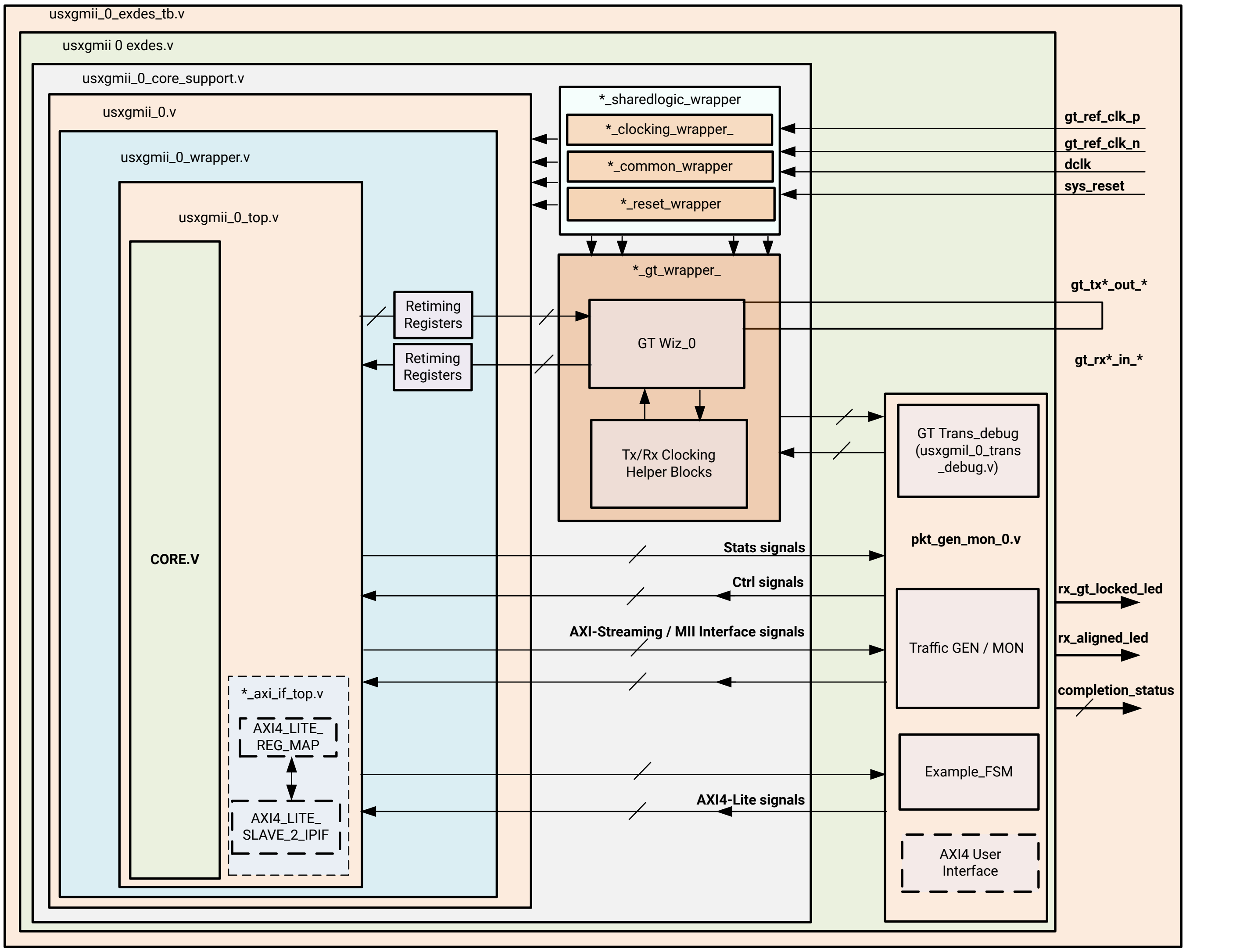
Figure 19: **Multiple Core Example Design Hierarchy**



Example Design Hierarchy (GT in Example Design)

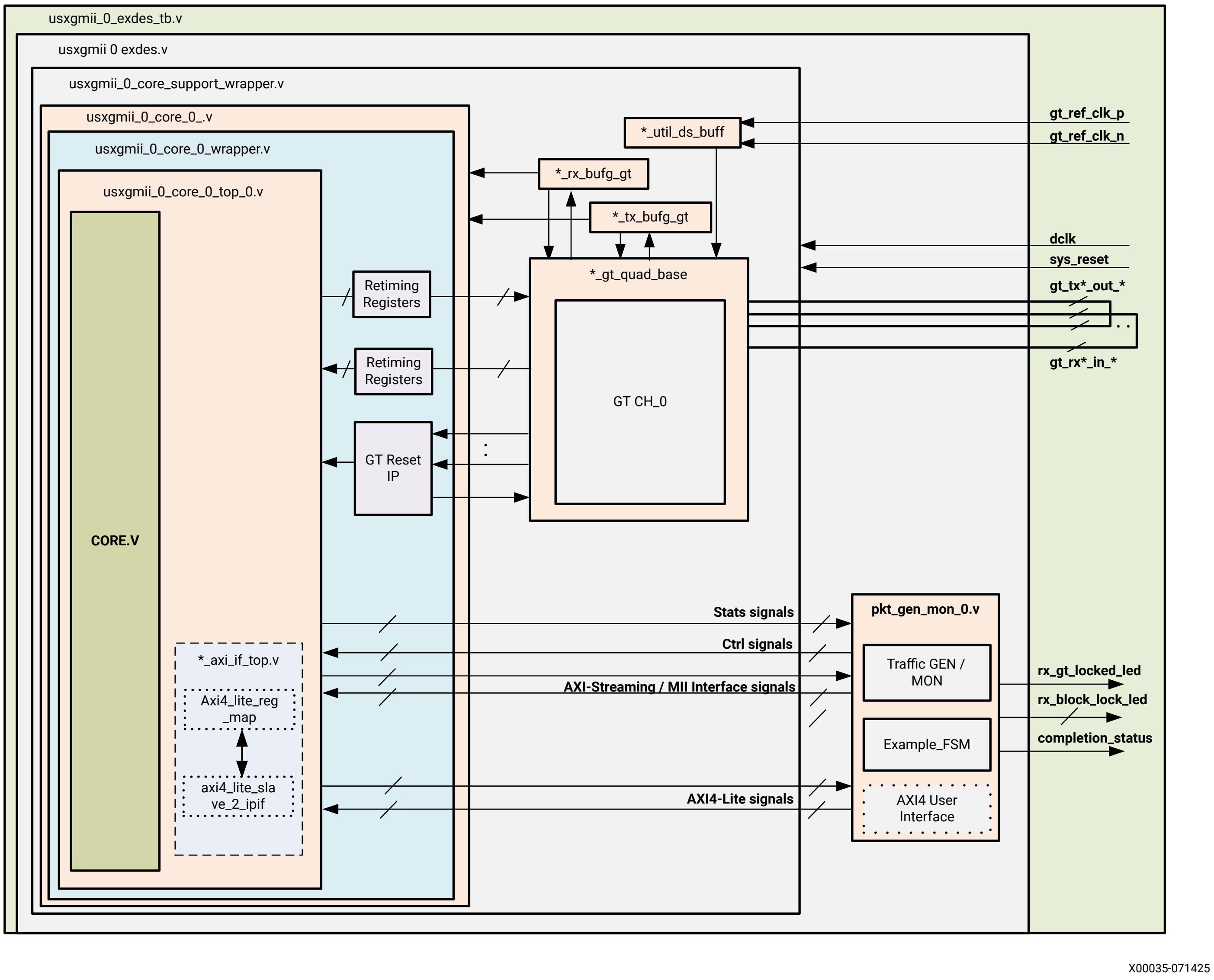
When the USXGMII Ethernet subsystem is added to AMD Vivado™ IP integrator, the Run Block Automation IP/Core and GT (Serial transceivers) will get connected with some helper blocks as per the core configuration. There is a reset interface IP, internal to USXGMII Ethernet IP, used to release TX/RX `mstreset` to the Versal device GT and check for TX/RX `mstresetdone` status and reset sequencing to GT.

Figure 20: **Single Core with GT in Example Design Hierarchy**



X00039-071425

Figure 21: **Single Core with GT in Example Design Hierarchy (Versal Adaptive SoC)**



The previous figures show the instantiation of various modules and their hierarchy for a single core configuration of the `usxgmii_0` example design when the GT (serial transceiver) is outside the core, that is, it is in the example design. This hierarchical example design is delivered when you select the Include GT subcore in example design option from the GT Selection and Configuration tab.

The usxgmii_0_core_support.v is present in the hierarchy when you select the Include GT subcore in example design option from the GT Selection and Configuration tab or the Include Shared Logic in example design option from the Shared Logic tab. This instantiates the `usxgmii_0_sharedlogic_wrapper.v` module and the `usxgmii_0.v` module for the Include Shared Logic in example design option. The usxgmii_0_gt_wrapper.v module will be present when you select the GT subcore in example design option.

The user interface available is the same as mentioned in Chapter 2: Core Overview.

The `usxgmii_0.v` module instantiates the necessary the sync registers/retiming pipeline registers for the synchronization of data between the core and the GT.

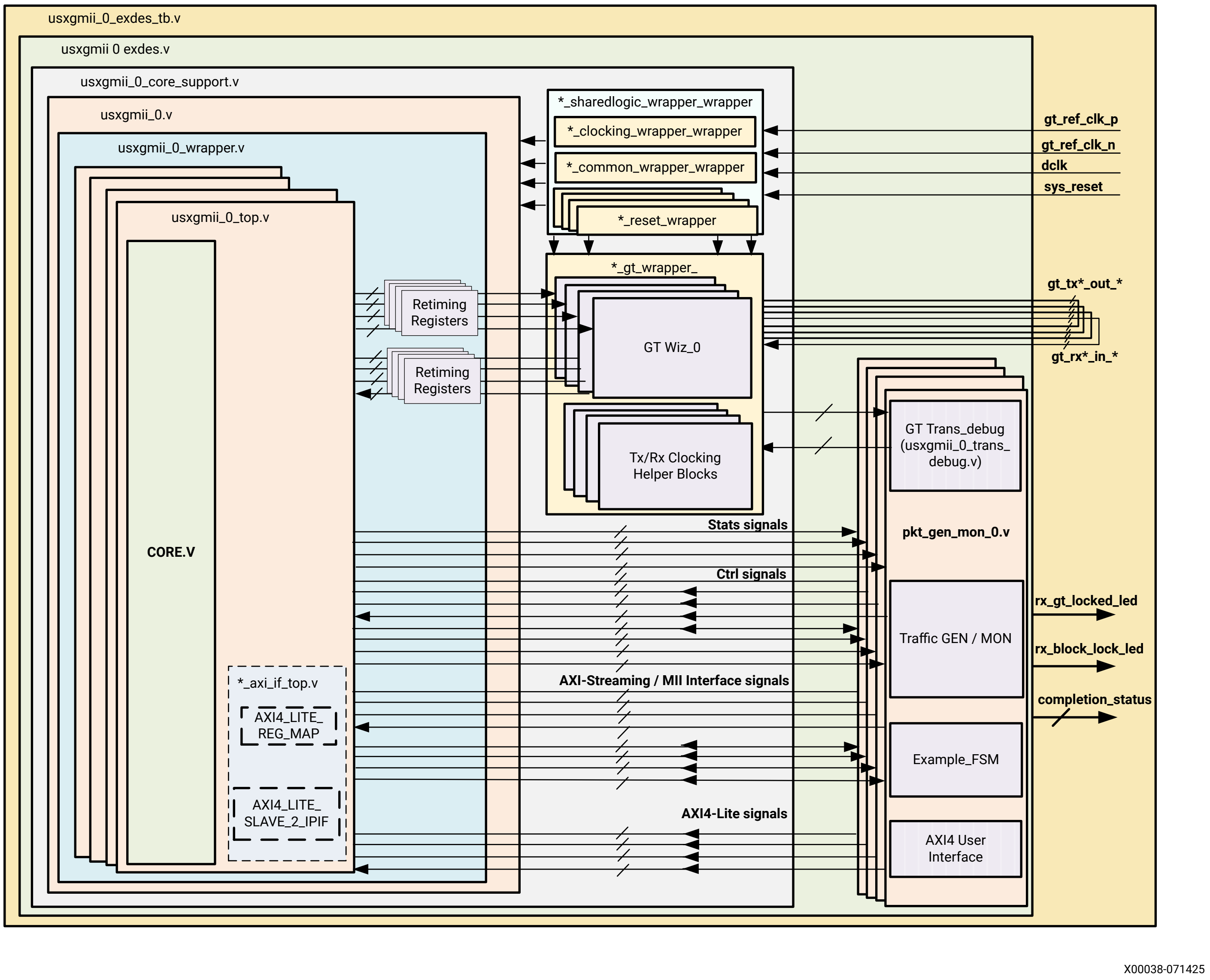
The `usxgmii_0_pkt_gen_mon` module is used to generate the data packets for sanity testing. The packet generation and checking is controlled by a Finite State Machine (FSM) module.

Description of optional modules are as follows:

- `usxgmii _0_sharedlogic_wrapper`: this module is present in the example design when you select the Include GT subcore in example design option from the GT Selection and Configuration tab or Include Shared Logic in the Example Design from the Shared Logic tab. This module brings all modules that can be shared between multiple IP cores and designs outside the core.
- `usxgmii_0_gt_wrapper`: this module is present in the example design when you select the Include GT subcore in example design option from the GT Selection and Configuration tab. This module has instantiations of the GT along with various helper blocks. The clocking helper blocks are used to generate the required clock frequency for the core.

The following figure shows the instantiation of various modules and their hierarchy for the multiple core configuration of the `usxgmii_0` example design when the GT is in the example design.

Figure 22: **Multiple Core with GT in Example Design Hierarchy (UltraScale/UltraScale+)**

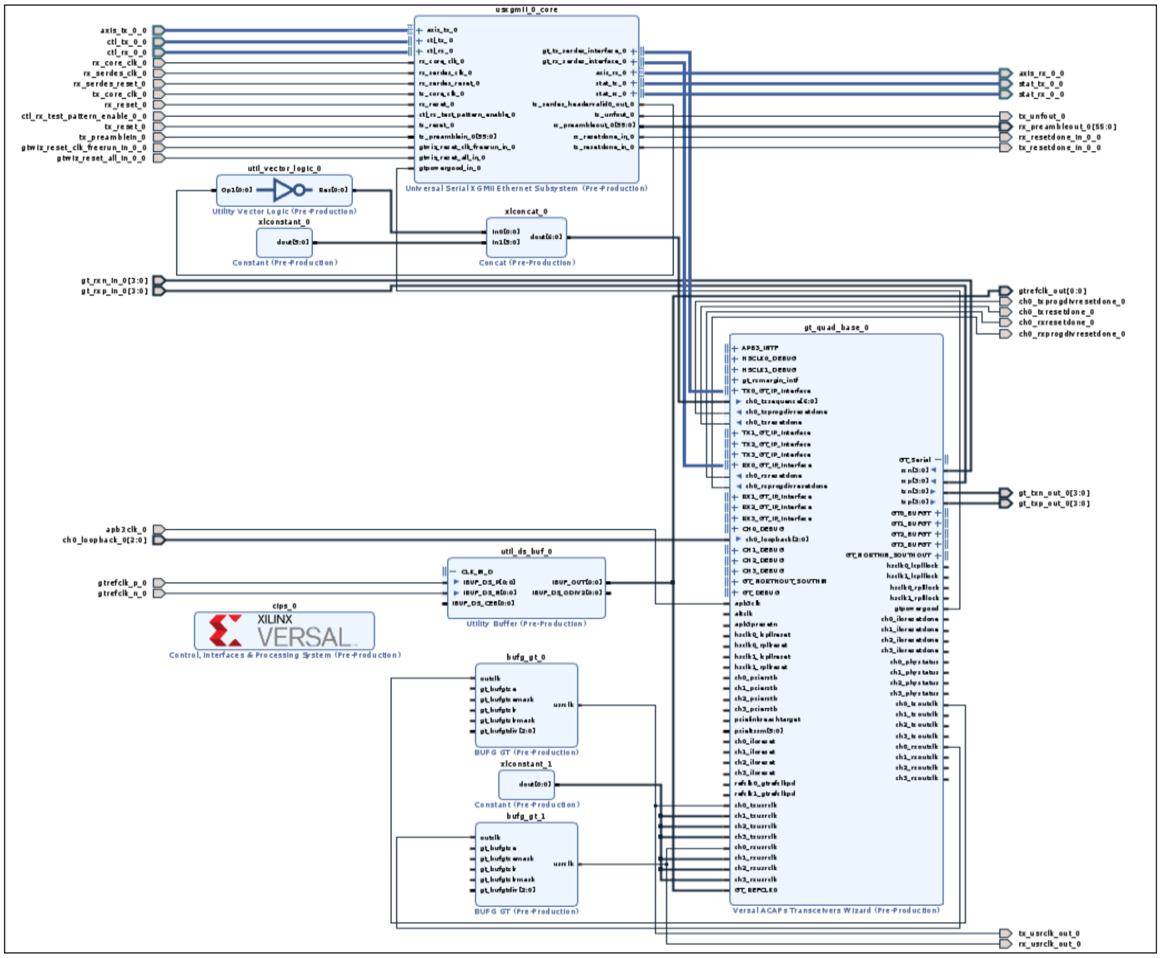


For Versal platforms, the gt_quad_base (GT Wizard for Versal device) are a part of the example design only, and USXGMII Ethernet Subsystem IP and GT (Serial transceiver) IP are connected in the block design using the IP integrator.

The following figure is a block design, where USXGMII Ethernet example design is connected in the IP integrator. See the *Vivado Design Suite Tutorial: Designing IP Subsystems Using IP Integrator* (UG995) for more information on IP integrator.

***Note*:** When the USXGMII Ethernet subsystem is added to Vivado IP integrator, the run Block Automation IP/Core and GT (Serial transceivers) is connected with some helper blocks as per the core configuration. There is a reset interface IP, internal to USXGMII Ethernet IP, used to release TX/RX `mstreset` to Versal device GT and check for TX/RX `mstresetdone` status and reset sequencing to GT.

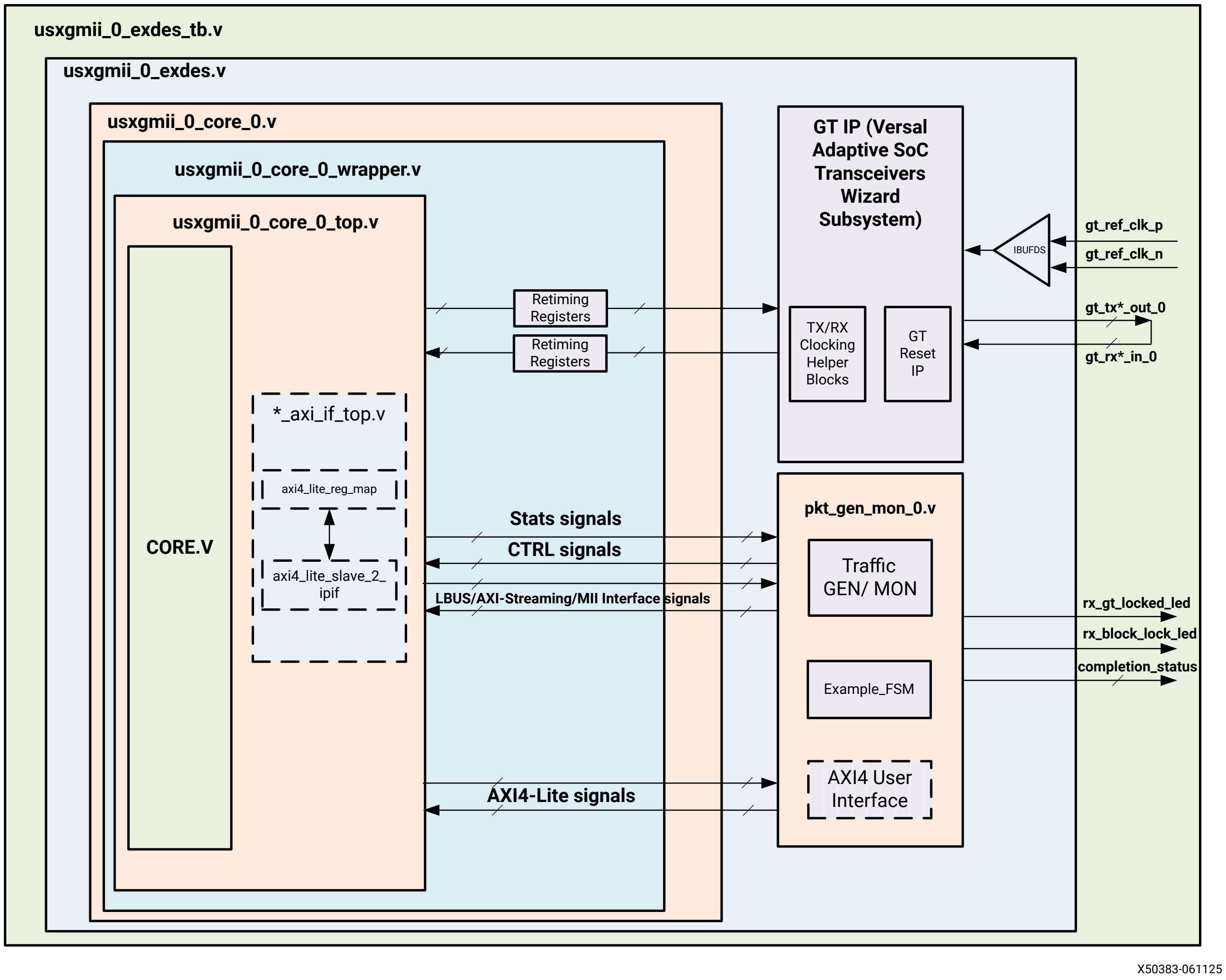
Figure 23: **USXGMII Ethernet Block Design**



Example Design with GT Wizard Subsystem (gtwiz_versal IP)

The following figure shows the USXGMII example design with GT Wizard Subsystem (gtwiz_versal IP). In the Exdes top, the USXGMII and gtwiz_versal IPs are integrated through RTL instantiation. TX/RX clocking helper blocks and GT Reset Controller IP are inside the gtwiz_versal IP.

Figure 24: **Single Core with GT Wizard Subsystem in Example Design Hierarchy**



For more information on the gtwiz_versal IP, see the *Versal Adaptive SoC Transceiver Subsystem Product Guide* (PG442).

The simulation and implementation flow of USXGMII IP example design with GT Wizard Subsystem (gtwiz_versal IP) is same as USXGMII IP example design with legacy GT Wizard (quad_base IP).

User Interface

General purpose I/Os (GPIOs) are provided to control the example design. The user input and user output ports are described in the following table.

Table 213: **User Input and User Output Ports**

Name	Direction	Description
sys_reset	Input	Reset for USXGMII core
gt_ref_clk_p	Input	Differential input clk to GT.
gt_ref_clk_n	Input	Differential input clk to GT.
dclk	Input	Stable/free running input clk to GT.
rx_gt_locked_led_0	Output	Indicates that GT has been locked.
rx_block_lock_led_0	Output	Indicates RX block lock has been achieved.
restart_tx_rx_0	Input	This signal is used to restart the packet generation and reception or the data sanity test when the packet generator and the packet monitor are in idle state.
completion_status	Output	This signal represents the test status/ result. 5’d0 Test did not run. 5’d1 - PASSED 25GE/10GE CORE TEST SUCCESSFULLY COMPLETED 5'd2 - No block lock on any lanes. 5'd3 - Not all lanes achieved block lock. 5'd4 - Some lanes lost block lock after achieving block lock. 5'd5 - No lane sync on any lanes. 5'd6 - Not all lanes achieved sync. 5'd7 - Some lanes lost sync after achieving sync. 5'd8 - No alignment status or rx_status was achieved. 5'd9 - Loss of alignment status or rx_status after both were achieved. 5'd10 - TX timed out. 5'd11 - No TX data was sent. 5'd12 - Number of packets received did not equal the number of packets sent. 5'd13 - Total number of bytes received did not equal the total number of bytes sent. 5'd14 - A protocol error was detected. 5'd15 - Bit errors were detected in the received packets. 5'd31 - Test is stuck in reset.

Shared Logic Implementation

Shared logic includes all the shareable modules that can be present as part of the core or in the Example Design.

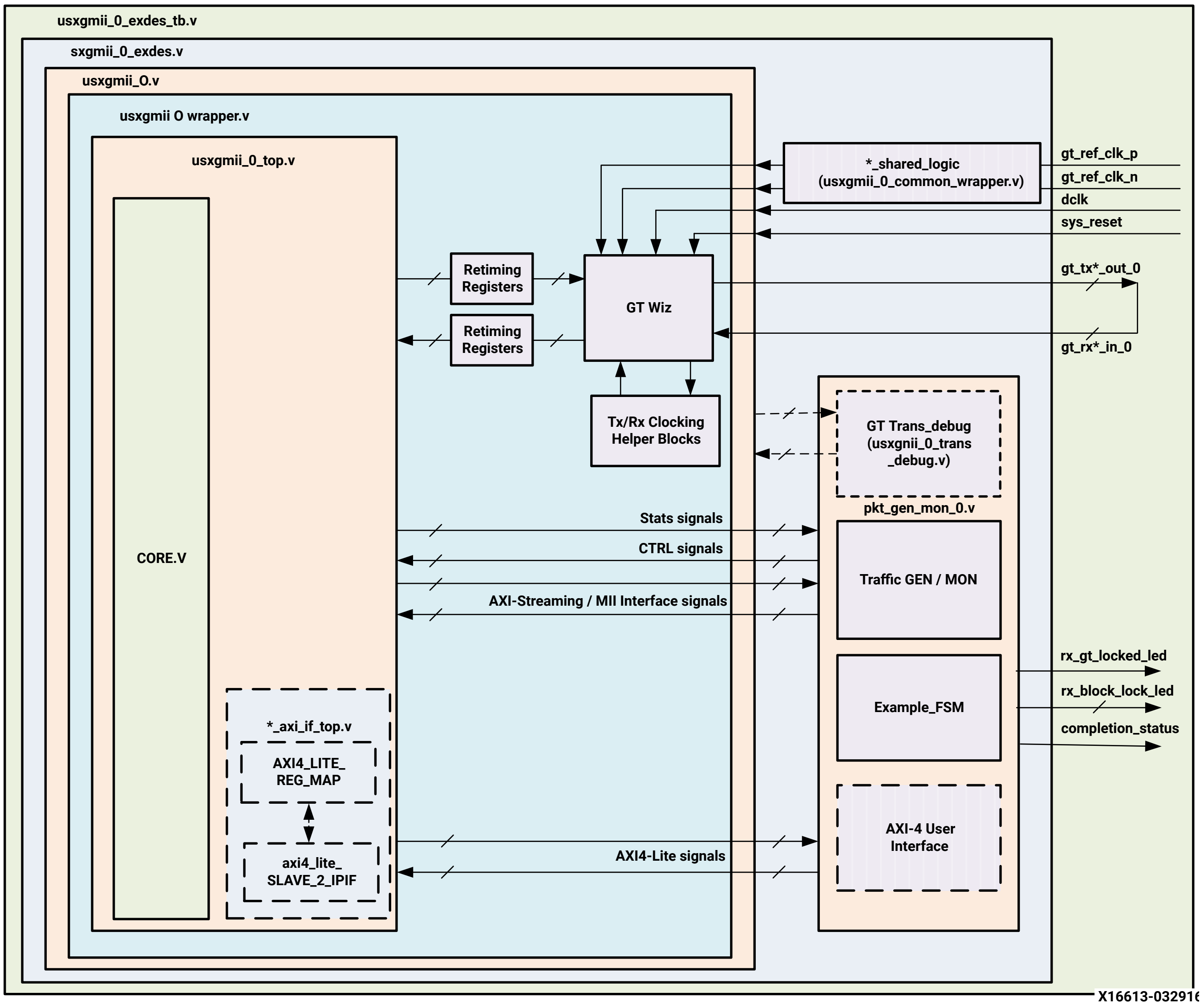
By default GT common, reset logic and clocking modules are present inside the IP core. In case of the following conditions, these modules will be placed outside the core so that they can be shared with other designs.

- When you select the Include GT subcore in example design option in the GT Selection and Configuration tab.
- When you select the Include Shared Logic in Example Design option in the Shared Logic tab.

When the shared logic in the example design is selected, a new `usxgmii_*_core_support.v` module will be instantiated between the `usxgmii_*_exdes.v` and DUT (that is, `usxgmii_*.v`). This module will have all the sub modules that can be shared between multiple designs.

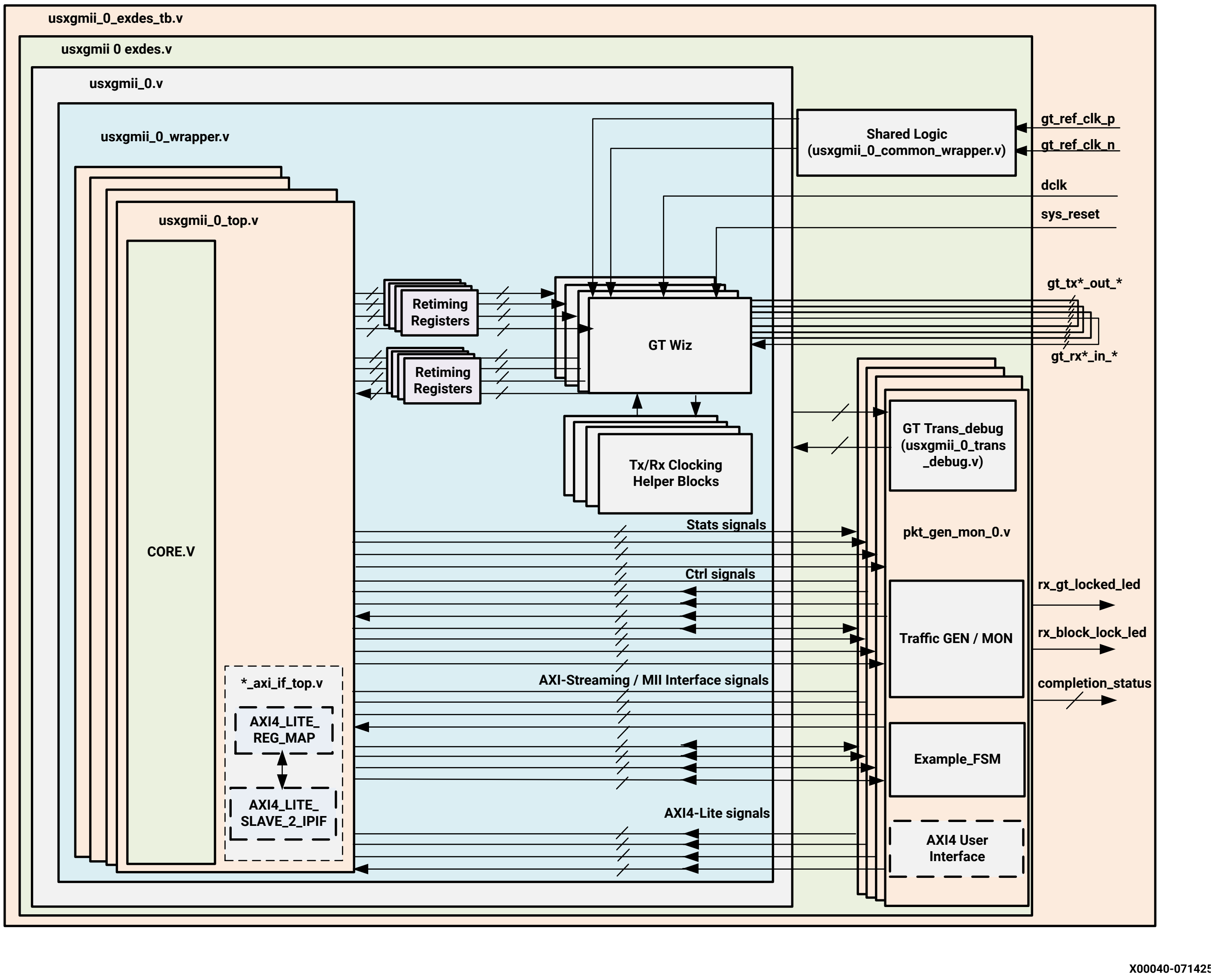
The following figure shows the implementation when shared logic is instantiated in the example design for single core.

Figure 25: **Single Core Example Design Hierarchy with Shared Logic Implementation**



The following image shows the implementation when shared logic is instantiated in the example design for multiple cores.

Figure 26: **Multiple Core Example Design Hierarchy with Shared Logic Implementation**



These modules are part of the shared logic wrapper:

- *_clocking_wrapper

  This module contains all the clocking resources that can be shared with other core instances.

- *_common_wrapper

  This module contains the GT common module that can be shared with other core instances.

- *_reset_wrapper

  This module contains the reset logic for the specified configuration.

AXI4-Lite Interface Implementation

In order to instantiate the AXI4-Lite interface to access the control and status registers of the USXGMII IP Core, enable the Include AXI4-Lite checkbox in the Configuration Tab of the Vivado IDE. This option enables the `usxgmii_0_axi_if_top` module (which contains `usxgmii_0_pif_registers` with the `usxgmii_0_slave_2_ipif` module). You can access the AXI4-Lite interface logic registers (control, status, and statistics) from the `usxgmii_0_pkt_gen_mon` module.

This mode enables the following features:

- You can configure all the control (CTL) ports of the core through the AXI4-Lite interface. This operation is performed by writing to a set of address locations with the required data to the register map interface.
- You can access all the status and statistics registers from the core through the AXI4-Lite interface. This operation is performed by reading the address locations for the status and statistics registers through register map.

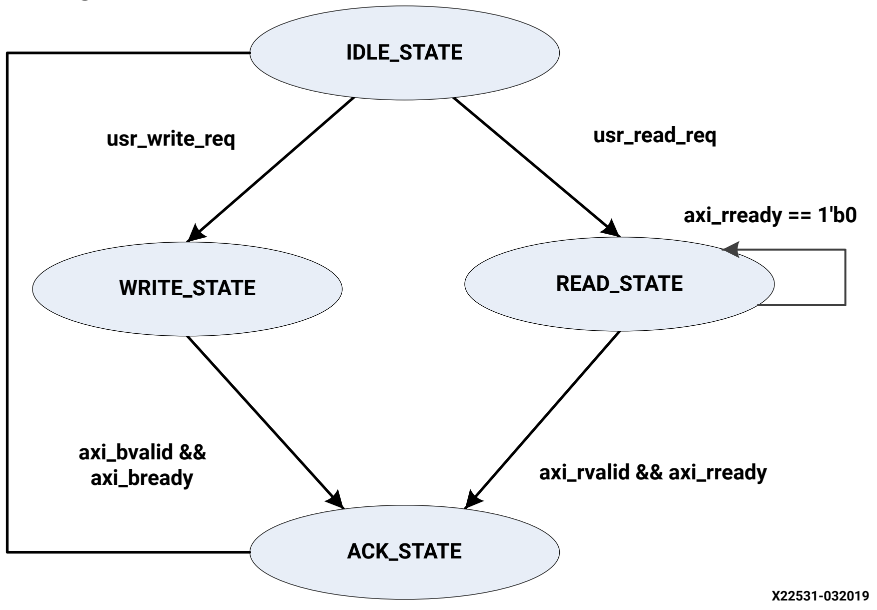
AXI4 Interface User Logic

The following sections provide the AXI4-Lite interface state machine control and ports.

User State Machine

The read and write through the AXI4-Lite slave module interface is controlled by a state machine as shown in the following figure.

Figure 27: **User State Machine for AXI4-Lite Interface**



Following is a functional description of each state.

- IDLE_STATE: By default, the FSM is in IDLE_STATE. When the `user_read_req` signal becomes High, then it moves to the READ_STATE else if the `user_write_req` signal is High, it moves to WRITE_STATE else it remains in IDLE_STATE.
- WRITE_STATE: You provide S_AXI_AWVALID, S_AXI_AWADDR , S_AXI_WVALID, S_AXI_WDATA, and S_AXI_WSTRB in this state to write to the register map through AXI. When S_AXI_BVALID and S_AXI_BREADY from the AXI slave are High, it moves to ACK_STATE. If any write operation happens in any illegal addresses, the S_AXI_BRESP [1:0] signal indicates 2'b10 that asserts the write error signal.
- READ_STATE: You provide S_AXI_ARVALID and S_AXI_ARADDR in this state to read from the register map through AXI. When S_AXI_RVALID and S_AXI_RREADY are High then it moves to ACK_STATE. If any read operation happens from any illegal addresses, the S_AXI_RRESP [1:0] signal indicates 2'b10 that asserts the read error signal.
- ACK_STATE: The state moves to IDLE_STATE.

Table 214: **AXI User Interface Ports**

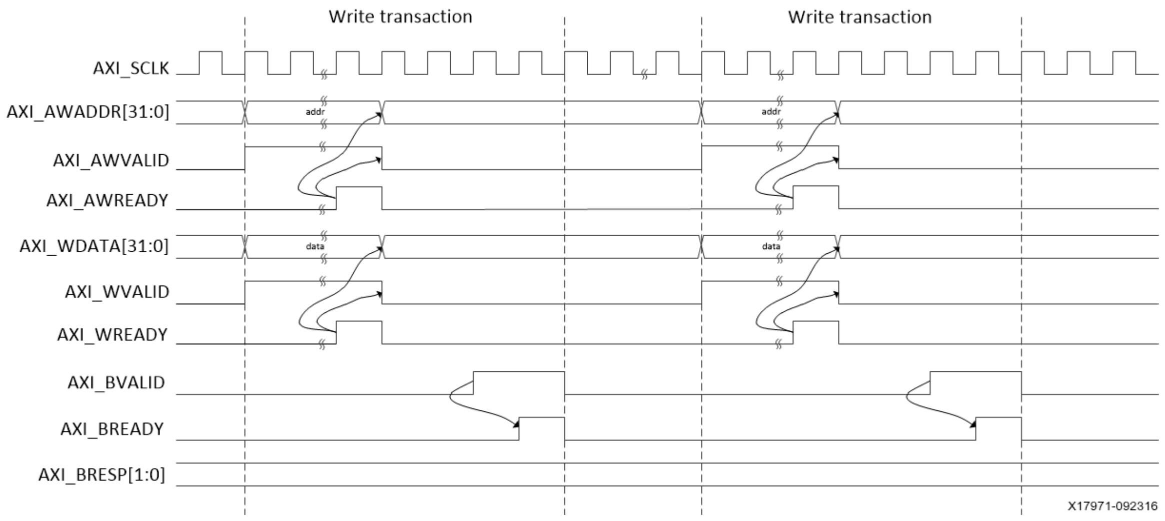
Name	Size	Direction	Description
S_AXI_ACLK	1	Input	AXI clock signal
S_AXI_ARESETN	1	Input	AXI active-Low synchronous reset
S_AXI_PM_TICK	1	Input	PM tick user input
S_AXI_AWADDR	32	Input	AXI write address
S_AXI_AWVALID	1	Input	AXI write address valid
S_AXI_AWREADY	1	Output	AXI write address ready

Table 214: **AXI User Interface Ports** *(cont'd)*

Name	Size	Direction	Description
S_AXI_WDATA	32	Input	AXI write data
S_AXI_WSTRB	4	Input	AXI write strobe. This signal indicates which byte lanes hold valid data.
S_AXI_WVALID	1	Input	AXI write data valid. This signal indicates that valid write data and strobes are available.
S_AXI_WREADY	1	Output	AXI write data ready
S_AXI_BRESP	2	Output	AXI write response. This signal indicates the status of the write transaction. ‘b00 = OKAY ‘b01 = EXOKAY ‘b10 = SLVERR ‘b11 = DECERR
S_AXI_BVALID	1	Output	AXI write response valid. This signal indicates that the channel is signaling a valid write response.
S_AXI_BREADY	1	Input	AXI write response ready.
S_AXI_ARADDR	32	Input	AXI read address
S_AXI_ARVALID	1	Input	AXI read address valid
S_AXI_ARREADY	1	Output	AXI read address ready
S_AXI_RDATA	32	Output	AXI read data issued by slave
S_AXI_RRESP	2	Output	AXI read response. This signal indicates the status of the read transfer. ‘b00 = OKAY ‘b01 = EXOKAY ‘b10 = SLVERR ‘b11 = DECERR
S_AXI_RVALID	1	Output	AXI read data valid
S_AXI_RREADY	1	Input	AXI read ready. This signal indicates the user/master can accept the read data and response information.

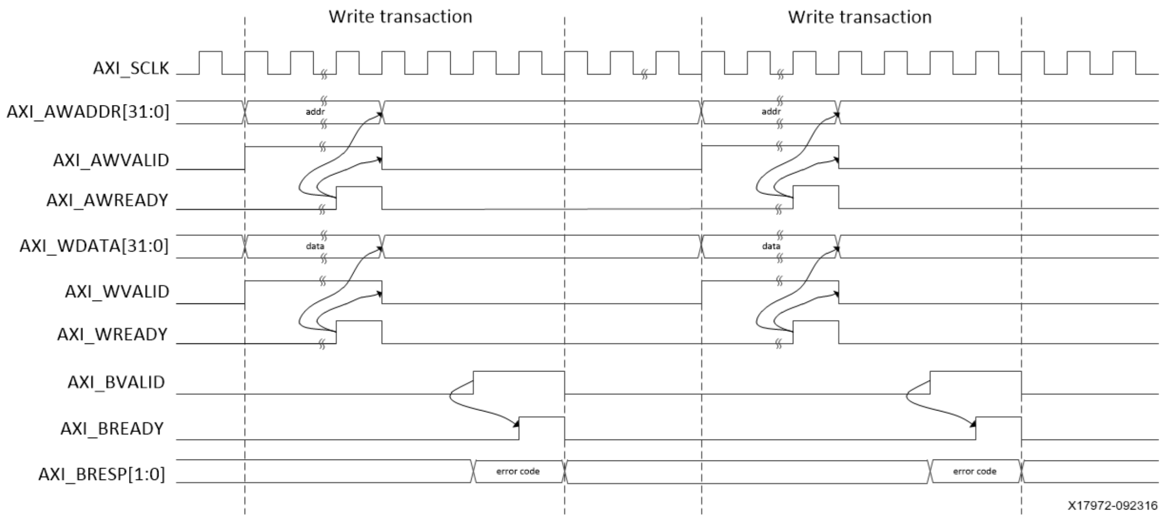
Valid Write Transactions

Figure 28: **AXI4-Lite User Side Write Transaction**



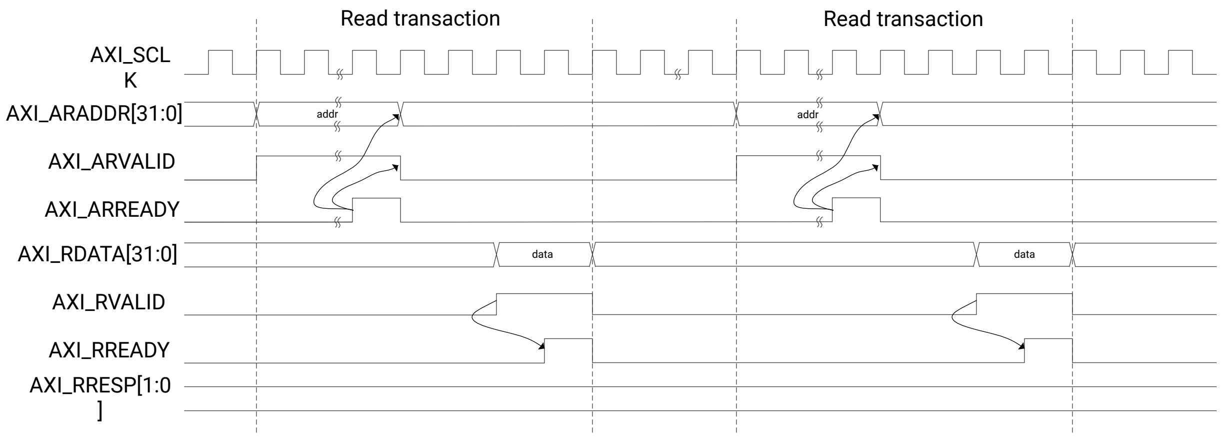
Invalid Write Transactions

Figure 29: **AXI4-Lite User Side Write Transaction with Invalid Write Address**



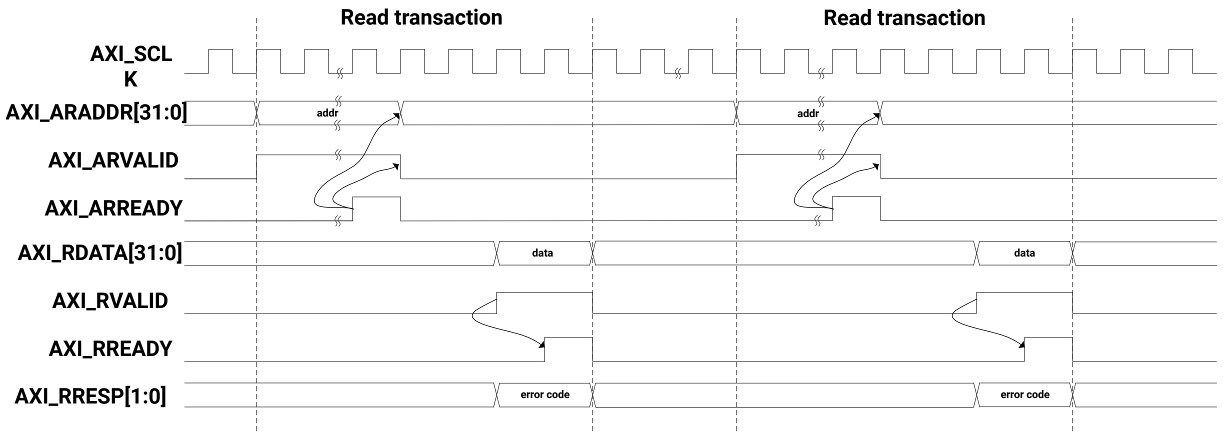
Valid Read Transactions

Figure 30: **AXI4-Lite User Side Read Transaction**



Invalid Read Transactions

Figure 31: **AXI4-Lite User Side Read Transaction with Invalid Read Address**



Chapter 7

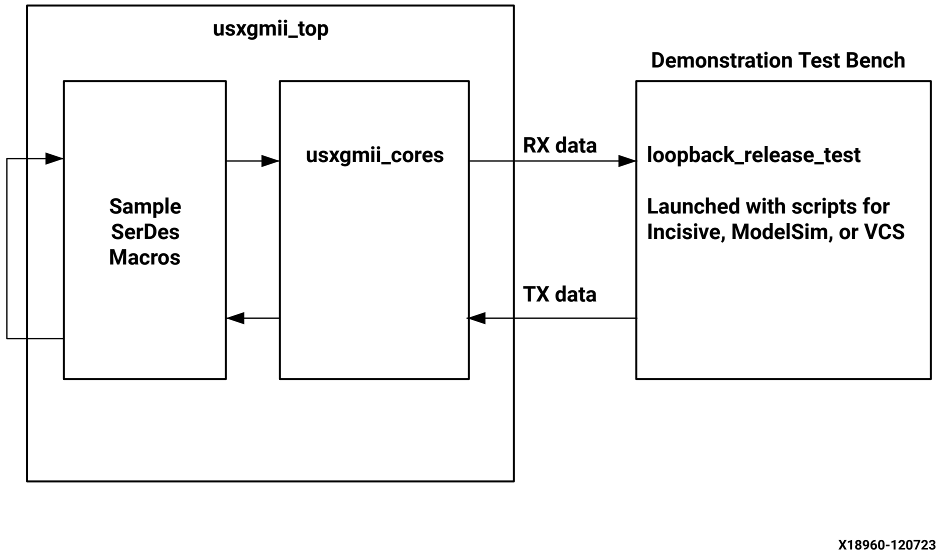
Test Bench

The USXGMII IP Core delivery contains a simple example test bench tested against common third-party simulation tools.

Each release includes a demonstration test bench that performs a loopback test on the complete core. For convenience, scripts are provided to launch the test bench from several popular simulators. The test program exercises the datapath to check that the transmitted frames are received correctly. RTL simulation models for the core are included. You must provide the correct path for the transceiver simulation model according to the latest simulation environment settings in your version of the AMD Vivado™ Design Suite.

Test Bench

Figure 32: **Test Bench**



Verification, Compliance, and Interoperability

The USXGMII IP Core has been designed to the requirements of the EDCS 1467841-NBASE-T USXGMII: Copper PHY ERS specification, revision 1.4.

Simulation

This USXGMII IP Core has been thoroughly tested using a constrained random test environment that verifies the correct operation of each feature of the core.

Synthesis and Implementation

For details about synthesis and implementation, see *Vivado Design Suite User Guide: Designing with IP* (UG896).

Debugging

This appendix includes details about resources available on the AMD 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)

IMPORTANT! *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 AMD Adaptive 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.

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 USXGMII IP Core 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).

Appendix B

Additional Resources and Legal Notices

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.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, do the following:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **AMDDesignTools → 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, do the following:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Support Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Support.

References

These documents provide supplemental material useful with this guide:

1. IEEE Standard for Ethernet
2. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
3. *Vivado Design Suite User Guide: Designing with IP* (UG896)
4. USXGMII Specification
5. *Vivado Design Suite User Guide: Getting Started* (UG910)
6. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
7. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
8. *UltraScale Architecture GTY Transceivers User Guide* (UG578)
9. *UltraScale Architecture GTH Transceivers User Guide* (UG576)
10. *Versal Adaptive SoC GTY and GTYP Transceivers Architecture Manual* (AM002)
11. *Vivado Design Suite Tutorial: Designing IP Subsystems Using IP Integrator* (UG995)
12. *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973)
13. *Versal Adaptive SoC Transceiver Subsystem Product Guide* (PG442)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/04/2025 Version 2.0	
IP Facts	Removed Versal auto-negotiation note.
Features	Removed Versal auto-negotiation support note.
Performance and Resource Utilization	Clarified device support for auto-negotiation.
Configuration Tab	Updated table and figures.
User Parameters	Updated auto-negotiation parameter.

Section	Revision Summary
07/18/2025 Version 2.0	
Transceiver Interface	Updated table
Configuration Tab	Updated figure and table
MAC Options Tab	Updated figure
GT Selection and Configuration Tab	Updated figure
Shared Logic Tab	Updated figure
Example Design with GT Wizard Subsystem (gtwiz_versal IP)	Added section
05/30/2024 Version 1.2	
Licensing and Ordering	Modified the section
12/12/2023 Version 1.2	
IP Facts	Added support for AMD Artix™ UltraScale+™
05/17/2022 Version 1.2	
Statistics Interface Ports	Updated port descriptions
12/08/2022 Version 12	
Riviera-PRO Simulator	Added section
06/29/2022 Version 1.2	
Multiple sections	Updated for AMD Vivado 2022.1 Release
08/05/2021 Version 1.2	
Multiple sections	Updated for AMD Vivado 2021.1 Release
02/10/2021 Version 1.2	
Multiple sections	Updated for AMD Versal Device
06/26/2020 Version 1.1	
Statistics Interface Ports	Updated Port Descriptions
04/24/2019 Version 1.1	
Chapter 3: Product Specification	Updated figures and tables
Chapter 5: Design Flow Steps	Updated figures and tables
10/04/2017 Version 1.0	
Initial public release	

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 2017-2025 Advanced Micro Devices, Inc. AMD, the AMD Arrow logo, Artix, Kintex, UltraScale, UltraScale+, Versal, Virtex, Vitis, Vivado, Zynq, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.