10G/25G High Speed Ethernet Subsystem v1.2

Product Guide

Vivado Design Suite

PG210 April 6, 2016

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The Xilinx® 10G/25G High Speed Ethernet subsystem implements the 25G Ethernet Media Access Controller (MAC) with a Physical Coding Sublayer (PCS) as specified by the 25G Ethernet Consortium. MAC and physical coding sublayer/physical medium attachment (PCS/PMA) or PCS/PMA alone are available. Legacy operation at 10 Gb/s is supported.

Features

- Designed to the Ethernet requirements for 10/25 Gb/s operation specified by IEEE 802.3 Clause 49, IEEE 802.3by, and the 25G Ethernet Consortium
- Includes complete Ethernet MAC and PCS/PMA functions or standalone PCS/PMA
- Simple packet-oriented user interface
- Comprehensive statistics gathering
- Status signals for all major functional indicators
- Delivered with a top-level wrapper including functional transceiver wrapper, IP netlist, sample test scripts, and Vivado® Design Suite tools compile scripts
- BASE-R PCS sublayer operating at 10 Gb/s or 25 Gb/s
  - Optional clause 74 BASE-KR FEC sublayer
  - Optional Auto-Negotiation
  - Optional clause 108 25G Reed-Solomon Forward Error Correction (RS-FEC) sublayer
- Custom Preamble mode
- Optional IEEE 1588 1-step and 2-step timestamping

Facts Table	
Core Specifics	
Supported Device Family(1)	Zynq UltraScale+ MPSoC Virtex® UltraScale+™, Kintex® UltraScale+™ Virtex UltraScale™, Kintex UltraScale™
Supported User Interfaces	AXI, XGMII, or XXVMII
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog
Test Bench	Verilog
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Verilog
Supported S/W Driver	N/A
Tested Design Flows(2)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Synopsis or Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Note: Xilinx recommends that you join the 25 Gigabit Ethernet Consortium to gain access to the 25G specification. For more information on membership, visit the 25G Ethernet Consortium.

Overview

This document details the features of the 10G/25G Ethernet Subsystem as defined by the 25G Ethernet Consortium [Ref 1]. PCS functionality is defined by *IEEE Standard 802.3, 2012, Section 4, Clause 49, Physical Coding Sublayer (PCS) for 64B/66B, type 10GBASE-R* [Ref 2]. For 25G operation, clock frequencies are increased to provide a serial interface operating at 25.78125 Gb/s to leverage the latest high-speed serial transceivers. The low latency design is optimized for UltraScale™ architecture devices.

Feature Summary

25G Supported Features

- Complete Ethernet MAC and PCS functions
- Designed to Schedule 3 of the 25G Consortium
- Statistics and diagnostics
- 66-bit SerDes interface using Xilinx GTY transceiver operating with Asynchronous Gearbox enabled
- Pause Processing including *IEEE std. 802.3* Annex 31D (Priority based Flow Control)
- Low latency
- Custom preamble and adjustable Inter Frame Gap
- Configurable for operation at 10 Gb/s (Clause 49)

10G Supported Features

- Complete MAC and PCS functions
- IEEE 802.3 Clause 49
- Statistics and diagnostics
- 66-bit SerDes interface
- Custom preamble and adjustable Inter Frame Gap

Optional Features

- Clause 73 Auto-Negotiation
- Clause 72.6.10 Link Training
- Clause 74 FEC - shortened cyclic code (2112, 2080)
- Clause 108 RS-FEC (Reed-Solomon FEC)
- PCS only version with XGMII/XXVMII interface (See the Port Descriptions – PCS Variant.)
- AXI4-Stream interface
- AXI4-Lite control and status interface

Applications

IEEE Std 802.3 enables several different Ethernet speeds for Local Area Network (LAN) applications, and 25 Gb/s is the latest addition to the standard. The capability to interconnect devices at 25 Gb/s Ethernet rates becomes especially relevant for next-generation data center networks where:

(i) To keep up with increasing CPU and storage bandwidth, rack or blade servers need to support aggregate throughputs faster than 10 Gb/s (single lane) or 20 Gb/s (dual lane) from their Network Interface Card (NIC) or LAN-on-Motherboard (LOM) networking ports;

(ii) Given the increased bandwidth to endpoints, uplinks from Top-of-Rack (TOR) or Blade switches need to transition from 40 Gb/s (four lanes) to 100 Gb/s (four lanes) while ideally maintaining the same per-lane breakout capability;

(iii) Due to the expected adoption of 100GBASE-CR4/KR4/SR4/LR4, SerDes and cabling technologies are already being developed and deployed to support 25 Gb/s per physical lane, twisted pair, or fiber.

Licensing and Ordering Information

License Checkers

If the IP requires a license key, the key must be verified. The 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 error. License checkpoints are enforced by the following tools:

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

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

License Type

10G/25G Ethernet PCS/PMA (10G/25G BASE-R)

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

For more information, visit the 10G/25G Ethernet Subsystem page.

Standalone 10G/25G Ethernet MAC and PCS/PMA (10G/25G EMAC + 10G/25G BASE-R/KR) or 10G/25G BASE-KR

Note: The 10G/25G Ethernet MAC + BASE-R and 10GBASE-KR/25GBASE-KR IP features require separate fee-based licensing.

These Xilinx IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase one or more licenses for the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the 10G/25G Ethernet Subsystem page.

Product Specification

Figure 2-1 shows the block diagram of the 10G/25G Ethernet subsystem, not including the GTY transceiver.

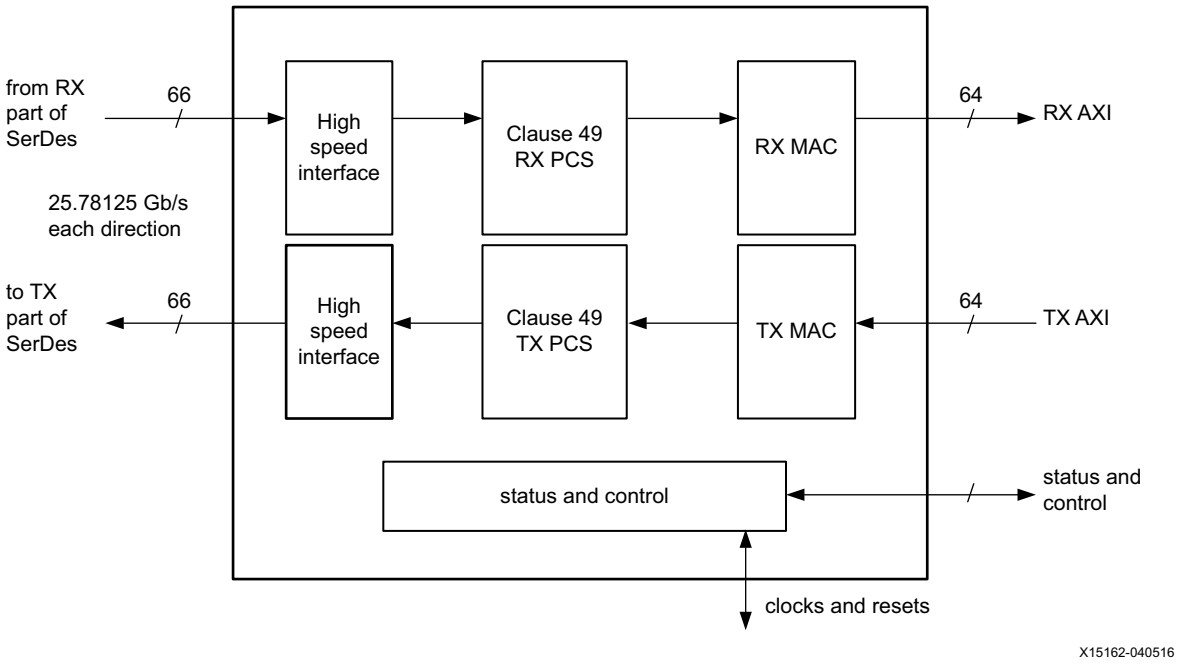

Figure 2-1: **Core Block Diagram**

A PCS-only variant of the core is also available. The block diagram is shown in Figure 2-2.

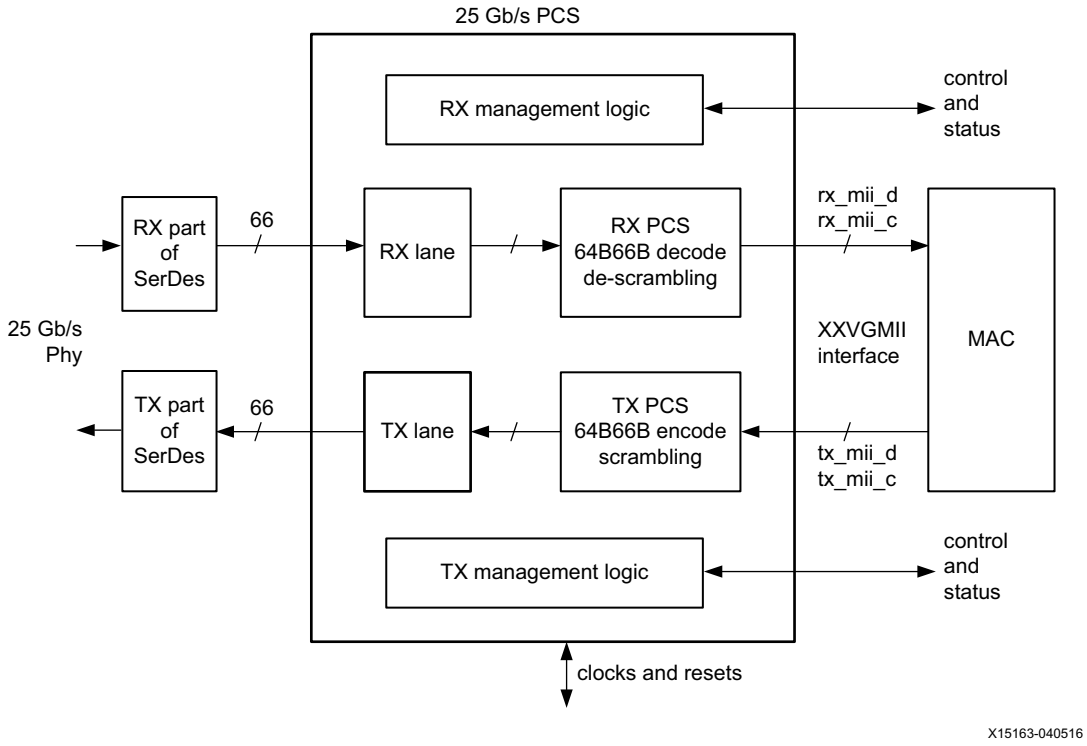

Figure 2-2: **Block Diagram of PCS-Only Core Variant**

Standards

The 10G/25G Ethernet core is designed to the standard specified in the *25G and 50G Ethernet Consortium* [Ref 1] and the *IEEE Std 802.3* including *IEEE 802.3by* [Ref 2]

Performance and Resource Utilization

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

Port Descriptions

The following tables lists the ports for the 10G/25G Ethernet subsystem with integrated MAC and PCS. These signals are usually found at 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

Table 2-1 shows the transceiver I/O ports for the 10G/25G Ethernet subsystem. Refer to Clocking in Chapter 3 for details regarding each clock domain.

Table 2-1: **Transceiver I/O**

Name	Direction	Description	Clock Domain
gt_tx_reset	Input	Reset for the gigabit transceiver (GT) TX.	Asynch
gt_rx_reset	Input	GT RX reset.	Asynch
ctl_gt_reset_all	Input	Active-High asynchronous reset for the transceiver startup Finite State Machine (FSM). Note that this signal also initiates the reset sequence for the entire 10G/25G Ethernet subsystem.	Asynch
refclk_n0	Input	Differential reference clock input for the SerDes, negative phase.	Refer to Clocking.
refclk_p0	Input	Differential reference clock input for the SerDes, positive phase.	Refer to Clocking.
rx_serdes_data_n0	Input	Serial data from the line; negative phase of the differential signal	Refer to Clocking.
rx_serdes_data_p0	Input	Serial data from the line; positive phase of the differential signal	Refer to Clocking.
tx_serdes_data_n0	Output	Serial data to the line; negative phase of the differential signal.	Refer to Clocking.
tx_serdes_data_p0	Output	Serial data to the line; positive phase of the differential signal.	Refer to Clocking.
tx_serdes_clkout	Output	When present, same as tx_clk_out.	Refer to Clocking.

AXI4-Stream Clocks and Resets

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

Name	Direction	Description	Clock Domain
rx_clk_out	Output	Receive Local bus clock. All signals between the HSEC and the user-side logic are synchronized to the positive edge of this signal. The AXI4-Stream clock is 390.625 MHz. When the RX FIFO is included, the RX AXI4-Stream clock is an input and should be equal to or greater than tx_clk_out.	Refer to Clocking.
tx_clk_out	Output	Transmit Local bus clock. All signals between the HSEC and the user-side logic are synchronized to the positive edge of this signal. The AXI4-Stream clock is 390.625 MHz.	Refer to Clocking.
rx_reset	Input	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.	Asynch
tx_reset	Input	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.	Asynch

Transmit AXI4-Stream Interface

Table 2-3 shows the AXI4-Stream transmit interface signals.

Table 2-3: **AXI4-Stream Transmit Interface Signals**

Signal	Direction	Description
tx_axis_tdata[63:0]	In	AXI4-Stream data (64-bit interface)
tx_axis_tkeep[7:0]	In	AXI4-Stream Data Control (64-bit interface)
tx_axis_tvalid	In	AXI4-Stream Data Valid input
tx_axis_tuser	In	AXI4-Stream User Sideband interface. Equivalent to the tx_errin signal. 1 indicates a bad packet has been received. 0 indicates a good packet has been received.
tx_axis_tlast	In	AXI4-Stream signal indicating End of Ethernet Packet.
tx_axis_tready	Out	AXI4-Stream acknowledge signal to indicate to start the Data transfer.

Data Lane Mapping

For transmit data `tx_axis_tdata[63:0]`, the port is divided into lane 0 to lane 7 (see Table 2-4).

Table 2-4: **tx_axis_tdata Lanes**

Lane/ tx_axis_tkeep	tx_axis_tdata[63:0] Bits
0	7:0
1	15:8
2	23:16
3	31:24
4	39:32
5	47:40
6	55:48
7	63:56

Normal Transmission

The timing of a normal frame transfer is shown in Figure 2-3. When the client wants to transmit a frame, it asserts the `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 packet is indicated to the core by `tx_axis_tlast` asserted for 1 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.

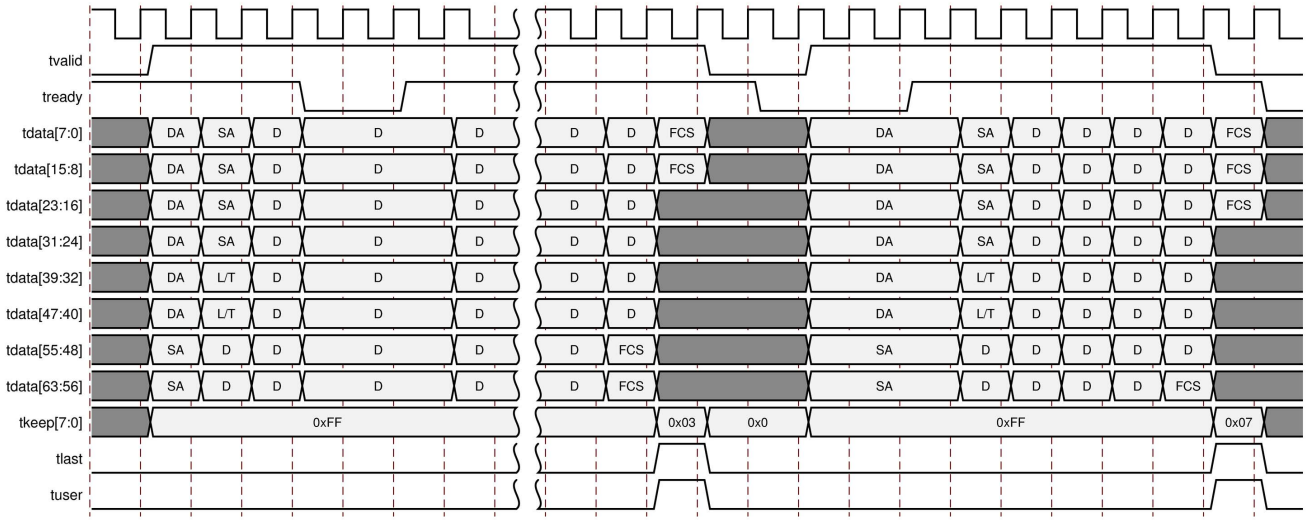

Figure 2-3: **Normal Frame Transfer**

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 deasserting `tx_axis_tuser` High while `tx_axis_tlast` is High (See Figure 2-5).
- An implicit underrun, in which a frame transfer is aborted by deasserting `tx_axis_tvalid` without asserting `tx_axis_tlast`.

When either of the two scenarios occurs during a frame transmission, the core inserts error codes into the data stream to flag the current frame as an errored frame. It remains the responsibility of the client to re-queue the aborted frame for transmission, if necessary.

Receive AXI4 Stream Interface

Table 2-5 shows the AXI4-Stream receive interface signals.

Table 2-5: **AXI4-Stream Receive Interface Signals**

Signal	Direction	Description
rx_axis_tdata[63:0]	Out	AXI4-Stream Data to upper layer
rx_axis_tkeep[7:0]	Out	AXI4-Stream Data Control to upper layer
rx_axis_tvalid	Out	AXI4-Stream Data Valid
rx_axis_tuser	Out	AXI4-Stream User Sideband interface. 1 indicates a bad packet has been received. 0 indicates a good packet has been received.
rx_axis_tlast	Out	AXI4-Stream signal indicating an end of packet.

Data Lane Mapping

For receive data `rx_axis_tdata`, the port is divided into lane 0 to lane 7 (see Table 2-6).

Table 2-6: **rx_axis_tkeep Lanes**

Lane/ rx_axis_tkeep	rx_axis_tdata Bits
0	7:0
1	15:8
2	23:16
3	31:24
4	39:32
5	47:40
6	55:48
7	63:56

Normal Frame Reception

The timing of a normal inbound frame transfer is represented Figure 2-4. 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 frame checks are completed. This is after the FCS field has been received. The core asserts the `rx_axis_tuser` signal to indicate that the frame was successfully received and that the frame should be analyzed by the client. This is also the end of packet signaled by `rx_axis_tlast` asserted for one cycle.

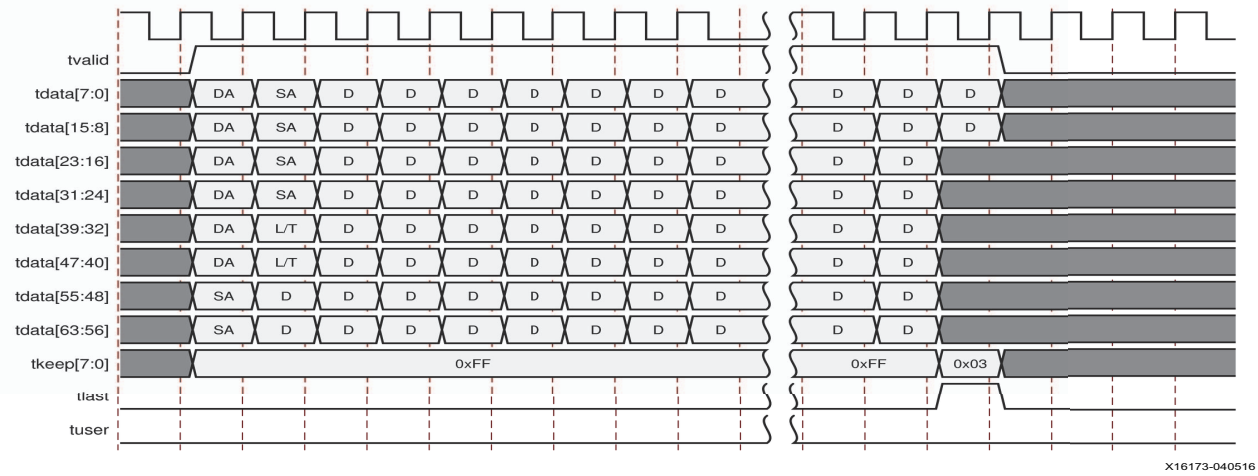

Figure 2-4: **Normal Frame Reception**

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 Figure 2-5. 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 then 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, MTU Size Enable Frames are enabled.

- Any control frame that is received is not exactly the minimum frame length unless Control Frame Length Check Disable is set.
- The XGMII data stream contains error codes.

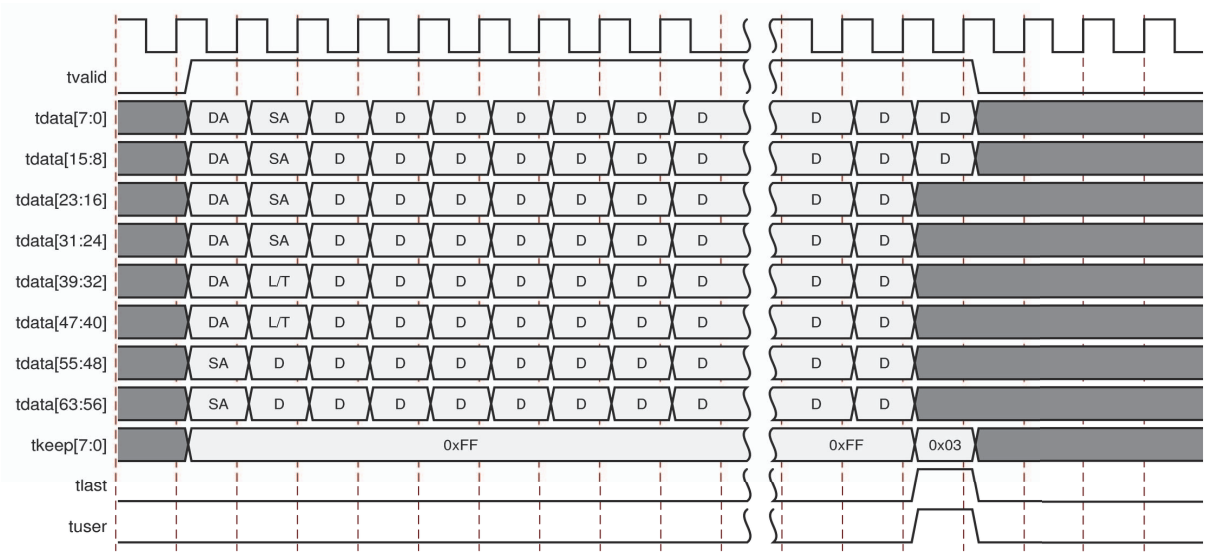

Figure 2-5: **Frame Reception with Errors**

AXI4-Stream Control and Status Ports

Table 2-7: **AXI4-Stream Interface–TX Path Control/Status Signals**

Name	Direction	Description	Clock Domain
tx_preamblein [55:0]	Input	This is the custom preamble which is a separate input port rather than being in-line with the data. It should be valid during the start of packet.	tx_clk_out
ctl_tx_ipg_value[3:0]	Input	This signal can be optionally present. The ctl_tx_ipg_value defines the target average minimum Inter Packet Gap (IPG, in bytes) inserted between AXI4-Stream packets. Valid values are 8 to 12. The ctl_tx_ipg_value can also be programmed to a value in the 0 to 7 range, but in that case, it is interpreted as meaning "minimal IPG", so only Terminate code word IPG is inserted; no Idles are ever added in that case and that produces an average IPG of around 4 bytes when random-size packets are t ransmitted.	tx_clk_out
ctl_tx_enable	Input	TX Enable. This signal is used to enable the transmission of data when it is sampled as a 1. When sampled as a 0, only idles are transmitted by the core. This input should not be set to 1 until the receiver it is sending data to (that is, the receiver in the other device) is fully synchronized and ready to receive data (that is, the other device 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.	tx_clk_out
ctl_tx_send_rfi	Input	Transmit Remote Fault Indication (RFI) code word. If this input is sampled as a 1, the TX path only transmits Remote Fault code words. This input should be set to 1 until the RX path is fully synchronized and is ready to accept data from the link partner.	tx_clk_out
ctl_tx_send_lfi	Input	Transmit Local Fault Indication (LFI) code word. Takes precedence over Remote Fault Indication (RFI).	tx_clk_out
ctl_tx_send_idle	Input	Transmit Idle code words. If this input is sampled as a 1, the TX path only transmits Idle code words. This input should be set to 1 when the partner device is sending RFI code words.	tx_clk_out
ctl_tx_fcs_ins_enable	Input	Enable FCS insertion by the TX core. If this bit is set to 0, the core does not add FCS to packet. It this bit is set to 1, the core calculates and adds the FCS to the packet. This input cannot be changed dynamically between packets.	tx_clk_out

Table 2-7: **AXI4-Stream Interface–TX Path Control/Status Signals *(Cont'd)***

Name	Direction	Description	Clock Domain
ctl_tx_ignore_fcs	Input	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 this input is Low and a packet with bad FCS is being transmitted, it is not binned as good. If this input is High, 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. ***Note:*** Statistics are reported as if there was no FCS error.	tx_clk_out
stat_tx_local_fault	Output	A value of 1 indicates the receive decoder state machine is in the TX_INIT state. This output is level sensitive.	tx_clk_out
ctl_tx_custom_preamble_enable	Input	When asserted, this signal treats the side band of a packet on the AXI4-Stream as a custom preamble instead of inserting a standard preamble.	tx_clk_out

Table 2-8: **AXI4-Stream Interface–RX Path Control/Status Signals**

Name	Direction	Description	Clock Domain
rx_preambleout [55:0]	Output	This is the preamble, and now a separate output instead of inline with data as was done with previous releases.	rx_clk_out
ctl_rx_enable	Input	RX Enable. For normal operation, this input must be set to 1. When this input is set the 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.	rx_clk_out
ctl_rx_check_preamble	Input	When asserted, this input causes the MAC to check the preamble of the received frame.	rx_clk_out
ctl_rx_check_sfd	Input	When asserted, this input causes the MAC to check the Start of Frame Delimiter of the received frame.	rx_clk_out
ctl_rx_force_resync	Input	RX force resynchronization input. This signal is used to force the RX path to reset and re-synchronize. A value of 1 forces the reset operation. A value of 0 allows normal operation. Note that this input should normally be Low and should only be pulsed (1 cycle minimum pulse).	rx_clk_out

Table 2-8: **AXI4-Stream Interface–RX Path Control/Status Signals *(Cont'd)***

Name	Direction	Description	Clock Domain
ctl_rx_delete_fcs	Input	Enable FCS removal by the RX core. If this bit is set to 0, the core does not remove the FCS of the incoming packet. If this bit is set to 1, the core deletes the FCS to the received packet. Note that FCS is not deleted for packets that are less than or equal to 8 bytes long. This input should only be changed while the corresponding reset input is asserted.	rx_clk_out
ctl_rx_ignore_fcs	Input	Enable FCS error checking at the AXI4-Stream interface by the RX core. If this bit is set to 0, a packet received with an FCS error is sent with the rx_errout pin asserted during the last transfer (rx_eopout and rx_enaout sampled 1). If this bit is set to 1, the core does not flag an FCS error at the AXI4-Stream interface. ***Note:*** The statistics are reported as if the packet is good. The signal stat_rx_bad_fcs, however, reports the error.	rx_clk_out
ctl_rx_max_packet_len[14:0]	Input	Any packet longer than this value is considered to be oversized. If a packet has a size greater than this value, the packet is truncated to this value and the rx_errout signal is asserted along with the rx_eopout signal. Packets less than 64 bytes are dropped. ctl_rx_max_packet_len[14] is reserved and must be set to 0.	rx_clk_out
ctl_rx_min_packet_len[7:0]	Input	Any packet shorter than this value is considered to be undersized. If a packet has a size less than this value, the rx_errout signal is asserted during the rx_eopout asserted cycle.	rx_clk_out
stat_rx_framing_err[2-1:0]	Output	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 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.	rx_clk_out
stat_rx_framing_err_valid	Output	Valid indicator for stat_rx_framing_err. When sampled as a 1, the value on stat_rx_framing_err is valid.	rx_clk_out
stat_rx_local_fault	Output	This output is High when stat_rx_internal_local_fault or stat_rx_received_local_fault is asserted. This output is level sensitive.	rx_clk_out

Table 2-8: **AXI4-Stream Interface–RX Path Control/Status Signals *(Cont'd)***

Name	Direction	Description	Clock Domain
stat_rx_block_lock[1-1:0]	Output	Block lock status. A value of 1 indicates that block lock is achieved as defined in Clause 49.2.14 and MDIO register 3.32.0 This output is level sensitive.	rx_clk_out
stat_rx_remote_fault	Output	Remote fault indication status. If this bit is sampled as a 1, it indicates a remote fault condition was detected. If this bit is sampled as a 0, remote fault condition does not exist. This output is level sensitive.	rx_clk_out
stat_rx_bad_fcs[2-1:0]	Output	Bad FCS indicator. The value on this bus indicates packets received with a bad FCS, but not a stomped FCS during a cycle. A stomped FCS is defined as the bitwise inverse of the expected good FCS. This output is pulsed for one clock cycle to indicate an error condition. Note that pulses can occur in back to back cycles.	rx_clk_out
stat_rx_stomped_fcs[2-1:0]	Output	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. Note that pulses can occur in back to back cycles.	rx_clk_out
stat_rx_truncated	Output	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. Note that pulses can occur in back to back cycles.	rx_clk_out
stat_rx_internal_local_fault	Output	High when an internal local fault is generated due to any one of the following: test pattern generation or high bit error rate. Note that this signal remains High as long as the fault condition persists.	rx_clk_out
stat_rx_received_local_fault	Output	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.	rx_clk_out

Table 2-8: **AXI4-Stream Interface–RX Path Control/Status Signals *(Cont'd)***

Name	Direction	Description	Clock Domain
stat_rx_hi_ber	Output	High Bit Error Rate (BER) indicator. When set to 1, the BER is too high as defined by IEEE Std. 802.3. Corresponds to MDIO register bit 3.32.1 as defined in Clause 49.2.14. This output is level sensitive.	rx_clk_out
ctl_rx_custom_preamble_enable	Input	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_clk_out

Miscellaneous Status/Control Signals

Table 2-9 shows the miscellaneous status and control I/O signals.

Table 2-9: **Miscellaneous Status/Control Ports**

Name	Direction	Description	Clock Domain
dclk	Input	Dynamic reconfiguration port (DRP) clock input. The required frequency is indicated on the readme file for the release.	Refer to Clocking.
stat_rx_valid_ctrl_code	Output	Indicates that a PCS block with a valid control code was received.	rx_clk_out
ctl_local_loopback	Input	Loopback enable. A value of 1 enables loopback as defined in Clause 49. Corresponds to management data input/output (MDIO) register bit 3.0.14 as defined in Clause 45. This input should only be changed while the corresponding reset input is asserted.	Asynch
stat_rx_got_signal_os	Output	Signal OS indication. If this bit is sampled as a 1, it indicates that a Signal OS word was received. Note that Signal OS should not be received in an Ethernet network.	rx_clk_out
ctl_rx_process_lfi	Input	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.	rx_clk_out

Table 2-9: **Miscellaneous Status/Control Ports *(Cont'd)***

Name	Direction	Description	Clock Domain
ctl_rx_test_pattern	Input	Test pattern checking enable for the RX core. A value of 1 enables test mode as defined in Clause 49. Corresponds to MDIO register bit 3.42.2 as defined in Clause 45. Checks for scrambled idle pattern.	rx_clk_out
ctl_tx_test_pattern	Input	Test pattern generation enable for the TX core. A value of 1 enables test mode as defined in Clause 49. Corresponds to MDIO register bit 3.42.3 as defined in Clause 45. Generates a scrambled idle pattern.	tx_clk_out
stat_rx_test_pattern_mismatch[1-1:0]	Output	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.	rx_clk_out
ctl_rx_data_pattern_select	Input	Corresponds to MDIO register bit 3.42.0 as defined in Clause 45.	rx_clk_out
ctl_rx_test_pattern_enable	Input	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.	rx_clk_out
ctl_tx_data_pattern_select	Input	Corresponds to MDIO register bit 3.42.0 as defined in Clause 45.	tx_clk_out
ctl_tx_test_pattern_enable	Input	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.	tx_clk_out
ctl_tx_test_pattern_seed_a[57:0]	Input	Corresponds to MDIO registers 3.34 through to 3.37 as defined in Clause 45.	tx_clk_out
ctl_tx_test_pattern_seed_b[57:0]	Input	Corresponds to MDIO registers 3.38 through to 3.41 as defined in Clause 45.	tx_clk_out
ctl_tx_test_pattern_select	Input	Corresponds to MDIO register bit 3.42.1 as defined in Clause 45.	tx_clk_out

Statistics Interface Ports

Table 2-10 and Table 2-11 show the Statistics interface I/O ports.

Table 2-10: **Statistics Interface - RX Path**

Name	Direction	Description	Clock Domain
stat_rx_total_bytes[4-1:0]	Output	Increment for the total number of bytes received.	rx_clk_out
stat_rx_total_packets[2-1:0]	Output	Increment for the total number of packets received.	rx_clk_out
stat_rx_total_good_bytes[14-1:0]	Output	Increment for the total number of good bytes received. This value is only non-zero when a packet is received completely and contains no errors.	rx_clk_out
stat_rx_total_good_packets	Output	Increment for the total number of good packets received. This value is only non-zero when a packet is received completely and contains no errors.	rx_clk_out
stat_rx_packet_bad_fcs	Output	Increment for packets between 64 and ctl_rx_max_packet_len bytes that have Frame Check Sequence (FCS) errors.	rx_clk_out
stat_rx_packet_64_bytes	Output	Increment for good and bad packets received that contain 64 bytes.	rx_clk_out
stat_rx_packet_65_127_bytes	Output	Increment for good and bad packets received that contain 65 to 127 bytes.	rx_clk_out
stat_rx_packet_128_255_bytes	Output	Increment for good and bad packets received that contain 128 to 255 bytes.	rx_clk_out
stat_rx_packet_256_511_bytes	Output	Increment for good and bad packets received that contain 256 to 511 bytes.	rx_clk_out
stat_rx_packet_512_1023_bytes	Output	Increment for good and bad packets received that contain 512 to 1,023 bytes.	rx_clk_out
stat_rx_packet_1024_1518_bytes	Output	Increment for good and bad packets received that contain 1,024 to 1,518 bytes.	rx_clk_out
stat_rx_packet_1519_1522_bytes	Output	Increment for good and bad packets received that contain 1519 to 1522 bytes.	rx_clk_out
stat_rx_packet_1523_1548_bytes	Output	Increment for good and bad packets received that contain 1,523 to 1,548 bytes.	rx_clk_out
stat_rx_packet_1549_2047_bytes	Output	Increment for good and bad packets received that contain 1,549 to 2,047 bytes.	rx_clk_out
stat_rx_packet_2048_4095_bytes	Output	Increment for good and bad packets received that contain 2,048 to 4,095 bytes.	rx_clk_out
stat_rx_packet_4096_8191_bytes	Output	Increment for good and bad packets received that contain 4,096 to 8,191 bytes.	rx_clk_out

Table 2-10: **Statistics Interface - RX Path *(Cont'd)***

Name	Direction	Description	Clock Domain
stat_rx_packet_8192_9215_bytes	Output	Increment for good and bad packets received that contain 8,192 to 9,215 bytes.	rx_clk_out
stat_rx_packet_small	Output	Increment for all packets that are less than 64 bytes long. Packets that are less than 64 bytes are dropped.	rx_clk_out
stat_rx_packet_large	Output	Increment for all packets that are more than 9,215 bytes long.	rx_clk_out
stat_rx_unicast	Output	Increment for good unicast packets.	rx_clk_out
stat_rx_multicast	Output	Increment for good multicast packets.	rx_clk_out
stat_rx_broadcast	Output	Increment for good broadcast packets.	rx_clk_out
stat_rx_oversize	Output	Increment for packets longer than ctl_rx_max_packet_len with good FCS.	rx_clk_out
stat_rx_toolong	Output	Increment for packets longer than ctl_rx_max_packet_len with good and bad FCS.	rx_clk_out
stat_rx_undersize	Output	Increment for packets shorter than stat_rx_min_packet_len with good FCS.	rx_clk_out
stat_rx_fragment	Output	Indicates the increment for packets shorter than stat_rx_min_packet_len with bad FCS.	rx_clk_out
stat_rx_vlan	Output	Increment for good 802.1Q tagged VLAN packets.	rx_clk_out
stat_rx_inrangeerr	Output	Increment for packets with Length field error but with good FCS.	rx_clk_out
stat_rx_jabber	Output	Increment for packets longer than ctl_rx_max_packet_len with bad FCS.	rx_clk_out
stat_rx_pause	Output	Increment for 802.3x MAC Pause packet with good FCS.	rx_clk_out
stat_rx_user_pause	Output	Increment for priority based pause packets with good FCS.	rx_clk_out
stat_rx_bad_code[1-1:0]	Output	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.	rx_clk_out

Table 2-10: **Statistics Interface - RX Path *(Cont'd)***

Name	Direction	Description	Clock Domain
stat_rx_bad_sfd	Output	Increment 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.	rx_clk_out
stat_rx_bad_preamble	Output	Increment 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.	rx_clk_out

Table 2-11: **Statistics Interface - TX Path**

Name	Direction	Description	Clock Domain
stat_tx_total_bytes[4-1:0]	Output	Increment for the total number of bytes transmitted.	tx_clk_out
stat_tx_total_packets	Output	Increment for the total number of packets transmitted.	tx_clk_out
stat_tx_total_good_bytes[14-1:0]	Output	Increment for the total number of good bytes transmitted. This value is only non-zero when a packet is transmitted completely and contains no errors.	tx_clk_out
stat_tx_total_good_packets	Output	Increment for the total number of good packets transmitted.	tx_clk_out
stat_tx_bad_fcs	Output	Increment for packets greater than 64 bytes that have FCS errors.	tx_clk_out
stat_tx_packet_64_bytes	Output	Increment for good and bad packets transmitted that contain 64 bytes.	tx_clk_out
stat_tx_packet_65_127_bytes	Output	Increment for good and bad packets transmitted that contain 65 to 127 bytes.	tx_clk_out
stat_tx_packet_128_255_bytes	Output	Increment for good and bad packets transmitted that contain 128 to 255 bytes.	tx_clk_out
stat_tx_packet_256_511_bytes	Output	Increment for good and bad packets transmitted that contain 256 to 511 bytes.	tx_clk_out
stat_tx_packet_512_1023_bytes	Output	Increment for good and bad packets transmitted that contain 512 to 1,023 bytes.	tx_clk_out
stat_tx_packet_1024_1518_bytes	Output	Increment for good and bad packets transmitted that contain 1,024 to 1,518 bytes.	tx_clk_out
stat_tx_packet_1519_1522_bytes	Output	Increment for good and bad packets transmitted that contain 1,519 to 1,522 bytes.	tx_clk_out

Table 2-11: **Statistics Interface - TX Path *(Cont'd)***

Name	Direction	Description	Clock Domain
stat_tx_packet_1523_1548_bytes	Output	Increment for good and bad packets transmitted that contain 1,523 to 1,548 bytes.	tx_clk_out
stat_tx_packet_1549_2047_bytes	Output	Increment for good and bad packets transmitted that contain 1,549 to 2,047 bytes.	tx_clk_out
stat_tx_packet_2048_4095_bytes	Output	Increment for good and bad packets transmitted that contain 2,048 to 4,095 bytes.	tx_clk_out
stat_tx_packet_4096_8191_bytes	Output	Increment for good and bad packets transmitted that contain 4,096 to 8,191 bytes.	tx_clk_out
stat_tx_packet_8192_9215_bytes	Output	Increment for good and bad packets transmitted that contain 8,192 to 9,215 bytes.	tx_clk_out
stat_tx_packet_small	Output	Increment for all packets that are less than 64 bytes long.	tx_clk_out
stat_tx_packet_large	Output	Increment for all packets that are more than 9,215 bytes long.	tx_clk_out
stat_tx_unicast	Output	Increment for good unicast packets.	tx_clk_out
stat_tx_multicast	Output	Increment for good multicast packets.	tx_clk_out
stat_tx_broadcast	Output	Increment for good broadcast packets.	tx_clk_out
stat_tx_vlan	Output	Increment for good 802.1Q tagged VLAN packets.	tx_clk_out
stat_tx_pause	Output	Increment for 802.3x MAC Pause packet with good FCS.	tx_clk_out
stat_tx_user_pause	Output	Increment for priority based pause packets with good FCS.	tx_clk_out
stat_tx_frame_error	Output	Increment for packets with tx_errin set to indicate an End of Packet (EOP) abort.	tx_clk_out

Pause Interface

Table 2-12 through Table 2-14 show the Pause interface I/O ports.

Table 2-12: **Pause Interface–Control Ports**

Name	Direction	Description	Clock Domain
ctl_rx_pause_enable[9-1:0]	Input	RX pause enable signal. This input is used to enable the processing of the pause quanta for the corresponding priority. Note that this signal only affects the RX user interface, not the pause processing logic.	rx_clk_out
ctl_tx_pause_enable[9-1:0]	Input	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.	tx_clk_out

Table 2-13: **Pause Interface–RX Path**

Name	Direction	Description	Clock Domain
ctl_rx_enable_gcp	Input	A value of 1 enables global control packet processing.	rx_clk_out
ctl_rx_check_mcast_gcp	Input	A value of 1 enables global control multicast destination address processing.	rx_clk_out
ctl_rx_check_ucast_gcp	Input	A value of 1 enables global control unicast destination address processing.	rx_clk_out
ctl_rx_pause_da_ucast[47:0]	Input	Unicast destination address for pause processing.	rx_clk_out
ctl_rx_check_sa_gcp	Input	A value of 1 enables global control source address processing.	rx_clk_out
ctl_rx_pause_sa[47:0]	Input	Source address for pause processing.	rx_clk_out
ctl_rx_check_etype_gcp	Input	A value of 1 enables global control ethertype processing.	rx_clk_out
ctl_rx_check_opcode_gcp	Input	A value of 1 enables global control opcode processing.	rx_clk_out
ctl_rx_opcode_min_gcp[15:0]	Input	Minimum global control opcode value.	rx_clk_out
ctl_rx_opcode_max_gcp[15:0]	Input	Maximum global control opcode value.	rx_clk_out
ctl_rx_etype_gcp[15:0]	Input	Ethertype field for global control processing.	rx_clk_out
ctl_rx_enable_pcp	Input	A value of 1 enables priority control packet processing.	rx_clk_out
ctl_rx_check_mcast_pcp	Input	A value of 1 enables priority control multicast destination address processing.	rx_clk_out
ctl_rx_check_ucast_pcp	Input	A value of 1 enables priority control unicast destination address processing.	rx_clk_out

Table 2-13: **Pause Interface–RX Path *(Cont'd)***

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

Table 2-13: **Pause Interface–RX Path *(Cont'd)***

Name	Direction	Description	Clock Domain
ctl_rx_check_opcode_ppp	Input	A value of 1 enables priority pause opcode processing.	rx_clk_out
ctl_rx_opcode_ppp[15:0]	Input	Priority pause opcode value.	rx_clk_out
stat_rx_pause_req[9-1:0]	Output	Pause request signal. When the RX receives a valid pause frame, it sets the corresponding bit of this bus to a 1 and keep it at 1 until the pause packet has been processed.	rx_clk_out
ctl_rx_pause_ack[9-1:0]	Input	Pause acknowledge signal. This bus is used to acknowledge the receipt of the pause frame from the user logic.	rx_clk_out
ctl_rx_check_ack	Input	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.	rx_clk_out
ctl_rx_forward_control	Input	A value of 1 indicates that the core forwards control packets. A value of 0 causes core to drop control packets.	rx_clk_out
stat_rx_pause_valid[9-1:0]	Output	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.	rx_clk_out
stat_rx_pause_quanta[8:0][15:0]	Output	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].	rx_clk_out

Table 2-14: **Pause Interface–TX Path**

Name	Direction	Description	Clock Domain
ctl_tx_pause_req[9-1:0]	Input	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.	tx_clk_out
ctl_tx_pause_quanta[8:0][15:0]	Input	These nine buses indicate the quanta to be transmitted for each of the eight priorities in priority based pause operation and the global pause operation. The value for stat_tx_pause_quanta[8] is used for global pause operation. All other values are used for priority pause operation.	tx_clk_out
ctl_tx_pause_refresh_timer [8:0][15:0]	Input	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 value for stat_tx_pause_refresh_timer[8] is used for global pause operation. All other values are used for priority pause operation.	tx_clk_out
ctl_tx_da_gpp[47:0]	Input	Destination address for transmitting global pause packets.	tx_clk_out
ctl_tx_sa_gpp[47:0]	Input	Source address for transmitting global pause packets.	tx_clk_out
ctl_tx_ethertype_gpp[15:0]	Input	Ethertype for transmitting global pause packets.	tx_clk_out
ctl_tx_opcode_gpp[15:0]	Input	Opcode for transmitting global pause packets.	tx_clk_out
ctl_tx_da_ppp[47:0]	Input	Destination address for transmitting priority pause packets.	tx_clk_out
ctl_tx_sa_ppp[47:0]	Input	Source address for transmitting priority pause packets.	tx_clk_out
ctl_tx_ethertype_ppp[15:0]	Input	Ethertype for transmitting priority pause packets.	tx_clk_out
ctl_tx_opcode_ppp[15:0]	Input	Opcode for transmitting priority pause packets.	tx_clk_out

Table 2-14: **Pause Interface–TX Path *(Cont'd)***

Name	Direction	Description	Clock Domain
ctl_tx_resend_pause	Input	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 should be pulsed to 1 for one cycle at a time.	tx_clk_out
stat_tx_pause_valid[9-1:0]	Output	If a bit of this bus is set to 1, the core has transmitted a pause packet. If bit[8] is set to 1, a global pause packet is transmitted. All other bits cause a priority pause packet to be transmitted.	tx_clk_out

Auto-Negotiation Ports

Table 2-15 shows the additional ports used for Auto-Negotiation. These signals are found at the `*wrapper.v` hierarchy file.

Table 2-15: **Additional Ports for Auto-Negotiation**

Port Name	Direction	Description	Clock Domain
an_clk	Input	Input Clock for the auto-negotiation circuit. The required frequency is indicated in the readme file for the release.	Refer to Clocking.
an_reset	Input	Asynchronous active High reset.	Asynch
ctl_autoneg_enable	Input	Enable signal for autonegotiation.	an_clk
ctl_autoneg_bypass	Input	Input to disable autonegotiation and bypass the autonegotiation function. If this input is asserted, then autonegotiation is turned off, but the PCS is connected to the outputs to allow operation.	an_clk
ctl_an_nonce_seed[7:0]	Input	8-bit seed to initialize the nonce field Polynomial generator.	an_clk
ctl_an_pseudo_sel	Input	Selects the polynomial generator for the bit 49 random bit generator. If this input is 1, then the polynomial is x^7+x^6+1. If this input is zero, then the polynomial is x^7+x^3+1.	an_clk
ctl_restart_negotiation	Input	This input is used to trigger a restart of the auto negotiation, regardless of what state the circuit is currently in.	an_clk

Table 2-15: **Additional Ports for Auto-Negotiation *(Cont'd)***

Port Name	Direction	Description	Clock Domain
ctl_an_local_fault	Input	This input signal is used to set the local_fault bit of the transmit link codeword.	an_clk
Signals Used for Pause Ability Advertising			
ctl_an_pause	Input	This input signal is used to set the PAUSE bit, (C0), of the transmit link codeword. This signal might not be present if the core does not support pause.	an_clk
ctl_an_asmdir	Input	This input signal is used to set the ASMDIR bit, (C1), of the transmit link codeword. This signal might not be present if the core does not support pause.	an_clk
Ability Signal Inputs			
ctl_an_ability_1000base_kx	Input	These inputs identify the Ethernet protocol abilities that is advertised in the transmit link codeword to the link partner. A value of 1 indicates that the interface advertises that it supports the protocol.	an_clk
ctl_an_ability_100gbase_cr10	Input		an_clk
ctl_an_ability_100gbase_cr4	Input		an_clk
ctl_an_ability_100gbase_kp4	Input		an_clk
ctl_an_ability_100gbase_kr4	Input		an_clk
ctl_an_ability_10gbase_kr	Input		an_clk
ctl_an_ability_10gbase_kx4	Input		an_clk
ctl_an_ability_25gbase_cr	Input		an_clk
ctl_an_ability_25gbase_cr1	Input		an_clk
ctl_an_ability_25gbase_kr	Input		an_clk
ctl_an_ability_25gbase_kr1	Input		an_clk
ctl_an_ability_40gbase_cr4	Input		an_clk
ctl_an_ability_40gbase_kr4	Input		an_clk
ctl_an_ability_50gbase_cr2	Input		an_clk
ctl_an_ability_50gbase_kr2	Input		an_clk
ctl_an_fec_request	Input	Used to control the clause 74 FEC request bit in the transmit link codeword. This signal might not be present if the IP core does not support clause 74 FEC.	an_clk

Table 2-15: **Additional Ports for Auto-Negotiation *(Cont'd)***

Port Name	Direction	Description	Clock Domain
ctl_an_fec_ability_override	Input	Used to control the clause 74 FEC ability bit in the transmit link codeword. If this input is set, then the FEC ability bit in the transmit link codeword is cleared. This signal might not be present if the IP core does not support clause 74 FEC.	an_clk
ctl_an_cl91_fec_ability	Input	This bit is used to indicate clause 91 FEC ability.	an_clk
ctl_an_cl91_fec_request	Input	This bit is used to request clause 91 FEC.	an_clk
stat_an_link_cntl_1000base_kx[1:0]	Output	Link Control outputs from the auto negotiation controller for the various Ethernet protocols. Settings are as follows: • 00: DISABLE; PCS is disconnected; • 01: SCAN_FOR_CARRIER; RX is connected to PCS; • 11: ENABLE; PCS is connected for mission mode operation. • 10: not used	an_clk
stat_an_link_cntl_100gbase_cr10[1:0]	Output		an_clk
stat_an_link_cntl_100gbase_cr4[1:0]	Output		an_clk
stat_an_link_cntl_100gbase_kp4[1:0]	Output		an_clk
stat_an_link_cntl_100gbase_kr4[1:0]	Output		an_clk
stat_an_link_cntl_10gbase_kr[1:0]	Output		an_clk
stat_an_link_cntl_10gbase_kx4[1:0]	Output		an_clk
stat_an_link_cntl_25gbase_cr[1:0]	Output		an_clk
stat_an_link_cntl_25gbase_cr1[1:0]	Output		an_clk
stat_an_link_cntl_25gbase_kr[1:0]	Output		an_clk
stat_an_link_cntl_25gbase_kr1[1:0]	Output		an_clk
stat_an_link_cntl_40gbase_cr4[1:0]	Output		an_clk
stat_an_link_cntl_40gbase_kr4[1:0]	Output		an_clk
stat_an_link_cntl_50gbase_cr2[1:0]	Output		an_clk
stat_an_link_cntl_50gbase_kr2[1:0]	Output		an_clk
stat_an_fec_enable	Output	Used to enable the use of clause 74 FEC on the link.	an_clk
stat_an_rs_fec_enable	Output	Used to enable the use of clause 91 FEC on the link.	an_clk
stat_an_tx_pause_enable	Output	Used to enable station-to-station (global) pause packet generation in the transmit path to control data flow in the receive path.	an_clk
stat_an_rx_pause_enable	Output	Used to enable station-to-station (global) pause packet interpretation in the receive path, to control data flow from the transmitter.	an_clk

Table 2-15: **Additional Ports for Auto-Negotiation *(Cont'd)***

Port Name	Direction	Description	Clock Domain
stat_an_autoneg_complete	Output	Indicates the auto-negotiation is complete and RX link status from the PCS has been received.	an_clk
stat_an_parallel_detection_fault	Output	Indicated a parallel detection fault during auto-negotiation.	an_clk
stat_an_lp_ability_1000base_kx	Output	These signals indicate the advertised protocol from the link partner. They all become valid when the output signal stat_AN_lp_Ability_Valid is asserted. A value of 1 indicates that the protocol is advertised as supported by the link partner.	an_clk
stat_an_lp_ability_100gbase_cr10	Output		an_clk
stat_an_lp_ability_100gbase_cr4	Output		an_clk
stat_an_lp_ability_100gbase_kp4	Output		an_clk
stat_an_lp_ability_100gbase_kr4	Output		an_clk
stat_an_lp_ability_10gbase_kr	Output		an_clk
stat_an_lp_ability_10gbase_kx4	Output		an_clk
stat_an_lp_ability_25gbase_cr	Output		an_clk
stat_an_lp_ability_25gbase_kr	Output		an_clk
stat_an_lp_ability_40gbase_cr4	Output		an_clk
stat_an_lp_ability_40gbase_kr4	Output		an_clk
stat_an_lp_ability_25gbase_cr1	Output	Indicates the advertised protocol from the link partner. Becomes valid when the output signal stat_AN_lp_Extended_Ability_Valid is asserted. A value of 1 indicates that the protocol is advertised as supported by the link partner.	an_clk
stat_an_lp_ability_25gbase_kr1	Output	Indicates the advertised protocol from the link partner. Becomes valid when the output signal stat_AN_lp_Extended_Ability_Valid is asserted. A value of 1 indicates that the protocol is advertised as supported by the link partner.	an_clk
stat_an_lp_ability_50gbase_cr2	Output	Indicates the advertised protocol from the link partner. Becomes valid when the output signal stat_AN_lp_Extended_Ability_Valid is asserted. A value of 1 indicates that the protocol is advertised as supported by the link partner.	an_clk

Table 2-15: **Additional Ports for Auto-Negotiation *(Cont'd)***

Port Name	Direction	Description	Clock Domain
stat_an_lp_ability_50gbase_kr2	Output	Indicates the advertised protocol from the link partner. Becomes valid when the output signal stat_AN_lp_Extended_Ability_Valid is asserted. A value of 1 indicates that the protocol is advertised as supported by the link partner.	an_clk
stat_an_lp_pause	Output	This signal indicates the advertised value of the PAUSE bit, (C0), in the receive link codeword from the link partner. It becomes valid when the output signal stat_AN_lp_Ability_Valid is asserted.	an_clk
stat_an_lp_asm_dir	Output	This signal indicates the advertised value of the ASMDIR bit, (C1), in the receive link codeword from the link partner. It becomes valid when the output signal stat_AN_lp_Ability_Valid is asserted.	an_clk
stat_an_lp_fec_ability	Output	This signal indicates the advertised value of the FEC ability bit in the receive link codeword from the link partner. It becomes valid when the output signal stat_AN_lp_Ability_Valid is asserted.	an_clk
stat_an_lp_fec_request	Output	This signal indicates the advertised value of the FEC Request bit in the receive link codeword from the link partner. It becomes valid when the output signal stat_AN_lp_Ability_Valid is asserted.	an_clk
stat_an_lp_autoneg_able	Output	This output signal indicates that the link partner is able to perform autonegotiation. It becomes valid when the output signal stat_AN_lp_Ability_Valid is asserted.	an_clk
stat_an_lp_ability_valid	Output	This signal indicates when all of the link partner advertisements become valid.	an_clk

Table 2-15: **Additional Ports for Auto-Negotiation *(Cont'd)***

Port Name	Direction	Description	Clock Domain
an_loc_np_data[47:0]	Input	Local Next Page codeword. This is the 48 bit codeword used if the loc_np input is set. In this data field, the bits NP, ACK, and T, bit positions 15, 14, 12, and 11, are not transferred as part of the next page codeword. These bits are generated in the Auto-Negotiation Intellectual Property Core (ANIPC). However, the Message Protocol bit, MP, in bit position 13, is transferred.	an_clk
an_lp_np_data[47:0]	Output	Link Partner Next Page Data. This 48-bit word is driven by the ANIPC with the 48-bit next page codeword from the remote link partner.	an_clk
ctl_an_loc_np	Input	Local Next Page indicator. If this bit is 1, the ANIPC transfers the next page word at input loc_np_data to the remote link partner. If this bit is 0, the ANIPC does not initiate the next page protocol. If the link partner has next pages to send, and the loc_np bit is clear, the ANIPC transfers null message pages.	an_clk
ctl_an_lp_np_ack	Input	Link Partner Next Page Acknowledge. This is used to signal the ANIPC that the next page data from the remote link partner at output pin lp_np_data has been read by the local host. When this signal goes High, the ANIPC acknowledges reception of the next page codeword to the remote link partner and initiate transfer of the next codeword. During this time, the ANIPC will remove the lp_np signal until the new next page information is available.	an_clk

Table 2-15: **Additional Ports for Auto-Negotiation *(Cont'd)***

Port Name	Direction	Description	Clock Domain
stat_an_loc_np_ack	Output	This signal is used to indicate to the local host that the local next page data, presented at input pin loc_np_data, has been taken. This signal pulses High for 1 clock period when the ANIPC samples the next page data on input pin loc_np_data. When the local host detects this signal High, it must replace the 48 bit next page codeword at input pin loc_np_data with the next 48 bit codeword to be sent. If the local host has no more next pages to send, it must clear the loc_np input.	an_clk
stat_an_lp_np	Output	Link Partner Next Page. This signal is used to indicate that there is a valid 48 bit next page codeword from the remote link partner at output pin lp_np_data. This signal is driven Low when the lp_np_ack input signal is driven High, indicating that the local host has read the next page data. It remains Low until the next codeword becomes available on the lp_np_data output pin, the lp_np output is driven High again.	an_clk
stat_an_lp_ability_extended_fec[1:0]	Output	This output indicates the extended FEC abilities as defined in Schedule 3.	an_clk
stat_an_lp_extended_ability_valid	Output	When this bit is 1, it indicates that the detected extended abilities are valid.	an_clk
stat_an_lp_rf	Output	This bit indicates link partner remote fault.	an_clk

Table 2-15: **Additional Ports for Auto-Negotiation *(Cont'd)***

Port Name	Direction	Description	Clock Domain
stat_an_start_tx_disable	Output	When ctl_autoneg_enable is High and ctl_autoneg_bypass is Low, this signal, stat_an_start_tx_disable, cycles High for 1 clock cycle at the very start of the TX_DISABLE phase of auto-negotiation. That is, when auto-negotiation enters state TX_DISABLE, this output will cycle High for 1 clock period. It effectively signals the start of auto-negotiation.	an_clk
stat_an_start_an_good_check	Output	When ctl_autoneg_enable is High and ctl_autoneg_bypass is Low, this signal, stat_an_start_an_good_check, cycles High for 1 clock cycle at the very start of the AN_GOOD_CHECK phase of auto-negotiation. That is, when auto-negotiation enters the state AN_GOOD_CHECK, this output will cycle High for 1 clock period. It effectively signals the start of link training. However, if link training is not enabled, that is. if the input ctl_lt_training_enable is Low, the stat_an_start_an_good_check output effectively signals the start of mission-mode operation.	an_clk

Link Training Ports

Table 2-16 shows the Link Training ports.

Table 2-16: **Link Training Ports**

Port Name	Direction	Description	Clock Domain
ctl_lt_training_enable	Input	Enables link training. When link training is disabled, all PCS lanes function in mission mode.	tx_serdes_clk
ctl_lt_restart_training	Input	This signal triggers a restart of link training regardless of the current state.	tx_serdes_clk
ctl_lt_rx_trained[1-1:0]	Input	This signal is asserted to indicate that the receiver finite impulse response (FIR) filter coefficients have all been set, and that the receiver portion of training is complete.	tx_serdes_clk

Table 2-16: **Link Training Ports *(Cont'd)***

Port Name	Direction	Description	Clock Domain
stat_lt_signal_detect[1-1:0]	Output	This signal indicates when the respective link training state machine has entered the SEND_DATA state, in which normal PCS operation can resume.	tx_serdes_clk
stat_lt_training[1-1:0]	Output	This signal indicates when the respective link training state machine is performing link training.	tx_serdes_clk
stat_lt_training_fail[1-1:0]	Output	This signal is asserted during link training if the corresponding link training state machine detects a time-out during the training period.	tx_serdes_clk
stat_lt_frame_lock[1-1:0]	Output	When link training has begun, these signals are asserted, for each physical medium dependent (PMD) lane, when the corresponding link training receiver is able to establish a frame synchronization with the link partner.	rx_serdes_clk
stat_lt_preset_from_rx[1-1:0]	Output	This signal reflects the value of the preset control bit received in the control block from the link partner	rx_serdes_clk
stat_lt_initialize_from_rx[1-1:0]	Output	This signal reflects the value of the initialize control bit received in the control block from the link partner	rx_serdes_clk
stat_lt_k_p1_from_rx0[1:0]	Output	This 2-bit field indicates the update control bits for the k+1 coefficient, as received from the link partner in the control block	rx_serdes_clk
stat_lt_k0_from_rx0[1:0]	Output	This 2-bit field indicates the update control bits for the k0 coefficient, as received from the link partner in the control block.	rx_serdes_clk
stat_lt_k_m1_from_rx0[1:0]	Output	This 2-bit field indicates the update control bits for the k-1 coefficient, as received from the link partner in the control block.	rx_serdes_clk
stat_lt_stat_p1_from_rx0[1:0]	Output	This 2-bit field indicates the update status bits for the k+1 coefficient, as received from the link partner in the status block.	rx_serdes_clk
stat_lt_stat0_from_rx0[1:0]	Output	This 2-bit fields indicates the update status bits for the k0 coefficient, as received from the link partner in the status block.	rx_serdes_clk

Table 2-16: **Link Training Ports *(Cont'd)***

Port Name	Direction	Description	Clock Domain
stat_lt_stat_m1_from_rx0[1:0]	Output	This 2-bit field indicates the update status bits for the k-1 coefficient, as received from the link partner in the status block	rx_serdes_clk
ctl_lt_pseudo_seed0[10:0]	Input	This 11-bit signal seeds the training pattern generator.	tx_serdes_clk
ctl_lt_preset_to_tx[1-1:0]	Input	This signal is used to set the value of the preset bit that is transmitted to the link partner in the control block of the training frame.	tx_serdes_clk
ctl_lt_initialize_to_tx[1-1:0]	Input	This signal is used to set the value of the initialize bit that is transmitted to the link partner in the control block of the training frame.	tx_serdes_clk
ctl_lt_k_p1_to_tx0[1:0]	Input	This 2-bit field is used to set the value of the k+1 coefficient update field that is transmitted to the link partner in the control block of the training frame.	tx_serdes_clk
ctl_lt_k0_to_tx0[1:0]	Input	This 2-bit field is used to set the value of the k0 coefficient update field that is transmitted to the link partner in the control block of the training frame.	tx_serdes_clk
ctl_lt_k_m1_to_tx0[1:0]	Input	This 2-bit field is used to set the value of the k-1 coefficient update field that is transmitted to the link partner in the control block of the training frame.	tx_serdes_clk
ctl_lt_stat_p1_to_tx0[1:0]	Input	This 2-bit field is used to set the value of the k+1 coefficient update status that is transmitted to the link partner in the status block of the training frame.	tx_serdes_clk
ctl_lt_stat0_to_tx0[1:0]	Input	This 2-bit field is used to set the value of the k0 coefficient update status that is transmitted to the link partner in the status block of the training frame.	tx_serdes_clk
ctl_lt_stat_m1_to_tx0[1:0]	Input	This 2-bit field is used to set the value of the k-1 coefficient update status that is transmitted to the link partner in the status block of the training frame,.	tx_serdes_clk
stat_lt_rx_sof[1-1:0]	Output	This output is High for 1 RX SerDes clock cycle to indicate the start of the link training frame.	rx_serdes_clk

Port Descriptions – PCS Variant

This section shows the 10G/25G PCS core ports. These are the ports when the PCS-only option is provided. There are no FCS functions. The PCS does not contain the Pause and Flow Control ports. The system interface is XXVMII. Table 2-17 shows the PCS variant I/O ports.

Table 2-17: **PCS Variant I/O Ports**

Name	Direction	Description	Clock Domain
stat_tx_local_fault	Output	A value of 1 indicates the transmit encoder state machine is in the TX_INIT state. This output is level sensitive.	tx_mii_clk
ctl_rx_prbs31_test_pattern_enable	Input	Corresponds to MDIO register bit 3.42.5 as defined in Clause 45. Takes first precedence.	rx_clk_out
ctl_rx_test_pattern_enable	Input	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.	rx_clk_out
ctl_rx_data_pattern_select	Input	Corresponds to MDIO register bit 3.42.0 as defined in Clause 45.	rx_clk_out
ctl_rx_test_pattern	Input	Test pattern enable for the RX core to receive scrambled idle pattern. Takes third precedence.	rx_clk_out
ctl_tx_prbs31_test_pattern_enable	Input	Corresponds to MDIO register bit 3.42.4 as defined in Clause 45. Takes first precedence.	tx_mii_clk
ctl_tx_test_pattern_enable	Input	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.	tx_mii_clk
ctl_tx_test_pattern_select	Input	Corresponds to MDIO register bit 3.42.1 as defined in Clause 45.	tx_mii_clk
ctl_tx_data_pattern_select	Input	Corresponds to MDIO register bit 3.42.0 as defined in Clause 45.	tx_mii_clk
ctl_tx_test_pattern_seed_a[57:0]	Input	Corresponds to MDIO registers 3.34 through to 3.37 as defined in Clause 45.	tx_mii_clk

Table 2-17: **PCS Variant I/O Ports *(Cont'd)***

Name	Direction	Description	Clock Domain
ctl_tx_test_pattern_seed_b[57:0]	Input	Corresponds to MDIO registers 3.38 through to 3.41 as defined in Clause 45.	tx_mii_clk
ctl_tx_test_pattern	Input	Scrambled idle Test pattern generation enable for the TX core. A value of 1 enables test mode. Takes third precedence.	tx_mii_clk
stat_rx_fifo_error	Output	Receive clock compensation FIFO error indicator. A value of 1 indicates the clock compensation FIFO under or overflowed. This condition only occurs if the PPM difference between the recovered clock and the local reference clock is greater than ±200 ppm. If this output is sampled as a 1 in any clock cycle, the corresponding port must be reset to resume proper operation.	rx_mii_clk
stat_rx_local_fault	Output	A value of 1 indicates the receive decoder state machine is in the RX_INIT state. This output is level sensitive.	rx_clk_out
stat_rx_hi_ber	Output	High Bit Error Rate (BER) indicator. When set to 1, the BER is too high as defined by the 802.3. Corresponds to MDIO register bit 3.32.1 as defined in Clause 45. This output is level sensitive.	rx_clk_out
stat_rx_block_lock[1-1:0]	Output	Block lock status for each PCS lane. A value of 1 indicates the corresponding lane has achieved a block lock as defined in Clause 49. Corresponds to MDIO register bit 3.50.7:0 and 3.51.11:0 as defined in Clause 45. This output is level sensitive.	rx_clk_out

Table 2-17: **PCS Variant I/O Ports *(Cont'd)***

Name	Direction	Description	Clock Domain
stat_rx_error	Output	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 3.43.15:0 as defined in Clause 45. This output is pulsed for one clock cycle.	rx_clk_out
stat_rx_valid_ctrl_code	Output	Indicates that a PCS block with a valid control code was received.	rx_clk_out
stat_rx_error_valid	Output	Increment valid indicator. If this signal is a 1 in any clock cycle, the value of stat_rx_error_valid[0:0] is valid.	rx_clk_out
stat_rx_bad_code	Output	Increment for 64B/66B code violations. This signal indicates the number of 64b/66b words received with an invalid block or if a wrong 64b/66b block sequence was detected. This output can be used to generate MDIO register 3.33:7:0 as defined in Clause 45.	rx_clk_out
stat_rx_bad_code_valid	Output	Increment valid indicator. If this signal is a 1 in any clock cycle, the value of stat_rx_bad_code[0:0] is valid.	rx_clk_out
stat_rx_framing_err	Output	Increment value for number of bad sync header bits detected. The value of this bus is only valid in the same cycle that the corresponding stat_rx_framing_err_valid is a 1.	rx_clk_out
stat_rx_framing_err_valid	Output	Increment valid indicator. If this signal is a 1 in any clock cycle, the value of stat_rx_framing_err[0:0] is valid.	rx_clk_out

Transceiver Interface Ports

Table 2-18 shows the transceiver I/O ports.

Table 2-18: **Transceiver I/O**

Name	Direction	Description	Clock Domain
GT_reset	Input	Active-High reset for the transceiver startup FSM. Note that this signal also initiates the reset sequence for the entire 10G/25G Ethernet IP core.	Asynch
refclk_n0	Input	Differential reference clock input for the SerDes, negative phase.	Refer to Clocking.
refclk_p0	Input	Differential reference clock input for the SerDes, negative phase.	Refer to Clocking.
rx_serdes_data_n0	Input	Serial data from the line; negative phase of the differential signal.	Refer to Clocking.
rx_serdes_data_p0	Input	Serial data from the line; positive phase of the differential signal.	Refer to Clocking.
tx_serdes_data_n0	Output	Serial data to the line; negative phase of the differential signal.	Refer to Clocking.
tx_serdes_data_p0	Output	Serial data to the line; positive phase of the differential signal.	Refer to Clocking.
tx_serdes_clkout	Output	When present, same as tx_clk_out.	Refer to Clocking.

XXVMII Interface Ports

Table 2-19 shows the XXVMII I/O ports.

Table 2-19: **XXVMII Interface Ports**

Name	Direction	Description	Clock Domain
rx_mii_d[64-1:0]	Output	Receive XLGMII/CGMII Data bus.	rx_mii_clk
rx_mii_c[8-1:0]	Output	Receive XLGMII/CGMII Control bus.	rx_mii_clk
rx_mii_clk	Input	Receive XLGMII/CGMII Clock input.	Refer to Clocking.
tx_mii_d[64-1:0]	Input	Transmit XLGMII/CGMII Data bus.	tx_mii_clk
tx_mii_c[8-1:0]	Input	Transmit XLGMII/CGMII Control bus.	tx_mii_clk
rx_clk_out	Output	This is the reference clock for RX PCS stats.	Refer to Clocking.
tx_clk_out (or tx_mii_clk)	Output	This output is used to clock the TX mii bus. Data is clocked on the positive edge of this signal.	Refer to Clocking.
rx_mii_reset	Input	Reset input for the RX mii interface.	Asynch
tx_mii_reset	Input	Reset input for the TX mii interface.	Asynch

Miscellaneous Status/Control Ports

Table 2-20 shows the miscellaneous status/control ports.

Table 2-20: **Miscellaneous Status/Control Ports**

Name	Direction	Description	Clock Domain
dclk	Input	DRP clock input. The required frequency is indicated on the readme file for the release.	Refer to Clocking.
ctl_local_loopback	Input	When High, this signal places the transceiver into the PMA loopback state.	Asynch

Register Space

The 10/25 G Ethernet can be optionally configured with AXI4-Lite registers to access the configuration and status signals.

AXI4-Lite Ports

Table 2-21 describes the port list for the AXI processor interface.

Table 2-21: **AXI Ports**

Signal	Direction	Description
s_axi_aclk	In	AXI4-Lite clock. Range between 10 MHz and 300 MHz
s_axi_aresetn	In	Asynchronous active-Low reset
s_axi_awaddr[31:0]	In	Write address Bus
s_axi_awvalid	In	Write address valid
s_axi_awready	Out	Write address acknowledge
s_axi_wdata[31:0]	In	Write data bus
s_axi_wstrb[3:0]	In	Strobe signal for the data bus byte lane
s_axi_wvalid	Out	Write data valid
s_axi_wready	Out	Write data acknowledge
s_axi_bresp[1:0]	Out	Write transaction response
s_axi_bvalid	Out	Write response valid
s_axi_bready	In	Write response acknowledge
s_axi_araddr[31:0]	In	Read address bus
s_axi_arvalid	In	Read address valid
s_axi_arready	Out	Read address acknowledge
s_axi_rdata[31:0]	Out	Read data output

Table 2-21: **AXI Ports *(Cont'd)***

Signal	Direction	Description
s_axi_rresp[1:0]	Out	Read data response
s_axi_rvalid	Out	Read data/response valid
s_axi_rready	In	Read data acknowledge
pm_tick	In	Top level signal to read statistics counters; requires MODE_REG[30] to be set to 0.

Additional information for the operation of the AXI4 bus is found in "Xilinx 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 should 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.

In order for the programmed configurations to take effect, it is necessary to issue `tx_reset` and `rx_reset`, which are active-High.

Table 2-22: **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
0x001C	CONFIGURATION_VL_LENGTH_REG: 001C
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

Table 2-22: **Configuration Register Map *(Cont'd)***

Hex Address	Register Name/Link to Description
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
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
0x00D0	CONFIGURATION_RSFEC_REG: 00D0
0x00D4	CONFIGURATION_FEC_REG: 00D4
0x00E0	CONFIGURATION_AN_CONTROL_REG1: 00E0
0x00E4	CONFIGURATION_AN_CONTROL_REG2: 00E4
0x00F8	CONFIGURATION_AN_ABILITY: 00F8
0x0100	CONFIGURATION_LT_CONTROL_REG1: 0100
0x0104	CONFIGURATION_LT_TRAINED_REG: 0104
0x0108	CONFIGURATION_LT_PRESET_REG: 0108
0x010C	CONFIGURATION_LT_INIT_REG: 010C
0x0110	CONFIGURATION_LT_SEED_REG0: 0110
0x0130	CONFIGURATION_LT_COEFFICIENT_REG0: 0130

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.

Table 2-23: **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
0x0444	STAT_RX_RSFEC_STATUS_REG: 0444
0x0448	STAT_RX_FEC_STATUS_REG: 0448
0x0450	STAT_TX_FLOW_CONTROL_REG1: 0450
0x0454	STAT_RX_FLOW_CONTROL_REG1: 0454
0x0458	STAT_AN_STATUS: 0458
0x045C	STAT_AN_ABILITY: 045C
0x0460	STAT_AN_LINK_CTL: 0460
0x0464	STAT_LT_STATUS_REG1: 0464
0x0468	STAT_LT_STATUS_REG2: 0468
0x046C	STAT_LT_STATUS_REG3: 046C
0x0470	STAT_LT_STATUS_REG4: 0470
0x0474	STAT_LT_COEFFICIENT0_REG: 0474
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]. `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.

Table 2-24: **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
0x0670	STAT_RX_RSFEC_CORRECTED_CW_INC_LSB: 0670
0x0674	STAT_RX_RSFEC_CORRECTED_CW_INC_MSB: 0674
0x0678	STAT_RX_RSFEC_UNCORRECTED_CW_INC_LSB: 0678
0x067C	STAT_RX_RSFEC_UNCORRECTED_CW_INC_MSB: 067C
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
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

Table 2-24: **Statistics Counters *(Cont'd)***

Hex Address	Register Name/Link to Description
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
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

Table 2-24: **Statistics Counters *(Cont'd)***

Hex Address	Register Name/Link to Description
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
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

Table 2-24: **Statistics Counters *(Cont'd)***

Hex Address	Register Name/Link to Description
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
0x0920	STAT_FEC_INC_CORRECT_COUNT_LSB: 0920
0x0924	STAT_FEC_INC_CORRECT_COUNT_MSB: 0924
0x0928	STAT_FEC_INC_CANT_CORRECT_COUNT_LSB: 0928
0x092C	STAT_FEC_INC_CANT_CORRECT_COUNT_MSB: 092C

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 reset.

A description of each signal is found in Port Descriptions.

GT_RESET_REG: 0000

Table 2-25: **GT_RESET_REG: 0000**

Bits	Default	Type	Signal
0	0	RW	ctl_gt_reset_all

RESET_REG: 0004

Table 2-26: **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 2-27: **MODE_REG: 0008**

Bits	Default	Type	Signal
30	1	RW	tick_reg_mode_sel
31	0	RW	ctl_local_loopback

CONFIGURATION_TX_REG1: 000C

Table 2-28: **CONFIGURATION_TX_REG1: 000C**

Bits	Default	Type	Signal
0	1	RW	ctl_tx_enable
1	1	RW	ctl_tx_fcs_ins_enable
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
13:10	12	RW	ctl_tx_ipg_value
14	0	RW	ctl_tx_test_pattern
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 2-29: **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
7	0	RW	ctl_rx_test_pattern
8	0	RW	ctl_rx_test_pattern_enable
9	0	RW	ctl_rx_data_pattern_select
10	-	-	Reserved
11	0	RW	ctl_rx_custom_preamble_enable

CONFIGURATION_RX_MTU: 0018

Table 2-30: **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

CONFIGURATION_VL_LENGTH_REG: 001C

Table 2-31: **CONFIGURATION_VL_LENGTH_REG: 001C**

Bits	Default	Type	Signal
15:0	20,479	RW	ctl_tx_vl_length_minus1
31:16	20,479	RW	c ctl_rx_vl_length_minus1

TICK_REG: 0020

Table 2-32: **TICK_REG: 0020**

Bits	Default	Type	Signal
0	0	WO	tick_reg

CONFIGURATION_REVISION_REG: 0024

Table 2-33: **CONFIGURATION_REVISION_REG: 0024**

Bits	Default	Type	Signal
7:0	1	RO	major_rev
15:8	2	RO	minor_rev
31:24	0	RO	patch_rev

CONFIGURATION_TX_TEST_PAT_SEED_A_LSB: 0028

Table 2-34: **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 2-35: **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 2-36: **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 2-37: **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 2-38: **CONFIGURATION_1588_REG: 0038**

Bits	Default	Type	Signal
0	0	RW	ctl_tx_ptp_1step_enable
2	0	RW	ctl_ptp_transpclk_mode
26:16	0	RW	ctl_tx_ptp_latency_adjust

CONFIGURATION_TX_FLOW_CONTROL_REG1: 0040

Table 2-39: **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 2-40: **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 2-41: **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 2-42: **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 2-43: **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 2-44: **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 2-45: **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 2-46: **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 2-47: **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 2-48: **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 2-49: **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 2-50: **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 2-51: **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 2-52: **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 2-53: **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 2-54: **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 2-55: **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 2-56: **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 2-57: **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

Table 2-58: **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 2-59: **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 2-60: **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 2-61: **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 2-62: **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 2-63: **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 2-64: **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 2-65: **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 2-66: **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 2-67: **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 2-68: **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 2-69: **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 2-70: **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 2-71: **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 2-72: **CONFIGURATION_RX_FLOW_CONTROL_SA_REG1_MSB: 00C4**

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

CONFIGURATION_RSFEC_REG: 00D0

Table 2-73: **CONFIGURATION_RSFEC_REG: 00D0**

Bits	Default	Type	Signal
0	0	RW	ctl_rsfec_enable
1	0	RW	ctl_rsfec_consortium_25g
2	0	RW	ctl_rx_rsfec_enable_indication
3	0	RW	ctl_rx_rsfec_enable_correction
5	0	RW	ctl_rsfec_ieee_error_indication_mode

CONFIGURATION_FEC_REG: 00D4

Table 2-74: **CONFIGURATION_FEC_REG: 00D4**

Bits	Default	Type	Signal
0	0	RW	ctl_fec_rx_enable
1	0	RW	ctl_fec_tx_enable
2	0	RW	ctl_fec_enable_error_to_pcs

CONFIGURATION_AN_CONTROL_REG1: 00E0

Table 2-75: **CONFIGURATION_AN_CONTROL_REG1: 00E0**

Bits	Default	Type	Signal
0	0	RW	ctl_autoneg_enable
1	1	RW	ctl_autoneg_bypass
9:2	0	RW	ctl_an_nonce_seed
10	0	RW	ctl_an_pseudo_sel
11	0	RW	ctl_restart_negotiation
12	0	RW	ctl_an_local_fault

CONFIGURATION_AN_CONTROL_REG2: 00E4

Table 2-76: **CONFIGURATION_AN_CONTROL_REG2: 00E4**

Bits	Default	Type	Signal
0	0	RW	ctl_an_pause
1	0	RW	ctl_an_asmdir
16	0	RW	ctl_an_fec_10g_request
17	0	RW	ctl_an_fec_ability_override
18	0	RW	ctl_an_cl91_fec_request
19	0	RW	ctl_an_cl91_fec_ability
20	0	RW	ctl_an_fec_25g_rs_request
21	0	RW	ctl_an_fec_25g_baser_request

CONFIGURATION_AN_ABILITY: 00F8

Table 2-77: **CONFIGURATION_AN_ABILITY: 00F8**

Bits	Default	Type	Signal
0	0	RW	ctl_an_ability_1000base_kx
1	0	RW	ctl_an_ability_10gbase_kx4
2	0	RW	ctl_an_ability_10gbase_kr
3	0	RW	ctl_an_ability_40gbase_kr4
4	0	RW	ctl_an_ability_40gbase_cr4
5	0	RW	ctl_an_ability_100gbase_cr10
6	0	RW	ctl_an_ability_100gbase_kp4
7	0	RW	ctl_an_ability_100gbase_kr4
8	0	RW	ctl_an_ability_100gbase_cr4
9	0	RW	ctl_an_ability_25gbase_krcr_s
10	0	RW	ctl_an_ability_25gbase_krcr
11	0	RW	ctl_an_ability_25gbase_kr1
12	0	RW	ctl_an_ability_25gbase_cr1
13	0	RW	ctl_an_ability_50gbase_kr2
14	0	RW	ctl_an_ability_50gbase_cr2

CONFIGURATION_LT_CONTROL_REG1: 0100

Table 2-78: **CONFIGURATION_LT_CONTROL_REG1: 0100**

Bits	Default	Type	Signal
0	0	RW	ctl_lt_training_enable
1	0	RW	ctl_lt_restart_training

CONFIGURATION_LT_TRAINED_REG: 0104

Table 2-79: **CONFIGURATION_LT_TRAINED_REG: 0104**

Bits	Default	Type	Signal
0	0	RW	ctl_lt_rx_trained

CONFIGURATION_LT_PRESET_REG: 0108

Table 2-80: **CONFIGURATION_LT_PRESET_REG: 0108**

Bits	Default	Type	Signal
0	0	RW	ctl_lt_preset_to_tx

CONFIGURATION_LT_INIT_REG: 010C

Table 2-81: **CONFIGURATION_LT_INIT_REG: 010C**

Bits	Default	Type	Signal
0	0	RW	ctl_lt_initialize_to_tx

CONFIGURATION_LT_SEED_REG0: 0110

Table 2-82: **CONFIGURATION_LT_SEED_REG0: 0110**

Bits	Default	Type	Signal
10:0	0	RW	ctl_lt_pseudo_seed0

CONFIGURATION_LT_COEFFICIENT_REG0: 0130

Table 2-83: **CONFIGURATION_LT_COEFFICIENT_REG0: 0130**

Bits	Default	Type	Signal
1:0	0	RW	ctl_lt_k_p1_to_tx0
3:2	0	RW	ctl_lt_k0_to_tx0
5:4	0	RW	ctl_lt_k_m1_to_tx0
7:6	0	RW	ctl_lt_stat_p1_to_tx0
9:8	0	RW	ctl_lt_stat0_to_tx0
11:10	0	RW	ctl_lt_stat_m1_to_tx0

Status Registers

Table 2-84 to Table 2-100 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 2-84: **STAT_TX_STATUS_REG1: 0400**

Bits	Default	Type	Signal
0	0	RO LH	stat_tx_local_fault

STAT_RX_STATUS_REG1: 0404

Table 2-85: **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

STAT_STATUS_REG1: 0408

Table 2-86: **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 2-87: **STAT_RX_BLOCK_LOCK_REG: 040C**

Bits	Default	Type	Signal
0	0	RO LL	stat_rx_block_lock

STAT_RX_RSFEC_STATUS_REG: 0444

Table 2-88: **STAT_RX_FEC_STATUS_REG: 0444**

Bits	Default	Type	Signal
0	0	RO LL	stat_rx_rsfec_lane_alignment_status
1	0	RO LL	stat_rx_rsfec_hi_ser

STAT_RX_FEC_STATUS_REG: 0448

Table 2-89: **STAT_RX_FEC_STATUS_REG: 0448**

Bits	Default	Type	Signal
0	0	RO LL	stat_fec_rx_lock
16	0	RO LL	stat_fec_lock_error

STAT_TX_FLOW_CONTROL_REG1: 0450

Table 2-90: **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 2-91: **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: 0458

Table 2-92: **STAT_AN_STATUS: 0458**

Bits	Default	Type	Signal
0	0	RO	stat_an_fec_enable
1	0	RO	stat_an_rs_fec_enable
2	0	RO	stat_an_autoneg_complete
3	0	RO	stat_an_parallel_detection_fault
4	0	RO	stat_an_tx_pause_enable
5	0	RO	stat_an_rx_pause_enable
6	0	RO LH	stat_an_lp_ability_valid
7	0	RO	stat_an_lp_autoneg_able
8	0	RO	stat_an_lp_pause
9	0	RO	stat_an_lp_asm_dir

Table 2-92: **STAT_AN_STATUS: 0458 *(Cont'd)***

Bits	Default	Type	Signal
10	0	RO	stat_an_lp_rf
11	0	RO	stat_an_lp_fec_10g_ability
12	0	RO	stat_an_lp_fec_10g_request
13	0	RO LH	stat_an_lp_extended_ability_valid
17:14	0	RO	stat_an_lp_ability_extended_fec
18	0	RO	stat_an_lp_fec_25g_rs_request
19	0	RO	stat_an_lp_fec_25g_baser_request

STAT_AN_ABILITY: 045C

Table 2-93: **STAT_AN_ABILITY: 045C**

Bits	Default	Type	Signal
0	0	RO	stat_an_lp_ability_1000base_kx
1	0	RO	stat_an_lp_ability_10gbase_kx4
2	0	RO	stat_an_lp_ability_10gbase_kr
3	0	RO	stat_an_lp_ability_40gbase_kr4
4	0	RO	stat_an_lp_ability_40gbase_cr4
5	0	RO	stat_an_lp_ability_100gbase_cr10
6	0	RO	stat_an_lp_ability_100gbase_kp4
7	0	RO	stat_an_lp_ability_100gbase_kr4
8	0	RO	stat_an_lp_ability_100gbase_cr4
9	0	RO	stat_an_lp_ability_25gbase_krcr_s
10	0	RO	stat_an_lp_ability_25gbase_krcr
11	0	RO	stat_an_lp_ability_25gbase_kr1
12	0	RO	stat_an_lp_ability_25gbase_cr1
13	0	RO	stat_an_lp_ability_50gbase_kr2
14	0	RO	stat_an_lp_ability_50gbase_cr2

STAT_AN_LINK_CTL: 0460

Table 2-94: **STAT_AN_LINK_CTL: 0460**

Bits	Default	Type	Signal
1:0	0	RO	stat_an_link_cntl_1000base_kx
3:2	0	RO	stat_an_link_cntl_10gbase_kx4
5:4	0	RO	stat_an_link_cntl_10gbase_kr
7:6	0	RO	stat_an_link_cntl_40gbase_kr4
9:8	0	RO	stat_an_link_cntl_40gbase_cr4
11:10	0	RO	stat_an_link_cntl_100gbase_cr10
13:12	0	RO	stat_an_link_cntl_100gbase_kp4
15:14	0	RO	stat_an_link_cntl_100gbase_kr4
17:16	0	RO	stat_an_link_cntl_100gbase_cr4
19:18	0	RO	stat_an_link_cntl_25gbase_krcr_s
21:20	0	RO	stat_an_link_cntl_25gbase_krcr
23:22	0	RO	stat_an_link_cntl_25gbase_kr1
25:24	0	RO	stat_an_link_cntl_25gbase_cr1
27:26	0	RO	stat_an_link_cntl_50gbase_kr2
29:28	0	RO	stat_an_link_cntl_50gbase_cr2

STAT_LT_STATUS_REG1: 0464

Table 2-95: **STAT_LT_STATUS_REG1: 0464**

Bits	Default	Type	Signal
0	0	RO	stat_lt_initialize_from_rx
16	0	RO	stat_lt_preset_from_rx

STAT_LT_STATUS_REG2: 0468

Table 2-96: **STAT_LT_STATUS_REG2: 0468**

Bits	Default	Type	Signal
0	0	RO	stat_lt_training
16	0	RO	stat_lt_frame_lock

STAT_LT_STATUS_REG3: 046C

Table 2-97: **STAT_LT_STATUS_REG3: 046C**

Bits	Default	Type	Signal
0	0	RO	stat_lt_signal_detect
16	0	RO	stat_lt_training_fail

STAT_LT_STATUS_REG4: 0470

Table 2-98: **STAT_LT_STATUS_REG4: 0470**

Bits	Default	Type	Signal
0	0	RO LH	stat_lt_rx_sof

STAT_LT_COEFFICIENT0_REG: 0474

Table 2-99: **STAT_LT_COEFFICIENT0_REG: 0474**

Bits	Default	Type	Signal
1:0	0	RO	stat_lt_k_p1_from_rx0
3:2	0	RO	stat_lt_k0_from_rx0
5:4	0	RO	stat_lt_k_m1_from_rx0
7:6	0	RO	stat_lt_stat_p1_from_rx0
9:8	0	RO	stat_lt_stat0_from_rx0
11:10	0	RO	stat_lt_stat_m1_from_rx0

STAT_RX_VALID_CTRL_CODE: 0494

Table 2-100: **STAT_RX_VALID_CTRL_CODE: 0494**

Bits	Default	Type	Signal
0	0	RO LH	stat_rx_valid_ctrl_code

Statistics Counters

Table 2-101 through Table 2-236 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.

STATUS_CYCLE_COUNT_LSB: 0500

Table 2-101: **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 2-102: **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 2-103: **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 2-104: **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 2-105: **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 2-106: **STAT_RX_BAD_CODE_MSB: 0664**

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

STAT_RX_RSFEC_CORRECTED_CW_INC_LSB: 0670

Table 2-107: **STAT_RX_RSFEC_CORRECTED_CW_INC_LSB: 0670**

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

STAT_RX_RSFEC_CORRECTED_CW_INC_MSB: 0674

Table 2-108: **STAT_RX_RSFEC_CORRECTED_CW_INC_MSB: 0674**

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

STAT_RX_RSFEC_UNCORRECTED_CW_INC_LSB: 0678

Table 2-109: **STAT_RX_RSFEC_UNCORRECTED_CW_INC_LSB: 0678**

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

STAT_RX_RSFEC_UNCORRECTED_CW_INC_MSB: 067C

Table 2-110: **STAT_RX_RSFEC_UNCORRECTED_CW_INC_MSB: 067C**

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

STAT_TX_FRAME_ERROR_LSB: 06A0

Table 2-111: **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 2-112: **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 2-113: **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 2-114: **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 2-115: **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 2-116: **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 2-117: **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 2-118: **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 2-119: **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 2-120: **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 2-121: **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 2-122: **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 2-123: **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 2-124: **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 2-125: **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 2-126: **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 2-127: **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 2-128: **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 2-129: **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 2-130: **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 2-131: **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 2-132: **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 2-133: **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 2-134: **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 2-135: **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 2-136: **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 2-137: **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 2-138: **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 2-139: **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 2-140: **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 2-141: **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 2-142: **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 2-143: **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 2-144: **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 2-145: **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 2-146: **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 2-147: **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 2-148: **STAT_TX_PACKET_SMALL_MSB: 078C**

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

STAT_TX_BAD_FCS_LSB: 07B8

Table 2-149: **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 2-150: **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 2-151: **STAT_TX_UNICAST_LSB: 07D0**

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

STAT_TX_UNICAST_MSB: 07D4

Table 2-152: **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 2-153: **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 2-154: **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 2-155: **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 2-156: **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 2-157: **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 2-158: **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 2-159: **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 2-160: **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 2-161: **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 2-162: **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 2-163: **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 2-164: **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 2-165: **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 2-166: **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 2-167: **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 2-168: **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 2-169: **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 2-170: **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 2-171: **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 2-172: **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 2-173: **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 2-174: **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 2-175: **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 2-176: **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 2-177: **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 2-178: **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 2-179: **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 2-180: **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 2-181: **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 2-182: **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 2-183: **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

Table 2-184: **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 2-185: **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 2-186: **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 2-187: **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 2-188: **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 2-189: **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 2-190: **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 2-191: **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 2-192: **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 2-193: **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 2-194: **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 2-195: **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 2-196: **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 2-197: **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 2-198: **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 2-199: **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 2-200: **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 2-201: **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 2-202: **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 2-203: **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 2-204: **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 2-205: **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 2-206: **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 2-207: **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 2-208: **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 2-209: **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 2-210: **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 2-211: **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 2-212: **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 2-213: **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 2-214: **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 2-215: **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 2-216: **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 2-217: **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 2-218: **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 2-219: **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 2-220: **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 2-221: **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 2-222: **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 2-223: **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 2-224: **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 2-225: **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 2-226: **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 2-227: **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 2-228: **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 2-229: **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 2-230: **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 2-231: **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 2-232: **STAT_RX_TEST_PATTERN_MISMATCH_MSB: 091C**

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

STAT_FEC_INC_CORRECT_COUNT_LSB: 0920

Table 2-233: **STAT_FEC_INC_CORRECT_COUNT_LSB: 0920**

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

STAT_FEC_INC_CORRECT_COUNT_MSB: 0924

Table 2-234: **STAT_FEC_INC_CORRECT_COUNT_MSB: 0924**

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

STAT_FEC_INC_CANT_CORRECT_COUNT_LSB: 0928

Table 2-235: **STAT_FEC_INC_CANT_CORRECT_COUNT_LSB: 0928**

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

STAT_FEC_INC_CANT_CORRECT_COUNT_MSB: 092C

Table 2-236: **STAT_FEC_INC_CANT_CORRECT_COUNT_MSB: 092C**

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

Designing with the Subsystem

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

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 Xilinx 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 Vivado® Design Suite with characteristics suited to your board.

Know the Degree of Difficulty

Xilinx high-speed Ethernet IP core (HSEC) 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 HSEC 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 HSEC subsystem 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 HSEC core to Xilinx 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

This section describes the clocking for all the 10G/25G configurations at the component support wrapper layer. There are three fundamentally different clocking architectures depending on the functionality and options:

- PCS/PMA Only Clocking
- 10G/25G MAC with PCS/PMA Clocking
- Low Latency 10G/25G MAC with PCS/PMA Clocking

Also described is Auto-Negotiation and Link Training Clocking.

PCS/PMA Only Clocking

The clocking architecture for the 10G/25G PCS is illustrated below. There are three clock domains in the datapath, as illustrated by the dashed lines in Figure 3-1.

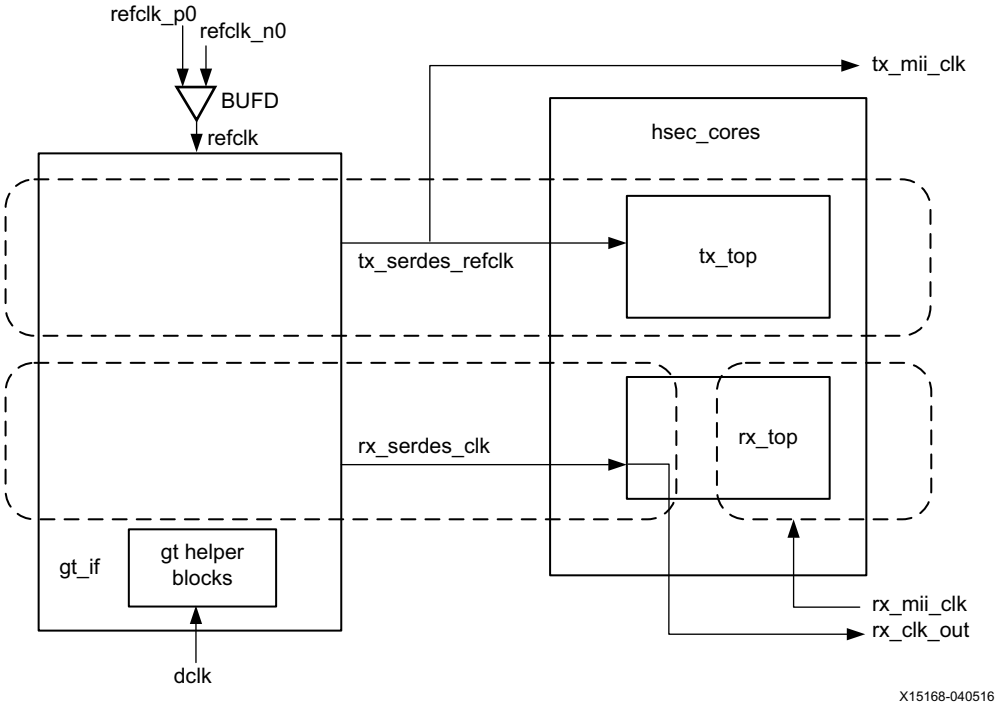

Figure 3-1: **PCS/PMA Clocking**

refclk_p0, refclk_n0, tx_serdes_refclk

The `refclk` differential pair is required to be an input to the FPGA. The example design includes a buffer to convert this clock to a single-ended signal `refclk`, which is used as the reference clock for the GT block. The `tx_serdes_refclk` is directly derived from `refclk`. Note that `refclk` must be chosen so that the `tx_mii_clk` meets the requirements of 802.3, which is within 100 ppm of 390.625 MHz for 25G and 156.25 MHz for 10G.

tx_mii_clk

The `tx_mii_clk` is an output which is the same as the `tx_serdes_refclk`. The entire TX path is driven by this clock. You must synchronize the TX path `mii` bus to this clock output. All TX control and status signals are referenced to this clock.

rx_serdes_clk

The `rx_serdes_clk` is derived from the incoming data stream within the GT block. The incoming data stream is processed by the RX core in this clock domain.

rx_clk_out

The `rx_clk_out` output signal is presented as a reference for the RX control and status signals processed by the RX core. It is the same frequency as the `rx_serdes_clk`.

rx_mii_clk

The `rx_mii_clk` input is required to be synchronized to the RX XXVGMII data bus. This clock and the RX XXVGMII bus must be within 100 ppm of the required frequency, which is 390.625 MHz for 25G and 156. 25 MHz for 10G.

dclk

The `dclk` signal must be a convenient stable clock. It is used as a reference frequency for the GT helper blocks which initiate the GT itself. In the example design, a typical value is 75 MHz, which is readily derived from the 300 MHz clock available on the VCU107 evaluation board. Note that the actual frequency must be known to the GT helper blocks for proper operation.

10G/25G MAC with PCS/PMA Clocking

The clocking architecture for the 10/25G MAC with PCS/PMA clocking is illustrated below. This version of the subsystem includes FIFOs in the RX and TX. There are three clock domains in the data path, as illustrated by the dashed lines in Figure 3-2.

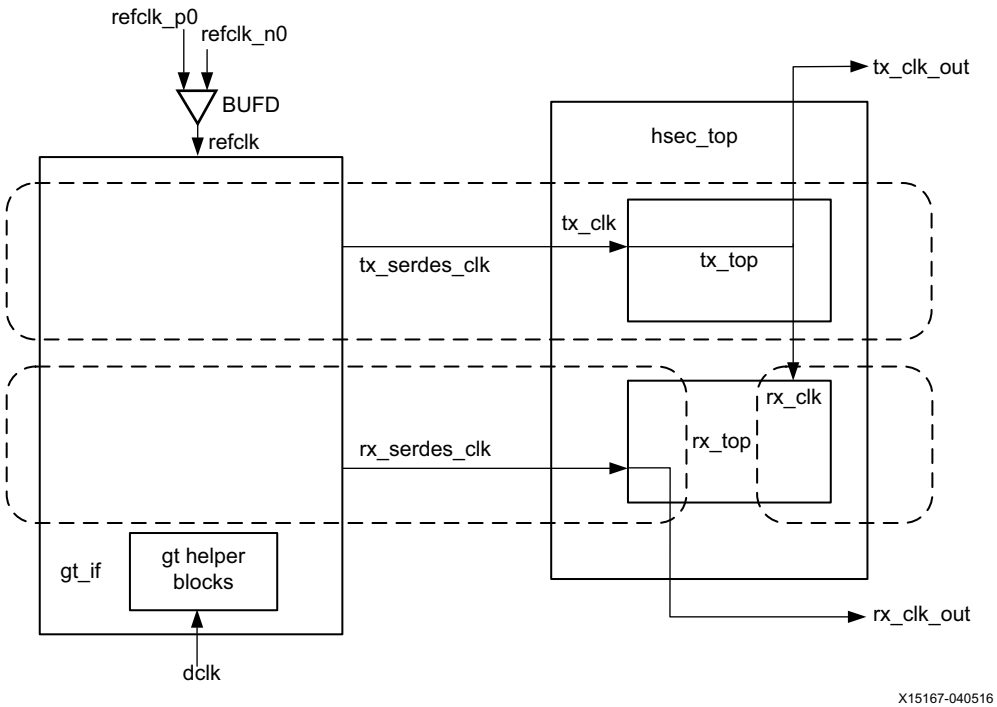

Figure 3-2: **10G/25G MAC with PCS/PMA Clocking**

refclk_p0, refclk_n0, tx_serdes_refclk

The `refclk` differential pair is required to be an input to the FPGA. The example design includes a buffer to convert this clock to a single-ended signal `refclk`, which is used as the reference clock for the GT block. The `tx_serdes_refclk` is directly derived from `refclk`. Note that `refclk` must be chosen so that the `tx_serdes_refclk` meets the requirements of 802.3, which is within 100 ppm of 390.625 MHz for 25G and 156.25 MHz for 10G.

tx_clk_out

This clock is used for clocking data into the TX AXI4-Stream Interface and it is also the reference clock for the TX control and status signals. It is the same frequency as `tx_serdes_refclk`.

rx_clk_out

The `rx_clk_out` output signal is presented as a reference for the RX control and status signals processed by the RX core. It is the same frequency as the `rx_serdes_clk`.

rx_clk

The `rx_clk` input to the RX core is not presented in the example design. Instead, it is connected to the `tx_clk` which also drives the TX core. When connected in this manner, the RX AXI4-Stream Interface and the TX AXI4-Stream Interface are on the same clock domain, which in most cases is the preferred mode of operation for the system side datapath. If desired, you can disconnect the `rx_clk` input from the `rx_top` module and drive the RX AXI4-Stream Interface with a different clock than the TX AXI4-Stream Interface. In this case, the frequency of the `rx_clk` must be equal to or greater than the `tx_clk`.

dclk

The `dclk` signal must be a convenient stable clock. It is used as a reference frequency for the GT helper blocks which initiate the GT itself. In the example design, a typical value is 75 MHz, which is readily derived from the 300 MHz clock available on the VCU107 evaluation board.

Note: The actual frequency must be known to the GT helper blocks for proper operation.

Low Latency 10G/25G MAC with PCS/PMA Clocking

The clocking architecture for the Low Latency 10/25G MAC with PCS/PMA clocking is illustrated in Figure 3-3. Low latency is achieved by omitting the RX and TX FIFOs, which results in different clocking arrangement. There are two clock domains in the datapath, as illustrated by the dashed lines in Figure 3-3.

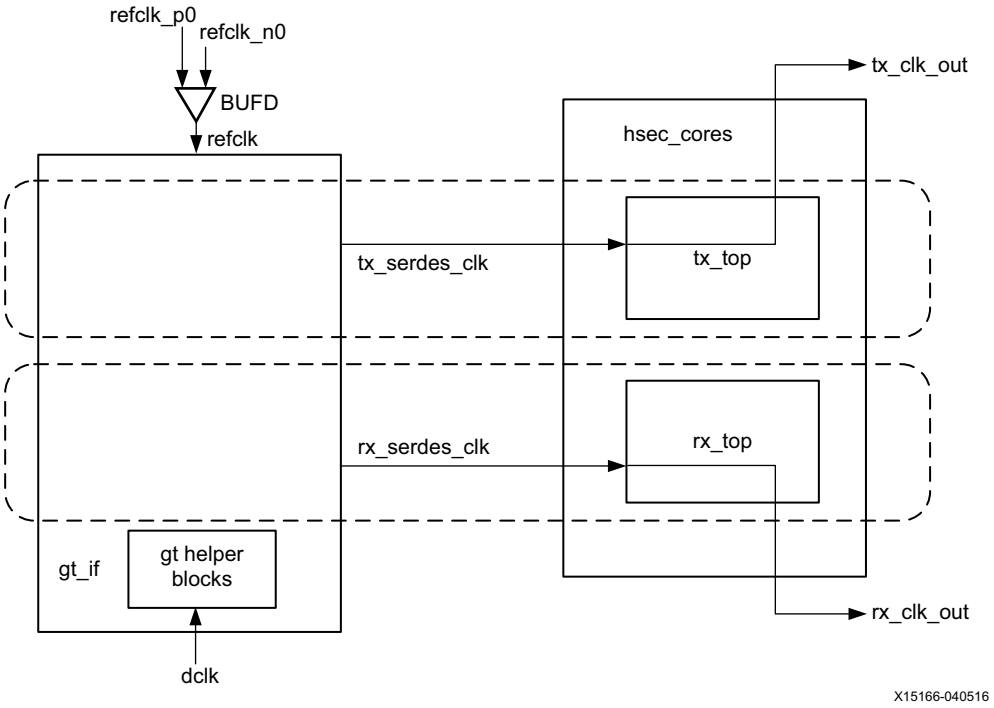

Figure 3-3: **Low Latency 10G/25G MAC with PCS/PMA Clocking**

refclk_p0, refclk_n0, tx_serdes_refclk

The `refclk` differential pair is required to be an input to the FPGA. The example design includes a buffer to convert this clock to a single-ended signal `refclk`, which is used as the reference clock for the GT block. The `tx_serdes_refclk` is directly derived from `refclk`. Note that `refclk` must be chosen so that the `tx_serdes_refclk` meets the requirements of 802.3, which is within 100 ppm of 390.625 MHz for 25G, and 156.25 MHz for 10G.

tx_clk_out

This clock is used for clocking data into the TX AXI4-Stream Interface and it is also the reference clock for the TX control and status signals. It is the same frequency as `tx_serdes_refclk`. Because there is no TX FIFO, you must respond immediately to the `tx_axis_tready` signal.

rx_clk_out

The `rx_clk_out` output signal is presented as a reference for the RX control and status signals processed by the RX core. It is the same frequency as the `rx_serdes_clk`. Because there is no RX FIFO, this is also the clock which drives the RX AXI4-Stream Interface. In this arrangement, `rx_clk_out` and `tx_clk_out` are different frequencies and have no defined phase relationship to each other.

dclk

The `dclk` signal must be a convenient stable clock. It is used as a reference frequency for the GT helper blocks which initiate the GT itself. In the example design, a typical value is 75 MHz, which is readily derived from the 300 MHz clock available on the VCU107 evaluation board. Note that the actual frequency must be known to the GT helper blocks for proper operation.

Auto-Negotiation and Link Training Clocking

The clocking architecture for the Auto-Negotiation and Link Training blocks are illustrated in Figure 3-4. Note that these blocks are not included unless the 25GBASE-KR or 25GBASE-CR feature is selected.

The Auto-Negotiation and Link Training blocks function independently from the MAC and PCS, and therefore they are on different clock domains.

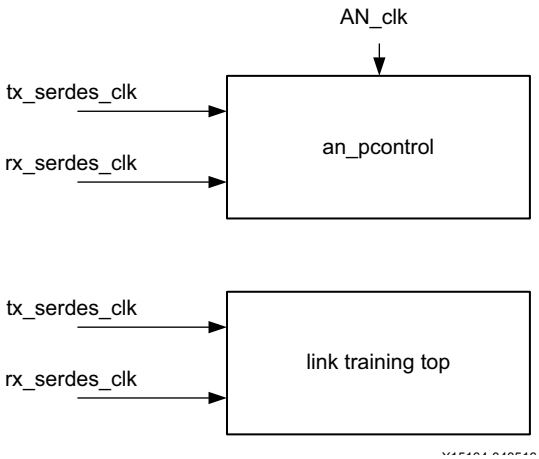

Figure 3-4: **Auto-Negotiation and Link Training Clocking**

tx_serdes_clk

The `tx_serdes_clk` drives the TX line side logic for the Auto-Negotiation and Link Training. The DME frame is generated on this clock domain.

rx_serdes_clk

The `rx_serdes_clk` drives the RX line side logic for the Auto-Negotiation and Link Training.

AN_clk

The AN_clk drives the Auto-Negotiation state machine. All ability signals are on this clock domain. The AN_clk can be any convenient frequency. In the example design, AN_clk is connected to the dclk input, which has a typical frequency of 75 MHz. The AN_clk frequency must be known to the Auto-Negotiation state machine because it is the reference for all timers.

Resets

Figure 3-5 shows the reset structure for the 10G/25G Ethernet MAC with PCS/PMA as implemented at the component support wrapper layer. Clocks are not shown for clarity.

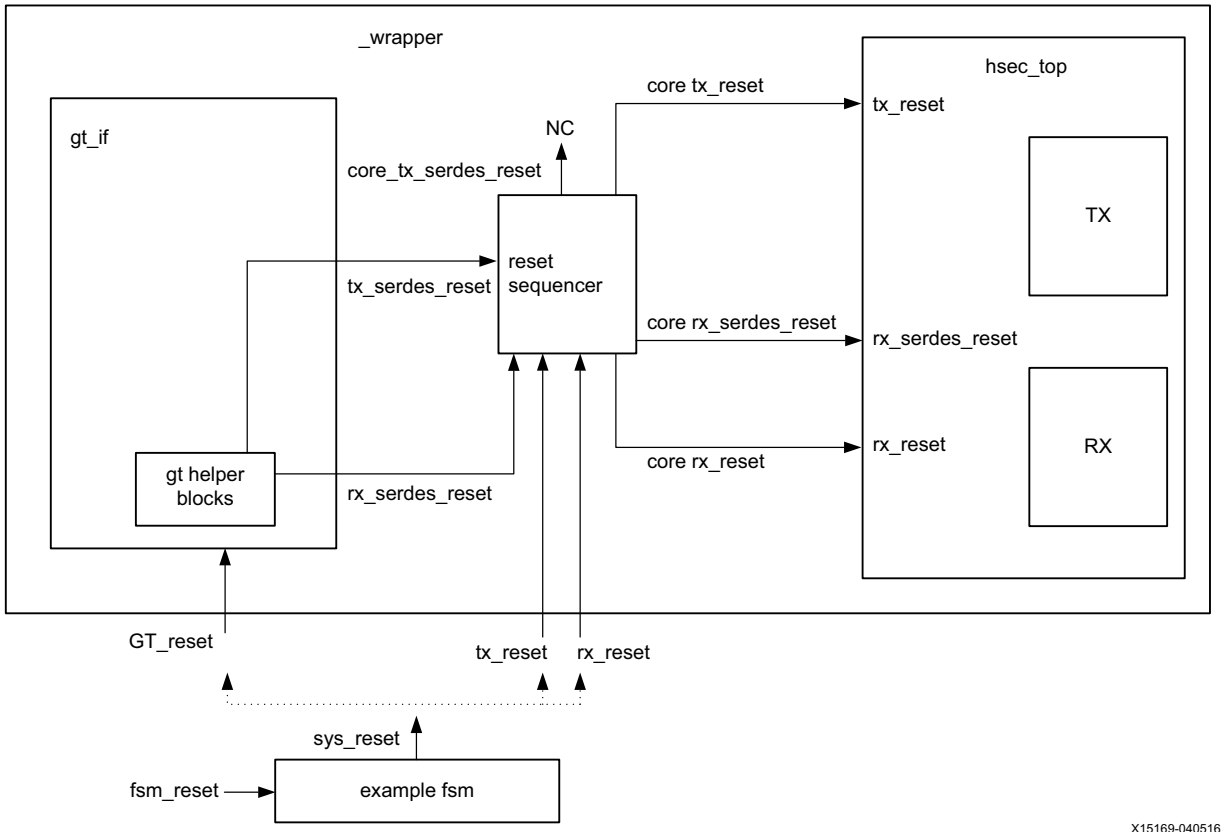

Figure 3-5: **Reset Structure**

Component Support Layer Resets

In the example design, a single reset is used to reset the entire wrapper layer. Using the external stimulus fsm_reset, the example_fsm block issues the signal sys_reset which is connected to the three _wrapper resets. Therefore, the example design demonstrates that all three wrapper resets can be released simultaneously and correct operation follows.

Wrapper Resets

The _wrapper layer of the hierarchy is assumed to be what you instantiate in your own design. There are three resets to be handled as follows:

- GT_reset
- tx_reset
- rx_reset

You do not need to be concerned with timing the reset signals; this is taken care of by the reset_sequencer block.

GT_reset

The `GT_reset` is the asynchronous active-High reset input to the GT. You do not need to be concerned with the internal resets of the GT because this is taken care of by the GT helper blocks.

tx_reset

The `tx_reset` is the asynchronous active-High reset for the TX path logic of the 10G/25G Ethernet IP core. While it is connected to the GT reset in the example design, this reset can be asserted at any time to reset the TX path independently without disturbing the RX path.

rx_reset

The `rx_reset` is the asynchronous active-High reset for the RX path logic of the 10G/25G Ethernet IP core. While it is connected to the GT reset in the example design, this reset can be asserted at any time to reset the RX path independently without disturbing the TX path.

LogiCORE Example Design Clocking and Resets

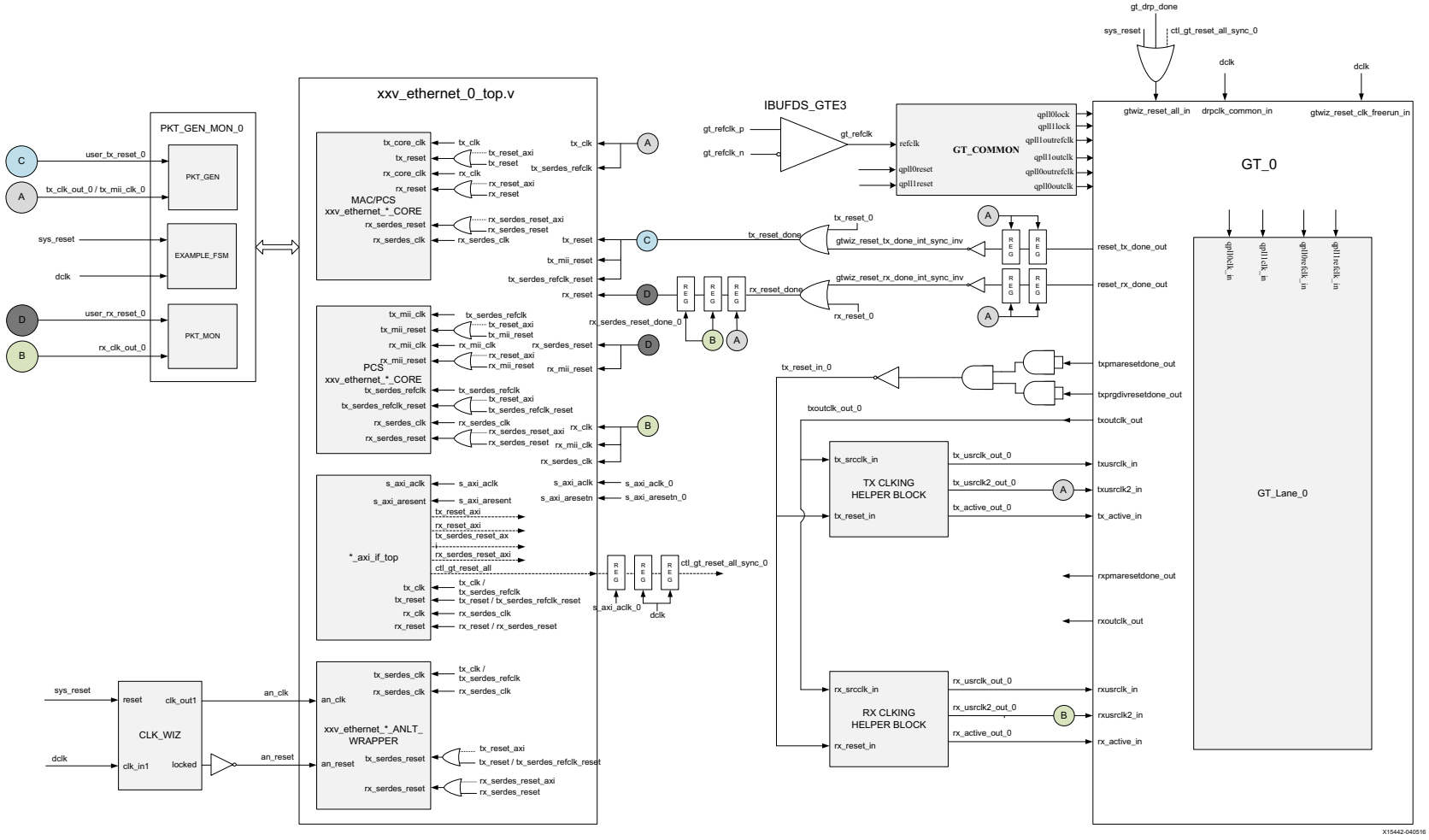

Figure 3-6: **Detailed Diagram of Single Core - Synchronous Clock Mode**

Figure 3-6 through Figure 3-9, illustrate the clocking and reset structure when you implement the Example Design using the Vivado tools.

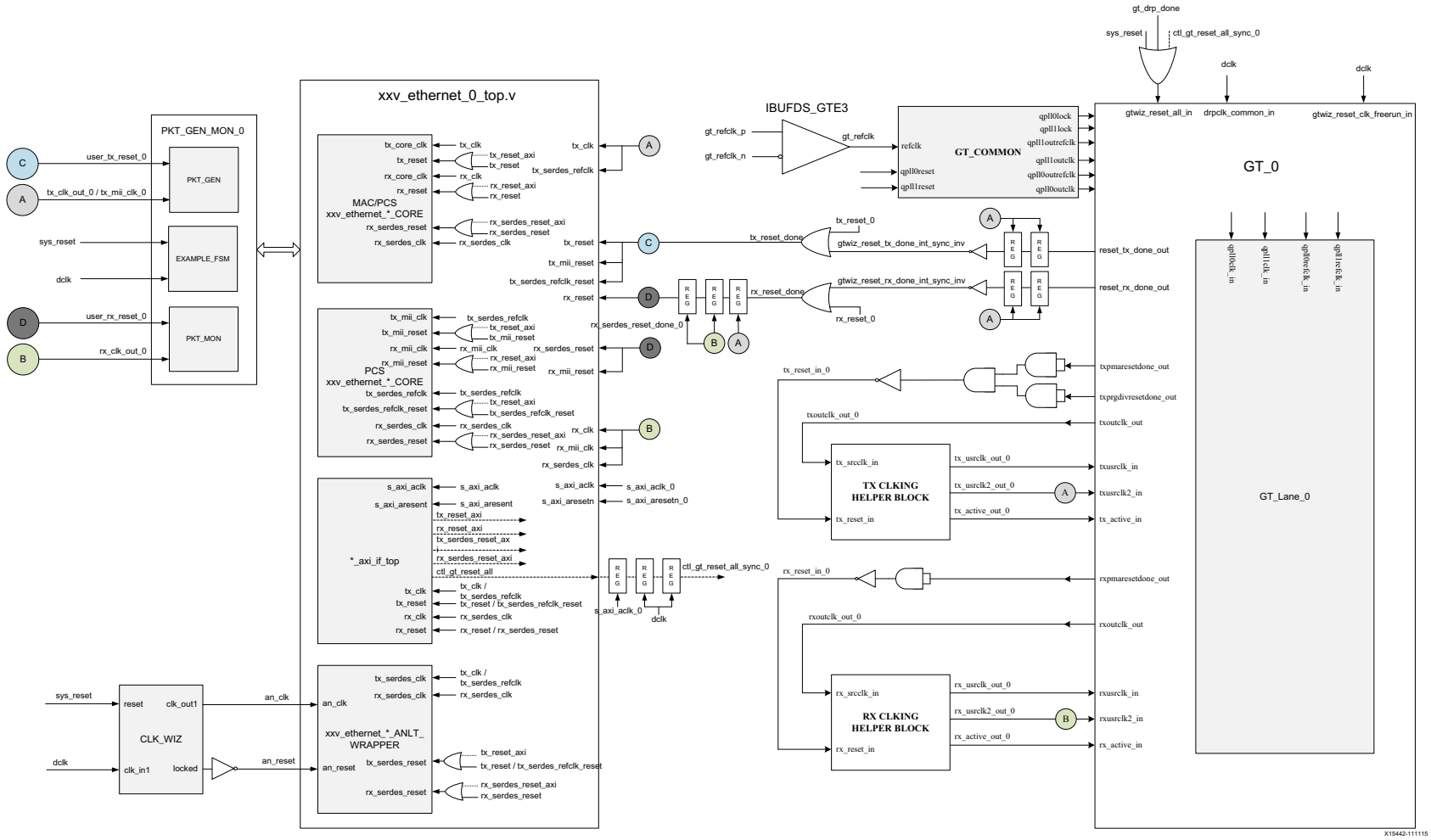

Figure 3-7: **Detailed Diagram of Single Core - Asynchronous Clock Mode**

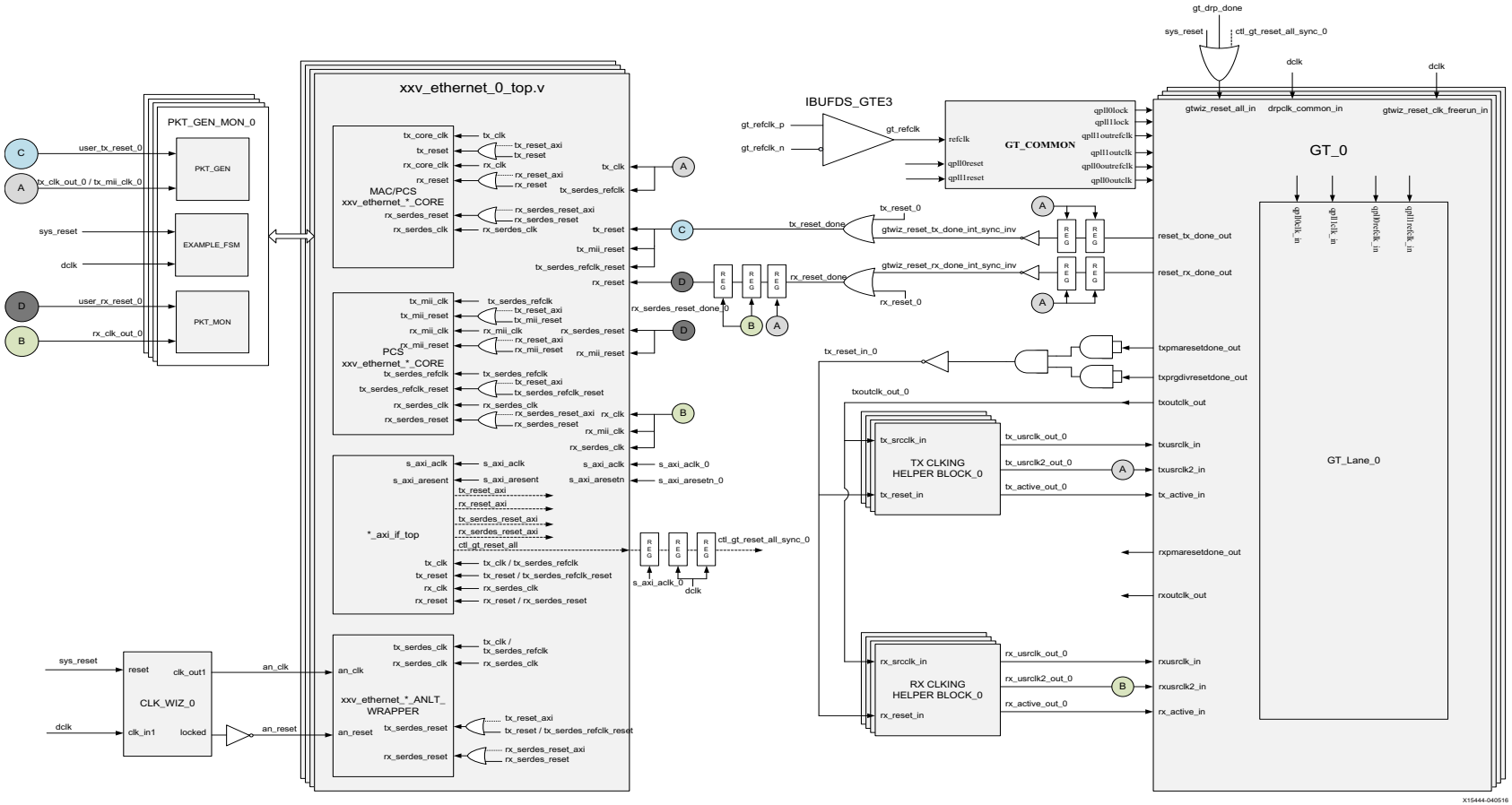

Figure 3-8: **Detailed Diagram of Multiple Cores - Synchronous Clock Mode**

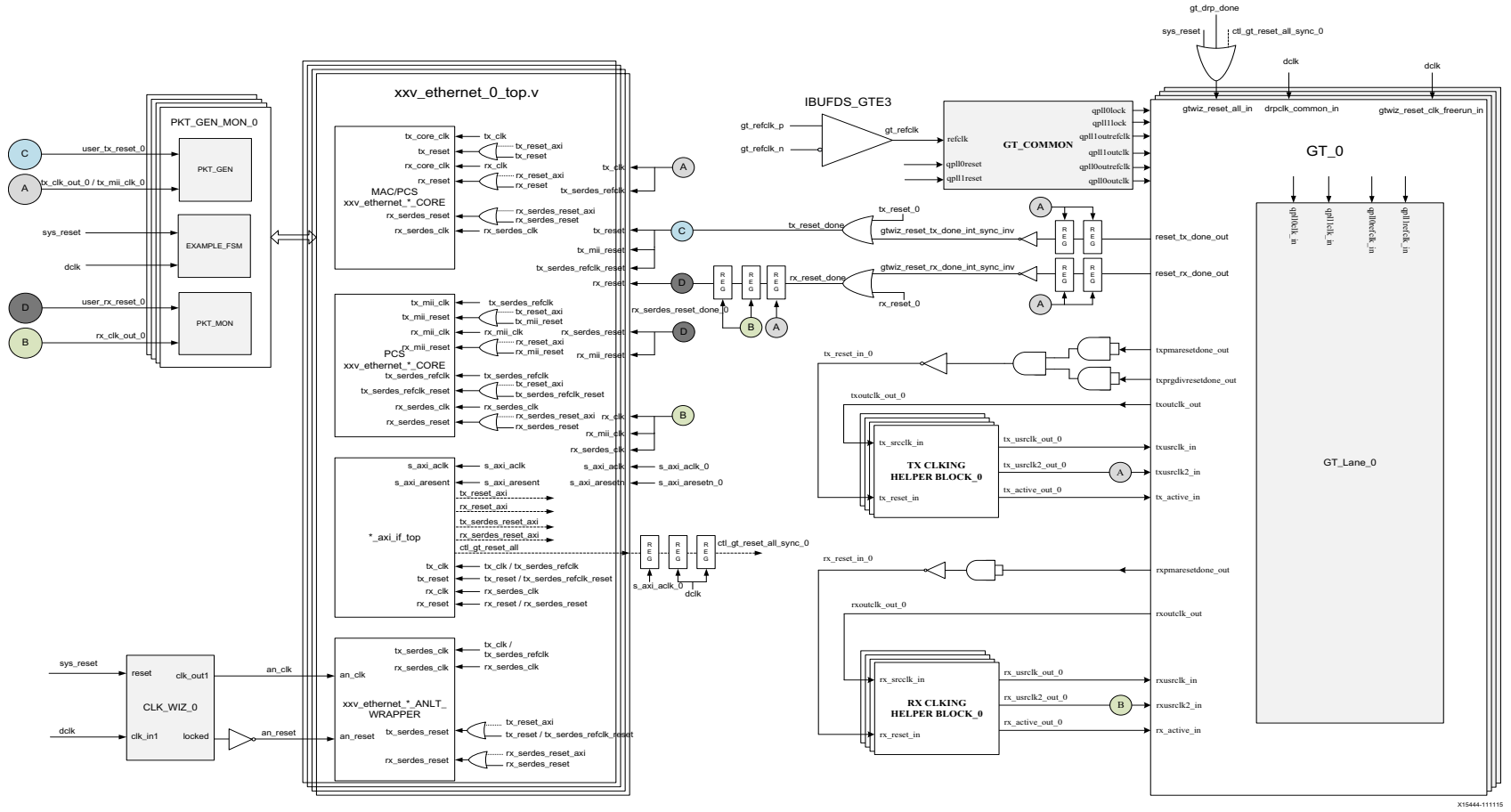

Figure 3-9: **Detailed Diagram of Multiple Cores - Asynchronous Clock Mode**

Support for IEEE Standard 1588 Version 2

Overview

This section details the packet timestamping function of the 10G/25G Ethernet subsystem when the MAC layer is included. The timestamping option must be specified at the time of generating the subsystem from the IP catalog or ordering the IP Core asynchronously. This feature provides one-step and two-step IEEE 1588 functionality.

Ethernet frames are timestamped at both ingress and egress. The option can be used for implementing all kinds of IEEE 1588 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 1588 Precision Time Protocol (PTP).

IEEE 1588 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 should 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 which are selected during subsystem generation.

- 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 should count from 0 through the full range up to 264 -1 before wrapping around.

Egress

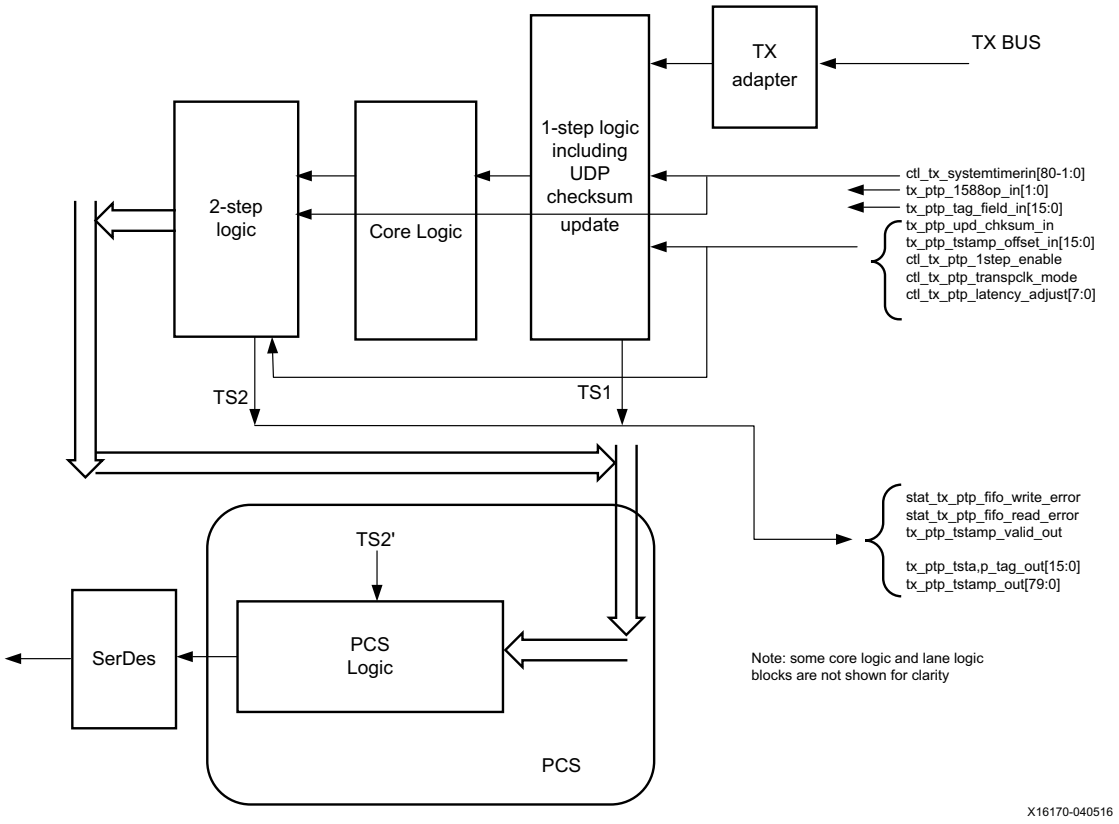

Figure 3-10: **Egress**

As seen in the preceding figure, timestamping logic exists in two locations depending on whether 1-step or 2-step operation is desired. 1-step operation requires the user datagram protocol (UDP) checksum and FCS updates and therefore the FCS core logic is used.

The TS references are defined as follows:

- TS1: The output timestamp signal when a 1-step operation is selected.
- TS2: The output timestamp signal when a 2-step operation is selected.
- TS2': The plane to which both timestamps are corrected.

TS2 always has a correction applied so that it is referenced to the TS2' plane. TS1 might or might not have the TS2' correction applied, depending on the value of the signal `ctl_tx_ptp_latency_adjust[10:0]`.

Based on rate and clock mode (Ordinary or Transparent), the suggested default values of the `ctl_tx_ptp_latency_adjust[10:0]` signal are as follows:

- 25G – Ordinary Clock = 395
- 25G – Transparent Clock = 471
- 10G – Ordinary Clock = 970
- 10G – Transparent Clock = 1177

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, 1-step, or 2-step) and also indicates, for 1-step frames, whether there is a UDP checksum field to update.

If using the ToD format, then for both 1-step and 2-step operation, the full captured 80-bit ToD timestamp is returned to the client logic using the additional ports defined in Table 3-1.

If using the Correction Field format, then for both 1-step and 2-step operation, the full captured 64-bit timestamp is returned to the client logic using the additional ports defined in Table 3-1 (with the upper bits of data set to zero as defined in the table).

If using the ToD format, then for 1-step operation, the full captured 80-bit ToD timestamp is inserted into the frame. If using the Correction Field format, then for 1-step operation, the captured 64-bit timestamp is summed with the existing Correction Field contained within the frame and the summed result is overwritten into the original Correction Field of the frame. Supported frame types for 1-step timestamping are:

- Raw Ethernet
- UDP/IPv4
- UDP/IPv6

For 1-step UDP frame types, the UDP checksum is updated in accordance with IETF RFC 1624. For all 1-step frames, the Ethernet Frame Check Sequence (FCS) field is calculated after all frame modifications have been completed. For 2-step transmit operation, all Precision Time Protocol (PTP) frame types are supported.

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 should be taken.
- Two-step operation is required and a tag value (user-sequence ID) is provided as part of the command field; the frame should 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.
- 1-step operation is required
  - For the ToD timer and timestamp format a timestamp offset value is provided as part of the command port; the frame should be timestamped, and the timestamp should be inserted into the frame at the provided offset (number of bytes) into the frame.
  - For the Correction Field format, a Correction Field offset value is provided as part of the command port; the frame should be timestamped, and the captured 64-bit Timestamp is summed with the existing Correction Field contained within the frame and the summed result is overwritten into original Correction Field of the frame.

    For 1-step operation, following the frame modification, the cyclic redundancy check (CRC) value of the frame should also be updated/recalculated. For UDP IPv4 and IPv6 PTP formatted frames, the checksum value in the header of the frame needs to be updated/recalculated.
- For 1-step UDP frame types, the UDP checksum is updated in accordance with IETF RFC 1624.
  - If using the ToD format, in order for this update function to work correctly, the original checksum value for the frame sent for transmission should be calculated using a zero value for the timestamp data. This particular restriction does not apply when using the Correction Field format.
  - If using the Correction Field format, a different restriction does apply; the separation between the UDP Checksum field and the Correction Field within the 1588 PTP frame header is a fixed interval of bytes, supporting the 1588 PTP frame definition. This is a requirement to minimize the latency through the MAC since both the checksum and the correction field must both be fully contained in the MAC pipeline in order for the checksum to be correctly updated. This particular restriction does not apply to the ToD format because the original timestamp data is calculated as a zero value; consequently the checksum and timestamp position can be independently located within the frame.

Ingress

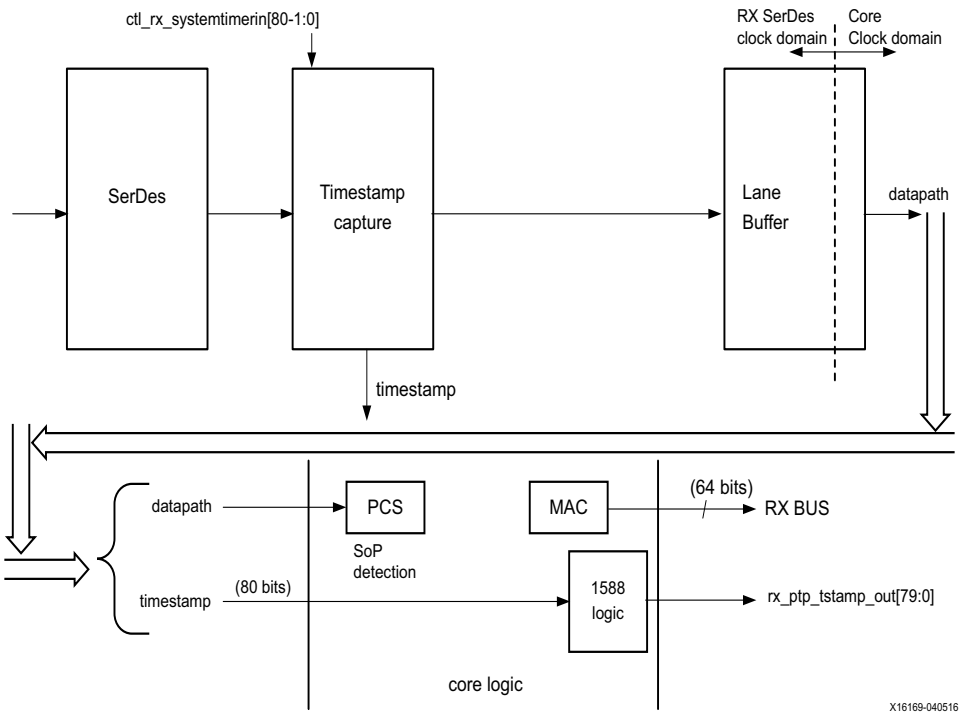

Figure 3-11: **Ingress**

The ingress logic does not parse the ingress packets to search for 1588 (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 an Start Of Packet (SOP). These 80-bit timestamps are output on the system side. The timestamp is valid during the SoP cycle and when ena_out = 1.

Port Descriptions

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

Table 3-1: **1588v2 Port List and Descriptions**

Local Fault Indication	Direction	Description	Clock Domain
IEEE 1588 Interface – TX Path			
ctl_tx_systemtimerin[80-1:0]	Input	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.	tx_serdes_clk
tx_ptp_tstamp_valid_out	Output	This bit indicates that a valid timestamp is being presented on the TX system interface.	tx_clk_out
tx_ptp_tstamp_tag_out[15:0]	Output	Tag output corresponding to tx_ptp_tag_field_in[15:0]	tx_clk_out
tx_ptp_tstamp_out[80-1:0]	Output	Timestamp for the transmitted packet SOP corresponding to the time at which it passed the capture plane. Time format same as timer input.	tx_clk_out
tx_ptp_1588op_in[1:0]	Input	The signal should be valid on the first cycle of the packet. 2'b00 – No operation: no timestamp will be taken and the frame will not be modified. 2'b01 – 1-step: a timestamp should be taken and inserted into the frame. 2'b10 – 2-step: a timestamp should be taken and returned to the client using the additional ports of 2-step operation. The frame itself will not be modified. 2'b11 – Reserved: act as No operation.	tx_clk_out
ctl_tx_ptp_1step_enable	Input	When set to 1, this bit enables 1-step operation.	tx_clk_out
ctl_ptp_transpclk_mode	Input	When set to 1, this input places the timestamping logic into transparent clock mode. In this mode, the system timer input is interpreted as a correction value. The TX will add the correction value to the TX timestamp according to the process defined in IEEE 1588v2. The sign bit of the correction value is assumed to be 0 (positive time). It is expected that the corresponding incoming PTP packet correction field has already been adjusted with the proper RX timestamp.	tx_clk_out

Table 3-1: **1588v2 Port List and Descriptions *(Cont'd)***

Local Fault Indication	Direction	Description	Clock Domain
tx_ptp_tag_field_in[15:0]	Input	The usage of this field is dependent on the 1588 operation. The signal should be valid on the first cycle of the packet. •For No operation, this field will be ignored. •For 1-step and 2-step this field is a tag field. This tag value will be returned to the client with the timestamp for the current frame using the additional ports of 2-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.	tx_clk_out
ctl_tx_ptp_latency_adjust[10:0]	Input	This bus can be used to adjust the 1-step TX timestamp with respect to the 2-step timestamp. The units of bits [10:3] are nanoseconds and bits [2:0] are fractional nanoseconds.	tx_clk_out
stat_tx_ptp_fifo_write_error	Output	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.	tx_clk_out
stat_tx_ptp_fifo_read_error	Output	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_clk_out
IEEE 1588 Interface – RX Path			
ctl_rx_systemtimerin[80-1:0]	Input	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_serdes_clk
rx_ptp_tstamp_out[80-1:0]	Output	Timestamp for the received packet SOP corresponding to the time at which it passed the capture plane. Note that this signal will be valid on the first cycle of the packet.	rx_clk_out

IEEE 1588 PTPV2 Functional Description

The IEEE 1588 feature of the HSEC provides accurate timestamping of Ethernet frames at the hardware level for both the ingress and egress directions.

Timestamps are captured according to the input clock source above. However, it is required that this time source be in the same clock domain as the SerDes. This might require re-timing by an external circuit provided by the user.

All ingress frames receive a timestamp. It is up to you to interpret the received frames and determine whether a particular frame contains PTP information (by means of its Ethertype) and if the timestamp needs to be retained or discarded.

Egress frames are timestamped if they are tagged as PTP frames. The timestamps of egress frames are matched to their user-supplied tags.

Timestamps for incoming frames are presented at the user interface during the same clock cycle as the start of packet. You can then append the timestamp to the packet as required.

By definition, a timestamp is captured coincident with the passing of the SOP through the capture plane within the HSEC. This is illustrated in the following schematic diagrams:

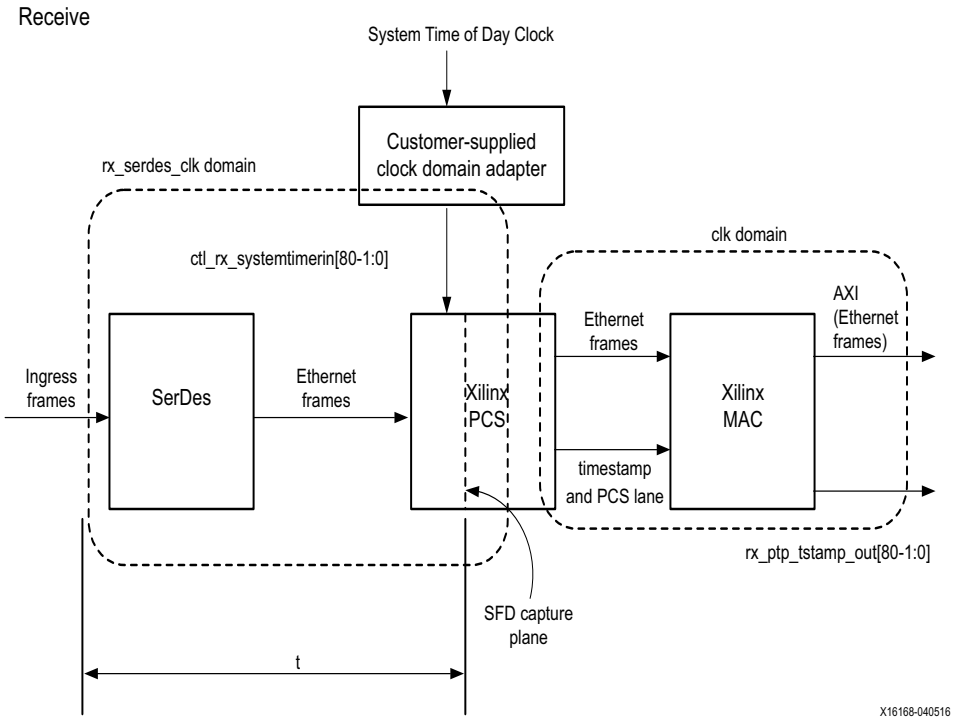

Figure 3-12: **Receive**

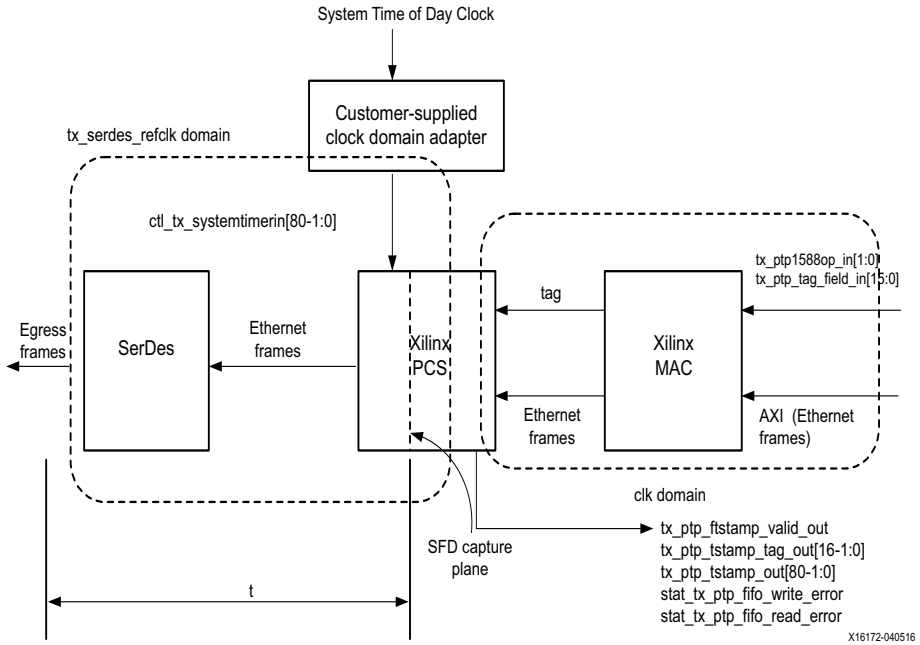

Figure 3-13: **Transmit**

Performance

The following table shows the accuracy of the timestamp capture for a typical 10G/25G Ethernet subsystem. Refer to the preceding figures for the definition of timing parameters.

The following table presents typical test results for "t". The values include the delay through a SerDes. Contact Xilinx for more information for more information and the latest SerDes data.

Table 3-2: **Accuracy of the Timestamp Capture**

Ethernet Port	clk (MHz)	t min (nsec)	t max (nsec)	SerDes phase FIFO variation (nsec)	time of day clock crossing jitter (nsec)
25 GE Ingress	390.625	38	39	+/- 1	+/- 2.56
25 GE Egress	390.625	30	31	+/- 1	+/- 2.56

Note that the SerDes clock is assumed to be 390.625 MHz.

In a typical application, the difference between the ingress and egress capture times is important for determining absolute time. The PTP algorithm can use asymmetry information to improve accuracy.

The 1588v2 feature requires that all clock frequencies be known in order to make internal calculations. The clock frequencies should be specified at the time the PTP IP core is ordered in order for the timestamp correction to work properly.

In a typical application, the PTP algorithm (or servo, not part of this IP) will remove jitter over the course of time (many packet samples). It is advantageous for the jitter to be as small as possible in order to minimize the convergence time as well as minimizing slave clock drift.

RS-FEC Support

Overview

This section describes the optional RS-FEC function of the 10G/25G Ethernet subsystem. The RS-FEC option must be specified at the time of generating the subsystem from the IP catalog or ordering the IP Core asynchronously.

The RS-FEC block is positioned between the PCS and PMA as illustrated below.

With reference to the following diagram, the clocks and resets of the RS-FEC core are equivalent to the transceiver signals, with the transceiver resets being active-High.

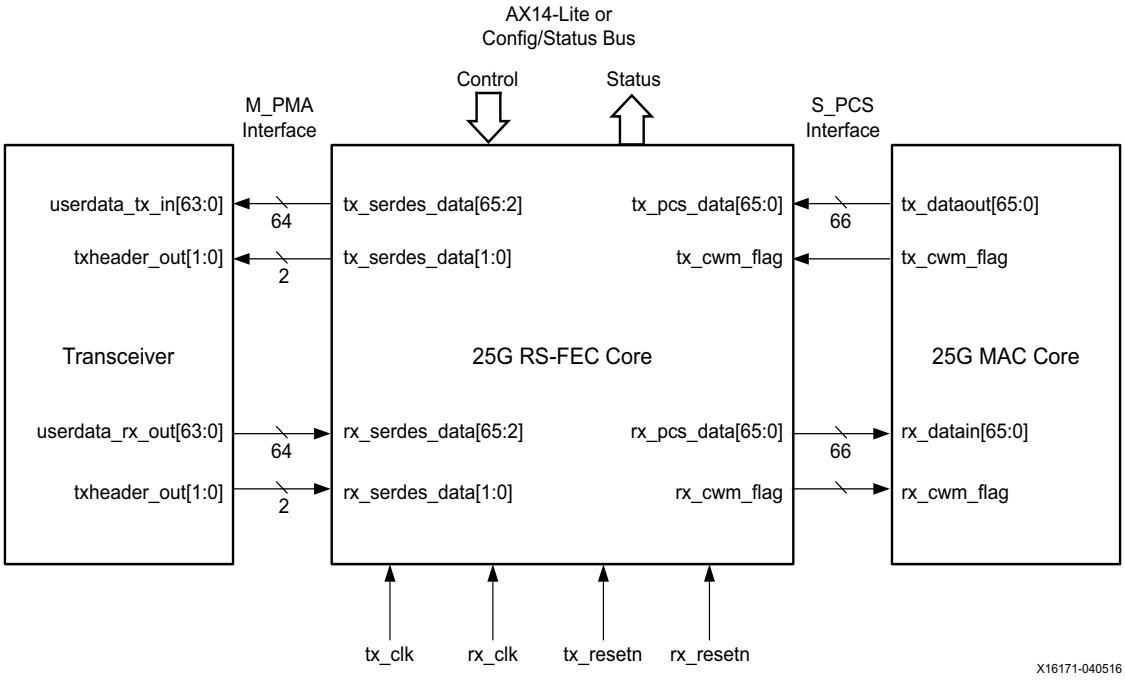

Figure 3-14: **RS-FEC Block Diagram**

The internal details of the RS-FEC are beyond the scope of this document. Refer to IEEE 802.3 Clause 108 and Schedule 3 of the 25G Ethernet Consortium [Ref 1].

Further information can also be found in the *25G IEEE 802.3by Reed-Solomon Forward Error Correction LogiCORE IP Product Guide* (PG217) [Ref 11].

Port Descriptions

Table 3-3: **RS-FEC Port List and Descriptions**

Port	Direction	Description	Clock Domain
RS-FEC Control Signals			
ctl_rsfec_ieee_error_indication_mode	Input	• 1: Core conforms to the IEEE RS-FEC specification. • 0: If FEC_bypass_correction_enable and FEC_bypass_indication_enable are asserted, the RS decoder is bypassed.	
ctl_rsfec_consortium_25g	Input	Switches between IEEE Clause 108 and 25G Ethernet Consortium mode. • 1 = 25G Consortium specification mode; • 0 = IEEE 802.3by mode Note that some variants of the 10G/25G Subsystem can have individual RX and TX consortium signals.	
ctl_rsfec_enable	Input	Enable RS-FEC function. Note that some variants of the 10G/25G Subsystem can have individual RX and TX enable signals.	
ctl_rx_rsfec_enable_correction	Input	The setting on this bit takes effect after rx_resetn has been asserted Low (~rx_serdes_reset). Equivalent to MDIO register 1.200.0 • 0: Decoder performs error detection without error correction (see IEEE 802.3802.3by section 91.5.3.3). • 1: the decoder also performs error correction.	
ctl_rx_rsfec_enable_indication	Input	The setting on this bit takes effect after rx_resetn has been asserted Low (~rx_serdes_reset). Equivalent to MDIO register 1.200.1 • 0: Bypass the error indication function (see IEEE Std 802.3by section 91.5.3.3). • 1: Decoder indicates errors to the PCS sublayer,	
ctl_rx_vl_length_minus1[15:0]	Input	Normally set to 20,479 (4FFF hex). The normal value is equivalent to (16,383 x 5 -4) = 81,916.	
ctl_rx_vl_marker_id0[63:0]	Input	Equivalent to the RX PCS lane 0 alignment marker defined in IEEE 802.3 Clause 82 for 40 G Ethernet.	
ctl_rx_vl_marker_id1[63:0]	Input	Equivalent to the PCS lane 1 alignment marker.	

Table 3-3: **RS-FEC Port List and Descriptions *(Cont'd)***

Port	Direction	Description	Clock Domain
ctl_rx_vl_marker_id2[63:0]	Input	Equivalent to the PCS lane 2 alignment marker.	
ctl_rx_vl_marker_id3[63:0]	Input	Equivalent to the PCS lane 3 alignment marker.	
ctl_tx_vl_length_minus1[15:0]	Input	Normally set to 20479 (decimal). The normal value is equivalent to (16,383 x 5 -4) = 81,916.	
ctl_tx_vl_marker_id0[63:0]	Input	Equivalent to the TX PCS lane 0 alignment marker defined in IEEE 802.3 Clause 82 for 40 G Ethernet.	
ctl_tx_vl_marker_id1[63:0]	Input	Equivalent to the PCS lane 1 alignment marker.	
ctl_tx_vl_marker_id2[63:0]	Input	Equivalent to the PCS lane 2 alignment marker.	
ctl_tx_vl_marker_id3[63:0]	Input	Equivalent to the PCS lane 3 alignment marker	
RS-FEC Status Signals			
stat_rx_rsfec_corrected_cw_inc	Output	Increment for corrected errors.	
stat_rx_rsfec_uncorrected_cw_inc	Output	Increment for uncorrected errors.	
stat_rx_rsfec_err_count_inc[2:0]	Output	Increment for detected errors.	
stat_rx_rsfec_hi_ser	Output	Set to one if the number of RS-FEC symbol errors in a window of 8192 codewords exceeds the threshold K = 417 and is set to zero otherwise.	
stat_rx_rsfec_lane_alignment_status	Output	A value of 1 indicates that the RX RS-FEC block has achieved alignment on the data from the transceiver.	
stat_tx_rsfec_lane_alignment_status	Output	A value of 1 indicates that the TX RS-FEC block has achieved alignment on the incoming PCS data.	

RS-FEC Functional Description

The RS-FEC feature of the 10G/25G subsystem provides error correction capability according to IEEE 802.3 Clause 108 or Schedule 3 of the 25G Ethernet Consortium.

The feature requires the insertion of PCS alignment markers as defined in IEEE 802.3 Table 82-2. Inputs are provided for the alignment markers and also for the value of words between alignment markers.

It is possible to bypass the RS-FEC function by means of the enable signals. This will bypass the RS-FEC function and connect the PCS directly to the transceiver, with the benefit of reduced latency. Refer to *25G IEEE 802.3by Reed-Solomon Forward Error Correction LogiCORE IP Product Guide* (PG217) [Ref 11] for the latest latency performance data in the various bypass modes, defined as follows:

- FEC Bypass Correction: The decoder performs error detection without correction, (see IEEE Std 802.3by section 108.5.3.2. The latency is reduced in this mode (see *25G IEEE 802.3by Reed-Solomon Forward Error Correction LogiCORE IP Product Guide* (PG217) [Ref 11] for latency figures).
- FEC Bypass Indication: In this mode there is correction of the data but no error indication. An additional signal, `rx_hi_ser`, is generated in this mode to reduce the likelihood that errors in a packet are not detected.The RS decoder counts the number of symbol errors detected in consecutive non-overlapping blocks of 8192 codewords (see IEEE Std 802.3by section 108.5.3.2. The latency is reduced in this mode.
- Decoder Bypass: The RS decoder can be bypassed by setting the IEEE Error indication Low when the correction bypass and indication bypass are High.

Status/Control Interface

The Status/Control interface allows you to set up the 10G/25G Ethernet core configuration and to monitor its status. This sections describes in more detail some of the Status and Control signals.

stat_rx_framing_err and stat_rx_framing_err_valid

These signals are used to keep track of sync header errors. This set of buses is used to keep track of sync header errors. The `stat_rx_framing_err` output indicates how many sync header errors were received and it is qualified (that is, the value is only valid) when the corresponding `stat_rx_framing_err_valid` is sampled as a 1.

stat_rx_block_lock

This bit indicates that the interface has achieved sync header lock as defined by IEEE Std. 802.3. A value of 1 indicates block lock is achieved.

stat_rx_local_fault

This output is High when `stat_rx_internal_local_fault` or `stat_rx_received_local_fault` is asserted. This is output is level sensitive.

RX Error Status

The core provides status signals to identify 64b/66b words and sequences violations and CRC32 checking failures.

All signals are synchronous with the rising-edge of `clk` and a detailed description of each signal follows.

stat_rx_bad_fcs[1:0]

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 CRC32 error is detected, the received packet is marked as containing an error and is sent with `rx_errout` asserted during the last transfer (the cycle with `rx_eopout` asserted), unless `ctl_rx_ignore_fcs` is asserted. This signal is asserted for one clock period for each CRC32 error detected.

stat_rx_bad_code

This signal indicates how many cycles the RX PCS receive state machine is in the RX_E state as defined by IEEE Std. 802.3.

Pause Processing

The 10G/25G Ethernet 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 behavior 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 10G/25G Ethernet core automatically calculates and adds the FCS to the packet. For priority pause packets the 10G/25G Ethernet 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 10G/25G Ethernet 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 10G/25G Ethernet 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 10G/25G Ethernet 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.

   Packets that pass step one are forwarded to you only if `ctl_rx_forward_control` is set to 1.

2. If step one passes, the packet is checked to determine if it is a global pause packet.
3. If step two 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;
```

where 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;
```

where 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;
```

where 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. 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.

   If one of the bits of `ctl_rx_pause_enable[8:0]` is set to 0 (disabled) when the pause processing is in step 2 or later, the core completes the steps as normal until it comes back to step 1.

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 Figure 3-15 with each step shown on the waveform.

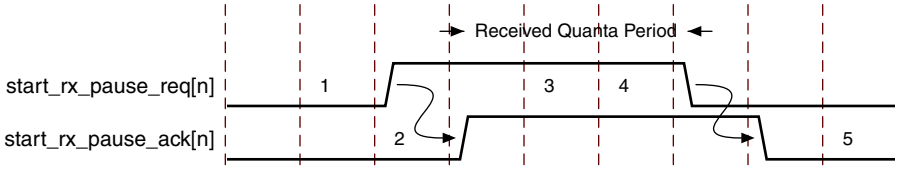

Figure 3-15: **RX Pause Interface Example**

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.

Auto-Negotiation

A block diagram of the 10G/25G Ethernet core with Auto-Negotiation (AN) and Link Training (LT) is shown in Figure 3-16.

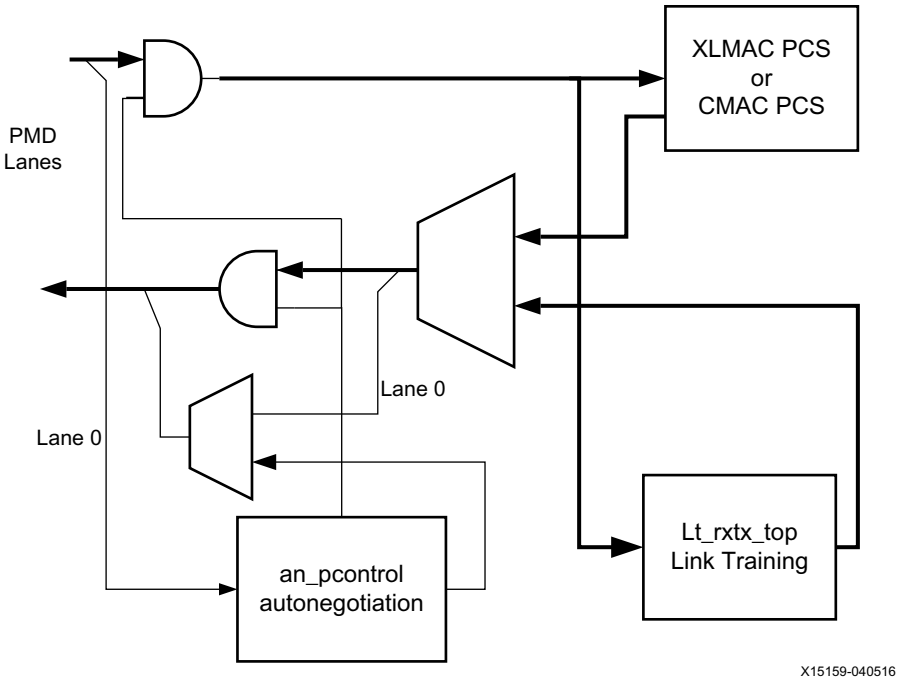

Figure 3-16: **Core with Auto-Negotiation and Link Training**

The auto-negotiation function allows an Ethernet device to advertise the modes of operation it possesses to another device at the remote end of a backplane Ethernet link and to detect corresponding operational modes the other device might be advertising. The objective of this auto-negotiation function is to provide the means to exchange information between two devices and to automatically configure them to take maximum advantage of their abilities. It has the additional objective of supporting a digital signal detect to ensure that the device is attached to a link partner rather than detecting a signal due to crosstalk.

When auto-negotiation is complete, ability is reported according to the available modes of operation.

Link Training is performed after auto-negotiation if the Link Training function is supported by both ends of the link. Link Training is typically required due to frequency-dependent losses which can occur as digital signals traverse the backplane. The primary function of the Link Training block included with this core is to provide register information and a training sequence over the backplane link which is then analyzed by a receiving circuit (part of the transceiver). The other function of the Link Training block is to communicate training feedback from the receiver to the corresponding transmitter so that its equalizer circuit (part of the transceiver) can be adjusted as required. The decision-making algorithm is not part of this core.

When auto-negotiation and Link Training are complete, the datapath is switched to mission mode (the PCS), as shown in Figure 3-16.

Overview

Figure 3-17 shows the position of the auto-negotiation function in the OSI reference model.

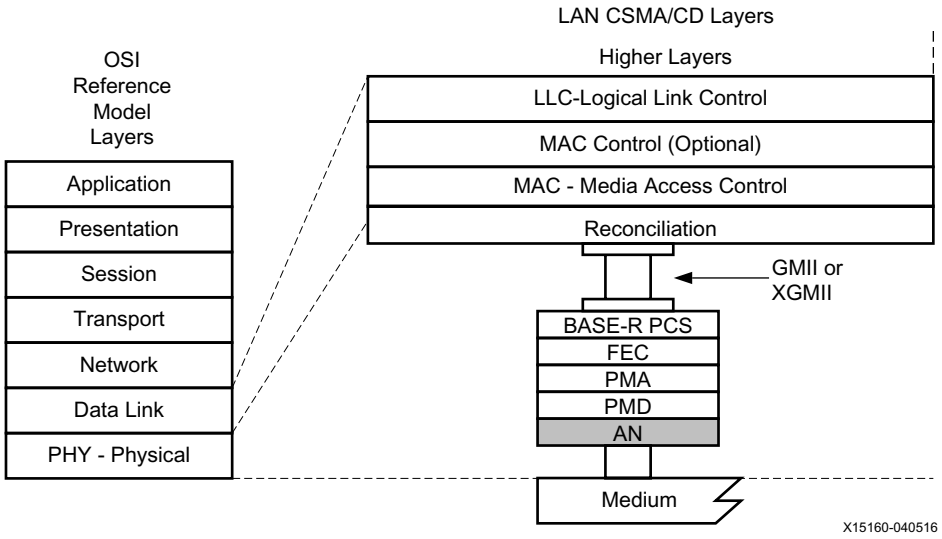

Figure 3-17: **Auto-Negotiation Function in the OSI Model**

The Auto-Negotiation Intellectual Property Core (ANIPC) implements the requirements as specified in Clause 73, IEEE Std 802.3-2012, including those amendments specified in IEEE Std. P802.3ba and 802.3ap.

The functions of the ANIPC core are listed in clause 73, specifically Figure 73-11, Arbitration state diagram, in section 73.10.4, State Diagrams.

During normal mission mode operation, with link control outputs set to (bin)11, the bit operating frequency of the transceiver input and output is typically 10.3125 or 25.78125 Gb/s. However, the Dual Manchester Encoding (DME) bit rate used on the lane during Auto-Negotiation is different to the mission mode operation. To accommodate this requirement, the ANIPC core uses over-sampling and over-driving to match the 156.25 Mb/s Auto-Negotiation speed (DME clock frequency 312.5 MHz) with the mission mode 10.3125 or 25.78125 Gb/s physical lane speed.

Functional Description

autoneg_enable

When the `autoneg_enable` input signal is set to 1, auto-negotiation begins automatically at power-up, or if the carrier signal is lost, or if the input `restart_negotiation` signal is cycled from a 0 to a 1. All of the Ability input signals as well as the two input signals `PAUSE` and `ASM_DIR` are tied Low or High to indicate the capability of the hardware. The `nonce_seed[7:0]` input must be set to a unique non-zero value for every instance of the auto-negotiator. This is important to guarantee that no dead-locks occur at power-up. If two link partners connected together attempt to auto-negotiate with their `nonce_seed[7:0]` inputs set to the same value, the auto-negotiation fails continuously. The `pseudo_sel` input is an arbitrary selection that is used to select the polynomial of the random bit generator used in bit position 49 of the DME pages used during auto-negotiation. Any selection on this input is valid and does not result in any adverse behavior.

Link Control

When auto-negotiation begins, the various link control signals are activated, depending on the disposition of the corresponding Ability inputs for those links. Subsequently, the corresponding link status signals are monitored by the ANIPC hardware for an indication of the state of the various links that are connected. If particular links are unused, the corresponding link control outputs are unconnected, and the corresponding link-status inputs should be tied Low. During this time, the ANIPC hardware sets up a communication link with the link partner and uses this link to negotiate the capabilities of the connection.

Autoneg Complete

When Auto-Negotiation is complete, the `autoneg_complete` output signal is asserted. In addition, the output signal `an_fec_enable` is asserted if the Forward Error Correction hardware is to be used; the output signal `tx_pause_en` is asserted if the transmitter hardware is allowed to generate PAUSE control packets, the output signal `rx_pause_en` is asserted if the receiver hardware is allowed to detect PAUSE control packets, and the output link control of the selected link is set to its mission mode value (bin)11.

Link Training

Link Training is performed after Auto Negotiation converges to a backplane or copper technology. Technology selection can also be the result of a manual entry or parallel detection. Link training might be required due to frequency-dependent losses that can occur as digital signals traverse the backplane or a copper cable. The primary function of the Link Training core is to provide register information and a training sequence over the backplane link which is then analyzed by a receiving circuit which is not part of the core.

The other function of the core is to communicate training feedback from the receiver to the corresponding transmitter so that its equalizer circuit (not part of the core) can be adjusted as required. The two circuits comprising the core are the receive Link Training block and the transmit Link Training block.

IMPORTANT: *The logic responsible for adjusting the transmitter pre-emphasis taps must be supplied external to this IP core.*

Transmit

The Link Training transmit block constructs a 4,384-bit frame which contains a frame delimiter, control channel, and link training sequence. It is formatted as shown in Figure 3-18.

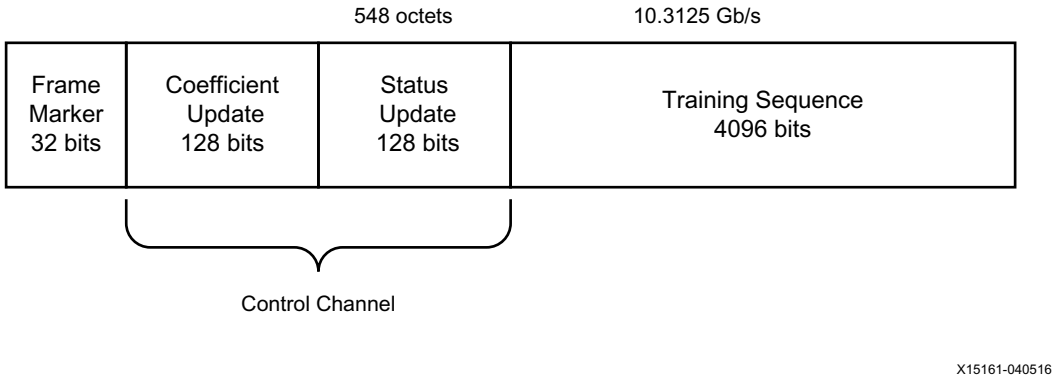

Figure 3-18: **Link Training Frame Structure**

Xilinx recommends that the control channel bits not be changed by the Link Training algorithm while the transmit state machine is in the process of transmitting them, or they can be received incorrectly, possibly resulting in a DME error. This time begins when `tx_SOF` is asserted and ends at least 288 bit times later, or approximately 30 ns.

Although the coefficient and status contain 128 bit times at the line rate, the actual signaling rate for these two fields is reduced by a factor of 8. Therefore the DME clock rate is one quarter of the line rate.

Frame Marker

The frame marker consists of 16 consecutive 1s followed by 16 consecutive 0s. This pattern is not repeated in the remainder of the frame.

Coefficient and Status

Because the DME signaling rate for these two fields is reduced by a factor of 8, each coefficient and status transmission contain 128/8=16 bits each numbered from 15:0. Table 3-4 and Table 3-5 define these bits in the order in which they are transmitted starting with bit 15 and ending with bit 0.

Table 3-4: **Coefficient and Update Field Bit Definitions**

Bits	Name	Description
15:14	Reserved	Transmitted as 0, ignored on reception.
13	Preset	1 = Preset coefficients 0 = Normal operation
12	Initialize	1 = Initialize coefficients 0 = Normal operation
11:6	Reserved	Transmitted as 0, ignored on reception.
5:4	Coefficient (+1) update	1 1 = reserved 0 1 = increment 1 0 = decrease 0 0 = hold
3:2	Coefficient (0) update	1 1 = reserved 0 1 = increment 1 0 = decrease 0 0 = hold
1:0	Coefficient (-1) update	1 1 = reserved 0 1 = increment 1 0 = decrease 0 0 = hold

Table 3-5: **Status Report Field Bit Definitions**

Bits	Name	Description
15	Receiver ready	1 = The local receiver has determined that training is complete and is prepared to receive data. 0 = The local receiver is requesting that training continue.
14:6	Reserved	Transmitted as 0, ignored on reception.
5:4	Coefficient (+1) update	0 1 = minimum 1 1 = maximum 1 0 = updated 0 0 = not_updated
3:2	Coefficient (0) update	1 1 = maximum 0 1 = minimum 1 0 = updated 0 0 = not_updated
1:0	Coefficient (-1) update	1 1 = maximum 0 1 = minimum 1 0 = updated 0 0 = not_updated

The functions of each bit are defined in IEEE Std. 802.3, Clause 72. Their purpose is to communicate the adjustments of the transmit equalizer during the process of link training. The corresponding signal names are defined in Table 2-15.

Training Sequence

The training sequence consists of a pseudo-random bit sequence (PRBS) of 4,094 bits followed by two zeros, for a total of 4,096 bits. The PRBS is transmitted at the line rate of 10.3125 or 25.78125 Gb/s. The PRBS generator receives an 11-bit seed from an external source. Subsequent to the initial seed being loaded, the PRBS generator continues to run with no further intervention being required.

The PRBS generator itself is implemented with a circuit which corresponds to the following polynomial:

$$G(x) = 1 + x9 + x11$$

Receive

The receive block implements the frame alignment state diagram shown in IEEE Std. 802.3, Clause 72, Figure 72-4.

Frame Lock State Machine

The frame lock state machine searches for the frame marker, consisting of 16 consecutive 1s followed by 16 consecutive 0s. This functionality is fully specified in IEEE Std. 802.3, Clause 72, Fig. 72-4. When frame lock has been achieved, `frame_lock` is set to a value of TRUE.

Received Data

The receiver outputs the control channel with the bit definitions defined in Table 3-4 and Table 3-5 and signal names defined in Port Descriptions.

If a DME error has occurred during the reception of a particular DME frame, the control channel outputs are not updated but retain the value of the last received good DME frame and are updated when the next good DME frame is received.

Design Flow Steps

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

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 4]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 4] 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 core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 6].

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

Configuration Tab

The Configuration tab (Figure 4-1) provides the basic core configuration options.

Default values are pre-populated in all tabs.

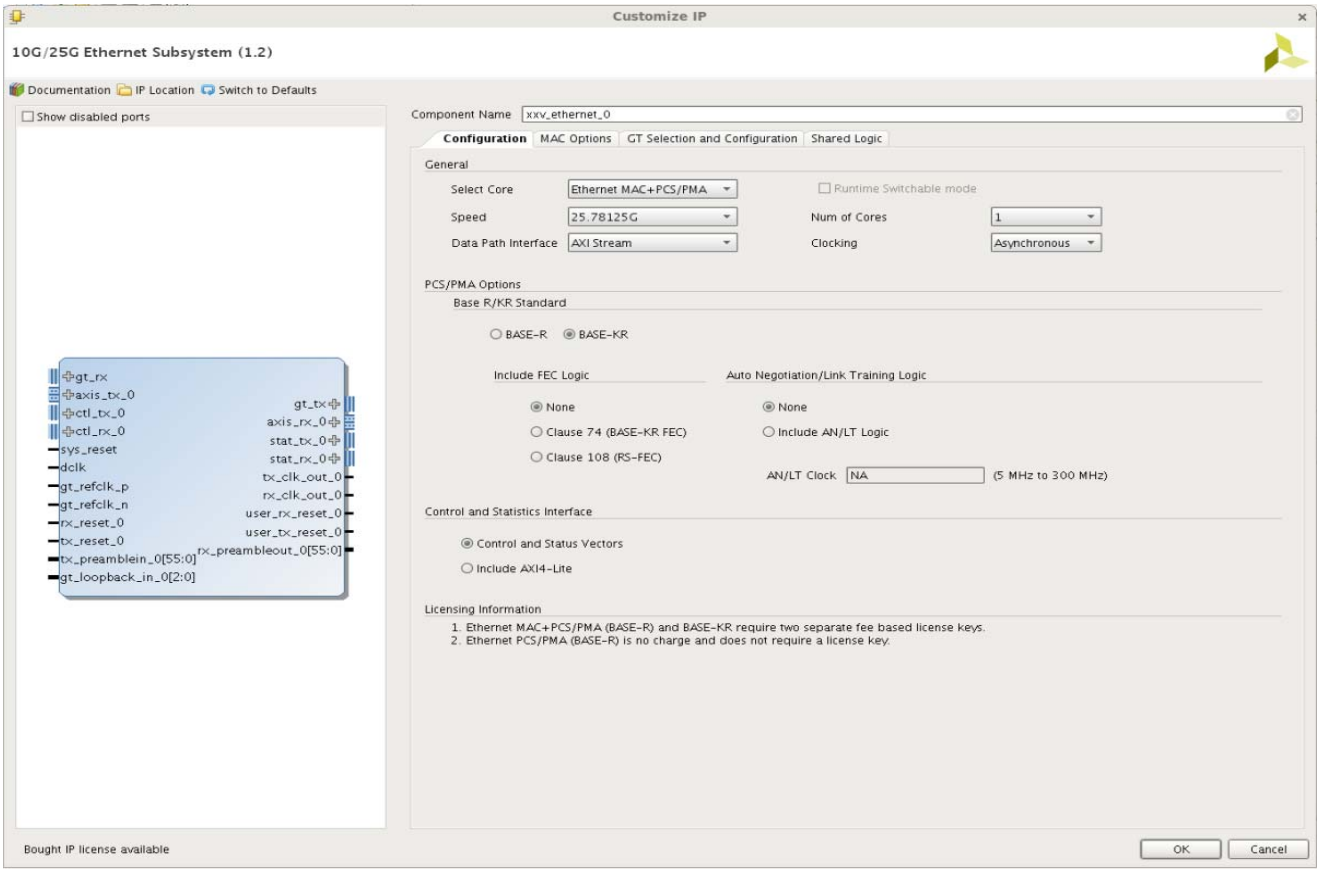

Figure 4-1: **Configuration Tab**

Table 4-1: **Configuration Options**

Option	Values	Default
General		
Select Core	Ethernet MAC+PCS/PMA Ethernet PCS/PMA	Ethernet MAC+PCS/PMA
Speed	25.7812G 10.3125G	25.7812G
Runtime Switchable Mode	0, 1	0
Num of Cores	1 2 3 4	1
Clocking	Synchronous Asynchronous	Asynchronous
Data Path Interface	AXI Stream(1) Media Independent Interface (MII)(2)	AXI Stream
PCS/PMA Options		
Base-R Base-KR	Base-R Base-KR	Base-KR
Include FEC Logic(3)	None Clause 74 (BASE-KR FEC) Clause 108 (RS-FEC)(4)	None
Auto Negotiation/Link Training Logic	None Include AN/LT Logic	None
AN/LT Clock (5 MHz to 300 MHz)	5 MHz to 300 MHz	75 MHz
Control and Statistics interface	Control and Status Vectors Include AXI4-Lite	Control and Status Vectors

Notes:

1. The AXI4-Stream interface is visible and is the only option for the Ethernet MAC+PCS/PMA core.
2. The MII interface is visible and is the only option for the Ethernet PCS/PMA core.
3. Include FEC logic is not supported for Base-R.
4. Clause 108 (RS-FEC) is not supported for 10G speed and also for runtime switchable mode.

MAC Options Tab

The MAC Options tab (Figure 4-2) provides additional core configuration options.

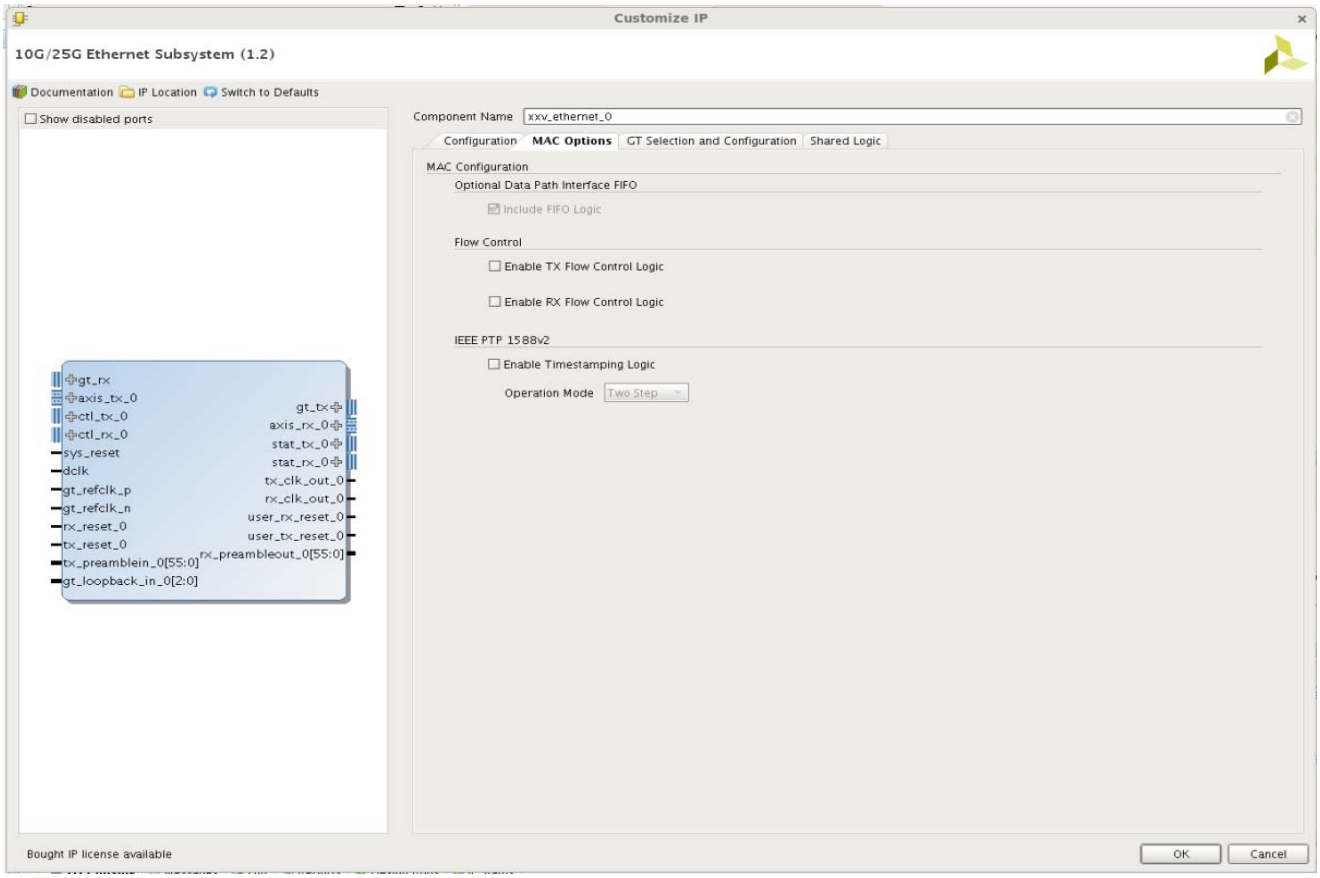

Figure 4-2: **MAC Options Tab**

Table 4-2: **MAC Options**

Option	Values	Default
Optional Data Path Interface FIFO Options		
Include FIFO Logic	Checked, Unchecked	Checked
Flow Control Options		
Enable TX Flow Control Logic	Checked, Unchecked	Unchecked
Enable RX Flow Control Logic	Checked, Unchecked	Unchecked
IEEE PTP 1588v2 Options		
Enable Timestamping Logic	Checked, Unchecked	Unchecked
Operation Mode	One Step Two Step	Two Step

GT Selection and Configuration Tab

The GT Selection and Configuration tab (Figure 4-3) enables you to configure the serial transceiver features of the core.

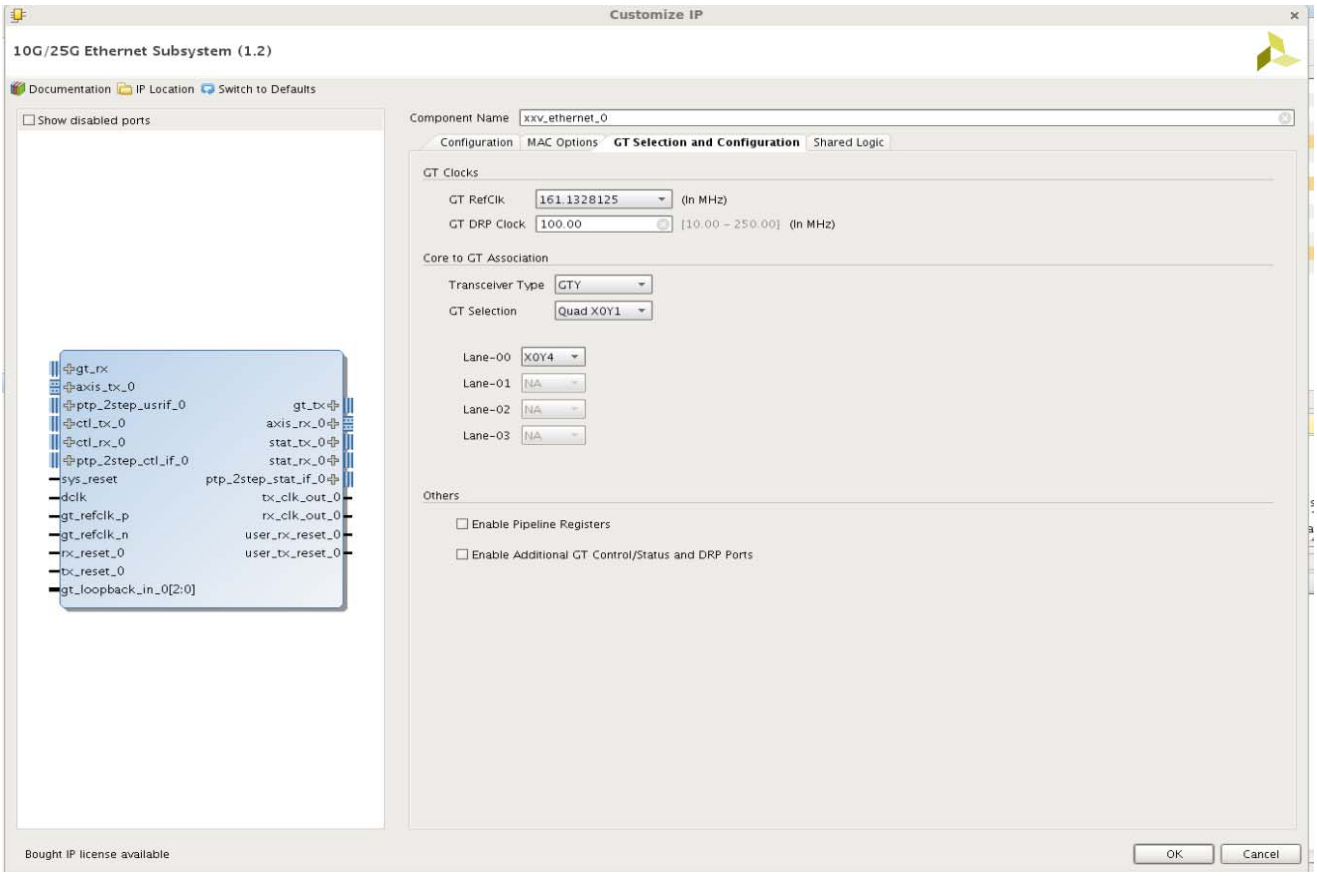

Figure 4-3: **GT Selection and Configuration Tab**

Table 4-3: **GT Clocks Options**

Option	Values	Default
GT Clocks Options		
GT RefClk (In MHz)	161.1328125 195.3125 201.4160156 257.8125 322.265625	161.1328125
GT DRP Clock (In MHz)	10 – 250 MHz	100.00
Core to Transceiver Association Options		
Transceiver 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	
Other Options		
Enable Pipeline Register	Checked, Unchecked	Unchecked
Enable Additional GT Control and Status Ports	Checked, Unchecked	Unchecked

Shared Logic Tab

The Shared Logic tab (Figure 4-4) enables you to use shared logic in either the core or the example design.

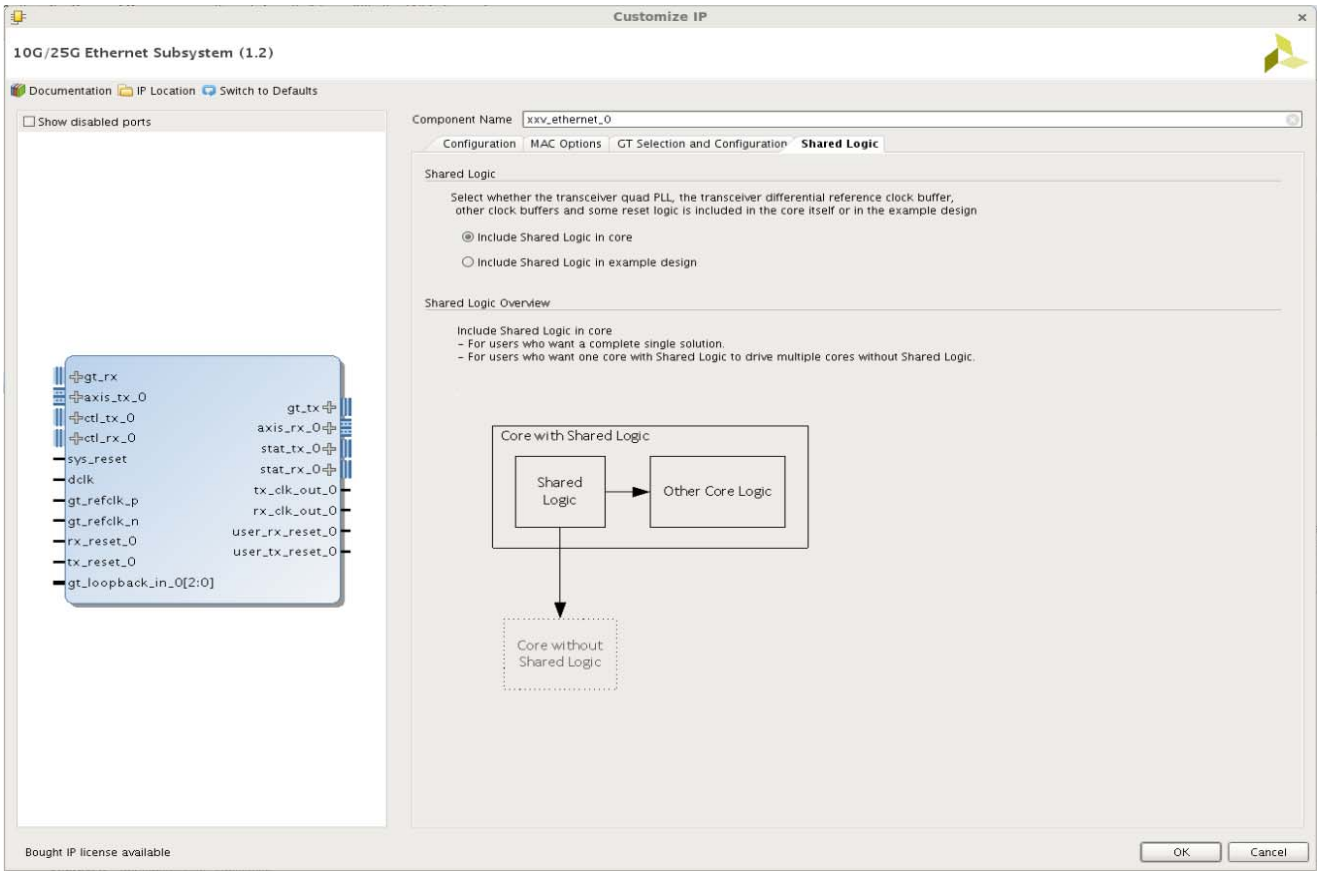

Figure 4-4: **Shared Logic Tab**

Table 4-4: **Shared Logic Options**

Options	Default
Include Shared Logic in Core Include Shared Logic in Example	Include Shared Logic in Core

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

This section is not applicable for this core.

Device, Package, and Speed Grade Selections

This section is not applicable for this core.

Clock Frequencies

This section is not applicable for this core.

Clock Management

This section is not applicable for this core.

Clock Placement

This section is not applicable for this core.

Banking

This section is not applicable for this core.

Transceiver Placement

This section is not applicable for this core.

I/O Standard and Placement

This section is not applicable for this core.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5].

Example Design

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

Overview

Figure 5-1 shows the instantiation of various modules and their hierarchy for a single core configuration of xxv_ethernet_0 example design.

The top module `xxv_ethernet_0_exdes.v` is an instantiation of the DUT (that is, `xxv_ethernet_0.v`), `pkt_gen_mon_0.v`, and Optional modules, such as `xxv_ethernet_0_sharedlogic_wrapper.v` and `xxv_ethernet_0_trans_debug.v`.

The `xxv_ethernet_0.v` module is an instantiation of the `xxv_ethernet_0_wrapper.v` module, which is instantiated from `xxv_ethernet_0_top.v` and the GT, along with various helper blocks.

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.

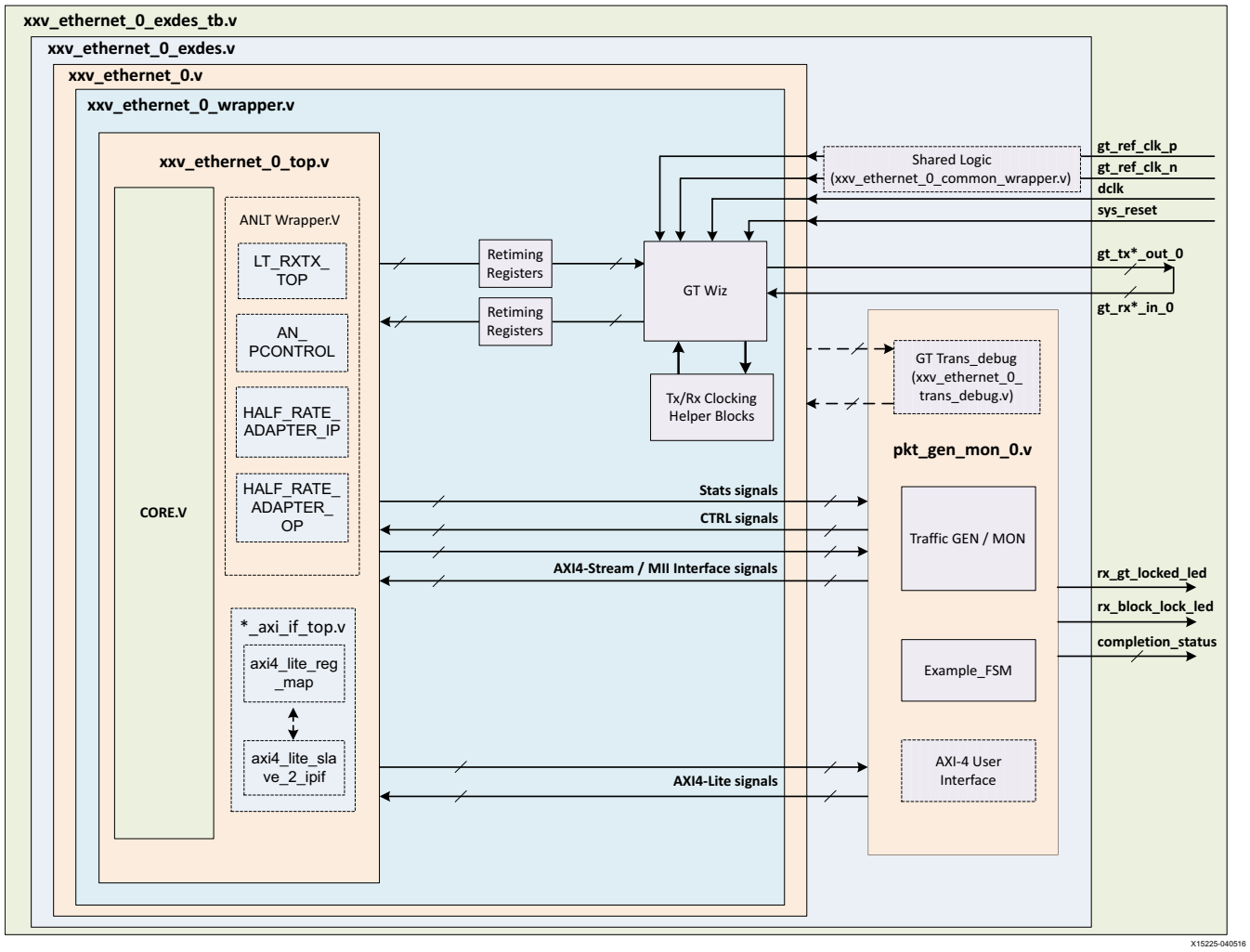

Figure 5-1: **Single Core Example Design Hierarchy**

Following are the user interfaces available for different configurations.

- MAC/PCS configuration:
  - AXI4-Stream for datapath interface
  - AXI4-Lite for control and statistics interface
- PCS configuration:
  - MII for datapath interface
  - AXI4-Lite for control and statistics interface

The `xxv_ethernet_0_pkt_gen_mon` module is used to generate the data packets for sanity testing. The packet generation and checking is controlled by a FSM module.

The optional modules are described as follows:

- **xxv_ethernet _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 the Vivado IDE or the **Runtime Switchable** mode option in the in the Configuration Tab. This module brings out all the GT channel DRP ports, and some control and status ports of the transceiver module out of the xxv_ethernet core.
- **xxv_ethernet_sharedlogic_wrapper**: This module is present in the example design when you select the **Include Shared Logic in Example Design** from the Shared Logic Tab of the Vivado IDE. This module brings the transceiver common module out of the 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 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 GT and the core.
- **TX / RX Sync register**: The TX Sync register double synchronizes the data from the core to the GT with respect to the `tx_clk`. The RX Sync register double synchronizes the data from the GT to the core with respect to the `rx_serdes_clk`.

Note: For Runtime Switchable, if Auto Negotiation/Link training is selected in Vivado IDE, then AN/LT operation will be performed only with 10G data rate during switchings.

Figure 5-2 shows the instantiation of various modules and their hierarchy for the multiple core configuration of xxv_ethernet_0 example design.

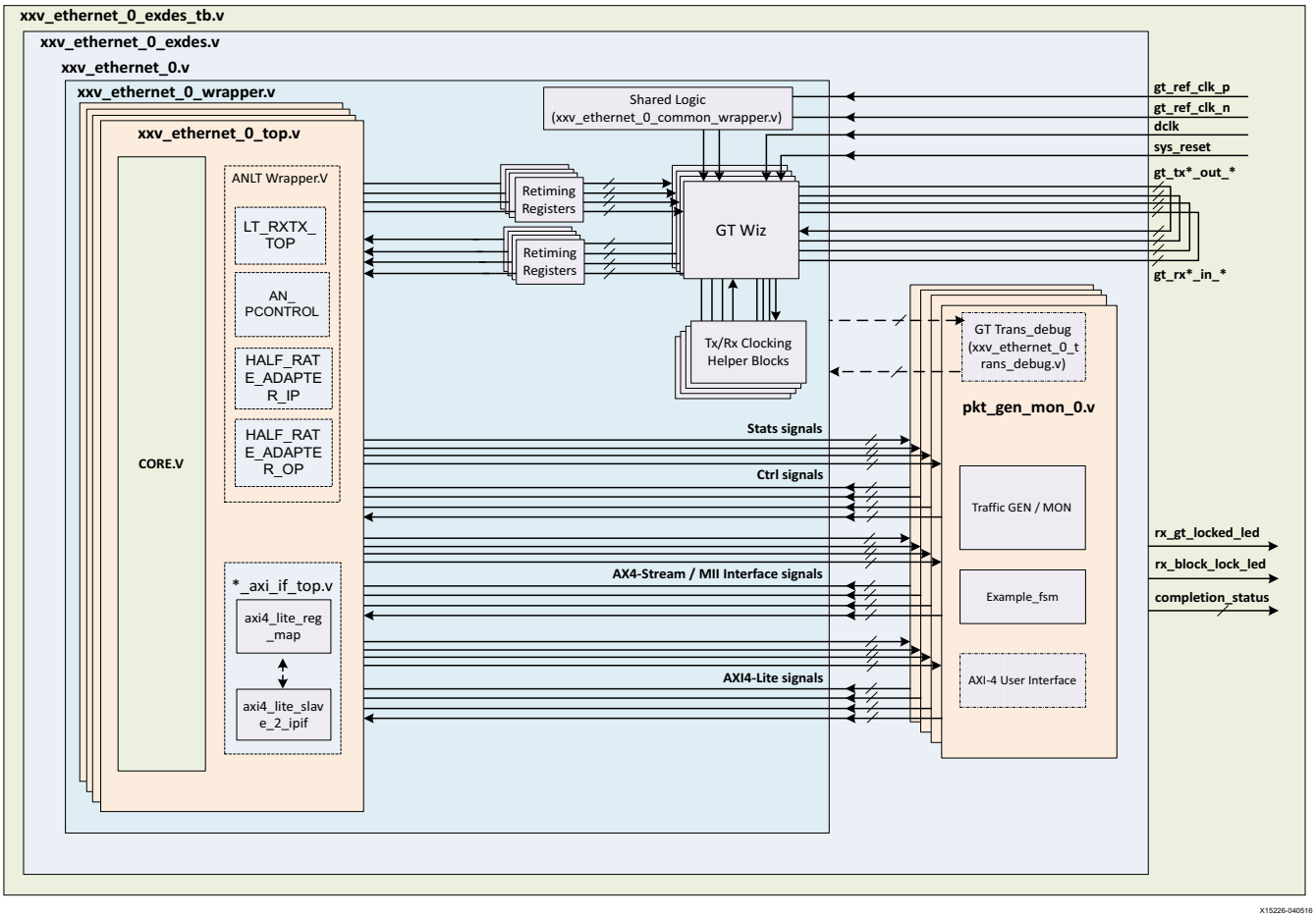

Figure 5-2: **Multiple Core Example Design Hierarchy**

User Interface

General purpose I/Os (GPIOs) are provided to control the example design. The user input and user output ports are described in Table 5-1.

Table 5-1: **User Input and User Output Ports**

Name	Size	Direction	Description
sys_reset	1	Input	Reset for xxv_ethernet core.
gt_ref_clk_p	1	Input	Differential input clk to GT.
gt_ref_clk_n	1	Input	Differential input clk to GT.
dclk	1	Input	Stable/free running input clk to GT.
rx_gt_locked_led_0	1	Output	Indicates that GT has been locked.
rx_block_lock_led_0	1	Output	Indicates RX block lock has been achieved.
restart_tx_rx_0	1	Input	This signal is used to restart the packet generation and reception for the data sanity test when the packet generator and the packet monitor are in idle state.
completion_status	5	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.

Table 5-1: **User Input and User Output Ports *(Cont'd)***

Name	Size	Direction	Description
mode_change_0	1	Input	This port is available only when **Runtime Switchable** is selected in Vivado IDE and this is used to switch the core speed.
core_speed_0	1	Input	This signal indicates the speed with which the core is working: 1'b1 = 10G and 1'b0 = 25G

Duplex Mode of Operation

In this mode of operation, both the transmitter and receiver of the core are active and loopback is provided at the GT output interface, that is, output is fed back as input. Packet generation and monitor modules are active in this mode. The generator module is responsible for generating the desired number of packets and transmit to the core using the available data interface. The monitor module checks the packets from the receiver.

Figure 5-3 shows the duplex mode of operation.

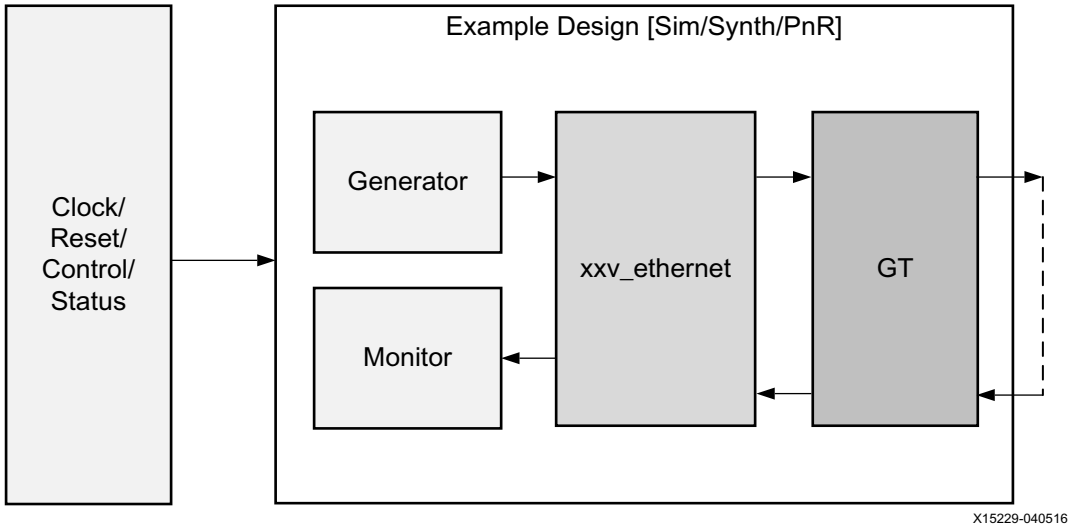

Figure 5-3: **Duplex Mode of Operation**

Shared Logic Implementation

Shared logic includes the GT common module that is part of the core or the example design.

- By default, the GT common module is available with the core.
- To instantiate the GT common module in the example design, select the **Include Shared Logic in Example Design** option in the GT Selection and Configuration Tab of the Vivado IDE.

Figure 5-4 shows the implementation when shared logic is instantiated in the example design for single core.

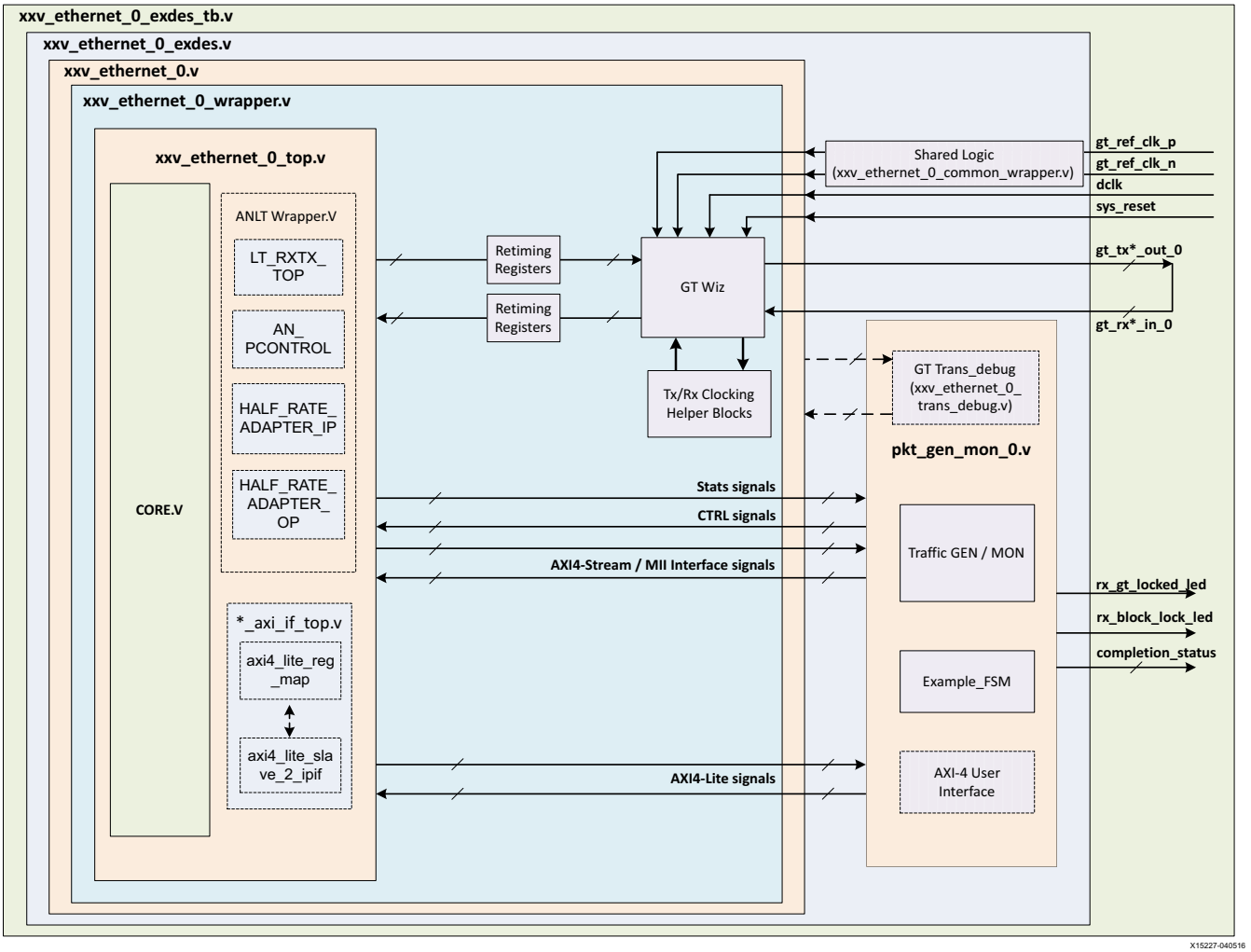

Figure 5-4: **Single Core Example Design Hierarchy With Shared Logic Implementation**

Figure 5-5 shows the implementation when shared logic is instantiated in the example design for multiple cores.

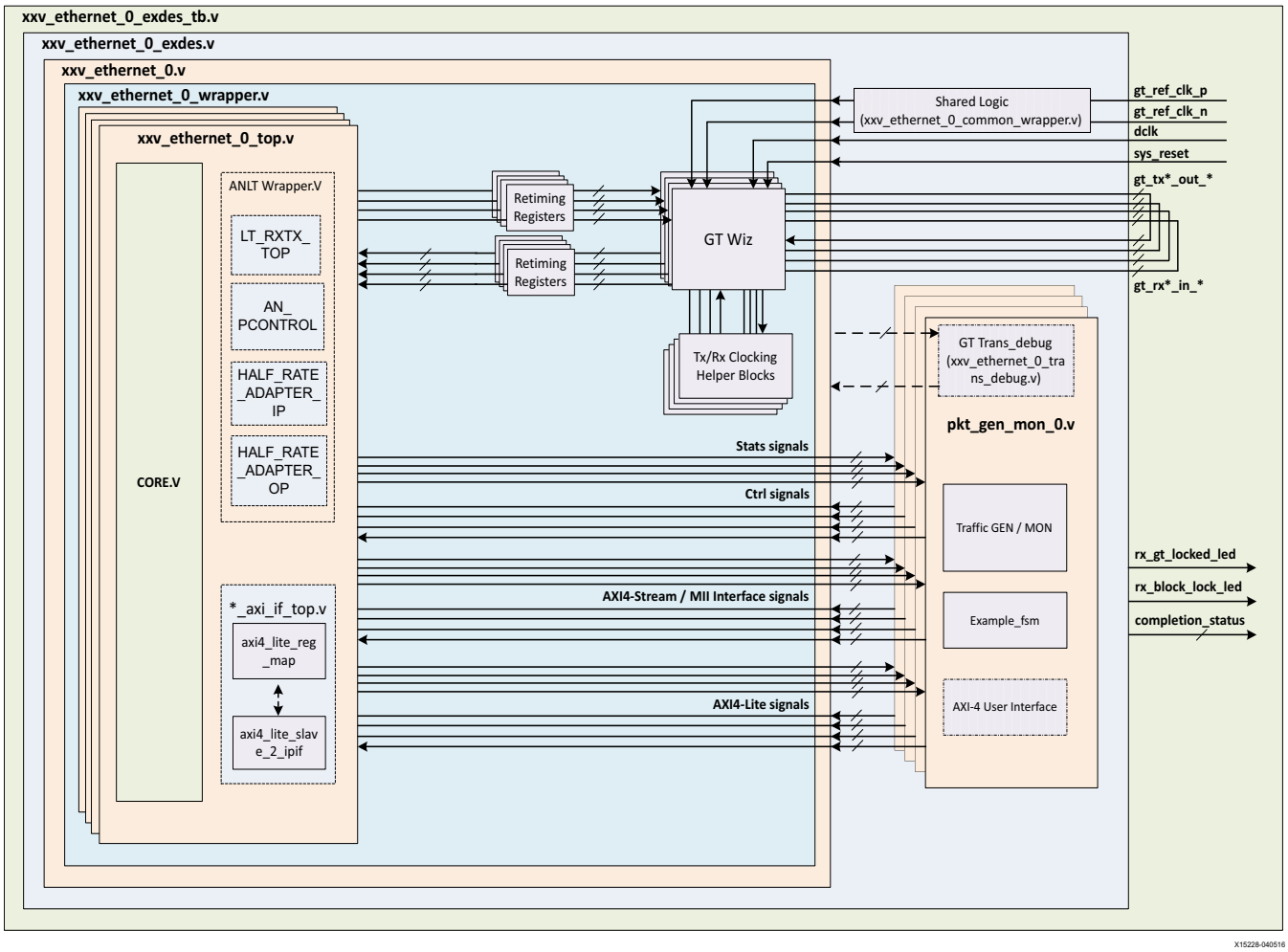

Figure 5-5: **Multiple Core Example Design Hierarchy With Shared Logic Implementation**

AXI4-Lite Interface Implementation

In order to instantiate the AXI4-Lite interface to access the control and status registers of the xxv_ethernet core, enable the **Include AXI4-Lite** check box in the Configuration Tab of the Vivado IDE. This option enables the `xxv_ethernet _0_axi_if_top` module (which contains `xxv_ethernet_0_pif_registers` with the `xxv_ethernet _0_slave_2_ipif` module). You can access the AXI4-Lite interface logic registers (control, status and statistics) from the `xxv_ethernet _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.

Batch Mode Test Bench

Each batch mode release of the 10G/25G Ethernet subsystem includes a demonstration test bench that performs a loopback test on the complete subsystem. For your convenience, scripts are provided to launch the test bench from several industry-standard simulators. The test program exercises the datapath to check that the transmitted frames are received correctly. Register Transfer Level (RTL) simulation models for the subsystem 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 Vivado® Design Suite.

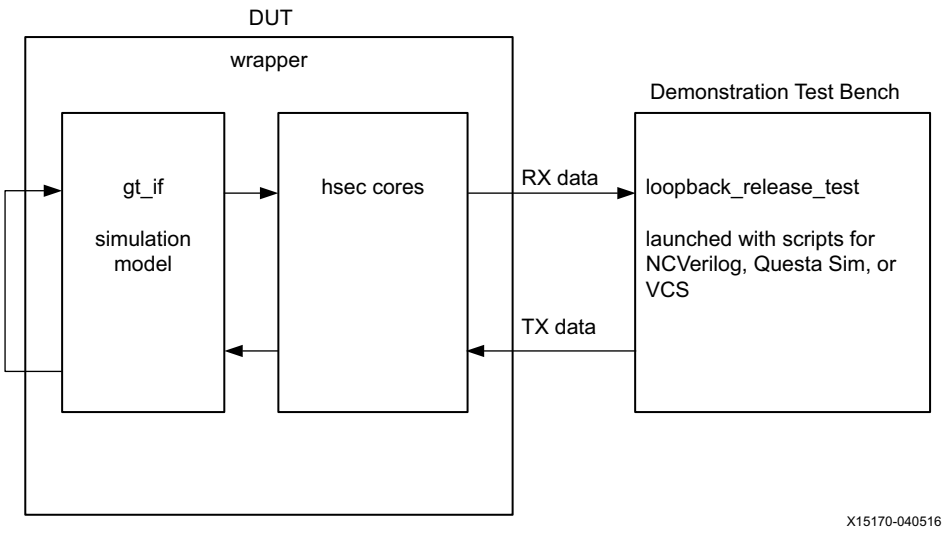

Figure 6-1: **Test Bench**

Appendix A

Migrating and Upgrading

This appendix contains information about upgrading to a more recent version of the IP core in the Vivado Design Suite.

A script is provided for upgrading the design to a more recent version of the Vivado® Design Suite. The script is found in the `/compile/xilinx/upgrade_IP` directory. Run this script for the following upgrades:

- To upgrade the transceiver wrapper to the latest version.
- To use the latest transceiver simulation model, which if not upgraded can result in simulation errors.

TIP: *Before running the script, save a copy of the original design in the event that you need to revert to the previous version.*

Debugging

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

TIP: *If the IP generation halts with an error, there might be a license issue. See License Checkers in Chapter 1 for more details.*

Finding Help on Xilinx.com

To help in the design and debug process when using the 10G/25G Ethernet, the Xilinx 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.

Documentation

This product guide is the main document associated with the 10G/25G Ethernet. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Refer to the Xilinx Ethernet IP Solution Center.

Answer Records

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

Answer Records for this subsystem can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the 10G/25G Ethernet

AR: 64710

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this product when used as described in the product documentation. Xilinx 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 contact Xilinx Technical Support, go to the Xilinx Support web page.

Debug Tools

There are many tools available to address 10G/25G Ethernet design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

The 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 Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 8].

Reference Boards

Various Xilinx development boards support the 10G/25G Ethernet. These boards can be used to prototype designs and establish that the core can communicate with the system.

- UltraScale™ FPGA evaluation boards
  - VCU107
  - VCU108

Simulation Debug

Each 10G/25G Ethernet release includes a sample simulation test bench. This consists of a loopback from the TX side of the user interface, through the TX circuit, looping back to the RX circuit, and checking the received packets at the RX side of the user interface.

Each release includes a sample instantiation of a Xilinx transceiver corresponding to the device selected by the customer. The loopback simulation includes a path through the transceiver.

The simulation is run using provided scripts for several common industry-standard simulators.

If the simulation does not run properly from the scripts, the following items should be checked.

Simulator License Availability

If the simulator does not launch, you might not have a valid license. Ensure that the license is up to date. It is also possible that your organization has a license available for one of the other simulators, so try all the provided scripts.

Library File Location

Each simulation script calls up the required Xilinx library files. These are called by the corresponding liblist* file in the bin directory of each release.

There can be an error message indicating that the simulator is unable to find certain library files. In this case, the path to the library files might have to be modified. Check with your IT administrator to ensure that the paths are correct.

Version Compatibility

Each release has been tested according the Xilinx tools version requested by the customer. If the simulation does not complete successfully, you should first ensure that a properly up-to-date version of the Xilinx tools is used. The preferred version is indicated in the README file of the release, and is also indicated in the simulation sample log file included with the release.

Slow Simulation

Simulations can appear to run slowly under some circumstances. If a simulation is unacceptably slow, the following suggestions might improve the run-time performance.

1. Use a faster computer with more memory.
2. Make use of a Platform Load Sharing Facility (LSF) if available in your organization.
3. Bypass the Xilinx transceiver (this might require that the customer create their own test bench.
4. Send fewer packets. This can be accomplished by modifying the appropriate parameter in the provided sample test bench.

Simulation Fails Before Completion

If the sample simulation fails or hangs before successfully completing, it is possible that a timeout has occurred. Ensure that the simulator timeouts are long enough to accommodate the waiting periods in the simulation, for example during the lane alignment phase.

Simulation Completes But Fails

If the sample simulation completes with a failure, contact Xilinx technical support. Each release is tested prior to shipment and normally completes successfully. Consult the sample simulation log file for the expected behavior.

The simulation debug flow for Questa® SIM is illustrated in Figure B-1. A similar approach can be used with other simulators.

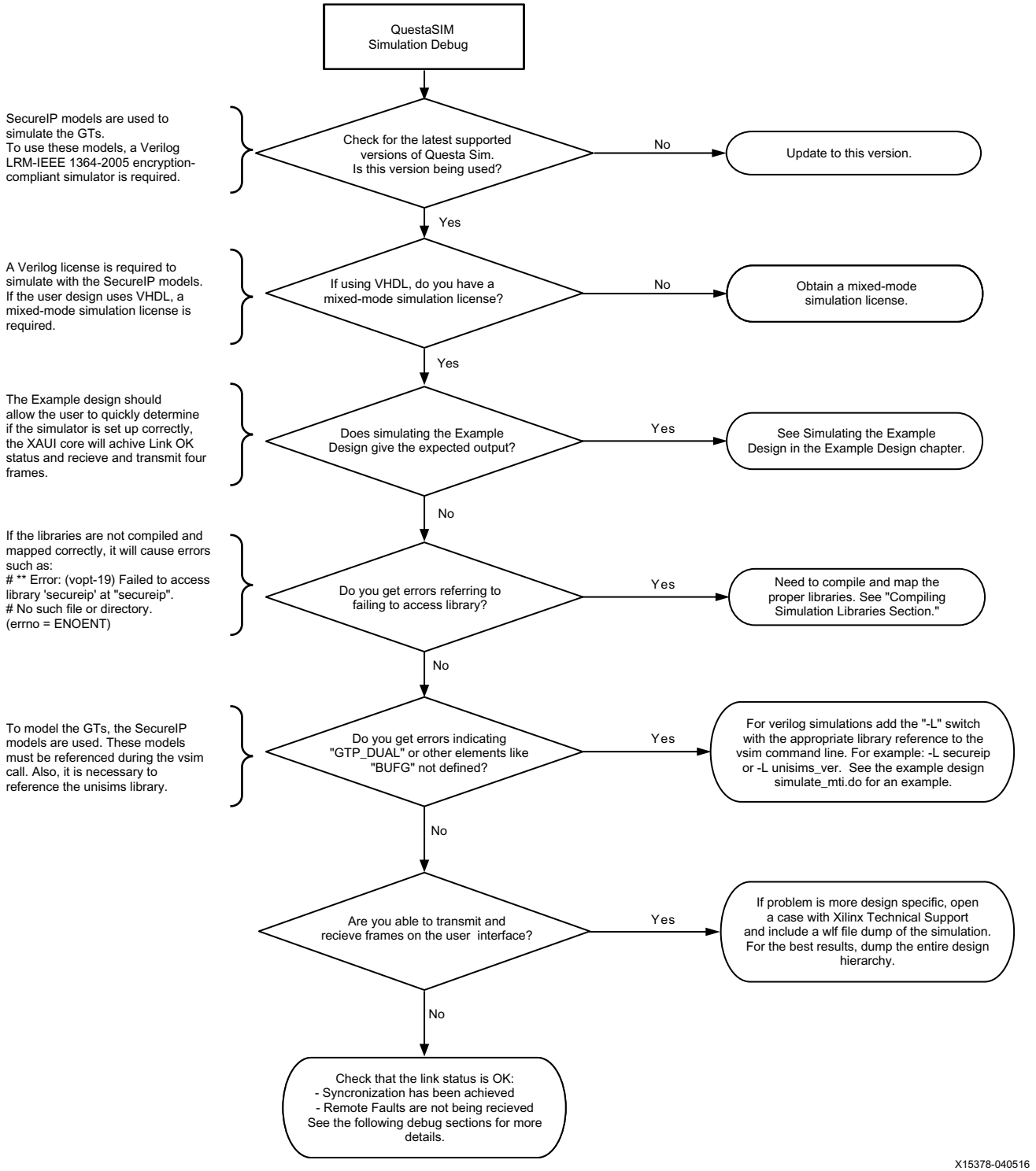

Figure B-1: **Questa SIM Simulation Debug Flow**

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the debug feature for debugging the specific problems.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using mixed-mode clock managers (MMCMs) in the design, ensure that all MMCMs have obtained lock by monitoring the LOCKED port.
- If your outputs go to 0, check your licensing.

Timing

Ensure that timing is met according to the Vivado tools before attempting to implement the IP in hardware.

Transceiver Specific Checks

- Ensure that the polarities of the txn/txp and rxn/rxp lines are not reversed. If they are, these can be fixed by using the `TXPOLARITY` and `RXPOLARITY` ports of the transceiver.
- Check that the transceiver is not being held in reset or still being initialized. The

RESETDONE outputs from the transceiver indicate when the transceiver is ready.

- Place the transceiver into parallel or serial near-end loopback.
- If correct operation is seen in the transceiver serial loopback, but not when loopback is performed through an optical cable, it might indicate a faulty optical module.
- If the core exhibits correct operation in the transceiver parallel loopback but not in serial loopback, this might indicate a transceiver issue.
- A mild form of bit error rate might be solved by adjusting the transmitter Pre-Emphasis and Differential Swing Control attributes of the transceiver.

Ethernet Specific Checks

A number of issues can commonly occur during the first hardware test of an 10G/25G Ethernet. These should be checked as indicated below.

It is assumed that the 10G/25G Ethernet has already passed all simulation testing which is being implemented in hardware. This is a pre-requisite for any kind of hardware debug.

The usual sequence of debugging is to proceed in the following sequence:

1. Clean up signal integrity.
2. Ensure that the SerDes achieves clock data recovery (CDR) lock.
3. Check that the 10/25G IP has achieved word sync.
4. Proceed to Interface and Protocol debug.

Signal Integrity

When bringing up a board for the first time and the 10/25G Ethernet does not seem to be achieving word sync, the most likely problem is related to signal integrity. Signal integrity issues must be addressed before any other debugging can take place.

Signal integrity should be debugged independently from the 10G/25G Ethernet. The following procedures should be carried out. (Note that it assumed that the PCB itself has been designed and manufactured in accordance with the required trace impedances and trace lengths, including the requirements for skew set out in the IEEE 802.3 specification.)

- Transceiver Settings
- Checking For Noise
- Bit Error Rate Testing

If assistance is required for transceiver and signal integrity debugging, contact Xilinx technical support.

N/P Swapping

If the positive and negative signals of a differential pair are swapped, then data cannot be correctly received on that lane. You should verify that the link has the correct polarity of each differential pair.

Clocking and Resets

Refer to the Clocking and Resets in Chapter 3 for these requirements.

Ensure that the clock frequencies for both the 10G/25G Ethernet as well as the Xilinx Transceiver reference clock match the configuration requested when the subsystem was ordered. The core clock has a minimum frequency associated with it. The maximum core clock frequency is determined by timing constraints. The minimum core clock frequency is derived from the required Ethernet bandwidth plus the margin reserved for clock tolerance, wander and jitter.

The first thing to verify during debugging is to ensure that resets remain asserted until the clock is stable. It must be frequency-stable as well as free from glitches before the 10G/25G Ethernet is taken out of reset. This applies to both the SerDes clock as well as the core clock.

If any subsequent instability is detected in a clock, the 10G/25G Ethernet must be reset. One example of such instability is a loss of CDR lock. The user logic should determine all external conditions which would require a reset (e.g. clock glitches, loss of CDR lock, power supply glitches, etc.).

Configuration changes cannot be made unless the subsystem is reset. An example of a configuration change would be setting a different maximum packet length. Check the description for the particular signal on the port list to determine if this requirement applies to the parameter that is being changed.

RX Debug

Consult the port list section for a description of the diagnostic signals which are available to debug the RX.

stat_rx_block_lock

This signal indicates that the receiver has detected and locked to the word boundaries as defined by a 01 or 10 control or data header. This is the first step to ensure that the 10/25G Ethernet IP is functioning normally.

CAUTION! *Under some conditions of no signal input, the SerDes receiver exhibits a steady pattern of alternating 1010101…. This can cause erroneous block lock, but still indicates that the receiver has detected the pattern.*

stat_rx_bad_fcs

A bad FCS indicates a bit error in the received packet. An FCS error could be due to any number of causes of packet corruption such as noise on the line.

stat_rx_local_fault

A local fault indication can be locally generated or received. Some causes of a local fault are:

- block lock not complete
- high bit error rate
- overflow or underflow

Loopback Check

If the Ethernet packets are being transmitted properly according to 802.3, there should not be RX errors. However, the signal integrity of the received signals must be verified first.

To aid in debug, a local loopback can be performed with the signal `ctl_local_loopback`. This connects the TX SerDes to the RX SerDes, effectively bypassing potential signal integrity problems. The transceiver is placed into "PMA loopback", which is fully described in the transceiver product guide. In this way, the received data can be checked against the transmitted packets to verify that the logic is operating properly

Protocol Interface Debug

To achieve error-free data transfers with the 10G/25G Ethernet, the 802.3 specification should be followed. Note that signal integrity should always be ensured before proceeding to the protocol debug.

Diagnostic Signals

There are many error indicators available to check for protocol violations. Carefully read the description of each one to see if it is useful for a particular debugging problem.

The following is a suggested debug sequence:

1. Ensure that Word sync has been achieved.
2. Make sure there are no descrambler state errors.
3. Eliminate CRC32 errors, if any.
4. Make sure the protocol is being followed correctly.
5. Ensure that there are no overflow or underflow conditions when packets are sent.

Statistics Counters

After error-free communication has been achieved, the statistics indicators can be monitored to ensure that traffic characteristics meet expectations. Note that some signals are strobes only, which means that the counters are not part of the subsystem. This is done so that the counter size can be customized. Counters are optionally available with the AXI interface.

Appendix C

Additional Resources and Legal Notices

Xilinx Resources

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

References

These documents provide supplemental material useful with this product guide:

1. 25G and 50G Ethernet Consortium Schedule 3 version 1.4 (August 28, 2014) (http://25gethernet.org/)
2. IEEE Standard 802.3-2012 (standards.ieee.org/findstds/standard/802.3-2012.html)
3. IEEE P802.3by/D01 (ieee802.org/3/by/)
4. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
5. *Vivado Design Suite User Guide: Designing with IP* (UG896)
6. *Vivado Design Suite User Guide: Getting Started* (UG910)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
9. *Vivado Design Suite User Guide - Implementation* (UG904)
10. *Vivado Design Suite AXI Reference Guide* (UG1037)
11. *25G IEEE 802.3by Reed-Solomon Forward Error Correction LogiCORE IP Product Guide* (PG217) -- Only available in a web lounge.
12. IEEE Standard 1588-2008, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems" (standards.ieee.org/findstds/standard/1588-2008.html)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/06/2016	1.2	• Added UltraScale+ support. • Added new section that has RSFEC, 1588 1-step and 2-step support. • Added new IEEE 1588 Timestamping section. • Added rx_preambleout [55:0] for both AXI4-Stream interfaces. • Added tx_preamblein [55:0] for AXI4-Stream interface. • Added registers to the Configuration, Status, and Counter register maps. • Changed custom preamble from in-band to out-of-band. • Added text about pm_tick and TIC_REG to Statistics Counter section • Changed polarity of the tx_axis_tuser and rx_axis_tuser signals. • Updated Figure 3-13 and Figure 3-14. • Removed VLane Adjust Mode from Table 4-2. • Removed LBUS material. • Added ctl_tx_ipg_value[3:0] to Table C-4.
12/02/2015	1.1	• Updated the performance and resource utilization data link.
11/18/2015	1.1	• Added a link to the performance and resource utilization data on the web. • Added the stat_rx_valid_ctrl_code, ctl_tx_custom_preamble_enable, and ctl_rx_custom_preamble_enable signal. • Updated the tx_axis_ tuser signal description. • Updated the Normal Transmission and Aborting a Transmission information in the Transmit AXI4-Stream Interface section. • Added Vivado IDE option details in Design Flow Steps chapter. • Added new information in Example Design chapter.
09/30/2015	1.0	Initial Xilinx release.

Please Read: Important Legal Notices