LatticeCORE

2.5 Gbps Ethernet MAC IP Core User's Guide

May 2014
IPUG98_1.1

Table of Contents

Introduction

This document provides technical information about the Lattice 2.5 Gbps Ethernet Media Access Controller (2.5GMAC) IP core.

The 2.5GMAC IP core supports the ability to transmit and receive data between a host processor and an Ethernet network. The main function of the Ethernet MAC is to ensure that the Media Access rules specified in the 802.3 IEEE standard are met while transmitting a frame of data over Ethernet. On the receiving side, the Ethernet MAC extracts the different components of a frame and transfers them to higher applications through the FIFO interface.

The 2.5GMAC IP core comes with the following documentation and files:

- Protected netlist/database
- Behavioral RTL simulation model
- Source files for instantiating and evaluating the core

Quick Facts

Table 1-1 gives quick facts about the 2.5GMAC IP core.

Table 1-1. 2.5GMAC IP Core Quick Facts

		2.5GMAC IP Configuration
Core Requirements	FPGA Families Supported	LatticeECP3™
	Minimal Device Needed	LFE3-17EA-8FTN256C
Resource Utilization	Data Path Width	16
	LUTs[2]	2100
	sysMEM EBRs[2]	1
	Registers[2]	1290
Design Tool Support	Lattice Implementation	Lattice Diamond® 1.3
	Synthesis	Synopsys® Synplify® Pro for Lattice E-2011.03L
		Mentor Graphics® Precision® RTL
	Simulation	Aldec® Active-HDL™ 8.2 Lattice Edition (Verilog and VHDL)
		Mentor Graphics® ModelSim® 6.3f SE (Verilog only)
		Cadence® NC-Verilog® (Linux only)

Features

- Compliant to IEEE 802.3-2005 standard
- Generic 8-bit host interface
- 16-bit wide internal data path
- Generic transmit and receive FIFO interface
- Full-duplex operation
- Transmit and receive statistics vector
- Programmable Inter-Packet Gap (IPG)
- Multicast address filtering

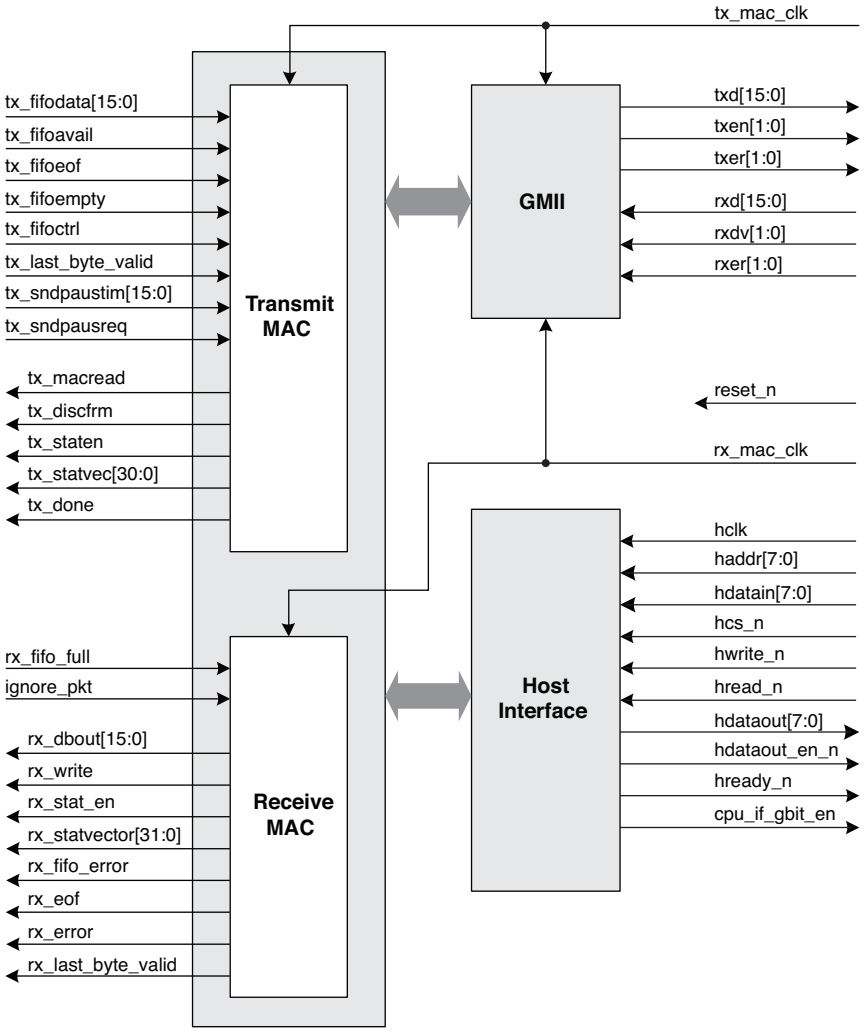
- Supports:
  - Full-duplex control using PAUSE frames
  - VLAN tagged frames
  - Automatic padding of short frames
  - Multicast and Broadcast frames
  - Optional FCS transmission and reception
  - Jumbo frames of any length

The 2.5GMAC IP core is a fully synchronous machine composed of Transmit and Receive MAC sections that operate independently to support full duplex operation.

The block diagram of the 2.5GMAC IP core is shown in Figure 2-1. The major functional modules are:

- Host Interface
- Receive MAC
- Transmit MAC
- Internal Buffers and FIFO Interfaces
- GMII

Figure 2-1. Core Block Diagram



Functional Overview

The 2.5GMAC IP core transmits and receives data between a client application and an Ethernet network. The main function of the Ethernet MAC is to ensure that the Media Access rules specified in the 802.3 IEEE standard are met while transmitting and receiving Ethernet frames. Figure 2-2, Figure 2-3, and Figure 2-4 show some of the frame formats of data transmitted and received on the Ethernet network that the 2.5GMAC IP core supports.

On the receiving side, the Ethernet MAC extracts the different components of a frame and transfers them to higher applications through the client FIFO interface.

The data received from the GMII interface is first buffered until sufficient data is available to be processed by the Receive MAC (Rx MAC). The Preamble and the Start-of-Frame Delimiter (SFD) information are then extracted from the incoming frame to determine the start of a valid frame. The Receive MAC checks the address of the received packet and validates whether the frame can be received before transferring it into the FIFO. Only valid frames are transferred into the FIFO (runts and fragments are discarded). The Rx MAC also provides a statistics vector on a per packet basis that can be used by the application. The 2.5GMAC IP core always calculates CRC to check whether the frame was received error-free.

On the transmit side, the Tx MAC is responsible for controlling access to the physical medium. The Tx MAC reads data from an external client Tx FIFO, formats this data into an Ethernet packet and passes it to the GMII module.

The Tx MAC reads data from the Tx Client FIFO when the client indicates a packet is available, and the Tx MAC is in its appropriate state. The Tx MAC pre-fixes the Preamble and the Start-of-Frame Delimiter information to the data and appends the Frame Check Sequence at the end of the data.

Figure 2-2. Un-Tagged Ethernet Frame Format

PREAMBLE	SFD	DESTINATION ADDRESS	SOURCE ADDRESS	LENGTH/ TYPE	DATA/PAD	FRAME CHECK SEQUENCE
7 bytes	1 byte	6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes

Figure 2-3. VLAN-Tagged Ethernet Frame Format

PREAMBLE	SFD	DESTINATION ADDRESS	SOURCE ADDRESS	VLAN TAG HEADER	LENGTH/ TYPE	DATA/PAD	FRAME CHECK SEQUENCE
7 bytes	1 byte	6 bytes	6 bytes	4 bytes	2 bytes	46-1500 bytes	4 bytes

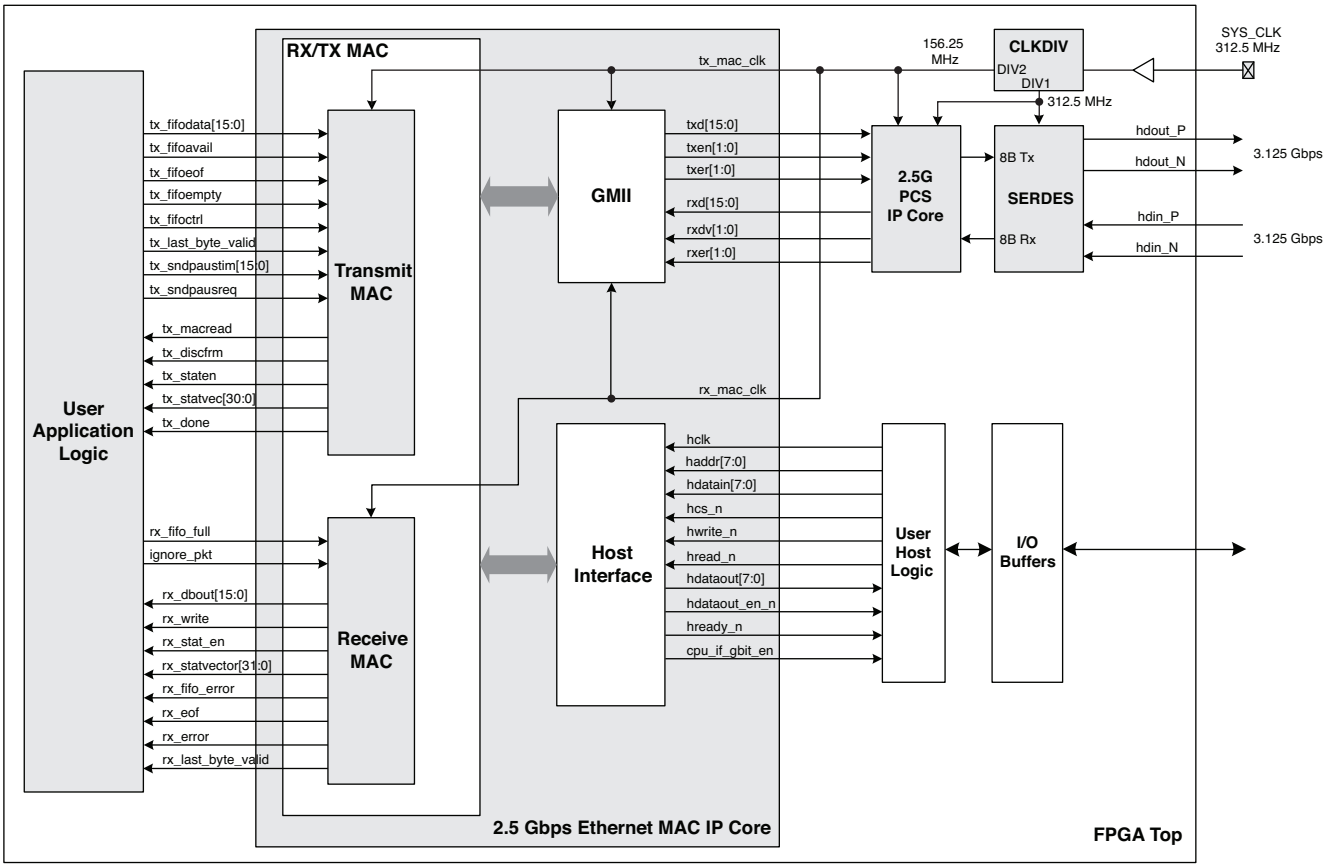
Figure 2-4. Ethernet Control Pause Frame Format

PREAMBLE	SFD	DESTINATION ADDRESS 01-80-C2-00-00-01	SOURCE ADDRESS	LENGTH/TYPE 88-08	MAC CTL OP_CODE 00-01	OP_CODE PARAMS/RSV	FRAME CHECK SEQUENCE
7 bytes	1 byte	6 bytes	6 bytes	2 bytes	2 bytes	60 bytes	4 bytes

A Tagged frame includes a 4-byte VLAN Tag field, which is located between the Source Address field and the Length/Type field. The VLAN Tag field includes the VLAN Identifier and other control information needed when operating with Virtual Bridged LANs as described in IEEE P802.1Q.

Note that the 2.5GMAC's GMII interface utlizes a 16-bit data interface. Although this interface conforms to the timing specified in IEEE 802.3 Clause 35, the data bus width and clock frequency do not conform to the IEEE standard. Therefore, it may be difficult to find external PHY devices that directly connect. To overcome this issue, Lattice provides a 2.5G PCS IP core with a 16-bit data path and 156.25 Mhz clock that is directly compatible with the 2.5GMAC GMII interface. Figure 2-5 shows how the Lattice 2.5GMAC and Lattice 2.5G PCS can be combined to provide a full Ethernet client interface to Ethernet physical layer interface for 2.5Gbps applications.

Figure 2-5. System Block Diagram



Core Signal Descriptions

Table 2-1 lists the I/O signals for the 2.5GMAC IP core.

Table 2-1. 2.5GMAC IP Core Input and Output Signals

Port Name	Type	Active State	Description
Clocks and Reset/Other			
rxmac_clk	Input	N/A	**Receive MAC Application Interface Clock**. This clock is used by the client application and MAC. All outputs driven by the Rx MAC on the client side are synchronous to this clock. *Note: this clock can be viewed as a "word" clock, since all Rx MAC 16-bit words are aligned with this clock. This clock is derived from the system sys_clk (156.25 MHz for 2.5 Gbps operation).*
txmac_clk	Input	N/A	**Transmit MAC Application Interface Clock**. This clock is used by the client application and MAC. All inputs to the Tx MAC on the client side should be synchronous to this clock. *Note: this clock can be viewed as a "word" clock, since all Tx MAC 16-bit words should be aligned with this clock. This clock is derived from the system sys_clk (156.25 MHz for 2.5 Gbps operation).*
hclk	Input	N/A	**Host Clock**. This is the Host Bus clock, and is used to clock the Host Bus interface.
reset_n	Input	Low	**Reset**. This is an active low asynchronous signal that resets the internal registers and internal logic. When activated, the I/O signals are driven to their inactive levels.
cpu_if_gbit_en	Output	High	**CPU Interface 1G Mode Enabled Indication**. This signal is always high in the 2.5GMAC IP core. It exists for backward compatibility to 1 Gbps MAC.
Host Interface			
hcs_n	Input	Low	**Chip Select**. This is an active low signal used to select the core for register read/write operations.
haddr[7:0]	Input	N/A	**Address**. This selects one of the internal core registers.
hdatain[7:0]	Input	N/A	**Data Bus Input**. The CPU writes to the internal registers through the data bus.
hwrite_n	Input	Low	**Host Write**. This active low signal is used to write data to the selected register.
hread_n	Input	Low	**Host Read**. This active low signal is used to read data from the selected register.
hready_n	Output	Low	**Ready**. This is an active low signal used to indicate the end of transfer. For write operations, hready_n is asserted after data is accepted (written). For read operations hready_n is asserted after data on the hdataout bus is ready to be driven out.
hdataout_en_n	Output	Low	**Data Out Enable**. This signal is driven low whenever the 2.5GMAC IP core outputs valid data onto the hdataout bus. This signal can be used to build a bi-directional data bus.
hdataout[7:0]	Output	N/A	**Data Bus Output**. The CPU reads the internal registers through the data bus.
Transmit MAC Application Interface			
tx_fifodata[15:0]	Input	N/A	**Transmit FIFO Read Data Bus**. The data from the FIFO is presented on this bus. When the Ethernet frame reaches the physical layer, the upper byte is transmitted first (bits 15:8). Also, for each byte, the least significant bit is transmitted first at the physical layer (bit D8 for the upper byte; bit D0 for the lower byte).
tx_fifoavail	Input	High	**Transmit FIFO Data Available**. When asserted, this signal indicates that the TxFIFO has data ready for transmission on the GMII interface. Once this signal is asserted by the client, a short delay later the frame will be transmitted. The client needs to use an appropriate threshold on the client FIFO to indicate that a frame is ready to be sent and use that threshold as the tx_fifo_avail signal.
tx_fifoeof	Input	High	**Transmit FIFO End of Frame**. This signal is asserted along with the last byte of frame data indicating the end of the frame.
tx_fifoempty	Input	High	**Transmit FIFO Empty**. This signal indicates that the TxFIFO is empty. When this signal is asserted and the 2.5GMAC IP core is reading the FIFO, the under-run condition is transferred to the network through the txer signal.

Table 2-1. 2.5GMAC IP Core Input and Output Signals (Continued)

Port Name	Type	Active State	Description
tx_sndpaustim[15:0]	Input	N/A	**PAUSE Frame Timer**. This signal indicates the PAUSE time value that should be sent in the PAUSE frame.
tx_fifoctrl	Input	N/A	**FIFO Control Frame**. This signal indicates whether the current frame in the Tx FIFO is a control frame or a data frame. It is qualified by the tx_fifoavail signal. The following values apply: • 1 = Control frame • 0 = Normal frame
tx_staten	Output	High	**Transmit Statistics Vector Enable**. When asserted, the contents of the statistics vector bus tx_statvec are valid.
tx_last_byte_valid	Input	High	**Transmit Last Byte Valid**. When tx_fifoeof asserts, the state of tx_last_byte_valid indicates whether or not the last 16-bit word on tx_fifodata bus has one or two bytes that are valid. If high, both bytes are valid. If low, only the most significant byte is valid and the least significant byte is ignored.
tx_macread	Output	High	**Transmit FIFO Read**. This is the 2.5GMAC IP core Tx FIFO read request, asserted by the 2.5GMAC IP core when it intends to read the client FIFO. The core will first assert the tx_macread signal if the client FIFO is not empty (i.e., tx_fifoempty = 0), after which the tx_macread may de-assert (based on MAC processing) or re-assert (based on MAC processing and if tx_fifoempty is still 0). The tx_macread signal should be tied to the client FIFO read pin, and the FIFO empty pin should be tied to the tx_fifoempty of the MAC core.
tx_statvec[30:0]	Output	N/A	**Transmit Statistics Vector**. This bus includes useful information about the frame that was just transmitted. The corresponding bit locations of this bus are defined as follows: • tx_statvec[0] – UNICAST frame • tx_statvec[1] – Multicast frame • tx_statvec[2] – BROACAST frame tx_statvec[3] - Bad FCS frame • tx_statvec[4] – JUMBO frame • tx_statvec[5] – FIFO under-run • tx_statvec[6] – PAUSE frame • tx_statvec[7] – VLAN tagged frame • tx_statvec[21:8] – Number of bytes in the transmitted frame • tx_statvec[22] – Not used • tx_statvec[23] – Not used • tx_statvec[24] – Not used • tx_statvec[25] – Not used • tx_statvec[29:26] – Not used • tx_statvec[30] – FCS generation is disabled and a short frame was transmitted
tx_done	Output	High	**Transmit Done**. This signal is asserted for one clock cycle after transmitting a frame if no errors were present in transmission.
tx_discfrm	Output	High	**Discard Frame**. This signal is asserted at the end of a frame transmit process if the 2.5GMAC IP core detected a FIFO under-run. The user application normally moves the pointer to next frame in this condition.
GMII Signals			
txd[15:0]	Output	N/A	**Transmit Data** sent to the SERDES/PCS core. When the Ethernet frame reaches the physical layer, the upper byte is transmitted first (bits 15:8). Also, for each byte, the least significant bit is transmitted first at the physical layer (bit D8 for the upper byte; bit D0 for the lower byte).
txen[1:0]	Output	High	**Transmit Enable**. Indicates that the bytes within the transmit data are valid. The most significant bit indicates that the most significant byte of txd is valid and similarly for the least significant bit and byte.
txer[1:0]	Output	High	**Transmit Error**. Indicates that the bytes within the transmit data have an error. The most significant bit indicates that the most significant byte of txd is in error and similarly for the least significant bit and byte.

Table 2-1. 2.5GMAC IP Core Input and Output Signals (Continued)

Port Name	Type	Active State	Description
rxd[15:0]	Input	High	**Receive Data Bus**. These GMII Rx data inputs (bytes valid whenever respective rxdv bit is asserted) come from the SERDES/PCS or from a GMII PHY interface. At the physical layer, the upper byte arrives first (bits 15:8). Also, for each byte, the least significant bit arrives first at the physical layer (bit D8 for the upper byte; bit D0 for the lower byte).
rxdv[1:0]	Input	High	**Receive Data Valid**. Indicates the data bytes on the rxd bus are valid. The most significant bit indicates that the most significant byte of rxd is valid and similarly for the least significant bit and byte.
rxer[1:0]	Input	High	**Receive Data Error**. Indicates that the bytes within the receive data have an error. The most significant bit indicates that the most significant byte of rxd is in error and similarly for the least significant bit and byte.
Receive MAC Application Interface			
rx_fifo_full	Input	High	**Receive FIFO Full**. This signal indicates the Rx FIFO is full and cannot accept any more data. This is an error condition and should never happen.
rx_write	Output	High	**Receive FIFO Write**. This signal is asserted by the 2.5GMAC IP core to request a FIFO write.
rx_dbout[15:0]	Output	N/A	**Receive FIFO Data Output**. This bus contains the data that is to be written into the Rx FIFO. At the physical layer, the upper byte arrives first (bits 15:8). Also, for each byte, the least significant bit arrives first at the physical layer (bit D8 for the upper byte; bit D0 for the lower byte).
rx_stat_vector[31:0]	Output	N/A	**Receive Statistics Vector**. This bus indicates the events encountered during frame reception. This bus is qualified by the rx_stat_en signal. The definition of each signal is explained in "Receive MAC (Rx MAC)" on page 12 of this user's guide. The corresponding bit locations of this bus are defined as follows: • rx_statvec[15:0] – Frame Byte Count • rx_statvec[16] – VLAN Tag Detected • rx_statvec[17] – Pause Frame • rx_statvec[18] – Control Frame • tx_statvec[19] – Unsupported Opcode • rx_statvec[20] – Unused • rx_statvec[21] – Broadcast Address • rx_statvec[22] – Multicast Address • rx_statvec[23] – Receive OK • rx_statvec[24] – Length Check Error • rx_statvec[25] – CRC Error • rx_statvec[26] – Packet Ignored • rx_statvec[27] – Unused • rx_statvec[28] – Unused • rx_statvec[29] – IPG Violation • rx_statvec[30] – Short Frame • rx_statvec[31] – Long Frame
rx_stat_en	Output	High	**Receive Statistics Vector Enable**. When asserted, this signal indicates that the contents of the rx_stat_vector bus is valid.
ignore_next_pkt	Input	High	**Ignore Next Packet**. This signal is asserted by the host to prevent a Rx FIFO Full condition. The Rx MAC continues dropping packets as long as this signal is asserted. This is an asynchronous signal.
rx_eof	Output	High	**End Of Frame**. Indicates the last word of the current packet is on the rx_dbout data bus.
rx_last_byte_valid	Output	High	**Receive Last Byte Valid**. When tx_eof asserts, the state of rx_last_byte_valid indicates whether or not the last 16-bit word on rx_ dbout bus has one or two bytes that are valid. If high, both bytes are valid. If low, only the most significant byte is valid and the least significant byte is ignored.

Table 2-1. 2.5GMAC IP Core Input and Output Signals (Continued)

Port Name	Type	Active State	Description
rx_error	Output	High	**Receive Packet Error**. When asserted, this signal indicates the packet contains error(s). This signal is qualified with the rx_eof signal. The rx_error signal will be asserted for any of the following conditions: • The rxer signal on the GMII is asserted by the PHY during frame reception. • There are Rx FCS errors on received frames. • There is a length check error on the received frame.
rx_fifo_error	Output	High	**Receive FIFO Error**. This signal is asserted when the external Rx FIFO is full (rx_fifo_full signal asserted) and the Rx FIFO is being written to by the Rx MAC (rx_write is asserted). When this error signal is asserted the rx_write signal will be de-asserted as long as the rx_fifo_error signal is asserted. The rx_fifo_error signal will be de-asserted when the end of packet (rx_eof) exits the receive FIFO (*Note: In this errored condition, data will continue to be pulled out of the Rx FIFO, even while the rx_write signal has been de-asserted*).

Host Interface

The Host Interface module is a fully synchronous module that runs off the host clock. A number of registers are initialized via the Host interface to ensure that the 2.5GMAC IP core functions as intended. The write operation to an internal register is initiated when the hcs_n and hwrite_n signals are asserted. The address of the targeted register is placed on the haddr bus, while the valid data is placed on the hdatain bus. The contents of the address and data busses should remain unchanged until the 2.5GMAC IP core asserts the hready_n signal. The signals hcs_n, hwrite_n and hread_n must remain unchanged until hready_n is asserted.

A register read is initiated by asserting the hcs_n and hread_n signals, while keeping the hwrite_n signal deasserted. The address of the targeted register is placed on the haddr bus. The 2.5GMAC IP core places the content of the targeted register on the hdataout bus and qualifies it with the assertion of hready_n signal. The haddr bus should not change until the hready_n signal is asserted.

Figure 2-12 shows the timing diagram associated with the host interface write and read operations.

Receive MAC (Rx MAC)

The main function of the Rx MAC is to accept the formatted data from the GMII interface and pass it to the host application through an external FIFO. In this process, the Rx MAC performs the following functions:

- Detect the start of frame
- Compare the MAC address
- Re-calculate CRC
- Process the control frame and pass it to the flow control module.

The Rx MAC operation is determined by programming the MODE and TX_RX_CTL registers. These register definitions and bit descriptions can be found in Table 2-3 on page 16.

Programming the MODE and TX_RX_CTL registers can control the Receive MAC operation. The various events that occur during the reception of a frame are logged into the rx_stat_vector signal and the TX_RX_STS register. At the end of reception, the rx_stat_en signal is asserted to qualify the rx_stat_vector signal. The 2.5GMAC IP core can report a wealth of information such as

- FIFO overflow
- CRC error
- Receive error
- Short frame reception

- Long frame reception
- IPG violation

By default, the entire frame, except the preamble and SFD bytes, is sent to the FIFO via the Rx MAC application interface signals. If the user does not want to receive the FCS, the core can be programmed to strip the FCS field as well as any PAD bytes in the frame and send the rest to the FIFO.

Receiving Frames

The frames received by the Rx MAC are analyzed and the Preamble and SFD bytes are stripped off the frame before it is transferred to an external FIFO. The client data interface between the MAC and the FIFO is 16 bits wide.

The default behavior of the MAC is to transfer the unmodified frame after stripping off the Preamble and SFD bytes. This behavior can be changed by setting bit [1] of the TX_RX_CTL register. When bit [1] is set, the Rx MAC strips the Preamble, SFD, FCS bytes and the PAD bytes, if any. Note that the Rx MAC assumes a received frame has PAD bytes if a 64 byte packet is received with its Length/Type field set to a value of less than 46 bytes.

Once the frame is ready to be written into the FIFO, the Rx MAC asserts the rx_write signal, then presents the data on the rx_dbout bus. The rx_write signal is asserted as long as the frame is being written. After transferring the entire frame into the FIFO, the Rx MAC asserts rx_eof indicating the end of the frame and sets the state of rx_last_byte_valid to indicate if one or both of the bytes in the last word are valid. If the frame is received with errors, rx_error is asserted along with rx_eof. If the frame is received with no errors, rx_error remains de-asserted. In either case, a rich set of statistics vectors is presented, containing information about the frame that was received. The statistics vector bus, rx_stat_vector, is qualified by the assertion of rx_stat_en.

If the RxFIFO becomes full, rx_fifo_full is asserted and the frame data is lost. Therefore, the FIFO full condition must be avoided at all times. The rx_fifo_error signal will be asserted along with rx_eof for all frames written into the FIFO while it is full.

The Rx MAC goes to the IDLE state when it is done receiving the frame. This is indicated by bit[10] of the TX_RX_STS register. If the Rx MAC is disabled while it is in the process of receiving a frame, it goes to the IDLE state after it completes the current frame reception.

Address Filtering

The Rx MAC offers several address filtering methods the user can employ to effectively block unwanted frames. It also provides a Promiscuous mode, in which all supported filtering schemes are abandoned and the Rx MAC receives all the frames irrespective of the address they contain.

By default, the Rx MAC is configured to filter and discard Broadcast frames (i.e. all bits of the received DA == 1) and multicast frames (i.e. bit[0] of the received DA == 1). The MAC can be configured to receive broadcast frames by setting bit [7] of the TX_RX_CTL register.

Multicast frames are received only when bit [4] of the TX_RX_CTL register is set. When set, multicast frames are subject to filtering that is dependent on a 64-bit hash table lookup. The 64-bit hash table is organized as eight, 8-bit registers. The six middle bits of the most significant byte of the CRC calculated for the destination address field of the frame, are used to address one of the 64 bits of the hash table. The three most significant bits of the calculated CRC select one of the eight tables, and the three least significant bits select a bit. The frame is received only if the retrieved bit is set. The IP core registers specifying the hash tables contents are described in “Internal Registers” on page 16. An example of C programming language code that can be used to determine hash table contents based on the multicast addresses to be received is given in “Code Listing for Multicast Bit Selection Hash Algorithm in C Language” on page 42.

All other regular frames are filtered based on the Rx MAC address programmed into the MAC_ADDR_0, MAC_ADDR_1 and MAC_ADDR_2 registers.

Filtering Based on Frame Length

The default minimum Ethernet frame size is 64 bytes. By default, the Rx MAC is configured to ignore frames shorter than 64 bytes. The user can configure the MAC to receive shorter frames by setting bit [8] of the TX_RX_CTL register. Whenever a short frame is received, the appropriate bit is set in the statistics vector, marking it as a Short frame.

The Rx MAC has been designed to receive frames larger than the standard specified maximum as easily as any other frame. This ensures the MAC can work in environments that can generate jumbo frames. However, for statistics purposes, the user can set the maximum length of the frame in the MAX_PKT_SIZE register. When a received frame is larger than the number in this register, bit [31] of the Receive Statistics Vector bus is set, marking it as a Long frame.

Receiving a PAUSE Frame

When the Rx MAC receives a PAUSE frame, the Tx MAC continues with the current transmission, then pauses for the duration indicated in the PAUSE time. During this time, the Tx MAC can transmit Control frames.

Although PAUSE frames may contain the Multicast Address, Multicast filtering rules do not apply to them. If bit [3] of the TX_RX_CTL register is set, the Rx MAC will signal the Tx MAC to stop transmitting for the duration specified in the frame. If this bit is reset, the Rx MAC assumes the Tx MAC does not have the PAUSE capability and/or does not wish to be paused and will not signal it to stop transmitting. If the drop control, bit[6] in the TX_RX_CTL register, is set then the PAUSE frame is received but dropped internal to the MAC and is not transferred to the client FIFO interface. Otherwise, the PAUSE frame is received and transferred to the FIFO.

Statistics Vector

By default, a Statistics Vector is generated for all received frames transferred to the external FIFO. If the user wants the Rx MAC to ignore all incoming frames, then the input signal ignore_next_pkt must be asserted. In this case, a frame that should have been received is ignored and the Rx MAC sets the Packet Ignored bit (bit 26) of the Statistics Vector.

The MAX_PKT_SIZE register is programmed by the user as a threshold for setting the Long Frame bit of the Statistics Vector. This value is used for un-tagged frames only. The Receive MAC will add “4” to the value specified in this register for all VLAN tagged frames when checking against the number of bytes received in the frame. This is because all VLAN tagged frames have an additional four bytes of data.

When a tagged frame is received, the entire VLAN tag field is stored in the VLAN_TAG register. Additionally, every time a statistics vector is generated, some of the bits are written into the corresponding bit locations [9:1] of the TX_RX_STS register. This is done so the user can get this information via the Host interface.

The description of the bits in the Statistics Vector bus is shown in Table 2-2.

Table 2-2. Receive Statistics Vector Descriptions

Bit	Description
31	**Long Frame**. This bit is set when a frame longer than specified in the **MAX_PKT_SIZE** register is received.
30	**Short Frame**. This bit is set when a frame shorter than 64 bytes is received.
29	**IPG Violation**. This bit is set when a frame is received before the IPG timer runs out (96 bit times).
28	**Not Used**. This bit always returns a zero.
27	**Not Used**. This bit always returns a zero.
26	**Packet Ignored**. When set, this bit indicates the incoming packet is to be ignored.
25	**CRC Error**. This bit is set when a frame is received with an error in the CRC field.
24	**Length Check Error**. This bit is set if the number of data bytes in the incoming frame do not match the value in the length field of the frame.
23	**Receive OK**. This bit is set if the frame is received without any error.
22	**Multicast Address**. This bit is set to indicate the received frame contains a Multicast Address.
21	**Broadcast Address**. This bit is set to indicate the received frame contains a Broadcast Address.

Table 2-2. Receive Statistics Vector Descriptions

Bit	Description
20	**Not Used**. This bit always returns a zero.
19	**Unsupported Opcode**. This bit is set if the received control frame has an unsupported opcode. In this version of the IP, only the opcode for PAUSE frame is supported.
18	**Control Frame**. This bit is set to indicate that a Control frame was received.
17	**PAUSE Frame**. This bit is set when the received Control frame contains a valid PAUSE opcode.
16	**VLAN Tag Detected**. This bit is set when the 2.5GMAC IP core receives a VLAN Tagged frame.
15:0	**Frame Byte Count**. This bus contains the length of the frame that was received. The frame length includes the DA, SA, L/T, TAG, DATA, PAD and FCS fields.

Transmit MAC (Tx MAC)

The Tx MAC is responsible for controlling access to the physical medium. The Tx MAC reads data from an external Tx FIFO when the FIFO is not empty and it detects an active tx_fifoavail. The Tx MAC then formats this data into an Ethernet packet and passes it to the GMII module.

The Tx MAC is disabled while Tx_en is low (Bit_3 of the MODE register) and should only be enabled after the associated registers are properly initialized. Once enabled, the Tx MAC will continuously monitor the FIFO interface for an indication that frame(s) are ready to be transmitted. Tx MAC and the Tx FIFO interface operations are synchronous to txmac_clk derived from sys_clk.

In full-duplex operation, it is possible for the receiver's buffer to fill up rapidly. In such cases, the receiver sends flow control (PAUSE) frames to the transmitter, requesting that it stop transmitting frames. When the receiver is able to free the buffers, the transmitter completes transmitting the current frame and stops for the duration specified in the PAUSE frame.

Transmitting Frames

By default, the Transmit MAC is configured to generate the FCS pattern for the frame to be transmitted. However, this can be prevented by setting bit[2] of the Tx_RX_CTL register. This feature is useful if the frames being presented for transmission already contain the FCS field. When FCS field generation by the MAC is disabled, it is the user's responsibility to ensure that short frames are properly padded before the FCS is generated. If the MAC receives a frame to transmit that is shorter than 64 bytes when FCS generation is disabled, the frame is sent as is and a statistic vector for the condition is generated.

The DA, SA, L/T, and DATA fields are derived from higher applications through the FIFO interface and then encapsulated into an Un-tagged Ethernet frame. This frame is not sent over the network until the network has been idle for a minimum of Inter-Packet Gap (IPG) time. The Frame encapsulation consists of adding the Preamble bits, the Start of Frame Data (SFD) bits and the CRC check sum to the end of the frame (FCS). If padding is not disabled, all short frames are padded with hexadecimal 00.

The input signal tx_fifoeof is asserted along with the last set of data transfer to indicate the end of the frame. The tx_last_byte_valid signal is asserted or not on the last word of the frame (tx_fifoeof asserted) to indicate whether one or both bytes of the last word are valid. If not asserted, only the most significant byte is valid. The Tx MAC requires a continuous stream of data for the entire frame. There cannot be any bubbles of "no data transfer" within a frame. If the MAC is able to transmit the frame without any errors, the tx_done signal is asserted. Once the transmission has ended, data on the tx_stat_vector bus is presented to the host, including all the statistical information collected in the process of transmitting the frame. Data on this bus is qualified by assertion of the tx_staten signal.

After the Transmit MAC is done transmitting a frame, it waits for more frames from the FIFO interface. During this time, it goes to an idle state that can be detected by reading the TX_RX_STS register. Since the MODE register can be written at any time, the Tx MAC can be disabled while it is actively transmitting a frame. In such cases, the MAC will completely transmit the current frame and then return to the idle state. The control registers should be programmed only after the MAC has returned to the IDLE state.

External Transmit FIFO

The interface between the Tx MAC and the external, client side FIFO is 16 bits wide. When the Ethernet frame reaches the physical layer, the upper byte is transmitted first (bits 15:8). Also within each byte, the least significant bit is transmitted first at the physical layer (bit D8 for the upper byte; bit D0 for the lower byte).

Logic inside the MAC signals if the frame ready for transmission at the head of the FIFO is a Control frame. This is done so the Tx MAC can continue transmission of a Control frame while it is paused.

FIFO Under-flow

If a FIFO underflow occurs, the FIFO logic must assert tx_fifoempty. If at least 64 bytes have been transmitted, the Tx MAC aborts the transmission by asserting tx_er. In addition, the Tx MAC inserts erroneous CRC bits into the packet to guarantee the receiver will detect the error in the packet. If less than 64 bytes have been transmitted when the FIFO underflow occurs, the MAC will pad the remaining bytes before ending the transmission. In either case, the MAC asserts tx_discfrm indicating an error during transmission.

Transmitting PAUSE Frame

Two different methods are used for transmitting a PAUSE frame. In the first method, the application layer forms a PAUSE frame and submits it for transmission via the FIFO. In the other method, the application layer signals the Tx MAC directly to transmit a PAUSE frame. This is accomplished by asserting tx_sndpausreq. In this case the Tx MAC will complete transmission of the current packet and then transmit a PAUSE frame with the PAUSE time value supplied through the tx_sndpaustim bus.

Internal Data Buffer and FIFO Interfaces

On the receive side, an internal FIFO buffer is used to support the dropping of packets less than 64 bytes and to provide additional data buffering for normal packets. The core provides a feature where the user can block all the frames that are shorter than the minimum frame length of 64 bytes in the 2.5GMAC IP core itself (for example collision fragments, runt frames, and such). This prevents these frames from reaching the user's application.

The 2.5GMAC IP core provides two independent interfaces for use with external Transmit and Receive FIFOs. This feature enables the 2.5GMAC IP core to support full duplex operation.

Internal Registers

The 2.5GMAC IP core internal registers are initialized through the generic Host Interface. These rules apply when accessing the internal registers:

- With the 8-bit Host Interface, the individual bytes of the registers are accessed through their corresponding addresses, with the lower address pointing to the lower byte.
- The reserved bits should be programmed to 0. These bits are invalid, and should be discarded when read.
- All registers except the MODE register can be written into only when the core is disabled, i.e., MAC is in the IDLE state (Tx_en and Rx_en low in the MODE register). The MODE register is the only register that can be written to after the 2.5GMAC IP core is no longer disabled.

Table 2-3 lists the 2.5GMAC IP core registers accessible via the Host Interface. The registers are either Read/Write (R/W) or Read Only (RO) for status reporting purposes. The values of the registers immediately after the Reset Condition is removed from the 2.5GMAC IP core (POR Value in Hexadecimal format) are also given.

Table 2-3. 2.5GMAC IP Core Internal Registers

Register Description	Mnemonic	I/O Address	POR Value
Mode register	MODE	00H - 01H	0000H
Transmit and Receive Control register	TX_RX_CTL	02H - 03H	0000H
Maximum Packet Size register	MAX_PKT_SIZE	04H - 05H	05EEH
Inter-Packet Gap register	IPG_VAL	08H - 09H	000CH
2.5GMAC IP core Address register 0	MAC_ADDR_0	0AH - 0BH	0000H

Table 2-3. 2.5GMAC IP Core Internal Registers (Continued)

Register Description	Mnemonic	I/O Address	POR Value
2.5GMAC IP core Address register 1	MAC_ADDR_1	0CH - 0DH	0000H
2.5GMAC IP core Address register 2	MAC_ADDR_2	0EH - 0FH	0000H
Transmit and Receive Status	TX_RX_STS	12H - 13H	0000H
Reserved		14H - 15H	0000H
Reserved		16H - 17H	0000H
VLAN Tag Length/type register	VLAN_TAG	32H - 33H	0000H
Multicast_table_0	MLT_TAB_0	22H - 23H	0000H
Multicast_table_1	MLT_TAB_1	24H - 25H	0000H
Multicast_table_2	MLT_TAB_2	26H - 27H	0000H
Multicast_table_3	MLT_TAB_3	28H - 29H	0000H
Multicast_table_4	MLT_TAB_4	2AH - 2BH	0000H
Multicast_table_5	MLT_TAB_5	2CH - 2DH	0000H
Multicast_table_6	MLT_TAB_6	2EH - 2FH	0000H
Multicast_table_7	MLT_TAB_7	30H - 31H	0000H
Pause_opcode	PAUS_OP	34H - 35H	0080H

Register Descriptions

MODE (R/W)

Mnemonic: MODE
POR Value = 0001H

Name	Range	Description
Rsvd	15:4	**Reserved**.
Tx_en	3	**Transmit Enable**. When this bit is set, the Tx MAC is enabled to transmit frames. When reset, the Tx MAC completes transmission of the packet currently being processed, then stops.
Rx_en	2	**Receive Enable**. When this bit is set, the Rx MAC is enabled to receive frames. When reset, the Rx MAC completes reception of the packet currently being processed, then stops.
FC_en	1	**Flow-control Enable**. When set, this bit enables the flow control functionality of the Tx MAC. This bit should be set to enable the Tx MAC to transmit a PAUSE frame via the tx_sndpausreq and tx_sndpaustim[15:0] MAC input ports.
Gbit_en	1	**Read Only**. Always set to '1'.

Transmit and Receive Control (R/W)

Mnemonic: TX_RX_CTL
POR Value = 0000H

This register can be overwritten only when the Rx MAC and the Tx MAC are disabled. This register controls the various features of the MAC.

Name	Range	Description
Rsvd	15:9	**Reserved**.
Receive_short	8	**Receive Short Frames**. When high, enables the Rx MAC to receive frames shorter than 64 bytes.
Receive_brdcst	7	**Receive Broadcast**. When high, enables the Rx MAC to receive broadcast frames
Rsvd	6	**Reserved**.
Rsvd	5	**Reserved**.
Receive_mltcst	4	**Receive Multicast**. When high, the multicast frames will be received per the filtering rules for such frames. When low, no Multicast (except PAUSE) frames will be received.
Receive_pause	3	**Receive PAUSE**. When set, the Rx MAC will indicate the Rx PAUSE frame reception to the Tx MAC and thereby cause the Tx MAC to pause sending data frames for the period specified within the Rx PAUSE frame. Note this indication is independent of the Drop_control bit setting.
Tx_dis_fcs	2	**Transmit Disable FCS**. When set, the FCS field generation is disabled in the Tx MAC.
Discard_fcs	1	**Rx Discard FCS and Pad**. When set, the FCS and any of the padding bytes of an IEEE 802.3 frame are stripped off the frame before it is transferred to the Rx FIFO. When low, the entire frame is transferred into the Rx FIFO. Note: Discarding padding bytes is only applicable to pure IEEE 802.3 frames (such as in backplane applications) and will not function on Ethernet frames (IP, UDP, ICMP, etc.) where the length field is now interpreted as a protocol type field.
Prms	0	**Promiscuous Mode**. When asserted, all filtering schemes are abandoned and the Rx MAC receives frames with any address.

Maximum Packet Size (R/W)

Mnemonic: MAX_PKT_SIZE
POR Value = 05EEH (1518 decimal)

This register can be overwritten only when the MAC is disabled. All frames longer than the value (number of bytes) in this register will be tagged as long frames.

Name	Range	Description
Max_frame	15:0	Maximum size of the packet than can be handled by the core.

IPG (Inter-Packet Gap) (R/W)

Mnemonic: IPG_VAL
POR Value = 000CH

Name	Range	Description
Rsvd	15:5	Reserved.
IPG	4:0	Inter-packet gap value in units of bytes (one word clock for every two bytes of IPG).

MAC Address Register {0,1,2} (R/W), Set of Three

Mnemonic: MAC_ADDR
POR Value = 0000H

The MAC Address Registers 0-2 contain the Ethernet address of the port. The MAC Address Register [0] has the two bytes that are transmitted first and the MAC Address Register [2] has the two bytes that are transmitted last. Bit[8] through Bit[15] are transmitted first while bit[0] through bit[7] are transmitted last.

Note that the MAC address is stored in the registers in Hexadecimal form. For example, setting the MAC Address to AC-DE-48-00-00-80 would require writing 0xAC (octet 0) to address 0x0B (high byte of Mac_addr[15:0]), 0xDE (octet 1) to address 0x0A (Low byte of Mac_addr[15:0]), 0x48 (octet 2) to address 0x0D (high byte of Mac_addr[15:0]), 0x00 (octet 3) to address 0x0C (Low byte of Mac_addr[15:0]), 0x00 (octet 4) to address 0x0F (high byte of Mac_addr[15:0]), and 0x80 (octet 5) to address 0x0E (Low byte of Mac_addr[15:0]). Note Octet 0 is transmitted first and Octet 5 is transmitted last.

Name	Range	Description
Mac_addr	15:0	Ethernet address assigned to the port supported by the 2.5GMAC IP core.

Transmit and Receive Status (RO)

Mnemonic: TX_RX_STS
POR Value = 0000H

This register reports events that have occurred during packet reception and transmission.

Name	Range	Description
Rsvd	15:11	**Reserved**.
Rx_idle	10	**Receive MAC Idle**. Receive MAC in idle condition used to reset configurations by CPU interface.
Tagged_frame	9	**Tagged Frame**. Tagged frame received.
Brdcst_frame	8	**Broadcast Frame**. Indicates that a Broadcast packet was received.
Multcst_frame	7	**Multicast Frame**. Indicates that a Multicast packet was received.
IPG_shrink	6	**IPG Shrink**. Received frame with shrunk IPG (IPG < 96 bit time).
Short_frame	5	**Short Packet**. Indicates that a packet shorter than 64 bytes has been received.
Long_frame	4	**Too Long Packet**. Indicates receipt of a packet longer than the maximum allowable packet size specified in the **MAX_PKT_SIZE** register.
Error frame	3	**Rx_er Asserted**. Indicates the frame was received with the rx_er signal asserted.
CRC	2	**CRC Error**. Indicates a packet was received with a CRC error.
Pause_frame	1	**PAUSE Frame**. Indicates a PAUSE frame was received.
Tx_idle	0	**Transmit MAC Idle**. Transmit MAC in idle condition, used to reset configurations by CPU interface.

VLAN Tag (RO)

Mnemonic: VLAN_TAG
POR Value = 0000H.

The VLAN tag register has the VLAN TAG field of the most recent tagged frame that was received. This is a read only register.

Name	Range	Description
VLAN	15:0	This field defines length/type of field of the VLAN tag when inserted into transmitted frames.

Multicast Tables (R/W), Set of Eight

Mnemonic: MLT_TAB_[0-7]
POR Value = 0000H.

When the core is programmed to receive multicast frames, a filtering scheme is used to decide whether the frame should be received or not. The six middle bits of the most significant byte of the CRC value, calculated for the destination address, are used as a key to the 64-bit hash table. The three most significant bits select one of the eight tables, and the three least significant bits select a bit. The frame is received only if this bit is set.

Name	Range	Description
Multicast_table_[0-7]	7:0	**Multicast Table**. Eight tables that make a 64-bit hash.

Pause Opcode (R/W)

Mnemonic: PAUS_OP
POR Value = 0001H

This register contains the PAUSE Opcode, This will be compared against the Opcode in the received PAUSE frame. This value will also be included in any PAUSE frame transmitted by the 2.5GMAC IP core. Bit 15 is transmitted first and bit 0 is transmitted last.

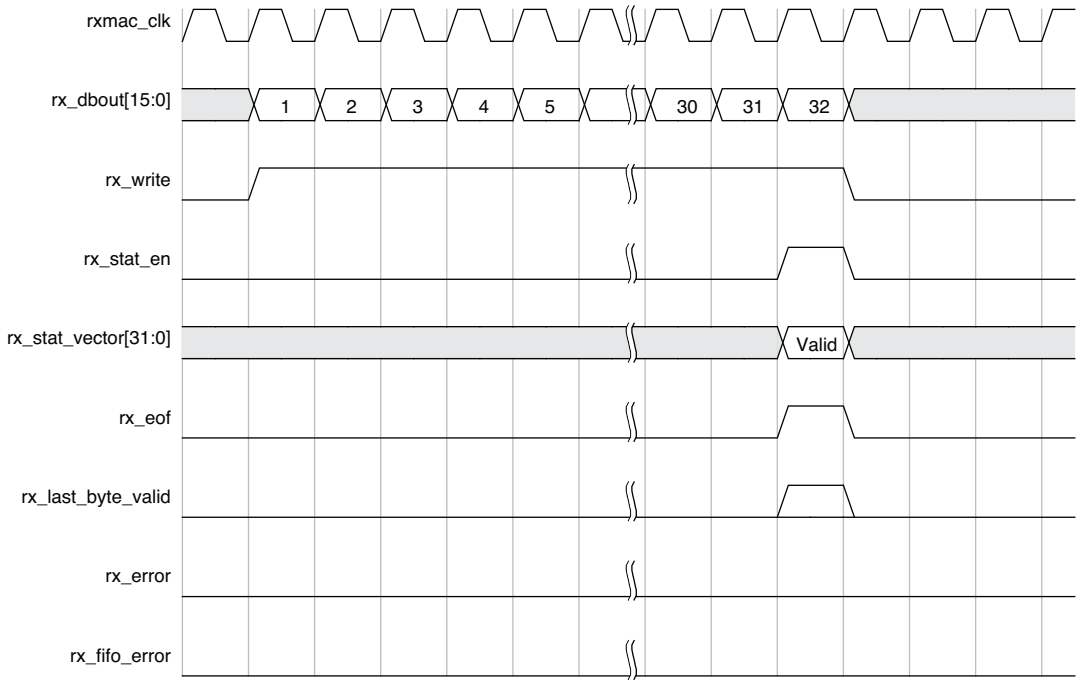
Name	Range	Description
Pause_OpCode	15:0	**PAUSE Opcode**.

Timing Specifications

This section contains operational timing diagrams applicable to the 2.5GMAC IP core interfaces.

Reception of a 64-Byte Frame Without Error – Rx MAC Application Interface

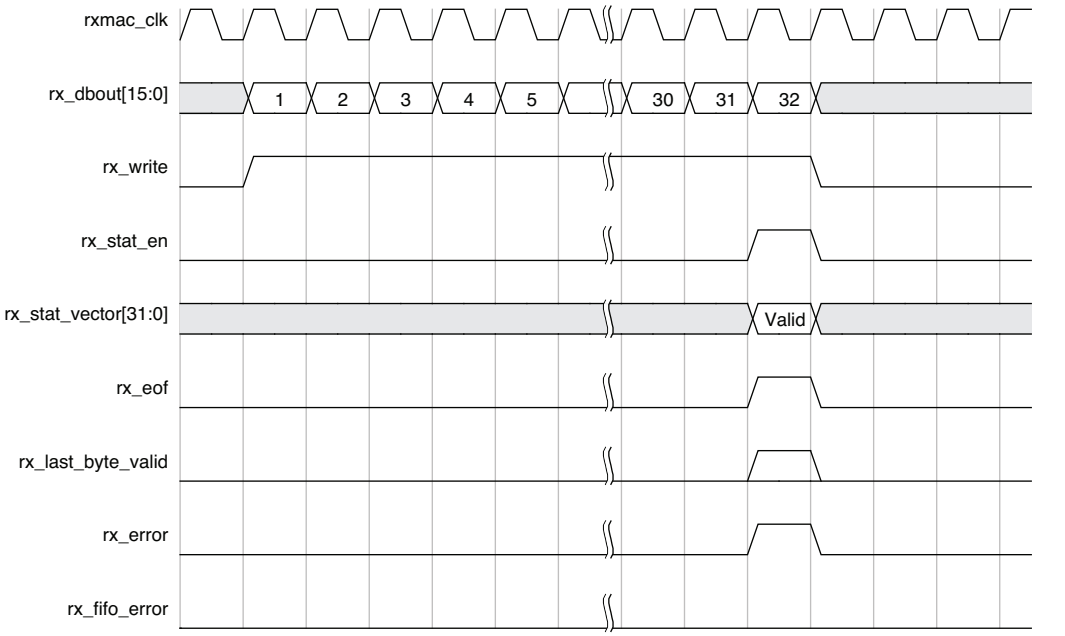
Figure 2-6. Reception of a 64-byte Frame Without Error



Reception of a 64-byte Frame with Error(s) – Rx MAC Application Interface

The signal rx_error is asserted to indicate that the 64-byte frame was received with error(s).

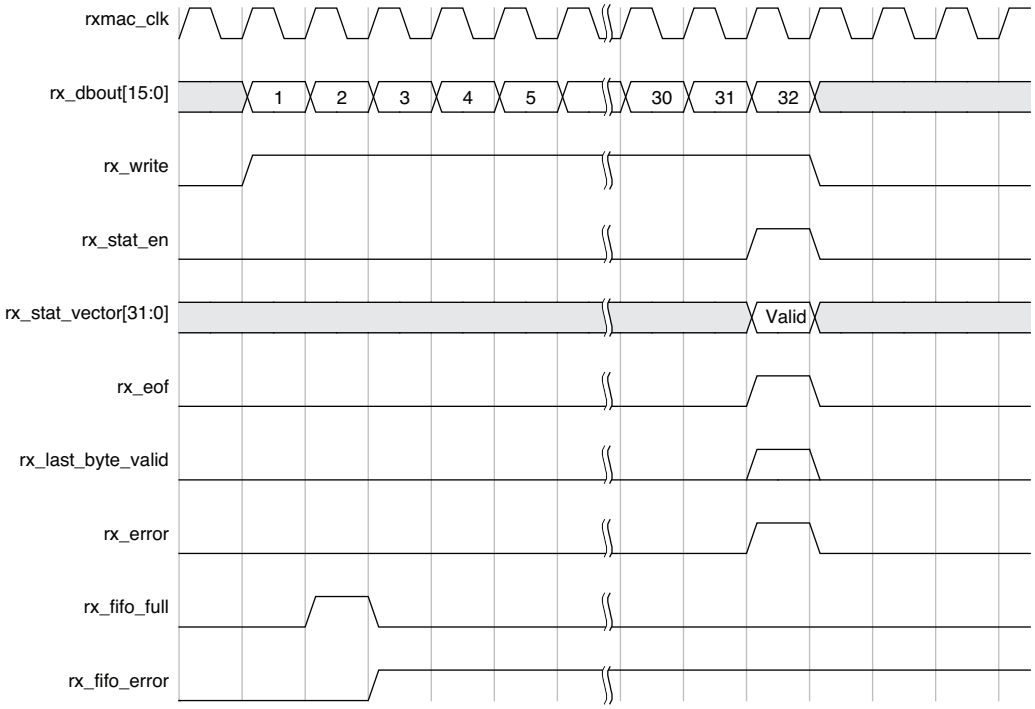
Figure 2-7. Reception of a 64-byte Frame with Error



Reception of a 64-Byte Frame with FIFO Overflow - Rx MAC Application Interface

The FIFO writing operation is suspended whenever an overflow condition occurs. When this condition occurs, the 2.5GMAC IP core asserts rx_fifo_error. This signal should be sampled along with rx_eof in order to process the error condition.

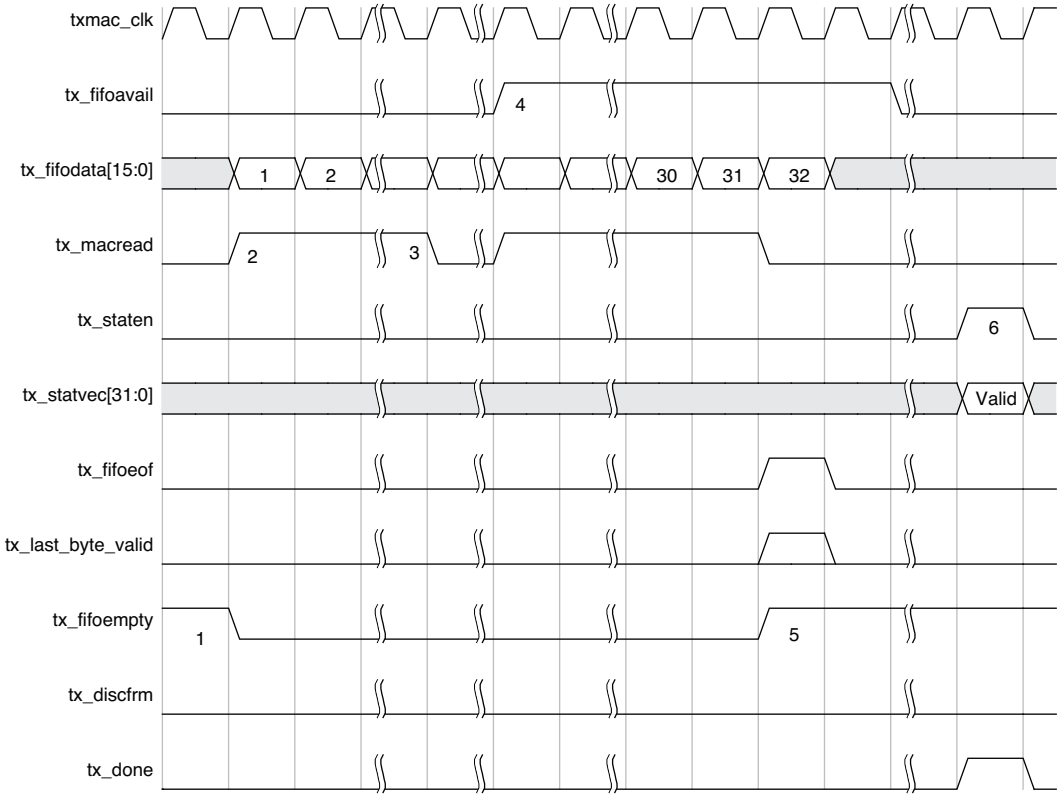
Figure 2-8. Reception of a 64-byte Frame with FIFO Overflow



Successful Transmission of a 64-Byte Frame -Tx MAC Application Interface

The assertion of tx_fifoavail indicates a frame is ready to be transmitted. It does not trigger the MAC transmit process. The MAC actually pre-reads the FIFO as soon as tx_fifoempty goes low (indicating data is present in the Tx FIFO). The tx_fifoempty signal, the tx_fifoavail signal and the MAC Tx state machine all determine when to read the Tx FIFO. The 2.5GMAC IP core reads the FIFO and the data is transmitted until tx_fifoeof is asserted. Once the frame is transmitted, tx_staten is asserted to qualify the statistic vector, tx_statvec. The signal tx_done is asserted to indicate a successful transmission. This is shown in Figure 2-9, and described as follows in detail.

Figure 2-9. Transmission of a 64-byte Frame without Error



1. The Tx FIFO is initially empty (tx_fifoempty is asserted) and the MAC Tx state machine is in an idle state.
2. The client interface begins loading the Tx FIFO. Once the tx_fifoempty signal is de-asserted the tx_macread is asserted and the MAC Tx state machine goes into a transmit state. The MAC performs a “pre-read” and continues to assert the tx_macread as long as the Tx FIFO is not empty (tx_fifoempty stays low). The pre-read typically is 7 to 8 bytes.
3. If tx_fifoavail is low after the pre-read, the Tx MAC will de-assert the tx_macread until the tx_fifoavail is asserted.
4. tx_fifoavail is set high by the client interface indicating all bytes of the packet are now available for transmission. The Tx MAC re-asserts tx_macread and reads all bytes until tx_fifoeof indicates the last byte.
5. Once the complete packet is read out by the Tx MAC, it checks the status of tx_fifoempty and tx_fifoavail again. If either tx_fifoavail is low or tx_fifoempty is high at the end of the packet, the MAC de-asserts tx_macread. If tx_fifoavail is high and tx_fifoempty is low, (FIFO is not empty) the Tx MAC will do another pre-read (not shown in this figure) and start the transmit process over.
6. Once the frame is transmitted, tx_staten is asserted to qualify the statistic vector, tx_statvec. The signal tx_done is asserted to indicate a successful transmission.

Additional Notes:

- The MAC Tx state machine may do other processing, such as inserting IPG after a previous packet is transmitted, so even if tx_fifoempty is low and tx_fifoavail is high the MAC may not assert the tx_macread right away.
- There are many ways to interface client logic and FIFOs to the MAC - here are two examples:

1.) An appropriate threshold level for the Tx FIFO is chosen which indicates that a full frame is ready to be sent. This is used as the tx_fifoavail signal. If there are variable size packets (some small) one may want to OR this threshold with a frames_present signal. frames_present is a frames counter that keeps track of EOPs into and out of the FIFO. Whenever the FIFO is Not Almost Empty and there is at least one full packet in the FIFO tx_fifoavail is set high to indicate it is safe to transmit a packet.

2.) Decouple the tx_fifoempty and tx_fifoavail signals from the actual Tx FIFO and generate tx_fifoavail and tx_fifoempty signals from a simple state machine:

I) Keep tx_fifoavail low and tx_fifoempty high - load Tx FIFO

II) When packet is fully loaded into Tx FIFO assert tx_fifoavail high and de-assert tx_fifoempty (set to not empty)

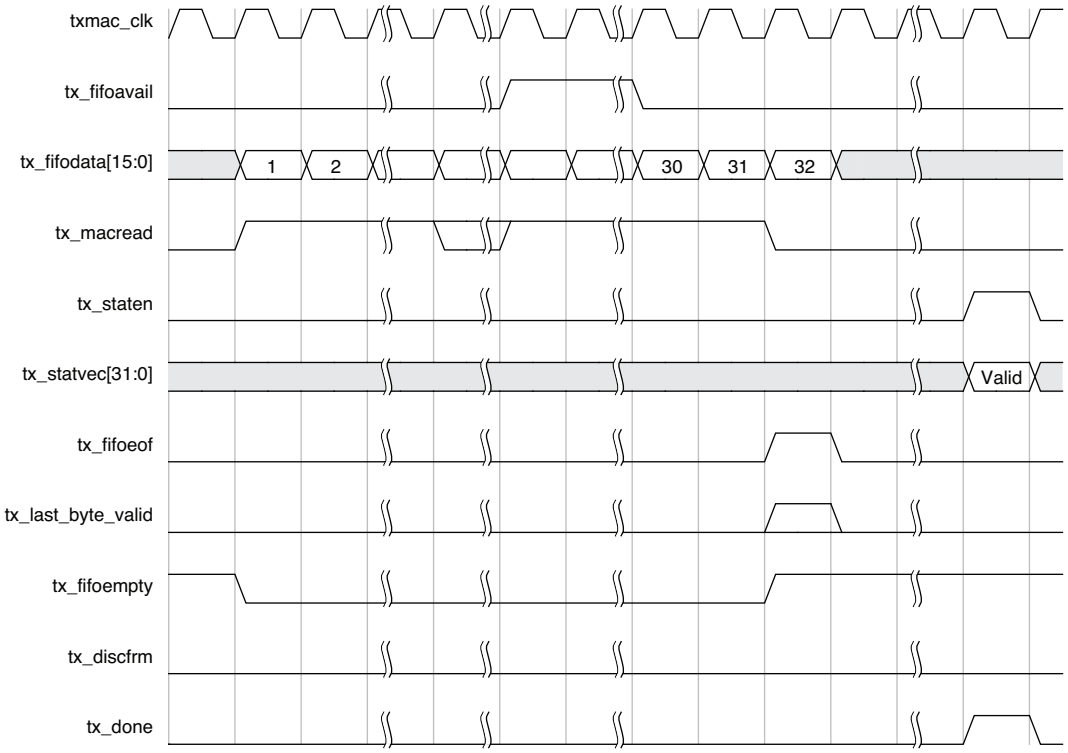
III) When packet is completely pulled out of FIFO, de-assert tx_fifoavail and force tx_fifoempty high

Successful Transmission of a 64-byte Frame with FIFO Empty – Tx MAC Application Interface

tx_fifoempty is asserted along with tx_fifoeof to indicate that the complete 64-byte frame has been read.

The frame is transmitted as a valid frame and tx_done is asserted at the end of transmission.

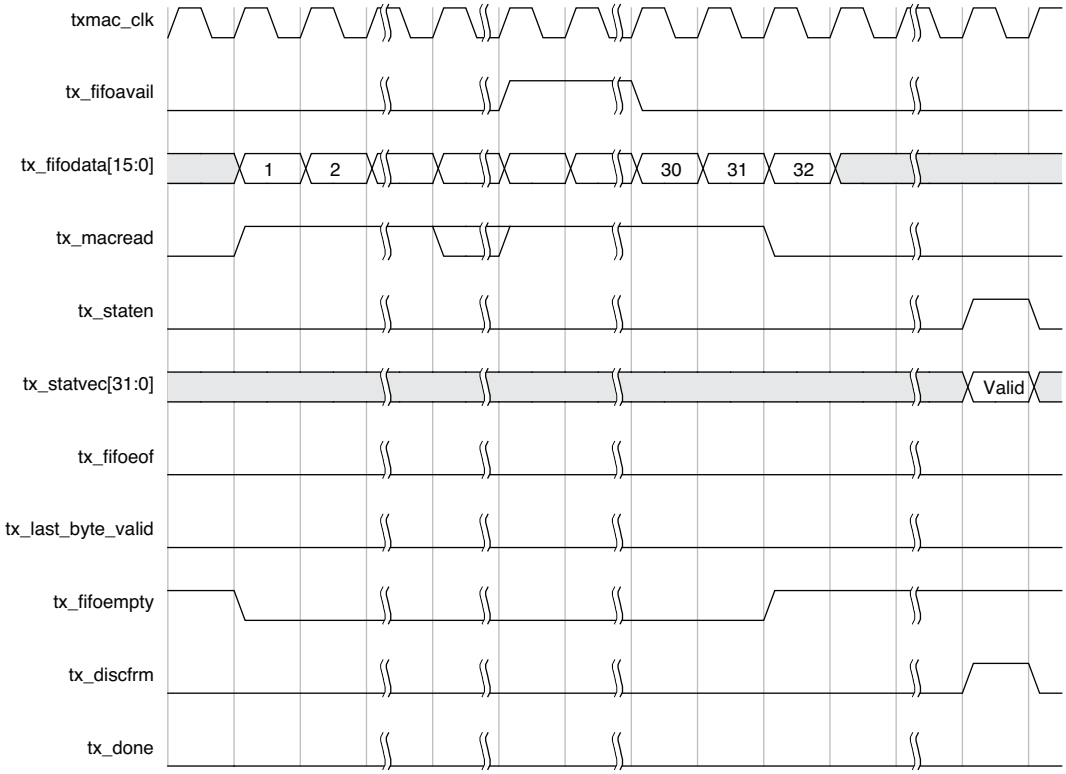
Figure 2-10. Successful Transmission of a 64-byte Frame with FIFO Empty



Aborted Transmission Due to FIFO Empty – Tx MAC Application Interface

If the tx_fifoempty is asserted while the Tx MAC is in the process of reading a frame, the MAC will stop reading the frame and assert tx_disfrm to indicate an erroneous transmission. The frame transmission is abandoned when this occurs. Note in Figure 2-11 the tx_fifoempty was asserted before the end of the frame (no tx_fifoeof asserted).

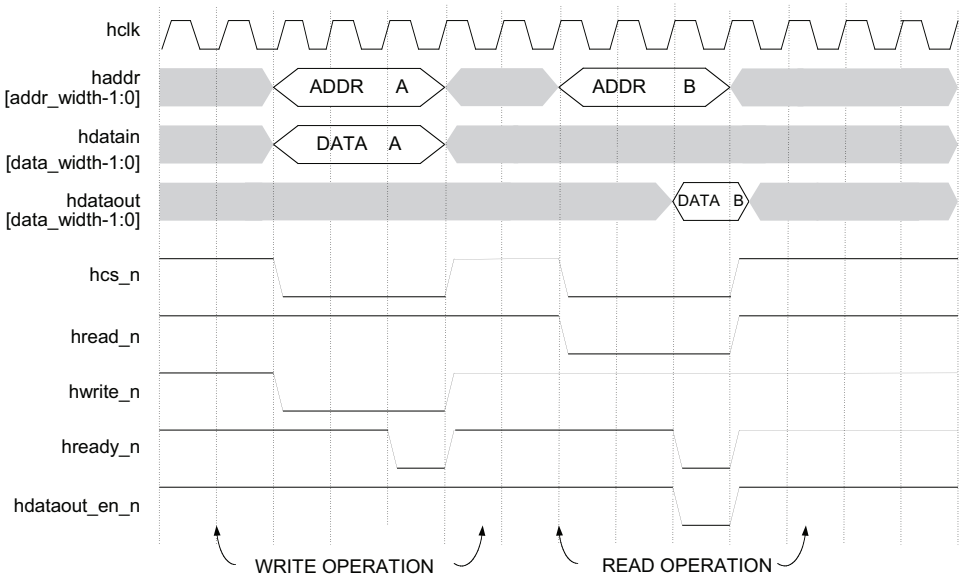
Figure 2-11. Aborted Transmission Due to FIFO Empty



Host Interface Read/Write Operation

During a write operation, haddr associated with hdatain, hcs_n and hwrite_n performs a write operation to an internal register. The end of transaction is indicated by assertion of hready_n. During a read operation, haddr associated with hcs_n and hread_n forms a write operation. The end of transaction is indicated by the assertion of hready_n and hdataout_en_n along with the valid read data on hdataout.

Figure 2-12. Host Interface Read/Write Operation

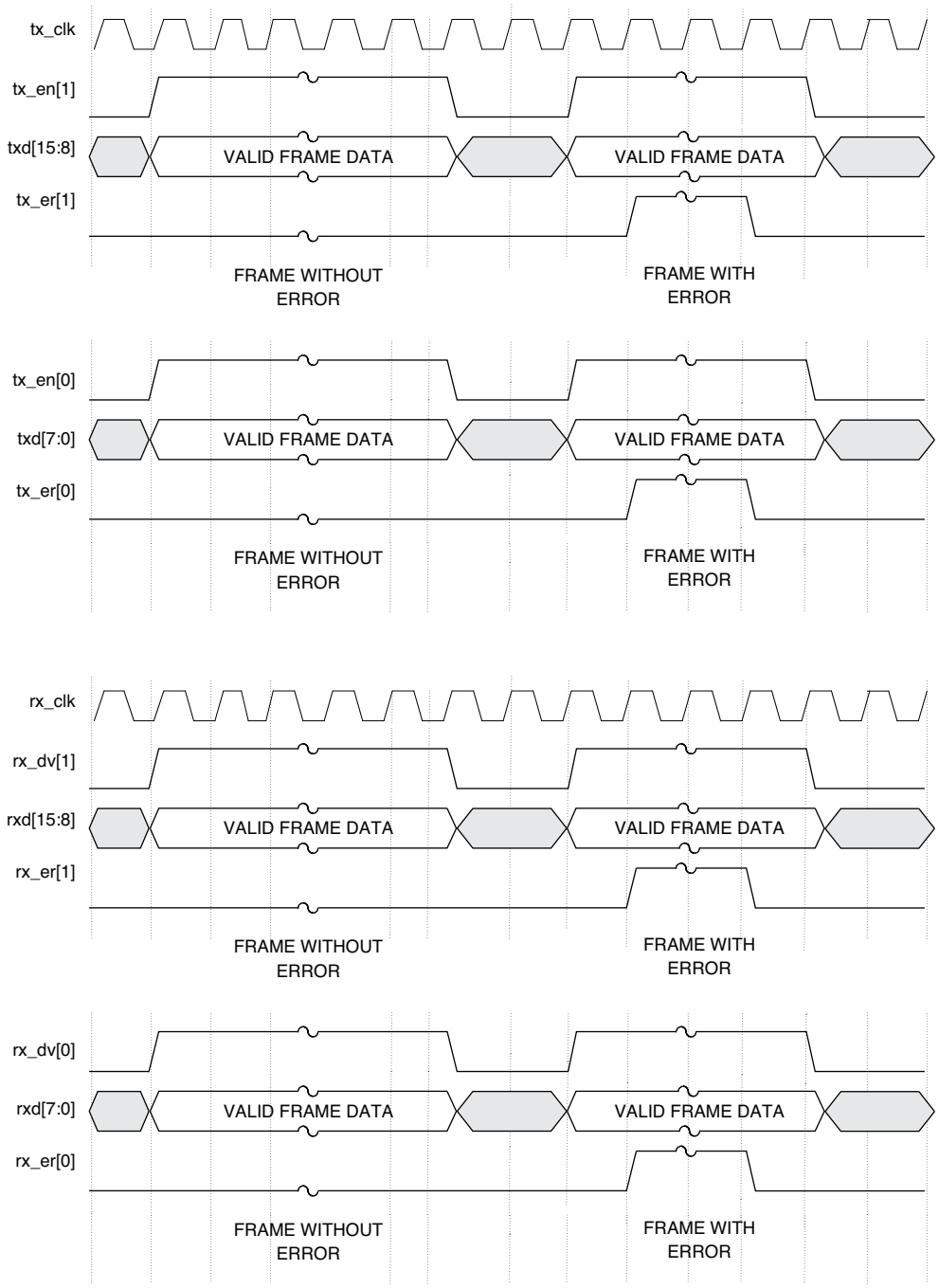


GMII Transmit and Receive Operations

txd and tx_en are driven synchronous to the tx_clk during transmit operations. When the frame being transmitted has an error, tx_er is asserted.

When receiving data, rxd and rx_dv are sampled on the rising edge of rx_clk. An error in the frame is indicated when rx_er is asserted.

Figure 2-13. GMII Transmit and Receive Operations



Synthesis/Simulation Tools Selection

The 2.5GMAC IP core evaluation capability supports multiple synthesis and simulation tool flows. These options allow the user to select desired tool support.

IP Core Generation and Evaluation

This chapter provides information on how to generate the Lattice 2.5GMAC IP core using the Diamond software IPexpress tool, and how to include the core in a top-level design.

Licensing the IP Core

An IP core- and device-specific license is required to enable full, unrestricted use of the 2.5GMAC IP core in a complete, top-level design. Instructions on how to obtain licenses for Lattice IP cores are given at:

www.latticesemi.com/products/intellectualproperty/aboutip/isplevercoreonlinepurchas.cfm

Users may download and generate the 2.5GMAC IP core and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license. The 2.5GMAC IP core also supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited time (approximately four hours) without requiring an IP license. See "Hardware Evaluation" on page 33 for further details. However, a license is required to enable timing simulation, to open the design in the Diamond EPIC tool, and to generate bitstreams that do not include the hardware evaluation timeout limitation.

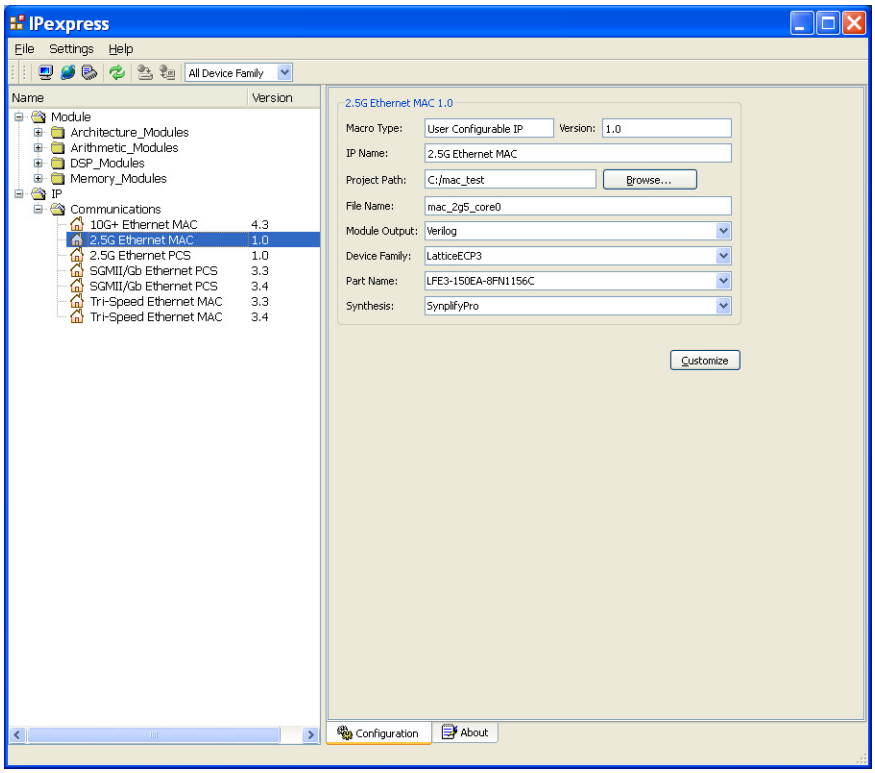
Getting Started

The 2.5GMAC IP core is available for download from the Lattice IP Server using the IPexpress tool. The IP files are automatically installed using ispUPDATE technology in any customer-specified directory. After the IP core has been installed, the IP core will be available in the IPexpress GUI dialog box shown in Figure 4-1.

The IPexpress tool GUI dialog box for the 2.5GMAC IP core is shown in Figure 4-1. To generate a specific IP core configuration the user specifies:

- **Project Path** – Path to the directory where the generated IP files will be located.
- **File Name** – "username" designation given to the generated IP core and corresponding folders and files.
- **(Diamond) Module Output** – Verilog or VHDL.
- **Device Family** – Device family to which IP is to be targeted. Only families that support the particular IP core are listed.
- **Part Name** – Specific targeted part within the selected device family.

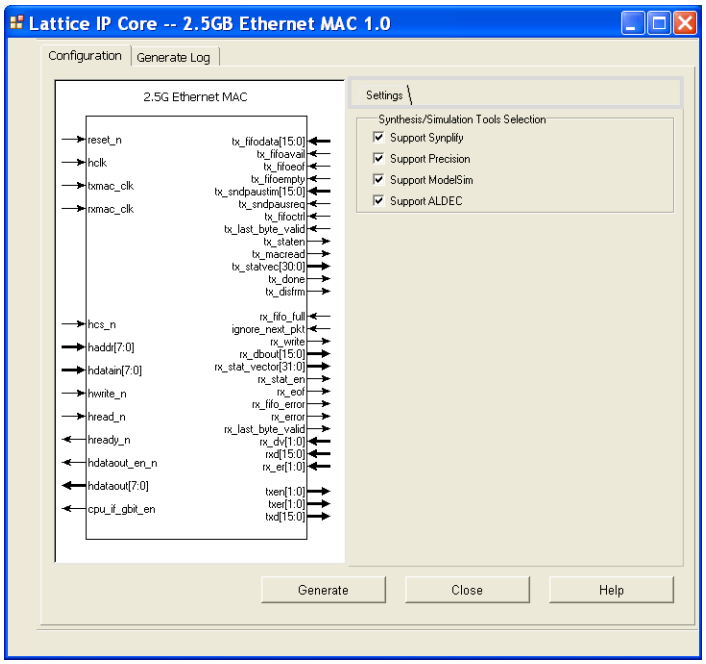
Figure 4-1. IPexpress Tool Dialog Box (Diamond Version)



Note that if the IPexpress tool is called from within an existing project, Project Path, Module Output, Device Family and Part Name default to the specified project parameters. Refer to the IPexpress tool online help for further information.

To create a custom configuration, the user clicks the **Customize** button in the IPexpress tool dialog box to display the 2.5GMAC IP core Configuration GUI, as shown in Figure 4-2. From this dialog box, the user can generate the IP core.

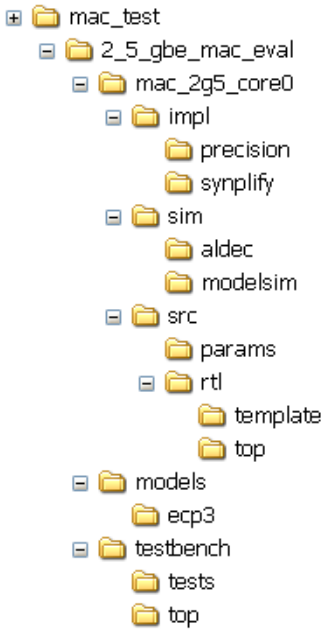
Figure 4-2. 2.5GMAC IP Core - Configuration GUI (Diamond Version)



IPexpress-Created Files and Top Level Directory Structure

When the user clicks the **Generate** button in the IP Configuration dialog box, the IP core and supporting files are generated in the specified “Project Path” directory. The directory structure of the generated files is shown in Figure 4-3.

Figure 4-3. LatticeECP3 2.5GMAC IP Core Directory Structure



The design flow for IP created with the IPexpress tool uses a post-synthesized module (NGO) for synthesis and a protected model for simulation. The post-synthesized module is customized and created during the IPexpress tool generation.

Table 4-1 provides a list of key files and directories created by the IPexpress tool and how they are used. The IPexpress tool creates several files that are used throughout the design cycle. The names of most of the created files are customized to the user's module name specified in the IPexpress tool. These are all of the files needed to implement and verify the 2.5GMAC IP core in a top-level design.

Table 4-1. File List

File	Description
<*username*>.lpc	This file contains the IPexpress tool options used to recreate or modify the core in the IPexpress tool.
<*username*>.ipx	The IPX file holds references to all of the elements of an IP or Module after it is generated from the IPexpress tool (Diamond version only). The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP/Module generation GUI when an IP/Module is being re-generated.
<*username*>.ngo	This file provides the synthesized IP core.
<*username*>_bb.v	This file provides the synthesis black box for the user's synthesis.
<*username*>_inst.v	This file provides an instance template for the 2.5GMAC IP core.
<*username*>_beh.v	This file provides the front-end simulation library for the 2.5GMAC IP core.

Table 4-2 provides a list of key additional files providing IP core generation status information and command line generation capability are generated in the user's project directory.

Table 4-2. Additional Files

File	Description
<*username*>_generate.tcl	Created when GUI "Generate" button is pushed, invokes generation, may be run from command line.
<*username*>_generate.log	Diamond synthesis and map log file.
<*username*>_gen.log	IPexpress IP generation log file

The **\<2_5_gbe_mac_eval>** and subtending directories provide files supporting 2.5GMAC IP core evaluation. The **\<2_5_gbe_mac_eval>** directory shown in Figure 4-3 contains files and folders with content that is constant for all configurations of the 2.5GMAC. The **\<*username*>** subfolder (**\mac_2g5_core0** in this example) contains files and folders with content specific to the username configuration.

The **\<2_5_gbe_mac_eval>** directory is created by IPexpress the first time the core is generated and updated each time the core is regenerated. A **\<*username*>** directory is created by IPexpress each time the core is generated and regenerated each time the core with the same file name is regenerated. A separate \<username> directory is generated for cores with different names, e.g. **\<mac_core0>, \<mac_core1>**, etc.

Instantiating the Core

The generated 2.5GMAC IP core package includes black-box (<*username*>_bb.v) and instance (<*username*>_inst.v) templates that can be used to instantiate the core in a top-level design. An example RTL top-level reference source file that can be used as an instantiation template for the IP core is provided in **\<*project_dir*>\2_5_gbe_mac_eval\<*username*>\src\rtl\top**. Users may also use this top-level reference as the starting template for the top-level for their complete design.

Running Functional Simulation

Simulation support for the 2.5GMAC IP core is provided for Aldec Active-HDL (Verilog and VHDL) simulator, Mentor Graphics ModelSim (Verilog only) simulator, and Cadence NC-Verilog (Linux only) simulator.

The functional simulation includes a configuration-specific behavioral model of the 2.5GMAC IP core, which is instantiated in an FPGA top level along with some test logic (MAC client side FIFO loop back logic and registers with Read/Write Interface). This FPGA top, which is referred to as the Test Application Design, is instantiated in an

evaluation test bench that configures FPGA test logic registers and 2.5GMAC IP core registers. The test bench also sources Ethernet packets to the Test Application, and monitors packets from Test Application.

More information on the simulation and the Test Application Design can be found in “Application Support” on page 35.

The generated IP core package includes the configuration-specific behavior model (<username>_beh.v) for functional simulation in the “Project Path” root directory. Lattice does not provide a test bench for evaluating this IP core in isolation. However, a functional simulation capability is provided in which <username>_beh.v is instantiated in the Test Application Design described in Application Support section of this document.

The simulation script supporting ModelSim evaluation simulation is provided in **\<*project_dir*>\2_5_gbe_mac_eval\<*username*>\sim\modelsim**.

The simulation script supporting Active-HDL evaluation simulation is provided in **\<*project_dir*>\2_5_gbe_mac_eval\<*username*>\sim\aldec.**

The Test Application Design is instantiated in a test-bench provided in **\<*project_dir*>\2_5_gbe_mac_eval\testbench**.

Both ModelSim and Active-HDL simulation is supported via test bench files provided in **\<*project_dir*>\2_5_gbe_mac_eval\testbench**. Models required for simulation are provided in the corresponding **\models** folder.

Users may run the Active-HDL evaluation simulation by doing the following:

1. Open Active-HDL.
2. Under the Tools tab, select **Execute Macro**.
3. Browse to folder **\<*project_dir*>\2_5_gbe_mac_eval\<*username*>\sim\aldec** and execute one of the “do” scripts shown.

Users may run the ModelSim evaluation simulation by doing the following:

1. Open ModelSim.
2. Under the File tab, select **Change Directory** and choose the folder **<*project_dir*>\2_5_gbe_mac_eval\<*username*>\sim\modelsim**.
3. Under the Tools tab, select **Execute Macro** and execute the ModelSim “do” script shown.

Synthesizing and Implementing the Core in a Top-Level Design

Synthesis support for the 2.5GMAC IP core is provided for Mentor Graphics Precision or Synopsys Synplify. The 2.5GMAC IP core itself is synthesized and is provided in NGO format when the core is generated in IPexpress. Users may synthesize the core in their own top-level design by instantiating the core in their top-level as described previously and then synthesizing the entire design with either Synplify or Precision RTL Synthesis.

The following text describes the evaluation implementation flow for Windows platforms. The flow for Linux and UNIX platforms is described in the Readme file included with the IP core.

Two example top-level reference source files are provided to support 2.5GMAC top-level synthesis and implementation.

One top file is for a 2.5GMAC IP core only implementation in isolation. This design is intended only to provide an accurate indication of the device utilization associated with the 2.5GMAC IP core itself and should not be used as an actual implementation example.

The other top file is for the Test Application Design, and includes both the 2.5GMAC IP core and additional loop back test logic. This is the same top used in the functional simulation described in the previous section.

Both top-level files, G25_mac_core_only_top.v (core only) and G25_mac_top.v (Application), are provided in **\<*project_dir*>\2_5_gbe_mac_eval\<*username*>\src\rtl\top**.

In Diamond, push-button implementation of both reference designs is supported via the project files, <*username*>_reference_eval.ldf and <*username*>_core_only_eval.ldf located in **\<*project_dir*>\2_5_gbe_mac_eval\<*username*>\impl\(*synplify or precision*)**.

Using Project Files With Synplify in Diamond

1. Choose **File > Open > Project**.
2. Browse to **\<*project_dir*>\2_5_gbe_mac_eval\<*username*>\impl\synplify** in the Open Project dialog box.
3. Select and open <*username*>_reference_eval.ldf or <*username*>_core_only_eval.ldf. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.
4. Implement the complete design via the standard Diamond GUI flow.

Hardware Evaluation

The 2.5GMAC IP core supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited period of time (approximately four hours) without requiring the purchase of an IP license. It may also be used to evaluate the core in hardware in user-defined designs.

Enabling Hardware Evaluation in Diamond

Choose **Project > Active Strategy > Translate Design Settings**. The hardware evaluation capability may be enabled/disabled in the Strategy dialog box. It is enabled by default.

Updating/Regenerating the IP Core

By regenerating an IP core with the IPexpress tool, you can modify any of its settings including: device type, design entry method, and any of the options specific to the IP core. Regenerating can be done to modify an existing IP core or to create a new but similar one.

Regenerating an IP Core in Diamond

To regenerate an IP core in Diamond:

1. In IPexpress, click the **Regenerate** button.
2. In the Regenerate view of IPexpress, choose the IPX source file of the module or IP you wish to regenerate.
3. IPexpress shows the current settings for the module or IP in the Source box. Make your new settings in the T**arget** box.
4. If you want to generate a new set of files in a new location, set the new location in the **IPX Target File** box. The base of the file name will be the base of all the new file names. The IPX Target File must end with an .ipx extension.
5. Click **Regenerate.** The module's dialog box opens showing the current option settings.
6. In the dialog box, choose the desired options.
7. To import the module into your project, if it's not already there, select **Import IPX to Diamond Project** (not available in stand-alone mode).

8. Click **Generate**.

9. Check the Generate Log tab to check for warnings and error messages.

10. Click **Close**.

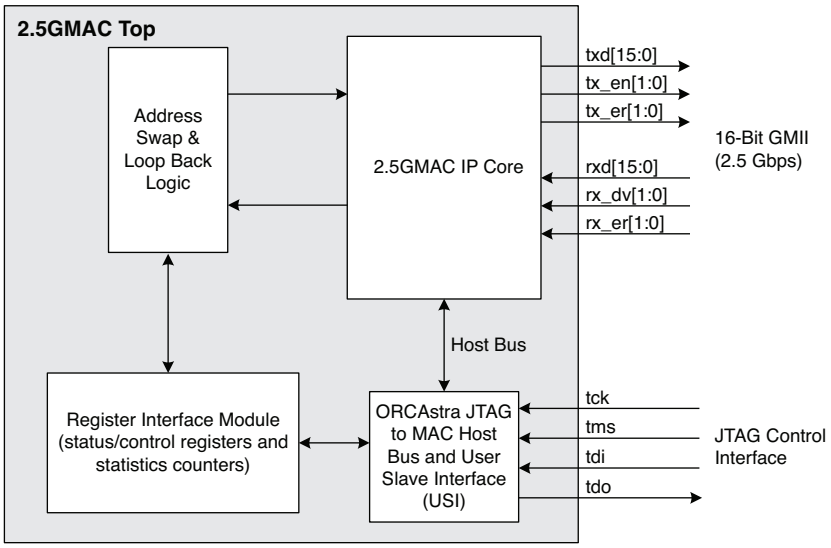
The IPexpress package file (.ipx) supported by Diamond holds references to all of the elements of the generated IP core required to support simulation, synthesis and implementation. The IP core may be included in a user's design by importing the .ipx file to the associated Diamond project. To change the option settings of a module or IP that is already in a design project, double-click the module's .ipx file in the File List view. This opens IPexpress and the module's dialog box showing the current option settings. Then go to step 6 above.

This chapter provides application support information for the 2.5GMAC IP core.

Test Application Design

The 2.5GMAC IP core evaluation package includes a reference design that can be used to instantiate, simulate, map, place and route the 2.5GMAC IP core in an example working design. This reference design provides a loop back path for packets on the MAC Rx/Tx client interface, through a FIFO and associated logic. Ethernet packets are sourced to the GMII and looped back on the MAC Rx/Tx client FIFO interface. Source and destination addresses in the ethernet frame can be swapped so the looped back packets on the Tx GMII have the correct source and destination addresses. Figure 5-1 shows a block diagram of the test application design.

Figure 5-1. Test Application Design



The main blocks and functions of the test application design are as follows:

The Test Logic Module

This module includes the address swap logic, loop back FIFO and associated control logic, and miscellaneous control and status glue logic between the MAC Rx/Tx Client interface and the register interface module.

The JTAG ORCAstra to Host Bus/USI module

This module converts the JTAG bus to the host bus and a user slave interface bus. Using the JTAG bus, a user can access the internal 2.5GMAC IP core registers, as well as the test application registers. The MAC registers (accessed via the host bus) and test logic registers (accessed via the USI) are memory mapped as described in "Test Application Registers" on page 37. A JTAG Bus Test bench driver is provided to ease simulation with this interface.

The Register Interface Module

This module is accessed through the JTAG bus via the user slave interface (USI). The module contains registers used by the test application for control and status of the IP core. In addition this module contains 16 bit statistics counters fed by the 2.5GMAC IP core's Rx/Tx statistics interfaces. These counters can be read and cleared through the JTAG bus. An address map and description of these registers is given in "Test Application Registers" on page 37.

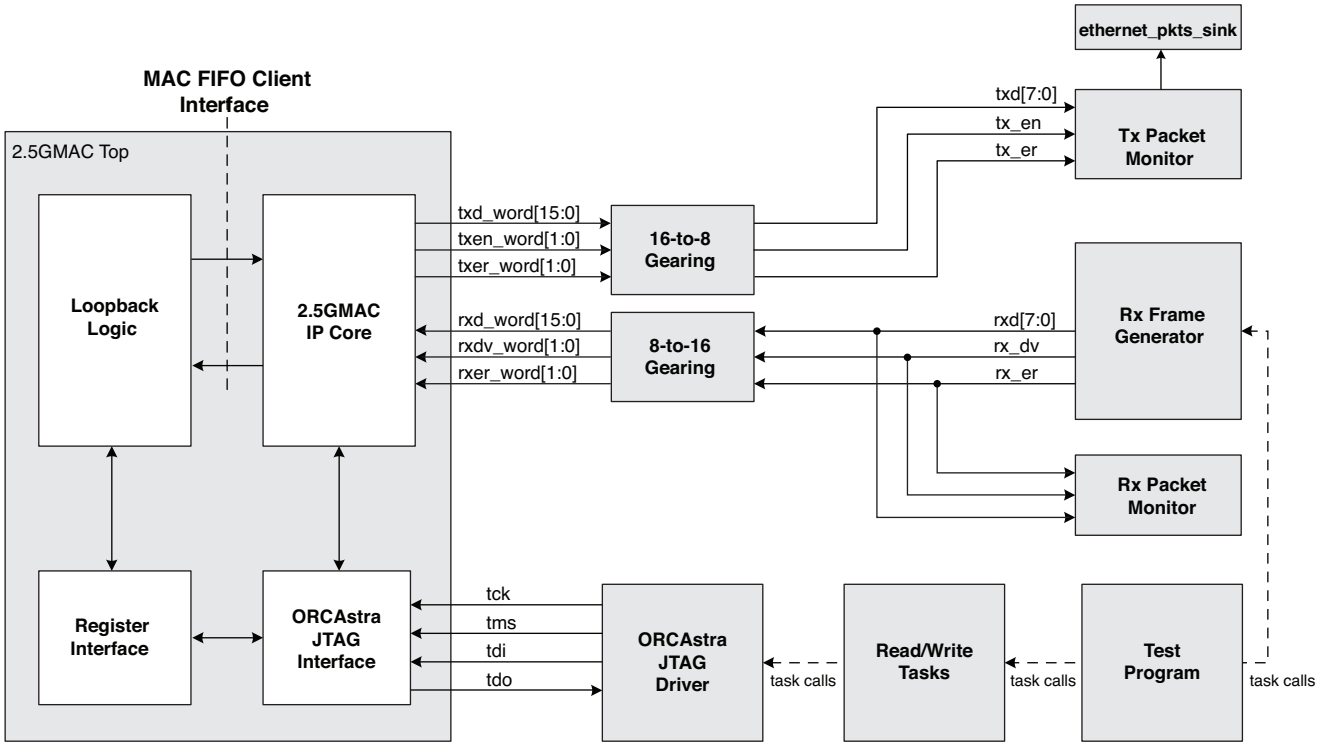
2.5GMAC Support Logic

This logic includes I/O, registers, clock dividers and multiplexers used by the 2.5GMAC IP core and test application.

Simulation of the Test Application

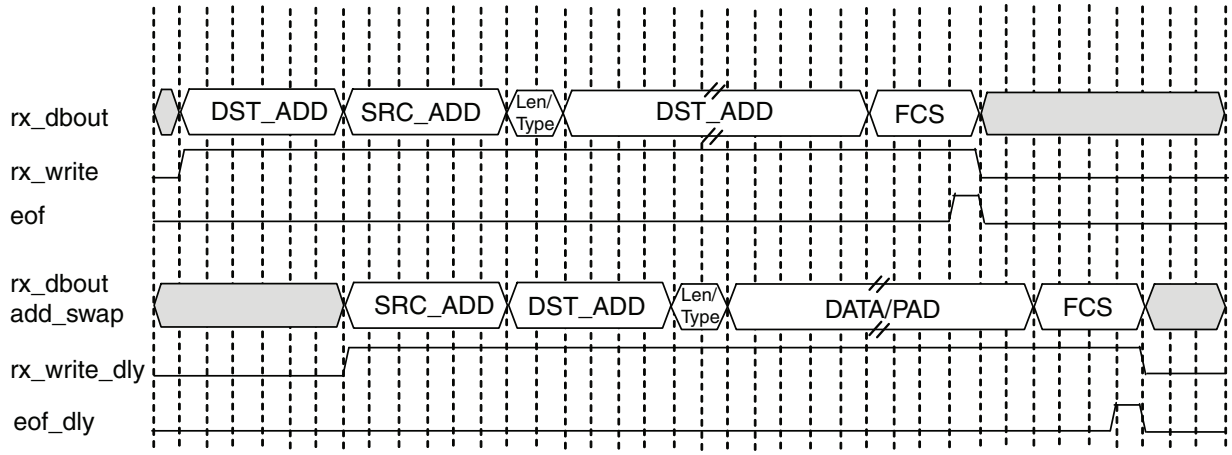
Figure 5-2 shows a block diagram of the test bench setup provided with the Test Application Design. All Accesses to the internal 2.5GMAC registers and test application design registers can be accomplished through the testcase.v file (via a JTAG ORCAstra driver). In addition, variable size and types of ethernet frames can be sourced to the 2.5GMAC Rx GMII port through the testcase.v file (via the Rx frame generator). These received packets get looped back inside the test application design and are monitored on the 2.5GMAC Tx GMII by a packet monitor. The monitored packets are logged in a file named ethernet_pkts_sink. The testcase.v file can be found in **\<*project_dir*>\2_5_gbe_mac_eval\testbench\tests**. While the ethernet_pkts_sink file will be created and placed in **\<*project_dir*>\2_5_gbe_mac_eval\core_name\sim\(modelsim or aldec)** directory once the simulation is run and completed.

Figure 5-2. Block Diagram of Test Bench Setup



Note that the user can easily edit the testcase.v file to configure and monitor any registers they desire in the test application design, as well modify the type or number of packets they wish to drive to the test application design. Also note that the destination and source addresses can be swapped by enabling the swap control bit in the test control register. Figure 5-3 shows a timing waveform of data and control signals on the ingress and egress sides of the address swap module (when address swap is enabled).

Figure 5-3. Timing Waveform



Test Application Registers

There are two address spaces in the test application design. The first space begins at base address 0x800 and is used for addressing the IP core internal registers. Refer to Table 2-3 on page 16 for a listing of 2.5GMAC IP Core internal registers. The first address in Table 2-3 on page 16 starts at offset 0x00 and ends at 0x35. Adding the 0x800 base address to an IP core register address from Table 2-3 on page 16 forms the memory mapped register address.

The second address space shown in Table 5-1 starts at address 0x8000 and ends at 0x8037. This space contains ID, control, status and statistics registers used by the test application.

The REGINTF logic block in the test application provides the address decoding, for the RO and R/W registers used by the test logic and statistics counters. All registers are 8 bits wide and are byte addressable. Note that the statistics counter registers are composed of two 8 bit registers, a low and a high byte register, therefore, in order to access these registers, two byte accesses must be made. For example, to access to all 16 bits of the RXOKCNT would require an access to both 0x8019 (high byte) and 0x8018 (low byte). Note that since the statistics counter registers are clear on read (COR), the high byte should be read first before reading the low byte, since a read of the low byte clears all the combined 16 bits of the low and high registers. The address map for the test application related resisters are listed in Table 5-1.

Table 5-1. Test Application Related Registers

Address	Register Description	Mnemonic	Type
0x08000	VERsion/IDentification Register	VERID	RO
0x08001	TeST CoNTroL Register	TSTCNTL	RW
0x08002	TeST CoNTroL Register 2	TSTCNTL_2	RW
0x08003	MAC CoNTroL Register	MACCNTL	RW
0x08004	PAUSe TiMeR Register - Low byte	PAUSTMRL	RW
0x08005	PAUSe TiMeR Register - High byte	PAUSTMRH	RW
0x08006	FIFO Almost Full Threshold Register - Low	FIFOAFTL	RW
0x08007	FIFO Almost Full Threshold Register - High	FIFOAFTH	RW
0x08008	FIFO Almost Empty Threshold Register - Low	FIFOAETL	RW
0x08009	FIFO Almost Empty Threshold Register - High	FIFOAETH	RW
0x0800a	Rx Status Register	RXSTATUS	RO/COR
0x0800b	Tx Status Register	TXSTATUS	RO/COR

Table 5-1. Test Application Related Registers

Address	Register Description	Mnemonic	Type
0x0800c, 0x0800d	Rx Packet Ignored Counter Register (L,H)	RXPICNT	RO/COR
0x0800e, 0x0800f	Rx Length Check Error CouNTer (L,H)	RXLCECNT	RO/COR
0x08010, 0x08011	Rx Long Frames CouNTer Register (L,H)	RXLFCNT	RO/COR
0x08012, 0x08013	Rx Short Frames CouNTer Register (L,H)	RXSFCNT	RO/COR
0x08014, 0x08015	Rx IPG violations CouNTer Register (L,H)	RXIPGCNT	RO/COR
0x08016, 0x08017	Rx CRC errors CouNTer Register (L,H)	RXCRCCNT	RO/COR
0x08018, 0x08019	Rx OK packets CouNTer Register (L,H)	RXOKCNT	RO/COR
0x0801a, 0x0801b	Rx Control Frame CouNTer Register (L,H)	RXCFCNT	RO/COR
0x0801c, 0x0801d	Rx Pause Frame CouNTer Register (L,H)	RXPFCNT	RO/COR
0x0801e, 0x0801f	Rx Multicast Frame CouNTer Register (L,H)	RXMFCNT	RO/COR
0x08020, 0x08021	Rx Broadcast Frame CouNTer Register (L,H)	RXBFCNT	RO/COR
0x08022, 0x08023	Rx VLAN tagged Frame CouNTer Register (L,H)	RXVFCNT	RO/COR
0x08024, 0x08025	Tx Unicast Frame CouNTer Register (L,H)	TXUFCNT	RO/COR
0x08026, 0x08027	Tx Pause Frame CouNTer Register (L,H)	TXPFCNT	RO/COR
0x08028, 0x08029	Tx Multicast Frame CouNTer Register (L,H)	TXMFCNT	RO/COR
0x0802a, 0x0802b	Tx Broadcast Frame CouNTer Register (L,H)	TXBFCNT	RO/COR
0x0802c, 0x0802d	Tx VLAN tagged Frame CouNTer Register (L,H)	TXVFCNT	RO/COR
0x0802e, 0x0802f	Tx BAD FCS Frame CouNTer Register (L,H)	TXBFCCNT	RO/COR
0x08030, 0x08031	Tx Jumbo Frame CouNTer Register (L,H)	TXJFCNT	RO/COR
0x8032, 0x08031	Rx Byte Count	RXBYTECNT	RO/COR
0x08034, 0x08035	Tx Byte Count	TXBYTECNT	RO/COR
0x08036, 0x08037	Tx Underrun	TXURUNCNT	RO/COR
0x08038 - 0x0FFFF	Unused		

Register Descriptions

Version/Identification (RO)

Mnemonic: VERID

POR Value = A2H

Name	Range	Description
Version_ID [7:0]	7:0	**Version ID**: Echoes back the version and ID of the application (A2H).

Test Control Register (R/W)

Mnemonic: TSTCNTL

POR Value = 00H

Name	Range	Description
Rsvd	7:4	Reserved
pkt_loop_clksel	3	**Packet Loop Clock Select**: When this bit is set High, sys_clk is tied to rx_clk. When the bit is Low sys_clk is sourced from an external IO pin.
Rsvd	2	**Reserved**.
loop back enable	1	**Loop back Enable**: When this bit is set High the client side loop back is enabled. When the bit is Low the Loop back is disabled.
destination/source address swap	0	**Swap Destination and Source Addresses:** When this bit is set High the Ethernet Destination and source addresses are swapped. When the bit is Low there is no address swapping.

Test Control Register 2 (R/W)

Mnemonic: TSTCNTL 2

POR Value = 00H

Name	Range	Description
testcontrol[7:0]	7:0	Reserved.

MAC Control Register (R/W)

Mnemonic: MACCNTL

POR Value = 00H

Name	Range	Description
Rsvd [7:5]	7-5	Reserved
ignore next packet	4	**ignore next packet**: This bit asserts the ignore_next_pkt pin on the 2.5GMAC IP core. See Table 2-1 on page 9 for more information.
tx_fifo_empty	3	**tx_fifo_empty**: This bit gets ORed with the FIFO empty signal. The ORed output sets the tx_fifoempty pin on the 2.5GMAC IP core. See Table 2-1 on page 9 for more information on this 2.5GMAC IP core signal. Note by setting this bit you can mimic the Tx FIFO being empty. Also note that the application design only has one loopback FIFO. So the Tx FIFO is the same as the Rx FIFO.
rx_fifo full	2	**rx_fifo full**: This bit gets ORed with the FIFO full signal. The ORed output sets the rx_fifo_full pin on the 2.5GMAC IP core. See Table 2-1 on page 9 for more information on this 2.5GMAC IP core signal. Note by setting this bit you can mimic the rx fifo being full. Also note that the application design only has one loopback FIFO. So the Tx FIFO is the same as the Rx FIFO.
fifo control frame	1	**fifo control frame**: This bit sets the tx_fifoctrl pin on the 2.5GMAC IP core. See Table 2-1 on page 9 for more information on this 2.5GMAC IP core signal.
send pause request	0	**send pause request**: This bit gets ORed with the Tx FIFO almost full signal. The ORed output sets the tx_sndpausreq pin on the 2.5GMAC IP core. See Table 2-1 on page 9 for more information on this 2.5GMAC IP core signal. When this bit is set High OR the FIFO is almost full tx_sndpausreq is asserted, otherwise tx_sndpausreq is de-asserted.

Pause Timer Register - Low Byte (R/W)

Mnemonic: PAUSTMRL

POR Value = 00H

Name	Range	Description
pause timer Low bits [7:0]	7:0	**Pause Timer Low bits**: These reg bits set the tx_sndpaustim[7:0] pins on the 2.5GMAC IP core. See Table 2-1 on page 9 for more information on this 2.5GMAC IP core signal.

Pause Timer Register - High Byte (R/W)

Mnemonic: PAUSTMRH

POR Value = 00H

Name	Range	Description
pause timer High bits [7:0]	7:0	**Pause Timer High bits**: These bits set the tx_sndpaustim[15:8] pins on the 2.5GMAC IP core. See Table 2-1 on page 9 for more information on this 2.5GMAC IP core signal.

FIFO Almost Full Threshold Register - Low (R/W)

Mnemonic: FIFOAFTL

POR Value = 00H

Name	Range	Description
FIFO almost Full Low bits [7:0]	7:0	**FIFO almost Full Low bits**: These bits set the loop back FIFO Almost Full threshold [7:0] bits

FIFO Almost Full Threshold Register - High (R/W)

Mnemonic: FIFOAFTH

POR Value = 00H

Name	Range	Description
Rsvd [7:1]	7 - 1	Reserved.
FIFO almost Full High bit [0]	0	**FIFO almost Full High bit**: This bit sets the loop back FIFO Almost Full threshold [8] bit.

FIFO Almost Empty Threshold Register - Low (R/W)

Mnemonic: FIFOAETL

POR Value = 00H

Name	Range	Description
FIFO almost Empty Low bits [7:0]	7:0	**FIFO almost Empty Low bits**: These bits set the loop back FIFO Almost Empty threshold [7:0] bits

FIFO Almost Full Threshold Register - High (R/W)

Mnemonic: FIFOAETH

POR Value = 00H

Name	Range	Description
Rsvd [7:1]	7 - 1	Reserved.
FIFO almost Empty High bit [0]	0	**FIFO almost Empty High bit**: This bit sets the loop back FIFO Almost Empty threshold [8] bit.

Rx Status Register (RO/COR)

Mnemonic: RXSTATUS

POR Value = 00H

Name	Range	Description
Rsvd [7:2]	7:2	Reserved.
rx_error	1	**rx_error**: This bit is set high and latched if the rx_fifo_error signal is asserted on 2.5GMAC IP core pin. See Table 2-1 on page 9 for more information on this 2.5GMAC IP core signal.
rx_fifo_error	0	**rx_fifo_error**: This bit is set high and latched if the rx_error signal is asserted on 2.5GMAC IP core pin. See Table 2-1 on page 9 for more information on this 2.5GMAC IP core signal.

TXSTATUS (RO/COR)

Mnemonic: TXSTATUS

POR Value = 00H

Name	Range	Description
Rsvd [7:2]	7:2	Reserved.
tx_fifo_full	1	**tx_fifo_full**: This bit is set high and latched if the tx_fifo_full signal from the Loop back FIFO is asserted.
tx_discfrm	0	**tx_discfrm**: This bit is set high and latched if the tx_discfrm signal is asserted on 2.5GMAC IP core pin. See Table 2-1 on page 9 for more information on this 2.5GMAC IP core signal.

The counter registers listed in Table 5-2 are all 16 bits with 8 bit low and 8 bit high address locations. The counters count different Rx and Tx statistics as defined by the tx_statvec and rx_statvec statistics vectors define in Table 2-1 on page 9. All counters have a power-on reset (POR) value of 0x0000.

Table 5-2. Counter Registers

Address	Register Description	Mnemonic	Type
0x0800c	Rx Packet Ignored Counter Register	RXPICNT	(RO/COR)
0x0800e	Rx Length Check Error CouNTer	RXLCECNT	(RO/COR)
0x08010	Rx Long Frames CouNTer Register	RXLFCNT	(RO/COR)
0x08012	Rx Short Frames CouNTer Register	RXSFCNT	(RO/COR)
0x08014	Rx IPG violations CouNTer Register	RXIPGCNT	(RO/COR)
0x08016	Rx CRC errors CouNTer Register	RXCRCCNT	(RO/COR)
0x08018	Rx OK packets CouNTer Register	RXOKCNT	(RO/COR)
0x0801a	Rx Control Frame CouNTer Register	RXCFCNT	(RO/COR)
0x0801c	Rx Pause Frame CouNTer Register	RXPFCNT	(RO/COR)

0x0801e	Rx Multicast Frame CouNTer Register	RXMFCNT	(RO/COR)
0x08020	Rx Broadcast Frame CouNTer Register	RXBFCNT	(RO/COR)
0x08022	Rx VLAN tagged Frame CouNTer Register	RXVFCNT	(RO/COR)
0x08024	Tx Unicast Frame CouNTer Register	TXUFCNT	(RO/COR)
0x08026	Tx Pause Frame CouNTer Register	TXPFCNT	(RO/COR)
0x08028	Tx Multicast Frame CouNTer Register	TXMFCNT	(RO/COR)
0x0802a	Tx Broadcast Frame CouNTer Register	TXBFCNT	(RO/COR)
0x0802c	Tx VLAN tagged Frame CouNTer Register	TXVFCNT	(RO/COR)
0x0802e	Tx BAD FCS Frame CouNTer Register	TXBFCCNT	(RO/COR)
0x08030	Tx Jumbo Frame CouNTer Register	TXJFCNT	(RO/COR)

Table 5-3 lists test application I/Os.

Table 5-3. Counter Registers

FPGA Signal Name	FPGA Pin	Notes
reset_n	Input	Active Low
rx_clk	Input	Rx GMII clock (Rx word clock)
rx_dv[1:0]	Input	Rx GMII data valid
rx_er[1:0]	Input	Rx GMII error
rxd[15:0]	Input	Rx GMII data bits 0 to 15
sys_clk	Input	System clock (at word clock rate)
tx_clk	Input	Tx GMII clock (Tx word clock)
tx_en[1:0]	Output	Tx GMII enable
tx_er[1:0]	Output	Tx GMII error
txd[15:0]	Output	Tx GMII data bits 0 to 15
rxmac_clk	Output	Rx MAC clock – used internally can be brought out
txmac_clk	Output	Tx MAC Clock – used internally can be brought out
hclk	Input	Host bus clock

Code Listing for Multicast Bit Selection Hash Algorithm in C Language

The code listing below is to aid software developers in programming the multicast tables in the 2.5GMAC IP core when the core is programmed to receive multicast frames.

When a software developer wishes to accept a specific multicast address, they should follow the hash algorithm illustrated in the C language code listing to determine which filter bit in the multicast registers to set. The C algorithm returns the multicastcast_table register index (0 to 7) as well as the bit within the register that needs to be set (0 to 7) based on a given multicast destination address input to the algorithm. Several bits can be set to accept several multicast addresses. If all 64 multicast filter register bits are set to 1, then all received multicast addresses will be passed to the MAC client interface.

```
#include <stdio.h>
#include <stdlib.h>

//Hexadecimal equivalent of the CRC
//equn.
#define CRC_POLYNOMIAL 0x04c11db6

int main(int argc, unsigned char *argv[])
{
    //The Multicast address is held in a 6 byte
    //array
```

```
unsigned char multi_addr[6];
// variables
unsigned long int crc;
unsigned int val;
int i, j, bit;
int carry;
int register_no, register_bit;
crc = 0xffffffff;

// check number of arguments
if (argc != 7) {
    printf("Invoke eth_crc with arguments specifying a MAC Address.\n");
    printf("Use hex format and blanks.\n");
    printf("Example:\n");
    printf("eth_crc 01 A2 B3 C4 D5 E6\n");
    system("PAUSE");

    return 1;
}

printf("\n DA   ");
// Input data from command line
for (j=0; j<6; j++)
{
    sscanf(argv[j+1], "%x", &val);
    multi_addr[j] = (unsigned char) val;
    printf("%s", argv[j+1]);
}
printf("\n");

// check for multicast destination address
if ((multi_addr[0] & 0x01) == 0)
{
    printf(" Not a multicast address\n");
    printf(" Bit 0 of MSB must be 1\n\n");
    system("PAUSE");
    return 1;
}

// The following loops create the 32-bit crc
// value.
// loop through each byte of the address.
for(i=0; i<6; i++)
{
    // Loop through each byte bit of that byte.
    for(bit=0; bit<8; bit++)
    {
        carry = (crc >> 31)^((multi_addr[i] & (1 << bit)) >> bit);
        crc <<= 1;
        if (carry)
        crc = (crc ^ CRC_POLYNOMIAL) | carry;
    }
}

// Extract the middle 6 bits from the MSB of the 32bit CRC value,
// this six bit value is used to index a
// unique filter bit.
printf(" crc  %lx \n", crc);
crc >>= 25;  // 2.5GMAC refers to Bit 30..25
```

```
    crc &= 0x3F; // mask six bit

    //Find the multicast register number and
    //bit of that register to set.
    printf(" hash %lx \n", crc);
    register_no = crc >> 3;
    register_bit = crc & 7;
    printf (" register_no  %lx\n register_bit %lx \n\n", register_no, register_bit);

    system("PAUSE");
    return 0;
}
```

The 2.5GMAC IP core has been validated using an Ethernet application design targeted to a LatticeECP3-95, speed grade -8 FPGA. The FPGA design included this IP core, a data path loopback function on the MAC client interface, a 1000BASE-X PCS on the MAC GMII interface, a SERDES operating at 1.25 Gbps, and miscellaneous logic to control and read configuration/status registers. The FPGA was mounted on a demo board that included a 1000BASE-X SFP module. An optical cable was used to link the demo board to an external Spirent Testcenter Ethernet Analyzer. Figure 6-1 illustrates the setup.

Figure 6-1. 2.5GMAC Hardware Validation Setup

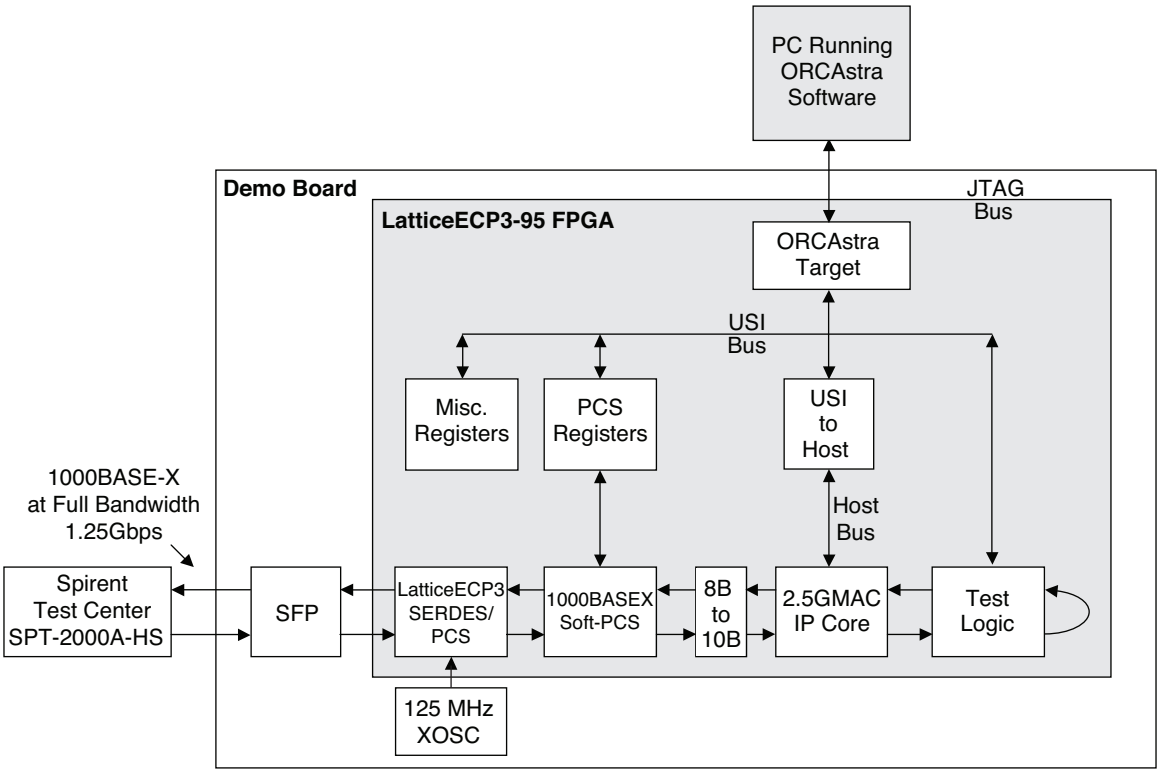


Table 1 lists the tests performed during the validation process.

Table 1. Validation Test List

Packet Flow, Unicast
Packet Flow, Multicast
Packet Flow, Broadcast
Packet Flow, VLAN Tagged
Packet Flow, Jumbo
Packet Flow, Short
Packet Flow, Long Duration
Packet Flow, Sparse Bandwidth, Unicast
Packet Flow, Over Bandwidth, IPG=8-12
Packet Flow, Force CRC Error,discard CRC
Packet Flow, Force CRC Error, pass CRC
Flow Cntl, RX Pause, RX Drop, TX Normal
Flow Cntl, RX Pause, RX Pass, TX Normal
Flow Cntl, TX Pause, Flow Cntl Disabled
Flow Cntl, TX Pause, Flow Cntl Enabled
Addr Filtering, Dest Addr = MAC Addr
Addr Filtering, Dest Addr != MAC Addr
Promiscuous Mode
MAC Statistics Check
MAC Status Registers Check
MAC Control Registers Check

Chapter 7:

Support Resources

This chapter contains information about Lattice Technical Support, additional references, and document revision history.

Lattice Technical Support

There are a number of ways to receive technical support.

E-mail Support

techsupport@latticesemi.com

Local Support

Contact your nearest Lattice Sales Office.

Internet

www.latticesemi.com

IEEE

IEEE offers publications and technology standards on its web site at www.ieee.org.

References

- HB1009, *LatticeECP3 Family Handbook*

Revision History

Date	Document Version	IP Core Version	Change Summary
March 2012	01.0	1.0	Initial release.
May 2014	1.1	1.0	Updated LatticeECP3 FPGAs section. Changed part number in Ordering Part Number.
			Updated Lattice Technical Support information.

Appendix A:

Resource Utilization

This appendix gives resource utilization information for the LatticeECP3 FPGA using the 2.5GMAC IP core.

LatticeECP3 FPGAs

Table A-1. Performance and Resource Utilization[1]

Slices	LUTs	Registers	EBRs	External Pins	f_{MAX} (MHz)
1480	2100	1290	1	203	156.25

1. Performance and utilization data are generated targeting an LFE3-150EA-8FN1156C device using Lattice Diamond 1.3 and Synplify Pro for Lattice E-2011.03L software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP3 family.

Ordering Part Number

The Ordering Part Number (OPN) for the 2.5G Ethernet MAC IP core targeting LatticeECP3 devices is 2PT5-MAC-E3-U.