Triple Speed Ethernet MAC Core User Guide

UG-CORE-TSEMAC-v6.1
December 2025
www.efinixinc.com

Contents

Introduction

The Triple Speed Ethernet MAC core is a configurable core that complies with the IEEE Std. 802.3-2008 specification. The core supports Ethernet speeds of 2500 Mbps, 1000 Mbps, 100 Mbps, and 10 Mbps, with full duplex transfer mode.

Note: Starting with IP version 7.0:

- Packets filtered by address filtering or broadcast filtering are silently dropped when `crc_fwd` = `0`
- When AXI Data Width = 8 and TX/RX FIFO is disabled, **Cut Through** mode is implemented to achieve minimum latency.
  - TX and RX AXI4-Stream interface operates on `tx_mac_aclk` and `gm_rx_c` clock domains respectively in **Cut Through** mode. Hence, the `tx_axis_clk` and `rx_axis_clk` ports are not used and connect to logic `0`.
  - If `tx_axis_mac_tvalid` is deasserted before the end of the frame in **Cut Through** mode, the frame is terminated and flagged as a bad frame. See Terminating Bad TX Frames on page 32.
  - See Clock Sources on page 9 and Cut Through Mode on page 29 for more details.

Use the IP Manager to select IP, customize it, and generate files. The Triple Speed Ethernet MAC core has an interactive wizard to help you set parameters. The wizard also has options to create a testbench and/or example design targeting an Efinix® development board.

Features

- Compliant with the IEEE Std. 802.3-2008 specification
- Supports VLAN frame and jumbo frame
- Supports 2500 Mbps, 1000 Mbps, 100 Mbps, and 10 Mbps in full-duplex mode
- 8-bit, 16-bit, and 32-bit AXI4-Stream user interface for packet transfer
- RGMII, MII, RMII, GMII, and SGMII PHY interfaces
- Optional internal FIFO buffers
- Programmable source and destination MAC addresses
- Programmable inter packet gap (IPG)
- Automatic padding on short frames
- Frame check sequence or CRC generation, checking, and forwarding
- Frame length check against user-defined MTU frame length for received packets
- Frame length check against the LENGTH field for received packets
- Supports broadcast, unicast, and multicast address filtering
- Supports pause frame flow control
- Management data I/O (MDIO) for PHY device management
- AXI4-Lite or APB user interface for MAC configuration register access
- Statistic reporting
- Includes an example design targeting:
  - Trion T120 BGA324 Development Board
  - Titanium Ti60 F225 Development Board
  - Titanium Ti375 N1156 Development Board

Device Support

Table 1: Triple Speed Ethernet MAC Core Device Support

FPGA Family	Supported Device
Trion	All
Titanium	All
Topaz	All

Resource Utilization

Note: The resources and performance values provided are based on some of the supported FPGAs. These values are just guidance and may change depending on the device resource utilization, design congestion, and user design.

Table 2: Resource Utilization

FPGA	Mode	LUTs	FFs	Memory Block	DSP Block	Efinity® Version[1]
T120F324	Cut-Through Mode (GMII)	2,131 / 112,128 (1.90%)	1,369 / 112,128 (1.22%)	12 / 1,056 (1.14%)	0 / 320 (0%)	2025.1
	Store-Forward Mode (GMII)	2,568 / 112,128 (2.29%)	1782 / 112,128 (1.59%)	23 / 1,056 (2.18%)	0 / 320 (0%)	
Ti375N1156	Cut-Through Mode (SGMII)	2,820 / 362,880 (0.77%)	1,515 / 362,880 (0.42%)	12 / 2,688 (0.45%)	0 / 1,344 (0%)	
	Store-Forward Mode (SGMII)	3,401 / 362,880 (0.94%)	2,025 / 362,880 (0.56%)	21 / 2,688 (0.78%)	0 / 1,344 (0%)	
Tz75N676	Cut-Through Mode (SGMII)	2,820 / 129,600 (2.17%)	1,515 / 129,600 (1.17%)	12 / 960 (1.25%)	0 / 480 (0%)	
	Store-Forward Mode (SGMII)	3,401 / 129,600 (2.62%)	2,025 / 129,600 (1.56%)	21 / 960 (2.19%)	0 / 480 (0%)	

Note: The resource utilization data is derived based on the default settings.

Release Notes

You can refer to the IP Core Release Notes for more information about the IP core changes. The IP Core Release Notes are available in the Efinity page of the Support Center under each Efinity software release version.

Note: You must be logged in to the Support Center to view the IP Core Release Notes.

[1] System Verilog 2005.

Functional Description

The Triple Speed Ethernet MAC core does the following functions:

- Convert TX data packets from the AXI4-Stream to the PHY interface format.
- Decodes RX data packets from PHY interface format to AXI4-Stream.
- Generates the pause frame to be transmitted at the PHY interface.
- Identifies the pause frame and implements priority flow control through the pause mechanism based on the decoded `PAUSE_QUANT`.
- Drops RX frames with a mismatched address through a bit-wise address filtering mask.
- Drops RX frames with broadcast address.

The Triple Speed Ethernet MAC core supports multiple industry-standard PHY interfaces. Table 3: Supported PHY Interfaces and Speed on page 7 provides an overview of the data bus widths and supported speeds.
For detailed descriptions, refer to TX PHY Interface on page 20 and RX PHY Interface on page 25.

Table 3: Supported PHY Interfaces and Speed

PHY Interface	Data Bus Width	Speed			
		2.5 G	1 G	10/100/1000M	10/100M
RGMII	4-bit (Double data rate)	-	✓	✓	✓
MII	4-bit	-	-	-	✓
RMII	2-bit	-	-	-	✓
GMII	8-bit	-	✓	-	-
SGMII	16-bit	✓	✓	✓	✓

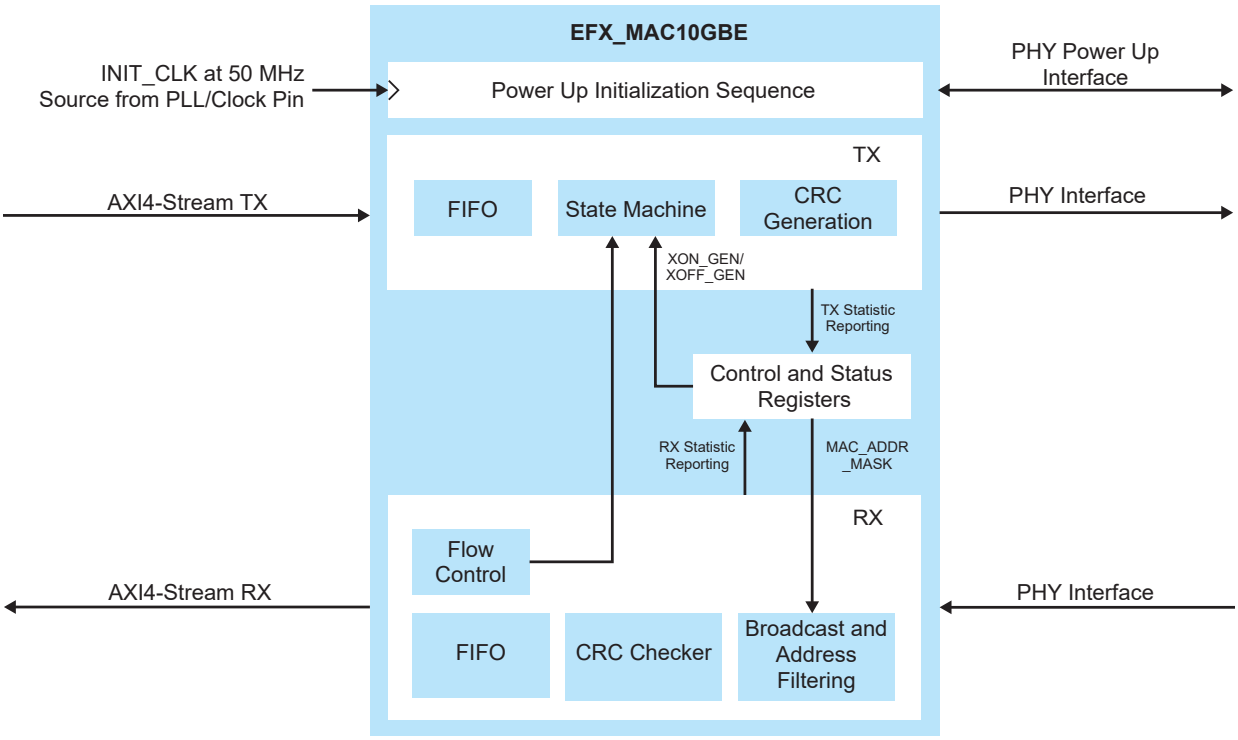
Block Diagram

The Triple Speed Ethernet MAC core or `efx_mac1gbe` comprises of the following modules:

- Power-up sequence to handshake with the FPGA transceiver in SGMII mode.
- AXI4-stream user interface
- TX engine with CRC generation
- RX engine with CRC checker
- PHY interface
- Statistics reporting
- Priority flow control module
- Broadcast and address filtering

Figure 1: Triple Speed Ethernet MAC Core Sub-Modules on page 8 shows the modules inside the Triple Speed Ethernet MAC core.

Figure 1: Triple Speed Ethernet MAC Core Sub-Modules



Note: The power-up sequence module is instantiated for SGMII mode only. Details of the power-up sequence are described in Power Up Handshake with FPGA Transceiver on page 14.

Clock Sources

The Triple Speed Ethernet MAC core clock sources are shown in the following table.

Table 4: Clock Sources in the Triple Speed Ethernet MAC Core

Clock Name	Frequency		Description
tx_axis_clk	Refer to **Table 5: Minimum Clock Frequency Requirement for tx_axis_clk and rx_axis_clk** on page 10.		AXI4-Stream TX clock
rx_axis_clk	Refer to **Table 5: Minimum Clock Frequency Requirement for tx_axis_clk and rx_axis_clk** on page 10.		AXI4-Stream RX clock
s_axi_aclk	Trion: ≤ 100 MHz Others: ≤ 200 MHz		Register access interface clock
init_clk	≤ 50 MHz		To facilitate the power-up sequence of the FPGA transceiver PHY. Can be sourced from the PLL or CLKIN pin.
RGMII/ GMII/ SGMII			
tx_mac_aclk	125 MHz		*RGMII/ GMII* Transmit path reference clock. Provides timing reference for the RGMII and GMII transmit interface.
	2500 Mbps	156.25 MHz	*SGMII* • Sourced from the FPGA transceiver forwarded clock to be source synchronous and eliminate additional clock PPM. • The same forwarded clock is also used to clock the 1G PCS in the FPGA transceiver.
	1000 Mbps	62.5 MHz	
	100 Mbps	6.25 MHz	
	10 Mbps	0.625 MHz	
gm_rx_c	125 MHz		*GMII* GMII receive clock sourced from the PHY.
	2500 Mbps	156.25 MHz	*SGMII* • Sourced from the FPGA Transceiver Forwarded Clock to be Source Synchronous and eliminate additional clock PPM. • The same Forwarded Clock is also used to clock the 1G PCS in the FPGA Transceiver.
	1000 Mbps	62.5 MHz	
	100 Mbps	6.25 MHz	
	10 Mbps	0.625 MHz	
rgmii_rxc	1000 Mbps	125 MHz	RGMII receive clock sourced from the PHY.
	100 Mbps	25 MHz	
	10 Mbps	2.5 MHz	
RMII			
rmii_clk_ref	50 MHz		RMII reference clock.
MII			
mii_txc	100 Mbps	25 MHz	MII transmit clock sourced from the PHY.
	10 Mbps	2.5 MHz	
mii_rxc	100 Mbps	25 MHz	MII receive clock sourced from the PHY.
	10 Mbps	2.5 MHz	

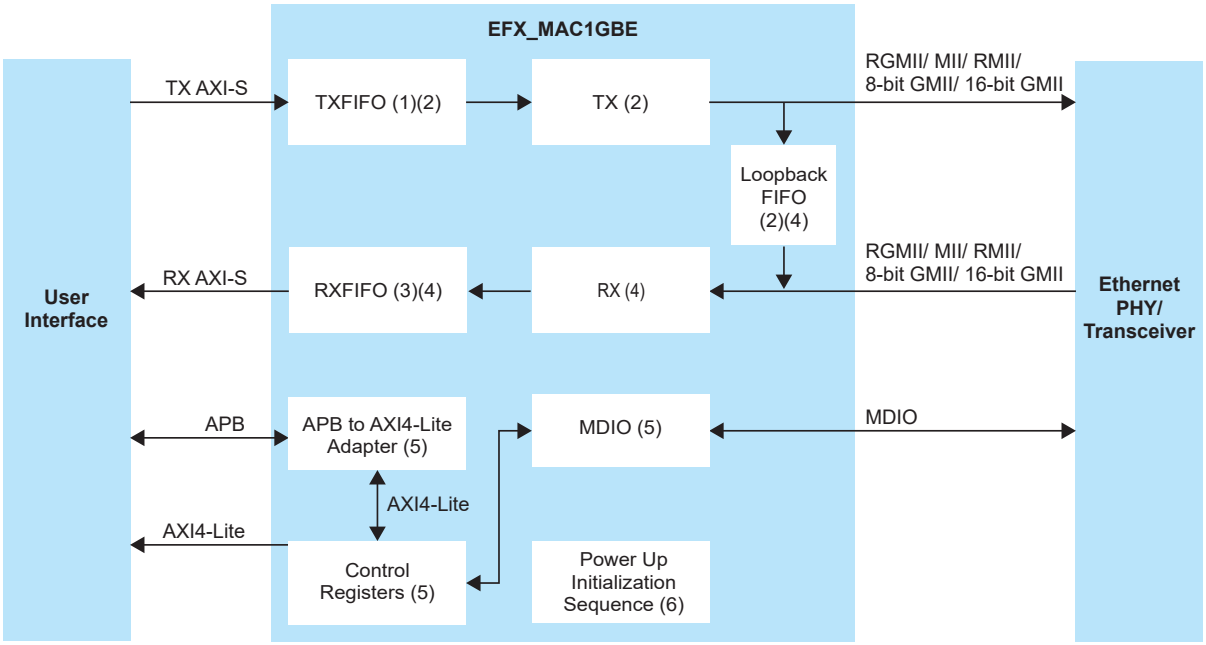
To prevent the AXI4-stream interface from limiting the performance, its clocks must meet the minimum frequency requirement listed in Table 5: Minimum Clock Frequency Requirement for tx_axis_clk and rx_axis_clk on page 10.

Table 5: Minimum Clock Frequency Requirement for `tx_axis_clk` and `rx_axis_clk`

Speed (Mbps)	AXI4-Stream Data Width	Min. Clock Frequency (MHz)
2500	8	312.5
	16	156.25
	32	78.125
1000	8	125
	16	62.5
	32	31.25
100	8	12.5
	16	6.25
	32	3.125
10	8	1.25
	16	0.625
	32	0.3125

Figure 2: Triple Speed Ethernet MAC Core Clock Domain on page 10 shows the clock domains in the Triple Speed Ethernet MAC core.

Figure 2: Triple Speed Ethernet MAC Core Clock Domain



Notes: (1) tx_axis_clk
(2) tx_mac_aclk/ mii_txc/ rmii_clk_ref
(3) rx_axis_clk
(4) rgmii_rxc/ mii_rxc/ rmii_clk_ref/ gm_rx_c
(5) s_axi_clk
(6) init_clk

Note: In **Cut Through** mode, the AXI4-stream interface operates on the `tx_mac_aclk` and `gm_rx_c` clock domains. The `tx_axis_clk` and`rx_axis_clk` ports are not used in this mode and should be logic `0`.

Reset Signals

The Triple Speed Ethernet MAC core has 3 reset signals. These signals are asynchronous. `init_rst_n` is active low, but `mac_reset` and `proto_reset` are active high.

- `mac_reset` (Global Reset)
- `proto_reset` (Protocol Reset)
- `init_rst_n` (Initialization Reset)

During the FPGA power-up, intitalize `mac_reset` and `proto_reset` to `1`, and `init_rst_n` to `0` to ensure the Triple Speed Ethernet MAC core is held in a known reset state.

- `mac_reset` (Global Reset)
  - Resets all logic within the Triple Speed Ethernet MAC core, including control and status registers.
  - Only the power-up sequence module remains unaffected and operational.
- `proto_reset` (Protocol Reset)
  - Resets both the TX and RX data paths, as well as the statistics counters.
  - Does not affect the power-up sequence module.
- `init_rst_n` (Initialization Reset)
  - Used when pairing the Triple Speed Ethernet MAC core with an FPGA transceiver in SGMII mode.
  - Requires a free-running `init_clk`.
  - Upon `IN_USER`, deassert `init_rst_n` to start the power-up handshake with the FPGA ransceiver.
  - Must remain deasserted throughout normal operation.
  - Detailed power-up sequence is described in **Power Up Handshake with FPGA Transceiver** on page 14.

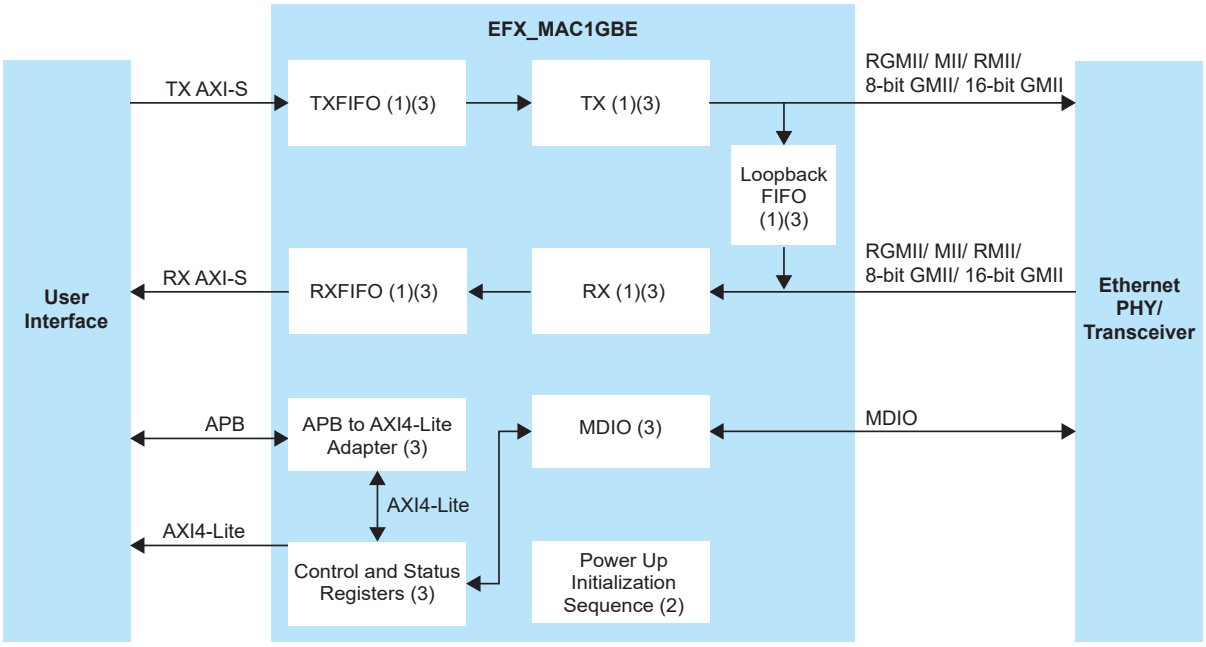
For synchronization reset, all three resets are asynchronous assertion or synchronous deassertion.

- `init_rst_n` is synchronized internally to `init_clk`.
- `mac_reset` and `proto_reset` are synchronized internally to their respective clock domains.
- Each reset synchronizer introduces a 2 – 3 clock cycle latency upon deassertion.

Note: To ensure proper operation after reset deasssertion, you must delay data transmission by at least 3 clock cycles following the release of the reset signal. This delay allows sufficient time for the internal reset synchronizers to stabilize before any AXI4-Stream or PHY interface activity begins. Failure to observe this delay may result in undefined behavior or corrupted frame transmission during the startup process.

The Figure 3: Triple Speed Ethernet MAC Core Reset Domains on page 13 shows the reset domains in the Triple Speed Ethernet MAC core.

Figure 3: Triple Speed Ethernet MAC Core Reset Domains



Notes: (1) proto_reset
(2) init_rst_n
(3) mac_reset

Power Up Handshake with FPGA Transceiver

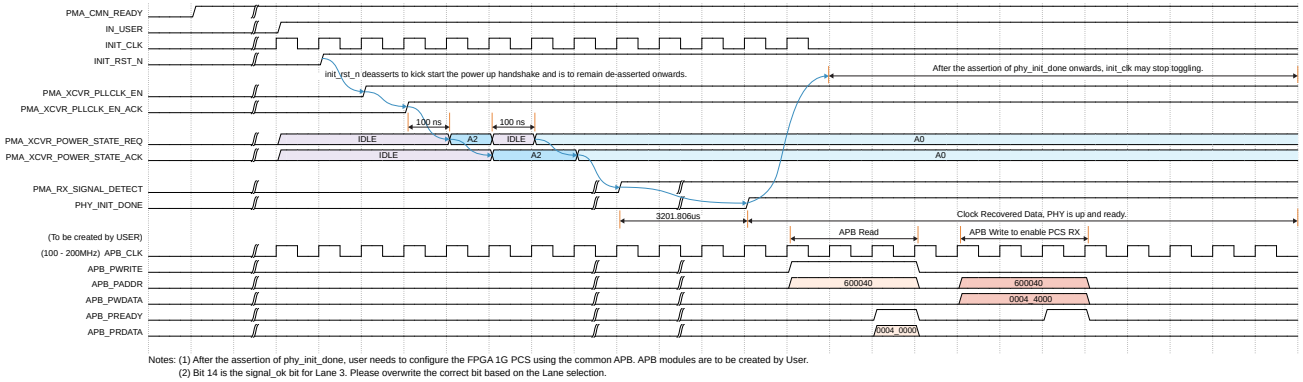
Before starting the operation of the Triple Speed Ethernet MAC core, it needs to perform the power-up handshake with the FPGA transceiver.

The Triple Speed Ethernet MAC core includes a built-in power-up sequence module to handshake with the FPGA transceiver PHY. For the details on the power-up sequence in the FPGA transceiver, refer to the power-up sequence chapter of the **Titanium SGMII 1G and 2.5G User Guide**. **Figure 4: Power Up Handshake with the FPGA Transceiver PHY and PCS** on page 14 shows the power-up handshake between the Triple Speed Ethernet MAC core and the FPGA transceiver.

The power-up sequence between the Triple Speed Ethernet MAC core and the FPGA transceiver PHY is clocked by `init_clk`. The assertion of `phy_init_done` indicates that the FPGA transceiver PHY is CDR-lock-to-data.

After the assertion of `phy_init_done`, you need to configure the FPGA transceiver PCS which uses the common APB interface. The APB interface is a common interface that is shared across all the 4 lanes within a Quad. You need to create APB modules to configure the PCS, based on the register mapping in the **Titanium SGMII 1G and 2.5G User Guide**. **Figure 4: Power Up Handshake with the FPGA Transceiver PHY and PCS** on page 14 includes an illustration of APB Write to enable PCS RX in lane 2.

Figure 4: Power Up Handshake with the FPGA Transceiver PHY and PCS



Notes: (1) After the assertion of phy_init_done, user needs to configure the FPGA 1G PCS using the common APB. APB modules are to be created by User.
(2) Bit 14 is the signal_ok bit for Lane 3. Please overwrite the correct bit based on the Lane selection.

Important: When the FPGA PHY is in reset, both `mac_reset` and `init_rst_n` need to be asserted too. Once the FPGA PHY recovers from the reset, the user needs to repeat the entire power-up handshake by de-asserting the `init_rst_n` to restart the power-up sequence.

Ethernet Packets, Frames, and IPG

Figure 5: 802.3 Ethernet Packet and Frame Structure for Non-VLAN Tagged Frame on page 15 illustrates the Ethernet packet and frame structure for non-VLAN tagged frame.

Figure 5: 802.3 Ethernet Packet and Frame Structure for Non-VLAN Tagged Frame

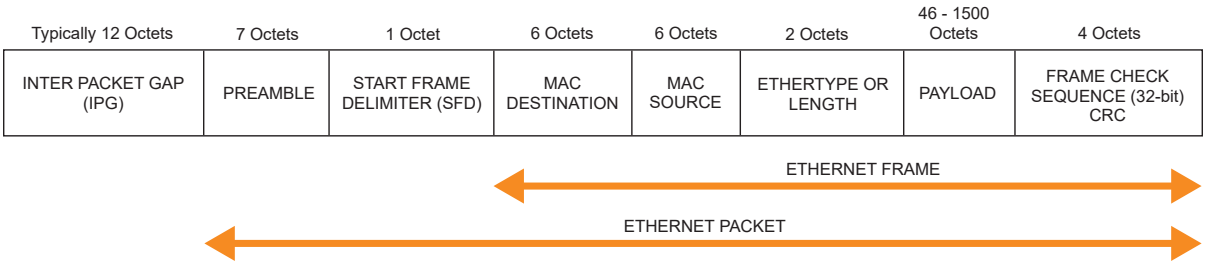
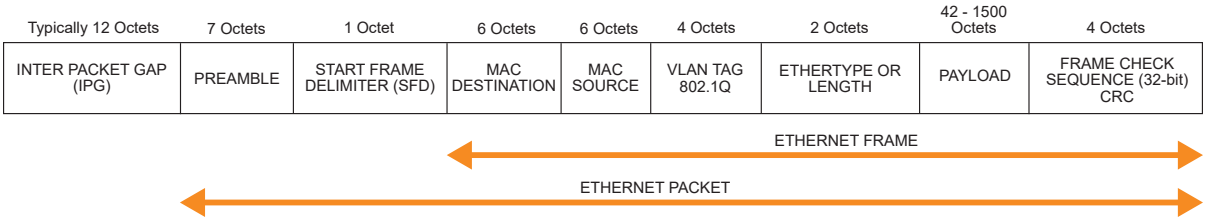


Figure 6: 802.3 Ethernet Packet and Frame Structure for VLAN Tagged Frame on page 15 illustrates the Ethernet packet and frame structure for VLAN tagged frame.

Figure 6: 802.3 Ethernet Packet and Frame Structure for VLAN Tagged Frame



User Interface AXI4-Stream

AXI4-stream is a user interface comprising TX channel and RX channel.

TX AXI ST Interface

At TX AXI ST interface, transmission starts when `tx_axis_mac_tready = 1`. The assertion of `tx_axis_mac_tready` indicates that Triple Speed Ethernet MAC core is ready to accept the TX data streaming from the user.

To transmit, assert `tx_axis_mac_tvalid`, along with `tx_axis_mac_tdata`, `tx_axis_mac_tstrb`, `tx_axis_mac_tlast`, and `tx_axis_mac_tuser`.

- `tx_axis_mac_tdata` contains the data streaming to be transmitted to the selected PHY interface. Tranmission must begin with header information – destination address, source destination, ethernet length or type, and followed by the payload data.
- `tx_axis_mac_tlast` indicates the final transfer in the data streaming.
- `tx_axis_mac_tstrb` indicates the positional validity of the data byte within the data streaming.
- `tx_axis_mac_tuser` indicates an error in the data streaming.

For Ethernet application, `tx_axis_mac_tdata` must contain streaming data, i.e., data needs to be streaming and continuous, During streaming, all bytes must be valid, i.e., `tx_axis_mac_tstrb` = all 1s, except for the final transfer when `tx_axis_mac_tlast` is asserted. At `tx_axis_mac_tlast`, `tx_axis_mac_tstrb` indicates the number of valid bytes in the final word, starting from the least significant byte. For example, refer to **Figure 7: Streaming Data at TX AXI ST Interface** on page 16.

In the event of a non-data streaming, it is interpreted as a bad frame and is dropped. Similarly, asserting `tx_axis_mac_tuser` during transmission results in a bad frame. Refer to **Terminating Bad TX Frames** on page 32.

When `tx_axis_mac_tready = 0`, the Triple Speed Ethernet MAC core is not ready to accept any TX data. Hence, you need to hold all current `tx_axis_mac`signals, which are `tx_axis_mac_tdata`, `tx_axis_mac_tlast`, `tx_axis_mac_tvalid`, `tx_axis_mac_tstrb`, and `tx_axis_mac_tuser`, until `tx_axis_mac_tready` returns to `1`. Refer to **Figure 8: User to Retain tx_axis_mac Signals Until tx_axis_mac_tready Recovers** on page 17.

Figure 7: Streaming Data at TX AXI ST Interface

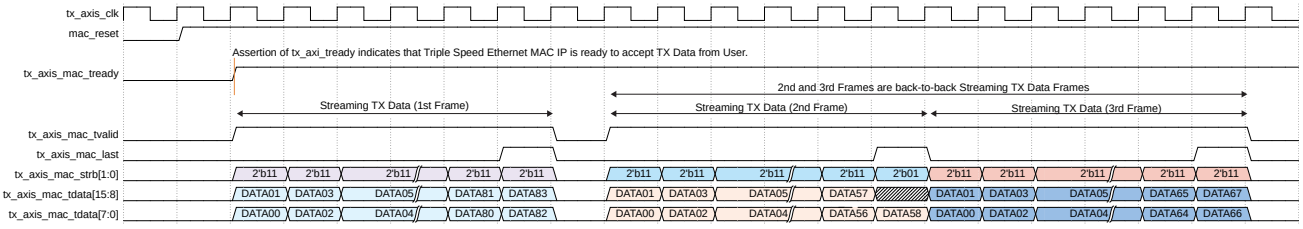
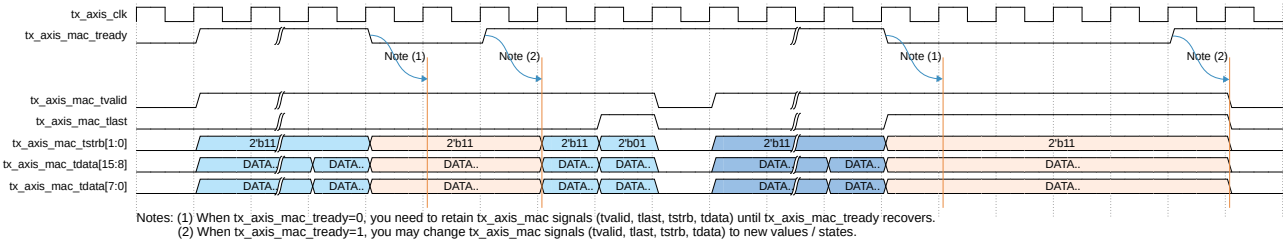


Figure 8: User to Retain `tx_axis_mac` Signals Until `tx_axis_mac_tready` Recovers



In **Cut Through** mode, if `tx_axis_mac_tvalid` goes low before end-of-frame, the Triple Speed Ethernet MAC core ends the transmission with an error frame and drops the rest of the packet. However, in **Cut Through mode with width-adaptation** or **Store Forward** mode, when `tx_axis_mac_tvalid = 0`, all `tx_axis_mac` signals for that cycle are ignored but no error occurs. Refer to Terminating Bad TX Frames on page 32.

Note: When the Triple Speed Ethernet MAC core is configured in **Cut Through mode with width-adaptation** or **Store Forward** mode, the TX AXI ST interface operates in the `tx_axis_clk` domain. You must ensure that the `tx_axis_mac_tvalid`, `tx_axis_mac_tdata`, `tx_axis_mac_tstrb`, `tx_axis_mac_tlast`, and `tx_axis_mac_tuser` signals are properly registered and synchronous to `tx_axis_clk` before entering the Triple Speed Ethernet MAC core. When the Triple Speed Ethernet MAC core is configured in **Cut Through** mode, the TX AXI ST interface operates in the `tx_mac_aclk` domain, and these signals must be synchronous to `tx_mac_aclk`.

Note: The `tx_axis_mac_tstrb` signal in the Triple Speed Ethernet MAC core functions as TKEEP in AXI4-Stream terminology, indicating a valid byte. The name is preserved as TSTRB for backward compatibility.

RX AXI ST Interface

At the RX AXI ST interface, `rx_axis_mac_tvalid` indicates the validity of the received data frame. `rx_axis_mac_tdata` carries the data decoded from the selected PHY Interface, while `rx_axis_mac_tlast` marks the final transfer of the frame. `rx_axis_mac_tstrb` indicates the positional validity of the data byte within the RX data frame, and `rx_axis_mac_tuser` signals that the received frame contains an error. See Erroneous RX Frame on page 35 for further details.

When `crc_fwd = 1`, the `rx_axis_mac_tdata` port includes both the RX data frame and CRC octets. More details on CRC forwarding are provided in CRC Generation and Check on page 19 and Figure 10: RX Data Frame and CRC Octets at AXI ST Interface when crc_fwd = 1 on page 18.

If the AXI4-Stream interface bandwidth exceeds the PHY interface bandwidth (for example, when using a wider AXI data width or a faster AXI clock, the RX AXI stream may appear non-streaming even for valid frames. This occurs because the PHY provides data at a lower rate than the AXI interface can accept, resulting in intermittent valid cycles (`rx_axis_mac_tvalid` deasserted between data beats). This behavior is expected and does not indicate an error.

Note: Back pressure on the RX path through `rx_axis_mac_tready` is not supported and may result in data corruption. Therefore, it is recommended to initialize `rx_axis_mac_tready` to `1` to ensure reliable operation.

Figure 9: RX Data Frame at AXI ST Interface when `crc_fwd = 0`

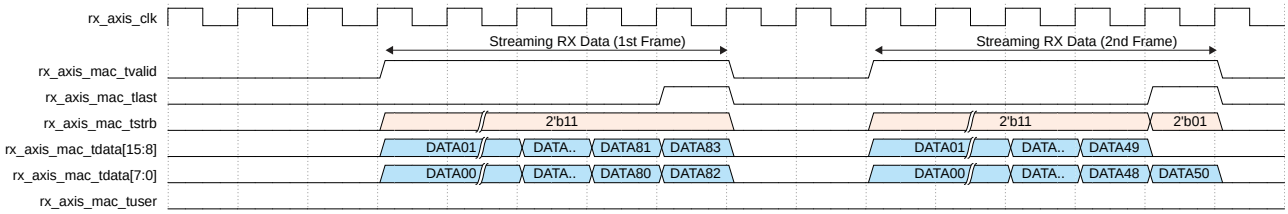
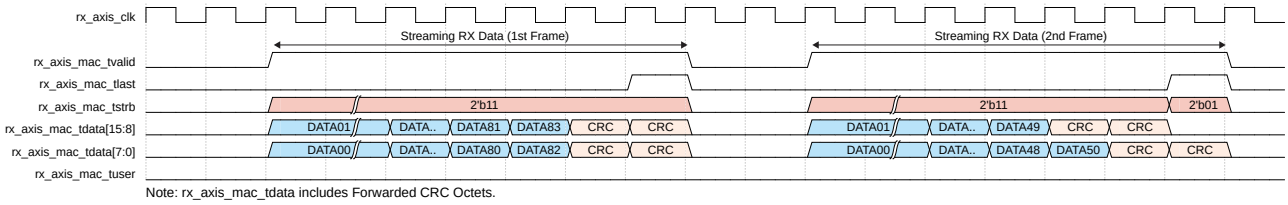


Figure 10: RX Data Frame and CRC Octets at AXI ST Interface when `crc_fwd = 1`



Note: rx_axis_mac_tdata includes Forwarded CRC Octets.

CRC Generation and Check

At the TX channel, the Triple Speed Ethernet MAC core automatically generates CRC octets, appends them to the end of frame, and transmits to the selected PHY interface.

At the RX channel, the Triple Speed Ethernet MAC core decodes the Ethernet frames and checks them against 4 CRC octets that trail the Ethernet frame. At the RX AXI ST interface, by default, the CRC octets are not forwarded to the `rx_axis_mac_tdata` port. To enable CRC forwarding to the `rx_axis_mac_tdata` port, configure the `crc_fwd` configuration bit to `1`.

The CRC polynomial that is used in Triple Speed Ethernet MAC core is $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$.

Programmable Inter Packet Gap (IPG)

The Triple Speed Ethernet MAC core supports a programmable IPG, configurable from 8 to 63 octets. The default IPG value can be set during IP generation in the IP Configuration window under **Programmable Inter Packet Gap**.

After the core is generated, the IPG can also be updated dynamically by writing to the `tx_ipg_length` register. Refer to **Control and Status Registers** on page 55 for register configuration guidelines.

Based on the configured IPG value, the MAC automatically inserts the minimum required gaps between transmitted frames. The implementation of IPG insertion is illustrated in **Figure 17: Conversion from TX AXI ST to SGMII TX** on page 24.

TX PHY Interface

At the PHY interface, Triple Speed Ethernet MAC core schedules transmit packets based on the following priority (highest to lowest):

1. Pause mechanism triggered by a received pause frame.
2. Request to transmit a pause frame (`xon_gen`/ `xoff_gen`)
3. User data from TX AXI4-stream interface (`tx_axis_mac_tvalid`, `tx_axis_mac_tlast`, `tx_axis_mac_tstrb`, `tx_axis_mac_tdata`, and `tx_axis_mac_tuser`)

User data transfer from the `tx_axis_mac` signals interface always has the lowest priority. The Triple Speed Ethernet MAC core maps `tx_axis_mac_tdata` directly onto the transmit data bus of the selected PHY interface. Therefore, `tx_axis_mac_tdata` must begin with the Ethernet frame header, i.e., Destination address, source address, length / type, and optional VLAN tags. In the figures of this user guide, the header octets are shown as DATA, to focus on the conversion between the TX AXI ST interface and the selected PHY interface.

When a higher-priority event (pause mechanism or pause frame request) is asserted during an ongoing transfer, the Triple Speed Ethernet MAC core waits until the current frame transmission is completed before servicing the higher-priority request.

You can get details on the transmission priority handling in **Priority Flow Control (Duplex Mode)** on page 40.

RGMII TX

For the RGMII interface, the **Double Data I/O (DDIO)** option must be enabled for the transmission of clock, data, and control signals. Table 6: Block Editor Settings for rgmii_txc, rgmii_txd, and rgmii_tx_ctl on page 21 shows the recommended Block Editor settings for `rgmii_txc`, `rgmii_txd`, and `rgmii_tx_ctl`.

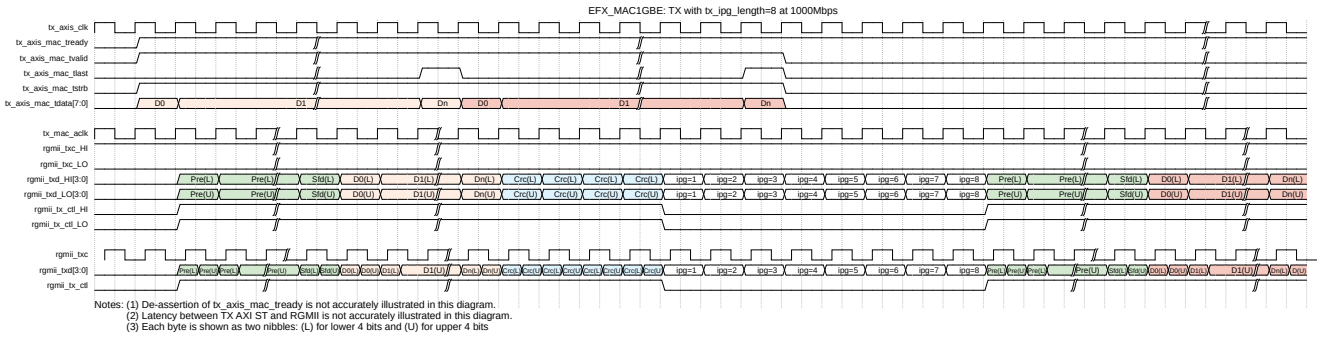
Table 6: Block Editor Settings for `rgmii_txc`, `rgmii_txd`, and `rgmii_tx_ctl`

Parameter	Setting		
	rgmii_txc	**rgmii_txd [3:0]**	**rgmii_tx_ctl**
Mode	output	output	output
Register Option	register	register	register
Double Data I/O Option	resync	resync	resync
Clock Pin Name	-	-	-
Pin Name (HI)	rgmii_txc_HI	rgmii_txd_HI	rgmii_tx_ctl_HI
Pin Name (LO)	rgmii_txc_LO	rgmii_txd_LO	rgmii_tx_ctl_LO
Enable invert clock	Yes (Trion) No (Titanium)	No	No
Output Clock Pin Name	clk_125m_90deg	clk_125m [(2)]	clk_125m

In the Efinity Interface Designer, use the **Create Block/ Create Bus** menu to add these signals and configure them according to Table 6: Block Editor Settings for rgmii_txc, rgmii_txd, and rgmii_tx_ctl on page 21.

Figure 11: Conversion from TX AXI ST to RGMII TX at 1000 Mbps on page 21 to Figure 13: Conversion from TX AXI ST to RGMII TX at 100 Mbps on page 22 illustrates the conversion from TX AXI ST to RGMII TX at different operating speeds with the recommended block editor settings in Table 6: Block Editor Settings for rgmii_txc, rgmii_txd, and rgmii_tx_ctl on page 21.

Figure 11: Conversion from TX AXI ST to RGMII TX at 1000 Mbps



[(2)] Select **clk_125m** if **RGMII Transmit Clock Delay** is **Disabled** (Edge-aligned).

Figure 12: Conversion from TX AXI ST to RGMII TX at 100 Mbps

with RGMII Transmit Clock Delay Enabled (Center-aligned)

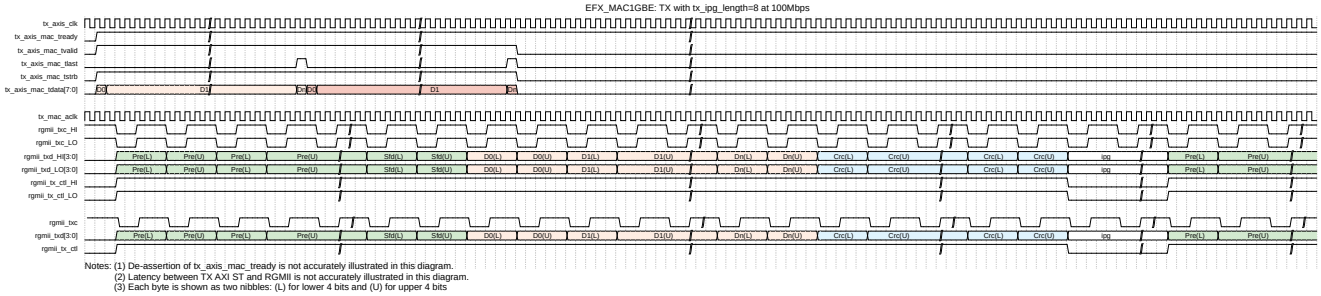
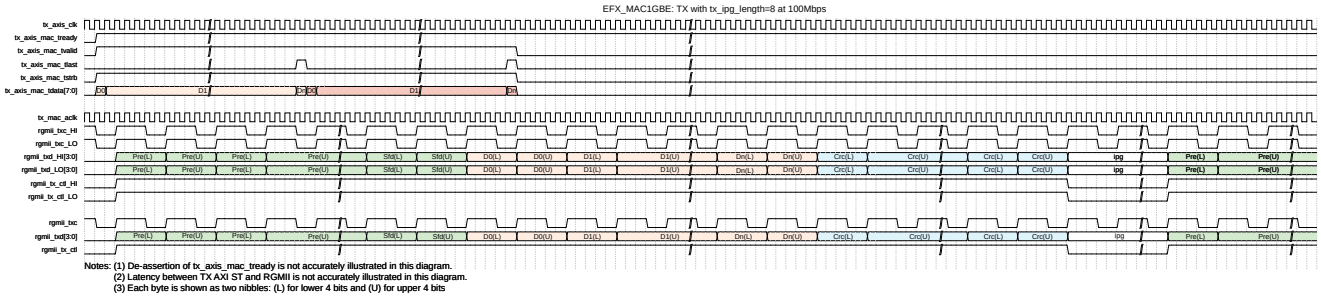


Figure 13: Conversion from TX AXI ST to RGMII TX at 100 Mbps

with RGMII Transmit Clock Delay Disabled (Edge-aligned)



The RGMII specification requires a clock-to-data skew to ensure correct sampling. This skew can be introduced either by PCB trace routing or by enabling the **RGMII Transmit Clock Delay** option in the GUI. This option only applies to 10 Mbps and 100 Mbps operation.

At 1000 Mbps, the required clock-to-data skew is achived internally by selecting a clock with a 90-degree phase shift (`clk_125m_90deg`) relative to the clock used to drive the data signals as the output clock for the `rgmii_txc` DDIO register. This configuration introduces an approximate 2 ns delay, aligning the `rgmii_txc` edge at the center of the data valid window for the transmitted RGMII data and control signals.

At 100 Mbps, the `rgmii_txc_HI` and `rgmii_txc_LO` signals are generated by dividing `tx_mac_aclk` by 5. The transmitted data on `rgmii_txd_HI` and `rgmii_txd_LO` are duplicated 5 times.

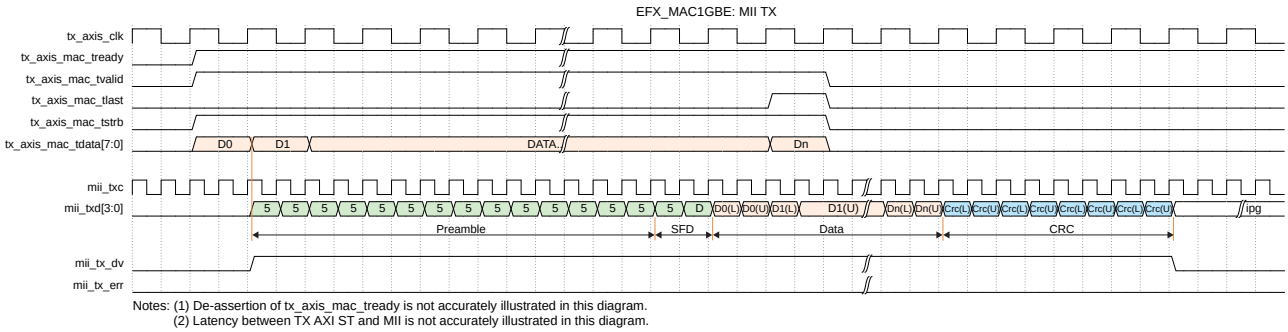
At 10 Mbps, the behavior is similar to 100 Mbps. The difference is that `rgmii_txc_HI` and `rgmii_txc_LO` are generated by dividing `tx_mac_aclk` by 50 instead of 5. The transmitted data on `rgmii_txd_HI` and `rgmii_txd_LO` are duplicated 50 times instead of 5 times.

By default, the **RGMII Transmit Clock Delay** option is enabled, resulting in center-aligned operation.

MII TX

Figure 14: Conversion from TX AXI ST to MII TX on page 23 illustrates the conversion from TX AXI ST to MII TX.

Figure 14: Conversion from TX AXI ST to MII TX

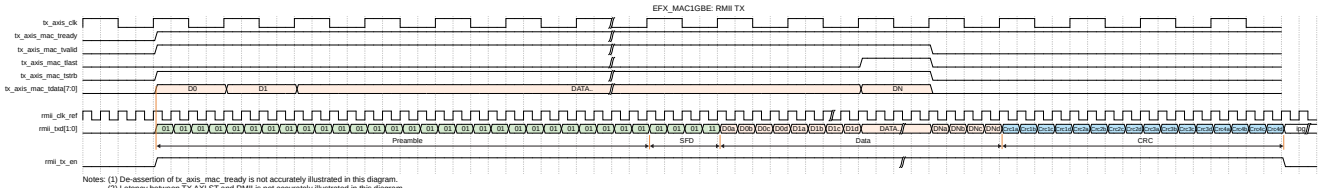


Notes: (1) De-assertion of tx_axis_mac_tready is not accurately illustrated in this diagram.
(2) Latency between TX AXI ST and MII is not accurately illustrated in this diagram.

RMII TX

Figure 15: Conversion from TX AXI ST to RMII TX on page 23 illustrates the conversion from TX AXI ST to RMII TX.

Figure 15: Conversion from TX AXI ST to RMII TX

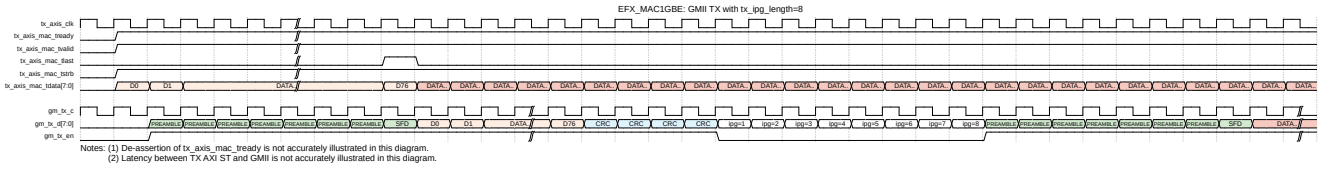


Notes: (1) De-assertion of tx_axis_mac_tready is not accurately illustrated in this diagram.
(2) Latency between TX AXI ST and RMII is not accurately illustrated in this diagram.
(3) Each byte is sent over 4 RMII cycles as di-bits a-d, where a = [1:0], b = [3:2], c = [5:4], d = [7:6]

GMII TX

Figure 16: Conversion from TX AXI ST to GMII TX on page 23 illustrates the conversion from TX AXI ST to GMII TX.

Figure 16: Conversion from TX AXI ST to GMII TX

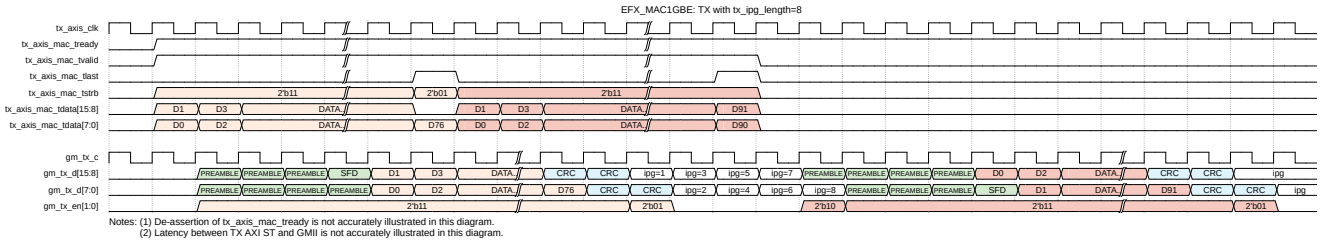


Notes: (1) De-assertion of tx_axis_mac_tready is not accurately illustrated in this diagram.
(2) Latency between TX AXI ST and GMII is not accurately illustrated in this diagram.

SGMII TX

Figure 17: Conversion from TX AXI ST to SGMII TX on page 24 illustrates the conversion from TX AXI ST to SGMII TX.

Figure 17: Conversion from TX AXI ST to SGMII TX



RX PHY Interface

At the RX path, the Triple Speed Ethernet MAC core decodes the selected PHY interface (RGMII, MII, RMII, GMII, or SGMII) and outputs the corresponding data stream on the RX AXI ST interface. If the received Ethernet frame begins at the upper octet, the Triple Speed Ethernet MAC core realigns the streaming data so that `rx_axis_mac_tstrb` starts with 2'b11.

Every Ethernet Frame received by the Triple Speed Ethernet MAC core is checked against the following criteria:

- *Frame Length*—Verification of minimum and maximum frame size.
- *FCS Check*—Validation of the frame check sequence.
- *Error Frame Detection*—Identification of corrupted or invalid frames.
- *Pause Frame Recognition*—Detection and processing of PAUSE control frames.
- *Address or Broadcast Filtering*—Filtering based on programmed Triple Speed Ethernet MAC core addresses and broadcast rules.

Note: Frame Length and FCS Check are performed only for non-error frames received from the PHY. If the PHY reports an error frame (e.g., via `gm_rx_err`), the Triple Speed Ethernet MAC core bypasses Frame Length and FCS checking for that frame. Consequently the `aRxFrameMismatchedLength` and `aRxFrameCheckSequenceErrors` statistic counters do not increment even if the frame exhibits length mismatch or FCS errors.

More details on the RX frame handling are provided in **Erroneous RX Frame** on page 35, **Address Filtering and Broadcast Filtering at RX** on page 37, and **Decoding Pause Frame at RX** on page 41.

RGMII RX

For the RGMII interface, the **Double Data I/O (DDIO)** option must be enabled for the receive data and control signals. Table 7: Block Editor Settings for rgmii_rxd, and rgmii_rx_ctl Targeting Trion Devices on page 26 and Table 8: Block Editor Settings for rgmii_rxd, and rgmii_rx_ctl Targeting Titanium/ Topaz Devices on page 26 show the recommended Block Editor settings for `rgmii_rxd` and `rgmii_rx_ctl`.

Table 7: Block Editor Settings for `rgmii_rxd`, and `rgmii_rx_ctl` Targeting Trion Devices

With First Sampling of RGMII Receive Data Option Set to Rising Edge

Parameter	Setting	
	rgmii_rxd [3:0]	**rgmii_rx_ctl**
Mode	Input	Input
Register Option	Register	Register
Double Data I/O Option	Resync	Resync
Clock Pin Name	rgmii_rxc	rgmii_rxc
Pin Name (HI)	rgmii_rxd_HI	rgmii_rx_ctl_HI
Pin Name (LO)	rgmii_rxd_LO	rgmii_rx_ctl_LO

Note: Due to the limited availability of GPIO pins with capability on the Trion T120 BGA324 development board, the example design with external PHY targeting this device cannot assign the RGMII control signals (`rgmii_tx_ctl` and `rgmii_rx_ctl`) to a standard DDIO block.

To accommodate for this specific hardware limitation, the following workaround is implemented in `tsemac_ex.v`:

```
assign rgmii_tx_ctl = rgmii_tx_ctl_HI | rgmii_tx_ctl_LO;
assign rgmii_rx_ctl_HI = rgmii_rx_ctl;
assign rgmii_rx_ctl_LO = rgmii_rx_ctl;
```

Table 8: Block Editor Settings for `rgmii_rxd`, and `rgmii_rx_ctl` Targeting Titanium/ Topaz Devices

With First Sampling of RGMII Receive Data Option Set to Falling Edge

Parameter	Setting	
	rgmii_rxd [3:0]	**rgmii_rx_ctl**
Mode	Input	Input
Register Option	Register	Register
Double Data I/O Option	Pipeline	Pipeline
Clock Pin Name	rgmii_rxc	rgmii_rxc
Pin Name (HI)	rgmii_rxd_LO	rgmii_rx_ctl_LO
Pin Name (LO)	rgmii_rxd_HI	rgmii_rx_ctl_HI

In the Efinity Interface Designer, use the **Create Block/ Create Bus** menu to add these signals and configure them according to Table 7: Block Editor Settings for rgmii_rxd, and rgmii_rx_ctl Targeting Trion Devices on page 26 and Table 8: Block Editor Settings for rgmii_rxd, and rgmii_rx_ctl Targeting Titanium/ Topaz Devices on page 26.

Figure 18: Conversion from RGMII RX to RX AXI ST at 1000 Mbps (Trion Block Editor Settings) on page 27 and Figure 19: Conversion from RGMII RX to RX AXI ST at 1000 Mbps (Titanium/ Topaz Block Editor Settings on page 27 illustrate the conversion from RGMII RX to RX AXI ST at 1000 Mbps with the recommended block editor settings in Table 7: Block Editor Settings for rgmii_rxd, and rgmii_rx_ctl Targeting Trion Devices on page 26 and Table 8: Block Editor Settings for rgmii_rxd, and rgmii_rx_ctl Targeting Titanium/ Topaz Devices on page 26.

Figure 18: Conversion from RGMII RX to RX AXI ST at 1000 Mbps (Trion Block Editor Settings)

Based on Table 7: Block Editor Settings for rgmii_rxd, and rgmii_rx_ctl Targeting Trion Devices on page 26

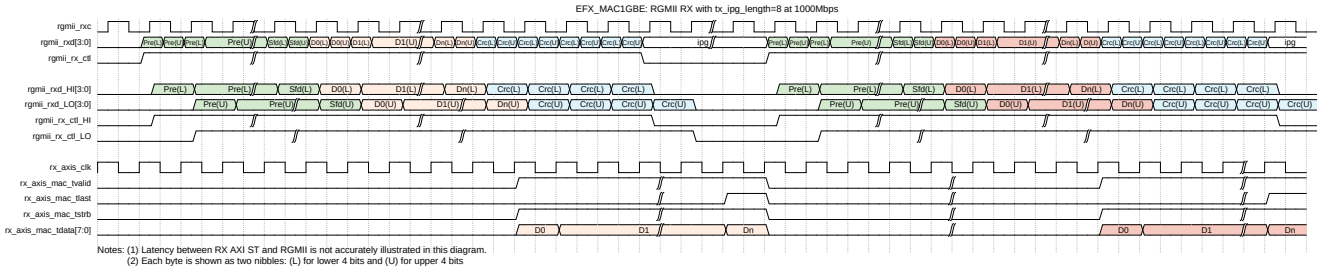
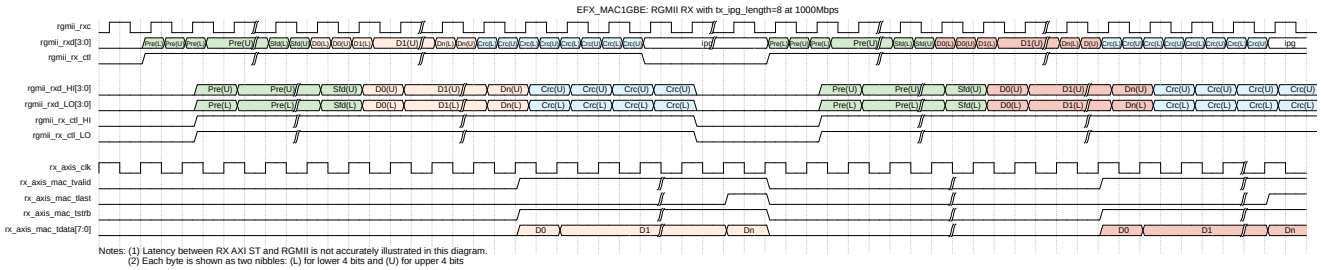


Figure 19: Conversion from RGMII RX to RX AXI ST at 1000 Mbps (Titanium/ Topaz Block Editor Settings

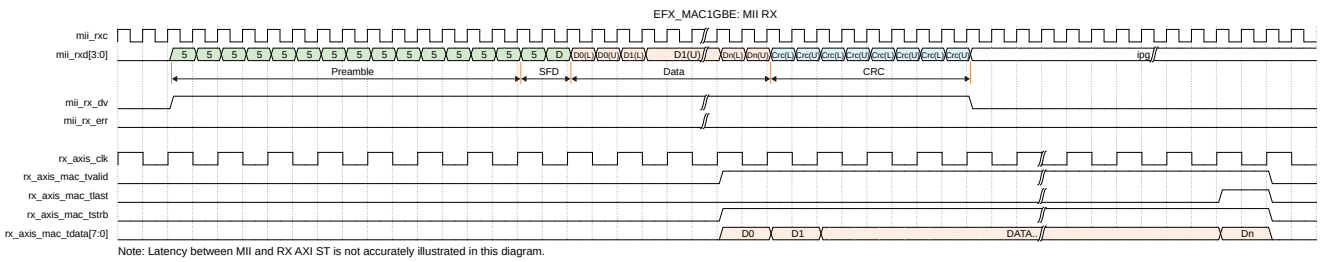
Based on Table 8: Block Editor Settings for rgmii_rxd, and rgmii_rx_ctl Targeting Titanium/ Topaz Devices on page 26



MII RX

Figure 20: Conversion from MII RX to RX AXI ST on page 27 illustrates the conversion from MII RX to RX AXI ST.

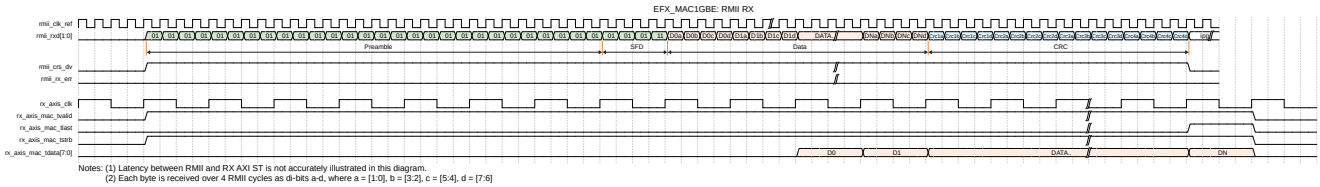
Figure 20: Conversion from MII RX to RX AXI ST



RMII RX

Figure 21: Conversion from RMII RX to RX AXI ST on page 28 illustrates the conversion from RMII RX to RX AXI ST.

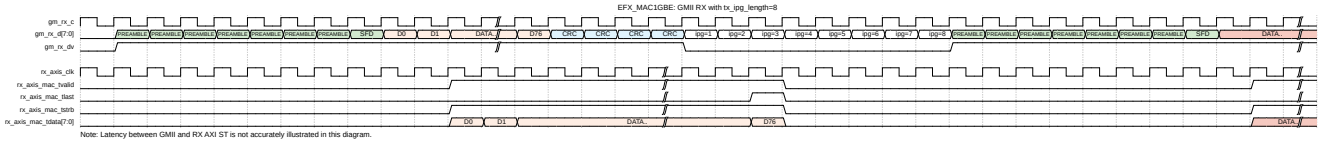
Figure 21: Conversion from RMII RX to RX AXI ST



GMII RX

Figure 22: Conversion from GMII RX to RX AXI ST on page 28 illustrates the conversion from GMII RX to RX AXI ST.

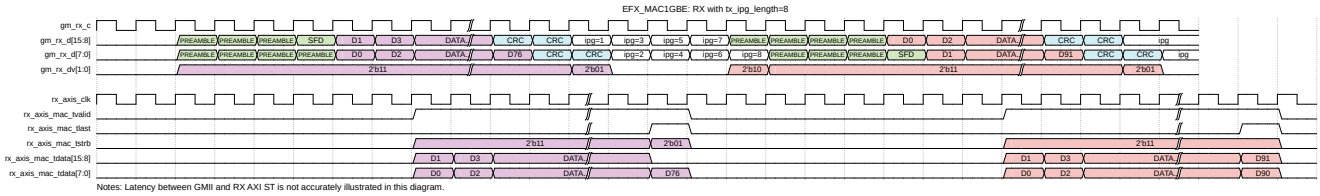
Figure 22: Conversion from GMII RX to RX AXI ST



SGMII RX

Figure 23: Conversion from SGMII RX to RX AXI ST on page 28 illustrates the conversion from SGMII RX to RX AXI ST.

Figure 23: Conversion from SGMII RX to RX AXI ST



Data Streaming Modes for Transmission

In Triple Speed Ethernet MAC core, 3 data streaming modes are available for the TX channel:

- **Cut Through** mode
- **Cut Through** with width-adaptation
- **Store Forward** mode

Cut Through Mode

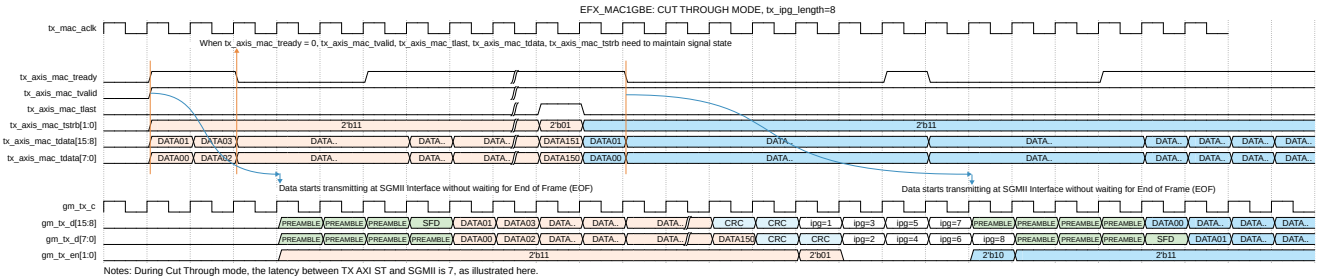
During **Cut Through** mode, the Triple Speed Ethernet MAC core starts transmitting an Ethernet frame before the entire packet has been received on the AXI4-stream interface. This reduces latency compared to **Store Forward** mode.

Cut-Through Mode is available when the AXI data width matches the PHY data width and the transmit FIFO is disabled:

- AXI4 Data Width = 8, PHY Interface = 8-bit GMII
- AXI4 Data Width = 16, PHY Interface = 16-bit GMII (SGMII)

If an error occurs during transmission in this mode, the MAC terminates the frame as an error frame. See Terminating Bad TX Frames on page 32 for details.

Figure 24: Cut Through Mode



Cut Through Mode with Width Adaptation

This mode behaves like **Cut Through** mode, but is used when the AXI data width does not match the PHY data width and the transmit FIFO is disabled. In this case, a shallow FIFO is inserted to perform width adaptation between the AXI4-stream interface and the PHY interface.

- Transmission still starts before the end of the frame is received, minimizing latency
- The shallow FIFO is only used for width conversion and does not buffer the entire frame.

Error handling follows the same rules as in **Cut Through** mode.

Store Forward Mode

In **Store Forward** mode, the MAC buffers the entire frame in the transmit FIFO before initiating transmission on the PHY interface. The TXFIFO must be large enough to accommodate the complete frame, including Ethernet headers and optional VLAN tags. Transmission begins only after the entire frame has been received (`tx_axis_mac_tlast` asserted), subject to the transmit priority rules described in TX PHY Interface on page 20.

The minimum TXFIFO depth required is calculated as follows:

- Without VLAN tag:

$$\text{TXFIFO Depth} \geq \frac{14\ \textit{Header Bytes} + \textit{Maximum Payload Size}}{\left(\frac{\text{AXI Data Width}}{8}\right)}$$

- With VLAN tag:

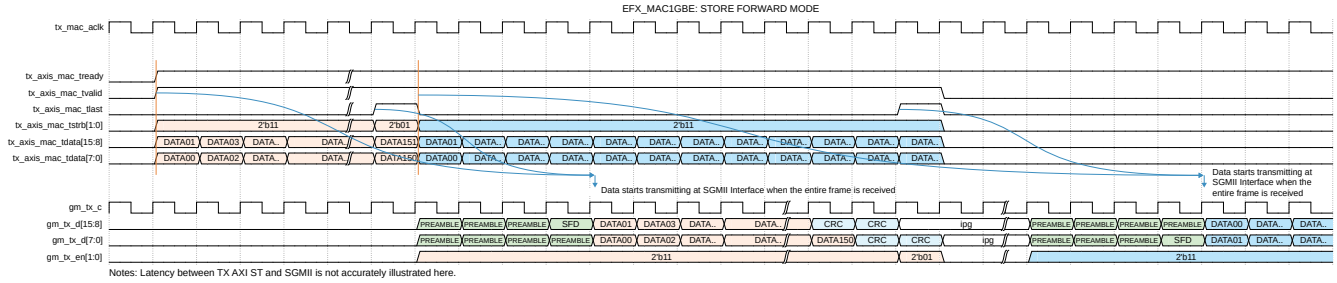
$$\text{TXFIFO Depth} \geq \frac{18\ \textit{Header Bytes} + \textit{Maximum Payload Size}}{\left(\frac{\text{AXI Data Width}}{8}\right)}$$

If the TXFIFO overflows before the entire frame is stored, the incomplete frame is discarded and is not transmitted.

During operation, when `tx_axis_mac_tvalid = 1`, the MAC stores the corresponding `tx_axis_mac_tdata`, `tx_axis_mac_tstrb`, and `tx_axis_mac_tlast`. When `tx_axis_mac_tvalid = 0`, no data is stored for that cycle. The frame is not terminated, and transmission still proceeds once the complete frame is received.

However, if `tx_axis_mac_tstrb` deasserts before `tx_axis_mac_tlast` (i.e., dropped bytes within the middle of a frame), the MAC treats the frame as invalid. In this case, the frame is flagged as an error frame and terminated. See Terminating Bad TX Frames on page 32 for more details.

Figure 25: Store Forward Mode



Data Streaming Modes for Reception

The Triple Speed Ethernet MAC core supports 2 data streaming modes on the RX channel, depending on the configuration of the receive FIFO and AXI interface width:

- **Cut Through** mode
- **FIFO** mode

In both modes, the Triple Speed Ethernet MAC core begins forwarding received data to the RX AXI4-Stream interface as soon as the frame starts arriving from the PHY interface.

The key difference lies in latency and buffering behavior. The **Cut Through** mode provides minimum latency, while **FIFO** mode introduces additional delay due to buffering and potential clock domain crossing.

Cut Through Mode

During **Cut Through** mode, the Triple Speed Ethernet MAC core streams data directly from the PHY interface to the RX AXI4-Stream interface without intermediate buffering.

This mode offers the lowest latency and is available when the AXI data width matches the PHY data width and the RX FIFO is disabled:

- AXI4 Data Width = 8, PHY Interface = 8-bit GMII
- AXI4 Data Width = 16, PHY Interface = 16-bit GMII (SGMII)

FIFO Mode

During **FIFO** mode, the Triple Speed Ethernet MAC core begins forwarding data as the frame is received but includes an internal FIFO to provide elasticity and safe clock domain crossing between the PHY and AXI clock domains.

This mode is required when the two clocks are asynchronous or operate at different frequencies.

Because data is written into and read out from FIFO, this mode introduces slightly higher latency compared to **Cut Through** mode.

The minimum RX FIFO depth depends on the PHY interface types and maximum frame size:

- For SGMII Interface:
  - Without VLAN tag: RXFIFO Depth ≥ (14 Header Bytes + Maximum Payload Size) / 2
  - With VLAN tag: RXFIFO Depth ≥ (18 Header Bytes + Maximum Payload Size) / 2
- For other interfaces (RGMII, MII, RMII, GMII)
  - Without VLAN tag: RXFIFO Depth ≥ (14 Header Bytes + Maximum Payload Size)
  - With VLAN tag: RXFIFO Depth ≥ (18 Header Bytes + Maximum Payload Size)

Automatic Padding for Short TX Frames

To conform to the Ethernet format, the minimum payload size is 46 for non-VLAN tagged and 42 for VLAN tagged frames. In the event of short TX packets, the Triple Speed Ethernet MAC core automatically pads with zeros to fulfill the minimum payload size.

Terminating Bad TX Frames

The following conditions are the causes of a frame being terminated as a bad frame at TX channels:

- Non-Compliant `tx_axis_mac_tlast`
- Assertion of `tx_axis_mac_tuser`
- Non-streaming TX data
- Dropped `tx_axis_mac_tvalid` (**Cut Through** mode only)
- TXFIFO overflow (**Store Forward** mode only)

Non-Compliant `tx_axis_mac_tlast`

An Ethernet frame should consist minimally of the Ethernet header, which includes the following:

- Destination address
- Source address
- Optional VLAN tag
- Ethernet type/length

If `tx_axis_mac_tlast` is asserted within the Ethernet header, it is known as a non-compliant `tx_axis_mac_tlast`.

When `tx_axis_mac_tlast` is asserted at the first cycle for the following configurations:

- AXI Data Width = 8 (all supported PHY interfaces)
- AXI Data Width = 16 (SGMII mode)

The frame is dropped silently, and there is no indicator of the dropped frame. No statistic counter is incremented in this case.

However, if `tx_axis_mac_tlast` is asserted from the 2nd cycle onwards but still within the header, the header is considered incomplete. In this condition, the Triple Speed Ethernet MACcore terminates the frame and asserts the corresponding PHY error signal (e.g., `gm_tx_err` for SGMII or GMII interface).

Assertion of tx_axis_mac_tuser

The assertion of tx_axis_mac_tuser indicates that there is an error in the data streaming transmission. Upon the assertion of tx_axis_mac_tuser and tx_axis_mac_tvalid, the Triple Speed Ethernet MAC core terminates the frame with an error indication and discards the remaining portion of the frame.

When tx_axis_mac_tuser is asserted at the first cycle, the frame is dropped silently, and there is no indicator of the dropped frame. No statistic counter is incremented in this case.

However, if tx_axis_mac_tuser is asserted while tx_axis_mac_tlast remains low from the 2nd cycle onwards but still within the Ethernet header, the header is considered incomplete. In this condition, the Triple Speed Ethernet MAC core terminates the frame, discards the remaining portion, and asserts the corresponding PHY error signal (e.g., gm_tx_err for SGMII or GMII interface).

When tx_axis_mac_tvalid is low, all tx_axis_mac signals (tdata, tstrb, tlast, tuser) interface are considered invalid, as described in **TX AXI ST Interface** on page 16. Therefore, an assertion of tx_axis_mac_tuser during a cycle where tx_axis_mac_tvalid is deasserted does not indicate an error condition.

However, in **Cut Through** mode, if tx_axis_mac_tvalid is deasserted prematurely (before the end of frame), the Triple Speed Ethernet MAC core terminates the transmission as a bad frame, as described in **Dropped tx_axis_mac_tvalid Before End of Frame** on page 34.

Non-Streaming TX Data

For Ethernet applications, tx_axis_mac_tdata must be streaming and continuous throughout a frame. This requires that tx_axis_mac_tstrb be fully asserted (all 1's) on every cycle, except the final cycle of the frame indicated by tx_axis_mac_tlast.

- 8-bit AXI data width;
  - tx_axis_mac_tstrb is always 1'b1 until the tx_axis_mac_tlast cycle.
- 16-bit AXI data width;
  - tx_axis_mac_tstrb is 2'b11 on all cycles except the tx_axis_mac_tlast cycle, where it may be 2'b01 or 2'b11 depending on whether the frame ends on an odd or even byte boundary.
- 32-bit AXI data width;
  - tx_axis_mac_tstrb must be 4'b1111 on all cycles except the tx_axis_mac_tlast cycle. On the final cycle, it must indicate the valid bytes as a contiguous sequence starting form the Least Significant Bytes (i.e., 4'b0001, 4'b0011, 4'b0111). Sparse byte enables are not supported.

If non-streaming TX data is detected (for example, if tx_axis_mac_tstrb is not fully asserted during a non-tlast cycle), the Triple Speed Ethernet MAC core terminates the transmission with an error frame and discards the remaining data. This event increments the aTxIncontinuityFramesErrors counter.

Dropped `tx_axis_mac_tvalid` Before End of Frame

Streaming data is defined as continuous valid data bytes from the start of frame (SOF) until the end of frame (EOF). This requires that `tx_axis_mac_tvalid` remains asserted for every cycle between SOF to EOF.

Cut Through Mode

In **Cut Through** mode, if `tx_axis_mac_tvalid` is deasserted before EOF, the continuity of streaming data is broken. When this occurs, the Triple Speed Ethernet MAC core terminates the transmission with an error frame and drops the remaining frame. This event increments the `aTxIncontinuityFramesErrors` counter.

Cut-Through Mode with Width-Adaptation or Store-Forward Mode

In **Cut Through mode with width-adaptation** or **Store Forward** mode, a dropped `tx_axis_mac_tvalid` simply indicates that the corresponding column of `tx_axis_mac` signals are not written into TXFIFO.

Therefore, in **Cut Through mode with width-adaptation** or **Store Forward** mode, a dropped `tx_axis_mac_tvalid` does not result in an error frame.

TX FIFO Overflow

In **Store Forward** mode, the TX FIFO may overflow under the following conditions:

- *FIFO depth insufficient for one full frame*—When the write and read sides of the TX FIFO operate at equal bandwidth, the FIFO must be sized large enough to store an entire maximum-sized frame, as described in **Store Forward Mode** on page 30.
- *Write Bandwidth Exceeds Read Bandwidth*—If the FIFO is sized to accommodate a full frame, overflow can still occur when the write side (TX AXI-ST interface) provides data at a rate higher than the read side (Triple Speed Ethernet MAC core transmit engine) can drain it. In this condition, the FIFO may fill before the Triple Speed Ethernet MAC core completes transmitting the earlier data.

For both scenarios, once a TX FIFO overflow is detected, the Triple Speed Ethernet MAC core silently drops the entire frame without transmitting it. Each overflow event increments the `aTxFifoOverflowFramesErrors` counter.

Erroneous RX Frame

When an error condition is detected, the Triple Speed Ethernet MAC core validates the frame against the following precedence rules. Only the highest-priority error counter is incremented.

For any error listed below, the Triple Speed Ethernet MAC core performs the following actions:

- Asserts `rx_axis_mac_tuser` to flag the frame as invalid.
- Increments the `ifInErrors` statistic counter.

Table 9: RX Statistic Counter Precedence

Priority	Condition	Statistic Counter Incremented
1 (Highest)	Undersized (Received frame is < 64 bytes)	`etherStatsUndersizePkts`
2	Oversized (Received frame exceeds the configured `frm_length`)	`etherStatsOversizePkts`
3	FCS Error (Received frame has valid size but fails CRC)	`aFrameCheckSequenceErrors`
4 (Lowest)	Length Mismatch (Received frame has valid size, but Length/Type field does not match actual payload)	`aRxFrameMismatchedLength`

Undersized RX Frame

An Ethernet frame must carry a minimum payload size of 46 bytes for non-VLAN tagged frames and 42 bytes for VLAN tagged frames. If a received frame does not meet this requirement, it is flagged as an undersized and handled according to the precedence rules in Table 9: RX Statistic Counter Precedence on page 35.

Oversized RX Frame

Each received frame is checked against the configured `frm_length` value. If the physical frame length exceeds this configured limit, the frame is flagged as an oversized and handled according to the precedence rules in Table 9: RX Statistic Counter Precedence on page 35.

Frame Check Sequence (FCS) Error

Each received frame undergoes a cyclic redundancy check (CRC) validation. If the calculated CRC does not match the received FCS field, the frame is flagged as an FCS Error. This event increments the statistic counter `aFrameCheckSequenceErrors`, provided that the frame is not undersized or oversized.

Mismatched Length RX Frame

For Ethernet frame with a payload size ≤1500 bytes, the Triple Speed Ethernet MAC core compares the physical frame length against the value in the Ethernet Type/ Length field. If a difference occurs, the frame is flagged as a mismatched length frame.

Note: As per the precedence rules in Table 9: RX Statistic Counter Precedence on page 35, the `aRxFrameMismatchedLength` counter is incremented only if the frame has a valid size and a valid FCS.

Non-Streaming RX Data Frame (SGMII)

A *Non-Streaming RX Data Frame* is a condition specific to the SGMII interface. It occurs when any of the 2-bit `gm_rx_dv` signal is deasserted before the end-of-frame.

Preamble or SFD Exception—If a non-streaming condition occurs during a preamble or a start frame delimiter (SFD), the sequence is silently discarded. No statistic counters are incremented.

Frame Termination—If a non-streaming condition occurs after an SFD is detected:

1. The Triple Speed Ethernet MAC core immediately terminates the frame.
2. The frame received up to the point of termination is validated according to the precedence rules in Table 9: RX Statistic Counter Precedence on page 35.
3. Depending on the length of data received before the break, the frame is counted as undersized, oversized or an FCS error.

Error Frame

An *Error Frame* occurs when the PHY asserts the error signal during reception, which are `gm_rx_err` (GMII or SGMII), `rmii_rx_err` (RMII), `mii_rx_err` (MII) , or RGMII decoded error.

Preamble or SFD Exception—If an error signal is asserted during a preamble or a start frame delimiter (SFD), the sequence is silently discarded. No statistic counters are incremented.

Frame Termination—If an error signal is asserted after an SFD is detected:

1. The Triple Speed Ethernet MAC core immediately terminates the frame.
2. The frame received up to the point of termination is validated according to the precedence rules in Table 9: RX Statistic Counter Precedence on page 35.
3. The last 4 octets of a data received before termination are interpreted as a frame check sequence (FCS). As these octets do not represent the valid checksum for the truncated frame, the FCS validation fails. If the frame length during termination is valid (neither undersized nor oversized), this resulting FCS failure will increment the `aRxFrameCheckSequenceErrors` counter.

Address Filtering and Broadcast Filtering at RX

The Triple Speed Ethernet MAC core supports unicast, multicast, and broadcast filtering on incoming RX Ethernet frames.

- **Unicast/ Multicast Filtering**
  - Configure the `mac_addr_mask` register to apply bit-wise filtering.
    - Bit = 1: Corresponding address bit is compared
    - Bit = 0: Corresponding address bit is ignored.

See Table 10: Examples of Unicast, Multicast, and Broadcast Filtering Operation on page 38 for examples.

Note: Address filtering can be completely disabled by setting the `promis_en` bit to `1`.

- **Broadcast Filtering**
  - When `broadcast_filter_en` is set, all RX frames with `Destination Address = 48'hFFFF_FFFF_FFFF` are dropped.

The default value of this control register can be configured in the IP Configuration window under broadcast filtering.

Table 10: Examples of Unicast, Multicast, and Broadcast Filtering Operation

broadcast_filter_en	mac_address_mask	mac_addr	Destination Address[3]	Filtered and Dropped / Output at RX AXI ST	Application
Unicast					
Don't Care	48'hFFFF_FFFF_FFFF	48'hAE3A_2020_0080	48'hAE3A_2020_0080	Output at RX AXI ST	Unicast
Don't Care	48'hFFFF_FFFF_FFFF	48'hAE3A_2020_0080	48'h3E3F_7A20_0080	Filtered & Dropped	Unicast
Multicast					
Don't Care	48'hFFFF_FFFF_FFFF	48'h0100_5E0A_0B0C	48'h0100_5E0A_0B0D	Filtered & Dropped	Multicast
Don't Care	48'hFFFF_FFFF_FFFF	48'h0100_5E0A_0B0C	48'hAE3A_2020_0080	Filtered & Dropped	Multicast
Don't Care	48'hFFFF_FFFF_FFFF	48'h0100_5E0A_0B0C	48'h0100_5E0A_0B0C	Output at RX AXI ST	Multicast
Don't Care	48'hFFFF_FF00_0000	48'h3333_ABCD_0000	48'h3333_ABCD_1234	Output at RX AXI ST	Multicast
Broadcast					
1	Don't Care	Don't Care	48'hFFFF_FFFF_FFFF	Filtered & Dropped	Broadcast
0	Don't Care	Don't Care	48'hFFFF_FFFF_FFFF	Output at RX AXI ST	-
Address Filtering Is Not Enabled					
0	48'h0000_0000_0000	48'hAE3A_2020_0080	48'hAE3A_2020_0080	Output at RX AXI ST	-

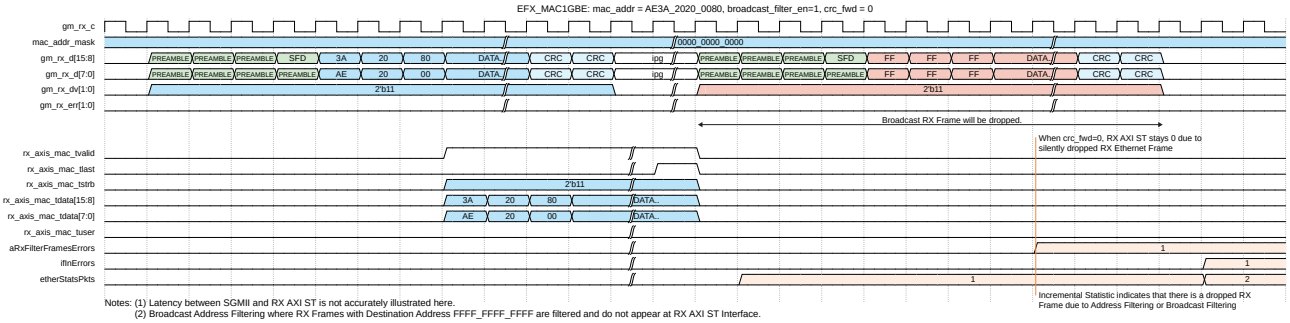
When address filtering or broadcast filtering is enabled, any received frame with a mismatched destination address is subject to being filtered. The exact handling of these filtered frames depends on the `crc_fwd` control bit.

- When `crc_fwd = 0`: Filtered RX frames are silently dropped and will not appear on the RX AXI Stream interface, as shown in Figure 26: Silent Drop of Filtered RX Frame when crc_fwd = 0 on page 39.
- When `crc_fwd = 1`: Filtered RX frames are forwarded but marked as invalid. The frame data is still presented on the RX AXI Stream interface, but it is terminated with the `rx_axis_mac_tuser` signal asserted, as shown in Figure 27: Filtered RX Frame when crc_fwd = 1 on page 39.

Regardless of the `crc_fwd` settings, you can monitor the `aRxFilterFramesErrors` statistic counter to track the total number of frames that have been filtered due to address mismatches.

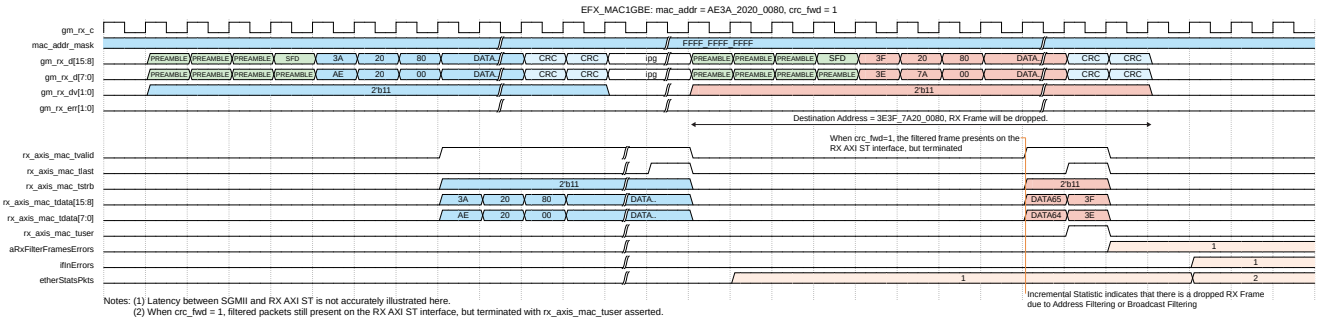
[3] Destination address of the incoming RX Ethernet frame.

Figure 26: Silent Drop of Filtered RX Frame when crc_fwd = 0



Notes: (1) Latency between SGMII and RX AXI ST is not accurately illustrated here.
(2) Broadcast Address Filtering where RX Frames with Destination Address FFFF_FFFF_FFFF are filtered and do not appear at RX AXI ST Interface.

Figure 27: Filtered RX Frame when crc_fwd = 1



Notes: (1) Latency between SGMII and RX AXI ST is not accurately illustrated here.
(2) When crc_fwd = 1, filtered packets still present on the RX AXI ST interface, but terminated with rx_axis_mac_tuser asserted.

Priority Flow Control (Duplex Mode)

The Triple Speed Ethernet MAC core supports the flow control at both the TX and RX paths.

Send Pause Frame at TX

To generate a PAUSE frame for flow control, the Triple Speed Ethernet MAC core provides two configuration bits of `xoff_gen` and `xon_gen`.

- *Initiate Pause (XOFF)*—Program the desired pause duration into the `pause_quant` register, then assert the `xoff_gen` bit.
- *Resume Transmission (XON)*—Assert the `xon_gen` bit. This triggers a PAUSE frame with a quanta value of zero.

Both the `xon_gen` and `xoff_gen` are latched signals. After the PAUSE frame has been sent, the user must manually clear the bit by writing a `0` to prepare for subsequent operations.

Priority and Override Behavior

A PAUSE frame requests take priority over data frame transmission. However, if a pause request is triggered during an ongoing data frame, the PAUSE frame is queued and transmitted immediately after the current frame completes.

The Triple Speed Ethernet MAC core supports dynamic updates to the pause request. If `xon_gen` or `xoff_gen` is asserted before the previous request's pause quanta has been transmitted, the Triple Speed Ethernet MAC core cancels the ongoing request and transmits only the most recent request. For example, if an `xon_gen` request is made when there is a pending `xoff_gen` request (and before the XOFF frame's pause quanta value is sent), the Triple Speed Ethernet MAC core discards the XOFF request, and only the XON frame will be transmitted. The mechanism ensures efficient use of bandwidth link by preventing the transmission of redundant PAUSE frames.

Figure 28: Triggering pause frame generation using `xoff_gen` and the corresponding Statistic Reporting increments

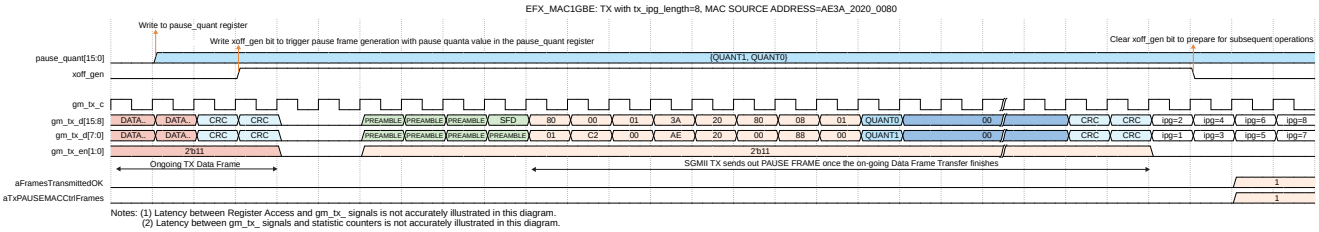
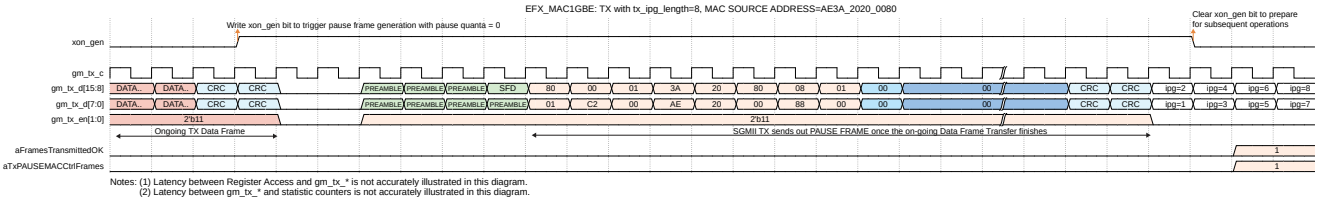


Figure 29: Triggering pause frame generation using `xon_gen` and the corresponding Statistic Reporting increments



Decoding Pause Frame at RX

On the RX path, the Triple Speed Ethernet MAC core monitors incoming Ethernet frames to detect PAUSE frames, which are uniquely identified by the reserved destination address 01-80-C2-00-00-01. When a PAUSE frame is detected, the Triple Speed Ethernet MAC core extracts the `RX_QUANT` value and forwards it to the TX logic. This value directly controls how long the transmitter pauses the data transmission. One quanta corresponds to the time required to transmit 512 bits at the current link speed.

The RX PAUSE frames are not subject to address filtering. A received frame is validated as a PAUSE frame if it meets all of the following criteria:

- Matches the reserved destination address 01-80-C2-00-00-01
- Contains the correct EtherType 0x8808 and PAUSE opcode 0x0001
- Passes the CRC check

If a CRC error is detected, the RX PAUSE frame is deemed invalid, the extracted `RX_QUANT` is discarded, and no PAUSE operation is triggered. The received PAUSE frame is always forwarded to the RX AXI stream interface like regular frame.

A valid `RX_QUANT` from a received PAUSE frame has the highest priority over other transmitter operations, including locally generated PAUSE requests (`xon_gen` or `xoff_gen`) and TX AXI stream data. If a frame transmission is currently in progress, the new PAUSE command is queued and takes effect immediately after the current frame completes.

You can disable the RX PAUSE frame mechanism by setting the `pause_ignore` configuration bit to 1. When set, the Triple Speed Ethernet MAC core ignores the decoded `RX_QUANT` value, and no PAUSE operation occurs.

Note: The Triple Speed Ethernet MAC core updates the PAUSE timer whenever a new PAUSE frame is received. If another PAUSE frame arrives while a previous PAUSE is active, the new `RX_QUANT` value replaces the remaining timer value – it does not accumulate or extend it. A PAUSE frame with `RX_QUANT` = `0` clears the timer immediately and resumes transmission.

Statistic Reporting

The Triple Speed Ethernet MAC core provides statistic reporting for both TX and RX operations. Each counter is 16 bits wide. When a counter reaches its maximum value (16'hFFFF), it saturates and does not roll over. To clear all counters, the user must set the `cnt_reset` bit, which resets them to `0`. All statistical values shown in the statistic-related diagrams are reported in hexadecimal format.

The following registers are the statistic reporting for TX:

- `aFramesTransmittedOK`
- `aTxPAUSEMACCtrlFrames`
- `ifOutErrors`
- `aTxFifoOverflowFramesErrors`
- `aTxIncontinuityFramesErrors`

The following registers are the statistic reporting for RX:

- `aFramesReceivedLen`
- `etherStatsUndersizePkts`
- `etherStatsOversizePkts`
- `aRxFrameMismatchedLength`
- `aFrameCheckSequenceErrors`
- `aRxFilterFramesErrors`
- `ifInErrors`
- `aRxPAUSEMACCtrlFrames`
- `aFramesReceivedOK`
- `etherStatsPkts`

aFramesTransmittedOK

This counter reports the number of frames successfully transmitted on the selected PHY interface. It includes streaming TX data frames with `tx_axis_mac_tuser = 0`, short frames with auto-padding, and TX PAUSE frames.

aTxPAUSEMACCtrlFrames

This counter reports the number of PAUSE frames transmitted due to `xon_gen` or `xoff_gen` requests. The behavior of TX PAUSE frames is described in **Send Pause Frame at TX** on page 40.

ifOutErrors

This counter reports the number of bad TX frames terminated due to errors during transmission. The termination criteria include:

- Assertion of `tx_axis_mac_tuser`.
- Non-streaming TX data.
- Dropped `tx_axis_mac_tvalid` before end of frame (**Cut-Through** mode only).
- Frames discarded due to TX FIFO overflow in **Store-Forward** mode.

This counter does not increment for frames that are silently dropped due to non-compliant `tx_axis_mac_tlast`, or when `tx_axis_mac_tuser` is asserted during the first cycle of the frame. These conditions result in the frame being discarded without an error indication or a statistic update.

aTxFifoOverflowFramesErrors

This counter reports the number of TX frames discarded due to TX FIFO overflow in **Store Forward** mode. The counter increments whenever an overflow occurs. In **Cut Through** mode or **Cut Through** mode with width adaptation, this counter remains at `0`.

aTxIncontinuityFramesErrors

This counter reports the number of transmitted frames that exhibit non-streaming TX data behavior (invalid `tx_axis_mac_tstrb` patterns) and dropped `tx_axis_mac_tvalid` before end-of-frame in **Cut Through** mode.

aFramesReceivedLen

This counter reports the length of each RX frame received at the RX AXI ST interface.

The frame length includes:

- Destination Address (6 octets)
- Source Address (6 octets)
- Optional VLAN tag (4 octets)
- Ethertype/ Length (2 octets)
- Payload
- Frame Check Sequence (4 octets)

This counter reflects the length of the received frame only if the frame is error-free as illustrated in Figure 33: RX Statistics - aRxPAUSEMACCtrlFrames on page 45. For a list of erroneous RX frames, refer to Erroneous RX Frame on page 35.

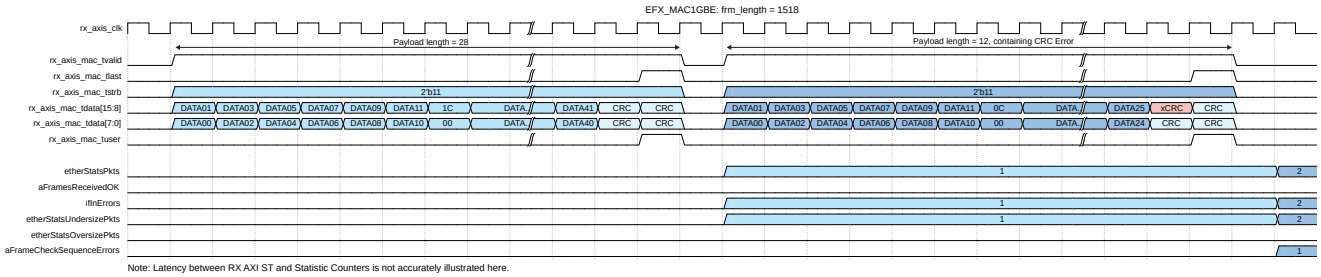
etherStatsUndersizePkts

This counter reports the number of undersized RX frames:

- Payload < 46 bytes for non-VLAN tagged frames
- Payload < 42 bytes for VLAN tagged frames

For these frames, `rx_axis_mac_tuser` is asserted. See Erroneous RX Frame on page 35.

Figure 30: RX Statistic - etherStatsUndersizePkts

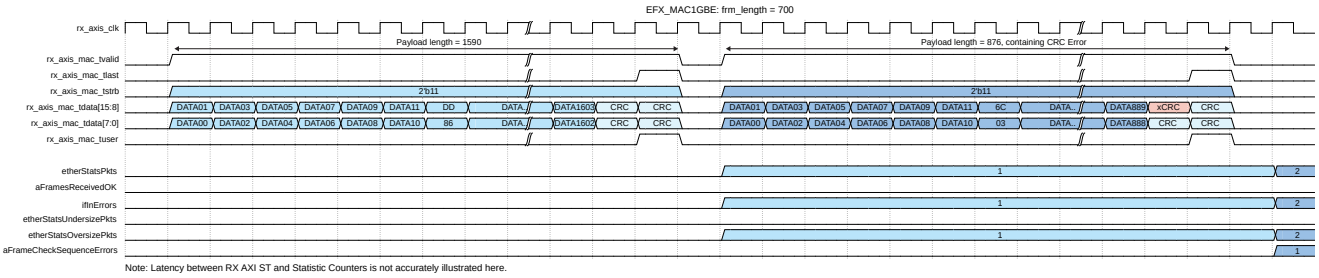


etherStatsOversizePkts

This counter reports the number of oversized RX frames that have a frame length greater than the configured `frm_length` value.

For these frames, `rx_axis_mac_tuser` is asserted. See Erroneous RX Frame on page 35.

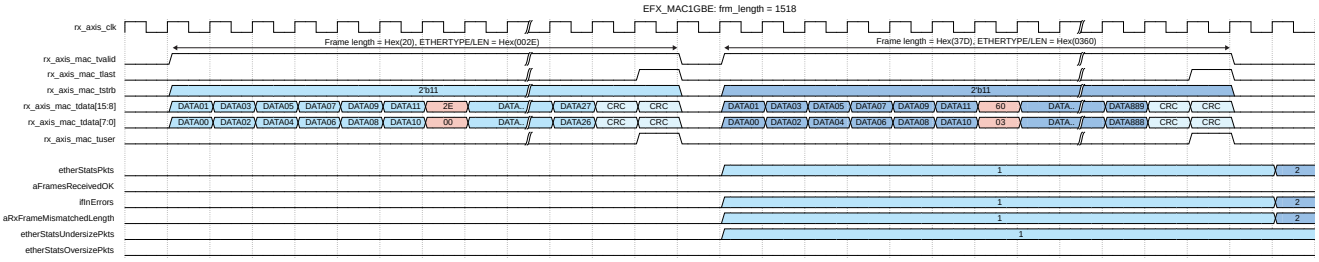
Figure 31: RX Statistic - etherStatsOversizePkts



aRxFrameMismatchedLength

This counter reports the number of received frames where the actual physical frame length does not match the value in the Ethertype/ Length field. It increments only for valid, non-error frames. For such frames, `rx_axis_mac_tuser` is asserted. **Figure 32: RX Statistic - Mismatched Length RX Frame** on page 44 illustrates the `aRxFrameMismatchedLength` statistic updates relative to the RX frames at AXI ST interface.

Figure 32: RX Statistic - Mismatched Length RX Frame



aFrameCheckSequenceErrors

This counter reports the number of received frames that fail the CRC check. For each frame with a CRC error, the Triple Speed Ethernet MAC core asserts `rx_axis_mac_tuser` to indicate an error condition on the RX AXI ST interface.

There is no increment for:

- Frames flagged as error frames by the PHY.
- Frames filtered due to address filtering or broadcast filtering.

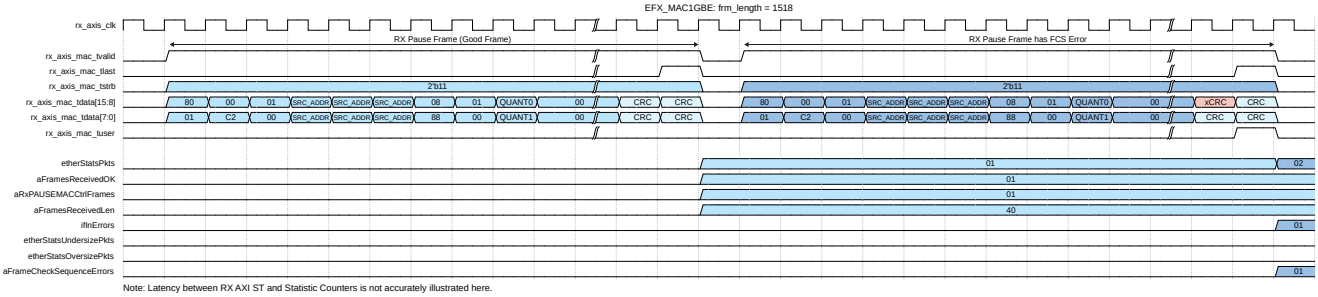
aRxFilterFramesErrors

This counter reports the number of RX frames dropped due to broadcast filtering or address filtering. See Erroneous RX Frame on page 35 and Address Filtering and Broadcast Filtering at RX on page 37 for details on broadcast and address filtering.

aRxPAUSEMACCtrlFrames

This counter reports the number of RX frames decoded as PAUSE frames.

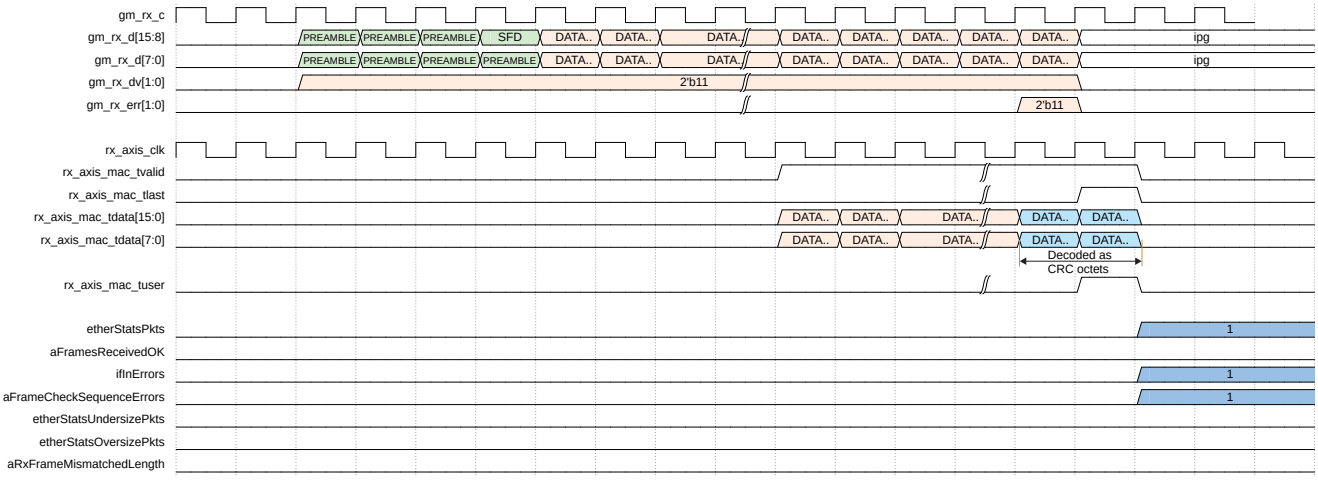
Figure 33: RX Statistics - aRxPAUSEMACCtrlFrames



ifInErrors

This counter reports the total number of erroneous RX frames, see **Erroneous RX Frame** on page 35, as well as frames dropped due to broadcast or address filtering, see **Address Filtering and Broadcast Filtering at RX** on page 37. For all erroneous frames, `rx_axis_mac_tuser` is asserted, except in the case of frames dropped due to broadcast or address filtering.

Figure 34: RX Statistic Reporting - Error Frame



aFramesReceivedOK

This counter reports the total number of error-free RX frames received at the RX AXI ST interface. For the list and definitions of erroneous RX frames, see **Erroneous RX Frame** on page 35.

etherStatsPkts

This counter reports the total number of RX Ethernet frames received, including frames dropped due to broadcast or address filtering.

Latency

In **Cut Through** mode, the Triple Speed Ethernet MAC core begins transmitting an Ethernet frame before the entire packet is received on the AXI4-Stream interface. This reduces overall latency, as summarized in Table 11: Latency of PHY Interface in Cut Through Mode on page 46. For further details, refer to Cut Through Mode on page 29.

Table 11: Latency of PHY Interface in Cut Through Mode

PHY Interface	Latency	
	TX Path	RX Path
SGMII	7 cycles	7 cycles
GMII	11 cycles	7 cycles

Note: Starting with IP version 7.1, an additional cycle of latency is introduced to the TX path for **Cut Through mode with width adaptation** on Trion devices. This change was implemented to improve timing performance.

IP Manager

The Efinity® IP Manager is an interactive wizard that helps you customize and generate Efinix® IP cores. The IP Manager performs validation checks on the parameters you set to ensure that your selections are valid. When you generate the IP core, you can optionally generate an example design targeting an Efinix development board and/or a testbench. This wizard is helpful in situations in which you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core for different projects.

Note: Not all Efinix IP cores include an example design or a testbench.

Generating the Triple Speed Ethernet MAC Core with the IP Manager

The following steps explain how to customize an IP core with the IP Configuration wizard.

1. Open the IP Catalog.
2. Choose **Ethernet > Triple Speed Ethernet MAC** core and click **Next**. The **IP Configuration** wizard opens.
3. Enter the module name in the **Module Name** box.

   Note: You cannot generate the core without a module name.
4. Customize the IP core using the options shown in the wizard. For detailed information on the options, refer to the *Customizing the Triple Speed Ethernet MAC* section.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an Efinix® development board and/or testbench. These options are turned on by default.
6. (Optional) In the **Summary** tab, review your selections.
7. Click **Generate** to generate the IP core and other selected deliverables.
8. In the **Review configuration generation** dialog box, click **Generate**. The Console in the **Summary** tab shows the generation status.

   Note: You can disable the **Review configuration generation** dialog box by turning off the **Show Confirmation Box** option in the wizard.
9. When generation finishes, the wizard displays the **Generation Success** dialog box. Click **OK** to close the wizard.

The wizard adds the IP to your project and displays it under **IP** in the Project pane.

Generated Files

The IP Manager generates these files and directories:

- **<module name>_define.svh**—Contains the customized parameters.
- **<module name>_tmpl.sv**—Verilog HDL instantiation template.
- **<module name>_tmpl.vhd**—VHDL instantiation template.
- **<module name>.sv**—IP source code.
- **settings.json**—Configuration file.
- **<kit name>_devkit**—Has generated RTL, example design, and Efinity® project targeting a specific development board.
- **Testbench**—Contains generated RTL and testbench files.

Customizing the Triple Speed Ethernet MAC

The core has parameters so you can customize its function. You set the parameters in the **General** tab of the core's IP Configuration window.

Table 12: Triple Speed Ethernet MAC Core Parameters

Name	Options	Description
TSE MAC Version	Any	Triple Speed Ethernet MAC version. Default: 16
Use MAC with Transceiver	Enable, Disable	Enables use of the Triple Speed Ethernet MAC Core with FPGA transceiver (available only on supported devices). When checked, the following options are auto-configured: • PHY Interface Type: SGMII • AXI4 Data Width: 16 • TX FIFO: Disable • RX FIFO: Disable • Register Interface: APB Default: Disable
PHY Interface Type	RGMII, MII, RMII, GMII, SGMII	Defines the PHY interface type. Default: RGMII
AXI4 Data Width	8, 16, 32	Defines AXI4-Stream data width. Default: 8
TX FIFO	Enable, Disable	Enables or disables the TX FIFO in the core. Must be enabled if tx_axis_clk ≠ PHY clock frequency. See Store Forward Mode on page 30. Default: Enable
TX FIFO Depth	16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384	Defines TX FIFO depth. Must be greater than frame length. For jumbo frames > 16384, TX FIFO must be disabled. See Store Forward Mode on page 30 for minimum depth requirements. Default: 2048
RX FIFO	Enable, Disable	Enables or disables the RX FIFO. Must be enabled if rx_axis_clk ≠ PHY clock frequency. Default: Enable
RX FIFO Depth	16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384	Defines RX FIFO depth. Default: 2048
Programmable Inter Packet Gap	8 - 63	Default value for `tx_ipg_length`. Default: 12 **Note:** In **Store-Forward** mode, the effective IPG may be governed by the buffering mechanism.

Name	Options	Description
Maximum Transmission Unit Frame Length	64 – 16018	Default value for `frm_length`. Includes headers, payload, and FCS. Received frames larger than maximum tranmission unit (MTU) are flagged as oversized; payloads < 46 bytes (or < 42 bytes for VLAN Tagged frames) are flagged as undersized. Default: 1518
MAC Source Address	Any	Default value for `mac_addr`. Used when generating TX pause frames and for address filtering comparison. Default: 48'h0000_0000_0000
Broadcast Filtering	Enable, Disable	Default value for `broadcast_filter_en`. Default: Enable
Enable local loopback support	Enable, Disable	Enables or disables local loopback capability on the selected PHY interface. When checked, `loop_ena` can be toggled at run time to enable or disable loopback. Default: Disable
RGMII Transmit Clock Delay	Enable (Center-aligned), Disable (Edge-aligned)	Defines alignment of RGMII transmit clock (rgmii_txc) with data (rgmii_txd). Applies to 10 / 100Mbps only. Default: Enabled (Center-aligned)
First sampling of RGMII Receive Data	Falling Edge, Rising Edge	Defines the first sampling edge for RGMII receive data. Applies to 1000 Mbps only. Default: Falling edge
Register Interface	AXI4-Lite, APB	Defines register interface for accessing the control and status registers in Control and Status Registers on page 55. Default: AXI4-Lite

The maximum capabilities of the Triple Speed Ethernet MAC core depend on the FPGA device family and speed grade.

Table 13: Triple Speed Ethernet MAC core Maximum Capabilities

Device Family	Lower speed grade	Higher speed grade
Trion	X	✓ Max speed: 1 Gbps Jumbo packet size: Up to 4 kB for Store-Forward Mode
Topaz	✓ (Except for SGMII with AXI Data Width = 8)	✓
Titanium	✓	✓

Note:

- Lower speed grade refers to C3 for Trion and Titanium devices, and C2 for Topaz devices.
- Higher speed grade refers to C4 for Trion and Titanium devices, and C3 for Topaz devices.

Ports

Table 14: Triple Speed Ethernet MAC Core Port Interface

Port Name	Direction	Bus Width	Clk Domain	Description
Clock and Reset[4]				
init_clk	input	1	-	Freq = 50MHz or below
init_rst_n	input	1	Async	Reset all Power-Up Handshake between Soft IP and the FPGA PHY.
tx_mac_aclk	input	1	-	See **Table 4: Clock Sources in the Triple Speed Ethernet MAC Core** on page 9 for more details.
rx_mac_aclk	output	1	-	
mac_reset	input	1	Async	Active high system global reset.
proto_reset	input	1	Async	Active high protocol reset on TX and RX path.
MAC AXI ST Interface: TX				
tx_axis_clk	Input	1	-	TX AXI ST Interface. See **TX AXI ST Interface** on page 16 for more details.
tx_axis_mac_tdata	input	AXI_WIDTH	tx_axis_clk	
tx_axis_mac_tvalid	input	AXI_WIDTH/8	tx_axis_clk	
tx_axis_mac_tlast	input	1	tx_axis_clk	
tx_axis_mac_tstrb	input	AXI_WIDTH/8	tx_axis_clk	
tx_axis_mac_tuser	input	1	tx_axis_clk	
tx_axis_mac_tready	output	1	tx_axis_clk	
MAC AXI ST Interface: RX				
rx_axis_clk	input	1	-	RX AXI ST Interface. See **RX AXI ST Interface** on page 18 for more details.
rx_axis_mac_tdata	output	AXI_WIDTH	rx_axis_clk	
rx_axis_mac_tvalid	output	1	rx_axis_clk	
rx_axis_mac_tlast	output	1	rx_axis_clk	
rx_axis_mac_tstrb	output	AXI_WIDTH/8	rx_axis_clk	
rx_axis_mac_tuser	output	1	rx_axis_clk	
rx_axis_mac_tready	input	1	rx_axis_clk	
GMII Interface				
gm_tx_c	output	1	-	Provides timing reference for the transfer of `gm_tx_` signals to the PHY. Source from `tx_mac_aclk`.
gm_tx_d	output	GMII_WIDTH	gm_tx_c	GMII transmits data bus to PHY.
gm_tx_en	output	GMII_WIDTH/8	gm_tx_c	Indicates valid data on the transmit data bus when asserted.

[4] For details on Clock and Reset, refer to **Clock Sources** and **5.3 Reset Signals**.

Port Name	Direction	Bus Width	Clk Domain	Description
gm_tx_err	output	GMII_WIDTH/8	gm_tx_c	Indicates to the PHY that the transmitted frame is invalid when asserted.
gm_rx_c	input	1	–	GMII receives clock sourced from the PHY.
gm_rx_d	input	GMII_WIDTH	gm_rx_c	GMII receives data bus from PHY.
gm_rx_dv	input	GMII_WIDTH/8	gm_rx_c	Indicates valid data on the receive data bus when asserted. Continue to assert this signal during frame reception, from the first preamble byte until the last byte of the CRC field is received.
gm_rx_err	input	GMII_WIDTH/8	gm_rx_c	Indicates that the received frame contains errors when asserted.
RGMII Interface				
rgmii_txc_HI	output	1	–	MSB of the RGMII transmit clock signal to the PHY. Connect to **Pin Name (HI)** if the **Double Data I/O** option turns on for `rgmii_txc` bus in the Interface Designer. Set the clock pin name reference to **90°** phase shift of `tx_mac_aclk`.
rgmii_txc_LO	output	1	–	LSB of RGMII transmit clock signal to the PHY. Connect to **Pin Name (LO)** if the **Double Data I/O** option turns on for `rgmii_txc` bus in Interface Designer. Set the clock pin name reference to **90°** phase shift of `tx_mac_aclk`.
rgmii_txd_HI	output	4	tx_mac_aclk	The 4 most significant bits (MSB) transmit data to the PHY. Connect to **Pin Name (HI)** if the **Double Data I/O** option turns on for `rgmii_txd` bus in Interface Designer. Set the clock pin name reference to `tx_mac_aclk`.
rgmii_txd_LO	output	4	tx_mac_aclk	4 least significant bit (LSB) of RGMII transmit data to the PHY. Connect to **Pin Name (LO)** if the **Double Data I/O** option turns on for `rgmii_txd` bus in Interface Designer. Set the clock pin name reference to `tx_mac_aclk`.
rgmii_tx_ctl_HI	output	1	tx_mac_aclk	MSB of the transmit control signal to the PHY. Connect to **Pin Name (HI)** if the **Double Data I/O** option turns on for `rgmii_tx_ctl` bus in Interface Designer. Set the clock pin name reference to `tx_mac_aclk`.

Port Name	Direction	Bus Width	Clk Domain	Description
rgmii_tx_ctl_LO	output	1	tx_mac_aclk	LSB of transmit control signal to the PHY. Connect to **Pin Name (LO)** if the **Double Data I/O** option turns on for `rgmii_tx_ctl` bus in Interface Designer. Set the clock pin name reference to `tx_mac_aclk`.
rgmii_rxc	input	1	–	RX clock from PHY. See **Table 4: Clock Sources in the Triple Speed Ethernet MAC Core** on page 9 for frequency requirement.
rgmii_rxd_HI	input	4	rgmii_rxc	4 MSB of RGMII receive data from the PHY. Connect to **Pin Name (HI)** if the **Double Data I/O** option turns on for `rgmii_rxd` bus in Interface Designer. Set the clock pin name reference to `rgmii_rxc`.
rgmii_rxd_LO	input	4	rgmii_rxc	4 LSB of RGMII receive data from the PHY. Connect to **Pin Name (LO)** if the **Double Data I/O** option turns on for `rgmii_rxd` bus in the Interface Designer. Set the clock pin name reference to `rgmii_rxc`.
rgmii_rx_ctl_HI	input	1	rgmii_rxc	MSB of receive control signal from the PHY. Connect to **Pin Name (HI)** if the **Double Data I/O** option turns on for `rgmii_rx_ctl` bus in Interface Designer. Set the clock pin name reference to `rgmii_rxc`.
rgmii_rx_ctl_LO	input	1	rgmii_rxc	LSB of receive control signal from the PHY. Connect to **Pin Name (LO)** if the **Double Data I/O** option turns on for `rgmii_rx_ctl` bus in Interface Designer. Set the clock pin name reference to `rgmii_rxc`.
MII Interface				
mii_txc	input	1	–	TX clock from PHY. See **Table 4: Clock Sources in the Triple Speed Ethernet MAC Core** on page 9 for frequency requirement.
mii_txd	output	4	mii_txc	Transmit data to PHY.
mii_tx_dv	output	1	mii_txc	Transmit data valid.
mii_tx_err	output	1	mii_txc	An assertion indicates that the PHY that the frame was sent is invalid.
mii_rxc	input	1	–	RX clock from PHY. See **Table 4: Clock Sources in the Triple Speed Ethernet MAC Core** on page 9 for frequency requirement.

Port Name	Direction	Bus Width	Clk Domain	Description
mii_rxd	input	4	mii_rxc	Receive data from PHY.
mii_rx_dv	input	1	mii_rxc	Data valid. This signal indicates that the data on `mii_rxd [3:0]` is valid. The signal stays asserted during frame reception, from the first preamble byte until the last byte of the CRC field is received.
mii_rx_err	input	1	mii_rxc	Assert this signal to indicate that the received frame contains errors.
RMII Interface				
rmii_clk_ref	input	1	–	See Table 4: Clock Sources in the Triple Speed Ethernet MAC Core on page 9.
rmii_txd	output	2	rmii_clk_ref	Transmit data to the PHY.
rmii_tx_en	output	1	rmii_clk_ref	The assertion indicates that the data on the transmit data bus is valid.
rmii_rxd	input	2	rmii_clk_ref	Receive data from the PHY.
rmii_crs_dv	input	1	rmii_clk_ref	Receive valid data strobe.
rmii_rx_err	input	1	rmii_clk_ref	Assert this signal to indicate that the received frame contains errors.
APB Interface				
s_paddr	input	10	s_axi_aclk	APB address.
s_pwdata	input	32	s_axi_aclk	APB write data.
s_pwrite	input	1	s_axi_aclk	APB write.
s_penable	input	1	s_axi_aclk	APB enable.
s_psel	input	1	s_axi_aclk	APB select.
s_prdata	output	32	s_axi_aclk	APB ready data.
s_pready	output	1	s_axi_aclk	APB ready.
s_perr	output	1	s_axi_aclk	APB slave error.
AXI-4 Lite Interface				
s_axi_aclk	input	1	–	Register interface clock. SeeTable 4: Clock Sources in the Triple Speed Ethernet MAC Core on page 9 for frequency requirement.
s_axi_awaddr	input	10	s_axi_aclk	AXI4-Lite write address bus.
s_axi_awvalid	input	1	s_axi_aclk	AXI4-Lite write address valid strobe.
s_axi_awready	output	1	s_axi_aclk	AXI4-Lite write address ready signal.
s_axi_wdata	input	32	s_axi_aclk	AXI4-Lite write data.
s_axi_wvalid	input	1	s_axi_aclk	AXI4-Lite write data valid strobe.
s_axi_wready	output	1	s_axi_aclk	AXI4-Lite write-ready signal.
s_axi_bresp	output	2	s_axi_aclk	AXI4-Lite write response.
s_axi_bvalid	output	1	s_axi_aclk	AXI4-Lite write response valid strobe.
s_axi_bready	input	1	s_axi_aclk	AXI4-Lite write response ready signal.

Port Name	Direction	Bus Width	Clk Domain	Description
s_axi_araddr	input	10	s_axi_aclk	AXI4-Lite read address bus.
s_axi_arvalid	input	1	s_axi_aclk	AXI4-Lite read address valid strobe.
s_axi_arready	output	1	s_axi_aclk	AXI4-Lite read address ready signal.
s_axi_rresp	output	2	s_axi_aclk	AXI4-Lite read response.
s_axi_rdata	output	32	s_axi_aclk	AXI4-Lite read data.
s_axi_rvalid	output	1	s_axi_aclk	AXI4-Lite read data valid strobe.
s_axi_rready	input	1	s_axi_aclk	AXI4-Lite read data ready signal.
MDIO Interface				
Mdo	output	1	–	Output data signal for communication with the PHY's configuration and status registers.
MdoEn	output	1	–	Output data enable signal for the MDIO bidirectional bus. In the Interface Designer, you have to create an `INOUT` tristate port, set **Mdi** as **input**, **Mdo** as **output**, and **MdoEn** as **output enable**.
Mdi	input	1	–	Input data signal for communication with the PHY's configuration and status registers. Always set to high when not in use.
Mdc	output	1	–	MDIO clock management . Derived from s_axi_aclk based on supplied configuration data.
PHY Power Up Sequence				
PMA_CMN_READY	input	1	Async	Output from PHY to indicate the readiness of PHY calibration
PMA_XCVR_PLLCLK_EN_ACK	input	1	Async	Refer to Power Up Handshake with FPGA Transceiver on page 14.
PMA_XCVR_POWER_STATE_ACK	input	4	Async	
PMA_RX_SIGNAL_DETECT	input	1	Async	
PMA_XCVR_PLLCLK_EN	output	1	init_clk	
PMA_XCVR_POWER_STATE_REQ	output	4	init_clk	
phy_init_done	output	1	Async	

Control and Status Registers

As a general guideline, control registers should be programmed only when no data traffic is in progress.

Modifying configuration settings while the transmission or receiving traffic is active may result in unpredictable behavior or data corruption.

In practice, you are recommended to program control registers only when the datapath is disabled (by clearing `tx_ena` or `rx_ena`), with the following exceptions that may be configured at any time:

- `xon_gen`
- `xoff_gen`
- `pause_quant`
- `tx_ena`
- `rx_ena`

Additionally, the following registers may be updated safely when the TX AXI ST interface is idle:

- `tx_addr_ins`
- `tx_dst_addr_ins`
- `dst_mac_addr`

Table 15: Triple Speed Ethernet MAC Core Configuration Registers

Dword Offset	Name	R/W	Description
0x00	VERSION	R	Indicates the version of the IP core. Default: The default value is configurable through **TSE MAC Version** field in the IP Configuration window.
0x08	Command_Config	R/W	Command configuration settings. Refer to Table 16: Command_Config Register Field Description on page 56 for more details.
0x0C	mac_addr [31:0]	R/W	Source MAC address. Example: If the MAC address is set to 01-1B-43-17-7B-CD: Write 0x43177BCD to register 0x0C Write 0x0000011B to register 0x10 Default: The default value is configurable through **MAC Source Address** field in the IP Configuration window.
0x10	mac_addr [47:32]	R/W	
0x14	frm_length [15:0]	R/W	Indicates the maximum packet length. Default: The default value is configurable through **Maximum Transmission Unit Frame Length** field in the IP Configuration window.
0x18	pause_quant [15:0]	R/W	Defines the pause quanta value inserted into the generated PAUSE frame triggered by xoff_gen. Each PAUSE frame specifies a requested pause duration in units of pause quanta, where one quanta equals 512 bit times. Example: If pause_quant is set to 8, the pause time equals 8 × 512-bit times. Default: 0x0
0x5c	tx_ipg_length [5:0]	R/W	Transmit interpacket gap value. Default: The default value is configurable through the **Programmable Inter Packet Gap** field in the IP Configuration window.

Table 16: Command_Config Register Field Description

Bit	Name	R/W	Description
0	tx_ena	R/W	Enables the transmission path. Default: 1
1	rx_ena	R/W	Enables the receive path. Default: 1
2	xon_gen	R/W	Pause frame generation. Assert high to trigger a pause frame with 0 pause quanta to resume transmission. See **Send Pause Frame at TX** on page 40 for more details. Default: 0
3	Reserved		
4	promis_en	R/W	Promiscuous mode. Assert high to disable MAC address filtering. Default: 0
5	Reserved		
6	crc_fwd	R/W	Assert high to forward the CRC value to the AXI4-Stream RX data rx_axis_mac_tdata. Default: 0
7	Reserved		
8	pause_ignore	R/W	Assert high to disable pause frame control. Default: 0
9	tx_addr_ins	R/W	Assert high to insert the transmit packet's source MAC address with the value in registers 0xC and 0x10. Default: 0
10 -14	Reserved		
15	loop_ena	R/W	Controls local PHY loopback at runtime. When `loop_ena = 1`, the MAC's TX path is looped back to the RX path on the selected PHY interface. **Note:** Local loopback support must be enabled in the IP Configuration window for this register bit to have any effect. Default: 0
16 - 18	eth_speed [2:0]	R/W	Ethernet MAC transfer speed. For RGMII and RMII interface, correct configuration of the `eth_speed` setting is required to ensure proper clock generation and data sampling behavior. The setting must be configured before any traffic starts. 3'b100: 1000 Mbps 3'b010: 100 Mbps 3'b001: 10 Mbps Default: • RGMII: 3'b100 • RMII: 3'b010 **Note:** For MII, GMII, and SGMII modes, the `eth_speed` setting is not applicable and can be ignored.
19 - 21	Reserved		

Bit	Name	R/W	Description
22	xoff_gen	R/W	Triggers the transmission of a pause frame. When set to 1, the MAC generates a pause frame using the quanta value specified in pause_quant (0x18). The pause frame requests the link partner to pause transmission for pause_quant × 512 bit times. See **Send Pause Frame at TX** on page 40 for additional details on pause frame operation. Default: 0
23 - 30	Reserved		
31	cnt_reset	R/W	Assert high to reset all the statistic counter registers. Default: 0 **Note:** For accurate counter values, this reset should only be performed when the Triple Speed Ethernet MAC core is idle (no active TX or RX operations). To ensure this, follow this sequence: 1. Disable the datapath by clearing `tx_ena` and `rx_ena`. 2. Assert `cnt_reset` to clear counters. 3. Re-enable `tx_ena` and `rx_ena`.

Table 17: MDIO Configuration Registers

Dword Offset	Name	R/W	Description
0x100	Bit 7:0 – Divider	R/W	To derive the Mdc clock frequency. The Mdc frequency = s_axi_aclk frequency / Divider. Default: 0x64
	Bit 8 - NoPre		0: Send the MDIO with preamble data. 1: Send the Mdio without preamble data. Default: 0x0
0x104	Bit 0 - RdEn	System Controlled	Assert high to read the MDIO bus. Default: 0x0
	Bit 1 - WrEn		Assert high to write to the MDIO bus. Default: 0x0
0x108	Bit [4:0] – RegAddr	R/W	MDIO configuration register address. Default: 0x0
	Bit [7:5] – Reserved		–
	Bit [12:8] - PhyAddr		5-bit PHY address. Default: 0x0
0x10c	WrData [15:0]	R/W	MDIO write data. Default: 0x0
0x110	RdData [15:0]	R	MDIO read data. Default: 0x0
0x114	Bit 0 - LinkFailStatus	R	Fail to link with the PHY MDIO. Default: 0x0
	Bit 1 - BusyStatus		PHY MDIO is busy. Default: 0x0
	Bit 2 - NvalidStatus		Invalid Status (qualifier for the valid scan result). Default: 0x0

Table 18: Receive Supplementary Registers

Dword Offset	Name	R/W	Description
0x140	broadcast_filter_en	R/W	Controls the filtering of broadcast packets. When set to 1, all received packets with the broadcast destination MAC address (FF:FF:FF:FF:FF:FF) are discarded by the MAC. When 0, broadcast packets are accepted. See **Address Filtering and Broadcast Filtering at RX** on page 37 for more details. Default: The default value is configurable through the **Broadcast Filtering** field in the IP Configuration window.
0x144	mac_addr_mask [31:0]	R/W	Specifies the MAC address mask for address filtering. Mask bits set to 1 are compared; bits set to 0 are ignored. Effective only when promis_en = 0. Refer to **Address Filtering and Broadcast Filtering at RX** on page 37 for more details. Default: 0x0
0x148	mac_addr_mask [47:32]	R/W	

Table 19: Transmit Supplementary Registers

Dword Offset	Name	R/W	Description
0x180	tx_dst_addr_ins [0]	R/W	Assert high to insert the destination MAC address value in 0x184 and 0x188 to TX packet destination MAC address data bytes. Default: 0x0
0x184	dst_mac_addr [31:0]	R/W	Destination MAC address. Default: 0x0
0x188	dst_mac_addr [47:32]	R/W	

Table 20: Statistic Counter Registers

For detailed information, refer to **Statistic Reporting** on page 42.

Dword Offset	Name	R/W	Description
0x68	aFramesTransmittedOK [15:0]	R	Frames successfully transmitted, including pause frames. Default: 0x0
0x6c	aFramesReceivedOK [15:0]	R	Frames successfully received, excluding error frames. Includes pause frames. Default: 0x0
0x70	aFrameCheckSequenceErrors [15:0]	R	Received frames with CRC errors. Default: 0x0
0x80	aTxPAUSEMACCtrlFrames [15:0]	R	Pause frames transmitted. Default: 0x0
0x84	aRxPAUSEMACCtrlFrames [15:0]	R	Pause frames received. Default: 0x0
0x88	ifInErrors [15:0]	R	Erroneous RX frames and frames dropped due to broadcast or address filtering. Default: 0x0
0x8c	ifOutErrors [15:0]	R	Frames terminated as a bad frame on the TX channel. Default: 0x0
0x98	aRxFrameMismatchedLength [15:0]	R	Received frames with mismatched physical length vs Length field. Default: 0x0
0x9c	aRxFilterFramesErrors [15:0]	R	Filtered packets due to address mismatch. Default: 0x0
0xb4	etherStatsPkts [15:0]	R	Total frames received, including erroneous frames. Default: 0x0
0xb8	etherStatsUndersizePkts [15:0]	R	Frames < 64 bytes (excludes error frames). Includes pause frames. Default: 0x0
0xbc	etherStatsOversizePkts [15:0]	R	Frames exceeding frm_length (excludes error frames). Default: 0x0
0xc0	aFramesReceivedLen[15:0]	R	Length of received frame. Default: 0x0
0xc4	aTxFifoOverflowFramesErrors[15:0]	R	Transmitted frames discarded due to TX FIFO overflow in **Store Forward** mode. Default: 0x0
0xc8	aTxIncontinuityFramesErrors[15:0]	R	Transmitted frames with continuity errors. Default: 0x0

Triple Speed Ethernet MAC Example Design

You can choose to generate the example design when generating the core in the IP Manager Configuration window. Compile the example design project and download the **.hex** or **.bit** file to your board. To generate example design, the **Example Design Deliverables Option** signal must be enabled.

Important: Efinix tested the example design generated with the default parameter options only.

Example Design with External PHY

The example design with external PHY demonstrates the functionality of the Triple Speed Ethernet MAC core running at 1000 Mbps. It includes two test modes:

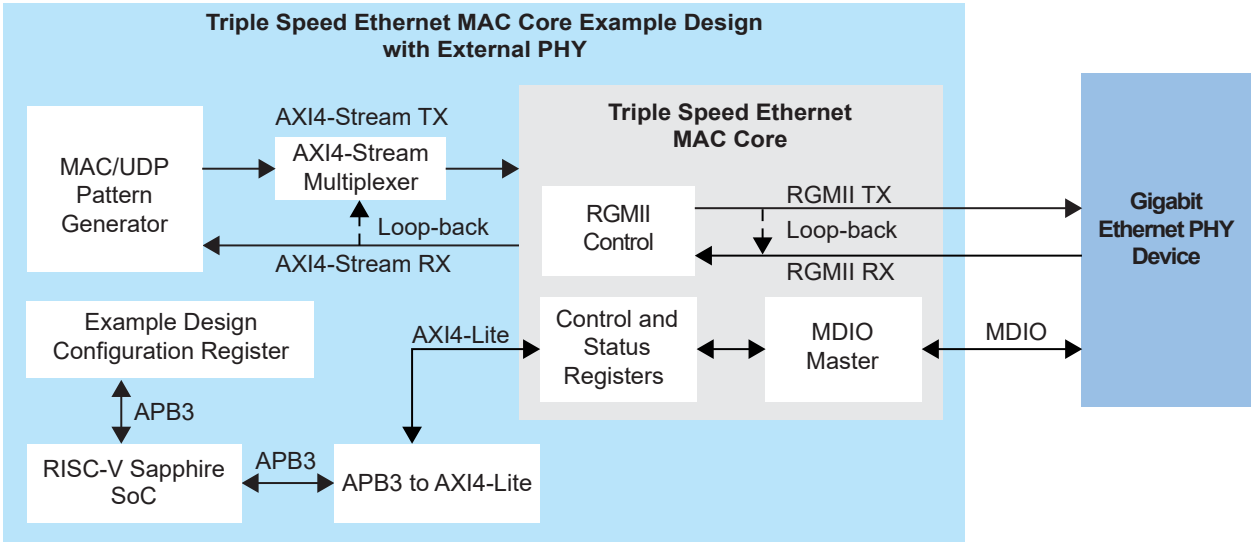
- *Normal mode*—The MAC or UDP pattern generator constructs and initiates TX packets for the TX path of the Triple Speed Ethernet MAC core.
- *Linked-Partner mode*—The TX packets are constructed by the other end of the network, for example a computer.

The example design consists of:

- *MAC or UDP pattern generator*—Generates the MAC/UDP TX packet to the Triple Speed Ethernet MAC core though AXI4-ST TX interface
- *RISC-V Sapphire SOC*—Allows you to configure the Triple Speed Ethernet MAC core configuration register and example design configuration register. Refer to Table 23: Configuration Registers for Example Design with External PHY on page 74 for more details.

Note: The example design with external PHY is validated with RISC-V Sapphire core version 2.2. You are advised not to upgrade the SoC core to a newer version, as this may lead to system incompatibility or functional failure.

Figure 35: Example Design with External PHY Block Diagram



Required Hardware

The example design with external PHY uses the following hardware from the Trion T120 BGA324 Development Kit:

- Trion T120 BGA324 development board
- Micro-USB cable
- 12 V power adapter

You also need the following hardware, which you provide yourself:

- USB-to-UART converter module
- Ethernet cable
- Computer

The example design with external PHY uses the following hardware from the Titanium Ti60 F225 Development Kit:

- Titanium Ti60 F225 Development Board
- USB-C data cable
- 12 V power adapter
- Ethernet Connector Daughter Card

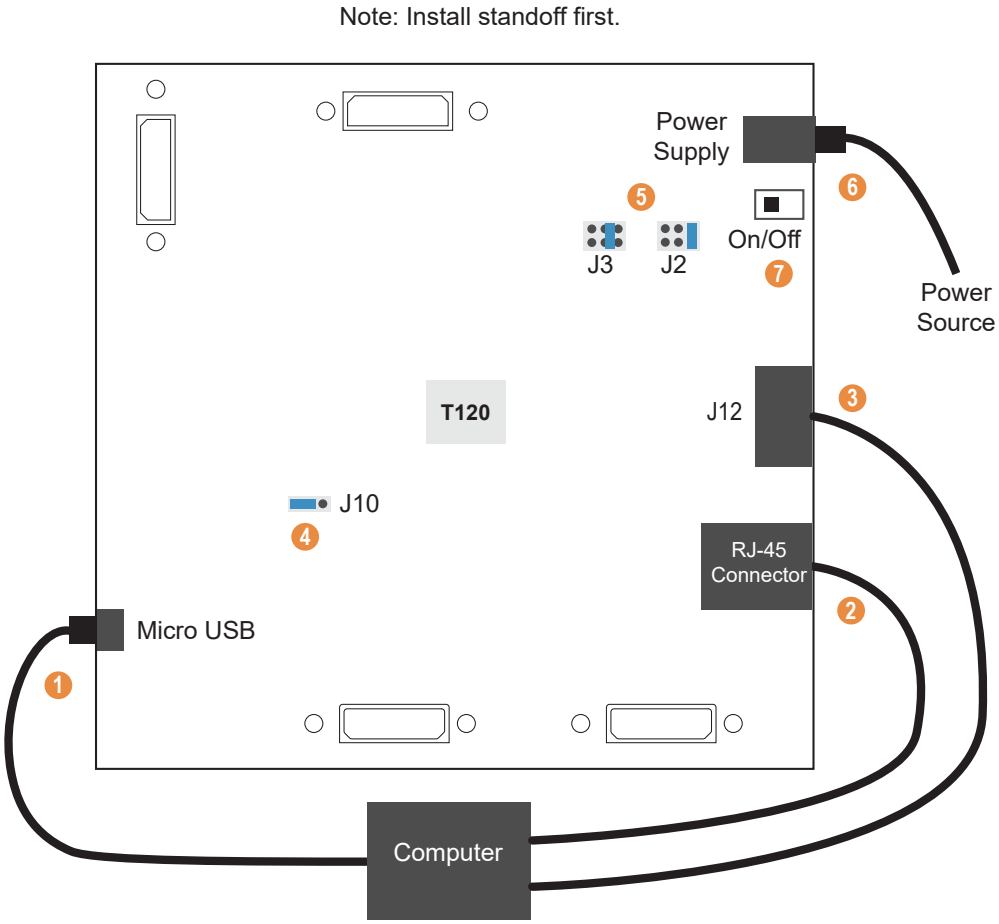
You also need the following hardware, which you provide yourself:

- Ethernet cable
- Computer

Hardware Set Up for Trion® T120 BGA324 Development Board

The following figure shows the hardware setup steps for Trion T120 BGA324 Development Board example design. If you have not already done so, attach standoffs to the board.

Figure 36: Trion® T120 BGA324 Development Board Hardware Setup



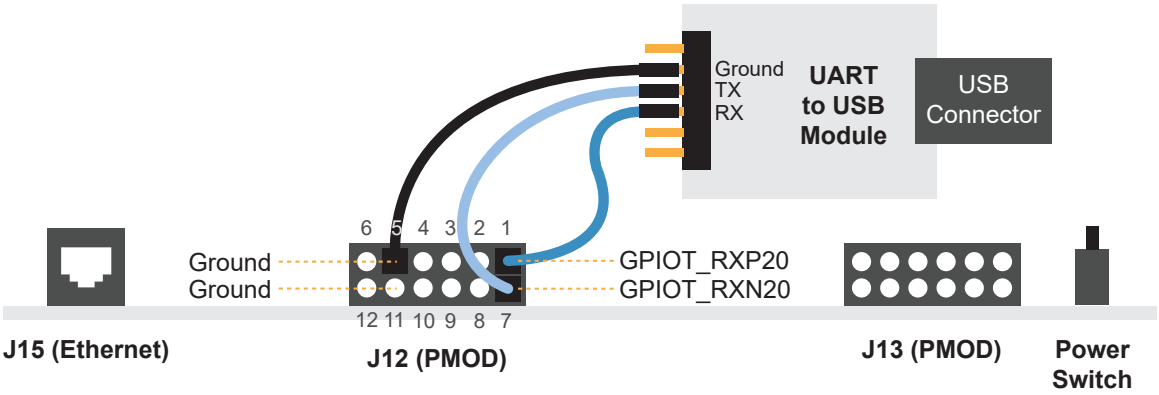
1. Connect a USB cable to the board and to your computer.
2. Connect an Ethernet cable to the RJ-45 receptable on the board and to your computer.
3. Connect a USB-to-UART module to the J12 header. See Set Up a USB-to-UART Module on page 64.
4. On header J10, connect pins 2 – 3 with a jumper (default) to use the on-board oscillator.
5. Leave the jumper on J2 at the defaults (connecting pins 1 -2). On J3, connect pins 1 – 2.
6. Connect the 12 V power supply to the board connector from a power source.
7. Turn on the board.

Note: Ensure that you install the driver for the correct interface. Refer to the Trion T120 BGA324 Development Kit User Guide for information about installing the drivers.

Set Up a USB-to-UART Module

The Trion T120 BGA324 Development Board does not have a USB-to-UART converter. Therefore, you need to use a separate USB-to-UART converter module. A number of modules are available from various vendors; any USB-to-UART module should work.

Figure 37: Connect the UART Module to I/O header J12

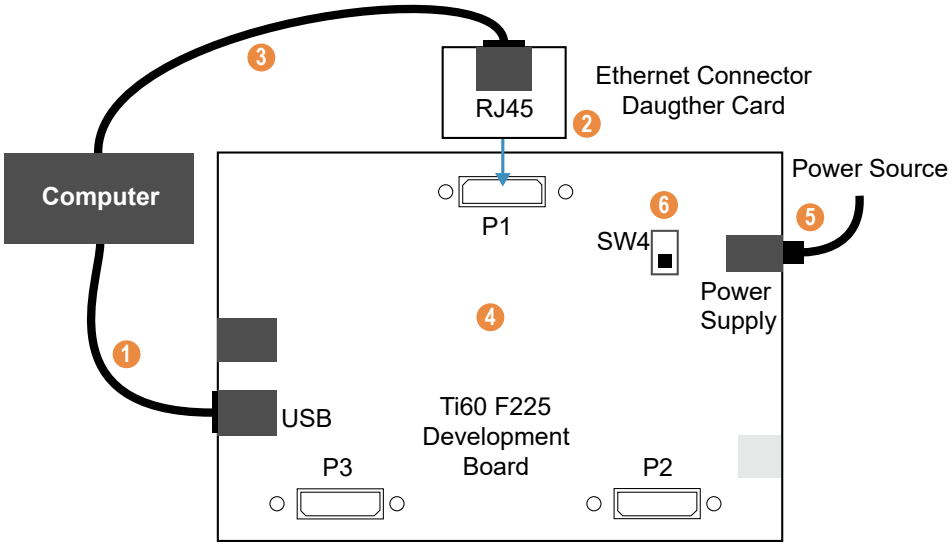


1. Connect the UART's RX, TX and ground pins to the Trion T120 BGA324 Development Board using:
   - RX – GPIOT_RXP20, which is labeled as 1 on header J12
   - TX – GPIOT_RXN20, which is labeled as 7 on header J12
   - Ground – Ground, connect to one of the GND pins on header J12
2. Plug the UART module into a USB port on your computer. The driver should install automatically if needed.

Hardware Set Up for Titanium Ti60 F225 Development Board

The following figure shows the hardware setup steps for Titanium Ti60 F225 Development Board example design. If you have not already done so, attach standoffs to the board.

Figure 38: Titanium Ti60 F225 Development Board Hardware Setup



1. Connect a USB-C cable to the board and to your computer.
2. Attach the Ethernet Connector Daughter Card to header P1 of the development board.
3. Connect an Ethernet cable to the RJ-45 receptacle on the Ethernet Connector Daughter Card and to your computer.
4. Leave the board jumper settings to default. Refer to the development kit User Guide for information about the default jumper settings.
5. Connect the 12 V power supply to the board connector from a power source.
6. Turn on the board.

Note: Ensure that you install the driver for the correct interface. Refer to the Titanium Ti60 F225 Development Kit User Guide for information about installing the drivers.

Set Up a USB-to-UART Module

The Titanium Ti60 F225 development board has a USB-to-UART converter connected to the Ti60's `GPIOL_01` and `GPIOL_02` pins. To use the UART, connect a USB cable to the FTDI USB connector on the development board and to your computer.

Note: The board has an FTDI chip to bridge communication from the USB connector. FTDI interface 2 on Ti60 communicates with the on-board UART. You do not need to install a driver for this interface because when you connect the Titanium Ti60 F225 development board to your computer, Windows automatically installs a driver for it.

Using the Example with the RISC-V SDK

Before working with the software included with this example design, you should already be familiar with Sapphire SoC and RISC-V SDK. Specifically, you should know how to:

- Launch Eclipse using the **run_eclipse.bat** file (Windows) or **run_eclipse.sh file** (Linux) scripts.
- Set up global environment variables.
- Create and build a software project.
- Debug using the OpenOCD debugger.
- Open a UART terminal.

Learn more: Refer to the RISC-V SDK chapter of the Sapphire RISC-V SoC Hardware and Software User Guide for detailed instructions on how to perform these tasks.

Test Modes

The example design with external PHY includes software for the Sapphire SoC in the tseDemo directory. In addition to the source code, Efinix provides compile binary files using predefined settings so you can easily try out the two test modes:

- Normal Test Mode – Use the **tseDemo.elf** file in the **build\test_mode_0** directory.
- Linked-partner Test Mode – Use the **tseDemo.elf** file in the **build\test_mode_1** directory.

You can also customize the test modes by changing the variables in the Sapphire SoC software driver code, **embedded_sw\sapphire\bsp\efinix\EfxSapphireSoc\app\tseDemo.h** file. However, you must rebuild the software if you change any variable.

Figure 39: Defining Variables in tseDemo.h File

```
/************************** Project Header File **************************/
#define PRINTF_EN           1
#define TEST_MODE           1   //0 : Normal Mode; 1 : Link partner Test Mode;

#define PAT_NUM             100
#define PAT_DLEN            100
#define PAT_IPG             100
#define PAT_TYPE            0  //0 : UDP Pattern; 1 : MAC Pattern;
#define DST_MAC_H          0xffff
#define DST_MAC_L          0xffffffff
#define SRC_MAC_H          0xeae8
#define SRC_MAC_L          0x5e0060c8
#define SRC_IP             0xc0a80164
#define DST_IP             0xc0a80165
#define SRC_PORT           0x0521
#define DST_PORT           0x2715
```

Normal Mode Test

In Normal mode, the MAC or UDP pattern generator constructs and initiates TX packets through the TX path of the MAC core.

Figure 40: Normal Mode Data Flow

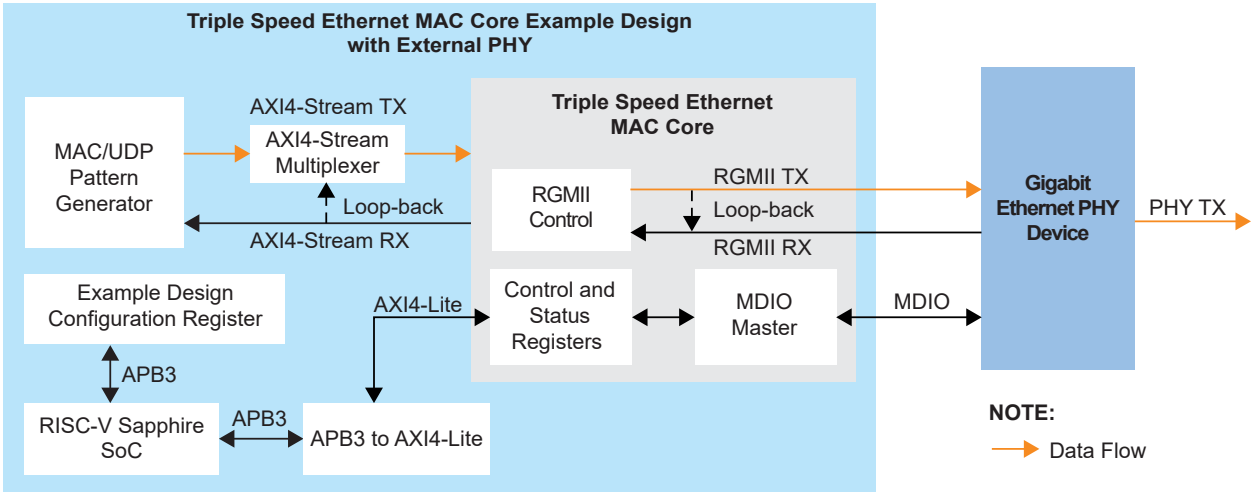


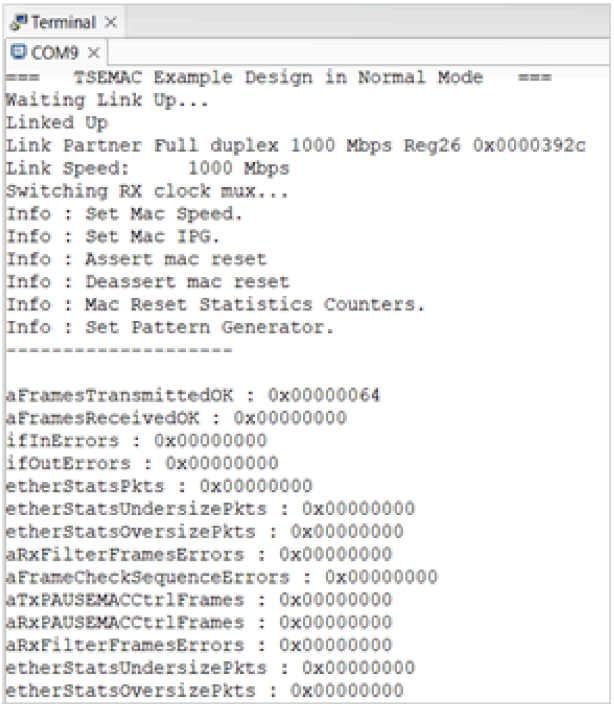
Table 21: Normal Mode Macros

Macro Name	Description
PAT_NUM	Number of packets to transmit. Setting PAT_NUM = 0 enables unlimited packet transmission.
PAT_DLEN	Length of the UDP/ MAC pattern transmit packet (payload length).
PAT_IPG	Interpacket gap (IPG) between frames. One IPG unit equals 8ns.
PAT_TYPE	Type of packet pattern: • 0: MAC pattern • 1: UDP pattern

The following are the criteria for the test:

- If `PAT_NUM = 0`, the `aFramesTransmittedOK` counter continues to increment until it wraps around at `0xFFFF`
- If `PAT_NUM` is non-zero, the `aFramesTransmittedOK` counter matches the programmed `PAT_NUM`.
- The `ifOutErrors` counter remains at `0`.
- The captured packet in Wireshark matches the transmitted pattern.

Figure 41: Normal Mode Sample Telnet Session Output

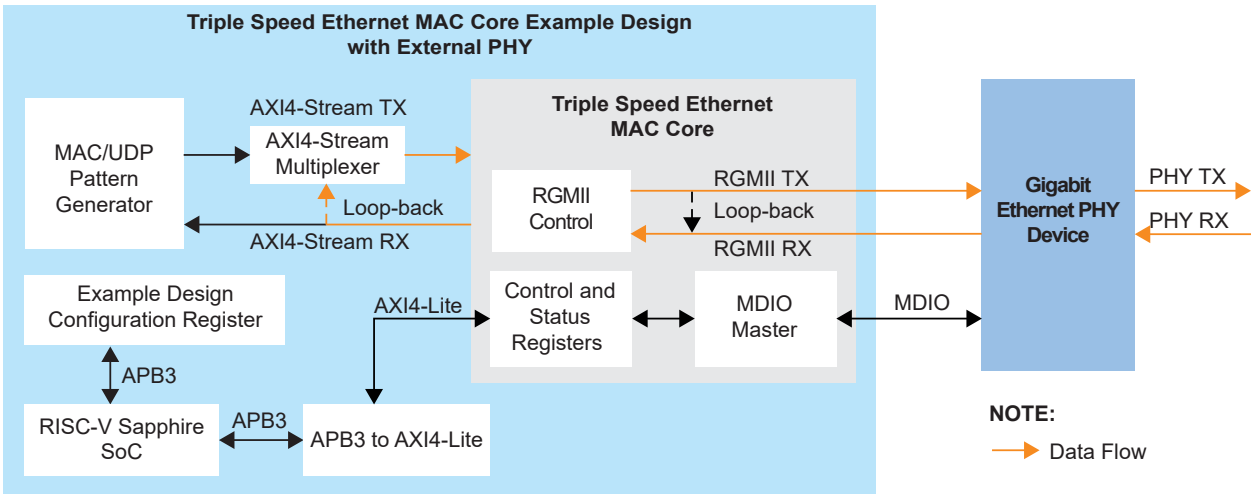


Linked Partner Test

In Linked-Partner mode, the transmit packet are constructed and sent by an external network device, such as a PC or another Ethernet-enabled system. The AXI4-Stream loop-back is enabled in this mode, allowing the core to receive and re-transmit incoming packets directly through the enabled AXI4-Stream loop-back path without using the internal MAC or a UDP pattern generator.

This mode demonstrates the MAC RX or TX datapath integration with an external Ethernet partner and verifies successful bi-directional data transfer between the FPGA and a real network device.

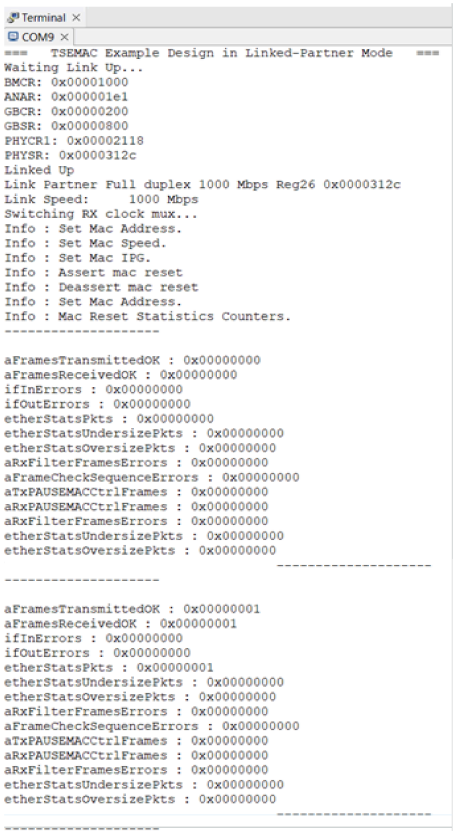
Figure 42: Linked Partner Mode Data Flow



The test is a 'PASS' when the following conditions are achieved:

- `aFramesTransmittedOK` and `aFramesReceivedOK` counters are equal.
- The `ifInErrors` counter remains at `0`.
- The packets captured using Wireshark match the packets transmitted by the external device.

Figure 43: Linked Partner Mode Sample Telnet Session Output



Note: You can use any terminal program (such as Putty, Termite, or the built-in Eclipse Terminal to monitor the statistic counters - `aFramesTransmittedOK` and `ifOutErrors`.

Realigning the PHY Clock

To ensure proper transmission and reception, the PHY clock must be correctly aligned with the data signals:

There are two ways to realign the PHY clock with respect to data:

- Option 1: Use the **PHY RGMII clock-to-data delay** in the Sapphire SoC software driver function.
- Option 2: Configure the **RGMII Transmit Clock Delay** and **First Sampling of RGMII Received Data** parameter in the General tab of the IP Configuration window.

Using the PHY RGMII Clock-to-Data Delay (Trion only)

For Trion devices, you can control the transmit and receive clock delay of the PHY through the Sapphire SoC software driver.

This is done using the `PhyDlySetRXTX()` function in the **main.c** file.

To adjust the `TX_delay` and `RX_delay`, modify the function call in: **\embedded_sw \sapphire\software\standalone\tseDemo\src\main.c**

This method allows flexible software-level tuning of the PHY delay values to achieve optimal RGMII clock-to-data alignment for both transmit and receive paths.

Figure 44: Defining Delay in main.c Function File

```
void main() {
    u32 prompt=0;
    u32 speed;

    PhyNego(1);
    bsp_init();
    if(TEST_MODE == 1){
     bsp_print("===   TSEMAC Example Design in Linked-Partner Mode   ===");
    } else {
     bsp_print("===   TSEMAC Example Design in Normal Mode   ===");
    }
    PhyDlySetRXTX(8, 0); //change PHY RX_Delay and Tx_Delay settings here
    speed = PhyNormalInit();
    MacNormalInit(speed);
    StartRunning();
}
```

Table 22: RX_delay and TX_delay Delay Settings

Delays	Values	Description
RX_delay	0 – 15	Clock to data setup delay for RGMII RX path with 1000 Mbps. Recommended values: • 0: Smallest delay of around 0.81 ns • 2: Delay of around 1.1 ns • 8: Delay of around 2.17 ns • 15: Largest delay of around 2.92 ns
TX_delay	0 – 15	Clock to data setup delay for RGMII TX path with 1000 Mbps. Recommended values: • 0: Smallest delay of around 0.021 ns • 2: Delay of around 0.4 ns • 8: Delay of around 1.7 ns • 15: Largest delay of around 3.3 ns

RGMII Dynamic Clock Switch (Titanium only)

In RGMII interfaces, clock-to-data skew caused by PCB trace length mismatches between the clock (`rgmii_rxc`) and data (`rgmii_rxd`) lines is a critical factor that affects reliable data sampling. At 1000 Mbps, even a small difference in trace length can lead to a timing misalignment of several hundred ps which is significant for double data rate (DDR) sampling.

To compensate for this skew and ensure that the clock edge is properly aligned with the center of the data valid window, the example design uses a PLL with core feedback. The PLL reconstructs an internal receive clock that is phase-aligned with the incoming RGMII signals at the FPGA input pins, effectively cancelling the skew introduced by PCB routing. This ensures accurate setup and hold timing at 1000 Mbps operation.

The receive clock (`rgmii_rxc`) provided by the external PHY operates at different frequencies depending on the negotiated Ethernet link speed:

- 2.5 MHz for 10 Mbps
- 25 MHz for 100 Mbps
- 125 MHz for 1000 Mbps

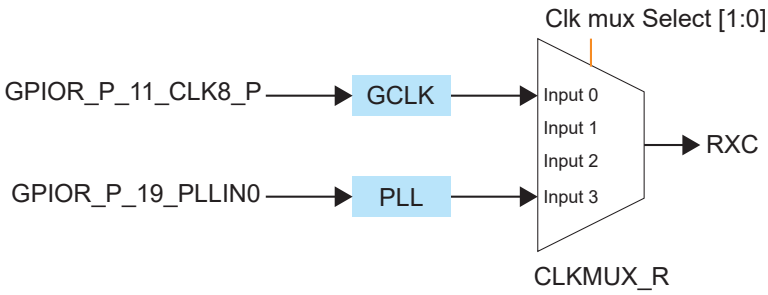
Because the PLL input path has a minimum input frequency requirement and cannot reliably lock at very low frequencies such as 2.5 MHz, a dynamic clock mux is implemented to support all link speeds.

The dynamic clock mux automatically switches between the PLL and the direct GPIO clock path based on the detected link speed:

- At 10/100 Mbps, the RX clock is too slow for PLL operation, so the PHY RX clock is routed directly through the GPIO input path.
- At 1000 Mbps, the faster 125 MHz RX clock is routed through the PLL, enabling phase alignment with core feedback to compensate for PCB trace mismatch and maintain timing accuracy.

This allows the same RGMII interface to dynamically support 10/100/1000 Mbps operation while maintaining clock integrity, data alignment and timing reliability across all link speeds, even when PCB trace skew between the clock and data lines exists.

Figure 45: Dynamic Clock MUX in Titanium Ti60 F225 Example Design



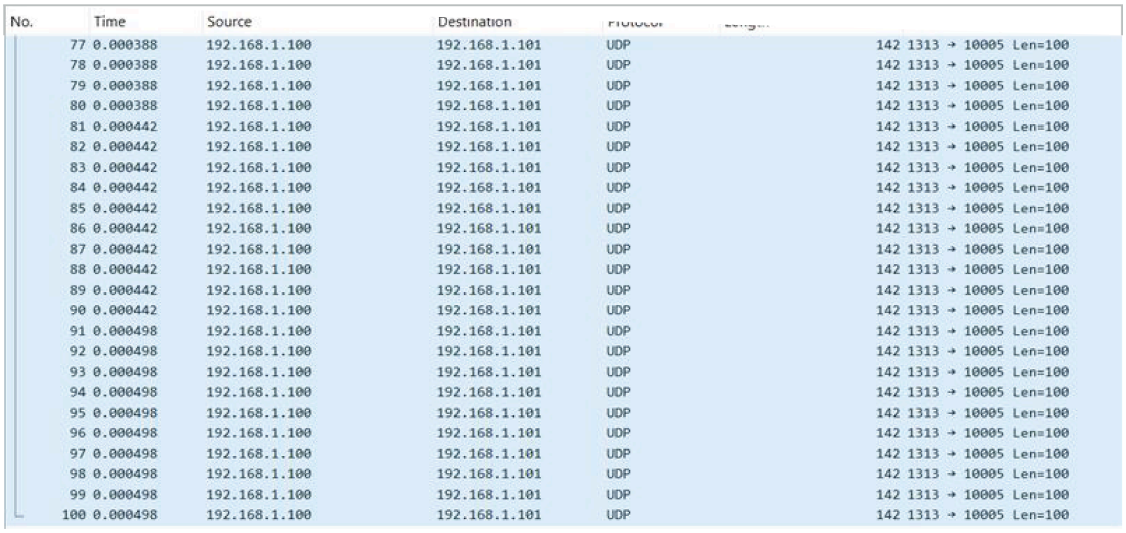
Using Wireshark

Wireshark is a free, open-source network packet analyzer used to capture and inspect Ethernet traffic. Efinix recommends using Wireshark to verify packet consistency between the transmitted and received packets, especially when the FPGA design is connected to a PC or other external network partner.

Follow the following steps to use Wireshark.

1. Download and install Wireshark.
   - Visit www.wireshark.org to download and install the latest version of Wireshark.
2. Launch Wireshark.
   - Open the Wireshark software on your computer. Under the **Capture** category, you will see a list of available network interfaces.
3. Select the Ethernet interface.
   - Choose the active Ethernet interface connection to the FPGA board. Wireshark begins capturing packets that are being transmitted and received through this interface.

Figure 46: Wireshark Sample Output (Normal Mode, UDP Pattern)



No.	Time	Source	Destination	Protocol	Length	
77	0.000388	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
78	0.000388	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
79	0.000388	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
80	0.000388	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
81	0.000442	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
82	0.000442	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
83	0.000442	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
84	0.000442	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
85	0.000442	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
86	0.000442	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
87	0.000442	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
88	0.000442	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
89	0.000442	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
90	0.000442	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
91	0.000498	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
92	0.000498	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
93	0.000498	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
94	0.000498	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
95	0.000498	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
96	0.000498	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
97	0.000498	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
98	0.000498	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
99	0.000498	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100
100	0.000498	192.168.1.100	192.168.1.101	UDP		142 1313 → 10005 Len=100

Example Design Configuration Registers

Default value for all DWORD offsets are

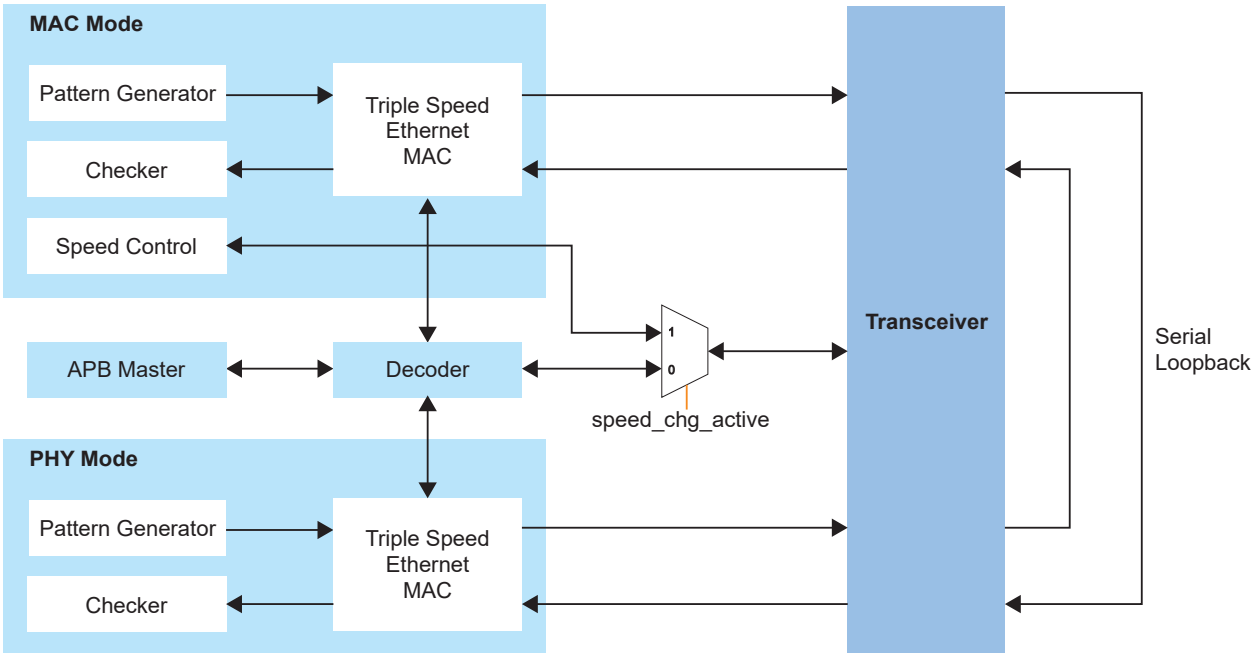
Table 23: Configuration Registers for Example Design with External PHY

Dword Offset	Name	R/W	Description
0x200	mac_sw_rst	R/W	Assert high to trigger the *proto_reset* signal. The register output is tied to the *proto_reset* signal.
0x204	Bit 0 - axi4_st_mux_select	R/W	0: Transmit AXI4-Stream TX packet from pattern generator. 1: Loopback AXI4-Stream RX path to AXI4-Stream TX path.
	Bit 1 - pat_mux_select	R/W	0: Select UDP pattern generator. 1: Select the MAC pattern generator.
0x208	Bit 0 - udp_pat_gen_en	R/W	Assert high then low to trigger a UDP packet.
	Bit 1 - mac_pat_gen_en	R/W	Assert high then low to trigger MAC packet.
0x20c	Bit [15:0] - pat_gen_num	R/W	The number of transmit packet to send. Zero indicates unlimited packets.
	Bit [31:16] - pat_gen_ipg	R/W	The UDP/MAC interpacket gap.
0x210	pat_dst_mac [31:0]	R/W	The UDP/MAC pattern's destination MAC address.
0x214	pat_dst_mac [47:32]	R/W	
0x218	pat_src_mac [31:0]	R/W	The UDP/MAC pattern's source MAC address
0x21c	pat_src_mac [47:32]	R/W	
0x220	pat_mac_dlen [15:0]	R/W	The length of a MAC pattern packet. The complete MAC pattern packet is *pat_mac_dlen* + 14 bytes (excluding CRC).
0x224	pat_src_ip [31:0]	R/W	The UDP pattern's source IP address.
0x228	pat_dst_ip [31:0]	R/W	The UDP pattern's destination IP address.
0x22c	Bit [15:0] pat_src_port	R/W	The UDP pattern's source port.
	Bit [31:16] pat_dst_port		The UDP pattern's destination port.
0x230	pat_udp_dlen [15:0]	R/W	The length of a UDP pattern packet. The complete UDP pattern packet is *pat_udp_dlen* + 45 bytes (excluding CRC).

Example Design with FPGA Transceiver

Efinix provides the flollowing example design. The example design targets Titanium Ti375 N1156 Development Kit.

Figure 47: Triple Speed Ethernet MAC Core's Transceiver Example Design



The example design illustrates the connectivity among the Triple Speed Ethernet MAC, FPGA 1G PCS, and the user's logic, as shown in **Figure 47: Triple Speed Ethernet MAC Core's Transceiver Example Design** on page 75. The user's logic consists of 2 pattern generators and 2 checkers (denoted by `efx_mac1gbe_exp_pat_gen` and `efx_mac1gbe_exp_checker` modules respectively), APB modules (denoted by `efx_mac1gbe_exp_apb_master` and `efx_mac1gbe_exp_apb_decoder`), and a virtual I/O debugger.

The APB decoder decodes the APB address and routes the APB request to the corresponding APB slaves based on the address map in **Table 24: Address Map for the Transceiver Example Design** on page 76. The APB master contains a built-in RAM that captures the `APB_PRDATA` and the `APB_PADDRESS` of all APB Read commands. Ports `ram_dout_d` and `ram_dout_a` display the captured `APB_PRDATA` and the corresponding `APB_PADDRESS`.

Table 24: Address Map for the Transceiver Example Design

Start Address	End Address	Registers
PCS		
0xC0_0400	0xC0_043F	PCS (Lane 0) registers
0xC0_04C0	0xC0_04FF	PCS (Lane 3) registers
MAC		
0xE0_0000	0xE0_01FF	MAC (Lane 0) Configuration Registers
0xE0_3000	0xE0_31FF	MAC (Lane 3) Configuration Registers

Upon the assertion of `IN_USER`, the example design awaits the PLL to lock. The assertion of `PLL_LOCK` releases the resets in the Triple Speed Ethernet MAC core and the FPGA PCS. This is followed by the PHY power-up sequence handshake, resulting in the assertion of `PHY_INIT_DONE`. Upon the assertion of `PHY_INIT_DONE`, the APB modules become operational and are used to write the corresponding `signal_ok` register bit to `1`, enabling the RX path, as depicted in the **Titanium SGMII 1G and 2.5G User Guide**.

After achieving synchronization status, the pattern generator starts transmitting data frames into the TX AXI ST interface of the Triple Speed Ethernet MAC core. Throughout the operation, the checker actively checks the RX path of the Triple Speed Ethernet MAC core for error frame and assertion of `RX_AXI_TUSER`. Test passes when `ln_data_cnt` and `ln_frame_cnt` are incremented with `ln_err_cnt` and `ln_bad_frame_cnt` remaining `0`.

Available Macros in the Transceiver Example Design

The example design includes useful macros that allow you to further customize.

- `AN_L3_PHY_MODE`
  - Configure **Lane 3** as **PHY** mode and **Lane 0** as **MAC** mode for auto negotiation.
  - Pre-requisite: Enable **SGMII Auto Negotiation (AN)** in the Interface Designer with speed of 10/100/1000 Mbps.
- `AN_L0_PHY_MODE`
  - Configure **Lane 0** as **PHY** mode and **Lane 3** as **MAC** mode for auto negotiation.
  - Pre-requisite: Enable **SGMII Auto Negotiation (AN)** in the Interface Designer with speed of 10/100/1000 Mbps.
- `L3_1G/ L3_100M/ L3_10M`
  - Sets **Lane 3 SGMII_MODE** to **1G/ 100M/ 10M**, respectively.
  - If none of these macros are defined, **Lane 3** is set to **2.5G** by default.
- `L0_1G/ L0_100M/ L0_10M`
  - Sets **Lane 0 SGMII_MODE** to **1G/ 100M/ 10M**, respectively.
  - If none of these macros are defined, **Lane 0** is set to **2.5G** bydefault.
- `JUMBO`
  - Configure the pattern generator to transmit alternate normal data frames and jumbo frames.
  - Pre-requisite: Change the **Maximum Transmission Unit Frame Length** to **16018** in Triple Speed Ethernet MAC IP Configuration.

By default, the following macros are enabled:

- `AN_L3_PHY_MODE`
- `L3_100M`
- `L0_1G`

Enabling the Macros in the Transceiver Example Design

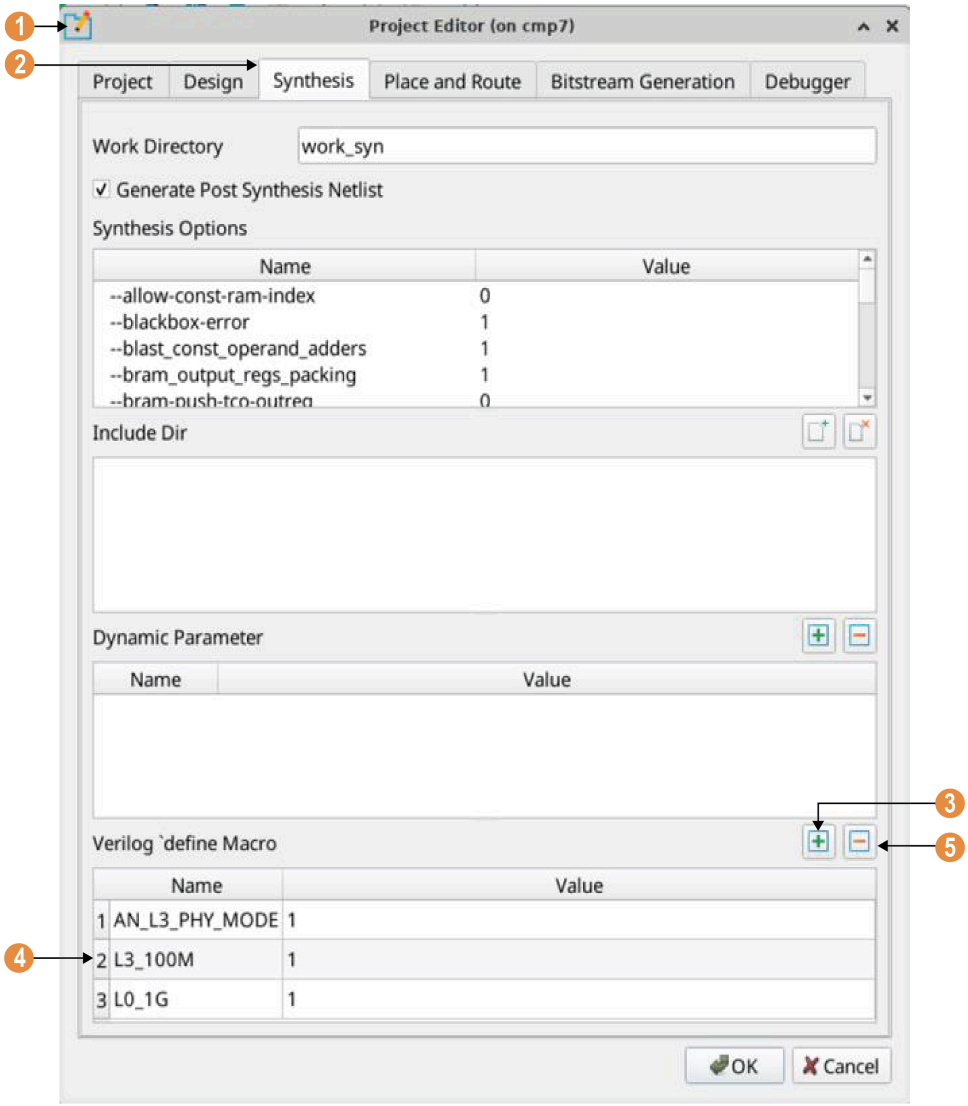
To enable macros for Efinity compilation.

1. In the active example design, go to **File > Edit Project**. The **Project Editor** opens.
2. Go to **Synthesis** tab.
3. At the **Verilog define Macro** section, click **+** to add new macros.
4. Type in the name of the macro and set **Value** to **1** .

> **Note:** When enabling the macros, always set the **Value** to **1**. Setting the macro to a different value does not disable the macro.

5. To disable a macro, you have to remove it from **Verilog define Macro** section. To remove a macro, select the desired macro and click **-**.

Figure 48: Enabling Macro in the Example Design



Generating the Example Design

Use the following steps to generate the example design from the IP Catalog.

1. Create a new project.
2. Select the Triple Speed Ethernet MAC core from the IP Catalog and set the following configurations.
   - Turn on **Use MAC with Transceiver** option.
   - **Maximum Transmission Unit Frame Length:**: 16018
3. In the **Deliverables** tab, turn on the **Example Design** (**Ti375N1156_devkit**).
4. Click **Generate**.

The example design is available in **<project path>\ip\<ip-module-name>\Ti375N1156_devkit** directory.

To run the example design in **Auto Negotiation** mode (default):

1. Open the example design at **<project path>\ip\<ip-module-name>\efx_ethernet_1g_exp.xml**
2. Compile the design.
3. Download the design to the Titanium Ti375 N1156 Development Kit.
4. Based on the macros enabled, read the auto negotiation advertised abilities register for the lane in PHY mode.
5. Then, override the following bits:
   - Bit 12 = 1'b1 (Full duplex)
   - Bit [11:10] = 2'b01 (100 Mbps)
   - Bit 15 = 1 (Link up)
6. Upon assertion of `PHY_INIT_DONE`, write the corresponding `signal_ok` register bit to `1` to enable the RX path.
7. The test passes when:
   - `ln_data_cnt` and `ln_frame_cnt` increments.
   - `ln_err_cnt` and `ln_bad_frame_cnt` remained `0`.

To run the example design in **2.5G Jumbo** mode:

1. Open the example design at **<project path>\ip\<ip-module-name>\efx_ethernet_1g_exp.xml**
2. Change the **Speed** option to **2.5Gbps** for both **Lane 0** and **Lane 3** in the Interface Designer.
3. Press the **Generate Interface Output files**.
4. Only enable the JUMBO macro in Project Editor.
5. Compile the design.
6. Download the design to the Titanium Ti375 N1156 Development Kit.
7. Upon assertion of `PHY_INIT_DONE`, write the corresponding `signal_ok` register bit to `1` to enable the RX path.
8. The test passes when:
   - `ln_data_cnt` and `ln_frame_cnt` increments.
   - `ln_err_cnt` and `ln_bad_frame_cnt` remained `0`.

Table 25: Transceiver Example Design Project Files

File Name	Description
efx_ethernet_1g_exp.sv	Example design of top-level wrapper.
<user_given_ip_name>.sv [5]	The generated Triple Speed Ethernet MAC core file is based on the user configuration in Efinity IP Manager.
Timing_Ti375.sdc	Constraint file for example design.
efx_mac1gbe_exp_apb_master.sv[5]	APB master controller module.
efx_mac1gbe_exp_apb_decoder.v [5]	APB address decoder module.
efx_mac1gbe_exp_pat_gen.sv	Pattern generator module.
efx_mac1gbe_exp_checker.sv	Checker module to validate data received.
Efx_mac1gbe_exp_speed_ctrl.sv [5]	Auto negotiation sequence module.
debug_top.v	Verilog file for EFX debug module.
debug_profile.json	Virtual I/O Debugger core file. Load this file in the Efinity Virtual I/O Debugger to customize the example design. See **Virtual I/O Debugger Settings** on page 81.

[5] This module is encrypted.

Virtual I/O Debugger Settings

This example design has a Virtual I/O Debugger interface to probe and control the signals of the example design. To control the settings, refer to the Virtual I/O Debugger settings.

Note: In the following table, the signal names are prefixed with `l0_` and `l3_`, to indicate and distingush these instances. For a detailed signal description, refer to Titanium SGMII 1G and 2.5G User Guide and Table 14: Triple Speed Ethernet MAC Core Port Interface on page 50.

Table 26: Efinity Virtual I/O Debugger Settings

Signal Name	Width	Probe / Source	Description
General Signals			
pll_locked	1	Probe	Lock signal from PLL, indicating that the clock sources (`INIT_CLK` and `APB_CLK`) are stable. This signal kicks start the operation of the example design, where the assertion of `PLL_LOCK` releases the resets in the Triple Speed Ethernet MAC core and the FPGA PCS.
PCS Interface			
Instance Lane 0			Refer to the signal descriptions in `Signals per Quad` table in the *Signals* chapter of Titanium SGMII 1G and 2.5G User Guide.
l0_sync_status	1	Probe	
l0_sgmii_speed	2	Probe	
Instance Lane 3			
l3_sync_status	1	Probe	
l3_sgmii_speed	2	Probe	
Triple Speed Ethernet MAC Interface			
pma_cmn_ready	1	Probe	Refer to the signal descriptions in Table 14: Triple Speed Ethernet MAC Core Port Interface on page 50.
Instance Lane 0			
l0_init_done	1	Probe	
l0_rx_signal_detect	1	Probe	
Instance Lane 3			
l3_init_done	1	Probe	
l3_rx_signal_detect	1	Probe	
APB Interface and Status			
usr_apb_start	1	Source	This signal enables you to manually trigger APB requests. Efinix recommends that you control this signal by selecting **Active-High** from the drop-down list in the control column. This signal operates based on its rising edge, i.e., the assertion of this signal triggers 1 APB request. Prior assertion of this signal, APB signals (`usr_apb_write`, `usr_apb_addr`, and `usr_apb_pwdata`) need to be assigned and stable. To trigger another APB request, this signal needs to return to `0` first, then reassert to trigger the subsequent APB request.

Signal Name	Width	Probe / Source	Description
usr_apb_write	1	Source	APB Write.
usr_apb_addr	24	Source	APB Address.
usr_apb_pwdata	32	Source	APB Write Data.
Auto Negotiation Interface			
Instance Lane 0			The assertion of this signal indicates that the auto negotiation sequence is ongoing and no APB access is allowed.
l0_speed_chg_active	1	Probe	
l3_speed_chg_active	1	Probe	
Checker			
l0_data_cnt	8	Probe	*Data counter*—Increments on every cycle when valid data is received on the MAC RX AXI ST Interface
l3_data_cnt	8	Probe	
l0_err_cnt	8	Probe	*Error counter*—Increments on every cycle when erroneous data is received on the MAC RX AXI ST Interface
l3_err_cnt	8	Probe	
l0_frame_cnt	8	Probe	*Frame counter*—Increments for every frame received on the MAC RX AXI ST Interface
l3_frame_cnt	8	Probe	
l0_bad_frame_cnt	8	Probe	*Bad frame counter*—Increments for every bad frame received on the MAC RX AXI ST Interface
l3_bad_frame_cnt	8	Probe	

Triple Speed Ethernet MAC Testbench

You can choose to generate the testbench when generating the Triple Speed Ethernet MAC core in the IP Manager Configuration window.

Note: You must include all **.v** and **.sv** files generated in the **/testbench** directory in your simulation.

Important: Efinix tested the testbench generated with the default parameter options only.

Efinix provides a ready-to-use Modelsim simulation script to help you quickly run the testbench. To execute the testbench, run the following command in a terminal application: `vsim -c -do modelsim.do` in a terminal application. You must have Modelsim installed on your computer to use this script.

Table 27: Testbench Files

Directory	Description
../Testbench	Contains the example design and testbench files.
../Testbench/aldec	Contains the encrypted simulation source code for Aldec.
../Testbench/modelsim	Contains the encrypted simulation source code for ModelSim.
../Testbench/ncsim	Contains the encrypted simulation source code for NCSim.
../Testbench/synopsys	Contains the encrypted simulation source code for Synopsys VCS.

The simulation testbench is set up in a loopback configuration, where the RGMII TX interface is directly connected to the RGMII RX interface. The test performs a *Transmit MAC Packet Test*, which verifies data integrity through transmit-receive loopback checking. **Packet length** is set to **100**, **Packet number** is set to **1000**, and **Packet type** is set to **MAC pattern**. You can configure the MAC speed, pattern generator type, and packet length in the testbench.

During simulation, the pattern generator sends packets and the checker verifies each received packet against the transmitted one.

If a packet matches correctly, the terminal prints a message similar to the following:

Each line indicates the simulation timestamp and the packet sequence number successfully received.

```
---- Configure TSE MAC IP register setting ---
9610000 - Correct MAC packet  0, received
18330000 - Correct MAC packet 1, received
27050000 - Correct MAC packet 2, received
35770000 - Correct MAC packet 3, received
44490000 - Correct MAC packet 4, received
53210000 - Correct MAC packet 5, received
61930000 - Correct MAC packet 6, received
70650000 - Correct MAC packet 7, received
79370000 - Correct MAC packet 8, received
88090000 - Correct MAC packet 9, received
96810000 - Correct MAC packet 10, received
```

Note: The testbench supports simulation for Triple Speed Ethernet MAC core only.

Note: Contact Efinix support at the **Support Center** to obtain the testbench for the Triple Speed Ethernet MAC core with FPGA PCS.

Differences with Ethernet 10G MAC Core

When `crc_fwd = 1`:

- Triple Speed Ethernet MAC core forward CRC octets on `rx_axis_mac_tdata`. The signals `rx_axis_mac_tvalid` remain asserted until the last CRC octet, with `rx_axis_mac_tlast` asserted at the last CRC octet.
- Ethernet 10G MAC core, in contrast, ends `TVALID` and `TLAST` at the final data octet. The CRC octets are driven on `TDATA` immediately after the last data octet.

Revision History

Table 28: Revision History

Date	Document Version	IP Version	Description
December 2025	6.1	7.1	Removed note in Clock sources and Terminating Bad Tx Frames. (DOC-2850) Added note in RGMII RX of RX PHY Interface. Updated sub-topics in Terminating Bad TX Frames and added new sub-topic Non-Streaming TX Data. Updated all informations in the sub-topics and added new sub-topic Non-Streaming RX Data Frame (SGMII) under Erroneous RX Frame. Updated Send Pause Frame at TX of Priority Flow Comtrol (Duplex Mode). Updated information in aTxIncontinuityFramesErrors. Added note in Latency and Example Design with External PHY.
December 2025	6.0	7.0	New document release replacing the current document. (DOC-2710)
November 2024	5.2	6.3	Updated IP Version in Revision History. (DOC-2185)
November 2024	5.1	6.4	Added Topaz in Device Support. (DOC-2176) Added IP Version in Revision History. (DOC-2185)
June 2024	5.0	–	Updated Realigning the PHY Clock section. (DOC-1888) Removed section Using the RGMII rgmii_rxc_edge and rgmii_txc_dly Signals. Updated waveforms for Output Signals Waveform with DDIO Enabled and RGMII Transmit Clock Delay Enabled (Center-aligned) and Output Signals Waveform with DDIO Enabled and RGMII Transmit Clock Delay Disabled (Edge-aligned). Added new waveforms for Input Signals Waveform with DDIO Enabled and First Sampling of RGMII Receive Data Using Falling Edge and Input Signals Waveform with DDIO Enabled and First Sampling of RGMII Receive Data Using Rising Edge. Added new data RGMII Transmit Clock Delay and First Sampling of RGMII Receive Data in table Triple Speed Ethernet MAC Core Parameters. Added a note for table Block Editor Settings for rgmii_rx_ctl. Added important note in Example Design and Testbench regarding using default parameters options only. (DOC-1781)
December 2023	4.9	–	Updated Titanium resource utilization. (DOC-1601)

Date	Document Version	IP Version	Description
June 2023	4.8	–	Added Device Support and release notes sections. (DOC-1234) Updated rgmii_txc settings for example design and to run other designs at 1000 Mbps.
March 2023	4.7	–	Added-in Normal Mode and Linked-Partner Mode sub-topic in Using Wireshank topic. (DOC-1176) Updated program in Using the PHY RGMII Clock-to-Data Delay (Trion Only).
February 2023	4.6	–	Added Ti60 F225 Development Board example design support. (DOC-1152)
February 2023	4.5	–	Added note about the resource and performance values in the resource and utilization table are for guidance only.
December 2022	4.4	–	Updated example design. (DOC-1015) Added New in version section.
October 2022	4.3	–	Updated ports width and descriptions and added new ports. Updated registers width and added new registers. Corrected AXI4-Stream Data Width parameter option. Corrected s_axi_wdata width. Updated example design directory and file names. Added Description about realigning PHY clock with top-module signals.
September 2022	4.2	–	Added support for GMII, RMII, and MII PHY interfaces. Updated core IP manager parameters.
April 2022	4.1	–	Added note about RX packet error feature is not supported and user can ignore warnings on rgmii_rx_ctl_HI signal. (DOC-787)
December 2021	4.0	–	Added example design in IP manager. Corrected PHY IC description. Updated example design RISC-V SoC to Sapphire. Updated supported FIFO depth to 16384.
November 2021	3.3	–	Corrected Block Editor Settings for rgmii_rx_ctl and rgmii_tx_ctl signals.
October 2021	3.2	–	Updated Block Editor Settings for rgmii_rx_ctl and rgmii_tx_ctl signals. (DOC-586)
October 2021	3.1	–	Added note to state that the f_{MAX} in Resource Utilization and Performance, and Example Design Implementation tables were based on default parameter settings.
June 2021	3.0	–	Added missing rgmii_txc_LO interface description in the ports table. Added parameters to set in Interface Designer DDIO settings. Corrected Output Signals Waveform with DDIO Enabled figure. Added note about including all **.v** generated in testbench folder is required for simulation.

Date	Document Version	IP Version	Description
December 2020	2.0	–	Updated user guide for Efinix® IP Manager which includes added IP Manager topics, updated parameters, and user guide structure.
October 2020	1.2	–	Corrected Bit 0 - axi4_st_mux_select of the example design configuration registers description. Updated normal and link test mode descriptions. Added aFramesReceivedLen[15:0] to statistic counter register. (DOC-318) Added information about adding delay to re-allign the PHY clock in the example design. (DOC-318) Added settings to enable the hard DDIO blocks in Interface Designer and waveforms after integrating the hard DDIO blocks into the core. (DOC-318)
August 2020	1.1	–	Updated the Using this Example with the RISC-V SDK topic.
July 2020	1.0	–	Initial release.