Ethernet 10G MAC Core User Guide

UG-CORE-ETHERNET-10G-MAC-v1.4
June 2025
www.elitestek.com

Contents

Introduction **4**

Features **5**

Device Support **5**

Resource Utilization **5**

Release Notes **6**

Functional Description **7**

- Block Diagram 7
- Clock Sources 8
- Reset Signals 9
- Power Up Handshake with FPGA Transceiver 10
- Ethernet Packets, Frames, and IPG 11
- User Interface AXI ST 64 12
  - TX AXI ST 64 12
  - RX AXI ST 64 14
- CRC Generation and Check 15
- Programmable Inter Packet Gap (IPG) 15
- XGMII Interface 16
  - XGMII TX 16
  - XGMII RX 17
- Data Streaming Mode for Transmission 19
  - Cut Through Mode 19
  - Store Forward Mode 20
- Automatic Padding for Short TX Frames 22
- Terminating Bad TX Frames 23
  - Non-Compliant TX_AXI_TLAST 23
  - Assertion of TX_AXI_TUSER 24
  - Non-Streaming TX Data 25
  - Dropped TX_AXI_TVALID Before End of Frame 26
  - TX FIFO Overflow 26
- Erroneous RX Frame 27
  - Undersized RX Frame 27
  - Oversized RX Frame 27
  - Mismatched Length RX Frame 27
  - Frame Check Sequence (FCS) Error 27
  - Non-Streaming RX Data Frame 28
  - Error Frame 28
- Address Filtering and Broadcast Filtering at RX 29
- Priority Flow Control (Duplex Mode) 32
  - Send Pause Frame at TX 32
  - Decoding Pause Frame at RX 33
- Link Fault Sequence 35
- Statistic Reporting 39
  - cnt_tx_frame_transmitted_good 40
  - cnt_tx_frame_pause_mac_ctrl 40
  - cnt_tx_frame_is_fe 40
  - cnt_tx_frame_error_txfifo_overflow 41
  - rpt_rx_frame_length 41
  - cnt_rx_frame_undersized 42
  - cnt_rx_frame_oversized 42
  - cnt_rx_frame_mismatched_length 43
  - cnt_rx_frame_error_fcs 43
  - cnt_rx_frame_filtered_by_address 44

cnt_rx_frame_pause_mac_ctrl..... 44
cnt_rx_frame_errors..... 45
cnt_rx_frame_received_good..... 47
cnt_rx_frame_received_total..... 47

Latency..... 48

Efinity: IP Catalog, Interface Designer, and Integration..... 48
IP Catalog..... 48
Interface Designer..... 48
Top Level Module..... 49

IP Manager..... 50

Customizing the Ethernet 10G MAC..... 51

Ports..... 53

Ethernet 10G MAC Example Design..... 56
Example Design Macros..... 58
Enabling the Macros in the Example Design..... 59
Generating the Example Design..... 60
Virtual I/O Debugger Settings..... 65

Ethernet 10G MAC Testbench..... 70

Acronyms and Abbreviations..... 71

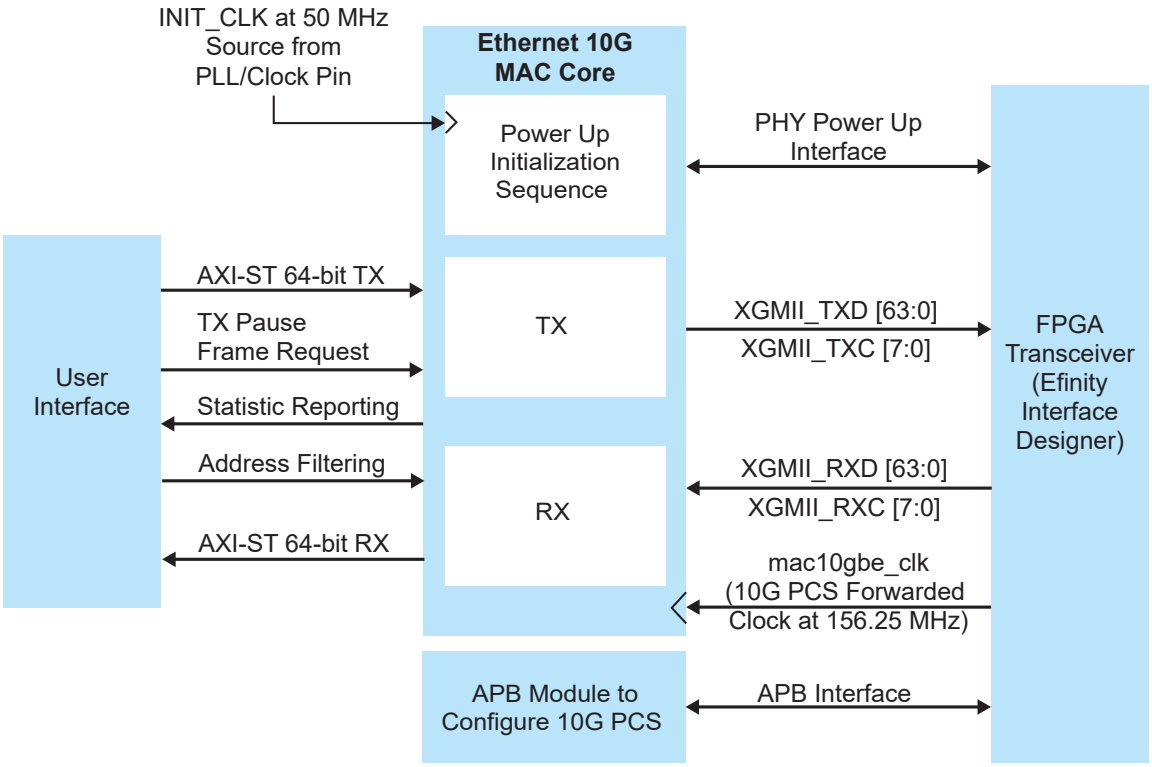
Revision History..... 72

Introduction

The Ethernet 10G MAC core is a configurable core that supports Ethernet speeds up to 10 Gbps in full duplex transfer mode. The core complies with the IEEE Std. 802.3-2008 specification and operates at a frequency of 156.25 MHz.

Additionally, the Ethernet 10G MAC core is designed to interact with the user's logic and the FPGA Ethernet 10G physical coding sublayer (PCS) as shown in **Figure 1: Ethernet 10G MAC Core Interaction with User's Logic and the FPGA Ethernet 10G PCS** on page 4. The interaction among the user's logic, Ethernet 10G MAC core, and the FPGA PCS are based on a per-lane basis.

Figure 1: Ethernet 10G MAC Core Interaction with User's Logic and the FPGA Ethernet 10G PCS



At the FPGA transceiver's end, the APB interface is shared per quad basis, which is a common interface shared across 4 lanes in the FPGA Ethernet 10G PCS. For more information, see TJ-Series Ethernet 10GBase-KR User Guide.

Use the IP Manager to select IP, customize it, and generate files. The Ethernet 10G 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 Elitestek® development board.

Features

- Complies with IEEE Std. 802.3-2008 specification
- Supports IEEE Std. 802.1Q VLAN tagged Ethernet frames
- Supports 10 Gbps Ethernet in full duplex transfer mode
- Single clock operates at 156.25 MHz
- AXI4 ST 64-bit user interface at transmit and receiver interfaces
- Configurable **Cut Through** mode to transfer the Ethernet frame with minimum latency
- Configurable **Store Forward** mode where the entire Ethernet frame is received first before the transfer begins
- Programmable inter packet gap (IPG)
- Automatic padding for short frames
- Frame check sequence and CRC generation, includes checking and forwarding
- Automatic termination of bad frames
- Frame length check based on user-defined MTU frame length
- Automatic frame length check against ETHERTYPE/LEN field for frames
- Broadcast, Unicast, and multicast address filtering
- Flow control through pause frame generation and decoding
- Statistics reporting

Device Support

Ethernet 10G MAC core is supported in TJ-Series and TP-Series FPGA with transceiver. Refer to the TJ-Series Selector Guide and TP-Series Selector Guide to get the latest device list that supports transceivers.

Resource Utilization

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

Table 1: TJ-Series Resource Utilization

FPGA	Mode	Logic Elements (Logic, Adders, Flipflops, etc.)	Memory Blocks	DSP Blocks	Efinity® Version
TJ375N1156X C4	Cut Through	7,190/362,880 (1.98 %)	0/2,688 (0.19 %)	0/1,344 (0 %)	2025.1
	Store Forward	9,479/362,880 (2.61%)	5/2,688 (0.30 %)	0/1,344 (0 %)	

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

(1) System Verilog 2005.

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 on the Efinity Downloads page under each Efinity software release version.

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

Functional Description

The Ethernet 10G MAC core does the following functions:

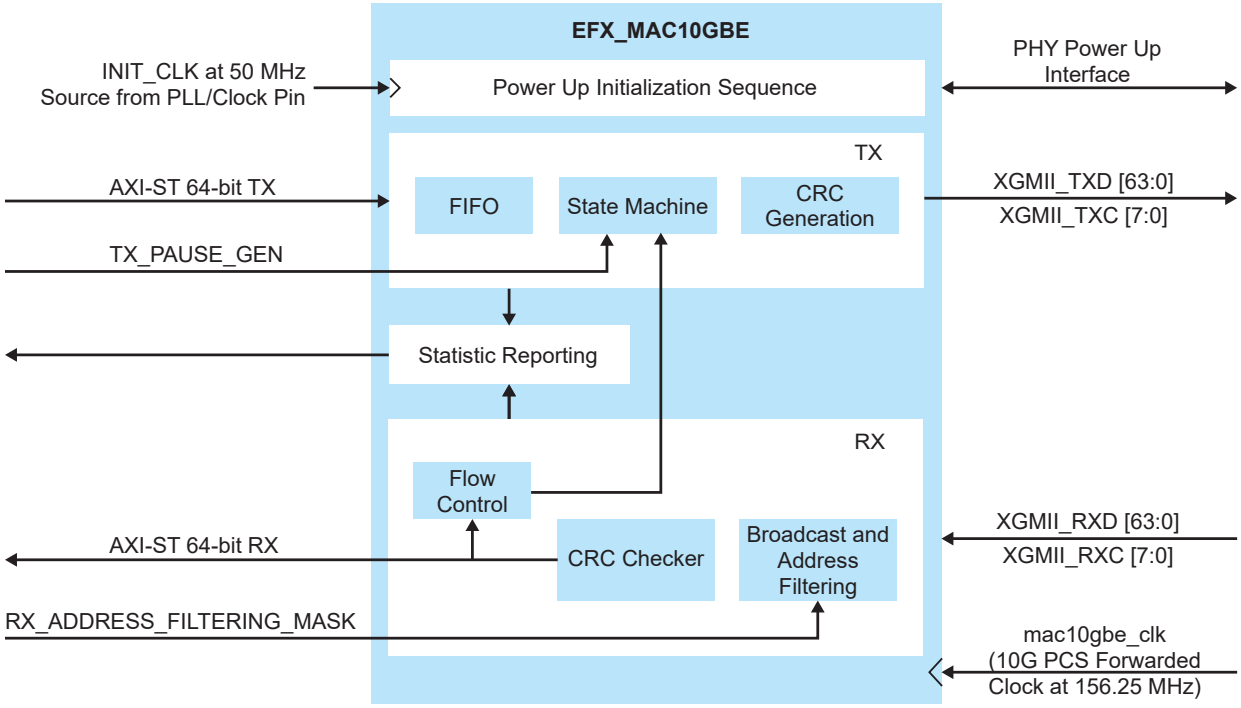
- Converts TX data packets from AXI ST to XGMII format.
- Decodes RX Data Packets from XGMII format to AXI ST.
- Generates the pause frame to be transmitted at the XGMII Interface.
- Identifies the pause frame and implements the flow control through the pause mechanism based on the decoded `PAUSE_QUANT`.
- Drops RX Frames with mismatched address through a bit-wise address filtering mask.
- Drops RX Frames with broadcast address.

Block Diagram

The Ethernet 10G MAC core or `efx_mac10gbe` comprises of the following modules:

- Power-up sequence to handshake with the FPGA transceiver.
- AXI ST 64-bit user interface.
- TX engine with CRC generation.
- RX engine with CRC checker.
- XGMII interface.
- Statistics reporting.
- Priority flow control module.
- Broadcast and address filtering.

Figure 2: Ethernet 10G MAC Core Sub-Modules



Clock Sources

The Ethernet 10G MAC core has 2 independent clock sources:

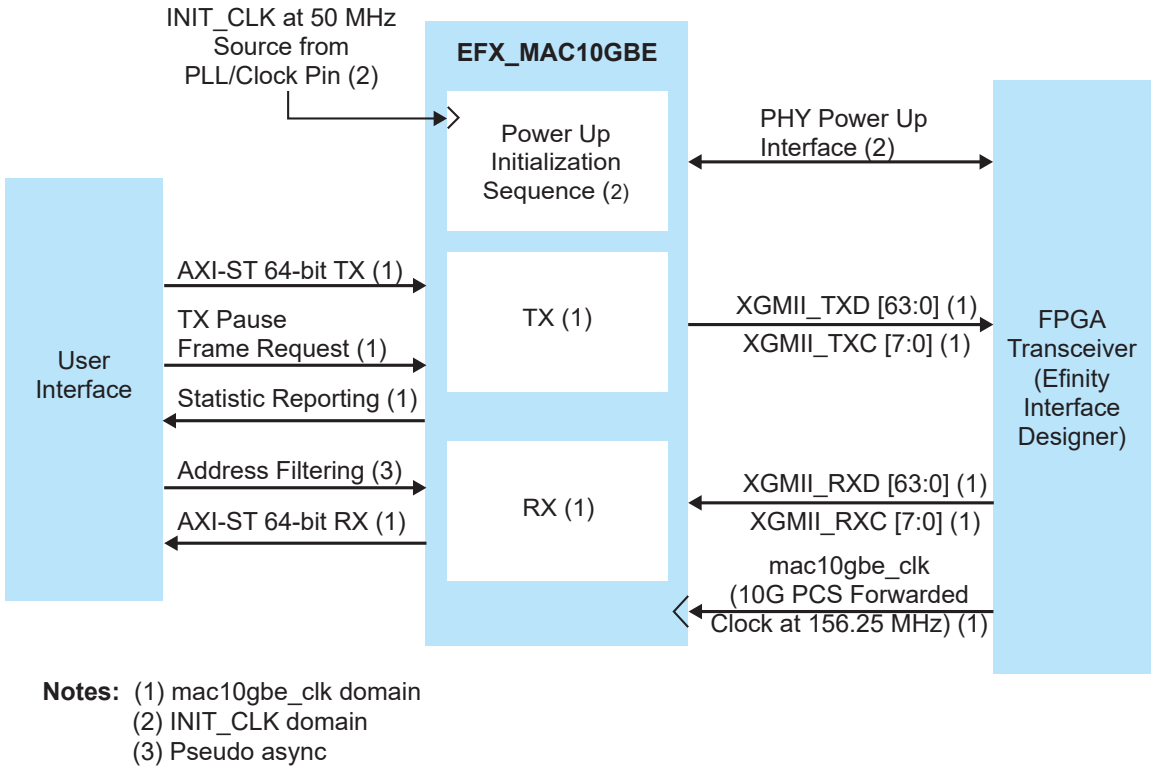
- `init_clk`
- `mac10gbe_clk`

Table 2: Ethernet 10G MAC Core Clock Sources

Clock Name	Frequency	Description
INIT_CLK	≤ 50 MHz	To facilitate the power-up sequence of the FPGA transceiver PHY. Can be sourced from the PLL or CLKIN pin.
MAC10GBE_CLK	156.25 MHz	Sourced from the FPGA transceiver forwarded clock to be the source synchronous. Eliminates additional clock PPM. The forwarded clock is used to clock the 10G PCS in the FPGA transceiver.

The **Figure 3: Ethernet 10G MAC Core Clock Domain** on page 8 shows the clock domains in the Ethernet 10G MAC core.

Figure 3: Ethernet 10G MAC Core Clock Domain



Notes: (1) mac10gbe_clk domain
(2) INIT_CLK domain
(3) Pseudo async

Reset Signals

The Ethernet 10G MAC core has 3 reset signals. These signals are active low and asynchronous.

- `init_rst_n`
- `mac_reset_n`
- `cnt_rst_n`

During the FPGA power-up, all three reset signals need to be initialized to 0 to reset the entire Ethernet 10G MAC core to initial known states.

The `init_clk` signal needs to be sourced and free running. When the FPGA is in user mode, you need to deassert `init_rst_n` to kick-start the power-up sequence with the FPGA PHY. The `init_rst_n` signal is to remain de-asserted throughout the operation. Details of the power-up sequence is being described in **Power Up Handshake with FPGA Transceiver** on page 10.

When `mac_reset_n` is asserted, it resets the core logic of the Ethernet 10G MAC core, such as TX and RX, and statistic reporting. The power-up sequence module remains intact and operational.

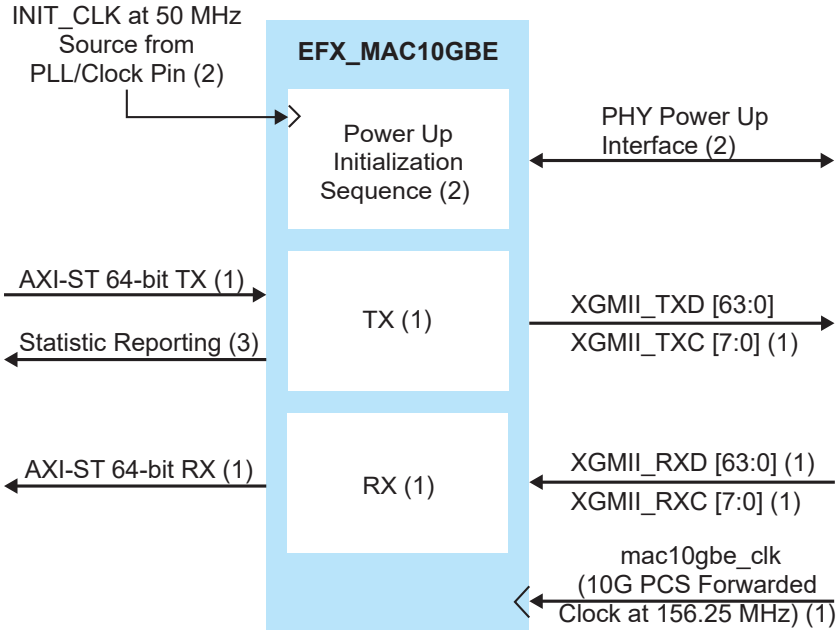
When `cnt_rst_n` is asserted, it resets all the statistic reporting in TX and RX. Other than the statistic counters, the logic of the Ethernet 10G MAC core is still operational. The power-up sequence module remains intact.

These 3 reset signals are asynchronous, but the deassertion of each reset signals is synchronous. For `init_rst_n`, there is an internal reset synchronizer to align the deasertion of `init_rst_n` to `init_clk`. For `mac_reset_n` and `cnt_rst_n`, there is a respective internal reset synchronizer to synchronize the deasertion of `mac_reset_n` and `cnt_rst_n` to `mac10gbe_clk`.

Each reset synchronizer has a latency of 2 to 3 clock cycles.

The **Figure 4: Ethernet 10G MAC Core Reset Domains** on page 9 shows the reset domains in the Ethernet 10G MAC core.

Figure 4: Ethernet 10G MAC Core Reset Domains



Notes: (1) MAC_RESET_N domain
(2) INIT_RST_N domain
(3) CNT_RST_N domain

Power Up Handshake with FPGA Transceiver

Before starting the operation of the Ethernet 10G MAC core, it needs to perform the power-up handshake with the FPGA transceiver.

The Ethernet 10G 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 TJ-Series Ethernet 10GBase-KR User Guide. **Figure 5: Power Up Handshake with the FPGA Transceiver PHY and PCS** on page 10 shows the power-up handshake between the Ethernet 10G MAC core and the FPGA transceiver.

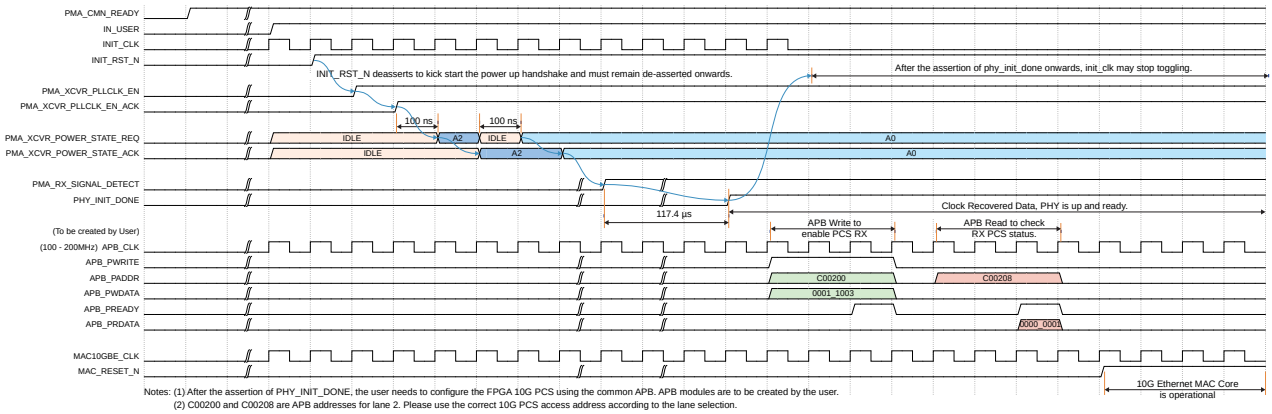
The power-up sequence between the Ethernet 10G MAC core and the FPGA transceiver PHY is clocked by `init_clk`. The assertion of `phy_init_done` indicates that the FPGA transceiver PHY is CDR-lock-to-data.

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 with a Quad.

Note: You need to create APB modules to configure the PCS. Refer to table 8, 10, and 21 in the Register Map chapter of the TJ-Series Ethernet 10GBase-KR User Guide.

Figure 5: Power Up Handshake with the FPGA Transceiver PHY and PCS on page 10 includes an illustration of APB Write to enable PCS RX in lane 2.

Figure 5: Power Up Handshake with the FPGA Transceiver PHY and PCS



Important: When the FPGA PHY is in reset, all 3 reset signals must also be asserted. 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 6: 802.3 Ethernet Packet and Frame Structure for Non-VLAN Tagged on page 11 illustrates the normal Ethernet packet and frame structure for non-VLAN tagged frame.

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

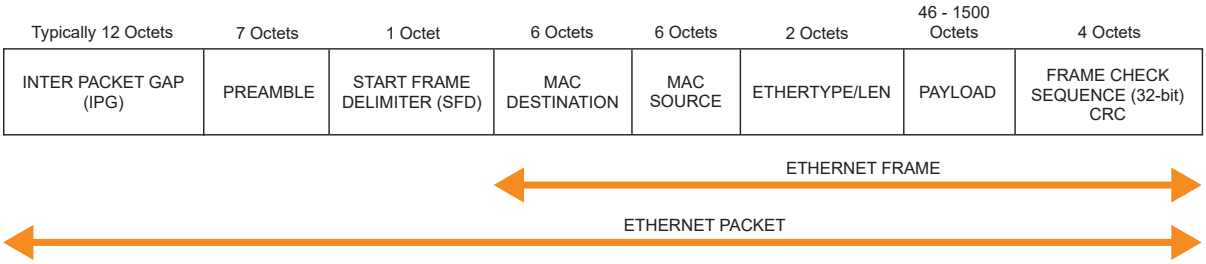
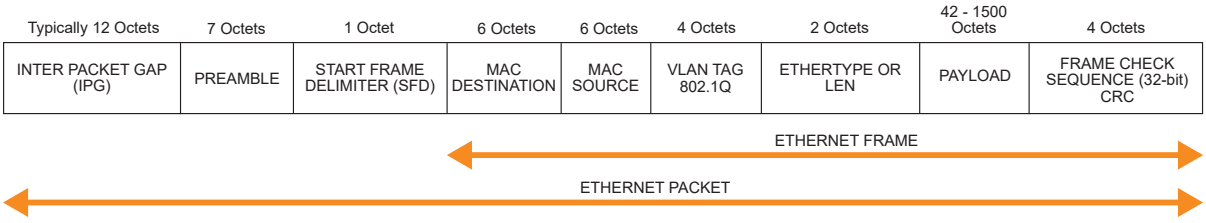


Figure 7: 802.3 Ethernet Packet and Frame Structure for VLAN Tagged on page 11 illustrates the Ethernet packet and frame structure for VLAN tagged frame.

Figure 7: 802.3 Ethernet Packet and Frame Structure for VLAN Tagged



Note: In the 2025.1 release and later, both non-VLAN tagged and VLAN tagged Ethernet frames are supported.

User Interface AXI ST 64

AXI ST 64 is a user interface comprising TX channel and RX channel.

TX AXI ST 64

At TX AXI ST interface, transmission starts when TX_AXI_TREADY = 1. The assertion of TX_AXI_TREADY indicates that Ethernet 10G MAC core is ready to accept the TX data streaming.

To transmit, assert TX_AXI_TVALID, along with TX_AXI_TDATA, TX_AXI_TKEEP, TX_AXI_TLAST, TX_AXI_TUSER. TX_AXI_TDATA contains 64-bits data streaming to be transmitted to XGMII_TXD. TX_AXI_TLAST indicates the last byte data in the data streaming. TX_AXI_TKEEP indicates the positional validity of the byte data within the data streaming. The assertion of TX_AXI_TUSER indicates that there is an error in the data streaming.

For Ethernet application, TX_AXI_TDATA must contain streaming data, i.e., data needs to be streaming and continuous, where TX_AXI_TKEEP = FF except at TX_AXI_TLAST. Streaming data, at TX_AXI_TLAST, means TX_AXI_TKEEP = {01, 03, 07, 0F, 1F, 3F, 7F, FF}. Refer to **Figure 8: Streaming Data at TX AXI ST Interface** on page 12.

In the event of a non-data streaming, it is interpreted as a bad frame and is dropped. The same goes with TX_AXI_TUSER, i.e., the asserted TX_AXI_TUSER results in a bad frame. Refer to **Terminating Bad TX Frames** on page 23.

When TX_AXI_TREADY = 0, the Ethernet 10G MAC core is not ready to accept any TX data. Hence, you need to hold or retain all the current TX_AXI signals, which are TX_AXI_TDATA, TX_AXI_TLAST, TX_AXI_TVALID, TX_AXI_TKEEP, and TX_AXI_TUSER, until TX_AXI_TREADY recovers to 1. Refer to **Figure 9: User to Retain TX_AXI Signals Until TX_AXI_TREADY Recovers** on page 13.

Figure 8: Streaming Data at TX AXI ST Interface

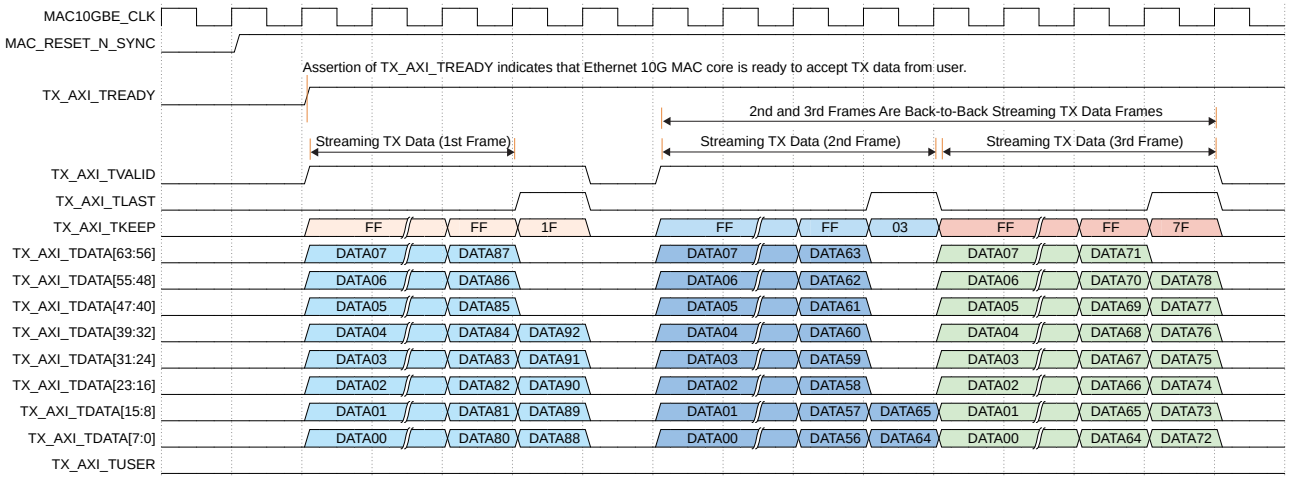
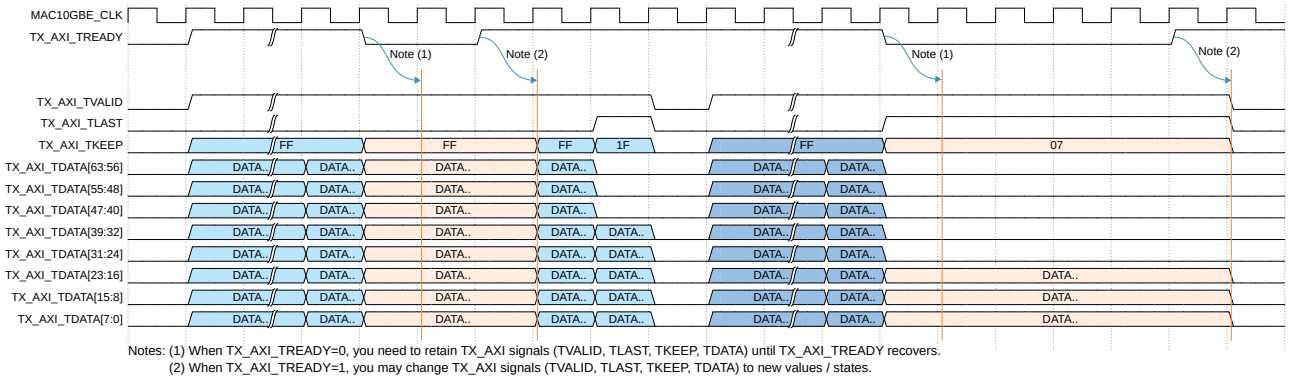
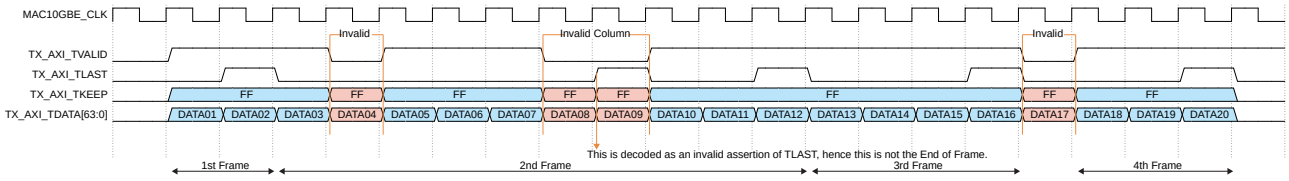


Figure 9: User to Retain TX_AXI Signals Until TX_AXI_TREADY Recovers



When `TX_AXI_TREADY` = `1`, the Ethernet 10G MAC core effectively decodes when `TX_AXI_TVALID` is asserted. When `TX_AXI_TVALID` = `0`, the entire column of `TX_AXI` signals (TDATA, TKEEP, TLAST, TUSER) is deemed as invalid. Refer to **Figure 10: Invalid TX_AXI Signals Column When TX_AXI_TVALID = 0** on page 13.

Figure 10: Invalid TX_AXI Signals Column When `TX_AXI_TVALID` = `0`



Important: User Interface AXI ST 64 operates in `mac10gbe_clk` domain. You need to ensure that the `TX_AXI_TVALID`, `TX_AXI_TDATA`, `TX_AXI_TKEEP`, `TX_AXI_TLAST`, and `TX_AXI_TUSER` signals are registered and synchronous to `mac10gbe_clk` prior to entering the Ethernet 10G MAC core.

RX AXI ST 64

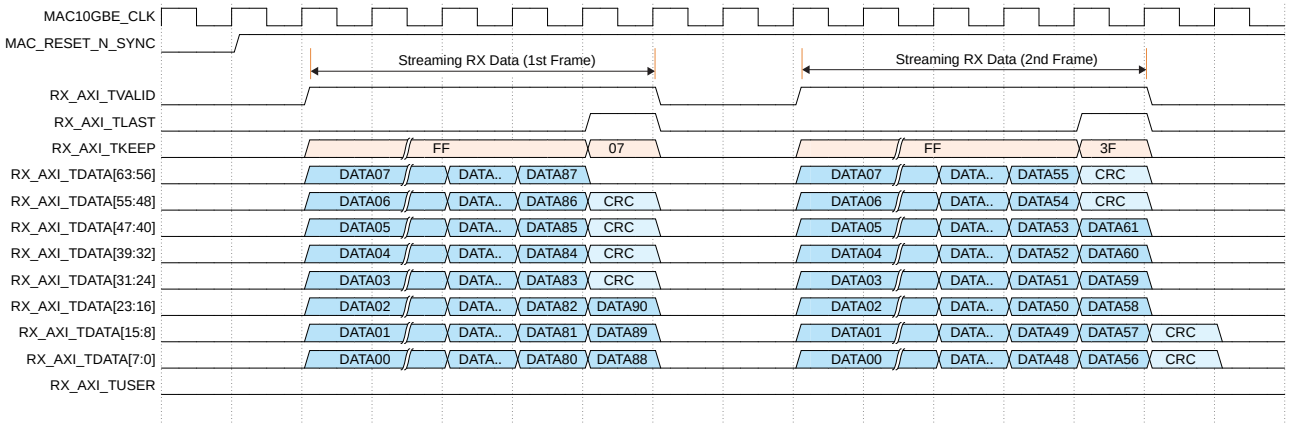
At RX AXI ST interface, the port RX_AXI_TREADY is excluded in the Ethernet 10G MAC core. The RX_AXI_TVALID indicates the validity of the RX data frame. RX_AXI_TDATA contains 64-bits data decoded from XGMII_RXD. RX_AXI_TLAST indicates the last byte of data in the RX data frame. RX_AXI_TKEEP indicates the positional validity of the byte data within the RX data frame. The assertion of RX_AXI_TUSER indicates that there is an error in the RX data frame. Refer to **Erroneous RX Frame** on page 27.

At RX channel, the incoming RX Ethernet packets are decoded based on the PREAMBLE, IDLE, and TERMINATE octets without any back pressure.

For Ethernet application, RX data are streaming and continuous, i.e., RX_AXI_TKEEP = FF except at RX_AXI_TLAST. Streaming data at RX_AXI_TLAST, means RX_AXI_TKEEP = {01, 03, 07, 0F, 1F, 3F, 7F, FF}. Any non-streaming of RX data frame results in RX_AXI_TUSER=1. For more details on the non-streaming RX data frame, refer to **Non-Streaming RX Data Frame** on page 28.

At RX AXI ST interface, port RX_AXI_TDATA contains RX data frame and CRC forwarding. Refer to **Figure 11: RX Data Frame at AXI ST Interface** on page 14. For more details on CRC Forwarding, refer to **CRC Generation and Check** on page 15.

Figure 11: RX Data Frame at AXI ST Interface



Notes: RX_AXI_TDATA includes forwarded CRC octets.

CRC Generation and Check

At the TX channel, the Ethernet 10G MAC core automatically generates CRC octets, appends them to the end of Frame, and transmits to XGMII TX interface.

At the RX channel, Ethernet 10G MAC core decodes the Ethernet frames and checks against 4 CRC octets that are trailing at the Ethernet frame. At the RX AXI ST interface, the CRC octets are forwarded to the `RX_AXI_TDATA` ports. The CRC forwarding is illustrated in **Figure 11: RX Data Frame at AXI ST Interface** on page 14.

The CRC polynomial that is used in Ethernet 10G 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 Ethernet 10G MAC core features a programmable IPG where you can configure any value between 9 octets to 15 octets. Based on your settings on the programmable IPG, the minimum gaps are inserted between the frames. The IPG implementation is illustrated in **Figure 12: Conversion from TX AXI ST to XGMII TX** on page 16.

Note: For the 2025.1 release and later, the `PREAMBLE` may start at Octet0 or Octet4 based on the user-defined Programmable Inter Packet Gap.

XGMII Interface

XGMII interface connects with the Ethernet 10G PCS in the FPGA transceiver.

XGMII TX

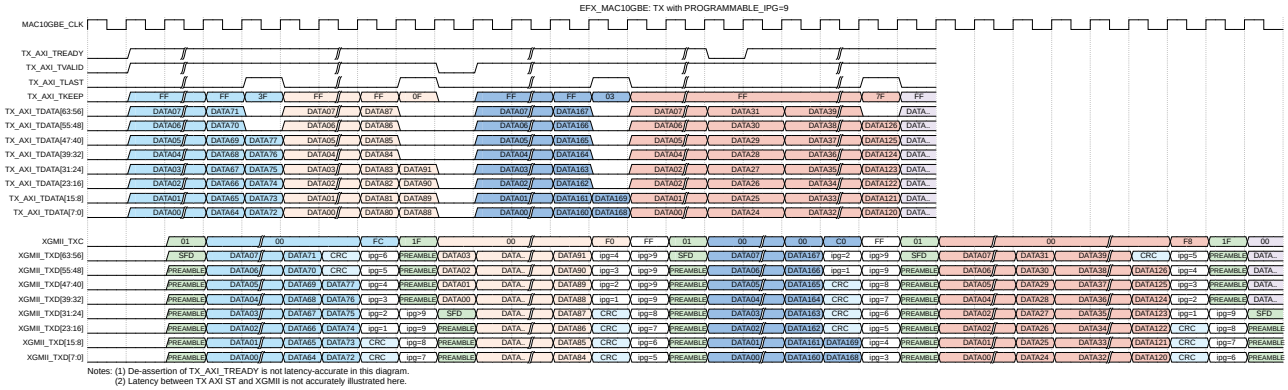
At XGMII TX, the Ethernet 10G MAC core outputs the TX Ethernet frames based on the following priority in descending order:

1. Faulty link condition
2. Pause mechanism from RX Pause frame
3. Request to send TX Pause frame
4. `TX_AXI_TVALID`, `TX_AXI_TLAST`, `TX_AXI_TKEEP`, `TX_AXI_TDATA`, and `TX_AXI_TUSER`.

Data transfer from the TX_AXI interface takes the lowest priority. The Ethernet 10G MAC core converts `TX_AXI_TDATA` directly into `XGMII_TXD`. Hence, `TX_AXI_TDATA` must start with header information, i.e., destination address, source destination, Ethernet length or type and/or VLAN tags.

Note: In all the figures containing `TX_AXI_TDATA`, the header octets are labeled as `DATA` to illustrate the conversion between TX AXI interface and XGMII TX.

Figure 12: Conversion from TX AXI ST to XGMII TX



Priority flow control takes 2nd and 3rd priority, while faulty link condition has the highest priority. Whenever a higher priority event arrives at the TX channel, it needs to wait for the on-going transfer to complete before the higher priority event takes effect. Details of priority handling are described in **Priority Flow Control (Duplex Mode)** on page 32 and **Link Fault Sequence** on page 35.

Based on the user-defined Programmable Inter Packet Gap, XGMII TX also inserts the minimum inter packet gap (IPG) between data frames and/or TX Pause frames. However, faulty link condition and RX Pause mechanism start at Octet0 without the minimum IPG.

Refer to **Figure 26: Transmitting Pause Frame and Corresponding Statistic Reporting Increments** on page 32, **Figure 27: Priority at XGMII TX between TX_PAUSE_GEN and TX_AXI_ST** on page 32, **Figure 28: Priority Flow Control at XGMII TX** on page 34, **Figure 29: Link Condition – Detection and Transmission** on page 36, and **Figure 30: Faulty Link Condition during Pause Mode with RX_QUANT** on page 37 for more details on the priority handling and IPG insertion at XGMII TX.

XGMII RX

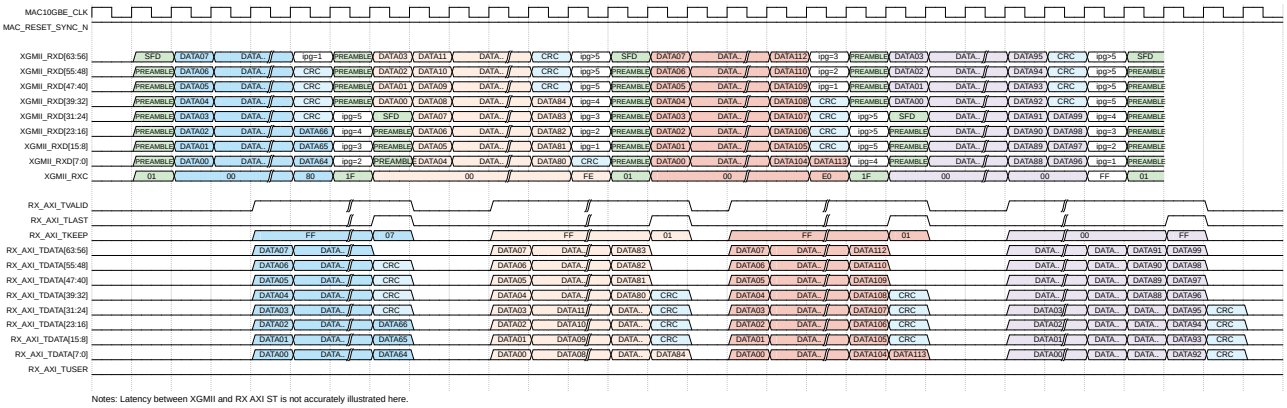
Every incoming RX Ethernet frame from the FPGA Ethernet 10G PCS is decoded based on PREAMBLE, IDLE, and TERMINATE octets. Every RX frame is checked for the following criteria:

- Physical frame length
- FCS check
- Error Frame
- Pause frame
- Address or broadcast filtering

More details on the RX frame handling is described in **Erroneous RX Frame** on page 27, **Address Filtering and Broadcast Filtering at RX** on page 29, and **Decoding Pause Frame at RX** on page 33.

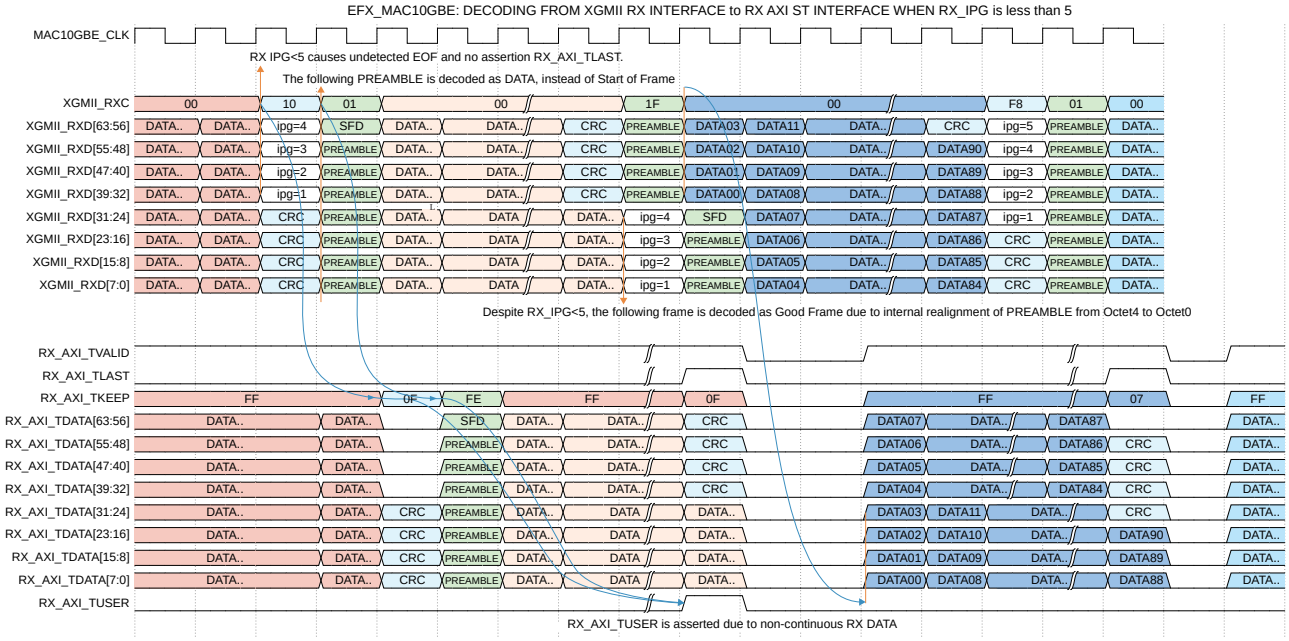
Generally, at the RX, the Ethernet 10G MAC core decodes XGMII_RXD and XGMII_RXC, and outputs them at the RX AXI ST interface. The Ethernet 10G MAC core is capable of decoding XGMII packets with IPG ≥ 5 octets and handling PREAMBLE starting at either Octet0 or Octet4. For the Ethernet frames that start at Octet4, the Ethernet 10G MAC core realigns the data streaming so that RX_AXI_TKEEP starts with FF.

Figure 13: Conversion from XGMII RX to RX AXI ST



Important: With reference to the 10G Ethernet protocol, the minimum RX IPG is 5. XGMII RX Packet with < 5 octets of IPG, especially when the RX IPG are occupying octet4, octet5, octet6 and octet7, causes the Ethernet 10G MAC core to incorrectly decode PREAMBLE, IDLE and TERMINATE octets, results in incorrect data frames at RX AXI ST 64 interfaces, as illustrated by **Figure 14: Incorrect Decoding when RX IPG < 5** on page 18.

Figure 14: Incorrect Decoding when RX IPG < 5



Note: Latency between XGMII_RX signals and RX_AXI signals are not accurately illustrated here.

Data Streaming Mode for Transmission

In Ethernet 10G MAC core, there are 2 data streaming modes for TX channel:

- **Cut Through** mode
- **Store Forward** mode

Note: The **Store Forward** mode is available in Efinity software version 2024.2.294.1.19 and later.

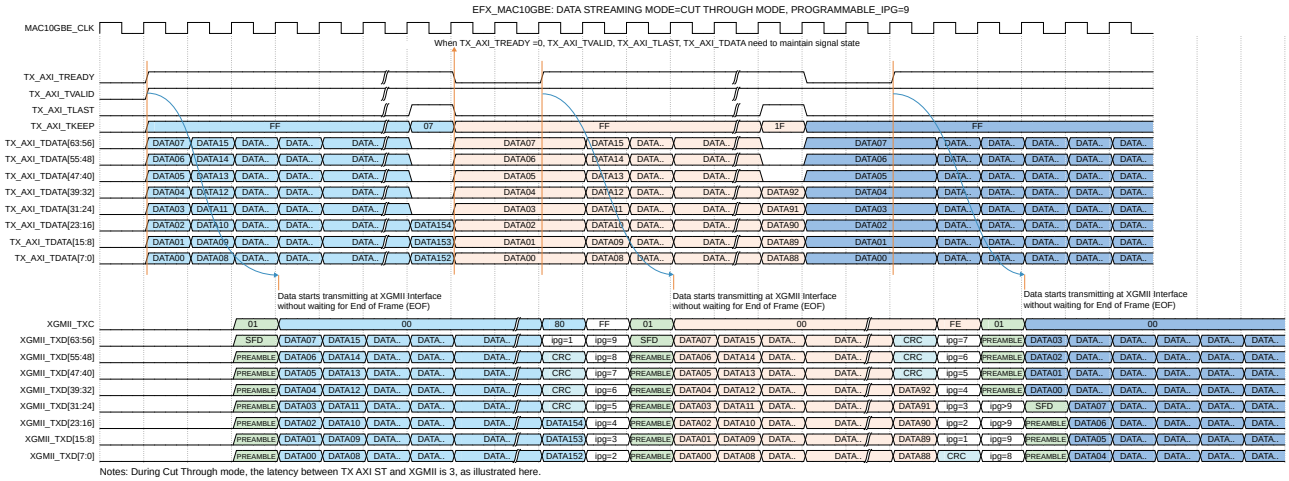
Cut Through Mode

During **Cut Through** mode, the Ethernet 10G MAC core starts transmitting the Ethernet packet without waiting for the end of frame (EOF).

In the event of errors during **Cut Through** mode, the Ethernet frame becomes error frame, i.e., `XGMII_TXD` = `FE` and `XGMII_TXC` = `1`, and are terminated. More details on the error TX frame is described in **Terminating Bad TX Frames** on page 23.

Figure 15: Cut Through Mode on page 19 illustrates the operation of 3 Ethernet packets during **Cut Through** mode.

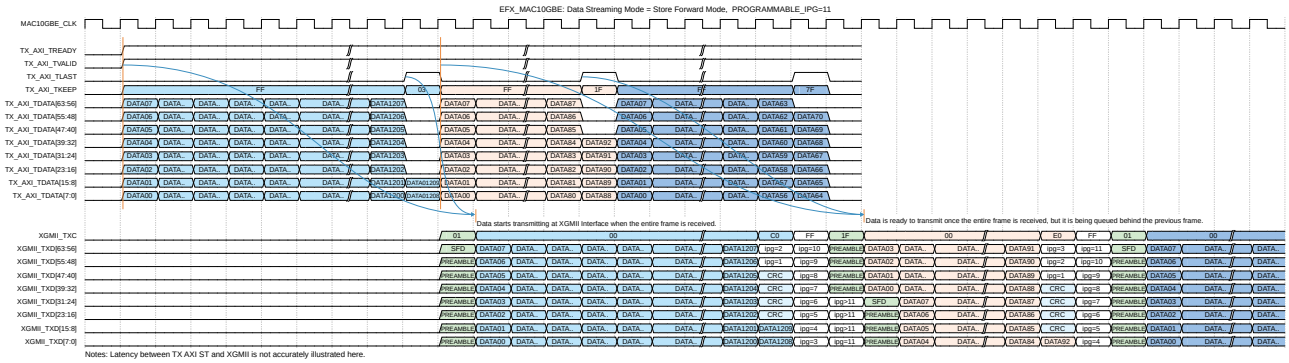
Figure 15: Cut Through Mode



Store Forward Mode

During **Store Forward** mode, Ethernet 10G MAC core starts transmitting the Ethernet packet when the entire packet is received. **Figure 16: Store Forward Mode** on page 20 illustrates the **Store Forward** operation of 3 Ethernet packets.

Figure 16: Store Forward Mode



During **Store Forward** mode, you need to configure the `TXFIFO_DEPTH` to store the full TX packet. The depth of TXFIFO must be large enough to store the maximum length of the TX Frames intended for transfer. The `TXFIFO_DEPTH` setting is also dependent on the VLAN tag enablement, since there are additional bytes for VLAN tag. The minimum `TXFIFO_DEPTH` is `8`, which is to accommodate the minimum payload size.

$$\text{TXFIFO\_DEPTH}_{\text{non VLAN tag}} \geq \frac{14\ \text{Header Bytes} + \text{Maximum Payload Size}}{8}$$

$$\text{TXFIFO\_DEPTH}_{\text{VLAN tag}} \geq \frac{18\ \text{Header Bytes} + \text{Maximum Payload Size}}{8}$$

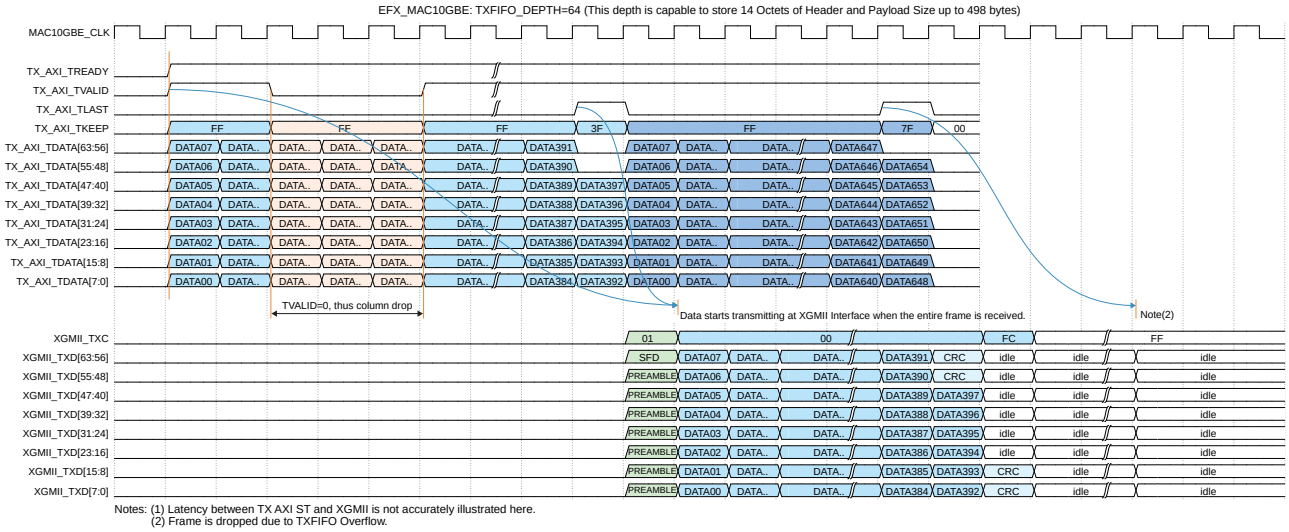
The storing mechanism of TXFIFO is based on the entirety of the frame, as shown by **Figure 17: Storing Mechanism of TXFIFO during Store Forward Mode** on page 21. When `TX_AXI_VALID` = `1`, TXFIFO stores the corresponding `TX_AXI_TLAST`, `TX_AXI_TKEEP` and `TX_AXI_TDATA`. When `TX_AXI_VALID` = `0`, TXFIFO stops storing, but still waits for the full frame before starts transmitting.

In the event of TXFIFO overflow, where `TXFIFO_DEPTH` is insufficient to store the entire full frame with maximum frame size, the entire frame is scrapped.

Once the entire frame is stored, TXFIFO starts transmitting the Ethernet frame at XGMII TX interface subjecting to the priority as described in **XGMII TX** on page 16. However, in the event of non-streaming data, the Ethernet frame becomes an error frame, i.e., `XGMII_TXD` = `FE`, `XGMII_TXC` = `1`, and is terminated.

More details on the Error TX frame is described in **Terminating Bad TX Frames** on page 23.

Figure 17: Storing Mechanism of TXFIFO during Store Forward Mode

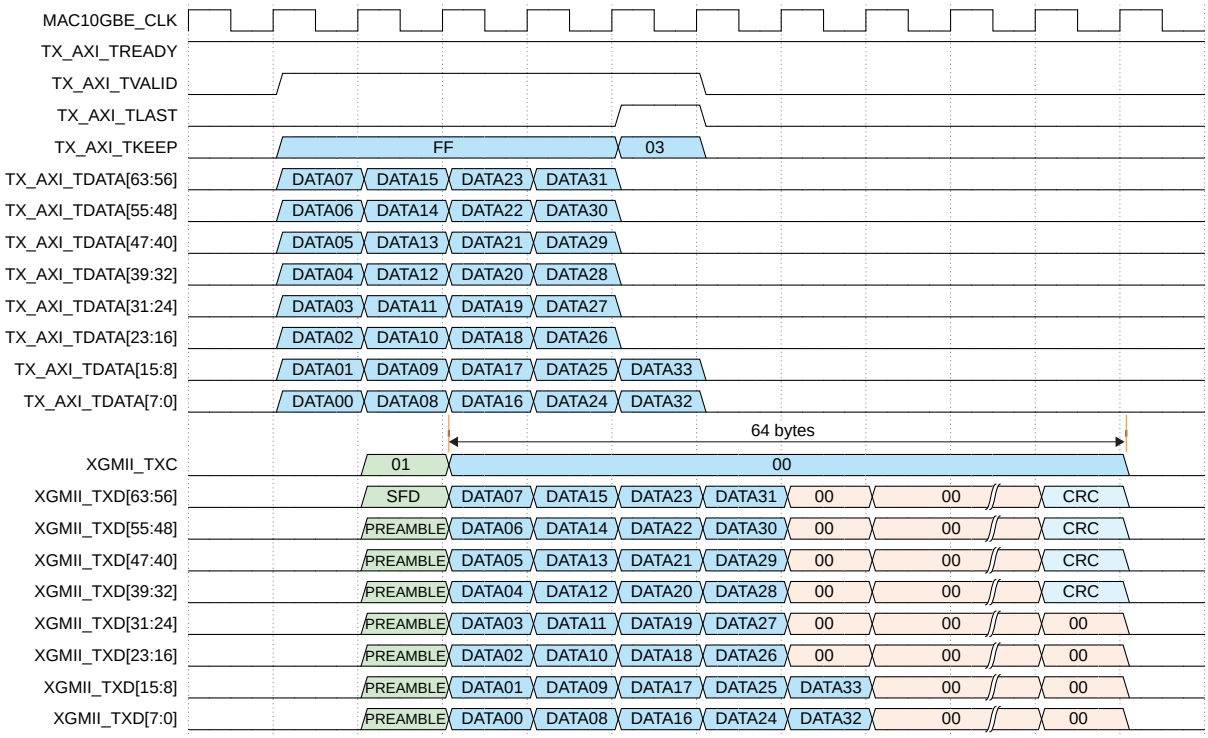


Notes: (1) Latency between TX AXI ST and XGMII is not accurately illustrated here.
(2) Frame is dropped due to TXFIFO Overflow.

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 Ethernet 10G MAC core automatically pads with zeros to fulfill the minimum payload size.

Figure 18: Auto Padding



Notes: Latency between TX AXI ST and XGMII is not accurately illustrated here.

Terminating Bad TX Frames

The following are the causes of error at TX channels:

- Non-Compliant `TX_AXI_TLAST`
- Assertion of `TX_AXI_TUSER`
- Non-streaming TX data
- Dropped `TX_AXI_TVALID` (**Cut Through** mode only)
- TXFIFO overflow (**Store Forward** mode only)

Non-Compliant TX_AXI_TLAST

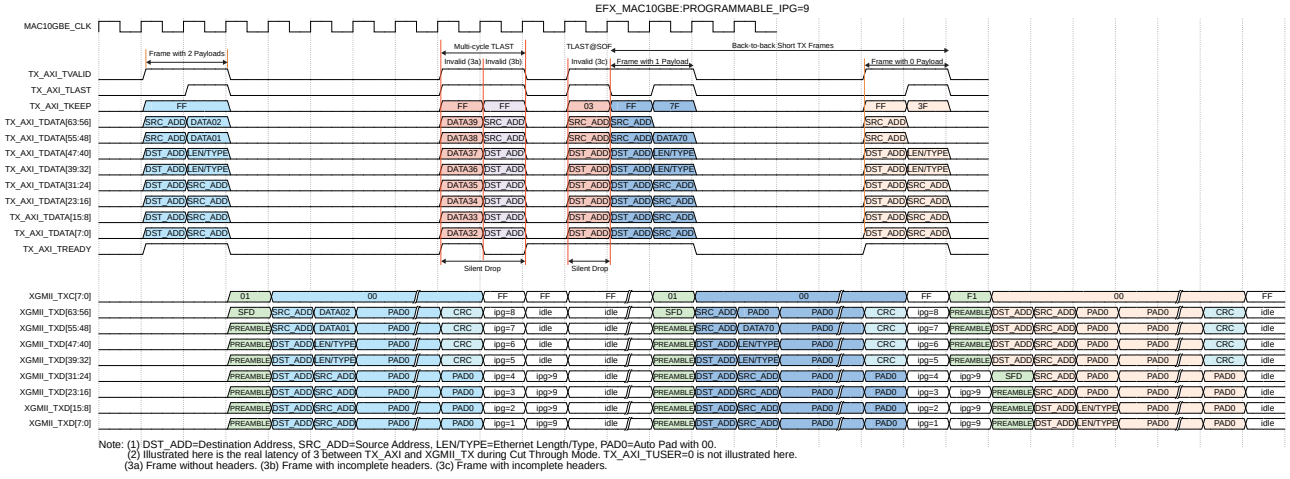
An Ethernet frame should consist minimally destination address, source address, optional VLAN tag and ETHERTYPE/LEN. This means that an Ethernet frame spans across a minimum of 2 clock cycles, where `TX_AXI_TLAST` should be pulse based.

Figure 19: Invalid TX Frames Due to Non-Compliant TX_AXI_TLAST on page 23 illustrates a case of non-compliant `TX_AXI_TLAST`.

- At label with (3c), the Ethernet frame spans only 1 clock cycle. Here, `TX_AXI_TLAST` is asserted at start of frame, results in silent drop of the ≤ 8 octets short frame with incomplete headers.
- At labels (3a) and (3b), `TX_AXI_TLAST` spans multi-clock cycles, Ethernet 10G MAC core decodes it as 2 invalid short frames and silently drop the 2 frames of ≤ 8 octets.

Note: There is no indicator on the silently dropped TX frame. In replacement of the silently dropped TX Frame, XGMII TX transmits IDLE.

Figure 19: Invalid TX Frames Due to Non-Compliant `TX_AXI_TLAST`

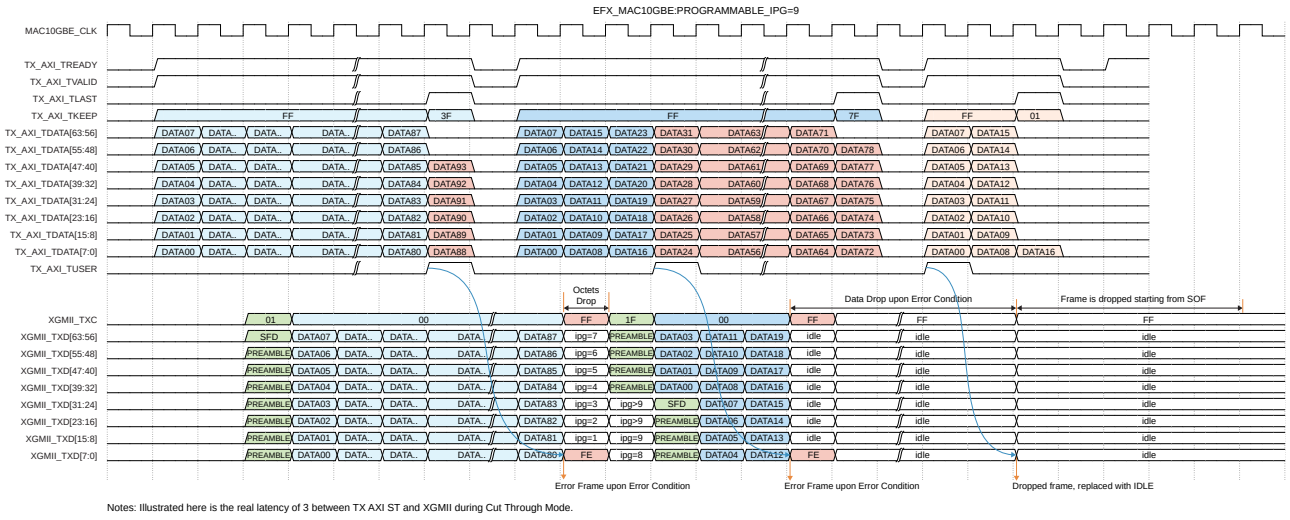


Note: (1) DST_ADD=Destination Address, SRC_ADD=Source Address, LEN/TYPE=Ethernet Length/Type, PAD0=Auto Pad with 00.
(2) Illustrated here is the real latency of 3 between TX_AXI and XGMII_TX during Cut Through Mode. TX_AXI_TUSER=0 is not illustrated here.
(3a) Frame without headers. (3b) Frame with incomplete headers. (3c) Frame with incomplete headers.

Assertion of TX_AXI_TUSER

The assertion of TX_AXI_TUSER indicates that there is an error in the data streaming. Upon the assertion of TX_AXI_TUSER, the Ethernet 10G MAC core terminates with error frame, i.e., XGMII_TXD = FE, XGMII_TXC = 1, and drops any remaining data frame. Refer to **Figure 20: Error Frame Due to Assertion of TX_AXI_TUSER** on page 24.

Figure 20: Error Frame Due to Assertion of TX_AXI_TUSER



Notes: Illustrated here is the real latency of 3 between TX AXI ST and XGMII during Cut Through Mode.

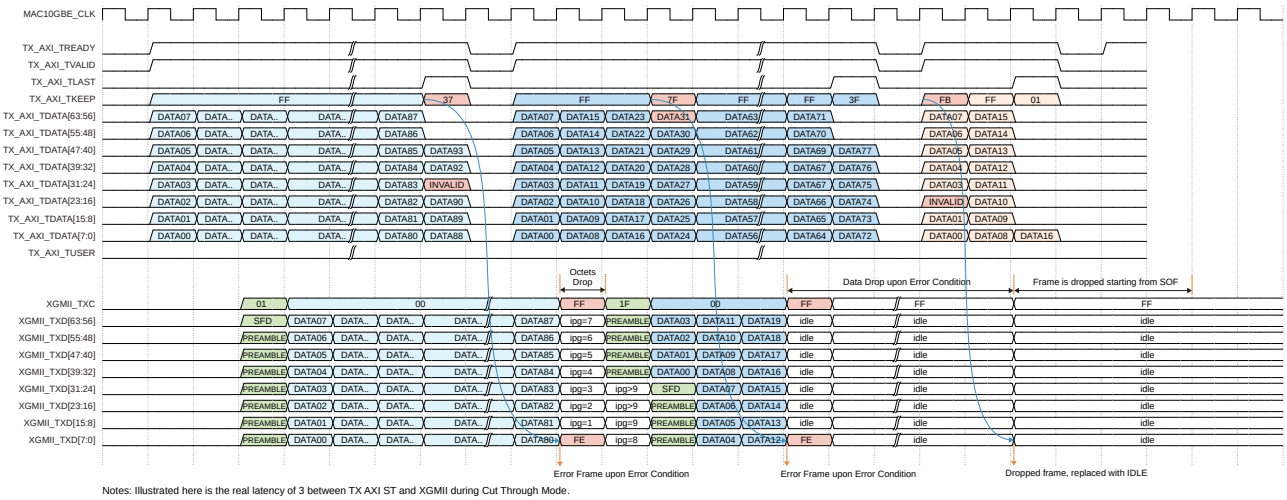
Note: When TX_AXI_TVALID = 0, the whole column of TX_AXI signals (TDATA, TKEEP, TLAST, TUSER) is deemed as invalid, as described in **TX AXI ST 64** on page 12. Any assertion of TX_AXI_TUSER during TX_AXI_TVALID = 0 is not an error.

Non-Streaming TX Data

For Ethernet application, `TX_AXI_TDATA` needs to be streaming and continuous, where `TX_AXI_TKEEP` = `FF` except at `TX_AXI_TLAST`. Streaming data, at `TX_AXI_TLAST`, means `TX_AXI_TKEEP` = {01, 03, 07, 0F, 1F, 3F, 7F, FF}.

Upon the occurrence of non-streaming TX data, Ethernet 10G MAC core terminates with error frame, i.e., `XGMII_TXD` = `FE`, `XGMII_TXC` = `1`, and drops the remaining of the data frame. If the error occurs at the start of frame (SOF), Ethernet 10G MAC core drops the entire frame and outputs `IDLE` octets. Refer to **Figure 21: Error Frame Due to Non-Streaming TX Data** on page 25.

Figure 21: Error Frame Due to Non-Streaming TX Data



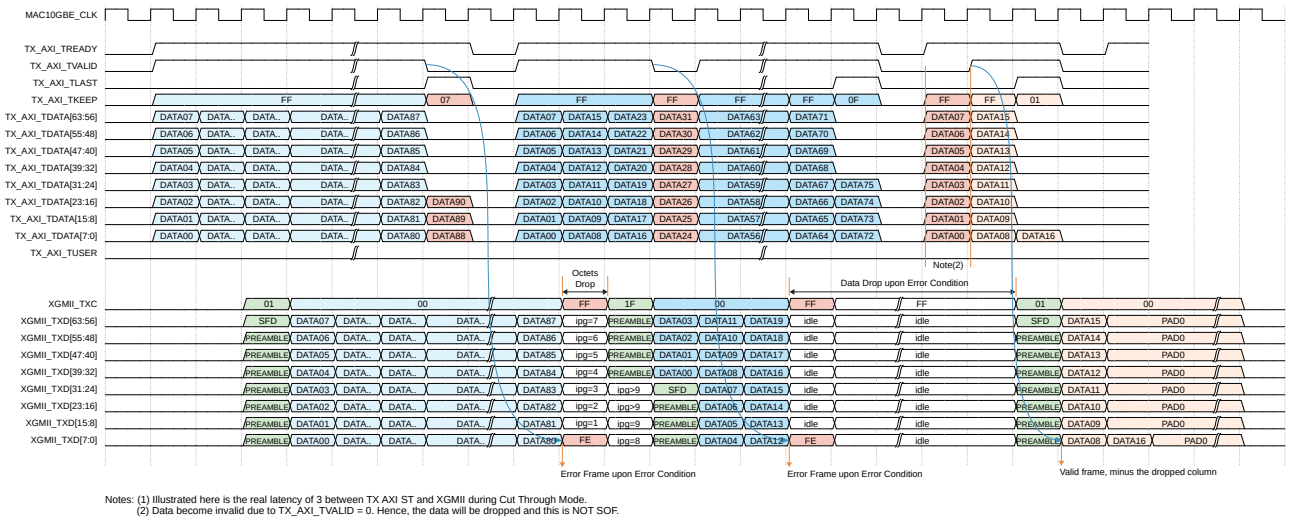
Notes: Illustrated here is the real latency of 3 between TX AXI ST and XGMII during Cut Through Mode.

Dropped TX_AXI_TVALID Before End of Frame

Streaming data means continuous DATA bytes from the start of frame (SOF) until the end of frame (EOF). This means that TX_AXI_TVALID should remain asserted from SOF until EOF. In **Cut Through** mode, a dropped TX_AXI_TVALID during the course of streaming data breaks the continuity of a streaming data. When this occurs, the XGMII TX outputs an error frame. Refer to **Figure 22: Error Frame Due to Dropped TX_AXI_TVALID (Cut Through Mode Only)** on page 26.

Note: When TX_AXI_TVALID = 0, the entire column of TX AXI signals (TDATA, TKEEP, TLAST, and TUSER) is deemed as invalid. **Figure 22: Error Frame Due to Dropped TX_AXI_TVALID (Cut Through Mode Only)** on page 26 illustrates how TX_AXI_TVALID defines the start of frame (SOF).

Figure 22: Error Frame Due to Dropped TX_AXI_TVALID (Cut Through Mode Only)



Notes: (1) Illustrated here is the real latency of 3 between TX AXI ST and XGMII during Cut Through Mode.
(2) Data become invalid due to TX_AXI_TVALID = 0. Hence, the data will be dropped and this is NOT SOF.

However, in **Store Forward** mode, the Ethernet 10G MAC core stores the entire frame before starts transmitting. Based on the storing mechanism shown in **Figure 17: Storing Mechanism of TXFIFO during Store Forward Mode** on page 21, a dropped TX_AXI_TVALID means that the whole column of TX_AXI_* signals (TDATA, TKEEP, TLAST, and TUSER) is not stored in the TXFIFO. Upon TX_AXI_TLAST, i.e., once the entire frame is stored, the Ethernet 10G MAC core starts transmitting the Ethernet frame at the XGMII TX interface. Hence, a dropped TX_AXI_TVALID does not warrant an error frame during **Store Forward** mode.

TX FIFO Overflow

During **Store Forward** mode, you need to configure the TXFIFO_DEPTH value to store the TX_AXI_TDATA. In the event of a TX FIFO overflow, where TXFIFO_DEPTH is insufficient to store the entire full frame with maximum frame size, the entire frame is scrapped. The XGMII TX outputs IDLE frame in the replacement of the scrapped transfer. **Figure 35: TX Statistic – cnt_tx_frame_error_txfifo_overflow** on page 41 shows the TX FIFO overflow condition, together with the corresponding statistic reporting.

Erroneous RX Frame

Similar to the TX, the Ethernet 10G MAC core asserts `RX_AXI_TUSER` to indicate an error in the RX data frame. The following lists the types of error occurring at RX channel:

- Undersized RX frame
- Oversized RX frame
- RX frame length mismatch
- FCS error
- Non-streaming RX data frame
- Error frame

Undersized RX Frame

For Ethernet frame, the minimum payload size is 46 for non-VLAN tagged and 42 for VLAN tagged frame. If the RX frame does not meet the minimum payload size, it is flagged as an *undersized RX frame*. When this occurs, the Ethernet 10G MAC core asserts `RX_AXI_TUSER` and increments the RX statistic reporting, `cnt_rx_frame_undersized`. **Figure 36: RX Statistic – Undersized RX Frame** on page 42 describes an *undersized RX frame*.

Oversized RX Frame

At RX, every decoded RX frame is checked for its physical frame length and compares with the user's configured **Maximum Transmission Unit Frame Length**. If the physical frame length is larger than **Maximum Transmission Unit Frame Length**, it is flagged as an *oversized RX frame*. When this occurs, the Ethernet 10G MAC core asserts `RX_AXI_TUSER` and increments the RX statistic reporting, `cnt_rx_frame_oversized`. **Figure 37: RX Statistic – Oversized RX Frame** on page 42 describes an *oversized RX frame*.

Mismatched Length RX Frame

For Ethernet frame with payload size ≤ 1500, the Ethernet 10G MAC core cross-checks the physical frame length against its ETHERTYPE/LEN field. If the physical frame length mismatches the value in the field, it is flagged as *RX frame with mismatched length*. When this occurs, the Ethernet 10G MAC core asserts `RX_AXI_TUSER` and increments the RX statistic reporting, `cnt_rx_frame_mismatched_length`. **Figure 38: RX Statistic – Mismatched Length RX Frame** on page 43 describes a *RX frame with mismatched length*.

Frame Check Sequence (FCS) Error

Every decoded RX frame undergoes CRC check at XGMII RX. If the CRC check fails, then there is an error in the RX frame. When this occurs, the Ethernet 10G MAC core asserts `RX_AXI_TUSER` and increments the RX statistic reporting, `cnt_rx_frame_error_fcs`. **Figure 39: RX Statistic – RX Frame with FCS Error** on page 43 describes a *RX frame with FCS error*. FCS error is also known as CRC error.

Non-Streaming RX Data Frame

A non-streaming RX frame contains non-continuous data; hence, it is invalid and not counted as part of the RX frame length. The RX frame length calculation may affect the statistic reporting concerning *oversized RX frame*, *undersized RX frame* and *RX frame with mismatched length*.

When a non-streaming RX frame occurs, the Ethernet 10G MAC core asserts `RX_AXI_TUSER` and increments the RX statistic reporting, `cnt_rx_frame_errors`. The Ethernet 10G MAC core converts `XGMII_RXD` and `XGMII_RXC` into `RX_AXI_TDATA` and `RX_AXI_TKEEP` respectively, based on WYSIWYG concept. **Figure 41: RX Statistic Reporting – Non-Streaming RX Data Frame** on page 45 describes a *non-streaming RX frame*.

Error Frame

Error frame is `XGMII_RXD` = `FE` and `XGMII_RXC` = `1`. When error frame occurs, the final 4 octets are decoded as CRC, results in CRC error. Here, the Ethernet 10G MAC core asserts `RX_AXI_TUSER` and increments the RX statistic reporting, `cnt_rx_frame_errors` and `cnt_rx_frame_error_fcs`. Refer to **Figure 42: RX Statistic Reporting – Error Frame** on page 46.

Address Filtering and Broadcast Filtering at RX

The Ethernet 10G MAC core supports unicast, multicast, and broadcast filtering for the incoming RX Ethernet frame. When any of these filtering is enabled, each of the incoming RX Ethernet frame is filtered if the destination address mismatches the user-configured **MAC Source Address**.

When the broadcast filtering is enabled, all RX frames with destination address of 48'hFFFF_FFFF_FFFF are filtered. To enable broadcast filtering, you need to set the **Broadcast Filtering** option to **Enable**.

To enable **Unicast** or **Multicast Address Filtering**, you should drive a pseudo-async signal to port `rx_address_filtering_mask`. The `rx_address_filtering_mask` is a 48-bit width input that allows you to apply bit-wise address filtering to the RX frame coming from the Ethernet 10G PCS. Asserting any bit to 1 means that the respective address bit is compared. Setting the bit to 0 means that the respective address bit is not compared. **Table 3: Examples of Unicast, Multicast, and Broadcast Filtering Operation** on page 30 shows the examples of unicast, multicast, and broadcast filtering operation.

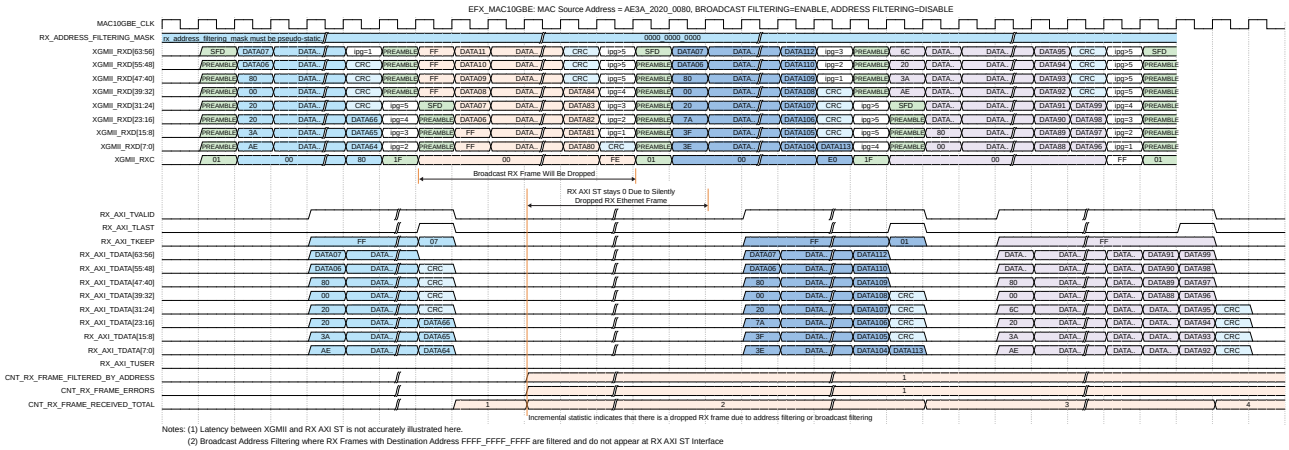
During the address and broadcast filtering operations, any RX Ethernet frame with mismatched address is filtered and silently dropped. Dropped RX frame does not appear at RX AXI ST64 interface, hence you may need to monitor the statistic reporting, namely `cnt_rx_frame_filtered_by_address`, to track the filtered (and dropped) RX frames. **Figure 23: Silent Drop of Broadcast RX Frame** on page 31, **Figure 24: Silent Drop of Broadcast RX Frame during Unicast Address Filtering** on page 31, and **Figure 25: Multicast Address Filtering** on page 31 illustrate on the broadcast, unicast and multicast filtering operations together with the incremental statistic reporting.

Table 3: Examples of Unicast, Multicast, and Broadcast Filtering Operation

Broadcast Filtering	rx_address_ filtering_mask	MAC SOURCE ADDRESS	Destination Address[2]	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_0000	48'hAE3A_ 2020_0080	48'h3E3F_ 7A20_0080	Filtered & Dropped	Multicast
Don't Care	48'h0FF0_ 00FF_FFFF	48'hAE3A_ 2020_0080	48'h3F3E_ 7A20_0080	Filtered & Dropped	Multicast
Don't Care	48'h0FF0_ 00FF_FFFF	48'hAE3A_ 2020_0080	48'h3E3F_ 7A20_0080	Output at RX AXI ST	Multicast
Don't Care	48'h6FFA_ A5FF_FFFF	48'hAE3A_ 2020_0080	48'h3E3F_ 7A20_0080	Output at RX AXI ST	Multicast
Broadcast					
ENABLE	Don't Care	Don't Care	48'hFFFF_ FFFF_FFFF	Filtered & Dropped	Broadcast
DISABLE	Don't Care	Don't Care	48'hFFFF_ FFFF_FFFF	Output at RX AXI ST	-
Address Filtering Is Not Enabled					
DISABLE	48'h0000_ 0000_0000	48'hAE3A_ 2020_0080	48'hAE3A_ 2020_0080	Output at RX AXI ST	-

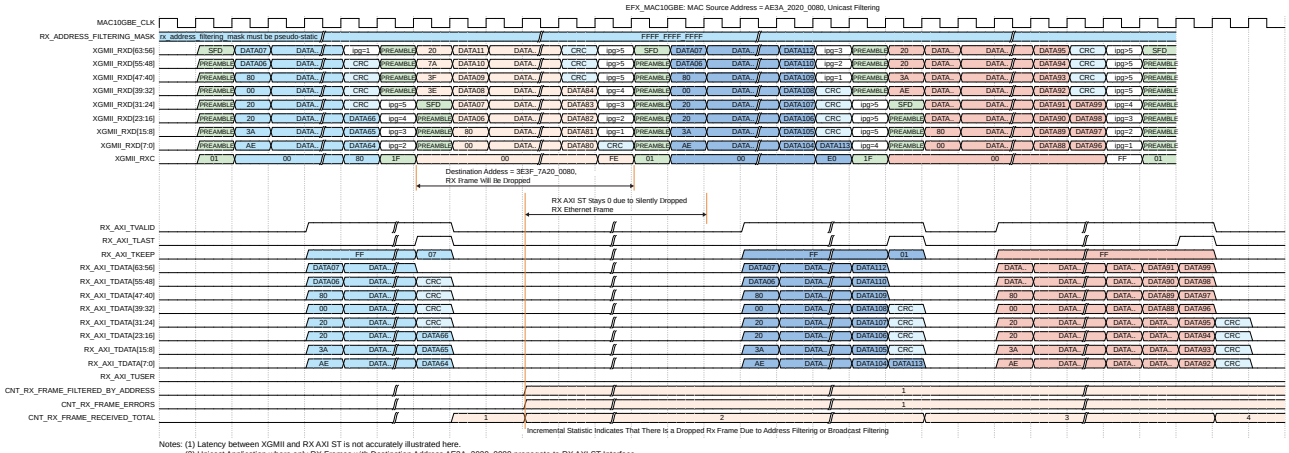
[2] Destination address of the incoming RX Ethernet frame.

Figure 23: Silent Drop of Broadcast RX Frame



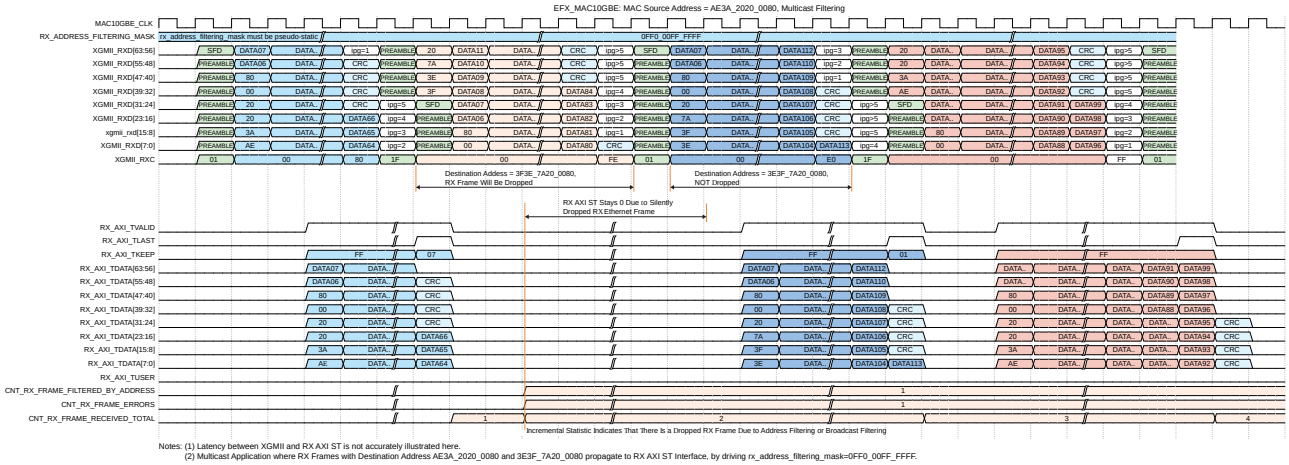
Notes: (1) Latency between XGMII and RX AXI ST is not accurately illustrated here.
(2) Broadcast Address Filtering where RX Frames with Destination Address FFFF_FFFF_FFFF are filtered and do not appear at RX AXI ST Interface

Figure 24: Silent Drop of Broadcast RX Frame during Unicast Address Filtering



Notes: (1) Latency between XGMII and RX AXI ST is not accurately illustrated here.
(2) Unicast Application where only RX Frames with Destination Address AE3A_2020_0080 propagate to RX AXI ST Interface

Figure 25: Multicast Address Filtering



Notes: (1) Latency between XGMII and RX AXI ST is not accurately illustrated here.
(2) Multicast Application where RX Frames with Destination Address AE3A_2020_0080 and 3E3F_7A20_0080 propagate to RX AXI ST Interface, by driving rx_address_filtering_mask=0FF0_00FF_FFFF.

Priority Flow Control (Duplex Mode)

The Ethernet 10G MAC core supports the flow control at TX and RX duplex mode.

Send Pause Frame at TX

To send PAUSE frame, assert `tx_pause_gen` to generate `PAUSE_REQUEST`. port `tx_pause_gen` is pulse-based. Hence, you must assert it for 1 clock cycle only. The assertion of port `tx_pause_gen` generates the `PAUSE_REQUEST`, and Ethernet 10G MAC core captures the request by asserting the output port `tx_pause_busy`. The `PAUSE_REQUEST` is only effective when `tx_pause_busy` = `1`. **Figure 26: Transmitting Pause Frame and Corresponding Statistic Reporting Increments on page 32** shows how the `PAUSE_REQUEST` is sent from `tx_pause_gen` to XGMII TX interface, together with the corresponding statistics reporting.

In terms of priority, the `PAUSE_REQUEST` has a higher priority over the DATA frame. However, if the `PAUSE_REQUEST` comes during the DATA frame transfer, then the `PAUSE_REQUEST` queues up, and is served once the on-going DATA frame transfer finishes.

Port `TX_PAUSE_BUSY` = `1` indicates that there is a `PAUSE_REQUEST` in the queue. When `TX_PAUSE_BUSY` = `1`, you must not assert any new `PAUSE_REQUEST`.

Figure 26: Transmitting Pause Frame and Corresponding Statistic Reporting Increments

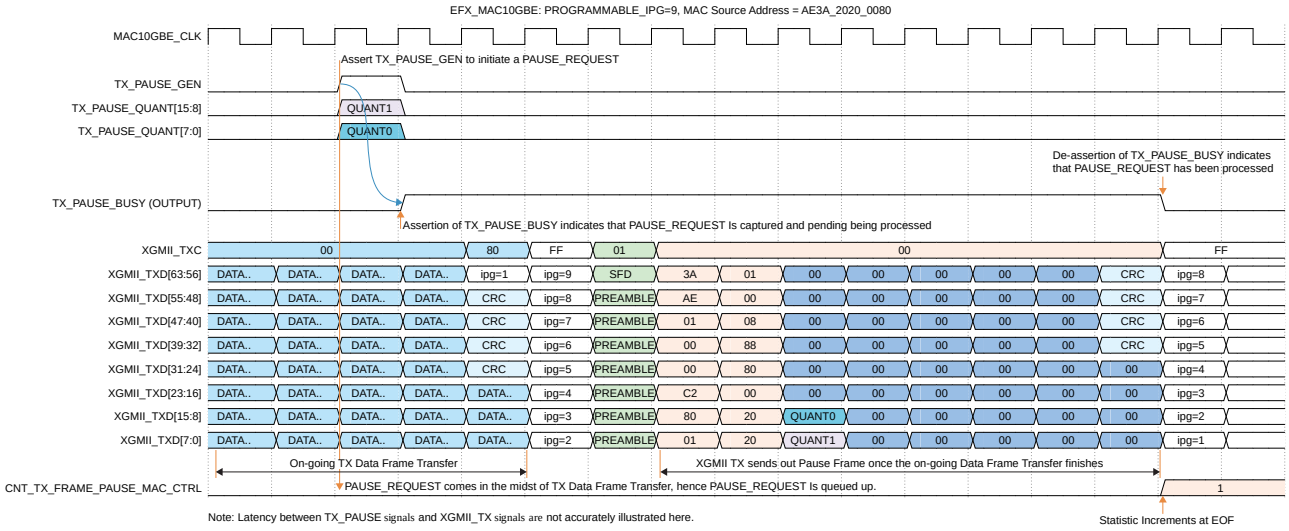
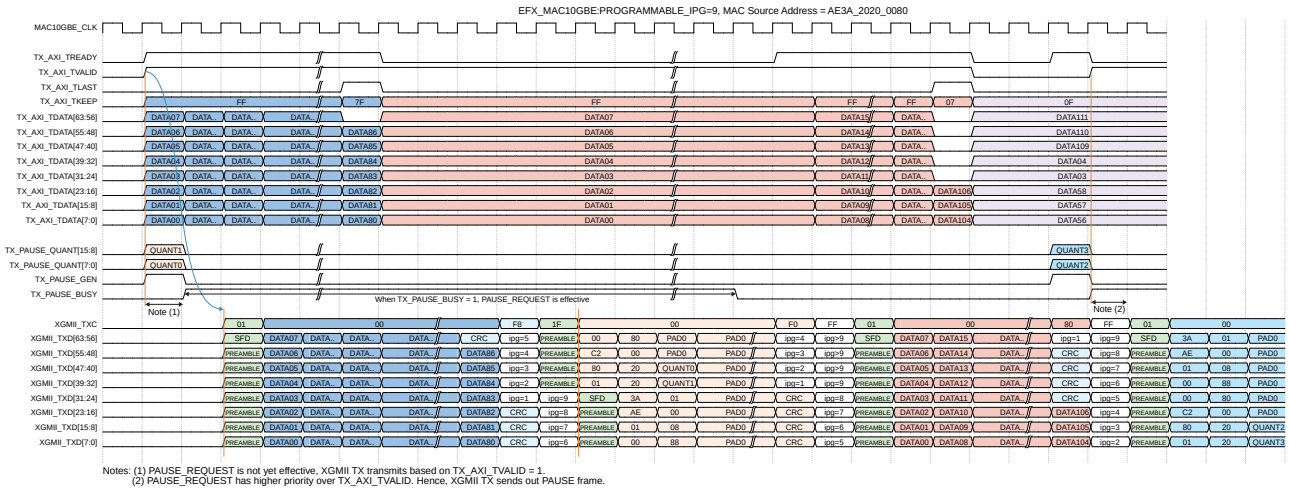


Figure 27: Priority at XGMII TX between TX_PAUSE_GEN and TX_AXI_ST



Decoding Pause Frame at RX

At RX, the Ethernet 10G MAC core decodes and identifies the pause frame from the RX Ethernet frames which are coming from the Ethernet 10G PCS. The RX Pause frame is decoded based on the reserved address `01-80-C2-00-00-01`. Once the RX pause frame is identified, it is decoded to extract the embedded `RX_QUANT`. Once decoded, the `RX_QUANT` value is directed to the TX channels, where the pause mechanism takes place with the `RX_QUANT` value.

A quanta is the time that is required to transmit 512 bits at the speed of the port. For Ethernet 10G MAC core, `RX_QUANT` = `1` signifies that the XGMII TX pauses for 64 octets. When the pause mechanism takes place, the XGMII TX outputs `IDLE` for the duration of `RX_QUANT`. The duration of `RX_QUANT` starts the pause count at Octet0.

You may assert port `rx_ignore` to disregard the decoded `RX_QUANT` and the pause mechanism does not take place. Port `rx_ignore` must be assigned with pseudo-static input.

Note: The RX Pause frame is not subjected to address filtering. The RX Pause frame is decoded purely based on the pause frame unique address, frame type ID, pause opcode, and CRC octets.

RX Pause frame is only valid if it passes a CRC check. If the RX Pause frame contains a CRC Error, then the RX Pause frame is erroneous. The extracted `RX_QUANT` is invalid and is not propagated to the TX channel. At the same time, like the normal RX frames, the RX Pause frame is converted from the XGMII interface and outputs at the RX AXI ST interface. Refer to **Figure 40: RX Statistic – RX Pause Frame** on page 44 for more details.

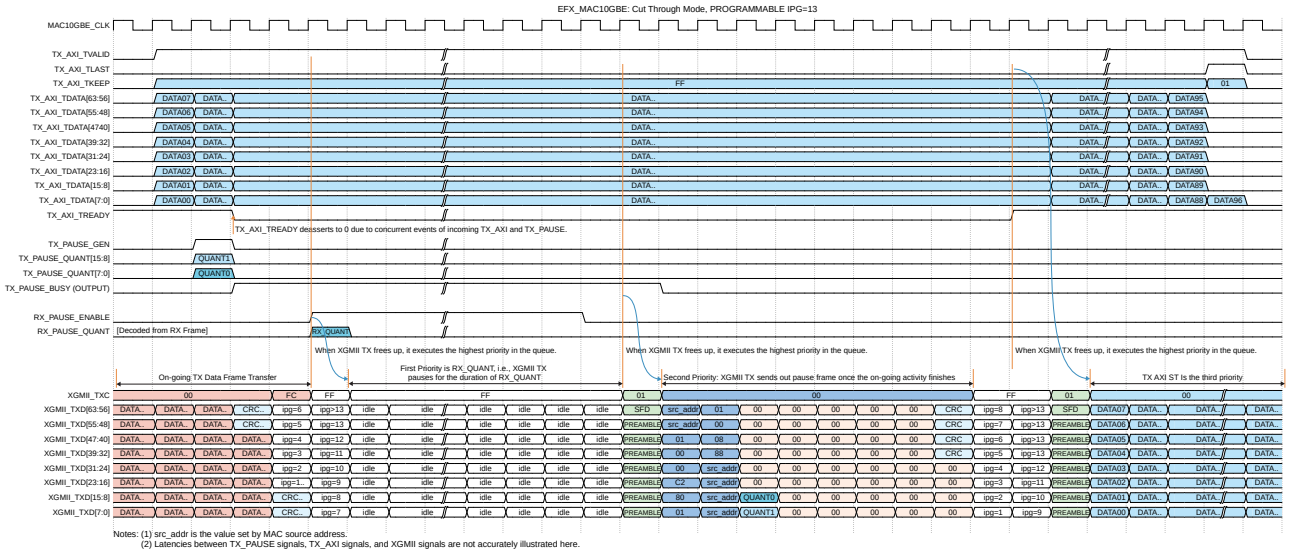
Note: RX Pause frame, whether undersized or oversized, is processed as valid RX Pause frame.

Upon arriving at TX channel, the decoded `RX_QUANT` has higher priority over `PAUSE_REQUEST` (triggered by `tx_pause_gen`) and TX AXI ST data frame transfer. However, if the decoded `RX_QUANT` arrives during the transfer of the TX pause frame or TX data frame, the pause mechanism queues up and only takes place once the on-going TX activity completes.

Figure 28: Priority Flow Control at XGMII TX on page 34 describes the priority flow control among the 3 events. In this illustration, both the `IPG` and `IDLE` are inter-changeable, and carry the value `07`. An `IDLE` is used to illustrate the XGMII TX output when `RX_QUANT` is effective, while the `IPG` is used to illustrate the mandatory insertion of gaps based on the user-configured Programmable Inter Packet Gap between the TX Ethernet frames.

Note: The Ethernet 10G MAC core is only able to process one `RX_QUANT` at a time. When `RX_QUANT` is active and effective at TX channel, if there is a new incoming RX Pause frame, the new `RX_QUANT` is ignored and blocked from disrupting the on-going (or queued) Pause mechanism.

Figure 28: Priority Flow Control at XGMII TX



Notes: (1) src_addr is the value set by MAC source address.
(2) Latencies between TX_PAUSE signals, TX_AXI signals, and XGMII signals are not accurately illustrated here.

Link Fault Sequence

The link fault sequence indicates the condition of the link between 2 Ethernet devices. The Ethernet 10G MAC core comes with **Detect Link Fault** capability. By setting **Detect Link Fault** option to **Enable**, the Ethernet 10G MAC core detects, decodes and communicates the relevant sequences. When the **Detect Link Fault** option is set to **Disable**, there will be no link fault handling in Ethernet 10G MAC core. The following describes the mechanism of the link fault handling when **Detect Link Fault** option is set to **Enable**.

There are 3 types of fault sequences: Local Fault, Remote Fault, and Link Interrupt. A sequence is based on a block of 32-bits. At RX channel, every block of 32-bits data, together with its corresponding control bits, i.e., (`XGMII_RXD`[31:0] and `XGMII_RXC`[3:0]) or (`XGMII_RXD`[63:32] and `XGMII_RXC`[7:4]), are decoded to determine the type of the link fault sequences.

When ≥ 4 consecutive sequences of the same fault type occur at the RX channel, the Ethernet 10G MAC core has detected a faulty link condition. The condition for the fault sequences is as follows:

- *Local Fault*—When XGMII RX detects ≥ 4 consecutive Local Fault sequence, it updates the link condition and communicates with XGMII TX to transmit a remote fault sequence.
- *Remote Fault*—When XGMII RX detects ≥ 4 consecutive Remote Fault sequence, it updates the link condition and communicates with XGMII TX to transmit `Idle`.
- *Link Interrupt*—When XGMII RX detects ≥ 4 consecutive Link Interrupt sequence, it updates the link condition and communicates to the XGMII TX to transmit `Idle`.

To exit from a faulty link condition, the Ethernet 10G MAC core RX needs ≥128 consecutive non-fault sequences.

If ≥ 4 consecutive sequences of the same fault type does not occur at the RX channel, the link condition remains the same. The following are the scenario examples:

- If the current link condition is No Link Fault, a single or 2 local fault sequences do not constitute a new faulty link condition. Hence, the Ethernet 10G MAC core retains the current No Link Fault status. XGMII TX continues to serve `RX_QUANT`, `tx_pause_gen`, or `TX_AXI` signals based on the original priority.
- If current link condition is Remote Fault, an alternating sequence of no-fault and local fault does not warrant a new faulty link condition. Hence, the link condition maintains the existing Remote Fault status. XGMII TX continues to output `IDLE`.
- If the current link condition is Local Fault, a data frame with a frame length of 146 (the data frame is preceded with `PREAMBLE` and `SFD`, payload size = 128 octets), not including RX IPG, contributes a total of 39 no-fault sequences. Hence, the link condition persists with Local Fault. XGMII TX continues to transmit Remote Fault sequences.

Refer to **Table 4: Link Fault Condition: Detection and Transmission** on page 36, which describes the decoding mechanism of the link condition by XGMII RX and the XGMII TX transmission based on the link condition.

Table 4: Link Fault Condition: Detection and Transmission

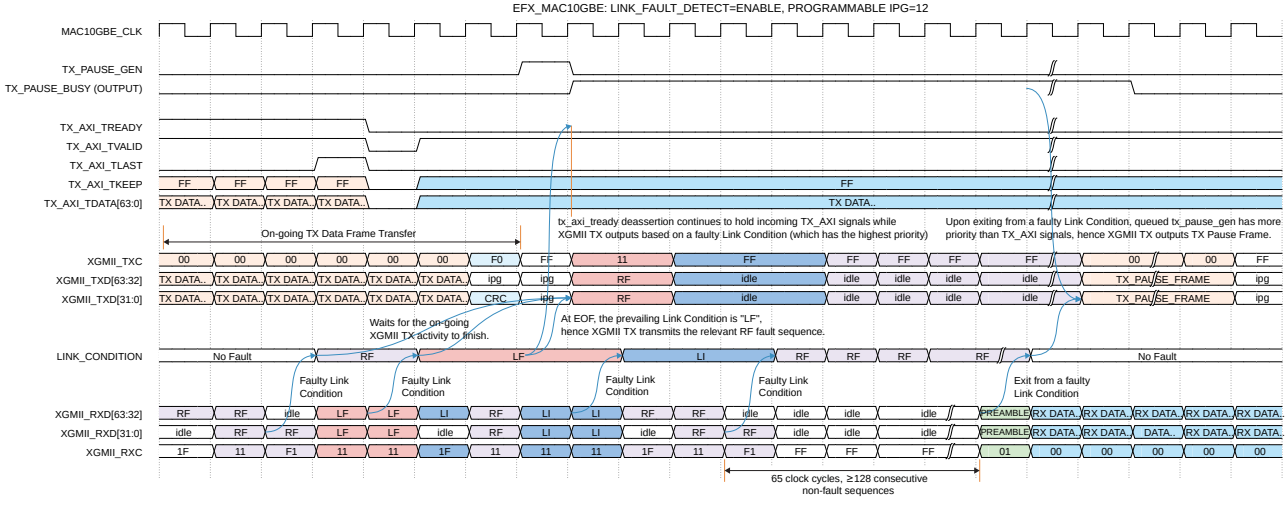
Fault Detection at XGMII RX	Link Condition	XGMII TX Transmission
≥ 4 consecutive local fault	Local Fault	Remote fault sequence
≥ 4 consecutive remote fault	Remote Fault	`Idle`
≥ 4 consecutive link interrupt	Link Interrupt	`Idle`
≥ 128 consecutive non-fault sequences	No Link Fault	One of the following, in descending priority: **1.** `Idle` due to `RX_QUANT` **2.** Pause frame from `tx_pause_gen` **3.** Data from TX AXI **4.** `Idle` due to no activity at TX channel.
< 4 consecutive sequences of the same fault type	No update on link condition, i.e., retain the current link condition	Depends on the current link condition.

At XGMII TX, a faulty link condition has the highest priority. However, if a faulty link condition occurs in middle of the TX transfer, whether it is TX data frame or TX Pause frame, XGMII TX continues with the on-going TX transfer until EOF. Upon EOF, if the link fault condition still prevails, XGMII TX outputs the relevant fault sequences. When XGMII TX is addressing a faulty link condition, the following may occur:

- `TX_AXI_TREADY` deasserts to hold any incoming `TX_AXI` signals.
- `tx_pause_busy` asserts in response to any assertion from `tx_pause_gen` input port.

Refer to **Figure 29: Link Condition – Detection and Transmission** on page 36, which illustrates how the link condition is updated and how XGMII TX prioritizes a faulty link condition above `tx_pause_gen` and `TX_AXI` signals.

Figure 29: Link Condition – Detection and Transmission

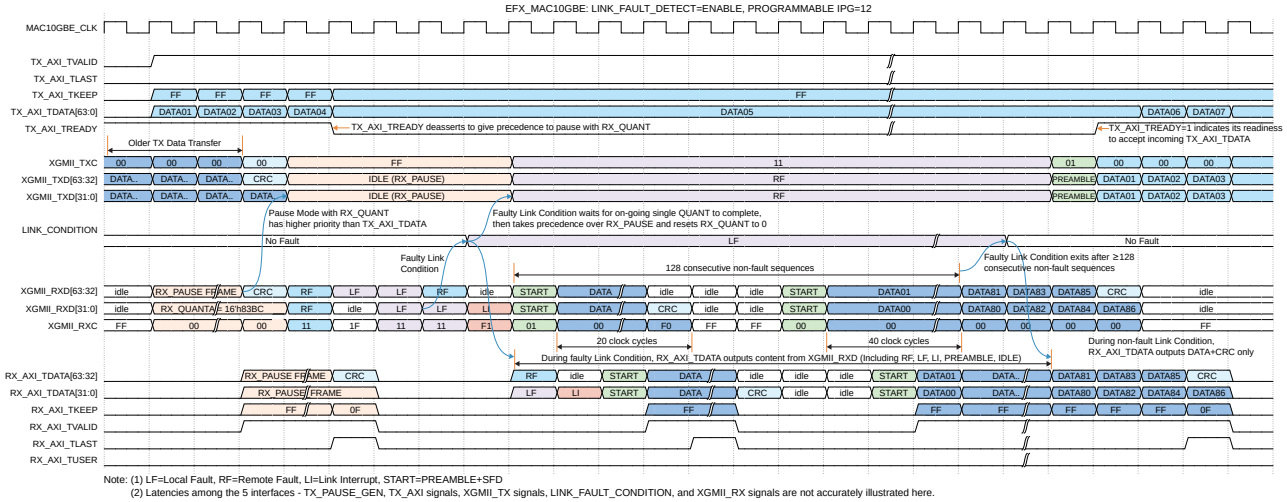


Note: (1) LF=LOCAL FAULT, RF=REMOTE FAULT, LI=LINK INTERRUPT
(2) Latencies among the 5 interfaces - TX_PAUSE_GEN, TX_AXI signals, XGMII_TX signals, LINK_FAULT_CONDITION and XGMII_RX signals are not accurately illustrated here.

If a faulty link condition occurs in the during Pause mode with RX_QUANT, it will wait for the current single Quant to complete. Upon the completion of the current single Quant, if the faulty link condition still prevails, it terminates the Pause state and resets the remaining RX_QUANT to 0. Upon terminating the Pause mode, XGMII TX transmits the relevant fault sequences, depending on the prevailing faulty link condition. Refer to **Figure 30: Faulty Link Condition during Pause Mode with RX_QUANT** on page 37.

Note: One RX_QUANT means pause for 64 octets, i.e., a single Quant spans 8 clock cycles.

Figure 30: Faulty Link Condition during Pause Mode with RX_QUANT

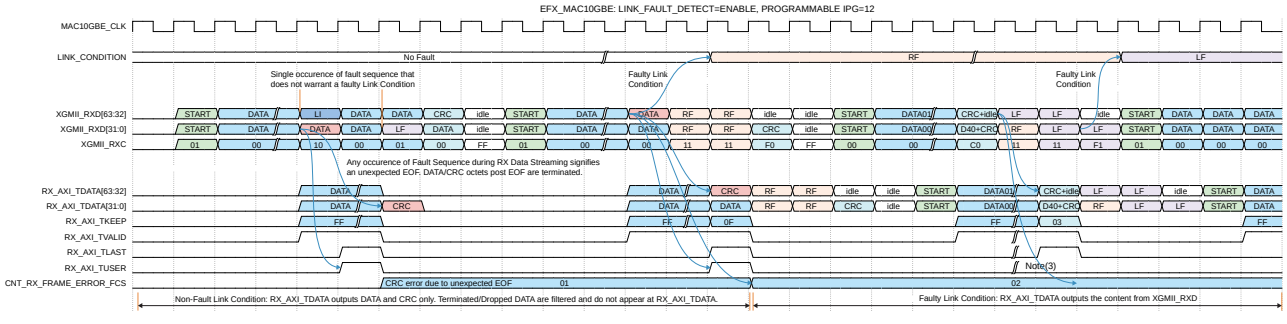


Note: (1) LF=Local Fault, RF=Remote Fault, LI=Link Interrupt, START=PREAMBLE+SFD
(2) Latencies among the 5 interfaces - TX_PAUSE_GEN, TX_AXI signals, XGMII_TX signals, LINK_FAULT_CONDITION, and XGMII_RX signals are not accurately illustrated here.

Figure 30: Faulty Link Condition during Pause Mode with RX_QUANT on page 37 illustrates the behavior of the signals at RX AXI ST interface during faulty and non-faulty link condition. During faulty link condition, RX_AXI_TDATA outputs the content from XGMII_RXD, which is inclusive of Local Fault sequence, Remote Fault sequence, Link Interrupt sequence, Idle octet, PREAMBLE, Data and CRC. Upon exiting a faulty link condition and adopting non-fault link condition, AXI RX ST outputs only data and CRC. During a faulty link condition, XGMII_RXD = IDLE appears as 07 at RX_AXI_TDATA. When there is no faulty link condition, XGMII_RXD = IDLE translates to 00 at RX_AXI_TDATA.

An occurence of any fault sequence may not necessarily warrant a faulty link condition. If there is any single occurrence of fault sequence at XGMII_RXD during the ongoing streaming RX data, i.e., between PREAMBLE and EOF (EOF is denoted by TERMINATE octet), it signifies an unexpected end of frame. In this scenario, RX_AXI_TUSER asserts due to CRC mismatch. Other statistics such as cnt_rx_frame_undersized, cnt_rx_frame_oversized, and cnt_rx_frame_mismatched_length may increment too, depending on the actual value of the frame length when the link fault condition occurs. The behavior of RX_AXI_TKEEP, RX_AXI_TLAST, and RX_AXI_TVALID signals remain the same even if there is a faulty link condition. Refer to **Figure 31: RX AXI Interface during Faulty Link Condition** on page 37.

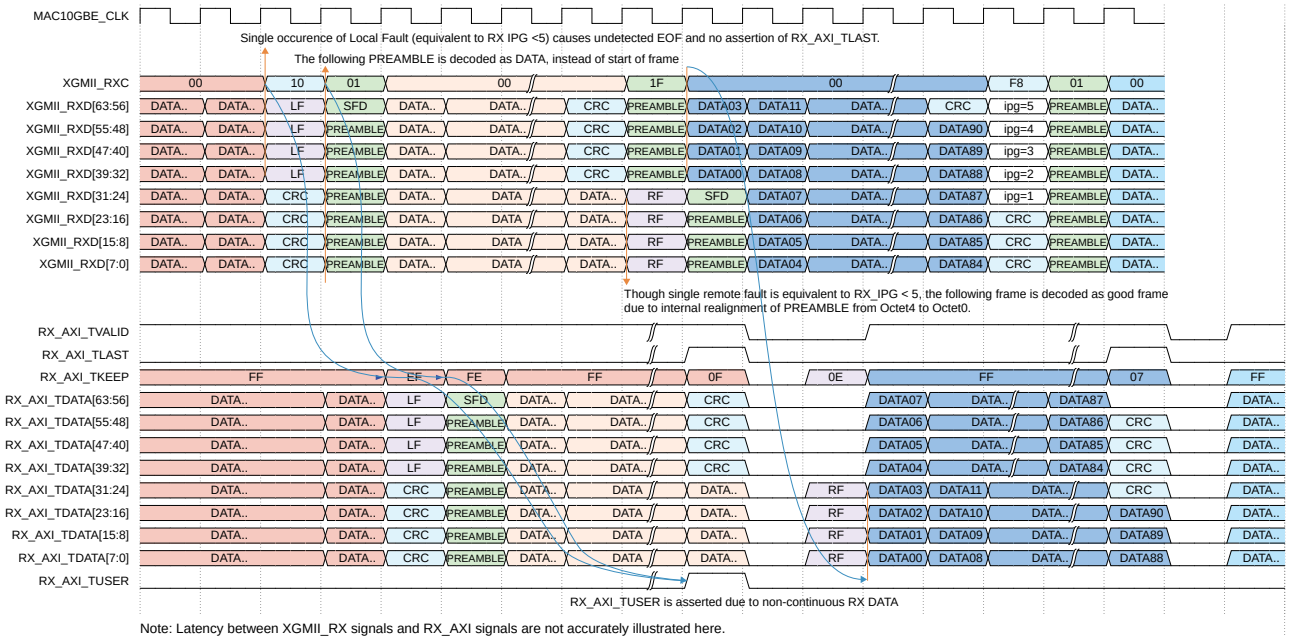
Figure 31: RX AXI Interface during Faulty Link Condition



Note: (1) LF=LOCAL FAULT, RF=REMOTE FAULT, LI=LINK INTERRUPT, START=PREAMBLE+SFD
(2) Latencies among the 3 interfaces - LINK_CONDITION, XGMII_TX signals and RX_AXI signals are not accurately illustrated here.
(3) Legit EOF because fault sequence RF is after CRC. Hence, no CRC error. TUSER is not asserted because there is no CRC error.

A single occurrence of Link Fault sequence occupies 4 octets. In the event where the singular Link Fault sequence occurs at the end of frame (EOF) and followed immediately by PREAMBLE, as illustrated in **Figure 32: Single Occurrence of Link Fault Causing Incorrect Decoding of XGMII RX Frames** on page 38, it is equivalent to RX_IPG < 5.

Figure 32: Single Occurrence of Link Fault Causing Incorrect Decoding of XGMII RX Frames



Note: Latency between XGMII_RX signals and RX_AXI signals are not accurately illustrated here.

Statistic Reporting

The Ethernet 10G MAC core supports statistic reporting for TX and RX. Each statistic is a 32-bit bus width, except for `rpt_rx_frame`, which is 14-bit width.

Any statistic registers, upon reaching the ceiling, stays at 32'hFFFF_FFFF. When this occurs, you need to reset all statistic registers by asserting `cnt_rst_n`. Upon reset, all statistic registers become `0`.

Note: In the statistic related figures, the statistic reporting is in hexadecimal, not decimal.

The following registers are the statistic reporting for TX:

- `cnt_tx_frame_transmitted_good`
- `cnt_tx_frame_pause_mac_ctrl`
- `cnt_tx_frame_is_fe`
- `cnt_tx_frame_error_txfifo_overflow`

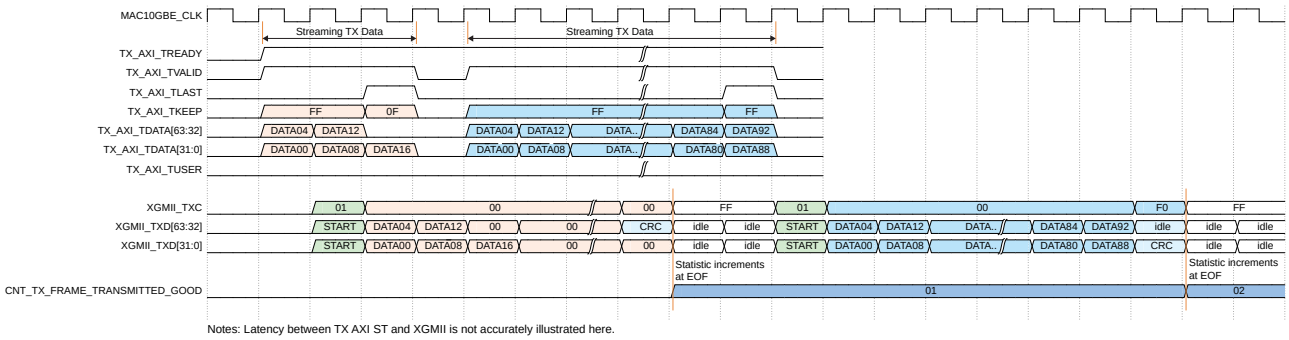
The following registers are the statistic reporting for RX:

- `rpt_rx_frame_length`
- `cnt_rx_frame_undersized`
- `cnt_rx_frame_oversized`
- `cnt_rx_frame_mismatched_length`
- `cnt_rx_frame_error_fcs`
- `cnt_rx_frame_filtered_by_address`
- `cnt_rx_frame_errors`
- `cnt_rx_frame_pause_mac_ctrl`
- `cnt_rx_frame_received_good`
- `cnt_rx_frame_received_total`

cnt_tx_frame_transmitted_good

The statistic `cnt_tx_frame_transmitted_good` reports the number of TX frames successfully transferred at XGMII interface, i.e., streaming TX DATA frame with `TX_AXI_TUSER` = `0`. This statistic count includes short frames with auto padding and TX pause frames too. This statistic is incremental at a clock cycle after the end of frame.

Figure 33: TX Statistic – `cnt_tx_frame_transmitted_good`



cnt_tx_frame_pause_mac_ctrl

The statistic `cnt_tx_frame_pause_mac_ctrl` reports the number of TX Pause frames successfully transferred from `tx_pause_gen` to XGMII Interface. TX pause frame is described in **Send Pause Frame at TX** on page 32. This statistic is incremental at a clock cycle after the end of frame. Refer to **Figure 26: Transmitting Pause Frame and Corresponding Statistic Reporting Increments** on page 32.

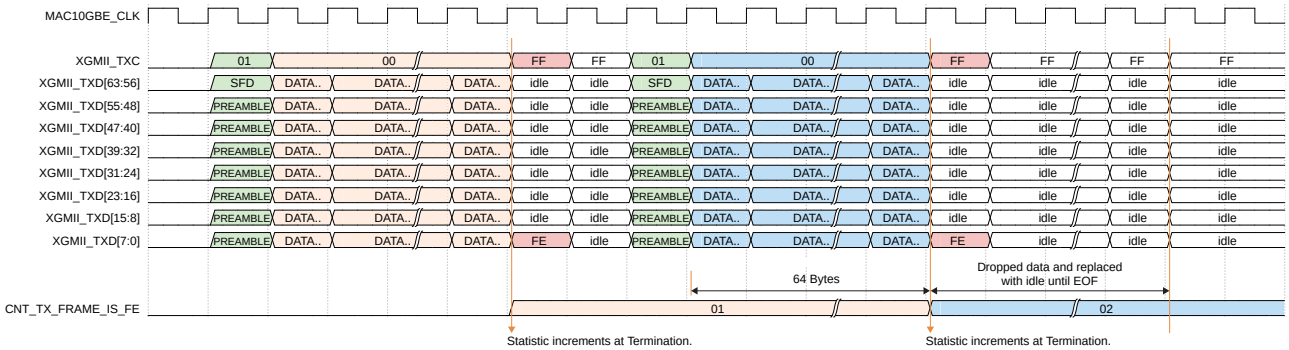
cnt_tx_frame_is_fe

The statistic `cnt_tx_frame_is_fe` reports the number of bad TX frames being terminated. This statistic does not include the following:

- The silently dropped TX frames due to non-compliant `TX_AXI_TLAST`.
- The scrapped TX frames due to `TXFIFO_Overflow` during **Store Forward** mode.

Bad TX frames criteria are described in **Assertion of TX_AXI_TUSER** on page 24, **Non-Streaming TX Data** on page 25, and **Dropped TX_AXI_TVALID Before End of Frame** on page 26. This statistic is incremental at termination as shown in **Figure 34: TX Statistic – cnt_tx_frame_is_fe** on page 40.

Figure 34: TX Statistic – `cnt_tx_frame_is_fe`

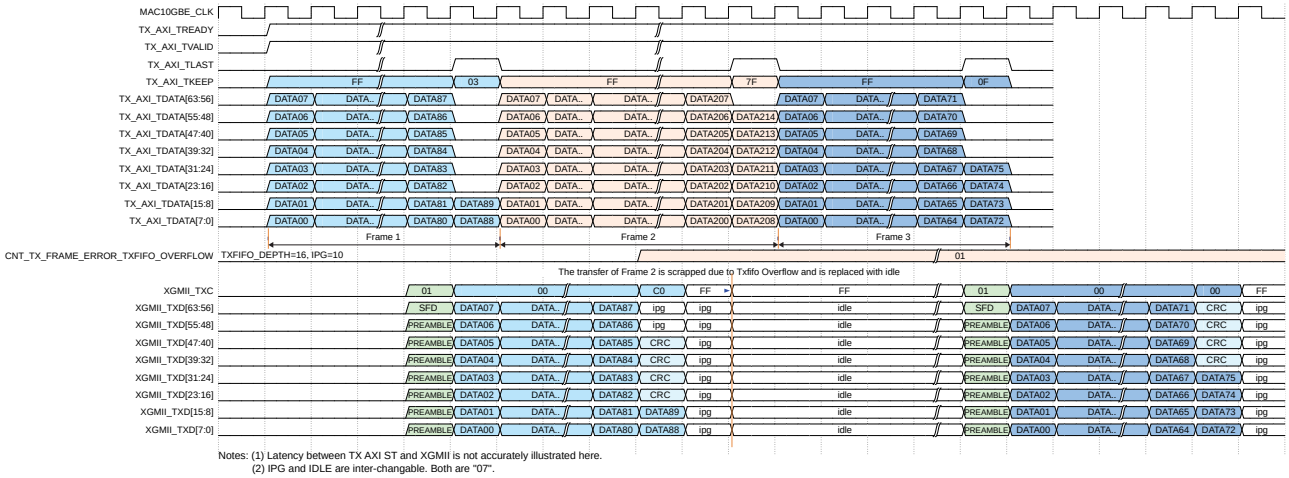


cnt_tx_frame_error_txfifo_overflow

The statistic `cnt_tx_frame_error_txfifo_overflow` reports the number of TX frames being scrapped due to TX FIFO overflow, during **Store Forward** mode. Refer to **TX FIFO Overflow** on page 26. This statistic is incremental whenever there is an overflow event.

During **Cut Through** mode, this statistic stays `0`.

Figure 35: TX Statistic – `cnt_tx_frame_error_txfifo_overflow`



rpt_rx_frame_length

The statistic `rpt_rx_frame_length` reports the length of each RX frames received at the RX AXI ST Interface. Frame length is inclusive of the headers (6 octets of destination address, 6 octets of source address, the optional 4 octets of VLAN tag, and 2 octets of ETHERTYPE/LEN), payload, and 4 octets of FCS. This statistic register updates at the same clock cycle as the end of frame. Refer to the figures, where statistic `rpt_rx_frame_length` is illustrated.

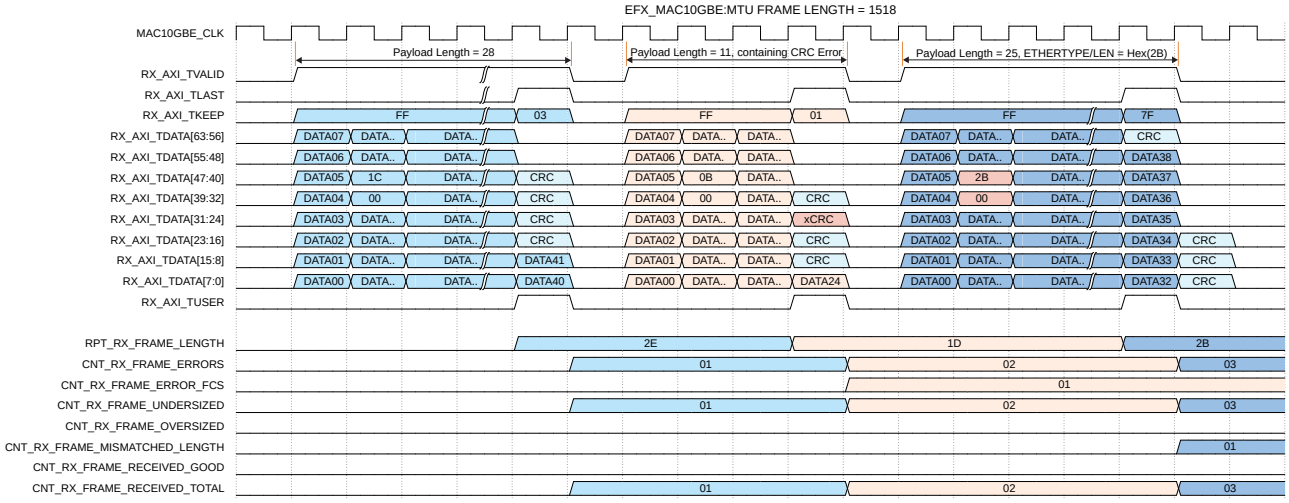
The RX frames that are filtered away due to broadcast, unicast, and multicast filtering do not appear at the RX AXI ST. Hence, this statistic does not report the frame length for the dropped RX frames.

cnt_rx_frame_undersized

The statistic `cnt_rx_frame_length` reports the number of RX frames with payload size < 46 for the non-VLAN tagged frames, or < 42 for VLAN tagged frames. This statistic register updates at a clock cycle after the end of frame. **Figure 36: RX Statistic – Undersized RX Frame** on page 42 shows the `cnt_rx_frame_undersized` statistic updates relative to the RX frames at AXI ST interface.

Note: The `RX_AXI_TUSER` is asserted to `1` as described in **Undersized RX Frame** on page 27.

Figure 36: RX Statistic – Undersized RX Frame

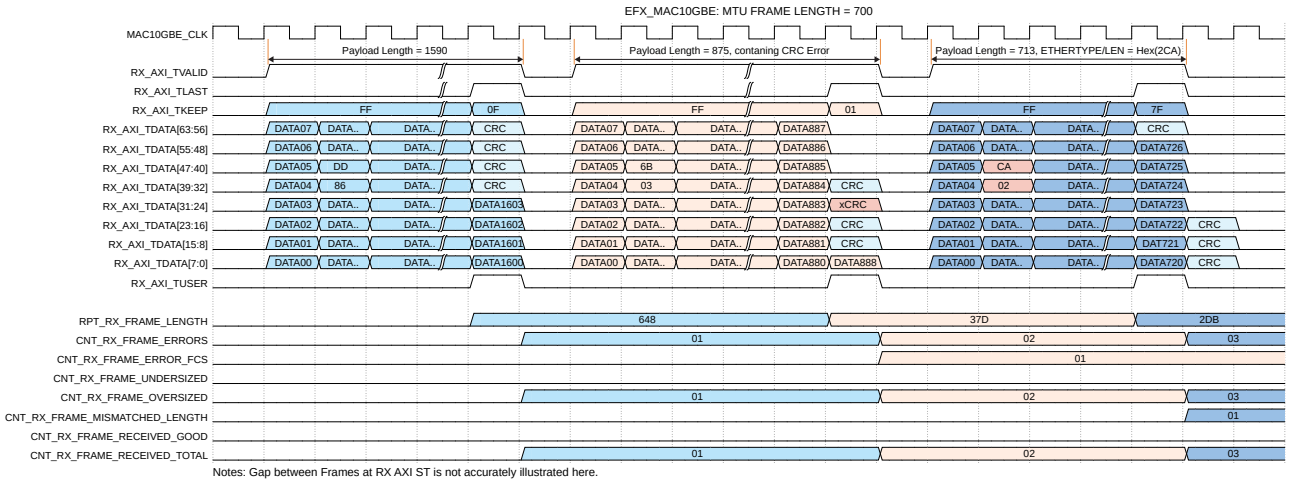


Notes: Gap between Frames at RX AXI ST is not accurately illustrated here.

cnt_rx_frame_oversized

The statistic `cnt_rx_frame_oversized` reports the number of RX frames with frame length > MTU frame length. This statistic register updates at a clock cycle after the end of frame. The RX_AXI_TUSER is asserted to 1 as described in **Oversized RX Frame** on page 27. The **Figure 37: RX Statistic – Oversized RX Frame** on page 42 shows the `cnt_rx_frame_oversized` statistic updates relative to the RX frames at AXI ST interface.

Figure 37: RX Statistic – Oversized RX Frame

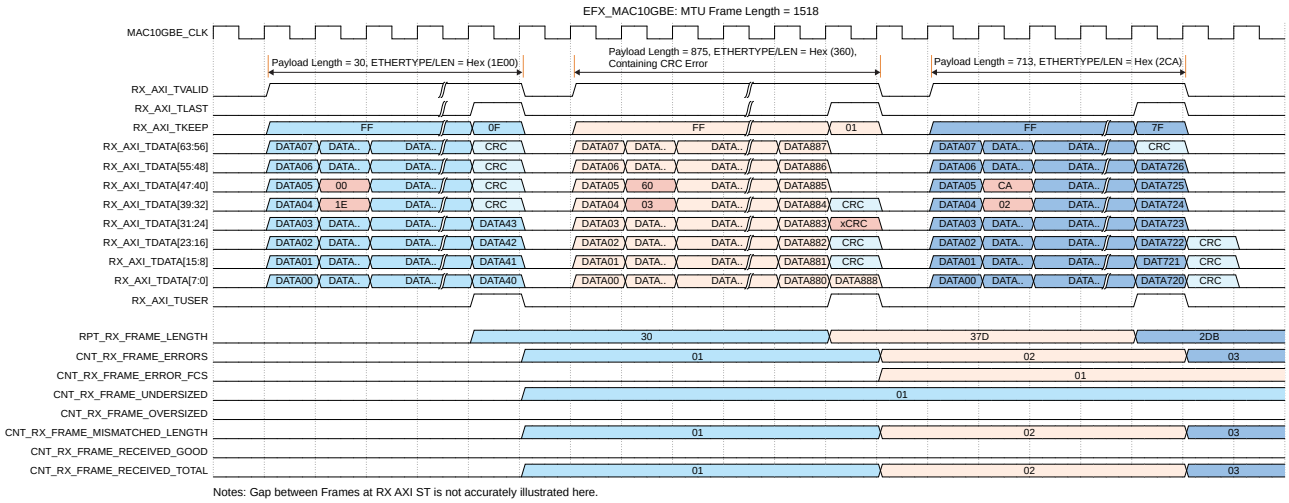


Notes: Gap between Frames at RX AXI ST is not accurately illustrated here.

cnt_rx_frame_mismatched_length

The statistic `cnt_rx_frame_mismatched_length` reports the number of RX frames where the physical frame length mismatches the value in the ETHERTYPE/LEN field. This statistic register updates at a clock cycle after the end of frame. The RX_AXI_TUSER is asserted to 1 as described in **Mismatched Length RX Frame** on page 27. The **Figure 38: RX Statistic – Mismatched Length RX Frame** on page 43 shows the `cnt_rx_frame_mismatched_length` statistic updates relative to the RX frames at AXI ST interface.

Figure 38: RX Statistic – Mismatched Length RX Frame

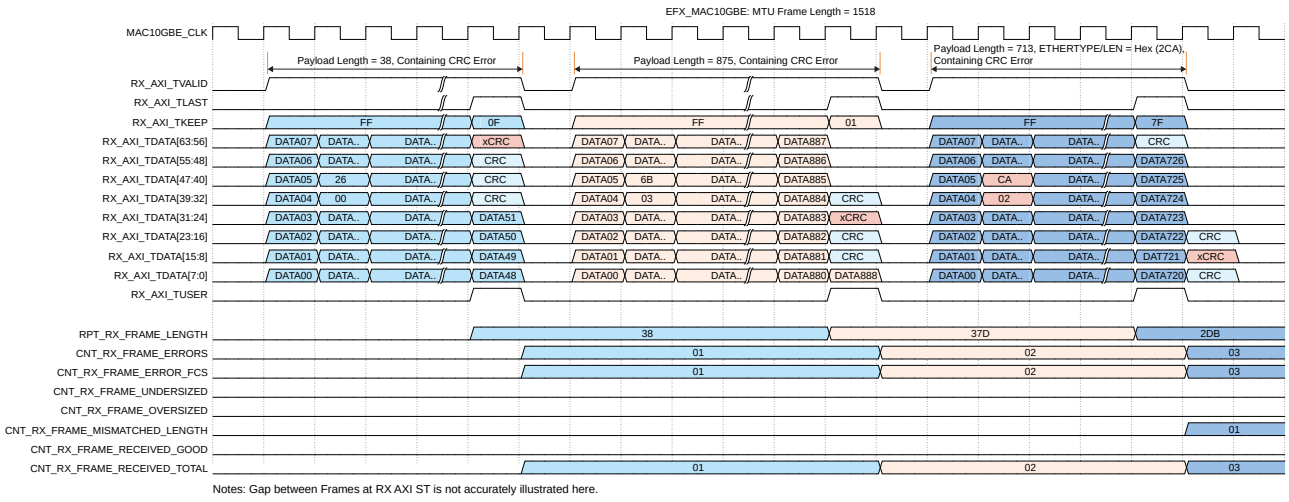


Notes: Gap between Frames at RX AXI ST is not accurately illustrated here.

cnt_rx_frame_error_fcs

The statistic `cnt_rx_frame_error_fcs` reports the number of RX frames with CRC error. This statistic register updates at a clock cycle after the end of frame. The `RX_AXI_TUSER` is asserted to `1` as described in **Frame Check Sequence (FCS) Error** on page 27. **Figure 39: RX Statistic – RX Frame with FCS Error** on page 43 shows the `cnt_rx_frame_error_fcs` statistic updates relative to the RX frames at AXI ST interface.

Figure 39: RX Statistic – RX Frame with FCS Error



Notes: Gap between Frames at RX AXI ST is not accurately illustrated here.

cnt_rx_frame_filtered_by_address

The statistic `cnt_rx_frame_filtered_by_address` reports the number of RX frames being dropped due to broadcast and address filtering. For details, refer to **Address Filtering and Broadcast Filtering at RX** on page 29.

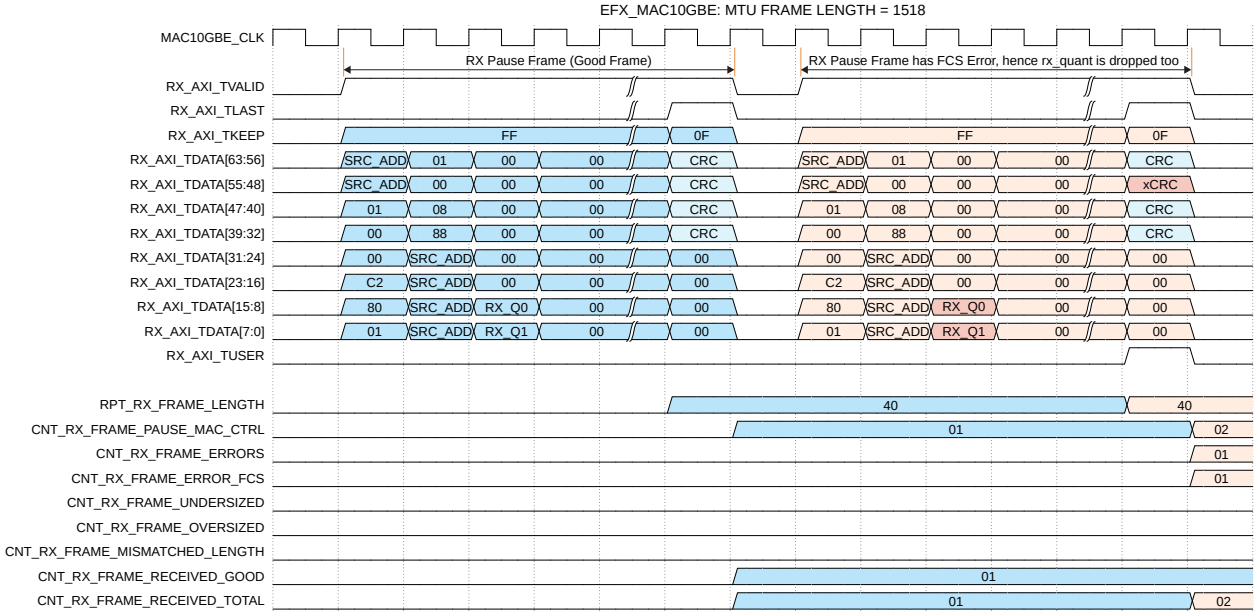
The statistic reporting is illustrated in

- **Figure 23: Silent Drop of Broadcast RX Frame** on page 31
- **Figure 24: Silent Drop of Broadcast RX Frame during Unicast Address Filtering** on page 31
- **Figure 25: Multicast Address Filtering** on page 31

cnt_rx_frame_pause_mac_ctrl

The statistic `cnt_rx_frame_pause_mac_ctrl` reports the number of RX frames decoded as Pause frame as shown in **Figure 40: RX Statistic – RX Pause Frame** on page 44.

Figure 40: RX Statistic – RX Pause Frame



Notes: (1) SRC_ADD=SOURCE ADDRESS, RXQ_Q0=RX_QUANT0, RXQ_Q1=RX_QUANT1.
(2) Gaps between Frames at RX AXI ST is not accurately illustrated here.

cnt_rx_frame_errors

The statistic `cnt_rx_frame_errors` reports the number of RX frames with the error conditions as follow:

- Undersized RX frame
- Oversized RX frame
- RX frame length mismatch
- FCS error
- Non-Streaming RX data frame
- Error frame
- Dropped RX frame due to broadcast and address filtering

This statistic register updates at a clock cycle after the end of frame.

Note: For non-dropped RX frames, `RX_AXI_TUSER` is asserted to `1` as described in **Erroneous RX Frame** on page 27.

Figure 41: RX Statistic Reporting – Non-Streaming RX Data Frame on page 45 shows the `cnt_rx_frame_errors` statistic updates due to non-streaming RX data frame. As described in **Non-Streaming RX Data Frame** on page 28, when the RX data frame is non-streaming, `RX_AXI_TUSER` asserts to 1 at EOF. The statistic `cnt_rx_frame_errors` also increments a clock cycle after EOF.

Figure 41: RX Statistic Reporting – Non-Streaming RX Data Frame

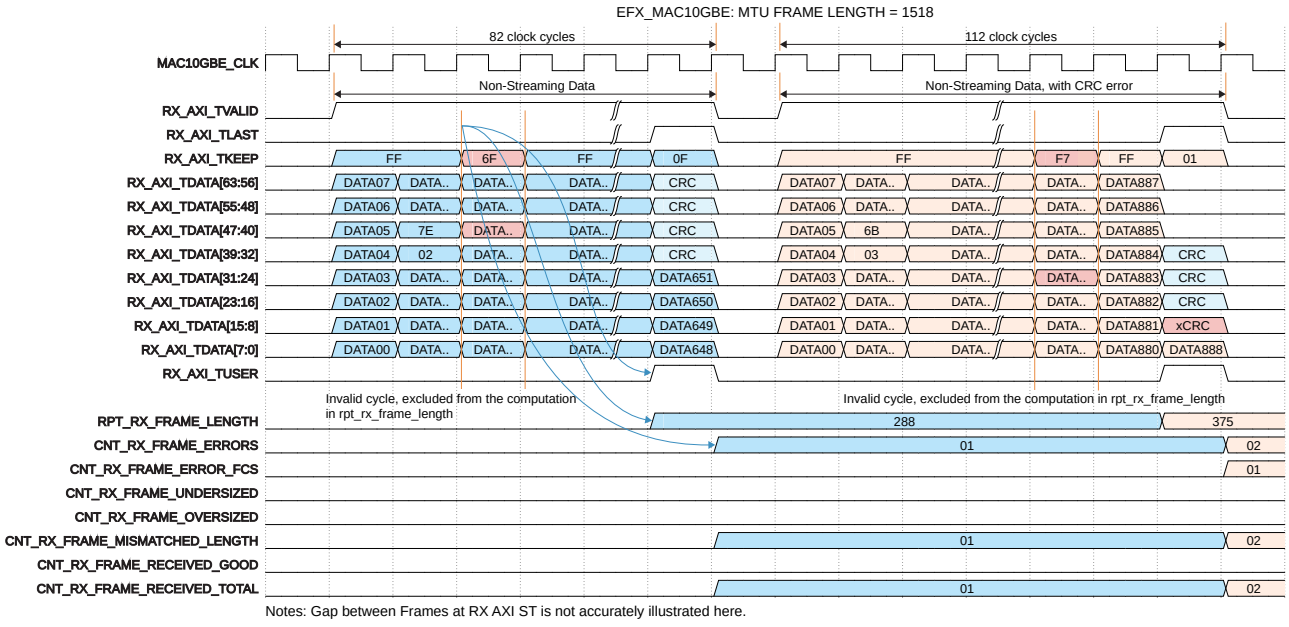
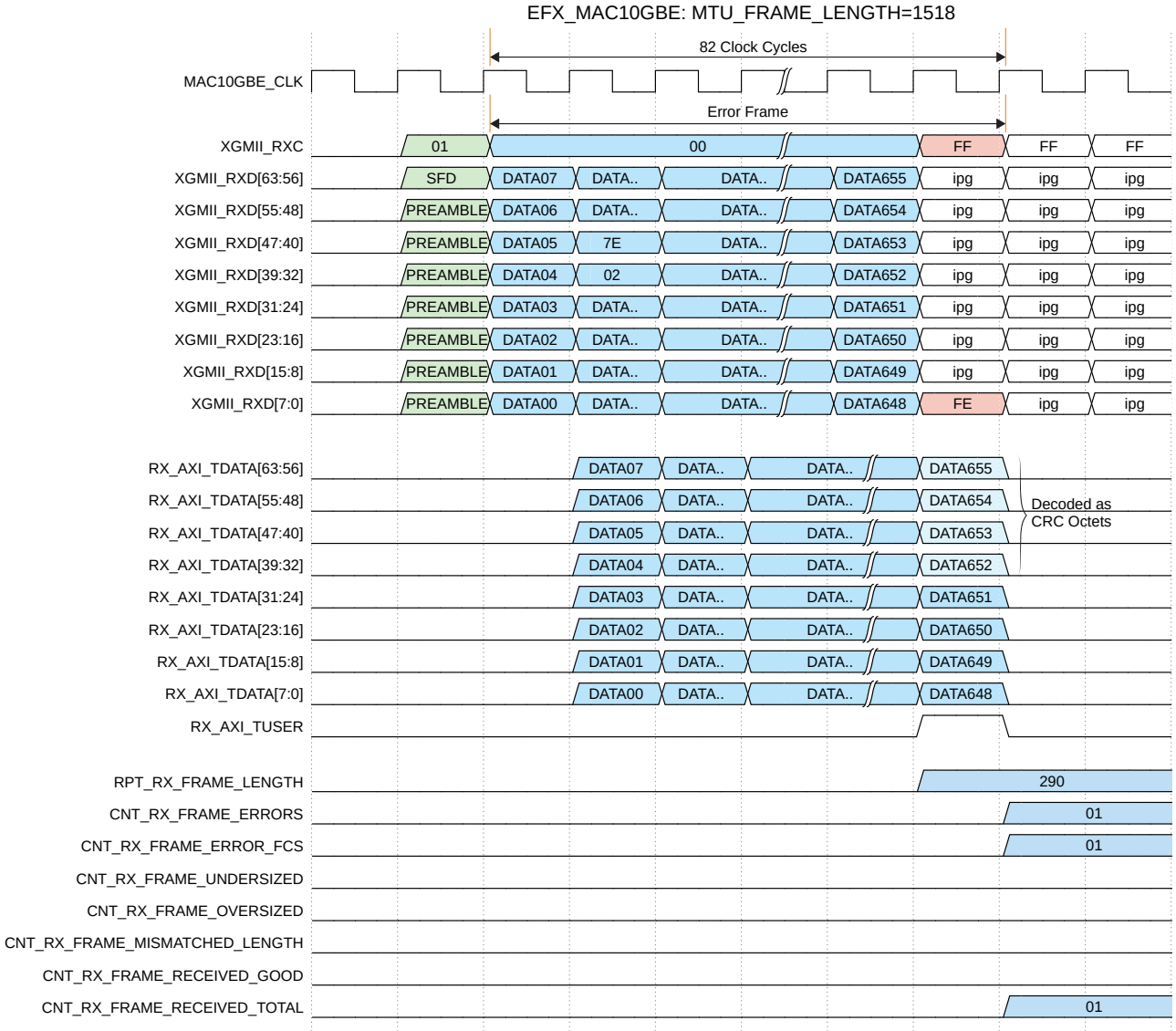


Figure 42: RX Statistic Reporting – Error Frame



Notes: Latency between XGMII RX and RX AXI ST is not accurately illustrated here.

cnt_rx_frame_received_good

The statistic `cnt_rx_frame_received_good` reports the number of error-free RX frames at the RX AXI ST interface. An error-free RX frame has the following characteristics:

- Streaming RX data frame or Pause frame.
- Data content matches the corresponding CRC.
- Destination address matches with MAC source address, with reference to `rx_address_filtering_mask` and the enabling of broadcast filtering.
- Payload size is ≥ 46 octets, i.e., frame length is ≥ 64 for non-VLAN tagged and ≥ 68 for VLAN tagged frames.
- Frame length is ≤ MTU frame length.
- For payload size ≤ 1500 octets, the physical frame length matches the values in the ETHERTYPE/LEN field.

cnt_rx_frame_received_total

The statistic `cnt_rx_frame_received_total` reports the total number of RX Ethernet frames at RX. This statistic includes the dropped RX frames due to broadcast and address filtering.

Latency

In the 2025.1 release and later, the latency of the Ethernet 10G MAC core has been improved. During **Cut Through** mode, the new latency of the TX frame transmission is 3 clock cycles, while the RX frame transmission has a new latency of 5 or 6 clock cycles.

At the RX path, there is a new configuration named **Optimize Timing**. The setting for this configuration is defaulted to **Enable**. When set to **Enable**, the 1st level of pipeline registers are synthesized to improve timing closure and the total latency in RX path is 6 clock cycles. Alternatively, depending on the complexity of your design and timing report, you can set this configuration to **Disable**. When set to **Disable**, there will be no pipeline registers being synthesized and the total latency in RX path is reduced to `5`.

Efinity: IP Catalog, Interface Designer, and Integration

The Ethernet 10G MAC core is designed to interact with the user's logic and the FPGA Ethernet 10G PCS as shown in **Figure 1: Ethernet 10G MAC Core Interaction with User's Logic and the FPGA Ethernet 10G PCS** on page 4. To combine the Ethernet 10G MAC core, the FPGA Ethernet 10G PCS and user's logic together as a whole design, you need to perform the following steps:

1. Configure and generate the Ethernet 10G MAC core from the IP Catalog.
2. Configure and generate the FPGA Ethernet 10G PCS using the Interface Designer.
3. Create a top-level module to integrate the Ethernet 10G MAC core (from step 1), the FPGA Ethernet 10G PCS (from step 2) and user's logic.

IP Catalog

The Ethernet 10G MAC core is ready to use from IP Catalog. For steps to customize an IP core with the IP Configuration wizard, refer to **Generating the Ethernet 10G MAC Core with the IP Manager** on page 50.

Interface Designer

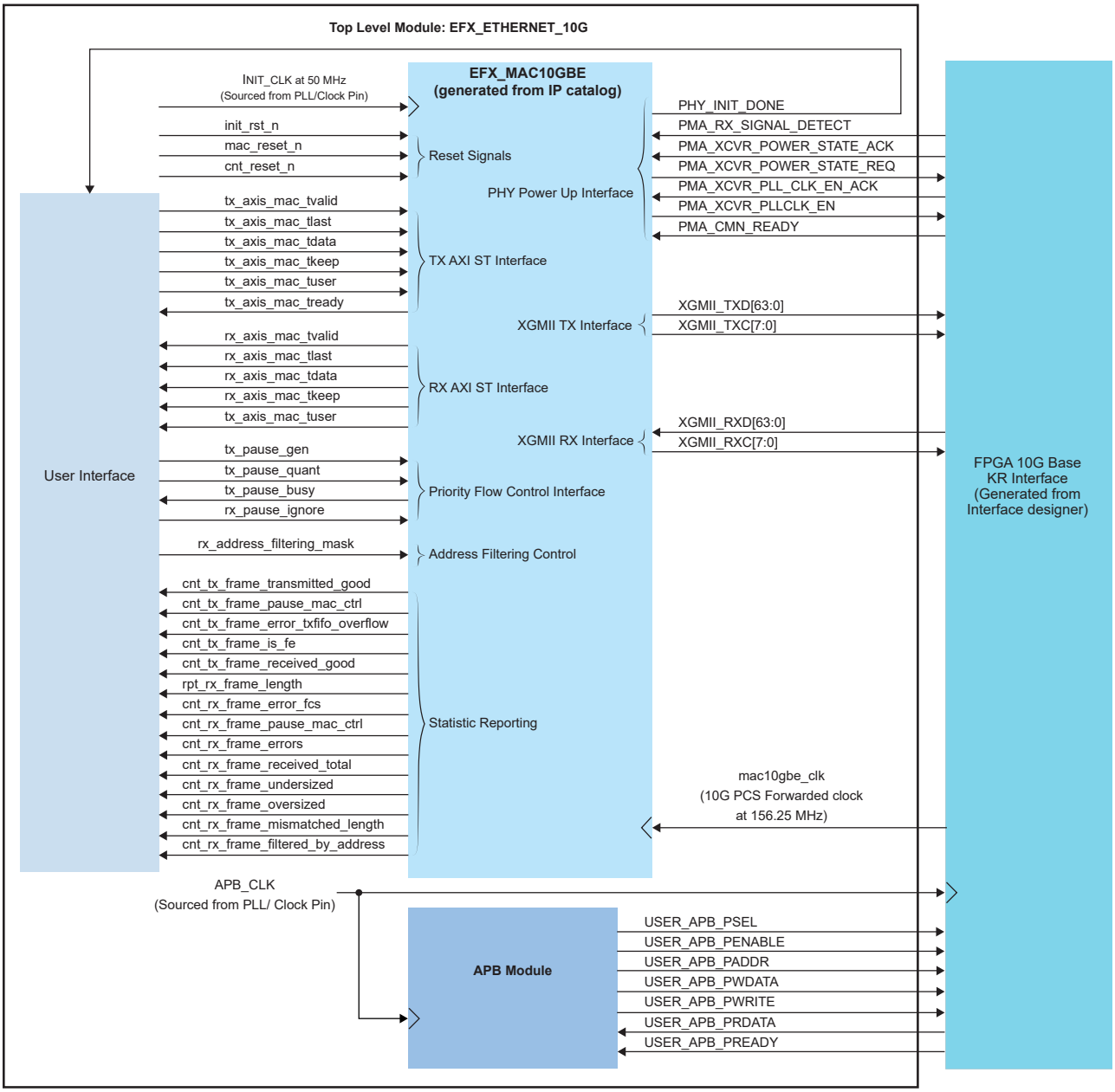
Refer to *Chapter 16 10Gbase-KR Interface* of TJ-Series Interfaces User Guide.

Top Level Module

You need to create a top-level module to integrate the Ethernet 10G MAC core (generated from the IP Catalog) and the FPGA 10G PCS (generated from the Interface Designer).

Figure 43: Top-Level Module for Efinity Compilation on page 49 illustrates the integration of one lane of Ethernet 10G MAC and PCS for Efinity compilation.

Figure 43: Top-Level Module for Efinity Compilation



IP Manager

The Efinity® IP Manager is an interactive wizard that helps you customize and generate Elitestek® 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 Elitestek 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 Elitestek IP cores include an example design or a testbench.

Generating the Ethernet 10G 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** > **10G 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 Ethernet 10G MAC** on page 51 section.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an Elitestek® 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.
- **efx_ethernet_10g_exp**—Has generated RTL, example design, and Efinity® project targeting a specific development board.
- **Testbench**—Contains simulation models. Testbench is not available.

Customizing the Ethernet 10G 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 5: Ethernet 10G MAC Core Parameters

Name	Options	Description
Data Streaming Mode	Cut Through, Store Forward	Default: **Cut Through** In **Cut Through** mode, the XGMII packet is transferred with minimum latency, without waiting for the complete packet. In **Store Forward** mode, the XGMII packet is only transferred when the entire packet is fully stored.
Programmable Inter packet Gap	9 - 15	Default: 12 In **Store Forward** mode, the effective IPG may be governed by the **Store Forward** mechanism, where the packet is only transferred when the entire packet is fully stored.
Maximum Transmission Unit Frame Length	64 - 16018	Default: 1518 **Maximum Transmission Unit Frame Length** is the total count of headers (destination address, source address, VLAN tag, type/length), data payload and frame check sequence. Refer to `frame length` depicted in **Figure 6: 802.3 Ethernet Packet and Frame Structure for Non-VLAN Tagged** on page 11 and **Figure 7: 802.3 Ethernet Packet and Frame Structure for VLAN Tagged** on page 11. Any RX packet with physical length > **Maximum Transmission Unit Frame Length** will be flagged as `oversized frame`. Any RX packet with payload < 46 (or < 42 for VLAN tagged frames) will also with `undersized frame`.
MAC Source Address	–	Default: 48'h0000_0000_0000 This **MAC Source Address** is used in the following scenarios: • When generating TX pause frame. • When comparing against address filtering. This **MAC Source Address** is not used when converting from User TX AXI ST inputs to XGMII TX packets.
Broadcast Filtering	Enable, Disable	Default: Enable **Broadcast Filtering**. Disable **Broadcast Filtering**.
TXFIFO Depth	8 – 2048	Default: 512 Set the depth of FIFO for storing the Full TX packet during **Store Forward** mode. The depth of FIFO must be large enough to store the maximum length of the TX frames intended for transfer. Refer to **Store Forward Mode** on page 20. This parameter is not effective during **Cut Through** mode.
Link Fault Detection	Enable, Disable	Default: Enable **Link Fault Detection**. Disable **Link Fault Detection**.
Optimize Timing	Enable, Disable	Default: Enable **Optimize Timing**. This enables the synthesis of pipeline registers to improve on timing. Disable **Optimize Timing**. No pipeline registers is being synthesized. You can disable if the mac10gbe_clk in your design is meeting the targeted 156.25 MHz.

Name	Options	Description
TX_PKT_CNT_DW	–	Default: Display in post IP generation. In **Store Forward** mode, the XGMII packet is only transferred when the entire packet is fully stored. `TX_PKT_CNT_DW` specifies the data width of the maximum number of frames that can be stored in the TXFIFO at one time, based on $2^{TX\_PKT\_CNT\_DW}-1$. This parameter is meant to optimize resource utilization. The default value is 3. • `TX_PKT_CNT_DW` = 1 → stores 1 frame • `TX_PKT_CNT_DW` = 2 → stores up to 3 frames • `TX_PKT_CNT_DW` = 3 → stores up to 7 frames • `TX_PKT_CNT_DW` = 4 → stores up to 15 frames • `TX_PKT_CNT_DW` = 5 → stores up to 31 frames • `TX_PKT_CNT_DW` = 6 → stores up to 63 frames • `TX_PKT_CNT_DW` = 7 → stores up to 127 frames • `TX_PKT_CNT_DW` = 8 → stores up to 255 frames • `TX_PKT_CNT_DW` = 9 → stores up to 511 frames • `TX_PKT_CNT_DW` = 10 → stores up to 1023 frames Elitestek recommends user to correlate both parameters `TXFIFO_DEPTH` and `TX_PKT_CNT_DW`, based on the TX data frame length, to optimize the resource utilization. For example, if the maximum and the minimum frame length is 64 and 16 respectively, you should set `TXFIFO_DEPTH` = `8` and `TX_PKT_CNT` = `3`. Alternatively, you may set `TXFIFO_DEPTH` = `16` and `TX_PKT_CNT` = `4`. • A `TXFIFO_DEPTH` = `8` can stores up to 4 frames of the smallest frame length (optimized resource). • A `TXFIFO_DEPTH` = `16` can stores up to 8 frames of the smallest frame length (this setting is not recommended because this adds to memory and counter resources). **Note:** • This parameter is not available in the IP Configuration Interface, but it is available in the post-generated IP. • The default setting of this parameter is `3`, mainly to optimize resources and to cater for the typical use case during **Store Forward** mode. • In the case where there is a need for the TXFIFO to store > 7 frames per time, you may manually change this setting in the following directories: – **ip/<ip_name>/<ip_name>.sv**. This is the location of the generated Ethernet 10G MAC Core. – **ip/<ip_name>/ efx_ethernet_10g_exp/ ip_name>.sv**. This is the location of the generated example design.

Ports

Table 6: Ethernet 10G MAC Core Ports Interface

Port Name	Direction	Bus Width	Clock Domain	Description
Clock and Reset[3]				
init_clk	Input	1	–	Frequency = 50 MHz or below
init_rst_n	Input	1	Async	Reset all power-up handshake between the core and the FPGA PHY.
mac10gbe_clk	Input	1	–	Frequency = 156.25 MHz
mac_reset_n	Input	1	Async	Reset all the core logic of the Ethernet 10G MAC core, which includes statistics reporting.
cnt_rst_n	Input	1	Async	Assert this port to reset all statistics reporting.
MAC AXI ST Interface: TX				
tx_axis_mac_tdata	Input	64	mac10gbe_clk	AXI ST 64 TX Interface. Refer to **TX AXI ST 64** on page 12.
tx_axis_mac_tvalid	Input	1	mac10gbe_clk	
tx_axis_mac_tlast	Input	1	mac10gbe_clk	
tx_axis_mac_tkeep	Input	8	mac10gbe_clk	
tx_axis_mac_tuser	Input	1	mac10gbe_clk	
tx_axis_mac_tready	Output	1	mac10gbe_clk	
MAC AXI ST Interface: RX				
rx_axis_mac_tdata	Output	64	mac10gbe_clk	AXI ST 64 RX Interface. Refer to **RX AXI ST 64** on page 14.
rx_axis_mac_tvalid	Output	1	mac10gbe_clk	
rx_axis_mac_tlast	Output	1	mac10gbe_clk	
rx_axis_mac_tkeep	Output	8	mac10gbe_clk	
rx_axis_mac_tuser	Output	1	mac10gbe_clk	
MAC Priority Flow Control[4]				
rx_pause_ignore	Input	1	mac10gbe_clk	Assert this signal to disable the priority flow control. This is a pseudo static user's input signal.
tx_pause_gen	Input	1	mac10gbe_clk	Assert this to trigger TX Pause request, i.e. to send TX Pause frame to XGMII TX interface, with the quant values from tx_pause_quant. This signal is pulse based, and should only be asserted when tx_pause_busy is 0.
tx_pause_busy	Output	1	mac10gbe_clk	An indicator of the status of TX Pause request. When this signal is 1, this means there is TX pause request being queued, pending processed. Any assertion on tx_pause_gen or tx_pause_quant will not be processed when this signal is 1. This signal will become 0 when the TX Pause request has been processed. When this signal is 0, it also means that user can request a new TX Pause request by asserting tx_pause_gen.
tx_pause_quant	Input	16	Pseudo Async	Drive this bus with the desired quant value to be included during the TX Pause request. 1 quant is defined as the pause time to transfer 512 bits. Quant value should be stable before asserting tx_pause_gen.
MAC Broadcast Filtering & Address Filtering[5]				

(3) For details on Clock and Reset, refer to **Clock Sources** on page 8 and **5.3 Reset Signals**.
(4) For details on MAC Priority Flow Control, refer to **Priority Flow Control (Duplex Mode)** on page 32.
(5) For details on MAC broadcast and address filtering, refer to **Address Filtering and Broadcast Filtering at RX** on page 29.

Port Name	Direction	Bus Width	Clock Domain	Description
rx_address_ filtering_mask	Input	48	Pseudo Async	This is a bit-wise filtering mask for address filtering. Any bit, when asserted, is compared bit-wise against source address and any mismatched comparison is filtered or dropped. When any of the bit-wise mask is set to 0, the corresponding bit is considered matched.
MAC Statistic Reporting: TX[6]				
cnt_tx_frame_ transmitted_good	Output	32	mac10gbe_clk	Reports the number of TX frames successfully transferred at XGMII Interface.
cnt_tx_frame_ pause_mac_ctrl	Output	32	mac10gbe_clk	Reports the number of TX Pause frames successfully transferred from tx_pause_gen to XGMII Interface.
cnt_tx_frame_error_ txfifo_overflow	Output	32	mac10gbe_clk	Reports the number of TX frames being scrapped due to TXFIFO overflow, during **Store Forward** mode. During **Cut Through** mode, this statistic stays `0`.
cnt_tx_frame_is_fe	Output	32	mac10gbe_clk	Reports the number of Bad TX frames being terminated. This statistic does not include the silently dropped TX frames due to non-compliant TX_AXI_TLAST or the scrapped TX frames due to TXFIFO Overflow.
MAC Statistic Reporting: RX[6]				
rpt_rx_frame_length	Output	14	mac10gbe_clk	Reports the length of each RX frames received at RX AXI ST Interface.
cnt_rx_frame_ received_good	Output	32	mac10gbe_clk	Reports the number of Good Frames at RX AXI ST.
cnt_rx_frame_ received_total	Output	32	mac10gbe_clk	Reports the number of all frames at RX, i.e. good frames, bad frames, and pause frames. This statistic includes dropped frames.
cnt_rx_frame_errors	Output	32	mac10gbe_clk	Reports the number bad frames and dropped frames at RX.
cnt_rx_frame_ pause_mac_ctrl	Output	32	mac10gbe_clk	Reports the number of pause frames at RX.
cnt_rx_frame_ error_fcs	Output	32	mac10gbe_clk	Reports the number of RX frames containing CRC error.
cnt_rx_frame_ undersized	Output	32	mac10gbe_clk	Reports the number of short frames at RX, i.e. payload size < 46.
cnt_rx_frame_ oversized	Output	32	mac10gbe_clk	Reports the number of RX frames with physical length > MTU_FRAME_LENGTH.
cnt_rx_frame_ mismatched_ length	Output	32	mac10gbe_clk	Reports the number of RX frames with physical length mismatching the ETHERTYPE/LEN field. This statistic is only applicable for RX frames with payload size ≤ 1500.
cnt_rx_frame_ filtered_by_ address	Output	32	mac10gbe_clk	Reports the number of RX frames being silently dropped due to address mismatch or RX frames with broadcast address. This statistic is subjected to the enabling of **Broadcast Filtering** and `rx_address_filtering_mask`.
XGMII Interface				
XGMII_TXD	Output	64	mac10gbe_clk	XGMII TX Data.
XGMII_TXC	Output	8	mac10gbe_clk	XGMII TX Control.
XGMII_RXD	Input	64	mac10gbe_clk	XGMII RX Data.
XGMII_RXC	Input	8	mac10gbe_clk	XGMII RX Control.
PHY Power Up Sequence				
PMA_CMN_READY	Input	1	Async	Output from PHY to indicate the readiness of PHY calibration.

[6] For details on MAC statistic reporting, refer to **Statistic Reporting** on page 39 .

Port Name	Direction	Bus Width	Clock Domain	Description
PMA_XCVR_ PLLCLK_ EN_ACK	Input	1	Async	Refer to **Power Up Handshake with FPGA Transceiver** on page 10.
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	

Ethernet 10G 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.

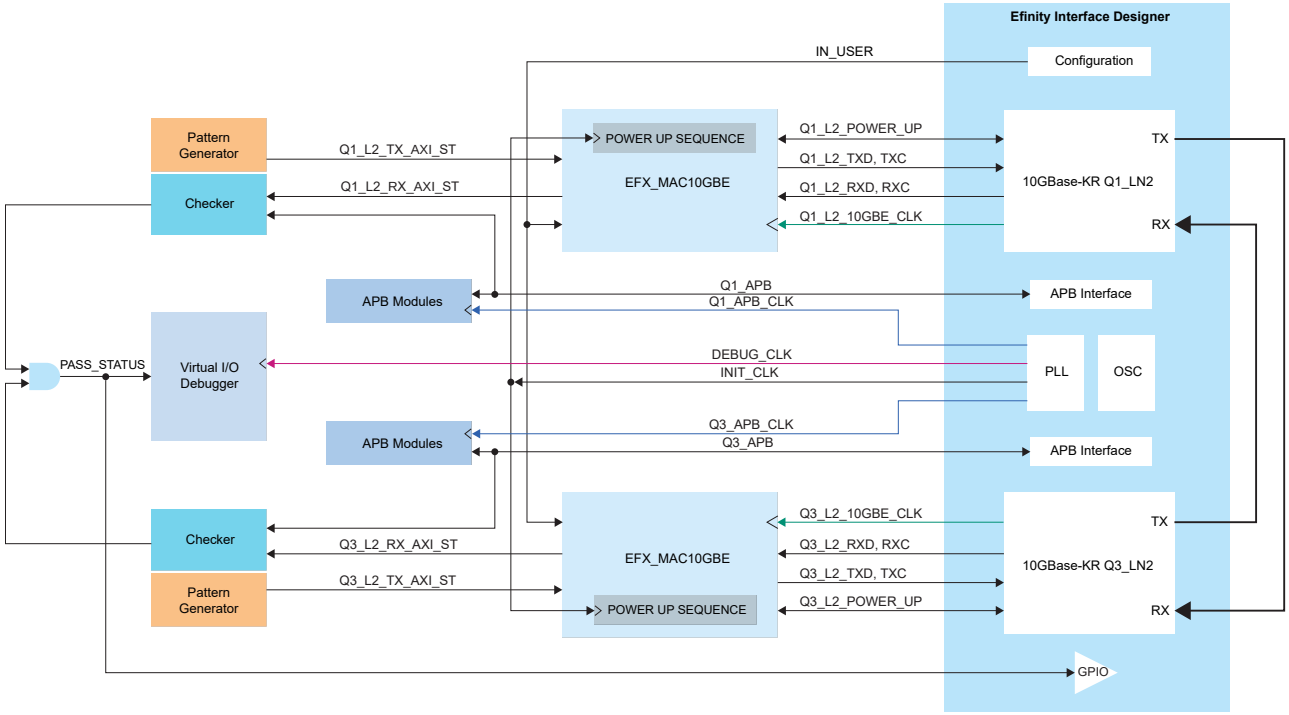
Important: Elitestek tested the example design generated with the default parameter options only.

Note: The Ethernet 10G MAC core example design is available in Efinity software version 2024.2.294.1.19 and later.

The example design demonstrates the connectivity of the Ethernet 10G MAC core, FPGA 10G PCS, and user's logic, as shown in **Figure 43: Top-Level Module for Efinity Compilation** on page 49 and **Figure 44: Example Design for Ethernet 10G MAC Core** on page 56. The user's logic has 2 pattern generators and checkers (denoted by `efx_mac10gbe_exp_pat_gen` and `efx_mac10gbe_exp_checker` modules respectively), APB modules (denoted by `efx_mac10gbe_exp_apb_master` and `efx_mac10gbe_exp_apb_halt`), and a Virtual I/O Debugger.

The example design further demonstrates the essential APB commands and handshakes to enable the RX path upon the CDR-Locked-to-Data, as described in **Power Up Handshake with FPGA Transceiver** on page 10. The APB master has built-in ROM and RAM.

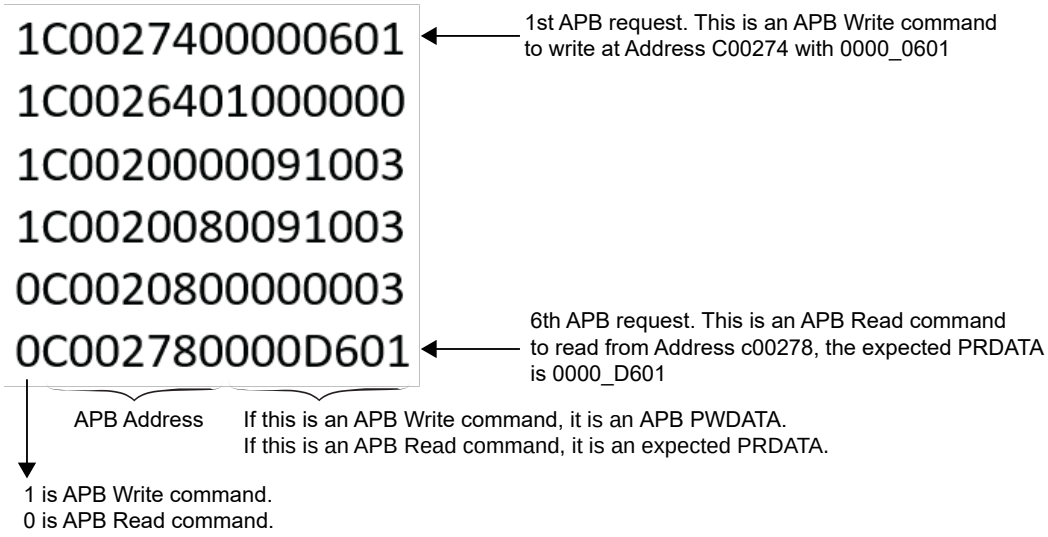
Figure 44: Example Design for Ethernet 10G MAC Core



The built-in ROM stores the configuration settings to configure and enable the RX path and/or Auto Negotiation (Clause 37). With **efx_q1_rom_mif_an.mem** as an example, the **Figure 45: Format of an APB ROM Content** on page 57 describes the format of the ROM content.

Concurrently, the built-in RAM captures the APB_PRDATA and APB_PADDRESS of all APB read commands. The ram_dout_d and ram_dout_a ports display the captured APB_PRDATA and the corresponding APB_PADDRESS.

Figure 45: Format of an APB ROM Content



In user mode, the example design awaits the PLL to lock. The assertion of PLL_LOCK releases the resets in the Ethernet 10G 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 start the APB configuration based on the built-in ROM content. At the end of the APB configuration, the RX path in the FPGA becomes functional and achieves BLOCK LOCK status. Upon achieving BLOCK LOCK status, the pattern generator starts transmitting data frames into the TX AXI ST interface of the Ethernet 10G MAC core.

For the entire operation, the checker actively checks the RX path of the Ethernet 10G MAC core for error frame and assertion of RX_AXI_TUSER. Depending on the macro enablement, the checker also checks for successful Auto Negotiation state and KR Training. Finally, the PASS_STATUS is asserted if all the passing criterias are met.

Example Design Macros

The example design includes useful macros that allow you to customize and enable the KR Training and Auto Negotiation (Clause 37) in the FPGA 10G PCS. Using the macros, you can plug in the example design into the **Functional Verification Testbench** to perform extensive functional verification. By default, the macros are not enabled.

The following are the available macros in the example design:

- `KR ENABLE`
  - Enable **KR Training** in the FPGA 10G PCS.
  - Pre-requisite: Enable **KR Training** in the Interface Designer.
  - Ineffective when the `LOOPBACK` macro is enabled.
- `AN ENABLE`
  - Enable **Auto Negotiation (Clause 37)** in the FPGA 10G PCS.
  - Pre-requisite: Enable **Auto Negotiation (Clause 37)** in the Interface Designer.
  - Ineffective when the `LOOPBACK` macro is enabled.
- `LOOPBACK`
  - Redirect the XGMII TX to the XGMII RX. Available for Q1_L2 instance only.
  - When enabling this `LOOPBACK` macro, the Ethernet 10G MAC core is segregated from the FPGA 10G PCS. Hence, the PCS features are not available, resulting in `KR_ENABLE` and `AN_ENABLE` become ineffective.
- `VERIF`
  - This macro provides the flexibility to migrate and plug in the example design into **Functional Verification Testbench**.
  - When this macro is enabled, you may use the driver in the **Functional Verification Testbench** to inject vectors into the Ethernet 10G MAC core. Simultaneously, you may use the scoreboard in the **Functional Verification Testbench** to check and verify the output from the Ethernet 10G MAC core.
  - The Virtual I/O Debugger is disconnected when this macro is enabled.
  - When this macro is not enabled, you can use the Virtual I/O Debugger to drive the input vectors into the Ethernet 10G MAC core.

Enabling the Macros in the Example Design

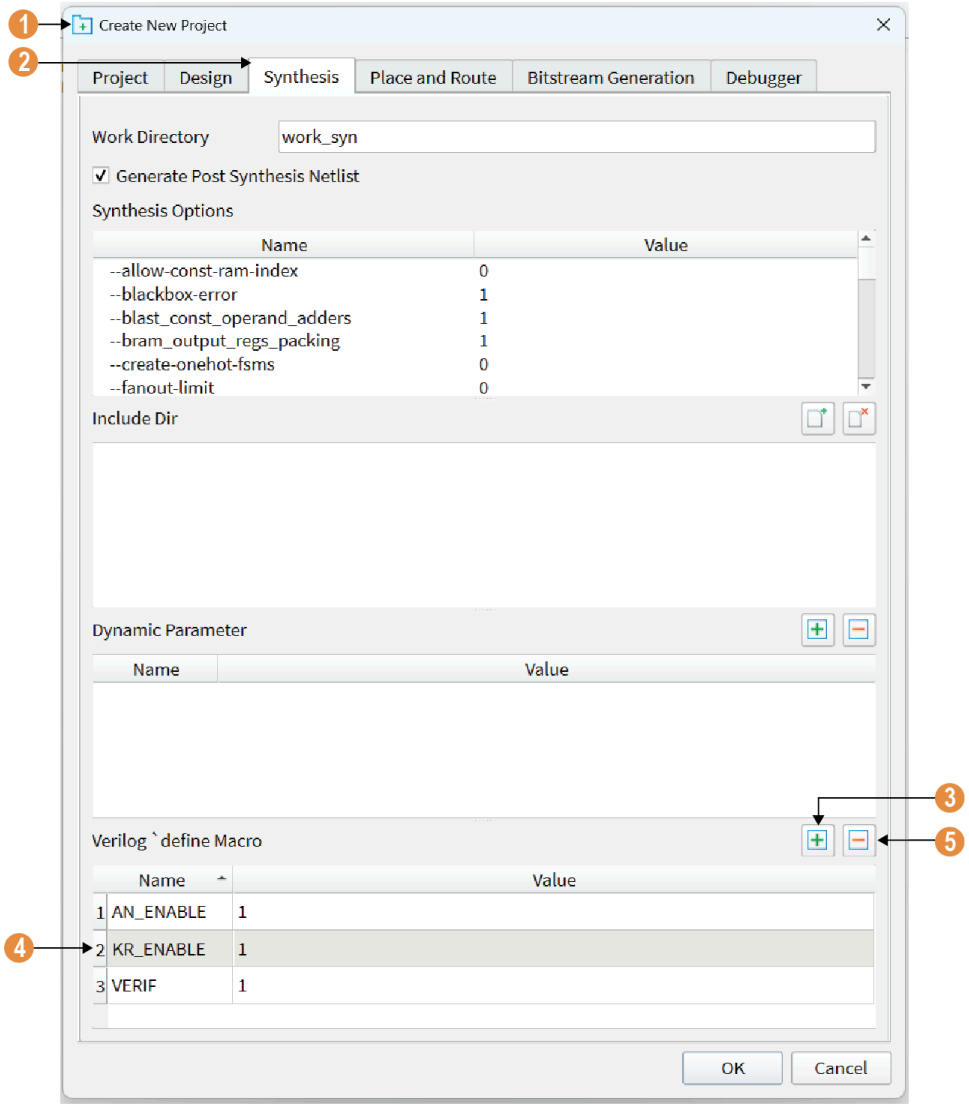
To enable macros for Efinity compilation.

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 46: 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 Ethernet 10G MAC core from the IP Catalog and set your configurations.
3. In the **Deliverables** tab, turn on the **Example Design** (**efx_ethernet_10g_exp**).
4. Click **Generate**.

The example design is available in **<project path>\ip\<ip-module-name>\efx_ethernet_10g_exp** directory.

To migrate the example design into the **Functional Verification Testbench**:

1. Close the existing project.
2. To open the example design, select **<project path>\ip\<ip-module-name>\efx_ethernet_10g_exp\efx_ethernet_10g_exp.xml**
3. Enable the `VERIF` macro. This step is mandatory.

Note: You may enable or disable any macro combination of the `KR_ENABLE`, `AN_ENABLE`, or `LOOPBACK`.

4. Compile the design.
5. In the **Functional Verification Testbench**, instantiate and connect the example design based on **Table 7: Example Design for Functional Verification Testbench** on page 61.
6. Connect to drive input vectors from the **Functional Verification Testbench** into the Ethernet 10G MAC core.
7. If you enable the `LOOPBACK` macro, you need to tie off all inputs of the APB modules.

The PHY power up sequence interface is still connected to the FPGA PHY. Hence, in user mode, the power-up sequence handshake takes place and the forwarded clock is *ON* to clock the Ethernet 10G MAC core.

Note: The `LOOPBACK` macro is only applicable to `Q1_L2` in the example design.

To load the example design into the FPGA development board:

1. Close the existing project.
2. To open the example design, select **<project path>\ip\<ip-module-name>\efx_ethernet_10g_exp\efx_ethernet_10g_exp.xml**
3. Disable the `VERIF` macro. This step is mandatory.

Note: You may enable any macro combination of `KR_ENABLE`, `AN_ENABLE`, or `LOOPBACK`.

4. Compile the design.
5. Download the design to your TJ-Series TJ375 N1156X Development Board.

Table 7: Example Design for Functional Verification Testbench

Signal Name	Direction	Width	Description
Common Clocks and Signals			
INIT_CLK	Input	1	Connect from the PLL_TR1 in FPGA peripheral.
Q1_APB_CLK	Input	1	
Q3_APB_CLK	Input	1	
PLL_LOCKED	Input	1	
IN_USER	Input	1	Connect from the FPGA peripheral.
Instance Q1 Lane 2: Per Lane Clock and Reset			
Q1_L2_10gbe_clk	Input	1	Connect with the FPGA transceiver's Q1 lane 2 interface.
Q1_L2_PCS_RST_N_RX	Output	1	
Q1_L2_PCS_RST_N_TX	Output	1	
Instance Q1 Lane 2: PCS Interface			
Q1_L2_BLOCK_LOCK	Input	1	Connect with the FPGA transceiver's Q1 lane 2 interface.
Q1_L2_PCS_STATUS	Input	1	
Q1_L2_IRQ	Input	1	
Q1_L2_HI_BER	Input	1	
Q1_L2_PHY_INTERRUPT	Input	1	
Q1_L2_PMA_TX_ELEC_IDLE	Output	1	
Q1_L2_ETH_EEE_ALERT_EN	Output	1	
Instance Q1 Lane 2: KR Training Interface			
Q1_L2_KR_TRAINING_ENABLE	Output	1	Connect with the FPGA transceiver's Q1 lane 2 interface.
Q1_L2_KR_RESTART_TRAINING	Output	1	
Q1_L2_KR_TRAINING	Input	1	
Q1_L2_KR_FRAME_LOCK	Input	1	
Q1_L2_KR_LOCAL_RX_TRAINED	Input	1	
Q1_L2_KR_SIGNAL_DETECT	Input	1	
Q1_L2_KR_TRAINING_FAILURE	Input	1	
Q1_L2_restart_kr_training_IN	Input	1	Connect from the drivers in the **Functional Verification Testbench**.
Instance Q1 Lane 2: PHY Power Up Sequence Interface			
Q1_PMA_CMN_READY	Input	1	Connect with the FPGA transceiver's Q1 lane 2 interface.
Q1_L2_PMA_XCVR_PLLCLK_EN	Output	1	
Q1_L2_PMA_XCVR_PLLCLK_EN_ACK	Input	1	
Q1_L2_PMA_XCVR_POWER_STATE_REQ	Output	4	
Q1_L2_PMA_XCVR_POWER_STATE_ACK	Input	4	
Q1_L2_PMA_RX_SIGNAL_DETECT	Input	1	
Instance Q1 Lane 2: XGMII Interface			
Q1_L2_TXD	Output	64	Connect with the FPGA transceiver's Q1 lane 2 interface.
Q1_L2_TXC	Output	8	
Q1_L2_RXD	Input	64	
Q1_L2_RXC	Input	8	

Signal Name	Direction	Width	Description
Instance Q1 Lane 2: Input to Ethernet 10G MAC Core			
in_Q1_L2_cnt_rst_n	Input	1	Connect from the drivers in the **Functional Verification Testbench**.
in_Q1_L2_rx_pause_ignore	Input	1	
in_Q1_L2_rx_address_filtering_mask	Input	48	
in_Q1_L2_tx_pause_gen	Input	1	
in_Q1_L2_tx_pause_quant	Input	16	
Instance Q1: APB Interface			
Q1_USER_APB_PSEL	Output	1	Connect with the FPGA transceiver's Q1 interface.
Q1_USER_APB_PWRITE	Output	1	
Q1_USER_APB_PENABLE	Output	1	
Q1_USER_APB_PADDR	Output	24	
Q1_USER_APB_PWDATA	Output	32	
Q1_USER_APB_PRDATA	Input	32	
Q1_USER_APB_PREADY	Input	1	
Q1_USER_APB_PSLVERR	Input	1	
Instance Q1: APB Modules			
in_Q1_ram_usr_wren_w	Input	1	Connect from the drivers in the **Functional Verification Testbench**, applicable only to debug the APB modules in the example design. For functional use case, initialize each signal to `0`.
in_Q1_ram_usr_addr_w	Input	6	
in_Q1_usr_apb_start_w	Input	1	
in_Q1_usr_apb_addr_w	Input	24	
in_Q1_usr_apb_write_w	Input	1	
in_Q1_usr_apb_pwdata_w	Input	32	
Instance Q3 Lane 2: Per Lane Clock and Reset			
Q3_L2_10gbe_clk	Input	1	Connect with the FPGA transceiver's Q3 lane 2 interface.
Q3_L2_PCS_RST_N_RX	Output	1	
Q3_L2_PCS_RST_N_TX	Output	1	
Instance Q3 Lane 2: PCS Interface			
Q3_L2_BLOCK_LOCK	Input	1	Connect with the FPGA transceiver's Q3 lane 2 interface.
Q3_L2_PCS_STATUS	Input	1	
Q3_L2_IRQ	Input	1	
Q3_L2_HI_BER	Input	1	
Q3_L2_PHY_INTERRUPT	Input	1	
Q3_L2_PMA_TX_ELEC_IDLE	Output	1	
Q3_L2_ETH_EEE_ALERT_EN	Output	1	
Instance Q3 Lane 2: KR Training Interface			
Q3_L2_KR_TRAINING_ENABLE	Output	1	Connect with the FPGA transceiver's Q3 lane 2 interface.
Q3_L2_KR_RESTART_TRAINING	Output	1	
Q3_L2_KR_TRAINING	Input	1	
Q3_L2_KR_FRAME_LOCK	Input	1	
Q3_L2_KR_LOCAL_RX_TRAINED	Input	1	
Q3_L2_KR_SIGNAL_DETECT	Input	1	
Q3_L2_KR_TRAINING_FAILURE	Input	1	
Q3_L2_restart_kr_training_IN	Input	1	Connect from the driver in the **Functional Verification Testbench**.

Signal Name	Direction	Width	Description
Instance Q3 Lane 2: PHY Power Up Sequence Interface			
Q3_PMA_CMN_READY	Input	1	Connect with the FPGA transceiver's Q3 lane 2 interface.
Q3_L2_PMA_XCVR_PLLCLK_EN	Output	1	
Q3_L2_PMA_XCVR_PLLCLK_EN_ACK	Input	1	
Q3_L2_PMA_XCVR_POWER_STATE_REQ	Output	4	
Q3_L2_PMA_XCVR_POWER_STATE_ACK	Input	4	
Q3_L2_PMA_RX_SIGNAL_DETECT	Input	1	
Instance Q3 Lane 2: XGMII Interface			
Q3_L2_TXD	Output	64	Connect with the FPGA transceiver's Q3 lane 2 interface.
Q3_L2_TXC	Output	8	
Q3_L2_RXD	Input	64	
Q3_L2_RXC	Input	8	
Instance Q3 Lane 2: Input to Ethernet 10G MAC Core			
in_Q3_L2_cnt_rst_n	Input	1	Connect from the drivers in the **Functional Verification Testbench.**
in_Q3_L2_rx_pause_ignore	Input	1	
in_Q3_L2_rx_address_filtering_mask	Input	48	
in_Q3_L2_tx_pause_gen	Input	1	
in_Q3_L2_tx_pause_quant	Input	16	
Instance Q3: APB Interface			
Q3_USER_APB_PSEL	Output	1	Connect with the FPGA transceiver's Q3 interface.
Q3_USER_APB_PWRITE	Output	1	
Q3_USER_APB_PENABLE	Output	1	
Q3_USER_APB_PADDR	Output	24	
Q3_USER_APB_PWDATA	Output	32	
Q3_USER_APB_PRDATA	Input	32	
Q3_USER_APB_PREADY	Input	1	
Q3_USER_APB_PSLVERR	Input	1	
Instance Q3: APB Modules			
in_Q3_ram_usr_wren_w	Input	1	Connect from the drivers in the **Functional Verification Testbench**, applicable only when debugging the APB modules in the example design. For functional use case, initialize each signal to `0`.
in_Q3_ram_usr_addr_w	Input	6	
in_Q3_usr_apb_start_w	Input	1	
in_Q3_usr_apb_addr_w	Input	24	
in_Q3_usr_apb_write_w	Input	1	
in_Q3_usr_apb_pwdata_w	Input	32	
JTAG Interface			
jtag_vio_CAPTURE	Input	1	Connect from the drivers in the **Functional Verification Testbench**, applicable only when debugging the APB modules in the example design. For functional use case, initialize each signal to `0` and leave the output port unconnected.
jtag_vio_DRCK	Input	1	
jtag_vio_RESET	Input	1	
jtag_vio_RUNTEST	Input	1	
jtag_vio_SEL	Input	1	
jtag_vio_SHIFT	Input	1	
jtag_vio_TCK	Input	1	
jtag_vio_TDI	Input	1	
jtag_vio_TMS	Input	1	

Signal Name	Direction	Width	Description
jtag_vio_UPDATE	Input	1	
jtag_vio_TDO	Output	1	
Example Design Checker			
PASS_STATUS	Output	1	Connect to the scoreboard or checker in the **Functional Verification Testbench**.

Table 8: Example Design Project Files

File Name	Description
efx_ethernet_10g_exp.sv	Example design of top-level wrapper.
<user_given_ip_name>.sv[7]	The generated Ethernet 10G MAC file based on user configuration in Efinity IP Manager.
efx_ethernet_10g_exp.sdc	Constraint file for example design.
efx_q1_rom_mif.mem	APB ROM initialization Hex file for Quad 1.
efx_q3_rom_mif.mem	APB ROM initialization Hex file for Quad 3.
efx_q1_rom_mif_an.mem	APB ROM initialization Hex file for Quad 1. To be used with `AN_ENABLE` macro.
efx_q3_rom_mif_an.mem	APB ROM initialization Hex file for Quad 3. To be used with `AN_ENABLE` macro.
efx_mac10gbe_exp_apb_master.v[7]	APB master controller module.
efx_mac10gbe_exp_apb_halt.v[7]	APB halt controller module.
efx_mac10gbe_exp_pat_gen.v	Pattern generator module.
efx_mac10gbe_exp_checker.sv	Checker module to validate data received.
efx_resetsync.v	Reset the synchronizer module.
efx_asyncreg.v	Asynchronous register 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 65.

[7] This module is encrypted.

Virtual I/O Debugger Settings

This example design has a plug-in interface known as Virtual I/O Debugger. This interface allows you to download the bitstream to your device. After the downloads are complete, you can use the Virtual I/O Debugger interface to probe and control the signals of the example design. To control the settings, refer to Virtual I/O Debugger settings.

Important: The Virtual I/O Debugger is available if the `VERIF` macro is disabled.

Note: In the following table, the signal names are prefixed with `q1_`, `q1_l2_`, `q3_`, and `q3_l2_` to indicate and distingush these instances. For detailed signal description, refer to and tables on the settings in **Virtual I/O Debugger Settings** on page 65.

Table 9: VIO0: General Signals Settings

Signal Name	Width	Probe / Source	Description
pll_locked	1	Probe	Lock signal from PLL, indicating that the clock sources (INIT_CLK, Q1_APB_CLK, Q3_APB_CLK and DEBUG_CLK) are stable. Refer to **Figure 44: Example Design for Ethernet 10G MAC Core** on page 56. This signal starts the operation of the example design, where the assertion of `PLL_LOCK` releases the resets in the Ethernet 10G MAC core and the FPGA PCS.
PASS_STATUS	1	Probe	Status of checkers. The passing criteria of checkers may vary, depending on the macro enablement. The assertion of this signal signifies that all the passing criterias are met.

Table 10: VIO0: KR Training Interface Settings

Signal Name	Width	Probe / Source	Description
Instance Q1 Lane 2			Refer to the signal description in `Signals per Lane` table in the *Signal* chapter of TJ-Series Ethernet 10GBase-KR User Guide.
q1_l2_kr_training	1	Probe	
q1_l2_kr_frame_lock	1	Probe	
q1_l2_kr_local_rx_trained	1	Probe	
q1_l2_kr_signal_detect	1	Probe	
q1_l2_kr_training_failure	1	Probe	
Instance Q3 Lane 2			
q3_l2_kr_training	1	Probe	
q3_l2_kr_frame_lock	1	Probe	
q3_l2_kr_local_rx_trained	1	Probe	
q3_l2_kr_signal_detect	1	Probe	
q3_l2_kr_training_failure	1	Probe	
Instance Q1 Lane 2 and Q3 Lane 2			Refer to `Power-Up Sequence` and `Asserting signal_ok` figures for signal behavior in the *Power Up Sequence* chapter of TJ-Series Ethernet 10GBase-KR User Guide Elitestek recommends that you control this signal column by selecting **Active-High** from a drop-down list in the control column.
q1_l2_restart_kr_training_IN	1	Source	
q3_l2_restart_kr_training_IN	1	Source	

Table 11: VIO0: PCS Interface Settings

Signal Name	Width	Probe / Source	Description
Instance Q1 Lane 2			Refer to the signal description in `Signals per Lane` table in the *Signal* chapter of TJ-Series Ethernet 10GBase-KR User Guide.
q1_l2_block_lock	1	Probe	
q1_l2_hi_ber	1	Probe	
q1_l2_irq	1	Probe	
q1_l2_pcs_status	1	Probe	
q1_l2_phy_interrupt	1	Probe	
Instance Q3 Lane 2			
q3_l2_block_lock	1	Probe	
q3_l2_hi_ber	1	Probe	
q3_l2_irq	1	Probe	
q3_l2_pcs_status	1	Probe	
q3_l2_phy_interrupt	1	Probe	

Table 12: VIO0: Ethernet 10G MAC Interface Settings

Instance Q1 Lane 2

Signal Name	Width	Probe / Source	Description
q1_cmn_ready	1	Probe	Refer to the signal description in `control_register` table in the *Register Map* chapter of the TJ-Series Ethernet 10GBase-KR User Guide.
q1_l2_init_done	1	Probe	
q1_l2_rx_signal_detect	1	Probe	
q1_l2_cnt_rst_n	1	Source	
q1_l2_cnt_tx_frame_transmitted_good	32	Probe	
q1_l2_cnt_tx_frame_pause_mac_ctrl	32	Probe	
q1_l2_cnt_tx_frame_error_txfifo_overflow	32	Probe	
q1_l2_cnt_tx_frame_is_fe	32	Probe	
q1_l2_cnt_rx_frame_received_good	32	Probe	
q1_l2_rpt_rx_frame_length	14	Probe	
q1_l2_cnt_rx_frame_error_fcs	32	Probe	
q1_l2_cnt_rx_frame_pause_mac_ctrl	32	Probe	
q1_l2_cnt_rx_frame_errors	32	Probe	
q1_l2_cnt_rx_frame_received_total	32	Probe	
q1_l2_cnt_rx_frame_undersized	32	Probe	
q1_l2_cnt_rx_frame_oversized	32	Probe	
q1_l2_cnt_rx_frame_mismatched_length	32	Probe	
q1_l2_cnt_rx_frame_filtered_by_address	32	Probe	
q1_l2_tx_pause_busy	1	Probe	
q1_l2_rx_pause_ignore	1	Source	
q1_l2_rx_address_filtering_mask	48	Source	
q1_l2_tx_pause_gen	1	Source	
q1_l2_tx_pause_quant	16	Source	

Table 13: VIO0: Ethernet 10G MAC Interface Settings

Instance Q3 Lane 2

Signal Name	Width	Probe / Source	Description
q3_cmn_ready	1	Probe	Refer to the signal description in `control_register` table in the *Register Map* chapter of the TJ-Series Ethernet 10GBase-KR User Guide.
q3_l2_init_done	1	Probe	
q3_l2_rx_signal_detect	1	Probe	
q3_l2_cnt_tx_frame_transmitted_good	32	Probe	
q3_l2_cnt_tx_frame_pause_mac_ctrl	32	Probe	
q3_l2_cnt_tx_frame_error_txfifo_overflow	32	Probe	
q3_l2_cnt_tx_frame_is_fe	32	Probe	
q3_l2_cnt_rx_frame_received_good	32	Probe	
q3_l2_rpt_rx_frame_length	14	Probe	
q3_l2_cnt_rx_frame_error_fcs	32	Probe	
q3_l2_cnt_rx_frame_pause_mac_ctrl	32	Probe	
q3_l2_cnt_rx_frame_errors	32	Probe	
q3_l2_cnt_rx_frame_received_total	32	Probe	
q3_l2_cnt_rx_frame_undersized	32	Probe	
q3_l2_cnt_rx_frame_oversized	32	Probe	
q3_l2_cnt_rx_frame_mismatched_length	32	Probe	
q3_l2_cnt_rx_frame_filtered_by_address	32	Probe	
q3_l2_tx_pause_busy	1	Probe	
q3_l2_cnt_rst_n	1	Source	
q3_l2_rx_pause_ignore	1	Source	
q3_l2_rx_address_filtering_mask	48	Source	
q3_l2_tx_pause_gen	1	Source	
q3_l2_tx_pause_quant	16	Source	

Table 14: VIO0: Q1 and Q3 APB Interface and Status Settings

Signal Name	Width	Probe / Source	Description
q1_apb_rom_end	1	Probe	The assertion of this signal indicates that all the configuration settings stored in the built-in APB ROM are executed.
q3_apb_rom_end	1	Probe	
q1_usr_apb_start	1	Source	This signal enables you to manually trigger APB requests. Pre-requisite: Before asserting this signal, you need to ensure that signal `q1_apb_rom_end` is asserted, indicating that the APB master is available to grant the APB commands from this signal. Elitestek recommends that you control this signal column by selecting **Active-High** from a 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 to the 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 re-assert to trigger the subsequent APB request.
q3_usr_apb_start	1	Source	
q1_usr_apb_write	1	Source	APB write.
q1_usr_apb_addr	24	Source	APB address.
q1_usr_apb_pwdata	32	Source	APB write data.
q3_usr_apb_write	1	Source	APB write.
q3_usr_apb_addr	24	Source	APB address.
q3_usr_apb_pwdata	32	Source	APB write data.

Table 15: VIO0: Q1 and Q3 User APB Settings

Signal Name	Width	Probe / Source	Description
q1_ram_usr_wren	1	Source	Drive this signal to permanent `0` to retrieve the APB PRDATA with the corresponding APB_PADDRESS from the built-in APB RAM.
q3_ram_usr_wren	1	Source	
q1_ram_usr_addr	5	Source	Every APB read operation stores the APB PRDATA and the APB_PADDRESS into the built-in APB RAM in an ascending order. If you set the ram_usr_addr to ‘h00, the ram_dout_d and ram_dout_a respectively display the APB PRDATA and APB_PADDRESS of the 1st APB read command. Setting ram_usr_addr to ‘h01 displays the results of the 2nd APB read command, and so on.
q1_ram_dout_d	32	Probe	
q1_ram_dout_a	24	Probe	
q3_ram_usr_addr	5	Source	
q3_ram_dout_d	32	Probe	
q3_ram_dout_a	24	Probe	

Table 16: VIO1: Q1 and Q3 Checker Settings

Signal Name	Width	Probe / Source	Description
Checker: RX TUSER			The assertion of this signal indicates that there is no error in the RX data path. If this signal is `0`, it indicates that there are errors in the RX PCS or/and Ethernet 10G MAC core.
Q1_L2_rx_tuser_pass	1	Probe	
Q3_L2_rx_tuser_pass	1	Probe	
Checker: Error Frame at TX			The assertion of this signal indicates that there is no error frame in the RX data path.
Q1_L2_fe_pkt_pass	1	Probe	
Q3_L2_fe_pkt_pass	1	Probe	
Checker: KR Training Status			The assertion of this signal indicates that the 10G-KR link training is successful.
Q1_L2_kr_training_pass	1	Probe	
Q3_L2_kr_training_pass	1	Probe	
Checker: PCS Fault Status			The assertion of this signal indicates that both the TX and RX paths in the 10G PCS are fault free. Refer to the status register description in `Status Register` table in the *Register Map* chapter of the TJ-Series Ethernet 10GBase-KR User Guide.
Q1_L2_pcs_fault_pass	1	Probe	
Q3_L2_pcs_fault_pass	1	Probe	
Checker: Phy Interrupt			The assertion of this signal indicates that during the PHY power up handshake, the transition of the PHY power state transition are per expectation.
Q1_L2_phy_interrupt_pass	1	Probe	
Q3_L2_phy_interrupt_pass	1	Probe	
Checker: Auto Negotiation			The assertion of this signal indicates that the 10G PCS has achieved successful auto negotiation with link partner. Refer to the auto negotiation link partner register description in `usxgmii_an_lp_register` table in the *Register Map* chapter of the TJ-Series Ethernet 10GBase-KR User Guide .
Q1_L2_auto_nego_pass	1	Probe	
Q3_L2_auto_nego_pass	1	Probe	

Ethernet 10G MAC Testbench

The IP Manager also generates encrypted source code to allow you to simulate with various simulators for verification.

To generate the testbench deliverables:

1. Create a new project.
2. Select the **Ethernet 10G MAC Core** from the IP Catalog and set your configurations.
3. In the **Deliverable** tab, ensure that you turn on one (or more) of the following:
    - **Testbench**
    - **Testbench/ncsim**
    - **Testbench/synopsys**
    - **Testbench/aldec**
    - **Testbench/modelsim**
4. Click **Generate**.
5. The testbench deliverables are available in **<project path>\ip\<ip-module-name>\Testbench directory**.

Contact Elitestek Support to obtain the testbench for the Ethernet 10G MAC core with the FPGA PCS.

Note: The Ethernet 10G MAC core testbench is available in Efinity software version 2024.2.294.1.19 and later.

Acronyms and Abbreviations

Table 17: Acronyms and Abbreviations

Term	Definition
APB	Advanced Peripheral Bus
AXI ST	Advanced eXtensible Interface Streaming
CRC	Cyclic Redundancy Check
EOF/EOP	End of Frame/End of Packet
FCS	Frame Check Sequence
FPGA	Field Programmable Gate Array
IPG	Inter Packet Gap or Inter Frame Gap
MAC	Media Access Control
MTU	Maximum Transfer Unit
PCS	Physical Coding Sublayer
PHY	The Physical Layer (the first and lowest layer in the seven-layer OSI model)
PLL	Phase Locked Loop
RX	Receiver Channel
SFD	Start Frame Delimiter
SOF/SOP	Start of Frame or Start of Packet
TX	Transmit Channel
XGMII	10 Gigabit Media Independent Interface

Revision History

Table 18: Revision History

Date	Document Version	IP Version	Description
June 2025	1.4	1.4	Updated device support. (SIP-957) (DOC-2566) Updated more details in example design. Updated figure Example Design for Ethernet 10G MAC Core. Updated figure Conversion from XGMII RX to RX AXI ST, Cut Through Mode, Silent Drop of Broadcast RX Frame, Priority at XGMII TX between TX_PAUSE_GEN and TX_AXI_ST, Priority Flow Control at XGMII TX, Link Condition - Detection and Transmission, Faulty Link Condition during Pause Mode with RX_QUANT, RX AXI Interface during Faulty Link Condition, RX Statistic – RX Pause Frame, and Top-Level Module for Efinity Compilation.
May 2025	1.3	1.3	Added TX_PKT_CNT_DW in Customizing the Ethernet 10G MAC. (DOC-2539) Updated Cut Through Mode and Assertion of TX_AXI_TUSER topic.
May 2025	1.2	1.2	Added new standard in Features. (DOC-2510) Added note in Programmable Inter Packet Gap (IPG) Changed topic title of Auto Padding for Short TX Frames to Automatic Padding for Short TX Frames. Updated the following topics – Resource Utilization, Ethernet Packets, Frames, and IPG, TX AXI ST 64, RX AXI ST 64, XGMII TX, XGMII RX, Store Forward Mode, Automatic Padding for Short TX Frames, Non-Compliant TX_AXI_TLAST, Assertion of TX_AXI_TUSER, Non-Streaming TX Data, Dropped TX_AXI_TVALID Before End of Frame, Undersized RX Frame, Send Pause Frame at TX, Decoding Pause Frame at RX, Link Fault Sequence, and Latency. Updated **Figure 19: Invalid TX Frames Due to Non-Compliant TX_AXI_TLAST** on page 23, **Figure 20: Error Frame Due to Assertion of TX_AXI_TUSER** on page 24, **Figure 21: Error Frame Due to Non-Streaming TX Data** on page 25, **Figure 22: Error Frame Due to Dropped TX_AXI_TVALID (Cut Through Mode Only)** on page 26, **Figure 29: Link Condition – Detection and Transmission** on page 36, **Figure 31: RX AXI Interface during Faulty Link Condition** on page 37, and **Figure 32: Single Occurrence of Link Fault Causing Incorrect Decoding of XGMII RX Frames** on page 38 Added **Figure 14: Incorrect Decoding when RX IPG < 5** on page 18 and **Figure 27: Priority at XGMII TX between TX_PAUSE_GEN and TX_AXI_ST** on page 32. Updated Customizing the Ethernet 10G MAC.

Date	Document Version	IP Version	Description
February 2025	1.1	1.1	Added Example Design and Testbench topics. Updated topics – Reset Signals, Non-Compliant TX_AXI_TLAST, Link Fault Sequence, XGMII RX, Cut Through Mode, Store Forward Mode, Dropped TX_AXI_TVALID Before End of Frame, TX FIFO Overflow, Address Filtering and Broadcast Filtering at RX, Decoding Pause Frame at RX, Statistic Reporting, cnt_tx_frame_is_fe, cnt_tx_frame_error_ txfifo_overflow, cnt_rx_frame_filtered_by_address, and error rpt_rx_frame_length. Added important note in Power Up Handshake with FPGA Transceiver. Updated note in Data Streaming Mode for Transmission and Virtual I/O Debugger Settings topic. Added and updated note in Decoding Pause Frame at RX. Updated statement in rpt_rx_frame_length topic. Corrected link at Interface Designer. Updated **Figure 31: RX AXI Interface during Faulty Link Condition** on page 37, **Figure 1: Ethernet 10G MAC Core Interaction with User's Logic and the FPGA Ethernet 10G PCS** on page 4, **Figure 2: Ethernet 10G MAC Core Sub-Modules** on page 7, **Figure 3: Ethernet 10G MAC Core Clock Domain** on page 8, **Figure 4: Ethernet 10G MAC Core Reset Domains** on page 9. Redesigned tables in Virtual I/O Debugger Settings topic. Updated **Table 3: Examples of Unicast, Multicast, and Broadcast Filtering Operation** on page 30. Updated **Figure 36: RX Statistic – Undersized RX Frame** on page 42, **Figure 37: RX Statistic – Oversized RX Frame** on page 42, **Figure 38: RX Statistic – Mismatched Length RX Frame** on page 43, **Figure 39: RX Statistic – RX Frame with FCS Error** on page 43, **Figure 40: RX Statistic – RX Pause Frame** on page 44, **Figure 41: RX Statistic Reporting – Non-Streaming RX Data Frame** on page 45, and **Figure 42: RX Statistic Reporting – Error Frame** on page 46. (SIP-827)
December 2024	1.0	1.0	Initial release. (DOC-2163) Added IP Version in Revision History. (DOC-2185)