SUMMARY

mxeth_ver2 and mxeth.2 block consists of a MAC block and peripheral circuits. MAC block uses GEM_GXL which is 3PIP from Cadence.

mxeth.2 SoftIP supports mxs40e platform. The mxeth.2 block is derived directly from the mxeth_ver2 IP with no functional changes.

The Gigabit Ethernet MAC (GEM_GXL) module implements a 10/100/1000 Mbps Ethernet MAC compatible with the IEEE 802.3 standard. The GEM_GXL 3PIP has multiple design compile directives to configure design to support different features. In mxeth, GEM_GXL is configured to supports following major features: full duplex operation with 1536 max packet size; 4 PHY interface modes can be selected (MII, GMII, RMII and RGMII); Support 10/100/1000Mbps PHY speed; Has 3 priority transmit and receive queues; Has 16 type-1 and 16 type-2 screener registers; Supports 802.1AS(PTP) and IEEE 1588; Supports AVB; DMA with AXI or AHB interface for external queue memory access.mxeth_ver2 and mxeth.2 is a wrapper design to integrate GEM_GXL with a few glue logics for MAC operation. They are clock and reset control, AHB slave interface, AHB2APB bridge, AXI address merge (in the case of AXI Master), RGMII interface conversion and SPRAM for TX/RX buffer. mxeth_ver2 and mxeth.2 AHB and AXI master interface can be chosen from parameter AXI_MASTER_PRESENT. Apart from the interface change, the master AHB interface is controlled by clk_slow, while AXI master interface is on clk_mem. From this point on mxeth_ver2 and mxeth.2 is referred to as mxeth in the document.

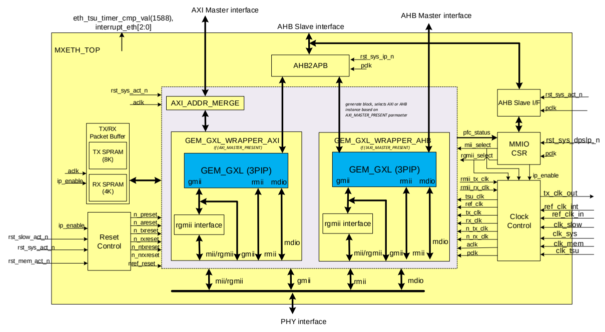


Figure 1-1 Block Diagram of mxeth_top

DE

-
- **Figure 1-2 Block Diagram of mxeth_top**
- Uses toolkit mxtk_rst_syn1 for reset synchronizer cell.
- Uses toolkit mxtk_rst_and for reset "AND" logic cell.
- Uses mxambatk_ahb_slv_if public cells from mxambatk soft IP
- 2k/4k/8k/16k 128bit s40esram (mxeth_ver2)
- Cadence MAC IP – gem_gxl revision r1p09
- Dependent IPs: N/A
- BROS Template Version Used: AV
- IPS state: IPS4
- Complexity Number: 3

FEATURES, BENEFITS, TRADEOFFS, LIMITATIONS

- 10/100/1000 Mbps Ethernet MAC compatible with the IEEE 802.3
- AXI interface (64 bit)
- AHB interface (32 bit)
- Supports MII, RMII, GMII and RGMII PHY interface
- Supports IEEE Std 802.1AS and IEEE 1588
- Support for 802.3az EEE
- Support for 802.1Qbb priority based flow control
- Receive and transmit IP, TCP, and UDP checksum offload
- Automatic pad and CRC generation on transmitted frames
- MDIO interface for PHY management
- Supports 802.1Qav traffic shaping on two highest priority transmit queues
- Supports strict priority, DWRR, or Enhanced Transmission Selection (ETS – 802.1Qaz) on transmit queues
- IEEE802.1AS, 802.1Qav, 802.1Qbb for AVB application
- 3 transmit and receive priority queues
- Full duplex
- Max frame length is 1536 bytes
- This IP works with MXS40 SRSS.
- AIP/LEDA rules set compliant: YES.
- Gate Count (AHB Master / AXI Master) = ~234kgate/321kgate (NAND2 equivalent at minimum configurations). Fmax = 200MHz. (AXI interface speed), 100 MHz (AHB interface speed).
- "Does this IP block affect overall chip performance?" NO.
- Is this block critical to Chip Integration? NO.
- Is the block intended for use in a single product?" NO.

TABLES OF LINKS

APPENDIX 3 – TABLE OF LINKS

DESIGN METRICS SUMMARY

Unique Spec ID	Metric	Result
DM.2	Gate Count Including SCAN @ 200MHz (AXI clock) and @ 100MHz (Peri clock). (AHB Master/AXI Master)	252K/310K
DM.22	Silicon area of referenced hard IP using minimum parameters. To find the total hard IP area for a particular set of parameters, add the delta values from the SoftIP_Parameters tab to this number. Silicon area is the actual silicon area after resizing during mask creation. This may differ from the drawn area reported by the layout editor.	45.5ksqum
DM.10	DFT Stuck At Fault (SAF) coverage	99.81%

SoftIP_DesignMetrics

CONFIGURATION OPTIONS

Multiple design parameters are available in this block to allow design time configuring TSU clock source and instantiate clock pre-scaler. TX/RX buffer size is also configurable but still needs changing SRAM address width design directive provided in gem_gxl.v.

TEST

This is a Soft IP will be tested using full scan ATPG patterns. SRAM is tested using memory BIST. At IPS3, the DFT coverage is 99.81%

TYPICAL APPLICATION

The Gigabit Ethernet MAC (GEM_GXL) module implements a 10/100/1000 Mbps Ethernet MAC compatible with the IEEE 802.3 and AVB standard. It can be used in high speed network in automotive. The IP also supports IEEE Std 802.1AS – Timing and Synchronization for Time-Sensitive Application in Bridged LAN's.

FMEA

mxeth is configurable to support applications. This IP can be configured to support 10M/100M/1000M MII speed, AHB/AXI system bus, different packet buffer size. This bring challenges to design complexity, Verification and Validation coverage.

COMPETITIVE IPs

IP feasibility analysis is captured in memo WJIN-425.

DC AND AC SPECIFICATIONS

The target operation frequency is
AXI interface – 200MHz
AHB interface – 100MHz
PHY interface (RGMII/GMII – 10/100/1000Mbps, RMII – 10/100Mbps, MII – 10/100Mbps)

SoftIP_BlockPinList

SoftIP_AC_Specs

POWER-PERFORMANCE AND AREA

This block employs standard clock gating techniques to reduce active power. The block operation in Active/LPActive and Sleep/LPSleep modes of the chip. The block will be unpowered in DeepSleep (and lower modes). At IPS3 the gate count estimate is 310k gates (NAND2 equivalent), the SRAM area is 45.5 ksqum.

PUBLIC CELLS, CATEGORY, PCIOS, DESCRIPTION

SoftIP_PublicCells

Table of Contents

Table of Tables

Table of Figures

1. PURPOSE/SCOPE

1.1 Purpose

This specification documents the requirements of the MXS40 soft IP Ethernet MAC block mxeth.

1.2 Scope

The requirements are set by the NPP Definition team and must determine the performance and functionality that all target products need. The development process creates a circuit design that fulfills these requirements.

For each phase of the block development (IPS1-4), a corresponding IPS scorecard needs to be created and signed off by peers and customers (http://npdis.design.cypress.com/servlet/ChecklistServlet).

1.3 BROS History

IP Vault Revision	BROS Revision	Reason for Update
N/A	**	Initial BROS
N/A	*A	Update for IPS3_DESIGN for Player
N/A	*B	Update for IPS3_DESIGN
N/A	*C	Update for IPS3_DESIGN after changing timing constraints
N/A	*D	Update for IPS3
2.1.2_IPS3	*E	Update for IPS3
2.1.2_IPS3	*F	Update for IPS4

Table 1 mxeth Block History

1.4 IP Version History

gem_gxl revision r1p09.

2. RESPONSIBILITIES

The Vice President of New Product Development or Senior VP of Product Engineering/Test Engineering or Vice President of IP is responsible for defining the block development and review process.

The Chip Lead and Product Line Design Director have the responsibility for approving the block requirements (Section 4).

The Chip Lead of the lead NPP or the lead of the Center of Excellence is responsible for the block requirements and block preparation. This document is meant to be a living document and updated as the block development proceeds.

3. SOFT IP: EXTERNAL SPECIFICATION

3.1 Feature Overview

mxeth is a wrapper of 3PIP MAC IP (GEM_GXL) to make it HOBTO compliant for MXS40 platform.

Feature of ethernet MAC:

- Support both Full store and forward mode and Partial store and forward mode for full-duplex operation
- 10Mbit/s, 100Mbit/s or 1GMbit/s operation
- MII, RMII, GMII and RGMII PHY interface modes
- Max frame length is 1536 bytes.
- 3 transmit and receive priority queues with internal 8K transmit packet buffer and 4K receive buffer.
- Supports IEEE Std 802.1Qav – Forwarding and Queuing Enhancements for Time-Sensitive Streams
- Supports IEEE Std 802.1AS – Timing and Synchronization for Time-Sensitive Application in Bridged LAN's
- Supports IEEE Std 1588 – Precision Time Protocol
- IEEE802.1AS, 802.1Qav, 802.1Qbb for AVB application (Audio Video Bridging Systems)
- 16 screening type 1 registers to support up to 16 received priority queues
- 16 screening type 2 registers which combine matching of VLAN priority, EtherType, Compare A, Compare B, and Compare C comparisons
- IEEE 802.3 Pause frame and MAC PFC priority based pause frame support
- Receive and transmit IP, TCP, and UDP checksum offload
- Automatic pad and CRC generation on transmitted frames
- MDIO interface for PHY management
- Supports strict priority, DWRR, or Enhanced Transmission Selection (ETS – 802.1Qaz) on transmit queues
- Support for 802.3az EEE
- AMBA AXI DMA Master Interface (32-bit address width, 64-bit data width)
- AMBA AHB DMA Master Interface(32-bit address width, 32-bit data width)

Feature of mxeth wrapper:

- Supports 2 design parameters to configure IP for different TX/RX packet buffer size.
- TSU clock source can be configured through design parameters.

- Implement an AXI address merge to merge AXI read/write address channel to reduce address routing overhead in platform.
- Implements an AHB master interface based on parameter AXI_MASTER_PRESENT. The AHB master runs on clk_slow.

Features provided by GEM_GXL but are **NOT** supported in mxeth:
- Half-duplex
- Collision
- TBI
- PCS
- LSO (Large Send Offload) and RSC (Receive Side Coalescing)
- External address matching
- Wake on Lan

3.2 System Diagram

This section explains how this IP is used in a system.

TVII-B-H-8M is the 1st NPP product integrating this IP. Below Figure illustrates its block diagram.

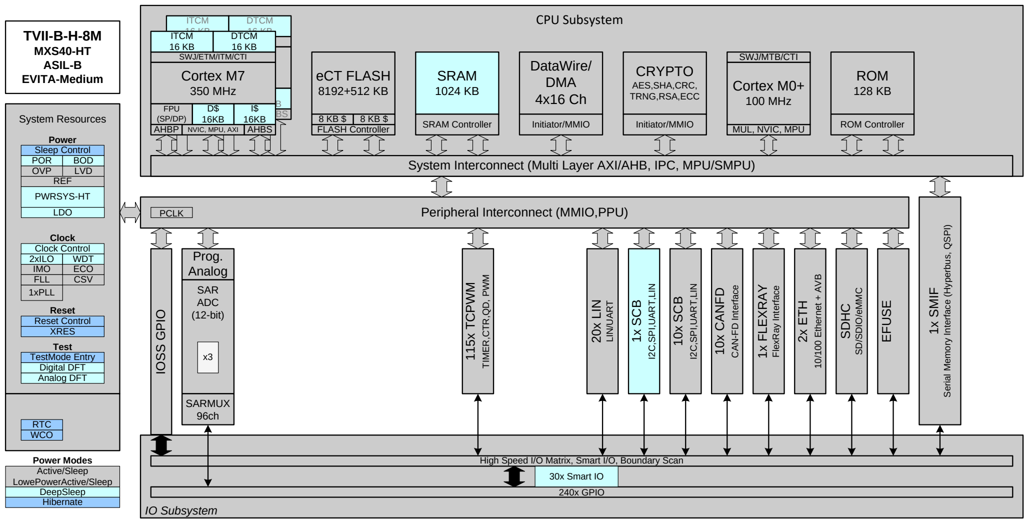


Figure 3-1 System Diagram

3.3 Block Diagram

This section explains the major components within the IP. GAUS-060 describes the changes done in mxeth_ver2 (mxeth.2 IP is directly derived from mxeth_ver2) in detail.

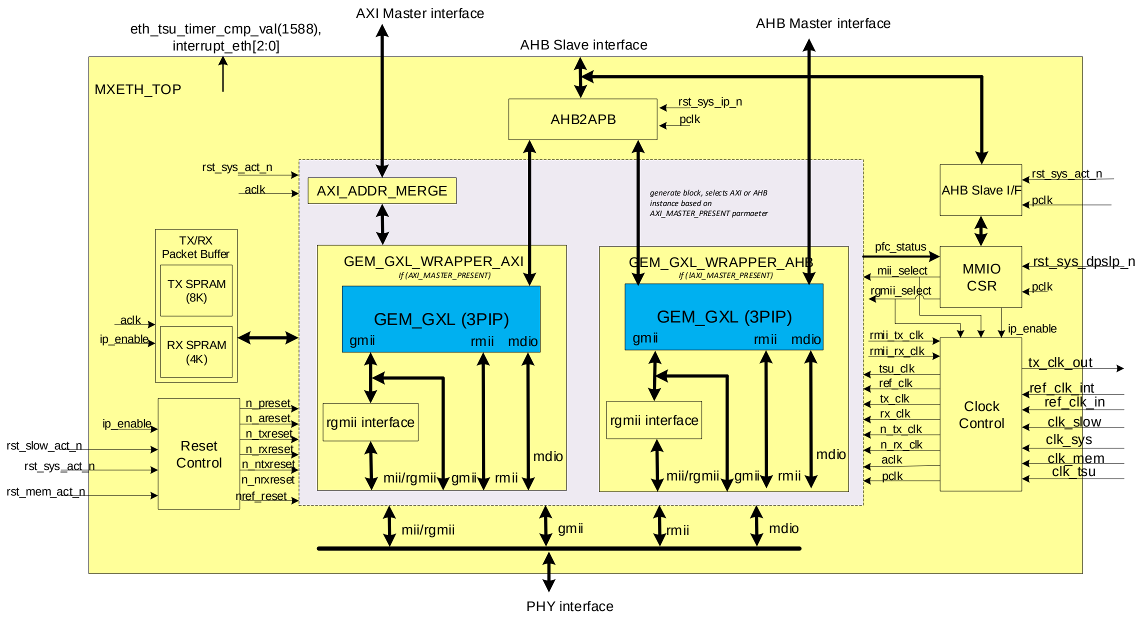


Figure 3-2 High-level Block Diagram

- GEM_GXL (3PIP):

  3PIP for MAC functionality, which doesn't support RGMII interface.

  Note that the RGMII and GMII/MII are mutually exclusive and design time configurable in GEM_GXL IP. To support runtime configurable for different type of interfaces, RGMII design directive is commented out to remove the logic instantiation in GEM_GXL IP since the GMII and MII interface are superset to RGMII. The RGMII module is instantiated at the mxeth wrapper instead and mux with configuration register to select between MII, RMII, GMII or RGMII.

- rgmii interface:

  GEM_GXL can only be configured to support either GMII or RGMII interface, through design compile directive. It doesn't meet mxeth IP feature request. The solution is to configure GEM_GXL supporting GMII and reuse rgmii interface conversion module of GEM_GXL in mxeth wrapper to implement RGMII interface outside GEM_GXL.

- GEM_GXL_WRAPPER

  Integrate rgmii interface module and gem_gxl module with additional glue logic for data path mux in order to support both GMII and RGMII.

- TX/RX Packet Buffer

TX and RX buffer implemented using single port SRAM.

- AHB2APB

  The bus conversion b/w AHB and APB. The Verilog code is obtained from TVI design database.

- AXI_ADDR_MERGE

  The mxambatk module to implement AXI address channel merge, to reduce address routing overhead in platform. Read and Write address are serial and Read channel is given higher priority.

- AHB Slave interface and MMIO CSR

  Apart from the registers defined inside 3PIP, several control and status registers are required. mxambatk_ahb_slv_if is used for AHB slave interface.

- Clock Control

  Implement clock mux for different PHY interface mode and scan mode. Clock gating cells are manually inserted to gate off clock when IP is disabled to save power.

- Reset Control
  Generate multiple resets in for different clock domain.

3.4 Functional Description

This IP targets automotive product. GEM_GXL is configured to provide the following features (CDT-251808): 10/100/1000Mbps MAC, Support MII, GMII, RMII. RGMII is implemented in mxeth wrapper. DMA with AXI 64 bits data width(or AHB 32 bits data width), running at 200MHz(100 MHz), both full store and forward mode and partial store and forward mode, Support TSU, no PCS, 3 priority Queues, 16 type-1 and 16 type-2 screen registers. 1.5K max packet length. When AHB Master interface is used, Gigabit ethernet mode must not be enabled.

The block only works at active power mode.

3.4.1 Operating Modes

Acronyms and Abbreviations

Acronym	Description
BD	Buffer Descriptor
CBS	Credit Based Shaping
CFI	Canonical Format Indicator
EEE	Energy Efficient Ethernet (IEEE Std 802.3az)
EOF	End Of Frame
FCS	Frame Check Sequence
IP	Internet Protocol
IPG	Interpacket Gap

Acronym	Description
LAN	Local Area Network (IEEE Std 802)
LLDP	Link Layer Discovery Protocol (IEEE Std 802.1AB)
LPI	Low-Power Idle (IEEE Std 802.3az)
MAC	Media Access Control (IEEE Std 802)
MDC	Management Data Clock
MII	Media Independent Interface
NSP	Non-Standard Preamble
PCS	Physical Coding Sub-Layer
PFC	Priority based Flow Control (IEEE Std 802.1Qbb)
PHY	Physical sublayer
PPPoE	Point-to-Point Protocol over Ethernet
PTP	Precision Time Protocol (IEEE Std 1588)
RFC 791	DARPA Internet Program Protocol Specification
SerDes	Serialiser/Deserialiser
SFD	Start of Frame Delimiter
SGMII	Serial Gigabit Media Independent Interface
SNAP	Subnetwork Access Protocol
SOF	Start Of Frame
TCP	Transfer Control Protocol
TS	Timestamp
TSU	Timestamp Unit
UDP	User Datagram Protocol
VLAN	Virtual LAN (IEEE Std 802.1Q)
XFI	10-Gigabit/s Electrical Interface Specification

3.4.1.1 Operation of the GEM_GXL

The contents in this section describe mxeth MAC and DMA functionalities. They are copied from Cadence GEM_GXL r1p9 data sheet with necessary updates for mxeth specific design configuration.

3.4.1.1.1 DMA interface

The GEM_GXL DMA provides a scatter gather type capability for packet data storage.

In mxeth, the GEM_GXL's DMA is configured in packet buffering mode, where Single-Port memories are used to buffer multiple frames. This allows mxeth using one of the programmable operation modes:

- Full Store and Forward
- Partial Store and Forward

In full Store and Forward mode, the entire frame will be fully buffered before being passed between the DMA and MAC. Received erroneous packets are automatically dropped before they are send to system memory, thus reducing AXI activity.

In partial store and forward mode, the transmitter will only begin to forward the packet to the MAC when there is enough packet data stored in the packet buffer. Likewise, when the receive partial store and forward mode is activated, the receiver will only begin to forward the packet to the external AHB or AXI slave when enough packet data is stored in the packet buffer. The amount of packet data required to activate the forwarding process is programmable via watermark registers which are located at the same address as the partial store and forward enable bits (find details in section 3.4.1.1.1.3).

Further key features are:

- Transmit TCP/IP checksum offload.
- Priority queuing.
- Support for TCP/IP advanced offloads to reduce CPU overhead.
- Supports manual RX packet flush capabilities.
- Optional RX packet flush when there is lack of resource.
- Optional burst padding at end of packet and end of buffer to maximize AXI efficiency.
- TX/RX timestamp capture to Buffer Descriptor entry.

3.4.1.1.1.1 Using AXI interface

GEM_GXL's AXI master interface provides separate data and address connections for Reads and Writes, which allow simultaneous, bidirectional data transfers. GEM_GXL supports multiple outstanding transactions on both the AXI Read and Write address channels, up to the limit of 16, **but mxeth only sets the depth to 2 to save IP size *(`define gem_axi_access_pipeline_bits 4'h1*).** The pipeline allows that issuing of Read and Write requests from the DMA (on AXI AR and AW channels) are decoupled from the AXI slave responses (on AXI R and B channels). The issuing of outstanding transactions can span multiple frames, maintaining high data transfer rates. **In mxeth, read and write address channels are merged to reduce overhead of platform address routine. Therefore, read and write address request can only be issued in serial.**

The GEM_GXL will buffer the descriptor accesses locally to avoid the underlying DMA from pausing.

- TX and RX descriptor reads are issued up-front and stored in a local buffer to feed the underlying DMA when required. This optimizes performance and avoids the need for the underlying DMA to pause while new descriptor fetches are sent to the system bus.

  `gem_axi_tx_descr_rd_buff_bits: DMA can issue and buffer TX BD reading in advance when AXI read channel is available (e.g. TX Buffer has no room for new frame)

  `gem_axi_rx_descr_rd_buff_bits: DMA can issue and buffer RX BD reading in advance when AXI read channel is available (e.g. DMA is storing received frame into the system memory while following frames are received)

- TX and RX descriptor writes issued by the underlying DMA are buffered locally to avoid holding up the underlying DMA if and when the system delays the completion of descriptor writes. Note a descriptor write transaction is not considered complete until the write response (BRESP) associated with that transaction has arrived.

`gem_axi_tx_descr_wr_buff_bits: DMA can buffer TX BD writing while AXI writing channel is not available or RESP has not arrived (If BD of transmitted frame is not written back, transmitted frame will not be removed from TX buffer and next transmission will not start)

`gem_axi_rx_descr_wr_buff_bits: DMA can buffer RX DB writing while AXI writing channel is not available or RESP has not arrived (If BD of received frame is not written back, received frame will not be removed from RX buffer)

mxeth defines all 4 descriptors buffer depth as “1”, means set depth to “2”. The GEM_GXL IP can issue 2 outstanding AXI transactions and locally buffer descriptors to improve performance for high latency systems. The decision to limit the outstanding AXI transaction to 2 is due to the architecture decision of fast infrastructure to merge read and write of an AXI IP. The read issued by mxeth will be for reading TX/RX descriptors and TX data packets whereas the write issued by mxeth will be for TX/RX descriptors writeback and RX data packets. Hence, if the AXI outstanding transaction is more than the TX/RX descriptor FIFO depth, it will cause bottle neck at the ethernet line since the TX/RX descriptor writebacks will be blocked due to higher priority given to read i.e. reading descriptors and data packets. With the TX/RX descriptor FIFO depth set to 2 deep respectively, the AXI outstanding transaction needs to be 2 as well.

The GEM_GXL does not provide AXI ID support and encompasses its own buffering to support high latency systems using a single ID.

The maximum burst lengths the GEM_GXL will use are programmable through *dma_config* register.

3.4.1.1.1.2 Full Store and Forward Using Packet Buffer DMA

MAC only starts transmission once the complete transmit frame is written into the TX buffer. From simulation, the frame will be flushed only when MAC completes the transmission and TX BD written back is done (stored in BD buffer).

DMA only starts fetching frame from buffer once the complete frame received. From simulation, the frame will be flushed only when DMA completes frame fetching and RX BD written back is done (stored in BD buffer). Only good received frames are written out of the DMA, so no fragments will exist in the buffers due to MAC Receiver errors. There is still the possibility of fragments due to DMA errors, for example used bit read on the second buffer of a multi-buffer frame.

When the GEM_GXL DMA is configured in the full store and forward mode, a receive over run condition occurs when the receive Packet Buffer Memory is full, or because an AXI (or AHB) error occurred. In all other modes (Partial store and forward mode), a receive overrun condition occurs when either the AXI bus was not granted quickly enough, or because an AXI error occurred, or because a new frame has been detected by the receive block when the status update or write back for the previous frame has not yet finished. For a receive over run condition, the receive over run interrupt is asserted and the buffer currently being written is recovered. The next frame that is received whose address is recognized reuses the buffer.

The benefits of full store and forward mode are:

- Discard packets that are received with errors before they are partially written out of DMA thus saving AXI bandwidth and driver processing overhead.
- Retry collided transmit frames from the buffer, thus saving AXI bus bandwidth (not applicable for mxeth).
- Implement transmit IP/TCP/UDP checksum generation offload.
- Allow half-duplex mode (not applicable for mxeth) and multi-buffer frames.

3.4.1.1.1.3 Partial Store and Forward Using Packet Buffer DMA

This feature is enabled via the TX and RX partial store and forward programmable registers. When the transmit partial store and forward mode is activated, the transmitter will only begin to forward the packet to the MAC when there is enough packet data stored in the packet buffer. Likewise, when the receive partial store and forward mode is activated, the receiver will only begin to forward the packet to the external AHB or AXI slave when enough packet data is stored in the packet buffer.
The amount of packet data required to activate the forwarding process is programmable via watermark registers which are located at the same address as the partial store and forward enable bits. Note that the minimum operational value for the TX partial store and forward watermark is 20. There is no operational limit for the RX partial store and forward watermark. To reduce the bandwidth requirements of the receive buffer manager the receive buffer size can be increased above its default value of 128 bytes by writing to the DMA configuration register.

Enabling partial store and forward is a useful means to reduce latency, also allows less TX/RX buffer size (Note: mxeth supports both full store and partial store mode, TX/RX buffer size needs to be consider to fulfill full store requirement).

Partial store and forward feature is not present in AHB Master Interface if the queue size is greater than 1. Thus, this feature should not be used at the software level in the case of AHB master interface.

3.4.1.1.1.4 DMA transaction

GEM_GXL DMA uses separate transmit and receive lists of buffer descriptors, with each descriptor describing a buffer area in system memory. This allows Ethernet packets to be broken up and scattered around the system memory.

All read operations are routed to the AXI (or AHB) read channel and all write operations to the AXI (or AHB) write channel. Both read and write channels may operate simultaneously. Arbitration logic is used when multiple requests are active on the same channel (e.g. when the transmit DMA requests a transmit data read at the same time the receive DMA requests a receive descriptor read). In these cases, the receive DMA is granted the bus before the transmit DMA. However the vast majority of requests are either receive data writes or transmit data reads both of which can operate in parallel and can execute simultaneously (Note: mxeth implements AXI address channel merge (if AXI interface is enabled) to process read and write requests sequentially, that is, it can’t execute address channel in parallel).

Transfer size is set to 64-bit words by default in the *network_config* register and burst length can be programmed in the range from single access up to 4 accesses per burst using the *dma_config* register.

3.4.1.1.1.5 Receive DMA Buffers

Received frames, optionally including FCS, are written to receive buffers located in system memory. The receive buffer depth (*rx_buf_size[7:0]*) is programmable in the range of 64 bytes to 16 Kbytes in the dma configuration register, with the default being 1536 bytes (default value is determined by design compile directive `gem_rx_buffer_length_def) and maximum packet length mxeth can support is 1536 bytes. If received frames are being routed to different priority queues (via the packet inspection screeners – refer to section “Priority Queuing in the DMA”), it is possible to program different receive buffer depths for each queue. For queue 0, the receive buffer depth is programmed through the DMA Configuration register (offset 0x10). For the other queues, they are programmed in the independent queue configuration registers (starting from offset 0x4a0). The default is 128 bytes.

The start location for each receive buffer is stored in system memory in a list of receive buffer descriptors at an address location pointed to by the receive buffer queue pointer. The base address of the receive buffer queue pointer must be configured by software using the receive buffer queue base address registers (*receive_q_ptr, receive_q1_ptr, receive_q2_ptr*).

The number of words in each buffer descriptor (BD) is dependent on the operating mode. Each BD word is defined as 32 bits. The first two words (Word 0 and Word 1) are used for all BD modes.

In Extended Buffer Descriptor mode (*dma_config[28]* = 1), two BD words (Word 2 and Word 3) are added for timestamp capture if timestamp capture mode is enabled (*rx_bd_control[5:4]* > 0_h). There are therefore either two or four BD words in each BD entry depending on the operating mode, and every BD entry will have the same number of words. To summarize:

- Every descriptor will be 64 bits wide when descriptor time capture mode is disabled.
- Every descriptor will be 128 bits wide when descriptor time capture mode is enabled.

The following description details the functionality of Word 0 and Word 1. Each list entry consists of the same first two words. The first (i.e. Word 0) contains the start location of the receive buffer and the second (i.e. Word 1) the receive status. If the length of a receive frame exceeds the DMA buffer length, the status word for the used buffer is written with zeroes except for the "start of frame" bit, which is always set for the first buffer in a frame. Bit zero of the address field is written to "1" to show the buffer has been used. The receive buffer manager then reads the location of the next receive buffer and fills that with the next part of the received frame data. Receive buffers are filled until the frame is complete and the final buffer descriptor entry table for details of the receive buffer descriptor list.

When using receive descriptor timestamp capture (*dma_config[28]* = 1), bit 2 of Word 0 is used to indicate a valid timestamp has been captured in the BD. The use of bit 2 for this purpose also necessitates the data buffer being located on a 64-bit address boundary (mxeth only supports 32-bit address).

Each receive buffer start location is a word address. The start of the first buffer in a frame can be offset by up to three bytes depending on the value written to bits 15 and 14 of the network configuration register (*receive_buffer_offset[1:0]*) and bit 2 of Word 0.

Receive Buffer Offset Configuration Bit 2 of Word 0	**Receive_Buffer_Offset[1]**	**Receive_Buffer_Offset[0]**	**Number of Bytes Offset**
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3

Table 2: Receive Buffer Byte Offset Configuration

If the start location of the buffer is offset the available length of the first buffer is reduced by the corresponding number of bytes.

Bit	Function
Word 0	
31:3	Address [31:3] of beginning of buffer
2	Address [2] of beginning of buffer or In Extended Buffer Descriptor Mode, indicates a valid timestamp in the BD entry.
1	Wrap - marks last descriptor in receive buffer descriptor list.
0	Ownership - needs to be "0" for the GEM_GXL to write data to the receive buffer. The GEM_GXL sets this to "1" once it has successfully written a frame to memory. Software has to clear this bit before the buffer can be used again.
Word 1	
31	Global all ones broadcast address detected.
30	Multicast hash match.
29	Unicast hash match.
28	External address match.
27	Unused.
26:25	Specific Address register match. Encoded as follows: 00 - Specific Address 1 register match (lowest priority) 01 - Specific Address 2 register match 10 - Specific Address 3 register match 11 - Specific Address 4 register match (highest priority) If more than one specific address is matched only one of them is indicated with priority 4 down to 1.
24	This bit has a different meaning depending on whether RX checksum offloading is enabled (network_configuration[24] = 1). **With RX checksum offloading disabled:** Type ID register match found, bit 22 and bit 23 indicate which Type ID register causes the match. **With RX checksum offloading enabled:** 0 - The frame was not SNAP encoded and/or had a VLAN tag with the CFI bit set. 1 - The frame was SNAP encoded and had either no VLAN tag or a VLAN tag with the CFI bit not set.
23:22	This bit has a different meaning depending on whether RX checksum offloading is enabled. **With RX checksum offloading disabled:** Type ID register match. Encoded as follows: 00 - Type ID Match 1 register 01 - Type ID Match 2 register 10 - Type ID Match 3 register 11 - Type ID Match 4 register If more than one Type ID is matched only one of them is indicated with priority 4 down to 1. **With RX checksum offloading enabled:** 00 - Neither the IP header checksum nor the TCP/UDP checksum was checked. 01 - The IP header checksum was checked and was correct. Neither the TCP nor UCP checksum was checked. 10 - Both the IP header and TCP checksum were checked and were correct. 11 - Both the IP header and UDP checksum were checked and were correct.
21	VLAN tag detected - Type ID of 8100$_h$. For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag has a Type ID of 8100$_h$.
20	Priority tag detected - Type ID of 8100$_h$ and null VLAN identifier. For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag has a Type ID of 8100$_h$ and a null VLAN identifier.

Bit	Function
19:17	VLAN priority - only valid if bit 21 is set. 000 - Priority 0 (lowest) BK Background 001 - Priority 1 BE Best Effort 010 - Priority 2 EE Excellent Effort 011 - Priority 3 CA Critical Applications 100 - Priority 4 VI Video, <100ms latency and jitter 101 - Priority 5 VO Voice, <10ms latency and jitter 110 - Priority 6 IC Internetwork Control 111 - Priority 7 (highest) NC Network Control
16	Canonical Format Indicator (CFI) bit - only valid if bit 21 is set.
15	End of Frame - when set the buffer contains the end of a frame. If End of Frame is not set, then the only valid status bit is Start of Frame (bit 14).
14	Start of Frame - when set the buffer contains the start of a frame. If both bits 15 and 14 are set, the buffer contains a whole frame.
13	This bit has a different meaning depending on whether jumbo frames and ignore FCS mode are enabled (network_configuration[3], network_configuration[26]). If neither mode is enabled this bit will be "0". **With jumbo frame mode enabled:** Additional bit for length of frame (bit 13), that is concatenated with bits [12:0] **With ignore FCS mode enabled and jumbo frames disabled:** This indicates per frame FCS status as follows: 0 - Frame had good FCS 1 - Frame had bad FCS, but was copied to memory as ignore FCS is enabled.
12:0	These bits represent the length of the received frame which may or may not include FCS depending on whether FCS discard mode is enabled (network_configuration[17] = 1). **With FCS discard mode disabled:** Least significant 12 bits for length of frame including FCS. If jumbo frames are enabled, these 12 bits are concatenated with bit 13 of the descriptor. **With FCS discard mode enabled:** Least significant 12 bits for length of frame excluding FCS. If jumbo frames are enabled, these 12 bits are concatenated with bit 13 of the descriptor.

Table 3: Receive Buffer Descriptor Entry

When Descriptor Timestamp Capture mode is enabled, the following table identifies the added descriptor words.

Bit	Function
Word 2	
31:30	Timestamp seconds [1:0] (see Note1)
29:0	Timestamp nanoseconds [29:0] (see Note1)
Word 3	
31:10	Unused
9:0	Timestamp seconds [11:2] (see Note1)
	Note 1: The timestamp mode is controlled using the RX BD control register (rx_bd_control). **The timestamp bits are written back to the last buffer descriptor of a frame only.**

Table 4: Receive Buffer Descriptor Entry when time stamp capture mode enabled

To receive frames, the receive buffer descriptors must be initialized by writing an appropriate address to bits [31:2] (or [31:3] for timestamp capture mode) in the first word of each list entry. Bit 0 must be written with "0". Bit 1 is the wrap bit and indicates the last entry in the buffer descriptor list.

The start location of the receive buffer descriptor list must be written with the receive buffer queue base address before reception is enabled (*network_control[2]=1*). Once reception is enabled, any writes to the receive buffer queue base address register are ignored. **In IP**

implementation, writing receive buffer queue base address register could consume 3 AXI(or AHB) clock cycles to take effect. Therefore, reception can't be enabled until 3 AXI(or AHB) clock after receive buffer queue base address register is updated. This restriction need to be taken care by SW (refer to 3.4.1.7.2 for details). When read, it will return the current pointer position in the descriptor list, though this is only valid and stable when receive is disabled.

If the filter block indicates that a frame should be copied to memory, the receive data DMA operation starts writing data into the receive buffer. If an error occurs, the buffer is recovered.

A counter in GEM_GXL represents the receive buffer queue pointer and it is not visible through its register interface. The receive buffer queue pointer increments by two or four words after each buffer has been used. It re-initializes to the receive buffer queue base address if any descriptor has its wrap bit set.

As receive buffers are used, the receive buffer manager sets bit 0 of the first word of the descriptor to "1" indication the buffer has been used.

Software should search through the "Used" bits in the buffer descriptors to find out how many frames have been received, checking the Start of Frame and End of Frame bits.

When in full store and forward mode only good received frames are written out of the DMA, so no fragments will exist in the buffers due to MAC Receiver errors. There is still the possibility of fragments due to GEM_GXL DMA errors, for example used bit read on the second buffer of a multi-buffer frame.

If bit 0 of the receive buffer descriptor is already set when the receive buffer manager reads the location of the receive buffer, then the buffer has been already used and cannot be used again until software has processed the frame and cleared bit 0. In this case, the "buffer not available" bit in the receive status register is set and an interrupt triggered. The Receive Resource Errors statistics register is also incremented.

When GEM_GXL DMA is configured in the full store and forward mode, it can be selected whether received frames should be automatically discarded when no AXI(or AHB) buffer resource (“Used” bit set) is available. This feature is selected via bit 24 of the “*dma_config*” register (by default, the received frames are not automatically discarded). If this feature is off, then received packets will remain to be stored in the RX Packet Buffer Memory until system memory resource next becomes available. This may lead to an eventual packet buffer overflow if packets continue to be received when bit 0 (“Used” bit) of the receive buffer descriptor remains set. Note that after a “Used” bit has been read, the receive buffer manager will re-read the location of the receive buffer descriptor every time a new packet is received.

When GEM_GXL DMA is configured for packet buffer mode, the upper bits of the data buffer address stored in bits [31:2] in the first word of each list entry can be dynamically altered in real-time without physically changing the system memory holding the list entry. This feature is useful if the destination has to be selected based on CPU usage or other flow control hardware. It is achieved using a MUX structure whereby it can be defined whether the upper 4 bits of the 32-bit data-buffer AXI address should come from the descriptor list entry or from a programmable register. Refer to the "*Receive Data Buffer Address Mask*" (dma_addr_or_mask) register for further details. Note that any changes to this register will be ignored while GEM_GXL DMA is currently processing a receive packet. It will only affect the next full packet to be written to system memory.

3.4.1.1.1.6 Transmit DMA Buffer

Frames to transmit are stored in one or more transmit buffers. Transmit frames can be between 1 and 1536 bytes long (determined by design compile directive `gem_jumbo_max_length), so it is possible to transmit frames longer than the maximum length (1518 bytes) specified in the IEEE Std 802.3 standard. It should be noted that zero length buffers are allowed and that the maximum number of buffers permitted for each transmit frame is 128.

The start location for each transmit buffer is stored in system memory in a list of transmit buffer descriptors at a location pointed to by the transmit buffer queue pointer. The base address for this queue pointer must be configured by software using the transmit buffer queue base address registers (*transmit_q_ptr, transmit_q1_ptr, transmit_q2_ptr*).

The number of words in each buffer descriptor (BD) is dependent on the operating mode. Each BD word is defined as 32 bits. The first two words (Word 0 and Word 1) are used for all BD modes.

In Extended Buffer Descriptor mode (*dma_config[29]* = 1), two BD words (Word 2 and Word 3) are added for timestamp capture mode if timestamp capture mode is enabled (*tx_bd_control[5:4]* > 0h). There are therefore either two or four BD words in each BD entry depending on the operating mode, and every BD entry will have the same number of words. To summarize:

- Every descriptor will be 64 bits wide when descriptor time capture mode is disabled.
- Every descriptor will be 128 bits wide when descriptor time capture mode is enabled.

The following description details the functionality of Word 0 and Word 1. Each list entry consists of the same first two words. The first (Word 0) is the byte address of the transmit buffer and the second (Word 1) contains the transmit control and status. For the packet buffer DMA, the start location for each AXI buffer is a byte address, the bottom bits of the address being used to offset the start of the data from the data-word boundary (i.e. bits 2, 1, and 0 are used to offset the address for 64-bit data paths).

Frames can be transmitted with or without automatic CRC generation. If CRC is automatically generated, pad will also be automatically generated to take frames to a minimum length of 64 bytes. When CRC is not automatically generated (as defined in Word 1 of the transmit buffer descriptor), the frame is assumed to be at least 64 bytes long and pad is not generated.

Bit	Function
	Word 0
31:0	Byte address of buffer
	Word 1
31	Used - must be "0" for the GEM_GXL to read data to the transmit buffer. The GEM_GXL sets this to "1" for the first buffer of a frame once it has been successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap - marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.
29	Retry limit exceeded, transmit error detected
28	Unused
27	Transmit frame corruption due to AXI error - set if an error occurs whilst midway through reading through reading transmit frame from the AXI, including RRESP/BRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, FCS shall be bad and `TX_ER` asserted). Also set if single frame is too large for the transmit packet buffer memory size.
26	Transmit error detected.
25:24	Reserved
23	For Extended Buffer Descriptor Mode this bit indicates a timestamp has been captured in the BD. Otherwise unused.
22:20	Transmit IP/TCP/UDP checksum generation offload errors: 000 - No error 001 - The Packet was identified as a VLAN type, but the header was not fully complete, or had an error in it. 010 - The Packet was identified as a SNAP type, but the header was not fully complete, or had an error in it. 011 - The Packet was not of an IP type, or the IP packet was invalidly short, or the IP was not of type IPv4/IPv6 100 - The Packet was not identified as VLAN, SNAP or IP. 101 - Non supported packet fragmentation occurred. For IPv4 packets, the IP checksum was generated and inserted. 110 - Packet type detected was not TCP or UDP, TCP/UDP checksum was therefore not generated. For IPv4 packets, the IP checksum was generated and inserted. 111 - A premature end of packet was detected and the TCP/UDP checksum could not be generated.
19:17	Reserved
16	No CRC to be appended by MAC. When set this implies that the data in the buffers already contains a valid CRC and hence no CRC or padding is to be appended to the current frame by the MAC. This control bit must be set for the first buffer in a frame and will be ignored for the subsequent buffers of a frame. Note that this bit must be "0" when using the transmit IP/TCP/UDP checksum generation offload, otherwise checksum generation and substitution will not occur. Note this bit must also be "0" when TX Partial Store and Forward mode is active.
15	Last buffer, when "1" this bit will indicate the last buffer in the current frame has been reached.
14	Reserved

Bit	Function
13:0	Length of buffer.

Table 5: Transmit Buffer Descriptor Entry – No LSO Frame

When Descriptor Timestamp Capture mode is enabled, the following table identifies the added descriptor words.

Bit	**Function**
	Word 2
31:30	Timestamp seconds [1:0] (see Note1)
29:0	Timestamp nanoseconds [29:0] (see Note1)
	Word 3
31:10	Unused
9:0	Timestamp seconds [11:2] (see Note1)
	Note 1: The timestamp mode is controlled using the TX BD register (tx_bd_control). After transmission the timestamp bits are written back only to the first buffer descriptor.

Table 6: Transmit Buffer Descriptor Entry when time stamp capture mode enabled

To transmit frames, the buffer descriptors must be initialized by writing an appropriate byte address to bits [31:0] in the first word (Word 0) of each descriptor list entry.

The second word (Word 1) of the transmit buffer descriptor is initialized with control information that indicates the length of the frame, whether or not the MAC is to append CRC and whether the buffer is the last buffer in the frame.

After transmission, the status bits are written back to the second word of the first buffer along with the “Used” bit. Bit 31 is the “Used” bit which must be “0” when the control word is read if transmission is to take place. It is written to “1” once the frame has been transmitted. Bits [29:20] indicate various transmit error conditions. Bit 23 indicates a valid timestamp has been captured in the BD. Bit 30 is the “Wrap” bit which can be set for any buffer within a frame. If no wrap bit is encountered the queue pointer continues to increment.

The transmit buffer queue base address register can only be updated whilst transmission is disabled or halted; otherwise any attempted write will be ignored. When transmission is halted the transmit buffer queue pointer will maintain its value. Therefore when transmission is restarted the next descriptor read from the queue will be from immediately after the last successfully transmitted frame. Whilst transmit is disabled (*network_control[3]* = 0), the transmit buffer queue pointer resets to point to the address indicated by the transmit buffer queue base address register. Note that disabling receive does not have the same effect on the receive buffer queue pointer.

Once the transmit queue is initialized, transmit is activated by writing to the transmit start bit (*network_control[9]*). Transmit is halted when a buffer descriptor with its “Used” bit set is read, a transmit error occurs, or by writing to the transmit halt bit of the network control register (*network_control[9]*). Transmission is suspended if a pause frame is received while the pause enable bit is set in the network control register (*network_control[13]*). Rewriting the start bit while transmission is active is allowed. This is implemented with a *tx_go* variable which is readable in the transmit status register at bit location 3. The *tx_go* variable is reset when:

1. Transmit is disabled
2. A buffer descriptor with its ownership bit set is read.
3. Bit 10, *tx_halt_clk*, of the network control register is written.
4. There is a transmit error such as too many retries or a transmit under run.

Once *tx_go* cleared, DMA will stop fetch new packet from system memory and MAC will not complete transmission until packet buffer is empty.

To set transmit_go write to bit 9, tx_start_clk, of the network control register (*network_control*). Transmit halt does not take effect until any ongoing transmit finishes.

If the transmit buffer list is incorrectly set up in such a way that a used bit is read mid-way through a multi buffer frame, transmission will stop. If cut-through is in operation and the MAC has actually started transmitting the frame which has its used bit set, the MAC treats it as a transmit error, and asserts tx_er truncates the frame and corrupts the FCS.

3.4.1.1.1.7 DMA Bursting

When performing data transfers, the burst length used can be programmed using bits [4:0] of the DMA configuration register. Either single accesses (burst length = 1) of incrementing bursts of up to 16 can be used as appropriate. **GEM_GXL supports AXI4 therefore the maximum length can be 256, While MXS40 platform uses a subset of the AXI3 release of the AMBA AXI protocol specification and AXI3 supports maximum burst length 16. User can configure burst length of up to 16, but it is our recommendation to use 4 to achieve quicker arbitration on the bus for other masters.**

When there is sufficient space and enough data to be transferred, the burst of programmed length will be used. If there is not enough data or space available, for example when at the end of a packet or buffer, burst lengths of less than the programmed burst length value will be issued. Single accesses will be used when a 4 Kbyte boundary will be crossed by the burst in order not to violate the AXI specification.

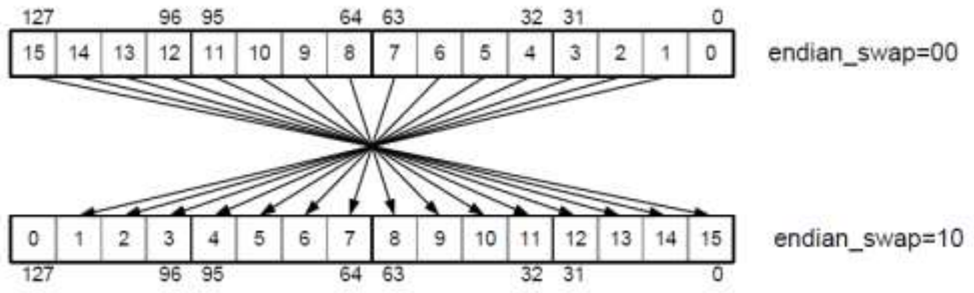
When GEM_GXL DMA is configured for packet buffer mode, an option to force DMA to pad the remaining bursts at the end of a buffer or EOP to the programmed burst length value is available via bits 26 and 25 of the *dma_config* register. Bit 26 will control the TX and bit 25 the RX. For RX, the data to burst is padded with "0"s up to the burst boundary defined by burst length. For TX, the extra data that is read is ignored by the DMA. This feature has been included for performance reasons when AXI slaves that are being accessed by GEM_GXL perform better when accessed using fixed length bursts. Note enabling this feature will not break AXI 4Kbyte boundary rule.

GEM_GXL DMA will not terminate fixed length bursts early if receive/transmit operation is disabled by writing to *network_control* register bit 2/3.

3.4.1.1.1.8 DMA Endianism

The default configuration of the DMA is to use little endian format. In order to interface to different CPU systems with different endianism requirements, the GEM_GXL DMA may be programmed to swap the endianism using bits 6 and 7 of the DMA Configuration Register (*dma_config*). Bit 6 controls the endianism of the management operations and bit 7 controls the endianism of the data operations.

When either of these modes are enabled, the order of the bytes on the AXI(or AHB) are swapped for either or both of the buffer management and data operations, as illustrated in the diagram below for a 128-bit system for data operations (bit 7 set). The same swapping applies for management operations if bit 6 is set.



3.4.1.1.1.9 DMA Packet Buffer

The packet buffer DMA mode allows multiple packets to be buffered in both transmit and receive directions and allows the DMA to withstand variable levels of access latencies on the AXI fabric. This mode offers the most efficient use of the AXI(or AHB) bandwidth.

The following figure illustrates the structure of the GEM_GXL data paths.

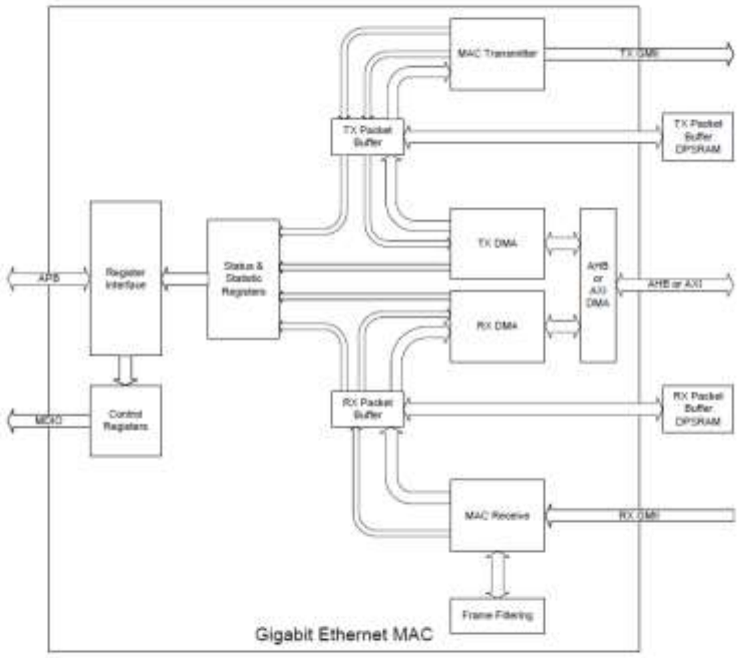


Figure 3-3 GEM_GXL Data Path Structure

Two additional memories, TX Packet Buffer and RX Packet Buffer are required to be added externally to the GEM_GXL to provide the packet buffering. These must be connected to the transmitter and receiver packet buffer interfaces at the top level. **The external memory is single port SRAM in mxeth block.**

In the transmit direction, the DMA will continue to fetch packet data up to a limit of 256 packets, or until the TX Packet Buffer Memory is full. In the receive direction, if the RX Packet Buffer Memory becomes full, then an overflow will occur. An overflow will also occur if the limit of 256 packets is breached.

Transmit Buffer

The Transmit Packet Buffer (TX Packet Buffer) will continue attempting to fetch frame data from the system memory until the TX Packet Buffer Memory is full or up to a limit of 256 packets, at which point it will attempt to maintain its full level.

To accommodate the status and statistics associated with each frame, 2 words per packet are reserved at the end of the packet data. If the packet was bad and requires to be dropped, the status and statistics are the only information held on that packet. Storing the status in the packet buffer is required in order to decouple the TX DMA interface of the buffer from the MAC Transmitter interface, to update the MAC status/statistics and to generate interrupts in the order in which the packets that they represent were fetched from the system memory.

If any errors occur on the AXI (or AHB) whilst reading the transmit frame then the fetching of packet data from memory is halted. The MAC Transmitter will continue to fetch packet data, thereby emptying the TX Packet Buffer Memory and allowing any good non-erroneous frames to be transmitted successfully. Once these have been fully transmitted, the status/statistics for the erroneous frame will be updated and software will be informed via an interrupt that an AXI (or AHB) error occurred. This way, the error is reported in the correct packet order.

The TX Packet Buffer will only attempt to read more frame data from the system memory when space is available in the TX Packet Buffer Memory. If space is not available it must wait until a packet fetched by the MAC Transmitter completes transmission and is subsequently removed from the TX Packet Buffer Memory. Note that if full store and forward mode is active and if a single frame is fetched that is too large for the TX Packet Buffer Memory, the frame is flushed and the TX DMA halted with an error status. This is because a complete frame must be written into the TX Packet Buffer Memory before transmission can begin (If frame is split into multiple buffers, DMA will measure the buffer length against the available TX Packet Buffer room to decide if it will keep fetching).

In full store and forward mode, once the complete transmit frame is written into the TX Packet Buffer Memory, a trigger is sent across to the MAC Transmitter, which will then begin reading the frame from the TX Packet Buffer Memory. Since the whole frame is present and stable in the TX Packet Buffer Memory an underflow of the MAC Transmitter is not possible. The frame is kept in the TX Packet Buffer Memory until notification is received from the MAC Transmitter that the frame data has either been successfully transmitted or can no longer be re-transmitted. When this notification is received the frame is flushed from TX Packet Buffer Memory to make room for a new frame to be fetched from system memory. The frame is removed from the packet buffer on the fly after the frame is successfully transmitted.

In partial store and forward mode, a trigger is sent across to the MAC transmitter as soon as sufficient packet data is available in the internal buffer, which will then begin fetching the frame from the packet buffer memory. If, after this point, the MAC transmitter is able to fetch data from the packet buffer faster than the DMA can fill it, an underflow of the transmitter is possible. In this case, the transmission is terminated early, and the packet buffer is flushed. Transmission can only be restarted by writing to the transmit START bit, or by toggling the *trigger_dma_tx_start* input.

Receive Packet Buffer

The Receive Packet Buffer (RX Packet Buffer) stores frames from the MAC Receiver along with their status and statistics. Frames with errors are flushed from the RX Packet Buffer Memory, whilst good frames are pushed onto the AXI (or AHB) Master Interface.

When programmed in full store and forward mode, if the frame has an error the frame data is immediately flushed from the RX Packet Buffer Memory allowing subsequent frames to utilize the freed up space. The status and statistics for bad frames are still used to update the GEM_GXL registers.

To accommodate the status and statistics associated with each frame, up to 2 words (one being for descriptor timestamp capture when enabled) per packet are reserved at the end of the packet data. If the packet was bad and requires to be dropped, the status and statistics are the only information held on that packet.

The RX Packet Buffer will also detect a full condition such that an overflow condition can be detected. If this occurs subsequent packets will be dropped and an RX overflow interrupt is raised.

For full store and forward, the RX DMA will only begin packet fetches once the status and statistics for a frame are available. If the frame has a bad status due to a frame error, the status and statistics are passed onto the GEM_GXL registers. If the frame has a good status, the information is used to read the frame from the RX Packet Buffer Memory and burst onto the AXI Master Interface using the DMA buffer management protocol.

If partial store and forward mode is active, the DMA will begin fetching the packet data before the status is available. As soon as the status becomes available, the DMA will fetch this information before continuing to fetch the remainder of the frame. Once the last frame data has been transferred to the FIFO, the status and statistics are updated to the GEM_GXL registers.

3.4.1.1.1.10 Priority Queuing in GEM_GXL DMA

The GEM_GXL DMA uses 3 transmit and 3 receive queues (configurable through design directives). Each queue has an independent list of buffer descriptors pointing to separate data streams.

In the transmit direction, higher priority queues are always serviced before lower priority queues. This strict priority scheme requires the user to ensure that high priority traffic is constrained such that lower priority traffic will have the required bandwidth. The DMA will determine the next queue to service by initiating a sequence of buffer descriptor reads interrogating the ownership bits of each. The buffer descriptor corresponding to the highest priority queue is read first. If the ownership bit of this descriptor is "1", then the DMA will progress to reading the 2nd highest priority queue's descriptor. If that ownership bit read of this lower priority queue is "1", then the DMA will read the 3rd highest priority queue's descriptor, and so on. If all the descriptors return an ownership bit set, then a resource error has occurred, an interrupt is generated and transmission is automatically halted.

Transmission can only be restarted by setting the start bit (*tx_start_clk*) in the *network_control* register (*network_control[9]*). The GEM_GXL DMA will need to identify the highest available queue for transmit from when the start bit in the *network_control* register is written to and the TX is in a halted state, or when the last word of any packet has been fetched from system memory.

The GEM_GXL transmit DMA will maximize the effectiveness of priority queuing by ensuring that high priority traffic be transmitted as early as possible after being fetched from system memory. High priority traffic will be pushed to the MAC layer depending on traffic shaping being enabled and the associated credit value for that queue, before any lower priority traffic that may pre-exist in the TX Packet Buffer. This is achieved by separating the TX Packet Buffer Memory into regions. The size of each region determines the amount of memory space allocated per queue.

For each queue, there is an associated transmit buffer queue base address register. For the lowest priority queue (Queue 0), the TX Buffer Queue Base Address register is located at offset address 0x01C. For queues 1 to 2 the transmit buffer queue base address registers are located at sequential offset addresses starting at 0x440.

In the receive direction each data packet is written to system memory in the order that it is received. For each queue, there is an independent set of receive buffers for each queue. For each queue, there is an associated receive buffer queue base address register. For the lowest priority queue (Queue 0), the RX Buffer Queue Base Address register is located at offset address 0x018. For queues 1 to 2 the receive buffer queue base address registers are located at sequential offset addresses starting at 0x480. Every received packet will pass through a programmable screening algorithm which will allocate to that frame a particular queue to route it to. The interface to the screeners is via two banks of programmable registers, Screening Type 1 (*screen_type_1_register_0* to *screen_type_1_register_15*) registers and Screening Type 2 (*screen_type_2_register_0* to *screen_type_2_register_15*) registers.

Screening Type 1 registers allow routing received frames based on particular IP and UDP fields extracted from the received frames. Specifically these fields are DS (Differentiated Services field of IPv4 frames), TC (Traffic Class field of IPv6 frames) and/or the UDP destination port. These fields are compared against the values stored in each of the Screening Type 1 registers. If the result of this comparison is positive, then the received frame is routed to the priority queue specified in that Screening Type 1 register. Refer to Screening Type 1 register description for further programming details.

Screening Type 2 registers operate independently of Screening Type 1 registers and offer additional match capabilities, extending the capabilities into vendor specific protocols. The Type 2 screening allows a screen to be configured that is the combination of all or any of the following comparisons.

1) An enabled field VLAN Priority. A VLAN Priority match will be performed if the VLAN priority enable is set. The extracted priority field in the VLAN header is compared against 3 bits within the Screening Type 2 register itself.
2) An enabled field EtherType. The EtherType field inside the Screening Type 2 register maps to 1 of 8 EtherType Match registers. The extracted EtherType is compared against the EtherType 2 (*screening_type_2_ethertype*) register designated by this EtherType field.
3) An enabled field Compare A.
4) An enabled field Compare B.
5) An enabled field Compare C.

Compare A, Compare B, and Compare C each have an Enable bit and Compare Register field. The Compare Register field is a pointer to a configured OFFSET, VALUE, and MASK. If enabled the compare is true if the data at the OFFSET into the frame, ANDed with the MASK value is equal to the COMPARE value. Either a 16 bit comparison or a 32 bit comparison is done. This selection is made via a control bit in the associated compare word1. If a 16 bit comparison is selected, then a 16 bit mask is also available to the user to select which bits should be compared. If the 32 bit compare option is selected, then no mask is available. The byte at the OFFSET number of bytes from the index start is compared through bits [15:8] of the configured VALUE and MASK. The OFFSET can be configured to be from 0 to 127 bytes from either the start of the frame, the byte following the EtherType field, the byte following the end of the IP header (IPv4 or IPv6) or the byte following the end of the TCP/UDP header. Note the logic to decode the IP header or the TCP/UDP header is reused from the TCP/UDP/IP checksum offload logic and therefore has the same restrictions on use (the main limitation is that IP fragmentation is not supported). Refer to Checksum Offload for IP, TCP and UDP section for further details. The Compare Register field points to a single pool of 32 compare registers. Compare A, Compare B, and Compare C use a common set of compare registers.

Note Compare A, Compare B, and Compare C together allow matching against an arbitrary 48-bits of data and so can be used to match against a MAC address.

All enabled comparisons are ANDed together to form the overall Type 2 screener match. Refer to Screening Type 2 register descriptions for further programming details.

Each screener register is programmable via the AHB Slave Interface. Although this is not recommended, it is possible that more than one screener register will be programmed to match against a single frame. If this happens there are 2 cases to consider.

1) If a received frame matches against multiple screeners of the same type, then the frame will route to the queue mapped by the screener located at the lowest numeric offset address. E.g. if Screening Type 2 Register 0 and Screening Type 2 Register 1 both match, then the frame will be routed to the queue identified in queue_number[3:0] of Screening Type 2 Register 0.
2) If a received frame matches against a Screening Type 2 and a Screening Type 1, then the Screening Type 1 will take precedence.

When a screener is matched, the received frame will be routed to a queue defined inside bits [3:0] of the screener register (*screen_type_1_register_i* or *screen_type_2_register_i*). Unmatched frames are routed to Queue 0.

The interrupt outputs from the GEM_GXL match the number of supported priority queues. Only DMA related events are reported using the individual interrupt outputs, as the GEM_GXL can relate these events to specific queues. All other events generated within the GEM_GXL are reported in the interrupt associated with the lowest priority queue (Queue 0). For the lowest priority queue the interrupt status register is located at offset address 0x024. For all other priority queues, the Interrupt Status register is located at sequential offset addresses starting at 0x400.

3.4.1.1.2 Advanced DMA Features to Reduce CPU Overhead

3.4.1.1.2.1 Receive Header/Data Splitting

Receive header data splitting is a feature which when enabled will force the receive DMA to split a received frame into its header and payload constituent parts. The header is not dropped, but instead separated from the payload and placed into its own DMA receive buffer. For example, a TCP/IPv4 frame will be split such that it's Ethernet, IPv4 and TCP header is written into one or more buffers, and the TCP payload is written into one or more separate buffers. A status bit in the receive descriptors will identify whether the descriptor is pointing to a header buffer or a payload buffer. Another status bit will identify whether the descriptor is pointing to the last buffer of a header (the header may be split over multiple buffers). The length of the header will also be written in the descriptor that points to the last buffer of the header.

When header/data splitting is enabled, all received frames will have their L2/L3/L4 headers separated. Note that even standard Ethernet only frames containing just 14 bytes of header will be separated. Bit 5 of *dma_config* register enables/disables header data splitting. When enabled there are specific bits of the DMA receive buffer descriptor (refer to Receive Buffer Descriptor Entry table for details).

There are certain limitations to using this feature:

- Header Data Splitting applies to all received frames. Every frame received which includes a data payload will be split into at least 2 data buffers.
- Header Data Splitting is not available in partial store and forward mode of operation. It should only be enabled when the full store and forward mode of operation is active.

3.4.1.1.3 Transmit Scheduling Algorithm

When multiple priority queues have been selected, the transmit scheduler is automatically included in the design and is responsible for selecting the next queue to be serviced either from the attached DMA or from the external FIFO interface when the DMA is not included. There are four scheduling algorithms available to the user and, with some exceptions detailed further below, the user can select one of the four modes for each queue.

- 802.1Qav Support – Credit Based Shaping

A credit-based shaping algorithm is available on the two highest priority active queues and is defined in 802.1Qav: Forwarding and Queuing Enhancements for Time-Sensitive Streams. Traffic shaping is enabled via the CBS (Credit Based Shaping) Control register (0x4bc) or the general transmit scheduling control register (offset 0x580). These two registers are aliased, so updating one register will automatically update the other. Note it is permitted to enable CBS only on the second highest priority queue and not on the highest, in which case the highest priority queue would always take precedence. Enabling CBS on a queue will enable a counter which stores the amount of transmit ‘credit’, measured in bytes that a particular queue has. A queue may only transmit if it has non-negative credit. If a queue has data to send, but is held off from doing as another queue is transmitting, then credit will accumulate in the credit counter at the rate defined in the IdleSlope register for that queue. IdleSlope is the rate of change of credit when waiting to transmit and must be less than the value of the portTransmitRate. When this queue is transmitting the credit counter is decremented at the rate of sendSlope which is defined as the portTransmitRate – IdleSlope. A queue can accumulate negative credit when transmitting which will hold off any other transfers from that queue until credit returns to a non-negative value. No transfers are halted when a queue’s credit becomes negative, it will accumulate negative credit until the transfer completes.

To ensure that the CBS scheduling is completely accurate a single transmit buffer should be used per Ethernet frame (rather than multi-buffer transmit frames).

- Fixed Priority

Any of the active queues can be selected as fixed priority and this is the default mode of operation for all queues. The queue index is used as the priority, where a higher index will have a higher priority than a lower index. The scheduler will always attempt to transmit from fixed priority queues with the highest priority. I.e. A fixed priority queue with a high queue index will always take precedence over a priority queue with a lower index.

- Deficit Weighted Round Robin (DWRR)

Any of the active queues can be selected as DWRR. If DWRR is required, then at least two of the active queues should be selected as DWRR. It should not be used in conjunction with ETS, as both algorithms operating together would make little practical sense. A DWRR enabled queue has lower priority than a CBS enabled queue or a fixed priority queue with a higher index.

The DWRR algorithm works by scanning all non-empty queues in sequence. Each queue is allocated a 'deficit counter' and an 8-bit weighting (or quantum) value. The value of the deficit counter is the maximum number of bytes that can be sent at the current time. If the deficit counter of the scanned queue is greater than the length of the packet waiting for transmission then the packet will be transmitted and the value of the deficit counter is decremented by the packet size. If it is not greater the scheduler will skip to the next DWRR enabled queue. If there is insufficient credit to transmit, the queue is simply skipped. If the queue is empty, the value of the deficit counter is reset to 0. If all queues have insufficient credit then each tx_clk cycle every queue's deficit counter is incremented by its quantum value until a queue's deficit counter obtains sufficient credit to transmit its first queued frame. The higher the quantum value chosen the quicker deficit counter will reach the required value. If all DWRR queues have the same weighting, then all queues will be granted the same overall bandwidth. The weighting value is stored in four programmable registers starting at offset 0x590. Since the design supports up to 16 priority queues, four physical registers are required. Refer to the register descriptions for further details.

Note that if fixed priority queues are to be used in conjunction with DWRR, the fixed priority queues must be at a higher index value than the DWRR queues. A consequence of this is that the enabled DWRR queues shall form a contiguous set of queues starting from queue 0.

If CBS is also used in conjunction with DWRR, the DWRR queues will share the remaining bandwidth after the CBS allocation has been deducted.

Note that if transmit cut-thru is should not be enabled if the transmit scheduler is used.

- Enhanced Transmission Selection (ETS)

The ETS algorithm is defined in 802.1Qaz: Enhanced Transmission Selection for Bandwidth Sharing between Traffic and allows traffic on specific queues to be bandwidth limited. Any of the active queues can be selected as ETS. If ETS is required, then at least two of the active queues should be selected as ETS. It should not be used in conjunction with DWRR as both algorithms operating together would make little practical sense. An ETS enabled queue has lower priority than a CBS enabled queue or a fixed priority queue with a higher index.

For each ETS enabled queue, the user should program the bandwidth requirement for each queue as a percentage of total bandwidth (an 8-bit register is used and the sum of values programmed should not exceed decimal 100). This will be the maximum bandwidth to be granted to that queue. The actual scheduling algorithm operates in a round-robin style from lowest indexed queues up to the highest indexed queue in sequence. The bandwidth allocation percentage is stored in four programmable registers starting at offset 0x590 – these are the same registers used for DWRR. Since the design supports up to 16 priority queues, four physical registers are required. Refer to the register descriptions for further details.

If CBS is also used in conjunction with ETS, the sum of the ETS queue percentages should equal the remaining bandwidth after the CBS allocation has been deducted.

Note that if transmit cut-thru is should not be enabled if the transmit scheduler is used.

Fixed priority has some limitation, refer to CDT#253873.

YUJI-37 provides simulation details of each scheduling algorithm.

3.4.1.1.4 MAC transmitter

The MAC Transmitter operates in full duplex and transmits frames in accordance with the Ethernet IEEE 802.3 standard.

A small input buffer receives data through the DMA, which, depending on the data_bus_width control bits in the *network_config* register, will extract data in 128-bit form. All subsequent processing prior to the final output is performed in bytes.

Transmit data can be output using one of four PHY interface – MII, RMII, GMII or RGMII. Note: RGMII is implemented outside GEM_GXL module.

Frame assembly starts by adding preamble and the start frame delimiter (SFD). Data is taken from the TX packet buffer a word at a time. If necessary, padding is added to take the frame length to 60 bytes. CRC is calculated using an order 32 bit polynomial. This is inverted and appended to the end of the frame taking the frame length to a minimum of 64 bytes. If the “No CRC” bit (bit 16) is set in the second word (Word 1) of the last buffer descriptor of a transmit frame neither pad nor CRC are appended.

In full duplex mode, frames are transmitted immediately. Back to back frames are transmitted 96 bit times apart to guarantee the Interpacket Gap.

In all modes of operation, if the TX DMA under runs, a bad CRC is automatically appended using the same mechanism as jam insertion and the TX_ER signal is asserted. For a properly configured system this should never happen and also it is impossible if configured to use the DMA with packet buffers, as the complete frame is buffered in TX Packet Buffer Memory.

3.4.1.1.5 MAC receiver

The MAC Receiver block checks for valid preamble, FCS, alignment and length, presents received frames to the DMA and stores the frames destination address for use by the address checking block.

Ethernet frames are normally stored in the receive buffer in the AXI memory complete with the FCS. Setting the fcs_remove bit in the Network Configuration register (*network_config[17]*) causes frames to be stored without their corresponding FCS. The reported frame length field is reduced by four bytes to reflect this operation.

The MAC Receive block signals to the register block to increment the alignment, CRC (FCS), short frame, long frame, jabber or receive symbol errors when any of these exception conditions occur.

If bit 26 is set in the *network_config* register, CRC errors will be ignored and CRC erroneous frames will not be discarded, though the *fcs_errors* statistics register will still be incremented. Additionally if configured to use the DMA and not enabled for jumbo frames mode, then bit 13 of the receive buffer descriptor Word 1 will be updated to indicate the FCS validity for the particular frame. This is useful for applications such as EtherCAT whereby individual frames with FCS errors must be identified.

Received frames can be checked for length field error by setting the *length_field_error_frame_discard* bit of the *network_config* register (bit 16). When this bit is “1”, the receiver compares a frame’s measured length with the length field (bytes 13 and 14) extracted from the frame. The frame is discarded if the measured length is shorter. This checking procedure is for received frames between 64 bytes and 1518 bytes in length.

Each discarded frame is counted in the 10-bit length field *fcs_errors* statistics register. Frames where the length field value is greater than or equal to 0600_h (1536) will not be checked.

3.4.1.1.6 Checksum Offload for IP, TCP and UDP

GEM_GXL can be programmed to perform IP, TCP, and UDP checksum offloading in both receive and transmit directions which is enabled by setting bit 24 in the Network Configuration register for receive and bit 11 in the *dma_config* register for transmit.

IPv4 packets contain a 16-bit checksum field, which is the 16-bit 1’s complement of the 1’s complement sum of all 16-bit words in the header. TCP and UDP packets contain a 16-bit checksum field, which is the 16-bit 1’s complement of the 1’s complement sum of all 16-bit words in the header, the data and a conceptual IP pseudo header.

To calculate these checksums in software requires each byte of the packet to be processed. For TCP and UDP this can use a large amount of processing power. Offloading the checksum calculation to hardware can result in significant performance improvements.

For IP, TCP, and UDP checksum offload to be useful, the operating system containing the protocol stack must be aware that this offload is available so that it can make use of the fact that the hardware can either generate or verify the checksum.

3.4.1.1.6.1 Receiver Checksum Offload

When receive checksum offloading is enabled in the GEM_GXL, the IPv4 header checksum is checked as per RFC791, where the packet meets the following criteria:

- If present, the VLAN header must be four octets long and the CFI bit must not be “1”. (Also for receive one stacked VLAN is supported.)

- Encapsulation must be RFC 894 Ethernet Type Encoding or RFC 1042 SNAP Encoding or PPPoE Encoding.
- IPv4 packet.
- IP header is of valid length.
- IP options are supported.

The GEM_GXL also checks the TCP checksum as per RFC 793, or UDP checksum as per RFC 768, if the following criteria are met:

- IPv4 or IPv6 packet.
- IP options and all IPv6 extension headers (i.e. hop-by-hop, routing and destination) are supported (except for fragmentation headers).
- Good IP header checksum (if IPv4).
- IP fragmentation is not supported. (If a packet is fragmented, then the checksum will not be checked)
- TCP or UDP packet.

When an IP, TCP, or UDP frame is received, the receive buffer descriptor gives an indication if the GEM_GXL was able to verify the checksums. There is also an indication if the frame had SNAP encapsulation. These indication bits will replace the Type ID match indication bits when the receive checksum offload is enabled. For details of these indication bits refer to table Receive Buffer Descriptor Entry.

If any of the checksums are verified incorrect by the GEM_GXL, the packet is discarded and the appropriate statistics counter incremented.

3.4.1.1.6.2 Transmitter Checksum Offload

The transmitter checksum offload is only available if GEM_GXL is configured to use the DMA in packet buffer mode and full store and forward mode is enabled. This is because the complete frame to be transmitted must be read into the TX Packet Buffer Memory before the checksum can be calculated and written back into the headers at the beginning of the frame.

Transmitter checksum offload is enabled by setting bit 11 in the DMA Configuration register. When enabled, it will monitor the frame as it is written into the TX Packet Buffer Memory to automatically detect the protocol of the frame. Protocol support is identical to the receiver checksum offload.

For transmit checksum generation and substitution to occur, the protocol of the frame must be recognized and the frame must be provided without the FCS field, by making sure that bit 16 of the transmit descriptor Word 1 is clear (VLAN tagged frames will be recognized but stacked VLAN tagged frames will not be recognized). If the frame data already had the FCS field, this would be corrupted by the substitution of the new checksum fields.

If these conditions are met, the transmit checksum offload engine will calculate the IP, TCP, and UDP checksums as appropriate. Once the full packet is completely written into TX Packet Buffer Memory, the checksums will be valid and the relevant memory locations will be updated for the new checksum fields as per standard IP/TCP and UDP packet structures.

If the transmitter checksum engine is prevented from generating the relevant checksums, bits [22:20] of the Transmit Buffer Descriptor Entry will be updated to identify the reason for the error. Note that the frame will still be transmitted but without the checksum substitution, as typically the reason that the substitution did not occur was that the protocol was not recognized.

3.4.1.1.7 MAC Filtering Block

The filter block determines which frames should be written to the GEM_GXL DMA.

Whether a frame is passed depends on what is enabled in the *network_config* register, the contents of the Specific Address (*spec_add**i**_top, spec_add**i**_bottom*), Type ID Match (*spec_type**i***) and Hash (*hash_top, hash_bottom*) registers and the frame’s destination address and type field.

Ethernet frames are transmitted a byte at a time, least significant bit first. The first six bytes of an Ethernet frame make up the destination address. The first bit of the destination address, which is the LSB of the first byte of the frame, is the group or individual bit. This is “1” for multicast addresses and “0” for unicast ones. The “all ones” address is the broadcast address and a special case of multicast.

The GEM_GXL supports recognition of specific source or destination addresses. The number of specific source or destination address filters is four. Each specific address filter consists of two registers, Specific Address Bottom i register and Specific Address Top i register. Specific Address Bottom i register stores the first four bytes of the compares source or destination address. Specific Address Top i register contains the last two bytes of this address, a control bit to select between source or destination address filtering and a 6-bit byte mask field to allow masking certain bytes during the comparison. The first filter (Filter 1) is slightly different to all other filters in that there is no byte mask. Instead address comparison against individual bits of Specific Address 1 register can be masked using the unique Specific Address Mask 1 register. The addresses stored in all filters can be specific (unicast), group (multicast), local or universal.

The destination or source address of received frames is compared against the data stored in the Specific Address registers once they have been activated. The addresses are deactivated at reset or when their corresponding Specific Address Bottom i register is written. They are activated when the corresponding Specific Address Top i register is written. If a receive frame address matches an active address, the frame is written to the DMA memory if used.

Frames may be filtered using the Type ID field for matching. Four Type ID Match registers exist and each can be enabled for matching by writing a “1” to the MSB (bit 31) of the respective register. When a frame is received, the matching is implemented as an OR function of the various types of match.

The content of each Type ID register (when enabled) is compared against the length/Type ID of the frame being received (e.g. bytes 13 and 14 in non-VLAN and non-SNAP encapsulated frames) and copied to system memory if a match is found. The encoded Type ID match bits (Word 0, bit 22 and bit 23) in the receive buffer descriptor status are set indication which Type ID Match register generated the match, if the receive checksum offload is disabled.

The reset state of the Type ID Match registers is “0”, hence each is initially disabled.

The following example illustrates the use of the address and Type ID Match registers for a MAC address of 21:43:65:87:A9:CB:

Preamble	55h
SFD	D5h
DA (Octet 0 – LSB)	21h
DA (Octet 1)	43h
DA (Octet 2)	65h
DA (Octet 3)	87h
DA (Octet 4)	A9h
DA (Octet 5 - MSB)	CBh
SA (Octet 0 - LSB)	1.)
SA (Octet 1)	1.)
SA (Octet 2)	1.)
SA (Octet 3)	1.)
SA (Octet 4)	1.)
SA (Octet 5 - MSB)	1.)
Type ID (MSB)	43h
Type ID (LSB)	21h

*Note: * -Contains the address of the transmitting device.*

The sequence above shows the beginning of an Ethernet frame. Byte order of transmission is from top to bottom as shown. For a successful match to Specific Address 1, the following address match registers must be set up:

spec_add1_bottom (0x088)	87654321h
spec_add1_top (0x08C)	0000CBA9h

And for a successful match to the Type ID, the Specific Address 1 register must be set up:

spec_type1 (0x0A8)	80004321h

3.4.1.1.7.1 Broadcast Address

Frames with the broadcast address of FFFFFFFFFFFFh are stored to memory only if the no_broadcast bit in the network_config register is set to "0".

3.4.1.1.7.2 Hash Addressing

The 64-bit Hash register (*hash_top*/*hash_bottom*) takes up two locations in the memory map. The least significant bits are stored in *hash_bottom* register and the most significant bits in *hash_top* register.

The unicast hash enable and the multicast hash enable bits in the *network_config* register enable the reception of hash matched frames. The destination address is reduced to a 6-bit index into the 64-bit Hash register using the following hash function ({hash_top,hash_bottom}[$2^{\text{hash\_index}[5:0]}$]). The hash function is an XOR of every sixth bit of the destination address.

```
hash_index[05] = da[05] ^ da[11] ^ da[17] ^ da[23] ^ da[29] ^ da[35] ^ da[41] ^ da[47]
hash_index[04] = da[04] ^ da[10] ^ da[16] ^ da[22] ^ da[28] ^ da[34] ^ da[40] ^ da[46]
hash_index[03] = da[03] ^ da[09] ^ da[15] ^ da[21] ^ da[27] ^ da[33] ^ da[39] ^ da[45]
hash_index[02] = da[02] ^ da[08] ^ da[14] ^ da[20] ^ da[26] ^ da[32] ^ da[38] ^ da[44]
hash_index[01] = da[01] ^ da[07] ^ da[13] ^ da[19] ^ da[25] ^ da[31] ^ da[37] ^ da[43]
hash_index[00] = da[00] ^ da[06] ^ da[12] ^ da[18] ^ da[24] ^ da[30] ^ da[36] ^ da[42]
```

`da[00]` represents the least significant bit of the first byte received, that is, the multicast/unicast indicator, and `da[47]` represents the most significant bit of the last byte received.

If the hash index points to a bit that is set in the Hash register then the frame will be matched according to whether the frame is multicast or unicast.

A multicast match will be signaled if the multicast hash enable bit is set, `da[00]` is "1" and the hash index points to a bit set in the Hash register.

A unicast match will be signaled if the unicast hash enable bit is set, `da[00]` is "1" and the hash index points to a bit set in the Hash register.

To receive all multicast frames, the Hash register must be set with all "1" and the multicast hash enable bit must be set to "1" in the *network_config* register.

3.4.1.1.7.3 Copy All Frames (or Promiscuous Mode)

If the copy all frames bit is set in the *network_config* register then all frames (except those that are too long, too short, have FCS errors or have RX_ER asserted during reception) will be copied to memory. Frames with FCS errors will be copied if bit 26 is set in the *network_config* register.

3.4.1.1.7.4 Disable Copy of Pause Frames

Pause frames can be prevented from being written to memory by setting the disable copying of pause frames control bit 23 in the *network_config* register. When set, pause frames are not copied to system memory regardless of the copy all frames bit, whether a hash match is found, a type ID match is identified or if a destination address match is found.

3.4.1.1.7.5 VLAN Support

An Ethernet encoded 802.1Q VLAN tag looks like this:

TPID (Tag Protocol Identifier) 16 bits	**TCI (Tag Control Information) 16 bits**
8100_h	First 3 bits priority, then CFI bit, last 12 bits VID

The VLAN tag is inserted at the 13th byte of the frame adding an extra four bytes to the frame. To support these extra four bytes, the GEM_GXL can accept frame lengths up to 1536 bytes by setting bit 8 in the *network_config* register.

If the VID (VLAN identifier) is null (000_h) this indicates a priority-tagged frame.

The following bits in the receive buffer descriptor status word give information about VLAN tagged frames:

- Bit 21 set if receive frame is VLAN tagged (i.e. Type ID of 8100_h).
- Bit 20 set if receive frame is priority tagged (i.e. Type ID of 8100_h and null VID). (If bit 20 is set bit 21 will be set also.)
- Bit 19, 18, and 17 set to priority if bit 21 is set.
- Bit 16 set to CFI if bit 21 is set.

GEM_GXL can be configured to reject all frames except VLAN tagged frames by setting the discard non-VLAN frames bit in the *network_config* register.

3.4.1.1.8 IEEE 1588 and IEEE 802.1AS Support

IEEE Std 1588 is a standard for precision time synchronization in local area networks. It works with the exchange of special Precision Time Protocol (PTP) frames. The PTP messages can be transported over IEEE Std 802.3/Ethernet, over Internet Protocol Version 4 (IPv4) or over Internet Protocol Version 6 (IPv6) as described in the annex of IEEE Std 1588-2008.

Most IEEE Std 1588 functionality can be implemented in software but for greatest accuracy hardware assist is required to detect when PTP event messages pass the MII interface (clock timestamp point).

Synchronization between master and slave clocks is a two stage process.

First the offset between the master and slave clocks is corrected by the master sending a Sync frame to the slave with a follow up frame containing the exact time the Sync frame was sent. Hardware assist modules at the master and slave side detect exactly when the Sync frame was sent by the master and received by the slave. The slave then corrects its clock to match the master clock.

Second the transmission delay between the master and slave is corrected. The slave sends a delay request frame to the master which sends a delay response frame in reply. Hardware assist modules at the master and slave side detect exactly when the delay request frame was sent by the slave and received by the master. The slave will now have enough information to adjust its clock to account for delay. For example if the slave was assuming zero delay the actual delay will be half the difference between transmit and receive time of the delay request frame (assuming equal transmit and receive times) because the slave clock will be lagging the master clock by the delay time already.

Following figures illustrate how synchronization is done using PTP frame for propagation delay calculation and clock offset correction.

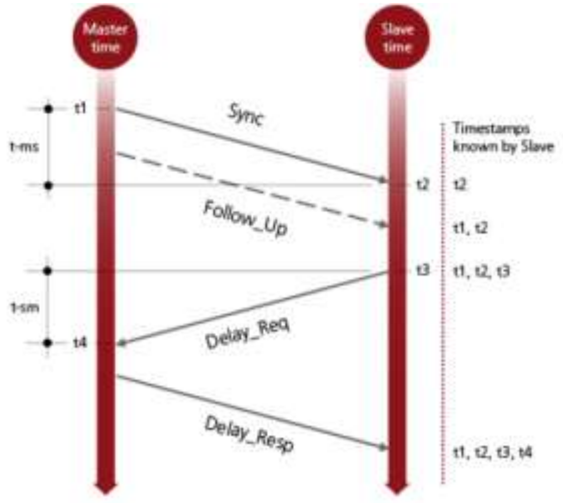


Figure 3-4 propagation delay calculation

Propagation delay is Tmpd = ((t4-t3) + (t2-t1)) / 2.

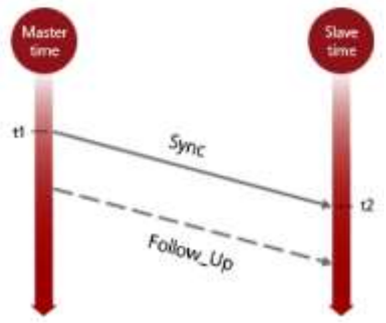


Figure 3-5 offset delay calculation

Clock offset = t2-t1-Tmpd.

Slave time can be adjusted by “clock offset” to synchronize to master time. Through multiple PTP exchange, timer_increment register (*tsu_timer_incr* and *tsu_timer_incr_sub_nsec*) can be programmed to make TSU timer counting at same speed as master clock, on the other words, to synchronize to master clock.

For hardware assist it is necessary to timestamp when Sync and Delay_Req messages are sent and received. The timestamp is taken when the message timestamp point passes the clock timestamp point. For Ethernet the message timestamp point is the SFD and the clock timestamp point is MII interface. (The IEEE Std 1588 spec refers to Sync and Delay_Req messages as event messages as these require time stamping. Follow up, delay response and management messages do not require time stamping and are referred to as general messages.)

IEEE Std 1588 version 2 (IEEE Std 1588-2008) defines two additional PTP event messages. These are the peer delay request (Pdelay_Req) and peer delay response (Pdelay_Resp) messages. These messages are used to calculate the delay on a link. Nodes at both ends of a link send both types of frames (regardless of whether they contain a master or slave clock). The Pdelay_Resp massage contains the time at which a Pdelay_Req was received and is itself an event message. The time at which a Pdelay_Resp message is received is returned in a Pdelay_Resp_Follow_Up message.

IEEE Std 1588 version 2 introduces transparent clocks of which there are two kinds, peer-to-peer (P2P) and end-to-end (E2E). Transparent clocks measure the transit time of event messages through a bridge and amend a correction field within the message to allow for the transit time. P2P transparent clocks additionally correct for the delay in the receive path of the link using the information gathered from the peer delay frames. With P2P transparent clocks Delay_Req messages are not used to measure link delay. This simplifies the protocol and makes larger systems more stable.

The GEM_GXL recognizes ten different encapsulations for PTP event messages:

1) IEEE Std 1588 version 1 (UDP/IPv4 multicast)
2) IEEE Std 1588 version 1 (UDP/IPv4 multicast with VLAN)
3) IEEE Std 1588 version 2 (UDP/IPv4 multicast)
4) IEEE Std 1588 version 2 (UDP/IPv4 multicast with VLAN)
5) IEEE Std 1588 version 2 (UDP/IPv4 unicast)
6) IEEE Std 1588 version 2 (UDP/IPv4 unicast with VLAN)
7) IEEE Std 1588 version 2 (UDP/IPv6 multicast)
8) IEEE Std 1588 version 2 (UDP/IPv6 multicast with VLAN)
9) IEEE Std 1588 version 2 (Ethernet multicast)
10) IEEE Std 1588 version 2 (Ethernet multicast with VLAN)

Note: IEEE Std 1588 version 1 (IEEE Std 1588-2002)

Unicast PTP frame recognition is enabled via bit 20 of the *network_ocnfig* register. The unicast addresses themselves are programmable via the PTP Unicast IP Destination Address (*rx_ptp_unicast/tx_ptp_unicast*) register. The first holds the RX unicast IP destination address and the other the TX unicast destination address. The PTP Unicast IP Destination Address register should only be changed when the unicast PTP frame recognition is disabled.

Example of a Sync frame in the IEEE Std 1588 version 1 format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 – 5)	
SA (Octets 6 – 11)	
Type (Octets 12 – 13)	0800h
IP stuff (Octets 14 – 22)	
UDP (Octet 23)	11h
IP stuff (Octets 24 – 29)	
IP DA (Octets 30 - 32)	E00001h
IP DA (Octet 33)	81h or 82h or 83h or 84h
source IP port (Octets 34 – 35)	
dest IP port (Octets 36 – 37)	013Fh
other stuff (Octets 38 – 42)	
version PTP (Octet 43)	01h
other stuff (Octets 44 – 73)	
control (Octet 74)	00h
other stuff (Octets 75 – 168)	

Example of a Delay_Req frame in the IEEE Std 1588 version 1 format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 – 5)	
SA (Octets 6 – 11)	
Type (Octets 12 – 13)	0800h

IP stuff (Octets 14 – 22)	
UDP (Octet 23)	11h
IP stuff (Octets 24 – 29)	
IP DA (Octets 30 - 32)	E00001h
IP DA (Octet 33)	81h or 82h or 83h or 84h
source IP port (Octets 34 – 35)	
dest IP port (Octets 36 – 37)	013Fh
other stuff (Octets 38 – 42)	
version PTP (Octet 43)	01h
other stuff (Octets 44 – 73)	
control (Octet 74)	01h
other stuff (Octets 75 – 168)	

For IEEE Std 1588 version 1 messages Sync and Delay_Req frames are indicated by the GEM_GXL if the frames type field indicates TCP/IP, UDP protocol is indicated, the destination IP address is 224.0.1.129/130/131/132, the destination UDP port is 319 and the control field is correct. The control field is 00_h for Sync frames and 01_h for Delay_Req frames.

For IEEE Std 1588 version 2 messages the type of frame is determined by looking at the message type field in the first byte of the PTP frame. Whether a frame is version 1 or version 2 can be determined by looking at the version PTP field in the second byte of version 1 and version 2 PTP frames.

In version 2 messages Sync frames have a message type value of 0_h, Delay_Req frames have 1_h, Pdelay_Req frames have 2_h and Pdelay_Resp frames have 3_h.

Example of a Sync frame in the IEEE Std 1588 version 2 (UDP/IPv4) format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 – 5)	
SA (Octets 6 – 11)	
Type (Octets 12 – 13)	0800h
IP stuff (Octets 14 – 22)	
UDP (Octet 23)	11h
IP stuff (Octets 24 – 29)	
IP DA (Octets 30 - 33)	E0000181h
source IP port (Octets 34 – 35)	
dest IP port (Octets 36 – 37)	013Fh
other stuff (Octets 38 – 41)	
message type (Octet 42)	00h
version PTP (Octet 43)	02h

Example of a Pdelay_Req frame in the IEEE Std 1588 version 2 (UDP/IPv4) format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 – 5)	
SA (Octets 6 – 11)	
Type (Octets 12 – 13)	0800h
IP stuff (Octets 14 – 22)	
UDP (Octet 23)	11h
IP stuff (Octets 24 – 29)	
IP DA (Octets 30 - 33)	E000006Bh
source IP port (Octets 34 – 35)	
dest IP port (Octets 36 – 37)	013Fh
other stuff (Octets 38 – 41)	
message type (Octet 42)	02h
version PTP (Octet 43)	02h

Example of a Sync frame in the IEEE Std 1588 version 2 (UDP/IPv6) format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 – 5)	
SA (Octets 6 – 11)	
Type (Octets 12 – 13)	86DDh
IP stuff (Octets 14 – 19)	
UDP (Octet 20)	11h
IP stuff (Octets 21 – 37)	
IP DA (Octets 38 - 53)	FF0X000000000181h
source IP port (Octets 54 – 55)	
dest IP port (Octets 56 – 57)	013Fh
other stuff (Octets 58 – 61)	
message type (Octet 62)	00h
other stuff (Octets 63 – 93)	
version PTP (Octet 94)	02h

Example of a Pdelay_Req frame in the IEEE Std 1588 version 2 (UDP/IPv6) format:

Preamble/SFD	55555555555555D5h
DA (Octets 0 – 5)	
SA (Octets 6 – 11)	
Type (Octets 12 – 13)	86DDh
IP stuff (Octets 14 – 19)	
UDP (Octet 20)	11h
IP stuff (Octets 21 – 37)	
IP DA (Octets 38 - 53)	FF0200000000006Bh

source IP port (Octets 54 – 55)	
dest IP port (Octets 56 – 57)	013Fh
other stuff (Octets 58 – 61)	
message type (Octet 62)	03h
other stuff (Octets 63 – 93)	
version PTP (Octet 94)	02h

Example of a Sync frame in the IEEE Std 1588 version 2 (Ethernet multicast) format. For the multicast address 011B19000000h Sync and Delay_Req frames are recognized depending on the message type field, 00h for Sync and 01h for Delay_Req:

Preamble/SFD	55555555555555D5h
DA (Octets 0 – 5)	011B19000000h
SA (Octets 6 – 11)	
Type (Octets 12 – 13)	88F7h
message type (Octet 14)	00h
version PTP (Octet 15)	02h

Example of a Pdelay_Req frame in the IEEE Std 1588 version 2 (Ethernet multicast) format, these need a special multicast address so they can get through ports blocked by the spanning tree protocol. For the multicast address 0180C200000Eh Sync, Pdelay_Req and Pdelay_Resp frames are recognized depending on the message type field, 00h for Sync, 02h for Pdelay_Req and 03h for Pdelay_Resp:

Preamble/SFD	55555555555555D5h
DA (Octets 0 – 5)	0180C200000Eh
SA (Octets 6 – 11)	
Type (Octets 12 – 13)	88F7h
message type (Octet 14)	02h
version PTP (Octet 15)	02h

The GEM_GXL contains a timestamp unit (TSU) which consists of a timer and registers to capture the time at which PTP event frames cross the message timestamp point. The registers are accessible through the GEM_GXL's AHB slave interface. An interrupt is issued when a capture register is updated.

3.4.1.1.8.1 Support for Time Stamping and Timestamp Accuracy

The MAC has the responsibility of sampling the TSU timer value when the TX or RX SOF event of the frame passes the MII boundary. This event is an existing signal synchronous to MAC TX/RX clock domains. The MAC uses the sampled timestamp to insert the timestamp into transmitted PTP

Sync frames (if one step sync feature is enabled), or to pass to the GEM_GXL's register block to capture the timestamp (TS) in registers, or to pass to the GEM_GXL DMA to insert into TX or RX buffer descriptors. For each of these, the SOF event, which is captured in the TX and RX clock domains respectively, is synchronized to the TSU clock domain and the resulting signal is used to sample the TSU count value. This value will be kept stable for an entire frame, or specifically at least 64 TX/RX clock cycles, since the minimum frame size in Ethernet is 64 bytes and worst case is a transfer rate of 1 byte per cycle. It is used as the source for all the components within the GEM_GXL that require the timestamp value. Since the SOF event had to pass a clock boundary, there is a degree of inaccuracy in the captured timestamp. The level of inaccuracy depends on the frequency of the TSU clock (tsu_clk). There will be no more than 1 clock cycle of inaccuracy.

In the best case, the SOF event (which is in the TX/RX clock domain) just meets the setup time of the TSU clock domain at the input to the first synchronization FlipFlop. The captured TS is N+2, but it really should be N+1. In the worst case, the captured TS is also N+2, but it really should be N.

3.4.1.1.8.2 Single Step Time Stamping

Support of one step clock for TX Sync frames can be enabled by setting bit 24 in the *network_control* register. In this mode the timestamp field, within the IEEE Std 1588 version 2 Sync frame, is replaced by the TSU timestamp value at the time the Sync frame SOF passes the MII Interface To use single step time stamping, the sampled timestamp must be stable before the point at which GEM_GXL requires to insert the timestamp. This can be guaranteed by enforcing a rule that TSU clock (tsu_clk) is greater than 1/8th the frequency of TX clock (TX_CLK) or RX clock (RX_CLK).

3.4.1.1.8.3 Timestamp Capture in Registers

There are four 80-bit timestamp status registers that capture the time at which PTP event frames are transmitted and received.

- {tsu_ptp_rx_msb_sec, tsu_ptp_rx_sec, tsu_ptp_rx_nsec}
- {tsu_ptp_tx_msb_sec, tsu_ptp_tx_sec, tsu_ptp_tx_nsec}
- {tsu_peer_rx_msb_sec, tsu_peer_rx_sec, tsu_peer_rx_nsec}
- {tsu_peer_tx_msb_sec, tsu_peer_tx_sec, tsu_peer_tx_nsec}

An interrupt is issued when these registers are updated.

3.4.1.1.8.4 Timestamp Capture in DMA Descriptors

The TX and/or RX timestamp can optionally be captured in an extended buffer descriptor when configured using bits [29:28] in the dma_config register. The timestamp can be captured for a number of frame types (PTP event or PTP general, or all frames, or none as defined in TX BD Control/RX BD Control registers) and a bit within Buffer Descriptor Word 0/1 is used to indicate that the timestamp is present.

3.4.1.1.8.5 Controlling the Timestamp Unit

The timer is implemented as a 102-bit register with the upper 48 bits counting seconds, the next 30 bits counting nanoseconds and the lowest 24 bits counting sub-nanoseconds. The lower 54 bits roll over when they have counted to one second. An interrupt is generated when the seconds increment. The timer value can be read, written and adjusted through the AHB Slave Interface. The timer is clocked with TSU clock.

There are two modes of operation to control the way the timer varies over time. These are:

Increment timer by a fixed value every TSU clock. This is increment mode.

Increment timer by a fixed value for a fixed number of clocks, followed by an alternative increment value for a single clock. This is alternative increment mode. This is the legacy mode and not recommended to use.

The timer count value can be compared to a programmable comparison value. For the comparison, the 48 bits of the seconds value and the upper 22 bits of the nanoseconds value are used. A signal is output from the core to indicate when the TSU timer count value is equal to the comparison value stored in the TSU timer comparison value registers (0x0DC, 0x0E0 and 0x0E4). An interrupt can be issued when the timer count value and the comparison value in the IEEE 1588 Timer Comparison Value registers (*tsu_msb_sec_cmp, tsu_sec_cmp and tsu_nsec_cmp*) is equal. The interrupt can be enabled with bit 29 in the interrupt enable (*int_enable*) register.

IEEE Std 802.1AS is mostly a subset of IEEE Std 1588. There is one difference in that IEEE Std 802.1AS uses the Ethernet multicast address 0180C200000E$_h$ for Sync frame recognition whereas IEEE Std 1588 does not. GEM_GXL is designed to recognize Sync frames with both IEEE Std 1588 and IEEE Std 802.1AS addresses and so can support their frame recognition simultaneously.

IEEE Std 802.1AS is mostly a subset of IEEE Std 1588. There is one difference in that IEEE Std 802.1AS uses the Ethernet multicast address 0180C200000E$_h$ for Sync frame recognition whereas IEEE Std 1588 does not. GEM_GXL is designed to recognize Sync frames with both IEEE Std 1588 and IEEE Std 802.1AS addresses and so can support their frame recognition simultaneously.

3.4.1.1.8.5.1 Increment Mode

The amount by which the timer increments each clock cycle is controlled by the timer increment (*tsu_timer_incr*) register. Bits [7:0] are the default increment value in nanoseconds and additional 16-bits of sub-nanoseconds resolution are available using the timer increment sub_nsec register (*tsu_timer_incr_sub_nsec*). If the rest of the timer increment register is written with “0” the timer increments by the value in bits [7:0], plus the value in timer increment sub_nsec register, each clock cycle.

The timer increment sub_nsec register allows a resolution of approximately 15 femtoseconds (1ns/65536).

3.4.1.1.8.5.2 Alternative Increment Mode

Bits [15:8] of the timer increment register are the alternative increment value in nanoseconds and bits [23:16] are the number of increments after which the alternative increment value is used. If bits [23:16] are 00_h then the alternative increment value will never be used.

3.4.1.1.8.5.3 Timer Adjust Mode (Not supported by mxeth)

Adjust the timer by adding or subtracting 1ns from the timer. This mode is controlled by gem_tsu_inc_ctrl and gem_tsu_ms inputs. This is timer adjust mode.

3.4.1.1.9 MAC 802.3 Pause Frame Support

The GEM_GXL supports both hardware controlled pause of the transmitter upon reception of a pause frame and hardware generated pause frame transmission.

Note: See Clause 31, and Annex 31A and 31B of the IEEE Std 802.3 for a full description of pause operation.

The start of an IEEE Std 802.3 pause frame looks like this:

Destination Address	**Source Address**	**Type (MAC Control Frame)**	**Pause Opcode**	**Pause Time**
$0180C2000001_h$	6 bytes	8808_h	0001_h	2 bytes

3.4.1.1.9.1 IEEE Std 802.3 Pause Frame Reception

Bit 13 of the *network_config* register is the pause enable control for reception. If this bit is set transmission will pause if a non zero pause quantum frame is received.

If a valid pause frame is received then the Receive Pause Quantum (*pause_time*) register is updated with the new frame’s pause time regardless of whether a previous pause frame is active or not. An interrupt (either bit 12 or 13 of the *int_status* register) is triggered when a pause frame is received, but only if the interrupt has been enabled. Pause frames received with non-zero quanta are indicated through the interrupt bit 12 of the Interrupt Status (*int_status*) register. Pause frames received with zero quanta are indicated on bit 13 of the Interrupt Status (*int_status*) register.

Once the Receive Pause Quantum (*pause_time*) is loaded and the frame currently being transmitted has been sent, no new frames are transmitted until the pause time reaches zero. The loading of a new pause time, and hence pausing of transmission, occurs since the GEM_GXL is operating in full duplex mode. A valid pause frame is defined as having a destination address that matches either the address stored in Specific Address 1 register or if it matches the reserved address of 0180C2000001$_h$. It must also have the MAC control frame Type ID of 8808$_h$ and have the pause opcode of 0001$_h$.

Pause frames that have FCS or other errors will be treated as invalid and will be discarded. IEEE Std 802.3 pause frames that are received after Priority based Flow Control (PFC) has been negotiated will also be discarded. Valid pause frames received will increment the Pause Frames Received statistics register.

The Receive Pause Quantum register decrements every 512 bit times once transmission has stopped. For test purposes, the retry test bit can be set (bit 12 in the *network_config* register) which causes the Receive Pause Quantum register to decrement every TX_CLK cycle once transmission has stopped.

The interrupt (bit 13 in the *int_status* register) is asserted whenever the Receive Pause Quantum register decrements to zero and it is enabled. This interrupt is also set when a zero quantum pause frame is received.

3.4.1.1.9.2 IEEE Std 802.3 Pause Frame Transmission

Automatic transmission of pause frames is supported through the transmit pause frame bits of the *network_control* register. If either bit 11 or bit 12 of the *network_control* register is written with “1”, an IEEE Std 802.3 pause frame will be transmitted providing the MAC Transmitter is enabled (bit 3) in the *network_control* register.

Pause frame transmission will happen immediately if transmit is inactive or if transmit is active between the current frame and the next frame due to be transmitted.

Transmitted pause frames comprise of the following:

- A destination address of 0180C2000001$_h$
- A source address taken from Specific Address 1 register
- A Type ID of 8808$_h$ (MAC control frame)

- A pause opcode of 0001$_h$
- A pause quantum register
- Fill of 00$_h$ to take the frame to minimum frame length
- Valid FCS

The pause quantum used in the generated frame will depend on the trigger source for the frame as follows:

- If bit 11 is written with “1”, the pause quantum will be taken from the Transmit Pause Quantum (*tx_pause_quantum*) register. The Transmit Pause Quantum register resets to a value of 0xFFFF giving maximum pause quantum as initial value.
- If bit 12 is written with “1”, the pause quantum will be zero.

After transmission, a pause frame transmitted interrupt will be generated (bit 14 of the *int_status* register) and the only statistics register that will be incremented will be the Pause Frames Transmitted (*pause_frames_txed*) register.

Pause frames can also be transmitted by the MAC using normal frame transmission methods.

3.4.1.1.10 MAC PFC Priority Based Pause Frame Support

GEM_GXL supports PFC Priority Based Pause transmission and reception. Before PFC pause frames can be received, bit 16 of the Network Control register must be set to “1”.

Note: Refer to IEEE Std 802.1Qbb for a full description of priority based pause operation.

The start of a PFC pause frame looks like this:

Destination Address	**Source Address**	**Type (MAC Control Frame)**	**Pause Opcode**	**Priority Enable Vector**	**Pause Time**
0180C2000001$_h$	6 bytes	8808$_h$	0101$_h$	2 bytes	2 bytes

3.4.1.1.10.1 PFC Pause Frame Reception

The ability to receive and decode priority based pause frames is enabled by setting bit 16 of the Network Control (*network_control*) register. When this bit is set, GEM_GXL will match either classic IEEE Std 802.3 pause frames or PFC priority based pause frames. Once a priority based pause frame has been received and matched, then from that moment on the GEM_GXL will only match on priority based pause frames (this is IEEE Std 802.1Qbb requirement, known as PFC negotiation). Once priority based pause has been negotiated, any received IEEE Std 802.3x format pause frames will not be acted upon.

If a valid priority based pause frame is received then GEM_GXL will decode the frame and determine which, if any, of the 8 priorities require to be paused. Up to 8 pause time registers are then updated with the 8 pause times extracted from the frame regardless of whether a previous pause operation is active or not. An interrupt (bit 12 of the *int_status* register) is triggered when a non-zero PFC pause frame is received, but only if the interrupt has been enabled (bit 12 of the interrupt mask register). Pause frames received with non-zero quantum are indicated through the interrupt bit 12 of the interrupt status register. PFC pause frames received with zero quantum cannot trigger an interrupt; that is bit 13 is never set for PFC pause frames. The loading of a new pause time occurs since the GEM_GXL is operating in full duplex mode. A valid pause frame is defined as having a destination address that matches either the address stored in Specific Address 1 register of if it matches the reserved address of 0180C2000001h. It must also have the MAC control frame Type ID of 8808h and have the pause opcode 0101h.

Pause frames that have FCS or other errors will be treated as invalid and will be discarded. Valid pause frames received will increment the Pause Frames Received statistic register.

The Receive Pause Quantum (*pause_time*) register decrement every 512 bit times immediately, following the PFC frame reception. For test purposes, the retry test bit can be set (bit 12 in the *network_config* register) which causes the Receive Pause Quantum (*pause_time*) register to decrement every RX_CLK cycle once transmission has stopped.

3.4.1.1.10.2 PFC Pause Frame Transmission

Automatic transmission of pause frames is supported through the transmit priority based pause frame bit of the Network Control (*network_control*) register. If bit 17 of the Network Control (*network_control*) register is written with “1”, a PFC pause frame will be transmitted providing the MAC Transmitter is enabled (bit 3) in the Network Control (*network_control*) register. When bit 17 of the Network Control (*network_control*) register is set to “1”, the fields of the priority base pause frame will be built using the values stored in the Transmit PFC Pause (*tx_pfc_pause*) register.

Pause frame transmission will happen immediately if transmit is inactive or if transmit is active between the current frame and the next frame due to be transmitted.

Transmitted pause frames comprise of the following:

- A destination address of 0180C2000001h
- A source address taken from Specific Address 1 register
- A Type ID of 8808h (MAC control frame)
- A pause opcode of 0101h
- A priority enable vector taken from the Transmit PFC Pause (*tx_pfc_pause*) register

- 8 pause quanta in 4 registers (tx_pause_quantum, tx_pause_quantum1, tx_pause_quantum2, tx_pause_quantum3)
- Fill of 00h to take the frame to minimum frame length
- Valid FCS

The pause quantum registers used in the generated frame will depend on the trigger source for the frame as follows:

- If bit 17 of the Network Control (network_control) register is written with “1” then the priority enable vector of the priority based pause frame will be set equal to the value stored in the Transmit PFC Pause (*tx_pfc_pause*) register bits [7:0]. For each entry equal to “0” in the Transmit PFC Pause (*tx_pfc_pause*) register [15:8], the pause quantum field of the pause frame associated with that entry will be taken from the Transmit Pause Quantum (*tx_pause_quantum*) register. For each entry equal to "1" in the Transmit PFC Pause (*tx_pfc_pause*) register [15:8], the pause quantum associated with that entry will be "0". The Transmit Pause Quantum (*tx_pause_quantum*) register resets to a value of FFFF$_h$ giving maximum pause quantum as initial value.
- The pause quantum registers are classed as static and these registers should be updated only when no PFC frame is transmitted.
- To use the eight priority pause quanta stored in the four Transmit Pause Quantum (*tx_pause_quantum*) registers, set bit 24 to “1” in the Network Control (network_control) register.

After transmission, a pause frame transmit interrupt will be issued (bit 14 of the int_status register) and the only statistics register that will be incremented will be the Pause Frames Transmitted (*pause_frames_txed*) register.

PFC Pause frames can also be transmitted by the MAC using normal frame transmission methods.

3.4.1.1.11 Energy Efficient Ethernet support

IEEE Std 802.3az adds support for energy efficiency to Ethernet (EEE). These are the key features of IEEE Std 802.3az:

1. Allows a system’s transmit path to enter a low power mode if there is nothing to transmit.
2. Allows a PHY to detect whether its link partner’s transmit path is in low power mode, therefore allowing the system’s receive path to enter low power mode.
3. Link remains up during lower power mode and no frames are dropped.
4. Asymmetric, one direction can be in low power mode while the other is transmitting normally.

5. LPI (Low Power idle) signaling is used to control entry and exit to and from low power modes.
6. LPI signaling can only take place if both sides have indicated support for it through auto-negotiation.

IEEE Std 802.3az operation:

1. Low-power control is done at MII/GMII (reconciliation sublayer).
2. As an architectural convenience in writing the 802.3az it is assumed that transmission is deferred by asserting carrier sense, in practice it will not be done this way. This system will know when it has nothing to transmit and only enter low power mode when it is not transmitting.
3. LPI should not be requested unless the link has been up for at least one second.
4. LPI is signaled on the GMII transmit path by asserting 0x01 on TXD with TX_EN low and TX_ER high.
5. A PHY on seeing LPI requested on MII/GMII will send the sleep signal before going quiet. After going quiet it will periodically emit refresh signals.
6. The sleep, quiet and refresh periods are defined in Table 78-2 of IEEE Std 802.3az. For 1000BASE-X the sleep period is 20 microseconds, the quiet period 2.5 milliseconds and the refresh period 20 microseconds. For 100BASE-TX the sleep period (T_s) is 100 microseconds, the quiet periods (T_{qt}) /(T_{qr}) are 20/24 milliseconds, and the refresh period (T_r) is 100 microseconds. 10BASE-T is not supported.
7. 1000BASE-X /100BASE-TX is required to go quiet after sleep is signaled.
8. LPI mode ends by transmitting normal idle for the wake time. There is a default time for this but it can be adjusted in software using the Link Layer Discovery Protocol (LLDP) described in Clause 79 of IEEE Std 802.3az.
9. LPI is indicated at the receive side when sleep and refresh signaling has been detected.

3.4.1.1.12 LPI Operation in Cadence IP (GEM_GXL)

Auto-negotiation:

1) Indicate EEE capability using auto-negotiation.

For the transmit path:

1) If the link has been up for 1 second and there is nothing being transmitted, write to the LPI bit (bit 19) in the Network Control register.
2) Wake up by clearing the LPI bit in the Network Control register.

For the receive path:

1) Wait for an interrupt to indicate that LPI has been received.
2) Wait for an interrupt to indicate that regular idle has been received and then re-enable the receive path.

3.4.1.1.13 Jumbo Frames

The jumbo frames enable bit in the Network Configuration register (bit 3) allows GEM_GXL in its initial state to receive jumbo frames up to a software configurable number of bytes in size. This operation is not part of IEEE Std 802.3 specification and is normally disabled. When jumbo frames are enabled, frames received with a frame size greater than the configured value are discarded. Oversize frame received register (*excessive_rx_length*) will count the number of too long frame received.

The jumbo frames maximum length can be controlled using the Jumbo-Frame Maximum Length register (*jumbo_max_length*). Its default value is 1536 bytes.

- If jumbo frame is enabled (*network_config[3]*).
  -- *jumbo_max_length* register has default value 1536, user don't need to set it for 1536 bytes transfer.
  -- User can modify *jumbo_max_length* register, and the maximum length of frame received is determined by *jumbo_max_length* register.
  -- the value of *receive_1536_byte_frames* does not care.
- If jumbo_frame is disabled
  -- if *receive_1536_byte_frames* is set, maximum length will be 1518.
  -- if *receive_1536_byte_frames* is not set, maximum length will be 1500.
  -- the value of *jumbo_max_length* does not care.

In mxeth, the maximum length of jumbo frame is 1536 bytes. The TX and RX buffer size doesn't allow longer frame to be transmitted or received in full store and forward mode.

jumbo_frame	receive_1536_byte_frames	Frame to be received
0	0	Standard frame with payload length <= 1500 (standard frame)
0	1	Frame with payload length <=1518 (standard frame + undefined frame)
1	X	Any length defined in jumbo_max_length register (Any frame)

Table 7: Jumbo-Frame configuration

0x05ee (1518) is mentioned in below link. It's possible some of the devices supports 0x05ee in length field. Therefore GEM_GXL enable this option to be compatible with those devices.
http://www.ieee802.org/3/frame_study/0411/3d0_1.pdf

3.4.1.2 PHY interface and 10/100/1000Mbps mode selection

mxeth can be configured to work at one of MII, RMII, GMII or RGMII PHY interface mode.

CTL.ETH_MODE	**Network_config[0] (speed)**	**Network_config[10] (gigabit_mode_enable)**	**PHY mode**
2'd0	0	0	MII - 10Mbps
2'd0	1	0	MII – 100Mbps
2'd1	0	1	GMII – 1000Mbps
2'd2	0	0	RGMII – 10Mbps (4bits/Cycle)
2'd2	1	0	RGMII – 100Mbps (4bits/Cycle)
2'd2	0	1	RGMII – 1000Mbps (8bits/Cycle)
2'd3	0	0	RMII – 10Mbps
2'd3	1	0	RMII – 100Mbps

Table 8: PHY mode and speed configuration

CTL.ETH_MODE must be set before IP enabled, PHY speed (*network_config*) can only be programmed after IP enabled.

The PHY interface ports of mxeth are mux with PHY mode control registers to support MII, RMII, GMII and RGMII.

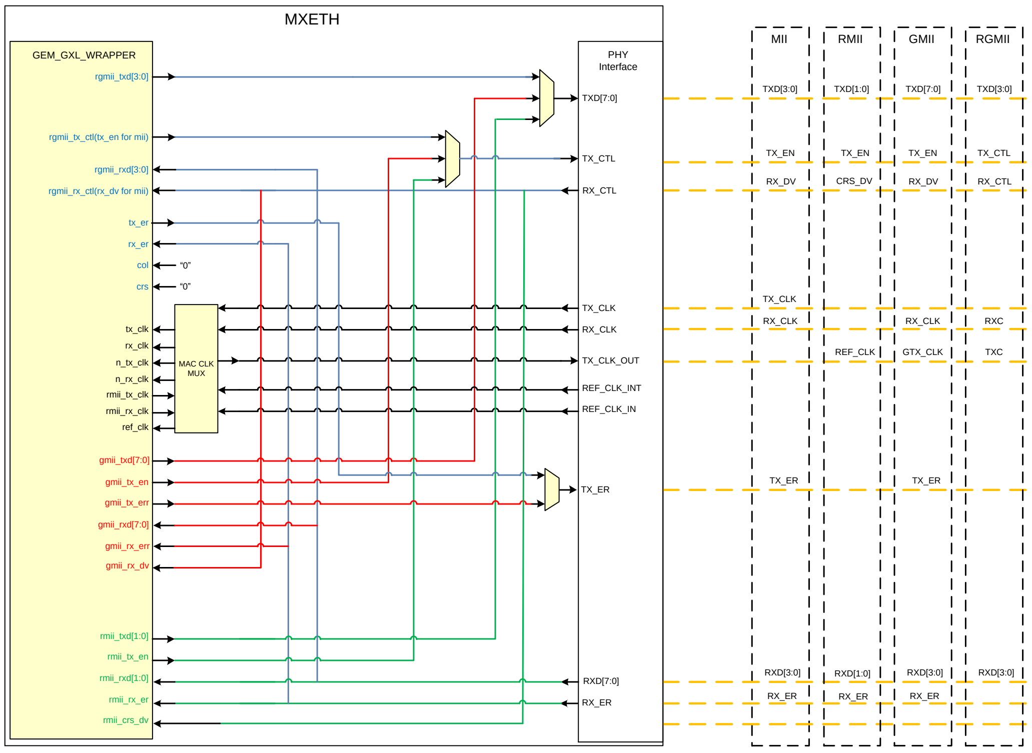


Figure 3-6: PHY interface multiplex

The connection to GPIO is described in section 3.4.2.

3.4.1.3 MDIO

mdio is a single bi-directional tristate signal going between the GEM_GXL and one or more PHYs. The GEM_GXL signals mdio_in, mdio_out and mdio_en are provided to control a chip-level tri-state buffer. Alternatively mdio may be implemented as a pull-down. In this case the enable to the pull-down should be driven with ~mdio_out & mdio_en.

PHY maintenance register (*phy_management*) is implemented as a shift register. Writing to the register starts a shift operation which is signaled as complete when bit two is set in the network status register (about 2000 pclk cycles later when bits [18:16] are set to 010 in the network configuration register). An interrupt is generated as this bit is set.

During this time, the MSB of the register is output on the mdio_out pin and the LSB updated from the mdio_in pin with each MDC cycle. This causes the transmission of a PHY management frame on MDIO. See section 22.2.4.5 of the IEEE 802.3 standard.

Reading during the shift operation will return the current contents of the shift register. At the end of the management operation the bits will have shifted back to their original locations. For a read operation the data bits will be updated with data read from the PHY. It is important to write the correct values to the register to ensure a valid PHY management frame is produced.

MDC should not toggle faster than 2.5 MHz (minimum period of 400 ns), as defined by the IEEE 802.3 standard. mdc is generated by dividing down pclk. Three bits in the network configuration register determine by how much pclk should be divided to produce MDC. The default is 32 which is acceptable for pclk running up to 80 MHz.

3.4.1.4 PHY interface clock configuration

- MII:
  - Both TX and RX clocks are supplied by PHY.
- RMII:
  - Both TX and RX clocks are supplied by reference clock.
  - CTRL.REFCLK_SRC_SEL selects reference clock source from on-chip system resource or from HSIO.
    - Block will enable TX_CLK_OUT to supply reference clock to PHY, when REFCLK_SRC_SEL is “1” (select on chip clock source)
  - CTRL.REFCLK_DIV divides reference clock and generate expected frequency (50MHz).
- GMII:
  - RX clock is supplied by PHY.
  - TX clock is supplied by reference clock.
  - CTRL.REFCLK_SRC_SEL selects reference clock source from on-chip system resource or from HSIO. In s40 platform, HSIO is the source of reference clock (CDT#300513).
  - CTRL.REFCLK_DIV divides reference clock and generate expected frequency (125MHz).
  - Block will enable TX_CLK_OUT to supply TX clock to PHY in GMII mode.
- RGMII:
  - RX clock is supplied by PHY.
  - TX clock is supplied by reference clock.
  - CTRL.REFCLK_SRC_SEL selects reference clock source from on-chip system resource or from HSIO. In s40 platform, HSIO needs to be the source of reference clock (CDT#300513) if it’s 125MHz.

- CTRL.REFCLK_DIV divides reference clock and generate expected frequency (125/25/2.5MHz).
- Block will enable TX_CLK_OUT to supply TX clock to PHY in RGMII mode.

3.4.1.5 SPRAM Size for TX/RX packet buffer

To guarantee maximum bandwidth in all modes of operation, the general rule for the RAM is a size equal to twice the anticipated maximum frame size. Having a RAM that is twice the maximum frame size allows one frame to be forwarded while the next frame is being written. For example, in the transmit direction one frame will be transmitted through the MAC while another frame is being read from the host AMBA interface. This configuration ensures maximum throughput when maximum sized frames are transmitted / received consecutively, as both halves of the DMA can operate concurrently.

For transmit, each priority queue has an independent reserved area of RAM, and the above rule should be met on each queue's RAM segment – I.e. When priority queuing mode and full store and forward mode is enabled, every configured region of the attached memory attributed to a priority queue must be greater than 2X the maximum transmitted frame length if it's full store and forward mode. For receive, all queues share the single RAM space and the above rule need only be applied to the full RAM. mxeth now deploys 8K TX RAM and 4K RX RAM:

TX RAM	8K	Queue0	4K
		Queue1	2K
		Queue2	2K
RX RAM	4K		

Table 9: packet buffer size in mxeth

- Reducing RAM size

The RAM sizes described above are cautious and are not always necessary. If maximum sized frames are not transmitted consecutively, then a smaller RAM size is permitted. In full store and forward mode, the minimum RAM size for correct functional operation must be greater than the max frame length. Two design parameters are used to configure TX buffer size and TX buffer size, refer to section 4.2 for details.

3.4.1.6 Operation frequency requirement

GEM_GXL can be configured to optionally use an external single port memory. For this mode of operation all memory signals are synchronous to aclk (AXI clock or AHB clock), which is connected to clk_mem in case of AXI and clk_slow for AHB and both the receive and transmit memory must be therefore be connected to aclk. Single port external memory operation however has some frequency limitations, where the aclk frequency must be faster than the MAC data rate. The following table details required minimum operating frequencies for all possible configurations and MAC speeds:

DMA bus width	MAC rate	Minimum AXI Frequency
64	1Gbps	65MHz
64	100Mbps	10MHz
64	10Mbps	10MHz

Table 10: Minimum AXI frequency for MAC speeds

DMA bus width	MAC rate	Minimum AHB Frequency
32	100Mbps	15MHz
32	10Mbps	10MHz

Table 11: Minimum AHB frequency for MAC speeds

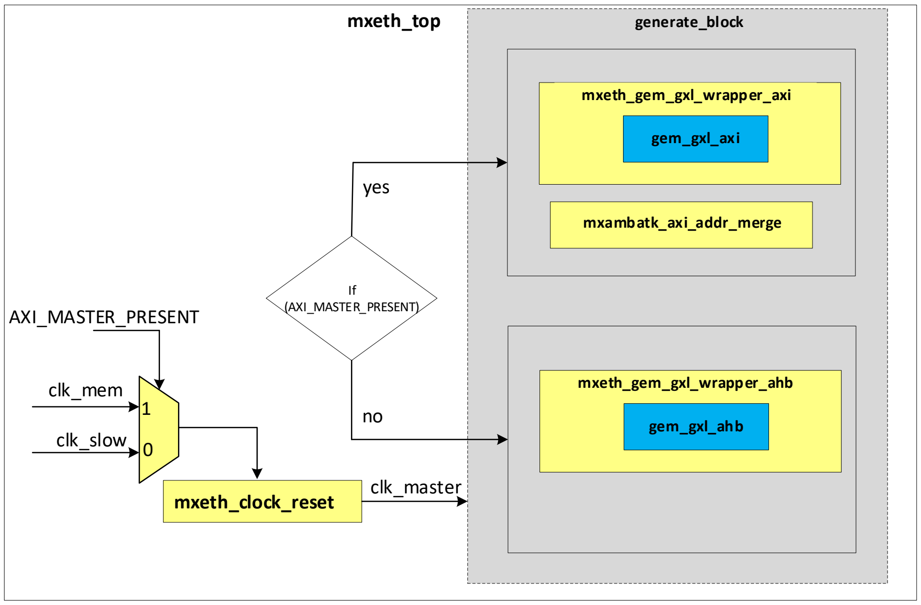


Figure 3-7 C**lock structure for AXI and AHB interface**

Other clock minimum frequency requirements:

- CLK_TSU – 5MHz
- CLK_SYS – 10MHz

3.4.1.7 Transmit and Receive Programming

3.4.1.7.1 SW programming to start transmission

a) Set MAC operation mode and enable IP

Configure MAC operation mode (MII/GMII/RGMII/RMII) and reference clock selection (if RMII mode is selected) by writing to "***CTL***" register. Enable IP, configure MAC operation speed by writing to "***network_config***" register. Note: MAC operation mode must be set before IP enabled, while MAC operation speed mode can only be programmed after IP enabled.

b) Enable transmit

Write "1" to register "***network_control.enable_transmit***".

c) Allocate an area of system memory for transmit data

This does not have to be contiguous, varying byte lengths can be used if they conclude on byte borders.

d) Program transmit buffer descriptor

Write buffer addresses to word zero of the transmit buffer descriptor entries and control information (the length of the frame, CRC append, last buffer in the frame) to word one. Add an extra descriptor to the end of the queue with its used bit set (bit 31 in word1 set to 1). This last descriptor in the queue may also have its wrap bit set (bit 1 in word 0 set to 1) in addition to its used bit. When transmit is enabled at least one entry in the buffer descriptor ring must have its used bit set. When the buffer descriptor ring is initialized for the first time there must be a used bit set before or at the buffer descriptor with the wrap bit. (refer to 3.4.1.1.1.6 for transmit buffer descriptor entry description).

e) Program transmit buffer for frame to be transmitted

Write data for transmission into the buffers pointed to by the descriptors. Details about transmit buffers and descriptors can be found in user guide page 58 ~ page 65.

f) Set transmit buffer descriptor pointer

Write address of transmit buffer descriptor list and control information to transmit buffer queue pointer registers (***transmit_q*_ptr***). When priority queues are used, each queue's buffer descriptor queue pointer should be updated and Q0's pointer should be updated last of all. Note 1) the transmit buffer queue pointer registers can only be updated when transmission is disabled or halted. 2) Transmit queues are default enabled. If only one queue is used, another 2 queues must be disabled through programming ***transmit_q*_ptr***.

g) Set transmission schedule

Select the transmission schedule algorithm by writing to "***tx_sched_ctrl***", the transmission schedule algorithm of each queue can be different. Default selection is fixed priority, also configure "***cbs_idleslope_q_****" is CBS is selected and configure "***bw_rate_limit_q0to3***" if DWRR/ETS is selected.

h) Interrupt setting

Enable appropriate interrupts by writing to "***int_enable***" or disable interrupt by writing to "***int_disable***". If priority queuing feature is enabled, this interrupt represents the interrupt dedicated to Q0, and there are dedicated interrupts for each priority queue and can be individually enabled/disabled by writing to "***int_q*_enable***"/ "***int_q*_disable***". Note that, only DMA related events are reported using the individual interrupt outputs. All other events are reported in the interrupt associate with Q0.

i) Start transmission

Write to the transmit start bit ("***tx_start_pclk***") in the network control register.

Once "*tx_start_pclk*" bit is set and synchronized to DMA clock (aclk) domain, DMA data fetch process is started from the address of internal transmit buffer queue pointer of the highest priority queue, the fetched frame data will be written into internal TX packet buffer (SRAM). At the end of the packet data, the status and statistics of that frame are also written into TX packet buffer. The internal transmit buffer queue pointer increments by two or four words after each buffer queue has been used, depending on the descriptor size (user guide page 58). The pointer is re-initialized to transmit buffer queue base address (*transmit_q*_ptr*) when the current descriptor has its wrap bit set. Detailed description about DMA fetch is given in YUJI-37, section 5.

After 1st packet has been completely fetched out from system memory and stored in internal packet buffer, transmission on MAC bus is started. From then on, other frames fetching and current packet transmission are operated in parallel. Note that, The DMA will only attempt to read more frame from the system memory when space is available in the internal packet buffer memory. If space is not available it must wait until a packet fetched by the MAC completes transmission and is subsequently removed from the packet buffer memory.

After a frame has been transmitted, the status bits are written back to the word one of the first buffer descriptor along with the used bit. Interrupt (bit 7-transmit complete) is set. Transmit status can be read though register "*transmit_status*", bit 3 "*transmit_go*" indicates the transmission is ongoing, bit 3 "*transmit_complete*" indicates a frame has been transmitted. Individual bits of this status register may be cleared by writing 1 to them.

DMA data fetching is halted when all queues descriptors used bit are read. The transmit buffer queue pointer will maintain its value. Therefore, when transmission is restarted, the next descriptor read from the queue will be from immediately after the last successfully transmitted frame.

Transmission Programming Example:

1) Configure MXETH MMIO registers

```
//Common settings. The configuration for CTL register and network_config register.
/////////MII configuration begin
//Note: In MII mode, REF_CLK is not required
CTL        = (0 << 0)  ;              //MII mode
CTL        = ((0 << 0)
           |(1 << 31));               //Enable IP

network_config = ((0<<0)      //0: 10M, 1:100M
                 |(1<<1)      //Full duplex mode
                 |(1<<4)      //Copy all frames
                 |(0<<10)      //Gigabit mode disable, 1: 1G, 0: 10M/100M
                 |(1<<17)      //Remove FCS field when writing frames to memory
                 |(3<<18)      //MDC clock division: 48
                 |(1<<21));    //64-bit DMA bus width
/////////MII configuration end

/////////GMII configuration begin
//Note: In GMII mode, REF_CLK must be routed from HSIO with FREQ=125MHz.
CTL        = (1 << 0) ;   //GMII mode
CTL        = ((1 << 0)
             |(0 << 2)     //REF_CLK from HSIO
             |(0 << 8)     //Division: REFCLK_DIV+1=1
             |(1 << 31));  //Enable IP

network_config = ((1<<1)      //Full duplex mode
                 |(1<<4)      //Copy all frames
                 |(1<<10)     // Gigabit mode enable, 1: 1G, 0: 10M/100M
```

```
                         |(1<<17)    //Remove FCS field when writing frames to memory
                         |(3<<18)    //MDC clock division: 48
                         |(1<<21));  //64-bit DMA bus width
/////////GMII configuration end

/////////RGMII configuration begin
//Note: In RGMII-1G mode, REF_CLK must be routed from HSIO with FREQ=125MHz.
//    In RGMII-10M/100M mode, REF_CLK can be routed from HSIO or internal PLL, in the following example, we suppose
REF_CLK is routed from HSIO with FREQ=50MHz
CTL         = (2 << 0)  ;   //RGMII mode
CTL         = ((2 << 0)
               |(0 << 2)     //REF_CLK from HSIO
              |(1 << 8)     //Division: REFCLK_DIV+1=2 for 100M, REFCLK_DIV+1=20 for 10M
             |(1 << 31));  //Enable IP

network_config = ((1<<0)      //0: 10M, 1:100M
                         |(1<<1)      //Full duplex mode
                         |(1<<4)      //Copy all frames
                         |(0<<10)    //Gigabit mode disable, 1: 1G, 0: 10M/100M
                         |(1<<17)    //Remove FCS field when writing frames to memory
                         |(3<<18)    //MDC clock division: 48
                         |(1<<21));  //64-bit DMA bus width
/////////RGMII configuration end
/////////RMII configuration begin
//Note: In RMII mode, REF_CLK can be routed from HSIO or internal PLL, in the following example, we suppose REF_CLK is
routed from PLL with FREQ=200MHz
CTL         = (3 << 0)  ;   //RMII mode
CTL         = ((3 << 0)
               |(1 << 2)     //REF_CLK from internal PLL
               |(3 << 8)     //Division: REFCLK_DIV+1=4
               |(1 << 31)); //Enable IP
```

2) Configure GEM_GXL registers

```
network_config = ((0<<0)      //0: 10M, 1:100M
                         |(1<<1)      //Full duplex mode
                         |(1<<4)      //Copy all frames
                         |(0<<10)    //Gigabit mode disable, 1: 1G, 0: 10M/100M
                         |(1<<17)    //Remove FCS field when writing frames to memory
                         |(3<<18)    //MDC clock division: 48
                         |(1<<21));  //64-bit DMA bus width
/////////RMII configuration end

dma_config    = ((4<<0)             //AXI burst length up to 4
                       |(3<<8)             //Uses full configured RX address space
                       |(1<<10)           //Uses full configured TX address space
                       |(0x18<<16));    //Receive buffer size: 1536 bytes
axi_max_pipeline = ((0x2<0)        //Max. outstanding transaction number of AR channel:2
                        |(0x2<<8));        //Max. outstanding transaction number of AW channel:2

//Enable transmit functionality
network_control   = 0x1<<3;        //Enable transmission

//Transmit buffer list allocation, refer to B.2
//Transmit buffer queue base address
transmit_q2_ptr    = ETH_Q2_TX_DESCR_BASE_ADDR;
transmit_q1_ptr    = ETH_Q1_TX_DESCR_BASE_ADDR;
transmit_q0_ptr    = ETH_Q0_TX_DESCR_BASE_ADDR;

//Transmission scheduler configuration
tx_sched_ctrl      = ((0x0<<0)       //Q0: Fixed priority enabled
                          |(0x1<<2)        //Q1: CBS enabled
                          |(0x1<<4));      //Q2: CBS enabled
cbs_idleslope_q_a    = (0x3B9ACA0<<0); //Q2 idleslope: 0x3B9ACA0
```

```
cbs_idleslope_q_b   = (0x1DCD650<<0); //Q1 idleslope: 0x1DCD650


//Enable interrupt
int_enable       = 0xffffffff;        //Enable all interrupts
int_q1_enable  = 0xffffffff;         //Enable all interrupts dedicated to Q1
…

//Start transmission
network_control   = ((0x1<<3)      //Enable transmission
                    |(0x1<<9));      //Start transmission
```

3) Transmit Buffer Allocation

Take Q0 as example, supposing all packets are single-buffer frame, and there are 5 frames to be transmitted on Q0. The transmit buffer entry size is 2 words as TX extended BD mode is not enabled and we are using 32-bit addressing mode. The descriptors are programmed in 32-bit mode, the buffers are programmed in 8-bit mode.

```
//Descriptor Entry, 32-bit programming mode
ETH_Q0_TX_DESCR_BASE_ADDR     = ETH_Q0_TX_BUFFER0_BASE_ADDR;      //Buffer0 word0, address
ETH_Q0_TX_DESCR_BASE_ADDR+0x4 = (0x00008000                       //Last buffer of the frame
                                 |BUFFER0_LENGTH);                //Buffer/Frame length
ETH_Q0_TX_DESCR_BASE_ADDR+0x8 = ETH_Q0_TX_BUFFER1_BASE_ADDR;      //Buffer1 address
ETH_Q0_TX_DESCR_BASE_ADDR+0xC = (0x00008000                       //Last buffer of the frame
                                 |BUFFER1_LENGTH);                //Buffer/Frame length
ETH_Q0_TX_DESCR_BASE_ADDR+0x10= ETH_Q0_TX_BUFFER2_BASE_ADDR;      //Buffer2 word0, address
ETH_Q0_TX_DESCR_BASE_ADDR+0x14= (0x00008000                       //Last buffer of the frame
                                 |BUFFER2_LENGTH);                //Buffer/Frame length
ETH_Q0_TX_DESCR_BASE_ADDR+0x18= ETH_Q0_TX_BUFFER3_BASE_ADDR;      //Buffer3 word0, address
ETH_Q0_TX_DESCR_BASE_ADDR+0x1C= (0x00008000                       //Last buffer of the frame
                                 |BUFFER3_LENGTH);                //Buffer/Frame length
ETH_Q0_TX_DESCR_BASE_ADDR+0x20= ETH_Q0_TX_BUFFER4_BASE_ADDR;      //Buffer4 word0, address
ETH_Q0_TX_DESCR_BASE_ADDR+0x24= (0x00008000                       //Last buffer of the frame
                                 |BUFFER4_LENGTH);                //Buffer/Frame length

//Add an extra descriptor with used bit (bit 31) set. If this is the last descriptor in the queue, also set wrap bit (bit 30).
ETH_Q0_TX_DESCR_BASE_ADDR+0x2C= 0x80000000;            //used bit set

//Transmit buffer initialization, 8-bit programming mode
ETH_Q0_TX_BUFFER0_BASE_ADDR    = Frame0_byte0;
ETH_Q0_TX_BUFFER0_BASE_ADDR+0x1= Frame0_byte1;
ETH_Q0_TX_BUFFER0_BASE_ADDR+0x2= Frame0_byte2;
…
ETH_Q0_TX_BUFFER0_BASE_ADDR+(m-1)= Frame0_bytem;   //m=BUFFER0_LENGTH-1


…
ETH_Q0_TX_BUFFER4_BASE_ADDR    = Frame4_byte0;
ETH_Q0_TX_BUFFER4_BASE_ADDR+0x1= Frame4_byte1;
ETH_Q0_TX_BUFFER4_BASE_ADDR+0x2= Frame4_byte2;
…
ETH_Q0_TX_BUFFER4_BASE_ADDR+(n-1)= Frame4_byten;   //n=BUFFER4_LENGTH-1
```

3.4.1.7.2 SW programming to start receiving

a) Configure MAC operation mode and enable IP

Configure MAC operation mode (MII/GMII/RGMII/RMII) and reference clock selection (if RMII mode is selected) by writing to "***CTL***" register. Enable IP, configure MAC operation speed by writing to “***network_config***” register. Note: MAC operation mode must be set before IP enabled, while MAC operation speed can only be programmed after IP enabled.

b) Allocate an area of system memory for receive data buffers

Refer to user guide page 52 about the receive buffer depth configuration (through writing to ***dma_config*** register for Q0 and ***dma_rxbuf_size_q**** for all other priority queues). The default receive buffer size of mxeth is 1536 bytes, which means the maximum length frame supported by mxeth (1536 bytes) can be stored into a single buffer by default.

c) Program receive buffer descriptor

Allocate an area of system memory for receive buffer descriptor list. Set-up the receive buffer list by writing buffer addresses to word 0 of the receive buffer descriptor, mark all descriptor entries as owned by mxeth, i.e. bit 0 of word 0 set to 0. Add an extra descriptor to the end of the queue with its used bit set (bit 0 in word 0 set to 1). This last descriptor in the queue may also have its wrap bit set (bit 1 in word 0 set to 1) in addition to its used bit. When receive is enabled at least one entry in the buffer descriptor ring needs its used bit set so it is not sufficient to set the wrap bit of the last buffer in the queue without also setting its used bit. (Refer to user guide page 417).

d) Set receive buffer descriptor pointer

Write start address of receive buffer descriptor list and control information to receive buffer queue pointer register (***receive_q*_ptr***). Note, these registers must be written before receiving is enabled.

e) Program frame receiving filter

Refer to user guide page 92 ~ page 96 for filter functionality, to see cases in which the frame will be written into system memory. Then determines how to configure related registers. ″ Generally, frame will be written into system memory if “***network_config.copy_all_frames***” is set; ″ Otherwise, frame is written into system memory when a specific address match (***spec_add1_bottom ~ spec_add4_bottom, spec_add1_top ~ spec_add4_top***) or a type ID match (***spec_type1 ~ spec_type4***) or a hash address match (***hash_bottom, hash_top***) is found, this is achieved by the filter functionality. Refer to user guide page 92~page 94 for details. ″ Frame is written to system memory if it’s a broadcast address (destination address=0xFFFFFFFFFFFF) and “***network_config.no_broadcast***” is disabled. ″ Pause frame is NOT written to system memory if “***network_config.disable_copy_of_pause_frames***” is set, regardless of the copy all frames bit, whether a hash match is found, a type ID match (0x8808) is identified or if a destination address (0x010000c28001) match is found.

f) Program screener to route frame to specific Queue

Refer to user guide page 74~page 76 about the screener functionality, which allows user to route received frames to different priority queues. Note that the received frame is routed to queue 0 by default. In mxeth, there are 16 type 1 screeners (***screening_type_1_register_0 ~ screening_type_1_register_15***) and 16 type2 screeners (***screening_type_2_register_0 ~ screening_type_2_register_15***). For type 2 screener, 8 ethertype matches (***screening_type_2_ethertype_reg_0~ screening_type_2_ethertype_reg_7***) and 32 compare matches are supported (***type2_compare_0_word_0~ type2_compare_0_word_15 and type2_compare_0_word_1 ~ type2_compare_15_word_1***).

g) Jumbo frame configuration

If the frame length to be received is larger than 1518bytes, then write to “***network_config.jumbo_frames***” or “***network_config.receive_1536_byte_frames***” to enable frame length up to 1536bytes to be accepted (mxeth supports maximum frame length of 1536 bytes).

h) Other features for frame receiving

Refer to user guide (mainly page 90~page 92) for other reception related features, such as FCS remove, discard non-VLAN frames and receive frame’s checksum offload, etc.

i) Interrupt setting

Enable appropriate interrupts by writing to “***int_enable***” or disable interrupt by writing to “***int_disable***”. “***int_status***” is used to ascertain the interrupt source. If priority queuing feature is enabled, this interrupt represents the interrupt dedicated to Q0, and there are dedicated interrupts for each priority queue and can be individually enabled/disabled by writing to "***int_q*_enable***"/ "***int_q*_disable***". Note that, only DMA related events are reported using the individual interrupt outputs. All other events are reported in the interrupt associate with Q0.

j) Enable receiving

Enable receive by writing to "***network_control.enable_receive***".

If the filter block indicates that the frame should be copied to memory, frame data will be written into RX packet buffer (SRAM) during frame reception. At the end of the received frame, the status and statistics of that frame (such as whether it is an errored frame, which queue the frame is destined, the frame length, etc.) are also written into the RX packet buffer. As mxeth only supports full store and forward mode, the DMA only begin packet fetches once the status and statistics for a frame are available.

A receive overrun condition occurs when the RX packet buffer is full. If this occurs subsequent packet data will be dropped and a receive overrun interrupt is raised. The frame currently being written will be flushed from the packet buffer to recover the buffer, allowing subsequent frames to utilize the freed-up packet buffer space. Also, if the frame has an error, such as FCS error, frame length error, the frame data will be flushed for packet buffer recovery, but the status and statistics of frame are always held regardless of the frame is errored or not.

If a frame is received without errors, and meets the requirements to be copied to system memory, the frame data will be written to receive buffers that receive buffer queue pointer points to by DMA.

Once a frame has been completely and successfully received and written to system memory. The receive buffer descriptor entry will be updated with the reason for address match and other information. The owner bit of word 0 is set. Once this is complete, a receive complete interrupt(bit 1) is set. "*receive_status.receive_complete*" is also set.

The internal receive buffer queue pointer increments by two or four words after each buffer queue has been used, depending on the descriptor size (user guide page 52). The pointer is re-initialized to receive buffer queue base address (*receive_q*_ptr*) if the current descriptor has its wrap bit set. User guide page 57 has detailed description about the process of received frame, receive buffer descriptor and receive status.

Receiving Programming Example:

1) Configure MXETH MMIO registers

Same as transmission programming.

2) Configure GEM_GXL registers

```
dma_config    = ((4<<0)           //AXI burst length up to 4
                |(3<<8)           //Uses full configured RX address space
                |(1<<10)          //Uses full configured TX address space
                |(0x18<<16));     //Receive buffer size: 1536 bytes
```

```
axi_max_pipeline = ((0x2<0)        //Max. outstanding transaction number of AR channel:2
                   |(0x2<<8));     //Max. outstanding transaction number of AW channel:2

//Receive buffer list allocation, refer to A.2
//Receive buffer queue base address
receive_q2_ptr    = ETH_Q2_RX_DESCR_BASE_ADDR;
receive_q1_ptr    = ETH_Q1_RX_DESCR_BASE_ADDR;
receive_q0_ptr    = ETH_Q0_RX_DESCR_BASE_ADDR;

//Filter functionality
//Address match
spec_add1_bottom = 0x56789abc;          //Specific address1: 0x123456789abc
spec_add1_top    = 0x00001234;          //Specific address1: 0x123456789abc
…
//Type match
spec_type1     = ((0x8801<<0)           //Type ID match1: 0x8808
                 | (1<<31));            //Enable copy of matched frames
…
//Screener functionality
//Type 1 screener
screening_type_1_register_0 = ((0x0<<0)          //Route matched frame to Q0
                              |(0x12<<4)         //DS/TC match value:0x12
                              |(0x1<<28));       //DS/TC match enable
…
//Type 2 screener
screening_type_2_register_0 = ((0x0<<0)          //Route matched frame to Q0
                              |(0x0<<9)          //Compare w/ Screener type2 Ethertype register0
                              |(0x1<<12));       //Ethertype match enable
screening_type_2_ethertype_reg_0 = 0x0800<<0;    //Ethertype match value:0x0800

//Enable interrupt
int_enable       = 0xffffffff;       //Enable all interrupts
int_q1_enable    = 0xffffffff;       //Enable all interrupts dedicated to Q1
…

//Enable receive functionality
network_control   = 0x1<<2;          //Enable receive
```

3) Receive Buffer Allocation

Take Q0 as example, supposing 5 receive buffers are allocated for Q0. The receive buffer entry size is 2 words as RX extended BD mode is not enabled and we are using 32-bit addressing mode. The buffer/descriptor are programmed in 32-bit mode.

```
ETH_Q0_RX_DESCR_BASE_ADDR     = ETH_Q0_RX_BUFFER0_BASE_ADDR;   //Buffer0 word0, address
ETH_Q0_RX_DESCR_BASE_ADDR+0x8 = ETH_Q0_RX_BUFFER1_BASE_ADDR;   //Buffer1 word0, address
ETH_Q0_RX_DESCR_BASE_ADDR+0x10= ETH_Q0_RX_BUFFER2_BASE_ADDR;   //Buffer2 word0, address
ETH_Q0_RX_DESCR_BASE_ADDR+0x18= ETH_Q0_RX_BUFFER3_BASE_ADDR;   //Buffer3 word0, address
ETH_Q0_RX_DESCR_BASE_ADDR+0x20= ETH_Q0_RX_BUFFER4_BASE_ADDR;   //Buffer4 word0, address
//Add an extra descriptor with used bit(bit 0) set. If this is the last descriptor in the queue, also set wrap bit(bit 1).
ETH_Q0_RX_DESCR_BASE_ADDR+0x28= 0x1;                           //used bit set
```

3.4.1.7.3 Timing requirement for SW programming

3.4.1.7.3.1 Latency for enable_receive to take effect after IP is enabled

Programming enable_receive will not take effect immediately as there are multiple levels of cross domain synchronous logic in the path from enable_receive register to internal receive logic. 2 factors need to be considered:

1) T0 – Time from CTL.ENABLED set “1” to MAC rx_clk presents

   enable_receive only works when rx_clk presents and IP disable will switch off rx_clk. After IP is enabled, there will be multiple rx_clk cycles delayed for rx_clk presents inside MAC. The number of cycle is determined by different ETH mode.

2) T1 – Time from enable_receive set “1” to receiving logic ready for work

   Enable_receive is pclk domain register, and it needs to be synchronized to rx_clk domain for frame receiving control.

List T0 and T1 as per different MAC operation mode in the following table.

	MII 10M	**MII 100M**	**RMII 10M**	**RMII 100M**	**GMII 1G**	**RGMII 10M**	**RGMII 100M**	**RGMII 1G**
T0(ns)	2000	200	2100	2100	40	2800	280	56
T1(ns)	800	80	800	80	16	800	80	16

Table 12: Timing requirement for enabling receive

Based on above timing requirement, user must ensure enough latency to allow MAC starting to receive first frame, after IP enabled and enabling receiving.

3.4.1.7.3.2 Latency for *receive_q*_ptr* to take effect

In IP implementation, writing receive buffer queue base address register could consume 3 AXI clock cycles to take effect. Therefore, reception can't be enabled until 3 AXI clock after receive buffer queue base address register is updated. This restriction need to be taken care by SW.

3.4.1.8 Interrupt

There are multiple interrupt conditions that are detected within the GEM_GXL. These are ORed to make a single interrupt (or multiple interrupts if priority queuing is enabled). Depending on the overall system design this may be passed through a further level of interrupt collection (interrupt controller). On receipt of the interrupt signal, the CPU shall enter the interrupt handler. Refer to your documentation for the interrupt controller to identify that it is the GEM_GXL that is generating the interrupt. To ascertain which interrupt, read the interrupt status register. Note that in the default configuration this register will clear itself after being read, though this may be configured to be write-one-to-clear if desired.

At reset all interrupts are disabled. To enable an interrupt, write to interrupt enable register with the pertinent interrupt bit set to 1. To disable an interrupt, write to interrupt disable register with the pertinent interrupt bit set to 1. To check whether an interrupt is enabled or disabled, read interrupt mask register: if the bit is set to 1, the interrupt is disabled.

The IP implements an interrupt status register for each interrupt. Interrupt 0 register is located at IP's internal register address 0x24 while the interrupt status registers are located at address starting from 0x400.

3.4.1.9 TSU compare output

GEM_GXL TSU timer compare valid is wired out the block.

TSU timer comparison valid, synchronized to tsu_clk or pclk. Asserted high when upper 70 bits of TSU timer count value are equal to programmed comparison value. Details see 3.4.1.1.8.5.

3.4.2 Off-chip Interfaces

3.4.2.1 MII Interface

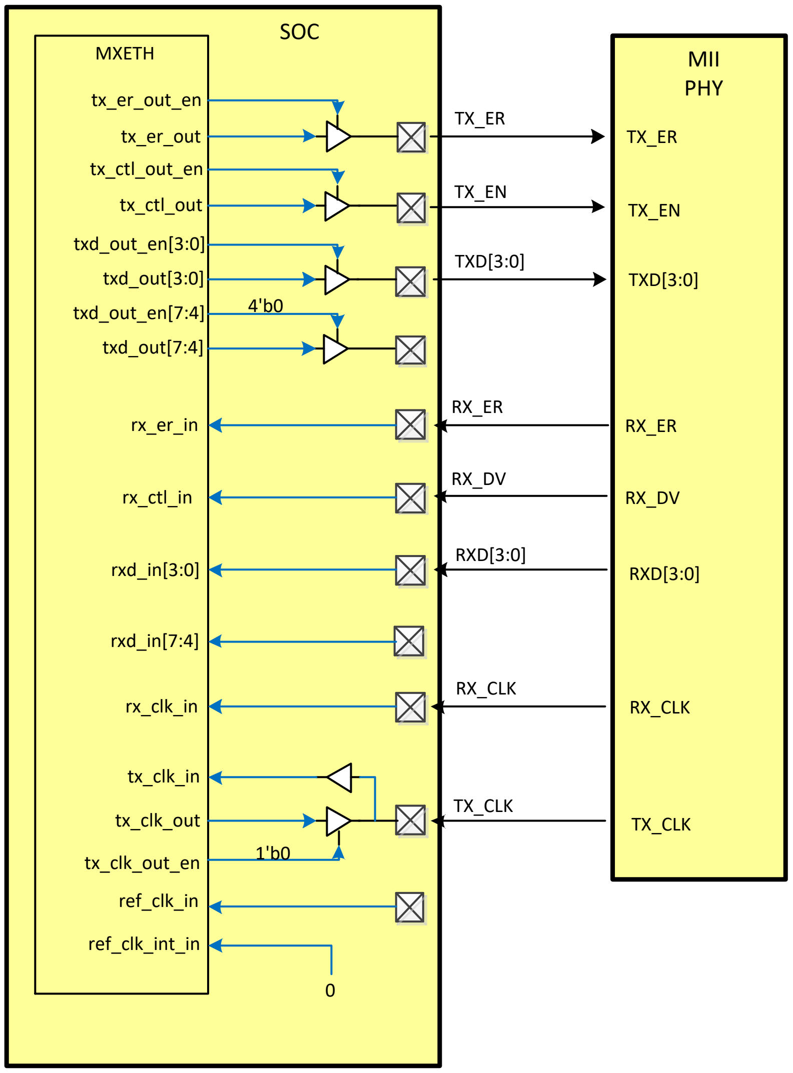


Figure 3-8: PHY connection in MII mode

3.4.2.2 RMII interface

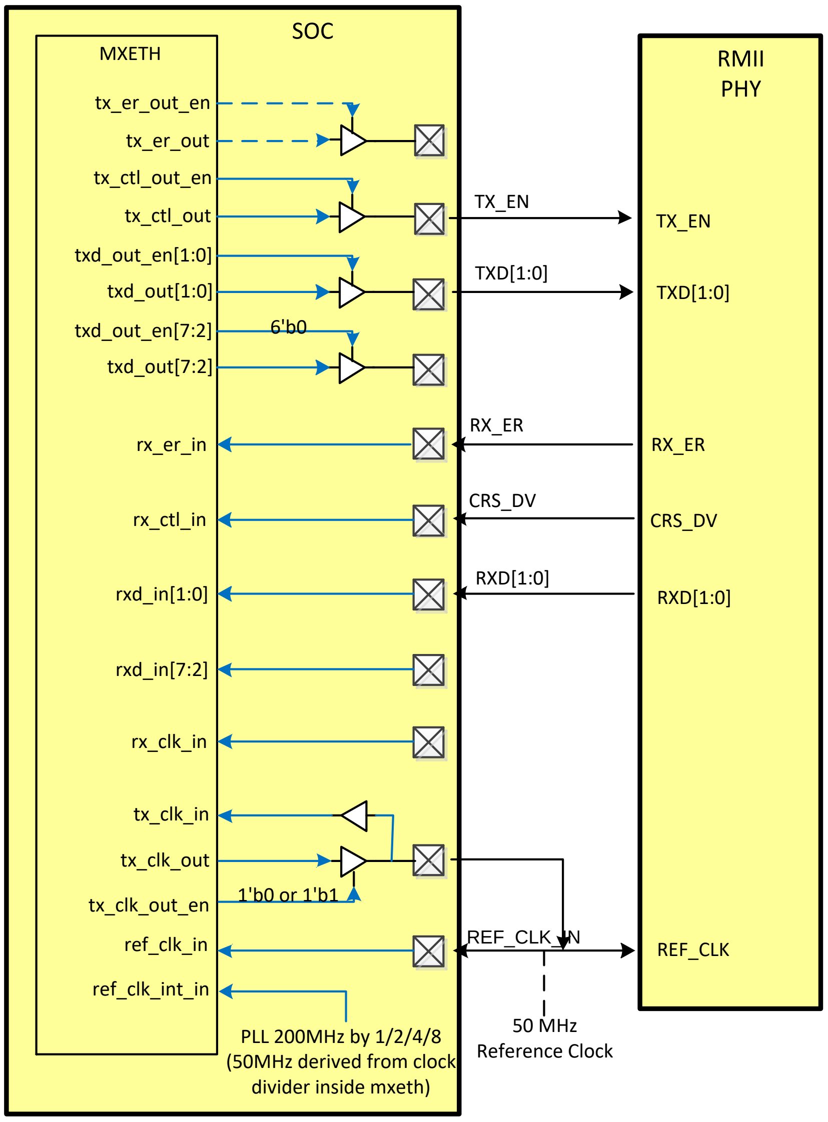


Figure 3-9: PHY connection in RMII mode

3.4.2.3 GMII interface

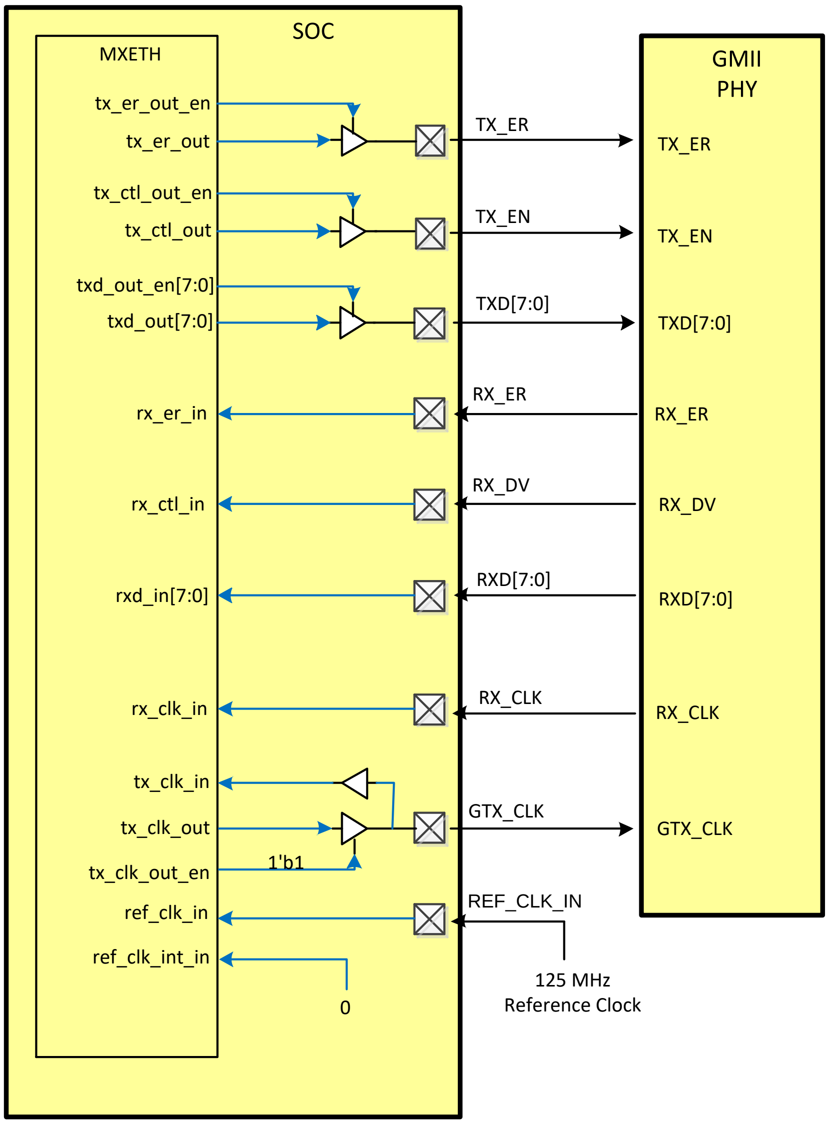


Figure 3-10: PHY connection in GMII mode

3.4.2.4 RGMII interface

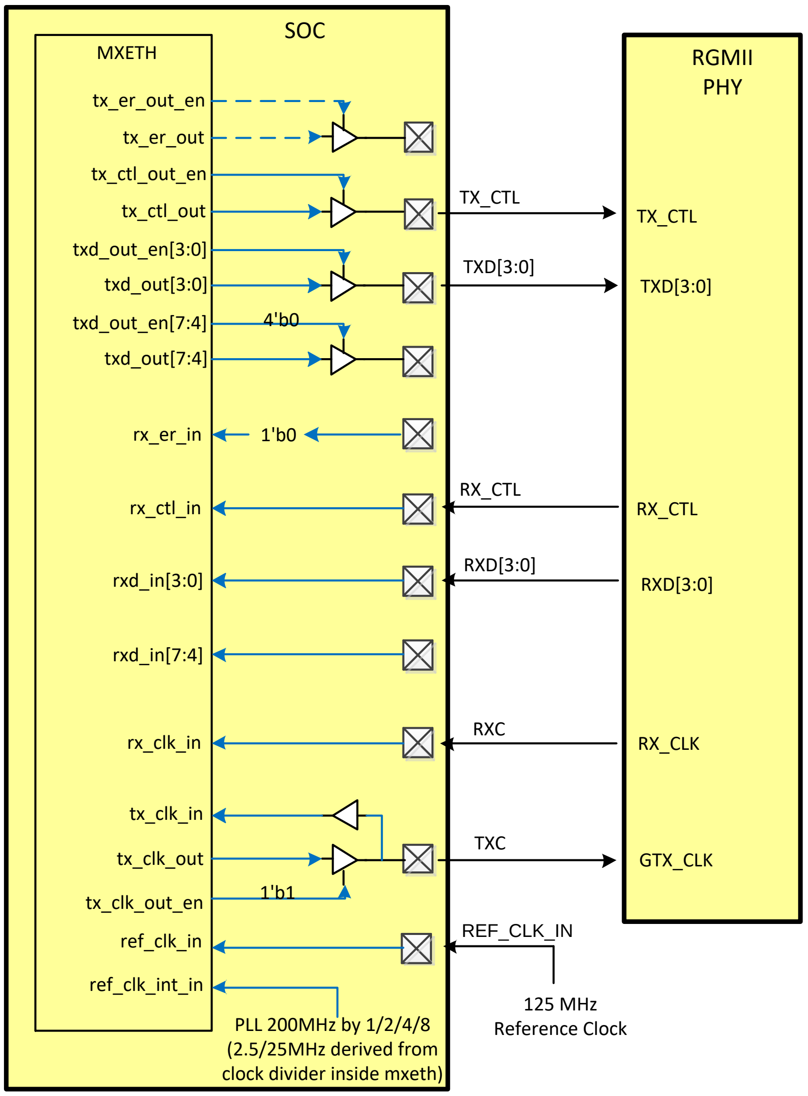


Figure 3-11: PHY connection in RGMII mode

3.4.3 Memory Map, Register Definitions, Firmware

The initial version of the SAS memory map is created by Cadence AE as per our GEM_GXL configuration. Based on that, we have made some modifications to fit our HOBTO flow and further feature set changing.

There are several types of registers in GEM_GXL. They are summarized in below table:

Reg Type	AHB Response	Comment
Normal Reg	OK	Registers used for block functions
Unused Reg	OK	Registers can be accessed. These registers are not used

		because the functions are not enabled through GEM_GXL design directive, but they are not removed from design.
Removed Reg	AHB ERROR	Registers can not be accessed. They are not used and removed from design.
Hidden Reg	OK	Registers exist but not listed in GEM_GXL datasheet.

Table 13: GEM_GXL register attributes

Details can be found in SAS memory map MXS40-IP-ETH.xls.

3.4.3.1 MMIO Register Access Restriction

- After the IP enable (CTL.ENABLED="0"→"1"), TX clock (TX_CLK) and RX clock (RX_CLK) are gated for up to 14 cycles of each clock. MMIO registers can be accessed after both clocks are available. MMIO registers must not be accessed during 14 cycles of both clocks after the IP enable.

  Note: In GMII, RGMII and RMII modes, reference clock (REF_CLK) need to be supplied before the IP enable. Otherwise, MMIO registers must not be accessed during 14 cycles of TX_CLK and TX_CLK after REF_CLK is supplied.

- The transmit must be enabled (*network_control[3]* = 1), before the transmit start is set (*network_control[9]* = 1).

3.4.4 Instruction Set

MXS40 platform support little endian only. GEM_GXL IP support programmable endian.

3.4.5 Recovery

In case of unexpected states or prohibited register settings, mxeth will not work properly. Reset IP through disabling will reset IP to default state.

3.4.6 Initialize

Following table lists the default state of reset, disable and low power mode.

	IP disable /	Deep Sleep	Hibernate

	Reset		
mxeth configure register	Retain	Retain	“X”
gem_gxl configuration register	Default	“X”	“X”
status register	Default	“X”	“X”
txd_out[7:0] tx_er_out tx_ctl_out	Low	“X”	“X”
txd_out_en[7:0] tx_er_out_en tx_ctl_out_en	Low	“X”	“X”
tx_clk_out	ref_clk_in	“X”	“X”
tx_clk_out_en	Low	“X”	“X”
interrupt	Low	“X”	“X”
AHB output	Still work	“X”	“X”
AXI output	Default state	“X”	“X”

Table 14: IP status after reset/enable/wakeup

3.4.7 Pin Status

Output enable signal will be high once the IP is enabled. Thus, if there is any unused or illegal configuration, IP may output unwanted transaction on PHY interface and AXI bus (or AHB bus). It is design specific and not listed in gem_gxl datasheet.

4. SOFT IP: INTEGRATION SPECIFICATION

This chapter contains internal Cypress documentation of how to integrate this block into a subsystem and product. This comprises the block pin list, on-chip interfaces, integration guidelines, design metrics, and characterization/test information.

The integration deliverables for this IP are listed in IP Deliverables Worksheet in the Appendix.

4.1 Pins

Block pins for public cells are described in the following table:

SoftIP_BlockPinList

4.2 Parameters and IP-reuse

3PIP GEM_GXL has many parameterization options to configure the IP during compilation stage. This is achieved using Verilog `define compiler directives in an include file called gem_gxl_defs.v. To aid in debug, these configurations are readable through the AHB address space starting from address 0x280.

mxeth also provides multiple parameters in wrapper design to select different SRAM type and size, presence of reference clock divider, select between AXI and AHB master interface and TSU clock. Refer to SAS memory map MXS40-IP-ETH.xlsm for details.

4.2.1 Configure of TX/RX packet buffer

128 bit data width single port ARM SRAM or SNPS SRAM can be selected in the block as TX/RX packet buffer. The size of SRAM is configurable. Several SRAM size options are provided through design time parameter for further possibility of SRAM area saving by reducing queue number or depth of each packet buffer. The SRAM size is determined by

- Queue number;
- Partial store or full store mode;
- maximum frame size;

The current configuration of mxeth is:

- 3 queues;
- Support both full store and forward mode and partial store and forward mode;

- Max 1536 jumbo frame;
- 8KB TX packet buffer and 4KB RX packet buffer;

Following 2 tables depict how to configure packet buffer size with these 2 parameters and GEM_GXL design directives.

TX_PACKET_BUFFER_SIZE	TX packet buffer SRAM Size	gem_gxl directive `gem_tx_pbuf_addr	Queue number is determined by gem_gxl design directive. It's recommended to remain 4K (buffer 2 packet) or 2K (buffer 1 packet) for each transmit queue.
2'b00	16K	10	
2'b01	8K	9	
2'b10	4K	8	
2'b11	2k	7	

Following directives need to be proper defined for queue number, segment size and segment number.
`` `define dma_priority_queue1 ``
`` `define dma_priority_queue2 ``
`` `define dma_priority_queue3 ``
`` `gem_tx_pbuf_queue_segment_size ``
`` `gem_tx_pbuf_num_segments_q0 ``
`` `gem_tx_pbuf_num_segments_q1 ``
`` `gem_tx_pbuf_num_segments_q2 ``
`` `gem_tx_pbuf_num_segments_q3 ``			

Table 15: TX packet buffer size configuration

RX_PACKET_BUFFER_SIZE	RX packet buffer SRAM Size	Gem_gxl directive `gem_rx_pbuf_addr	RX buffer size is independent to queue number. It's recommended to remain 4K (buffer 2 packet) or 2K (buffer 1 packet) for receive.
2'b00	4K	8	
2'b01	2K	7	

Table 16: RX packet buffer size configuration

Current mxeth configuration is highlighted green.

4.2.2 Configure of DMA priority queues and segments

To support priority queues, RAM space will be split into equal sized segments. The size of the segments are configurable using the define `` `gem_tx_pbuf_queue_segment_size``. `` `gem_tx_pbuf_queue_segment_size`` only needs to be set if priority queue is enabled and defines how many of the SPRAM address bits specified in `` `gem_tx_pbuf_addr``. `gem_tx_pbuf_queue_segment_size` will be used for the number of segments. `` `gem_tx_pbuf_num_segments_qx `` will be used to define segments number for each queue. Current mxeth configuration is

```
`define dma_priority_queue1
`define dma_priority_queue2
`define gem_tx_pbuf_queue_segment_size 2 // Eight segments,
`define gem_tx_pbuf_num_segments_q0 1 // Queue 0 uses 2 segments, Queue 1
and 2 uses 1 segment.
```

`gem_tx_pbuf_num_segments_q1 and `gem_tx_pbuf_num_segments_q2 are not defined as they only use 1 segment.

4.2.3 gem_gxl configuration

Define	Description	Reasoning
`define gem_no_pcs	Define if PCS is included.	No PCS needed. Using external PHY that includes PCS logic.
//`define gem_pcs_legacy_if //`define gem_pcs_10b_if //`define gem_pcs_20b_if	PCS related defines.	No PCS needed and hence, undefined.
// `define gem_use_rgmii	Select whether to expose GMII/MII or an RGMII interface.	Select only to expose GMII/MII (8bit interface) and copied the RGMII module to the mxeth_wrapper to select the interface in runtime instead of design time.
`define gem_include_rmii	Select whether an additional RMII interface is included. Uncomment to include a separate RMII port.	Support RMII as well and hence keep this define.
// `define gem_int_loopback	Uncomment for internal loopback	No loop back support in MAC. Loop back is expected to be supported in PHY.
// `define gem_ext_fifo_interface	Include low latency FIFO interface only (no DMA).	Comment out as DMA is used.
// `define gem_host_if_soft_select	Define to add a software programmable option to statically select between a low latency FIFO host interface and the SRAM based packet buffer DMA.	Comment out as DMA is used and no desire to use low latency FIFO.
// `define gem_tx_add_fifo_if	Define to add a TX FIFO interface in addition to AXI/AHB DMA.	Comment out as the 4 TX queue size should be able to handle streaming instead of having additional fifo to internally buffer the data packets.
// `define gem_user_io `define gem_user_out_width 0	Define user I/O widths	Comment out since there is no usage on programmable input nor output.

`define gem_user_in_width 0		
// `define gem_no_stats	Define whether the GEM stats registers are required	Commented out as GEM statistic registers are required for debug.
`define gem_no_snapshot	Define whether snapshot statistic registers are required.	Defined as no snapshot as there is no need for snapshot of the statistic register. This saves 984 flops.
// `define gem_irq_read_clear	Define to cause interrupt status register is cleared upon read. Comment out to cleared interrupt status register when writing 1 to the respective bit.	Comment out as it is conventional to have write 1 to clear for interrupt status bit.
`define gem_jumbo_max_length 14'd1536	Define max size jumbo frames.	The value 1536 is used instead of 16383 as only the maximum length and not jumbo size is expected for automotive usages.
`define num_spec_add_filters 4	Define the number of maskable specific address filters required.	Keep as default.
`define gem_rx_pipeline_delay 10	Define the length of the pipeline used in gem_rx to delay received data and allow time for address matching.	Keep as default.
`define gem_emac_bus_width_axi 64	Define EMAC bus width	64 bits to match the AXI bus width.
`deine gem_emac_bus_width_ahb 32	Define EMAC bus width	32 bits to match the AHB bus width
`define gem_dma_bus_width_axi 64	Define DMA bus width	64 bits to match the AXI bus width.
`define gem_dma_bus_width_ahb 32	Define DMA bus width	64 bits to match the AHB bus width.
`define gem_dma_addr_width_axi 32	Define DMA address width	32 bits to match AXI infrastructure
`define gem_dma_addr_width_ahb 32	Define DMA address width	32 bits to match AHB infrastructure
`define gem_hprot_value 4'b0001	Define value of hprot when driving request to AHB.	Not applicable. Does not matter for mxeth as it is only using AXI at request.
`define gem_axi_axi	Define if DMA should use native AXI or AHB	Define as AXI
`define gem_axi_ahb	Define if DMA should use native AXI or AHB	Define as AHB
`define gem_axi_access_pipeline_bits 4'h1	Define AXI DMA pipelining bits that determines the depth on request.	Define as 1 to set the pipelining depth to 2. This for descriptor writebacks and RX data writes to the system memory.

`define gem_axi_tx_descr_rd_buff_bits 4'h1	Define AXI TX descriptor read buffer number of bits to determine the depth of the buffer.	Define as 1 to set the TX descriptor read buffer depth as 2. This buffer stores the descriptors after reading it from system memory while waiting for DMA to fetch the data to the TX SRAM.
`define gem_axi_rx_descr_rd_buff_bits 4'h1	Define AXI RX descriptor read buffer number of bits to determine the depth of the buffer.	Define as 1 to set the RX descriptor read buffer depth as 2. This buffer stores the descriptor after reading it from system memory while waiting for DMA to fetch data from RX SRAM.
`define gem_axi_tx_descr_wr_buff_bits 4'h1	Define AXI TX descriptor write buffer number of bits to determine the depth of the buffer.	Define as 1 to set the TX descriptor write buffer depth as 2. This buffer stores the information for writeback after data is transmit out at ethernet line from TX SRAM.
`define gem_axi_rx_descr_wr_buff_bits 4'h1	Define AXI RX descriptor write buffer number of bits to determine the depth of the buffer.	Define as 1 to set the RX descriptor write buffer depth as 2. This buffer stores the information for writeback after data is transmit out of RX SRAM to system memory.
`define gem_axi_prot_value 3'b010	Define value for arprot and awprot AXI request interface.	Define as 3'b010 to indicate, normal, non-secure, and data access type since the write request are for descriptor writebacks and RX data writes. Whereas the read requests are for reading descriptors and TX data.
`define gem_axi_awcache_value 4'b0000	Define value for awcache AXI output.	Keep as default (non-cacheable and non-bufferable).
`define gem_axi_arcache_value 4'b0000	Define value for arcache AXI output.	Keep as default (non-cacheable and non-bufferable).
`define gem_rx_pkt_buffer `define gem_tx_pkt_buffer	Define RX and TX packet buffer respectively	Define both RX and TX packet buffer to indicate external buffer (external SRAM) to store the TX/RX data.
`define gem_spram	Define packet buffer mode of operation to use SPRAM or DPRAM.	Define as SPRAM since the frequency of AXI clock is more than 2x of MAC data rate.
`define gem_rx_pbuf_data 128 `define gem_tx_pbuf_data 128	Define the RX and TX packet buffer SRAM data width.	Define as 128 as per recommendation if SPRAM is used.
`define gem_rx_pbuf_addr 8	Define the width of RX packet buffer address	Define as 8 (depth of 256) to access 4KB (128bits x 256).

`define gem_tx_pbuf_addr 9	Define the width of TX packet buffer address	Define as 9 (depth of 512) to access 8KB (128bits x 512). There is 1 physical TX SRAM but split to 4 TX queue logically.
`define dma_priority_queue1 `define dma_priority_queue2 //`define dma_priority_queue3 // `define dma_priority_queue4 // `define dma_priority_queue5 // `define dma_priority_queue6 // `define dma_priority_queue7 // `define dma_priority_queue8 // `define dma_priority_queue9 // `define dma_priority_queue10 // `define dma_priority_queue11 // `define dma_priority_queue12 // `define dma_priority_queue13 // `define dma_priority_queue14 // `define dma_priority_queue15	Define the number of priority queues.	Define for 3 priority queues. Provide flexibility for SW to use up to 3 priorities for different uses. As a Bridge port that is attached to end station, it is only required to have 2 queues to support AVB. However, another queue can be added to provide network control packets isolated from user data traffic. Please see back up on reason section on snippet of IEEE 802.1Q-2014.
`define gem_pbuf_cutthru	Define for partial store and forward mode (cutthru)	mxeth provides flexibility on SW choosing partial store and forward or full store and forward.
// `define gem_pbuf_rsc	Define for enabling Receive Side Coalescing	Comment out. Not supported.
// `define gem_pbuf_lso	Define for Large Segment Offload	Comment out. Not supported.
`define gem_tx_pbuf_queue_segment_size 2	Define for TX packet buffer queue segment size	Define as 1 to indicate the TX packet buffer (SRAM) is divided to 4 segments each holding 1/4 of the total SPRAM space.
`define gem_tx_pbuf_num_segments_q0 1 //`define gem_tx_pbuf_num_segments_q1 0 //`define gem_tx_pbuf_num_segments_q2 0 //`define gem_tx_pbuf_num_segments_q3 0 //`define gem_tx_pbuf_num_segments_q4 0 //`define gem_tx_pbuf_num_segments_q5 0 //`define gem_tx_pbuf_num_segments_q6 0 //`define gem_tx_pbuf_num_segments_q7 0 //`define gem_tx_pbuf_num_segments_q8 0 //`define gem_tx_pbuf_num_segments_q9 0 //`define gem_tx_pbuf_num_segments_q10 0	Define how many segments are located to each queue. The number of bits are in power of 2.	Q0 will have 1/2 of the total SPRAM space. Q1 and Q2 will have 1/4 each. Providing the lowest priority with the bigger space as higher priority with smaller chunks can go through first.

//`define gem_tx_pbuf_num_segments_q11 0 //`define gem_tx_pbuf_num_segments_q12 0 //`define gem_tx_pbuf_num_segments_q13 0 //`define gem_tx_pbuf_num_segments_q14 0 //`define gem_tx_pbuf_num_segments_q15 0		
`define num_type1_screeners 8'd16 `define num_type2_screeners 8'd16	Define the screening algorithm the receive direction used to identify receive queue information.	Defining type 1 to have 16 and type 2 to have 16. Provides flexibility to application to map 16 different types of VLAN to the 4 RX Queue. This matches TVI.
`define num_scr2_ethtype_regs 8'd8	Define how many ethertype matches. Only valid if num_type2_screeners are enabled.	Defining 8 ethtype matches.
`define num_scr2_compare_regs 8'd32	Define how many field compare matches. Only valid if num_type2_screeners are enabled.	Defining 32 compare register.
// `define gem_exclude_cbs	Define whether to include credit based shaper algorithm for the top 2 queues (TX queue) per 802.1Qav.	Comment out to include CBS feature.
`define gem_rx_fifo_size 10 `define gem_rx_base2_fifo_size 4'b1010 `define gem_rx_fifo_cnt_width 4 `define gem_tx_fifo_size 10 `define gem_tx_base2_fifo_size 4'b1010 `define gem_tx_fifo_cnt_width 4	Defines specific to FIFO basd DMA implementation. Defines are ignore d if `gem_rx_pkt_buffer is defined.	Not applicable
`define gem_tsu	Define to enable tsu feature	Define to enable TSU.
`define gem_pfc_multi_quantum	Define for 8 pause priority quantums	Define to enable PFC
`define gem_revision_reg_value 32'h00070109	Define for module revision	Keep as default since it is indication of GEM_GXL revision.
`define gem_phy_id_top 16'h0007 `define gem_phy_id_bot 16'h0109	Define for PCS id.	Not applicable as PCS is not enabled.
`define gem_dma_bus_width_def 2'b01	Define DMA bus width in network config register. 2'b00 – 32 bits 2'b01 – 64 bits 2'b10 – 128 bits	Define as 2'b01 for 64 bits access.

`define gem_mdc_clock_div 3'b011	Define default value for mdc_clock_div in the network config register.	Define as 3'b011 as divide pclk by 48 for frequency up to 120MHz. pclk is connected to clk_sys (up to 100MHz). Mdc clock needs to be below 2.5MHz.
`define gem_endian_swap_def 2'b00	Define default value for DMA endianism in DMA configuration register	Define as 2'b00 to indicate little indian for both descriptors and data packets.
`define gem_rx_pbuf_size_def 2'b11	Define default value of RX DMA packet buffer memory sizes in DMA configuration register.	Define as 2'b11 to indicate use full configured memory size.
`define gem_tx_pbuf_size_def 1'b1	Define default value of TX DMA packet buffer memory sizes in DMA configuration register.	Define as 1'b1 to indicate use full configured memory size.
`define gem_rx_buffer_length_def 8'd24	Define the default receive buffer length (integer value). The value determines the default size of the buffer use in system memory when writing received data and can be over-written by writing to the DMA configuration register.	Define as 8'd24 to indicate 1536 bytes.
`define gem_tsu_clk	Define to choose clock tsu from tsu_clk rather than pclk.	Define to choose clock tsu from tsu_clk.
// `define gem_ext_tsu_timer	Define to enable external timestamp port	Comment out.

Table 17: gem_gxl configuration in mxeth

SoftIP_Parameters

4.3 Specification Tables

Numerical specifications are tabulated into operating condition (OP) requirements, transient (AC) specifications, and steady-state (DC) specifications.

4.3.1 MDIO AC spec:

Following figures are snapshot of signal timing characteristics from IEEE Std 802.3, clause 22.3.

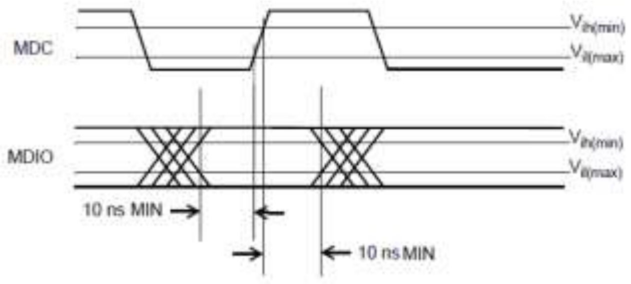


Figure 22–18—MDIO sourced by STA

Figure 4-1: MDIO AC timing waveform for signal sourced by MAC

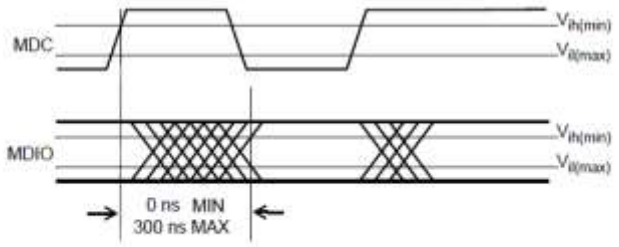


Figure 22–19—MDIO sourced by PHY

Figure 4-2: MDIO AC timing waveform for signal sourced by PHY

4.3.2 MII AC spec:

Following figures are snapshot of signal timing characteristics from IEEE Std 802.3, clause 22.3.

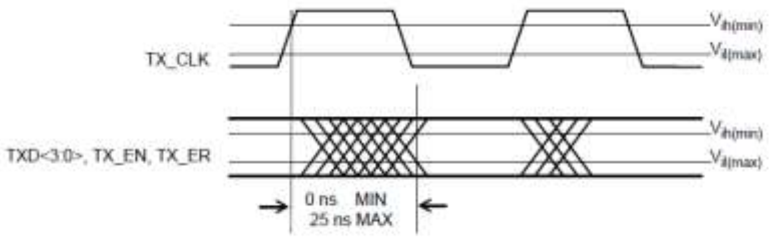


Figure 22–16—Transmit signal timing relationships at the MII

Figure 4-3: Transmit signal timing relationships at MII

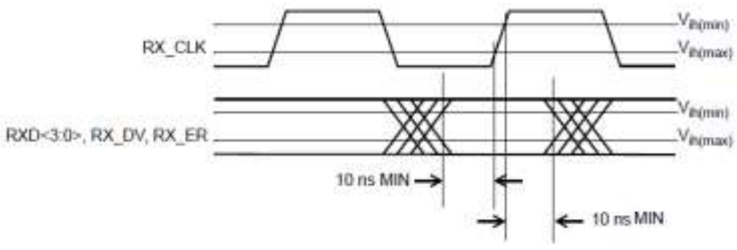


Figure 22–17—Receive signal timing relationships at the MII

Figure 4-4: Receive signal timing relationships at MII

4.3.3 RMII AC spec:

Following AC parameter table is copied from RMII spec.

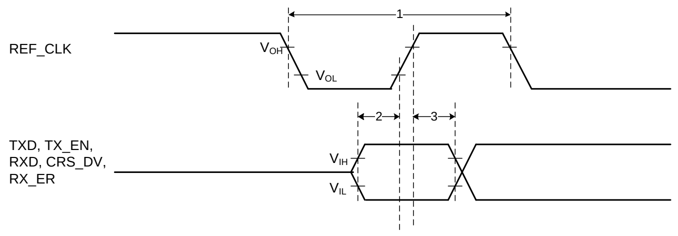


Figure 4-5: AC timing waveform for RMII

Symbol	Parameter	Min	Typ	Max	Units
	REF_CLK Frequency		50		MHz
	REF_CLK Duty Cycle	35		65	%
Tsu	TXD[1:0], TX_EN,RXD[1:0], CRS_DV, RX_ER Data Setup to REF_CLK rising edge	4			ns
Thold	TXD[1:0], TX_EN,RXD[1:0], CRS_DV, RX_ER Data hold from REF_CLK rising edge	2			ns

Table 18: AC parameters at RMII

4.3.4 GMII AC spec:

Following figures are copied from IEEE Std 802.3, clause 35.5.2.

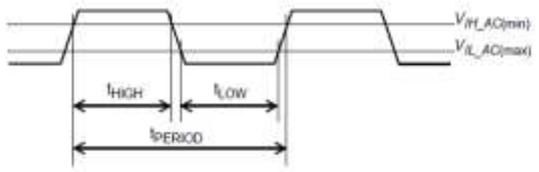


Figure 35–21—GTX_CLK and RX_CLK timing parameters at receiver input

Figure 4-6: GMII GTX and RX_CLK timing at receiver input

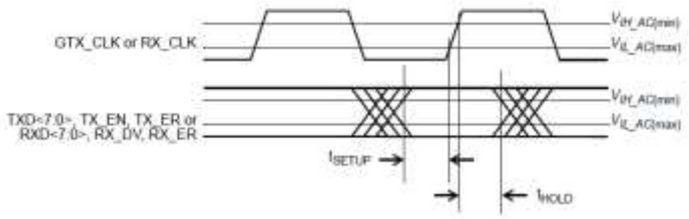


Figure 35–23—GMII signal timing at receiver input

Figure 4-7: GMII signal timing at receiver input

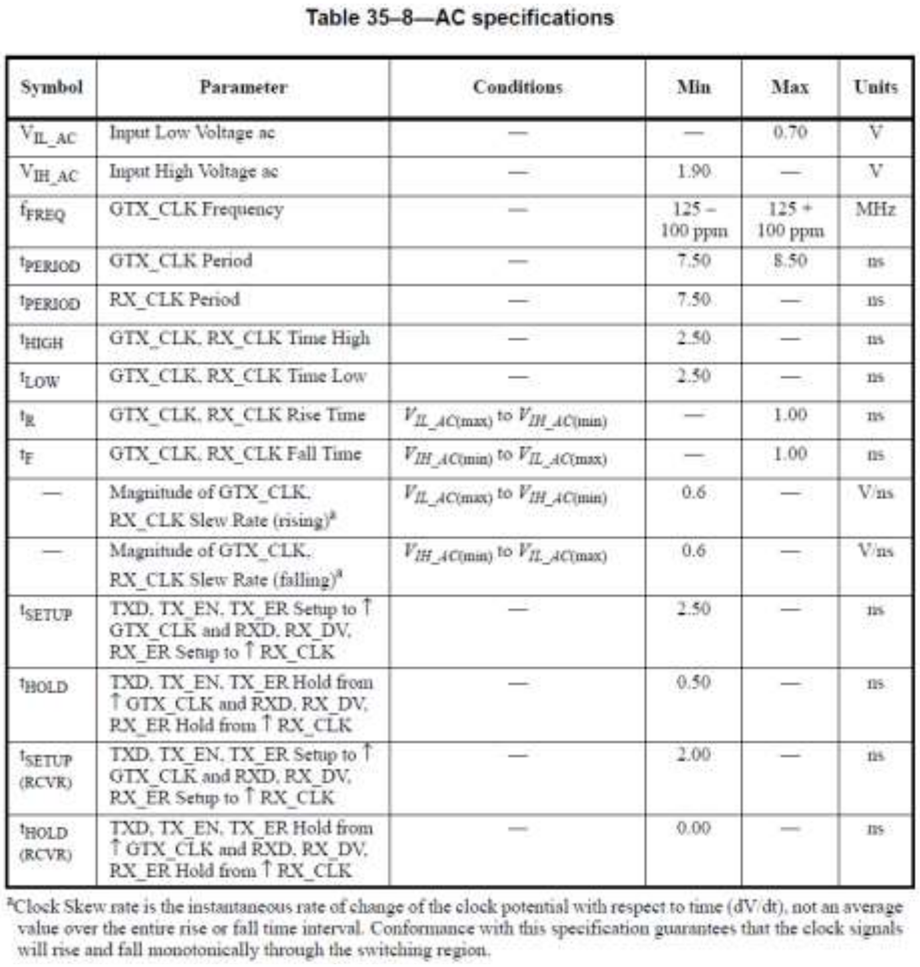


Table 35–8—AC specifications

Symbol	Parameter	Conditions	Min	Max	Units
V_{IL_AC}	Input Low Voltage ac	—	—	0.70	V
V_{IH_AC}	Input High Voltage ac	—	1.90	—	V
f_{FREQ}	GTX_CLK Frequency	—	125 – 100 ppm	125 + 100 ppm	MHz
t_{PERIOD}	GTX_CLK Period	—	7.50	8.50	ns
t_{PERIOD}	RX_CLK Period	—	7.50	—	ns
t_{HIGH}	GTX_CLK, RX_CLK Time High	—	2.50	—	ns
t_{LOW}	GTX_CLK, RX_CLK Time Low	—	2.50	—	ns
t_R	GTX_CLK, RX_CLK Rise Time	$V_{IL\_AC(max)}$ to $V_{IH\_AC(min)}$	—	1.00	ns
t_F	GTX_CLK, RX_CLK Fall Time	$V_{IH\_AC(min)}$ to $V_{IL\_AC(max)}$	—	1.00	ns
—	Magnitude of GTX_CLK, RX_CLK Slew Rate (rising)[a]	$V_{IL\_AC(max)}$ to $V_{IH\_AC(min)}$	0.6	—	V/ns
—	Magnitude of GTX_CLK, RX_CLK Slew Rate (falling)[a]	$V_{IH\_AC(min)}$ to $V_{IL\_AC(max)}$	0.6	—	V/ns
t_{SETUP}	TXD, TX_EN, TX_ER Setup to ↑ GTX_CLK and RXD, RX_DV, RX_ER Setup to ↑ RX_CLK	—	2.50	—	ns
t_{HOLD}	TXD, TX_EN, TX_ER Hold from ↑ GTX_CLK and RXD, RX_DV, RX_ER Hold from ↑ RX_CLK	—	0.50	—	ns
t_{SETUP} (RCVR)	TXD, TX_EN, TX_ER Setup to ↑ GTX_CLK and RXD, RX_DV, RX_ER Setup to ↑ RX_CLK	—	2.00	—	ns
t_{HOLD} (RCVR)	TXD, TX_EN, TX_ER Hold from ↑ GTX_CLK and RXD, RX_DV, RX_ER Hold from ↑ RX_CLK	—	0.00	—	ns

[a]Clock Skew rate is the instantaneous rate of change of the clock potential with respect to time (dV/dt), not an average value over the entire rise or fall time interval. Conformance with this specification guarantees that the clock signals will rise and fall monotonically through the switching region.

Table 19: AC parameters at GMII

4.3.5 OpenRGMII AC spec

Following figures are the AC timing parameter snapshot copied from OpenRGMII spec, section 7.2. mxeth only supports DoD mode (delay on destination).

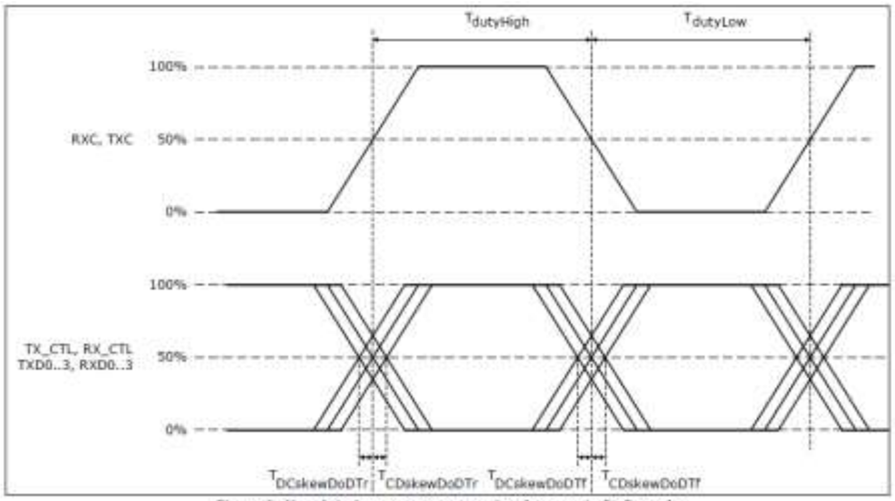
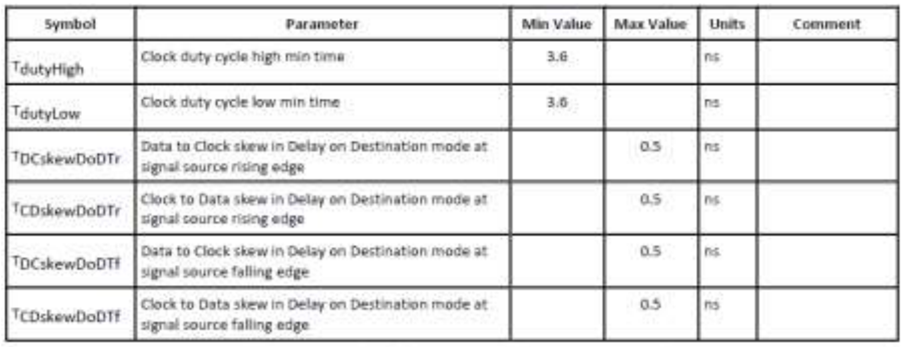


Figure 3: Signal timing parameters at signal source in DoD mode

Figure 4-8: AC timing waveform for signal source in DoD mode



Symbol	Parameter	Min Value	Max Value	Units	Comment
$T_{dutyHigh}$	Clock duty cycle high min time	3.6		ns	
$T_{dutyLow}$	Clock duty cycle low min time	3.6		ns	
$T_{DCskewDoDTr}$	Data to Clock skew in Delay on Destination mode at signal source rising edge		0.5	ns	
$T_{CDskewDoDTr}$	Clock to Data skew in Delay on Destination mode at signal source rising edge		0.5	ns	
$T_{DCskewDoDTf}$	Data to Clock skew in Delay on Destination mode at signal source falling edge		0.5	ns	
$T_{CDskewDoDTf}$	Clock to Data skew in Delay on Destination mode at signal source falling edge		0.5	ns	

Table 2: Duty cycle and skews at signal source in DoD mode

Table 20: AC parameter for signal source in DoD mode

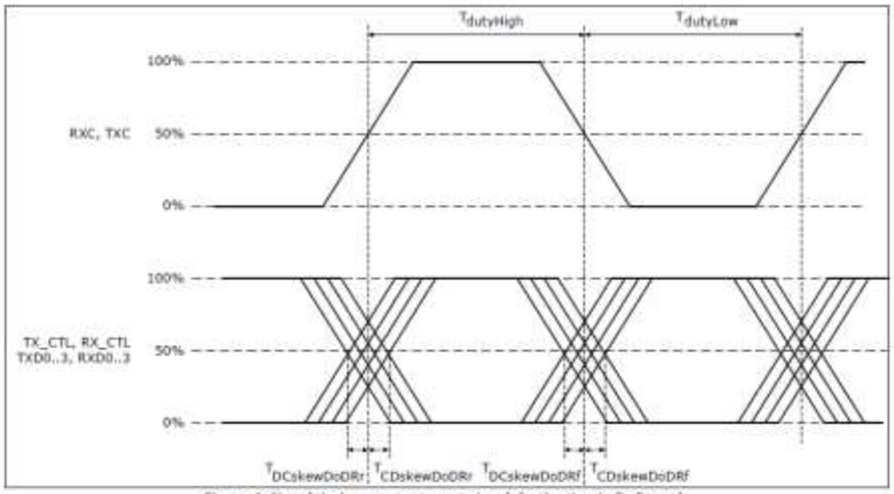
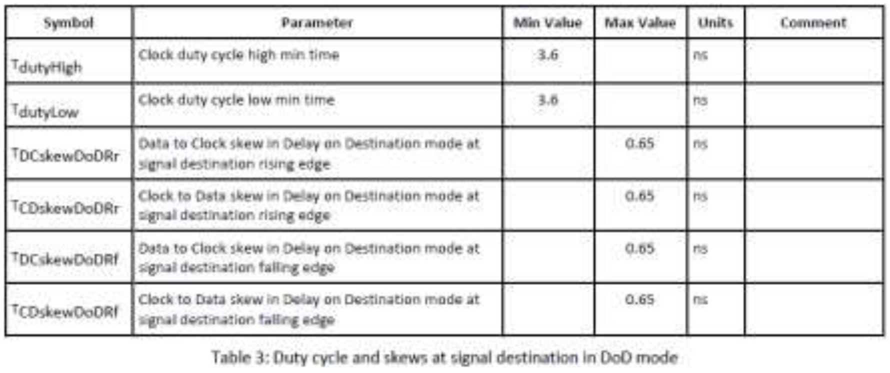


Figure 4: Signal timing parameters at signal destination in DoD mode

Figure 4-9: AC timing waveform for signal destination in DoD mode



Symbol	Parameter	Min Value	Max Value	Units	Comment
$T_{dutyHigh}$	Clock duty cycle high min time	3.6		ns	
$T_{dutyLow}$	Clock duty cycle low min time	3.6		ns	
$T_{DCskewDoDRr}$	Data to Clock skew in Delay on Destination mode at signal destination rising edge		0.65	ns	
$T_{CDskewDoDRr}$	Clock to Data skew in Delay on Destination mode at signal destination rising edge		0.65	ns	
$T_{DCskewDoDRf}$	Data to Clock skew in Delay on Destination mode at signal destination falling edge		0.65	ns	
$T_{CDskewDoDRf}$	Clock to Data skew in Delay on Destination mode at signal destination falling edge		0.65	ns	

Table 3: Duty cycle and skews at signal destination in DoD mode

Table 21: AC parameter for signal destination in DoD mode

SoftIP_Op_Conditions

SoftIP_AC_Specs

SoftIP_DC_Specs

4.4 On-Chip Interfaces

AMBA AHB lite/AHB master.

AMBA AXI4 master interface. The address of read channel and write channel are merged through mxambatk_axi_addr_merge module, before it is connected with CPUSS.

CDT-257222 has a discussion on whether TX/RX SRAM buffer needs to add ECC protection. The conclusion is, there is not necessity to add such circuit since SW has already provide end-to-end protection.

CDT-251541 has a decision to use Single Port SRAM for TX/RX buffer instead of Dual Port SRAM. The system bandwidth requirement to use SPRAM is put in post#9 of the CDT.

4.5 Clocks, Resets, and High-Fanout Nets

This section gives an overview of the clocks, resets, and high-fanout nets used in this block.

4.5.1 Clocks

There are 7 clock input ports and 2 clock output ports in the block.

Clock port	Direction	Description
clk_sys	in	system clock for AHB operation.

Clock port	Direction	Description
clk_tsu	in	TSU clock for TSU timer.
clk_mem	in	Fast clock for AXI operation.
clk_slow	in	Slow clock for the AHB operation.
tx_clk_in	in	MAC transmit clock, used by the MAC transmit block. In 10/100 MII mode, tx_clk_in provides MAC transmit clock at either 2.5 MHz or 25 MHz as determined by the external PHY MII clock input. In GMII and RGMII mode，MAC transmit clock is not supplied by external PHY clock input, hence tx_clk_in is not used. RMII mode uses tx_clk_in if internal reference clock (CTL.REFCLK_SRC_SEL=”1”) is selected, although MAC transmit clock is not supplied by external PHY. See 4.5.1.2.2.
rx_clk_in	in	MAC receive clock, used by the MAC receive block. In MII, GMII and RGMII modes, rx_clk_in provides MAC receive clock at either 2.5 MHz, 25 MHz or 125MHz as determined by the external PHY MII clock input. In RMII mode，MAC receive clock is not supplied by external PHY clock input, hence rx_clk_in is not used.
ref_clk_in	in	Reference clock supplied off chip through HSIO.
ref_clk_int	in	Reference clock supplied on chip from PLL.
tx_clk_out	out	Clock sourced from ref_clk_in or ref_clk_int, provided by MAC to PHY in RMII, GMII and RGMII mode.
mdc_out	out	Clock output for MDC interface. It’s generated from clk_sys.

Table 22: Clock ports of mxeth

There are 8 clock domains being supplied by the clock input ports described in above table, as per different PHY modes. All of these clock domains are treated as asynchronous to each other inside the block.

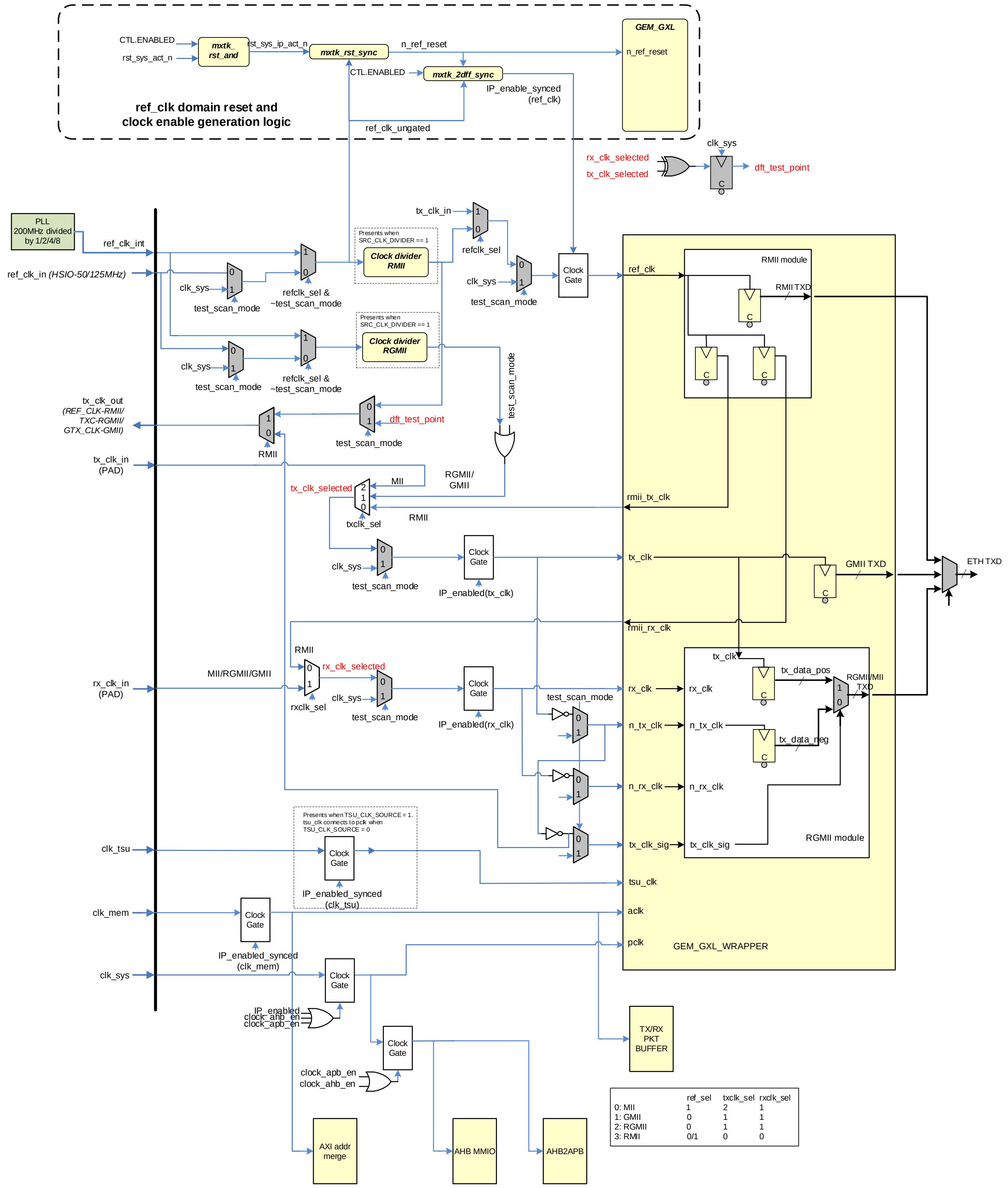
Clock domain	Function	Sync	Clock Frequency	Clock input port	Source
System Clock (clk_sys)	AHB, APB, MDC and TSU	Async	100MHz	clk_sys	PERI
AXI Clock (aclk)	AXI DMA	Async	200MHz	clk_mem	CPUSS
AHB Clock (hclk)	AHB DMA	Async	100MHz	clk_slow	CPUSS

Clock domain	Function	Sync	Clock Frequency	Clock input port	Source
TSU Clock (tsu_clk)	TSU timer	Async	200MHz	clk_tsu	clk_hf
PHY Transmit Clock (tx_clk)	MII	Async	2.5MHz/ 25MHz	tx_clk_in	PHY
	RMII	Sync to ref_clk	2.5MHz/ 25MHz	Generated internally	Divided by DFF from reference clock (REF_CLK: 50MHz +/-50ppm)
	GMII	Async	125MHz +/- 100ppm	ref_clk_in or ref_clk_int	Directly connected to reference clock input which is from HSIO or PLL.
	RGMII	Async	125MHz/ 25MHz/ 2.5MHz +/- 50ppm	ref_clk_in or ref_clk_int	Directly connected to reference clock input which is from HSIO or PLL.
Inverted Transmit Clock (n_tx_clk)		Inverted version of transmit clock. It is used for RGMII mode.			
PHY Receive Clock (rx_clk)	MII	Async	2.5MHz/ 25MHz	rx_clk_in	PHY
	RMII	Sync to ref_clk	2.5MHz/ 25MHz	internally generated	Divided by DFF from reference clock
	GMII	Async	125MHz	rx_clk_in	PHY
	RGMII	Async	125MHz/ 25MHz/ 2.5MHz	rx_clk_in	PHY
Inverted Receive Clock (n_rx_clk)		Inverted version of receive clock. It is used for RGMII mode.			
Reference Clock (ref_clk)	MII/ GMII/ RGMII	Only presents when RMII mode is selected.			

Clock domain	Function	Sync	Clock Frequency	Clock input port	Source
	RMII	Async	50MHz + 50ppm	ref_clk_in	Off chip clock source (HSIO)
				ref_clk_int	On chip clock source (PLL)

Table 23: Clock domains and source

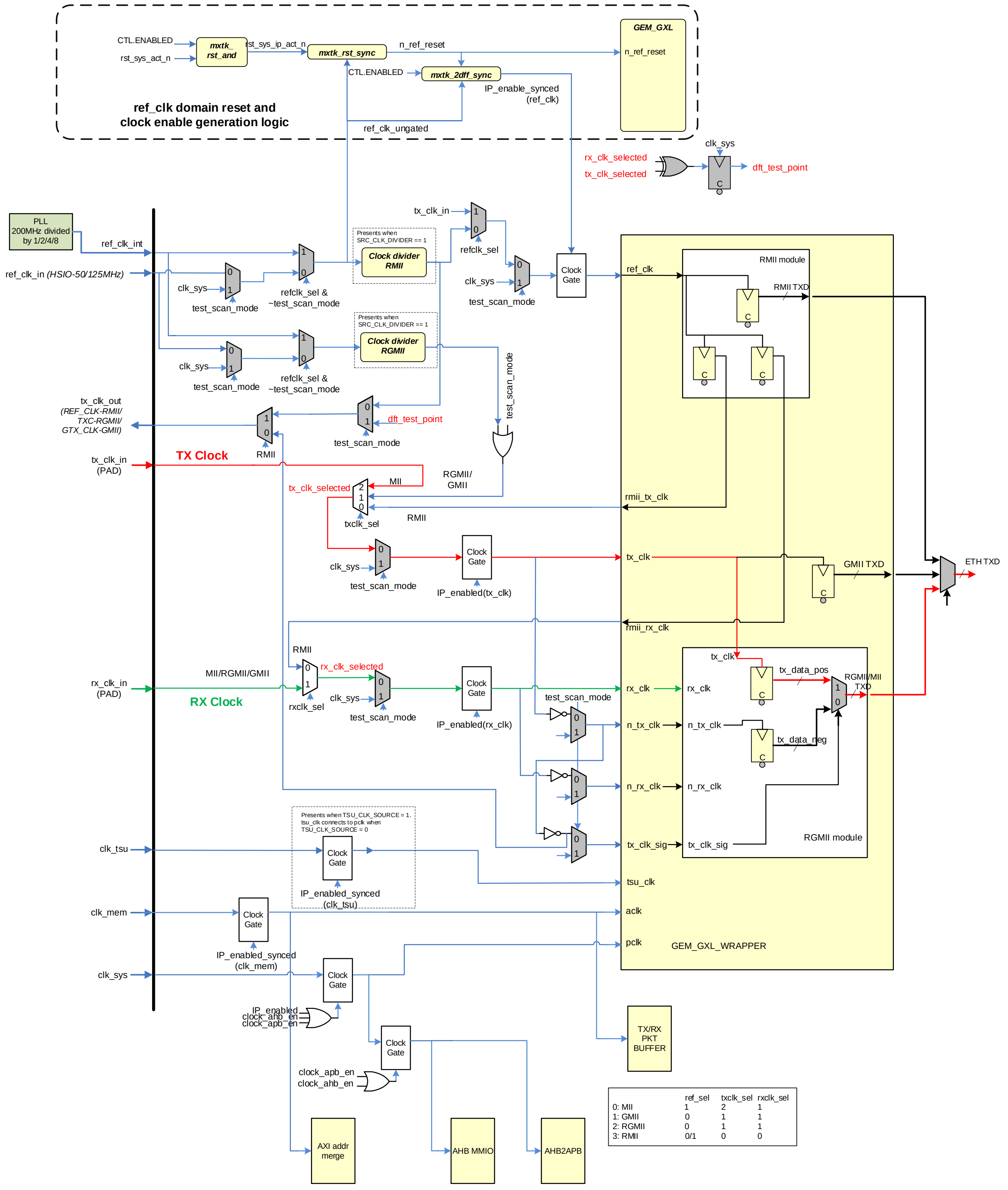
Clock structure:



	ref_sel	txclk_sel	rxclk_sel
0: MII	1	2	1
1: GMII	0	1	1
2: RGMII	0	1	1
3: RMII	0/1	0	0

Figure 4-10: Clock structure of mxeth

4.5.1.1 MII

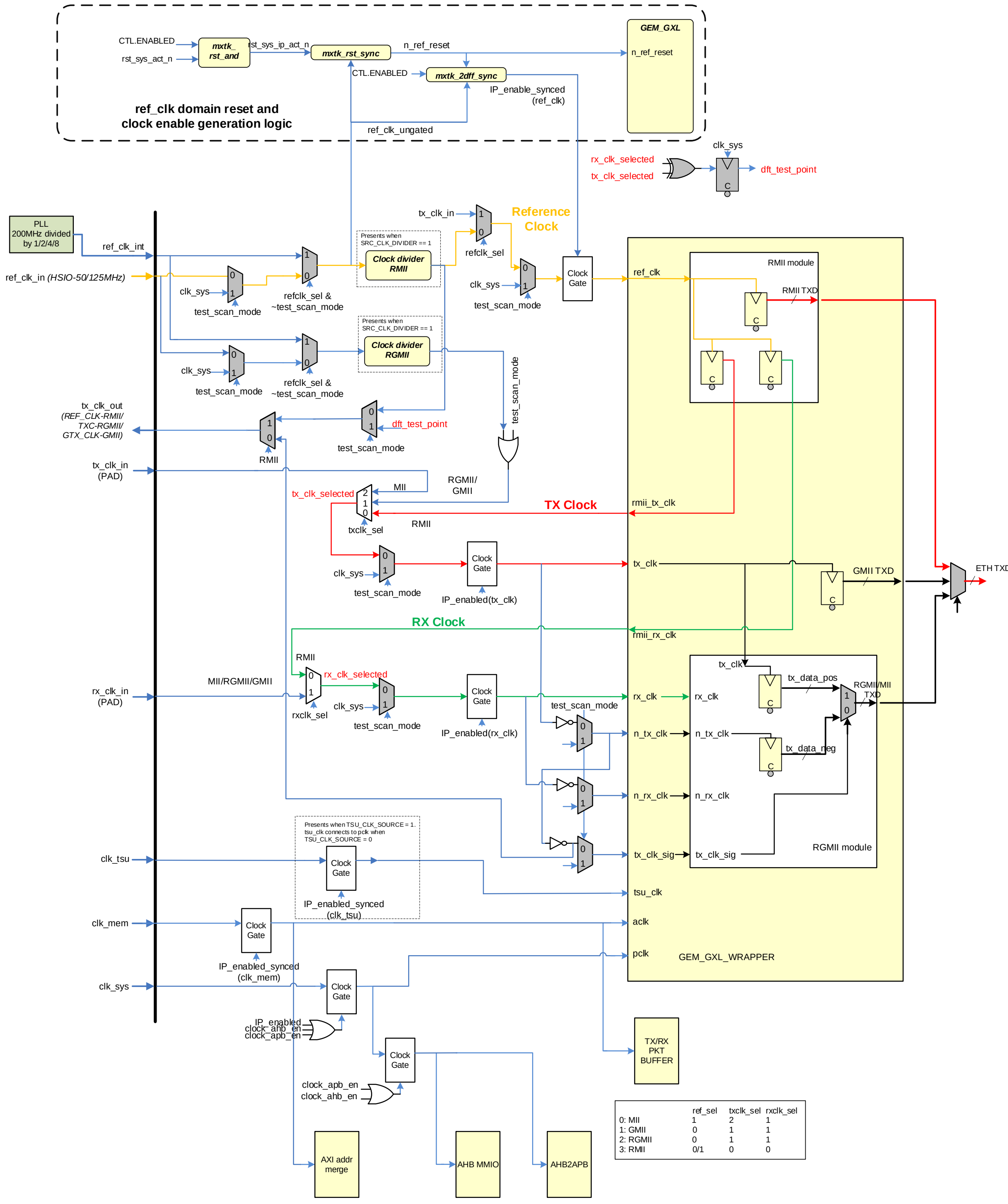


	ref_sel	txclk_sel	rxclk_sel
0: MII	1	2	1
1: GMII	0	1	1
2: RGMII	0	1	1
3: RMII	0/1	0	0

Figure 4-11 MII Mode Clock Path

4.5.1.2 RMII

4.5.1.2.1 External Reference Clock Used (CTL.REFCLK_SRC_SEL="0")



	ref_sel	txclk_sel	rxclk_sel
0: MII	1	2	1
1: GMII	0	1	1
2: RGMII	0	1	1
3: RMII	0/1	0	0

Figure 4-12 RMII Mode Clock Path (External Reference Clock Used)

4.5.1.2.2 Internal Reference Clock Used (CTL.REFCLK_SRC_SEL=”1”)

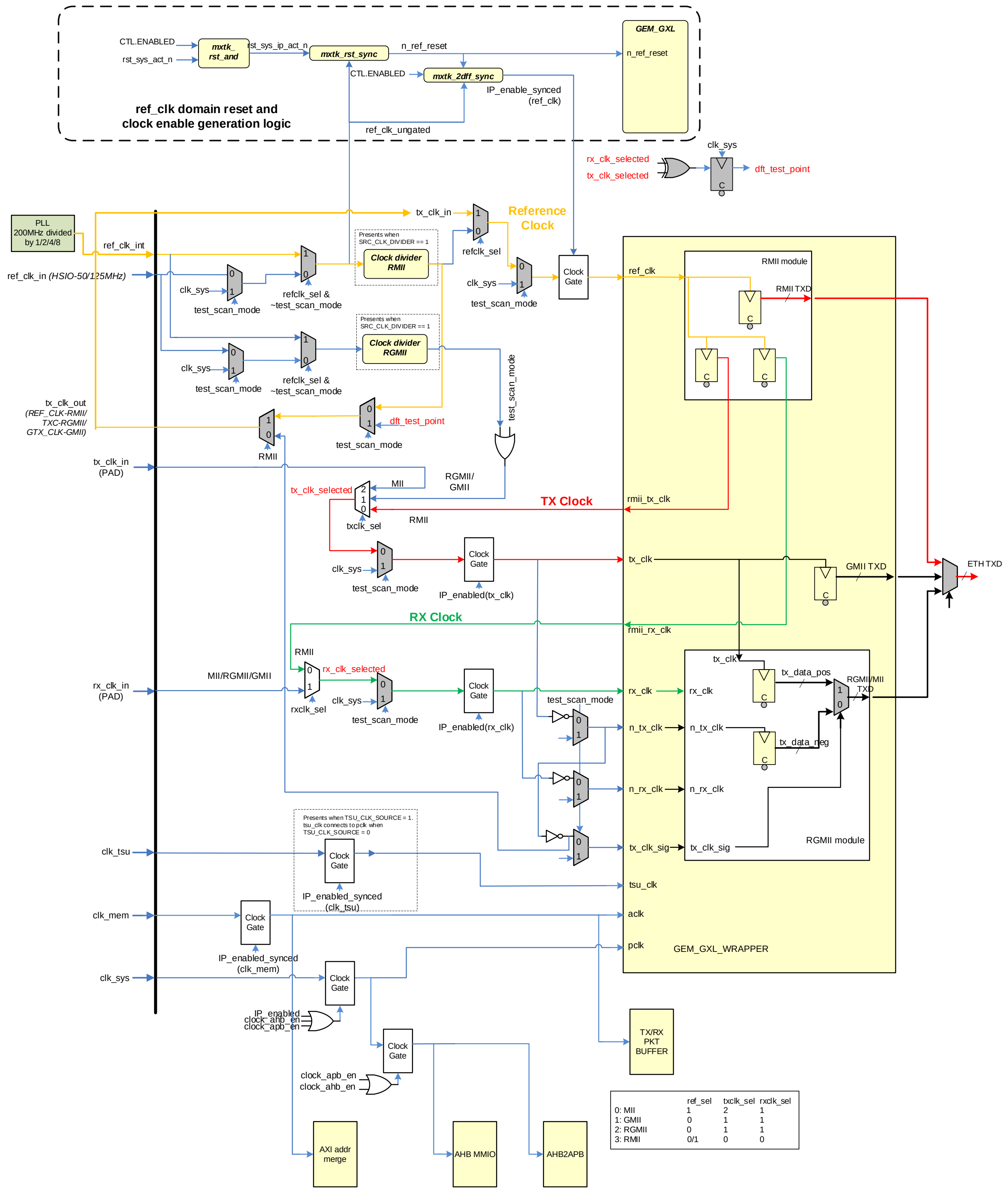


Figure 4-13 RMII Mode Clock Path (Internal Reference Clock Used)

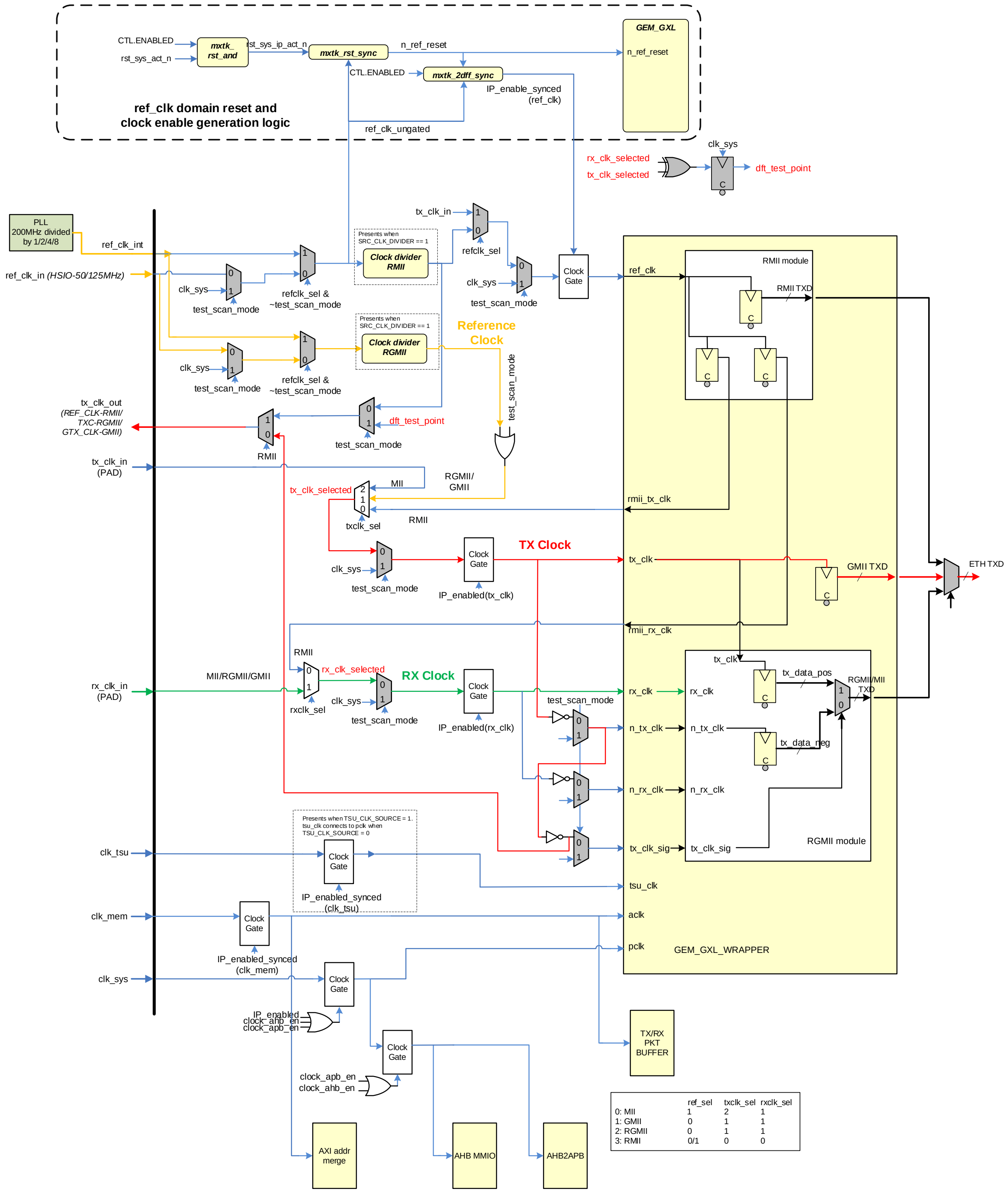
In this mode, the reference clock is routed as follows:

1. Internal reference clock is divided by divider in mxeth

2. Goes out from tx_clk_out to I/O (clock for PHY)
3. Loops back from I/O to tx_clk_in
4. Enters GEM_GXL to generate TX/RX clocks
5. TX/RX data are launched/captured by these clocks

This is done to align the phase of the reference clock as much as possible to that of RX data which is returned from PHY.

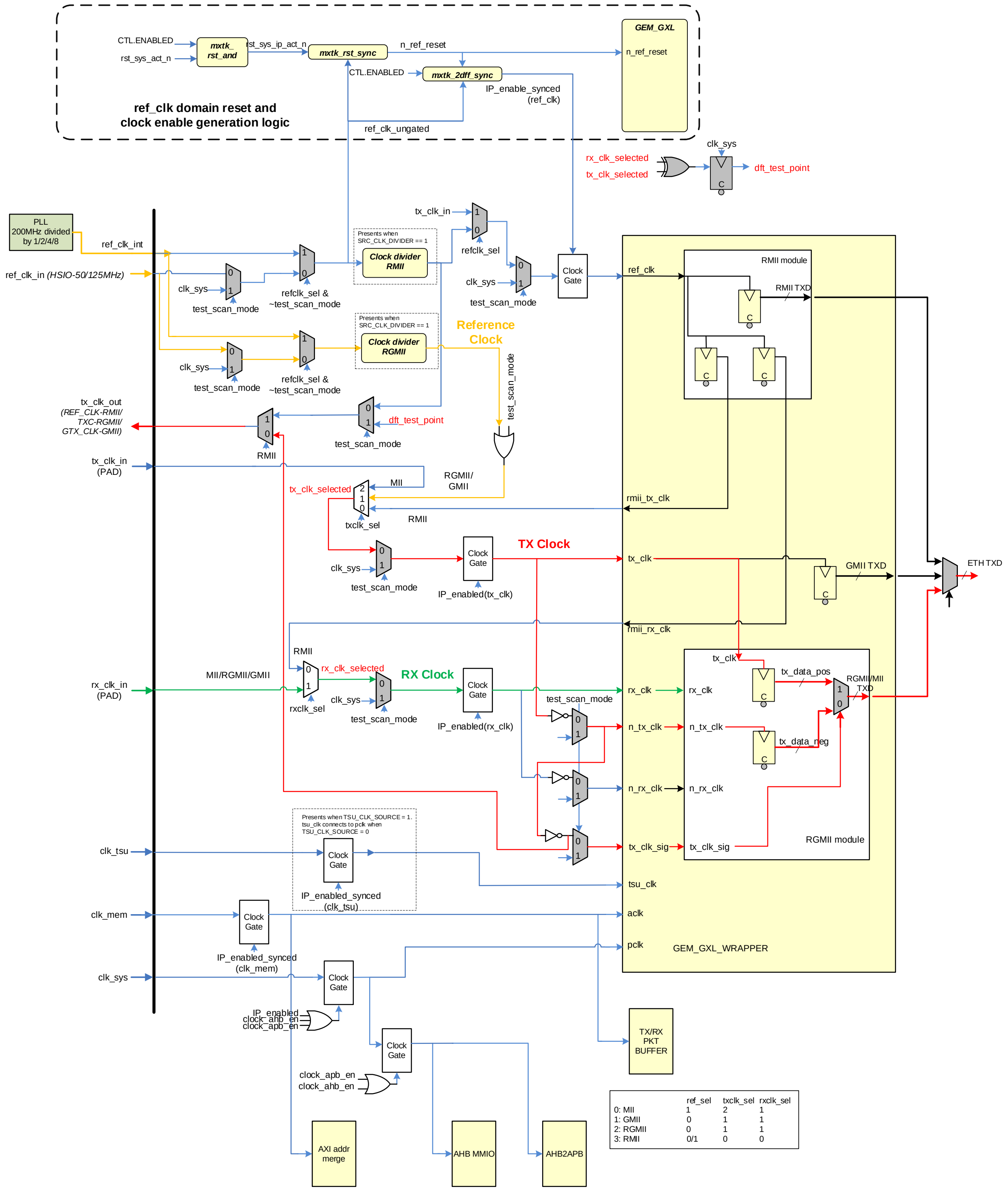
4.5.1.3 GMII



	ref_sel	txclk_sel	rxclk_sel
0: MII	1	2	1
1: GMII	0	1	1
2: RGMII	0	1	1
3: RMII	0/1	0	0

Figure 4-14 GMII Mode Clock Path

4.5.1.4 RGMII



	ref_sel	txclk_sel	rxclk_sel
0: MII	1	2	1
1: GMII	0	1	1
2: RGMII	0	1	1
3: RMII	0/1	0	0

Figure 4-15 RGMII Mode Clock Path

4.5.2 Resets

As active domain peripheral, this IP is following SAS IP-Rules Chapter section 2.1.5 on reset hookup and control. This IP has 3 reset source input from SRSS or PERI.

1) rst_sys_act_n – Low active, asynchronous Active reset signals associated with clock clk_sys. It will reset entire gem_gxl module after being synchronized to separate clock domain and glue logic in wrapper module, except for the retention mmio register outside gem_gxl.

2) rst_sys_dpslp_n – Low active, asynchronous DeepSleep reset signals associated with clock clk_sys. It will reset retention mmio registers outside gem_gxl.

3) rst_mem_act_n – Low active, asynchronous Active and DeepSleep reset signals associated with clock clk_mem. It will reset AXI clock domain logic in AXI addr merge block (AXI domain logic inside gem_gxl is reset by reset signal synchronized from rst_sys_act_n).

4) rst_slow_act_n – Low active, asynchronous Active and DeepSleep reset signals associated with clock clk_slow. AHB domain logic inside gem_gxl is reset by reset signal synchronized from rst_sys_act_n.

When IP is disabled, entire gem_gxl including the MMIO registers inside gem_gxl block will be reset. MMIO is still accessible, access to 3PIP is quietly ignored: Writes dropped and Reads return 0x0. The MMIO register outside gem_gxl is retention register and will keep value while IP is disabled.

Note, GEM_GXL 3PIP MMIO registers has only one reset input through APB slave interface, this input is hookded up with rst_sys_ip_act_n, which makes all MMIO registers inside 3PIP non-retainable in order to save power in DeepSleep mode. (This is not compliant with the requirement in SAS IP rules chapter: MMIO configuration register should be retained to save wakeup time without SW reconfiguration, tracked in CDT-257212).

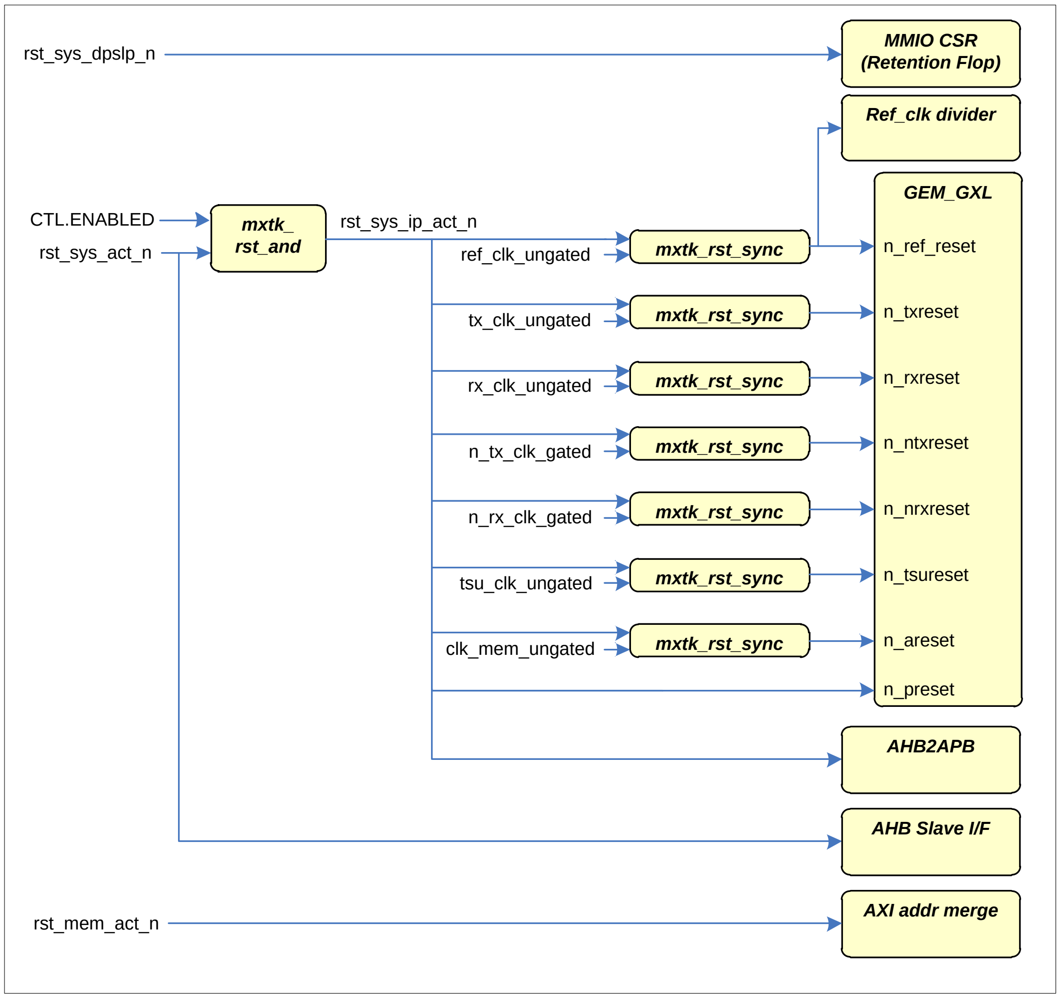


Figure 4-16: Reset structure of mxeth

4.6 Supply Rails and Power Domains

This section gives an overview of the supply rails and power domains used in this block. Power modes are described in Section 3.4.

4.6.1 Power modes

The Block locates in active power domain and works in ACTIVE, LPACTIVE, SLEEP and LPSLEEP modes. No power mode changing existing in the block. The retention registers outside gem_gxl will be still powered in DEEPSLEEP mode to keep state for MMIO register which need to retain the value. The SRAM in the block doesn’t retain value in DEEPSLEEP mode.

There is no power switch integrated in the block.

Power Mode	Block Status
ACTIVE	This block is fully operational in this mode with clock running with power on.
LPACTIVE	LPACTIVE (LOWPOWERACTIVE) is similar as ACTIVE, with some tradeoffs applied to reduce current.
SLEEP	CPU is sleep in this mode but the function is still on. All counter clocks are ON.
LPSLEEP	LPSLEEP (LOWPOWERSLEEP) relates to LPACTIVE in the same way that SLEEP relates to ACTIVE.
DEEPSLEEP	In this mode, the vccact power to this block is off and no clock is provided, hence the logic is not functional. All retention registers will keep their state with power vccret.
HIBERNATE	In this mode, the entire block is powered off and no clock is provided.

Table 24: Block status in power modes

4.6.2 Power rails

Vccact will be cut off in HIBERNATE and DEEPSLEEP mode, but supplied in ACTIVE/LPACTIVE/SLEEP/LPSLEEP mode.Vccdpslp is supplied in DEEPSLEEP mode for retention registers and off in HIBERNATE mode.

4.6.3 Clock/Reset integration

Clock/Reset integration for active domain block is defined in SAS as below figure.

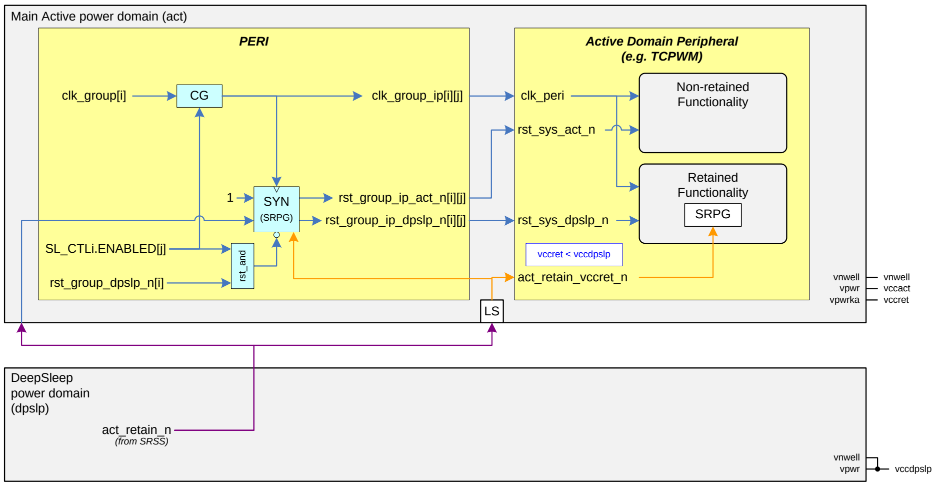


Figure 4-17: clock reset integration (Quoted from SAS)

rst_sys_dpslp_n will be "1" in deepsleep mode, while rst_sys_act_n is "x" due to power off.

4.6.4 SRAM integration

The power integration for both TX and RX SRAMs follows “Switched peripheral SRAM without retention” documented in the “IP rules and guidelines” section of the SAS.

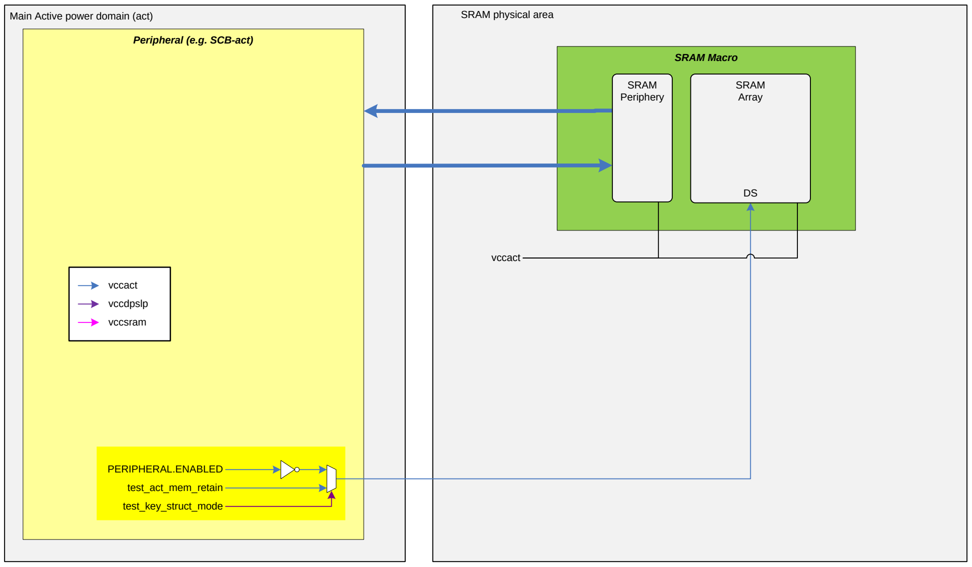


Figure 4-18: SRAM integration (Quoted from SAS)

Following is quoted from SAS:

“Note that the idea is to save some power when the IP is disabled during Active mode, without the cost of a separate power switch. Effectively power is only really saved if the power switch is part of the SRAM (e.g. S40S Synopsys). When that is not the case then the power save will be negligible.

In DeepSleep power to the SRAM is off and therefore will not Retain. In Active mode when disabled the SRAM will Retain its’ contents, but that is just a side effect of not adding an external power switch.”

4.7 Design For Test (DFT)

This section gives an overview of the DFT approach and explains where to find information about test modes, methods, and coverage results.

This block will use ATPG full scan for DFT as it’s a pure synchronized digital design. The fault coverage entitlement for this IP is 99.8%.

At IPS3, the DFT coverage is 99.81%. It’s obtained through ATPG scan and manual exclusion of the faults inside s40esram macro which has been claimed covered by s40esram. Most coverage loss is redundant fault which are also shown in TVI product (YOHI-15).

4.8 IP Block Metrics

The following table contains metrics that may be used in chip-level budgets.

SoftIP_DesignMetrics

4.9 Integration Guidelines

4.9.1 Tools and Views

This subsection describes the tools and views provided to support those tools.

IP deliveries are defined in section 8.1 - IP Deliverables Worksheet.

IP Tool set are defined in PCIOS 001-97729.

4.9.2 Verification Simulation

Full chip integration verification needs to cover all block interface function well to ensure all interface are properly hookup in chip level. Power aware simulation need to be run for mode transition (ACTIVE, DEEPSLEEP and HIBERNATE) to verify reset, power control signal and UPF integration.

The PHY interface programmable wrapper and mentor 1G ETH QVIP are required to be integrated in chip level to run integration test with real PHY transaction.

4.9.3 FPGA Guidelines

PSVP/SVS validation strategy plan is defined in RMKN-66 based on IP feasibility memo WJIN-425. Additionally, “FPGA” define switch bypasses the SCAN MUXes and clock gating cells for FPGA implementation.

4.9.4 Silicon Place & Route Guidelines

4.9.5 Static Timing

- Static timing modes required by this IP
  - FUNC_MAX_HFCLK_FREQ (Synthesis Super-set mode)
  - FUNC_ETH_GMII
  - FUNC_ETH_RGMII

  - FUNC_ETH_MII
  - FUNC_ETH_RMII
- IP-specific timing constraint file
  - mxeth/syn/mxeth_top/cst/mxeth_top_clocks_hobto.tcl
  - mxeth/syn/mxeth_top/cst/mxeth_top_io_case_analysis_hobto.tcl
  - mxeth/syn/mxeth_top/cst/mxeth_top_io_delay_hobto.tcl
  - mxeth/syn/mxeth_top/cst/mxeth_top_timing_exceptions.tcl
- Timing requirement for RGMII DDR output

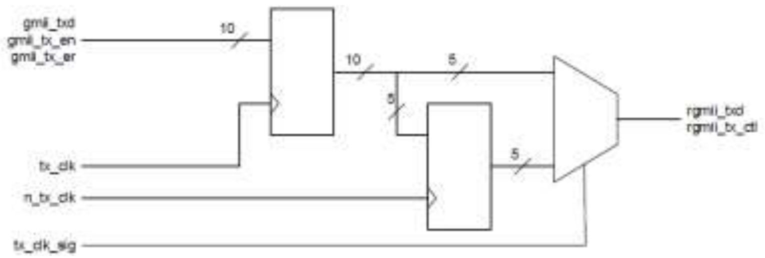


Figure 4-19: rgmii tx output data path

As shown in above figure, tx_clk_sig must arrive the selection port of rgmii_txd MUX when the inputs of MUX are stable.

- Divided clock for RMII mode

In FUNC_ETH_RMII mode, both TX_CLK and RX_CLK is generated from flipflop inside GEM_GXL.

2 clocks need to be created on Q port of DFF:

u_gem_gxl_wrapper/u_gem_gxl/i_rmii/tx_clk_reg/Q

u_gem_gxl_wrapper/u_gem_gxl/i_rmii/rx_clk_reg/Q

5. SOFT IP: INTERNAL DESIGN

5.1 MUX of MII/RMII/GMII/RGMII input and output path.

rgmii interface module is implemented outside GEM_GXL to afford capability to support both GMII and RGMII interface through MMIO register programming.

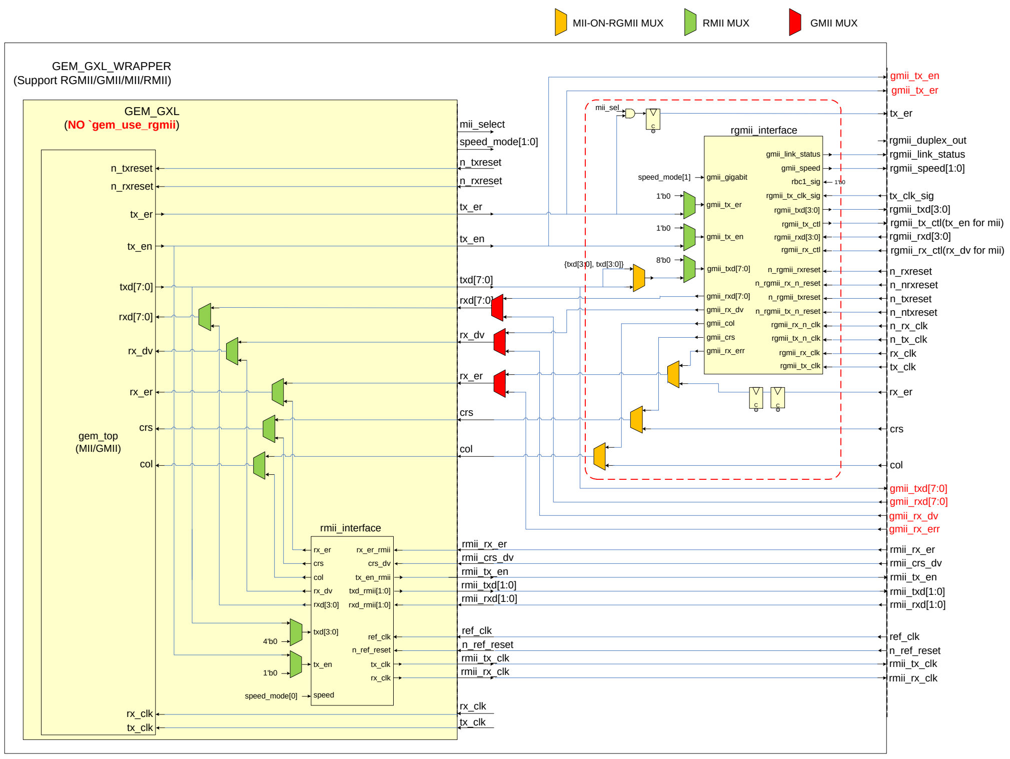


Figure 5-1: gem_gxl_wrapper for RGMII interface

Glue logic is used for selection of 4 different PHY interface clock path and data path selection, which is controlled by MMIO register CTL.ETH_MODE.

6. REFERENCE DOCUMENTS

6.1 Document Control Specifications

00-00064 – Cypress Record Retention Policy

6.2 Funding Specifications

Type	Specification	Title
NPP	002-23140	TVII-B-H NEW PRODUCT PLAN

6.3 Feature / Requirements Definition Specifications

Type	Specification	Title
SAS	001-98134	MXS40 SYSTEM ARCHITECTURE SPECIFICATION (SAS)
EROS	002-21850	TVII-B-H CHIP EROS
PCIOS	001-97729	MXS40 PLATFORM CHIP INTEGRATION OBJECTIVE SPEC (PCIOS)

6.4 External/Industry Standards

Specification	Title
AHB	AMBA 3 AHB Lite Protocol Specification Version 1.0
AXI3	AMBA® AXI™ and ACE™ Protocol Specification
IEEE Std 802.3	IEEE Standard for Ethernet (GMII – clause 35, MII – clause 22, MDIO – clause 22/45)
RMII	RMII™ Specification
RGMII	Reduced Pin-count Interface For Gigabit Ethernet Physical Layer Devices
RFC768 (UDP)	User Datagram Protocol
RFC791 (IPv4)	INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION
RFC793 (TCP)	TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION
RFC894 (Ethernet encoding)	A Standard for the Transmission of IP Datagrams over Ethernet Networks

RFC1042 (SNAP encoding)	A Standard for the Transmission of IP Datagrams over IEEE 802 Networks
IEEE Std 1588™	IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems
IEEE Std 802.1AS™ (PTP)	IEEE Standards for Local and Metropolitan Area Networks: Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks.
IEEE Std 802.1Qav™ (CBS)	Forwarding and Queuing Enhancements for Time-Sensitive Streams
IEEE Std 802.1BA™ (AVB)	IEEE Standard for Local and metropolitan area networks—Audio Video Bridging (AVB) Systems
IEEE Std 802.1Q™ (VLAN)	IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks.
IEEE Std 802.1Qbb™ (PFC)	IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks—Amendment: Priority-based Flow Control
IEEE Std 802.1Qaz™ (ETS)	IEEE Standards for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks – Amendment 18: Enhanced Transmission Selection for Bandwidth Sharing Between Traffic Classes
IEEE Std 802.3az™	IEEE Standard for Local and Metropolitan Area Networks – Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications – Amendment 5: Media Access Control Parameters, Physical Layers, and Management Parameters for Energy-Efficient Ethernet (IEEE Std 802.3: IEEE Standard for Ethernet Clause 78)

All specs can be found in SVN location under mxeth project.

There is no corporate requirement spec for Ethernet interface as documented in WJIN-430.

The block AXI interface is subset of AXI3 with additional few AXI4 features.

SoftIP_IndustryStandards

6.5 Dependent IP Specifications

No IP depends on mxeth.

6.6 Pre-requisite IP Specifications

Type	Specification	Title
SILICON BROS	002-10199	mxambatk SoftIP Block Requirements Objective Spec (BROS)
	001-97499	MXTK SOFTIP BLOCK REQUIREMENTS OBJECTIVE SPEC (BROS)
	002-17622	S40ESRAM HARDIP BLOCK REQUIREMENTS OBJECTIVE SPEC (BROS)

6.7 Other relevant specs for this IP <add to/remove as needed>

6.8 Required Memos

Type	Memo	Title
COMPETITIVE ANALYSIS	YOHI-1	IP Consolidation study: Ethernet MAC IP comparison
ENTITLEMENT STUDY	WJIN-425	Ethernet MAC 3PIP: GEM_GXL Feasibility Analysis for TVII-B-E family
FMEA		FMEA, via PM System-Current View

6.9 Third Party IP References (3PIP), Planning

Document	Title
01-01014	THIRD PARTY IP APPROVED VENDOR LIST
GEM_GXL User Guide Manual	Gigabit Ethernet MAC with DMA, 1588, AVB and PCS (GEM_GXL)

6.10 Related Memos, Other References

Memo/Spec	Title
002-16632	MXETH SOFT IP VERIFICATION REQUIREMENTS
BING-19	VMXETH: GEM_GXL IP REGRESSION STATUS AND

Memo/Spec	Title
	CODE COVERAGE
GAUS-060	MXETH_VER2 DESIGN CHANGES
LIJT-6	COMPARISON BETWEEN VERSION 1P06F3 AND 1P09 OF CADENCE ETHERNET MAC GEM_GXL
MXKU-86	HYPOTHESIS ON CAN-ETHERNET GATEWAY PROCESSOR BUSINESS
MXKU-90	AUTOMOTIVE ETHERNET PHY AND SWITCH AVAILABLE IN MARKET
RMKN-66	TVII-B-H-8M SYSTEM VALIDATION PLAN
WJIN-420	3PIP GIGABIT ETHERNET MAC (CADENCE) INTEGRATION – CHDC QUOTE
WJIN-425	ETHERNET MAC 3PIP: GEM_GXL FEASIBILITY ANALYSIS FOR TVII-B-E FAMILY
WJIN-430	MXETH IPS0 SRB REVIEW STRATEGY – NO CORPORATE REQUIREMENT SPEC
YING-280	MXETH IPS0 VERB
YING-338	MXETH: CADENCE SUPPORT TICKETS SUMMARY
YOHA-5	ETHERNETMAC TEST RESULT OF PHASE.2 EVALUATION
YOHI-15	GEM_GXL DDFT IPS3 MILESTONE REVIEW
YUJI-37	MXETH: DOCUMENTATION FOR SIMULATION RESULTS
YUJI-48	MXETH: PERFORMANCE EVALUATION WITH DIFFERENT DESIGN CONFIGURATIONS
VJYM-89	TVII-C-2D-6M SYSTEM VALIDATION - ES100 CLOSURE
AVEN-59	CYTVII-C-2D-6M-327-B0 SI: SYSTEM VALIDATION ES100 CLOSURE
VVG-499	TVII-C-2D-6M: B0/B1 PCHAR CLOSURE
VVG-502	TVII-C-2D-6M B1: MXETH2.1 CHARACTERIZATION DATA

7. RECORDS

All records shall be maintained as specified in the Cypress Record Retention Policy spec #00-00064.

8. POSTING SHEETS/FORMS/APPENDIX:

8.1 APPENDIX 1 – IP Deliverables Worksheet

This must be submitted with customers and end users (e.g.; CIC group) in the signature loop.

8.2 APPENDIX 2 – Supporting Documents (If Applicable)

8.3 APPENDIX 3 – TABLE OF LINKS

APPENDIX 1 – IP Deliverables Worksheet

DDC Name	mxeth	Technology	
Desc	Ethernet MAC for TVII-B-H-8M	DDC Type	soft
Blocks	mxeth_top	Target Product	TVII-B-H-8M
IP Owner	ying,gls,wjin	CIC Lead	kxsh,hirt,gls
Customer Rep	mtok,gls	Approved By	hirt kxsh gls mtok
Rejected By		Approval waiting from	
State of IP Form	Approved		
Date Submited	6/26/2018 17:31	Date Modified	6/26/2018 17:31
Block	mxeth_top		
Block Type	HOBTO_Peripheral_Soft_IP		
Functional_HDL	Y		
Logic_Sims	Y		
Assertions			
Constraints	Y		
Synthesis_Scripts	Y		
DFT_Scripts	Y		
STA_Scripts			
MAP_File			
RTL_lint_log	Y		
LEC_logs	Y		
CDC_logs	Y		
UPF_file	Y		
Test_Models			
FPGA_Targeted_Model	Y		
MVRC_Logs_Waivers	Y		
Production_Test_Vectors	Y		
Char_Test_Vectors	Y		
HOBTO_Int_Verif	Y		
Power_Analysis	Y		
Verification_IP	Y		

APPENDIX 2 – Supporting Documents (If Applicable)

SoftIP_DetailedData

SoftIP_VTList

SoftIP_CRD

APPENDIX 3 – TABLE OF LINKS

(RETURN TO SUMMARY PAGE)

PM IP Matrix	IP Dashboard	IP Vault
SoftIP_BlockPinList	SoftIP_Parameters	SoftIP_PublicCells
SoftIP_IndustryStandards	SoftIP_Op_Conditions	SoftIP_AC_Specs
SoftIP_DC_Specs	SoftIP_DesignMetrics	SoftIP_CRD
SoftIP_VTList	SoftIP_DetailedData	

Rev.	ECN No.	Orig. of Change	Description of Change
**	6599492	GAUS	IPS0. This IP is ported from mxeth. The change is to include a parameter (AXI_MASTER_PRESENT) to change from AXI and AHB Master interface. Another pin clk_slow is added as AHB master runs on clk_slow. Additional change in mxeth.2 (along with the changes of mxeth_ver2 described before) is a reference to instantiation of mxsramwrap IP to make the soft IP technology independent.
*A	6656038	GAUS	IPS3. DFT coverage of 99.8% received. Backend tool (Synthesis, STA, LINT, LEC, MVRC) with Verification completed for IPS3. BROS completed with the required gate count and DDFT coverage.
*B	6965266	JIRD SLLO	Summary: Corrected description about supported platform (s40e only) 3.4.1.6: Added required frequencies for 32-bits bus width 4.5.1: Changed clock structure diagram to latest design 4.5.1.x: Added clock path diagram for each mode 3.4.1.6: Added required frequencies for AHB 32-bits bus width (CDT-361378) Added “AN INFINEON TECHNOLOGIES COMPANY” on header.
*C	7013652	JIRD	SUMMARY: Updated gate counts SUMMARY: Updated ATPG coverage number 3.4.3.1: Added new section to describe MMIO register access restriction 4.5.1: Fixed RGMII tx_clk_out path in the clock structures
*D	7042727	JIRD	SUMMARY: Updated gate counts 3.4.3.1: Updated the number of clock cycles about MMIO register access restriction
*E	7083445	JIRD	1.3: Added IPS3 tag
*F	7723729	JIRD	SUMMARY: Updated IPS state to IPS4 6.10: Added related MEMOs of Si validations and characterization