Ptp10GExtension

TSU Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	0.2
Date	25.10.2024

Copyright Notice

Disclaimer

The information available to you in this document/code may contain errors and is subject to periods of interruption. While NetTimeLogic GmbH does its best to maintain the information it offers in the document/code, it cannot be held responsible for any errors, defects, lost profits, or other consequential damages arising from the use of this document/code.

NETTIMELOGIC GMBH PROVIDES THE INFORMATION, SERVICES AND PRODUCTS AVAILABLE IN THIS DOCUMENT/CODE "AS IS," WITH NO WARRANTIES WHATSOEVER. ALL EXPRESS WARRANTIES AND ALL IMPLIED WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY RIGHTS ARE HEREBY DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO EVENT SHALL NETTIMELOGIC GMBH BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL AND EXEMPLARY DAMAGES, OR ANY DAMAGES WHATSOEVER, ARISING FROM THE USE OR PERFORMANCE OF THIS DOCUMENT/CODE OR FROM ANY INFORMATION, SERVICES OR PRODUCTS PROVIDED THROUGH THIS DOCUMENT/CODE, EVEN IF NETTIMELOGIC GMBH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IF YOU ARE DISSATISFIED WITH THIS DOCUMENT/CODE, OR ANY PORTION THEREOF, YOUR EXCLUSIVE REMEDY SHALL BE TO CEASE USING THE DOCUMENT/CODE.

Overview

NetTimeLogic's PTP Timestamp Unit allows to create a PTP Ordinary Clock or Boundary Clock with OneStep support according to IEEE1588-2019/2008 (PTP). It takes timestamp for PTP frames and allows additional OneStep timestamp insertion if intercepting the XGMII. This Document describes the 10 Gigabit Extension to the PTP Timestamp Unit

Key Features:

- PTP Timestamp Unit according to IEEE1588-2019/2008 and IEEE802.1AS
- PTP frame detection and parsing
- Optional Passthrough Mode
- Optional PTP One Step functionality
- PTP event frame timestamping
- Optional Meta Information to safely match timestamps and frames
- Optional timestamp buffers for each frame type to handle also bursts of frames and high PTP frame rates (requires Meta Information)
- Taps path between MAC and PHY
- Synchronization accuracy: +/- 25ns
- Support for Layer 2 (Ethernet) and Layer 3 (Ipv4 and IPv6), Peer to Peer (P2P) and End to End (E2E).
- Support for Unicast Frames
- Master and Slave support
- Full line speed
- AXI4Lite register set
- Configurable Interrupt
- PHY Delay compensation with automatic link speed detection (in driver)
- XGMII Interface support
- Timestamp resolution with 50 MHz system clock: 10ns
- Optional High-Resolution Timestamping with 250MHz: 4ns

Revision History

This table shows the revision history of this document.

Version	Date	Revision
0.1	04.03.2022	First draft
0.2	25.10.2024	IEEE1588-2008 => IEEE1588-2019/2008

Table 1: Revision History

Content

Definitions

Definitions	
Ordinary Clock	A synchronization end node according to IEEE1588 that can take a Master and Slave role
Transparent Clock	A network node (Switch) that is IEEE1588 aware and compensates network jitter
Default Profile	PTP Profile according to IEEE1588
Power Profile	PTP Profile according to C37.238-2011
Utility Profile	PTP Profile according to IEC 61850 9-3
TSN Profile	PTP Profile according to IEEE802.1AS

Table 2: Definitions

Abbreviations

Abbreviations	
AXI	AMBA4 Specification (Stream and Memory Mapped)
IRQ	Interrupt, Signaling to e.g. a CPU
PRP	Parallel Redundancy Protocol (IEC 62439-3)
HSR	High-availability Seamless Redundancy (IEC 62439-3)
PTP	Precision Time Protocol (See also IEEE1588)
MAC	Media Access Controller
PHY	Physical Media Access Controller
OC	Ordinary Clock
TC	Transparent Clock
TS	Timestamp
ETH	Ethernet
TB	Testbench
LUT	Look Up Table
FF	Flip Flop

RAM	Random Access Memory
ROM	Read Only Memory
FPGA	Field Programmable Gate Array
VHDL	Hardware description Language for FPGA's

Table 3: Abbreviations

1 Introduction

1.1 Context Overview

The PTP Timestamp Unit is meant as a co-processor handling PTP timestamps on a network port. It taps or passes through the Media Independent Interface ((R)(G)MII) on the Ethernet path between the MAC, Switch or Redundancy core and PHYs where it detects and timestamps PTP traffic and optionally stores meta information and buffers the timestamps or optionally does OneStep frame modifications. It compensates the timestamps for the PHY delays and generates a configurable interrupt whenever a timestamp is ready for a specific frame type. The CPU shall use these timestamps in a PTP software stack like PTP4l, PTPd, etc. to synchronize the clock which is the base for the timestamps (Slave) or to distribute time (Master).
The PTP Timestamp Unit is specifically designed for Systems on Chip (SoC) where a CPU and FPGA part are often combined on the same silicon. In addition to the MII tap, there is also a possibility to pass the MII through the TSU for OneStep operation also there is a possibility to feed PTP timestamp signals from another source (e.g. PTP detector in a MAC), this is possible with e.g. a Xilinx® Zynq device. In this case the frame processing is omitted and only the timestamp and register part is used.
The PTP Timestamp Unit is designed to work in cooperation with the Counter Clock core from NetTimeLogic (not a requirement). It contains an AXI4Lite slave for configuration, status supervision ad fetching of the timestamps from a CPU.

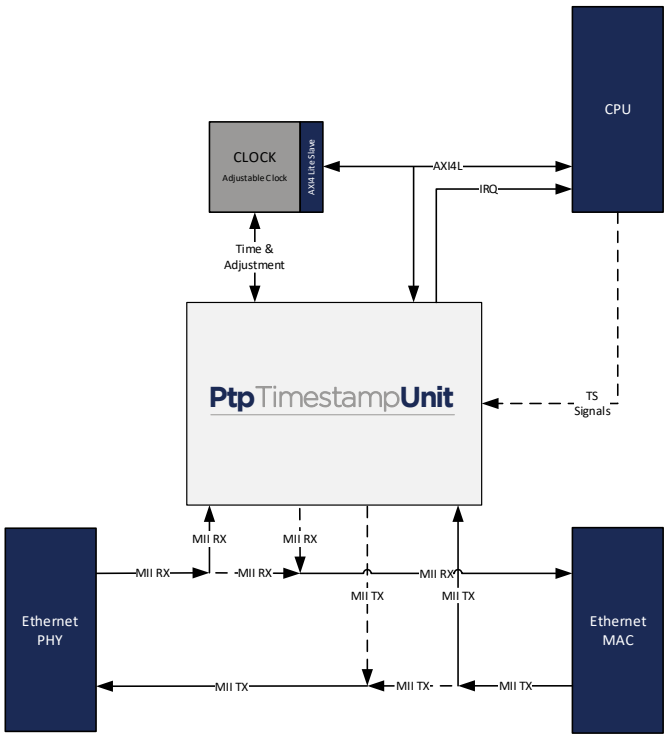

Figure 1: Context Block Diagram

1.2 Function

The PTP Timestamp Unit part is a PTP TSU according to IEEE1588-2019/2008 and IEEE802.1AS. It detects PTP frames when they pass the MII and takes timestamps of PTP event frames. It also compensates the PHY delays so the timestamps reflect the time when the PTP frames entered or left the PHY on the cable, which is the defined timestamp point according to IEEE1588-2019/2008. It can optionally also insert timestamps on the fly for OneStep operation

1.3 Architecture

For 10 Gigabit the PTP Timestamp Unit was extended by a 10G XGMII wrapper which basically handles all 10G traffic and just connects the PTP Timestamp Unit to it so it can inject PTP traffic and receives PTP traffic only, allowing the PTP core to run on a much slower link speed rate since PTP message rates are well below 10G. This resulted in the following Architecture:

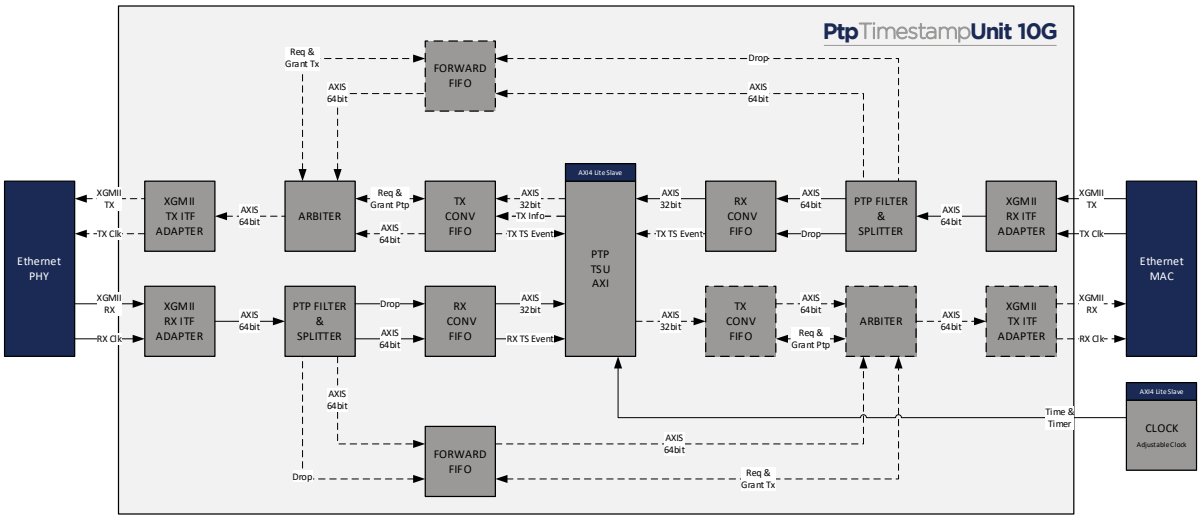

Figure 2: Architecture Block Diagram

XGMII TX Interface Adapter

The XGMII TX Interface Adapter converts a 64bit AXI stream running on the XGMII clock (156,25 MHz) to XGMII stream in 64bit mode (without DDR). It generates the interframe gap of minimum 12 byte times, adds the Preamble and Start of Frame Delimiter (SFD) and encodes the incoming AXI stream to the XGMII interface (64bit data, 8bit control) using the control lines and special characters according to the XGMII specification. Start of frame shall always be on Lane 0.
It is instantiated on the TX side of the PCS and on the RX side of the MAC.

XGMII RX Interface Adapter

The XGMII RX Interface Adapter converts a XGMII stream in 64bit mode (without DDR) to a 64bit AXI stream running on the XGMII clock (156,25 MHz). It removes the Preamble and Start of Frame Delimiter (SFD) and decodes the incoming XGMII stream (64bit data, 8bit control) to the AXI stream interface using the control lines and special characters according to the XGMII specification. Start of frame shall always be on Lane 0 (or 4).
It is instantiated on the RX side of the PCS and on the TX side of the MAC.

Arbiter

The Arbiter arbitrates between the 10G forwarding path from the MAC to the PCS and the PTP TSU core. It has request and grant lines and will do arbitration in a round robin manner so that no path is starving.

PTP Filter and Splitter

This module splits the 64bit AXI stream coming from the XGMII RX Interface Adapter into a 64bit AXI stream for the TSU and a 64bit AXI stream for the forwarding path. It contains a frame parser which will check if a frame is a PTP frame or not and will assert the according drop signal towards one of the connected FIFOs.

TX Conversion FIFO

The TX Conversion FIFO has multiple functionalities. It converts the 32bit AXI stream from the PTP TSU to a 64bit AXI stream and uses an asynchronous store-and-forward FIFO to do the clock domain crossing and to be able to provide the frame with 10G towards the XGMII TX Interface Adapter.
In addition, it detects the start of the frame and signals this to the PTP core (this is done here since the path towards the XGMII Interface adapter is fully deterministic and it is also a preparation for the one-step mode). The challenge here is to align the timestamp event signal to the frame without inserting additional jitter.

RX Conversion FIFO

The RX Conversion FIFO has also multiple functionalities. It converts the 64bit AXI stream from the XGMII Interface Adapter to a 32bit AXI stream and uses an asynchronous store-and-forward FIFO to do the clock domain crossing and to be able to handle the frame with 10G from the XGMII RX Interface Adapter before converting to the slower clock speed. It also has a drop input which is used to explicitly drop the incoming frame. This is used by the PTP Filter and Splitter to drop Non-PTP frames towards the PTP TSU.
In addition, it detects the start of the frame and signals this to the PTP core (this is done here since the path from the XGMII Interface adapter is fully deterministic). The challenge here is to align the timestamp event signal to the frame without inserting additional jitter.

Forwarding FIFOs

This is a store-and-forward or cut-through frame drop FIFO, which means it can drop a whole frame if it runs into an overflow condition. This is required since the PTP core will use a small amount of bandwidth which could lead to an overload condition. It also has a drop input which is used to explicitly drop the incoming frame. This is used by the PTP Filter and Splitter to drop PTP frames on the forwarding path. It also has a request and grant signal which is used for the arbitration. It is instantiated on the RX and TX forwarding path.

PTP Timestamp Unit

This is the actual PTP Timestamp Unit IP core from NetTimeLogic, it receives and transmits PTP frames via 32bit AXI stream interfaces and has timestamp inputs which can trigger a TX or RX timestamp in the core when the SFD on the corresponding path was detected, it provides the TX frame length and receives the calculated RX and TX delays with the timestamp indications.

2 Design Description

The following chapters describe the internals of the PTP Timestamp Unit 10G: starting with the Top Level, which is a collection of subcores, followed by the description of all subcores.

2.1 Top Level – PTP Ordinary Clock

2.1.1.1 Parameters

The core must be parametrized at synthesis time. It has the same parameters as the Non-10G version, additionally the following parameters are available.

Name	Type	Size	Description
RxDelayNanosecond 10000_Gen	integer	1	PHY receive delay (10Gbit)
TxDelayNanosecond 10000_Gen	integer	1	PHY transmit delay (10Gbit)

Table 4: Parameters

2.1.1.2 Entity Block Diagram

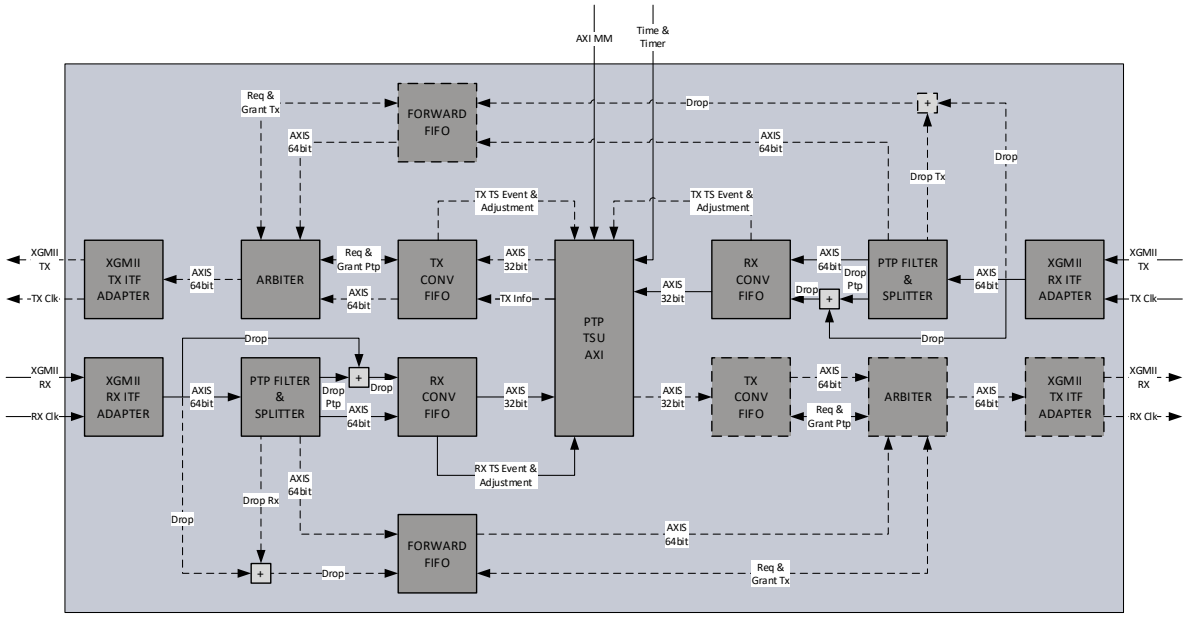

Figure 3: PTP Timestamp Unit 10G

2.1.1.3 Entity Description

XGMII TX Interface Adapter

The XGMII TX Interface Adapter converts a 64bit AXI stream running on the XGMII clock (156,25 MHz) to XGMII stream in 64bit mode (without DDR). It generates the interframe gap of minimum 12 byte times, adds the Preamble and Start of Frame Delimiter (SFD) and encodes the incoming AXI stream to the XGMII interface (64bit data, 8bit control) using the control lines and special characters according to the XGMII specification. Start of frame shall always be on Lane 0.
It is instantiated on the TX side of the PCS and on the RX side of the MAC.
See 2.2.1 for more details.

XGMII RX Interface Adapter

The XGMII RX Interface Adapter converts a XGMII stream in 64bit mode (without DDR) to a 64bit AXI stream running on the XGMII clock (156,25 MHz). It removes the Preamble and Start of Frame Delimiter (SFD) and decodes the incoming XGMII stream (64bit data, 8bit control) to the AXI stream interface using the control lines and special characters according to the XGMII specification. Start of frame shall always be on Lane 0 (or 4).
It is instantiated on the RX side of the PCS and on the TX side of the MAC.
See 2.2.1 for more details.

Arbiter

The Arbiter arbitrates between the 10G forwarding path from the MAC to the PCS and the PTP TSU core. It has request and grant lines and will do arbitration in a round robin manner so that no path is starving.
See 2.2.2 for more details.

PTP Filter and Splitter

This module splits the 64bit AXI stream coming from the XGMII RX Interface Adapter into a 64bit AXI stream for the TSU and a 64bit AXI stream for the forwarding path. It contains a frame parser which will check if a frame is a PTP frame or not and will assert the according drop signal towards one of the connected FIFOs.
See 2.2.3 for more details.

TX Conversion FIFO

The TX Conversion FIFO has multiple functionalities. It converts the 32bit AXI stream from the PTP TSU to a 64bit AXI stream and uses an asynchronous store-

and-forward FIFO to do the clock domain crossing and to be able to provide the frame with 10G towards the XGMII TX Interface Adapter.
In addition, it detects the start of the frame and signals this to the PTP core (this is done here since the path towards the XGMII Interface adapter is fully deterministic and it is also a preparation for the one-step mode). The challenge here is to align the timestamp event signal to the frame without inserting additional jitter.
See 2.2.4 for more details.

RX Conversion FIFO

The RX Conversion FIFO has also multiple functionalities. It converts the 64bit AXI stream from the XGMII Interface Adapter to a 32bit AXI stream and uses an asynchronous store-and-forward FIFO to do the clock domain crossing and to be able to handle the frame with 10G from the XGMII RX Interface Adapter before converting to the slower clock speed. It also has a drop input which is used to explicitly drop the incoming frame. This is used by the PTP Filter and Splitter to drop Non-PTP frames towards the PTP TSU.
In addition, it detects the start of the frame and signals this to the PTP core (this is done here since the path from the XGMII Interface adapter is fully deterministic). The challenge here is to align the timestamp event signal to the frame without inserting additional jitter.
See 2.2.5 for more details.

Forwarding FIFOs

This is a store-and-forward or cut-through frame drop FIFO, which means it can drop a whole frame if it runs into an overflow condition. This is required since the PTP core will use a small amount of bandwidth which could lead to an overload condition. It also has a drop input which is used to explicitly drop the incoming frame. This is used by the PTP Filter and Splitter to drop PTP frames on the forwarding path. It also has a request and grant signal which is used for the arbitration. It is instantiated on the RX and TX forwarding path.
See 2.2.6 for more details.

PTP Ordinary Clock

This is the actual PTP Timestamp Unit IP core from NetTimeLogic, it receives and transmits PTP frames via 32bit AXI stream interfaces and has timestamp inputs which can trigger a TX or RX timestamp in the core when the SFD on the corresponding path was detected, it provides the TX frame length and receives the calculated RX and TX delays with the timestamp indications.

See the PTP Timestamp Unit Manual for details.

2.1.1.4 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
PassThrough_Gen	-	boolean	1	Whether the MII is passed through or tapped
OneStepSupport_Gen	-	boolean	1	Whether OnStep support shall be there, requires pass through
DefaultProfile Support_Gen	-	boolean	1	Support for Default Profile
PowerProfile Support_Gen	-	boolean	1	Support for Power Profile
UtilityProfile Support_Gen	-	boolean	1	Support for Utility Profile
TsnProfile Support_Gen	-	boolean	1	Support for IEEE802.1AS
UnicastProfile Support_Gen	-	boolean	1	Support for Unicast PTP messages
Layer2Support_Gen	-	boolean	1	Support for Layer 2 Mapping
Lay-er3v4Support_Gen	-	boolean	1	Support for IPv4 Mapping
Lay-er3v6Support_Gen	-	boolean	1	Support for IPv6 Mapping
RedTagSupport_Gen	-	boolean	1	If HSR or FRER tags shall be detected
MetaInfo_Gen	-	boolean	1	If Meta Information shall be stored with the timestamp (this is required if Buffer-ing shall be done)

DelayReqRxBuffer Depth_Gen	-	natural	1	Buffer Depth for Delay Request RX Timestamps. 0 = No buffering (>0 requires MetaInfo_Gen = true)
DelayReqTxBuffer Depth_Gen	-	natural	1	Buffer Depth for Delay Request TX Timestamps. 0 = No buffering (>0 requires MetaInfo_Gen = true)
PDelayReqRxBuffer Depth_Gen	-	natural	1	Buffer Depth for Peer Delay Request RX Timestamps. 0 = No buffering (>0 requires MetaInfo_Gen = true)
PDelayReqTxBuffer Depth_Gen	-	natural	1	Buffer Depth for Peer Delay Request TX Timestamps. 0 = No buffering (>0 requires MetaInfo_Gen = true)
PDelayRespRxBuffer Depth_Gen	-	natural	1	Buffer Depth for Peer Delay Response RX Timestamps. 0 = No buffering (>0 requires MetaInfo_Gen = true)
PDelayRespTxBuffer Depth_Gen	-	natural	1	Buffer Depth for Peer Delay Response TX Timestamps. 0 = No buffering (>0 requires MetaInfo_Gen = true)

SyncRxBuffer-Depth_Gen	-	natural	1	Buffer Depth Sync RX Timestamps. 0 = No buffering (>0 requires MetaIn-fo_Gen = true)
SyncTxBuffer-Depth_Gen	-	natural	1	Buffer Depth for Sync TX Timestamps. 0 = No buffering (>0 re-quires MetaIn-fo_Gen = true)
ClockClkPeriodNano-second_Gen	-	natural	1	Integer Clock Period
RxDelayNanosecond 10000_Gen	-	integer	1	RX Delay of the PHY in Nanosecond
TxDelayNanosecond 10000_Gen	-	integer	1	TX Delay of the PHY in Nanosecond
HighResSupport_Gen	-	boolean	1	If a high-resolution clock SysClkNx with alignment to SysClk is used
HighResFreq Multiply_Gen	-	natural	1	Multiplication factor of the high-resolution clock compared to SysClk
AxiAddressRange Low_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size
Sim_Gen	-	boolean	1	If in Testbench simulation mode
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysClkNx_ClkIn	in	std_logic	1	High-resolution clock (multiple of Sys Clock)

SysRstN_RstIn	in	std_logic	1	System Reset
Time Input				
ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted PTP Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted PTP Clock Time valid
XGMii RX Clk/Rst Input				
XGMiiRxClk_ClkIn	in	std_logic	1	RX Clock
XGMiiRxRstN_RstIn	in	std_logic	1	Reset aligned with RX Clock
XGMii TX Clk/Rst Input				
XGMiiTxClk_ClkIn	in	std_logic	1	TX Clock
XGMiiTxRstN_RstIn	in	std_logic	1	Reset aligned with TX Clock
XGMii RX Data Input/Output				
XGMiiRxCtl_Ena	in/ out	std_logic	8	RX Data valid
XGMiiRxData_Dat	in/ out	std_logic_vector	64	RX Data
XGMii TX Data Input/Output				
XGMiiTxCtl_Ena	in/ out	std_logic	1	TX Data valid
XGMiiTxData_Dat	in/ out	std_logic_vector	64	TX Data
AXI4 Lite Slave				
AxiWriteAddrValid _ValIn	in	std_logic	1	Write Address Valid
AxiWriteAddrReady _RdyOut	out	std_logic	1	Write Address Ready
AxiWriteAddrAddress _AdrIn	in	std_logic_vector	32	Write Address
AxiWriteAddrProt _DatIn	in	std_logic_vector	3	Write Address Protocol
AxiWriteDataValid _ValIn	in	std_logic	1	Write Data Valid
AxiWriteDataReady _RdyOut	out	std_logic	1	Write Data Ready
AxiWriteDataData _DatIn	in	std_logic_vector	32	Write Data
AxiWriteDataStrobe _DatIn	in	std_logic_vector	4	Write Data Strobe
AxiWriteRespValid _ValOut	out	std_logic	1	Write Response Valid

AxiWriteRespReady_RdyIn	in	std_logic	1	Write Response Ready
AxiWriteResp Response_DatOut	out	std_logic_vector	2	Write Response
AxiReadAddrValid_ValIn	in	std_logic	1	Read Address Valid
AxiReadAddrReady_RdyOut	out	std_logic	1	Read Address Ready
AxiReadAddrAddress_AdrIn	in	std_logic_vector	32	Read Address
AxiReadAddrProt_DatIn	in	std_logic_vector	3	Read Address Protocol
AxiReadDataValid_ValOut	out	std_logic	1	Read Data Valid
AxiReadDataReady_RdyIn	in	std_logic	1	Read Data Ready
AxiReadData Response_DatOut	out	std_logic_vector	2	Read Data
AxiReadDataData_DatOut	out	std_logic_vector	32	Read Data Response
Interrupt Output				
Irq_EvtOut	out	std_logic	1	Level high Interrupt

Table 5: PTP Timestamp Unit XGMII

2.2 Design Parts

The PTP Timestamp Unit core consists of a couple of subcores. Each of the subcores itself consist again of smaller function block. The following chapters describe these subcores and their functionality.

2.2.1 XGMII Interface Adapter

2.2.1.1 Entity Block Diagram

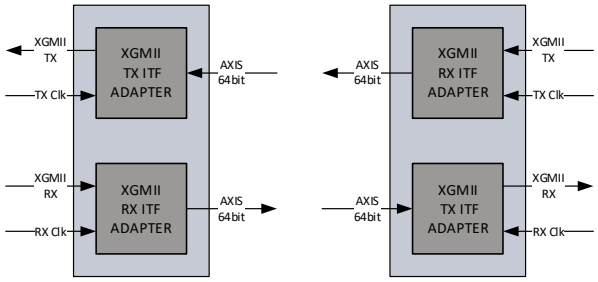

Figure 4: XGMII Interface Adapter

2.2.1.2 Entity Description

To have a deterministic delay, to have only a minimum of clock domains in the design and to save resources, the AXI stream interface and the XGMII interface shall run on the same clock per direction (RX/TX).
There are two different versions of the interface adapter, a MAC and PHY side version. One is connected to the PHY and the other one to the MAC. Internally the same components are used to handle XGMII, it is only a naming scheme.

For the interface adapter, all clocks are inputs, even if on a higher level the clock is passed through, so it will be also an output of the overall design.

XGMII TX Interface Adapter

The XGMII TX Interface Adapter converts a 64bit AXI stream running on the XGMII clock (156,25 MHz) to XGMII stream in 64bit mode (without DDR). It generates the interframe gap of minimum 12 byte times, adds the Preamble and Start of Frame Delimiter (SFD) and encodes the incoming AXI stream to the XGMII interface (64bit data, 8bit control) using the control lines and special characters according to the XGMII specification. Start of frame shall always be on Lane 0.
It is instantiated on the TX side of the PCS and on the RX side of the MAC.

XGMII RX Interface Adapter

The XGMII RX Interface Adapter converts a XGMII stream in 64bit mode (without DDR) to a 64bit AXI stream running on the XGMII clock (156,25 MHz). It removes the Preamble and Start of Frame Delimiter (SFD) and decodes the incoming XGMII stream (64bit data, 8bit control) to the AXI stream interface using the control lines and special characters according to the XGMII specification. Start of frame shall always be on Lane 0 (or 4).
It is instantiated on the RX side of the PCS and on the TX side of the MAC.

2.2.1.3 Entity Declaration

Name	Dir	Type	Size	Description
Ports				
XGMii RX Clk/Rst Input				
XGMiiRxClk_ClkIn	in	std_logic	1	RX Clock
XGMiiRxRstN_RstIn	in	std_logic	1	Reset aligned with RX Clock
XGMii TX Clk/Rst Input				
XGMiiTxClk_ClkIn	in	std_logic	1	TX Clock
XGMiiTxRstN_RstIn	in	std_logic	1	Reset aligned with TX Clock
XGMii RX Data Input				
XGMiiRxCtl_EnaIn	in	std_logic	8	RX Data valid
XGMiiRxData_DatIn	in	std_logic_vector	64	RX Data
XGMii TX Data Output				
XGMiiTxCtl_EnaOut	out	std_logic	1	TX Data valid
XGMiiTxData_DatOut	out	std_logic_vector	64	TX Data
Axi Input				
AxisValid_ValIn	in	std_logic	1	AXI Stream frame input
AxisReady_ValOut	out	std_logic	1	
AxisData_DatIn	in	std_logic_vector	64	
AxisStrobe_ValIn	in	std_logic_vector	8	
AxisKeep_ValIn	in	std_logic_vector	8	
AxisLast_ValIn	in	std_logic	1	
AxisUser_DatIn	in	std_logic_vector	3	
Axi Output				
AxisValid_ValOut	out	std_logic	1	AXI Stream frame output
AxisReady_ValIn	in	std_logic	1	
AxisData_DatOut	out	std_logic_vector	64	
AxisStrobe_ValOut	out	std_logic_vector	8	

AxisKeep_ValOut	out	std_logic_vector	8	
AxisLast_ValOut	out	std_logic	1	
AxisUser_DatOut	out	std_logic_vector	3	

Table 6: MAC XGMII Adapter

Name	Dir	Type	Size	Description
Ports				
XGMii RX Clk/Rst Input				
XGMiiRxClk_ClkIn	in	std_logic	1	RX Clock
XGMiiRxRstN_RstIn	in	std_logic	1	Reset aligned with RX Clock
XGMii TX Clk/Rst Input				
XGMiiTxClk_ClkIn	in	std_logic	1	TX Clock
XGMiiTxRstN_RstIn	in	std_logic	1	Reset aligned with TX Clock
XGMii RX Data Output				
XGMiiRxCtl_EnaOut	out	std_logic	8	RX Data valid
XGMiiRxData_DatOut	out	std_logic_vector	64	RX Data
XGMii TX Data Input				
XGMiiTxCtl_EnaIn	in	std_logic	1	TX Data valid
XGMiiTxData_DatIn	in	std_logic_vector	64	TX Data
Axi Input				
AxisValid_ValIn	in	std_logic	1	AXI Stream frame input
AxisReady_ValOut	out	std_logic	1	
AxisData_DatIn	in	std_logic_vector	64	
AxisStrobe_ValIn	in	std_logic_vector	8	
AxisKeep_ValIn	in	std_logic_vector	8	
AxisLast_ValIn	in	std_logic	1	
AxisUser_DatIn	in	std_logic_vector	3	
Axi Output				
AxisValid_ValOut	out	std_logic	1	AXI Stream frame output
AxisReady_ValIn	in	std_logic	1	
AxisData_DatOut	out	std_logic_vector	64	
AxisStrobe_ValOut	out	std_logic_vector	8	
AxisKeep_ValOut	out	std_logic_vector	8	
AxisLast_ValOut	out	std_logic	1	
AxisUser_DatOut	out	std_logic_vector	3	

Table 7: PHY XGMII Adapter

2.2.2 Arbiter

2.2.2.1 Entity Block Diagram

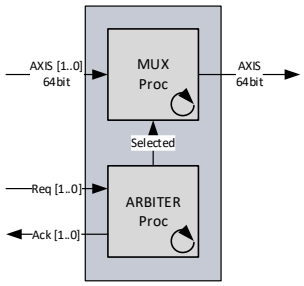

Figure 5: Arbiter

2.2.2.2 Entity Description

The Arbiter was designed to have minimal delay and maximum throughput since it needs to provide valid Data at every clock cycle once started and shall not slow down the forwarding path.
In the design the Arbiter is controlled by two Forward FIFOs in Store-And-Forward mode. As soon as the FIFO has valid data available it will assert the Req line and will wait with providing data with the valid signal until the Ack is set. At the end of the frame if there is still data available in the FIFO it will assert the Req line again. This assures a zero delay switchover or when frames come back to back. The Ack signal is changing with one clock cycle delay when access is granted.
To guarantee that no path starves it switches between the paths if both of them have ready data to send, this also guarantees that the maximum delay one port has to wait until it is served again is the maximum size of one frame (if both request only when data is valid). If no path wanted to send and at the next moment both of them want to send, path 1 has priority over path 2. Back-pressure is directly forwarded to the path selected. The path not selected doesn't get the Ack line set and the AXI ready signal is low. If no path is selected (after the last frame was sent and no path has data ready to send) the Ack signals are low and the AXI ready signal to both input ports is low.

2.2.2.3 Entity Declaration

Name	Dir	Type	Size	Description
Ports				
System				

XGMiiClk_ClkIn	in	std_logic	1	XGMII Clock
XGMiiRstN_RstIn	in	std_logic	1	XGMII Reset
Arbiter Request Input				
Req_ValIn	in	std_logic_vector	2	Request access
Arbiter Acknowledge Output				
Ack_ValOut	in	std_logic_vector	2	Acknowledge access
Axi Input				
AxisValid_ValIn	in	Axis64_Itf Valid_Type	2	AXI Stream frame input
AxisReady_ValOut	out	Axis64_Itf Ready_Type	2	
AxisData_DatIn	in	Axis64_Itf Data_Type	2	
AxisStrobe_ValIn	in	Axis64_Itf Strobe_Type	2	
AxisKeep_ValIn	in	Axis64_Itf Keep_Type	2	
AxisLast_ValIn	in	Axis64_Itf Last_Type	2	
AxisUser_DatIn	in	Axis64_Itf User_Type	2	
Axi Output				
AxisValid_ValOut	out	std_logic	1	AXI Stream frame output
AxisReady_ValIn	in	std_logic	1	
AxisData_DatOut	out	std_logic_vector	64	
AxisStrobe_ValOut	out	std_logic_vector	8	
AxisKeep_ValOut	out	std_logic_vector	8	
AxisLast_ValOut	out	std_logic	1	
AxisUser_DatOut	out	std_logic_vector	3	

Table 8: Arbiter

2.2.3 PTP Filter and Splitter

2.2.3.1 Entity Block Diagram

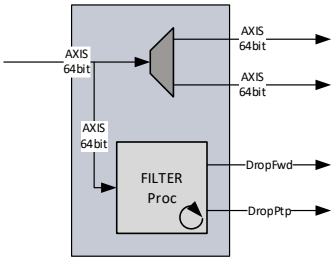

Figure 6: PTP Filter and Splitter

2.2.3.2 Entity Description

The Filter and Splitter has two functionalities: The first functionality is that it splits the AXI stream into two AXI streams and makes sure that data transfers are done when both output AXI streams are ready, even though the output AXI streams will always signal valid because the Arbiter and Splitter is connected to two Forwarding FIFOs which signal ready always.
The second functionality is that it parses the incoming frame to check if it is a PTP frame or not. The parser checks several things:

- Destination MAC for the dedicate PTP Multicast Mac Addresses
- Etherthertype dedicated to PTP if in Layer 2 mode
- Destination IP Addresses for the dedicated PTP Multicast IP Addresses and UDP Destination Ports dedicated to PTP if in Layer 3 (UDP/IPv4 or IPv6) mode
- VLAN if used, which shifts byte alignment by 4
- Redundancy Tags, which shifts byte alignment by 6

If it is a PTP frame it asserts the drop signal for the forwarding path. If it is not a PTP frame it will assert the drop signal for the PTP receive path.
The drop signal shall be asserted for a single clock cycle whenever the frame type is detected.
The parser must be able to run at full speed without the need for backpressure.

2.2.3.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				

UnicastProfile Support_Gen	-	Boolean	1	If Unicast Frames shall be handled
Ports				
System				
XGMiiClk_ClkIn	in	std_logic	1	XGMII Clock
XGMiiRstN_RstIn	in	std_logic	1	XGMII Reset
Arbiter Acknowledge Output				
Drop_ValOut	out	std_logic_vector	2	Drop indication if it is a PTP frame or not. It will either be forwarded to the MAC or the PTP core (either or)
Axi Input				
AxisValid_ValIn	in	std_logic	1	AXI Stream frame input
AxisReady_ValOut	out	std_logic	1	
AxisData_DatIn	in	std_logic_vector	64	
AxisStrobe_ValIn	in	std_logic_vector	8	
AxisKeep_ValIn	in	std_logic_vector	8	
AxisLast_ValIn	in	std_logic	1	
AxisUser_DatIn	in	std_logic_vector	3	
Axi Output				
AxisValid_ValOut	out	Axis64_Itf Valid_Type	2	AXI Stream frame output
AxisReady_ValIn	in	Axis64_Itf Ready_Type	2	
AxisData_DatOut	out	Axis64_Itf Data_Type	2	
AxisStrobe_ValOut	out	Axis64_Itf Strobe_Type	2	
AxisKeep_ValOut	out	Axis64_Itf Keep_Type	2	
AxisLast_ValOut	out	Axis64_Itf Last_Type	2	
AxisUser_DatOut	out	Axis64_Itf User_Type	2	

Table 9: PTP Filter and Splitter

2.2.4 TX Conversion FIFO

2.2.4.1 Entity Block Diagram

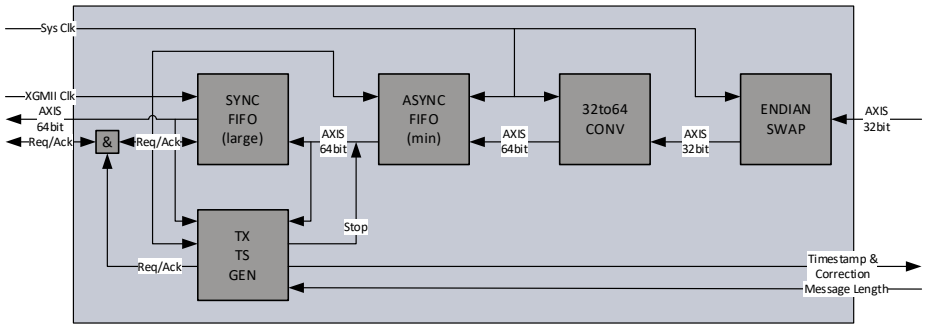

Figure 7: TX Conversion FIFO

2.2.4.2 Entity Description

The two Conversion FIFOs mainly differ on the Timestamp handling.
Both Conversion FIFOs handle the clock domain crossing between the XGMII clock (RX or TX) and the System Clock with a Cut-Through Asynchronous FIFO, handle the width conversion between a 32bit and 64bit AXI stream, handle the speed difference between 10G and ~1G with a Store-And-Forward Synchronous FIFO as well as the byte order on AXI stream. The PTP part expects the MSB in the highest valid Byte of AXI stream, which is the inversed order compared to standard AXI stream.

The purpose of the Synchronous FIFO in the TX Conversion FIFO has also two purposes: first, it guarantees that a PTP frame can be sent fast enough on the path towards the Interface Adapter since the Interface Adapter requires a continuous stream once started, second it allows to have a determinist forwarding behavior after the timestamp point.

The TX Timestamper waits until data is valid from the Asynchronous FIFO and immediately stops the transfer. At this moment the PTP core already provided the frame length of the frame that it started to send which is then stored and the request line asserted towards the Arbiter and waited for the acknowledge.
As soon as the acknowledge from the Arbiter comes the transfer is released. It then calculates the correction value of the timestamps based on the frame length and asserts the timestamp event signal together with the correction value. It then waits until the last byte is pushed to the Synchronous FIFO and then stops the transfer again. It then waits until the frame starts towards the Interface adapter, releases

the request signal and waits until it is completed, then releases the transfer for the next frame. This way, the delay from the point the acknowledge comes until the frame is sent towards the Interface Adapter is determinist, based on the frame length and some other delays on the PTP side. In addition, it has a generic for the TX PHY delay which is added to the correction value.
This mechanism allows one-step and two-step timestamping

The PTP TSU needed the following modifications:
The timestamping part needs to use the correction values in its calculations.
The TX frame parser needs to provide a calculated frame length to the TX FIFO and the alignment has to be that the frame length is valid when the first AXI transfer is valid.
The frame and timestamp alignment have to be ensured to work with the Conversion FIFOs

2.2.4.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
NoTimestamping_Gen	-	boolean	1	If no Timestamp shall be generated and only the conversion
Depth_Gen	-	natural	1	Sync FIFO Size
Ready_Gen	-	boolean	1	If the FIFO shall be able to make back pressure
AdditionalDelay_Gen	-	natural	1	PHY TX Delay
ClockClkPeriodNanosecond_Gen	-	natural	1	Integer Clock Period
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
XGMii Clk/Rst Input				
XGMiiClk_ClkIn	in	std_logic	1	XGMII Clock
XGMiiRstN_RstIn	in	std_logic	1	Reset aligned with

				XGMII Clock
Arbiter Request Output				
Req_ValOut	out	std_logic	1	Request access
Arbiter Acknowledge Input				
Ack_ValIn	in	std_logic	1	Acknowledge access
Timestamp Output				
Timestamp Event_EvtOut	out	std_logic	1	Timestamp event
Timestamp Correction_DatOut	out	std_logic_vector	32	Correction value to the Timestamp
Frame Length Input				
FrameLength_DatIn	in	std_logic_vector	16	Frame Length of the PTP frame on AXI
FrameLength_ValIn	in	std_logic	1	Frame Length valid
Axi Input				
AxisValid_ValIn	in	std_logic	1	AXI Stream frame input
AxisReady_ValOut	out	std_logic	1	
AxisData_DatIn	in	std_logic_vector	32	
AxisStrobe_ValIn	in	std_logic_vector	4	
AxisKeep_ValIn	in	std_logic_vector	4	
AxisLast_ValIn	in	std_logic	1	
AxisUser_DatIn	in	std_logic_vector	3	
Axi Output				
AxisValid_ValOut	out	std_logic	1	AXI Stream frame output
AxisReady_ValIn	in	std_logic	1	
AxisData_DatOut	out	std_logic_vector	64	
AxisStrobe_ValOut	out	std_logic_vector	8	
AxisKeep_ValOut	out	std_logic_vector	8	
AxisLast_ValOut	out	std_logic	1	
AxisUser_DatOut	out	std_logic_vector	3	

Table 10: TX Conversion FIFO

2.2.5 RX Conversion FIFO

2.2.5.1 Entity Block Diagram

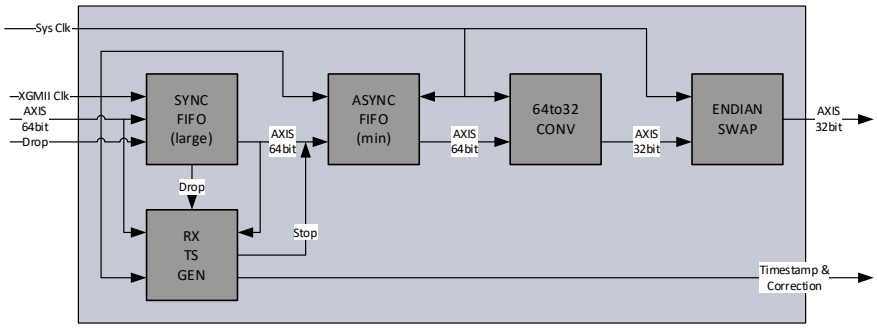

Figure 8: RX Conversion FIFO

2.2.5.2 Entity Description

The two Conversion FIFOs mainly differ on the Timestamp handling.
Both Conversion FIFOs handle the clock domain crossing between the XGMII clock (RX or TX) and the System Clock with a Cut-Through Asynchronous FIFO, handle the width conversion between a 32bit and 64bit AXI stream, handle the speed difference between 10G and ~1G with a Store-And-Forward Synchronous FIFO as well as the byte order on AXI stream. The PTP part expects the MSB in the highest valid Byte of AXI stream, which is the inversed order compared to standard AXI stream.

The purpose of the Synchronous FIFO in the RX Conversion FIFO has two purposes: it allows to drop Non-PTP frames and it does the speed conversion from 10G to 1G also allowing bursts of PTP Frames since the Interface Adapter can not be slowed down.

The RX Timestamper has an internal counter (20bit) which is incremented by 1 for each clock cycle (wrapping though zero=. When a start of a frame is detected on the path from the Interface Adapter it stores the counter value and if the frame is not dropped (due to overflow condition or if Non-PTP) this counter value is pushed to a FIFO which can store as many counter snapshots as 64byte sized frame can be in the Synchronous FIFO. Once a frame starts towards the PTP core, it takes another snapshot of the counter and popes the counter value from the FIFO. It then calculates the difference between the counter values and converts it then into the timestamp correction value in nanoseconds and asserts it together with the timestamp event signal. It then waits until the frame has been completely pushed

to the FIFO and stops the transfer for some time to allow the PTP core to do its calculations even with back to back frames. In addition, it has a generic for the RX PHY delay which is added to the correction value.

2.2.5.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
NoTimestamping_Gen	-	boolean	1	If no Timestamp shall be generated and only the con-version
Depth_Gen	-	natural	1	Sync FIFO Size
Ready_Gen	-	boolean	1	If the FIFO shall be able to make back pressure
AdditionalDelay_Gen	-	natural	1	PHY TX Delay
ClockClkPeriodNano-second_Gen	-	natural	1	Integer Clock Period
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
XGMii Clk/Rst Input				
XGMiiClk_ClkIn	in	std_logic	1	XGMII Clock
XGMiiRstN_RstIn	in	std_logic	1	Reset aligned with XGMII Clock
Drop Input				
Drop_ValIn	in	std_logic	1	If frame will be dropped
Timestamp Output				
Timestamp Event_EvtOut	out	std_logic	1	Timestamp event
Timestamp Correction_DatOut	out	std_logic_vector	32	Correction value to the Timestamp
Axi Input				
AxisValid_ValIn	in	std_logic	1	AXI Stream frame input
AxisReady_ValOut	out	std_logic	1	

AxisData_DatIn	in	std_logic_vector	64	
AxisStrobe_ValIn	in	std_logic_vector	8	
AxisKeep_ValIn	in	std_logic_vector	8	
AxisLast_ValIn	in	std_logic	1	
AxisUser_DatIn	in	std_logic_vector	3	
Axi Output				
AxisValid_ValOut	out	std_logic	1	AXI Stream frame output
AxisReady_ValIn	in	std_logic	1	
AxisData_DatOut	out	std_logic_vector	32	
AxisStrobe_ValOut	out	std_logic_vector	4	
AxisKeep_ValOut	out	std_logic_vector	4	
AxisLast_ValOut	out	std_logic	1	
AxisUser_DatOut	out	std_logic_vector	3	

Table 11: RX Conversion FIFO

2.2.6 Forwarding FIFO

2.2.6.1 Entity Block Diagram

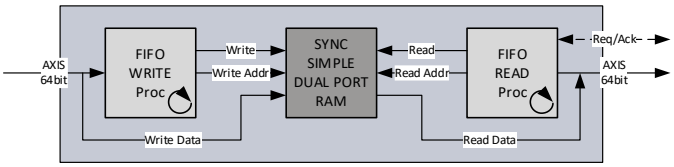

Figure 9: Forwarding FIFO

2.2.6.2Entity Description

The FIFO was designed to have minimal delay and maximum throughput since it needs to provide valid Data at every clock cycle once started.
It consists of a Read and Write process which access a Synchronous Simple Dual Port RAM (one read and one write port).
The RAM is 76bits wide: 64bit Data, 8bit Keep, 1bit Last, 3bit User
The FIFO could generate a ready signal on the input if it runs into a full condition, however this is disabled via a generic since the source can not be slowed down (Interface adapter). If the FIFO runs into an overflow condition it will drop the whole frame currently coming in. This ensures that only valid frames are forwarded. Droping on overflow is done for both Store-And-Forward and Cut-Through FIFO types. This requires, that for the Cut-Through case the FIFO must be large enough to store a maximum sized frame. The Store-And-Forward FIFO has an additional Drop input which allows to drop an incoming frame. The Drop signal can be set until the Last signal is asserted. After this the frame is stored and can not be dropped. The AXI stream is synchronous to the corresponding XGMII clock per direction (RX/TX). There is no clock domain crossing in the FIFO. Optionally it has a Request and Acknowledge signal for Arbitration. The Request signal will be asserted after the last frame has completed and new data is ready to be sent. As long as the Request signal is asserted but no Acknowledge is done, no AXI transfer will happen.

2.2.6.3Entity Declaration

Name	Dir	Type	Size	Description
Ports				

XGMii Clk/Rst Input				
XGMiiClk_ClkIn	in	std_logic	1	XGMII Clock
XGMiiRstN_RstIn	in	std_logic	1	Reset aligned with XGMII Clock
Arbiter Request Output				
Req_ValOut	out	std_logic	1	Request access
Arbiter Acknowledge Input				
Ack_ValIn	in	std_logic	1	Acknowledge access
Drop Input				
Drop_ValIn	in	std_logic	8	Asserted when an incoming frame shall be dropped
Drop Output				
Drop_ValOut	out	std_logic	1	Asserted when an incoming frame is dropped
Axi Input				
AxisValid_ValIn	in	std_logic	1	AXI Stream frame input
AxisReady_ValOut	out	std_logic	1	
AxisData_DatIn	in	std_logic_vector	64	
AxisStrobe_ValIn	in	std_logic_vector	8	
AxisKeep_ValIn	in	std_logic_vector	8	
AxisLast_ValIn	in	std_logic	1	
AxisUser_DatIn	in	std_logic_vector	3	
Axi Output				
AxisValid_ValOut	out	std_logic	1	AXI Stream frame output
AxisReady_ValIn	in	std_logic	1	
AxisData_DatOut	out	std_logic_vector	64	
AxisStrobe_ValOut	out	std_logic_vector	8	
AxisKeep_ValOut	out	std_logic_vector	8	
AxisLast_ValOut	out	std_logic	1	
AxisUser_DatOut	out	std_logic_vector	3	

Table 12: Forwarding FIFO

2.3 Clocking and Reset Concept

2.3.1 Clocking

To keep the design as robust and simple as possible, the whole Ordinary Clock, including the Counter Clock and all other cores from NetTimeLogic are run in one clock domain. This is considered to be the system clock. Per Default this clock is 50MHz. Where possible also the interfaces are run synchronous to this clock. For clock domain crossing asynchronous Fifos with gray counters or message patterns with meta-stability flip-flops are used. Clock domain crossings for the AXI interface is moved from the AXI slave to the AXI interconnect.

Clock	Frequency	Description
System		
System Clock	50MHz (Default)	System clock where the TSU runs on as well as the counter clock etc.
XGMII Interface		
XGMII RX Clock	156.25MHz	Asynchronous, external receive clock
XGMII TX Clock	156.25MHz	Asynchronous, external transmit clock
AXI Interface		
AXI Clock	50MHz (Default)	Internal AXI bus clock, same as the system clock

Table 13: Clocks

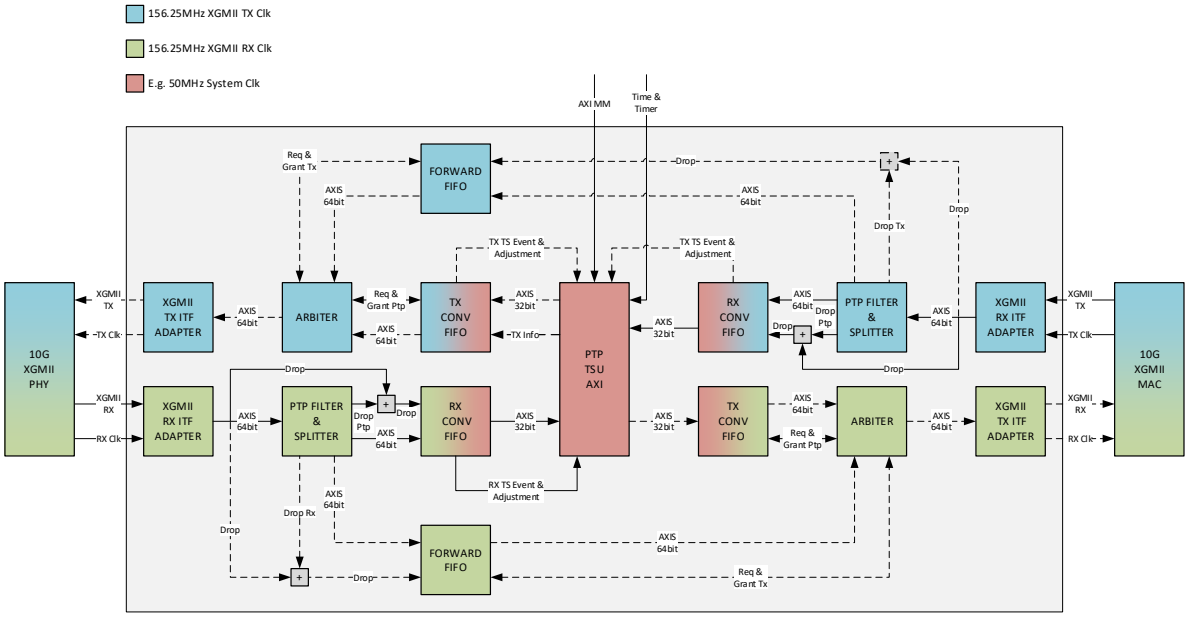

Figure 10: Clocking Concept

2.3.2 Reset

In connection with the clocks, there is a reset signal for each clock domain. All resets are active low. All resets can be asynchronously set and shall be synchronously released with the corresponding clock domain. All resets shall be asserted for the first couple (around 8) clock cycles. All resets shall be set simultaneously and released simultaneously to avoid overflow conditions in the core. See the reference designs top file for an example of how the reset shall be handled.

Reset	Polarity	Description
System		
System Reset	Active low	Asynchronous set, synchronous release with the system clock
XGMII Interface		
XGMII RX Reset	Active low	Asynchronous set, synchronous release with the (R)(G)MII RX clock
XGMII TX Reset	Active low	Asynchronous set, synchronous release with the (R)(G)MII TX clock
AXI Interface		
AXI Reset	Active low	Asynchronous set, synchronous release with the AXI clock, which is the same as the system clock

Table 14: Resets

A List of tables

B List of figures