

1Gb Ethernet Module Description

	Organisatie / Organization	Datum / Date
Auteur(s) / Author(s): Eric Kooistra	ASTRON	
Controle / Checked: Andre Gunst	ASTRON	
Goedkeuring / Approval: Andre Gunst	ASTRON	
Autorisatie / Authorisation: Handtekening / Signature	ASTRON	

© ASTRON 2010
 All rights are reserved. Reproduction in whole or in part is
 prohibited without written consent of the copyright owner.

Distribution list:

Group:	Others:
Andre Gunst (AG, ASTRON) Eric Kooistra (EK, ASTRON) Daniel van der Schuur (DS, ASTRON) Arpad Szomoru (AS, JIVE) Harro Verkouter (HV, JIVE) Jonathan Hargreaves (JH, JIVE) Salvatore Pirruccio (SP, JIVE)	Gijs Schoonderbeek (GS, ASTRON) Sjouke Zwier (SZ, ASTRON)

Document history:

Revision	Date	Author	Modification / Change
0.1	2010-07-14	Eric Kooistra	Draft in progress.
0.2	2010-09-10	Eric Kooistra	Draft in progress.
0.3	2010-09-22	Eric Kooistra	Added SW view of HW control state machine.
0.4	2010-09-28	Eric Kooistra	Described Tx IP header checksum support. Added section on ETH module memory usage.
0.5	2010-10-01	Eric Kooistra	Use continue reg wr access instead on control reg wr.
0.6	2010-11-03	Eric Kooistra	Separate interface section into SW section and HW section. Force DST MAC to this node MAC for Tx response header. Changed Rx, Tx packet register to big endian. Made VHDL implementation more basic, e.g. showing more clearly how eth_hdr is reused.
0.7	2010-12-01	Eric Kooistra	Reordered sections to avoid forward references. Added sections on software functions and module generics.
1.0	2010-02-02	Eric Kooistra	Updated after review on Dec 8, 2010 with EK, DS, JH, SP. Corrected PCS control by keeping auto negotiate enabled.

Table of contents:

1	Introduction.....	7
1.1	Purpose.....	7
1.2	Module overview	7
1.2.1	Compatibility with 10GbE.....	8
2	Software interface	9
2.1	Node control operation.....	9
2.1.1	Multiple hosts.....	9
2.1.2	Frame support.....	9
2.2	HW – SW interaction.....	9
2.2.1	MM ETH interrupt	11
2.2.2	Status polling	11
2.2.3	Endianess	11
2.3	MM ETH registers	11
2.3.1	Demux.....	11
2.3.2	Config.....	12
2.3.3	Control	12
2.3.4	Frame.....	13
2.3.5	Status.....	13
2.3.6	Continue.....	14
2.4	MM ETH packet buffer	14
2.4.1	Packet length	14
2.4.2	Response Tx header support	15
2.5	MM TSE IP registers and settings	15
2.6	Software functions.....	15
3	Hardware interface	16
3.1	Clock domains.....	16
3.2	Parameters.....	16
3.3	MM interface	16
3.4	ST interface	17
3.5	PHY interface	17
4	Application.....	18
4.1	SOPC design	18
4.2	Synthesis.....	19
4.3	Software main	19
4.4	Known issues	19
5	Design	20
5.1	Architecture	20
5.2	Packet buffering	20
6	Implementation	22
6.1	TSE IP wrapper	22
6.2	Header handling.....	22
6.3	CRC handling.....	23
6.4	UDP off-load.....	23
6.5	Rx buffer.....	23
6.6	Control.....	24
6.7	MM registers	24
6.8	MM packet buffer	25
7	Verification.....	26
7.1	Simulation.....	26

7.1.1	Test bench for TSE IP	26
7.1.2	Test bench for ETH module.....	27
7.1.3	Test benches for a UniBoard SOPC design with ETH module	27
7.2	Target hardware	28
8	Appendix: Ethernet protocols	29
8.1	Ethernet frame	29
8.2	IPv4 header	29
8.3	ARP header.....	29
8.4	ICMP header	30
8.5	UDP header	30
9	Appendix: TSE IP	31
9.1	Generics	31
9.2	PCS control registers	31
9.3	MAC control registers.....	32
9.3.1	COMMAND_CONFIG register bits	32
9.3.2	FIFO control	33
10	Appendix: List of files	34
10.1	Firmware VHDL.....	34
10.1.1	TSE IP	34
10.1.2	ETH module	34
10.1.3	UNB_TSE design.....	35
10.2	Software C, H.....	35
10.2.1	Module	35
10.2.2	Main	35
10.3	Simulation.....	36
10.4	Synthesis.....	36

References:

1. www.altera.com, "Triple Speed Ethernet MegaCore, Function User Guide", ug_ethernet.pdf
2. www.altera.com, "Avalon Interface Specifications", mnl_avalon_spec.pdf
3. <http://en.wikipedia.org>
4. https://svn.astron.nl/UniBoard_FP7/UniBoard/trunk, the UniBoard FP7 SVN repository (\$UNB)
5. "Specification for module interfaces using VHDL records", ASTRON-RP-380, Eric Kooistra
6. "DP Streaming Module Description (The ready signal of the streaming interface)", ASTRON-RP-382, Eric Kooistra
7. "Common Library Memory and Register Component Descriptions", ASTRON-RP-415, Eric Kooistra
8. "UNB_Common Module Description", ASTRON-RP-426, Eric Kooistra

Terminology:

ARP	Address Resolution Protocol (to link an IP address to a MAC address)
AVS	Avalon interface Slave (Avalon slave wrapper of a MM slave peripheral)
CRC	Cyclic Redundancy Check
DHCP	Dynamic Host Configuration Protocol (for IP address assignment)
DMA	Direct Memory Access
DP	Data Path
DSP	Digital Signal Processing
DST	Destination
DUT	Device Under Test

eof	End Of Frame
eop	End Of Packet
err	Error signal
FIFO	First In First Out
Firmware	Digital logic
FPGA	Field Programmable Gate Array
GbE	Gigabit Ethernet
GMII	Gigabit MII
GUI	Graphical User Interface
HDL	Hardware Description Language
HW	Hardware
ICMP	Internet Control Message Protocol (for ping)
ISR	Interrupt Service Routine
HDL	Hardware Description Language (typically VHDL or Verilog)
IC	Integrated Circuit
I/O	Input/Output
IP	Internet Protocol, Intellectual Property
ISR	Interrupt Service Routine
LLC	Logical Link Control
LOFAR	Low Frequency Array
LS	Least Significant
LVDS	Low-Voltage Differential Signaling
M9K	Altera RAM block unit of size 1 kByte
MAC	Media Access Control
MDIO	Management Data Input/Output
MI	Media Independent Interface
MISO	Master In Slave Out
MM	Memory Mapped interface (part of Altera Avalon interface)
MMS	MM Slave
MOSI	Master Out Slave In
MS	Most Significant
Nof	Number of
OSI	Open System Interconnection
PCS	Physical Coding Sub-layer
PHY	Physical layer (PMA and PCS)
PMA	Physical Medium Attachment
RO	Read Only
RTL	Register Transfer Level
RW	Read Write
SDC	Synopsys Design Constraint
SHA	Sender Hardware Address
SGMII	Serial GMII
SISO	Source In Sink Out
SPA	Sender Protocol Address
SRC	Source
SFD	Start of Frame Delimiter
sof	Start Of Frame
Software	Software embedded on the Nios II microprocessor or on software on a PC host
sop	Start Of Packet
SOPC	System On a Programmable Chip (Altera firmware system builder tool)
SOSI	Source Out Sink In
ST	Streaming interface (part Altera Avalon interface)
SW	Software
TBD	To Be Done/Decided
THA	Target Hardware Address
TPA	Target Protocol Address
TSE	Triple Speed Ethernet (Altera GbE interface IP)
UDP	User Datagram Protocol

UNB	https://svn.astron.nl/UniBoard_FP7/UniBoard/trunk
VHDL	Very High Speed IC HDL
WDI	Watchdog Interrupt

1 Introduction

1.1 Purpose

This document is a user guide and a description of the 1 Giga bit per second Ethernet (ETH) module [3].

The ETH module connects the 1GbE port of an FPGA node to an on chip microprocessor soft core. The ETH module is intended for the UniBoard using Stratix IV FPGA, Nios II soft core microprocessor and the Triple Speed Ethernet (TSE) MAC IP from Altera. However the module architecture is generic so it can easily be adapted to fit other Ethernet MAC IP.

Section 2 describes the software interface of the ETH module and contains sufficient information for on chip microprocessor software development. The subsequent sections describe the hardware interface, application, design, implementation and verification of the ETH module and its internal details.

1.2 Module overview

Figure 1 shows the interfaces of the ETH module. On the line side the ETH module uses the TSE IP from Altera [1] to receive and transmit Ethernet packets to and from the PHY interface. On the application side the ETH module provides a memory mapped (MM) slave interface to receive and transmit control packets and an UDP streaming (ST) interface for direct data path access. The MM interface also serves to set up the TSE IP registers and the registers of the ETH module. On the UniBoard the MM master is typically the Nios II soft core microprocessor.

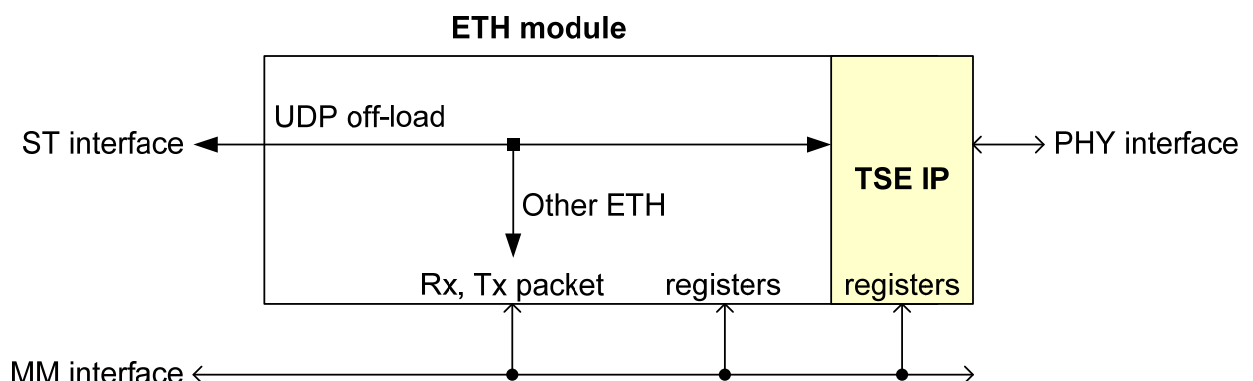


Figure 1: ETH module interfaces

The purpose of the ETH module is to present the received frames to the microprocessor and to transmit frames when enabled to do so by the microprocessor. Typically the main task of a soft core microprocessor in a UniBoard FPGA node is to handle Ethernet packets from the control computer. The real time processing performance of a soft core microprocessor is limited though. Therefore the ETH module offers several features to minimize the processing load for the microprocessor:

- Off-load UDP packets for direct data path access
- Buffer the other ETH packets via an input FIFO while the microprocessor is busy
- Provide status information on the received packet
- Prepare the header of the response packet
- Calculate the IP header checksum for received and transmitted IP packets

1.2.1 Compatibility with 10GbE

The architecture of this 1GbE ETH module also suits 10GbE, because the Ethernet protocol is independent of the speed. However there is at least one issue, because the 1Gb ETH module internally works with 32 bit words @ 125 MHz whereas the 10GbE MAC uses 64 bit long words @ 156.25 MHz. Furthermore for 10GbE the UDP off-load port would be the main usage. The micro processor interface would only need to be used to handle ping and ARP, DHCP to set up the UDP/IP link.

2 Software interface

2.1 Node control operation

All received packets that are not for the streaming UDP off-load ports will be presented to the microprocessor via the Rx packet register. The control computer communicates with a node via UDP/IP using IPv4. The control computer sends a request and then the node sends a response. The microprocessor in the node interprets the packet header and handles the packet payload. For light control tasks like reporting status on temperature and voltages the payload can be handled entirely in software. For heavy control tasks like setting a block of data path coefficients or reporting data path statistics the software typically sets up a DMA transfer between the packet register and the addressed data path component. The DMA component is not part of the ETH module, but could easily be added in an SOPC builder design. The microprocessor then instructs the DMA component to do the data transfer between the addressed data component and the Rx or Tx packet payload. Alternatively for streaming data path IO the ETH module can be set up with one or more dedicated UDP ports to off-load these packets completely from the microprocessor.

2.1.1 Multiple hosts

With the default Rx-Tx, request-response operation the control computer will typically not send the next request packet until it has received a response from the node. However the node must have sufficient packet buffer memory to cope with multiple Rx frames because it is connected to a network and because it can have multiple PC hosts. Via the network broadcast messages will also appear at the node. Using more than one host can be useful to have one PC host to control the node and use another PC host to monitor some node status.

2.1.2 Frame support

The ETH module firmware provides extra support for ARP, IP, ICMP, UDP and DHCP frames, see Appendix 8 for their packet structure [3]. ARP is used for informing routers about what MAC address belongs to what IP address. IP is the Internet Protocol for communicating packets across a network. ICMP/IP is used for ping. UDP/IP is used for node control and for data path UDP off-load ports. DHCP/UDP/IP is used for dynamic IP address assignment.

The ETH module prepares the header of the response Tx frame based on the header of the Rx frame. The Tx frame payload is undefined. The last header details and the payload of the Tx frame need to be filled in by software on the microprocessor. For IP the ETH module calculates the IP header checksum.

2.2 HW – SW interaction

Figure 2 shows the control state machine of the ETH module from a software point of view. The state machine is a Mealy-type state machine. For example, the state machine goes from state RECEIVE to state PENDING when done is active (i.e. NOT busy) and then it sets RX_AVAIL to 1 and issues the IRQ. Else when busy is active, then the state machine remains in state RECEIVE. A reset causes the state machine to start in state IDLE.

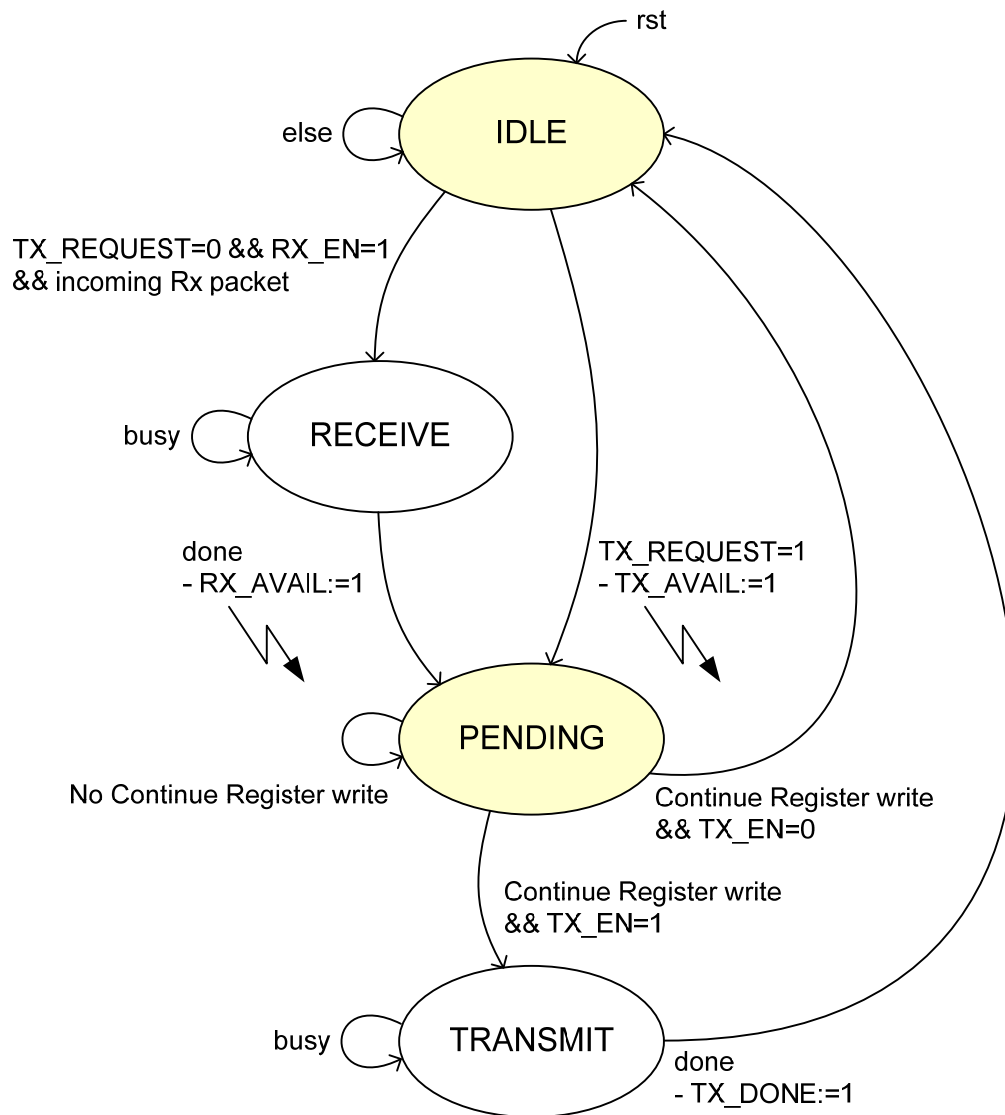


Figure 2: Software view of the ETH control state machine

The ETH module has two modes:

1. Rx-Tx operation
2. Tx insert

The ETH module does not start a transmission by itself. Therefore it interrupts the microprocessor when a new frame is available in the Rx buffer or to acknowledge a transmit insert request. There can only be one interrupt cause at a time and the SW can find out which by reading the status register. To let the HW continue the SW needs to write the continue register. Typically the HW will then transmit the frame, but it is also allowed to continue without transmit.

It is allowed to operate the ETH module in separate Rx and Tx operation, meaning that the software can receive multiple packets before it sends their response packets. This requires that the Rx packet buffer is copied to processor memory and implies that the Tx frame header response support can not be used directly. A frame transmit is then always issued via a transmit insert request.

2.2.1 MM ETH interrupt

The ETH module issues an interrupt when RX_AVAIL or TX_AVAIL in the status register is '1'. The ETH module removes the interrupt when the microprocessor does a (dummy) write access to the status register. As a minimum an ETH interrupt service routine (ISR) should read the status register and then write the status register to acknowledge the interrupt request (IRQ). Before returning the ISR must keep on reading the status register until it is zero, to ensure that the ETH module has removed the IRQ.

The ETH module software (Appendix 10.2) provides an ISR that reads the status register and clears the ETH interrupt. The read data is available via an ISR function argument for further processing in a task. In this way the ISR is as short as possible. Thanks to the handshake handling of Figure 2 the SW and HW remain in phase without the need to (temporarily) disable the interrupts.

2.2.2 Status polling

Status register polling instead of using interrupts is not preferred, but it is allowed. Status register polling can also be useful to monitor TX_DONE. When polling is used in combination with interrupts care must be taken that the ETH control state machine remains in a known state, because otherwise the polling loop could cause the SW to hang. This could happen for example when the ISR clears the status register after a new receive, because RX_EN was set too soon, while the task with the polling loop has not yet seen TX_DONE active from the previous transmit.

2.2.3 Endianness

The Nios II architecture is little endian. Words and half words are stored in memory with the more-significant bytes at higher addresses. Bit 31 indicates the MSBit and bit 0 indicates the LSBit of a word.

The ETH module MM interface has to be accessed per word. Access per byte is not supported. The ETH registers are little endian. The TSE IP registers are little endian, except for the MAC_0,1 register which are big endian. The Rx and Tx packet register is big endian to suit the network order.

2.3 MM ETH registers

Table 1 lists the MM registers that are available in the ETH module. The number of demux ports for UDP off-load streams is 2, but can be changed by regenerating the ETH module.

Name	Address (words)	Size (words)	Read/Write	Description
Demux	0	2	RW	UDP demux ports
Config	2 = nof UDP ports + 0	4	RW	ETH module configuration
Control	6 = nof UDP ports + 4	1	RW	Rx control and Tx control
Frame	7 = nof UDP ports + 5	1	R	Rx packet information
Status	8 = nof UDP ports + 6	1	R	Rx status and Tx status
Continue	9 = nof UDP ports + 7	1	W	ETH module continue

Table 1: ETH module registers

2.3.1 Demux

The demux register defines 2 UDP ports for data path off-load streams, see Table 2. The UDP_PORT_# bits are defined in Table 3.

Offset (words)	Bits 31:0
0	UDP_PORT_0
1	UDP_PORT_1

Table 2: Demux register

Bits	Field name	Description
[31:17]	RSVD	-
[16]	UDP_PORT_EN	When '1' then the UDP port is enabled to off-load traffic else not used
[15:0]	UDP_PORT_NR	If a received frame matches an enabled UDP port number then it is passed on to that the ST UDP off-load interface. Otherwise the received frame is kept inside the ETH module and passed on for further analysis.

Table 3: Demux register bits for UDP off-load port control

2.3.2 Config

The config register defines the Ethernet MAC and IPv4 addresses of this node and the UDP port number for node control, see Table 4.

Offset (words)	Bits 31:0
2	MAC_ADDRESS_LO
3	MAC_ADDRESS_HI
4	IP_ADDRESS
5	UDP_CTRL_PORT

Table 4: Config register

The MAC_ADDRESS_LO and MAC_ADDRESS_HI fields are defined in Table 5.

MAC word	Node MAC address bits
MAC_ADDRESS_LO[31:0]	MAC_ADDRESS[31:0]
MAC_ADDRESS_HI[15:0]	MAC_ADDRESS[47:32]
MAC_ADDRESS_HI[31:16]	RSVD

Table 5: MAC address definition

For example MAC_ADDRESS [47:0] = 0x123456789ABC corresponds to MAC address 12-34-56-78-9A-BC.
For example IP_ADDRESS[31:0] = 0x05060708 corresponds to IP address 5:6:7:8.

The UDP_CTRL_PORT bits are defined in Table 6.

Bits	Field name	Description
[31:16]	RSVD	-
[15:0]	UDP_CTRL_PORT	Defines the UDP port number that is used for node control. Only used to report IS_UDP_CTRL_PORT in the Frame register (see Table 9)

Table 6: UDP control port

2.3.3 Control

The control register provides the microprocessor means to control the access to the Rx and Tx buffer, see Table 7. The CONTROL bits are defined in Table 8.

Offset (words)	Bits 31:0
6	CONTROL

Table 7: Control register

Bits	Field name	Description
[31:30]	RSVD	-
[29:18]	TX_NOF_WORDS	Number of words in the Tx frame, including the word align field (see Figure 16) and excluding the CRC.
[17:16]	TX_EMPTY	Number of LS octets in the last word of the Tx frame that are not valid.
[15:3]	RSVD	-
[2]	TX_REQUEST	Request to insert an extra Tx frame.
[1]	TX_EN	Enable the ETH Tx to send out frames from the Tx frame buffer.
[0]	RX_EN	Enable the ETH Rx to pass on frames to the Rx frame buffer.

Table 8: Control register bits

2.3.4 Frame

The frame register provides information on the received frame in the Rx buffer, see Table 9. The FRAME bits are defined in Table 10.

Offset (words)	Bits 31:0
7	FRAME

Table 9: Frame register

Bits	Field name	Description
[31:15]	RSVD	-
[14]	IS_UDP_CTRL_PORT	When '1' then IS_UDP='1' and the UDP port number matches the UDP_CTRL_PORT
[13]	IS_UDP	When '1' then IS_IP='1' and the IPv4 protocol type indicates UDP
[12]	IS_ICMP	When '1' then IS_IP='1' and the IPv4 protocol type indicates ICMP
[11]	IP_ADDRESS_MATCH	When '1' then IS_IP='1' and the IPv4 address matches this node IP_ADDRESS
[10]	IP_CHECKSUM_OK	When '1' then IS_IP='1' and the IPv4 header checksum is OK
[9]	IS_IP	When '1' then the Ethernet type indicates IPv4
[8]	IS_ARP	When '1' then the Ethernet type indicates ARP
[7]	MAC_ADDRESS_MATCH	When '1' then the DST_MAC address matches this node MAC_ADDRESS
[6]	RSVD	-
[5:0]	ETH_MAC_ERROR	When '0' then OK, else TSE IP error indication: [5] = collision error (can only occur in half duplex mode) [4] = PHY error on GMII [3] = receive frame truncated due to FIFO overflow [2] = CRC-32 error [1] = invalid length [0] = OR of [1:5]

Table 10: Frame register bits

2.3.5 Status

The status register provides the microprocessor status information on the Rx and Tx buffer, see Table 11. The STATUS bits are defined in Table 12.

Offset (words)	Bits 31:0
8	STATUS

Table 11: Status register

Bits	Field name	Description
[31:30]	RSVD	-
[29:18]	RX_NOF_WORDS	Number of words in the Rx frame, including the word align field (see Figure 16) and including the CRC (because CRC_FWD=1, see Table 26).
[17:16]	RX_EMPTY	Number of octets in the last word of the Rx frame that are not valid.
[15:3]	RSVD	-
[2]	TX_AVAIL	When '1' then the Tx buffer is available for inserting a Tx frame.
[1]	TX_DONE	When '1' then the Tx frame in the Tx buffer has been send.
[0]	RX_AVAIL	When '1' then there is a new Rx frame available in the Rx buffer.

Table 12: Status register bits

2.3.6 Continue

A write access to the continue register triggers the ETH module to act upon the new control data. This will cause the state machine in Figure 2 to leave the PENDING state. The continue register has no contents, see Table 13.

Offset (words)	Bits 31:0
9	void

Table 13: Continue register

Note: It is not possible to use a control register write access to make the ETH module continue, because then a control register write access to set the TX_REQUEST bit can cause the ETH module to leave the PENDING state before the software tasks loop has taken care of it. Using a control register read access is also not suitable, because a control register read access can be necessary in a read-modify-write sequence. Therefore if a register like the control register has multiple purposes, then it is necessary to define a separate event register like the continue register.

2.4 MM ETH packet buffer

Table 14 shows the MM Rx packet buffer and the MM Tx packet buffer that are available in the ETH module.

Name	Address (words)	Size (words)	Read/Write	Description
Rx packet	0	0x800	RW	Rx packet buffer 2 kBytes to fit a 1520 octet frame
Tx packet	0x800	0x800	RW	Tx packet buffer 2 kBytes to fit a 1520 octet frame

Table 14: ETH module Rx and Tx packet buffers

Rx and Tx frames shorter than 11 words are not supported and get discarded.

For an ETH/IP Tx packet the ETH module will calculate and overwrite the IP header checksum field. The ETH module does not calculate and overwrite the UDP checksum.

2.4.1 Packet length

The packet length includes the word align field (2), DST_MAC (6), SRC_MAC (6), Ethernet Type (2), payload (max. 1500, or max. 9000 for jumbo frames) and CRC (4), so in total 1520 or 9020 bytes.

If jumbo frames are supported then the packet buffer sizes become 8 kByte or 9 kByte and the Tx packet base address becomes 8*1024 or 9*1024 dependent on the maximum jumbo frame size compile parameter setting in the ETH module VHDL (see section 5.2).

To use jumbo frames requires regeneration of the ETH module logic. The TX_NOF_WORDS field in the control register and the RX_NOF_WORDS field in the status register are 14 bit, so they already suit jumbo frame sizes. For more details on the Rx and Tx packet buffer sizes see section 5.2.

2.4.2 Response Tx header support

When a new frame is available in the Rx packet buffer, then the ETH module has also prepared the header for the response packet in the Tx packet buffer. The Tx header is a copy of the Rx header except for the modifications listed in Table 15.

Protocol	Reference	Modification
ETH	Figure 16	Use SRC MAC for DST MAC address. Force SRC MAC to this node MAC address as set in config register (Table 4).
ETH/ARP	Figure 18	Force the operation field to ARP reply = 2. Force SHA field to this node MAC address as set in config register (Table 4). Force SPA field to this node IP address as set in config register (Table 4). Use Rx SHA for THA. Use Rx SPA for TPA.
ETH/IP	Figure 17	Force IP header checksum field to 0. Swap IP DST address and IP SRC address.
ETH/IP/ICMP	Figure 19	Force type field to ICMP reply = 0. Force ICMP checksum field to 0.
ETH/IP/UDP	Figure 20	Swap UDP DST port and UDP SRC port. Force UDP checksum field to 0.

Table 15: Tx response header modifications

2.5 MM TSE IP registers and settings

The TSE IP has been generated without statistic counts, no supplementary MACs and no multi cast hash table. Appendix 9.1 lists the parameter settings of the TSE IP.

The PCS and MAC registers in the TSE IP need to be written to operate the TSE IP for the ETH module in a UniBoard node. Appendix 9.2 and 9.3 lists the programmable settings for the TSE IP. The ETH registers should have been set before the TSE IP Rx and Tx are enabled.

2.6 Software functions

The ETH module software consists of avs_eth.c/h and avs_eth_regs.h and provides public variables and public functions and macros to control the ETH module peripheral. Table 16 lists the software functions that are available.

Function	Modification
ETH_Init()	Init the ETH device peripheral by hooking a default or a custom ISR.
ETH_Setup	Setup the ETH module with a MAC address and other IP and UDP properties. It also calls the private TSE_Setup() function which takes care of the TSE IP setting conform appendix 9.

Table 16: Public ETH functions in the avs_eth.c/h software module

3 Hardware interface

3.1 Clock domains

Figure 3 shows the three clock domains that are used by the ETH module:

- eth_clk = PHY 125 MHz reference clock for the TSE IP
- mm_clk = MM clock for the memory-mapped bus with the NIOS II processor
- st_clk = ST clock for the streaming UDP off-load interface

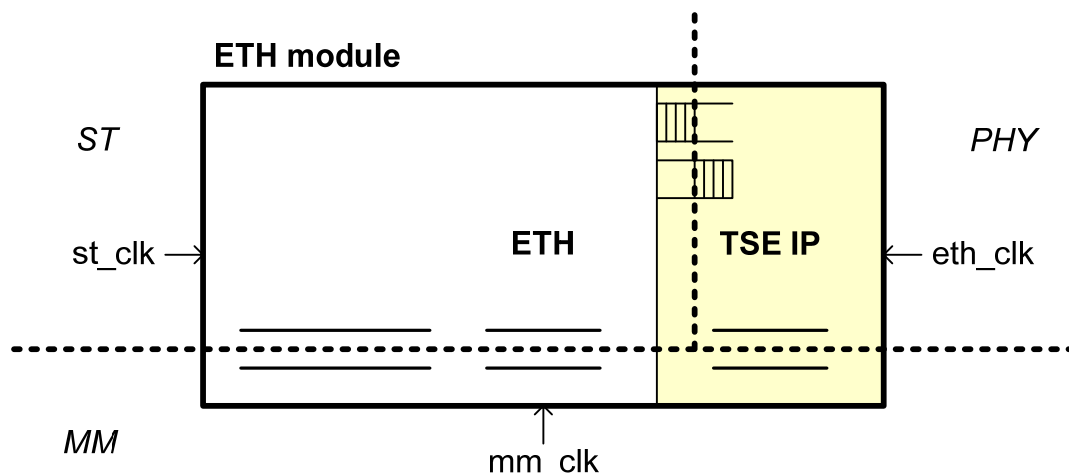


Figure 3: Clock domains of the ETH module

3.2 Parameters

Table 17 lists the VHDL generics that can be changed when the ETH module is instantiated in another HDL design or in an SOPC system.

Generic	Type	Description
g_cross_clock_domain	Boolean	Default TRUE. Use FALSE, when the mm_clk and st_clk (see Figure 3) are the same, else use TRUE, to ensure that the MM register information reliably crosses the clock domain.

Table 17: ETH module parameters

3.3 MM interface

Table 18 defines the interface signals for the MM ETH registers.

Signal	Type	Description
rddata[31:0]	MISO	Read data word, valid 1 clock cycle after rd
address[3:0]	MOSI	Word address range to fit the ETH registers of Table 1
wrdata[31:0]	MOSI	Write data word, must be valid with wr
wr	MOSI	Write strobe
rd	MOSI	Read strobe

Table 18: MM ETH registers interface signals

Table 19 defines the interface signals for the MM ETH packet register.

Signal	Type	Description
rddata[31:0]	MISO	Read data word, valid 2 clock cycles after rd.
address[11:0]	MOSI	Word address range to fit an Rx packet and a Tx packet.
wrdata[31:0]	MOSI	Write data word, must be valid with wr.
Wr	MOSI	Write strobe.
Rd	MOSI	Read strobe.

Table 19: MM ETH packet register interface signals

For the MM interface definition of the MM TSE IP registers see [1].

3.4 ST interface

Table 20 shows the UDP off-load interface ST signals.

Signal	Type	Description
ready	SISO	Backpressure flow control signal. The ready latency is RL = 1.
data[31:0]	SOSI	Data word, byte [31:24] is the MSByte and is transmitted or received first.
empty[1:0]	SOSI	Indicates the number of invalid bytes in the last data word marked by eop.
valid	SOSI	Data valid strobe.
sop	SOSI	Start of packet strobe.
eop	SOSI	End of packet strobe.
channel	SOSI	Support two UDP off-load channels 0 or 1.

Table 20: MM ETH packet register interface signals

3.5 PHY interface

The TSE IP is configured for 1000BASE-X via two 1.25 Gbps LVDS lines, one for Tx and one for Rx. In an SOPC system these IO signals appear as conduit interface signals.

4 Application

4.1 SOPC design

Thanks to an AVS wrapper component and a hardware description TCL file the ETH module is also available within SOPC builder, see Appendix 10.1. Figure 4 shows the ETH module called `avs_eth` in the `sopc_tse.sopc` SOPC system. The `avs_eth` does not (yet) present the UDP off-load ST interface in the GUI.

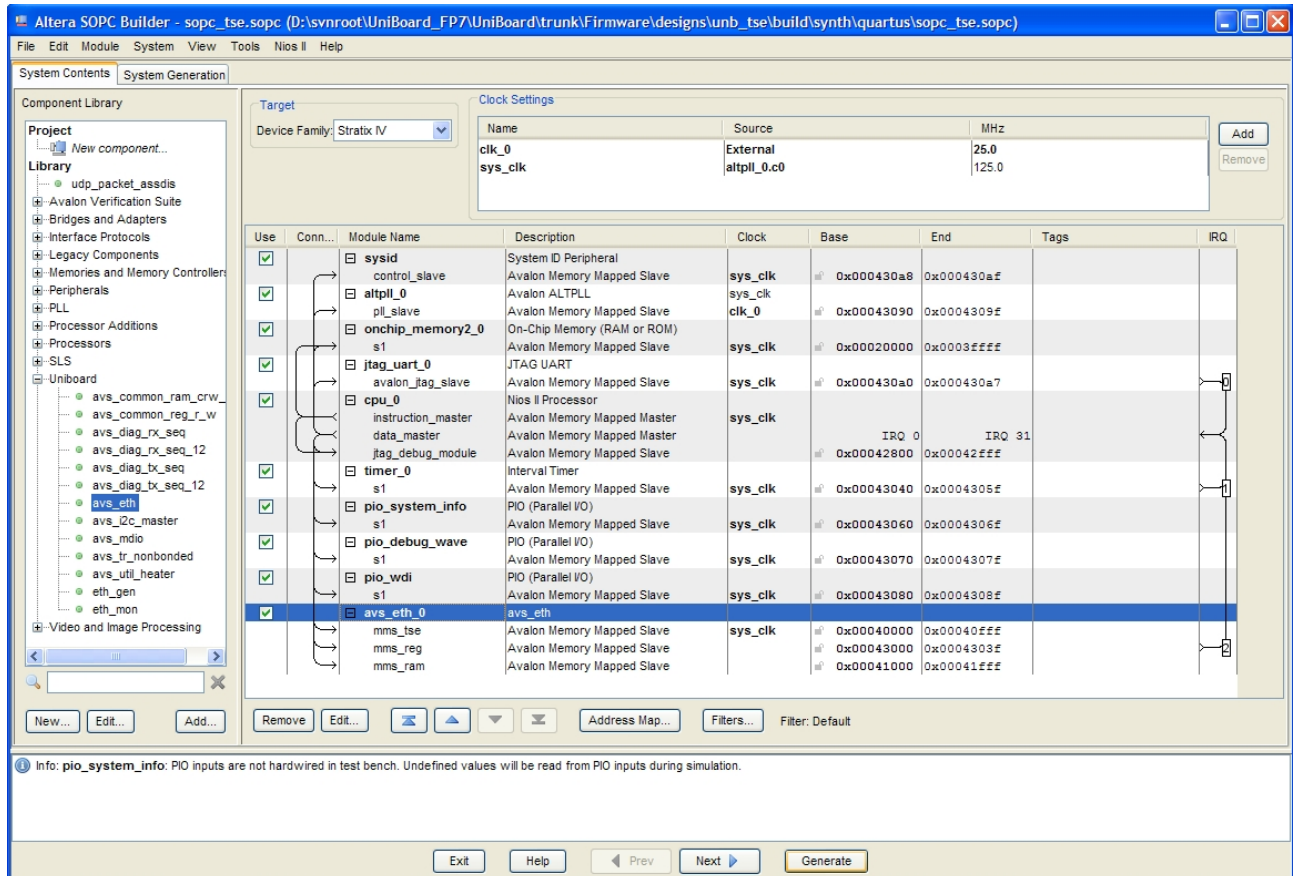


Figure 4: SOPC builder system socp_tse.sopc with the ETH module

The SOPC Builder tool generates the `sopc_tse.vhd` VHDL file from the system defined by `sopc_tse.sopc` in Figure 4. Figure 5 shows the `unb_tse` design that instantiates this `sopc_tse` system. The `unb_tse` design uses the `unb_node_ctrl` and `unb_system_info` components from the `unb_common` library [8] to support the `sopc_tse` system in a design that can run on any of the 8 nodes of the UniBoard.

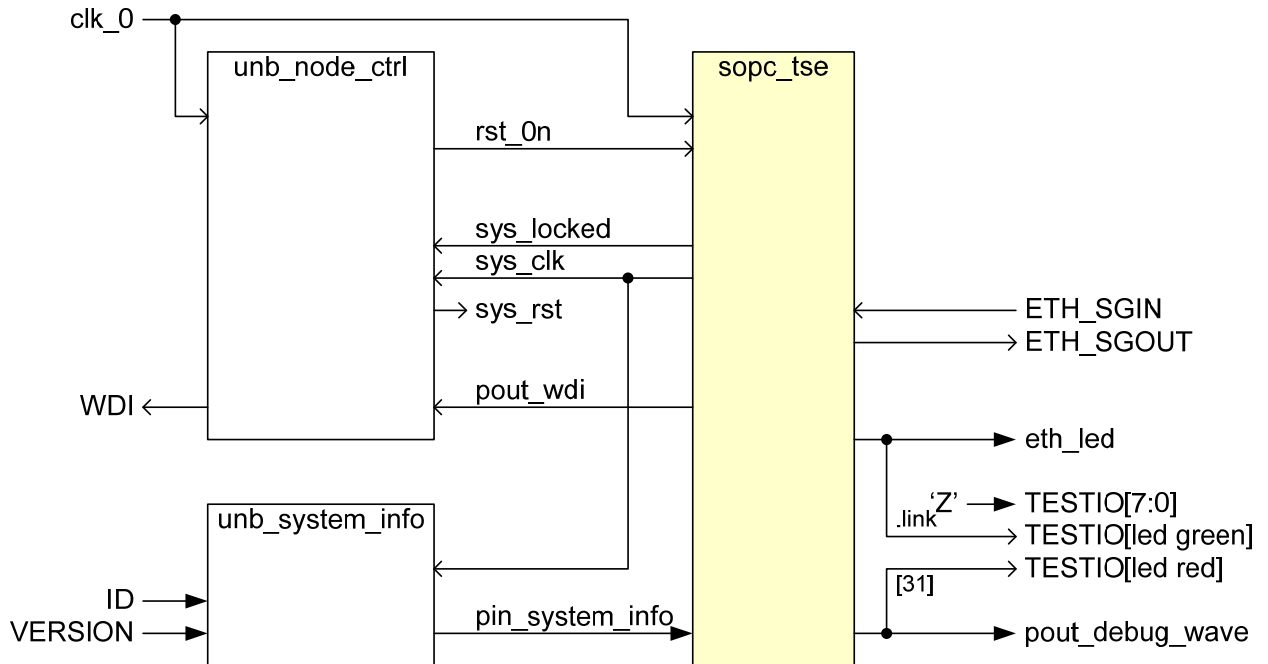


Figure 5: Block diagram of the unb_tse design on an UniBoard FPGA

The IO signals of unb_tse design in Figure 5 are connected to pins of the FPGA or they are left open for monitoring in simulation only. The clk_0 connects to the 25 MHz input ETH_CLK pin of each UniBoard FPGA node and is used as reference for the PLL in the sopc_tse system. The sys_clk is 125 MHz and in this unb_tse example designs it is used for the mm_clk, eth_clk and for the st_clk in Figure 3.

4.2 Synthesis

The unb_tse design can run on all 8 FPGA nodes of the UniBoard, see Appendix 10.4.

4.3 Software main

The ETH applications software contains some main() example functions on how to use them, see Appendix 10.2. The software determines how the unb_tse design behaves. It is not necessary to synthesize the logic again when the software has changed.

4.4 Known issues

The readme.txt in \$UNB/Firmware/MegaWizard//tse_sgmii_lvds/doc contains a list of known issues regarding the ETH module.

5 Design

5.1 Architecture

Figure 6 shows the block diagram of the ETH module. The packet transfer is done via ST interface components. The ETH control component handles the interface between the ST components and the MM packet register. The MM packet register provides the access for a microprocessor to the Rx packet and Tx packet. Via the ETH and TSE IP registers a microprocessor can control and monitor the ETH module.

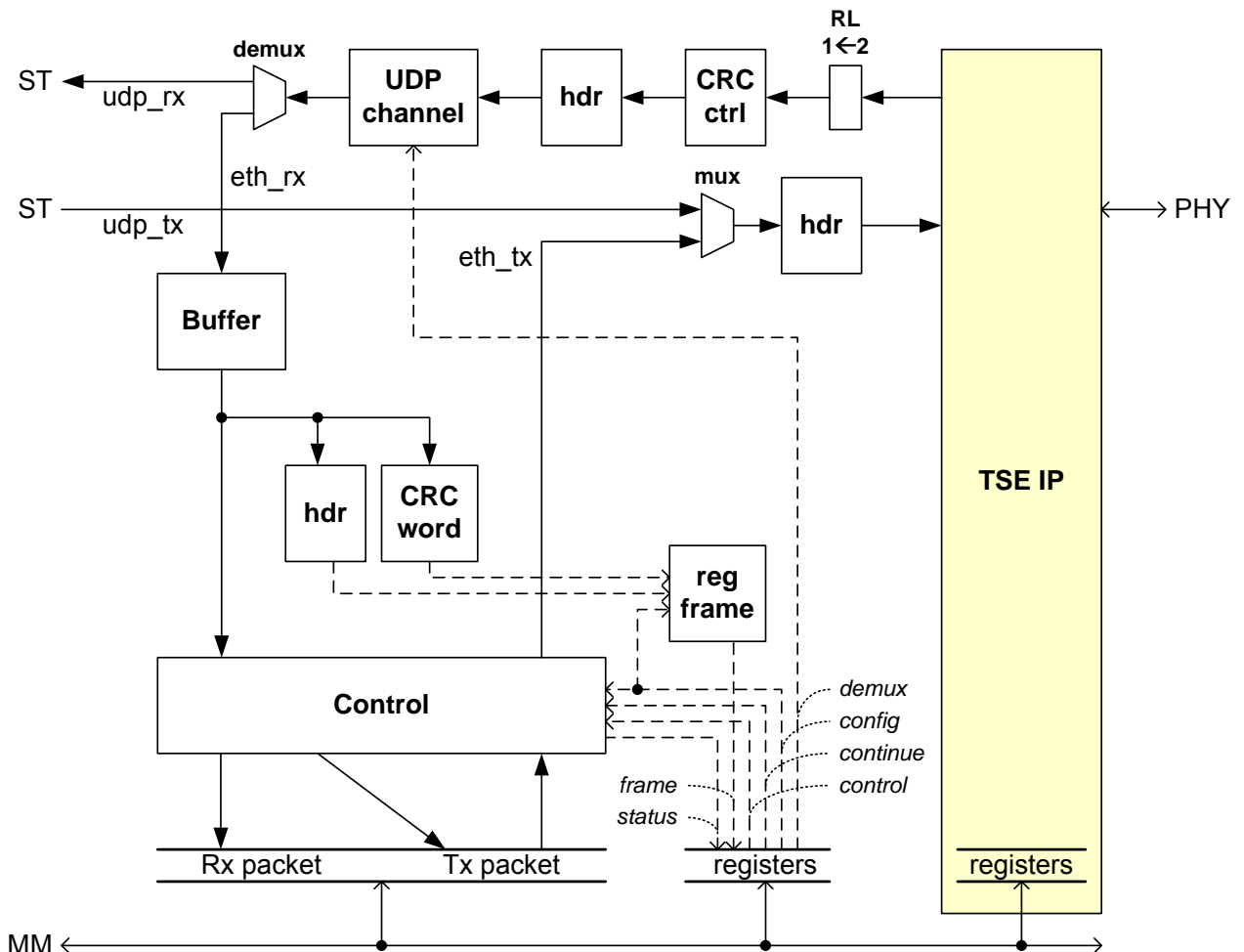


Figure 6: Top level block diagram of the ETH module

5.2 Packet buffering

The TSE IP contains Rx and Tx FIFOs to allow the data to reliably cross the ETH clock domain and the ST clock domain. These FIFOs are 256 x 32b-words deep, because that just fits in 1 M9K RAM block = 1 kByte, so using less words does not save memory resources (note that the maximum data width for a M9K RAM block is 36 bit).

The Rx buffer contains a FIFO to buffer received frames that can not yet be handled by the software. If the Rx packet register is available then a received frame gets passed on with minimal latency otherwise a new frame gets buffered. Dependent on the expected throughput for this node the FIFO buffer typically needs to

be able to store at least one maximum size frame. It may be a requirement to have FIFO depth that is a power of 2, because the Altera MegaWizard does not show intermediate sizes. Therefore use a depth of 2 (or even 4) kByte rather than 1.5 kByte for default Ethernet frames and a depth of 16 (or even 32) kByte rather than 9 kByte if jumbo frames need to be supported.

The Rx packet register and the Tx Packet register both have to be able to store one maximum size frame. These MM memories do not have to have a size that is a power of 2. The Rx and Tx packet are stored in a single buffer, so it is possible to choose 1.5 kByte bytes for default Ethernet which results in $2 \times 1.5 = 3$ M9K RAM blocks. However it is fine to spend one more M9K and use 2 kBytes for the Rx and Tx packet, so 4 M9K RAM blocks (the reason is that this avoids the simulation warnings for addresses that are outside the available RAM range which occur also when the MM slave is not selected). For jumbo frames one can use 9 kByte, so $2 \times 9 = 18$ M9K RAM blocks.

The other functions in the ETH module have no frame buffering. Table 21 summarizes the memory usage of the ETH module. Whether jumbo frames are supported depends on a compile constant in eth(pkg).vhd. Use c_eth_frame_sz=2 kByte for default Ethernet frame size and use c_eth_frame_sz=9 kByte for jumbo frame support.

Function	Buffer	Nof kBytes = nof M9K RAM
TSE IP	Rx FIFO	1
	Tx FIFO	1
Rx buffer	Buffer FIFO	2 (16)
Rx packet	MM register	2 (9)
Tx packet	MM register	2 (9)
Total	ETH module	8 (36 kByte)

Table 21: ETH module memory usage overview (values for jumbo frames in brackets)

6 Implementation

This chapter describes the HDL implementation of the ETH module. The sections follow the functions in the top level block diagram of Figure 6. The ST mux, demux and RL adapter come from the DP library [6].

6.1 TSE IP wrapper

The ETH module is written in generic VHDL. Hence it can be used on any platform. The TSE IP component is Altera specific, but typically such a component will be available on other platforms as well. Therefore the TSE IP component is wrapped by a generic TSE component to clearly separate it from the rest of the ETH module HDL, see Appendix 10.1.

The Rx output from the TSE IP has ready latency RL = 2 so first a DP latency adapter is used to adapt the RL to 1.

6.2 Header handling

The `eth_hdr` component shown in Figure 7 provides a uniform means for handling Ethernet frame headers and is reused throughout the ETH module.

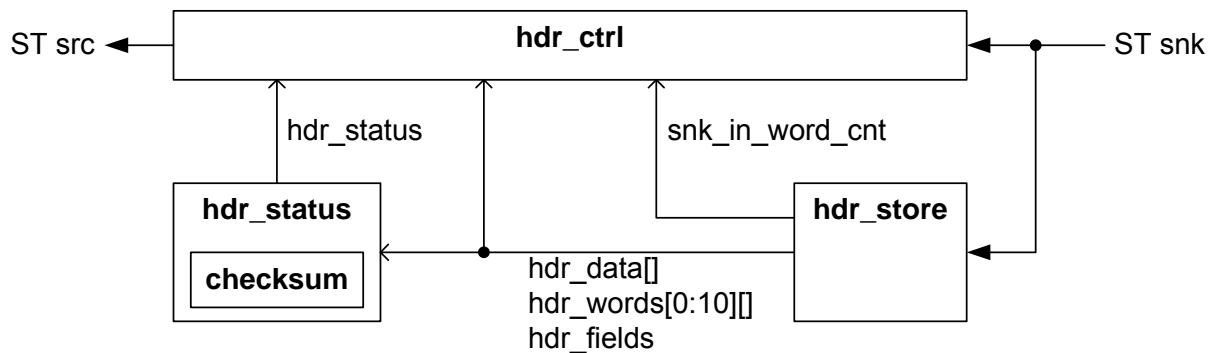


Figure 7: Block diagram for uniform Ethernet header handling

The header of all supported Ethernet protocols (ARP, IP/UDP and IP/ICMP, see Appendix 8) fits into exactly 11 words. Therefore `eth_hdr_store` extracts the first 11 words from an Ethernet packet and makes them available as a word array and as record fields. The component has three output formats:

1. `hdr_data[31:0]`, the currently active header word
2. `hdr_words[0:11][31:0]`, the 11 word header store
3. `hdr_fields`, combinatorial mapping of the 11 word header store to various Ethernet packet header fields

Which format to use depends on the application. The logic for words or header fields that are not used, will get optimized away during synthesis.

The `eth_hdr_status` interprets the header fields and determines the status information from the Ethernet header after every asserted sop. This concerns status information like is it an ARP frame, is it an UDP frame, the UDP port number, etc. The `eth_hdr_status` calculates the IP header checksum using `eth_checksum` or it reads the IP header checksum from the frame.

The `eth_hdr_ctrl` outputs the stored header and the rest of the payload. Frames shorter than 11 words are discarded by `eth_hdr_ctrl`. For IP frames `eth_hdr_ctrl` uses the calculated IP header checksum value to

replace it in the frame. For receive this checksum will be 0 when OK and indicate an IP header error otherwise. For transmit this checksum is the required IP header checksum.

6.3 CRC handling

The `eth_crc_ctrl` component uses DP shiftreg to replace the true CRC frame word by the ST error field information. This avoids the need for the ST error field in subsequent blocks. Hence after the `eth_crc_ctrl` the CRC word in a frame carries an enumerated value, 0=OK and >0 to indicate the TSE IP error code (same as `ETH_MAC_ERROR` field defined in Table 10).

The `eth_crc_word` component extracts the CRC word from the Ethernet frame tail. The `eth_crc_word` acts as a stream monitor, but it can be in the stream because it connects its sink to its source. Dependent on `snk_in.empty` the CRC word is in the last tail word or straddled in the last two tail words. This component does not distinguish on whether the CRC word is the true CRC word or the enumerated CRC derivative from `eth_crc_ctrl`.

6.4 UDP off-load

If UDP off-load is enabled then `eth_channel` maps the UDP port to the appropriate ST channel value dependent on the MM demux register settings described in section 2.3.1. The DP demultiplexer and DP multiplexer are used to separate and combine the UDP off-load traffic and the other ETH traffic.

6.5 Rx buffer

Figure 8 shows a block diagram of `eth_buffer` that is used for the Rx buffer. All frames that are not for UDP off-load are stored by the input FIFO in `eth_buffer`. Via some source control signals a frame can be requested from the FIFO. These source control signals act like the SISO ready signal, but instead of operating per data word they operate per entire frame. The input FIFO needs to be able to hold at least one frame, to allow the microprocessor to handle the current Rx frame. If the input FIFO gets almost full, then the first frame in the FIFO will be flushed to maintain integrity of the frames in the FIFO.

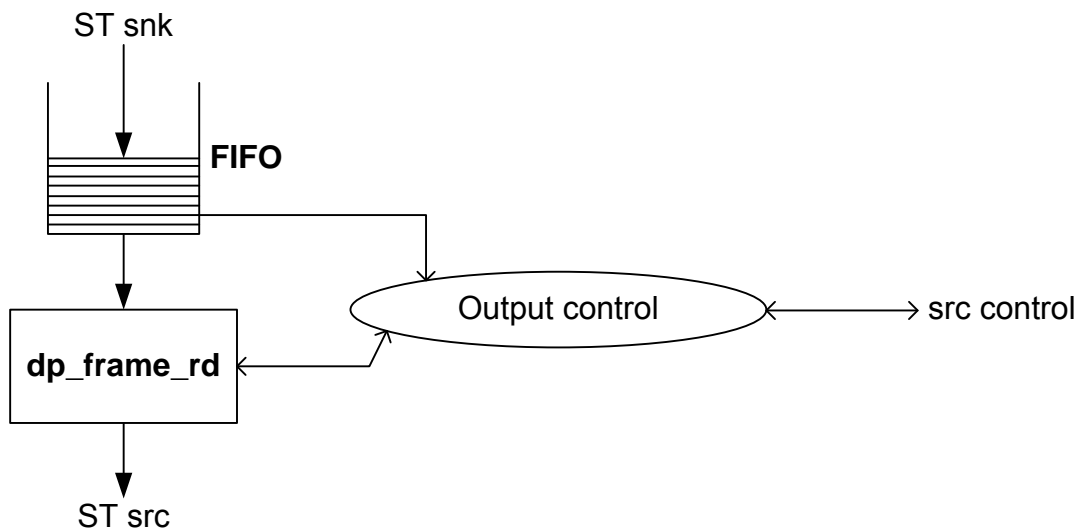


Figure 8: Block diagram for `eth_buffer`

When the frame is output then the status of this frame is presented in the frame register (see section 2.3.4), via `eth_hdr` and `eth_mm_reg_frame` as shown in Figure 6.

6.6 Control

Figure 9 shows a block diagram of ETH control. The ovals indicate combinatorial or clocked processes in VHDL, and are typically named with prefix 'p_'. The ETH control takes care of access to the Rx packet and Tx packet buffer and handles the handshake control with the software conform the state machine of Figure 2. The ETH control can request a frame from the Rx buffer via the receive control signals and it can transmit a frame when enabled to do so by the microprocessor.

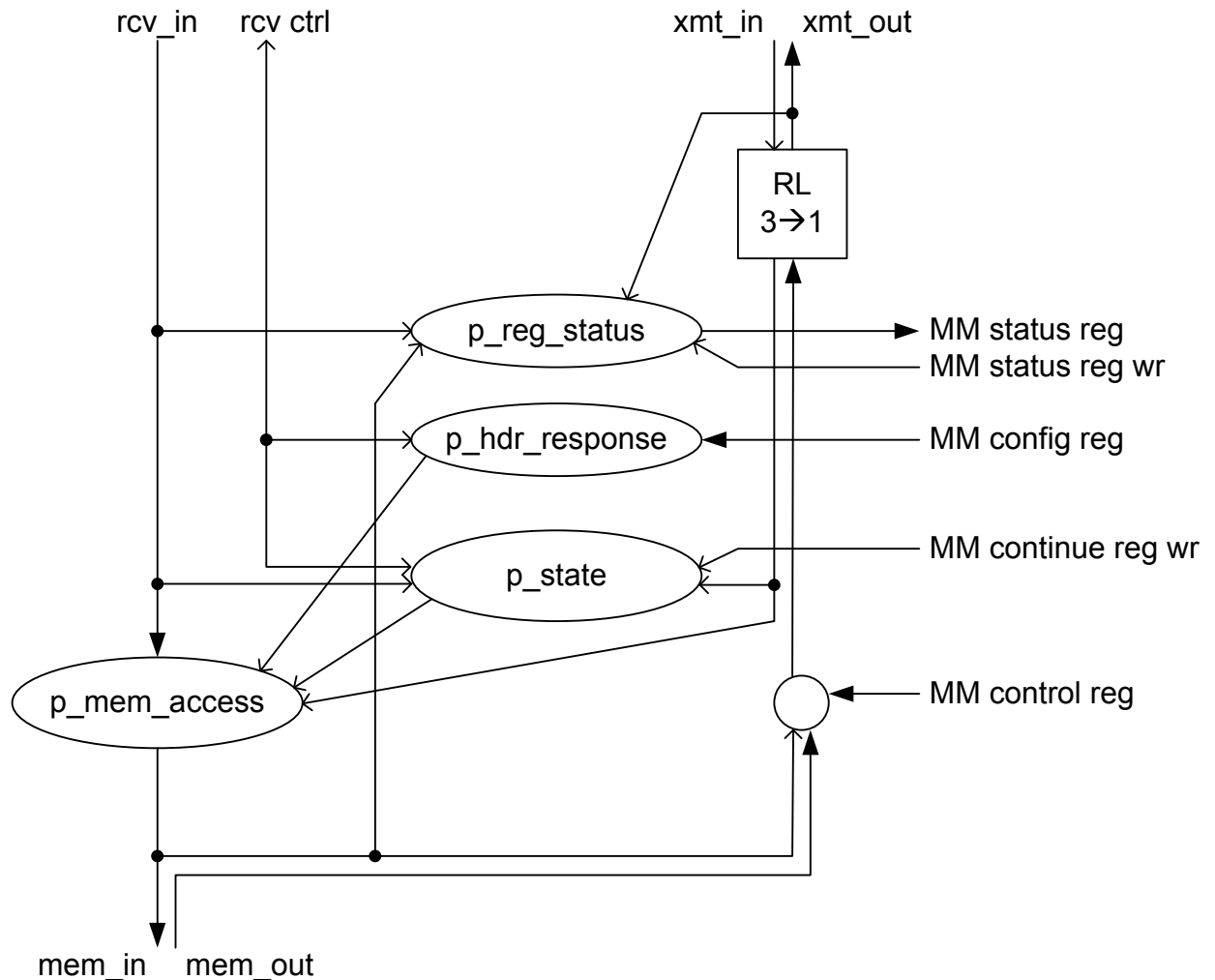


Figure 9: Block diagram for `eth_control`

6.7 MM registers

Figure 10 shows the block diagram for memory-mapped (MM) registers of the ETH module. The MM registers are defined in section 2.3. The MM registers of the TSE IP are not shown, because they have a dedicated MM slave interface. If the MM clock domain differs from the ST clock domain then the clock domain crossing logic has to be inserted via the generic `g_cross_clock_domain` [7]. The representation of the status register in the MM clock domain provides the `RX_AVAIL` and `TX_AVAIL` fields that are used to create the ETH interrupt described in section 2.2.1.

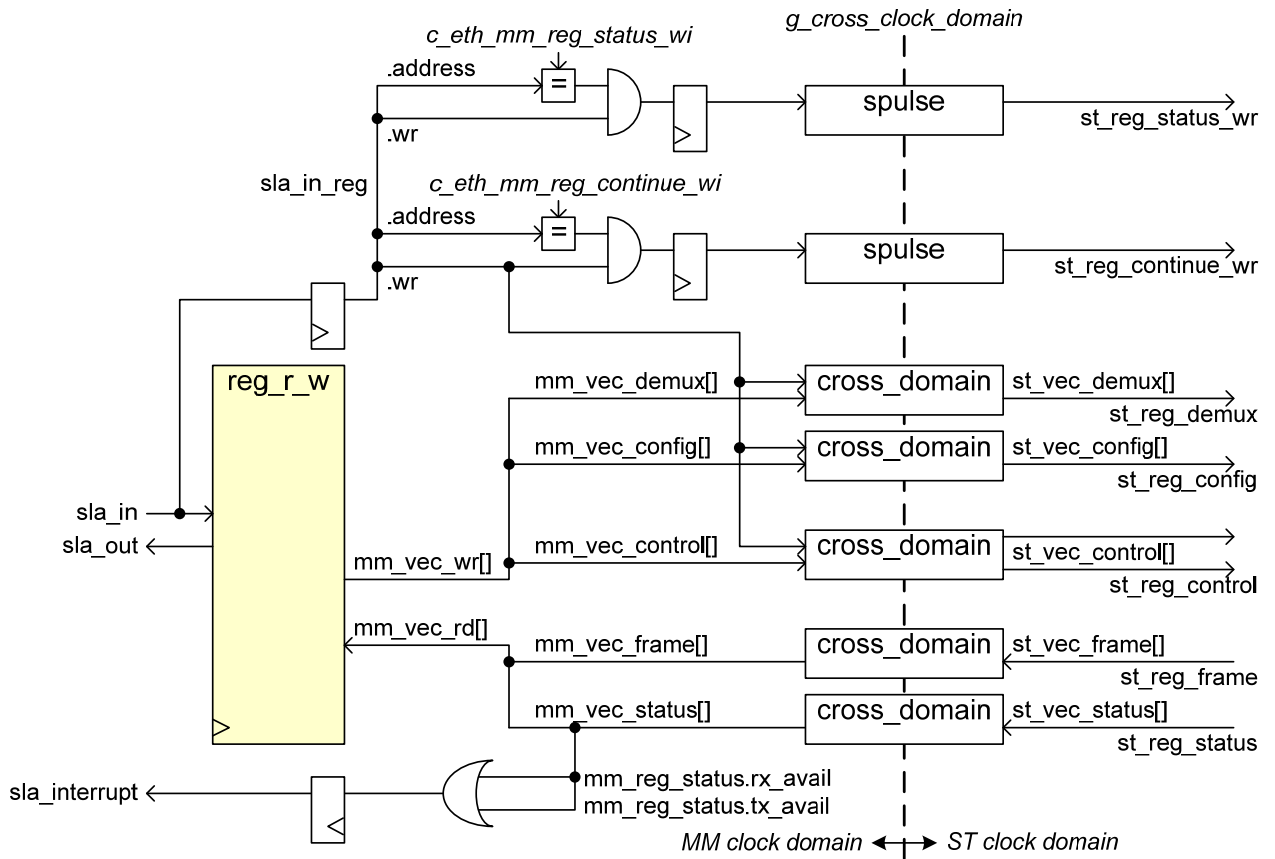


Figure 10: Block diagram for eth_mm_registers

6.8 MM packet buffer

The MM packet buffer is defined in section 2.4 and contains sufficient RAM to store an Rx packet and a Tx packet. The RAM is a dual port RAM with separate clocks, so any clock domain crossing between the MM clock domain and the ST clock domain is taken care of by default if necessary.

7 Verification

7.1 Simulation

The verification of the ETH module is done in a step by step approach. First the test bench generated by the MegaWizard was run to have a reference. Then a more compact test bench was made to verify and understand the TSE IP (section 7.1.1). After that another test bench was made to verify the ETH module with the TSE IP inside and also another instance of the TSE IP to ease interfacing a model of the control computer (section 7.1.2). Finally a test bench for a UniBoard design using the ETH module was made (section 7.1.3). This design includes the ETH module in an SOPC Builder system with a Nios II, so it can also verify the software.

7.1.1 Test bench for TSE IP

The `tb_tse` is a VHDL test bench to experiment and verify the TSE IP. The `tb_tse` test bench is much more concise than the test bench that gets generated for the TSE IP by the MegaWizard. Furthermore the `tb_tse` uses procedures from the `tb_tse_pkg` package to more easily stimulate and verify the TSE IP DUT.

Item	Description
<code>proc_tse_setup()</code>	Initializes the PCS registers according to Table 24. Initializes the MAC registers according to Table 25 and Table 26.
<code>proc_tse_tx_packet()</code>	Transmit a packet to the DUT. Supported types: ETH, ARP, ICMP, UDP. Supported payload data: byte count or word count.
<code>proc_tse_rx_packet()</code>	Receive and verify a packet

Table 22: `tb_tse_pkg`

Table 22 list the main items in the `tb_tse_pkg`. The packet procedures also handle the stream empty field in case the number of octets in the payload is not a multiple of 4, with 4 octets per word.

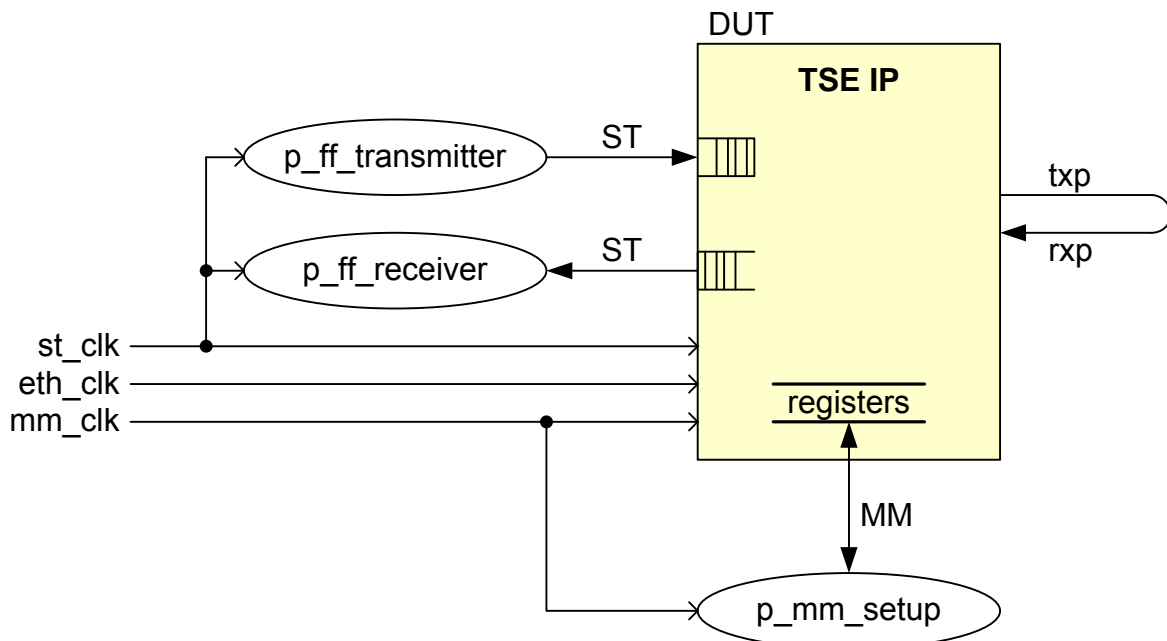


Figure 11: Architecture of `tb_tse` to verify the TSE IP

Figure 11 shows how the TSE IP is tested. The transmitter process uses the `proc_tse_tx_packet()` procedure to send packets to the DUT. The PHY of the TSE IP is connected as loopback, so the transmitted packets also get received by this DUT. The receiver process receives these packets from the DUT and verifies the payload. The receiver process continually loops and calls `proc_tse_rx_packet()`.

7.1.2 Test bench for ETH module

The `tb_eth` is a VHDL test bench to experiment and verify the entire ETH module. The `tb_eth` uses the procedures from the `tb_tse_pkg` package to set up the TSE IP and to transmit to the DUT and receive packets from the DUT.

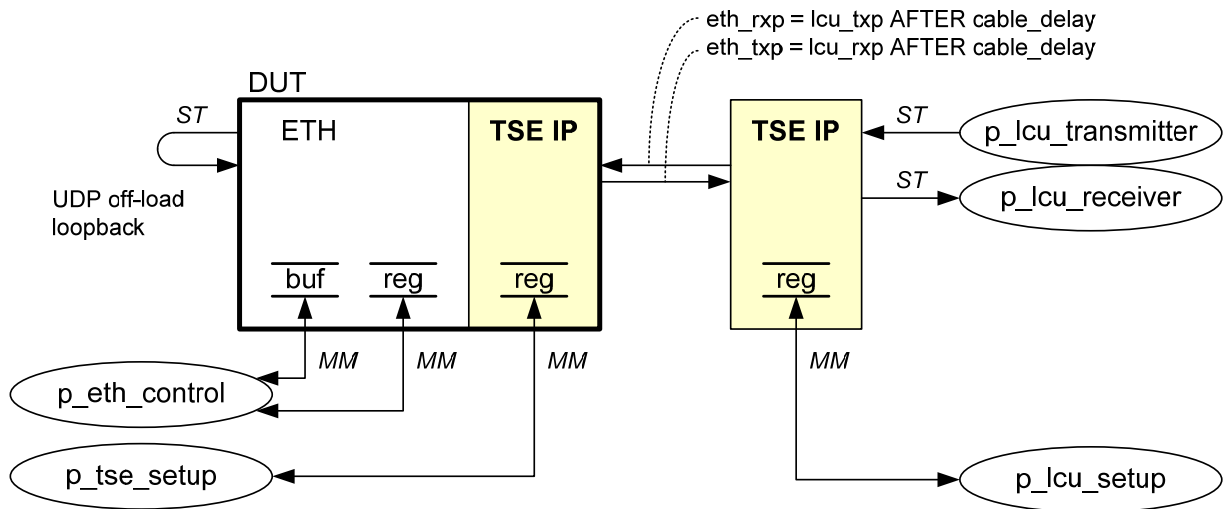


Figure 12: Architecture of `tb_eth` to verify the ETH module

Figure 12 shows how the ETH DUT is tested using another TSE IP instance to provide a MM interface for the control computer or local control unit (LCU). The LCU behaviour is modelled via the `p_lcu_transmitter` and `p_lcu_receiver` VHDL processes. The behaviour of the on chip microprocessor soft core is modelled by the `p_eth_control` VHDL process.

7.1.3 Test benches for a UniBoard SOPC design with ETH module

Figure 13 shows how the `unb_tse` design of Figure 5 is tested using similar stimuli as with `tb_eth` of Figure 12. Note that the behaviour of the `unb_tse` design is determined partly by the `main()` software function that runs on the Nios II inside the `sopc_tse` system.

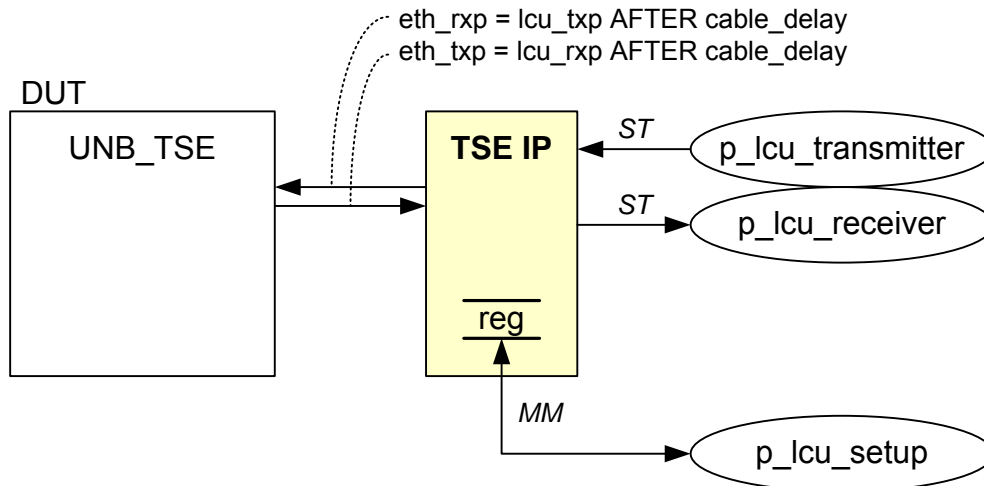


Figure 13: Architecture of tb_unb_tse to verify the ETH module in a design

Figure 14 shows how the unb_tse design of Figure 5 is tested for multiple nodes like with UniBoard. The behaviour of the Ethernet switch that connects the nodes is modelled by simply connecting the Ethernet links in a ring. For the purpose of letting each node sent to its neighbour node this limited model of the Ethernet switch is sufficient.

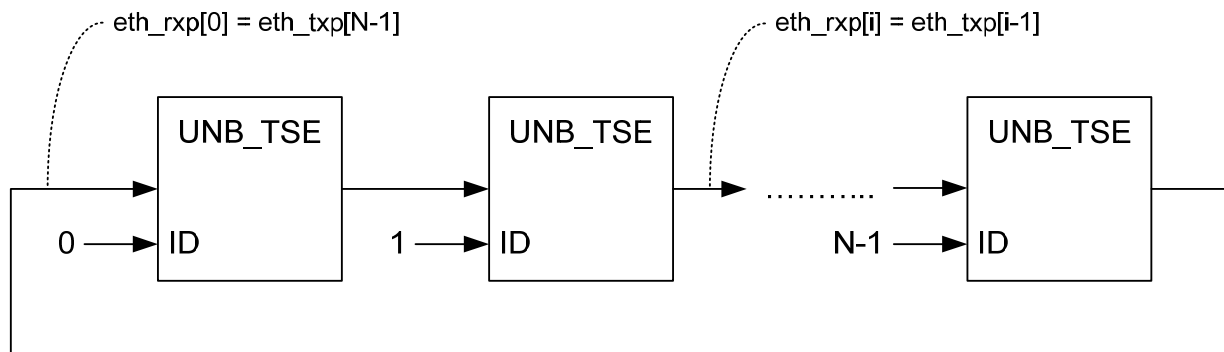


Figure 14: Architecture of tb_unb_tse_board to verify multiple nodes

Figure 15 shows how the unb_tse design of Figure 5 is tested for one node. The external loopback connection can be disconnected to verify internal TSE MAC IP loopback.

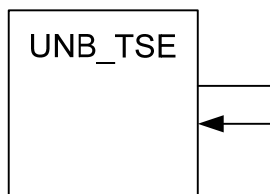


Figure 15: Architecture of tb_unb_tse_loopback to verify one node

7.2 Target hardware

The unb_tse design can run on all nodes of the UniBoard. If the Nios II runs the program with the main() function selected by ETH_MAIN_TX_RX (see Appendix 10.2) with NOF_NODES=8, then each node will transmit a frame to its neighbour every second.

8 Appendix: Ethernet protocols

8.1 Ethernet frame

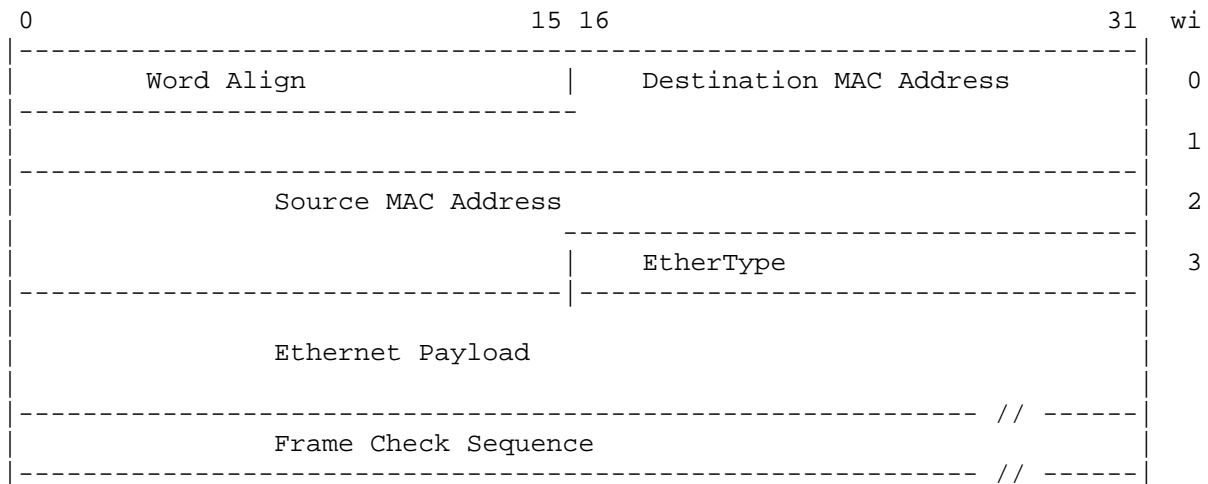


Figure 16: Ethernet frame format (with payload word alignment)

8.2 IPv4 header

IPv4 is the Internet Protocol. The Ethernet type for IP v4 is EtherType 0x800.

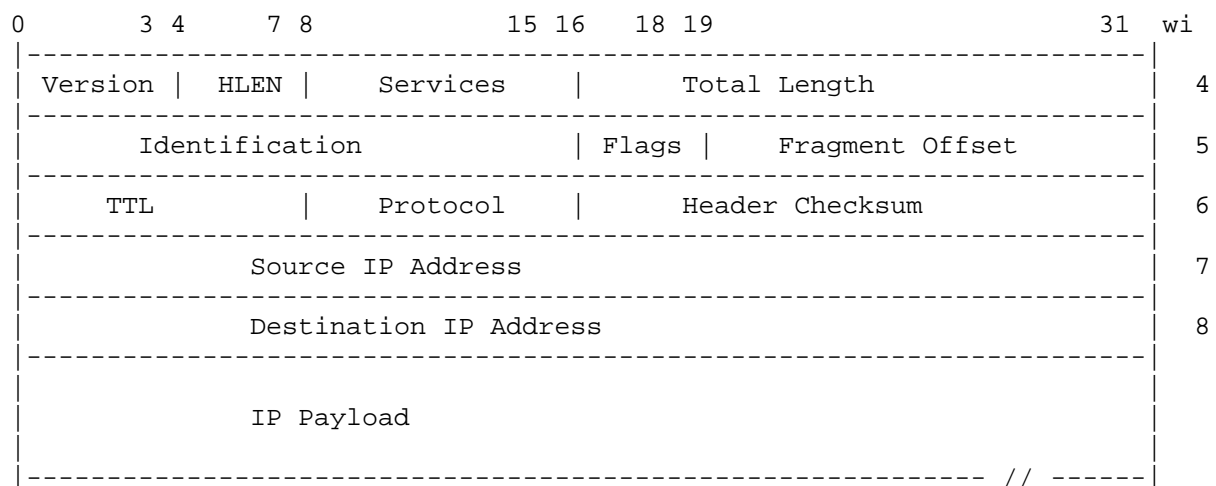


Figure 17: IPv4 header format

8.3 ARP header

ARP is the Address Resolution Protocol. The Ethernet type for ARP is EtherType 0x806.

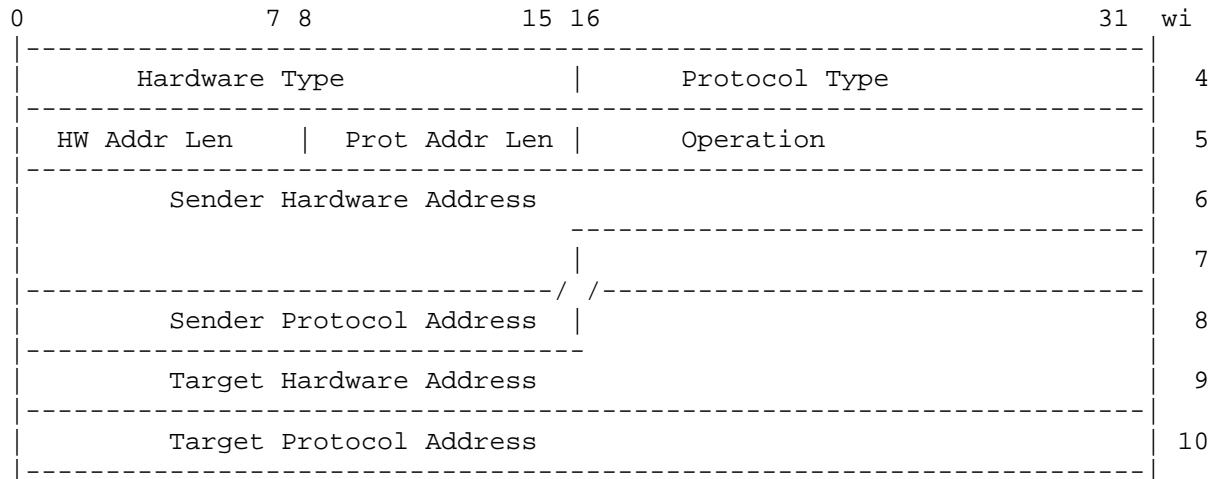


Figure 18: ARP header format

8.4 ICMP header

ICMP is Internet Control Message Protocol for ping. The IP protocol for ICMP is 1.

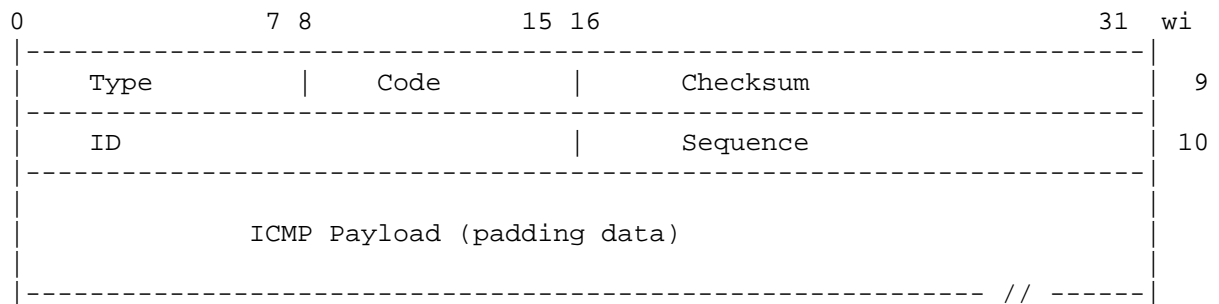


Figure 19: ICMP header format

8.5 UDP header

UDP is the User Datagram Protocol. The IP protocol for ICMP is 17.

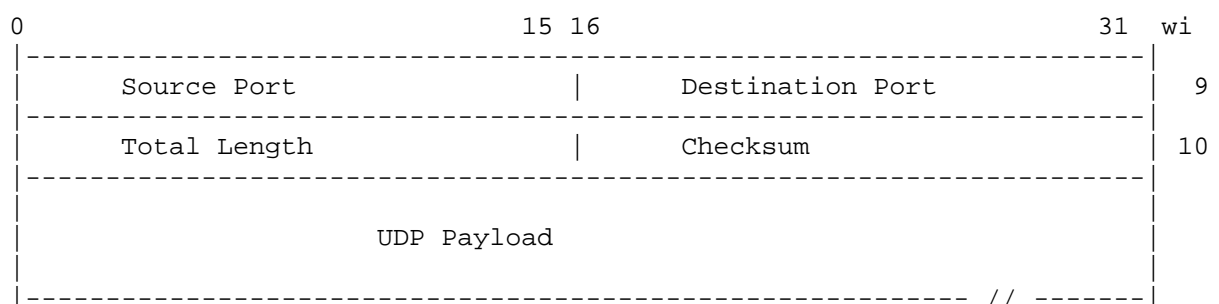


Figure 20: UDP header format

9 Appendix: TSE IP

The TSE IP MegaCore from Altera [1] provides the MAC and PHY for accessing Ethernet at 1000BASE-X. On UniBoard the PHY interface uses one LVDS line for Tx and one for Rx.

9.1 Generics

The TSE IP needs to be regenerated if its parameters in Table 23 are changed.

Parameter	Setting	Comment
ENABLE_SHIFT16	1	Align packet headers to 32 bit, useful for Nios II data handling.
ENABLE_SUP_ADDR	0	An extra MAC addresses can e.g. be used as service MAC for tests.
ENA_HASH	0	A multi cast hash table can be used to address all nodes at once.
STAT_CNT_ENA	0	PHY statistics counts are useful for monitoring, but not really needed.
EG_FIFO	256	Egress TX_FIFO_DEPTH in nof 32 bit words (256 fits 1 M9K).
ING_FIFO	256	Ingress RX_FIFO_DEPTH in nof 32 bit words (256 fits 1 M9K).
ENABLE_SGMII	0	Use 1000BASE-X direct mode for PHY interface.

Table 23: TSE IP generic settings

9.2 PCS control registers

Register	Address	Access	Value	Comment
REV	0x22	R	0x0901	PCS IP version number 9.1
IF_MODE	0x28	W	0x0008	Enable 1000BASE-X gigabit mode and gigabit speed
CONTROL	0x00	R	0x1140	Gigabit speed and full duplex (RO), auto-negotiation enabled (RW)
STATUS	0x02	R	0x000D	Link status bit 2 = '1' indicates that a valid link has been established
CONTROL	0x00	W	0x1140	Keep auto negotiate enabled (is reset default)

Table 24: TSE IP PCS control register values

9.3 MAC control registers

Register	Address	Access	Value	Comment
REV	0x000	R	0x00000901	ETH module version 0x0000 and TSE IP version 0x0901
COMMAND_CONFIG	0x008	W	0x0100004B	See Table 26
MAC_0	0x00C	W	0x78563412	E.g. MAC address 12-34-56-78-9A-BC. The MAC address for the TSE IP needs to be in big endian.
MAC_1	0x010	W	0x0000BC9A	E.g. MAC address 12-34-56-78-9A-BC.
TX_IPG_LENGTH	0x05C	W	0x0000000C	Inter packet gap 12 bytes
FRM_LENGTH	0x014	W	0x000005EE	Receive max frame length typical 1518
RX_SECTION_EMPTY	0x01C	W	0x00000F0	Default Rx FIFO depth - 16, >3
RX_SECTION_FULL	0x020	W	0x0000010	Default 16
TX_SECTION_EMPTY	0x024	W	0x00000F0	Default Tx FIFO depth - 16, >3
TX_SECTION_FULL	0x028	W	0x0000010	Default 16, > about 8 otherwise no Tx
RX_ALMOST_EMPTY	0x02C	W	0x0000008	Default 8
RX_ALMOST_FULL	0x030	W	0x0000008	Default 8
TX_ALMOST_EMPTY	0x034	W	0x0000008	Default 8
TX_ALMOST_FULL	0x038	W	0x0000003	Default TX_READY_LATENCY + 3 = 3
TX_CMD_STAT	0x0E8	R	0x00040000	[18]=1 TX_SHIFT16, [17]=0 OMIT_CRC
RX_CMD_STAT	0x0EC	R	0x02000000	[25]=1 RX_SHIFT16

Table 25: TSE IP MAC control register values

9.3.1 COMMAND_CONFIG register bits

Table 26 explains the COMMAND_CONFIG register bits that are relevant to the UniBoard.

Bit	Name	Value	Description
0	TX_ENA	1	Enable tx data path
1	RX_ENA	1	Enable rx data path
2	XON_GEN	0	-
3	ETH_SPEED	1	Enable 1GbE operation
4	PROMIS_EN	0	When 1 then receive all frames
5	PAD_EN	0	When 1 enable receive padding removal (requires Ethernet Type = payload length)
6	CRC_FWD	1	Enable receive CRC forward
7	PAUSE_FWD	0	-
8	PAUSE_IGNORE	0	-
9	TX_ADDR_INS	0	When 1 then the TSE IP overwrites Tx SRC_MAC with MAC_0,1 or one of the supplemental MAC.
[10]	HD_ENA	0	-
[11]	EXCESS_COL	0	-
[12]	LATE_COL	0	-
[13]	SW_RESET	0	When 1 then the TSE IP disables Tx and Rx, clear statistics and flushes the receive FIFO.
[14]	MHAS_SEL	0	When 1 then select multicast address resolutions hash-code mode.
[15]	LOOP_ENA	0	-
[18:16]	TX_ADDR_SEL[2:0]	0	TX_ADDR_INS insert MAC_0,1 or one of the supplemental MAC.
[19]	MAGIC_EN	0	-
[20]	SLEEP	0	-
[21]	WAKEUP	0	-

[22]	XOFF_GEN	0	-
[23]	CNT_FRM_ENA	0	-
[24]	NO_LGTH_CHECK	1	When 0 then check payload length of received frames (requires Ethernet Type = payload length).
[25]	ENA_10	0	-
[26]	RX_ERR_DISC	0	When 1 then discard erroneous frames (requires store and forward mode, so rx_section_full=0). When 0 then pass on with rx_err[0]=1
[27]	DISABLE_RD_TIMEOUT	0	
[30:28]	RSVD	0	
[31]	CNT_RESET	0	When 1 clear statistics.

Table 26: TSE IP MAC COMMAND_CONFIG register bits

9.3.2 FIFO control

Table 27 explains the meaning of the FIFO fill level settings. The section full and empty levels operate at application level and indicate whether there is enough data in the FIFO to start reading a packet from it or whether there is enough space in it to continue writing a packet to it. The almost full and empty levels operate at somewhat lower level and are used to ensure that overflow and underflow are handled correctly.

Term	Description
TX_SECTION_FULL	There is enough data in the FIFO to start reading it, when 0 then store and forward.
RX_SECTION_FULL	There is enough data in the FIFO to start reading it, when 0 then store and forward.
TX_SECTION_EMPTY	There is not much empty space anymore in the FIFO, warn user via signal ff_tx_septy.
RX_SECTION_EMPTY	There is not much empty space anymore in the FIFO, inform remote device via XOFF flow control.
TX_ALMOST_FULL	Assert signal ff_tx_a_full and deassert signal ff_tx_rdy. Furthermore TX_ALMOST_FULL = TX_READY_LATENCY+3.
RX_ALMOST_FULL	Assert signal ff_rx_a_full and if the user is not ready indicated by signal ff_rx_rdy then break off the reception with an error to avoid FIFO overflow.
TX_ALMOST_EMPTY	Assert signal ff_tx_a_empty and if the FIFO does not contain an eop yet then break off the transmission with an error to avoid FIFO underflow.
RX_ALMOST_EMPTY	Assert signal ff_rx_a_empty.

Table 27: FIFO terminology

Typical FIFO settings:

```

TX_SECTION_FULL   = 16 > 8   = TX_ALMOST_EMPTY
RX_SECTION_FULL   = 16 > 8   = RX_ALMOST_EMPTY
TX_SECTION_EMPTY  = D-16 < D-3 = TX_FIFO_DEPTH - TX_ALMOST_FULL
RX_SECTION_EMPTY  = D-16 < D-8 = RX_FIFO_DEPTH - RX_ALMOST_FULL

```

TX_FIFO_DEPTH = 1 M9K = 256*32b = 1k * 8b is sufficient when the Tx user respects signal ff_tx_rdy. To store a complete ETH packet would require 1518 byte, so 2 M9K = 2k * 8b. RX_FIFO_DEPTH = 1 M9K = 256*32b = 1k * 8b is sufficient when the Rx user signal ff_rx_rdy is sufficiently active.

10 Appendix: List of files

Not all files for the ETH module and the unb_tse example design in the SVN repository [4] are listed in this section, only the top level files, packages, test benches and the project files. For more general information on how to use the files see [7].

10.1 Firmware VHDL

10.1.1 TSE IP

The TSE IP component is generated with the Altera MegaWizard and kept at:

\$UNB/Firmware/modules/MegaWizard/tse_sgmii_lvds

File	Description
tse_sgmii_lvds.vhd	TSE IP MegaWizard top level component for synthesis.
tse_sgmii_lvds.vho	TSE IP MegaWizard top level component for simulation.

Table 28: TSE IP VHDL source and test bench files

The tse_sgmii_lvds.vhd in Table 28 defines the MegaWizard file that can generate the TSE IP files that are needed for simulation and synthesis. These generated files are also kept in SVN, so it is not necessary to run the MegaWizard to recreate them.

10.1.2 ETH module

The ETH module hardware files are kept at: \$UNB/Firmware/modules/tse

Table 29 lists the TSE IP wrapper files. By using a wrapper component the TSE IP from Altera is clearly distinguished from the generic HDL of the rest of the ETH module. If another TSE IP would be used then typically only the wrapper component will need to be adapted, all other ETH module HDL can remain the same.

File	Description
tse(pkg).vhd	TSE IP wrapper IO record definitions.
tse(stratix4).vhd	TSE IP wrapper for tse_sgmii_lvds.
tse.vhd	General TSE component with wrapper using IO records [5]

Table 29: Wrapper files for the TSE IP

Table 30 lists the ETH module package and top level entity files for Figure 6.

File	Description
eth_layers(pkg).vhd	General ETH, ARP, IPv4, ICMP, UDP and DHCP definitions (see Appendix 8).
eth(pkg).vhd	ETH module specific definitions
eth.vhd	ETH module top level of Figure 6

Table 30: Source files for the ETH module

The ETH module in eth.vhd uses ST and MM record types [5]. To make the ETH module available in SOPC Builder it is necessary to use standard logic interface signal types. Therefore the AVS_ETH wrapper component shown in Table 31 is needed.

File	Description
avs_eth.vhd	ETH module wrapper for standard Avalon Interface IO.
avs_eth_hw.tcl	Hardware description file to make the ETH module available in SOPC Builder

Table 31: Wrapper files for the AVS_ETH module

File	Description
tb_tse(pkg).vhd	Procedures to set up the TSE IP and to Tx and Rx a packet
tb_eth(pkg).vhd	Procedures to access the ETH registers and packet buffer via the MM interface
tb_tse.vhd	Test bench to verify the TSE IP via tse.vhd (Figure 11)
tb_eth.vhd	Test bench to verify ETH module via eth.vhd (Figure 12)
tb_tb_eth.vhd	Test bench to run multiple settings of tb_eth.vhd (Figure 12)
tb_eth_checksum.vhd	Test bench to verify the IP header checksum calculation in eth_checksum.vhd

Table 32: Test bench files for ETH module

10.1.3 UNB_TSE design

The files for the unb_tse example design with the ETH module are kept at: \$UNB/Firmware/designs/unb_tse.

File	Description
sopc_tse.sopc	SOPC Builder design file of Figure 4.
unb_tse.vhd	Node design of Figure 5 that includes the socp_tse system with the AVS_ETH module inside and that is suitable for all UniBoard modes.

Table 33: Source files for unb_tse design

File	Description
tb_unb_tse.vhd	Test bench of Figure 13 with LCU that transmits packets to the unb_tse.
tb_unb_tse_board.vhd	Test bench of Figure 14 with one or more unb_tse that are connected in a ring
tb_unb_tse_loopback.vhd	Test bench of Figure 15 with one the unb_tse and external loopback

Table 34: Test bench files for unb_tse design

10.2 Software C, H

10.2.1 Module

The UniBoard software modules are stored at: \$UNB/Firmware/software/modules/src.

File	Description
avs_eth_regs.h	Constants and macros to MM access the ETH module.
avs_eth.h	Public functions to set up the ETH module and to hook the ISR
avs_eth.c	Implements for avs_eth.h

Table 35: Module software C, H files

10.2.2 Main

The UniBoard software applications with main() examples for the modules are stored at:

\$UNB/Firmware/software/apps

Table 36 lists the main() function examples that are available for the ETH module.

Main()	Description
ETH_MAIN	Report received frame and reply using the default header and same length. Modelsim simulate by sending e.g. 1 frame with tb_unb_tse.vhd.
ETH_MAIN_TX_RX	Node transmits raw Ethernet frames to next node. Report received frames. Modelsim simulate using tb_unb_tse_board.vhd for ≥ 1 nodes or tb_unb_tse_loopback.vhd for 1 node.
ETH_MAIN_TSE	Read some PCS and MAC registers and report. Modelsim simulate using tb_unb_tse.vhd.

Table 36: Application software main C files

All these ETH main() report results via the JTAG UART using printf(). On target hardware the JTAG UART output can be observed in a Nios II Command Shell. In Modelsim simulation the JTAG UART output appears in the Modelsim transcript window. In simulation the main() functions also provide program status information in the Wave window via the debug wave parallel output (signal pout_debug_wave in Figure 5).

10.3 Simulation

The simulation project files are located in:

\$UNB/Firmware/modules/tse/build/sim/modelsim/ → for ETH module tse_lib library.

\$UNB/Firmware/designs/unb_tse/build/synth/quartus/sopc_tse_sim/ → for the unb_tse design.

File	Description
tse.mpf	Modelsim project file that builds the tse_lib module library and provides the simulation configurations for the ETH module test benches of Table 32.
setup_sim.do	Modelsim simulation do-file generated by SOPC Builder for socp_tse.sopc, only used as an example. For simulation use unb_tse.mpf.
unb_tse.mpf	Modelsim project file for the unb_tse design. Builds the unb_tse work library and provides the simulation configurations for the unb_tse test benches of Table 34.

Table 37: Simulation project files

10.4 Synthesis

The synthesis files are kept at: \$UNB/Firmware/designs/unb_tse/build/synth/quartus

File	Description
unb_tse.qsf	Quartus II Project File.
unb_tse.qsf	Quartus II Settings File, lists all source files
unb_tse.sdc	Synopsys Design Constraint file, constraints the clock

Table 38: Synthesis project and settings files