Ethernet IP Core Design Document

Author: Igor Mohor

IgorM@opencores.org

Rev. 0.3

October 29, 2002

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
0.1	09/09/02	Igor Mohor	First Draft
0.2	22/10/02	Igor Mohor	Description of Core Modules added (figure), Some test description added.
0.3	29/10/02	Igor Mohor	Some figures added.

List of Contents

List of Tables

List of Figures

1 Introduction

1.1 Ethernet IP Core Introduction

The Ethernet IP Core is a MAC (Media Access Controller). It connects to the Ethernet PHY chip on one side and to the WISHBONE SoC bus on the other. The core has been designed to offer as much flexibility as possible to all kinds of applications.

The chapter 2 describes file hierarchy, description of modules, core design considerations and constants regarding the Ethernet IP Core.

The chapter 3 describes test bench file hierarchy, description of modules, test bench design considerations, description of test cases and constants regarding the test bench.

1.2 Ethernet IP Core Features

The following lists the main features of the Ethernet IP core.

- Performing MAC layer functions of IEEE 802.3 and Ethernet
- Automatic 32-bit CRC generation and checking
- Delayed CRC generation
- Preamble generation and removal
- Automatically pad short frames on transmit
- Detection of too long or too short packets (length limits)
- Possible transmission of packets that are bigger than standard packets.
- Full duplex support

- 10 and 100 Mbps bit rates supported
- Automatic packet abortion on Excessive deferral limit, too small inter packet gap, when enabled
- Flow control and automatic generation of control frames in full duplex mode (IEEE 802.3x)
- Collision detection and auto retransmission on collisions in half duplex mode (CSMA/CD protocol)
- Complete status for TX/RX packets
- IEEE 802.3 Media Independent Interface (MII)
- WISHBONE SoC Interconnection Rev. B2 and B3 compliant interface
- Internal RAM for holding 128 TX/RX buffer descriptors
- Interrupt generation an all events

1.3 Ethernet IP Core Directory Structure

Following picture shows the structure of directories of the Ethernet IP core.

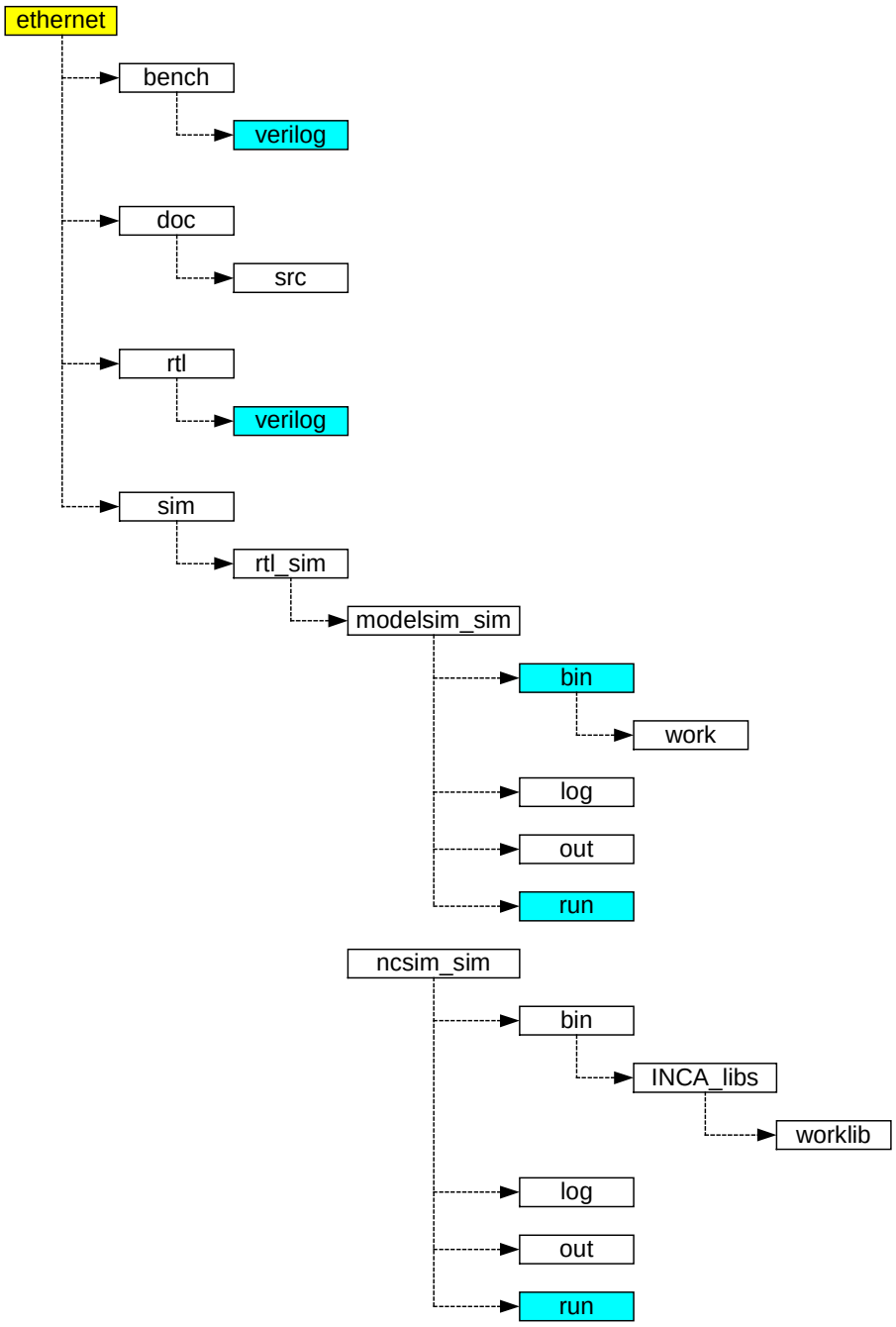

Figure 1: Ethernet IP Core Core Directory Structure

There are two major parts of the Verilog code in the **ethernet** directory. First one is the code for the Ethernet MAC IP core. The Verilog files are in the **ethernet\rtl\verilog** subdirectory. The second one is the code for the Ethernet MAC Testbench. These files are used together with files for the Ethernet MAC. There are also some exceptions, but those will be mentioned later. The Verilog files are in the **ethernet\bench\verilog subdirectory**.

The documentation is in the subdirectory **ethernet\doc**. Documentation consists of Ethernet IP Core Data Sheet, Ethernet IP Core Specification and Ethernet IP Core Design document.

ethernet\sim subdirectory is used for running simulation – testbench. The **rtl_sim** subdirectory is used for RTL (functional) simulation of the core. There are two sets of scripts for running the simulation. First set is used for running the testbench using NCSim simulator. Second set is used for running the testbench using ModelSIM simulator. Both are using the similar directory structure:

- **bin** – includes various scripts needed for running Ncsim simulator
- **run** – the directory from which the simulation is run. It provides a script for starting the simulation and a script for cleaning all the results produced by previous simulation runs
- **log** – Ncvlog, Ncelab and Ncsim log files are stored here for review.
- **out** – simulation output directory – simulation stores all the results into this directory (dump files for viewing with Signalscan, testbench text output etc.)

Generated files from synthesis tools, like gate level Verilog and log files, are stored in the **ethernet\syn** subdirectory and its subdirectories.

2

Ethernet MAC IP Core

2.1 Overview

The Ethernet MAC IP Core consists of seven main units: WISHBONE interface, transmit module, receive module, control module, MII module, status module and register module. Many of these modules have sub-modules. Module and sub-module operations are described later in this section.

2.1.1 WISHBONE Interface

Consists of both master and slave interfaces and connects the core to the WISHBONE bus. Master interface is used for storing the received data frames to the memory and loading the data that needs to be sent from the memory to the Ethernet core. Interface is WISHBONE Revision B.2 and B.3 compatible (selectable with a define ETH_WISHBONE_B3 in the eth_defines.v file).

2.1.2 Transmit Module

Performs all transmitting related operations (preamble generation, padding, CRC, etc.).

2.1.3 Receive Module

Performs all reception related operations (preamble removal, CRC check, etc).

2.1.4 Control Module

Performs all flow control related operations when Ethernet is used in full duplex mode.

2.1.5 MII Module (Media Independent Module)

Provides a Media independent interface to the external Ethernet PHY chip.

2.1.6 Status Module

Records different statuses that are written to the related buffer descriptors or used in some other modules.

2.1.7 Register Module

Registers that are used for Ethernet MAC operation are in this module.

2.2 Core File Hierarchy

The hierarchy of modules in the Ethernet core is shown here with file tree. Each file implements one module in a hierarchy. RTL source files of the Ethernet core are in the **ethernet\rtl\verilog** subdirectory.

ethernet

. sim

. . rtl_sim

. . . src

```
.   .   .   run
.   rtl
.   .   verilog
.   .   .   eth_top.v
.   .   .   eth_crc.v
.   .   .   eth_cop.v
.   .   .   eth_miim.v
.   .   .   eth_defines.v
.   .   .   timescale.v
.   .   .   eth_random.v
.   .   .   eth_fifo.v
.   .   .   eth_wishbone.v
.   .   .   eth_maccontrol.v
.   .   .   eth_rxaddrcheck.v
.   .   .   eth_txstatem.v
.   .   .   eth_transmitcontrol.v
.   .   .   eth_txethmac.v
.   .   .   generic_spram.v
.   .   .   eth_rxcounters.v
.   .   .   eth_rxstatem.v
.   .   .   eth_outputcontrol.v
.   .   .   eth_register.v
.   .   .   eth_receivecontrol.v
.   .   .   eth_registers.v
.   .   .   eth_shiftreg.v
.   .   .   eth_txcounters.v
.   .   .   eth_clockgen.v
.   .   .   eth_rxethmac.v
.   .   .   eth_macstatus.v
.   doc
.   .   eth_speci.pdf
.   .   eth_design_document.pdf
```

. . Ethernet Datasheet (prl.).pdf
. . src
. . . eth_speci.doc
. . . eth_design_document.doc
. . . Ethernet Datasheet (prl.).doc
. bench
. . verilog
. . . tb_ethernet.v
. . . tb_eth_defines.v
. . . tb_cop.v
. . . eth_host.v
. . . eth_memory.v

2.3 Description of Core Modules

The module **eth_top.v** consists of sub modules **eth_miim.v**, **eth_registers.v**, **eth_maccontrol.v**, **eth_txethmac.v**, **eth_rxethmac.v**, **eth_wishbone.v**, **eth_macstatus.v** and some logic for synchronizing, multiplexing and registering outputs.

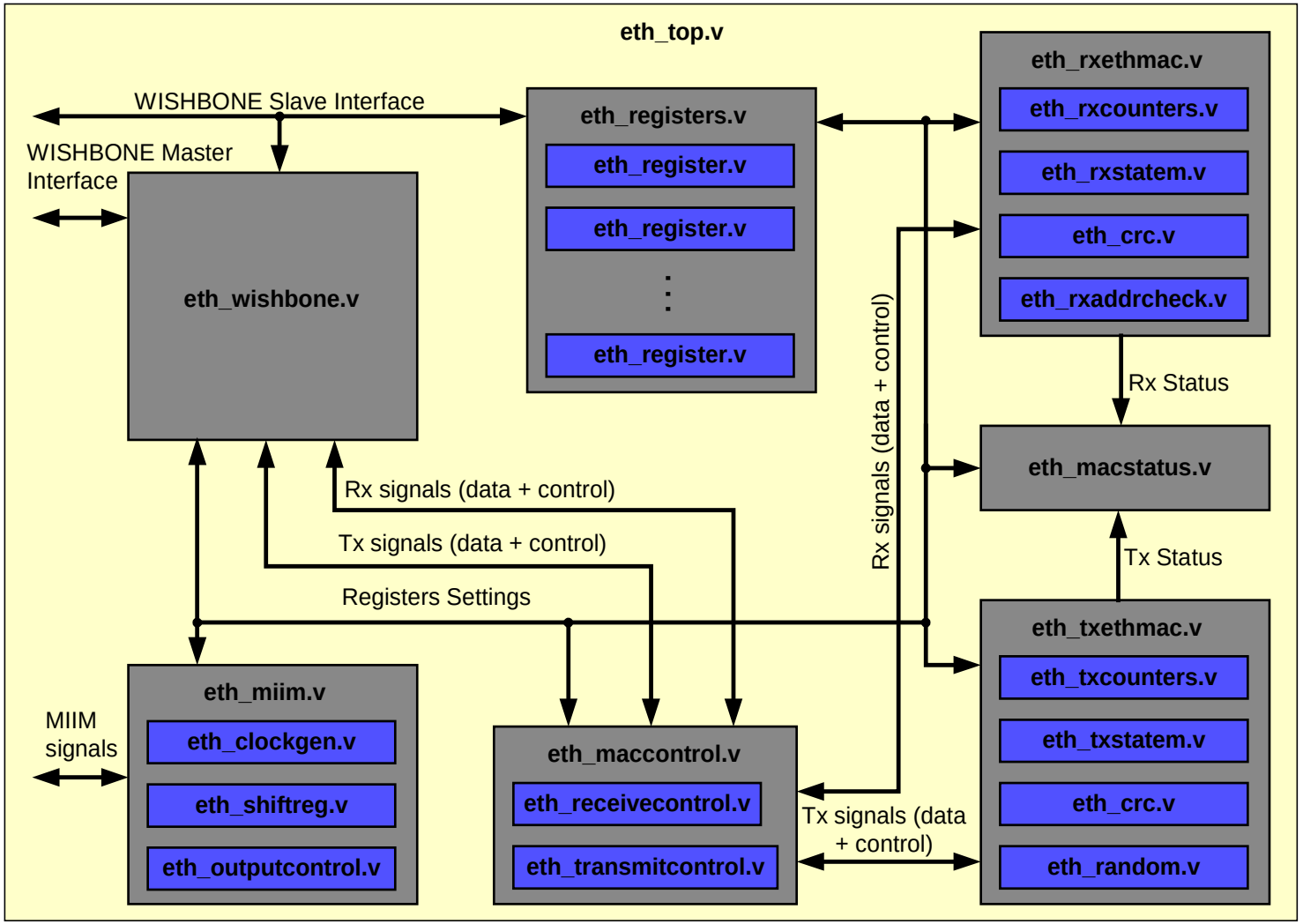

Figure 2: Core Modules

2.3.1 Description of the MII module (eth_miim.v)

The MII module (Media Independent Interface) is an interface to the external Ethernet PHY chip. It is used for setting PHY's configuration registers and reading status from it. The interface consists of only two signals: clock (**MDC**) and bi-directional data signal (**MDIO**). Bi-directional MDIO signal needs to be combined from input signal **Mdi**, output signal **Mdo**, and enable signal **MdoEn** in additional module. This is done because the same Ethernet core will be implemented in both ASIC and FPGA.

The MII module is the top module for the MII and consists of several sub modules (**eth_clockgen.v**, **eth_shiftreg.v**, **eth_outputcontrol.v)** and additional logic. This logic is used for generating number of signals:

- Synchronized request for write (**WriteDataOp**), read (**ReadStatusOp**) and scan (**ScanStatusOp**) operations.
- Signal for updating the MIIRX_DATA register (**UpdateMIIRX_DATAReg**)
- Counter (**BitCounter**) is the primary counter for the MII Interface (many operations depend on it.
- Byte select signals used when data is shifted out (**ByteSelect [3:0]**).
- Signals used for latching the input data (**LatchByte [1:0]**).

When there is a need to read or write the data from the PHY chip, several operations need to be performed:

- MIIMODER register needs to be set:
  - Clock divider needs to be set to provide clock signal **Mdc** of the appropriate frequency (read PHY documentation to obtain the value of the Mdc frequency)
  - Preamble generation might be disabled (if PHY supports transmissions without the preamble). By default 32-bit preamble is transmitted.
  - MII Module might be reset prior to its usage.
- PHY Address (several PHY chips might be connected to the MII interface) and address of the register within the selected PHY chip need to be set in the MIIADDRESS register.
- If there is a need to write data to the selected register, data needs to be written to the MIITX_DATA register.
- Writing appropriate value to the MIICOMMAND register starts requested operation.

- If “Read status” or ”Scan status” operation were requested than the value that was received from the PHY can be read from the MIIRX_DATA register.

MIISTATUS register reflects the status of the MII module. The **LinkFail** status is cleared only after the read to the PHY’s status register (address 0x1) returns status that is OK.

2.3.2.1 Description of the eth_outputcontrol module

This module performs two tasks:

- Generates MII serial output signal (**Mdo**)
- Generates enable signal (**MdoEn**) for the **Mdo**.

Since the MII serial data signal is a bi-directional signal, these two signals need to be combined together with the MII serial input signal (**Mdi**) in additional module that is not part of the Ethernet MAC IP Core.

The eth_outputcontrol module also generates the MII preamble. When MII preamble is enabled (bit 8 in the MIIMODER register set to 0), 32-bit preamble is transmitted prior to the data.

2.3.2.2 Description of the eth_clockgen module

The eth_clockgen module is used for:

- Generating MII clock signal (**Mdc**). This is output clock signal used for clocking the MII interface of the Ethernet PHY chip. You should read the specification for the used PHY chip to properly set the Mdc frequency (usually frequencies up to 10 MHz can be used)
- Generating **MdcEn** signal. This signal is an enable signal. All flip-flops used in the MII are clocked with the high frequency clock **Clk**. The reduced frequency (equal to Mdc) is obtained by using the MdcEn signal.

Mdc is obtained by dividing the **Clk** signal with the value that is written in the MIIMODER register (any value within range [1:255]).

2.3.2.3 Description of the eth_shiftreg module

The eth_shiftreg module is used for:

- Serialize the data that goes towards Ethernet PHY chip (Mdo)
- Parallelize input data that comes from Ethernet PHY chip (Mdi) and temporally store it to the **Prsd** register. This value is then stored to the MIIRX_DATA register.
- Generating LinkFail signal (bit 0 of the MIISTATUS register reflects its value).

2.3.2 Description of the Receive module (eth_rxethmac.v)

The Receive module is in charge for receiving data. External PHY chip receives serial data from the physical layer (cable), assembles it to nibbles and sends to the receive module (**MRxD [3:0]**) together with the “data valid” marker (**MRxDV**). The receive module then assembles this data nibbles to data bytes, and sends them to the WISHBONE interface module together with few signals that mark start and end of the data. Receive module also removes the preamble and the CRC.

The Receive module consists of four sub modules:

- **eth_crc** – Cyclic Redundancy Check (CRC) module
- **eth_rxaddrcheck** – Address recognition module
- **eth_rxcounters** – Various counters needed for packet reception
- **eth_rxstatem** – State machine for Receive module

Besides the above sub modules, eth_rxethmac module also consists of logic that is used for:

- Generating **CrcHash** value and **CrcHashGood** marker that are used in address recognition system.
- Latching the data that is received from the PHY chip (**RxData**).
- Generating **Broadcast** and **Multicast** marker (when packets with broadcast or multicast destination address are received).
- Generating **RxValid**, **RxStartFrm**, **RxEndFrm** signals that are marking valid data.

Receiver can operate in various modes. For that reason number of registers need to be configured prior to Receiver's use.

Signals related to the receiver operation are:

- **HugEn** – Reception of big packets is enabled (packets, bigger than the standard Ethernet packets). When HugEn is disabled, packets that smaller or equal to MaxFL and bigger or equal to MinFL are received. (MaxFL and MinFL are set in the PACKETLEN register).
- **DlyCrcEn** – Delayed CRC (Cyclic Redundancy Check) is enabled. CRC checking starts 4 bytes after the data becomes valid. This option is useful when additional data is added to the data frame.
- **r_IFG** – Minimum Inter Frame Gap Enable. When this signal is set to zero, minimum inter frame gap is required between two packets. After this time receiver starts with reception again. When r_IFG is set to 1, no inter packet gap is needed. All frames are received regardless to the IFG.
- **r_Pro, r_Bro, r_Iam** and registers **MAC, HASH0** and **HASH1** are used for address recognition.

2.3.2.1 Description of the CRC (Cyclic Redundancy Check) module (eth_crc.v)

This module is used for validating the correctness of the incoming packet by checking the CRC value of the packet. CRC module is also used for the CRC generation for the TX module.

To better understand the CRC checking, here is a brief description how CRC is send and checked.

Before a transmitter sends the data, it appends the CRC (this CRC is calculated from the data) to it. This means that the packet is now bigger for 4 bytes. Receiver receives this data (that **also includes the CRC of the data**) and calculates a new CRC value from it (received CRC is also used for the CRC calculation). If the new CRC differs from the "CRC Magic Number" (0xc704dd7b), then received data differs from the sent data and **CrcError** signal is set.

2.3.2.2 Description of the address recognition module (eth_rxaddrcheck.v)

The address recognition module decides whether the packet will be received or not. Ethernet IP core starts receiving all packets regardless to their destination address. Destination address is then checked in the eth_rxaddrcheck sub module. Frame reception depends on few conditions:

- If **r_Pro** bit is set in the MODER register (Promiscuous mode), then all frames are received regardless to their destination address. If r_Pro bit is cleared then destination address is checked.
- If **r_Bro** bit is set in the MODER register then all frames containing broadcast addresses are rejected (r_Pro must be cleared).
- **MAC** – MAC address of the used Ethernet MAC IP Core. This is individual address of the used Ethernet core. When r_Pro bit is cleared then every destination address is compared to the MAC address. Frame is accepted only when two address match.
- When **r_Iam** signal is set then besides checking the MAC address, hash table algorithm is used. The Ethernet controller maps any 48-bit address into one of 64 bits. If that bit is set in the HASH registers (**HASH0** and **r_HASH1** are making one 64-bit hash register), then frame is accepted.

As said before, packet reception always starts regardless of the destination address of the incoming packet. As soon as the destination address is received, it is checked if it matches with any of the above-mentioned conditions. If the match doesn't occur than the reception of the whole packet is aborted (signal **RxAbort** is set to 1). The packet is not written to the memory and receive buffer is flushed.

2.3.2.3 Description of the rxcounters module (eth_rxcounters.v)

The module consists of three counters, which are:

- **ByteCnt** – generally used counter in the receive module.
- **IFGCounter** – used for counting the IFG (inter frame gap)
- **DlyCrcCnt** – counter, used when delayed CRC operation is enabled.

Besides that a number of comparators are in this module, used for various purposes.

2.3.2.4 Description of the rxstatem module (eth_rxstatem.v)

There is just one state machine used in the receive module of the Ethernet IP core. This module is placed in the eth_rxstatem sub-module.

The state machine has six different states:

- Idle state
- Drop state
- Preamble state
- SFD (standard frame delimiter) state
- Data 0 state
- Data 1 state

State machine (SM) goes to the drop state (**StateDrop**) after the reset and immediately after that to the idle state (**StateIdle**) because **MRxDV** is set to 0. As soon as there is a valid data available on the PHY's data lines (**MRxD**), PHY informs receiver about that by setting the **MRxDV** signal to one.

Normally receiver expects preamble at the beginning of each packet. Standard preamble is 7 byte long (0xee). After that a one-byte SFD (start frame delimiter) is expected (0xde). If we put this together, then sample 0xdeeeeeee is expected (LSB received first).

Because the Ethernet IP core can also accept packets that don't have a standard 7-byte preamble but only the SFD, receiver's SM waits for the first 0x5 nibble (it is not important whether this nibble is part of the preamble or of the SFD). If the received character differs from the expected nibble, then the SM goes to the preamble state (**StatePreamble**) and remains there until the correct nibble (0x5) is received. Once the 0x5 nibble is received, SM goes to the SFD state (**StateSFD**) where it waits for the 0xd nibble.

From here two things, depending on the value of the **IFGCounterEq24** signal, may occur (next paragraph describes IFGCounterEq24 signal). If IFGCounterEq24 is set then:

- SM goes to the data0 state (**StateData0**) where lower data nibble is received and then to the data1 state (**StateData1**) where higher data nibble is received. SM goes back to the data0 state. SM continues going from data state 0 to data state 1 and vice versa until whole data packet is received and end of packet is detected (PHY clears the **MRxDV** signal). Once the data valid signal is cleared, SM goes to the idle state (**StateIdle**) and everything starts again.

else (IFGCounterEq24 is cleared)

- SM goes to the drop state (**StateDrop**) and remains there until the end of valid data is reported (PHY clears the **MRxDV** signal). After that SM goes to the idle state (**StateIdle**) and everything starts again.

Signal **IFGCounterEq24** is used for detecting the proper gap between two consecutive received frames (Inter Frame Gap). By the standard this gap must be at least 960 ns for 100 Mbps mode or 9600ns for 10 Mbps mode. If the gap is appropriate (equal or greater than requested), then **IFGCounterEq24** is set to 1. Signal **IFGCounterEq24** is also set to 1, when IFG bit in the MODER register is set (minimum inter frame gap is not checked). If the IFG gap between two frames is too small, frame won't be accepted but dropped.

2.3.3 Description of the Transmit module (eth_txethmac.v)

The Transmit module (TX) is in charge for transmitting data. TX module gets data that needs to be transmitted from WISHBONE interface (WBI) module in the byte form. Besides that it also receives signals that mark start of the data frame (**TxStartFrm**) and end of the data frame (**TxEndFrm**). As soon as the TX module needs next data byte, it sets the **TxUsedData** and WBI module provides the next byte.

TX module sets number of signals to inform WBI module on one side and Ethernet PHY chip on the other about the operation status (done, retry, abort, error, etc.).

The Transmit module consists of four sub modules:

- **eth_crc** – Cyclic Redundancy Check (CRC) module generates 32-bit CRC that is appended to the data field.
- **eth_random** – Generates random delay that is needed when back off is performed (after the collision)
- **eth_txcounters** – Various counters needed for packet transmission
- **eth_txstatem** – State machine for TX module

Signals, connected to the Ethernet PHY chip are:

- Data nibble **MTxD**. This is the data that will be sent on the Ethernet by the PHY.
- Transmit enable **MTxEn** tells PHY that data **MTxD** is valid and transmission should start.
- Transmit error **MTxErr** tells PHY that an error happened during the transmission.

Signals, connected to the upper layer module (WBI module) are:

- Transmit packet done **TxDone** (see next paragraph)
- Transmit packet retry **TxRetry** (see next paragraph)
- Transmit packet abort **TxAbort** (see next paragraph)
- **TxUsedData**;

Every transmission ends in one of the following ways:

- Transmission is successfully finished. Signal **TxDone** is set.
- Transmission needs to be repeated. Signal **TxRetry** is set. This happens when a normal collision occurs (in half-duplex mode).
- Transmission is aborted. Signal TxAbort is set. This happens in the following situations:
    - Packet is too big (bigger than the max. packet (See MAXFL field of the PACKETLEN register)).
    - Underrun occurs (WBI module can not provide data on time).
    - Excessive deferral occurs (TX state machine remains in the defer state for too long).
    - Late collision occurs (late collision is every collision that happens later than COLLVALID bytes after the preamble (See COLLCONF register)).
    - Maximum number of collisions happens (See MAXRET field of the COLLCONF register).

Besides all previously mentioned signals, TX module provides other signals:

- **WillTransmit** notifies the receiver that transmitter will start transmitting. Receiver stops receiving until WillTransmit is cleared.
- Generating the collision reset signal (“collision detected” asynchronously comes from the PHY chip and is synchronized to the TX clock signal). **ResetCollision** signal is used to reset synchronizing flip-flop.
- Collision window **ColWindow** marks a window within every collision is treated as a valid (regular) collision. After a collision packet is retransmitted. Every collision that occurs after that is a late collision (packets with late collision are aborted).
- Retry counter **RetryCnt**.
- **Data_Crc, Enable_Crc** and **Initialize_Crc** that are used for CRC generation.

2.3.3.1 Description of the CRC (Cyclic Redundancy Check) module (eth_crc.v)

This module is used for CRC calculation. The calculated CRC is appended to the data frame. This module is also used in the RX module for CRC checking.

2.3.3.2 Description of the random module (eth_random.v)

When a collision occurs, TX module first sends a “jam” pattern (0x99999999) and then stops transmitting. Before a retransmission starts, TX performs a backoff. TX waits before it starts transmitting for some amount of time. The amount of time is “semi” random and is calculated in the eth_random module. Binary Exponential algorithm is used for that purpose. Backoff time is random within predefined limits. This limits increase with the number of collisions.

2.3.3.3 Description of the TX counters module (eth_txcounters.v)

There are three counters in the eth_txcounters module. These counters are only used in the TX modules.

The **DlyCrcCnt** counter is used when a delayed CRC generation is needed to count

The nibble counter **NibCnt** count nibbles while **ByteCnt** counts bytes. Which one of the counters is used depends off the needed resolution.

2.3.3.4 Description of the TX state machine module (eth_txstatem.v)

The TX module has one general state machine that is in the eth_txstatem module. This state machine has eleven states:

- StateIdle
- StatePreamble
- StateData0
- StateData1
- StatePAD
- StateFCS
- StateIPG
- StateJam
- StateJam_q
- StateBackOff
- StateDefer

After the reset defer state (**StateDefer**) is activated. After that the state machine goes to the “Inter Packet Gap” state (**StateIPG**) and then to the idle state (**StateIdle**). Why this is so, is not important at the moment.

Let’s start with the description after the state machine comes to the idle state. This is the most often used state. When transmitter has nothing to do, it waits in the idle mode for the transmission request. Wishbone Interface (WBI) requests the transmission by setting the **TxStartFrm** signal to 1 for two clock cycles (together with the first byte of the data that needs to be sent). This forces the state machine (SM) to go to the preamble state (**StatePreamble**). In the preamble state **MTxEn** signal is set to 1, informing the Ethernet PHY chip that transmission will start. Together with the MTxEn signal, data signal **MTxD** is set to the preamble value 0x5. After the preamble is sent (0x5555555), SFD is sent (Start Frame Delimiter (0xd)). After that SM goes to the data0 state (**StateData0**) and signal **TxUsedData** is set to inform the WBI to provide next data byte. LSB nibble of the data byte is sent and then SM goes to the data1 state (**StateData1**), where the MSB nibble of the data byte is sent. SM continues to switch between the data0 and data1 states until the end of the packet. When there is just one byte left to be send, WBI sets the signal **TxEndFrm** that marks the last byte of the data that needs to be sent.

From here, there are several possibilities:

- If the **data length** is **greater or equal** to the minimum frame length (value written in the **MINFL** field of the PACKETLEN register) and **CRC** is **enabled** (bit CRCEN in the MODER register is set to 1 **or** bit CRC of the transmit descriptor is set to 1) then SM goes to the **StateFCS** state where the 32-bit CRC value, calculated from the data, is appended. Then the SM goes to the defer state (**StateDefer**), then to the “Inter Packet Gap” state (**StateIPG**) and from there to the idle state (**StateIdle**) where everything starts again.
- If the **data length** is **greater or equal** to the minimum frame length (value written in the **MINFL** field of the PACKETLEN register) and **CRC** is **disabled** (bit CRCEN in the MODER register is set to 0 **and** bit CRC of the transmit descriptor is set to 0) then SM goes to the defer state (**StateDefer**), then to the “Inter Packet Gap” state (**StateIPG**) and from there to the idle state (**StateIdle**) where everything starts again.
- If the **data length** is **smaller** than the minimum frame length (value written in the **MINFL** field of the PACKETLEN register) and **padding** is **enabled** (bit PAD in the MODER register is set to 1 **or** bit PAD of the transmit descriptor is set to 1), then the SM goes to the pad state (**StatePAD**) where data is padded with zeros until the minimum frame length is achieved. Then the SM goes to the **StateFCS** state where the 32-bit CRC value, calculated from the data, is appended. Then the SM goes to the defer state (**StateDefer**), then to the “Inter Packet Gap” state (**StateIPG**) and from there to the idle state (**StateIdle**) where everything starts again.
- If the **data length** is **smaller** than the minimum frame length (value written in the **MINFL** field of the PACKETLEN register), **padding** is **disabled** (bit PAD in the MODER register is set to01 **and** bit PAD of the transmit descriptor is set to 0) and **CRC** is **enabled** (bit CRCEN in the MODER register is set to 1 **or** bit CRC of the transmit descriptor is set to 1) then the SM goes to the **StateFCS** state where the 32-bit CRC value, calculated from the data, is appended. Then the SM goes to the defer state (**StateDefer**), then to the “Inter Packet Gap” state (**StateIPG**) and from there to the idle state (**StateIdle**) where everything starts again.
- If the **data length** is **smaller** than the minimum frame length (value written in the **MINFL** field of the PACKETLEN register), **padding** is **disabled** (bit PAD in the MODER register is set to01 **and** bit PAD of the transmit descriptor is set to 0) and **CRC** is **disabled** (bit CRCEN in the MODER register is set to 0 **and** bit CRC of the transmit descriptor is set to 0) then the SM goes to the defer state (**StateDefer**), then to the “Inter Packet Gap” state (**StateIPG**) and from there to the idle state (**StateIdle**) where everything starts again.

2.3.4 Description of the Control module (eth_maccontrol.v)

The Control module is in charge for data flow control, when Ethernet IP Core is in the 100Mbps full duplex operating mode.

Control module consists of multiplexing logic and two sub modules:

- **eth_transmitcontrol**
- **eth_receivecontrol**

Flow control is done by sending and receiving **pause control frames**.

When the device that is connected to the WISHBONE interface of Ethernet IP Core (usually a processor) cannot process all those packets that it has received (and is still receiving), it requests a **pause** from the other station that is sending packets. The pause is requested by sending a pause control frame to the other station (see Ethernet IP Core Specification for details about the control frame). As soon as the other station receives pause request, it stops transmitting. The transmission is restarted after the requested pause time passes or pause request is switched off. The transmit flow control is done in the **eth_transmitcontrol** module. See description of the eth_transmitcontrol module for more details.

When the Ethernet IP Core receives a pause request, it stops transmitting for the requested time. This is done in the **eth_receivecontrol** module. See description of the eth_receivecontrol module for more details.

Multiplexing logic is used for multiplexing data and control signal used in normal transmission with data and control signals used for control frame transmission (see signals **TxUsedDataOut, TxAbortOut, TxDoneOut, TxEndFrmOut, TxStartFrmOut**).

When control frames are sent, padding and CRC generation is automatically switched on (see **PadOut** and **CrcEnOut** signals).

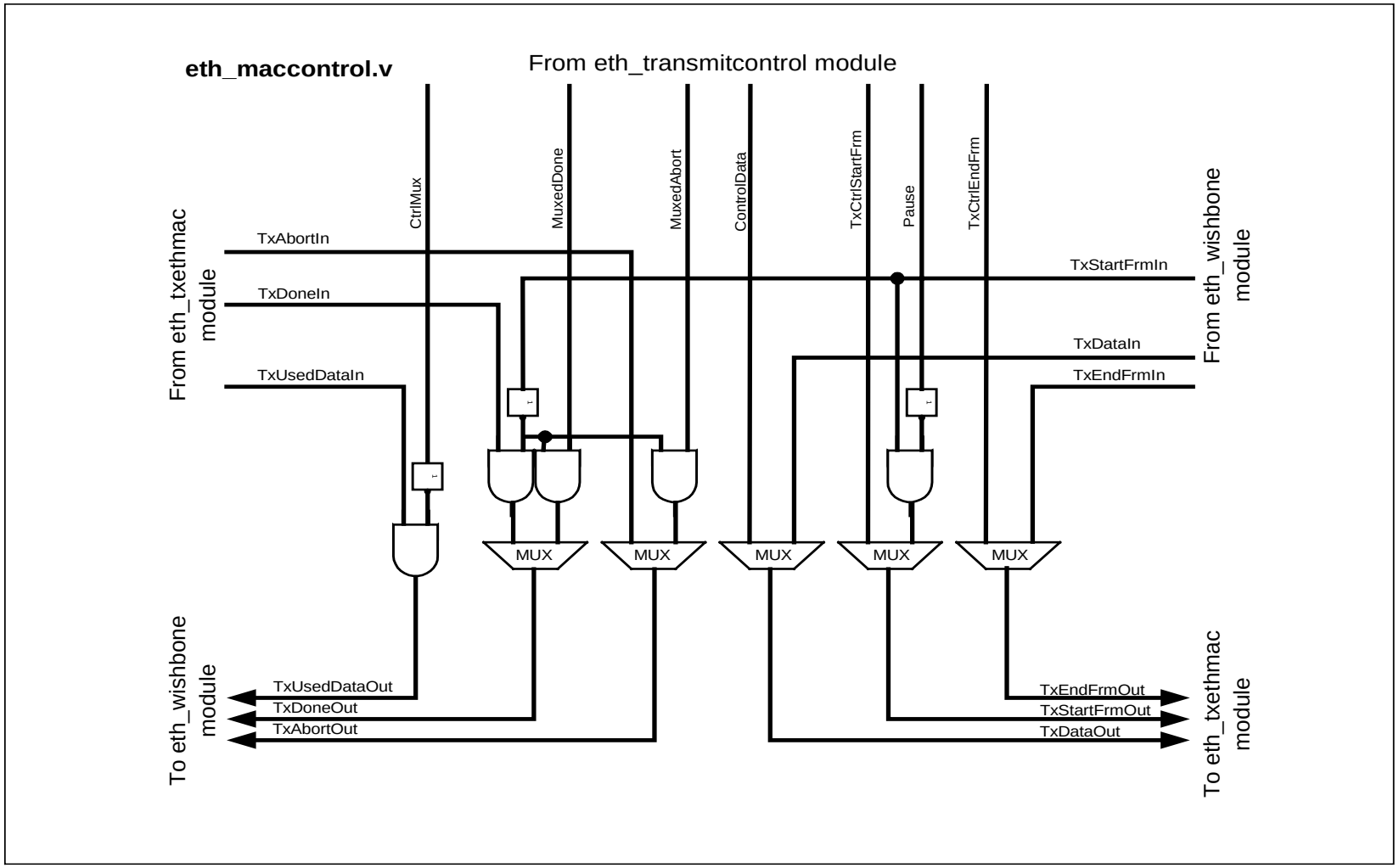

Figure 3: Multiplexing Data and Control Signals in Control Module

2.3.4.1 Description of the CRC (Cyclic Redundancy Check) module (eth_crc.v)

This module is used for CRC calculation. The calculated CRC is appended to the

2.3.5 Description of the Status module (eth_macstatus.v)

The Status module is in charge for monitoring the Ethernet MAC operations. Module monitors several conditions and after every completed operation (received or sent frame), it writes a status to the related buffer descriptor. Not all statuses are written to the buffer descriptors. See following sections for more details.

Statuses for received frames are normally latched at the end of the reception stage (when signal TakeSample goes to 1). Soon after that statuses are reset (when signal LoadRxStatus goes to 1).

2.3.5.1 Rx Error (LatchedMRxErr)

This error notifies that the PHY detected an error while receiving a frame. In this case frame reception is aborted and no error reported. When invalid symbol is accepted, frame is still received and invalid symbol error reported in the Rx BD.

2.3.5.2 Rx CRC Error (LatchedCrcError)

This error notifies that a frame with invalid CRC was received. Such frame is normally received except that the CRC error status is set in the related Rx BD. If received frame is a control frame (pause frame), then pause timer value is not set.

2.3.5.3 Rx Invalid Symbol (InvalidSymbol)

This error notifies that a frame with invalid symbol was received. Invalid symbol is reported by the PHY when it is operating in the 100 Mbps mode (PHY sets data lines to 0xe when symbol error is detected).

2.3.5.4 Rx Late Collision (RxLateCollision)

When a late collision occurs, frame is normally received and late collision reported in the Rx BD. Late collision reflects the abnormal operation on the Ethernet (should never happen). See COLLCONF register in the Ethernet IP Core Specification for more details about the late collision.

2.3.5.5 Rx Short Frame (ShortFrame)

Short frames are normally (by default) aborted. This means that their appearance is not recorded anywhere. However if their reception is enabled (by setting the RECSMALL bit in the MODER register to 1), then the SF bit is set to 1 in the Rx BD when a short frame appears. Minimum length is defined in the PACKETLEN register in the Ethernet IP Core Specification.

2.3.5.6 Rx Big Frames (ReceivedPacketTooBig)

By default the reception of the big frames is switched off. If frame that is bigger than the maximum frame specified in the PACKETLEN register (See the Ethernet IP Core Specification) is received, then frame reception is automatically stopped at the maximum value (no big frame status is written anywhere). If reception of the big frames is enabled (See HUGEN bit in the MODER register in the Ethernet IP Core Specification), then the TL bit is set in the Rx BD when packet bigger then the maximum size is received.

2.3.5.7 Rx Dribble Nibble (DribbleNibble)

DN bit is set in the Rx BD when an extra nibble is received as a part of the frame (frame is not byte aligned). CRC error occurs at the same time, so both errors are simultaneously reported.

2.3.5.8 Tx Retry Count (RetryCntLatched)

After every frame is transmitted the number of retries is written to the RTRY field of the Tx BD. The retry count gives information about that how many times transmitter retried before successfully transmitting a frame.

2.3.5.9 Tx Retry Limit (RetryLimit)

When a number of retransmission attempts is bigger then specified in the COLLCONF register (see Ethernet IP Core Specification), frame transmission is aborted and bit RL is set in the Tx BD.

2.3.5.10 Tx Late Collision (LateCollLatched)

Late collision should never occur. If it occurs during the frame transmission, the transmission is aborted and LC status is written to the associated Tx BD. See COLLVALID field of the COLLCONF register (Ethernet IP Core Specification) for more information on late collision.

2.3.5.11 Tx Defer (DeferLatched)

When frame was deferred before being sent successfully (i.e. the transmitter had to wait for Carrier Sense before sending because the line was busy), the DF bit is set in the associated Tx BD. This is not a collision indication. Collisions are indicated in RTRY.

2.3.5.12 Tx Carrier Sense Lost (CarrierSenseLost)

When Carrier Sense is lost during a frame transmission, bit CS is set in the associated Tx BD. Status is written after the frame is sent.

Following statuses are not part of the Status Module. They are generated in the Wishbone module and used in the Tx and Rx BD.

2.3.5.13 Tx Underrun (UnderRun)

Underrun is detected in the WISHBONE module and reported in the Tx BD after frame transmission is aborted due to the underrun. This means that the host was not able to provide data is being transmitted on time. This is not a normal condition and should never happen.

2.3.5.14 Rx Overrun (OverRun)

Overrun is detected in the WISHBONE module and reported in the Rx BD. When Overrun status is set, It means that the host was not able to store received data to the memory on time and Rx FIFO overrun happened. Some of the data was lost.

2.3.5.15 Rx Miss (Miss)

When Ethernet MAC is configured to accept all frames regardless of their destination address (PRO bit is set in the MODER register (see Ethernet IP Core Specification)), MISS bit tells if a received frame contains a valid address or not.

Additionally following signals are generated in the status module:

- **ReceivedLengthOK** reports when the received frame has a valid length
- **ReceiveEnd** reports the end of the reception. This signal is used in the control module for resetting several flip-flops and setting the pause timer.

2.3.6 Description of the Registers module (eth_registers.v)

Functionality of registers is described in the Ethernet IP Core Specification.

Although all registers are described as 32-bit registers, only the actually needed width is used. Other bits are fixed to zero (ignored on write and read as zero). Each register is instantiated with two parameters, width and reset value. Reset value defines whether register clears its value to zero or set to some predefined value after the reset.

2.3.6.1 Description of the eth_register module (eth_register.v)

This module contains one single register. The width of the register and its reset value are defined with two parameters:

- WIDTH
- RESET_VALUE.

2.3.7 Description of the WISHBONE interface module (eth_wishbone.v)

Module has multiple functions:

- It is the interface between the Ethernet Core and other devices (memory, host). Two WISHBONE interfaces (slave and master) are used for this manner.
- Contains buffer descriptors (in the internal RAM).
- Contains receive and transmit FIFO.
- Contains synchronization logic for signals that spread through different clock domains.
- Transmit related function that reads TX BD and then starts WISHBONE master interface, fills the TX FIFO and then starts the transmission. At the end it writes status to the related TX BD.
- Receive related function that reads RX BD, assembles incoming bytes to words and then writes them to the RX FIFO. They are then written to the memory through the WISHBONE master interface. At the end it writes status to the related RX BD.

2.3.7.1 WISHBONE Slave Interface

Ethernet registers and buffer descriptors (BD) are all accessed through the same WISHBONE Slave Interface. Registers are located in the eth_registers module, while BDs are saved in the internal RAM within the eth_wishbone module. Selection between registers and BD accesses is done in the eth_top module. This means that all accesses that reach eth_wishbone module are meant for buffer descriptors (See following Buffer Descriptor section for more details). All output signals (from slave WISHBONE interface) can be registered or not. Selection is done with ETH_REGISTERED_OUTPUTS define in the eth_defines.v file.

2.3.7.2 WISHBONE Master Interface

The Ethernet core uses WISHBONE **master** interface for accessing the memory space where the buffers (data) are stored. Both, the receiver and the transmitter access data through the same WISHBONE master interface. For this purposes a state machine is build. The state machine multiplexes access from TX and RX modules (See MasterWbTX and MasterWbRX signals). Following signals are used in the state machine:

- MasterWbTX
- MasterWbRX
- ReadTxDataFromMemory_2
- WriteRxDataToMemory
- MasterAccessFinished
- cyc_cleared

When a Receiver receives data from the Ethernet and needs to store it to the memory, it asserts the **WriteRxDataToMemory** signal. Write access can start immediately or is delayed (depending if another access is already in progress, the type of the previous access and number of requested accesses). **MasterWbRX** is set to 1 when receiver uses the WISHBONE bus.

When a Transmitter needs to send data, it reads the data from the memory. **ReadTxDataFromMemory_2** is asserted when transmitter needs data from the memory. Read access can start immediately or is delayed (depending if another access is already in progress, the type of the previous access and number of

requested accesses). **MasterWbTX** is set to 1 when transmitter uses the WISHBONE bus.

Every WISHBONE access is finished when slave asserts acknowledge or error signal. Both signals are joined together in **MasterAccessFinished** signal.

After every access, **m_wb_cyc_o** signal must be cleared to zero because of the traffic COP limitations. When there are two consecutive single accesses performed one after another, state machine goes to the “temporary idle” state where signal **cyc_cleared** is set and **m_wb_cyc_o** cleared to zero. After that a normal read or write operation starts. At the moment only single accesses are supported (block or burst accesses are not supported).

Accesses to/from addresses that are not word-aligned are supported.

When transmitter needs to send data that is stored in the memory at non-aligned address, following procedure is used:

Pointer to the TX buffer is stored to three different registers: **TxPointerMSB**, **TxPointerLSB** and **TxPointerLSB_rst**. TxPointerMSB is used for accessing the word-aligned memory. After every WISHBONE access TxPointerMSB is incremented and points to the next word in the memory. TxPointerLSB bits remain unchanged during the whole operation of packet sending. Since word accesses are performed, valid data does not necessarily start at byte 0 (could be byte 0, 1, 2 or 3). TxPointerLSB is used only at the beginning (when accessing the first data word) for proper selection of the start byte (TxData and TxByteCnt signals depend on it). After the read access, TxLength needs to be decremented for the number of the valid bytes (1 to 4). After the first read all bytes are valid so this two bits are reset to zero. For this reason TxPointerLSB_rst is used. This signal is the same as TxPointerLSB except that it resets to zero after the first read access.

When receiver need to store data to the memory at word-unaligned address, the following procedure is used:

Buffer descriptor pointer is stored to two different registers: **RxPointerMSB** and **RxPointerLSB_rst**. Accesses are always performed to word-aligned locations. For that reason the RxPointerMSB with two LSB bits fixed to zero are used. Byte select signals (**RxByteSel**) are used for solving the alignment problem. (I.e. If RxPointer is 0x1233, then word access to 0x1230 is performed and RxByteSel is set to 0x1). RxPointerLSB_rst signal is used for **RxByteSel**, **RxByteCnt**, **RxValidBytes** and **RxDataLatched1** signals generation. RxByteSel is used as byte select signal when writing data to the memory through the wishbone

interface. After the first write access, RxPointerLSB_rst is reset to zero and all byte selects (RxByteSel) become valid (only word accesses are performed).

RxByteCnt counts bytes within the word. It is used for proper latching of the input data, setting the conditions when to write data to the RX FIFO and to mark when the last byte is received through the Ethernet. **RxValidBytes** marks how many bytes are valid within the last word that is written to the memory.

Note: Even when not all bytes are valid when writing the last word to the memory, full word is written (invalid bytes are written as zeros).

2.3.7.3 Tx and Rx Buffer Descriptors

Buffer descriptors are located in the internal RAM at addresses between 0x400 and 0x7ff. Each BD is 8 bytes long (4 bytes for status and 4 bytes for pointer). Access to buffer descriptors is only possible when Ethernet MAC Controller is not in reset (See RST bit in the MODER register). As soon as the READY bit is set in the TX BD (READY bit in the RX BD), descriptor cannot be changed until transmitter clears that bit to zero (receiver). There are totally 128 buffer descriptors that can be used for both, transmit (TX) or receive (RX). Number of TX BD is defined in the TX_BD_NUM register. The rest are used for RX BD.

Example:

If value 0x32 is written in the TX_BD_NUM register, it means that there are 50 TX BD and 78 RX BD (128-50)).

Tx BDs are accessible between 0x400 and 0x58c (8 x 0x32 + 0x400 - 4).

Rx BDs are accessible between 0x590 (8 x 0x32 + 0x400) and 0x7fc.

For detailed description of the buffer descriptors, please read the Buffer Descriptors (BD) section of the Ethernet IP Core Specification.

Single port RAM is used for buffer descriptors (smaller). Three devices can access RAM:

- Host through the WISHBONE slave interface

- Transmitter
- Receiver

Smart access multiplexing is done with a state machine (see generation of the WbEn, RxEn and TxEn signals). Multiplexing depends on the **RxEn_needed** and **TxEn_needed** signals.

RxEn_needed informs the state machine that the receiver needs to access a buffer descriptor in the RAM (needs to write a status (after receiving a frame) to it or needs an empty buffer descriptor to start with the reception).

After the reset **RxBDRead** is set to 1 and **RxBDReady** is set to zero. This means that there is a need to read an empty buffer descriptor from the RAM (signal RxEn_needed is set to 1). A read cycle to the **RxBDAddress** is started. If a BD that is not mark as empty is read, the same procedure is repeated. As soon as a BD that is marked as empty (bit EMPTY set to 1) is read, a pointer related to the same BD is needed. Another read is performed to the address where pointer is stored (**RxBDAddress** + **RxPointerRead**). After that there is no need for receiver to read the BDs and signal RxEn_needed is cleared to zero with **RxPointerRead** signal. Reception of the frame starts automatically. When a frame is received, signal **ShiftEnded** is set to 1. This signal clears RxBDReady signal, which then sets RxEn_needed to 1. Status is written to the related receive BD, address is incremented and read to the next BD is started.

TxEn_needed tells to the state machine there is a need that transmitter accesses the buffer descriptors in RAM. Operation of the TX BD is very similar to the operation of the Rx BD. In this case used signals are **TxBDRead**, **TxBDReady**, **TxPointerRead**, **TxStatusWrite**.

2.3.7.4 Tx and Rx FIFO

Both, TX and RX sides have FIFO-s. Defines related to the FIFO-s are in the eth_defines.v file:

- TX_FIFO_CNT_WIDTH, TX_FIFO_DEPTH, TX_FIFO_DATA_WIDTH for TX FIFO
- RX_FIFO_CNT_WIDTH, RX_FIFO_DEPTH, RX_FIFO_DATA_WIDTH for RX FIFO

Currently both FIFO-s are 16-words deep.

After the TX BD is read (both status and pointer), data is read from the memory through the master Wishbone interface and stored to the TX FIFO. Actual transmission starts as soon as the TX FIFO is full (to keep the possibility of the underruns as low as possible). When there is space for at least one word in the FIFO, another read is performed.

After the RX BD is read (both status and pointer) and there is some incoming data in the FIFO (at least one word), write to the memory is immediately performed. Reception of the next frame is possible after all data is written to the memory (FIFO is empty).

2.3.7.5 Synchronization Logic

Typical approach was that at least two flip-flops were used when crossing different clock domains. Those signals that crossed clock domains and were available long time before it's actual use were not synchronized.

3

Ethernet MAC IP Core Testbench

3.1 Overview

Ethernet MAC IP Core testbench consists of a whole environment for testing Ethernet MAC IP Core, including Ethernet PHY model, WISHBONE bus models with bus monitors and test cases, which use those models to stimulate transactions through the Ethernet. Those transactions are checked in many different modes.

3.2 Testbench File Hierarchy

The hierarchy of modules in the Testbench of the Ethernet MAC IP Core is shown here with file tree. Each file here implements one module in a hierarchy. Source files of the Testbench are in the **ethernet\bench\verilog** subdirectory.

File list missing

3.2.1 Testbench Module Hierarchy

Module hierarchy is shown in detail in the following picture. Description of modules and their connections is in the chapter 3.3, Description of Testbench Modules.

Missing figure

3.3 Description of Testbench Modules

The module tb_ethernet.v is used as testing environment and it incorporates beside all test submodules, functions and tasks also Unit Under Test (Ethernet MAC IP Core). Description of tasks is covered in chapter Description of Testcases, while all test submodules are described in the following chapters.

3.3.1 Description of Ethernet PHY module

Ethernet PHY module simulates simplified Intel LXT971A PHY chip.

Ethernet PHY provides two clock signals to the Ethernet MAC Core: transmit clock (mtx_clk_o) and receive clock (mrx_clk_o). Depending on the control bits, TX and RX clock operate at 2.5 MHz for 10 Mbps operation or 25 MHz for 100 Mbps operation (only bit [13] is used for clock frequency setting). TX and RX clock signals are not synchronous. When Ethernet link is not up, RX clock has a random frequency between 2 MHz and 40 MHz.

PHY has an MIIM interface, which is connected to the Ethernet core. All transactions are monitored and every error/warning reported. Besides that PHY has several registers implemented in it (Control, Status and two Identification registers).

PHY provides carrier sense and collision signals. Both signals can be set through several tasks.

When transmitting data (PHY is receiving data), PHY controls the protocol (preamble, sfd, writes length and data to its memory).

When PHY sends data to the Ethernet MAC, it can generate various preambles (different length, wrong preamble). It takes data from its memory. Testbench needs to write data to PHY's memory before PHY can start with transmission.

3.3.2 Description of WB submodules

3.3.2.1 wb_bus_monitor submodule

The module **wb_bus_monitor.v** monitors the WB Bus and tries to see WB Protocol Errors. There are two point-to-point WB buses:

- WB master from Ethernet MAC IP Core that goes to the WB Slave Behavioral unit (for writing and reading data)
- WB slave from WB Master Behavioral unit to the Ethernet MAC IP Core (used for accessing registers and buffer descriptors)

There are also two WB bus monitors, one for each WB bus.

3.3.2.2 wb_master_behavioral submodule

The module **wb_master_behavioral.v** is used to initiates WB cycles to WB Slave in the Ethernet MAC IP Core. That is controlled by top-level. This module also includes a submodule **wb_master32.**v, which is used to generate proper WB cycles. The length and type of each cycle is controlled by **wb_master_behavioral.v** module. This module also incorporates a block of SRAM.

3.3.2.3 wb_slave_behavioral submodule

The module **wb_slave_behavioral.v** responds to cycles initiated by WB Master in the Ethernet MAC IP Core. When to respond and a type of cycle termination is controlled by top-level. This module also incorporates a block of SRAM.

3.4 Description of Testcases

Add some description to this section (like in PCI)

3.4.1 Description of MAC Registers and BD Tests

There are several tests to test the MAC Registers and Buffer Descriptors (**test_access_to_mac_reg**):

Following test cases are for testing Ethernet MAC internal registers:

- Walking 1 with single cycles across MAC registers.
- Test maximum register values and register values after writing inverse reset values and hard reset of the MAC.

Following test cases are for testing Ethernet MAC buffer descriptors:

- Walking 1 with single cycles across MAC buffer descriptors.
- Test buffer descriptors. RAM preserves values after hard reset of the MAC and resetting the logic.

3.4.2 Description of MIIM Module Tests

There are several tests for testing MII Management module:

- Test clock divider of MII management module with all possible frequencies.
- Test various readings from 'real' PHY registers.
- Test various writings to 'real' PHY registers (control and non-writable registers)
- Test reset PHY through MII management module
- Test 'walking one' across PHY address (with and without preamble)

- Test 'walking one' across PHY's register address (with and without preamble)
- Test 'walking one' across PHY's data (with and without preamble)
- Test reading from PHY with wrong PHY address (host reading high 'z' data)
- Test writing to PHY with wrong PHY address and reading from correct one
- Test sliding stop scan command immediately after read request (with and without preamble)
- Test sliding stop scan command immediately after write request (with and without preamble)
- Test BUSY and NVALID status durations during write (with and without preamble)
- Test BUSY and NVALID status durations during write (with and without preamble)
- Test BUSY and NVAILD status durations during scan (with and without preamble)
- Test scan status from PHY with detecting LINKFAIL bit (with and without preamble)
- Test scan status from PHY with sliding LINKFAIL bit (with and without preamble)
- Test sliding stop scan command immediately after scan request (with and without preamble)
- Test sliding stop scan command after 2nd scan (with and without preamble)