Tri-Mode Ethernet MAC v0.1

Jared Casper
FOSSH

November 3, 2009

1 Introduction

FOSSH's Tri-mode Ethernet MAC core was designed from the ground up to be a compact, fast, no-frills core to faciliate streaming data from a widget to a connected computer. Because the core is meant to be small and fast, many features that are not often used or would not be useful for streaming data are left out. For example, the core only supports full-duplex operation in that it does not even look at the CRS (carrier-sense) and COL (collision detection) signals from the PHY. That said, it does have features not present in other cores to make it easy to stream data. For example, it will generate the ethernet and udp headers for you so you can simple give it data and it will take care of sending it for you.

The core is in active development and this document describes version 0.1.

1.1 Features

Features that are implemented and functional:

- Interfaces with the PHY using GMII running 10/100/1000 Mb/s. Support for RGMII is planned.
- Full-duplex support only.
- Raw ethernet frame generation, allowing a pure data input stream.
- Optional UDP/IP packet generation with automatic packetization and framentation of an input data stream.
- Input and output fifos decoupled from the core logic, so the clock rate of application logic is not tied to the speed of the ethernet link (barring bandwidth issues).
- Full core uses 288 Spartan 6 logic slices and 2 block RAMs. For comparison, the smallest Spartan 6 has 600 slices and 12 block RAMs. The largest Spartan 6 has 23,038 slices and 268 block RAMs.

Features that are in development or planned:

- Multiple input/output fifo for streams to be sent to/from different addresses.
- Optional configuration registers instead of configuration ports for use with a micro controller.
- Automatic flow control via PAUSE frames.
- RGMII interface.

1.2 Obtaining

Stable releases of the MAC core can be downloaded from http://www.fossh.com/. Read-only public access to the development repository is available using git at the respository git://jcasper.stanford.edu/fossh/repo.git.

2 Interface

2.1 Parameters

Name	Type	Description
USR_CLK_PERIOD	Integer	Period, in ns, of usr_clk. Needed to to generate a sutable clock for the mii management port (mdc) from usr_clk.
RX_CRC_CHECK	Boolean	If true, the CRC of incoming packets will be checked for accuracy. If false, no such check is performed.

2.2 Clocks, etc.

Name	Width	Direction	Description
usr_clk	1	I	User clock used to clock data into and out of the core through the user and management interfaces.
clk_125	1	I	125 MHz clock used to supply phy_GTX_CLK.
reset_n	1	I	Active-low reset. Resets all flops and buffers in the core.
debug	*	O	Various signals used during debugging.

2.3 User Interface

All signals in the user interface are synchronous with usr_clk.

2.3.1 Configuration

Instead of implementing internal registers to indicate the source and destination MAC address, these values are simply ports into the core. This allowed the implementation to be expanded to a wider variety of future interfaces. An optional wrapper is in development that will add registers to the management interface (Section 2.3.4) and remove these ports.

The following ports are part of the base MAC module and are always used:

Name	Width	Direction	Description
gmii	1	I	Indicate that the core should be communicate with the phy in GMII mode (i.e. 1Gbit/s). Otherwise MII mode (10/100Mbit/s) will be used. Unfortunately there doesn't appear to be a standard way to get the negotiated link speed from the PHY (why is this not in the 802.3 standard is beyond me).
promiscous	1	I	Indicate that the core should filter incoming packages to those with destination of src_mac_addr.
jumboframes	1	I	Indicates that jumbo frames are okay to send.
src_mac_addr	48	I	MAC address to use as the source of transmitted ethernet frames and the filter for what received frames to not discard when not in promiscous mode.
dst_mac_addr	48	I	MAC address to use as the destination of transmitted ethernet frames.
ethertype	16	I	EtherType/Length field for outgoing packets.
ifg	10	I	Number of cycles inserted between packets. Should be $\geq$ 0x0D.

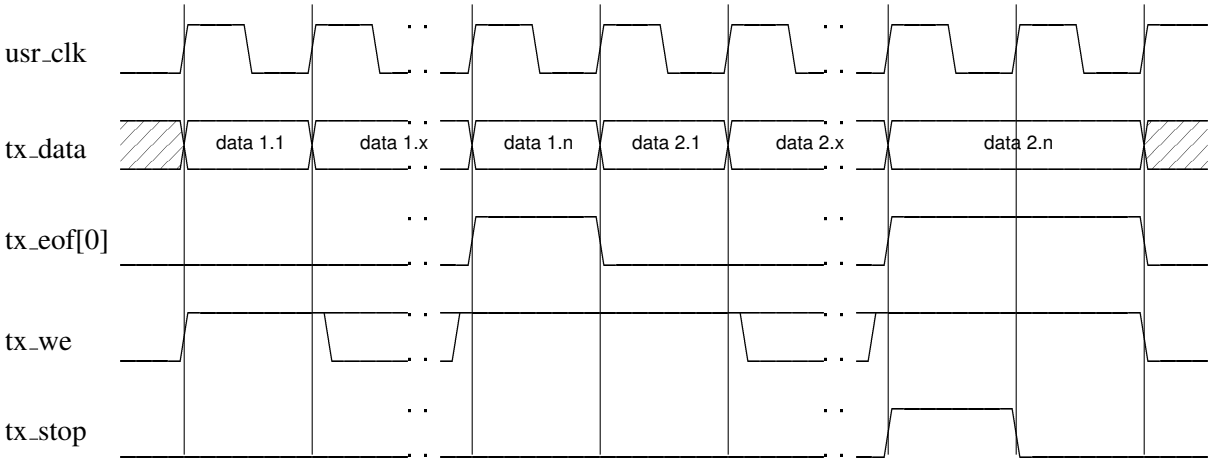

Figure 1: Timing diagram of two back to back 32-bit aligned packet transmissions, with the final word of the second packet being interrupted by a full input fifo.

2.3.2 Transmission

The interface used to send packets/frames is a standard FIFO write interface with addition of an end-of-frame port (tx_eof) that is used to split incoming data into frames.

Name	Width	Direction	Description
tx_data	32	I	Data to be sent to the phy for transmission. Data is queued to be sent when tx_we is high and tx_stop is low. Following ethernet standards, bytes are sent in big endian order
tx_eof	4	I	Used to indicate which bytes in the word being written (tx_data) is the last of the frame. Because data is sent big endian, setting tx_eof[0] to 1 indicates that the entire 32-bit should be sent (and is the last word in this packet).
tx_we	1	I	Write enable for transmission. Queues tx_data to be sent to the phy when tx_stop is low.
tx_stop	1	O	Indicates that the input buffers are full and no more data can be queued for transmission.

Figure 1 illustrates normal transmission of a couple of 32-bit aligned packets, with the final word of the second packet being temporarily stopped by a full input fifo.

If the input fifo runs out of data while the MAC is sending a packet, an error condition will be raised and the packet will not be sent. Operating at 1Gbps, the PHY will transmit approximately 100MB/s from the input fifo. Therefore, if data word is supplied every clock, the user clock must be at least 25 MHz, an easy target for most FPGAs. Alternatively, if the user clock is faster, data need not be supplied on each clock. For example, a 100 MHz user clock means that once data is first written to the input fifo, the rest of the data can come at a rate as slow as one word every 4 clocks.

It is important to note that bytes in tx_data are sent in big endian mode, following ethernet standards.

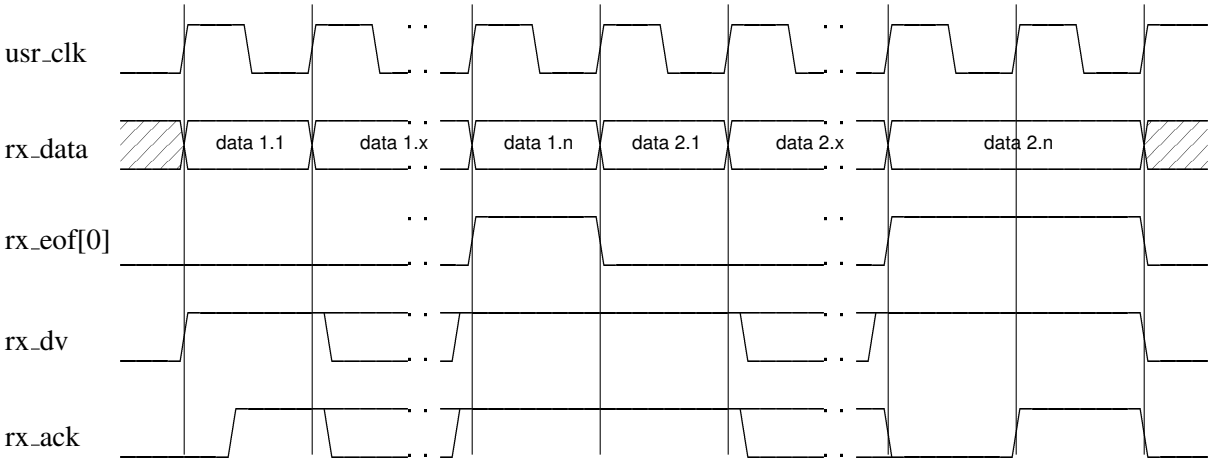

Figure 2: Timing diagram of receiving two back to back 32-bit aligned packets, with the final word of the second packet being interrupted by the user.

This means tx_data[31:16] is sent first, followed by tx_data[15:8], and so on. Care should be taken if value smaller than 32 bits are written and their ordering.

2.3.3 Reception

The interface used to receive packets is a standard FIFO read interface, with the addition of an end-of-frame port (rx_eof).

Name	Width	Direction	Description
rx_data	32	O	Data received by the MAC from the phy. Valid when rx_dv is high.
rx_eof	4	O	Used to indicate that rx_data is the last word in the received frame. Mapped to rx_data just as tx_eof maps to tx_data.
rx_dv	1	O	Indicates that rx_data is valid data that was received by the MAC.
rx_ack	1	I	Acknowledges the receipt of rx_data and causes the MAC to move to the next received word.

Figure 2 illustrating the timing of receiving a packet. If the user logic does not empty the reception fifo fast enough, it will fill up and the the MAC will drop incoming frames.

2.3.4 Management

The following ports are used to read and write the MII management registers on the PHY. If they are left unconnected, the MII management module will be optimized away by the synthesis tools.

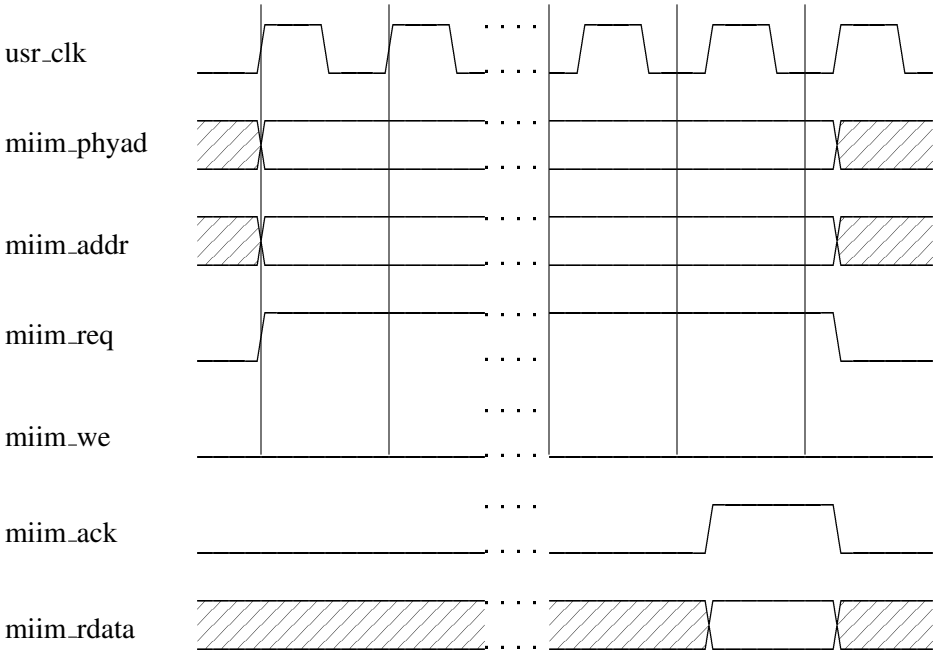

Figure 3: Timing diagram of MII management read.

Name	Width	Direction	Description
miim_phyad	5	I	Management address of the PHY to communicate with.
miim_addr	5	I	Address of PHY register to read/write to.
miim_wdata	16	I	Data to write to the PHY register.
miim_we	1	I	Write Enable, write miim_wdata to the PHY register miim_addr.
miim_rdata	16	O	Data read from the PHY register.
miim_req	1	I	Sends a read request to the PHY for miim_addr.
miim_ack	1	O	Indicates that the requested operation (read or write) has been performed. If a read was requested, miim_rdata contains the data read from the PHY.

The management interface is a simple memory-style read/write interface. To read a management register, put the address on `miim_phyad` and `miim_addr` and hold `miim_req` high until `miim_ack` is raised. `miim_rdata` will hold the read data the cycle `miim_ack` is raised. This is shown in Figure 3.

To perform a write to the management register, hold `miim_phyad`, `miim_addr`, and `miim_wdata` valid and `miim_we` and `miim_req` high until `miim_ack` is asserted, at which point the write has been performed. This is illustrated in Figure 4.

2.4 UDP/IP Wrapper

When the UDP/IP Wrapper is used, the ethertype port is removed and the following configuration ports are added.

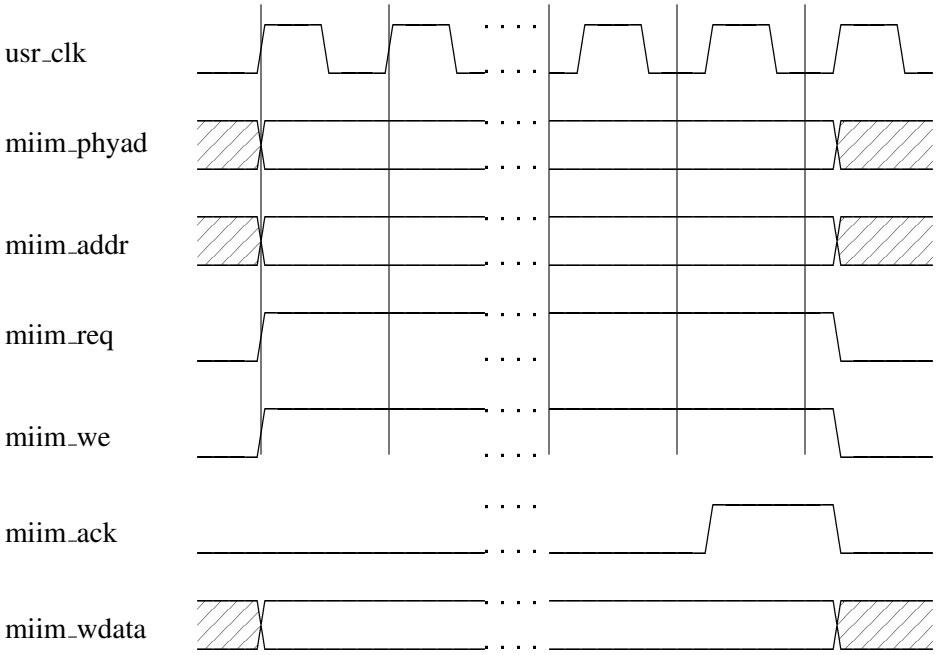

Figure 4: Timing diagram of MII management write.

Name	Width	Direction	Description
src_ip	32	I	IP address to use as the source of trasmitted UDP/IP packets.
src_port	16	I	UDP port to use as the source of transmitted UDP/IP packets.
dst_ip	32	I	IP address to send UDP/IP packets to.
dst_port	16	I	UDP port to send UDP/IP packets to.
ttl	8	I	Value for the initial Time-To-Live field of UDP/IP packets.

The transmission, reception, and management ports are all the same, except instead of a four bit tx_eof, there is a single bit tx_eop, because the UDP wrapper currently does not support sending sub 32-bit aligned packets.

2.5 PHY

The PHY ports are the standard GMII ports that should be tied directly to pins connected to an ethernet PHY.

Name	Width	Direction	Description
phy_TXD	8	O	These pins are the standard MII/GMII interface pins described numerous other places.
phy_TX_EN	1	O	
phy_TX_ER	1	O	
phy_TX_CLK	1	I	
phy_GTX_CLK	1	O	
phy_RXD	8	I	
phy_RX_DV	1	I	
phy_RX_ER	1	I	
phy_RX_CLK	1	I	
phy_MDC	1	O	
phy_MDIO	1	IO	

3 Architecture

The MAC core is split up into three main components: tranmission, reception, and reconciliation. Figureřeffig:mac-blocks shows a block diagram of how these components work together. The transmission blocks, **txfifo**, **tx_engine**, and **crc_gen**, are responsible for taking user data and sending it to the reconciliation module, which sends it to the ethernet phy. Likewise the recption blocks, **rx_usr_if**, **rx_pkt_fifo**, **rxfifo**, **rx_engine**, and **crc_chk**, are responsible for taking data from the reconciliation layer, throwing away bad frames, and presenting it to the user. The reconciliation layer handles with the GMII interface to the phy and deals with differences between 10/100Mbit operation and gigabit operation.

I have tried to make the verilog as readable and nicely commented as possible, so hopefully the code is self documenting. This section just gives an overview of how things work.

3.1 Reconciliation

The reconcilation modules provides a common interface to the rest of the core regardless of the speed of the ethernet link. This is done by providing internal clocks to both the tx (`int_tx_clk`) and rx (`int_rx_clk`) modules which can be used to send data to the reconciliation layer eight bits a clock.

If `gmii` is high we are operating at gigabit speeds and the phy sends and receives a full eight bits of data each 8ns. For transmission, `clk_125` is used as both `int_tx_clk` and sent as `phy_GTXCLK`, synchronizing all tranmission to that one 125 MHz clock. Likewise, all reception is synchronized to the 125 MHz `phy_RXCLK`, which is sent as `int_rx_clk`.

if `gmii` is low we are operating at 10 or 100 Mbits per second and only four bits of the databus to the phy is used. For transmission, the phy expects four bits every cycle of the phy supplied `phy_TXCLK`; to maintain an eight bit interface with the rest of the mac, `phy_TXCLK` is divided in half and sent as `int_tx_clk`. Likewise, the phy sends four bits every `phy_RXCLK`, so this is divided in half and sent as `int_rx_clk`.

3.2 Transmission

For transmission, **tx_engine** runs a simple state machine which controls what is sent to the reconciliation layer, pulling data from the configuration ports for the header, **txfifo**, a 36 bit wide asynchronous fifo, for the payload, and **crc_gen** for the final CRC checksum. **txfifo** is 36 bits wide to hold both the data and the corresponding end of frame bits.

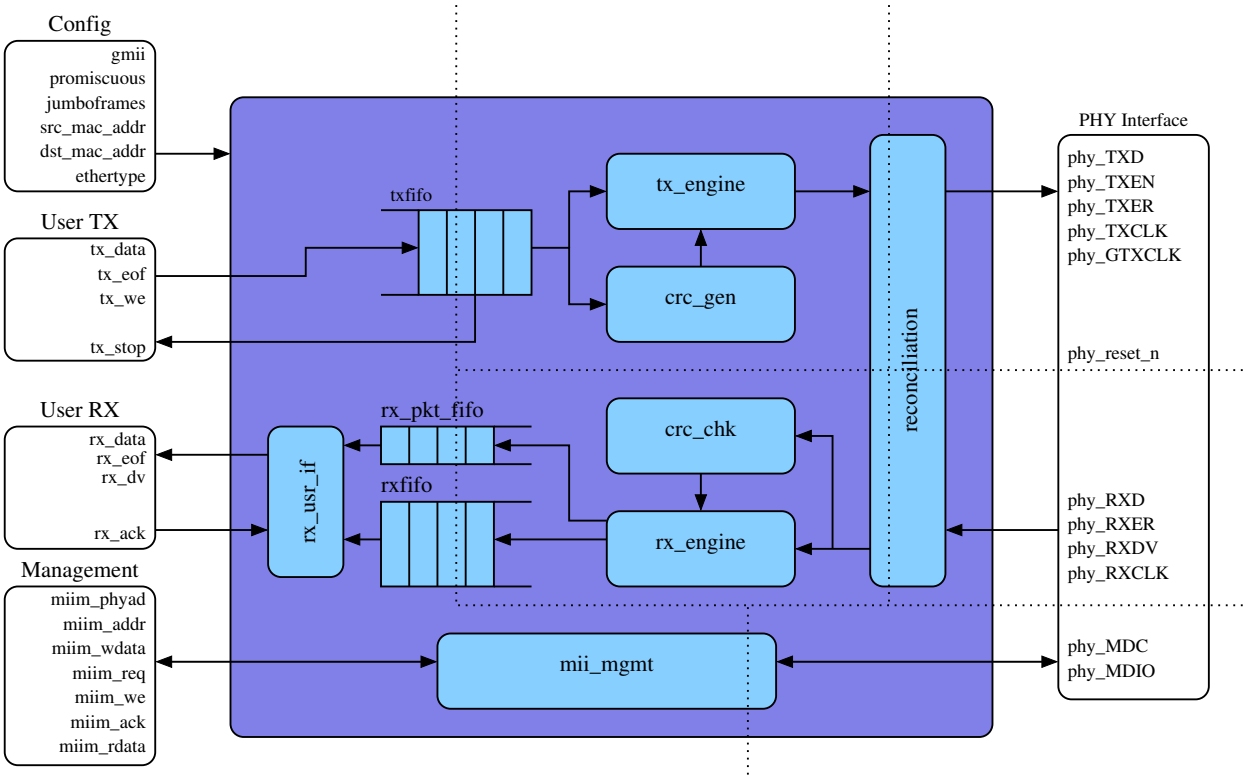

Figure 5: Block diagram of the MAC core.

If **txfifo** runs out of data, **tx_engine** raises `phy_TXER` to drop the packet being sent and continues to drain **txfifo** until an end-of-frame is indicated. It then moves on to the next packet in **txfifo**.

A frame being sent will also be dropped if the frame ends too early or too late. A frame must be 46 bytes long to be sent. The `jumobframes` input changes when **tx_engine** thinks a frame as grown too large and should be dropped. If `jumboframes` is low, the frame will be dropped if larger than 1500 bytes, otherwise it will be dropped if larger than 9000. Note that the phy may or may not accept frames larger than the standard 1500 bytes, but the MAC has no way of knowing this.

Once a packet has successfully been sent, **tx_engine** wait for an inter frame gap, set by the `IFG` parameter to the **tx_engine** module, which defaults to 12 cycles and begin sending the next frame if it is available.

3.3 Reception

Reception is controlled by both **rx_engine** and **rx_usr_if**. **rx_engine** receives data from the phy and passes all the data unaltered, except for the preamble, to the **rxfifo**, once a full packet is received it writes a '1' to the single bit asynchronous fifo **rx_pkt_fifo** to indicate to **rx_usr_if** that the data in **rxfifo** is ready to send to the user. If an error is detected during packet reception, either because `phy_RXER` was asserted, the frame was too short or too long, or the destination MAC address didn't match and `promiscuous` was not asserted, then a '0' is written to the **rx_pkt_fifo** to indicate to **rx_usr_if** that a bad packet is in **rxfifo**. **rx_usr_if** will then drain the bad packet from **rxfifo** without ever presenting it to the user.

3.4 Management

Implementation of the management interface is just a straightforward state machine which performs the requested read and write operations using the MII managment protocol. See section 22.2.2 of IEEE 802.3-2008, or any phy's datasheet, for a description of the protocol.

3.5 Clocking

There are six different clock domains within the MAC core, delineated by the dotted lines in Figure 5.

- `usr_clk` This is the only clock that the user of the MAC needs to be aware of, as all user signals into the MAC are synchronized to this clock. It can be any speed as long as it is fast enough as detailed in Section 2.3.2.
- `int_tx_clk` This clock is used by all of the transmission modules and is generated by the reconciliation layer depending on the speed of the ethernet link. It is either `clk_125` or half of `phy_TXCLK`.
- `int_rx_clk` The reception version of `int_tx_clk`, also generated by the reconiliation layer. It is either `phy_RXCLK` or half of `phy_RXCLK`.
- `phy_TXCLK/phy_GTXCLK` This is used to clock data out of the FPGA to the phy. `phy_TXCLK` is supplied by the phy and `phy_GTXCLK` is `clk_125`.
- `phy_RXCLK` This is used to clock data into the FPGA from the phy and is supplied by the phy.
- phy_MDC This is used to clock data in and out of the phy's management port. It is generated from `usr_clk` based on the `USR_CLK_PERIOD` parameter to the MAC core. It should have a period greater than 400ns as per section 22.2.2.11 of IEEE 802.3-2008.