WEB SERVER
FINAL REPORT

CSEE4840 Embedded System Design
Prof. Steven A. Edwards

Franklin Ma
Howard Wang
Victor Wang
William Wong

TABLE OF CONTENTS

OVERVIEW

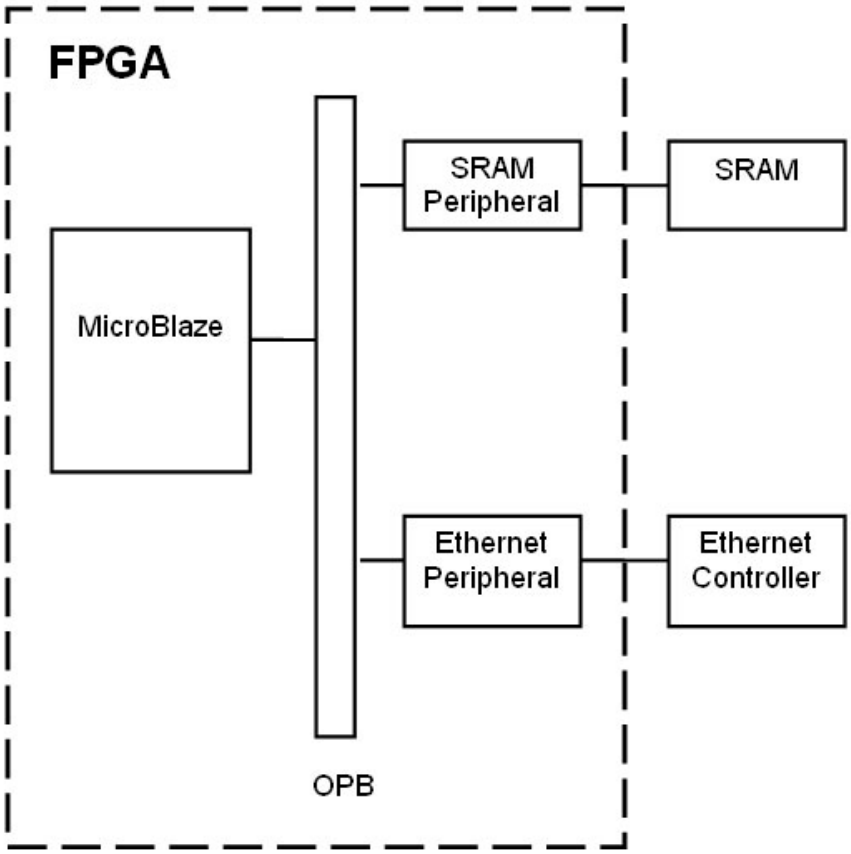
Our aim is to implement a web server on the XESS XSB-300E that will accept requests from clients for data through HTTP over TCP/IP through the onboard Ethernet controller. We will design the VHDL modules that interface with the SRAM, where we will likely store the data that is to be transmitted to the client (unless there is sufficient on-chip memory), and the on-board Ethernet controller. The following protocols will be necessary for our web server:

HTTP	web page request/response
TCP	reliable communications
IP	low-level data transport
ICMP *	diagnostics (Ping)
ARP	resolve IP address into MAC address

We will be programming these protocols in low-level C to be carried out by MicroBlaze.

* ICMP will be used for testing purposes and will be removed to decrease the instruction size if it is possible to fit our code into the BRAM.

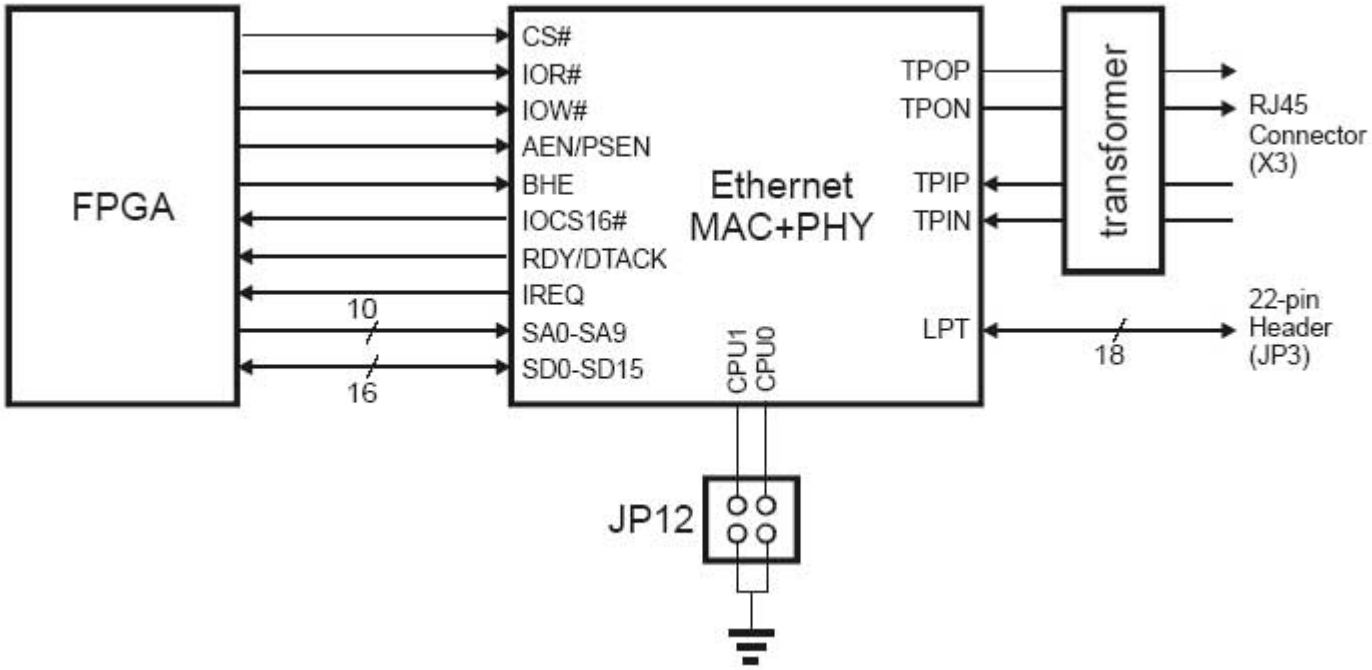
Block Diagram – General Archtecture:



ETHERNET

We will be using the AX88796 Ethernet Controller and communicating with it through the custom peripheral on the OPB. The Ethernet Controller interface is outlined below:

AX88796 Ethernet Controller



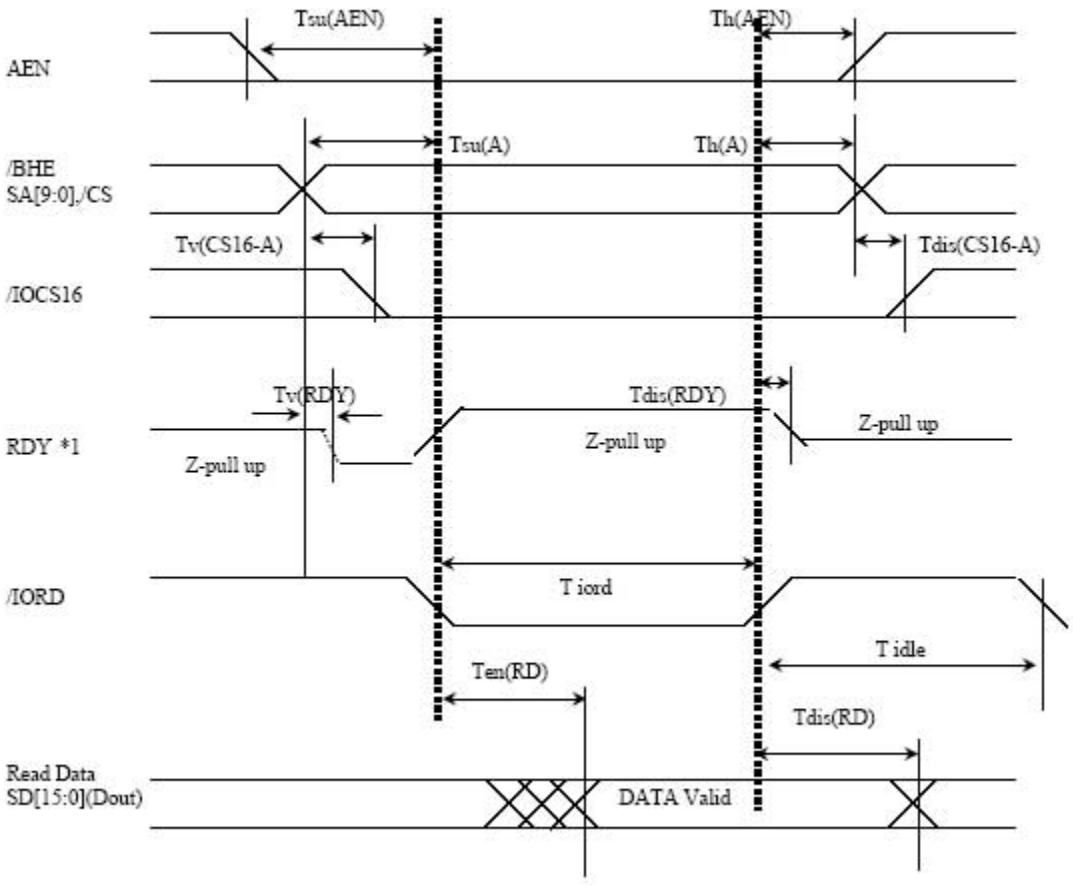
Interfacing with the Controller

In general, we will drive the Ethernet Controller by simulating read and write cycles on an ISA bus as follows:

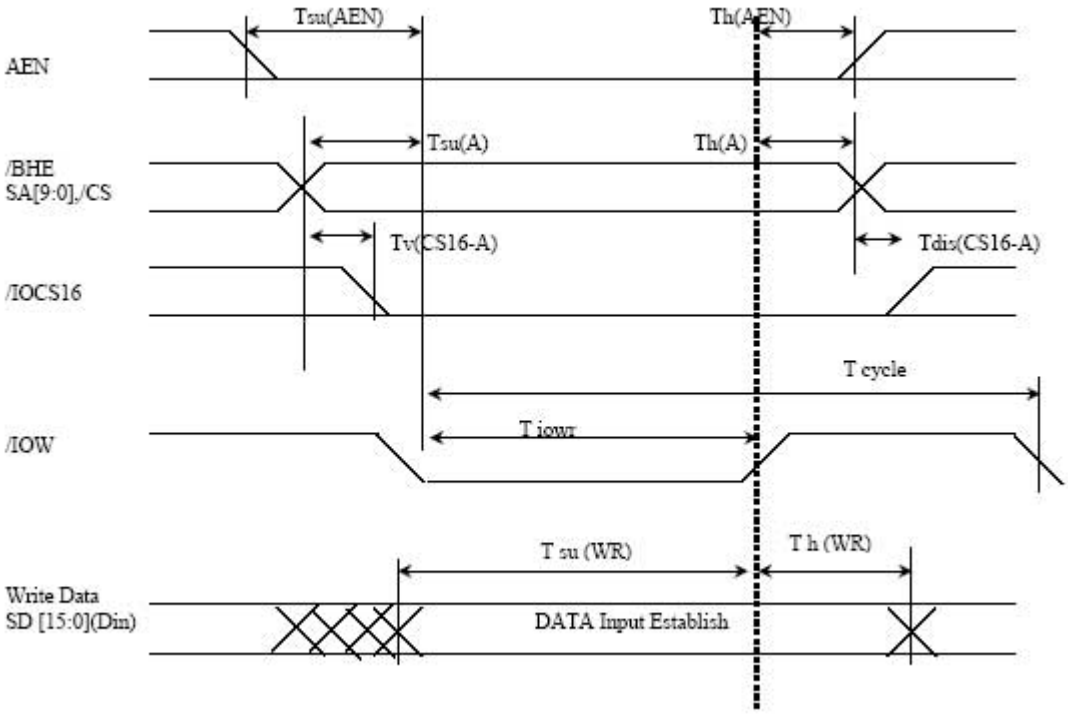
• Set the address lines.
• During the read cycle, the data lines from the MicroBlaze will serve as inputs. During a write cycle, they will be outputs and be set with the data to be written.
• Set the appropriate read or write signals (active low).
• If it is a read cycle, the Ethernet controller will drive the bus with the appropriate data.
• Deassert the data if it's a read and latch the data received from the MicroBlaze if it's a write.
• Disable output drivers in order to free up data bus.

Timing Diagram for the Ethernet Controller

Read Cycle:

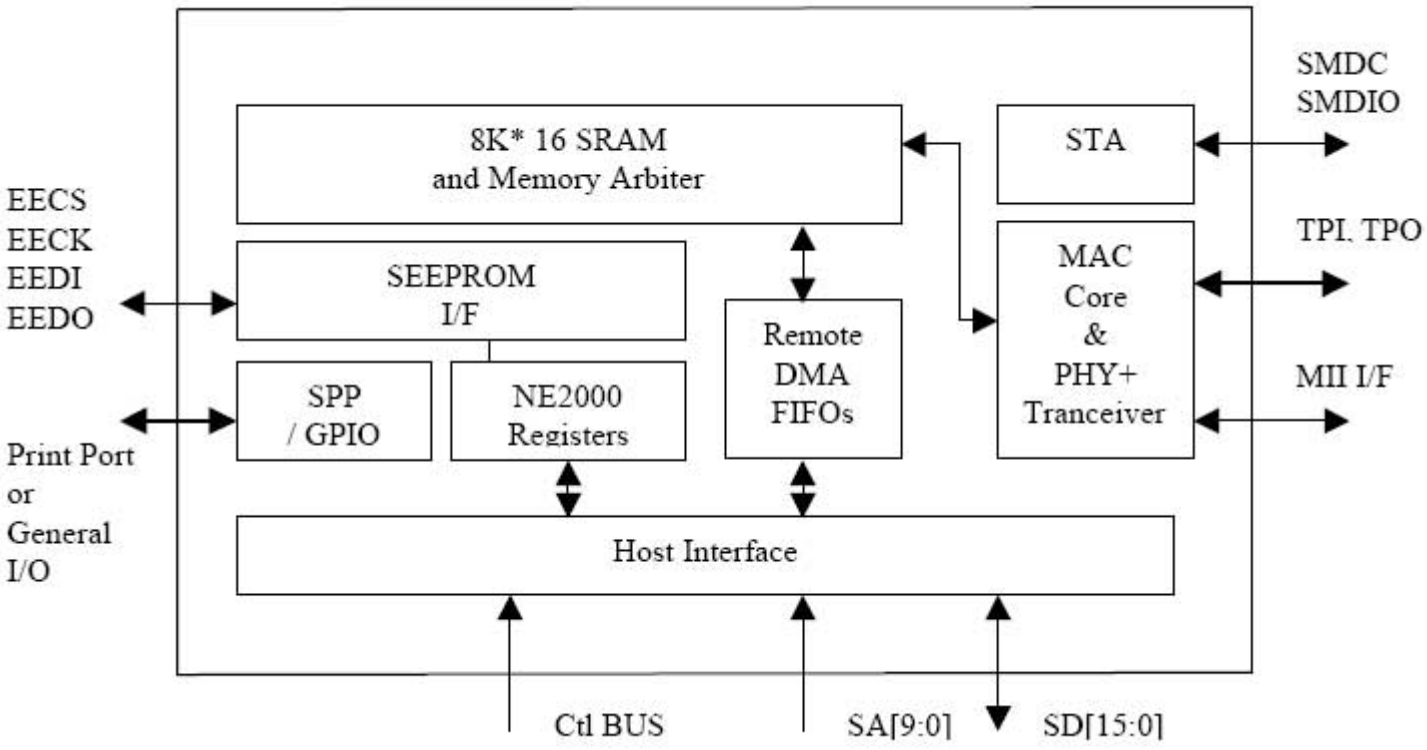


Write Cycle:



Here is the block diagram of the AX88796:

AX88796 Block Diagram



Packets will be constructed and transmitted according to the Ethernet protocol. We will be initializing and interfacing with the controller through the control registers outlined in the datasheet of the AX88796.

Initialization of Ethernet Controller

1) Write 0x21 to the command register to abort current DMA operations
2) Wait 2 ms for it to take effect
3) Write 0x00 to the data control register to enable 8-bit word transfers
4) Write 0x00 to the remote byte count registers to clear remote byte count
5) Write 0x20 to the receive configuration register to set NIC to monitor mode
6) Write 0x02 to the transmission configuration register to set NIC to internal loopback mode
7) Set the Tx start, Rx start, Rx stop and boundary pages by writing the corresponding registers
8) Write 0xff to the interrupt status registers to clear the interrupt flags
9) Write 0x00 to the interrupt mask register to mask all interrupts
10) Write 0x61 to the command register to abort DMA operations
11) Wait 2ms for this to take effect
12) Set the MAC address of the NIC by writing to the physical address registers
13) Set the current Rx page by writing the corresponding register
14) Write 0x20 to the command register to abort DMA operations
15) Write 0x04 to the receive configuration register to set NIC to accept broadcasts
16) Write 0x00 to the transmission configuration register to set normal transfer operations

17) Write 0xff to the interrupt status register to clear interrupt flags
18) Write 0x22 to the command register to start the NIC

Data Transmission

It is necessary to fetch and process incoming packets in small portions. The remote DMA controller is used for this process. Incoming packets will be stored in the SRAM on the controller before attempting to process them. There are two main processes of data transmission: packet reception and packet transmission.

Packet Reception

During packet reception, we will need to complete the following series of tasks:

- Finding the address and length of packet.
- Checking for reception of multiple packets.
- Checking the packet for error and the packet buffer for overflow.

Packet reception is achieved by a local DMA write. The received packets are stored in a buffer ring shown below:

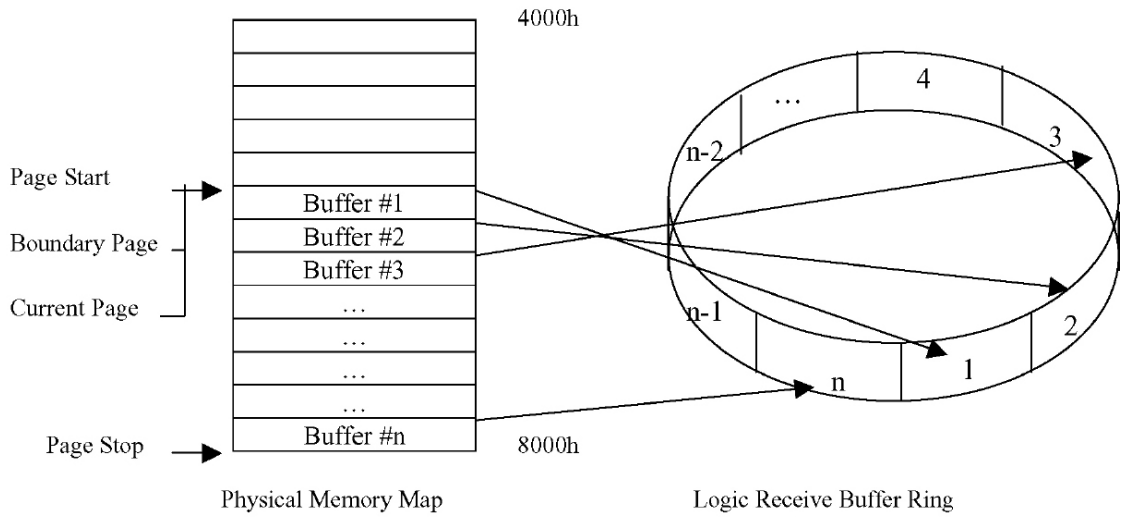


Fig - 9 Receive Buffer Ring At Initialization

The location of the Receive Buffer Ring is set by two registers: Page Start, and Page Stop. Page Start points to the beginning of the buffer ring and Page Stop points to the end of the buffer ring. The DMA address moves through the ring, and whenever it reaches the Page Stop Address, it is reset to the Page Start address. Two registers (Current Page and Boundary Page) is used to initialize the buffer ring and keep the place of all data. The Current Page Register points to the first buffer used to store a packet. It is used to restore DMA for writing status or in case of errors, and acts as a writer pointer. The Boundary Page Register points to the first packet in the buffer that has not yet been read by the host, used to initialize RDMA for removing data, and

acts as a read pointer. If the DMA address of the next buffer is equal to the contents of the Page Stop Register and the Boundary Pointer Register, reception of packet is aborted. If the length of the packet is large enough that it uses up the buffer that attempts to store it, the DMA links the next buffer to the current buffer to store the remaining packet. A maximum of 6 buffers will be linked. If the buffer ring Overflow bit indicates overflow, then we proceed to follow the "Resend" procedure as detailed in the Ethernet Controller manual. It is important to note that packet reception needs to be tested to ensure data's quality. Some testing will involve different packet sizes and high rates of transmission.

When receiving a frame the NIC actually adds its own header to the Ethernet header(shown below) indicating the error status, pointer to the next block and length of frame.

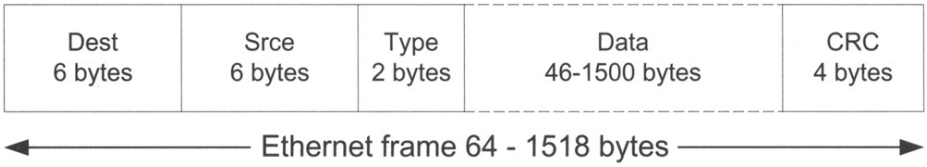
Packet Transmission

During packet transmission, we will need to complete the following series of tasks:

- Start the NIC state machine.
- Write Ethernet header and packet data into packet buffer.
- Set length of the packet, making sure that the length would be rounded up if less than 64 bytes.

Local DMA read is used during the transmission of an Ethernet packet. In order to begin transmission, Transmit Page Start Register and Transmit Byte Count Registers need to be initialized. The TXP bit in the Command Register must be set, the Transmit Status Register must be cleared, and at least one byte has entered the FIFO.

Here is the Ethernet frame format:

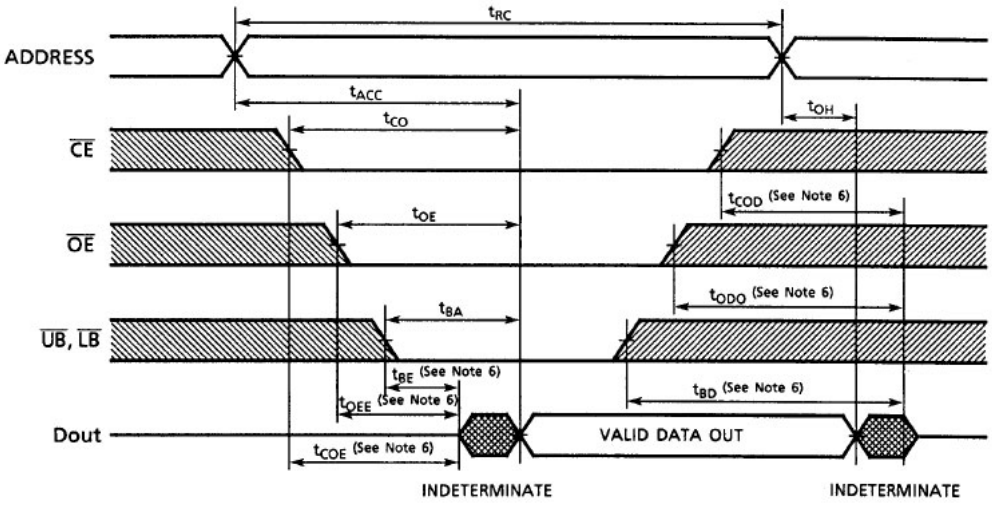


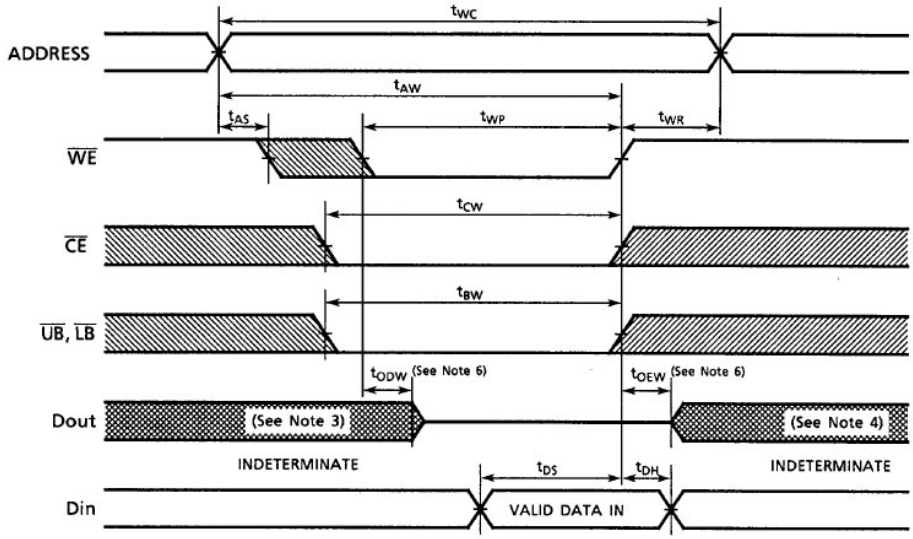
Remote DMA

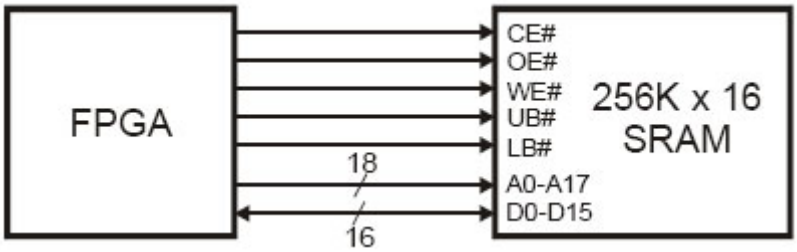
Remote Start Address Registers are pointers that point to the beginning of a block to be moved. Remote Byte Count Registers indicate the number of bytes to be transferred. Write moves data from host to local buffer memory beginning at Remote Start Address, and will do so until byte counter reaches zero. Read reads local buffer memory to the host.

SRAM

We will be storing the data of our web server on the off-chip SRAM, which will be interfaced with the MicroBlaze through a custom peripheral which will handle the initialization write and the subsequent read accesses according to the protocols defined in the





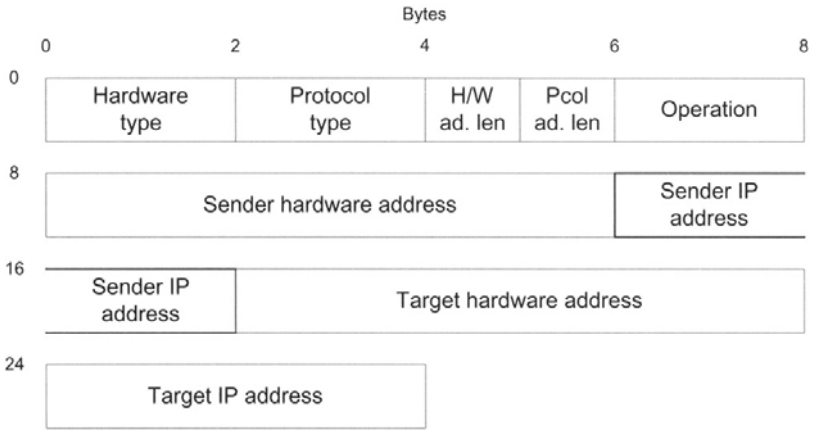


Because our OPB ethernet peripheral had to control the interface from the MicroBlaze to both the SRAM and the Ethernet controller, we mapped an overall peripheral arbiter chip select by matching the first 9 bits of the memory mapped IO. This ensured that we had enough bits left to address both the ethernet controller's control registers and DMA addresses as well as the address space of the SRAM. We used the next two bits to serve as a chip select signal for the SRAM and Ethernet hardware itself and for our memory controller, which contained the state machine's that provided the correct timings for the various signals necessary for interfacing with these peripherals. If the ethernet chip select was high, the ethernet controller interface state machine would be selected and bridge the MicroBlaze and the controller. If the SRAM chip select was high, the ethernet state machine would be inactive and the SRAM state machine was active, to interface with the SRAM according to its own protocols. This allowed us to operate each peripheral in mutual exclusion to the other to ensure that they both arent driving the OPB simulaneously.

PROTOCOLS

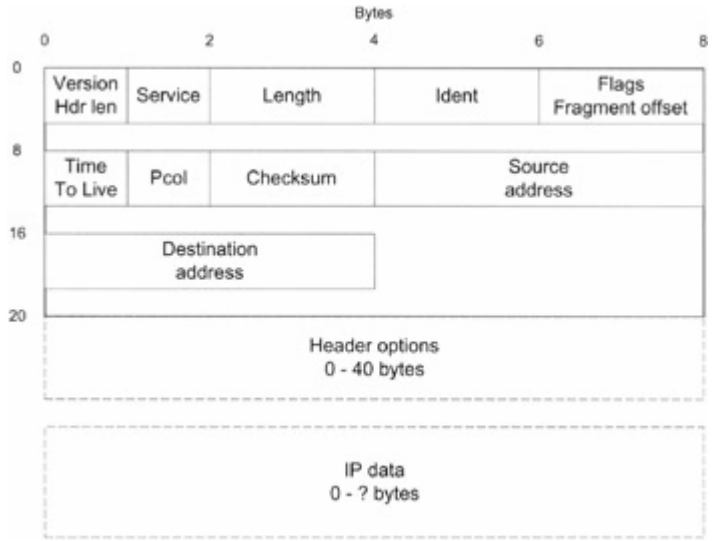
Address Resolution Protocol (ARP) is used to find the MAC address of a node in the network when only the IP address is known. A node sends subnet broadcast containing the IP address that needs to be resolved and the node that matches the IP sends a response containing its MAC.

The following shows the packet sent in an ARP response:



Internet Protocol (IP) is used to convey the TCP segments between hosts. An IP header plus the data block is known as the IP datagram.

The following diagram displays the IP datagram:



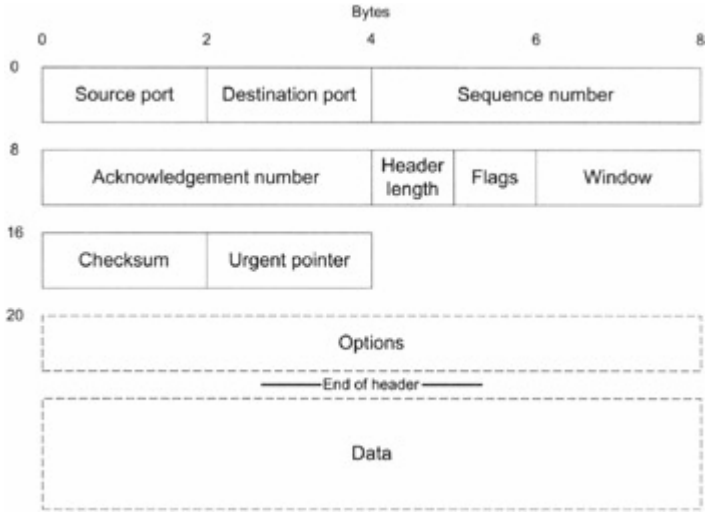
ICMP (Internet Control Message Protocol) is used for network diagnostics. An ICMP message is contained in the data field of an IP datagram. We will be using the ICMP Echo Request (Ping) to check the lower protocol layers. This is done even before we have implemented the web server.

The following diagram is an ICMP message:

TCP/IP provides a reliable connection between two sockets on a network. Parameters that define a socket are the IP address of the client and server and the port number. A web server will respond only to incoming requests on port 80.

As soon as the clients HTTP request has been received, a TCP data segment will be sent out. A TCP segment consists of a header and the data block.

The following diagram displays the TCP segment format:



TESTING AND CHALLENGES

Testing

Our testing methodology was basically to build from ground up. We started with the hardware, studying the data sheets for each of the Ethernet controller and its timing diagrams. We based our design on the peripherals already written by JayCam. When we got the Ethernet peripheral to work, we began to write the functions that interface with the controller. From this we built a diagnostics program to check that we could perform a remote DMA read and write. We modeled our code after the code written in the TCP/IP lean book. We had to make various changes in order for it to function since the book's code was written for a different microcontroller and based on C++. Our primary goal was to fit everything into BRAM. We only took the sections of the code that we felt were necessary and we monitored how much of the BRAM our instructions were using. This basically wrote to certain registers and then read back from them comparing the values. Then we wrote the program to parse an incoming packet. We tested this by pinging the board with a laptop. When we were able to successfully retrieve the header in the packet, we moved on to working on writing the function to perform ARP, ICMP, and IP. We ran into some problems here and so moved to testing the packet transmission from the board, looking at the packet with Ethereal on a laptop. Finally, we wrote the TCP code and HTTP code.

Challenges

Challenges that our group faced include understanding the hardware code and software protocols.

One of the more annoying problems was that for some of the boards the Ethernet controller seemed to not function properly sometimes. We found out because the NIC controller would not initialize, when it worked for on another day.

LESSONS LEARNED

Howard Wang:

It's clichéd, but true: No pain, no gain. If only the marginal gain for each unit of pain we experienced during this project was much greater. Setbacks galore plagued our group, both stemming from our mistakes and uncontrollable exogenous factors. We had envisioned a webserver capable of the entire suite of standard web protocols with packet headers and parameters generated on the fly, optimized for an embedded system. This was much more difficult than we could have envisioned.

Having the protocol code at your disposal is one thing, understanding that code is a completely different thing, and actually implementing that code with the limited instruction space provided by the BRAM of the microblaze was a complete nightmare. Code optimization went from an afterthought to the top of our priority list, after each step we took and after every function we coded, it was necessary to optimize the code in order to fit in our extremely limited instruction space, otherwise, debugging was impossible because compilation was impossible. Know your code, and know what you want out of it.

Study the datasheets well and know the system before you try anything. This will save you much time as you won't find yourself floundering around from file to file uncertain of the function of a certain piece of code.

Plan ahead, and delegate tasks accordingly. Admittedly, a big problem of ours was the availablity of each of our individual group members. We oftentimes ended up taking shifts in the lab without any of our other group members present. Without your group members there to help you catch mistakes and give you ideas, its extremely difficult, especially when you've been working at it for 12+ hours at a time. Additionally, anything thing that you may do becomes a mystery to the others when they come to work on it, and this causes a huge waste of time parsing the other persons code to see what they've done, where they left off, and what else needs to be done.

Watch out for dead/semi-dead boards. We ended spending hours trying to debug what seemed like a perfectly fuctional system just minutes before, and lost a whole lot of time because some of the ethernet controllers just gave up on working.

Don't be afraid to ask for help. Everybody else is doing it. Don't get proud. Know when to give in.

But through all this, I have admitted learned more than I would have imagined about networking, custom hardware design, and low level coding. Prior to this class, I had done an extremely limited amount of coding in C, barely rembering simple syntax rules. I had never worked with FPGA'a and HDLs and the concepts and basic structures which were foreign to me before are now the methodologies that I know to follow when developing custom hardware. Networking was also completely foreign to me as well. I never understood the underlying protocols and packet structures and the complex exchanges that go on between clients and servers on a

network. Starting a project of this level on such a background did indeed prove to be and extremely difficult task, one which cost me many hours of sleep and sanity. But in the end, the knowledge and project experience gained from such an endeavor was more than I could have hoped for. It really was a case of you get what you put in.

William Wong:

I learned a lot about networking while doing this project, which included the different levels of the protocol stack and the type of actions that must be performed. The actual implementation was much harder than we expected and we had to do a lot of debugging. I ended up doing a lot more software than I expected. On the bright side, that gave me a good understanding of the internals of each protocol. I think it would have been smarter if we chose a project related to a topic that we were more familiar with, rather than try to tackle something completely new to us. Planning ahead is also very important.

Franklin Ma:

This project gave me a good opportunity to learn Internet Protocols. After a couple of weeks of working with the web server, I have a better understanding of how the internet works. TCP/IP, UDP, etc used to be foreign words to me but I have become more familiar with such terms and the meaning behind them. Also, I was always confused on how to implement hardware and software or where they both came together, but I also learned various bits and pieces so that overall I have a better understanding of computers in general.

Victor Wang:

I pray to the many deities of Engineering that those engineers who also want to brave this course in the future start their projects early. It is very important to find out that whatever brilliant design plan that had formulated in your minds, though feasible, are not quite acknowledged by the special laws that govern these Xilinx boards. At this time, I'd like to quote the Murphy's Law for your personal enlightenment.

"Every solution breeds new problems"

On a more serious note, I've learned much about TCP/IP and the workings of Ethernet, and Ethernet communication. More importantly, I've learned how to dive into hardware and patiently understand the flaws of both man and machine.

CODE

memoryctrl.vhd

```
------------------------------
-- CSEE 4840 Embedded System Design
--
-- Webserver
-- Franklin Ma, Howard Wang, Victor Wang, William Wong
--
------------------------------

-- Based on Jaycam ethernet vhdl

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity memoryctrl is
    Port ( rst : in std_logic;
           clk : in std_logic;
           cs : in std_logic; -- any of my devices selected
           opb_select : in std_logic; -- original select

           rnw : in std_logic;

           eth_io : in std_logic;
           fullword : in std_logic;
         read_early : out std_logic;
         write_early : out std_logic;
           bus_req : out std_logic;

           videocycle : out std_logic;
           hihalf : out std_logic;
           wr_req : out std_logic;
           rd_req : out std_logic;
           xfer : out std_logic;
           ce0 : out std_logic;
           ce1 : out std_logic;
           rres : out std_logic;

         ram_output_enable : out std_logic;
         ram_WE : out std_logic;
         sram_ce : in std_logic
                   );
end memoryctrl;

architecture Behavioral of memoryctrl is
```

```
signal r_idle, r_common, r_w32, r_ra, r_rb, r_rc, r_xfer : std_logic;
signal r_weth1, r_weth2, r_weth3 : std_logic;
signal r_v1, r_v0, r_v2 : std_logic;
signal wr_req_i, rd_req_i, videocycle_i : std_logic;
signal sram_rst : std_logic;


  signal WE : std_logic;
  signal output_enable : std_logic;

  constant STATE_BITS : integer := 3;
  constant Idle     : std_logic_vector(0 to 2) := "000";
  constant Selected : std_logic_vector(0 to 2) := "001";
  constant Read     : std_logic_vector(0 to 2) := "011";
  constant Xfer1     : std_logic_vector(0 to 2) := "111";
  constant Xfer2     : std_logic_vector(0 to 2) := "110";

  signal present_state, next_state : std_logic_vector(0 to 2);

begin

process(rst, clk)
begin
      if rst = '1' or cs='0' then
          r_idle <= '1';
          r_common <= '0';
          r_w32 <= '0';
          r_ra <= '0'; r_rb <= '0'; r_rc <= '0'; r_xfer <= '0'; r_weth1 <= '0';
          r_weth2 <= '0'; r_weth3 <= '0';

      elsif clk'event and clk='1' then

      r_idle <= (r_idle and not cs) or (r_xfer) or (not opb_select);
      r_common <= opb_select and (r_idle and cs);

        r_weth1 <= opb_select and (r_common and not rnw and eth_io);
        r_weth2 <= opb_select and (r_weth1);
        r_weth3 <= opb_select and (r_weth2);
      r_w32 <= opb_select and (r_common and not rnw and fullword);--

      r_ra <= opb_select and (r_common and rnw);
      r_rb  <= opb_select and (r_ra);
        r_rc <= opb_select and r_rb and (fullword or eth_io);

      r_xfer <= opb_select and ( (r_common and not rnw and not fullword and not eth_io)
             or (r_w32) --
             or (r_ra and not fullword and not eth_io) --b
             or (r_rb) --c
                 or (r_weth1));

      end if;
end process;

  fsm_seq : process(clk, sram_rst)
  begin
   if (sram_ce='1') then
```

```
    if sram_rst = '1' then
      present_state <= Idle;
    elsif clk'event and clk = '1' then
      present_state <= next_state;
    end if;

      xfer <= present_state(0);
      rres <= present_state(0);

      read_early <= '1'; -- hmmm   -- r_ra and eth_io;
      write_early <= '1';
      hihalf <= '0';

      wr_req_i <= not WE;
      rd_req_i <= WE;
      wr_req <= wr_req_i;
      rd_req <= rd_req_i;
      bus_req <= rd_req_i or wr_req_i or r_common;

      ce0 <= '1';
      ce1 <= '1';


   else
      xfer <= r_xfer;
      rres <= r_xfer;

      read_early <= r_ra and eth_io;
      write_early <= not ((r_common and not rnw and eth_io) or (r_weth1) or r_weth2);
      hihalf <= r_w32 or (r_ra and fullword);

      wr_req_i <= (r_common and not rnw and not eth_io) or (r_w32) or (r_weth1) or
(r_weth2) or (r_weth3);
      rd_req_i <= (r_common and rnw) or (r_ra and (fullword or eth_io));
      wr_req <= wr_req_i;
      rd_req <= rd_req_i;
      bus_req <= rd_req_i or wr_req_i or r_common;

      ce0 <= r_rb or (r_rc and eth_io);
      ce1 <= r_rb or r_rc;
   end if;
--   end if;

  end process fsm_seq;


  fsm_comb : process(sram_rst, present_state, sram_ce, OPB_Select, rnw)
  begin
--  if (sram_ce = '1') then
    sram_rst <= '1';
    WE <= '1';
    output_enable <= '0';
    if rst = '1' then
      next_state <= Idle;
    else
      case present_state is
        when Idle =>
```

```
          if sram_ce = '1' then
            next_state <= Selected;
          else
            WE <= '1';--
          output_enable <= '1';--
            next_state <= Idle;
          end if;

        when Selected =>
          if OPB_Select = '1' then
            if RNW = '1' then
              sram_rst <= '0';
              next_state <= Read;
            else
              WE <= '0';
              next_state <= Xfer1;
            end if;
          else
            next_state <= Idle;
          end if;

        when Read =>
--          if OPB_Select = '1' then
            output_enable <= '1';
            next_state <= Xfer1;
--          else
--            next_state <= Idle;
--          end if;

        -- State encoding is critical here: xfer must only be true here
        when Xfer1 =>
          next_state <= Idle;


        when others =>
          next_state <= Idle;
      end case;
--    end if;

ram_output_enable <= not output_enable;
ram_we <= WE;


    end if;
  end process fsm_comb;




end Behavioral;
```

opb_ethernet.vhd

```
------------------------------
-- CSEE 4840 Embedded System Design
--
```

```
-- Webserver
-- Franklin Ma, Howard Wang, Victor Wang, William Wong
--
-- based on Jaycam ethernet vhdl
------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity opb_ethernet is
  generic (
    C_OPB_AWIDTH    : integer := 32;
    C_OPB_DWIDTH    : integer := 32;
    C_BASEADDR      : std_logic_vector := X"2000_0000";
    C_HIGHADDR      : std_logic_vector := X"2000_00FF";
    C_ETHERNET_DWIDTH : integer := 16;
    C_ETHERNET_AWIDTH : integer := 10
    );

    Port (  OPB_Clk : in std_logic;
                      OPB_Rst : in std_logic;
                      OPB_ABus : in std_logic_vector (31 downto 0);
                      OPB_BE : in std_logic_vector (3 downto 0);
                      OPB_DBus : in std_logic_vector (31 downto 0);
                      OPB_RNW : in std_logic;
                      OPB_select : in std_logic;
                      OPB_seqAddr : in std_logic;
                              io_clock: in std_logic;
                      Sln_DBus : out std_logic_vector (31 downto 0);
                      Sln_errAck : out std_logic;
                      Sln_retry : out std_logic;
                      Sln_toutSup : out std_logic;
                      Sln_xferAck : out std_logic;
                      PB_A : out std_logic_vector (19 downto 0);
                      PB_UB_N : out std_logic;
                      PB_LB_N : out std_logic;
                      PB_WE_N : out std_logic;
                      PB_OE_N : out std_logic;
                      RAM_CE_N : out std_logic;
                      ETHERNET_CS_N : out std_logic;
                      ETHERNET_RDY : in std_logic;
                      ETHERNET_IREQ : in std_logic;
                      ETHERNET_IOCS16_N : in std_logic;


                      PB_D : inout std_logic_vector (15 downto 0)
                  );
end opb_ethernet;

architecture Behavioral of opb_ethernet is
```

```
signal addr_mux : std_logic_vector(19 downto 0);
signal video_addr : std_logic_vector (19 downto 0);
signal video_data : std_logic_vector (15 downto 0);
signal i : integer;
signal cs : std_logic;

signal fullword, eth_io, read_early, write_early : std_logic ;
signal videocycle, amuxsel, hihalf : std_logic;
signal rce0, rce1, rreset : std_logic;
signal xfer : std_logic;
signal wr_req, rd_req, bus_req : std_logic;
signal pb_rd, pb_wr : std_logic;

signal sram_ce : std_logic;
signal ethernet_ce : std_logic;


signal rnw, ram_we, ram_oe, oe : std_logic;

signal addr : std_logic_vector (23 downto 0);

signal be : std_logic_vector (3 downto 0);
signal pb_bytesel : std_logic_vector (1 downto 0);

signal wdata : std_logic_vector (31 downto 0);
signal wdata_mux : std_logic_vector (15 downto 0);

signal rdata : std_logic_vector (15 downto 0);  -- register data read - FDRE


component memoryctrl
Port (        rst : in std_logic;
           clk : in std_logic;
           cs : in std_logic; -- any of my devices selected
           opb_select : in std_logic; -- original select
           rnw : in std_logic;
           eth_io : in std_logic;
           fullword : in std_logic;
           read_early : out std_logic;
           write_early : out std_logic;
           videocycle : out std_logic;
           hihalf : out std_logic;
           wr_req : out std_logic;
           rd_req : out std_logic;
           bus_req : out std_logic;
           xfer : out std_logic;
           ce0 : out std_logic;
           ce1 : out std_logic;
           rres : out std_logic;
         ram_output_enable : out std_logic;
         ram_WE : out std_logic;
         sram_ce : in std_logic);
end component;

component pad_io
    Port ( sys_clk : in std_logic;
```

```
          io_clock : in std_logic;
          read_early : in std_logic;
          write_early : in std_logic;
          rst : in std_logic;
          PB_A : out std_logic_vector(19 downto 0);
          PB_UB_N : out std_logic;
          PB_LB_N : out std_logic;
          PB_WE_N : out std_logic;
          PB_OE_N : out std_logic;
          RAM_CE_N : out std_logic;
          ETHERNET_CS_N : out std_logic;
          ETHERNET_RDY : in std_logic;
          ETHERNET_IREQ : in std_logic;
          ETHERNET_IOCS16_N : in std_logic;
          PB_D : inout std_logic_vector(15 downto 0);

          pb_addr : in std_logic_vector(19 downto 0);
          pb_ub : in std_logic;
          pb_lb : in std_logic;
          pb_wr : in std_logic;
          pb_rd : in std_logic;
          ram_ce : in std_logic;
        ethernet_ce : in std_logic;
          pb_dread : out std_logic_vector(15 downto 0);
          pb_dwrite : in std_logic_vector(15 downto 0));
end component;


begin



-- the controller state machine
memoryctrl1 : memoryctrl
port map    (     rst => OPB_Rst,
                  clk => OPB_Clk,
                  cs => cs,
                  opb_select => OPB_select,
                  rnw => rnw,

                  fullword => fullword,
                  eth_io =>eth_io,
                  read_early => read_early,
                  write_early => write_early,
                  videocycle => videocycle,
                  hihalf => hihalf,
                  wr_req => wr_req,
                  rd_req => rd_req,
                        bus_req => bus_req,

                  xfer => xfer,
                  ce0 => rce0,
                  ce1 => rce1,
                  rres => rreset,


                  ram_output_enable => ram_oe,
```

```
                  ram_WE => ram_we,
                sram_ce => sram_ce
                );

-- PADS

pad_io1 : pad_io
port map    (     sys_clk => OPB_Clk,
                  io_clock => io_clock,
                        read_early => read_early,
                        write_early => write_early,
                  rst => OPB_Rst,
                  PB_A => PB_A,
                  PB_UB_N => PB_UB_N,
                  PB_LB_N => PB_LB_N,
                  PB_WE_N => PB_WE_N,
                  PB_OE_N => PB_OE_N,
                  RAM_CE_N => RAM_CE_N,
                  ETHERNET_CS_N => ETHERNET_CS_N,
                  ETHERNET_RDY => ETHERNET_RDY,
                  ETHERNET_IREQ => ETHERNET_IREQ,
                  ETHERNET_IOCS16_N => ETHERNET_IOCS16_N,
                  PB_D => PB_D,

                  pb_addr => addr_mux,
                  pb_wr => pb_wr,
                  pb_rd => pb_rd,
                  pb_ub => pb_bytesel(1),
                  pb_lb => pb_bytesel(0),
                  ram_ce => sram_ce,
                  ethernet_ce => ethernet_ce,

                  pb_dread  => rdata,
                  pb_dwrite => wdata_mux);


      addr_mux <= (addr(20 downto 2)) & (addr(1) or hihalf);

      fullword <= be(2) and be(0);

      wdata_mux <= wdata(15 downto 0) when ((addr(1) or hihalf) = '1') else wdata(31
downto 16);




pb_rd <= rd_req or videocycle;
pb_wr <= wr_req;
oe <= ram_oe when (sram_ce='1') else pb_rd;

cs <= OPB_select when OPB_ABus(31 downto 23) = "000000001" else '0';
sram_ce <= '1' when addr(22 downto 21)="00" and (bus_req = '1') else '0';
ethernet_ce <= '1' when addr(22 downto 21) ="01" else '0';
eth_io <= '1' when addr(22 downto 21) /= "10" else '0';
```

```
process (OPB_Clk, OPB_Rst)
begin

-- register rw
      if OPB_Clk'event and OPB_Clk ='1' then

            if OPB_Rst = '1' then
                  rnw <= '0';
            else
                  rnw <= OPB_RNW;
            end if;

      end if;

-- register adresses A23 .. A0
      if OPB_Clk'event and OPB_Clk ='1' then
         for i in 0 to 23 loop
                  if OPB_Rst = '1' then
                        addr(i) <= '0';
                  else
                        addr(i) <= OPB_ABus(i);
                  end if;
      end loop;
      end if;

      -- register BE
      if OPB_Clk'event and OPB_Clk ='1' then
                  if OPB_Rst = '1' then
                        be <= "0000";
                  else
                        be <= OPB_BE;
                  end if;
      end if;

-- register data write
      if OPB_Clk'event and OPB_Clk ='1' then
         for i in 0 to 31 loop
                  if OPB_Rst = '1' then
                        wdata(i) <= '0';
                  else
                        wdata(i) <= OPB_DBus(i);
                  end if;
      end loop;
      end if;

-- ce0/ce1 enables writing MSB (low) / LSB (high) halves

--always @(posedge OPB_Rst or posedge OPB_Clk) begin (synchrounous or asynchronous
reset??)

            for i in 0 to 15 loop
                  if OPB_Rst = '1' then
                        Sln_DBus(i) <= '0';

                  elsif OPB_Clk'event and OPB_Clk ='1' then
                        if rreset = '1' then
                              Sln_DBus(i) <= '0';
```

```
                        elsif (rce1 or rce0) = '1' then
                              Sln_DBus(i) <= rdata(i);
                        end if;
                  end if;
            end loop;

            for i in 16 to 31 loop
                  if OPB_Rst = '1' then
                        Sln_DBus(i) <= '0';
                  elsif OPB_Clk'event and OPB_Clk ='1' then

                        if rreset = '1' then
                              Sln_DBus(i) <= '0';
                        elsif rce0 = '1' then
                              Sln_DBus(i) <= rdata(i-16);
                        end if;
                  end if;
            end loop;


end process;

-- tie unused to ground
Sln_errAck <= '0';
Sln_retry <= '0';
Sln_toutSup <= '0';

Sln_xferAck <= xfer;


end Behavioral;
```

pad_io.vhd

```
------------------------------
-- CSEE 4840 Embedded System Design
--
-- Webserver
-- Franklin Ma, Howard Wang, Victor Wang, William Wong
--
------------------------------

-- Based on Jaycam ethernet vhdl

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;

entity pad_io is
    Port ( sys_clk : in std_logic;
           io_clock : in std_logic;
           read_early : in std_logic;
           write_early : in std_logic;
```

```
            rst : in std_logic;
            PB_A : out std_logic_vector(19 downto 0);
            PB_UB_N : out std_logic;
            PB_LB_N : out std_logic;
            PB_WE_N : out std_logic;
            PB_OE_N : out std_logic;
            RAM_CE_N : out std_logic;
            ETHERNET_CS_N : out std_logic;
            ETHERNET_RDY : in std_logic;
            ETHERNET_IREQ : in std_logic;
            ETHERNET_IOCS16_N : in std_logic;
            PB_D : inout std_logic_vector(15 downto 0);

            pb_addr : in std_logic_vector(19 downto 0);
            pb_ub : in std_logic;
            pb_lb : in std_logic;
            pb_wr : in std_logic;
            pb_rd : in std_logic;
            ram_ce : in std_logic;
            ethernet_ce : in std_logic;
            pb_dread : out std_logic_vector(15 downto 0);
            pb_dwrite : in std_logic_vector(15 downto 0));
end pad_io;

architecture Behavioral of pad_io is

component FDCE
port (C : in std_logic;
            CLR : in std_logic;
            CE : in std_logic;
            D : in std_logic;
            Q : out std_logic);
end component;

component FDPE
port (C : in std_logic;
            PRE : in std_logic;
            CE : in std_logic;
            D : in std_logic;
            Q : out std_logic);
end component;

attribute iob : string;
attribute iob of FDCE : component is "true";
attribute iob of FDPE : component is "true";

component OBUF_F_24
port (O : out STD_ULOGIC;
            I : in STD_ULOGIC);
end component;

component IOBUF_F_24
port (O : out STD_ULOGIC;
            IO : inout STD_ULOGIC;
            I : in STD_ULOGIC;
            T : in STD_ULOGIC);
```

```
end component;

signal io_half : std_logic;
signal pb_addr_1: std_logic_vector(19 downto 0);
signal pb_dwrite_1: std_logic_vector(15 downto 0);
signal pb_tristate: std_logic_vector(15 downto 0);
signal pb_tristate_1: std_logic_vector(15 downto 0);
signal pb_dread_a: std_logic_vector(15 downto 0);
signal we_n, pb_we_n1: std_logic;
signal oe_n, pb_oe_n1: std_logic;
signal lb_n, pb_lb_n1: std_logic;
signal ub_n, pb_ub_n1: std_logic;
signal ethce_n, eth_ce_n1 : std_logic;
signal ramce_n, ram_ce_n1: std_logic;
signal dataz : std_logic;

signal rd_ce, wr_ce, din_ce, rd_early, wr_early : std_logic;

--attribute equivalent_register_removal: string;
--attribute equivalent_register_removal of pb_tristate_1 : signal is "no";
--attribute equivalent_register_removal of pb_dwrite_1 : signal is "no";


begin

------------------------ !!!!!!!!!!!!!!!!!!!! -----------------------------
--process (io_clock)
--begin
--  if io_clock'event and io_clock = '1' then
--    io_half <= sys_clk;
--  end if;
--end process;
io_half <= not sys_clk;
--------------------------------------------------------------------------------

process(rst, sys_clk)
begin
      if rst='1' then
          rd_early <= '0';
          wr_early <= '0';
        elsif sys_clk'event and sys_clk='1' then
          rd_early <= read_early;
          wr_early <= write_early;
      end if;
end process;

dataz <= (not pb_wr) or pb_rd;
pb_tristate_1 <= "1111111111111111" when dataz ='1' else "0000000000000000";



-- control
pb_we_n1 <= not pb_wr; -- or (not io_half and wr_early);

pb_oe_n1 <= not pb_rd; --or (not io_half and rd_early);

pb_lb_n1 <= '0' when (ram_ce='1') else (not pb_lb);
```

```
pb_ub_n1 <= '0' when (ram_ce='1') else (not pb_ub);

ram_ce_n1 <= not ram_ce;

eth_ce_n1 <= not ethernet_ce;


-- I/O BUFFERS

webuf : OBUF_F_24
port map (O => PB_WE_N,
                  I => pb_we_n1);

oebuf : OBUF_F_24
port map (O => PB_OE_N,
                  I => pb_oe_n1);

ramcebuf : OBUF_F_24
port map (O => RAM_CE_N,
                  I => ram_ce_n1);

ethcebuf : OBUF_F_24
port map (O => ETHERNET_CS_N,
                  I => eth_ce_n1);

--  ETHERNET_RDY : in std_logic;
--  ETHERNET_IREQ : in std_logic;
--  ETHERNET_IOCS16_N : in std_logic;




ubbuf : OBUF_F_24
port map (O => PB_UB_N,
                  I => pb_ub_n1);

lbbuf : OBUF_F_24
port map (O => PB_LB_N,
                  I => pb_lb_n1);

abuf : for i in 0 to 19 generate
      abuf : OBUF_F_24 port map (
            O => PB_A(i),
            I => pb_addr(i));
end generate;

dbuf : for i in 0 to 15 generate
      dbuf : IOBUF_F_24 port map (
            O => pb_dread(i),
            IO => PB_D(i),
            I => pb_dwrite(i),
            T => pb_tristate_1(i));
end generate;
```

```
end Behavioral;
```

opb_ethernet_v2_1_0.mpd

```
#------------------------------
# CSEE 4840 Embedded System Design
#
# Webserver
# Franklin Ma, Howard Wang, Victor Wang, William Wong
#
#------------------------------
## Microprocessor Peripheral Definition


BEGIN opb_ethernet, IPTYPE = PERIPHERAL, EDIF=TRUE

OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL

BUS_INTERFACE BUS = SOPB, BUS_STD = OPB, BUS_TYPE = SLAVE

## Generics for VHDL
PARAMETER c_baseaddr     = 0xFFFFFFFF,DT = std_logic_vector,MIN_SIZE=0x100, BUS=SOPB
PARAMETER c_highaddr     = 0x00000000, DT = std_logic_vector, BUS=SOPB
PARAMETER c_opb_awidth   = 32,         DT = integer
PARAMETER c_opb_dwidth   = 32,         DT = integer
PARAMETER c_ethernet_dwidth   = 16,        DT = integer
PARAMETER c_ethernet_awidth   = 10,        DT = integer

## Ports
PORT opb_abus    = OPB_ABus,    DIR = IN, VEC = [0:(c_opb_awidth-1)],     BUS = SOPB
PORT opb_be      = OPB_BE,      DIR = IN, VEC = [0:((c_opb_dwidth/8)-1)], BUS = SOPB
PORT opb_clk     = "",          DIR = IN, SIGIS=CLK,                BUS = SOPB
PORT opb_dbus    = OPB_DBus,    DIR = IN, VEC = [0:(c_opb_dwidth-1)],     BUS = SOPB
PORT opb_rnw     = OPB_RNW,     DIR = IN,                                 BUS = SOPB
PORT opb_rst     = OPB_Rst,     DIR = IN,                                 BUS = SOPB
PORT opb_select  = OPB_select,  DIR = IN,                                 BUS = SOPB
PORT opb_seqaddr = OPB_seqAddr, DIR = IN,                                 BUS = SOPB
PORT sln_dbus    = Sl_DBus,     DIR = OUT, VEC = [0:(c_opb_dwidth-1)],    BUS = SOPB
PORT sln_errack  = Sl_errAck,   DIR = OUT,                                BUS = SOPB
PORT sln_retry   = Sl_retry,    DIR = OUT,                                BUS = SOPB
PORT sln_toutsup = Sl_toutSup,  DIR = OUT,                                BUS = SOPB
PORT sln_xferack = Sl_xferAck,  DIR = OUT,                                BUS = SOPB

PORT ETHERNET_CS_N = "",      DIR = OUT,  IOB_STATE=BUF
PORT ETHERNET_RDY = "",       DIR = IN
PORT ETHERNET_IREQ = "",      DIR = IN
PORT ETHERNET_IOCS16_N = "",  DIR = IN

PORT PB_D        = "",        DIR = INOUT,      VEC = [c_ethernet_dwidth-1:0],
3STATE=FALSE, IOB_STATE=BUF
PORT PB_A    = "",            DIR = OUT,  VEC = [19:0], 3STATE=FALSE, IOB_STATE=BUF
PORT PB_OE_N       = "",            DIR = OUT, IOB_STATE=BUF
PORT PB_WE_N       = "",            DIR = OUT, IOB_STATE=BUF
PORT PB_UB_N       = "",            DIR = OUT, IOB_STATE=BUF
```

```
PORT PB_LB_N       = "",            DIR = OUT, IOB_STATE=BUF
PORT RAM_CE_N      = "",            DIR = OUT, IOB_STATE=BUF

PORT io_clock = "", DIR=IN


END
```

opb_ethernet_v2_1_0.pao

```
#------------------------------
# CSEE 4840 Embedded System Design
#
# Webserver
# Franklin Ma, Howard Wang, Victor Wang, William Wong
#
#------------------------------


lib opb_ethernet_v1_00_a memoryctrl
lib opb_ethernet_v1_00_a pad_io
lib opb_ethernet_v1_00_a opb_ethernet
```

clkgen.v

```
module clkgen(
FPGA_CLK1,

sys_clk,
pixel_clock,
io_clock,
fpga_reset

);

input FPGA_CLK1;
output sys_clk, pixel_clock, io_clock, fpga_reset;


wire clk_ibuf, clk1x_i, clk2x_i;

//wire clk1x, clk05x;

//wire clk_ibuf, clk1x_i, clk05x_i, clk2x_i;
wire locked;

assign pixel_clock = 0; //new

IBUFG clkibuf(.I(FPGA_CLK1), .O(clk_ibuf));
BUFG bg1 (.I(clk1x_i), .O(sys_clk));
//BUFG bg05 (.I(clk05x_i), .O(pixel_clock));
BUFG bg2 (.I(clk2x_i), .O(io_clock));

// synopsys translate_off
// defparam vdll.CLKDV_DIVIDE = 2.0 ;
// synopsys translate_on
```

```
// synthesis attribute CLKDV_DIVIDE of vdll is 2
//CLKDLL vdll(.CLKIN(clk_ibuf), .CLKFB(sys_clk), .CLK0(clk1x_i),
//            .CLKDV(clk05x_i), .CLK2X(clk2x_i), .RST(1'b0), .LOCKED(locked));
CLKDLL vdll(.CLKIN(clk_ibuf), .CLKFB(sys_clk), .CLK0(clk1x_i),
            .CLK2X(clk2x_i), .RST(1'b0), .LOCKED(locked));

assign fpga_reset = ~locked;


endmodule
```

clkgen_v2_1_0.mpd

```
##############################################################################
##
## Microprocessor Peripheral Definition : generated by psfutil
##
## Template MPD for Peripheral:MicroBlaze_Brd_ZBT_ClkGen
##
##############################################################################

BEGIN clkgen ,IPTYPE = IP

## Peripheral Options
#OPTION IPTYPE = IP
OPTION HDL = VERILOG

OPTION IMP_NETLIST = TRUE


## Ports
PORT FPGA_CLK1 = "", DIR = IN , IOB_STATE = BUF

PORT sys_clk = "", DIR = OUT
PORT pixel_clock = "", DIR = OUT
PORT fpga_reset = "", DIR = OUT
PORT io_clock ="", DIR = OUT


END
```

clkgen_v2_1_0.pao

```
##############################################################################
##
## Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.  Xilinx, Inc.
##
## MicroBlaze_Brd_ZBT_ClkGen_v2_0_0_a.pao
##
## Peripheral Analyze Order
##
##############################################################################
```

```
lib clkgen_v1_00_a clkgen
```

main.c

```
//-----------------------------
// CSEE 4840 Embedded System Design - System Constraints File
//
// Webserver
// Franklin Ma, Howard Wang, Victor Wang, William Wong
//
//-----------------------------

#include "xparameters.h"
#include "xintc_l.h"
#include "ethernet.h"

int main()
{

  int ret;

  int i;
  int addr;
  char j, k;

  myip[3] = 192;
  myip[2] = 168;
  myip[1] = 100;
  myip[0] = 100;

  print("\r\n");
  print("Hello World!\r\n") ;
  print("Running diagnostics\r\n");
   if (ret=diagnostics()) {
     print("Diagnostics failed. internal error number ");
     putnum(ret);
     print("\r\n");

   } else {
     print("Diagnostics successful.\r\n");
   }

   //if(init())
   if(!init_ether())
     {
       print("Ethernet NIC not present or not initializing correctly\r\n");
     return 1;
   }

   print("waiting for packets\r\n");
   //putnum(nicin.nic.len);
   while(1)
     {
```

```
    rxin = rxout = 0;
    atend = 0;
    if(get_ether())
    {

      //for(i=0;i<sizeof(nicin);i++) {
      //  putnum(((unsigned char*)&nicin)[i]);
      //   print("-----\n\r");
      //}
      if (nicin.eth.pcol == PCOL_ARP) {
         print("ARP TIME\r\n");
         arp_recv();         // ..is it ARP?{
        } else if (nicin.eth.pcol == PCOL_IP) {
         print("IP TIME");
         if (ipcol == PICMP)                          // ICMP?
               icmp_recv();
         // ..or is it IP?
      }
    }
  }

/*
j = 0;
for ( i = 0 ; i < 5 ; i++ ) {
  addr = 0x00C00000 + (i << 2);
  XIo_Out16(addr, j);
  print("writing ");
  putnum(j);
  print(" to ");
  putnum(addr);
  print("\r\n");
  j += 17;
}

j = 0;
for ( i = 0 ; i < 5 ; i++ ) {
  addr = 0x00C00000 + (i << 2);
  k = XIo_In16(addr);
  putnum(addr);
  print(" got ");
  putnum(k);
  if ( k != j ) {
    print(" expected ");
    putnum(j);
  }
  print("\r\n");
  j += 17;
}
*/

/*
  XIo_Out16(0xC00008, 12);
  putnum(XIo_In16(0xC00008));
  print("\r\n");
  putnum(XIo_In16(0xC0000C));
  print("\r\n");
*/
```

```
   return 0;
}
```

ethernet.h

```
//-----------------------------
// CSEE 4840 Embedded System Design - System Constraints File
//
// Webserver
// Franklin Ma, Howard Wang, Victor Wang, William Wong
//
//-----------------------------

#include "xio.h"
#include "xbasic_types.h"

#ifndef BYTE
#define BYTE unsigned char
#endif
#ifndef WORD
#define WORD unsigned short
#endif
#ifndef BOOL
#define BOOL char
#endif
//#ifndef LWORD
//#define LWORD unsigned long
//#endif

#define outnic(addr, data) XIo_Out16(NIC_BASE+addr, data)
#define innic(addr) (XIo_In16(NIC_BASE+addr))

#define PROMISC  0              /* Set non-zero to accept all packets */

/* Ethernet definitions.. */
#define MINFRAME  60
#define MINFRAMEC 64
#define CRCLEN    4
#define MAXFRAME  1514
#define MAXFRAMEC 1518

/* NE2000 definitions */
#define NIC_BASE (0x00A00400) // Base I/O address of the NIC card
#define DATAPORT (0x10*2)
#define NE_RESET (0x1f*2)

/* 8390 Network Interface Controller (NIC) page0 register offsets */
#define CMDR    (0x00*2)            /* command register for read & write */
#define PSTART  (0x01*2)            /* page start register for write */
#define PSTOP   (0x02*2)            /* page stop register for write */
#define BNRY    (0x03*2)            /* boundary reg for rd and wr */
#define TPSR    (0x04*2)            /* tx start page start reg for wr */
#define TBCR0   (0x05*2)            /* tx byte count 0 reg for wr */
#define TBCR1   (0x06*2)            /* tx byte count 1 reg for wr */
#define ISR     (0x07*2)            /* interrupt status reg for rd and wr */
```

```
#define RSAR0   (0x08*2)           /* low byte of remote start addr */
#define RSAR1   (0x09*2)           /* hi byte of remote start addr */
#define RBCR0   (0x0A*2)           /* remote byte count reg 0 for wr */
#define RBCR1   (0x0B*2)           /* remote byte count reg 1 for wr */
#define RCR     (0x0C*2)           /* rx configuration reg for wr */
#define TCR     (0x0D*2)           /* tx configuration reg for wr */
#define DCR     (0x0E*2)           /* data configuration reg for wr */
#define IMR     (0x0F*2)           /* interrupt mask reg for wr */

/* NIC page 1 register offsets */
#define PAR0    (0x01*2)           /* physical addr reg 0 for rd and wr */
#define CURRP   (0x07*2)           /* current page reg for rd and wr */
#define MAR0    (0x08*2)           /* multicast addr reg 0 for rd and WR */

/* NIC page 3 register offsets */
//#define RTL9346CR 0x01          /* RTL 9346 command reg */
//#define RTL3    0x06            /* RTL config reg 3 */

/* NIC RAM definitions */
//#define RAMPAGES 0x20           /* Total number of 256-byte RAM pages */
#define TXSTART  0x41           /* Tx buffer start page */
#define TXPAGES  8              /* Pages for Tx buffer */
#define RXSTART  (TXSTART+TXPAGES)      /* Rx buffer start page */
#define RXSTOP   0x80   /* Last Rx buffer page */
#define DCRVAL   0x00           /* Value for data config reg for word-wide tx */

/* IP protocol definitions */

#define PCOL_ARP    0x0806  // Protocol type: ARP
#define PCOL_IP     0x0800  //                IP

#define ARPREQ     0x0001   // ARP request & response IDs
#define ARPRESP    0x0002

#define PICMP       1       // IP protocol values: ICMP
#define PTCP        6       //                     TCP
#define PUDP        17      //                     UDP

#define ETHHDR_LEN  6       // Ethernet frame header length
#define IPHDR_LEN   20      // IP, TCP and ICMP header lengths
#define TCPHDR_LEN  20
#define ICMPHDR_LEN 4       // (only include type, code & csum in ICMP hdr)
#define TCPOPT_LEN  4       // Length of TCP MSS option
#define TCPSYN_LEN  24      // TCP header length including MSS option
#define MAXPING_LEN 212     // Max length of Ping data

#define TFIN        0x01    // Option flags: no more data
#define TSYN        0x02    //           sync sequence nums
#define TRST        0x04    //           reset connection
#define TPUSH       0x08    //           push buffered data
#define TACK        0x10    //           acknowledgement
#define TURGE       0x20    //           urgent

#define TCP_MSS     1460    // Max Segment Size for TCP

#define ECHOPORT    7       // TCP Port numbers: echo
#define DAYPORT     13      //                   daytime
```

```
#define HTTPORT     80      //                  HTTP

/* end IP definitions */


#define MACLEN   6
BYTE myeth[MACLEN] = {0x00, 0x04, 0xa3, 0, 0, 0};
int next_page;
BYTE curr_rx_page;


typedef struct {                // NIC hardware packet header
    BYTE stat;                  //     Error status
    BYTE next;                  //     Pointer to next block
    WORD len;                   //     Length of this frame incl. CRC
} NICHEADER;

typedef struct {                // Ethernet frame header
    BYTE dest[MACLEN];          //     Dest & srce MAC addresses
    BYTE srce[MACLEN];
    WORD pcol;                  //     Protocol
} ETHERHEADER;

typedef struct {                // NIC and Ethernet headers combined
    NICHEADER nic;
    ETHERHEADER eth;
} NICETHERHEADER;

NICETHERHEADER nicin;           // Buffer for incoming NIC & Ether hdrs

/* Globals from driver.h */
BOOL checkflag;             // Checksum flag & byte values
BYTE checkhi, checklo;

BYTE ungot_byte;
BOOL ungot;
/* end driver.h globals */

WORD rxin, rxout;               // Rx buffer counts (NIC RAM is buffer)
BOOL atend;

#define TXBUFFLEN 64
BYTE txbuff[TXBUFFLEN];         // Tx buffer
int txin, txout;

/* Prototypes */
//void setnic_addr(WORD addr);
WORD init_ether(void);
WORD diagnostics(void);
WORD delay_ms(WORD mult);
void swapw(WORD *val);
void putnic_data(BYTE *data, WORD len);//
void setnic_addr(WORD addr);
void put_ether(void *data, WORD dlen); //
void xmit_ether(WORD dlen);
WORD get_ether(void);
void getnic_data(BYTE *data, int len);//
```

```
BYTE nicwrap(WORD page);
void check_byte(BYTE b);
BYTE getch_net(void);
BYTE getnic_byte(void);
BOOL match_byte(BYTE b);
BOOL get_word(WORD *w);
BOOL match_word(WORD w);
BOOL skip_lword(void);
BOOL skip_word(void);
BOOL get_lword(BYTE *lw);
BOOL match_lword(BYTE *lw);
void putch_net(BYTE b);
void put_word(WORD w);
void put_byte(BYTE b);
void put_data(BYTE *data, WORD len);//
void put_lword(BYTE *lw);
void discard_data(void);
void check_lword(BYTE *lw);
void put_ip(void);
BOOL skip_byte(void);
void copy_rx_tx(BYTE dest, BYTE srce, BYTE len);
void putnic_byte(BYTE b);
BOOL arp_recv(void);

/* from p16_ip.h from tcp book lib */
BYTE ipcol;                 // IP protocol byte
BYTE myip[4];               // My IP adress
BYTE locip[4], remip[4];    // Local & remote IP addresses

int locport, remport;      // ..and TCP port numbers
BYTE rseq[4], rack[4];      // TCP sequence & acknowledge values
int concount;              // Connection count (for high word of my seq num)
BYTE rflags, tflags;        // Rx and Tx flags
int rpdlen, tpdlen;        // Length of user data in Rx, Tx buffer
signed long iplen;          // Incoming/outgoing IP length word
BYTE d_checkhi, d_checklo;  // Checksum value for data
/* end tcp book lib */

/* Initialize card given driver type and base addr.
** Return driver type, 0 if error */
WORD init_ether(void)
{
    WORD ok=0, i;
    BYTE b;

    //reset_ether();
    //delay_ms(2);
    //NIC_RESET = 0;
    delay_ms(2);
    outnic(NE_RESET, innic(NE_RESET));  /* Do reset */
    delay_ms(2);
    if ((innic(ISR) & 0x80) == 0)       /* Report if failed */
    {
        print("NIC init error\r\n");
    }
    else
    {
```

```
        print("Ethernet card init successful!\r\n");
        outnic(CMDR, 0x21);                 /* Stop, DMA abort, page 0 */
        delay_ms(2);                        /* ..wait to take effect */
        outnic(DCR, DCRVAL);
        outnic(RBCR0, 0x00);                   /* Clear remote byte count */
        outnic(RBCR1, 0x00);
        outnic(RCR, 0x20);                  /* Rx monitor mode */
        outnic(TCR, 0x02);                  /* Tx internal loopback */
        outnic(TPSR, TXSTART);              /* Set Tx start page */
        outnic(PSTART, RXSTART);            /* Set Rx start, stop, boundary */
        outnic(PSTOP, RXSTOP);
        outnic(BNRY, (BYTE)(RXSTOP-1));
        outnic(ISR, 0xff);                  /* Clear interrupt flags */
        outnic(IMR, 0x00);                     /* Mask all interrupts */
        outnic(CMDR, 0x61);                 /* Stop, DMA abort, page 1 */
        delay_ms(2);
        for (i=0; i<6; i++)                 /* Set Phys addr */
            outnic(PAR0+i, myeth[i]);
        for (i=0; i<8; i++)                 /* Multicast accept-all */
            outnic(MAR0+i, 0xff);
        outnic(CURRP, RXSTART+1);           /* Set current Rx page */
        next_page = RXSTART + 1;
// Set LED 0 to be a 'link' LED, not 'collision' LED
// It would be nice if the following code worked, but the upper bits of the
// RTL config3 register are obstinately read-only, so it doesn't!
//        outnic(CMDR, 0xe0);                 /* DMA abort, page 3 */
//        outnic(RTL9346CR, 0xc0);            /* Write-enable config regs */
//        outnic(RTL3, 0x10);                 /* Enable 'link' LED */
//        outnic(RTL9346CR, 0x00);            /* Write-protect config regs */
        outnic(CMDR, 0x20);                 /* DMA abort, page 0 */
      //#if PROMISC
        //outnic(RCR, 0x14);                  /* Accept broadcasts and all packets!*/
      //#else
        outnic(RCR, 0x04);                  /* Accept broadcasts */
      //#endif
        outnic(TCR, 0x00);                     /* Normal Tx operation */
        outnic(ISR, 0xff);                  /* Clear interrupt flags */
        outnic(CMDR, 0x22);                 /* Start NIC */
        ok = 1;
    }
    return(ok);
}



WORD diagnostics(void) {
  //  outnic(CMDR, 0x21);               // stop AX88796
  delay_ms(2);

  outnic(CMDR,0x61);
  outnic(PSTART, 0x4E);

  outnic(CMDR,0x21);
  outnic(PSTOP, 0x3E);

  outnic(CMDR,0x61);
  if(innic(PSTART) != 0x4e)
```

```
    return 1;

  outnic(CMDR,0x21);                // switch to page 0
  if(innic(PSTOP) != 0x3e)
    return 2;

  if(innic(0x16*2) != 0x15)
    return 3;

  if(innic(0x12*2) != 0x0c)
    return 4;

  if(innic(0x13*2) != 0x12)
    return 5;

  return 0;
}

WORD delay_ms(WORD mult){
  WORD i;
  WORD delay = 5000000*mult;
  for(i=0; i<delay; i++);
  return 1;
}

/* Swap the bytes in a word */
void swapw(WORD *val)
{
  WORD temp_word;
  temp_word = *val;
    *val = (temp_word >> 8) | ((temp_word << 8) & 0xff00);
}

void putnic_data(BYTE *data, WORD len)
{
    len += len & 1;                     /* Round length up to an even value */
    outnic(ISR, 0x40);                  /* Clear remote DMA interrupt flag */
    outnic(RBCR0, len);                 /* Byte count */
    outnic(RBCR1, 0);
    outnic(CMDR, 0x12);                 /* Start, DMA remote write */
    while (len--)                        /* O/P bytes */
        outnic(DATAPORT, *data++);
    len = 255;                          /* Done: must ensure DMA complete */
    while (len && (innic(ISR)&0x40)==0)
        len--;
}

/* Set the 'remote DMA' address in the NIC's RAM to be accessed */
void setnic_addr(WORD addr)
{
    outnic(ISR, 0x40);                  /* Clear remote DMA interrupt flag */
    outnic(RSAR0, addr&0xff);           /* Data addr */
    outnic(RSAR1, addr>>8);
}

/* Send Ethernet packet given payload len */
void put_ether(void *data, WORD dlen)
```

```
{
    outnic(ISR, 0x0a);              /* Clear interrupt flags */
    setnic_addr(TXSTART<<8);
    putnic_data(nicin.eth.srce, MACLEN);
    putnic_data(myeth, MACLEN);
    swapw(&nicin.eth.pcol);
    putnic_data(&nicin.eth.pcol, 2);
    putnic_data(data, dlen);
}

/* Transmit the Ethernet frame */
void xmit_ether(WORD dlen)
{
    dlen += MACLEN+MACLEN+2;
    if (dlen < MINFRAME)
        dlen = MINFRAME;            /* Constrain length */
    outnic(TBCR0, dlen);            /* Set Tx length regs */
    outnic(TBCR1, dlen >> 8);
    outnic(CMDR, 0x24);             /* Transmit the packet */
}

/* Get packet into buffer, return length (excl CRC), or 0 if none available */

WORD get_ether(void)
{
    WORD len=0, curr;
    BYTE bound;
    WORD oset;
    WORD i;

    if (innic(ISR) & 0x10)              /* If Rx overrun.. */
    {
      //printf(" NIC Rx overrun ");
        init_ether();                   /* ..reset controller (drastic!) */
    }
    outnic(CMDR, 0x60);                 /* DMA abort, page 1 */
    curr = innic(CURRP);                /* Get current page */
    outnic(CMDR, 0x20);                 /* DMA abort, page 0 */
    if (curr != next_page)              /* If Rx packet.. */
    {
        //putnum(curr);
        //print(" ");

        //putnum(next_page);
        //print("\r\n");

        curr_rx_page = next_page;
        setnic_addr((WORD)next_page<<8);

      getnic_data((BYTE *)&nicin, sizeof(nicin));

      //print("######nicin: ");
      //putnum(nicin.nic.stat);
      //print(" ");
      //putnum(nicin.nic.next);
      //print(" ");
      //putnum(nicin.nic.len);
```

```
      //print(" ");
      //putnum(nicin.eth.dest);
      //print(" ");
      //putnum(nicin.eth.srce);
      //print(" ");
      //putnum(nicin.eth.pcol);
      //print("\r\n");

            swapw(&nicin.nic.len);//
            //swapw(&nicin.eth.pcol);//



        len = nicin.nic.len;            /* Take length from stored header */

        if ((nicin.nic.stat&1) && len>=MINFRAMEC && len<=MAXFRAMEC)
        {                               /* If hdr is OK, get packet */
            len -= MACLEN+MACLEN+2+CRCLEN;
            rxin = len;
            //swapw(&nicin.eth.pcol);
          //#if DEBUG
            //printf(" Rx%lu>", len);
          //#endif
        }
        else                            /* If not, no packet data */
        {
        //if((nicin.nic.stat&1))
          //{
            //print("stat ok\r\n");
          //}
        //if(len>=MINFRAMEC)
          //{
            //print("min ok\r\n");
          //}
        //if(len<=MAXFRAMEC)
          //{
            //print("max ok\r\n");
          //}

            len = 0;
            print("NIC pkt err \r\n");
        }

                                      /* Update next packet ptr */
        if (nicin.nic.next>=RXSTART && nicin.nic.next<RXSTOP)
            next_page = nicin.nic.next;
        else                            /* If invalid, use prev+1 */
        {
        //printf(" NIC ptr err ");
            next_page = nicwrap(next_page + 1);
            len = 0;
        }                               /* Update boundary register */
        bound = nicwrap(next_page - 1);
        outnic(BNRY, bound);
    }
    return(len);                        /* Return length excl. CRC */
}
```

```
/* Get data from NIC's RAM into the given buffer */
void getnic_data(BYTE *data, int len)
{
    BYTE b;

    outnic(ISR, 0x40);                  /* Clear remote DMA interrupt flag */
    outnic(RBCR0, len&0xff);                 /* Byte count */
    outnic(RBCR1, 0);
    outnic(CMDR, 0x0a);                 /* Start, DMA remote read */
    while (len--)                       /* Get bytes */
    {
        b = innic(DATAPORT);
      //print("** ");
        *data++ = b;
      //putnum(b);
      //print("\r\n");
    }
    //print("\r\n");
}

/* Wrap an NIC Rx page number */
BYTE nicwrap(WORD page)
{
   if (page >= RXSTOP)
       page += RXSTART - RXSTOP;
   else if (page < RXSTART)
       page += RXSTOP - RXSTART;
   return(page);
}


/*-----------------START IP PROTOCOLS---------------------*/

/* Handle an ARP message */
BOOL arp_recv(void)
{
    BOOL ret = 0;

   //if(nic

    //DEBUG_PUTC('a');
    if (match_byte(0x00) && match_byte(0x01) &&     // Hardware type
        match_byte(0x08) && match_byte(0x00) &&     // ARP protocol
        match_byte(6) &&  match_byte(4) &&          // Hardware & IP lengths
        match_word(ARPREQ) &&                       // ARP request
        skip_lword() && skip_word() &&              // Sender's MAC addr
        get_lword(remip) &&                        // Sender's IP addr
        skip_lword() && skip_word() &&              // Null MAC addr
        match_lword(myip))                         // Target IP addr (me?)
    {
      //DEBUG_PUTC('>');
      //print("0000000");
        ret = 1;
        txin = 0;
      put_word(0x0001);                            // Hardware type
        put_word(0x0800);                           // ARP protocol
```

```
        put_byte(6);                                  // Hardware & IP lengths
        put_byte(4);
        put_word(ARPRESP);                            // ARP response
        put_data(myeth, MACLEN);                      // My MAC addr
        put_lword(myip);                             // My IP addr
        put_data(nicin.eth.srce, MACLEN);             // Remote MAC addr
        put_lword(remip);                            // Remote IP addr
        put_ether(txbuff, txin);                      // Send to NIC
        xmit_ether(txin);                             // Transmit
        //DEBUG_PUTC('A');
    }
    else {
      print("out");
      discard_data();
      }
    return(ret);
}

/* Add byte to checksum value */
void check_byte(BYTE b)
{
    if (checkflag)
    {
        if ((checklo = b+checklo) < b)
        {
            if (++checkhi == 0)
                checklo++;
        }
    }
    else
    {
        if ((checkhi = b+checkhi) < b)
        {
            if (++checklo == 0)
                checkhi++;
        }
    }
    checkflag = !checkflag;
}

/* Get a byte from network buffer; if end, set flag */
BYTE getch_net(void)
{
    BYTE b=0;

    atend = rxout >= rxin;
//print("!1");
//putnum(!atend);
//print("2!");
    if (!atend)
    {
      //print("yea");
        b = getnic_byte();
      putnum(b);
      print("----\r\n");
        rxout++;
        check_byte(b);
```

```
    }
    return(b);
}

/* Return a byte from the NIC RAM */
BYTE getnic_byte(void)
{
    BYTE b;

    outnic(RBCR0, 1);                   /* Byte count */
    outnic(RBCR1, 0);
    outnic(CMDR, 0x0a);                 /* Start, DMA remote read */
    b = innic(DATAPORT);
    return(b);
}

/* Match an incoming byte value, return 0 not matched, or end of message */
BOOL match_byte(BYTE b)
{

    //return(b==getch_net() && !atend);
    BYTE a;
    a = getch_net();
    print("byte: ");
    putnum(a);
    print("\r\n");
    print("correct: ");
    putnum(b);
    print("\r\n");
    return(b==a && !atend);
}

/* Get an incoming word value, return 0 if end of message */
BOOL get_word(WORD *w)
{
    BYTE hi, lo;

    hi = getch_net();
    lo = getch_net();
    w = ((WORD)hi<<8) | (WORD)lo;
    return(!atend);
}

/* Match an incoming byte value, return 0 not matched, or end of message */
BOOL match_word(WORD w)
{
    WORD inw;

    return(get_word(inw) && inw==w);
}

/* Skip an incoming lword value, return 0 if end of message */
BOOL skip_lword(void)
{
    getch_net();
    getch_net();
    getch_net();
```

```
    getch_net();
    return(!atend);
}

/* Skip an incoming word value, return 0 if end of message */
BOOL skip_word(void)
{
    getch_net();
    getch_net();
    return(!atend);
}

/* Get an incoming lword value, return 0 if end of message */
//BOOL get_lword(LWORD *lw)
BOOL get_lword(BYTE *lw)
{
  lw[3] = getch_net();//
  lw[2] = getch_net();//
  lw[1] = getch_net();//
  lw[0] = getch_net();//
  //lw.b[3] = getch_net();
  //lw.b[2] = getch_net();
  //lw.b[1] = getch_net();
  //lw.b[0] = getch_net();
  return(!atend);
}

/* Match longword */
//BOOL match_lword(LWORD *lw)
BOOL match_lword(BYTE *lw)
{
  //return (match_byte(lw.b[3]) && match_byte(lw.b[2]) &&
  //    match_byte(lw.b[1]) && match_byte(lw.b[0]));
  return (match_byte(lw[3]) && match_byte(lw[2]) &&
        match_byte(lw[1]) && match_byte(lw[0]));
}

/* Put a byte into the network buffer */
void putch_net(BYTE b)
{
    if (txin < TXBUFFLEN)
        txbuff[txin++] = b;
    check_byte(b);
}

/* Send a word out to the SLIP link, then add to checksum */
void put_word(WORD w)
{
    putch_net(w >> 8);
    putch_net(w);
}

/* Send a byte to the network buffer */
void put_byte(BYTE b)
{
    putch_net(b);
}
```

```
void put_data(BYTE *data, WORD len)//
{
    while (len--)
        putch_net(*data++);
}

//void put_lword(LWORD *lw)
void put_lword(BYTE *lw)
{
  //putch_net(lw.b[3]);
  //putch_net(lw.b[2]);
  //putch_net(lw.b[1]);
  //putch_net(lw.b[0]);
    putch_net(lw[3]);
    putch_net(lw[2]);
    putch_net(lw[1]);
    putch_net(lw[0]);
}

/* Discard incoming data */
void discard_data(void)
{
    while (!atend)
        getch_net();
}

/* Respond to an ICMP message (e.g. ping) */
BOOL icmp_recv(void)
{
    BOOL ret=0;
    WORD csum;

    //DEBUG_PUTC('c');
    rpdlen = 0;
    if (match_byte(8) && match_byte(0) && get_word(csum))
    {
        while (skip_byte())                 // Check data
            rpdlen++;
        ret = (checkhi==0xff) && (checklo==0xff);
        if (ret && rpdlen<=MAXPING_LEN)
        {                                   // If OK and not bcast..
        //DEBUG_PUTC('>');
            checkhi = checklo = 0;          // Clear checksum
            put_ip();                       // IP header
            put_word(0);                    // ICMP type and code
            csum += 0x0800;                 // Adjust checksum for resp
            if (csum < 0x0800)              // ..including hi-lo carry
                csum++;
            put_word(csum);                 // ICMP checksum
            put_ether(txbuff, txin);        // Send ICMP response
            copy_rx_tx(txin, IPHDR_LEN+ICMPHDR_LEN, rpdlen);
            xmit_ether(IPHDR_LEN+ICMPHDR_LEN+rpdlen);
            //DEBUG_PUTC('I');
        }
    }
    return(ret);
```

```
}

//void check_lword(LWORD *lw)
void check_lword(BYTE *lw)
{
  //check_byte(lw.b[3]);
  //check_byte(lw.b[2]);
  //check_byte(lw.b[1]);
  //check_byte(lw.b[0]);
    check_byte(lw[3]);
    check_byte(lw[2]);
    check_byte(lw[1]);
    check_byte(lw[0]);
}

/* Send out an IP datagram header, given data length */
void put_ip(void)
{
    static BYTE id=0;

    txin = 0;
    checkhi = checklo = 0;          // Clear checksum
    checkflag = 0;
    put_byte(0x45);                 // Version & hdr len */
    put_byte(0);                    // Service
    put_word(iplen);
    put_byte(0);                    // Ident word
    put_byte(++id);
    put_word(0);                    // Flags & fragment offset
    put_byte(100);                  // Time To Live
    put_byte(ipcol);                // Protocol
    check_lword(myip);            // Include addresses in checksum
    check_lword(remip);
    put_byte(~checkhi);             // Checksum
    put_byte(~checklo);
    put_lword(myip);              // Source & destination IP addrs
    put_lword(remip);
}

/* Skip an incoming byte value, return 0 if end of message */
BOOL skip_byte(void)
{
    getch_net();
    return(!atend);
}

/* Copy a block from NIC Rx to Tx buffers (not crossing page boundaries) */
void copy_rx_tx(BYTE dest, BYTE srce, BYTE len)
{
    BYTE b;

    outnic(ISR, 0x40);                  /* Clear remote DMA interrupt flag */
    dest += sizeof(ETHERHEADER);
    srce += sizeof(NICETHERHEADER);
    while (len--)
    {
        outnic(RSAR0, srce);
```

```
        outnic(RSAR1, curr_rx_page);
        b = getnic_byte();
        outnic(RSAR0, dest);
        outnic(RSAR1, TXSTART);
        putnic_byte(b);
        srce++;
        dest++;
    }
}

/* Put the given byte into the NIC's RAM */
void putnic_byte(BYTE b)
{
    outnic(RBCR0, 1);                   /* Byte count */
    outnic(RBCR1, 0);
    outnic(CMDR, 0x12);                 /* Start, DMA remote write */
    outnic(DATAPORT, b);
}
/* EOF */
```

SYSTEM.MHS

```
#------------------------------
# CSEE 4840 Embedded System Design - System Constraints File
#
# Webserver
# Franklin Ma, Howard Wang, Victor Wang, William Wong
#
#------------------------------

# Parameters
PARAMETER VERSION = 2.1.0


#### PORTS

# Clock Port
PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN

# UART Ports
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN

# Ethernet Ports
PORT PB_D = PB_D, DIR = INOUT, VEC=[15:0]
PORT PB_A = PB_A, DIR = OUT,   VEC=[19:0]
PORT PB_OE_N = PB_OE_N, DIR = OUT
PORT PB_WE_N = PB_WE_N, DIR = OUT
PORT PB_UB_N = PB_UB_N, DIR = OUT
PORT PB_LB_N = PB_LB_N, DIR = OUT
PORT RAM_CE_N = RAM_CE_N, DIR = OUT

PORT ETHERNET_CS_N = ETHERNET_CS_N, DIR = OUT
PORT ETHERNET_IOCS16_N = ETHERNET_IOCS16_N, DIR = IN
PORT ETHERNET_RDY = ETHERNET_RDY, DIR = IN
```

```
PORT ETHERNET_IREQ = ETHERNET_IREQ, DIR = IN

#### PERIPHERALS

# Ethernet peripheral

BEGIN opb_ethernet
 PARAMETER INSTANCE = ethernet_peripheral
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00800000
 PARAMETER C_HIGHADDR = 0x00FFFFFF

 PORT OPB_CLK = sys_clk
 BUS_INTERFACE SOPB = myopb_bus

 PORT ETHERNET_CS_N = ETHERNET_CS_N
 PORT ETHERNET_RDY = ETHERNET_RDY
 PORT ETHERNET_IREQ = ETHERNET_IREQ
 PORT ETHERNET_IOCS16_N = ETHERNET_IOCS16_N

 PORT PB_D = PB_D
 PORT PB_A = PB_A
 PORT PB_OE_N = PB_OE_N
 PORT PB_WE_N = PB_WE_N
 PORT PB_UB_N = PB_UB_N
 PORT PB_LB_N = PB_LB_N
 PORT RAM_CE_N = RAM_CE_N

 PORT io_clock = io_clock
END

# The MICROBLAZE

BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 0
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
 PORT Interrupt = intr
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
END

# Interrupt Controller

BEGIN opb_intc
 PARAMETER INSTANCE = intc
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0xFFFF0000
 PARAMETER C_HIGHADDR = 0xFFFF00FF
 PORT OPB_Clk = sys_clk
 PORT Intr =  uart_intr
 PORT Irq =   intr
```

```
 BUS_INTERFACE SOPB = myopb_bus
END

# Block RAM for code and data is connected through two LMB busses
# to the Microblaze, which has two ports on it for just this reason.

# Data LMB bus

BEGIN lmb_v10
 PARAMETER INSTANCE = d_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_data_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001FFF
 BUS_INTERFACE SLMB = d_lmb
 BUS_INTERFACE BRAM_PORT = conn_0
END

# Instruction LMB bus

BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_instruction_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001FFF
 BUS_INTERFACE SLMB = i_lmb
 BUS_INTERFACE BRAM_PORT = conn_1
END

# The actual block memory

BEGIN bram_block
 PARAMETER INSTANCE = bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

# Clock divider to make the whole thing run

BEGIN clkgen
 PARAMETER INSTANCE = clkgen_0
 PARAMETER HW_VER = 1.00.a
 PORT FPGA_CLK1 = FPGA_CLK1
```

```
 PORT io_clock = io_clock
 PORT sys_clk = sys_clk
 PORT fpga_reset = fpga_reset
END

# The OPB bus controller connected to the Microblaze
# All peripherals are connected to this

BEGIN opb_v20
 PARAMETER INSTANCE = myopb_bus
 PARAMETER HW_VER = 1.10.a
 PARAMETER C_DYNAM_PRIORITY = 0
 PARAMETER C_REG_GRANTS = 0
 PARAMETER C_PARK = 0
 PARAMETER C_PROC_INTRFCE = 0
 PARAMETER C_DEV_BLK_ID = 0
 PARAMETER C_DEV_MIR_ENABLE = 0
 PARAMETER C_BASEADDR = 0x0fff1000
 PARAMETER C_HIGHADDR = 0x0fff10ff
 PORT SYS_Rst = fpga_reset
 PORT OPB_Clk = sys_clk
END

# UART: Serial port hardware

BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
 BUS_INTERFACE SOPB = myopb_bus
 PORT OPB_Clk = sys_clk
 PORT RX=RS232_RD
 PORT TX=RS232_TD
 PORT INTERRUPT = uart_intr
END
```

SYSTEM.MSS

```
#-----------------------------
# CSEE 4840 Embedded System Design - System Constraints File
#
# Webserver
# Franklin Ma, Howard Wang, Victor Wang, William Wong
#
#-----------------------------

PARAMETER VERSION = 2.2.0
PARAMETER HW_SPEC_FILE = system.mhs

BEGIN PROCESSOR
 PARAMETER HW_INSTANCE = mymicroblaze
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.00.a
```

```
END

BEGIN OS
  PARAMETER PROC_INSTANCE = mymicroblaze
  PARAMETER OS_NAME = standalone
  PARAMETER OS_VER = 1.00.a
  PARAMETER STDIN = myuart
  PARAMETER STDOUT = myuart
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = myuart
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = intc
 PARAMETER DRIVER_NAME = intc
 PARAMETER DRIVER_VER = 1.00.c
END

# Use null drivers for peripherals that don't need them
# This supresses warnings

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_data_controller
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_instruction_controller
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = ethernet_peripheral
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END
```

MAKEFILE

```
# Makefile for CSEE 4840

SYSTEM = system

MICROBLAZE_OBJS = \
      c_source_files/main.o   #c_source_files/ether.o

LIBRARIES = mymicroblaze/lib/libxil.a

ELF_FILE = $(SYSTEM).elf
```

```
NETLIST = implementation/$(SYSTEM).ngc

# Bitstreams for the FPGA

FPGA_BITFILE = implementation/$(SYSTEM).bit
MERGED_BITFILE = implementation/download.bit

# Files to be downloaded to the SRAM

SRAM_BINFILE = implementation/sram.bin
SRAM_HEXFILE = implementation/sram.hex

MHSFILE = $(SYSTEM).mhs
MSSFILE = $(SYSTEM).mss

FPGA_ARCH = spartan2e
DEVICE = xc2s300epq208-6

LANGUAGE = vhdl
PLATGEN_OPTIONS = -p $(FPGA_ARCH) -lang $(LANGUAGE)
LIBGEN_OPTIONS = -p $(FPGA_ARCH) $(MICROBLAZE_LIBG_OPT)

# Paths for programs

XILINX = /usr/cad/xilinx/ise6.2i
ISEBINDIR = $(XILINX)/bin/lin
ISEENVCMDS = LD_LIBRARY_PATH=$(ISEBINDIR) XILINX=$(XILINX) PATH=$(ISEBINDIR):$(PATH)

XILINX_EDK = /usr/cad/xilinx/edk6.2i
EDKBINDIR = $(XILINX_EDK)/bin/lin
EDKENVCMDS = LD_LIBRARY_PATH=$(ISEBINDIR):$(EDKBINDIR) XILINX=$(XILINX)
XILINX_EDK=$(XILINX_EDK) PATH=$(ISEBINDIR):$(EDKBINDIR):$(PATH)

MICROBLAZE = $(XILINX_EDK)/gnu/microblaze/lin
MBBINDIR = $(MICROBLAZE)/bin
XESSBINDIR = /usr/cad/xess/bin

# Executables

PLATGEN = $(EDKENVCMDS) $(EDKBINDIR)/platgen
LIBGEN = $(EDKENVCMDS) $(EDKBINDIR)/libgen

XST = $(ISEENVCMDS) $(ISEBINDIR)/xst
XFLOW = $(ISEENVCMDS) $(ISEBINDIR)/xflow
BITGEN = $(ISEENVCMDS) $(ISEBINDIR)/bitgen
DATA2MEM = $(ISEENVCMDS) $(ISEBINDIR)/data2mem
XSLOAD = $(XESSBINDIR)/xsload
XESS_BOARD = XSB-300E

MICROBLAZE_CC = $(MBBINDIR)/mb-gcc
MICROBLAZE_CC_SIZE = $(MBBINDIR)/mb-size
MICROBLAZE_OBJCOPY = $(MBBINDIR)/mb-objcopy

# External Targets

all :
      @echo "Makefile to build a Microprocessor system :"
```

```
	@echo "Run make with any of the following targets"
	@echo "  make libs     : Configures the sw libraries for this system"
	@echo "  make program  : Compiles the program sources for all the processor
instances"
	@echo "  make netlist  : Generates the netlist for this system ($(SYSTEM))"
	@echo "  make bits     : Runs Implementation tools to generate the bitstream"
	@echo "  make init_bram: Initializes bitstream with BRAM data"
	@echo "  make download : Downloads the bitstream onto the board"
	@echo "  make netlistclean: Deletes netlist"
	@echo "  make hwclean  : Deletes implementation dir"
	@echo "  make libsclean: Deletes sw libraries"
	@echo "  make programclean: Deletes compiled ELF files"
	@echo "  make clean    : Deletes all generated files/directories"
	@echo "  make cd       : Deletes some stuff and downloads"
	@echo " "
	@echo "  make <target> : (Default)"
	@echo "      Creates a Microprocessor system using default initializations"
	@echo "      specified for each processor in MSS file"


bits : $(FPGA_BITFILE)

netlist : $(NETLIST)

libs : $(LIBRARIES)

program : $(ELF_FILE)

init_bram : $(MERGED_BITFILE)

cd: clpart download

clpart:
	rm -f implementation/system.ngc implementation/ethernet_peripheral_wrapper.ngc
implementation/cache/ethernet_peripheral_wrapper.ngc

clean : hwclean libsclean programclean
	rm -f bram_init.sh platgen.log platgen.opt libgen.log
	rm -f _impact.cmd xflow.his

hwclean : netlistclean
	rm -rf implementation synthesis xst hdl
	rm -rf xst.srp $(SYSTEM)_xst.srp

netlistclean :
	rm -f $(FPGA_BITFILE) $(MERGED_BITFILE) \
	  $(NETLIST) implementation/$(SYSTEM)_bd.bmm

libsclean :
	rm -rf mymicroblaze/lib

programclean :
	rm -f $(ELF_FILE) $(SRAM_BITFILE) $(SRAM_HEXFILE)

#
# Software rules
#
```

```
MICROBLAZE_MODE = executable

# Assemble software libraries from the .mss and .mhs files

$(LIBRARIES) : $(MHSFILE) $(MSSFILE)
	$(LIBGEN) $(LIBGEN_OPTIONS) $(MSSFILE)

# Compilation

MICROBLAZE_CC_CFLAGS =
#MICROBLAZE_CC_OPT = -O3 #-mxl-gp-opt
MICROBLAZE_CC_OPT = -Os #-mxl-gp-opt
MICROBLAZE_CC_DEBUG_FLAG =#  -gstabs
MICROBLAZE_INCLUDES = -I./mymicroblaze/include/ # -I
MICROBLAZE_CFLAGS = \
	$(MICROBLAZE_CC_CFLAGS)\
	-mxl-barrel-shift \
	$(MICROBLAZE_CC_OPT) \
	$(MICROBLAZE_CC_DEBUG_FLAG) \
	$(MICROBLAZE_INCLUDES)

$(MICROBLAZE_OBJS) : %.o : %.c
	PATH=$(MBBINDIR) $(MICROBLAZE_CC) $(MICROBLAZE_CFLAGS) -c $< -o $@

# Linking

# Uncomment the following to make linker print locations for everything
MICROBLAZE_LD_FLAGS = -Wl,-M
//MICROBLAZE_LINKER_SCRIPT = -Wl,-T -Wl,mylinkscript
MICROBLAZE_LIBPATH = -L./mymicroblaze/lib/
MICROBLAZE_CC_START_ADDR_FLAG= -Wl,-defsym -Wl,_TEXT_START_ADDR=0x00000000
MICROBLAZE_CC_STACK_SIZE_FLAG=  -Wl,-defsym -Wl,_STACK_SIZE=0x200
MICROBLAZE_LFLAGS = \
	 -xl-mode-$(MICROBLAZE_MODE) \
	$(MICROBLAZE_LD_FLAGS) \
	$(MICROBLAZE_LINKER_SCRIPT) \
	$(MICROBLAZE_LIBPATH) \
	$(MICROBLAZE_CC_START_ADDR_FLAG) \
	$(MICROBLAZE_CC_STACK_SIZE_FLAG)

$(ELF_FILE) :  $(LIBRARIES) $(MICROBLAZE_OBJS)
	PATH=$(MBBINDIR) $(MICROBLAZE_CC) $(MICROBLAZE_LFLAGS) \
		$(MICROBLAZE_OBJS) -o $(ELF_FILE)
	$(MICROBLAZE_CC_SIZE) $(ELF_FILE)

#
# Hardware rules
#

# Hardware compilation : optimize the netlist, place and route

$(FPGA_BITFILE) : $(NETLIST) \
		etc/fast_runtime.opt etc/bitgen.ut data/$(SYSTEM).ucf
	cp -f etc/bitgen.ut implementation/
	cp -f etc/fast_runtime.opt implementation/
	cp -f data/$(SYSTEM).ucf implementation/$(SYSTEM).ucf
```

```
	$(XFLOW) -wd implementation -p $(DEVICE) -implement fast_runtime.opt \
		$(SYSTEM).ngc
	cd implementation; $(BITGEN) -f bitgen.ut $(SYSTEM)

# Hardware assembly: Create the netlist from the .mhs file

$(NETLIST) : $(MHSFILE)
	$(PLATGEN) $(PLATGEN_OPTIONS) -st xst $(MHSFILE)
	$(XST) -ifn synthesis/$(SYSTEM)_xst.scr
#	perl synth_modules.pl < synthesis/xst.scr > xst.scr
#	$(XST) -ifn xst.scr
#	rm -r xst xst.scr
#	$(XST) -ifn synthesis/$(SYSTEM).scr

#
# Downloading
#

# Add software code to the FPGA bitfile

$(MERGED_BITFILE) : $(FPGA_BITFILE) $(ELF_FILE)
	$(DATA2MEM) -bm implementation/$(SYSTEM)_bd \
	  -bt implementation/$(SYSTEM) \
	  -bd $(ELF_FILE) tag bram -o b $(MERGED_BITFILE)

# Create a .hex file with data for the SRAM

$(SRAM_HEXFILE) : $(ELF_FILE)
	$(MICROBLAZE_OBJCOPY) \
		-j .sram_text -j .sdata2 -j .sdata -j .rodata -j .data \
		-O binary $(ELF_FILE) $(SRAM_BINFILE)
	./bin2hex  -a 60000 < $(SRAM_BINFILE) > $(SRAM_HEXFILE)

# Download the files to the target board

download-sram : $(MERGED_BITFILE) $(SRAM_HEXFILE)
	$(XSLOAD) -ram -b $(XESS_BOARD) $(SRAM_HEXFILE)
	$(XSLOAD) -fpga -b $(XESS_BOARD) $(MERGED_BITFILE)

download : $(MERGED_BITFILE)
	$(XSLOAD) -fpga -b $(XESS_BOARD) $(MERGED_BITFILE)
```

SYSTEM.UCF

```
#----------------------------
# CSEE 4840 Embedded System Design - System Constraints File
#
# Webserver
# Franklin Ma, Howard Wang, Victor Wang, William Wong
#
#----------------------------

# NOTE: probably need some other clocks in here besides system clock
# the XSB manual says there is an exsiting 25 Mhz ethernet clock
# and CLKC is already 100 Mhz so why would we need another clock?
# ETHER-CLK connects to AX88796 pin 79
# also note that the PB
```

```
net sys_clk period = 20.000;
net io_clock period = 9.000;

net FPGA_CLK1 loc="p77";

net RS232_TD loc="p71";
net RS232_RD loc="p73";

# Ethernet
net ETHERNET_CS_N loc="p82";
net ETHERNET_RDY loc="p81";
net ETHERNET_IREQ loc="p75";
net ETHERNET_IOCS16_N loc="p74";


# OPB_ETHERNET
net PB_OE_N  loc="p125";
net PB_WE_N  loc="p123";
net PB_UB_N  loc="p146";
net PB_LB_N  loc="p140";

net RAM_CE_N loc="p147";

# OPB_Data
net PB_D<0> loc="p153";
net PB_D<1> loc="p145";
net PB_D<2> loc="p141";
net PB_D<3> loc="p135";
net PB_D<4> loc="p126";
net PB_D<5> loc="p120";
net PB_D<6> loc="p116";
net PB_D<7> loc="p108";
net PB_D<8> loc="p127";
net PB_D<9> loc="p129";
net PB_D<10> loc="p132";
net PB_D<11> loc="p133";
net PB_D<12> loc="p134";
net PB_D<13> loc="p136";
net PB_D<14> loc="p138";
net PB_D<15> loc="p139";

#OPB_Address
net PB_A<0> loc="p83";
net PB_A<1> loc="p84";
net PB_A<2> loc="p86";
net PB_A<3> loc="p87";
net PB_A<4> loc="p88";
net PB_A<5> loc="p89";
net PB_A<6> loc="p93";
net PB_A<7> loc="p94";
net PB_A<8> loc="p100";
net PB_A<9> loc="p101";
net PB_A<10> loc="p102";
net PB_A<11> loc="p109";
net PB_A<12> loc="p110";
```

```
net PB_A<13> loc="p111";
net PB_A<14> loc="p112";
net PB_A<15> loc="p113";
net PB_A<16> loc="p114";
net PB_A<17> loc="p115";
net PB_A<17> loc="p115";
net PB_A<18> loc="p121";
net PB_A<19> loc="p122";
```