SOBA Server: RTP Streaming Audio over Ethernet
Final Project Report ver. 1.0

Warriors:

Avraham Shinnar	as1619@columbia.edu
Benjamin Dweck	bjd2102@columbia.edu
Oliver Irwin	omi3@columbia.edu
Sean White	sw2061@columbia.edu

"...no pain no tears"
– *ASIX AX88796L Ethernet controller documentation*

May 10, 2005

Contents

List of Figures

1 Overview

This document describes the design of our system, development process, and advice for future project groups that want to develop similar systems. A full listing of our source code is included as Appendix A

The SOBA server is a streaming audio server based on the XESS XSB-300E FPGA board. We implemented it using IP's included in the Xilinx EDK 6.1 in addition to our own IP's, and some C code. The system encodes analog audio input into 16-bit samples and streams the digitized audio out the ethernet port in RTP packets, ready to be played in real time by an RTP client.

The system is comprised of three primary subsystems: the audio codec, the microblaze, and the ethernet controller. Each is described in further detail in the following sections.

2 System Design

2.1 General design and layout

Figure 1 shows a general block diagram for our system.

The system accepts line level audio via the AKM AK4565 audio codec where the signal is converted from analog to 16-bit signed digital samples. The samples are fetched from the audio codec by our audio sampler peripheral which generates an interrupt each time a complete sample arrives. The 32-bit Microblaze soft processor then transmits the sample from the audio sampler to the ethernet peripheral via the onboard peripheral bus (OPB). The ethernet peripheral places the samples in the ASIX AX88796L Ethernet controller's onboard RAM as the payload of a preloaded real-time transport protocol (RTP) packet. As each packet's payload is filled, it is transmitted onto the IP network as a UDP packet. An RTP client on the machine at the destination IP address receives the packets and plays the streaming audio in real time.

In addition the components mentioned above, we also designed, implemented and tested an audio control peripheral discussed below which is not included in our final design.

Address Space on the OPB Bus

Ethernet Controller:	`0x00800000` – `0x00FFFFFF`
Audio Sampler:	`0x10000100` – `0x100001FF`
UART:	`0xFEFF0100` – `0xFEFF01FF`
Interrupt Controller:	`0xFFFF0000` – `0xFFFF00FF`

2.2 Audio Codec

The AKM AK4565 Audio Codec has two separate intefaces: (1) a control interface used to configure the codec and (2) a transceiver interface used to send and recieve audio samples. The control interface is connected to the FPGA via the off-FPGA shared peripheral bus while the transceiver interface is connected to the FPGA via separate signal lines. We implemented two separate OPB peripherals to use the two interfaces. In our final design we did not include the audio control peripheral because the ethernet controller also uses the off-FPGA shared peripheral bus and our design did not require the audio control peripheral (we relied on the default codec values).

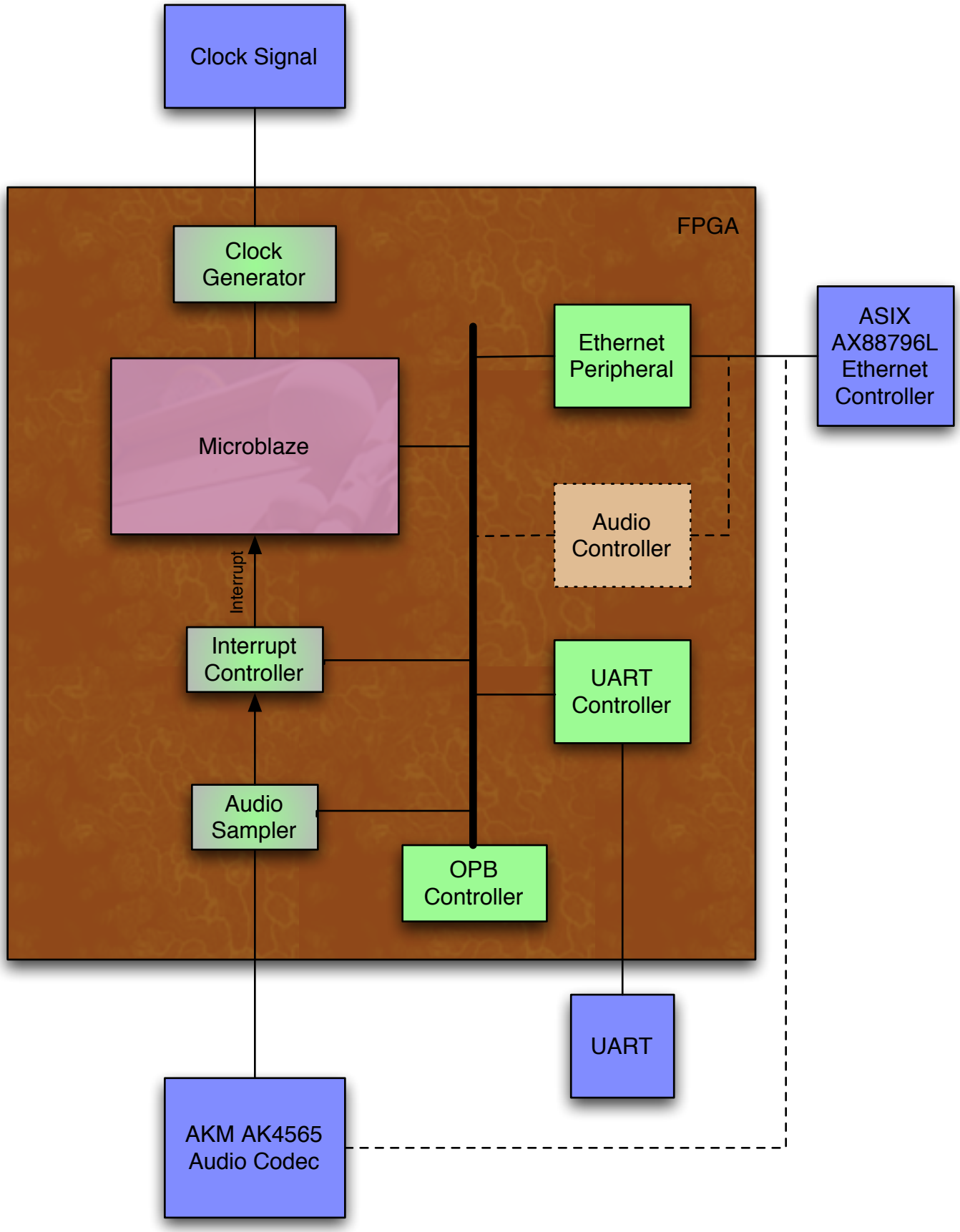

Figure 1: System Block Diagram

2.2.1 Audio Control Peripheral

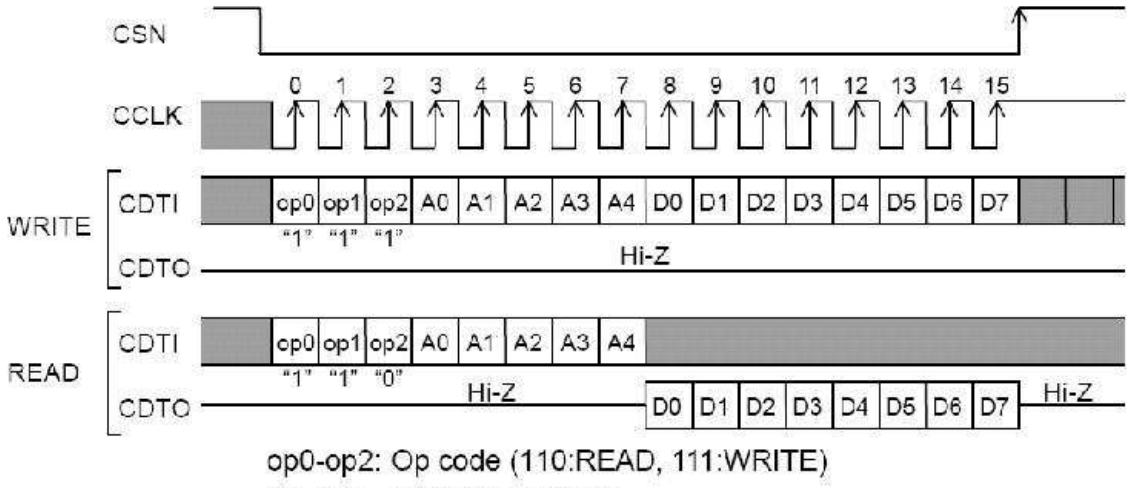

Figure 2: Audio Controller Timing Diagram

The timing diagram for the audio control signals is shown Figure 2. Our audio control peripheral, shown in Figure 3, takes read/write requests from the Microblaze on the OPB bus and translates them into read/write requests to specific registers in the audio codec. The audio control peripheral is allocated a byte's worth of address space on the OPB bus. The lowest address byte designates the addressed register on the audio codec.

2.2.2 Audio Sample Peripheral

The timing diagram for the audio transceiver signals is shown in Figure 4. Our audio sampler peripheral is only capable of reading samples from the codec and cannot transmit them to the codec as our design requires only receiving samples. Upon receiving a sample, the peripheral raises the interrupt signal which is connected to the interrupt controller. The interrupt controller then raises the interrupt line on the Microblaze which calls the BSP's general interrupt handler which, in turn, calls our audio sample interrupt handler. The handler fetches the sample from the audio sample peripheral at its address on the OPB bus. This location is read-only.

The block diagram for the audio sampler peripheral is shown in Figure 5. It generates the appropriate clocks for the audio codec from the OPB clock and continually shifts in bits from the codec, raising an interrupt each time it receives a complete sample.

2.3 Ethernet Codec

To use the ASIX AX88796L Ethernet controller, we created a peripheral based on the JAYCam design which included a memory controller, pads for i/o, and the ethernet controller itself. Usage consists of initializing the chip, filling the on-board SRAM with a packet skeleton, filling the payload, and initiating transmission. We also created a diagnostic program to verify that we could read and write registers on the chip.

The memory control maintains the state machine that manages signals to the Ethernet chip. The PAD I/O component is responsible for boosting the current of the lines that leave the FPGA on the PB so that they can drive chips on the shared PB. The Ethernet component incorporates the memory controller and

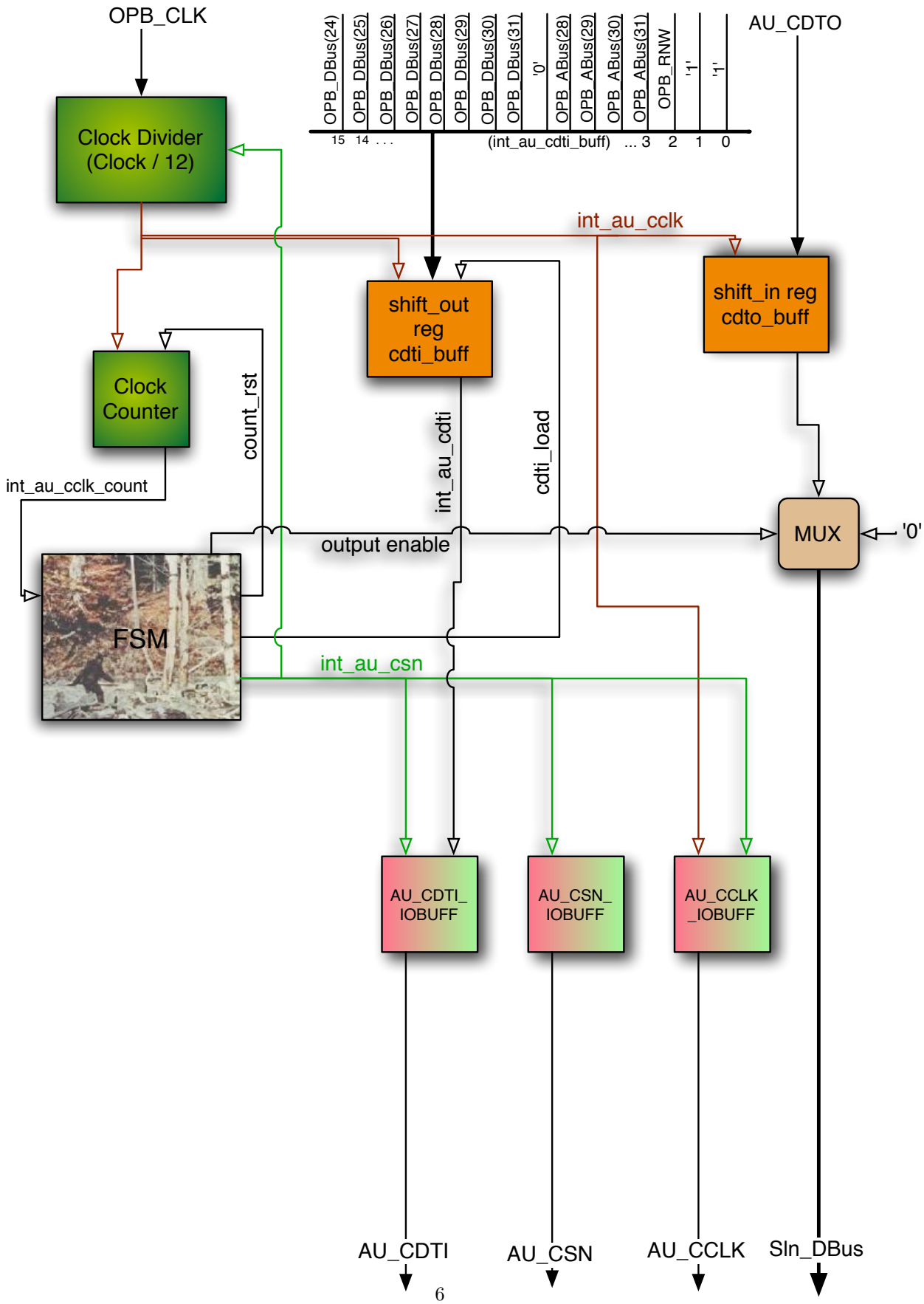

Figure 3: Audio Controller Block Diagram

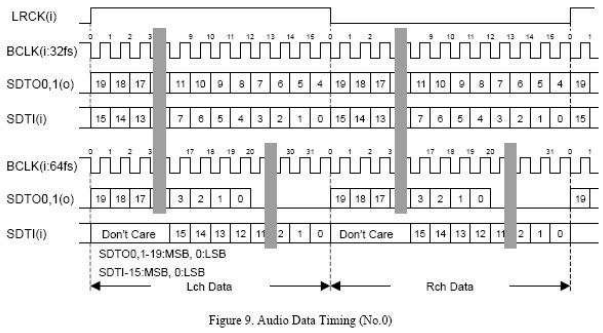

Figure 4: Audio Sampler Timing Diagram

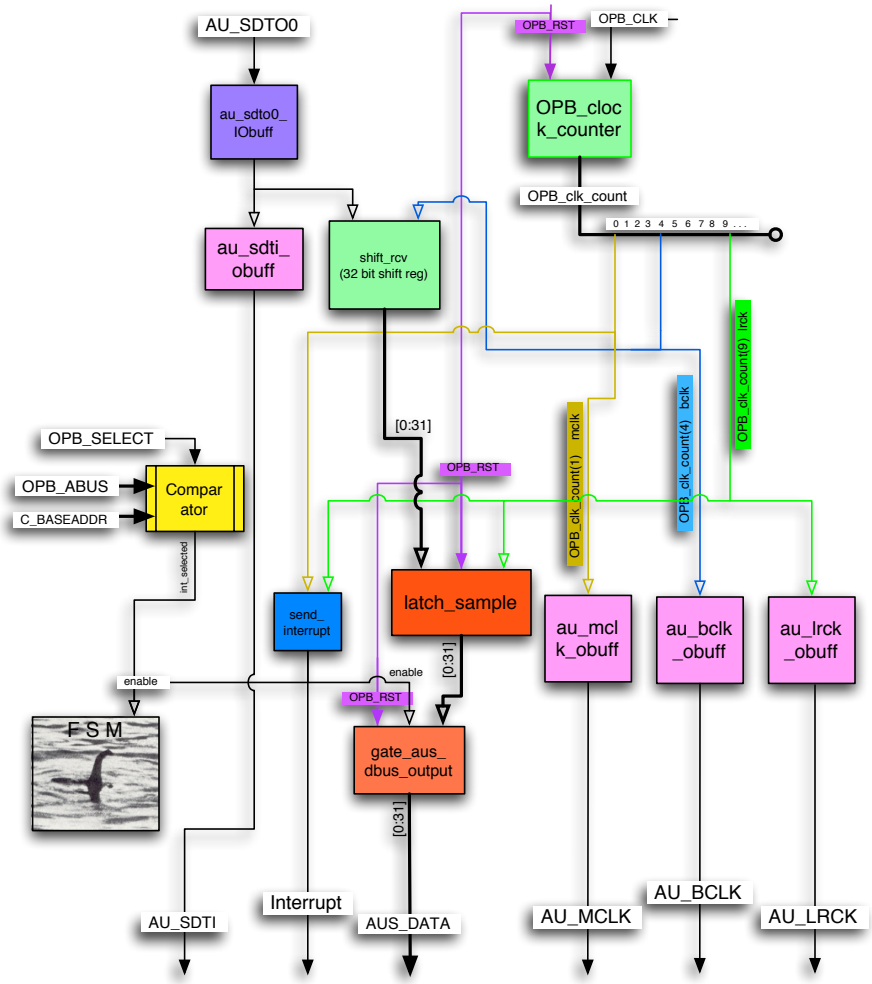

Figure 5: Audio Sampler Block Diagram

PAD I/O while maintaining the logic for managing the OPB peripheral selection and setting address line activity as well as data line direction based on OPB signals.

The timing diagram (Figure 7)and block diagram (Figure 6) for the Ethernet component are shown below. Our timing follows the timing diagram except we add an extra cycle between taking the chip select active and taking the IOWR active.

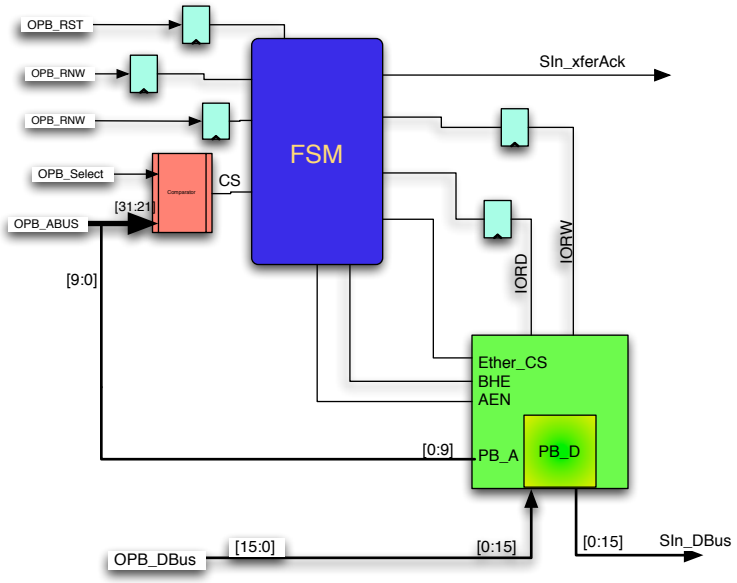

Figure 6: Ethernet Block Diagram

As described in the design document, we implemented diagnostic functionality so we could verify our ability to read and write registers on the Ethernet Chip and generally check the health of the chip. In particular, there are certain registers that have default values that can be checked to verify accurate reading of registers. See the diagnostics function in ether.c (Section A.1.2)

Our initialization process also closely followed our design document. The following steps were taken:

1. Write 21h to Command Register to abort current DMA operations.
2. Wait 2 milliseconds (timout for inter-frame gap timer)
3. Write 01h to Data Control Register to enable 16-bit word transfers.
4. Write 00h to both Remote Byte Count Registers to zero out DMA counter.
5. Write 00h to Interrupt Mask Register to mask interrupts.
6. Write ffh to Interrupt Status Register to clear interrupt flags
7. Write 20h to Receive Configuration Register to put NIC in monitor mode
8. Write 02h to Transmit Configuration Register to put NIC in loop-back mode
9. Set RX Start and Stop, Boundary, and TX start page.
10. Reset interrupt mask and flags.
11. Write 22h to Command Register to start NIC
12. Write 00h to Transmit Configuration Register to set normal transmit operation

Writing to registers involved writing directly to the memory location of the register based on OPB mapping. Writing data to the on-chip RAM required a remote DMA write which involved first setting the target address and then the target byte count in appropriate registers. Once this was done, the data was placed on the dataport a word at a time.

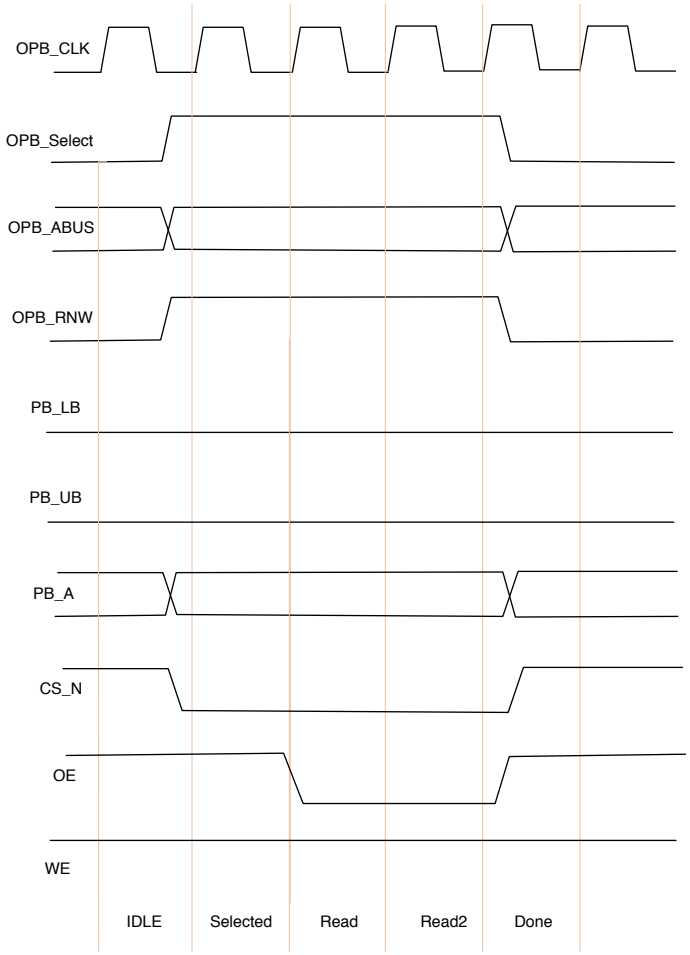

Figure 7: Ethernet Timing Diagram (for Read Only)

2.4 Microblaze

Our Microblaze code is used to initialize chips, interrupts, and stuff the default data into packet headers. Our code size was shrunk below the Microblaze 4K limit so we were able to keep all of our code on BRAM.

2.5 RTP Packet Structure and Construction

The packets sent via ethernet are RTP packets which are UDP/IP packets with an additional set of headers for streaming media. We write most of the IP, UDP, and RTP header data once to the Ethernet chip's on-board RAM during initialization and only update the payload, sequence number, and time stamp. We have code for updating the UDP checksum as well but decided not to use it to reduce the amount of code in the system.

The header structure is shown in Table 1.

	0 2 4 6 8 10 12 14	16 18 20 22 24 26 28 30
MAC_hdr	Ethernet destination address	
	Ethernet destination address	Ethernet source address
	Ethernet source address	
	Length / Type	Blank (no bits here)
ip_hdr	Version ip_v / IHL ip_hl / TOS ip_tos	Total Length ip_len
	Identification ip_id	Flags (see below) / Fragment Offset ip_off
	Time to Live ip_ttl / Protocol ip_proto	Header Checksum ip_sum
	Source Address ip_src	
	Destination Address ip_dst	
udp_hdr	Source Port udp_sport	Destination Port udp_dport
	Length udp_ulen	Checksum udp_sum
rtp_hdr	ver / P / X / CC / M / PT	Sequence Number
	Timestamp	
	SSRC	

Table 1: Header Structure for packets. Includes actual bit/byte values in many cases.

To view packets that we sent over Ethernet, we used Ethereal, an excellent network packet sniffing tool which can be found at `http://www.ethereal.com/`. To test reception of UDP packets, we used netcat on Linux. To test reception of RTP packets, we used rtpdump which can be found at `http://www.cs.columbia.edu/IRT/software/rtptools/`

2.6 RTP Client

We used a number of different ways to receive our RTP packets and play them. Initially, we used rtpdump and then played the audio using CoolEdit 2000. Once we were confident of the audio working, we moved to mplayer as our regular RTP client.

```
mplayer -rawaudio on:channels=2:samplesize=2:bitrate=48100 rtp://128.59.144.169:3042
```

3 Process

We started off by splitting up the work into two main components. Avi and Sean worked on the Ethernet peripheral and BJ and Oliver worked on the audio peripheral. We experienced significant difficulties developing our Ethernet peripheral from scratch and ended up basing our vhdl off the JAYCam project's Ethernet peripheral. Once the Ethernet peripheral was sending packets and the audio peripheral was digitizing audio, the two components were combined into a single system that utilized the Microblaze for control. Avi and Sean did the initial integration and then the team as a whole worked in shifts to get the client receiving audio and to verify proper packet construction.

4 Lessons Learned and Advice for Future Projects

Sean Initially, we wanted to create a pure hardware system that had no Microblaze and passed audio data directly from the audio peripheral to the Ethernet peripheral. However, time constraints and difficulties with the ethernet codec lead us to reduce our requirements such that the Microblaze became necessary for managing the initialization of peripherals and the passing of data between peripherals. I certainly learned that getting the Ethernet to work properly was more difficult than I thought. I'll keep this in mind when I next scope out an embedded systems project.

Some additional lessons learned:

Understand how the whole build system works. Take the time to read the docs and figure out what each file means and how it gets constructed. It'll save you time later on when you're trying to understand why a pin doesn't get connected.

Before writing any code, start out with pencil and paper. Write out your block diagrams and wave forms first and double check them twice. It helps you plan out how you'll pass data, which bits to pass, and how to structure your FSM. It also helps you communicate with other team members and anyone trying to provide advice about the project.

Compare your expected waveforms to reality: use the oscilloscope. We couldn't understand why we had problems with our Ethernet communication until we actually looked at the pins on the codec chip and saw that they weren't doing what we thought they were doing. Make sure your oscilloscope handles high frequencies so you don't miss anything.

Don't forget the jumpers for Ethernet projects or any other strange hardware requirements. The board we used required two jumpers in place before the ethernet chip would read our waveforms correctly. Most of the boards in the lab don't have them in place so make sure you have the hardware configured correctly.

Symphony provides a great free VHDL simulation tool for Linux and Windows. Use it to see if your waveforms are acting the way you expect them to act.

Do not believe data sheets if you can help it. Treat them as possible suggestions but don't trust them.

Watch your code size if you intend to put everything in BRAM. Our code grew larger than the allocated code space and we didn't realize it so only part of our code for the Microblaze was actually being executed. Strange behavior comes of this.

Avi Hardware is evil!!!

BJ This project gave me a good opportunity to learn VHDL. After spending weeks coding up the audio hardware and testing and debugging the VHDL, I feel very comfortable with VHDL. I also feel that I have a better idea of the timing issues involved in hardware and how to work with them. I have previously studied low-level programming and different hardware protocols but I have never before implemented my own hardware interface and controlled it with software. I now feel that I would be comfortable designing my own embedded system which was not the case before I took this course. My wife baught me a Virtex-4 based Xilinx board for my birthday last week so I hope to work on many fun embedded projects in the future!

Oliver The greatest thing I learned in embedded system was grand scheme of organization and implementation of an embedded system. VHDL was new to me before this class and now I feel I have a good handle on how a project needs to be coded, debugged, re-coded, debugged, again and again. I found it interesting to think about how fsm design allows settings to happen concurrently in the same clock signal. VHDL is a marriage between code and clock that exists on a level that I can only compare to gate level layout in VLSI. My work this semester was mostly accomplished by understanding the operation audio codec and proofreading VHDL code. I also tested pins of the board with the oscilloscope to debug and test codec output. This was the best way to initially see if our codec was operating reasonably. My understanding of UDP and TCP has also been born from this class, and I now understand the basic handling of headers and checksums.

5 Acknowledgements

Many thanks to Marcio Buss, Stephen Edwards, Josh Weinberg and JAYCam, Raj, and Christian for their advice.

A Code listing

A.1 C Source

A.1.1 ether.h

```
/*-------------------------------
# CSEE 4840 Embedded System Design
#
# SOBA Server
#
#  Team Warriors: Avraham Shinnar   as1619@columbia.edu
#                 Benjamin Dweck    bjd2102@columbia.edu
#                 Oliver Irwin      omi3@columbia.edu
#                 Sean White        sw2061@columbia.edu
#
-------------------------------*/

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

#define dprint print

#ifndef BYTE
#define BYTE unsigned char
#endif
#ifndef WORD
#define WORD unsigned short
#endif

// define the size of a packet

#define MAC_HEADER_SIZE 14
#define IP_HEADER_SIZE 20
#define UDP_HEADER_SIZE 8
#define RTP_HEADER_SIZE 12

#define HEADER_SIZE (MAC_HEADER_SIZE+IP_HEADER_SIZE+UDP_HEADER_SIZE+RTP_HEADER_SIZE)

#define IP_LENGTH (PACKET_SIZE-MAC_HEADER_SIZE)
#define UDP_LENGTH (IP_LENGTH-IP_HEADER_SIZE)

#define IP_CHECKSUM_OFFSET 24
#define UDP_CHECKSUM_OFFSET 40
#define RTP_SEQNUM_OFFSET 44
#define RTP_TIMESTAMP_OFFSET 46

#define DATA_CHUNK_SIZE 16
#define DATA_CHUNKS_IN_PACKET 90

#define PAYLOAD_SIZE (DATA_CHUNKS_IN_PACKET*DATA_CHUNK_SIZE)
#define PACKET_SIZE (PAYLOAD_SIZE+HEADER_SIZE)

#define PACKET_BUFFERS 1
```

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

// 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 CURR    (0x07*2) // current page reg for rd and wr
#define MAR0    (0x08*2) // multicast addr reg 0 for rd and WR

// Buffer Length and Field Definition Info
#define TXSTART  0x41              // Tx buffer start page
#define TXPAGES  8                 // Pages for Tx buffer
#define RXSTART  (TXSTART+TXPAGES) // Rx buffer start page
#define RXSTOP   0x7e              // Rx buffer end page for word mode


#define BASE_ADDR (XPAR_ETHERNET_PERIPHERAL_BASEADDR+0x400)
// macros for reading and writting registers
#define outnic(addr, data) XIo_Out16(NIC_BASE+addr, data)
#define innic(addr) (XIo_In16(NIC_BASE+addr))

#define PACKET_END_ADDRESS PACKET_START_ADDRESS+PACKET_SIZE
#define PACKET_START_ADDRESS  (TXSTART << 8)

#define PAYLOAD_START_ADDRESS PACKET_START_ADDRESS+HEADER_SIZE;
#define UDP_CHECKSUM_ADDRESS PACKET_START_ADDRESS+UDP_CHECKSUM_OFFSET
#define RTP_SEQNUM_ADDRESS PACKET_START_ADDRESS+RTP_SEQNUM_OFFSET


/* function prototypes, external interface. */
BYTE init();
BYTE output_sample(WORD *sample);
int diagnostics();
```

```
void wait(int mult);


/* symbolic names for various register bits */

#define ISR_PTX  0x02 // packet transmitted with no error
#define ISR_RXE  0x04 // recieve error
#define ISR_RDC  0x40 // Remote DMA Successful

#define CR_STOP  0x01
#define CR_START 0x02
#define CR_TXP   0x04
#define CR_READ  0x08
#define CR_WRITE 0x10
#define CR_ABORT 0x20
#define CR_COMPLETE 0x20
#define CR_PAGE0 0x00
#define CR_PAGE1 0x40
```

A.1.2 ether.c

```
/*-----------------------------------
# CSEE 4840 Embedded System Design
#
# SOBA Server
#
#  Team Warriors: Avraham Shinnar  as1619@columbia.edu
#                 Benjamin Dweck   bjd2102@columbia.edu
#                 Oliver Irwin     omi3@columbia.edu
#                 Sean White       sw2061@columbia.edu
#
-----------------------------------*/

#include "ether.h"

#include "xparameters.h"
#include "xbasic_types.h"
#include "xio.h"
#include "xintc_l.h"

/* function prototypes, internal helpers */
static BYTE send(WORD *data, WORD addr, WORD dmalen);
static BYTE transmit(WORD sendlen);

/* note that at one point we had lots more functions.  the code
   indicates where they were. Many of them were flattened for code
   space reasons (the compiler had some inlining issues, possibly due
   to out smashing the stack). */

/* static data */

/* stored without the one's complement */
/* note: ip checksum can actually be constant!! */
/* all the pseudo headers and most of the headers on the udp_checksum
   can be calculated in advance.  we just need to ask ethereal what it
```

```
   should be :).  since time_stamp and sequence number are increments,
   we can just add (with carries) these increments to the new initial
   values each new packet.  so all we need is the update stuff in append. */
/* static unsigned long udp_checksum; */

/* the real note: udp checksum can actually be (and now is) 0.  This
   means that it is ignored.  We originally wrote code to calculate a
   proper udp checksum (it is still in the code, just commented out).
   we later commented it out and decided to use 0 due to code space
   limitations.  (this also allowed us to put back in diagnostics).

   note to future people: smashing your stack is a bad idea.
 */
static WORD cur_payload_addr;

struct {
  WORD padding;
  WORD sequence_number;
  unsigned long time_stamp;
} dynamic_data;

/* #define UDP_INIT_CHECKSUM (~((WORD)(0xaa78))); */
/* #define UDP_INIT_CHECKSUM (~((WORD)(0xb15d))); */

static BYTE header[HEADER_SIZE] = {
  /* create the static portion of the packets */
  /* MAC header (14 bytes) */
  /*   Destination MAC address (6 bytes): fixed */
  /* 0-5 * 0x00,0x0C, 0xF1, 0x73, 0x4C, 0x9C,
   */
   /* Micro4 MAC */
     0x00,0x07,0xE9,0x43,0x8A,0x38,
  /*   Source MAC address (6 bytes): fixed */
  /* 6 */ 0x00,
  /* 7 */ 0x0D,
  /* 8 */ 0x60,
  /* 9 */ 0x7F,
  /* 10 */0xF9,
  /* 11 */0xAF,
  /*    Length (2 bytes): fixed */
  /* used as a type*/
  /* 12 */ 0x08,
  /* 13 */ 0x00,
  /*  IP header (20 bytes) */
  /*    Version (4 bits): fixed */
  /*    IHL (4 bits): fixed */
  /* 14 */ 0x45,
  /*    TOS (1 byte): fixed */
  /* 15 */ 0x00,
  /*    Total Length (2 bytes): fixed */
  /* 16 */ IP_LENGTH>>8,
  /* 17 */ IP_LENGTH & (0xff),
  /*    Identification (2 bytes): fixed */
```

```
/* 18 */ 0x00,
/* 19 */ 0x00,
/*    Flags (3 bits): fixed */
/*    Fragment Offset (13 bits): fixed */
/* 20 */ 0x00,
/* 21 */ 0x00,
/*    TTL (1 byte): fixed */
/* for now, this should die on contact*/
/* 22 */ 0x04,
/*    Protocol (1 byte): fixed */
/* UDP=17 */
/* 23 */ 0x11,
/*    Header Checksum (2 bytes): updated. see update_ip_checksum */
 // Micro4
    0x8A,0x63,
/*    Source IP (4 bytes): fixed */
/* 26 */ 0x80,
/* 27 */ 0x3B,
/* 28 */ 0x95,
/* 29 */ 0xA2,
/*    Destination IP (4 bytes): fixed */
// Micro4
    128,59,144,169,
///* 30 */ 128,
///* 31 */ 59,
///* 32 */ 144,
///* 33 */ 168,
/* UDP header (6 Bytes) */
/*   Source Port (2 bytes): fixed */
/* 34 */ 0x0B,
/* 35 */ 0xE2,
/*   Destination Port (2 bytes): fixed */
   /* in decimal: 3042 */
/* 36 */ 0x0B,
/* 37 */ 0xE2,
/*   Length (2 bytes): fixed */
/* 38 */ UDP_LENGTH>>8,
/* 39 */ UDP_LENGTH & 0xff,
/*   Checksum (2 bytes): updated */
/* 40 */ 0,
/* 41 */ 0,
/* RTP header (12 bytes) */
/*   Version (2 bits): fixed */
/* 10 */
/*   Padding (1 bit): fixed */
/* 0 */
/*   Extension (1 bit): fixed */
/* 0 */
/*   CSRC count (4 bits): fixed */
/* 0 */
/* 42 */ 0x80,
/*   Marker bit (1 bit): fixed */
/* 0 */
```

```
  /*   Payload type (7 bits): fixed */
  /* L16=10 (2 channel( (see
     http://www.networksorcery.com/enp/protocol/rtp.htm) */
  /* 43 */ 0x0a,
  /*   Sequence number (16 bits): updated */
  /* 44 */ 0x00,
  /* 45 */ 0x00,
  /*   Time stamp (32 bits): updated */
  /* 46 */ 0x00,
  /* 47 */ 0x00,
  /* 48 */ 0x00,
  /* 49 */ 0x00,
  /*   SSRC (32 bits): fixed */
  /* should be a random value: I don't think this is what is meant :)
     -- since we don't change our source transport address and don't
     handle multiple synchronization source within the same RTP
     channel, this should not be a problem. */
  /* 50 */ 0x42,
  /* 51 */ 0x42,
  /* 52 */ 0x42,
  /* 53 */ 0x42
  /*   CSRC list (0 bits): fixed */
  /* we are evil and don't give proper attribution to sources. */
};


/* a simple wait function that burns clock cycles. */
void wait(int mult){
  volatile int j=0, i=1000000;
  for(; i > 0; --i) {
    for(; mult > 0; --mult) {
      // a smart compiler with aggresive inliner should unroll this
      ++j;
    }
  }
}

/*
   INITIALIZATION
*/

/* Diagnostics based on page 31 of the Ethernet Controller manual
   Write on two pages first and then read to avoid being fooled by
   data latched in both write and read.
*/

int diagnostics() {
  //  outnic(CMDR, 0x21);                  // stop AX88796
  wait(10);

  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;
}

/* initializes the ethernet card and our data structures. sets up the
   static part of the packet header in memory. */
BYTE init() {
  int ret;

  outnic(NE_RESET, innic(NE_RESET));  // trigger a reset
  wait(2);
  if ((innic(ISR) & 0x80) == 0)       // Report if failed
    {
      print("  Ethernet card failed to reset!\r\n");
      print("Ethernet NIC not present or not initializing correctly\r\n");
      return 1;
    }
  else
    {
      dprint("Ethernet card reset successful!\r\n");
      /*      ether_reset(); */ /* Reset Ethernet card, */

      /* Write 21h to Command Register to abort current DMA operations. */
      outnic(CMDR, 0x21);
      /* Wait 2 milliseconds (timout for inter-frame gap timer) */
      wait(10);
      /* Write 01h to Data Control Register to enable 16-bit word transfers. */
      outnic(DCR, 0x01);
      /* Write 00h to both Remote Byte Count Registers to zero out DMA counter. */
      outnic(RBCR0, 0x00);
      outnic(RBCR1, 0x00);
      /* Write 00h to Interrupt Mask Register to mask interrupts. */
      outnic(IMR, 0x00);
      /* Write ffh to Interrupt Status Register to clear interrupt flags */
```

```
      outnic(ISR, 0xff);
      /* Write 20h to Receive Configuration Register to put NIC in monitor mode */
      outnic(RCR, 0x20);
      /* Write 02h to Transmit Configuration Register to put NIC in loop-back mode */
      outnic(TCR, 0x02);
      /* Set RX Start and Stop, Boundary, and TX start page. */
      outnic(PSTART, RXSTART);
      outnic(PSTOP, RXSTOP);
      outnic(BNRY, (BYTE)(RXSTOP-1));
      outnic(TPSR, TXSTART);
      /* Reset interrupt mask and flags. */
      outnic(ISR, 0xFF);    // clear interrupt status register
      outnic(IMR, 0x00);    // Mask completion irq
      /* Write 22h to Command Register to start NIC */
      outnic(CMDR, 0x22);
      /* Write 00h to Transmit Configuration Register to set normal transmit operation */
      outnic(TCR, 0x00);

      dprint("Ethernet card intialization complete!\r\n");
    }

  /* init_packet_setup(); */
  dynamic_data.sequence_number = 0;
  dynamic_data.time_stamp = 0;

/*    udp_checksum = UDP_INIT_CHECKSUM; */
  cur_payload_addr = PAYLOAD_START_ADDRESS;


  /* now lets initialize the buffers. */
  return send((WORD *)header, PACKET_START_ADDRESS, 54);//HEADER_SIZE);
}


/* actually transmits a (filled in) packet. */
static BYTE transmit(WORD sendlen)
{
  int j;
  outnic(TPSR, TXSTART); // set Transmit Page Start Register
  outnic(TBCR0, (sendlen&0xff)); // set Transmit Byte Count
  outnic(TBCR1, (sendlen>>8));

  outnic(CMDR, CR_COMPLETE|CR_TXP); // start transmission
  j=1000;
  while(j-->0 && !(innic(ISR) & ISR_PTX));
  if(!j){
    return 1;
  }

  outnic(ISR,0xFF);
  return 0;
}
```

```
/* low level function that does dma to send data to card. */
/* based on code from Josh's group. */
static BYTE send(WORD *data, WORD addr, WORD dmalen){
  int i, j, h;
  WORD counter;
  WORD word;

  counter = dmalen>>1;

  outnic(RSAR0, (addr&0xff)); // set DMA starting address
  outnic(RSAR1, (addr>>8));

  outnic(ISR, 0xFF); // clear ISR

  outnic(RBCR0, (dmalen&0xff)); // set Remote DMA Byte Count
  outnic(RBCR1, (dmalen>>8));

  outnic(CMDR, CR_WRITE|CR_START); // start the DMA write - 0x12

  // change order of MS/LS since DMA
  // writes LS byte in 15-8, and MS byte in 7-0
  for(i=0; i<counter; i++){
    word = (data[i]<<8)|(data[i]>>8);
    outnic(DATAPORT, word);
  }

  if(!(innic(ISR) & ISR_RDC)){
    //    print("Data - DMA did not finish\r\n");
    return 1;
  }

  outnic(CMDR,CR_COMPLETE);

  return 0;
}

/* external interface */
/* takes a 2 word sample, and sends it out.  sends packet if needed */
/* 0 return is success */
BYTE output_sample(WORD *sample) {
  BYTE ret = 0;
/*   WORD check; */

  /*   ret = append(sample); */
   ret = send(sample, cur_payload_addr, 4);

   /* if(ret)
      return ret; */
  cur_payload_addr+=4;
  /* update checksums */

/*   udp_checksum += *(sample+1); */
```

```
/*    udp_checksum += *(sample); */
/*    while (udp_checksum>>16) */
/*      udp_checksum = (udp_checksum & 0xffff) + (udp_checksum >> 16); */

  if(cur_payload_addr >= PACKET_END_ADDRESS) {
    /* finalize_packet() */
    /* take the ones complement, as well as shoving it into a word */
/*      while (udp_checksum>>16) */
/*        udp_checksum = (udp_checksum & 0xffff) + (udp_checksum >> 16); */

/*      check = ~((WORD)(udp_checksum&0xffff)); */
/*      check = 0; */
/*      ret = send(&check, UDP_CHECKSUM_ADDRESS, 2); */
/*      if(ret) { */
/*        print("Error writing udp checksum\r\n"); */
/*        return ret; */
/*      } */
    ret = send(((WORD *)&dynamic_data)+1, RTP_SEQNUM_ADDRESS, 6);

    if(ret) {
      print("Error writing rtp header data\r\n");
    }

    if(ret) {
      print("Error finalizing packet. dropping.\r\n");
    } else {
      ret = transmit(PACKET_SIZE);
      //    print("probable success transmitting packet. happy! happy!\r\n");
    }

    /*      next_packet_setup(); */
    ++dynamic_data.sequence_number;
    dynamic_data.time_stamp += 20;

/*      udp_checksum = dynamic_data.sequence_number + (unsigned short)((dynamic_data.time_st
/*      udp_checksum += UDP_INIT_CHECKSUM; */
/*      while (udp_checksum>>16) */
/*        udp_checksum = (udp_checksum & 0xffff) + (udp_checksum >> 16); */
    cur_payload_addr=PAYLOAD_START_ADDRESS;
  }

  return ret;
}
```

A.1.3 main.c

```
/*-------------------------------
# CSEE 4840 Embedded System Design
#
# SOBA Server
#
#  Team Warriors: Avraham Shinnar  as1619@columbia.edu
#                 Benjamin Dweck   bjd2102@columbia.edu
#                 Oliver Irwin     omi3@columbia.edu
#                 Sean White       sw2061@columbia.edu
```

```
#
-------------------------------*/

#include "xparameters.h"
#include "xbasic_types.h"
#include "xio.h"
#include "ether.h"
#include "xintc_l.h"

unsigned long uart_out = 0;
BYTE new_data=0;

void audio_sampler_handler(void *callback)
{
  // microblaze_disable_interrupts();
  new_data=1;
  //microblaze_enable_interrupts();
}

int main()
{
  int ret;

   // FIRST: Run diagnostics, init ethernet controller, and stuff packets

   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()) {
     print("Ethernet NIC not present or not initializing correctly\r\n");
     return 1;
   }

   /* initializing interrupts */
   XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR, XPAR_AUDIO_SAMPLER_INTERRUPT_MASK);
   XIntc_mMasterEnable(XPAR_INTC_SINGLE_BASEADDR);
   XIntc_RegisterHandler(XPAR_INTC_SINGLE_BASEADDR, XPAR_INTC_AUDIO_SAMPLER_INTERRUPT_INTR
                         audio_sampler_handler, (void*)0);

    microblaze_enable_interrupts();

    for (;;) {
      /* new_data is set in the interrupt handler when new data is ready. */
      if(new_data == 1) {
```

```
        new_data = 0;

        /* grab the data from the audio controller*/
        uart_out = XIo_In32(XPAR_AUDIO_SAMPLER_BASEADDR);
        /* the standard audio format is little endian, so we swap the bytes:
         we have 4 bytes, they get reordered from 1234 to 2143. */
        uart_out = ((uart_out >> 24 & 0xff) << 16) |
          ((uart_out >> 16 & 0xff) << 24) |
          (uart_out >> 8 & 0xff) |
          ((uart_out & 0xff) << 8);

        /* output the new sample to the ethernet card */
        output_sample((WORD *)(&uart_out));
      }

    }
  return 0;
}
```

A.2 clkgen

A.2.1 data/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
```

A.2.2 data/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
```

A.2.3 hdl/verilog/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
```

A.3 Audio Controller

Note that this code, while functional, was not used because of the conflict with the ethernet device. This could be resolved by writing an intelligent bus master, but we did not have time.

A.3.1 data/opb_audio_cntlr_v2_1_0.pao

```
##############################################################################
#
# opb_audio_cntlr pao file
#
##############################################################################

lib opb_audio_cntlr_v1_00_a opb_audio_cntlr
```

A.3.2 data/opb_audio_cntlr_v2_1_0.mpd

```
##############################################################################
##
## Microprocessor Peripheral Definition
##
##############################################################################

BEGIN opb_audio_cntlr, IPTYPE = PERIPHERAL, EDIF=TRUE

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

## Generics for VHDL
PARAMETER c_baseaddr     = 0xFFFFFFFF, DT = std_logic_vector, MIN_SIZE = 0xFF
PARAMETER c_highaddr     = 0x00000000, DT = std_logic_vector
PARAMETER c_opb_awidth   = 32,         DT = integer
PARAMETER c_opb_dwidth   = 32,         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 AU_CSN       = "",           DIR=OUT,          IOB_STATE=BUF
PORT PB_D         = "",           DIR = INOUT,      VEC = [15:0], 3STATE=FALSE, IOB_STATE=BUF

END
```

A.3.3 data/opb_audio_cntlr.vhd

```
------------------------------------------------------------------------------
--
```

```
-- Audio Controller OPB Peripheral
--
-- Benjamin Dweck
-- bjd2102@columbia.edu
--
-- Oliver Irwin
-- omi3@columbia.edu
--
------------------------------------------------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------
--
-- Clock Divider
--
-- Notes: Clock output starts out low after reset
--
-- Used to divide OPB_Clk to generate AU_CCLK
--
------------------------------------------------------------------------------
```

```
library ieee;
use ieee.std_logic_1164.all;

entity clock_divider is

  generic (
    FACTOR : integer := 10);    -- Clock divide-by factor

  port (
    clk_in : in std_logic;      -- Input clock
    rst : in std_logic;         -- Reset signal
    clk_out : out std_logic);   -- Output clock

end clock_divider;

architecture arch of clock_divider is

  signal count : integer := 0;
  signal output : std_logic := '0';

begin

  clk_out <= output;

  divide: process(clk_in, rst)
  begin
    if (rst = '1') then
      output <= '0';
      count <= 0;
    elsif (clk_in'event and clk_in='1') then
```

```
      count <= count + 1;

      if (count = (FACTOR/2)-1) then
        output <= NOT output;
        count <= 0;
      end if;

    end if;
  end process divide;

end arch;
```

```
--------------------------------------------------------------------------------
--
-- 5-bit Unsigned Negative Edge Triggered Up Counter
-- with Asynchronous Reset
--
-- Used to keep track of number of AU_CCLK cycles
--
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
  port(
    clk, reset : in  std_logic;
    count      : out std_logic_vector(4 downto 0)
  );
end counter;

architecture arch of counter is

  signal tmp: std_logic_vector(4 downto 0);

begin

  process (clk, reset)
  begin
    if (reset = '1') then
      tmp <= "00000";
    elsif (clk'event and clk = '0') then
      tmp <= tmp + 1;
    end if;
  end process;

  count <= tmp;

end arch;
```

```
--------------------------------------------------------------------------------
--
-- 16-bit Negative Edge Triggered Right Shift Register
-- with Active High Load and Serial Out
--
-- Used to latch onto data to be output to the audio codec and
-- shift it out serially.
--
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity shift_out is
  port(
    C, ALOAD : in  std_logic;
    D        : in  std_logic_vector(15 downto 0);
    SO       : out std_logic
  );
end shift_out;

architecture arch of shift_out is

  signal tmp : std_logic_vector(15 downto 0);

begin

  process (C, ALOAD, D)
  begin
    if (ALOAD='1') then
      tmp <= D;
    elsif (C'event and C='0') then
      tmp <= '0' & tmp(15 downto 1);
    end if;
  end process;

  SO <= tmp(0);

end arch;
```

```
--------------------------------------------------------------------------------
--
-- 8-bit Positive Edge Triggered Right Shift Register
-- with Serial In and Parallel Out
--
-- Used to receive data serially from the audio codec
--
--------------------------------------------------------------------------------
```

```
library ieee;
use ieee.std_logic_1164.all;

entity shift_in is
  port(
    SI : in  std_logic;
    C  : in  std_logic;
    D  : out std_logic_vector(7 downto 0)
  );
end shift_in;

architecture arch of shift_in is

  signal tmp : std_logic_vector(7 downto 0);

begin

  process (C)
  begin
    if (C'event and C='1') then
      tmp <= SI & tmp(7 downto 1);
    end if;
  end process;

  D <= tmp;

end arch;
```

```
------------------------------------------------------------------------------
--
-- Audio Peripheral
--
------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity opb_audio_cntlr is

  generic (
    C_OPB_AWIDTH : integer                     := 32;
    C_OPB_DWIDTH : integer                     := 32;
    C_BASEADDR   : std_logic_vector(0 to 31) := X"00000000";
    C_HIGHADDR   : std_logic_vector(0 to 31) := X"FFFFFFFF");

  port (
    -- OPB Input Signals
    OPB_Clk      : in  std_logic;
    OPB_Rst      : in  std_logic;
    OPB_ABus     : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE       : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
```

```
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW      : in  std_logic;
    OPB_select   : in  std_logic;
    OPB_seqAddr  : in  std_logic;        -- Sequential Address

    -- OPB Output Signals
    Sln_DBus     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck   : out std_logic;        -- (unused)
    Sln_retry    : out std_logic;        -- (unused)
    Sln_toutSup  : out std_logic;        -- Timeout suppress
    Sln_xferAck  : out std_logic;        -- Transfer acknowledge

    -- Audio IO Signals
    AU_CSN       : out   std_logic;                      -- Audio Chip Select (Active LOW)
    PB_D         : inout std_logic_vector(15 downto 0);  -- Off-FPGA Peripheral
                                                         -- Data Bus

--    AU_CCLK      : out std_logic;        -- Audio Cntl Clock
--    AU_CDTI      : out std_logic;        -- Audio Cntl Data In  (TO Codec)
--    AU_CDTO      : in  std_logic);       -- Audio Cntl Data Out (FROM Codec)

end opb_audio_cntlr;


architecture behavioral of opb_audio_cntlr is

  -- Audio Address and Data Bus Widths
  constant AU_AWIDTH : integer := 8;    -- Width of audio controller address
  constant AU_DWIDTH : integer := 8;    -- Width of audio controller data word

  -- Internal buffer/utillity signals
  signal int_au_csn            : std_logic := '1';    -- Used as audio codec chip select (
  signal int_au_cclk           : std_logic;           -- Used as serial clock signal to au
  signal int_au_cclk_count     : std_logic_vector(4 downto 0);      -- Used to keep track
  signal int_au_cclk_count_rst : std_logic;                    -- To reset cclk counter
  signal int_au_cdti           : std_logic;                                  -- AU_CD
  signal int_au_cdti_buff      : std_logic_vector(15 downto 0);  -- Input to CDTI shift reg
  signal int_au_cdti_load      : std_logic;                                  -- CDTI
  signal int_au_cdto_buff      : std_logic_vector(AU_DWIDTH-1 downto 0);    -- Data to be

  -- Internal utillity signals
  signal int_selected      : std_logic;         -- Internal chip_select signal
  signal int_output_enable : std_logic;         -- Enable output from peripheral to OPB bus

  -- State constants
  -- Critical: Sln_xferAck is generated directly from state bit 0!
  constant STATE_BITS : integer := 2;
  constant Idle       : std_logic_vector(0 to STATE_BITS-1) := "00";
  constant Transfer   : std_logic_vector(0 to STATE_BITS-1) := "01";
  constant Ack        : std_logic_vector(0 to STATE_BITS-1) := "11";

  signal present_state, next_state : std_logic_vector(0 to STATE_BITS-1);
```

```
  -- Clock divider (used with OPB_Clk to generate AU_CCLK)
  component clock_divider is
    generic (FACTOR : integer := 10);     -- Divide 50 MHz OPB_Clk => 5 MHz AU_CCLK
    port (
      clk_in  : in  std_logic;
      rst     : in  std_logic;
      clk_out : out std_logic);
  end component;

  -- CCLK Counter
  component counter is
    port(
      clk   : in  std_logic;
      reset : in  std_logic;
      count : out std_logic_vector(4 downto 0));
  end component;

  -- Transmit Register (AU_CDTI)
  component shift_out is
    port (
      C, ALOAD : in  std_logic;
      D        : in  std_logic_vector(15 downto 0);
      SO       : out std_logic);
  end component;

  -- Receive Register (AU_CDTO)
  component shift_in is
    port (
      SI : in  std_logic;
      C  : in  std_logic;
      D  : out std_logic_vector(7 downto 0));
  end component;

  -- Output buffer for AU_CSN
  component OBUF_F_8
    port (
      O : out std_ulogic;               -- Out to Audio codec
      I : in  std_ulogic);              -- In from Audio codec
  end component;

  -- Tristate output buffer for AU_CCLK and AU_CDTI
  component IOBUF_F_8
    port (
      O : out std_ulogic;               -- Out from the buffer
      I : in  std_ulogic;               -- In to the buffer
      IO: inout std_logic;              -- In/Out to pin
      T : in  std_ulogic);              -- Active-low output enable
  end component;

begin  -- behavioral

  -- A / 6 Clock Divider to generate CCLK
  clkdiv: clock_divider
```

```
  generic map (FACTOR => 12)
  port map (OPB_Clk, int_au_csn, int_au_cclk);

-- CCLK Counter
ccounter: counter
  port map (int_au_cclk, int_au_cclk_count_rst, int_au_cclk_count);

-- Transmit Register
cdti_buff: shift_out
  port map (int_au_cclk, int_au_cdti_load, int_au_cdti_buff, int_au_cdti);

-- Receive Register
cdto_buff: shift_in
  port map (PB_D(2), int_au_cclk, int_au_cdto_buff);

-- Output buffers for output signals
au_csn_obuff: OBUF_F_8
  port map (
    O => AU_CSN,
    I => int_au_csn);

au_cclk_iobuff: IOBUF_F_8
  port map (
    O  => null,
    I  => int_au_cclk,
    IO => PB_D(0),
    T  => int_au_csn);

au_cdti_iobuff: IOBUF_F_8
  port map (
    O  => null,
    I  => int_au_cdti,
    IO => PB_D(1),
    T  => int_au_csn);

dbus_iobuff_gen: for i in 3 to 15 generate
  dbus_iobuff : IOBUF_F_8
    port map (
      O  => null,
      I  => '0',
      IO => PB_D(i),
      T  => int_au_csn);
end generate;

end generate dbus_iobuff_gen;
-- Produce chip select signal by decoding OPB address and checking OPB_select
int_selected <=
  '1' when OPB_select = '1' and
      OPB_ABus(0 to C_OPB_AWIDTH-AU_AWIDTH-1) =
      C_BASEADDR(0 to C_OPB_AWIDTH-AU_AWIDTH-1)
  else '0';

-- Unused outputs
```

```
Sln_errAck   <= '0';
Sln_retry    <= '0';
Sln_DBus(0 to C_OPB_DWIDTH-AU_DWIDTH-1) <= (others => '0');

-- Tie OPB_ABus and OPB_DBus to the AU_CDTI buffer
int_au_cdti_buff(0) <= '1';           -- Set Op Code bits
int_au_cdti_buff(1) <= '1';
int_au_cdti_buff(2) <= NOT OPB_RNW;
int_au_cdti_buff(3) <= OPB_ABus(31);  -- Set Register Address bits
int_au_cdti_buff(4) <= OPB_ABus(30);
int_au_cdti_buff(5) <= OPB_ABus(29);
int_au_cdti_buff(6) <= OPB_ABus(28);
int_au_cdti_buff(7) <= '0';
int_au_cdti_buff(8) <= OPB_DBus(31);   -- Set Control Data bits
int_au_cdti_buff(9) <= OPB_DBus(30);
int_au_cdti_buff(10) <= OPB_DBus(29);
int_au_cdti_buff(11) <= OPB_DBus(28);
int_au_cdti_buff(12) <= OPB_DBus(27);
int_au_cdti_buff(13) <= OPB_DBus(26);
int_au_cdti_buff(14) <= OPB_DBus(25);
int_au_cdti_buff(15) <= OPB_DBus(24);

-- Tie the 0'th state bit to Sln_xferAck
Sln_xferAck <= present_state(0);

-- Process to qualify OPB output using int_output_enable
register_opb_outputs: process (OPB_Rst, OPB_RNW, int_output_enable)
begin
  if OPB_Rst = '1' then
    Sln_DBus(C_OPB_DWIDTH-AU_DWIDTH to C_OPB_DWIDTH-1) <= (others => '0');
  else
    if int_output_enable = '1' and OPB_RNW = '1' then
      Sln_DBus(C_OPB_DWIDTH-AU_DWIDTH to C_OPB_DWIDTH-1) <= int_au_cdto_buff;
    else
      Sln_DBus(C_OPB_DWIDTH-AU_DWIDTH to C_OPB_DWIDTH-1) <= (others => '0');
    end if;
  end if;
end process register_opb_outputs;

-- Sequential part of the FSM
fsm_seq : process(OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then          -- Asynchronous reset to Idle state
    present_state <= Idle;
  elsif OPB_Clk'event and OPB_Clk = '1' then   -- Positive edge OPB_Clk
    present_state <= next_state;               -- advance to next state
  end if;
end process fsm_seq;

-- Combinational part of the FSM
fsm_comb : process(OPB_Clk, present_state, int_selected, int_au_cclk_count)
begin
```

```
case present_state is

  when Idle =>
    Sln_toutSup <= '0';             -- Disable Supress-Timeout
    int_au_csn <= '1';              -- Deselect Audio Codec
    int_au_cdti_load <= '1';        -- Open input to CDTI shift reg
    int_output_enable <= '0';       -- Disable output to Sln_DBus
    int_au_cclk_count_rst <= '1';   -- Reset AU_CCLK counter
    if int_selected = '1' then
      next_state <= Transfer;
    else
      next_state <= Idle;
    end if;

  when Transfer =>
      if int_selected = '1' then
        Sln_toutSup <= '1';             -- Supress OPB Timeout
        int_au_cdti_load <= '0';        -- Latch audio codec address/data from OPB
        int_au_cclk_count_rst <= '0';   -- Start counter

        if int_au_cclk_count = "10000" then  -- IF last (16th) CCLK cycle elapsed...
          int_au_csn <= '1';                 -- Deselect audio codec
          int_output_enable <= '1';          -- Enable Sln_DBus output
          next_state <= Ack;                 -- Ack transfer
        else                                 -- ELSE...
          int_au_csn <= '0';                 -- Keep audio codec selected
          int_output_enable <= '0';          -- Keep Sln_DBus output supressed
          next_state <= Transfer;            -- Continue transfering data to audio codec
        end if;

      else                          -- If deselected by OPB master => Idle
        Sln_toutSup <= '0';
        int_au_csn <= '1';
        int_au_cdti_load <= '1';
        int_output_enable <= '0';
        int_au_cclk_count_rst <= '1';
        next_state <= Idle;
      end if;

  when Ack =>                   -- Send ACK
    Sln_toutSup <= '1';         -- Keep timeout supressed
    int_au_csn <= '1';          -- Deselect audio codec
    int_au_cdti_load <= '0';
    int_au_cclk_count_rst <= '0';
    int_output_enable <= '1';   -- Enable OPB output
    next_state <= Idle;

  when others =>              -- ELSE => Idle
    Sln_toutSup <= '0';
    int_au_csn <= '1';
    int_au_cdti_load <= '1';
    int_output_enable <= '0';
    int_au_cclk_count_rst <= '1';
```

```
        next_state <= Idle;

    end case;

  end process fsm_comb;

end behavioral;
```

A.4 Audio Sampler

A.4.1 data/opb_audio_sampler_v2_1_0.pao

```
#-------------------------------
#- CSEE 4840 Embedded System Design - Audio Sampling OPB Peripheral (pao)
#-
#- SOBA Server
#-
#-  Team Warriors: Avraham Shinnar  as1619@columbia.edu
#-                 Benjamin Dweck   bjd2102@columbia.edu
#-                 Oliver Irwin     omi3@columbia.edu
#-                 Sean White       sw2061@columbia.edu
#-
#-------------------------------

##############################################################################
#
# opb_audio_sampler pao file
#
##############################################################################

lib opb_audio_sampler_v1_00_a opb_audio_sampler
```

A.4.2 data/opb_audio_sampler_v2_1_0.mpd

```
#-------------------------------
#- CSEE 4840 Embedded System Design - Audio Sampling OPB Peripheral (mpd)
#-
#- SOBA Server
#-
#-  Team Warriors: Avraham Shinnar  as1619@columbia.edu
#-                 Benjamin Dweck   bjd2102@columbia.edu
#-                 Oliver Irwin     omi3@columbia.edu
#-                 Sean White       sw2061@columbia.edu
#-
#-------------------------------

##############################################################################
##
## Microprocessor Peripheral Definition
##
## Peripheral: Audio Sampler
##
##############################################################################

BEGIN opb_audio_sampler, IPTYPE = PERIPHERAL, EDIF=TRUE

OPTION IMP_NETLIST = TRUE
```

```
# Bus Interface
BUS_INTERFACE BUS = SOPB, BUS_STD = OPB, BUS_TYPE = SLAVE

## Generics for VHDL
PARAMETER c_baseaddr      = 0xFFFFFFFF, DT = std_logic_vector, MIN_SIZE = 0xFF
PARAMETER c_highaddr      = 0x00000000, DT = std_logic_vector
PARAMETER c_opb_awidth    = 32,         DT = integer
PARAMETER c_opb_dwidth    = 32,         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 aus_dbus     = Sl_DBus,      DIR = OUT, VEC = [0:(c_opb_dwidth-1)],    BUS = SOPB
PORT aus_errack   = Sl_errAck,    DIR = OUT,                                BUS = SOPB
PORT aus_retry    = Sl_retry,     DIR = OUT,                                BUS = SOPB
PORT aus_toutsup  = Sl_toutSup,   DIR = OUT,                                BUS = SOPB
PORT aus_xferack  = Sl_xferAck,   DIR = OUT,                                BUS = SOPB

PORT Interrupt = "", DIR = OUT, SENSITIVITY = LEVEL_HIGH, SIGIS = INTERRUPT, INTERRUPT_PRIO

#PORT AU_CLK      = "",          DIR=IN,         SIGIS=CLK

PORT AU_MCLK     = "",          DIR=OUT,        IOB_STATE=BUF
PORT AU_LRCK     = "",          DIR=OUT,        IOB_STATE=BUF
PORT AU_BCLK     = "",          DIR=OUT,        IOB_STATE=BUF
PORT AU_SDTI     = "",          DIR=OUT,        IOB_STATE=BUF
PORT AU_SDTO0    = "",          DIR=IN,         IOB_STATE=BUF

END
```

A.4.3 hdl/vhdl/opb_audio_sampler_v2_1_0.vhd

```
------------------------------------
-- CSEE 4840 Embedded System Design - Audio Sampling OPB Peripheral
--
-- SOBA Server
--
--  Team Warriors: Avraham Shinnar  as1619@columbia.edu
--                 Benjamin Dweck   bjd2102@columbia.edu
--                 Oliver Irwin     omi3@columbia.edu
--                 Sean White       sw2061@columbia.edu
--
------------------------------------


------------------------------------------------------------------------------
--
-- 32-bit Positive Edge Triggered Left Shift Register
```

```
-- with Serial In and Parallel Out
--
-- Used to receive samples serially from the audio codec
--
----------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity shift_32 is
  port(
    SI : in  std_logic;
    C  : in  std_logic;
    D  : out std_logic_vector(31 downto 0)
  );
end shift_32;

architecture arch of shift_32 is

  signal tmp : std_logic_vector(31 downto 0);

begin

  process (C)
  begin
    if C'event and C='1' then
      tmp <= tmp(30 downto 0) & SI;
    end if;
  end process;

  D <= tmp;

end arch;
```

```
----------------------------------------------------------------------------------
--
-- 32-bit Positive Edge Triggered Latch
-- with Asynchronous Reset
--
-- Used to receive samples serially from the audio codec
--
----------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity latch_32 is
  port(
    R : in  std_logic;
    C : in  std_logic;
    D : in  std_logic_vector(31 downto 0);
```

```
    Q : out std_logic_vector(31 downto 0)
  );
end latch_32;

architecture arch of latch_32 is

  signal tmp : std_logic_vector(31 downto 0);

begin

  process (C, R)
  begin
    if R='1' then
      tmp <= X"00000000";
    elsif C'event and C='1' then
      tmp <= D;
    end if;
  end process;

  Q <= tmp;

end arch;
```

```
--
-- Audio Sampler Peripheral
--
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity opb_audio_sampler is

  generic (
    C_OPB_AWIDTH : integer                     := 32;
    C_OPB_DWIDTH : integer                     := 32;
    C_BASEADDR   : std_logic_vector(0 to 31) := X"00000000";
    C_HIGHADDR   : std_logic_vector(0 to 31) := X"FFFFFFFF");

  port (
    -- OPB Input Signals
    OPB_Clk      : in  std_logic;
    OPB_Rst      : in  std_logic;
    OPB_ABus     : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE       : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW      : in  std_logic;
    OPB_select   : in  std_logic;
    OPB_seqAddr  : in  std_logic;        -- Sequential Address
```

```
    -- OPB Output Signals
    AUS_DBus     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    AUS_errAck   : out std_logic;         -- (unused)
    AUS_retry    : out std_logic;         -- (unused)
    AUS_toutSup  : out std_logic;         -- Timeout suppress
    AUS_xferAck  : out std_logic;         -- Transfer acknowledge

    -- Interrupt
    Interrupt    : out std_logic;         -- Interrupt Signal

    -- Audio IO Signals
    AU_MCLK      : out std_logic;         -- Audio Chip Master Clock
    AU_LRCK      : out std_logic;         -- Audio Left/Right Channel Clock
    AU_BCLK      : out std_logic;         -- Audio Bit Clock
    AU_SDTI      : out std_logic;         -- Audio Data In (TO Codec)
    AU_SDTO0     : in  std_logic);        -- Audio Data Out 0 (FROM Codec)

end opb_audio_sampler;

architecture behavioral of opb_audio_sampler is

    -- Clock counter used to generate audio transmission clocks
    signal opb_clk_count : std_logic_vector(10 downto 0);

    -- Internal Utility Signals
    signal int_selected    : std_logic;    -- Decoded chip select signal
    signal int_aus_dbus_en : std_logic;    -- Enables output to AUS_DBus
    signal int_shift_out   : std_logic_vector(31 downto 0);    -- Output of shift-in regist
    signal int_sample      : std_logic_vector(31 downto 0);    -- Last received sample

    -- Internal Buffer Signals
    signal int_interrupt : std_logic;
    signal int_au_mclk   : std_logic;
    signal int_au_bclk   : std_logic;
    signal int_au_lrck   : std_logic;
    signal int_au_sdto0  : std_logic;

    -- State constants
    constant Idle : std_logic := '0';
    constant Xfer : std_logic := '1';

    signal present_state, next_state : std_logic;

    -- 32-bit Shift Left Register with Serial In and Parallel Out
    component shift_32 is
        port (
            SI : in  std_logic;
            C  : in  std_logic;
            D  : out std_logic_vector(31 downto 0));
    end component;

    -- 32-bit Latch with Asynchronous Reset
```

```
    component latch_32 is
        port (
            R : in  std_logic;
            C : in  std_logic;
            D : in  std_logic_vector(31 downto 0);
            Q : out std_logic_vector(31 downto 0));
    end component;

    -- Output buffer for FPGA outputs to audio codec
    component OBUF_F_8 is
      port (
        O : out std_ulogic;      -- Out to Audio codec
        I : in  std_ulogic);     -- In from Audio codec
    end component;

    component IBUF is
      port (
        I : in  std_logic;
        O : out std_logic);
    end component;

begin

    -- Receiving Shift Register
    shift_rcv : shift_32
        port map (SI => int_au_sdto0,
                  C  => int_au_bclk,
                  D  => int_shift_out);

    -- Sample Latch
    latch_sample : latch_32
        port map (R => OPB_Rst,
                  C => int_au_lrck,
                  D => int_shift_out,
                  Q => int_sample);

    -- AU_MCLK Buffer
    au_mclk_buff : OBUF_F_8
        port map (
          O => AU_MCLK,
          I => int_au_mclk);

    -- AU_BCLK Buffer
    au_bclk_buff : OBUF_F_8
        port map (
          O => AU_BCLK,
          I => int_au_bclk);

    -- AU_LRCK Buffer
    au_lrck_buff : OBUF_F_8
        port map (
          O => AU_LRCK,
          I => int_au_lrck);
```

```
-- AU_SDTI Buffer
au_sdti_buff : OBUF_F_8
    port map (
      O => AU_SDTI,
      I => int_au_sdto0);

-- AU_MCLK Buffer
au_sdto0_buff : IBUF
    port map (
      O => int_au_sdto0,
      I => AU_SDTO0);

-- Produce chip select signal by decoding OPB address and checking OPB_select
int_selected <= '1' when OPB_select = '1' and
                         OPB_ABus(0 to C_OPB_AWIDTH-9) =
                         C_BASEADDR(0 to C_OPB_AWIDTH-9)
                 else '0';


-- Tie unused OPB slave outputs
AUS_errAck <= '0';
AUS_retry <= '0';
AUS_toutSup <= '0';

-- Tie OPB ack to output enable signal
AUS_xferAck <= int_aus_dbus_en;

-- Tie unused audio output
--AU_SDTI <= '0';

-- Tie off-FPGA signals to internal buffer signals
INTERRUPT <= int_interrupt;

--int_au_sdto0 <= AU_SDTO0;

-- Generate Audio Transmission Clocks
int_au_mclk <= opb_clk_count(1);
int_au_bclk <= opb_clk_count(4);
int_au_lrck <= NOT opb_clk_count(9);

opb_clk_count_proc : process(OPB_Clk, OPB_Rst)
begin
    if OPB_Rst = '1' then
        opb_clk_count <= "00000000000";
    elsif OPB_Clk'event and OPB_Clk='1' then
        opb_clk_count <= opb_clk_count + 1;
    end if;
end process opb_clk_count_proc;

-- Gate output to AUS_DBus with int_aus_dbus_en
gate_aus_dbus_output : process (OPB_Rst, OPB_RNW, int_aus_dbus_en)
begin
```

```
    if OPB_Rst = '1' then
        AUS_DBus(0 to C_OPB_DWIDTH-1) <= (others => '0');
    else
        if int_aus_dbus_en = '1' then
            AUS_DBus(0 to C_OPB_DWIDTH-1) <= int_sample(C_OPB_DWIDTH-1 downto 0);
        else
            AUS_DBus(0 to C_OPB_DWIDTH-1) <= (others => '0');
        end if;
    end if;
end process gate_aus_dbus_output;

-- Send interrupt pulse upon receiving sample
-- SMALL HACK: Using int_au_mclk to reset the interrupt
send_interrupt : process (int_au_mclk, int_au_lrck, int_interrupt)
begin
    if int_au_mclk='1' then
        int_interrupt <= '0';
    elsif int_au_lrck'event and int_au_lrck='1' then
        int_interrupt <= '1';
    end if;
end process send_interrupt;

-- Sequential part of the FSM
fsm_seq : process(OPB_Clk, OPB_Rst)
begin
    if OPB_Rst = '1' then          -- Asynchronous reset to Idle state
        present_state <= Idle;
    elsif OPB_Clk'event and OPB_Clk = '1' then     -- Positive edge OPB_Clk
        present_state <= next_state;               -- advance to next state
    end if;
end process fsm_seq;

-- Combinational part of the FSM
fsm_comb : process (present_state, int_selected)
begin
    int_aus_dbus_en <= '0';

    case present_state is

        when Idle =>
            int_aus_dbus_en <= '0';
            if int_selected='1' then
                next_state <= Xfer;
            else
                next_state <= Idle;
            end if;

        when Xfer =>
            int_aus_dbus_en <= '1';
            next_state <= Idle;

        when others =>
            int_aus_dbus_en <= '0';
```

```
                next_state <= Idle;

        end case;
    end process fsm_comb;

end behavioral;
```

A.5 opb_ethernet

A.5.1 data/opb_ethernet_v2_1_0.pao

```
#-------------------------------
# CSEE 4840 Embedded System Design
#
# SOBA Server
#
#  Team Warriors: Avraham Shinnar  as1619@columbia.edu
#                 Benjamin Dweck  bjd2102@columbia.edu
#                 Oliver Irwin    omi3@columbia.edu
#                 Sean White      sw2061@columbia.edu
#
#
# opb_ethernet pao file
#
#-------------------------------


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

A.5.2 data/opb_ethernet_v2_1_0.mpd

```
#-------------------------------
# CSEE 4840 Embedded System Design
#
# SOBA Server
#
#  Team Warriors: Avraham Shinnar  as1619@columbia.edu
#                 Benjamin Dweck  bjd2102@columbia.edu
#                 Oliver Irwin    omi3@columbia.edu
#                 Sean White      sw2061@columbia.edu
#
#-------------------------------
## 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=FALSI
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 = "", RAM_CE_N, DIR = OUT, IOB_STATE=BUF

PORT io_clock = "", DIR=IN


END
```

A.5.3 hdl/vhdl/memoryctrl.vhd

```
--------------------------------
-- CSEE 4840 Embedded System Design
--
-- SOBA Server
--
--  Team Warriors: Avraham Shinnar  as1619@columbia.edu
--                 Benjamin Dweck   bjd2102@columbia.edu
--                 Oliver Irwin     omi3@columbia.edu
--                 Sean White       sw2061@columbia.edu
--
--------------------------------

-- 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;

           vreq : in std_logic;
           video_ce : out 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;

begin

process(rst, clk)
begin
        if rst = '1' 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_rb and not fullword and not eth_io)
                  or (r_rc)
                  or (r_weth2));

        end if;

end process;



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 (
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;

xfer <= r_xfer;
rres <= r_xfer;
ce0 <= r_rb or (r_rc and eth_io);
ce1 <= r_rb or r_rc;


-- the video state machine


-- smw: killed the video state machine and set everything to 0

videocycle_i <= (r_v1 or r_v0) and not (rd_req_i or wr_req_i);
--videocycle <= videocycle_i;
videocycle <= '0';
video_ce <= '0';
```

```
r_v0 <='0';
r_v1 <='0';
r_v2 <='0';


end Behavioral;
```

A.5.4 hdl/vhdl/opb_ethernet.vhd

```
----------------------------------
-- CSEE 4840 Embedded System Design
--
-- SOBA Server
--
--  Team Warriors: Avraham Shinnar  as1619@columbia.edu
--                 Benjamin Dweck   bjd2102@columbia.edu
--                 Oliver Irwin     omi3@columbia.edu
--                 Sean White       sw2061@columbia.edu
--
-- 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 video_req : std_logic;
signal video_ce : std_logic;
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 : 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;
            vreq : in std_logic;
            video_ce : out 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,
                        vreq => video_req,
                        video_ce => video_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);


        amuxsel <= videocycle;

        addr_mux <= video_addr when (amuxsel = '1') else (addr(20 downto 2) & (addr(1) or

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

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


-- prepare control signals

process(videocycle, be, addr(1), hihalf, rd_req, wr_req)
begin

if videocycle='1' then pb_bytesel <= "11";
  elsif bus_req='1'then
    if addr(1)='1' or hihalf='1' then
      pb_bytesel <= be(1 downto 0);
    else
      pb_bytesel <= be(3 downto 2);
    end if;
  else
    pb_bytesel <= "00";
end if;

end process;

pb_rd <= rd_req or videocycle;
pb_wr <= wr_req;
--pb_wr <= OPB_Clk;

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" and (bus_req = '1') else '0';
eth_io <= '1' when addr(22 downto 21) /= "00" 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;

-- the fun begins
-- 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;


        if OPB_Clk'event and OPB_Clk ='1' then
                if      video_ce = '1' then
                        video_data <= rdata;
                end if;
        end if;

end process;

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

Sln_xferAck <= xfer;


end Behavioral;
```

A.5.5 hdl/vhdl/pad_io.vhd

```
----------------------------------
-- CSEE 4840 Embedded System Design
--
-- SOBA Server
--
--  Team Warriors: Avraham Shinnar  as1619@columbia.edu
--                 Benjamin Dweck   bjd2102@columbia.edu
--                 Oliver Irwin     omi3@columbia.edu
--                 Sean White       sw2061@columbia.edu
--
----------------------------------

-- 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 <= "1111111111111111" when dataz ='1' else "0000000000000000";

-- address
aff : for i in 0 to 19 generate
        aff : FDCE port map (
                C => io_clock, CLR => rst,
      CE => io_half,
                D => pb_addr(i),
                Q => pb_addr_1(i));
end generate;

-- data
din_ce <= io_half xor rd_early;
dff : for i in 0 to 15 generate
        drff : FDPE port map (
                C => io_clock, PRE => rst,
      CE => din_ce,
                D => pb_dread_a(i),
                Q => pb_dread(i));

        dwff : FDPE port map (
                C => io_clock, PRE => rst,
      CE => io_half,
                D => pb_dwrite(i),
                Q => pb_dwrite_1(i));

        dtff : FDPE port map (
                C => io_clock, PRE => rst,
                CE => io_half,
                D => pb_tristate(i),
                Q => pb_tristate_1(i));

end generate;

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

weff : FDPE port map (
                C => io_clock, PRE => rst,
              CE => wr_ce,

              D =>  we_n,
                Q => pb_we_n1);

oe_n <= not pb_rd or (not io_half and rd_early);
rd_ce <= io_half or rd_early;
oeff : FDPE port map (
                C => io_clock, PRE => rst,
```

```
                CE => rd_ce,
                D => oe_n,
                Q => pb_oe_n1);

lb_n <= not pb_lb;
lbff : FDPE port map (
                C => io_clock, PRE => rst,
                CE => io_half,
                D => lb_n,
                Q => pb_lb_n1);

ub_n <= not pb_ub;
ubff : FDPE port map (
                C => io_clock, PRE => rst,
                CE => io_half,
                D => ub_n,
                Q => pb_ub_n1);

ramce_n <= not ram_ce;
ramceff : FDPE port map (
                C => io_clock,
                PRE => rst,
      CE => io_half,
                D => ramce_n,
                Q => ram_ce_n1);

ethce_n <= not ethernet_ce;
ethceff : FDPE port map (
                C => io_clock,
                PRE => rst,
      CE => io_half,
                D => ethce_n,
                Q => eth_ce_n1);


-- 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_1(i));
end generate;

dbuf : for i in 0 to 15 generate
        dbuf : IOBUF_F_24 port map (
                O => pb_dread_a(i),
                IO => PB_D(i),
                I => pb_dwrite_1(i),
                T => pb_tristate_1(i));
end generate;



end Behavioral;
```

A.5.6 hdl/vhdl/opb_ethernet.vhd.old

Note that this file represents our original attempt to get ethernet working without using JAYcam's code. We did not get it working, but felt that it might be useful to future groups (we did get parts of it functioning correctly). This code is not, however actually used in our project.

```
----------------------------------
-- CSEE 4840 Embedded System Design
--
-- SOBA Server
--
--  Team Warriors: Avraham Shinnar  as1619@columbia.edu
--                 Benjamin Dweck   bjd2102@columbia.edu
--                 Oliver Irwin     omi3@columbia.edu
--                 Sean White       sw2061@columbia.edu
--
-- our original attempt getting ethernet working.  not used.
----------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity opb_ethernet is
```

```
  generic (
    C_OPB_AWIDTH : integer                      := 32;
    C_OPB_DWIDTH : integer                      := 32;
    C_BASEADDR   : std_logic_vector(0 to 31) := X"00000000";
    C_HIGHADDR   : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_ETHERNET_DWIDTH : integer                    := 16;  -- Number of address lines on the
    C_ETHERNET_AWIDTH : integer                    := 10); -- Number of data lines on the RA

  port (
    OPB_Clk      : in  std_logic;
    OPB_Rst      : in  std_logic;
    OPB_ABus     : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE       : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW      : in  std_logic;
    OPB_select   : in  std_logic;
    OPB_seqAddr  : in  std_logic;         -- Sequential Address
    Sln_DBus     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck   : out std_logic;         -- (unused)
    Sln_retry    : out std_logic;         -- (unused)
    Sln_toutSup  : out std_logic;         -- Timeout suppress
    Sln_xferAck  : out std_logic;         -- Transfer acknowledge

    PB_D : inout std_logic_vector(C_ETHERNET_DWIDTH-1 downto 0);
    PB_A : out std_logic_vector(C_ETHERNET_AWIDTH-1 downto 0);
    PB_OE : out std_logic;                -- active low, IOR
    PB_WE : out std_logic;                -- active low, IOW
    PB_UB : out std_logic;                -- active low, BHE_N
    PB_LB : out std_logic;                -- active low, AE_N
    ETHERNET_CS_N : out std_logic;        -- chip select (active)
    ETHERNET_RDY : in std_logic;          -- insert wait state during RW when low
    ETHERNET_IREQ  : in std_logic;        -- interrupt request output
    ETHERNET_IOCS16_N : in std_logic;  -- indicates 16 bit address when low
    ether_clk : in std_logic);
end opb_ethernet;

architecture Behavioral of opb_ethernet is

  component OBUF_F_24
    port (
      O : out std_logic;                  -- the pin
      I : in std_logic);                  -- signal to pin
  end component;

  component IOBUF_F_24
    port (
      O : out std_logic;                  -- signal from pin
      I : in std_logic;                   -- signal to pin
      IO: inout std_logic;                -- the pin
      T : in std_logic);                  -- 1 = drive IO with O
  end component;

  signal abuf_input : std_logic_vector(0 to C_ETHERNET_AWIDTH-1);
```

```
  signal dbuf_output, dbuf_input : std_logic_vector(0 to C_ETHERNET_DWIDTH-1);
  signal tristate_control : std_logic;

  signal RNW : std_logic;
  signal chip_select : std_logic;
  signal output_enable : std_logic;

  type states is (Idle, Selected, Read0, Read1, Read2, Read3, Write1, Write2, Write3, Tran
  signal present_state, next_state : states;

  signal ETHERNET_CS_N_i : std_logic;
  signal PB_OE_i : std_logic;
  signal PB_WE_i : std_logic;
  signal ae_n_i : std_logic;
  signal bhe_n_i : std_logic;
  signal be      : std_logic_vector(0 to C_OPB_DWIDTH/8-1);
  signal fullword : std_logic;
  signal state_value : std_logic_vector(0 to 7);

  signal pb_read : std_logic;
  signal pb_write: std_logic;

begin

ADDRPADGEN: for i in 0 to C_ETHERNET_AWIDTH-2 generate
  addrpad : OBUF_F_24 port map (
    O => PB_A(C_ETHERNET_AWIDTH-1-i),
    I => abuf_input(i));
end generate;

  addr0pad : OBUF_F_24 port map (
    O => PB_A(0),
    I => '0' );



DATAPADGEN: for i in 0 to C_ETHERNET_DWIDTH-1 generate
  datapad : IOBUF_F_24 port map (
    O  => dbuf_output(i),
    IO => PB_D(C_ETHERNET_DWIDTH-1-i),
    I  => dbuf_input(i),
    T  => tristate_control);
end generate;

--     datapad9 : IOBUF_F_24 port map (
--     O  => dbuf_output(15),
--     IO => PB_D(0),
--     I  => '1',
--     T  => '0');

    ethernet_cs_n_pad : OBUF_F_24 port map (
    O => ETHERNET_CS_N,
    I => ETHERNET_CS_N_i);
```

```
    pb_oe_pad : OBUF_F_24 port map (
    O => PB_OE,
    I => PB_OE_i);

    pb_we_pad : OBUF_F_24 port map (
    O => PB_WE,
    I => PB_WE_i);


    pb_ub_pad : OBUF_F_24 port map (                    -- BHE_N
    O => PB_UB,
    I => bhe_n_i);


    pb_lb_pad : OBUF_F_24 port map (                    -- AE_N
    O => PB_LB,
    I => ae_n_i);

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

  state_value <= X"02" when present_state = Idle
              else X"03" when present_state = Selected
              else X"04" when present_state = Read0
              else X"05" when present_state = Read1
              else X"06" when present_state = Read2
              else X"07" when present_state = Read3
              else X"08" when present_state = Write1
              else X"09" when present_state = Write2
              else X"0A" when present_state = Write3
              else X"0B" when present_state = Transfer_done
              else X"0C";

  register_opb_inputs: process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      be <= "0000";
      dbuf_input <= (others => '0');
      abuf_input <= (others => '0');
      RNW <= '0';
    elsif OPB_Clk'event and OPB_Clk = '1' then
      be <= OPB_BE;
--      dbuf_input <= OPB_DBus(C_OPB_DWIDTH-C_ETHERNET_DWIDTH to C_ETHERNET_DWIDTH-1);
      dbuf_input <= OPB_DBus(C_OPB_DWIDTH-C_ETHERNET_DWIDTH to C_OPB_DWIDTH-3) & "11";
      abuf_input <= OPB_ABus(C_OPB_AWIDTH-C_ETHERNET_AWIDTH to C_OPB_AWIDTH-1);
--      abuf_input <= OPB_ABus(C_OPB_AWIDTH-1 downto C_OPB_AWIDTH-C_ETHERNET_AWIDTH);
--      abuf_input <= OPB_ABus(2 to 11);
      -- check and abuf input is getting the correct address data from the OPB_Bus
      -- also checked pins and the Ethernet controller is getting data on the
      -- data lines.
      -- this value should be only 10 bits at the top, not bits 18-29
      --abuf_input <= OPB_ABus(C_OPB_AWIDTH-3-(C_ETHERNET_AWIDTH-1) to C_OPB_AWIDTH-3);
      RNW <= OPB_RNW;
```

```
    end if;
  end process register_opb_inputs;

  register_opb_outputs: process (OPB_Clk, OPB_Rst, OPB_select, chip_select)
  begin
      if OPB_Rst = '1' then
        Sln_DBus(0 to C_ETHERNET_DWIDTH-1) <= (others => '0');
      elsif OPB_Clk'event and OPB_Clk = '1' then
        if (chip_select = '1') then
          Sln_DBus(0 to C_ETHERNET_DWIDTH-1) <= dbuf_output(0 to 15);
--          Sln_DBus(0 to C_ETHERNET_DWIDTH-1) <= dbuf_output(0 to 11)  & state_value(4 to
        else
          Sln_DBus(0 to C_ETHERNET_DWIDTH-1) <= (others => '0');
        end if;
      end if;
  end process register_opb_outputs;


  ethernet_pb_outputs: process (OPB_Rst,OPB_Clk, chip_select, pb_read, pb_write)
  begin  -- process
    if OPB_Rst = '1'  then
      ETHERNET_CS_N_i <= '1';
      ae_n_i <= '1';
      bhe_n_i <= '1';
      PB_OE_i <= '1';              -- active low
      PB_WE_i <= '1';             -- active low
    elsif OPB_Clk'event and OPB_Clk = '1' then
      if chip_select = '1' then
        ETHERNET_CS_N_i <= '0';
        ae_n_i <= '0';
        bhe_n_i <= '0';
      else
        ETHERNET_CS_N_i <= '1';
        ae_n_i <= '1';
        bhe_n_i <= '1';
      end if;

      if pb_read = '0' then
        PB_OE_i <= '0';
      else
        PB_OE_i <= '1';
      end if;
      if pb_write = '0' then
        PB_WE_i <= '0';
      else
        PB_WE_i <= '1';
      end if;

    end if;


  end process;
```

```
  -- Unused outputs
  Sln_errAck   <= '0';
  Sln_retry    <= '0';

    Sln_DBus(C_ETHERNET_DWIDTH to C_OPB_DWIDTH-1) <= (others => '0');

  chip_select <= -- OPB_select when OPB_Abus(31 downto 20)=X"FF1" else '0';

         '1' when OPB_select = '1' and
        OPB_Abus(0 to 11) =
            C_BASEADDR(0 to 11) else
         '0';

--  chip_select <= '0';

--          OPB_ABus(0 to C_OPB_AWIDTH-3-C_ETHERNET_AWIDTH) =
--          C_BASEADDR(0 to C_OPB_AWIDTH-3-C_ETHERNET_AWIDTH) else


  -- Sequential part of the FSM
  fsm_seq : process(OPB_Clk, OPB_Rst)
  begin
      if OPB_Rst = '1' then
        present_state <= Idle;
      elsif OPB_Clk'event and OPB_Clk = '1' then
      present_state <= next_state;
    end if;
  end process fsm_seq;



  -- Combinational part of the FSM
  fsm_comb : process(OPB_Rst, present_state, chip_select, OPB_Select, RNW)
  begin

   if OPB_Rst = '1' then
        tristate_control <='1';
        output_enable <= '0';
        next_state <= Idle;
        pb_read <= '1';             -- active low
        pb_write <= '1';             -- active low
   else
      case present_state is
        when Idle =>
          tristate_control <=RNW;
          output_enable <= '0';
          pb_read <= '1';             -- active low
          pb_write <= '1';             -- active low
          if chip_select = '1' then
           next_state <= Selected;
          else
            next_state <= Idle;
          end if;
```

```
when Selected =>
  tristate_control <= RNW;
  output_enable <= '0';
  pb_read <= not RNW;              -- active low
  pb_write <= RNW;               -- active low
  if OPB_Select = '1' then
    if RNW = '1' then
       next_state <= Read0;
    else
       next_state <= Write1;
    end if;
  else
    next_state <= Idle;
  end if;

when Read0 =>                       -- we don't handle reads yet.
  tristate_control <= RNW;
  if OPB_Select = '1' then
     output_enable <= '0';
     pb_read <= '0';              -- active low
     pb_write <= '1';              -- active low
     next_state <= Read1;
   else
     pb_read <= '1';              -- active low
     pb_write <= '1';              -- active low
     output_enable <= '0';
     next_state <= Idle;
   end if;

when Read1 =>                       -- we don't handle reads yet.
  tristate_control <= RNW;
  if OPB_Select = '1' then
     output_enable <= '0';
     next_state <= Read2;
     pb_read <= '0';              -- active low
     pb_write <= '1';              -- active low
   else
     output_enable <= '0';
     next_state <= Idle;
     pb_read <= '1';              -- active low
     pb_write <= '1';              -- active low
   end if;--

when Read2 =>
  tristate_control <= RNW;
  if OPB_Select = '1' then
     pb_read <= '1';              -- active low
     pb_write <= '1';              -- active low
     output_enable <= '1';
     next_state <= Transfer_done;
   else
     pb_read <= '1';              -- active low
```

```
      pb_write <= '1';              -- active low
      output_enable <= '0';
      next_state <= Idle;
    end if;--


when Read3 =>
  tristate_control <=RNW;
  if OPB_Select = '1' then
      pb_read <= '0';              -- active low
      pb_write <= '1';              -- active low
      output_enable <= '1';
      next_state <= Transfer_done;
    else
      pb_read <= '1';              -- active low
      pb_write <= '1';              -- active low
      output_enable <= '0';
      next_state <= Idle;
    end if;--
          --
when Write1 =>
  next_state <= Write2;
  tristate_control <=RNW;
  output_enable <= '0';
  pb_read <= '1';              -- active low
  pb_write <= '0';              -- active low

when Write2 =>
  tristate_control <=RNW;
  output_enable <= '0';
    next_state <= Write3;
  pb_read <= '1';              -- active low
  pb_write <= '0';              -- active low
                                --
when Write3 =>
  next_state <= Transfer_done;
  tristate_control <=RNW;
  output_enable <= '0';
  pb_read <= '1';              -- active low
  pb_write <= '0';              -- active low

when Transfer_done =>
  tristate_control <=RNW;          -- 0
  next_state <= Idle;
  output_enable <= '0';
  pb_read <= '1';              -- active low
  pb_write <= '1';              -- active low

when others =>
  tristate_control <=RNW;
  next_state <= Idle;
  output_enable <= '0';
  pb_read <= '1';              -- active low
```

```
          pb_write <= '1';              -- active low


      end case;
    end if;
  end process fsm_comb;

  Sln_toutSup <= '0';                    -- -- tie this high in case we take too long
  Sln_xferAck <= '1' when present_state = Transfer_done else '0';


--  bhe_n_i <=  '0';                      -- UB

end Behavioral;

-- Local Variables:
-- compile-command: "ghdl -a opb_ethernet.vhd"
-- End:
```

A.6 misc

A.6.1 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

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 instanc
        @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 imp

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)
```

A.6.2 system.mhs

```
#-----------------------------
# CSEE 4840 Embedded System Design - System Constraints File
#
# SOBA Server
#
#  Team Warriors: Avraham Shinnar  as1619@columbia.edu
#                 Benjamin Dweck   bjd2102@columbia.edu
#                 Oliver Irwin     omi3@columbia.edu
#                 Sean White       sw2061@columbia.edu
#
#-----------------------------

# 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

# Audio Codec Ports

PORT AU_CSN = 0b1, DIR=OUT

PORT AU_MCLK  = AU_MCLK, DIR=OUT
PORT AU_LRCK  = AU_LRCK, DIR=OUT
PORT AU_BCLK  = AU_BCLK, DIR=OUT
PORT AU_SDTI  = AU_SDTI, DIR=OUT
PORT AU_SDTO0 = AU_SDTO0, 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
# PARAMETER C_BASEADDR = 0xFF100000
# PARAMETER C_HIGHADDR = 0xFF1FFFFF

 PORT OPB_CLK = sys_clk
# PORT ETHER_CLK = ETHER_CLK
# do we need to add an ethernet clock here?
 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

# Audio Sampler Peripheral

BEGIN opb_audio_sampler
 PARAMETER INSTANCE = audio_sampler
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x10000100
 PARAMETER C_HIGHADDR = 0x100001FF
 BUS_INTERFACE SOPB = myopb_bus
# PORT AU_CLK = audio_clk
 PORT OPB_Clk = sys_clk
 PORT AU_MCLK    = AU_MCLK
 PORT AU_LRCK    = AU_LRCK
 PORT AU_BCLK    = AU_BCLK
 PORT AU_SDTI    = AU_SDTI
 PORT AU_SDTO0   = AU_SDTO0
 PORT INTERRUPT = sampler_intr
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 & sampler_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 = 0x00000FFF
 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 = 0x00000FFF
 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
```

A.6.3 system.mss

```
#-----------------------------
# CSEE 4840 Embedded System Design - System Constraints File
#
# SOBA Server
#
#  Team Warriors: Avraham Shinnar  as1619@columbia.edu
#                 Benjamin Dweck   bjd2102@columbia.edu
#                 Oliver Irwin     omi3@columbia.edu
#                 Sean White       sw2061@columbia.edu
#
#-----------------------------
```

```
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

BEGIN DRIVER
 PARAMETER HW_INSTANCE = audio_sampler
```

```
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END
```

A.6.4 data/system.ucf

```
#-----------------------------------
# CSEE 4840 Embedded System Design - System Constraints File
#
# SOBA Server
#
#  Team Warriors: Avraham Shinnar  as1619@columbia.edu
#                 Benjamin Dweck   bjd2102@columbia.edu
#                 Oliver Irwin     omi3@columbia.edu
#                 Sean White       sw2061@columbia.edu
#
#-----------------------------------

# 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 pixel_clock period = 36.000;
net io_clock period = 9.000;
#net ICLK period = 30.000;

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";

# Audio Codec
net AU_CSN    loc="p165";
net AU_MCLK   loc="p167";
net AU_LRCK   loc="p168";
net AU_BCLK   loc="p166";
net AU_SDTI   loc="p169";
net AU_SDTO0  loc="p173";
```