ICC: Interactive Chat Client

Embedded Systems Design CSEE4840

May 11, 2004

Salman Baset
sa2086@columbia.edu

Bogdan Caprita
bc2008@columbia.edu

Sharon Price
sbp2001@columbia.edu

Table of Contents

1 Overview

1.1 What We Plan to Build

Whether through instant messaging (IM), internet relay chat (IRC), or one of the other protocols, most people have used some sort of chat client. We decided to build our own version, allowing people to send text messages through Xilinx boards.

1.2 How We Plan to Build Our Project

We plan to connect two boards together through their ethernet ports. Raw ethernet packets will be used for two reasons. First, none of the major benefits of TCP/IP or UDP are necessary – an occasional dropped packet, for instance, will not cause much of a problem. Secondly, implementing a TCP/IP stack would be more than one semester's project by itself and is therefore not feasible for this project.

A simple, traditional user interface will be displayed. The user will see a log of the conversation on the top portion of the screen, while the bottom few lines will contain the message currently being typed.

1.3 Expected Challenges

- Sending and receiving packets.
- Timing on the OPB.
- Setting up the video output.

2 Design Documents

2.1 Done in Hardware

We added the interface for the MicroBlaze processor needed to communicate with the AX88796 (ne2000 compatible) ethernet chip in hardware, in the OPBus. We set the ethernet chip in ISA bus mode. OPB has been designed so that it communicates with the ethernet chip in WORD mode. The ethernet chip offers BYTE or WORD mode, which can be selected by writing to the appropriate register.

We initially intended to use the VGA monitor attached to the board for video output. We wanted to create a chat client interface. It was however not possible to integrate the video VHDL code with the ethernet code due to the stringent requirements of the latter. We present our video design below for completeness, as it appears in our source code listing.

The video was done in hardware (displaying characters instead of pixels to speed up the frame buffer related processing).
We needed fast video that used little memory bandwidth so that it would work along with the stringent requirements of the ethernet chip.

The ethernet chip handles incoming packet buffering using a circular queue in the on-chip SRAM memory. To retrieve packet data and to prepare outgoing packets, we initiate in software DMA transfers to the on-chip SRAM. There was no need for hardware optimizations to the ethernet functionality. From 0x40 to 0x7F there are 256 bytes long pages (buffers) that can be used to write and read to the ethernet chip. The buffer space is divided into 2 blocks. One is only used for reading, and the other is only used for writing. The buffers we set up contains 6 pages for writing, and the remaining pages for reading. The choice was arbitrary. Essentially, we could use half of the on chip SRAM for reading, and half for writing.

The OPB contains an address multiplexor that selects between the video and the processor, giving priority to the video. Only the 32-bit addresses starting with binary 0000_0000_1 are considered for peripheral I/O. We use bits 20 to 1 from the address put on the address bus by the processor. That is why, when accessing ne2000 registers, we need to multiply the register addresses by 2 (shift left by 1 bit).

2.1.1 Memory Map

Video starts at 0x00800000.
Ethernet starts at 0x00A00400, which is the base I/O address of the NIC card.
From 0x00A00400 to 0x00A0041F the registers are mapped
From 0x4000 to 0x7FFF on-chip SRAM (not memory mapped)

The memory controller arbitrates among processor, video, and ethernet. Because the ethernet chip transfers are slow, we delayed these operations by 4 cycles:

```
r_reth1 <= opb_select and (r_common and rnw and eth_io);
r_reth2 <= opb_select and r_reth1;
r_reth3 <= opb_select and r_reth2;

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);
```

where r_common signals a memory I/O operation.

Video is delayed by 2 cycles, the amount of time that it takes to perform I/O on the frame buffer.

```
r_v0 <= videocycle_i;
r_v1 <= r_v0;
```

where video_cycle signals a request by the video controller for memory data.

We use a 100 MHz I/O clock, the 25 MHz Video clock, and the 50 MHz MicroBlaze system clock. The video clock had issues in being compiled onto the board.

2.1.2 Pin Connections

```
net sys_clk period = 18.000;
net pixel_clock period = 36.000;
net io_clock period = 9.000;
#net ICLK period = 30.000;

net FPGA_CLK1 loc="p77"; #use 100 MHz clock (old loc="p77")

net PB_A<0> loc="p83"; #BAR1
net PB_A<1> loc="p84"; #BAR2
net PB_A<2> loc="p86"; #BAR3
net PB_A<3> loc="p87"; #BAR4
net PB_A<4> loc="p88"; #BAR5
net PB_A<5> loc="p89"; #BAR6
net PB_A<6> loc="p93"; #BAR7
net PB_A<7> loc="p94"; #BAR8
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<18> loc="p121";
net PB_A<19> loc="p122";

net PB_D<0> loc="p153"; #LEFT_A
net PB_D<1> loc="p145"; #LEFT_B
net PB_D<2> loc="p141"; #LEFT_C
net PB_D<3> loc="p135"; #LEFT_D
net PB_D<4> loc="p126"; #LEFT_E
net PB_D<5> loc="p120"; #LEFT_F
net PB_D<6> loc="p116"; #LEFT_G
net PB_D<7> loc="p108"; #LEFT_DP
net PB_D<8> loc="p127"; #RIGHT_A
net PB_D<9> loc="p129"; #RIGHT_B
net PB_D<10> loc="p132"; #RIGHT_C
net PB_D<11> loc="p133"; #RIGHT_D
net PB_D<12> loc="p134"; #RIGHT_E
net PB_D<13> loc="p136"; #RIGHT_F
net PB_D<14> loc="p138"; #RIGHT_G
net PB_D<15> loc="p139"; #RIGHT_DP


net PB_LB_N loc="p140";      #BAR9
net PB_UB_N loc="p146";  #BAR10
net PB_WE_N loc="p123";
net PB_OE_N loc="p125";


net RAM_CE_N loc="p147";

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

net VIDOUT_CLK loc="p23";
net VIDOUT_BLANK_N loc="p24";
net VIDOUT_HSYNC_N loc="p8";
net VIDOUT_VSYNC_N loc="p7";


net VIDOUT_RCR<0> loc="p41";
net VIDOUT_RCR<1> loc="p40";
net VIDOUT_RCR<2> loc="p36";
net VIDOUT_RCR<3> loc="p35";
```

```
net VIDOUT_RCR<4> loc="p34";
net VIDOUT_RCR<5> loc="p33";
net VIDOUT_RCR<6> loc="p31";
net VIDOUT_RCR<7> loc="p30";
net VIDOUT_RCR<8> loc="p29";
net VIDOUT_RCR<9> loc="p27";

net VIDOUT_GY<0> loc="p9" ;
net VIDOUT_GY<1> loc="p10";
net VIDOUT_GY<2> loc="p11";
net VIDOUT_GY<3> loc="p15";
net VIDOUT_GY<4> loc="p16";
net VIDOUT_GY<5> loc="p17";
net VIDOUT_GY<6> loc="p18";
net VIDOUT_GY<7> loc="p20";
net VIDOUT_GY<8> loc="p21";
net VIDOUT_GY<9> loc="p22";

net VIDOUT_BCB<0> loc="p42";
net VIDOUT_BCB<1> loc="p43";
net VIDOUT_BCB<2> loc="p44";
net VIDOUT_BCB<3> loc="p45";
net VIDOUT_BCB<4> loc="p46";
net VIDOUT_BCB<5> loc="p47";
net VIDOUT_BCB<6> loc="p48";
net VIDOUT_BCB<7> loc="p49";
net VIDOUT_BCB<8> loc="p55";
net VIDOUT_BCB<9> loc="p56";

net RS232_TD loc="p71";
net RS232_RD loc="p73";
#net RS232_CTS loc="p69";
#net RS232_RTS loc="p70";

net AU_CSN_N loc="p165";
net AU_BCLK loc="p166";
net AU_MCLK loc="p167";
net AU_LRCK loc="p168";
net AU_SDTI loc="p169";
net AU_SDTO0 loc="p173";

# Ports of opb_videodec
#net IPort<0> loc="p188";
#net IPort<1> loc="p189";
#net IPort<2> loc="p191";
#net IPort<3> loc="p192";
```

```
#net IPort<4> loc="p193";
#net IPort<5> loc="p194";
#net IPort<6> loc="p198";
#net IPort<7> loc="p199";

#net HPort<0> loc="p174";
#net HPort<1> loc="p175";
#net HPort<2> loc="p176";
#net HPort<3> loc="p178";
#net HPort<4> loc="p179";
#net HPort<5> loc="p180";
#net HPort<6> loc="p181";
#net HPort<7> loc="p187";

#net IDQ loc="p205";
#net ICLK loc="p185";
#net IPGH loc="p200";
#net IPGV loc="p201";
#net ITRI loc="p204";
#net ITRDY loc="p206";


# Video decoder I2C Bus
#net VID_I2C_SCL loc="p6";
#net VID_I2C_SDA loc="p5";
```

2.1.3 Registers

Registers are organized into pages. 2 pages of ethernet registers:

Page 0 is the main page. The most critical registers to ethernet operation are the command register, interrupt status register, and the reset registers.

Page 1 contains registers which are used for setting up reception of multicast packets.

Page 0 register offsets

DATAPORT (0x10) - where we store in ethernet memory
NE_RESET (0x1f) - reset state
CMDR (0x00) - command register for read & write
PSTART (0x01) - page start register for write
PSTOP (0x02) - page stop register for write
BNRY (0x03) - boundary register for read and write
TPSR (0x04) - tx start page start register for write
TBCR0 (0x05) - tx byte count 0 register for write

TBCR1 (0x06) - tx byte count 1 register for write
ISR (0x07) - interrupt status register for read and write
RSAR0 (0x08) - low byte of remote start addr
RSAR1 (0x09) - hi byte of remote start addr
RBCR0 (0x0A) - remote byte count register 0 for write
RBCR1 (0x0B) - remote byte count register 1 for write
RCR (0x0C) - rx configuration register for write
TCR (0x0D) - tx configuration register for write
DCR (0x0E) - data configuration register for write
IMR (0x0F) - interrupt mask register for write

Page 1 register offsets

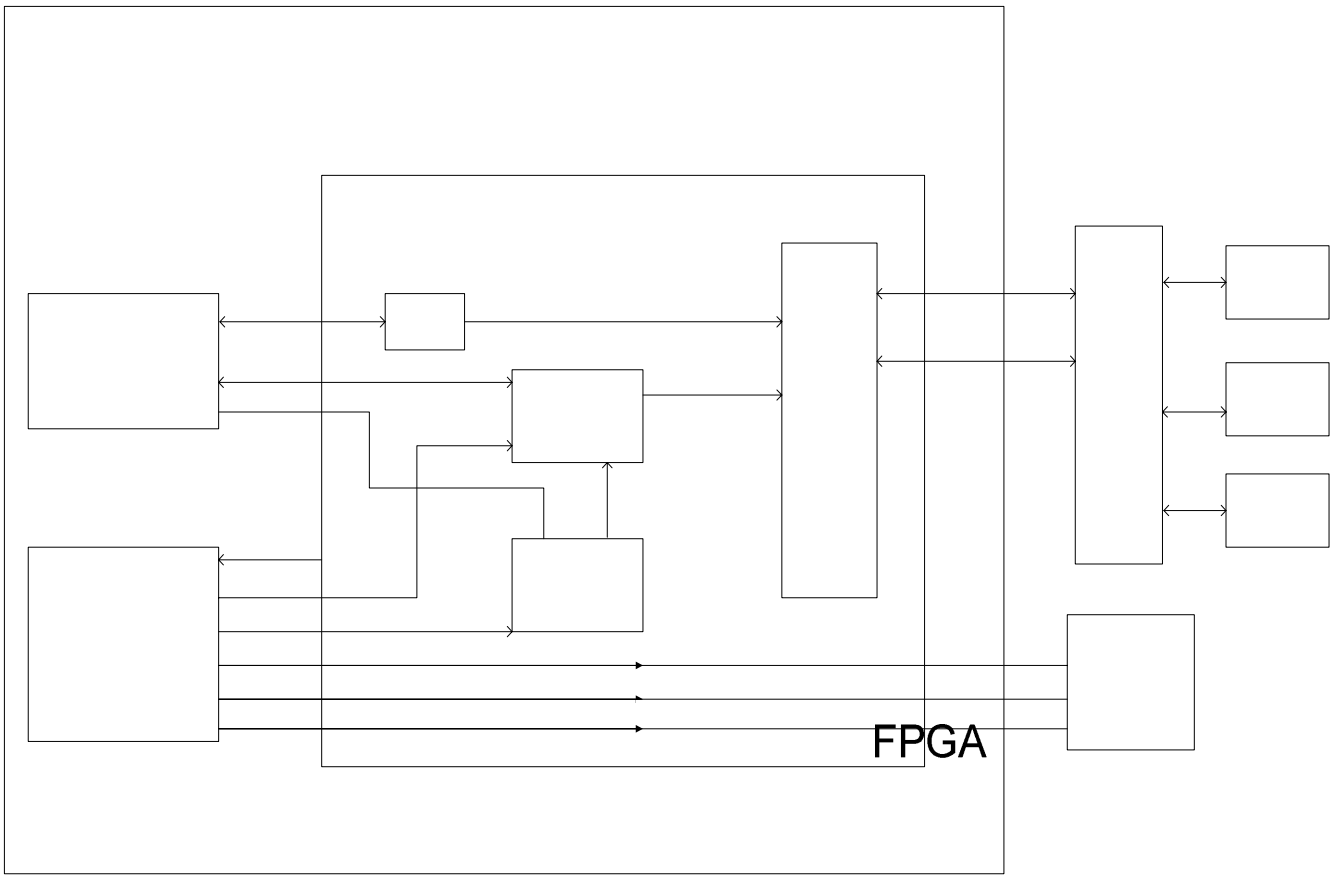
PAR0 (0x01) - physical addr register 0 for read and write
CURR (0x07) - current page register for read and write
MAR0 (0x08) - multicast addr register 0 for read and WRITE

Values for Buffer Length and Field Definition Info, used to divde the on-chip SRAM into sending and receiving halves.

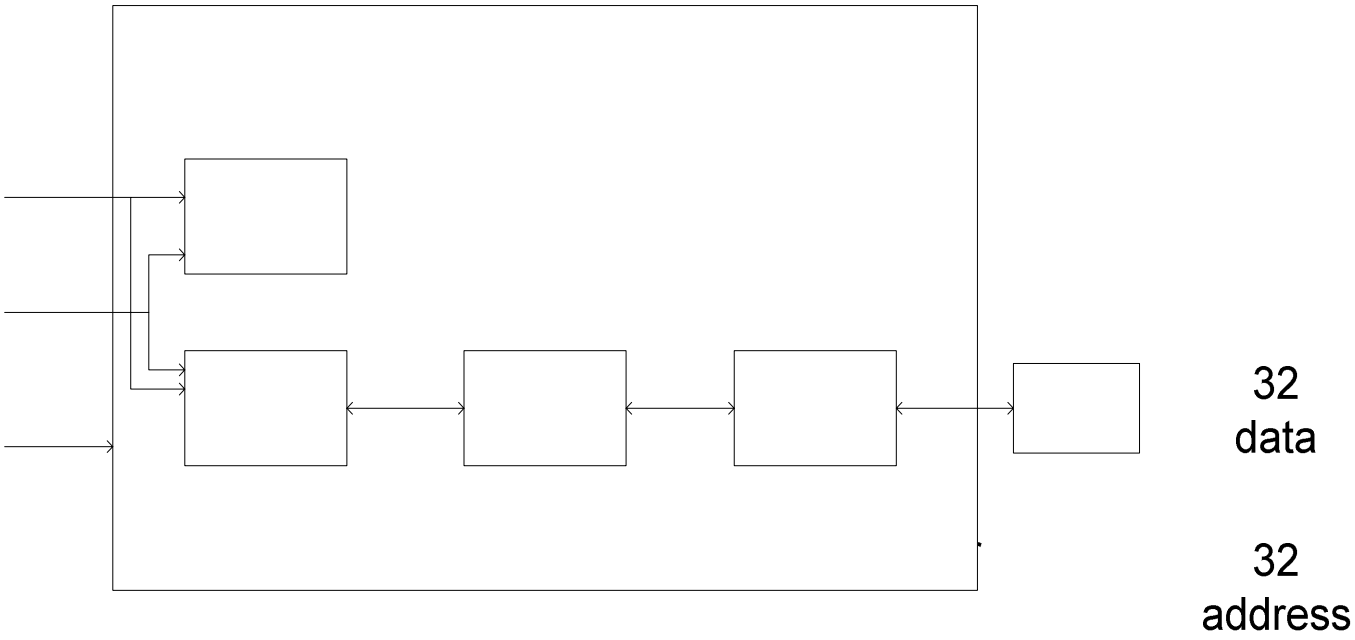
TXSTART 0x40 - Tx buffer start page
TXPAGES 6 - Pages for Tx buffer
RXSTART (TXSTART+TXPAGES) - Rx buffer start page
RXSTOP 0x7e - Rx buffer end page for word mode
DCRVAL 0x49 - DCR values for word mode

The buffers are organized in a circular list, a buffer Ring. The current page register points to the first buffer used to store a packet and is used to restore the DMA for writing status to the buffer ring or for restoring the DMA address in the event of a run packet, a CRC, or Frame alignment error. The boundary register points to the first packet int he Ring not yet read by the host. If local DMA address reaches the boundary, reception is aborted.

2.1.4 Block Diagram of FPGA



2.1.5 Ethernet Chip



2.2 Done in Software

In software we implemented the ethernet device driver. Talking to the registers of the ethernet chip and initiating DMA operations, we were able to write a library of initialization, packet send, and packet receive code. As noted above, register values were multiplied by 2 from their actual values.

We send raw ethernet packets. We multicast packets using the multicast address FFFFFF. The data is preceded by destination and source addresses, a length field, and is followed by a CRC check sum that is computed and appended by the ethernet chip automatically. See tables for packet format.

2.2.1 Received Packet

D15 D8	D7 D0
Next Packet Pointer	Receive Status
Receive Byte Count 1	Receive Byte Count 0
Destination Address 1	Destination Address 0
Destination Address 3	Destination Address 2
Destination Address 5	Destination Address 4
Source Address 1	Source Address 0
Source Address 3	Source Address 2
Source Address 5	Source Address 4
Type/Length 1	Type/Length 0
Data 1	Data 0
…	…

2.2.2 Sent Packet

D15 D8	D7 D0
Receive Status	Next Packet Pointer
Receive Byte Count 0	Receive Byte Count 1
Destination Address 0	Destination Address 1
Destination Address 2	Destination Address 3
Destination Address 4	Destination Address 5
Source Address 0	Source Address 1
Source Address 2	Source Address 3
Source Address 4	Source Address 5
Type/Length 0	Type/Length 1
Data 0	Data 1
…	…

2.2.3 Initialization

- reset the chip and divide the memory into 2 halves
- check proper initialization by reading ISR value
- abort DMA (if in progress)
- set WORD width
- setup page and boundary registers for reception
- setup accept-all multicast

2.2.4 Packet send

- setup remote DMA operation for transferring packet to the on-chip SRAM
- by specifying address and length of data. The corresponding registers
- are RSAR0/1 and RBCR0/1.
- initiate transmission of packet by specifying the
- number of bytes to be sent (registers TBCR0/1) and then setting TPSR to
- TXSTART, and write 0x24 into the command register.
- busy poll the ISR until completion
- after the remote DMA completes, we expect a 0x42 value in the ISR

2.2.5 Packet receive

- when the ISR has the first bit set, then this indicates that the chip
- has received a packet. Packet reception is basically a 2 step process
- first step: determine the length of the packet received
- second step: receive the packet
- packet receiption is done by a remote DMA read operation described below:
- setup remote DMA operation for transferring packet from the on-chip SRAM
- by specifying address and length of data. The corresponding registers
- are RSAR0/1 and RBCR0/1.
- read data WORDs from the DATAPORT register.
- after the remote DMA completes, we expect a 0x41 value in the ISR

We used this in our chat client, also written in software. We poll for received packets in a busy loop. The chat client used the UART interrupt handler to receive characters from the user into a queue buffer. We adapted the interrupt handler from the code.

The logic built around the send/receive ethernet functions allowed us to build packets from each line typed by the user, send and receive them over ethernet, and display them.

Collision / bad packet detection:
We do not care about retransmission due to packet collision.

Minicom interface - We modified the source code for the minicom serial

communication software in order to display the message lines in a chat-like interface.

2.3 Experimental setup

We used two stations an connected the boards via an ethernet hub. Because of this setup, we could use multicast to send packets from one board to the other. The attached PC's ran our modified minicom version to offer a chat user interface.

2.4 Testing/Development

Initially, we tried to test remote DMA operation, which did not occur according to the manual specifications. Quite a while later we realized that the manual was incomplete/faulty.

We tested sending packets less than 46 bytes in length. 46 is the minimum amount of data that can be sent in the ethernet. We must otherwise pad the data up to 46.

Once sending was working, we focused on receiving.

We used the Ethereal packet sniffer to monitor network traffic, and observe whethere packets are send correctly from the boards. We could also detect this way if a failed reception was due to faulty sending or receiving.

We have used a diagnostic routine to test proper ethernet chip initialization by reading values and comparing them against expected numbers from the manual.

In writing the ethernet driver, we referenced similar code written for PC's and various other hardware platforms for ne2000 compatible chips.

2.5 Unfinished Business

We were unable to finish the video output, and instead had to use a modified minicom for the user interface. The modified version of minicom displays when the ethernet card has been successfully reset and initialized (or the appropriate error message should a problem arise). The top of the window displays a log of the conversation, as well as the name of the person who sent each message. The bottom line displays the message currently being typed, which is moved to the log and sent when the enter key is pressed.

The reason that video could not be implemented as originally intended was because the display is delayed by four cycles so the video cannot access the bus often enough, which would cause glitches.

Implementing the pixel_clock for the video proved to be a daunting task. Even though we had the 50 MHz clock, dividing the frequency caused problems. The VHDL compiler did not know how to place the components, and we are not skilled enough to do manual placement.

3 Conclusions

3.1 Who Did What

- OPB – Salman, Bogdan, Sharon
- Ethernet – Bogdan, Salman, Sharon
- Display (not used) - Sharon
- Minicom - Bogdan
- Report – Bogdan, Salman, Sharon

3.2 Lessons Learned

Bus contention is much easier when only one peripheral is used. One of the hardest problems we faced was how to deal with the OPB, and having more peripherals that need to access it just makes things harder.

One of the most helpful thing in building the project was existing code. Both the code from the labs, as well as other people's, was critical in figuring out how to write our own code. Collaborating with another group, jay-cam, proved to be invaluable for getting the ethernet up and running. Sharing knowledge was extremely important.

The documentation provided was useful for something, but relying on the manuals often proved to be a bad idea. Most notably, the manuals said to use registers that had nothing to with what the manual claimed they did.

VHDL is really a design language, not a programming language. Having never used VHDL before, it was very easy to treat it like just another programming language for describing algorithms. However, it is very difficult to code effectively with that mindset. Getting used to the idea of designing rather than simply programming makes it much easier to write code, as well as increase the quality of the code.

As always, with any project, starting early is imperative. Our group got off to a rather late start, and after many problems with the OPB and ethernet controller, we only ended up with a working product the night before the presentation.

3.3 Things to Do Differently

- Due to lack of hardware, we did not test transmission early on.
- We tried to interact with the ethernet chip by doing a remote DMA read/write operation. This did not work initially due to timing issues, which we should have checked more thoroughly before testing.
- We did not check the actual transmission of packets on the wire since we did not have any support in either PC to check the packets. Putting an additional ethernet

card in one of the PC's would have allowed better testing.

- The timing is slightly different on each board, so if the timing is off any one part, the code will have different behavior on each board. If timing constraints are too close to the actual performance of the hardware, some boards work better than others.

3.4 Advice for Future Projects

- Try to use as few of the peripherals as possible. The more things added on, the harder it is to get any of them to work.
- Work with other groups using the same peripherals.

4 Source Code

4.1 Software Files

4.1.1 ether_reg.h

Defines the Ethernet Registers and Functions

```
#ifndef _ETHER_REG_H
#define _ETHER_REG_H

#include "icc.h"

// 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  0x40             // Tx buffer start page
#define TXPAGES  6                // Pages for Tx buffer
#define RXSTART  (TXSTART+TXPAGES) // Rx buffer start page
#define RXSTOP   0x7e             // Rx buffer end page for word mode
#define DCRVAL   0x49             // DCR values for word mode

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

```
//Structure Headers for Ethernet
#define MACLEN 6
typedef struct                  /* Net driver configuration data */
{
    WORD dtype;                     /* Driver type */
    BYTE myeth[MACLEN];             /* MAC (Ethernet) addr */
    WORD ebase;                     /* Card I/O base addr */
    WORD next_pkt;                  /* Next (current) Rx page */
} CONFIGNE;

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



// Ether Function Prototypes
void diag();
int init_etherne(CONFIGNE* cp);
void resetnic(CONFIGNE* cp);
void dmaRead(void *data, WORD addr, WORD length);
void dmaWrite(WORD *data, WORD addr, WORD length);
int send(WORD len, WORD *packet);
void getnic(WORD addr, BYTE data[], WORD len);
void receive(int* len, BYTE* buffer);
//void delay(int mult);



//Global Variables
#define WORDMODE 1

static int promisc=0;
// packet prototype


#endif
```

4.1.2 icc.h

Declares the BYTE and WORD variables. Serves as a general header

```
#ifndef _ICC_H
#define __ICC_H

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

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

void delay(int mult);

#endif
```

4.1.3 etherFunc.c

Contains functions for resetting, and diagnosis of ethernet

```
#include "ether_reg.h"

void delay(int mult){
  int i;
  int delay = 1000000*mult;
  for(i=0; i<delay; i++);
}


int init_etherne(CONFIGNE* cp)
{
  outnic(NE_RESET, innic(NE_RESET));  // Do reset
  delay(2);
  if ((innic(ISR) & 0x80) == 0)       // Report if failed
  {
    print("  Ethernet card failed to reset!\r\n");
    return 0;
  }
  else
  {
    print("Ethernet card reset successful...");
    resetnic(cp); // Reset Ethernet card,
    print("Ethernet card intialization complete!\r\n");
    //return 1;
  }
  return 1;
}

void resetnic(CONFIGNE* cp){
  int i;
  WORD curr;

  outnic(CMDR, 0x21);  // Abort and DMA and stop the NIC
  delay(2);

  outnic(DCR, DCRVAL);   // Set word-wide access

  outnic(RBCR0, 0x00); // Clear the count regs
  outnic(RBCR1, 0x00);

  outnic(IMR, 0x00);   // Mask completion irq
  outnic(ISR, 0xFF);   // clear interrupt status register

  outnic(RCR, 0x20);   // 0x20  Set to monitor mode
  outnic(RCR, 0x02);

  // Set Rx start, Rx stop, Boundary and TX start regs
  outnic(PSTART, RXSTART);
  outnic(PSTOP, RXSTOP);
  outnic(BNRY, RXSTOP-1);
```

```
  outnic(TPSR, TXSTART);

  outnic(CMDR, 0x61);                 /* Stop, DMA abort, page 1 */
  delay(2);
  //for (i=0; i<6; i++)                 /* Set Phys addr */
  //  outnic(PAR0+i, cp->myeth[i]);
  for (i=0; i<8; i++)                 /* Multicast accept-all */
    outnic(MAR0+i, 0xff);
  outnic(CURR, RXSTART+1);            /* Set current Rx page */
  cp->next_pkt = RXSTART + 1;
  outnic(CMDR, 0x20);                 /* DMA abort, page 0 */
  outnic(RCR, promisc ? 0x14 : 0x04); /* Allow broadcasts, maybe all
pkts */
  outnic(TCR, 0);                     /* Normal Tx operation */
  outnic(ISR, 0xff);                  /* Clear interrupt flags */
  outnic(CMDR, 0x22);                 /* Start NIC */

}

// diagnostic that tests nic functionality
void diag(){
  int i;

  print("\r\n*************** Beginning of Diagnostic
***************\r\n\r\n");

  print("         Command Register Page Switching Test\r\n");
  print("Switching to page 1\r\n");
  outnic(CMDR, 0x61);

  print("Writing to and reading from 0x0D (value should be 0x4e): ");
  outnic(0x0D*2,0x4d);
  putnum(innic(0x0D*2));
  print("\r\n");

  print("Switching to page 0\r\n");
  outnic(CMDR, 0x21);

  print("Reading from reg offset 0x0D: ");
  putnum(innic(0x0D*2));
  print("\r\n");

  print("Switching to page 1\r\n");
  outnic(CMDR, 0x61);
  print("Reading from reg offset 0x0D (should be 0x4e): ");
  putnum(innic(0x0D*2));
  print("\r\n\r\n");

  print("         Default Value Test\r\n");
  print("Switching to page 0\r\n");
  outnic(CMDR,21);
  print("Reading from 0x16 (value should be 0x15): ");
  putnum(innic(0x16*2));
  print("\r\n");
  print("Reading from 0x12 (value should be 0x0c): ");
  putnum(innic(0x12*2));
  print("\r\n");
```

```
  print("Reading from 0x13 (value should be 0x12): ");
  putnum(innic(0x13*2));
  print("\r\n\r\n*************** End of Diagnostic
***************\r\n\r\n");
}


void initpacket(Xuint8 *packet)
{

  packet[0]=0xFF;
  packet[1]=0xFF;
  packet[2]=0xFF;
  packet[3]=0xFF;
  packet[4]=0xFF;
  packet[5]=0xFF;
  packet[6]=0x00;
  packet[6]=0x0D;
  packet[8]=0x60;
  packet[9]=0x7F;
  packet[10]=0xF9;
  packet[11]=0xAF;
  packet[12]=0x08;
  packet[13]=0x00;
  packet[14]=0x45;
  packet[15]=0x00;
  packet[16]=0x00;
  packet[17]=0xBE;
  packet[18]=0x00;
  packet[19]=0x00;
  packet[20]=0x00;
  packet[21]=0x00;
  packet[22]=0x01;
  packet[23]=0x11;
  packet[24]=0xB8;
  packet[25]=0xEE;
  packet[26]=0x80;
  packet[27]=0x3B;
  packet[28]=0x95;
  packet[29]=0xF9;
  packet[30]=0xE4;
  packet[31]=0x05;
  packet[32]=0x06;
  packet[33]=0x07;
  packet[34]=0x0C;
  packet[35]=0x2C;
  packet[36]=0x1A;
  packet[37]=0x85;
  packet[38]=0x00;
  packet[39]=0xAA;
  packet[40]=0x00;
  packet[41]=0x00;
  //extra byte
  packet[42]=0x00;


}
```

4.1.4 etherSend.c

Function for Sending Data over the Ethernet

```
#include "ether_reg.h"

static int incr = 0;

int send(WORD len, WORD *data){

  int i, j, h;
  WORD counter;
  WORD word;
  WORD addr = (TXSTART << 8);

  counter = len>>1;

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

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

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

  outnic(TBCR0, (len&0xff)); // set Transmit Byte Count
  outnic(TBCR1, (len>>8));

  outnic(CMDR, 0x12); // start the DMA write

  // 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);
  }
  incr++;

  if(!(innic(ISR)&0x40)){
    print("Data - DMA did not finish\r\n");
    return 1;
  }

  outnic(TPSR, TXSTART); // set Transmit Page Start Register
  outnic(CMDR, 0x24); // start transmission

  j=1000;
  while(j-->0 && !(innic(ISR)&0x02));
  if(!j){
    return 1;
  }

  outnic(ISR,0xFF);

  return 0;
}
```

4.1.5 etherReceive.c

Function for Receiving Data over the Ethernet

```
#include "ether_reg.h"

void receive(int* len, BYTE* buffer) {
  WORD packetaddr;
  NICHDR nichdr;
  BYTE temp;
  int i;

  //get size of header
  getnic((WORD)(innic(TBCR1)-1)<<8, (BYTE*)&nichdr, sizeof(NICHDR));

  *len = nichdr.len;

  packetaddr = ((innic(TBCR1)-1)<<8)+sizeof(NICHDR);
  getnic((WORD)packetaddr, (BYTE*)buffer, *len);

  outnic(ISR,0xff);

  for(i=14;i < *len;i=i+2){
    temp=buffer[i];
    buffer[i]=buffer[i+1];
    buffer[i+1]=temp;
  }
}

// Get a packet from a given address in the NIC's RAM
void getnic(WORD addr, BYTE data[], WORD len)
{
    register int count;
    register WORD *dataw;

    count = WORDMODE ? len>>1 : len;    /* Halve byte count if word I/P
*/
    outnic(ISR, 0x40);                  /* Clear remote DMA interrupt
flag */
    outnic(RBCR0, len&0xff);            /* Byte count */
    outnic(RBCR1, len>>8);
    outnic(RSAR0, addr&0xff);           /* Data addr */
    outnic(RSAR1, addr>>8);
    outnic(CMDR, 0x0a);                 /* Start, DMA remote read */
#if WORDMODE
    dataw = (WORD *)data;               /* Use pointer for speed */
    while(count--)                      /* Get words */
        *dataw++ = innic(DATAPORT);
    if (len & 1)                        /* If odd length, do last byte
*/
        *(BYTE *)dataw = innic(DATAPORT);
#else
    while(count--)                      /* Get bytes */
        *data++ = innic(DATAPORT);
```

```
#endif

    /*    print("ISR After Reception: ");
    putnum(innic(ISR));
    print("\r\nCount: ");
    putnum(count);
    print("\r\n");
    */

}
```

4.1.6 queue.h

Declares functions/structures for the circular buffer

```
#ifndef __SALMANS_QUEUE
#define __SALMANS_QUEUE

#define LEN 2

struct queue {
  int iHead;
  int iTail;
  char pchBuffer[LEN];
};

void init_queue(struct queue* pQueue);

int enqueue(struct queue* pQueue, char ch);

int dequeue(struct queue* pQueue, char* pch);

#endif
```

4.1.7 queue.c

Contains circular buffer functions

```
/* SALMANS_QUEUE (circular, too)*/

#include "queue.h"

void init_queue(struct queue* pQueue) {
  pQueue->iHead = 0;
  pQueue->iTail = 0;
}

int enqueue(struct queue* pQueue, char ch) {
  //Overflow condition

  if( (pQueue->iHead == pQueue->iTail+1) ||
      (pQueue->iHead == 0 && pQueue->iTail == LEN-1)) {
    return 0;
  }

  //Insert Character at Tail
  pQueue->pchBuffer[pQueue->iTail] = ch;

  //Adjust the Tail pointer
  if(pQueue->iTail == LEN-1) {
    pQueue->iTail = 0;
  } else {
    pQueue->iTail = pQueue->iTail+1;
  }

  return 1;

}//enqueue


int dequeue(struct queue* pQueue, char* pch) {
  //Underflow
  if(pQueue->iTail == pQueue->iHead) {
    return 0;
  }

  //Get Character
  *pch = pQueue->pchBuffer[pQueue->iHead];

  if(pQueue->iHead == LEN-1) {
    pQueue->iHead = 0;
  } else {
    pQueue->iHead += 1;
  }

  return 1;
}//dequeue
```

```
void printQueue(struct queue* pQueue) {
  int i,j;

}
```

4.1.8 isr.c

Contains the UART interrupt handler

```
#include "xbasic_types.h"
#include "xio.h"
#include "xparameters.h"
#include "xuartlite_l.h"
#include "queue.h"

int uart_interrupt_count = 0;
char uart_character;
extern struct queue q;

/*
 * Interrupt service routine for the UART
*/
void uart_handler(void *callback)
{
  Xuint32 IsrStatus;

  Xuint8 incoming_character;

  /* Check the ISR status register so we can identify the interrupt
source */
  IsrStatus = XIo_In32(XPAR_MYUART_BASEADDR + XUL_STATUS_REG_OFFSET);

  if ((IsrStatus & (XUL_SR_RX_FIFO_FULL | XUL_SR_RX_FIFO_VALID_DATA))
!= 0) {
    /* The input FIFO contains data: read it */
    incoming_character =
      (Xuint8) XIo_In32( XPAR_MYUART_BASEADDR + XUL_RX_FIFO_OFFSET );

    //Enqueue Characters into Circular Buffer
    if (enqueue(&q, incoming_character))
      ++uart_interrupt_count;

  }

  if ((IsrStatus & XUL_SR_TX_FIFO_EMPTY) != 0) {
    /* The output FIFO is empty: we can send another character */
  }

}//uart_handler
```

4.1.9 main.c

Main c file.

```
/* Project
   Sharon Price
   Salman Baset
   Bogdan Caprita
*/

#include "xbasic_types.h"
#include "xio.h"
#include "xintc_l.h"
#include "xuartlite_l.h"
#include "queue.h"


#include "ether_reg.h"

//WORD packet[100];

Xuint16 pack_buf[256]; // 512 bytes
BYTE *packet;

CONFIGNE configs[2];


#define W 640
#define H 480

#define C_WIDTH 8
#define C_HEIGHT 8

#define WIDTH  (W/C_WIDTH)
#define HEIGHT (H/C_HEIGHT)


#define VGA_START 0x00800000


#define RED 0xE0
#define GREEN 0x1C
#define BLUE 0x03
#define BLACK 0
#define FONT_COLOR RED | GREEN


// defined in isr.c
extern void uart_handler(void *callback);
extern int uart_interrupt_count;
extern char uart_character;

// flag to check if it's a printable char
int isPrint;

// the circular buffer
```

```
struct queue q;

int x, y;

void printChar(char c, int x, int y)
{
  //XIo_Out8(VGA_START + (y-1) * WIDTH + x, c);
  char ch[2]={'\0'};
  ch[1]=c;
  print(c);
}


/* scrolls screen up by amount lines */
int scrolling(int amount)
{
  //memmove(VGA_START, VGA_START + amount*WIDTH, (HEIGHT - amount) *
WIDTH);
  //memset(VGA_START + (HEIGHT - amount) * WIDTH, 0, amount * WIDTH);
}

/*
 * setup_interrupts: Initialize the interrupt sources and handlers
 *
 * Should be called once when the system starts
 *
 * The main _interrupt_handler() function from Xilinx
 *
 * Saves and restores CPU context, etc.
 *
 * Sees which interrupts are pending, and for each it
 *     acknowledges the interrupt and
 *     calls a user-defined interrupt handler in
Xintc_InterruptVectorTable
 *
 * Place interrupt service routines in isr.c to ensure they are placed
in
 * the proper memory segment.
 */
void setup_interrupts()
{
  /*
   * Reset the interrupt controller peripheral
   */

  /* Disable the interrupt signal */
  XIntc_mMasterDisable(XPAR_INTC_SINGLE_BASEADDR);

  /* Disable all interrupt sources */
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,0);

  /* Acknowledge all possible interrupt sources
     to make sure none are pending */
  XIntc_mAckIntr(XPAR_INTC_SINGLE_BASEADDR, 0xffffffff);

  /*
   * Install the UART interrupt handler
```

```
   */

  XIntc_InterruptVectorTable[XPAR_INTC_MYUART_INTERRUPT_INTR].Handler =
    uart_handler;

  /*
   * Enable interrupt sources
   */

  /* Enable CPU interrupts */
  microblaze_enable_interrupts();

  /* Enable interrupts from the interrupt controller */
  XIntc_mMasterEnable(XPAR_INTC_SINGLE_BASEADDR);

  /* Tell the interrupt controller to accept interrupts from the UART
*/
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,
XPAR_MYUART_INTERRUPT_MASK);

  /* Enable UART interrupt generation */
  XUartLite_mEnableIntr(XPAR_MYUART_BASEADDR);
}

void send_data(int len, BYTE* data) {

  int i=0;
  //print("In Send Data Function \r\n");

  if(len < 46) {
    for(i=14+len;i<14+46;i++)
      data[i]=0;
    len=46;
  }

  if(len%2!=0) {
    len++;
    data[13+len]='X';
  }


  /*for(i=0;i<14+len;i++){
    putnum(data[i]);
    print(" ");
    if(i%10 == 0 && i!= 0) {
      print("\r\n");
    }
  }
  */

  //  print("\r\nSending\r\n");

  send(len,(WORD*)data);

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

}
```

```
int main()
{
  char c;
  int i, tmp_1, len;
  char buf[2], buf2[2];
  char buffer[10];
  int line_count=0;
  BYTE line[100];
  WORD *pkt;
  WORD word;
  WORD curr;
  CONFIGNE* cp;

  cp = &configs[0];
  packet = (BYTE*)pack_buf;


  // Enable the instruction cache: makes the code run 6 times faster
  //microblaze_enable_icache();

  microblaze_enable_icache();

  print("\r\n\r\n---------------Starting-------------\r\n\r\n");
  //diag();

  // reset the NIC card
  if(!init_etherne(cp)){
    print("Ethernet NIC not detected");
    return;
  }


  //Initialize the Circular Buffer
  buf[1] = '\0';
  init_queue(&q);

  //Setup Interrupts
  setup_interrupts();

  // erasing screen
  x = -1; y = 1;
  //isPrint=1;

  //Initialize the Packet
  initpacket(line); // put some header


  //print("\r\nSending:\r\n");

  while (1) {

    // Disable interrupts
    microblaze_disable_interrupts();

    //Get Character from Buffer Queue
    if (dequeue(&q, buf)){
```

```
    //print(buf);
    //print("Dequeue\r\n");
    microblaze_enable_interrupts();

    switch (buf[0]){
    case '\r':
    /*print(" ");
    for(i=0;i<line_count;i++){
      buf[0]=line[i];
      print(buf);
    }
    */
    send_data(line_count, (BYTE*)line);
    line_count=0;
    x = -1; break;
    case '\n':
    line_count=0;
    y++; break;
    case 8 :
    line[14+ line_count ] =  0;
    default:
    line[14+line_count]=buf[0];
    line_count++;

    isPrint = 1;
    x ++;
    if (x > WIDTH-1 ){
      x = 0;
      y ++;
    }
    }

} else {
 //Enable interrupts
    microblaze_enable_interrupts();
}


 curr=0;
 buf2[1]='\0';
 if( (curr=innic(ISR)&0x01)!=0){
    receive(&len, packet);

    //      print("\r\nReceived: ");
    //putnum(len);
    /*for(i=14;i<len-4;i++){

    buf2[0] = (char)packet[i];

    if(packet[i]==0x00)
      break;

    print(buf2);
    }*/
    packet[len-5] = 0;
    print(packet+14);
    print("\n");
```

```
      //print("\r\n\r\nSending: \r\n");
      //print("\r\nSending: \r\n");
    }

    // see if need to scroll
    //    microblaze_disable_interrupts();
    if (y > HEIGHT){
      y -= 1;
      //scrolling(1);
    }

  }

  return 0;

}//main
```

4.2 Hardware Files

4.2.1 memoryctrl.vhd

Contains the statemachine for interaction with Ethernet and video. Also contains the code for 'long' ethernet read and writes

```
#include "xbasic_types.h"
#include "xio.h"
#include "xparameters.h"
#include "xuartlite_l.h"
#include "queue.h"

int uart_interrupt_count = 0;
char uart_character;
extern struct queue q;

/*
 * Interrupt service routine for the UART
*/
void uart_handler(void *callback)
{
  Xuint32 IsrStatus;

  Xuint8 incoming_character;

  /* Check the ISR status register so we can identify the interrupt
source */
  IsrStatus = XIo_In32(XPAR_MYUART_BASEADDR + XUL_STATUS_REG_OFFSET);

  if ((IsrStatus & (XUL_SR_RX_FIFO_FULL | XUL_SR_RX_FIFO_VALID_DATA))
!= 0) {
    /* The input FIFO contains data: read it */
    incoming_character =
      (Xuint8) XIo_In32( XPAR_MYUART_BASEADDR + XUL_RX_FIFO_OFFSET );

    //Enqueue Characters into Circular Buffer
    if (enqueue(&q, incoming_character))
      ++uart_interrupt_count;

  }

  if ((IsrStatus & XUL_SR_TX_FIFO_EMPTY) != 0) {
    /* The output FIFO is empty: we can send another character */
  }

}//uart_handler
```

4.2.2 opb_xs300.vhd

```
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_xsb300 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"
    );

    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;
                      pixel_clock : in std_logic;
            io_clock: in std_logic;
                      UIO_DBus : out std_logic_vector (31 downto 0);
                      UIO_errAck : out std_logic;
                      UIO_retry : out std_logic;
                      UIO_toutSup : out std_logic;
                      UIO_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;

                      VIDOUT_CLK : out std_logic;
                      VIDOUT_RCR : out std_logic_vector (9 downto 0);
                      VIDOUT_GY : out std_logic_vector (9 downto 0);
                      VIDOUT_BCB : out std_logic_vector (9 downto 0);
                      VIDOUT_BLANK_N : out std_logic;
                      VIDOUT_HSYNC_N : out std_logic;
                      VIDOUT_VSYNC_N : out std_logic;
                      PB_D : inout std_logic_vector (15 downto 0)
                    );
```

```
end opb_xsb300;

architecture Behavioral of opb_xsb300 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 svga
    Port ( clk : in std_logic;
           pix_clk : in std_logic;
           rst : in std_logic;
           video_data : in std_logic_vector(15 downto 0);
           video_addr : out std_logic_vector(19 downto 0);
           video_req : out std_logic;
           vidout_clk : out std_logic;
           vidout_RCR : out std_logic_vector(9 downto 0);
           vidout_GY : out std_logic_vector(9 downto 0);
           vidout_BCB : out std_logic_vector(9 downto 0);
           vidout_BLANK_N : out std_logic;
           vidout_HSYNC_N : out std_logic;
           vidout_VSYNC_N : out std_logic);
end component;

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

svga1 : svga
port map (  clk => OPB_Clk,
                        pix_clk => pixel_clock,
                        rst => OPB_Rst,
                        video_addr => video_addr,
                        video_data => video_data,
                        video_req => video_req,
```

```
                        VIDOUT_CLK => VIDOUT_CLK,
                        VIDOUT_RCR => VIDOUT_RCR,
                        VIDOUT_GY => VIDOUT_GY,
                        VIDOUT_BCB => VIDOUT_BCB,
                        VIDOUT_BLANK_N => VIDOUT_BLANK_N,
                        VIDOUT_HSYNC_N => VIDOUT_HSYNC_N,
                        VIDOUT_VSYNC_N => VIDOUT_VSYNC_N);


-- 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 hihalf));

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

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


-- 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;

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
                        UIO_DBus(i) <= '0';

                  elsif OPB_Clk'event and OPB_Clk ='1' then
                        if rreset = '1' then
                              UIO_DBus(i) <= '0';
                        elsif (rce1 or rce0) = '1' then
                              UIO_DBus(i) <= rdata(i);
                        end if;
                  end if;
            end loop;
```

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

                        if rreset = '1' then
                              UIO_DBus(i) <= '0';
                        elsif rce0 = '1' then
                              UIO_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
UIO_errAck <= '0';
UIO_retry <= '0';
UIO_toutSup <= '0';

UIO_xferAck <= xfer;


end Behavioral;
```

4.2.3 pad_io.vhd

```
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;
```

4.2.4 svga.vhd

```
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;
```

4.2.5 svga_timing.vhd

```
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 svga_timing is
    Port ( pixel_clock : in std_logic;
           reset : in std_logic;
           h_synch_delay : out std_logic;             -- h_synch
delayed 2 clocks to line up with DAC pipeline
           v_synch_delay : out std_logic;             -- v_synch
delayed 2 clocks to line up with DAC pipeline
           comp_synch : out std_logic;                      --
composite synch for DAC
           blank : out std_logic;                           --
composite blanking
           vga_ram_read_address : out std_logic_vector(19 downto 0));
end svga_timing;

architecture Behavioral of svga_timing is

constant SRAM_DELAY : integer := 3;

-- 640 X 480 @ 60Hz with a 25.175 MHz pixel clock
constant H_ACTIVE : integer := 640;
constant H_FRONT_PORCH : integer := 16;
--constant H_SYNCH : integer := 96;
constant H_BACK_PORCH : integer := 48;
constant H_TOTAL : integer := 800;

--constant H_ACTIVE : integer := 10;
--constant H_FRONT_PORCH : integer := 3;
--constant H_BACK_PORCH : integer := 3;
--constant H_TOTAL : integer := 20;




constant V_ACTIVE : integer := 480;
constant V_FRONT_PORCH : integer := 11;
--constant V_SYNCH : integer := 2;
constant V_BACK_PORCH : integer := 31;
constant V_TOTAL : integer := 524;

signal line_count: std_logic_vector (9 downto 0);           -- counts
the display lines
signal pixel_count : std_logic_vector (10 downto 0);  -- counts the
pixels in a line
```

```
signal h_synch : std_logic;
      -- horizontal synch
signal v_synch : std_logic;
      -- vertical synch
signal h_synch_delay0 : std_logic;
      -- h_synch delayed 1 clock
signal v_synch_delay0 : std_logic;
      -- v_synch delayed 1 clock

signal h_c_synch : std_logic;             -- horizontal component of
comp synch
signal v_c_synch : std_logic;             -- vertical component of comp
synch
signal h_blank : std_logic;               -- horizontal blanking
signal v_blank : std_logic;               -- vertical blanking

signal reset_vga_ram_read_address : std_logic;  -- flag to reset the
ram address during VBI
signal hold_vga_ram_read_address : std_logic;   -- flag to hold the
address during HBI

signal temp: STD_LOGIC_VECTOR (19 downto 0);

begin


      process (pixel_clock, reset)



      begin
-- CREATE THE HORIZONTAL LINE PIXEL COUNTER
            if reset = '1' then
                                    -- on reset set pixel counter to 0
                        pixel_count <= "00000000000";

      elsif pixel_clock'event and pixel_clock='1' then
      -- positive clk edge

            if pixel_count = (H_TOTAL - 1) then
            -- last pixel in the line
                        pixel_count <= "00000000000";
            -- reset pixel counter
            else
                        pixel_count <= pixel_count +1;
            end if;

      end if;

-- CREATE THE HORIZONTAL SYNCH PULSE
      if reset = '1' then
                              -- on reset set h_sync to 0
                h_synch <= '0';
                              -- remove h_synch

      elsif pixel_clock'event and pixel_clock='1' then
      -- positive clk edge
```

```
            if pixel_count = (H_ACTIVE + H_FRONT_PORCH -1) then   --
start of h_synch
                  h_synch <= '1';
            elsif pixel_count = (H_TOTAL - H_BACK_PORCH -1) then  --
end of h_synch
                  h_synch <= '0';
            end if;
      end if;


-- CREATE THE VERTICAL FRAME LINE COUNTER
      if reset = '1' then
                              -- on reset set line counter to 0
                  line_count <= "0000000000";

      elsif pixel_clock'event and pixel_clock='1' then
      -- positive clk edge

            if ((line_count = V_TOTAL - 1) and (pixel_count = H_TOTAL -
1)) then                              -- last pixel in the line
                        line_count <= "0000000000";
                                          -- last pixel in last line of
frame
            elsif pixel_count = (H_TOTAL - 1) then
                              line_count <= line_count + 1;
                  -- increment line counter
            end if;

      end if;

-- CREATE THE VERTICAL SYNCH PULSE
      if reset = '1' then
                  v_synch <= '0';
                              -- remove v_synch

      elsif pixel_clock'event and pixel_clock='1' then
      -- positive clk edge

            if line_count = (V_ACTIVE + V_FRONT_PORCH -1) and
pixel_count = (H_TOTAL - 1) then
                  v_synch <= '1';
                              -- start of v_synch
            elsif line_count = (V_TOTAL - V_BACK_PORCH - 1) and
pixel_count = (H_TOTAL - 1) then
                  v_synch <= '0';
                              -- end of v_synch
            end if;

      end if;

-- ADD TWO PIPELINE DELAYS TO THE SYNCHs COMPENSATE FOR THE DAC
PIPELINE DELAY
      if reset = '1' then
                  h_synch_delay0 <= '0';
                  v_synch_delay0 <= '0';
                  h_synch_delay  <= '0';
```

```
                  v_synch_delay  <= '0';

      elsif pixel_clock'event and pixel_clock='1' then            --
positive clk edge
                  h_synch_delay0 <= h_synch;
                  v_synch_delay0 <= v_synch;
                  h_synch_delay  <= h_synch_delay0;
                  v_synch_delay  <= v_synch_delay0;

      end if;

-- CREATE THE HORIZONTAL BLANKING SIGNAL
-- the "-2" is used instead of "-1" because of the extra register delay
-- for the composite blanking signal
      if reset = '1' then
                        -- on reset
                  h_blank <= '0';
                        -- remove the h_blank

      elsif pixel_clock'event and pixel_clock='1' then            --
positive clk edge

            if (pixel_count = (H_ACTIVE -2)) then
      -- start of HBI
                        h_blank <= '1';
            elsif (pixel_count = (H_TOTAL -2)) then
      -- end of HBI
                        h_blank <= '0';
            end if;

      end if;

-- CREATE THE VERTICAL BLANKING SIGNAL
-- the "-2" is used instead of "-1"  in the horizontal factor because
of the extra
-- register delay for the composite blanking signal
      if reset = '1' then
                        -- on reset
                  v_blank <= '0';
                        -- remove v_blank

      elsif pixel_clock'event and pixel_clock='1' then            --
positive clk edge

            if line_count = (V_ACTIVE - 1) and pixel_count = (H_TOTAL -
2) then
                  v_blank <= '1';
                        -- start of VBI
            elsif line_count = (V_TOTAL - 1) and pixel_count = (H_TOTAL
- 2) then
                  v_blank <= '0';
                        -- end of VBI
            end if;

      end if;
```

```
-- CREATE THE COMPOSITE BANKING SIGNAL
      if reset = '1' then
                        -- on reset
                  blank <= '0';
                              -- remove blank

      elsif pixel_clock'event and pixel_clock='1' then            --
positive clk edge
            if (h_blank or v_blank) = '1' then
                  -- blank during HBI or VBI
                        blank <= '1';
            else
                  blank <= '0';
                              -- active video do not blank
            end if;

      end if;

-- CREATE THE HORIZONTAL COMPONENT OF COMP SYNCH
-- the "-2" is used instead of "-1" because of the extra register delay
-- for the composite synch
      if reset = '1' then
                        -- on reset
                  h_c_synch <= '0';
                        -- remove h_c_synch

      elsif pixel_clock'event and pixel_clock='1' then            --
positive clk edge

            if pixel_count = (H_ACTIVE + H_FRONT_PORCH -2) then
                  h_c_synch <= '1';
                        -- start of h_c_synch
            elsif (pixel_count = (H_TOTAL - H_BACK_PORCH -2)) then
                  h_c_synch <= '0';
                        -- end of h_c_synch
            end if;

      end if;

-- CREATE THE VERTICAL COMPONENT OF COMP SYNCH
      if reset = '1' then
                        -- on reset
                  v_c_synch <= '0';
                        -- remove v_c_synch

      elsif pixel_clock'event and pixel_clock='1' then            --
positive clk edge
            if line_count = (V_ACTIVE + V_FRONT_PORCH - 1) and
pixel_count = (H_TOTAL - 2) then
                  v_c_synch <= '1';
                        -- start of v_c_synch
            elsif line_count = (V_TOTAL - V_BACK_PORCH - 1) and
pixel_count = (H_TOTAL - 2) then
                  v_c_synch <= '0';
                        -- end of v_c_synch
            end if;
```

```
      end if;

-- CREATE THE COMPOSITE SYNCH SIGNAL
      if reset = '1' then
                        -- on reset
                  comp_synch <= '0';
                        -- remove comp_synch
      elsif pixel_clock'event and pixel_clock='1' then
            comp_synch <= v_c_synch xor h_c_synch;
      end if;

-- CREATE THE RAM ADDRESS COUNTER
      if reset = '1' then
                        -- on reset
                  temp <= "00000000000000000000";

      elsif pixel_clock'event and pixel_clock='1' then            --
positive clk edge

            if reset_vga_ram_read_address = '1' then
                  temp <= "00000000000000000000";
            elsif hold_vga_ram_read_address = '0' then
                  temp <= temp + 1;
            end if;
      end if;
      vga_ram_read_address <= temp;

-- CREATE THE CONTROL SIGNALS FOR THE VGA RAM ADDRESS COUNTER
      if reset = '1' then
                        -- on reset
                  reset_vga_ram_read_address <= '0';

      elsif pixel_clock'event and pixel_clock='1' then            --
positive clk edge
            if line_count = V_ACTIVE -1 and pixel_count =((H_TOTAL -1)-
SRAM_DELAY) then
                  reset_vga_ram_read_address <= '1';
      -- reset the address counter at the end of active video
            elsif line_count = V_TOTAL -1 and pixel_count =((H_TOTAL -
1) - SRAM_DELAY) then
                  reset_vga_ram_read_address <= '0';
      -- re-enable the address counter at the start of active video
            end if;

      end if;

      if reset = '1' then
                        -- on reset
                  hold_vga_ram_read_address <= '0';

      elsif pixel_clock'event and pixel_clock='1' then            --
positive clk edge
            if pixel_count = ((H_ACTIVE-1) - SRAM_DELAY) then     --
hold the address counter at the end of active video
                  hold_vga_ram_read_address <= '1';
            elsif pixel_count = ((H_TOTAL -1) - SRAM_DELAY) then
```

```
                hold_vga_ram_read_address <= '0';
        -- re-enable the address counter at the start of active video
              end if;

        end if;

        end process;

end Behavioral;
```

4.2.6 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, pxclk_i, pxclk;
wire locked;


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

// synopsys translate_off
// synopsys translate_on
CLKDLL vdll(.CLKIN(clk_ibuf), .CLKFB(sys_clk),
       .CLK0(clk1x_i),
       .CLK2X(clk2x_i),
       .RST(1'b0), .LOCKED(locked)
);

CLKDLL pixvdll (.CLKIN(sys_clk), .CLKFB(pxclk),
        .CLK0(pxclk_i),
        .CLKDV(pixel_clock),
        .RST(1'b0),
        .LOCKED()
);

BUFG bg3 (.I(pxclk_i), .O(pxclk));


assign fpga_reset = ~locked;


endmodule
```

4.2.7 memoryctrl.vhd

Memory controller we wrote to integrate video buffer with our chat client

```
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_reth1, r_reth2, r_reth3 : 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';
            r_reth1 <= '0'; r_reth2 <= '0'; r_reth3 <= '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 and not eth_io);
        r_rb  <= opb_select and (r_ra);
        r_rc <= opb_select and (r_rb and fullword);

          r_reth1 <= opb_select and (r_common and rnw and eth_io);
          r_reth2 <= opb_select and r_reth1;
          r_reth3 <= opb_select and r_reth2;


        r_xfer <= opb_select and (
                    (r_common and not rnw and not fullword and not eth_io)
                or (r_w32)
                    or (r_weth2)
                    or (r_rb and not fullword)
                or (r_rc)
                    or (r_reth3)
          );

        end if;

    end process;


    read_early <= r_reth2;
    write_early <= not ((r_common and not rnw and eth_io) or (r_weth1) or
    r_weth2);
    hihalf <= r_w32 or (r_ra and fullword);

    wr_req_i <= (r_common and not rnw and not eth_io) or (r_w32) or
    (r_weth1) or (r_weth2) or (r_weth3);
    rd_req_i <= (r_common and rnw) or (r_ra and fullword) or r_reth1 or
    r_reth2;
    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_reth3;
    ce1 <= r_rb or r_rc or r_reth3;
```

```
videocycle_i <= vreq and not ( rd_req_i or wr_req_i or r_common );
videocycle <= videocycle_i;

process (clk, rst)
begin
  if rst = '0' then
    r_v0 <= '0';
    r_v1 <= '0';
  elsif clk'event and clk = '1' then
    r_v0 <= videocycle_i;
    r_v1 <= r_v0;
  end if;
end process;

video_ce <= r_v1;

end Behavioral;
```