JAYcam

Real Time Video System

Final Report

Yaniv Schiller – yhs2001@columbia.edu
AvrumTilman – amt77@columbia.edu
Josh Weinberg – jmw211@columbia.edu

Table of Contents

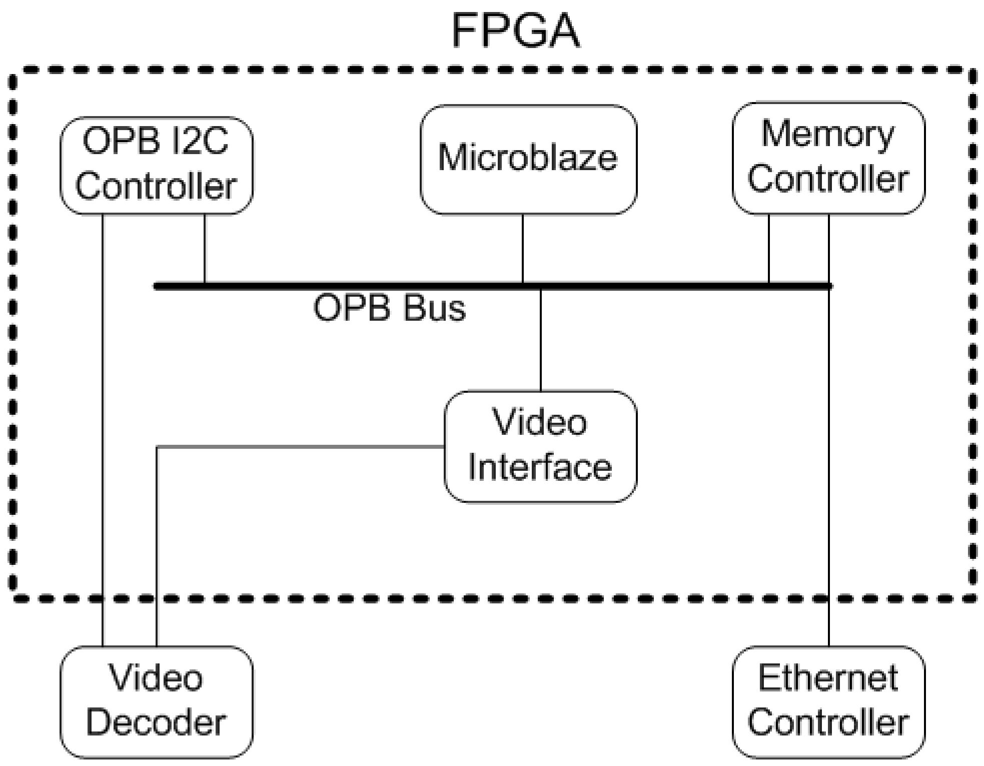
1. Introduction

1.1. Abstract

In broad terms, we will be using the FPGA and its peripherals to construct a simple web cam that will take in analog video, convert the video into a digital bit stream, and then broadcast the data on the Internet. The flow of data from the input camera to the Internet/LAN connected client computer involves several essential stages outlined below.

- Analog video input using the on-board Phillips SAA7114H chip
  The SAA7114H is a super fast chip whose primary purpose is "to capture and scale video images to be provided as a digital video stream through the image port of a VGA controller, for display via VGA's frame buffer, or for capture to system memory" (Phillips Manual page 4) .We will use this chip to capture an analog signal, convert it to a digital signal, and then use the on-chip scaler to scale down the frame size.
- Video compression/decimation
  If the newly created digital signal is still to big for the FPGA to handle, we will slow down the frame rate and further decrease the frame size. If Ethernet bandwidth becomes a problem, we may implement a simple compression protocol.
- UDP packet creation
  Once we have our ideal digital video signal, we will pack the video bytes along with other information (i.e. video frame number and other video details) into the data portion of UDP packets. This process will be implemented in software. The choice of UDP over raw Ethernet packets was made for a couple reasons. Firstly, there is a lot of source code floating around for UDP/IP stacks and drivers that are compatible with the on-board Ethernet controller. And secondly, the use of a transport layer protocol (such as UDP) will make client-side programming much simpler (i.e. it is very easy to open a UDP socket and receive/send data using Java).
- Internet broadcasting using on-board ASIX AX88796L Ethernet controller
  As the packets are created, we will send them to the Ethernet controller for transmission on the Internet/LAN. The packets will be broadcasted out instead of being sent to a specific IP.

1.2. Global System Diagram



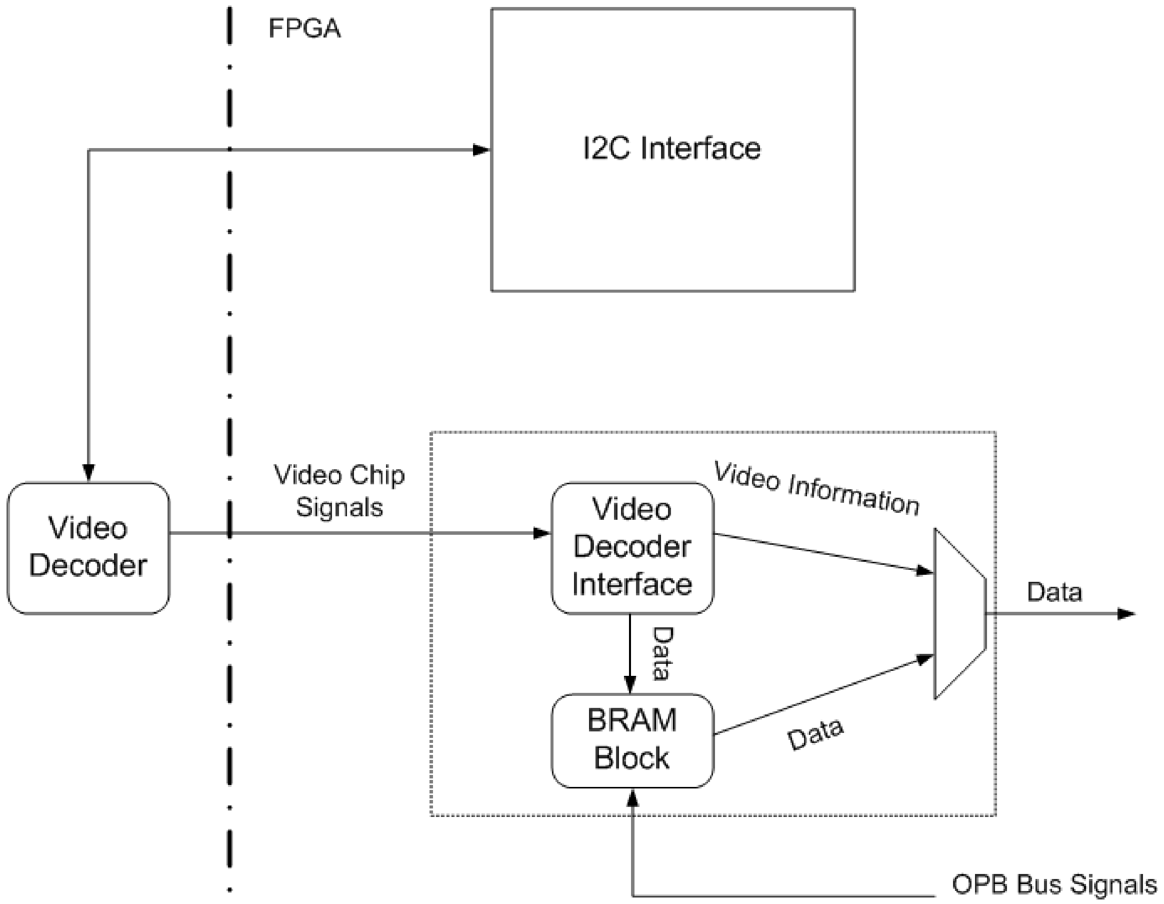
1.3. Global Memory Map

Ethernet: 00A0_0000 → 00A0_FFFFh
Video: 0180_0000h → 0180_3FFFh

2. Video Sub-System

We use the Philips SAA7114H chip to receive and digitize the data. This chip supports NTSC, PAL, and SECAM formatted video. It will take video from either an RCA or an S-Video source.

2.1. Diagram



2.2. Memory Map

Video Memory (Block Rams): 0180_0000h → 0180_3FFFh

2.3. Data Acquisition

Receiving the video data was done using a modified version of Marcio Buss's code. The Philips chip was configured using the I2C bus controller. We then switched two of the registers to support S-Video or RCA depending on what data source we were using. Once the Philips chip was configured video data was written to FPGA Block Ram's to temporarily store it until the JAYCAM software could read it.

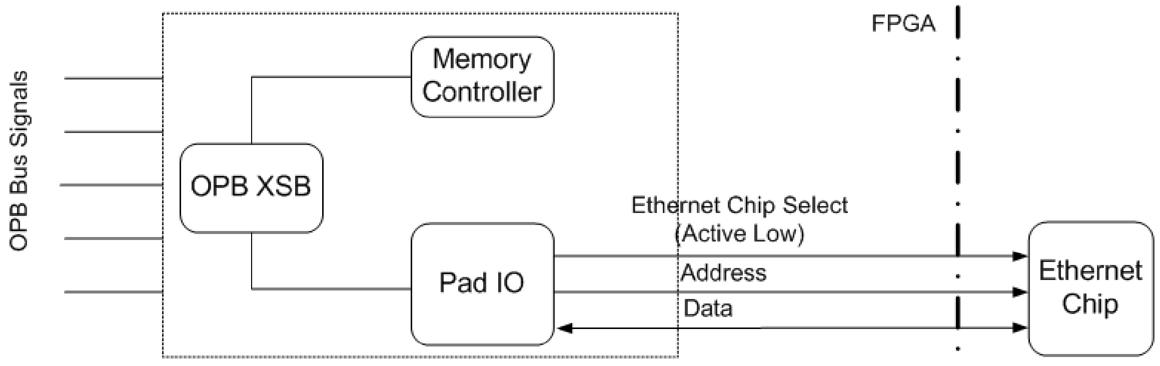
2.4. Other Issues

For detailed descriptions on video initialization and data capture please refer to ObTrak’s (Marcio Buss’s) final report documentation. The only change made from Marcio’s code was that only 160 pixels per line are saved instead of 320 pixels.

3. Ethernet Sub-System

The Ethernet subsystem is essential to JAYCAM's operation as it provides the ability to move captured data to other internet enabled computers and sources capable of receiving UDP packets. Since JAYCAM was developed for broadcasting video, it only takes advantage of the Ethernet's transmission capabilities.

3.1. Diagram

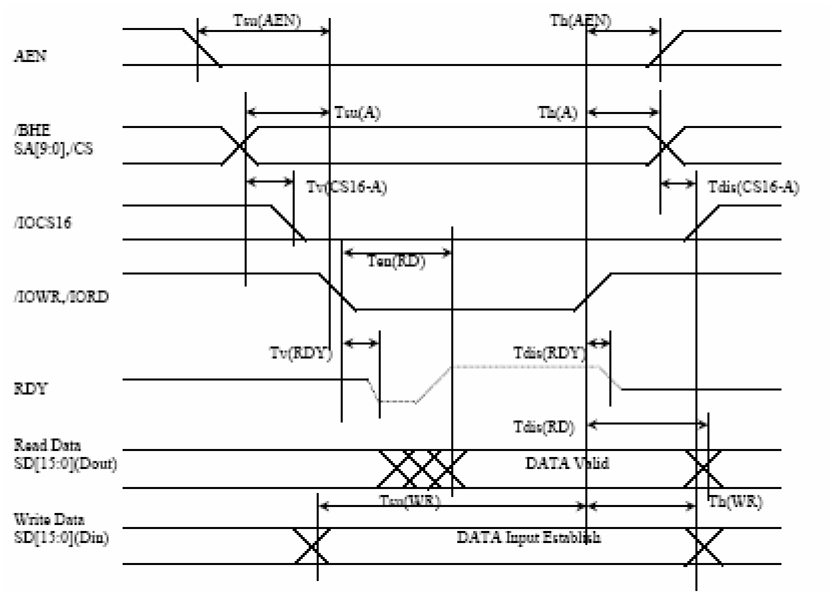


3.2. Memory Map

NE2000 Registers: 00A0_0000h → 00A0_001Fh
SRAM (Local Memory): 00A00_4000h → 00A0_7FFFh

3.3. Ethernet Controller

The Ethernet controller that JAYCAM implemented used the ISA bus timing scheme supplied by the Ethernet chip's manufacturer. Unfortunately, the chip does not respond correctly to this timing. As displayed in the timing diagram (Figure 3.3.1), the Bus High Enable, Address Enable, Address, Chip Select, and Write Enable lines should be able to be asserted simultaneously. In practice this is not the correct timing. Rather, in order for the chip to respond correctly on a consistent basis, the Chip Select signal needs to be asserted before the Write Enable signal. Furthermore, the diagram specifies a 5ns minimum setup time for write-data before the data can be latched. In reality, this is not enough time for certain types of write operations (i.e. DMA remote write). In the end, the Ethernet controller allots 4 cycles for all Ethernet write operations. During the first cycle, all signals, except for Write Enable, are asserted. During the next cycle, Write Enable is asserted causing the address and data signals to become valid. A third "wait" cycle is inserted to account for the DMA write operations that require the data to be valid longer. The last cycle is required for data disabling before releasing the hold on the chip.



3.3.1. ISA Bus Timing Diagram

To make the above timing possible and to save a bit of time, we introduced a 100 MHz clock. That is, the second and third cycles are started on the falling edge of the system clock (i.e. ½ the system clock = 100 MHz). Even though the state-machine is controlled by the 50 MHz system clock, the IO pads at the FPGA boundary run at 100 MHz. This cuts down on the time between state change and signal transfer between the FPGA and Ethernet chip. (To create the 100 MHz clock we doubled the system clock using a DLL that resides at the FPGA boundary.)

3.4. Ethernet Initialization

Before the Ethernet chip (or NIC) is ready to transmit data it must be initialized. This step involves writing specific values to the NE2000 registers. The sequence of steps is outlined below.

1) Abort any current DMA operations and put the NIC in STOP mode. This can be accomplished by writing 21h to the Command Register.
2) Wait at least 2 milliseconds for the inter-frame gap timer to timeout.
3) Enable 16-bit word transfers to and from the chip. This can be accomplished by writing 01h to the Data Control Register.
4) Set the registers that control DMA counter. This can be accomplished by writing 00h to both Remote Byte Count Registers.
5) Place a mask on the interrupts. This can be accomplished by writing 00h to the Interrupt Mask Register.
6) Clear interrupt flags. This can be accomplished by writing FFh to the Interrupt Status Register. (Note: Using the above mask, to clear an interrupt flag, a ‘1’ must be written to the flag bit. Therefore, to clear all flags, write FFh to the register).

7) Put the NIC in monitor mode and loop-back mode. This can be accomplished by writing 20h to the Receive Configuration Register and 02h to the Transmit Configuration Register respectively.
8) Set the RX Start, RX Stop, Boundary and TX start pages. These values depend on the specifics of the system. In our implementation we used 8, 256 Byte pages for transmission buffer space. The first of these pages starts at address 0h, with respect to the on-chip memory.
9) Reset the interrupt mask and flags (see steps 5, 6). This step is probably not necessary although it is resident in many different NE2000 drivers.
10) Start the NIC and set normal transmit operation. This can be accomplished by writing 22h to the Command Register and 00h to the Transmit Configuration Register respectively.

At this point the NIC is ready to send and receive packets.

3.5. Ethernet Transmission

As discussed above, JAYCAM only uses the transmit capabilities of the NIC. Like initialization, to initiate a transmission, it is required to write certain values to certain NE2000 registers.

1) Perform Remote DMA Write
    a. Set internal address to write to. This can be accomplished by writing the low order address bits to the first Remote Start Address Register and the higher order address bits to the second Remote Start Address Register.
    b. Clear the interrupt flags.
    c. Set the number of bytes the DMA should transfer to local memory. This can be accomplished by writing the lower order length bits to the first Remote Byte Count Register and the higher order length bits to the second Remote Byte Count Register.
    d. Set the number of bytes to transmit from local memory. This can be accomplished by writing the lower order length bits to the first Transmit Byte Count Register and the higher order length bits to the second Transmit Byte Count Register.
    e. If not done already, or changed since initialization, set the address of the transmit start page. This is done by writing the higher order byte (i.e. page) of the local memory address to transmit from. This value is written to the Transmit Start Address Register.
    f. Initiate the DMA operation. This can be accomplished by writing 12h to the Command Register.
    g. Write data to the dataport register. Note that the data has to be continuously written in order that the DMA operation does not timeout. In our implementation, word transfers are used so length/2 writes to the dataport are performed.

h. Block until an interrupt signals that the DMA has completed. If after a certain amount of time, the DMA has not completed, abort the operation by writing 21h to the Command Register.
i. If the DMA completes, clear the interrupt flags and start the transmission by writing 24h to the Command Register.
j. Block until an interrupt signals that the transmission was successful. If after a certain amount of time, the transmission has not occurred, assume the transmission failed.

4. Microblaze

JAYCAM uses the built in microprocessor to set up and initialize the video source and the Ethernet controller, as well as control the flow of data from the video source (Block Rams) to the NIC.

4.1. Setup and Initialization

The first step in our program is to configure the video registers. During development these registers were the hardest to get straight. Not only was the manual not specific about certain values, but the hardware seemed to require several optional registers too. After investing large amounts of time to this issue, Marcio was able to compile the correct set of register values to get the chip working. See ObTrak's final project documentation for detailed information on these registers.

We did manage to find the registers that controlled the type video source (S-Video or RCA), as we used different pieces of equipment during testing. There are two such registers that need to be configured.

Register Number	S-Video Value	Analog input value
0x02	0xE8	0xE4
0x09	0xC0	0x40

After video initialization, a quick Ethernet diagnostic is run to see if the NIC is present. This diagnostic includes NE2000 register reads and writes, as well as a reset check. If this diagnostic passes, the NIC is reset and initialized (See Ethernet Subsystem section for details).

After NIC set up, a packet skeleton is created. Since all packets are the same (they are broadcasted as opposed to being sent to a specific computer), the Ethernet, IP, and UDP headers do not change. As such, the standard header is written to the skeleton. This header contains the header for the Ethernet frame, the IP packet header, and the UDP packet header. This is an essential part of the operation as the static headers save rewrites when adding data to the packet. The program then goes into a continuous loop that reads the video data and writes it to the NIC.

4.2. Data Flow Control

The basic C outline for reading the video from hardware was done by Marcio. However it was modified to fit the needs of JAYCAM. First of all, the initial hardware had a 320x480 video resolution (it is actually two different frames of 320x240 that are meant to be vertically interlaced). We had modified the hardware so that it was now a 160x480 resolution. It was important to modify the number of columns in hardware so that a 32-bit read could used to retrieve 4 valid pixels instead of being limited to doing 8 bit reads and then skipping pixels. The first step in decimating the rows was to discard the second half of the interlaced signal. This means that the final resolution is 160x240 for each frame. We then added counter to track the line number and only process every other line. We did not decimate the rows in hardware since we saw no way to optimize the data acquisition by decimating further. In addition, while initially this line skipping was added only to cut the resolution down to the planned 120 lines, by skipping the line, we were able to use that time to write the data to the NIC for transmission. In fact, without this time, transmission would not be possible.

The acquisition of data from the Block Rams is done in the write_video method. An array is passed into this method for data storage. This array is actually a pointer to the data portion of the packet. Although the packet is an array of bytes, it is passed around in different access formats (16-bit words when writing to the NIC, and 32-bit words when saving data). This allows different parts of the program to access the array in the most optimal manner. For example, when the video is read in, it can be read 32 bits at a time. However, the Ethernet chip only allows 16-bit writes, so it is best to have data as 16-bit words. The requirement to store the data in the array as opposed to writing the data to the NIC on the fly was due to the NICs requirement that data be streamed continuously so as not to timeout (See Ethernet Transmission).

4.3. Packet Construction

(All values field values are in hex, unless specified otherwise)

4.3.1. Mac Header (14 bytes)

Destination MAC Address (6 bytes): 01, 00, 5E, 05, 06, 07
Source MAC Address (6 bytes): 00, 0D, 60, 7F, F9, AF
Length (2 Bytes): 08, 00

4.3.2. IP Header (20 Bytes)

Version (4 bits): 0100 (binary)
IHL (4 bits): 0101 (binary)
TOS (1 byte): 0
Total Length (2 bytes): 00, BE
Identification (2 bytes): 00, 00
Flags (3 bits): 000 (binary)
Fragment Offset (13 bits): 0000000000000 (binary)
TTL (1 byte): 01
Protocol (1 byte): 11 (indicates a UDP packet)
Header Checksum (2 bytes): B9, 45
Source IP (4 bytes): 80, 3B, 95, A2 ($\rightarrow$ 128.59.149.162)
Destination IP (4 bytes): E4, 05, 06, 07 ($\rightarrow$ 228.5.6.7)

4.3.3. UDP Header (6 Bytes)

Source Port (2 bytes): 0B, E2 ($\rightarrow$ 3042)
Destination Port (2 bytes): 1A, 85 ($\rightarrow$ 6789)
Length (2 bytes): 00, AA ($\rightarrow$ 170)

Checksum (2 bytes): 00, 00 (disable the checksum)

4.3.4. Data (162 bytes)

Screen Position Information (2 bytes):

Because the screen is split into 120 lines (160 pixels, one line, in a packet), each packet will have a position ranging from 0 to 119.

Data (160 bytes)

5. Client

5.1. Description

The Java client has two simple classes. The Canvas class is in charge of displaying the image, while the Client class does all the rest of the processing. The client classes creates the canvas class of the appropriate size, opens a UDP MulticastSocket. The socket then waits for incoming packets. When a packet is received, it is processed and the image is displayed on the screen.

The format of the data in the packet is simple. The first two bytes are reserved for the row number and the next 160 are the brightness values of the pixels in the row. These brightness values represent a grayscale pixel. Each pixel is represented as one byte. Simplification was the key in the design of the packet organization. All information that was not needed and did not need to be constantly resent was not sent in order to simplify packet creation on the server side.

The interaction between the Client program and the Canvas is done via a shared BufferedImage. When the Client receives a packet the data is written to the buffer. The Client then informs the Canvas to repaint the image so that it displays the new information.

The client code has also been abstracted to allow for easy change of the format of the pixels. Each byte in the packet is first converted to the correct signed form. Then the getFullColor method is called on the byte. This method converts the received byte into the correct color format. Currently it copies the same to value to red, green, and blue to create a grayscale image. If the image was switched to color the only work that would be needed would be to change getFullColor to parse the received value and create the correct RGB value for display.

5.2. Testing

In order to test the client we wrote a basic server to simulate the embedded system. This server cycled through static images and sent them out over a multicast socket. Each packet was designed to mimic how the embedded system would send out packets. The image data was compressed to 1 byte per pixel. Then each line was sent out as a packet. This server allowed us to fully test the client and ensure that it was 100% operational before connecting it to the embedded system. In the end this server provided an added benefit of helping us create the packet header.

The client had been written to accept multicast packets, while JAYCAM was sending broadcast packets. While we had assumed that this was merely different names for the same thing, once we connected the client to JAYCAM we quickly determined that multicast and broadcast were not the same. Luckily we had the server which was sending valid packets. We used packet sniffers (Ethereal on Mac and Iris on Windows) to examine the packets being sent out by the java server. More specifically we were able to observe the source MAC, destination MAC, IP Source and Destination, Packet Number, and IP Header Checksum. We could then correct the packets that JAYCAM was transmitting by copying these values into the static header (See Microblaze Setup and Initialization section). After this, the client was then able to receive and process the data.

6. Miscellaneous

6.1. Lessons Learned

Yaniv

From the beginning this project suffered from a strong dependence on the Xilinx manuals. After studying the Ethernet and Video chip in exhausting details, we found that it wasn't the understanding that was going to be problematic, it's the implementation.

After being misled by the manuals in both the Video Chip and the Ethernet card, we learned to question every parameter. In the Video card, it took some time to discover which settings were wrong in initialization. We sat with Marcio for many hours looking into problems that turned out to be under-specifications in the manual.
Later, when we worked on the Ethernet card, we found very late in development that timing had to be slowed down to get the chip to correctly read the data. This was an annoying bug to catch. We experienced writes to strange portions of the DMA. This manifested itself with packets sending repeated portions of the packet instead of the whole test packet.

On another note, I learned a lot about the laxities of the IP/UDP protocols. Though it was great that we didn't have to change any part of the header, it seems to defy the basic rules of the protocol. I am certain that under further testing we will find that our packets do not travel through certain routers or switches because of 1) the made up MAC and IP addresses and 2) the non-unique packet identification numbers.

Avrum

I learned how important it is to consider all aspects of a project when designing other parts. In too many of my other CS classes I was simply able to rely on the lower level and assume that it worked. I would instantiate a class in Java and then use it without considering how it worked or how long it took. While this approach is fine for most class assignment it created numerous problems with this project. If there are multiple approaches to solving a problem you must remember to take the more efficient one. Instead of using `(x%2)` to check for every other line use `(x&0x1)` to increase performance.

Also, just because code doesn't do anything doesn't mean it doesn't waste time. Initially we were reading every line from the video even if we were only sending out half of them. We had figured it didn't really make a difference since we were just overwriting the data anyway. However, this wasted valuable time and was the difference between sending 80 lines and 120 lines. It was only when we started executing code only when we needed it that we were able to achieve optimal performance.

Another thing I learned is to make it simple to make changes at the highest levels. It was very simple to make a change in the Java, it was fairly simple to make a change in the C, and it was a pain to make a change in the hardware. By making as many changes as possible at the top level we were able to maximize our coding time and have a more configurable system.

Josh
Never trust manufacturers when they tell you something should work with there product. I learned this hard way with the Ethernet. Cristian and I wrote the hardware to interface with NIC according to the timing diagram supplied in the NIC's manual. Some parts worked, but others didn't. After spending weeks with the assumption that the hardware worked, we finally decided to rewrite the hardware. We ditched the timing diagram and basically wrote our own. Amazingly, it worked!

In general this class exposed me to the interaction between hardware and software, and more specifically how complex and sensitive this interaction really is. Little changes in low-level code can make the difference between correct timing and incorrect timing. I learned to take advantage of tricks, like bit shifting (instead of using the everyday operations of multiplying and dividing), in order to save time and system resources.

I also learned not to get ahead of myself! It is much worse being disappointed when you find out something doesn't work when you are confident it works than when you are a little skeptical and more open minded to possible errors. There were many times during the project that I thought things worked, and I was happy. But, when these things in fact turned out to be buggy I was crushed!

All in all, the course taught me a lot about embedded systems and the interaction between hardware and software. I now know what it means to write hardware, and I am excited to work with this stuff in the future!

6.2. Group Member Responsibilities

Group Member	**Responsibilities**
Yaniv Schiller	Client, Microblaze Software, Ethernet Software
Avrum Tilman	Video hardware (I2C bus), Client, Microblaze Software
Josh Weinberg	Microblaze Software, Ethernet Hardware, Ethernet Software

All group members contributed to documentation.

6.3. Special Thanks

We would like to send a shout out to all those who helped us make JAYCAM a reality. Firstly, Marcio Buss is the video man. His effort in getting video up and running was amazing and we thank him greatly for allowing us to use his code. Lastly, Cristian Soviani was unbelievable! The amount of time he spent with us getting the Ethernet to work, and just helping us out in general was unmatched.

Thanks so much you guys, you really made this project possible!

7. Appendix

7.1. System Code

7.1.1. Makefile

```
# Makefile for CSEE 4840, Jaycam

SYSTEM = system

MICROBLAZE_OBJS = \
	c_source_files/jaycam.o \
	c_source_files/etherSend.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/ise6.1i
ISEBINDIR = $(XILINX)/bin/lin
ISEENVCMDS = LD_LIBRARY_PATH=$(ISEBINDIR) XILINX=$(XILINX) PATH=$(ISEBINDIR)

XILINX_EDK = /usr/cad/xilinx/edk3.2

MICROBLAZE = /usr/cad/xilinx/gnu
MBBINDIR = $(MICROBLAZE)/bin
XESSBINDIR = /usr/cad/xess/bin

# Executables

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

# External Targets

all :
	@echo "Makefile to build a Microprocessor system :"
	@echo "Run make with any of the following targets"
	@echo "  make libs     : Configures the sw libraries for this system"
	@echo "  make program  : Compiles the program sources for all the
processor instances"
	@echo "  make netlist  : Generates the netlist for this system
($(SYSTEM))"
	@echo "  make bits     : Runs Implementation tools to generate the
bitstream"
	@echo "  make init_bram: Initializes bitstream with BRAM data"
	@echo "  make download : Downloads the bitstream onto the board"
	@echo "  make netlistclean: Deletes netlist"
	@echo "  make hwclean  : Deletes implementation dir"
	@echo "  make libsclean: Deletes sw libraries"
	@echo "  make programclean: Deletes compiled ELF files"
	@echo "  make clean    : Deletes all generated files/directories"
	@echo " "
	@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)

clean : hwclean libsclean programclean
	rm -f bram_init.sh
	rm -f _impact.cmd

hwclean : netlistclean
	rm -rf implementation synthesis xst hdl
	rm -rf xst.srp $(SYSTEM).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)
	PATH=$$PATH:$(MBBINDIR) XILINX=$(XILINX) XILINX_EDK=$(XILINX_EDK) \
	 perl -I $(XILINX_EDK)/bin/nt/perl5lib $(XILINX_EDK)/bin/nt/libgen.pl \
	  $(LIBGEN_OPTIONS) $(MSSFILE)

# Compilation

MICROBLAZE_CC_CFLAGS =
MICROBLAZE_CC_OPT = -O3 #-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) : $(MHSFILE)
        XILINX=$(XILINX) XILINX_EDK=$(XILINX_EDK) \
        perl -I $(XILINX_EDK)/bin/nt/perl5lib $(XILINX_EDK)/bin/nt/platgen.pl \
          $(PLATGEN_OPTIONS) -st xst $(MHSFILE)
        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 : $(MERGED_BITFILE) $(SRAM_HEXFILE)
        $(XSLOAD) -ram -b $(XESS_BOARD) $(SRAM_HEXFILE)
        $(XSLOAD) -fpga -b $(XESS_BOARD) $(MERGED_BITFILE)
```

7.1.2. system.mhs

```
##############################################################
# CSEE 4840 Embedded System Design
# Jaycam
# Yaniv Schiller - yhs2001@columbia.edu
# Avrum Tilman - amt77@columbia.edu
# Josh Weinberg - jmw211@columbia.edu
##############################################################

# Parameters
PARAMETER VERSION = 2.0.0

# Global Ports

PORT PB_A = PB_A, DIR = OUT, VEC = [19:0]
PORT PB_D = PB_D, DIR = INOUT, VEC = [15:0]
PORT PB_LB_N = PB_LB_N, DIR = OUT
PORT PB_UB_N = PB_UB_N, DIR = OUT
PORT PB_WE_N = PB_WE_N, DIR = OUT
PORT PB_OE_N = PB_OE_N, DIR = OUT
PORT RAM_CE_N = RAM_CE_N, DIR = OUT
PORT ETHERNET_CS_N = ETHERNET_CS_N, DIR = OUT
PORT ETHERNET_RDY = ETHERNET_RDY, DIR = IN
PORT ETHERNET_IREQ = ETHERNET_IREQ, DIR = IN
PORT ETHERNET_IOCS16_N = ETHERNET_IOCS16_N, DIR = IN
PORT VIDOUT_CLK = VIDOUT_CLK, DIR = OUT
PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N, DIR = OUT
PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N, DIR = OUT
PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N, DIR = OUT
PORT VIDOUT_RCR = VIDOUT_RCR, DIR = OUT, VEC = [9:0]
PORT VIDOUT_GY = VIDOUT_GY, DIR = OUT, VEC = [9:0]
PORT VIDOUT_BCB = VIDOUT_BCB, DIR = OUT, VEC = [9:0]
PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN
PORT AU_CSN_N =  AU_CSN_N, DIR=OUT
PORT AU_BCLK =  AU_BCLK, DIR=OUT
PORT AU_MCLK = AU_MCLK, DIR=OUT
PORT AU_LRCK = AU_LRCK, DIR=OUT
PORT AU_SDTI = AU_SDTI, DIR=OUT
PORT AU_SDTO0 = AU_SDTO0, DIR=IN

#Signals for video decoder I2C Bus
PORT VID_I2C_SCL = VID_I2C_SCL, DIR = INOUT
PORT VID_I2C_SDA = VID_I2C_SDA, DIR = INOUT

# Signals of opb_videodec module
PORT IPort = IPort, DIR=IN, VEC=[7:0]
PORT HPort = HPort, DIR=IN, VEC=[7:0]
PORT IDQ = IDQ, DIR=IN
PORT ICLK = ICLK, DIR=IN
PORT IPGV = IPGV, DIR=IN
PORT IPGH = IPGH, DIR=IN
PORT ITRI = ITRI, DIR=OUT
PORT ITRDY = ITRDY, DIR=OUT

# Sub Components
BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_ADDR_TAG_BITS = 6
 PARAMETER C_CACHE_BYTE_SIZE = 2048
 PARAMETER C_ICACHE_BASEADDR = 0x00860000
 PARAMETER C_ICACHE_HIGHADDR = 0x0087FFFF
 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

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

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

BEGIN opb_xsb300
 PARAMETER INSTANCE = xsb300
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00800000
 PARAMETER C_HIGHADDR = 0x00FFFFFF
 PORT PB_A = PB_A
 PORT PB_D = PB_D
 PORT PB_LB_N = PB_LB_N
 PORT PB_UB_N = PB_UB_N
 PORT PB_WE_N = PB_WE_N
 PORT PB_OE_N = PB_OE_N
 PORT RAM_CE_N = RAM_CE_N
 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 OPB_Clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT io_clock = io_clock
 PORT VIDOUT_CLK = VIDOUT_CLK
 PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N
 PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N
 PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N
 PORT VIDOUT_RCR = VIDOUT_RCR
 PORT VIDOUT_GY = VIDOUT_GY
 PORT VIDOUT_BCB = VIDOUT_BCB
 BUS_INTERFACE SOPB = myopb_bus
END

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 pixel_clock = pixel_clock
 PORT fpga_reset = fpga_reset
```

```
END

BEGIN opb_videodec
 PARAMETER INSTANCE = videodec
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x01800000
 PARAMETER C_HIGHADDR = 0x01803FFF
 PORT IPort = IPort
 PORT HPort = HPort
 PORT IDQ = IDQ
 PORT ICLK = ICLK
 PORT IPGV = IPGV
 PORT IPGH = IPGH
 PORT ITRI = ITRI
 PORT ITRDY = ITRDY
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN opb_i2ccontroller
 PARAMETER INSTANCE = i2c
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xFEFF0200
 PARAMETER C_HIGHADDR = 0xFEFF02ff
 PORT VID_I2C_SCL = VID_I2C_SCL
 PORT VID_I2C_SDA = VID_I2C_SDA
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN lmb_lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_lmb_bram_if_cntlr_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x000007FF
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

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
 PORT OPB_Clk = sys_clk
 PORT Interrupt = uart_intr
 BUS_INTERFACE SOPB = myopb_bus
 PORT RX=RS232_RD
 PORT TX=RS232_TD
END

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

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_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END
```

7.1.3. system.mss

```
PARAMETER VERSION = 2.0.0
PARAMETER HW_SPEC_FILE = system.mhs

BEGIN PROCESSOR
 PARAMETER HW_INSTANCE = mymicroblaze
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER EXECUTABLE = system.elf
 PARAMETER COMPILER = microblaze-gcc
 PARAMETER ARCHIVER = microblaze-ar
 PARAMETER DEFAULT_INIT = EXECUTABLE
 PARAMETER STDIN = myuart
 PARAMETER STDOUT = myuart
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = xsb300
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_lmb_bram_if_cntlr_0
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = myuart
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER LEVEL = 0
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = intc
 PARAMETER DRIVER_NAME = intc
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER LEVEL = 0
END
```

7.1.4. system.ucf

```
##############################################################
# CSEE 4840 Embedded System Design
# Jaycam
# Yaniv Schiller - yhs2001@columbia.edu
# Avrum Tilman - amt77@columbia.edu
# Josh Weinberg - jmw211@columbia.edu
##############################################################

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";
```

7.1.5. clkgen.v

```
//-----------------------------------------------------------------------------
// CSEE 4840 Embedded System Design
// Jaycam
// Yaniv Schiller - yhs2001@columbia.edu
// Avrum Tilman - amt77@columbia.edu
// Josh Weinberg - jmw211@columbia.edu
//------------------------------------------------------------------------------

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 locked;

assign pixel_clock = 0;

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)
);

assign fpga_reset = ~locked;
endmodule
```

7.1.6. clkgen_2_0_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


## Ports
PORT FPGA_CLK1 = "", DIR = IN , IOB_STATE = BUF
PORT sys_clk = "", DIR = OUT
PORT pixel_clock = "", DIR = OUT
PORT io_clock ="", DIR = OUT
PORT fpga_reset = "", DIR = OUT
END
```

7.1.7. clkgen_2_0_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
```

7.2. Video Subsystem VHDL Code

7.2.1. opb_i2ccontroller.vhd

```
Opb_i2ccontroller.vhd
--------------------------------------------------------------------------------
-- CSEE 4840 Embedded System Design
-- Jaycam
-- Yaniv Schiller - yhs2001@columbia.edu
-- Avrum Tilman - amt77@columbia.edu
-- Josh Weinberg - jmw211@columbia.edu
-- Written by ObTrack, MudRover, Jaycam
--------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;


entity opb_i2ccontroller is  -- USER --
  generic
  (
    C_OPB_AWIDTH       : integer          := 32;
    C_OPB_DWIDTH       : integer          := 32;
    C_BASEADDR         : std_logic_vector := X"FEFF0200";
    C_HIGHADDR         : std_logic_vector := X"FEFF02FF");

  port
  (
    --Required OPB bus ports, do not add to or delete
    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_Clk      : in  std_logic;
    OPB_DBus     : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW      : in  std_logic;
    OPB_Rst      : in  std_logic;
    OPB_select   : in  std_logic;
    OPB_seqAddr  : in  std_logic;
    VID_I2C_DBus     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    VID_I2C_errAck   : out std_logic;
    VID_I2C_retry    : out std_logic;
    VID_I2C_toutSup  : out std_logic;
    VID_I2C_xferAck  : out std_logic;

    -- USER --
    VID_I2C_SCL      : inout std_logic;
    VID_I2C_SDA      : inout std_logic
  );

end entity opb_i2ccontroller; --USER--

--------------------------------------------------------------------------------
-- architecture
--------------------------------------------------------------------------------

architecture imp of opb_i2ccontroller is --USER--

component IOBUF_F_12
 port (
    O : out STD_ULOGIC;
    IO : inout STD_ULOGIC;
    I : in STD_ULOGIC;
    T : in STD_ULOGIC);
end component;

signal wdata : std_logic_vector(0 to 7);
signal rdata : std_logic_vector(0 to 7);
signal rnw  : std_logic;
signal cs, xfer : std_logic;
signal q0,q1 : std_logic;
signal i2c_din : std_logic;

begin


sda_pad : IOBUF_F_12 port map (
  I => wdata(0),
  IO => VID_I2C_SDA,
  O =>  i2c_din,
  T => wdata(1)
);

scl_pad : IOBUF_F_12 port map (
  I => wdata(2),
  IO => VID_I2C_SCL,
  O => open,
  T => wdata(3)
);

-- Chip select, memory mapped. XIoOut8 for selecting the I2C controller
process (OPB_select, OPB_ABus)
begin

  if(OPB_select='1' and OPB_ABus(0 to 23)=C_BASEADDR(0 to 23)) then
    cs <= '1'; else
    cs <= '0';
  end if;
end process;


-- I2C Bus SDA interconnection
process (OPB_Clk,OPB_Rst)
begin
  if (OPB_Rst='1') then
    wdata <= "11111111";
    rdata <= "00000000";

  elsif OPB_Clk'event and OPB_Clk = '1' then
    rnw <= OPB_RNW;

    if (q1 = '0' and q0 = '1' and rnw='0') then
       wdata <= OPB_DBus(0 to 7);
    end if;

    if (q1='1') then
       rdata <= "00000000";
    elsif (q1='0' and q0='1' and rnw = '1') then
       rdata <= i2c_din & "0000000";
    end if;

  end if;
end process;


process (OPB_Clk,OPB_Rst)
begin
  if (OPB_Rst = '1') then
    q0 <= '0';
    q1 <= '0';
  elsif OPB_Clk'event and OPB_Clk='1' then
    q1 <= not q1 and q0;
    q0 <= not q1 and not q0 and cs;
  end if;
end process;
xfer <= q1;
VID_I2C_xferAck <= xfer;
VID_I2C_DBus <= rdata & X"000000";
VID_I2C_errAck <= '0';
VID_I2C_retry <= '0';
VID_I2C_toutSup <= '0';
end architecture imp;
```

7.2.2. block_ram.vhd

```
-------------------------------------------------------------------------------
-- CSEE 4840 Embedded System Design
-- Jaycam
-- Yaniv Schiller - yhs2001@columbia.edu
-- Avrum Tilman - amt77@columbia.edu
-- Josh Weinberg - jmw211@columbia.edu
-- Written by Marcio
-- Modified for use in Jaycam
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

-- Four RAMB4_S8_S8 components instantiated.
-- Each one stores 8 bits of information (luma)
-- on each memory cell. Block 0 stores pixels
-- 0,4,8, etc. Block 1 stores pixels 1, 5, 9, etc,
-- Block 2 stores pixels 2, 6, 10, etc. and
-- Block 3 stores pixels 3, 7, 11, etc.
-- and so on.
entity block_ram is
  port (
    -- Address generated by video decoder intf module
    -- (video_decoder_intf.vhd). All block-RAMs see
    -- the same 9 *upper* bits. The remaining 2 *lower*
    -- bits are used to choose which block to store.
    waddr : in std_logic_vector (10 downto 0);

    -- Luminance data coming from the video decoder
    -- The video decoder is actually being configured
    -- to transmit 16-bit data (upper bits are luma,
    -- lower bits are chroma). However, the chroma
    -- bits are just being disconsidered as of now.
    data_in : in std_logic_vector (7 downto 0);

    -- Read address. Generated by microblaze every
    -- time one executes XIO_In32. Microblaze reads
    -- four pixels at a time: pixel "i" from block
    -- 0, pixel "i+1" from block 1, pixel "i+2"
    -- from block 2 and pixel "i+3" from block 3.
    -- That's why the *lower* bits of addr are used.
    raddr : in std_logic_vector (8 downto 0);

    -- Data going to microblaze. The 32 bits read
    -- correspond to 4 pixels, each one coming
    -- from a specific block RAM.
    data_out : out std_logic_vector (31 downto 0);

    -- IDQ is '1' when valid data is
    -- coming from video decoder
    idq : in std_logic;

    -- clock for port B is ICLK
    -- from video decoder
    iclk : in std_logic;

    -- From the video decoder
    ipgh : in std_logic;
    -- clock for port A is
    -- clk from CPU
    clock : in std_logic;

    -- Read enable
    read_enable : in std_logic;

    -- Reset
    reset : in std_logic
  );
end block_ram;

architecture structural of block_ram is

-- Dual-port block RAM used for storing data coming from video decoder
-- Port B is written by the video decoder intf, Port A is read by CPU.
-- See  "http://www.xilinx.com/bvdocs/appnotes/xapp173.pdf"
component RAMB4_S8_S8
  generic (
    INIT_00, INIT_01, INIT_02, INIT_03, INIT_04, INIT_05,
    INIT_06, INIT_07, INIT_08, INIT_09, INIT_0a, INIT_0b,
    INIT_0c, INIT_0d, INIT_0e, INIT_0f: bit_vector(255 downto 0)
    :=X"0000000000000000000000000000000000000000000000000000000000000000"
  );
  port  (
    DIA,DIB    : in STD_LOGIC_VECTOR (7 downto 0);
    ENA,ENB    : in STD_logic;
    WEA,WEB    : in STD_logic;
    RSTA,RSTB  : in STD_logic;
    CLKA,CLKB  : in STD_logic;
    ADDRA,ADDRB : in STD_LOGIC_VECTOR (8 downto 0);
    DOA,DOB    : out STD_LOGIC_VECTOR (7 downto 0)
  );
end component;

-- i_clock is ICLK from video decoder
-- opb_clock is opb_clk from OPB bus
signal i_clock : std_logic;
signal opb_clock : std_logic;

-- Read enable
signal r_en : std_logic;

-- Reset
signal rst : std_logic;

-- Shared address bus for all 4 block RAMs
signal addr_a : std_logic_vector (8 downto 0);
signal addr_b : std_logic_vector (8 downto 0);

-- Enable signals for distinct blocks
signal enb0, enb1, enb2, enb3 : std_logic;

-- Data coming from video decoder interface to B ports
signal data_in_signal : std_logic_vector (7 downto 0);

-- Data going to OPB Bus from A ports
signal data_out_a0 : std_logic_vector (7 downto 0);
signal data_out_a1 : std_logic_vector (7 downto 0);
signal data_out_a2 : std_logic_vector (7 downto 0);
signal data_out_a3 : std_logic_vector (7 downto 0);


begin

block_0: RAMB4_S8_S8   -- 512 words of 8 bits
port map
(
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb0,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a0, DOB => open
);


block_1: RAMB4_S8_S8 -- 512 words of 8 bits
port map
(
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb1,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a1, DOB => open
);


block_2: RAMB4_S8_S8 -- 512 words of 8 bits
port map
(
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb2,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a2, DOB => open
);


block_3: RAMB4_S8_S8 -- 512 words of 8 bits
port map
(
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb3,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a3, DOB => open
);

-- Enable signals for each block for writing

-- don't skip any pixels
--enb0 <= idq and ipgh and not waddr(1) and not waddr(0); --not waddr(2) and not
waddr(1) and not waddr(0);  -- "000" -> Y0
--enb1 <= idq and ipgh and     waddr(1) and not waddr(0); --not waddr(2) and
waddr(1) and not waddr(0);  -- "010" -> Y2
--enb2 <= idq and ipgh and not waddr(1) and not waddr(0); --    waddr(2) and not
waddr(1) and not waddr(0);  -- "100" -> Y4
--enb3 <= idq and ipgh and     waddr(1) and not waddr(0); --    waddr(2) and
waddr(1) and not waddr(0);  -- "110" -> Y6

-- skip every other pixel
--enb0 <= idq and ipgh and not waddr(2) and not waddr(1) and not waddr(0);  --
"000" -> Y0
--enb1 <= idq and ipgh and not waddr(2) and     waddr(1) and not waddr(0);  --
"010" -> Y2
--enb2 <= idq and ipgh and     waddr(2) and not waddr(1) and not waddr(0);  --
"100" -> Y4
--enb3 <= idq and ipgh and     waddr(2) and     waddr(1) and not waddr(0);  --
"110" -> Y6

-- write every fourth pixel
enb0 <= idq and ipgh and not waddr(3) and not waddr(2) and not waddr(1) and not
waddr(0);  -- "0000" -> Y0
```

```
enb1 <= idq and ipgh and not waddr(3) and     waddr(2) and not waddr(1) and not
waddr(0);  -- "0100" -> Y4
enb2 <= idq and ipgh and     waddr(3) and not waddr(2) and not waddr(1) and not
waddr(0);  -- "1000" -> Y8
enb3 <= idq and ipgh and     waddr(3) and     waddr(2) and not waddr(1) and not
waddr(0);  -- "1100" -> Y12

-- Data out merger
data_out(31 downto 24) <= data_out_a0;
data_out(23 downto 16) <= data_out_a1;
data_out(15 downto 8)  <= data_out_a2;
data_out(7 downto 0)   <= data_out_a3;

-- Data in
data_in_signal <= data_in;


-- Actual bits addressing block RAMs, port A
addr_a <= raddr;

-- Actual bits addressing block RAMs, port B
addr_b <= "00" & waddr(10 downto 4);

-- Connect clocks and reset
i_clock <= iclk;
opb_clock <= clock;
rst <= reset;

-- Read enable
r_en <= read_enable;

end structural;
```

7.2.3. opb_videodec.vhd

```
-------------------------------------------------------------------------------
-- CSEE 4840 Embedded System Design
-- Jaycam
-- Yaniv Schiller - yhs2001@columbia.edu
-- Avrum Tilman - amt77@columbia.edu
-- Josh Weinberg - jmw211@columbia.edu
-- Written by Marcio
-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;


entity opb_videodec is
  generic (
    C_OPB_AWIDTH : integer := 32;
    C_OPB_DWIDTH : integer := 32;
    C_BASEADDR   : std_logic_vector := X"0180_0000";   -- 512 positions of 32
    C_HIGHADDR   : std_logic_vector := X"0180_3FFF");  -- bits plus extra room.
                                                       -- Each 32 bits in the
                                                       -- block RAMs stores 4
                                                       -- pixels' luminance

  port (
    -- Global signals
    OPB_Clk : in std_logic;
    OPB_Rst : in std_logic;

    -- OPB signals
    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;

    -- Slave signals
    VIDEC_DBus    : out std_logic_vector (31 downto 0);
    VIDEC_errAck  : out std_logic;
    VIDEC_retry   : out std_logic;
    VIDEC_toutSup : out std_logic;
    VIDEC_xferAck : out std_logic;

    -- Coming from SAA7114H
    IPort       : in std_logic_vector (7 downto 0);
    HPort       : in std_logic_vector (7 downto 0);
    IDQ         : in std_logic;
    ICLK        : in std_logic;
    IPGV        : in std_logic;
    IPGH        : in std_logic;
    ITRI        : out std_logic;
    ITRDY       : out std_logic
  );
end opb_videodec;

architecture structural of opb_videodec is

-- Buffered version of the signals
-- with the same name in the entity
signal buf_iclk : std_logic;
signal buf_ipgh : std_logic;
signal buf_ipgv : std_logic;
signal buf_idq : std_logic;
signal buf_iport : std_logic_vector (7 downto 0);
signal buf_hport : std_logic_vector (7 downto 0);
signal buf_itri : std_logic;
signal buf_itrdy : std_logic;

-- Latched versions of the above buffered signals
signal latched_ipgh  : std_logic;
signal latched_ipgv  : std_logic;
signal latched_idq   : std_logic;
signal latched_iport : std_logic_vector (7 downto 0);
signal latched_hport : std_logic_vector (7 downto 0);

-- Signals used when reading from block
-- ram and filling status register
signal cs : std_logic;
signal ce : std_logic;
signal rnw : std_logic;
signal xfer : std_logic;

-- raddr(8 downto 0) is used to address the
-- block RAM. OPB_ABus(13) and OPB_ABus(12), which
-- correspond to raddr(11) and raddr(10), are
-- used to address the filling status register
signal raddr : std_logic_vector (11 downto 0);

-- Signals used by the filling level status
-- The video decoder interface sends a set
-- of signals indicating how much of the
-- current line it has already written into
-- the block RAMs (1/4, 1/2, 3/4 and 1)
-- Microblaze keeps polling this signal
signal filling_level : std_logic_vector(3 downto 0);

-- Count the number of lines being written by the video decoder
signal line_counter : std_logic_vector(15 downto 0);

-- Count the frame (Actually, it's the frame ID
signal frame_counter : std_logic_vector(1 downto 0);

-- Data coming from video decoder interface
signal data_from_decoder : std_logic_vector(15 downto 0);
```

```
-- Data bus and latched data bus
signal data_from_bram : std_logic_vector (31 downto 0);
signal data_bus_ce : std_logic_vector (31 downto 0);

-- Signals for the block ram state machine
signal q2, q1, q0 : std_logic;

-- Coming from video_decoder_intf, going to block_ram
signal intf_idq_out  : std_logic;
signal intf_iclk_out : std_logic;
signal waddr : std_logic_vector (10 downto 0);
signal luma_data : std_logic_vector (7 downto 0);

-- Dummy signals. Reserved for future enhancements
-- We currently not write from microblze (XIo_Out)
signal wdata : std_logic_vector (31 downto 0);
signal be    : std_logic_vector (3 downto 0);

component block_ram is
  port (
    waddr       : in std_logic_vector (10 downto 0);
    data_in     : in std_logic_vector (7 downto 0);
    raddr       : in std_logic_vector (8 downto 0);
    data_out    : out std_logic_vector (31 downto 0);
    idq         : in std_logic;
    iclk        : in std_logic;
    ipgh        : in std_logic;
    clock       : in std_logic;
    read_enable : in std_logic;
    reset       : in std_logic
  );
end component;

component video_decoder_intf is
  port (
    iport     : in std_logic_vector (7 downto 0);
    hport     : in std_logic_vector (7 downto 0);
    idq_in    : in std_logic;
    iclk_in   : in std_logic;
    ipgh      : in std_logic;
    ipgv      : in std_logic;
    data      : out std_logic_vector (15 downto 0);
    waddr     : out std_logic_vector (10 downto 0);
    idq_out   : out std_logic;
    iclk_out  : out std_logic;
    fil_level : out std_logic_vector(3 downto 0);
    line_count: out std_logic_vector(15 downto 0);
    frame_id  : out std_logic_vector(1 downto 0);
    reset     : in std_logic
  );
end component;

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

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

component OBUF
  port(
    O:  out std_ulogic;
    I:  in  std_ulogic
    );
end component;
```

```
component FD
  port (
    C : in std_logic;
    D : in std_logic;
    Q : out std_logic);
end component;

  -- Setting the iob attribute to "true" ensures that instances of these
  -- components are placed inside the I/O pads and are therefore very fast

attribute iob : string;
attribute iob of FD : component is "true";

begin

itrdy_buf : OBUF port map  (
  O => ITRDY,
  I => buf_itrdy
);

itri_buf : OBUF port map  (
  O => ITRI,
  I => buf_itri
);

vbuf : IBUFG port map (
  I => ICLK,
  O => buf_iclk
);

ipgh_pinbuf : IBUF port map (
  I => IPGH,
  O => buf_ipgh
);

ipgh_pinlatch : FD port map (
  C => buf_iclk,
  D => buf_ipgh,
  Q => latched_ipgh
);

ipgv_pinbuf : IBUF port map (
  I => IPGV,
  O => buf_ipgv
);

ipgv_pinlatch : FD port map (
  C => buf_iclk,
  D => buf_ipgv,
  Q => latched_ipgv
);

idq_pinbuf : IBUF port map (
  I => IDQ,
  O => buf_idq
);

idq_pinlatch : FD port map (
  C => buf_iclk,
  D => buf_idq,
  Q => latched_idq
);

databus : for i in 0 to 7 generate
  I_data_pad : IBUF port map (
      I => IPORT(i),
      O => buf_iport(i));

  I_data_ff : FD port map (
      C => buf_iclk,
      D => buf_iport(i),
      Q => latched_iport(i));

  H_data_pad : IBUF port map (
      I => HPORT(i),
      O => buf_hport (i));

  H_data_ff : FD port map (
      C => buf_iclk,
      D => buf_hport(i),
      Q => latched_hport(i));
end generate;

u1 : block_ram
port map
(
    waddr => waddr,
    data_in => luma_data,
    raddr => raddr(8 downto 0),
    data_out => data_from_bram,
    idq => intf_idq_out,
    iclk => intf_iclk_out,
    ipgh => latched_ipgh,
    clock => OPB_Clk,
    read_enable => '1',
    reset => OPB_Rst
);

u2 : video_decoder_intf
port map (
    iport => latched_iport,
    hport => latched_hport,
    idq_in => latched_idq,
    iclk_in => buf_iclk, -- For tests, use OPB_Clk
    ipgh => latched_ipgh,
    ipgv => latched_ipgv,
    data => data_from_decoder,
    waddr => waddr,
    idq_out  => intf_idq_out,
    iclk_out => intf_iclk_out,
    fil_level => filling_level,
    line_count => line_counter,
    frame_id => frame_counter,
    reset => OPB_Rst
);

-- Chip select for block RAM - port A of block RAMs is memory mapped
-- The binary number is X"0180" concatenated with binary "00"
cs <= OPB_select when OPB_ABus(31 downto 14) = "000000011000000000" else '0';


-- Latching read address. Used to address port A of block RAMs
process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_RST = '1' then
      raddr <= "000000000000";
    else
      raddr <= OPB_ABus(13 downto 2);
    end if;
  end if;
end process;


-- Latching RNW signal
process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_Rst = '1' then
      rnw <= '0';
    else
      rnw <= OPB_RNW;
    end if;
  end if;
end process;


-- Latching BE signal (byte enable). Dummy signal
process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_Rst = '1' then
      be <= "0000";
    else
      be <= OPB_BE;
    end if;
  end if;
end process;


-- The following process is dummy. It is used to
-- create a mux between this entity and OPB_DBus
process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_Rst = '1' then
      wdata <= X"0000_0000";
    else
      wdata <= OPB_DBus;
    end if;
  end if;
end process;


-- State machine for reading the block RAM
process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk='1' then
    q2 <= (not q2 and q1) or (q2 and not q1);
    q1 <= (cs and not q2 and not q1) or (q2 and not q1);
    q0 <= q2 and not q1;
  end if;
end process;


-- CE is data latch enable
ce <= q2 and not q1 and rnw;

-- Latch the data coming from the block RAM
-- or from the filling status register
-- at address 01803FFC
process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst='1' then
    data_bus_ce <= X"00000000";
  elsif OPB_Clk'event and OPB_Clk='1' then
    if ce='1' then
      if raddr(11)='1' and raddr(10)='1' then
        data_bus_ce <= "0000000000000000000000000000" & filling_level;
      elsif raddr(11)='1' and raddr(10)='0' then
        data_bus_ce <= "0000000000000000000000000000000" & latched_ipgv;
      elsif raddr(11)='0' and raddr(10)='1' then
        data_bus_ce <= X"0000" & line_counter;
      elsif raddr(11)='0' and raddr(10)='0' and raddr(9)='1' then
        data_bus_ce <= "0000000000000000000000000000000" & frame_counter(0);
      else
        data_bus_ce <= data_from_bram;
      end if;
    else
      data_bus_ce <= X"00000000";
    end if;
  end if;
end process;
```

```
-- Connect luma bits from video decoder interface to
-- block RAMs input data bus
luma_data <= data_from_decoder(15 downto 8);

-- XFER is transfer acknowledge
xfer <= q0;

-- Slave data bus
VIDEC_DBus(31 downto 0) <= data_bus_ce;

-- Tie unused signals to zero
VIDEC_errAck <= '0';
VIDEC_retry <= '0';
VIDEC_toutSup <= '0';

VIDEC_xferAck <= xfer;

buf_itri <= '1';
buf_itrdy <= '1';

end structural;
```

7.2.4. video_decoder_intf.vhd

```
-------------------------------------------------------------------------------
-- CSEE 4840 Embedded System Design
-- Jaycam
-- Yaniv Schiller - yhs2001@columbia.edu
-- Avrum Tilman - amt77@columbia.edu
-- Josh Weinberg - jmw211@columbia.edu
-- Written by Marcio
-------------------------------------------------------------------------------


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity video_decoder_intf is
  port (
    iport     : in std_logic_vector (7 downto 0);
    hport     : in std_logic_vector (7 downto 0);
    idq_in    : in std_logic;
    iclk_in   : in std_logic;
    ipgh      : in std_logic;
    ipgv      : in std_logic;
    data      : out std_logic_vector (15 downto 0);
    waddr     : out std_logic_vector (10 downto 0);
    idq_out   : out std_logic;
    iclk_out  : out std_logic;
    fil_level : out std_logic_vector(3 downto 0);
    line_count: out std_logic_vector(15 downto 0);
    frame_id  : out std_logic_vector(1 downto 0);
    reset     : in std_logic
  );
end video_decoder_intf;

architecture structural of video_decoder_intf is

signal active     : std_logic;
signal pix_count  : std_logic_vector (10 downto 0);
signal pixel_addr : std_logic_vector(10 downto 0);

signal line_counter : std_logic_vector(15 downto 0);
signal frame_counter : std_logic_vector(1 downto 0);

-- The following signals indicate how much of the
-- line was already written into  the  block  RAM
signal one_fourth     : std_logic;
signal half_line      : std_logic;
signal three_quarters : std_logic;
signal entire_line    : std_logic;
signal filling_level  : std_logic_vector(3 downto 0);

begin

-- pixel address - where to store valid pixels in the block RAMs
process (iclk_in, reset)
begin
 if reset='1' then
    pixel_addr <= "00000000000";
 elsif iclk_in'event and iclk_in='1' then
   if ipgh='0' then
     pixel_addr <= "00000000000";
   elsif idq_in = '1' and active = '1' then
     pixel_addr <= pixel_addr + 1;
   end if;
 end if;
end process;

-- count the actual data coming from iport and hport.
-- Some data is control (FF, 00 , 00 , SAV business)
-- Reset the counter whenever ipgh is zero
process (iclk_in, reset)
begin
  if reset='1' then
    pix_count <= "00000000000";
  elsif iclk_in'event and iclk_in='1' then
    if idq_in='1' then
      if ipgh='0' then
        pix_count <= "00000000000";
      else
        pix_count <= pix_count + 1;
      end if;
    end if;
  end if;
end process;

-- count the number of lines
process (iclk_in, reset)
begin
  if reset='1' then
    line_counter <= X"0000";
  elsif iclk_in'event and iclk_in='1' then
    if ipgv='0' then
      line_counter <= X"0000";
    elsif ipgh='1' and pix_count=719 then
      line_counter <= line_counter + 1;
    end if;
  end if;
end process;

-- give the frame ID
process (iclk_in, reset)
begin
  if reset='1' then
    frame_counter <= "00";
  elsif iclk_in'event and iclk_in='1' then
    if line_counter = 239 and pix_count=719 then
      frame_counter <= frame_counter+1;
    end if;
  end if;
end process;

-- Active means we are within
-- the horizontal line active video
process (iclk_in)
begin
```

```
 if iclk_in'event and iclk_in='1' then
   if ipgh='0' then
     active <= '0';
   elsif pix_count = 1 then
     active <= '1';
   elsif pix_count=720 then
     active <= '0';
   end if;
 end if;
end process;

-- Set output signals according to where
-- in the current line the video decoder
-- is writing the block RAM
process (iclk_in, reset)
begin
 if reset='1' then
   one_fourth <= '0';
 elsif iclk_in'event and iclk_in='1' then
   if pix_count=0 then
     one_fourth <= '0';
   elsif pix_count=161 then
     one_fourth <= '1';
   end if;
 end if;
end process;

process (iclk_in, reset)
begin
 if reset='1' then
   half_line <= '0';
 elsif iclk_in'event and iclk_in='1' then
   if pix_count=0 then
     half_line <= '0';
   elsif pix_count=321 then
     half_line <= '1';
   end if;
 end if;
end process;

process (iclk_in, reset)
begin
 if reset='1' then
   three_quarters <= '0';
 elsif iclk_in'event and iclk_in='1' then
   if pix_count=0 then
     three_quarters <= '0';
   elsif pix_count=481 then
     three_quarters <= '1';
   end if;
 end if;
end process;

process (iclk_in, reset)
begin
 if reset='1' then
   entire_line <= '0';
 elsif iclk_in'event and iclk_in='1' then
   if pix_count=0 then
     entire_line <= '0';
   elsif pix_count=641 then
     entire_line <= '1';
   end if;
 end if;
end process;


filling_level(0) <= one_fourth;
filling_level(1) <= half_line;
filling_level(2) <= three_quarters;
filling_level(3) <= entire_line;
```

```
-- Output signals of this entity

fil_level <= filling_level;
line_count <= line_counter;
frame_id <= frame_counter;

data(15 downto 8) <= iport;
data(7 downto 0)  <= hport;

waddr <= pixel_addr;
iclk_out <= iclk_in;
idq_out <= idq_in;

end structural;



-- Test generator
-- signal pixel_data : std_logic_vector(15 downto 0);

-- begin

-- -- pixel data
-- process (iclk_in, reset)
-- begin
--   if reset='1' then
--     pixel_data <= X"00FF";
--   elsif iclk_in'event and iclk_in = '1' then
--       pixel_data <= pixel_data + X"100";
--   end if;
-- end process;

-- -- pixel address - where to store in the block RAMs
-- process (iclk_in, reset)
-- begin
--   if reset='1' then
--      pixel_addr <= "00000000000";
--   elsif iclk_in'event and iclk_in='1' then
--      pixel_addr <= pixel_addr + 1;
--   end if;
-- end process;


-- data <= pixel_data;
-- waddr <= pixel_addr;
-- iclk_out <= iclk_in;
-- idq_out <= '1';

-- end structural;
```

7.2.5. opb_i2ccontroller_v2_0_0.pao

```
###############################################################
#
# opb_i2ccontroller pao file
#
###############################################################

#lib proc_common_v1_00_b    proc_common_pkg
#lib proc_common_v1_00_b    pselect
#lib proc_common_v1_00_b    or_muxcy
#lib ipif_common_v1_00_a    ipif_pkg
#lib ipif_common_v1_00_a    ipif_steer
#lib opb_bus_attach_v1_00_a reset_mir
#lib opb_bus_attach_v1_00_a opb_bus_attach
#lib opb_ipif_ssp0_v1_00_a  opb_ipif_ssp0
```

```
# --USER-- add all user core source files and change the following source to
#          your top level core name and library

lib opb_i2ccontroller_v1_00_a opb_i2ccontroller
lib opb_videodec_v1_00_a opb_videodec

lib opb_videodec_v1_00_a block_ram

lib opb_videodec_v1_00_a video_decoder_intf
```

7.2.6. opb_video_dec_v2_0_0.pao

```
lib opb_videodec_v1_00_a opb_videodec
lib opb_videodec_v1_00_a block_ram
lib opb_videodec_v1_00_a video_decoder_intf
```

7.2.7. opb_i2ccontroller_v2_0_0.mpd

```
###################################################################
##
## Microprocessor Peripheral Definition : generated by psfutil
##
###################################################################

BEGIN opb_i2ccontroller, IPTYPE = PERIPHERAL, EDIF=TRUE # --USER-- change core
name

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

## Generics for VHDL or Parameters for Verilog
PARAMETER c_baseaddr     = 0xFFFFFFFF, DT = std_logic_vector, MIN_SIZE = 0xFF
PARAMETER c_highaddr     = 0x00000000, DT = std_logic_vector
#PARAMETER c_mir_baseaddr = 0xFFFFFFFF, DT = std_logic_vector, MIN_SIZE = 0xFF
#PARAMETER c_mir_highaddr = 0x00000000, DT = std_logic_vector
#PARAMETER c_user_id_code = 3,          DT = integer
#PARAMETER c_include_mir  = 0,          DT = integer
PARAMETER c_opb_awidth   = 32,         DT = integer
PARAMETER c_opb_dwidth   = 32,         DT = integer
#PARAMETER c_family       = spartan2,   DT = string

# --USER-- Add user core parameters

## 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,                             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 VID_I2C_DBus    = Sl_DBus,     DIR = OUT, VEC = [0:(c_opb_dwidth-1)],    BUS
= SOPB
PORT VID_I2C_errAck  = Sl_errAck,   DIR = OUT,                                BUS
= SOPB
PORT VID_I2C_retry   = Sl_retry,    DIR = OUT,                                BUS
= SOPB
PORT VID_I2C_toutSup = Sl_toutSup,  DIR = OUT,                                BUS
= SOPB
PORT VID_I2C_xferAck = Sl_xferAck,  DIR = OUT,                                BUS
= SOPB
```

```
# --USER-- change to user core ports

PORT VID_I2C_SCL          = "",          DIR = INOUT, 3STATE=FALSE, IOB_STATE=BUF
PORT VID_I2C_SDA          = "",          DIR = INOUT, 3STATE=FALSE, IOB_STATE=BUF

END
```

7.2.8. opb_videodec_v2_0_0.mpd

```
################################################################################
##
## Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.
##
## opb_videodec_v2_0_0.mpd
##
## Microprocessor Peripheral Definition
##
################################################################################

PARAMETER VERSION = 2.0.0

BEGIN opb_videodec, IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
#OPTION CORE_STATE = DEVELOPMENT

# Define bus interface
BUS_INTERFACE BUS=SOPB, BUS_STD=OPB, BUS_TYPE=SLAVE

# Generics for vhdl or parameters for verilog
PARAMETER C_OPB_AWIDTH = 32, DT=integer
PARAMETER C_OPB_DWIDTH = 32, DT=integer
PARAMETER C_BASEADDR = 0x01800000, DT=std_logic_vector, MIN_SIZE=0x100, BUS=SOPB
PARAMETER C_HIGHADDR = 0x01803FFF, DT=std_logic_vector, BUS=SOPB


# Global Signals
PORT OPB_Clk = "", DIR=IN, BUS=SOPB, SIGIS=CLK
PORT OPB_Rst = OPB_Rst, DIR=IN, BUS=SOPB

# OPB_VIDEODEC slave signals
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_DBus = OPB_DBus, DIR=IN, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT OPB_RNW = OPB_RNW, DIR=IN, BUS=SOPB
PORT OPB_select = OPB_select, DIR=IN, BUS=SOPB
PORT OPB_seqAddr = OPB_seqAddr, DIR=IN, BUS=SOPB
PORT VIDEC_DBus = Sl_DBus, DIR=OUT, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT VIDEC_errAck = Sl_errAck, DIR=OUT, BUS=SOPB
PORT VIDEC_retry = Sl_retry, DIR=OUT, BUS=SOPB
PORT VIDEC_toutSup = Sl_toutSup, DIR=OUT, BUS=SOPB
PORT VIDEC_xferAck = Sl_xferAck, DIR=OUT, BUS=SOPB

#OPB_VIDEODEC I/O signals
PORT IPort = "", DIR=IN, VEC=[7:0], IOB_STATE=BUF
PORT HPort = "", DIR=IN, VEC=[7:0], IOB_STATE=BUF
PORT IDQ = "", DIR=IN, IOB_STATE=BUF
PORT ICLK = "", DIR=IN, IOB_STATE=BUF
PORT IPGV = "", DIR=IN, IOB_STATE=BUF
PORT IPGH = "", DIR=IN, IOB_STATE=BUF
PORT ITRI = "", DIR=OUT, IOB_STATE=BUF
PORT ITRDY = "", DIR=OUT, IOB_STATE=BUF

END
```

7.3. Ethernet VHDL Subsystem

7.3.1. memoryctrl.vhd

```
--------------------------------------------------------------------------------
-- CSEE 4840 Embedded System Design
-- Jaycam
-- Yaniv Schiller - yhs2001@columbia.edu
-- Avrum Tilman - amt77@columbia.edu
-- Josh Weinberg - jmw211@columbia.edu
--------------------------------------------------------------------------------
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 (r_weth3);
rd_req_i <= (r_common and rnw) or (r_ra and (fullword or eth_io));
wr_req <= wr_req_i;
rd_req <= rd_req_i;
bus_req <= rd_req_i or wr_req_i or r_common;
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
process (clk, rst)
begin
  if rst = '0' then
    r_v0 <= '0';
    r_v1 <= '0';
  elsif clk'event and clk = '1' then
    r_v0 <= not r_v1 and (r_v0 or not vreq);
    r_v1 <= r_v0;
  end if;
end process;
-- TO FIX
-- generate videocycle and videoce 2 cycles later
videocycle_i <= (r_v1 or r_v0) and not (rd_req_i or wr_req_i);
--videocycle <= videocycle_i;
videocycle <= '0';
process (clk, rst)
begin
  if rst = '0' then
    r_v2 <='0';
    video_ce <= '0';
  elsif clk'event and clk = '1' then
    r_v2 <= videocycle_i;
    video_ce <= r_v2;
  end if;
end process;
end Behavioral;
```

7.3.2. opb_xsb300.vhd

```
-------------------------------------------------------------------------------
-- CSEE 4840 Embedded System Design
-- Jaycam
-- Yaniv Schiller - yhs2001@columbia.edu
-- Avrum Tilman - amt77@columbia.edu
-- Josh Weinberg - jmw211@columbia.edu
-------------------------------------------------------------------------------
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;
```

7.3.3. pad_io.vhd

```
-------------------------------------------------------------------------------
-- CSEE 4840 Embedded System Design
-- Jaycam
-- Yaniv Schiller - yhs2001@columbia.edu
-- Avrum Tilman - amt77@columbia.edu
-- Josh Weinberg - jmw211@columbia.edu
-------------------------------------------------------------------------------

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;
```

7.3.4. opb_xsb300_v2_0_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 opb_xsb300_v1_00_a opb_xsb300
lib opb_xsb300_v1_00_a memoryctrl
lib opb_xsb300_v1_00_a svga
lib opb_xsb300_v1_00_a svga_timing
lib opb_xsb300_v1_00_a pad_io
```

7.3.5. opb_xsb300_v2_0_0.mpd

```
################################################################################
##
## Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.
##
## opb_emc_v2_0_0.mpd
##
## Microprocessor Peripheral Definition
##
################################################################################
PARAMETER VERSION = 2.0.0
BEGIN opb_xsb300, IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
#OPTION CORE_STATE = DEVELOPMENT
# Define bus interface
BUS_INTERFACE BUS=SOPB, BUS_STD=OPB, BUS_TYPE=SLAVE
# Generics for vhdl or parameters for verilog
PARAMETER C_OPB_AWIDTH = 32, DT=integer
PARAMETER C_OPB_DWIDTH = 32, DT=integer
PARAMETER C_BASEADDR = 0xFFFFFFFF, DT=std_logic_vector, MIN_SIZE=0x100, BUS=SOPB
PARAMETER C_HIGHADDR = 0x00000000, DT=std_logic_vector, BUS=SOPB

# Signals
PORT OPB_Clk = "", DIR=IN, BUS=SOPB, SIGIS=CLK
PORT OPB_Rst = OPB_Rst, DIR=IN, BUS=SOPB
# OPB slave signals
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_DBus = OPB_DBus, DIR=IN, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT OPB_RNW = OPB_RNW, DIR=IN, BUS=SOPB
PORT OPB_select = OPB_select, DIR=IN, BUS=SOPB
PORT OPB_seqAddr = OPB_seqAddr, DIR=IN, BUS=SOPB
PORT UIO_DBus = Sl_DBus, DIR=OUT, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT UIO_errAck = Sl_errAck, DIR=OUT, BUS=SOPB
PORT UIO_retry = Sl_retry, DIR=OUT, BUS=SOPB
PORT UIO_toutSup = Sl_toutSup, DIR=OUT, BUS=SOPB
PORT UIO_xferAck = Sl_xferAck, DIR=OUT, BUS=SOPB

PORT PB_A = "", DIR=OUT, VEC=[19:0], IOB_STATE=BUF
PORT PB_LB_N = "", DIR=OUT, IOB_STATE=BUF
PORT PB_UB_N = "", DIR=OUT, IOB_STATE=BUF
PORT PB_D = "", DIR=INOUT, VEC=[15:0], 3STATE=FALSE, IOB_STATE=BUF
PORT PB_WE_N = "", DIR = OUT, IOB_STATE=BUF
PORT PB_OE_N = "", DIR = OUT, IOB_STATE=BUF
PORT RAM_CE_N = "", RAM_CE_N, DIR = OUT, IOB_STATE=BUF
PORT ETHERNET_CS_N = "", ETHERNET_CS_N, DIR = OUT, IOB_STATE=BUF
PORT ETHERNET_RDY = "", ETHERNET_RDY, DIR = IN
PORT ETHERNET_IREQ = "", ETHERNET_IREQ, DIR = IN
PORT ETHERNET_IOCS16_N = "", ETHERNET_IOCS16_N, DIR = IN
PORT pixel_clock = "", DIR=IN
PORT io_clock = "", DIR=IN
PORT VIDOUT_CLK = "", DIR=OUT, IOB_STATE=BUF
PORT VIDOUT_RCR = "", DIR=OUT, VEC=[9:0]
PORT VIDOUT_GY  = "", DIR=OUT, VEC=[9:0]
PORT VIDOUT_BCB = "", DIR=OUT, VEC=[9:0]
PORT VIDOUT_BLANK_N = "", DIR=OUT
PORT VIDOUT_HSYNC_N = "", DIR=OUT
PORT VIDOUT_VSYNC_N = "", DIR=OUT

END
```

7.4. Microblaze C Code

7.4.1. etherSend.c

```
/**
```

```
 * CSEE 4840 Embedded System Design
 * Jaycam
 * Yaniv Schiller - yhs2001@columbia.edu
 * Avrum Tilman - amt77@columbia.edu
 * Josh Weinberg - jmw211@columbia.edu
 */

#include "jaycam.h"

static BYTE incr = 0;

BYTE send(WORD *data, WORD addr, WORD dmalen, WORD sendlen){
  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(TBCR0, (sendlen&0xff)); // set Transmit Byte Count
  outnic(TBCR1, (sendlen>>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;
}
```

7.4.2. jaycam.h

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

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

```
// define the size of a packet
#define PACKET_SIZE 204

// 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 DCRVAL   0x01              // 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)

// Private prototypes
void diag();
int init_etherne();
void resetnic();
BYTE send(WORD *data, WORD addr, WORD dmalen, WORD sendlen);

void write_video(int start, int end, int line, Xuint32 *data);
void delay(int mult);
```

7.4.3. jaycam.c

```
/**
 * CSEE 4840 Embedded System Design
 * Jaycam
 * Yaniv Schiller - yhs2001@columbia.edu
 * Avrum Tilman - amt77@columbia.edu
 * Josh Weinberg - jmw211@columbia.edu
 */

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

#define W 640
#define H 480
#define VGA_START 0x00800000
#define BRAM_START 0x01800000
```

```
// Register addresses for SAA7114H configuration
unsigned char registers [] = {

  // Video decoder "generic" registers
  0x01, 0x08, // Recommended setting
  0x02, 0xE4, // Analog input control 1 and input selection Svideo:E8 Analog:E4
  0x03, 0x10, // Analog input control 2
  0x04, 0x90, // Analog input control 3
  0x05, 0x90, // Analog input control 4
  0x06, 0xEB, // Horizontal Sync Start (delay)
  0x07, 0xE0, // Horizontal Sync Stop (delay)
  0x08, 0x59, // Sync control
  0x09, 0x40, // Luminance control Svideo:C0 Analog 40
  0x0A, 0x80,
  0x0B, 0x44,
  0x0C, 0x40,
  0x0D, 0x00,
  0x0E, 0x89,
  0x0F, 0x2A, // Chrominance gain
  0x10, 0x0E, // Chrominance control
  0x11, 0x00,
  0x12, 0x46, // RT signal control
  0x13, 0x00,
  0x14, 0x00,
  0x15, 0x11,
  0x16, 0xFE,
  0x17, 0x40,

  0x18, 0x40,
  0x19, 0x80,
  0x1A, 0x00,
  0x1B, 0x00,
  0x1C, 0x00,
  0x1D, 0x00,
  0x1E, 0x00,
  0x30, 0x08, // Audio clock stuff
  0x31, 0x08, // Audio clock stuff
  0x32, 0x02,

  // Following 1 is added
  0x33, 0x00,

  0x34, 0xCD,
  0x35, 0xCC,
  0x36, 0x3A,

  // Following 1 is added
  0x37, 0x00,

  0x38, 0x03,
  0x39, 0x10,
  0x3A, 0x00,


  // Following 5 are added
  0x3B, 0x00,
  0x3C, 0x00,
  0x3D, 0x00,
  0x3E, 0x00,
  0x3F, 0x00,

  0x40, 0x40,
  0x41, 0xFF,
  0x42, 0xFF,
  0x43, 0xFF,
  0x44, 0xFF,
  0x45, 0xFF,
  0x46, 0xFF,
  0x47, 0xFF,
  0x48, 0xFF,
  0x49, 0xFF,
  0x4A, 0xFF,
  0x4B, 0xFF,
  0x4C, 0xFF,
  0x4D, 0xFF,
  0x4E, 0xFF,
  0x4F, 0xFF,
  0x50, 0xFF,
  0x51, 0xFF,
  0x52, 0xFF,
  0x53, 0xFF,
  0x54, 0xFF,
  0x55, 0xFF,
  0x56, 0xFF,
  0x57, 0xFF,

  0x58, 0x40,
  0x59, 0x47,
  0x5A, 0x06,
  0x5B, 0x03,


  // Following 1 is added
  0x5C, 0x00,

  0x5D, 0x3E,
  0x5E, 0x00,

  // Following 1 is added
  0x5F, 0x00,


  0x80, 0x30,
  0x83, 0x01,
  0x84, 0xA0,
  0x85, 0x10,
  0x86, 0x45,
  0x87, 0x01,
  0x88, 0xF0,

  // Task A Registers
  0x90, 0x00,
  0x91, 0x08,
  0x92, 0x10,
  0x93, 0xC0,
  0x94, 0x10,
  0x95, 0x00,
  0x96, 0xD0,
  0x97, 0x02,
  0x98, 0x0A,
  0x99, 0x00,
  0x9A, 0xF2,
  0x9B, 0x00,
  0x9C, 0xD0, // Horizontal output window size upper bits \ 0xD002 = 720
  0x9D, 0x02, // Horizontal output window size lower bits /          by
  0x9E, 0xF0, // Vertical output window size upper bits  \  0xF000 = 240
  0x9F, 0x00, // Vertical output window size lower bits  /
  0xA0, 0x01,
  0xA1, 0x00,
  0xA2, 0x00,
  0xA4, 0x80,
  0xA5, 0x40,
  0xA6, 0x40,
  0xA8, 0x00,
  0xA9, 0x04,
  0xAA, 0x00,
  0xAC, 0x00,
  0xAD, 0x02,
  0xAE, 0x00,
  0xB0, 0x00,
  0xB1, 0x04,
  0xB2, 0x00,
  0xB3, 0x04,
  0xB4, 0x00,
  0xB8, 0x00,

  // Following 3 were added
  0xB9, 0x00,
  0xBA, 0x00,
  0xBB, 0x00,

  0xBC, 0x00,

  // Following 3 were added
  0xBD, 0x00,
  0xBE, 0x00,
  0xBF, 0x00,


  // Task B Registers
  0xC0, 0x00,
  0xC1, 0x08,
  0xC2, 0x10,
  0xC3, 0xC0,
  0xC4, 0x10,
  0xC5, 0x00,
  0xC6, 0xD0,
  0xC7, 0x02,
  0xC8, 0x0A,
  0xC9, 0x00,
  0xCA, 0xF2,
  0xCB, 0x00,
  0xCC, 0xD0,
  0xCD, 0x02,
  0xCE, 0xF0,
  0xCF, 0x00,
  0xD0, 0x01,
  0xD1, 0x00,
  0xD2, 0x00,
  0xD4, 0x80,
  0xD5, 0x40,
  0xD6, 0x40,
  0xD8, 0x00,
  0xD9, 0x04,
  0xDA, 0x00,
  0xDC, 0x00,
  0xDD, 0x02,
  0xDE, 0x00,
  0xE0, 0x00,
  0xE1, 0x04,
  0xE2, 0x00,
  0xE3, 0x04,
  0xE4, 0x00,
  0xE8, 0x00,
  0xE9, 0x00,
  0xEA, 0x00,
  0xEB, 0x00,
  0xEC, 0x00,
  0xED, 0x00,
  0xEE, 0x00,
  0xEF, 0x00,

  // Reset sequence.
  0x88, 0xD8,
  0x88, 0xF8,
  0xFF, 0xFF,};

int w = 0xFF;

void i2c_delay()
{
  int i;
  for (i = 0; i < 10000; i++);
}

// Write "level" to SCL
void SCLw(int level)
{
  if (level == 0)
    w &= 0xDF;
  else
    w |= 0x2F;

  // Assert the clock on SCL according to level
  XIo_Out8(0xFEFF0200, w);
  i2c_delay();
}

// Write "level" to SDA
void SDAw(int level)
{
  if (level == 0)
    w &= 0x7F;
  else
    w |= 0x8F;

  // Assert the clock on SDA according to level
  XIo_Out8(0xFEFF0200, w);
  i2c_delay();
}

// Read from SDA
int SDAr()
{
  int MSB = XIo_In8(0xFEFF0200);

  MSB = MSB >> 7;
  MSB &= 1;

  i2c_delay();
  return MSB;
}

// Tristate for SDA
void SDAt(int rnw)
{
  if (rnw == 0)
    w &= 0xBF;
  else
    w |= 0x4F;

  // Assert the clock on SDA according to level
  XIo_Out8(0xFEFF0200, w);
  i2c_delay();
}

// Tristate for SCL //
void SCLt(int rnw)
{
  if (rnw == 0)
    w &= 0xEF;
  else
    w |= 0x1F;

  // Assert the clock on SDA according to level
  XIo_Out8(0xFEFF0200, w);
  i2c_delay();
}

// Send the start sequence
void send_start( void )
{
  SCLt(0);
  SDAt(0);
  SCLw(1);
  SDAw(0);
  SCLw(0);
}

// Send the restart sequence
// Needed for read register
// This function must be entered with SDA High
void re_start( void )
{
  SCLw(1);
  SDAw(0);
  SCLw(0);
}


// Send stop sequence
void send_stop( void )
{
  SCLw(0);
  SDAw(0);
  SCLw(1);
  SDAw(1); // Should leave with both lines high to indicate finish
}

// Check acknowledge
int check_ack(void)
{
  int theresult;
  SDAt(1);
  SCLw(1);
  theresult=SDAr();
  SCLw(0);
  SDAw(1); // Set the output before it becomes active to eliminate spike
  SDAt(0);
  return theresult;
}

// Send one bit
void send_bit(int x)
{
  x = x & 1;
  SDAw(x);
  SCLw(1);
  SCLw(0);
}

// Send an entire bit
void send_byte(int byte)
{
  int i;
  for (i = 7; i >= 0; i--)
    {
      send_bit(byte >> i);
    }
}

// Read a register from the video decoder
int read_register( int sub_address )
{
  int input = 0;
  int id;

  send_start();

  // Write slave address for SAA7114H is 43H
  send_byte(0x42);

  check_ack();
  // Send the subaddress
  send_byte(sub_address);

  check_ack();

  re_start();

  // Read address
  send_byte(0x43);

  check_ack();

  SDAt(1);
  for( id=8 ; id>0 ; id=id-1 )
    {
      input=input<<1;
      SCLw(1);
      input=input|SDAr();
      SCLw(0);
    }
  // Set the output prior to enable to eliminate spike and
  // make compatible with Restart
  SDAw(1);
  SDAt(0);
  SCLw(1);
  SCLw(0);
  send_stop();
  return input;
}

// Write a register into the video decoder
void write_register(int sub_address, int data)
{
  int i;

  // Start conditions
  send_start();

  for (i = 0; i < 5; i++)
    i2c_delay();

  // Write slave address for SAA7114H is 42H
  send_byte(0x42);

  check_ack();

  send_byte(sub_address);

  check_ack();

  send_byte(data);

  check_ack();

  send_stop();
}

BYTE packet[PACKET_SIZE];

int main()
{
  Xuint8 bw_1pixel;
  Xuint32 bw_4pixels;
  Xuint32 luma_4pixels;
  Xuint32 current_level;
  Xuint32 vga_addr;

  BYTE my_count=0;

  WORD word;
  WORD *hdr;
```

```
Xuint32 *data;

int frame_id;
int line_number = 0;
int frame_line = 0;
int start, end;
int i, j, x;
int line_section;

print("\r\n\r\n-------------  Welcome to JayCAM ------------------\r\n\r\n");

microblaze_enable_icache();

print("***** Configuring the Video Decoder *****\r\n\r\n");
// Start the bus  protocol  by  sending
// a stop handshaking (SDA=1 and SCL=1)
send_stop();

i = 0;

// Configure the video decoder SAA7114H //
while (registers[i] != 0xFF) {
  write_register (registers[i], registers[i+1]);
  i+=2;
}

// Wait and then check how the Philips chip responded to the configuration
for (i=0; i<100000000;i++);
if(read_register(0x1F) == 0xA1){
  print("Video decoder configured!\r\n");
}
else{
  print("Video decoder not configured!...");
  putnum(read_register(0x1F));
  print("\r\n");
  return;
}

print("\r\n\r\n***** Ethernet Diagnostic *****\r\n\r\n");
diag();
print("\r\n\r\n***** Ethernet Configuration *****\r\n\r\n");

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

print("\r\n\r\n***** Starting JayCAM *****\r\n\r\n");

// create non-changing packet header (ethernet, ip, udp headers)
print("Creating Ethernet, IP, and UDP headers...");
packet[0]=0x01;
packet[1]=0x00;
packet[2]=0x5e;
packet[3]=0x05;
packet[4]=0x06;
packet[5]=0x07;
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]=0xB9;
packet[25]=0x45;
packet[26]=0x80;
packet[27]=0x3B;
packet[28]=0x95;
packet[29]=0xA2;
packet[30]=0xE4;
packet[31]=0x05;
packet[32]=0x06;
packet[33]=0x07;
packet[34]=0x0B;
packet[35]=0xE2;
packet[36]=0x1A;
packet[37]=0x85;
packet[38]=0x00;
packet[39]=0xAA;
packet[40]=0x00;
packet[41]=0x00;
packet[42]=0x00; //extra byte
print("Done!\r\n");

print("Writing headers to memory...");
// access packet contents as 16-bit words
hdr = (WORD *)&packet;

outnic(RSAR0, 0); // set DMA starting address
outnic(RSAR1, TXSTART);

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

outnic(RBCR0, 42); // set Remote DMA Byte Count
outnic(RBCR1, 0);

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<21; i++){
  word = (hdr[i]<<8)|(hdr[i]>>8);
  outnic(DATAPORT, word);
}

if(innic(ISR)&0x40){
  print("Done!\r\n");
}
else{
  print("Error! DMA did not finish. Restart the board\r\n\r\n");
  return;
}

// set 32-bit data pointer for writing to the packet
// 21 words => 42 bytes of header information
data = (Xuint32 *)(hdr+21);

// Wait and then check how the Philips chip responded to the configuration
for (i=0; i<100000000;i++);
if(read_register(0x1F) == 0xA1){
  print("Starting video capture...Sending data!\r\n");
}
else{
  print("Video failed to start!\r\n\r\n");
  return;
}

// start grabbing data
while (1)
```

```
{
  frame_line = 0;
  line_number = 0;

  // Wait for the vertical synchronism
  while (!(XIo_In32(0x01802FFC)));

  // each iteration corresponds to a full line
  while (1)
  {
    // only use 1 half of the frames
    // (discard interlaced frame)
    if(!XIo_In32(0x018008FC))
      continue;

    // save only one-half of the lines
    frame_line = frame_line + 1;
    if(frame_line&0x1)
      line_number++;

    packet[43] = (BYTE)(line_number&0xFF);
    data = (Xuint32 *)(&packet[44]);

    // This variable indicates how much of
    // the current line has  been  already
    // written into the block RAMs
    current_level = 0x0001;
    if(!(frame_line&0x1)) {
    for (line_section = 0; line_section < 4; line_section++, data+=10)
    {
      // Wait for the current line to be  25%, 50%, 75%, and 100% filled.
      // The loop then executes 4 times
      while (!(XIo_In32(0x01803FFC) & current_level))
      {
        // reset if a new frame is detected
        if (!XIo_In32(0x01802FFC))
          break;
      }

      if (current_level == 0x01) {
        start = 0;
        end = 160;
      }
      else if (current_level == 0x02) {
        start = 160;
        end = 320;
      }
      else if (current_level == 0x04) {
        start = 320;
        end = 480;
      }
      else if (current_level == 0x08) {
        start = 480;
        end = 640;
      }

      // reset if a new frame is detected
      if (!XIo_In32(0x01802FFC))
        break;

      write_video(start, end, line_number, data); // put data in the packet
      current_level = current_level << 1;

      // reset if a new frame is detected
      if (!XIo_In32(0x01802FFC))
        break;
    } // end of line for loop
    }
    // send every other line
    if(frame_line&0x1)
      send((WORD *)&packet[42], (TXSTART<<8)+42, 162, 204);
      // reset if a new frame is detected
      if (!XIo_In32(0x01802FFC))
          break;
    } // end of line while loop
  } // end of frame while loop

  print("\r\nExiting JayCAM\r\n\r\n");
  return 0;
}


void write_video(int start, int end, int line, Xuint32 *data)
{
  int nPixs;
  Xuint32 luma_4pixels;
  Xuint32 bram_addr;

  nPixs = (end - start);
  bram_addr = BRAM_START + (start>>2);
  while (nPixs > 0)
  {
    luma_4pixels = XIo_In32(bram_addr+0);
    *data = luma_4pixels;
    data++;

    luma_4pixels = XIo_In32(bram_addr+4);
    *data = luma_4pixels;
    data++;

    bram_addr += 8;
    nPixs -= 32;

    // reset if a new frame is detected
    if (!XIo_In32(0x01802FFC))
      break;
  }
}

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

  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*2,0x4e);
  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");
}

int init_etherne()
{
  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!\r\n");
    resetnic(); // Reset Ethernet card,
    print("Ethernet card intialization complete!\r\n");
  }
  return 1;
}

void resetnic(){
  int i, count;

  outnic(CMDR, 0x21);  // Abort and DMA and stop the NIC
  delay(10);

  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(TCR, 0x02);  // put nic in normal operation

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

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

  // put nic in start mode
  outnic(CMDR, 0x22); // start nic
  outnic(TCR, 0x00);  // put nic in normal operation
}

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

7.5. Client Java Code

7.5.1. Client.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.awt.image.*;
import java.net.*;
public class Client
{
    private static int getFullColor(int byteColor)
    {
       int r=byteColor&0xe0;
       int g=(byteColor&0x1c)<<3;
       int b=(byteColor&0x03)<<6;

       r=g=b=byteColor;
       return (r<<16)|(g<<8)|b;
    }
    public static void main(String args[]) throws Exception
    {
       int height=Integer.parseInt(args[2]);
       int width=Integer.parseInt(args[3]);
       Screen scn = new Screen(width+20,height+20,BufferedImage.TYPE_INT_RGB);
       JFrame app = new JFrame("Jaycam - RTVS");
       InetAddress group = InetAddress.getByName(args[0]);
       MulticastSocket s = new MulticastSocket(Integer.parseInt(args[1]));
       s.joinGroup(group);
       app.setSize(width+20,height+40);
       app.getContentPane().add(scn);
       WindowListener l = new WindowAdapter()
           {
              public void windowClosing(WindowEvent e)
              {
                  System.exit(0);
              }
           };

       app.addWindowListener(l);
       app.show();

       int len;
       boolean temp=true;
       int frameNum=0;
       long start=0,end=0;
       while(true)
       {
           byte[] buf = new byte[2000];
           DatagramPacket recv = new DatagramPacket(buf, buf.length);
           s.receive(recv);
           len=recv.getLength();
           int nrows=1;
           int ncols=160;
           int row=MyUtil.unsigned(buf[1])-1;
           //get packet data
           for(int r=0;r<nrows;r++) {
              for(int c=0;c<ncols;c++)
                  try{
                      synchronized(scn.img){
                         scn.img.setRGB(c,row+r,
                             getFullColor(MyUtil.unsigned(buf[r*ncols+c+2])));
                      }
                  } catch(Exception e){
                  }
//            scn.reDrawLine(r);
           }
           if(row==0)
             scn.repaint();
       }

    }
}
```

7.5.2. Screen.java

```
import java.awt.*;
import java.awt.image.*;
public class Screen extends Canvas
{
    public BufferedImage img;
    public Screen(int w, int h, int t)
    {
       img = new BufferedImage(w,h,t);
    }
    public void paint(Graphics g)
    {
       synchronized(img)
       {
           g.drawImage(img,0,0,this);
       }
    }
    public void reDrawLine(int l) {
       int width;
       BufferedImage imgLine;
       Graphics g=getGraphics();
       if(g!=null) {
           synchronized(img) {
              width=img.getWidth();
              imgLine=img.getSubimage(0,l,width,1);
           }
           g.drawImage(img,0,l,width,1,this);
       }
    }
}
```

7.5.3. MyUtil.java

```
/*
 * Utility class to do some basic conversions
 * Avrum Tilman
 * amt77@columbia.edu
 * avrum@cs.columbia.edu
 */
public class MyUtil
{
        public static byte[] subArray(byte[] input, int start, int length)
        {
                byte[] out = new byte[length];

                for(int l=0;l<length;l++)
                        out[l]=input[start+l];
                return out;
        }

        public static byte[] convert(int val)
         //get a byte array for the given integer
        {
                byte[] out = new byte[256];
                byte[] ret;
                Integer parse;
                int mod, l=0;

                while(val>0)
                {
                        mod=val%256;
                        val=val/256;
                        parse = new Integer(unsigned(mod));
                        out[l]=parse.byteValue();
                        l++;
                }

                ret = new byte[l];
                for(int i=0;i<l;i++)
                {
```

```
				ret[i]=out[l-i-1];
			}

			return ret;
		}

		public static int convert(byte[] val)
		//convert the given byte array back into an integer
		{
			int ret;

			ret=unsigned(val[0]);
			for(int l=1;l<val.length;l++)
			{
				ret=ret*256+unsigned(val[l]);
			}

			return ret;
		}

		public static int unsigned(int x)
		//keep the bit value correct for unsigned number
		{
			if (x>127)
			{
				return (x-256);
			}
			return x;
		}

		public static int unsigned(byte x)
		//get the real integer value given an unsigned byte
		{
			if(x<0)
			{
				return (x+256);
			}
			return x;
		}

		public static void printBytes(byte[] data)
		//print all the bytes for debugging purposes
		{
			for(int i=0;i<data.length;i++)
				System.out.print(Integer.toBinaryString(data[i]) + " ");
			System.out.println("");
		}
}
```

7.5.4. Server.java

```
import java.io.*;
import javax.imageio.*;
import java.awt.*;
import java.awt.image.*;
import java.net.*;
public class Server {

    private static int getFullColor(int byteColor) {
        int r = byteColor & 0xe0 >> 5;
        int g = (byteColor & 0x1c) >> 2;
        int b = (byteColor & 0x3);

        Color temp = new Color(r, g, b);
        //return temp.getRGB();
        return ((r * 32) << 16) + ((g * 32) << 8) + b * 64;
    }

    public static void main(String args[]) throws Exception {
        int rows = Integer.parseInt(args[3]);

        InetAddress group = InetAddress.getByName(args[0]);
        int port = Integer.parseInt(args[1]);
        MulticastSocket s = new MulticastSocket();
        //      s.joinGroup(group);

        String mypics[] =
            {
               //                                    "aaron.jpg",
               "glavine.jpg",
               //"mattingly.jpg",
               "rollins.jpg",
               "piazza.jpg"
            };
        int n=0;
        while(true){
            n++;
            //                  for (int n = 0; n < mypics.length; n++) {
            BufferedImage img = ImageIO.read(new File(mypics[n%mypics.length]));
            int height = img.getHeight();
            int width = img.getWidth();
            //      System.out.println(width + "x" + height);

            int count;
            byte[] buf;
            // System.out.println("Rows=" + rows);
            int numPackets =
               (int) Math.ceil((double) height / (double) rows);
            //                  int[] pixels = new int[height * width];
            //                  int l = 0;
            for (int g = 0; g < numPackets; g++) {
               buf = new byte[3 + rows * width];
               buf[0] = (byte) MyUtil.unsigned(rows);
               buf[1] = (byte) MyUtil.unsigned(width);
               buf[2] = (byte) MyUtil.unsigned(g * rows);
               count = 3;
               //          System.out.println(g + " * " + rows + " = " + g*rows);
               for (int r = g * rows; r < (g + 1) * rows; r++) {
                   //          System.out.println("Row=" + r);
                   if (r >= height) {
                      buf[0] = (byte) MyUtil.unsigned(r % rows);
                      if (buf[0] == 0)
                          buf[0] = (byte) 4;
                      break;
                   }
                   for (int c = 0; c < width; c++) {
                      int out;
                      Color px = new Color(img.getRGB(c, r));
                      out = px.getRed()&0xe0;
                      out |= (px.getGreen()&0xe0)>>3;
                      out |= (px.getBlue()&0xc0)>>6;

                      buf[count++] = (byte) MyUtil.unsigned(out);
                   }
               }
               //          continue;
               if (count != 3) {
                   //              System.out.println("SERVER: Sending packet " + g
+ " of length " + count);
                   //          if(g==0)
                   //  MyUtil.printBytes(buf);
                   s.send(new DatagramPacket(buf, count, group, port));
                   //                               scn.repaint();
               }

            }
            Thread.sleep(500);
        }
```

```
    }
}
```

7.5.5. Makefile

```
all: clean compile
compile: Client.java Server.java Screen.java MyUtil.java
	javac *.java
client: compile
	java Client "228.5.6.7" 6789 120 160 &
server: compile
	java Server "228.5.6.7" 6789 rollins.jpg 4 &
stop:
	ps -aux | grep Server | awk '{ print "kill " $$2 }' > killp
	sh killp
	rm -rf killp
clean:
	rm -rf *.class *~
```