SPYCAM

REPORT

KSHITIJ MISRA
AMIT MEHTA
KEN TANG

Table of Contents

1. Introduction

1.1 Abstract

The SPYCAM project team proposes to build a security camera using the XESS XSB300 board. MPEG (motion JPEG images) will be taken from an internet camera and will be fed in to the FPGA using the Ethernet. These images will be stored on the SRAM, where there will be a program provided by the SPYCAM team that decompresses these JPEG images and this program must be severely modified to compile with the Microblaze.

These decompressed images will be moved from the SRAM on to the screen. The details of how this decompression was achieved are provided in section 3. The clock cycle distribution to Ethernet reads and writes and SRAM reads and writes and video requests are discussed in more detail in section 4, State Machine. Section 3 provides details on the implementation of the ethernet. The team members have included some personal notes in section 5 and section 6 is a repository of the code we used in this projects.

1.2 Peripherals used with the FPGA

The SPYCAM projects mainly make use of two peripherals. The peripherals used are as mentioned below:

- SRAM, Toshiba TC55V16256J
- Ethernet Controller AX88796
- INTELLINET Internet Camera

The INTELLINET internet camera is the source for our MPEG images. The camera has its own embedded server system in it. The camera works on several protocols such as TCP, UDP, and FTP. We attempted to utilize the TCP functionality of the camera using a GET command asking for the file “serveraddress/jpg/image.jpg”. The server would be able to return images with up to 10 levels of JPEG compression and with sizes of 160 x 120, 320 x 240, and 640 x 480.

1.3 Overall Design Schematic

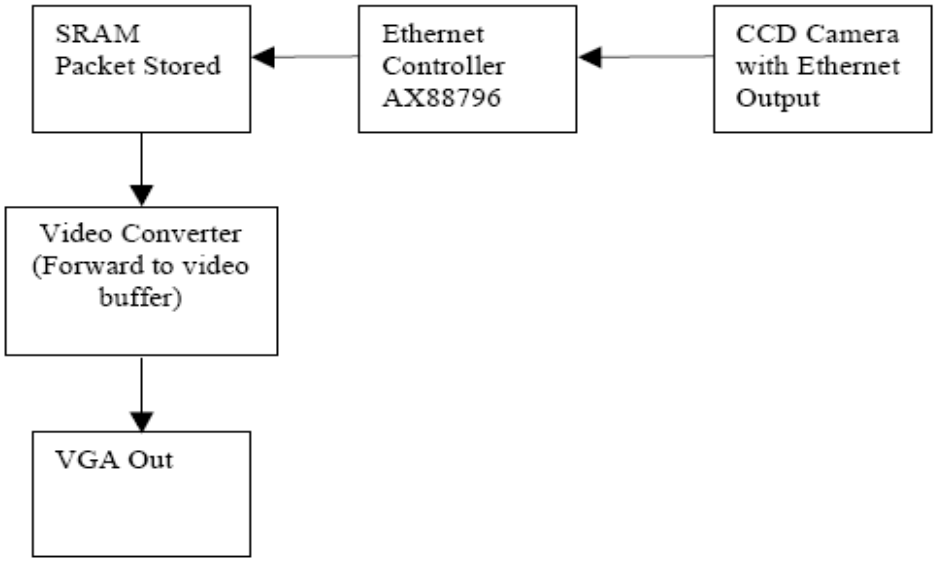

Figure 1.1 Spycam design schematic

1.4 System Block Diagram

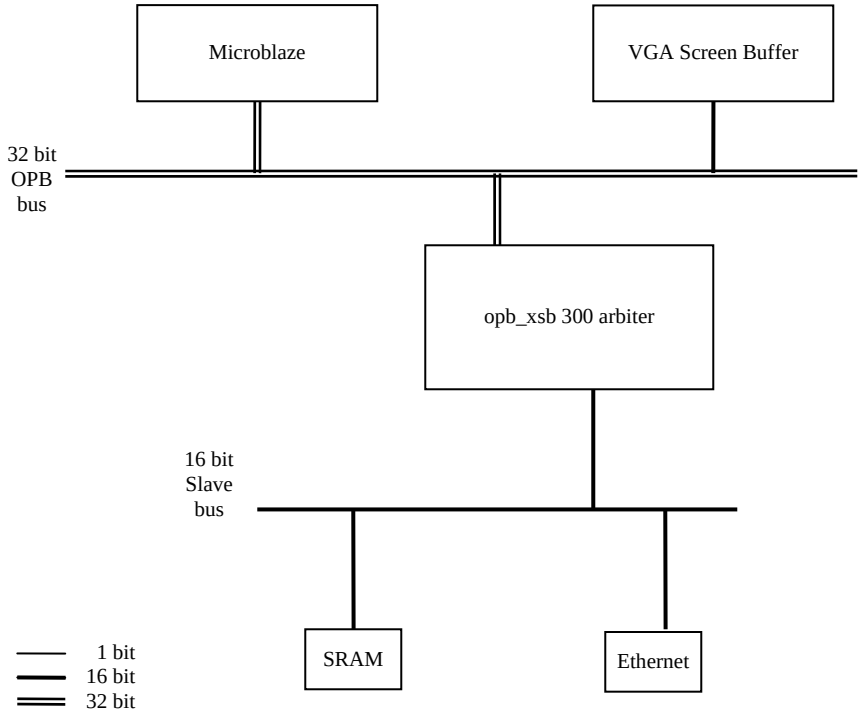

Figure 1.2 Spycam block diagram

As can be seen from the block diagram, the SRAM and the Ethernet reside on the 16-bit slave bus and require the OPB Bridge to translate the various 32-bit, 16-bit and the 8-bit accesses coming in from the microblaze. The OPB bridge peripheral is described in further detail in section 4.1.1.

2. Ethernet Controller

2.1 Ethernet Timing

The Ethernet reads and write cycles require precise timing for them to operate, which we discovered after a trying many timing combinations that made sense based upon the Ethernet Controller documentation but failed to provide consistent results.

The read and write timings that we finally used with the original Jaycam Ethernet hardware were as described below:

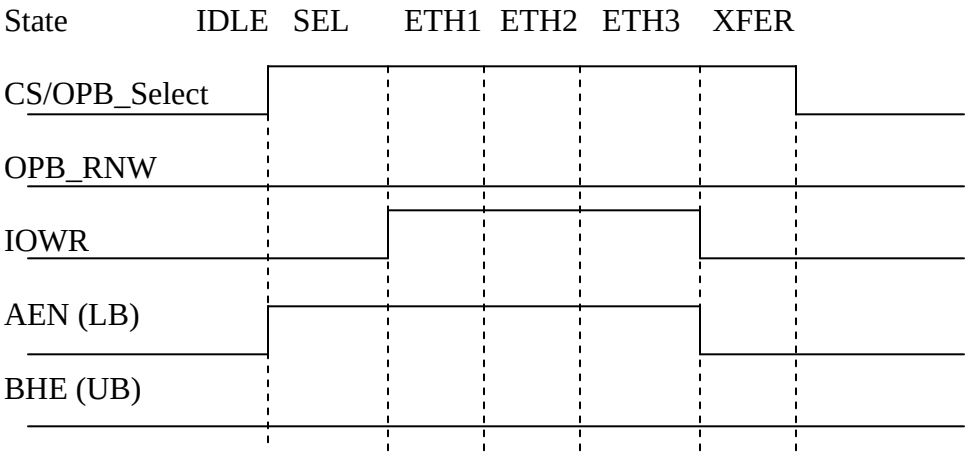

Figure 2.1 Ethernet Write cycle timing

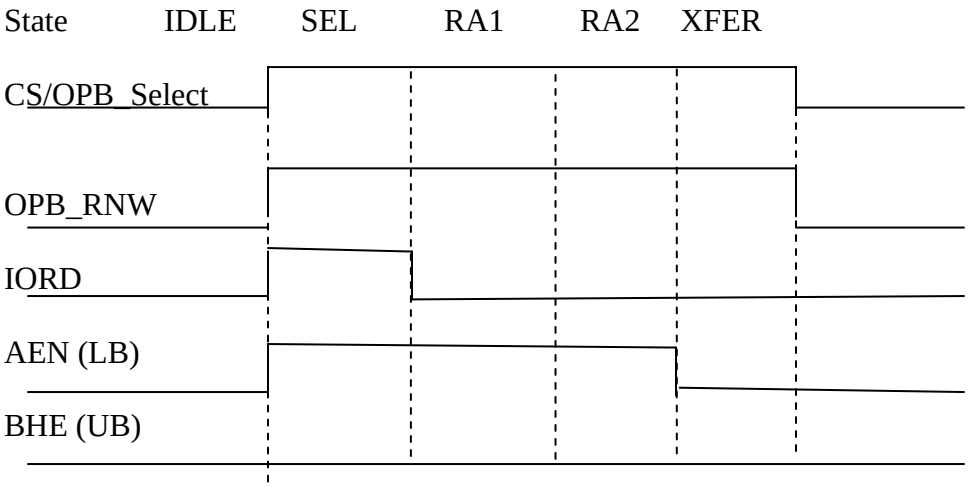

Figure 2.2 Ethernet read cycle timing

The timing of ISA bus accesses as provided in the documentation has also been pasted in figure 2.3 for reference.

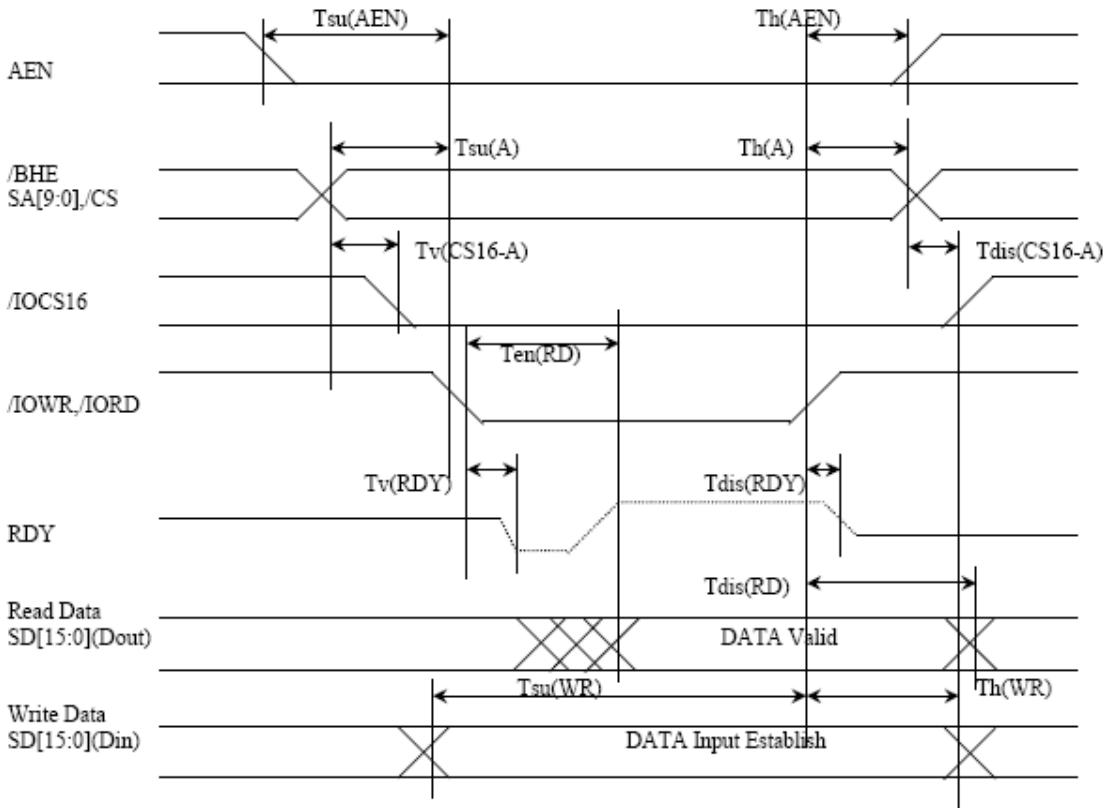

Figure 2.3 Ethernet ISA bus access timing. Source: ASIX Electronics Corporation.

2.2 Ethernet Driver

The Ethernet was initialized using the C drivers that we obtained from the projects completed in 2004 and 2005. We were able to recover the Ethernet driver from the jaycam project and use it to send and receive Ethernet packets. The jaycam project only contained the etherSend.c capability, however, since we really needed to receive packets rather than send them, we had to draw on other projects completed in 2004 such as the ICC project to implement this functionality.

The driver composed of the following file structure:

2.2.1 main.c

This file includes the `main()` function used to run the program.

2.2.2 etherReceive.c

etherReceive.c contains critical functions used to read packets to the MAC layer

2.2.3 etherSend.c

etherSend.c contains critical functions used to write packets to the MAC layer

2.2.4 etherFunc.c

This is critical to the testing of the ethernet controller. It contains `diag()` that writes certain registers on the ethernet controller and then reads them. It, thus, verifies if the correct values were written to the registers and if the ethernet controller changes its state according to the set values. This file also contains initialization functions such as `init_ethernet()` and `resetnic()` used to reset the ethernet controller.

2.2.5 Xilnet library

These Microblaze EDK version 7.1 contained the new XilNet version 2.0.0 library to configure the TCP/IP stack on the microblaze. We used this stack to send and receive packets via the ethernet controller.

src/api/xilsock.c
src/ipv4/tcp.c
src/ipv4/ip.c
ipv4/icmp.c
src/ipv4/udp.c
src/net/eth.c
src/net/mac.c
src/net/arp.c
src/net/xilnet_config.c

The C source files used to implement the TCP and UDP protocols were borrowed directly from the XilNet library and more information regarding them can be obtained from the source documentation that can be found at the following location on all iLab computers:

```
/usr/cad/xilinx/edk7.1i/sw/lib/sw_services/xilnet_v2_00_a/doc/xilnet_v2_00_a.pdf
```

Include files
ether_reg.h
/src/include/net/*.h

2.3 Ethernet Transmission

We used the `send()` and `receive()` functions in etherSend.c and etherReceive.c respectively to augment the eth.c write and read functions respectively. These `send()` and `receive()` functions are the MAC layer write and read functions that enable data to the transferred to and from the ethernet buffer by instituting DMA writes and reads. The `XEmac_SendFrame()` and `XEmac_RecvFrame()` functions implemented in etherFunc.c refer to the `send()` and `receive()` functions to link the XilNet library to the device specific MAC layer IO functions.

3. JPEG Decompression

3.1 Overview

One of the other tasks for project was to do decompress the MPEG (motion JPEG images) which was coming from our internet camera, INTELLINET. These images would feed into the FPGA via an Ethernet cable and would be written on the SRAM on the FPGA. These images would be then taken and displayed on the screen via the video out option of the FPGA.

We started our JPEG decompression phase by downloading the DJPEG software package from the link provided in the project report from the 2005 project, digital picture frame. The digital picture frame project group's idea was to feed the JPEG images via a serial cable on to the FPGA and then decompress the JPEG images and display them on the screen, but they were unable to cope up with the challenge posed by the DJPEG code and were not able to manipulate the software and make it compile with the Microblaze. Thus, they ended up doing their project on software.

Cristian Soviani, the TA for the class, told us to start hacking DJPEG software and at the same time look for some other software on the Web because the DJPEG was very verbose software and it would take us more time than what we had in our hands. Using the Internet, we found two different programs for doing JPEG decompression. One of them was by some students from Cornell University and one was from students at Stanford University. One of them was too big in size to fit on the SRAM, and the other one only dealt with lossy JPEG images, while we wanted something that dealt with both lossy and lossless JPEG images.

Professor Stephen Edwards was our savior at this point and he directed us to this other JPEG decompression program, which was much better than the others we came across. The JPEG decompression took us around three weeks to accomplish. We were able to display the JPEG image on the screen.

The main idea behind JPEG Decompression and our methods to tackle the challenge is mentioned below:

3.2 JPEG File Structure

The Jpg file is composed of pieces called "segments". A segment is a stream of bytes with `length <= 65535` and its beginning is specified with a marker. A marker = 2 bytes beginning with 0xFF, and ending with a byte from `0` to `0xFF`. Before entering the while loop, the program searches for the SOI (SOI = Start of Image = 'FFD8') which mark the beginning of the file. A brief description of the markers used in these programs:

SOF_MK:	Start of Frame 0	FFC0
DHT_MK:	Define Huffman Table	FFC4

DQT_MK:	Define Quantization Table	FFDB
DRI_MK:	Define Restart Interval	FFDD
SOS_MK:	Start Of Scan	FFDA
EOI_MK:	End of Image	FFD9
COM_MK:	Comment	FFFE
EOF:	End of File	

The default case is the when the segment are either unimportant as the program does not care about it or when it wants to purposely skip a segment.

A typical flow of the program is as follows:

SOI
APP < ...>
DQT <quantisation table 1>
DQT <quantisation table 2>
SOF <image info, size …>
DHT <huffmann table 1>
DHT <huffmann table 2>
DHT <huffmann table 3>
DHT <huffmann table 4>
SOS <encoded data>

This essentially translates into the following flow diagram.

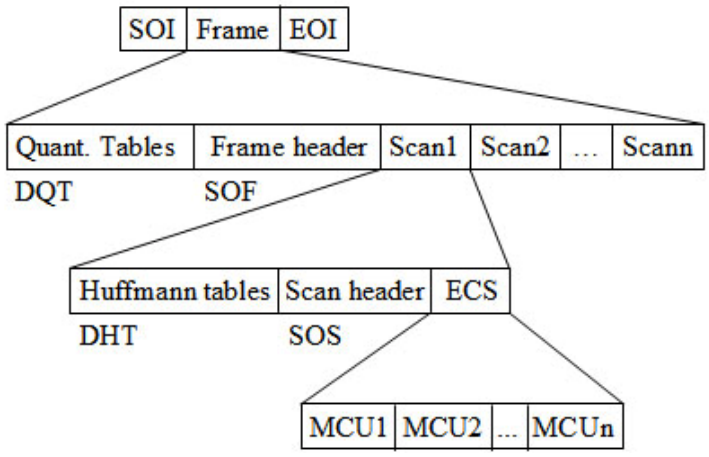

Figure 3.1 Flow diagram for the jpeg decompression program from SOI to EOI
Source: http://www.es.ele.tue.nl/~jef/education/5kk10/

As illustrated, the program starts off at SOI in jpeg.c and goes into the procedures scan1, which further breaks it down into Huffman tables and so on.

3.3 Algorithm Description

The code uses the Huffman Algorithm that uses inputs and outputs as shown in figure 3.2 below and creates a weighted tree to provide unambiguous binary encoding based on color frequency.

Input = table of weighted sub-trees
(Initially leaves/symbols)
Output = "Huffman Code Tree"

1. Sort the table by weight
2. Pair up the two least weight entries
3. Insert the new tree at the right position
4. Goto 2 until table is a single tree

Example:

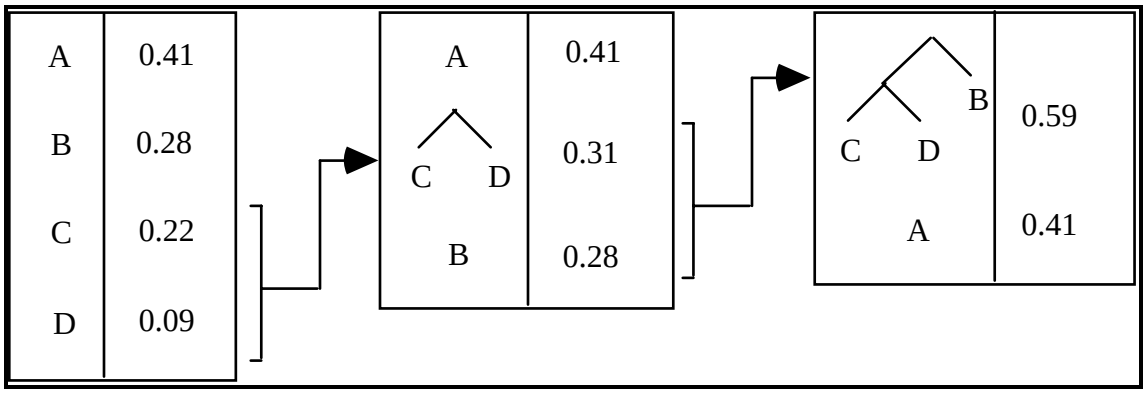

Figure 3.2 Huffman tree creation from JPEG header.
Source: http://www.es.ele.tue.nl/~jef/education/5kk10/

The jpeg decoder is written by Pierre Guerrier and later modified by Koen van Eijk.

3.3.1 Brief Overview of decoding a JPG file

In JPG, the common RGB color scheme is transformed into Y, Cb and Cr. The decoder reads from the JPG file the sampling factors that are specified for each of the image component [Y, Cb, Cr] for both horizontal and vertical sampling. It then calculates the dimensions of the MCU (Minimum Coded Unit) as well as how many MCUs are in the whole image. It then decodes the MCU in a loop for all these blocks until an EOI (End of Image) marker is reached. The decoder decodes every Data Unit (8x8 pixel block for each of the 3 image components, therefore a vector of 64) in the order as follows:

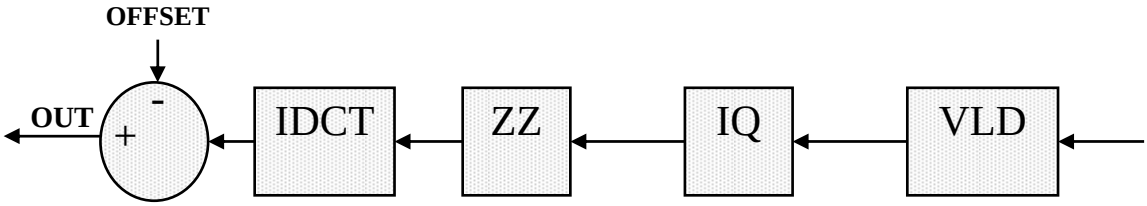

Figure 3.3 JPEG decoding algorithm
Source: http://www.es.ele.tue.nl/~jef/education/5kk10/

1. Huffman Decoder
2. Inverse quantize the 64 vector:
   `for (i=0;i<=63;i++) vector[i]*=quant[i]`
3. Re-order from zig-zag (ZZ) the 64 vector into an 8x8 block
4. Apply the Inverse DCT transform to the 8x8 block
5. Level shift samples (add 128 to the all 8-bit signed values in the 8x8 blocks
6. Transform YCbCr to RGB

The decoded `(Hmax*8 x Vmax*8)` truecolor pixel block is written into the (R,G,B) image buffer.

3.4 Requirements of the C code

The JPEG program originally made by Koen van Eijk and Pierre Guerrier was a standalone Jpeg decoder for UNIX. It came as no surprise that it takes a jpeg binary file, *.jpg, as an input and dumps a Sun Raster format file with *.ras extension after it has finished executing the program.

One main challenge was to follow the considerations below while we implemented the JPEG algorithm on the highly constrained environment of the Microblaze.

1. To modify the program code so it doesn't take in a binary file as an input but instead takes in jpeg data directly from streaming ethernet packets.
2. Remove all C stdio.h or other library functions that will work for gcc but not for microblaze-gcc.
3. Dynamic memory allocations using the heap memory space (`malloc()`) are used extensively in the program. As described in lecture 2 in class, because of danger of exhausting memory which is also critical as we only have limited 512K in the memory should be avoided if possible. All `malloc()` uses are replaced using pointers and static memory allocations.

3.5 Source Code Overview

The C program is structured as follows:

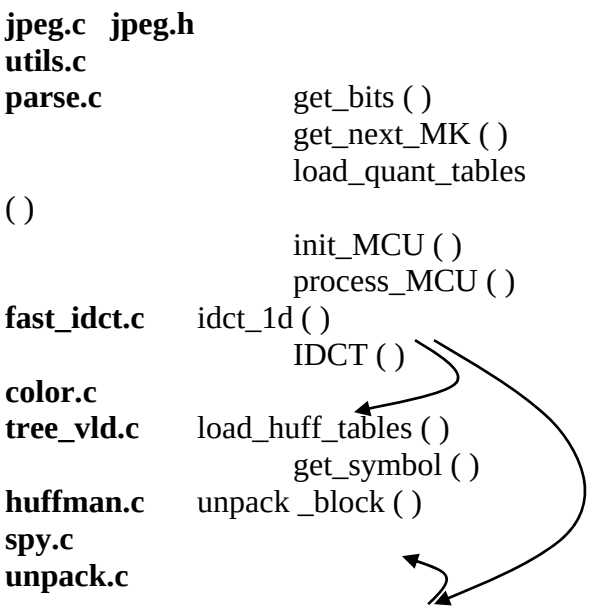

Figure 3.4 JPEG algorithm C source code structure
Source: http://www.es.ele.tue.nl/~jef/education/5kk10/

The following section contains a detailed explanation of the important functions and key considerations that went into editing each of the original JPEG C source files for use with the microblaze.

3.5.1 jpeg.c

This file contains the main driver for the program.

The variable `fi` and `fo` are used extensively in the program for direct access of byte information:

```
FILE   *fi;              /* input File stream pointer     */
FILE    *fo;             /* output File stream pointer   */
```

Instead of taking `fi` as the input, we declared another variable of type unsigned character:

```
unsigned char vfile[];
```

The array is defined using data taken from a small jpeg image during the testing phase:

```
unsigned char vfile[] = {255, 216, 255, 224, 0, 16, 74, 70, 73, 70, 0, 1, 1, 0, 0 …… };
```

Two other important variables are also defined:

```
int  vfile_size;
unsigned char *vfile_ptr;     /* Our NEW Input File stream Pointer*/
```

`vfile_size` stores the size of the array.
`vfile_ptr` keep tracks of the reading position as was originally taken care of while using standard I/O library functions.

The program then goes into a while loop which contain a large case statement.

3.5.2 parse.c

This contains the main driver for the program. Below you will find a brief scheme of decoding a JPG file.

The decoder reads the sampling factors from the JPG file. The JPEG file is composed of units known as MCUs or Minimum Coded Units. It finds out the dimensions of a single `MCU (Hmax*8,Vmax*8)`. It then finds out the total number of MCUs are in the whole image. Following this, the program decodes every `MCU` present in the original image (a loop for all these blocks, or until the EOI marker is found [it should be found when the loop finishes, otherwise you'll get an incomplete image]). It decodes an MCU by decoding every “Data Unit” in the `MCU` in the order mentioned before, and finally, writes the decoded `(Hmax*8 x Vmax*8)` truecolor pixel block into the (R,G,B) image buffer.

3.6 Memory Use

The data which will be received from the CCD camera is estimated to be around 384 kb in size. This data will be stored temporally on the VGA buffer segment of the SRAM and then it will be transferred to the screen. The VGA buffer on the SRAM is mapped from address 0x0080000 to 0x0086000.

4. State Machine

4.1 Overview

According to the project description, our state machine is supposed to read and write from the SRAM and from the Ethernet chip and display the JPEG images on the VGA screen buffer. In this section, we will provide an introduction of the approach to writing the opb_xsb bridge component and lay the ground work that we will use to point out the differences between the three different state machines in sections 4.2, 4.3 and 4.4.

As shown in figure 1.2, the OPB arbiter bridge is responsible for allocating the slave bus to requests from the microblaze and the video. The code for opb_xsb300.vhd that lies at the center of this hardware description is located in section 6.1.1. It relies on the components pad_io.vhd, memoryctrl.vhd, vga_timing.vhd and vga.vhd. pad_io.vhd is used to buffer the signals generated by the state machine located in memoryctrl.vhd.

4.1.1 State Machine Control Signals

The memoryctrl.vhd accepts requests from the video out known as `vreq`, RAM I/O requests from the microblaze characterized by the signal `(cs and !eth_io)` and Ethernet requests (if present) are denoted by the signal `eth_io`.

4.1.2 Control Signal Generation

These signals are generated by using the following rules in all three state machines.

```
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';
```

`cs` is set to 1 whenever either sram or ethernet are requested on the OPB. `sram_ce` and `ethernet_ce` are set if the address bits corresponding to the sram and ethernet are set to 00 an 01 respectively. These bits are bits 22 and 21 of the 32-bit OPB_address. `eth_io` is also set to 1 when these address bits are set to 01.

Some other important control signals in opb_xsb300.vhd are:

`onecycle` and `fullword`: when `fullword` is set to 1, it means that the microblaze is requesting a 32-bit access from the sram. These signals are the inverse of each other, i.e. `onecycle` = !`fullword`. Depending on whether `fullword` is set (or `onecycle` is not set), the state machine in memoryctrl.vhd switches to different states during a read or write cycle thus providing more time for 32 bits to be read or written. While `onecycle` is used in two of the three state machines

`rce0` and `rce1`: these signals map on to `ce0` and `ce1` in memoryctrl.vhd. They inform the pad_io as to how many bytes need to be read or written. `rce0` refers to the upper bytes (31…16) whereas `rce1` refers to the lower bytes (15…0). For instance if only `rce0` or `rce1` is set to 1 that implies that it's a 16-bit IO request and only the bits 15…0 are written. If however, `rce1` is also set then the remaining 16-bits are also written.

4.2 SRAM-Video State Machine

The SRAM state machine was obtained from state machine in lab 5 2004. It is very robustly designed which is why we chose to work with this. Please refer to figure 4.2. This state machine only handled video and sram and has thus formed the basis of our jpeg decoder sub-project. We always begin in the idle state. As soon as the sram is selected upon receiving an OPB_Select and an sram address from the microblaze, the Common state is entered. We then check whether `OPB_RNW` was 1 or 0. If it is set to 1, then we proceed to the RA1 state. For 16-bit reads, we would proceed immediately to RA2 and then end at the XFER state. However, for 32-bit reads, we move directly to the RB1 state from the COMMON state and proceed along RB2 and RB3 to end at the XFER state. If RNW is set to 0, we move on to the transfer state for 16-bit writes, i.e. `onecycle` = 1 (or `fullword`= 0) and to the W state for 32-bit writes, subsequently entering the XFER state.

Once at the XFER state, the state machine returns to the IDLE state and waits for another chip select. See opb_xsb300.vhd in section 6.4.3a and memoryctrl.vhd in section 6.4.4a.

4.3 Deploying Ethernet

Our initial attempts to attach ethernet to the state machine in section 4.2 were simplistic. We assumed that the timing of the sram read and write cycles would be close enough to the timings of the ethernet IO processes. We thus decided to just change output `ethernet_ce` = 1 instead of `sram_ce` = 1 if an ethernet address was requested.

4.4 Jaycam State Machine

This is the state machine that successfully allows us to compile the ethernet and sram together. However, it doesn't operate video reads from the sram. For read cycles we step into the RA state and continue on to the RB state automatically. If it's a 16-bit read then we move to the XFER state from RB, but for 32-bit reads we continue to the RC state before going to the XFER state. The way writes are handled to the SRAM is the same as the FSM in section 4.2. The major difference, however, is the addition of three ethernet states. If `eth_io` is set in the opb_xsb300.vhd, then we move to the WETH1 for ethernet writes only. For ethernet reads we use the sram read states RA and RB. From WETH1 we proceed to WETH2 and then WETH3, which acts merely as delay states. From WETH3 we move to the XFER state. From the XFER state, as always we enter the IDLE state and wait for another `CS`. See corresponding opb_xsb300.vhd in section 6.4.3b and memoryctrl.vhd in section 6.4.4b.

4.5 Ethernet Priority State Machine

We started creating the bridge for the SPYCAM project by understanding the state machine described in section 4.2. The bridge from the lab 5 controlled the communication between the SRAM and VGA screen buffer. The bridge for our project needs to communicate between the SRAM, the VGA screen buffer and the Ethernet. If this state machine would have worked correctly, our deliverables would have included the complete Spycam project. However, since this state machine failed to provide desired the datapath between the three peripherals, our deliverable does not include ethernet send and receive functionality.

After hacking the lab5 for several days, we came up with our own state machine which would control the communication to the Ethernet. In our state machine, 3 states are assigned for each, the Ethernet read and the Ethernet Write. This can be seen in the Figure 4.3. See corresponding opb_xsb300.vhd in section 6.4.3c and memoryctrl.vhd in section 6.4.4c.

We start from the IDLE state and if `CS` = 1 or `OPB_Select` = 1, we move to the common state. In the common state we control communication to the Ethernet, SRAM and Video. If there is a 32-bit write request, the state machine transfers onto the R_W state from Common and then from there it will go to the XFER state as in 4.2. After XFER, the state

machine always will go back to the IDLE state and then depending on the values of the CS and OPB_Select it will go to the Common state and handle the further requests.

RNW is the controls the operation of read and write. When there is an Ethernet request, eth_io = 1 and then state machine then see's what is the value of the RNW signal, if RNW = 1, it implies that the state machine will make a transition to the RETH state, which is the Ethernet read state, and if RNW = 0, it means there is an Ethernet write request and the state machine will go to the WETH state. In both read and write states for the Ethernet, we provide two buffer states to insure that a proper data is read and written respectively.

These two buffer states for the Ethernet read are RETH2 and REHT3, and the two buffer states for the Ethernet write are WETH2 and WEHT3. When the state machine is in the RETH1 State, if OPB_Select = 1, then we move to the RETH2 state and again, if OPB_Select = 1, then we move to RETH3 from R_RETH2. From RETH3, there is a transition to the XFER state which implies that the state machine has finished its routine request of the Ethernet and it will go back to the IDLE state.

Similarly when in the state WETH1, if OPB_Select =1 then we will go to WETH2 state and from there to WETH3. Once the state machines leaves the WETH3 state, it goes to XFER and from there back to IDLE and then to COMMON and it checks for which peripheral is machine a request.

5. Miscellaneous

5.1 Approach to this documentation

Even though only the JPEG decompression aspect of this program worked correctly, we are happy to be able to provide a wealth of information about the ethernet controller that we learned during our toil through this project. The purpose of this writing is to help future groups understand how to navigate through the ethernet and not make the same mistakes we made. This is why we have drawn and explained the two state machines that we couldn't use for our purposes in addition to the one that we ended up using. We have also explained why the other two state machines were unsuitable. Moreover, we hope this documentation will help a future group understand and improve the JPEG decompression and the Ethernet peripheral implementation.

5.2 Challenges Faced

The state machine was the biggest challenge in our project and it took us more than three weeks to understand how it actually works. We started the hardware implementation of our project by hacking previous year's projects that were using the SRAM and Ethernet chip. According to Professor Edward's advice we started our process by understanding the state machine used by the JAYCAM, a project from Spring 2004. The JAYCAM group were taking analog video images, then converting them into digital images and storing them on memory and broadcasting them using the Ethernet chip. The state

machine used by the JAYCAM group is as shown below in Figure 4.1. *"FULLWORD"* was the most confusing signal that is used in the JAYCAM state machine, it took us almost 2-3 days to figure out what was the signals implication in the state machine. It actually implies a 32 nit access of a address.

Cristian Soviani, the TA for the Embedded Design class, advised us against using the JAYCAM project and asked to try to understand and modify the LAB5 Homework assignment's state machine for the year 2004. The Lab 5 handled the SRAM and video part but lacked the Ethernet read and write. The state machine for lab 5 is shown below in Figure 4.2. In the lab 5 state machine, the signal `HalfCycle != Fullword`. A simple modification was to set the Ethernet chip enable instead of the SRAM chip enable when an Ethernet address was requested by the microblaze.

After several hours spent on hacking the LAB5 code, we made a revision to the state machine by adding 3 states for the Ethernet read and 3 states for the Ethernet writes. Cristian approved the state machine and asked us to go ahead and implement it. The state machine is as showed in Figure 4.2. On compiling the state machine, we learnt that the state machine was not able to handle the Ethernet control signals as it was supposed to and yielded errors. The state machine that we came up with is as shown in Figure 4.3. We employed some signals which were different from Lab 5 such as bus_request.

Towards the end of the semester, 5 days before the deadline to submit the project, Cristian teamed us, (SPYCAM team) with the POPI team because both the groups were using the SRAM and Ethernet in hardware. We created a new state machine. We simulated the data path for which we found very agreeable results in "VSIM", where all the 8-bit, 16-bit and 32-bit reads and writes from the microblaze were proceeding as expected. The Ethernet controller's registers were also reading as expected. However, it failed to run the diagnostics correctly, even though we were able to see one register respond correctly. In order to debug this issue, we made the Ethernet timing even more conservative by adding 2 more state to the ethernet reads and writes. We used a `while` loop to write a constant value to one of the 16-bit registers to diagnose the ethernet function. The oscilloscope still didn't show the chip enables asserted at the same as the ethernet chip select was 0 (active). In addition to the challenge faced by the two groups, the SPYCAM team had the additional challenge of making sure that video reads took place in the idle state.

At this point, Professor Edwards advised us to hack the state machine of a Project from the year 2005, the Internet radio broadcaster. The internet radio broadcaster groups were capable to manipulate the JAYCAM code.

5.3 Lessons Learned

We learned that the documentation is shaky at best and does not provide clear cohesive instructions to students who want to implement it into workable C or VHDL code. Our experience with the Ethernet documentation tells us that even though we followed the timing diagram in figure 2.3 to the letter, and reviewed the corresponding state machine

that we generated several times, it didn’t read or write the registers correctly and when it did, the SRAM would not behave correctly which was important for us to test such a complicated and large program.

One important piece of advice to future groups would be to read as many reports written by previous groups as possible as they contain insight that only comes as one nears the end of the semester and when there isn’t enough time to fix the problem. Working with other groups who are trying to implement the same peripheral can help weed out potential bugs and get a better understanding of the logic behind a particular operation.

5.4 Special Accolade

Our special thanks go to Cristian Soviani. This man stayed in the labs way past the hours he was supposed to and tried to help everyone, even though time was always a constraint. Besides his amazing attitude and the ability to laugh at mistakes even after hours in the lab, he is brilliant and well versed with embedded systems.

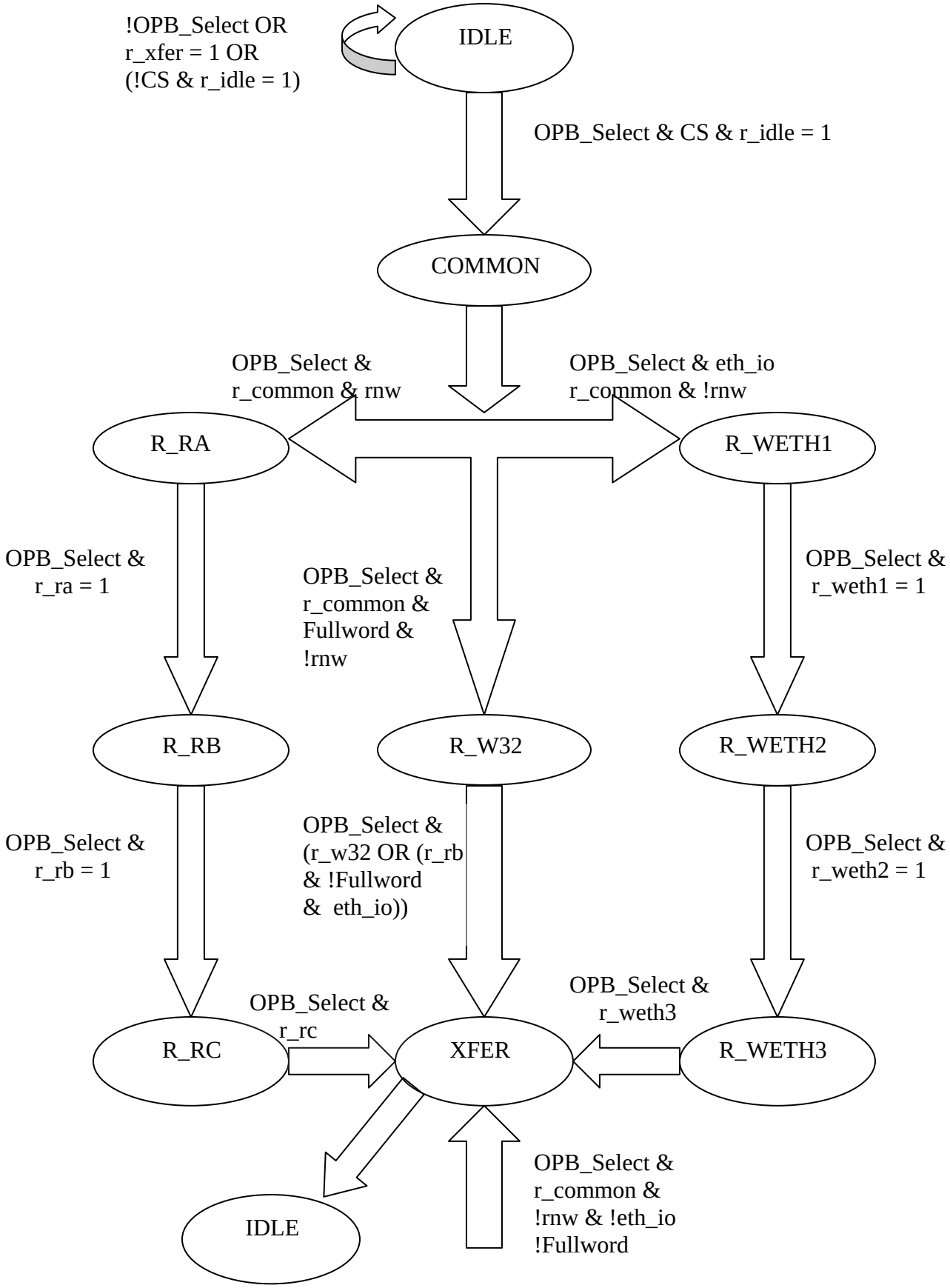
!OPB_Select OR
r_xfer = 1 OR
(!CS & r_idle = 1)
IDLE
OPB_Select & CS & r_idle = 1
COMMON
OPB_Select &
r_common & rnw
OPB_Select & eth_io
r_common & !rnw
R_RA
R_WETH1
OPB_Select &
r_ra = 1
OPB_Select &
r_common &
Fullword &
!rnw
OPB_Select &
r_weth1 = 1
R_RB
R_W32
R_WETH2
OPB_Select &
r_rb = 1
OPB_Select &
(r_w32 OR (r_rb
& !Fullword
& eth_io))
OPB_Select &
r_weth2 = 1
OPB_Select &
r_rc
OPB_Select &
r_weth3
R_RC
XFER
R_WETH3
IDLE
OPB_Select &
r_common &
!rnw & !eth_io
!Fullword

Figure 4.2

t.

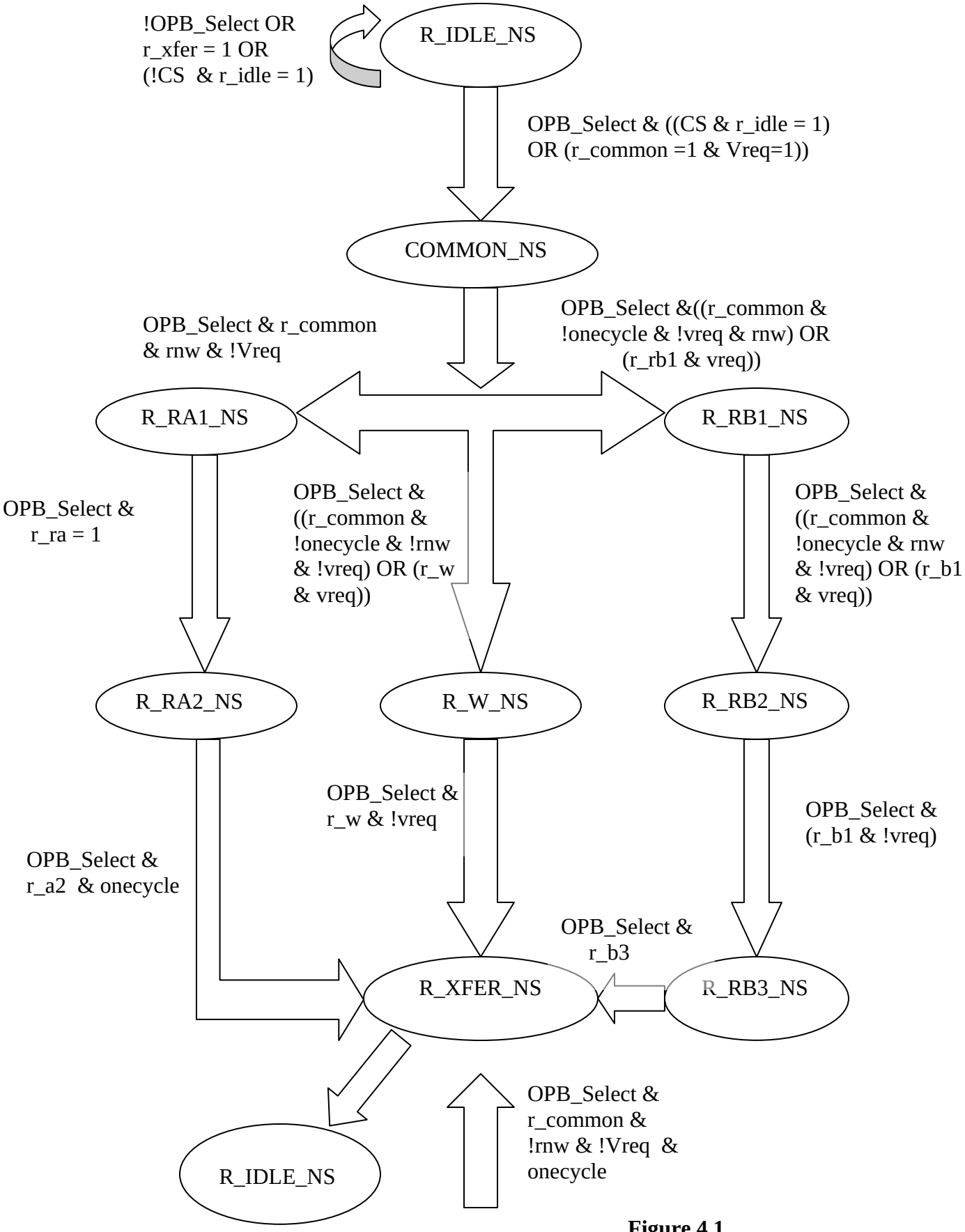
!OPB_Select OR
r_xfer = 1 OR
(!CS & r_idle = 1)
R_IDLE_NS
OPB_Select & ((CS & r_idle = 1)
OR (r_common =1 & Vreq=1))
COMMON_NS
OPB_Select & r_common
& rnw & !Vreq
OPB_Select &((r_common &
!onecycle & !vreq & rnw) OR
(r_rb1 & vreq))
R_RA1_NS
R_RB1_NS
OPB_Select &
r_ra = 1
OPB_Select &
((r_common &
!onecycle & !rnw
& !vreq) OR (r_w
& vreq))
OPB_Select &
((r_common &
!onecycle & rnw
& !vreq) OR (r_b1
& vreq))
R_RA2_NS
R_W_NS
R_RB2_NS
OPB_Select &
r_w & !vreq
OPB_Select &
(r_b1 & !vreq)
OPB_Select &
r_a2 & onecycle
OPB_Select &
r_b3
R_XFER_NS
R_RB3_NS
OPB_Select &
r_common &
!rnw & !Vreq &
onecycle
R_IDLE_NS

Figure 4.1

&

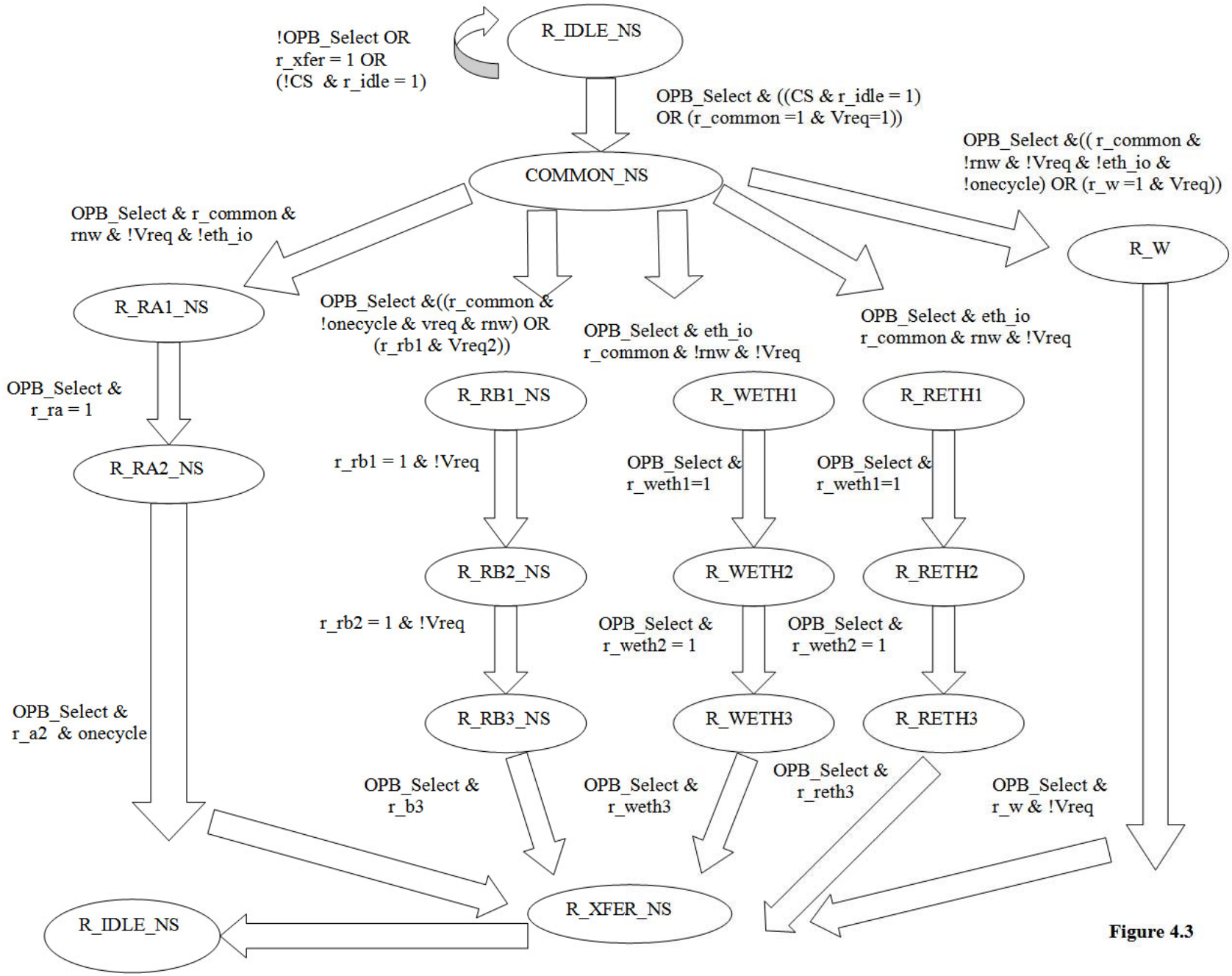
R_IDLE_NS
!OPB_Select OR
r_xfer = 1 OR
(!CS & r_idle = 1)
OPB_Select & ((CS & r_idle = 1)
OR (r_common =1 & Vreq=1))
COMMON_NS
OPB_Select &((r_common &
!rnw & !Vreq & !eth_io &
!onecycle) OR (r_w =1 & Vreq))
R_W
OPB_Select & r_common &
rnw & !Vreq & !eth_io
R_RA1_NS
OPB_Select &((r_common &
!onecycle & vreq & rnw) OR
(r_rb1 & Vreq2))
OPB_Select & eth_io
r_common & !rnw & !Vreq
OPB_Select & eth_io
r_common & rnw & !Vreq
OPB_Select &
r_ra = 1
R_RA2_NS
R_RB1_NS
R_WETH1
R_RETH1
r_rb1 = 1 & !Vreq
OPB_Select &
r_weth1=1
OPB_Select &
r_weth1=1
R_RB2_NS
R_WETH2
R_RETH2
r_rb2 = 1 & !Vreq
OPB_Select &
r_weth2 = 1
OPB_Select &
r_weth2 = 1
OPB_Select &
r_a2 & onecycle
R_RB3_NS
R_WETH3
R_RETH3
OPB_Select &
r_b3
OPB_Select &
r_weth3
OPB_Select &
r_reth3
OPB_Select &
r_w & !Vreq
R_XFER_NS
R_IDLE_NS

Figure 4.3

6. Source Code

6.1 system.mhs

```
# Parameters
PARAMETER VERSION = 2.1.0

# Global Ports

PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN

# What needs to be done here???
# Need the SRAM peripheral
# Need the Ethernet peripheral
# Need the Video Out Peripheral

# Slave Bus
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

# SRAM
PORT RAM_CE_N = RAM_CE_N, DIR = OUT

# Ethernet Ports
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

# Video Out
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]

# OPB Bridge

BEGIN opb_xsb300
 PARAMETER INSTANCE = xsb300
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00800000
 PARAMETER C_HIGHADDR = 0x00FFFFFF
# to talk the microblaze
 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 OPB_Clk = sys_clk
# to talk to the SRAM
 PORT RAM_CE_N = RAM_CE_N
# to talk to the ETHERNET
 PORT ETHERNET_CS_N = ETHERNET_CS_N
 PORT ETHERNET_RDY = ETHERNET_RDY
 PORT ETHERNET_IREQ = ETHERNET_IREQ
 PORT ETHERNET_IOCS16_N = ETHERNET_IOCS16_N
# to talk to the VIDEO DECODER
 PORT pixel_clock = pixel_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
```

```
# BRAM example peripheral

BEGIN opb_bram
 PARAMETER INSTANCE = bram_peripheral
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xFEFF1000
 PARAMETER C_HIGHADDR = 0xFEFF17ff
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
END

# The actual block memory

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

# The main processor core

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

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

# Data LMB bus

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

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_data_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE SLMB = d_lmb
 BUS_INTERFACE BRAM_PORT = conn_0
END

# Instruction LMB bus

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

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_instruction_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE SLMB = i_lmb
 BUS_INTERFACE BRAM_PORT = conn_1
END

# Clock divider to make the whole thing run
```

```
BEGIN clkgen
 PARAMETER INSTANCE = clkgen_0
 PARAMETER HW_VER = 1.00.a
 PORT FPGA_CLK1 = FPGA_CLK1
 PORT sys_clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT fpga_reset = fpga_reset
END

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

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

# UART: Serial port hardware

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

# Interrupt Control

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
```

6.2 system.mss

```
 PARAMETER VERSION = 2.2.0
 PARAMETER HW_SPEC_FILE = system.mhs


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


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


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

# Use null drivers for peripherals that don't need them
# This supresses warnings
BEGIN DRIVER
 PARAMETER HW_INSTANCE = bram_peripheral
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

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

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

6.3 system.ucf

```
net sys_clk period = 18.000;
net pixel_clock period = 36.000;

net FPGA_CLK1 loc="p77";

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

# SLave address
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";

# Slave Data (16 bit)
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

# SLAVE BUS
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";

#Chip Select for SRAM
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";

# Screen buffer
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";
```

6.4 OPB Arbiter VHDL Source Code

6.4.1 opb_xsb300_v2_1_0.pao

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


lib opb_xsb300_v1_00_a opb_xsb300
lib opb_xsb300_v1_00_a memoryctrl
lib opb_xsb300_v1_00_a vga
lib opb_xsb300_v1_00_a vga_timing
lib opb_xsb300_v1_00_a pad_io
```

6.4.2 opb_xsb300_v2_1_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

OPTION 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 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
```

6.4.3a opb_xsb300.vhd (Section 4.2)

```
------------------------------------------------------------------------------
--
-- OPB bus bridge for the XESS XSB-300E board
--
-- Includes a memory controller, a VGA framebuffer, and glue for the SRAM
--
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.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;
    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;
    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

  constant C_MASK : integer := 0; -- huge address window as we are a bridge

  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 onecycle : std_logic ;
  signal videocycle, amuxsel, hihalf : std_logic;
  signal rce0, rce1, rreset : std_logic;
  signal xfer : std_logic;
  signal pb_wr, pb_rd : std_logic;

  signal sram_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 vga
    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;
      select0 : in std_logic;
      rnw : in std_logic;
      vreq : in std_logic;
      onecycle : in std_logic;
      videocycle : out std_logic;
      hihalf : out std_logic;
      pb_wr : out std_logic;
      pb_rd : out std_logic;
      xfer : out std_logic;
      ce0 : out std_logic;
      ce1 : out std_logic;
      rres : out std_logic;
      video_ce : out std_logic);
  end component;

  component pad_io
    port (
      clk : 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;
      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;
      pb_dread : out std_logic_vector(15 downto 0);
      pb_dwrite : in std_logic_vector(15 downto 0));
  end component;

begin

  -- Framebuffer

  vga1 : vga
    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);

-- Memory control/arbitration state machine

  memoryctrl1 : memoryctrl port map (
    rst => OPB_Rst,
    clk => OPB_Clk,
    cs => cs,
    select0 => OPB_select,
    rnw => rnw,
    vreq => video_req,
    onecycle => onecycle,
    videocycle => videocycle,
    hihalf => hihalf,
    pb_wr => pb_wr,
    pb_rd => pb_rd,
    xfer => xfer,
    ce0 => rce0,
    ce1 => rce1,
    rres => rreset,
    video_ce => video_ce);

-- I/O pads

  pad_io1 : pad_io port map (
    clk => OPB_Clk,
    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,
    PB_D => PB_D,
    pb_addr => addr_mux,
    pb_rd => pb_rd,
    pb_wr => pb_wr,
    pb_ub => pb_bytesel(1),
    pb_lb => pb_bytesel(0),
    ram_ce => sram_ce,
    pb_dread  => rdata,
    pb_dwrite => wdata_mux);

  sram_ce <= pb_rd or pb_wr;

  amuxsel <= videocycle;

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

  onecycle <= (not be(3)) or (not be(2)) or (not be(1)) or (not be(0));

  wdata_mux <= wdata(15 downto 0) when ((addr(1) or hihalf) = '1')
```

```
              else wdata(31 downto 16);

process(videocycle, be, addr(1), hihalf, pb_rd, pb_wr)
begin
  if videocycle = '1' then
    pb_bytesel <= "11";
  elsif pb_rd='1' or pb_wr='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;

cs <= OPB_select when OPB_ABus(31 downto 20) = X"008" else '0';

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;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_RST = '1' then
      addr <= X"000000";
    else
      addr <= OPB_ABus(23 downto 0);
    end if;
  end if;
end process;

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;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_Rst = '1' then
      wdata <= X"00000000";
    else
      wdata <= OPB_DBus;
    end if;
  end if;
end process;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if video_ce = '1' then
      video_data <= rdata;
    end if;
  end if;
end process;

-- Write the low two bytes if rce0 or rce1 is enabled

process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    UIO_DBus(15 downto 0) <= X"0000";
  elsif OPB_Clk'event and OPB_Clk = '1' then
    if rreset = '1' then
      UIO_DBus(15 downto 0) <= X"0000";
```

```
      elsif (rce1 or rce0) = '1' then
        UIO_DBus(15 downto 0) <= rdata(15 downto 0);
      end if;
    end if;
  end process;

  -- Write the high two bytes if rce0 is enabled

  process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      UIO_DBus(31 downto 16) <= X"0000";
    elsif OPB_Clk'event and OPB_Clk = '1' then
      if rreset = '1' then
        UIO_DBus(31 downto 16) <= X"0000";
      elsif rce0 = '1' then
        UIO_DBus(31 downto 16) <= rdata(15 downto 0);
      end if;
    end if;
  end process;

  -- unused outputs

  UIO_errAck <= '0';
  UIO_retry <= '0';
  UIO_toutSup <= '0';

  UIO_xferAck <= xfer;

end Behavioral;
```

6.4.3b opb_xsb300.vhd (Section 4.4)

```
------------------------------------------------------------------------------
--
-- OPB bus bridge for the XESS XSB-300E board
--
-- Includes a memory controller, a VGA framebuffer, and glue for the SRAM
--
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.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;
    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;
        --Adding ethernet ports
    ETHERNET_CS_N : out std_logic;
    ETHERNET_RDY : in std_logic;
    ETHERNET_IREQ : in std_logic;
```

```
    ETHERNET_IOCS16_N : in std_logic;
    --end ethernet ports
    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

  constant C_MASK : integer := 0; -- huge address window as we are a bridge

  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 onecycle : std_logic ;
  signal videocycle, amuxsel, hihalf : std_logic;
  signal rce0, rce1, rreset : std_logic;
  signal xfer : std_logic;
  signal pb_wr, pb_rd : std_logic;
  signal ethernet_ce: std_logic;
  signal bus_req: std_logic;

  signal sram_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 vga
    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;
      select0 : in std_logic;
      rnw : in std_logic;
      vreq : in std_logic;
      onecycle : in std_logic;
      videocycle : out std_logic;
      hihalf : out std_logic;
      pb_wr : out std_logic;
      pb_rd : out std_logic;
      xfer : out std_logic;
      ce0 : out std_logic;
      ce1 : out std_logic;
      rres : out std_logic;
```

```
      video_ce : out std_logic
      -- ethernet signals added
      eth_io : in std_logic;
      --bus_req : out std_logic
      --end ethernet signals
      );
  end component;

  component pad_io
    port (
      clk : 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;
      --begin ethernet
      ETHERNET_CS_N : out std_logic;
      ETHERNET_RDY : in std_logic;
      ETHERNET_IREQ : in std_logic;
      ETHERNET_IOCS16_N : in std_logic;
      --end ethernet
      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;
      --begin ethernet
      ethernet_ce : in std_logic;
      --end ethernet
      pb_dread : out std_logic_vector(15 downto 0);
      pb_dwrite : in std_logic_vector(15 downto 0));
  end component;

begin

  -- Framebuffer

  vga1 : vga
    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);

-- Memory control/arbitration state machine

  memoryctrl1 : memoryctrl port map (
    rst => OPB_Rst,
    clk => OPB_Clk,
    cs => cs,
    select0 => OPB_select,
    rnw => rnw,
    vreq => video_req,
    onecycle => onecycle,
    videocycle => videocycle,
    hihalf => hihalf,
    pb_wr => pb_wr,
    pb_rd => pb_rd,
    xfer => xfer,
    ce0 => rce0,
    ce1 => rce1,
    rres => rreset,
    video_ce => video_ce
    -- ethernet signals added
    eth_io => eth_io,
    --bus_req => bus_req
    --end ethernet signals
```

```
    );

-- I/O pads

  pad_io1 : pad_io port map (
    clk => OPB_Clk,
    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_rd => pb_rd,
    pb_wr => pb_wr,
    pb_ub => pb_bytesel(1),
    pb_lb => pb_bytesel(0),
    ram_ce => sram_ce,
    --added
    ethernet_ce => ethernet_ce,
    pb_dread  => rdata,
    pb_dwrite => wdata_mux);

    --must be changed because a read / write access may not be chip specific anymore
  --sram_ce <= pb_rd or pb_wr;

  amuxsel <= videocycle;

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

  onecycle <= (not be(3)) or (not be(2)) or (not be(1)) or (not be(0));

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

  process(videocycle, be, addr(1), hihalf, pb_rd, pb_wr)
  begin
    if videocycle = '1' then
      pb_bytesel <= "11";
    elsif pb_rd='1' or pb_wr='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;

  --Changing cs because we need cs = '1' when either ethernet or sram is needed
  --cs <= OPB_select when OPB_ABus(31 downto 20) = X"008" else '0';
  bus_req <= pb_rd or pb_wr;
  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';
-- ending changes
-- Thought process: How about not changing anything except which chip is
-- selected. If I can control whether RAM_CE_N or ETHERNET_CE_N are on or
-- off and simultaneously try to talk to them if they were the same type
-- of input output creature, then why wouldn't they respond correctly?

  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;
```

```
process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_RST = '1' then
      addr <= X"000000";
    else
      addr <= OPB_ABus(23 downto 0);
    end if;
  end if;
end process;

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;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_Rst = '1' then
      wdata <= X"00000000";
    else
      wdata <= OPB_DBus;
    end if;
  end if;
end process;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if video_ce = '1' then
      video_data <= rdata;
    end if;
  end if;
end process;

-- Write the low two bytes if rce0 or rce1 is enabled

process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    UIO_DBus(15 downto 0) <= X"0000";
  elsif OPB_Clk'event and OPB_Clk = '1' then
    if rreset = '1' then
      UIO_DBus(15 downto 0) <= X"0000";
    elsif (rce1 or rce0) = '1' then
      UIO_DBus(15 downto 0) <= rdata(15 downto 0);
    end if;
  end if;
end process;

-- Write the high two bytes if rce0 is enabled

process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    UIO_DBus(31 downto 16) <= X"0000";
  elsif OPB_Clk'event and OPB_Clk = '1' then
    if rreset = '1' then
      UIO_DBus(31 downto 16) <= X"0000";
    elsif rce0 = '1' then
      UIO_DBus(31 downto 16) <= rdata(15 downto 0);
    end if;
  end if;
end process;

-- unused outputs

UIO_errAck <= '0';
UIO_retry <= '0';
UIO_toutSup <= '0';

UIO_xferAck <= xfer;
```

```
end Behavioral;
```

6.4.3c opb_xsb300.vhd (Section 4.5 with modifications for Ethernet)

```
------------------------------------------------------------------------------
--
-- OPB bus bridge for the XESS XSB-300E board
--
-- Includes a memory controller, a VGA framebuffer, and glue for the SRAM
--
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.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;
    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;
        --Adding ethernet ports
    ETHERNET_CS_N : out std_logic;
    ETHERNET_RDY : in std_logic;
    ETHERNET_IREQ : in std_logic;
    ETHERNET_IOCS16_N : in std_logic;
    --end ethernet ports
    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

  constant C_MASK : integer := 0; -- huge address window as we are a bridge

  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 onecycle : std_logic ;
signal videocycle, amuxsel, hihalf : std_logic;
signal rce0, rce1, rreset : std_logic;
signal xfer : std_logic;
signal pb_wr, pb_rd : std_logic;
signal ethernet_ce: std_logic;
signal bus_req: std_logic;

signal sram_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 vga
  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;
    select0 : in std_logic;
    rnw : in std_logic;
    vreq : in std_logic;
    onecycle : in std_logic;
    videocycle : out std_logic;
    hihalf : out std_logic;
    pb_wr : out std_logic;
    pb_rd : out std_logic;
    xfer : out std_logic;
    ce0 : out std_logic;
    ce1 : out std_logic;
    rres : out std_logic;
    video_ce : out std_logic
    -- ethernet signals added
    --eth_io : in std_logic;
    --bus_req : out std_logic
    --end ethernet signals
    );
end component;

component pad_io
  port (
```

```
      clk : 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;
      --begin ethernet
      ETHERNET_CS_N : out std_logic;
      ETHERNET_RDY : in std_logic;
      ETHERNET_IREQ : in std_logic;
      ETHERNET_IOCS16_N : in std_logic;
      --end ethernet
      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;
      --begin ethernet
      ethernet_ce : in std_logic;
      --end ethernet
      pb_dread : out std_logic_vector(15 downto 0);
      pb_dwrite : in std_logic_vector(15 downto 0));
  end component;

begin

  -- Framebuffer

  vga1 : vga
    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);

-- Memory control/arbitration state machine

  memoryctrl1 : memoryctrl port map (
    rst => OPB_Rst,
    clk => OPB_Clk,
    cs => cs,
    select0 => OPB_select,
    rnw => rnw,
    vreq => video_req,
    onecycle => onecycle,
    videocycle => videocycle,
    hihalf => hihalf,
    pb_wr => pb_wr,
    pb_rd => pb_rd,
    xfer => xfer,
    ce0 => rce0,
    ce1 => rce1,
    rres => rreset,
    video_ce => video_ce
    -- ethernet signals added
    --eth_io => eth_io,
    --bus_req => bus_req
    --end ethernet signals
```

```
    );

-- I/O pads

  pad_io1 : pad_io port map (
    clk => OPB_Clk,
    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_rd => pb_rd,
    pb_wr => pb_wr,
    pb_ub => pb_bytesel(1),
    pb_lb => pb_bytesel(0),
    ram_ce => sram_ce,
    --added
    ethernet_ce => ethernet_ce,
    pb_dread  => rdata,
    pb_dwrite => wdata_mux);

    --must be changed because a read / write access may not be chip specific anymore
  --sram_ce <= pb_rd or pb_wr;

  amuxsel <= videocycle;

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

  onecycle <= (not be(3)) or (not be(2)) or (not be(1)) or (not be(0));

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

  process(videocycle, be, addr(1), hihalf, pb_rd, pb_wr)
  begin
    if videocycle = '1' then
      pb_bytesel <= "11";
    elsif pb_rd='1' or pb_wr='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;

  --Changing cs because we need cs = '1' when either ethernet or sram is needed
  --cs <= OPB_select when OPB_ABus(31 downto 20) = X"008" else '0';
  bus_req <= pb_rd or pb_wr;
  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';
-- ending changes
-- Thought process: How about not changing anything except which chip is
-- selected. If I can control whether RAM_CE_N or ETHERNET_CE_N are on or
-- off and simultaneously try to talk to them if they were the same type
-- of input output creature, then why wouldn't they respond correctly?
```

```
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;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_RST = '1' then
      addr <= X"000000";
    else
      addr <= OPB_ABus(23 downto 0);
    end if;
  end if;
end process;

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;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_Rst = '1' then
      wdata <= X"00000000";
    else
      wdata <= OPB_DBus;
    end if;
  end if;
end process;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if video_ce = '1' then
      video_data <= rdata;
    end if;
  end if;
end process;

-- Write the low two bytes if rce0 or rce1 is enabled

process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    UIO_DBus(15 downto 0) <= X"0000";
  elsif OPB_Clk'event and OPB_Clk = '1' then
    if rreset = '1' then
      UIO_DBus(15 downto 0) <= X"0000";
    elsif (rce1 or rce0) = '1' then
      UIO_DBus(15 downto 0) <= rdata(15 downto 0);
    end if;
  end if;
end process;

-- Write the high two bytes if rce0 is enabled

process (OPB_Clk, OPB_Rst)
```

```
  begin
    if OPB_Rst = '1' then
      UIO_DBus(31 downto 16) <= X"0000";
    elsif OPB_Clk'event and OPB_Clk = '1' then
      if rreset = '1' then
        UIO_DBus(31 downto 16) <= X"0000";
      elsif rce0 = '1' then
        UIO_DBus(31 downto 16) <= rdata(15 downto 0);
      end if;
    end if;
  end process;

  -- unused outputs

  UIO_errAck <= '0';
  UIO_retry <= '0';
  UIO_toutSup <= '0';

  UIO_xferAck <= xfer;

end Behavioral;
```

6.4.4a memoryctrl.vhd (Section 4.2)

```
--------------------------------------------------------------------------------
--
-- Memory controller state machine
--
-- Arbitrates between requests from the processor and video system
-- The video system gets priority
-- Also handles 32- to 16-bit datapath width conversion (sequencing)
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity memoryctrl is
  port (
    rst : in std_logic;
    clk : in std_logic;
    cs : in std_logic;                  -- chip select (address valid)
    select0 : in std_logic;             -- select (true through whole cycle)
    rnw : in std_logic;                 -- read/write'
    vreq : in std_logic;                -- video request
    onecycle : in std_logic;            -- one- or two-byte access (not four)

    videocycle : out std_logic;         -- acknowledges vreq
    hihalf : out std_logic;             -- doing bytes 2,3 of 32-bit access
    pb_wr : out std_logic;              -- Write request to off-chip memory
    pb_rd : out std_logic;              -- Read request to off-chip memory
    xfer : out std_logic;               -- Transfer acknowledge for OPB

    ce0 : out std_logic;                -- lower 16 bit OPB data latch enable
    ce1 : out std_logic;                -- upper 16 bit OPB data latch enable
    rres : out std_logic;               -- clear OPB data output latches

    video_ce : out std_logic);          -- video buffer latch enable
end memoryctrl;

architecture Behavioral of memoryctrl is
  -- State machine bits
  -- largely one-hot, but one ra and one rb bit can be true simultaneously
  signal r_idle : std_logic;
  signal r_common : std_logic;
  signal r_ra1 : std_logic;
  signal r_ra2 : std_logic;
  signal r_rb1 : std_logic;
  signal r_rb2 : std_logic;
  signal r_rb3 : std_logic;
  signal r_w : std_logic;
  signal r_xfer : std_logic;
```

```
  signal vcycle_1 : std_logic;
  signal vcycle_2 : std_logic;

  signal r_idle_next_state : std_logic;
  signal r_common_next_state : std_logic;
  signal r_ra1_next_state : std_logic;
  signal r_ra2_next_state : std_logic;
  signal r_rb1_next_state : std_logic;
  signal r_rb2_next_state : std_logic;
  signal r_rb3_next_state : std_logic;
  signal r_w_next_state : std_logic;
  signal r_xfer_next_state : std_logic;

begin

  -- Sequential process for the state machine

  process (clk, rst)
  begin
    if rst = '1' then
      r_idle <= '1';
      r_common <= '0';
      r_ra1 <= '0';
      r_ra2 <= '0';
      r_rb1 <= '0';
      r_rb2 <= '0';
      r_rb3 <= '0';
      r_w <= '0';
      r_xfer <= '0';
    elsif clk'event and clk='1' then
      r_idle <= r_idle_next_state;
      r_common <= r_common_next_state;
      r_ra1 <= r_ra1_next_state;
      r_ra2 <= r_ra2_next_state;
      r_rb1 <= r_rb1_next_state;
      r_rb2 <= r_rb2_next_state;
      r_rb3 <= r_rb3_next_state;
      r_w <= r_w_next_state;
      r_xfer <= r_xfer_next_state;
    end if;
  end process;

  -- Combinational next-state logic

  r_idle_next_state <= (r_idle and (not cs)) or r_xfer or (not select0);
  r_common_next_state <= select0 and
                         ((r_idle and cs) or (r_common and vreq));
  r_ra1_next_state <= select0 and
                      r_common and (not vreq) and rnw;
  r_ra2_next_state <= select0 and
                      r_ra1;
  r_rb1_next_state <= select0 and
                      ((r_common and not onecycle and not vreq and rnw) or
                       (r_rb1 and vreq));
  r_rb2_next_state <= select0 and
                      (r_rb1 and (not vreq));
  r_rb3_next_state <= select0 and
                      r_rb2;
  r_w_next_state <= select0 and
                    ((r_common and (not rnw) and (not vreq) and
                      (not onecycle)) or
                     (r_w and vreq));
  r_xfer_next_state <= select0 and
                       ((r_common and onecycle and
                         (not rnw) and (not vreq)) or
                        (r_w and (not vreq)) or (r_ra2 and onecycle) or r_rb3);

  -- Combinational output logic

  pb_wr <= (r_common and (not rnw) and (not vreq)) or (r_w and (not vreq));
  pb_rd <= vreq or (r_common and (not vreq) and rnw) or (r_rb1 and (not vreq));

  hihalf <= (r_w and (not vreq)) or (r_rb1 and (not vreq));

  ce0 <= r_ra2;
  ce1 <= r_rb3;
  rres <= r_xfer;
  xfer <= r_xfer;

  -- Two-cycle delay of video request
  -- (implicitly assumes video cycles always succeed)
```

```
  process (clk, rst)
  begin
    if(rst = '1') then
      vcycle_1 <= '0';
      vcycle_2 <= '0';
    elsif clk'event and clk='1' then
      vcycle_1 <= vreq;
      vcycle_2 <= vcycle_1;
    end if;
  end process;

  videocycle <= vreq;
  video_ce <= vcycle_2;

end Behavioral;
```

6.4.4b memoryctrl.vhd (Section 4.4)

```
--------------------------------------------------------------------------------
--
-- Memory controller state machine
--
-- Arbitrates between requests from the processor and video system
-- The video system gets priority
-- Also handles 32- to 16-bit datapath width conversion (sequencing)
--
-- Modified on 4/29/2006 to work with Spycam
--
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity memoryctrl is
  port (
    rst : in std_logic;
    clk : in std_logic;
    cs : in std_logic;                  -- chip select (address valid)
    select0 : in std_logic;             -- select (true through whole cycle)
    rnw : in std_logic;                 -- read/write'
    vreq : in std_logic;                -- video request
    onecycle : in std_logic;            -- one- or two-byte access (not four)
    videocycle : out std_logic;         -- acknowledges vreq
    hihalf : out std_logic;             -- doing bytes 2,3 of 32-bit access
    pb_wr : out std_logic;              -- Write request to off-chip memory
    pb_rd : out std_logic;              -- Read request to off-chip memory
    xfer : out std_logic;               -- Transfer acknowledge for OPB

    ce0 : out std_logic;                -- lower 16 bit OPB data latch enable
    ce1 : out std_logic;                -- upper 16 bit OPB data latch enable
    rres : out std_logic;               -- clear OPB data output latches

    video_ce : out std_logic;           -- video buffer latch enable
    eth_io : in std_logic);             -- ethernet ports added

    --bus_req : out std_logic           -- bus_req = 1 if SRAM and ETH are
                                     --not in use
end memoryctrl;

architecture Behavioral of memoryctrl is
  -- State machine bits
  -- largely one-hot, but one ra and one rb bit can be true simultaneously
  signal r_idle : std_logic;
  signal r_common : std_logic;
  signal r_ra1 : std_logic;
  signal r_ra2 : std_logic;
  signal r_rb1 : std_logic;
  signal r_rb2 : std_logic;
  signal r_rb3 : std_logic;
  signal r_w : std_logic;
  signal r_video : std_logic;

-- ETHERNET  state signals
  signal r_weth1 : std_logic ;
  signal r_weth2 : std_logic ;
  signal r_weth3 : std_logic ;
  --signal r_reth1 : std_logic ;
```

```
  --signal r_reth2 : std_logic ;
  --signal r_reth3 : std_logic ;

  signal r_xfer : std_logic;

  signal vcycle_1 : std_logic;
  signal vcycle_2 : std_logic;

-- NEXT state Signls
  signal r_idle_next_state : std_logic;
  signal r_common_next_state : std_logic;
  signal r_ra1_next_state : std_logic;
  signal r_ra2_next_state : std_logic;
  signal r_rb1_next_state : std_logic;
  signal r_rb2_next_state : std_logic;
  signal r_rb3_next_state : std_logic;
-- ETHERNET next state signals
  signal r_weth1_next_state : std_logic ;
  signal r_weth2_next_state : std_logic ;
  signal r_weth3_next_state : std_logic ;
  --signal r_reth1_next_state : std_logic ;
  --signal r_reth2_next_state : std_logic ;
  --signal r_reth3_next_state : std_logic ;
  signal r_w_next_state : std_logic;
  signal r_video_next_state : std_logic;
  signal r_xfer_next_state : std_logic;

--testing new implementation

begin

  -- Sequential process for the state machine

  process (clk, rst)
  begin
    if rst = '1' then
      r_idle <= '1';
      r_common <= '0';
      r_ra1 <= '0';
      r_ra2 <= '0';
      r_rb1 <= '0';
      r_rb2 <= '0';
      r_rb3 <= '0';
      r_w <= '0';
      r_video <= '0';
      r_weth1 <= '0';
      r_weth2 <= '0';
      r_weth3 <= '0';
      --r_reth1 <= '0';
      --r_reth2 <= '0';
      --r_reth3 <= '0';
      r_xfer <= '0';
    elsif clk'event and clk='1' then
      r_idle <= r_idle_next_state;
      r_common <= r_common_next_state;
      r_ra1 <= r_ra1_next_state;
      r_ra2 <= r_ra2_next_state;
      r_rb1 <= r_rb1_next_state;
      r_rb2 <= r_rb2_next_state;
      r_rb3 <= r_rb3_next_state;
      r_w <= r_w_next_state;
      r_video <= r_video_next_state;
      --ethernet read/write states
      r_weth1 <= r_weth1_next_state;
      r_weth2 <= r_weth2_next_state;
      r_weth3 <= r_weth3_next_state;
      --r_reth1 <= r_reth1_next_state;
      --r_reth2 <= r_reth2_next_state;
      --r_reth3 <= r_reth3_next_state;
      r_xfer <= r_xfer_next_state;
    end if;
  end process;

  -- Combinational next-state logic

  --r_idle_next_state <= (r_idle and (not cs)) or r_xfer or (not select0);
  r_idle_next_state <= (r_idle and (not cs)) or r_xfer or (not select0) or (r_video and
not vreq);
  r_video_next_state <= (r_idle and vreq);
  --r_common_next_state <= select0 and ((r_idle and cs) or (r_common and vreq));
  r_common_next_state <= select0 and (r_idle and cs and not vreq);
```

```
  --r_ra1_next_state <= select0 and r_common and (not vreq) and rnw;
  r_ra1_next_state <= select0 and r_common and rnw;
  r_ra2_next_state <= select0 and r_ra1 and not eth_io;
  --r_rb1_next_state <= select0 and ((r_common and not onecycle and not vreq and rnw) or
(r_rb1 and vreq));
  --r_rb2_next_state <= select0 and (r_rb1 and (not vreq));
  --r_rb3_next_state <= select0 and r_rb2;
  r_rb1_next_state <= select0 and (r_common and not onecycle and rnw);
  r_rb2_next_state <= select0 and (r_rb1 and not eth_io);
  r_rb3_next_state <= select0 and r_rb2;
  --r_w_next_state <= select0 and ((r_common and (not rnw) and (not vreq) and (not
onecycle)) or (r_w and vreq));
  --Ethernet read/write states
  r_weth1 <= select0 and ((r_w and eth_io) or (r_common and onecycle and not rnw and
eth_io)
                          or (r_ra1 and onecycle and eth_io) or (r_rb1 and eth_io));
  r_weth2 <= select0 and r_weth1;
  r_weth3 <= select0 and r_weth2;
  -- extra 32-bit write state
  r_w_next_state <= select0 and ((r_common and (not rnw) and (not onecycle)));
  --r_xfer_next_state <= select0 and
  --                      ((r_common and onecycle and (not rnw)) or
  --                       (r_w and (not vreq)) or (r_ra2 and onecycle) or r_rb3);
  r_xfer_next_state <= select0 and
                       ((r_common and onecycle and (not rnw) and (not eth_io)) or
                        (r_w and not eth_io) or (r_ra2 and onecycle and not eth_io) or
(r_rb3 and not eth_io)
                        or r_weth2);

  -- Combinational output logic

  --state in which the slave bus (ethernet / ram) is written
  --pb_wr <= (r_common and (not rnw) and (not vreq)) or (r_w and (not vreq));
  pb_wr <= (r_common and (not rnw)) or (r_w);
  --state in which slave bus (ethernet / ram) is read
  --pb_rd <= vreq or (r_common and (not vreq) and rnw) or (r_rb1 and (not vreq));
  -- giving read from slave a little more time
  pb_rd <= r_video or (r_common and rnw) or (r_rb1);

  --hihalf <= (r_w and (not vreq)) or (r_rb1 and (not vreq));
  hihalf <= (r_w) or (r_rb1);

  ce0 <= r_ra2;
  ce1 <= r_rb3;
  rres <= r_xfer;
  xfer <= r_xfer;

  -- Two-cycle delay of video request
  -- (implicitly assumes video cycles always succeed)
  process (clk, rst)
  begin
    if(rst = '1') then
      vcycle_1 <= '0';
      vcycle_2 <= '0';
    elsif clk'event and clk='1' then
      vcycle_1 <= r_video;
      vcycle_2 <= vcycle_1;
    end if;
  end process;

  videocycle <= r_video;
  video_ce <= vcycle_2;

end Behavioral;
```

6.4.4c memoryctrl.vhd (Section 4.5)

```
----------------------------------------------------------------------------------
--
-- Memory controller state machine
--
-- Arbitrates between requests from the processor and video system
-- The video system gets priority
-- Also handles 32- to 16-bit datapath width conversion (sequencing)
--
-- Modified on 4/29/2006 to work with Spycam
--
```

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

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity memoryctrl is
  port (
    rst : in std_logic;
    clk : in std_logic;
    cs : in std_logic;                  -- chip select (address valid)
    select0 : in std_logic;             -- select (true through whole cycle)
    rnw : in std_logic;                 -- read/write'
    vreq : in std_logic;                -- video request
    onecycle : in std_logic;            -- one- or two-byte access (not four)

    videocycle : out std_logic;         -- acknowledges vreq
    hihalf : out std_logic;             -- doing bytes 2,3 of 32-bit access
    pb_wr : out std_logic;              -- Write request to off-chip memory
    pb_rd : out std_logic;              -- Read request to off-chip memory
    xfer : out std_logic;               -- Transfer acknowledge for OPB

    ce0 : out std_logic;                -- lower 16 bit OPB data latch enable
    ce1 : out std_logic;                -- upper 16 bit OPB data latch enable
    rres : out std_logic;               -- clear OPB data output latches

    video_ce : out std_logic          -- video buffer latch enable
    -- ethernet ports added
    eth_io : in std_logic;
    --bus_req : out std_logic            -- bus_req = 1 if SRAM and ETH are
                                              --not in use

    );
    --end ethernet ports
end memoryctrl;

architecture Behavioral of memoryctrl is
  -- State machine bits
  -- largely one-hot, but one ra and one rb bit can be true simultaneously
  signal r_idle : std_logic;
  signal r_common : std_logic;
  signal r_ra1 : std_logic;
  signal r_ra2 : std_logic;
  signal r_rb1 : std_logic;
  signal r_rb2 : std_logic;
  signal r_rb3 : std_logic;
  signal r_w : std_logic;
  -- ethernet state signals
  signal r_weth1 : std_logic ;
  signal r_weth2 : std_logic ;
  signal r_weth3 : std_logic ;
  signal r_reth1 : std_logic ;
  signal r_reth2 : std_logic ;
  signal r_reth3 : std_logic ;
  signal r_xfer : std_logic;

  signal vcycle_1 : std_logic;
  signal vcycle_2 : std_logic;

  signal r_idle_next_state : std_logic;
  signal r_common_next_state : std_logic;
  signal r_ra1_next_state : std_logic;
  signal r_ra2_next_state : std_logic;
  signal r_rb1_next_state : std_logic;
  signal r_rb2_next_state : std_logic;
  signal r_rb3_next_state : std_logic;
  -- ethernet next state signals
  signal r_weth1_next_state : std_logic ;
  signal r_weth2_next_state : std_logic ;
  signal r_weth3_next_state : std_logic ;
  signal r_reth1_next_state : std_logic ;
  signal r_reth2_next_state : std_logic ;
```

```
  signal r_reth3_next_state : std_logic ;
  signal r_w_next_state : std_logic;
  signal r_xfer_next_state : std_logic;
-- Signal for Video Request.
  signal vreq2 : std_logic;

begin

  -- Sequential process for the state machine

  process (clk, rst)
  begin
    if rst = '1' then
      r_idle <= '1';
      r_common <= '0';
      r_ra1 <= '0';
      r_ra2 <= '0';
      r_rb1 <= '0';
      r_rb2 <= '0';
      r_rb3 <= '0';
      r_w <= '0';
      r_weth1 <= '0';
      r_weth2 <= '0';
      r_weth3 <= '0';
      r_reth1 <= '0';
      r_reth2 <= '0';
      r_reth3 <= '0';
      r_xfer <= '0';
    elsif clk'event and clk='1' then
      r_idle <= r_idle_next_state;
      r_common <= r_common_next_state;
      r_ra1 <= r_ra1_next_state;
      r_ra2 <= r_ra2_next_state;
      r_rb1 <= r_rb1_next_state;
      r_rb2 <= r_rb2_next_state;
      r_rb3 <= r_rb3_next_state;
      r_w <= r_w_next_state;
      --ethernet read/write states
      r_weth1 <= r_weth1_next_state;
      r_weth2 <= r_weth2_next_state;
      r_weth3 <= r_weth3_next_state;
      r_reth1 <= r_reth1_next_state;
      r_reth2 <= r_reth2_next_state;
      r_reth3 <= r_reth3_next_state;
      r_xfer <= r_xfer_next_state;
    end if;
  end process;

--  new logic from state machine
  r_idle_next_state <= (r_idle and (not cs)) or r_xfer or (not select0);
  r_common_next_state <= select0 and
                         ((r_idle and cs) or (r_common and vreq2));
  r_ra1_next_state <= select0 and r_common and (not vreq2) and rnw and (not eth_io);
  r_ra2_next_state <= select0 and r_ra1;
  r_rb1_next_state <= select0 and ((r_common and not onecycle and not vreq2 and rnw and
(not eth_io)) or
                      (r_rb1 and vreq2));
  r_rb2_next_state <= select0 and (r_rb1 and (not vreq2));
  r_rb3_next_state <= select0 and r_rb2;
  r_w_next_state <= select0 and ((r_common and (not rnw) and (not vreq2) and
                     (not onecycle) and (not eth_io)) or
                    (r_w and vreq2));
  r_weth1_next_state <= select0 and (r_common and not rnw and (not vreq2) and eth_io);
  r_weth2_next_state <= select0 and (r_weth1);
  r_weth3_next_state <= select0 and (r_weth2);
  r_reth1_next_state <= select0 and (r_common and rnw and eth_io and not(vreq2));
  r_reth2_next_state <= select0 and (r_reth1);
  r_reth3_next_state <= select0 and (r_reth2);
  r_xfer_next_state <= select0 and
                       ((r_common and onecycle and
                         (not rnw) and (not vreq2) and not(eth_io)) or
```

```
                        (r_w and (not vreq)) or (r_ra2 and onecycle) or r_rb3 or r_weth3
or r_reth3);

  -- Combinational output logic

  --state in which the slave bus (ethernet / ram) is written
  pb_wr <= (r_common and (not rnw) and (not vreq)) or (r_w and (not vreq))
            or r_weth1 or r_weth2 or r_weth3;
  --state in which slave bus (ethernet / ram) is read
  --pb_rd <= vreq or (r_common and (not vreq) and rnw) or (r_rb1 and (not vreq));
  -- giving read from slave a little more time
  pb_rd <= vreq or (r_common and (not vreq) and rnw) or r_ra1 or (r_rb1 and (not vreq))
                   or r_reth1 or r_reth2 or r_reth3;


  hihalf <= (r_w and (not vreq)) or (r_rb1 and (not vreq));

  ce0 <= r_ra2;
  ce1 <= r_rb3;
  rres <= r_xfer;
  xfer <= r_xfer;

  -- Two-cycle delay of video request
  -- (implicitly assumes video cycles always succeed)
  process (clk, rst)
  begin
    if(rst = '1') then
      vcycle_1 <= '0';
      vcycle_2 <= '0';
    elsif clk'event and clk='1' then
      vcycle_1 <= vreq;
      vcycle_2 <= vcycle_1;
    end if;
  end process;

  videocycle <= vreq;
  video_ce <= vcycle_2;

end Behavioral;
```

6.4.5 pad_io.vhd

```
----------------------------------------------------------------------------------
--
-- I/O Pads and associated circuity for the XSB-300E board
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
-- Pad drivers, a little glue logic, and flip-flops for most bus signals
--
-- All signals, both control and the data and address busses, are registered.
-- FDC and FDP flip-flops are forced into the pads to do this.
--
-- Only the data bus is mildly complex: it latches data in both directions
-- as well as delaying the tristate control signals a cycle.
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- include UNISIM libs

entity pad_io is
  port (
    clk : 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;
```

```
    --begin ethernet
    ETHERNET_CS_N : out std_logic;
    ETHERNET_RDY : in std_logic;
    ETHERNET_IREQ : in std_logic;
    ETHERNET_IOCS16_N : in std_logic;
    --end ethernet
    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; --added
    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

  -- Flip-flop with asynchronous clear

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

  -- Flip-flop with asynchronous preset

  component FDP
    port (
      C : in std_logic;
      PRE : 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 FDC : component is "true";
  attribute iob of FDP : component is "true";

  -- Fast off-chip output buffer, low-voltage TTL levels, 24 mA drive
  -- I is the on-chip signal, O is the pad

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

  -- Fast off-chip input/output buffer, low-voltage TTL levels, 24 mA drive
  -- T is the tristate control input, IO is the pad,

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

  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_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 ramce_n, ram_ce_n1: std_logic;
  --add ethernet
  signal ethernet_ce_n, ethernet_ce_n1: std_logic;
  signal dataz : std_logic;

begin
```

```
-- Write enable

we_n <= not pb_wr;

we_ff : FDP port map (
  C => clk, PRE => rst,
  D => we_n,
  Q => pb_we_n1);

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

-- Output Enable

oe_n <= not pb_rd;

oe_ff : FDP port map (
  C => clk,
  PRE => rst,
  D => oe_n,
  Q => pb_oe_n1);

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

-- RAM Chip Enable

ramce_n <= not ram_ce;

ramce_ff : FDP port map (
  C => clk, PRE => rst,
  D => ramce_n,
  Q => ram_ce_n1);
ramce_pad : OBUF_F_24 port map (
  O => RAM_CE_N,
  I => ram_ce_n1);

ethernet_ce_n <= not ethernet_ce;

ethernetce_ff : FDP port map (
  C => clk, PRE => rst,
  D => ethernet_ce_n,
  Q => ethernet_ce_n1);

ethernetce_pad : OBUF_F_24 port map (
  O => ETHERNET_CS_N,
  I => ethernet_ce_n1);

-- Upper byte enable

ub_n <= not pb_ub;

ub_ff : FDP port map (
  C => clk, PRE => rst,
  D => ub_n,
  Q => pb_ub_n1);

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

-- Lower byte enable

lb_n <= not pb_lb;

lb_ff : FDP port map (
  C => clk,
  PRE => rst,
  D => lb_n,
  Q => pb_lb_n1);

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

-- 20-bit address bus

addressbus : for i in 0 to 19 generate
```

```
    address_ff : FDC port map (
      C => clk, CLR => rst,
      D => pb_addr(i),
      Q => pb_addr_1(i));

    address_pad : OBUF_F_24 port map (
      O => PB_A(i),
      I => pb_addr_1(i));
  end generate;

  -- 16-bit data bus

  dataz <= (not pb_wr) or pb_rd;

  databus : for i in 0 to 15 generate
    dtff : FDP port map (               -- Trisate enable
      C => clk,
      PRE => rst,
      D => dataz,
      Q => pb_tristate(i));

    drff : FDP port map (
      C => clk,
      PRE => rst,
      D => pb_dread_a(i),
      Q => pb_dread(i));

    dwff : FDP port map (
      C => clk,
      PRE => rst,
      D => pb_dwrite(i),
      Q => pb_dwrite_1(i));

    data_pad : IOBUF_F_24 port map (
      O => pb_dread_a(i),
      IO => PB_D(i),
      I => pb_dwrite_1(i),
      T => pb_tristate(i));
  end generate;

end Behavioral;
```

6.4.6 vgatiming.vhd

```
--------------------------------------------------------------------------------
--
-- VGA timing and address generator
--
-- Fixed-resolution address generator.  Generates h-sync, v-sync, and blanking
-- signals along with a 20-bit RAM address.  H-sync and v-sync signals are
-- delayed two cycles to compensate for the DAC pipeline.
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
--------------------------------------------------------------------------------

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

entity vga_timing is
  port (
    pixel_clock  : in std_logic;
    reset        : in std_logic;
    h_sync_delay : out std_logic;
    v_sync_delay : out std_logic;
    blank        : out std_logic;
    vga_ram_read_address : out std_logic_vector(19 downto 0));
end vga_timing;

architecture Behavioral of vga_timing is

  constant SRAM_DELAY : integer := 3;

-- 640 X 480 @ 60Hz with a 25.175 MHz pixel clock
  constant H_ACTIVE      : integer := 640;
  constant H_FRONT_PORCH : integer := 16;
```

```
  constant H_BACK_PORCH  : integer := 48;
  constant H_TOTAL       : integer := 800;

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

  signal line_count  : std_logic_vector (9 downto 0);   -- Y coordinate
  signal pixel_count : std_logic_vector (10 downto 0);  -- X coordinate

  signal h_sync : std_logic;  -- horizontal sync
  signal v_sync : std_logic;  -- vertical sync

  signal h_sync_delay0 : std_logic; -- h_sync delayed 1 clock
  signal v_sync_delay0 : std_logic; -- v_sync delayed 1 clock

  signal h_blank : std_logic;         -- horizontal blanking
  signal v_blank : std_logic;         -- vertical blanking

  -- flag to reset the ram address during vertical blanking
  signal reset_vga_ram_read_address : std_logic;

  -- flag to hold the address during horizontal blanking
  signal hold_vga_ram_read_address : std_logic;

  signal ram_address_counter : std_logic_vector (19 downto 0);

begin

  -- Pixel counter

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      pixel_count <= "00000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_TOTAL - 1) then
        pixel_count <= "00000000000";
      else
        pixel_count <= pixel_count + 1;
      end if;
    end if;
  end process;

  -- Horizontal sync

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      h_sync <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_ACTIVE + H_FRONT_PORCH - 1) then
        h_sync <= '1';
      elsif pixel_count = (H_TOTAL - H_BACK_PORCH - 1) then
        h_sync <= '0';
      end if;
    end if;
  end process;

  -- Line counter

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      line_count <= "0000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if ((line_count = V_TOTAL - 1) and (pixel_count = H_TOTAL - 1)) then
        line_count <= "0000000000";
      elsif pixel_count = (H_TOTAL - 1) then
        line_count <= line_count + 1;
      end if;
    end if;
  end process;

  -- Vertical sync

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      v_sync <= '0';
```

```
  elsif pixel_clock'event and pixel_clock = '1' then
    if line_count = (V_ACTIVE + V_FRONT_PORCH -1) and
       pixel_count = (H_TOTAL - 1) then
      v_sync <= '1';
    elsif line_count = (V_TOTAL - V_BACK_PORCH - 1) and
          pixel_count = (H_TOTAL - 1) then
      v_sync <= '0';
    end if;
  end if;
end process;

-- Add two-cycle delays to h/v_sync to compensate for the DAC pipeline

process ( pixel_clock, reset )
begin
  if reset = '1' then
    h_sync_delay0 <= '0';
    v_sync_delay0 <= '0';
    h_sync_delay  <= '0';
    v_sync_delay  <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    h_sync_delay0 <= h_sync;
    v_sync_delay0 <= v_sync;
    h_sync_delay  <= h_sync_delay0;
    v_sync_delay  <= v_sync_delay0;
  end if;
end process;

-- Horizontal blanking

-- The constants are offset by two to compensate for the delay
-- in the composite blanking signal

process ( pixel_clock, reset )
begin
  if reset = '1' then
    h_blank <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if pixel_count = (H_ACTIVE - 2) then
      h_blank <= '1';
    elsif pixel_count = (H_TOTAL - 2) then
      h_blank <= '0';
    end if;
  end if;
end process;

-- Vertical Blanking

-- The constants are offset by two to compensate for the delay
-- in the composite blanking signal

process ( pixel_clock, reset )
begin
  if reset = '1' then
    v_blank <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if line_count = (V_ACTIVE - 1) and pixel_count = (H_TOTAL - 2) then
      v_blank <= '1';
    elsif line_count = (V_TOTAL - 1) and pixel_count = (H_TOTAL - 2) then
      v_blank <= '0';
    end if;
  end if;
end process;

-- Composite blanking

process ( pixel_clock, reset )
begin
  if reset = '1' then
    blank <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if (h_blank or v_blank) = '1' then
      blank <= '1';
    else
      blank <= '0';
    end if;
  end if;
end process;

-- RAM address counter
```

```
  -- Two control signals:

  -- reset_ram_read_address is active from the end of each field until the
  -- beginning of the next

  -- hold_vga_ram_read_address is active from the end of each line to the
  -- start of the next

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

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      hold_vga_ram_read_address <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = ((H_ACTIVE - 1) - SRAM_DELAY) then
        -- hold the address counter at the end of active video
        hold_vga_ram_read_address <= '1';
      elsif pixel_count = ((H_TOTAL - 1) - SRAM_DELAY) then
        -- re-enable the address counter at the start of active video
        hold_vga_ram_read_address <= '0';
      end if;
    end if;
  end process;

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      ram_address_counter <= "00000000000000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if reset_vga_ram_read_address = '1' then
        ram_address_counter <= "00000000000000000000";
      elsif hold_vga_ram_read_address = '0' then
        ram_address_counter <= ram_address_counter + 1;
      end if;
    end if;
  end process;

  vga_ram_read_address <= ram_address_counter;

end Behavioral;
```

6.4.7 vga.vhd

```
--------------------------------------------------------------------------------
--
-- VGA video generator
--
-- Uses the vga_timing module to generate hsync etc.
-- Massages the RAM address and requests cycles from the memory controller
-- to generate video using one byte per pixel
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity vga is
  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 vga;

architecture Behavioral of vga is

  -- Fast low-voltage TTL-level I/O pad with 12 mA drive

  component OBUF_F_12
    port (
      O : out STD_ULOGIC;
      I : in STD_ULOGIC);
  end component;

  -- Basic edge-sensitive flip-flop

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

  -- Force instances of FD into pads for speed

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

  component vga_timing
    port (
      h_sync_delay         : out std_logic;
      v_sync_delay         : out std_logic;
      blank                : out std_logic;
      vga_ram_read_address : out std_logic_vector (19 downto 0);
      pixel_clock          : in std_logic;
      reset                : in std_logic);
  end component;

  signal r                      : std_logic_vector (9 downto 0);
  signal g                      : std_logic_vector (9 downto 0);
  signal b                      : std_logic_vector (9 downto 0);
  signal blank                  : std_logic;
  signal hsync                  : std_logic;
  signal vsync                  : std_logic;
  signal vga_ram_read_address   : std_logic_vector(19 downto 0);
  signal vreq                   : std_logic;
  signal vreq_1                 : std_logic;
  signal load_video_word        : std_logic;
  signal vga_shreg              : std_logic_vector(15 downto 0);

begin

  st : vga_timing port map (
    pixel_clock => pix_clk,
    reset => rst,
    h_sync_delay => hsync,
    v_sync_delay => vsync,
    blank => blank,
    vga_ram_read_address => vga_ram_read_address);

  -- Video request is true when the RAM address is even

  -- FIXME: This should be disabled during blanking to reduce memory traffic

  vreq <= not vga_ram_read_address(0);

  -- Generate load_video_word by delaying vreq two cycles

  process (pix_clk)
  begin
    if pix_clk'event and pix_clk='1' then
      vreq_1 <= vreq;
```

```
    load_video_word <= vreq_1;
  end if;
end process;

-- Generate video_req (to the RAM controller) by delaying vreq by
-- a cycle synchronized with the pixel clock

process (clk)
begin
  if clk'event and clk='1' then
    video_req <= pix_clk and vreq;
  end if;
end process;

-- The video address is the upper 19 bits from the VGA timing generator
-- because we are using two pixels per word and the RAM address counts words

video_addr <= '0' & vga_ram_read_address(19 downto 1);

-- The video shift register: either load it from RAM or shift it up a byte

process (pix_clk)
begin
  if pix_clk'event and pix_clk='1' then
    if load_video_word = '1' then
      vga_shreg <= video_data;
    else
      -- Shift the low byte of read video data into the high byte
      vga_shreg <= vga_shreg(7 downto 0) & "00000000";
    end if;
  end if;
end process;

-- Copy the upper byte of the video word to the color signals
-- Note that we use three bits for red and green and two for blue.

r(9 downto 7) <= vga_shreg (15 downto 13);
r(6 downto 0) <= "0000000";
g(9 downto 7) <= vga_shreg (12 downto 10);
g(6 downto 0) <= "0000000";
b(9 downto 8) <= vga_shreg (9 downto 8);
b(7 downto 0) <= "00000000";

-- Video clock I/O pad to the DAC

vidclk : OBUF_F_12 port map (
  O => VIDOUT_clk,
  I => pix_clk);

-- Control signals: hsync, vsync, and blank

hsync_ff : FD port map (
  C => pix_clk,
  D => not hsync,
  Q => VIDOUT_HSYNC_N );

vsync_ff : FD port map (
  C => pix_clk,
  D => not vsync,
  Q => VIDOUT_VSYNC_N );

blank_ff :  FD port map (
  C => pix_clk,
  D => not blank,
  Q => VIDOUT_BLANK_N );

-- Three digital color signals

rgb_ff : for i in 0 to 9 generate

  r_ff : FD port map (
    C => pix_clk,
    D => r(i),
    Q => VIDOUT_RCR(i) );

  g_ff : FD port map (
    C => pix_clk,
    D => g(i),
    Q => VIDOUT_GY(i) );

  b_ff : FD port map (
```

```
      C => pix_clk,
      D => b(i),
      Q => VIDOUT_BCB(i) );

  end generate;

end Behavioral;
```

6.5 Clock generator

6.5.1 clkgen_v2_1_0.pao

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

BEGIN clkgen

## 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 fpga_reset = "", DIR = OUT


END
```

6.5.2 clkgen_v2_1_0.mpd

```
###################################################################################
##
## 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
```

6.5.3 clkgen.v

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

BEGIN clkgen

## 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 fpga_reset = "", DIR = OUT
END
```

6.6 Ethernet Driver Source Code

6.6.1 ether_reg.h

```
#ifndef _ETHER_REG_H
#define _ETHER_REG_H

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

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

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


/* NIC page0 register offsets */
#define CMDR    (0x00*2) /* command register for read & write*/
#define PSTART  (0x01*2) /* page start register for write*/
#define PSTOP   (0x02*2) /* page stop register for write*/
#define BNRY    (0x03*2) /* boundary reg for rd and wr*/
#define TPSR    (0x04*2) /* tx start page start reg for wr*/
#define TBCR0   (0x05*2) /* tx byte count 0 reg for wr*/
#define TBCR1   (0x06*2) /* tx byte count 1 reg for wr*/
#define ISR     (0x07*2) /* interrupt status reg for rd and wr*/
#define RSAR0   (0x08*2) /* low byte of remote start addr*/
#define RSAR1   (0x09*2) /* hi byte of remote start addr*/
#define RBCR0   (0x0A*2) /* remote byte count reg 0 for wr*/
#define RBCR1   (0x0B*2) /* remote byte count reg 1 for wr*/
#define RCR     (0x0C*2) /* rx configuration reg for wr*/
#define TCR     (0x0D*2) /* tx configuration reg for wr*/
#define DCR     (0x0E*2) /* data configuration reg for wr*/
#define IMR     (0x0F*2) /* interrupt mask reg for wr*/

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

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

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


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

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



/* Ether Function Prototypes*/
```

```
   void diag(); /* diagnostic to check if the ethernet is working*/
void delay(int mult); /*delay loop*/
int init_etherne(CONFIGNE* cp);
void resetnic(CONFIGNE* cp);
void dmaRead(void *data, WORD addr, WORD length);
void dmaWrite(WORD *data, WORD addr, WORD length);
int send(WORD len, WORD *packet);
void getnic(WORD addr, BYTE data[], WORD len);
void receive(int* len, BYTE* buffer);


/* Mac functions for the xilinx xilnet library 2.0.0 */
void XEmac_SendFrame(unsigned int MYMAC_BASEADDR,BYTE *pkt, int len);
/* return size integer*/
int XEmac_RecvFrame(unsigned int MYMAC_BASEADDR, BYTE *buf);


/*Global Variables*/
#define WORDMODE 1

static int promisc=0;
/* packet prototype */


#endif
```

6.6.2 etherFunc.c

```
#include "ether_reg.h"

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


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

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

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

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

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

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

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

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

```
  outnic(TPSR, TXSTART);

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

}

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

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

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

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

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

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

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

  print("         Default Value Test\r\n");
  print("Switching to page 0\r\n");
  outnic(CMDR,21);
  print("Reading from 0x16 (value should be 0x15): ");
  putnum(innic(0x16*2));
  print("\r\n");
  print("Reading from 0x12 (value should be 0x0c): ");
  putnum(innic(0x12*2));
  print("\r\n");
  print("Reading from 0x13 (value should be 0x12): ");
  putnum(innic(0x13*2));
  print("\r\n\r\n*************** End of Diagnostic ***************\r\n\r\n");
}


void initpacket(Xuint8 *packet)
{

  packet[0]=0xFF;
  packet[1]=0xFF;
  packet[2]=0xFF;
  packet[3]=0xFF;
  packet[4]=0xFF;
  packet[5]=0xFF;
  packet[6]=0x00;
  packet[6]=0x0D;
  packet[8]=0x60;
  packet[9]=0x7F;
  packet[10]=0xF9;
  packet[11]=0xAF;
  packet[12]=0x08;
  packet[13]=0x00;
  packet[14]=0x45;
  packet[15]=0x00;
  packet[16]=0x00;
```

```
  packet[17]=0xBE;
  packet[18]=0x00;
  packet[19]=0x00;
  packet[20]=0x00;
  packet[21]=0x00;
  packet[22]=0x01;
  packet[23]=0x11;
  packet[24]=0xB8;
  packet[25]=0xEE;
  packet[26]=0x80;
  packet[27]=0x3B;
  packet[28]=0x95;
  packet[29]=0xF9;
  packet[30]=0xE4;
  packet[31]=0x05;
  packet[32]=0x06;
  packet[33]=0x07;
  packet[34]=0x0C;
  packet[35]=0x2C;
  packet[36]=0x1A;
  packet[37]=0x85;
  packet[38]=0x00;
  packet[39]=0xAA;
  packet[40]=0x00;
  packet[41]=0x00;
  //extra byte
  packet[42]=0x00;

}

void XEmac_SendFrame(unsigned int MYMAC_BASEADDR,BYTE *pkt, int len){
    int a = send (len, (WORD) *pkt);
    return a;
}

int XEmac_RecvFrame(unsigned int MYMAC_BASEADDR, BYTE *buf){
    int len;
    receive (&len, *buf);
    return len;
}
```

6.6.3 etherReceive.c

```
#include "ether_reg.h"

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

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

  *len = nichdr.len;

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

  outnic(ISR,0xff);

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

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

    count = WORDMODE ? len>>1 : len;    /* Halve byte count if word I/P */
    outnic(ISR, 0x40);                  /* Clear remote DMA interrupt flag */
    outnic(RBCR0, len&0xff);            /* Byte count */
    outnic(RBCR1, len>>8);
```

```
    outnic(RSAR0, addr&0xff);           /* Data addr */
    outnic(RSAR1, addr>>8);
    outnic(CMDR, 0x0a);                 /* Start, DMA remote read */
#if WORDMODE
    dataw = (WORD *)data;               /* Use pointer for speed */
    while(count--)                      /* Get words */
        *dataw++ = innic(DATAPORT);
    if (len & 1)                        /* If odd length, do last byte */
        *(BYTE *)dataw = innic(DATAPORT);
#else
    while(count--)                      /* Get bytes */
        *data++ = innic(DATAPORT);
#endif

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

}
```

6.6.4 etherSend.c

```
//Function to write packets to the MAC layer
#include "ether_reg.h"

static int incr = 0;

//"len" is the int length of the packet whose address is
//contained in the pointer "data"
int send(WORD len, WORD *data){

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

  counter = len>>1;

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

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

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

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

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

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

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

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

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

  outnic(ISR,0xFF);

  return 0;
```

```
}
```

6.7 JPEG Decompression Source Code

6.7.1 jpeg.h

```
/*#define SPY*/
/* Leave structures in memory,output something and dump core in the event
   of a failure: */
#define DEBUG 0
#define VGA_START 0x00800000


/*---------------------------------*/
/* JPEG format parsing markers here */
/*---------------------------------*/

#define SOI_MK 0xFFD8         /* start of image      */
#define APP_MK 0xFFE0         /* custom, up to FFEF */
#define COM_MK 0xFFFE         /* commment segment    */
#define SOF_MK 0xFFC0         /* start of frame      */
#define SOS_MK 0xFFDA         /* start of scan       */
#define DHT_MK 0xFFC4         /* Huffman table       */
#define DQT_MK 0xFFDB         /* Quant. table                */
#define DRI_MK 0xFFDD         /* restart interval    */
#define EOI_MK 0xFFD9         /* end of image                */
#define MK_MSK 0xFFF0

#define RST_MK(x)      ( (0xFFF8&(x)) == 0xFFD0 )
                       /* is x a restart interval ? */

#define X_SIZE  50
#define Y_SIZE  67
#define N_COMP  3
#define MCU_SX 16
#define MCU_SY 16

/*---------------------------------------------------------- */
/* all kinds of macros here                                  */
/*---------------------------------------------------------- */

#define first_quad(c)   ((c) >> 4)        /* first 4 bits in file order */
#define second_quad(c)  ((c) & 15)

#define HUFF_ID(hclass, id)       (2 * (hclass) + (id))

#define DC_CLASS        0
#define AC_CLASS        1


/*-----------------------------------------------------*/
/* JPEG data types here                                  */
/*-----------------------------------------------------*/

typedef union {               /* block of pixel-space values */
  unsigned char       block[8][8];
  unsigned char       linear[64];
} PBlock;

typedef union {               /* block of frequency-space values */
  int block[8][8];
  int linear[64];
} FBlock;


/* component descriptor structure */

typedef struct {
  unsigned char       CID;    /* component ID */
  unsigned char       IDX;    /* index of first block in MCU */

  unsigned char       HS;     /* sampling factors */
  unsigned char       VS;
  unsigned char       HDIV;   /* sample width ratios */
  unsigned char       VDIV;

  char         QT;     /* QTable index, 2bits        */
  char         DC_HT;  /* DC table index, 1bit */
```

```
  char         AC_HT;  /* AC table index, 1bit */
  int          PRED;   /* DC predictor value */
} cd_t;


/*-------------------------------------------*/
/* global variables here                     */
/*-------------------------------------------*/


extern cd_t   comp[3]; /* for every component, useful stuff */

extern PBlock *MCU_buff[10];  /* decoded component buffer */
                             /* between IDCT and color convert */
extern int    MCU_valid[10];  /* for every DCT block, component id then -1 */

extern PBlock *QTable[4];     /* three quantization tables */
extern int    QTvalid[4];     /* at most, but seen as four ... */

extern FILE *fi;
extern FILE *fo;

/* picture attributes */
extern int x_size, y_size;    /* Video frame size     */
extern int rx_size, ry_size;  /* down-rounded Video frame size */
                              /* in pixel units, multiple of MCU */
extern int MCU_sx, MCU_sy;    /* MCU size in pixels   */
extern int mx_size, my_size;  /* picture size in units of MCUs */
extern int n_comp;            /* number of components 1,3 */

/* processing cursor variables */
extern int in_frame, curcomp, MCU_row, MCU_column;
                      /* current position in MCU unit */

/* RGB buffer storage */
extern unsigned char *ColorBuffer;   /* MCU after color conversion */
extern unsigned char *FrameBuffer;   /* complete final RGB image */

extern PBlock        *PBuff;
extern FBlock        *FBuff;

/* process statistics */
extern int stuffers;   /* number of stuff bytes in file */
extern int passed;     /* number of bytes skipped looking for markers */

extern int verbose;

extern int count;


/*-----------------------------------------*/
/* prototypes from utils.c                 */
/*-----------------------------------------*/

extern void show_FBlock(FBlock *S);
extern void show_PBlock(PBlock *S);
/* extern void bin_dump(FILE *fi);  */

extern int      ceil_div(int N, int D);
extern int      floor_div(int N, int D);
extern void     reset_prediction();
extern int     reformat(unsigned long S, int good);
/* extern void     free_structures(); */
/* extern void suicide();  */
/* extern void aborted_stream(FILE *fi, FILE *fo); */
extern void    RGB_save(FILE *fo);

/*-----------------------------------------*/
/* prototypes from parse.c                 */
/*-----------------------------------------*/

extern void    clear_bits();
extern unsigned long  get_bits(unsigned char **vfile_ptr, int *vfile_size_ptr, int
number);
extern unsigned char  get_one_bit(unsigned char **vfile_ptr, int *vfile_size_ptr);
extern unsigned int   get_size(unsigned char **vfile_ptr, int *vfile_size_ptr);
extern unsigned int     get_next_MK(unsigned char **vfile_ptr, int *vfile_size_ptr);
extern int     load_quant_tables(unsigned char **vfile_ptr, int *vfile_size_ptr);
extern int      init_MCU();
extern void     skip_segment(unsigned char **vfile_ptr, int *vfile_size_ptr);
extern int      process_MCU(unsigned char **vfile_ptr, int *vfile_size_ptr);
```

```
/*-----------------------------------------*/
/* prototypes from fast_idct.c              */
/*-----------------------------------------*/

extern void    IDCT(const FBlock *S, PBlock *T);

/*-----------------------------------------*/
/* prototypes from color.c              */
/*-----------------------------------------*/

extern void    color_conversion();

/*-----------------------------------------*/
/* prototypes from table_vld.c or tree_vld.c */
/*-----------------------------------------*/

extern int     load_huff_tables(unsigned char **vfile_ptr, int *vfile_size_ptr);
extern unsigned char    get_symbol(unsigned char **vfile_ptr, int *vfile_size_ptr, int
select);

/*-----------------------------------------*/
/* prototypes from huffman.c            */
/*-----------------------------------------*/

extern void    unpack_block(unsigned char **vfile_ptr, int *vfile_size_ptr, FBlock *T,
int comp);
               /* unpack, predict, dequantize, reorder on store */

/*-----------------------------------------*/
/* prototypes from spy.c                */
/*-----------------------------------------*/

extern void    trace_bits(int number, int type);
extern void    output_stats(char *dumpfile);
```

6.7.2 color.c

```
#include <stdlib.h>
#include <stdio.h>

#include "jpeg.h"

/* Ensure number is >=0 and <=255                       */
#define Saturate(n)    ((n) > 0 ? ((n) < 255 ? (n) : 255) : 0)

/*-------------------------------------------------------------*/


/* internal color conversion utility */
/*
static unsigned char
get_comp(int n, int i, int j)
{
  int ip = i >> comp[n].VDIV; / * get coordinates in n-th comp pixels * /
  int jp = j >> comp[n].HDIV; / * within the MCU * /

  return MCU_buff[comp[n].IDX+comp[n].HS*(ip>>3)+(jp>>3)]->block[ip&7][jp&7];
  / * this is the right block in MCU, and this right sample * /
}
*/
/* and alternate macro, a little faster */
/*
#define get_comp(t,n,i,j) { int ip = i >> comp[n].VDIV; \
                            int jp = j >> comp[n].HDIV; \
t = MCU_buff[comp[n].IDX+comp[n].HS*(ip>>3)+(jp>>3) ]->block[ip&7][jp& 7]; \
                     }
*/

/*-----------------------------------------*/
/* rules for color conversion:                */
/*  r = y             +1.402  v       */
/*  g = y -0.34414u    -0.71414v       */
/*  b = y +1.772  u                    */
/* Approximations: 1.402 # 7/5 = 1.400        */
/*             .71414 # 357/500 = 0.714 */
/*             .34414 # 43/125        = 0.344  */
```

```
/*             1.772  = 443/250        */
/*-------------------------------------*/
/* Approximations: 1.402 # 359/256 = 1.40234 */
/*             .71414 # 183/256 = 0.71484 */
/*             .34414 # 11/32 = 0.34375 */
/*             1.772 # 227/128 = 1.7734 */
/*-------------------------------------*/

void
color_conversion(void)
{
  int  i, j;
  unsigned char y,cb,cr;
  signed char rcb, rcr;
  long r,g,b;
  long offset;

  for (i = 0; i < MCU_sy; i++)   /* pixel rows */
    {
      int ip_0 = i >> comp[0].VDIV;
      int ip_1 = i >> comp[1].VDIV;
      int ip_2 = i >> comp[2].VDIV;
      int inv_ndx_0 = comp[0].IDX + comp[0].HS * (ip_0 >> 3);
      int inv_ndx_1 = comp[1].IDX + comp[1].HS * (ip_1 >> 3);
      int inv_ndx_2 = comp[2].IDX + comp[2].HS * (ip_2 >> 3);
      int ip_0_lsbs = ip_0 & 7;
      int ip_1_lsbs = ip_1 & 7;
      int ip_2_lsbs = ip_2 & 7;
      int i_times_MCU_sx = i * MCU_sx;

      for (j = 0; j < MCU_sx; j++)   /* pixel columns */
        {
          int jp_0 = j >> comp[0].HDIV;
          int jp_1 = j >> comp[1].HDIV;
          int jp_2 = j >> comp[2].HDIV;

          y  = MCU_buff[inv_ndx_0 + (jp_0 >> 3)]->block[ip_0_lsbs][jp_0 & 7];
          cb = MCU_buff[inv_ndx_1 + (jp_1 >> 3)]->block[ip_1_lsbs][jp_1 & 7];
          cr = MCU_buff[inv_ndx_2 + (jp_2 >> 3)]->block[ip_2_lsbs][jp_2 & 7];

          rcb = cb - 128;
          rcr = cr - 128;

          r = y + ((359 * rcr) >> 8);
          g = y - ((11 * rcb) >> 5) - ((183 * rcr) >> 8);
          b = y + ((227 * rcb) >> 7);

          offset = 3 * (i_times_MCU_sx + j);
          ColorBuffer[offset + 2] = Saturate(r);
          ColorBuffer[offset + 1] = Saturate(g);
          ColorBuffer[offset + 0] = Saturate(b);
          /* note that this is SunRaster color ordering */
        }
    }
}
```

6.7.3 utils.c

```
#include <stdlib.h>
#include <stdio.h>
#include <ether_reg.h>
#include <jpeg.h>


/* Prints a data block in frequency space. */
void
show_FBlock(FBlock *S)
{
  int i,j;

  for (i=0; i<8; i++) {
    for (j=0; j<8; j++)
      fprintf(stderr, "\t%d", S->block[i][j]);
    fprintf(stderr, "\n");
  }
}
```

```
/* Prints a data block in pixel space. */
void
show_PBlock(PBlock *S)
{
  int i,j;

  for (i=0; i<8; i++) {
    for (j=0; j<8; j++)
      fprintf(stderr, "\t%d", S->block[i][j]);
    fprintf(stderr, "\n");
  }
}

/* Prints the next 800 bits read from file `fi'. */
/* The following are never used in the code */
/*
void
bin_dump(FILE *fi)
{
  int i;

  for (i=0; i<100; i++) {
    unsigned int bitmask;
    int c = fgetc(fi);

    for (bitmask = 0x80; bitmask; bitmask >>= 1)
      fprintf(stderr, "\t%1d", !!(c & bitmask));
    fprintf(stderr, "\n");
  }
}
*/


/*-----------------------------------------*/
/* core dump generator for forensic analysis */
/*-----------------------------------------*/

/*
void
suicide(void)
{
  int *P;

  fflush(stdout);
  fflush(stderr);
  P = NULL;
  *P = 1;
}
*/

/*-----------------------------------------*/

/*
void
aborted_stream(FILE *fi, FILE *fo)
{
  fprintf(stderr, "%ld:\tERROR:\tAbnormal end of decompression process!\n",
         ftell(fi));
  fprintf(stderr, "\tINFO:\tTotal skipped bytes %d, total stuffers %d\n",
         passed, stuffers);

  fclose(fi); */

  /* The following is not needed as it stores the incomplete image in raster format */
  /*  if (DEBUG) RGB_save(fo); */ /*else free_structures();*/

 /* fclose(fo);

  if (DEBUG) suicide(); else exit(1);
}
*/

/*---------------------------------------------------------*/

/* Returns ceil(N/D). */
int
ceil_div(int N, int D)
{
  int i = N/D;

  if (N > D*i) i++;
```

```
  return i;
}


/* Returns floor(N/D). */
int
floor_div(int N, int D)
{
  int i = N/D;

  if (N < D*i) i--;
  return i;
}


/*-------------------------------------------------------------*/

/* For all components reset DC prediction value to 0. */
void
reset_prediction(void)
{
  int i;

  for (i=0; i<3; i++)
    comp[i].PRED = 0;
}


/*-------------------------------------------------------------*/


/* Transform JPEG number format into usual 2's-complement format. */
int
reformat(unsigned long S, int good)
{
  int St;

  if (!good)
    return 0;
  St = 1 << (good-1);  /* 2^(good-1) */
  if (S < (unsigned long) St)
    return (S+1+((-1) << good));
  else
    return S;
}


/*-------------------------------------------------------------*/


/*
void
free_structures(void)
{
  int i;
*/
  /*  for (i=0; i<4; i++) if (QTvalid[i]) free(QTable[i]);*/

  /* free(ColorBuffer)
  free(FrameBuffer) */

      /* for (i=0; MCU_valid[i] != -1; i++) free(MCU_buff[i]);
  }*/



/*-----------------------------------------*/
/* this is to save final RGB image to disk   */
/* using the sunraster uncompressed format   */
/*-----------------------------------------*/

/* Sun raster header */

typedef struct {
  unsigned long       MAGIC;
  unsigned long       Width;
  unsigned long       Heigth;
  unsigned long       Depth;
  unsigned long       Length;
  unsigned long       Type;
  unsigned long       CMapType;
```

```
  unsigned long        CMapLength;
} sunraster;


void
RGB_save(FILE *fo)
{
  sunraster *FrameHeader;
  sunraster newFrameHeader;
  int i;
  int j;
  unsigned long bigendian_value;
  int row_init = 100;
  unsigned char pr, pg, pb;
  Xuint32 a, pixel;

  FrameHeader = &newFrameHeader;
  /*  FrameHeader = (sunraster *) malloc(sizeof(sunraster)); */
  FrameHeader->MAGIC      = 0x59a66a95L;
  FrameHeader->Width      = 2 * ceil_div(x_size, 2); /* round to 2 more */
  FrameHeader->Heigth     = y_size;
  FrameHeader->Depth      = (n_comp>1) ? 24 : 8;
  FrameHeader->Length     = 0;        /* not required in v1.0 */
  FrameHeader->Type       = 0;        /* old one */
  FrameHeader->CMapType   = 0;        /* truecolor */
  FrameHeader->CMapLength = 0;        /* none */

  /* Frameheader must be in Big-Endian format */
#if BYTE_ORDER == LITTLE_ENDIAN

#define MACHINE_2_BIGENDIAN(value)\
 ((( (value) & (unsigned long)(0x000000FF)) << 24) | \
  (( (value) & (unsigned long)(0x0000FF00)) << 8) | \
  (( (value) & (unsigned long)(0x00FF0000)) >> 8) | \
  (( (value) & (unsigned long)(0xFF000000)) >> 24))

  bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->MAGIC);
  fwrite(&bigendian_value, 4, 1, fo);

  bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->Width);
  fwrite(&bigendian_value, 4, 1, fo);

  bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->Heigth);
  fwrite(&bigendian_value, 4, 1, fo);

  bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->Depth);
  fwrite(&bigendian_value, 4, 1, fo);

  bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->Length);
  fwrite(&bigendian_value, 4, 1, fo);

  bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->Type);
  fwrite(&bigendian_value, 4, 1, fo);

  bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->CMapType);
  fwrite(&bigendian_value, 4, 1, fo);

  bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->CMapLength);
  fwrite(&bigendian_value, 4, 1, fo);

#else
  fwrite(FrameHeader, sizeof(sunraster), 1, fo);
#endif

  print("Entered into RGB_save\r\n");
  print("y_size is = "); putnum(y_size); print("\r\n");
  print("x_size is = "); putnum(x_size); print("\r\n");

/* Commenting out any writes to an actual file
  for (i=0; i<y_size; i++) {
    fwrite(FrameBuffer+n_comp*i*x_size, n_comp, FrameHeader->Width, fo);
}
*/



/*a is the pixel location
for(a=0; a<640*480; a++)
    XIo_Out8(VGA_START + a,0);
```

```
*/


/*
for (i=0; i<20; i++)
{
  for (j=0; j<x_size; j++) {
    pixel = (int)FrameBuffer+n_comp*i*j;
    print("Line number = "); putnum(i); print("\r\n");
    print("\t"); putnum((pixel%134541312)/16);
}
}
*/


/*
  for(a=100; a<200; a++){
    XIo_Out8(VGA_START + 100 + 640*a , 0x6f);
    XIo_Out8(VGA_START + 200 + 640*a , 0xff);
    XIo_Out8(VGA_START + a + 640*100 , 0xff);
    XIo_Out8(VGA_START + a + 640*200 , 0xff);
  }

*/
/*a is the pixel location
for(a=0; a<640*480; a++)
    XIo_Out8(VGA_START + a,0x00);
*/

for (i=0; i<y_size; i++)
{
  for (j=0; j<x_size; j++) {
    pb = FrameBuffer[(i*x_size + j)*n_comp];
    pg = FrameBuffer[(i*x_size + j)*n_comp+1];
    pr = FrameBuffer[(i*x_size + j)*n_comp+2];

    pixel = (pr&0xE0) | ((pg&0xE0) >> 3) | ((pb&0xC0) >> 6);

    XIo_Out8(VGA_START + 100 + j + 640*(row_init+i), pixel);
  }
}



}
```

6.7.4 fast_int_idct.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>

#include "jpeg.h"

#define Y(i,j)         Y[8*i+j]
#define X(i,j)         (output->block[i][j])

/* This version is IEEE compliant using 16-bit arithmetic. */

/* The number of bits coefficients are scaled up before 2-D IDCT: */
#define S_BITS          3
/* The number of bits in the fractional part of a fixed point constant: */
#define C_BITS         14

#define SCALE(x,n)     ((x) << (n))

/* This version is vital in passing overall mean error test. */
#define DESCALE(x, n)  (((x) + (1 << ((n)-1)) - ((x) < 0)) >> (n))

#define ADD(x, y)      ((x) + (y))
#define SUB(x, y)      ((x) - (y))
#define CMUL(C, x)     (((C) * (x) + (1 << (C_BITS-1))) >> C_BITS)

/* Butterfly: but(a,b,x,y) = rot(sqrt(2),4,a,b,x,y) */
#define but(a,b,x,y)   { x = SUB(a,b); y = ADD(a,b); }

/* Inverse 1-D Discrete Cosine Transform.
```

```
   Result Y is scaled up by factor sqrt(8).
   Original Loeffler algorithm.
*/
static void
idct_1d(int *Y)
{
  int z1[8], z2[8], z3[8];

  /* Stage 1: */
  but(Y[0], Y[4], z1[1], z1[0]);
  /* rot(sqrt(2), 6, Y[2], Y[6], &z1[2], &z1[3]); */
  z1[2] = SUB(CMUL( 8867, Y[2]), CMUL(21407, Y[6]));
  z1[3] = ADD(CMUL(21407, Y[2]), CMUL( 8867, Y[6]));
  but(Y[1], Y[7], z1[4], z1[7]);
  /* z1[5] = CMUL(sqrt(2), Y[3]);
     z1[6] = CMUL(sqrt(2), Y[5]);
  */
  z1[5] = CMUL(23170, Y[3]);
  z1[6] = CMUL(23170, Y[5]);

  /* Stage 2: */
  but(z1[0], z1[3], z2[3], z2[0]);
  but(z1[1], z1[2], z2[2], z2[1]);
  but(z1[4], z1[6], z2[6], z2[4]);
  but(z1[7], z1[5], z2[5], z2[7]);

  /* Stage 3: */
  z3[0] = z2[0];
  z3[1] = z2[1];
  z3[2] = z2[2];
  z3[3] = z2[3];
  /* rot(1, 3, z2[4], z2[7], &z3[4], &z3[7]); */
  z3[4] = SUB(CMUL(13623, z2[4]), CMUL( 9102, z2[7]));
  z3[7] = ADD(CMUL( 9102, z2[4]), CMUL(13623, z2[7]));
  /* rot(1, 1, z2[5], z2[6], &z3[5], &z3[6]); */
  z3[5] = SUB(CMUL(16069, z2[5]), CMUL( 3196, z2[6]));
  z3[6] = ADD(CMUL( 3196, z2[5]), CMUL(16069, z2[6]));

  /* Final stage 4: */
  but(z3[0], z3[7], Y[7], Y[0]);
  but(z3[1], z3[6], Y[6], Y[1]);
  but(z3[2], z3[5], Y[5], Y[2]);
  but(z3[3], z3[4], Y[4], Y[3]);
}

/* Inverse 2-D Discrete Cosine Transform. */
void
IDCT(const FBlock *input, PBlock *output)
{
  int Y[64];
  int k,l;

  /* Pass 1: process rows. */
  for (k = 0; k < 8; k++) {

    /* Prescale k-th row: */
    for (l = 0; l < 8; l++)
      Y(k,l) = SCALE(input->block[k][l], S_BITS);

    /* 1-D IDCT on k-th row: */
    idct_1d(&Y(k,0));
    /* Result Y is scaled up by factor sqrt(8)*2^S_BITS. */
  }

  /* Pass 2: process columns. */
  for (l = 0; l < 8; l++) {
    int Yc[8];

    for (k = 0; k < 8; k++) Yc[k] = Y(k,l);
    /* 1-D IDCT on l-th column: */
    idct_1d(Yc);
    /* Result is once more scaled up by a factor sqrt(8). */
    for (k = 0; k < 8; k++) {
      int r = 128 + DESCALE(Yc[k], S_BITS+3); /* includes level shift */

      /* Clip to 8 bits unsigned: */
      r = r > 0 ? (r < 255 ? r : 255) : 0;
      X(k,l) = r;
    }
  }
}
```

6.7.5 parse.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "jpeg.h"


/*--------------------------------------------------------------------------*/


/* utility and counter to return the number of bits from file */
/* right aligned, masked, first bit towards MSB's           */

static unsigned char bit_count;      /* available bits in the window */
static unsigned char window;

unsigned long
get_bits(unsigned char **vfile_ptr, int *vfile_size_ptr, int number)
{
  int i, newbit;
  unsigned long result = 0;
  unsigned char aux, wwindow;

  if (!number)
    return 0;

  for (i = 0; i < number; i++) {
    if (bit_count == 0) {
      /*      wwindow = fgetc(fi); */
      wwindow = **vfile_ptr;
      (*vfile_ptr)++;
      (*vfile_size_ptr)++;

      if (wwindow == 0xFF) {
        aux = **vfile_ptr;
        (*vfile_ptr)++;
        (*vfile_size_ptr)++;

        switch (aux) { /* skip stuffer 0 byte */
        case EOF:
        case 0xFF:
          /*      fprintf(stderr, "%ld:\tERROR:\tRan out of bit stream\n", count-
(*vfile_size_ptr)); */
          exit(1);
          /* aborted_stream(fi, fo); */
          break;

        case 0x00:
          stuffers++;
          break;

        default:
          if (RST_MK(0xFF00 | aux))
            /*      fprintf(stderr, "%ld:\tERROR:\tSpontaneously found restart!\n",
                    count-(*vfile_size_ptr));
          fprintf(stderr, "%ld:\tERROR:\tLost sync in bit stream\n",
          count-(*vfile_size_ptr));  */
          exit(1);
          /* aborted_stream(fi, fo); */
          break;
        }
      }

      bit_count = 8;
    }
    else wwindow = window;
    newbit = (wwindow>>7) & 1;
    window = wwindow << 1;
    bit_count--;
    result = (result << 1) | newbit;
  }
  return result;
}


void
clear_bits(void)
```

```
{
  bit_count = 0;
}


unsigned char
get_one_bit(unsigned char **vfile_ptr, int *vfile_size_ptr)
{
  int newbit;
  unsigned char aux, wwindow;

  if (bit_count == 0) {
    wwindow = **vfile_ptr;
    /*    wwindow = fgetc(fi); */
    (*vfile_ptr)++;
    (*vfile_size_ptr)--;

    if (wwindow == 0xFF) {
      /*  switch (aux = fgetc(fi)) {  */     /* skip stuffer 0 byte */
      aux = **vfile_ptr;
      (*vfile_ptr)++;
      (*vfile_size_ptr)--;
      switch (aux) {   /* skip stuffer 0 byte */
      case EOF:
      case 0xFF:
        /*      fprintf(stderr, "%ld:\tERROR:\tRan out of bit stream\n", count-
(*vfile_size_ptr)); */
        exit(1);
        /* aborted_stream(fi, fo); */
        break;

      case 0x00:
        stuffers++;
        break;

      default:
        if (RST_MK(0xFF00 | aux))
          /*  fprintf(stderr, "%ld:\tERROR:\tSpontaneously found restart!\n",
                  count-(*vfile_size_ptr));
        fprintf(stderr, "%ld:\tERROR:\tLost sync in bit stream\n",
        count-(*vfile_size_ptr)); */
        exit(1);
        /* aborted_stream(fi, fo); */
        break;
      }
    }

    bit_count = 8;
  }
  else
    wwindow = window;

  newbit = (wwindow >> 7) & 1;
  window = wwindow << 1;
  bit_count--;
  return newbit;
}

/*-------------------------------------------------------------*/

/*
unsigned int
get_size(FILE *fi)
{
  unsigned char aux;

  aux = fgetc(fi);
  return (aux << 8) | fgetc(fi);
 }   */


unsigned int
get_size(unsigned char **vfile_ptr, int *vfile_size_ptr)
{
  unsigned char aux;
  unsigned char i;

  aux = **vfile_ptr;
  (*vfile_ptr)++;

  i = **vfile_ptr;
```

```
  (*vfile_ptr)++;

  *vfile_size_ptr = *vfile_size_ptr - 2;

  return (aux << 8) | i;      /* big endian */
}

/*-------------------------------------------------------------*/


void
skip_segment(unsigned char **vfile_ptr, int *vfile_size_ptr)       /* skip a segment we
don't want */
{
  unsigned int size;
  char  tag[5];
  int i;

  size = get_size(vfile_ptr, vfile_size_ptr);
  print("This is the Skip_segment in utils.c\r\n");

  /*  printf("The size in skip_segment  is : %d \n", size); */
    /*  size = get_size(fi);  */
  if (size > 5) {
    for (i = 0; i < 4; i++) {
        tag[i] = **vfile_ptr;
        (*vfile_ptr)++;
        (*vfile_size_ptr)--;
    }
    /* tag[i] = fgetc(fi);*/
    tag[4] = '\0';
    /*if (verbose)
      fprintf(stderr, "\tINFO:\tTag is %s\n", tag);*/
    size -= 4;
  }

  *vfile_ptr = *vfile_ptr+size-2;
  *vfile_size_ptr = *vfile_size_ptr-size+2;
  /*  fseek(fi, size-2, SEEK_CUR); */
}


/*-----------------------------------------------------------------*/
/* find next marker of any type, returns it, positions just after */
/* EOF instead of marker if end of file met while searching ...      */
/*-----------------------------------------------------------------*/

unsigned int
get_next_MK(unsigned char **vfile_ptr, int *vfile_size_ptr)
{
  unsigned char c;
  int ffmet = 0;
  int locpassed = -1;
  int i;

  passed--;    /* as we fetch one anyway */

  /* while ((c = fgetc(fi)) != (unsigned int) EOF) { */

  /* there might be bug where the EOF pointer has to move 1 ahead */

  i = 0;
  while (i != *vfile_size_ptr) {
    print("Got to get_next_MK\r\n");
    c = **vfile_ptr;
    /*    printf("The value of c is : %d \n", c);*/
    i++;
    *vfile_ptr = (*vfile_ptr)+1;
    switch (c) {
    case 0xFF:
      ffmet = 1;
      break;
    case 0x00:
      ffmet = 0;
      /*   printf("The count here is %d \n", *vfile_size_ptr - i);  */
      break;
    default:
      /*
      if (locpassed > 1)
      fprintf(stderr, "NOTE: passed %d bytes\n", locpassed);*/
      if (ffmet) {
```

```
        /*
        printf("Reached here \n");
        printf("The value for vfile_ptr is %d \n", *vfile_ptr);
        */

        *vfile_size_ptr = *vfile_size_ptr - i;    /* size decreases */

        /* printf("The value for vfile_size_ptr is %d \n", *vfile_size_ptr);*/
        return (0xFF00 | c);
       }
       ffmet = 0;
       break;
     }
     locpassed++;
     passed++;
   }

   /* fgetc advances the position indicator for the stream */
   (*vfile_ptr)++;
   /*  printf("EOF is returned");*/
   *vfile_size_ptr = *vfile_size_ptr - i;     /* size decreases */
   return (unsigned int) EOF;
}


/*-------------------------------------------------------------*/
/* loading and allocating of quantization table                */
/* table elements are in ZZ order (same as unpack output)   */
/*-------------------------------------------------------------*/

int
load_quant_tables(unsigned char **vfile_ptr, int *vfile_size_ptr)
{
  char aux;
  unsigned int size, n, i, id, x;

  size = get_size(vfile_ptr, vfile_size_ptr);
  /*  size = get_size(fi);    */  /* this is the tables' size */
  n = (size - 2) / 65;

  for (i = 0; i < n; i++) {
    aux = **vfile_ptr;
    (*vfile_ptr)++;
    (*vfile_size_ptr)--;
    /*    aux = fgetc(fi); */

    if (first_quad(aux) > 0) {
      /*     fprintf(stderr, "\tERROR:\tBad QTable precision!\n");  */
      return -1;
    }
    id = second_quad(aux);
    /*if (verbose)
      fprintf(stderr, "\tINFO:\tLoading table %d\n", id);
    */
    /*   if(id >= 4) printf("Error id >= 4\n"); */
    /*     QTable[id] = (PBlock *) malloc(sizeof(PBlock)); */

    if (QTable[id] == NULL) {
      /*   fprintf(stderr, "\tERROR:\tCould not allocate table storage!\n"); */
      exit(1);
    }
    QTvalid[id] = 1;
    for (x = 0; x < 64; x++) {
      QTable[id]->linear[x] = **vfile_ptr;
      (*vfile_ptr)++;
      (*vfile_size_ptr)--;
    }

    /* QTable[id]->linear[x] = fgetc(fi);*/
      /*
         -- This is useful to print out the table content --
         for (x = 0; x < 64; x++)
         fprintf(stderr, "%d\n", QTable[id]->linear[x]);
      */
  }
  return 0;
}


/*-------------------------------------------------------------*/
/* initialise MCU block descriptors                      */
```

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

int
init_MCU(void)
{
  int i, j, k, n, hmax = 0, vmax = 0;

  for (i = 0; i < 10; i++)
    MCU_valid[i] = -1;

  k = 0;

  for (i = 0; i < n_comp; i++) {
    if (comp[i].HS > hmax)
      hmax = comp[i].HS;
    if (comp[i].VS > vmax)
      vmax = comp[i].VS;
    n = comp[i].HS * comp[i].VS;

    comp[i].IDX = k;
    for (j = 0; j < n; j++) {
      MCU_valid[k] = i;
      /*      if(k>=10) printf("Error k >= 10\n"); */
      /* MCU_buff[k] = (PBlock *) malloc(sizeof(PBlock));*/
      if (MCU_buff[k] == NULL) {
        /*      fprintf(stderr, "\tERROR:\tCould not allocate MCU buffers!\n"); */
        exit(1);
      }
      k++;
      if (k == 10) {
        /*      fprintf(stderr, "\tERROR:\tMax subsampling exceeded!\n"); */
        return -1;
      }
    }
  }

  MCU_sx = 8 * hmax;
  MCU_sy = 8 * vmax;
  for (i = 0; i < n_comp; i++) {
    comp[i].HDIV = (hmax / comp[i].HS > 1);  /* if 1 shift by 0 */
    comp[i].VDIV = (vmax / comp[i].VS > 1);  /* if 2 shift by one */
  }

  mx_size = ceil_div(x_size,MCU_sx);
  my_size = ceil_div(y_size,MCU_sy);
  rx_size = MCU_sx * floor_div(x_size,MCU_sx);
  ry_size = MCU_sy * floor_div(y_size,MCU_sy);

  return 0;
}


/*--------------------------------------------------------------*/
/* this takes care for processing all the blocks in one MCU */
/*--------------------------------------------------------------*/

int
process_MCU(unsigned char **vfile_ptr, int *vfile_size_ptr)
{
  int  i;
  long offset;
  int  goodrows, goodcolumns;

  if (MCU_column == mx_size) {
    MCU_column = 0;
    MCU_row++;
    if (MCU_row == my_size) {
      in_frame = 0;
      return 0;
    }

    /*if (verbose)
      fprintf(stderr, "%ld:\tINFO:\tProcessing stripe %d/%d\n", count-(*vfile_size_ptr),
      MCU_row+1, my_size);*/
  }

  for (curcomp = 0; MCU_valid[curcomp] != -1; curcomp++) {
    unpack_block(vfile_ptr, vfile_size_ptr, FBuff, MCU_valid[curcomp]); /* pass index to
HT,QT,pred */
    IDCT(FBuff, MCU_buff[curcomp]);
  }
```

```
  /* YCrCb to RGB color space transform here */
  if (n_comp > 1)
    color_conversion();
  else
    memmove(ColorBuffer, MCU_buff[0], 64);

  /* cut last row/column as needed */
  if ((y_size != ry_size) && (MCU_row == (my_size - 1)))
    goodrows = y_size - ry_size;
  else
    goodrows = MCU_sy;

  if ((x_size != rx_size) && (MCU_column == (mx_size - 1)))
    goodcolumns = x_size - rx_size;
  else
    goodcolumns = MCU_sx;

  offset = n_comp * (MCU_row * MCU_sy * x_size + MCU_column * MCU_sx);

  for (i = 0; i < goodrows; i++)
    memmove(FrameBuffer + offset + n_comp * i * x_size,
            ColorBuffer + n_comp * i * MCU_sx,
            n_comp * goodcolumns);

  MCU_column++;
  return 1;
}
```

6.7.6 huffman.c

```
#include <stdlib.h>
#include <stdio.h>
#include "jpeg.h"


/*---------------------------------------*/
/* private huffman.c defines and macros */
/*---------------------------------------*/

#define HUFF_EOB              0x00
#define HUFF_ZRL              0xF0

/*-----------------------------------------*/
/* some constants for on-the-fly IQ and IZZ */
/*-----------------------------------------*/

static const int G_ZZ[] = {
   0,  1,  8, 16,  9,  2,  3, 10,
  17, 24, 32, 25, 18, 11,  4,  5,
  12, 19, 26, 33, 40, 48, 41, 34,
  27, 20, 13,  6,  7, 14, 21, 28,
  35, 42, 49, 56, 57, 50, 43, 36,
  29, 22, 15, 23, 30, 37, 44, 51,
  58, 59, 52, 45, 38, 31, 39, 46,
  53, 60, 61, 54, 47, 55, 62, 63
};

/*-------------------------------------------------*/
/* here we unpack, predict, unquantify and reorder */
/* a complete 8*8 DCT block ...                    */
/*-------------------------------------------------*/

void
unpack_block(unsigned char **vfile_ptr, int *vfile_size_ptr, FBlock *T, int select)
{
  unsigned int i, run, cat;
  int value;
  unsigned char       symbol;

  /* Init the block with 0's: */
  for (i=0; i<64; i++) T->linear[i] = 0;

  /* First get the DC coefficient: */
  symbol = get_symbol(vfile_ptr, vfile_size_ptr, HUFF_ID(DC_CLASS, comp[select].DC_HT));
  value = reformat(get_bits(vfile_ptr, vfile_size_ptr, symbol), symbol);

#ifdef SPY
```

```
  trace_bits(symbol, 1);
#endif

  value += comp[select].PRED;
  comp[select].PRED = value;
  T->linear[0] = value * QTable[comp[select].QT]->linear[0];

  /* Now get all 63 AC values: */
  for (i=1; i<64; i++) {
    symbol = get_symbol(vfile_ptr, vfile_size_ptr,  HUFF_ID(AC_CLASS,
comp[select].AC_HT));
    if (symbol == HUFF_EOB) break;
    if (symbol == HUFF_ZRL) { i += 15; continue; }
    cat = symbol & 0x0F;
    run = (symbol>>4) & 0x0F;
    i += run;
    value = reformat(get_bits(vfile_ptr, vfile_size_ptr, cat), cat);

#ifdef SPY
    trace_bits(cat, 1);
#endif

    /* Dequantify and ZigZag-reorder: */
    T->linear[G_ZZ[i]] = value * QTable[comp[select].QT]->linear[i];
  }
}
```

6.7.7 tree_vld.c

```
#include <stdlib.h>
#include <stdio.h>
#include <jpeg.h>


/*---------------------------------------*/
/* private huffman.c defines and macros */
/*---------------------------------------*/

/* Number of HTable words sacrificed to bookkeeping: */
#define GLOB_SIZE             32

/* Memory size of HTables: */
#define MAX_SIZE(hclass)             ((hclass)?384:64)

/* Available cells, top of storage: */
#define MAX_CELLS(hclass)     (MAX_SIZE(hclass) - GLOB_SIZE)

/* for Huffman tree descent */
/* lower 8 bits are for value/left son */

#define GOOD_NODE_FLAG        0x100
#define GOOD_LEAF_FLAG        0x200
#define BAD_LEAF_FLAG         0x300
#define SPECIAL_FLAG          0x000
#define HUFF_FLAG_MSK         0x300

#define HUFF_FLAG(c)          ((c) & HUFF_FLAG_MSK)
#define HUFF_VALUE(c)         ((unsigned char)( (c) & (~HUFF_FLAG_MSK) ))


/*---------------------------------------*/
/* some static structures for storage */
/*---------------------------------------*/

static unsigned int    DC_Table0[MAX_SIZE(DC_CLASS)],
                       DC_Table1[MAX_SIZE(DC_CLASS)];

static unsigned int    AC_Table0[MAX_SIZE(AC_CLASS)],
                       AC_Table1[MAX_SIZE(AC_CLASS)];

static unsigned int    *HTable[4] = {
                                     &DC_Table0[0], &DC_Table1[0],
                                     &AC_Table0[0], &AC_Table1[0]
                                   };


/*-----------------------------------------------------------*/
/* Loading of Huffman table, with leaves drop ability          */
```

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

int load_huff_tables(unsigned char **vfile_ptr, int *vfile_size_ptr)
{
  char aux;
  int size, hclass, id;
  int LeavesN, NodesN, CellsN;
  int MaxDepth, i, k, done;
  int NextCellPt;      /* where shall we put next cell */
  int NextLevelPt;     /* where shall node point to */
  unsigned int flag;

  size = get_size(vfile_ptr, vfile_size_ptr); /* this is the tables' size */

  size -= 2;

  while (size>0) {

    /* aux = fgetc(fi); */
    aux = **vfile_ptr;
    (*vfile_ptr)++;
    (*vfile_size_ptr)--;

    hclass = first_quad(aux); /* AC or DC */
    id = second_quad(aux);    /* table no */
    if (id>1) {
      fprintf(stderr, "\tERROR:\tBad HTable identity %d!\n",id);
      return -1;
    }
    id = HUFF_ID(hclass, id);
    /*if (verbose)
      fprintf(stderr, "\tINFO:\tLoading Table %d\n", id);*/
    size--;
    CellsN = NodesN = 1;      /* the root one */
    LeavesN = 0;

    for (i=0; i<MAX_CELLS(hclass); i++)
      HTable[id][i] = SPECIAL_FLAG;   /* secure memory with crash value */

    /* first load the sizes of code elements */
    /* and compute contents of each tree level */
    /* Adress  Content         */
    /* Top             Leaves 0       */
    /* Top-1   Nodes  0       */
    /* ......  .......        */
    /* Top-2k  Leaves k       */
    /* Top-2k-1        Nodes  k       */

    MaxDepth = 0;
    for (i=0; i<16; i++) {
      /*      LeavesN = HTable[id][MAX_SIZE(hclass)-2*i-1] = fgetc(fi); */
      LeavesN = HTable[id][MAX_SIZE(hclass)-2*i-1] = **vfile_ptr;
      (*vfile_ptr)++;
      (*vfile_size_ptr)--;

      CellsN = 2*NodesN; /* nodes is old */
      NodesN = HTable[id][MAX_SIZE(hclass)-2*i-2] = CellsN-LeavesN;
      if (LeavesN) MaxDepth = i;
    }
    size-=16;

    /* build root at address 0, then deeper levels at */
    /* increasing addresses until MAX_CELLS reached */

    HTable[id][0] = 1 | GOOD_NODE_FLAG;             /* points to cell _2_ ! */
    /* we give up address one to keep left brothers on even adresses */
    NextCellPt = 2;
    i = 0;             /* this is actually length 1 */

    done = 0;

    while (i<= MaxDepth) {

      /* then load leaves for other levels */
      LeavesN = HTable[id][MAX_SIZE(hclass)-2*i-1];
      for (k = 0; k<LeavesN; k++)
        if (!done) {
          /*  HTable[id][NextCellPt++] = fgetc(fi) | GOOD_LEAF_FLAG;        */
          HTable[id][NextCellPt++] = **vfile_ptr | GOOD_LEAF_FLAG;
          (*vfile_ptr)++;
          (*vfile_size_ptr)--;
```

```
          if (NextCellPt >= MAX_CELLS(hclass)) {
            done = 1;
            fprintf(stderr, "\tWARNING:\tTruncating Table at depth %d\n", i+1);
          }
        }
      /*       else fgetc(fi); */     /* throw it away, just to keep file sync */
      else {
        (*vfile_ptr)++;
        (*vfile_size_ptr)--;
      }

      size -= LeavesN;

      if (done || (i == MaxDepth)) { i++; continue; }
      /* skip useless node building */

      /* then build nodes at that level */
      NodesN = HTable[id][MAX_SIZE(hclass)-2*i-2];

      NextLevelPt = NextCellPt+NodesN;
      for (k = 0; k<NodesN; k++) {
        if (NextCellPt >= MAX_CELLS(hclass)) { done = 1; break; }

        flag = ((NextLevelPt|1) >=
                MAX_CELLS(hclass)) ? BAD_LEAF_FLAG : GOOD_NODE_FLAG;
        /* we OR by 1 to check even right brother within range */
        HTable[id][NextCellPt++] = (NextLevelPt/2) | flag;
        NextLevelPt += 2;
      }

      i++;     /* now for next level */
    }  /* nothing left to read from file after maxdepth */

    /*if (verbose)
      fprintf(stderr, "\tINFO:\tUsing %d words of table memory\n", NextCellPt);*/

    /*
      -- this is useful for displaying the uploaded tree --
      for(i=0; i<NextCellPt; i++) {
      switch (HUFF_FLAG(HTable[id][i])) {
      case GOOD_NODE_FLAG:
      fprintf(stderr, "Cell %X: Node to %X and %X\n", i,
      HUFF_VALUE(HTable[id][i])*2,
      HUFF_VALUE(HTable[id][i])*2 +1);
      break;
      case GOOD_LEAF_FLAG:
      fprintf(stderr, "Cell %X: Leaf with value %X\n", i,
      HUFF_VALUE(HTable[id][i]) );
      break;
      case SPECIAL_FLAG:
      fprintf(stderr, "Cell %X: Special flag\n", i);
      break;
      case BAD_LEAF_FLAG:
      fprintf(stderr, "Cell %X: Bad leaf\n", i);
      break;
      }
      }
      */

  }     /* loop on tables */
  return 0;
}

/*-----------------------------------*/
/* extract a single symbol from file */
/* using specified huffman table ... */
/*-----------------------------------*/

unsigned char
get_symbol(unsigned char **vfile_ptr, int *vfile_size_ptr, int select)
{
  int cellPt;

  cellPt = 0; /* this is the root cell */

  while (HUFF_FLAG(HTable[select][cellPt]) == GOOD_NODE_FLAG)
    /*    cellPt = get_one_bit(fi) | (HUFF_VALUE(HTable[select][cellPt])<<1); */
    cellPt = get_one_bit(vfile_ptr, vfile_size_ptr) |
(HUFF_VALUE(HTable[select][cellPt])<<1);
```

```
  switch (HUFF_FLAG(HTable[select][cellPt])) {
  case SPECIAL_FLAG:
    fprintf(stderr, "%ld:\tERROR:\tFound forbidden Huffman symbol !\n",
            count-(*vfile_size_ptr));
    exit(1);
    /* aborted_stream(fi, fo); */
    break;

  case GOOD_LEAF_FLAG:
    return HUFF_VALUE(HTable[select][cellPt]);
    break;

  case BAD_LEAF_FLAG:
    /* how do we fall back in case of truncated tree ? */
    /* suggest we send an EOB and warn */
    fprintf(stderr, "%ld:\tWARNING:\tFalling out of truncated tree !\n",
            count-(*vfile_size_ptr));
    return 0;
    break;

  default:
    break;
  }
  return 0;
}
```

6.7.8 main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ether_reg.h>
#include "jpeg.h"


/* real declaration of global variables here */
/* see jpeg.h for more info                  */

cd_t   comp[3];             /* descriptors for 3 components */
PBlock *MCU_buff[10];       /* decoded DCT blocks buffer */
int    MCU_valid[10]; /* components of above MCU blocks */

PBlock *QTable[4];     /* quantization tables */
int    QTvalid[4];


int   x_size,y_size;    /* Video frame size    */
int   rx_size,ry_size;  /* down-rounded Video frame size (integer MCU) */
int   MCU_sx, MCU_sy;   /* MCU size in pixels  */
int   mx_size, my_size;        /* picture size in units of MCUs */
int   n_comp;           /* number of components 1,3 */

unsigned char *ColorBuffer; /* MCU after color conversion */
unsigned char *FrameBuffer; /* complete final RGB image */
FBlock       *FBuff;      /* scratch frequency buffer */
PBlock       *PBuff;      /* scratch pixel buffer */

int     in_frame, curcomp;   /* frame started ? current component ? */
int     MCU_row, MCU_column; /* current position in MCU unit */

FILE    *fi;                  /* input  File stream pointer */
FILE    *fo;                  /* output File stream pointer */

int stuffers = 0;      /* stuff bytes in entropy coded segments */
int passed = 0;               /* bytes passed when searching markers */

int count = 0;         /* stores the number of elements in vfile */
unsigned char FrameBuffer_stat [ X_SIZE * Y_SIZE * N_COMP];
unsigned char ColorBuffer_stat [ MCU_SX * MCU_SY * N_COMP];
unsigned char FBuff_stat [sizeof(FBlock)];
unsigned char PBuff_stat [sizeof(PBlock)];
PBlock QTable_stat[4];
PBlock MCU_buff_stat[10];
/*unsigned char vfile[1883];
 assumes a size of 100000 integers */
unsigned char vfile[] = {255, 216, 255, 224, 0, 16, 74, 70, 73, 70, 0, 1, 1, 0, 0, 1, 0,
1, 0, 0, 255, 219, 0, 67, 0, 8, 6, 6, 7, 6, 5, 8, 7, 7, 7, 9, 9, 8, 10, 12, 20, 13, 12,
11, 11, 12, 25, 18, 19, 15, 20, 29, 26, 31, 30, 29, 26, 28, 28, 32, 36, 46, 39, 32, 34,
```

44, 35, 28, 28, 40, 55, 41, 44, 48, 49, 52, 52, 52, 31, 39, 57, 61, 56, 50, 60, 46, 51,
52, 50, 255, 219, 0, 67, 1, 9, 9, 9, 12, 11, 12, 24, 13, 13, 24, 50, 33, 28, 33, 50, 50,
50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
50, 50, 50, 50, 255, 192, 0, 17, 8, 0, 67, 0, 50, 3, 1, 34, 0, 2, 17, 1, 3, 17, 1, 255,
196, 0, 31, 0, 0, 1, 5, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8,
9, 10, 11, 255, 196, 0, 181, 16, 0, 2, 1, 3, 3, 2, 4, 3, 5, 5, 4, 4, 0, 0, 1, 125, 1, 2,
3, 0, 4, 17, 5, 18, 33, 49, 65, 6, 19, 81, 97, 7, 34, 113, 20, 50, 129, 145, 161, 8, 35,
66, 177, 193, 21, 82, 209, 240, 36, 51, 98, 114, 130, 9, 10, 22, 23, 24, 25, 26, 37, 38,
39, 40, 41, 42, 52, 53, 54, 55, 56, 57, 58, 67, 68, 69, 70, 71, 72, 73, 74, 83, 84, 85,
86, 87, 88, 89, 90, 99, 100, 101, 102, 103, 104, 105, 106, 115, 116, 117, 118, 119, 120,
121, 122, 131, 132, 133, 134, 135, 136, 137, 138, 146, 147, 148, 149, 150, 151, 152, 153,
154, 162, 163, 164, 165, 166, 167, 168, 169, 170, 178, 179, 180, 181, 182, 183, 184, 185,
186, 194, 195, 196, 197, 198, 199, 200, 201, 202, 210, 211, 212, 213, 214, 215, 216, 217,
218, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 241, 242, 243, 244, 245, 246, 247,
248, 249, 250, 255, 196, 0, 31, 1, 0, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1,
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 255, 196, 0, 181, 17, 0, 2, 1, 2, 4, 4, 3, 4, 7, 5, 4, 4,
0, 1, 2, 119, 0, 1, 2, 3, 17, 4, 5, 33, 49, 6, 18, 65, 81, 7, 97, 113, 19, 34, 50, 129,
8, 20, 66, 145, 161, 177, 193, 9, 35, 51, 82, 240, 21, 98, 114, 209, 10, 22, 36, 52, 225,
37, 241, 23, 24, 25, 26, 38, 39, 40, 41, 42, 53, 54, 55, 56, 57, 58, 67, 68, 69, 70, 71,
72, 73, 74, 83, 84, 85, 86, 87, 88, 89, 90, 99, 100, 101, 102, 103, 104, 105, 106, 115,
116, 117, 118, 119, 120, 121, 122, 130, 131, 132, 133, 134, 135, 136, 137, 138, 146, 147,
148, 149, 150, 151, 152, 153, 154, 162, 163, 164, 165, 166, 167, 168, 169, 170, 178, 179,
180, 181, 182, 183, 184, 185, 186, 194, 195, 196, 197, 198, 199, 200, 201, 202, 210, 211,
212, 213, 214, 215, 216, 217, 218, 226, 227, 228, 229, 230, 231, 232, 233, 234, 242, 243,
244, 245, 246, 247, 248, 249, 250, 255, 218, 0, 12, 3, 1, 0, 2, 17, 3, 17, 0, 63, 0, 243,
246, 23, 175, 117, 58, 64, 39, 117, 71, 198, 35, 143, 33, 70, 61, 133, 52, 73, 125, 183,
118, 249, 246, 134, 216, 78, 206, 55, 122, 116, 235, 237, 90, 139, 246, 104, 150, 252,
92, 79, 44, 103, 237, 113, 149, 88, 8, 243, 27, 3, 176, 36, 113, 234, 106, 207, 218, 173,
167, 181, 150, 121, 237, 226, 103, 155, 80, 86, 75, 119, 152, 171, 34, 132, 3, 119, 31,
79, 167, 53, 250, 29, 56, 219, 236, 220, 240, 37, 53, 220, 196, 103, 191, 77, 251, 154,
117, 216, 66, 190, 83, 27, 73, 232, 15, 28, 83, 93, 239, 131, 58, 51, 79, 189, 6, 89,
118, 114, 163, 220, 98, 183, 117, 31, 179, 52, 58, 222, 203, 136, 159, 204, 189, 132,
197, 135, 7, 120, 4, 100, 143, 92, 81, 121, 29, 180, 122, 214, 163, 56, 189, 180, 120,
167, 183, 147, 102, 201, 129, 32, 132, 81, 131, 233, 147, 210, 169, 242, 218, 252, 191,
135, 146, 98, 82, 212, 231, 36, 251, 112, 135, 206, 63, 105, 242, 177, 159, 51, 202, 202,
227, 215, 56, 233, 85, 166, 107, 180, 135, 205, 102, 152, 70, 120, 15, 179, 229, 201,
233, 206, 43, 111, 237, 80, 90, 91, 88, 72, 173, 36, 179, 173, 153, 79, 37, 24, 108, 231,
251, 252, 241, 89, 215, 201, 24, 210, 35, 243, 158, 19, 115, 20, 81, 164, 62, 76, 196,
151, 198, 56, 100, 237, 245, 174, 90, 209, 106, 238, 198, 244, 221, 221, 174, 78, 240,
38, 246, 249, 23, 175, 160, 162, 174, 180, 127, 57, 227, 189, 21, 203, 42, 75, 153, 150,
167, 160, 251, 168, 191, 226, 103, 116, 49, 255, 0, 45, 79, 242, 21, 110, 203, 77, 158,
246, 111, 42, 218, 22, 149, 192, 220, 216, 232, 163, 213, 137, 224, 15, 115, 83, 165,
147, 223, 120, 130, 226, 222, 50, 20, 180, 172, 89, 200, 200, 69, 0, 18, 199, 216, 10,
232, 244, 221, 47, 81, 212, 30, 75, 120, 116, 187, 200, 52, 149, 183, 159, 203, 142, 88,
153, 126, 208, 254, 91, 5, 121, 79, 27, 152, 146, 48, 189, 23, 183, 173, 123, 85, 49, 42,
157, 37, 175, 78, 167, 143, 59, 202, 71, 61, 38, 157, 101, 12, 33, 230, 212, 145, 242,
197, 49, 105, 11, 76, 50, 0, 36, 110, 249, 87, 184, 232, 77, 83, 150, 210, 193, 134, 18,
250, 84, 63, 244, 222, 212, 168, 252, 209, 155, 249, 87, 85, 127, 29, 205, 183, 135, 173,
161, 159, 195, 209, 64, 194, 121, 25, 154, 72, 38, 80, 131, 98, 0, 115, 191, 169, 60,
117, 199, 29, 43, 149, 145, 120, 32, 227, 35, 210, 157, 10, 146, 168, 175, 204, 255, 0,
15, 242, 51, 231, 81, 107, 67, 54, 242, 202, 91, 117, 14, 193, 90, 38, 56, 89, 99, 96,
200, 199, 211, 35, 191, 177, 193, 172, 233, 7, 202, 223, 74, 215, 19, 189, 179, 49, 76,
50, 184, 196, 145, 176, 202, 184, 244, 35, 252, 226, 168, 95, 66, 145, 109, 146, 18, 76,
18, 130, 99, 220, 114, 84, 142, 168, 125, 198, 71, 212, 16, 107, 122, 173, 218, 211, 251,
255, 0, 67, 182, 155, 190, 198, 187, 38, 88, 253, 104, 169, 72, 230, 138, 240, 164, 189,
230, 116, 39, 161, 121, 226, 47, 30, 168, 68, 144, 68, 179, 222, 44, 50, 73, 52, 171, 26,
136, 213, 67, 176, 201, 60, 228, 236, 28, 122, 86, 142, 137, 20, 214, 147, 79, 45, 141,
229, 181, 211, 125, 146, 113, 229, 218, 92, 238, 124, 152, 219, 111, 201, 193, 63, 54,
58, 3, 88, 237, 168, 125, 150, 57, 101, 2, 98, 230, 236, 178, 152, 74, 35, 0, 241, 35,
15, 156, 169, 43, 208, 253, 209, 206, 42, 196, 94, 33, 19, 128, 151, 127, 106, 42, 14,
71, 218, 38, 251, 90, 3, 235, 130, 170, 235, 245, 70, 207, 181, 118, 126, 246, 80, 105,
43, 175, 248, 22, 56, 234, 67, 84, 200, 54, 110, 113, 31, 148, 239, 63, 76, 16, 204, 228,
254, 60, 230, 157, 61, 131, 193, 149, 187, 186, 179, 181, 151, 175, 149, 60, 224, 56,
250, 168, 201, 31, 142, 43, 66, 109, 125, 76, 100, 29, 91, 40, 71, 221, 58, 189, 201, 31,
77, 130, 49, 39, 225, 187, 241, 172, 134, 215, 124, 165, 242, 237, 86, 237, 98, 29, 60,
169, 133, 162, 126, 8, 170, 73, 250, 179, 18, 123, 214, 145, 173, 90, 110, 209, 137, 148,
112, 246, 214, 108, 207, 188, 183, 150, 36, 18, 16, 175, 17, 233, 44, 76, 29, 15, 252, 8,
113, 84, 78, 231, 211, 111, 70, 214, 43, 18, 137, 213, 177, 192, 101, 60, 254, 106, 91,
244, 173, 89, 47, 209, 145, 102, 50, 180, 82, 72, 88, 102, 73, 10, 57, 219, 140, 254,
242, 53, 195, 117, 31, 121, 107, 50, 234, 233, 37, 183, 188, 38, 225, 101, 101, 182, 151,
31, 191, 146, 98, 50, 54, 142, 72, 85, 28, 145, 216, 154, 120, 156, 91, 84, 220, 36, 181,
59, 104, 210, 213, 52, 108, 19, 201, 230, 138, 167, 45, 210, 197, 51, 198, 79, 40, 197,
79, 225, 69, 112, 73, 59, 179, 84, 180, 18, 222, 81, 113, 104, 21, 216, 12, 133, 133,
216, 244, 86, 4, 152, 152, 251, 16, 89, 9, 250, 85, 83, 35, 70, 204, 142, 10, 186, 156,
21, 61, 65, 170, 240, 204, 208, 57, 96, 170, 202, 192, 171, 163, 116, 117, 61, 65, 171,
172, 35, 185, 143, 118, 100, 145, 84, 96, 72, 163, 116, 177, 143, 73, 23, 171, 129, 217,
135, 62, 162, 189, 74, 85, 189, 139, 105, 175, 117, 152, 202, 10, 100, 126, 121, 199,
222, 164, 71, 137, 183, 153, 76, 216, 85, 46, 124, 181, 83, 192, 250, 145, 81, 173, 180,
179, 100, 219, 180, 87, 35, 214, 23, 12, 127, 17, 212, 126, 34, 165, 2, 254, 210, 25,
139, 68, 145, 169, 137, 148, 125, 162, 56, 212, 100, 244, 203, 48, 206, 61, 179, 90, 85,

```
197, 199, 217, 185, 83, 252, 44, 40, 210, 92, 214, 99, 174, 229, 177, 251, 13, 153, 205,
231, 13, 40, 80, 35, 77, 199, 238, 231, 63, 54, 0, 233, 222, 160, 93, 170, 170, 133, 92,
2, 86, 121, 150, 76, 100, 34, 156, 198, 167, 4, 242, 237, 131, 143, 69, 205, 53, 62, 127,
46, 69, 120, 238, 90, 50, 216, 153, 163, 9, 2, 19, 140, 244, 3, 204, 63, 47, 64, 49, 234,
105, 142, 224, 130, 170, 206, 192, 177, 118, 119, 251, 210, 49, 234, 205, 253, 7, 97, 94,
98, 148, 235, 182, 147, 209, 239, 183, 245, 250, 27, 217, 65, 107, 184, 198, 37, 152,
179, 28, 177, 57, 39, 212, 209, 69, 21, 221, 100, 101, 112, 165, 4, 171, 6, 82, 67, 14,
65, 7, 4, 81, 69, 2, 45, 33, 23, 173, 139, 168, 161, 156, 255, 0, 122, 72, 149, 155, 243,
35, 52, 77, 109, 109, 107, 243, 67, 107, 108, 173, 216, 249, 42, 72, 252, 197, 20, 87,
157, 56, 175, 107, 177, 186, 126, 233, 90, 73, 30, 86, 221, 35, 179, 159, 86, 57, 197,
54, 138, 43, 209, 74, 198, 1, 69, 20, 83, 3, 255, 217};
int  vfile_size;
unsigned char *vfile_ptr;     /* Our NEW Input File stream Pointer*/
FILE *ftemp;
/*int vfileout_ptr;    Our NEW Output File stream Pointer*/
/*--------------------------------------------------------------------*/
/*             MAIN           MAIN           MAIN              */
/*--------------------------------------------------------------------*/

int
main(int argc, char* argv[])
{

  char fnam[256];
  char *p;
  unsigned int aux, mark;
  int n_restarts, restart_interval, leftover; /* RST check */
  int i,j;
  unsigned int c;
  // verbose = 0;

 /* if (argc != 2) {
    fprintf(stderr, "Please provide a JPEG file as argument.\n");
    exit(0);
  } */

        print("\r\n\r\n-------------  Welcome to spycam ------------------\r\n\r\n");
        xil_printf("startup \r\n");
        xilnet_eth_init_hw_addr("00:60:08:27:11:7b"); //arbitrary
        xilnet_eth_init_hw_addr_tbl();
        xilnet_ip_init("192.168.0.1");
        diag();


  for(i=0;i<4;i++)
    QTable[i] = &QTable_stat[i];

  for(j=0;j<10;j++)
    MCU_buff[j] = &MCU_buff_stat[j];


  /* Check for presence of .jpg file extension: */
  /*if ((p = strrchr(argv[1], '.')) && !strcmp(p, ".jpg")) */
    /* Indeed such extension; remove it: */
/*    *p = '\0';
  sprintf(fnam, "%s.jpg", argv[1]); */


/*  fi = fopen(fnam,"rb");
  if (fi == NULL) {
    fprintf(stderr, "Could not open input file %s.\n", fnam);
    exit(0);
  } */

/* init vfile from TCP/IP */
/* Dun use fi & fo from now on */
/* Should be using vfile_ptr */




/*  while ((c = fgetc(fi)) != (unsigned int) EOF) {
    vfile[count] = c;
    count++;
  } */

  count = 1883;

  /*  for (i=0; i<count; i++)
      printf("c is : %d \n", vfile[i]);   */
```

```
    /* for (i=0;i<count;i++) {
      printf("%d, ", vfile[i]);
      } */


      print("\r\n\r\nCount = "); putnum(count);
      print("\r\n");

      /*
for (i=0;i<1883;i++) {
      vfile[i] = vfile_int[i];
}
       No good memory hogger */
vfile_size = count;
vfile_ptr = vfile;
print("The size of vfile_size is : ");
putnum(vfile_size);
print("\r\n");

  /* Prepare raster format output file: */

/*
sprintf(fnam, "%s.ras", argv[1]);
fo=fopen(fnam,"wb");
if (fo == NULL) {
  fprintf(stderr, "Could not open output file %s.\n", fnam);
  exit(0);
}
sprintf(fnam, "%s.spy", argv[1]);
*/

/* First find the SOI marker: */
/*  printf("This is vfile_ptr = %d \n", vfile_ptr);*/
aux = get_next_MK(&vfile_ptr, &vfile_size);
/*  printf("This is aux: %d\n", aux);
printf("This is vfile_ptr = %d \n", vfile_ptr);
*/

if (aux != SOI_MK)
  exit(1);
  /*aborted_stream(fi, fo); */


/* no idea
if (verbose)
  fprintf(stderr, "%ld:\tINFO:\tFound the SOI marker!\n", count-vfile_size);
*/
in_frame = 0;
restart_interval = 0;
for (i = 0; i < 4; i++)
  QTvalid[i] = 0;

/* Now process segments as they appear: */

/*    print("The count is now: "); */
  putnum(vfile_size); print("\r\n");


  do {

    mark = get_next_MK(&vfile_ptr, &vfile_size);
    print("Yes... it is done \r\n");
    print("vfile_size is "); putnum(vfile_size); print("\r\n");
    print("Mark is "); putnum(mark); print("\r\n");

     /* for debugging purpose */

    /*      print("Mark is now %d \n", mark);*/
    switch (mark) {

    case SOF_MK:
            print("Entered SOF_MK\r\n");

      /*if (verbose)
        fprintf(stderr, "%ld:\tINFO:\tFound the SOF marker!\n",count-vfile_size);
      */
      in_frame = 1;
      get_size(&vfile_ptr, &vfile_size);    /* header size, don't care */
```

```
        /* load basic image parameters */
        /* fgetc(fi);    */

        /* precision, 8bit, don't care */

        vfile_ptr++;
        vfile_size--;

        y_size = get_size(&vfile_ptr, &vfile_size);
        x_size = get_size(&vfile_ptr, &vfile_size);


        print("y_size is "); putnum(y_size); print("\r\n");
        print("x_size is "); putnum(x_size); print("\r\n");

        /*if (verbose)
          fprintf(stderr, "\tINFO:\tImage size is %d by %d\n", x_size, y_size);
        */
        /*      n_comp = fgetc(fi);  */       /* # of components */

        n_comp = *vfile_ptr;
        vfile_ptr++;
        vfile_size--;

        /*
        if (verbose) {
          fprintf(stderr, "\tINFO:\t");

        switch (n_comp)
          {
          case 1:
            fprintf(stderr, "Monochrome");
            break;
          case 3:
            fprintf(stderr, "Color");
            break;
          default:
            fprintf(stderr, "Not a");
            break;
          }
        fprintf(stderr, " JPEG image!\n");
        }
*/
      for (i = 0; i < n_comp; i++) {
        /* component specifiers */
        /* comp[i].CID = fgetc(fi); */
        comp[i].CID = *vfile_ptr;
        vfile_ptr++;
        vfile_size--;

        /* aux = fgetc(fi); */
        aux = *vfile_ptr;
        vfile_ptr++;
        vfile_size--;

        comp[i].HS = first_quad(aux);
        comp[i].VS = second_quad(aux);

        /* comp[i].QT = fgetc(fi); */
        comp[i].QT = *vfile_ptr;
        vfile_ptr++;
        vfile_size--;
      }
      /*
      if ((n_comp > 1) && verbose)
        fprintf(stderr, "\tINFO:\tColor format is %d:%d:%d, H=%d\n",
                comp[0].HS * comp[0].VS,
                comp[1].HS * comp[1].VS,
                comp[2].HS * comp[2].VS,
                comp[1].HS);
*/
      if (init_MCU() == -1)
        exit(1);
        /* aborted_stream(fi, fo); */

      if ((x_size > X_SIZE) || (y_size > Y_SIZE) || (n_comp > N_COMP)){
        /*      printf("This will not work, recheck your definitions"); */
        break;
      }

      print("Everything is perfect, and DJPEG will Decompress\r\n");
```

```
  FrameBuffer = FrameBuffer_stat;
  ColorBuffer = ColorBuffer_stat;
  FBuff = (FBlock*) FBuff_stat;
  PBuff = (PBlock*) PBuff_stat;
  /* dimension scan buffer for YUV->RGB conversion
    FrameBuffer =
    (unsigned char *) malloc((size_t) x_size * y_size * n_comp);
    ColorBuffer =
    (unsigned char *) malloc( (size_t) MCU_sx * MCU_sy * n_comp);
    FBuff = (FBlock *) malloc(sizeof(FBlock));
    PBuff = (PBlock *) malloc(sizeof(PBlock));*/

  /* Added print statements to check the different Values*/


  print("MCU_sx : "); putnum(MCU_sx); print("\r\n");
  /*
  printf("MCU_sy %d\n", MCU_sy);
  printf("x_size %d\n", x_size);
  printf("y_size %d\n", y_size);
  printf("n_comp %d\n", n_comp);
  printf("FrameBuffer %d\n", (size_t) x_size * y_size * n_comp);
  printf("ColorBuffer %d\n", (size_t) MCU_sx * MCU_sy * n_comp);
  printf("Fbuff %d\n", sizeof(FBlock));
  printf("Pbuff %d\n", sizeof(PBlock));
  printf("Qtable_stat %d\n",sizeof(QTable[4]));
  printf("MCU_buff_stat %d\n",sizeof(MCU_buff_stat[10]));
  */

  if ((FrameBuffer == NULL) || (ColorBuffer == NULL) ||
      (FBuff == NULL) || (PBuff == NULL) ) {
    /*     fprintf(stderr, "\tERROR:\tCould not allocate pixel storage!\n"); */
    exit(1);
  }

  break;


case DHT_MK:
  print("Entered DHT_MK\r\n");
  /*
  if (verbose)
    fprintf(stderr, "%ld:\tINFO:\tDefining Huffman Tables\n", count-vfile_size);
  */
if (load_huff_tables(&vfile_ptr, &vfile_size) == -1)
    exit(1);
    /* aborted_stream(fi, fo); */
     print("Reached the end of DHT_MK.\r\n");
  break;

case DQT_MK:
  print("Entered DQT_MK\r\n");
  /* if (verbose)
    fprintf(stderr,
    "%ld:\tINFO:\tDefining Quantization Tables\n", count-vfile_size);*/
  if (load_quant_tables(&vfile_ptr, &vfile_size) == -1)
    exit(1);
    /* aborted_stream has to be deleted at the end */
    /* aborted_stream(fi, fo); */
  break;


case DRI_MK:
  print("Entered DRI_MK\r\n");

  /*  get_size(fi); */    /* skip size */
  get_size(&vfile_ptr, &vfile_size);

  /*      restart_interval = get_size(fi); */
  restart_interval = get_size(&vfile_ptr, &vfile_size);
  /*if (verbose)
    fprintf(stderr, "%ld:\tINFO:\tDefining Restart Interval %d\n",
    count-vfile_size, restart_interval);*/
  break;



case SOS_MK:              /* lots of things to do here */
  print("Entered SOS_MK\r\n");
  /*if (verbose)
    fprintf(stderr, "%ld:\tINFO:\tFound the SOS marker!\n", count-vfile_size);
```

```
*/
get_size(&vfile_ptr, &vfile_size);
/* get_size(fi); */ /* don't care */

/*      aux = fgetc(fi); */
aux = *vfile_ptr;
vfile_ptr++;
vfile_size--;


if (aux != (unsigned int) n_comp) {
  /*     fprintf(stderr, "\tERROR:\tBad component interleaving!\n"); */
  exit(1);
  /* aborted_stream(fi, fo); */
}

for (i = 0; i < n_comp; i++) {
  /* aux = fgetc(fi); */
  aux = *vfile_ptr;
  vfile_ptr++;
  vfile_size--;

  if (aux != comp[i].CID) {
    /*     fprintf(stderr, "\tERROR:\tBad Component Order!\n"); */
    exit(1);
    /* aborted_stream(fi, fo); */
  }
  /*     aux = fgetc(fi); */
  aux = *vfile_ptr;
  vfile_ptr++;
  vfile_size--;



  comp[i].DC_HT = first_quad(aux);
  comp[i].AC_HT = second_quad(aux);
}
/*  get_size(fi); fgetc(fi);  */       /* skip things */
get_size(&vfile_ptr, &vfile_size);
vfile_ptr++;
vfile_size--;

/* change from here */

MCU_column = 0;
MCU_row = 0;
clear_bits();
reset_prediction();


/* main MCU processing loop here */
if (restart_interval) {
  n_restarts = ceil_div(mx_size * my_size, restart_interval) - 1;
  leftover = mx_size * my_size - n_restarts * restart_interval;
  /* final interval may be incomplete */

  for (i = 0; i < n_restarts; i++) {
    for (j = 0; j < restart_interval; j++)
      process_MCU(&vfile_ptr, &vfile_size);
    /* proc till all EOB met */

    aux = get_next_MK(&vfile_ptr, &vfile_size);
    if (!RST_MK(aux)) {
      /*     fprintf(stderr, "%ld:\tERROR:\tLost Sync after interval!\n",
             count-vfile_size); */
      exit(1);
      /* aborted_stream(fi, fo); */
    }
    /*else if (verbose)
      fprintf(stderr, "%ld:\tINFO:\tFound Restart Marker\n", count-vfile_size);*/

    reset_prediction();
    clear_bits();
  }              /* intra-interval loop */
}
else
  leftover = mx_size * my_size;

/* process till end of row without restarts */
for (i = 0; i < leftover; i++)
  process_MCU(&vfile_ptr, &vfile_size);
```

```
      in_frame = 0;

      break;


    case EOI_MK:
      print("Entered EOI_MK\r\n");
      /* if (verbose)
       fprintf(stderr, "%ld:\tINFO:\tFound the EOI marker!\n", count-vfile_size);
      */
      if (in_frame)
       exit(1);
       /* aborted_stream(fi, fo); */

      /* if (verbose)
       fprintf(stderr, "\tINFO:\tTotal skipped bytes %d, total stuffers %d\n",
       passed, stuffers);*/
      /* fclose(fi); */

      /* look at RGB_save(fo) */

      RGB_save(fo);
      /*      fclose(fo); */
      /* free_structures();*/
#ifdef SPY
      output_stats(fnam);
#endif
      /*  fprintf(stderr, "\nDone.\n"); */
      exit(0);
      break;


    case COM_MK:
      print("Entered COM_MK\r\n");
      /*if (verbose)
       fprintf(stderr, "%ld:\tINFO:\tSkipping comments\n", count-vfile_size);
       skip_segment(&vfile_ptr, &vfile_size);*/
       /* skip_segment(fi); */
      break;

    case EOF:
      print("Entered EOF\r\n");
      /*
      if (verbose)
       fprintf(stderr, "%ld:\tERROR:\tRan out of input data!\n", count-vfile_size);
      */
      exit(1);
      /* aborted_stream(fi, fo); */


      default:
       print("Entered into this default case.\r\n");

       if ((mark & MK_MSK) == APP_MK) {
         /*if (verbose)
           fprintf(stderr, "%ld:\tINFO:\tSkipping application data\n",
           count-vfile_size);*/
         skip_segment(&vfile_ptr, &vfile_size);
         print("The vfile_ptr = "); putnum(vfile_ptr); print("\r\n");
         print("The vfile_size = "); putnum(vfile_size); print("\r\n");

         break;
       }

       if (RST_MK(mark)) {
        reset_prediction();
        break;
       }
       /* if all else has failed ... */
       /*      fprintf(stderr, "%ld:\tWARNING:\tLost Sync outside scan, %d!\n",
              count-vfile_size, mark);  */
        /* aborted_stream has to be deleted at the end */
       /* aborted_stream(fi, fo); */
       exit(1);
       break;
       }
    }
    while (1);
    return 0;
}
```