Embedded Image Capture

CSEE 4840 Final Report – May 2010

Albert Jimenez Alexander Glass Nektarios Tsoutsos
School of Engineering and Applied Science
Columbia University
{alj2110, amg2229, nt2283}@columbia.edu

1 Overview

For our final project, we chose a number of DE2 functions and integrated them. The DE2 supports real time video capture, VGA display, Ethernet transmit and receive, SDRAM memory, SRAM memory, PS2 interface, 7 Segment displays, LEDs, dip switches and many others. Our project leverages all these functions supported by the DE2 to perform hardware based compression of Jpeg images captured via the video input. Our images are displayed in a custom GUI via the VGA display and support two different transfer modes.

In one of the transfer modes, images are transmitted board-to-board via UDP. A user initiates a transfer via the keyboard and an image is sent to another board. The image is saved to the SDRAM of the remote board and when the remote user scrolls through the local image list, they can view the transmitted image. In the second transfer mode, images are transmitted board-to-computer via UDP. The computer runs a server program developed with BSD sockets that waits for file transfers from the board. After a new image is received, the server program writes the image to disk and opens the image with an "Eye of Gnome", an image viewing utility provided with Ubuntu.

The project was significantly challenging to complete and we were only able to confirm the correct functionality of all implementations on the final day of the deadline. Some of the hardest challenges we faced included the following: Trying to fit the jpeg encoder and appropriate memory buffers within the constraints of the device; with additional memory elements our jpeg encoding would have been at the full VGA size of 640x480 instead of the scaled down size we were forced to use. Writing to the SRAM frame buffer was problematic because of the synchronization issues involved, such as determining the start of a new frame and contention for SRAM data during reads and writes. The Ethernet component board-to-computer transfer involved several challenges due to packets being discarded by the BSD socket code and operating system related issues to the Ethernet interface with Ubuntu. Despite seeing our packets with Wireshark, (an open source packet capture tool), our host application refused to acknowledge receiving new packets via the *recvfrom* method.

2 Hardware description

Here we present a top level view of the system:

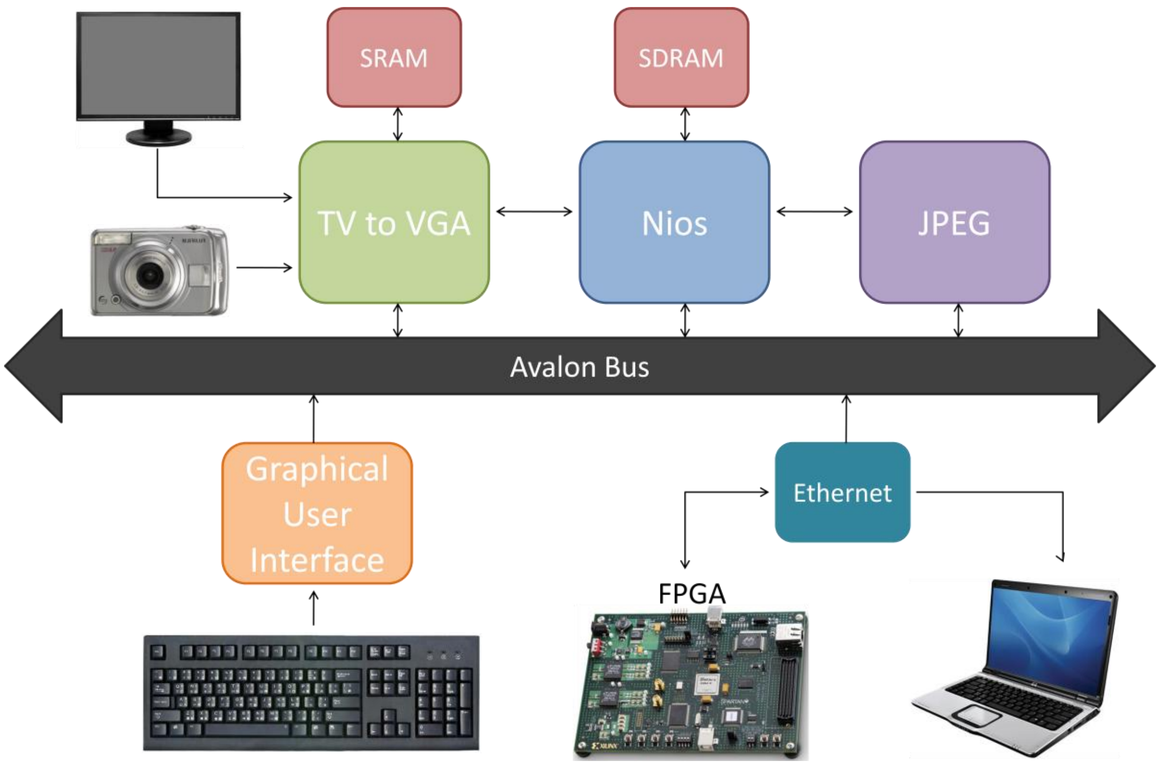


Figure 1 Top level view of the system

2.1 The Video Capture Hardware

The Graphical User Interface

The user interface offers a variety of functionality to capture, store, send and view images. All of the functionality is achieved through a ps2 keyboard that communicates with the processor. The interface has two main display modes, full screen and true resolution. It also allows the user to freeze the live image in either display modes. In order to achieve a still image, pixel information is continually stored in an SRAM, which can store one fourth of a full screen image uncompressed. When a stored image is viewed at full screen, the image is interpolated in order to fill the entire screen. In real-resolution mode, stored images are displayed in the exact resolution in which they are stored.

The user also has the ability to save multiple images on an SDRAM and view them in either full or real-resolution mode. Once images are saved to the SDRAM, they can be sent remotely to either a remote board or a computer.

Keyboard Control Signals

Keyboard Button	Functionality
Spacebar	Freeze image
Up and down keys	Change screen mode
Delete	Request image deletion
Left key	Move to next image
Right key	Move to previous image
Enter	Save image
Right Shift Enter	Send image to remote board
Left Shift Enter	Send compressed image to computer

TV to VGA Module

The TV to VGA module is responsible for converting the TV signal into a format that can be displayed on a VGA. The module is also responsible for storing a local frame buffer on SRAM and communicating with the NIOS processor in order to send and receive images. The Altera DE2_TV tutorial was used as a starting point to convert and manipulate the incoming TV signal.

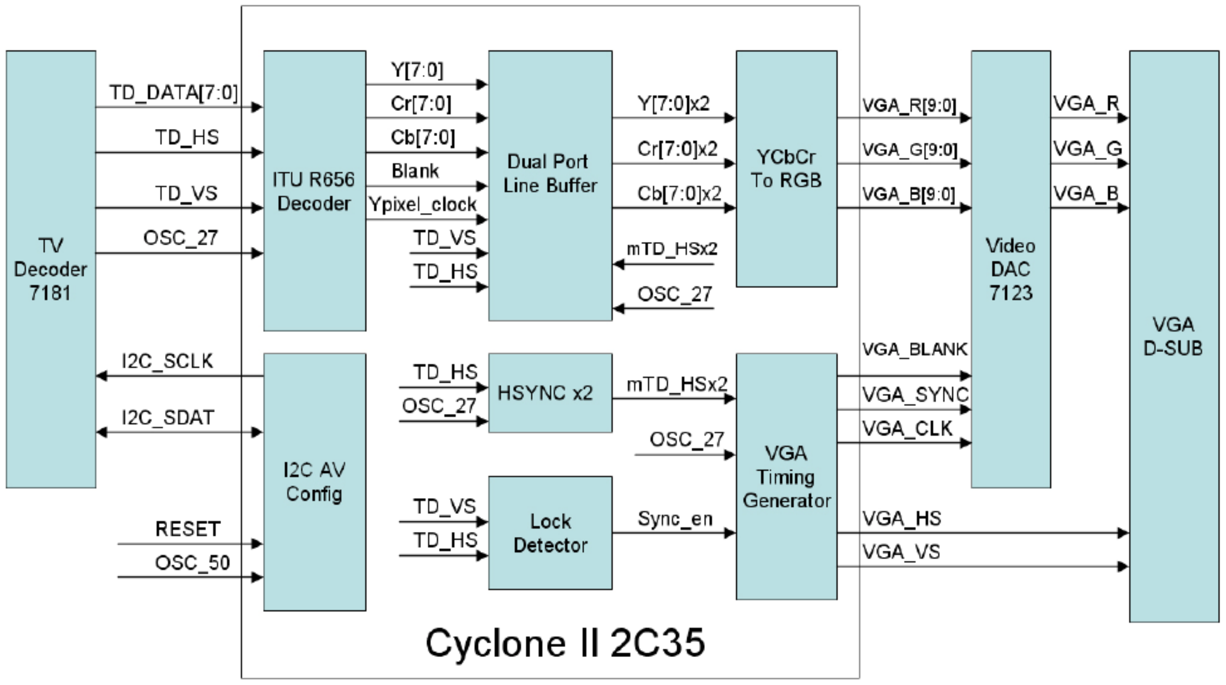


Figure 2 Block diagram of the tv to vga convertor

The initial conversion is achieved by a tv decoder chip (ADV7181), which takes standard tv signal at 50 or 60 hz. This project was tested at 60 hz using the NTSC format. The tv decoder is connected to a I2C AV configuration chip which sets up and synchronizes the decoder. The output of the tv decoder is sent to a ITU R565 chip, which is responsible for down-sampling pixels and extracting color information in YCbCr format. This format describes the luminance and chroma of pixels; however in order for this information to be used by the VGA display it must be converted to standard RGB format, which encodes pixel color information in three color channels. The color conversion is done by the YCbCr to RGB module and the result is sent to digital to analog convertor in order to be used directly by the VGA.

Proper synchronization is achieved by the VGA timing generator, which uses the original tv synchronization signals and provides the necessary horizontal and vertical sync to the VGA. These synchronization signals were also used in order to keep vertical and horizontal counts that indicate exactly where on the screen a pixel is being displayed. Such counts were also essential in mapping the screen raster onto a consistent location in SRAM and sending the correct horizontal and vertical positions when storing and saving images.

Sending From the Frame Buffer to SDRAM

Address Number	**Functionality**
0	Send x position
1	Send y position
2	Send request
3	Receive pixel

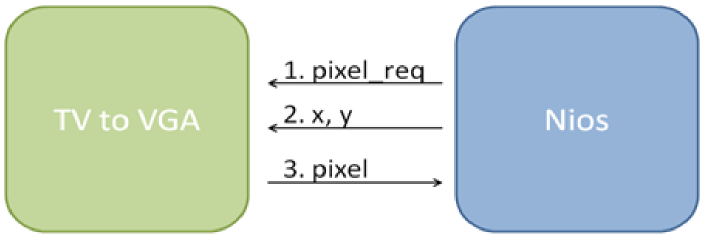


Figure 3 Image exchange protocol from the tv to vga module to the nios

When the user presses enter, the image on the screen is first captured on the SRAM and then sent to be stored on the SDRAM. A strict protocol is maintained between the NIOS processor and the TV to VGA module, to ensure that the image is sent properly. First the processor sends a pixel request signal to the TV to VGA module, along with the x and y position. Once it receives the pixel, the processor stores it in SDRAM and repeats the protocol with new x and y positions until the entire image is received.

Image Data Structure

```
// the structure that holds all saved images
unsigned char saved_images[MAX_NUM_IMAGES][3] [IMG_X_SIZE] [IMG_Y_SIZE];
// data structure for linking images
struct image
{
	struct image *prev;
	int is_taken
	int image_num;
	struct image *next;
};
```

All images are stored in a four dimensional structured called saved_images that can hold up to 30 images. The first index refers to a particularly saved image. The second index represents the color channel (0, 1, 2 correspond to red, green, blue respectively). The last two indexes are used to access a particular pixel given its x and y position.

A separate structure, called image, is used to link the stored images and store information such as the image number and whether the image slot is actually taken. These fields are especially useful when deleting an image; when a delete occurs, the processor simply unlinks the image without having to zero out the image's values.

Sending From the SDRAM to the Frame Buffer

Address Number	Functionality
21	Send x position
22	Send y position
24	Send request
25	Send pixel

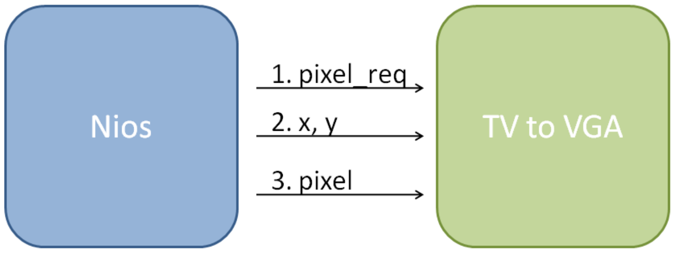


Figure 4 Image exchange protocol from the the nios to the tv to vga module

When the user hits the left or right button to view stored images, the images are retrieved from the SDRAM and sent to the TV to VGA module. A protocol similar to storing images on SDRAM is used. The main difference is that the roles between the processor and the TV to VGA module are reversed. The processor begins by submitting a send request and continues by sending the pixel and its corresponding x and y position. This is repeated until the entire image is sent to the TV to VGA module.

Sending Images to a Remote Board

When the user presses both the right shift and the enter key, the current image is sent to a remote board through an Ethernet connection. The local processor begins by sending a request signal indicating the intention to send an image. When the remote board receives the signal, an interrupt arises and the remote processor enters a function which first reserves memory for the new image and then sends an acknowledgement message. The local board then sends the image, pixel by pixel. The protocol ensures that all packets are received by forcing hand-shaking between the boards; the boards alternatively exchange messages until the entire image is sent. In case that the network connection is broken, the protocol also ensures that neither processor stalls and simply leaves an image partially finished so that both processors can continue functioning.

Header	X_pos	Y_pos	Data_length	Data
[0]	[1:2]	[3:4]	[5:6]	[7:7+Data_length]

Each packet is broken up into 1 byte sections. The first section, the header, indicates the nature of the packet (whether it's a request, an acknowledgement or data packet). Packets containing pixel information store the start x and y values, followed by the number of bytes of pixel information that has been sent. This is followed by up to 800 bytes of pixel information. Since pixels are 16 bits each, each pixel takes up two sections in the packet.

Header Number	Packet Type
0	Ignored
1	Request to send image
2	Acknowledgement of a received pixel
3	Packet contains a pixel

Once the remote board receives a packet, it constructs the image being sent by considering the start x and y values and placing each pixel in the proper location. Once the image transfer is done, the user can view the image along with other images taken locally.

2.2 The JPEG module

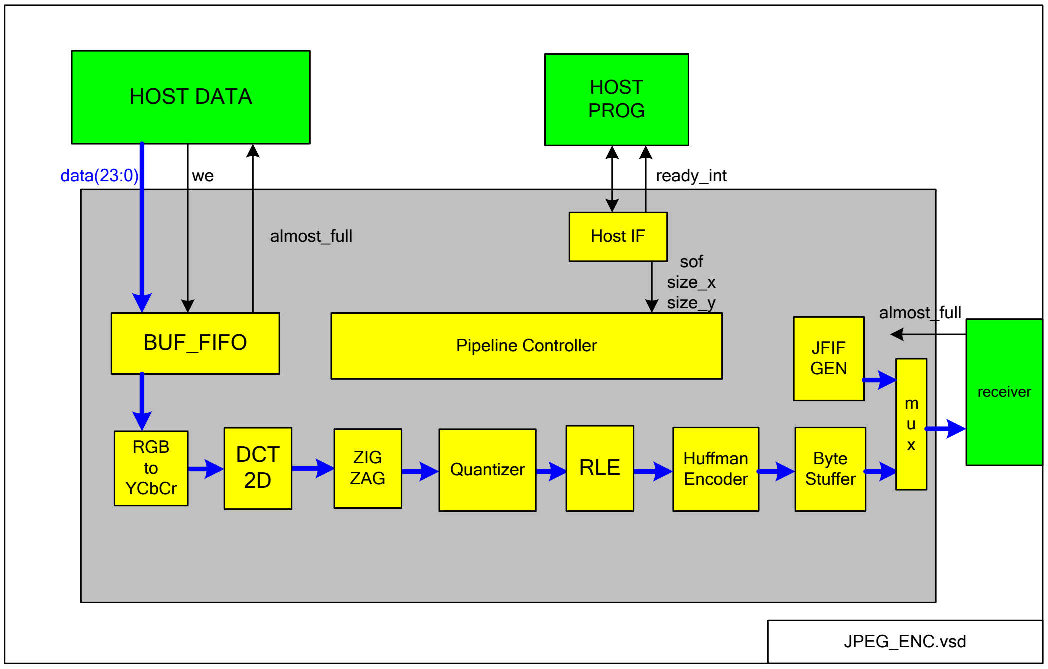


Figure 4 Block diagram of the JPEG module[1]

[1] Source: *mkjpeg* project documentation

One of the core components in this project is the JPEG module. This module implements a JPEG encoder in hardware and generates the contents of a .jpg file along with the corresponding JFIF header. The module that we use in this project is based on the *mkjpeg* module of *Opencores.org*, created by *Michal Krepa*. This module was originally designed for Xilinx FPGAs that use Microblaze and the OPB bus, and it is written in VHDL. The next figure presents a basic block diagram of the original JPEG module.

As shown on the block diagram above, the JPEG module consists of a Host Programming Interface that writes and reads configuration registers and also supplies the quantizer table that controls the compression quality. The raw image data is supplied in raster format to a line buffer along with a write enable signal. The module then saves each pixel in an internal buffer and starts compressing blocks of pixels in a pipelined fashion. In case the internal buffer is almost full, a control signal is returned to indicate that the sender needs to throttle the input. The size of this buffer can be configured, and if the buffer is small, the module has to stall more often, to empty this buffer. If the buffer is configured to be bigger, the module will run faster, however it will require more FPGA area. This is effectively an area-speed tradeoff.

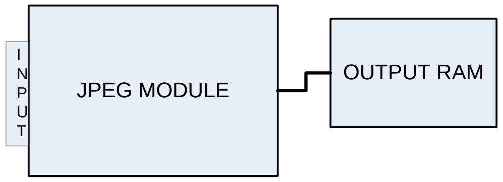


Figure 5 Abstruct Input and Output to JPEG module

For the compressed output, the module supplies data signals, address signals and a write enable signal, so that a memory can be plugged in and accept the data. Inside the JPEG module there are the following subunits:

- Color space converter unit
- Two Dimension Forward Discrete Cosine Transform unit
- Zig Zag scan unit
- Quantizer unit
- Run Length Encoder unit
- Huffman encoder unit
- Byte Stuffer unit
- JFIF header generator unit

The module supports the JPEG ITU-T T.81 compression standard, and a JFIF version 1.01 header format.

Address Space

Here we present the address space of the JPEG module. This includes all the memory locations that can be accessed using the Host Programming Interface.

Address	Functionality	Access
0x0 0000	Start Encoder register	WO
0x0 0004	Image Size Register	R/W
0x0 000C	Encoder Status Register	RO
0x0 0014	Output Length Register	RO
0x00 0100 to 0x00 01FF	Quantizer Values (Luminance)	WO
0x00 0200 to 0x00 02FF	Quantizer Values (Chrominance)	WO

Configuration

The JPEG module is configurable using a top level VHDL package. In this project, in order to save on FPGA area, we trade performance, and we use the smallest input buffer possible. The parameter that sets the input buffer size accepts values from 0 to 8, with 0 corresponding to the smallest area and slowest performance, and 8 corresponding to highest area but fastest performance. We can also set the maximum horizontal size of an image. Since in this project we only transmit images with dimensions 336 by 232 pixels, we configure the maximum line to 336 pixels. This setting also affects the used area in the FPGA, so effectively we also save area by setting the correct size here.

JPEG Module Connections

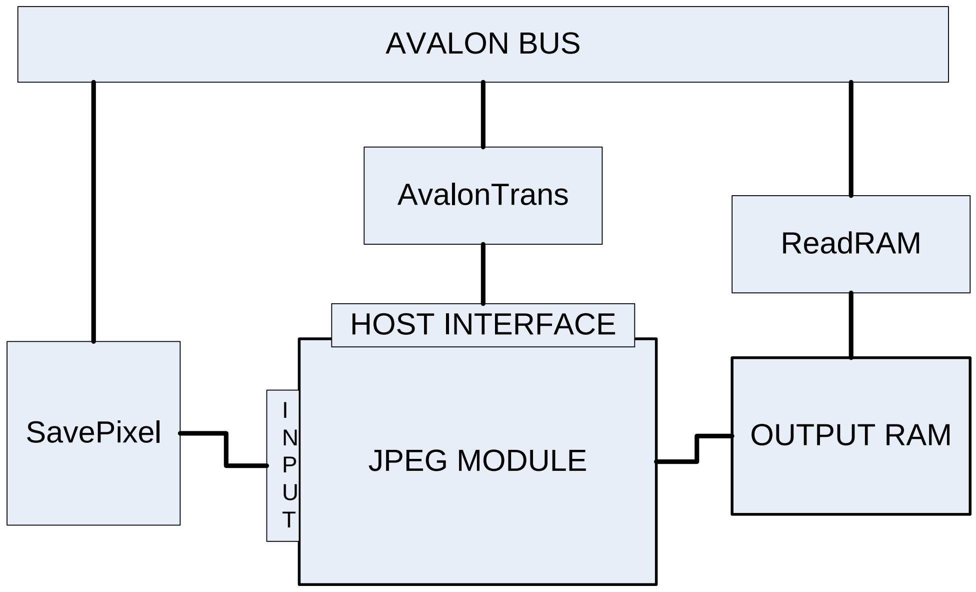


Figure 6 Connections of the JPEG Module

Communication with the JPEG module

The JPEG module described earlier is designed to work on a Xilinx FPGA that supports the OPB bus. In this project we use an Altera FPGA that uses the Avalon Bus. This means that we had to implement an Avalon peripheral (*"AvalonTrans"*) that translates the Avalon signals to OPB signals, respecting the timings that the underlying module expects. The Avalon peripheral generates signals for the Host

Programming Interface of the JPEG module, and at an abstract level allows reading and writing to the JPEG module registers. The Avalon Translation peripheral is implemented in VHDL using continuous assignments, and the correct timings for Read and Write are implemented using the SOPC builder peripheral parameters.

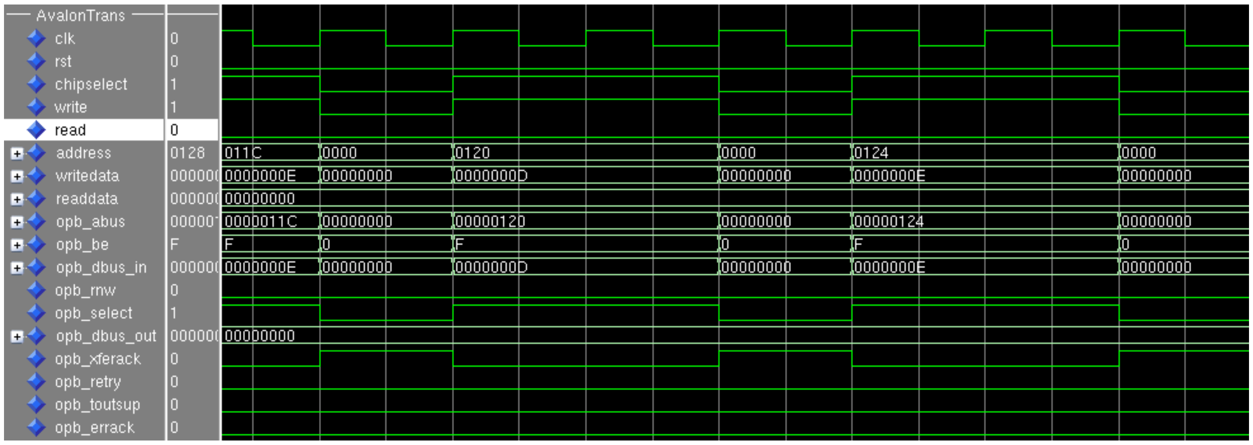

AvalonTrans
clk 0
rst 0
chipselect 1
write 1
read 0
address 0128 011C 0000 0120 0000 0124 0000
writedata 0000000E 00000000 0000000D 00000000 0000000E 00000000
readdata 00000000
opb_abus 0000011C 00000000 00000120 00000000 00000124 00000000
opb_be F F 0 F 0 F 0
opb_dbus_in 0000000E 00000000 0000000D 00000000 0000000E 00000000
opb_rnw 0
opb_select 1
opb_dbus_out 00000000
opb_xferack 0
opb_retry 0
opb_toutsup 0
opb_errack 0

Figure 7 AvalonTrans peripheral waveforms

Output of the JPEG module

For the output of the JPEG module, we use a double ported synchronous RAM of size 10000 by 8 bits. The size of this RAM effectively limits the size of the biggest compressed image the system supports. However, in our system this rarely is a problem, since compressed images are usually smaller than 10000 bytes. The only limiting factor is the total number of M4K elements used in the FPGA, and our experiments showed that despite the fact that the total area is less than the maximum, the number of M4K elements is exceeded if we increase the output RAM above around 10000 memory locations.

The write port of the double ported RAM is connected to the output of the JPEG module. The JPEG module supplies a data signal, an address signal and a write enable signal, and these are the only signals needed to write data to this output RAM. In order to read for the RAM however, we need an Avalon peripheral (*"ReadRAM"*) that supplies the read address to this RAM and then sends the results to a NIOS processor. This peripheral is implemented in VHDL and needs special timing to provide the correct results.

The read address that is supplied from the peripheral to the output RAM is actually the address supplied from NIOS to the peripheral (effectively, the address is copied from the peripheral to the output RAM). The output RAM will have the result ready on the next clock cycle, so the peripheral will have the data ready on the bus after 2 clock cycles. This means that the peripheral should block all outputs on cycle 0 and cycle 1, before supplying the correct result on clock cycle 2 and afterwards. To implement this functionality correctly, we had to modify the Read and Write settings for this peripheral in the SOPC builder. Next we present a timing diagram for this peripheral.

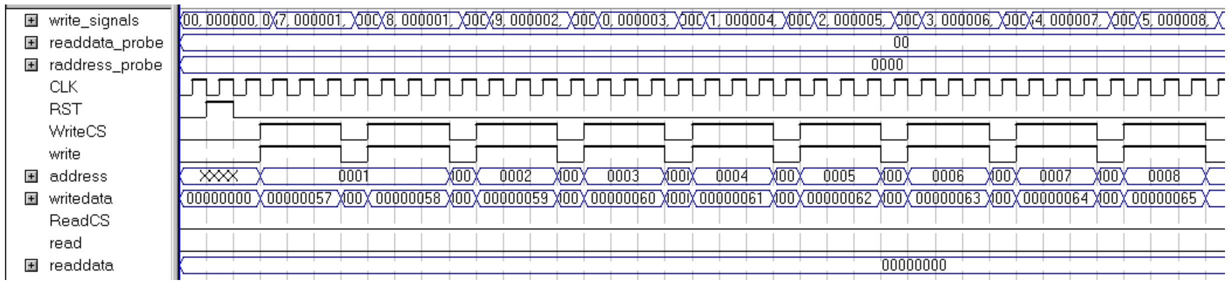


Figure 8 Filling RAM with data manually. Normally this is done by the JPEG module

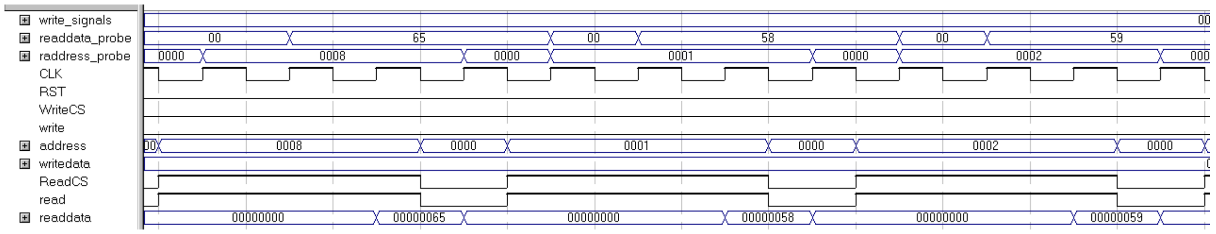


Figure 9 Detail of the ReadRam timing. Data is available after 2 cycles

Sending Pixels to the JPEG Module

In order to send data from the NIOS processor to the line buffer at the input of the JPEG component, we need an Avalon peripheral (*"SavePixel"*). This component has two different modes of operation. In the write mode, the NIOS processor sends data to the peripheral, which in turn generates a 24 bit signal representing an RGB pixel with 8 bits per color. The peripheral also generates the write enable signal that <u>must</u> be asserted for only 1 cycle for every different pixel. In order to do that the peripheral needs to keep state and deassert the write enable after 1 cycle. In order of this to work properly, the Avalon Write signal should return to zero after sending 1 pixel, to reset the state of the peripheral.

The second mode of operation is the read mode. The NIOS processor reads the *"buffer is almost full"* signal that is returned from the JPEG module back to *"SavePixel"* peripheral. The peripheral sends this value back to NIOS, which periodically polls for this signal, in order to throttle the pixel sending process.

Next, we present a timing diagram for this peripheral.

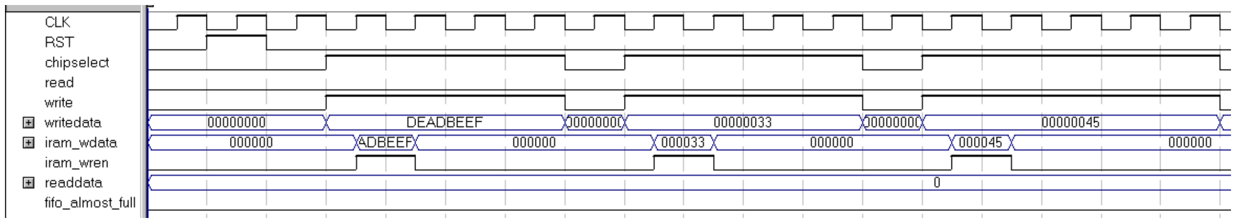


Figure 10 Timing for SavePixel while writing to JPEG input

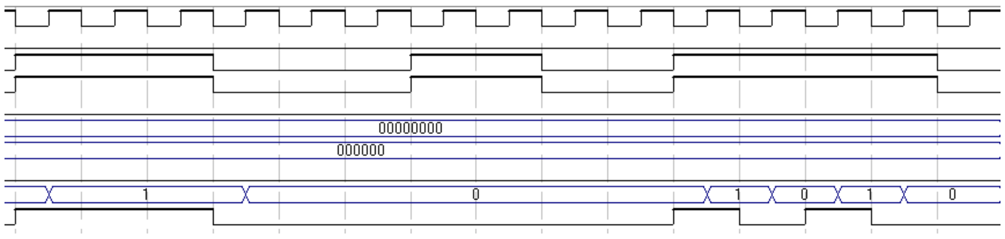


Figure 11 Timing for SavePixel while reading the "buffer almost full" signal

Debugging

One major concern for this part of the project, was how to debug the JPEG module and verify that all peripherals work as expected. In order to do that, we used ModelSim along with a VHDL testbench that simulates the Avalon Signals and also converts the contents of the output memory into a file on the host operating system. The testbench that we used is based on the one provided with *mkjpeg* project on *Opencores.org.* In order to implement the correct operation, we modified the original testbench to use out peripheral that translates the Avalon signals to OPB and also generate Avalon signals (simulating a NIOS processor), instead of OPB signals. Next, we present a screenshots of the used testbench.

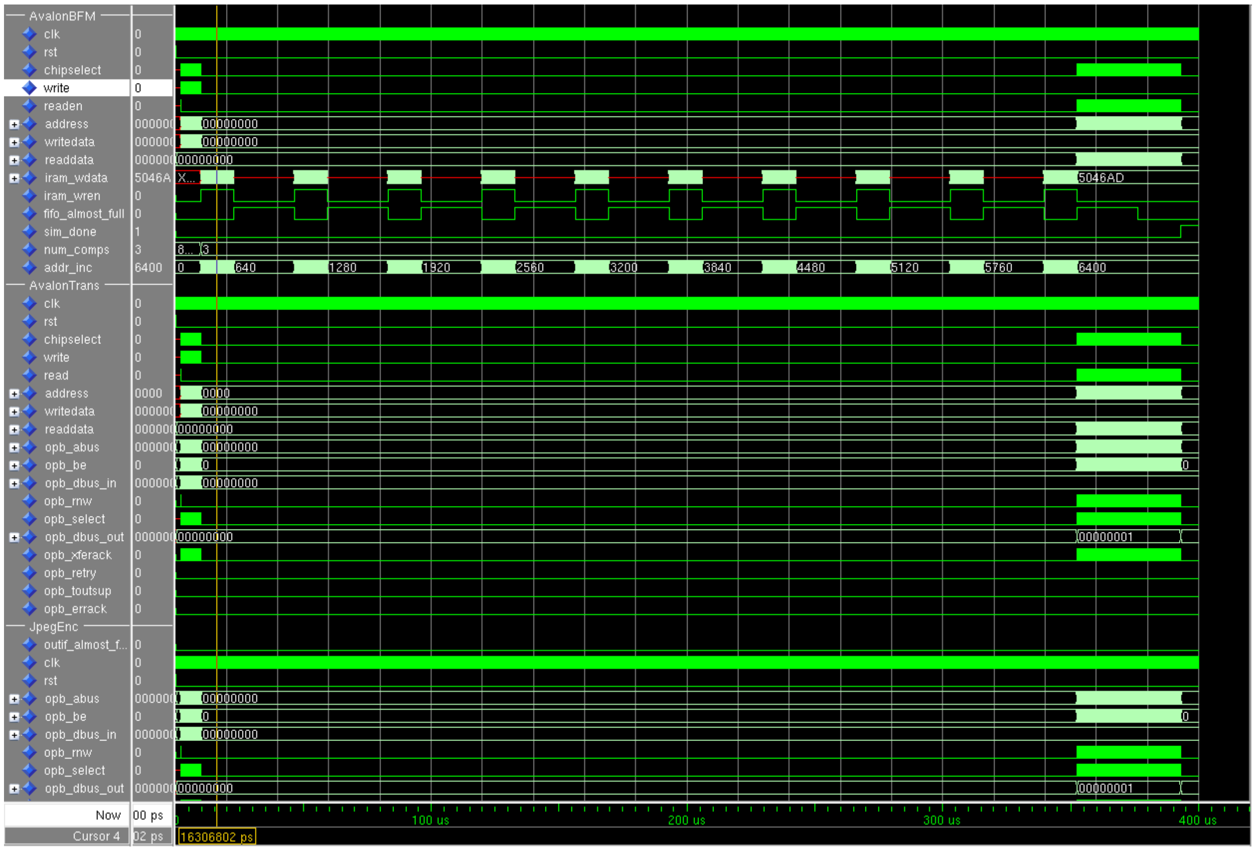


Figure 12 JPEG Module Simulation

As we can see in the timing diagram above, the *iram_wdata* signal is interrupted several times due to the fact that the input buffer is *almost full*.

SOPC Builder Timings

In order to get correct timings we had to modify the Avalon timings in the SOPC builder. Here we present the timings used for the 3 peripherals that the JPEG needs.

	Setup	**Read**	**Write**	**Hold**
AvalonTrans	0	3	2	0
ReadRAM	0	3	1	0
SavePixel	0	3	3	0

2.3 The NIOS hardware

In this project we use the SOPC builder to generate a NIOS II/s CPU. Our experiments showed that this CPU design works better and faster that the simpler NIOS II/e CPU. The NIOS uses the SDRAM memory, the TV_TO_VGA peripheral, the 3 peripherals necessary for communicating with the JPEG, and several peripherals needed for the Ethernet to function.

3 Software description

3.1 The JPEG software

In order to make the jpeg module to work, we used software that asserts read and write commands on the Avalon Bus. Specifically, we used IOWR_32DIRECT and IORD_32DIRECT macros, to perform the following tasks:

- Save both quantizer matrices to the JPEG module
- Save value 7 (=start) to Start Register
- Save the image size to Size Register. The lower 8 bits represent y, and the 8 higher bits represent x.
- Loop though all input pixels and save them to JPEG line buffer. After writing a pixel, poll the *"buffer is full"* signal.
- If buffer is almost full, add artificial delay, by subtracting 1 from a large number in a loop
- Block and poll the status Register, that indicates the encoding is complete
- Read the length of the image from the length register
- Read the output memory byte by byte

3.2 The Ethernet Software: Udpclient & Udpserver

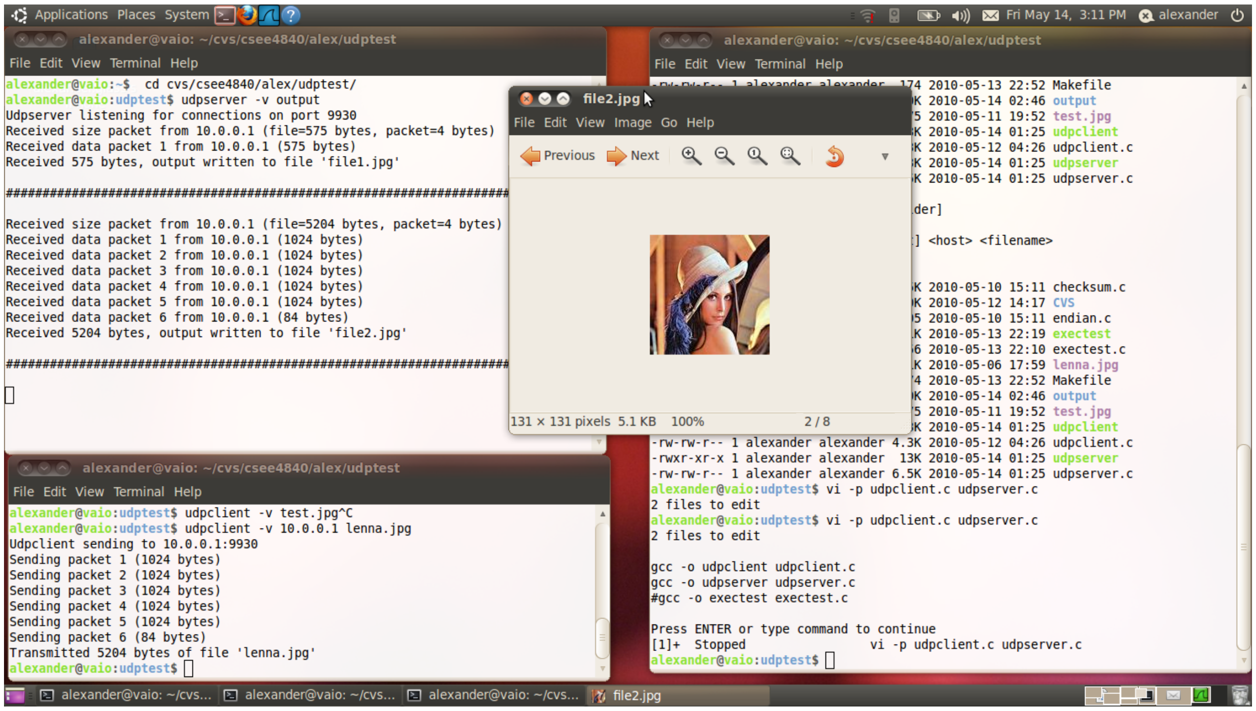
We developed two C programs for testing the file transfer protocol. The Udpserver accepts command line arguments for a port, output directory, and verbose option. The port specifies which port the server listens for connections. The output directory selects via relative path which folder receives new incoming files. The verbose option has 2 modes, verbose and very verbose. In very verbose all packet data is displayed byte by byte. In regular verbose mode, each packet is displayed as a packet number and packet size.

Usage: udpserver [-v | vv] [-p port] [folder]

After setting up a Udpserver to listen for incoming connections on all interfaces, transfers can be initiated via the Udpclient program. The remote computer can transmit via the local area network, Internet or even over wireless connections. While the Udpclient application is not directly used by the application, it was important for testing and developing the file transfer protocol outside the lab where the DE2 was not available.

Usage: udpclient [-v | -vv] [-p port] <host> <filename>

The transfer protocol is described in the following activity diagram. A client initiates a connection with the server by sending a start of data packet which contains a 4 byte payload of zeros. The next packet is a filesize packet, also 4 bytes indicating the amount of data the server should wait to receive. The server then waits to receive the entire file contents, packet by packet and writes them to the disk. After the entire file is received, the server forks a new process and executes the "Eye of Gnome" utility provided with Ubuntu to display the incoming image. If an existing copy of "Eye of Gnome" is open, the previous image is replaced with the current image.


Applications Places System
Fri May 14, 3:11 PM
alexander
alexander@vaio: ~/cvs/csee4840/alex/udptest
File Edit View Terminal Help
file2.jpg
File Edit View Image Go Help
Previous
Next
131 × 131 pixels 5.1 KB 100% 2 / 8
alexander@vaio: ~/cvs...
file2.jpg

```
alexander@vaio:~$  cd cvs/csee4840/alex/udptest/
alexander@vaio:udptest$ udpserver -v output
Udpserver listening for connections on port 9930
Received size packet from 10.0.0.1 (file=575 bytes, packet=4 bytes)
Received data packet 1 from 10.0.0.1 (575 bytes)
Received 575 bytes, output written to file 'file1.jpg'

######################################################################

Received size packet from 10.0.0.1 (file=5204 bytes, packet=4 bytes)
Received data packet 1 from 10.0.0.1 (1024 bytes)
Received data packet 2 from 10.0.0.1 (1024 bytes)
Received data packet 3 from 10.0.0.1 (1024 bytes)
Received data packet 4 from 10.0.0.1 (1024 bytes)
Received data packet 5 from 10.0.0.1 (1024 bytes)
Received data packet 6 from 10.0.0.1 (84 bytes)
Received 5204 bytes, output written to file 'file2.jpg'

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

```
alexander@vaio:udptest$ udpclient -v test.jpg^C
alexander@vaio:udptest$ udpclient -v 10.0.0.1 lenna.jpg
Udpclient sending to 10.0.0.1:9930
Sending packet 1 (1024 bytes)
Sending packet 2 (1024 bytes)
Sending packet 3 (1024 bytes)
Sending packet 4 (1024 bytes)
Sending packet 5 (1024 bytes)
Sending packet 6 (84 bytes)
Transmitted 5204 bytes of file 'lenna.jpg'
alexander@vaio:udptest$
```

```
-rw-rw-r-- 1 alexander alexander  174 2010-05-13 22:52 Makefile
                                    K 2010-05-14 02:46 output
                                    5 2010-05-11 19:52 test.jpg
                                    K 2010-05-14 01:25 udpclient
                                    K 2010-05-12 04:26 udpclient.c
                                    K 2010-05-14 01:25 udpserver
                                    K 2010-05-14 01:25 udpserver.c

                                  der]

                                  ] <host> <filename>


                                    K 2010-05-10 15:11 checksum.c
                                    K 2010-05-12 14:17 CVS
                                    5 2010-05-10 15:11 endian.c
                                    K 2010-05-13 22:19 exectest
                                    6 2010-05-13 22:10 exectest.c
                                    K 2010-05-06 17:59 lenna.jpg
                                    4 2010-05-13 22:52 Makefile
                                    K 2010-05-14 02:46 output
                                    5 2010-05-11 19:52 test.jpg
                                    K 2010-05-14 01:25 udpclient
-rw-rw-r-- 1 alexander alexander 4.3K 2010-05-12 04:26 udpclient.c
-rwxr-xr-x 1 alexander alexander  13K 2010-05-14 01:25 udpserver
-rw-rw-r-- 1 alexander alexander 6.5K 2010-05-14 01:25 udpserver.c
alexander@vaio:udptest$ vi -p udpclient.c udpserver.c
2 files to edit
alexander@vaio:udptest$ vi -p udpclient.c udpserver.c
2 files to edit

gcc -o udpclient udpclient.c
gcc -o udpserver udpserver.c
#gcc -o exectest exectest.c

Press ENTER or type command to continue
[1]+  Stopped                 vi -p udpclient.c udpserver.c
alexander@vaio:udptest$
```

Figure 13 Udpserver software in action

The Udpclient software was ported to the NIOS and the protocol remained the same. However with the NIOS environment, no socket wrappers were available and everything had to be done manually. A network library was developed to automate the entire minutia. This included fragmenting packets, calculating IP and UDP checksums, creating methods to set IP addresses and MAC addresses. We were able to use source code from Lab 2 and augment it to form our network library.

The first version of the NIOS software used the Altera host file system component to read a file from the disk via the Jtag UART USB component. This required running the project in debug mode with the NIOS IDE but proved a vital method for testing while the Jpeg encoder component was still in development. After the Jpeg encoder and GUI component code was ready, it was easy to copy the network libraries into the new project and invoke library methods to send encoded Jpegs without modification.

A number of difficulties significantly slowed down the development process of the Ethernet component for what otherwise could have been a much more sophisticated system. One of the primary difficulties in developing the Ethernet code was the inability to test easily. One persistent annoyance during the development was the NFS configuration in the lab. If the lab computers are disconnected from their Ethernet jacks so as to test them with a direct connection to the DE2, they completely stop responding. Also, the computers in the lab did not have a copy of Wireshark available so it was necessary to bring a laptop. Further complicating things was that our laptops were all running Windows and the BSD sockets code would not natively run in Windows. To do our testing we used an Ubuntu live CD which had its share of drawbacks because it would lose all configuration information every time it was reset. On the days that either the laptop chargers or the live CD was forgotten it was another day lost of development.

The first major obstacle with the Ethernet code was getting the Udpserver to receive packets from the board. Wireshark was reading our packets just fine but the Udpserver was not realizing any data was being sent. We created methods to set source and client IP addresses and MAC addresses and then closely examined the differences between packets generated by the Udpclient program and the NIOS IDE. We noticed the Udpclient was setting the not fragment bit and calculating UDP checksums. We modified our packet generation routine to also set the “don't fragment” bit. This did not have any impact.

Next, we tried to implement the UDP checksum at the advice of some discussions on Internet forums. However, this advice was misguided and the UDP checksum was optional for the recvfrom method so it had no impact on the receipt of the packets. Calculating the UDP checksum turned out to be rather confusing to implement as most of the sample code we found on the Internet was incorrect. In the end we wrote our own method to calculate the UDP checksums. One confusion issue about generating the UDP checksum is that it includes the UDP header as well as a pseudo IP header. There are redundancies in what is contained between the pseudo IP header and the UDP header. Both contain the UDP Length field from the IP header and the Payload length from the UDP header which are the same value.

Unfortunately even after implementing the UDP checksum our packets were still being ignored by the Udpserver. We eventually got data to transfer after increasing the packet payload buffer size from 16 bytes to 1KB. The problem was that our filesize packet was still being ignored even though the data packets were transferring. It turned out that we had forgotten to send the minimum packet size. The TransmitPacket function provided with the Altera board was not creating a minimum packet size. We fixed this by manually checking if the packet met the minimum size and if not padding the packet with zeros.

The first version of our data transfer protocol had synchronization issues due to the implementation with Ubuntu's networking code. Originally we sent only the filesize as a 4 byte packet to signal the start of data. With the board directly connected to the laptop, the Ethernet interface on the laptop eth0 would be in the disabled state until receiving its first packet. If that first packet happened to the filesize packet, the packet would be dropped and instead only the data packets would send. In this case the Udpserver would get out of sync and frequently report an incorrect size such a (filesize=-132429987) which would deadlock. To deal with these problems, a start of data packet was introduced into the protocol so that the Udpserver would not interpret a filesize packet until first observing the start of data packet. With additional development time, a true handshake would have been the desired approach to correctly deal with synchronization issues.

Altera Host Filesystem

During development, we needed a way to load binary data into the NIOS until the Jpeg compression component was completed. We were able to create a NIOS project that used the Host File system preset in the NIOS IDE. In this mode we were restricted to running in debug mode of the IDE but this enabled us read/write access to files on the host filesystem, i.e. files located within the NIOS workspace folder on the desktop machine we were using for development. One of the limitations of this method was the extremely slow transfer speeds for reading and writing. The read speed of this interface topped at about 2.1 KB/s and the write speed was even slower at about 1KB/s. This meant we were limited to testing with images at about 30 KB unless we wanted to wait more than half a minute for the data to load.

SDRAM

During our labs we used the SRAM to run code from the NIOS. In this project the SRAM is being used to store our frame buffer because it has significant speed advantages to the SDRAM. According to the tutorials we found on the course webpage, we were able to add the SDRAM to our project with SOPC builder. The tutorial was for a Quartus 5 and we have Quartus 7 in the lab. In the instructions, they claimed that using the SDRAM without a PLL would trigger error messages when the project started up in the NIOS IDE. However when we tested the SDRAM without the PLL, it appeared to work just fine. A proof of concept program was developed to load a binary image from the disk and store it to the SDRAM and then write the same file back to the console and the disk. We then compared the files to see if they were identical and this was proof that our SDRAM component worked without a PLL.

Unfortunately, later in development we determined that the SDRAM would in fact have problems without the PLL with a more complicated project. The NIOS IDE would refuse to run on the processor and as a result, a PLL was created using the SOPC builder. The PLL provided a clock delayed by 3 NS compared to the processor, which allowed for the correct functioning of the SDRAM.

4. Concluding Remarks

4.1 Advice to future generations

Our first advice to future generations is to start really early with your project! You will soon find out that nothing works as expected. The tools always fail when you need then the most. In our case, we spend probably 80% of our time trying to make things that should obviously work, to actually work. Even on the day of our presentation all lab accounts froze, due to limited disk space on the main server, and we were almost unable to present. So, our advice is to finish as soon as possible all the milestones, and have time to space at the end of the semester.

Also, try to pick a project that is doable in the time that you have, but still be creative. This will always be appreciated. Finally, make sure that you do all 3 labs and understand how the NIOS and Avalon work.

4.2 Distribution of Tasks

Albert got the video capture working, storing the VGA frame to the SRAM developed the User Interface and implemented the board-to-board transfer. Alex build test projects for the Altera host file system, SDRAM test applications, udpclient and server applications, wrote code to generate bitmaps manually, designed the protocol for board to board transfer and helped with debugging the Encoder software. Nektarios researched the Jpeg Encoder module from OpenCores, translated incompatible VHDL constructs such as packages and records, built OPB to Avalon translation modules, debugged the correct operation of the encoder module using the supplied test bench, created wave forms, test benches and Avalon bus functional model. All 3 of us worked on the documentation and presentation of this project.

4.3 Lessons Learned

Assumptions often lead to problems later, make sure to use test benches to verify hardware modules. Murphy's Law can and always will affect your project, take precautions where possible. Recompiling hardware is very slow; form a hypothesis about the source of your problem and be selective in the ways you approach testing it. Build hardware incrementally with extensive testing at each stage. Casting in VHDL makes no sense as there are multiple ways to cast to the same type. Creating VHDL is often verbose and syntax is difficult to remember. Creating a simple VHDL cheat sheet with boilerplate VHDL significantly cuts down the amount of time it takes to write frequently used constructs.

5. Source Code

Software

```
DM9000A.h:

#ifndef   __DM9000A_H__
#define   __DM9000A_H__

#define IO_addr     0
#define IO_data     1

#define NCR    0x00  /* Network  Control Register REG. 00 */
```

```
#define NSR    0x01  /* Network  Status Register  REG. 01 */
#define TCR    0x02  /* Transmit Control Register REG. 02 */
#define RCR    0x05  /* Receive  Control Register REG. 05 */
#define ETXCSR 0x30  /* TX early Control Register REG. 30 */
#define MRCMDX 0xF0  /* RX FIFO I/O port command: READ a byte from RX SRAM */
#define MRCMD  0xF2  /* RX FIFO I/O port command READ  from RX SRAM */
#define MWCMD  0xF8  /* TX FIFO I/O port command WRITE into TX FIFO */
#define ISR    0xFE  /* NIC Interrupt Status Register REG. FEH */
#define IMR    0xFF  /* NIC Interrupt Mask   Register REG. FFH */

#define NCR_set    0x00
#define TCR_set    0x00
#define TX_REQUEST 0x01  /* TCR REG. 02 TXREQ Bit [0] = 1 polling
                       Transmit Request command */
#define TCR_long   0x40  /* packet disable TX Jabber Timer */
#define RCR_set    0x30  /* skip CRC_packet and skip LONG_packet */
#define RX_ENABLE  0x01  /* RCR REG. 05 RXEN
                       Bit [0] = 1 to enable RX machine */
#define RCR_long   0x40  /* packet disable RX Watchdog Timer */
#define PASS_MULTICAST 0x08 /* RCR REG. 05 PASS_ALL_MULTICAST
                       Bit [3] = 1: RCR_set value ORed 0x08 */
#define BPTR_set   0x3F  /* BPTR REG. 08 RX Back Pressure Threshold:
                       High Water Overflow Threshold setting
                       3KB and Jam_Pattern_Time = 600 us */
#define FCTR_set   0x5A  /* FCTR REG. 09 High/ Low Water Overflow Threshold
                       setting 5KB/ 10KB */
#define RTFCR_set  0x29  /* RTFCR REG. 0AH RX/TX Flow Control Register
                       enable TXPEN + BKPM(TX_Half) + FLCE(RX) */
#define ETXCSR_set 0x83  /* Early Transmit Bit [7] Enable and
                       Threshold 0~3: 12.5%, 25%, 50%, 75% */
#define INTR_set   0x81  /* IMR REG. FFH: PAR +PRM, or 0x83: PAR + PRM + PTM */
#define PAR_set    0x80  /* IMR REG. FFH: PAR only, RX/TX FIFO R/W
                       Pointer Auto Return enable */

#define PHY_reset  0x8000  /* PHY reset: some registers back to
                         default value */
#define PHY_txab   0x05e1  /* set PHY TX advertised ability:
                         Full-capability + Flow-control (if necessary) */
#define PHY_mode   0x3100  /* set PHY media mode: Auto negotiation
                         (AUTO sense) */

// #define STD_DELAY       20      /* standard delay 20 us */
#define STD_DELAY 40

#define DMFE_SUCCESS    0
#define DMFE_FAIL       1

#define TRUE            1
#define FALSE           0

#define DM9000_PKT_READY  0x01  /* packets ready to receive */
#define PACKET_MIN_SIZE   0x40  /* Received packet min size */
#define MAX_PACKET_SIZE   1522  /* RX largest legal size packet
                            with fcs & QoS */
#define DM9000_PKT_MAX    3072  /* TX 1 packet max size without 4-byte CRC */
```

```
//-------------------------------------------------------------------------
extern void          dm9000a_iow(unsigned int reg, unsigned int data);
extern unsigned int  dm9000a_ior(unsigned int reg);
extern void          phy_write(unsigned int reg, unsigned int value);

/* DM9000_init I/O routine */
extern unsigned int  DM9000_init(unsigned char *mac_address);

/* Transmit One Packet TX I/O routine */
extern unsigned int  TransmitPacket(unsigned char *data_ptr,
                              unsigned int tx_len);

/* Receive One Packet I/O routine */
extern unsigned int  ReceivePacket(unsigned char *data_ptr,
                             unsigned int *rx_len);
//-------------------------------------------------------------------------

#endif
```

LCD.h:

```
#ifndef   __LCD_H__
#define   __LCD_H__

//  LCD Module 16*2
#define lcd_write_cmd(base, data)                     IOWR(base, 0, data)
#define lcd_read_cmd(base)                            IORD(base, 1)
#define lcd_write_data(base, data)                    IOWR(base, 2, data)
#define lcd_read_data(base)                           IORD(base, 3)
//-------------------------------------------------------------------------
void  LCD_Init();
void  LCD_Show_Text(char* Text);
void  LCD_Line2();
void  LCD_Test();
//-------------------------------------------------------------------------

#endif
```

VGA.h:

```
#ifndef   __VGA_H__
#define   __VGA_H__

#define VGA_0_BASE BINARY_VGA_CONTROLLER_0_BASE

//  VGA Parameter
#define VGA_WIDTH     640
#define VGA_HEIGHT    480
#define MAX_LINE_LENGTH 78
#define OSD_MEM_ADDR  VGA_WIDTH*VGA_HEIGHT

//  VGA Set Function
#define Vga_Write_Ctrl(base,value)          IOWR(base, OSD_MEM_ADDR   , value)
#define Vga_Cursor_X(base,value)            IOWR(base, OSD_MEM_ADDR+1 , value)
#define Vga_Cursor_Y(base,value)            IOWR(base, OSD_MEM_ADDR+2 , value)
```

```
#define Vga_Cursor_Color_R(base,value)      IOWR(base, OSD_MEM_ADDR+3 , value)
#define Vga_Cursor_Color_G(base,value)      IOWR(base, OSD_MEM_ADDR+4 , value)
#define Vga_Cursor_Color_B(base,value)      IOWR(base, OSD_MEM_ADDR+5 , value)
#define Vga_Pixel_On_Color_R(base,value)    IOWR(base, OSD_MEM_ADDR+6 , value)
#define Vga_Pixel_On_Color_G(base,value)    IOWR(base, OSD_MEM_ADDR+7 , value)
#define Vga_Pixel_On_Color_B(base,value)    IOWR(base, OSD_MEM_ADDR+8 , value)
#define Vga_Pixel_Off_Color_R(base,value)   IOWR(base, OSD_MEM_ADDR+9 , value)
#define Vga_Pixel_Off_Color_G(base,value)   IOWR(base, OSD_MEM_ADDR+10 , value)
#define Vga_Pixel_Off_Color_B(base,value)   IOWR(base, OSD_MEM_ADDR+11 , value)
#define Vga_Set_Pixel(base,x,y)             IOWR(base, (y)*VGA_WIDTH+(x), 1)
#define Vga_Clr_Pixel(base,x,y)             IOWR(base, (y)*VGA_WIDTH+(x), 0)

//-------------------------------------------------------------------------
typedef union VGA_Ctrl_Reg
{
  struct _VGA_Ctrl_Flags
  {
    unsigned char RED_ON      : 1;
    unsigned char GREEN_ON    : 1;
    unsigned char BLUE_ON     : 1;
    unsigned char CURSOR_ON   : 1;
    unsigned char RESERVED    : 4;
  }VGA_Ctrl_Flags;
  unsigned char Value;
}VGA_Ctrl_Reg;
//-------------------------------------------------------------------------
void Set_Cursor_XY(unsigned int X,unsigned int Y);
void Set_Cursor_Color(unsigned int R,unsigned int G,unsigned int B);
void Set_Pixel_On_Color(unsigned int R,unsigned int G,unsigned int B);
void Set_Pixel_Off_Color(unsigned int R,unsigned int G,unsigned int B);
void Clr_Screen();
void Divide_Screen(int line);
//-------------------------------------------------------------------------

extern void put_vga_char(char c, unsigned int column, unsigned int row);
extern void put_vga_string(char *str, unsigned int column, unsigned int row);
extern void put_cursor(char c, unsigned int column, unsigned int row);


#endif
```

alt_up_ps2_port.h:

```
#ifndef __PS2_INTERFACE_H__
#define __PS2_INTERFACE_H__

#include <io.h>
#include <alt_types.h>
#include "system.h"
#include "alt_up_ps2_port_regs.h"

#define ALT_UP_PS2_BASE PS2_0_BASE

/**
 * @brief The Enum type for PS/2 device type
 **/
typedef enum {
  // @brief Indicate that the device is a PS/2 Mouse
  PS2_MOUSE = 0,
  // @brief Indicate that the device is a PS/2 Keyboard
  PS2_KEYBOARD = 1,
  // @brief The program cannot determine what type the device is
  PS2_UNKNOWN = 2
} PS2_DEVICE;

#define DEFAULT_PS2_TIMEOUT_VAL  700000

#define PS2_SUCCESS (0)
#define PS2_TIMEOUT (-1)
#define PS2_ERROR   (-2)

#define PS2_ACK     (0xFA)

////////////////////////////////////////////////////////////
// Control Register Operations

/**
 * @brief Read the contents of the Control register for the PS/2 port
 *
 * @return Register contents (32 bits, bits 10, 8 and 0 are used for
 * CE, RI and RE respectively. Other bits are reserved)
 **/
alt_u32 read_ctrl_reg();

/**
 * @brief Set the contents of the Control register
 *
 * @param ctrl_data -- contents to be written into the Control register
 *
 **/
void write_ctrl_reg(alt_u32 ctrl_data);

/**
 * @brief Extract the RI (Read Interrupt) bit from the Control register
 *
 * @param ctrl_reg -- the Control register
 *
 * @return 8-bit number, where bit 0 is the value of the RI bit
 **/
alt_u8 read_RI_bit(alt_u32 ctrl_reg);

/**
 * @brief Extract the RE (Read Interrupt Enable) bit from the Control register
 *
 * @param ctrl_reg -- the Control register
 *
 * @return 8-bit number, where bit 0 is the value of the RE bit
 **/
alt_u8 read_RE_bit(alt_u32 ctrl_reg);
```

```
/**
 * @brief Extract the CE (Command Error) bit from the Control register
 *
 * @param ctrl_reg -- the Control register
 *
 * @return 8-bit number, where bit 0 is the value of the CE bit
 **/
alt_u8 read_CE_bit(alt_u32 ctrl_reg);

////////////////////////////////////////////////////////////
// Data Register Operations

/**
 * @brief Read the contents of the Data register
 *
 * @return 32 bits of the Data register. Bits 31-16 indicate the number
 * of available bytes in the FIFO (RAVAIL), bits 7-0 are the data received
 * from the PS/2 device
 *
 **/
alt_u32 read_data_reg();

/**
 * @brief Read the DATA byte from the Data register
 *
 * @param data_reg  -- Data register
 *
 * @return Bits 7-0 of the Data register
 **/
alt_u8 read_data_byte(alt_u32 data_reg);

/**
 * @brief Find the number of bytes available to read in the FIFO buffer
 * of the PS/2 port
 *
 * @param data_reg  -- the Data register
 *
 * @return The number represented by bits 31-16 of the Data register
 **/
alt_u16 read_num_bytes_available(alt_u32 data_reg);

////////////////////////////////////////////////////////////
// Actions

/**
 * @brief Check the PS/2 peripheral's mode (whether it is a keyboard or
 *        a mouse)
 *
 * @return PS2_MOUSE for mouse, or PS2_KEYBOARD for keyboard
 *
 * @note This operation will reset the PS/2 peripheral. Usually,
 * it should be used only at the beginning of a program.
 **/
PS2_DEVICE get_mode();
```

```
/**
 * @brief Clear the FIFO's contents
 **/
void clear_FIFO();

/**
 * @brief Wait for the acknowledge byte (0xFA) from the PS/2 peripheral
 *
 * @param timeout -- the number of cycles of timeout
 *
 * @return \c PS2_SUCCESS on receving ACK signal, or \c PS2_TIMEOUT on timeout.
 **/
int wait_for_ack(unsigned timeout);

/**
 * @brief Send a one-byte command to the PS/2 peripheral
 *
 * @param byte -- the one-byte command to be sent
 *
 * @return \c PS2_ERROR if the CE bit of the Control register is
 * set to 1, otherwise \c PS2_SUCCESS
 **/
int write_data_byte(alt_u8 byte);

/**
 * @brief Send a one-byte command to the PS/2 peripheral and
 *        wait for the ACK signal
 *
 * @param byte -- the one-byte command to be sent.
 *        See <tt> alt_up_ps2_port_regs.h </tt> in the sdk directory or
 *        any reference for the PS/2 protocol for details.
 *
 * @return PS2_ERROR if the CE bit of the Control register is set to 1, or
 *         PS2_TIMEOUT on timeout, or
 *         PS2_SUCCESS if the ACK signal is received before timeout
 **/
int write_data_byte_with_ack(alt_u8 byte, unsigned timeout);

/**
 * @brief Read the DATA byte from the PS/2 FIFO,
 *        using a user-defined timeout value
 *
 * @param byte  -- the byte read from the FIFO for the PS/2 Core
 * @param time_out  -- the user-defined timeout value. Setting
 *        time_out to 0 will disable the time-out mechanism
 *
 * @return \c PS2_SUCCESS on reading data, or \c PS2_TIMEOUT on timeout
 **/
int read_data_byte_with_timeout(alt_u8 *byte, alt_u32 time_out);

#endif

alt_up_ps2_port_regs.h:

#ifndef __ALT_UP_PS2_PORT_REGS_H__
```

```
#define __ALT_UP_PS2_PORT_REGS_H__

/*
 * Data Register
 */
#define ALT_UP_PS2_PORT_DATA_REG                  0
#define IOADDR_ALT_UP_PS2_PORT_DATA(base)         \
        __IO_CALC_ADDRESS_NATIVE(base, ALT_UP_PS2_PORT_DATA_REG)
#define IORD_ALT_UP_PS2_PORT_DATA(base)           \
        IORD(base, ALT_UP_PS2_PORT_DATA_REG)
#define IOWR_ALT_UP_PS2_PORT_DATA(base, data)     \
        IOWR(base, ALT_UP_PS2_PORT_DATA_REG, data)

#define ALT_UP_PS2_PORT_DATA_REG_DATA_MSK      (0x000000FF)
#define ALT_UP_PS2_PORT_DATA_REG_DATA_OFST     (0)
#define ALT_UP_PS2_PORT_DATA_REG_RVALID_MSK    (0x00008000)
#define ALT_UP_PS2_PORT_DATA_REG_RVALID_OFST   (15)
#define ALT_UP_PS2_PORT_DATA_REG_RAVAIL_MSK    (0xFFFF0000)
#define ALT_UP_PS2_PORT_DATA_REG_RAVAIL_OFST   (16)

/*
 * Control Register
 */
#define ALT_UP_PS2_PORT_CONTROL_REG               1
#define IOADDR_ALT_UP_PS2_PORT_CONTROL(base)      \
        __IO_CALC_ADDRESS_NATIVE(base, ALT_UP_PS2_PORT_CONTROL_REG)
#define IORD_ALT_UP_PS2_PORT_CONTROL(base)        \
        IORD(base, ALT_UP_PS2_PORT_CONTROL_REG)
#define IOWR_ALT_UP_PS2_PORT_CONTROL(base, data)  \
        IOWR(base, ALT_UP_PS2_PORT_CONTROL_REG, data)

#define ALT_UP_PS2_PORT_CONTROL_RE_MSK            (0x00000001)
#define ALT_UP_PS2_PORT_CONTROL_RE_OFST           (0)
#define ALT_UP_PS2_PORT_CONTROL_RI_MSK            (0x00000100)
#define ALT_UP_PS2_PORT_CONTROL_RI_OFST           (8)
#define ALT_UP_PS2_PORT_CONTROL_CE_MSK          (0x00000400)
#define ALT_UP_PS2_PORT_CONTROL_CE_OFST         (10)

#endif
```

basic_io.h:

```
#ifndef   __basic_io_H__
#define   __basic_io_H__

#include <io.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include "system.h"
#include "sys/alt_irq.h"

//  for GPIO
#define inport(base)                                       IORD(base, 0)
#define outport(base, data)                                IOWR(base, 0, data)
```

```
#define get_pio_dir(base)                           IORD(base, 1)
#define set_pio_dir(base, data)                     IOWR(base, 1, data)
#define get_pio_irq_mask(base)                      IORD(base, 2)
#define set_pio_irq_mask(base, data)                IOWR(base, 2, data)
#define get_pio_edge_cap(base)                      IORD(base, 3)
#define set_pio_edge_cap(base, data)                IOWR(base, 3, data)

//  for SEG7 Display
#define seg7_show(base,data)                        IOWR(base, 0, data)

//  for Time Delay
#define msleep(msec)                                usleep(1000*msec);
#define Sleep(sec)                                  msleep(1000*sec);

#endif
```

gui.h:

```
#ifndef _GUI_H
#define _GUI_H  1

#define IMG_X_SIZE  340
#define IMG_Y_SIZE  238
#define THUMBNAIL_X_SIZE  680
#define THUMBNAIL_Y_SIZE  238
#define Y_OFF   0
#define X_OFF   0
#define MAX_NUM_IMAGES 30
#define IMAGE_PATH  "/mnt/host/sram.bmp"

#define FILE_SIZE_OFFSET    2
#define DATA_OFFSET         10
#define PIXEL_WIDTH_OFFSET  18
#define PIXEL_HEIGHT_OFFSET 22

struct image
{
    struct image *prev;
    int is_taken;
    int image_num;
    struct image *next;
};

void write_image(struct image *);
void send_image();
void save_image();
void delete_current_image();
void send_thumbnails();
void freeze_toggle();
void full_screen(int is_full_screen);
int get_num_images();
int get_current_img_num();
int save_remote_image();

unsigned char saved_images[MAX_NUM_IMAGES][3][IMG_Y_SIZE][IMG_X_SIZE];
```

```
#endif

jpeg_encoder.h:

#ifndef JPEG_ENCODER_H
#define JPEG_ENCODER_H

int compress_jpeg();

extern unsigned char jpeg_image[15000];
extern unsigned int jpeg_image_size;

#endif // JPEG_ENCODER_H

misc.h:

#ifndef MISC_H_
#define MISC_H_

#define NETWORK_ENABLED

void alex_init();
void gotosleep(unsigned int seconds);

#endif /*MISC_H_*/

network.h:

#ifndef NETWORK_H_
#define NETWORK_H_

struct SPacket {
  char* data;
  unsigned int size;
};

typedef struct SPacket Packet;

extern const unsigned char SOURCE_MAC_ADDR[];
extern const unsigned char DEST_MAC_ADDR[];
extern const int ETHERNET_HEADER_SIZE;
extern const int IP_HEADER_SIZE;
extern const int UDP_HEADER_SIZE;

//#define MIN_PACKET_SIZE 64
#define MIN_PACKET_SIZE 128
#define NETWORK_MAX_PACKET_SIZE 1024

#define HEADER_OFFSET_SOURCE_MAC 0
#define HEADER_OFFSET_DEST_MAC 6
```

```
#define HEADER_OFFSET_IP_TOTAL_LENGTH_H 16
#define HEADER_OFFSET_IP_TOTAL_LENGTH_L 17

#define HEADER_OFFSET_PACKET_ID_H 18
#define HEADER_OFFSET_PACKET_ID_L 19

#define HEADER_OFFSET_PROTOCOL 23

#define HEADER_OFFSET_IP_CHECKSUM_H 24
#define HEADER_OFFSET_IP_CHECKSUM_L 25

// MSB
#define HEADER_OFFSET_SOURCE_IP_3 26
#define HEADER_OFFSET_SOURCE_IP_2 27
#define HEADER_OFFSET_SOURCE_IP_1 28
#define HEADER_OFFSET_SOURCE_IP_0 29

// MSB
#define HEADER_OFFSET_DEST_IP_3 30
#define HEADER_OFFSET_DEST_IP_2 31
#define HEADER_OFFSET_DEST_IP_1 32
#define HEADER_OFFSET_DEST_IP_0 33

#define HEADER_OFFSET_UDP 34
#define HEADER_OFFSET_SOURCE_PORT_H 34
#define HEADER_OFFSET_SOURCE_PORT_L 35

#define HEADER_OFFSET_DEST_PORT_H 36
#define HEADER_OFFSET_DEST_PORT_L 37

#define HEADER_OFFSET_UDP_LENGTH_H 38
#define HEADER_OFFSET_UDP_LENGTH_L 39

#define HEADER_OFFSET_UDP_CHECKSUM_H 40
#define HEADER_OFFSET_UDP_CHECKSUM_L 41

#define HEADER_OFFSET_UDP_DATA 42

#define NETWORK_UDP_PACKET_SIZE (ETHERNET_HEADER_SIZE + IP_HEADER_SIZE +
UDP_HEADER_SIZE)

void network_init();
void network_interrupt_handler();
void network_send_packet(const char*, const unsigned int);
void network_format_packet(Packet*, const char*, const unsigned int);
unsigned short network_btoles(unsigned short val);
void network_display_packet(unsigned char*);

typedef unsigned short u16;
typedef unsigned long u32;

unsigned short network_calc_ip_checksum(unsigned char[], const int);
u16 network_calc_udp_checksum (u16, unsigned char*, unsigned char*, char padding,
unsigned char*);
```

```
void network_send_file(const char* filename);

void network_set_source_port(unsigned short);
void network_set_dest_port(unsigned short);
void network_set_source_ip(unsigned char, unsigned char, unsigned char, unsigned
char);
void network_set_dest_ip(unsigned char, unsigned char, unsigned char, unsigned char);

#endif /*NETWORK_H_*/
```

ps2_keyboard.h:

```
#ifndef __PS2_KEYBOARD_H__
#define __PS2_KEYBOARD_H__
#include "alt_up_ps2_port.h"

#define KB_RESET 0xFF
#define KB_SET_DEFAULT 0xF6
#define KB_DISABLE 0xF5
#define KB_ENABLE 0xF4
#define KB_SET_TYPE_RATE_DELAY 0xF3

/**
 * @brief The Enum type for the type of keyboard code received
 **/
typedef enum
  {
    /** @brief --- Make Code that corresponds to an ASCII character.
      For example, the ASCII Make Code for letter <tt>A</tt> is 1C
     */
    KB_ASCII_MAKE_CODE = 1,
    /** @brief --- Make Code that corresponds to a non-ASCII character.
      For example, the Binary (Non-ASCII) Make Code for
      <tt>Left Alt</tt> is 11
     */
    KB_BINARY_MAKE_CODE = 2,
    /** @brief --- Make Code that has two bytes (the first byte is E0).
      For example, the Long Binary Make Code for <tt>Right Alt</tt>
      is "E0 11"
     */
    KB_LONG_BINARY_MAKE_CODE = 3,
    /** @brief --- Normal Break Code that has two bytes (the first byte is F0).
      For example, the Break Code for letter <tt>A</tt> is "F0 1C"
     */
    KB_BREAK_CODE = 4,
    /** @brief --- Long Break Code that has three bytes (the first two bytes
      are E0, F0). For example, the Long Break Code for <tt>Right Alt</tt>
      is "E0 F0 11"
     */
    KB_LONG_BREAK_CODE = 5,
    /** @brief --- Codes that the decode FSM cannot decode
     */
    KB_INVALID_CODE = 6
  } KB_CODE_TYPE;
```

```
/**
 * @brief Get the make code of the key when a key is pressed
 *
 * @param decode_mode -- indicates which type of code
 * (Make Code, Break Code, etc.) is received from the keyboard when the
 * key is pressed
 *
 * @param buf -- points to the location that stores the make code of
 *              the key pressed
 * @note For KB_LONG_BINARY_MAKE_CODE and KB_BREAK_CODE, only the
 *       second byte is retured. For KB_LONG_BREAK_CODE, only the
 *       third byte is returned
 *
 * @return \c PS2_TIMEOUT on timeout, or \c PS2_ERROR on error,
 *         otherwise \c PS2_SUCCESS
 **/
extern int read_make_code(KB_CODE_TYPE *decode_mode, alt_u8 *buf);

/**
 * @brief Set the repeat/delay rate of the keyboard
 *
 * @param rate -- an 8-bit number that represents the repeat/delay rate
 *                of the keyboard
 *
 * @return PS2_SUCCESS on success, otherwise PS2_ERROR
 **/
extern alt_u32 set_keyboard_rate(alt_u8 rate);

/**
 * @brief Send the reset command to the keyboard
 *
 * @return \c PS2_SUCCESS on passing the BAT (Basic Assurance Test),
 *         otherwise \c PS2_ERROR
 **/
extern alt_u32 reset_keyboard();

#endif
```

ps2_mouse.h:

```
#ifndef __PS2_MOUSE_H__
#define __PS2_MOUSE_H__

#include "alt_up_ps2_port.h"

//mouse comamnds
#define MOUSE_SET_SAMPLE_RATE 0xF3
#define MOUSE_RESET 0xFF
#define MOUSE_SET_DEFAULT 0xF6
#define MOUSE_DISABLE_DATA_REPORTING 0xF5
#define MOUSE_ENABLE_DATA_REPORTING 0xF4
#define MOUSE_SET_SAMPLE_RATE 0xF3
#define MOUSE_READ_DATA 0xEB
#define MOUSE_SET_STREAM_MODE 0xEA
#define MOUSE_REQUEST_STATUS 0xE9
```

```
#define MOUSE_SET_RESOLUTION 0xE8

//Mouse modes (incomplete list)
#define MOUSE_STREAM_MODE 0xEA
#define MOUSE_REMOTE_MODE 0xF0

/**
 * @brief Reset the mouse
 *
 * @return \c PS2_SUCCESS on BAT is passed, otherwise \c PS2_ERROR
 **/
alt_u8 reset_mouse();

/**
 * @brief Set the operation mode of the mouse
 *
 * @param byte -- the byte representing the mode (see macro
 *                definitions for details)
 * @sa PS/2 Mouse document
 *
 * @return \c PS2_SUCCESS on receiving acknowledgment
 **/
int set_mouse_mode(alt_u8 byte);

#endif
```

DM9000A.c:

```c
#include <stdio.h>
#include "DM9000A.h"
#include "basic_io.h"

void dm9000a_iow(unsigned int reg, unsigned int data)
{
  IOWR(DM9000A_BASE, IO_addr, reg);
  usleep(STD_DELAY);
  IOWR(DM9000A_BASE, IO_data, data);
}

unsigned int dm9000a_ior(unsigned int reg)
{
  IOWR(DM9000A_BASE, IO_addr, reg);
  usleep(STD_DELAY);
  return IORD(DM9000A_BASE, IO_data);
}


void phy_write(unsigned int reg, unsigned int value)
{
  /* set PHY register address into EPAR REG. 0CH */
  dm9000a_iow(0x0C, reg | 0x40); /* PHY register address setting,
                        and DM9000_PHY offset = 0x40 */

  /* fill PHY WRITE data into EPDR REG. 0EH & REG. 0DH */
  dm9000a_iow(0x0E, ((value >> 8) & 0xFF));   /* PHY data high_byte */
  dm9000a_iow(0x0D, value & 0xFF);            /* PHY data low_byte */

  /* issue PHY + WRITE command = 0xa into EPCR REG. 0BH */
  dm9000a_iow(0x0B, 0x8);                     /* clear PHY command first */
  IOWR(DM9000A_BASE, IO_data, 0x0A);  /* issue PHY + WRITE command */
  usleep(STD_DELAY);
  IOWR(DM9000A_BASE, IO_data, 0x08);  /* clear PHY command again */
  usleep(50);  /* wait 1~30 us (>20 us) for PHY + WRITE completion */
}

/* DM9000_init I/O routine */
unsigned int DM9000_init(unsigned char *mac_address)
{
  unsigned int  i;

  /* set the internal PHY power-on (GPIOs normal settings) */
  dm9000a_iow(0x1E, 0x01);  /* GPCR REG. 1EH = 1 selected
                  GPIO0 "output" port for internal PHY */
  dm9000a_iow(0x1F, 0x00);  /* GPR  REG. 1FH GEPIO0
                  Bit [0] = 0 to activate internal PHY */
  msleep(5);        /* wait > 2 ms for PHY power-up ready */

  /* software-RESET NIC */
  dm9000a_iow(NCR, 0x03);   /* NCR REG. 00 RST Bit [0] = 1 reset on,
                  and LBK Bit [2:1] = 01b MAC loopback on */
  usleep(20);       /* wait > 10us for a software-RESET ok */
```

```
dm9000a_iow(NCR, 0x00);   /* normalize */
dm9000a_iow(NCR, 0x03);
usleep(20);
dm9000a_iow(NCR, 0x00);

/* set GPIO0=1 then GPIO0=0 to turn off and on the internal PHY */
dm9000a_iow(0x1F, 0x01);  /* GPR PHYPD Bit [0] = 1 turn-off PHY */
dm9000a_iow(0x1F, 0x00);  /* PHYPD Bit [0] = 0 activate phyxcer */
msleep(10);       /* wait >4 ms for PHY power-up */

/* set PHY operation mode */
phy_write(0,PHY_reset);   /* reset PHY registers back to the default state */
usleep(50);               /* wait >30 us for PHY software-RESET ok */
phy_write(16, 0x404);     /* turn off PHY reduce-power-down mode only */
phy_write(4, PHY_txab);   /* set PHY TX advertised ability:
                         ALL + Flow_control */
phy_write(0, 0x1200);     /* PHY auto-NEGO re-start enable
                         (RESTART_AUTO_NEGOTIATION +
                         AUTO_NEGOTIATION_ENABLE)
                         to auto sense and recovery PHY registers */
msleep(5);                /* wait >2 ms for PHY auto-sense
                         linking to partner */

/* store MAC address into NIC */
for (i = 0; i < 6; i++)
  dm9000a_iow(16 + i, mac_address[i]);

/* clear any pending interrupt */
dm9000a_iow(ISR, 0x3F);  /* clear the ISR status: PRS, PTS, ROS, ROOS 4 bits,
                 by RW/C1 */
dm9000a_iow(NSR, 0x2C);  /* clear the TX status: TX1END, TX2END, WAKEUP 3 bits,
                 by RW/C1 */

/* program operating registers~ */
dm9000a_iow(NCR, NCR_set); /* NCR REG. 00 enable the chip functions
                  (and disable this MAC loopback mode back to normal) */
dm9000a_iow(0x08, BPTR_set); /* BPTR  REG.08  (if necessary) RX Back Pressure
                    Threshold in Half duplex moe only:
                    High Water 3KB, 600 us */
dm9000a_iow(0x09, FCTR_set);  /* FCTR  REG.09  (if necessary)
                     Flow Control Threshold setting
                     High/ Low Water Overflow 5KB/ 10KB */
dm9000a_iow(0x0A, RTFCR_set); /* RTFCR REG.0AH (if necessary)
                     RX/TX Flow Control Register enable TXPEN, BKPM
                     (TX_Half), FLCE (RX) */
dm9000a_iow(0x0F, 0x00);      /* Clear the all Event */
dm9000a_iow(0x2D, 0x80);      /* Switch LED to mode 1 */

/* set other registers depending on applications */
dm9000a_iow(ETXCSR, ETXCSR_set); /* Early Transmit 75% */

/* enable interrupts to activate DM9000 ~on */
dm9000a_iow(IMR, INTR_set);   /* IMR REG. FFH PAR=1 only,
                     or + PTM=1& PRM=1 enable RxTx interrupts */
  /* enable RX (Broadcast/ ALL_MULTICAST) ~go */
  dm9000a_iow(RCR , RCR_set | RX_ENABLE | PASS_MULTICAST);
  /* RCR REG. 05 RXEN Bit [0] = 1 to enable the RX machine/ filter */

  /* RETURN "DEVICE_SUCCESS" back to upper layer */
  return  (dm9000a_ior(0x2D)==0x80) ? DMFE_SUCCESS : DMFE_FAIL;
}

unsigned int TransmitPacket(unsigned char *data_ptr, unsigned int tx_len)
{
  unsigned int i;

  /* mask NIC interrupts IMR: PAR only */
  dm9000a_iow(IMR, PAR_set);

  /* issue TX packet's length into TXPLH REG. FDH & TXPLL REG. FCH */
  dm9000a_iow(0xFD, (tx_len >> 8) & 0xFF);  /* TXPLH High_byte length */
  dm9000a_iow(0xFC, tx_len & 0xFF);         /* TXPLL Low_byte  length */

  /* wirte transmit data to chip SRAM */
  IOWR(DM9000A_BASE, IO_addr, MWCMD);  /* set MWCMD REG. F8H
                                  TX I/O port ready */
  for (i = 0; i < tx_len; i += 2) {
    usleep(STD_DELAY);
    IOWR(DM9000A_BASE, IO_data, (data_ptr[i+1]<<8)|data_ptr[i] );
  }

  /* issue TX polling command activated */
  dm9000a_iow(TCR , TCR_set | TX_REQUEST);  /* TXCR Bit [0] TXREQ auto clear
                                after TX completed */

  /* wait TX transmit done */
  while(!(dm9000a_ior(NSR)&0x0C))
    usleep(STD_DELAY);

  /* clear the NSR Register */
  dm9000a_iow(NSR,0x00);

  /* re-enable NIC interrupts */
  dm9000a_iow(IMR, INTR_set);

  /* RETURN "TX_SUCCESS" to upper layer */
  return  DMFE_SUCCESS;
}

unsigned int ReceivePacket(unsigned char *data_ptr, unsigned int *rx_len)
{
  unsigned char rx_READY, GoodPacket;
  unsigned int  Tmp, RxStatus, i;

  RxStatus = rx_len[0] = 0;
  GoodPacket=FALSE;

  /* mask NIC interrupts IMR: PAR only */
  dm9000a_iow(IMR, PAR_set);
```

```
/* dummy read a byte from MRCMDX REG. F0H */
rx_READY = dm9000a_ior(MRCMDX);

/* got most updated byte: rx_READY */
rx_READY = IORD(DM9000A_BASE,IO_data)&0x03;
usleep(STD_DELAY);

/* check if (rx_READY == 0x01): Received Packet READY? */
if (rx_READY == DM9000_PKT_READY) {

  /* got RX_Status & RX_Length from RX SRAM */
  IOWR(DM9000A_BASE, IO_addr, MRCMD); /* set MRCMD REG. F2H
                                 RX I/O port ready */
  usleep(STD_DELAY);
  RxStatus = IORD(DM9000A_BASE,IO_data);
  usleep(STD_DELAY);
  rx_len[0] = IORD(DM9000A_BASE,IO_data);

  /* Check this packet_status GOOD or BAD? */
  if ( !(RxStatus & 0xBF00) && (rx_len[0] < MAX_PACKET_SIZE) ) {
    /* read 1 received packet from RX SRAM into RX buffer */
    for (i = 0; i < rx_len[0]; i += 2) {
     usleep(STD_DELAY);
     Tmp = IORD(DM9000A_BASE, IO_data);
     data_ptr[i] = Tmp & 0xFF;
     data_ptr[i+1] = (Tmp>>8) & 0xFF;
    }
    GoodPacket = TRUE;
  } else {
    /* this packet is bad, dump it from RX SRAM */
    for (i = 0; i < rx_len[0]; i += 2) {
     usleep(STD_DELAY);
     Tmp = IORD(DM9000A_BASE, IO_data);
    }
    printf("\nError\n");
    rx_len[0] = 0;
  }
} else if (rx_READY) { /* status check first byte:
                     rx_READY Bit[1:0] must be "00"b or "01"b */

  /* software-RESET NIC */

  dm9000a_iow(NCR, 0x03);   /* NCR REG. 00 RST Bit [0] = 1 reset on,
                 and LBK Bit [2:1] = 01b MAC loopback on */
  usleep(20);       /* wait > 10us for a software-RESET ok */
  dm9000a_iow(NCR, 0x00);   /* normalize */
  dm9000a_iow(NCR, 0x03);
  usleep(20);
  dm9000a_iow(NCR, 0x00);
  /* program operating registers~ */
  dm9000a_iow(NCR, NCR_set); /* NCR REG. 00 enable the chip functions
                 (and disable this MAC loopback mode back to normal) */
  dm9000a_iow(0x08, BPTR_set);  /* BPTR  REG.08  (if necessary) RX Back Pressure
                      Threshold in Half duplex moe only:
                         High Water 3KB, 600 us */
    dm9000a_iow(0x09, FCTR_set);  /* FCTR  REG.09  (if necessary)
                         Flow Control Threshold setting High/Low Water
                         Overflow 5KB/ 10KB */
    dm9000a_iow(0x0A, RTFCR_set); /* RTFCR REG.0AH (if necessary)
                         RX/TX Flow Control Register
                         enable TXPEN, BKPM (TX_Half), FLCE (RX) */
    dm9000a_iow(0x0F, 0x00);      /* Clear the all Event */
    dm9000a_iow(0x2D, 0x80);      /* Switch LED to mode 1 */
    /* set other registers depending on applications */
    dm9000a_iow(ETXCSR, ETXCSR_set); /* Early Transmit 75% */
    /* enable interrupts to activate DM9000 ~on */
    dm9000a_iow(IMR, INTR_set);   /* IMR REG. FFH PAR=1 only,
                         or + PTM=1& PRM=1 enable RxTx interrupts */
    /* enable RX (Broadcast/ ALL_MULTICAST) ~go */
    dm9000a_iow(RCR , RCR_set | RX_ENABLE | PASS_MULTICAST);
      /* RCR REG. 05 RXEN Bit [0] = 1 to enable the RX machine/ filter */
  }

  return GoodPacket ? DMFE_SUCCESS : DMFE_FAIL;
}

LCD.c:

#include <unistd.h>
#include <string.h>
#include <io.h>
#include "system.h"
#include "LCD.h"

void LCD_Init()
{
  lcd_write_cmd(LCD_16207_0_BASE,0x38);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x0C);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x01);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x06);
  usleep(2000);
  lcd_write_cmd(LCD_16207_0_BASE,0x80);
  usleep(2000);
}

void LCD_Show_Text(char* Text)
{
  int i;
  for(i=0;i<strlen(Text);i++) {
    lcd_write_data(LCD_16207_0_BASE,Text[i]);
    usleep(2000);
  }
}

void LCD_Line2()
{
  lcd_write_cmd(LCD_16207_0_BASE,0xC0);
  usleep(2000);
}

void LCD_Test()
{
  char Text1[16] = "<NIOS II on UP4>";
  char Text2[16] = "Nice to See You!";
  //  Initial LCD
  LCD_Init();
  //  Show Text to LCD
  LCD_Show_Text(Text1);
  //  Change Line2
  LCD_Line2();
  //  Show Text to LCD
  LCD_Show_Text(Text2);
}

alt_up_ps2_port.c:

#include <nios2.h>
#include "alt_up_ps2_port.h"

PS2_DEVICE get_mode()
{
  alt_u8 byte;
  //send the reset request, wait for ACK
  int status = write_data_byte_with_ack(0xff, DEFAULT_PS2_TIMEOUT_VAL);
  if (status == PS2_SUCCESS) {
    // reset succeed, now try to get the BAT result, AA means passed
    status = read_data_byte_with_timeout(&byte, DEFAULT_PS2_TIMEOUT_VAL);
    if (status == PS2_SUCCESS && byte == 0xAA) {
      //get the 2nd byte
      status = read_data_byte_with_timeout(&byte, DEFAULT_PS2_TIMEOUT_VAL);
      if (status == PS2_TIMEOUT) {
       //for keyboard, only 2 bytes are sent(ACK, PASS/FAIL), so timeout
       return PS2_KEYBOARD;
      } else if (status == PS2_SUCCESS && byte == 0x00) {
       //for mouse, it will sent out 0x00 after sending out ACK and PASS/FAIL.
       return PS2_MOUSE;
      }
    }
  }
  // when writing data to the PS/2 device, error occurs...
  return PS2_UNKNOWN;
}

void clear_FIFO()
{
  // The DATA byte of the data register will be automatically cleared after
  // a read, so we simply keep reading it until there are no available bytes
  alt_u16 num = 0;
  alt_u32 data_reg = 0;
  do {
    // read the data register (the DATA byte is cleared)
    data_reg = read_data_reg();
    // get the number of available bytes from the RAVAIL part of data register
    num = read_num_bytes_available(data_reg);
  } while (num > 0);
}

////////////////////////////////////////////////////////////
// Control Register Operations
void write_ctrl_reg(alt_u32 ctrl_data)
{
  IOWR_ALT_UP_PS2_PORT_CONTROL(ALT_UP_PS2_BASE, ctrl_data);
}

alt_u32 read_ctrl_reg()
{
  alt_u32 ctrl_reg = IORD_ALT_UP_PS2_PORT_CONTROL(ALT_UP_PS2_BASE);
  return ctrl_reg;
}

alt_u8 read_RI_bit(alt_u32 ctrl_reg)
{
  alt_u8 ri = (alt_u8) ((ctrl_reg & ALT_UP_PS2_PORT_CONTROL_RI_MSK)
                  >> ALT_UP_PS2_PORT_CONTROL_RI_OFST);
  return ri;
}

alt_u8 read_RE_bit(alt_u32 ctrl_reg)
{
  alt_u8 re = (alt_u8) ((ctrl_reg & ALT_UP_PS2_PORT_CONTROL_RE_MSK)
                  >> ALT_UP_PS2_PORT_CONTROL_RE_OFST);
  return re;
}

alt_u8 read_CE_bit(alt_u32 ctrl_reg)
{
  alt_u8 re = (alt_u8) ((ctrl_reg & ALT_UP_PS2_PORT_CONTROL_CE_MSK)
                  >> ALT_UP_PS2_PORT_CONTROL_CE_OFST);
  return re;
}

////////////////////////////////////////////////////////////
// Data Register Operations

alt_u32 read_data_reg()
{
  alt_u32 data_reg = IORD_ALT_UP_PS2_PORT_DATA(ALT_UP_PS2_BASE);
  return data_reg;
}

alt_u16 read_num_bytes_available(alt_u32 data_reg)
{
  alt_u16 ravail = (alt_u16)((data_reg & ALT_UP_PS2_PORT_DATA_REG_RAVAIL_MSK )
                        >> ALT_UP_PS2_PORT_DATA_REG_RAVAIL_OFST);
  return ravail;
}
```

```
alt_u8 read_data_byte(alt_u32 data_reg)
{
  alt_u8 data = (alt_u8) ( (data_reg & ALT_UP_PS2_PORT_DATA_REG_DATA_MSK)
                     >> ALT_UP_PS2_PORT_DATA_REG_DATA_OFST) ;
  return data;
}

int write_data_byte(alt_u8 byte)
{
  //note: data are only located at the lower 8 bits
  //note: the software send command to the PS2 peripheral through the data
  //         register rather than the control register
  IOWR_ALT_UP_PS2_PORT_DATA(ALT_UP_PS2_BASE, byte);
  alt_u32 ctrl_reg = IORD_ALT_UP_PS2_PORT_DATA(ALT_UP_PS2_BASE);
  if ( read_CE_bit(ctrl_reg) ) {
    //CE bit is set --> error occurs on sending commands
    return PS2_ERROR;
  }
  return PS2_SUCCESS;
}

int write_data_byte_with_ack(alt_u8 byte, unsigned timeout)
{
  //note: data are only located at the lower 8 bits
  //note: the software send command to the PS2 peripheral through the data
  //         register rather than the control register
  int send_status = write_data_byte(byte);
  if ( send_status != PS2_SUCCESS ) {
    // return on sending error
    return send_status;
  }

  int ack_status = wait_for_ack(timeout);
  return ack_status;
}

int read_data_byte_with_timeout(alt_u8 *byte, alt_u32 time_out)
{
  alt_u32 data_reg = 0;
  alt_u16 num = 0;
  alt_u32 count = 0;
  for (;;) {
    count++;
    data_reg = read_data_reg();
    num = read_num_bytes_available(data_reg);
    if (num > 0) {
      *byte = read_data_byte(data_reg);
      return PS2_SUCCESS;
    }
    //timeout = 0 means to disable the timeout
    if ( time_out!= 0 && count > time_out) {
      return PS2_TIMEOUT;
    }
  }
}

int wait_for_ack(unsigned timeout)
{
  alt_u8 ack = 0;
  alt_u8 data = 0;
  alt_u8 status = PS2_SUCCESS;
  for (;;) {
    status = read_data_byte_with_timeout(&data, timeout);
    if ( status == PS2_SUCCESS ) {
      if (data == PS2_ACK)
       return PS2_SUCCESS;
    } else {
      return status;
    }
  }
  return PS2_TIMEOUT;
}
```

gui.c:

```
#include <io.h>
#include <system.h>
#include <stdio.h>
#include "gui.h"

struct image images[MAX_NUM_IMAGES];

struct image *current_image;
struct image *prev_image = NULL;
int num_images = 0;
int current_image_num = 0;

int initialize_gui()
{
    send_thumbnails();
    // initialize images
    current_image = &images[0];
    current_image->image_num = 0;
    current_image->is_taken = 0;
    current_image->prev = current_image;
    current_image->next = current_image;

    int save_image_req = 0;
    int send_image_req = 0;
    int del_image_req = 0;
    int send_thumbnails_req;
    /*
    for (;;)
    {
        // send number of stored images
        IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 24, current_image_num);
        // check if need to get image
        save_image_req = IORD_16DIRECT(TV_TO_VGA_INST_BASE, 8);
        send_image_req = IORD_16DIRECT(TV_TO_VGA_INST_BASE, 20);
        del_image_req = IORD_16DIRECT(TV_TO_VGA_INST_BASE, 26);

        //check if need to refresh thumbnails
        //send_thumbnails_req = IORD_16DIRECT(TV_TO_VGA_INST_BASE, 52);
        if (send_thumbnails_req == 1)
        {
                //send_thumbnails();
        }
        else if (save_image_req == 1 && num_images < MAX_NUM_IMAGES)
        {
           // printf("saving-----%d\n", send_image_req);
            save_image();
            //write_image();
        }
        else if (num_images >= 1 && send_image_req != 0 && save_image_req != 1)
        {
            send_image(send_image_req);
        }
        else if (del_image_req == 1 && num_images > 0)
        {
            delete_current_image();
        }
        del_image_req = 0;
        send_image_req = 0;
        save_image_req = 0;
    }*/
}

int get_current_img_num()
{
    return current_image->image_num;
}

int get_num_images()
{
    return num_images;
}

int freeze = 0;
void freeze_toggle()
{
    freeze = !freeze;
    printf("freeze: %d\n", freeze);
    IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 50, freeze);
}

void full_screen(int is_full_screen)
{
    IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 52, is_full_screen);
}

void delete_current_image()
{
    printf("deleting image:\n");
    current_image->prev->next = current_image->next;
    current_image->next->prev = current_image->prev;
    current_image->is_taken = 0;
    num_images--;

    if (num_images != 0)
        send_image(2);
    else
        send_image(2);
    printf("number of images: %d\n", num_images);
}

void send_image(int direction)
{

    current_image = (direction == 2) ? current_image->next :
                    current_image->prev;
    printf("current image: %d\n", current_image->image_num);
    int i = current_image->image_num;
    printf("sending image\n");
    unsigned char byte_r, byte_g, byte_b;
    int pixel;
    int x, y;
    //int y_start = (direction == 2) ? 0 : IMG_Y_SIZE;
    //int y_end = (direction == 1) ? 0 : IMG_Y_SIZE;
    //int y_inc = (direction == 2) ? 1 : -1;
    int x_start = (direction == 2) ? 0 : IMG_X_SIZE;
    int x_end = (direction == 1) ? 0 : IMG_X_SIZE;
    int x_inc = (direction == 2) ? 1 : -1;
    //for (y = y_start; (y < y_end && direction == 2) || (y > y_end && direction ==
1); y += y_inc)
    //{
      //  for(x = 0; x < IMG_X_SIZE; x += 1)
        //{
    for (x = x_start; (x < x_end && direction == 2) || (x > x_end && direction == 1);
x += x_inc)
    {
        for(y = 0; y < IMG_Y_SIZE; y += 1)
        {

            // send request
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 16, 1);

            // set requested x and y positions
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 10, x);
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 12, y);

            // get pixel from memory
            byte_r = saved_images[i][0][y][x];
            byte_g = saved_images[i][1][y][x];
            byte_b = saved_images[i][2][y][x];

            if (num_images == 0)
                pixel = 0;
            else
```

```
                pixel = ((byte_r >> 3) << 11) | ((byte_g >> 2) << 5) |((byte_b >> 3)
<< 0);

            // send pixel
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 14, pixel);

            // reset request
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 16, 0);
        }
    }
    printf("sent\n");
}

int save_remote_image()
{
    printf("creating new image\n");

    num_images++;
    printf("num images: %d\n", num_images);

    unsigned char byte_r, byte_g, byte_b;
    int pixel;
    int x, y;

    // find next free slot
    int i;
    for (i = 0; i < MAX_NUM_IMAGES; i++)
    {
        if (!images[i].is_taken)
        {
            current_image = &images[i];
            break;
        }
    }

    if (prev_image == NULL)
    {
        current_image->prev = current_image;
        current_image->next = current_image;
    }
    else if (num_images == 2)
    {
        prev_image->prev = current_image;
        prev_image->next = current_image;
        current_image->prev = prev_image;
        current_image->next = prev_image;
    }
    else if (num_images >= 3)
    {
        current_image->prev = prev_image;
        current_image->next = prev_image->next;
        prev_image->next->prev = current_image;
        prev_image->next = current_image;

    }
    current_image->is_taken = 1;
    current_image->image_num = i;
    prev_image = current_image;
    printf("image num: %d\n", i);
    // return image number
    return i;
}

void save_image()
{
    num_images++;
    unsigned char byte_r, byte_g, byte_b;
    int pixel;
    int x, y;

    // find next free slot
    int i;
    for (i = 0; i < MAX_NUM_IMAGES; i++)
    {
        if (!images[i].is_taken)
        {
            current_image = &images[i];
            break;
        }
    }

    if (prev_image == NULL)
    {
        current_image->prev = current_image;
        current_image->next = current_image;
    }
    else if (num_images == 2)
    {
        prev_image->prev = current_image;
        prev_image->next = current_image;
        current_image->prev = prev_image;
        current_image->next = prev_image;
    }
    else if (num_images >= 3)
    {
        current_image->prev = prev_image;
        current_image->next = prev_image->next;
        prev_image->next->prev = current_image;
        prev_image->next = current_image;

    }
    current_image->is_taken = 1;
    current_image->image_num = i;
    prev_image = current_image;
    printf("saving image: %d/%d\n", i, num_images);
    for (y = 0; y < IMG_Y_SIZE; y += 1)
    {
        for(x = 0; x < IMG_X_SIZE; x += 1)
        {
            // send request
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 4, 1);

            // set requested x and y positions
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 0, x);
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 2, y);

            // get pixel data
            pixel = IORD_16DIRECT(TV_TO_VGA_INST_BASE, 6);

            // enter pixel data into image matrix
            byte_r = (unsigned char) (((pixel >> 11) & 0x1F) << 3);
            byte_g = (unsigned char) (((pixel >>  5) & 0x3F) << 2);
            byte_b = (unsigned char) (((pixel >>  0) & 0x1F) << 3);

            // save pixel
            saved_images[i][0][y][x] = byte_r;
            saved_images[i][1][y][x] = byte_g;
            saved_images[i][2][y][x] = byte_b;
        }
    }
    //write_image(current_image);
    /*struct image *iter = current_image;
    printf("printing images:\n");
    for (i = 0; i < MAX_NUM_IMAGES; i++)
    {
        printf("%d: %d\n", iter->image_num, iter->is_taken);
        iter = iter->next;
    }*/
    printf("saved\n");
    printf("number of images: %d\n", num_images);

}

void send_thumbnails()
{
    printf("sending thubnail\n");
    unsigned char byte_r, byte_g, byte_b;
    int pixel;
    int x, y;
    for (y = 0; y < THUMBNAIL_Y_SIZE; y += 1)
    {
        for(x = 0; x < THUMBNAIL_X_SIZE; x += 1)
        {

            // send request
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 48, 1);

            // set requested x and y positions
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 42, x);
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 44, y);

            // get pixel from memory
            byte_r = 200;
            byte_g = 50;
            byte_b = 250;
            if (num_images == 0)
                pixel = 0;
            else
                pixel = ((byte_r >> 3) << 11) | ((byte_g >> 2) << 5) |((byte_b >> 3)
<< 0);

            // send pixel
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 46, pixel);

            // reset request
            IOWR_16DIRECT(TV_TO_VGA_INST_BASE, 48, 0);
        }
    }
    printf("sent\n");
}




void write_image(struct image *image)
{
    int i = image->image_num;
    unsigned char BITMAP_HEADER[] = {
        0x42, 0x4D, 0xA6, 0xAD, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x36, 0x00, 0x00,
0x00, 0x28, 0x00,
        0x00, 0x00, 0xE4, 0x02, 0x00, 0x00, 0x14, 0x00, 0x00, 0x00, 0x01, 0x00, 0x18,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x70, 0xAD, 0x00, 0x00, 0xEB, 0x0A, 0x00, 0x00, 0xEB, 0x0A, 0x00,
0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    };

    printf("Bitmap header size is %d\n", (int) sizeof(BITMAP_HEADER));
    int file_size = (IMG_X_SIZE*IMG_Y_SIZE) * 3 + sizeof(BITMAP_HEADER);

    // write byte 0 (LSB)
    BITMAP_HEADER[FILE_SIZE_OFFSET+0] = (file_size >>  0) & 0xFF;
    // write byte 1
    BITMAP_HEADER[FILE_SIZE_OFFSET+1] = (file_size >>  8) & 0xFF;
    // write byte 2
    BITMAP_HEADER[FILE_SIZE_OFFSET+2] = (file_size >> 16) & 0xFF;
    // write byte 3 (MSB)
    BITMAP_HEADER[FILE_SIZE_OFFSET+3] = (file_size >> 24) & 0xFF;

    printf("Bitmap file size: %02X %02X %02X %02X\n",
        BITMAP_HEADER[FILE_SIZE_OFFSET+0],
        BITMAP_HEADER[FILE_SIZE_OFFSET+1],
        BITMAP_HEADER[FILE_SIZE_OFFSET+2],
        BITMAP_HEADER[FILE_SIZE_OFFSET+3]
    );

    // write byte 0 (LSB)
    BITMAP_HEADER[PIXEL_WIDTH_OFFSET+0] = (unsigned char)((IMG_X_SIZE >>  0) & 0xFF);
    // write byte 1
    BITMAP_HEADER[PIXEL_WIDTH_OFFSET+1] = (unsigned char)((IMG_X_SIZE >>  8) & 0xFF);
    // write byte 2
    BITMAP_HEADER[PIXEL_WIDTH_OFFSET+2] = (unsigned char)((IMG_X_SIZE >> 16) & 0xFF);
    // write byte 3 (MSB)
    BITMAP_HEADER[PIXEL_WIDTH_OFFSET+3] = (unsigned char)((IMG_X_SIZE >> 24) & 0xFF);

    printf("Bitmap width: %02X %02X %02X %02X\n",
        BITMAP_HEADER[PIXEL_WIDTH_OFFSET],
        BITMAP_HEADER[PIXEL_WIDTH_OFFSET+1],
        BITMAP_HEADER[PIXEL_WIDTH_OFFSET+2],
        BITMAP_HEADER[PIXEL_WIDTH_OFFSET+3]
    );

    // write byte 0 (LSB)
    BITMAP_HEADER[PIXEL_HEIGHT_OFFSET+0] = (unsigned char)((IMG_Y_SIZE >>  0) &
0xFF);
    // write byte 1
    BITMAP_HEADER[PIXEL_HEIGHT_OFFSET+1] = (unsigned char)((IMG_Y_SIZE >>  8) &
0xFF);
    // write byte 2
    BITMAP_HEADER[PIXEL_HEIGHT_OFFSET+2] = (unsigned char)((IMG_Y_SIZE >> 16) &
0xFF);
    // write byte 3 (MSB)
    BITMAP_HEADER[PIXEL_HEIGHT_OFFSET+3] = (unsigned char)((IMG_Y_SIZE >> 24) &
0xFF);

    printf("Bitmap height: %02X %02X %02X %02X\n",
        BITMAP_HEADER[PIXEL_HEIGHT_OFFSET],
        BITMAP_HEADER[PIXEL_HEIGHT_OFFSET+1],
        BITMAP_HEADER[PIXEL_HEIGHT_OFFSET+2],
        BITMAP_HEADER[PIXEL_HEIGHT_OFFSET+3]
    );

    FILE* fh = fopen(IMAGE_PATH, "wb");
    if(fh == NULL) {
      printf("Failed to open the file\n");
      return;
    }

    fwrite(BITMAP_HEADER, 1, sizeof(BITMAP_HEADER), fh);

    printf("Writing data: \n");
    int x, y, c;
    for (y = 0; y < IMG_Y_SIZE; ++y)
    {
        printf(".");
        for(x = 0; x < IMG_X_SIZE; ++x) {
            for (c = 2; c >= 0; c--){
                fwrite(&saved_images[i][c][y][x], 1, sizeof(char), fh);
            }
        }
    }

    printf("\nDone!\n");
    fclose(fh);

    return;
}



jpeg_encoder.c:

#include <io.h>
#include <system.h>
#include <stdio.h>
#include <stdlib.h>
#include "gui.h"
#include "network.h"
#include "misc.h"
#include "jpeg_encoder.h"

#define SAVEPIXEL(data)            IOWR_32DIRECT(SAVEPIXEL_BASE, 0, (data))
#define READQUEUE                  IORD_32DIRECT(SAVEPIXEL_BASE, 0)
#define READBYTE(offset)           IORD_32DIRECT(READRAM_BASE, (offset) * 4)

#define SAVEREG(address, data)     IOWR_32DIRECT(AVALONTRANS_BASE, (address) *4,
(data))
#define READREG(address)           IORD_32DIRECT(AVALONTRANS_BASE, (address) *4)

#define ENC_START_REG       0x0000
#define IMAGE_SIZE_REG      0x0004
#define ENC_STS_REG         0x000C
#define ENC_LENGTH_REG      0x0014
#define QUANTIZER_RAM_LUM   0x0100
#define QUANTIZER_RAM_CHR   0x0200

unsigned char jpeg_image[15000];
unsigned int jpeg_image_size;

const char luminance[] = {
   0x10, 0x0B, 0x0C, 0x0E, 0x0C, 0x0A, 0x10, 0x0E,
   0x0D, 0x0E, 0x12, 0x11, 0x10, 0x13, 0x18, 0x28,
   0x1A, 0x18, 0x16, 0x16, 0x18, 0x31, 0x23, 0x25,
   0x1D, 0x28, 0x3A, 0x33, 0x3D, 0x3C, 0x39, 0x33,
   0x38, 0x37, 0x40, 0x48, 0x5C, 0x4E, 0x40, 0x44,
   0x57, 0x45, 0x37, 0x38, 0x50, 0x6D, 0x51, 0x57,
   0x5F, 0x62, 0x67, 0x68, 0x67, 0x3E, 0x4D, 0x71,
   0x79, 0x70, 0x64, 0x78, 0x5C, 0x65, 0x67, 0x63
};

const char chrominance[] = {
  0x11, 0x12, 0x12, 0x18, 0x15, 0x18, 0x2F, 0x1A,
  0x1A, 0x2F, 0x63, 0x42, 0x38, 0x42, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63
};

//const unsigned int image[];
//unsigned char raster[3][336][232];

//unsigned char raster[3][3][2];
//unsigned char jpeg_image[1];


int compress_jpeg(int num) {
    volatile unsigned int i, j, k, m, temp4, temp1, temp2, temp3;

    i = READREG(ENC_STS_REG);
    printf("enc_status_reg = %x\n", i );

    i = READREG(ENC_START_REG);
    printf("enc_start_reg = %x\n", i );

    i = READREG(IMAGE_SIZE_REG);
    printf("image_size_reg = %x\n", i );

    printf("Writing Image size 336x232 pixels\n");
    SAVEREG(IMAGE_SIZE_REG,0x015000E8);

    i = READREG(IMAGE_SIZE_REG);
    printf("image_size_reg = %x\n", i );

    i = READREG(ENC_LENGTH_REG);
    printf("\nlength = %u\n", i);


    printf(" loading luminance RAM\n");
    for (j=0; j<64; j++) {
        SAVEREG(QUANTIZER_RAM_LUM+j*4, luminance[j]);
        printf(".");
    }
    printf("\n luminance RAM saved\n");

    i = READREG(ENC_STS_REG);
    printf("enc_status_reg = %x\n", i );

    printf(" Loading chrominance RAM\n");

    for (j=0; j<64; j++) {
        SAVEREG(QUANTIZER_RAM_CHR+j*4, chrominance[j]);
        printf(".");
    }
    printf("\n chrominance RAM saved\n");

    i = READREG(ENC_STS_REG);
    printf("enc_status_reg = %x\n", i );
    printf ("Starting encoder\n");
    SAVEREG(ENC_START_REG, 0x7);

    i = READREG(ENC_START_REG);
    printf("enc_start_reg = %x\n", i );

    i = READREG(ENC_STS_REG);
    printf("enc_status_reg = %x\n", i );

    i = READREG(IMAGE_SIZE_REG);
    printf("image_size_reg = %x\n", i );

    i = READREG(ENC_STS_REG);
    printf("enc_status_reg = %x\n", i );

    for (i=0; i<232; i++) { //y
        for (j=0; j<336; j++) { //x
            temp1 = saved_images[num][2][i][j] & 0xff;
            temp2 = (((unsigned int)saved_images[num][1][i][j])<<8) & 0xff00;
            temp3 = (((unsigned int)saved_images[num][0][i][j])<<16) & 0xff0000;
            temp4 = (temp3 >> 16) | temp2 | (temp1 << 16);
            //temp4 = ((unsigned int)saved_images[num][0][i][j])<<16 |
saved_images[num][1][i][j] | saved_images[num][2][i][j];
            //printf("%x\n",temp4);
            SAVEPIXEL(temp4);

            if ((k = READQUEUE)) {
                m =10000;
                while (m--);
            }
            //printf("%d, %d\n",i,j);
        }

        //getchar();
        m =1000;
        while (m--);
        //printf(".");
    }

    printf("\n");
    k = 0;
    //getchar();
    //while (k < 2) {
        k = READREG(ENC_STS_REG);
        printf(".");
        m =100000;
        while (m--);
    //}

    i = READREG(ENC_STS_REG);
    printf("\nenc_status_reg = %x\n", i );
            m =100000;
        while (m--);

    i = READREG(ENC_LENGTH_REG);
    jpeg_image_size = i+2;
    printf("\nlength = %u\n", i);

    m=0;
    //printf("\n %05x:   ",m);
    for (j=0; j<jpeg_image_size; j++) {
        m++;
        k = READBYTE(j);
        jpeg_image[j]=k;
        //printf("%02X ",k);jpeg_image_size
        //printf("%02X ",jpeg_image[j]);
        if (m%16==0) printf("\n");   //printf("\n %05x:   ",m);
    }

    i = READREG(ENC_STS_REG);
    printf("\nenc_status_reg = %x\n", i );

    i = READREG(ENC_LENGTH_REG);
    printf("\nlength = %u\n", i);

    printf("\ndone\n");

    /* alex's code */
    // send a zero then filesize packet
    int zero = 0;
    printf("sending zero packet\n");
    network_send_packet((char*)&zero, sizeof(zero));
    gotosleep(1);
    printf("sending filesize packet\n");
    network_send_packet((char*)&jpeg_image_size, sizeof(jpeg_image_size));
    gotosleep(1);
    printf("sending data\n");
    network_send_packet(jpeg_image, jpeg_image_size);
    /* end alex's code */
  return 0;
}
```

main.c:

```
/* TEAM-SYNTHESIS  */
#include <string.h>
#include <ctype.h>
#include "basic_io.h"
#include "DM9000A.h"
#include <alt_types.h>
#include "alt_up_ps2_port.h"
#include "ps2_keyboard.h"
#include "ps2_mouse.h"
#include "LCD.h"
#include "VGA.h"
#include "gui.h"
#include "jpeg_encoder.h"
#include "misc.h"
```

```c
#include "network.h"

#define IMG_X_SIZE  550
#define IMG_Y_SIZE  360

#define MAX_MSG_LENGTH 800  // must be odd
#define DIV_LINE VGA_HEIGHT - 70
#define NUM_CHAT_LINES 24
#define USER_ROW 26

// Ethernet MAC address.  Choose the last three bytes yourself
unsigned char mac_address[6] = { 0x03, 0x62, 0x6E, 0x11, 0x02, 0x0F  };

unsigned int interrupt_number;

unsigned int receive_buffer_length;
unsigned char receive_buffer[1600];
unsigned char user_name[MAX_MSG_LENGTH];
unsigned char chat_buffer[NUM_CHAT_LINES][MAX_MSG_LENGTH];

KB_CODE_TYPE decode_mode;

#define UDP_PACKET_PAYLOAD_OFFSET 42
#define UDP_PACKET_LENGTH_OFFSET 38

#define UDP_PACKET_PAYLOAD (transmit_buffer + UDP_PACKET_PAYLOAD_OFFSET)

unsigned char transmit_buffer[MAX_MSG_LENGTH];
unsigned char transmit_buffer_template[] = {
  // Ethernet MAC header
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // Destination MAC address
  0x01, 0x60, 0x6E, 0x11, 0x02, 0x0F, // Source MAC address
  0x08, 0x00,                        // Packet Type: 0x800 = IP

  // IP Header
  0x45,                // version (IPv4), header length = 20 bytes
  0x00,                // differentiated services field
  0x00,0x9C,           // total length: 20 bytes for IP header +
                       // 8 bytes for UDP header + 128 bytes for payload
  0x3d, 0x35,          // packet ID
  0x00,                // flags
  0x00,                // fragment offset
  0x80,                // time-to-live
  0x11,                // protocol: 11 = UDP
  0xa3,0x43,           // header checksum: incorrect
  0xc0,0xa8,0x01,0x01, // source IP address
  0xc0,0xa8,0x01,0xff, // destination IP address

  // UDP Header
  0x67,0xd9, // source port port (26585: garbage)
  0x27,0x2b, // destination port (10027: garbage)
  0x00,0x88, // length (136: 8 for UDP header + 128 for data)
  0x00,0x00, // checksum: 0 = none

  // UDP payload
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67
};

//int send_value;
//int *sending;
int x, y;
int x_pos, y_pos;

int remote_img_num;
int remote_img_num2;

void fill_buffer() {
    UDP_PACKET_PAYLOAD[0] = 3;
    UDP_PACKET_PAYLOAD[1] = x & 0xff;
    UDP_PACKET_PAYLOAD[2] = x >> 8;
    UDP_PACKET_PAYLOAD[3] = y & 0xff;
    UDP_PACKET_PAYLOAD[4] = y >> 8;
    int p = 0;
    while (p < (MAX_MSG_LENGTH-7) && x < IMG_X_SIZE && y < IMG_Y_SIZE)
    {

        unsigned char byte_r = saved_images[remote_img_num][0][x][y];
        unsigned char byte_g = saved_images[remote_img_num][1][x][y];
        unsigned char byte_b = saved_images[remote_img_num][2][x][y];


        UDP_PACKET_PAYLOAD[p+7] = (((byte_g >> 2) & 0x7) << 5) | (byte_b >> 3);
        p++;
        UDP_PACKET_PAYLOAD[p+7] = ((byte_r >> 3) << 3) | (byte_g >> 5);
        p++;
        x++;
        if (x == IMG_X_SIZE)
        {
            x = 0;
            y++;
            //printf("y: %d, %d, %d, %d\n", y, byte_r, byte_g, byte_b);
            //printf("%d, %d\n", UDP_PACKET_PAYLOAD[p+6], UDP_PACKET_PAYLOAD[p+5]);
        }
    }
    if (y == IMG_Y_SIZE)
    {
        printf("Sent\n");
        y = -1;
    }

    //printf("Sent (%d)\n", p);
    UDP_PACKET_PAYLOAD[5] = p & 0xff;
    UDP_PACKET_PAYLOAD[6] = p >> 8;
    TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + (p+6) + 1);

}

static void ethernet_interrupt_handler() {
  unsigned int receive_status;

  receive_status = ReceivePacket(receive_buffer, &receive_buffer_length);

  if (receive_status == DMFE_SUCCESS) {

#if 1

    /*printf("\n\nReceive Packet Length = %d", receive_buffer_length);
    for(i=0;i<receive_buffer_length;i++) {
      if (i%8==0) printf("\n");
      printf("0x%.2X,", receive_buffer[i]);
    }
    printf("\n");*/
#endif

    if (receive_buffer_length >= 14) {
      //  A real Ethernet packet
      if (receive_buffer[12] == 8 && receive_buffer[13] == 0 &&
      receive_buffer_length >= 34) {
    // An IP packet
    if (receive_buffer[23] == 0x11)
    {
      // A UDP packet
      if (receive_buffer_length >= UDP_PACKET_PAYLOAD_OFFSET)
      {
//-----------------------------------------------------------------------------
        //int i = atoi(receive_buffer + UDP_PACKET_PAYLOAD_OFFSET);
        int request = (receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[0];

        // type 1 = begin *sending
        if (request == 1)// && *sending != 1)
        {
            remote_img_num2 = save_remote_image();
            x_pos = 0;
            y_pos = 0;
            printf("Receiving\n");
            //*sending = 1;
            // type 2 = ack
            UDP_PACKET_PAYLOAD[0] = 2;
            TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + 1 + 1);
        }
        // type 3 = *sending pixel data
        else if (request == 3)// && *sending == 1)
        {
            x_pos = (receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[1];
            x_pos |=  (receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[2] << 8;
            y_pos = (receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[3];
            y_pos |=  (receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[4] << 8;
            int num = (receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[5];
            num |=  (receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[6] << 8;
            //printf("num (%d): ", num);
            int i = 7;
            while (i < num+7 && i < MAX_MSG_LENGTH && y_pos < IMG_Y_SIZE)
            {
                unsigned char byte_r, byte_g, byte_b;
                byte_b = (receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[i] & 0x1f << 3;
                byte_g = (((receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[i+1] & 0x7)
<< 5)
                    | (((receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[i] >> 5) << 2);
                byte_r = ((receive_buffer + UDP_PACKET_PAYLOAD_OFFSET)[i+1] >> 3) <<
3;


                saved_images[remote_img_num2][0][x_pos][y_pos] = byte_r;
                saved_images[remote_img_num2][1][x_pos][y_pos] = byte_g;
                saved_images[remote_img_num2][2][x_pos][y_pos] = byte_b;

                i += 2;
                x_pos++;
                //printf("\nx_pos = %d\n", x);
                if (x_pos == IMG_X_SIZE)
                {
                    x_pos = 0;
                    //printf("y_pos = %d, %d, %d, %d\n", y_pos, byte_r, byte_g,
byte_b);
                    y_pos++;
                }
            }
            if (y == IMG_Y_SIZE)
            {
                printf("setting it to 0\n");
               // *sending = 0;
            }
            else
            {
                // type 2 = received
                UDP_PACKET_PAYLOAD[0] = 2;
                TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + 1 + 1);
            }
            /*while (y != -1 && timer < 500000)
            {
                timer++;
            }
            printf("Done\n");
            sending = 0;*/
        }
        else if (request == 2 && y != -1) //&& *sending == 1 && y != -1)
        {
            //timer = 0;
            fill_buffer();
        }
        // type 3 = finished *sending
        else if (request == 4)// && *sending == 1)
        {
            printf("Finish receiving\n");
           // *sending = 0;
        }




//-----------------------------------------------------------
      }
    } else {
      printf("Received non-UDP packet\n");
    }
      } else {
    printf("Received non-IP packet\n");
      }
    } else {
      printf("Malformed Ethernet packet\n");
    }

  } else {
    printf("Error receiving packet\n");
  }

  /* Display the number of interrupts on the LEDs */
  interrupt_number++;
  outport(SEG7_DISPLAY_BASE, interrupt_number);

  /* Clear the DM9000A ISR: PRS, PTS, ROS, ROOS 4 bits, by RW/C1 */
  dm9000a_iow(ISR, 0x3F);

  /* Re-enable DM9000A interrupts */
  dm9000a_iow(IMR, INTR_set);
}


void compute_checksum()
{
    // increment ip id
    int packet_id = (transmit_buffer[18] << 8) + transmit_buffer[19] + 1;
    transmit_buffer[18] = packet_id & 0xff00 >> 8;
    transmit_buffer[19] = packet_id & 0xff;
    printf("packed_id: %x\n", packet_id);

    int sum = 0;
    int i;
    for (i = 0; i < 10; i++)
    {
        if (i != 5)
            sum += (transmit_buffer[14 + i*2] << 8) + transmit_buffer[15 + i*2];
    }
    // turn to one's complement
    sum = (sum & 0xffff) + (sum >> 16);

    // complement
    sum = ~sum;

    transmit_buffer[24] = sum >> 8;
    transmit_buffer[25] = sum & 0x00ff;
    printf("checksum: %x\n", sum);
}

void handel_input()
{
  unsigned int curMsgChar = 0;
  alt_u8 key = 0;
  int status = 0;
  unsigned int packet_length;
  int capital = 0;
  UDP_PACKET_PAYLOAD[0] = 0;
  for (;;) {
    // wait for the user's input and get the make code
    status = read_make_code(&decode_mode, &key);
    if (key == 0xf0)
        printf("key up\n");

    if (status == PS2_SUCCESS) {

      // print out the result
      switch ( decode_mode ){
      case KB_ASCII_MAKE_CODE :
        key = (capital) ? key : tolower(key);
        printf("%c", key);

    if (curMsgChar < MAX_MSG_LENGTH) {
      int i;
      int len = strlen(UDP_PACKET_PAYLOAD);
      UDP_PACKET_PAYLOAD[len+1] = 0;
      for (i = len; len > 0 && i > curMsgChar; i--)
      {
        UDP_PACKET_PAYLOAD[i] = UDP_PACKET_PAYLOAD[i-1];
      }
      UDP_PACKET_PAYLOAD[curMsgChar] = key;
      curMsgChar++;
    }
    break ;
      case KB_LONG_BINARY_MAKE_CODE :
    // fall through
      case KB_BINARY_MAKE_CODE :
    switch (key) {
    case  0x5a: //enter key: send the msg
      packet_length = 8 + 1;//curMsgChar;
      transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
      transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length & 0xff;
      //compute_checksum();

      /*alt_u8 x, y;
      for (y = 1; y < 5; y++)
      {
        for (x = 1; x < 4; x++)
        {
            UDP_PACKET_PAYLOAD[0] = x;
            printf("%d\n", x);
            TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + 1 + 1);
        }
      }*/

      //int i;
      //for (i = 0; i < 1000; i++)
      //{
        if (capital == 2 && get_num_images() > 0)
        {
            remote_img_num = get_current_img_num();
            printf("Sending\n");
            y = 0;
            x = 0;
            UDP_PACKET_PAYLOAD[0] = 1;
            TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + 1 + 1);
        }
        if (capital == 1 && get_num_images() > 0)
        {

           printf("Compressing Image\n");

           compress_jpeg(get_current_img_num());


        }
        else if (get_num_images() < MAX_NUM_IMAGES)
        {
            save_image();
        }

      return;
    case 0x29: //space key
      printf("key: space\n");
      freeze_toggle();
      break;
    default:
      // delete char if encounter delete key
      if (key == 0x66)
      {
        printf("key: delete\n");
        if (get_num_images() > 0)
            delete_current_image();
      }
```

```
    }
        // shift encountered
        if (key == 0x12)
        {
          printf("key: left shift\n");
          capital = 1;
        }
        if (key == 0x59)
        {
          printf("key: right shift\n");
          capital = 2;
        }
      if (key == 0x74)
      {
        printf("key: right\n");
        if (get_num_images() >= 1)
            send_image(2);
      }
      if (key == 0x6B)
      {
        printf("key: left\n");
        if (get_num_images() >= 1)
            send_image(1);
      }
      if (key == 0x75)
      {
        printf("key: up\n");
        full_screen(0);
      }
      if (key == 0x72)
      {
        printf("key: down\n");
        full_screen(1);
      }

    break ;
      case KB_BREAK_CODE :
        // shift encountered
        if (key == 0x59 || key == 0x12)
        {
          capital = 0;
        }
    //printf("break: %c\n", key);
      default :
    break ;
      }
    }
  }
}


int main(int argc, char* argv[])
{
  /* Added by alex */
  alex_init();
  network_set_source_ip(10, 0, 0, 15);
  network_set_dest_ip(10, 0, 0, 1);
  network_set_source_port(9931);
  network_set_dest_port(9930);
  /* end alex */

  //send_value = 0;
  //sending = &send_value;
  int i;
  for (i = 0; i < 128; i++)
  {
    transmit_buffer[i] = transmit_buffer_template[i];
  }

  int curMsgChar = 0;


  // Initalize the DM9000 and the Ethernet interrupt handler
  //DM9000_init(mac_address);
  //interrupt_number = 0;
  alt_irq_register(DM9000A_IRQ, NULL, (void*)ethernet_interrupt_handler);

  printf("Ready to send messages\n");


  // Clear the payload
  for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0; curMsgChar--) {
    UDP_PACKET_PAYLOAD[curMsgChar] = 0;
  }


  for (;;)
  {
    handel_input();
  }

  printf("Program terminated normally\n");
  return 0;

 ErrorExit:
  printf("Program terminated with an error condition\n");

  return 1;
}

misc.c:

#include <time.h>
#include "basic_io.h"
#include "DM9000A.h"
#include "LCD.h"
#include "network.h"

void gotosleep(unsigned int seconds) {
  time_t before, now;
  before = now = time(NULL);

  int elapsed = 0;
  while(1) {
    now = time(NULL);
    if(before != now) {
      before = now;
      if(elapsed++ == seconds) return;
    }
  }
} // end gotosleep

// Initialize the display and ethernet interface
void alex_init() {
  // Initialize the LCD and display a welcome message
  LCD_Init();
  LCD_Show_Text("TEAM-SYNTHESIS");

  // Clear the LEDs to zero (will display interrupt count)
  outport(SEG7_DISPLAY_BASE, 0);

  printf("Network Transfer: Team-Synthesis (c) 2010\n");
  //printf("Initializing ethernet\n");

  network_init();
} // end init
```

network.c:

```
#include <string.h>
#include <stdio.h>
#include <alt_types.h>
#include "DM9000A.h"
#include "network.h"
#include "misc.h"
#include "basic_io.h"

const unsigned char SOURCE_MAC_ADDR[] = { 0x01, 0x60, 0x6E, 0x11, 0x02, 0x41 };
//const unsigned char DEST_MAC_ADDR[] = { 0x00, 0x1A, 0x80, 0xD6, 0x4D, 0xC1 };
const unsigned char DEST_MAC_ADDR[] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

const int ETHERNET_HEADER_SIZE = 14;
const int IP_HEADER_SIZE = 20;
const int UDP_HEADER_SIZE = 8;

unsigned int network_interrupt_number;

unsigned int network_rx_buffer_length;
unsigned char network_rx_buffer[1600];

extern unsigned char network_udp_packet[];

// Send a packet of data with the specified size, if over MTU split
// the packet into chunks
```

```
void network_send_packet(const char* data, const unsigned int size) {
  static unsigned int bytesSent = 0;
  Packet packet;

  int i;
  for(i=0; i<size; i+=NETWORK_MAX_PACKET_SIZE) {
    // truncate the packet if needed
    unsigned int packetSize = (size-i);
    if(packetSize > NETWORK_MAX_PACKET_SIZE) packetSize = NETWORK_MAX_PACKET_SIZE;
    network_format_packet(&packet, data+i, packetSize);

    int res = TransmitPacket(packet.data, packet.size);
    if (res == DMFE_SUCCESS) {
      //printf("Packet sent: %d\n", packet.size);
      bytesSent += size;
      outport(SEG7_DISPLAY_BASE, bytesSent);
    } else {
      printf("\nNetwork Failure: Transmission failed\n");
    }

    // free memory allocated in network_format_packet
    free(packet.data);
  }

} // end network_send_packet

// Format a packet for delivery in a Packet structure.  If size is less
// than MIN_PACKET_SIZE, add '0' bytes of padding to the packet.  Caller
// must free packet->data.
void network_format_packet(
    Packet* packet, const char* payload, const unsigned int payloadSize) {

  // add extra byte for null termination
  int size = NETWORK_UDP_PACKET_SIZE + payloadSize;
  if(size < MIN_PACKET_SIZE) size = MIN_PACKET_SIZE;
  char* packetData = (char*) malloc(size);
  // clear the packet
  memset(packetData, 0, size);
  memcpy(packetData, network_udp_packet, NETWORK_UDP_PACKET_SIZE);

  /* Set MAC addresses */
  memcpy(packetData, DEST_MAC_ADDR, 6);
  memcpy(packetData + 6, SOURCE_MAC_ADDR, 6);
  //memcpy(packetData, SOURCE_MAC_ADDR, 6);
  //memcpy(packetData + 6, DEST_MAC_ADDR, 6);

  /* Set the UDP Length: Header plus data */
  unsigned short udpLength = UDP_HEADER_SIZE + payloadSize;
  //printf("%10s=%d\n", "UDP LENGTH", udpLength);

  packetData[HEADER_OFFSET_UDP_LENGTH_H] = udpLength >> 8;
  packetData[HEADER_OFFSET_UDP_LENGTH_L] = udpLength & 0xFF;

  /* Set UDP Payload: copy the message to the udp packet */
  memcpy(packetData + NETWORK_UDP_PACKET_SIZE, payload, payloadSize);
  /* Set the IP Length: Header plus UDP Length */
  unsigned short ipLength = IP_HEADER_SIZE + udpLength;
  //printf("%10s=%d\n", "IP LENGTH", ipLength);

  packetData[HEADER_OFFSET_IP_TOTAL_LENGTH_H] = ipLength >> 8;
  packetData[HEADER_OFFSET_IP_TOTAL_LENGTH_L] = ipLength & 0xFF;

  /* Set the IP Checksum */
  int ipOffset = ETHERNET_HEADER_SIZE;
  short ipChecksum =
    network_calc_ip_checksum(packetData+ipOffset, IP_HEADER_SIZE);

  packetData[HEADER_OFFSET_IP_CHECKSUM_H] = ipChecksum >> 8;
  packetData[HEADER_OFFSET_IP_CHECKSUM_L] = ipChecksum & 0xFF;

  char padding = (udpLength%2==1);
  // enable padding if udplength is odd
  //printf("Padding=%d, udpLength=%d\n", padding, udpLength);
  char* udpPayload = (packetData + HEADER_OFFSET_UDP);
  // byte 3 is the MSB of the address
  char* sourceIp = packetData+HEADER_OFFSET_SOURCE_IP_3;
  char* destIp = packetData+HEADER_OFFSET_DEST_IP_3;

  u16 udpChecksum =
    network_calc_udp_checksum (udpLength, sourceIp, destIp, padding, udpPayload);

  packetData[HEADER_OFFSET_UDP_CHECKSUM_H] = udpChecksum >> 8;
  packetData[HEADER_OFFSET_UDP_CHECKSUM_L] = udpChecksum & 0xFF;

  packet->data = packetData;
  //HEADER_OFFSET_UDP_DATA + payloadSize;
  packet->size = size;
} // end network_format_packet

void network_set_source_ip(unsigned char a3, unsigned char a2, unsigned char a1,
unsigned char a0) {
  network_udp_packet[HEADER_OFFSET_SOURCE_IP_3] = a3;
  network_udp_packet[HEADER_OFFSET_SOURCE_IP_2] = a2;
  network_udp_packet[HEADER_OFFSET_SOURCE_IP_1] = a1;
  network_udp_packet[HEADER_OFFSET_SOURCE_IP_0] = a0;
} // end network_set_source_ip

void network_set_dest_ip(unsigned char a3, unsigned char a2, unsigned char a1,
unsigned char a0) {
  network_udp_packet[HEADER_OFFSET_DEST_IP_3] = a3;
  network_udp_packet[HEADER_OFFSET_DEST_IP_2] = a2;
  network_udp_packet[HEADER_OFFSET_DEST_IP_1] = a1;
  network_udp_packet[HEADER_OFFSET_DEST_IP_0] = a0;
} // end network_set_dest_ip

// add up an array of byte values to calculate the checksum
unsigned short network_calc_ip_checksum(unsigned char ipHeader[], const int SIZE) {
  unsigned int checksum = 0;
  int i;
  for(i=0; i<SIZE; i+=2) {
    // even bytes are high order
    int val = (ipHeader[i] << 8) | ipHeader[i+1];
    checksum += val;
  }

  // calc ones comp sum by adding upper 16 bits
  checksum += ((unsigned int) checksum) >> 16;
  // get complement
  return ~checksum & 0xFFFF;
} // end network_calc_ip_checksum

u16 network_calc_udp_checksum (u16 len_udp, unsigned char* src_addr, unsigned char*
dest_addr, char padding, unsigned char* buff) {
  u16 prot_udp = 17;
  u16 padd = 0;
  u16 word16;
  u32 sum;
  int i;

  // Find out if the length of data is even or odd number. If odd,
  // add a padding byte = 0 at the end of packet
  if ((padding & 1) == 1) {
    padd = 1;
    buff[len_udp] = 0;
  }

  //initialize sum to zero
  sum = 0;

  // make 16 bit words out of every two adjacent 8 bit words and
  // calculate the sum of all 16 bit words
  for (i = 0; i < len_udp + padd; i += 2) {
    word16 = ((buff[i] << 8) & 0xFF00) + buff[i + 1];
    sum = sum + (unsigned long) word16;
  }

  // add the UDP pseudo header which contains the IP source and destination addresses
  for (i = 0; i < 4; i += 2) {
    word16 = ((src_addr[i] << 8) & 0xFF00) + src_addr[i + 1];
    sum = sum + word16;
  }

  for (i = 0; i < 4; i = i + 2) {
    word16 = ((dest_addr[i] << 8) & 0xFF00) + dest_addr[i + 1];
    sum = sum + word16;
  }

  // the protocol number and the length of the UDP packet
  sum = sum + prot_udp + len_udp;

  // keep only the last 16 bits of the 32 bit calculated sum and add the carries
  while (sum >> 16)
    sum = (sum & 0xFFFF) + (sum >> 16);
  // Take the one's complement of sum
  sum = ~sum;

  return ((u16) sum);
} // end network_calc_udp_checksum

// Read a binary file using the altera host-fs and send it over the network
void network_send_file(const char* filename) {
  char filepath[256];
  snprintf(filepath, sizeof(filepath), "/mnt/host/%s", filename);

  FILE* fh = fopen (filepath, "rb");
  if (fh == NULL) {
    printf("Cannot open file: %s\n", filepath);
    exit(1);
  }

  fseek(fh, 0, SEEK_END);
  long fileSize = ftell(fh);
  fseek(fh, 0, SEEK_SET); // rewind

  // send a zero then filesize packet
  int zero = 0;
  printf("sending zero packet\n");
  network_send_packet((char*)&zero, sizeof(zero));
  gotosleep(2);
  printf("sending filesize packet\n");
  network_send_packet((char*)&fileSize, sizeof(fileSize));

  // send the file in 1KB increments
  unsigned char buff[1024];
  int i=0;
  for(i=0; i<sizeof(buff); ++i) buff[i] = (char)i;
  size_t bytesRead = 0;

  do {
    size_t readSize = fread(buff, 1, sizeof(buff), fh);

#ifdef DEBUG
    for(i = 0; i < sizeof(buff); i++) {
      if(i != 0 && i % 16 == 0) printf("\n");
      if(i == 0 || i % 16 == 0) printf("%06X: ", (int)(bytesRead+i));
      printf("%02X ", buff[i]);
    }
    printf("\n");
#endif // DEBUG

    if(readSize > 0) network_send_packet(buff, readSize);
    bytesRead += readSize;

#ifndef DEBUG
    printf("%d bytes sent\n", (int) bytesRead);
#endif

  } while(!feof(fh));
  fclose (fh);
} // end network_send_file

// big to little endian short
unsigned short network_btoles(unsigned short val) {
  return (val << 8) | (val >> 8);
} // end btoles

// check for ascii values out of range
int containsNonPrintableAscii(const char* msg) {
  int i;
  int bufflen = strlen(msg);
  for(i=0; i<bufflen; ++i) {
    unsigned char c = msg[i];
    if((c < 32 && c != 10) || c > 127) return 1;
  }
  return 0;
} // containsNonPrintableAscii

void network_interrupt_handler() {
  unsigned int receive_status;

  receive_status = ReceivePacket(network_rx_buffer, &network_rx_buffer_length);

  if (receive_status == DMFE_SUCCESS) {

    //printf("\n\nReceive Packet Length = %d\n", network_rx_buffer_length);

    if (network_rx_buffer_length >= 14) {
      //  A real Ethernet packet
      if (network_rx_buffer[12] == 8 && network_rx_buffer[13] == 0 &&
          network_rx_buffer_length >= 34) {
        // An IP packet
        if (network_rx_buffer[23] == 0x11) {
          // A UDP packet
          char* msg = network_rx_buffer+HEADER_OFFSET_UDP_DATA;
          if( containsNonPrintableAscii(msg) ) {
             //printf("message contained non printable ascii\n");
          } else if (network_rx_buffer_length >= HEADER_OFFSET_UDP_DATA) {
            int bufflen = strlen(msg)+1;
            char* buff = (char*) malloc(bufflen);
            strncpy(buff, msg, bufflen);
            //printf("Received: %s\n", buff);
          }
        } else {
          //printf("Received non-UDP packet\n");
        }
      } else {
        //printf("Received non-IP packet\n");
      }
    } else {
      //printf("Malformed Ethernet packet\n");
    }
  } else {
    //printf("Error receiving packet\n");
  }

  /* Display the number of interrupts on the LEDs */
  network_interrupt_number++;
  outport(SEG7_DISPLAY_BASE, network_interrupt_number);

  /* Clear the DM9000A ISR: PRS, PTS, ROS, ROOS 4 bits, by RW/C1 */
  dm9000a_iow(ISR, 0x3F);

  /* Re-enable DM9000A interrupts */
  dm9000a_iow(IMR, INTR_set);
} // network_interrupt_handler

// Initalize the DM9000 and the Ethernet interrupt handler
void network_init() {
  // copy the constant into a buffer to avoid warning
  char buffer[6];
  memcpy(buffer, SOURCE_MAC_ADDR, sizeof(buffer));
  DM9000_init(buffer);
  network_interrupt_number = 0;
  //alt_irq_register(DM9000A_IRQ, NULL, (void*)network_interrupt_handler);
} // end network_init

// write the port to the network_udp_packet global
void network_set_source_port(unsigned short port) {
  network_udp_packet[HEADER_OFFSET_SOURCE_PORT_H] = (port & 0xFF00) >> 8;
  network_udp_packet[HEADER_OFFSET_SOURCE_PORT_L] = port & 0xFF;
} // end network_set_source_port

// write the port to the network_udp_packet global
void network_set_dest_port(unsigned short port) {
  network_udp_packet[HEADER_OFFSET_DEST_PORT_H] = (port & 0xFF00) >> 8;
  network_udp_packet[HEADER_OFFSET_DEST_PORT_L] = port & 0xFF;
} // end network_set_dest_port

unsigned char network_udp_packet[] = {
  // Ethernet MAC header
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     //   0-5: Destination MAC address
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     //  6-11: Source MAC address
  0x08, 0x00,                             // 12-13: Packet Type: 0x800 = IP

  // IP Header
  0x45,                //     0-7: version (IPv4), 32 bit words (5 words = 20 bytes)
  0x00,                //    8-15: differentiated services field
                       //   16-18: unused
  0x0F,0x0E,           //   19-31: Total Length: 20 bytes for IP header +
                       //          8 bytes for UDP header + x bytes for payload

  0x00, 0x00,          //   32-47: packet ID
  0x40,                //   48-50: flags (don't fragment)
  //0x00,                //   48-50: flags (none)
  0x00,                //   51-63: fragment offset
  0x40,                //   64-71: time-to-live
  0x11,                //   72-79: protocol: 11 = UDP
  0x00,0x00,           //   80-95: header checksum
  0x00,0x00,0x00,0x00, //  96-127: source IP address
  0x00,0x00,0x00,0x00, // 128-159: destination IP address

  // UDP Header
  0x00,0x00,           //    0-15: source port port (26585: garbage)
  0x00,0x00,           //   16-31: destination port (10000: garbage)
  0x00,0x00,           //   32-47: length (136: 8 for UDP header + 128 for data)
  0x00,0x00           //   48-63: checksum: 0 = none
};

ps2_keyboard.c:

#include "ps2_keyboard.h"

#define NUM_SCAN_CODES  102

////////////////////////////////////////////////////////////////////
// Table of scan code, make code and their corresponding values
// These data are useful for developing more features for the keyboard
//
alt_u8 *key_table[NUM_SCAN_CODES] = {
  "A", "B", "C", "D", "E", "F", "G", "H",
  "I", "J", "K", "L", "M", "N", "O", "P",
  "Q", "R", "S", "T", "U", "V", "W", "X",
  "Y", "Z", "0", "1", "2", "3", "4", "5",
  "6", "7", "8", "9", "`", "-", "=", "\\",
  "BKSP", "SPACE", "TAB", "CAPS", "L SHFT", "L CTRL", "L GUI", "L ALT",
  "R SHFT", "R CTRL", "R GUI", "R ALT", "APPS", "ENTER", "ESC", "F1",
  "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9",
  "F10", "F11", "F12", "SCROLL", "[", "INSERT", "HOME", "PG UP",
  "DELETE", "END", "PG DN", "U ARROW", "L ARROW", "D ARROW", "R ARROW", "NUM",
  "KP /", "KP *", "KP -", "KP +", "KP ENTER", "KP .", "KP 0", "KP 1",
  "KP 2", "KP 3", "KP 4", "KP 5", "KP 6", "KP 7", "KP 8", "KP 9",
  "]", ";", "'", ",", ".", "/"
};

alt_u8 ascii_codes[NUM_SCAN_CODES] = {
  'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
  'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
  'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
  'Y', 'Z', '0', '1', '2', '3', '4', '5',
  '6', '7', '8', '9', '`', '-', '=', 0,
  0x08, 0, 0x09, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0x0A, 0x1B,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, '[', 0, 0,
  0, 0x7F, 0, 0, 0, 0, 0, 0,
  0, '/', '*', '-', '+', 0x0A, '.', '0', '1',
  '2', '3', '4', '5', '6', '7', '8', '9',
  ']', ';', '\'', ',', '.', '/'
};

alt_u8 single_byte_make_code[NUM_SCAN_CODES] = {
  0x1C, 0x32, 0x21, 0x23, 0x24, 0x2B, 0x34, 0x33,
  0x43, 0x3B, 0x42, 0x4B, 0x3A, 0x31, 0x44, 0x4D,
  0x15, 0x2D, 0x1B, 0x2C, 0x3C, 0x2A, 0x1D, 0x22,
  0x35, 0x1A, 0x45, 0x16, 0x1E, 0x26, 0x25, 0x2E,
  0x36, 0x3D, 0x3E, 0x46, 0x0E, 0x4E, 0x55, 0x5D,
  0x66, 0x29, 0x0D, 0x58, 0x12, 0x14,    0, 0x11,
  0x59,    0,    0,    0,    0, 0x5A, 0x76, 0x05,
  0x06, 0x04, 0x0C, 0x03, 0x0B, 0x83, 0x0A, 0x01,
  0x09, 0x78, 0x07, 0x7E, 0x54,    0,    0,    0,
  0,    0,    0,    0,    0,    0,    0, 0x77,
  0, 0x7C, 0x7B, 0x79,    0, 0x71, 0x70, 0x69,
  0x72, 0x7A, 0x6B, 0x73, 0x74, 0x6C, 0x75, 0x7D,
  0x5B, 0x4C, 0x52, 0x41, 0x49, 0x4A };

alt_u8 multi_byte_make_code[NUM_SCAN_CODES] = {
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0x1F, 0,
  0, 0x14, 0x27, 0x11, 0x2F, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0x70, 0x6C, 0x7D,
  0x71, 0x69, 0x7A, 0x75, 0x6B, 0x72, 0x74, 0,
  0x4A, 0, 0, 0, 0x5A, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0 };
/////////////////////////////////////////////////////////////////////

// States for the Keyboard Decode FSM
typedef enum
  {
    STATE_INIT,
    STATE_LONG_BINARY_MAKE_CODE,
    STATE_BREAK_CODE ,
    STATE_DONE
  } DECODE_STATE;

//helper function for get_next_state
alt_u8 get_multi_byte_make_code_index(alt_u8 code)
{
  alt_u8 i;
  for (i = 0; i < NUM_SCAN_CODES; i++ ) {
    if ( multi_byte_make_code[i] == code )
      return i;
  }
  return NUM_SCAN_CODES;
}

//helper function for get_next_state
alt_u8 get_single_byte_make_code_index(alt_u8 code)
{
  alt_u8 i;
  for (i = 0; i < NUM_SCAN_CODES; i++ ) {
    if ( single_byte_make_code[i] == code )
      return i;
  }
  return NUM_SCAN_CODES;
}

//helper function for read_make_code
/* FSM Diagram (Main transitions)
 * Normal bytes: bytes that are not 0xF0 or 0xE0
 ______
 |     |
 |     |
 |    INIT ------ 0xF0 ----> BREAK CODE
 |     |                     /  |
 |     |                    /   |
 |    0xE0                 /    |
 Normal |                  /    Normal
 |     |      ----0xF0--->      |
 |     V     /                  |
 |    LONG  /                   V
 | MAKE/BREAK --- Normal ----> DONE
 |    CODE                       ^
 X-------------------------------|

*/

DECODE_STATE get_next_state(DECODE_STATE state,
                        alt_u8 byte,
                        KB_CODE_TYPE *decode_mode,
                        alt_u8 *buf)
{
  DECODE_STATE next_state = STATE_INIT;
  alt_u16 idx = NUM_SCAN_CODES;
  switch (state) {
  case STATE_INIT:
    if ( byte == 0xE0 ) {
      next_state = STATE_LONG_BINARY_MAKE_CODE;
    } else if (byte == 0xF0) {
      next_state = STATE_BREAK_CODE;
    } else {
      idx = get_single_byte_make_code_index(byte);
      if ( (idx < 40 || idx == 68 || idx > 79) && ( idx != NUM_SCAN_CODES ) ) {
       *decode_mode = KB_ASCII_MAKE_CODE;
       *buf= ascii_codes[idx];
      } else {
       *decode_mode = KB_BINARY_MAKE_CODE;
       *buf = byte;
      }
      next_state = STATE_DONE;
    }
    break;
  case STATE_LONG_BINARY_MAKE_CODE:
    if ( byte != 0xF0 && byte!= 0xE0) {
      *decode_mode = KB_LONG_BINARY_MAKE_CODE;
      *buf = byte;
      next_state = STATE_DONE;
    } else {
      next_state = STATE_BREAK_CODE;
    }
    break;
  case STATE_BREAK_CODE:
    if ( byte != 0xF0 && byte != 0xE0) {
      *decode_mode = KB_BREAK_CODE;
      *buf = byte;
      next_state = STATE_DONE;
    } else {
      next_state = STATE_BREAK_CODE;
    }
    break;
  default:
    *decode_mode = KB_INVALID_CODE;
    next_state = STATE_INIT;
  }
  return next_state;
}

int read_make_code(KB_CODE_TYPE *decode_mode, alt_u8 *buf)
{
  alt_u8 byte = 0;
  int status_read =0;
  *decode_mode = KB_INVALID_CODE;
  DECODE_STATE state = STATE_INIT;
  do {
    status_read = read_data_byte_with_timeout(&byte, 0);
    //FIXME: When the user press the keyboard extremely fast, data may get
    //occasionally get lost

    if (status_read == PS2_ERROR)
      return PS2_ERROR;

    state = get_next_state(state, byte, decode_mode, buf);
  } while (state != STATE_DONE);

  return PS2_SUCCESS;
}

alt_u32 set_keyboard_rate(alt_u8 rate)
{
  alt_u8 byte;
  // send the set keyboard rate command
  int status_send = write_data_byte_with_ack(0xF3, DEFAULT_PS2_TIMEOUT_VAL);
  if ( status_send == PS2_SUCCESS ) {
    // we received ACK, so send out the desired rate now
    status_send = write_data_byte_with_ack(rate & 0x1F,
                                   DEFAULT_PS2_TIMEOUT_VAL);
  }
  return status_send;
}

alt_u32 reset_keyboard()
```

```
{
  alt_u8 byte;
  // send out the reset command
  int status = write_data_byte_with_ack(0xff, DEFAULT_PS2_TIMEOUT_VAL);
  if ( status == PS2_SUCCESS) {
    // received the ACK for reset, now check the BAT result
    status = read_data_byte_with_timeout(&byte, DEFAULT_PS2_TIMEOUT_VAL);
    if (status == PS2_SUCCESS && byte == 0xAA) {
      // BAT succeed
    } else {
      // BAT failed
      status == PS2_ERROR;
    }
  }
  return status;
}
```

PC Software (linux)

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <errno.h>
#include <stdint.h>

#define BUFFER_SIZE 1024
#define MAX_FILE_SIZE 16384
#define PORT 9930
#define SRV_IP "127.0.0.1"

int g_verbose = 0;
char* g_filename = NULL;
char* g_serverAddr = SRV_IP;
unsigned short g_port = PORT;

typedef struct sockaddr SA;

enum {
      VERBOSE = 1,
      VERY_VERBOSE = 2
```

```
};

void diep(char*);
int parseArgs(int, char**);
void displayUsage(const char*);
void sendfile(int, SA*, const char*);

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

      if(!parseArgs(argc, argv)) {
            displayUsage(argv[0]);
            return 1;
      }

      printf("Udpclient sending to %s:%d\n", g_serverAddr, (int) g_port);

      struct sockaddr_in si_other;
  int i, slen = sizeof(si_other);

      int sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
  if (sockfd == -1) diep("socket");

  memset((char *) &si_other, 0, sizeof(si_other));
  si_other.sin_family = AF_INET;
  si_other.sin_port = htons(g_port);
  if (inet_aton(g_serverAddr, (struct in_addr*) &si_other.sin_addr)==0) {
    fprintf(stderr, "inet_aton() failed\n");
    exit(1);
  }

      sendfile(sockfd, (SA*) &si_other, g_filename);

  int err = close(sockfd);
      if(err) diep("close()");
  return 0;
} // end main

void sendfile(int sockfd, SA* si_other, const char* filename) {
      FILE* fh = fopen(filename, "r");

      if(fh == NULL) {
            fprintf(stderr, "Failed to open file: %s\n", filename);
            exit(1);
      }

      char buffer[BUFFER_SIZE];
      size_t size, bytes_sent = 0;
      int i = 0;

      fseek(fh, 0, SEEK_END);
      int32_t filesize = (int32_t) ftell(fh);
      rewind(fh);

      int32_t zero = 0;
      size = sendto(sockfd, &zero, sizeof(int32_t), 0, si_other, sizeof(SA));
```

```
        if(size == -1) diep("sendto()");

        if(g_verbose == VERY_VERBOSE)
                printf("Sent the start of data packet\n");

        size = sendto(sockfd, &filesize, sizeof(int32_t), 0, si_other, sizeof(SA));
        if(size == -1) diep("sendto()");

        // send the filesize to the server, convert to standard size for portability
        if(g_verbose == VERY_VERBOSE)
                printf("Sent filesize packet=%d (%d bytes)\n", filesize,
                        (unsigned int) sizeof(filesize));

        // transmit the file
        do {
                size = fread(buffer, sizeof(char), sizeof(buffer), fh);

                if(ferror(fh)) {
                        diep("sendfile()");
                        exit(1);
                }

                if(size == 0) break;

                        if (g_verbose == VERY_VERBOSE) {
                                int j;
                                printf("%07x ", (int)(i*sizeof(buffer)));
                                for(j = 0; j < size; j+=2) {
                                        // print out in same format as od -x
                                        printf("%02x%02x ",
                                                        (unsigned char) buffer[j+1],
                                                        (unsigned char) buffer[j]);
                                }
                                printf("\n");
                        } else if (g_verbose == VERBOSE) {
                                printf("Sending packet %d (%d bytes)\n", i+1, (int) size);
                        }

                size = sendto(sockfd, buffer, size, 0, si_other, sizeof(SA));
          if (size == -1) diep("sendto()");
                bytes_sent += size;
                ++i;
        } while(!feof(fh) && bytes_sent < MAX_FILE_SIZE);

        printf("Transmitted %d bytes of file '%s'\n", (int) bytes_sent, filename);
} // end sendfile

void diep(char *s) {
  perror(s);
  exit(1);
} // end diep

// parse program arguments, return 0 on failure 1 on success
int parseArgs(int argc, char* argv[]) {
        static struct option long_options[] = {
                {"verbose", no_argument, &g_verbose, 1},
                {"quiet", no_argument, &g_verbose, 0},
                {"port", required_argument, 0, 'p'},
                {0,0,0,0} // terminate the structure
        };

        while(1) {
                int option_index = 0;
                int c = getopt_long(argc, argv, "v::p:", long_options, &option_index);

                if (c == -1) break;

                int port;
                switch (c) {
                        case 0:
                                /* If this option set a flag, do nothing else now. */
                                if (long_options[option_index].flag != 0)     break;
                                break;

                        case 'p':
                                port = atoi(optarg);
                                if(port <= 0 || port >= 65536) {
                                    printf("Invalid port value %d\n", port);
                                    return 0;
                                }
                                g_port = (unsigned short) port;
                                break;

                        case 'v':
                                if (optarg == NULL)
                                    g_verbose = VERBOSE;
                                else if (strcmp(optarg, "v") == 0)
                                    g_verbose = VERY_VERBOSE;
                                else
                                    return 0;
                                break;

                        case '?':
                                /* getopt_long already printed an error message. */
                                return 0;

                        default:
                                abort ();
                }
        } // end while

        // check that the host and filename are the last remaining items
        if(optind+2 != argc) return 0;
        g_serverAddr = argv[optind];
        g_filename = argv[optind+1];
        return 1;
} // end parseArgs

void displayUsage(const char* self) {
        fprintf(stderr, "Usage: %s [-v | -vv] [-p port] <host> <filename>\n", self);
```

```
} // end displayUsage


#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <errno.h>
#include <signal.h>
#include <stdint.h>

#define EOG_BIN "/usr/bin/eog"
#define BUFFER_SIZE 2048
#define PORT 9930

int g_abort = 0;
int g_verbose = 0;
char* g_destdir = NULL;
unsigned short g_port = PORT;

enum {
      VERBOSE = 1,
      VERY_VERBOSE = 2
};

typedef struct sockaddr SA;

void diep(char*);
int parseArgs(int, char**);
void displayUsage(const char*);
const char* recvfile(int sockfd, SA* si_other);
void sigHandler(int s);
void displayImage(const char*);

int main(int argc, char* argv[]) {
      struct sigaction sigIntHandler;
      sigIntHandler.sa_handler = sigHandler;
      sigemptyset(&sigIntHandler.sa_mask);
      sigIntHandler.sa_flags = 0;
      sigaction(SIGINT, &sigIntHandler, NULL);

      struct sockaddr_in si_me, si_other;
  int sockfd, i;

      if(!parseArgs(argc, argv)) {
            displayUsage(argv[0]);
            return 1;
      }

      printf("Udpserver listening for connections on port %d\n", (int) g_port);
```

```
       if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
    diep("socket");

  memset((char *) &si_me, 0, sizeof(si_me));
  si_me.sin_family = AF_INET;
  si_me.sin_port = htons(g_port);
  si_me.sin_addr.s_addr = htonl(INADDR_ANY);
  if (bind(sockfd, (SA*) &si_me, sizeof(si_me)) == -1) diep("bind");

       while(!g_abort) {
              const char* filename = recvfile(sockfd, (SA*) &si_other);
              printf("\n#########################################");
              printf("#########################################\n\n");
              displayImage(filename);
       }

  close(sockfd);
  return 0;
} // end main

// receive a file and write its content to the destination directory
// returns filename
const char* recvfile(int sockfd, SA* si_other) {
       static int fileNum = 0;
       static char filename[1024];
       sprintf(filename, "%s/file%d.jpg", g_destdir, ++fileNum);
       FILE* fh = fopen(filename, "wb");

       if(fh == NULL) {
              fprintf(stderr, "Failed to open file: %s\n", filename);
              exit(1);
       }

       char buffer[BUFFER_SIZE];
       size_t packetSize, bytes_recv = 0;
       socklen_t slen = sizeof(SA);
       int i = 0;
       int32_t fileSize;

       // first receive an 0 start of data packet
       while(1) {
              packetSize =
                     recvfrom(sockfd, &fileSize, sizeof(int32_t), 0, si_other, &slen);
              if(packetSize == -1) diep("recvfrom()");
              if(fileSize == 0) break;
              printf("Waiting for start of data packet\n");
       }

       // receive the actual file size packet
       packetSize =
              recvfrom(sockfd, &fileSize, sizeof(int32_t), 0, si_other, &slen);
       if (packetSize == -1) diep("recvfrom()");

       // convert si_other back to sockaddr_in and get the ip address of client
```

```
	struct sockaddr_in* si = (struct sockaddr_in*) si_other;

	char* clientAddr = inet_ntoa( (struct in_addr) si->sin_addr);
  if (clientAddr == NULL) {
    fprintf(stderr, "inet_ntoa() failed\n");
    exit(1);
  }

	if(g_verbose == VERBOSE) {
		printf("Received size packet from %s (file=%d bytes, packet=%u 
bytes)\n",
			clientAddr, (unsigned int)fileSize, (unsigned int)packetSize);
	}

	do {
		packetSize = recvfrom(sockfd, buffer, sizeof(buffer), 0, si_other, 
&slen);
	  if (packetSize == -1) diep("recvfrom()");
		if (packetSize == 0) break;
		bytes_recv += packetSize;

		si = (struct sockaddr_in*) si_other;

		clientAddr = inet_ntoa( (struct in_addr) si->sin_addr);
		if (clientAddr == NULL) {
	fprintf(stderr, "inet_ntoa() failed\n");
	    exit(1);
	}

		if (g_verbose == VERY_VERBOSE) {
			int j;
			printf("%07x ", (int)(i*sizeof(buffer)));
			for(j = 0; j < packetSize; j+=2) {
				// print out in same format as od -x
				if(j + 1 < packetSize) {
					printf("%02x%02x ",
							(unsigned char) buffer[j+1],
							(unsigned char) buffer[j]);
				} else {
					printf("00%02x ", (unsigned char) buffer[j]);
				}
			}
			printf("\n");
		} else if (g_verbose == VERBOSE) {
			printf("Received data packet %d from %s (%d bytes)\n", i+1,
				clientAddr, (unsigned int) packetSize);
		}

		size_t bytesWritten = fwrite(buffer, sizeof(char), packetSize, fh);
		//printf("wrote %u bytes\n", (unsigned int) bytesWritten);
		if(bytesWritten != packetSize) diep("fwrite()");

		++i;
	} while(bytes_recv < fileSize);
	fclose(fh);
```

```
        char basefilename[1024];
        sprintf(basefilename, "file%d.jpg", fileNum);
        printf("Received %d bytes, output written to file '%s'\n",
                (int)bytes_recv, basefilename);

        return filename;
} // end recvfile

void diep(char *s) {
  perror(s);
  exit(1);
} // end diep

// display an image using eye of gnome by launching a new process
void displayImage(const char* filename) {
        pid_t pid = fork();

        if(pid == 0) { // child
                //printf("Opening %s\n", filename);
                execl(EOG_BIN, EOG_BIN, filename, NULL);
                exit(0);
        } else if(pid < 0) {
                fprintf(stderr, "Failed to fork new process\n");
                exit(1);
        } else { // parent
                // do nothing
        }
}

// parse program arguments, return 0 on failure 1 on success
int parseArgs(int argc, char* argv[]) {
        static struct option long_options[] = {
                {"verbose", no_argument, &g_verbose, 1},
                {"quiet", no_argument, &g_verbose, 0},
                {"port", required_argument, 0, 'p'},
                {0,0,0,0} // terminate the structure
        };

        while(1) {
                int option_index = 0;
                int c = getopt_long(argc, argv, "v::p:", long_options, &option_index);

                if (c == -1) break;

                int port;
                switch (c) {
                        case 0:
                                /* If this option set a flag, do nothing else now. */
                                if (long_options[option_index].flag != 0)     break;
                                break;

                        case 'p':
                                port = atoi(optarg);
                                if(port <= 0 || port >= 65536) {
```

```
                        printf("Invalid port value %d\n", port);
                        return 0;
                    }
                    g_port = (unsigned short) port;
                    break;

                case 'v':
                    if (optarg == NULL)
                        g_verbose = VERBOSE;
                    else if (strcmp(optarg, "v") == 0)
                        g_verbose = VERY_VERBOSE;
                    else
                        return 0;
                    break;

                case '?':
                    /* getopt_long already printed an error message. */
                    return 0;

                default:
                    abort ();
            }
    } // end while

    // check that the filename is the last remaining item
    if(optind+1 != argc) return 0;

    // strip last '/' from dir
    g_destdir = argv[optind];
    if(g_destdir == NULL) {
        g_destdir = "."; // use the local directory
        return 1;
    }
    int len = strlen(g_destdir);
    if(g_destdir[len-1] == '/') g_destdir[len-1] = '\0';
    return 1;
} // end parseArgs

void displayUsage(const char* self) {
    fprintf(stderr, "Usage: %s [-v | -vv] [-p port] [folder]\n", self);
} // end displayUsage

// capture ctrl+c for controlled shutdown
void sigHandler(int s) {
    g_abort = 1;
} // end sigHandler
```

Hardware

embedded_image_capture.vhd

```
--
-- Video Chat System top-level module
--
-- Modified from DE2_TOP
-- by Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
--
-- From an original by Terasic Technology, Inc.
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity embedded_image_capture is

  port (
    -- Clocks

    OSC_27,                                            -- 27 MHz
    OSC_50,                                            -- 50 MHz
    EXT_CLOCK : in std_logic;                        -- External Clock

    -- Buttons and switches

    KEY : in std_logic_vector(3 downto 0);           -- Push buttons
    DPDT_SW : in std_logic_vector(17 downto 0);               -- DPDT switches

    -- LED displays
       LED_RED : out std_logic_vector(17 downto 0);      -- Red LEDs

    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
       : out std_logic_vector(6 downto 0);

    -- SRAM

    SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
    SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18 Bits
    SRAM_UB_N,                                     -- High-byte Data Mask
    SRAM_LB_N,                                     -- Low-byte Data Mask
    SRAM_WE_N,                                     -- Write Enable
    SRAM_CE_N,                                     -- Chip Enable
    SRAM_OE_N : out std_logic;                     -- Output Enable

    -- SDRAM

    DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
    DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
    DRAM_LDQM,                                     -- Low-byte Data Mask
    DRAM_UDQM,                                     -- High-byte Data Mask
    DRAM_WE_N,                                     -- Write Enable
```

```
DRAM_CAS_N,                                         -- Column Address Strobe
DRAM_RAS_N,                                         -- Row Address Strobe
DRAM_CS_N,                                          -- Chip Select
DRAM_BA_0,                                          -- Bank Address 0
DRAM_BA_1,                                          -- Bank Address 1
DRAM_CLK,                                           -- Clock
DRAM_CKE : out std_logic;                           -- Clock Enable

-- I2C bus

I2C_DATA : inout std_logic; -- I2C Data
I2C_CLK : inout std_logic;   -- I2C Clock

-- VGA output

VGA_CLK,                                         -- Clock
VGA_HS,                                          -- H_SYNC
VGA_VS,                                          -- V_SYNC
VGA_BLANK,                                       -- BLANK
VGA_SYNC : out std_logic;                        -- SYNC
VGA_R,                                           -- Red[9:0]
VGA_G,                                           -- Green[9:0]
VGA_B : out std_logic_vector(9 downto 0);        -- Blue[9:0]

-- Video Decoder

TD_DATA : in std_logic_vector(7 downto 0);       -- Data bus 8 bits
TD_HS,                                           -- H_SYNC
TD_VS : in std_logic;                            -- V_SYNC
TD_RESET : out std_logic;                        -- Reset

--  Ethernet Interface

ENET_DATA : inout unsigned(15 downto 0);         -- DATA bus 16 Bits
ENET_CMD,           -- Command/Data Select, 0 = Command, 1 = Data
ENET_CS_N,                                       -- Chip Select
ENET_WR_N,                                       -- Write
ENET_RD_N,                                       -- Read
ENET_RST_N,                                      -- Reset
ENET_CLK : out std_logic;                        -- Clock 25 MHz
ENET_INT : in std_logic;                         -- Interrupt

-- RS-232 interface

UART_TXD : out std_logic;                           -- UART transmitter
UART_RXD : in std_logic;                            -- UART receiver

-- 16 X 2 LCD Module

LCD_ON,                      -- Power ON/OFF
LCD_BLON,                    -- Back Light ON/OFF
LCD_RW,                      -- Read/Write Select, 0 = Write, 1 = Read
LCD_EN,                      -- Enable
LCD_RS : out std_logic;      -- Command/Data Select, 0 = Command, 1 = Data
LCD_DATA : inout std_logic_vector(7 downto 0); -- Data bus 8 bits
```

```
    -- PS/2 port

    PS2_DATA,                    -- Data
    PS2_CLK : inout std_logic     -- Clock

    );
      signal rst, freeze, full_screen, send_image, del_image: std_logic;
      signal move_right, move_left: std_logic;
      signal sdram_clk    : std_logic;
      signal mSEG7_HEX : std_logic_vector(31 downto 0);
      signal DATA_to_OTG : std_logic_vector(15 downto 0);
      signal DATA_to_SRAM : std_logic_vector(15 downto 0);
      --     I2C
      signal SCL_OE_N, SDA_OE_N, SCL, SDA : std_logic;

      --     CPU
      signal CPU_RESET : std_logic;
      signal DRAM_BA : std_logic_vector(1 downto 0);
      signal DRAM_QM : std_logic_vector(1 downto 0);

end embedded_image_capture;


architecture datapath of embedded_image_capture is

------------------------------------------------------------------
------------------ J P E G   S I G N A L S ----------------------
------------------------------------------------------------------

  --OPB exports
  signal OPB_ABus          : std_logic_vector(31 downto 0);
  signal OPB_BE            : std_logic_vector(3 downto 0);
  signal OPB_DBus_in       : std_logic_vector(31 downto 0);
  signal OPB_RNW           : std_logic;
  signal OPB_select        : std_logic;
  signal OPB_DBus_out      : std_logic_vector(31 downto 0);
  signal OPB_XferAck       : std_logic;
  signal OPB_retry         : std_logic;
  signal OPB_toutSup       : std_logic;
  signal OPB_errAck        : std_logic;

  --pixel line buffer
  signal iram_wdata              : std_logic_vector(23 downto 0);
  signal iram_wren               : std_logic;
  signal iram_fifo_afull  : std_logic;

  --output ram
  signal ram_byte                : std_logic_vector(7 downto 0);
  signal ram_wren                : std_logic;
  signal ram_wraddr              : std_logic_vector(23 downto 0);
  signal outif_almost_full       : std_logic := '0';

  signal outram_raddress  : std_logic_vector(13 downto 0);       --outram 14 bits
  signal outram_readdata  : std_logic_vector(7 downto 0);
  signal jpeg_reset              : std_logic := '0';

-----------------------------------------------------------------
------------- E N D  J P E G  S I G N A L S ---------------------
-----------------------------------------------------------------


      component I2C_AV_Config port(
             -- Host Side
             iCLK,
             iRST_N       : in std_logic;
             -- I2C Side
             I2C_SCLK     : out std_logic;
             I2C_SDAT     : inout std_logic);
      end component;

      component Reset_Delay port(
             iCLK : in std_logic;
             oRESET : out std_logic
             );
      end component;

      component nios_0 port(
                  -- 1 global signals:
             clk,
             reset_n : std_logic;

             ------------- J P E G  S T U F F --------------

             raddr_from_the_readram                         : out std_logic_vector(13
downto 0);
             q_to_the_readram                               : in std_logic_vector(7
downto 0);

             OPB_DBus_out_to_the_avalontrans  : in std_logic_vector(31 downto 0);
             OPB_XferAck_to_the_avalontrans          : in std_logic;
             OPB_retry_to_the_avalontrans            : in std_logic;
             OPB_toutSup_to_the_avalontrans          : in std_logic;
             OPB_errAck_to_the_avalontrans           : in std_logic;
             OPB_ABus_from_the_avalontrans           : out std_logic_vector(31 downto
0);
             OPB_BE_from_the_avalontrans             : out std_logic_vector(3 downto
0);
             OPB_DBus_in_from_the_avalontrans : out std_logic_vector(31 downto 0);
             OPB_RNW_from_the_avalontrans            : out std_logic;
             OPB_select_from_the_avalontrans  : out std_logic;
             iram_wdata_from_the_savepixel           : out std_logic_vector(23 downto
0);
             iram_wren_from_the_savepixel            : out std_logic;
             fifo_almost_full_to_the_savepixel       : in std_logic;


             ----------- E N D  J P E G  S T U F F ---------
```

```
             -- the_TV_to_VGA_inst
             OSC_27_to_the_TV_to_VGA_inst,
             OSC_50_to_the_TV_to_VGA_inst : in  std_logic;
             SRAM_ADDR_from_the_TV_to_VGA_inst : out std_logic_vector(17 downto 0);
             SRAM_CE_N_from_the_TV_to_VGA_inst : out std_logic;
             SRAM_DQ_to_and_from_the_TV_to_VGA_inst  : inout std_logic_vector(15
downto 0);
             SRAM_LB_N_from_the_TV_to_VGA_inst,
             SRAM_OE_N_from_the_TV_to_VGA_inst,
             SRAM_UB_N_from_the_TV_to_VGA_inst,
             SRAM_WE_N_from_the_TV_to_VGA_inst : out std_logic;
             TD_D_to_the_TV_to_VGA_inst : in std_logic_vector(7 downto 0);
             TD_HS_to_the_TV_to_VGA_inst,
             TD_VS_to_the_TV_to_VGA_inst : in  std_logic;
             VGA_SYNC_from_the_TV_to_VGA_inst,
             VGA_VS_from_the_TV_to_VGA_inst,
             VGA_BLANK_from_the_TV_to_VGA_inst,
             VGA_HS_from_the_TV_to_VGA_inst,
             VGA_CLOCK_from_the_TV_to_VGA_inst : out  std_logic;
             VGA_G_from_the_TV_to_VGA_inst,
             VGA_B_from_the_TV_to_VGA_inst,
             VGA_R_from_the_TV_to_VGA_inst : out std_logic_vector(9 downto 0);
             reset_to_the_TV_to_VGA_inst : in std_logic;

             freeze_b_to_the_TV_to_VGA_inst,
             full_screen_b_to_the_TV_to_VGA_inst,
             send_image_to_the_TV_to_VGA_inst,
             del_image_to_the_TV_to_VGA_inst,
             move_right_to_the_TV_to_VGA_inst,
             move_left_to_the_TV_to_VGA_inst : in std_logic;

             -- the_sdram
             zs_addr_from_the_sdram_0 : out std_logic_vector(11 downto 0);
             zs_dq_to_and_from_the_sdram_0 : inout std_logic_vector(15 downto 0);
             zs_dqm_from_the_sdram_0,
             zs_ba_from_the_sdram_0   : out std_logic_vector(1 downto 0);
             zs_cas_n_from_the_sdram_0,
             zs_cke_from_the_sdram_0,
             zs_cs_n_from_the_sdram_0,
             zs_ras_n_from_the_sdram_0,
             zs_we_n_from_the_sdram_0  : out std_logic;


             -- the_I2C_0
             scl_pad_i_to_the_I2C_0 : in std_logic;
             scl_pad_o_from_the_I2C_0,
             scl_padoen_o_from_the_I2C_0  : out std_logic;
             sda_pad_i_to_the_I2C_0 : in std_logic;
             sda_pad_o_from_the_I2C_0,
             sda_padoen_o_from_the_I2C_0,

             -- the_lcd_16207_0
             LCD_E_from_the_lcd_16207_0,
             LCD_RS_from_the_lcd_16207_0,
             LCD_RW_from_the_lcd_16207_0 : out std_logic;
```

```
            LCD_data_to_and_from_the_lcd_16207_0 : inout std_logic_vector(7 downto
0);

            -- the_SEG7_Display
            oSEG0_from_the_SEG7_Display,
            oSEG1_from_the_SEG7_Display,
            oSEG2_from_the_SEG7_Display,
            oSEG3_from_the_SEG7_Display,
            oSEG4_from_the_SEG7_Display,
            oSEG5_from_the_SEG7_Display,
            oSEG6_from_the_SEG7_Display,
            oSEG7_from_the_SEG7_Display : out std_logic_vector(6 downto 0);


            -- the_DM9000A
            ENET_DATA_to_and_from_the_DM9000A : inout unsigned(15 downto 0);
            ENET_CMD_from_the_DM9000A,
            ENET_CS_N_from_the_DM9000A,
            ENET_RD_N_from_the_DM9000A,
            ENET_RST_N_from_the_DM9000A,
            ENET_WR_N_from_the_DM9000A : out std_logic;
            ENET_INT_to_the_DM9000A : in std_logic;

            -- the_uart_0
            rxd_to_the_uart_0 : in std_logic;
            txd_from_the_uart_0 : out std_logic;

            -- the_ps2_0
            PS2_CLK_to_and_from_the_ps2_0,
            PS2_DAT_to_and_from_the_ps2_0  : inout std_logic
            );
      end component;

begin

      i2c: I2C_AV_Config port map(
            --    Host Side
            iCLK        => OSC_27,
            iRST_N      => rst,
            --    I2C Side
            I2C_SCLK    => I2C_CLK,
            I2C_SDAT    => I2C_DATA);



      nios: nios_0 port map (
            -- 1 global signals:
            clk         => OSC_50,
            reset_n     => CPU_RESET,

            -- the_TV_to_VGA_inst
            OSC_27_to_the_TV_to_VGA_inst                => OSC_27,
            OSC_50_to_the_TV_to_VGA_inst                => OSC_50,
            SRAM_ADDR_from_the_TV_to_VGA_inst     => SRAM_ADDR,
            SRAM_CE_N_from_the_TV_to_VGA_inst     => SRAM_CE_N,
            SRAM_DQ_to_and_from_the_TV_to_VGA_inst => SRAM_DQ,
            SRAM_LB_N_from_the_TV_to_VGA_inst      => SRAM_LB_N,
            SRAM_OE_N_from_the_TV_to_VGA_inst      => SRAM_OE_N,
            SRAM_UB_N_from_the_TV_to_VGA_inst      => SRAM_UB_N,
            SRAM_WE_N_from_the_TV_to_VGA_inst      => SRAM_WE_N,
            TD_D_to_the_TV_to_VGA_inst             => TD_DATA,
            TD_HS_to_the_TV_to_VGA_inst                  => TD_HS,
            TD_VS_to_the_TV_to_VGA_inst                  => TD_VS,
            VGA_BLANK_from_the_TV_to_VGA_inst      => VGA_BLANK,
            VGA_B_from_the_TV_to_VGA_inst          => VGA_B,
            VGA_CLOCK_from_the_TV_to_VGA_inst      => VGA_clk,
            VGA_G_from_the_TV_to_VGA_inst          => VGA_G,
            VGA_HS_from_the_TV_to_VGA_inst         => VGA_HS,
            VGA_R_from_the_TV_to_VGA_inst          => VGA_R,
            VGA_SYNC_from_the_TV_to_VGA_inst => VGA_SYNC,
            VGA_VS_from_the_TV_to_VGA_inst         => VGA_VS,
            freeze_b_to_the_TV_to_VGA_inst         => freeze,
            full_screen_b_to_the_TV_to_VGA_inst          => full_screen,
            send_image_to_the_TV_to_VGA_inst       => send_image,
            del_image_to_the_TV_to_VGA_inst        => del_image,
            move_right_to_the_TV_to_VGA_inst       => move_right,
            move_left_to_the_TV_to_VGA_inst        => move_left,
            reset_to_the_TV_to_VGA_inst                  => rst,

            -- the_sdram
            zs_addr_from_the_sdram_0               => DRAM_ADDR,
            zs_ba_from_the_sdram_0                 => DRAM_BA,
            zs_cas_n_from_the_sdram_0              => DRAM_CAS_N,
            zs_cke_from_the_sdram_0                => DRAM_CKE,
            zs_cs_n_from_the_sdram_0               => DRAM_CS_N,
            zs_dq_to_and_from_the_sdram_0          => DRAM_DQ,
            zs_dqm_from_the_sdram_0                => DRAM_QM,
            zs_ras_n_from_the_sdram_0              => DRAM_RAS_N,
            zs_we_n_from_the_sdram_0               => DRAM_WE_N,


            -- the_I2C_0
            scl_pad_i_to_the_I2C_0    =>     I2C_CLK,
            scl_pad_o_from_the_I2C_0  =>     SCL,
            scl_padoen_o_from_the_I2C_0      =>     SCL_OE_N,
            sda_pad_i_to_the_I2C_0    =>     I2C_DATA,
            sda_pad_o_from_the_I2C_0  =>     SDA,
            sda_padoen_o_from_the_I2C_0      =>     SDA_OE_N,

            -- the_lcd_16207_0
            LCD_E_from_the_lcd_16207_0=>     LCD_EN,
            LCD_RS_from_the_lcd_16207_0      =>     LCD_RS,
            LCD_RW_from_the_lcd_16207_0      =>     LCD_RW,
            LCD_data_to_and_from_the_lcd_16207_0   =>     LCD_DATA,

            -- the_SEG7_Display
            oSEG0_from_the_SEG7_Display      =>     HEX0,
            oSEG1_from_the_SEG7_Display      =>     HEX1,
            oSEG2_from_the_SEG7_Display      =>     HEX2,
            oSEG3_from_the_SEG7_Display      =>     HEX3,
            oSEG4_from_the_SEG7_Display      =>     HEX4,
            oSEG5_from_the_SEG7_Display      =>     HEX5,
            oSEG6_from_the_SEG7_Display      =>     HEX6,
            oSEG7_from_the_SEG7_Display      =>     HEX7,


            -- the_DM9000A
            ENET_CMD_from_the_DM9000A =>     ENET_CMD,
            ENET_CS_N_from_the_DM9000A=>     ENET_CS_N,
            ENET_DATA_to_and_from_the_DM9000A      =>     ENET_DATA,
            ENET_INT_to_the_DM9000A   =>     ENET_INT,
            ENET_RD_N_from_the_DM9000A=>     ENET_RD_N,
            ENET_RST_N_from_the_DM9000A      =>     ENET_RST_N,
            ENET_WR_N_from_the_DM9000A=>     ENET_WR_N,

            -- the_uart_0
            rxd_to_the_uart_0   =>     UART_RXD,
            txd_from_the_uart_0 =>     UART_TXD,

            -- the_ps2_0
            PS2_CLK_to_and_from_the_ps2_0    =>     PS2_CLK,
            PS2_DAT_to_and_from_the_ps2_0    =>     PS2_DATA,


            ------------------- J P E G  P E R I P H E R A L S -------------------

            raddr_from_the_readram                    => outram_raddress,
            q_to_the_readram                          => outram_readdata,

            OPB_DBus_out_to_the_avalontrans    => OPB_DBus_out,
            OPB_XferAck_to_the_avalontrans     => OPB_XferAck,
            OPB_retry_to_the_avalontrans       => OPB_retry,
            OPB_toutSup_to_the_avalontrans     => OPB_toutSup,
            OPB_errAck_to_the_avalontrans      => OPB_errAck,
            OPB_ABus_from_the_avalontrans      => OPB_ABus,
            OPB_BE_from_the_avalontrans        => OPB_BE,
            OPB_DBus_in_from_the_avalontrans  => OPB_DBus_in,
            OPB_RNW_from_the_avalontrans       => OPB_RNW,
            OPB_select_from_the_avalontrans   => OPB_select,

            iram_wdata_from_the_savepixel      => iram_wdata,
            iram_wren_from_the_savepixel       => iram_wren,
            fifo_almost_full_to_the_savepixel => iram_fifo_afull

            ----------------- E N D  J P E G  P E R I P H E R A L S -------------

    );

    ------------------------ J P E G  M O D U L E -------------------

    jpeg_reset <= DPDT_SW(1);
    LED_RED(1) <= jpeg_reset;

      jpeg : entity work.JpegEnc port map (
       CLK                => OSC_50,
        RST                 => jpeg_reset,

        -- OPB
        OPB_ABus            => OPB_ABus,
        OPB_BE              => OPB_BE,
        OPB_DBus_in         => OPB_DBus_in,
        OPB_RNW             => OPB_RNW,
        OPB_select          => OPB_select,
        OPB_DBus_out        => OPB_DBus_out,
        OPB_XferAck         => OPB_XferAck,
        OPB_retry           => OPB_retry,
        OPB_toutSup         => OPB_toutSup,
        OPB_errAck          => OPB_errAck,

        -- IMAGE RAM
        iram_wdata          => iram_wdata,
        iram_wren           => iram_wren,
        iram_fifo_afull     => iram_fifo_afull,

        -- OUT RAM
        ram_byte            => ram_byte,
        ram_wren            => ram_wren,
        ram_wraddr          => ram_wraddr,
        outif_almost_full   => '0' --outif_almost_full
    );

       -------------- E N D  J P E G  M O D U L E --------------

       ----------------- O U T P U T  M E R O R Y ---------------

       outram : entity work.OutputRAM port map (
              clk                                => OSC_50,
              raddr                       => outram_raddress,
              waddr                       => ram_wraddr(13 downto 0),            --
outram 14 bits
              data                        => ram_byte,
              we                                 => ram_wren,
              q                                  => outram_readdata
       );

       ----------------- E N D  O U T P U T  R A M -----------------------



       sdram_pll: work.sdram_pll port map (
              inclk0 => OSC_50,
              c0                  => DRAM_CLK
       );

       delay: Reset_Delay port map (
              iCLK => OSC_50,
              oRESET => CPU_RESET
              );

--     DRAM_CLK  <= sdram_clk;
--     DRAM_UDQM <= DRAM_QM(1);
--     DRAM_LDQM <= DRAM_QM(0);
--     DRAM_BA_1 <= DRAM_BA(1);
--     DRAM_BA_0 <= DRAM_BA(0);

       -- TV
       TD_RESET     <= '1';      --     Bypress 27 MHz
       rst                 <= DPDT_SW(0);

       --     I2C Tri-inout
--     I2C_CLK      <=     'Z' when SCL_OE_N   = '1' else SCL;
--     I2C_DATA     <=     'Z' when SDA_OE_N = '1' else SDA;

  DRAM_BA_1 <= DRAM_BA(1);
  DRAM_BA_0 <= DRAM_BA(0);
  DRAM_UDQM <= DRAM_QM(1);
  DRAM_LDQM <= DRAM_QM(0);

       process (OSC_50)
       variable clk : std_logic := '0';
       begin
             if rising_edge(OSC_50) then
                   clk := not clk;
                   ENET_CLK <= clk;
             end if;
       end process;

       -- Handle asynchrounous signals
       process (OSC_27)
       variable count : integer := 0;
       begin
             if rising_edge(OSC_27) then
                   count := count + 1;
                   if count = 2_700_000 then
                         count := 0;
                         freeze <= DPDT_SW(17);
                         full_screen <= not DPDT_SW(16);
                         send_image <= not key(3);
                         del_image <= not key(2) and key(3);
                         move_left <= not key(1);
                         move_right <= not key(0) and key(1);
                   end if;
             end if;
       end process;




end datapath;
```

TV_to_VGA.vhd:

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

```
-- TV to VGA converter
--
-- Modified from de2_raster_vga.vhd
-- by Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
-- using YCbCr2RGB.v and itu_r656_decoder.v from Terasic Technology, Inc.
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity TV_to_VGA is

  port (
    reset,
    OSC_27,
      OSC_50: in std_logic;

    read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  std_logic_vector(4 downto 0);
    readdata   : out std_logic_vector(15 downto 0);
    writedata  : in  std_logic_vector(15 downto 0);

    -- SRAM
    SRAM_DQ : inout std_logic_vector(15 downto 0);   -- Data bus 16 Bits
    SRAM_ADDR : out std_logic_vector(17 downto 0);   -- Address bus 18 Bits
    SRAM_UB_N,                                       -- High-byte Data Mask
    SRAM_LB_N,                                       -- Low-byte Data Mask
    SRAM_WE_N,                                       -- Write Enable
    SRAM_CE_N,                                       -- Chip Enable
    SRAM_OE_N : out std_logic;                       -- Output Enable

             -- TV
    TD_D     : in std_logic_vector(7 downto 0);                         -- Data bus 8
bits
    TD_HS,                                                   -- H_SYNC
    TD_VS : in std_logic;                                    -- V_SYNC

             -- VGA
    VGA_CLOCK,
                                                     -- Clock
    VGA_HS,
      -- H_SYNC
    VGA_VS,
      -- V_SYNC
    VGA_BLANK,
      -- BLANK
    VGA_SYNC : out std_logic;
      -- SYNC
    VGA_R,
      -- Red[9:0]
    VGA_G,
      -- Green[9:0]
    VGA_B : out std_logic_vector(9 downto 0);                    -- Blue[9:0]
```

```
            -- Control signals
        freeze_b, full_screen_b,
            send_image, del_image,
            move_right, move_left : in std_logic

    );

end TV_to_VGA;

architecture rtl of TV_to_VGA is

      -- Constants

      -- VGA constants
      constant vga_x_size                   : integer := 740;
      constant vga_y_size                   : integer := 504;

      -- Screen constants
      constant screen_x_offset  : integer := 60;--97;
      constant screen_y_offset  : integer := 28;--51;
      constant screen_x_size          : integer := vga_x_size - screen_x_offset;
      constant screen_y_size          : integer := vga_y_size - screen_y_offset -
8;

      -- Image constants
      constant image_x_size           : integer := screen_x_size / 2;
      constant image_y_size           : integer := screen_y_size / 2;
      constant image_x_offset   : integer := (screen_x_size - image_x_size) / 2;
      constant image_y_offset   : integer := 60;
      constant image_border           : integer := 2;

      -- Image list constants
      constant image_list_offset: integer := 0;
      constant image_list_border: integer := 0;
      constant image_list_spacing     : integer := 0;

      -- Signals


      -- VGA counters
      signal vga_x_count, vga_y_count: unsigned(9 downto 0);
      signal EndOfLine, EndOfField : std_logic;
      signal x_reset, y_reset : std_logic;

      -- Screen counters
      signal screen_x_count, screen_y_count: unsigned(9 downto 0);
      signal is_within_screen   : std_logic;

      -- Image signal
      signal image_x_count, image_y_count: unsigned(9 downto 0);
      signal is_within_image    : std_logic;
      signal is_image_frame     : std_logic;
      signal is_within_image_list     : std_logic;
```

```
   -- Image list signals
   signal image_list_x_count, image_list_y_count: unsigned(9 downto 0);

   -- SRAM signals
   signal byteenable : std_logic_vector(1 downto 0);
   signal write_data, read_data : std_logic_vector(15 downto 0);
   signal ram_addr, image_addr, screen_addr : unsigned(17 downto 0);
   signal get_image_addr, save_image_addr : unsigned(17 downto 0);
   signal addr_tmp : unsigned(19 downto 0);
   signal sram_data, req_data : std_logic_vector(15 downto 0);

   -- VGA signals
   signal red_r, green_r, blue_r : std_logic_vector(9 downto 0);
   signal VGA_HS_r, VGA_VS_r : std_logic;

   -- Sending image signals
   signal save_image_x, save_image_y: unsigned(9 downto 0);
   signal got_x, got_y, sent_data : std_logic := '0';

   -- Receiving image signals
   signal getting_image : std_logic := '0';
   signal pixel : unsigned(15 downto 0);
   signal get_image_x, get_image_y : unsigned(9 downto 0);
   signal got_x2, got_y2, got_data : std_logic := '0';
   signal current_image : unsigned(9 downto 0);
   signal num_images : unsigned(9 downto 0) := "00" & x"00";

   -- Receivig thumbnails signals



   --
   signal save_image, addr0, write_to_ram: std_logic := '0';
   signal red_pixel, green_pixel, blue_pixel     : std_logic_vector(9 downto 0);
   signal freeze_k, full_screen_k, freeze, full_screen : std_logic;


   -- Components

   -- itu_r656_decoder and YCbCr2RGB Signals
   signal Y, Cb, Cr : std_logic_vector(7 downto 0);           -- 4:4:4 signals
   signal mTD_HSx2 : std_logic;
   signal Red, Green, Blue : std_logic_vector(9 downto 0);

   component itu_r656_decoder port(
         CLOCK  : in std_logic;
         TD_D   : in std_logic_vector(7 downto 0);
         TD_HS,
         TD_VS  : in std_logic;
         Y,
         Cb,
         Cr           : out std_logic_vector(7 downto 0);
HSx2,
         blank  : out std_logic);
   end component;
```

```
	component YCbCr2RGB port(
		-- inptus:
		iY,
		iCb,
		iCr		: in std_logic_vector(7 downto 0);
		iRESET,
		iCLK	: in std_logic;
		-- outputs:
		Red,
		Green,
		Blue	: out std_logic_vector(9 downto 0)
		);
	end component;

	component VGA_Synchronization port(
		-- inputs:
		OSC_27,
		reset,
		TD_HS,
		TD_VS,
		mTD_HSx2	: in std_logic;
		-- outputs:
		VGA_SYNC,
		VGA_CLOCK,
		VGA_HS,
		VGA_VS		: out std_logic;
		h_reset,
		v_reset			: out std_logic);
	end component;

	component sram port (
		-- inputs:
		address : IN STD_LOGIC_VECTOR (17 DOWNTO 0);
		byteenable : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
		chipselect : IN STD_LOGIC;
		read : IN STD_LOGIC;
		write : IN STD_LOGIC;
		writedata : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
		-- outputs:
		SRAM_ADDR : OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
		SRAM_CE_N : OUT STD_LOGIC;
		SRAM_DQ			: INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
		SRAM_LB_N : OUT STD_LOGIC;
		SRAM_OE_N : OUT STD_LOGIC;
		SRAM_UB_N : OUT STD_LOGIC;
		SRAM_WE_N : OUT STD_LOGIC;
		readdata	: OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
	end component;


begin
	freeze <= freeze_b or freeze_k;
	full_screen <= not full_screen_b or full_screen_k;
```

```
     -- Set vga_x_count and vga_y_count

EndOfLine <= '1' when vga_x_count = vga_x_size - 1 else '0';
EndOfField <= '1' when vga_y_count = vga_y_size - 1 else '0';

process (OSC_27)
begin
  if rising_edge(OSC_27) then
    if reset = '1' or VGA_HS_r = '0' then
      vga_x_count <= (others => '0');
   elsif EndOfLine = '1' then
     vga_x_count <= (others => '0');
    else
      vga_x_count <= vga_x_count + 1;
    end if;
  end if;
end process;

process (OSC_27)
begin
  if rising_edge(OSC_27) then
    if reset = '1' or VGA_VS_r = '0' then
      vga_y_count <= (others => '0');
    elsif EndOfLine = '1' then
      if EndOfField = '1' then
        vga_y_count <= (others => '0');
      else
        vga_y_count <= vga_y_count + 1;
      end if;
    end if;
  end if;
end process;

     -- Sub-components

     -- ITU Decoder
     itu: itu_r656_decoder port map(
            CLOCK  => OSC_27,
            TD_D   => TD_D,
            TD_HS  => TD_HS,
            TD_VS  => TD_VS,
            Y             => Y,
            Cb            => Cb,
            Cr            => Cr,
     HSx2   => mTD_HSx2,
            blank  => VGA_BLANK
     );

     -- Color Space Convertor (YCbCr to RGB)
     color_space_convertor: YCbCr2RGB port map(
            Red           => red_r,
            Green  => green_r,
            Blue   => blue_r,
            iY            => Y,
            iCb           => Cb,
		iCr		=> Cr,
		iRESET	=> RESET,
		iCLK	=> OSC_27
	);

	-- Synchronization
	sync: VGA_Synchronization port map(
		OSC_27		=> OSC_27,
		reset		=> reset,
		TD_HS		=> TD_HS,
		TD_VS		=> TD_VS,
		mTD_HSx2	=> mTD_HSx2,
		VGA_SYNC	=> VGA_SYNC,
		VGA_CLOCK	=> VGA_CLOCK,
		VGA_HS		=> VGA_HS_r,
		VGA_VS		=> VGA_VS_r,
		h_reset			=> x_reset,
		v_reset			=> y_reset
	);

	VGA_HS <= VGA_HS_r;
	VGA_VS <= VGA_VS_r;

	-- Send pixel value when requested
	-- address:
	--		0	- receive x position
	--		1	- receive y position
	--		2 - get save_image signal
	--		3 - send data
	process (OSC_50)
	begin
		if rising_edge(OSC_50) then

			-- get freeze
			if (chipselect = '1' and write = '1' and address = "11001") then
				freeze_k <= writedata(0);
			end if;

			-- get full_screen
			if (chipselect = '1' and write = '1' and address = "11010") then
				full_screen_k <= writedata(0);
			end if;

			-- get number of stored images
			if (chipselect = '1' and write = '1' and address = "01100") then
				num_images <= unsigned(writedata(9 downto 0));
			end if;

			-- delete image
			if (chipselect = '1' and read = '1' and address = "01101") then
				if (del_image = '1') then
					readdata <= x"0001";
				else
					readdata <= x"0000";
				end if;
```

```
                    end if;

                    -- Send image to the sdram

                    -- get x_pos
                    if (save_image = '1' and chipselect = '1' and write = '1' and
address = "00000") then
                          save_image_x <= unsigned(writedata(9 downto 0));
                          got_x <= '1';
                    end if;
                    -- get y_pos
                    if (save_image = '1' and chipselect = '1' and write = '1' and
address = "00001") then
                      save_image_y <= unsigned(writedata(9 downto 0));
                      got_y <= '1';
                    end if;
                    -- get request
                    if (chipselect = '1' and write = '1' and address = "00010") then
                          -- save_image just went high
                          if (save_image = '0' and writedata = x"0001") then
                                save_image <= '1';
                                got_x <= '0';
                                got_y <= '0';
                                sent_data <= '0';
                          -- save_image just went low
                          elsif (save_image = '1' and writedata = x"0000") then
                                save_image <= '0';
                          end if;
                    end if;
                    -- send data and set ack to 1
                    if (save_image = '1' and got_x = '1' and got_y = '1' and
chipselect = '1' and read = '1' and address = "00011") then
                          readdata <= std_logic_vector(sram_data);
                          sent_data <= '1';
                          save_image <= '0';
                    end if;
                    -- tell nios to send request so that i can send data
                    if chipselect = '1' and read = '1' and address = "00100" then
                          if send_image = '1' and save_image = '0' then
                                      readdata <= x"0001";
                          else
                                      readdata <= x"0000";
                          end if;
                    end if;

                    -- Get an image from the sdram

                    -- get x_pos
                    if (getting_image = '1' and chipselect = '1' and write = '1' and
address = "00101") then
                          get_image_x <= unsigned(writedata(9 downto 0));
                          got_x2 <= '1';
                    end if;
                    -- get y_pos
```

```vhdl
                    if (getting_image = '1' and chipselect = '1' and write = '1' and
address = "00110") then
                      get_image_y <= unsigned(writedata(9 downto 0));
                      got_y2 <= '1';
                    end if;
                    -- get pixel
                    if (getting_image = '1' and chipselect = '1' and write = '1' and
address = "00111") then
                      pixel <= unsigned(writedata);
                      got_data <= '1';
                    end if;
                    -- get request
                    if (chipselect = '1' and write = '1' and address = "01000") then
                          -- getting_image just went high
                          if (getting_image = '0' and writedata = x"0001") then
                                getting_image <= '1';
                                got_x2 <= '0';
                                got_y2 <= '0';
                                got_data <= '0';
                          -- getting_image just went low
                          elsif (got_data = '1' and writedata = x"0000") then
                                getting_image <= '0';
                                got_data <= '0';
                          end if;
                    end if;
                    -- send data and set ack to 1
                    if (getting_image = '1' and got_x2 = '1' and got_y2 = '1' and
chipselect = '1' and read = '1' and address = "01001") then
                          readdata <= std_logic_vector(pixel);
                          got_data <= '1';
                          getting_image <= '0';
                    end if;
                    -- tell nios to send request so that i can send data
                    if chipselect = '1' and read = '1' and address = "01010" then
                          if getting_image = '0' then
                                if move_left = '1' then
                                      readdata <= x"0001";
                                elsif move_right = '1' then
                                  readdata <= x"0002";
                                end if;
                          else
                                      readdata <= x"0000";
                          end if;
                    end if;

                    -- Get thumbnails

              end if;
       end process;

       process (OSC_27)
       variable count : integer := 0;
       begin
              if rising_edge(OSC_50) then
                    count := count + 1;
                    if count = 9_000_000 then
                          count := 0;
                          if move_right = '1' and current_image < num_images - 1
then
                                current_image <= current_image + 1;
                          elsif move_left = '1' and current_image > 0 then
                                current_image <= current_image - 1;
                          end if;
                    end if;
              end if;
      end process;


      -- Determine where within the vga screen
      is_within_screen <= '1' when (vga_x_count >= screen_x_offset and vga_x_count
<= screen_x_size + screen_x_offset)
                                                                      and
(vga_y_count >= screen_y_offset and vga_y_count <= screen_y_size + screen_y_offset)
else '0';
      is_within_image <= '1' when (screen_x_count >= image_x_offset and
screen_x_count <= image_x_size + image_x_offset)
                                                                      and
(screen_y_count >= image_y_offset and screen_y_count <= image_y_size +
image_y_offset) else '0';
      is_image_frame <= '1' when not is_within_image = '1' and (screen_x_count >=
image_x_offset - image_border and screen_x_count <= image_x_size + image_x_offset +
image_border)
                                                                      and
(screen_y_count >= image_y_offset - image_border and screen_y_count <= image_y_size +
image_y_offset + image_border) else '0';
      is_within_image_list <= '1' when is_within_screen = '1' and screen_y_count >
(image_y_offset + image_y_size + image_border) else '0';

      -- Viewable screen
      screen_x_count <= vga_x_count - screen_x_offset when vga_x_count >=
screen_x_offset else (others => '0');
      screen_y_count <= vga_y_count - screen_y_offset when vga_y_count >=
screen_y_offset else (others => '0');

      -- Image section
      image_x_count <= screen_x_count - image_x_offset when screen_x_count >=
image_x_offset;
      image_y_count <= screen_y_count - image_y_offset when screen_y_count >=
image_y_offset;

      -- Image list section
      image_list_x_count <= screen_x_count;
      image_list_y_count <= screen_y_count - (image_y_offset + image_y_size +
image_border);


      -- SRAM

      -- Addresses
```

```
	screen_addr <= unsigned(screen_x_count(9 downto 1)) +
unsigned(screen_y_count(9 downto 1)) * image_x_size;
	image_addr <= unsigned(image_x_count(8 downto 0)) + unsigned(image_y_count(8
downto 0)) * image_x_size;
	--save_image_addr <= unsigned(save_image_x(9 downto 1)) +
unsigned(save_image_y(9 downto 1)) * (screen_x_size / 2);
	save_image_addr <= unsigned(save_image_x(8 downto 0)) +
unsigned(save_image_y(8 downto 0)) * image_x_size;
	get_image_addr <= unsigned(get_image_x(8 downto 0)) + unsigned(get_image_y(8
downto 0)) * image_x_size;
	ram_addr <= get_image_addr when (got_data = '1') else save_image_addr when
(save_image = '1') else screen_addr when (full_screen = '1' or freeze = '0') else
image_addr;
	-- Write data
	write_to_ram <= got_data or (not freeze and is_within_screen and not
save_image and screen_y_count(0) and screen_x_count(0));
	write_data <= std_logic_vector(pixel) when got_data = '1' else red_r(9 downto
5) & green_r(9 downto 4) & blue_r(9 downto 5);
	-- Read data
	req_data <= sram_data;
	red_pixel <= sram_data(15 downto 11) & "00000";
	green_pixel <= sram_data(10 downto 5) & "0000";
	blue_pixel <= sram_data(4 downto 0) & "00000";
	sram_frame_buffer: sram port map(
		chipselect   => '1',
		write			=> write_to_ram,
		read			=> not got_data and (freeze or save_image),
		address			=> std_logic_vector(ram_addr),
		writedata		=> write_data,
		byteenable   => "11",
		readdata		=> sram_data,
		SRAM_DQ			=> SRAM_DQ,
		SRAM_ADDR		=> SRAM_ADDR,
		SRAM_UB_N		=> SRAM_UB_N,
		SRAM_LB_N		=> SRAM_LB_N,
		SRAM_WE_N		=> SRAM_WE_N,
		SRAM_CE_N		=> SRAM_CE_N,
		SRAM_OE_N		=> SRAM_OE_N);


	-- VGA display

--	VGA_R <= red_r when freeze = '0' or save_image = '1' else
--				sram_data(15 downto 11) & "00000" when
is_within_screen = '1' else
--				(others => '0');
--	VGA_G <= green_r when freeze = '0' or save_image = '1' else
--				sram_data(10 downto 5) & "0000" when
is_within_screen = '1' else
--				(others => '0');
--	VGA_B <= blue_r when freeze = '0' or save_image = '1' else
--				sram_data(4 downto 0) & "00000" when
is_within_screen = '1' else
--				(others => '0');
```

```
	process (OSC_27)
	variable vga_r_r, vga_g_r, vga_b_r : std_logic_vector(9 downto 0);
	begin
		if rising_edge(OSC_27) then
					-- when sending data to the sram
					if ((got_data = '1' or save_image = '1') and (freeze
= '1' and full_screen = '0')) then
							if is_within_image = '1' then
								VGA_R <= red_pixel;
								VGA_G <= green_pixel;
								VGA_B <= blue_pixel;
							elsif is_image_frame = '1' then
								VGA_R <= (others => '1');
								VGA_G <= (others => '1');
								VGA_B <= (others => '1');
							else
								VGA_R <= (others => '0');
								VGA_G <= (others => '0');
								VGA_B <= (others => '0');
							end if;
					elsif (freeze = '0' and full_screen = '1' and
is_within_screen = '1') or
									(freeze = '0' and
full_screen = '0' and is_within_image = '1') then
								VGA_R <= red_r;
								VGA_G <= green_r;
								VGA_B <= blue_r;
					elsif (freeze = '1' and full_screen = '1' and
is_within_screen = '1') or
									(freeze = '1' and
full_screen = '0' and is_within_image = '1') then
							VGA_R <= red_pixel;
							VGA_G <= green_pixel;
							VGA_B <= blue_pixel;
					elsif full_screen = '0' and is_image_frame = '1'
then
							VGA_R <= (others => '1');
							VGA_G <= (others => '1');
							VGA_B <= (others => '1');
					else
							VGA_R <= (others => '0');
							VGA_G <= (others => '0');
							VGA_B <= (others => '0');
					end if;

		end if;
	end process;
end rtl;
```

JPEG Hardware and TB code

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
  use IEEE.STD_LOGIC_TEXTIO.ALL;

library STD;
  use STD.TEXTIO.ALL;

library work;
  use work.GPL_V2_Image_Pkg.ALL;
  use WORK.MDCT_PKG.all;
  use WORK.MDCTTB_PKG.all;
  use work.JPEG_PKG.all;

entity AvalonBFM is
  port
  (
        CLK                : in  std_logic;
        RST                : in  std_logic;

        -- Avalon
                          chipselect          : out std_logic;
                          write                     : out std_logic;
                          readen                    : out std_logic;
                          address                   : out std_logic_vector(31 downto
0);
                          writedata           : out std_logic_vector(31 downto 0);
                          readdata                  : in   std_logic_vector(31 downto
0);

                          -- HOST DATA
                          iram_wdata          : out std_logic_vector(C_PIXEL_BITS-1
downto 0);
                          iram_wren           : out std_logic;
                          fifo_almost_full    : in  std_logic;

                          sim_done           : out std_logic
    );
end entity AvalonBFM;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
----------------------------------- ARCHITECTURE ------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
architecture RTL of AvalonBFM is

  signal num_comps   : integer;
  signal addr_inc    : integer := 0;
-------------------------------------------------------------------------------
-- Architecture: begin
-------------------------------------------------------------------------------
begin
```

```
  ------------------------------------------------------------------
  -- code
  ------------------------------------------------------------------
  p_code : process

    ----------------------------------------------------------------
    -- HOST WRITE
    ----------------------------------------------------------------
    procedure host_write
      (
        signal clk               : in    std_logic;
        constant C_ADDR          : in    unsigned(31 downto 0);
        constant C_WDATA         : in    unsigned(31 downto 0);

                          signal chipselect         : out std_logic;
                          signal write                    : out std_logic;
                          signal readen                   : out std_logic;
                          signal address                  : out std_logic_vector(31
downto 0);
                          signal writedata          : out std_logic_vector(31 downto
0)

      ) is
    begin
                   chipselect   <= '0';
                   write                     <= '0';
                   readen              <= '0';
                   address             <= (others => '0');
                   writedata    <= (others => '0');

      wait until rising_edge(clk);

                   chipselect   <= '1';
                   write                     <= '1';
                   readen              <= '0';
                   address             <= std_logic_vector(C_ADDR);
                   writedata    <= std_logic_vector(C_WDATA);

      wait until rising_edge(clk);

      wait until rising_edge(clk);

                   chipselect   <= '0';
                   write                     <= '0';
                   readen              <= '0';
                   address             <= (others => '0');
                   writedata    <= (others => '0');

      assert false
      report CR&"Host write access, address = " & HexImage(C_ADDR) & ",data written =
" & HexImage(C_WDATA) &CR
      severity note;
      --wait until rising_edge(clk);

    end procedure host_write;

    -----------------------------------------------------------------
    -- HOST READ
    -----------------------------------------------------------------
    procedure host_read
      (
        signal clk          : in    std_logic;
        constant C_ADDR     : in    unsigned(31 downto 0);
        variable RDATA      : out   unsigned(31 downto 0);

                          signal chipselect         : out std_logic;
                          signal write                    : out std_logic;
                          signal readen                   : out std_logic;
                          signal address                  : out std_logic_vector(31
downto 0);
                          signal writedata          : out std_logic_vector(31 downto
0);
                          signal readdata           : in   std_logic_vector(31 downto
0)
      )
    is
      variable data_r : std_logic_vector(31 downto 0);
    begin
                    chipselect   <= '0';
                    write                    <= '0';
                    readen             <= '0';
                    address            <= (others => '0');
                    writedata    <= (others => '0');

      wait until rising_edge(clk);

                    chipselect   <= '1';
                    write                    <= '0';
                    readen             <= '1';
                    address            <= std_logic_vector(C_ADDR);
                    writedata    <= (others => '0');

      wait until rising_edge(clk);

      wait until rising_edge(clk);

      RDATA := unsigned(readdata);
      data_r := readdata;

                    chipselect   <= '0';
                    write                    <= '0';
                    readen             <= '0';
                    address            <= (others => '0');
                    writedata    <= (others => '0');

      assert false
      report CR&"Host read access, address = " & HexImage(C_ADDR) & ",data read = " &
HexImage(data_r) &CR
      severity note;

      --wait until rising_edge(clk);

    end procedure host_read;


    --------------------------------------
    -- read text image data
    --------------------------------------
    procedure read_image is
      file infile          : TEXT open read_mode is "test.txt";
      constant N           : integer := 8;
      constant MAX_COMPS   : integer := 3;
      variable inline      : LINE;
      variable tmp_int     : INTEGER := 0;
      variable y_size      : INTEGER := 0;
      variable x_size      : INTEGER := 0;
      variable matrix      : I_MATRIX_TYPE;
      variable x_blk_cnt   : INTEGER := 0;
      variable y_blk_cnt   : INTEGER := 0;
      variable n_lines_arr : N_LINES_TYPE;
      variable line_n      : INTEGER := 0;
      variable pix_n       : INTEGER := 0;
      variable x_n         : INTEGER := 0;
      variable y_n         : INTEGER := 0;
      variable data_word   : unsigned(31 downto 0);
      variable image_line  : STD_LOGIC_VECTOR(0 to MAX_COMPS*MAX_IMAGE_SIZE_X*IP_W-
1);

      constant C_IMAGE_RAM_BASE : unsigned(31 downto 0) := X"0010_0000";

      variable x_cnt       : integer;
      variable data_word2  : unsigned(31 downto 0);
      variable num_comps_v : integer;
    begin
      READLINE(infile,inline);
      READ(inline,num_comps_v);
      READLINE(infile,inline);
      READ(inline,y_size);
      READLINE(infile,inline);
      READ(inline,x_size);

      num_comps <= num_comps_v;

      if y_size rem N > 0 then
        assert false
          report "ERROR: Image height dimension is not multiply of 8!"
          severity Failure;
      end if;
      if x_size rem N > 0 then
        assert false
          report "ERROR: Image width dimension is not multiply of 8!"
          severity Failure;
      end if;

      if x_size > C_MAX_LINE_WIDTH then
        assert false
          report "ERROR: Image width bigger than C_MAX_LINE_WIDTH in JPEG_PKG.VHD! "
&
                 "Increase C_MAX_LINE_WIDTH accordingly"
          severity Failure;
      end if;

      addr_inc <= 0;

      -- image size
      host_write(CLK, X"0000_0004", to_unsigned(x_size,16) & to_unsigned(y_size,16),
               chipselect, write, readen, address, writedata);

      iram_wren <= '0';
      for y_n in 0 to y_size-1 loop
        READLINE(infile,inline);
        HREAD(inline,image_line(0 to num_comps*x_size*IP_W-1));
        x_cnt := 0;
        for x_n in 0 to x_size-1 loop
          data_word := X"00" & UNSIGNED(image_line(x_cnt to x_cnt+num_comps*IP_W-1));
          if C_PIXEL_BITS = 24 then
            data_word2(7 downto 0)   := data_word(23 downto 16);
            data_word2(15 downto 8)  := data_word(15 downto 8);
            data_word2(23 downto 16) := data_word(7 downto 0);
          else
            data_word2(4 downto 0)   := data_word(23 downto 19);
            data_word2(10 downto 5)  := data_word(15 downto 10);
            data_word2(15 downto 11) := data_word(7 downto 3);
          end if;

          iram_wren  <= '0';
          iram_wdata <= (others => 'X');
          while(fifo_almost_full = '1') loop
            wait until rising_edge(clk);
          end loop;

          --for i in 0 to 9 loop
          --  wait until rising_edge(clk);
          --end loop;

          iram_wren <= '1';
          iram_wdata <= std_logic_vector(data_word2(C_PIXEL_BITS-1 downto 0));
          wait until rising_edge(clk);

          x_cnt := x_cnt + num_comps*IP_W;

          addr_inc <= addr_inc + 1;
        end loop;
      end loop;
      iram_wren <= '0';
    end read_image;

    ------------------
    type ROMQ_TYPE is array (0 to 64-1)
            of unsigned(7 downto 0);

  constant qrom_lum : ROMQ_TYPE :=
  (
  -- 100%
  --others => X"01"

  -- 85%
  --X"05", X"03", X"04", X"04",
  --X"04", X"03", X"05", X"04",
  --X"04", X"04", X"05", X"05",
  --X"05", X"06", X"07", X"0C",
  --X"08", X"07", X"07", X"07",
  --X"07", X"0F", X"0B", X"0B",
  --X"09", X"0C", X"11", X"0F",
  --X"12", X"12", X"11", X"0F",
  --X"11", X"11", X"13", X"16",
  --X"1C", X"17", X"13", X"14",
  --X"1A", X"15", X"11", X"11",
  --X"18", X"21", X"18", X"1A",
  --X"1D", X"1D", X"1F", X"1F",
  --X"1F", X"13", X"17", X"22",
  --X"24", X"22", X"1E", X"24",
  --X"1C", X"1E", X"1F", X"1E"

  -- 100%
  --others => X"01"

  -- 75%
   --X"08", X"06", X"06", X"07", X"06", X"05", X"08", X"07", X"07", X"07", X"09",
X"09", X"08", X"0A", X"0C", X"14",
   --X"0D", X"0C", X"0B", X"0B", X"0C", X"19", X"12", X"13", X"0F", X"14", X"1D",
X"1A", X"1F", X"1E", X"1D", X"1A",
   --X"1C", X"1C", X"20", X"24", X"2E", X"27", X"20", X"22", X"2C", X"23", X"1C",
X"1C", X"28", X"37", X"29", X"2C",
   --X"30", X"31", X"34", X"34", X"34", X"1F", X"27", X"39", X"3D", X"38", X"32",
X"3C", X"2E", X"33", X"34", X"32"

   -- 15 %
   --X"35", X"25", X"28", X"2F",
   --X"28", X"21", X"35", X"2F",
   --X"2B", X"2F", X"3C", X"39",
   --X"35", X"3F", X"50", X"85",
   --X"57", X"50", X"49", X"49",
   --X"50", X"A3", X"75", X"7B",
   --X"61", X"85", X"C1", X"AA",
   --X"CB", X"C8", X"BE", X"AA",
   --X"BA", X"B7", X"D5", X"F0",
   --X"FF", X"FF", X"D5", X"E2",
   --X"FF", X"E6", X"B7", X"BA",
   --X"FF", X"FF", X"FF", X"FF",
   --X"FF", X"FF", X"FF", X"FF",
   --X"FF", X"CE", X"FF", X"FF",
   --X"FF", X"FF", X"FF", X"FF",
   --X"FF", X"FF", X"FF", X"FF"

   -- 50%
   X"10", X"0B", X"0C", X"0E", X"0C", X"0A", X"10", X"0E",
   X"0D", X"0E", X"12", X"11", X"10", X"13", X"18", X"28",
   X"1A", X"18", X"16", X"16", X"18", X"31", X"23", X"25",
   X"1D", X"28", X"3A", X"33", X"3D", X"3C", X"39", X"33",
   X"38", X"37", X"40", X"48", X"5C", X"4E", X"40", X"44",
   X"57", X"45", X"37", X"38", X"50", X"6D", X"51", X"57",
   X"5F", X"62", X"67", X"68", X"67", X"3E", X"4D", X"71",
   X"79", X"70", X"64", X"78", X"5C", X"65", X"67", X"63"
  );

  constant qrom_chr : ROMQ_TYPE :=
  (
   -- 50% for chrominance
  X"11", X"12", X"12", X"18", X"15", X"18", X"2F", X"1A",
  X"1A", X"2F", X"63", X"42", X"38", X"42", X"63", X"63",
  X"63", X"63", X"63", X"63", X"63", X"63", X"63", X"63",
  X"63", X"63", X"63", X"63", X"63", X"63", X"63", X"63",
  X"63", X"63", X"63", X"63", X"63", X"63", X"63", X"63",
  X"63", X"63", X"63", X"63", X"63", X"63", X"63", X"63",
  X"63", X"63", X"63", X"63", X"63", X"63", X"63", X"63",
  X"63", X"63", X"63", X"63", X"63", X"63", X"63", X"63"

  -- 75% chrominance
  --X"09", X"09", X"09", X"0C", X"0B", X"0C", X"18", X"0D",
  --X"0D", X"18", X"32", X"21", X"1C", X"21", X"32", X"32",
  --X"32", X"32", X"32", X"32", X"32", X"32", X"32", X"32",
  --X"32", X"32", X"32", X"32", X"32", X"32", X"32", X"32",
  --X"32", X"32", X"32", X"32", X"32", X"32", X"32", X"32",
  --X"32", X"32", X"32", X"32", X"32", X"32", X"32", X"32",
  --X"32", X"32", X"32", X"32", X"32", X"32", X"32", X"32",
  --X"32", X"32", X"32", X"32", X"32", X"32", X"32", X"32"

  --X"08", X"06", X"06", X"07", X"06", X"05", X"08", X"07", X"07", X"07", X"09",
X"09", X"08", X"0A", X"0C", X"14",
  --X"0D", X"0C", X"0B", X"0B", X"0C", X"19", X"12", X"13", X"0F", X"14", X"1D",
X"1A", X"1F", X"1E", X"1D", X"1A",
  --X"1C", X"1C", X"20", X"24", X"2E", X"27", X"20", X"22", X"2C", X"23", X"1C",
X"1C", X"28", X"37", X"29", X"2C",
  --X"30", X"31", X"34", X"34", X"34", X"1F", X"27", X"39", X"3D", X"38", X"32",
X"3C", X"2E", X"33", X"34", X"32"
   --others => X"01"
  );

    variable data_read  : unsigned(31 downto 0);
    variable data_write : unsigned(31 downto 0);
    variable addr       : unsigned(31 downto 0);


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

```
-- BEGIN
-------------------------------------------------------------------------------
begin
  sim_done <= '0';
  iram_wren <= '0';

  while RST /= '0' loop
    wait until rising_edge(clk);
  end loop;

  for i in 0 to 100 loop
    wait until rising_edge(clk);
  end loop;



  host_read(CLK, X"0000_0000", data_read,
            chipselect, write, readen, address, writedata, readdata);


  host_read(CLK, X"0000_0004", data_read,
            chipselect, write, readen, address, writedata, readdata);

  -- write luminance quantization table
  for i in 0 to 64-1 loop
    data_write := X"0000_00" & qrom_lum(i);
    addr := X"0000_0100" + to_unsigned(4*i,32);
    host_write(CLK, addr, data_write,
            chipselect, write, readen, address, writedata);

  end loop;

  -- write chrominance quantization table
  for i in 0 to 64-1 loop
    data_write := X"0000_00" & qrom_chr(i);
    addr := X"0000_0200" + to_unsigned(4*i,32);
    host_write(CLK, addr, data_write,
            chipselect, write, readen, address, writedata);

  end loop;


  data_write := to_unsigned(1,32) + shift_left(to_unsigned(3,32),1);

  -- SOF & num_comps
  host_write(CLK, X"0000_0000", data_write,
            chipselect, write, readen, address, writedata);

  -- write BUF_FIFO with bitmap
  read_image;

  -- wait until JPEG encoding is done
  host_read(CLK, X"0000_000C", data_read,
            chipselect, write, readen, address, writedata, readdata);
  while data_read /= 2 loop
```

```
      host_read(CLK, X"0000_000C", data_read,
                chipselect, write, readen, address, writedata, readdata);
    end loop;

    sim_done <= '1';

    wait;

  end process;


end architecture RTL;
-------------------------------------------------------------------------------
-- Architecture: end
-------------------------------------------------------------------------------
```

```
-- Quartus II VHDL Template
-- Simple Dual-Port RAM with different read/write addresses but
-- single read/write clock

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity HeaderRam is

	generic
	(
		DATA_WIDTH : natural := 8;
		ADDR_WIDTH : natural := 10
	);

	port
	(
		clk		: in std_logic;
		raddr	: in std_logic_vector((ADDR_WIDTH-1) downto 0) ;
		waddr	: in std_logic_vector((ADDR_WIDTH-1) downto 0);
		d			: in std_logic_vector((DATA_WIDTH-1) downto 0);
		we		: in std_logic := '1';
		q			: out std_logic_vector((DATA_WIDTH -1) downto 0)
	);

end HeaderRam;

architecture rtl of HeaderRam is

	-- Build a 2-D array type for the RAM
	subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
	type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

	-- Declare the RAM signal.
--  signal ram : memory_t := (others => x"FA");

	signal ram : memory_t :=
(0=>x"FF",1=>x"D8",2=>x"FF",3=>x"E0",4=>x"00",5=>x"10",6=>x"4A",7=>x"46",8=>x"49",9=>
x"46",

	10=>x"00",11=>x"01",12=>x"01",13=>x"00",14=>x"00",15=>x"01",16=>x"00",17=>x"01
",18=>x"00",19=>x"00",

	20=>x"FF",21=>x"C0",22=>x"00",23=>x"11",24=>x"08",25=>x"01",26=>x"20",27=>x"01
",28=>x"60",29=>x"03",

	30=>x"01",31=>x"21",32=>x"00",33=>x"02",34=>x"11",35=>x"01",36=>x"03",37=>x"11
",38=>x"01",39=>x"FF",

	40=>x"DB",41=>x"00",42=>x"43",43=>x"00",44=>x"01",45=>x"01",46=>x"01",47=>x"01
",48=>x"01",49=>x"01",
        50=>x"01",51=>x"01",52=>x"01",53=>x"01",54=>x"01",55=>x"01",56=>x"01",57=>x"01
",58=>x"01",59=>x"01",

        60=>x"01",61=>x"01",62=>x"01",63=>x"01",64=>x"01",65=>x"01",66=>x"01",67=>x"01
",68=>x"01",69=>x"01",

        70=>x"01",71=>x"01",72=>x"01",73=>x"01",74=>x"01",75=>x"01",76=>x"01",77=>x"01
",78=>x"01",79=>x"01",

        80=>x"01",81=>x"01",82=>x"01",83=>x"01",84=>x"01",85=>x"01",86=>x"01",87=>x"01
",88=>x"01",89=>x"01",

        90=>x"01",91=>x"01",92=>x"01",93=>x"01",94=>x"01",95=>x"01",96=>x"01",97=>x"01
",98=>x"01",99=>x"01",

        100=>x"01",101=>x"01",102=>x"01",103=>x"01",104=>x"01",105=>x"01",106=>x"01",1
07=>x"01",108=>x"FF",109=>x"DB",

        110=>x"00",111=>x"43",112=>x"01",113=>x"01",114=>x"01",115=>x"01",116=>x"01",1
17=>x"01",118=>x"01",119=>x"01",

        120=>x"01",121=>x"01",122=>x"01",123=>x"01",124=>x"01",125=>x"01",126=>x"01",1
27=>x"01",128=>x"01",129=>x"01",

        130=>x"01",131=>x"01",132=>x"01",133=>x"01",134=>x"01",135=>x"01",136=>x"01",1
37=>x"01",138=>x"01",139=>x"01",

        140=>x"01",141=>x"01",142=>x"01",143=>x"01",144=>x"01",145=>x"01",146=>x"01",1
47=>x"01",148=>x"01",149=>x"01",

        150=>x"01",151=>x"01",152=>x"01",153=>x"01",154=>x"01",155=>x"01",156=>x"01",1
57=>x"01",158=>x"01",159=>x"01",

        160=>x"01",161=>x"01",162=>x"01",163=>x"01",164=>x"01",165=>x"01",166=>x"01",1
67=>x"01",168=>x"01",169=>x"01",

        170=>x"01",171=>x"01",172=>x"01",173=>x"01",174=>x"01",175=>x"01",176=>x"01",1
77=>x"FF",178=>x"C4",179=>x"00",

        180=>x"1F",181=>x"00",182=>x"00",183=>x"01",184=>x"05",185=>x"01",186=>x"01",1
87=>x"01",188=>x"01",189=>x"01",

        190=>x"01",191=>x"00",192=>x"00",193=>x"00",194=>x"00",195=>x"00",196=>x"00",1
97=>x"00",198=>x"00",199=>x"01",

        200=>x"02",201=>x"03",202=>x"04",203=>x"05",204=>x"06",205=>x"07",206=>x"08",2
07=>x"09",208=>x"0A",209=>x"0B",

        210=>x"FF",211=>x"C4",212=>x"00",213=>x"1F",214=>x"01",215=>x"00",216=>x"03",2
17=>x"01",218=>x"01",219=>x"01",

        220=>x"01",221=>x"01",222=>x"01",223=>x"01",224=>x"01",225=>x"01",226=>x"00",2
27=>x"00",228=>x"00",229=>x"00",
        230=>x"00",231=>x"00",232=>x"01",233=>x"02",234=>x"03",235=>x"04",236=>x"05",2
37=>x"06",238=>x"07",239=>x"08",

        240=>x"09",241=>x"0A",242=>x"0B",243=>x"FF",244=>x"C4",245=>x"00",246=>x"B5",2
47=>x"10",248=>x"00",249=>x"02",

        250=>x"01",251=>x"03",252=>x"03",253=>x"02",254=>x"04",255=>x"03",256=>x"05",2
57=>x"05",258=>x"04",259=>x"04",

        260=>x"00",261=>x"00",262=>x"01",263=>x"7D",264=>x"01",265=>x"02",266=>x"03",2
67=>x"00",268=>x"04",269=>x"11",

        270=>x"05",271=>x"12",272=>x"21",273=>x"31",274=>x"41",275=>x"06",276=>x"13",2
77=>x"51",278=>x"61",279=>x"07",

        280=>x"22",281=>x"71",282=>x"14",283=>x"32",284=>x"81",285=>x"91",286=>x"A1",2
87=>x"08",288=>x"23",289=>x"42",

        290=>x"B1",291=>x"C1",292=>x"15",293=>x"52",294=>x"D1",295=>x"F0",296=>x"24",2
97=>x"33",298=>x"62",299=>x"72",

        300=>x"82",301=>x"09",302=>x"0A",303=>x"16",304=>x"17",305=>x"18",306=>x"19",3
07=>x"1A",308=>x"25",309=>x"26",

        310=>x"27",311=>x"28",312=>x"29",313=>x"2A",314=>x"34",315=>x"35",316=>x"36",3
17=>x"37",318=>x"38",319=>x"39",

        320=>x"3A",321=>x"43",322=>x"44",323=>x"45",324=>x"46",325=>x"47",326=>x"48",3
27=>x"49",328=>x"4A",329=>x"53",

        330=>x"54",331=>x"55",332=>x"56",333=>x"57",334=>x"58",335=>x"59",336=>x"5A",3
37=>x"63",338=>x"64",339=>x"65",

        340=>x"66",341=>x"67",342=>x"68",343=>x"69",344=>x"6A",345=>x"73",346=>x"74",3
47=>x"75",348=>x"76",349=>x"77",

        350=>x"78",351=>x"79",352=>x"7A",353=>x"83",354=>x"84",355=>x"85",356=>x"86",3
57=>x"87",358=>x"88",359=>x"89",

        360=>x"8A",361=>x"92",362=>x"93",363=>x"94",364=>x"95",365=>x"96",366=>x"97",3
67=>x"98",368=>x"99",369=>x"9A",

        370=>x"A2",371=>x"A3",372=>x"A4",373=>x"A5",374=>x"A6",375=>x"A7",376=>x"A8",3
77=>x"A9",378=>x"AA",379=>x"B2",

        380=>x"B3",381=>x"B4",382=>x"B5",383=>x"B6",384=>x"B7",385=>x"B8",386=>x"B9",3
87=>x"BA",388=>x"C2",389=>x"C3",

        390=>x"C4",391=>x"C5",392=>x"C6",393=>x"C7",394=>x"C8",395=>x"C9",396=>x"CA",3
97=>x"D2",398=>x"D3",399=>x"D4",

        400=>x"D5",401=>x"D6",402=>x"D7",403=>x"D8",404=>x"D9",405=>x"DA",406=>x"E1",4
07=>x"E2",408=>x"E3",409=>x"E4",
        410=>x"E5",411=>x"E6",412=>x"E7",413=>x"E8",414=>x"E9",415=>x"EA",416=>x"F1",4
17=>x"F2",418=>x"F3",419=>x"F4",

        420=>x"F5",421=>x"F6",422=>x"F7",423=>x"F8",424=>x"F9",425=>x"FA",426=>x"FF",4
27=>x"C4",428=>x"00",429=>x"B5",

        430=>x"11",431=>x"00",432=>x"02",433=>x"01",434=>x"02",435=>x"04",436=>x"04",4
37=>x"03",438=>x"04",439=>x"07",

        440=>x"05",441=>x"04",442=>x"04",443=>x"00",444=>x"01",445=>x"02",446=>x"77",4
47=>x"00",448=>x"01",449=>x"02",

        450=>x"03",451=>x"11",452=>x"04",453=>x"05",454=>x"21",455=>x"31",456=>x"06",4
57=>x"12",458=>x"41",459=>x"51",

        460=>x"07",461=>x"61",462=>x"71",463=>x"13",464=>x"22",465=>x"32",466=>x"81",4
67=>x"08",468=>x"14",469=>x"42",

        470=>x"91",471=>x"A1",472=>x"B1",473=>x"C1",474=>x"09",475=>x"23",476=>x"33",4
77=>x"52",478=>x"F0",479=>x"15",

        480=>x"62",481=>x"72",482=>x"D1",483=>x"0A",484=>x"16",485=>x"24",486=>x"34",4
87=>x"E1",488=>x"25",489=>x"F1",

        490=>x"17",491=>x"18",492=>x"19",493=>x"1A",494=>x"26",495=>x"27",496=>x"28",4
97=>x"29",498=>x"2A",499=>x"35",

        500=>x"36",501=>x"37",502=>x"38",503=>x"39",504=>x"3A",505=>x"43",506=>x"44",5
07=>x"45",508=>x"46",509=>x"47",

        510=>x"48",511=>x"49",512=>x"4A",513=>x"53",514=>x"54",515=>x"55",516=>x"56",5
17=>x"57",518=>x"58",519=>x"59",

        520=>x"5A",521=>x"63",522=>x"64",523=>x"65",524=>x"66",525=>x"67",526=>x"68",5
27=>x"69",528=>x"6A",529=>x"73",

        530=>x"74",531=>x"75",532=>x"76",533=>x"77",534=>x"78",535=>x"79",536=>x"7A",5
37=>x"82",538=>x"83",539=>x"84",

        540=>x"85",541=>x"86",542=>x"87",543=>x"88",544=>x"89",545=>x"8A",546=>x"92",5
47=>x"93",548=>x"94",549=>x"95",

        550=>x"96",551=>x"97",552=>x"98",553=>x"99",554=>x"9A",555=>x"A2",556=>x"A3",5
57=>x"A4",558=>x"A5",559=>x"A6",

        560=>x"A7",561=>x"A8",562=>x"A9",563=>x"AA",564=>x"B2",565=>x"B3",566=>x"B4",5
67=>x"B5",568=>x"B6",569=>x"B7",

        570=>x"B8",571=>x"B9",572=>x"BA",573=>x"C2",574=>x"C3",575=>x"C4",576=>x"C5",5
77=>x"C6",578=>x"C7",579=>x"C8",

        580=>x"C9",581=>x"CA",582=>x"D2",583=>x"D3",584=>x"D4",585=>x"D5",586=>x"D6",5
87=>x"D7",588=>x"D8",589=>x"D9",
	590=>x"DA",591=>x"E2",592=>x"E3",593=>x"E4",594=>x"E5",595=>x"E6",596=>x"E7",5
97=>x"E8",598=>x"E9",599=>x"EA",

	600=>x"F2",601=>x"F3",602=>x"F4",603=>x"F5",604=>x"F6",605=>x"F7",606=>x"F8",6
07=>x"F9",608=>x"FA",609=>x"FF",

	610=>x"DA",611=>x"00",612=>x"0C",613=>x"03",614=>x"01",615=>x"00",616=>x"02",6
17=>x"11",618=>x"03",619=>x"11",

	620=>x"00",621=>x"3F",622=>x"00", others => x"00");

begin

	process(clk)
	begin
	if(rising_edge(clk)) then
		if(we = '1') then
			ram(to_integer(unsigned(waddr))) <= d;
		end if;

		-- On a read during a write to the same address, the read will
		-- return the OLD data at the address
		q <= ram(to_integer(unsigned(raddr)));
	end if;
	end process;

end rtl;

library ieee;
	use ieee.std_logic_1164.all;
	use ieee.numeric_std.all;

entity OutputRAM is

	generic
	(
		DATA_WIDTH : natural := 8;
		ADDR_WIDTH : natural := 14
	);

	port
	(
		clk		: in std_logic;
		raddr	: in std_logic_vector(ADDR_WIDTH-1 downto 0);
		waddr	: in std_logic_vector(ADDR_WIDTH-1 downto 0);
		data	: in std_logic_vector((DATA_WIDTH-1) downto 0);
		we		: in std_logic := '1';
		q		: out std_logic_vector((DATA_WIDTH -1) downto 0)
	);

end OutputRAM;

architecture rtl of OutputRAM is
```

```
       -- Build a 2-D array type for the RAM
       subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
       -- 16KB
       type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

       -- Declare the RAM signal.
       signal ram : memory_t;

begin

       process(clk)
       begin
       if(rising_edge(clk)) then
             if(we = '1') then
                   ram(to_integer(unsigned(waddr))) <= data;
             end if;

             -- On a read during a write to the same address, the read will
             -- return the OLD data at the address
             q <= ram(to_integer(unsigned(raddr)));
       end if;
       end process;

end rtl;

library ieee;
  use ieee.std_logic_1164.all;

entity SavePixel is
  port
  (

             CLK                        : in   std_logic;
             RST                        : in   std_logic;

             --Avalon
             chipselect                 : in   std_logic;
             write                      : in   std_logic;
             read                       : in   std_logic;
             address                    : in   std_logic_vector(7 downto 0);
             writedata                  : in   std_logic_vector(31 downto 0);
             readdata                   : out  std_logic_vector(31 downto 0);

             -- HOST DATA
             iram_wdata          : out std_logic_vector(23 downto 0);   --
C_PIXEL_BITS
             iram_wren           : out std_logic;
             fifo_almost_full    : in  std_logic

       );

end SavePixel;

architecture rtl of SavePixel is
```

```
	signal asserted			: std_logic := '0';

begin

	process(CLK)
	begin
		if rising_edge(CLK) then
			if RST = '1' then
				iram_wdata	<= (others => '0');
				iram_wren	<= '0';
				readdata	<= (others => '0');
				asserted	<= '0';
			else
				if chipselect = '1' then
					if read = '1' then
						readdata	<= X"0000000" & "000" &
fifo_almost_full;
						iram_wren	<= '0';
						iram_wdata	<= (others => '0');
						asserted	<= '0';
					end if;

					if write = '1' then
						--must be asserted for 1 cycle
						if asserted = '0' then
							iram_wdata	<= writedata(23 downto 0);
							iram_wren	<= '1';

							readdata	<= (others => '0');
							asserted	<= '1';
						else
							iram_wdata	<= (others => '0');
							iram_wren	<= '0';

							readdata	<= (others => '0');
						end if;
					end if;
				else
					iram_wdata	<= (others => '0');
					iram_wren	<= '0';
					readdata	<= (others => '0');
					asserted	<= '0';
				end if;
			end if;
		end if;
	end process;

end rtl;

library ieee;
  use ieee.std_logic_1164.all;

entity AvalonTrans is
  port
  (
            CLK                 : in  std_logic;
            RST                 : in  std_logic;

            --Avalon
            chipselect          : in   std_logic;
            write               : in   std_logic;
            read                : in   std_logic;
            address             : in   std_logic_vector(15 downto 0);
            writedata           : in   std_logic_vector(31 downto 0);
            readdata            : out  std_logic_vector(31 downto 0);

            -- OPB
            OPB_DBus_out : in   std_logic_vector(31 downto 0);
            OPB_XferAck  : in   std_logic;
            OPB_retry           : in   std_logic;
            OPB_toutSup  : in   std_logic;
            OPB_errAck          : in   std_logic;
            OPB_ABus            : out  std_logic_vector(31 downto 0);
            OPB_BE                     : out  std_logic_vector(3 downto 0);
            OPB_DBus_in  : out  std_logic_vector(31 downto 0);
            OPB_RNW             : out  std_logic;
            OPB_select          : out  std_logic

      );

end AvalonTrans;

architecture rtl of AvalonTrans is

begin

      OPB_ABus <= (X"0000" & address) when chipselect = '1' else (others => '0');
      OPB_BE <= X"F" when chipselect = '1' else X"0";
      OPB_RNW      <= '1' when chipselect = '1' and read = '1' else '0';
      OPB_DBus_in <= writedata when chipselect = '1' and write = '1' else (others =>
'0');
      readdata <= OPB_DBus_out when chipselect = '1' and read = '1' else (others =>
'0');
      OPB_select <= chipselect;

end rtl;

library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

entity ReadRam is
  port
  (
            CLK                               : in   std_logic;
            RST                               : in   std_logic;

            --Avalon
            chipselect                        : in         std_logic;
```

```
             read                             : in         std_logic;
             address                          : in         std_logic_vector(13 downto
0);          --outram 14 bits
             readdata                         : out  std_logic_vector(31 downto 0);

             --OutputRam
             raddr                            : out  std_logic_vector(13 downto 0);
             --outram 14 bits
             q                                       : in         std_logic_vector(7
downto 0)
       );

end ReadRam;

architecture rtl of ReadRam is

       signal memdata : std_logic_vector(7 downto 0);
       signal ready   : integer := 0;

begin

       process(CLK)
       begin
             if rising_edge(CLK) then

                   if RST = '1' then
                         raddr               <= (others => '0');
                         readdata            <= (others => '0');
                         ready               <= 0;
                   end if;

                   if chipselect = '1' and read = '1' then
                         ready               <= ready + 1;
                         raddr               <= address;

                         if ready > 1 then
                               readdata     <= X"000000" & q;
                         else
                               readdata     <= (others => '0');
                         end if;

                   else
                         raddr               <= (others => '0');
                         readdata            <= (others => '0');
                         ready               <= 0;
                   end if;

             end if;
       end process;

end rtl;

#include <io.h>
#include <system.h>
#include <stdio.h>
```

```c
#include <stdlib.h>

#define SAVEPIXEL(data)             IOWR_32DIRECT(SAVEPIXEL_BASE, 0, (data))
#define READQUEUE                   IORD_32DIRECT(SAVEPIXEL_BASE, 0)
#define READBYTE(offset)            IORD_32DIRECT(READRAM_BASE, (offset) * 4)

#define SAVEREG(address, data)      IOWR_32DIRECT(AVALONTRANS_BASE, (address) *4,
(data))
#define READREG(address)            IORD_32DIRECT(AVALONTRANS_BASE, (address) *4)

#define ENC_START_REG       0x0000
#define IMAGE_SIZE_REG      0x0004
#define ENC_STS_REG         0x000C
#define ENC_LENGTH_REG      0x0014
#define QUANTIZER_RAM_LUM   0x0100
#define QUANTIZER_RAM_CHR   0x0200

const char luminance[] = {
   0x10, 0x0B, 0x0C, 0x0E, 0x0C, 0x0A, 0x10, 0x0E,
   0x0D, 0x0E, 0x12, 0x11, 0x10, 0x13, 0x18, 0x28,
   0x1A, 0x18, 0x16, 0x16, 0x18, 0x31, 0x23, 0x25,
   0x1D, 0x28, 0x3A, 0x33, 0x3D, 0x3C, 0x39, 0x33,
   0x38, 0x37, 0x40, 0x48, 0x5C, 0x4E, 0x40, 0x44,
   0x57, 0x45, 0x37, 0x38, 0x50, 0x6D, 0x51, 0x57,
   0x5F, 0x62, 0x67, 0x68, 0x67, 0x3E, 0x4D, 0x71,
   0x79, 0x70, 0x64, 0x78, 0x5C, 0x65, 0x67, 0x63
};

const char chrominance[] = {
  0x11, 0x12, 0x12, 0x18, 0x15, 0x18, 0x2F, 0x1A,
  0x1A, 0x2F, 0x63, 0x42, 0x38, 0x42, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63,
  0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63, 0x63
};

//const unsigned int image[];
unsigned char raster[3][336][232];
unsigned char jpeg_image[17000];

//unsigned char raster[3][3][2];
//unsigned char jpeg_image[1];

int compress_jpeg(unsigned char[][][], unsigned char[]);

int main() {
    compress_jpeg(raster, jpeg_image);
}

int compress_jpeg(unsigned char input[][][], unsigned char output[]) {
    volatile unsigned int i, j, k, m, temp4;//, temp1, temp2, temp3, pos=0;
```

```
i = READREG(ENC_STS_REG);
printf("enc_status_reg = %x\n", i );

i = READREG(ENC_START_REG);
printf("enc_start_reg = %x\n", i );

i = READREG(IMAGE_SIZE_REG);
printf("image_size_reg = %x\n", i );

printf("Writing Image size 336x232 pixels\n");
SAVEREG(IMAGE_SIZE_REG,0x015000E8);

i = READREG(IMAGE_SIZE_REG);
printf("image_size_reg = %x\n", i );

i = READREG(ENC_LENGTH_REG);
printf("\nlength = %u\n", i);


printf(" loading luminance RAM\n");
for (j=0; j<64; j++) {
    SAVEREG(QUANTIZER_RAM_LUM+j*4, luminance[j]);
    printf(".");
}
printf("\n luminance RAM saved\n");

i = READREG(ENC_STS_REG);
printf("enc_status_reg = %x\n", i );

printf(" Loading chrominance RAM\n");

for (j=0; j<64; j++) {
    SAVEREG(QUANTIZER_RAM_CHR+j*4, chrominance[j]);
    printf(".");
}
printf("\n chrominance RAM saved\n");

i = READREG(ENC_STS_REG);
printf("enc_status_reg = %x\n", i );


printf ("Starting encoder\n");
SAVEREG(ENC_START_REG, 0x7);

i = READREG(ENC_START_REG);
printf("enc_start_reg = %x\n", i );

i = READREG(ENC_STS_REG);
printf("enc_status_reg = %x\n", i );

i = READREG(IMAGE_SIZE_REG);
printf("image_size_reg = %x\n", i );

i = READREG(ENC_STS_REG);
printf("enc_status_reg = %x\n", i );
```

```
        for (i=0; i<232; i++) { //y
            for (j=0; j<336; j++) { //x
//                temp1 = image[pos] & 0xff;
//                temp2 = image[pos] & 0xff00;
//                temp3 = image[pos++] & 0xff0000;
//                temp4 = (temp3 >> 16) | temp2 | (temp1 << 16);
                temp4 = raster[0][j][i] | raster[1][j][i] | raster[2][j][i];
                //printf("%x\n",temp4);
                SAVEPIXEL(temp4);

                if ((k = READQUEUE)) {
                    m =10000;
                    while (m--);
                }
                //printf("%d, %d\n",i,j);
            }

            //getchar();
            m =1000;
            while (m--);
            //printf(".");
        }

        printf("\n");
        k = 0;
        //getchar();
        //while (k < 2) {
            k = READREG(ENC_STS_REG);
            printf(".");
            m =100000;
            while (m--);
        //}

        i = READREG(ENC_STS_REG);
        printf("\nenc_status_reg = %x\n", i );
                m =100000;
            while (m--);

        i = READREG(ENC_LENGTH_REG);
        printf("\nlength = %u\n", i);

        m=0;
        //printf("\n %05x:   ",m);
        for (j=0; j<i+2; j++) {
            m++;
            k = READBYTE(j);
            output[j]=k;
            //printf("%02X ",k);
            printf("%02X ",output[j]);
            if (m%16==0) printf("\n");   //printf("\n %05x:   ",m);
        }

        i = READREG(ENC_STS_REG);
        printf("\nenc_status_reg = %x\n", i );
```

```
    i = READREG(ENC_LENGTH_REG);
    printf("\nlength = %u\n", i);

    printf("\ndone\n");

  return 0;
}
```