

Imperium Accelero 9000 Image Processor

EE 552: Final Report

Ian Ferguson : ianf@ualberta.ca

Nathan Liesch : nliesch@ualberta.ca

Luis Ramirez : lr Ramirez@uoa@hotmail.com

Mark Willson : mdwillson@shaw.ca

Declaration of Original Content

The design elements of this project and report are entirely the original work of the authors, except as follows:

- Symmetrical 3x3 Convolver diagram from Atmel [9]
- crt_test.vhd based upon DPF Group [8]
- 8-bit VGA circuit based upon Carson, Benj [15]
- VGA timing constants based upon VESA [16]
- RS232 code based upon NNTrainer Group [12]
- VDAC external circuitry based on Analog Devices [7]
- Lcd module based on CDMA lcd driver[18]
- Debounce.vhd uses dual configurable debouncer developed by Reading Book Grp [20]
- lpm_ram vhd created from Altera Quartus II

Signed:

Luis Ramirez	DATE
--------------	------

Ian Ferguson	DATE
--------------	------

Nathan Liesch	DATE
---------------	------

Mark Willson	DATE
--------------	------

Abstract

In the computationally-intensive graphics industry, complex image processing demands hardware innovations in order to improve performance. The goal of the Imperium Accelero 9000 image processor is to accelerate the process of filtering an image on a PC by using a dedicated FPGA. The benefit of using FPGA generated hardware is that it can be easily customized to suit a wide variety of digital signal processing (DSP) applications, including digital image processing. The intent of the IA9k is to allow the filtering algorithms make optimal use of the hardware in order to streamline the overall process of filtering a digital image. The IA9k chip uses linear and 3x3 convolution modules to implement color inversion, emboss, sharpen, blur, and edge recognition filters on 96x64 (pixels) 8-bit grayscale images. The chip is also capable of displaying the original and filtered images on a monitor using a custom 8-bit VGA output circuit. Images are transferred between PC and IA9k using a RS-232 link to download/upload uncompressed bitmap images.

This project was successfully implemented using an Altera APEX20KE200 FPGA mounted on a NIOS development board. The final implementation of the IA9K used 91% of the available logic elements, 92 % of the memory elements, and consisted of approximately 4900 lines of VHDL code.

Table of Contents

<u>Achievements</u>	5
<u>Description of Operation</u>	7
<u>Features</u>	12
<u>FPGA Logic and Memory Usage</u>	14
<u>Design Verification</u>	15
<u>External HDL Components</u>	21
<u>Experiments and Characterization</u>	22
<u>References</u>	28
<u>Appendix</u>	30
<u>VHDL Code Index</u>	32
1.0 User Interface Code	33
2.0 Display Interface Code.....	34
3.0 Memory Interface Code	35
4.0 RS232 Interface Code	36
5.0 Filter Interface Code	38
<u>Test Cases/Simulation Index</u>	40
1.0 User Interface Simulations.....	41
2.0 Display Interface Simulations	42
3.0 Memory Interface Simulations	43
4.0 RS232 Interface Simulations	44
5.0 Filter Interface Simulations	45
6.0 Convolver Calculation Simulations	47
6.0 Negative Filter Simulations	48
<u>VHDL Testbench Index</u>	49
1.0 User Interface Testbench	50
2.0 Memory Interface Testbench.....	51
3.0 RS232 Interface Testbench.....	52
4.0 Filter Interface Testbench	53
5.0 Convolver Calculation Testbench.....	54

Achievements

During the design of the Imperium Accelero several achievements were made:

The display unit has been instantiated and thoroughly tested, since ultimately it will be used to show how well the filters are working. The display module interface testing program was written for both 8 bits color or grayscale and 24 bit color. This diversity came from experimenting with a 24 bit VDAC that was going to be implemented to achieve a more impressive demonstration. Simulations and a stable output image have confirmed the correct synchronization pulses timing and functioning of the display module.

After long hours of laboring over a previously developed SDRAM controller, it was found that there were problems beyond the understanding of the teaching staff and development crew. The SDRAM controller was simulated numerous times and found to have the correct functionality. Pin assignments were reviewed three times, and the controller was re-written numerous times to see if approaching the same problem with different angles would provide additional insight to previous shortcomings. However, the FPGA was not able to properly interface with the SDRAM. Possible problems were narrowed down to: the RAM not working properly and/or the previously developed SDRAM controller did not have the correct timing information. Though the SDRAM controller did not work, there was a lot of knowledge gained about RAM, and its proper function.

Since the SDRAM controller took up three weeks and there were no additional testing procedures to try. It was dropped for LPM_RAM. The LPM_RAM took about an hour to implement and a memory controller took a day to develop. The ram was much easier to implement, and, because it was all internal it was easier to simulate and debug. Use of the LPM_RAM sped up the development process.

Considerable design time has been spent in the convolver calculation circuitry. Completed design of convolution calculation was achieved through a number of iterations in development. The circuit has highly optimized performance to achieve speeds faster than system clock. The logic elements used by the convolver have also been improved through various register optimizations while maintaining maximum flexibility in algorithm, to be able to implement a number of different filters.

Visual characterization of filter performance has not completed been completed. There are numerous parameters that can be adjusted in order to tweak the appearance of the filtered image and it will be necessary to adjust them for the characteristics of our system. Further development is required to interface the filters with the memory controller and, lastly, with the rest of the circuit.

The RS232 input implementation took a month of testing to complete development and implementation. While code from previous groups was taken, many changes were made to removed the usage of FIFOs in the code as well as implement features such as time out processes and packet counters. Also, a status stage logic process was developed for the

user interface to have increased control over the function of the RS232 module. The decision to include the ability to send data back to the PC was made late in the project. Currently the code is developed and in simulation with a few timing errors still present in the code which have not allowed for a successful test. These will hopefully be complete for the demonstration of the project.

Increased knowledge of the program MITY has been gained through the debugging process of the RS232 implementation. MITY is a versatile program that allows the transfer of data from the computer to another RS232 device. Much has also been learned about the RS232 protocol itself and the format of RS232 packets and timing characteristics when up/downloading.

Due to the nature of the project, additional research has gone into the bitmap file format. It was found that ideally the image is stored in a 24 bit format. With this in mind, the project was originally developed with 24 bit operations. However, due to development setback, the design has shifted to 8 bit. This has led to researching how to take the 24 bit image and convert it into 8 bits. C code was developed to create a mif file to be used in early testing, and to create an image that can be downloaded into the FPGA. Later, code was developed to convert the uploaded image into a bmp.

Description of Operation

The overall purpose of the design is to import an image, apply any number of filters to it, and display the filtered picture. The main objective is to speed up the filtering process by increasing the level of parallelism in the filtering algorithms.

A top view of the filtering process:

1. The FPGA takes a picture from a computer using RS232 and stores it into RAM.
2. The FPGA waits for further instructions from the user.
3. The user can display the stored picture, to confirm it was downloaded correctly (this step can be skipped)
4. A filter is selected and applied. This involves some, on the fly, configuring of the FPGA to set the filters to the proper algorithm.
5. The filtered image is displayed.
6. The user may now repeat steps 3-5 or upload the picture back to the PC via RS232.

By breaking up the circuit into sub circuits, design time was maximized between the individual members of the design team. The circuit was broken into the following sub-circuits: the user interface, the memory controller, the LCD, the display module, the RS232 communication, and the filters.

The following section outlines the basic circuits that were developed. For design information please refer to Design Verification.

The User Interface:

The first function of the user interface is to properly capture user input and provide the user with a visual indication of the current status of the IA9K. The interface consists of a LCD screen to display menu / status screens, and two push buttons (Enter and Select) to allow the user to navigate the menus and control the IA9K. The LCD menus allow the user to step through the downloading of an image, selecting a filtering algorithm, and uploading an image if the user chooses to do so.

A flowchart is provided on the following page to better illustrate the control flow and menu interactions present in the User Interface.

The second function of the user interface is to interpret data received from the user and enable/disable the correct sub modules required to process the user's request. The user interface is also responsible for connecting all of the signals between sub modules and using multiplexers to decide which data paths are used since the sub modules may receive data from multiple modules. However, the actual handshaking and synchronization between enabled modules is left to the individual modules themselves. This decision was made in an attempt to reduce the complexity of the user interface since the requirements of the individual modules is quite varied and in many cases specific to each module.

A schematic is provided on the following page to provide an overview of the sub-modules and their interconnections within the User Interface

The LCD:

The LCD module uses a 1-line 16 character text message format to displays the options available to the user and the current status of the Imperium Accelero on the LCD screen. The module is also responsible for powering up the display, writing the characters to the screen, and refreshing the screen when needed. The LCD menu system was chosen over using CRT video-overlay menus in order to reduce complexity of the display module and to allow for more development time on the filter module.

The Memory Controller:

Due to problems with the SDRAM controller, LPM_RAM was used as an alternative form of storage. The LPM_RAM stores the original picture and the modified picture simultaneously to allow for further modification of the original image beyond the first filter. This duality of picture also allows the user to be able to see the original image and the filtered image to evaluate the filter's effect. However, due to this functionality, the size of picture the FPGA would be able to store would be limited to an 64x96 pixel image, with 8 bit, monochrome picture depth.

The LPM_RAM is a way of storing information that has no external interface to the rest of the circuit, thus, a memory controller was develop interface the LPM_RAM. There are three distinct ways memory is accessed: to store information from RS232 or from the filters, to display the image, or to read the image from ram into the filters. Writing differs from the other two simply because it stores data into the ram, it also uses a handshaking bit to make sure the data going into the ram is valid data. Displaying the data required specific access of data, but at very specific times. This is due to the horizontal and vertical sync that are required to interface to the monitor. This particular access mode continuously wraps itself around to allow the image to be displayed. Therefore, it requires the ram to continuously be ready to send data. There are two handshaking signals, pixel request and next row, that are used to interface the memory to the display module. These handshaking bits allow data to stay synchronize with the display. Lastly, the filters need data be ready for it constantly, but only the data only needs to be fed into the filters once. Therefore, a simple read signal is sent to the memory controller to initialize the data transfer. Once the memory has started, it puts new data on the data bus every three clock cycles until it reaches the end of the picture.

The Display Module:

The Display module is responsible for sending the correct output to an external circuit that takes this output and converts it to the correct voltage levels for proper display on a CRT. The display module was originally developed to work with a VDAC. However, it is currently used in conjunction with a simple resistor circuit to implement 8 bit grayscale. Grayscale is currently best suited to show the true capabilities of the filters as this allows 255 possible brightness levels which greatly minimizes rounding error and improves image quality.

RS232:

The RS232 handles the download/upload of the picture. Data is send using a software application called MTTY. The RS232 input modules then are responsible for recognizing data send in RS232 protocol format and converting it to a serial signal to be

sent to RAM. As the modules receive data at 9600 baud, the global clock signal must be divided down significantly to compensate for the slow speed of the input data. Input data is stored into a shift register and once full is put onto the data bus and a handshake signal to the RAM is asserted notifying it that the data on the line is valid. The module is programmed with a 10 second idle time out which will notify the control unit if the input line has remained idle for too long as well as a packet counter to confirm when the image has been received. Once a filtered image is stored in RAM, the IA9K will also have the ability to send the image back to a PC using MTTY to receive the file. The RS232 output modules send handshake signals to the RAM requesting 24 bits of data. These signals are broken into 3 8 bit signals which are shifted out onto the serial port once they have been formatted to meet RS232 protocol.

Filters:

The filters take the original picture data and manipulate it according the specified filter. It is responsible for enabling the convolution or linear module necessary to give the desired effect and setting the weight of the 3x3 convolution coefficients if convolution is used. The convolution and linear modules are responsible for handling their own memory requests concerning the original and filtered image data since they each have different data requirements. For example, the convolution uses a large shift register to gather a 3x3 grid of image data while the linear filters only require 3 pixels at a time to process an image (*data requirements for each module are discussed further below*).

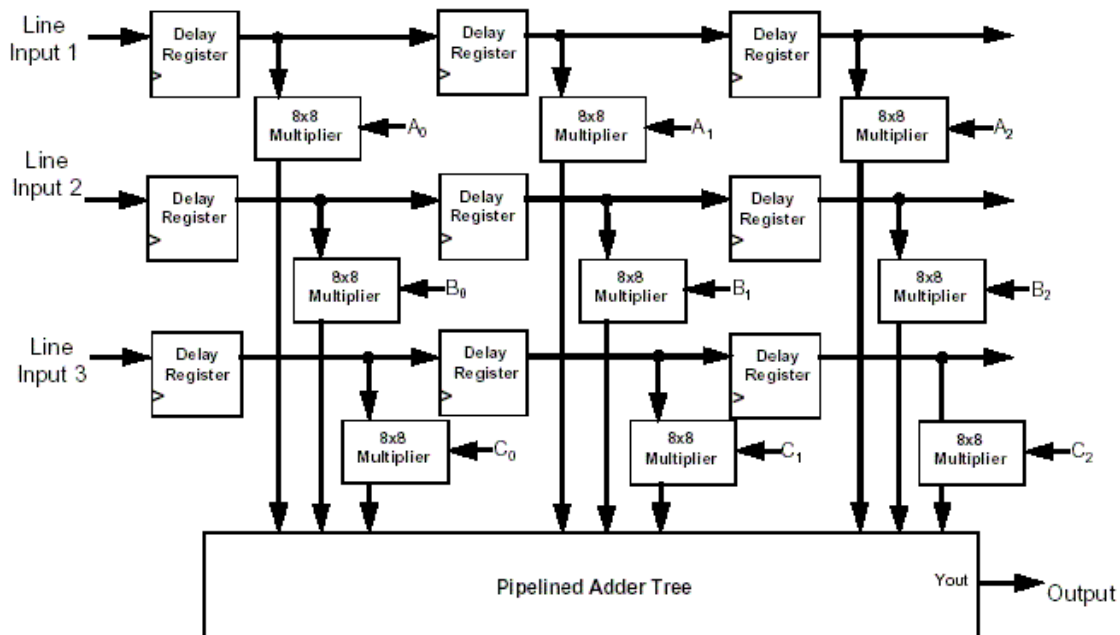
Linear or Pixel-by-Pixel:

Each pixel of the filtered image can be determined independently of surrounding pixels and thus processes the data in a linear or pixel-by-pixel manner. The negative filter has been implemented which subtracts the pixel value from the maximum possible value of 255.

Convolver:

The convolver lies at the heart of the filtering design. It takes data in from the original image, from the memory controller, computes through a 3x3 convolution and returns a new center pixel every 3 cycles. This filtered pixel is sent to another memory controller to store the filtered image, so that a display unit can put it on the screen. The block diagram for this full-programmable convolver is given below.

Fully-Programmable 3x3 Convolver



Reference: Atmel [9];

The coefficients of the convolver matrix can be changed to provide a wide variety of effects upon the image. The following are typical convolver matrix coefficients that have been implemented.

heavy blur matrix

```
1 2 1
2 4 2
1 2 1
```

light blur matrix

```
1 1 1
1 1 1
1 1 1
```

edge enhance matrix

```
-1 -1 -1
-1 9 -1
-1 -1 -1
```

emboss matrix

```
-1 0 0
0 0 0
0 0 1
```

find edges matrix

```
-1 -1 -1
-1 8 -1
-1 -1 -1
```

sharpen matrix

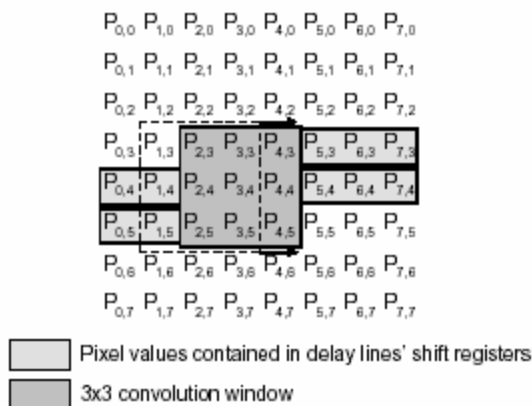
```
-1 -2 -1
-2 16 -2
-1 -2 -1
```

Reference: <http://dot.tt.hut.fi/dip/> [10]

The switch from SDRAM to a much smaller picture in lpm_ram also enabled a change in the way in which the image data is accessed. The original top-level convolver design

called for accessing three lines of the picture each clock cycle. This would have been a very inefficient method of memory access and could have drastically reduced overall system performance if memory retrieval was a bottleneck for the system. Instead with the smaller image pixel values are fed line by line, from top to bottom, until 2 complete lines and the first 3 pixels of a third line are contained within a series of shift registers. At that point, all the pixels belonging to the first 3x3 convolution window are available inside the coprocessor. From that moment on, each new pixel value inserted into the chain of shift registers effectively displaces the convolution window to a new adjacent position until the whole image has been visited. This type of data access could easily be extended to larger image. The image would be split up into several vertical strips of width of the shift register minus a pixel for overlap and each of these strips would be computed using this technique. This would retain the advantages of reduced memory access while allowing more practical sized images to be processed. If SDRAM were later to be functional this algorithm for memory access would likely simply be extended to a larger image.

As part of the filter module the convolver module will instantiate a shift register to obtain a 3x3 window of original pixel data from memory to pass on to the actual convolution calculation. The shift register will require 2 complete rows of pixel data plus 3 additional pixels (shown below) to store the 3x3 window.



Reference: http://www.grm.polymtl.ca/circus/data/MUG1997_paper.pdf [19]

Features

The goal of the project was to output the image to a CRT along with some menu commands displayed on a separate LCD screen. The CRT display code was designed to output a 640x480 pixel 24-bit color image however the memory restrictions resulted in only a 96x64 pixel 8 bit grayscale image being stored and filtered. To utilize the maximum amount of the screen, the image data was repeated for six pixels both horizontally and vertically for a display of 576x384.

The primary purpose of the project is to implement a digital image filter. The main filter algorithm is a convolution style filter which is known to be very processor intensive and would be very suitable for hardware acceleration. Simply by adjusting the convolution kernel values a wide range of filtering effects can be achieved. Additionally, a more

straightforward class of filters that are applied pixel by pixel have been implemented. The list of filters supported include: negative, emboss, sharpen, blur, and edge detection.

The system downloads an uncompressed image file from a PC initially using an RS-232 interface. This was fairly easy to implement using the RS232 cores developed by previous groups, with the only drawback being the slow rate at which the data is transferred. Ideally the FPGA would have been connected directly to the system PCI bus to allow even greater data transfer rates however the budget for our project does not allow for that type of implementation.

It was necessary to provide some sort of control system that will allow user input for which filter to apply and when to download a new image for the host computer. The menu commands are text-based and displayed on a small LCD screen provided with the Altera Apex20KE FPGA.

FPGA Logic and Memory Usage

The 3x3 convolution calculation unit is designed with 8bit inputs and 9bit signed convolution coefficients and takes up 3048 logic elements. The convolution will also require a shift register that will take approximately 1000 logic elements if the size image is chosen as 64x96.

VGA display module as compiled uses 257/8,320 logic elements.

Due to switching to lpm ram, the memory controller now takes up 413 logic elements. However, there are also 49152 memory elements that are also used up due to the switch. Since this controller will be instantiated twice, the total logic elements required are 926 and 98304 memory elements.

RS 232 code was compiled and found to take up approximately 2% of the total logic elements. 262 logic elements are used in the implementation.

The LE usage of the user interface consists primarily of the circuit needed to drive the lcd screen, with the state logic and multiplexers necessary to enable / disable the other modules attributing to a small amount of the UI logic. The lcd driver was implemented purely in combinational logic due to the constraints placed on internal memory. While the initial intent was to use an lcd driver to take advantage of lpm_rom and maximize the number logic elements available, it was dropped in favor of the combinational lcd driver in order to maximize the available EAB's instead. Maximizing the number of EAB's took precedence because of the restriction placed on the circuit by using internal lpm ram to store image data as opposed to external SDRAM. The final implementation of the UI module consisted of 543 logic elements.

Module	Logic elements	Memory elements
RS232	262	0
Display	257	0
Memory Controller	926	98304
Filter (Conv + Neg)	5565	0
User Interface	534	0
Total	7544	98304

The Total Number of Logic Elements Used is 7544 which amounts to 91 % of the of the 8320 available logic blocks contained in the Altera Apex20ke fpga.

Memory usage is 98,304 / 106,496 amounting to 92 % of the total EAB's.

Design Verification

User Interface

The lcd component of the user interface was based on the CDMA group's lcd driver. The code was modified to run using the 33.333 MHz NIOS clock as opposed to the 25 MHz clock for which the module was originally designed for. The lcd display was chosen to be a single line 16 character display which was sufficient to display the menu screens and adequately describe the status of the image processor. The 33.333 MHz clock had to be divided down to 2.5 kHz in order to satisfy the refresh requirements of the lcd and the display screens were changed to reflect the IA9K menus.

Next, the dual configurable debouncer was configured to suit the timings of the NIOS board. The debouncer was set to divide the 33.333 MHz frequency by 2^{16} using a 16 bit wide `lpm_counter`. Dividing the clock meant that the button signal was sampled at roughly 1.9 ms, enough to miss the effect of the bouncing signal which occurs for approximately 1 ms after a button is pressed. The second part of the configuration included setting the buffer to collect 20 samples during the duration of the button press and take the most frequently occurring value as the result. Increasing the amount buffering helps to throw away any residual bouncing in the signal or inadvertent button taps, however, buffering a button too much can result in missed button presses if the button is not held long enough to fill the buffer.

The final component of the user Interface was to incorporate the other sub modules into the push button LCD menu. In order to reduce the complexity in the user interface code it was decided early in the project that each module would have its own enable signal. When enabled each unit is responsible for communicating between other entities and determining its own status. After enabling a set of modules the user interface will wait until those modules report a fail or success status back and then disable each module, effectively suspending / resetting the module until the next time it is enabled.

Memory Controller

The memory controller is responsible for interfacing the `lpm ram` with the rest of the circuit. The memory controller was developed to take up most of the room the FPGA had available. This resulted in a 24bit wide data bus with an address width of 11bits. The total memory of a picture then was 49152 bits. A downloadable picture has dimensions of 64 x 96 pixels.

The memory controller currently has three modes of operation. In the write mode the controller waits for a data valid bit to take the data on the data_in databus. This data is then brought stored in the current memory location. The address is incremented to wait for new data and another valid data bit. Once the whole picture has been stored, the memory controller returns a done bit to the control unit. The write mode has been optimized from a four-cycle write to a two-cycle write. The optimization came from predicting the next memory address required to write. Handshaking has been reduced to the valid signal and can be accounted for most of the speed up. The number of states used to implement this design was dropped to two allowing it to take up less space and making the overall design faster.

The read mode puts new data on the data lines every three cycles. This was done to simplify data acquisition on the filter side. The original design would take a data request from the filters. The memory controller would return a value to the filters. However, this design was improved since the filters need a new pixel every cycle. Since the filter must run through all the values, the memory controller can be automated to constantly supply data every three cycles. This means that the filters will get data exactly when they are needed. This simplified the algorithm used by the memory controller because there are no data request bits to look for and it can configure itself when it requires to move onto the next address.

A display mode was developed to send the data to the display module. This mode was developed because the display module needed data request handshaking signals, but also allowed for looser timing requirements than the filters. Since the display module will always request the next pixel, therefore it does not need random access of memory, the memory controller can exploit this and literally have the next value the display module will request available before the display module requests it. This simple prediction allows the memory controller to reliably have the correct value on the data lines, no need to worry about set up and hold times.

Display

Display is a VHDL entity specifically designed to utilize the simple digital-to-analog converter to output to the CRT monitor. The module outputs this data to the output signals crt_out as well as generating the horizontal and vertical synchronization pulses.

The timing for the horizontal and vertical synch pulses are based upon the VESA standards for VGA output at 640 x 480 resolution however during testing it was discovered that the constants needed to be adjusted slightly for the characteristics of our board in order to achieve a stable display. With the current values of the constants a stable display has been achieved.

Horizontal refresh	40.5 kHz
Active	640 pixels
Front Porch	43 pixels
Sync	64 pixels
Back Porch	87 pixels
Entire line	824 pixels

Vertical refresh	76 Hz
Active	480 lines
Front Porch	9 lines
Sync	3 lines
Back Porch	30 lines
Entire frame	522 lines

The simulated waveforms of generating the first horizontal and vertical sync pulses are attached.

A simple resistor voltage divider circuit is used to convert the digital outputs of the FPGA to the analog inputs for the CRT. The CRT requires voltages in the range 0V to 0.7V and this must be produced from the 8-bit digital outputs and is then connected to all three colour inputs red, green, and blue to produce a grayscale image.

2.5V Outputs:

Each bit has twice the significance of the preceding bit, so its output voltage should be twice as high. The maximum voltage for a VGA colour signal is 0.7V.

$$0.7V = V_{\text{bit } 0} + V_{\text{bit } 1} + V_{\text{bit } 2} + V_{\text{bit } 3} + V_{\text{bit } 4} + V_{\text{bit } 5} + V_{\text{bit } 6} + V_{\text{bit } 7}$$

$$0.7V = 1X + 2X + 4X + 8X + 16X + 32X + 64X + 128X$$

$$0.7V = 255X$$

$$X = 0.002975V$$

For the most significant bit:

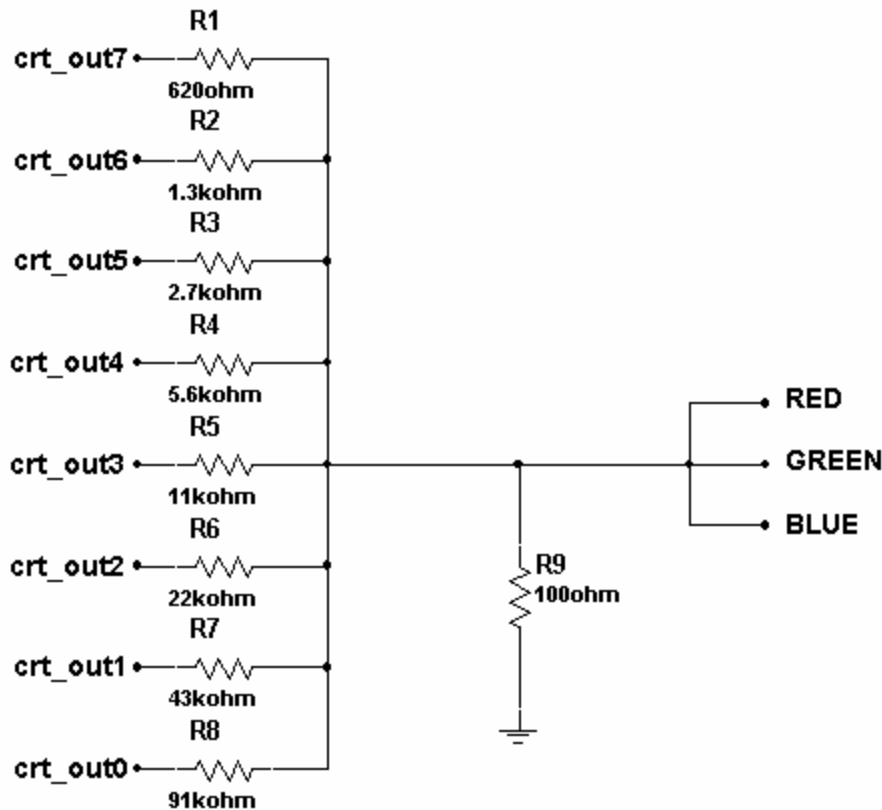
$$R_1 = V_{DD} \times 1000 / V - 1000 = 2.5 \text{ V} \times 1000 / 0.3808V = 6200$$

Similarly for the least significant bit:

$$R_8 = V_{DD} \times 1000 / V - 1000 = 2.5 \text{ V} \times 1000 / 0.002975V = 91kO$$

Bit	Voltage Per Bit	Resistor Value
0	0.002975V	91kO
1	0.00595V	43kO
2	0.0119V	22kO
3	0.0238V	11kO
4	0.0476V	5.6kO
5	0.0952V	2.7kO
6	0.1904V	1.3kO
7	0.3808V	620O

VGA DAC Circuit (for 2.5V digital outputs)



RS 232

RS 232 Interface was developed using previous code from the NN Trainer group which was modified slightly to meet our needs. Currently, the transmit functionality of the code has been removed. Though there is documentation from previous groups for RS232, some of it was found to be irrelevant due to the project board they used was different. Also, much of the documentation was ambiguous and further research was required to implement the interface properly.

The input data pushed through the RS232 input module is set using dividers such that the value of the input bit is read in the middle of it's value to eliminate possible error before being stored in the shift register. The packet recognition module (RS232_In.vhd) begins working once activity is detected on the line and each time a valid bit is detected, the shift register is enabled and takes the new bit. The module ignores the start and stop bits of a packet and does not store them. Using the MTTTY program, the data is actually sent in an 11-bit packet using 2 stop bits rather than the 1 that RS232_in is looking for. This is to give a "buffer" between each packet in case of timing issues as a precaution though there was no know problems using only 1 stop bit. The RS232_in module is "fooled" into thinking there is only 1 stop bit as once it finds the first stop bit, the process will no

longer be active until the next start bit is detected. The module also contains a packet counter to confirm the proper number of bytes have been received as well as a 10 second time out counter which will set a flag if the line has been inactive for too long. Both of these functions will notify the control unit of the current status of the RS232 module and disable it to move to the next state of operation.

Once the image has been modified, it can then be sent back to a PC. The modules perform the reverse operation, taking parallel data from RAM and sending it serially back to the PC. Once enabled, the RS232 output module will handshake with RAM to request data and simultaneously assert the shift register to read the data bus and begin shifting the data out serially at the proper rate in RS232 protocol format. Again, the output modules have packet counters and time out processes which will report on the status of the upload to the control unit. At the time of submission, RS232 output is in the final stages of implementation but is not yet complete.

Clock division for 33.333MHz was calculated to be 109 for input.

$$\begin{aligned} 33,333,000 / 9600 &= 3472.18 \Rightarrow 3472 \\ 3472 / 16 &= 217 * \\ 217/2 &= 108.5 \Rightarrow 109 \end{aligned}$$

* As each bit is read, to ensure that the bit is read at the center of its value, an offset of 8 is used in the code to measure the bit.

As with previous groups, a standard RS232 data packet was used with 1 start bit (low), 1 stop bit (high) and 8 data bits. For 19200 baud it may be more beneficial to add a second stop bit as the outputs data onto the data bus is slightly slow and delaying the read bit to the RAM by 10us gives the output line enough time to hold the proper values

Upon receiving 3 valid packets the module writes the data to the input_byte line and asserts the read bit to notify the memory controller that valid data is on the bus. Several test programs were developed to track number of packets received as well as the integrity of the data sent. These in turn proved the functionality of the RS232 modules was performing as expected

The RS232 output modules function very similar to the inputs. Once enabled, the RS_232_out module will send a handshake signal to the RAM and the output shift register simultaneously. The handshake to the RAM employs the same signal the display unit uses to request pixels (a MUX decides whether RS232 or Display can currently use the signal). The RAM, while in idle, displays the pixel at address 0, then increments to display the next 24 bits of data. Meanwhile, the shift register has also been given the read signal from the RS232 module and will bring in the data currently on the bus. It will then break the 24 bit vector into 3 8 bit vectors and tack a 0 and 1 stop and start bit onto the vector and then begin shifting it serially onto the output pin of the serial header. As with the input module, the RS232 out code contains both a packet counter and a time out process to notify the control unit of the current state of the module. Should either upload success or fail be reached, the control unit disables the module and continues operation.

At the time of report submission, the RS232 upload code is in the final stages of testing but is not yet completely functional.

Clock division for 33.333MHz was calculated to be 1736 for input.

$$33,333,000 / 9600 = 3472.18 \Rightarrow 3472$$

$$3472 / 2 = 1736$$

Convolution

The convolution operation on a pixel neighborhood can produce a wide range of numerical values. It is therefore important to adjust each value to match the range of the display or memory storage device. One method, termed clipping, is utilized to truncate the values to the bounds of the display range. Another method that generally produces better results operates by reducing the scale of values to fit the display range. The latter method is called normalization and is implemented by our design. The bias value for each filter reflects the absolute value of the sum of the negative coefficients convolution matrix. The bias represents the total deviation of the output values from the lower bound of the display range. The normalization value reflects the sum of the absolute values of the elements in the convolution kernel. The bias is added to each output value to ensure that it lies within the lower boundary of the display range, and the normalization value is used to scale down the output values to fit the upper boundary of the display range.

A crucial part of the convolver module is the convolution shift register. The csr register is responsible for sending a 3x3 window of pixel data to the convolution calculation in order for the calculation to proceed. The csr register shifts image data in from the memory one pixel at a time (each clock cycle) until 2 complete rows of pixels plus 3 additional pixels of the image data have entered the register. At this point the register sets a csr_full signal high to indicate the window contains valid image that the convolver calculation can process. The register continues to shift pixel data in until the end of the image is detected using a row and column counters and a fixed image size. At this point the csr_done signal is set high so that no more data is requested from memory and the convolution calculation knows to finish its current calculation. Edge detection is also accomplished using row and column counters and is necessary since the edge pixels cannot be calculated with 3x3 convolution and must be copied in their original form to the memory where the new filtered image is stored.

External HDL Components

The memory controller with a SDRAM controller first developed by ALTERA and later modified by Manticore. However, the SDRAM controller was dropped because it was not able to interface with physical memory. Due to this setback, lpm ram was implemented through the use of the Megafunction wizard.

Previously used HDL components in the user interface code include the lcd_out module necessary to provide the output signals to the lcd pins and the lcd module which contains the control logic to load the character or data commands to lcd_out. These components represent a critical part of the user interface as they are responsible for displaying the current state of the image processor to the user and verifying their input. Due to the time constraints of the project and the importance of being able to display the status of the processor, the decision was made to use previously synthesized and tested lcd components. The CDMA lcd module was chosen for its clean interface and well documented code.

The dual configurable debouncer originally developed by the Reading Book group [20] was used to properly debounce all push-buttons in the user interface.

Both RS232 download / upload modules were also based on previous RS232 efforts by the Neural Network Trainer group [12].

All verification of External HDL and IP Components have been integrated into the design verification and testing sections of the module they belong to.

Experiments and Characterization

User Interface

LCD Display Testing

All LCD menus were displayed using a push button to transition between states to ensure that all 16 characters on each menu / status screen were correct. The LCD screen powered up correctly and all screens were refreshed appropriately on state transitions.

Push Button Debouncing

The correct debouncing operation of the push buttons was verified using a 4-bit counter with a debounced push button to increment the counter. One press of the push button resulted in the clock incrementing once, double presses resulted in two increments, and holding the button down and then releasing resulted in one increment.

User Interface Finite State Machine Testing

The integration of all modules into the user interface was accomplished by introducing one module at a time and using a push button to force the user interface through unused states. First, images were downloaded into the unit via rs232 and lighting LED's on the board when the lpm_ram data matched that of the data being sent in on rs232. Next, the CRT module was incorporated to display the newly acquired image. Filters were then included and tested by processing each filter and displaying filtered image data to the CRT display. Finally, the RS232 upload module was incorporated into the interface and the uploaded image data was sent back to the PC for further analysis.

Memory Controller

The initial design called for the use of SDRAM due to the large storage size. SDRAM would allow for 24 bit pictures to be stored in a flexible size of resolution. The SDRAM controller went through a large number of experiments to verify it was functioning correctly. Initial experiments involved confirming that the simulation was giving the correct waveforms. These were found to have the proper behavior and so experimentation moved into the actual hardware implementation. Increasingly less robust implementations were used to input data through DIP switches and have it read back to LED's. All these experiments kept failing and we narrowed down the possible problems to either a fault in the ram, the FPGA board, or timing requirements within the ram. Since data storage is crucial to the overall design, further testing of the SDRAM was dropped and development of LPM_RAM was initiated.

Development increased rapidly once LPM_RAM was used. Since this type of memory storage uses components within the FPGA, it can be simulated and the output can be viewed directly on a waveform.

The first algorithm developed was write access to the ram. A state machine was built to recognize when a write is requested, set the correct address for the ram, take the data in, and then set the write enable signal to store the data. This approach took too long. Though RS232 is fairly slow, other modules, like the filters, will need a relatively faster

response to be able to write to the ram. The states were gradually reduced to two states. In the first state it waits for a write signal, the next address is converted and sent to the ram that is basically waiting for a write enable, once the signal is set high it grabs the data and puts it on the data lines into the ram. The next step is giving the ram a write enable and preparing the next address. This two-step method has a fast enough turn around time to be able to meet the RS232 and filter requirements, and it has the added benefit of taking up less logic elements. The RS232 interface with the ram was tested by loading a file with a particular set of data that could be searched for. First couple of tests showed the RS232 did not have the correct clock dividers, further tests confirmed the proper function of the interface by turning on an LED once the data was found.

The second access method developed involved the interface with the display module. This was pursued because this test could be used as a visual confirmation that the RS232 was actually storing to the ram. Due to the small size of the picture, the display module expands the picture to fit more of the screen. To realize this implementation, the display module needs to repeat the same pixels as it scans horizontally and an entire row of pixels as it scans vertically. This complexity is divided into the display and the memory controller. The display sends a pixel request when it requires a new pixel, and a row request when it requires the memory to jump to the next row. The memory controller keeps internal counters to make sure the display gets the correct pixel. Due to these internal counters, the memory controller is able to accurately predict the next pixel and have the data available before the request for the next pixel is there. This prediction comes from the fact that the display does not require random access to the memory. The display module will always ask for the next pixel in the column, even on a next row request, it is certain what pixel it will require next. By exploiting this unique relationship, the ram is able to easily keep up with display. Testing the interface this interface did not work on the first run. There was a delay added to the display module to allow the ram to make sure it is starting at the beginning of the picture, and the next row signal was found to be arriving slightly late. To correct the second problem, the if statement that was used to recognize the next row request was taken out of a nested loop. This considerably improved the reliability of the interface.

Lastly, a way of reading the ram was developed for the filters. This interface also has a relationship that can be exploited. The filters use a large shift register. Therefore, every clock cycle a new pixel is required to be there. Since the ram stores three pixels per address, the filters are getting three pixels per memory access. This means that the data the filters use only needs to be there every three cycles. This means that handshaking signals, like those used by the display module, are not required. Once the filters send a signal requiring data, the memory sequentially puts new data every three cycles, linearly moving through the memory until the memory controller has parsed through the entire picture. At this point the display was working which allowed us to test the filters.

Video DAC

The video DAC resistor divider network was thoroughly tested to ascertain its performance and compliance with the VGA specification. Each bit of the signal `crt_out` was turned on individually and the resulting voltage was measured versus the calculated value. The circuit obviously contained certain simplifying assumptions thus certain non-

linearities in the digital to analog conversion were expected, but all variations were found to be in an acceptable range.

Results of Video DAC Experiments

Bit	Calculated Voltage Per Bit	Measured Voltage Per Bit
0	0.002975V	0.008V
1	0.00595V	0.012V
2	0.0119V	0.015V
3	0.0238V	0.019V
4	0.0476V	0.04V
5	0.0952V	0.08V
6	0.1904V	0.14V
7	0.3808V	0.32V

As can be seen from the above table the measured voltages are in general lower than the expected value. This results in a slightly dim image on the screen. This can be compensated for simply by adjusting the monitor's brightness control. Most importantly the voltage never exceeded the maximum 0.7V in any tests that were performed.

More qualitative tests were then performed to check image quality. A test image of a smooth gradient from black to pure white was displayed to confirm the continuity of image brightness. Results were deemed acceptable with no noticeable jumps in image brightness. Finally some random sample images were displayed and all faithfully replicated the version produced on a standard computer setup.

RS232

Development of RS232 proceeded with development of the code from previous projects to meet the needs of our design. As other units in the design all relied heavily on exact timing to coordinate handshake and data between modules, the FIFO implementations in the code was removed as buffering data was deemed somewhat irrelevant and did little to improve our design. The RS232 input module was first developed and simulated using a clock divider of '1' meaning that on each rising edge of the clock, a new bit was being received. After confirmation that the simulations performed correctly, the clock dividers were calculated for the NIOS board. Taking the global clock speed of 33.333MHz, the values were calculated as described in the Design Verification section. The global clock was also verified on an oscilloscope by taking a reading from one of the output pins on the board. This was done mainly to gain a more *exact* number if possible as if the dividers were off considerably due to rounded values, the divider could be off by enough to result in lost data. Next, MTTTY was tested to ensure that it would send the RS232 stream which would match the code written to detect it. Wires were wrapped to the header of the serial cable and observed on an oscilloscope. MTTTY was then configured to send one byte continuously. As expected, a suitable packet of data was observed to be 1.0416 ms long, each bit being 104us. The waveform clearly showed a start and stop bit as well as recognizable data in the format desired with bits of lower value sent first. Once RAM was implemented successfully, the seven segment LEDs were used to check the data in a

packet. The LEDs were mapped directly without a decoder and thus use each segment of the LED to correspond to a 0 or 1. This allowed 2 bytes of data to be checked using the 2 LEDs rather than just one. Data was then send from the PC through MTTTY, stored on the board and read from RAM and visually confirmed on the LEDs.

Output is tested by taking the sent file from the board and reconfiguring the file give it the appropriate headers and comparing it with the filtered image on the board to gather any mistakes by inspection. Currently this portion of the code is not functional as the output data is not in the proper format to be received by the PC due to timing errors. This problem will hopefully be alleviated by the time of the demonstration.

Convolver

convolvercalc.vhd Development

The most processor intensive portion of the entire design is the actual convolver calculations. This resulted in certain challenges of meeting the system timing requirements. The global clock signal on the Nios development board is 33.3MHz and so the performance of that as a minimum was set. The significant design optimizations that were implemented include a binary tree adder instead of 9 simultaneous adds and replacing simple integer division with the LMP_DIVIDE megafunction with 10 stages of pipelining. The performance results are summarized below.

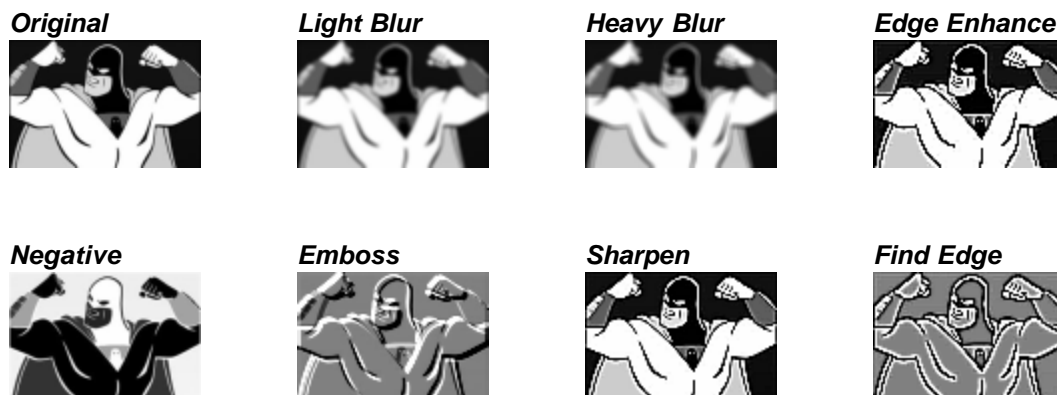
Version 0.1	
Status of Development	-nine 9bit multiplications in parallel -add nine 18bit numbers in one clock cycle -division not implemented yet
Performance	fmax of 12.94MHZ critical path 72.661ns 1691 logic elements used
Version 0.2	
Status of Development	-binary adder tree 4 stages -division not implemented yet
Performance	fmax of 117.03MHZ critical path 7.123ns 1429 logic elements used
Version 0.3	
Status of Development	-added normalization and bias inputs as integer types -added enable logic and division between 20bit integers
Performance	3105 logic elements used fmax of 3.17MHZ critical path 398.79ns
Version 0.4	
Status of Development	-added bias and normalization buffer registers and pipeline register between last addition and normalization division
Performance	2441 logic elements used critical path 124.423ns Clock clk has Internal fmax of 8.69 MHz between source register normalization7[11] and destination register Y_out_long1[0] (period= 115.018 ns)
Version 0.5	
Status of Development	-same as ver0.4 except reduced CONV_COEFF_WIDTH to 4
Performance	1235 logic elements used critical path 76.375ns Clock clk has Internal fmax of 13.72 MHz between source register normalization7[5] and destination register Y_out_long1[0] (period= 72.873 ns)
Version 0.6	
Status of Development	-replaced integer division with lpm_divide megafunction with 10 pipeline stages (lpm_divida.vhd)
Performance	2777 logic elements used critical path 20.542ns Clock clk has Internal fmax of 46.88 MHz between source register lpm_divida:lpm_divida_inst lpm_divide:lpm_divide_component abs_divider:divider alt_u_div:divider DFFDenominator[5][0] and destination register lpm_divida:lpm_divida_inst lpm_divide:lpm_divide_component abs_divider:divider alt_u_div:divider DFFStage[6][2] (period= 21.332 ns)
Version 1.0	
Status of Development	-bug fix for sign extension of incoming convolution coefficients (signext.vhd)
Performance	3,180 logic elements used critical path 18.365ns Clock clk has Internal fmax of 53.55 MHz between source register lpm_divida:lpm_divida_inst lpm_divide:lpm_divide_component abs_divider:divider alt_u_div:divider DFFDenominator[5][3] and destination register lpm_divida:lpm_divida_inst lpm_divide:lpm_divide_component abs_divider:divider alt_u_div:divider DFFStage[6][11] (period= 18.673 ns)

Filter Module

After the user interface had reached a degree of stability where the different filters could be selected and processed, testing on the individual filter began. The results of each filter were displayed on the CRT and then compared to the generic convolution effects packaged with the GNU Image Manipulation Program (GIMP) [21]. GIMP was chosen because it offers a generic convolution filter which allows the user to set the convolution matrix coefficients as well as normalization and bias coefficients.

In all cases the results produced by the IA9K were very similar to those produced by GIMP. The most notable exception was the filtering of the edge pixels around the perimeter of the image. The IA9K replaces the edge pixels with the edge pixels from the original image while the GIMP filter makes a copy of the edge data and extends it outward so that it can be used as the outer edge of the convolution matrix to produce a filtered result for the actual edge pixels. Also, the non-linearity in the D/A conversion scheme used by the display circuit could also be responsible for different shades in the inner pixels even if GIMP and IA9K yield the exact same convolution result. However, for the most part all pixels within the edges matched the results of the GIMP filtering.

Another factor that affects the overall quality of the filter is repetition of pixel data used by the display. Since every pixel is repeated 6x horizontally and vertically it has the effect of making the image appear more pixelated than usual which made it difficult to discern between light and heavy blur as well as edge enhance and sharpen filters which have only slight differences to begin with.



Images generated by GIMP [21]

At the time of this writing the RS232 upload was being tweaked in preparation of the final demonstration so a direct comparison between the GIMP and IA9K digital data was not possible as originally intended. Comparisons of the GIMP data with simulations confirmed that the data output by the convolver corresponded.

References

Memory:

- [1] Altera Corp. 2000. Altera SDR SDRAM Controller White Paper
- [2] *ARS Technica RAM guide*, Jon Stokes, accessed Feb 1, 2002 at URL http://www.arstechnica.com/paedia/r/ram_guide/ram_guide.part1-1.html
- [3] Wrobel, H., Snyder, J. (Motorola) 2001. MPC8xx SDRAM Interface. (Application Note AN2066ID).
- [17] Carson, Benj, et al. Manticore Group SDRAM controller

Pin Assignments:

- [4] Reference: Ng, D., et al; *The Dynamic Picture Frame Final Report*. April 11, 2002.
- [5] Reference: Walton, J. et al; *Where's Waldo Image Finder*. Dec 2001.
- [6] Reference: Smith, J. et al; *CDMA Group's LCD App Notes*. Dec 6, 2000.

Display:

- [7] VDAC external circuitry based on the ADV7120 datasheet Analog Devices. CMOS 80MHz, Triple 8-bit Video DAC ADV7120. <http://www.analog.com/UploadedFiles/Datasheets/173587347adv7120.pdf>
- [8] Ng, Dave et al, DPF Group, Winter 2002, CRT Synchronization and Sample VHDL Implementation Student App Note.
http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/2002_w/interfacing/CRT/CRT_App_Note.html
- [15] Carson, Benj, 8 Bits Per Pixel Student App Note.
http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/2002_w/circuits/8bpp_vga/
- [16] VESA standards; <http://www.vesa.org>

Filters:

- [9] Fully Programmable 3x3 Convolver Design
FPGA Application Engineering, Atmel Corporation. An Introduction to DSP Applications using the AT40K FPGA
<http://www.tecnun.com/asignaturas/tratamiento%20digital/dsp40k.pdf>
- [10] Convolver matrix values
Björklund, Kaj, Jakobsson, Markus, and Rontu, Markku; Digital Image Processing.
<http://dot.tt.hut.fi/dip/>
- [14] Altera, Application Note 82: Highly Optimized 2-D Convolvers in FLEX Devices. Feb. 1997: San Jose, Calif.
- [19] G. Bois, B. Bosi, Y. Savaria, 1997. A High Performance Reconfigurable Coprocessor for Digital Signal Processing :
http://www.grm.polymtl.ca/circus/data/MUG1997_paper.pdf

[21] The GNU Image Manipulation Program (GIMP) can be downloaded free under the GNU licensing agreement at <http://www.gimp.org>

Rs232:

[11] Reference: Carson, B. et al; *The HULK*, Dec, 2002

[12] Reference: Timmy Li, Andrew Ling; *Neural Network Trainer – RS 232 Protocol*

[13] Reference: *How Stuff Works . Com: How Serial Ports Work*

<http://www.howstuffworks.com>

User Interface:

[18] Smith, J., et al; *CDMA VHDL LCD Module*. Dec, 2000.

[20] Bazinet J., et al; *Dual Configurable Debouncer*. Dec. 2000.

VHDL Code Index

1.0 User Interface Code	33
2.0 Display Interface Code.....	34
3.0 Memory Interface Code	35
4.0 RS232 Interface Code	36
5.0 Filter Interface Code	37

1.0 User Interface Code

Entity Name	userinterface.vhd
Description	UserInterface is designed to respond to pushbutton inputs, display the appropriate menus, and enable/disable the proper submodules to accomplish the user request
Status	compiled -no errors

Entity Name	userinterface_pkg.vhd
Description	Holds constants and components for the user interface
Status	compiled -no errors

Entity Name	lcd.vhd
Description	to write messages out to the LCD by using 16-character text messages and control the powerup and refresh of the lcd screen.
Status	compiled -no errors

Entity Name	debounce.vhd
Description	entity used to de-bounce push buttons
Status	compiled -no errors

2.0 Display Interface Code

Entity Name	display.vhd
Description	Generates horizontal and vertical synchronization pulses for 640x480 8-bit grayscale
Status	compiled and simulated - No Known Bugs

Entity Name	display_pkg.vhd
Description	Holds component definition
Status	compiled and simulated - No Known Bugs

3.0 Memory Interface Code

Entity Name	mem_pkg.vhd
Description	Holds constants and components for the memory controller
Status	compiled -no known bugs

Entity Name	mem_ctrl.vhd
Description	Interface circuit between the lpm ram and the rest of the circuit, handles all aspects of access to the ram
Status	compiled -no known bugs

Entity Name	Ram.vhd
Description	LPM ram used to store the picture data
Status	compiled –no known bugs

4.0 RS232 Interface Code

Entity Name	Serial2led.vhd
Description	Top entity file. Port maps and input signals for lesser entities. Uses load_module.vhd as well as testbench add ons. Used code from NN Trainer Group (Modified). Also interfaces with test module to 7 seg LEDs for visual testing
Status	Compiled - No Known Bugs. May need alterations for proper interface with SDRAM.

Entity Name	RS232_In.vhd
Description	Main functional unit to receive data. Examines the input serial data. Confirms start bits and enables shift register. Uses myShiftRight.vhd and clkdiv.vhd. Used code from NN Trainer Group (Modified)
Status	Compiled - No Known Bugs. May need alterations for proper interface with LPM RAM.

Entity Name	clkdiv.vhd
Description	Divides down global clock signal to synchronize to 9600 baud using divisor constants. Used code from DPF Group (Modified hideous bug)
Status	Compiled - No Known Bugs.

Entity Name	myShiftRight.vhd
Description	Shift register which does the actual serial to parallel conversion. Outputs the data back onto the bus and set the data_valid bit which will notify the LPM RAM when it's time to read. Used code from NN Trainer Group (Unmodified)
Status	Compiled - No Known Bugs.

Entity Name	count.vhd
Description	Counter module which counts number of valid packets received. Can be outputted to 7 seg LEDs for visual confirmation.
Status	Compiled - No Known Bugs.

Entity Name	Load_module.vhd
Description	Main RS232 module for recognizing and handling input serial data.
Status	Compiled - No Known Bugs.

Entity Name	RS_232_out_top.vhd
Description	Main RS232 module for output. Maps the output module to clock divider
Status	Compiled – No known bugs

Entity Name	RS_232_out_top_pkg.vhd
Description	Package file for RS_232_out_top
Status	Compiled - No Known Bugs.

Entity Name	RS_232_out.vhd
Description	Main RS232 module for handling data output. Manages the handshake signals from RAM to the shift register
Status	Compiled – Some bugs. There are some issues around the timing of packets being sent onto the output line which are not recognizable by MTTY

Entity Name	clkdiv_out.vhd
Description	Clock divider for output
Status	Compiled - No Known Bugs.

Entity Name	myShiftout
Description	Shift register which outputs contents onto output pin to the PC
Status	Compiled - No Known Bugs.

5.0 Filter Interface Code

Entity Name	filter.vhd
Description	Top level module to perform filtering of image data
Status	compiled - no errors

Entity Name	filter_pkg.vhd
Description	Holds all constants for filter and convolver modules
Status	compiled - no errors

Entity Name	convolver.vhd
Description	Fully Programmable 3x3 Convolution Filter
Status	compiled - no errors

Entity Name	Conv_shift_reg.vhd
Description	Shift register to output a 3x3 window of image data to the convolution calc
Status	compiled - no errors

Entity Name	convolvercalc.vhd
Description	Performs the actual convolution calc
Status	compiled - no errors

Entity Name	lpm_divida.vhd
Description	Normalizes the output produced by the calculation (reduce a 22bit number back to an 8-bit value)
Status	compiled - no errors

Entity Name	signext.vhd
Description	Used by convcalc to sign extend an 18-bit two's complement number to 22-bits
Status	compiled - no errors

Entity Name	negative.vhd
Description	8-bit negative filter and memory interface module
Status	compiled - no errors