University of Toronto
Department of Electrical and Computer Engineering

ECE 532 – Digital Hardware
Final Project Group Report

Character Recognition using Neural Networks

Sari Onaissi 994556725
Nahi Abdul-Ghani 994553336
Hratch Mangassarian 994941391

March 31, 2008

Table of Contents

1. Introduction and Overview

Character recognition is the translation of images of handwritten or printed text into editable text. This is still an open problem and is the subject of active research due to the increasing demand for accurate and fast recognition methods. Such methods are and will be the basis of input devices for a wide variety of systems including handheld devices and computers. The goal of our project was the FPGA-based implementation of a character recognition module using neural networks. Neural networks are highly parallelizable; thus, a hardware implementation of such a network would result in a fast method. In our case, we are trying to recognize a subset of the characters in the English alphabet.

Artificial Neural Networks

An artificial neural network is a computational model based on biological neural networks. It consists of an interconnected group of artificial *neurons* and is often used to classify and find patterns in possibly noisy data. It is an adaptive system that changes its structure based on external information that flows through the network during a *training phase*.

In this project we used a simple neural network architecture, called the *perceptron* model. Figure 1 shows a perceptron neural network consisting of a single layer of S neurons. Each of these neurons is used to recognize a different character in the English alphabet. The neural network shown in Figure 1 contains R inputs. In this project, each of these inputs corresponds to an input pixel. Each neuron of Figure 1 generates a weighted sum of its inputs and passes the result through a hard-limit function known as an activation function, effectively implementing the following equation:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{w}$ is a vector of weights and $\mathbf{w}^T \cdot \mathbf{x}$ is the dot product (which computes a weighted sum). b is called the *bias*, a constant term that does not depend on any input value.

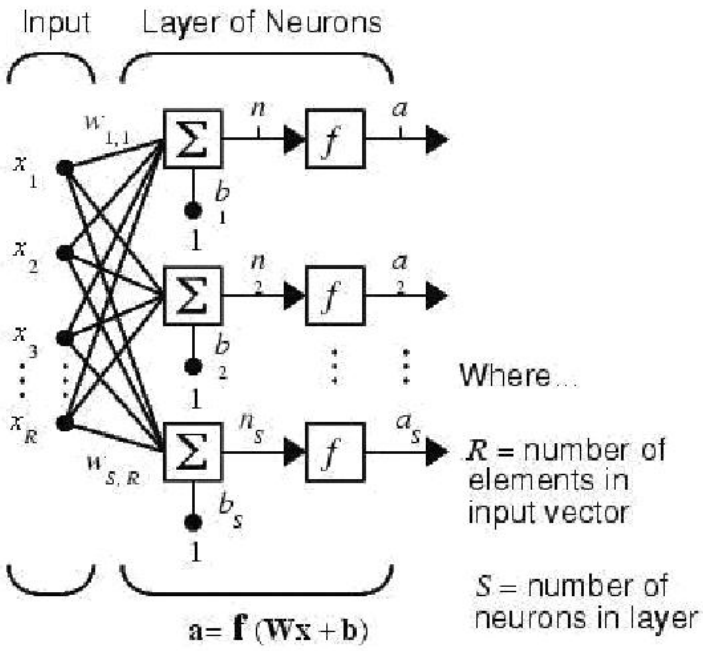


Figure 1. An artificial neural network

In order to obtain the weights **W** and the biases **b**, the neural network must go through a training phase, during which inputs are applied to the network and the computed outputs are compared to the expected outputs. As a result, the weights and the biases are adapted accordingly in an effort to conform with the expected behavior given by the training set. For each output $f_i(\mathbf{x})$, the corresponding input weight vector and the bias are updated as follows:

$$w_{i,j(new)} = w_{i,j(old)} + (target_i(\mathbf{x}) - f_i(\mathbf{x})) \cdot x_j$$

$$b_{i(new)} = b_{i(old)} + (target_i(\mathbf{x}) - f_i(\mathbf{x}))$$

where $target_i(\mathbf{x})$ denotes the expected/correct value of the output of $f_i(\mathbf{x})$ given input **x**.

Goals and Specs

The initial goals as laid out in our project proposals were the following. The plan was to use the *perceptron* neural network to recognize a subset of the characters in the English alphabet. We planned on starting by trying to recognize seven letters and possibly increasing the number of letters depending on our initial recognition results. Our system would take in an image of a character, which would be mapped to a 10x10 superpixel frame. This 10x10 (100 bit) input would then be given to the neural network, which would attempt to classify it as one of the

possible seven letters. Therefore, the perceptron neural network would have 100 inputs, 7 outputs and 700 connections.

The eventual implemented specs of the project are the following.We used a perceptron neural network to recognize the first *ten* letters of the English alphabet their capital forms, based on our recognition results. The inputs are provided as black-and-white, 288x352 resolution images that are mapped into 9x11 (99 bit) images to be given to the neural network, which attempts to classify them as one of the possible ten letters. Therefore, the perceptron neural network has 99 inputs, 10 outputs and 990 connections.

System Block Diagram

The block diagram in Figure 2 provides an overview of the major components of our system and their interconnections. A training set of seven images for each of the ten characters are initially stored in the BRAMs. Along with their corresponding expected target letters, they are sent to the training unit of the neural network by the MicroBlaze. The training unit tries to recognize the characters in the training set and adjusts the neural network weights and biases by comparing its output to the expected targets.

A 288x352 resolution image of a character can now be input by the user through the UART, which the MicroBlaze first sends to the reduction unit through the FSLs. The reduction unit reduces the resolution of the image to 9x11. The MicroBlaze then sends this reduced image to the testing unit of the neural network through FSLs. The testing unit will try to classify this image into one of the ten first characters of the English alphabet.

Reused IPs: MicroBlaze, UART, BRAM, FSL.

Custom IPs: Neural network (training and testing unit), Reduction unit

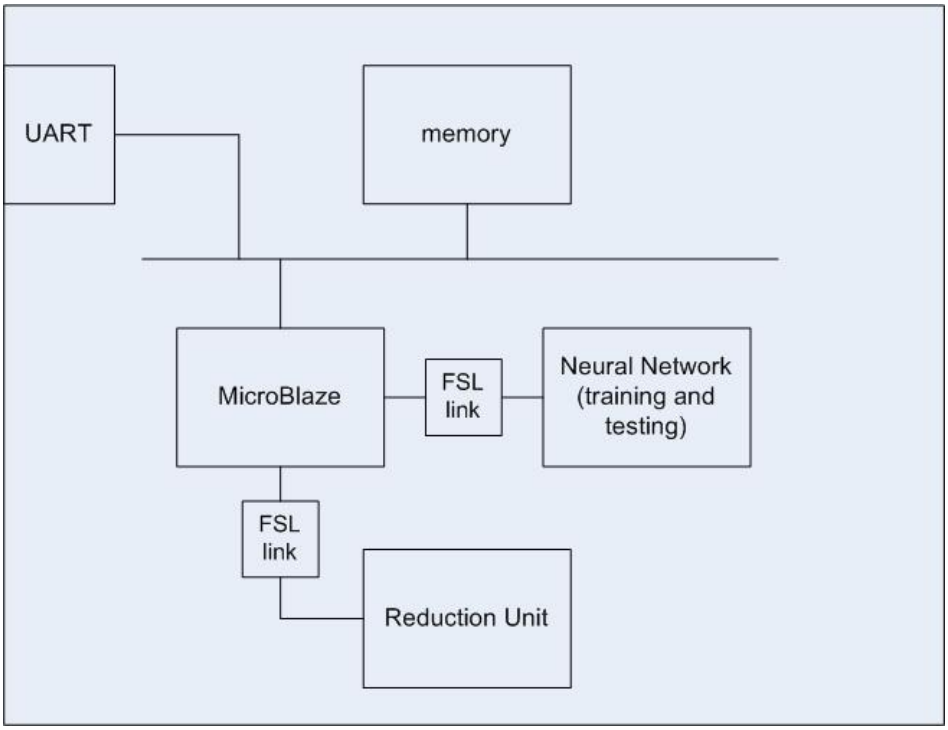


Figure 2. System block diagram

System Flow

Figure 3 describes the necessary steps for the system to recognize a noisy 'A' character.[1] Each 288x352 image of 'A' in its training set of seven images is reduced into a 9x11 image in software (using Matlab) and then stored in memory. These images are then passed to the training unit of the neural network in order to adjust the weights **W** and biases **b** used for character recognition. Once the weights and biases are set, the neural network testing unit can be used to recognize characters. In the testing phase in Figure 3, a noisy 288x352 image of 'A' is passed through the hardware reduction unit, which yields a 9x11 image of 'A'. This image is input to the testing unit of the neural network, which will hopefully classify it as an 'A' by setting the output of the neuron corresponding to 'A' to 1, and all other neuron outputs to 0.

[1] The described flow is done for all ten considered characters.

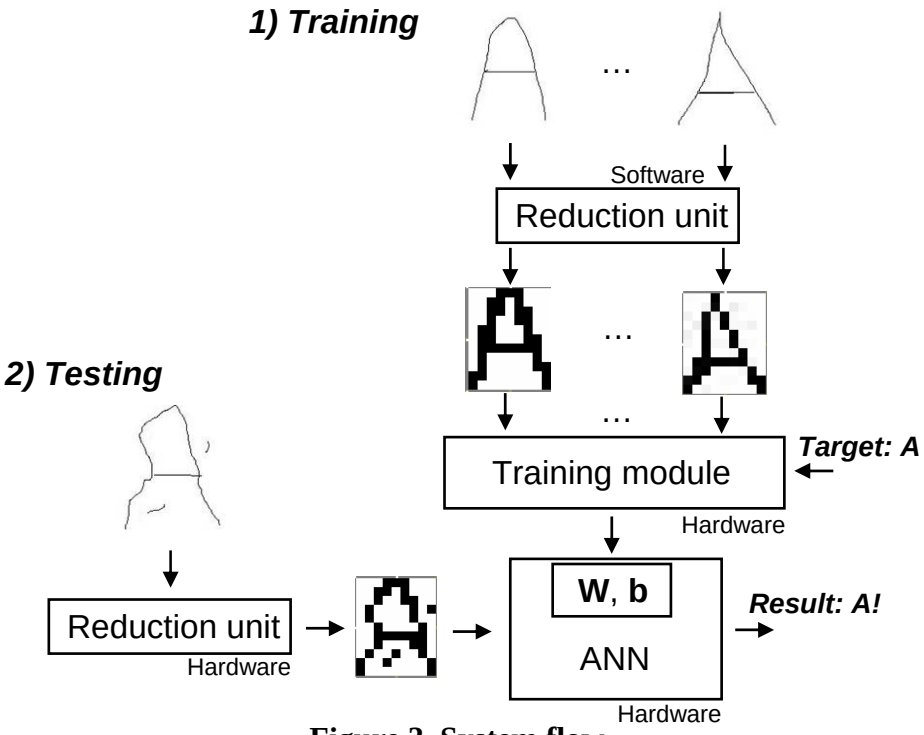


Figure 3. System flow

Matlab Simulations

Before implementing our system in hardware, we implemented the new modules using Matlab. The motivation for doing so was threefold:

- To make sure that single-layer perceptron networks are good enough for character recognition. Also, to decide on the number of iterations (epochs) for the training set (1,000).
- To decide how many characters to consider (ten) and a reasonable input resolution of the neural network (11x9 pixels) and the reduction unit (288x352).
- To have a golden model to compare our hardware implementation to.

Each of the training unit, testing unit and reduction unit were implemented using Matlab and were tweaked and tuned to produce reasonably good results before tackling the hardware implementation. The Matlab files are included in our design tree files described in the last section of this report.

2. Outcome

Results

Neural Network

	Used	Available	Percent
Slices	13969	13696	101%
Slice Flip Flops	9834	27392	35%
4 input LUTs	21863	27392	79%

Reduction Unit

	Used	Available	Percent
Slices	421	13696	3%
Slice Flip Flops	744	27392	2%
4 input LUTs	621	27392	2%

The outcome of our character recognition project obviously depends on the intricateness of the input images that are fed to it. Therefore coming up with some sort of percentage of correct classification is a subjective endeavor. Instead, we believe that it would be of greater interest to show a few instances of recognized and non-recognized characters in order to give a sense of the limits of our project to the reader. Keeping in mind that character recognition is by no means an easy task and is actually still an area of active research, our results were better than expected. We were occasionally surprised by the recognition of very noisy character images, which initially raised our expectations to a point where we were getting disappointed when other character images were not recognized correctly.

The characters shown in Figure 4 are the training sets for the first ten letters of the English alphabet. As can be seen, we tried to diversify the training set as much as possible in an effort to recognize a large number of character variations.

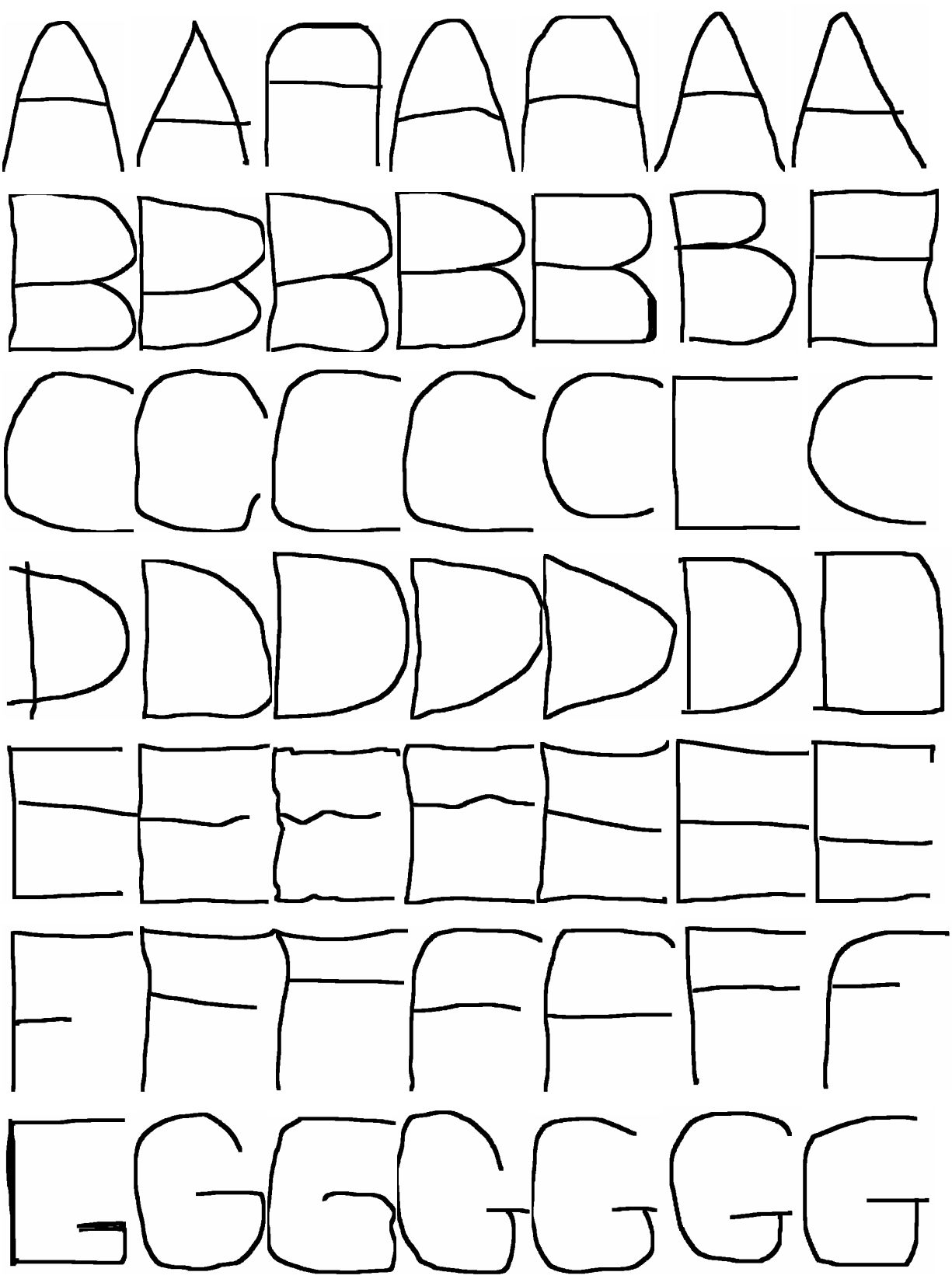


Figure 4. Training set

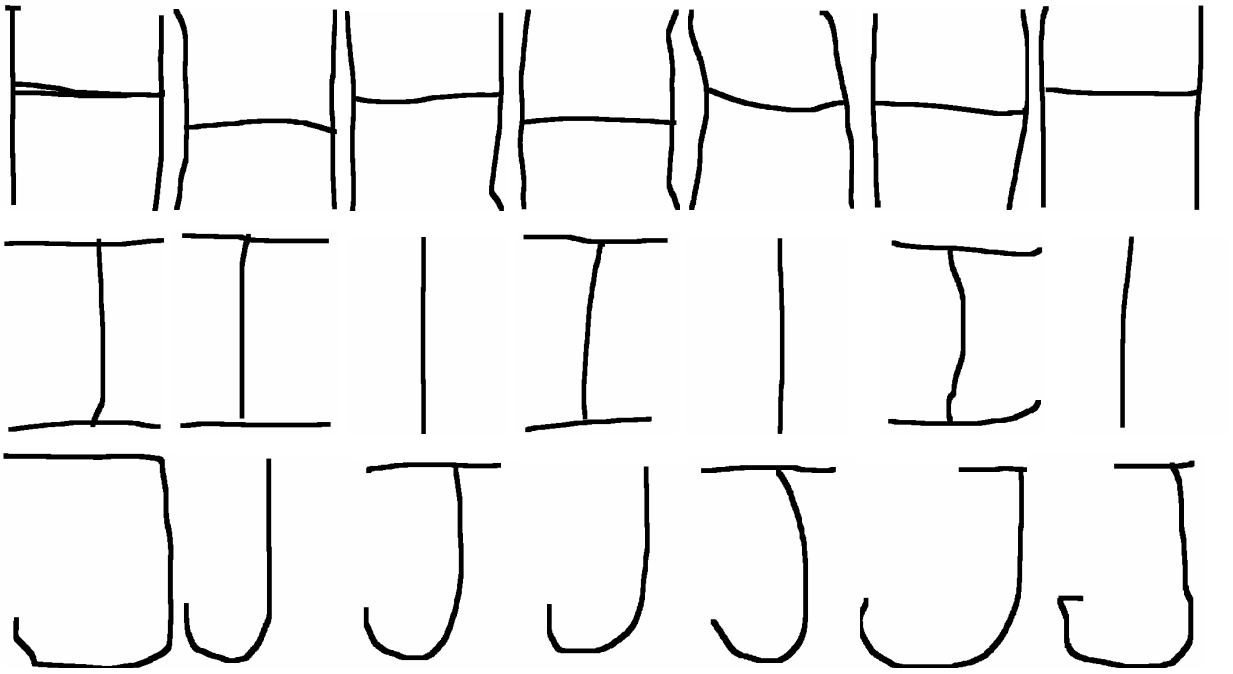
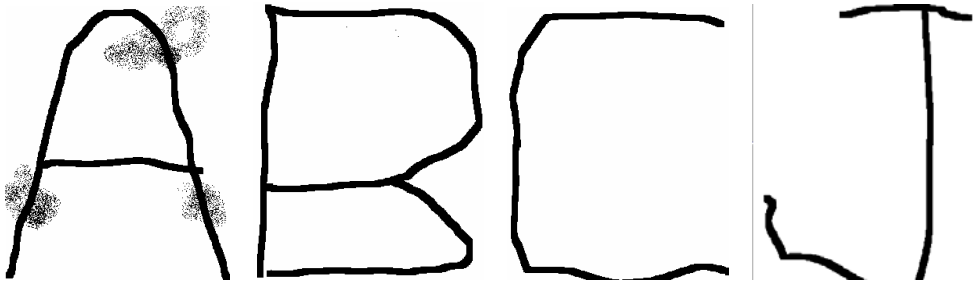
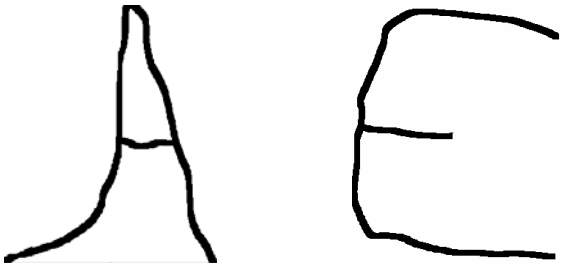


Figure 4. Training set (continuation)

The following are four examples of recognized characters. They are taken from the list of training and testing character images provided in the project design tree zip file sent to the professor.



The following are two examples of unrecognized characters along with the corresponding results given by our character recognition module. Again, they are taken from the list of training and testing character images provided in the project design tree zip file given to the professor.



The first ‘A’ was not detected, while the ‘E’ was detected incorrectly as a ‘C’.

Suggestions for future work

An important assumption of our character recognition module is that in the input character image, the letter is more-or-less bounded by or touches the image borders. In other words, it is should not be drawn too small in the given space. This caused some problems during the project demo and made the accuracy of our character recognition module seem less than what it really is. There are a number of ways to deal with this issue, the simplest and probably most effective of which is to crop the picture, either in software or preferably in hardware such that the character touches the image bounding box. If the aspect ratio turns out to be different than that of the training set, the image could be stretched in either direction. Such a module would just be at a pre-processing stage before the resulting cropped image is given to our existing character recognition module.

Other obvious areas of possible improvements are the accuracy of our module and the number of handled characters. The training set can be expanded and tuned more thoroughly to try to achieve better recognition accuracy. However, one should be careful not to over-train the neural network because that could lead to a degradation of recognition accuracy. As the accuracy improves, the neural network will be able to handle more characters in an acceptable manner. A possible way to increase the number of handled characters is to use more complex neural network models with multiple layers. Training such neural networks might require a significantly larger character training set. Another possible extension is the handling of lowercase letters.

An extension suggested by Alex Kaganov was the handling of variable image sizes. This would fall into the functionality of a cropping and resizing/stretching module mentioned in the first paragraph of this section, where a larger or smaller image is appropriately stretched to the required aspect ratio and size.

3. Project Blocks

MicroBlaze

The Microblaze is the main controller of the overall system. It runs the C code which coordinates the flow of data between various units.

FSL

This IP can be found in XPS. The Fast Simplex Link Data sheet, again found in XPS, was consulted to determine the required signals during read and write operations. There are 2 FSL links between the MicroBlaze and each of the neural network and reduction units.

The neural network module takes a 99-bit input image with some additional control signals to function properly. Therefore, the FSL state machine has to accept four 32-bit words from the FSL FIFO. With the arrival of the fourth word, a go signal is set high for the neural network to do the required operation while the FSL state machine is in wait mode. When the neural network results are ready, the FSL state machine writes a 32-bit result on the FSL FIFO. Only the ten least significant bits of the word are used to indicate which character was detected. The reduction unit takes a nine 32x32 blocks in parallel and reduces them into nine pixels. Therefore, the FSL state machine for the reduction unit accepts 288 32-bit words in groups of nine words. The details of the FSLs are described in the appropriate individual reports.

UART

An interface module with the outside world.

Neural network testing unit

The neural network testing unit is the hardware implementation of the artificial neural network shown in Figure 1. This module takes in a 9x11 reduced image, as a 99-bit input vector, where a black pixel is represented by a 0 and a white pixel by a 1. The weights and biases are already set at this point, and each neuron must be able to add all the weighted inputs along with a bias. Finally, the output of the neuron must be 1 if this sum is positive and 0 if the sum is negative. In other terms, the output of neuron j, y_j, must satisfy the following equation:

$$y_j = \neg MSB(b_j + \sum_{i=1}^{99} w_{j,i} x_i)$$

where MSB denotes the most significant bit and it must be inverted in order for y_j to be 1 if its argument is positive and 0 if it is negative.[2] y_j=1 means that the j^{th} character is recognized.

Figure 5 shows each neuron of the testing unit, which calculates the above equation in hardware. Note that there are ten of these modules, one for each neuron and used to recognize a different character. The weights $w_{j,1}$ to $w_{j,99}$ along with the bias b_j are stored in a weight bank register with 100 entries of 8-bits each. Note that the weights and the bias can be negative.

Next, each weight $w_{j,i}$ must be multiplied with the corresponding input x_i. We avoided building full-fledged multiplication units and instead used multiplexers as shown in Figure 5, using the fact that the inputs x_i can only be either 0 or 1. If x_i=1, the weight $w_{j,i}$ is selected. If x_i=0, an 8-bit zero vector is passed. The bias b_j is always passed.

Next, we need to sum all these passed signals (i.e, $w_{j,i}$'s for which x_i=1 along with the b_j). One way to do so was to use a tree of adders and do this summation combinationally. We opted for a more space-efficient sequential summation, using an 8-bit 100-to-1 multiplexer and a counter at its select lines to count from 0 to 99, thus selecting one of these lines at a time. An accumulator[3] is put at the output in order to keep adding, or "accumulating", the selected input values to the current running sum. By the time all the inputs to the 100-to-1 multiplexer are added, the output of the accumulator will be equal to

$$b_j + \sum_{i=1}^{99} w_{j,i} x_i$$

and it suffices to look at the MSB of this number and invert it to get y_j.

[2] This is true because we used 2's complement arithmetic.

[3] An accumulator consists of an adder followed by a register, whose output is connected to one of the inputs of the adder.

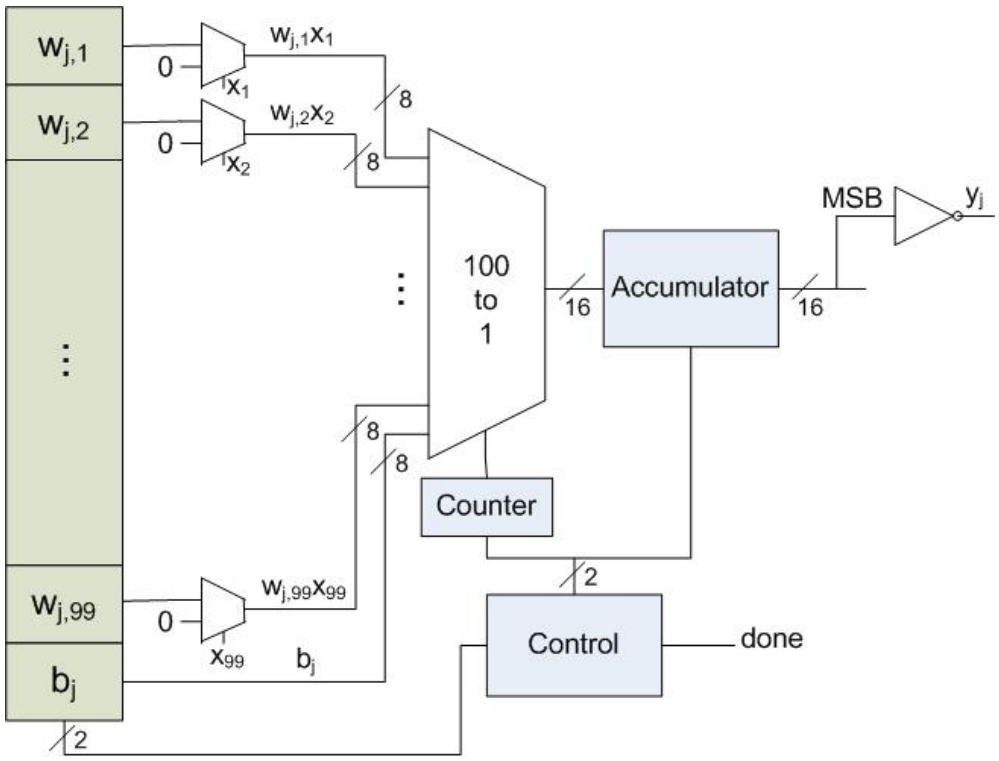


Figure 5. Testing unit for y_j

Neural network training unit

The neural network training unit is used to set up the weights **W** and biases **b** of the neural network according to the given training set. The updating equations for the weights and biases are the following:

$$w_{j,i(new)} = w_{j,i(old)} + (target_j(\mathbf{x}) - y_j) \cdot x_i$$

$$b_{j(new)} = b_{j(old)} + (target_j(\mathbf{x}) - y_j)$$

where $target_j(\mathbf{x})$ and y_j respectively denote the expected and actual output of neuron j given the input **x**. If the produced output differs from the target, the corresponding weights and bias must be increased or decreased accordingly.

The diagram in Figure 6 shows the hardware to do this for a single weight $w_{j,i}$. The first adder produces $-y_j$ by inverting the produced output of the testing unit y_j and adding 1 to it (negation using 2's complement). The following adder adds $target_j(\mathbf{x})$ to $-y_j$. The MSB of this number is then extended to create an 8-bit number equal to $target_j(\mathbf{x}) - y_j$.

Next, in order to multiply the result with the Boolean bit x_i, an 8-bit 2-to-1 multiplexer is used, which passes $target_j(\mathbf{x}) - y_j$ if x_i=1, and 8-bit zero vector if x_i=0. Finally, this value is added to the old value in $w_{j,i}$ using another 8-bit adder.

Note that 99x10=990 of these modules are needed, one for each weight $w_{j,i}$. A similar module is needed for each of the 10 biases. It is described in the appropriate individual report.

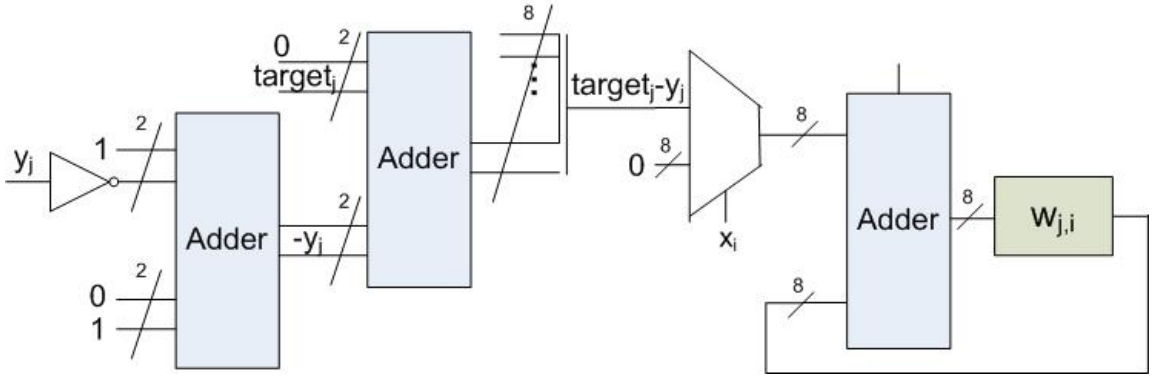


Figure 6. Training unit for $w_{j,i}$

Reduction unit

The goal of the reduction unit is to reduce the resolution of the 288x352 image into a 9x11 image by transforming each 32x32 pixels into a single "superpixel". We decided to color a superpixel white if at least 99% (i.e. 1014) of its corresponding original pixels are white.

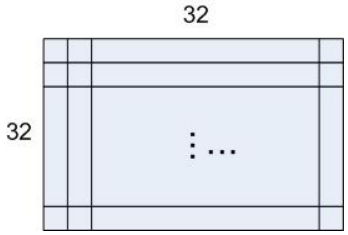


Figure 7. 32x32 pixels to be transformed to a single superpixel

In order to do this, we used a 32-bit shift register, which stored each row of the 32x32 pixels at a time. The output of the shift register is connected to an accumulator. The control circuitry loads the shift register with the next row of the 32x32 pixels until all 32x32 pixels are input to the accumulator, after which the control resets. At this point, the output of the accumulator is compared to 1014 using a comparator, which is basically subtracts one from the other and checks the MSB. The reduction unit shown in Figure 8 is instantiated 9 times. And the control circuitry uses every one of these units 11 times. This way all 9x11 superpixels are produced.

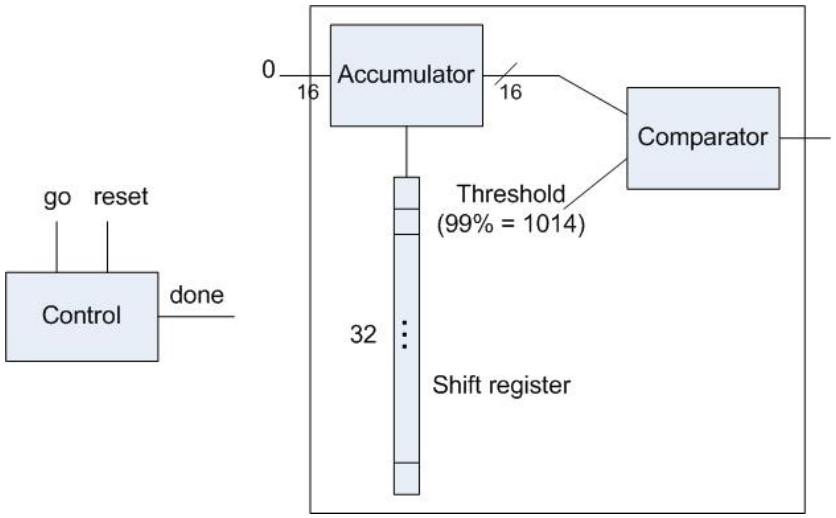


Figure 8. Reduction unit

C code

The C code is the main controller of the overall system including the MicroBlaze, the UART, the neural network and the reduction unit.

The first function that gets called is the training function. This function sends 70 images of size 9x11 to the neural network to train itself. With each image, a flag is sent to indicate the character. The data for each image is packed in four 32-bit words. After training, the program waits in an open loop for a 288x352 image to be sent via the UART. The C code receives the data serially one byte at a time. Once it receives 288 32-bit words it sends them to the reduction unit and waits for the 9-bit result. This is repeated 11 times until all the 99-bits are available from the reduction unit. These bits are later combined into 4 32-bit words using simple bitwise operations and sent to the neural network to be detected. The result of the neural network is read back and processed to know which character was detected and the result is sent to the UART.

4. Project Design Tree

The zip file characterRecognition.zip containing our design directory was given to the professor. It includes the following components:

- Our Matlab implementation of the three main character recognition modules, namely the training, testing and image reduction units.

- The Verilog code of each of these three units.
- ModelSim simulation .do files for all three units as well as some of the sub-modules used in these three units.
- The C code used to download the images to the FPGA.
- A list of training and testing character images.

The directory contains the following README file which shows the main subdirectories, folders and files that might be of interest to the user.

```
ISE_version
*************************************************************
* Contains the ISE projects for the reduction unit and the  *
* neural network                                            *
*                                                           *
* Subdirectories                                            *
*           ann_char_recog: the neural network folder       *
*                                                           *
*                 Subsubdirectory                           *
*                       do_files: all required files for    *
*                                 the unit simulation       *
*                                                           *
*           reduction unit: the reduction unit folder       *
*                           containing verilog files        *
*                 Subsubdirectory                           *
*                       do_files: all required files for    *
*                                 the unit simulation       *
*************************************************************
```

```
XPS_version
*****************************************************************************
* Contains the complete XPS project which has the reduction unit and the    *
* neural network as to IP blocks                                            *
*                                                                           *
* Main Subdirectories                                                       *
*           pcores: the cores for the project                               *
*                                                                           *
*                 Subsubdirectories                                         *
*                       xil_interface_v1_00_a: the folder for the artifical *
*                                              neural network the operates  *
*                                              on the 11x9 image            *
*                                                                           *
*                       xil_reduction_unit_v1_00_a: the folder for the      *
*                                              reduction unit that reduces  *
*                                              that reduces the image from  *
*                                              352x288 pixels to 11x9 pixels*
*                                                                           *
*           code: contains the C code (lab1.c) that performs the initial    *
*                 training for the                                          *
*                 neural network, then waits for an image to be sent via    *
*                 the UART and then                                         *
*****************************************************************************
matlab
*****************************************************************************
* Contains the matlab m-files for generating the ''C-code'' for each        *
* character to be trained, and for generating the a compatible binary form  *
* of the image to be sent over the UART                                     *
*                                                                           *
*                                                                           *
* Subdirectories                                                            *
*           training: has all the images used in the training.              *
*                     They exist in their 352x288                           *
*                     size as well as in their 11x9 size                    *
*                                                                           *
*                                                                           *
*                 user m-files:                                             *
*                       reduce_image.m: reduces image 'xx.bmp' from the     *
*                        original size to the 11x9 size. The result is      *
*                        stored in 'small.xx.bmp'.                          *
*                                                                           *
*                       convert_image_C.m: takes the reduced image and      *
*                        generates its 'equivalent' C_code used             *
*                        for training.                                      *
*                                                                           *
*                                                                           *
*          testing: transforms the 352x288 image into a binary form         *
*                   compatible with our system                              *
*                                                                           *
*                 user m-files:                                             *
*                       big_convert_image_binary.m: takes an 352x288 image  *
*                        'xx.bmp' and generates 'xx.bmp.code.binary'        *
*                                                                           *
*****************************************************************************
```