Artificial neural networking

Phase 2 Proposal

Submission Date: 10/07/2016 Wednesday 11:30am Lab (Lab Section 2)
TA: Mingxuan He, Annan Ma

Prepared by:

Ryan McBee
Taylor Lipson
Cheyenne Martinez
Jack Cottom

1. Executive Summary

Modern computers are strong computational devices capable of solving a mathematical and logical challenges previously not capable by humans. Computers are capable of being given a task, a set of instructions for how to do the task, and can do these instructions a few million times a second, far faster than humans can. This advantage might make it appear that computers could easily replace humans, but what computers gain in speed, they lack in their ability to learn and recognize patterns. This is where neural networking comes into play.

Neural networking is a concept by which a computer or device models it's computation based on how the human brain works. This means having a large system of intermediate nodes in between inputs and outputs, which carry varying weights, and these weights are capable of changing based on learning when an output is incorrect and adjusting for it. This learning step is typically very computationally heavy, requiring millions of iterations in some instances to learn the weights for a single input. This is why we want to use an fpga to create neural networks. The fpga can be optimized to do the calculations faster and more in parallel than a typical computer. To test our implementation of a neural network, we will apply it towards reading computer generated shapes and having it figure out which shape it is.

Successful design of the proposed accelerator will require the following resources:
- A list of digitized hand written characters represented as n x n matrix
- Reference Standard Cell Simulation Library for Mapped Design Verification
- Reference Standard Cell Technology Library for Final Design Layout Verification
- Verilog HDL Simulation and Design Synthesis Tool Chain

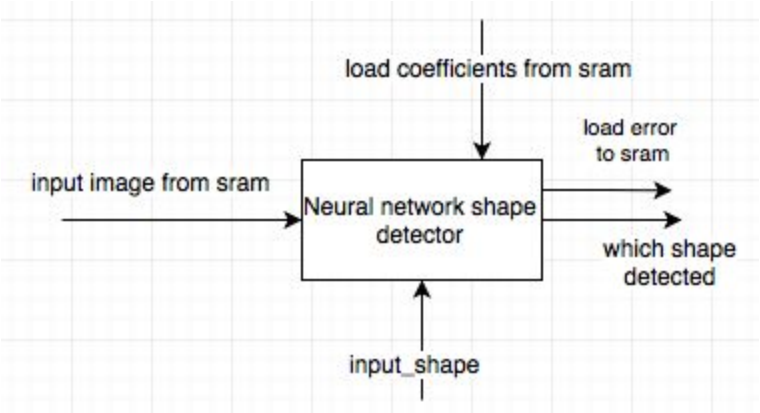
The following document contents will describe:
- Intended usage expectations and constraints for the design, in brief. (Section 1)
- Intended main implementation architecture. (Section 0)

1. Design Specifications

1.1 System Usage

1.1.1 System Usage Diagram

Figure 1: System Usage Diagram



The system described above is an artificial neural network that takes in an image to be analyzed. The system will take in whether it is learning or not, and whether it is supervised as not as it's different modes. It will then take in a preprocessed image and process the image to detect whether or not the image is found.

1.1.2 Implemented standards and algorithms

- Artificial Neural Network weight adjustment
    - Back propagation error detection
        - Creates a random neural network
        - A given known output is provided
        - The system then input an image to be processed, the error of the output is calculated, compared to the known output, and an error is formed.
        - This error is then propagated back into the networks to recalculate the weights.
- SRAM communication
    - Reading information from SRAM as input for image

1.2 Design Pinout

Table 1: Miscellaneous Pinout Table

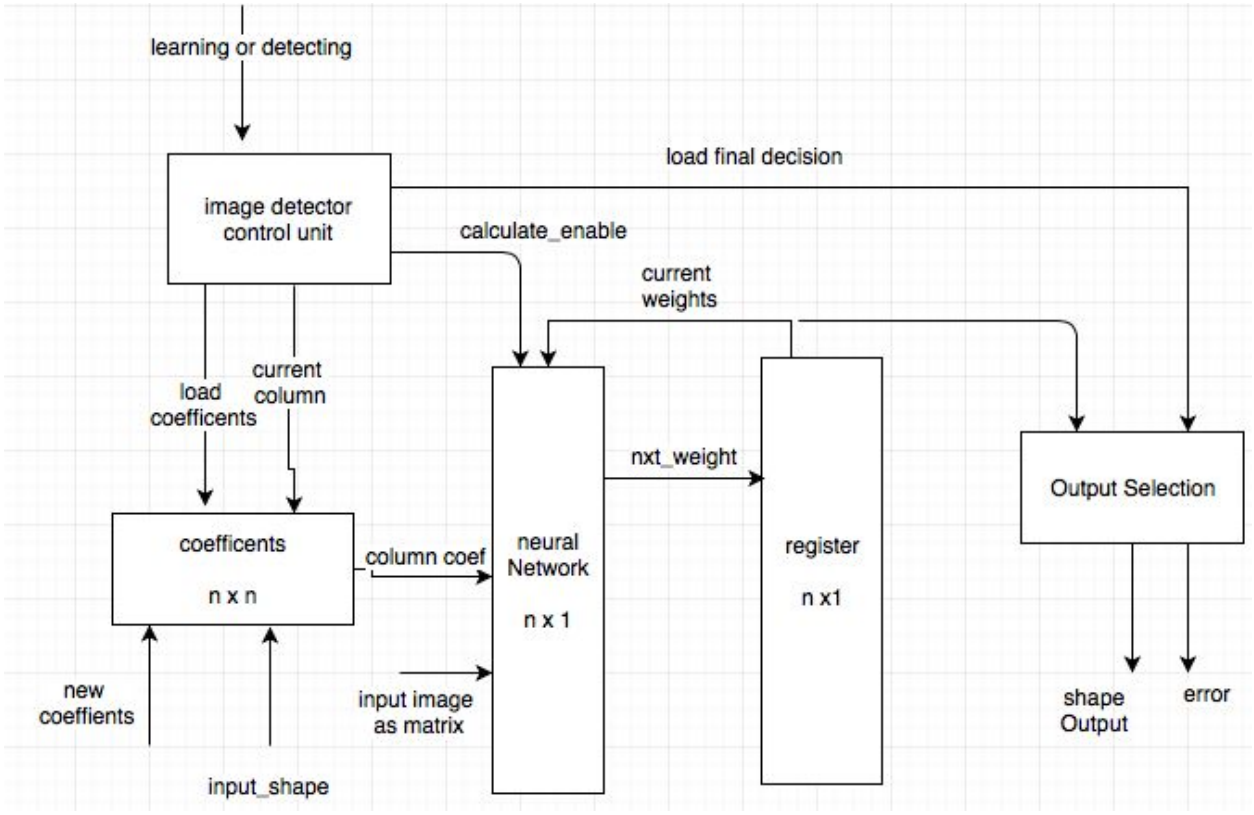
Signal Name	Direction (IN/OUT/Bidir)	Number of Bits	Description
vcc	PWR		Power Pin
gnd	GND		Ground Pin
clk	IN	1	System Clock (speed?)
n_rst	IN	1	Asynchronous Reset. (Active Low)

Table 2: Interface Pinout Table

Signal Name	Direction (IN/OUT/Bidir)	Number of Bits	Description
input_shape	IN	8	The shape that is being inputted if learning.
input_image	IN	8x8	Asserted when input is guaranteed to be a letter. (Active High)
load_coefficients	IN	8x8	Asserted when the weights of the aNN are to be adjusted based on the input data. (Active High)
error	OUT	1	Asserted high when the system is done processing the image.
shape_output	OUT	8	The shape detected by the system represented as an ascii.

3. Design Architecture

Figure 2: Artificial Neural Network, Letter Recognizer Diagram



The implementation for this architecture is depicted above. The input image is loaded from SRAM as an 8 x 8 matrix of 1's and 0's where 1 is the ink and 0 is the whitespace. The coefficients for the ANN are loaded in from outside the design to save a lot of time and space. The shape corresponding to the coefficients is also sent so the coefficients are stored in the correct location.

The image detector control unit is the FSM of the design. Signals are sent to the other blocks to tell what output is to be outputted, what shape is being tested, which coefficients to load, and if the design is currently loading coefficients or detecting a shape.

The coefficients are a series of n x n matrices where each cell contains a weight correlating to how often that cell is a 1 when matching a specific shape. Using these weights/coefficients, a weighted sum is calculated to determine how well an image matches. This can be compared to the total positive sum which is a perfect match.

Calculations matching the shape are performed in the Neural Network. The coefficients are multiplied with the corresponding bit and a sum of all the products is performed. The neural network divides this sum by the ideal sum of the shape it is currently checking and if the ratio is large enough, is counted as a matched shape.

The Output Selection block is a simple series of registers and that outputs the previous shape detected until the current image being processed has been determined to be a recognized pattern of a shape or not a pattern at all.

3.3 Operational Characteristics

3.3.1 High level overview

Our artificial neural network design has two distinct phases. The first phase involves loading in the needed information from memory. This data will include the shapes, stored at n x n byte matrix, names of the shapes, as a byte, and the n x n coefficients, from 0 - 1.0 corresponding to each shape. Our design will then load in these different values, and process them through our neural network circuit. The neural network will be implementing a cluster of n x n nodes. The input to the nodes are predetermined coefficients and the input is the image. The result of the calculations will then be fed into a combinational block that will calculate how close the value is to the desired image.

3.3.2 Interface overview

3.3.2.1 Loading Shaped

We are using a custom protocol to store the shapes we are analyzing. We are storing the shaped as an 8 x 8 byte matrix, where each bit can vary from 0-256. These images will be read from sram and will be integrated into the fpga using the Avalon Memory Mapped Interface. The coefficients used by the neurons and any other identifiers for the shape will also be loaded from memory into the neural network using the Avalon-MM interface.

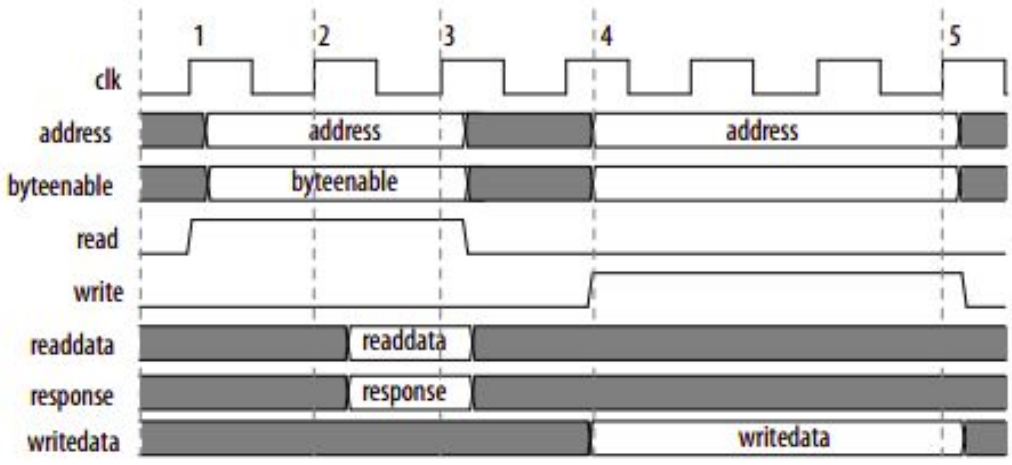
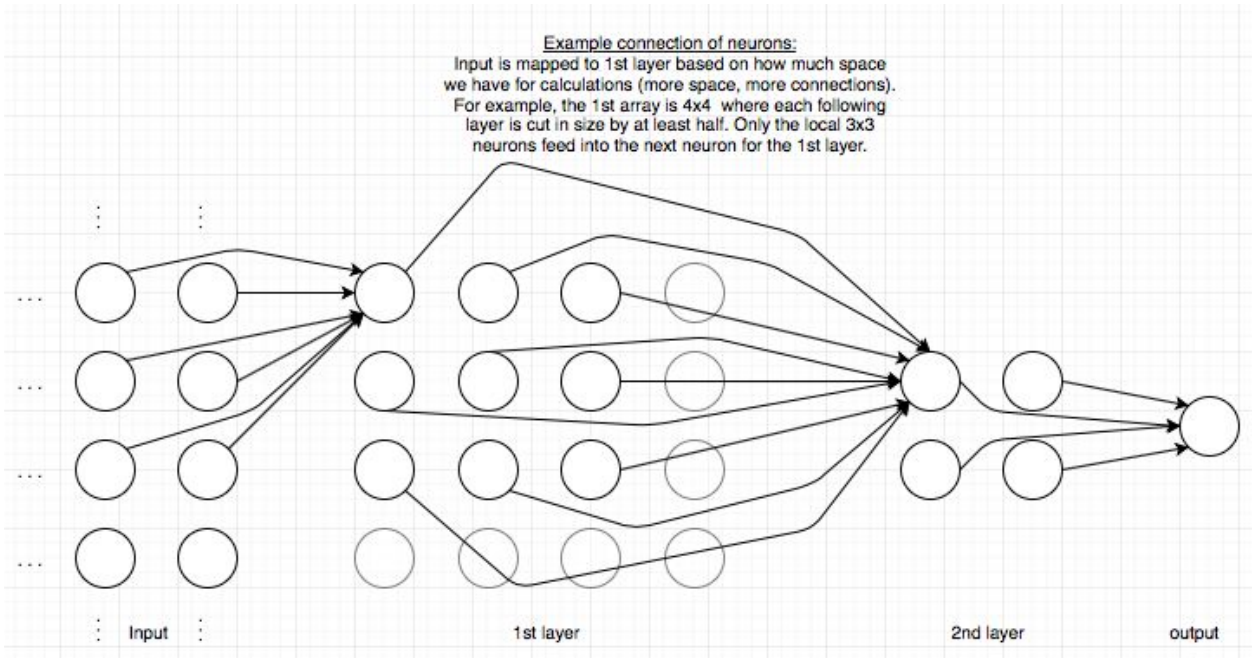


Figure 1: Read and Write Transfer with Fixed Wait-States at the slave interface. (Taken from Avalon interfacing manual)

3.3.2.2 Neural network implementation

The artificial neural network will be implemented by inputting an n x n byte matrix into multiple layers of neurons. Each neuron has connections that has a weight/coefficient corresponding to it. Each possible shape will have coefficients corresponding to the shape stored in SRAM. When the FSM signals that it is checking a specific shape, those values are loaded into the registers for calculations.

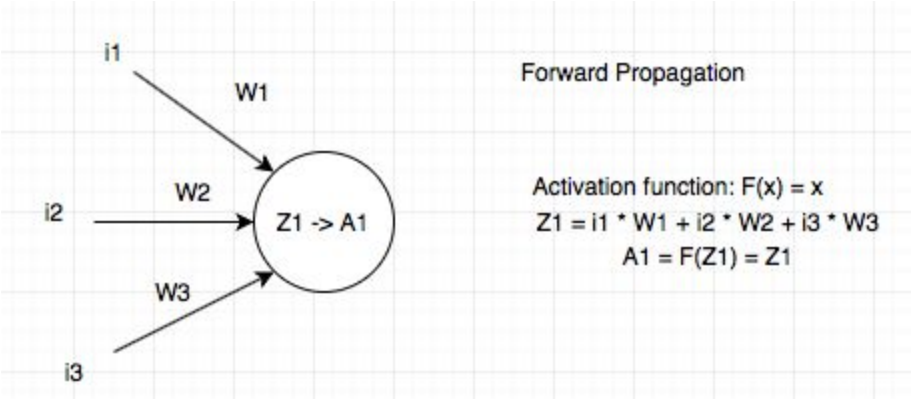
The first layer starts with as many neurons as there are pixels and each subsequent layer gets smaller and smaller until the final layer with a single neuron. In order to avoid any sequential logic, the number of neural connections to each neuron will possibly be limited depending on how much space we have for coefficients, multipliers, and adders. For an example possibility, the input is mapped to a first layer of 8x8 neurons where each neuron can see the local 3x3 pixels from the input. The second layer is a 4x4 layer of neurons where each neuron is mapped to a local 3x3 group of neurons from the first layer. Following is a 2x2 layer of neurons where once again each neuron is mapped to a local 3x3 group of neurons. Finally, the last layer is the output which sees all neurons of the previous layer. Before we can determine how many neural connections are actually possible, the required space needs to be determined.



To determine if an input is a match or not, forward propagation and backward propagation are used. Backward propagation is minimizing the error found in the output by changing the values of the weights of each neural connection. Due to constraints, initially this will just be calculated using python and the weights will be loaded in. A stretch goal would be to implement the

backward propagation within our ASIC design. On the other hand, forward propagation is what is used to determine our final ratio to see if the output matches.

The forward propagation algorithm involves multiplying, summing, and an activation function at each neuron. The output of the neuron is equivalent to the sum of the weights multiplied by their inputs all inputted into the activation function. To simplify the problem, we chose our activation function to be f(x) = x such that the output is just the sum of the products of the weights. In the example diagram below, Z1 is the sum of the product of the weights and their inputs, F(x) is the activation function, and A1 is the output of the neuron.



This process is repeated until the final node outputs a value is is divided by the ideal value for the specific shape, giving us a ratio between 0 and 1 where we can set a threshold to match the shape.

3.4 Requirements

For the neural network application to work to recognize shapes and determine errors we have to be aware of the connections required to complete such tasks. With this implementation we will focus to optimize for space as wells as time because having space for each neuron in the network will expand quickly because of the exhaustive connections to the next level. In order to tackle this issue, we will be implementing registers to break down the delay of sequential calculations, however, the computational time will increase so this will limit the size of images we will be able to process. Our initial goal is to process a 2x2 image and eventually expand our design to be able to process a 8x8 image in order to detect the shape depicted in the image. This of course will change the amount of space needed as the design progresses. Next, the clock frequency we are aiming for is 300MHz because our design will use sequential logic as a way to decrease processing time and combinational logic in multiplying the weights and finding the percent error. For the weights, we are looking at a 8-bit adder and effectively 3,584 combinational blocks. The percent error will add an additional combinational block.