Artificial Neural Network

David Kosa, Polly Jane Bates, Gerard Griest
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
dkosa@oakland.edu, pjbates@oakland.edu, gtgriest@oakland.edu

***Abstract*—This project presents the development of a handwritten digit recognition system using an Artificial Neural Network (ANN) implemented on an FPGA. The primary goal is to utilize FPGA technology to efficiently execute neural network options on hardware. The ANN comprises an input layer for 14 by 14 grayscale images, a hidden layer of 16 neurons, and an output layer of 10 neurons. Implementing the network in VHDL, we employed parameterized design and I/O text files. Finite State Machines (FSMs) serve control and timing purposes for iterating through neurons. Training data, including weights and biases, were pre-computed using MATLAB which was then used in the VHDL design. The ANN performed accurately over a variety of tests able to distinguish all 10 digits.**

I. Introduction

Neural networks are well-known in the area of data science and machine learning. Often implemented in high-level software, neural networks are especially good at determining patterns that are not necessarily apparent nor expected to human programmers. Neural networks come in many forms and are found in many applications from vehicles to smartphones. Neural networks consist of neurons. A neuron has an inherent quality of "activation". Neurons are located in layers categorized as input, hidden, and output layers. Each neuron sends its activation to every neuron in the next layer. The activation from a neuron is multiplied by a unique weight dependent on which neuron is sending and which neuron is receiving the activation. The receiving neuron of all these activation-weight products sums the inputs along with a bias. The bias is unique to every neuron in the network. A neuron's activation is decided by the sum of the weight-activation products and bias after going through an activation function. There are many options for activation functions such as the sigmoid, ReLU, hyperbolic tangent, etc. ReLU is a common choice due to its linear nature and simple computation (Figure 1). The result of the entire neural network is communicated by the activations of the output layer.

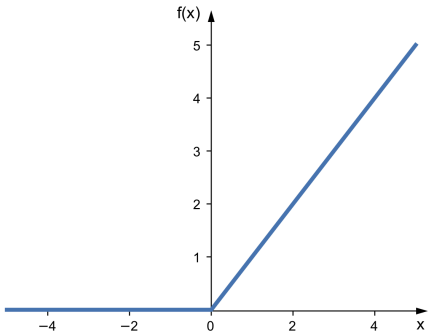

Figure 1: ReLU Activation Function Graph

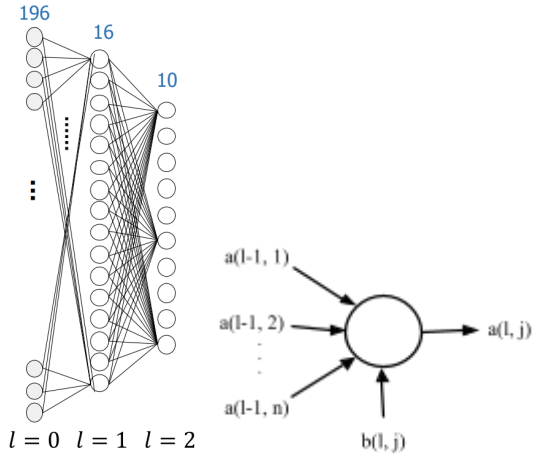

Figure 2: Neural Network and Neuron Illustration [1]

Neural networks must be trained with large data sets. During training an input will produce a likely incorrect output. A loss function will evaluate how different the actual output is from the desired output. Additionally, the weights and biases are adjusted to improve the accuracy of the output. This process of small adjustments is repeated for data in the set until the network is sufficiently trained. Once trained, the network will be able to correctly produce an output even if it has never encountered that input before.

Here we aim to implement an artificial neural network with hardware using FPGA technology rather than a software approach.

II. Methodology

This Artificial Neural Network (ANN) performs the classic operation of identifying handwritten numbers. The input layer accepts a 14 by 14 grayscale image of a single-digit handwritten number. The neural network consists of one hidden layer of 16 neurons and one output layer of 10 neurons. The ANN takes in no external inputs. The input activations are received as constants during synthesis via a text file. The ANN outputs a done signal and the digit it guesses in binary.

The design is broken down into the top file, which contains the layers, a control circuit, a comparator, and a display circuit. Within the layers are neurons. These components are elaborated on in this section.

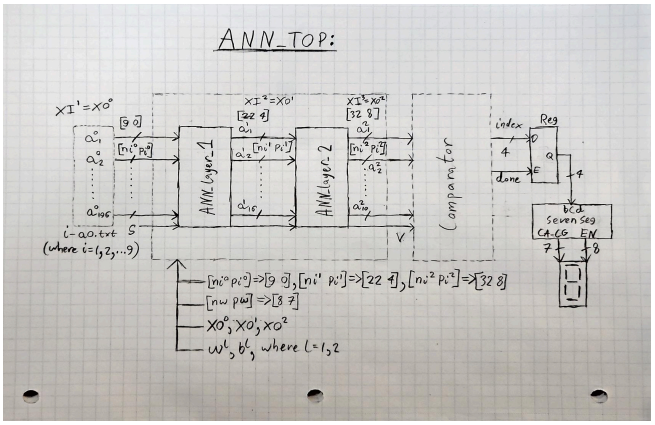

Figure 3: ANN Top Block Diagram

Implementation of the ANN in VHDL greatly benefits from parameterizing the code for the neurons and layers. Neurons are implemented using for generates and require numerous weight vectors which is aided when designed with parameterization in mind.

The program operates iteratively; based on the research of Dr. Llamocca, a pipelined approach will require more resources than is available in the Nexys-A7-100T Board to implement. The project's main purpose is to present the operation of the ANN on the Nexys-A7-100T. We chose to implement the iterative/multiply-and-accumulate design [1] because of limited space on the board and low throughput is acceptable for this project. The pipelined design may prove more useful given a high load of images to be processed which is not necessary for this project.

A. Neurons

The neurons are divided into two types, the hidden layer neuron and the output layer neuron. Both types take in the activation values of the neurons from the previous layer, a neuron selector, weights, and a bias. A bias select signal and accumulator register enable are also inputs to the neuron. Neuron selector, bias select, and accumulator register enable are controlled by the FSM in the layer design. All activations, weights, and biases are in signed fixed-point format. All weights and biases are in the format [8 7]. The activations of neurons in subsequent layers use a larger, more precise format. Activations vary in size with the inputs as [9 0], layer 1 activations as [22 4], and outputs as [32 8]. These sizes are based on a model described in [1]. The hidden layer neurons are slightly different from the output neurons in terms of the number of activations accepted from the previous layer.

Inside the neuron, the weights for the proper layer are multiplexed as input to a multiplier. The activations from the previous layer's neurons are multiplexed with the FSM outside the individual neuron design. A combinational multiplier was chosen for the design. An adder recursively adds the weight-activation products and the neuron bias. ReLU is the chosen activation function for all neurons. A general block diagram for the neuron is shown in Figure 4. The weights and biases are already obtained through training in MATLAB thanks to Dr. Llamocca and are read as constants using a text file for synthesis.

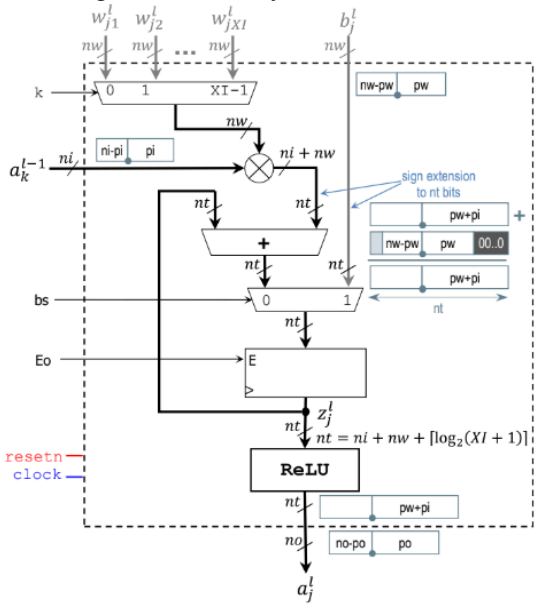

Figure 4: General Individual Neuron Block Diagram

B. Layer

One layer of the neural network consists of the neurons, an FSM, registers to capture input activations, and a multiplexor to choose activations. Layer 1, otherwise known as the hidden layer, consists of 16 neurons, while layer 2, or the output layer, consists of 10 neurons. Layer 1 accepts 196 input values corresponding to the values of the 14x14 grayscale image. The bias vector and weight matrix are also fed to the layer. The 16 neurons in layer 1 receive their corresponding biases and weights from the bias vector and weights matrix. A multiplexor chooses one activation to feed to all neurons. The layer outputs the activations from the neurons and a "v" signal to indicate the completion of the layer computation.

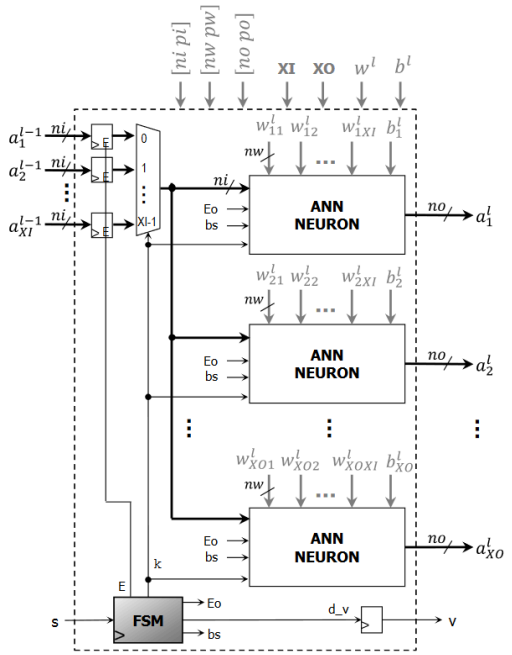

Figure 5: General Layer Block Diagram

C. FSM

As with most timing-based VHDL projects an FSM is required to initialize the order and operations of the code as a whole. This project is no different owing to an FSM to count and keep track of the neuron processes. While this FSM is simple in design (Figure 6) it has an important role in enabling the initial registers that move the inputs into the neurons as well as initializing the neurons to do their processes.

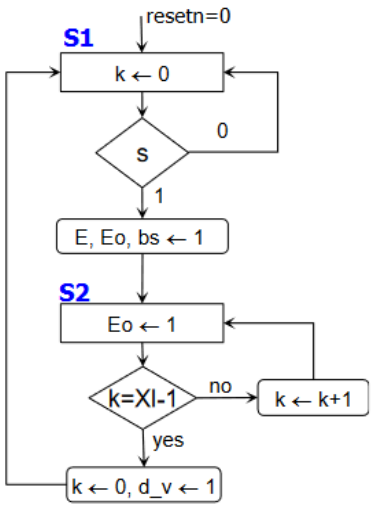

Figure 6: Block Diagram for the FSM

In S1, the circuit waits for the start signal to go high. Upon the assertion of start, the activations from the previous layer are captured onto the registers within the current layer. The accumulator registers are enabled for all neurons and the bias is selected to add to the running sum. The circuit then enters S2. During S2 the accumulator registers are enabled and the neuron selector, k, is incremented iterating through all the activations received. In the last iteration, the FSM asserts d_v to be high which is fed to a flip-flop within the layer. The v signal, the output of the flip-flop, indicates the completion of the layer computation.

D. Comparator

The output neuron with the highest activation value determines which handwritten digit, the neural network "thinks" the input image depicts. The comparator circuit distinguishes which neuron has the highest activation. The comparator circuit multiplexes through each of the output activations to compare with the current highest activation encountered held in the max register. The comparator block will output the greater of the two inputs and a Found Bigger Value flag. FBV is raised when the input from the MUX is greater than the input from the max register. When FBV is raised, the index register captures the index of the current neuron. The circuit outputs a done signal when the greatest neuron's index has been determined. The block diagram is depicted in Figure 7 and the FSM is depicted in Figure 8.

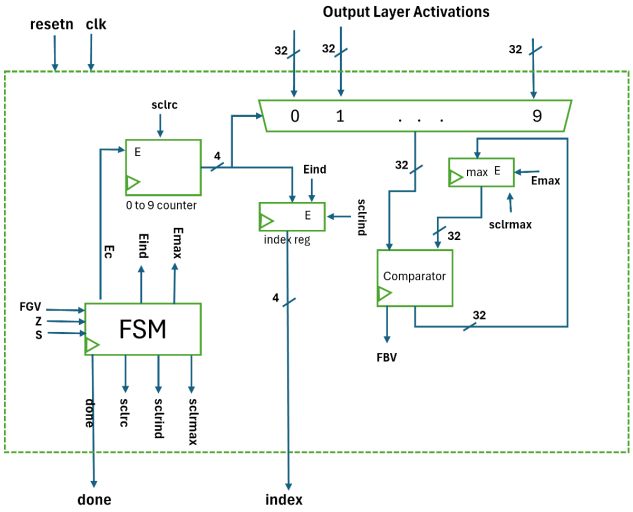

Figure 7: Output Activations Comparator Block Diagram

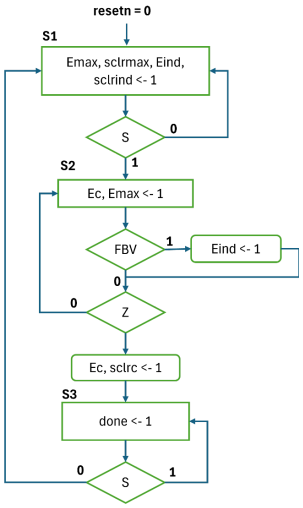

Figure 8: Comparator FSM

The FSM for the comparator begins in S1 with clearing registers and waiting for a start signal. Once started, the circuit enters S2 where the counter and max registers are always enabled. If a larger value is determined from the output neurons than is currently stored in the max register, the index register updates to reflect that determined neuron. Once all 10 neurons have been evaluated, the FSM transitions to S3 where it sends the done signal and awaits start reset.

E. Serializer and Seven Segment Display

For a visual indication of the ANN's function the index signal from the comparator is fed to a block that transforms a BCD number to signals that control a seven-segment display. This file was provided via Dr. Llamocca's website. Only one seven-segment display is required to output the ANN's guess.

F. Inputting Text Files

Incorporating the images (activation inputs), weights, and biases via synthesis was done using an impure function in Vivado. This function essentially reads each line of a text file and then stores it as a constant value within the FPGA. The activation inputs were synthesized in the ANN top file, while the weights and biases for each layer were synthesized in their corresponding layer file. If different activation inputs, weights, or biases needed to be modified, this could be done easily by renaming the parameter of the impure function wherever it is called in the ANN top file or layer file.

III. Experimental Setup

Before the ANN was implemented using Vivado, the ANN was tested utilizing MATLAB. This was done to gain an understanding of the ANN and what was to be built in hardware. As mentioned in the report, the ANN was trained prior to implementation "using a downsampled version (14x14 images) of the 60,000 element MNIST database" [1]. The ANN test in MATLAB resulted in an accuracy of 92.87%.

Once the neural network was built in Vivado, the neural network was then simulated in Vivado using a test bench. The test bench initiates the control circuit of the top file. It is not necessary to populate the input values in the test bench as they are set during synthesis. Behavioral simulations were conducted for ten input files to test each possible digit.

IV. Results

The operation of the comparator is described in the waveform shown in Figure 9.

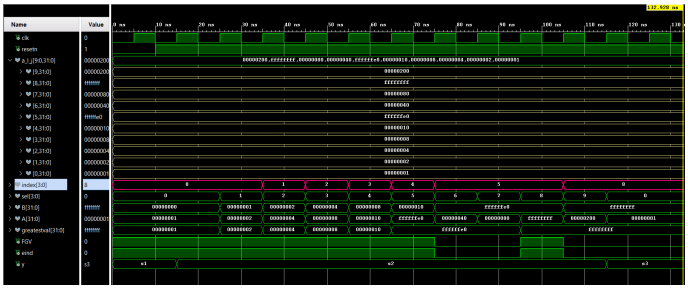

Figure 9: Comparator Behavioral Simulation

The simulation results show correct identification of the index as well as correct updating of the registers and counter signals.

The operation of the ANN is shown via behavioral simulation in Figure 10.

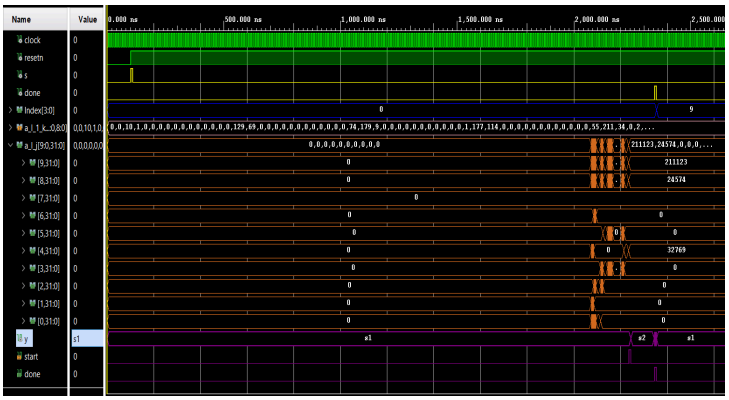

Figure 10: ANN top File Behavioral Simulation.

The ANN simulation in Figure 10 received a 14x14 grayscale image of a handwritten 9; the amount of time the ANN took to calculate the guess can be found by taking the difference between the 'done' signal and 's' signal (yellow signal). The total time between the 's' signal assertion and

the done signal assertion took approximately 2.23 microseconds. Layer 1 takes 196 clock cycles to iterate through all the inputs, layer 2 takes 16 clock cycles, and the comparator takes 10 clock cycles. This sums to 2.23 microseconds overall given a 100MHz clock and an extra cycle for initiation. The comparator sent the index value 9 (blue signal) to the index capture register, which means the ANN correctly guessed the input image to be a 9. The a_l_j value in Figure 10 (brown signal) represents the output of Layer 2, where the largest value is the ANN's guess of what the input image is. The a_l_1_k value (pink signal) represents the 14x14 image that was input to the ANN.

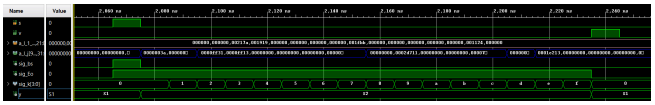

Figure 11: Layer 2 Behavioral Simulation

The behavior of a layer is shown in Figure 11. The pink signal records the inputs to Layer 2 consisting of 16 input activations. In the first state, the biases are added to the running sums held in the accumulator registers. In the second state, the neuron selector iterates through all 16 input activations to be added to the running sums. With each iteration of the neuron selector, the activations for neurons in Layer 2 are updated which is why the blue signal is changing throughout. Once all values are summed, the v signal is issued, and the layer returns to state 1.

All ten test files corresponding to each of the possible digits produced accurate outputs in behavioral simulation. However, when comparing the output activations of the second layer in simulation to the expected activations calculated in MATLAB, the nonzero values were at times about ±50 off from the MATLAB obtained values. This was the case for all test files. The minutia of inaccurate activation values ultimately did not impact the ability to correctly determine the depicted digit for all test files 0 through 9. Perhaps the inaccuracy is due to the fixed point math in our neural network. The complexity of design increases with every bit of precision though. If the discrepancy is not due to the difference in number systems used in MATLAB and the number systems in our FPGA-based design, it may be a slight error in the generated text files for synthesis. Ultimately, we do not know the source of the inaccuracies.

When flashed onto the board the neural network only worked for test files 1, 9, and 8 consistently. We are unsure as to why these files work and why the others do not. The other files would either produce the incorrect value or flicker an incorrect result momentarily before resetting to 0 on the display. Possible reasons for the malfunction were considered. The first considered reason was the bouncing of the button used to initiate the ANN and too long of a start pulse. Despite implementing a debounce and pulse detector block, the ANN still did not work. Second, the combinational delay was guessed to be too long for a 100MHz clock. Attempts to conduct a functional timing simulation were made but the results ultimately were never fully investigated. The low accuracy of the ANN when implemented on the board is concerning and the cause is still to be determined.

V. Conclusions

In conclusion, this project successfully demonstrates the implementation of a handwritten digit recognition system using FPGA-based Artificial Neural Networks (ANNs). The parameterization of the VHDL design and integration of ReLU activation functions allowed for efficient resource utilization within FPGA constraints. Our Finite State Machines (FSMs) managed control and timing, ensuring accurate computations, while pre-computed activations, weights, and biases from MATLAB facilitated seamless integration into the FPGA design. If this project were to be expanded further the need for a way to intake images and have it automatically compute the fixed point values above would be the first problem to resolve. This would allow for a seamless integration and general implementation of any image of a digit we could feed it to give us an output. In a future revision, we might also try to use a bigger board as the current 100T board is not big enough to house a pipelined neutral network. This would allow for a more general application of image recognition and a more efficient neural network. In the end, this project as a whole is a strong representation of neural networks in the VHDL and FPGA space and hopefully will inspire further experimentation for future projects.

VI. References

[1] D. Llamocca, "Fixed-point implementations for feed-forward artificial neural networks", *Integration*, vol. 92, pp. 1-14, September 2023, ISSN 0167-9260, doi: 10.1016/j.vlsi.2023.04.002. [Online].

[2] Llamoca, D. (2013). VHDL coding for fpgas. Retrieved April 18, 2024, from https://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html