Artificial neural networking

Phase 3 Proposal

Submission Date: 11/11/2016 Wednesday 11:30am Lab (Section 2)

TA: Mingxuan He, Annan Ma

Prepared by:

Ryan McBee : Ryan McBee

Taylor Lipson : [illegible]

Cheyenne Martinez : [illegible]

Jack Cottom : [illegible]

1. Executive Summary

Modern computers are strong computational devices capable of solving 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 model 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.

These weights are created by passing known inputs into the neural network, observing the results, and then adjusting the weights to produce an output closer to the desired output. This process is repeated a few thousand times with different input data to produce a flexible network that can recognize a variety of shapes. Since this process only happens once and then the final coefficients are known, putting learning onto an fpga does not seem advantageous.

The goal for our project is to create the framework of the artificial neural network, and have the ANN read in the predetermined learned coefficients. This type of implementation provides a faster and more efficient method of representing and processing the neural network as compared to using a purely software implementation

Successful design of the proposed accelerator will require the following resources:
- A list of digitized handwritten images represented as 8 x 8 matrices
- 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:

2. Design Specifications

2.1 System Usage

2.1.1 System Usage Diagram

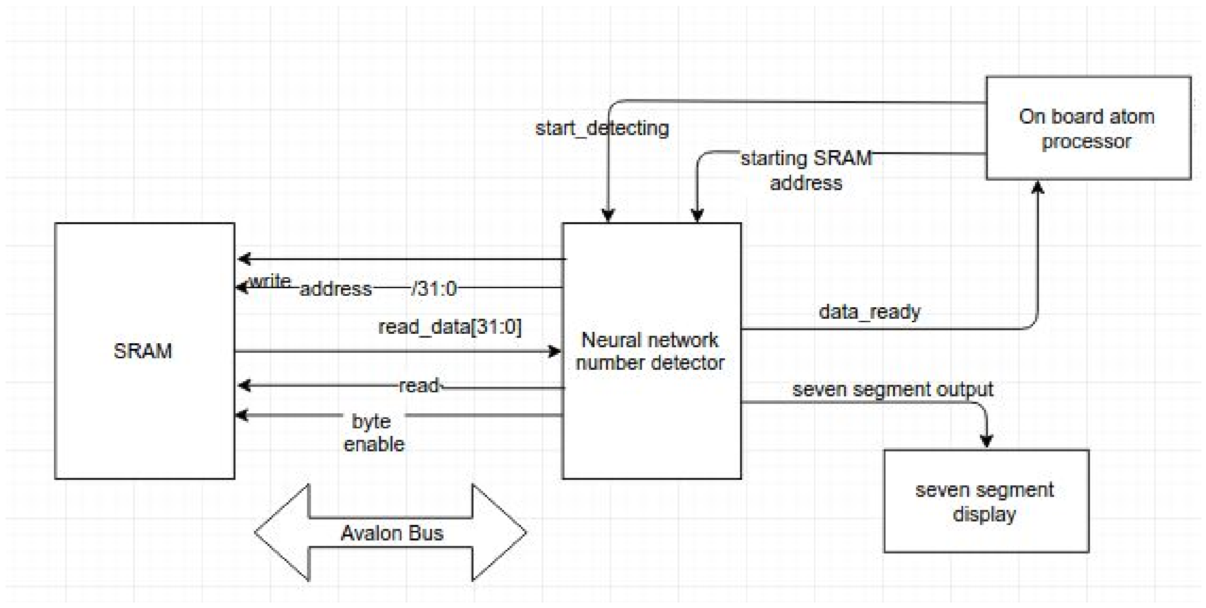


Figure 1: System Usage Diagram

The system described above is an artificial neural network. The network is controller by an onboard atom processor that send the neural network the location in memory of the data it will process and then a start signal. The ANN will then interface with SRAM to receive the image and coefficients used in the neural connections.

2.1.2 Implemented standards and algorithms

- Forward Propagation of Neural Network
  - Process by which the output is calculated from the input being passed through neural connections (described fully in 2.3.2.2)
- Activation function
  - Create a non linear activation function that will be used by each node as the final processing to each of its inputs
    - Look up table to find the desired value from a non linear transfer function
- SRAM communication
  - Reading information from SRAM as input for image. For our implementation, 1,232 weights are needed and two byte floating precision is to be used, so 2,464 bytes of memory is required for weights. Each input image requires 64 bytes of memory.

2.2 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 (100 MHz)
n_rst	IN	1	Asynchronous Reset. (Active Low)

Table 2: Interface Pinout Table

Signal Name	Direction (IN/OUT/Bidir)	Number of Bits	Description
Starting SRAM address	IN	32	The address where the data is being stored in SRAM
done_processing	OUT	1	Asserted high when the system is done processing the image.
7 segment connection	OUT	7	The shape detected by the system represented on a 7 segment display
start_detecting	IN	1	Asserted high when the Neural Network is to start detecting.
byte_enable	OUT	4	Per byte enables for signaling which bytes of the transfer are active. (Active High)
read	OUT	1	Asserted for Read Transfers (Active High)
readdata	IN	32	32-bit data bus for read transfers
write	OUT	1	Asserted for Write Transfers (Active High)
address	OUT	32	SRAM 32-bit Word Address

2.3 Operational Characteristics

2.3.1 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 as 8 x 8 byte matrix, and the coefficients for the neurons, as bytes. 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 nodes. The input to the nodes are predetermined coefficients and the input is the image. The final result of the calculations will display on a seven segment display which number the network thinks it has detected. Assuming that all logic gates have a rough delay of T, our combination block consists of at worst case a 16 bit multiplier (31T - 31 logic gates), a 16 bit adder (4T - 3 logic gates) for a total of 35T delay.

2.3.2 Interface overview

2.3.2.1 Loading Shape

We are using a custom protocol to store the shapes we are analyzing. We are storing the shapes as an 8 x 8 byte matrix, where each bit can vary from 0-255. 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.

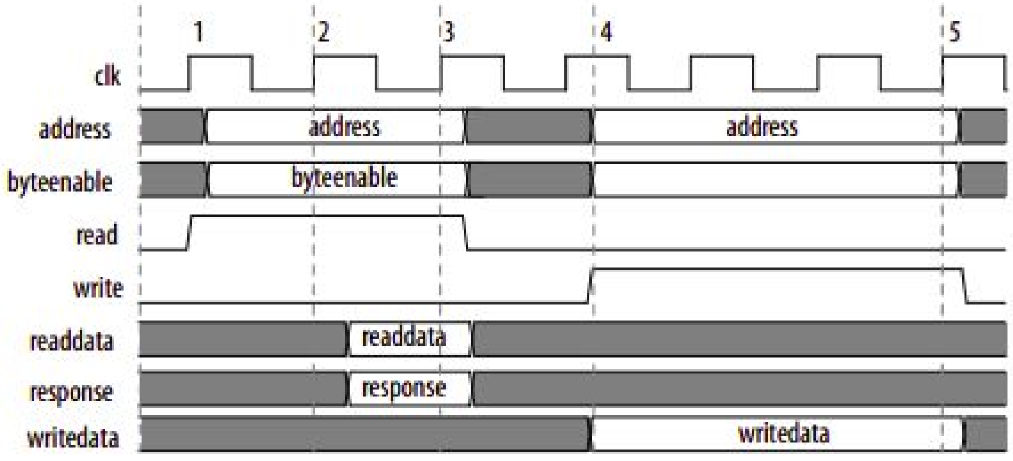


Figure 2: Read and Write Transfer with Fixed Wait-States at the slave interface. (Taken from Avalon interfacing manual)

2.3.2.2 Neural network implementation

The artificial neural network will be implemented by inputting an 8 x 8 byte matrix into two layers of neurons. Each neuron has connections that have a weight/coefficient corresponding to it. Every layer is smaller than the previous layer. Following these guidelines, the input is of size 64, so the first layer is 16 nodes, the second layer is 8 nodes, and the output will be 10 nodes. Each of the 10 final nodes represent a possible number that could be recognized.

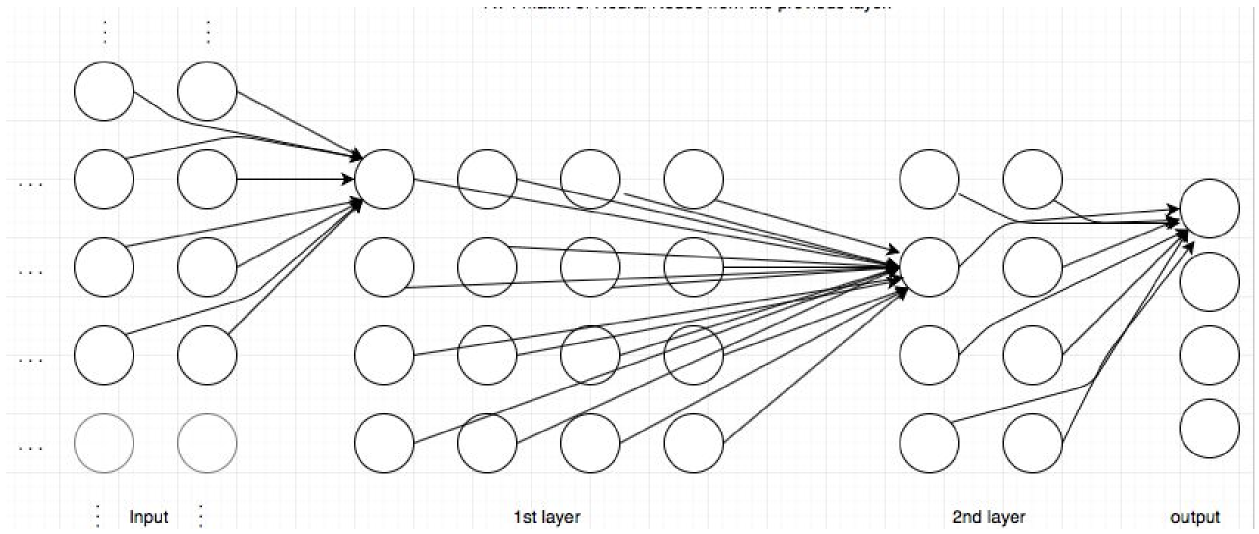


Figure 3: Neural Network Connections

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, this will be calculated using python and the weights will be loaded into the neural network to be sued. Forward propagation is then 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. 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 nodes output the value showing how much of a match the image was to each number.

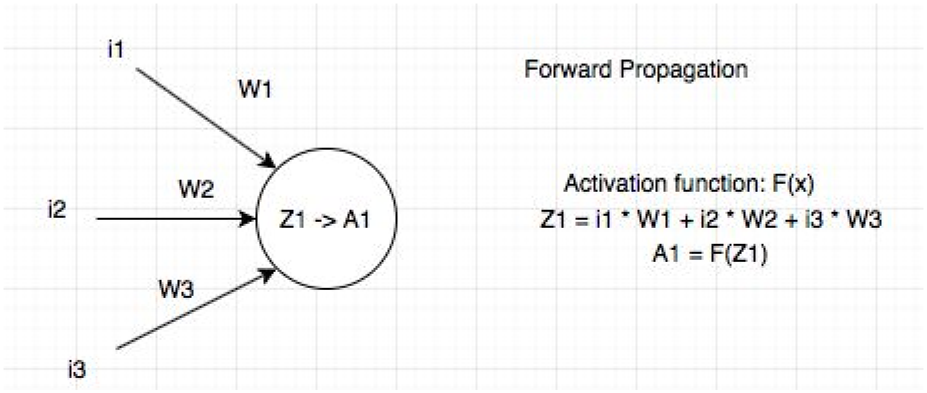


Figure 4: Single Node Calculation Diagram

Each node will have a 16 bit adder, 16 bit multiplier, and activation function circuitry with an accumulator. The accumulator starts at zero and on each clock cycle, adds the next product of inputs and weights together and sums it with the accumulator. A counter from outside each node inputs it's current value for a mux to choose which inputs and weights to multiply and add. The accumulator at the same time will output its value through the activation function which will be correct once the data is ready.

2.4 Requirements

For the neural network application to recognize shapes and determine errors, we have to be aware of the connections required to complete such tasks. We will focus to optimize for space as well as time because having space for each neuron in the network will expand quickly because of the exhaustive connections to the next level. At first, we thought about doing all of the node's multiplications in parallel but this would've resulted in 64 adders and multipliers per node. We redesigned our plan to save space and circuitry by having each node contain one adder and multiplier with an accumulator and performing the operations for one combination of input and weight each clock cycle. The number of neural connections also vastly increases how much space in SRAM we use. If our first layer had 64 nodes instead of 16, our first layer would need 16000 bytes of memory, and that is without the additionally layers that would also be larger. To save space in SRAM, we limited our first layer to a size of 16, second layer to a size of 8, and output layer to a size of 10 nodes for a total of 2,464 bytes needed to be stored in SRAM. Next, the clock frequency we are aiming for is 66.6 MHz 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.

3 Design Implementation
3.1 Design Architecture

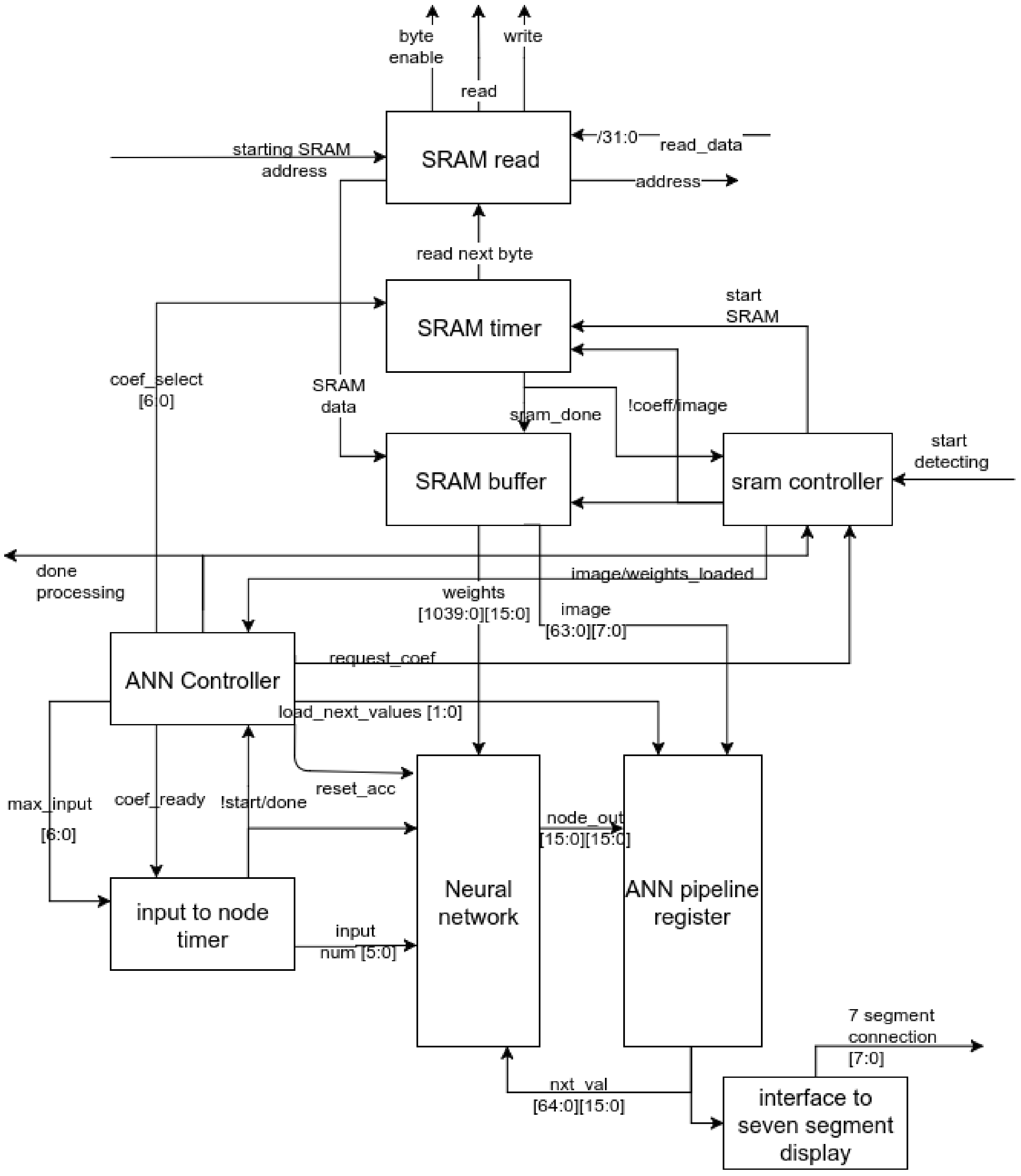


Figure 5: Architectural block diagram for Artificial Neural Network implementation

The implementation for this architectural block diagram is depicted above. The input image is loaded from SRAM as an 8 x 8 matrix of 16 bits. The coefficients for the ANN are loaded in from outside the design is stored into SRAM buffer for temporary storage.
The ANN control unit is the controller for the neural network. It sends signals to all of the timers and neural network blocks to progress the mathematical operations of the nodes. Since the nodes do the math of each input serially, the ANN controller controls the timer that selects the input, and once all inputs are done, stores the values correctly. The controller then signals to the SRAM controller to request more coefficients and repeats this process until all of the neural network layers have been processed.

The coefficients are the weights associated with each node to node connections and is used to show strength of connection between two different nodes. The output from the previous node is multiplied by the 16 bit fixed point scalar and is then fed into the next node. The new products are then added together and then sent on to the next next after it to continue the process. Each time the calculations run, the amount of nodes decrease, until the final layer, where there are 10 nodes, each representing a number that can be recognized. The output of this node represents the certainty that the number we are detecting match the one we want based on the coefficients.

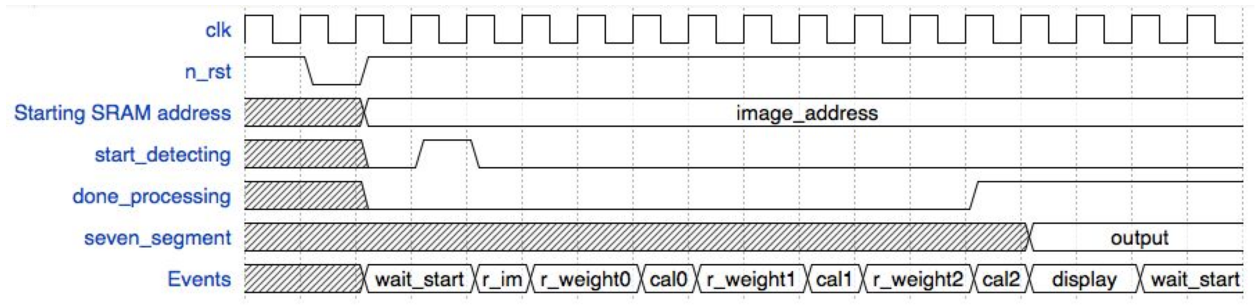


Figure 6: General waveforms of the operation of the entire architecture

The process begins once the start_detecting signal is asserted. Immediately, the image pointed to at the starting SRAM address is loaded into the SRAM buffer and formatted properly onto the pipeline registers. Afterwards, the layers of the neural network are rotated through by loading the weights of the first layer into the SRAM buffer and using them in the Neural Node connections. The state machine(s) wait on the calculations to finish and then the process is repeated until the final layer finishes where the display is shown. Finally, a transition back into the waiting to start state gives the option of running again.

3.2.1 Functional Block Diagrams

3.2.1.1 Node

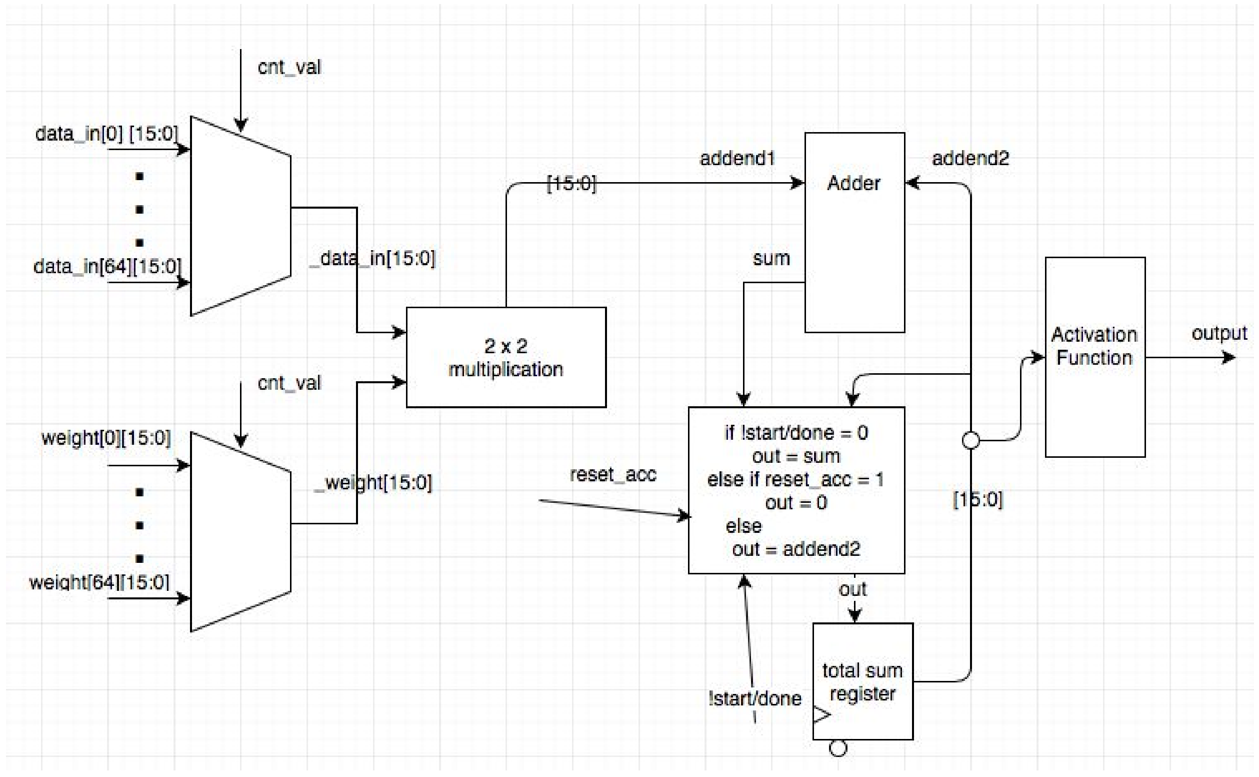


Figure 7: RTL diagram of a single neural node

The Neural node will be implemented with 2 multiplexers, an adder, a multiplier, and an activation function. The Multiplexers select which input and weight is being inputted to the rest of the circuit based on the counter value inputted into this block. The adder is an accumulator that starts from 0 and each clock cycle adds the product of the inputs and the weights to the current value in the accumulator. The value in the accumulator is also fed directly into the activation function combinational block and outputted. There is no data ready signal because these calculations are being done for many neural nodes so the counter/controller is a functional block up. Overall, this is performing a matrix multiplication between the inputs and weights. A timing diagram for the neural node is provided below.

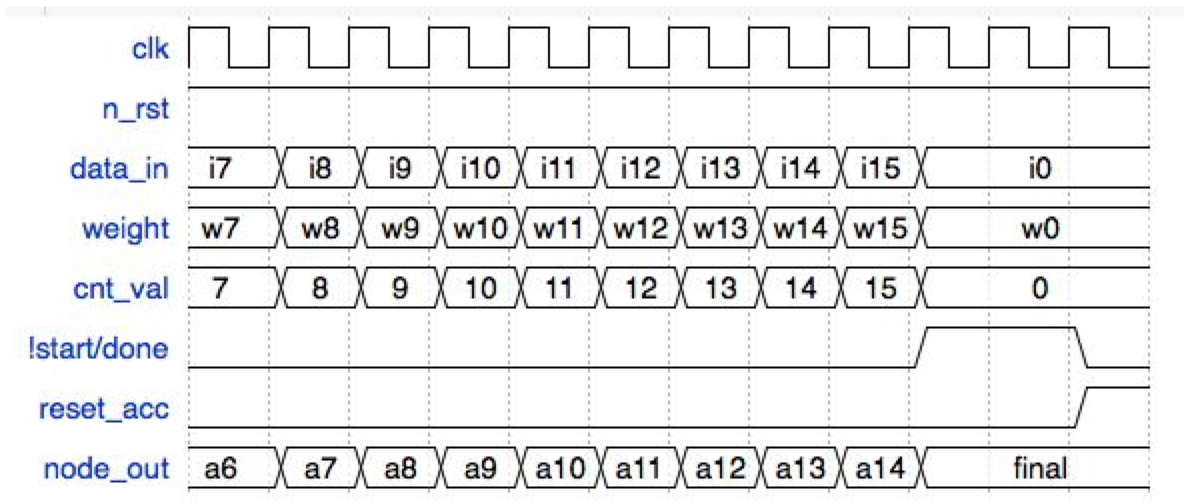


Figure 8: Timing diagram of a single neural node

As described in the timing diagram, the value of cnt_val chooses which data_in and weight from the multiplexer is used in the multiply and then accumulated. Our adder and multiplier originally started out as a sequential floating point blocks, however, after realizing the complexity, we swapped to fixed point such that it is all combinational. The clock rate is highly impacted by the critical path located between the multiplexer, multiplier, adder and the accumulator. The output is 1 cycle later than when the input was added because of the accumulator. Once the value from node_out has been read, the reset_acc is asserted and !start/done is set low, letting the process continue.

Name of Block	Category	Gate/FF Count	Area (um2)
Node			
Node Data mux	Combinational	65	48,750
Node weight mux	Combinational	65	48,750
Adder block	Combinational	38	28,500
Multiplication block	Combinational	40	30,000
Sum register	Combinational	16	12,000
Activation function	Combinational	20	15,000
Out Logic	Combinational	5	3,750

Figure 9: Node area requirements

The node has a lot of components that add up to about 186,750 um^2. This results from the number of large components in the node. The two multiplexers are equal in size because they require the same number of gates. The multiplication block also requires a large number of gates to compute the final value and it uses multiple full adders after the multiplication logic to do this. The adder uses full adders to compute the value and thus also requires about 38 gates. The sum register needs the 16 flip-flops for each bit to be stored. The output logic uses a small number of gates as a multiplexer to determine which output to use. Finally the activation function determines the final value that is to be output from the node and uses about 20 gates to do so.

3.2.1.1 Neural Network

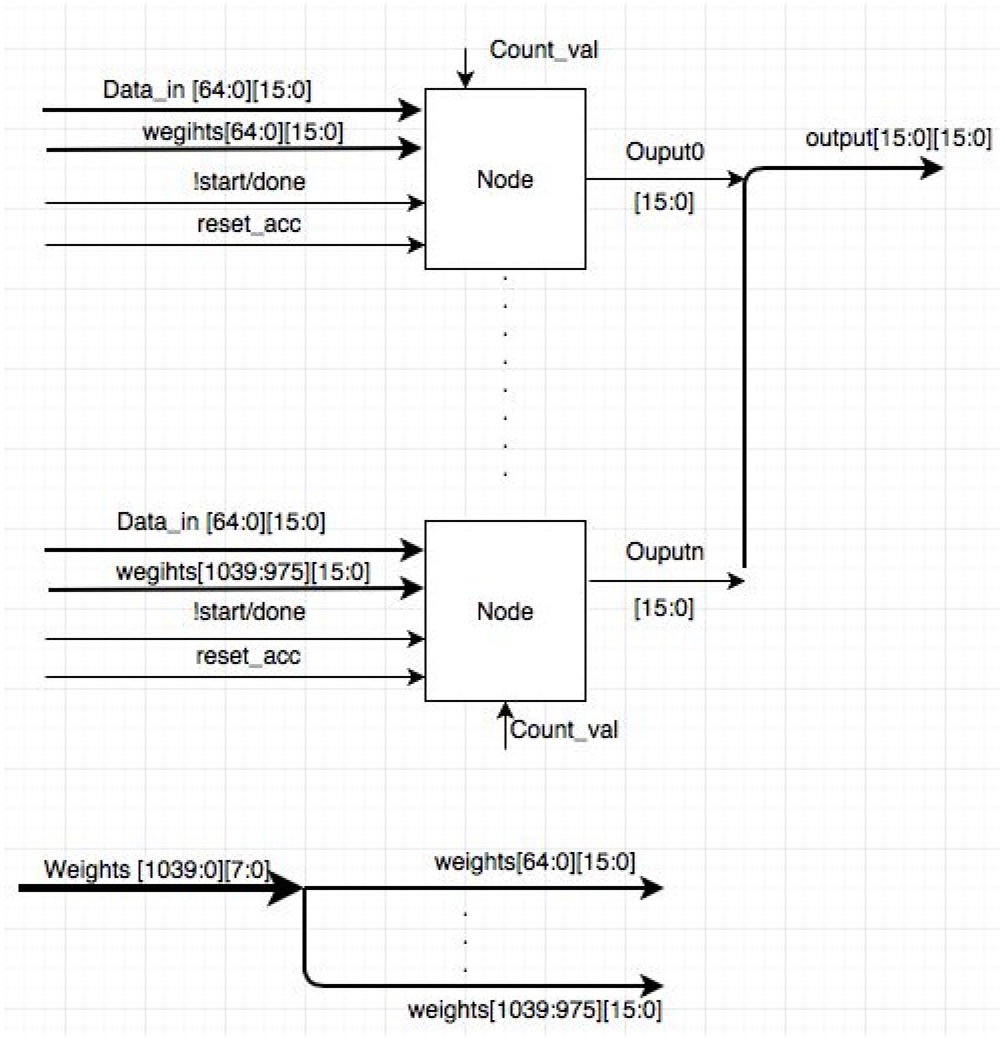


Figure 10: Functional Block Diagram of Neural Network

The Neural Network Block inputs takes the inputs and weights, puts them in their corresponding neural nodes, and then loops through the inputs/weights with a counter. Once the counter has reached it's flag_val, the calculations for that layer are done and the outputs are sent to registers for use in the next layer. A Timing diagram for a portion of when this block is run is shown in figure 10 below.

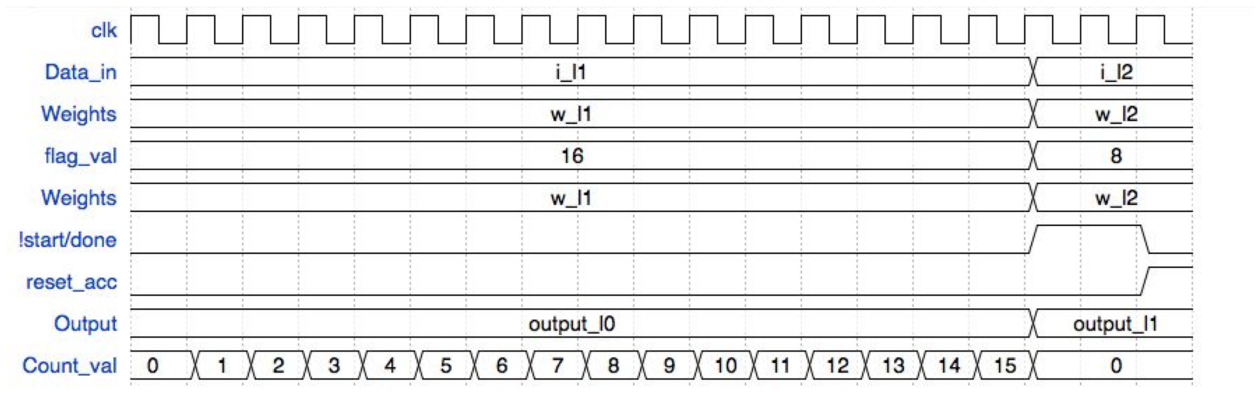


Figure 11: Timing diagram for Neural Network Block

For each layer, the Neural Network block has specific inputs and weights relative to each layer inputted. The flag_value is the value the counter will go up to. In the first layer, the flag_val would be 64 such the in each neural node, the 64 inputs would all be multiplied and accumulated with their set of 64 weights. In the second layer (the one described above), the flag_val is 16 and in the final layer the flag_val is 8. The !start/done signal disables the accumulator from adding any more of the values and then weights on the reset_acc signal to be asserted. The assertion of the reset_acc signal occurs after the output is stored.

3.2.1.3 Input to node timer

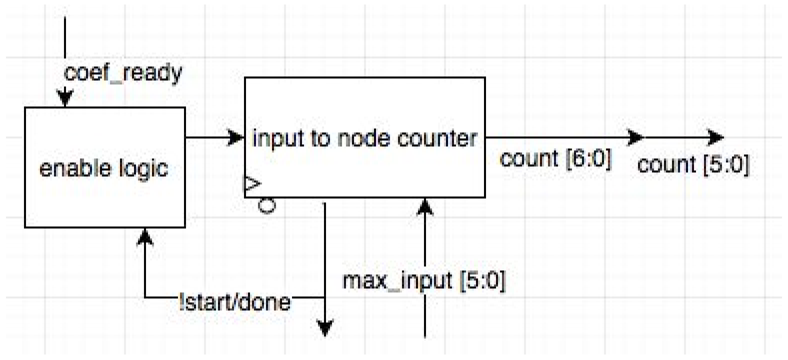


Figure 12: RTL for input to node timer

The input to node timer is used to control which input is being multiplied and added to the output accumulator. The input timer will count through all of the inputs, and once it is done, it will signal to the ANN controller that all calculation are done and that the next layer may be started. The !start/done signal also disables the accumulator in each of the nodes from adding anymore.

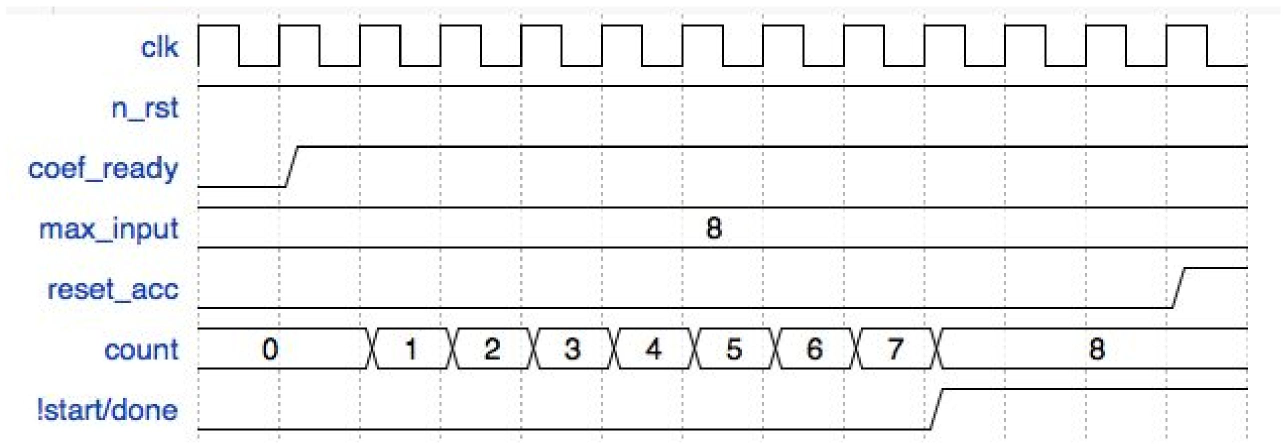
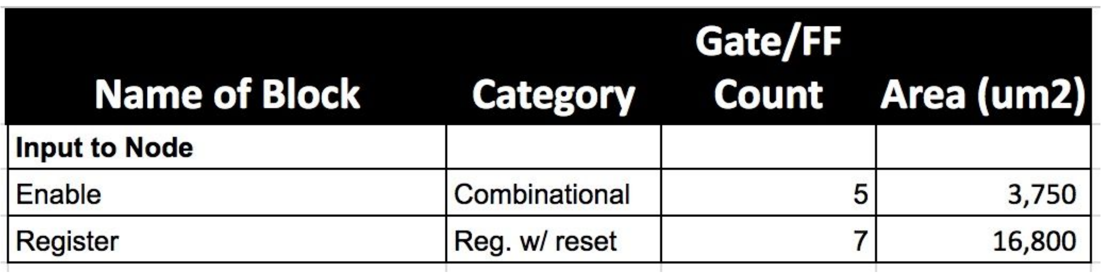


Figure 13: timing waveform for input to node timer

The input to node timer counts up to the maximum number of inputs fed into the system. This maximum value will change based on which layer node is being used. With the example waveform in figure 11, the max count is 8, so the timer will count up to 8. When the value is reached, the counter holds the value until coef_ready is set low, and then it is cleared and awaits coef_ready to go high again.



Name of Block	Category	Gate/FF Count	Area (um2)
Input to Node			
Enable	Combinational	5	3,750
Register	Reg. w/ reset	7	16,800

Figure 14: Input to node area requirements

The input to node module uses two components an enable register which determines the maximum number of inputs and this logic takes about 5 gates to compute. The register however has to use a large number of flip-flops to guarantee that the number can be stored. This leads to a final area of 20,550 um^2.

3.2.1.4 Seven segement display interface

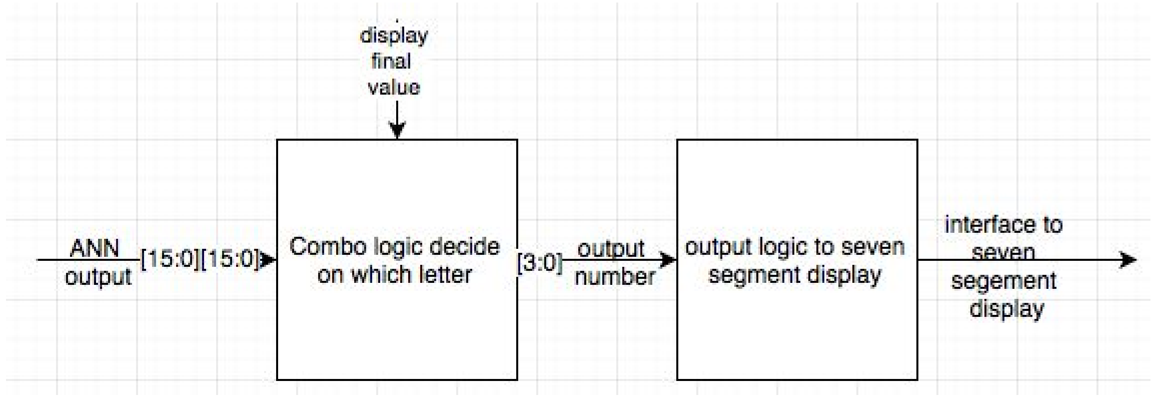
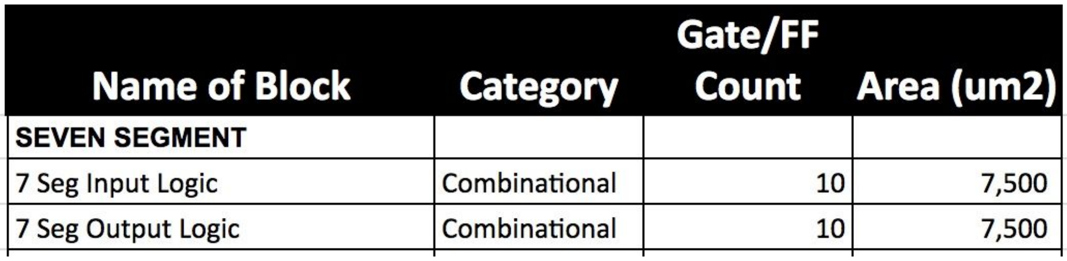

display
final
value
ANN
output
[15:0][15:0]
Combo logic decide on which letter
[3:0]
output
number
output logic to seven segment display
interface to seven segement display

Figure 15: Combinational logic to interface with seven segment display

The seven segment display interface is just two decoder blocks. The ANN output has 10 relevant nodes under consideration. The value of the nodes will be between 0 and 1. The decoder block will determine the maximum value out of all of the nodes and if it is above a certain threshold, output the number the node corresponded to. The number is represented in binary and decoded into a signals to interface with the seven segment display.

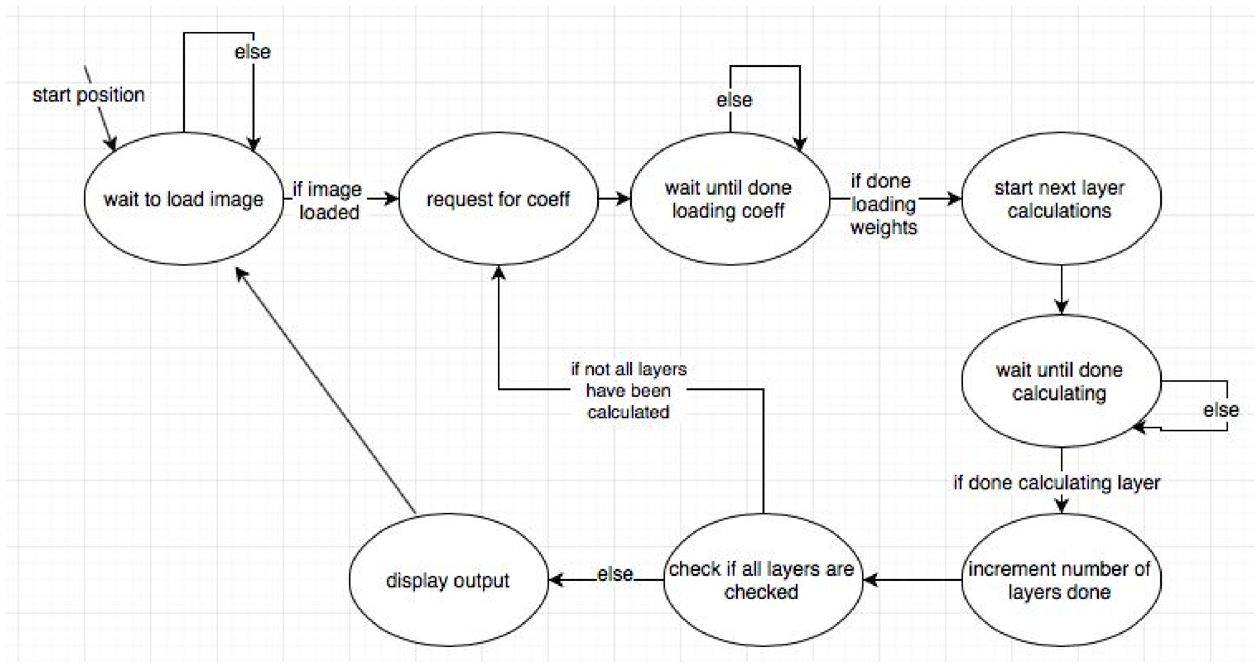


Name of Block	Category	Gate/FF Count	Area (um2)
SEVEN SEGMENT			
7 Seg Input Logic	Combinational	10	7,500
7 Seg Output Logic	Combinational	10	7,500

Figure 16: Area Components of the seven segment display block

The timing seen below shows how small this component is with a total area of just 15,000um^2. This includes the logic for both the input and output to the seven segment display. The logic performed is combinational but requires a large amount of computation which leads to a high number of gates in both the input and output logic.

3.2.1.5 ANN Controller



start position
else
wait to load image
if image loaded
request for coeff
else
wait until done loading coeff
if done loading weights
start next layer calculations
wait until done calculating
else
if done calculating layer
increment number of layers done
check if all layers are checked
if not all layers have been calculated
else
display output

Figure 17: ANN FSM

The ANN controller is a moore model state machine. It waits to be told that an image has been loaded before entering a loop to do calculations. The loop is run until all layer outputs have been calculated. To perform the calculations, the weights/coefficients corresponding to the current layer must be loaded, then the calculations are performed. The controller tells the timer what number to count to in order to perform the matrix multiplication inside each node. While the matrix multiplication (multiply, add, accumulate) the state machine waits for a done signal, at which point it either moves onto the next layer or displays the outputs. After display the output the state machine goes back into the waiting state to await input to start again.

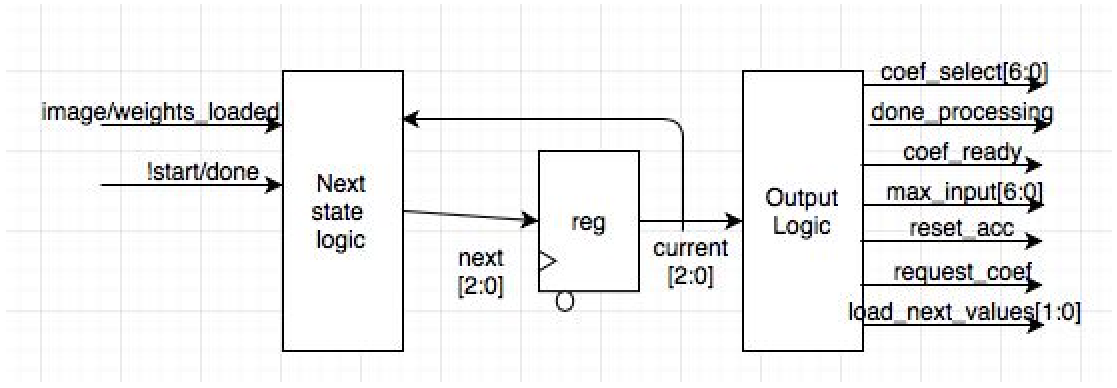


Figure 18: Moore Machine of ANN FSM

Name of Block	Category	Gate/FF Count	Area (um2)
ANN Controller			
Ann State control	Reg. w/ Reset	3	7,200
Ann next state control logic	Combinational	20	15,000
Ann Output logic	Combinational	10	7,500

Figure 19: Area Totals of ANN Controller

The table above shows the areas for the register which will only require 3 flip-flops for the controller because of the 8 states. There is also space required for both the next state logic as wells as the output logic used in this component. The next state control logic requires more gates and takes up the majority of this block it takes approximately 20 blocks for the large number of input and output bits. The output uses the current signal to determine a number of outputs that are used to control the neural network as well as the timer.

3.2.1.6 SRAM Controller

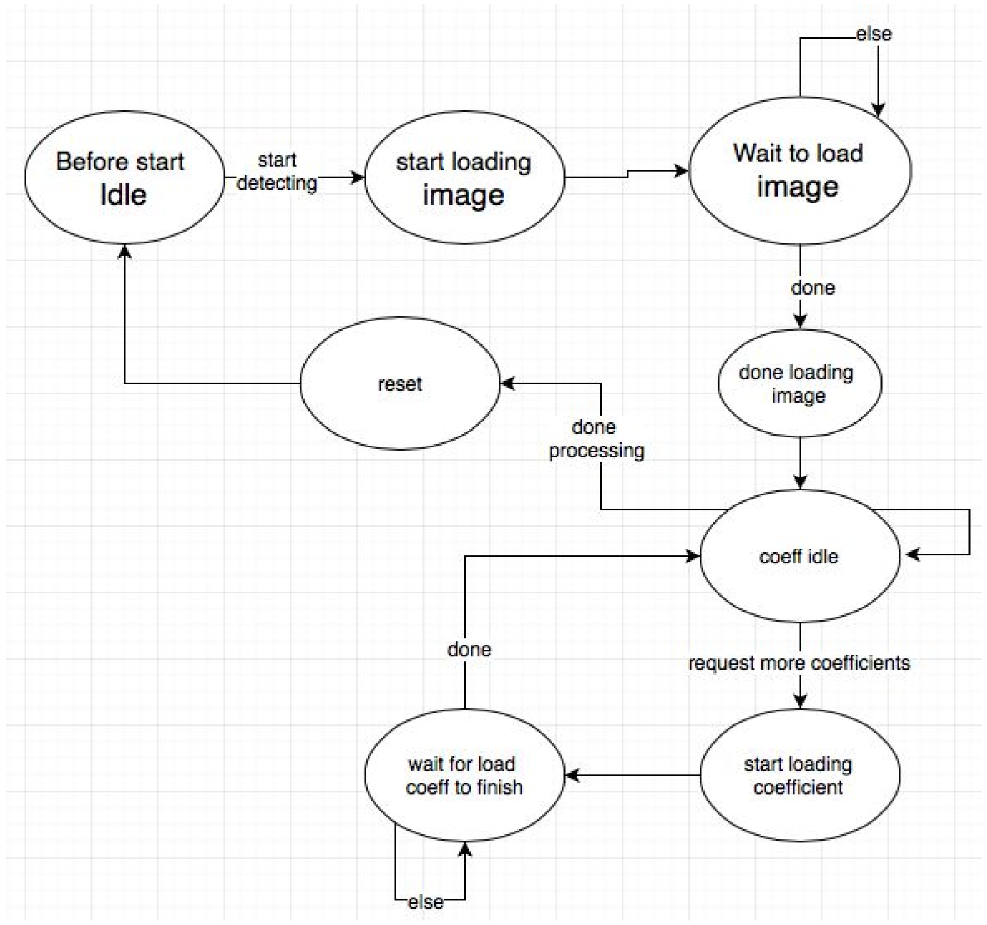


Figure 20: SRAM FSM

The SRAM controller is also a moore model FSM. Once the start detecting signal is asserted, an initial load of the input image from SRAM is performed. The state machine waits on the load to finish and then enters a coeff idle loop. The coeff idle loop waits until the ANN state machine has signaled that it is ready for the next weights/coefficients. Once the request more coefficients signal has been set high, the state machine signals SRAM to load the specific weights from memory, waits for it to finish loading, and then returns to coeff idle state. The process is repeated until all calculations for the neural network are complete, signalling that all the weights have been used and the state machine can reset and once again wait to start again.

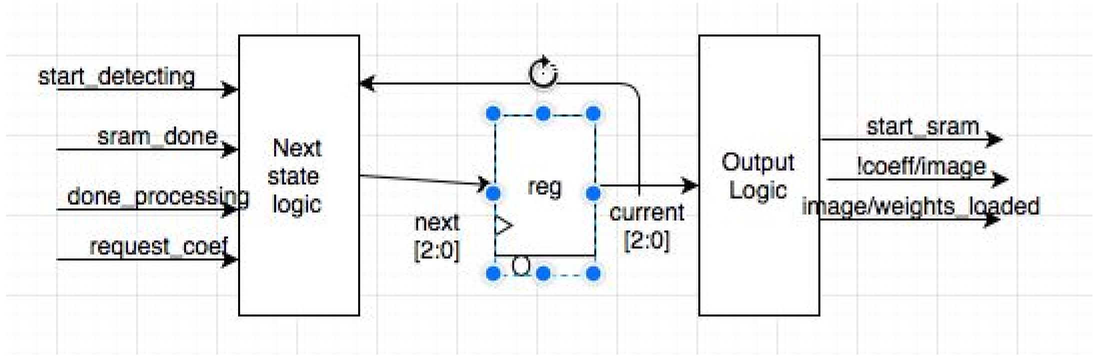


Figure 21: Moore Machine RTL for SRAM FSM

Name of Block	Category	Gate/FF Count	Area (um2)
RAM CONTROLLER			
RAM state control	Reg. w/ Reset	3	7,200
RAM next state control logic	Combinational	20	15,000
RAM output logic	Combinational	10	7,500

Figure 22: SRAM Controller Area Allotments

The area needed for this controller is 29,700 um^2. We use another state machine which requires a register, next state logic, and output logic. This uses about the same amount of space that is used for the ANN Controller for the same reasons as above. Most of the area is designated to the combinational logic needed to control the states and the output logic that is used for the outputs from the module.

3.2.1.7 ANN Pipeline Register

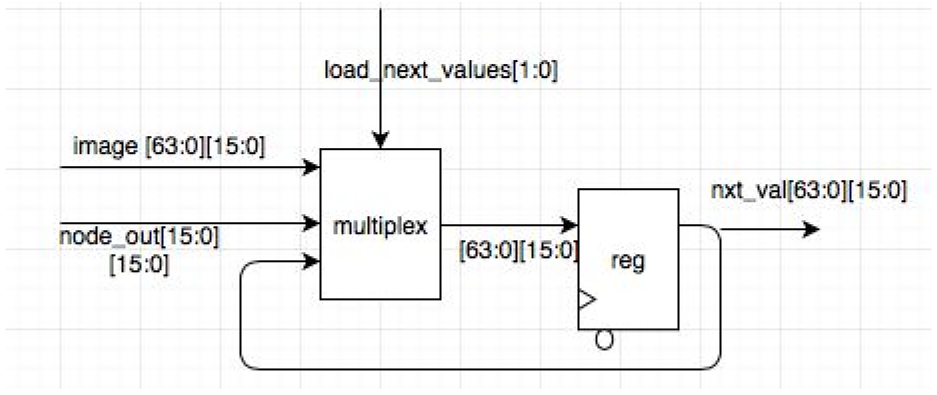
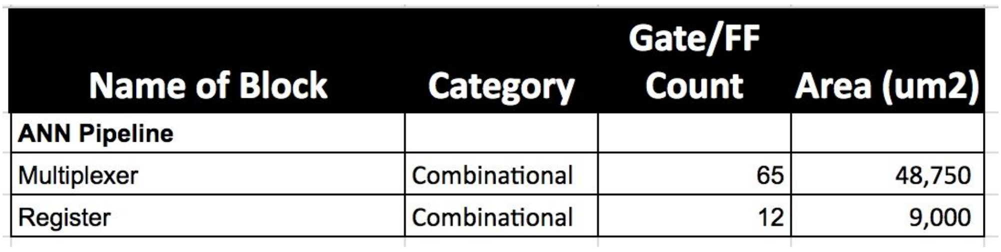


Figure 23: Moore Machine RTL for SRAM FSM

The pipeline register stores the output from each neural layer and inputs it into the next layer. Load_next_values is used to multiplex between loading the image, loading the output from the neural layer, and maintaining current state for when calculations are still ongoing.



Name of Block	Category	Gate/FF Count	Area (um2)
ANN Pipeline			
Multiplexer	Combinational	65	48,750
Register	Combinational	12	9,000

Figure 24: ANN Pipeline area sheet

The ANN pipeline module is very small in size with an area of 57,750 um^2. The majority of the size comes from the multiplexer because it contains so many gates. The register also requires 12 flip flops from the large amount of data stored in the register.

3.2.1.8 SRAM timer

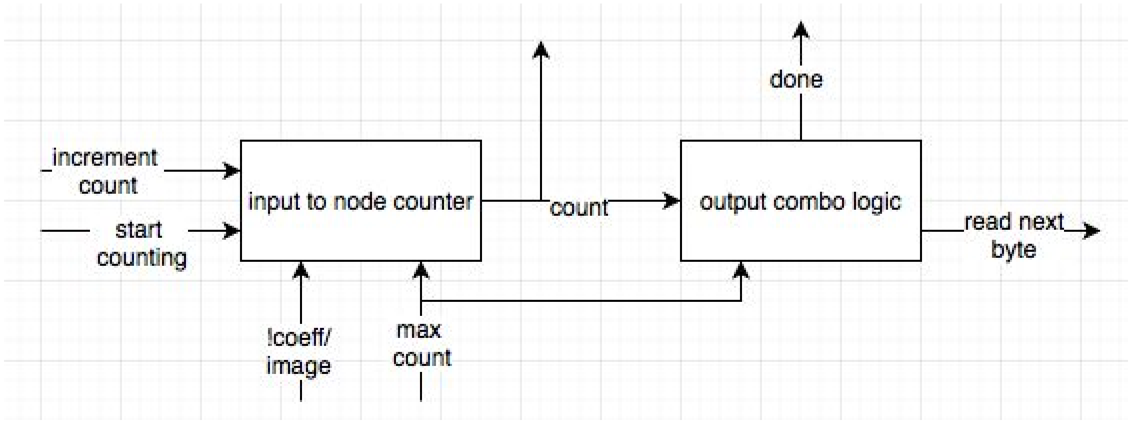


Figure 25: SRAM timer block diagram

The SRAM timer is used trigger a new read from the SRAM. The timer is given the amount of coefficients it needs to load and a start signal. It will then drive the SRAM read module to keep requesting data until all of the coefficients/image has been received.

3.2.1.9 SRAM Read

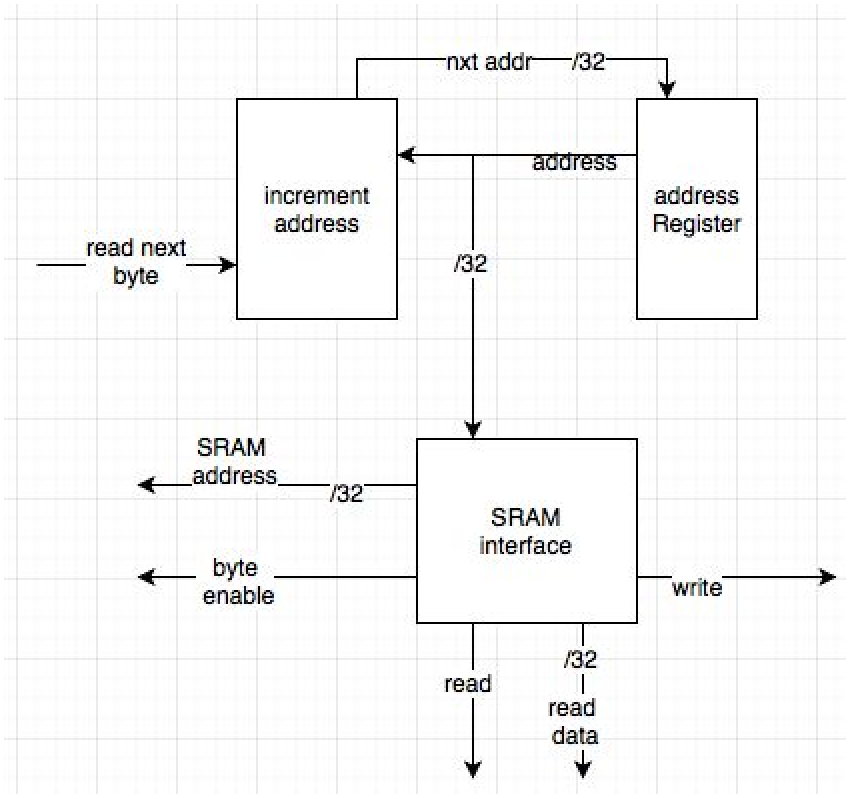


Figure 26: SRAM read block diagram

The SRAM read module is a wrapper module for the SRAM interface block. The block will load the initial SRAM address. Then each time the SRAM timer tells the SRAM read to get more information, the internal address will be incremented and then the address will be sent to the SRAM interface along with a read command to request more information.

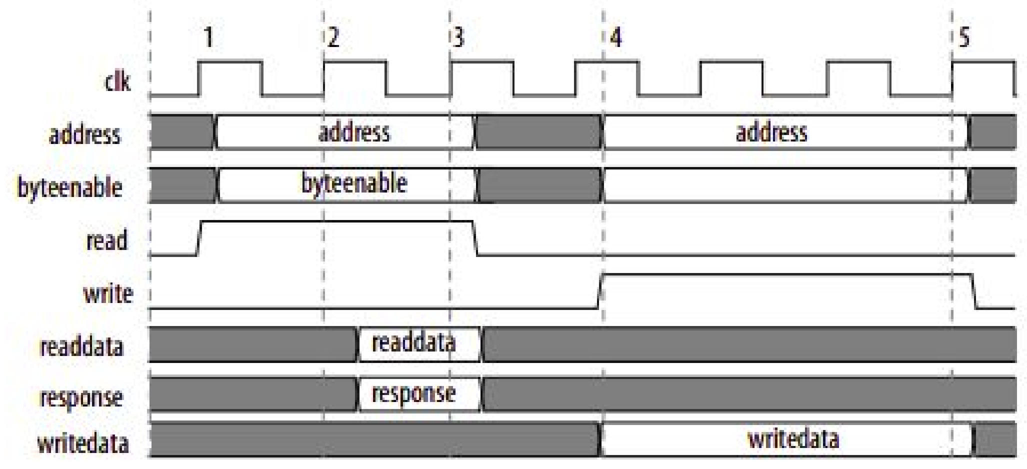
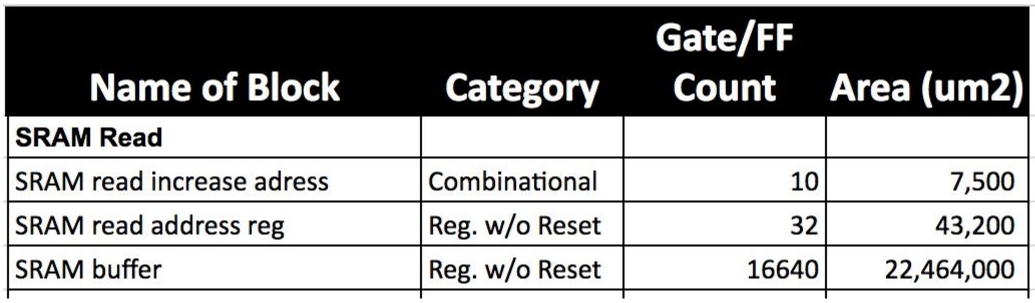


Figure 27: SRAM read timing diagram example



Name of Block	Category	Gate/FF Count	Area (um2)
SRAM Read			
SRAM read increase adress	Combinational	10	7,500
SRAM read address reg	Reg. w/o Reset	32	43,200
SRAM buffer	Reg. w/o Reset	16640	22,464,000

Figure 28: SRAM Read area requirements

The SRAM read takes up the most area by far because of the SRAM buffer. This ae mostly comes from the number of flip-flops needed to transfer data from the SRAM to the read module. We also have a significant amount of space allocated to the read address register for the 32 bits that need to be stored and updated. Finally, the increase address puts the total size at 22,514,700 um^2. The combinational logic is minimal but there is space required for the logic gates to update the address for the register.

3.2.2 Design Timing Analysis

A main concern in designing this neural network was determining if we could accomplish our goal with a reasonable time. After carefully reviewing our design we have come up with five main critical paths that will limit the speed of our clock. These paths can be seen in Figure 22 below and show how much of a range of time our components require. With our most critical path we can see that it will take substantially longer than any other component in this network and thus will leave a lot of idle time for our shorter paths.

Starting Component	Propagation Delay (ns)	Combinational Logic	Propagation Delay (ns)	Ending Component	Setup Time or Propagation Delay (ns)	Total Path Delay (ns)	Target Clock Period (ns)
node data_in	0.1	Multiplication, adder, and mux select	10.2	none_output	0.2	10.5	15
total sum register	0.1	adder, value select	6.6	total sum register	0.2	6.9	15
Next_byte_ready	0.1	SRAM interface	0.6	Read_data	5	5.7	15
Coef_ready	0.1	enable logic	1.2	count	0.2	1.5	15
ANN output	0.1	decide number, output logic	1	7 seg interface	0.2	1.3	15

Figure 29: Five most critical paths

3.2.2.1 Critical Path #1

In the node module we have a two very large combinational blocks that slow down the network substantially. We start at the input of the node move through the multiplexer which has a relatively slow delay of 0.6ns. This is because most of the gates required can calculate in parallel from the separate inputs. The second delay comes from the multiplication logic which is purely combinational. This incurs a delay of 3 ns because of all the gates that are required for the calculation. The adder has a larger delay of 6.0ns and produces the largest delay of the node. Using the full adder requires a large amount of time. This value is then used in the multiplexer selection which determines the output for the next calculation this only requires a delay of 0.6 ns. Finally, we have reached the total sum register which requires a setup time of 0.2 ns. In total the delay for this path in the node is 10.5ns and adding time for wiring we decided on a 15ns timer.

3.2.2.2 Critical Path #2

The second critical path also is seen in the node but this time the source starts at the total sum register which has a delay of 0.1ns. This value then moves to the register which again has a delay of 6.0ns and this output is sent to the value select multiplexer which has a delay of 0.6ns. Finally we return to the total sum register which takes 0.2 ns for the setup time of the flip-flops. This gives us a total time of 6.9 ns which is reasonably smaller than the previous critical path.

3.2.2.3 Critical Path #3

This third module is a small module but because it interfaces with the SRAM the delay is increased. We start the path at the address signal which comes from the address register and takes 0.1ns to be available and moves directly to the SRAM interface logic. This output takes about 0.6 ns to generate from the logic. Up until now the delaye is reasonably fast however the read from SRAM takes 5ns which puts us at a total of 5.7ns.

3.2.2.4 Critical Path #4

The fourth path is found in the Input to Node Timer block. The path starts at the coefficient ready signal which has a delay of 0.1ns. The signal is the fed to the enable logic block which takes about 1.2ns to complete. This is then given to the input to node counter which adds 0.2 ns to the delay. Finally we end up with a delay of 1.5ns.

3.2.2.5 Critical Path #5

The final critical path comes from the seven segment display module. This path starts with the ANN output which requires a delay of 0.1ns and then moves to the logic block used to decide the number for the display output. This log takes approximately 0.4ns. Then we have the output logic which requires a delay of 0.6ns and gives our final combinational total of 1ns. Displaying to the seven segment display should take an additional 0.2ns which leads to the final value of 1.3ns for the block.

4. Project Management

- Week 10: design budget
    - Low level RTL diagrams
        - Node: Cheyenne. Jack
    - Controller state diagram
        - Ryan
    - Python ANN simulation finished
        - Taylor
- Week 11: design budget
    - Compile Node in verilog
        - Cheyenne
    - Test bench for node
        - Jack
    - Generate coefficients for one shape
        - Taylor
    - How to load and store coefficients
        - Ryan
- Week 12: Design Review
    - Create purely combinational test for ANN
        - Cheyenne
    - Create test bench for combo data
        - Taylor
    - Start creating ANN sequential framework
        - Write controller: Ryan
        - Everything else: Jack
- Week 13
    - Work out bugs on neural network
        - Cheyenne, Taylor
    - Test sequential implementation logic and framework
        - Ryan, Jack
- Week 14

- Put the nodes into the sequential logic implementation
    - Cheyenne, Taylor, Jack
- Load in coefficients
    - Ryan

5. Success Criteria

Fixed Criteria

1. Test benches exist for all top level components and the entire design. The test benches for the entire design can be demonstrated or documented to cover all of the functional requirements given in the design specific success criteria. - 2 points
2. Entire design synthesizes completely, without any inferred latches, timing arcs, and, sensitivity list warnings. - 4 points
3. Source and mapped version of the complete design behave the same for all test cases. The mapped version simulates without timing errors except at time zero. - 2 points
4. A complete IC layout is produced that passes all geometry and connectivity checks. - 2 points
5. The entire design complies with targets for area, pin count, throughput (if applicable), and clock rate. The final targets for these parameters will be determined by course staff based on your design review. - 2 points

Design Specific Criteria

1. Create a single functional node - 1 point
2. Generate test example of ANN using python to verify verilog results - 1 point
3. Create a non linear node activation function in verilog - 1 point
4. Create a smaller (4 x 4 input image) version of the ANN in all combinational blocks - 2 points
5. Create a larger(8 x 8 input image) sequential version of the ANN - 2 points
6. Put the project on an fpga - 1 point