Artificial neural networking

Phase 3 Proposal

Submission Date: 10/28/2016 Wednesday 11:30am Lab (Section 2)

TA: Mingxuan He, Annan Ma

Prepared by:

Ryan McBee : [illegible]

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.

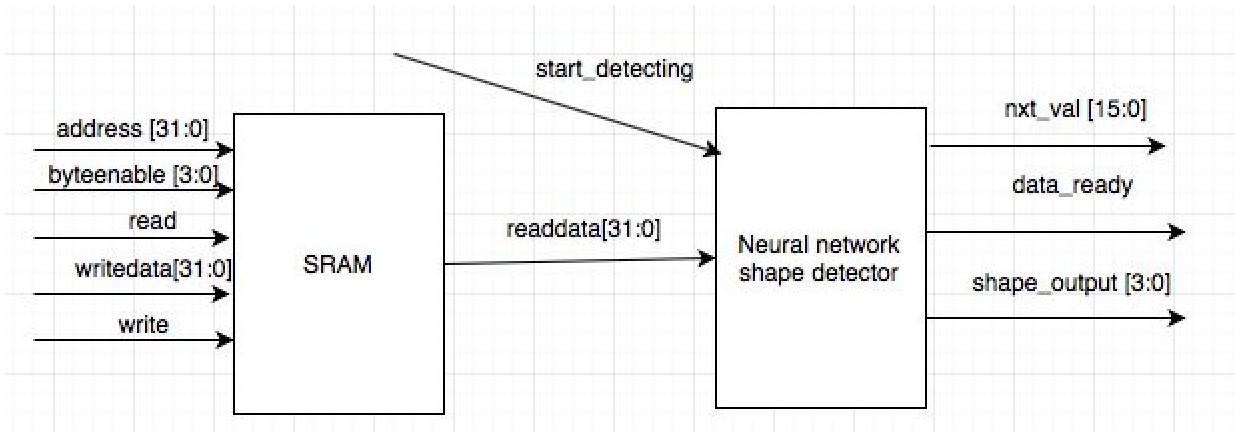
The goal for our project is to create the framework of the artificial neural network, and have the ANN read in the predetermined learned coefficients. This type of implementation provides a faster and more efficient method of representing and processing the neural network as compared to using a purely 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 that takes in an image to be analyzed. The system will take in a 8 x 8 byte grayscale image from SRAM and process it to detect which image we are looking at.

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, 1,184 weights are needed and two byte floating precision is to be used, so 2,368 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
nxt_val	OUT	16	The shape that is being inputted.
Data_ready	OUT	1	Asserted high when the system is done processing the image.
shape_output	OUT	4	The shape detected by the system represented as a binary match.
start_detecting	IN	1	Asserted high when the Neural Network is to start detecting.
byteenable	IN	4	Per byte enables for signaling which bytes of the transfer are active. (Active High)
read	IN	1	Asserted for Read Transfers (Active High)
readdata	OUT	32	32-bit data bus for read transfers
write	IN	1	Asserted for Write Transfers (Active High)
writedata	IN	32	32-bit data bus for write transfers
address	IN	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, names of the shapes, as a byte, and the coefficients for the neurons, as bytes, and the thresholds that match the shape, 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 final result of the calculations will then be fed into a combinational block that will calculate how close the value is to the desired image. Assuming that all logic gate 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 Shaped

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.

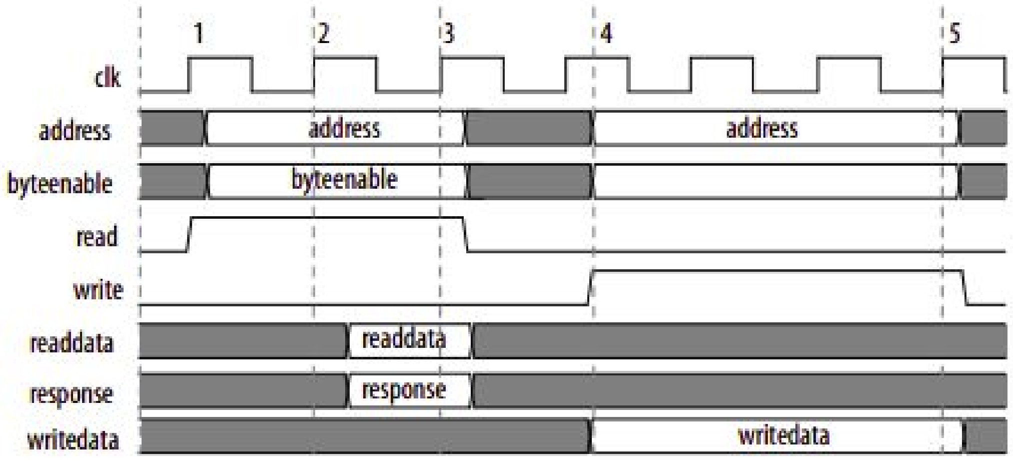


Figure 2: Read and Write Transfer with Fixed Wait-States at the slave interface. (Taken from Avalon interfacing manual)

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 4 nodes that output a binary representation that matches to the shape.

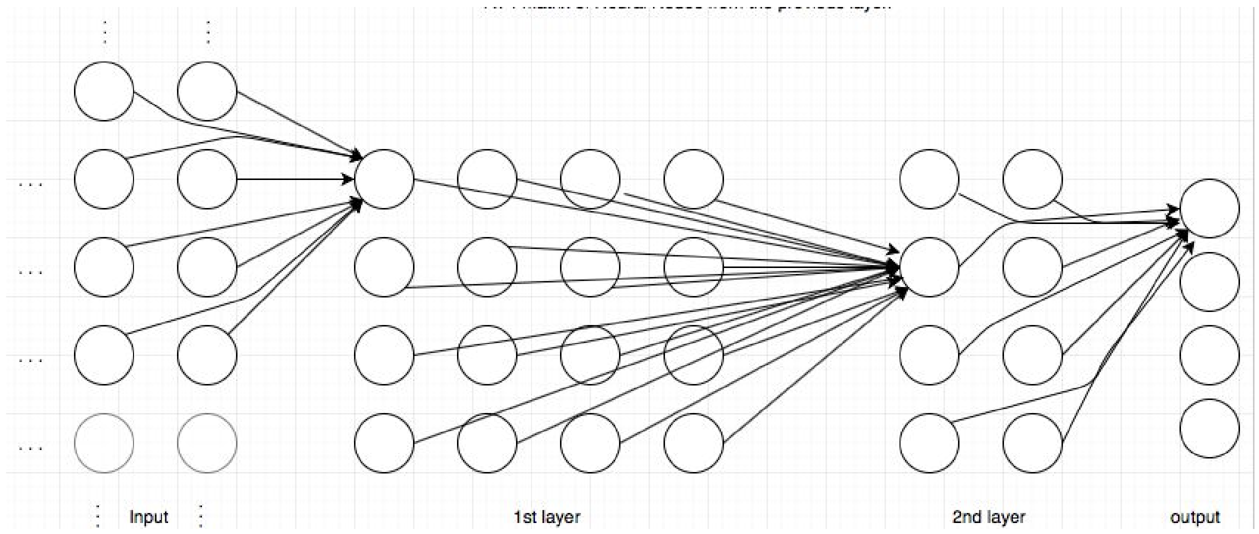


Figure 3: Neural Network Connections

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, initially this will just be calculated using python and the weights will be loaded in. 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. 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 a binary representation matching to a shape where all zeros is not a match.

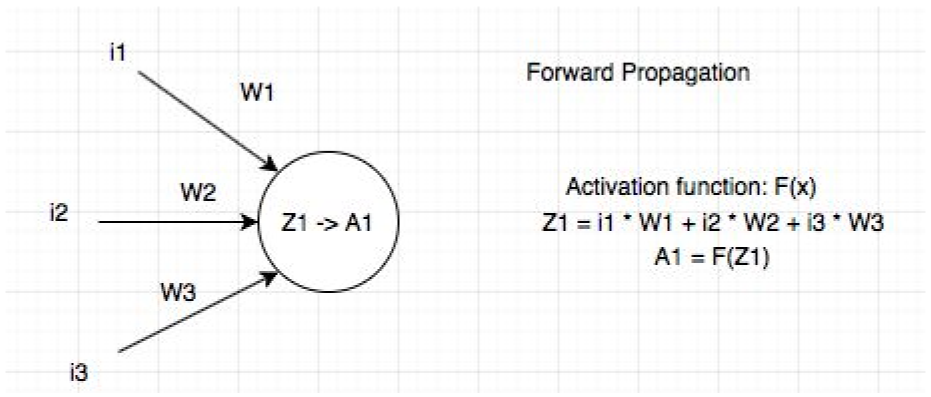


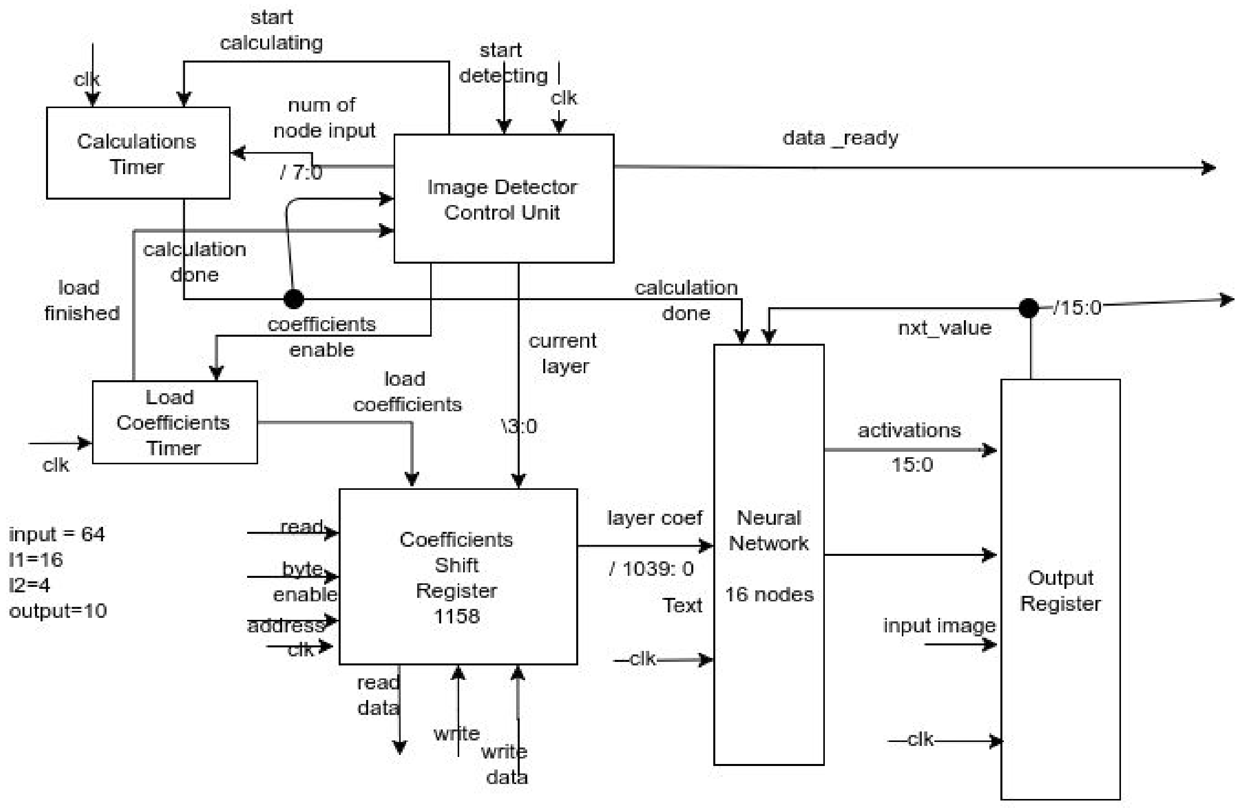
Figure 4: Single Node Calculation Diagram

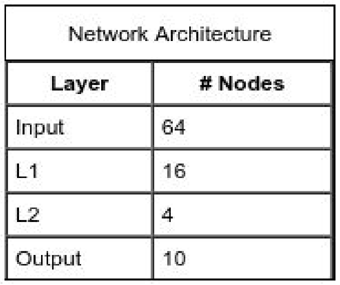
Each neural node will have a 16 bit floating point adder, 16 bit floating point multiplier and activation function circuitry with an accumulator. The accumulator starts at zero and on each clock cycle, adds the next product of inputs and weights together and sums it with the accumulator. A counter from 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 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. At first, we thought having each node do it's individual calculations with combinational logic, however, this would've results in a 64 adders and multipliers per node. To make this design possible, 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 8192 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 4 nodes for a total of 2,368 bytes needed to be stored in SRAM. Next, the clock frequency we are aiming for is 100 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.

3.1 Design Architecture





Network Architecture	
Layer	**# Nodes**
Input	64
L1	16
L2	4
Output	10

Figure 5: Artificial Neural Network implementation

The implementation for this architecture 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 a shift register for temporary storage. The timing diagram for the SRAM is figure 2 above.

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.

Two timing blocks are used to help handle the different timing constraints of loading and calculating. The first timer is the load coefficients timer. It is started by the controller after successful calculations for the nodes and ends the SRAM reading when all the desired coefficients have been loaded in. The other timer block handles the timing for the calculations sections of the nodes. It waits until calculations are finished, and then signals to the registers and the controller that calculations are finished.

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 8 bit floating 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 there is only one node left. The output of this node represents the certainty that the shape we are detecting match the one we want based on the coefficients.

3.2 Functional Block Diagrams

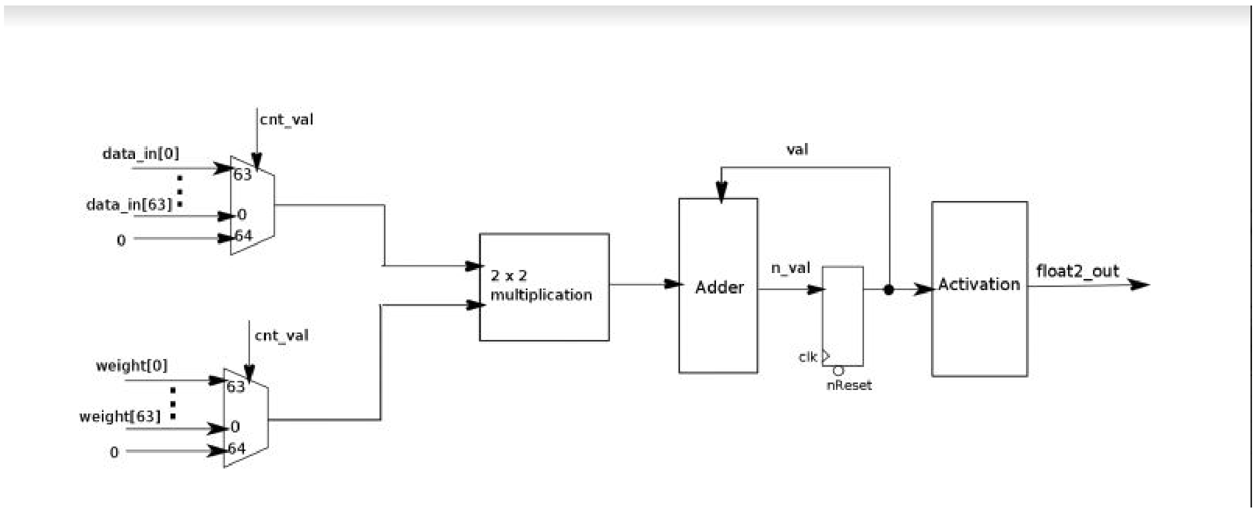


Figure 6: RTL diagram of a single neural node

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. A timing diagram for the neural node is provided below.

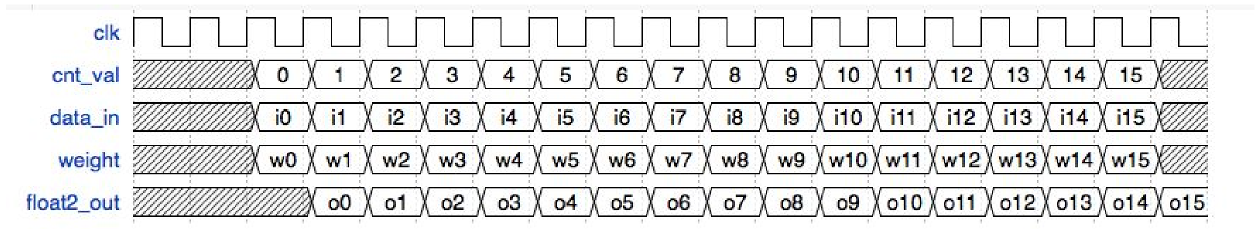


Figure 7: Timing diagram of a single neural node

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. The output is 1 cycle later than when the input was added because of the accumulator.

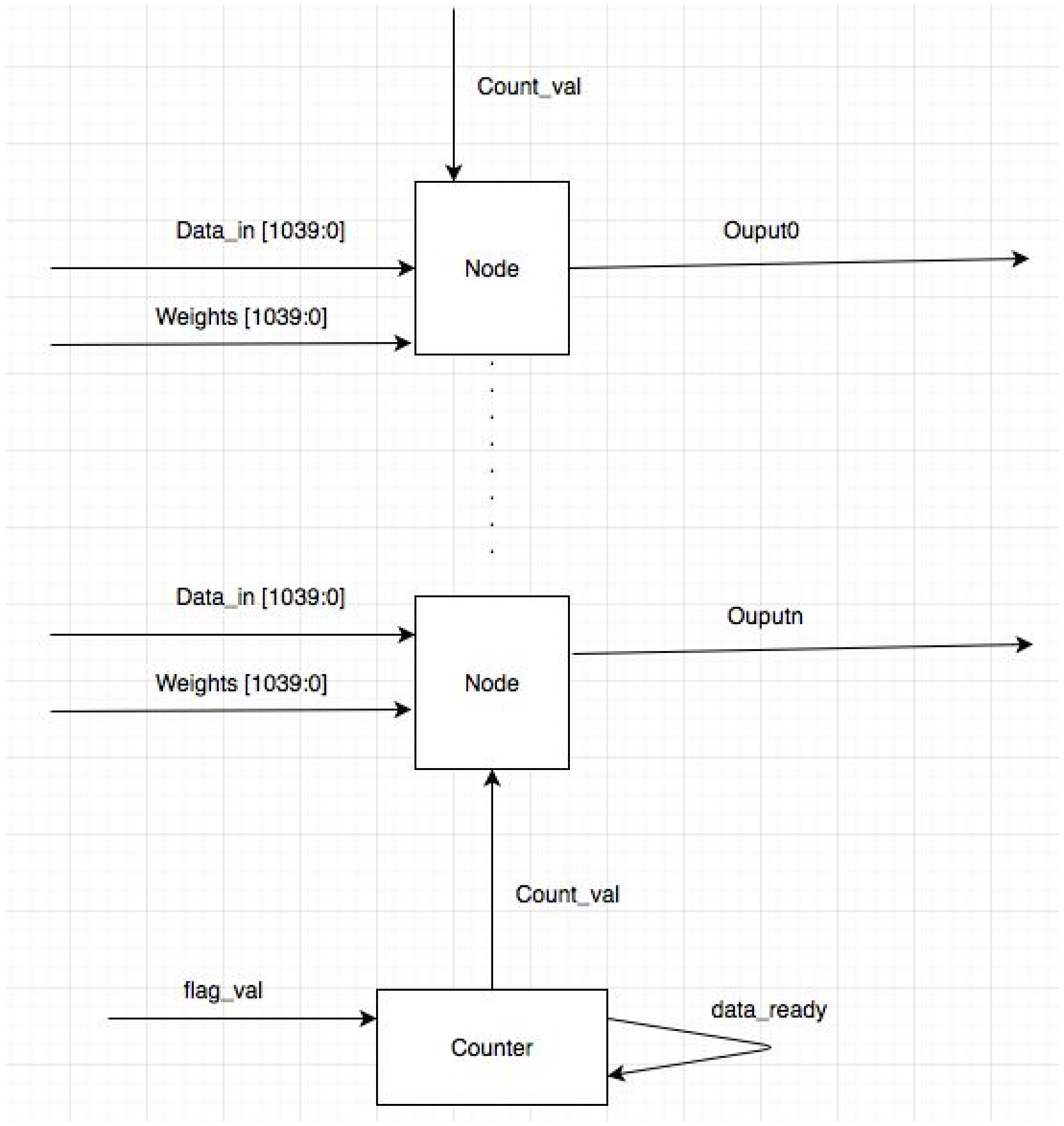


Figure 8: Functional Block Diagram of Neural Network

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 9 below.

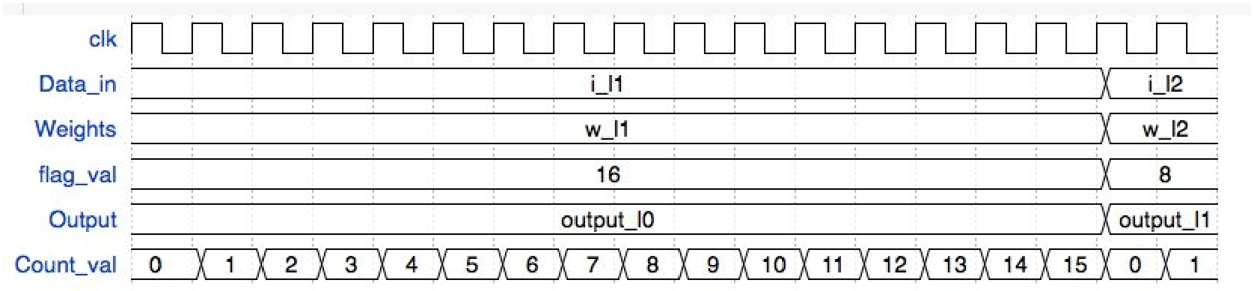


Figure 9:Timing diagram for Neural Network Block

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.

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