Handwritten Digit Recognizer using Neural Networks

Final Report
TA: Anirudh Sivakumar
Wednesday — 11:30

David Pimley
Chan Weng Yan
Dustin Andree
Vadim Nikiforov

ECE 33700
5/2/2018

Purdue University
Computer Engineering

Contents

1 Executive Summary

This project idea is a digit recognizer. It is designed to work with images consistent with the MNIST dataset[1], so it accepts 12×12 px grayscale images, and outputs a detected digit. It accepts an image, and a list of weights and biases for the network connections. While it doesn't have an internal method for training, it outputs a cost function for each set of images, allowing for it to be connected to an external training module. Furthermore, using SPI, the network model can be reprogrammed on the fly. Digit recognition is a very useful application, as it allows for devices to interpret data ranging from checks to signs to handwritten notes. Machine learning allows for very high accuracy for digit detection, and when applied to digit recognition can handle a wide range of fonts and handwriting styles. However, due to the structure of neural networks, sequential computation such as what is found in microcontrollers is not effective for a neural network. Instead, devices such as GPUs are often used due to their capacity for parallel computation. This device uses an architecture specialized just for multiplying weights and adding inputs, allowing for much faster computation for digit recognition. While on a microcontroller this system would require tens of thousands of sequential multiplications, on an ASIC it will be possible to run calculations in parallel, and implement pipelining for addition and multiplication. This is done by taking advantage of the MapReduce procedure, where in this case the mapping is applying weights to the biases, and the reduction is adding together all the inputs to a sigmoid, and then calculating the sigmoid output (which can be done by a lookup table.) Finally, this device allows for training to be done on separate hardware, which typically requires far more computational power than detection, and then allows the trained model to be uploaded on the fly using a standard interface (In this case, SPI) allowing for flexible operation.

To complete this design of a digit recognizer, the following will be needed:

- SST39LF200A (×16) Multi Purpose Flash Datasheet[2]
- 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 design proposal will include:

- A top level system usage diagram for an example configuration/usage of the digit recognizer circuit.
- The standards/algorithms to be implemented by the digit recognizer circuit.
- The design pinout of the neural network, and input output signals with their associated access codes.
- The specific design architecture to be utilized by the digit recognition circuit.

2 Design Specifications

2.1 System Usage

2.1.1 System Usage Diagram

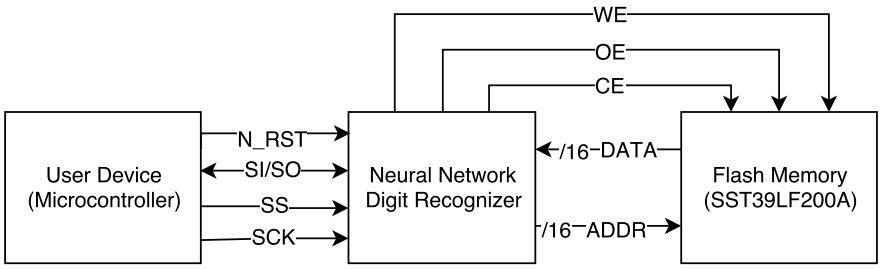


Figure 1: Example system usage diagram of Neural Network Digit Recognizer

This design is intended to accelerate the recognition of digits by implementing a neural network in an ASIC, allowing for parallel computation. However, the functionality can only be useful when interfaced with a general purpose computing device. The application depicted in the system usage diagram shows the recognizer interfacing with a microcontroller via SPI. An additional necessary module is flash memory, for which an SST39LF200A flash memory IC[2] is used. This is used to store the weights and biases for the neural network, and is used to prevent the need to re-upload the model every time. If the model needs to be changed, then it can be uploaded directly to the flash memory using an external chip. The microcontroller in this example would be used as a control device. While the majority of the work being done is accomplished within the Digit Recognizer, the control functionality comes from the user device performing the following tasks:

1. Enabling the chip using the SS, the SPI register will choose which operation to be done. The four operations are:
   (a) Let the digit recognizer know that image data will be sent over MOSI.
   (b) Ask the digit recognizer for the result of the previous operation.
   (c) Ask the digit recognizer for the cost of the previous operation.
   (d) Let the digit recognizer know the expected label[1] of each image (needed for calculating the cost[2] output)

[1]Label is defined as the expected output digit value for a given image.
[2]Cost is defined as the mean squared error between the expected and calculated labels for a set of digit images.

The normal usage for this system is depicted in Figure 2. For regular use, the user would only be going through the left branch of the chart, as this is what is used for digit recognition. However, if used for training, the right branch can be used, allowing for cost analysis and model modification.

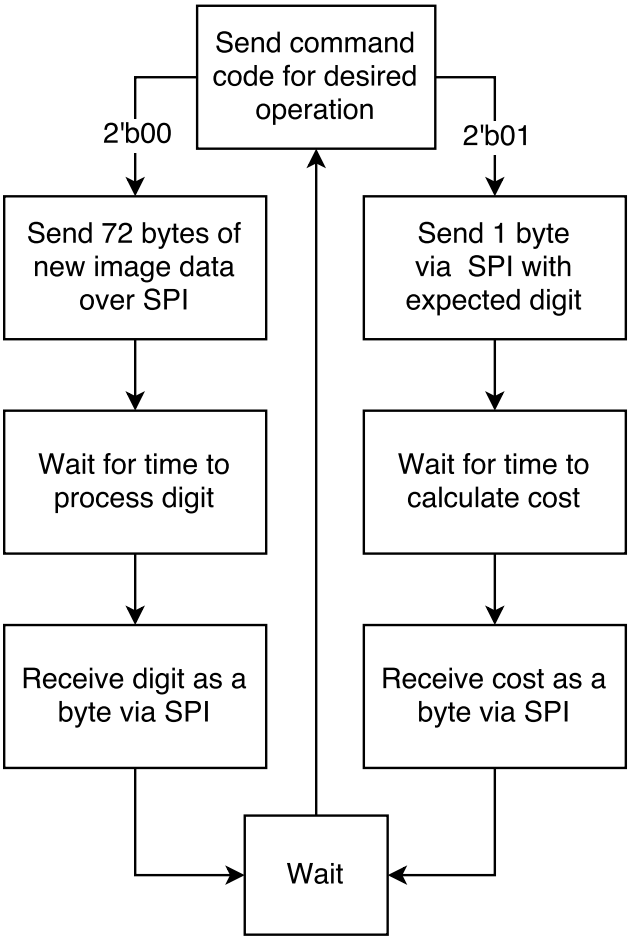


Figure 2: Normal usage flow diagram

2.1.2 Implemented Standards and Algorithms

SPI:

- Synchronous serial communication between the user device and the neural network digit recognizer.
- MOSI/MISO compatible with external device (such as a microcontroller).

Flash Memory:

- Permanent storage memory for bias/weight data.
- Communicates with a 16 bit address bus, a 16 bit data bus, and read,write, and chip enable lines.

Pipelining:

- "Pre-Processing", once values from a register are used, those values are updated with the next value to be utilized, if a calculation requires multiple cycles.
- Allows the same arithmetic unit to be used for multiple calculations at once.
- Staging is controlled by a network controller block, allowing for pipelining in the sigmoid ALU block.

ALU:

- Fixed Point Addition
  - 10 bit addition used for adding four 8 bit weighted inputs together.
  - 16 bit addition used for accumulating all weighted inputs.
  - 12 bit addition used for adding bias to summed weighted inputs
- Fixed Point Multiplication.
  - Used for multiplying 4 bit weights and 4 bit inputs
- Sigmoid Function.
  - Used for the output of the sigmoid neuron registers, calculated using a lookup table, and has 4 bit precision.

Signals:

Command Code	Description
`2'b00`	New image data will be sent over MOSI
`2'b01`	A new expected label will be sent over MOSI

Table 1: Device instruction bits for SPI

2.2 Design Pinout

Signal Name	Direction	# Bits	Description
PWR	power	N/A	System Power
GND	ground	N/A	System
CLK	in	1	System Clock
N_RST	in	1	Active low reset signal

Table 2: Miscellaneous Pinouts

Signal Name	Direction	# Bits	Description
MISO	out	1	Master in slave out signal
MOSI	in	1	Master out slave in signal
SCK	in	1	SPI clock
SS	out	1	Slave select Signal

Table 3: Pinouts to the user device

Signal Name	Direction	# Bits	Description
WE	out	1	Write Enable Signal
OE	out	1	Output Enable Signal
CE	out	1	Chip Enable Signal
DATA	in	16	Bias/weight data from the external flash memory
ADDR	out	16	Address for flash memory

Table 4: Pinouts to the flash memory

The signals specified in Table 2 represent any signals necessary for the chip to be powered and clocked. These are not labeled in the design architecture. Next, the pins in Table 3 are the signals that are exposed to any device that will use the neural network for detecting digits. Finally, the pins in Table 4 are the signals that are necessary for the neural network to interface with its flash memory.

2.3 Operational Characteristics

2.3.1 Sigmoid Neuron Computations

This systems ALU performs the necessary computations for a sigmoid neuron, visually depicted below in Figure 3. This takes several different stages, in which the inputs to the neuron are multiplied by a weight, and then summed. This resulting value is compared to the bias for the neuron, and the difference is outputted into a sigmoid function. In the figure there are k inputs and weights, represented by x_i and w_i. Additionally, the bias is b.

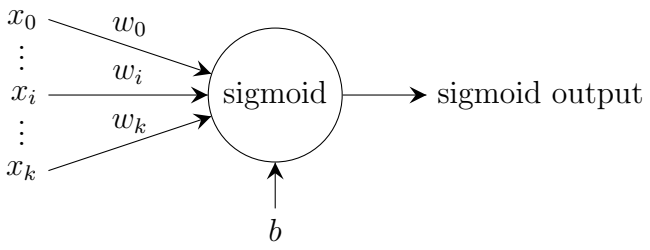


Figure 3: A top-level represention of a sigmoid neuron

Before using the sigmoid function, the output of the neuron is be represented as

$$z = \sum_i w_i x_i + b \tag{1}$$

where i is an index of an input to the sigmoid, and w_i is a weight to an input, and x_i is the pre-weighted value of the input. b is the bias for a particular sigmoid neuron.

The sigmoid function is defined as

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{2}$$

Where z is the output of the neuron before going through the sigmoid function. Note that internally the sigmoid function is computed by a lookup table with 4 bit precision.

2.3.2 Cost Output Computations

If this device is to be used with an external module for training, the cost (or effectiveness) of the current model can be requested if the device is given an expected output for each digit passed to it. This cost function, C, is defined [3] as

$$C(w, b) \equiv |y(x) - a|^2 \tag{3}$$

In this definition, w represents the current weights used for the network, and b represents the biases. x represents a single detected digit for an image, so this cost is calculated over multiple tests. a is a vector of outputs from the outer layer of the neural network, notated as

$$a = (c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9) \tag{4}$$

Where c_i is the level of confidence that a digit is i, encoded as an 4-bit number with 1 integer and 3 fractional bits..

$y(x)$ is a function for the expected output of the outer later of the network, and represented as a 10-value vector as is a, except the correct digit has a value of 1, while an incorrect value is 0. $|u|$ is the magnitude of a vector u.

2.3.3 Neural Network Structure

The general structure of the neural network in the device is depicted in Figure 4. The input layer simply contains the brightness values of the pixels of the 12×12 px. images. Due to the simplicity of this problem, there is one hidden layer with 8 sigmoid neurons, followed by an output layer of 10 sigmoid neurons, each representing a possible digit. The reason that 10 output neurons are used instead of encoding the result in 4 using binary is because this orientation is more effectively structured for a neural network[3] as the outputs are directly mapped to a detected digit. To find the final result, the index of the output neuron with the greatest value is found, and is used for the detected digit value. Internally, the input layer of the network is stored in the Pixel Data Registers, while the hidden and output layer are stored in the Sigmoid Registers. The maximum block at the end is part of the computations done by the Detected Digit Block. While this diagram depicts the system as a parallel combinational system, internally the computations are partially serialized using the Network Controller, with computations done by the Sigmoid ALU in order to reduce circuit area.

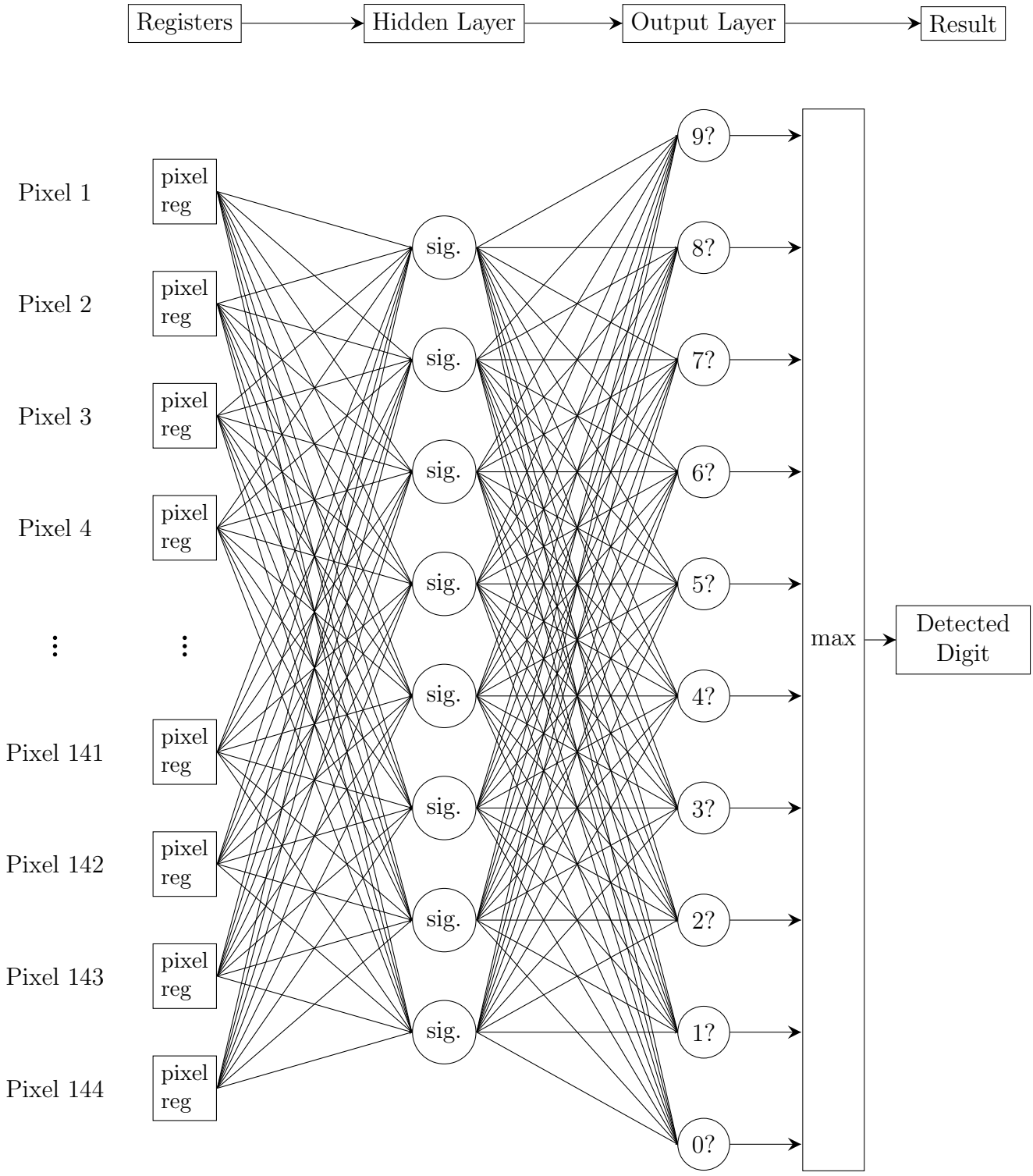


Figure 4: The structure of the internal neural network

2.3.4 Flash Memory Timing Requirements

The following timing parameters are obtained from the datasheet of the SST39LF200A flash memory IC.[2]

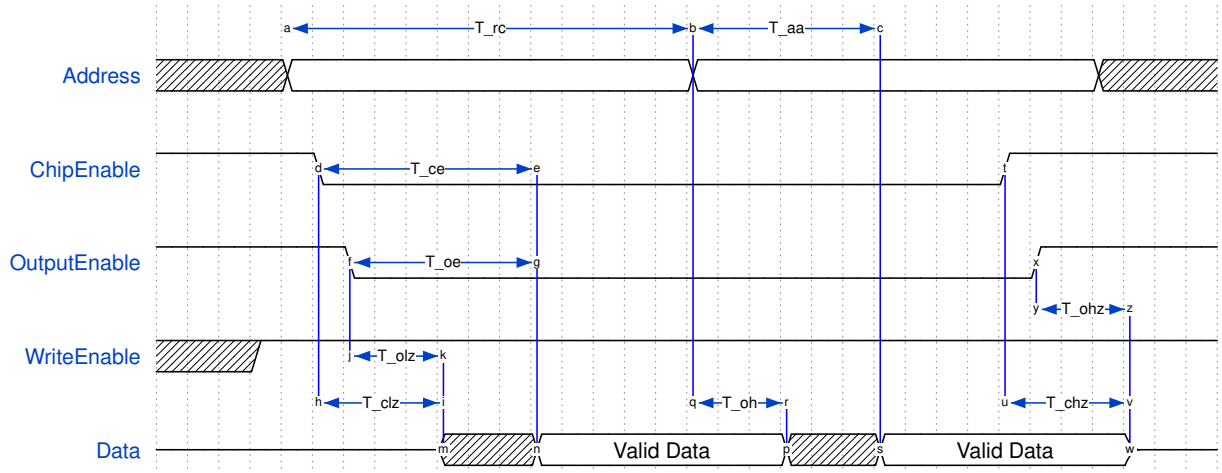


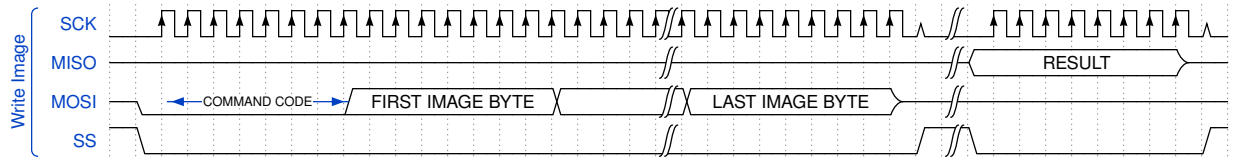
Figure 5: Flash memory timing diagram

Symbol	Parameter	Min	Max	Units
T_{RC}	Read Cycle Time	55		ns
T_{CE}	Chip Enable (CE) Access Time		55	ns
T_{AA}	Address Access Time		55	ns
T_{OE}	Output Enable (OE) Access Time		30	ns
T_{CLZ}	CE Low to Active Output	0		ns
T_{OLZ}	OE Low to Active Output	0		ns
T_{CHZ}	CE High to High-Z Output		15	ns
T_{OHZ}	OE High to High-Z Output		15	ns
T_{OH}	Output Hold from Address Change	0		ns

Table 5: Flash memory timing parameters

The operation of the flash memory is controlled by Chip Enable (CE) and Output Enable (OE), both have to be low for the system to obtain data from the outputs. CE is used for device selection. When CE is high, the chip is deselected and only standby power is consumed. OE is the output control and is used to gate data from the output pins. The data bus is in high impedance state when either CE or OE is high. Each read cycle gives a 16-bit data word which contains four weight/bias nibbles.

2.3.5 SPI Timing Requirements



(a) Timing diagram for writing image

(b) Timing diagram for expected writing label

Figure 6: SPI timing diagrams

To write an image for an operation to be performed on into the circuit, first the command code `2'b00` must be written through the SPI MOSI. immediately after the command code the first byte of the image is written, followed by the second, followed by the third etc. until all 72 bytes have been written. Note that each image byte contains data for two pixels. Then the user must wait a defined amount of time before the result byte can be read out from the SPI MISO. the SS is low whenever the SCK is running.

To write an expected label for the current image on into the circuit, first the command code `2'b01` must be written through the SPI MOSI. immediately after the command code the label value in binary is input. Then the user must wait a defined amount of time before the cost byte can be read out from the SPI MISO. the SS is low whenever the SCK is running.

2.4 Requirements

2.4.1 Summary

The digit recognizer in its current state was almost exclusively optimized for speed by allowing for parallel computation and multiplication of the image data by each respective weight and bias. Other design choices were made when creating the initial layout of the digit recognizer, of which most applied to the speed of the circuit. For one, all expensive operations, i.e the sigmoid function, will be executed by table lookup and linear interpolation rather than an ALU that would perform this operation sequentially. While this would require more space on the chip to store the data necessary to perform table lookup, it will reduce the amount of bottlenecks in the computational logic of the digit recognizer. Another method for the reduction of bottlenecks and the reduction of area used is within the actual image data being sent. SPI will be used not only for its simplicity but so that the image data can be clocked in at a higher rate (12 MHz). The image data itself will consist of 12×12 pixel grayscale images with each pixel being represented by a byte. Standard MNIST images are originally 28×28 pixels. However, in an attempt to optimize space and reduce the number of flip flops in the design the number of pixels was reduced to just 1/4 of the original MNIST standard images. This will require more overhead in order to obtain weights and biases, but will speed up the digit recognizer and require less space. The main bottleneck of our circuit will be sending in data via SPI to the SPI controller. While the intention is to use a 12 MHz SPI clock which will allow us to have a baud rate of 12 Mb/s, including all the time for computation leaves us with the following:

$$\text{Calculation time} = \underbrace{\left(\frac{12\cdot12\cdot4 \text{ bits/image}}{12\cdot10^6 \text{ bits/second}}\right)}_{\text{Time to transfer image}} + \underbrace{\left(\frac{(144\cdot8+8\cdot10) \text{ connections/image}}{\frac{10^9\cdot4}{55\cdot}\text{ connections/second}}\right)}_{\text{Time to compute sigmoids}} + 500\cdot10^{-9}\text{seconds} \quad (5)$$

$$\approx 66\cdot10^{-6}\frac{\text{seconds}}{\text{image}} \quad (6)$$

$$\text{Calculation rate} = \frac{1}{\text{Calculation time}} \approx 15,000\frac{\text{images}}{\text{second}} \quad (7)$$

The system clock that we will use will be 200MHz. This is because it is a clock speed that is reasonable to achieve, and will be able to keep up to the external bottlenecks for our design. The main bottleneck of our design will come from the flash memory and the communication between the SRAM and the Staging Registers used in loading the weights and biases that will be utilized to calculate the sigmoid output and ultimately the result of which handwritten digit was sent in to the circuit.

The read cycle time, T_{RC}, of the flash memory is 55 ns [2] which is for each 16-bit word accessed. Therefore it is necessary to optimize our design so that this will be the only bottleneck of the operation.

2.4.2 Network Requirements

Our target area for our design was derived from tables below. Most of the area comes from the Pixel Data Register and the Sigmoid Registers as these blocks need a lot of space to internally store all the data needed for the design. To reduce area usage of these blocks, the registers do not have a reset, and instead of holding all 24 by 24 8 bit pixels from the original image data, we reduced the image size to 12 by 12 4 bit pixels.

Many of the design choices such as the one mentioned were based on results obtained by running a python script to optimize how many bits would be used, the number of hidden layers, and the size of the incoming image from the MNIST database. This script would generate networks designed for images of size 1×1 pixels to the original 28×28 pixels. It also tested a hidden layer with 1-20 hidden neurons. To generate the model it first used 32 bit floating point math for training (both running feedforward and gradient descent). However, to then test the accuracy in the Digit Recognizer ASIC, it would then truncate the weights and biases to 4 bit values, as well as replace the sigmoid function with one equivalent to the multiplexer used in the design. The network would then run 10,000 images through the modified model, and report the results. Each test was run twice, and the maximum was chosen, as sometimes a low result was obtained due to poor starting values for the weights/biases of the network.

The heat map shown in Table 6 indicates the number of handwritten digits that were correctly identified when using a 1-bit sigmoid (*e.g.* a perceptron). The heat map was created by running through 4 epochs when varying the number of neurons in the hidden layer and the number of pixels on a side of the input image. The number in each cell represents the number correct out of 10000.

These results are important because they highlight that the system is able to run without high precision in its internal variables. While this shows the test with higher precision calculations, a test that better represents our design is shown in Table 7. Furthering our optimization we decided to assign each neuron/pixel pair an efficiency value. To do this it became necessary to create a table of the speed and area that each pair would take. Table 8 shows the connections between each neuron. As a last step the efficiency is calculated in Table 9 and Table 10 as the number of digits correctly recognized per the neural connections. An important thing to note about these calculations is that the lower numbers of neural connections are more efficient due the little area that these pairs consume.

HIDDEN NEURONS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
20	1716	1898	3376	3693	5066	6020	6817	7627	8111	7519	8622	8610	8883	8863	9004	9068	9034	9012	9064	9041	9103	9082	9020	9104	9103	9113	9099	9103
19	1703	2092	3360	3827	5104	6099	6933	7707	8269	8414	8562	8652	8929	8905	8992	9032	8962	8970	9014	8280	9108	9009	9109	9049	8957	9047	9120	8938
18	1764	2031	3226	3489	4993	5950	7032	7493	8215	8115	8495	8593	8591	8871	8913	8905	8970	9060	9021	9045	8928	9125	9017	9068	8988	9031	9045	8968
17	1720	1996	3090	3019	5109	6092	7092	7544	8131	8085	8526	7848	8697	8903	8967	8981	8923	9017	8929	8992	9085	9090	8953	9040	9046	9016	9062	9104
16	2040	2138	3272	3705	5045	5942	7060	7327	8043	8349	8474	8661	8823	8791	8894	8996	8938	8904	9030	9043	8998	9028	9012	9040	8953	9054	8976	9071
15	1756	1959	3335	3463	5441	6034	6572	7537	8123	8175	8472	8572	8720	8861	9105	8957	8888	8951	8972	8901	9059	8925	8984	9000	9024	9087	8955	8913
14	2040	2066	3382	3510	5059	6049	7038	7500	8043	8303	8351	8582	8500	8765	8735	9023	8977	8807	9035	8913	8984	8946	8865	8965	8952	9022	8945	9021
13	1756	2007	3195	3588	5153	6046	6807	7356	8031	8124	8324	8444	8642	8796	8748	8825	8960	8836	8911	8930	8911	8809	8951	8919	8924	8823	9002	8936
12	2040	1833	3185	3590	5049	5836	6872	7311	7948	8156	8393	8618	8685	8641	8615	8734	8905	8922	8912	8891	8926	8888	8983	8926	8927	8984	8994	8860
11	1703	1958	3236	3735	5096	5759	6875	7341	7897	7999	8438	8465	8672	8722	8592	8688	8844	8917	8821	8877	8617	8930	8918	8924	8791	8920	8838	8868
10	1603	1859	3380	3344	5209	5726	6710	7352	7958	7856	8328	8093	8591	8651	8762	8659	8705	8727	8856	8865	8712	8762	8879	8817	8939	8892	8843	8864
9	1764	2040	3336	3689	5050	5882	6613	7187	7927	7749	8199	8474	8575	8636	8552	8627	8650	8712	8769	8760	8791	8799	8833	8803	8814	8952	8774	8830
8	1720	2105	2867	3502	5026	5680	6794	7035	7746	7814	8126	8135	8208	8610	8636	8673	8654	8650	8524	8344	8688	8710	8601	8778	8827	8704	8642	8784
7	2040	1948	3201	3603	5096	5605	6573	6936	7530	7577	8036	8233	8187	8184	8252	8619	8309	8496	8692	8178	8730	8619	8489	8191	8737	8502	8342	8485
6	1711	1754	3048	3484	5120	5485	6200	6729	7621	7282	7198	7948	8042	8064	8418	7918	8304	8335	8424	8144	8332	8294	8441	7876	6128	7361	8635	8690
5	2032	1750	2701	3625	4776	5310	6357	6434	7157	7065	7501	6267	7667	5940	7980	7599	7308	7368	6681	8295	7544	7990	7501	7227	7877	7826	8260	8028
4	1716	1944	3053	3542	4536	5091	5587	6312	6120	6698	6812	6657	6673	6180	6158	6295	6721	7643	6090	6687	5782	7297	5861	6242	5328	7222	6779	6372
3	1444	2152	2455	3479	4147	4138	5245	4904	4941	4427	6217	3027	5669	5895	5826	4912	3683	6489	4504	4117	3652	3896	4687	3802	4896	5911	4444	5700
2	1764	1932	2555	3112	3256	3397	3873	3994	3756	3656	4249	3446	3089	3872	3633	3615	3639	2199	2972	3037	4053	3007	2953	3673	3014	2969	2746	3146
1	1703	1766	2069	1814	2288	1815	1961	1947	2534	2257	2048	2567	1930	2078	2000	1135	1550	2628	2625	2644	2073	1976	1903	1805	2475	1904	2092	1964

IMAGE WIDTH (PIXELS)

Table 6: Results when ran with a 1 bit sigmoid (perceptron)

HIDDEN NEURONS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
20	2025	2471	3677	3920	5449	6557	7653	7832	8731	8347	8800	8934	8840	9043	8968	9044	8952	8882	9069	8952	9065	8934	9051	9046	9020	8995	8896	8985
19	2025	2695	3604	3491	5718	6097	7712	8010	8576	8411	8482	8671	8760	8931	8972	8835	8827	8924	9017	8931	8999	8872	8967	9029	8917	8922	9010	9006
18	2025	2308	3452	3922	5731	6532	7766	7552	8508	8618	8394	8874	8816	8954	9024	8913	8978	9015	8855	9069	8962	8932	9007	9058	8981	9008	9013	9027
17	2025	2477	3312	3716	5563	6198	7571	8188	8662	8417	8785	8792	8784	8914	9057	8934	8687	8833	8938	8855	8870	8683	8860	8935	8996	8868	8905	8957
16	2025	2568	3076	3681	5329	6238	7516	7912	8559	8097	8545	8787	8610	8925	8869	9035	8871	8970	8993	8816	8904	8898	9056	8978	8979	8527	8840	9042
15	1956	2417	3359	3616	5588	6200	7923	7947	8438	8420	8351	8737	8678	8960	8784	8771	8869	8961	8749	8908	8903	8942	8810	8888	9028	8921	8887	8930
14	2025	2509	3296	3884	5631	6234	7579	7816	8485	8515	8350	8915	9013	8947	8745	8836	8981	8980	8862	8716	8975	8881	9055	9012	8920	8921	8940	8880
13	2025	2498	3483	3935	5712	6189	7345	7539	8435	8272	8420	8520	8639	8893	8796	8785	8697	8758	8851	8881	8654	8840	8869	8869	8681	8893	8951	8932
12	2025	2378	3072	3811	5101	5906	7306	7496	8446	8286	8592	8490	8848	8696	8755	8480	8770	8642	8456	8719	8704	8783	8895	8620	8841	8907	8811	8917
11	1956	2279	3708	3683	5767	6379	7555	7488	8023	8077	8395	8581	8618	8779	8720	8808	8812	8779	8596	8935	8847	8741	8768	8904	8588	8902	8846	8718
10	2025	2105	3373	3685	5807	6257	7457	7455	8101	7881	8337	8235	8405	8750	8119	8595	8524	8631	8557	8342	8779	8834	8778	8839	8772	8691	8771	8774
9	1803	2522	3603	3890	5670	5540	7568	6955	7269	8421	7693	8460	8496	8751	8585	8644	8419	8538	8566	8160	8713	8645	8535	8811	8930	8911	8749	8730
8	2025	2534	3616	3793	5651	5770	7556	7410	7915	7998	8382	8575	8234	8605	8566	8277	8718	8424	8120	8463	8711	8626	8684	7933	8520	8641	8352	8553
7	2025	2471	3295	3908	5596	5408	6804	6686	7557	7626	8276	8194	7566	8556	8405	8406	8700	8569	8482	8420	8370	8383	8251	8173	8328	8632	8046	8685
6	2025	2117	2847	3667	5246	4850	5914	6980	7870	7782	6992	7779	7599	5897	8217	8715	7760	6770	8158	7655	8317	8452	8002	8203	8205	8389	8513	8411
5	2025	2392	3443	3450	5055	5073	5915	6632	7151	6151	6754	7647	7244	7574	7922	7780	7602	6785	7377	6960	7653	7725	7222	7961	6316	7396	7428	7689
4	2025	2433	3158	3227	5046	4250	5609	5336	5405	6522	5009	6518	7154	6995	5868	6604	7550	5556	6811	6261	5293	5530	6490	7449	5498	7168	5504	6343
3	2025	2592	2547	3429	4508	4140	5736	5082	5401	5427	5481	4652	4632	4687	5394	4543	4786	4483	5636	5356	5444	5428	5563	6075	6007	4229	6061	3840
2	2079	2087	2487	2842	3686	3363	3576	3325	3915	3117	3093	3109	3016	3882	3854	3816	3498	3816	3935	3048	3713	3817	3018	2867	3871	2116	2906	2974
1	1956	1886	2076	2165	2126	1823	2085	2058	2092	2075	2088	2114	2081	2096	2087	2102	1982	2112	2097	2106	2094	2037	2126	2094	1962	2081	2085	2039

IMAGE WIDTH (PIXELS)

Table 7: Results when ran with 4 bit sigmoid and weights

The data in Table 6 and Table 7 seem to show that the networks perform at almost the same accuracy levels. However, there are several reasons to choose the higher precision network. The first is that the data used for training and testing the network is clean, but a 1 bit network would be much more susceptible to noise. Second, the 4 bit network is much more effective at training, as the cost values (computed in Equation 3) have a cleaner derivative. Thus, using 4 bit values allows the system to be hooked up to an system for training.

HIDDEN NEURONS	IMAGE WIDTH (PIXELS)																											
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
20	220	280	380	520	700	920	1180	1480	1820	2200	2620	3080	3580	4120	4700	5320	5980	6680	7420	8200	9020	9880	10780	11720	12700	13720	14780	15880
19	209	266	361	494	665	874	1121	1406	1729	2090	2489	2926	3401	3914	4465	5054	5681	6346	7049	7790	8569	9386	10241	11134	12065	13034	14041	15086
18	198	252	342	468	630	828	1062	1332	1638	1980	2358	2772	3222	3708	4230	4788	5382	6012	6678	7380	8118	8892	9702	10548	11430	12348	13302	14292
17	187	238	323	442	595	782	1003	1258	1547	1870	2227	2618	3043	3502	3995	4522	5083	5678	6307	6970	7667	8398	9163	9962	10795	11662	12563	13498
16	176	224	304	416	560	736	944	1184	1456	1760	2096	2464	2864	3296	3760	4256	4784	5344	5936	6560	7216	7904	8624	9376	10160	10976	11824	12704
15	165	210	285	390	525	690	885	1110	1365	1650	1965	2310	2685	3090	3525	3990	4485	5010	5565	6150	6765	7410	8085	8790	9525	10290	11085	11910
14	154	196	266	364	490	644	826	1036	1274	1540	1834	2156	2506	2884	3290	3724	4186	4676	5194	5740	6314	6916	7546	8204	8890	9604	10346	11116
13	143	182	247	338	455	598	767	962	1183	1430	1703	2002	2327	2678	3055	3458	3887	4342	4823	5330	5863	6422	7007	7618	8255	8918	9607	10322
12	132	168	228	312	420	552	708	888	1092	1320	1572	1848	2148	2472	2820	3192	3588	4008	4452	4920	5412	5928	6468	7032	7620	8232	8868	9528
11	121	154	209	286	385	506	649	814	1001	1210	1441	1694	1969	2266	2585	2926	3289	3674	4081	4510	4961	5434	5929	6446	6985	7546	8129	8734
10	110	140	190	260	350	460	590	740	910	1100	1310	1540	1790	2060	2350	2660	2990	3340	3710	4100	4510	4940	5390	5860	6350	6860	7390	7940
9	99	126	171	234	315	414	531	666	819	990	1179	1386	1611	1854	2115	2394	2691	3006	3339	3690	4059	4446	4851	5274	5715	6174	6651	7146
8	88	112	152	208	280	368	472	592	728	880	1048	1232	1432	1648	1880	2128	2392	2672	2968	3280	3608	3952	4312	4688	5080	5488	5912	6352
7	77	98	133	182	245	322	413	518	637	770	917	1078	1253	1442	1645	1862	2093	2338	2597	2870	3157	3458	3773	4102	4445	4802	5173	5558
6	66	84	114	156	210	276	354	444	546	660	786	924	1074	1236	1410	1596	1794	2004	2226	2460	2706	2964	3234	3516	3810	4116	4434	4764
5	55	70	95	130	175	230	295	370	455	550	655	770	895	1030	1175	1330	1495	1670	1855	2050	2255	2470	2695	2930	3175	3430	3695	3970
4	44	56	76	104	140	184	236	296	364	440	524	616	716	824	940	1064	1196	1336	1484	1640	1804	1976	2156	2344	2540	2744	2956	3176
3	33	42	57	78	105	138	177	222	273	330	393	462	537	618	705	798	897	1002	1113	1230	1353	1482	1617	1758	1905	2058	2217	2382
2	22	28	38	52	70	92	118	148	182	220	262	308	358	412	470	532	598	668	742	820	902	988	1078	1172	1270	1372	1478	1588
1	11	14	19	26	35	46	59	74	91	110	131	154	179	206	235	266	299	334	371	410	451	494	539	586	635	686	739	794

Table 8: Connections based on hidden neurons and pixel base

HIDDEN NEURONS	IMAGE WIDTH (PIXELS)																											
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
20	7.80	6.78	8.88	7.10	7.24	6.54	5.78	5.15	4.46	3.42	3.29	2.80	2.48	2.15	1.92	1.70	1.51	1.35	1.22	1.10	1.01	0.92	0.84	0.78	0.72	0.66	0.62	0.57
19	8.15	7.86	9.31	7.75	7.68	6.98	6.18	5.48	4.78	4.03	3.44	2.96	2.63	2.28	2.01	1.79	1.58	1.41	1.28	1.06	1.06	0.96	0.89	0.81	0.74	0.69	0.65	0.59
18	8.91	8.06	9.43	7.46	7.93	7.19	6.62	5.63	5.02	4.10	3.60	3.10	2.67	2.39	2.11	1.86	1.67	1.51	1.35	1.23	1.10	1.03	0.93	0.86	0.79	0.73	0.68	0.63
17	9.20	8.39	9.57	6.83	8.59	7.79	7.07	6.00	5.26	4.32	3.83	3.00	2.86	2.54	2.24	1.99	1.76	1.59	1.42	1.29	1.18	1.08	0.98	0.91	0.84	0.77	0.72	0.67
16	11.59	9.54	10.76	8.91	9.01	8.07	7.48	6.19	5.52	4.74	4.04	3.52	3.08	2.67	2.37	2.11	1.87	1.67	1.52	1.38	1.25	1.14	1.04	0.96	0.88	0.82	0.76	0.71
15	10.64	9.33	11.70	8.88	10.36	8.74	7.43	6.79	5.95	4.95	4.31	3.71	3.25	2.87	2.58	2.24	1.98	1.79	1.61	1.45	1.34	1.20	1.11	1.02	0.95	0.88	0.81	0.75
14	13.25	10.54	12.71	9.64	10.32	9.39	8.52	7.24	6.31	5.39	4.55	3.98	3.39	3.04	2.66	2.42	2.14	1.88	1.74	1.55	1.42	1.29	1.17	1.09	1.01	0.94	0.86	0.81
13	12.28	11.03	12.94	10.62	11.33	10.11	8.87	7.65	6.79	5.68	4.89	4.22	3.71	3.28	2.86	2.55	2.31	2.04	1.85	1.68	1.52	1.37	1.28	1.17	1.08	0.99	0.94	0.87
12	15.45	10.91	13.97	11.51	12.02	10.57	9.71	8.23	7.28	6.18	5.34	4.66	4.04	3.50	3.05	2.74	2.48	2.23	2.00	1.81	1.65	1.50	1.39	1.27	1.17	1.09	1.01	0.93
11	14.07	12.71	15.48	13.06	13.24	11.38	10.59	9.02	7.89	6.61	5.86	5.00	4.40	3.85	3.32	2.97	2.69	2.43	2.16	1.97	1.74	1.64	1.50	1.38	1.26	1.18	1.09	1.02
10	14.57	13.28	17.79	12.86	14.88	12.45	11.37	9.94	8.75	7.14	6.36	5.26	4.80	4.20	3.73	3.26	2.91	2.61	2.39	2.16	1.93	1.77	1.65	1.50	1.41	1.30	1.20	1.12
9	17.82	16.19	19.51	15.76	16.03	14.21	12.45	10.79	9.68	7.83	6.95	6.11	5.32	4.66	4.04	3.60	3.21	2.90	2.63	2.37	2.17	1.98	1.82	1.67	1.54	1.45	1.32	1.24
8	19.55	18.79	18.86	16.84	17.95	15.43	14.39	11.88	10.64	8.88	7.75	6.60	5.73	5.22	4.59	4.08	3.62	3.24	2.87	2.54	2.41	2.20	1.99	1.87	1.74	1.59	1.46	1.38
7	26.49	19.88	24.07	19.80	20.80	17.41	15.92	13.39	11.82	9.84	8.76	7.64	6.53	5.68	5.02	4.63	3.97	3.63	3.35	2.85	2.77	2.49	2.25	2.00	1.97	1.77	1.61	1.53
6	25.92	20.88	26.74	22.33	24.38	19.87	17.51	15.16	13.96	11.03	9.16	8.60	7.49	6.52	5.97	4.96	4.63	4.16	3.78	3.31	3.08	2.80	2.61	2.24	1.61	1.79	1.95	1.82
5	36.95	25.00	28.43	27.88	27.29	23.09	21.55	17.39	15.73	12.85	11.45	8.14	8.57	5.77	6.79	5.71	4.89	4.41	3.60	4.05	3.35	3.23	2.78	2.47	2.48	2.28	2.24	2.02
4	39.00	34.71	40.17	34.06	32.40	27.67	23.67	21.32	16.81	15.22	13.00	10.81	9.32	7.50	6.55	5.92	5.62	5.72	4.10	4.08	3.21	3.69	2.72	2.66	2.10	2.63	2.29	2.01
3	43.76	51.24	43.07	44.60	39.50	29.99	29.63	22.09	18.10	13.42	15.82	6.55	10.56	9.54	8.26	6.16	4.11	6.48	4.05	3.35	2.70	2.63	2.90	2.16	2.57	2.87	2.00	2.39
2	80.18	69.00	67.24	59.85	46.51	36.92	32.82	26.99	20.64	16.62	16.22	11.19	8.63	9.40	7.73	6.80	6.09	3.29	4.01	3.70	4.49	3.04	2.74	3.13	2.37	2.16	1.86	1.98
1	154.8	126.1	108.9	69.77	65.37	39.46	33.24	26.31	27.85	20.52	15.63	16.67	10.78	10.09	8.51	4.27	5.18	7.87	7.08	6.45	4.60	4.00	3.53	3.08	3.90	2.78	2.83	2.47

Table 9: Efficiency of the perceptron

HIDDEN NEURONS	IMAGE WIDTH (PIXELS)																											
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
20	9.20	8.83	9.68	7.54	7.78	7.13	6.49	5.29	4.80	3.79	3.36	2.90	2.47	2.19	1.91	1.70	1.50	1.33	1.22	1.09	1.00	0.90	0.84	0.77	0.71	0.66	0.60	0.57
19	9.69	10.13	9.98	7.07	8.60	6.98	6.88	5.70	4.96	4.02	3.41	2.96	2.58	2.28	2.01	1.75	1.55	1.41	1.28	1.15	1.05	0.95	0.88	0.81	0.74	0.68	0.64	0.60
18	10.23	9.16	10.09	8.38	9.10	7.89	7.31	5.67	5.19	4.35	3.56	3.20	2.74	2.41	2.13	1.86	1.67	1.50	1.33	1.23	1.10	1.00	0.93	0.86	0.79	0.73	0.68	0.63
17	10.83	10.41	10.25	8.41	9.35	7.93	7.55	6.51	5.60	4.50	3.94	3.36	2.89	2.55	2.27	1.98	1.71	1.56	1.42	1.27	1.16	1.03	0.97	0.90	0.83	0.76	0.71	0.66
16	11.51	11.46	10.12	8.85	9.52	8.48	7.96	6.68	5.88	4.60	4.08	3.57	3.01	2.71	2.36	2.12	1.85	1.68	1.51	1.34	1.23	1.13	1.05	0.96	0.88	0.78	0.75	0.71
15	11.85	11.51	11.79	9.27	10.64	8.99	8.95	7.16	6.18	5.10	4.25	3.78	3.23	2.90	2.49	2.20	1.98	1.79	1.57	1.45	1.32	1.21	1.09	1.01	0.95	0.87	0.80	0.75
14	13.15	12.80	12.39	10.67	11.49	9.68	9.18	7.54	6.66	5.53	4.55	4.13	3.60	3.10	2.66	2.37	2.15	1.92	1.71	1.52	1.42	1.28	1.20	1.10	1.00	0.93	0.86	0.80
13	14.16	13.73	14.10	11.64	12.55	10.35	9.58	7.84	7.13	5.78	4.94	4.26	3.71	3.32	2.88	2.54	2.24	2.02	1.84	1.67	1.48	1.38	1.27	1.16	1.05	1.00	0.93	0.87
12	15.34	14.15	13.47	12.21	12.15	10.70	10.32	8.44	7.73	6.28	5.47	4.59	4.12	3.52	3.10	2.66	2.44	2.16	1.90	1.77	1.61	1.48	1.38	1.23	1.16	1.08	0.99	0.94
11	16.17	14.80	17.74	12.88	14.98	12.61	11.64	9.20	8.01	6.68	5.83	5.07	4.38	3.87	3.37	3.01	2.68	2.39	2.11	1.98	1.78	1.61	1.48	1.38	1.23	1.18	1.09	1.00
10	18.41	15.04	17.75	14.17	16.59	13.60	12.64	10.07	8.90	7.16	6.36	5.35	4.70	4.25	3.45	3.23	2.85	2.58	2.31	2.03	1.95	1.79	1.63	1.51	1.38	1.27	1.19	1.11
9	18.21	20.02	21.07	16.62	18.00	13.38	14.25	10.44	8.88	8.51	6.53	6.10	5.27	4.72	4.06	3.61	3.13	2.84	2.57	2.21	2.15	1.94	1.76	1.67	1.56	1.44	1.32	1.22
8	23.01	22.63	23.79	18.24	20.18	15.68	16.01	12.52	10.87	9.09	8.00	6.96	5.75	5.22	4.56	3.89	3.64	3.15	2.74	2.58	2.41	2.18	2.01	1.69	1.68	1.57	1.41	1.35
7	26.30	25.21	24.77	21.47	22.84	16.80	16.47	12.91	11.86	9.90	9.03	7.60	6.04	5.93	5.11	4.51	4.16	3.67	3.27	2.93	2.65	2.42	2.19	1.99	1.87	1.80	1.56	1.56
6	30.68	25.20	24.97	23.51	24.98	17.57	16.71	15.72	14.41	11.79	8.90	8.42	7.08	4.77	5.83	5.46	4.33	3.38	3.66	3.11	3.07	2.85	2.47	2.33	2.15	2.04	1.92	1.77
5	36.82	34.17	36.24	26.54	28.89	22.06	20.05	17.92	15.72	11.18	10.31	9.93	8.09	7.35	6.74	5.85	5.08	4.06	3.98	3.40	3.39	3.13	2.68	2.72	1.99	2.16	2.01	1.94
4	46.02	43.45	41.55	31.03	36.04	23.10	23.77	18.03	14.85	14.82	9.56	10.58	9.99	8.49	6.24	6.21	6.31	4.16	4.59	3.82	2.93	2.80	3.01	3.18	2.16	2.61	1.86	2.00
3	61.36	61.71	44.68	43.96	42.93	30.00	32.41	22.89	19.78	16.45	13.95	10.07	8.63	7.58	7.65	5.69	5.34	4.47	5.06	4.35	4.02	3.66	3.44	3.46	3.15	2.05	2.73	1.61
2	94.50	74.54	65.45	54.65	52.66	36.55	30.31	22.47	21.51	14.17	11.81	10.09	8.42	9.42	8.20	7.17	5.85	5.71	5.30	3.72	4.12	3.86	2.80	2.45	3.05	1.54	1.97	1.87
1	177.8	134.7	109.3	83.27	60.74	39.63	35.34	27.81	22.99	18.86	15.94	13.73	11.63	10.17	8.88	7.90	6.63	6.32	5.65	5.14	4.64	4.12	3.94	3.57	3.09	3.03	2.82	2.57

Table 10: Efficiency of 4 bit sigmoid and weights

2.4.3 Area Requirements

The area requirements from each of the modules is shown in Table 11. The largest modules were the Pixel Data Register and the Sigmoid Register blocks. The size of these blocks comes from the number of neurons in the hidden layer and the number of pixels on one side of the incoming image. Thus the importance of the heat maps was to try and minimize the size of these blocks. The rest of the modules fell within reason for the size. We figured the size of the Sigmoid ALU would be the third largest due to its importance within our system and therefore its area was justified. The IO pins on our chip took the largest amount of space within our total design. On the table below is the actual Area each components took up in our design. We ended up being far below our estimated area. Our biggest savings in area was the IO pins, which were estimated to take much larger area than what they actually did. Also, the combinational blocks for the pixel data register and sigmoid registers were much more optimized than we anticipated saving a large chunk of space.

Name of Block	**Gate/FF Count**	**Estimated Area (μm^2)**	**Actual Area (μm^2)**
Detected Digit	168	148,800	229,498
Network Controller	258	223,200	518,616
Flash Memory Controller	225	241,800	139,212
SPI Input Controller	167	184,200	202,174
SPI Output Controller	130	115,650	84,240
Cost Calculator	582	513,150	380,698
Sigmoid ALU	1155	895,950	442,690
Sigmoid Register	1252	1,025,400	205,846
Pixel Data Register	2017	1,888,350	1,282,713
Total Modules	5954	5,236,500	4,200,768
IO Pins	48 pins	4,861,080	2,952,000
Total Area		10,097,580	7,617,600

Table 11: Area budgeting summary

3 Functional Design Implementation

3.1 Design Architecture

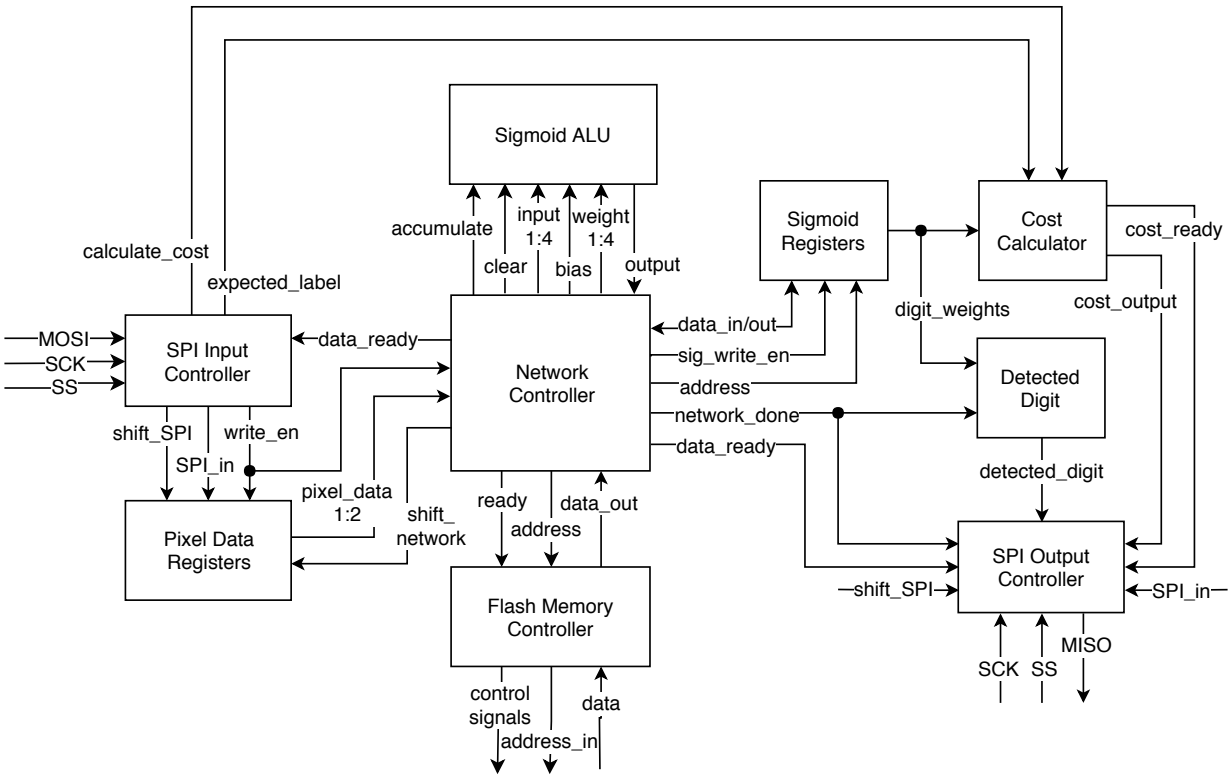


Figure 7: Digit recognition circuit design architecture

The design architecture of the digit recognition circuit is shown in Figure 7. All input first is fed to the SPI Input Controller where the data is filtered into one of a few different blocks. Each data destination block is determined by the input code. The input codes are laid out in Section 2.1.2. For the standard operating mode (output), image data is fed into the input controller and sent to a staging register used specifically for holding the data of each image. The network controller uses this data as well as data from the weight/bias staging registers (loaded from flash memory) to be input into the Sigmoid ALU to calculate probabilities for each number, *i.e.* 0-9. Depending on if the microcontroller asks for the cost analysis of the previous operation, the cost can also be output on the MISO line used for outputting the detected digit.

3.2 Functional Block Diagram

3.2.1 Generic Components

Positive Edge Detector: The positive edge detector, depicted in Figure 8, is used to detect edges on an incoming signal, and in particular the SCK line used for SPI communication. The timing for this module is depicted in Figure 9. Note that the synchronizer resets its registers to low, so the incoming signal line should be pulled low during reset. If this is not followed, the edge detector may detect a false positive edge.

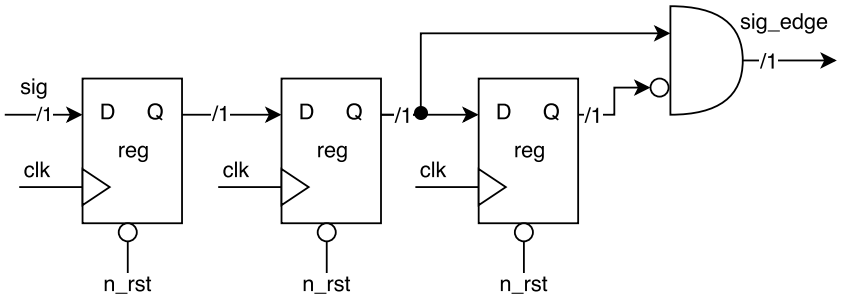


Figure 8: Positive Edge Detector with synchronization

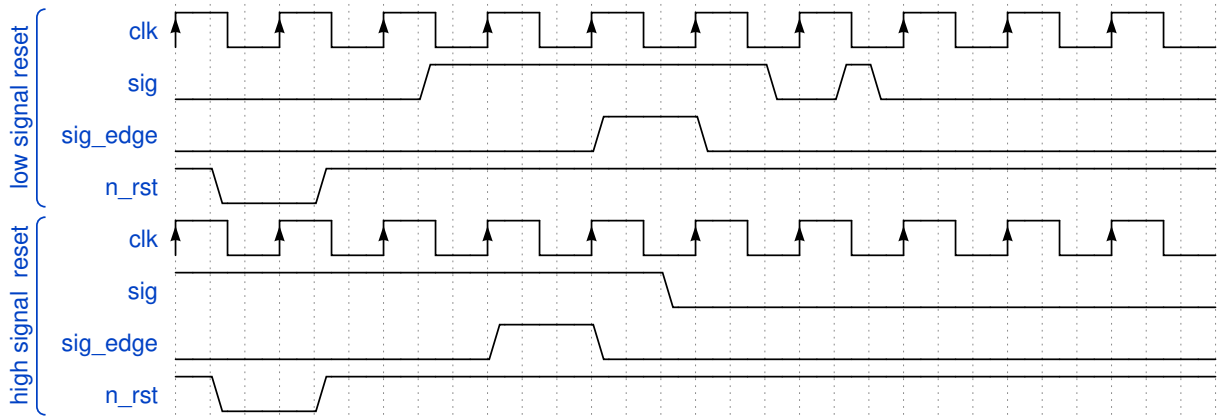


Figure 9: Positive Edge Detector timing diagram

Flexible Counter: The flexible counter, depicted in Figure 10 is a counter that can be scaled to an arbitrary number of bits. It can be dynamically controlled with a rollover value input that sets the value up to which the counter will count. It can be cleared to zero with a clear input, and will only count when enable is pulled high. To alert other components when the rollover value has been reached, it outputs high on its rollover_flag line. Note certain behaviors depicted in Figure 11: First, the counter will clear even when the enable line is disabled. Next,

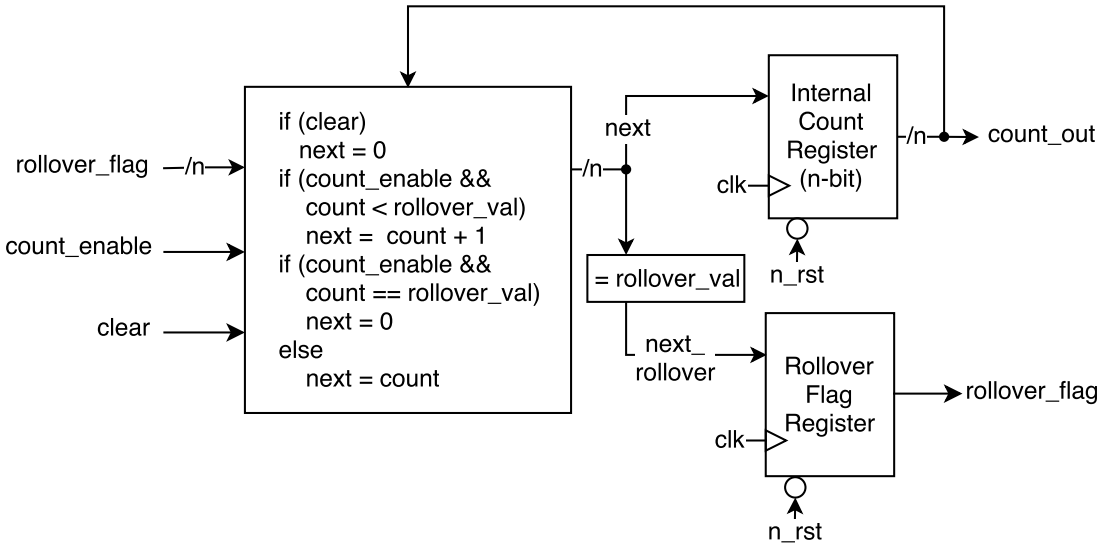


Figure 10: Flexible Counter block

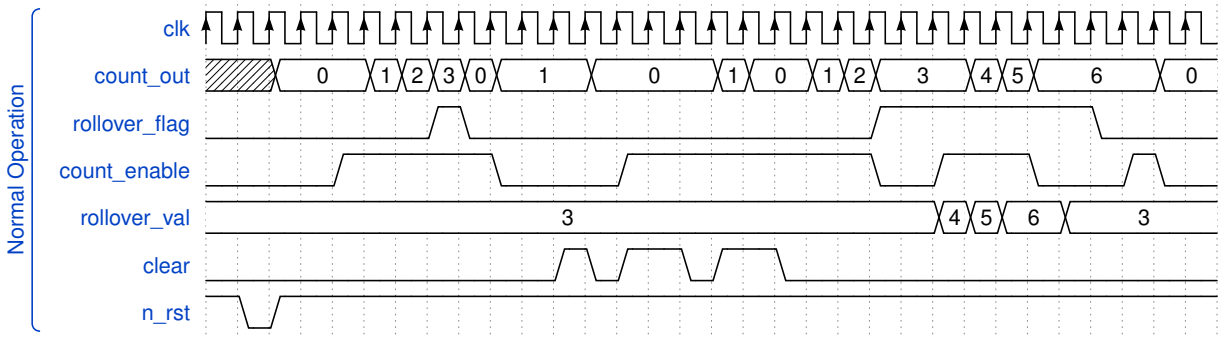


Figure 11: Flexible Counter timing diagram

Synchronizer: The synchronizer, depicted in Figure 12, ensures that all incoming signals are aligned with the system clock, in order to prevent metastability. As seen in Figure 13, the signal is delayed by two system clock cycles.

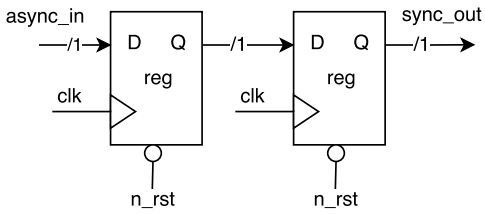


Figure 12: Synchronizer block

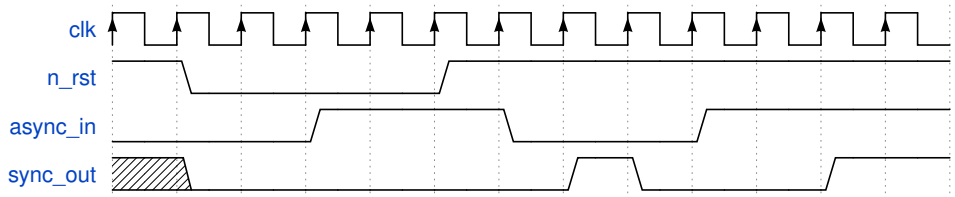


Figure 13: Synchronizer timing diagram

LSB STP Shift Register: The least-significant-bit-first serial-to-parallel shift register used for SPI communication is depicted in Figure 14.

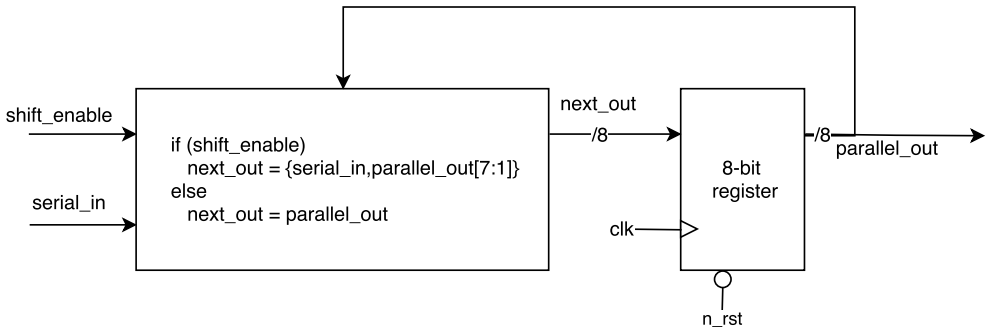


Figure 14: LSB STP shift register block

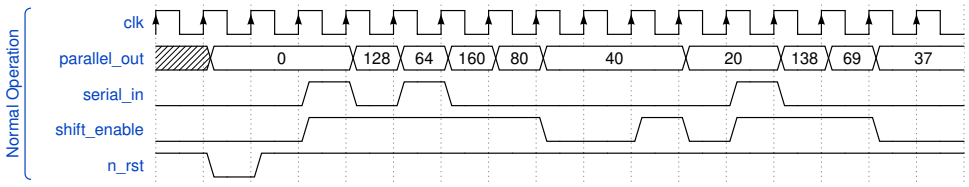


Figure 15: LSB STP Shift Register timing diagram

LSB PTS Shift Register: The least-significant-bit-first parallel-to-serial shift register used for SPI communication is depicted in Figure 16.

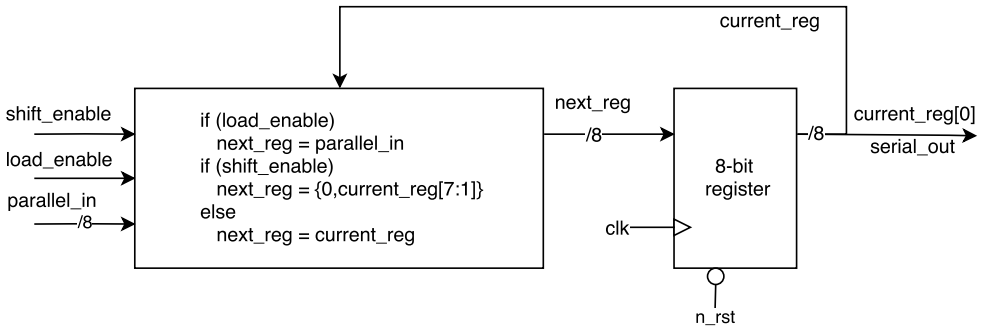


Figure 16: LSB PTS Shift Register block

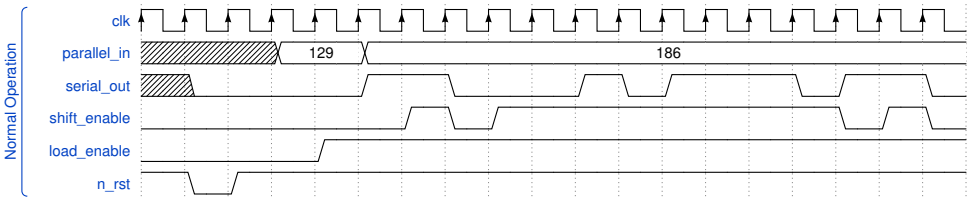


Figure 17: LSB PTS Shift Register timing diagram

3.2.2 SPI Input Controller

@TODO: fix rtl diagram

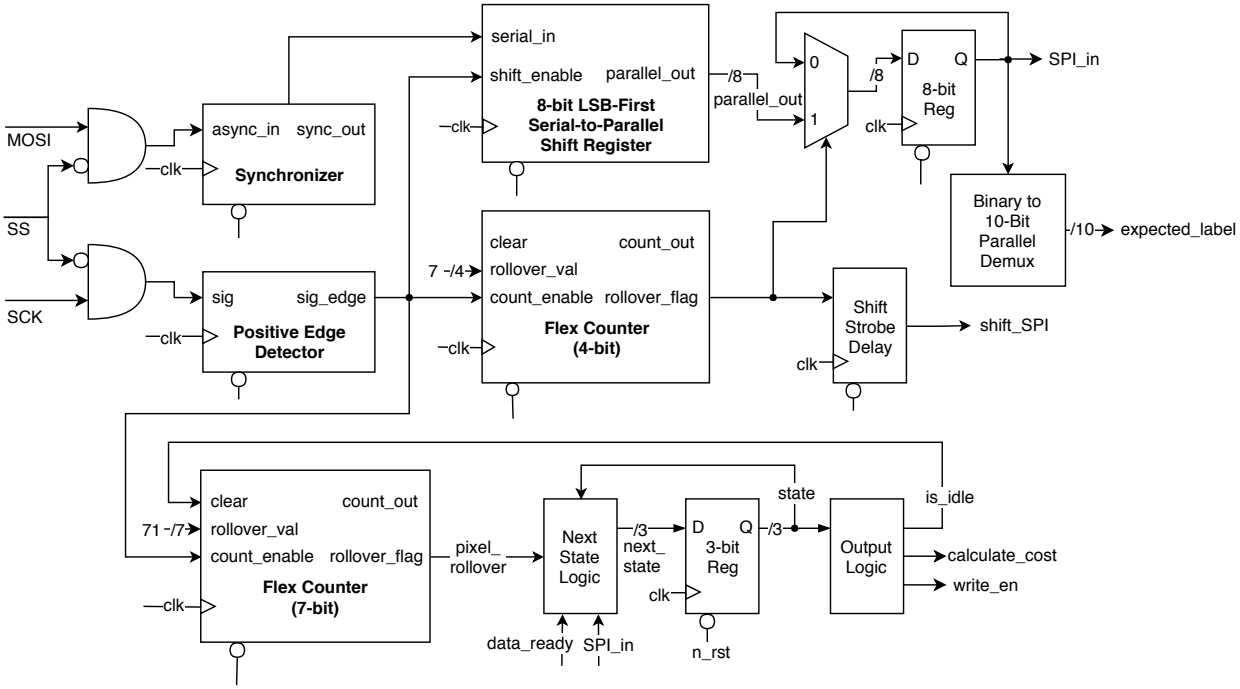


Figure 18: SPI Input Controller RTL diagram

The SPI input controller depicted in Figure 18 is used to manage data being sent into the device. It has two main functions: loading the pixel data into the network, and also loading the expected label into the cost calculator. It also stores the last byte sent via SPI into a buffer for other modules to use.

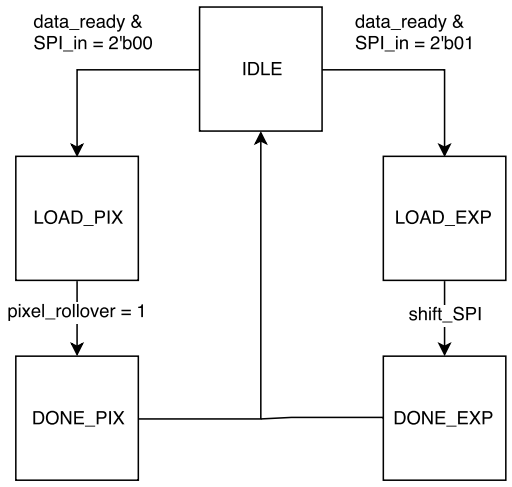


Figure 19: SPI Input Controller state transition diagram

As depicted in Figure 19, the controller can be in two main modes: Loading pixel data,

and loading label data. When loading pixel data, it will wait until all 196 pixel values are transmitted, and will not accept any other control codes.

state	is_idle	calculate_ cost	write_enable_pixel
IDLE	1	0	0
LOAD_PIX	0	0	1
DONE_PIX	0	0	0
LOAD_EXP	0	0	0
DONE_EXP	0	1	0

Table 12: SPI Input Controller state machine output logic

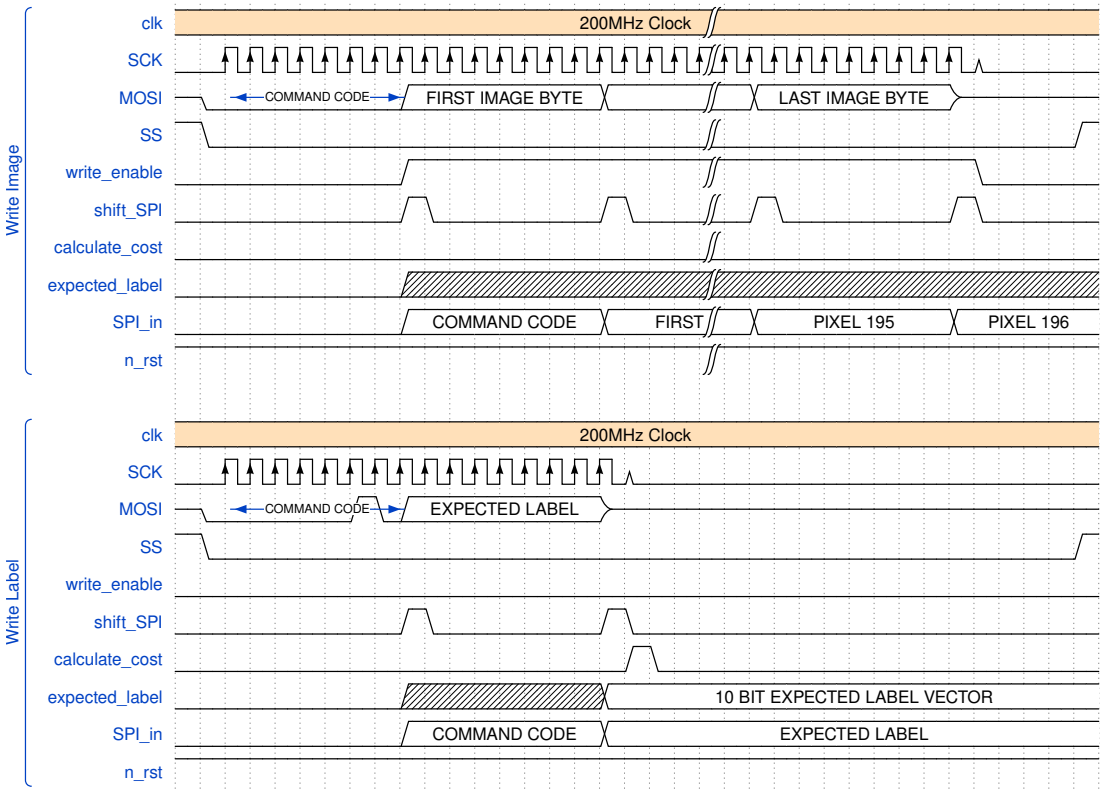


Figure 20: SPI Input Controller high level timing diagram

Depicted in Figure 20 is the timing for the SPI input controller. As this interacts with a 12MHz SPI input signal, much slower than the system clock, the timing diagram is in reference to the SPI clock and not the system clock. All the "strobe" signals in the diagram are actually held for one system clock cycle.

Name of Block	**Category**	**Gate/FF Count**	**Area (μm^2)**	**Comments**
Synchronizer	Reg. w/ Reset	2	4,800	
Pos. Edge Detector	Reg. w/ Reset	4	9,600	
4 Bit Flex Counter	Mixed	25	27,000	
8 Bit Flex Counter	Mixed	50	54,000	
2:1 Mux	Combinational	4	3,000	
Delay Register	Reg. w/ Reset	1	2,400	
3 Bit State Reg.	Reg. w/ Reset	3	7,200	
8 Bit Reg.	Reg. w/ Reset	8	10,800	
8 Bit LSB STP Shift Reg.	Mixed	40	43,200	
Binary to 10 Bit Demux	Combinational	30	22,400	
Total Estimated Area		**167**	**184,200**	
Total Actual Area			**202,174**	

Table 13: SPI Input Controller area table

The largest component of this is the 8 bit flexible counter. The justification for this is that it both requires an 8 bit register to store the value of the counter, and then an adder block for incrementing the count value, as well as multiplexers for enabling and resetting the counter. For similar reasons, the LSB STP takes the second highest number of components, as it also needs to have a register, as well as multiplexing for next state logic. The registers used need resets, as they generally send command and data signals to the rest of the circuit, making it important that their values are known. This is also true for the state registers. We went a little over on this estimate in our actual design, this was due to some extra logic we didnt think of originally to deal with generating errors from SPI when needed and some extra signals to the network controller

3.2.3 SPI Output Controller

@TODO: fix rtl diagram

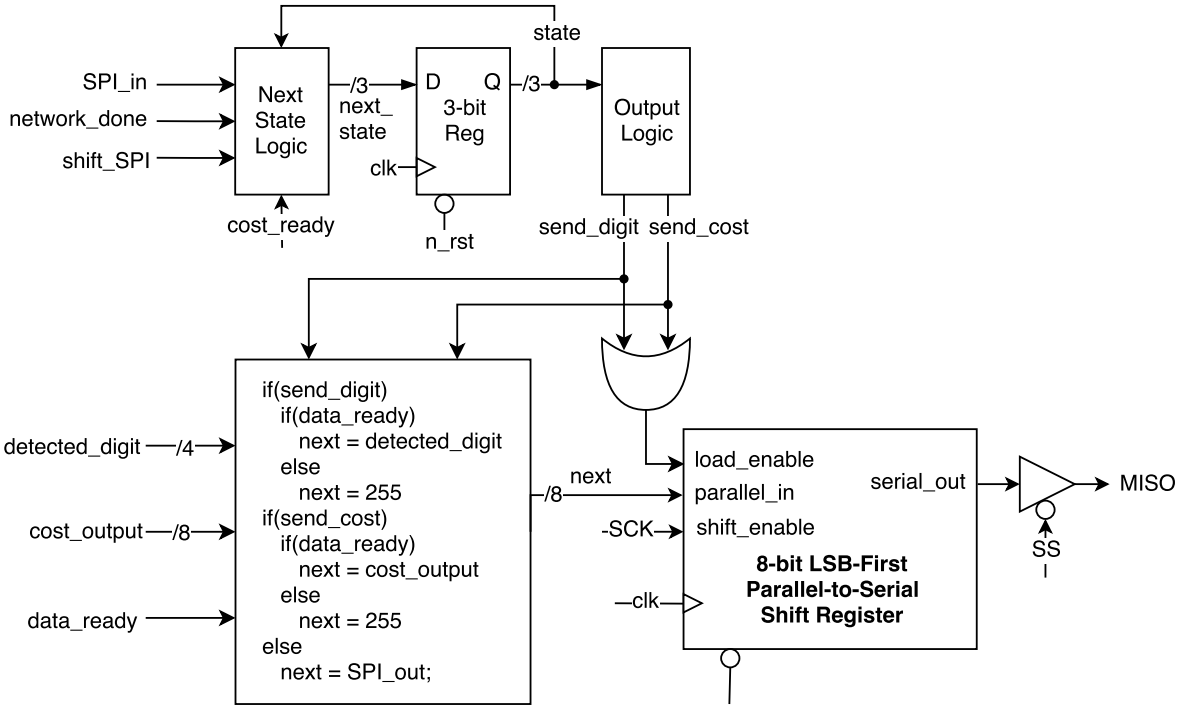


Figure 21: SPI Output Controller RTL diagram

The SPI output controller depicted in Figure 21 is responsible for sending data back to the user device. If at any point the data is not ready to be sent, it returns a value of 255. When returning a digit, this is impossible to obtain legitimately because digits can only be from 0 to 9. Similar, this is not a likely option for the cost calculation, given how it is computed. This makes it a usable error code (And if there is legitimately a 255 value for the cost output, there is something severely wrong with the model.)

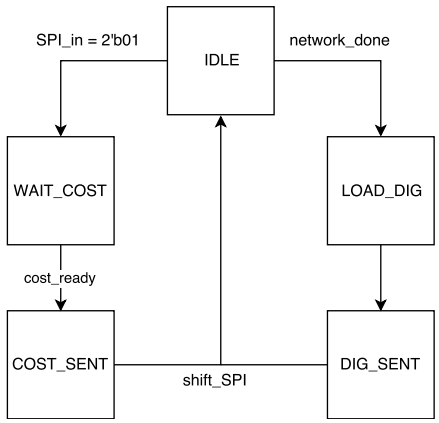


Figure 22: SPI Output Controller state transition diagram

state	send_digit	send_cost
IDLE	1	0
WAIT_COST	0	1
COST_SENT	0	1
LOAD_DIG	1	0
DIG_SENT	0	0

Table 14: SPI Output Controller state machine output logic

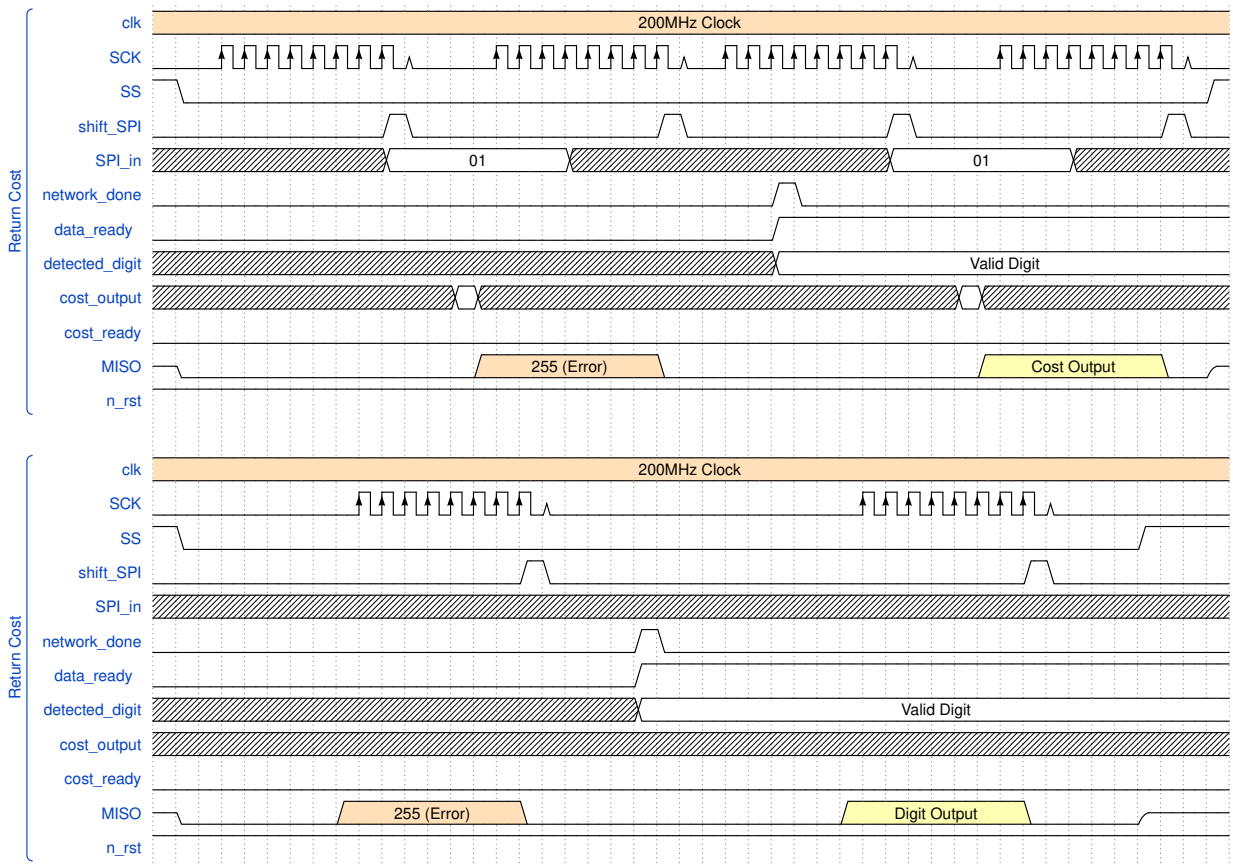


Figure 23: SPI Output Controller high level timing diagram

Depicted in Figure 23 is the timing for the SPI output controller. As this interacts with a 12MHz SPI input signal, much slower than the system clock, the timing diagram is in reference to the SPI clock and not the system clock. All the "strobe" signals in the diagram are actually held for one system clock cycle.

Name of Block	**Category**	**Gate/FF Count**	**Area (μm^2)**	**Comments**
3 Bit State Reg.	Reg. w/ Reset	3	7,200	
8 Bit LSB PTS Shift Reg.	Mixed	72	67,200	
OR Gate	Combinational	1	750	
Tri-state Buffer	Combinational	4	3,000	
Output Logic	Combinational	30	22,500	
Shifting Logic	Combinational	20	15,000	
Total Estimated Area		**130**	**115,650**	
Total Actual Area			**84,240**	

Table 15: SPI Output Controller area table

Table 15 shows the area budgeting for the SPI output controller. The largest component of the budgeting is the 8 Bit LSB PTS shift register, as this requires a register for storing the parallel input logic, as well as multiplexing for loading the input value, as well as shifting the input value. All the other components make up less than half of the area, as they are simpler combinational logic along with small registers. However, the registers still need to have a reset as they are used for a state machine.

3.2.4 Pixel Data Registers

The Pixel Data Registers is a series of 72 8 bit PTP Shift registers. These registers are responsible for storing the 4 bit value of every pixel in an image, and cycling through and outputting each pixel value as needed by the network controller. Two pixels are stored in each 8 bit register. The last four pixels will are outputted in parallel, to be used by the Network Controller. In the timing diagram, parallel-out0 refers to the output of the first Shift Register, feeding into the second Shift Register. This block is estimated to take up the most area, due to the fact that it needs to store 576 bits of data in internal registers. This block is designed with shift registers instead of addressable registers because addressable registers take up more space and are not needed for this register array. The block uses 72 PTPSRs.

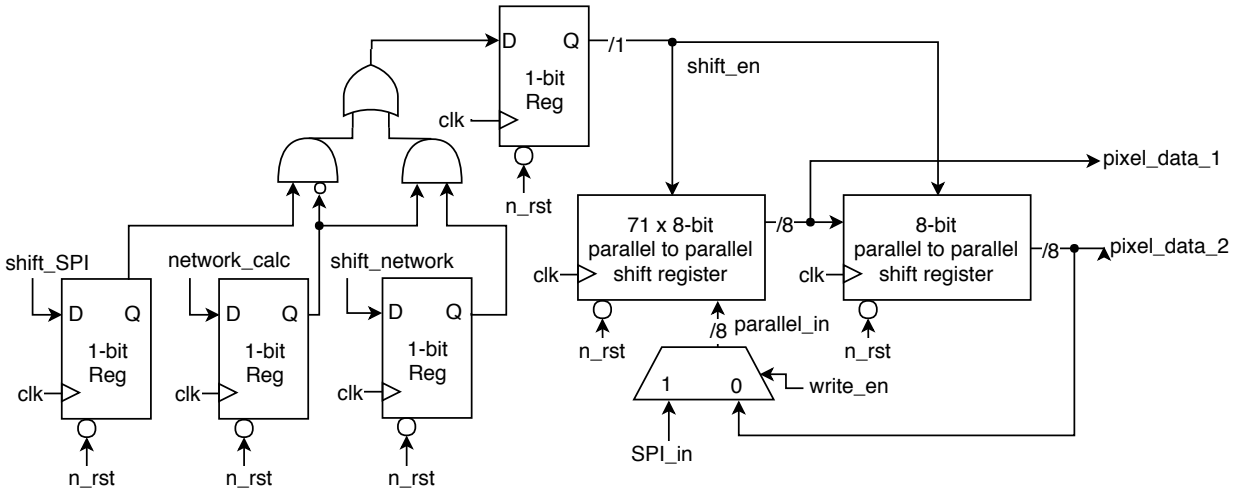


Figure 24: Pixel Data Register RTL diagram

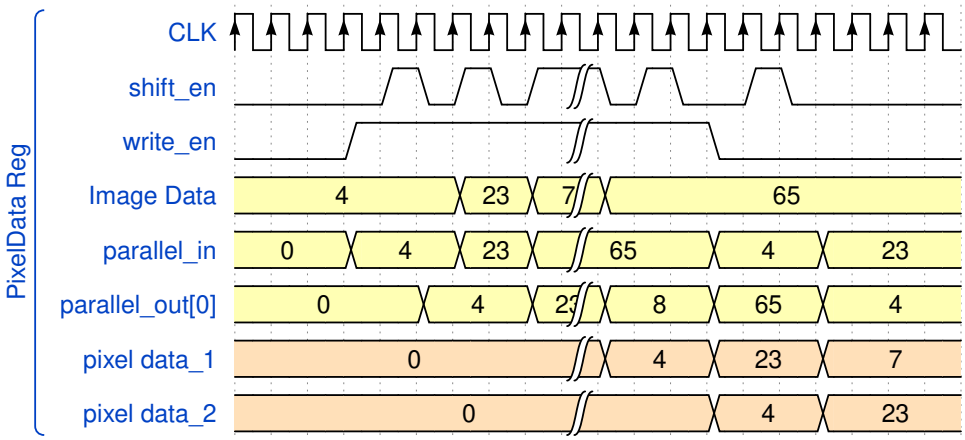


Figure 25: Pixel Data Register timing diagram

Name of Block	Category	Gate/FF Count	Area (μm^2)	Comments
Glue Logic	Combinational	41	30,750	
PTPSR array (reg.)	Reg. w/o Reset	576	777,600	72 instances of PTPSRs
PTPSR array (comb.)	Combinational	1400	1,080,000	
Total Estimated Area		**2017**	**1,888,350**	
Total Actual Area			**1,282,713**	

Table 16: Pixel Data Register area table

The largest component of this block is the PTPSR array. This needs to be very large as it needs to store the data for every pixel in the system. Each shift register both needs the register for storage, as well as combinational logic for enabling. The registers don't have a reset to conserve space, as they will be flushed with correct values every time a new image is sent to the device. A lot of area was saved in this block in the combinational logic. It was very well optimized especially since there are gates needed between every register in the shift register. this saved us about 30 percent of the area we estimated we needed.

PTP Shift Register: The 8 bit parallel-to-parallel shift register used for the pixel data register is depicted below.

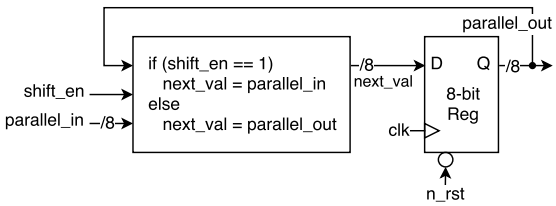


Figure 26: PTPSR RTL diagram

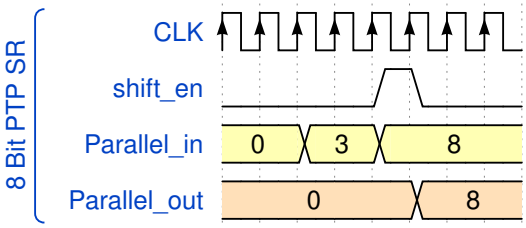


Figure 27: PTPSR timing diagram

Name of Block	Category	Gate/FF Count	Area (μm^2)	Comments
8 bit Register	Reg. w/o Reset	8	10,800	
Multiplexer	Combinational	20	15,000	
Total Area		**28**	**25,000**	

Table 17: PTPSR area table

The PTPSR needs an 8 bit register for storage, as well as multiplexing for next state logic.

3.2.5 Sigmoid Registers

The Sigmoid Registers are a block of Addressable Registers that are responsible for storing the sigmoid values needed for the hidden Layer, at addresses 0-7, and also storing the digit weights for each number, at addresses 8-17. The Digit Weights are constantly output on an additional bus. Data from any register 0-17 can be obtained from the data-out line by setting the address line to the appropriate register number. The Sigmoid Registers also take up a large amount of area due to needed all registers addressable and storing 144 bits of data internally. This block uses 18 Addressable Registers.

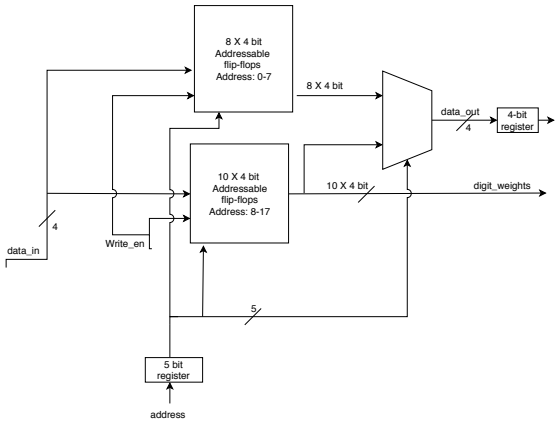


Figure 28: Sigmoid Register RTL diagram

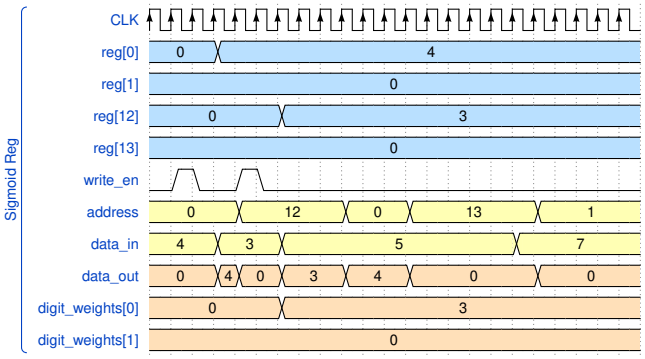


Figure 29: Sigmoid Register timing diagram

Name of Block	Category	Gate/FF Count	Area (μm^2)	Comments
Glue Logic	Combinational	100	75,000	
Addressable Register Array	Reg. w/o Reset	144	194,400	18 instances of Addressable Regs
	Combinational	1008	756,000	
Total Estimated Area		**1252**	**1,025,400**	
Total Actual Area			**205,846**	

Table 18: Sigmoid Register area table

The Addressable Register Array is made up of 18 of the Addressable Registers as shown in the next page.The glue logic is a combinational block to get the right address value for each bus. Similar to the pixel data registers, a lot of Area was saved in this block from the combinational logic being more optimized than we anticipated. Also, the final output large multiplexer was cut down in size because we didnt need all registers to go through it. The overall on this block we managed to hit about 20 percent of our estimated area.

Addressable Register: The Addressable Register, depicted below, is a special type of 8 bit register that will only change its value if the output write-en is set high and the output address-in is the same value as a previously set parameter ADDRESS. The output of the register is always available on the Data-out line.

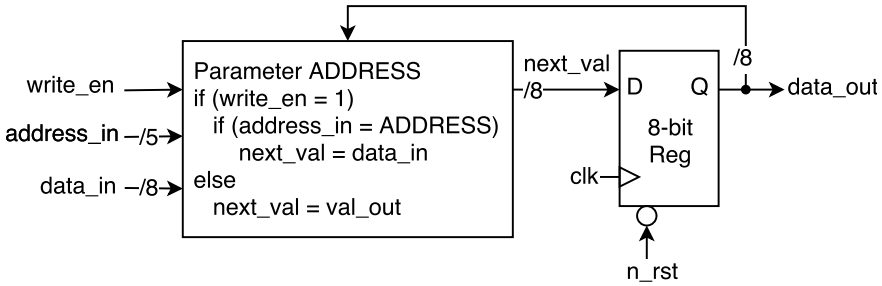


Figure 30: Addressable Register RTL diagram

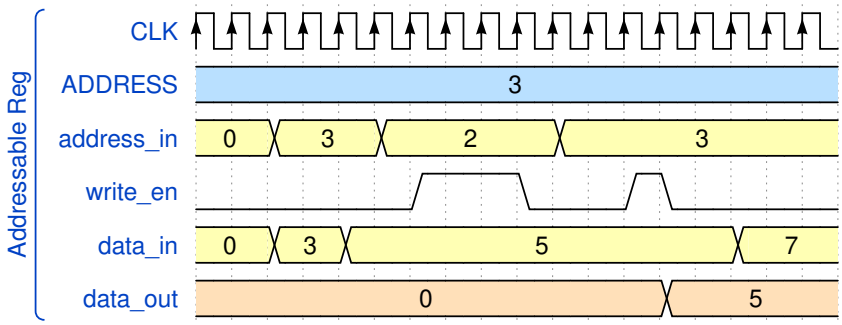


Figure 31: Addressable Register timing diagram

Name of Block	Category	Gate/FF Count	Area (μm^2)	Comments
Addressable Reg.	Reg. w/o Reset	8	10,800	
	Combinational	56	42,000	
Total Area		**64**	**52,800**	

Table 19: Addressable Register area table

Each Addressable Register has to be broken up into a register for storing the current data, as well was combinational logic that handles the multiplexing and logic necessary for loading new data, as well as checking the address.

3.2.6 Network Controller

The Network Controller directs the inputs and outputs of the neural network and is split up into three different state machines. The first state machine controls the I/O of the input layer to the hidden layer and the I/O of the hidden layer to the output layer. The main method of control for the first state machine is to control the counters for the hidden layer and output layer. Each counter represents the amount of inputs going into each neuron in hidden and output layers. The other state machines control the signals in the hidden and output layers, including the address lines to the flash controller, the pixel registers, and the sigmoid registers.

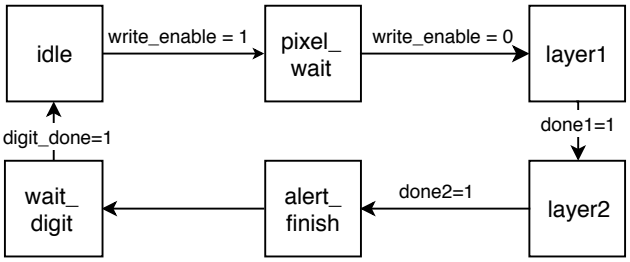


Figure 32: Network Controller top level state diagram

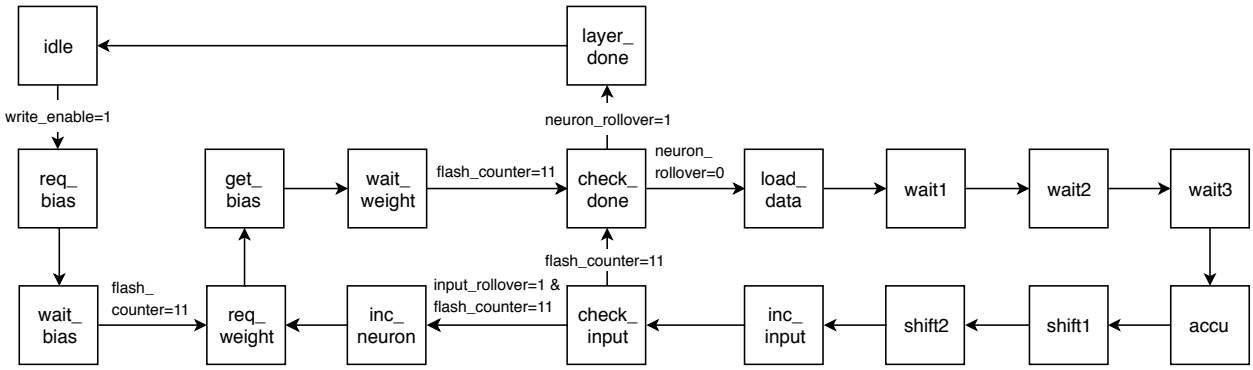


Figure 33: Network Controller layer 1 state diagram

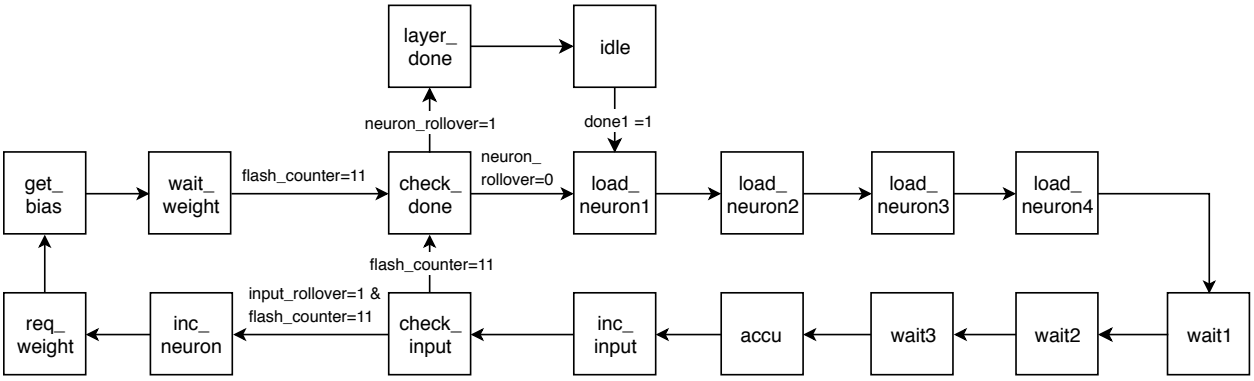


Figure 34: Network Controller layer 2 state diagram

State	input_rollover	neuron_rollover	network_done	data_ready
idle	0	0	0	1
pixel_wait	0	0	0	0
layer1	36	8	0	0
layer2	2	10	0	0
alert_finish	0	0	1	0

(a) Network Controller top level output logic

State	input_en	weight_en	bias_en	shift	sig_write	ready	acc	clr
idle	0	0	0	0	0	0	0	0
load_bias	0	0	1	0	0	1	0	1
load_weight1	0	1	0	0	0	1	0	0
check_done	0	0	0	0	0	0	0	0
load_data	1	0	0	0	0	1	0	0
wait1	0	0	0	0	0	0	0	0
accu	0	0	0	0	0	0	1	0
shift1	0	0	0	1	0	0	0	0
shift2	0	0	0	1	0	0	0	0
inc_input	0	0	0	0	0	0	0	0
check_input	0	0	0	0	0	0	0	0
inc_neuron	0	0	0	0	1	0	0	0
layer_done	0	0	0	0	0	0	0	0

(b) Network Controller layer 1 output logic

State	input_en	weight_en	bias_en	sig_write	ready	acc	clr
idle	0	0	0	0	0	0	0
load_bias	0	0	1	0	1	0	1
load_weight1	0	1	0	0	1	0	0
check_done	0	0	0	0	0	0	0
load_neuron1	1	0	0	0	0	0	0
load_neuron2	1	0	0	0	0	0	0
wait1	0	0	0	0	0	0	0
accu	0	0	0	0	0	1	0
inc_input	0	0	0	0	0	0	0
check_input	0	0	0	0	0	0	0
inc_neuron	0	0	0	1	0	0	0
layer_done	0	0	0	0	0	0	0

(c) Network Controller layer 2 output logic

Table 20: Cumulative Network Controller output logic

Name of Block	**Category**	**Gate/FF Count**	**Area (μm^2)**	**Comments**
State Registers	Reg. w/ Reset	5	12,000	32 states→5 bits
Next State Logic	Combinational	75	56,250	
Output Logic	Combinational	100	75,000	19 outputs
6 Bit Flex Counter	Mixed	36	36,900	Pixel Counter
4 Bit Flex Counter	Mixed	24	24,600	Flash Counter
3 Bit Flex Counter	Mixed	18	18,450	Neuron Counter
Total Estimated Area		**258**	**223,200**	
Total Actual Area			**518,616**	

Table 21: Network Controller area table

The main area for the network controller comes from the output logic, as this will be complicated compared to the output logic of other components. In total, it will need to control 19 bits of output, making it require more logic. While the exact size of logic is difficult to estimate, this is conservative to ensure there is enough space. This needs a reset to ensure that the state of the system is known. Finally, there are flex counters used that are computed to have a size similar to others used in the Digit Recognizer. The final area on this block ended up being a lot larger than we thought. This is due to the fact that we had to add several more states to the controller and another counter into it. Also the output logic had to be reworked a bit and more output signals were add. We also had to add several staging registers to many of the inputs and outputs to meet timing requirements for the design. This resulted in us being at about 200 percent of our estimated area. This area was made up in other blocks where we were able to save space.

3.2.7 Flash Memory Controller

The Flash Memory Controller (FMC) controls the inputs and outputs between the Network Controller. When there is no request for data from the Network Controller, it stays at the idle state. When the Network Controller is ready to request data, it sends a ready signal and the FMC read the address provided. The intermediate states controls the timing for sending the address from the Network Controller to the flash memory and getting data from the flash memory to send it back to the Network Controller.

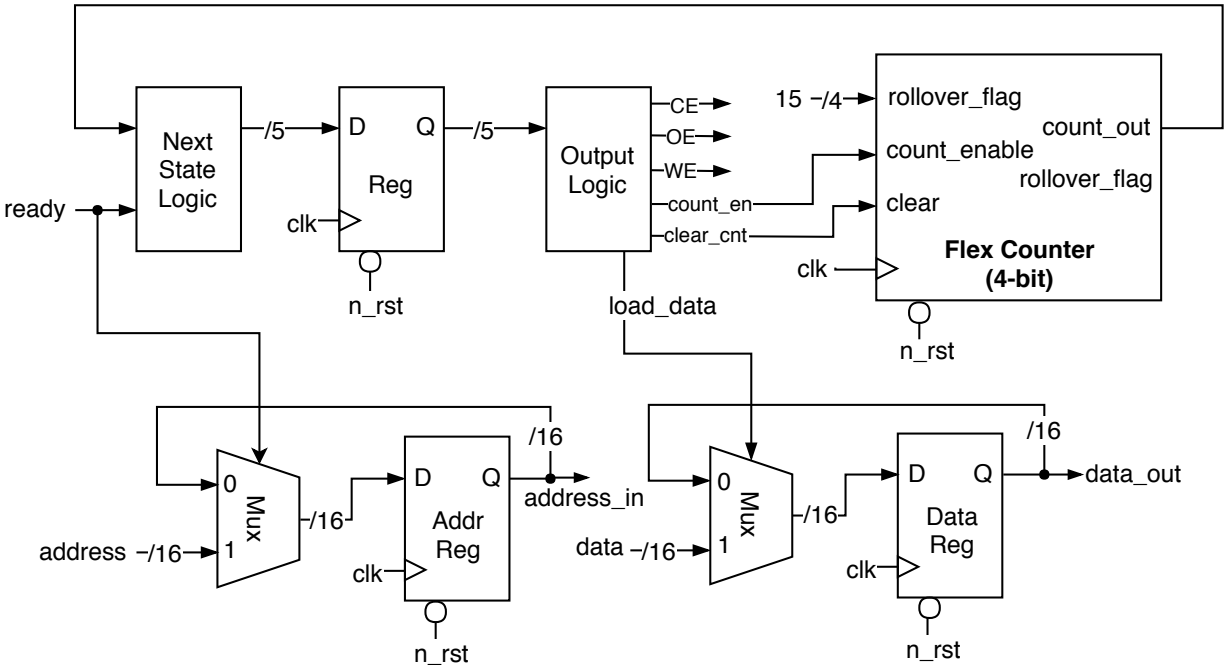


Figure 35: Flash Memory Controller RTL diagram

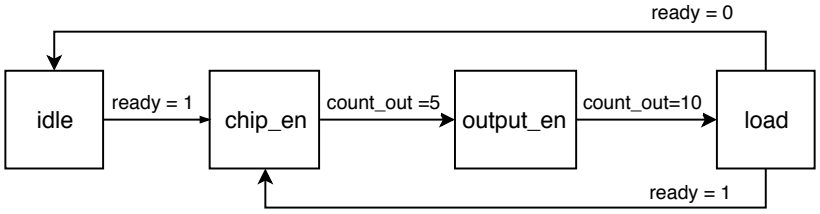


Figure 36: Flash Memory Controller state transition diagram

State	CE	OE	WE	count_en	clear_cnt	load_data
idle	1	1	1	0	1	0
chip_enable	0	1	1	1	0	0
output_enable	0	0	1	1	0	0
load	0	0	1	0	1	1

Table 22: Flash Memory Controller output logic

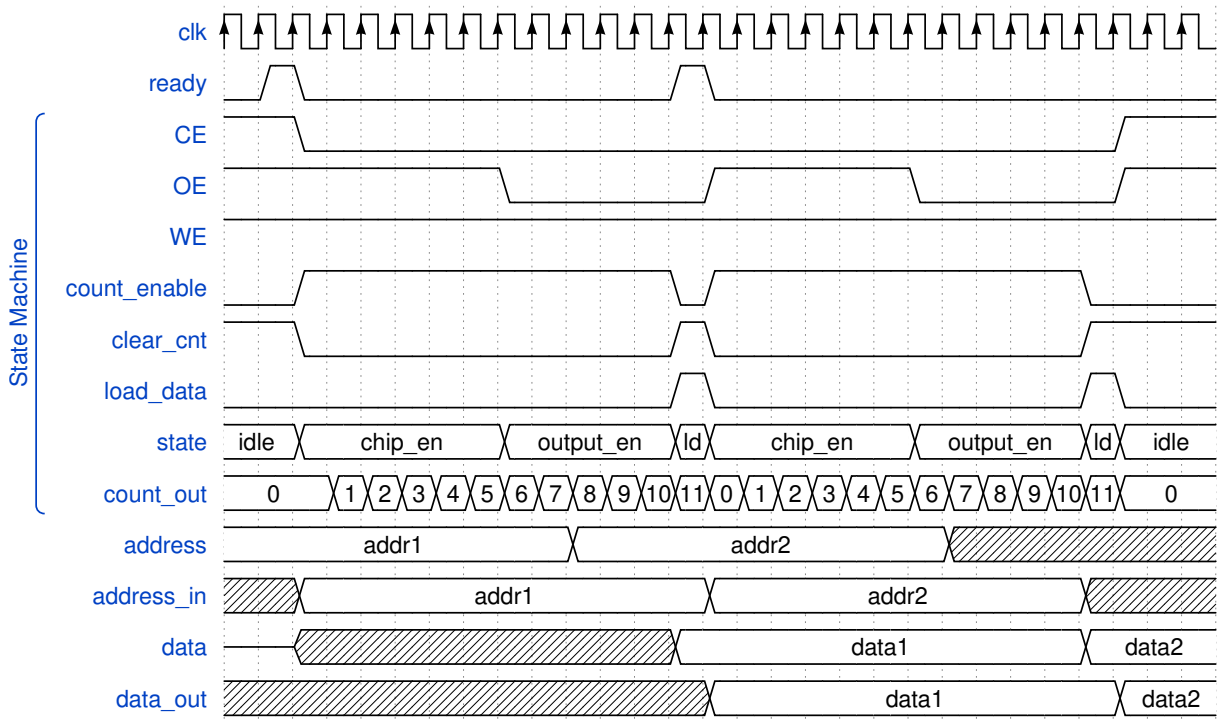


Figure 37: Flash memory controller timing diagram

Name of Block	**Category**	**Gate/FF Count**	**Area (μm^2)**	**Comments**
Next State Logic	Combinational	20	15,000	
State Register	Reg. w/ Reset	4	96,000	8 States→4 bits
Output Logic	Combinational	20	15,000	
4 Bit Flex Counter	Mixed	20	27,000	
16:1 Address Mux	Combinational	64	48,000	
Address Register	Reg. w/ Reset	16	38,400	
16:1 Data Mux	Combinational	64	48,000	
Data Register	Reg. w/ Reset	16	38,400	
Total Estimated Area		**224**	**238,400**	
Total Actual Area			**139,212**	

Table 23: Flash Memory area table

The largest components here are the address register and the data register with 16 registers each. They will have to be registers with resets such that the registers do not store random values especially because they are interfacing with an external flash memory. The other blocks are mainly combinational blocks. The state register will also be registers with resets to make sure that they go back to the idle state.

3.2.8 Sigmoid ALU

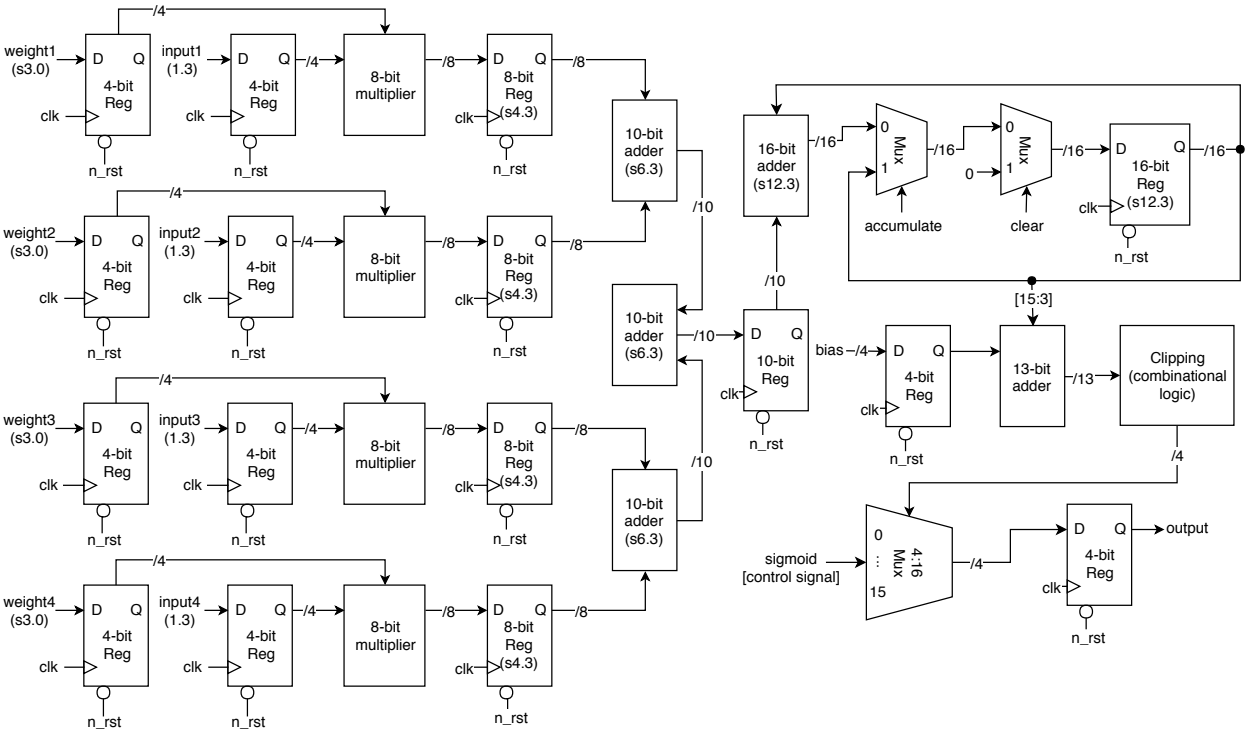


Figure 38: Sigmoid ALU RTL Diagram

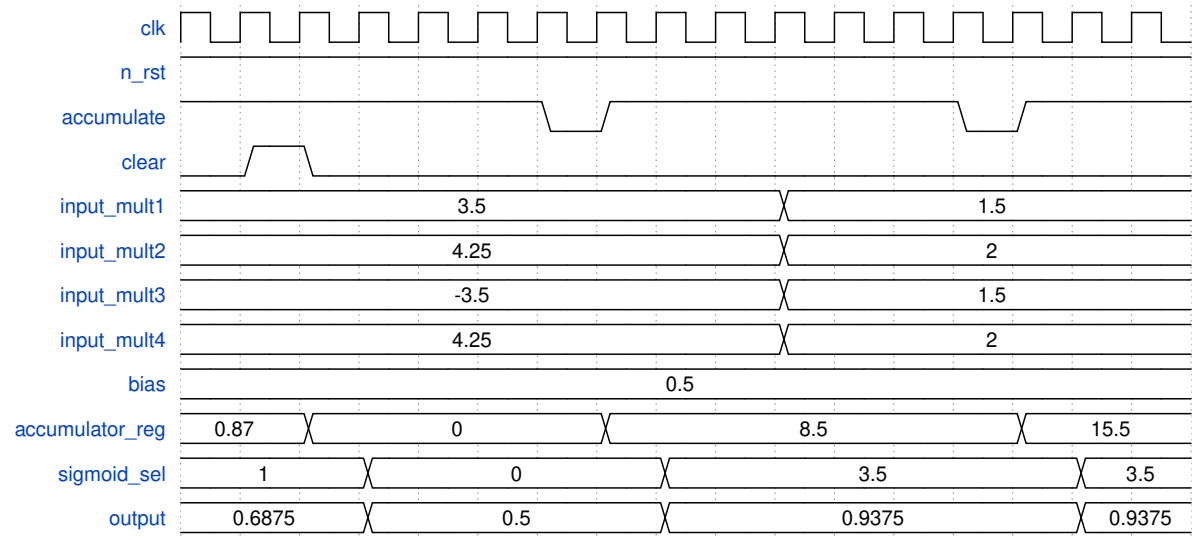


Figure 39: Sigmoid ALU timing diagram

Name of Block	**Category**	**Gate/FF Count**	**Area (μm^2)**	**Comments**
8 Bit Register #1	Reg. w/o Reset	8	10,800	
8 Bit Register #2	Reg. w/o Reset	8	10,800	
8 Bit Register #3	Reg. w/o Reset	8	10,800	
8 Bit Register #4	Reg. w/o Reset	8	10,800	
16 Bit Register	Reg. w/o Reset	16	21,600	
4 Bit Register	Reg. w/o Reset	4	5,400	
8 Bit Multiplier #1	Combinational	68	51,000	4 Bit Multiplier, 8 Bit Result
8 Bit Multiplier #2	Combinational	68	51,000	4 Bit Multiplier, 8 Bit Result
8 Bit Multiplier #3	Combinational	68	51,000	4 Bit Multiplier, 8 Bit Result
8 Bit Multiplier #4	Combinational	68	51,000	4 Bit Multiplier, 8 Bit Result
10 Bit Adder #1	Combinational	55	41,250	10 Bit CLA Adder
10 Bit Adder #2	Combinational	55	41,250	10 Bit CLA Adder
10 Bit Adder #3	Combinational	55	41,250	10 Bit CLA Adder
16 Bit Adder	Combinational	136	102,000	16 Bit CLA Adder
16:1 Mux #1	Combinational	64	48,000	
16:1 Mux #2	Combinational	64	48,000	
13 Bit Adder	Combinational	84	63,000	13 Bit CLA Adder
Clipping Logic	Combinational	42	192,000	
Sigmoid Mux	Combinational	256	192,000	4 Bit Select Signal, 4 bits Out
10 Bit Register	Reg. w/o Reset	10	13,500	
Total Estimated Area		**1155**	**895,950**	
Total Actual Area			**442,690**	

Table 24: Sigmoid ALU area table

The adders are carry-lookahead adders instead of ripple adders and are therefore bigger combinational blocks because they prioritize speed over area. Specifically, this is due to the extra carry propagation logic that is used. The registers here are registers without resets because they will be constantly flushed with new values and will not be used for control logic. This is done to save space for the system. The accumulator also has a synchronous clear, for when it does need to be reset by the network controller. One of the largest components is the Sigmoid Mux, as this is the multiplexer used as a 4 bit lookup table for the sigmoid function. However, this size is much smaller than earlier revisions of the design, which used a lookup table with an 8 bit control signal. The final actual Area used in this block was about 50 percent of what we expected. The compiler optimized the adders and multipliers much more than we anticipated. Even after adding additional registers into this block the final area was still well below our estimate.

3.2.9 Cost Calculator

The Cost Function Block (CFB) implements the loss function from Equation 3 when given a calculate_cost signal from the SPI Input Controller. The CFB operates as a state machine that accumulates the squared error between the output the expected label for each of the 10 elements in the input vectors. The CFB must first clock in the expected label values from the SPI Input Controller that are high for exactly one clock cycle. Once the values are clocked in to the Label Hold Registers, the State Register will determine which element in the vector to operate on. The exact order of operation for the calculations is shown in Figure 41. Once the accumulated cost has been calculated, the cost_ready signal is held high until a new calculate_cost signal is sent from the SPI Input Controller.

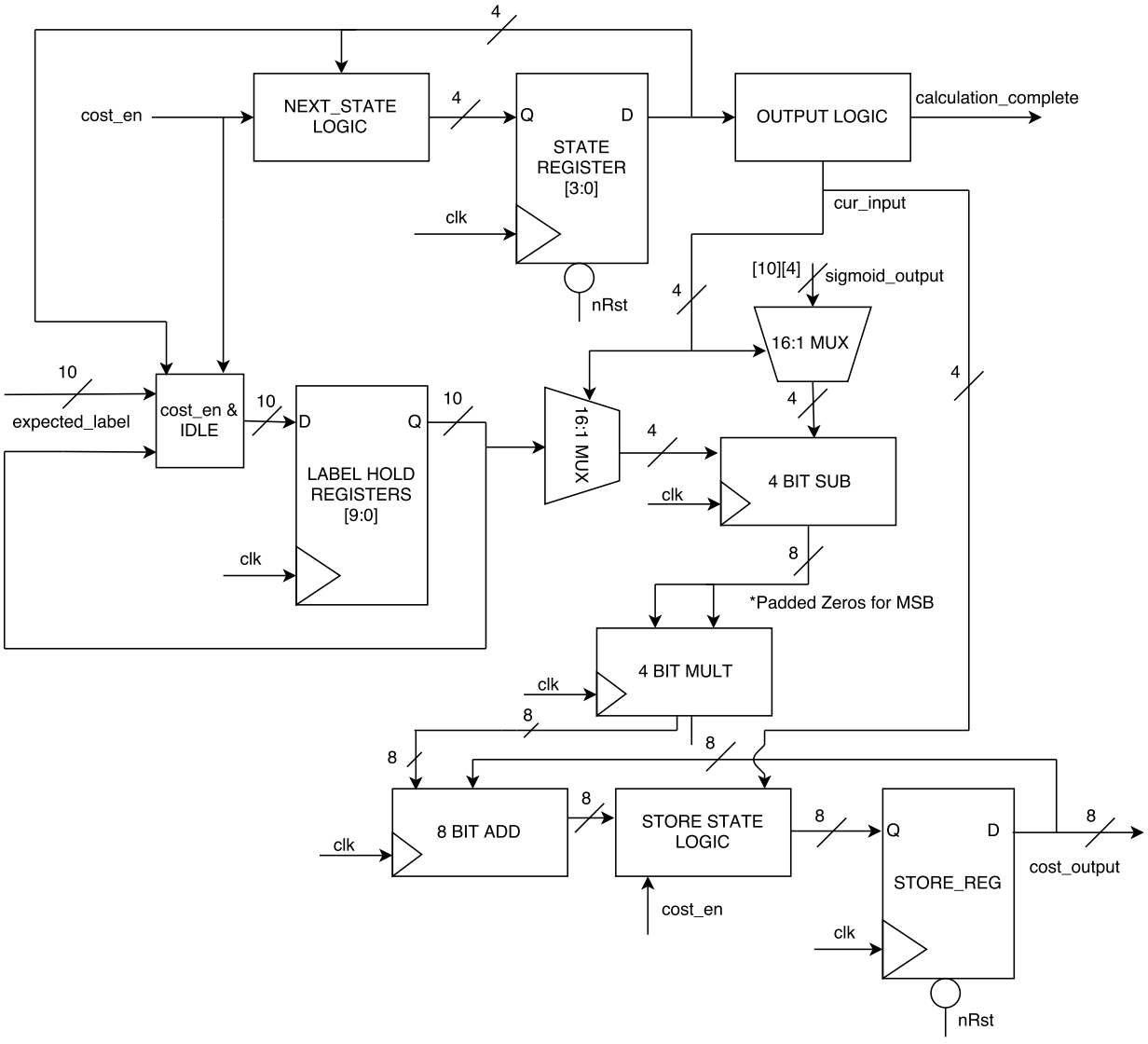


Figure 40: Cost Calculator RTL diagram

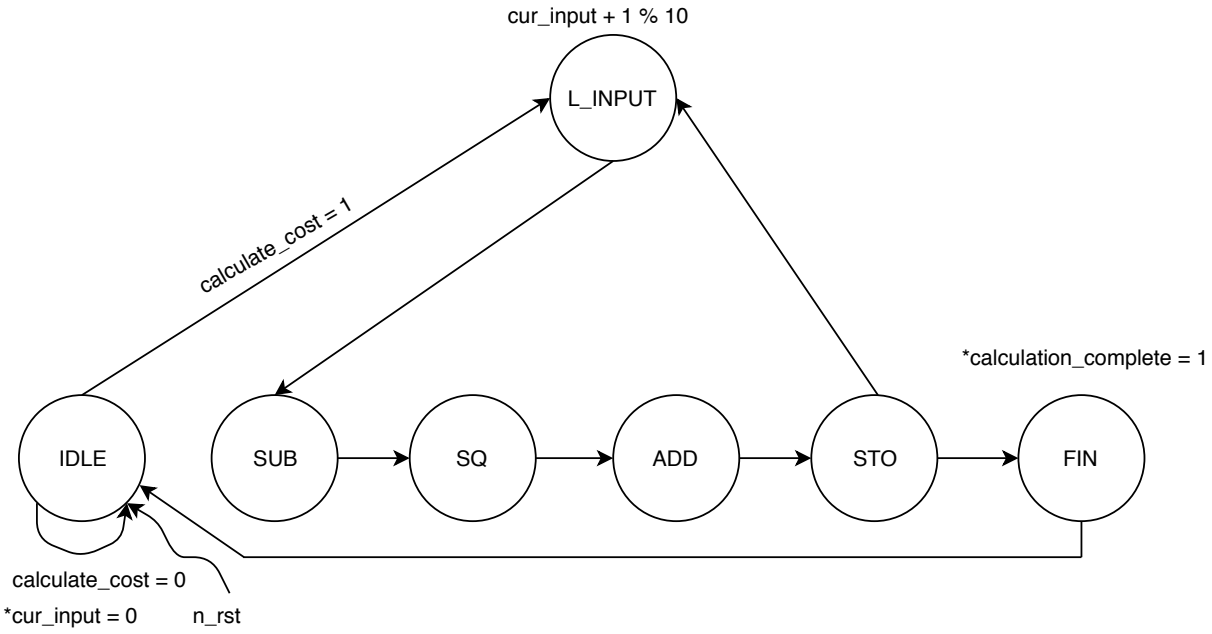


Figure 41: Cost Calculator state transition diagram

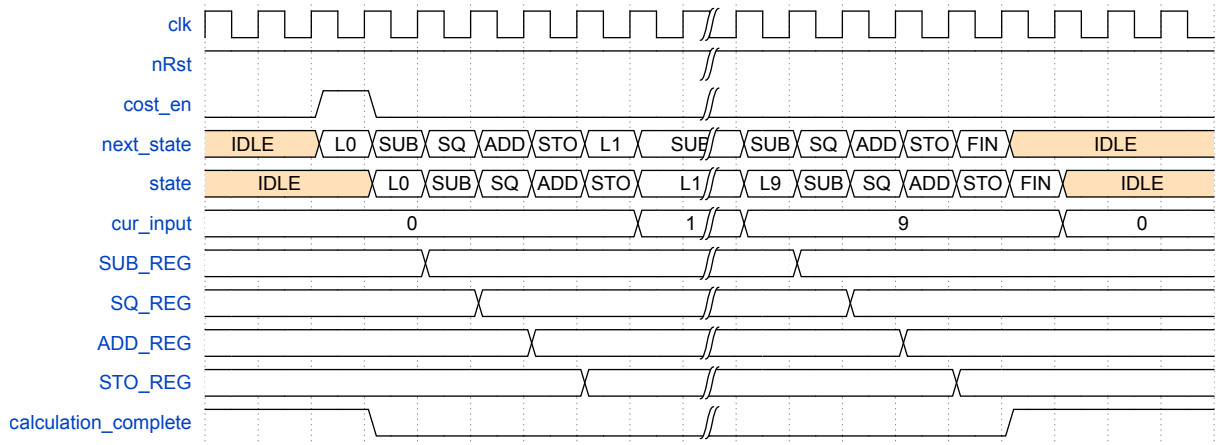


Figure 42: Cost Calculator timing diagram

Name of Block	**Category**	**Gate/FF Count**	**Area (μm^2)**	**Comments**
State Registers	Reg. w/ Reset	4	9,600	16 States
Next State Logic	Combinational	25	18,750	
Output Logic	Combinational	8	6,000	
16:1 Mux #1	Combinational	20	15,000	1 Bit Output
16:1 Mux #2	Combinational	80	60,000	4 Bit Output
Cost & IDLE	Combinational	4	3,000	
10:1 Mux	Combinational	40	30,000	
Label Hold Registers	Reg. w/ Reset	10	24,000	Labels from SPI Input
4 Bit Subtractor	Combinational	20	15,000	
4 Bit Multiplier	Combinational	280	210,000	
Mult. Product Comp.	Combinational	16	12,000	AND Gates for Product
8 Bit Adder	Combinational	40	30,000	
Subtractor Register	Reg. w/ Reset	8	19,200	Hold Result from Sub.
Multiplier Register	Reg. w/ Reset	8	19,200	Hold Result from Mult.
Adder Register	Reg. w/ Reset	8	19,200	Hold Result from Add
Store State Logic	Combinational	4	3,000	
Store Register	Reg. w/ Reset	8	19,200	
Total Estimated Area		**582**	**513,150**	
Total Actual Area			**380,698**	

Table 25: Cost Calculator area table

The largest component of the Cost Calculator is the 4 bit multiplier. This is because it needs to be optimized for speed instead of area, in order to meet the clock cycle of the system. As a multiplier has to perform multiple layers of addition, it needs to have area for each of those layers. The multiplexers used in the module have different areas because the size of the outputs are different. Finally, the adders and subtractors were given standard sizes used in other parts of the document. again much of the area saved in this block was due to the optimization of the adders and multipliers which we thought would take a lot more area than they actually did. Also in this block a few on the smaller pieces were redesigned in a more efficient way that helped reduce the area further. Overall our final area was about 70 percent of our estimated area.

3.2.10 Detected Digit

The Detected Digit Block (DDB) operates as the final output of the network before the SPI Output Controller by being the final step of the network operation. The DDB takes in a vector from the Sigmoid Registers which is the set of probabilities that the image input is a 0, 1, 2, ... and turns the probabilities into a 4 bit number which represents the number the image is most likely to be. The DDB is empirically a comparator to determine the max of the set of probabilities and output the number that the max probability corresponds to. It does this by finding the difference between two of the input probabilities and then finding the max between the two. The max is then sent to a register where it is held until a new max is found. When a new max is being loaded, the current index of the vector is loaded into the Digit Value Register which is the output register for the whole block.

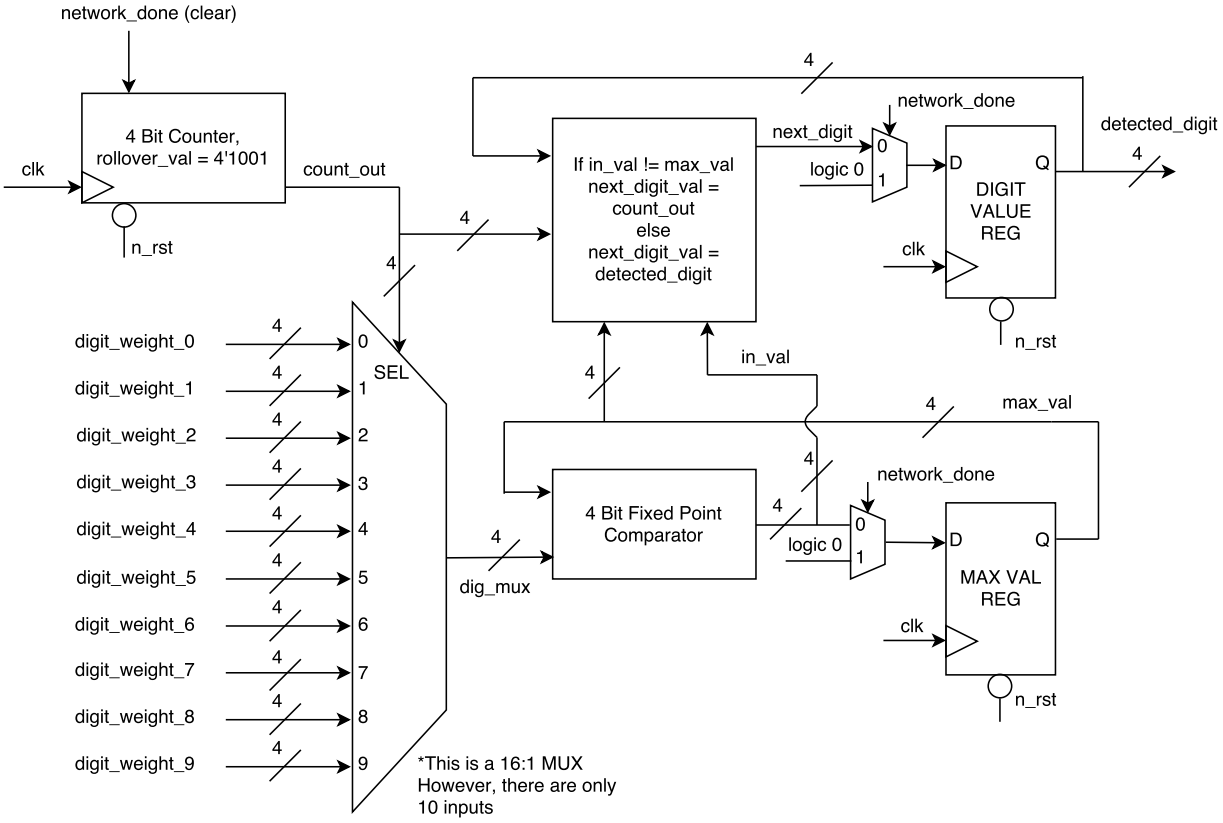


Figure 43: Detected Digit Block RTL diagram

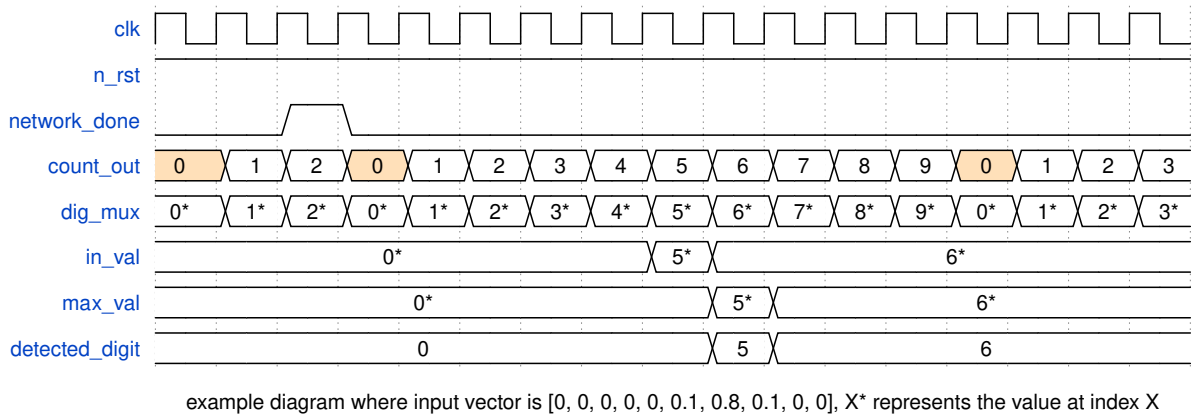


Figure 44: Detected Digit Block timing diagram

Name of Block	**Category**	**Gate/FF Count**	**Area (μm^2)**	**Comments**
4 Bit Counter	Mixed	24	27,600	24 Gates, 4 Reg. w/ Reset
16:1 Mux	Combinational	80	60,000	4 Bit Output
4 Bit Fixed Point Comparator	Combinational	20	15,000	4 Bit Subtractor
Max_Set Comb Block	Combinational	4	3,000	
2:1 Mux #1	Combinational	16	12,000	4 Bit Output
2:1 Mux #2	Combinational	16	12,000	4 Bit Output
Digit value Reg	Reg. w/ Reset	4	9,600	
Max value Reg	Reg. w/ Reset	4	9,600	Labels from SPI Input
Total Estimated Area		**168**	**148,800**	
Total Actual Area			**229,498**	

Table 26: Detected Digit area table

The fixed point comparator is a combinational block that does a 4 bit subtractor. The Max_Set Comb Block is a combinational block that does the logic to update the maximum value.

3.3 Design Timing Analysis

3.3.1 Predicted Top Paths

Our Target Clock is at 200 MHz. Thus each clock cycle is about 5 ns. None of our operations are currently multi-cycle operations so each block has a delay-constraint of 5 ns. Most of our predicted critical paths are in the Sigmoid ALU with one path coming from a multiplier in the cost calculator. Overall none of the critical paths take more than one clock cycle and thus should fit our design.

Start Block	**Tp (ns)**	**Comb. Logic**	**Tp (ns)**	**End**	**Tp (ns)**	**Total (ns)**	**Delay Constraint**	**Block**
Weight Inputs	0.1	8 Bit Multiplier	4.4	8 Bit Reg.	0.2	4.7	5 ns	Sigmoid ALU
4 Bit Subt. Reg.	0.1	4 Bit Multiplier	4.4	8 Bit Mult. Reg.	0.2	4.7	5 ns	Cost Calculator
10 Bit Reg.	0.1	16 Bit Adder + 2 MUX	2.8	16 Bit Reg.	0.2	3.1	5 ns	Sigmoid ALU
16 Bit Reg.	0.1	13 Bit Adder + Clipping Logic	2.5	4 Bit Reg.	0.2	2.8	5 ns	Sigmoid ALU
8 Bit Reg.	0.1	2 10 Bit Adders	2.0	10 Bit Reg.	0.2	2.3	5 ns	Sigmoid ALU

Table 27: Estimated top 5 critical paths

3.3.2 Actual Top Critical Paths

Start Block	**Tp (ns)**	**Comb. Logic**	**Tp (ns)**	**End**	**Tp (ns)**	**Total (ns)**	**Delay Constraint**	**Block**
Network Controller	0.8	Flash Counter	4.8	Count Reg.	0.0	5.82	5.87	Network Controller
Sigmoid ALU	0.8	Sigmoid Function	4.8	Output Reg.	0.0	5.629	5.753	Sigmoid ALU
Cost Calculator	0.8	Subtraction	5.0	Result Reg.	0.0	5.81	5.87	Cost Calculator
Network Controller	0.8	Output Logic	4.9	Address Reg.	0.0	5.78	5.88	FMC
Network Controller	0.8	Loading Input	5.0	Input Reg	0.0	5.64	5.86	Network Controller

Table 28: Actual top 5 unique critical paths

The actual critical paths, shown in Table 28, were different from the predicted path. Additionally, the delay constraints for the layout were different from the predicted 5ns, because of various delays due to the physical wiring of the clock that were not factored into the prediction. The major reason for the difference in predicted and final outputs is because the layout also has additional buffers due to fanout and wire lengths. Thus, the critical paths depended on the location of the modules, the fanout from a signal, in addition to the logic itself. Despite the extra delays, the design was able to meet all the delay constraints.

H	Path	Clock	ReqTime	Slack	Startpoint Pin	Endpoint Pin
	1	clk(leading)->clk(leading)	5.873	0.053	I0/top_network_controller/layer1State_reg[0]/Q	I0/top_network_controller/flashCounter/count_out_reg[3]/D
	2	clk(leading)->clk(leading)	5.753	0.061	I0/top_sigmoid_ALU/bias_reg_reg[3]/Q	I0/top_sigmoid_ALU/out_reg[1]/D
	3	clk(leading)->clk(leading)	5.872	0.062	I0/top_cost_calculator/IND_BLOCK/count_out_reg[...	I0/top_cost_calculator/sub_reg_reg[2]/D
	4	clk(leading)->clk(leading)	5.749	0.064	I0/top_sigmoid_ALU/bias_reg_reg[3]/Q	I0/top_sigmoid_ALU/out_reg[0]/D
	5	clk(leading)->clk(leading)	5.868	0.080	I0/top_cost_calculator/IND_BLOCK/count_out_reg[...	I0/top_cost_calculator/sub_reg_reg[3]/D
	6	clk(leading)->clk(leading)	5.881	0.084	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[0]/D
	7	clk(leading)->clk(leading)	5.880	0.085	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[2]/D
	8	clk(leading)->clk(leading)	5.781	0.087	I0/top_sigmoid_ALU/bias_reg_reg[3]/Q	I0/top_sigmoid_ALU/out_reg[2]/D
	9	clk(leading)->clk(leading)	5.869	0.094	I0/top_cost_calculator/IND_BLOCK/count_out_reg[...	I0/top_cost_calculator/sub_reg_reg[1]/D
	10	clk(leading)->clk(leading)	5.881	0.096	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[1]/D
	11	clk(leading)->clk(leading)	5.879	0.096	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[4]/D
	12	clk(leading)->clk(leading)	5.881	0.101	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[5]/D
	13	clk(leading)->clk(leading)	5.875	0.101	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[3]/D
	14	clk(leading)->clk(leading)	5.880	0.111	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[6]/D
	15	clk(leading)->clk(leading)	5.713	0.114	I0/top_sigmoid_ALU/bias_reg_reg[3]/Q	I0/top_sigmoid_ALU/out_reg[3]/D
	16	clk(leading)->clk(leading)	5.880	0.121	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[7]/D
	17	clk(leading)->clk(leading)	5.869	0.135	I0/top_network_controller/layer1State_reg[0]/Q	I0/top_network_controller/flashCounter/count_out_reg[2]/D
	18	clk(leading)->clk(leading)	5.876	0.145	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[8]/D
	19	clk(leading)->clk(leading)	5.884	0.155	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[9]/D
	20	clk(leading)->clk(leading)	5.883	0.162	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[15]/D
	21	clk(leading)->clk(leading)	5.884	0.164	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[10]/D
	22	clk(leading)->clk(leading)	5.883	0.174	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[12]/D
	24	clk(leading)->clk(leading)	5.885	0.177	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[13]/D
	23	clk(leading)->clk(leading)	5.884	0.177	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[11]/D
	25	clk(leading)->clk(leading)	5.885	0.179	I0/top_network_controller/topState_reg[2]/Q	I0/top_fmc/address_in_reg[14]/D
	26	clk(leading)->clk(leading)	5.862	0.224	I0/top_network_controller/layer2State_reg[2]/Q	I0/top_network_controller/input4_reg[0]/D

Figure 45: A screenshot from Innovus showing the top paths

3.3.3 Critical Path 1: Sigmoid ALU

This is a three unit path. The inputs we are assuming will take about 0.1 ns of delay and registers we assume take 0.1 ns of hold time. Most of the delay comes from the 8 Bit Multiplier. The multiplier takes in two 4 bit numbers and outputs an 8 bit number. the multiplier has 3 diagonals of adders and 4 rows. The largest path in this is across 7 adders which we estimate take 0.6 ns per adder. Then the final layer of the multiplier takes another 0.2 ns, adding up to 4.4 ns.

Block	**Tp (ns)**	**Delay Constraint**
Weight Input Pins	0.1	
8 Bit Multiplier	4.4	
8 Bit Register	0.2	
Total Delay	4.7	5 ns

Table 29: Critical Path 1

3.3.4 Critical Path 2: Cost Calculator

The 4 bit multiplier (8 bit output) is the largest path taken within the Cost Calculator Block. The multiplier takes in two 4 bit numbers and creates an 8 bit result. This requires 7 adder cells, each which have a propagation delay of 0.6 ns. This leads to a 4.2 ns delay, which goes up to 4.4 ns once the product component gates are taken into consideration.

Block	**Tp (ns)**	**Delay Constraint**
4 Bit Subtractor FF	0.1	
4 Bit Multiplier	4.4	
8 Bit Multiplier FF	0.2	
Total Delay	4.7	5 ns

Table 30: Critical Path 2

3.3.5 Critical Path 3: Sigmoid ALU

The 16 bit Adder takes up the largest time within this critical path. The CLA adder takes up 1.6 ns of propagation delay which is over half of the total in this path. The two multiplexers were analyzed by assuming that there would be one large multiplexer instead of a split into two smaller multiplexers. The propagation delay was then determined to be 0.6 ns for each multiplexer. The other time taken are due to setup / hold times for the registers.

Block	**Tp (ns)**	**Delay Constraint**
10 Bit Register	0.1	
16 Bit Adder	1.6	
Accumulate multiplexer	0.6	
clear multiplexer	0.6	
16 Bit Register	0.2	
Total Delay	3.1	5 ns

Table 31: Critical Path 3

3.3.6 Critical Path 4: Sigmoid ALU

This Path is a 4 block path. The start is from a register, so it takes 0.1 ns for the data to output. Multi bit adders we approximated to have a propagation delay of N/10 ns where N is the number of bits so the 13 Bit adder takes 1.3 ns. The Clipping Logic we believe will at most go through 6 gates. each gate has a delay of 0.2 ns so overall it has about 1.2 ns delay. Then finally the 4Bit Register has a 0.2 hold time. Adding up to a total path delay of 2.8 ns.

Block	**Tp (ns)**	**Delay Constraint**
16 Bit Register	0.1	
13 Bit Adder	1.3	
Clipping Logic	1.2	
4 Bit Register	0.2	
Total Delay	2.8	5 ns

Table 32: Critical Path 4

3.3.7 Critical Path 5: Sigmoid ALU

This Path is a 4 block path. The start is from a register, so it takes 0.1 ns for the data to output. Multi bit adders we approximated to have a propagation delay of N/10 ns where N is the number of bits so each 10 Bit adder takes 1.0 ns. Then finally the 4Bit Register has a 0.2 hold time. Adding up to a total path delay of 2.3 ns.

Block	**Tp (ns)**	**Delay Constraint**
8 Bit Register	0.1	
10 Bit Adder	1.0	
10 Bit Adder	1.0	
10 Bit Register	0.2	
Total Delay	2.3	5 ns

Table 33: Critical Path 5

4 Success Criteria

4.1 Fixed Criteria

4.1.1 Fixed Success 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.

Status: PASSED

```
mg79 ~...ece337_digit_recognizer-develop/verilog/source >ls tb*
tb_cost_calculator.sv                tb_sigmoid_ALU_4_way_adder.sv
tb_digit_decode.sv                   tb_sigmoid_ALU_multiplier.sv
tb_digit_recognizer_final.sv         tb_sigmoid_ALU_sigmoid_calculator.sv
tb_digit_recognizer.sv               tb_sigmoid_ALU.sv
tb_fmc.sv                            tb_sigmoidRegisters.sv
tb_gen_pos_edge_detect.sv            tb_SPI_input_controller.sv
tb_networkController.sv              tb_SPI_output_controller.sv
tb_pixelData.sv
```

Figure 46: Existence of Test Benches for all Modules

Test benches exist for all top level files. See Appendix A Section 7.2 for list of test bench files with their associated modules.

4.1.2 Fixed Success Criteria 2

Entire design synthesizes completely, without any inferred latches, timing arcs, and sensitivity list warnings.

Status: PASSED

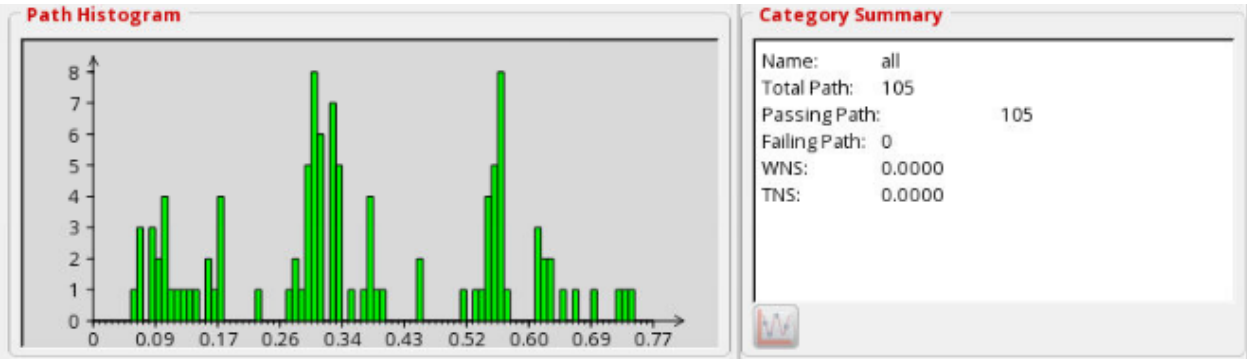


Figure 47: Correct Timing for all Critical Paths in the Design

Figure 47 shows the lack of negative slack paths within our design. Every single critical path within our design has a positive slack. In the synthesis of our design we ensured that no latches were created or sensitivity list warnings passed. See digit_recognizer_final.log (Appendix A Section 7.7) for the full successful synthesis log.

4.1.3 Fixed Success Criteria 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.

Status: PASSED

```
# |             ##       |
# |             ##..     |
# |                      |
# +----------------------+
# ** Info: Test Case   100
#    Time: 9299569 ns  Scope: tb_digit_recognizer_final File: source/tb_digit_recognizer_final.
sv Line: 228
# Result Digit:   9
# Expected Digit: 9
# Result Cost:    0
# Expected Cost:  0
#
# ** Info: Num tested:          100,  Num digit correct:          90
#    Time: 9312640 ns  Scope: tb_digit_recognizer_final File: source/tb_digit_recognizer_final.
sv Line: 247
# ** Info: Num tested:          100,  Num cost correct:          100
#    Time: 9312640 ns  Scope: tb_digit_recognizer_final File: source/tb_digit_recognizer_final.
sv Line: 248
```

Figure 48: 100 Image Test Vector Source Results

```
# |             ##       |
# |             ##..     |
# |                      |
# +----------------------+
# ** Info: Test Case   100
#    Time: 9299569 ns  Scope: tb_digit_recognizer_final File: source/tb_digit_recognizer_final.
sv Line: 228
# Result Digit:   9
# Expected Digit: 9
# Result Cost:    0
# Expected Cost:  0
#
# ** Info: Num tested:          100,  Num digit correct:          90
#    Time: 9312640 ns  Scope: tb_digit_recognizer_final File: source/tb_digit_recognizer_final.
sv Line: 247
# ** Info: Num tested:          100,  Num cost correct:          100
#    Time: 9312640 ns  Scope: tb_digit_recognizer_final File: source/tb_digit_recognizer_final.
sv Line: 248
```

Figure 49: 100 Image Test Vector Mapped Results

Figure 48 and Figure 49 show the results for the digit detection and cost for a 100 image input vector in source and mapped respectively. They both produce the same output and run in the same amount of time therefore satisfying FC 3. For more descriptive results refer to 100_imgs_cost_source.txt and 100_imgs_cost_mapped.txt (Appendix A Section 7.6).

4.1.4 Fixed Success Criteria 4

A complete IC layout is produced that passes all geometry and connectivity checks.

Status: PASSED

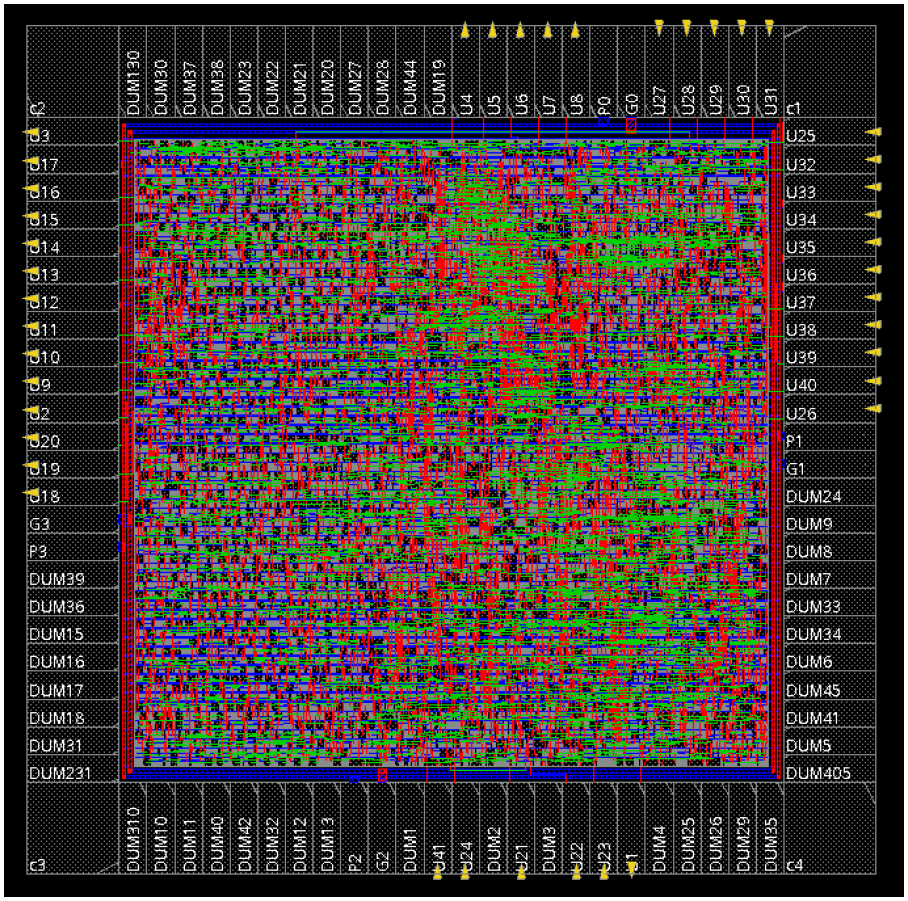


Figure 50: Full IC Design Layout

```
******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.3  MEM: 0.000M)
******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.4  MEM: 0.000M)
  Verification Complete : 0 Viols.  0 Wrngs.

**********End: VERIFY GEOMETRY**********
 *** verify geometry (CPU: 0:00:02.0  MEM: 1.0M)
```

Figure 51: Successful Geometry and Connectivity Checks

As shown from Figure 50 and Figure 51 the complete IC layout was able to be produced in Innovus with zero errors in regards to the geometry and connectivity of the chip. Furthermore, dummy pads were added in order to ensure that no gaps existed between the I/O pads.

4.1.5 Fixed Success Criteria 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. Failure to reach any of the targets will result a score of 1 out of 2 provided that you are within 50% on area, 10% on pin count, and 25% on throughput. Doing worse in any category will result in a score of 0 out of 2.

Status: PASSED

Design Status	Routed
Design Name	digit_recognizer_final
# Instances	32170
# Hard Macros	0
# Std Cells	32070
# Pads	100
# Net	5547
# Special Net	2
# IO Pins	41

Figure 52: Pin Count for Final IC

Floorplan/Placement Information

Total area of Standard cells	4200768.000 um^2
Total area of Standard cells(Subtracting Physical Cells)	2268864.000 um^2
Total area of Macros	0.000 um^2
Total area of Blockages	0.000 um^2
Total area of Pad cells	2952000.000 um^2
Total area of Core	4200768.000 um^2
Total area of Chip	7617600.000 um^2

Figure 53: Total Area for Final IC

```
** Info: Num tested:         10000,  Num correct:        8876
   Time: 850750040 ns  Scope: tb_digit_recognizer_final File:
```

Figure 54: Final Results of 10,000 Image Test Vector

From Figure 52 it is clear that the pin count is 41, however, once pins are added for power and ground this total sums up to 48 pins on our final IC. The final area with the area from the I/O pads was less than what was predicted. One reason for this was the area that each combinational gate was based on. Since the area was based on the XOR gate which is one of the larger of all logic gates, it makes sense that in many places our estimated combinational area was larger than what was actually synthesized by Innovus. See digit_recognizer_final.main.htm for a full description of pin count and total area of chip. To see the results of our 10,000 image test vector from the MNIST database, please refer to 10000_imgs.txt (Appendix A Section 7.6).

Parameter	Proposed	Actual
Pin Count	48	48
Area	10,097,580 μm^2	7,617,600 μm^2
Throughput	10000 imgs/sec	11763 imgs/sec
Clock	200 MHz	200 MHz

Table 34: Comparison of Proposed and Actual Design Parameters and Estimates

4.2 Design Specific Success Criteria

1. (1 point) Demonstrate by simulation of Verilog test benches that the complete design is able to detect at least 60% of digits.
2. (1 point) Demonstrate by simulation of Verilog test benches that the complete design is able to output a proper cost output for use in external training modules.
3. (2 points) Demonstrate by simulation of Verilog test benches that the complete design is able to correctly propagate a set of weights and biases through a sigmoid neuron.
4. (1 points) Demonstrate by simulation of Verilog test benches that the complete design is able to process 10,000 images per second.
5. (2 points) Demonstrate by simulation of Verilog test benches that the complete design is able to properly load in a 12×12 grayscale image through SPI, and output the specified error codes over SPI.
6. (1 points) Demonstrate by simulation of Verilog test benches that the complete design is able to correctly interface with a SST39LF200A flash memory chip by using specified timing parameters as a reference.

4.2.1 Design Specific Success Criteria 1

Status: PASSED

Our design was able to achieve 88.76 percent accuracy, well above the 60 percent target. Results for this can be seen in Figure 55 in design verification.

4.2.2 Design Specific Success Criteria 2

Status: PASSED

Our design is able to give out a proper cost after performing a detection on an image through SPI. The final cost output and its expected value can be seen in Figure 57 in Design Verification

4.2.3 Design Specific Success Criteria 3

Status: PASSED

Our design can take in weights and biases from a flash memory chip, and use them for calculations of a sigmoid neuron. This can be seen in Figure 56 in Design Verification.

4.2.4 Design Specific Success Criteria 4

Status: PASSED

Our design is able to process 11,763 images per second, Or about 10,000 images per 850 ms. This output can be seen in Figure 55 in Design Verification.

4.2.5 Design Specific Success Criteria 5

Status: PASSED

Our design is able to take in pixel data through SPI of a 12×12 grayscale image, and store those values in internal registers as seen in Figure 55 in Design Verification. Also, if the user requests the final output before it is ready our design properly outputs an error code through SPI as seen in Figure 59 in Design Verification.

4.2.6 Design Specific Success Criteria 6

Status: PASSED

Our design is able to interface with a SST39LF200A flash memory chip to receive data using reported timing parameters and specifications. The data sheet for the flash memory chip can be found in the directory tree. The use of the memory chip can be seen in Figure 58 of Design Verification.

5 Design Verification

5.1 Design Verification Overview

5.1.1 Top Level Tests

What to verify	Design modules involved	Verification Procedure Summary	DSSC(s) Proved	Use in final demo	Comments
Design can detect digits at a high speed with accuracy	Top Level	High speed test: Send multiple images and expected digits in quick succession, determine percent correct	DSCC 1 DSCC 4	Yes	Number of digits sent will be determined by how fast ASIC can be simulated
Correct chip response to input image sent over SPI	Top Level	Send image, wait for computation time, receive result	DSCC 3	Yes	Should be tested with a known valid image as results are probabilistic, do as part of high speed test
A valid cost is output after a computation on an image	Top Level	Send Valid image, wait for computation time, send expected digit, receive cost	DSCC 2	Yes	Do as part of high speed test
Design is able to retrieve weights and biases from a simulated memory chip	Top Level	Simulate SST39LF200A flash memory chip in Verilog, give design	DSCC 6	Yes	Do as part of high speed test
Correct Error output for invalid inputs	Top Level	Send Valid image, attempt to receive results before end of computation time	DSCC 5	Yes	

5.2 Testing Scenario Breakouts

Design can detect digits at a high speed with accuracy

- DSSC(s) Proved: 1 & 4
- Highest Level of Design Module(s) involved: Top Level Chip
- Test Bench Expectations/requirements:
  1. Send images and expected digits through simulated SPI
  2. Store result and calculate percent that design got correct
- No external or premade references are needed
- No pre/post processing is needed
- Main Verification Test Steps:
  1. Send image over SPI
  2. Receive and store result
  3. Repeat items 1 and 2 for all images
  4. Using expected digit and stored results, calculate percent correct

```
** Info: Num tested:         10000,  Num correct:        8876
   Time: 850750040 ns  Scope: tb_digit_recognizer_final File:
```

Figure 55: 10,000 images sent into design through SPI

The image above is the output of our top level test bench sending all 10,000 images in through the SPI. As can be seen the percent correct is 88.76 percent and it was able to do all 10,000 images in 850 ms. The full transcript of this output can be found in the 10000_imgs.txt (Appendix A Section 7.6) .

Correct chip response to input image sent over SPI

- DSSC(s) Proved: 3
- Highest Level of Design Module(s) involved: Top Level Chip
- Test Bench Expectations/requirements:
  1. Have image data and expected result
- No external or premade references are needed
- No pre/post processing is needed
- Main Verification Test Steps:

1. Send in valid image data over SPI
2. Compare result with expected value

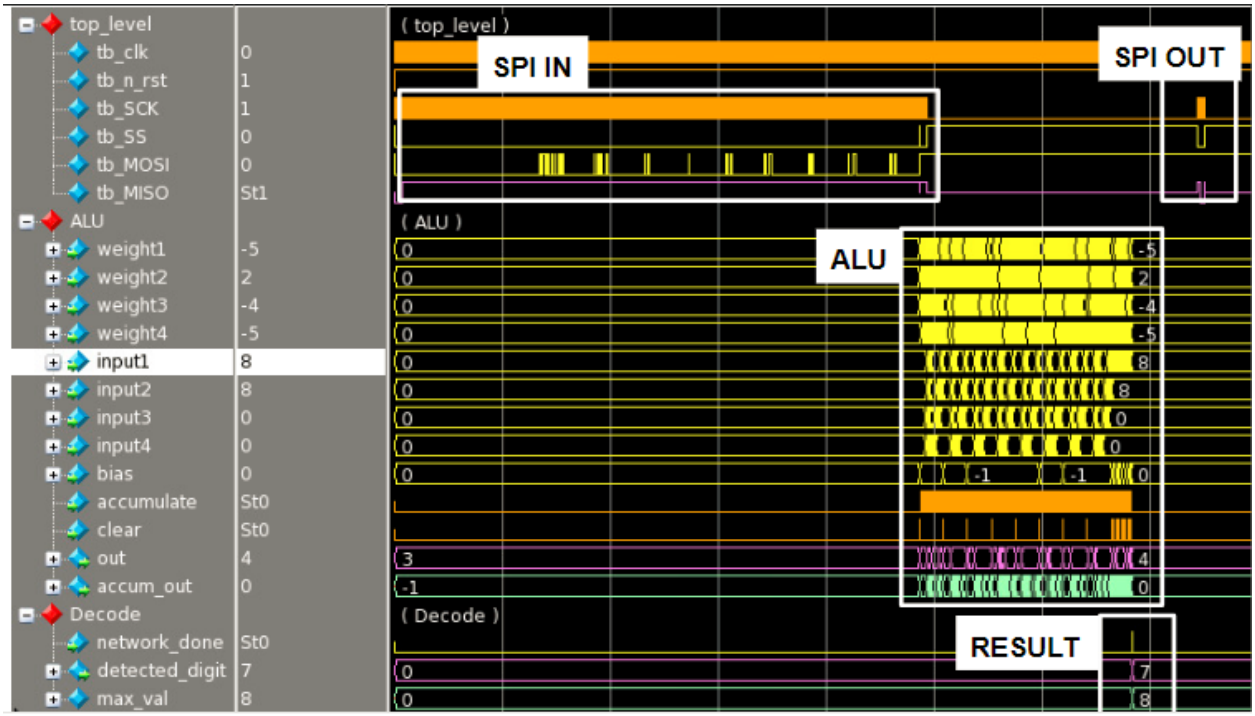


Figure 56: One Image of a 7 sent in through SPI

This is the waveform produced when a picture of a handwritten 7 is input. The image in fed into the design through the SPI. All weights and biases and pixels are then used in the ALU to determine the final result "detected-digit" which correctly generated a 7. Then the output is sent out via SPI again.

A valid cost is output after a computation on an image

- DSSC(s) Proved: 2
- Highest Level of Design Module(s) involved: Top Level Chip
- Test Bench Expectations/Requirements:
  1. Send expected label through SPI
  2. Correctly outputs cost given the precision of the arithmetic logic units
  3. Correctly outputs cost_ready signal when computations are complete
- No external or premade references are needed
- Post processing necessary in order to calculate correct cost of the operation

- Main Verification Test Steps:
    1. Send expected label through SPI
    2. Verify that the expected label is clocked in correctly into the label hold registers
    3. Verify that the state diagram is executed in order for correct operation of the cost calculator
    4. Compare results to computer generated result
    5. Repeat steps 1-4 for varying sets of input labels and confidence levels received from the simulated Sigmoid ALU/Registers

The validation of the cost calculator was done with the use of a top level test bench that would compare the calculated cost from the source / mapped file to that of a python generated list of all the expected values that the cost should take on. Figure 48 and Figure 49 show the cost results over a 100 image input vector. As shown in the figures the cost output is correct in every test image.

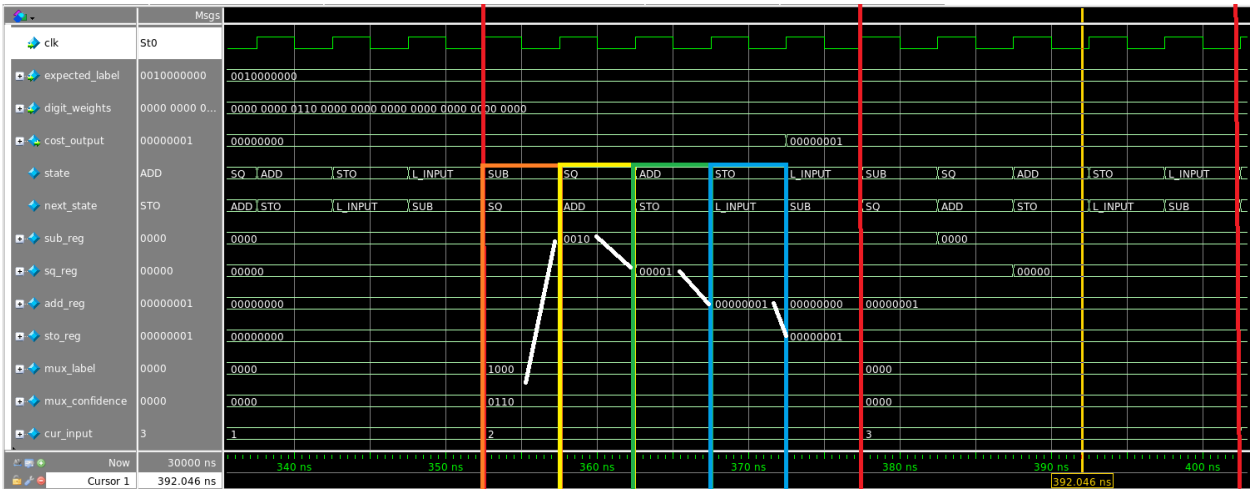


Figure 57: Example Cost Propagation of a 2 with Confidence 0.75

Figure 57 shows the expected operation of the cost calculator. Each red line represents the loading of a new confidence output from the output layer of sigmoid neurons and the respective expected level for each neuron. The orange, yellow, green, and blue boxes represent the SUBTRACT, SQUARE, ADD, and STORE states respectively. The total cost is calculated by finding the absolute difference between the expected label and the actual confidence of an output neuron. This value is then squared and accumulated over the entire set of output neurons. In this example the two input values are 4'b0110 (actual sigmoid output) and 4'b1.000 (expected value). These numbers are fixed point and represent decimal values of 0.75 and 1.00 respectively. The difference of these two values should then of course be 4'b0100 which in decimal corresponds to 0.25. The squaring of this value should then be 0.0625 or in binary 8'b00.000001 which is what the output of the cost calculator shows in this example.

Design is able to retrieve weights and biases from a simulated memory chip

- DSSC(s) Proved: 6
- Highest Level of Design Module(s) involved: Top Level Chip
- Test Bench Expectations/Requirements:
  1. Simulate SST39LF200A flash memory chip including timing constraints and operation
  2. Store all weights and biases needed by full design
- SST39LF200A flash memory chip data sheet as reference (Appendix A Section 7.5) : `http://ww1.microchip.com/downloads/en/DeviceDoc/S71117_13.pdf`
- No pre/post processing is needed
- Main Verification Test Steps:
  1. Simulate flash memory chip in source with all needed data stored
  2. Obtain correct data with corresponding address
  3. Output correct requested data to within correct timing

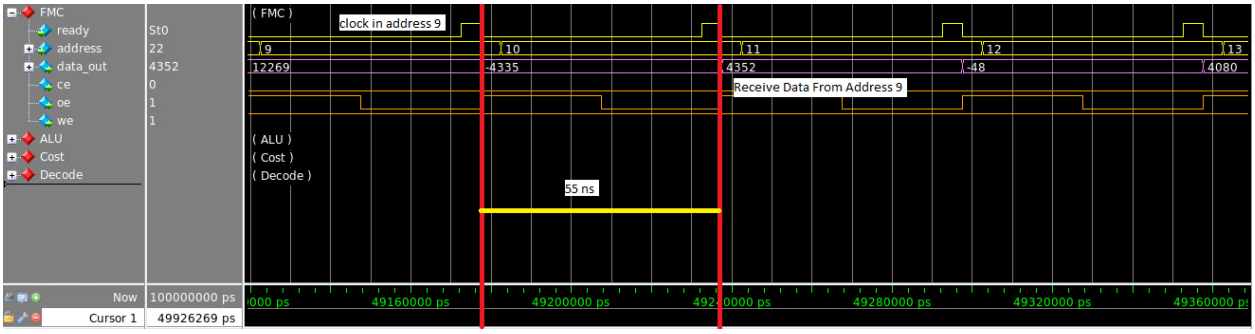


Figure 58: Example retrieval of stored weights and Biases from memory chip

The above figure shows the retrieval of data from the simulated memory chip. When we are ready to request new data we pulse the ready signal to load in the address 9. Then after about 55 ns as stated on the datasheet for the memory chip, the data at address 9 is output. In this case address 9 holds the value 4352.

Correct Error output for invalid inputs

- DSSC(s) Proved: 5
- Highest Level of Design Module(s) involved: Top Level Chip
- Test Bench Expectations/requirements:
    1. Will need to store a valid image data and expected error
- No external or premade references are needed
- No pre/post processing is needed
- Main Verification Test Steps:
    1. Send valid image data through SPI
    2. Attempt to retrieve result before calculations are finished

The SPI error output serve to provide feedback on the correct usage of the system. If a user requests the result of the network while the ALU is in progress or if the cost is requested before the cost is calculated an error code of 255 will be output on the MISO line. In Figure 59 the waveform shows the example error output if a digit is requested from the SPI controller while the ALU / Network is still running. A similar situation can be seen in Figure 60 where a cost is requested before the network has been completed and thus the cost calculated. In this example two separate bytes need to be sent. The first of which is the opcode for the cost calculation and the second being the expected label. Since in this example the ALU was still running, the SPI output the correct error code.

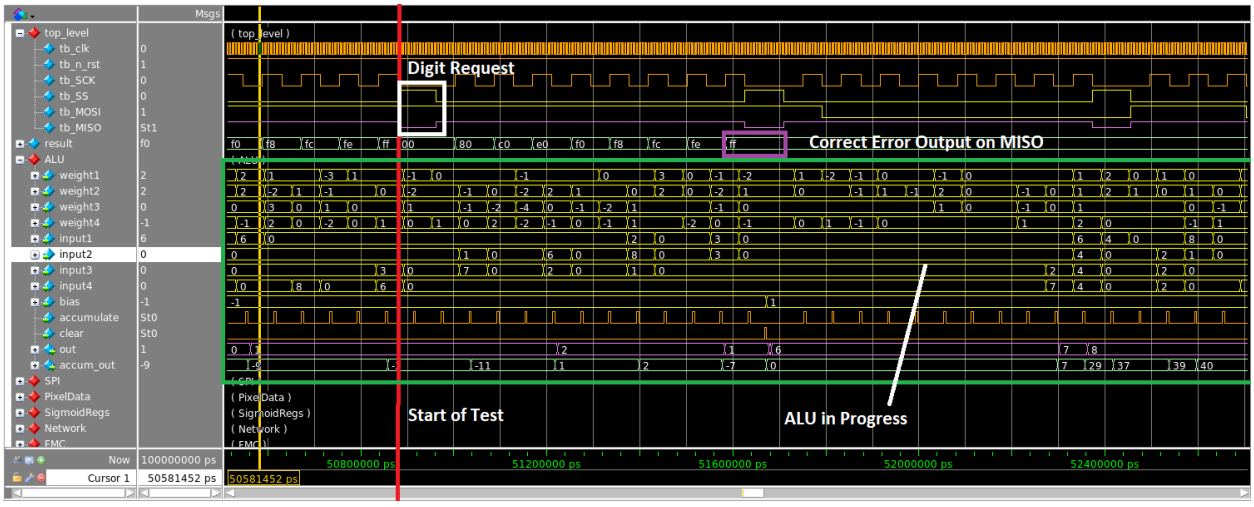


Figure 59: Example Error Output for Early Digit Request from SPI Controller

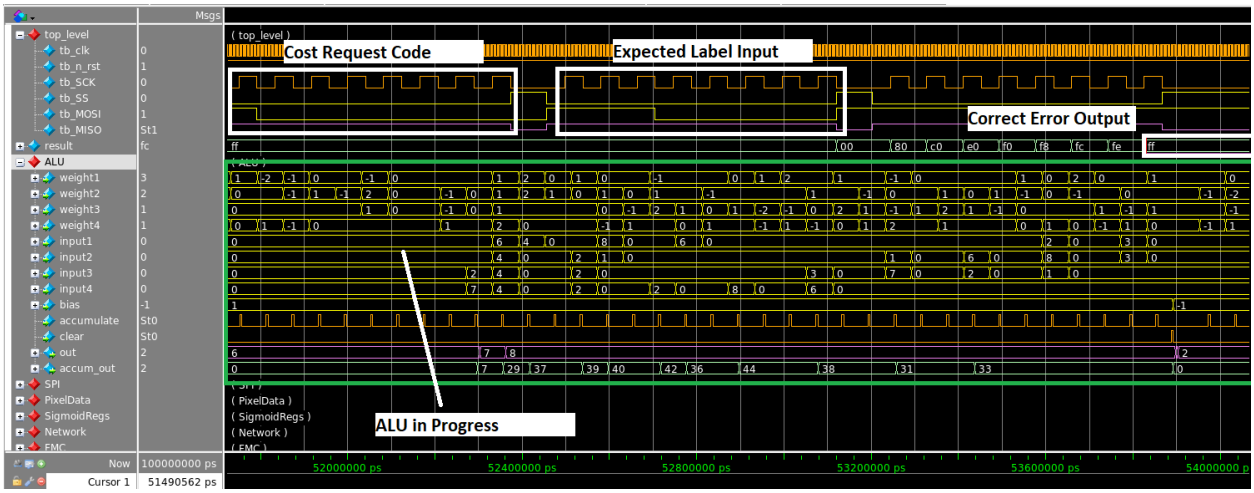


Figure 60: Example Error Output for Early Cost Request from SPI Controller

6 Project Timeline and Division of Tasks

6.1 Actual Division of Tasks

6.1.1 David Pimley

- Find and analyze costs and areas for the Cost Calculator and the Detected Digit Blocks.
- Generate Heat Maps to assist in design decisions.
- Write Cost Calculator Module and the associated test bench including python script to generate expected values.
- Write Detected Digit Module and the associated test bench including python script to generate expected values.
- Assist in Debugging ALU and Network Controller Modules.
- Created ReadMe for the use of make_concat.sh.
- Assist in creating reports and presentations.

6.1.2 Dustin Andree

- Find and analyze costs and areas for the Pixel Data Register and the Sigmoid Register Blocks.
- Write Pixel Data Register Module and the associated test bench.
- Write Sigmoid Register Module and the associated test bench.
- Write Initial Network Controller module and the associated test bench.
- Set up dummy pads and innovus.io file for innovus layout generation.
- Assist in Debugging ALU and Network Controller Modules.
- Assist in creating reports and presentations.

6.1.3 Vadim Nikiforov

- Find and analyze costs and areas for the SPI I/O Blocks.
- Write SPI Input Controller test bench.
- Write SPI Output Controller test bench.
- Write Sigmoid ALU Module and the associated test bench.
- Write script to compile entire design and the associated test bench including python script to generate the images in the required size from the MNIST database.

- Debug Innovus timing constraint errors.
- Assist in Debugging ALU and Network Controller Modules.
- Assist in creating reports and presentations.

6.1.4 Chan Weng Yan

- Find and analyze costs and areas for the Flash Memory Controller.
- Make and keep up to date RTL diagrams and state transition diagrams for the Network controller and the Sigmoid ALU.
- Write Flash Memory Controller Module and the associated test bench including finding expected outputs.
- Write SPI Input and Output Controller Module.
- Write the source code to simulate the external Flash Memory simulation.
- Assist in Debugging ALU and Network Controller Modules.
- Assist in creating reports and presentations.

7 Appendix A

This section shows the directory tree in our project folder. `/mg81/ECE337_Project/`

Note: Please use the project makefile instead of the default makefile (see Section 7.4).

7.1 Design Verilog Core Modules

1. **Top level structural Verilog code:** `source/digit_recognizer.sv`
2. **Combined Verilog code:** `source/digit_recognizer_final.sv`
3. **Cost Calculator Block:** `source/cost_calculator.sv`
   (a) **Magnitude comparator:** `source/abs_subtractor_4bit.sv`
   (b) **Helper file:** `source/adder_1bit.sv`
   (c) **Helper file:** `source/adder_nbit.sv`
   (d) **Helper file:** `source/adder_8bit.sv`
4. **Detected Digit Block:** `source/digit_decode.sv`
   (a) **Confidence level comparator:** `source/comparator.sv`
5. **Flash Memory Controller Block:** `source/fmc.sv`
6. **Network Controller Block:** `source/networkController.sv`
7. **Sigmoid ALU Block:** `source/sigmoid_ALU.sv`
   (a) **Helper file:** `source/sigmoid_ALU_4_way_adder.sv`
   (b) **Helper file:** `source/sigmoid_ALU_accumulator.sv`
   (c) **Helper file:** `source/sigmoid_ALU_multiplier.sv`
   (d) **Helper file:** `source/sigmoid_ALU_sigmoid_calculator.sv`
   (e) **Helper file:** `source/sigmoid_ALU_signed_adder.sv`
8. **Sigmoid Registers Block:** `source/sigmoidRegisters.sv`
   (a) **Helper file:** `source/sigmoidRegisters_addressableReg.sv`
9. **SPI Input Controller Block:** `source/SPI_input_controller.sv`
10. **SPI Output Controller Block:** `source/SPI_output_controller.sv`
11. **Pixel Data Registers:** `source/pixelData.sv`
    (a) **Helper file:** `source/pixelData_PTPSR.sv`
12. **Generic Components:**

(a) **Flexible Counter with Controlled Rollover:** `source/flex_counter.sv`
(b) **Positive Edge Detector:** `source/gen_pos_edge_detect.sv`
(c) **Parallel-to-serial Shift Register:** `source/gen_pts_sr.sv`
(d) **Serial-to-parallel Shift Register:** `source/gen_stp_sr.sv`
(e) **Reset to Logic Low Synchronizer:** `source/gen_sync.sv`
(f) **8-bit Multiplier:** `source/mult_4bit.sv`

7.2 Test Benches

1. **External Flash Memory source simulator:** `source/external_fm.sv`
2. **Cost Calculator Block testbench:** `source/tb_cost_calculator.sv`
3. **Detected Digit Block testbench:** `source/tb_digit_decode.sv`
4. **Top Level testbench:** `source/tb_digit_recognizer.sv`
5. **Combined testbench:** `source/tb_digit_recognizer_final.sv`
6. **Flash Memory Controller testbench:** `source/tb_fmc.sv`
7. **Generic Positive Edge Detector testbench:** `source/tb_gen_pos_edge_detect.sv`
8. **Network Controller testbench:** `source/tb_networkController.sv`
9. **Pixel Data Registers Block testbench:** `source/tb_pixelData.sv`
10. **Sigmoid ALU Helper testbench:** `source/tb_sigmoid_ALU_4_way_adder.sv`
11. **Sigmoid ALU Helper testbench:** `source/tb_sigmoid_ALU_multiplier.sv`
12. **Sigmoid ALU Helper testbench:** `source/tb_sigmoid_ALU_sigmoid_calculator.sv`
13. **Sigmoid ALU testbench:** `source/tb_sigmoid_ALU.sv`
14. **Sigmoid Registers Block testbench:** `source/tb_sigmoidRegisters.sv`
15. **SPI Input Controller testbench:** `source/tb_SPI_input_controller.sv`
16. **SPI Output Controller testbench:** `source/tb_SPI_output_controller.sv`

7.3 Test Vector Files

1. **Expected Cost Outputs:** `docs/cost_outputs.txt`
2. **Hidden Biases from Flash Memory:** `docs/hidden_biases.txt`
3. **Hidden Weights from Flash Memory:** `docs/hidden_weights.txt`
4. **Image Pixel Data:** `docs/images.txt`
5. **Output Biases from Flash Memory:** `docs/output_biases.txt`
6. **Output Biases from Flash Memory:** `docs/output_weights.txt`

7.4 Makefiles and Readme

1. **Readme on Project Makefile Usage:** `docs/Script_Documentation.txt`
2. **Default Makefile:** `makefile`
3. **Project Makefile:** `make_concat.sh`

7.5 Reports and Datasheet

1. **External Flash Memory Datasheet** `docs/flashmem_datasheet.pdf`
2. **Final Presentation** `docs/finalPresentation.pdf`
3. **Final Report** `docs/finalReport.pdf`

7.6 Transcripts, Evidences, Logfiles

1. **Complete Output Transcript:** `transcripts/10000_imgs.txt`
2. **Results for a 100 images mapped compilation while testing cost:** `transcripts/100_imgs_cost_mapped.png`
3. **Results for a 100 images mapped compilation while testing cost:** `transcripts/100_imgs_cost_mapped.txt`
4. **Results for a 100 images source compilation while testing cost:** `transcripts/100_imgs_cost_source.png`
5. **Results for a 100 images source compilation while testing cost:** `transcripts/100_imgs_cost_source.txt`
6. **Results for a 100 images mapped compilation without testing cost:** `transcripts/100_imgs_nocost_mapped.png`
7. **Results for a 100 images mapped compilation without testing cost:** `transcripts/100_imgs_nocost_mapped.txt`

8. **Results for a 100 images source compilation without testing cost:** `transcripts/100_imgs_nocost_source.png`
9. **Results for a 100 images source compilation without testing cost:** `transcripts/100_imgs_nocost_source.txt`
10. **Results for a 100 images mapped compilation while testing error codes:** `transcripts/errors_mapped.png`
11. **Results for a 100 images mapped compilation while testing error codes:** `transcripts/errors_mapped.txt`
12. **Results for a 100 images source compilation while testing error codes:** `transcripts/errors_source.png`
13. **Results for a 100 images source compilation while testing error codes:** `transcripts/errors_source.txt`

7.7 Innovus Reports

1. **Final Synthesis Logfile** `digit_recognizer_final.log`
2. **Summary Report text file:** `summaryReport/digit_recognizer_final.main.htm.ascii`
3. **Summary Report html file:** `summaryReport/digit_recognizer_final.main.htm`
4. **Timing Report:** `timingReports/digit_recognizer_final_postRoute_all.tarpt.gz`
5. **Timing Report Summary:** `timingReports/digit_recognizer_final_postRoute.summary.gz`
6. **Geometry Report:** `digit_recognizer_final.geom.rpt`
7. **Connectivity Report:** `digit_recognizer_final.conn.rpt`
8. **Mapped Report:** `reports/digit_recognizer_final.rep`

List of Figures

List of Tables

References Cited

[1] "Mnist for ml beginners," 6 2017. [Online]. Available: https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners

[2] *2 Mbit/4 Mbit/8 Mbit (x16) Multi-Purpose Flash*, Silicon Storage Technology, Inc., 4 2011.

[3] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[4] "Design success criteria and design budgeting," Purdue University, 3 2017.