Handwritten Digit Recognizer using Neural Networks

Phase 1 Project Proposal
TA: Anirudh
Wednesday | 11:30

David Pimley
Chan Weng Yan
Dustin Andree
Vadim Nikiforov

ECE 33700
2/12/2018

Purdue University
Computer Engineering

1 Executive Summary

This project idea is a digit recognizer. It is designed to work with images consistent with the MNIST dataset, so it accepts 14×14 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
- 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

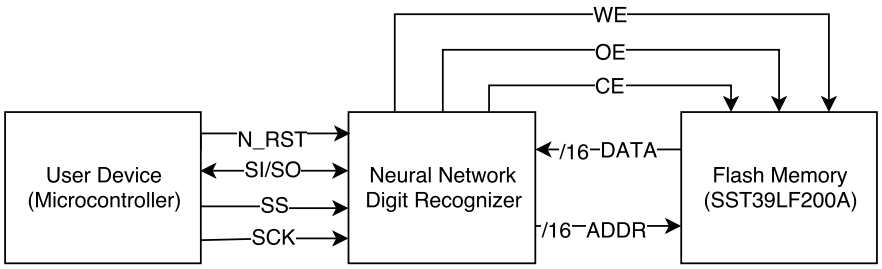

Figure 1: Example system usage diagram of neural network digit recognizer

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. This is used to store the weights and biases for the neural network, and is used to prevent the need to reupload 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 Neural Network Block, the control functionality comes from the user device performing the following tasks:

1. If necessary, reset the registers (everything but the Flash Memory) 2
2. 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 that the weights and bias will be changed in which case the registers will need to be flushed.
   (e) 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.

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
  - 16 bit signed addition used for bias addition to each weighted pixel.
- Fixed Point Multiplication
  - Extension of Fixed Point Addition, used to multiply matrices in parallel within the digit recognizer to speed up processing.
- Sigmoid Function
  - Used for the output of the sigmoid neuron registers, calculated using a lookup table.

Signals:

Command Code	Description
`3'b000`	New image data is being input over MOSI
`3'b001`	New expected label is being input over MOSI
`3'b010`	Output the cost of the last operation
`3'b011`	Output the result of the last operation
`3'b100`	The weights/biases have changed, flush registers

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: Miscillaneous 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 system's ALU performs the necessary computations for a sigmoid neuron, visually depicted below in Figure 2. 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.

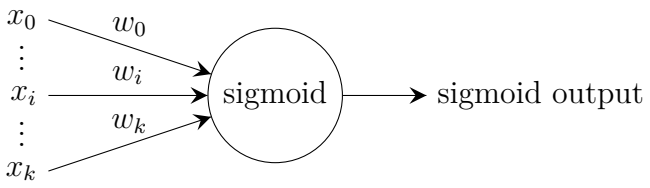

Figure 2: A top-level represention of a sigmoid neuron with k inputs

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

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

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

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

Where x is the output of the neuron before going through the sigmoid function.

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 as

$$C(w, b) \equiv \frac{1}{2n} \sum_x |y(x) - a|^2 \tag{3}$$

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. n represents the total number of training inputs run. 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 8-bit binary fraction between 0 and 1.

$y(x)$ 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 Flash Memory Timing Requirements

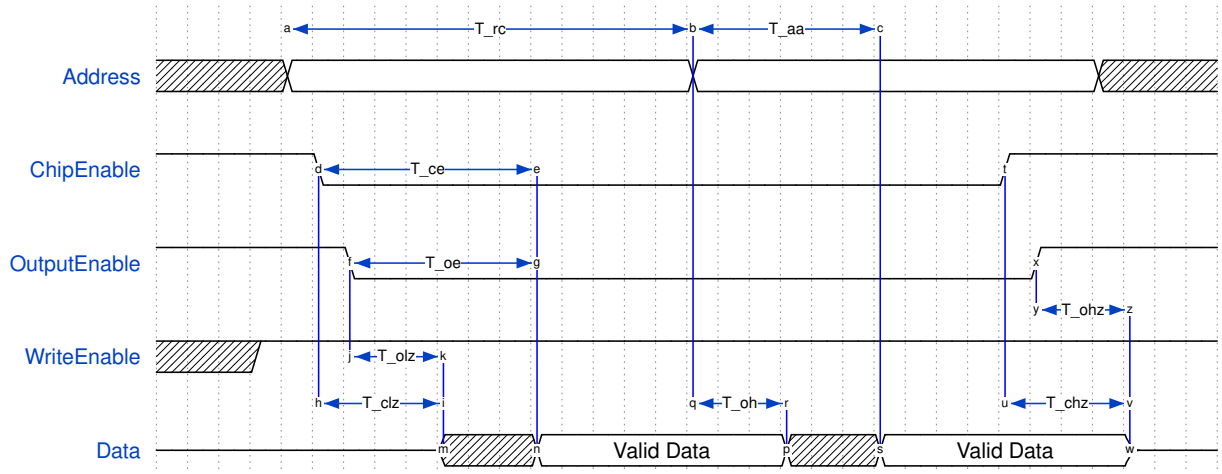

Figure 3: Flash Memory Timing Diagram

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

Figure 4: 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 which could either be two 8-bit weights or two 8-bit bias values.

2.3.4 SPI Timing Requirements

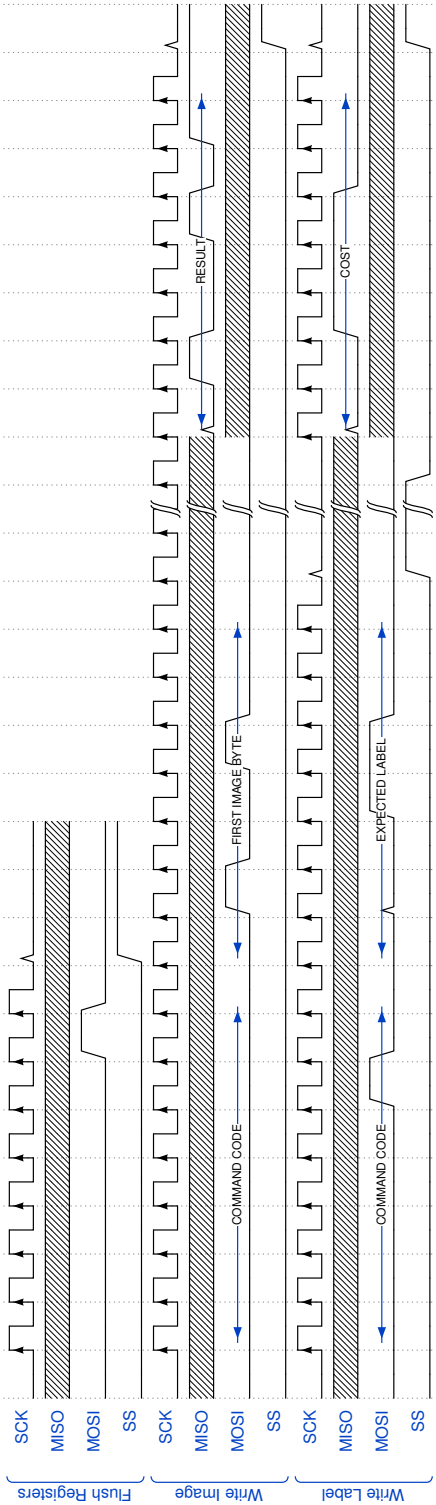

Figure 5: SPI Timing Diagram

2.4 Requirements

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 14×14 pixel grayscale images with each pixel being represented by a byte. Standard MNIST images are originally 28×28 pixel, 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 train the circuit, however, 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 clock which will allow us to have a baud rate of 12 Mb/s, or in our design:

$$\frac{12 \cdot 10^6 \cdot \text{bits/second}}{14 \cdot 14 \cdot 8 \text{ bits/image}} = 7653 \ \frac{\text{images}}{\text{second}}$$

The main bottleneck of our design will come from the SRAM 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 SRAM is 55 ns which is for each 16-bit word accessed.

3 Design Architecture

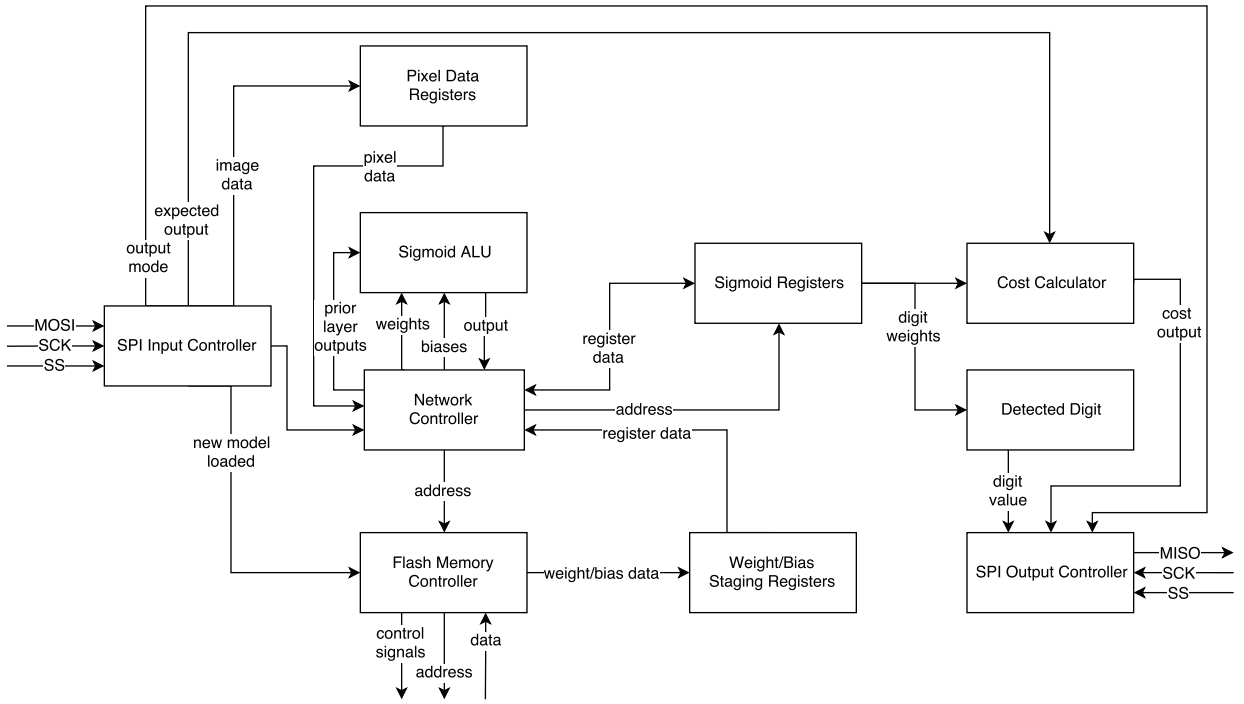

Figure 6: Digit recognition circuit design architecture

The design architecture of the digit recognition circuit is shown in Figure 6. 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.