FPGA BASED HARDWARE IMPLEMENTATION OF SPIKING NEURAL NETWORK FOR DIGIT RECOGNITION AND EMBEDDED FPGAS FOR SOC DESIGNS

A Project Report

submitted by

KORIVI RAVICHANDRA

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

JUNE 2017

THESIS CERTIFICATE

This is to certify that the thesis titled **FPGA BASED HARDWARE IMPLEMENTATION OF SPIKING NEURAL NETWORK FOR DIGIT RECOGNITION AND EMBEDDED FPGAS FOR SOC DESIGNS**, submitted by **Korivi Ravichandra**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Nitin Chandrachoodan
Project Guide
Associate Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: June 23, 2017

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude towards prof. Nitin Chandrachoodan for guiding me throughout the journey of this project.

ABSTRACT

Human Brain is an extremely efficient machine in solving problems such as image recognition because of their parallel, power efficient, fault tolerant and distributed network architecture. Artificial Neural Networks (ANNs), inspired by brain have found wide range of applications in the field of image recognition. Software based ANNs are inefficient because of the huge amount of computational power required and power consumption. Recent trend is to look for models which are more biologically plausible. Spiking Neural Networks (SNNs) are the third generation of neural networks which closely mimic biological neural networks by incorporating some of the biological mechanisms.

SNNs are event-based and asynchronous in nature. Because of this, state variables or parameters of neurons and synapses which are basic building of the network need not be evaluated at every time step but only when they are addressed by an event, thereby saves lot of power.

In this work, we will implement an FPGA based hardware of Spiking Neural Network for MNIST handwritten digit Recognition.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

SNN	Spiking Neural Networks
CNN	Convolutional Neural Networks
ANN	Artificial Neural Networks
STDP	Spike Timing Dependent Plasticity
FPGA	Field Programmable Gate Array
eFPGA	embedded FPGA
ASIC	Application Specific Integrated Circuit
LFSR	Linear Feedback Shift Register

NOTATION

v_m	Membrane potential of the neuron in volts
v_{rest}	Resting potential of the neuron in volts
v_{th}	Threshold potential of the neuron in volts
v_{reset}	Reset potential of the neuron in volts
t_{refr}	Refractory period of the neuron in seconds
g_e	Excitatory synapric conductance in $\mho$
g_i	Inhibitory synapric conductance in $\mho$
w	weight of synapse in $\mho$
E_{exc}	Equilibrium potential of the excitatory synapse in votls
E_{inh}	Equilibrium potential of the inhibitory synapse in volts

CHAPTER 1

INTRODUCTION

Becuase of their event based nature, SNNs are best suited for event-based daata. If we consider conventional vision sensors (6), they capture series of frames which contain enormous amounts of redundant data, wasting memory and computational resources to process them. Whereas data captured by the dynamic vision sensors contain incremental changes in frames in the form of streams of events, enabling SNNs to process this data with less computational power, data storage, and power consumption than ANNs.

1.1 Problems with Implementation

Eventhough SNNs have found to be as computationally powerful as conventional ANNs, practical SNNs implementations have not reached same accuracy levels as their counterparts on many machine learning tasks such as MNIST digit recognition. The main reason for this is non-differential nature of spike signals in SNNs but differential activation functions are vital to the learning using back propegation used in conventional ANNs.

Recently a technique called rate-based learning has been proposed by (1) in which different data representations were used between training and testing by training a convolutional ANN and developing conversion algorithms which convert the trained weights into equivalent SNN weights. However, in these methods, details of statistics in spike trains which are required for processing event-based senor data can not be precisely represented by the data representation used for training.

There is an other kind of technique called spike-based learning in which learning directly works on spike trains by devising biologically plausible learning rules such as STDP (2). However using this method, training is possible to a single layer using unsupervised learning techniques.

Another problem in implementing SNNs is the lack of available frame-free, event-based data sets. Full potential of the SNN cannot be assessed unless we have a data set which is tuned to work on ANN i.e., event-based data.

CHAPTER 2

NEURON AND SYNAPSE MODELS

2.1 Basic definitions

A biological neuron is surrounded by a cell membrane which always maintains a potential difference between its interior and its surroundings called the membrane potential of the neuron v_m. When stimulus to neuron is taken off, at steady state neuron's membrane potential reaches a value called the resting potential v_{rest} of the neuron, which is around $-65\ mV$. When a positive stimulating current is injected into the neuron, its membrane potential will increase and if it increases beyond a certain value called threshold potential $v_{th} \approx -52\ mV$, neuron generates a spike and resets to a voltage called reset potential $v_{reset} \approx -70\ mV$. Once neuron generates a spike, the neuron can not generate another spike with any amount of stimulus, until a certain duration of time is elapsed called the refractoty period $t_{refr} \approx 1\ ms$. The average number of spikes over long time that a neuron generates is called the firing rate f of the neuron, which is around $10\ Hz$.

2.2 An Isolated Neuron

Because of the charge building dielectric nature of the membrane, it acts like a capacitor C. Also the membrane is not a perfect dielectric, buts its charge leaks over time. Therefore the membrane acts also like a resistor of resistance R which keeps on dissipating its charge. So, the neuron can be described by the following differential equation:

$$\tau_m \frac{dv_m}{dt} = v_{rest} - v_m \tag{2.1}$$

where τ_m, v_m, v_{rest} are the time constant, membrane potential, and resting potential of the membrane respectively. Also, $\tau_m = R.C$, is approximately $20\ ms$ for biological

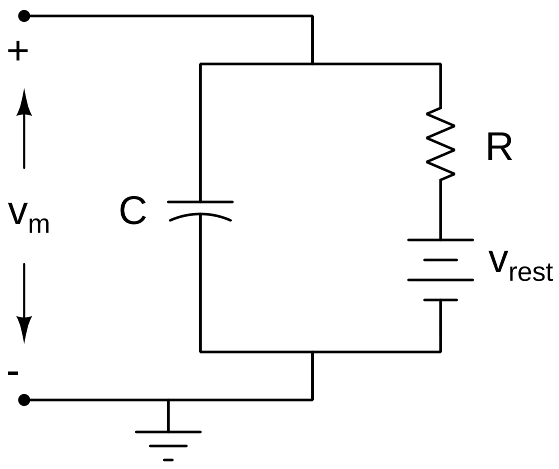

Figure 2.1: Electrical equivalent circuit of an isolated neuron

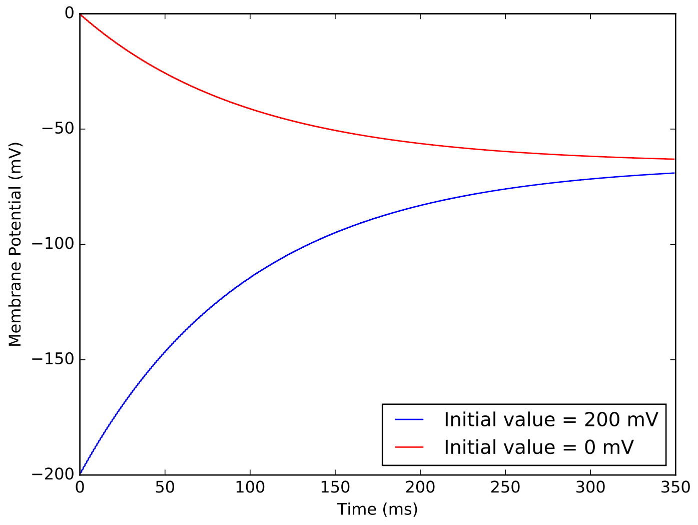

Figure 2.2: Response of an isolated neuron

neurons. From equation 2.1 an equivalent electrical circuit can also be obtained as shown in figure 2.1.

So, an isolated neuron reaches steady state when its membrane potential is equal to its resting potential. Figure 2.2 shows the response of an isolated neuron with two different starting values for the membrane potential. It is clear from the plot that the neuron reaches steady state when v_m is equal to v_{rest}.

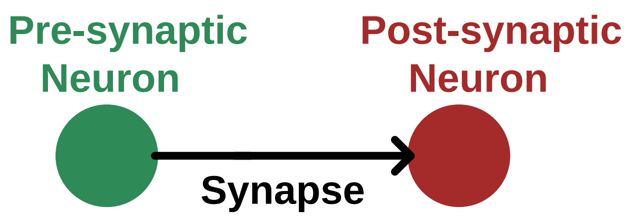

Figure 2.3: Synapse connecting two neurons

2.3 Model of a Synapse

The neuron which drives a synapse is called the pre-synaptic neuron and the synapse which is driven by the synapse is called the post-synaptic neuron.The electrical equivalent of a synapse has a battery of value equal to the equilibrium potential of the synapse, E, in series with the variable resistor of conductance g as shown in the figure 2.4. Where

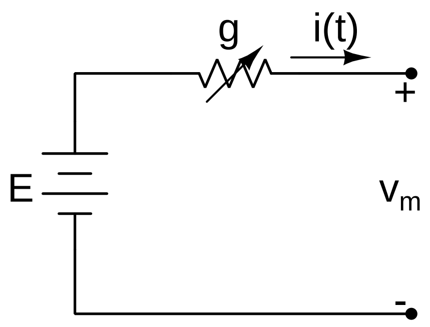

Figure 2.4: Electrical equivalent circuit of a synapse

v_m is the membrane potential of the post-synaptic neuron and $i(t)$ is the stimulating current injected into the post-synaptic neuron.

When a pre-synaptic neuron generates a spike, the conductance of the synapse increases instantaneously by a value equal to the weight w of the synapse. Otherwise, the conductance keeps on decaying exponentially with time described the equation 2.2.

$$\tau \frac{dg}{dt} = -g \tag{2.2}$$

The stimulating current $i(t)$ injected into the post-synaptic neuron by the synapse is given by the equation 2.3.

$$i(t) = g.(E - v_m) \tag{2.3}$$

There are two kinds of synapses namely excitatory and inhibitory synapses depending on the polarity of $i(t)$.

2.3.1 Excitatory Synapse

For an excitatory synapse, the value of equilibrium potential E_{exc} and the time constant τ_e are approximately $0\ V$ and $1\ ms$ in biological neural networks. So, equations which govern the variation of excitatory synapse conductance are give below.

$$\begin{aligned} g_e &= g_e + w; \ \texttt{when pre-synaptic neuron fires} \\ \tau_e \frac{dg_e}{dt} &= -g_e; \ \texttt{Otherwise} \end{aligned}$$

The current supplied by the excitatory synapse is given by

$$\begin{aligned} i(t) &= g_e.(E_{exc} - v_m) \\ &= g_e.(0 - v_m) \\ &= -g_e.v_m \end{aligned}$$

As will be discussed later, membrane potential v_m always varies between its thereshold voltage v_{th} and reset voltage v_{reset}. The typical values of v_{th} and v_{reset} are $-52\ mV$ and $-65\ mV$ respectively. So, the effect of a non-zero conductance g_e is that the current is always injected into the post-synaptic neuron, thereby increases its potential.

2.3.2 Inhibitory Synapse

For an inhibitory synapse, the value of equilibrium potential E_{inh} and the time constant τ_i are approximately $-100\ mV$ and $2\ ms$ in biological neural networks. So, equations which govern the variation of inhibitory synapse conductance are give below.

$$\begin{aligned} g_i &= g_i + w; \ \texttt{when pre-synaptic neuron fires} \\ \tau_i \frac{dg_i}{dt} &= -g_i; \ \texttt{Otherwise} \end{aligned}$$

The current supplied by the excitatory synapse is given by

$$
\begin{aligned}
i(t) &= g_i.(E_{inh} - v_m) \\
&= g_i.(-0.1 - v_m)
\end{aligned}
$$

So, the effect of a non-zero conductance g_i is that the current is always drawn out of the post-synaptic neuron, thereby decreases its potential.

2.4 Neuron connected to both types of synapses

Let us analyse the response of a neuron connected to both types of synapses as shown in figure 2.5. The electrical equivalent circuit of this network is given in the figure 2.6.

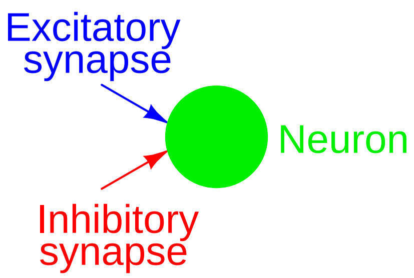

Figure 2.5: Neuron connected to both types of synapses

The current $i(t)$ injected into the neuron by both types of synapses is given by

$$
\begin{aligned}
i(t) &= g_i.(E_{inh} - v_m) + g_e.(E_{exc} - v_m) \\
&= g_i.(-0.1 - v_m) - g_e.v_m
\end{aligned}
$$

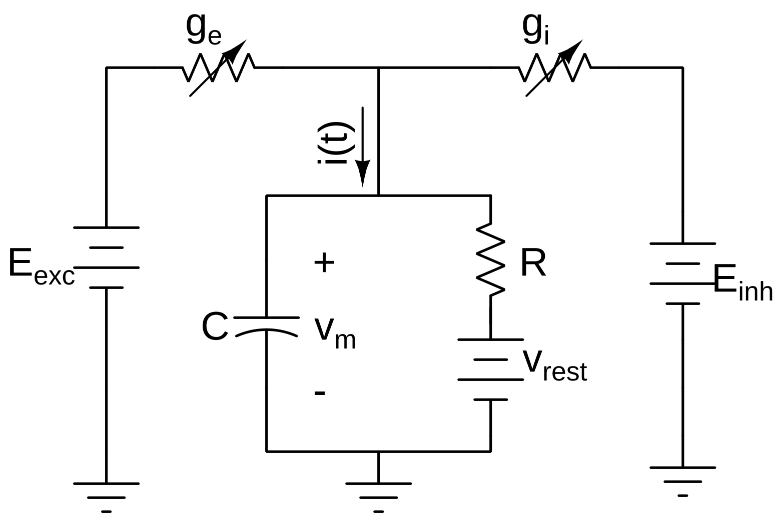

Figure 2.6: Electrical equivalent circuit of a Neuron connected to both types of synapses

So, the membrane potential v_m of the neuron is described the following differential equation

$$\begin{aligned}\tau_m \frac{dv}{dt} &= v_{rest} - v + i(t) \\ &= v_{rest} - v + g_i.(-0.1 - v_m) - g_e.v_m\end{aligned}$$

In biological neurons when the membrane potential reaches a certain potential known as the threshold potential v_{th} of the membrane, neuron generates a spike instantaneously and resets to a reset voltage v_{reset} and cannot generate a spike for a certain duration of time called the refractory period t_{refr} of the neuron. Typical approximate values of v_{th}, v_{reset} and t_{refr} are $-52\ mV$, $-70\ mV$ and $1\ ms$ respectively.

Since isolated spikes of a given neuron look alike, the form of the membrane potential does not carry any information. Rather, it is the number and the timing of spikes which matter.

The response of the circuit shown in figure 2.6 is shown in figure 2.7. The chosen parameter values are $v_{rest} = -65\ mV$, $v_{reset} = -70\ mV$, $v_{th} = -52\ mV$, $t_{refr} = 5\ ms$, $\tau_m = 100\ ms$, $\tau_e = 10\ ms$, $\tau_i = 5\ ms$, $w_e = 1\ \mho$, $w_i = 5\ \mho$.

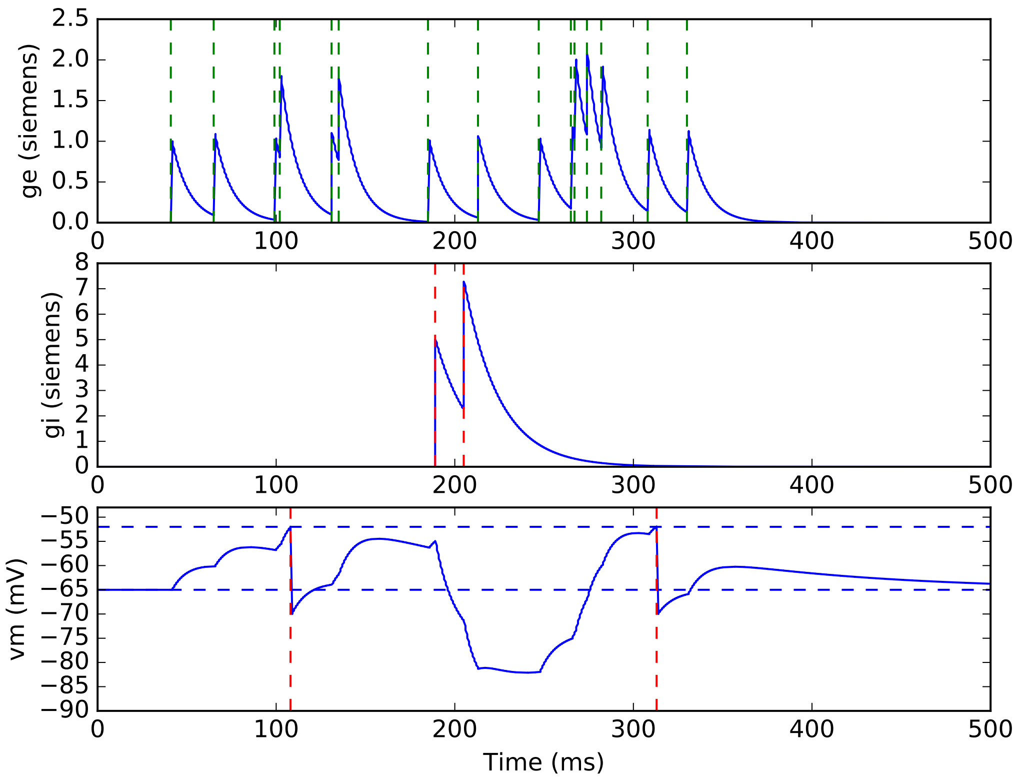

Figure 2.7: Nueron Response

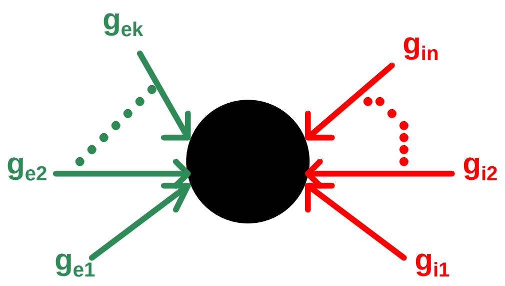

Figure 2.8: Nueron in a large network

2.4.1 Neuron Embedded in a large Network

If a neuron embedded in a large neural network has incoming excitatory synapses with conductances g_{e1}, g_{e2}, ..., g_{ek} and inhibitory synapses with conductances g_{i1}, g_{i2}, ...,

g_{im}, then the total stimulating current $i(t)$ is given by

$$\begin{aligned} i(t) &= -v_m.(g_{e1} + g_{e2} \ + ... + \ g_{en}) + (g_{i1} + g_{i2} + ... + \ g_{in}).(E_{inh} - v_m) \\ &= -v_m. \sum_{k=1} g_{ek} + (E_{inh} - v_m). \sum_{n=1} g_{in} \\ &= -v_m.g_{eacc} + (E_{inh} - v_m).g_{iacc} \end{aligned}$$

where $g_{eacc} = \sum_{k=1} g_{ek}$ and $g_{iacc} = \sum_{n=1} g_{in}$. The solution of the above differential equation is given by the following equation:

$$v_m = \begin{cases} (1 - exp(-\frac{1+g_{eacc}+g_{iacc}}{\tau_m}t).\frac{v_{rest}+g_{iacc}.E_{inh}}{1+g_{iacc}+g_{eacc}}, & \text{if } v_m < v_{th}; \\ v_{reset}, & \text{if } v_m > v_{th} \text{ and } t_{refr} \text{ elapsed;} \end{cases}$$

CHAPTER 3

MNIST DIGIT RECOGNITION PROBLEM

3.1 MNIST Database

MNIST is a database of handwritten digits, available from (5), which has a training set of 60000 examples, and a testing set of 10000 examples, with images centered in 28x28 pixel grid and pixel values ranging from 0 to 255. This database has been widely used as a benchmark in research concerning Image recognition.

3.2 MNIST Digit Recognition using Convolutional Networks

In Deep Neural Networks, the first layer learns primitive features, such as an edge in an image. Second layer learns more complex features, like a corner in an image. The process is repeated in successive layers until the system can reliably recognize images. Convolutional Neural Network (CNN), the most successful deep neural network arhi-

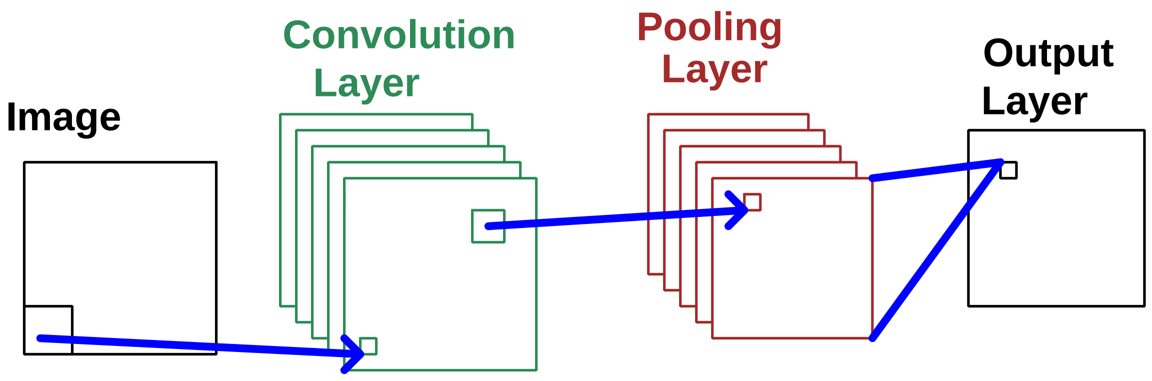

Figure 3.1: CNN architecture

tecture for image recognition, has basically three set of layers namely, convolutional, pooling and an output layer and is shown in figure 3.1. There is an input layer of 28x28 neurons which encode the pixel intensities of the MNIST image, which is followed by a convolutional layer with m feauture maps and nxn kernel associated with each feature map. So, if we choose a stride distance of 1, then convolutional layer consists of mx$(28 - n + 1)$x$(28 - n + 1)$ neurons. Each receptive field in a feature map has same set of weights and biases. The convolutional layer is followed by a pooling layer, and finally, there is an output layer of 10 neurons with fully connected connections from the pooling layer to classify the image. Backpropegation is the most widely used algorithm to train these networks.

CHAPTER 4

METHODOLOGY

Input to SNN should be in the form of spikes. So, the first step is to convert image into spike data. We used Poisson process to convert frame-based data to spike-based data. We used a python based simulator called Brian (3) to train our proposed network. Then we analyzed the accuracy of the network while applying different optimizations on the trained weights. Then we finally implemented the proposed network for MNIST handwritten digit recognition in hardware.

CHAPTER 5

IMAGE PRE-PROCESSING

The process of converting real world information such as intensity of pixel in an image to spike trains is called neural encoding. Each pixel in an image corresponds to a randomly distributed spike train with an average firing rate proportional to the intensity of that pixel. Here, firing rate is the number of spikes per unit time. We, use poisson distribution to generate the spike train as described in (4).

Let f be the required average firing rate of the poisson distributed spike train, then the average time period of the spike train is

$$T = \frac{1}{f}$$

In otherwords, T is the time duration between two spikes. That means, on an average, in time duration T, we can have at most one spike. If we choose simulation time step $\triangle t$ such that

$$\triangle t << T$$

then $\frac{\triangle t}{T} = f\triangle t$ gives the probability of spike occurrence in time $\triangle t$. If f_{max} is the maximum possible firing rate, then we need to choose $\triangle t$ such that

$$\triangle t << \frac{1}{f_{max}}$$

$$f_{max}\triangle t << 1$$

So, at each time step we will compare $\triangle t.f$ with a random number between 0 and 1. If the random number is less than $f.\triangle t$, then there is spike occurrence in that time step, otherwise there is no spike in that time step.

If $1/f_{max}$ or T_{min} approaches $\triangle t$, the generated spike train is not any more random in nature but uniformly distributed spike train. So, higher average firing rate will generate uniformly distributed spike train, which will in effect reduce the accuracy of recognition because spike trains generated by biological neurons are random in nature.

If we reduce $\triangle t$ to compensate for increase in f, we will have to simulate the network over many time steps or bins. Also f should not be too small because network will not be able to recognize the image. So, there is a clear trade-off between $\triangle t$ and f. The raster plot for an average firing rate 06 63.75 Hz is shown in figure 5.1.

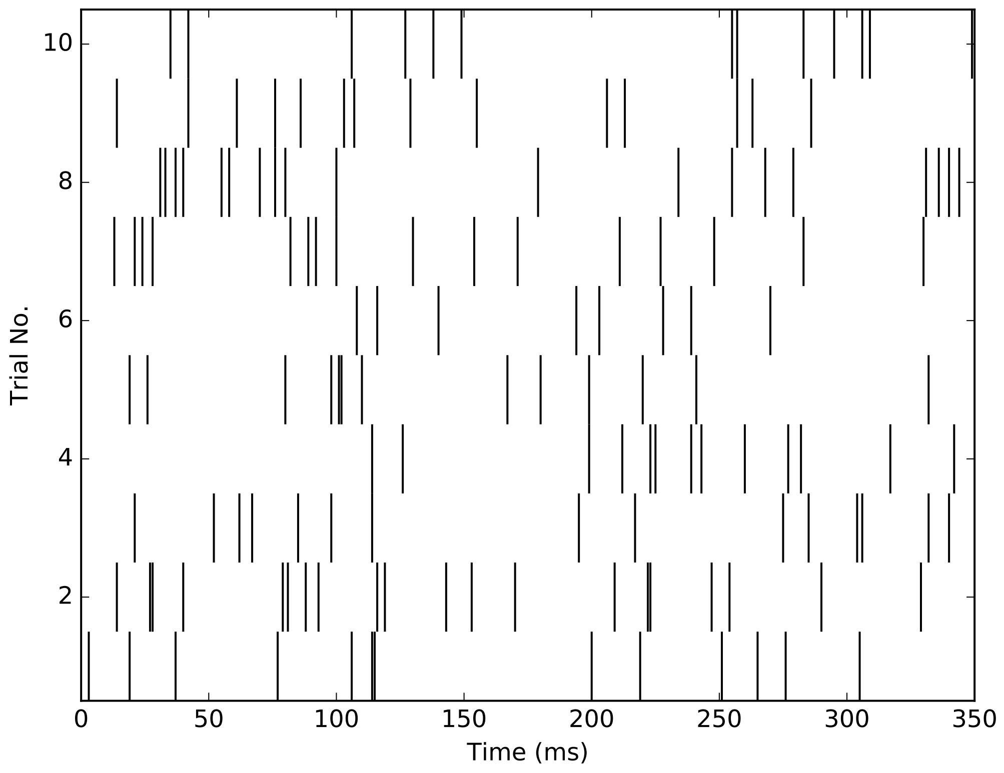

Figure 5.1: Raster plot for an average firing rate of 63.75 Hz

In the hardware implementation, we used 32-bit linear feedback shift register (LFSR) to generate random numbers. A 3-bit LFSR is shown in figure 5.2. Also, in our hardware implementations, we used a time step of 0.5 ms.

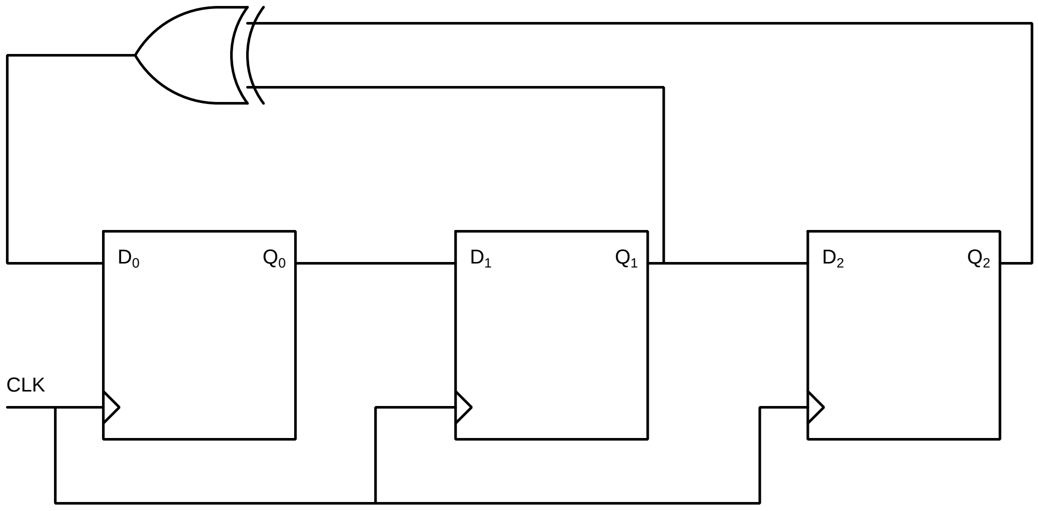

Figure 5.2: Linear Feedback Shift Register

CHAPTER 6

SNN ARCHITECTURE

The arhitecture and learing method are adopted from (2). There are 784 input neurons which carry no model but generate spike trians with an average frequency proportional to the pixel value of that input.

There is one layer of excitatory neurons. Input and excitatory layer of neurons are connected by excitatory synapses in all-to-all fashion.

There is one layer of inhibitory neurons and the number of inhibitory neurons is equal to the excitatory neurons. Here the inhibitory neuron also does not carry any model but generates a spike whenever its corresponding pre-synaptic excitatory neuron fires. There are excitatory synapses from excitatory neurons to inhibitory neurons in one-to-one fashion. Also there are inhibitory connections from the inhibitory neurons to the

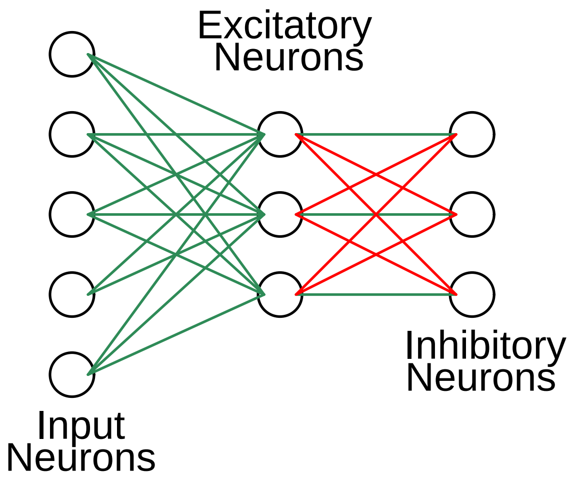

Figure 6.1: Network Architecture

the excitatory neurons connected in $i \mathrel{!}= j$ fashion, i.e, each excitatory neuron receives connections from all inhibitory neurons except from the one which it drives. Like input neuron, inhibitory neuron also carry no model but generates a spike whenever its corresponding driving excitatory neuron fires. So, whenever an excitatory neuron fires, membrane potential of all other neurons drops through the inhibitory connections from the inhibitory neuron corresponding to the fired excitatory neuron. This mechanism creates a competition among all excitatory neurons. Figure 6.1 shows a sample architecture with 5, 3 and 3 input, excitatory and inhibitory neurons respectively. Green and red colours in the figure represent forward and backward connections respectively.

6.1 Learning

During learning synapses act in two ways. They act like variable resistors and thus transmit spikes with a variable conductance. Additionally, they adapt weights depending on the activity of the neurons to which they are connected and thus can learn. All synapses from input neurons to excitatory neurons are learned using STDP as explained in (2). Each synapse keeps track of three values namely its weight w, pre-synaptic trace a_{pre}, which models the recent pre-synaptic spike history. Every time a pre-synaptic neuron fires, a_{pre} increases by 1, otherwise decays exponentially with time constant τ_{pre} as described by the equation 6.1.

$$\tau_{pre}\frac{da_{pre}}{dt} = -a_{pre} \tag{6.1}$$

When a post-synaptic neuron fires, the change in the weight of the synapse $\triangle w$ is given by equation 6.2

$$\triangle w = \eta(a_{pre} - a_{tar})(w_{max} - w)^{\mu} \tag{6.2}$$

where η is the learning rate, w_{max} is the maximum weight, and μ determines the dependence of the update on previous weight, a_{tar} is the target value of the pre-synaptic trace at the moment of a post-synaptic spike. So, when a synapse does not receive sufficient number of pre-synaptic spikes, the value of $a_{pre} - a_{tar}$ will be negative, and $\triangle w$ is negative. this mechanism ensures that pre-synaptic neurons that rarely lead to firing of the post-synaptic neuron will become more and more disconnected.

We used a python-based simulator called Brian (3) to train our network and obtain trained weights.

Brian(3) is a free, open source simulator for spiking neural networks. It is written in the Python programming language and is available on almost all platforms

6.2 Assigning Class to Excitatory Neurons

After the training is over, we keep the obtained weights constant and will present the entire training set to the network and assign each neuron a class, based on its highest average number of spikes to the ten classes of digits.

6.3 Testing

The activity of the class-assigned network is then used to measure the classification accuracy of the network. Each test image is presented to the network, and the predicted image is the class which has the highest average number of spikes over all its neurons. This type of learning, where network learns by itself is called unsupervised learning.

We variation of performance accuracy while varying the number of excitatory neurons is shown in table 6.1.

No. of Neurons	**Performance Accuracy (%)**
50	74.68
75	80.60
100	81.09
400	91.63

Table 6.1: Accuracy vs No. of Excitatory Neurons

CHAPTER 7

ALGORITHMS AND HARDWARE IMPLEMENTATION

7.1 Basic Version

Algorithm 1 Network

```
1: procedure NETWORK($image, nSpikes, scalingFactor$)
2:     $isInhFired[i] = 0$ for all $i$ in $nExcNeurons$
3:     $g_e[i][j] = 0$ for all $i$ in $nExcNeurons$ and $j$ in $nInputs$
4:     $g_i[i][j] = 0$ for all $i$ in $nExcNeurons$ and $j$ in $nInhNeurons$
5:     for $i = 0 \to nBins$ do
6:         for($i = 0 \to nExcNeurons$) $accG_e[i] = 0$
7:         for $j = 1 \to nInpNeurons$ do
8:             if(ISINPFIRED($image[j], scalingFactor$)) for ($k = 1 \to nExcNeurons$) $g_e[k][j] \leftarrow g_e[k][j] + weightMat[k][j]$
9:             else for ($k = 1 \to nExcNeurons$) $g_e[k][j] \leftarrow g_e[k][j] - (g_e[k][j]/\tau_e).\triangle t$
10:            for ($i = 1 \to nExcNeurons$) $accG_e[i] \leftarrow accG_e[i] + g_e[k][j]$
11:            if($i \Leftrightarrow 0$) $newImageFlag \leftarrow true$
12:            else $newImageFlag \leftarrow false$
13:            ExcNeuron($accG_e$, $accG_i$, $isInhFire$, $nSpikes$, $newImageFlag$)
14:            $accG_i[i] = 0$ for all $i$ in $nExcNeurons$
15:        end for
16:        for $j = 1 \to nInhNeurons$ do
17:            if($isInhFired[j]$) for($k = 0 \to nExcNeurons$) $g_i[k][j] = g_i[k][j] + 17$
18:            else $g_i[k][j] = g_i[k][j] - (gi[k][j]/\tau_i).\triangle t$
19:            $g_i[j][j] \leftarrow 0$;
20:            for ($i = 1 \to nInhNeurons$) $accG_i[i] \leftarrow accG_i[i] + g_i[k][j]$
21:        end for
22:    end for
23: end procedure
```

Algorithm 2 Excitatory Neuron

```
1: procedure ExcitatoryNeuron($accG_e, accG_i, isInhFired, nSpikes, newImageFlag$)
2:     if $newImageFlag$ then
3:         for $i = 1 \rightarrow nExcNeurons$ do
4:             $v_m[i] \leftarrow v_{rest}$
5:             $timer[i] \leftarrow 0$
6:         end for
7:     end if
8:     for $i = 1 \rightarrow nExcNeurons$ do
9:         if $(v_m[i] > threshold[i]) \wedge (timer[i] > refrPeriod)$ then
10:            $v_m[i] \leftarrow v_{reset}$
11:            $isInhFired[i] \leftarrow true$
12:            $timer[i] \leftarrow 0$
13:            $nSpikes[i] \leftarrow nSpikes[i] + 1$
14:        else
15:            $isInhFired[i] \leftarrow false$
16:            $timer[i] \leftarrow timer[i] + \triangle t$
17:            $drive[i] \leftarrow (accG_i[i].(v_{exc} - v_m[i]) - accG_e[i].v_m[i]).R$
18:            $\triangle v_m[i] \leftarrow (((v_{rest} - v_m[i]) + drive[i])/\tau_m).\triangle t$
19:            $v_m[i] \leftarrow v_m[i] + \triangle v_m[i]$
20:        end if
21:    end for
22: end procedure
```

Algorithm 3 Is Input Neuron Fired

```
1: procedure IsInpFired($pixel, scalingFactor$)
2:     $randNum \leftarrow rand()$
3:     $prob \leftarrow pixel.scalingFactor.\triangle t$
4:     if $prob > randNum$ then
5:         return $true$
6:     else
7:         return $false$
8:     end if
9: end procedure
```

7.2 Implementation

A network of 50 neurons is implemented. We followed Vivado Design Flow to implement hardware design. First an accelerator IP is created in vivado HLS with different optimizations . Then this IP is integrated with Zynq PS and other peripherals in Vivado IP integrator. Finally the design is mapped to ZedBoard and evaluated timing using SDK. The block diagram of the architecture is shown in Figure. 7.1. Important components of the design are the accelerator peripheral, Zynq PS, two block RAMS and an AXI timer. The descriptions of each of the following blocks are given below.

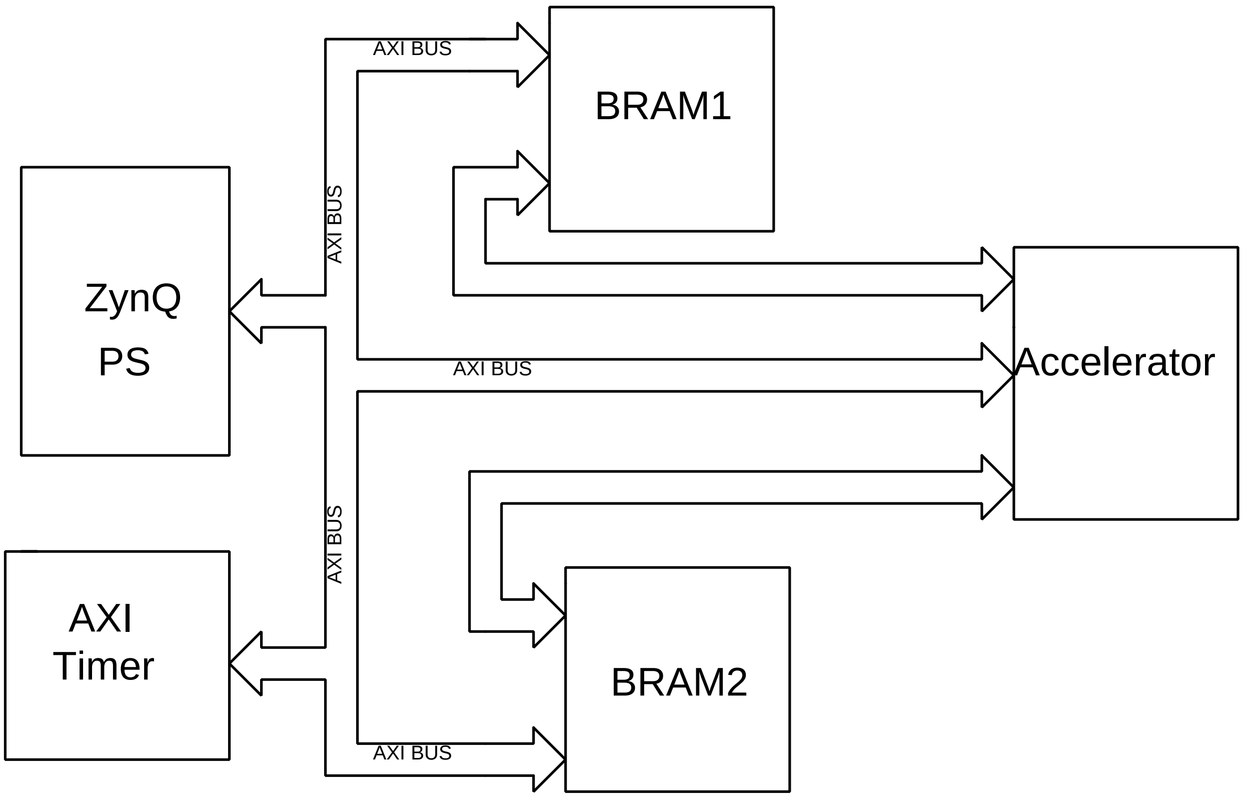

Figure 7.1: Block Diagram of the design

7.2.1 Accelerator

Accelerator is connected directly to two block RAMs and to Zynq PS through AXI Lite interface. The accelerator reads in pixel data of each image from BRAM1 and the calculated number of spikes generated by each neuron are writtern to BRAM2.

7.2.2 BRAMs

BRAM1 gets pixel array from Zynq PS and sends this data to the accelerator. Similarly, accelerator writes the number of spike generated by each neuron to BRAM2 and Zynq PS will read this data.

7.2.3 AXI Timer

This peripheral is used to get the time difference between the hardware and software execution.

7.2.4 Zynq Programmable System

Zynq PS which hosts ARM processor is conncted to the two block RAMS via BRAM interface.

7.2.5 Speed Optimized Hardware

The utilization reports of the area-optimized network are tabulated in 7.1. The design is running at $9.875\ ns$ and with an average latency of 14586737 clock cycles.

Type of Resource	**Numbers (%)**
LUTs	53200
RAMB36	96
RAMB18	50
DSPs	174

Table 7.1: Utilization of speed-optimized Network

7.2.6 Accuracy in Hardware and Time taken

The accuracy of this 50 neuron network implemented in hardware for 2000 test images is 74%. And the time taken for processing all 2000 images in $19.44\ seconds$.

7.3 Extended Version

The basic version of the algorithm can implement only fewer number of neurons. So, we introduced a time-multiplexed algorithm which can implement any number of neurons. And we successfully implemented a 400 neuron network using this algorithm.

Algorithm 4 State Update

```
1: procedure StateUpdate($neurNum, weightArr, geArr, giArr, pixelArr, scalingFactor, *nSpikes$)
2:     $accGe \leftarrow 0$
3:     $accGi \leftarrow 0$
4:     for $i = 1 \rightarrow nInputs$ do
5:         $prob \leftarrow pixelArr[i].\frac{scalingFactor}{8}.\triangle t$
6:         if $isInputFired(prob)$ then
7:             $geArr[i] \leftarrow geArr[i] + weightArr[i]$
8:         else
9:             $geArr[i] \leftarrow geArr[i] - geArr[i] * \frac{\triangle t}{\tau_e}$
10:        end if
11:        $accGe \leftarrow accGe + geArr[i]$
12:    end for
13:    for $i = 1 \rightarrow nExcNeurons$ do
14:        if $i \neq neurNum$ then
15:            if $isInhFired[i]$ then
16:                $giArr[i] \leftarrow geArr[i] + inhWeight$
17:            else
18:                $giArr[i] \leftarrow giArr[i] - giArr[i] * \frac{\triangle t}{\tau_i}$
19:            end if
20:            $accGi = accGi + giArr[i]$
21:        end if
22:    end for
23:    if $(v_m[neurNum] > threshold[neurNum]) \wedge (timer[neurNum] > t_{refr})$ then
24:        $v_m[neurNum] \leftarrow v_{reset}$
25:        $isInhiFired[neurNum] \leftarrow 1$
26:        $timer[neurNum] \leftarrow 0$
27:        $*nSpikes \leftarrow *nSpikes + 1$
28:    else
29:        $v_m[neurNum] \leftarrow v_m[neurNum] + ((v_{rest} - v_m[neurNum]) - accGe.v_m[neurNum] + accGi.(E_{inh} -$
   $v_m[neurNum])).\frac{\triangle t}{\tau_m}$
30:        $timer[neurNum] \leftarrow 0$
31:        $timer[neurNum] \leftarrow timer[neurNum] + \triangle t$
32:    end if
33: end procedure
```

Algorithm 5 Process Single Image

```
1: procedure ProcessSingleImage($pixelArr, scalingFactor, nSpikes, weightArr, geArr, giArr$)
2:     for $i = 1 \rightarrow nBins$ do
3:         for $j = 1 \rightarrow nExcNeurons$ do
4:             $StateUpdate(j, weightArr[j], geArr[j], giArr[j], \&nSpikes[j], pixelArr, scalingFactor)$
5:         end for
6:     end for
7: end procedure
```

CHAPTER 8

ANALYSIS OF WEIGHTS

8.1 Distribution of Trained Weights

Figure 8.1 shows the histogram of weights terminating on three neurons belonging to different classes. If we consider a particular neuron, it has only few synapses with

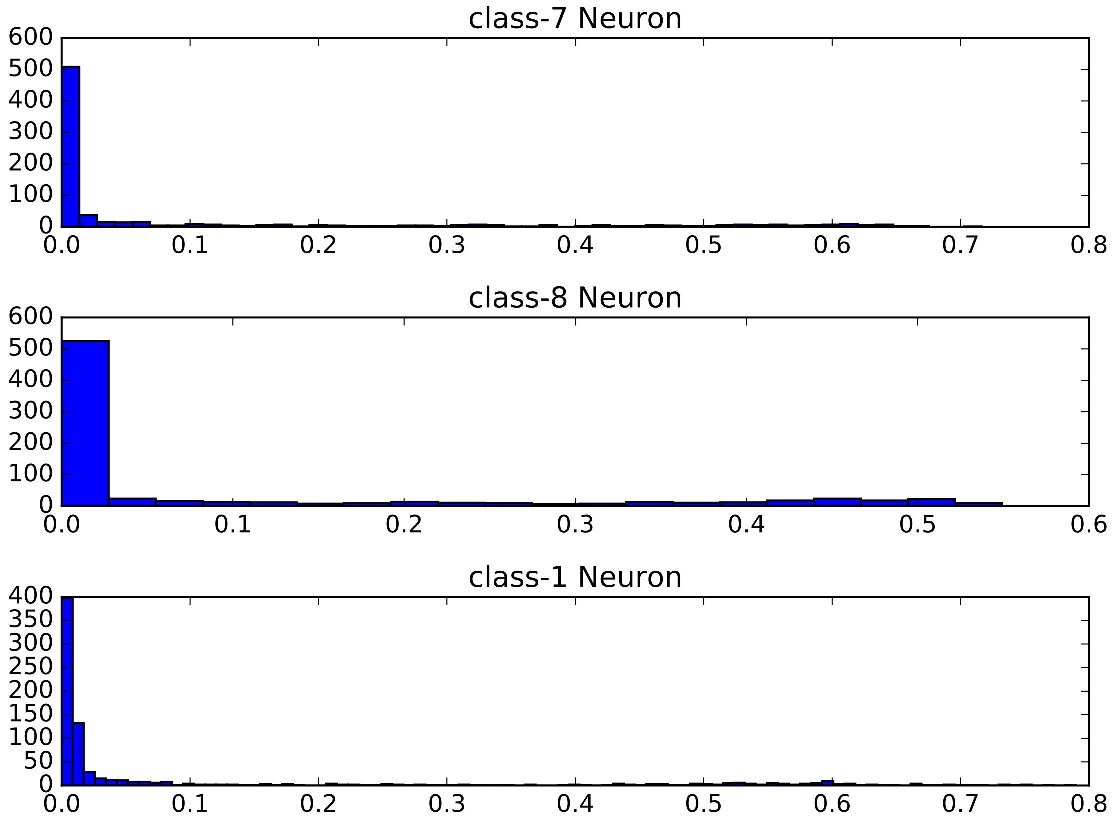

Figure 8.1: Histogram of weights to three neurons belonging to different classes (℧)

considerable weight. By considering this fact into consideration, we can save lot of memory needed to store weights. But different neurons have a different set of considerable weight synapses, posing a challenge to build the network. This challenge can be taken up in future work.

Also figure 8.2 shows the histogram of all weights in a 400 neuron network. It is seen that most of the synapses have negligible weight associated with them.

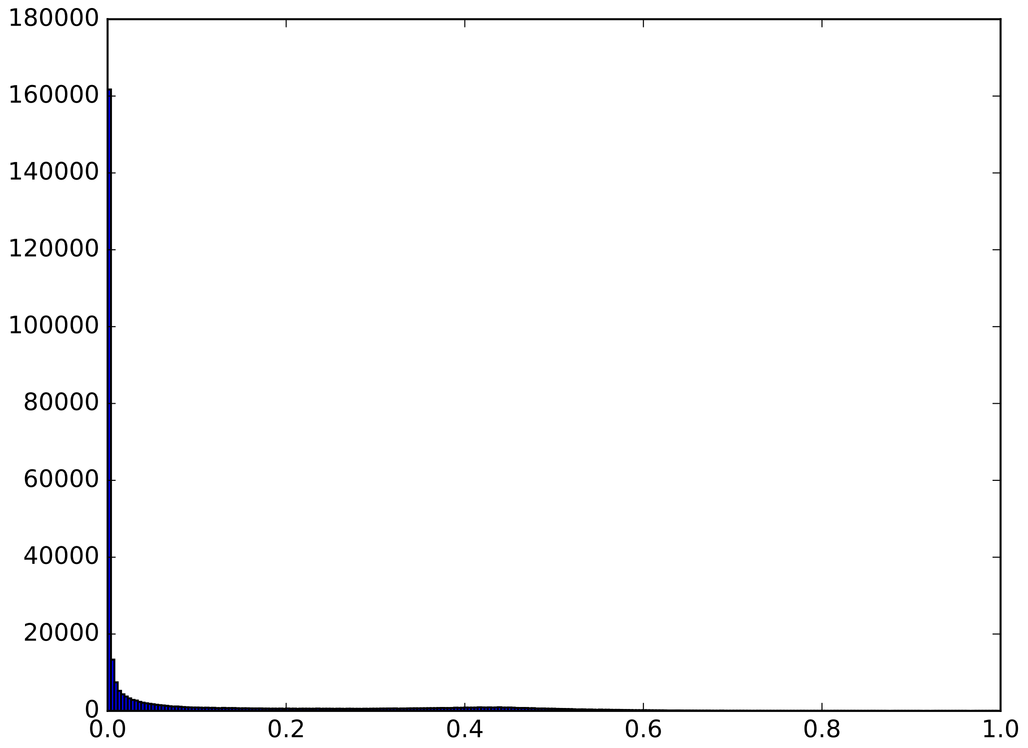

Figure 8.2: Histogram of all weights in the network (℧)

8.2 Imposing threshold on weights

Because of the huge number of negligible weights, we can put a threshold on weights such that all weights below the selected threshold are negligible. We have evaluated the performance accuracy of the network for a network of 100 neurons with 2000 test images while varying the threshold imposed on weights, and the results are tabulated in table 8.1.

Also the variation of performance accuracy over threshold imposed on weights is plotted in figure 8.3. As is obvious from the plot, we can increase the threshold upto 0.25 ℧ without compromising on accuracy, thereby neglecting approximately 80% of the weights.

Threshold on weight(℧)	**% No. of weights removed**	**Accuracy (in %)**
0.00	NA	78.95
0.05	66.30	78.45
0.10	71.97	79.40
0.15	76.24	80.05
0.20	82.41	79.45
0.25	79.57	79.00
0.30	85.39	77.40
0.35	88.45	72.95
0.40	91.75	67.00
0.45	94.37	54.40
0.50	96.24	48.55

Table 8.1: Effect of threshold on weights

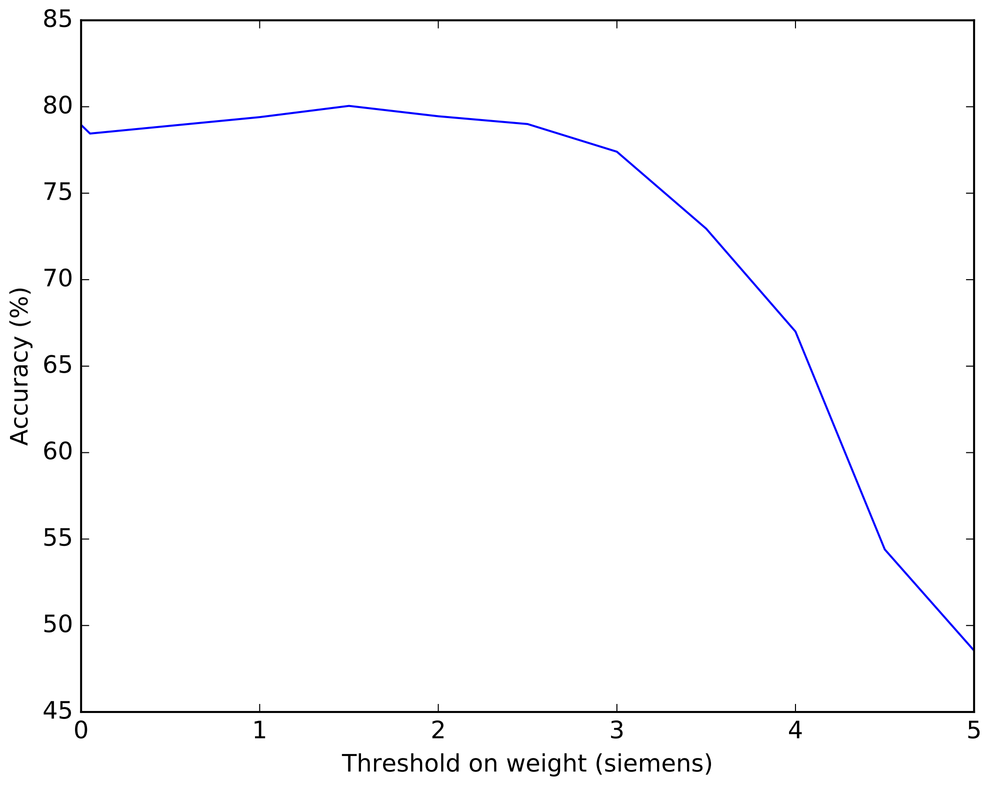

Figure 8.3: Accuracy vs Threshold imposed on weights

8.3 Quantization and Threshold on weights

For a network of 100 neurons, we tabulated the performance accuracy of the network while varying the bit width of weights in table 8.2, and the number of test images used are 2000.

Bit Width	**Accuracy (in %)**
2	64.6
3	38.4
4	32.75
6	59.95
8	74.7
9	78.3
12	78.7

Table 8.2: Effect of quantization

Now, we applied a threshold of $0.25\ \mho$ on weights along with quantization and the results are tabulated in table 8.3

Bit Width	**Accuracy (in %)**
2	64.6
4	70.75
6	76.3
8	78.15
9	78.45
12	78.75

Table 8.3: Effect of quantization and threshold on weights

The results are plotted in figure 8.4.

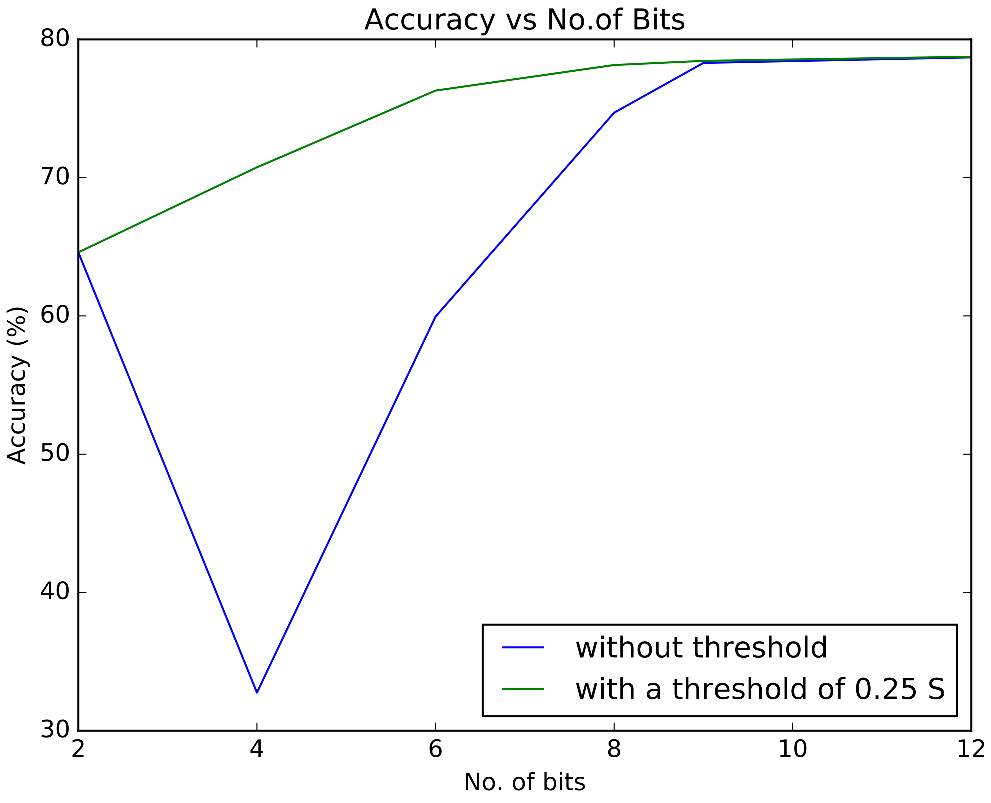

Figure 8.4: Effect of quantization and threshold on weights

APPENDIX A

eFPGA ARCHITECTURE EVALUATION FOR SOC DESIGNS

Quite often specifications are not well defined when a design project starts. Several customers/markets require multiple variants of an Application Specific Integrated Circuit (ASIC). Also some of the standards such as network standards keep on changing. Same ASIC may evolve over time for a specific portion of the logic such as FSM ogic, filters etc. So, portions of ASIC which need programmablity can be replaced by embedded FPGAs.

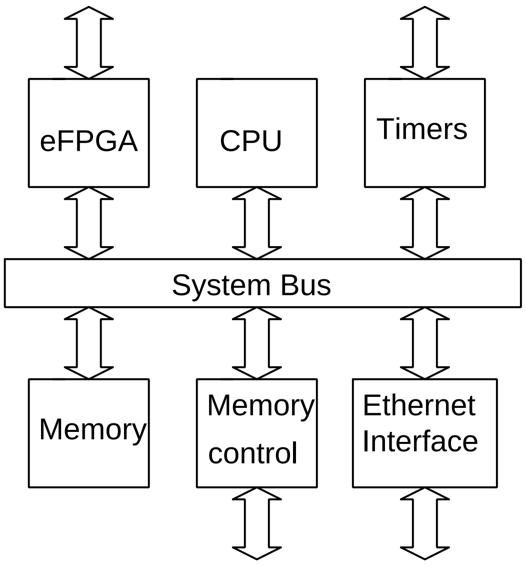

Figure A.1: eFPGA in SOC

A simple eFPGA fabric embedded in an SoC design is shown in figure A.1 This hardware programmability comes at the cost of increase in chip area. If the area overhead is reasonable, then we can use embedded FPGAs (eFPGAs).

A.1 eFPGA Design Flow

Select blocks hich can benifit from programmable fabric. The embedded programmable core should have enough headroom to accomodate variations in the design.The through-put requirements are as in hard-wired ASIC design. So, if the timing is slower in FPGA, then the additional area overhead should be considered in order to accomodate additional datapath to make programmable fabric as fast as ASIC.

A.2 EFLX embedded FPGA cores

EFLX, an eFPGA fabric from Flexlogix technologies (7), provides an array with any size, thousands of $I/O's$, optional DSP acceleration. optional RAM of any size/kind, multiple clocks abd software that maps our RTL into EFLX array.

A.3 Purpose and Scope of this work

The purpose and scope of this study is to map both control intensive and DSP intensive designs to EFLX (from Flexlogic) arrays and Xilinx architectures, and to evaluate the performance of these implemented designs compared to ASIC in term of speed and area.

A.4 Methodology

RTL designs are mapped to ASIC using Cadence RC synthesis tool with UMC-28 HPC node libraries. For FPGA designs, the netlist in .edif format is generated using Synplify Pro. This netlist is implemented into both EFLX arrays and Xilinx's Virtex architectures.

$$estimated\ ASIC\ area = \frac{Cellarea}{0.6}\ m^2$$

$$estimated\ EFLX\ area = \frac{3}{4}.targetfrequency.\frac{No.LUTs}{2700}\ m^2$$

The analysis is done for 2 DSP intensive designs and 2 control intensive designs as shown in table below.

A.5 Conclusion

The area overhead for control intensive designs is approximately 200 times as that of ASIC. And in DSP intensive designs, the area overhead is approximately 100 times as that of ASIC.

Table A.1

Design	speed(MHz)	Device	Clock	Feasible(MHz)	RBB	LUT	FF	DSP	BRAM	EFLX area in sq.m	ASIC area in sq.m	Overhead
DSP-1	200	Eflex-V4	clk-1	63.9	3718/4560	12936	7963	107/120	0	11.2467	0.102338	110
			clk-2	245.7								
		Xilinx-V4	clk-1	77.68	NA	12523	7963	107	0	NA		
			clk-2	229.15								
	20	Eflex-V4	clk-1	68.66	3846/4560	13337	7582	108/120	0	10.7915		105.5
			clk-2	144.07								
		Xilinx-V4	clk-1	82.36, 44.44	NA	12921	7582	108	0	NA		
			clk-2	255.16, 269.54								
DSP-2	104	Eflex-V4	clk-1	59.3	1759/2360	6268	4489	6(40)	0	3.0535	0.036556	83.53
			clk-2	1075.26								
		Xilinx-V4	clk-1	71.85	NA	6223	4489	6	0	NA		
			clk-2	1038.42								
	10.4	Eflex-V4	clk-1	53.63	1760/2360	6147	4479	6(40)	0	3.3112		90.57
			clk-2	1075.26								
		Xilinx-V4	clk-1	46.84	NA	6072	4479	6	0	NA		
			clk-2	77.166								
	104	Eflex-V7	clk-1	57.3	1881/2360	9930	4624	6(40)	0	5.006		136.95
			clk-2	1075.26								
		Xilinx-V7	clk-1	105.38	NA	4818	4479	6	0	NA		
			clk-2	1787.67								
	10.4	Eflex-V7	clk-1	58.034	1987/2360	9941	4479	6(40)	0	4.95		135.4
			clk-2	867.3								
		Xilinx-V7	clk-1	107.06	NA	3872	4535	6	0	NA		
			clk-2	1751.8								
CONT-1	204	Eflex-V4	clk	56.15	1305/2520	4805	2292	0	0	4.868	0.019392	251
		Xilinx-V4	clk	58.04	NA	4780	2292	0	0	NA		
	20.4	Eflex-V4	clk	50.29	1171/2520	4492	2239	0	0	5.081		262.04
		Xilinx-V4	clk	61.79	NA	4405	2239	0	0	NA		
	204	Eflex-V7	clk	51.82	1397/2520	8669	2357	0	0	9.63		506.8
		Xilinx-V7	clk	160.544	NA	3901	2350	0	0	NA		
CONT-2	204	Eflex-V4	clk	48.9	7332/15750	27549	15342	0	0	30.61	0.110363	278.27
		Xilinx-V4	clk	53.93	NA	27527	15342	0	0	NA		
	20.4	Eflex-V4	clk	43.1	7157/15750	26869	15309	0	0	34.71		315.54
		Xilinx-V4	clk	39.52	NA	26692	15309	0	0	NA		
	204	eFlex-V7	clk	39.6	9290/15750	54239	16838	0	0	77.91		708.27
		Xilinx-V7	clk	114.8, 132.01	NA	19808	15357	0	0	NA		

REFERENCES

[1] P.U.Diehl, D.Neil, J.Binas, M.Cook, S.C.Liu, M.Pfeiffer. *Fast-classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing*. IJCNN (Anchirag, AK), 1-8.doi:!0.1109/ijcnn.2015.7280696.

[2] P.U.Diehl, M. Cook. *Unsupervised Learning of Digit Recognition Using Spike-Timing-Dependent-Plasticity*. Front. Compt. Neurosci. 9:99. doi:10.3389/fncom.2015.00099.

[3] The Brian Neural Network Simulator, `http://briansimulator.org/`.

[4] L.F. Abbott. *Theoritical Neuroscience Computational and Mathematical Modelling of Neural Systems*. MIT Press.

[5] Yann LeCun, Corianna Cortes, Christopher J.C.Burges: The MNIST database of handwritten digits, `yann.lecun.com/exdb/mnist`

[6] IniLabs: Dynamic Vision Sensors, `http://inilabs.com`

[7] `http://www.flex-logix.com/`