DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2016

Design and Implementation of Digital Spiking Neurons for Ultra-low-Power In-cluster processors

SHARAN KUMAAR GANESAN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

KTH Information and
Communication Technology

KTH Royal Institute of Technology

Dept. of Information, Communication and Technology

Master's thesis
on

Design and Implementation of Digital Spiking Neurons for Ultra-low-Power In-cluster processors

Sharan Kumaar Ganesan
skgan@kth.se
Master of Science in System-on-Chip Design
Supervisor: Dr.Francesco Conti, ETH, Zurich
Examiner: Prof. Ahmed Hemani, KTH, Sweden

Abstract

Neuromorphic computing is a recent and growing field of research. Its conceptual attractiveness is due to the potential it has in deep learning applications such as sensor networks, low-power computer vision, robotics and other fields. Inspired by the functioning of brain, different neural network models have been devised, each with their own special focus on certain applications. Using such computing models are already helping us in different cases such as image, character and voice recognition, data analysis, stock market prediction, etc.

Among the multitude of artificial neural models available, spiking neurons are more deeply inspired by biological neural networks. Leaky, Integrate and Fire (LIF) neuron model is one such model that can reproduce a good number of functions, be simple and also extensible in structure. Current deep learning applications are tied to servers and datacenters for their power and resource hungry existence. This work aims at building a low power neuron core taking advantage of LIF neuron, that could possible result in independent battery powered devices. A hardware design of LIF neuron based scalable neural core is explored, constructed and analysis for power consumption is made.

Acknowledgment

It has been a good learning experience for me, to study for my master's degree and carry out my thesis at Royal Institute of Technology (KTH).

First of all, I would like to express my special thanks of gratitude to my supervisor Dr.Francesco Conti (ETHZ), for guidance and technical support that he provided me at every step of my work. He has been so considerate and encouraging through out the period of my work. I'm also thankful to Prof.Ahmed Hemani (KTH) and Prof.Luca Benini(ETHZ) for accepting me to carry out this work and giving me the opportunity to work on a topic that I enjoyed. This has given a valuable experience in neuromorphic hardware design. I'm grateful to Integrated systems laboratory at ETH Zurich, for giving me access to their design tools, technology and support.

Next, I would like to thank my parents, to whom I'm grateful and indebted for all the support they been giving me for all these years. I also thank everyone who have been inspiring and pushing me to do more.

Finally, I would like to express my special thanks to all my friends, and those who made my studies in KTH nothing but a pleasure.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ANN	Artificial Neural Network
ASIC	Application Specific Integrated Circuit
BPSA	Bitwise-Parallel Serial Adder
BSPA	Bitwise-Serial Parallel Adder
CNN	Convolutional Neural Network
DSE	Design Space Exploration
DTCM	Data Tightly Coupled Memory
FPA	Fully Parallel Adder
FPGA	Field Programmable Gate Array
FSA	Fully Serial Adder
GPU	Graphics Processing Unit
HDL	Hardware Description Language
ITCM	Instruction Tightly Coupled Memory
LIF	Leaky, Integrate and Fire
PSA	Parallel Serial Adder
PSA-8B	Parallel Serial Adder with Block size 8
RBF	Radial Basis Function
SNN	Spiking Neural Network
SPA	Serial-Parallel Adder
VCD	Value Change Dump

Chapter 1

1 Introduction

Current interest in neuromorphic computer architectures is enormous, due to its conceptual attractiveness and its potential applications to sensor networks, robotics, computer vision and other fields. Inspired by the functioning of brain, effort has been made to mimic the operations of its network of neurons. Scientists derived mathematical relations for most behaviors of a neuron which has resulted in different types of Artificial Neural Networks (ANNs).

Every neural network is specialized in its own ways and is targeted at a particular application. For eg., Convolutional Neural Networks (CNNs) are helpful in image/character recognition tasks such as reading speed signs on a car or reading signatures in a bank. Such usefulness comes with its own cost. These networks are so resource intensive that it requires server class computers to model, train and implement them. Neural networks run in the form of deep learning algorithms on cloud servers, offloading training, computations and giving a limited usage to users.

Though today's supercomputers and servers can successfully run a deep learning algorithm, their efficiency is not even close to those of the biological brain. Even if we can do certain tasks well, in terms of power consumption we are still far behind the brain. With 10^{16} bits per second of computations, Blue Gene consumes about 10 MW of power, which the brain can do with just 10 W. Existing architectures of neural networks sacrifice power efficiency to match computational power. Also, this efficiency restricts their usage to supercomputers and servers.

Currently, its not viable to run such algorithms on independent battery powered devices. Having these powerful networks in stand-alone devices such as mobile phones or any hand-held can be very helpful. Imagine having the ability to diagnose medical scan images in a remote village without connectivity, which is a small example use case. The resource and power hungry nature of such networks are largely keeping us from tapping its full potential.

The Goal of this thesis work, is to design a neural core, that can run with very low power consumption. Having a low power neural network, would open new vistas of computing. Disconnecting us from cloud servers that run deep learning algorithms can lead to a new class of intelligent devices and gadgets. Drones and Robots can have intelligent computer vision. Consumer devices can organize data and understand the needs of the users, transforming from being an electronic device to being an electronic assistant. Medical imaging devices will reach new heights. This would open endless possibilities of intelligent devices, without hogging our already limited supply of power.

With low power neural networks, larger and complex networks can be formed, making our devices more intelligent. This would help us focus on more complex tasks at a higher level of abstraction. Such possibilities can be made possible with the advent of low power neural network implementations, which would be the central idea of this thesis.

The rest of this thesis report is organized as follows. In chapter 2, fundamentals of neural networks, and its challenges are discussed. A brief discussion about existing neuromorphic hardware projects is carried out as related works in chapter 3. An overview of the hardware blocks and architecture related to this work is discussed in the next chapter. Following this, design exploration for single neuron is carried out in chapter 5. On chapter 6, design and implementation of neuron core with its results are presented. Finally, the work is concluded in chapter 7 & 8, with observations and future work respectively

Chapter 2
Neural Networks

2.1 Overview

Computers can crunch endless stream of numbers and give accurate results; can handle complex control tasks; store enormous amounts of data and access it at great speeds. However, there are classes of tasks which a computer can only process with large amount of effort, resource and power, which a human brain can do effortlessly. Common examples of such tasks are character recognition, image interpretation or reading texts.

Brains can adapt to new scenarios and enhance their knowledge by learning. They are capable of dealing with incomplete information and still produce the desired results. They don't require predefined algorithms since it can learn. No task specific initiation about problems are required, since they learn as one grows. It adapts by making new connections. They can just learn from representative examples, understand it and adapt itself to do similar tasks.

The neural way of processing is advantageous for the above addressed class of problems. This class of computing is called as *neural network* or *artificial neural network*. These are models of nervous system and its dense network of interconnected processing elements. These processing elements are called *neurons* and its consequences are what we call as learning.

Works on neural networks are not aimed at recreating a brain as such, because we have already understood that we don't have the technology to do so today. However, neural networks aim at modeling the basic elements, functions and isolated tasks of brain. Such neural networks can do tasks which are difficult to characterize in an analytic way, but has to be trained beforehand.

2.2 Biological Inspiration

The smallest entity in the brain, that can be called as a processing element are *nerve cells* or *neurons*. A neuron is an electrically excitable cell, that processes and transmits information through electrical and chemical signals [19]. Neurons can connect with other neurons to form a neural network (which comprises the Central Nervous System), which occur via, synapses. There are about 100 billion neurons in human brain with an average of $10^{14} - 10^{16}$ synapses in an adult's brain.

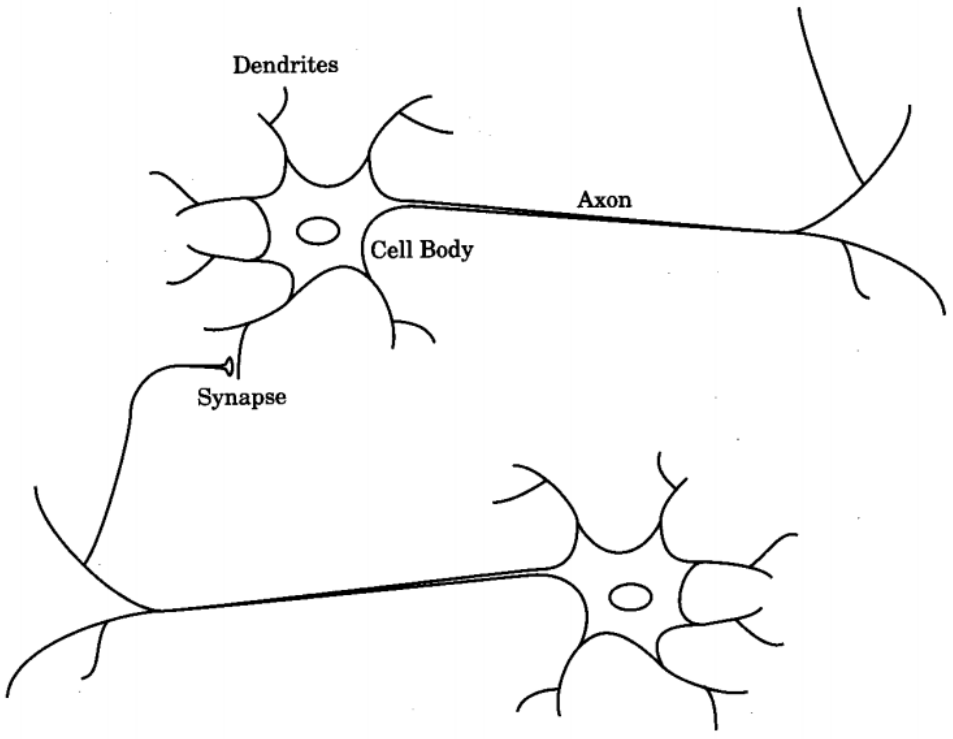


Figure 2.1: Basic structure of neuron [18]

Table 2.1: Structure and function of neuron

Structure	Function
Dendrites	Carry signals from other cells into the cell body
Cell Body(Soma)	Sum and threshold the incoming signals
Axon	Signal transfer to other cells
Synapse	Contact point between axon and dendrites

The basic structure of a neuron is shown in figure 2.1 and functions of different parts of the neuron are described in the table 2.1. Each and every biological functions are carried out by these structures in the neurons by making connections with

Table 2.2: List of Neuron models

Year	Model Name
1907	Integrate and Fire
1943	McCulloch and Pitts
1952	Hodgkin-Huxley
1958	Perceptron
1961	Fitzhugh-Nagumo
1965	Leaky integrate and fire
1981	Morris-Lecar
1986	Quadratic integrate and fire
1989	Hindmarsh-Rose
1998	Time-varying integrate and fire model
1999	Wilson Polynomial
2000	Integrate and fire or burst
2001	Resonate and fire
2003	Izhikevich
2003	Exponential integrate and fire
2004	Generalized integrate and fire
2005	Adaptive exponential integrate and fire
2009	Mihalas-Neibur
2013	TrueNorth

others.

Every neuron receives electrical impulses through dendrites from multiple other neurons. This is followed by responses in the form of electrical impulses sent out to other bunch of connected neurons or muscles through the axons. Each neurons could have thousands of such connections.

The synapses are structures that permit a neuron to pass electrochemical signals to other neurons. These are of two types, namely - excitatory and inhibitory synapses. A neurons activities are understood through its electric potential. Based on the incoming signals and the state of the cell body, the potential will be increased (excitatory synapse) or decreased (inhibitory synapse). The output impulse from a neuron occurs when the sum of input impulses exceed the threshold of the neuron. This impulse will then be transmitted to other neurons as described above.

The neuron activities and functions have been studied by different researchers and characterized over the past century and different models have been proposed 2.2. These models range from *phenomenological*, where the behavior of neuron is captured through mathematical abstract equations, to *biophysical*, where the electrophysiology of the neuron membrane has been modeled. The functioning of biological neurons are still not completely understood and is still an active research area. Over the course of time, different implementations of these models have been made. Basic details about these Artificial neurons are discussed in the next section.

2.3 Neural Network Fundamentals

2.3.1 Artificial Neuron

Artificial neurons are mathematical functions of biological neurons. Aritificial neurons (henceforth will be called just as neurons), like their biological counterparts, have an arbitrary number of inputs and an output. The basic structure of the neuron is portrayed in figure 2.2

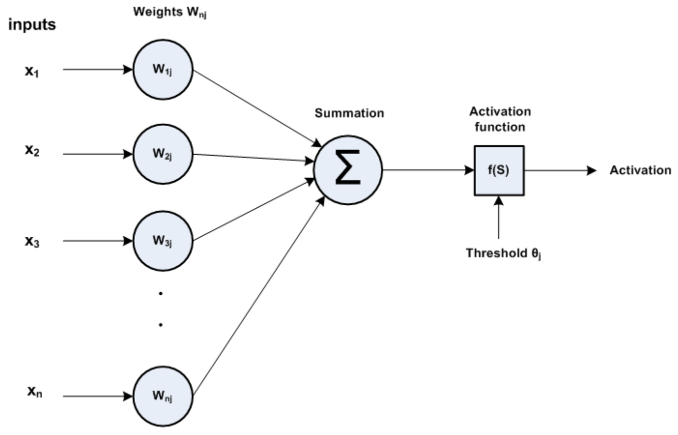


Figure 2.2: Artificial Neuron structure

All inputs x_i are multiplied with weights w_{ij}. The values are then summed up together (with a bias, if required). The result is propagated via an activation function to get an output.

There are three common transfer functions as discussed below.

- **Linear** - This is the simplest type. It is just the weighted sum of inputs plus bias. More useful in the first layer of a network.
- **Step** - The output of this function is either '1' or '0', depending on the threshold. It is also called a threshold function.
- **Sigmoid** - This is a non-linear function, which can be continuous, differential or monotone and is easily derivative. It is commonly found in multilayer perceptrons using a backpropagation algorithms.

2.3.2 Neural Network Architecture

A neural network's ability to solve problems comes from its architecture. It is the connections between individual neurons that produces results than the individual neurons. These networks have important topologies which are discussed below.

- Feed-forward networks
- Feedback networks
- Network layers

Feed-forward networks
In a feed-forward network, signals travel in one way only - from inputs to outputs. There are no feedback loops. They are extensively used in pattern recognition. They are organized as bottom-up or top-down.

Feedback networks
As the name suggests, these networks have a feedback from the higher layer to a lower one or inside the same layers. These are very powerful and can get extremely complicated. These are also dynamic networks and should be operated towards an equilibrium point.

Network layers
Any common type of neural network would contain three groups or layers of neurons: a layer of "input" neurons are connected with a layer of "hidden" layer(s), which are then connected to an "output" layer of neurons.

The hidden layer usually has one or more layers of neurons inside it, which represents the complexity of the constructed network.

2.3.3 Learning

A key element of any neural network is its learning. Looking at an analogy here, would be helpful in understanding it easier. A neural network learns similar to how we learn in our lives from experiences - we perform an action, observe the outcome and remember it. Neural networks learn the same way.

Every neural network possesses some knowledge in the form of connection weights. Modifying the value of weights in these networks is a function of learning, with which the network learns something. With the way learning is performed, neural networks can be classified into two types:

- **Fixed networks** where the weights cannot be changed. In these networks, the weights are fixed in prior based on the problem it is intended to solve.
- **Adaptive networks** are those in which weights can be changed. These networks can learn as they work.

From the point of view of their learning methods, neural networks can be classified into three categories.

- **Supervised Learning** - This method involves an external teacher, so that each output unit is taught how it has to react to input signals. For example, let's consider facial recognition. The teacher would show the network a bunch of faces with names. The network would learn and start guessing on its own and make mistakes, when the teacher would then give correct answers to correct its errors. Error-correction learning, reinforcement learning and stochastic learning are some paradigms of supervised learning.
- **Unsupervised Learning** - This method doesn't use an external teacher and learn based on its local information. This is employed when there isn't an example data set with answers. For example, imagine searching for patterns in a new data set or diving set of elements into groups based on unknown similarities. Hebbian learning and competitive learning are paradigms of unsupervised learning.
- **Reinforcement learning** - This method is totally based on observation and is employed significantly in robotics. For example, imagine a robot mouse going through a maze. If it turns left, it has cheese and if turns right, it has a cat. By taking each of these options the robot could adjust its weights and learn through observation. Markov decision process is one of the paradigms in this method of learning.

2.3.4 Applications

The ability of neural networks to observe, learn and restructure itself has made it very useful in the field of deep learning and artificial intelligence. Some standard use cases for neural networks are discussed below.

- **Pattern Recognition** - This is the most common application. Examples are facial recognition, optical character recognition, etc.
- **Time Series Prediction** - They can be used to make predictions like - will the stocks rise or fall tomorrow? Will it rain or be sunny this week?
- **Signal Processing** - Hearing implants and aids need to filter unnecessary noise and amplify important sounds. Neural networks can help here in processing audio signals and filter it appropriately.
- **Control** - Neural networks can be used to give movement to robots. They play a major role in the advancement of self-driving cars.
- **Sensors Fusion** - A thermometer can tell us the temperature of air. If we also knew the humidity, pressure, dew point, and air quality, etc., it would be

much more useful to make climate controlling decisions. Neural networks can help with such fusion.

- **Anomaly Detection** - Since neural networks are good in detecting patterns, they can be used to find unusual behavior. It can give health alerts, detect intrusions, or even be your personal assistant.

2.4 Neural Network Types

Artificial Neural Networks can be classified into different types based on their structure, application area or learning method. They bear limited resemblences to their biological counterparts, but are very effective at their intended tasks. This is mainly because of the topologies they use. Thus, it is safe to classify them based on their topologies or algorithms.

The major types of Neural networks are as listed below.

- Feedforward neural network
- Radial basis function (RBF) network
- Kohonen self-organizing network
- Learning vector quantization
- Recurrent neural network
- Modular neural networks
- Physical neural network
- Other types

Besides the major types mentioned here, there are further classifications inside each of these, which is shown in figure 2.3 below.

It has to be noted that most types of neural networks are software simulations and not hardware implementations. Not all networks are suitable for hardware. Out of these networks, spiking neural networks are considered to be suitable for hardware implementation. Neural networks in hardware are discussed in detail in the next section.

2.5 Neural Networks in Hardware

Majority of neural networks today are software implementations that run on work-stations and servers with a high-end hardware specifications. For an end-user to take advantage of it, the computations are usually carried out in cloud servers and only the

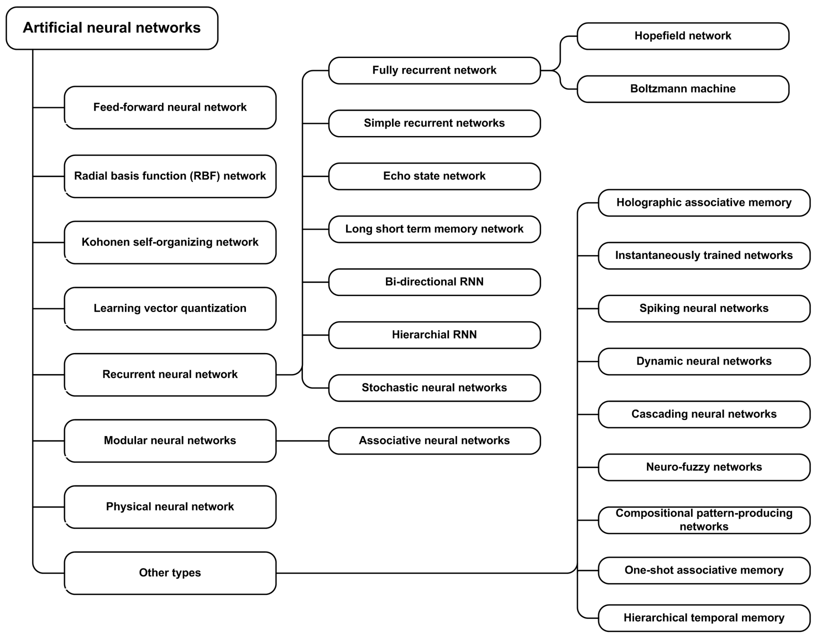

Artificial neural networks
Feed-forward neural network
Radial basis function (RBF) network
Kohonen self-organizing network
Learning vector quantization
Recurrent neural network
Modular neural networks
Physical neural network
Other types
Fully recurrent network
Simple recurrent networks
Echo state network
Long short term memory network
Bi-directional RNN
Hierarchial RNN
Stochastic neural networks
Associative neural networks
Hopefield network
Boltzmann machine
Holographic associative memory
Instantaneously trained networks
Spiking neural networks
Dynamic neural networks
Cascading neural networks
Neuro-fuzzy networks
Compositional pattern-producing networks
One-shot associative memory
Hierarchical temporal memory

Figure 2.3: Classification of ANN

results are given to the user. This approach though is effective initially, its definitely a bottleneck to tap its full potential. Besides, software implementations also have their limitations in speed due to operating systems and other drivers connecting them to hardware. So, it becomes important to be able to implement such networks on hardware, where we would get the following benefits:

- Higher operating speed by exploiting parallelism
- Reduced cost of systems in high volume applications
- Could result in standalone intelligent devices
- Optimization for particular operating conditions are possible e.g. smaller size, low power consumption and ability to run in hostile environments, etc

These advantages and the growing list of applications for such networks are good enough reasons to turn them into hardware. This is taken as the motivation for this work, where neural networks have been chosen to implement in hardware in either ASIC form or on FPGA platform. Digital spiking neurons are selected for this and details about it are discussed in section 2.7.

2.6 Challenges

Having superficially discussed the advantages of implementing neural networks in hardware, it is important to look at challenges faced in implementing them. Talking about bio-inspired architectures, there are numerous differences between our brains and computers. It is these differences that are our challenges.

If we think about the computer analogy of brain, brain would be hardware, mind will be software, computations will be algorithms and memory would be stored data. Having classified into these four area, we have problems in all of these four areas. In terms of hardware, though we can achieve high speeds of computation and massive parallelism, we are severely stuck when in comes to low power and high fan-out capacity. In fact, speed is not a need at all since a brain runs only at Hertz of frequencies. Speed is just a minor convenience for hardware. Next is the software, which cannot match our mind, atleast for now. It is totally something else which cannot be written in the form of algorithms. Most computations in brain are happening in analogue and not using software. Storage in brain is highly dense and doesn't consume much energy, whereas a computer memory units are modular, occupy large area and power.

Besides these differences, it also has to be noted that brain is fault-tolerant during normal conditions, and is vulnerable only during accidents and aging. Differences such as these have been discussed in details in the paper [2].

With so many challenges to conquer, we try to contribute to the part of reducing power consumption and area by working on simple and optimized implementation of a neuron core in this thesis.

2.7 Spiking Neural Networks

Spiking neural networks (SNN) are a set of neurons that communicate through spikes and compute through the timing of the spike. These spiking neurons have become popular since they mimic the spiking nature of biological neurons and can reproduce those neuron spiking patterns. There are cases where SNNs are more biologically plausible and more powerful than non-spiking ones [20] [21].

The Hodgkin-Huxley model [3] is one of the most detailed and best known models of spiking neurons. It clearly describes the subcellular level behaviors, the membrane current generation and propagation of neural spikes. And because of these high level of details, the Hodgkin-Huxley model is too complex to be used for a large scale simulation or hardware implementation. The Izhikevich model [4] is relatively recent model that is simple, has good performance both on computational efficiency and functional richness.

For large scale hardware implementations, simplified models such as integrate-and-fire model are preferred. This is due to limited hardware resources for any design and same is the case for this thesis. Such simpler models can emulate the spiking nature of neurons, most of its behaviors and also keep the cost of computation at a comparatively low level [22].

In this thesis, the neuron model used is the leaky, integrate and fire model [5] which has a lesser level of complexity than Hodgkin-Huxley and Izhikevich, which we think would lead us to a more low power implementation. Details about Leaky, Integrate and Fire models are disccused in forthcoming sections.

Chapter 3

3 Related Work

There are multiple existing works with the idea of implementing deep learning neural networks on hardware. Each of those implementations have been having their own focus areas using different platforms. Major players as platforms for hardware implementations are GPUs, FPGAs and ASICs. Implementations on GPU don't exactly fall in the hardware category, but it is a major contender for deep learning implementation with speed and efficiency. FPGAs are the next platform of choice for their parallelism and power efficiency compared to GPUs. Finally, ASICs are the most optimized and custom platform, which exhibits the advantages from FPGAs, but is too expensive and takes longer time to get your hands on the designed hardware. With what is known so far, they could be the best form of existence for a neural network.

Different research teams from different parts of the world have been working on these platforms over a decade. We will consider a few of those works in this chapter, to relate them to our work and serve as inspiration.

3.1 SpiNNaker

Spiking Neural Network Architecture (SpiNNaker) [6] is a computing engine with million cores, aimed at simulating behavior of upto a billion neurons in real time. Their goal was to simulate brain-scale sized neural networks for biological research. It employs an array of ARM9 cores, using packet communication through a custom interconnect fabric. With upto 1,036,800 ARM9 core, it uses 64Kbytes of data tightly coupled memory (DTCM) and 32Kbytes of instruction tightly coupled memory (ITCM) for each of its cores. The packets are of size 40 bits or 72-bits each, which are transmitted using a custom concurrent routing system. It is said that, during operation, the computing engine consumes upto 90kW of electrical power. The architecture of this system is discussed in detail in [8] and the associated software in [7].

3.2 NeuroGrid

NeuroGrid [9] is another neuromorphic system with a goal of simulating large models in real time. It can simulate a million neurons with billions of synaptic connections using 16 neuron cores embedded on a board that consumes 3W. Deviating from a fully digital implementation, they realized all electronic circuits except those of axonal arbors in analog. It also uses shared circuits for axons, synapses and dendrites, to reduce transistor count. They also have custom made routing interconnections and software to provide user access and reconfigurability.

3.3 Minitaur

Minitaur [10] is an event-driven, low power and high performance neural network accelerator implemented on a Spartan 6 FPGA. It is said that, this board can be integrated to existing systems and offload computationally expensive neural network tasks from a processor. The version discussed here implements spiking networks run with 1.5W of power and gives upto 65K neurons per board. It uses fixed point weights and parameters to configure and run the network.

3.4 ROLLS neuromorphic processor

Reconfigurable On-Line Learning Spiking Neuromorphic processor (ROLLS neuromorphic processor) [11] is a full custom mixed-signal implementation. It has learning circuits that can emulate the functioning of biological spiking neurons and synapses for exploring neuroscience models and building brain-inspired computing systems. Their existing design can run a wide range of activities such as recurrent and deep networks, with short-term and long-term plasticity. The device has 128K analog synapses and 256 neurons that has on-line learning capabilities. A prototype fabricated using 180nm process consumes approximately 4mW and takes an area of $51.4mm^2$.

3.5 TrueNorth

TrueNorth [12] is a neuromorphic chip produced by IBM. It is a multicore network of chips. It has a total of 4096 cores in its current iteration and each core has 256 programmable neurons. Each of these neurons has 256 programmable synapses, contributing to millions of synapses in the entire chip. It is an event driven chip, that circumevents Von-neumon architecture and is very energy efficient. It is reported that it consumes 45mW of power for a million synapses, which is a very small fraction of power consumed by conventional microprocessors.

TrueNorth uses an interconnected network of neurosynaptic cores. It implements "gray matter" with an intra-core crossbar memory and "white matter" using long-range connections using a spike based messaging network. The core has 256 axons and neurons each forming a 256x256 crossbar. Synapses hold information which result in spiking of a neuron, which is later fans out to all neurons through axons. The neurons are an enhancement from integrate and fire neurons with many more parameters. An overview of the core and its connections are depicted in figure 3.1 from [12]

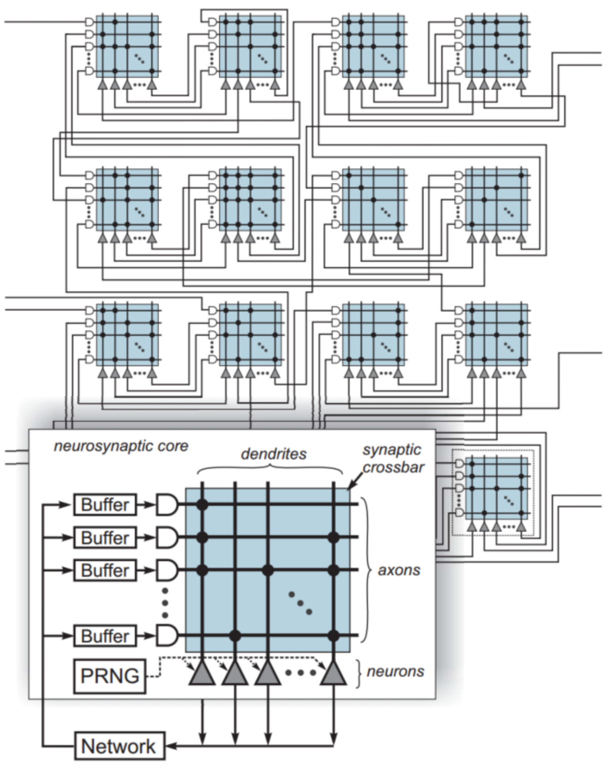


Figure 3.1: Overview of TrueNorth core

From Cassidey et al., [12] we can understand that the TrueNorth neuron can replicate 20 biological neuron functions and behaviors. The overall architecture of the neuron core, the construction and synthesis details have been discussed by Filipp Akopyan et al., [16] followed by Esser et al., [14] where its proven applications have been explained and demonstrated. Also, a new programming language named Corelet programming has been developed by the TrueNorth team, which is detailed in Amir et al., [15] explaining the how different applications can be mapped into these novel hardware cores. Simulation of this model can be done using their newly developed

Compass simulator which is explained in [13].

The power efficiency and computational accuracy of TrueNorth in different applications is a significant advancement and a direct inspiration for this thesis.

Chapter 4
4 Hardware Architecture and Design

4.1 Introduction

One of the most popular usage of neural networks today is in the field of image recognition and computer vision. Such networks usually have multiple layers, one input and output layer each, and the rest as hidden layers. These layers can form anything from a sparse network to a fully connected network. To achieve the same using hardware blocks, our existing neuron has employed suitable architectures, with optimizations at both architecture level and block level. This is being discussed in the following areas.

4.2 Architecture

The architecture of our network is TrueNorth inspired. It is known to be one of the most energy efficient implementations of neural network chips in the world today. A similar but simpler architecture has been followed for our neuron core and its usage, which is discussed in the following sections.

The TrueNorth uses an improved version of a LIF neuron with added parameters to reciprocate more biological functions. Along with it, they have a neuron core, routers and other design blocks such as a router, scheduler, etc., for it to function as a chip. Applications are mapped upon this chip as a platform. The neuron core is the heart of the chip, and this work is focussed on designing a similar neuron core with low power consumption.

4.3 Design

4.3.1 Neuron and connectivity

As shown in a regular neuron diagram, each neuron is connected to a set of dendrites, and the axon from a neuron is connected with dendrites of other neurons to form synapses. Technically, the synapses form the input for the next neuron, with the spikes in each axon being the output of the neuron. The same structure has been used for our hardware implementation with slight modifications, but without changing their logical chain of operations.

Every neuron in this case, will be connected with every other neuron through its mesh of synapses. This results in neurons of one level being input generators for a mesh of axons and dendrites in a logical next layer, forming synapses at each of their junctions. The number of neurons in our neuron core are equal to the number of dendrites or number of axons. The design is parametric and this can be increased or decreased in terms of logarithmic order.

4.3.2 Crossbar design

The aim is to develop a generic neuron core for maximum connectivity with least resources. Here, the dimension of the crossbar is related to the expected number of axons and dendrites in the design. Hence, the best possible design would be a square design similar to TrueNorth, with equal number of axons and dendrites. The number of axons/dendrites can be increased or decreased as required.

With number of axons being equal to number of dendrites, the resulting network would be a perfect square mesh, with synapses scaling itself as we increase or decrease the count. This network is a crossbar switching network, as shown in the figure 4.1.

The number of axons and dendrites are parameterized and can be changed as required (in powers of 2 to be aligning with digital design strategy). Each of the synapses formed by this mesh will also have an enable signal to dictate if its an active synapse or inactive synapse, giving us the control to change the status of a synapses whenever required.

4.3.3 Synaptic weights memory in crossbar

For the crossbar discussed above, each synapses in it will store 8 bits of data, which represents the synaptic weight. The weights are the core knowledge of the network. In our case, a network can be trained offline and the weights can be stored in synapses.

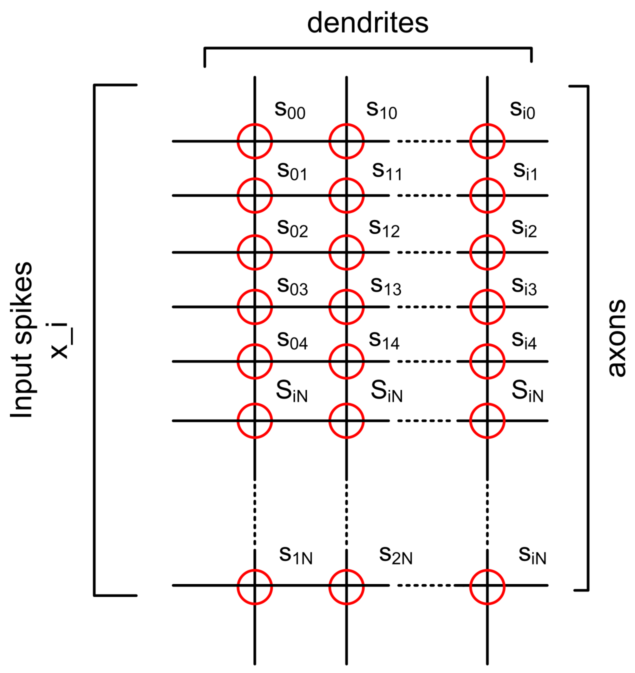


Figure 4.1: A crossbar architecture

These memory for the synapses are major memory consumers in our neuron core which is discussed in detail in a following section.

Apart from synaptic memory, we will also be storing neuron parameters. These, however are stored directly within each neuron and are not part of the synaptic crossbar.

4.3.4 Synaptic memory controller

Memories are the most important segments of any neural network design. With memories come always a memory controller that helps in fetching data and replacing them based on all of our operations. These memory transactions will be contributors for dynamic power in our design and is wise to keep this under our control. For this design, a synaptic memory controller is designed to read synaptic weights from memory.

4.3.5 Neuron core

A neuron core is a collection of neurons along with its connection of axons, dendrites and synapses. It can be said that it is one fully built functional body, that can act like one layer of a neural network. Based on our requirements it can be instantiated multiple times to form a bigger and denser neural network. We can interface external processors to this as required, to act as a coprocessor or a neural network accelerator. The operations of this neuron core will be deterministic and can be reconfigured as required with complete access to an user to change synaptic weights and other neuron parameters based on requirements.

4.3.6 Clock gating

Clock gating is a popular technique in digital design for low power circuits, which helps in reducing dynamic power dissipation by disconnecting clocks to inactive regions of the chip. This technique has been employed in the design for LIF neuron core as well.

Inside the neuron core, clocks have been divided into three activity branches - Neurons, weights memory controller and synapses. The operating period for each of these design areas are different from each other and not continuous. Hence, they've been giving separate controls to gate their clocks based on enable signals for the respective blocks. Clock gating would be discussed in detail in the section for LIF neuron core.

4.3.7 External interface to neuron core

The neuron core would be required to interface with external processors or micro-controllers to operate as a co-processor or accelerator. For this, the neuron parameters and controls, which are pinouts now, would have to be interfaced to a controller. The design of such a controller in a real "SNN" chip, as such TrueNorth, requires the design of a whole network-on-chip router to carry spikes to different cores within the chip and outside. This is beyond the scope of this work now, but could be carried out as a future work.

Chapter 5
5 Neuron design

5.1 Introduction

The focus of this chapter is the design of a spiking neuron in digital CMOS logic. Among the types of spiking neuron models discussed in section 2.7, it was decided to implement a Leaky, integrate and fire neuron due to its simplicity and extendability. Even state-of-the-art designs like TrueNorth [12] are variants of this neuron model. Hence this choice of neuron model is good for preliminary exploration such as this work.

5.2 LIF neuron

The Leaky, Integrate and Fire (LIF) neuron is one of the simplest spiking neuron models, popular also for the ease with which it can be analyzed and simulated. From a hardware perspective, its simplicity helps us achieve low area and power consumption. This neuron is modeled as a "leaky integrator" of its input I(t):

$$\tau_m \frac{\mathrm{d}v}{\mathrm{d}t} = -v(t) + RI(t) \tag{5.1}$$

Breaking it down, LIF neuron model can be described using five operations namely, synaptic integration, leak integration, threshold, spike firing and reset. These are represented by equations shown below.

The equation for synaptic integration is given as

$$V_j(t) = V_j(t-1) + \sum_{i=0}^{N=1} x_i(t)s_i \tag{5.2}$$

For Leak integration, the equation is

$$V_j(t) = V_j(t) - \lambda_j \tag{5.3}$$

Threshold, Fire and Reset operation is depicted in the pseudo code here

```
If V_j(t) >= alpha_j
   Spike
   V_j(t) = R_j
end if
```

The membrane potential $V_j(t)$ of j-th neuron at time step t can be denoted as the sum of potential from the previous time step with the synaptic input. The synaptic input is a product of the input spike to the synapse with its weights s_i. The resulting potential is then subtracted with the leak value. Finally, the prevailing membrane potential is compared with the neuron threshold value, to determine if the neuron has to spike and reset or not. If the membrane potential is greater than or equal to the neuron threshold, it fires a spike and resets the membrane potential to the reset voltage Rj. The different phases of neuron operation can be understood better from the figure 5.1.

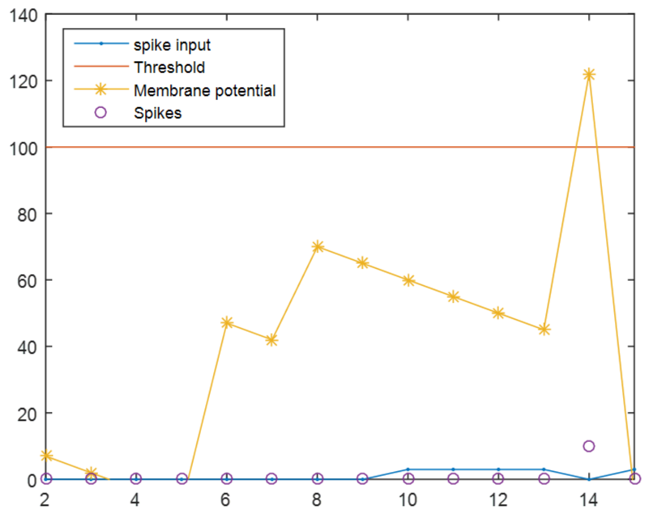


Figure 5.1: Simulation of LIF neuron

5.2.1 Neuron hardware architecture visualization

The figure 5.2 shown below depicts the proposed architecture for implementing the LIF neuron.

LIF neuron has three inputs which are: Membrane potential, spike input and the synaptic weights. The membrane potential can be initialized with zero or any value if required. Spikes are treated as events and are therefore represented by a single bit (1 means fired). The synaptic weights can also be set by the user based on the neuron model we wish to employ.

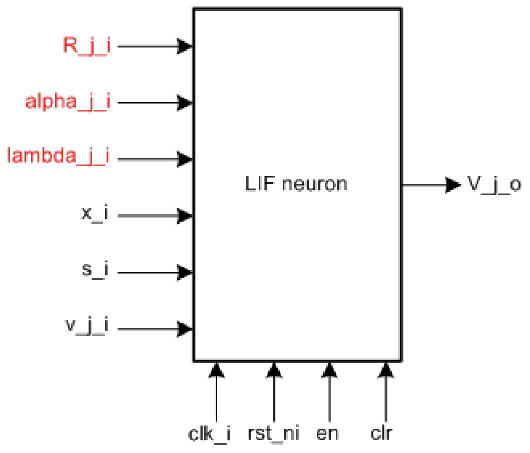


Figure 5.2: Top level block diagram of LIF neuron

The possible date type and the no. of bits for each of it are discussed below:

- Membrane potential - signed integer - 8 bits
- Spike input - 0,1 - 1 bit
- Spike output - 0,1 - 1 bit
- Synaptic weights - signed integer -8 bits
- Neuron threshold - signed integer -8 bits
- Leak value - unsigned integer -8 bits

Here, the size of the synaptic weights is a design parameter, and 8-bit is only the default value in this work. The neuron is a fully synchronous design, with inputs and outputs of the neuron working based on a single global clock. Clock gating technique is used as required through out the design.

5.2.2 Internal structure of the neuron

The internal structure of the neuron speaks about the control and data flow inside the LIF neuron model. It is depicted in the figure 5.3.

5.2.2.1 Synaptic Integration

Synaptic Integration block is fully synchronous and has two inputs - spike input $x_i(t)$ and synaptic weights $s_i(t)$. Spikes are the inputs for the neuron which can either be '0' or '1'. Synaptic weights refers to the strength or amplitude of connections between

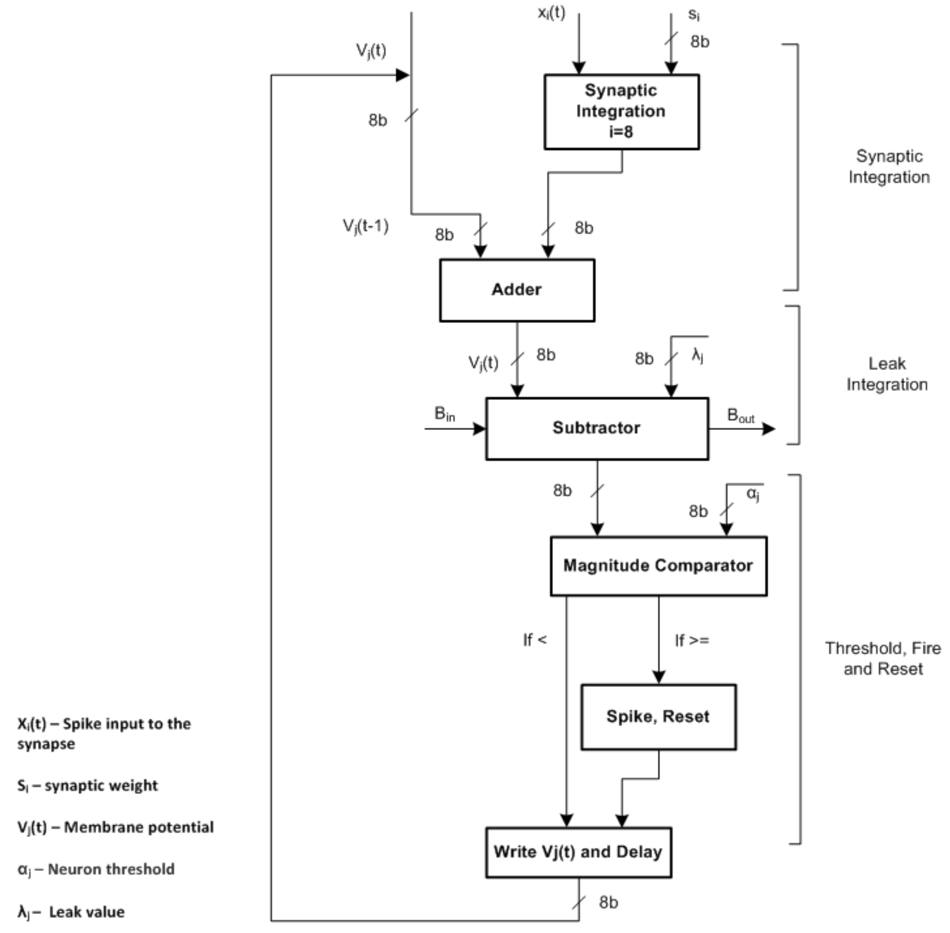


Figure 5.3: Hardware structure and flow of LIF neuron

two neurons. In this neuron model, they will be with 8 bits as signed integers ranging from -128 to 127.

Inside synaptic integration, the spike inputs are multiplied with all synaptic weights. In our case, we have about 8 synapses. Here, since the spikes are of only one bit, these inputs will be multiplied with the available weights using AND gates, which saves us from the use of expensive multipliers. The resulting synaptic sum will then be added with the existing membrane potential. At reset, the membrane potential will be ’zero’. The structure of synaptic integration is shown in figure 5.4.

The synaptic integration block would result in larger values beyond 8-bits when adding multiple input values. Here, the adder has been saturated, so that values larger than 8 bits are set to the highest possible values with the same 8-bit length.

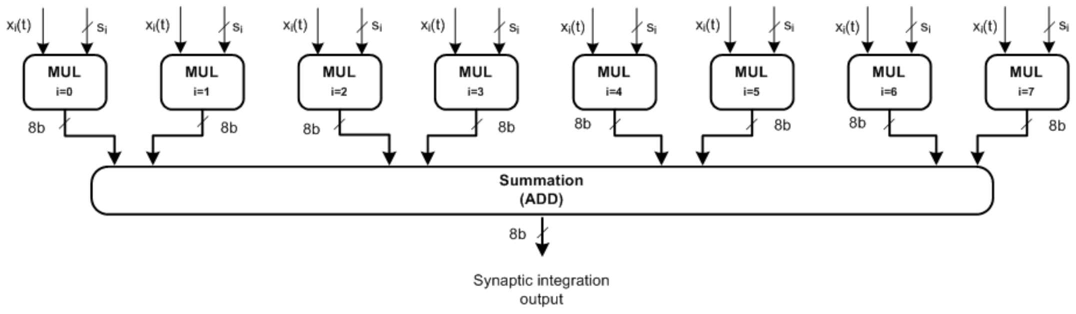


Figure 5.4: Internal architecture of Synaptic integration - Direct implementation

5.2.2.2 Leak, Integrate and Fire

After the synaptic sum is received from the synaptic integration module, the result is then added with the existing membrane potential V_j from the previous cycle. A user configurable leak value will be applied to it. After which, the result will be compared with a user configurable threshold value to decide on the spiking output. If the value is higher than the threshold, then a spike is generated. This is followed by the reducing the membrane potential V_j in the next cycle to a value given R_j. After the spiking and firing event, the result will be delayed by one clock cycle to act as input for the next set of incoming spikes.

A model of the same neuron with 8 synaptic inputs and weights was made in Matlab, and the plot of the results membrane potential and the spikes are as shown here. The membrane potential increases until it reaches the threshold value of 100, after which it fires a spike and resets the membrane potential value to zero. The input spikes are also shown in the figure.

5.3 Neuron model in Matlab

To understand the functioning of this type of neuron, the Leaky, integrate and fire neuron equations have been converted to a Matlab model. From the model we can have accurate numbers of membrane potential and spikes, based on the simulation inputs. From a generic implementation point of view, the neuron parameters have also been included, giving us flexibility in changing different parameters and observing how a LIF neuron behaves.

Apart from understanding the neuron, the motive is also to construct this basic model, to aid us in validation of our HDL design. The data from the model was used as a golden reference to evaluate our HDL design's proper functioning.

From HDL simulation of LIF neuron, the values were dumped into a text file and later compared with a similar dump from Matlab model. This way the operation of the neuron was verified for every instant for the signal in our design.

5.4 Implementation perspectives

5.4.1 Direct implementation

The first approach to build a neuron core using LIF neuron is a direct approach. We convert the Matlab model into an equivalent HDL code. As described earlier, the neuron would fire whenever the membrane potential is higher than threshold. The hardware architecture of a direct implementation is the fully-parallel adder model discussed in section 5.5.1

For a single neuron, depending on the number of synapses it has, the hardware units to build it will increase or decrease. This would in turn increase or decrease the area of design, and is going to have a direct impact on power consumption of neuron core.

Looking at the model and architecture of this neuron, we can see that it would be using multiplier and adders to perform synaptic integration, a subtractor to perform leak, and comparators to compare membrane potential with threshold. Thus a theoretical estimate of the gate count for a single neuron was done.

Gate Estimation

To begin with gate estimation was started assuming a total of 8 synapses. It has to be noted that for synaptic integration we consider AND gates for multiplication of spikes with weights, eliminating the need for multipliers. This was done owing to the fact that we are using a spiking neural network, where spikes are going to cost us 1-bit of data. For rest of neuron parameters, the specifications discussed in section 5.2.1 has been used.This gives us the estimated number of gates for a single neuron, which is shown in table 5.1.

Table 5.1: Gate estimation for LIF neuron with 8 synapses

Operator/logic	Usage count	No. of gates
Synaptic Integration	1	344
Adder	1	40
Subtract	1	56
Magnitude comparator	1	62
Flip-flops	3	40
Total gates		542

The results of a single neuron were extended to higher number of neurons to see how the requirements are scaling for a larger neuron core. From 8 synapses, a sweep was performed upto 1024, which gave us a prediction of how big the hardware neuron core could be. Though the exact numbers were not important, this helped us understand each segment as we scale up in size and number, which is shown in figure 5.5.

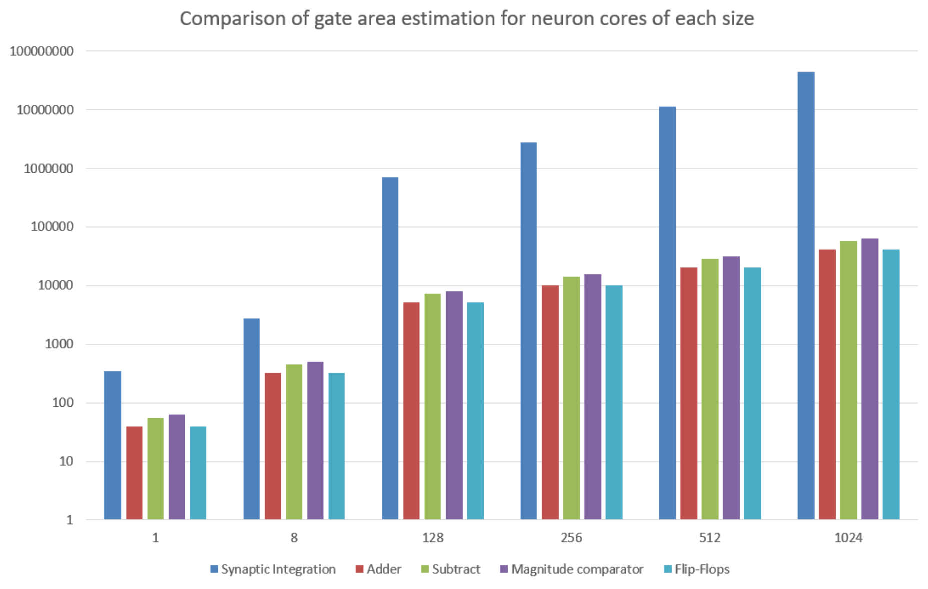


Figure 5.5: Comparison of Gate estimation for neuron cores

Now, it is clear about which segment of neuron is resource intensive, becoming a candidate area to focus our efforts in design space exploration and optimization. Synaptic integration, with the highest area and growth as we scale up, requires attention when implementing in hardware.

5.4.2 Implementation alternatives

With the direct implementation as discussed in previous section, we might end up with a large power consumption making it unsuitable for low power applications. Keeping this in mind, we could employ general methods, which reduce the area of any implementation, consequently reducing power consumption. So, we planned on reducing the gates, which could be achieved by different means which are discussed in short below.

- Reducing number of multipliers and adders
- One neuron each layer
- Reducing number of layers

5.4.2.1 Reducing Number of Multipliers and Adders

One of the most common ways of reducing area is to reduce the number of multiplier and adders. To reduce their number, the computations and calculations in our neuron has to be simplified and optimized. This will be considered in our design.

5.4.2.2 One Neuron each Layer

In our direct implementation, we have considered our neuron core to be having multiple neurons. This could be reduced to have a single neuron that is repeatedly used to compute the potential for every possible combination in a layer of our design. This could save us costs of gates.

5.4.2.3 Reducing Number of Layers

Since every neural network would consist of multiple layers, it is natural to have multiple instances of a single neuron core to form a larger and denser network. While this method is effective and fast, it is very expensive interms of area and power. Hence the idea of time multiplexing a single neuron cores is also considered here.

5.5 Design Space Exploration (DSE) with Synaptic Integration

In previous section, from figure 5.5, it was noticeable that synaptic integration is the most expensive region of neuron. Optimizations in this area would lead to good decrease in area and power as it is scaled up with the number of neurons and neuron cores. Different implementations are experimented in this section, keeping in mind the objective of getting low power consumption.

To have an Ultra low power neuron core, it was important to have a neuron with the smallest footprint. It was also important to relieve latency losses in our design. Finding the sweet spot between cell area and latency were our priorities here. So alternative ideas to reduce area of synaptic integration were considered and possible architectures are listed and discussed below

- Serial-Parallel Adder
- Parallel-Serial Adder
- Bitwise-Parallel Serial Adder
- Bitwise-Serial Parallel Adder
- Bitwise-Serial Serial Adder

5.5.1 Fully Parallel Adder

In synaptic integration, all available synapses are added together. From the discussion in direct implementation section 5.4.1, it can be seen that synaptic weights could be added in parallel using a fully parallel adder. A design with this architecture is termed as FPA design henceforth.

Here, incoming spikes are multiplied with synaptic weights before their summation as shown in figure 5.4. It is done inorder to consider only those weights which have active connections. Incoming spikes are always 1 bit, therefore multiplier are always implemented as a set of AND gates. This results in relatively lesser area. This will be an optimization implemented in all our designs discussed further. The updated image for synaptic integration is shown below.

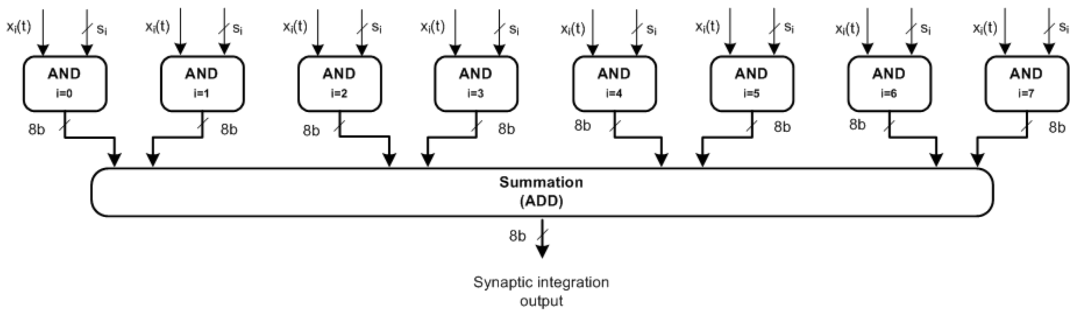


Figure 5.6: Synaptic Integration with AND gate

Table 5.2: Synthesis results for FPA neuron architecture

Parameter/Dendrites	8	16	32	64	128	256	512	1024
Combinational area	1044.00	1816.92	3433.68	6761.88	13632.84	27805.32	57141.36	117345.23
Buf/Inv area	54.72	57.60	57.60	63.00	68.76	43.92	48.96	47.88
Noncombinational area	118.80	129.60	140.40	151.20	162.00	172.80	183.60	194.40
Total cell area	1162.80	1946.52	3574.08	6913.08	13794.84	27978.12	57324.96	117539.63

This design has been synthesized and the results are shown in table 5.2. From the table it is evident that it occupies an increasing number of adders as the number of synapses in our design increases. For every increase in number of synapses, there is an increase in number of adders used in the design. This results in a large area. Theoretically this will be the design with the largest area and smallest latency.

5.5.2 Fully Serial Adder

From table 5.2 it is clear that FPA architecture occupies large area as the number of synapses increases from 8 to 1024. The immediate solution to reduce the area would be a fully serial adder design for synaptic integration. An adder-accumulator is used for this purpose, where synaptic weights are added one by one for every clock. A single full adder was used followed by a register to hold the current output to serve as an input for next addition. After a series of n additions over n clock cycles, we would have our final result.

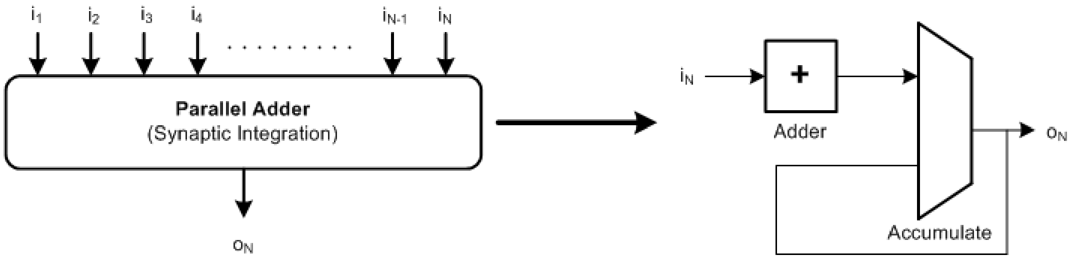


Figure 5.7: Fully Parallel to Fully Serial Adder for synaptic integration

As shown in figure 5.7, this design reduces the usage of adders from 'n-1' adders to a single adder. Ideally, this is the lowest possible area for synaptic integration. But, in terms of latency, this design is poor taking 'n' clock cycles. Hence, other implementation options for synaptic integration will be explored in following sections.

Table 5.3: Synthesis results for FSA architecture

Parameters/Dendrites	8	16	32	64	128	256	512	1024
Combinational area	708.94	1025.64	1642.32	2865.6	5270.76	10107.72	19757.16	38833.2
Buf/Inv area	57.96	68.76	81.72	124.92	194.04	355.32	646.2	1030.68
Noncombinational area	267.12	277.92	288.72	299.52	310.32	321.12	331.92	342.72
Total cell area	975.96	1303.56	1931.04	3165.12	5581.08	10428.84	20089.08	39175.02

5.5.3 LIF neuron with SPA

Multi-stage adders are a good way to optimize addition operations for a large number of inputs. Here, in a Serial-Parallel Adder (SPA), the adder tree is divided into two stages. An adder which adds inputs serially is employed for first stage, calling it a serial adder(SA). This is followed by an adder which sums inputs in parallel, calling it a parallel adder(PA). With these two elements, a two stage adder is proposed. A SA is used for the widest stage, and its results are then summed by using a PA.

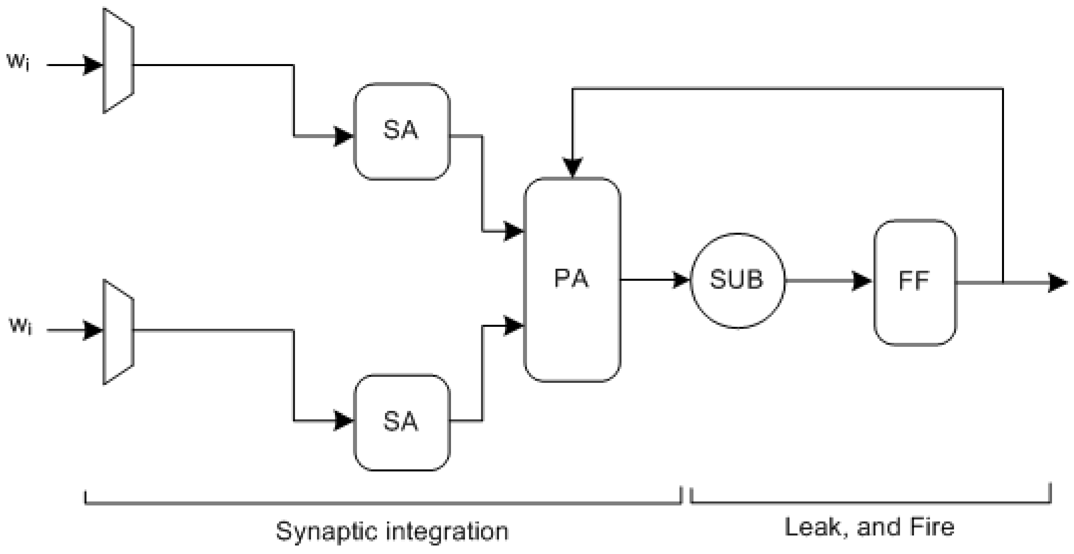


Figure 5.8: Serial Parallel Adder for synaptic integration

Here, PA waits for multiple inputs to be available from the SA stage, and processes it in a single cycle. Through this method, the latency incurred from a FSA design is partially mitigated due to the usage of PA in its second stage. Multiple instances of SA are used to process inputs simultaneously to further reduce latency in operations. At the end of every SA, flip flops are used to store its result for the next stage. This adds up to our area and power as the number of synapses and neurons are increased, which is reflected in our synthesis results for SPA architecture in table 5.4. These results were obtained with the SA stage processing 8 inputs for one operation by PA in second stage.

Table 5.4: Synthesis results for SPA architecture

Dendrites	8	16	32	64	128	256	512	1024
Total cell area	900.00	1362.24	2320.20	4214.52	7999.92	15555.96	30674.88	60858.36

The architecture of the SPA based LIF neuron is shown in figure 5.8. As the total number of synapses grows, the second stage PA would increase in size but not in latency. Whereas, the first stage wouldn't grow in area, the latency would grow as a trade-off.

SPA Exploration with block size

Here, it was noted that there was more space for exploration using different sizes for SA instead of a fixed size of 8. The number of inputs processed by SA/PA stage, which can be termed as "block size", was swept from 8 to 64 or so, depending on the total number of synapses. Results of different block sizes of first stage SA, have yielded varying results, which are also presented here in table 5.5

Table 5.5: Synthesis results for Block size sweep in SPA architecture

Dendrites	8	16	32	64	128				256				512				1024			
Block size	8	8	8	8	8	16	32	64	8	16	32	64	8	16	32	64	8	16	32	64
Total cell area	965.52	1259.64	1495.08	2133.72	3531.24	6574.32	11279.88	12857.40	5614.56	10722.24	18670.32	22477.68	11525.40	22725.36	43750.08	48927.52	23265.36	46784.88	90653.40	98609.68

With increasing block size of SA, total cell area of neuron becomes lesser. This is a direct impact of reduction in number of instances of SA in first stage, and number of inputs for PA in second stage. This trend can be witnessed in table 5.5, where synthesis results of different block sizes are presented. It has to be noted that, increasing block sizes also affect latency proportionally due to larger number of inputs to be processed serially.

5.5.4 LIF neuron with PSA

This design is the opposite of the SPA design. The adder tree is divided into two stages here too, where as the first stage is a parallel adder(PA) and second stage is a serial adder(SA). The PA in the first stage is restricted to 8 inputs at a time. When there are more than 8 input synaptic weights, the PA will go for consecutive rounds until all inputs are summed up. The results of the first stage are seamlessly sent to the next stage. Since we sum 8 inputs for every clock in the first stage, the result serves as input to SA in the second stage every clock cycle. After series of repetition, all the values would have been summed up. This is shown in figure 5.9.

Similar to the previous case, the PA was initially restricted to a block size of 8. Synthesized cell area for this architecture of neurons are presented below in table 5.6. It is clear that PSA architecture with a block size of 8, saves us cell area from 16 to 80% for a single neuron.

Table 5.6: Synthesis results for PSA architecture

Dendrites	8	16	32	64	128	256	512	1024
Total cell area	965.52	1259.64	1495.08	2133.72	3531.24	5614.56	11525.40	23265.36

PSA Exploration with Block size

There was more room for exploration by changing the block size of parallel adder which could vary the performance and cell area accordingly. This gave us more options to select the best possible architecture. The design was synthesized for a different values and their results are as shown in table 5.7

The results of the table shows the increasing cell area of PSA architecture as we increase the size of a block of PA. This is inline with theory, where the size of

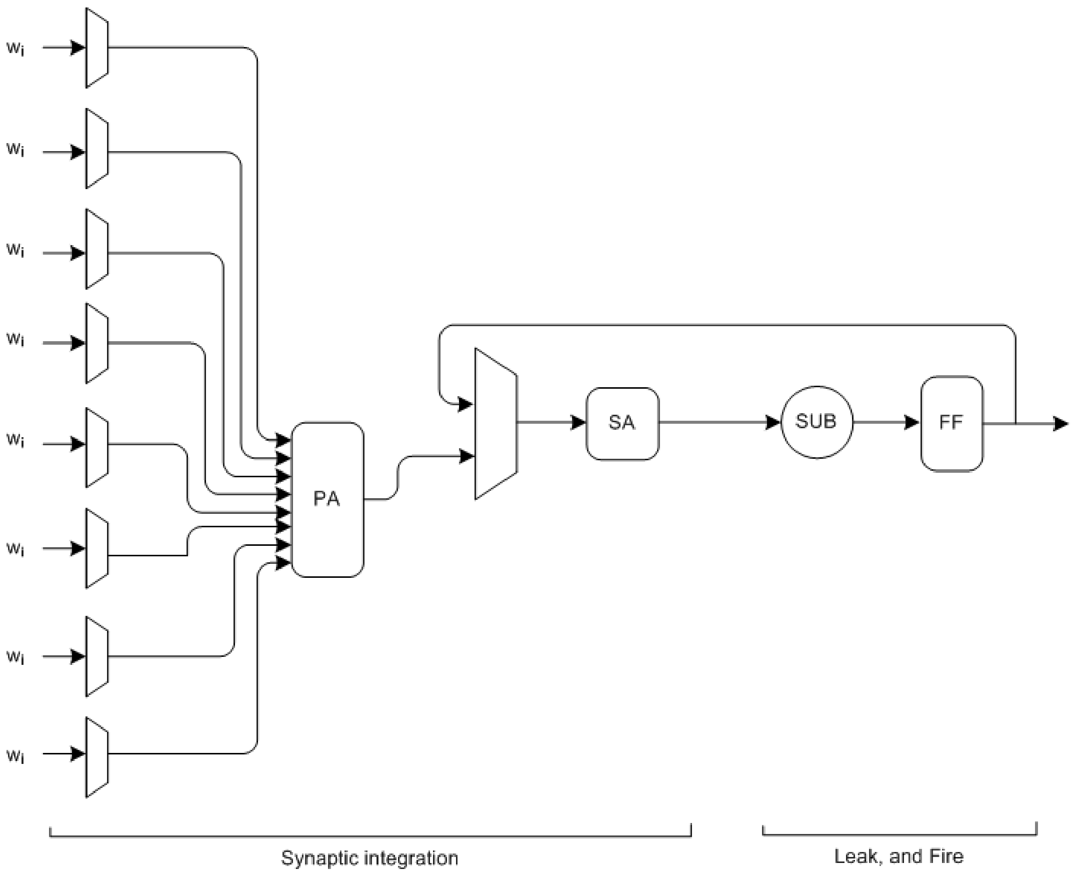


Figure 5.9: Parallel Serial Adder for synaptic integration

Table 5.7: Synthesis results for Block size sweep in PSA architecture

Dendrites	8	16	32	64	128				256				512				1024			
Block size	8	8	8	8	8	16	32	64	8	16	32	64	8	16	32	64	8	16	32	64
Total cell area	965.52	1259.64	1495.08	2133.72	3531.24	6574.32	11279.88	12857.40	5614.56	10722.24	18670.32	22477.68	11525.40	22725.36	43750.08	48927.52	23265.36	46784.88	90653.40	98609.68

PA would increase and eventually result in larger area and lower latency. It falls on the user to decide on how much one is willing to sacrifice area and power for latency.

5.5.5 LIF neuron with BPSA

Through out all previous methods, the weights were handled as bytes. In this architecture, a low-level structured approach - Bitwise operations have been followed. We are aware of the manual way by which we perform addition on a paper. Vertical bits are added together for every bit position starting with LSB and finally results of these are summed up to get a final result. A very similar approach has been followed in this.

All vertical bits have been added in parallel and stored. This results in using a 1 bit adder for the respective number of synapses rather than the usual 8 bit adders. Assuming we have 8 inputs, we will be adding the LSBs of these 8 inputs using a parallel adder (PA) which has an input width of 1 bit. Results of this bitwise operation are then added serially using our regular adder-accumulator or SA. This results in using lesser number of adders in our design.

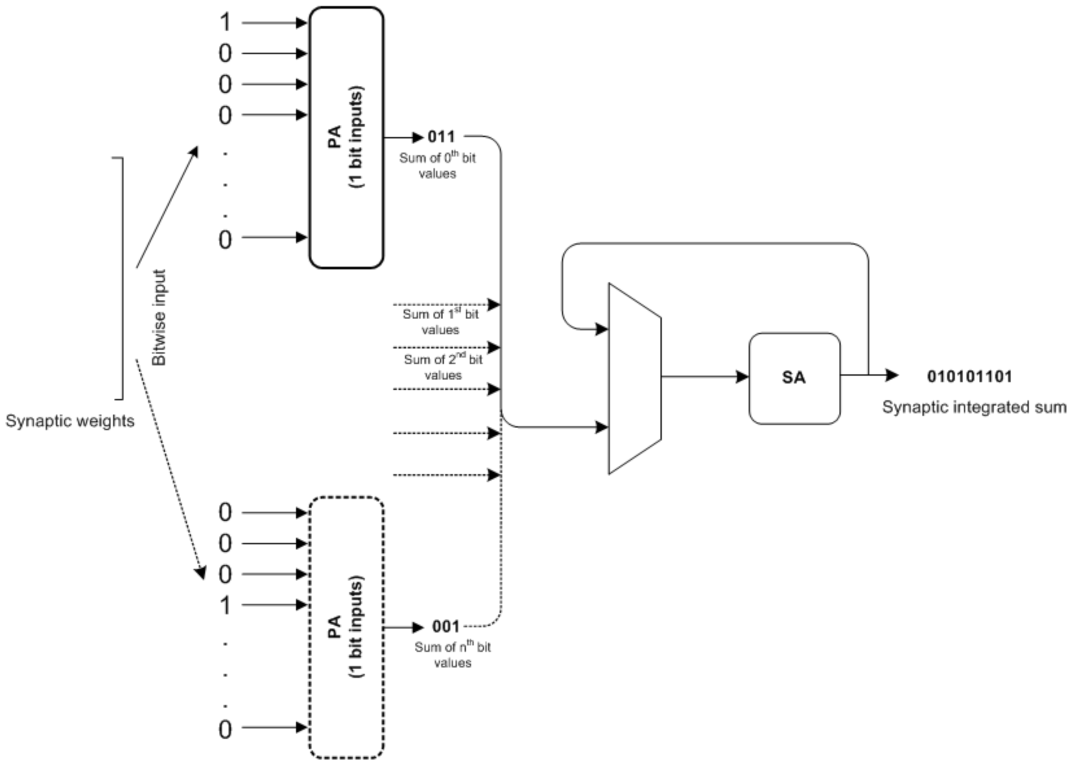


Figure 5.10: Bitwise-Parallel Serial Adder for synaptic integration

In this type of synaptic integration, the size of the parallel adder increases as the number of inputs to synaptic integration increases. And since our weights are of 8 bits each, we will using 8 instances of these n-input-1bit-PAs followed by 8 bits SA. This requires usage of large multiplexers before the parallel adder section, to drive the respective bits as inputs. And, these multiplexers prove to be expensive in terms of area. Total cell area of this architecture is listed in table 5.8

Table 5.8: Synthesis results for BPSA architecture

Dendrites	8	16	32	64	128	256	512	1024
BPSA	936	1221.84	1799.28	3044.16	6364.44	15332.76	40870.8	128301.8

Looking deeply at the architecture for this neuron, we noticed scope for improvement inside the PA. Since one could be adding any number of inputs using PA, it was decided to structure the PA with different stages, which could improve the cell area and prevent it from getting bloated due to congestion. By this, we number of inputs a parallel adder has to process was reduced by 2, 4 and 8 times, which helps in reducing its cell area, when the number of synapses are increased from 8 to 1024. Each of these were synthesized and was compared with regular implementation, results of which are shown below in table 5.9

Table 5.9: Synthesis results for BPSA architecture with structuring

Deindrites	8	16	32	64	128	256	512	1024
BPSA-PA	936	1221.84	1799.28	3044.16	6364.44	15332.76	40870.8	128301.8
BPSA_Struct_2PA	915.12	1124.64	1649.88	2780.28	5218.56	12147.48	32604.84	115823.52
BPSA_Struct_4PA	925.56	1121.76	1562.40	2555.28	4691.52	9296.28	22785.48	70306.20
BPSA_Struct_8PA	878.04	1114.56	1563.48	2408.04	4310.64	8154.36	17083.44	47084.04

For simplicity of comparing our results later, when we mention BPSA design, it points to the BPSA_Struct_8PA since it has the best results in this category.

5.5.6 LIF neuron with BSSA

The above said BPSA design was refined a bit more to arrive at this architecture. The usage of parallel adders in the previous architecture for bitwise addition of values in each position has been replaced with a serial addition operation. This results in even more lesser number of adders in our design. On the other hand, we are also increasing the latency of the design, getting close to the lowest latency of a fully serial design. The architecture of this adder design is shown in figure 5.11.

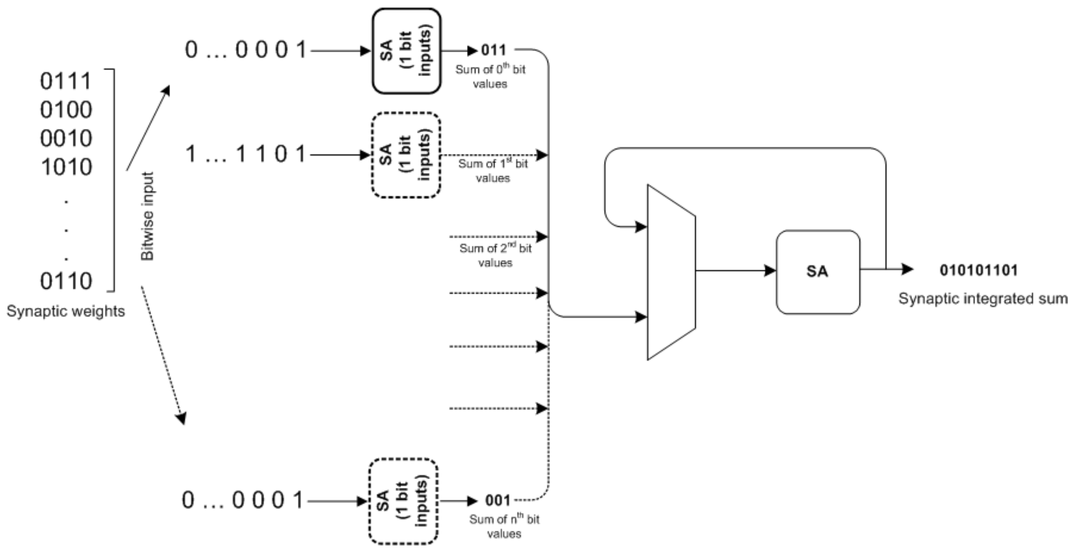


Figure 5.11: Bitwise-Serial Serial Adder for synaptic integration

Assuming we have 8 input weights, we will be adding the LSBs of the weights using a 1 bit adder serially, which is going to consume 7 clocks for add them. The same would be followed for the other digits as well. After these additions, the resulting values will be added regularly using a SA. This way, we reduce the amount of adders used in the parallel addition stage, but end up increasing the latency even more. Synthesis results of this architecture neuron is presented in table 5.10.

Table 5.10: Synthesis results for BSSA architecture

Dendrites	8	16	32	64	128	256	512	1024
BSSA	1214.64	1822.32	2284.56	4011.84	6139.44	10324.44	20973.96	40304.16

5.6 Observations from DSE

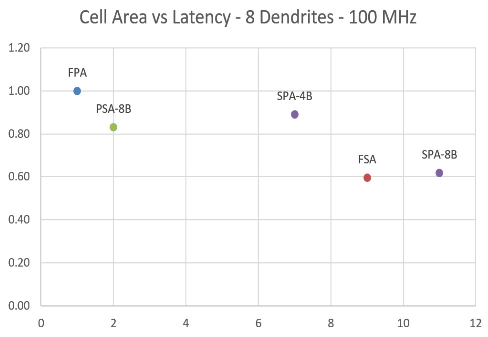
Having explored the entire space between a fully parallel adder to a fully serial adder design, it is important to analyze, understand and weigh on the pros and cons of each design. To do this, synthesis results of all the above architectures have been presented together here.

The plots shown in this section are a comparison of total cell area vs latency of each implementation of neuron discussed in section 5.5. The values in the plots are normalized with respect to the result of FPA design (whose exact value is shown in table 5.2).

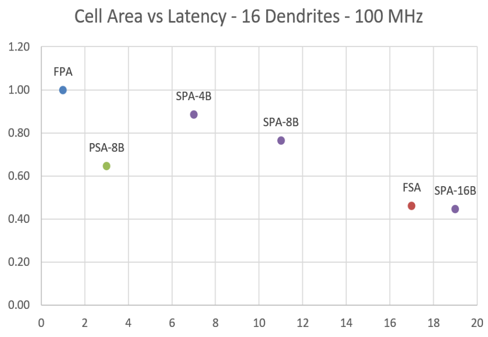
From the theory discussed and results shown in previous sections, it is known that FPA and FSA design has the largest and smallest cell area respectively in these architectures. This fact is also acceptable looking at the plots in figure 5.12, with a slight margin.

When it comes to SPA architecture, results from sweeping of block sizes have been included in the plot. As a result, we can see how changing block sizes of SA in first stage is changing the dynamics of the LIF neuron. With a smaller block size, required number of instances of SA are higher, which results in larger cell area. With a smaller size, the number of clocks required for SA to finish a set of inputs is low and this is directly reflected in the latency of this type of neuron. This property with latency is consistently displayed in all plots ranging from 8 dendrites to 1024 synapses. On the other hand, with higher the number of synapses, the design started consuming more area with lower block sizes. Upon investigating into the implementation, it was clear that with smaller block sizes, the memory had to feed input data to larger number of SAs simultaneously. This resulted in implementation of multiple multiplexers and decoders to retrieve data from a single memory block and drive it to.

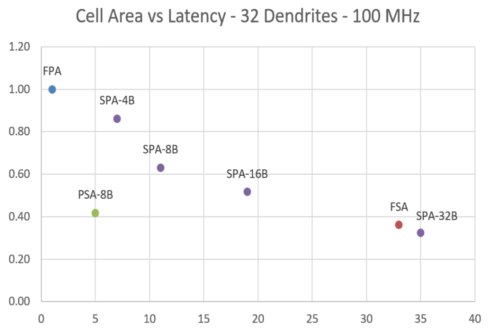
Figure 5.12: Comparison of results from different neuron architectures - Image 1



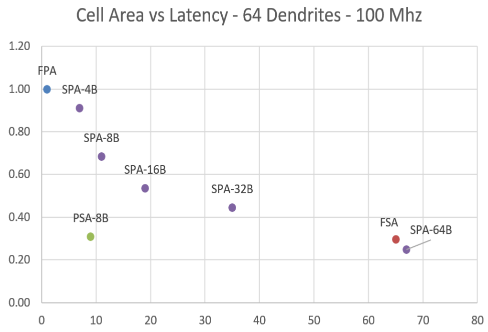
"Results with 8 Dendrites"



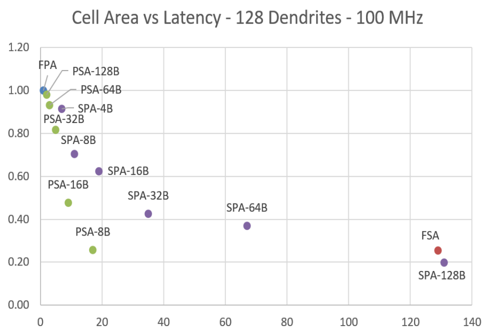
"Results with 16 Dendrites"



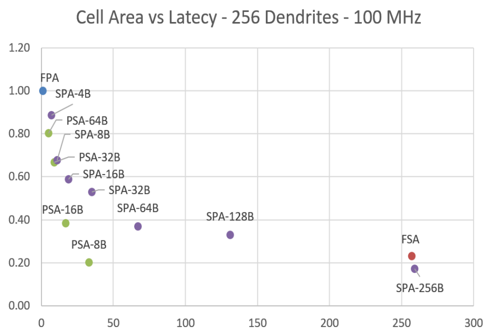
"Results with 32 Dendrites"



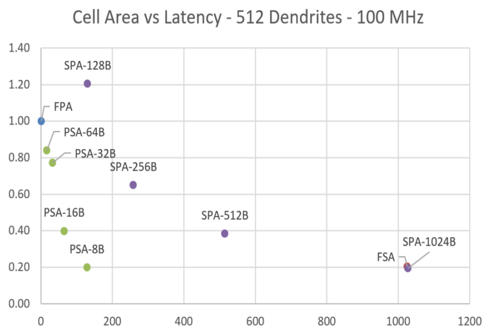
"Results with 64 Dendrites"



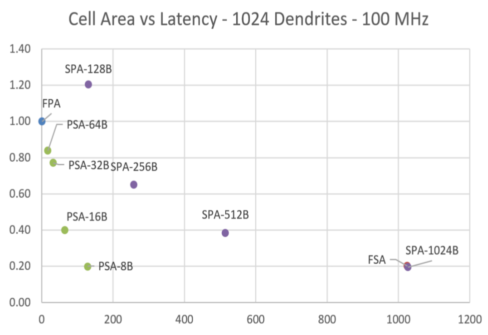
"Results with 128 Dendrites"



"Results with 256 Dendrites"

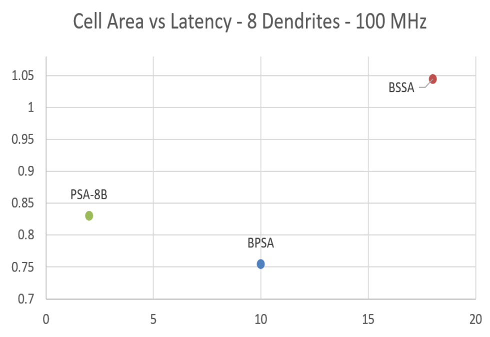


"Results with 512 Dendrites"

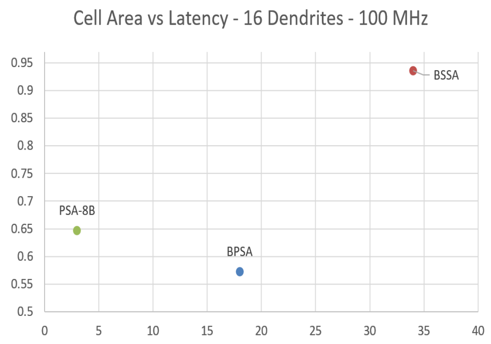


"Results with 1024 Dendrites"

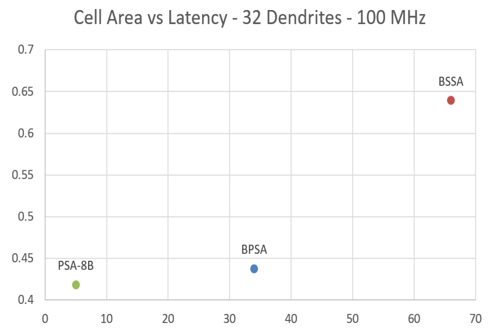
Figure 5.13: Comparison of results from different neuron architectures- Image 2



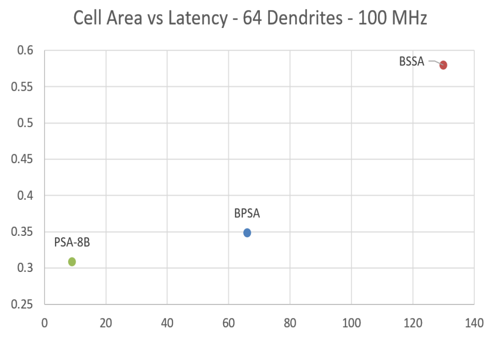
"Results with 8 Dendrites"



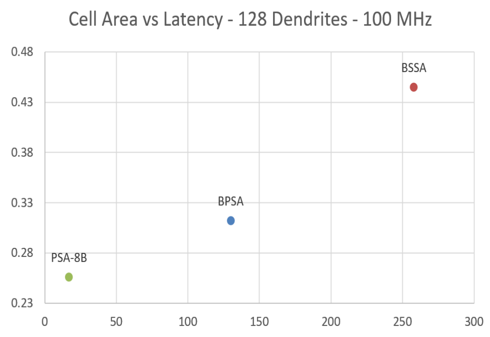
"Results with 16 Dendrites"



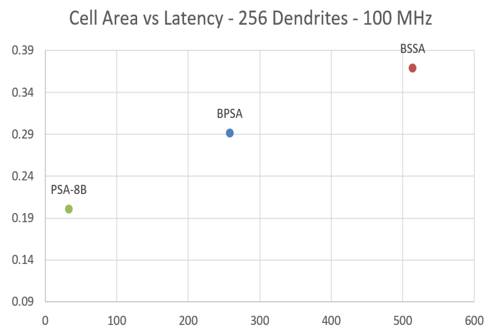
"Results with 32 Dendrites"



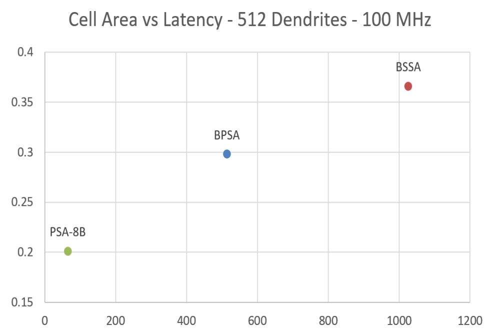
"Results with 64 Dendrites"



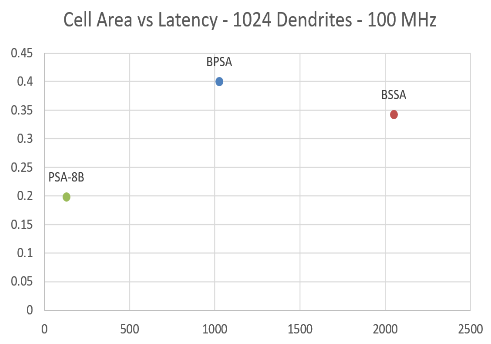
"Results with 128 Dendrites"



"Results with 256 Dendrites"



"Results with 512 Dendrites"



"Results with 1024 Dendrites"

Chapter 6

6 LIF Neuron Core

6.1 Neurons

Through out the previous sections, we have explored different architectures for implementing a LIF neuron. Making changes following each of the methods discussed in section 5.4.2 would change area and static power consumption in its own ways. With all those options to choose from, we base our selection based on the following parameters.

- Power
- Area
- Latency of operations
- Efficiency of operations

While our primary objective is to have lowest power consumption, power is usually influenced by area, efficiency of operations and speed of operations. So all of the above mentioned parameters have been considered while looking at results of each architecture. From those, we can see that we achieve a good equilibrium between low area and low latency with PSA-8B architecture as shown in section 5.5.4. Using this architecture and coupling this with proper clock gating technique, it is possible to bring down the leakage power as well as the dynamic power. Taking care of low area will help us reduce leakage power by transistors, where are dynamic power is reduced by clock gating techniques and controlled execution of operations.

Hence, PSA-8B neuron architecture has been selected to proceed with designing a neuron core.

6.2 LIF neuron with crossbar

Further to this, the next logical step is to construct the synaptic crossbar, which is one of the important segments of the neurosynaptic core. The crossbar is a matrix of the signals - axons and dendrites. The junction of axons and dendrites will be our synapses. For preliminary estimation, we will consider a crossbar with 8 axons and 8 synapses here, forming about 64 synapses. Each of our synaptic weights are with 8 bits amounting to 512 bits to store the synaptic weights for this crossbar.

Input spikes to a neuron core will be entering through the axons and exiting at the output of neurons as newer spikes. In this path, input spikes get multiplied with the synaptic weights in each of these synapses and proceed to the neuron computation part, where leak values are applied and compared with threshold. Based on the resulting values, spikes will be outputs of the neurons.

For ease in construction, the synaptic integration part has been design as a separate segment from the crossbar with memories. The value will be summed up for each of the dendrites and will serve as input to the neuron. The membrane potential from the neuron will also be given as a feedback to the dendrites, to sum along with weights in each of the synapses in their respective dendrites.

Synaptic connections can be enabled and disabled as required. A register access is provided for this purpose. Also, the weights of each synapses are given 8 bit registers, which can be updated as and when required by the processor, to run different neuron types for different applications.

The summary of synthesis reports for the LIF neuron with different number of synapses is given in table 6.1

Table 6.1: Area of LIF neuron for a range of synapses

Design/#Dendrites	8	16	32	64	128	256	512	1024
PSA-8B	965.52	1259.64	1495.08	2133.72	3531.24	5614.56	11525.40	23265.36

6.3 Weights controller

Though the actual neuron is only the parts with Leak, integration and fire, the neuron architectures from section 5.5 also contained a weights memory controller that helps in fetching synaptic weights and spikes. This is depicted in figure 6.2. Moving the controller out of the neuron design reduces the size of the actual neuron even more, in all of the above cases. This has been put to advantageous use in the later section when building the neuron core.

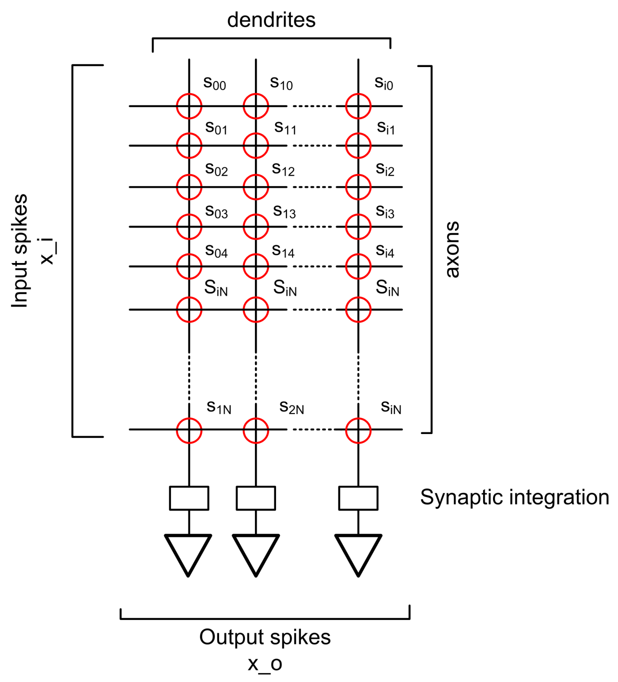


Figure 6.1: Architecture of LIF neuron with crossbar

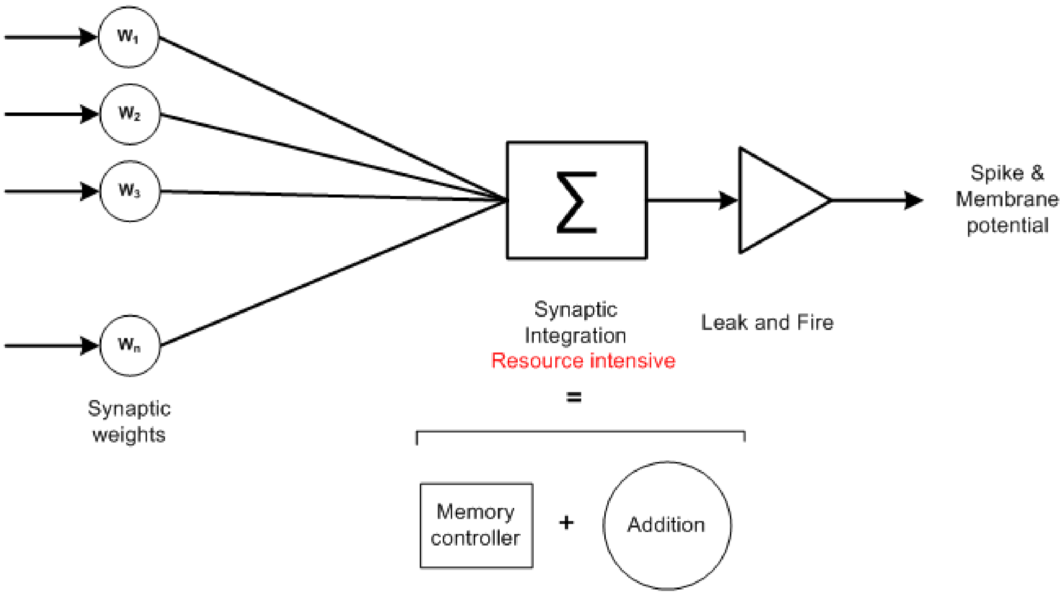


Figure 6.2: Architecture of neuron with its blocks

6.4 Neuron core memory architecture

6.4.1 Requirement analysis for LIF neuron core

The neuron core has a number of values to be stored for its proper functioning and to enable time multiplexing of neuron core. For this an estimate of the data that has to be stored was made. Initial estimates show that the neuron core would require upto MB of memory for a size of 256x256. This can be analysed in detail here. The memory requirement split up can be given as follows

Table 6.2: Memory requirements for neuron core while mulitplexing

Data variable	Size
n_axons	3 bits
n_dendrites	3 bits
syncon_en	1 bit
syn_weight	8 bits
input spikes	1 bit
output spikes	1 bit
membrane potential	8 bits
total	25 bits

The table 6.2 shown here details the memory requirement for a single neuron. With this as a base, the estimates can be scaled to estimate the requirements for larger neuron cores. When we increase the size of the neuron core to 8x8, the no of synapses get increased to 64, making it 512 bits to store the membrane potential alone. For 8 axons and 8 dendrites the input and output spikes can be stored using 16 bits. The synaptic connections enable becomes 64 bits, one for each synapse. The index to access each axon and dendrite will increase based on the size of the neuron. These memory requirement growth can be visualized as shown in 6.3

Table 6.3: Memory requirements for neuron when scaling to different sizes

Category	8	16	32	64	128	256	512	1024
N_axons	3	4	5	6	7	8	9	10
N_dendrites	3	4	5	6	7	8	9	10
Syn_con_en	64	256	1024	4096	16384	65536	262144	1048576
Syn_weight	512	2048	8192	32768	131072	524288	2097152	8388608
Input spike	8	16	32	64	128	256	512	1024
Output spike	8	16	32	64	128	256	512	1024
Mem.potential	512	2048	8192	32768	131072	524288	2097152	8388608
Total bits	1110	4392	17482	69772	278798	1114640	4457490	17827860
In bytes	138.75	549	2185.25	8721.50	34849.75	139330	557186	2228483
In Kbytes	1.35e-01	5.36e-01	2.13e+00	8.52e+00	3.40e+01	1.36e+02	5.44e+02	2.18e+03

This table for memory estimates, helps in visualizing the expected exponential increase of memory requirements. Though storage of axons, dendrites, spikes and membrane potential are important to function as a network of neuron cores, for a single neuron core they are stored in flip-flops, which could later be in additional memory blocks when the number of neuron core increases. Synaptic weights and synaptic connections enable/disable values, which are essential for the functioning of neuron core are alone a significant memory payload in our design. Organization and accessing of these memories have the potential to reduce power consumption, which is discussed in next section.

6.4.2 Synaptic weights memory

Synaptic memories are the most significant parts of any neural networks. Organizing them in a suitable manner was important since it would have implications in power consumption. Having decided that each synaptic weight will be of 8 bits, and its enable/disable control for 1 bit, the estimation will scale up based on number of synapses as previously shown in table 6.3. Recalling the architecture of PSA8 neuron, it was clear that for each cycle of clock, we would be processing 8 synapses together, which becomes 72 bits of data including enable/disable bits. This calls for a memory cell with 72 bits of data width, whereas the depth of the required memory cell depends on the number of neurons/synapses in the design.

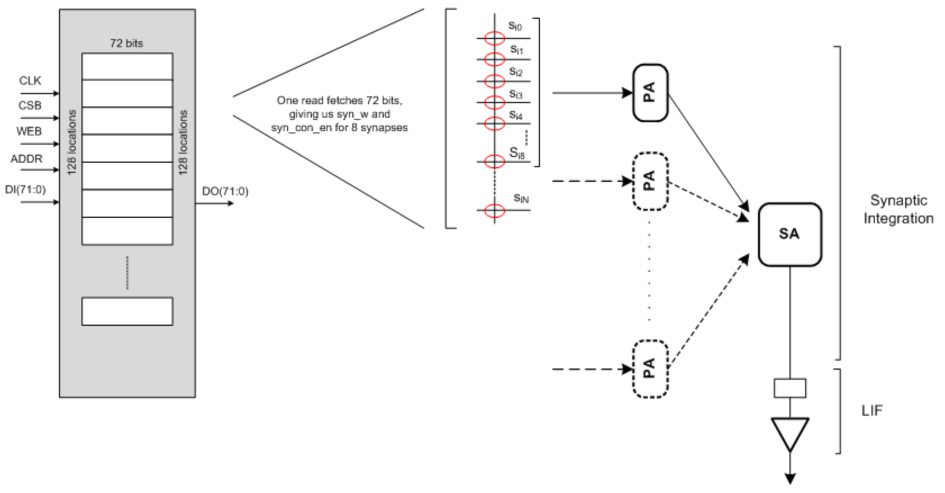


Figure 6.3: Fetching synaptic weights for synaptic integration

For this design, we have used a high density SRAM block with 512 cells of 144 width each, which enables us to store synaptic weights and their enables for two neurons together. Using this memory block, the memory needs of two neurons upto 4096 synapses can be catered. This memory block is replicated as we increase the number of neurons in our design, which is depicted in figure 6.4.

The above usage of memories helps us in reducing the number of clocks connected

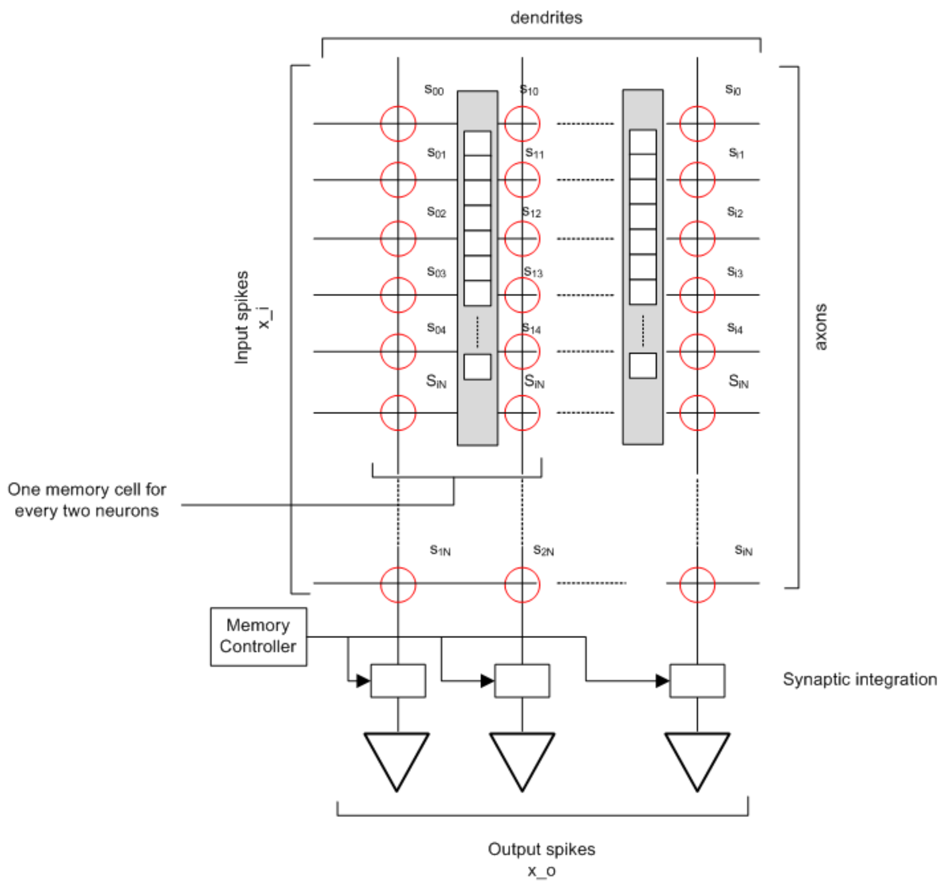


Figure 6.4: Synaptic weights memory organization in crossbar

to memories by half, with respect to the number of neurons. This can be reduced further with memories of larger width, but limited to its availability for this work. These weights will be loaded into the memory before the operation of neuron core with input spikes.

6.4.3 Enable-index memory

Apart from the memory blocks discussed above, we also have a smaller memory block which acts as an index to the weights memory. Basically, it is known that PSA8 architecture consumes weights in sets of 8 inputs which means that it uses 8 enable bits to indicate an active or inactive synapse. The 8 bits will all be accessed in the same clock cycle. With the idea of reducing the need to access memory often, an synapse index memory is placed. This index memory uses 1 bit to represent the 8 enable bits in weights memory, which is obtained by ORing the enable bits. This operation is performed as weights are written into weights memory during initialization, which is shown in figure 6.5. These index memories has the same address locations of access as that of weights memory, which enables the existing weights controller to be used for the same.

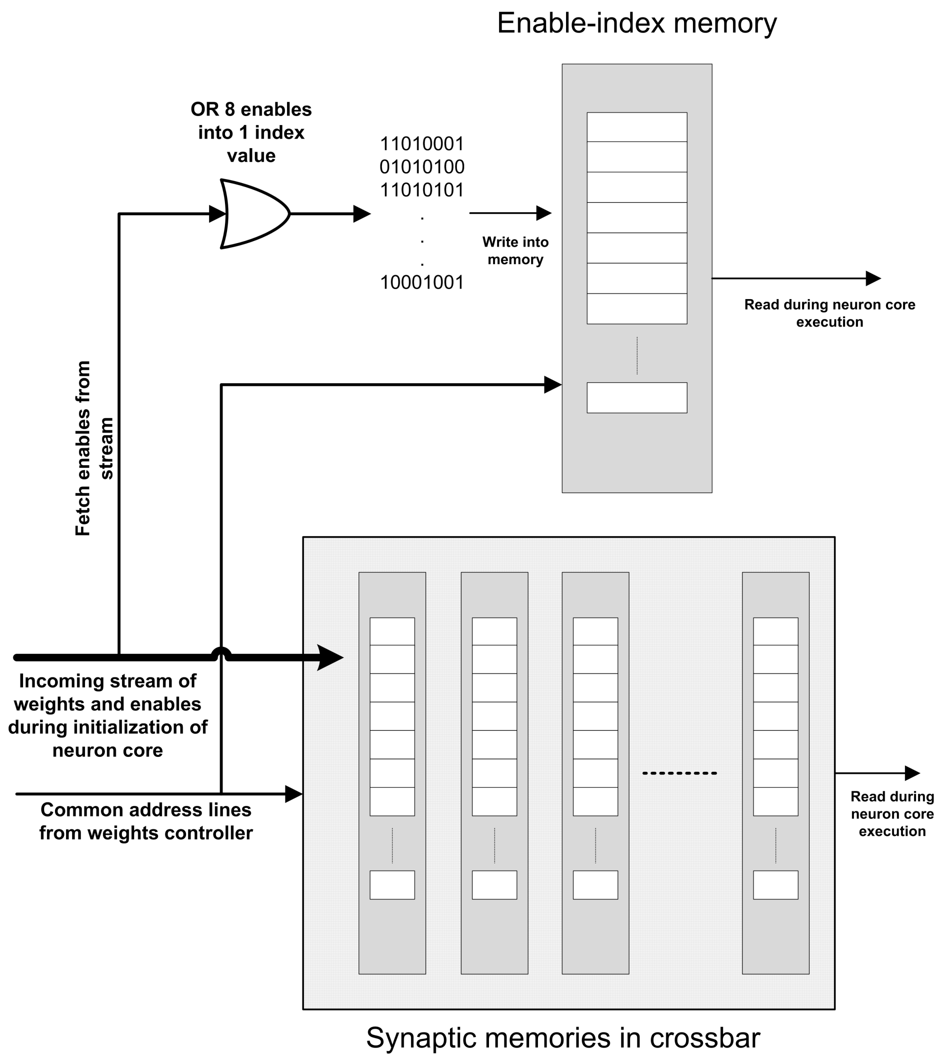

Enable-index memory
OR 8 enables into 1 index value
11010001
01010100
11010101
.
.
.
10001001
Write into memory
Read during neuron core execution
Fetch enables from stream
Incoming stream of weights and enables during initialization of neuron core
Common address lines from weights controller
Read during neuron core execution
Synaptic memories in crossbar

Figure 6.5: Storing enable-index bits in memory

During the operation of neuron core, the index memory is read to decide if a read can be performed for a set of 8 weights. 1 bit of this value represents 8 synapses for a single neuron, making it 8 bits to represent 8 neurons. Hence, a high-density SRAM with 256 8-bit cells has been used. This helps in storing enable-index values for upto 2048 synapses. A memory with higher depth can be selected, but was restricted due to its availability during this work. While a single memory block can support 8 neurons, the memory block is instantiated as required, when the number of neurons are increased.

This facility to check for enable-index for synapses helps us in reducing the dynamic power usage in our neuron core, which will be discussed in a following section in detail.

6.5 Clock gating in LIF neuron

Clock gating is a proven technique for reducing dynamic power in sequential circuits. It reduces dynamic power by stopping clock activity in non-functional areas, as dictated by design, and it has exhibited reasonable reduction in dynamic power [28].

The design of neuron being a fully synchronous one, unlike TrueNorth which is semi-synchronous logic design, is bound to have a high dynamic power consumption. To mitigate this, clock gating techniques have been employed at strategic spots inside the design of this neuron core. Here, the clock signals to major synchronous blocks have been connected with a latch based clock gate as shown in figure 6.6. This helps us in disabling the clock distribution when a particular segment of logic is not currently active. This has been done in a very aggressive and fine-grained fashion, as detailed in the following paragraphs.

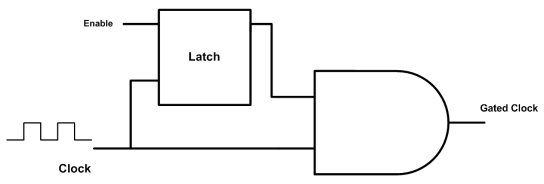


Figure 6.6: Latch based clock gating

Clock gated design segments

To achieve low dynamic power, most active segments inside the neuron core design have been clock gated. Hierarchically, three main segments under the neuron core are

- Memories holding synaptic weights
- Memories holding enable-index
- Weights controller
- Neurons

All of these segments have their own gated clocks that get enabled when they have to be operated. This gating architecture is shown in figure 6.7, and is also discussed below in detail.

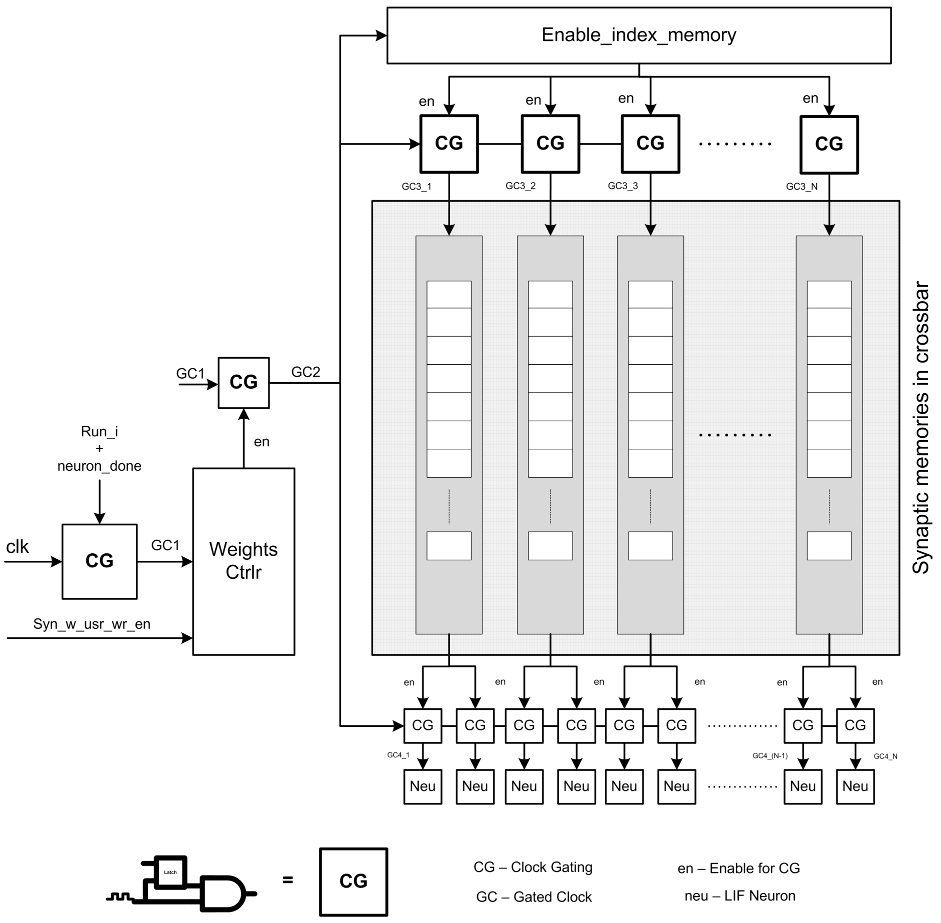


Figure 6.7: Clock gating architecture in neuron core

The synaptic memories are the most active areas of the design, which consistently have data being read and written into it, whenever the neurons are being operated. For the clock leading to the memory cells, they have been gated with enables for neuron core and weights controller. Since a data location is accessed only based on an address generated by memory controller, in every other case, the clock connected to the memory cells are kept inactive. Apart from this, the clock is active when a user edits memory locations too, which is during the initialization phase.

As for the weights controller, it can be said that this is the master of neuron core. Every access inside neuron core is controlled by this controller and the controller itself runs based on a "run" signal from outside of the core. The run signal is user controlled, in a way that we can give a "go" once synaptic weights are updated as per our needs in those memories. The clock flow would also stop after one time division of neuron core operations are complete.

For the neurons with PSA-8B architecture we use here, they operate on 8 synapses at a moment using their parallel adders. For each of these 8 synapses and every other synapse, there are individual enable signals. The weights in the synapses are active and accessible to incoming spikes only if it is an active synaptic connection. If inactive, there is no meaning in adding them up. Hence, the synaptic connection enable signals have been taken into consideration for gating clocks to a neuron. Only if atleast one of the 8 synapses in consideration are active at the moment, clock is active. If not, the clock is gated and operations don't happen until the next active synapse.

The enable-index memory discussed under section 6.4 plays a major part in gating of clocks to weights memory. This memory is gated with the same enable as weights controller. With the address from the controller, the enable-index bits are read, which act as enables for the clocks of weight memories. It wasn't reasonable to access the weights memory everytime even when a synapse is not active. The enable-index value being a representative of 8 synapses for a neuron, activates the clocking to weights memory when it is '1'. The weights memory being a larger memory has the potential to consume larger dynamic power as compared to the enable-index memory which is smaller in area and capacity.

With these, the active clocked segments of our neuron core are gated and controlled. This helps us in reducing dynamic power to a significant amount which would be discussed in next section.

6.6 Results and Observations

After the design of neuron core, it was important to know how it performs. Performance of such a design can be gauged using parameters such as area, latency, and power consumption. There are also other parameters like accuracy of results of computations, ease of mapping of applications, etc which are specific to the kind of network mapped on the SNN core. This being a generic neuron core, those parameters are not considered at this level of development.

Standard methods have been followed to take stats about area, and power usage from the design. For latency, its more of an architecture dependent one, which has been discussed in our design space exploration in section 5.5.

Calculation of Area

The completed design has been synthesized using Synopsys Design Compiler [29] with umcL65 [30] technology. Memories being an important part of our design, currently available memory cells from the same technology has been used for this work. The design has been synthesized for 400MHz operation. While synthesis, clock gating option was used. After synthesis, resulting netlist was used for power estimation and reports about area and timing were collected. The same process is again repeated for Placement and Routing (P&R) the design, where parasitics are also taken into account. The flow has been pictorially represented for synthesis and P&R in figures 6.8 and 6.9, respectively.

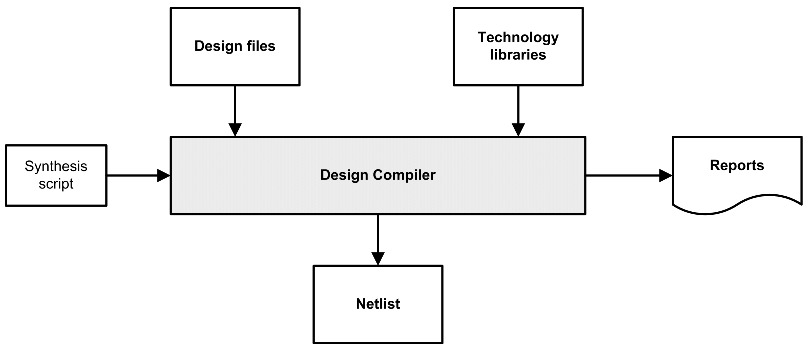


Figure 6.8: Synthesis flow using Design Compiler

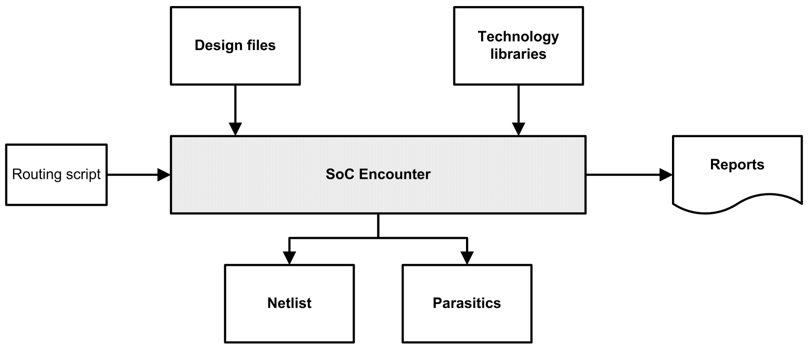


Figure 6.9: Placement and Routing flow using SoC Encounter

Calculation of Power

For any CMOS design, total power consumed are categorized into two major types [32].

- Static Power
- Dynamic Power

Static Power, is the power consumed when transistors are not switching. This is dissipated in multiple ways. A major percentage of static power results from source-to-drain sub-threshold leakage current, which is due to reduced threshold voltages which prevent the gate from turning off and allow leakage current (I_leak). Static power could also be due to reverse-biased diode leakage from the diffusion layers and the substrate.

Being dependent of voltage, temperature and state of transistors, the leakage power can be presented by the equation

$$LeakagePower = V * I_{leak} \tag{6.1}$$

Dynamic Power, is the power amounted by transistors due to switching of logic states. A voltage change charges or discharges the capacitive load of an external net. This voltage change also results in a short-circuit between the N and P transistors inside the gate. For digital designs, the dynamic power is a combination of switching power, caused by charging-discharging of external net and internal power, which is dissipated within the cell.

The two components of dynamic power are represented using the following equations

$$\begin{aligned} SwitchingPower &= \frac{1}{2} * C_{load} * V^2 * f \\ InternalPower &= (\frac{1}{2} * C_{int} * V^2 * f) + (V * I_{sc}) \end{aligned}$$

where C_{load} is load capacitance, C_{int} is internal capacitance, f is frequency of operation and I_{sc} is switching current.

Power Analysis Requirements

For power estimation, Synopsys PrimeTime [31] has been used. In order to perform power estimation, the following details were required and used.

- Netlist Data: This is the design netlist which is a result of synthesis. It is used to determine the design connectivity and type of cells used in design. It helps in accurately computing the capacitance on drivers.
- Cell Library: This is the technology library provided by ASIC libarary vendors. It has details about both static and dynamic power consumption internal to the cell.
- Signal Activity: This contains details about switching of signals in our design, which gives information about toggle rate directly contributing to dynamic power. This is used in the form of VCD dumps from our simulation tools.
- Net Parasitics: Parasitics are information about capcitance in our design taking into account the wires and fanouts. This is used when performing post-route power analysis and skipped when performing post-synthesis power analysis.

The Power analysis flow has been shown pictorially in figure 6.10

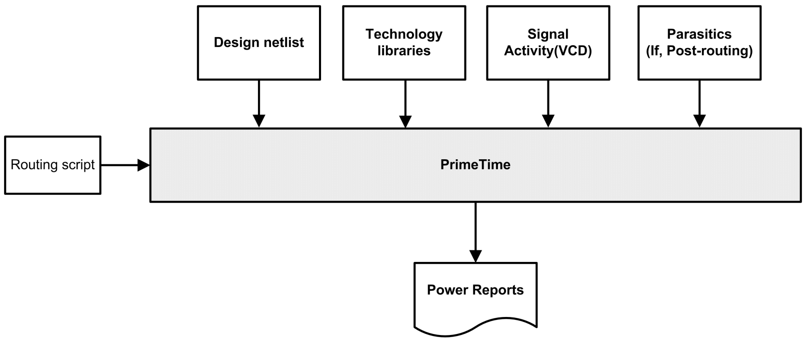


Figure 6.10: Power analysis flow using PrimeTime

6.6.1 Area and Latency

The LIF neuron with PSA8 architecture has been synthesized at 400 MHz frequency using umc65 technology. Latch based clock gating has been enabled while synthesis. Designs containing 8 to 256 neurons have been synthesized and their area results are as shown in table 6.4.

Table 6.4: Synthesis results for neuron core @ 400MHz with different number of neurons

Category/#Neurons	**8**	**16**	**32**	**64**	**128**	**256**
Combinational Area	10581.12	21064.32	42093.36	84193.92	168207.48	335894.41
Non-combinational Area	6354.36	12578.40	25062.84	50019.84	99897.12	199668.24
Total Buffer Area	74.16	78.48	155.52	621.00	585.72	607.68
Total Inverter Area	615.96	1257.48	2518.20	4781.88	10081.08	20300.40
Buf/Inv Area	690.12	1335.96	2673.72	5402.88	10666.80	20908.08
Macro/Black Box Area	246822.84	493645.69	987291.38	1974582.75	3949165.50	7898331.00
Total Area	263758.32	527288.41	1054447.58	2108796.51	4217270.10	8433893.65

From these results we can see that the total area increases as we increase the number of neurons, and agree to it which is as expected. The Blackbox area in the table is the area occupied by high-density SRAM cells. A plot of area occupied by different design blocks is shown in figure 6.11, which clearly shows that memory is the dominant part of neuron core of all sizes. We expect this to reflect in power analysis results also.

It has to be noted that the memory cells used here are larger than what is required for smaller neuron sizes. A larger size has been used to accommodate parameterized design later helping in scaling to larger number of neurons. So if we develop memory cells of exact size based on requirements, the total area occupied would be smaller than what is shown in this work.

PSA8 architecture was exhibiting a good balance between area and latency in our plots earlier, as shown in figure 5.12. This design can process 8 synapses in 2 clock cycles, which is only a bit higher than a fully parallel implementation. When neuron

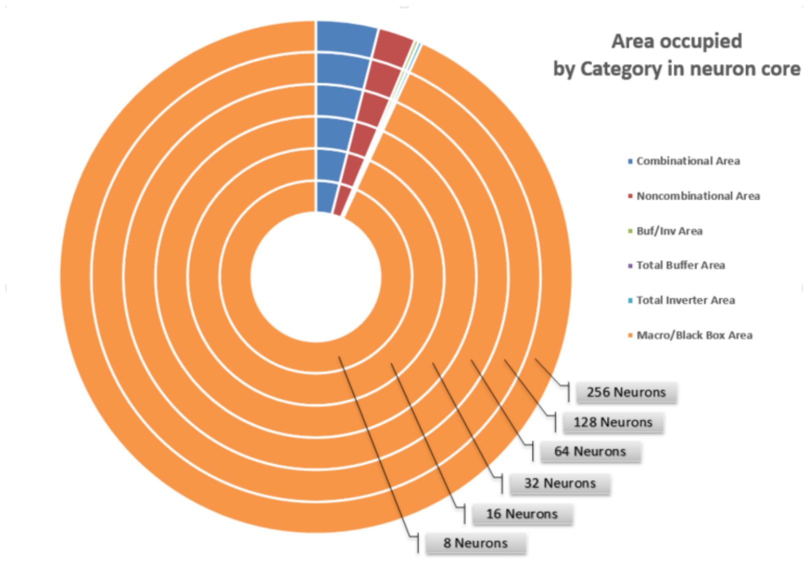


Figure 6.11: Area occupied by different design blocks

size increases, the latency increases too, which can be represented using equation 6.2.

$$Latency_{PSA} = \frac{N_s}{B} * t_{clkperiod} \tag{6.2}$$

where N_s is the number of synapses in a neuron, B is the Block size of parallel adder (which is 8 in this case), and $t_{clkperiod}$ is the period of clock used to run the design.

6.6.2 Power Estimation

For this work, the objective all along has been to design a neuron core with low power consumption. After a series of exploratory designs, PSA8 neuron architecture was selected and synthesized it for different sizes of neurons ranging from 8 to 256. After synthesis, the netlist has been used to generate switching activity file. In this case, a value change dump (VCD) file was generated using modelsim. This was then used with PrimeTime to estimate power usage, the flow of which was already shown in figure 6.10.

The power estimation has been performed considering two extreme cases of operation.

- **Fully active synapses**, the case where all synapses in the neuron core are active and contribute to spiking of neurons.

- **Fully inactive synapses**, the case where all synapses in the neuron core are inactive, thereby not contributing to spiking of neurons.

With the these two cases, random inputs and weights initialized from testbench, the neuron core has been simulated for a period of 12us at a frequency of 50 MHz (though design is synthesized for 400 MHz). Also, the operating voltage for the transistors have been set to 1.2V. Results of this power estimation for both of these cases have been shown in tables 6.5 and 6.6.

Table 6.5: Power estimation results for neuron core with fully active synapses

	Neuron core operating point = 100 MHz @ 1.2V, 8-bit Synapses						
	Power Estimation	**Post-Synthesis**					
Fully Active Synapses	**Power Category**	**8**	**16**	**32**	**64**	**128**	**256**
	Cell Dynamic Power	1.26E-03	3.20E-03	9.17E-03	2.94E-02	1.03E-01	1.98E-01
	Cell Leakage Power	1.88E-05	3.74E-05	7.46E-05	1.49E-04	2.96E-04	5.94E-04
	Total Power	1.28E-03	3.24E-03	9.24E-03	2.95E-02	1.03E-01	1.98E-01
	Other Stats						
	Memory Blocks Power	9.70E-04	2.58E-03	7.73E-03	2.55E-02	9.12E-02	1.74E-01
	Logic Blocks Power	3.05E-04	6.53E-04	1.51E-03	4.00E-03	1.17E-02	2.44E-02
	% power by memory blocks	76.10	79.83	83.67	86.44	88.63	87.70

From the table 6.5 we can see that, the cell dynamic power is dominating the total power estimate for all number of neurons. This is an indication that the neuron core is at its full operation level, without skipping any clocks due to our clock gating architecture. Since an optimized architecture has been selected to construct the neuron core, it can be said that the leakage power is quite under control. Looking into more details from the reports, it was found that the power consumption is mostly due to memory blocks, which are large in number and area. This is reflected in the table, showing that memory blocks contribute to from 76 to 88 percentage of total power of neuron core. This is a proof of the significant role of memories in a neural network design.

Table 6.6: Power estimation results for neuron core with fully inactive synapses

	Neuron core operating point = 100 MHz @ 1.2V, 8-bit Synapses						
	Power Estimation	**Post-Synthesis**					
Fully Inactive Synapses	**Power Category**	**8**	**16**	**32**	**64**	**128**	**256**
	Cell Dynamic Power	5.11E-05	1.24E-04	3.53E-04	1.13E-03	3.97E-03	7.53E-03
	Cell Leakage Power	1.81E-05	3.61E-05	7.20E-05	1.44E-04	2.86E-04	5.73E-04
	Total Power	6.92E-05	1.60E-04	4.25E-04	1.28E-03	4.26E-03	8.10E-03
	Other Stats						
	Memory Blocks Power	1.09E-05	2.19E-05	4.37E-05	8.74E-05	1.75E-04	3.50E-04
	Logic Blocks Power	5.83E-05	1.38E-04	3.82E-04	1.19E-03	4.08E-03	7.75E-03
	% power by memory blocks	15.79	13.67	10.28	6.85	4.11	4.32

In case of fully inactive synapses, from the table 6.6, the total power consumption is considerably lower. This is attributed to the low switching activity inside neuron core due to the clock gating architecture. Practically, the clock is active most of the times only on enable-index memory and weights controller. Since the synapses were not active, the clock gets gated to every other block of neuron core, which includes weights memory, synaptic integration and LIF spiking stage. This results in the low power estimates that are seen in the table 6.6. Even under this case, the enable-index memory which is active all the time is contributing a good share to total power consumption, ranging from 4 to 16 percentage of total.

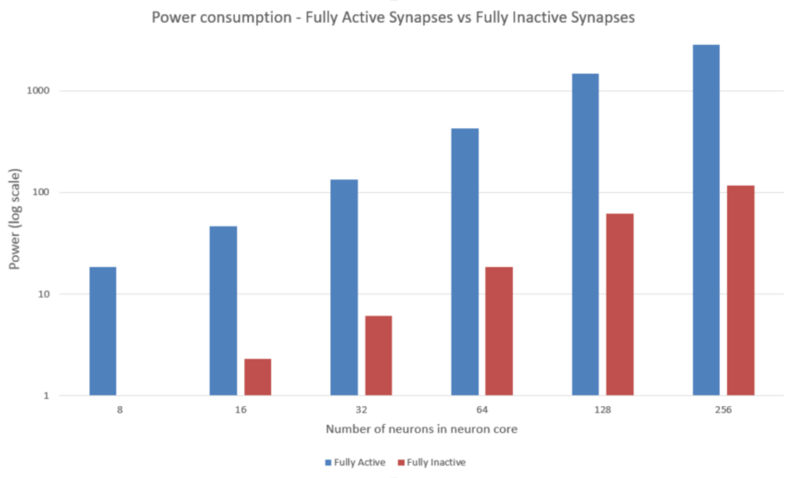


Figure 6.12: Comparison between power estimates for fully active and fully inactive synapses

The figure 6.12 is showing us the difference in power consumption between the two cases - fully active and fully inactive synapses in the neuron core. In a realtime operation of such neuron core, all synapses might not be active or inactive all-together, but would be random based on the application mapped upon it by users. It could be said that it is definitely going to be between the extreme cases considered in this work. The major take away from this is that, when a realtime application is mapped, the operational power consumption would be somewhere between the estimates of these cases.

The significance of memories in neuron core design is shown explicitly in our results. The figure 6.13 shows the contribution of memory blocks in terms of both area and power with respect to the total area and power. The high percentage of its contribution is a direct proof that design of efficient memory technologies and memory usage architectures have to be taken seriously.

With this, the number of spikes from neurons have been determined during collecting data for switching activity file. This helps us in understanding how much energy is spent on every spike in this neuron core, which could be compared to other contemporary designs of the same nature. The results for this are summarized in

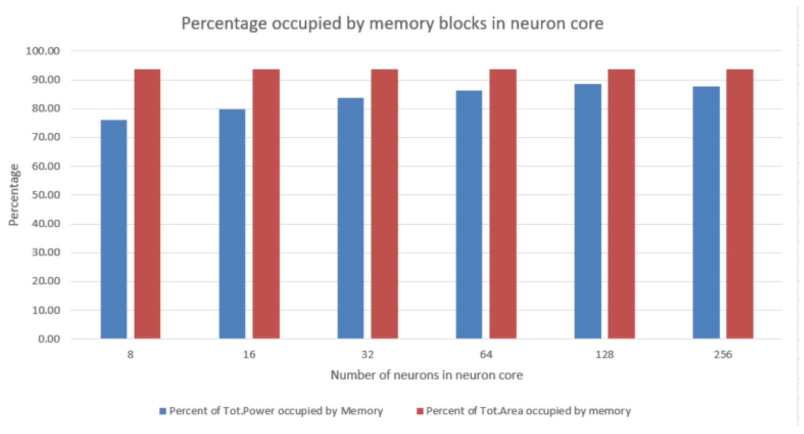


Figure 6.13: Memory block contribution to area and power

Table 6.7: Energy spent per spike for LIF neuron core

Post-synthesis - Neuron core with fully active synapses						
#Neurons	**8**	**16**	**32**	**64**	**128**	**256**
Joules/Spike	6.59483E-11	8.37155E-11	1.19534E-10	1.9073E-10	3.4453E-10	3.3515E-10

table 6.7.

Table 6.8: Post-routing power estimates for neuron core with 256 synapses

Post-routing, Operating point = 100 MHz @ 1.2V, 8-bit Synaptic weights		
Power/Design Category	**Full-Active Synapses**	**Fully-Inactive Synapses**
#Neurons	**256**	
Cell Dynamic Power	1.99E-01	7.56E-03
Cell Leakage Power	5.92E-04	5.70E-04
Total Power	2.00E-01	8.13E-03
Other Stats		
Memory Blocks Power	1.74E-01	3.50E-04
Logic Blocks Power	2.56E-02	7.78E-03
% power occupied by memory blocks	87.17	4.30
Joules/Spike	3.3718E-10	N/A

Besides, estimating power post-synthesis, effort was also taken to floor plan and route the design using SoC Enocunter. Automatic routing was employed with an utilization ratio of 0.9 to use the available space to its fullest. As a result of this, parasitic information and a routed netlist was received, helping in estimating a bit more accurate power usage results. Taking into account the large amount of time it takes to floor plan and route a design, the design with 256 neurons alone have been routed and used for power estimation. The results of post-routing power estimation are

Table 6.9: Comparison of TrueNorth and LIF Neuron Core

Comparison with State-of-the-art		
Categories	**TrueNorth Chip**	**LIF Neuron Core**
System	ASIC	ASIC
Design	Fully digital	Fully digital
Timing	Mixed Async/Sync	Fully Synchronous
No. of Cores	4096	1
Neurons	1 million	256 (Scalable)
Axons	1 million	256 (Scalable)
Synapses	256 million	65536
Neuron organization	16 x 16 array per core	1 x 256 array
Crossbar	256 x 256 SRAM array per core	256 x 256 SRAM array
Configurable Parameters	Synaptic connections, Axon type, Synapse values, Leak, Threshold	Synaptic connections, Number of axons & neurons, Synapse values, Leak, Threshold
Transistor Technology	Samsung's 28nm	UMC 65nm
Core area	$4.3cm^2$	$73.02mm^2$
Operating frequency	1 KHz (each time step)	100 MHz
Operating voltage	0.85 V	1.2 V
Active Power	65mW	0.19mW

shown in table 6.8, which are close to the estimates post-synthesis. The take aways from this results are the same as what is discussed with post-synthesis results.

6.6.3 Comparison with TrueNorth

Having estimated power usage of neuron core, it is useful to compare it with any of the state-of-the-art designs of a similar nature, which gives us more insights into design of neuromorphic hardwares. LIF neuron core is compared here with its inspiration TrueNorth in this section.

The comparison in table 6.9 is not an 1:1 comparison. The data about TrueNorth is for an entire chip, whereas in case of LIF neuron core, it is only for a single neuron core. There was also the development data [17] of TrueNorth, which was for a different dimension and an older technology. It is ignored here for the fact that it is not state-of-the-art anymore.

From [12], [16], we can understand that the neurosynaptic core in TrueNorth chip is also filled with other design blocks such as a Scheduler and Router. The chip is much larger with 4096 neurosynaptic cores, fabricated using 28nm and operated at 1KHz frequency for each time step. Since the neuron core in this work is not directly comparable with TrueNorth, to assess the range of results, the power usage of a single LIF neuron core multiplied with 4096 results in about 812.23 mW. This result is way higher than TrueNorth, but it has to be noted that LIF neuron core operates at 100 MHz at 1.2V using 65nm technology.

In a practical use case, it not necessary for the neuron core to be active and operate all the time. It could just wake up - like TrueNorth - at a 1ms interval of time. With the fact that LIF PSA8 neuron core could take upto 35 clock cycles in a 256 neuron core, at 100 MHz its about 35us. In a window of 1ms, this means that the design shows 3.5 to 4% activity. For 256 neuron core, we have an estimated 812.23mW power consumption at full activity. Therefore, for a lower bound approximate comparison with 5% activity, it could consume about 40mW average power which is definitely in TrueNorth range. To make this happen, an advanced clocking scheme has to be employed along with other specific solutions to reduce leakage power.

Also, as this design becomes more mature and gets developed further, a regular operation wouldn't always engage all the neuron core when stacked as an array. With this, it is expected that the dynamic power could definitely be lowered to a certain extent.

On the other hand, if a LIF neuron core with 16 neurons is taken into account, it uses about 3.24 uW of power, which is considerably lesser. Stacking of this neuron core as a 16 x 16 array to form 256 neurons, and eventually a bigger chip could be an interesting path to test and evaluate. With this method, it would also give designers a deeper way to control active power by clock gating, similar to what is already done in this work.

7

Chapter 7

Conclusion

In this work, we explored the idea of constructing a digital spiking neuron core, that can operate with low power consumption. A simple Leaky, integrate and fire (LIF) neuron was selected to be the neuron of choice, and its design was explored in detail. With the idea of finding the most suitable architecture for implementing this neuron, a design space exploration was carried out.

During this process, fully serial, fully parallel, serial parallel and parallel-serial adder architectures were explored. Also adder architectures related to bitwise operations, like bitwise parallel adder and bitwise serial adder architectures were also touched upon. After simulation, synthesis and analysis of this wide range of choices, parallel-serial architecture was selected for this work, for having a good balance between low area and better latency.

Further to this, a neuron core consisting of a crossbar, weights controller, synaptic weights memories and enable-index memories, was constructed. The objective of this work being designing a low power digital spiking neuron core, a lot of effort was put towards keeping the area low, which contributes towards leakage power of any Digital ASIC design. For this reason, the architecture of memories have been optimized, limited to a list of available standard memory cells during the time of development of this work.

Apart from this a significant amount of work has been done in the form of clocking gating the entire neuron core, to mitigate dynamic power usage during the operation of neuron core. The architecture of the neuron core itself has been designed to aid in efficient clock gating. Every segment of the neuron core receives an active clock only if the section has a need to be operated, and gated if not so. The design was simulated and synthesized for a range of neuron sizes.

To evaluate the effectiveness of design and clock gating strategies, power analysis was done for the neuron core at 100 MHz using 1.2 V operating voltage. The results have been summarized giving us a clear picture of where a fully synchronous design of neuron core can take us in terms of power efficiency. Also, this work shows the significance of memories in neuromorphic system design, which is shown to occupy a major share of power and area. This is a clear indication of where more effort has to be put to achieve smarter and efficient neuromorphic hardwares.

Overall, this work has presented a design space exploration for neuron design in digital hardware and has contributed architecturally to efficient design in terms of smaller area and lower power consumption, which could lead to further optimizations and improvements for neuromorphic hardware designs.

Chapter 8
Future Work

This work has covered preliminary estimates in terms of area and power for a simple neuromorphic hardware. But there is lots of room for improvement in this design. Neuromorphic hardwares, themselves are still in their early stages of development. Here, a list of possible areas of future work has been presented, which are related to this work and any neuromorphic hardware design.

Efficient memory architecture design
The memories in a neuron core could use some improvements with novel ways to access the data and reduce the number of transactions to read a set of data. Coming up with new architectures to use them for a neural network is a good place to focus future efforts. This could help us reduce total area and also dynamic power.

Smaller synaptic weights
Another possible area to explore could be reducing the size of synaptic weights from 8-bits. Choosing lesser bits would result in smaller range for representing weights, but this could also make convergence easier while training a network and also save large amount of memory and power while converting it to a hardware network.

Approximate computing
This is less explored type of computing where we could truncate the weights to their most significant bits and replacing rest of bits with random numbers. The values during computations wouldn't be accurate, but for a spiking network it could work out like discussed in ApproxANN [33]. If this works well, it could greatly reduce the memory requirements, showing direct improvements in area and power.

Low power and low area memory cell design
Effort can be made to bring improvements at transistor level to design standard cells that consume very less power and area. This will help in advancing neuromorphic hardware design with existing design tools and technologies.

Memristors

Memristors are one of the most anticipated technology since the advancements made with transistors that we use today. They could change the neuromorphic computing scenario completely, since they have the potential to store and access memory without spending much power on it. There are research teams working on this and this is a hopeful area.

Event based, more serial approach

Creating a design architecture that is total even-based and serial in operation could also be helpful. The latency of operations could be longer, which can always be mitigated using higher frequencies or parallelism to complete more work. This a less attractive path since, we could again hit our usual design boundaries in terms of area and power.

Construction of a larger network with single neuron core

There can also further work done in mapping single core to a multi-core neural network processing cluster, which would help in mapping realtime applications and assessing performance of a SNN. For this, one can either construct a cluster with larger number of cores and better power management or construct a limited size cluster with time multiplexed operation, which could save area and eventually power.

Bibliography

[1] Misra, Janardan, and Indranil Saha. ‘Artificial Neural Networks in Hardware: A Survey of Two Decades of Progress’. Neurocomputing, Artificial Brains, 74, no. 1–3 (December 2010): 239–55. doi:10.1016/j.neucom.2010.03.021.

[2] Ahmed, M. R., and B. K. Sujatha. ‘A Review on Methods, Issues and Challenges in Neuromorphic Engineering’. In 2015 International Conference on Communications and Signal Processing (ICCSP), 0899–0903, 2015. doi:10.1109/ICCSP.2015.7322626.

[3] A. L. Hodgkin and A. F. Huxley, ‘A quantitative description of membrane current and its application to conduction and excitation in nerve’, J Physiol, vol. 117, no. 4, pp. 500–544, Aug. 1952.

[4] E. M. Izhikevich, ‘Simple Model of Spiking Neurons’. [Online]. Available: http://www.izhikevich.org/publications/spikes.htm. [Accessed: 05-Sep-2016].

[5] Z. Wang, L. Guo, and M. Adjouadi, ‘A biological plausible Generalized Leaky Integrate-and-Fire neuron model’, in 2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2014, pp. 6810–6813.

[6] Painkras, E., L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber. ‘SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation’. IEEE Journal of Solid-State Circuits 48, no. 8 (August 2013): 1943–53. doi:10.1109/JSSC.2013.2259038.

[7] Furber, S. B., F. Galluppi, S. Temple, and L. A. Plana. ‘The SpiNNaker Project’. Proceedings of the IEEE 102, no. 5 (May 2014): 652–65. doi:10.1109/JPROC.2014.2304638.

[8] Furber, S. B., D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown. ‘Overview of the SpiNNaker System Architecture’. IEEE Transactions on Computers 62, no. 12 (December 2013): 2454–67. doi:10.1109/TC.2012.142.

[9] Benjamin, B. V., P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen. ‘Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations’. Proceedings of the IEEE 102, no. 5 (May 2014): 699–716. doi:10.1109/JPROC.2014.2313565.

[10] Neil, D., and S. C. Liu. 'Minitaur, an Event-Driven FPGA-Based Spiking Network Accelerator'. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 22, no. 12 (December 2014): 2621–28. doi:10.1109/TVLSI.2013.2294916.

[11] Qiao, Ning, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. 'A Reconfigurable on-Line Learning Spiking Neuromorphic Processor Comprising 256 Neurons and 128K Synapses'. Neuromorphic Engineering 9 (2015): 141. doi:10.3389/fnins.2015.00141.

[12] Cassidy, A. S., P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, et al. 'Cognitive Computing Building Block: A Versatile and Efficient Digital Neuron Model for Neurosynaptic Cores'. In The 2013 International Joint Conference on Neural Networks (IJCNN), 1–10, 2013. doi:10.1109/IJCNN.2013.6707077.

[13] Preissl, R., T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha. 'Compass: A Scalable Simulator for an Architecture for Cognitive Computing'. In High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for, 1–11, 2012. doi:10.1109/SC.2012.34.

[14] Esser, S. K., A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, et al. 'Cognitive Computing Systems: Algorithms and Applications for Networks of Neurosynaptic Cores'. In The 2013 International Joint Conference on Neural Networks (IJCNN), 1–10, 2013. doi:10.1109/IJCNN.2013.6706746.

[15] Amir, A., P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, et al. 'Cognitive Computing Programming Paradigm: A Corelet Language for Composing Networks of Neurosynaptic Cores'. In The 2013 International Joint Conference on Neural Networks (IJCNN), 1–10, 2013. doi:10.1109/IJCNN.2013.6707078.

[16] Akopyan, F., J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, et al. 'TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip'. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34, no. 10 (October 2015): 1537–57. doi:10.1109/TCAD.2015.2474396.

[17] Merolla, P., J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha. 'A Digital Neurosynaptic Core Using Embedded Crossbar Memory with 45pJ per Spike in 45nm'. In 2011 IEEE Custom Integrated Circuits Conference (CICC), 1–4, 2011. doi:10.1109/CICC.2011.6055294.

[18] Hagan, Martin T. ; Demuth, Howard B. ; Beale, Mark: Neural Network Design. Boston : PWS Publishing Company, 1996

[19] 'Neurons & Synapses - Memory & the Brain - The Human Memory'. [Online].

Available: http://www.human-memory.net/brain_neurons.html. [Accessed: 06-Sep-2016].

[20] W.Maass., "Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons", Advances in Neural Information Processing Systems, vol. 9, 1997.

[21] W.Maass., "Networks of spiking neurons: the third generation of neural network models", Neural Networks, vol.10, pp.1659-1671, August 1997.

[22] X. Jin, S. B. Furber, and J. V. Woods, 'Efficient modelling of spiking neural networks on a scalable chip multiprocessor', in 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 2008, pp. 2812–2819.

[23] Pedroni, B. U., S. Das, J. V. Arthur, P. A. Merolla, B. L. Jackson, D. S. Modha, K. Kreutz-Delgado, and G. Cauwenberghs. 'Mapping Generative Models onto a Network of Digital Spiking Neurons'. IEEE Transactions on Biomedical Circuits and Systems PP, no. 99 (2016): 1–18. doi:10.1109/TBCAS.2016.2539352.

[24] Schrauwen, B., M. D'Haene, D. Verstraeten, and J. Van Campenhout. 'Compact Hardware for Real-Time Speech Recognition Using a Liquid State Machine'. In 2007 International Joint Conference on Neural Networks, 1097–1102, 2007. doi:10.1109/IJCNN.2007.4371111.

[25] Zjajo, A., N. Mehta, and R. van Leuken. 'A 31 pJ/spike Hybrid Stochastic Neuromorphic Signal Processor'. In 2015 IEEE Signal Processing in Medicine and Biology Symposium (SPMB), 1–2, 2015. doi:10.1109/SPMB.2015.7405458.

[26] Indiveri, G., and S. C. Liu. 'Memory and Information Processing in Neuromorphic Systems'. Proceedings of the IEEE 103, no. 8 (August 2015): 1379–97. doi:10.1109/JPROC.2015.2444094.

[27] Seo, J. s, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, et al. 'A 45nm CMOS Neuromorphic Chip with a Scalable Architecture for Learning in Networks of Spiking Neurons'. In 2011 IEEE Custom Integrated Circuits Conference (CICC), 1–4, 2011. doi:10.1109/CICC.2011.6055293.

[28] G. S. R. Srivatsava, P. Singh, S. Gaggar, and S. K. Vishvakarma, 'Dynamic power reduction through clock gating technique for low power memory applications', in 2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), 2015, pp. 1–6.

[29] 'Design Compiler', Synopsys. [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler /Pages/default.aspx. [Accessed: 28-Aug-2016].

[30] 'UMC Free Library - 90 nm IPs'. [Online]. Available: http://freelibrary.faraday-tech.com/ips/65library.html. [Accessed: 28-Aug-2016].

[31] 'PrimeTime Static Timing Analysis', Synopsys. [Online]. Available: http://www.synopsys.com/Tools/Implementation/SignOff/PrimeTime/Pages /default.aspx. [Accessed: 28-Aug-2016].

[32] Gordon Yip, Product Marketing Manager, Synopsys, Inc., 'Expanding the Synopsys PrimeTime® Solution with Power Analysis', Jun-2006. [Online]. Available: https://www.synopsys.com/Tools/Implementation/SignOff/CapsuleModule /ptpx_wp.pdf. [Accessed: 28-Aug-2016].

[33] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, 'ApproxANN: An approximate computing framework for artificial neural network', in 2015 Design, Automation Test in Europe Conference Exhibition (DATE), 2015, pp. 701–706.

TRITA TRITA-ICT-EX-2016:191

www.kth.se