SPIKING NEURAL NETWORK ARCHITECTURE DESIGN AND PERFORMANCE EXPLORATION TOWARDS THE DESIGN OF A SCALABLE NEURO-INSPIRED SYSTEM FOR COMPLEX COGNITION APPLICATIONS

A Thesis Presented to the Department of

Computer Science

African University of Science and Technology

In Partial Fulfilment of the Requirements for the Degree of

MASTER of Computer Science

By

OJIUGWO CHUKWUKA N.

Abuja, Nigeria

December, 2017

CERTIFICATION

This is to certify that the thesis titled ‘Spiking Neural Network Architecture Design and Performance Exploration towards the Design of a Scalable Neuro-Inspired System for Complex Cognition Applications’ submitted to the school of postgraduate studies, African University of Science and Technology (AUST), Abuja, Nigeria for the award of the Master's degree is a record of original research carried out by Ojiugwo Chukwuka N. in the Department of Computer Science.

SPIKING NEURAL NETWORK ARCHITECTURE DESIGN AND PERFORMANCE EXPLORATION TOWARDS THE DESIGN OF A SCALABLE NEURO-INSPIRED SYSTEM FOR COMPLEX COGNITION APPLICATIONS

By

OJIUGWO Chukwuka N.

A THESIS APPROVED BY THE COMPUTER SCIENCE DEPARTMENT

RECOMMENDED: _____

Supervisor, Prof. Ben Abdallah

_____

Co-supervisor

_____

Head, Department of Computer Science

APPROVED: _____

Chief Academic Officer

_____

December 9th, 2017

ABSTRACT

Research into artificial neural networks (ANNs) is inspired by how information is dynamically and massively processed by biological neurons. Conventional ANNs research has received a wide range of applications including automation, but there are still problems of timing, power consumption, and massive parallelism. Spiking neural networks (SNNs), being the third-generation of neural networks, has drawn attention from a greater number of researchers due to the timing concept, which defines its closeness to biological Spiking Neural Network (bio-SNN tested) functions. Spike timing plays an important role in every spiking neuron and proves computationally more plausible than other conventional ANNs. The real biological and distinct neuron timing and spike firing can be modelled artificially using neurodynamics and spike neuron models. The spike timing dependent plasticity (STDP) learning rule also incorporates timing concepts and is suitable for training SNNs which describes general plasticity rules that depend on the actual timing of pre- and postsynaptic spikes. This work presents a software implementation of an SNN based on the Leaky Integrate-and-Fire (LIF) neuron model and STDP learning algorithm. Also, we present a novel hardware design and architecture of a lightweight neuro-processing core (NPC) to be implemented in a packet-switched based neuro-inspired system, named NASH. The NASH architecture uses the LIF neuron model and reduced flit format size that solves the problems of timing and high-power consumption. Software evaluation shows that our network tested 94% accuracy with MNIST datasets of handwritten digits.

DEDICATION

I dedicate this work to Almighty God for His abundant grace

and

The International Society of Saint Vincent De Paul (SSVP)

ACKNOWLEDGEMENT

I deeply appreciate my supervisor Prof. **Ben Abdallah** for his tireless guidance and motivation throughout the period of this research, especially the faith he showed in me. I cannot forget one of our Skype meetings at the beginning of our research when I was studying to understand the concept of my work: he called my name and said: '*Chukwuka! This is research because you don't know it, if you had known it then is no more research*'. Most importantly, he gave me 100% rights as a fulltime research member of BenLab, Adaptive System Lab, University of Aizu, Japan. The best!

I thank my HOD, Prof. Amos David and other faculty members and staff of Computer Science, AUST, for their wealth of knowledge.

I am indebted to AUST President, Prof. Kingston Nyamapfene, the entire management and members of AUST community for their gesture of love. Their care is beyond gaining academic knowledge. I thank you all immensely for all the hospitality you offered me.

I also welcome the future ambassadors of my class: Mark, Sylvanus, Valentine, Abraham (roomie), Modibbo, Lukeman, Joseph, Nasiru, Falalu, Habib, Usman, Chris, Eze, Gbenga, Tunde, Jabir, Nkiru, Rukayya, Mulikat, Latifat, Habibah, Racheal, and other AUST students and the host of friends that made my life in AUST worth living. Thank you all!

Finally, I appreciate my family for their fervent prayers and support. Thank you Mummy!

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

2D	two-dimensional
3D	three-dimensional
ANNs	Artificial Neural Networks
bio-SNNs	biological Spiking Neural Networks
CPU	Central Processing Unit
FCFS	First-Come-First-Served
FIFO	First-In-First-Out
FPGA	Field-programmable gate array
GENESIS	GEneral NEural SImulation System
GPU	Graphics Processing Unit
IBM	International Business Machine
IF	Integrate-and-fire
IoT	Internet of Things
IP	Intellectual properties
LIF	Leaky Integrate-and-fire
LTD	long-term depreciation
LTP	Long term potentiation
NASH	Neuro-inspired ArchitectureS on Hardware
NoC	Network-on-Chip
NPC	Neuro Processing Core
OASIS-NoC	OASIS Network-on-Chip
ODEs	Ordinary Differential Equations
PCs	Personal Computers
PEs	Program Elements
RFC	Re-transmission flow control
ROLLS	Reconfigurable On-Line Learning Spiking
RTL	Register-transfer level

SNNs	Spiking Neural Networks
SoC	System on Chip
SRM	Spike Response Model
STDP	Spike Timing Dependent Plasticity
VLSI	Very large-scale integration

CHAPTER ONE
INTRODUCTION

1.1 Neuro-Inspired Systems and Spiking Neurons

Artificial Neural Network is an attractive, competitive and colossal research area in artificial intelligence which is inspired by the incredible and powerful performance of the interconnected biological brain. According to one of the first inventors of neurocomputing, Robert Hecht-Nielsen, Neural Networks is defined as 'a computing system made up of some simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs'. The whole idea of biological neural networks of the brain gave birth to Artificial Neural Networks (ANNs), with scientists digging deep on how and the best way to mimic the brain functionalities using silicon chips.

It is based on the fact that the biological brain connection architecture can be mimicked with silicon and wires in place of living neurons and dendrites. The human brain is a structure made of 100 billion cells named neurons which connect thousands of cells by axons (von Bartheld, Bahney & Herculano-Houzel, 2016). Inputs from sensory organs and the external environment are accepted by dendrites which create electric impulses that rapidly travel within the neural networks. Messages are sent across from neurons to other neurons.

However, Deep Learning as an exciting research area in machine learning is concerned with developing different algorithms inspired by the structure and function of the brain. Artificial Neural Networks have been developed to solve various computational problems, but earlier research never considered timing issues which are the hallmark of Spiking Neural Networks (SNNs). Currently, most ANN models are built on extremely simplified brain dynamics (Ghosh-Dastidar & Adeli, 2009). They have been used as popular computational tools to solve complex classification, function estimation and pattern recognition problems.

Spiking Neural Networks integrate Spike-timing-dependent Plasticity (STDP), which serves as a technique of adjusting the strength of connection (synapse) between neurons in the brain based on the relative timing of a particular neuron's output and input action potentials. In the past decade, SNNs were developed that comprising spiking neurons. Information transmission in these neurons mimics the information transmission in natural neurons due to their inherent dynamic representation. The massive parallelism of the brain has focused researchers' minds on many competitive areas of neuro-inspired systems, and many algorithms have been developed to enable machines and systems, even IoT devices, to leverage brain performance. Neurocomputing has various and robust applications in science and technology. The applications of neuro-inspired systems have actually been found inspiring in the field of Biomedicine and Neuroscience because the collaboration between biological and electronic circuits has led to ultra-low-power and noise-robust chips that could serve the deaf, blind, and paralysed and that could also lead to advanced ear-inspired radio receivers (Sarpeshkar, 2012).

1.2 Research Background and Motivation

The background of this research centres on Spiking Neural Networks (SNNs) modelled with the Leaky Integrate-and-Fire (LIF) spiking neuron model that integrates the STDP learning algorithm. We are motivated by the efficient and parallel processing of the biological neuron. The biological brain implements massively parallel computations using a complex architecture that is different from the current Von Neumann machine. Our brain is a low-power, fault-tolerant, and high-performance machine. It consumes only about 20 W and brain circuits continue to operate as the organism needs even when the circuit is perturbed. The interconnection of the brain neurons drives our motivation to future on-chip systems.

1.3 Statement of Problem

Timing is a significant issue in implementing the neuro-inspired system and is not considered in conventional neural networks. Conventional neural networks encode information with static input coding (encoding pattern as 0011 and 0010, binary bits).

While in SNN, besides the pattern coding, the time-related parameters can be used to present the information which increases the information processing capacity of ANN. Implementing SNN with spiking neuron models solves this timing problem, hence the need for this research.

1.4 Research Aim and Objectives

This research is aimed at exploring the theoretical framework behind spiking neuron models and investigating the architecture of OASIS Network on Chip (OASIS-NoC). In addition to our research studies aim, we ought to be guided by the following objectives:

- To implement a software-based SNN using the Leaky Integrate-and-Fire neuron model;
- To train our network with STDP learning algorithm;
- To test our SNN for digit recognition with MNIST datasets of handwritten digits;
- To propose a novel scalable high-level Neuro-Inspired Architectures in Hardware (NASH) for future OASIS Network on Chip (OASIS-NoC).

1.5 Research Methodology

The methodology of this research is based on studying related literatures to get the general concepts. In our studies, we proposed a design for a novel high-level architecture, NASH for future on-chip systems. NASH main components were described.

We implemented a software-based Spiking Neural Network (SNN) using a Leaky Integrate-and-Fire (LIF) neuron. The SNN is trained with an STDP learning algorithm for digit recognition using MNIST datasets of handwritten digits.

The rest of the research work is organized in chapters as follows: Chapter 2 is the literature review on some related works, discussing SNNs, learning mechanisms, neuron spiking models and software simulators; Chapter 3 covers methodology and implementation, discussing STDP, STDP algorithm, and methods; Chapter 4 is on the design of scalable SNN based on NoC Architecture, discussing NoC, OASIS-NoC and proposing a novel NASH as the main contribution of this research; Chapter 5 contains results, analysis, and evaluation; and finally Chapter 6 discusses the research conclusion, challenges, and future work.

1.6 Research Contributions

Interconnection of many cores on a single chip has remained a bottleneck in system design since high power consumption, scalability and high throughput must be considered appropriately. Network on Chip is a promising solution for efficient interconnection of many cores on a single chip (Ahmed & Abdallah, 2012). This research leverages NoC architecture to propose a novel high-level and scalable Neuro-Inspired Architectures in Hardware for complex cognition applications. Hence, we made the following sub-contributions.

1. Study and implementation of a software-based Spiking Neural Network (SNN) using a Leaky Integrate-and-Fire (LIF) neuron model with a spike timing dependent plasticity (STDP) learning rule.

2. We proposed a design for a novel architecture and circuit development towards the implementation of a spiking neuro-inspired architecture. We performed hardware design and evaluation of a LIF Core for Neuro-inspired Spiking architecture.

CHAPTER TWO
LITERATURE REVIEW

2.1 Spiking Neural Networks (SNNs)

Artificial neural networks (ANNs) research is motivated by how information is processed by the human brain using biological neurons (Wang, Belatreche, Maguire & McGinnity, 2010). The application of ANNs has cut across various areas of life due to extensive researches carried out by researchers. The study of various works from researchers on artificial neural networks with implementations on hardware and software including hybrid implementation show how the natural neuron of the brain can be mimicked using circuits and wires. Hence, this area of research in Artificial Intelligence is quite interesting though it demands knowledge of Machine Learning and Neuroscience. Davies (2012) describes artificial neural networks as circuits made by the interconnection of artificial neurons, which mimic the behaviour of biological neurons. This type of neural network needs to be simulated through a custom component designed for the purpose. Simulators comprise both hardware and software components that compute mathematical models of biological neurons and synapses given the necessary parameters. Artificial neural networks are classified into three major generations based on computational unit (Basegmez, 2014; Davies, 2012; Long, 2008; Maass, 1997).

In their work, Artificial Neural Networks (ANNs) are broadly classified into three generations. The first generation is known as the perceptron model (Maass, 1997). This is composed of binary inputs with threshold function; it uses flip-flop technology and is equally deterministic while communicating with feed-forward neural networks (FFNNs). The good example of activation function used in the first generation is a step function depicted in Figure 2.1 below. The main characteristic of perceptron models is that they can only produce digital output, applied only in digital computations. Every Boolean can be computed by some multilayer perceptron of the single hidden layer.

The second generation is good for analog computations (Maass, 1997; Nessler, Maass & Pfeiffer, 2009). The second ANNs activation functions are based on computational unit with a continuous set of likely output values to a weighted sum of inputs. Hence, the commonly used activation functions in the second generation include sigmoid activation function, linear activation function, and hyperbolic tangent activation function (see Figure 2.1 (d) below).

Spike Neural Networks (SNNs) are referred to as third generation Neural Networks (Basegmez, 2014; Christophe, Mikkonen, Andalibi, Koskimies & Laukkarinen, 2015; Davies, 2012; Krunglevicius, 2016; Maass, 1997; Shrestha, Ahmed, Wang & Qiu, 2017). In Spiking Neural Networks, activation functions used are always modelled as differential equations. They are regarded as neuron's state with incoming spikes pushing the value higher, which either decays or fires over time. As cited in Wang et al. (2010), SNNs have received significant attention from researchers due to the timing of individual spike neurons that have been proved computationally more powerful and biologically plausible than traditional ANNs.

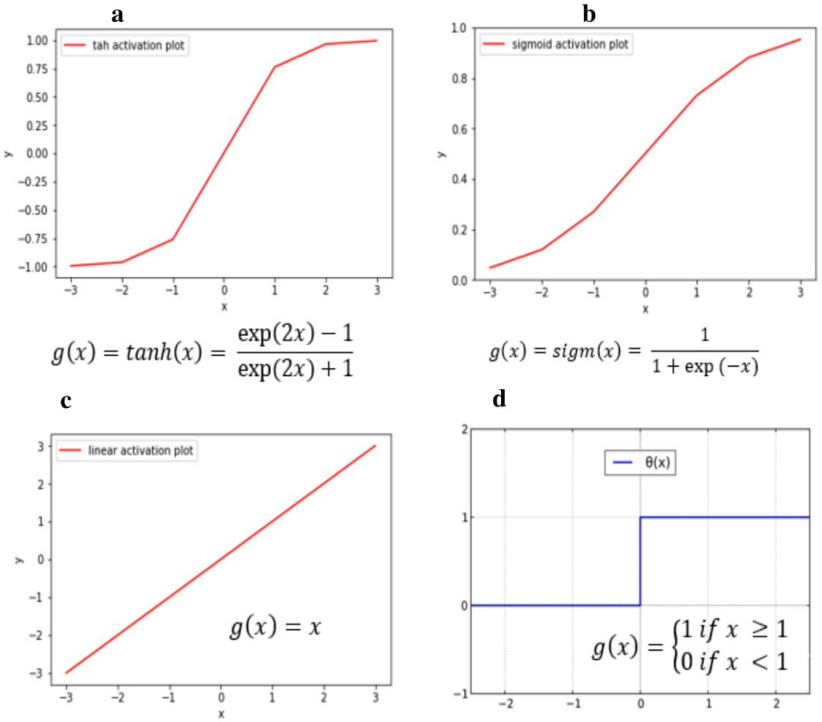

Figure 2.1: Activation functions (equations and curve)

2.2 Related Works

The application of Spiking Neural Networks in various extensive research projects and systems of neuron-inspired chips which implement efficient and low power consumption, attempting high computational accuracy by mimicking biological neuron performance, is a significant advancement and inspiration for this research. The inspiration and motivation from the related works and technologies made this research possible and interesting to carry out.

2.2.1 The IBM TrueNorth

A TrueNorth is a processor system designed by IBM solely for Spiking Neural Networks. It is a multicore neuromorphic chip developed by IBM with 4096 cores, and each core has 256 programmable neurons (Cassidy et al., 2013).

Hence, the neuron has 256 synapses which contribute to millions of synapses in a full TrueNorth chip. TrueNorth is an event-driven and highly energy efficient chip that is devoid of Von-Neumann architecture. Unlike conventional microprocessors, TrueNorth consumes a tiny fraction of the power, 45 mW of power for a million synapses, and at the same time uses an interconnected network of neurosynaptic cores. TrueNorth implements ‘gray matter’ on a spike-based messaging network with an intra-core crossbar memory and ‘white matter’ using long-range connections. The core has 256 axons and neurons, each approximating a complete 256 × 256 crossbar. In its implementation, synapses contain information which results in spiking of a neuron. The neurons are the resulting improvement on integrate and fire neurons with many more parameters. In the research work of Cassidy et al. (2013), it is noted that a TrueNorth neuron can replicate 20 biological neuron functions and behaviours. The general architecture of the neuron core, the construction and synthesis details of TrueNorth were discussed by Akopyan et al. (2015).

2.2.2 The ROLLS neuromorphic processor

Reconfigurable On-Line Learning Spiking neuromorphic processor (ROLLS neuromorphic processor) is a complete custom mixed-signal implementation (Qiao et al., 2015) because it is an open challenge to implementing compact, low-power artificial neural processing systems with real-time online learning capabilities. It is a full-custom mixed-signal VLSI device with neuromorphic learning circuits which mimic the biophysics of real spiking neurons and dynamic synapses for exploring the features of computational neuroscience models and for building brain-inspired computing devices. ROLLS neuromorphic processor is a useful learning circuit which mimics the functions of biological spiking neurons and synapses for exploring neuroscience models and building brain-inspired computing systems. The ROLLS neuromorphic processor device is made of 128K analog synapses and 256 neurons with online learning capabilities. The existing design is robust and capable enough to run a wide range of activities like recurrent and deep networks, with short-term and long-term plasticity. A prototype fabricated using a 180 nm processor consumes approximately 4 mW and takes an area of 51.4 mm^2, and produces results that showcase its potential. By supporting a wide range of cortical-like computational modules composed of plasticity mechanisms, this device enables the realization of intelligent autonomous systems with online learning capabilities.

2.2.3 The NeuroGrid

NeuroGrid is a computer hardware that is designed for the purpose of biological brain simulation. It uses analog and digital computation to mimic ion channel activity and to software structured connectivity patterns respectively. According to Benjamin et al. (2014) NeuroGrid is a good neuromorphic system with a goal of simulating massive models in real time. It can simulate up to a million neurons through billions of synaptic connections via 16 neuron cores embedded on a board that consumes 3 W. NeuroGrid processor also uses shared circuits as axons, synapses and dendrites to reduce transistor count, though designers of NeuroGrid systems face three major design choices: 1) a choice to emulate the

four neural elements − axonal arbor, synapse, dendritic tree, and soma − with dedicated or shared electronic circuits; 2) a choice to implement these electronic circuits in an analog or digital manner; and 3) a choice to interconnect arrays of these silicon neurons with a mesh or a tree network. The choices are to maximize the number of synaptic connections, maximize energy efficiency, and to maximize throughput respectively. In its operation, going outside from a fully digital implementation, they realized all electronic circuits. They also have custom-made routing interconnections and software that is capable of providing user access and reconfigurability.

2.2.4 The SpiNNaker

The research work of Painkras et al. (2013) defines Spiking Neural Network Architecture (SpiNNaker) as a computing engine with a million cores, meant to simulate the behaviour of up to a billion neurons in real time. The modelling of large systems of spiking neurons is computationally expensive regarding processing power and communication.

SpiNNaker is a massively parallel computer system developed to provide a cost-effective and flexible simulator for neuroscience simulation. It can model up to a billion neurons and a trillion synapses in biological real time. The initial objective was to simulate brain-scale sized neural networks for biological brain research. SpiNNaker deploys a cluster of ARM9 cores, using packet communication via a custom massively interconnected fabric. With up to 1,036,800 ARM9 core, SpiNNaker uses 64K bytes of data tightly coupled memory (DTCM) and 32K bytes 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 framework. In SpiNNaker operation, the computing engine consumes up to 90 kW of electrical power. The machine is built to mimic the brain's biological structure and behaviour, which exhibit massive parallelism and resilience to failure of individual components.

Given over one million cores, and one thousand simulated neurons per core, the SpiNNaker machine will be capable of simulating one billion neurons. This is equivalent to over 1% of the human brain's 85 billion neurons.

2.2.5 The 3D OASIS NoC

The 3D OASIS NoC serves as an excellent extension to 2D OASIS NoC architecture; the former overcomes the limitation of high communication cost, low throughput, and high power consumption unlike 2D OASIS-NoC architecture (Ahmed, Abdallah & Kuroda, 2010). Figure 2.2 below describes 3D Oasis-NoC, with 2×2×4 mesh topology. It has x_addr, y_addr and z_addr which define properties of 3D routing with X, Y and Z as their respective coordinates. It performs better than previous OASIS NoC with proven low power consumption.

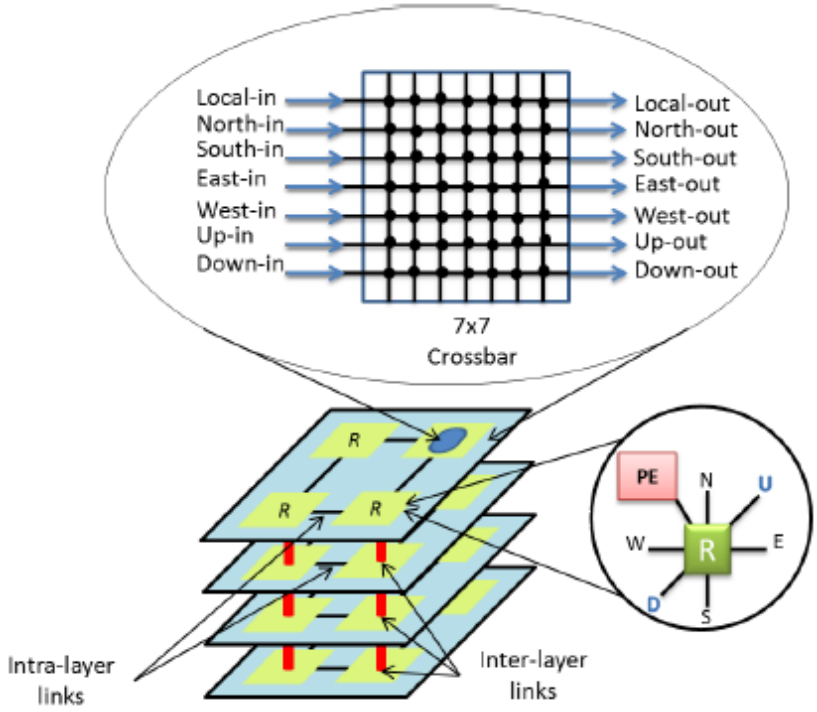

Figure 2.2: 3D OASIS NoC architecture (Ahmed, Abdallah & Kuroda, 2010)

2.3 Learning Mechanism

Artificial Neural network models involve various learning rules and algorithms. Learning defines a mechanism whereby a network adapts to an environment in which it is meant to act. According to Heskes and Kappen (1993), the outcome of adaptation processes of both artificial and biological neural networks defines the network general representation of its own environment. It is observed that different networks learn for the purpose of performing a particular function, which could be recognition, classification, or so on; this implies that various networks perform different functions based on training. However, despite the differences in the network functionalities, most learning algorithms share properties like training set, environment and learning parameters in common. It merely means that the learning environment and learning parameters are the paramount factors of any given neural network.

Machine Learning is needed more especially when human experience and direct computer programme performance are not enough to proffer a solution to a problem due to its complexity. It is quite certain that complexity and adaptivity usually demand machine learning (Ben-David & Shalev-Shwartz, 2014). It is pertinent to be careful when modelling neural networks in simulation to determine a precise form for the learning rule, because multiple learning algorithms have been developed, making it necessary to choose between several trade-offs amongst computational complexity, biological realism, and analytical tractability (Davies, 2012). Learning depends on a rule which determines the action of the network being trained based on that rule, learning algorithm. Learning algorithms of a Spiking Neural Network could involve offline or online learning.

2.3.1 Offline Learning Algorithms

The offline learning algorithms of any neural network have their network learning parameters updated in batches. The training dataset is obtained and processed in batches.

The batch updates denote the fact that the cost function is minimized based on the complete whole training data set. This type of learning rule does not support new data samples when the learning is in progress. The network must be retrained if it requires adding a new data sample to the existing training dataset. In the case of gradient descent, the total error on all training samples in the training dataset must be accumulated before applying gradient descent weight. This learning algorithm is never efficient when applied to a large network or a network with large datasets.

The first supervised learning algorithm, SpikeProp, serves as an adaptation of the classical backpropagation algorithm initially developed for SNNs which deployed fast temporal encoding using the spike response model. Performance analysis on several benchmark datasets demonstrated that SNNs with fast temporal coding could have similar results to rate-coded networks (Bohte, Kok & La Poutré, 2002).

However, issues like convergence speed for large datasets, problem of non-firing neurons, and the first spike priority, ignoring later spike and time-to-first spike coding, are optimized to improve SpikeProp's learning algorithm. According to the work of Moore et al*.,* as cited in Schrauwen & Campenhout (2004), repeated Bohte's research on the XOR problem was carried out, successfully training with a similar number of iterations on both large training rates and small training rates upon meeting the necessary conditions that all neurons must fire before the first weight update. It was shown in Bohte's work that only a small learning rate can be used because of approximation that exists in the threshold function.

Research also unveiled other learning algorithms to incorporate other learning parameters, besides the weights that could result in smaller network topologies and faster learning rule convergence (Schrauwen & Campenhout, 2004). Based on Analog Spiking Neuron, a new learning rule was presented that solves the neuron problem. This technique was greatly inspired by a continuous function, the recurrent analog network that could approximate the

spiking neuron's threshold function. Hitherto, the learning algorithm did not train the exact SNNs but rather the approximated SNNs, which is justified based on a model close to the exact and expected SNNs. The network can efficiently converge and, based on the Levenberg Marquardt method, the standard corrections and improvements of SpikeProp training algorithm have a fast convergence rate (Silva, Ruano & Ieee, 2006). In conclusion, according to McKennoch, Liu and Bushnell (2006), extended Resilient Propagation is used to improve the training speed from ANNs to SNNs. The performance measure of the typical XOR and IRIS datasets was rated about 80% faster while employing the SpikeProp technique.

The learning rule was based on irregular spiking that resembles natural neurons and gradient was not computed directly by this algorithm, rather estimated based on the correlation between the fluctuations in biological activities and the global reward signal. It is expected that neurons fire spike trains during the learning process.

In the work of Booij & Tat Nguyen (2005) a supervised learning algorithm for SNNs was presented and aimed at extending SpikeProp to handle multiple spikes framework, which involves making changes in parameters. However, due to changes in network learning parameters, the spiking time of a neuron will shift with the total number of spikes, though assuming that the spiking number was kept constant and backpropagating error values for each spike. The proposed rule was effectively utilized for classification based on a Poisson spike train. These gradient constructed algorithms are computationally powerful but are often viewed as non-biologically plausible because the learning rules usually require non-local spread of error signals across synapses of the system. A supervised learning algorithm has to do with a situation where the network is aware of the expected result, while the actual result is based error value. With offline learning, the weight changes depending on the complete data set of learning to define a global cost function. The training of a network using the offline learning rule repeats until a minimum of its cost function is reached.

2.3.2 Online Learning Algorithms

Online learning algorithms are the most commonly used machine learning approach for the neural networks. They involve updating learning parameters each time a new training sample is presented; the error being minimized for each sample. Online learning uses a stochastic process approach because the training example for each update is chosen randomly, which is required for learning and adapting in a continuously changing environment. In research work a simple online procedure to perform learning for a four-layer hierarchical neural network of two-dimensional integrate-and-fire neuronal maps was presented. The training was done through synaptic plasticity and adaptive network framework using an event-driven approach to optimize computation speed (Wysoski, Benuskova & Kasabov, 2008), and to simulate networks of large number of neurons. The training procedure was applied to a publicly available face recognition dataset, and the obtained performance was comparable to the optimized batch learning approach.

A research study by Soltic, Wysoski and Kasabov (2008) equally presented a simple artificial gustatory model of Spiking Neural Networks used for taste recognition, with its learning algorithm being developed using a simple integrate-and-fire neuron having rank order coded inputs. How the information encoding in a population of neurons influenced the performance of the networks was also analysed and the approach to two real-world datasets was tested. According to Alnajjar (in Alnajjar, Bin Mohd Zin & Murase, 2008), a self-adaptation system was developed to train a real mobile robot for optimal navigation in dynamic environments. It involves training some SNNs with the STDP learning property. The spike response model (SRM) was used, and the trained SNNs were stored in a tree-type memory structure that was used as experiences for the robot to enhance its navigation skill in new and previously trained environments. The memory was designed to have a simple searching mechanism.

Forgetting and online dynamic clustering techniques were used in order to control the memory size. Experimental results showed that a robot provided with learning and

memorizing capabilities was able to survive in complex and dynamic environments. The system used the minimum network structure required for performing obstacle avoidance tasks and its synaptic weights were changed online. However, more experimental data needs to be collected in order to demonstrate the robot navigation ability in a dynamic office environment. Also, the system still needs to be challenged in more complex environments.

In summary, there has been considerable research focus on developing offline approaches for SNNs, but very little has been achieved in developing online learning approach for SNNs. Developing efficient online learning approaches for SNNs is very important to increase their applicability to real-world problems and create intelligent systems that are capable of handling continuous streams of information, scaling up and adapting to continuously changing environments. Online Learning algorithm involves incremental learning.

The network weight changes made at a particular stage depend correctly only on the current input signal into the network. This is the natural procedure for time-varying network learning rules.

2.4 Neuron Spiking Models

Spiking Neural Networks (SNNs), known as the third generation of a neural network which regards timing as a significant parameter, depicts an outstanding level of realism in a neural simulation as compared to biological neuron spiking (Maass, 1997). Artificial neural networks define forecasting approaches that are based on simple mathematical models of the brain, with the primary objective of mimicking brain dynamic functions and performance. The objective of all neuron spiking models lies in their ability to model complex nonlinear relationships between the response parameters and their corresponding predictors in a given system.

In biophysics, action potentials are the consequence of currents that pass through ion channels found in the cell membrane (Gerstner & Kistler, 2002).

Hence, Hodgkin and Huxley tried to estimate these currents by describing their dynamics with mathematical models using differential equations by carrying out a series of experiments with a giant axon found in squid. The Hodgkin–Huxley equations remain the bases of neuron models. They invariably account for the numerous ion channels, the varieties of the synapse, and the specific spatial geometry of a particular neuron. In principle, spiking neuron model accounts for utilizing action potentials – spikes, and at the same time how this state relates to the spikes the neuron fires (Ahmed, Yusob & Hamed, 2014). As reviewed and cited by Ponulak and Kasinski (2011) the primary assumption behind the implementation of most of spiking neuron models is that it is the timing of spikes rather than the specific shape of spikes that carries neural information processing. There exist different models of spiking neuron models, but in this research, we are interested in just a few that are considered most significant and straightforward in describing and working with Spiking Neuron Networks (SNNs).

They include Hodgkin and Huxley, FitzHugh–Nagumo, Integrate-and-Fire, and Leaky Integrate-and-Fire neuron models.

2.4.1 Hodgkin and Huxley Neuron Model

This model is the basis and foundation of every other spiking neuron model. In 1952, Hodgkin and Huxley, as described and cited in the research work of Gerstner and Kistler (2002), carried out experiments on the giant axon of the squid and discovered three different types of ion current that includes, sodium, potassium, and a leak current mostly of Cl-ions, see Figure 2.3(b) below. Definite voltage-dependent ion channels, one for sodium and the other for potassium, control the flow of those ions through the cell membrane while the leak current takes care of other channel forms. The Figure 2.3(a) below shows a semi-permeable cell membrane that separates the interior of the cell from the extracellular liquid and acts as a capacitor. If an input current I(t) is injected into the cell.

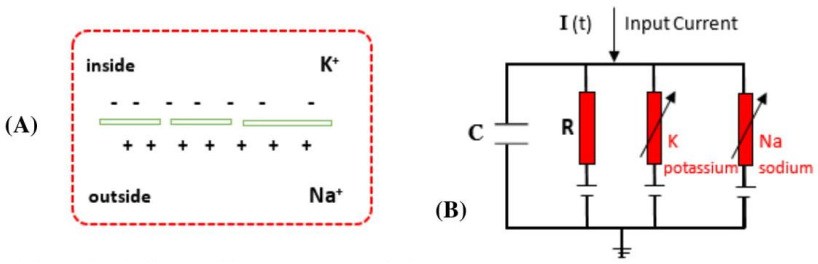

Figure 2.3: Hodgkin and Huxley Neuron Model

As seen in Figure 2.3(b) it may add a further charge on the capacitor, or leak through the channels in the cell membrane. Hence, from the experiment, it was observed from the cell membrane that the ion concentration inside the cell is different from that in the extracellular liquid while the battery shows the Nernst potential generated by the difference in ion concentration. In putting the whole concept in mathematical models and equations, we have that preserving electric charge on a piece of membrane means intuitively dividing input current $I(t)$ into capacitive current $I_C$that charges the capacitor C with current from other components, say I_k as depicted below:

$$1. \qquad I(t) = I_C(t) + \sum_k \big(I_k(t)\big)$$

Where the ∑ runs overall ion channels (various ions) as stated in the standard Hodgkin–Huxley model. By definition capacitance is well-defined as *C* = *Q*/*u* having *Q* as a charge and *u* the voltage across the capacitor; then, charging current becomes:

$$2. \qquad I_C = C\frac{du}{dt}$$

Then from equation 1 and 2 above, we could have:

$$3. \qquad C\frac{du}{dt} = -\sum_k I_k(t) + I(t)$$

The channels are characterized by their resistance or conductance. The leakage channel is defined by a voltage-independent conductance $g_L = \frac{1}{R}$; the conductance of the other ion channels is considered as voltage and time dependent. If all channels remain open, they transmit currents with a maximum conductance g_{Na} or g_K respectively; however, some of the channels are blocked. Hence, the probability that a channel is open can be expressed as additional parameters *m, n,* and *h*. The combined action of *m* and *h* controls the Na^+ channels. The K^+ gates are controlled by *n* (Gerstner & Kistler, 2002). Therefore, Hodgkin and Huxley in their experiment formulated the necessary three current components as:

$$4. \qquad \sum_k I_k = g_{Na} m^3 h(u - E_{Na}) + g_K n^4 (u - E_{\mathrm{k}}) + g_L (u - E_{\mathrm{L}})$$

The parameters E_{Na}, E_{K}, and $E_{\mathrm{L}}$are the reversal potentials which in addition with conductance serve as the empirical parameters. In conclusion, it has been shown that the Hodgkin–Huxley neuron models properties of the membrane potential, with regard to the actions noticeable from the biological neuron with a rapid increase at firing time, followed by absolute refractoriness (a short period when the neuron is unable to spike again), and the relative refractory period (a further time period when the membrane is depolarized and renewed firing here is much more difficult.

The Hodgkin-Huxley model is realistic and biologically plausible but very much more complex to analyse and expensive in terms of NNs simulation (Ahmed, Yusob & Hamed, 2014).

2.4.2 FitHugh-Nagumo Neuron Model

This model is a two-dimensional model derived from the four-dimensional Hodgkin–Huxley neuron model for simplicity. According to Zillmer (2007) the two-dimensional FitzHugh-Nagumo Neuron Model can be modelled as membrane potential and current variable. In 1961 and 1962 two great scientists, suggested the system and modelled the equivalent

circuit respectively (Izhikevich & FitzHugh, 2006). Hence the equations of FitzHugh–Nagumo Neuron Model are given as:

$$V = V - \frac{V^3}{3} - W + I$$

5.

$$W = \phi(V + a - bW)$$

This represents a two-dimensional simplified version of the Hodgkin–Huxley model during spike generation in squid giant axons. Having *V* as the membrane potential, *W* is a recovery variable and *I* is the magnitude of the stimulus current, where ϕ, *a* and *b* are positive constants with 0.08, 0.8 and 0.7 as original values respectively. It is possible to take the derivative of the above equation to have:

6. $$\frac{dv}{dt} = V - \frac{V^3}{3} - W + Iex$$

Finally, the FitzHugh–Nagumo model is a simplified version of the Hodgkin–Huxley model which models in a detailed manner activation and deactivation dynamics of a spiking neuron. As spiking model, it provides and allows effective phase plane analysis, excitability and refractory period.

Unfortunately, the FitzHugh–Nagumo model does not support bursting, has no self-sustained chaotic dynamics, and is difficult to adapt to neurons with specific properties (Zillmer, 2007).

2.4.3 Integrate-and-fire (IF) and Leaky IF (LIF) neuron models

The comprehensive high-dimensional Hodgkin−Huxley spiking neuron model is biologically plausible, but it is very complex to analyse and difficult to implement due to its complexity. This led to the call for simplified neuron models with the primary objective of reducing the

four-dimensional Hodgkin−Huxley model to a two-dimensional model. According to Gerstner and Kistler, 2002, the key clue of the four-dimensional reduction is to eliminate two of the four variables in the Hodgkin−Huxley model based on two qualitative observations (Gerstner & Kistler, 2002). The first observation is that in the Hodgkin−Huxley model, the timescale of the dynamics of the activation gate *m* seems to be much faster than that of the other variables *n, h*, and *V*. Hence, *m* can be taken as an instantaneous variable, replaced by its steady-state value m_0; this is also referred as a quasi-steady-state approximation.

The second observation lies on the Hodgkin-Huxley model *n* and *h* variables. Both *n* and *h* can be replaced by a single effective variable because their timescales are approximately the same. These assumptions have led to several two-dimensional models being proposed, such as the Morris-Lecar model and the FitzHugh-Nagumo model. These two-dimensional models are conductance-based models, in which the variables and parameters have distinct biological meanings, and can be measured experimentally. Unfortunately, the conductance-based models are also complex to analyse and simulate. However, the simplified models, on the other hand, are not biologically plausible though of course, they do address most critical features of neurons and are approximately less computationally intensive regarding their implementation.

- The three key properties of a simplified neuron-like that of the Integrate-and-fire model that could be addressed during modelling include:
- The ability to generate spikes if the membrane potential crosses a threshold;
- A reset value to initialize the membrane potential after firing;
- A given refractory period to decrease the neuron from generating another spike immediately.

The simplified models which possess the above features are very easy to implement and analyse, and are more popular in computational neuroscience.

They include the Integrate-and-fire (IF) model and the Leaky Integrate-and-fire (LIF) model as cited in Jin, 2010. These are probably the best-known spiking neuronal models. Figure 2.4 below shows a schematic diagram of the LIF model. LIF is an integrate-and-fire model with a 'leak' term added to the membrane potential to solve the memory problem in the system. The basic circuit of the LIF neuron model architecture is composed of a capacitor *C* in parallel with a resistor *R* driven by an input current I.

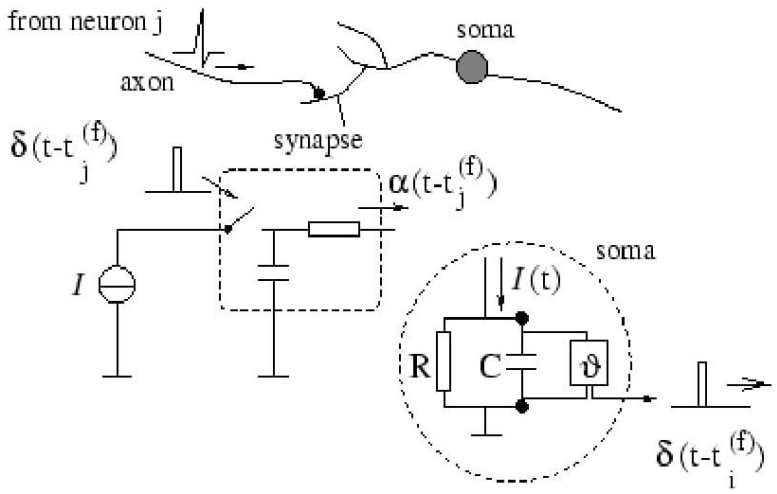

Figure 2.4: Leaky IF (LIF) neuron model (Jin, 2010)

Based on the circuit description, we could have:

$$I = \frac{V}{R} + C\frac{dV}{dt} \quad (7)$$

Then by introducing a time constant $\tau_m = RC$ of the 'leaky integrator', we could have the standard form of the LIF model as:

$$\tau_m \frac{dV}{dt} = -V + RI \quad (8)$$

Where *V* is the membrane potential and τ_m is the membrane time constant. In this model, if the membrane potential *V* reaches the threshold value V_{thresh}, the neuron fires then *V* is reset to a certain value V_{reset}. In the general version, the LIF model also incorporates an absolute

refractory period t_{abs}. If the neuron fired at time t, we stop the neuron dynamics for a period of tabs and start the dynamics again at time t + tabs with $V = V_{reset}$.

The LIF model is simple enough to implement and easy to analyse. However, it has a severe drawback − it is too simple to reproduce the versatile firing patterns of real neurons.

2.5 Software Simulators and Examples

Simulators could be hardware or software based depending on their architecture and operational environments. They are considered hardware simulators if they are integrated to run on hardware chips. Hence, simulators are software simulators if they are implemented to run on a standard computational unit like a standalone PC (Personal Computer) or a computer cluster.

Mathematical models and equations are used as a framework to implement neuron models in software simulators. The neuron models are developed by some lines of code which implement the mathematical model in the form of ordinary differential equations (ODEs) of a biological neuron. Such simulators simulate multiple neuron models even at the same run of a simulation since the neuron model here is implemented in software.

According to Brette et al. (2007) and Rast (2010), software simulators can be synchronous simulators (clock driven) or asynchronous simulators (event-driven), meaning they can usually run in discrete time or in abstract time respectively. The speed of software simulators regarding time is inversely proportional to the number of neurons being simulated per simulation.

Generally, the larger the number of neurons simulated, the slower the simulation, since the simulator substrate has a finite computational power shared between all the neurons. Therefore, the greater the number of neurons simulated, the greater the computational resources demanded from the simulator, and the greater the time required to perform a time

step in the simulation. Finally, for medium scale neural networks, the time relation goes below the real-time boundary, making it slower than the real-time.

2.5.1 Brian Simulator

Brian Simulator is a software for a Spiking Neural Network simulator written in Python (Brette & Goodman, 2008). It is an open source Python package for implementing simulations of networks of spiking neurons with the primary goal aim of maximizing users' development time and having execution speed as a secondary goal. It is possible to use Brian Simulator to implement multiple neuron models, but much slower than real-time (hardware) implementation, mostly when simulating large and complex neural networks. Brian Simulator is a clock-driven simulator (synchronous simulator), where all events occur on a fixed time grid with learning features. The primary focus of Brian Simulator is to make the writing of simulation code fast and flexible for the developer. It allows developers to spend more time on the details of their models, and less on their implementation. Neuron models are defined by writing differential equations in standard mathematical form, facilitating scientific communication and research. Brian Simulator is written in the Python and uses vector-based computation technique for efficient simulations (Brette & Goodman, 2011). The spike neural network researchers and developers are not restricted to using neuron and synapse models already built into the simulator.

2.5.2 NEURON Simulator

In 2007, Brette et al. described NEURON as a software environment simulator for creating and using models of biological neurons and neural circuits, as cited in Davies (2012).

This software simulator provides tools for suitably creating, managing, and using models in a way that is mathematically sound and computationally efficient. It is supported by a complete development environment to describe characteristics of neurons and neural circuits and is suitable for problems with complex anatomical and biophysical properties.

To advance simulations in time, users have a choice between built-in clock-driven methods (a backward Euler and a Crank-Nicholson variant both using fixed time step) and event-driven methods (fixed or variable time step which may be system-wide or local to each neuron, with second-order threshold detection).

NEURON simulator provides users with a choice of programming languages as most programming with the simulator can be done with hoc. Hoc is an interpreted language similar to C-syntax that has been extended to include functions specific to the domain of simulated neurons, with a graphical interface and object-oriented programming features. Recently, Python was adopted as an alternative interpreter for NEURON simulator. Its graphical user interface can be used to create and exercise models that have a wide range of complexity. With the GUI, it is possible to generate publication-quality results without having to write any program code at all. However, combining both approaches helps to exploit the strength of the simulator. Finally, NEURON as a software simulator allows multiple neuron models in the same simulation and equally implements learning features. It uses both a discrete clock-driven and event-driven time paradigm in its operation; also NEURON simulator runs slower than real-time.

2.5.3 GENESIS Simulator

GENESIS by Wilson et al. (1989) refers to GEneral NEural SImulation System. It is a software simulator that aims to reproduce the biological behavior of neural systems with details ranging from biochemical reactions to large-scale neural networks. GENESIS was the first simulator to adapt to a large-scale neural network. GENESIS uses a high-level simulation language to construct neurons and their networks.

Commands may be issued either interactively to a command prompt, by use of simulation scripts, or through the graphical interface.

A particular simulation is set up by writing a sequence of commands in the scripting language that creates the model itself and the graphical interface for a particular simulation.

The scripting language and the modules are powerful enough that only a few lines of script are needed to specify a sophisticated simulation. Finally, in principle GENESIS is a software simulator that allows multiple neuron models in the same simulation and implements learning features.

CHAPTER THREE
METHODOLOGY AND IMPLEMENTATION

3.1 Introduction to STDP

Artificial neural networks have involved spiking dynamics to attain greater computational efficiency, and such powerful and attractive features are being leveraged with on-chip implementation using dedicated neuromorphic hardware. Efficient and effective learning algorithms were difficult to implement before the arrival of spike-timing-dependent plasticity (STDP) which is considered the best Spiking Neural Network learning framework (Markram, Gerstner & Sjöström, 2012). They observed that the STDP learning rule adjusts its strength of connection concerning the time change between the pre- and post-synaptic spikes. According to Sjostrom and Gerstner (2010), STDP could be seen as an asymmetric type of Hebbian learning triggered by the tight temporal correlation between the spikes generated by the pre- and postsynaptic neuron.

The STDP learning rule which inspires learning together with information storage of the brain as described by Bi & Poo (2001) in their article titled ‘Synaptic Modification by Correlated Activity’ encourages repeated presynaptic spikes arrival few milliseconds before the postsynaptic action potential, indicating long-term potentiation (LTP) of that synapse. Meanwhile, the same repeated spike arriving after postsynaptic spikes indicates long-term depreciation (LTD) of the same synapse as depicted in Figure 5(a) below. Hence, the repeated occurrence of LTP and LTD is referred to as STDP that leads to continues change of synapse. This change of synapse is plotted to be a function of relative timing between pre- and postsynaptic action potential and such plot is called STDP curve or leaning window, see Figure 3.5(b) below.

3.2 STDP Model and Algorithm

The STDP learning mechanism or model defines a function of time; it is a relation of synaptic weight and the time of the spike occurrence between neurons. Unlike other learning algorithms like backpropagation, which considers error as a significant factor in learning and updating of weight parameter across different layers of a network, STDP updates the synaptic connection of the network weight (i.e. weight strengthening or weakening) based on the change in time between the pre- and postsynaptic neuron. Studies show that potentiation (increase in connection strength) occurs if a presynaptic neuron fires before a postsynaptic neuron, otherwise there will be depreciation (decrease in connection strength). The concept is motivated by timing, hence a Spiking Neural Network (SNN) differs from another conventional neural network due to timing concept. Timing is the major headache of SNN but the STDP learning solves this problem. The equation below defines the STDP learning function (model).

9. $$\text{STDP} = (\Delta t) = (\Delta w) = \begin{cases} A^{-} \exp * \left(\frac{\Delta t}{\tau -} \right), & if \ \ \Delta t \leq -2 \\ 0, & if \ -2 < \Delta t < 2 \\ A^{+} \exp * \left(\frac{\Delta t}{\tau +} \right), & if \ \ \Delta t \geq 2 \end{cases}$$

Where A^{-} and A^{+} are constants for –v and +v values of Δt between presynaptic and postsynaptic spikes; τ^{-} and τ^{+} are constants denoting excitation and inhibition values showing the steepness of the function. The model above can be represented graphically as an STDP curve as in Figure 3.5(b) below.

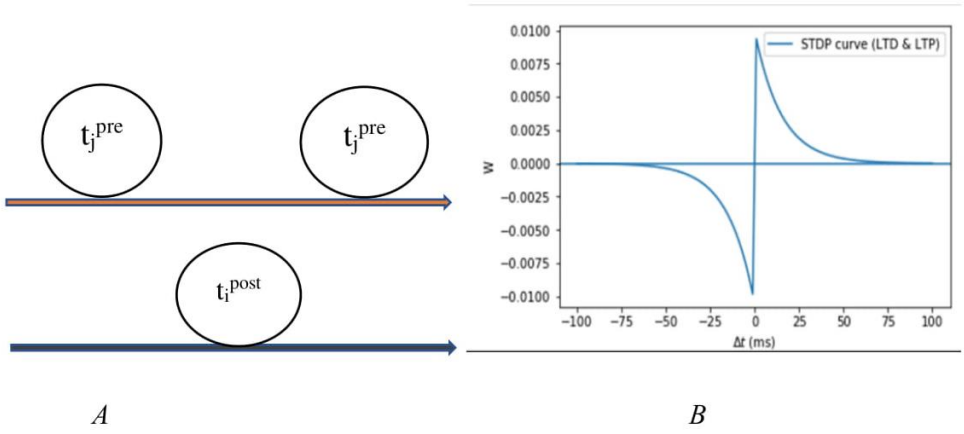

Figure 3.5: (a) LTD and LTP occurrence via pre- and postsynaptic behaviour; (b) STDP curve

The STDP curve shows a fitted correlation between the horizontal and vertical axes. The vertical axis represents the synaptic weight modification while the horizontal axis represents the time between pre- and postsynaptic spikes ($\Delta t = t_{pre} - t_{post}$). The weight change, Δw in the approximated equation 9 above is the function of time change, Δt. Hence, timing is important in studying and modelling SNN because a neuron embedded in a network could be flooded with thousands of inputs every second. The question becomes which ones are to be prioritized, which information the neuron should listen to and process. Timestamp helps to solve such problems during neural network development and learning. Therefore, to correctly choose and fine-tune the inputs from one's neighbours without any other information than that which is received from these neighbours themselves invokes the time factor. STDP learning rule is a good suite of spike neural networks. In modelling any neural network, it is necessary to determine a precise form of the learning rule. Series of trade-offs like computational complexity, biological realism, and analytical tractability, according to Song and Abbott (2001), make STDP the best learning algorithm. In their research, Spike Timing Dependent Plasticity rule considers every possible combination of pre- and post-synaptic spikes, modifying correspondingly the synaptic weight of the synaptic connection. The STDP algorithm is defined by the pseudocode below as adapted from Davies (2012).

FOREACH pre-synaptic spike **Do**

F**oreach** post-synaptic spike **Do**

Δt = **T**ime$_{pre}$ – **T**ime$_{post}$;

Δw = compute**C**hange**W**eight(Δt);

new_weight = update**S**ynaptic**W**eight (current weight, Δw);

E**nd**

END

Algorithm 1: STDP Algorithm

The above routine describes the standard STDP rule, which modifies the synaptic weights pairing the time of all the pre-synaptic spikes with the time of all the post-synaptic spikes. The function of the pseudocode includes, Time$_{pre}$: Time of the pre-synaptic spike; Time$_{post}$: Time of the post-synaptic spike; compute weight modification (): computes the weight modification with respect to the STDP curve; update synaptic weight (): updates the synaptic weight according to the current weight and the weight modification previously computed, applying also hard limits on the synaptic weight value, where appropriate.

3.3 Methods

In our research we adopted the software simulation approach. Python programming language and BRIAN 2 were used in our simulation; more and complete details of BRIAN 2 Simulator can be found in Goodman and Brette (2009) and Goodman, Stimberg, Yger and Brette (2014). This section discusses the dynamics of single neuron and synapse, our network architecture and its mechanism. It equally describes the MNIST dataset of LeCun, Cortes and Burges (1998) used to test our simulation. MNIST datasets comprise 60,000 training examples and 10,000 testing examples.

Training the algorithm with the dataset and its classification procedures provides us with reasonable benchmark adopted for this work (Diehl & Cook, 2015). Therefore, the simple training method proposed in this research work is in accordance with the knowledge of training only the output layer; the STDP training rule is meant to act alone for strengthening the positive pathways of the network. The SNN network studied here is a feed-forward network topology.

3.3.1 Computational Model

In our research study, the major criteria of selecting our computational model were based on its ability to mimic closely the function of the biological neural networks of the brain, its feasibility in terms of implementation, test and computational cost. Hence, the computational model adopted for this research comprised primary elements of our network (neurons and synapses). In our SNNs, neurons are modelled based on the Leaky Integrate-and-Fire neuron model which is considered to be a realistic and simplified model (Eugene, 2007; Gerstner & Kistler, 2002). The synapses adapted the Spike Timing Dependent Plasticity (STDP) model which is our major research focus; the connection between two neurons grows positive (LTP) if the post-synaptic neuron fires soon after the pre-synaptic neuron while decreasing (LTD) if the post-synaptic neuron fires before the pre-synaptic neuron, both leading to synaptic plasticity. Therefore, the computational model used is presented in more details in forthcoming sections which gives the composition of the entire Spiking Neural Network.

3.3.2 Spiking Neuron and Synapse Model

The Spiking Neuron Model adopted and used in this research is the Leaky Integrate-and-Fire (LIF) neuron model (Gerstner & Kistler, 2002). In the course of our research study, this particular neuron model is found to replicate the dynamic behaviour of neurons while being computationally simple, unlike the Hodgkin−Huxley model.

The Leaky Integrate-and-Fire neuron model's dynamic behaviour can be expressed in the form of an ordinary differential equation (ODE) as described in the work of Christophe, Mikkonen, Andalibi, Koskimies and Laukkarinen (2015) as:

$$10. \qquad \tau_m \frac{dV}{dt} = -V + RI(t)$$

Equation 10 describes a simple resistor-capacitor (RC) circuit model as shown in the circuit diagram below where the leakage term is due to the resistor and the integration of I(t). The equation models dynamically the parameters like Spiking threshold, Reset potential and Absolute refractory period.

```
# differential equation of Leaky Integrate-and-Fire model
eqs = '''
    dv/dt = ( -(v-v_rest) + R * I ) / tau : volt
    I = curr(t) : amp
'''
```

The above code defines LIF neuron is modelled using Brian 2 simulator.

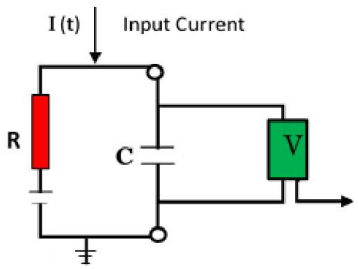

Figure 3.6: A block diagram of LIF spiking neuron circuit

In the LIF neuron model, the input current *I*(*t*) of the integrate-and-fire model is usually generated by the activity of presynaptic neurons. According to Gerstner and Kistler (2002) the total input current to neuron *i* is the sum over all current pulses as depicted in equation 11 below.

The simulation approach of the LIF neuron used is stimulation by the synaptic current. Unlike other approaches (e.g. stimulation by constant input current and stimulation by time-varying input current), this approach does not support both direct injections of constant and time-varying currents of any kind. Rather the neuron is stimulated by pre-synaptic spike arrival to postsynaptic (target) neuron which shows a realistic situation compared to natural neuron behaviour and functions, as defined in a synapse, when using our STDP learning rule. It shows that when a neuron is stimulated by pre-synaptic spike arrival. Thus; to obtain post-synaptic current we apply this equation:

$$11. \qquad I_i(t) = \sum_j wij \sum_f \propto \left(t - t_j^{(f)}\right)$$

Where $t_j^{(f)}$ represents the time of the *n*-th spike of the *j*-th pre-synaptic neuron; w_{ij} is the strength (weight) of synaptic efficacy between neuron *i* and neuron *j* respectively.

In addition, playing over with equation 10, ODE above we have:

$$12. \qquad \frac{dV}{dt} = \frac{1}{\tau m}(-V + IRm)$$

By introducing refractory period in equation 12, we obtain:

$$13. \qquad \frac{dV}{dx} = \begin{cases} \frac{1}{\tau_m}(-V + IR_m) t > t_{rest} \\ 0 otherwise \end{cases}$$

Therefore, in our research work and implementation, having adopted LIF neuron model as a framework of our Spiking Neural Network, the voltage membrane parameter *V* used is expressed in the form of an ordinary differential equation, ODE (Diehl & Cook, 2015) as:

14. $$\tau \frac{dv}{dt} = (E_{res-}V) + g_e(E_{exc} - V) + g_i(E_{inh} - V)$$

With parameters being defined E_{rest} (the membrane resting potential), E_{exc} (the equilibrium potential of an excitatory synapse), E_{inh} (the equilibrium potential of an inhibitory synapse), and g_e (the conductance of an excitatory synapse) and g_i (the conductance of an inhibitory synapse). As recorded in biology, we use a time constant τ, which is longer for excitatory neurons than for inhibitory neurons. However, if the neuron's membrane potential crosses its membrane threshold v_{thres}, the neuron fires and its membrane potential is reset back immediately to v_{reset}. Within the next few milliseconds after the reset, the neuron is in its refractory period and can no longer spike.

Synapse modelling involves creating synapse instances which specify their dynamics. The synapses are modelled by conductance changes, i.e., synapses increase their conductance instantaneously by the synaptic weight *w* when a presynaptic spike arrives at the synapse, otherwise the conductance decays exponentially. Then given the presynaptic neuron as excitatory, the dynamics of the conductance g_e are defined as:

15. $$\tau_{ge} \frac{dge}{dt} = -ge$$

Where τ_{ge} is the time constant of an excitatory postsynaptic potential. Also, if the presynaptic neuron is inhibitory, a conductance g_i is updated using the same equation but with the time constant of the inhibitory postsynaptic potential τ_{gi}. Using equations (14) and (15) above, the framework of our Spiking Neural Network is modelled using BRIAN 2 simulator.

3.3.3 Architecture and Network Learning

The SNN architecture is shown in Figure 3.7. It is composed of two layers, input and output. The first layer comprises 28 × 28 neurons, i.e., the first layer has neurons of 784 pixels each. Then the second layer (also called the processing layer) has various numbers of excitatory

and inhibitory neurons. Poisson spark-train which serves as network input is fed to the network via excitatory neuron.

The rates of every network neuron are directly proportional to their respective image pixel intensity. In the second layer we have inhibitory neurons which are connected to the excitatory neuron in a one-to-one correspondence. It means that each excitatory neuron should generate a spike in its equivalent inhibitory neuron, showing that the connectivity provides a lateral inhibition which causes competition between excitatory neurons. Synaptic conductance between the inhibitory and excitatory neuron is balanced by fixing the maximum of the synapse to 10 nS. However, every synapse in the network, beginning with input neurons, down to excitatory neurons, is learned with the STDP rule. The simulation speed was also improved by computing weight dynamics using synaptic traces (Morrison, Aertsen, Diesmann & Morrison, 2007). Hence, each synapse keeps record with another value, like the presynaptic trace x_{pre}, which models the current presynaptic spike occurrence. Each time a presynaptic spike arrives, the trace is incremented by 1, otherwise it decays exponentially. Therefore, if a postsynaptic spike arrives in the synapse, weight change Δwwill be calculated using:

$$16. \qquad \Delta w = \eta(Xpre - Xpost)(Wmax - w)^{\mu}$$

Where η is the learning rate, w_{max} is the maximum weight and μ determines the dependence of the update on last weight, x_{post} is the target value of the presynaptic trace at the moment of a postsynaptic spike. The target value is inversely proportional to the synaptic weight.

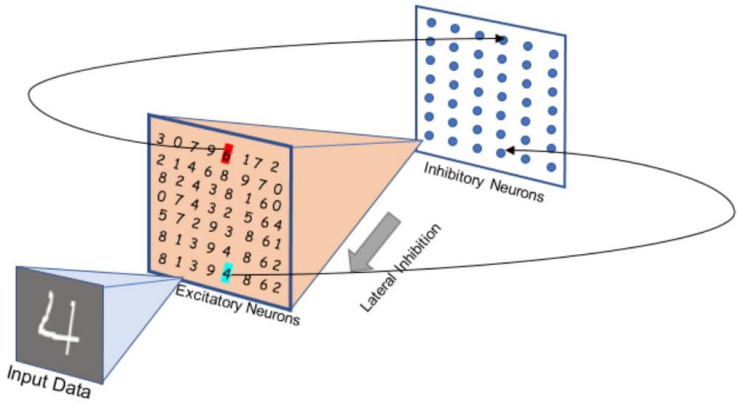

Figure 3.7: Network architecture (Diehl & Cook, 2015)

The intensity values of the 28 × 28 pixels MNIST image are converted to Poisson-spike with firing rates proportional to the intensity of the corresponding pixel.

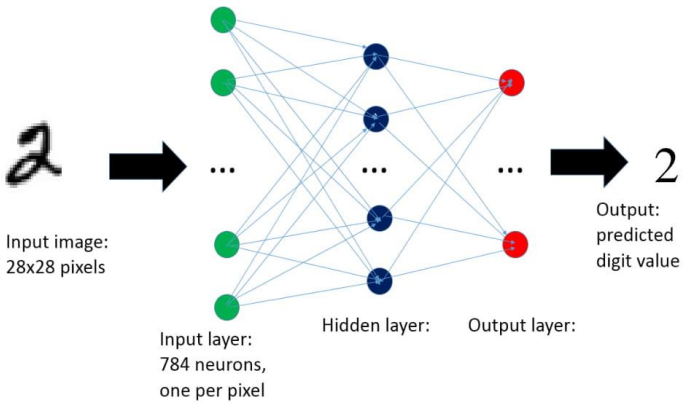

Figure 3.8: Network Training Topology

The network is of two layers showing 28 × 28 image fed through input layer. Prediction and classification occur at the hidden layer.

3.3.4 Data Encoding

MNIST dataset by LeCun, Cortes and Burges (1998) is the prerequisite of our network input encoding.

This dataset contains 60,000 training examples and 10,000 test examples of images with digits ranging from 0-9 each with 28 × 28-pixel dimension as depicted in Figure 3.8 above. The MNIST dataset is already preprocessed; we used the dataset to train our network. The input is presented to the network as Poisson distributed spike trains for 30ms having firing rates directly proportional to the intensity of the MNIST dataset images. The intensity of the pixel ranges from 0 to 255 and maximum intensity of 255 is divided by 4 indicating the input firing rates between 0 and 63.75 Hz.

	0	1	2	3	4	5	6	7	...	27
0	0	1	2	3	4	5	6	7	...	7
1	1	9	8	7	5	3	9	7	...	5
2	2	9	8	7	9	3	2	5	...	5
3	3	7	7	5	4	0	0	8	...	3
4	4	1	0	9	1	1	0	8	...	7
5	5	7	6	5	4	9	2	8	...	0
6	6	8	2	4	5	2	2	9	...	3
7	7	9	7	9	0	0	0	0	...	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
27	7	6	8	4	0	4	4	9	0	9

Figure 3.9: Image pixel (28 × 28) of network input

3.3.5 Network Training and Classification

In training the network with MNIST dataset, 60,000 training set examples; small portions were brought to the network and trained due to the low capacity of our PC resources.

The network made provision for the neurons to decay to their resting values by initially implementing 150 ms phase without any input to the network. After training was done, we set the learning rate to zero, fixed each neuron's spiking threshold, and allocated a class to each neuron, based on its highest response to the ten classes of digits over one presentation of the training set. Labels were never used except at this stage, showing that in training of the synaptic weights we do not use labels, due to the unsupervised learning nature of the network. Finally, during the training, the response of the class-assigned neurons is then used to rate the classification accuracy of the network on portions of the MNIST test datasets of 10,000 examples. The predicted digit is determined by averaging the responses of each neuron per class and selecting the class that has the highest average firing rate.

3.4 Chapter Summary

In summary, this chapter implements a software-based SNN. It discusses the STDP algorithm used, the architecture and topology and data encoding of the network. It equally highlights the training and classification of our network.

CHAPTER FOUR
ON THE DESIGN OF SCALABLE SNN BASED ON NOC ARCHITECTURE

4.1 Introduction

Research has shown that Network on Chip (NoC) is a promising solution for efficient interconnection of many cores on a single chip, unlike traditional shared-bus based systems (Ahmed & Abdallah, 2012). The peak performance of neuromorphic hardware systems and chips can be attained by leveraging the NoC architecture design. This is because spiking neuro systems mimic the biological Spiking Neural Networks (bio-SNNs) functions where massive parallelism, scalability and fault tolerance are the paramount factors and research focus; hence, the decline from conventional bus or System on Chip (SoC) architecture. Having described the implementation of our research in Chapter 3 on software-based; this chapter discusses the proposed hardware design of SNN based on scalable NoC architecture using OASIS-NoC developed at Adaptive Systems Laboratory.

4.2 Network on Chip (NoC)

This is a communication subsystem that exists on an integrated circuit, typically between intellectual property (IP) cores in SoC. NoC technology is merely an application of all networking techniques to on-chip communication to establish essential improvements over conventional bus and crossbar interconnections. Research studies have shown remarkable achievement on chip design through NoC architecture regarding scalability (Ahmed, Abdallah & Kuroda, 2010; Maekawa, 2010; Naeem, Chen, Lu & Jantsch, 2010; Zimmer & Mueller, 2012). A high level of parallelism is achieved because it is noted that every link in the NoC architecture can operate simultaneously on different data packets.

In the design of an NoC-enabled system, efficient routing algorithms are of high priority, this is because a good routing algorithm aims at minimizing the communication latency and power consumption while enhancing the system throughput (Ahmed & Abdallah, 2012).

Network on Chip provides a way of realizing interconnections on silicon and primarily overcome the limitations of bus-based system solutions that include lack of scalability, clock skew, lack of support for simultaneous communication and power consumption (Mori, Esch, Abdallah & Kuroda, 2010). This is achieved by having processing elements connected through a packet switched communication network that supports efficient routing algorithm techniques aiming at optimum performance.

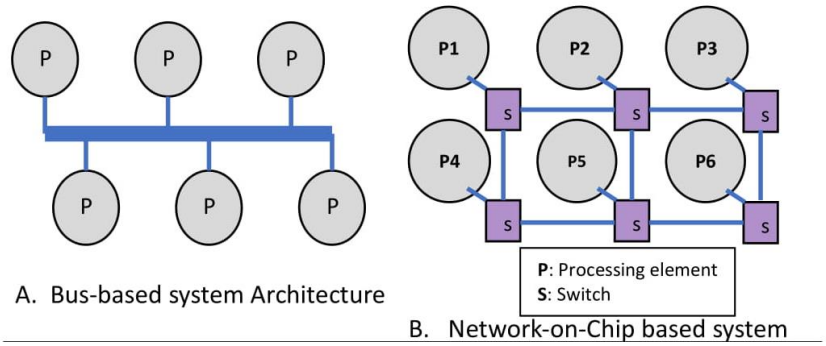

Figure 4.1: Block Diagrams of Bus-based and NoC based systems Explanation: In the above figures, (a). Shows a bus-based system with Processing Elements connected on a shared single bus while (b) shows a Network on Chip system with Processing Elements connects via switches

4.3 OASIS Network on Chip (OASIS-NoC)

The OASIS-based NoC framework was initially designed and named OASIS-1 by Mori, Esch, Abdallah and Kuroda (2010). The system is an n x m mesh-topology that uses wormhole switching. Wormhole switching is described as a First-Come-First-Served (FCFS) scheduler, and a novel re-transmission flow control (RFC). The FCFS scheduling is a simple algorithm and was carefully designed to be suitable for implementation in hardware. The RFC scheme is implemented in the processing elements (PEs) themselves.

It allows the source nodes to send the checksum and the PEs of destination nodes confirm the correctness. Hence, when flits got corrupted or dropped during the process of transmission, they must be retransmitted, thereby allowing new flits to be resent from source PEs. Therefore, with the help of this method, OASIS can correctly transmit data even in noisy environments. The major drawback of OASIS-1 is in its FCFS scheduler technology (algorithm). The FCFS has unbalanced network utilization when looking at overall performance, since some transactions are served by the scheduler and others are being delayed. This method causes unbalanced network utilization and increases the latency. Latency is increased because retransmission technique must resend flits when errors occur during transactions.

OASIS-1 was optimized as OASIS-2. The system optimization technique employs stall-go for avoiding buffer overflow and matrix arbiter for the scheduler. The OASIS-2 system framework also supports wormhole-like switching and virtual-cut-through forwarding method. The switching method which is chosen in a given instance depends on the level of packet fragmentation. Each router has input buffers which can store up to four flits. When a packet is divided into more than four flits, OASIS-2 chooses the wormhole switching technique. When packets are divided into less than four flits, the system chooses the virtual cut-through technique. In other words, when the buffer size is greater than or equal to the number of flits, virtual cut-through is used, but when the buffer size is less than or equal to the number of flits, wormhole switching is deployed**.** In summary, OASIS NoC is a complexity effective on-chip interconnection network. It uses a 4 × 4 mesh network, as shown in Figure 4.2 below.

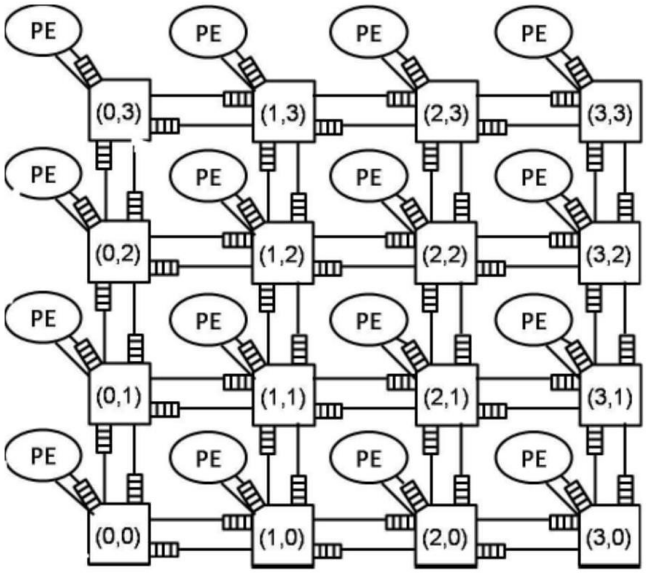

Figure 4.2: A 4 × 4 OASIS Mesh Topology

4.3.1 Switching and Packet Format

The on-chip cores of OASIS NoC architecture are arranged as a mesh of switches. Each switch has a maximum of five inputs, and five outputs ports that use single input/output pairs for communication with the resource while the remaining four pairs are connected neighbouring switches. The switch has 76 bits flit size, as shown in Figure 4.3 below. Flit structure has information of payload, destination address, and next port information. Flits are transmitted one by one using wormhole routing (one flit size is 76 bit which contains information payload, destination address, and next port information) and are buffered in input port FIFO. Each input port has FIFO which is 76 bits and depth is 4 with an increase in performance from buffer depth. Switches use XY coordinate routing technique. The architecture uses wormhole packet switching, sending messages via packets (many flits). Therefore, the switching has low latency, saves memory buffers and with proper routing algorithm deadlock can be avoided.

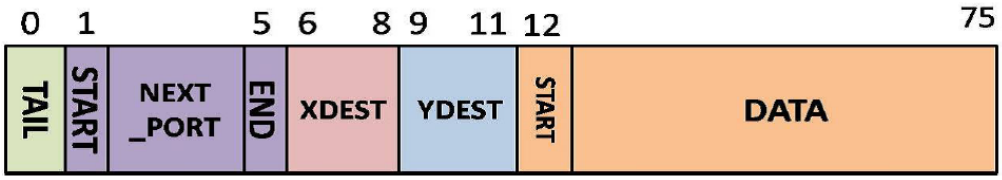

Figure 4.3: Packet of flit structure

Each flit contains enough routing information as: DATA (64 bit) called payload, **TAIL** (1 bit) is the last flits of a packet **NEXTPORT** (5 bit) means direction of output, **XDEST** (3 bit) is x-address destination, **YDEST** (3 bit) is y-address destination The NEXT PORT direction in next address is decided in input port just by comparing the destination address with the next address**.**

4.3.2 Switch Architecture and Buffer

In design, the OASIS switch hardware consists of **crossbar circuit**, 5 buffered **input ports** (where all the control flow and routing tasks are handled). The crossbar allows 5 different data from the 5 input ports to be routed to the next output port according to its destination and Next port. It has an **arbiter** or **Switch allocator** which uses a simple Round-Robin scheduling scheme, with high priority. The arbiter is used to grant data paths by evaluating how many available positions the FIFO has and the data size of all requesting FIFOs. It means using arbiter, the Next port information transmitted from input port to switch allocator is decoded to decide packet of the output direction. Hence, Switch buffer module is used for storing several flits. A buffer overflow occurs when the buffer cannot store all flits. Buffer (FIFO) is found in each of the five input ports, and each can hold a maximum of four (4) flits. The possibility of dropped flits increases since the buffer size is small when there is high traffic though large buffers consume extra power and area.

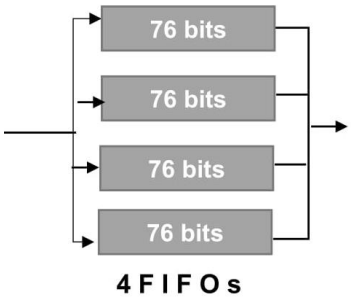

Figure 4.4: Buffer Design

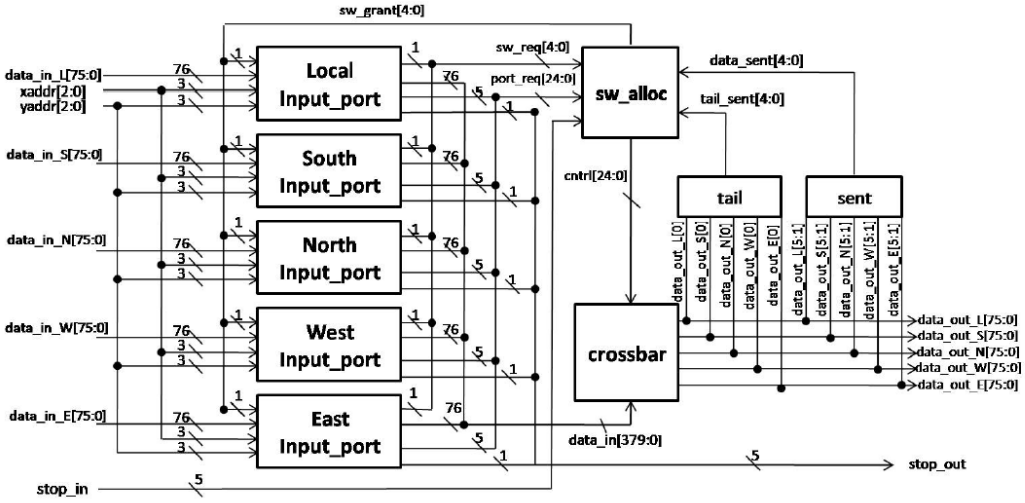

Figure 4.5: Shows One Switch Data Path

The L, S, N, W and E indicate the direction of port, Local, South, North, West and East respectively.

4.3.3 Flow Control and Routing Mechanism

Flow control shows how packets traverse the network path as specified by the routing algorithm. It allocates network resources to ensure correct packet transmission and reception. Thus, in OASIS NoC this technique is optimized using a Stall-Go flow control scheme which allows a limited number of flits to be stored into FIFO. The input flits could overflow if FIFO is full; hence the need of queuing technique. The Stall-Go mechanism is implemented to avoid FIFO being full. Therefore, the output signal is controlled by the arbiter that determine the output flits, if next switch is nearly full or not nearly full (see Figure 4.6).

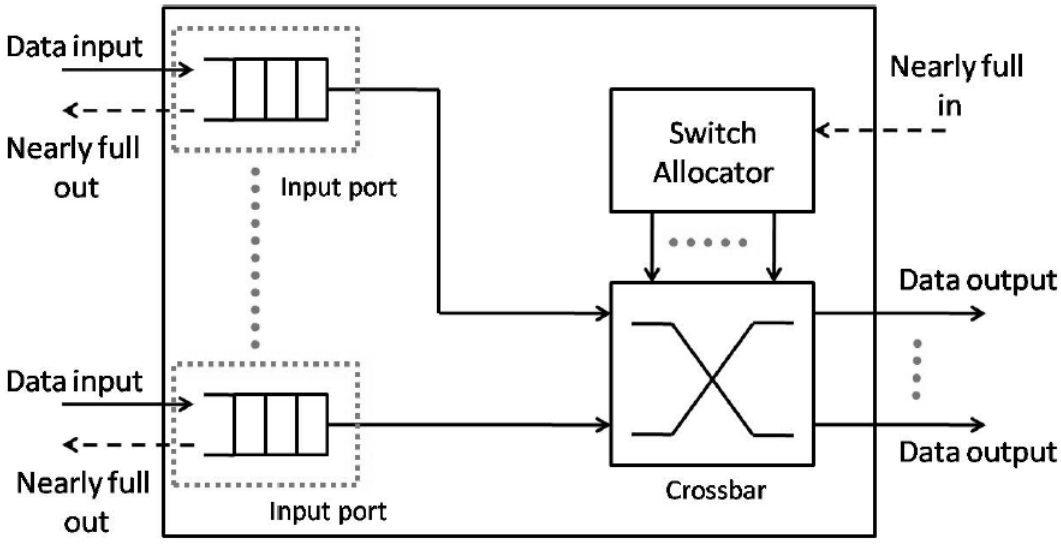

Figure 4.6: Stall-Go Flow Control Mechanism

Then, OASIS NoC makes use of deterministic routing approach. Every packet path is completely determined by the source and the destination addresses of the packet. The next port direction in next address is decided in input port. To compare next address and destination address, next port is decided for either NORTH, SOUTH, WEST, EAST OR LOCAL using the algorithm below.

ALGORITHM: Given **Y_des, X_des**: Y coordinates destination and X coordinate addresses respectively, **Y_Next_Add**, **X_Next_Add**: next nodes y and x – addresses respectively,

next_Add, des_Add: next address pair (X, Y) and destination address pair (X, Y) respectively.

IF (Y_des > Y_Next_Add) **THEN**
Next_Port = NORTH

ELSE IF (Y_des < Y_Next_Add) **THEN**
Next_Port = SOUTH

ELSE IF (X_des > X_Next_Add) **THEN**
Next_Port = EAST

ELSE IF (X_des < X_Next_Add) **THEN**
Next_Port = WEST

ELSE IF (next_Add = des_Add) **THEN**

Next_Port = LOCAL

END

Algorithm 2: X-Y Deterministic Routing Approach

4.4 High-Level Neuro-Inspired Architectures in Hardware

This section is the major contribution to our research. We implemented SNN using STDP learning algorithm in Chapter 3 by software simulation. We also studied the architecture of OASIS NoC in section 4.3. Now we propose High-Level Neuro-Inspired Architectures in Hardware. This contribution is meant to leverage the spiking neuro model, particularly the Leaky Integrate-and-Fire model reviewed in section 2.4 equally used to model our SNN in section 3.3.1, and NoC Architecture discussed in section 4.2, to improve the existing OASIS-NoC for future hardware chip implementation called High-Level Neuro-Inspired Architectures in Hardware (NASH).

Our primary contribution here is to propose the high-level NASH and describe its major component design for future on-chip hardware systems. The high-level NASH is shown in Figure 4.7 below.

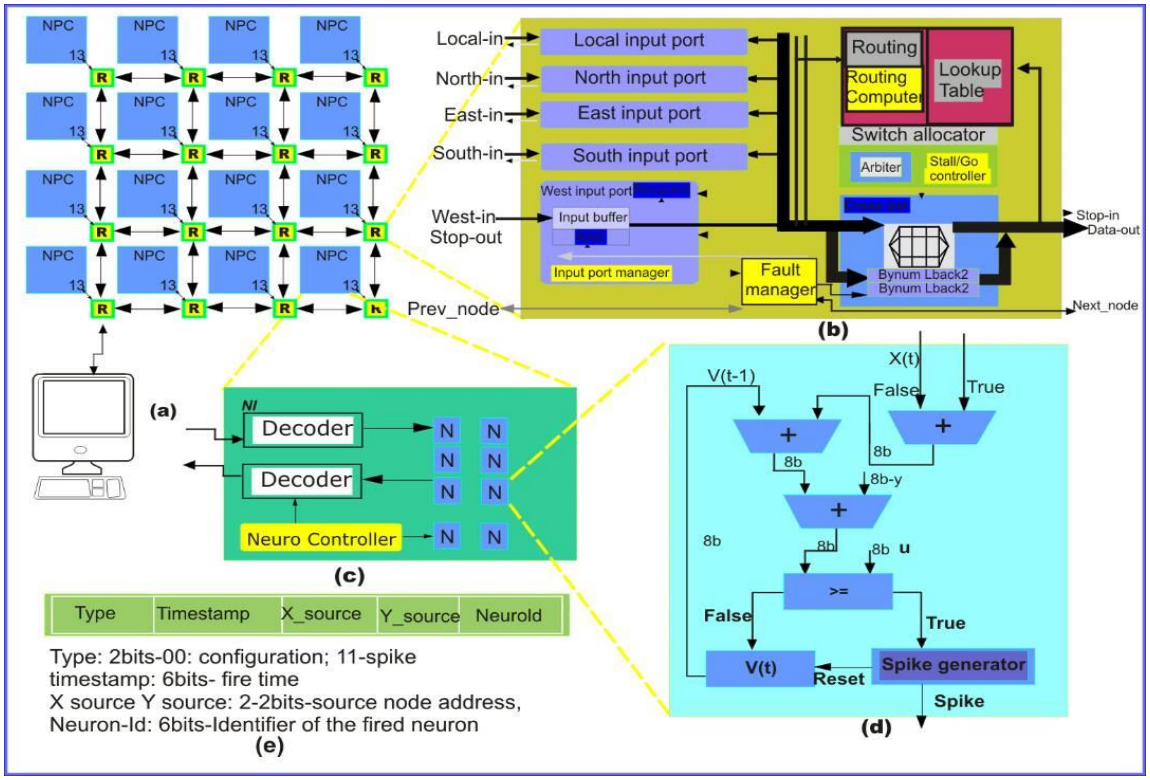

Figure 4.7: Block Diagram of High-Level Neuro-Inspired Architectures in Hardware

4.4.1 NASH Components Description

As we know, the biological brain implements massively parallel computations using a complex architecture that is different from the current Von Neumann machine. Our brain is a low-power, fault-tolerant, and high-performance machine. Hence, to implement neuro-inspired architectures in hardware (NASH), the component design is vital to ensure good performance. The NoC interconnection model is characterized by its topology, routing and, switching techniques, flow control, and arbiter. There are various trade-offs between hardware cost and performance, and design time and performance.

So, designers need a careful and deep understanding of all design choices of the following components and processes. The new high-level NASH depicted in Figure 4.7 has the following major components, namely:

4.4.1.1 High level of NASH architecture

This component describes the chip interconnection of our network which leverages the OASIS-NoC architecture discussed in section 4.3 (Ahmed & Abdallah, 2012). In Figure 4.8, we see a connection between a PC and scalable high-level 4 × 4 NoC (which could be FPGA or any other neuro hardware simulation device). This link is used for designing, downloading the RTL netlist, reconfiguration, and updating the weights and connections during network training.

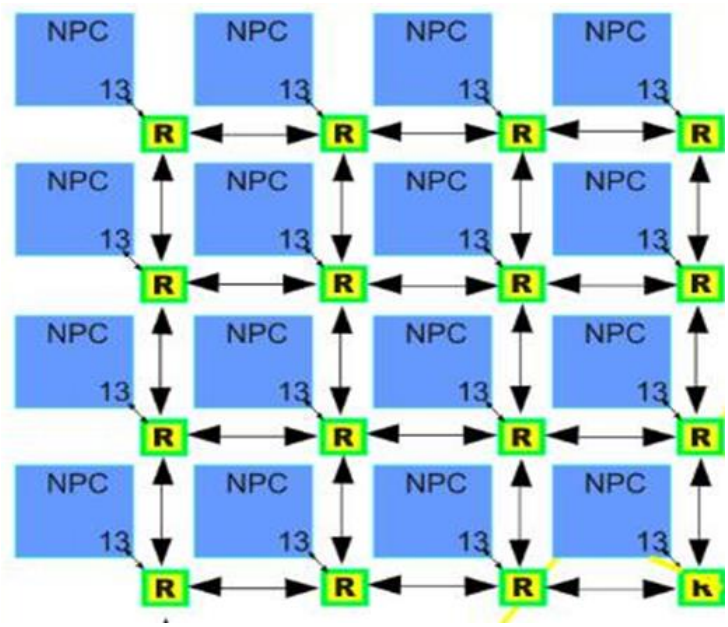

Figure 4.8: NASH High Level Chip Design

4.4.1.2 Multicast fault-tolerant router architecture

The flow control and routing mechanism of our high-level NASH adapted the approach of Stall-Go flow control and routing algorithm discussed in section 4.3.3 above. The OASIS router provides high-speed pipeline architecture (Mori, 2012). In Mori's research work, it provides extremely high bandwidth by distributing the propagation delay across multiple switches, hence, pipelining the packet transmission. Our three-stage pipelined router architecture uses a speculative strategy based on a simple look-ahead routing, where each flit additionally carries one hot encoded next port identifier used by the downstream routers,

in providing routing adaptation. Each router compares the current address and destination address to select the output port direction.

The OASIS router's main functions are classified into routing calculation, arbitration and flow control, and data transmission. In the first stage, input flits are stored in the input buffer. When the buffers are almost full it sends a signal to the upstream neighbour router. In the second stage, routing calculation is done by using the stored flits' information. The flits' stored module sends a Request signal to the arbitration module, and then the arbiter selects the winner to access the output and sends a Grant signal to the stored input module, this system based on least recently served scheme. The flow control is employed to avoid dropping flits, and it uses a state machine to manage the signals coming from downstream neighbour router about its input module status and also the state router's output data. Finally, the third stage which transmits flits that have included updated routing information and fixed payload to the adequate output direction. The router supports pipelined routing, so all calculations and comparisons can be executed in parallel. The multicast approach allows one-to-many communication (Furber, 2016; Galluppi, Rast, Davies & Furber, 2010). This is necessary because our chip is composed of many neuro-processing cores.

4.4.1.3 Neuro Processing Core (NPC)

This is also called a neuro-processing unit. It comprises neuron box, decoder, encoder and neuron controller. Then each neuron-box has eight different neurons, and each neuron is modelled with Leaking Integrate-and-Fire spiking neuron model discussed in section 3.3.1 of this work. Our neuro-processing core is an inspiration of TrueNorth, which implemented a new era of cognitive computing that brings forth the grand challenge of developing systems capable of processing massive amounts of noisy multisensory data (Akopyan et al., 2015).

4.4.1.4 Spiking Neuron Model

The spiking neuron model adapted in our proposed high-level NASH is the LIF (Leaky Integrate-and-Fire) neuron model which we reviewed in section 2.4, also used to model our SNN in section 3.3.1. Regarding design, complexity is considered better than other models we reviewed in section 2.4. The LIF core is used to achieve addition and multiplication operations. In our architecture, several inputs are connected to a single neuron, so, the inputs-weights multiplication and addition of every single neuron is done by LIF core which replaces the usual traditional sigmoid or tanh activation functions used in conventional neural networks. See Figure 4.9 below.

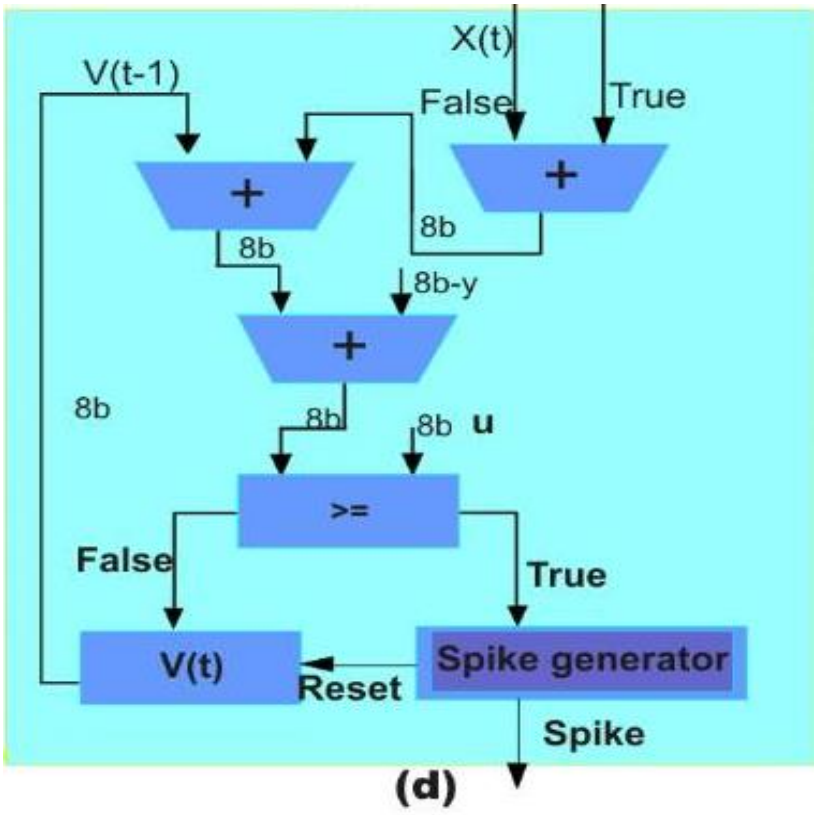

Figure 4.9: LIF Neuron Model Architecture

The LIF neuron core has a threshold value, μ which determines if there is spike generation or not. With reset, the model is reset to zero immediately when the positive spike is generated. It also has a specified refractory period value to ensure that the neuron does not fire again immediately after reset.

4.4.1.5 Flit Format

A packet is the unit of data that is routed between a source and destination of a network. The packet format of our high-level NASH comprises 18 bits. These include Type (2 bits), Timestamp (6 bits), X_{source} (2 bits), Y_{source} (2 bits) and Neuro_Id (6 bits). Hence, timestamp is modelled with the LIF spiking neuron model to ensure and keep updates of the time of spikes of every neuron input spike generated and transmitted.

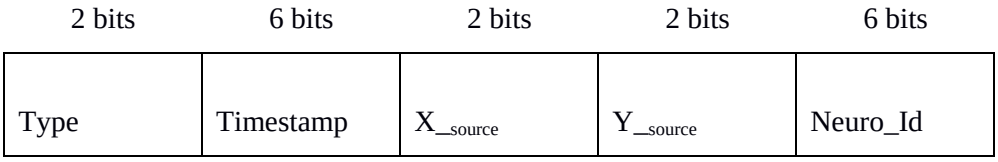

Figure 4.10: Packet Format

This packet format has smaller size, 18 bits length compared to the 76 bits length flit format found in section 4.3.1. Therefore, there is low power consumption here noticed due to this advance in flit format length.

4.5 Advantages of High-Level NASH Design over Traditional Bus-Systems

In future we plan to implement above high-level NASH in hardware which has some advantages and capability to solving the following problem, unlike the conventional bus-system-based architecture. First, NASH is designed to solve the large-scale implementation issues that arise from reconfigurability and programmability by adopting new efficient architecture based on a packet-switched network.

Second, the system will be able to adapt to different applications. To deal with this challenge, the system applies an on-chip learning mechanism, in which filters/weights are calculated directly on the system during training. This concept is described as online learning mechanism.

Third, the system is adapted to solve the problem of reliability issues; this is achieved by integrating a multicast routing algorithm during the router design. This technology ensures fault-tolerance of our high-level NASH architecture.

Fourth, the system consumes low power energy by implementing a Spiking Neural Network. This is because a Spiking Neural Network, being the third-generation artificial neural network, mimics closely the natural brain function that uses approximately 20 watts having over 100 billion neurons communicating at the same time. Also, it overcomes the problem of backpropagation of learning all the weights of every input to a neuron, high power consumption.

Finally, massive parallelism: our proposed high-level NASH will achieve a high level of parallelism since it is assumed it would be able to closely mimic the natural brain function using the spiking neuro model.

4.6 Applications of NASH on-chip system

This proposed NASH research is suitable for various applications, including:

- Image classification/recognition;
- Handwritten digit recognition as we have applied in MNIST dataset;
- Speech recognition;
- Sound processing.

4.7 Chapter Summary

This chapter has discussed the needs of NoC over Bus architecture. It looked at the OASIS-NoC architecture. Finally, we introduced our main contribution by proposing the design of a novel high-level NASH with its various component designs and discussion.

CHAPTER FIVE
RESULT, ANALYSIS, AND EVALUATION

5.1 Introduction

This chapter is dedicated to discuss and evaluate the result of the software-based SNN implementation in Chapter 3 for digit recognition and the LIF neuron core power verification for our proposed NASH architecture in Chapter 4, using Quartus II. The chapter also highlights some challenges we encountered during the simulation and recommends solutions. Finally, it concludes our research work and suggests other future works expected to enhance our work.

5.2 Result Analysis on Accuracy

Our SNN was trained and tested with MNIST datasets comprising 60,000 and 10,000 training and testing examples respectively. However, due to the limited capacity of our conventional personal computer (PC) used for the simulation, we encountered high space complexity issues on the system memory, we did not train many of the datasets. We trained two different sets of datasets of 800 and 1200 neurons and multiplied each by 2 (800 × 2 and 1200 × 2) during the training phase to increase their intensity. Each training set was tested with five different sets of testing examples ranging from 2000 to 500, 200, 100 and 50. Hence, for each test carried out we performed evaluations and recorded their result based on classification or recognition accuracy and error values.

Figures 5.1 and 5.2 show the virtualization of our results. Figure 5.1 is a graphical representation of accuracies against a number of test datasets while Figure 5.2 is the representation of error against a number of test datasets for both two different training sets we carried out. The result shows that accuracy is inversely proportional to error for both training datasets on the same ranges of testing datasets, see Figure 5.3. On that note, we noticed very high classification accuracies (86% for 1200 × 2 and 94% for 800 × 2) and very

high reduction in error value (7 with 0.59% for 1200 × 2 and 3 with 0.34% for 800 × 2) for both training datasets when tested with smaller datasets of 50. These deductions were visualized better in Table 5.1 and Figure 5.3. In summary, since the accuracy increased to 94% with minimal error of 0.34% by testing with smaller test datasets of 50 on smaller training datasets of 800 x 2; it implies that high accuracy will still be obtained by increasing our testing datasets if (and only if) our training datasets are increased and that justified the expectation of our benchmark.

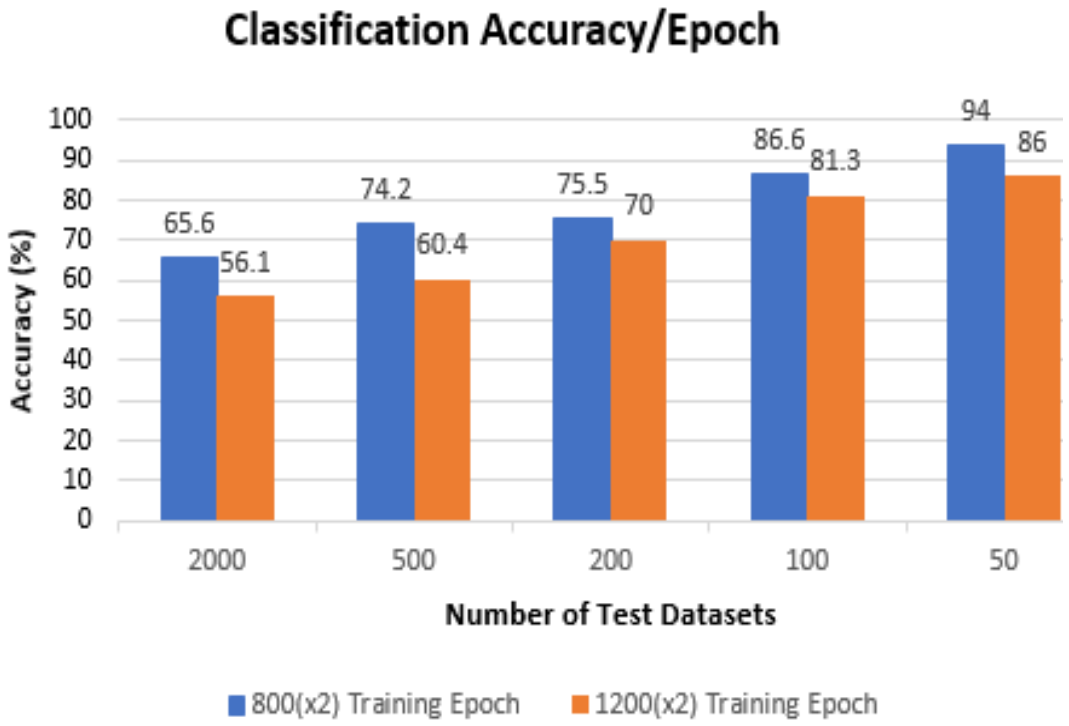

Figure 5.1: Classification accuracy

Both the training datasets increased in accuracy as the testing datasets are reduced. Hence, 800 × 2 and 1200 × 2 have better accuracy of 94% and 86% respectively at the testing datasets of 50 as shown in Figure 5.1 above.

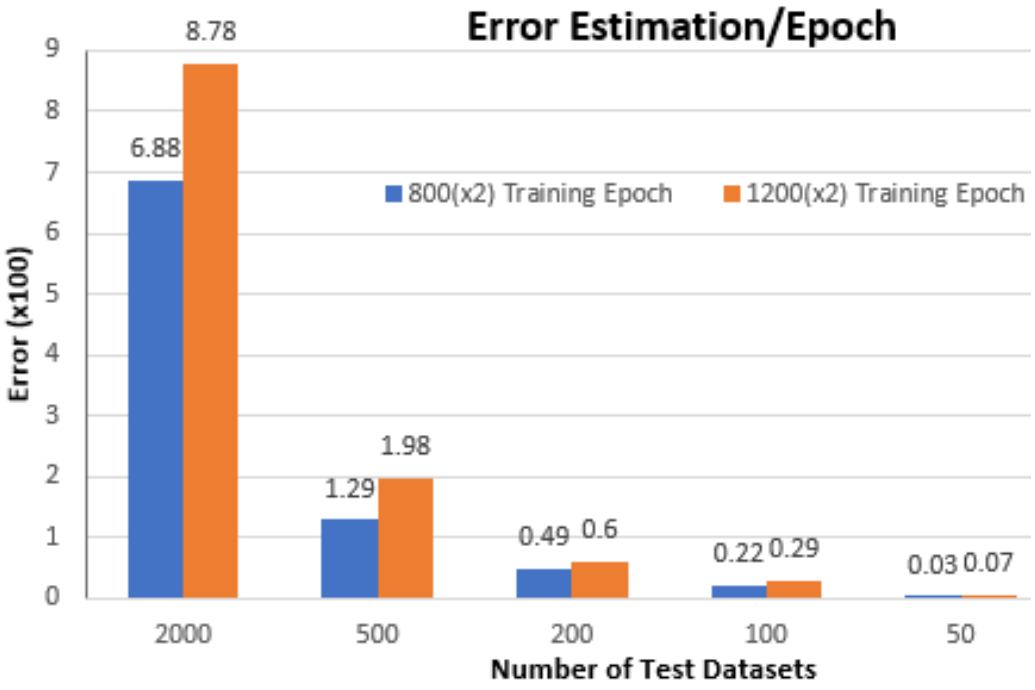

Figure 5.2: Error Estimation

Both the training datasets have their error decreased as the testing datasets reduce. Hence, 800 × 2 and 1200 × 2 have minimal error of 3 (0.34%) and 7(0.59%) respectively at the testing datasets of 50 as show in Figure 5.2 above.

Table 5.1: Accuracy vs Error evaluation

Training Datasets / Testing Datasets	A = 1200 x 2			B = 800 x 2		
	Accuracy	Error		Accuracy	Error	
	(%)	Value	(%)	(%)	Value	(%)
2000	56.10	878.00	74.91	65.60	688.00	77.22
500	60.40	198.00	16.89	74.20	129.00	14.48
200	70.00	60.00	5.12	75.50	49.00	5.50
100	81.30	29.00	2.47	86.60	22.00	2.47
50	86.00	7.00	0.59	94.00	3.00	0.34

Both the training datasets have their accuracy increased as the testing datasets reduce. Hence, 800 × 2 and 1200 × 2 have better accuracy of 94% and 86% respectively at the testing datasets of 50 as show in Table 5.1 above

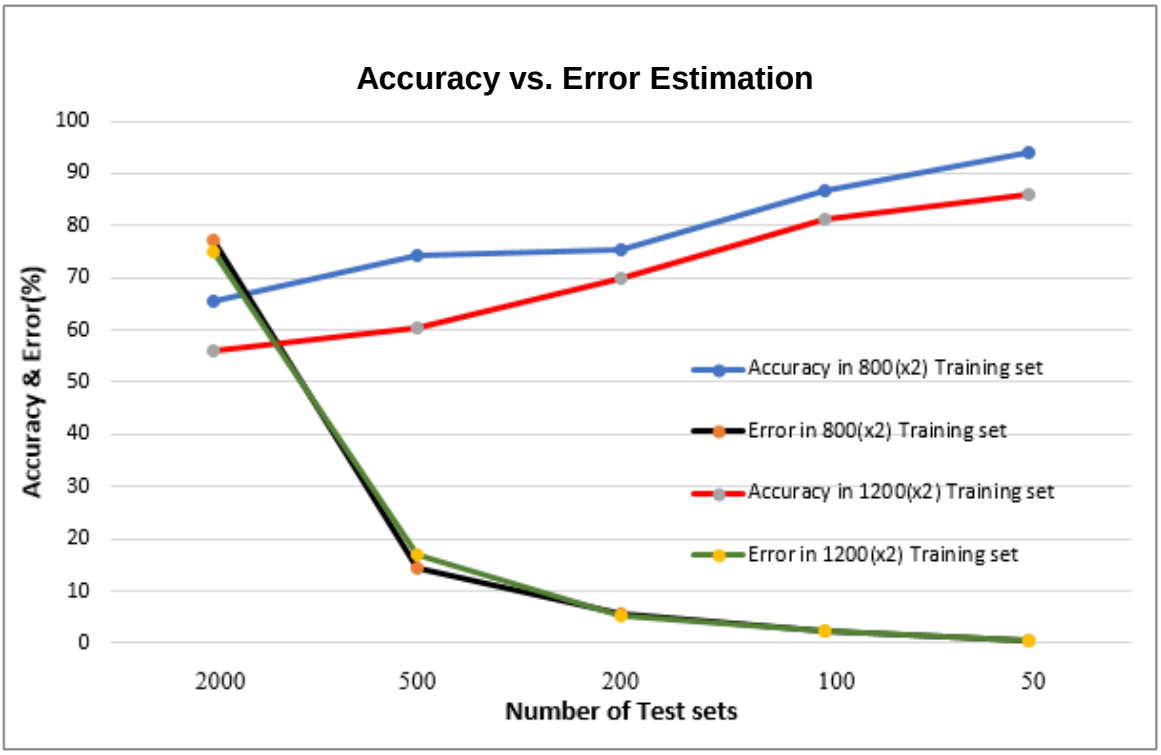

Figure 5.3: Accuracy vs Error evaluation

Both the training datasets have their accuracy increased as the testing datasets reduce and also have their error reduced exponentially as the testing datasets reduce. Hence, 800 × 2 and 1200 × 2 have better accuracy or minimal error of 94% or 3 (0.34%) and 86% 7 (0.59%) respectively at the testing datasets of 50 as shown in Figure 5.3 above. Therefore, 94% is our best accuracy.

5.3 Discussion

The result presented is based on the system specification of the conventional personal computer used for the implementation. The desktop computer used has memory (16 GiB), CPU (Intel core i.5, 3.20 GHz). It does not support training larger data sets due to memory space complexity. The system memory gets exhausted every time we try to increase our training dataset. The MNIST database of handwritten digits is a public, standard and real-world dataset that has a training set of 60,000 examples and a test set of 10,000 examples.

It is a subset of a larger set available from NIST. The digits have already been size-normalized and centred in a fixed-size 28 × 28 image. It is also described as a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending less or no efforts on pre-processing and formatting of the dataset. The MNIST dataset has been used by various researchers to test their Spiking Neural Network performance accuracy. Table 5.2 below shows various performances recorded by different researchers.

Table 5.2: Performance evaluation

Architecture	Preprocessing	Training-type	(Un-)supervised	Learning-rule	Performance
Dendritic neuron (Hussain et al., 2014)	Thresholding	Rate-based	Supervised	Morphology learning	90.30%
Spiking RBM (Merolla et a., 2011)	None	Rate-based	Supervised	Contrastive divergence, linear classifier	89.00%
Spiking RBM (O'Connor et a., 2013)	Enhanced training set to 120,000 examples	Rate-based	Supervised	Contrastive divergence	94.10%
Spiking convolutional neural network (Diehl et al., 2015)	None	Rate-based	Supervised	Backpropagation	99.10%
Spiking RBM (Neftci et al., 2013)	Thresholding	Rate-based	Supervised	Contrastive divergence	92.60%
Spiking RBM (Neftci et al., 2013)	Thresholding	Spike-based	Supervised	Contrastive divergence	91.90%
Spiking convolutional neural network (Zhao et al., 2014)	orientation detection, scaling, thresholding	spike-based	Supervised	Tempotron rule	91.30%
Two-layer network (Brader et al., 2007	Edge detection	Spike-based	Supervised	STDP with calcium variable	96.50%
Multi-layer hierarchical network (Brader et al., 2013	Orientation detection	Spike-based	Supervised	STDP with calcium variable	91.60%
Two-layer network (Querlioz et al., 2013)	None	spike-based	Unsupervised	Regular STDP	93.50%
Two-layer network (Diehl et al., 2015)	None	Spike-based	Unsupervised	Exponential STDP	95.00%
Two-layers Ne (this thesis)	**None**	**Spike-based**	**Unsupervised**	**Exp STDP**	**94%**

Based on our results we conclude that our implementation performance is within the benchmark accuracy. Hence this justifies our test of accuracy and performance for our proposed novel High-Level Neuro-Inspired Architectures in Hardware (NASH) discussed in Chapter 4.

5.4 Result Analysis on Power Consumption

In evaluating the power consumption for the proposed NASH, Quartus II was used to implement neurocomputing unit (NCU) LIF cores. One NCU LIF core and four NCU LIF cores were implemented and compiled on Quartus II; the power rating was recorded and compared the results as presented below.

Table 5.3: Comparison of Logic Element and Power

	LIF_Neuron_core (single)	NCU_4 (4 LIF_neuron_core)
Logic Elements(LE)	76	304
Core Dynamic Thermal Power	0.00 mW	0.01 mW
Core Static Thermal Power	98.52 mW	98.70 mW
I/O Power Thermal Power	52.26 mW	108.93 mW
Total Power Thermal Power	**150.78 mW**	**207.64 mW**

Table 5.4: Evaluation of Area report

	LIF_Neuron_core (single)	NCU_4 (4 LIF_neuron_core)
Combinational Area	151.354001 µm	562.856001 µm
Non-combinational Area	40.697999 µm	213.864000 µm
Total Cell Area	**192.051999 µm**	**776.720001 µm**

Table 5.5: Evaluation of Power report

	LIF_Neuron_core (single)	NCU_4 (4 LIF_neuron_core)
Cell Internal Power	5.2613 µW	20.5040 µW
Net Switching Power	3.6885 µW	14.8272 µW
Total Dynamic Power	**8.9498 µW**	**35.3312 µW**
Cell Leakage Power	3.8433 µW	14.3147 µW

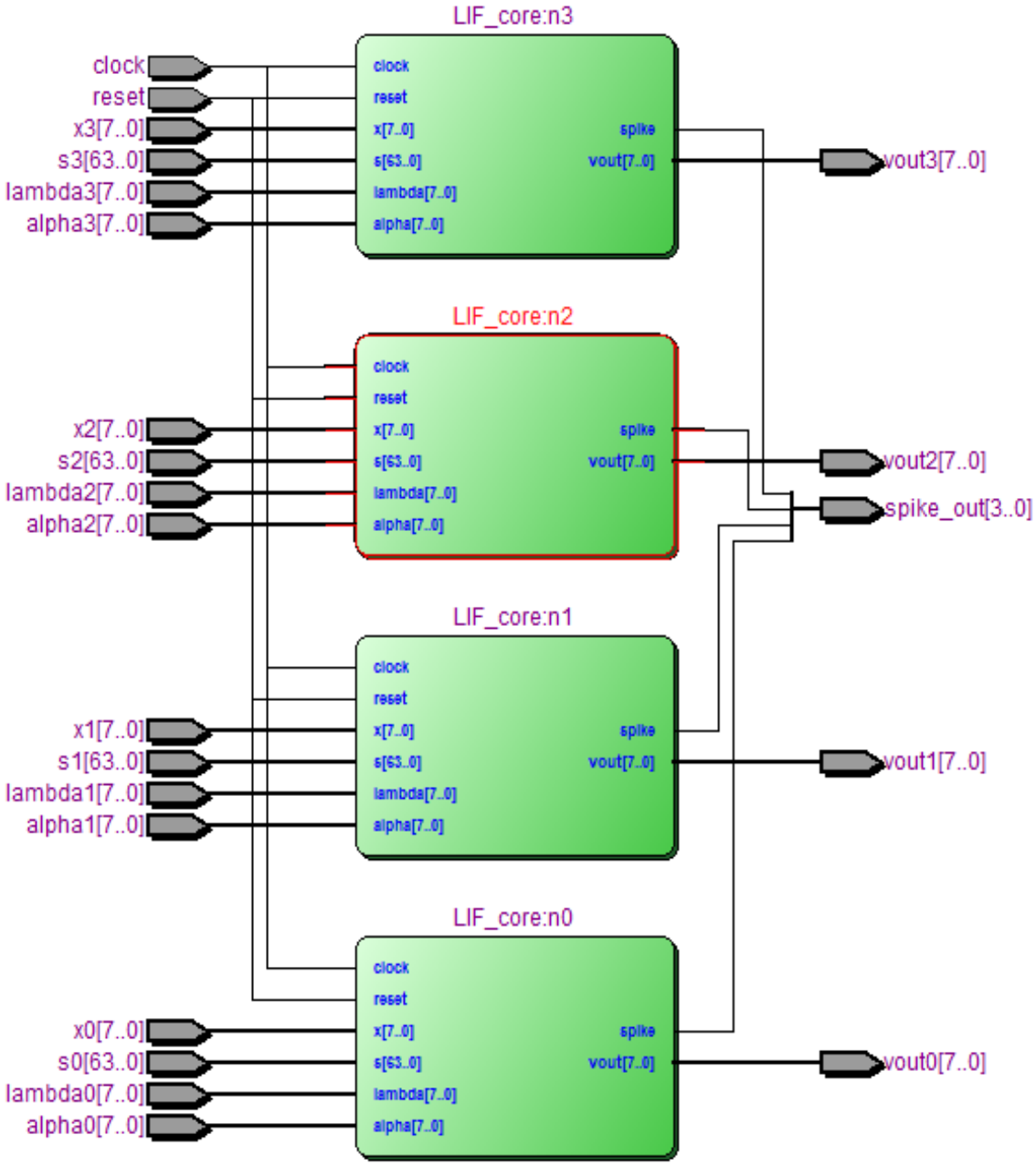

Figure 5.4: Schematic - NCU4 (4 LIF_neuron_core)

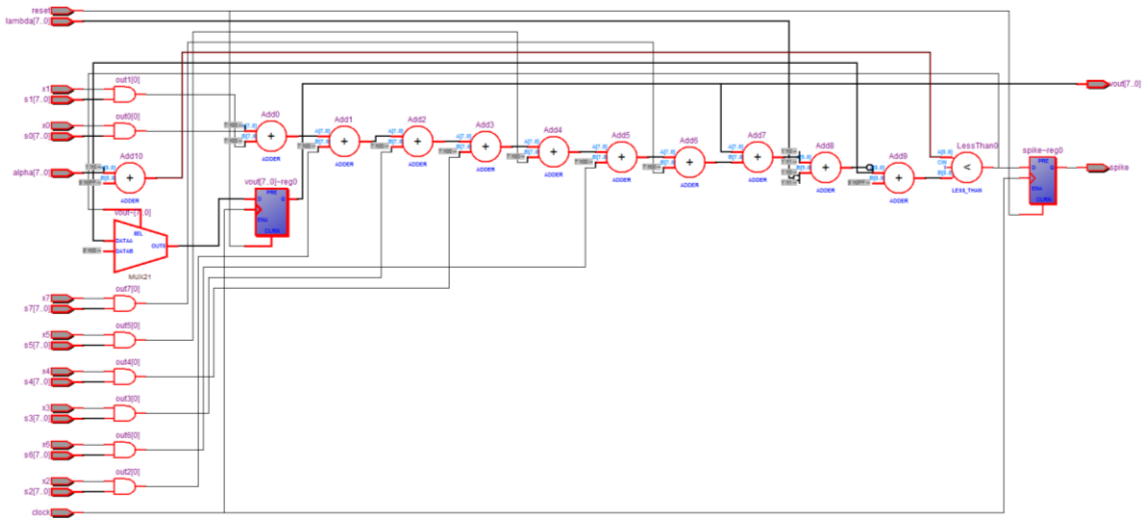

Figure 5.5: Schematic - NCU (LIF_neuron_core)

Both single and 4 NUC LIF cores are implemented in Verilog HDL, and after synthesis and analysis in Quartus II, the schematic design netlists viewed above were generated.

Table 5.6: I/O LIF neuron core

LIF core	Input		Output	
	Clock	1 bit	Membrane potential	8 bit
	Reset	1 bit	Output spike	1 bit
	Input spike	8 bit		
	Synaptic weight	64 bit		
	Leak value	8 bit		
	Threshold value	8 bit		
4 LIF core	Input		Output	
	Clock	1 bit	Membrane potential	32 bit
	Reset	1 bit	Output spike	4 bit
	Input spike	32 bit		
	Synaptic weight	256 bit		
	Leak value	32 bit		
	Threshold value	32 bit		

CHAPTER SIX
CONCLUSION AND FUTURE WORK

6.1 Introduction

This chapter summarizes our research work. It describes our research and our contributions, highlights some technical challenges encountered during the research, recommends solutions for those challenges and finally suggests future work needed to advance our research.

6.2 Conclusion

This thesis presents a Spiking Neural Network architecture design and performance exploration towards the design of a scalable Neuro-inspired system for complex cognition applications. Hence, in our studies to achieve the aim of our research, we made the following sub-contributions which include: (1) We came up with an architecture and circuit development towards the design-scalable Neuro-inspired System called NASH. (2) We performed hardware design and evaluation of a LIF Core for Neuro-inspired Spiking NASH System. (3) We implemented a software-based Spiking Neural Network (SNN) using Leaky Integrate-and-fire (LIF) neuron model with spike timing dependent plasticity (STDP) learning rule. Our SNN was applied in a digit recognition tested with MNIST datasets of handwritten digits and recorded its classification accuracy. We also evaluated power consumption of the on the design NASH for future on-chip systems.

6.3 Technical Challenges

Technical issues experienced were mostly space and time complexities. During our simulations, we ran short of memory space and were unable to simulate much of our dataset. Also, we experienced very low-speed simulation time; the lack of a steady power supply caused simulations process shutdown only to start over the next time.

6.4 Future Work

Research is a continuous process, hence, implementing SNN on hardware will improve our work, it will provide a better platform to analyse and compare all the necessities like power consumption. On the same note, another vast and good research future work is to implement our high-level NASH architecture on hardware, FPGA.

REFERENCES

Ahmed, F., Yusob, B., & Hamed, H. N. A. (2014). Computing with spiking neuron networks: A review. *International Journal of Advances in Soft Computing and Its Applications*, *6*(1). https://doi.org/10.1007/978-3-540-92910-9_10

Ahmed, A. B., & Abdallah, A. B. (2012). LA-XYZ: Low latency, high throughput look-ahead routing algorithm for 3D network-on-chip (3D-NoC) architecture. In *Proceedings - IEEE 6th International Symposium on Embedded Multicore SoCs, MCSoC 2012* (pp. 167–174). https://doi.org/10.1109/MCSoC.2012.24

Ahmed, A. B., Abdallah, A. B., & Kuroda, K. (2010). Architecture and design of efficient 3D network-on-chip (3D NoC) for custom multicore SoC. In *Proceedings - 2010 International Conference on Broadband, Wireless Computing Communication and Applications, BWCCA 2010* (pp. 67–73). https://doi.org/10.1109/BWCCA.2010.50

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., … Modha, D. S. (2015). 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*(10), 1537–1557. https://doi.org/10.1109/TCAD.2015.2474396

Alnajjar, F., Bin Mohd Zin, I., & Murase, K. (2008). A Spiking Neural Network with dynamic memory for a real autonomous mobile robot in dynamic environment. *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2207–2213.

Basegmez, E. (2014). *The next generation Neural Networks : Deep learning and Spiking Neural Networks*. Technische Universitat Munchen.

Ben-David, S., & Shalev-Shwartz, S. (2014). *Understanding machine learning: From theory to algorithms*. https://doi.org/10.1017/CBO9781107298019

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J. M., … Boahen, K. (2014). Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, *102*(5), 699–716. https://doi.org/10.1109/JPROC.2014.2313565

Bi, G., & Poo, M. (2001). Synaptic modification by correlated activity: Hebb's Postulate revisited. *Annual Review of Neuroscience*, *24*(1), 139–166. https://doi.org/10.1146/annurev.neuro.24.1.139

Bohte, S. M., Kok, J. N., & La Poutré, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*. https://doi.org/10.1016/S0925-

2312(01)00658-0

Booij, O., & Tat Nguyen, H. (2005). A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters*, *95*(6 SPEC. ISS.), 552–558. https://doi.org/10.1016/j.ipl.2005.05.023

Brette, R., & Goodman, D. F. M. (2011). Vectorized algorithms for Spiking Neural Network simulation. *Neural Computation*, *23*(6), 1503–1535. https://doi.org/10.1162/NECO_a_00123

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., … Destexhe, A. (2007). Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*. https://doi.org/10.1007/s10827-007-0038-6

Cassidy, A. S., Merolla, P., Arthur, J. V., Esser, S. K., Jackson, B., Alvarez-Icaza, R., … Modha, D. S. (2013). Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *Proceedings of the International Joint Conference on Neural Networks*. https://doi.org/10.1109/IJCNN.2013.6707077

Christophe, F., Mikkonen, T., Andalibi, V., Koskimies, K., & Laukkarinen, T. (2015). Pattern recognition with Spiking Neural Networks: A simple training method.

Davies, S. (2012). *Learning in Spiking Neural Networks*. The University of Manchester. Retrieved from http://www.manchester.ac.uk/escholar/uk-ac-man-scw:186210

Diehl, P. U., & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity, *9*(August), 1–9. https://doi.org/10.3389/fncom.2015.00099

Eugene, M. (2007). *Dynamical systems in neuroscience*. London: The MIT Press.

Furber, S. (2016). Large-scale neuromorphic computing systems. *Journal of Neural Engineering*, *13*(5), 51001. https://doi.org/10.1088/1741-2560/13/5/051001

Galluppi, F., Rast, A., Davies, S., & Furber, S. (2010). Neural information processing. Theory and algorithms. *Lecture notes in Computer Science (including subseries lecture notes in Artificial Intelligence and lecture notes in Bioinformatics)*, *6443*(PART 1), 58–65. https://doi.org/10.1007/978-3-642-17537-4

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models.* Cambridge University Press, Cambridge. https://doi.org/10.2277/0511075065

Ghosh-Dastidar, S., & Adeli, H. (2009). Spiking neural networks. *International Journal of Neural Systems*, *19*(4), 295–308. https://doi.org/10.1017/CBO9781107415324.004

Goodman, D., & Brette, R. (2009). The Brian simulator. *Frontiers in Neuroscience*. https://doi.org/10.3389/neuro.01.026.2009

Goodman, D., Stimberg, M., Yger, P., & Brette, R. (2014). Brian 2: Neural simulations on a variety of computational hardware. *BMC Neuroscience*, *15*(Suppl 1), P199. https://doi.org/10.1186/1471-2202-15-S1-P199

Hecht-Nielsen, R. (1990). Neurocomputing. Boston, Mass.: Addison-Wesley.

Heskes, T. M., & Kappen, B. (1993). On-line learning processes in artificial neural networks. *North-Holland Mathematical Library*, *51*(C), 199–233. https://doi.org/10.1016/S0924-6509(08)70038-2

Izhikevich, E., & FitzHugh, R. (2006). FitzHugh-Nagumo model. *Scholarpedia*, *1*(9), 1349. https://doi.org/10.4249/scholarpedia.1349

Jin, X. (2010). *Parallel simulation of neural networks on SpiNNaker Universal Neuromorphic Hardware*. University of Manchester.

Krunglevicius, D. (2016). Modified STDP triplet rule significantly increases neuron training stability in the learning of spatial patterns. *Advances in Artificial Neural Systems*, *2016*. Retrieved from http://dx.doi.org/10.1155/2016/1746514

LeCun, Y., Cortes, C., & Burges, C. (1998). THE MNIST DATABASE of handwritten digits. *The Courant Institute of Mathematical Sciences*, 1–10. Retrieved from http://yann.lecun.com/exdb/mnist/%5Cnhttp://yann.lecun.com/exdb/publis/index.html#lecun-98

Long, L. N. (2008). Scalable biologically inspired neural networks with Spike Time Based Learning. *IEE Symposium on Learning and Adaptive Behavior in Robotics Systems*.

Maass, W. (1997). Networks of spiking neurons : The third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671.

Maekawa, T. (2010). Survey of NoC Topologies NoC architecture. *Webextuaizuacjp*, 4–7. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Survey+of+NoC+Topologies#0

Markram, H., Gerstner, W., & Sjöström, P. J. (2012). Spike-timing-dependent plasticity: A comprehensive overview. *Frontiers in Synaptic Neuroscience*, 2–5. https://doi.org/10.3389/fnsyn.2012.00002

McKennoch, S., Liu, D. L. D., & Bushnell, L. G. (2006). Fast modifications of the SpikeProp algorithm. *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 3970–3977. https://doi.org/10.1109/IJCNN.2006.246918

Mori, K. (2012). *OASIS Network-on-Chip Prototyping on FPGA*.

Mori, K., Esch, A., Abdallah, A. B., & Kuroda, K. (2010). Advanced design issues for OASIS network-on-chip architecture. In *Proceedings - 2010 International Conference on Broadband, Wireless Computing Communication and Applications, BWCCA 2010* (pp. 74–79). https://doi.org/10.1109/BWCCA.2010.51

Morrison, A., Aertsen, A. D., Diesmann, M., & Morrison, A. (2007). Spike-Timing-Dependent plasticity in balanced random networks. *Neural Computation Massachusetts Institute of Technology*, *19*, 1437–1467. https://doi.org/10.1162/neco.2007.19.6.1437

Naeem, A., Chen, X., Lu, Z., & Jantsch, A. (2010). Scalability of weak consistency in NoC based multicore architectures. In *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems* (pp. 3497–3500). https://doi.org/10.1109/ISCAS.2010.5537833

Nessler, B., Maass, W., & Pfeiffer, M. (2009). STDP enables spiking neurons to detect hidden causes of their inputs. *Advances in Neural Information Processing Systems*, *22*, 1357–1365.

Orhan, E. (2012). The Leaky Integrate-and-Fire Neuron Model, (3), 1–6.

Painkras, E., Plana, L. A., Garside, J., Temple, S., Galluppi, F., Patterson, C., … Furber, S. B. (2013). SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, *48*(8), 1943–1953. https://doi.org/10.1109/JSSC.2013.2259038

Ponulak, F., & Kasinski, A. (2011). Introduction to Spiking Neural Networks: Information processing, learning and applications. *Acta Neurobiologiae Experimentalis*, *71*(4), 409–33. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/22237491

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., & Indiveri, G. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Frontiers in Neuroscience*, *9*(APR). https://doi.org/10.3389/fnins.2015.00141

Sarpeshkar, R. (2012). Ultra low power biomedical and bio-inspired systems. *Frontiers of Engineering, 2011 National Academy of Engineering Symposium*, 137–143.

Schrauwen, B., & Campenhout, J. Van. (2004). Improving spikeprop: Enhancements to an error-backpropagation rule for Spiking Neural Networks. *Proceedings of the 15th ProRISC Workshop*, *11*, 301–305. Retrieved from http://pdf.aminer.org/000/366/795/customizing_parallel_formulations_of_backpropagation_learning_algorithm_to_neural_network.pdf

Shrestha, A., Ahmed, K., Wang, Y., & Qiu, Q. (2017). Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning, 1999–2006.

Silva, S. M., Ruano, A. E., & Ieee. (2006). Application of the Levenberg-Marquardt method to the training of Spiking Neural Networks. In *2006 Ieee International Joint Conference on Neural Network Proceedings, Vols 1-10* (pp. 3978–3982). https://doi.org/10.1109/ICNNB.2005.1614882

Sjostrom, P. J., & Gerstner, W. (2010). Spike-timing-dependent plasticity. *Scholarpedia*, *2*(2), 1362. https://doi.org/10.1162/089976604773135041

Soltic, S., Wysoski, S. G., & Kasabov, N. K. (2008). Evolving Spiking Neural Networks for taste recognition. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 2091–2097). https://doi.org/10.1109/IJCNN.2008.4634085

Song, S., & Abbott, L. F. (2001). Cortical development and remapping through spike timing-dependent plasticity. *Neuron*, *32*(2), 339–350. https://doi.org/10.1016/S0896-6273(01)00451-2

von Bartheld, C. S., Bahney, J., & Herculano-Houzel, S. (2016). The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting. *Journal of Comparative Neurology*, *524*(18), 3865–3895. https://doi.org/10.1002/cne.24040

Wang, J., Belatreche, A., Maguire, L., & McGinnity, M. (2010). Online versus offline learning for Spiking Neural Networks: A review and new strategies. *IEEE 9th International Conference on Cyberntic Intelligent Systems*, 1–6. https://doi.org/10.1109/UKRICIS.2010.5898113

Wilson, M. A., Bhalla, U. S., Uhley, J. D., & Bower, J. M. (1989). GENESIS: A system for simulating neural networks. *Advances in Neural Information Processing Systems*, 485–492.

Wysoski, S. G., Benuskova, L., & Kasabov, N. (2008). Fast and adaptive network of spiking neurons for multi-view visual pattern recognition. In *Neurocomputing* (Vol. 71, pp. 2563–2575). https://doi.org/10.1016/j.neucom.2007.12.038

Zillmer, R. (2007). Simple neuron models : FitzHugh-Nagumo and Hindmarsh-Rose. *Networks*. Retrieved from http://www.fi.isc.cnr.it/users/alessandro.torcini/ARTICOLI/lezione4-zillmer.pdf

Zimmer, C., & Mueller, F. (2012). Fault resilient real-time design for NoC architectures. In *Proceedings - 2012 IEEE/ACM 3rd International Conference on Cyber-Physical Systems, ICCPS 2012* (pp. 75–84). https://doi.org/10.1109/ICCPS.2012.16